

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIAS E TECNOLOGIA
DO TOCANTINS
CAMPUS GURUPI
CURSO DE BACHARELHADO EM ENGENHARIA CIVIL**

GUSTAVO HENRIQUE DE ARAÚJO BORGES

**DESENVOLVIMENTO DE UM SIMULADOR, EM DOMÍNIOS VOLUMÉTRICOS,
PARA REPRESENTAR O FLUXO DE TENSÕES EM VIGAS**

**GURUPI
2021**

GUSTAVO HENRIQUE DE ARAÚJO BORGES

**DESENVOLVIMENTO DE UM SIMULADOR, EM DOMÍNIOS VOLUMÉTRICOS,
PARA REPRESENTAR O FLUXO DE TENSÕES EM VIGAS**

Trabalho de conclusão de curso apresentado como requisito parcial para obtenção do título de Bacharel no Curso Superior de Engenharia Civil do Instituto Federal do Tocantins, *Campus* Gurupi.

Orientador: Prof. Paulo Roberto de Oliveira Junior.

Dados Internacionais de Catalogação na Publicação (CIP)
Bibliotecas do Instituto Federal do Tocantins

B732d Borges, Gustavo Henrique de Araújo
 Desenvolvimento de um Simulador, em Domínios Volumétricos, para
 Representar o Fluxo de Tensões em Vigas. / Gustavo Henrique de Araújo
 Borges. – Gurupi, TO, 2021.
 230 f. : il. color.

 Trabalho de Conclusão de Curso (Bacharelado em Engenharia Civil) –
 Instituto Federal de Educação, Ciência e Tecnologia do Tocantins, Campus
 Gurupi, Gurupi, TO, 2021.

 Orientador: Paulo Roberto de Oliveira Junior

 1. Desenvolvimento de um simulador didático, para representar
 graficamente o fluxo de tensões de vigas a partir de elementos discretos.. I.
 Junior, Paulo Roberto de Oliveira. II. Título.

CDD 624

A reprodução total ou parcial, de qualquer forma ou por qualquer meio, deste documento é autorizada para fins de estudo e pesquisa, desde que citada a fonte.

Elaborado pelo sistema de geração automática de ficha catalográfica do IFTO com os dados fornecidos pelo(a) autor(a).

GUSTAVO HENRIQUE DE ARAÚJO BORGES

**DESENVOLVIMENTO DE UM SIMULADOR, EM DOMÍNIOS VOLUMÉTRICOS,
PARA REPRESENTAR O FLUXO DE TENSÕES EM VIGAS**

Trabalho de conclusão de curso apresentado como
requisito parcial para obtenção do título de
Bacharel no Curso Superior de Engenharia Civil
do Instituto Federal do Tocantins, *Campus Gurupi*.

Aprovado em: 09 / 12 / 2021

BANCA AVALIADORA

Prof. Paulo Roberto de Oliveira Junior
IFTO – *Campus Gurupi*

Prof. Me. Rodrigo Araújo Fortes
IFTO – *Campus Gurupi*

Prof. Me. Walisson Pereira de Sousa
IFTO – *Campus Araguaina*.



Documento assinado eletronicamente por **Paulo Roberto de Oliveira Junior, Servidor**, em 15/12/2021, às 15:53, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Rodrigo Araujo Fortes, Servidor**, em 15/12/2021, às 16:24, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Walisson Pereira de Sousa, Servidor**, em 15/12/2021, às 17:37, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade deste documento pode ser conferida no site http://sei.iftto.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1477582** e o código CRC **FBF321B6**.

Alameda Madrid, 545, — CEP 77410-470, Gurupi/TO — (63) 3311-5400
portal.iftto.edu.br — reitoria@iftto.edu.br

AGRADECIMENTOS

.Quero agradecer aos professores Walisson Sousa e Bruno Coelho, por incentivarem minha curiosidade (vocês impactaram muito na construção dos meus valores), ao meu professor e orientador Paulo Roberto, pela paciência e por nunca ter me desencorajado (mesmo diante desse tema), e por fim, agradeço aos meus amigos e companheiros da graduação(Caio Vinicius, Sarah Cortez, Moni Kelly, Mateus Pereira, Fernanda Martins, Tassila Silva) por me incentivarem e me darem apoio, sem vocês eu não teria a coragem de encarar uma pesquisa tão desafiadora.

*Na medida em que, a tecnologia
alcança a imaginação, da imaginação
é exigida responsabilidade.*

Neri Oxman

RESUMO

A adoção de simuladores, no estudo de sistemas complexos, atribui um grau de simplificação no estudo do fenômeno (tanto simuladores físicos quanto virtuais). Há uma variedade de simuladores estruturais físicos no mercado, porém a alta valorização no aspecto da deformação os torna semelhantes, enquanto isso, os simuladores virtuais de mercado tem o *postprocessing* majoritariamente orientado em gradiente de cores (ou seja, para compreender a tensão interna, é estudada a disposição do gradiente de cores, perdendo informações sobre o sentido dessa deformação no processo). A presente pesquisa teve por finalidade mostrar uma perspectiva alternativa da dinâmica interna de deformação expressando tanto a intensidade quando a direção da deformação. Para isso, foi desenvolvido um programa de simulação virtual, com um painel simples que permite a manipulação da geometria de ensaio. A discretização e dimensionamento da malha foi feita usando método de elementos finitos FEA, ou seja, uma abordagem numérica (e nesse caso, com elementos lineares). Foi imaginado como um simulador de vigas, porém se mostrou adaptativo, podendo ser expandido para outros conjuntos de estrutura (pilares, lajes, muros). O simulador teve sucesso em expressar o fluxo de deformações, e foi aprimorado com outras fases de análise (complementando, cada uma da sua forma, na visualização do comportamento do sistema). Essa pesquisa está focada no desenvolvimento do simulador, não tendo muito a contribuir na análise desses dados. Contudo, a implementação, experimentação e análise desses dados podem ser realizadas em pesquisas futuras.

Palavras-chave: simulador, FEA, fluxo, deformações .

ABSTRACT

The adoption of simulators, in the study of comprehensive systems, attributes a degree of simplification in the study of the phenomenon (both physical and virtual simulators). There are a variety of indispensable simulators on the market, but the high value in the aspect of deformation makes them similar, meanwhile, the market's virtual simulators have post-processing mostly oriented on a gradient of cores (that is, to understand the stress internally, the disposition of the nuclei gradient is studied, losing information about the direction of this deformation in the process). The present research had as important to show an alternative perspective of the internal deformation dynamics expressing both the intensity and the direction of the deformation. For that, a virtual simulation program was developed, with a simple panel that allows the manipulation of the test geometry. The mesh discretization and dimensioning was done using the FEA finite element method, that is, a numerical approach (and in this case, with linear elements). It was conceived as a beam simulator, however adaptable, and could be expanded to other sets of structures (pillars, slabs, walls). The simulator was successful in expressing the deformation flow, and was improved with other phases of analysis (complementing, each one of its forms, in the visualization of the system's behavior). This research is focused on simulator development, not having much to contribute in the analysis of these data. However, implementation, experimentation and data analysis can be carried out in future research.

Keywords: simulator, FEA, flow, deformations .

LISTA DE ILUSTRAÇÕES

Figura 01	Sistema Massa mola (a), Modelo Borges(2020) (b), Modelo Lobosco(2017) (c).	15
Figura 02	Simulação virtual da deformação de uma estrutura.	16
Figura 03	Curva Catenária.	19
Figura 04	Maquete estrutural / Modelo estrutural de Gaudi.	20
Figura 05	Objeto, volume de controle, ou sistema em análise Ω .	22
Figura 06	Geometria em diferentes fases de distorção.	23
Figura 07	Geometria contendo uma nuvem de pontos.	24
Figura 08	Geometria discretizada em: Barras.	24
Figura 09	Isolamento de um elemento de barra genérico.	26
Figura 10	Conversão do elemento de Barra para um elemento de Massa Mola.	27
Figura 11	Influencia da direção da deformação com o sinal da força da mola	28
Figura 12	Vetores normalizados.	28
Figura 13	Orbitas particulares de uma partícula com relação à outra.	30
Figura 14	Deformação genérica do sistema.	30
Figura 15	Transformação de coordenadas.	33
Figura 16	Parâmetro ‘dimensões’.	43
Figura 17	Parâmetro ‘discretizador’.	44
Figura 18	Conexões por Octante.	45
Figura 19	Exemplo de criação de pontos Manualmente.	46
Figura 20	Unidade ‘célula’ com os vértices.	48
Figura 21	Referência de Mapeamento.	48
Figura 22	Referência de Faces.	49
Figura 23	Refinamento isovalores.	53
Figura 24	Transformação de Coordenadas.	54
Figura 25	Slider em duas configurações distintas.	57
Figura 26	Índice em suas configurações distintas.	58
Figura 27	Controle de Isovalores em configurações distintas.	58
Figura 28	Controle da direita “resetar”, controle da esquerda “avançar”.	59
Figura 29	Interface Dispositivo Móvel.	59
Figura 30	Interface Monitor.	60

Figura 31	Estrutura Elementar.	61
Figura 32	Estrutura Elementar - Programa.	67
Figura 33	Estrutura Elementar Dimensionada.	68
Figura 34	Estrutura Elementar Resultado da Deformação.	68
Figura 35	Estrutura Elementar - Fases de Deformação	69
Figura 36	Domo 24 Elementos.	73
Figura 37	Domo 24 Elementos - Dimensionado.	74
Figura 38	Domo 162 Elementos.	75
Figura 39	Domo 162 Elementos - Dimensionado.	76
Figura 40	Pilar, Viga, Laje, Parede - Discretização.	77
Figura 41	Pilar, Viga - Dimensionamento.	78
Figura 42	Pilar, Viga - Isovalores.	78
Figura 43	Pilar, Viga - Fluxo.	79
Figura 44	Viga - Deformação.	79

LISTA DE QUADROS

Quadro 01	Métodos analíticos e numéricos (adaptado de Marchi, 2001)	13
Quadro 02	Função initPontos.	84
Quadro 03	Função addConexoesPontos.	85
Quadro 04	Função adicionarVetoresNosPontos.	86
Quadro 05	Função adicionarCoresNosPontos.	87
Quadro 06	Função deformarPontos.	87
Quadro 07	Função desenharPontos.	88
Quadro 08	Função desenharVetores.	90
Quadro 09	Função initConexoes.	93
Quadro 10	Função desenharConexoes.	94
Quadro 11	Função initCelulas.	97
Quadro 12	Função desenharCelulas.	98
Quadro 13	Função addConexoesCelulas.	99
Quadro 14	Função mapearVerticesCelula.	100
Quadro 15	Função refinamentoCelulas.	102
Quadro 16	Função desenharGradiente.	106
Quadro 17	Função initGeometria.	117
Quadro 18	Função posicionarMalha.	117
Quadro 19	Função parametrizarCores.	123
Quadro 20	Função initCanvas.	126
Quadro 21	Função initCamera.	126
Quadro 22	Função iniCena.	127
Quadro 23	Função initIlluminar.	127
Quadro 24	Função initRenderer.	129
Quadro 25	Função criarElemento.	130
Quadro 26	Função navegarPontos.	130
Quadro 27	Função initSliders.	130
Quadro 28	Função initGradienteInputs.	134
Quadro 29	Função initNav.	137
Quadro 30	Função initNext.	138
Quadro 31	Função initBefore.	139

Quadro 32	Função initMatrizes.	141
Quadro 33	Arquivo de Estilização.	152
Quadro 34	Arquivo de Html.	161
Quadro 35	Programa Principal.	162
Quadro 36	Pontos Domo 132 Elementos.	178
Quadro 37	initAgente.	219
Quadro 38	initAgentes.	219
Quadro 39	moverAgente.	220
Quadro 40	compararPosicionamento.	221
Quadro 41	verificarPosicionamentoAgente.	222
Quadro 42	desenharAgente.	223

SUMÁRIO

1 INTRODUÇÃO	11
2 DEFINIÇÃO DO PROBLEMA	13
3 JUSTIFICATIVA E RELEVÂNCIA DO TEMA	15
4 OBJETIVOS	17
4.1 GERAL	17
4.2 ESPECÍFICOS	17
5 REFERENCIAL TEÓRICO	18
5.1 MODELOS ESTRUTURAIS	18
5.1.1 Modelos Didáticos	20
5.2 MECÂNICA CONTÍNUA	21
5.2.1 Análise individual e generalizada de um elemento genérico	25
5.2.2 <i>Assembly</i> das formulações fragmentadas desenvolvidas individualmente em cada elemento	36
5.2.3 Adição de Condições de Contorno de Neumann e Dirichlet (respectivamente: deformações e esforços externos conhecidos)	36
5.2.4 Resolução Numérica do sistema matricial final para descobrir os graus de deformação.	37
5.3 LINGUAGEM JAVASCRIPT	38
6 METODOLOGIA DA PESQUISA	39
6.1 PROCEDIMENTOS METODOLÓGICOS E MÉTODOS DE COLETA	39
6.1.1 Pesquisa Bibliográfica	39
6.2 DESENVOLVIMENTO DO PROGRAMA	40
6.2.1 Criação	41
6.2.2 Cálculo	47
6.2.3 Isovalores	51
6.2.4 Fluxo	53
6.2.5 Deformação	54
6.2.6 Interface	54
7 RESULTADOS E DISCUSSÕES	59
7.1 PRECISÃO DE CÁLCULO	59
7.1.1 Manual	60

7.1.2 Dimensionamento pelo Programa	64
7.2 FUNCIONALIDADE DO PROGRAMA	68
7.2.1 Manual	68
7.2.2 Automático	75
8 CONCLUSÃO	78
9 REFERÊNCIAS	79
APÊNDICE A - VERTICES	82
APÊNDICE B - CONEXÕES	91
APÊNDICE C - CELULAS	95
APÊNDICE D - MALHA	115
APÊNDICE E - AMBIENTE 3D	124
APÊNDICE F - INTERFACE	128
APÊNDICE G - MATRIZES	137
APÊNDICE H - STYLE	148
APÊNDICE I - HTML	157
APÊNDICE J - PROGRAMA PRINCIPAL	158
APÊNDICE K - COORDENADAS GEOMETRIA DOMO	174
APÊNDICE L - FLUXO	217

1 INTRODUÇÃO

Eventos manifestados na natureza são fundamentalmente determinísticos e reversíveis, conhecendo o estado e contexto do sistema em um determinado instante do tempo, é possível prever seu comportamento em qualquer outro instante. Compreender a dinâmica entre um sistema e o contexto que ele está envolvido é essencial para muitas disciplinas científicas e de engenharia, contudo, pode ser complexo deduzir a descrição do fenômeno analiticamente (Lobosco 2017; Sanchez 2020).

A adoção de simuladores, para antecipação dessa dinâmica, atribui um grau de simplificação no estudo do fenômeno, pois convertem princípios analíticos da mecânica do sistema, em observações mais práticas e qualitativas. Nesse contexto, são inseridos modelos didáticos como ferramentas para aprendizagem, reduzindo a questão de “como descrever o fenômeno?”, para “o que é possível entender das observações?” (Sanchez 2020; Wolper 2019).

O comportamento do sistema depende do parâmetro associado (podem ser tensões externas, internas, interações termodinâmicas, entre outras). No caso de modelos didáticos estruturais, devido o interesse no aspecto da distorção, são avaliadas as relações entre a tensão e a cinemática de deformação do sistema.

Neste trabalho desenvolveu-se um simulador didático, para representar graficamente o fluxo de tensões de vigas a partir de elementos discretos - "Método dos Elementos Finitos" - (MEF). O estado do sistema será representado por uma malha de partículas, expressas como nós e interconectadas a sua vizinhança.

Enquanto modelos didáticos de estruturas, em sua maioria, foram especializados em exibir a deformação do sistema, esse se concentra em exibir os fluxos de tensões internas, para contribuir na interpretação de como esse fluxo influencia na deformação visualizada.

O Lagrangiano será utilizado para obter a função de mapeamento das deformações do sistema, a qual será implementada no Laplaciano da deformação, que expressa a dissipação das tensões através dos elementos. Os vetores gerados indicarão as linhas de fluxo do sistema (fluxo de deformação) e esta análise permitirá uma apreciação mais completa e didática para os estudantes e profissionais da área de engenharia.

O trabalho está organizado da seguinte forma: Seção I, Introdução histórica do uso de modelos estruturais - a influência da variação de escala na concepção estrutural é delineada. Para tanto, foram buscados autores como Oxman (2010); Oxman (2013); Torroja (2007);

Saramago (2011); Addis (2007); Hilson (1993); Moura (2016); Lobosco (2017); e Ricón (2018).

A homogeneização do domínio contínuo é discutida na Seção II. Nesse contexto, a generalização do volume de controle, e a descrição das deformações são introduzidas. Consequentemente, a função constitutiva é deduzida pelo gradiente e Laplaciano. Foram buscados autores como Mase (1970); Balankin (2004); Craizer (2002); Giacomini (2009); Torradi (2018); Anflor (2007); e Morton; Mayers (2005).

Seção III é dedicado a abordagem numérica. A discretização do domínio em unidades finitas é desenvolvida. São estabelecidas as propriedades para cada sub-elemento, e os requisitos para o espaço funcional. As funções para elementos individuais são implementadas. O mapeamento das tensões e deformações para elementos finitos, é estabelecido pela equação aproximada. Foram buscados autores como Jamba (2019); Rocho (2012); Bedin (2002); Torradi (2018); Giacomini (2009); Anfor (2007); Mesit (2010), Morton, Mayers (2005); Flanagan (2004); Dirksen (2015).

2 DEFINIÇÃO DO PROBLEMA

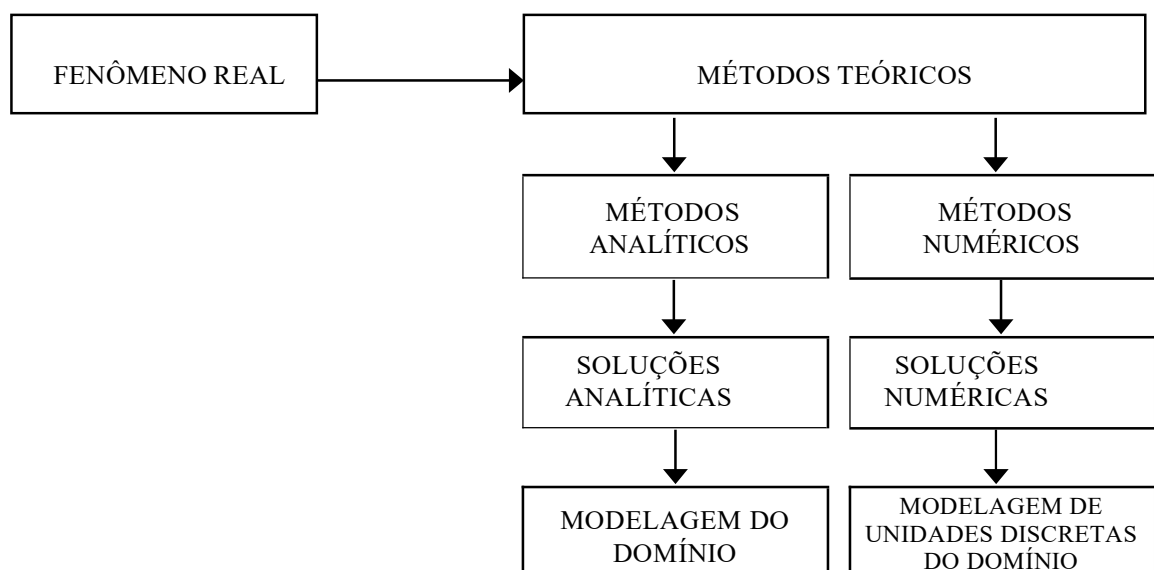
Fraturas e deformações se manifestam diariamente no mundo real, mas nem todos os materiais quebram da mesma maneira. Em computação gráfica, a simulação de fratura requer métodos para mudanças topológicas em grande escala (Wolper, 2019).

A trajetória da evolução científica paralela a invenção dos computadores dividiu o estudo desses fenômenos em duas classes: métodos analíticos e numéricos. Os métodos analíticos representam o fenômeno com equações matemáticas, restritas pela geometria, bem adaptados para elementos lineares, porém a complexidade da geometria pode resultar em erros de modelagem. (TANNEHILL et al., 1997). Enquanto isso, os métodos numéricos são mais adaptáveis (Wolper 2019, Giacomini, 2009),

Os métodos numéricos, além de representarem o fenômeno real por meio de modelos matemáticos, têm grandes vantagens em relação aos outros: a possibilidade para evolução temporal do processo de cálculo; emprego de problemas não-lineares; geometrias complicadas e condições de contornos variáveis (GIACOMINI, 2009).

Para incorporar os métodos numéricos na área da computação científica, foram desenvolvidas técnicas de discretização aplicadas nas equações diferenciais que representam o modelo matemático. Apesar da amplitude das abordagens de discretização, a maioria dos métodos de fratura e deformação se concentrou na mecânica da fratura (FM) (Wolper, 2019; Giacomini, 2009).

Quadro 01 - Métodos analíticos e numéricos (adaptado de Marchi, 2001)



Entre os métodos envolvidos com FM podem ser citados: síntese de elementos finitos (FES) (Oxman 2011), design de propriedade variável (VPD) (Oxman, 2010), método dos elementos finitos (MEF) (Wolper, 2019; Wolper, 2020), e método das diferenças finitas (MDF) (Torrati, 2018).

Embora precisos e populares, os métodos em malha (derivados do FM) como FES, VPD, MEF e MDF exigem rotinas de re-meshing computacionalmente intensas, e algoritmos complexos para gerenciamento de nós. (Wolper, 2020, Wolper, 2019).

A forte dependência de malha introduzida pelo FM resulta em renderização de cenário com solicitações de tempo e memória de grande escala. Contudo, é essa característica que permite a representação visual de como os esforços atuam em cada unidade do volume, e esse detalhe se comunica bem com as propostas de modelos didáticos, que visam demonstrar visualmente como um fenômeno se comporta (Sanchez, 2020; Wolper, 2020; Wolper, 2019).

Para expressar a intensidade de tensões e deformações é usual o emprego de isovalores, ou seja, em cada unidade do volume é descrita uma cor parametrizada pela deformação. Em regiões mais deformadas as cores serão quentes (tendendo ao vermelho), e em regiões pouco deformadas, as cores serão frias (tendendo ao azul). Porém nessa abordagem, é perdida a visualização do sentido dos vetores da deformação.

Seria possível fazer uma representação visual do fluxo interno das tensões e deformações que representasse tanto a intensidade quando a direção da deformação?

3 JUSTIFICATIVA E RELEVÂNCIA DO TEMA

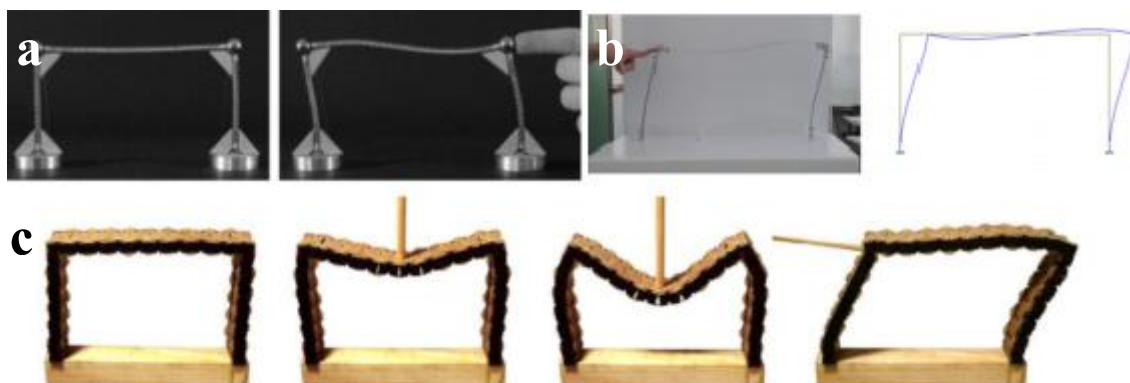
“Apreciar várias obras, e avalias suas características, pode refletir em suas próprias obras particulares” essa frase representa um Estado de Fluxo da arte, e pode ser adaptada para musica, inovações (alem de um amplo conjunto de atividades humanas).

Está envolvida com o aspecto fundamental da ‘inteligência’ (humana e artificial), o de que ela é resultado de estímulos externos (e das interações que esses promovem). Por isso, explorar perspectivas diferentes da mesma atividade tem o potencial de expandir e consolidar a compreensão de um indivíduo sobre o assunto.

Em especial na engenharia, são recorridas metodologias para fornecimento dessa ‘*perspectiva alternativa*’ (de conceitos teóricos estudados), sendo empregados modelos didáticos estruturais.

Há uma variedade de modelos didáticos estruturais no mercado, porém a alta valorização no aspecto da deformação os torna semelhantes. Apesar da aparência, e os materiais que os estruturam serem diferentes, o experimento é basicamente o mesmo (Figura 01).

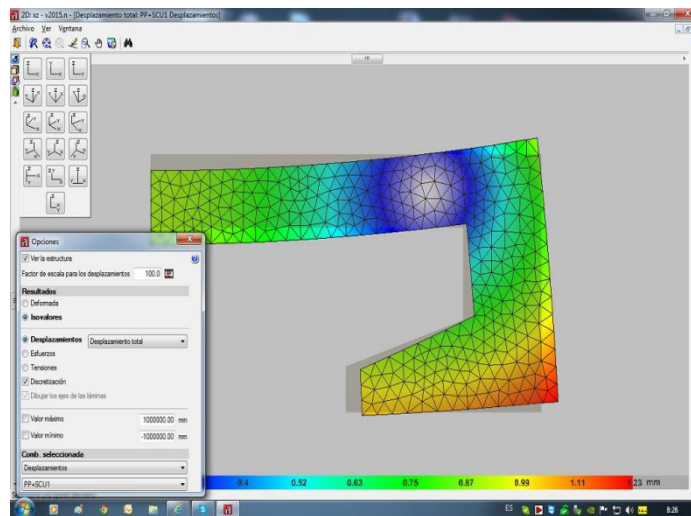
Figura 01 - Sistema Massa mola (a), Modelo Borges(2020) (b), Modelo Lobosco(2017) (c).



Fonte: Adaptado pelo autor.

Outra abordagem para expressar a intensidade das deformações é o uso de simuladores virtuais, onde o objeto é subdividido em pequenas unidades, e a intensidade da deformação é expressa por uma escala de cores (isovalores). Apesar dessa propriedade se comunicar bem com as propostas dos modelos didáticos, para compreender o fluxo da tensão interna, solicita ainda um estudo mais profundo do gradiente de cores (Figura 02).

Figura 02 - Simulação virtual da deformação de uma estrutura.



Fonte: Cypecad.

Uma possibilidade de visualização do sentido do fluxo interno dentro da estrutura é o uso de campos vetoriais para descrever os vetores de deformação em cada ponto, e traçando linhas de fluxo através desses vetores (e anexando os isovalores de cor representando a intensidade).

Assim, esse projeto se justifica por disponibilizar uma perspectiva alternativa para a observação da deformação interna de estruturas através do desenvolvimento de um simulador virtual que representa graficamente essas linhas de fluxo, convertendo a comparação de cores, ou visualização de deformações, nas simples questões de “Para onde o fluxo está fluindo?” e “Onde esse fluxo está sendo gerado?”.

4 OBJETIVOS

Esta pesquisa explorará o potencial de aplicar métodos de discretização na geração de componentes, incorporando a eles propriedades físicas e, estabelecendo uma interface gráfica que exiba o experimento. Esse trabalho sugere uma abordagem de análise envolvendo campos vetoriais. A abordagem busca fornecer um ensaio visualmente intuitivo.

4.1 GERAL

O objetivo geral será desenvolver um simulador gráfico de vigas, aplicando a lógica e computação de abordagens de elementos finitos, para apresentar o fluxo de deformações.

4.2 ESPECÍFICOS

- Calcular as equações aproximadas de deformação do sistema;
- Mapear o campo vetorial de tensão e deformação;
- Representar o mapeamento de deformações com um gradiente de cores;
- Representar graficamente linhas de fluxo nas direções apontadas pelos vetores do campo;
- Representar visualmente a deformação da peça estrutural.

5 REFERENCIAL TEÓRICO

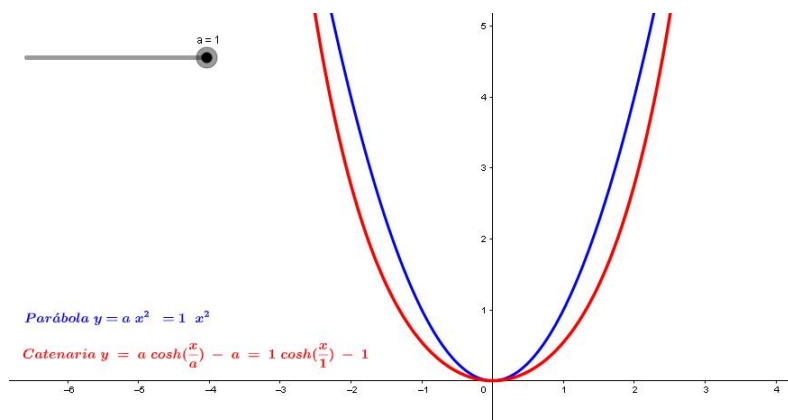
5.1 MODELOS ESTRUTURAIS

Na natureza a matéria é modelada, simulada, fabricada, e estruturada simultaneamente. Em comparação com essa produção integrada, até meados do início do século XX, as construções humanas não eram exatamente, otimizadas (Oxman, 2010; Oxman, 2013).

A história da concepção estrutural humana tem início na formação da natureza, com a primeira árvore, com o primeiro ser humano, é uma história sobre o que acontece quando a espécie humana é submetida à estímulos externos, perpetuando o desenvolvimento de ferramentas, abrigos, e comportamentos culturais. Contudo, mesmo que a humanidade esteja envolvida com a construção, a primeira descrição formal para essa classe de ferramenta foi manifestada apenas em meados de 1600, nos estudos de Robert Hooke sobre correntes, catenárias e arcos (Lobosco, 2017; Rincón, 2018).

Hooke descreveu com precisão a equação da curvatura representada pela corrente, conforme Figura 03, esclarecendo que sua forma é responsiva aos carregamentos externos, ou seja, a distorção da curvatura é influenciada pelas ações de externas, buscando o equilíbrio estático. Além disso, essa configuração de equilíbrio é mantida ao inverter o sistema de orientação, essa descrição invertida é denominada arco verdadeiro (as tensões de tração são convertidas em tensões de compressão), as implicações mais importantes desse estudo se manifestaram em aplicações práticas de otimização (Lobosco, 2017; Rincón, 2018).

Figura 03 - Curva Catenária.

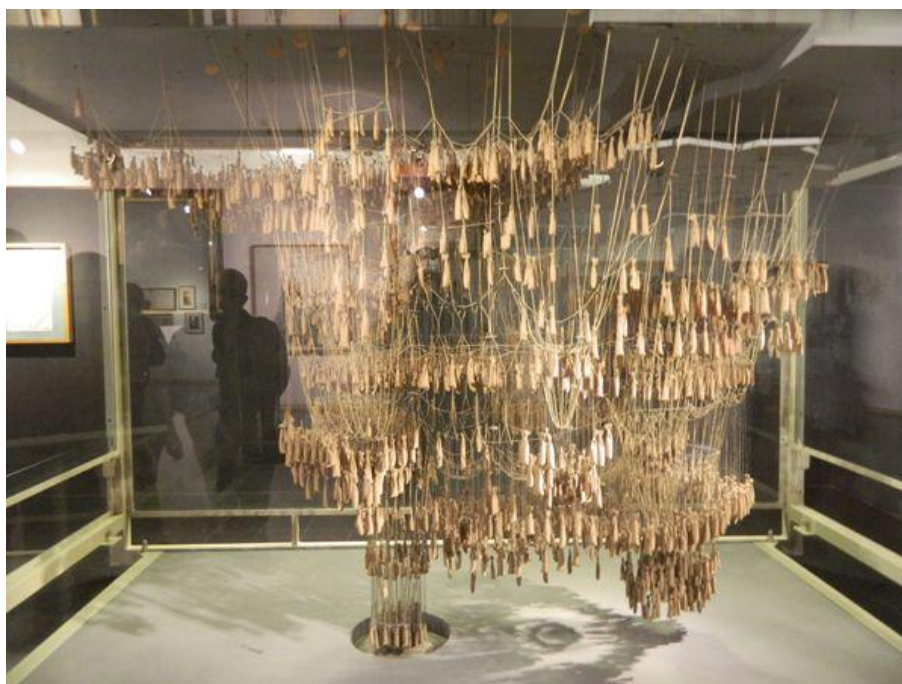


Fonte: Geogebra¹.

¹Disponível em : <<https://www.geogebra.org/m/cwub2bRZ>> . Acesso em : 23 mai. 2021.

Em 1800 a concepção estrutural estava orientada à construção de maquetes, devido o estado primitivo de estudos envolvendo a mecânica dos materiais. Na concepção da Catedral da Sagrada Família, Antoni Gaudi fez ampla aplicação de maquetes estruturais, desenvolveu as abobadas utilizando de sistemas complexos de correntes, conforme na Figura 04, permitindo uma modelagem estrutural complexa, porém estável, mas nesta metodologia, o design era controlado pelas propriedades estruturais (Lobosco, 2017).

Figura 04 - Maquete estrutural / Modelo estrutural de Gaudi.



Fonte: Acervo Estruturando¹.

No início do século XX, a era industrial trouxe um novo tipo de pensamento de design, ciências determinísticas já estavam estabelecidas, e com os novos conjuntos de materiais (aço, concreto, vidro, e polímeros) foi aberto um novo horizonte de possibilidades arquitetônicas. (Hilson, 1993; Saramago, 2011; Oxman, 2011).

Nesse ponto, o *design* e arquitetura conquistaram liberdade, pois a concepção estrutural não estava mais vinculada à geometria do sistema, mas sim à sua descrição determinística, ao mapeamento das propriedades do sistema estudado (através de métodos contínuos) e passou a compreender as leis da mecânica com aplicação prática, explorando o limite de plasticidade de cada material (Lobosco, 2017; Oxman, 2011).

¹ Disponível em : <<http://estruturandocivil.blogspot.com/2016/01/antoni-gaudi-e-sua-estrutura-gotica.html>> . Acesso em : 23 mai. 2021.

Os modelos estruturais, neste novo contexto, apresentam pontos negativos para concepção estrutural, apesar disso, ainda permitem contato com propriedades qualitativas do fenômeno, mas sem exigir domínio profundo da dinâmica do sistema. Modelos didáticos estruturais atuam simplificando o estudo do fenômeno, o tornando uma ferramenta valiosa para abordagem sintética das leis da mecânica dos sólidos (LEET et al., 2014; Pravia 2000).

5.1.1 Modelos Didáticos

Modelos didáticos, em sua maioria, visam uma máquina simples, que demonstre deformações, valorizando aspectos técnicos dos fenômenos. Contudo, a qualidade de um modelo didático deveria ser mensurada por suas características pedagógicas, como citado por Lobosco (2017):

A qualidade de um modelo didático não deve ser medida através da efetiva e rigorosa tradução do comportamento da estrutura em escala, nem mesmo pela reprodução dos valores numéricos de resistência e ruptura. Seu comportamento não deve necessariamente ser realístico, apenas, deve transmitir conceitualmente a lógica interna e os padrões de comportamento que fazem uma estrutura funcionar ou romper (LOBOSCO 2017).

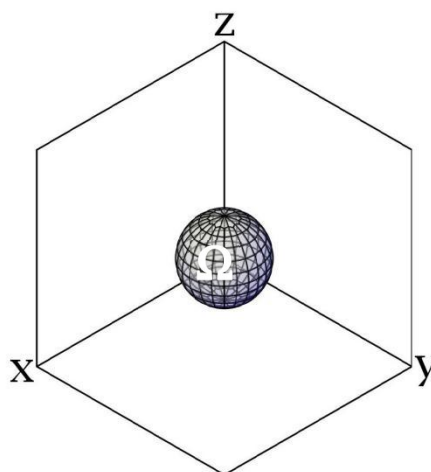
Em contraste aos modelos estruturais clássicos (maquetes de concepção estrutural), essas ferramentas de ensino (modelos didáticos) não deveriam ser usadas no dimensionamento/cálculo estrutural. Idealmente, deveriam ser aplicáveis, como uma experiência qualitativa, mesmo que os fenômenos desse experimento sejam quantitativos (Moura, 2016; Lobosco, 2017).

5.2 MECÂNICA CONTÍNUA

A mecânica contínua é uma abordagem matemática para descrição do comportamento de um sistema de interesse, o objeto é descrito como um domínio de controle Ω (Mase, 1970).

Para mapear o comportamento de sistemas físicos sólidos, e deformáveis, é suposto que o domínio é contínuo em seu volume de controle, conforme figura 05. Isso quer dizer que são desconsideradas as discontinuidades (espaço entre átomos e moléculas, ou divergências geométricas) (Mase, 1970; Balankin, 2004).

Figura 05 - Objeto, volume de controle, ou sistema em análise Ω .

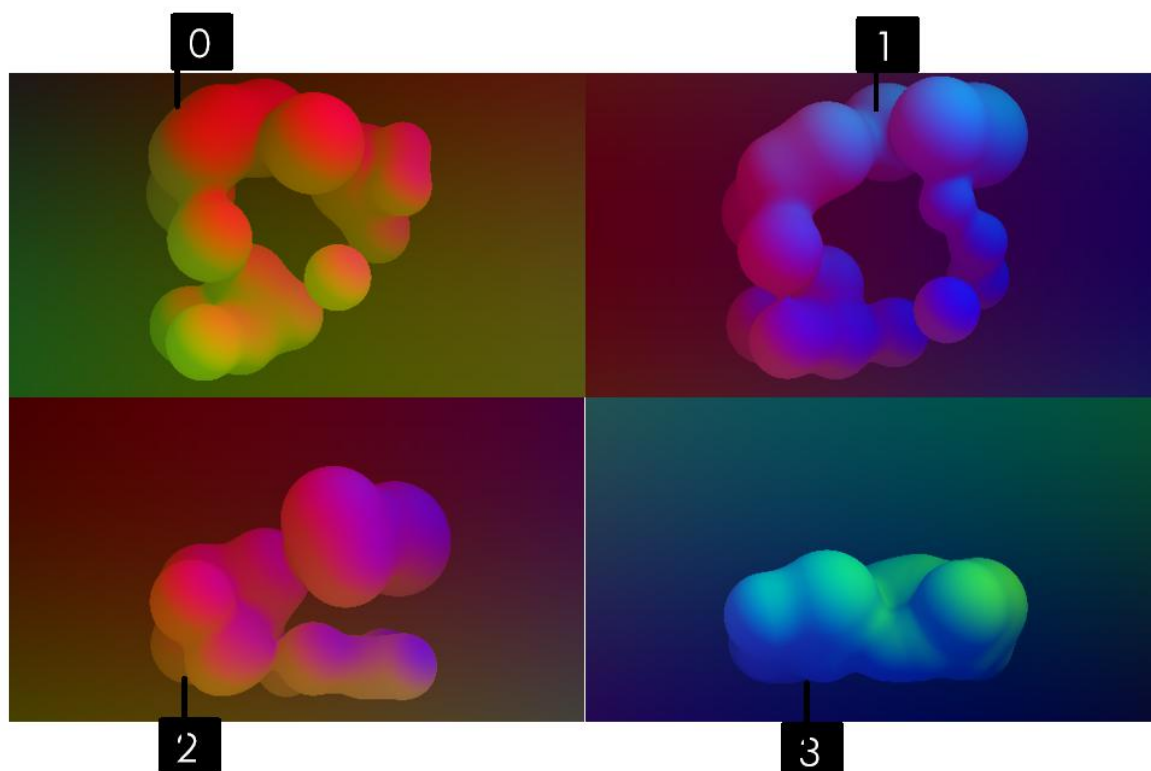


Fonte: Elaborado pelo autor, 2021.

A vantagem de ter um volume contínuo é a possibilidade de trabalhar com componentes infinitesimais, e atribuir propriedades de interesse, para descrever como essas propriedades se expressam globalmente no domínio (relação constitutiva). (Balankin, 2004).

O primeiro aspecto que realmente torna a mecânica contínua especial é essencialmente sobre a distorção de corpos sólidos, conforme figura 06, e é comum que a Deformação seja avaliada como Distorção, mas na realidade a deformação não é sobre distorção, mas sim sobre Movimento e Contenção, enquanto que a distorção é apenas uma consequência de como o conjunto reage à isso.

Figura 06 - Geometria em diferentes fases de distorção.



Fonte: Threejs examples (adaptado pelo autor)¹

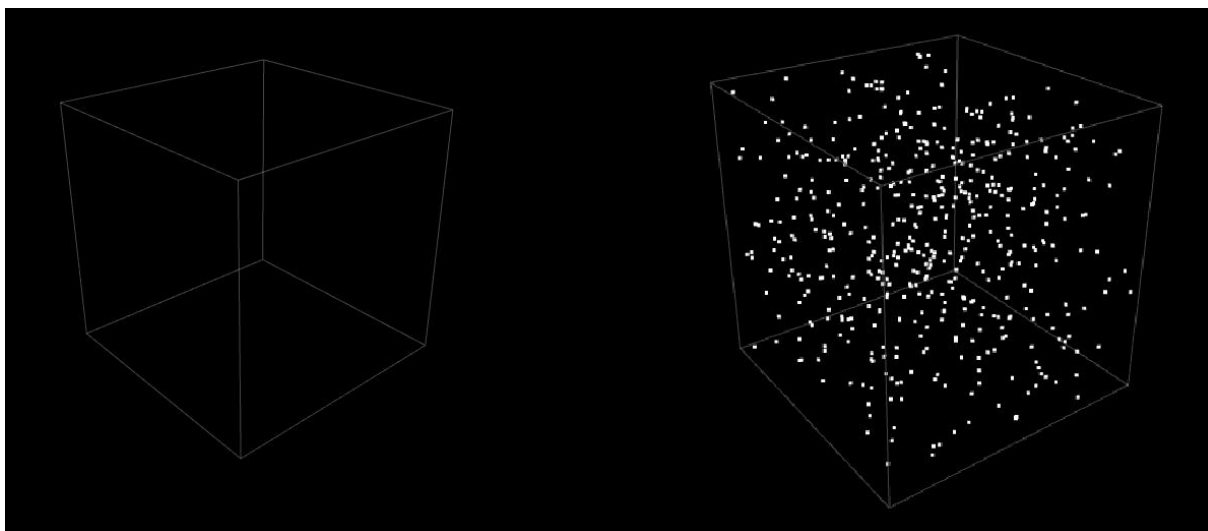
Contudo, fazer uma avaliação analítica (até do sistema mais simples) pode envolver extremamente complexidade, e dependendo de parâmetros incongruentes como assimetrias geométricas, graus de liberdade e condições de contorno variáveis, o sistema pode se mostrar impossível de ser resolvido.

Uma alternativa é o uso de uma malha Finita, ou seja, subdividir a geometria em pequenos elementos, e com esses simular o comportamento real do sistema. Essa desmembração em passos finitos permite a conversão da avaliação analítica para a aplicação de métodos numéricos, possibilitando a avaliação de geometrias complicadas, e condições de contorno variáveis (Torrati, 2018).

Para alcançar esse nível de adaptabilidade, primeiro a geometria deve ser discretizada (foram desenvolvidos extensivamente diversificados métodos de discretização), esse processo pode ser subdividido em duas principais etapas, a primeira é imaginar a geometria como uma nuvem de pontos (contidos pelo contorno da geometria), conforme figura 07.

¹Disponível em : <<https://threejs.org/examples/?l>> . Acesso em : 12 junho. 2021.

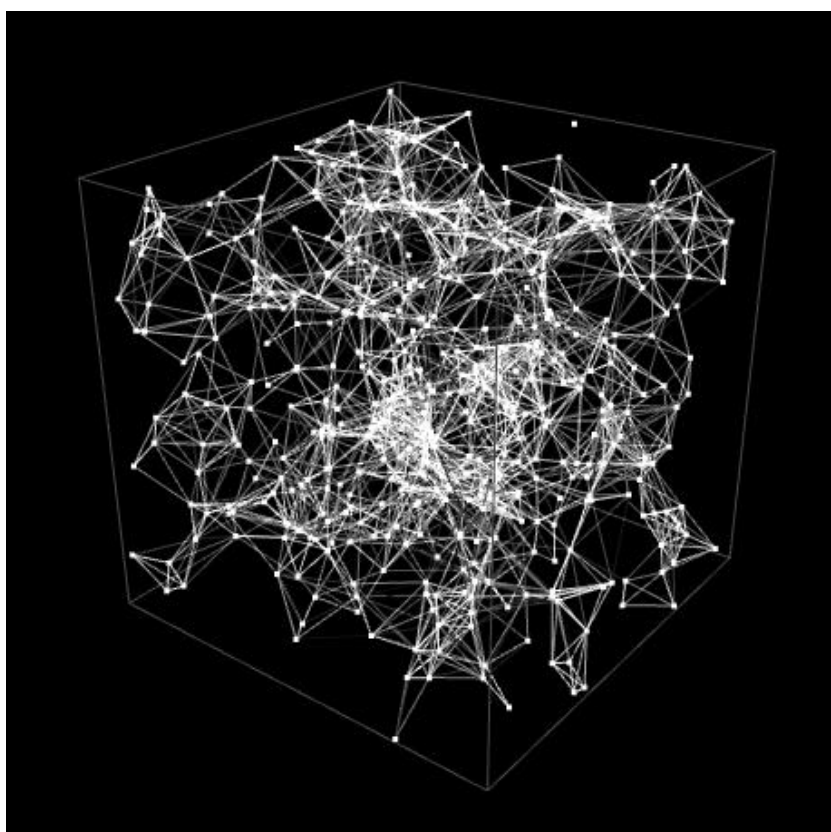
Figura 07 - Geometria contendo uma nuvem de pontos.



Fonte: Elaborado pelo autor, 2021.

Discretizar é escolher como esses pontos serão conectados para que a geometria seja preenchida. Podem ser usados elementos volumétricos (Paralelogramas ou Hexagramas) ou elementos unidimensionais (molas, barras, vigas), conforme figura 08.

Figura 08 - Geometria discretizada em: Barras.



Fonte: Elaborado pelo autor, 2021.

A escolha dos elementos que comporão a geometria resulta em mais implicações do que apenas no gerenciamento final dos nós, tem impacto direto na formulação dos elementos. A formulação de elementos volumétricos (como paralelogramas e hexagramas) envolve tensores tridimensionais de Cauchy, lei ideal dos Gases, Tensores de Green, e constante de Poisson (usados para garantir que o elemento mantenha estabilidade volumétrica), já em tensores unidimensionais (molas, barras, vigas) pode ser avaliada (se necessário) a influência dos tensores de Green e da constante de Poisson (Maset, 2010).

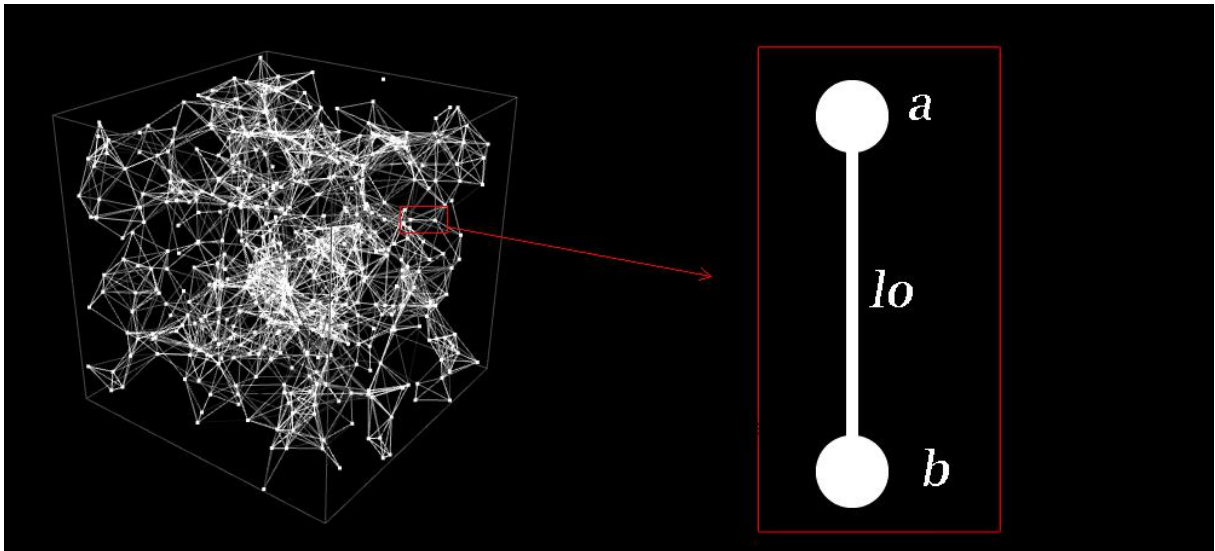
Devido a natureza dessa Pesquisa, para simplificar tanto no desenvolvimento da formulação (quando no gerenciamento futuro dos nós) a formulação será desenvolvida para elementos unidimensionais (molas e barras), e como o interesse estará voltado para a deformação global da geometria será desconsiderada a influencia da constante de Poisson nas conexões individuais. A formulação será desenvolvida em quatro etapas principais:

- I. Análise individual e generalizada de um elemento genérico;
- II. *Assembly* das formulações fragmentadas desenvolvidas individualmente em cada elemento;
- III. Adição de Condições de Contorno de Neumann e Dirichlet (respectivamente: deformações e esforços externos conhecidos);
- IV. Resolução Numérica do sistema matricial final para descobrir os graus de deformação.

5.2.1 Análise individual e generalizada de um elemento genérico

Deve ser Isolado um elemento genérico da malha discretizada da geometria, para analisar individualmente o comportamento dos elementos, conforme figura 09.

Figura 09 -Isolamento de um elemento de barra genérico.

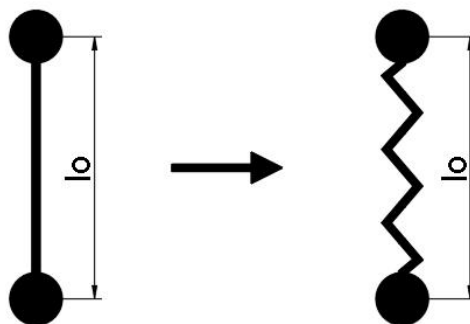


Fonte: Elaborado pelo autor, 2021.

Para fins didáticos, sistemas unidimensionais (mola e barra) podem ser avaliados como sistemas de massa mola, conforme Figura 10, (que são regidos pela lei de Hooke Eq. 0.1) .

$$F_s = k_s \cdot x \quad (0.1)$$

Figura 10 - Conversão do elemento de Barra para um elemento de Massa Mola.



Fonte: Elaborado pelo autor, 2021.

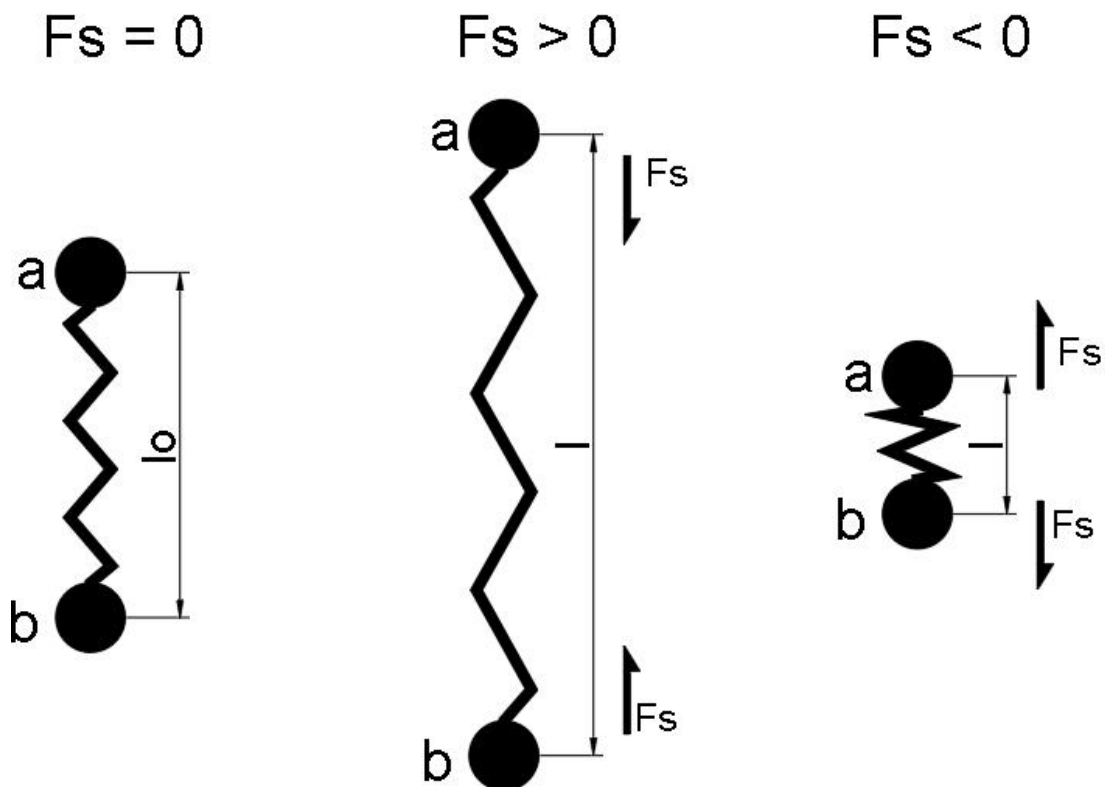
Convencionalmente, a Lei de Hooke (Eq. 0.1) é escrita com sinal negativo para representar a força que a mola produz, ou com sinal positivo para representar a força sendo aplicada na mola. Contudo, isso apenas é aplicado para modelos simplificados, ao lidar com um conjunto massivo de elementos se torna mais conveniente usar análises locais.

Quando as duas partículas (partículas/massas nas extremidades da mola, por convenção nomea-las de 'a' e 'b') são distanciadas (a uma distancia superior ao comprimento relaxado da mola = l_0) a força exercida pela mola é positiva, e quando as partículas são aproximadas (a uma distancia inferior a l_0) a força exercida pela mola é negativa (Eq 0.2, Figura 11) (Maset, 2010).

$$F_s = k_s \cdot (\| \vec{l} \| - l_0)$$

$$\Leftrightarrow F_s = k_s \cdot (\| \vec{a} - \vec{b} \| - l_0) \quad (0.2)$$

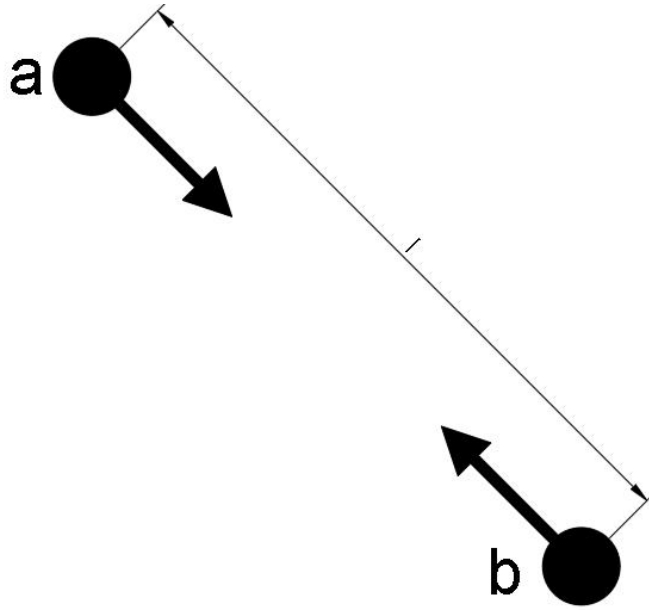
Figura 11 - Influencia da direção da deformação com o sinal da força da mola.



Fonte: Elaborado pelo autor, 2021.

Um método simples de gerenciar se as partículas estão sendo aproximadas ou distanciadas uma da outra é encontrando o vetor normalizado que aponta de uma para a outra, conforme figura 12,

Figura 12 - Vetores normalizados.



Fonte: Elaborado pelo autor, 2021.

sendo esse vetor ' $\hat{l}$ ', a normalização do vetor de distância ' $\vec{l}$ ' entre ' a ' e ' b ' (Eq. 0.3),

$$\hat{l} = \frac{\vec{l}}{\|\vec{l}\|} = \frac{\vec{a} - \vec{b}}{\|\vec{a} - \vec{b}\|} \quad (0.3)$$

e lembrando considerando que o vetor normalizado em ' a ' com relação à ' b ', é oposto ao vetor normalizado de ' b ' com relação à ' a ' (Eq. 0.4).

$$\begin{aligned} \widehat{l_{ba}} &= -\widehat{l_{ab}} \\ \Leftrightarrow \frac{\vec{a} - \vec{b}}{\|\vec{a} - \vec{b}\|} &= \frac{\vec{b} - \vec{a}}{\|\vec{a} - \vec{b}\|} \\ \Leftrightarrow \widehat{l_{ab}} &= \frac{\vec{b} - \vec{a}}{\|\vec{a} - \vec{b}\|} \\ \Leftrightarrow \widehat{l_{ba}} &= \frac{\vec{a} - \vec{b}}{\|\vec{a} - \vec{b}\|} \end{aligned} \quad (0.4)$$

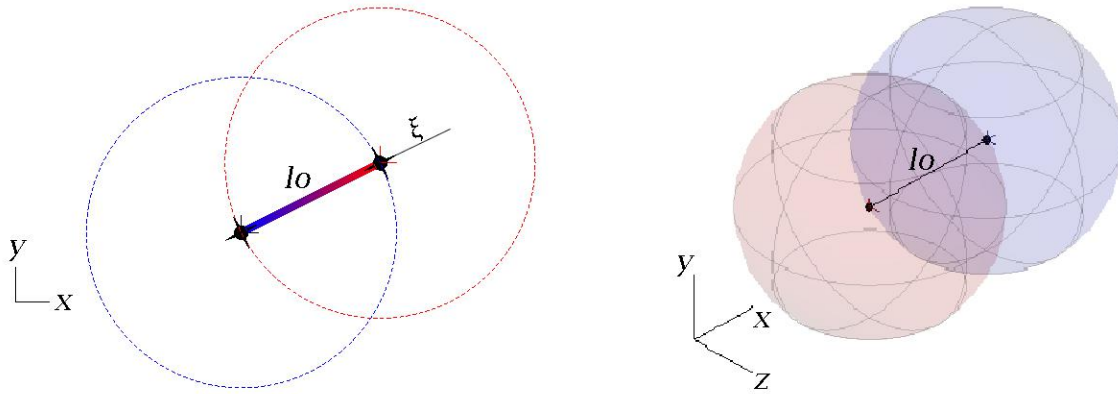
Para converter a força interna da mola (F_s) para sua correspondente nas partículas ' a ' e ' b ', a F_s precisa ser multiplicada pelo vetor normalizado particular de cada ponto.

Dessa maneira, uma força positiva vai significar que a partícula está se aproximando, enquanto que uma força negativa significará que ela estará se distanciando (Eq. 0.5).

$$\begin{aligned}
 F_a &= k_s \cdot (\|\vec{a} - \vec{b}\| - l_0) \cdot \widehat{ab} \\
 F_b &= k_s \cdot (\|\vec{a} - \vec{b}\| - l_0) \cdot \widehat{ba} \\
 \Leftrightarrow F_a &= k_s \cdot (\|\vec{a} - \vec{b}\| - l_0) \cdot \frac{\vec{b} - \vec{a}}{\|\vec{a} - \vec{b}\|} \\
 \Leftrightarrow F_b &= k_s \cdot (\|\vec{a} - \vec{b}\| - l_0) \cdot \frac{\vec{a} - \vec{b}}{\|\vec{a} - \vec{b}\|}
 \end{aligned} \tag{0.5}$$

Uma propriedade importante desse sistema é a semelhança do seu comportamento com estudos de orbitação, sendo nesse caso, a força de conexão da gravidade substituída pela força elástica da mola. Revelando uma característica impactante do sistema, a Equipotencialidade (enquanto o movimento da partícula obedecer a orbita traçada em torno da outra, a energia do sistema não será afetada), conforma Figura 13.

Figura 13 - Orbitas particulares de uma partícula com relação à outra.



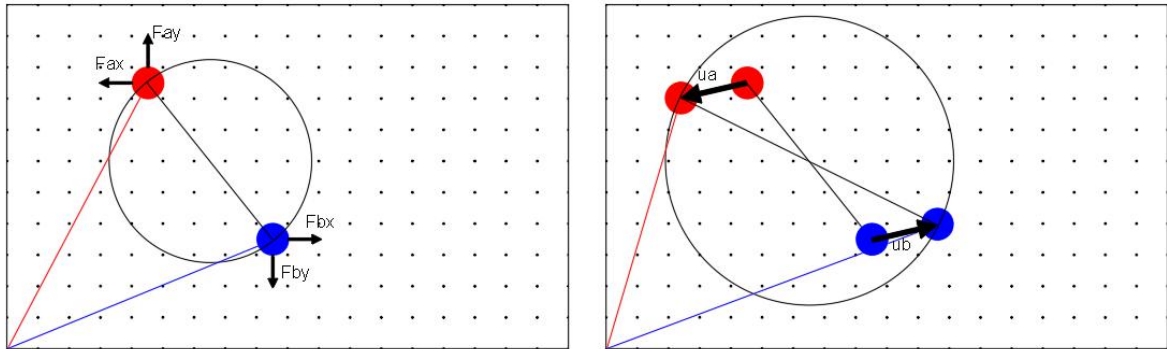
Fonte: Elaborado pelo autor, 2021.

O objetivo de desenvolver uma formulação que descreva esse sistema é prever qual será o movimento das partículas 'a' e 'b' (em consequencia de ações/carregamentos/esforços externos), e discutir como esse 'movimento' está relacionado com a 'conexão' entre as partículas. Nesse contexto, quando o 'movimento' está acontecendo no eixo da 'conexão' entre os pontos 'a' e 'b', 'movimento' é caracterizado como **Deformação**, e a conexão se

manifesta como uma espécie de **Contenção**. Quando o ‘movimento’ não afeta a distância entre os pontos ‘a’ e ‘b’, ‘movimento’ é caracterizado como translação/rotação/orbitação, e a conexão manifesta o ‘caminho’ descrito pela orbita.

Mas descrever uma previsão desse sistema (que possui duas condições de orbitação) pode ser surpreendentemente desafiador, conforme Figura 14,

Figura 14 - Deformação genérica do sistema (direita - sem deformação, esquerda - deformado).



Fonte: Elaborado pelo autor, 2021.

usar análise vetorial das Leis de Newton tornaria a formulação desnecessariamente complexa (pois esse tipo de análise não lida bem com condições de contorno variáveis). Contudo, como a formulação deseja apenas descrever o caminho percorrido pelas partículas ‘a’ e ‘b’ entre dois eventos, a mecânica Lagrangiana se mostra mais conveniente (Eq. 0.6) (pois nesse tipo de análise são trabalhadas ações no lugar de vetores).

$$\mathcal{L}_{(q_i, \dot{q}_i, t)} = T - U \quad (0.6)$$

Em resumo a equação diz que o *Lagrangiano* ($\mathcal{L}$) é igual a diferença entre a *Energia Cinética* (T) e a *Energia Potencial* (U), e seguindo-se com o formalismo de Lagrange, tem-se que a equação de movimento deve satisfazer à equação de Lagrange (Eq. 0.7),

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}_{(q_i, \dot{q}_i, t)}}{\partial \dot{q}_i} \right) = \frac{\partial \mathcal{L}_{(q_i, \dot{q}_i, t)}}{\partial q_i} \quad (0.7)$$

Portanto, a formulação do sistema pela equação de lagrange será desenvolvida em três etapas: calcular a *Energia Cinética do Sistema* (T), calcular a *Energia Potencial do Sistema* (U), e finalmente aplica-las na *equação de Lagrange* (Eq. 0.7).

Então o primeiro passo será calcular a *Energia Cinética do Sistema* (T) (Eq. 0.8),

$$\begin{aligned}
 T &= T_a + T_b \\
 \Leftrightarrow T &= \frac{m_a \cdot [v_a]^2}{2} + \frac{m_b \cdot [v_b]^2}{2} \\
 \Leftrightarrow T &= \frac{m_a \cdot [\dot{x}_a^2 + \dot{y}_a^2 + \dot{z}_a^2]}{2} + \frac{m_b \cdot [\dot{x}_b^2 + \dot{y}_b^2 + \dot{z}_b^2]}{2} \\
 \Leftrightarrow T &= \frac{m_a \cdot [\dot{\vec{a}}^2]}{2} + \frac{m_b \cdot [\dot{\vec{b}}^2]}{2} \\
 \Leftrightarrow T &= \frac{m}{2} \cdot \left\{ [\dot{\vec{a}}^2] + [\dot{\vec{b}}^2] \right\}
 \end{aligned} \tag{0.8}$$

o segundo passo será calcular a *Energia Potencial do Sistema* (U) (Eq. 0.9),

$$U = \frac{k_s}{2} [\|\vec{a} - \vec{b}\| - \vec{l}_0]^2 \tag{0.9}$$

(por conveniência de calculo, a partir de agora serão feitas considerações práticas: Primeiro, será imaginado que a estrutura está sobre um sistema cartesiano unidimensional ξ , e a expressão da Eq. 0.9 será reescrita em função das deformações $\vec{u}_{a\xi}$ e $\vec{u}_{b\xi}$ (Eq. 1.0)),

$$\begin{aligned}
 \vec{a} &= \vec{a}_0 + \vec{u}_{a\xi} \\
 \vec{b} &= \vec{b}_0 + \vec{u}_{b\xi} \\
 \vec{l}_0 &= \vec{b}_0 - \vec{a}_0 \\
 \therefore U &= \frac{k_s}{2} [\|\vec{a}_0 + \vec{u}_{a\xi} - \vec{b}_0 - \vec{u}_{b\xi}\| - \vec{l}_0]^2 \\
 \Leftrightarrow U &= \frac{k_s}{2} [\|\vec{a}_0 - \vec{b}_0 + \vec{u}_{a\xi} - \vec{u}_{b\xi}\| - \vec{l}_0]^2 \\
 \Leftrightarrow U &= \frac{k_s}{2} [\|\vec{l}_0 + (\vec{u}_{a\xi} - \vec{u}_{b\xi})\| - \vec{l}_0]^2
 \end{aligned} \tag{1.0}$$

$$\Leftrightarrow U = \frac{k_s}{2} [\bar{u}_{a\xi} - \bar{u}_{b\xi}]^2$$

e finalmente aplica-las na *equação de Lagrange* (Eq. 1.1)

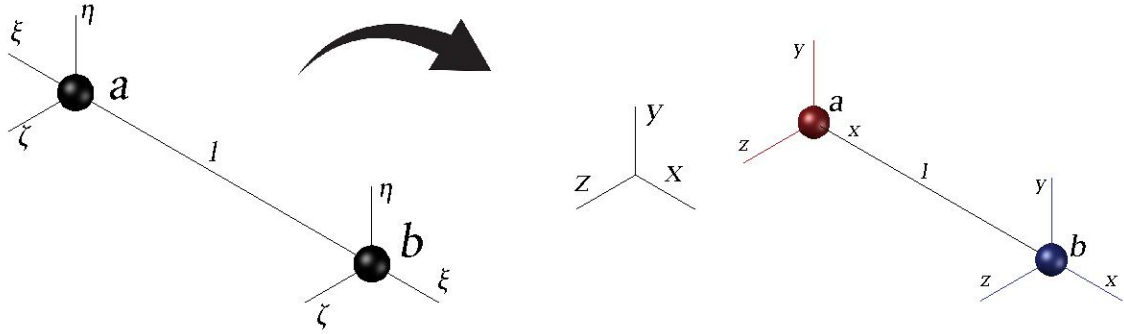
$$\begin{aligned} \mathcal{L}_{(q_i, \dot{q}_i, t)} &= T - U \\ \Leftrightarrow \mathcal{L}_{(q_i, \dot{q}_i, t)} &= \frac{m}{2} \cdot \left\{ [\dot{\bar{a}}^2] + [\dot{\bar{b}}^2] \right\} - \frac{k_s}{2} [\bar{u}_{a\xi} - \bar{u}_{b\xi}]^2 \\ \frac{d}{dt} \left(\frac{\partial \mathcal{L}_{(\bar{a}_i, \dot{\bar{a}}_i, t)}}{\partial \dot{\bar{a}}_i} \right) &= m \cdot \ddot{\bar{a}} + 0 \\ \frac{\partial \mathcal{L}_{(\bar{a}_i, \dot{\bar{a}}_i, t)}}{\partial \bar{a}_i} &= 0 - k_s \cdot [\bar{u}_{a\xi} - \bar{u}_{b\xi}] \\ & \\ \frac{d}{dt} \left(\frac{\partial \mathcal{L}_{(\bar{b}_i, \dot{\bar{b}}_i, t)}}{\partial \dot{\bar{b}}_i} \right) &= m \cdot \ddot{\bar{b}} + 0 \\ \frac{\partial \mathcal{L}_{(\bar{b}_i, \dot{\bar{b}}_i, t)}}{\partial \bar{b}_i} &= 0 - k_s \cdot [\bar{u}_{a\xi} - \bar{u}_{b\xi}] \cdot (-1) \\ & \\ \Leftrightarrow -m \cdot \ddot{\bar{a}} &= k_s \cdot [\bar{u}_{a\xi} - \bar{u}_{b\xi}] \\ \Leftrightarrow -m \cdot \ddot{\bar{b}} &= k_s \cdot [-\bar{u}_{a\xi} + \bar{u}_{b\xi}] \end{aligned} \tag{1.1}$$

As expressões da Eq. 1.1 podem ser reescritas em forma matricial (pois são expressões lineares) (Eq. 1.2),

$$\begin{aligned} \begin{bmatrix} -m \cdot \ddot{\bar{a}} \\ -m \cdot \ddot{\bar{b}} \end{bmatrix} &= k_s \cdot \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \bar{u}_{a\xi} \\ \bar{u}_{b\xi} \end{bmatrix} \\ \Leftrightarrow [k]_{\xi}^{2 \times 2} \cdot [u]_{\xi}^T &= [f]_{\xi}^T \end{aligned} \tag{1.2}$$

Contudo, a formulação foi desenvolvida para o eixo particular ' $\langle \xi \rangle$ ', mas os elementos estão nos eixos cartesianos $\langle x, y, z \rangle$. Por isso, para poder trabalhar nesses eixos a formulação deverá ser adaptada com transformações de coordenadas (Figura 15).

Figura 15 - Transformação de coordenadas.



Fonte: Elaborado pelo autor, 2021.

Qualquer vetor (contanto que não seja um tensor) pode ser transformado entre coordenadas sem prejudicar as propriedades do sistema, o primeiro passo é escrever os eixos unitários de cada sistema de coordenadas (Eq. 1.3)

Sistema Particular $\langle \xi \rangle$

$$\hat{\xi} = \langle \hat{i}_x, \hat{i}_y, \hat{i}_z \rangle$$

$$\Leftrightarrow \hat{\xi} = \frac{\langle \vec{b}_x - \vec{a}_x, \vec{b}_y - \vec{a}_y, \vec{b}_z - \vec{a}_z \rangle}{\|\vec{b} - \vec{a}\|}$$

Sistema Cartesiano $\langle x, y, z \rangle$

$$\hat{x} = \langle 1, 0, 0 \rangle$$

$$\hat{y} = \langle 0, 1, 0 \rangle$$

$$\hat{z} = \langle 0, 0, 1 \rangle$$

(1.3)

Usando o vetor $\vec{l}$ de exemplo, ele pode ser escrito de três maneiras em cada um dos sistemas de coordenadas (Eq. 1.4).

Sistema Particular $\langle \xi \rangle$

$$\vec{l} = \vec{l}_\xi$$

$$\vec{l} = \langle \|\vec{l}\|_\xi \rangle$$

Sistema Cartesiano $\langle x, y, z \rangle$

$$\vec{l} = \vec{l}_x + \vec{l}_y + \vec{l}_z$$

$$\vec{l} = \langle \|\vec{l}\|_x, \|\vec{l}\|_y, \|\vec{l}\|_z \rangle$$

(1.4)

$$\|\vec{l}\|_{\xi} = \vec{l} \cdot \hat{\xi} \quad \vec{l} = \|\vec{l}\|_x \cdot \hat{x} + \|\vec{l}\|_y \cdot \hat{y} + \|\vec{l}\|_z \cdot \hat{z}$$

Como o objetivo é escrever o sistema particular $\langle \xi \rangle$ com relação as coordenadas $\langle x, y, z \rangle$ (Eq. 1.5),

$$\begin{aligned} \|\vec{l}\|_{\xi} &= \vec{l} \cdot \hat{\xi} \\ \vec{l} &= \|\vec{l}\|_x \cdot \hat{x} + \|\vec{l}\|_y \cdot \hat{y} + \|\vec{l}\|_z \cdot \hat{z} \\ \therefore \|\vec{l}\|_{\xi} &= \left(\|\vec{l}\|_x \cdot \hat{x} + \|\vec{l}\|_y \cdot \hat{y} + \|\vec{l}\|_z \cdot \hat{z} \right) \cdot \hat{\xi} \\ \Leftrightarrow \|\vec{l}\|_{\xi} &= \left(\|\vec{l}\|_x \cdot \hat{x} \cdot \hat{\xi} + \|\vec{l}\|_y \cdot \hat{y} \cdot \hat{\xi} + \|\vec{l}\|_z \cdot \hat{z} \cdot \hat{\xi} \right) \\ \Leftrightarrow \|\vec{l}\|_{\xi} &= \left(\|\vec{l}\|_x \cdot (\hat{x} \cdot \hat{\xi}) + \|\vec{l}\|_y \cdot (\hat{y} \cdot \hat{\xi}) + \|\vec{l}\|_z \cdot (\hat{z} \cdot \hat{\xi}) \right) \end{aligned} \quad (1.5)$$

$$\Leftrightarrow \|\vec{l}\|_{\xi} = \begin{pmatrix} \|\vec{l}\|_x \cdot (\|\hat{x}\| \cdot \|\hat{\xi}\| \cdot \cos x\xi) \\ + \|\vec{l}\|_y \cdot (\|\hat{y}\| \cdot \|\hat{\xi}\| \cdot \cos y\xi) \\ + \|\vec{l}\|_z \cdot (\|\hat{z}\| \cdot \|\hat{\xi}\| \cdot \cos z\xi) \end{pmatrix}$$

O produto vetorial entre vetores unitários depende apenas do cosseno do ângulo (Eq. 1.6),

$$\begin{aligned} 1 &= \|\hat{x}\| \cdot \|\hat{\xi}\| \cdot \cos x\xi = \langle 1, 0, 0 \rangle \cdot \frac{\langle \vec{b}_{ox} - \vec{a}_{ox}, \vec{b}_{oy} - \vec{a}_{oy}, \vec{b}_{oz} - \vec{a}_{oz} \rangle}{\|\vec{b}_o - \vec{a}_o\|} \\ \Leftrightarrow 1 &= \cos x\xi = \frac{\vec{b}_{ox} - \vec{a}_{ox}}{\|\vec{b}_o - \vec{a}_o\|} \\ m &= \|\hat{y}\| \cdot \|\hat{\xi}\| \cdot \cos y\xi = \langle 0, 1, 0 \rangle \cdot \frac{\langle \vec{b}_{ox} - \vec{a}_{ox}, \vec{b}_{oy} - \vec{a}_{oy}, \vec{b}_{oz} - \vec{a}_{oz} \rangle}{\|\vec{b}_o - \vec{a}_o\|} \\ \Leftrightarrow m &= \cos y\xi = \frac{\vec{b}_{oy} - \vec{a}_{oy}}{\|\vec{b}_o - \vec{a}_o\|} \\ n &= \|\hat{z}\| \cdot \|\hat{\xi}\| \cdot \cos z\xi = \langle 0, 0, 1 \rangle \cdot \frac{\langle \vec{b}_{ox} - \vec{a}_{ox}, \vec{b}_{oy} - \vec{a}_{oy}, \vec{b}_{oz} - \vec{a}_{oz} \rangle}{\|\vec{b}_o - \vec{a}_o\|} \\ \Leftrightarrow n &= \cos z\xi = \frac{\vec{b}_{oz} - \vec{a}_{oz}}{\|\vec{b}_o - \vec{a}_o\|} \end{aligned} \quad (1.6)$$

Usando a notação da Eq. 1.6, a expressão pode ser ajustada (Eq. 1.7)

$$\begin{aligned}
\|\vec{l}\|_{\xi} &= \|\vec{l}\|_x \cdot \cos x\xi + \|\vec{l}\|_y \cdot \cos y\xi + \|\vec{l}\|_z \cdot \cos z\xi \\
\Leftrightarrow \|\vec{l}\|_{\xi} &= \|\vec{l}\|_x \cdot 1 + \|\vec{l}\|_y \cdot m + \|\vec{l}\|_z \cdot n \\
\Leftrightarrow [\vec{l}]_{\xi}^T &= [l, m, n] \cdot \begin{bmatrix} \vec{l}_x \\ \vec{l}_y \\ \vec{l}_z \end{bmatrix}_{x,y,z}^T \\
\Leftrightarrow [\vec{l}]_{\xi}^T &= [Th] \cdot [\vec{l}]_{x,y,z}^T
\end{aligned} \tag{1.7}$$

O vetore de transformação $[Th]$ pode ser usado para converter qualquer vetor (através desses dois tipos de coordenadas) (Eq. 1.8)

$$\begin{aligned}
[u]_{\xi}^T &= [Th] \cdot [u]_{x,y,z}^T \\
[f]_{\xi}^T &= [Th] \cdot [f]_{x,y,z}^T \\
\therefore [k]_{\xi}^{2 \times 2} \cdot [u]_{\xi}^T &= [f]_{\xi}^T \Leftrightarrow [k]_{\xi}^{2 \times 2} \cdot [u]_{\xi}^T = [Th] \cdot [f]_{x,y,z}^T
\end{aligned} \tag{1.8}$$

$$\begin{aligned}
\Leftrightarrow [k]_{\xi}^{2 \times 2} \cdot [Th] \cdot [u]_{x,y,z}^T &= [Th] \cdot [f]_{x,y,z}^T \\
\Leftrightarrow [Th]^T \cdot [k]_{\xi}^{2 \times 2} \cdot [Th] \cdot [u]_{x,y,z}^T &= [f]_{x,y,z}^T \\
\Leftrightarrow [k]_{x,y,z}^{6 \times 6} \cdot [u]_{x,y,z}^T &= [f]_{x,y,z}^T
\end{aligned}$$

Finalmente $[Th]^T \cdot [k]_{\xi}^{2 \times 2} \cdot [Th]$ será equivalente à $[k]_{x,y,z}^{6 \times 6}$ (Eq. 1.9)

$$\begin{aligned}
[k]_{x,y,z}^{6 \times 6} &= [Th]^T \cdot [k]_{\xi}^{2 \times 2} \cdot [Th] \\
\Leftrightarrow [k]_{x,y,z}^{6 \times 6} &= \begin{bmatrix} 0 & l \\ 0 & m \\ 0 & n \\ l & 0 \\ m & 0 \\ n & 0 \end{bmatrix}^T \cdot k_s \cdot \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}_{\xi}^{2 \times 2} \cdot \begin{bmatrix} l & m & n & 0 & 0 & 0 \\ 0 & 0 & 0 & l & m & n \end{bmatrix}
\end{aligned} \tag{1.9}$$

$$\Leftrightarrow [k]_{x,y,z}^{6 \times 6} = k_s \cdot \begin{bmatrix} ll & lm & ln & -ll & -lm & -ln \\ ml & mm & mn & -ml & -mm & -mn \\ nl & nm & nn & -nl & -nm & -nn \\ -ll & -lm & -ln & ll & lm & ln \\ -ml & -mm & -mn & ml & mm & mn \\ -nl & -nm & -nn & nl & nm & nn \end{bmatrix}_{x,y,z}^{6 \times 6}$$

Enfim, a expressão global para o elemento de massa mola está concluída. Essa abordagem (de formular com uma mola) foi adotada por ser mais intuitiva, para converter essa expressão para a descrição de uma barra é necessário trocar o coeficiente de rigidez k_s pelo de Green (Eq. 2.0)

$$k_s = \frac{EA}{l_0} \quad (2.0)$$

Portanto, a expressão local de deformação do elemento está completa (Eq. 2.1)

$$[k]_{x,y,z} \cdot [u]_{x,y,z}^T = [f]_{x,y,z}^T$$

$$\frac{EA}{l_0} \cdot \begin{bmatrix} ll & lm & ln & -ll & -lm & -ln \\ ml & mm & mn & -ml & -mm & -mn \\ nl & nm & nn & -nl & -nm & -nn \\ -ll & -lm & -ln & ll & lm & ln \\ -ml & -mm & -mn & ml & mm & mn \\ -nl & -nm & -nn & nl & nm & nn \end{bmatrix}_{x,y,z}^{6 \times 6} \cdot \begin{bmatrix} \bar{u}_x^a \\ \bar{u}_y^a \\ \bar{u}_z^a \\ \bar{u}_x^b \\ \bar{u}_y^b \\ \bar{u}_z^b \end{bmatrix}_{x,y,z}^T = \begin{bmatrix} \bar{f}_x^a \\ \bar{f}_y^a \\ \bar{f}_z^a \\ \bar{f}_x^b \\ \bar{f}_y^b \\ \bar{f}_z^b \end{bmatrix}_{x,y,z}^T \quad (2.1)$$

5.2.2 *Assembly* das formulações fragmentadas desenvolvidas individualmente em cada elemento

Com a formulação individual concluída, deverá ser desenvolvida uma formulação global. Esse processo é frequentemente chamado de *Assembly* (montagem) e consiste do processo simples de fazer a montagem das matrizes individuais em uma matriz global.

A matriz de rigidez Global é uma matriz quadrada com dimensões proporcionais aos graus de liberdade (n) do sistema (Eq. 2.2).

$$[K]_{x,y,z}^{nxn} = \begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & \dots & k_{1,n} \\ k_{2,1} & k_{2,2} & k_{2,3} & \dots & k_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & k_{n,3} & \dots & k_{n,n} \end{bmatrix}_{x,y,z}^{nxn} \quad (2.2)$$

Em resumo, quando o processo é iniciado os dados calculados em cada matriz $[k]_{x,y,z}^{6x6}$ deve ser somado a sua Coordenada correspondente dentro da matriz Global $[K]_{x,y,z}^{nxn}$.

5.2.3 Adição de Condições de Contorno de Neumann e Dirichlet (respectivamente: deformações e esforços externos conhecidos)

Existem dois principais tipos de condições de contorno, Neumann e Dirichlet, cada uma dessas tem suas implicações individuais. Condições de contorno de Neumann trabalham com ações expositivas na estrutura, ou seja, carregamentos, portanto são manifestadas na construção da matriz Global de Esforços $[F]_{x,y,z}^T$, aplicar esse tipo de condição de contorno significa injetar diretamente o valor desejado para o grau de liberdade de interesse dentro de $[F]_{x,y,z}^T$.

Já as condições de contorno de Dirichlet têm impacto mais severo na resolução numérica do sistema, esse tipo de condição de contorno trabalha com as deformações e apoios, portanto são manifestadas na construção da matriz Global de Deformações $[U]_{x,y,z}^T$, como nas condições de Neumann o primeiro passo é apenas injetar o valor desejado no grau de liberdade de interesse, contudo deve se tomar cuidado pois toda a expressão envolvida pelo grau de liberdade definido deixa de ser iterável.

Depois de aplicadas as Condições de Contorno a formulação global do sistema está concluída (Eq. 2.3).

$$\begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & \dots & k_{1,n} \\ k_{2,1} & k_{2,2} & k_{2,3} & \dots & k_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & k_{n,3} & \dots & k_{n,n} \end{bmatrix}_{x,y,z}^{nxn} \bullet \begin{bmatrix} \bar{u}_1 \\ \bar{u}_2 \\ \vdots \\ \bar{u}_n \end{bmatrix}_{x,y,z}^T = \begin{bmatrix} \bar{f}_1 \\ \bar{f}_2 \\ \vdots \\ \bar{f}_n \end{bmatrix}_{x,y,z}^T \quad (2.3)$$

$$\therefore [K]_{x,y,z}^{nxn} \bullet [U]_{x,y,z}^T = [F]_{x,y,z}^T$$

5.2.4 Resolução Numérica do sistema matricial final para descobrir os graus de deformação.

Os métodos mais simples para resolução numérica de sistemas lineares são os métodos de Jacobi e Gauss Seidel (são semelhantes pois o método de Gauss Seidel é um aprimoramento do método de Jacobi).

O método de Jacobi obedece a equação 2.4

$$u_i^f = \frac{1}{k_{i,i}} \bullet \left[f_i - \left(\sum_{j=1}^n k_{i,j} \bullet u_i^o \right) + k_{i,i} \bullet u_i^o \right] \quad \forall i = 1 \dots n \quad (2.4)$$

O método de Gauss Seidel funciona de forma similar, tendo como única diferença que o valor calculado na interação é imediatamente usado nas próximas interações (ou seja, são usados sempre os valores mais atualizados e mais precisos).

Existem várias condições de convergência, contudo a mais precisa e segura para o sistema é aplicar as variáveis iteradas no sistema formulado, e somar o Residual (R), contanto que o Residual (R) esteja abaixo da precisão exigida, o programa convergirá (Eq. 2.5).

$$R = \sum_i^n \left\| \left(\sum_j^n k_{i,j} \bullet u_i \right) - f_i \right\| \quad (2.5)$$

5.3 LINGUAGEM JAVASCRIPT

O Javascript é uma linguagem da Web, com uma comunidade que está sempre anexando diferentes componentes. A implementação dessa linguagem é extremamente utilizada no desenvolvimento de aplicativos e simuladores, por ser uma linguagem de alto nível, aceita que a hierarquia do código esteja vinculada a indentação, tornando o escopo mais limpo. Diferente de outras linguagens, é compilado nativamente pelos *browsers* (como chrome, opera, firefox, edge, etc), ou seja, o programa é compilado simultaneamente a sua execução (Flanagan, 2004; Dirksen, 2015).

Essa linguagem é orientada a objetos, ou seja, cada função ou *array* declarado são identificados como objetos pelo *kernel*. A consequência disso é que *design patterns* orientados a objetos (como as classes) são descartáveis, presentes apenas como um *sintaxe sugar* para programadores que vem de outras linguagens, e estão acostumados a trabalhar nesse modelo (Flanagan, 2004).

O uso de classes no javascript desencadeia uma herança, que é componível, mas resulta em conflitos de imutabilidade, seu *kernel* possui a herança prototipal que torna as funções mais recursivas, e adaptáveis (sendo uma abordagem mais adequada).

Além de manipular elementos HTML, e estilos CSS, pode trabalhar diretamente com o *Canvas*, elemento que garante acesso a renderizações gráficas dentro do navegador. A renderização de gráficos tridimensionais está envolvida com o canvas e o WebGL (que também é nativo da linguagem). Porém, existem outras ferramentas poderosas desenvolvidas pela comunidade para esse tipo de serviço (Dirksen, 2015).

O ‘three.js’ é uma biblioteca que desperta o potencial do WebGL, permitindo realizar o mesmo trabalho, com menos linhas de código, e de forma mais intuitiva. Além de convenções pré-processadas ele permite injeção de Shaders com GLSL (uma linguagem derivada de C++), o que torna o processo de modelação mais livre (Dirksen, 2015).

6 METODOLOGIA DA PESQUISA

Foi explorado como o simulador se comporta em geometrias distintas, sua validação dividida em duas etapas: a avaliação da precisão de cálculo, e avaliação da funcionalidade do programa tanto no modo ‘automático’ quanto no modo ‘manual’. As variáveis da malha gerada pelo simulador incluem modulo de elasticidade (E) indicando a rigidez, a força da peso (P) dentro do ambiente de modelagem, e o carregamento externo.

Os métodos aqui apresentados foram escritos em JavaScript, usando a biblioteca geométrica Three.js, que possibilita - entre outras coisas - a animação do sistema de partículas.

6.1 PROCEDIMENTOS METODOLÓGICOS E MÉTODOS DE COLETA

6.1.1 Pesquisa Bibliográfica

A pesquisa está subdividida em duas etapas, a primeira explorando fontes diversas, em bases de dados (repositórios de artigos), e *websites* com temas de computação gráfica e resolução de sistemas diferenciais numericamente. Os materiais buscados consistiram em dissertações, teses e anais de encontros científicos. A segunda etapa será o delineamento das informações que mais interessarem. Dessa forma essa etapa se dividirá da seguinte maneira:

- Leitura Exploratória (verificação da contribuição que o material poderá ter na pesquisa);
- Leitura Seletiva (dos materiais lidos, serão isolados os escolhidos para uso);
- Leitura Analítica (identificada e anotadas as informações contidas nas fontes que possam contribuir melhor);
- Leitura Interpretativa (comparação e compilamento das informações retiradas de cada autor em uma ideia apenas).

A finalização dessas etapas se dá com o desenvolvimento dos apontamentos bibliográficos, com o objetivo de organização do material lido e devidamente analisado.

6.2 DESENVOLVIMENTO DO PROGRAMA

O programa funciona em dois ‘modos’: “automático” e o “manual” . No “automático” a malha é gerada parametricamente e a estrutura é representada em cinco (5) fases: Discretização, Dimensionamento, Isovalores, Fluxo e Deformação. No “manual” a malha é inserida manualmente e a estrutura é representada em três fases: Discretização, Dimensionamento, e Deformação.

A descrição do programa está organizada em cinco (5) etapas (cada uma com seus subtopicos):

- Criação (malha, vértices, conexões, células)
- Cálculo (atualização dos elementos, vetores, células)
- Isovalores (malha gradiente)
- Fluxo (criação de agentes)
- Deformação (atualização dos elementos)
- Interface (arquitetura da responsividade do programa)

6.2.1 Criação

6.2.1.1 Malha

Deve ser criado um parâmetro Global que contém todas as variáveis importantes do programa, toda a aplicação será gerada em torno dos dados desse objeto (Eq. 2.6).

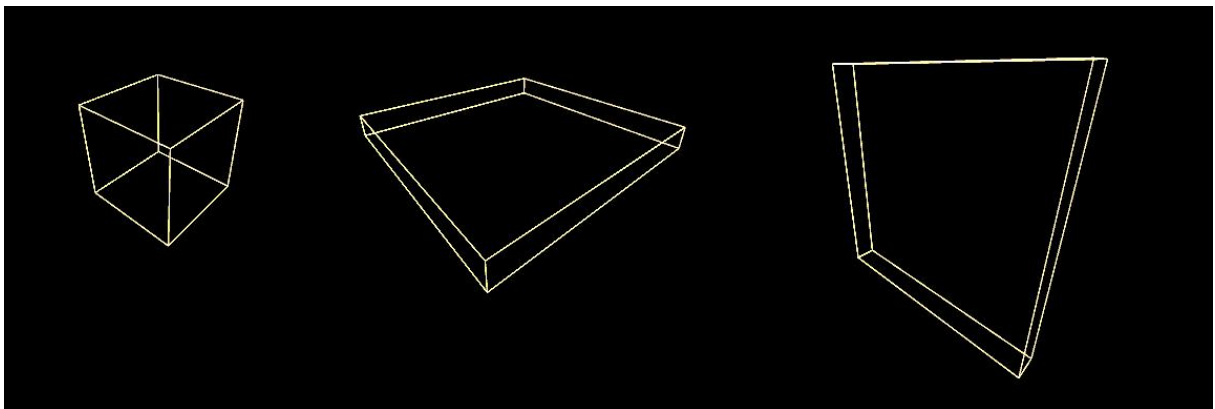
$$\text{parametros} = \{ \dots \text{vazio} \dots \} \quad (2.6)$$

6.2.1.1.1 Automático

São inseridos dois conjuntos de variáveis aos *parâmetros*, o primeiro conjunto será o que descreve as *dimensões* da malha (Eq. 2.7, Figura 16).

$$\text{dimensoes} = \{ x, y, z \} \quad (2.7)$$

Figura 16 - Parametro 'dimensões'.

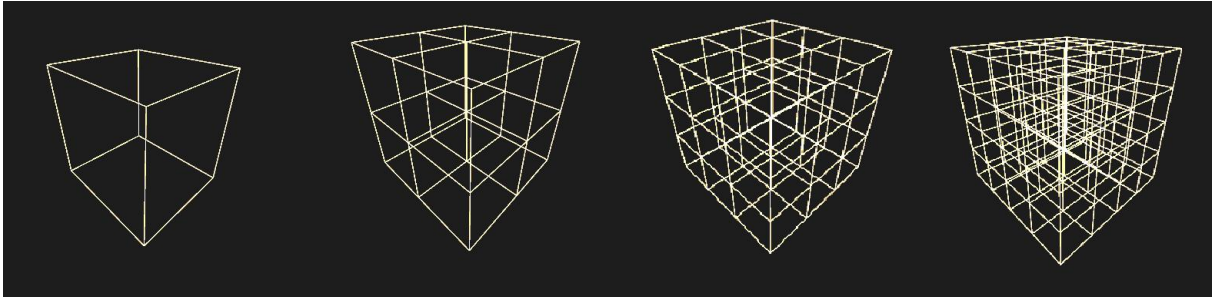


Fonte: Elaborado pelo autor, 2021.

O segundo conjunto age como *discretizador* da malha, que definirá em quantos conjuntos de *celulas* a malha será subdividida nos eixos coordenados (Eq. 2.8, Figura 17).

$$\text{discretizador} = \{ x, y, z \} \quad (2.8)$$

Figura 17 - Parametro ‘discretizador’.



Fonte: Elaborado pelo autor, 2021.

Injetados os parâmetros ‘*dimensoes*’ e ‘*discretizador*’ na função “*initGeometria*” (APÊNDICE D), é obtido o objeto *malha* que armazena fatores de dimensionamento (Eq. 2.9)

$$\text{malha} = \{ \begin{array}{l} \text{unidades: [discretizador],} \\ \text{dimensoes: [totais, discretas]} \end{array} \} \quad (2.9)$$

6.2.1.1.2 Manual

Caso esteja no modo manual, a malha não é construída parametricamente, portanto essa etapa não é executada.

6.2.1.2 Vértices

6.2.1.2.1 Automático

As coordenadas de vértice da malha são injetados na função ‘*initPontos*’ (APÊNDICE A), criando para cada vértice um objeto ‘*ponto*’. O conjunto total desses ‘*ponto*’ é armazenado por ‘*pontos*’ (Eq. 3.0).

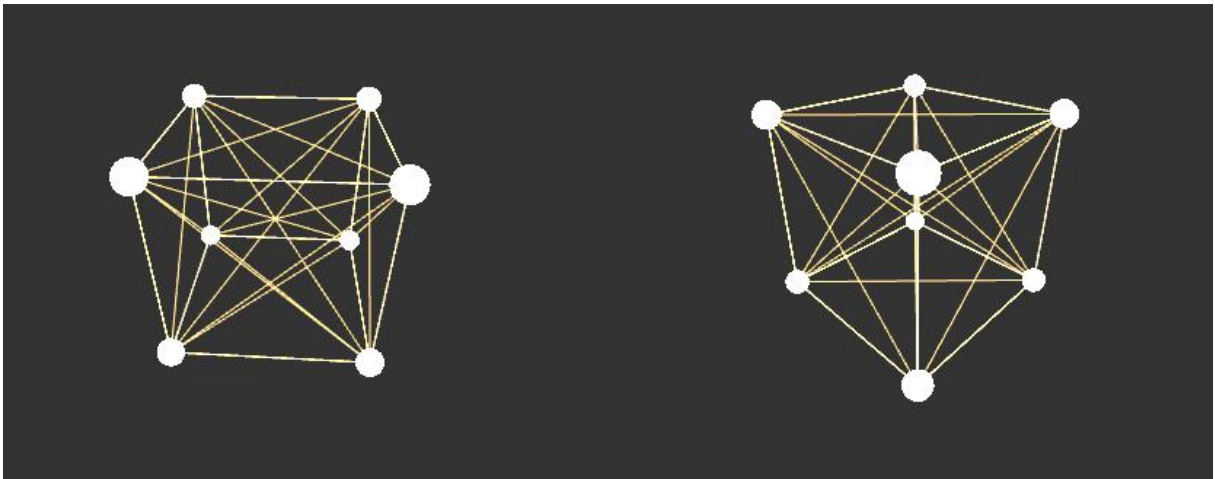
$$\begin{array}{l} \text{pontos} = \{ \text{ponto}_1, \text{ponto}_2, \text{ponto}_3, \dots \text{ponto}_n \} \\ \text{pontos} = \{ \text{ponto}_i \} \quad \forall i = 1, 2, 3 \dots n \end{array} \quad (3.0)$$

Cada objeto individual ‘*ponto_i*’ contém informações de identificação, posicionamento, conexões, cor e vetor associado. Contudo, na criação do objeto ele ainda não tem acesso a informações de conexão, cor, e vetor (então esses campos estarão vazios) (Eq. 3.1).

$$\begin{aligned}
 \text{ponto}_i = \{ \\
 & \text{identificador: } i, \\
 & \text{posicionamento: } \{ x, y, z \}, \\
 & \text{conexoes: } [\dots \text{vazio} \dots], \\
 & \text{vetor: } \dots \text{vazio} \dots, \\
 & \text{cor: } \dots \text{vazio} \dots \\
 & \} \quad \forall i = 1, 2, 3 \dots n
 \end{aligned}
 \tag{3.1}$$

Concluída a criação dos ‘*pontos*’ através da função ‘*addConexoesNosPontos*’ são adicionadas as ‘*conexoes*’ em cada ‘*ponto_i*’. Gerando células de 24 conexões por octante (Figura 18).

Figura 18 - Conexões por Octante.



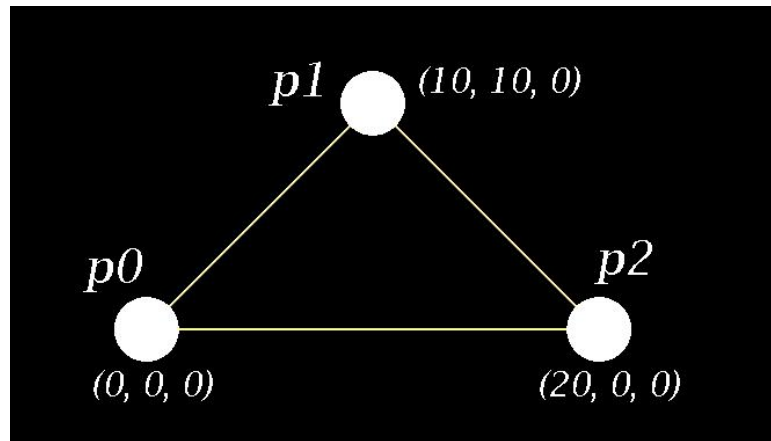
Fonte: Elaborado pelo autor, 2021.

Passando ‘*pontos*’ como argumento da função ‘*desenharPontos*’ (APÊNDICE A), os pontos são desenhados na tela nas coordenadas correspondentes.

6.2.1.2.2 Manual

Manualmente o processo é similar, a diferença principal é que não serão usados os vértices gerados de forma paramétrica para a ‘malha’. Nesse modo, serão vértices personalizados, sendo necessário injetar as conexões dos ‘*ponto_i*’ manualmente, conforme exemplo da figura 19 (Eq. 3.2).

Figura 19 - Exemplo de criação de pontos Manualmente.



Fonte: Elaborado pelo autor, 2021.

```
pontos = [  
  { id: 0, pos: {x: 0, y: 0, z: 0}, conexoes: [ p1, p2 ], cor: vazio, vetor: vazio },  
  { id: 1, pos: {x: 0, y: 10, z: 0}, conexoes: [ p0, p2 ], cor: vazio, vetor: vazio },  
  { id: 2, pos: {x: 20, y: 0, z: 0}, conexoes: [ p0, p1 ], cor: vazio, vetor: vazio },  
]
```

(3.2)

6.2.1.3 Conexões

6.2.1.3.1 Automático e Manual

O objeto ‘*pontos*’ é injetado na função ‘*initConexoes*’ (APÊNDICE B), retornando objetos de ‘*conexao*’ que contém (em cada um) informações sobre ponto de início (*pontoA*), fim (*pontoB*), e numeração da conexão (*nc*). Todos esses objetos individuais estão armazenados no objeto ‘*conexoes*’ (Eq. 3.3).

$conexoes = \{ conexao_1, conexao_2, conexao_3, \dots conexao_n \}$ (3.3)

$$\begin{aligned}
\text{conexoes} &= \{ \text{conexao}_i \} \quad \forall i = 1, 2, 3 \dots n \\
\text{conexao}_i &= \{ \\
&\quad \text{pontoA: } \{ \text{id}, \text{pos}, \text{cor} \}, \\
&\quad \text{pontoB: } \{ \text{id}, \text{pos}, \text{cor} \}, \\
&\quad \text{nc: 'id.pontoA|id.pontoB'} \\
&\} \quad \forall i = 1, 2, 3 \dots n
\end{aligned}$$

Passando ‘conexoes’ como argumento da função ‘desenharConexoes’ (APÊNDICE B), as *conexões* são desenhados na tela nas coordenadas correspondentes.

6.2.1.4 Células

6.2.1.4.1 Automático

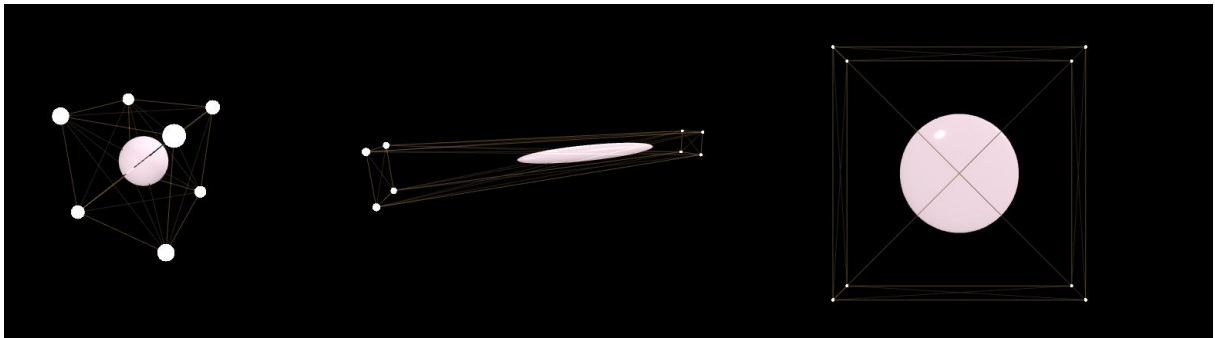
Com o objeto ‘malha’ injetado na função ‘initCelulas’(APÊNDICE C) é retornado o objeto ‘celulas’ que armazena o conjunto total de objetos ‘celula’ da malha (Eq. 3.4).

$$\begin{aligned}
\text{celulas} &= \{ \text{celula}_1, \text{celula}_2, \text{celula}_3, \dots \text{celula}_n \} \\
\text{celulas} &= \{ \text{celula}_i \} \quad \forall i = 1, 2, 3 \dots n \\
\text{celula}_i &= \{ \\
&\quad \text{identificador: } i, \\
&\quad \text{posicionamento: } \{ x, y, z \}, \\
&\quad \text{conexoes: } [\dots \text{vazio} \dots] \\
&\quad \text{vetores: } \dots \text{vazio} \dots, \\
&\quad \text{vertices: } \dots \text{vazio} \dots \\
&\} \quad \forall i = 1, 2, 3 \dots n
\end{aligned} \tag{3.4}$$

Passando diretamente ‘celulas’ como argumento da função ‘desenharCelulas’ (APÊNDICE C), as ‘celulas’ são desenhadas na tela nas coordenadas correspondentes com uma cor padronizada.

Em sequência, usando a função ‘addConexoesCelulas’ (APÊNDICE C), são adicionados, para cada ‘celula_i’, os vértices da malha que estão cercando-a, conforme figura 20.

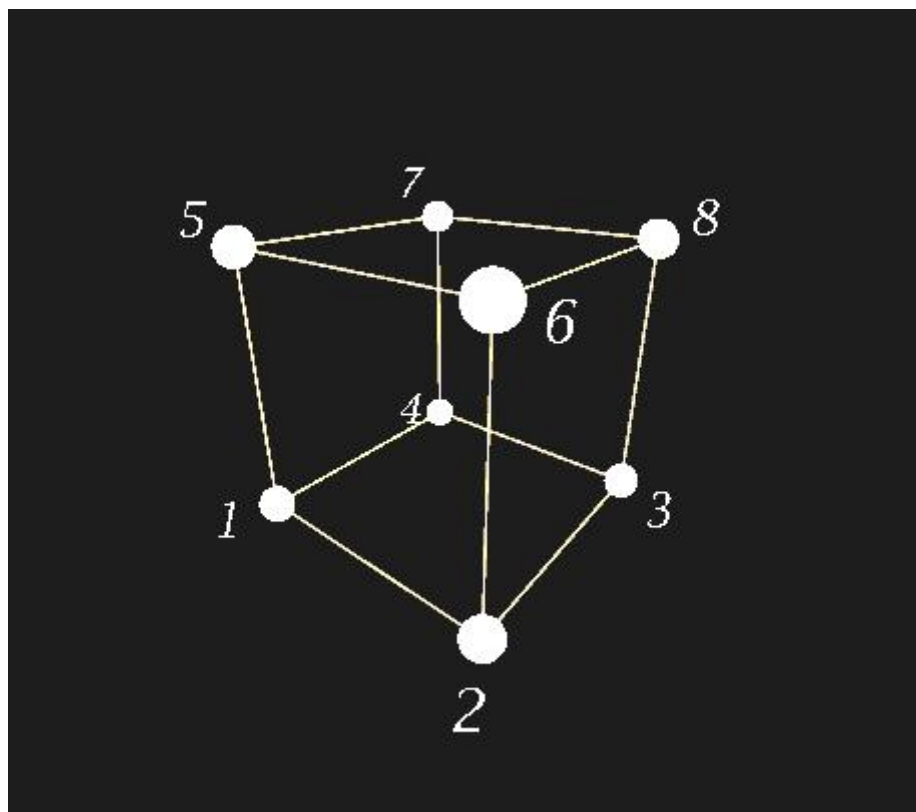
Figura 20 - Unidade ‘célula’ com os vértices.



Fonte: Elaborado pelo autor, 2021.

Os vértices precisam ser identificados (de acordo com suas posições com relação à ‘ $celula_i$ ’), essa ‘identificação é feita usando a função ‘*mapearVertices*’ (APÊNDICE C), conforme figura 21.

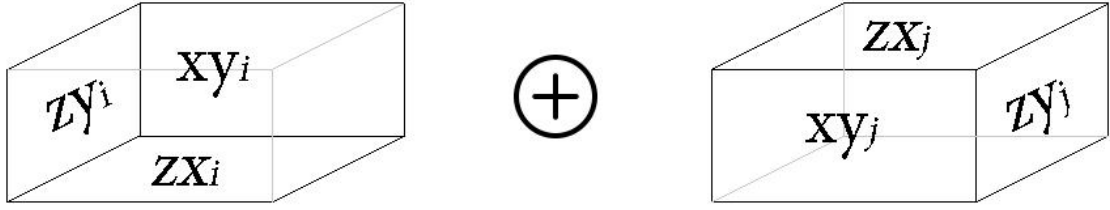
Figura 21 - Referência de Mapeamento.



Fonte: Elaborado pelo autor, 2021.

Para facilitar a comparação entre os vértices, a geometria de referência foi subdividida seis (6) faces, conforme figura 22.

Figura 22 - Referência de Faces.



Fonte: Elaborado pelo autor, 2021.

Com isso, a identificação dos vértices é resumida há verificação de com quais faces cada vértice está tendo contato (Eq. 3.5).

$$\begin{aligned}
 1 &= \{ 'zy_i', 'xy_j', 'zx_i' \} \\
 2 &= \{ 'zy_j', 'xy_j', 'zx_i' \} \\
 3 &= \{ 'zy_j', 'xy_i', 'zx_i' \} \\
 4 &= \{ 'zy_i', 'xy_i', 'zx_i' \} \\
 5 &= \{ 'zy_i', 'xy_j', 'zx_j' \} \\
 6 &= \{ 'zy_j', 'xy_j', 'zx_j' \} \\
 7 &= \{ 'zy_j', 'xy_i', 'zx_j' \} \\
 8 &= \{ 'zy_i', 'xy_i', 'zx_j' \}
 \end{aligned} \tag{3.5}$$

6.2.1.4.2 Manual

No modo ‘*manual*’, a criação e representação visual das células é desabilitada.

6.2.2 Cálculo

6.2.2.1 Cálculo Matrizes

6.2.2.1.1 Automático

São injetados os objetos ‘*pontos*’ e ‘*conexoes*’ na função ‘*initMatrizes*’ (APÊNDICE E). O número de graus de liberdade da malha será equivalente ao número de vértices (*pontos*) multiplicado por três (3), pois cada vértice pode ser deslocado em três coordenadas (x, y, z).

Para cada ‘conexao_i’ será construído um ‘*elemento_i*’ contendo informações de identificação, posição, rigidez, cosxξ, cosyξ, e coszξ (Eq. 3.6).

$$\begin{aligned} \text{elementos} &= \{ \text{elemento}_1, \text{elemento}_2, \text{elemento}_3, \dots, \text{elemento}_n \} \\ \text{elementos} &= \{ \text{elemento}_i \} \quad \forall i = 1, 2, 3 \dots n \end{aligned}$$

$$\begin{aligned} \text{elemento}_i &= \{ \\ &\quad \text{id: } i, \\ &\quad \text{a: } \{ \text{id, pos: } \{ x, y, z \} \}, \\ &\quad \text{b: } \{ \text{id, pos: } \{ x, y, z \} \} \\ &\quad \text{k: rigidez,} \\ &\quad \text{l: cosx}\xi, \\ &\quad \text{m: cosy}\xi, \\ &\quad \text{n: cosz}\xi \\ &\quad \} \end{aligned} \quad (3.6)$$

Cada ‘*elemento_i*’ é passado como argumento na função ‘*initMatrizesIndividuais*’ (APÊNDICE E), criando para cada uma matriz de rigidez $[k]_{x,y,z}^{6 \times 6}$ (conforme Eq. 1.9). Cada uma dessas matrizes é compilada na Matriz Global de Rigidez $[K]_{x,y,z}^{n \times n}$ pela função ‘*initMatrizGlobal*’ (APÊNDICE E).

Foram adotados apenas carregamentos de externos de pesos próprio de cada nó da malha (o *guizmo* do ambiente virtual tem ‘y’ como eixo vertical), enquanto que nos demais graus de liberdade o carregamento será nulo, essa formulação é desenvolvida pela função ‘*initVetorPeso*’ (APÊNDICE E), retornando a matriz global $[F]_{x,y,z}^T$ (Eq. 3.7).

$$\begin{aligned} [F]_{x,y,z}^T &= \{ F_1, F_2, F_3, \dots, F_n \} \\ [F]_{x,y,z}^T &= \{ F_i \} \quad \forall i = 1, 2, 3 \dots n \end{aligned} \quad (3.7)$$

$$F_i = \{ x: 0, y: \text{Peso}, z: 0 \}$$

Existe uma variedade de maneiras de implementar as condições de contorno, e como o interesse desse programa está em disponibilizar conjuntos bem alternativos de ensaio, as condições de contorno serão implementadas de maneira aleatória, sendo diferentes para cada

experimentação. Por a malha ser tridimensional, são necessários pelo menos três pontos de apoio para que a solução atingia convergência. Portanto, são escolhidos três vértices aleatórios dentro do conjunto de '*pontos_i*', e esses tem suas correspondentes de deformação em $[U]_{x,y,z}^T$ anuladas, essa formulação é desenvolvida pela função '*addCondiçõesDeContorno*' (APÊNDICE E), adicionando a eles uma propriedade de identificador (Eq. 3.8).

$$\begin{aligned}
 [U]_{x,y,z}^T &= \{ U_1, U_2, U_3, \dots U_n \} \\
 [U]_{x,y,z}^T &= \{ U_i \} \quad \forall i = 1, 2, 3 \dots n \\
 U_{ix} &= \{ \text{valor: 0, identificado: 'true'} \} \\
 U_{iy} &= \{ \text{valor: 0, identificado: 'true'} \} \\
 U_{iz} &= \{ \text{valor: 0, identificado: 'true'} \}
 \end{aligned} \tag{3.8}$$

Finalmente é feito o cálculo das deformações, a verificação de convergência está baseada no método de Gauss-Seidel, a implementação segue as Equações 2.4 e 2.5, essa formulação é gerenciada pela função '*verificarConvergência*' (APÊNDICE E).

Enquanto o *residual* estiver acima da *tolerância* as iterações prosseguirão. Quando finalizadas, o vetor de deformação dimensionado $[U]_{x,y,z}^T$ é compilado pela função '*modulosDosVetores*' retornando o objeto '*matrizes*', esse objeto contém os valores absolutos, normalizados de cada vetor, além da informação da máxima e mínima deformação do conjunto (Eq. 3.9).

$$\begin{aligned}
 matrizes &= \{ \text{vetores: } [\{ \text{vetor}_i \}], \text{min, max} \} \quad \forall i = 1, 2, 3 \dots n \\
 \text{vetor}_i &: \{ u, v, w, \text{abs, normalizado: } \{ nx, ny, nz \} \}
 \end{aligned} \tag{3.9}$$

6.2.2.1.2 Manual

O processo global é similar ao '*automático*', a única diferença é que no modo '*manual*' as condições de contorno $[U]_{x,y,z}^T$ e esforços externos $[F]_{x,y,z}^T$ são todas implementadas manualmente. Portanto, as funções '*initVetorPeso*' e '*addCondiçõesDeContorno*' (APÊNDICE E) não são executadas.

6.2.2.2 Atualização dos elementos

6.2.2.2.1 Automático e Manual

Passando o objeto '*matrizes*' como argumento da função '*adicionarVetoresNosPontos*' (APÊNDICE A), os '*pontos*' recebem seus respectivos vetores. Para o cálculo da cor cada '*ponto_i*' é passado como argumento da função '*adicionarCoresNosPontos*' (APÊNDICE A). Para atualizar as '*conexoes*', só é necessário re-executar a função '*initConexoes*', que o objeto será rebobinado com os vetores e cores correspondentes.

6.2.2.3 Vetores

6.2.2.3.1 Automático e Manual

Passando '*pontos*' e '*matrizes*' como argumento da função '*desenharVetores*' (APÊNDICE A), os '*vetores*' são desenhados na tela nas coordenadas correspondentes.

6.2.2.4 Células

6.2.2.4.1 Automático

Passando diretamente '*celulas*' como argumento da função '*refinamentoCelulas*' (APÊNDICE C), as '*celulas*' são desenhadas na tela nas coordenadas $\xi: 0.5$, $\eta: 0.5$, $\zeta: 0.5$ com a cor do vetor correspondente.

6.2.2.4.2 Manual

No modo '*manual*', a criação e representação visual das células é desabilitada.

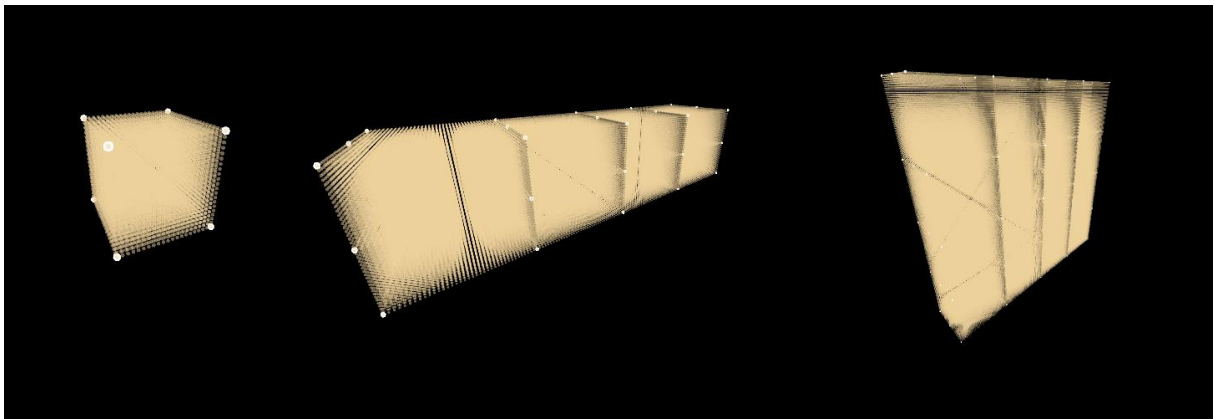
6.2.3 Isovalores

6.2.3.1 Malha gradiente

6.2.3.1.1 Automático

É adicionado no objeto '*parametros*' o objeto '*refinamento*', com esse é determinado o refinamento e a disposição na malha dos isovalores (Figura 23).

Figura 23 - Refinamento isovalores.

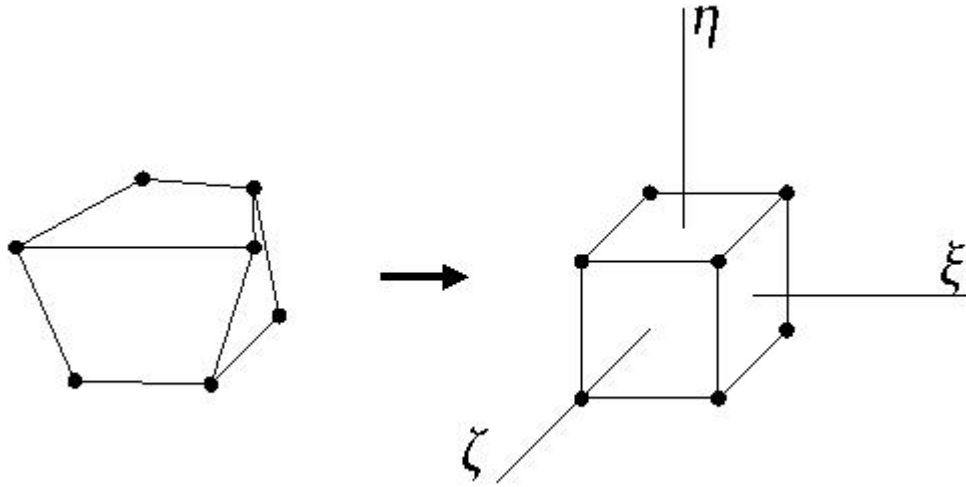


Fonte: Elaborado pelo autor, 2021.

Esses '*isovalores*' não passam de pontos intermediários, portanto, as informações de deformações associadas a cada um desses é resultado de uma interpolação com os valores da malha. A ideia é interpolar entre os vetores anexados ao octante da '*celula_i*', mas esse processo se torna particularmente desafiador quando a geometria não é simétrica.

Para que a abordagem geométrica do programa seja mais ampla, (e contornar o problema de trabalhar diretamente com geometrias sem simetria) essa interpolação é feita em um espaço de coordenada uniforme, para o qual qualquer geometria poderá ser convertida através de transformação de coordenadas, figura 24,

Figura 24 - Transformação de Coordenadas.



Fonte: Elaborado pelo autor, 2021.

essa conversão pode ser usado simples de funções lineares (Eq. 4.0),

$$\begin{aligned} N_i(t) &= \frac{(1 - t)}{2} \\ N_f(t) &= \frac{(1 + t)}{2} \end{aligned} \quad (4.0)$$

Contudo essas funções agem em elementos lineares, uma maneira de convertê-las para a geometria tridimensional é usando propriedades tensoriais, fazendo o produto para as extremidades conectadas em cada nó é retornado um conjunto de expressões muito convenientes por obedecerem as propriedades do delta de Kroniker (Eq. 4.1).

$$\begin{aligned} N_1(\xi, \eta, \zeta) &= N_f(\xi) \cdot N_i(\eta) \cdot N_i(\zeta) = \frac{1}{8} (1 + \xi) \cdot (1 - \eta) \cdot (1 - \zeta) \\ N_2(\xi, \eta, \zeta) &= N_f(\xi) \cdot N_f(\eta) \cdot N_i(\zeta) = \frac{1}{8} (1 + \xi) \cdot (1 + \eta) \cdot (1 - \zeta) \\ N_3(\xi, \eta, \zeta) &= N_i(\xi) \cdot N_f(\eta) \cdot N_i(\zeta) = \frac{1}{8} (1 - \xi) \cdot (1 + \eta) \cdot (1 - \zeta) \\ N_4(\xi, \eta, \zeta) &= N_i(\xi) \cdot N_i(\eta) \cdot N_i(\zeta) = \frac{1}{8} (1 - \xi) \cdot (1 - \eta) \cdot (1 - \zeta) \end{aligned} \quad (4.1)$$

$$N_5(\xi, \eta, \zeta) = N_f(\xi) \cdot N_i(\eta) \cdot N_f(\zeta) = \frac{1}{8} (1 + \xi) \cdot (1 - \eta) \cdot (1 + \zeta)$$

$$N_6(\xi, \eta, \zeta) = N_f(\xi) \cdot N_f(\eta) \cdot N_f(\zeta) = \frac{1}{8} (1 + \xi) \cdot (1 + \eta) \cdot (1 + \zeta)$$

$$N_7(\xi, \eta, \zeta) = N_i(\xi) \cdot N_f(\eta) \cdot N_f(\zeta) = \frac{1}{8} (1 - \xi) \cdot (1 + \eta) \cdot (1 + \zeta)$$

$$N_8(\xi, \eta, \zeta) = N_i(\xi) \cdot N_i(\eta) \cdot N_f(\zeta) = \frac{1}{8} (1 - \xi) \cdot (1 - \eta) \cdot (1 + \zeta)$$

Tanto a implementação dessa conversão de coordenadas quanto da representação gráfica na tela da aplicação do gradiente de deformação é realizada pela função ‘refinamentoCelulas’ (APÊNDICE C).

6.2.3.1.2 Manual

No modo ‘*manual*’, a criação e representação visual dos Isovalores é desabilitada.

6.2.4 Fluxo

6.2.4.1 Criação de agentes

6.2.4.1.1 Automático

O fluxo é representado por um conjunto de ‘*agentes*’ que trabalham de maneira individual, são inseridos na geometria e percorrem a direção determinada pelo vetor do vértice da malha mais próximo do ‘*agente_i*’. Essa iteração é realizada até que todos os agentes finalizem seus caminhos, com os caminhos processados, são salvos no objeto ‘*agentes*’ (Eq. 4.2).

$$\text{agentes} = \{ \text{agente}_1, \text{agente}_2, \text{agente}_3, \dots, \text{agente}_{U_n} \}$$

$$\text{agentes} = \{ \text{agente}_i \} \quad \forall i = 1, 2, 3 \dots n$$

$$\begin{aligned} \text{agente}_i = \{ & \\ \text{pos: } \{ x, y, z \}, & \\ \text{cor: } \{ r, g, b \}, & \\ \text{vetor: } \{ nx, ny, nz \}, & \end{aligned} \quad (4.2)$$

$$\text{path: ['posições percorridas pelo agente']}$$

$$\} \quad \forall i = 1, 2, 3 \dots n$$

A função *'initAgentes'* (APÊNDICE F) realiza esse processamento, instanciando todas essas propriedade para cada *'agente_i'*. Com a função *'desenharAgente'* (APÊNDICE F) são desenhadas linhas através do caminho descrito pela propriedade *'path'* de cada *'agente_i'* (herdando os vetores e cores).

6.2.4.1.2 Manual

No modo *'manual'*, a criação e representação visual do Fluxo é desabilitada.

6.2.5 Deformação

6.2.5.1 Atualização dos elementos

6.2.5.1.1 Automático e Manual

Através da função *'deformarPontos'* (APÊNDICE A), os *'pontos'* da malha são movimentados conforme os vetores associados a eles. Para atualizar as *'conexoes'* é re-executada a função *'initConexoes'* (APÊNDICE B), reprocessando os elementos *'conexao_i'* com as posições dos *'pontos'* deformados.

6.2.6 Interface

6.2.6.1 Arquitetura do programa

O programa principal é compilado por cinco (5) funções (que executam ou deixam de executar as funções anteriormente apresentadas, em ordem que permita a representação gráfica desejada em cada fase do programa):

- ajustarMalha;
- calcularMalha;
- desenharMalhaCalculada;
- desenharFluxo;

- desenharMalhaDeformada.

O papel de cada uma dessas funções é resetar e reconfigurar as variáveis para cada fase de apresentação do programa (que é estruturado em cinco (5) fases: Discretização, Dimensionamento, Isovalores, Fluxo e Deformação). A implementação dessas funções está no main.js (APÊNDICE K).

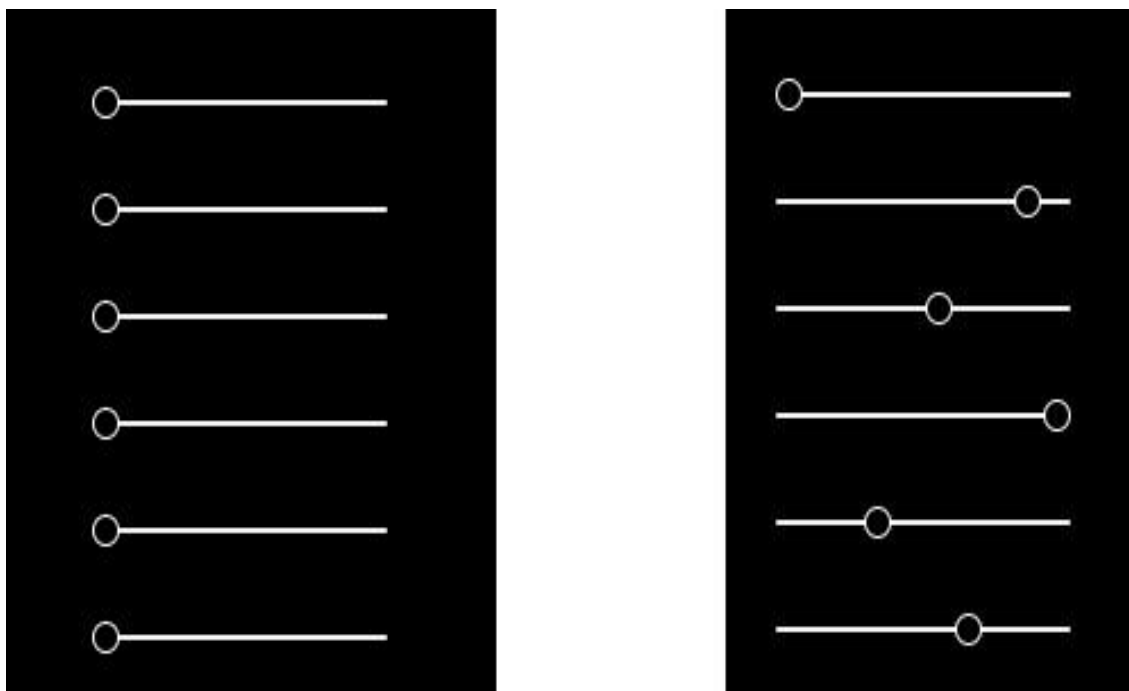
6.2.6.2 Menus

A comunicação do usuário com o programa acontece através de seis (06) menus: slider, índice, controle de isovalores, e resetar programa / avançar etapa.

6.2.6.2.1 Slider

O slider permite que o usuário determine dimensões e discretização da malha, sua implementação é feita pela função *'initSliders'* (APÊNDICE L), os três sliders superiores trabalham com o dimensionamento enquanto os três inferiores com a discretização da malha, Figura 25.

Figura 25 - Slider em duas configurações distintas.

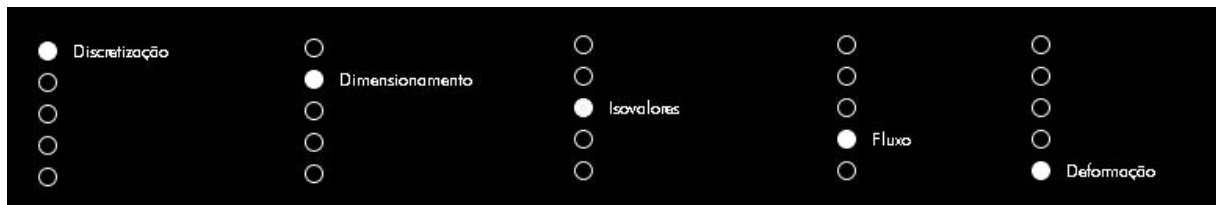


Fonte: Elaborado pelo autor, 2021.

6.2.6.2.2 Índice

O Índice deixa explícito para o usuário em qual fase o programa está sendo executado, sua implementação é feita pela função *‘navegarPontos’* (APÊNDICE L), a fase que está em execução tem sua marcação preenchida e o texto destacado, enquanto as outras não são preenchidas e o texto é escondido, Figura 26.

Figura 26 - Índice em suas configurações distintas.

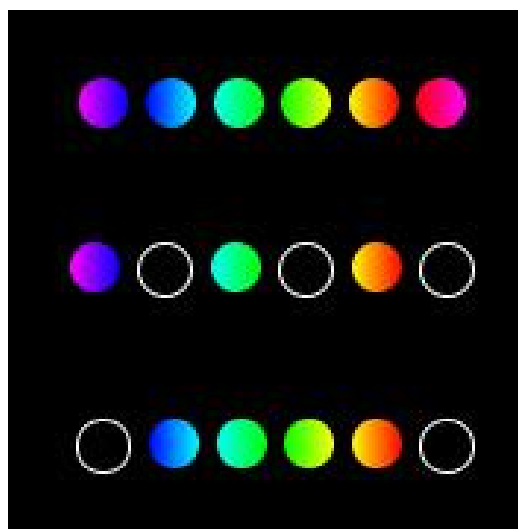


Fonte: Elaborado pelo autor, 2021.

6.2.6.2.3 Controle de Isovalores

O controle de Isovalores permite para o usuário esconder ou mostrar a malha de Gradiente e Fluxo associadas com as cores do Menu, sua implementação é feita pela função *‘initGradienteInputs’* (APÊNDICE L), os intervalos de tensão visíveis são preenchidos pela cor correspondente, ao clicar o usuário pode habilitar ou desabilitar essa visualização, Figura 27.

Figura 27 - Controle de Isovalores em configurações distintas.



Fonte: Elaborado pelo autor, 2021.

6.2.6.2.4 Resetar programa / Avançar etapa

Os botões que controlam o avanço do programa são os de resetar, e o de avançar, sua implementação é feita pelas funções *'initBefore'* e *'initNext'* (APÊNDICE L), ao clicar movimenta a execução do programa, Figura 28.

Figura 28 - Controle da direita “resetar”, controle da esquerda “avançar”.

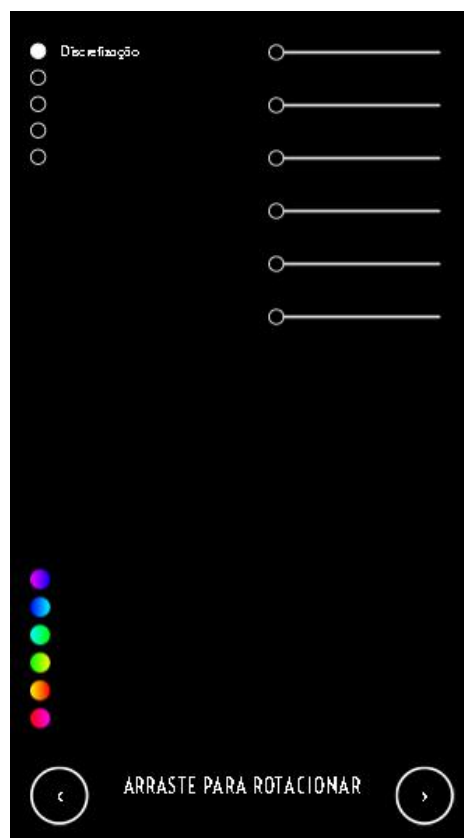


Fonte: Elaborado pelo autor, 2021.

6.2.6.3 Interface Completa

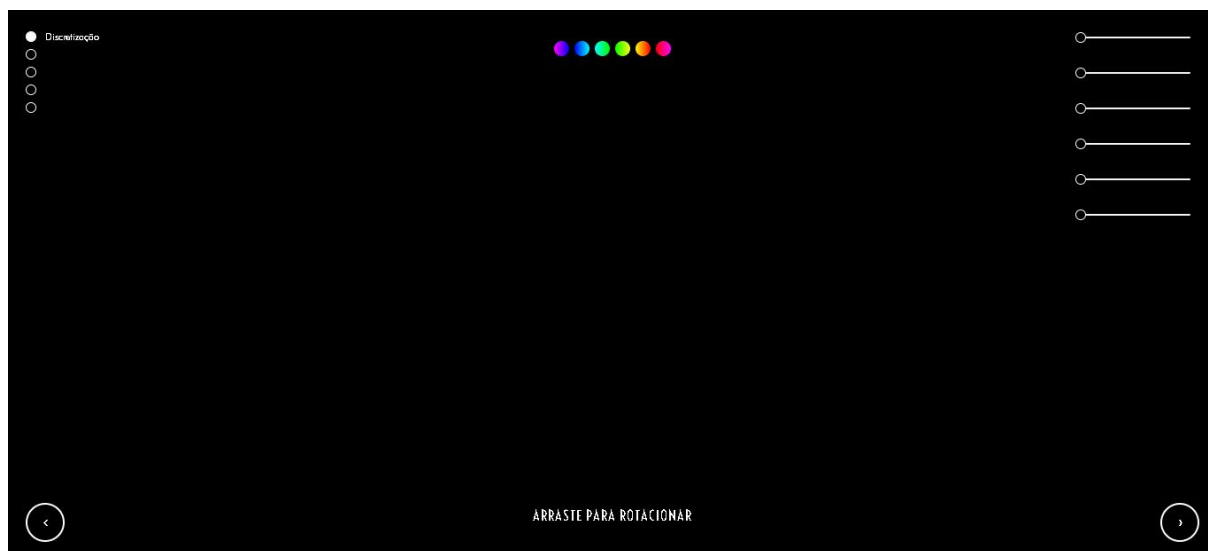
A responsividade foi ajustada para permitir que o programa seja executado em diferentes dispositivos, conforme Figuras 29 e 30.

Figura 29 - Interface Dispositivo Móvel.



Fonte: Elaborado pelo autor, 2021.

Figura 30 - Interface Monitor.



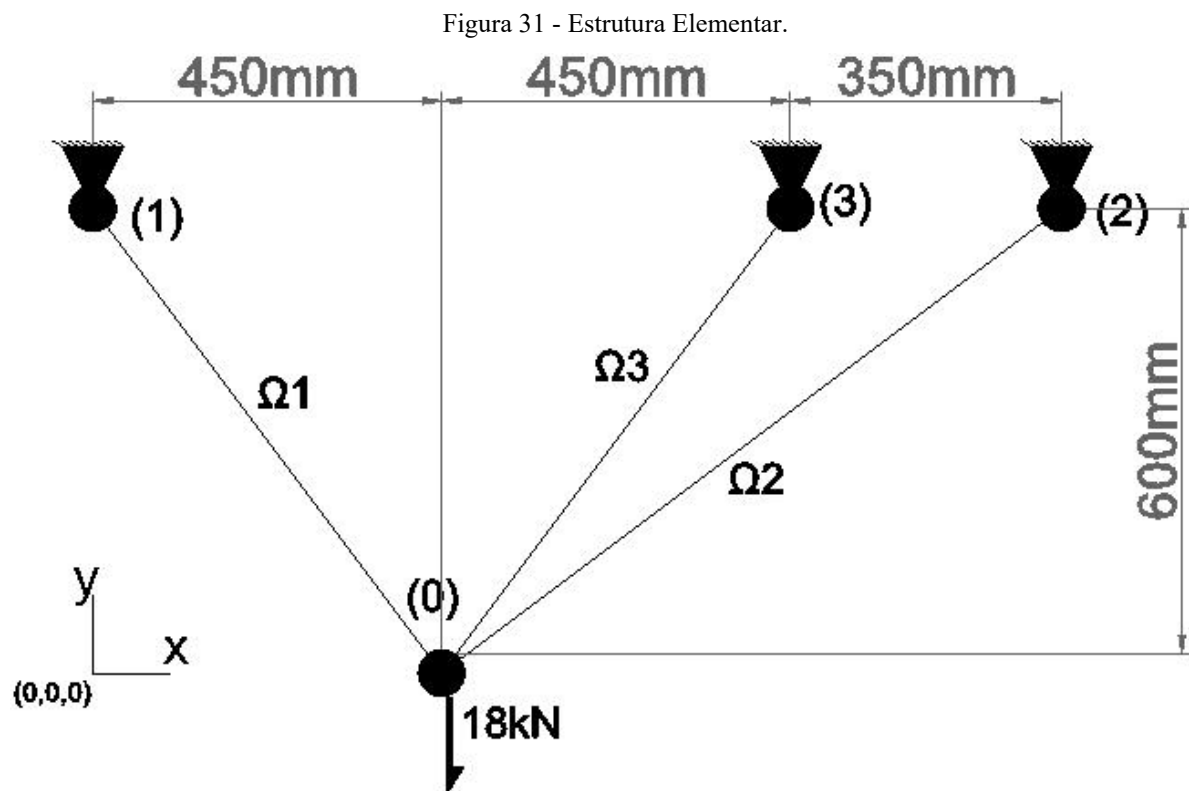
Fonte: Elaborado pelo autor, 2021.

7 RESULTADOS E DISCUSSÕES

O tópico de resultados e discussões será subdividido em duas etapas, a primeira é a avaliação da precisão de cálculo do programa, a segunda será a avaliação da funcionalidade do programa tanto no modo '*automático*' quanto no modo '*manual*'.

7.1 PRECISÃO DE CÁLCULO

A verificação da precisão será feita através da comparação dos resultados do dimensionamento de uma estrutura elementar feita *manualmente* com os resultados do *dimensionamento* do programa, a estrutura será a da figura 31.



Fonte: Elaborado pelo autor, 2021.

Para cálculo do coeficiente de rigidez foram adotados as mesmas áreas de sessão transversal e modulo de elasticidade para todas as conexões (Eq. 4.3).

$$\begin{aligned} A &= 250 \\ E &= 200 \cdot 10^3 \end{aligned} \quad (4.3)$$

7.1.1 Manual

O dimensionamento será desenvolvimento manual será realizado em três etapas, criação de matrizes de rigidez individuais (conforme eq. 4.4), e montagem da matriz global, adição das condições de contorno de Dirchlet e Neumann, e resolução do sistema linear.

$$\begin{aligned} l_o &= \sqrt{(\vec{b}_{ox} - \vec{a}_{ox})^2 + (\vec{b}_{oy} - \vec{a}_{oy})^2 + (\vec{b}_{oz} - \vec{a}_{oz})^2} \\ l &= \cos x\xi = \frac{\vec{b}_{ox} - \vec{a}_{ox}}{l_o} \\ m &= \cos y\xi = \frac{\vec{b}_{oy} - \vec{a}_{oy}}{l_o} \\ n &= \cos z\xi = \frac{\vec{b}_{oz} - \vec{a}_{oz}}{l_o} \\ k_s &= \frac{A \cdot E}{l_o} \\ [k]_{x,y,z} &= k_s \cdot \begin{bmatrix} ll & lm & ln & -ll & -lm & -ln \\ ml & mm & mn & -ml & -mm & -mn \\ nl & nm & nn & -nl & -nm & -nn \\ -ll & -lm & -ln & ll & lm & ln \\ -ml & -mm & -mn & ml & mm & mn \\ -nl & -nm & -nn & nl & nm & nn \end{bmatrix}_{x,y,z}^{6 \times 6} \end{aligned} \quad (4.4)$$

7.1.1.1 Criação das Matrizes de Rigidez Individuais

7.1.1.1.1 Elemento um (01)

$$\begin{aligned} l_o^{\Omega 1} &= \sqrt{(0 - 450)^2 + (600 - 0)^2 + (0 - 0)^2} = 750 \text{ mm} \\ l^{\Omega 1} &= \frac{0 - 450}{750} = -0.6 \\ m^{\Omega 1} &= \frac{600 - 0}{750} = 0.8 \\ n^{\Omega 1} &= \frac{0 - 0}{750} = 0.0 \end{aligned} \quad (4.5)$$

$$k_s^{\Omega 1} = \frac{250 \cdot 200 \cdot 10^3}{750} = 0.67 \cdot 10^5$$

$$[k]_{x,y,z}^{\Omega 1} = 0.67 \cdot 10^5 \cdot \begin{bmatrix} 0.36 & -0.48 & 0 & -0.36 & 0.48 & 0 \\ -0.48 & 0.64 & 0 & 0.48 & -0.64 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.36 & 0.48 & 0 & 0.36 & -0.48 & 0 \\ 0.48 & -0.64 & 0 & -0.48 & 0.64 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{x,y,z}^{6 \times 6}$$

$$\Leftrightarrow [k]_{x,y,z}^{\Omega 1} = 10^5 \cdot \begin{bmatrix} 0.24 & -0.32 & 0 & -0.24 & 0.32 & 0 \\ -0.32 & 0.4267 & 0 & 0.32 & -0.4267 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.24 & 0.32 & 0 & 0.24 & -0.32 & 0 \\ 0.32 & -0.4267 & 0 & -0.32 & 0.4267 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{x,y,z}^{6 \times 6}$$

7.1.1.1.2 Elemento dois (02)

$$l_o^{\Omega 2} = \sqrt{(1250 - 450)^2 + (600 - 0)^2 + (0 - 0)^2} = 1000 \text{ mm}$$

$$l^{\Omega 2} = \frac{1250 - 450}{1000} = 0.8$$

$$m^{\Omega 2} = \frac{600 - 0}{1000} = 0.6$$

$$n^{\Omega 2} = \frac{0 - 0}{1000} = 0.0$$

$$k_s^{\Omega 2} = \frac{250 \cdot 200 \cdot 10^3}{1000} = 0.5 \cdot 10^5$$

$$[k]_{x,y,z}^{\Omega 2} = 0.5 \cdot 10^5 \cdot \begin{bmatrix} 0.64 & 0.48 & 0 & -0.64 & -0.48 & 0 \\ 0.48 & 0.36 & 0 & -0.48 & -0.36 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.64 & -0.48 & 0 & 0.64 & 0.48 & 0 \\ -0.48 & -0.36 & 0 & 0.48 & 0.36 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{x,y,z}^{6 \times 6} \quad (4.6)$$

$$\Leftrightarrow [k]_{x,y,z}^{\Omega 2} = 10^5 \cdot \begin{bmatrix} 0.32 & 0.24 & 0 & -0.32 & -0.24 & 0 \\ 0.24 & 0.18 & 0 & -0.24 & -0.18 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.32 & -0.24 & 0 & 0.32 & 0.24 & 0 \\ -0.24 & -0.18 & 0 & 0.24 & 0.18 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{x,y,z}^{6 \times 6}$$

7.1.1.1.3 Elemento um (03)

$$l_o^{\Omega 3} = \sqrt{(900 - 450)^2 + (600 - 0)^2 + (0 - 0)^2} = 750 \text{ mm}$$

$$l^{\Omega 3} = \frac{900 - 450}{750} = 0.6$$

$$m^{\Omega 3} = \frac{600 - 0}{750} = 0.8$$

$$n^{\Omega 3} = \frac{0 - 0}{750} = 0.0$$

$$k_s^{\Omega 3} = \frac{250 \cdot 200 \cdot 10^3}{750} = 0.67 \cdot 10^5$$

$$[k]_{x,y,z}^{\Omega 3} = 0.67 \cdot 10^5 \cdot \begin{bmatrix} 0.36 & 0.48 & 0 & -0.36 & -0.48 & 0 \\ 0.48 & 0.64 & 0 & -0.48 & -0.64 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.36 & -0.48 & 0 & 0.36 & 0.48 & 0 \\ -0.48 & -0.64 & 0 & 0.48 & 0.64 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{x,y,z}^{6 \times 6} \quad (4.7)$$

$$\Leftrightarrow [k]_{x,y,z}^{\Omega 3} = 10^5 \cdot \begin{bmatrix} 0.24 & 0.32 & 0 & -0.24 & -0.32 & 0 \\ 0.32 & 0.4267 & 0 & -0.32 & -0.4267 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.24 & -0.32 & 0 & 0.24 & 0.32 & 0 \\ -0.32 & -0.4267 & 0 & 0.32 & 0.4267 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{x,y,z}^{6 \times 6}$$

7.1.1.2 Montagem da Matriz de Rigidez Global

$$[K]_{x,y,z}^{12 \times 12} =$$

$$10^5 \cdot \begin{bmatrix} .8 & .24 & 0 & -.24 & .32 & 0 & -.32 & -.24 & 0 & -.24 & -.32 & 0 \\ .24 & 1.04 & 0 & .32 & -.427 & 0 & -.24 & -.18 & 0 & -.32 & -.427 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -.24 & .32 & 0 & .24 & -.32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ .32 & -.427 & 0 & -.32 & .427 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -.32 & -.24 & 0 & 0 & 0 & 0 & .32 & .24 & 0 & 0 & 0 & 0 \\ -.24 & -.18 & 0 & 0 & 0 & 0 & .24 & .18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -.24 & -.32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .24 & .32 & 0 \\ -.32 & -.427 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .32 & .427 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{x,y,z}^{12 \times 12} \quad (4.8)$$

7.1.1.3 Adição das Condições de Contorno

7.1.1.3.1 Neumann

As condições de contorno de Neumann, (para esse problema), estão envolvidas com o vetor de esforços Globais $[F]_{x,y,z}^T$. Como apenas o nó zero (0) está sobre ação de carregamento, os demais nós receberam valores nulos (Eq. 4.9) .

$$[F]_{x,y,z}^T = \begin{bmatrix} \vec{f}_x^0 \\ \vec{f}_y^0 \\ \vec{f}_z^0 \\ \vec{f}_x^1 \\ \vec{f}_y^1 \\ \vec{f}_z^1 \\ \vec{f}_x^2 \\ \vec{f}_y^2 \\ \vec{f}_z^2 \\ \vec{f}_x^3 \\ \vec{f}_y^3 \\ \vec{f}_z^3 \end{bmatrix}_{x,y,z}^T = \begin{bmatrix} 0 \\ 18kN \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{x,y,z}^T \quad (4.9)$$

7.1.1.3.2 Dirichlet

As condições de contorno de Dirichlet, (para esse problema), estão envolvidas com o vetor de deformações Globais $[U]_{x,y,z}^T$. Como os nós 1, 2, e 3 estão apoiados, receberão valor de deformação nulo (Eq. 5.0) .

$$[U]_{x,y,z}^T = \begin{bmatrix} \bar{u}_x^0 \\ \bar{u}_y^0 \\ \bar{u}_z^0 \\ \bar{u}_x^1 \\ \bar{u}_y^1 \\ \bar{u}_z^1 \\ \bar{u}_x^2 \\ \bar{u}_y^2 \\ \bar{u}_z^2 \\ \bar{u}_x^3 \\ \bar{u}_y^3 \\ \bar{u}_z^3 \end{bmatrix}_{x,y,z} = \begin{bmatrix} \bar{u}_x^0 \\ \bar{u}_y^0 \\ \bar{u}_z^0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{x,y,z}^T \quad (5.0)$$

Resolvendo o sistema linear que integra as equações 4.8, 4.9, e 5.0, são dimensionados os valores de deformação (Eq. 5.1).

$$\begin{aligned} \bar{u}_x^0 &= 0.056 \text{ mm} \\ \bar{u}_y^0 &= -0.187 \text{ mm} \\ \bar{u}_z^0 &= 0.0 \text{ mm} \end{aligned} \quad (5.1)$$

7.1.2 Dimensionamento pelo Programa

O primeiro passo é injetar a geometria manualmente, portanto dentro da função ‘*atualizarMalha*’ (APÊNDICE K) o objeto ‘*pontos*’ deve ser preenchido conforme a equação 5.2.

$$\text{pontos} = [\begin{aligned} &\{ \text{identificador: 0, posicionamento: \{ x: 45, y: 0, z: 0 \}, conexoes: \{ identi} \\ &\{ \text{identificador: 1, posicionamento: \{ x: 0, y: 60, z: 0 \}, conexoes: \{ identi} \\ &\{ \text{identificador: 2, posicionamento: \{ x: 125, y: 60, z: 0 \}, conexoes: \{ ide} \\ &\{ \text{identificador: 3, posicionamento: \{ x: 90, y: 60, z: 0 \}, conexoes: \{ iden} \end{aligned}] \quad (5.2)$$

Os objetos de condição de contorno ‘*u*’ e ‘*f*’, junto do parâmetro de rigidez ‘*ks*’, devem ser criados, e passados como argumento da função ‘*initMatrizes*’ (APÊNDICE J), conforme equação 5.3.

```

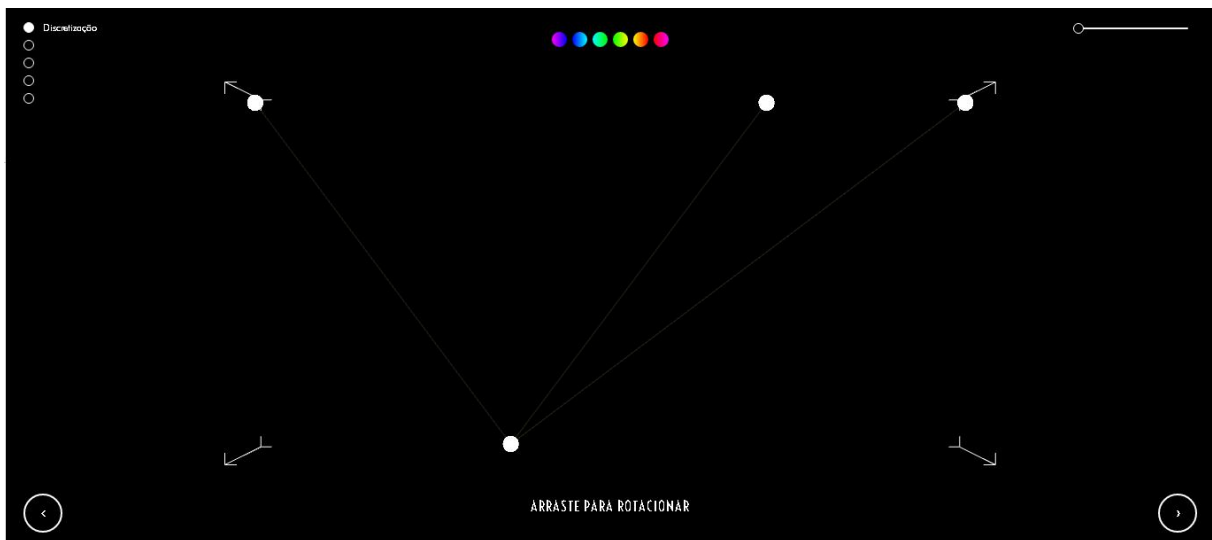
f = [
{ identificador: 0, x: 0, y: -1800000, z: 0 },
{ identificador: 1, x: 0, y: 0, z: 0 },
{ identificador: 2, x: 0, y: 0, z: 0 },
{ identificador: 3, x: 0, y: 0, z: 0 }
]
u = [
{ identificador: 0, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
{ identificador: 1, u: 0, v: 0, w: 0 },
{ identificador: 2, u: 0, v: 0, w: 0 },
{ identificador: 3, u: 0, v: 0, w: 0 }
]
parametros.rigidez = { A: 250, E: 200 * (10000) }

```

(5.3)

Ao mudar o parâmetro modo para ‘*manual*’, será exibida a geometria na tela, Figura 32.

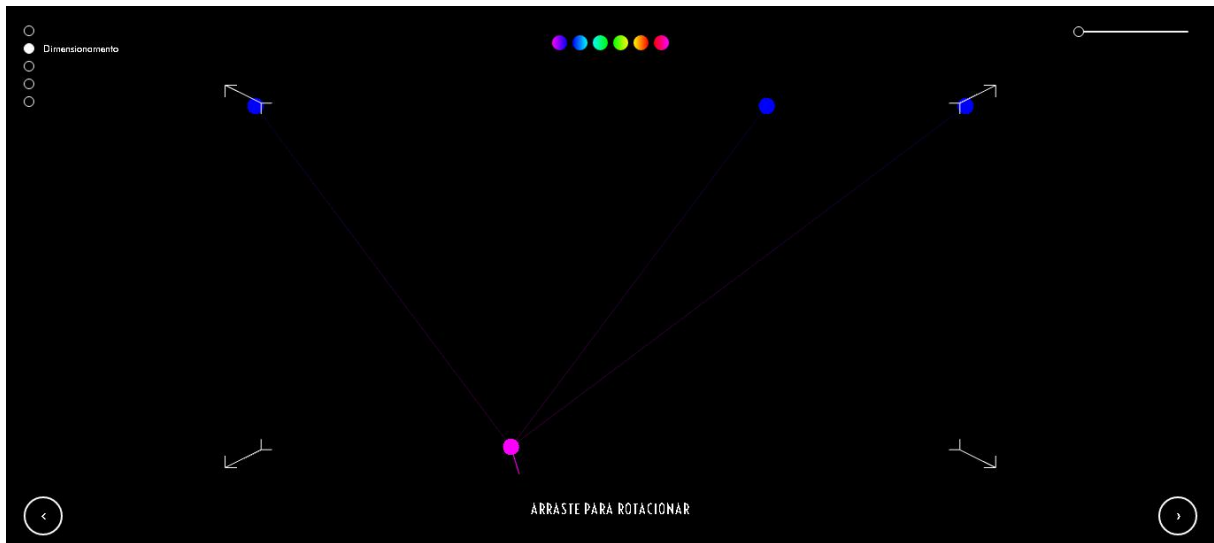
Figura 32- Estrutura Elementar - Programa.



Fonte: Elaborado pelo autor, 2021.

Clicando no menu de ‘*avançar*’ o programa faz o dimensionamento da geometria, expressando tanto a cor (com relação da deformação do ponto) quando um vetor apontando na direção da deformação, conforme Figura 33.

Figura 33- Estrutura Elementar Dimensionada.



Fonte: Elaborado pelo autor, 2021.

E também retorna no console os dados de deformação de cada grau de liberdade do sistema, mostrando um resultado preciso, demonstrando que a convergência do sistema está correta, conforme Figura 34. Além de expressar a deformação visualmente, conforme Figura 35

Figura 34- Estrutura Elementar Resultado da Deformação.

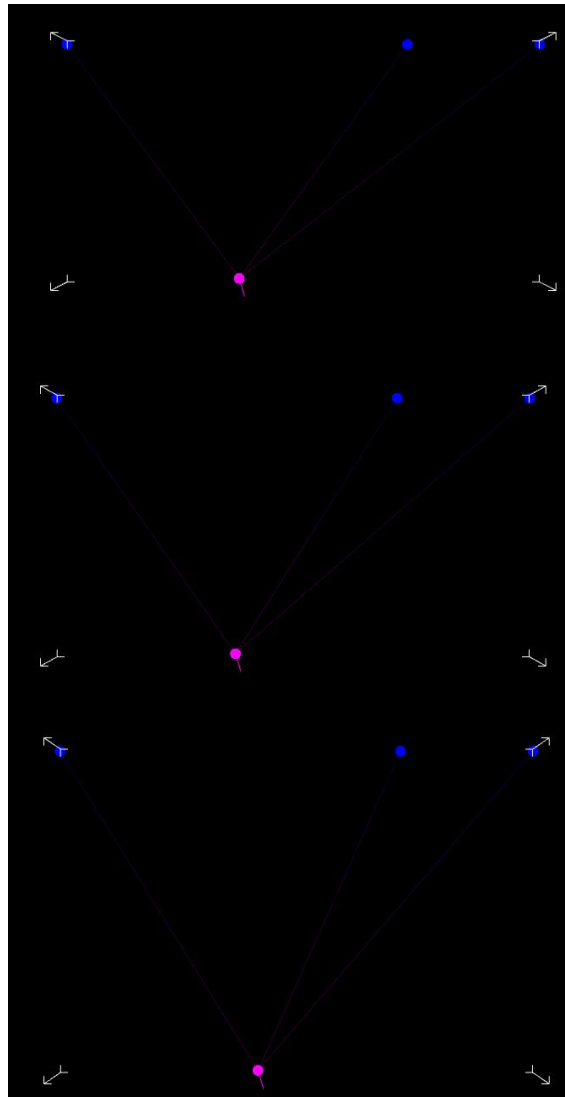
```

▶0: {id: '0|u', valor: 0.05617198334266177, identificado: 'true'}
▶1: {id: '0|v', valor: -0.18723994451829562, identificado: 'true'}
▶2: {id: '0|w', valor: 0, identificado: 'true'}
▶3: {id: '1|u', valor: 0, identificado: 'true'}
▶4: {id: '1|v', valor: 0, identificado: 'true'}
▶5: {id: '1|w', valor: 0, identificado: 'true'}
▶6: {id: '2|u', valor: 0, identificado: 'true'}
▶7: {id: '2|v', valor: 0, identificado: 'true'}
▶8: {id: '2|w', valor: 0, identificado: 'true'}
▶9: {id: '3|u', valor: 0, identificado: 'true'}
▶10: {id: '3|v', valor: 0, identificado: 'true'}
▶11: {id: '3|w', valor: 0, identificado: 'true'}

```

Fonte: Elaborado pelo autor, 2021.

Figura 35 - Estrutura Elementar - Fases de Deformação.



Fonte: Elaborado pelo autor, 2021.

7.2 FUNCIONALIDADE DO PROGRAMA

7.2.1 Manual

Para verificar se o programa converge para estruturas injetadas manualmente, foi feita a injeção de duas estruturas de domo, convencionalmente usadas para verificar a eficiência de programas de dimensionamento estrutural, disponibilizados por Papadrakakis 1981.

O processo para ambos é similar: o primeiro é um domo com 24 elementos, o segundo é um domo com 162 elementos.

7.2.1.1 Domo de 24 elementos

Primeiro passo é injetar a geometria manualmente, portanto dentro da função ‘*atualizarMalha*’ (APÊNDICE K) o objeto ‘*pontos*’ deve ser preenchido conforme a equação 5.4.

$$\begin{aligned} \text{pontos} = [& \\ & \{ \\ & \quad \text{identificador: 0,} \\ & \quad \text{posicionamento: \{ x: 43.3, y: 8.216, z: 51.2971 \},} \\ & \quad \text{conexoes: \{ identificador: [1, 2, 3, 4, 5, 6] \},} \\ & \quad \text{vetor: null, cor: null} \\ & \quad \}, \\ & \{ \\ & \quad \text{identificador: 1,} \\ & \quad \text{posicionamento: \{ x: 30.8, y: 6.216, z: 71.6492 \},} \\ & \quad \text{conexoes: \{ identificador: [0, 2, 6, 7, 12] \},} \\ & \quad \text{vetor: null, cor: null} \\ & \quad \}, \\ & \{ \\ & \quad \text{identificador: 2,} \\ & \quad \text{posicionamento: \{ x: 55.8, y: 6.216, z: 71.6492 \},} \\ & \quad \text{conexoes: \{ identificador: [0, 1, 3, 7, 8] \},} \\ & \quad \text{vetor: null, cor: null} \\ & \quad \}, \\ & \{ \\ & \quad \text{identificador: 3,} \\ & \quad \text{posicionamento: \{ x: 68.3, y: 6.216, z: 51.2971 \},} \\ & \quad \text{conexoes: \{ identificador: [0, 2, 4, 8, 9] \},} \\ & \quad \text{vetor: null, cor: null} \\ & \quad \}, \\ &] \end{aligned} \tag{5.4}$$

```

        {
            identificador: 4,
posicionamento: { x: 55.8, y: 6.216, z: 28.3479 },
            conexoes: { identificador: [ 0, 3, 5, 9, 10 ] },
            vetor: null, cor: null
        },
        {
            identificador: 5,
posicionamento: { x: 30.8, y: 6.216, z: 28.3479 },
            conexoes: { identificador: [ 0, 4, 6, 10, 11 ] },
            vetor: null, cor: null
        },
        {
            identificador: 6,
posicionamento: { x: 18.3, y: 6.216, z: 51.2971 },
            conexoes: { identificador: [ 0, 1, 5, 11, 12 ] },
            vetor: null, cor: null
        },
        {
            identificador: 7,
posicionamento: { x: 43.3, y: 0, z: 99.9971 },
            conexoes: { identificador: [ 1, 2 ] },
            vetor: null, cor: null
        },
        {
            identificador: 8,
posicionamento: { x: 86.6, y: 0, z: 74.9978 },
            conexoes: { identificador: [ 2, 3 ] },
            vetor: null, cor: null
        },
        {
            identificador: 9,
posicionamento: { x: 86.6, y: 0, z: 24.9993 },
            conexoes: { identificador: [ 3, 4 ] },
            vetor: null, cor: null
        },
        {
            identificador: 10,
posicionamento: { x: 43.3, y: 0, z: 0 },
            conexoes: { identificador: [ 4, 5 ] },
            vetor: null, cor: null
        },
        {
            identificador: 11,
posicionamento: { x: 0, y: 0, z: 24.9993 },
            conexoes: { identificador: [ 5, 6 ] },
            vetor: null, cor: null
        },
        {

```

```

        identificador: 12,
    posicionamento: { x: 0, y: 0, z: 74.9978 },
    conexoes: { identificador: [ 1, 6 ] },
    vetor: null, cor: null
    },
]

```

Os objetos de condição de contorno ‘ u ’ e ‘ f ’, junto do parâmetro de rigidez ‘ ks ’, devem ser criados, e passados como argumento da função ‘*initMatrizes*’ (APÊNDICE J), conforme equação 5.5.

$$\begin{aligned}
 f = [& \\
 & \{ \text{identificador: 0, x: 0, y: -7.81, z: 0} \}, \\
 & \{ \text{identificador: 1, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 2, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 3, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 4, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 5, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 6, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 7, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 8, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 9, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 10, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 11, x: 0, y: 0, z: 0} \}, \\
 & \{ \text{identificador: 12, x: 0, y: 0, z: 0} \} \\
 &] \\
 u = [& \\
 & \{ \text{identificador: 0, u: 'indefinido', v: 'indefinido', w: 'indefinido'} \}, \\
 & \{ \text{identificador: 1, u: 'indefinido', v: 'indefinido', w: 'indefinido'} \}, \\
 & \{ \text{identificador: 2, u: 'indefinido', v: 'indefinido', w: 'indefinido'} \}, \\
 & \{ \text{identificador: 3, u: 'indefinido', v: 'indefinido', w: 'indefinido'} \}, \\
 & \{ \text{identificador: 4, u: 'indefinido', v: 'indefinido', w: 'indefinido'} \}, \\
 & \{ \text{identificador: 5, u: 'indefinido', v: 'indefinido', w: 'indefinido'} \}, \\
 & \{ \text{identificador: 6, u: 'indefinido', v: 'indefinido', w: 'indefinido'} \}, \\
 & \{ \text{identificador: 7, u: 0, v: 0, w: 0} \},
 \end{aligned} \tag{5.5}$$

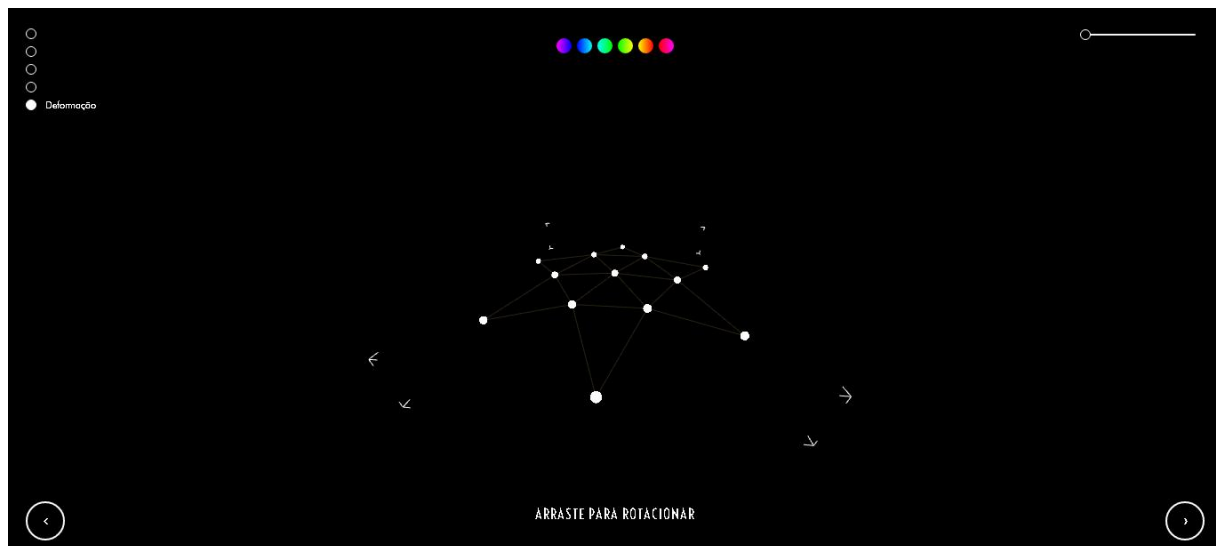
```

{ identificador: 8, u: 0, v: 0, w: 0 },
{ identificador: 9, u: 0, v: 0, w: 0 },
{ identificador: 10, u: 0, v: 0, w: 0 },
{ identificador: 11, u: 0, v: 0, w: 0 },
{ identificador: 12, u: 0, v: 0, w: 0 }
]
parametros.rigidez = { A: 1, E: 10**4 }

```

Ao mudar o parâmetro modo para '*manual*', será exibida a geometria na tela, Figura 36.

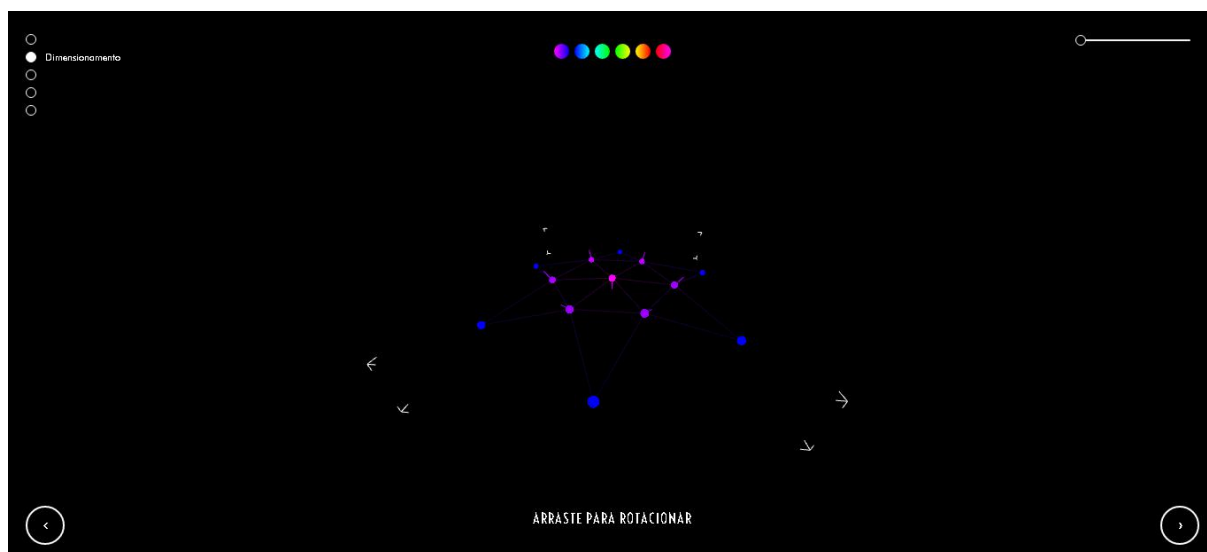
Figura 36- Domo 24 Elementos.



Fonte: Elaborado pelo autor, 2021.

Clicando no menu de '*avançar*' o programa faz o dimensionamento da geometria, expressando tanto a cor (com relação da deformação do ponto) quando um vetor apontando na direção da deformação, conforme Figura 37.

Figura 37 - Domo 24 Elementos - Dimensionado.



Fonte: Elaborado pelo autor, 2021.

7.2.1.2 Domo 162 Elementos

Primeiro passo é injetar a geometria manualmente, portanto dentro da função *‘atualizarMalha’* (APÊNDICE K) o objeto *‘pontos’* deve ser preenchido conforme a APÊNDICE X.

Os objetos de condição de contorno *‘u’* e *‘f’*, junto do parâmetro de rigidez *‘ks’*, devem ser criados, e passados como argumento da função *‘initMatrizes’* (APÊNDICE J).

Contudo, por serem muitas conexões, a implementação foi feita usando uma pequena rotina computacional, conforme equação 5.7.

```
f = []
u = []
parametros.rigidez = { A: 1, E: 1 }
for ( let i = 0; i <= 72; i++ ) {
    if ( i == 0 ) {
f.push( { identificador: i, x: 0, y: -100/(10**4), z: 0 } )
    } else {
f.push( { identificador: i, x: 0, y: 0, z: 0 } )
    }
}
```

(5.7)

```

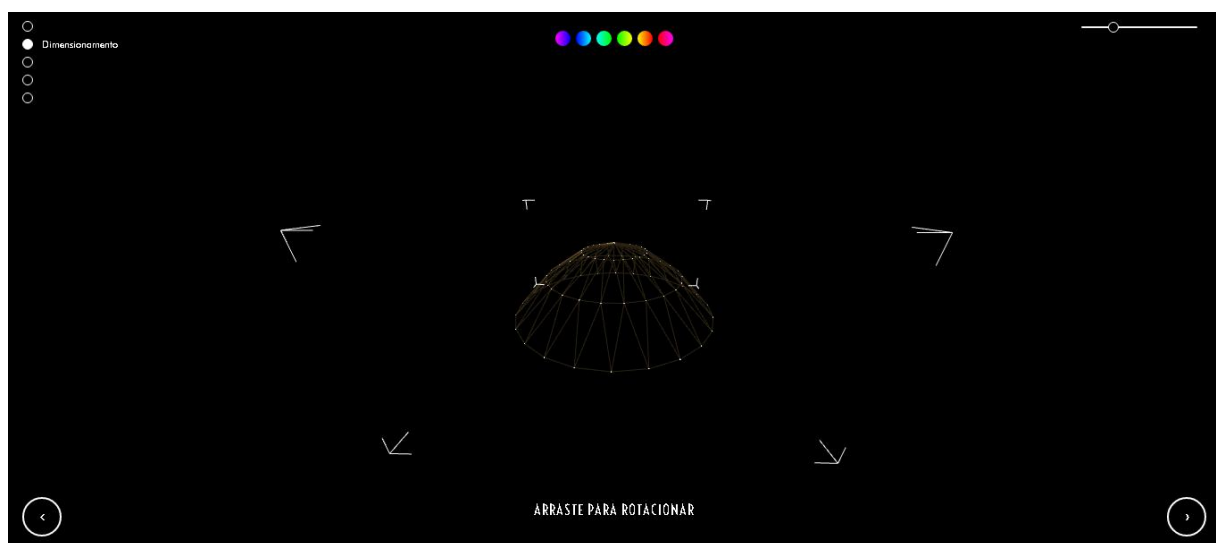
    }

    if ( i < 49 ) {
u.push( { identificador: i, u: 'indefinido', v: 'indefinido', w:
'indefinido' } )
    } else {
u.push( { identificador: i, u: 0, v: 0, w: 0 } )
    }
}

```

Ao mudar o parâmetro ‘*modo*’ de ‘*automatoco*’ para ‘*manual*’, será exibida a geometria na tela, Figura 38.

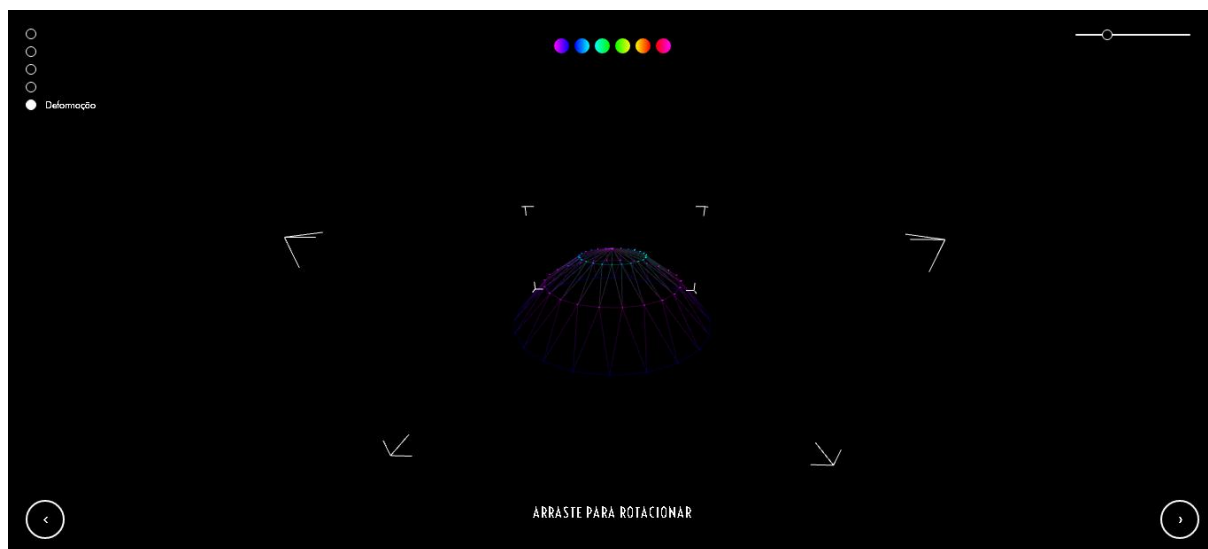
Figura 38- Domo 162 Elementos.



Fonte: Elaborado pelo autor, 2021.

Clicando no menu de ‘*avançar*’ o programa faz o dimensionamento da geometria, expressando tanto a cor (com relação da deformação do ponto) quando um vetor apontando na direção da deformação, conforme Figura 39.

Figura 39 - Domo 162 Elementos - Dimensionado.



Fonte: Elaborado pelo autor, 2021.

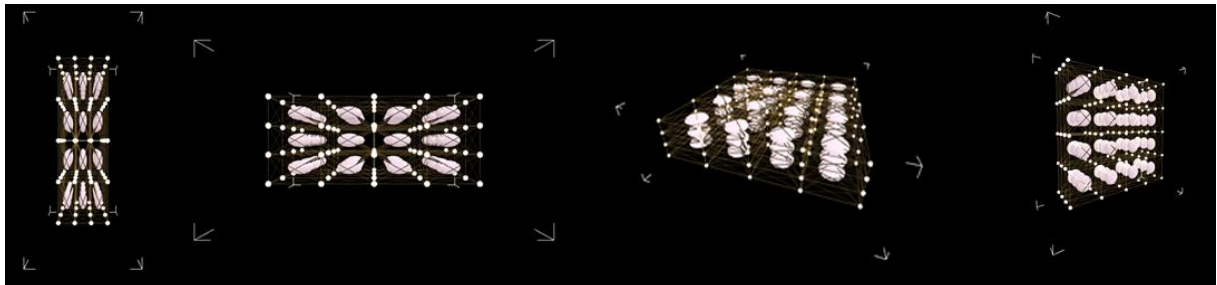
7.2.2 Automático

O funcionamento do programa no modo automático será verificado das cinco (05) fases do programa (Discretização, Dimensionamento, Isovalores, Fluxo, e Deformação).

7.2.2.1 Discretização

Foi feita a verificação dos controles e responsividade da malha aplicando quatro geometrias distintas (Pilar, Viga, Laje, Parede), conforme figura 40.

Figura 40 - Pilar, Viga, Laje, Parede - Discretização.

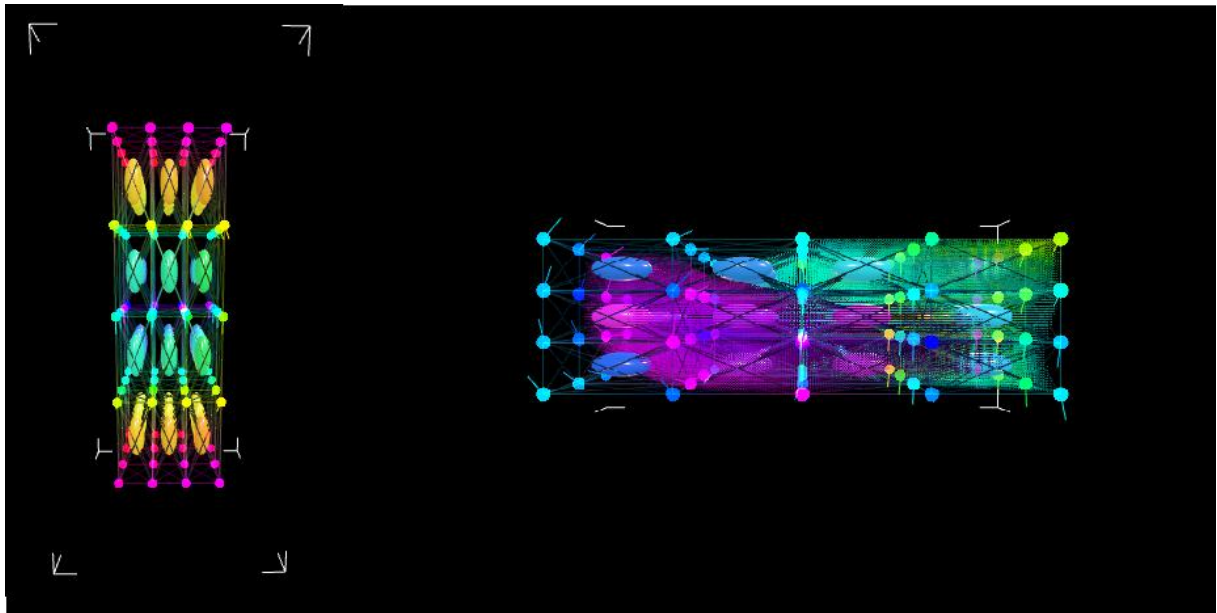


Fonte: Elaborado pelo autor, 2021.

7.2.2.2 Dimensionamento

A verificação da convergência e representação gráfica do dimensionamento das deformações foi feita em duas geometrias distintas, (Pilar e Viga), conforme Figura 41.

Figura 41 - Pilar, Viga - Dimensionamento.

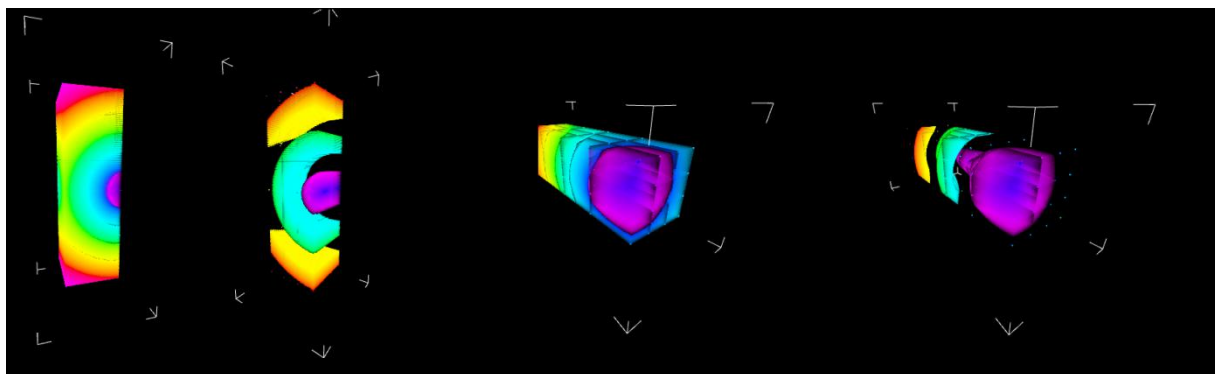


Fonte: Elaborado pelo autor, 2021.

7.2.2.3 Isovalores

A verificação da representação gráfica dos isovalores, e do menu de visualização e isolamento dos intervalos de deformação foi feita para duas geometrias distintas (Pilar e Viga), conforme Figura 42.

Figura 42 - Pilar, Viga - Isovalores.

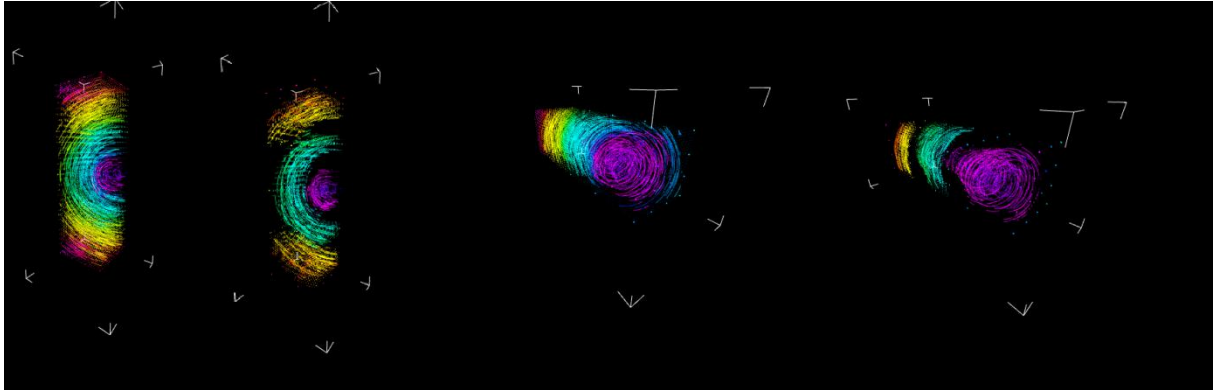


Fonte: Elaborado pelo autor, 2021.

7.2.2.4 Fluxo

A verificação da representação gráfica do fluxo de deformações foi feita para as duas geometrias distintas (Pilar e Viga), conforme figura 41.

Figura 43- Pilar, Viga - Fluxo.

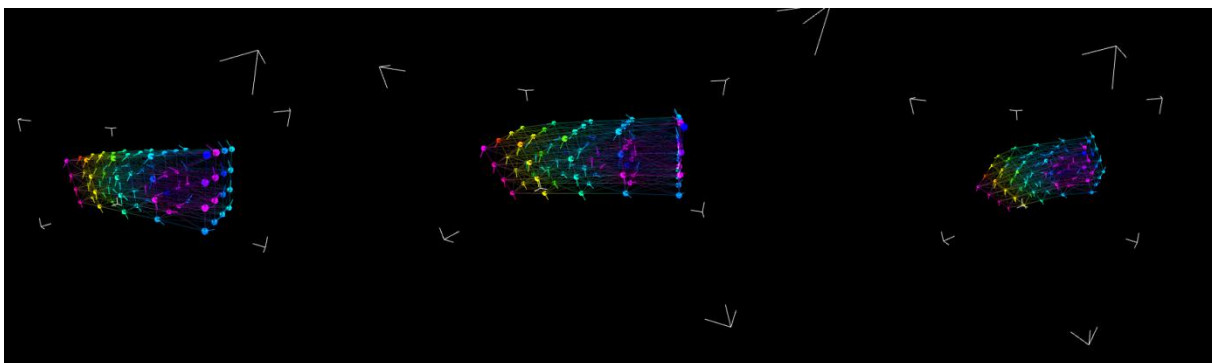


Fonte: Elaborado pelo autor, 2021.

7.2.2.5 Deformação

A verificação da representação gráfica da deformação foi feita para para a geometria da Viga, visualizando essa deformação em três graus de intensidade crescentes, conforme figura 42.

Figura 44 - Viga - Deformação.



Fonte: Elaborado pelo autor, 2021.

8 CONCLUSÃO

A intenção no desenvolvimento dessa pesquisa foi desenvolver um programa que mostrasse uma perspectiva alternativa de um aspecto muito importante da análise estrutural, a deformação. O simulador correspondeu a essa intenção, mostrando que é possível representar o fluxo de deformação.

Foi desenvolvido como um '*simulador para Vigas*', mas foi configurado de forma adaptativa, expandindo a abordagem do programa para outros conjuntos de estrutura (*pilares, lajes, muros de contenção*). O desenvolvimento baseado em elementos lineares apresentou resultados de dimensionamento competentes, e a metodologia adotada para interpolação (por produto de Tensores Lineares) se adequou às geometrias propostas (sem deixar nenhum artefato de descontinuidade entre elementos vizinhos).

Para facilitar a visualização do fluxo, foi tomado o cuidado de desenvolver uma interface que permite gerenciar intervalos de deformação que serão graficamente representados, além disso (para validação do programa) foi configurado um modo '*manual*' de operação no programa (que pode ser usado para dimensionar qualquer sistema complexo de barras que não seja gerado nativamente pelo programa no modo '*automático*').

Apesar do simulador ter se apresentado convenientemente dinâmico, adaptável, e da análise do fluxo poder trazer interpretações interessantes sobre o comportamento estrutural, essa pesquisa está focada no desenvolvimento do simulador, não tendo muito a contribuir na análise desses dados. Contudo, a implementação, experimentação e análise desses dados podem ser realizadas em pesquisas futuras.

9 REFERÊNCIAS

ADDIS, B. **Building: 3000 Years of Design, Engineering and Construction**. London: Phaidon, 2007

ANDREATTA, L. C.; DA SILVA RAMIRO, F. **O estudo de estruturas hiperestáticas à luz da teoria da Aprendizagem Significativa**. Revista Liberato, v. 18, n. 29, 2017.

BALANKIN, Alexander S. **Toward the mechanics of fractal materials: mechanics of continuum with fractal metric**. arXiv preprint arXiv:1409.5829, 2014.

BEDIN, Luciano. **A equação de Poisson-Boltzmann em regiões com fronteira irregular**. 2002.

BORGES, Gustavo H. A. B.; BRITO, Sarah C. **Desenvolvimento de Modelo Didático Qualitativo de Pórtico Hiperestático**. 11ª Jornada de Iniciação Científica e Extensão, 2020.

CRAIZER, Marcos. **Cálculo integral a várias variáveis**. Edicoes Loyola, 2002.

DAMIANCE, Patrícia R. M.; BODONI, Patrícia S. B.; BASTOS, José. R. M. **O modelo participativo na construção do saber científico em atividade de ensino**. Revista de Educação, v.14, n. 17, 2011

DIRKSEN, Jos. **Learning Three.js—the JavaScript 3D Library for WebGL**. Packt Publishing Ltd, 2015.

DURO-ROYO, Jorge; OXMAN, Neri. **Towards Fabrication Information Modeling (FIM): Four Case Models to Derive Designs informed by Multi-Scale Trans-Disciplinary Data**. MRS Online Proceedings Library, v. 1800, n. 1, p. 1-8, 2015.

FLANAGAN, David. **JavaScript: o guia definitivo**. Bookman Editora, 2004.

GIACOMINI, Fabiana de Fátima. **Verificação da forma de aplicar condições de contorno em problemas unidimensionais com o método dos volumes finitos**. 2009.

JAMBA, Osvaldo Pedro Candeia. **Resolução numérica da equação de Poisson**. 2019. Tese de Doutorado.

LEET, K. M.; UANG, C. M. e GILBERT, A M. **Fundamentos da Análise Estrutural-3**. AMGH Editora, 2014.

LOBOSCO, Tales. **Modelos Didáticos para Exploração e Compreensão Estrutural**. ENCONTRO NACIONAL DE ENSINO DE ESTRUTURAS EM ESCOLAS DE ARQUITETURA, v. 3, p. 796-809, 2017.

MASE, George E.; MASE, G. E. **Continuum mechanics**. New York: McGraw-Hill, 1970.

MESIT, Jaruan, "**Modeling And Simulation Of Soft Bodies**" (2010). Electronic Theses and Dissertations, 2004-2019. 1647.

MOURA, A. C. et al. **Desenvolvimento de um modelo reduzido didático qualitativo e quantitativo de treliça plana**. In: XLV Congresso Brasileiro de Educação em Engenharia, Natal/RN. Anais. Natal. 2016.

MORTON, Keith W.; MAYERS, David Francis. **Numerical solution of partial differential equations: an introduction**. Cambridge university press, 2005.

OXMAN, Neri. **Material-based design computation: Tiling behavior**. 2009.

OXMAN, Neri. **Structuring materiality: design fabrication of heterogeneous materials**. Architectural Design, v. 80, n. 4, p. 78-85, 2010.

OXMAN, Neri. Finite Element Synthesis. Proceedings of VRAP: **Advanced Research in Virtual and Rapid Prototyping in: Innovative Developments in Virtual and Physical Prototyping**", PJ Bártolo et al., published by Taylor & Francis, 2011.

OXMAN, Neri et al. **Biological Computation for Digital Design and Fabrication**. 2013.

PAPADRAKAKIS, M., 1981. Post-buckling Analysis of Spatial Structures by Vector Iteration Methods. Computers and Structures, vol. 14, pp. 393–402.

PRAVIA, Zacarias Martin Chamberlain; BORDIGNON, Rodrigo. **Modelos intuitivos para ensino de estabilidade das estruturas**. In: XVIII Congresso Brasileiro de Ensino de Engenharia, Ouro Preto. Anais... Ouro Preto, Minas Gerais. 2000.

SARAMAGO, R. C. P. **Ensino de estruturas nas escolas de Arquitetura do Brasil**. 2011. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2011.

RINCÓN, Carlos Alberto Matallana. **Diseño Paramétrico Enfocado a Diagnosticar Espacios Urbanos ee Pequeño Formato**, 2018.

SANCHEZ-GONZALEZ, Alvaro et al. **Learning to simulate complex physics with graph networks**. In: International Conference on Machine Learning. PMLR, 2020. p. 8459-8468.

SARAMAGO, R. C. P. **Ensino de estruturas nas escolas de Arquitetura do Brasil**. 2011. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2011.

SILVA, Fernanda Moronari da. **Cálculo de deflexões usando a Resistência dos Materiais e a Teoria da Elasticidade**. 2017.

TORATTI, Luiz Otávio. **Design de campos vetoriais em volumes usando RBF**. Tese de Doutorado. Universidade de São Paulo.

ANFLOR, Carla Tatiana Mota. **Otimização evolucionária e topológica em problemas governados pela equação de poisson empregando o método dos elementos de contorno**. 2007.

TORROJA, E. **Razón y Ser de los Tipos Estructurales**. Madrid: Consejo Superior de Investigaciones Científicas, 1957.

WOLPER, Joshua et al. **CD-MPM: continuum damage material point methods for dynamic fracture animation**. ACM Transactions on Graphics (TOG), v. 38, n. 4, p. 1-15, 2019.

WOLPER, Joshua et al. **AnisoMPM: Animating anisotropic damage mechanics: Supplemental document**. ACM Trans. Graph, v. 39, n. 4, 2020.

APÊNDICE A - VERTICES

Quadro 02 - Função initPontos.

initPontos

```
function initPontos( malha ) {  
    // organizando variaveis...  
    let dx, dy, dz  
    dx = malha.dimensoes.discretas.x  
    dy = malha.dimensoes.discretas.y  
    dz = malha.dimensoes.discretas.z  
  
    let ux, uy, uz  
    ux = malha.unidades.x  
    uy = malha.unidades.y  
    uz = malha.unidades.z  
    // ... variaveis organizadas.  
  
    // iniciando criação de pontos...  
    let pontos = []  
    let index = 0  
    for ( let x = 0; x <= ux ; x++ ) {  
        for ( let y = 0; y <= uy ; y++ ) {  
            for ( let z = 0; z <= uz ; z++ ) {  
  
                let fx, fy, fz  
  
                let r = 1  
                fx = r*( x * dx )  
                fy = r*( y * dy )  
                fz = r*( z * dz )  
  
                let identificador, posicionamento, conexoes  
                identificador = index++  
            }  
        }  
    }  
}
```

```

    posicionamento = { x: fx, y: fy, z: fz }
    conexoes = { identificador: [] }

    let ponto = { identificador, posicionamento, conexoes, vetor: null, cor: null }

    pontos.push( ponto )
  }
}
// ...finalizando criação de pontos
return pontos
}

export default initPontos

```

Fonte: Elaborado pelo autor, 2021.

Quadro 03 - Função addConexoesPontos.

addConexoesPontos

```

function addConexoesPontos( malha, pontos, parametros ) {
  // organizando variaveis...
  let dx, dy, dz
  dx = malha.dimensoes.discretas.x
  dy = malha.dimensoes.discretas.y
  dz = malha.dimensoes.discretas.z
  let maximaDistancia = Math.sqrt( dx * dx + dy * dy + dz * dz )
  // ... variaveis organizadas.

  for ( let a of pontos ) {
    for ( let b of pontos ) {
      let lx, ly, lz, distancia
      lx = Math.abs( ( b.posicionamento.x ) - ( a.posicionamento.x ) )
      ly = Math.abs( ( b.posicionamento.y ) - ( a.posicionamento.y ) )
      lz = Math.abs( ( b.posicionamento.z ) - ( a.posicionamento.z ) )

```

```

    distancia = Math.sqrt( lx * lx + ly * ly + lz * lz )

    if ( parametros.mode == 'manual' ) {

    } else {
        if ( distancia <= maximaDistancia && distancia > 0 ) {
            if ( lx > 0 && ly > 0 && lz > 0 ) {
                a.conexoes.identificador.push( b.identificador )
            }
            if ( lx == 0 && ly <= dy && lz <= dz ) {
                a.conexoes.identificador.push( b.identificador )
            }
            if ( ly == 0 && lx <= dx && lz <= dz ) {
                a.conexoes.identificador.push( b.identificador )
            }
            if ( lz == 0 && lx <= dx && ly <= dy ) {
                a.conexoes.identificador.push( b.identificador )
            }
        }
    }

}

return pontos
}

export default addConexoesPontos

```

Fonte: Elaborado pelo autor, 2021.

Quadro 04 - Função adicionarVetoresNosPontos.

adicionarVetoresNosPontos

```
function adicionarVetoresNosPontos( pontos, matrizes ) {  
  console.log( 'Adicionando Vetores U aos pontos...' )  
  for ( let i in pontos ) {  
    pontos[i].vetor = matrizes.vetores.vetores[i].vetor  
  }  
  console.log( '... foram adicionandos Vetores U aos pontos.' )  
  return pontos  
}  
export default adicionarVetoresNosPontos
```

Fonte: Elaborado pelo autor, 2021.

Quadro 05 - Função adicionarCoresNosPontos.

adicionarCoresNosPontos

```
import parametrizarCores from '../parametrizar cores/parametrizarCores.js'  
  
function adicionarCoresNosPontos( pontos, matrizes ) {  
  console.log( 'Adicionando Cores aos pontos...' )  
  for ( let i in pontos ) {  
    let min, max, ut  
    min = matrizes.vetores.min  
    max = matrizes.vetores.max  
    ut = pontos[i].vetor.abs  
    pontos[i].cor = parametrizarCores( min, max, ut )  
  }  
  console.log( '... foram adicionandos Cores aos pontos.' )  
  return pontos  
}  
  
export default adicionarCoresNosPontos
```

Fonte: Elaborado pelo autor, 2021.

Quadro 06 - Função deformarPontos.

deformarPontos

```

function deformarPontos( pontos ) {
  for ( let i in pontos ) {
    let u, v, w

    u = pontos[i].vetor.u
    v = pontos[i].vetor.v
    w = pontos[i].vetor.w

    pontos[i].posicionamento.x += u
    pontos[i].posicionamento.y += v
    pontos[i].posicionamento.z += w
  }
  return pontos
}

export default deformarPontos

```

Fonte: Elaborado pelo autor, 2021.

Quadro 07 - Função desenharPontos.

desenharPontos

```

import * as THREE from '../three.module.js'

function desenharPontos( cena, pontos, T, mt ) {
  console.log( 'Desenhando malha...' )
  // organizando variaveis...
  let dx, dy, dz
  dx = T.dimensoes.discretas.x
  dy = T.dimensoes.discretas.y
  dz = T.dimensoes.discretas.z

  let dm = ( dx >= dy ? ( dy >= dz ? dz : dy ) : ( dx ) ) * .6

  let posicionamentoPontos = new Float32Array( pontos.length * 3 )
  let coresPontos = new Uint8Array( pontos.length * 3 )

```

```

for ( let i in pontos ) {
    posicionamentoPontos[ ( 3 * i ) + 0 ] = pontos[i].posicionamento.x
    posicionamentoPontos[ ( 3 * i ) + 1 ] = pontos[i].posicionamento.y
    posicionamentoPontos[ ( 3 * i ) + 2 ] = pontos[i].posicionamento.z

    if ( pontos[i].cor == null ) {
        coresPontos[ ( 3 * i ) + 0 ] = Math.floor( 255 )
        coresPontos[ ( 3 * i ) + 1 ] = Math.floor( 255 )
        coresPontos[ ( 3 * i ) + 2 ] = Math.floor( 255 )
    } else {
        coresPontos[ ( 3 * i ) + 0 ] = pontos[i].cor.r * 255
        coresPontos[ ( 3 * i ) + 1 ] = pontos[i].cor.g * 255
        coresPontos[ ( 3 * i ) + 2 ] = pontos[i].cor.b * 255
    }
}

let pontosGeometria = new THREE.BufferGeometry()
pontosGeometria.setAttribute(                                'position',                                new
THREE.BufferAttribute( posicionamentoPontos, 3 ) )
pontosGeometria.setAttribute( 'color', new THREE.BufferAttribute( coresPontos, 3, true ) )
pontosGeometria.setDrawRange( 0, pontos.length )

let pontosMaterial = new THREE.PointsMaterial({
    vertexColors: THREE.VertexColors,
    onBeforeCompile: shader => {
        shader.fragmentShader = shader.fragmentShader.replace(
            `#include <clipping_planes_fragment>`,
            `
            if (length(gl_PointCoord - 0.5) > 0.5 ) discard; // make points round
            #include <clipping_planes_fragment>
            `
        );
    }
});

```

```

    },
    size: dm*mt,
    blending: THREE.AdditiveBlending,
    transparent: true
  })

```

```

let mesh = new THREE.Points( pontosGeometria, pontosMaterial )

```

```

cena.add(mesh)

```

```

  console.log( '... Malha desenhada' )
}

```

```

export default desenharPontos

```

Fonte: Elaborado pelo autor, 2021.

Quadro 08 - Função desenharVetores.

desenharVetores

```

import * as THREE from '../three.module.js'

```

```

function desenharVetores( estrutura, pontos, malha ) {

```

```

  console.log( 'Desenhando vetores...' )

```

```

  // organizando variaveis...

```

```

  let dx, dy, dz

```

```

  dx = malha.dimensoes.discretas.x

```

```

  dy = malha.dimensoes.discretas.y

```

```

  dz = malha.dimensoes.discretas.z

```

```

  let dm = ( dx >= dy ? ( dy >= dz ? dz : dy ) : ( dx ) ) * .5

```

```

  let posicionamentoVetores = new Float32Array( pontos.length * 3 * 2 )

```

```

  let coresVetores = new Uint8Array( pontos.length * 3 * 2 )

```

```

for ( let i in pontos ) {
    let nx, ny, nz

    nx = pontos[i].vetor.normalizado.nx * dm
    ny = pontos[i].vetor.normalizado.ny * dm
    nz = pontos[i].vetor.normalizado.nz * dm

    posicionamentoVetores[ ( 6 * i ) + 0 ] = pontos[i].posicionamento.x
    posicionamentoVetores[ ( 6 * i ) + 1 ] = pontos[i].posicionamento.y
    posicionamentoVetores[ ( 6 * i ) + 2 ] = pontos[i].posicionamento.z

    posicionamentoVetores[ ( 6 * i ) + 3 ] = pontos[i].posicionamento.x + nx
    posicionamentoVetores[ ( 6 * i ) + 4 ] = pontos[i].posicionamento.y + ny
    posicionamentoVetores[ ( 6 * i ) + 5 ] = pontos[i].posicionamento.z + nz

    coresVetores[ ( 6 * i ) + 0 ] = pontos[i].cor.r * 255
    coresVetores[ ( 6 * i ) + 1 ] = pontos[i].cor.g * 255
    coresVetores[ ( 6 * i ) + 2 ] = pontos[i].cor.b * 255

    coresVetores[ ( 6 * i ) + 3 ] = pontos[i].cor.r * 255
    coresVetores[ ( 6 * i ) + 4 ] = pontos[i].cor.g * 255
    coresVetores[ ( 6 * i ) + 5 ] = pontos[i].cor.b * 255

}

let vetoresGeometria = new THREE.BufferGeometry()
vetoresGeometria.setAttribute(                                     'position',                                     new
THREE.BufferAttribute( posicionamentoVetores, 3 ) )
vetoresGeometria.setAttribute( 'color', new THREE.BufferAttribute( coresVetores, 3,
true ) )
vetoresGeometria.setDrawRange( 0, pontos.length * 2 )

```

```
let vetoresMaterial = new THREE.LineBasicMaterial({
  vertexColors: true,
  blending: THREE.AdditiveBlending,
  transparent: true
})

let mesh = new THREE.LineSegments( vetoresGeometria, vetoresMaterial )

estrutura.add(mesh)

console.log( '... Vetores desenhados' )
}

export default desenharVetores
```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE B - CONEXÕES

Quadro 09 - Função initConexoes.

initConexoes
<pre>function initConexoes(pontos) { let conexoes = [] for (let a of pontos) { for (let b of pontos) { for (let i in b.conexoes.identificador) { if (a.identificador == b.conexoes.identificador[i]) { let inicio let final if (a.identificador < b.identificador) { inicio = a final = b } else { inicio = b final = a } let pontoA, pontoB pontoA = { id: inicio.identificador, pos: [inicio.posicionamento.x , inicio.posicionamento.y , inicio.posicionamento.z], cor: inicio.cor } pontoB = { id: final.identificador, pos: [final.posicionamento.x , final.posicionamento.y , final.posicionamento.z], cor: final.cor } } } } } }</pre>

```

        conexoes.push(
            {
                pontoA,
                pontoB,
                nc:
                `${inicio.identificador}|${final.identificador}` } )
        }
    }
}
}
return conexoes
}

```

```
export default initConexoes
```

Fonte: Elaborado pelo autor, 2021.

Quadro 10 - Função desenharConexoes.

desenharConexoes

```

import * as THREE from '../three.module.js'

function desenharConexoes( cena, conexoes ) {
    console.log( 'Desenhando conexoes...' )
    let index = 0
    let indexcores = 0
    let posicionamentoConexoes = new Float32Array( conexoes.length * 6 )
    let coresConexao = new Uint8Array( conexoes.length * 6 )
    let conexoesMaterial

    for ( let i in conexoes ) {
        posicionamentoConexoes[ index++ ] = conexoes[i].pontoA.pos[0]
        posicionamentoConexoes[ index++ ] = conexoes[i].pontoA.pos[1]
        posicionamentoConexoes[ index++ ] = conexoes[i].pontoA.pos[2]

        posicionamentoConexoes[ index++ ] = conexoes[i].pontoB.pos[0]
        posicionamentoConexoes[ index++ ] = conexoes[i].pontoB.pos[1]
        posicionamentoConexoes[ index++ ] = conexoes[i].pontoB.pos[2]
    }
}

```

```
if ( conexoes[i].pontoA.cor == null ) {
    let r, g, b
    r = 217
    g = 164
    b = 84
    coresConexao[ indexcores ++ ] = r
    coresConexao[ indexcores ++ ] = g
    coresConexao[ indexcores ++ ] = b

    coresConexao[ indexcores ++ ] = r
    coresConexao[ indexcores ++ ] = g
    coresConexao[ indexcores ++ ] = b

    conexoesMaterial = new THREE.LineBasicMaterial({
        color: 'rgb( 217, 164, 84 )',
        vertexColors: true,
        blending: THREE.AdditiveBlending,
        transparent: true,
        opacity: .1
    })
} else {
    coresConexao[ indexcores ++ ] = conexoes[i].pontoA.cor.r * 255
    coresConexao[ indexcores ++ ] = conexoes[i].pontoA.cor.g * 255
    coresConexao[ indexcores ++ ] = conexoes[i].pontoA.cor.b * 255

    coresConexao[ indexcores ++ ] = conexoes[i].pontoB.cor.r * 255
    coresConexao[ indexcores ++ ] = conexoes[i].pontoB.cor.g * 255
    coresConexao[ indexcores ++ ] = conexoes[i].pontoB.cor.b * 255

    conexoesMaterial = new THREE.LineBasicMaterial({
        vertexColors: true,
        blending: THREE.AdditiveBlending,
        transparent: true,
```

```
        opacity: .1
      })
    }

  }

  let conexoesGeometria = new THREE.BufferGeometry()
  conexoesGeometria.setAttribute( 'position', new
  THREE.BufferAttribute( posicionamentoConexoes, 3 ) )
  conexoesGeometria.setAttribute( 'color', new THREE.BufferAttribute( coresConexao, 3,
  true ) )
  conexoesGeometria.setDrawRange( 0, conexoes.length*6 )

  let conexoesMesh = new THREE.LineSegments( conexoesGeometria, conexoesMaterial )
  cena.add(conexoesMesh)

  console.log( '... Conexoes desenhadas' )
}

export default desenharConexoes
```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE C - CELULAS

Quadro 11 - Função initCelulas.

initCelulas

```
function initCelulas( malha ) {  
  console.log( 'Criando Celulas...' )  
  // organizando variaveis...  
  let dx, dy, dz  
  dx = malha.dimensoes.discretas.x  
  dy = malha.dimensoes.discretas.y  
  dz = malha.dimensoes.discretas.z  
  let ux, uy, uz  
  ux = malha.unidades.x  
  uy = malha.unidades.y  
  uz = malha.unidades.z  
  // ... variaveis organizadas.  
  
  // iniciando criação de celulas...  
  let celulas = []  
  let index = 0  
  for ( let x = 0; x < ux ; x++ ) {  
    for ( let y = 0; y < uy ; y++ ) {  
      for ( let z = 0; z < uz ; z++ ) {  
        let fx, fy, fz  
  
        fx = ( x * dx ) + dx/2  
        fy = ( y * dy ) + dy/2  
        fz = ( z * dz ) + dz/2  
  
        let identificador, posicionamento, conexoes  
        identificador = index++  
        posicionamento = { x: fx, y: fy, z: fz }  
        conexoes = []  
      }  
    }  
  }  
}
```

```
    let celula = { identificador, posicionamento, conexoes, vertices: null, vetores: null }

    celulas.push( celula )
  }
}
// ...finalizando criação de celulas
console.log( '... Celulas criadas.' )
return celulas
}
export default initCelulas
```

Fonte: Elaborado pelo autor, 2021.

Quadro 12 - Função desenharCelulas.

```
desenharCelulas

import * as THREE from '../three.module.js'

function desenharCelulas( celulas, esferas, estrutura, malha ) {
  console.log( 'Desenhando celulas...' )
  // organizando variaveis...
  let dx, dy, dz
  dx = malha.dimensoes.discretas.x
  dy = malha.dimensoes.discretas.y
  dz = malha.dimensoes.discretas.z
  let ux, uy, uz
  ux = malha.unidades.x
  uy = malha.unidades.y
  uz = malha.unidades.z
  // ... variaveis organizadas.

  for ( let celula of celulas ) { desenharCelula( esferas, celula.posicionamento, dx, dy, dz ) }
```

```

    estrutura.add( esferas )
    console.log( '... Celulas desenhadas' )
}
function desenharCelula( esferas, pos, dx, dy, dz ) {
    let material, geometria, mesh
    material = new THREE.MeshPhysicalMaterial({
        color: 'rgb(255,203,225)',
        wireframe: false,
        metalness: 0,
        roughness: 0,
        transparent: true,
        reflectivity: 1,
        refractionRatio: 1,
        opacity: 1
    })
    //geometria = new THREE.BoxGeometry( raio, raio, raio )
    geometria = new THREE.SphereGeometry( .5 , 40, 40 )

    mesh = new THREE.Mesh( geometria, material )
    mesh.scale.set( dx*.5, dy*.5, dz*.5 )
    mesh.position.set( pos.x, pos.y, pos.z )
    mesh.geometry.computeVertexNormals()
    mesh.castShadow = true;
    esferas.add(mesh)
}

export default desenharCelulas

```

Fonte: Elaborado pelo autor, 2021.

Quadro 13- Função addConexoesCelulas.

addConexoesCelulas

```

function addConexoesCelulas( malha, pontos, celulas ) {
    console.log( "Adicionando conexões nas Celulas..." )

```

```

// organizando variaveis...
let dx, dy, dz
dx = malha.dimensoes.discretas.x
dy = malha.dimensoes.discretas.y
dz = malha.dimensoes.discretas.z
let maximaDistancia = Math.sqrt( dx * dx + dy * dy + dz * dz ) / 2
// ... variaveis organizadas.

for ( let celula of celulas ) {
  for ( let ponto of pontos ) {
    let dx, dy, dz, lo
    dx = celula.posicionamento.x - ponto.posicionamento.x
    dy = celula.posicionamento.y - ponto.posicionamento.y
    dz = celula.posicionamento.z - ponto.posicionamento.z
    lo = Math.sqrt( dx * dx + dy * dy + dz * dz )
    if ( lo <= maximaDistancia ) {
      celula.conexoes.push( ponto )
    }
  }
}

console.log( "... as conexoes das Celulas foram adicionadas." )
return celulas
}

export default addConexoesCelulas

```

Fonte: Elaborado pelo autor, 2021.

Quadro 14 - Função mapearVerticesCelula.

mapearVerticesCelula

```

function mapearVerticesCelulas( celulas ) {
  console.log( "Mapeando Vertices das Celulas..." )
  //adicionando indice de faces conectadas aos pontos
  for ( let celula of celulas ) {

```

```

let v1, v2, v3, v4, v5, v6, v7, v8
for ( let ponto of celula.conexoes ) {
    let dx, dy, dz
    dx = celula.posicionamento.x - ponto.posicionamento.x
    dy = celula.posicionamento.y - ponto.posicionamento.y
    dz = celula.posicionamento.z - ponto.posicionamento.z
    ponto.face = []
    let fzy = ponto.face[0]
    let fzx = ponto.face[1]
    let fxy = ponto.face[2]

    if ( dx < 0 ) { fzy = ( 'zyf' ) } else { fzy = ( 'zyi' ) }
    if ( dy < 0 ) { fzx = ( 'zxf' ) } else { fzx = ( 'zxi' ) }
    if ( dz < 0 ) { fxy = ( 'xyf' ) } else { fxy = ( 'xyi' ) }

    if ( fzy == ( 'zyi' ) && fzx == ( 'zxf' ) && fxy == ( 'xyi' ) ) { v1 = ponto }
    if ( fzy == ( 'zyf' ) && fzx == ( 'zxf' ) && fxy == ( 'xyi' ) ) { v2 = ponto }
    if ( fzy == ( 'zyf' ) && fzx == ( 'zxi' ) && fxy == ( 'xyi' ) ) { v3 = ponto }
    if ( fzy == ( 'zyi' ) && fzx == ( 'zxi' ) && fxy == ( 'xyi' ) ) { v4 = ponto }
    if ( fzy == ( 'zyi' ) && fzx == ( 'zxf' ) && fxy == ( 'xyf' ) ) { v5 = ponto }
    if ( fzy == ( 'zyf' ) && fzx == ( 'zxf' ) && fxy == ( 'xyf' ) ) { v6 = ponto }
    if ( fzy == ( 'zyf' ) && fzx == ( 'zxi' ) && fxy == ( 'xyf' ) ) { v7 = ponto }
    if ( fzy == ( 'zyi' ) && fzx == ( 'zxi' ) && fxy == ( 'xyf' ) ) { v8 = ponto }
}
let vertices = [ v1, v2, v3, v4, v5, v6, v7, v8 ]

// Dividir os vertices por coordenada
let vX, vY, vZ, vU, vV, vW
vX = []; vY = []; vZ = []; vU = []; vV = []; vW = []
for ( let i in vertices ) {
    vX[i] = vertices[i].posicionamento.x
    vY[i] = vertices[i].posicionamento.y
    vZ[i] = vertices[i].posicionamento.z

```

```

    if ( vertices[i].vetor === null ) {} else {
        vU[i] = vertices[i].vetor.u
        vV[i] = vertices[i].vetor.v
        vW[i] = vertices[i].vetor.w
    }
}

// adicionar coordenadas de vertices e vetores nas celulas
if ( vertices[0].vetor === null ) {
    celula.vertices = { x: vX, y: vY, z: vZ }
    celula.vetores = null
} else {
    celula.vertices = { x: vX, y: vY, z: vZ }
    celula.vetores = { u: vU, v: vV, w: vW }
}
}

console.log( "... os vertices das Celulas foram mapeados." )
return celulas
}

export default mapearVerticesCelulas

```

Fonte: Elaborado pelo autor, 2021.

Quadro 15 - Função refinamentoCelulas.

refinamentoCelulas

```

import parametrizarCores from '../parametrizar cores/parametrizarCores.js'
import * as THREE from '../three.module.js'

function refinamentoCelulas( esferas, celulas, refinamento, matrizes, malha ) {
    // organizando variaveis...
    let dx, dy, dz
    dx = malha.dimensoes.discretas.x
    dy = malha.dimensoes.discretas.y

```

```

dz = malha.dimensoes.discretas.z
let ux, uy, uz
ux = malha.unidades.x
uy = malha.unidades.y
uz = malha.unidades.z
// ... variaveis organizadas.

console.log( "Refinamento das Celulas..." )
//ajustando variaveis - inicio
let unidades = { x: refinamento.y, y: refinamento.x, z: refinamento.z }
let dimensao = { x: 1/unidades.x, y: 1/unidades.y, z: 1/unidades.z }
//ajustando variaveis - final
let malhaCelulas = []
for ( let celula of celulas ) {
  for ( let i = -unidades.x; i <= unidades.x; i++ ) {
    for ( let j = -unidades.y; j <= unidades.y; j++ ) {
      for ( let k = -unidades.z; k <= unidades.z; k++ ) {
        let  $\xi$ ,  $\eta$ ,  $\zeta$ 
         $\xi$  = i * dimensao.x
         $\eta$  = j * dimensao.y
         $\zeta$  = k * dimensao.z
        let vertices, vetores
        vertices = celula.vertices
        vetores = celula.vetores
        let malhaCelula = mapearCoordenadas(  $\xi$ ,  $\eta$ ,  $\zeta$ , vertices, vetores )
        if ( matrizes == null || malhaCelula.vetor == null ) {} else {
          let min, max, ut
          min = matrizes.vetores.min
          max = matrizes.vetores.max
          ut = malhaCelula.vetor.abs
          malhaCelula.cor = parametrizarCores( min, max, ut )

          if (  $\xi$  == 0 &&  $\eta$  == 0 &&  $\zeta$  == 0 ) {

```

```

        let cor = malhaCelula.cor
        let pos = malhaCelula.posicionamento
        desenharCelula( esferas, pos, dx, dy, dz, cor )
    }
}
malhaCelulas.push( malhaCelula )
}
}
}
}
}
console.log( "... as Celulas foram refinadas." )
return malhaCelulas
}
export default refinamentoCelulas

function mapearCoordenadas(  $\xi$ ,  $\eta$ ,  $\zeta$ , vertices, vetores ) {
    //console.log( "Mapeando das coordenadas locais..." )
    let N, x, y, z
    x = y = z = 0
    let u, v, w, abs, unitario
    u = v = w = 0
    N = Ns(  $\xi$ ,  $\eta$ ,  $\zeta$  )
    for ( let i in N ) {
        x += vertices.x[i] * N[i]
        y += vertices.y[i] * N[i]
        z += vertices.z[i] * N[i]
        if ( vetores == null ) {} else {
            u += vetores.u[i] * N[i]
            v += vetores.v[i] * N[i]
            w += vetores.w[i] * N[i]
            abs = Math.sqrt( u * u + v * v + w * w )
            if ( abs != 0 ) { unitario = { u: u/abs, v: v/abs, w: w/abs } } else { unitario = { u: 0, v:
0, w: 0 } }

```

```

    }
  }
  let mapeamento
  if ( vetores == null ) {
    mapeamento = { vetor: null, posicionamento: { x, y, z } }
  } else {
    mapeamento = { vetor: { u: u, v: v, w: w, abs: abs, unitario: unitario }, posicionamento:
    { x, y, z }, cor: null }
  }

  //console.log( "... as coordenadas locais foram Mapeadas." )
  return mapeamento
}

function Ns(  $\xi$ ,  $\eta$ ,  $\zeta$  ) {
  //console.log( "Moldando as funções de forma..." )
  let N1, N2, N3, N4, N5, N6, N7, N8
  N1 = Nf(  $\xi$  ) * Ni(  $\eta$  ) * Ni(  $\zeta$  )
  N2 = Nf(  $\xi$  ) * Nf(  $\eta$  ) * Ni(  $\zeta$  )
  N3 = Ni(  $\xi$  ) * Nf(  $\eta$  ) * Ni(  $\zeta$  )
  N4 = Ni(  $\xi$  ) * Ni(  $\eta$  ) * Ni(  $\zeta$  )

  N5 = Nf(  $\xi$  ) * Ni(  $\eta$  ) * Nf(  $\zeta$  )
  N6 = Nf(  $\xi$  ) * Nf(  $\eta$  ) * Nf(  $\zeta$  )
  N7 = Ni(  $\xi$  ) * Nf(  $\eta$  ) * Nf(  $\zeta$  )
  N8 = Ni(  $\xi$  ) * Ni(  $\eta$  ) * Nf(  $\zeta$  )
  //console.log( "... as funções de forma foram Moldadas." )
  return [ N1, N2, N3, N4, N5, N6, N7, N8 ]

  function Ni( t ) { return ( (1/2)*( 1 - t ) ) }
  function Nf( t ) { return ( (1/2)*( 1 + t ) ) }
}

```

```

function desenharCelula( esferas, pos, dx, dy, dz, cor ) {
    let material, geometria, mesh
    let r, g, b, t
    r = Math.floor( cor.r * 255 )
    g = Math.floor( cor.g * 255 )
    b = Math.floor( cor.b * 255 )
    t = cor.t
    material = new THREE.MeshPhysicalMaterial({
        color: `rgb(${r},${g},${b})`,
        wireframe: false,
        metalness: 0,
        roughness: 0,
        transparent: true,
        reflectivity: 1,
        refractionRatio: 1,
        opacity: .8
    })
    //geometria = new THREE.BoxGeometry( raio, raio, raio )
    geometria = new THREE.SphereGeometry( .5 , 40, 40 )

    mesh = new THREE.Mesh( geometria, material )
    mesh.scale.set( dx*.5, dy*.5, dz*.5 )
    mesh.position.set( pos.x, pos.y, pos.z )
    mesh.geometry.computeVertexNormals()
    mesh.castShadow = true;
    esferas.add(mesh)
}

```

Fonte: Elaborado pelo autor, 2021.

Quadro 16 - Função desenharGradiente.

desenharGradiente

```

import * as THREE from '../three.module.js'
import parametrizarCores from '../parametrizar cores/parametrizarCores.js'

```

```
import initCelulas from './initCelulas.js'
import desenharCelulas from './desenharCelulas.js'

function desenharGradiente( cena, malhaCelulas, malha, matrizes, raio, visibilidade ) {
  console.log( 'Desenhando gradiente...' )
  let v1 = visibilidade.v1
  let v2 = visibilidade.v2
  let v3 = visibilidade.v3
  let v4 = visibilidade.v4
  let v5 = visibilidade.v5
  let v6 = visibilidade.v6
  // organizando variaveis...
  let dx, dy, dz
  dx = malha.dimensoes.discretas.x
  dy = malha.dimensoes.discretas.y
  dz = malha.dimensoes.discretas.z

  let dm = ( dx >= dy ? ( dy >= dz ? dz : dy ) : ( dx ) )

  let posP1 = []
  let cP1 = []
  let posP2 = []
  let cP2 = []
  let posP3 = []
  let cP3 = []
  let posP4 = []
  let cP4 = []
  let posP5 = []
  let cP5 = []
  let posP6 = []
  let cP6 = []
```

```
let pontosMaterial, pontosGeometria
```

```
//if ( malhaCelulas.length == 27 ) {} else {  
  let posicionamentoPontos = new Float32Array( malhaCelulas.length * 3 )  
  let coresPontos = new Uint8Array( malhaCelulas.length * 3 )  
  for ( let i in malhaCelulas ) {  
    posicionamentoPontos[ ( 3 * i ) + 0 ] = malhaCelulas[i].posicionamento.x  
    posicionamentoPontos[ ( 3 * i ) + 1 ] = malhaCelulas[i].posicionamento.y  
    posicionamentoPontos[ ( 3 * i ) + 2 ] = malhaCelulas[i].posicionamento.z  
  
    if ( malhaCelulas[i].vetor == null ) {  
      let r, g, b  
      r = 217  
      g = 164  
      b = 84  
  
      coresPontos[ ( 3 * i ) + 0 ] = Math.floor( r )  
      coresPontos[ ( 3 * i ) + 1 ] = Math.floor( g )  
      coresPontos[ ( 3 * i ) + 2 ] = Math.floor( b )  
  
      pontosMaterial = new THREE.PointsMaterial({  
        vertexColors: THREE.VertexColors,  
        onBeforeCompile: shader => {  
          shader.fragmentShader = shader.fragmentShader.replace(  
            `#include <clipping_planes_fragment>`,  
            `,  
            if (length(gl_PointCoord - 0.5) > 0.5 ) discard; // make points round  
            #include <clipping_planes_fragment>  
            `,  
          );  
        },  
        size: dm*.03,  
        //blending: THREE.AdditiveBlending,  
        transparent: true,
```

```
        opacity: .5
    })
} else {

    let min, max, ut, cor
    min = matrizes.vetores.min
    max = matrizes.vetores.max
    ut = malhaCelulas[i].vetor.abs
    cor = parametrizarCores( min, max, ut )

    let t = cor.t
    let intervalo = 1 / 6

    if ( t >= 0 && t < 1 * intervalo ) {
        posP1.push( malhaCelulas[i].posicionamento.x )
        posP1.push( malhaCelulas[i].posicionamento.y )
        posP1.push( malhaCelulas[i].posicionamento.z )
        cP1.push( cor.r * 255 )
        cP1.push( cor.g * 255 )
        cP1.push( cor.b * 255 )
    }
    if ( t >= 1 * intervalo && t < 2 * intervalo ) {
        posP2.push( malhaCelulas[i].posicionamento.x )
        posP2.push( malhaCelulas[i].posicionamento.y )
        posP2.push( malhaCelulas[i].posicionamento.z )
        cP2.push( cor.r * 255 )
        cP2.push( cor.g * 255 )
        cP2.push( cor.b * 255 )
    }
    if ( t >= 2 * intervalo && t < 3 * intervalo ) {
        posP3.push( malhaCelulas[i].posicionamento.x )
        posP3.push( malhaCelulas[i].posicionamento.y )
```

```
posP3.push( malhaCelulas[i].posicionamento.z )
cP3.push( cor.r * 255 )
cP3.push( cor.g * 255 )
cP3.push( cor.b * 255 )

}

if ( t >= 3 * intervalo && t <= 4 * intervalo ) {
    posP4.push( malhaCelulas[i].posicionamento.x )
    posP4.push( malhaCelulas[i].posicionamento.y )
    posP4.push( malhaCelulas[i].posicionamento.z )
    cP4.push( cor.r * 255 )
    cP4.push( cor.g * 255 )
    cP4.push( cor.b * 255 )

}

if ( t > 4 * intervalo && t <= 5 * intervalo ) {
    posP5.push( malhaCelulas[i].posicionamento.x )
    posP5.push( malhaCelulas[i].posicionamento.y )
    posP5.push( malhaCelulas[i].posicionamento.z )
    cP5.push( cor.r * 255 )
    cP5.push( cor.g * 255 )
    cP5.push( cor.b * 255 )

}

if ( t > 5 * intervalo && t <= 6 * intervalo ) {
    posP6.push( malhaCelulas[i].posicionamento.x )
    posP6.push( malhaCelulas[i].posicionamento.y )
    posP6.push( malhaCelulas[i].posicionamento.z )
    cP6.push( cor.r * 255 )
    cP6.push( cor.g * 255 )
    cP6.push( cor.b * 255 )

}

}
```

```

    }

}

if ( malhaCelulas[0].vetor == null ) {
    pontosGeometria = new THREE.BufferGeometry()
    pontosGeometria.setAttribute( 'position', new
    THREE.BufferAttribute( posicionamentoPontos, 3 ) )
    pontosGeometria.setAttribute( 'color', new THREE.BufferAttribute( coresPontos, 3,
    true ) )
    pontosGeometria.setDrawRange( 0, malhaCelulas.length )
    let mesh = new THREE.Points( pontosGeometria, pontosMaterial )
    cena.add(mesh)

} else {
    let intervalo = 1 / 6
    let pg1, pg2, pg3, pg4, pg5, pg6
    let pm1, pm2, pm3, pm4, pm5, pm6
    pg1 = new THREE.BufferGeometry()
    pg1.setAttribute( 'position', new THREE.Float32BufferAttribute( posP1, 3 ) )
    pg1.setAttribute( 'color', new THREE.Uint8BufferAttribute( cP1, 3, true ) )
    pg1.setDrawRange( 0, posP1.length )

    pg2 = new THREE.BufferGeometry()
    pg2.setAttribute( 'position', new THREE.Float32BufferAttribute( posP2, 3 ) )
    pg2.setAttribute( 'color', new THREE.Uint8BufferAttribute( cP2, 3, true ) )
    pg2.setDrawRange( 0, posP2.length )

    pg3 = new THREE.BufferGeometry()
    pg3.setAttribute( 'position', new THREE.Float32BufferAttribute( posP3, 3 ) )
    pg3.setAttribute( 'color', new THREE.Uint8BufferAttribute( cP3, 3, true ) )
    pg3.setDrawRange( 0, posP3.length )

```

```

pg4 = new THREE.BufferGeometry()
pg4.setAttribute( 'position', new THREE.Float32BufferAttribute( posP4, 3 ) )
pg4.setAttribute( 'color', new THREE.Uint8BufferAttribute( cP4, 3, true ) )
pg4.setDrawRange( 0, posP4.length )

pg5 = new THREE.BufferGeometry()
pg5.setAttribute( 'position', new THREE.Float32BufferAttribute( posP5, 3 ) )
pg5.setAttribute( 'color', new THREE.Uint8BufferAttribute( cP5, 3, true ) )
pg5.setDrawRange( 0, posP5.length )

pg6 = new THREE.BufferGeometry()
pg6.setAttribute( 'position', new THREE.Float32BufferAttribute( posP6, 3 ) )
pg6.setAttribute( 'color', new THREE.Uint8BufferAttribute( cP6, 3, true ) )
pg6.setDrawRange( 0, posP6.length )

pm1 = new THREE.PointsMaterial({
  vertexColors: THREE.VertexColors,
  onBeforeCompile: shader => {
    shader.fragmentShader = shader.fragmentShader.replace(
      `#include <clipping_planes_fragment>`,
      `
      if (length(gl_PointCoord - 0.5) > 0.5 ) discard; // make points round
      #include <clipping_planes_fragment>
      `
    );
  },
  size: dm*raio,
  //blending: THREE.AdditiveBlending,
  transparent: true,
  opacity: ( intervalo * 1 ),
  visible: vl
})
pm2 = new THREE.PointsMaterial({

```

```

vertexColors: THREE.VertexColors,
onBeforeCompile: shader => {
  shader.fragmentShader = shader.fragmentShader.replace(
    `#include <clipping_planes_fragment>`,
    `
    if (length(gl_PointCoord - 0.5) > 0.5 ) discard; // make points round
    #include <clipping_planes_fragment>
    `
  );
},
size: dm*raio,
//blending: THREE.AdditiveBlending,
transparent: true,
opacity: ( intervalo * 2 ),
visible: v2
})

pm3 = new THREE.PointsMaterial({
  vertexColors: THREE.VertexColors,
  onBeforeCompile: shader => {
    shader.fragmentShader = shader.fragmentShader.replace(
      `#include <clipping_planes_fragment>`,
      `
      if (length(gl_PointCoord - 0.5) > 0.5 ) discard; // make points round
      #include <clipping_planes_fragment>
      `
    );
  },
  size: dm*raio,
  //blending: THREE.AdditiveBlending,
  transparent: true,
  opacity: ( intervalo * 3 ),
  visible: v3
})

```

```

pm4 = new THREE.PointsMaterial({
  vertexColors: THREE.VertexColors,
  onBeforeCompile: shader => {
    shader.fragmentShader = shader.fragmentShader.replace(
      `#include <clipping_planes_fragment>`,
      `
      if (length(gl_PointCoord - 0.5) > 0.5 ) discard; // make points round
      #include <clipping_planes_fragment>
      `
    );
  },
  size: dm*raio,
  //blending: THREE.AdditiveBlending,
  transparent: true,
  opacity: ( intervalo * 4 ),
  visible: v4
})

pm5 = new THREE.PointsMaterial({
  vertexColors: THREE.VertexColors,
  onBeforeCompile: shader => {
    shader.fragmentShader = shader.fragmentShader.replace(
      `#include <clipping_planes_fragment>`,
      `
      if (length(gl_PointCoord - 0.5) > 0.5 ) discard; // make points round
      #include <clipping_planes_fragment>
      `
    );
  },
  size: dm*raio,
  //blending: THREE.AdditiveBlending,
  transparent: true,
  opacity: ( intervalo * 5 ),
  visible: v5
}

```

```

    })
    pm6 = new THREE.PointsMaterial({
      vertexColors: THREE.VertexColors,
      onBeforeCompile: shader => {
        shader.fragmentShader = shader.fragmentShader.replace(
          `#include <clipping_planes_fragment>`,
          `
          if (length(gl_PointCoord - 0.5) > 0.5 ) discard; // make points round
          #include <clipping_planes_fragment>
          `
        );
      },
      size: dm*raio,
      //blending: THREE.AdditiveBlending,
      transparent: true,
      opacity: ( intervalo * 6 ),
      visible: v6
    })

```

```

let mesh1 = new THREE.Points( pg1, pm1 )
let mesh2 = new THREE.Points( pg2, pm2 )
let mesh3 = new THREE.Points( pg3, pm3 )
let mesh4 = new THREE.Points( pg4, pm4 )
let mesh5 = new THREE.Points( pg5, pm5 )
let mesh6 = new THREE.Points( pg6, pm6 )

```

```

cena.add(mesh1)
cena.add(mesh2)
cena.add(mesh3)
cena.add(mesh4)
cena.add(mesh5)
cena.add(mesh6)

```

```
}

// }

console.log( '... Gradiente desenhado' )
}

export default desenharGradiente
```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE D - MALHA

Quadro 17 - Função initGeometria.

initGeometria
<pre>function initGeometria(parametros) { let Dx, Dy, Dz, Tx, Ty, Tz Dx = parametros.dimensoes.x Dy = parametros.dimensoes.y Dz = parametros.dimensoes.z Tx = parametros.discretizador.x Ty = parametros.discretizador.y Tz = parametros.discretizador.z let lx, ly, lz, dm dm = 10 lx = Dx * dm ly = Dy * dm lz = Dz * dm let dx, dy, dz, dimensoes, unidades unidades = { x: Tx, y: Ty, z: Tz } dx = lx / Tx dy = ly / Ty dz = lz / Tz dimensoes = { totais: { x: lx, y: ly, z: lz }, discretas: { x: dx, y: dy, z: dz } } let malha = { unidades, dimensoes } return malha } export default initGeometria</pre>

Fonte: Elaborado pelo autor, 2021.

Quadro 18 - Função posicionarMalha.

posicionarMalha

```
import * as THREE from '../three.module.js'

function posicionarMalha( pontos, estrutura, parametros ) {
  console.log( 'Posicionando Malha' )
  let minX, maxX, minY, maxY, minZ, maxZ
  let x, y, z
  x = null
  y = null
  z = null

  x = []
  y = []
  z = []
  for ( let ponto of pontos ) {
    x.push( ponto.posicionamento.x )
    y.push( ponto.posicionamento.y )
    z.push( ponto.posicionamento.z )
  }
  minX = Math.min( ...x )
  minY = Math.min( ...y )
  minZ = Math.min( ...z )

  maxX = Math.max( ...x )
  maxY = Math.max( ...y )
  maxZ = Math.max( ...z )

  parametros.tela = { x: maxX - minX, y: maxY - minY, z: maxZ - minZ }

  let xmed, ymed, zmed
  xmed = ( ( ( maxX - minX )/2 ) + minX )
  ymed = ( ( ( maxY - minY )/2 ) + minY )
  zmed = ( ( ( maxZ - minZ )/2 ) + minZ )
```

```

    estrutura.position.set( -xmed, -ymed, -zmed )
    let dx, dy, dz, dm

    dm = (( parametros.tela.x >= parametros.tela.y ? ( parametros.tela.y >= parametros.tela.z ?
parametros.tela.z : parametros.tela.y ) : parametros.tela.x ) / 2)
    if ( dm > 2 ) {
        dx = dy = dz = dm
    } else {
        dx = dy = dz = 2
    }

    desenharLimites( (minX - dx), (maxX + dx), (minY - dy), (maxY + dy), (minZ - dz),
(maxZ + dz), estrutura, parametros.tela )
}

function desenharLimites( minX, maxX, minY, maxY, minZ, maxZ, estrutura, tela ) {
    let dx, dy, dz, dm
    dm = (( tela.x >= tela.y ? ( tela.y >= tela.z ? tela.z : tela.y ) : tela.x ) / 5)
    if ( dm > 2 ) {
        dx = dy = dz = dm
    } else {
        dx = dy = dz = 2
    }
}
/*
    dx = tela.x / 10
    dy = tela.y / 10
    dz = tela.z / 10
*/
let limitesPos = [
    //4-3
    minX, minY, minZ, maxX, minY, minZ,
    //4-1

```

```

minX, minY, minZ, minX, minY, maxZ,
//1-2
minX, minY, maxZ, maxX, minY, maxZ,
//2-3
maxX, minY, maxZ, maxX, minY, minZ,

//8-7
minX, maxY, minZ, maxX, maxY, minZ,
//8-5
minX, maxY, minZ, minX, maxY, maxZ,
//5-6
minX, maxY, maxZ, maxX, maxY, maxZ,
//6-7
maxX, maxY, maxZ, maxX, maxY, minZ,

//7-3
maxX, maxY, minZ, maxX, minY, minZ,
//6-2
maxX, maxY, maxZ, maxX, minY, maxZ,
//5-1
minX, maxY, maxZ, minX, minY, maxZ,
//8-4
minX, maxY, minZ, minX, minY, minZ
]
let edgesPos = [
//4 - 3
minX, minY, minZ, ( minX + dx ), ( minY + 0 ), ( minZ + 0 ),
//4 - 8
minX, minY, minZ, ( minX + 0 ), ( minY + dy ), ( minZ + 0 ),
//4 - 1
minX, minY, minZ, ( minX + 0 ), ( minY + 0 ), ( minZ + dz ),

//3 - 4

```

```

maxX, minY, minZ, ( maxX - dx ), ( minY + 0 ), ( minZ + 0 ),
//3 - 7
maxX, minY, minZ, ( maxX + 0 ), ( minY + dy ), ( minZ + 0 ),
//3 - 2
maxX, minY, minZ, ( maxX + 0 ), ( minY + 0 ), ( minZ + dz ),

//2 - 1
maxX, minY, maxZ, ( maxX - dx ), ( minY + 0 ), ( maxZ + 0 ),
//2 - 6
maxX, minY, maxZ, ( maxX - 0 ), ( minY + dy ), ( maxZ + 0 ),
//2 - 3
maxX, minY, maxZ, ( maxX - 0 ), ( minY + 0 ), ( maxZ - dz ),

//1 - 2
minX, minY, maxZ, ( minX + dx ), ( minY + 0 ), ( maxZ + 0 ),
//1 - 5
minX, minY, maxZ, ( minX + 0 ), ( minY + dy ), ( maxZ + 0 ),
//1 - 4
minX, minY, maxZ, ( minX + 0 ), ( minY + 0 ), ( maxZ - dz ),

//8 - 7
minX, maxY, minZ, ( minX + dx ), ( maxY + 0 ), ( minZ + 0 ),
//8 - 4
minX, maxY, minZ, ( minX + 0 ), ( maxY - dy ), ( minZ + 0 ),
//8 - 5
minX, maxY, minZ, ( minX + 0 ), ( maxY - 0 ), ( minZ + dz ),

//7 - 8
maxX, maxY, minZ, ( maxX - dx ), ( maxY + 0 ), ( minZ + 0 ),
//7 - 3
maxX, maxY, minZ, ( maxX - 0 ), ( maxY - dy ), ( minZ + 0 ),
//7 - 6
maxX, maxY, minZ, ( maxX - 0 ), ( maxY - 0 ), ( minZ + dz ),

```

```

//6 - 5
maxX, maxY, maxZ, ( maxX - dx ), ( maxY + 0 ), ( maxZ + 0 ),
//6 - 2
maxX, maxY, maxZ, ( maxX - 0 ), ( maxY - dy ), ( maxZ + 0 ),
//6 - 7
maxX, maxY, maxZ, ( maxX - 0 ), ( maxY - 0 ), ( maxZ - dz ),

//5 - 6
minX, maxY, maxZ, ( minX + dx ), ( maxY + 0 ), ( maxZ + 0 ),
//5 - 1
minX, maxY, maxZ, ( minX + 0 ), ( maxY - dy ), ( maxZ + 0 ),
//5 - 8
minX, maxY, maxZ, ( minX + 0 ), ( maxY + 0 ), ( maxZ - dz ),
]

let limitesGeometria = new THREE.BufferGeometry()
limitesGeometria.setAttribute( 'position', new THREE.Float32BufferAttribute( limitesPos,
3 ) )

let limitesMaterial = new THREE.LineBasicMaterial({
  color: 'rgb( 0, 0, 0 )',
  //vertexColors: true,
  blending: THREE.AdditiveBlending,
  transparent: true,
  opacity: 1,
  visible: false
})

let limitesMesh = new THREE.LineSegments( limitesGeometria, limitesMaterial )
estrutura.add( limitesMesh )

let edgesGeometria = new THREE.BufferGeometry()

```

```

edgesGeometria.setAttribute( 'position', new THREE.Float32BufferAttribute( edgesPos,
3 ) )

let edgesMaterial = new THREE.LineBasicMaterial({
  color: 'rgb( 255, 255, 255 )',
  //vertexColors: true,
  blending: THREE.AdditiveBlending,
  //transparent: true,
  opacity: 1,
})

let edgesMesh = new THREE.LineSegments( edgesGeometria, edgesMaterial )
estrutura.add( edgesMesh )
}
export default posicionarMalha

```

Fonte: Elaborado pelo autor, 2021.

Quadro 19 - Função parametrizarCores.

parametrizarCores

```

function parametrizarCores( min, max, ut ) {
  let t = 1 - ( ( max - ut ) / ( max + min ) )

  let azul    = { r: 0, g: 0, b: 1 }
  let ciano   = { r: 0, g: 1, b: 1 }
  let verde   = { r: 0, g: 1, b: 0 }
  let amarelo = { r: 1, g: 1, b: 0 }
  let vermelho = { r: 1, g: 0, b: 0 }
  let magenta = { r: 1, g: 0, b: 1 }
  let intervalo = 1 / 6
  let cor

  if ( t >= 0 && t < 1 * intervalo ) {
    let pos = (t - 0 * intervalo) / intervalo
    cor = interpolarCor(azul, magenta, pos, t)
  }
}

```

```

    }
    if ( t >= 1 * intervalo && t < 2 * intervalo ) {
        let pos = (t - 1 * intervalo) / intervalo
        cor = interpolarCor(azul, ciano, pos, t)
    }
    if ( t >= 2 * intervalo && t < 3 * intervalo ) {
        let pos = (t - 2 * intervalo) / intervalo
        cor = interpolarCor(ciano, verde, pos, t)
    }
    if ( t >= 3 * intervalo && t <= 4 * intervalo ) {
        let pos = (t - 3 * intervalo) / intervalo
        cor = interpolarCor(verde, amarelo, pos, t)
    }
    if ( t > 4 * intervalo && t <= 5 * intervalo ) {
        let pos = (t - 4 * intervalo) / intervalo
        cor = interpolarCor(amarelo, vermelho, pos, t)
    }
    if ( t > 5 * intervalo && t <= 6 * intervalo ) {
        let pos = (t - 5 * intervalo) / intervalo
        cor = interpolarCor(vermelho, magenta, pos, t)
    }
    return cor
}

function interpolarCor( cor1, cor2, pos, t ) {
    let r = cor1.r + pos * (cor2.r - cor1.r)
    let g = cor1.g + pos * (cor2.g - cor1.g)
    let b = cor1.b + pos * (cor2.b - cor1.b)

    return { r, g, b, t }
}

```

```
export default parametrizarCores
```


APÊNDICE E - AMBIENTE 3D

Quadro 20 - Função iniCanvas.

```
initCanvas
```

```
import * as THREE from '../three.module.js'
function initCanvas() {
  let canvas = document.getElementById( 'canvas' )
  console.log( '2.1 - instanciando Canvas' )
  return canvas
}
export default initCanvas
```

Fonte: Elaborado pelo autor, 2021.

Quadro 21 - Função iniCamera.

```
initCamera
```

```
import * as THREE from '../three.module.js'
import { OrbitControls } from '../OrbitControls.js'

function initCamera( parametros ) {
  let camera = new THREE.PerspectiveCamera( 100, window.innerWidth /
window.innerHeight, 1, 4000 )
  let cameraZ
  if ( parametros.mode === 'manual' ) {
    cameraZ = (parametros.tela.x >= parametros.tela.y
      ? ( parametros.tela.x >= parametros.tela.z ? parametros.tela.x : (parametros.tela.z))
      : parametros.tela.y)
    camera.position.z = (cameraZ)* 1.2 + 5
  } else {
    cameraZ = (parametros.dimensoes.x > parametros.dimensoes.y
      ? ( parametros.dimensoes.x > parametros.dimensoes.z ? parametros.dimensoes.x :
(parametros.dimensoes.z))
      : parametros.dimensoes.y)
    camera.position.z = (cameraZ)* 4 + 45
  }
}
```

```
let control = new OrbitControls( camera, canvas )
control.minDistance = 10
control.maxDistance = 1000
console.log( '2.2 - instanciando Camera' )
return { camera, cameraZ }
}
export default initCamera
```

Fonte: Elaborado pelo autor, 2021.

Quadro 22 - Função iniCena.

initCena

```
import * as THREE from '../three.module.js'
function initCena() {
  let cena = new THREE.Scene()
  console.log( '2.3 - instanciando Cena' )
  return cena
}
export default initCena
```

Fonte: Elaborado pelo autor, 2021.

Quadro 23 - Função iniIluminar.

iniIluminar

```
import * as THREE from '../three.module.js'
function iniIluminar( cena ) {
  let light = new THREE.HemisphereLight(0xffffff, 0xffffff, .5)

  let backLight = new THREE.DirectionalLight(0xffffff, .2);
  backLight.position.set(-10, 20, 50);
  backLight.castShadow = true;

  let lightProbe = new THREE.LightProbe();
```

```
const dirLight = new THREE.DirectionalLight( 'white', .2 );
dirLight.position.set(10, 10, 10);
dirLight.castShadow = true;
dirLight.shadow.camera.top = 2;
dirLight.shadow.camera.bottom = - 2;
dirLight.shadow.camera.left = - 2;
dirLight.shadow.camera.right = 2;
dirLight.shadow.camera.near = 0.1;
dirLight.shadow.camera.far = 40;

const light2 = new THREE.PointLight( 0xff2200, 0.2 );
  light2.position.set( - 100, - 100, - 100 );
  cena.add( light2 )

addShadowedLight( 1, 1, 1, 0xffffffff, .1 )
addShadowedLight( 0.5, 1, - 1, 0xffffffff, .5 )

function addShadowedLight( x, y, z, color, intensity ) {

  const directionalLight = new THREE.DirectionalLight( color, intensity );
  directionalLight.position.set( x, y, z );
  cena.add( directionalLight );

  directionalLight.castShadow = true;

  const d = 1;
  directionalLight.shadow.camera.left = - d;
  directionalLight.shadow.camera.right = d;
  directionalLight.shadow.camera.top = d;
  directionalLight.shadow.camera.bottom = - d;

  directionalLight.shadow.camera.near = 1;
```

```

directionalLight.shadow.camera.far = 4;

directionalLight.shadow.mapSize.width = 1024;
directionalLight.shadow.mapSize.height = 1024;

directionalLight.shadow.bias = - 0.001;

}

cena.add(dirLight);
cena.add(lightProbe );
cena.add(backLight);
cena.add(light);
}
export default initIlluminar

```

Fonte: Elaborado pelo autor, 2021.

Quadro 24 - Função initRenderer.

```

initRenderer

```

```

import * as THREE from '../three.module.js'
function initRenderer( canvas ) {
  let renderer = new THREE.WebGLRenderer({ canvas, antialias: true, alpha: true })
  renderer.setPixelRatio( window.devicePixelRatio )
  renderer.setSize( window.innerWidth, window.innerHeight )
  renderer.outputEncoding = THREE.sRGBEncoding
  return renderer
}
export default initRenderer

```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE F - INTERFACE

Quadro 25 - Função criarElemento.

criarElemento
<pre>function criarElemento(nome, atributos) { const elemento = document.createElement(nome) const atributosAsArrays = Object.entries(atributos) atributosAsArrays.forEach(([key, valor]) => elemento.setAttribute(key, valor)) return elemento } export default criarElemento</pre>

Fonte: Elaborado pelo autor, 2021.

Quadro 26 - Função navegarPontos.

navegarPontos
<pre>function navegarPontos(target, links) { for (let link of links) { link.a.setAttribute('class', `nav-link`) } target.setAttribute('class', `nav-link nav-link-selected`) } export default navegarPontos</pre>

Fonte: Elaborado pelo autor, 2021.

Quadro 27 - Função initSliders.

initSliders
<pre>import criarElemento from "../criarElemento.js" import { ajustarMalha } from "../../main.js" function initSliders(body, parametros) { let dx, dy, dz, tx, ty, tz, form, cfg form = criarElemento('div', { class: 'form' }) body.append(form) if (parametros.mode == 'manual') {</pre>

```

    cfg = initSlider( form, 5 )
    cfg.addEventListener( 'input', event => {
        parametros.cfg = event.target.value;
        ajustarMalha()
    } )
} else {
    dx = initSlider( form, 10 )
    dy = initSlider( form, 10 )
    dz = initSlider( form, 10 )

    tx = initSlider( form, 4 )
    ty = initSlider( form, 4 )
    tz = initSlider( form, 4 )

    dx.addEventListener( 'input', event => { parametros.dimensoes.x = event.target.value;
ajustarMalha() } )
    dy.addEventListener( 'input', event => { parametros.dimensoes.y = event.target.value;
ajustarMalha() } )
    dz.addEventListener( 'input', event => { parametros.dimensoes.z = event.target.value;
ajustarMalha() } )
    tx.addEventListener( 'input', event => { parametros.discretizador.x = event.target.value;
ajustarMalha() } )
    ty.addEventListener( 'input', event => { parametros.discretizador.y = event.target.value;
ajustarMalha() } )
    tz.addEventListener( 'input', event => { parametros.discretizador.z = event.target.value;
ajustarMalha() } )
}

}

function initSlider( form, range ) {
    let input, box
    box = criarElemento( 'div', { class: 'box' } )
    input = criarElemento( 'input', {

```

```

    class: 'input',
    type: 'range',
    min: '1',
    max: `${range}`,
    value: '1',
    steps: '1'
  ))
  form.append( box )
  box.append( input )
  return input
}
export default initSliders

```

Fonte: Elaborado pelo autor, 2021.

Quadro 28 - Função initGradienteInputs.

initGradienteInputs

```

import criarElemento from "../criarElemento.js"
import { cfgRefinamentoGradiente, desenharMalhaCalculada } from "../../main.js"
import navegarPontos from './navegarPontos.js'
function initGradienteInputs( body, parametros, navLinks ) {
  let gradiente = criarElemento( 'div', { class: 'gradiente' } )
  body.append( gradiente )

  let labels = []
  let inputs = []
  for ( let i = 1; i <= 6; i++ ) {
    let gradienteItem, gradienteLabel, gradienteInput

    gradienteItem = criarElemento( 'div', { class: 'gradiente-itens' } )
    gradienteInput = criarElemento( 'input', {
      class: 'gradiente-input',
      type: 'checkbox',
      id: `v${i}`,

```

```
        checked: `checked`
      } )
      gradienteLabel = criarElemento( 'label', {
        for: `v${i}`,
        id: `v${i}`,
        class: `gradiente-label v${i}`
      } )

      gradienteItem.append( gradienteInput )
      gradienteItem.append( gradienteLabel )
      gradiente.append( gradienteItem )

      labels.push( gradienteLabel )
      inputs.push( gradienteInput )
    }

    let v1, v2, v3, v4, v5, v6
    v1 = inputs[0]
    v2 = inputs[1]
    v3 = inputs[2]
    v4 = inputs[3]
    v5 = inputs[4]
    v6 = inputs[5]

    v1.addEventListener( 'input', event => {
      parametros.visibilidade.v1 = event.target.checked;
      cfgRefinamentoGradiente();
      desenharMalhaCalculada();
      navegarPontos( navLinks[2].a, navLinks )
    } )

    v2.addEventListener( 'input', event => {
      parametros.visibilidade.v2 = event.target.checked;
```

```
    cfgRefinamentoGradiente();
    desenharMalhaCalculada();
    navegarPontos( navLinks[2].a, navLinks )
  } )

v3.addEventListener( 'input', event => {
  parametros.visibilidade.v3 = event.target.checked;
  cfgRefinamentoGradiente();
  desenharMalhaCalculada();
  navegarPontos( navLinks[2].a, navLinks )
} )

v4.addEventListener( 'input', event => {
  parametros.visibilidade.v4 = event.target.checked;
  cfgRefinamentoGradiente();
  desenharMalhaCalculada();
  navegarPontos( navLinks[2].a, navLinks )
} )

v5.addEventListener( 'input', event => {
  parametros.visibilidade.v5 = event.target.checked;
  cfgRefinamentoGradiente();
  desenharMalhaCalculada();
  navegarPontos( navLinks[2].a, navLinks )
} )

v6.addEventListener( 'input', event => {
  parametros.visibilidade.v6 = event.target.checked;
  cfgRefinamentoGradiente();
  desenharMalhaCalculada();
  navegarPontos( navLinks[2].a, navLinks )
} )

return gradiente
```

```
}
```

```
export default initGradienteInputs
```

Fonte: Elaborado pelo autor, 2021.

Quadro 29 - Função initNav.

initNav

```
import criarElemento from "../criarElemento.js"
```

```
function initNav( body, parametros ) {  
  let etapas = parametros.etapas  
  let nav = criarElemento( 'nav', { class: 'nav' } )  
  let navLinks = []  
  let index = 0  
  for ( let item of etapas ) {  
    let navItem, a, span  
    navItem = criarElemento( 'div', { class: `nav-item` } )  
  
    if ( index == 0 ) {  
      a = criarElemento( 'a', { class: `nav-link nav-link-selected`, href: `#` } )  
    } else {  
      a = criarElemento( 'a', { class: `nav-link`, href: `#` } )  
    }  
  
    span = criarElemento( 'span', { class: 'nav-label' } )  
    span.append( etapas[index].titulo )  
  
    navItem.append( a )  
    navItem.append( span )  
    nav.append( navItem )  
  
    navLinks.push( { a, span } )  
    index++  
  }  
}
```

```

    }
    for ( let item of navLinks ) {
        item.a.addEventListener( 'click', event => { navegarPontos( event.target, navLinks ) } )
    }
    body.append( nav )
    return navLinks
}
export default initNav

```

Fonte: Elaborado pelo autor, 2021.

Quadro 30 - Função initNext.

initNext

```

import criarElemento from "../criarElemento.js"
import {
    cfgRefinamentoGradiente,
    cfgRefinamentoFluxo,
    desenharMalhaCalculada,
    calcularMalha,
    desenharFluxo,
    desenharMalhaDeformada
} from "../main.js"
import navegarPontos from './navegarPontos.js'
function initNext( body, matrizes, navLinks, rodar ) {
    let next = criarElemento( 'div', { class: 'next' } )
    next.append( '➤ ' )
    next.addEventListener( 'click', event => executarNext( navLinks, matrizes, rodar ) )
    body.append( next )
    return next
}
function executarNext( navLinks, matrizes, rodar ) {
    if ( matrizes == null ) {
        calcularMalha()
        navegarPontos( navLinks[1].a, navLinks )
    }
}

```

```

    } else {
      if ( rodar.x === .002 ) {
        cfgRefinamentoGradiente()
        desenharMalhaCalculada()
        navegarPontos( navLinks[2].a, navLinks )
      }
      if ( rodar.x === .0025 ) {
        cfgRefinamentoFluxo()
        desenharFluxo()
        navegarPontos( navLinks[3].a, navLinks )
      }
      if ( rodar.x === .003 ) {
        desenharMalhaDeformada()
        navegarPontos( navLinks[4].a, navLinks )
      }
    }
  }
}

export default initNext

```

Fonte: Elaborado pelo autor, 2021.

Quadro 31 - Função initBefore.

initBefore

```

import criarElemento from "./criarElemento.js"
import { ajustarMalha } from "../main.js"
import navegarPontos from './navegarPontos.js'

function initBefore( body, navLinks, rodar ) {
  let before = criarElemento( 'div', { class: 'before' } )
  before.append( '⏪ ' )
  before.addEventListener( 'click', event => executarBefore( event.target, navLinks, rodar ) )
  body.append( before )
  return before
}

```

```
function executarBefore( target, navLinks, rodar ) {  
  if ( rodar.y == 0 || rodar.z == 0 ) {  
    ajustarMalha()  
    navegarPontos( navLinks[0].a, navLinks )  
  }  
}  
export default initBefore
```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE G - MATRIZES

Quadro 32 - Função initMatrizes.

initMatrizes

```
function initMatrizes( pontos, conexoes, f, u, rigidez ) {  
  console.log( conexoes )  
  let numeroPontos = pontos.length  
  let grausLiberdade = ( numeroPontos * 3 )  
  let erro = .001  
  let elementos = initElementos( conexoes )  
  let matrizesIndividuaisRigidez = initMatrizesIndividuais( elementos, grausLiberdade )  
  let matrizGlobalRigidez = initMatrizGlobal( matrizesIndividuaisRigidez, grausLiberdade )  
  let vetorPeso = initVetorPeso( numeroPontos, f )  
  let vetorDeformada = initVetorDeformada( grausLiberdade )  
  vetorDeformada = addCondicoesDeContorno( vetorDeformada, vetorPeso, numeroPontos,  
u )  
  let matrizes  
  intermediar( matrizGlobalRigidez, vetorDeformada, vetorPeso )  
  
  function initElementos( conexoes ) {  
    let elementos = []  
    for ( let conexao of conexoes ) {  
      let a, b, A, E, lo, k, l, m, n  
      if ( conexao.pontoB.id > conexao.pontoA.id ) {  
        a = { id: conexao.pontoA.id, pos: { x: conexao.pontoA.pos[0], y:  
conexao.pontoA.pos[1], z: conexao.pontoA.pos[2] } }  
        b = { id: conexao.pontoB.id, pos: { x: conexao.pontoB.pos[0], y:  
conexao.pontoB.pos[1], z: conexao.pontoB.pos[2] } }  
      } else {  
        a = { id: conexao.pontoB.id, pos: { x: conexao.pontoB.pos[0], y:  
conexao.pontoB.pos[1], z: conexao.pontoB.pos[2] } }  
        b = { id: conexao.pontoA.id, pos: { x: conexao.pontoA.pos[0], y:  
conexao.pontoA.pos[1], z: conexao.pontoA.pos[2] } }  
      }  
    }  
  }  
}
```

```

    }
    lo = Math.sqrt( ( b.pos.x-a.pos.x )**2 + ( b.pos.y-a.pos.y )**2 + ( b.pos.z-
a.pos.z )**2 )
    if ( rigidez == null ) { A = 250; E = 200*(1000) } else { A = rigidez.A; E =
rigidez.E; }

    k = ( ( A * E ) / ( 2 * lo ) )
    //k = 1
    l = ( b.pos.x - a.pos.x ) / lo
    m = ( b.pos.y - a.pos.y ) / lo
    n = ( b.pos.z - a.pos.z ) / lo
    elementos.push( { id: conexao.nc, a, b, k, l, m, n } )
  }
  return elementos
}

function initMatrizesIndividuais( elementos, grausLiberdade ) {
  let matrizesIndividuais = []
  let index = 0
  for ( let elemento of elementos ) {
    index++
    console.log( `A matriz do elemento será construída... %c|${ index } de
${ elementos.length }|`, 'color: white; background-color: black;' )
    let c, s, kij, a, b, k, l, m, n
    k = elemento.k
    l = elemento.l
    m = elemento.m
    n = elemento.n
    a = elemento.a
    b = elemento.b

    kij = {
      k11: k*(+l*l), k12: k*(+l*m), k13: k*(+l*n), k14: k*(-l*l), k15: k*(-l*m), k16: k*(-
l*n),

```

```

    k21: k*(+m*1), k22: k*(+m*m), k23: k*(+m*n), k24: k*(-m*1), k25: k*(-m*m),
k26: k*(-m*n),
    k31: k*(+n*1), k32: k*(+n*m), k33: k*(+n*n), k34: k*(-n*1), k35: k*(-n*m), k36:
k*(-n*n),
    k41: k*(-l*1), k42: k*(-l*m), k43: k*(-l*n), k44: k*(+l*1), k45: k*(+l*m), k46:
k*(+l*n),
    k51: k*(-m*1), k52: k*(-m*m), k53: k*(-m*n), k54: k*(+m*1), k55: k*(+m*m),
k56: k*(+m*n),
    k61: k*(-n*1), k62: k*(-n*m), k63: k*(-n*n), k64: k*(+n*1), k65: k*(+n*m), k66:
k*(+n*n)
}

```

```

let K = initMatrizdeRigidezVazia( grausLiberdade )

```

```

let ua = (a.id * 3)
let ub = (b.id * 3)
let va = (a.id * 3) + 1
let vb = (b.id * 3) + 1
let wa = (a.id * 3) + 2
let wb = (b.id * 3) + 2

```

```

K[ua][ua] = kij.k11
K[ua][va] = kij.k12
K[ua][wa] = kij.k13
K[ua][ub] = kij.k14
K[ua][vb] = kij.k15
K[ua][wb] = kij.k16

```

```

K[va][ua] = kij.k21
K[va][va] = kij.k22
K[va][wa] = kij.k23
K[va][ub] = kij.k24
K[va][vb] = kij.k25
K[va][wb] = kij.k26

```

```
K[wa][ua] = kij.k31
K[wa][va] = kij.k32
K[wa][wa] = kij.k33
K[wa][ub] = kij.k34
K[wa][vb] = kij.k35
K[wa][wb] = kij.k36
```

```
K[ub][ua] = kij.k41
K[ub][va] = kij.k42
K[ub][wa] = kij.k43
K[ub][ub] = kij.k44
K[ub][vb] = kij.k45
K[ub][wb] = kij.k46
```

```
K[vb][ua] = kij.k51
K[vb][va] = kij.k52
K[vb][wa] = kij.k53
K[vb][ub] = kij.k54
K[vb][vb] = kij.k55
K[vb][wb] = kij.k56
```

```
K[wb][ua] = kij.k61
K[wb][va] = kij.k62
K[wb][wa] = kij.k63
K[wb][ub] = kij.k64
K[wb][vb] = kij.k65
K[wb][wb] = kij.k66
```

```
matrizesIndividuais.push( K )
```

```
console.log( `.. a matriz do elemento foi construida. %c${ index } de
${ elementos.length }|`, 'color: white; background-color: black;' )
```

```

    }

    return matrizesIndividuais
}

function initMatrizdeRigidezVazia( grausLiberdade ) {
    let k = []
    for ( let i = 0; i < grausLiberdade; i++ ) {
        k[i] = []
        for ( let j = 0; j < grausLiberdade; j++ ) {
            k[i][j] = 0
        }
    }
    return k
}

function initMatrizGlobal( matrizesIndividuais, grausLiberdade ) {
    let matrizGlobal = initMatrizdeRigidezVazia( grausLiberdade )
    for ( let matrizIndividual of matrizesIndividuais ) {
        for ( let i = 0; i < grausLiberdade; i++ ) {
            for ( let j = 0; j < grausLiberdade; j++ ) {
                matrizGlobal[i][j] += matrizIndividual[i][j]
            }
        }
    }
    return matrizGlobal
}

function initVetorPeso( numeroPontos, f ) {
    let vetorPeso = []
    if ( f == null ) {
        for ( let i = 0; i < numeroPontos; i++ ) {
            vetorPeso[ 3 * i + 0 ] = 0
            vetorPeso[ 3 * i + 1 ] = -180000

```

```

        vetorPeso[ 3 * i + 2 ] = 0
    }
} else {
    for ( let i=0; i < f.length; i++ ) {
        vetorPeso[ 3 * i + 0 ] = f[i].x
        vetorPeso[ 3 * i + 1 ] = f[i].y
        vetorPeso[ 3 * i + 2 ] = f[i].z
    }
}
return vetorPeso
}

function initVetorDeformada( grausLiberdade ) {
    let vetorDeformada = []
    for ( let i=0; i < grausLiberdade; i++ ) {
        vetorDeformada[i] = { valor: 1, identificado: 'indefinido' }
    }
    return vetorDeformada
}

function addCondicoesDeContorno( vetorDeformada, vetorPeso, numeroPontos, u ) {
    // automático
    if ( u == null ) {
        for ( let i = 0; i < 3; i++ ) {
            let index = Math.floor( Math.random() * numeroPontos )
            vetorDeformada[ 3 * index + 0 ] = { valor: 0, identificado: 'true' }
            vetorDeformada[ 3 * index + 1 ] = { valor: 0, identificado: 'true' }
            vetorDeformada[ 3 * index + 2 ] = { valor: 0, identificado: 'true' }

            vetorPeso[ 3 * index + 0 ] = 0
            vetorPeso[ 3 * index + 1 ] = 0
            vetorPeso[ 3 * index + 2 ] = 0
        }
    } else {
        for ( let i = 0; i < u.length; i++ ) {

```

```
if ( u[i].u == 'indefinido' ) {
    vetorDeformada[ 3 * i + 0 ] = {
        id: `${u[i].identificador}|u`,
        valor: 1, identificado: 'indefinido'
    }
} else {
    vetorDeformada[ 3 * i + 0 ] = {
        id: `${u[i].identificador}|u`,
        valor: u[i].u, identificado: 'true'
    }
    vetorPeso[ 3 * i + 0 ] = 0
}

if ( u[i].v == 'indefinido' ) {
    vetorDeformada[ 3 * i + 1 ] = {
        id: `${u[i].identificador}|v`,
        valor: 1, identificado: 'indefinido'
    }
} else {
    vetorDeformada[ 3 * i + 1 ] = {
        id: `${u[i].identificador}|v`,
        valor: u[i].v, identificado: 'true'
    }
    vetorPeso[ 3 * i + 1 ] = 0
}

if ( u[i].w == 'indefinido' ) {
    vetorDeformada[ 3 * i + 2 ] = {
        id: `${u[i].identificador}|w`,
        valor: 1, identificado: 'indefinido'
    }
} else {
    vetorDeformada[ 3 * i + 2 ] = {
        id: `${u[i].identificador}|w`,
        valor: u[i].w, identificado: 'true'
    }
}
```

```
    }
    vetorPeso[ 3 * i + 2 ] = 0
}
/*
if ( u[i].identificado == 'true' ) {
    vetorDeformada[ 3 * i + 0 ] = {
        id: `${u[i].identificador}|u`,
        valor: u[i].u, identificado: 'true'
    }
    vetorDeformada[ 3 * i + 1 ] = {
        id: `${u[i].identificador}|v`,
        valor: u[i].v, identificado: 'true'
    }
    vetorDeformada[ 3 * i + 2 ] = {
        id: `${u[i].identificador}|w`,
        valor: u[i].w, identificado: 'true'
    }
    vetorPeso[ 3 * i + 0 ] = 0
    vetorPeso[ 3 * i + 1 ] = 0
    vetorPeso[ 3 * i + 2 ] = 0
} else {
    vetorDeformada[ 3 * i + 0 ] = {
        id: `${u[i].identificador}|u`,
        valor: 1, identificado: 'indefinido'
    }
    vetorDeformada[ 3 * i + 1 ] = {
        id: `${u[i].identificador}|v`,
        valor: 1, identificado: 'indefinido'
    }
    vetorDeformada[ 3 * i + 2 ] = {
        id: `${u[i].identificador}|w`,
        valor: 1, identificado: 'indefinido'
    }
}
```

```

    }*/
  }
}

return vetorDeformada
}
function verificarConvergencia( K, U, F ) {
  let e = 0
  for ( let i in K ) {
    if ( U[i].identificado == 'indefinido' ) {
      let T = 0
      for ( let j in K ) {
        T += (K[i][j] * U[j].valor )
      }
      e += (T - F[i])
    }
  }
  e = Math.abs( e )
  if ( e >= erro ) {
    console.log(`erro | %c${e}`, 'color: white; background-color: red;')
    calcularDeformacoes( K, U, F )

    return { e, K, U, F }
  } else {
    for ( let i in U ) {
      U[i].identificado = 'true'
    }
    console.log( ` %cDEFORMAÇÃO CALCULADA COM PRECISÃO! | ${e}`, 'color: white; background-color: green;', K,U )

    let vetores = modulosDosVetores( U )
    matrizes = { e, K, U, F, vetores }
  }
}

```

```

    return { e, K, U, F, vetores };
  }
}

function intermediar( K, U, F ) {
  let data = verificarConvergencia( K, U, F )
  let e = data.e
  let k = data.K
  let u = data.U
  let f = data.F
  while ( e >= erro ) {
    data = verificarConvergencia( k, u, f )
    e = data.e
    k = data.K
    u = data.U
    f = data.F
  }
}

function calcularDeformacoes( K, U, F ) {
  for ( let i in K ) {
    if ( U[i].identificado == 'indefinido' ) {
      let T, u
      T = 0
      for ( let j in K ) { T += (K[i][j] * U[j].valor) }
      if ( K[i][i] == 0 ) { u = 0 }
      else { u = (1/K[i][i])*( F[i] - ( T - (K[i][i]*U[i].valor) ) ) }
      U[i].valor = u
    }
  }
}

function modulosDosVetores( U ) {
  console.log( 'Iniciando a interpolação de cores...' )
  console.log( 'Calculando modulos de deformação...' )
  let vetores = []

```

```

let modulos = []
for ( let i = 0 ; i < numeroPontos ; i++ ) {
    let u, v, w, abs
    u = U[ ( i * 3 ) + 0 ].valor
    v = U[ ( i * 3 ) + 1 ].valor
    w = U[ ( i * 3 ) + 2 ].valor
    abs = Math.sqrt( u * u + v * v + w * w )

    let nx, ny, nz
    nx = u / abs
    ny = v / abs
    nz = w / abs

    let dados
    if ( abs === 0 ) { dados = { vetor: { u, v, w, abs, normalizado: { nx: 0, ny: 0, nz: 0 } } } }
    } else { dados = { vetor: { u, v, w, abs, normalizado: { nx, ny, nz } } } }

    vetores.push( dados )
    modulos.push( abs )
}
let min = Math.min( ...modulos )
let max = Math.max( ...modulos )

console.log( '... os modulos de deformação foram calculados.' )
console.log( '... a interpolação de cores foi concluída.' )
return { vetores, min, max }
}

return matrizes
}

export default initMatrizes

```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE H - STYLE

Quadro 33 - Arquivo de Estilização.

Style.css

```
@import url('https://fonts.googleapis.com/css?family=Voltaire');
```

```
@font-face {  
  font-family: 'Futura';  
  src: url('fontes/futura-light-bt.ttf');  
  font-weight: normal;  
  font-style: normal;  
}
```

```
@font-face {  
  font-family: 'Futura Condensed';  
  src: url('fontes/futura-light-condensed-bt.ttf');  
  font-weight: normal;  
  font-style: normal;  
}
```

```
* {  
  border: 0;  
  padding: 0;  
  margin: 0;  
  color: white;  
  font-family: 'Futura';  
  -webkit-font-smoothing: antialiased;  
}
```

```
body {  
  width: 100vw;  
  height: 100vh;  
  background: rgb(0,0,0);  
}
```

```
canvas {  
  z-index: -2;  
  width: 100vw;  
  height: 100vh;  
  max-width: 100vw;  
  max-height: 100vh;  
}
```

```
.nav {  
  display: flex;  
  flex-direction: column;  
  justify-content: top;  
  position: fixed;  
  height: 100vh;  
  top: 0;  
  left: 20px;  
  padding-top: 28px ;  
  padding-bottom: 28px ;  
}
```

```
.nav-item {  
  align-items: center;  
  display: flex;  
  flex-direction: row;  
  margin-bottom: 6px;  
  cursor: pointer;  
}
```

```
.nav-link:hover ~ .nav-label {  
  opacity: 1;  
}
```

```
.nav-label {  
  font-size: 11px;  
  font-weight: bold;  
  opacity: 0;  
  transition: opacity 0.1s;  
}
```

```
.nav-link {  
  border-radius: 50%;  
  border: solid 1px white;  
  height: 10px;  
  width: 10px;  
  margin-right: 10px;  
  transition: transform 0.1s;  
  
}
```

```
.nav-link-selected {  
  background: white;  
  
}
```

```
.nav-link-selected ~ .nav-label {  
  opacity: 1;  
}
```

```
#instructions{  
  position:absolute;  
  width:100%;  
  bottom:40PX;  
  margin: auto;  
  font-family:'Voltaire', sans-serif;  
  color:white;  
  font-size:16px;  
  letter-spacing:1px;  
  text-transform: uppercase;  
  text-align : center;  
  user-select: none;  
}
```

```
.next, .before{  
  font-family: 'Futura';  
  font-size: 10px;  
  color: white;  
  font-weight: bold;  
  border: solid 2px white;  
  border-radius: 50%;  
  position: fixed;  
  display: flex;  
  align-items: center;  
  justify-content: center;  
  cursor: pointer;  
  height: 40px;  
  width: 40px;  
  bottom: 20px;  
  position: fixed;  
}
```

```
.next{

    right: 20px;
}
.before{
    left: 20px;
}
.bottom {
    height: 40px;
    background: black;
    width: 100vw;
    position: fixed;
    bottom: 0;
}
.info {
    font-family: 'Times New Roman', Times, serif;
    font-size: 22px;
    font-style: italic;
    color: white;
    font-weight: bold;
    border: solid 2px white;
    border-radius: 50%;
    position: fixed;
    display: flex;
    align-items: center;
    justify-content: center;
    cursor: pointer;
    height: 40px;
    width: 40px;

    position: fixed;
    bottom: 30px;
    left: 20px;
```

```
}
```

```
.form {  
  display: flex;  
  position: absolute ;  
  flex-direction: column;  
  top: 6px;  
  right: 30px;  
  width: 130px;  
  min-height: 300px;  
}
```

```
.box {  
  width: 100%;  
  height: 40px;  
  display: flex;  
  align-items: center;  
  justify-content: center;  
}
```

```
.input {  
  appearance: none;  
  width: 100%;  
  height: 2px;  
  background: #f5f5f5;  
  border-radius: 2px;  
  outline: none;  
  border: none;  
  margin-top: 20px;  
}
```

```
.input::-webkit-slider-thumb{  
  appearance: none;  
  width: 12px;  
  height: 12px;  
  background-color: black;
```

```
border: solid 1px white;
border-radius: 50%;
cursor: pointer;
}
.checkbox{
  cursor: pointer;
}

.gradiente {
  position: absolute;
  display: flex;
  justify-content: center;
  flex-direction: row;
  top: 40px;
  left: 50%;
  transform: translateX( -50% );
}

.gradiente-itens{
  padding-left: 3px;
  padding-right: 3px;

}

.gradiente-itens input{
  display: none;
}

.gradiente-label{
  display: block;
  height: 17px;
  width: 17px;
  border-radius: 50%;
```

```
border: solid 1px white;
cursor: pointer;
}
.gradiente-itens input:checked ~ label.v1 {
border: none;
background-image: linear-gradient(to right, rgb(255,0,255), rgb(0,0,255));
}
.gradiente-itens input:checked ~ label.v2 {
border: none;
background-image: linear-gradient(to right, rgb(0,0,255), rgb(0,255,255));
}
.gradiente-itens input:checked ~ label.v3 {
border: none;
background-image: linear-gradient(to right, rgb(0,255,255), rgb(0,255,0));
}
.gradiente-itens input:checked ~ label.v4 {
border: none;
background-image: linear-gradient(to right, rgb(0,255,0), rgb(255,255,0));
}
.gradiente-itens input:checked ~ label.v5 {
border: none;
background-image: linear-gradient(to right, rgb(255,255,0), rgb(255,0,0));
}
.gradiente-itens input:checked ~ label.v6 {
border: none;
background-image: linear-gradient(to right, rgb(255,0,0), rgb(255,0,255));
}
```

```
@media screen and (max-width: 767px) {
.gradiente {
justify-content: center;
flex-direction: column;
}
```

```
    top: auto;
    bottom: 90px;
    left: 20px;
    transform: none;
}
.gradiente-itens{
    padding-left: 0px;
    padding-right: 0px;
    padding-top: 3px;
    padding-bottom: 3px;
}
.gradiente-label{
    height: 15px;
    width: 15px;
}
}
```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE I - HTML

Quadro 34 - Arquivo de Html.

Index

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="style.css">
  <title>Document</title>
</head>
<body>
  <canvas id="canvas"></canvas>

  <div id="instructions">Arraste para Rotacionar</div>

  <script src="main.js" type="module"></script>
</body>
</html>
```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE J - PROGRAMA PRINCIPAL

Quadro 35 - Programa Principal.

main

```
import * as THREE from './three.module.js'
// instanciando metodos ambiente
import initCanvas from './src/Ambiente 3D/initCanvas.js'
import initCamera from './src/Ambiente 3D/initCamera.js'
import initCena from './src/Ambiente 3D/initCena.js'
import initIlluminar from './src/Ambiente 3D/initIlluminar.js'
import initRenderer from './src/Ambiente 3D/initRenderer.js'
// instanciando metodos malha
import initGeometria from './src/malha/initGeometria.js'
import posicionarMalha from './src/malha/posicionarMalha.js'
// instanciando metodos vertices
import initPontos from './src/vertices/initPontos.js'
import addConexoesPontos from './src/vertices/addConexoesPontos.js'
import deformarPontos from './src/vertices/deformarPontos.js'
import adicionarCoresNosPontos from './src/vertices/adicionarCoresNosPontos.js'
import adicionarVetoresNosPontos from './src/vertices/adicionarVetoresNosPontos.js'
import desenharPontos from './src/vertices/desenharPontos.js'
import desenharVetores from './src/vertices/desenharVetores.js'
// instanciando metodos conexoes
import initConexoes from './src/conexoes/initConexoes.js'
import desenharConexoes from './src/conexoes/desenharConexoes.js'
// instanciando metodos celulas
import initCelulas from './src/celulas/initCelulas.js'
import desenharCelulas from './src/celulas/desenharCelulas.js'
import addConexoesCelulas from './src/celulas/addConexoesCelulas.js'
import mapearVerticesCelulas from './src/celulas/mapearVerticesCelulas.js'
import refinamentoCelulas from './src/celulas/refinamentoCelulas.js'
import desenharGradiente from './src/celulas/desenharGradiente.js'
// instanciando metodos de calculo
```

```
import initMatrizes from './initMatrizes.js'
// instanciando metodos de fluxo
import initAgentes from './src/fluxo/initAgentes.js'
// instanciar interface
import criarElemento from './src/interface/criarElemento.js'
import initNav from './src/interface/initNav.js'
import initSliders from './src/interface/initSliders.js'
import initGradienteInputs from './src/interface/initGradienteInputs.js'
import initBefore from './src/interface/initBefore.js'
import navegarPontos from './src/interface/navegarPontos.js'

// Declaração de variaveis
let parametros = {
  dimensoes: { x: 1, y: 1, z: 1 },
  discretizador: { x: 1, y: 1, z: 1 },
  refinamento: { x: 1, y: 1, z: 1 },
  mostrarPontos: true,
  mostrarLinhas: true,
  calcular: 2,
  gradiente: 2,
  fluxo: 2,
  deformar: 2,
  refinar: 2,
  visibilidade: { v1: true, v2: true, v3: true, v4: true, v5: true, v6: true },
  etapas: [
    { titulo: 'Discretização', conteudo: null },
    { titulo: 'Dimensionamento', conteudo: null },
    { titulo: 'Isovalores', conteudo: null },
    { titulo: 'Fluxo', conteudo: null },
    { titulo: 'Deformação', conteudo: null }
  ],
  rodar: { x: .004, y: .004, z: .004 },
```

```

    tela: { x: null, y: null, z: null },
    mode: null,
    rigidez: null,
    cfg: 3,
  }
  let canvas, camera, cena, renderer, group, estrutura, esferas
  let malha, pontos, conexoes, celulas, agentes, malhaCelulas, matrizes, cameraZ
  let body = document.body
  let nav, navItens, navLinks, next, before, bottom, form
  let f, u

  // App principal
  init()
  function init() {
    initEnv()
    ajustarMalha()
    initInterface( body, parametros )
    window.addEventListener( 'resize', resize )
    loop()
  }
  function loop() {
    group.rotation.y += parametros.rodar.x
    group.rotation.x += parametros.rodar.y
    group.rotation.z += parametros.rodar.z
    render()

    if ( agentes !== null ) {
      for ( let agente of agentes ) {
        let max = agente.path.length * 6
        agente.way += agente.passo
        agente.mesh.geometry.setDrawRange( 0, agente.way )
        if ( agente.way >= max ) {
          agente.way = 0

```

```

    }
    }
}
requestAnimationFrame( loop )
}

// Ferramentas de Modelagem
function ajustarMalha() {
    parametros.refinamento.x = parametros.refinamento.y = parametros.refinamento.z = 1
    cena = initCena()
    initIluminar( cena )
    malha = initGeometria( parametros )
    matrizes = null

    if ( parametros.mode === 'manual' ) {
        if ( parametros.cfg === 3 ) {
            pontos = [
                { identificador: 0, posicionamento: { x: 45, y: 0, z: 0 }, conexoes: { identificador:
[1, 2, 3] }, vetor: null, cor: null },
                { identificador: 1, posicionamento: { x: 0, y: 60, z: 0 }, conexoes: { identificador:
[0] }, vetor: null, cor: null },
                { identificador: 2, posicionamento: { x: 125, y: 60, z: 0 }, conexoes: { identificador:
[0] }, vetor: null, cor: null },
                { identificador: 3, posicionamento: { x: 90, y: 60, z: 0 }, conexoes: { identificador:
[0] }, vetor: null, cor: null }
            ]
            f = [
                { identificador: 0, x: 0, y: -1800000, z: 0 },
                { identificador: 1, x: 0, y: 0, z: 0 },
                { identificador: 2, x: 0, y: 0, z: 0 },
                { identificador: 3, x: 0, y: 0, z: 0 }
            ]
            u = [

```

```

    { identificador: 0, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 1, u: 0, v: 0, w: 0 },
    { identificador: 2, u: 0, v: 0, w: 0 },
    { identificador: 3, u: 0, v: 0, w: 0 }
  ]
  parametros.rigidez = { A: 250, E:200*(1000) }
}
if ( parametros.cfg == 1 ) {
  pontos = [
    {
      identificador: 0,
      posicionamento: { x: 43.3, y: 8.216, z: 51.2971 },
      conexoes: { identificador: [ 1, 2, 3, 4, 5, 6 ] },
      vetor: null, cor: null
    },
    {
      identificador: 1,
      posicionamento: { x: 30.8, y: 6.216, z: 71.6492 },
      conexoes: { identificador: [ 0, 2, 6, 7, 12 ] },
      vetor: null, cor: null
    },
    {
      identificador: 2,
      posicionamento: { x: 55.8, y: 6.216, z: 71.6492 },
      conexoes: { identificador: [ 0, 1, 3, 7, 8 ] },
      vetor: null, cor: null
    },
    {
      identificador: 3,
      posicionamento: { x: 68.3, y: 6.216, z: 51.2971 },
      conexoes: { identificador: [ 0, 2, 4, 8, 9 ] },
      vetor: null, cor: null
    },

```

```
{
  identificador: 4,
  posicionamento: { x: 55.8, y: 6.216, z: 28.3479 },
  conexoes: { identificador: [ 0, 3, 5, 9, 10 ] },
  vetor: null, cor: null
},
{
  identificador: 5,
  posicionamento: { x: 30.8, y: 6.216, z: 28.3479 },
  conexoes: { identificador: [ 0, 4, 6, 10, 11 ] },
  vetor: null, cor: null
},
{
  identificador: 6,
  posicionamento: { x: 18.3, y: 6.216, z: 51.2971 },
  conexoes: { identificador: [ 0, 1, 5, 11, 12 ] },
  vetor: null, cor: null
},
{
  identificador: 7,
  posicionamento: { x: 43.3, y: 0, z: 99.9971 },
  conexoes: { identificador: [ 1, 2 ] },
  vetor: null, cor: null
},
{
  identificador: 8,
  posicionamento: { x: 86.6, y: 0, z: 74.9978 },
  conexoes: { identificador: [ 2, 3 ] },
  vetor: null, cor: null
},
{
  identificador: 9,
  posicionamento: { x: 86.6, y: 0, z: 24.9993 },
```

```

    conexoes: { identificador: [ 3, 4 ] },
    vetor: null, cor: null
  },
  {
    identificador: 10,
    posicionamento: { x: 43.3, y: 0, z: 0 },
    conexoes: { identificador: [ 4, 5 ] },
    vetor: null, cor: null
  },
  {
    identificador: 11,
    posicionamento: { x: 0, y: 0, z: 24.9993 },
    conexoes: { identificador: [ 5, 6 ] },
    vetor: null, cor: null
  },
  {
    identificador: 12,
    posicionamento: { x: 0, y: 0, z: 74.9978 },
    conexoes: { identificador: [ 1, 6 ] },
    vetor: null, cor: null
  },
]

f=[
  { identificador: 0, x: 0, y: -7.81, z: 0 },
  { identificador: 1, x: 0, y: 0, z: 0 },
  { identificador: 2, x: 0, y: 0, z: 0 },
  { identificador: 3, x: 0, y: 0, z: 0 },
  { identificador: 4, x: 0, y: 0, z: 0 },
  { identificador: 5, x: 0, y: 0, z: 0 },
  { identificador: 6, x: 0, y: 0, z: 0 },
  { identificador: 7, x: 0, y: 0, z: 0 },
  { identificador: 8, x: 0, y: 0, z: 0 },

```

```

    { identificador: 9, x: 0, y: 0, z: 0 },
    { identificador: 10, x: 0, y: 0, z: 0 },
    { identificador: 11, x: 0, y: 0, z: 0 },
    { identificador: 12, x: 0, y: 0, z: 0 }
  ]
  u = [
    { identificador: 0, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 1, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 2, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 3, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 4, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 5, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 6, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 7, u: 0, v: 0, w: 0 },
    { identificador: 8, u: 0, v: 0, w: 0 },
    { identificador: 9, u: 0, v: 0, w: 0 },
    { identificador: 10, u: 0, v: 0, w: 0 },
    { identificador: 11, u: 0, v: 0, w: 0 },
    { identificador: 12, u: 0, v: 0, w: 0 }
  ]
  parametros.rigidez = { A: 1, E: 10**4 }
}
if ( parametros.cfg == 4 ) {
  pontos = [
    { identificador: 0, posicionamento: { x: 0, y: 40, z: 0 }, conexoes: { identificador:
[ 1, 2 ] }, vetor: null, cor: null },
    { identificador: 1, posicionamento: { x: 30, y: 40, z: 0 }, conexoes: { identificador:
[0,2,3,4] }, vetor: null, cor: null },
    { identificador: 2, posicionamento: { x: 0, y: 0, z: 0 }, conexoes: { identificador:
[0,1,3] }, vetor: null, cor: null },
    { identificador: 3, posicionamento: { x: 30, y: 0, z: 0 }, conexoes: { identificador:
[2,1,4] }, vetor: null, cor: null },
    { identificador: 4, posicionamento: { x: 60, y: 0, z: 0 }, conexoes: { identificador:

```

```

[1,3] }, vetor: null, cor: null }
]
f=[
    { identificador: 0, x: 0, y: 0, z: 0 },
    { identificador: 1, x: 10000, y: 0, z: 0 },
    { identificador: 2, x: 0, y: 0, z: 0 },
    { identificador: 3, x: 0, y: -10000, z: 0 },
    { identificador: 4, x: 0, y: 0, z: 0 }
]
u=[
    { identificador: 0, u: 0, v: 0, w: 0 },
    { identificador: 1, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 2, u: 0, v: 'indefinido', w: 'indefinido' },
    { identificador: 3, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
    { identificador: 4, u: 0, v: 0, w: 0 }
]
parametros.rigidez = { A: 1, E: 10**6 }
}
if ( parametros.cfg == 5 ) {
    pontos = [
        { identificador: 0, posicionamento: { x: 0, y: 0, z: 0 }, conexoes: { identificador:
[ 1 ] }, vetor: null, cor: null },
        { identificador: 1, posicionamento: { x: 40, y: 0, z: 0 }, conexoes: { identificador:
[0,2] }, vetor: null, cor: null },
        { identificador: 2, posicionamento: { x: 0, y: 30, z: 0 }, conexoes: { identificador:
[1] }, vetor: null, cor: null }
    ]
    f=[
        { identificador: 0, x: 0, y: 0, z: 0 },
        { identificador: 1, x: 0, y: -10000, z: 0 },
        { identificador: 2, x: 0, y: 0, z: 0 }
    ]
    u=[

```

```

        { identificador: 0, u: 0, v: 0, w: 0 },
        { identificador: 1, u: 'indefinido', v: 'indefinido', w: 'indefinido' },
        { identificador: 2, u: 0, v: 0, w: 0 }
    ]
    parametros.rigidez = { A: 200 * ( 10**(-4)), E: 200 * (10**6) }
}
console.log( pontos )
rebubinarGrupos()
posicionarMalha( pontos, estrutura, parametros )
rebubinarMalha()
desenharPontos( estrutura, pontos, malha, .4 )
desenharConexoes( estrutura, conexoes )
} else {
    pontos = initPontos( malha )
    pontos = addConexoesPontos( malha, pontos, parametros )
    rebubinarGrupos()
    posicionarMalha( pontos, estrutura, parametros )
    rebubinarMalha()
    desenharPontos( estrutura, pontos, malha, .4 )
    desenharConexoes( estrutura, conexoes )
    desenharGradiente( estrutura, malhaCelulas, malha, matrizes, 1 ,
parametros.visibilidade )
    desenharCelulas( celulas, esferas, estrutura, malha )
}
camera = initCamera( parametros ).camera
cameraZ = initCamera( parametros ).cameraZ
parametros.rodar.x = .004
parametros.rodar.y = .004
parametros.rodar.z = .004
}

```

```

function calcularMalha() {
    console.log( conexoes )
    matrizes = initMatrizes( pontos, conexoes, f, u, parametros.rigidez )
    pontos = adicionarVetoresNosPontos( pontos, matrizes )
    pontos = adicionarCoresNosPontos( pontos, matrizes )
    colorirMalha()
    rebubinarMalha()
    posicionarMalha( pontos, estrutura, parametros )
    desenharVetores( estrutura, pontos, malha )
    desenharPontos( estrutura, pontos, malha, .4 )
    desenharConexoes( estrutura, conexoes )

    if ( parametros.mode != 'manual' ) {
        desenharCelulas( celulas, esferas, estrutura, malha )
        desenharGradiente(      estrutura,      malhaCelulas,      malha,      matrizes,      .01,
parametros.visibilidade )
        refinamentoCelulas( esferas, celulas, parametros.refinamento, matrizes, malha )
    }

    parametros.rodar.x = .002
    parametros.rodar.y = 0
    parametros.rodar.z = 0
}

function desenharMalhaCalculada() {
    rebubinarMalha()
    colorirMalha()
    posicionarMalha( pontos, estrutura, parametros )
    desenharPontos( estrutura, pontos, malha, .1 )
    if ( parametros.mode != 'manual' ) {
        desenharGradiente(      estrutura,      malhaCelulas,      malha,      matrizes,      .05,
parametros.visibilidade )
    }
}

```

```

    parametros.rodar.x = .0025
    parametros.rodar.y = 0
    parametros.rodar.z = 0
}

function desenharFluxo() {
    desenharMalhaCalculada()
    rebubinarMalha()
    colorirMalha()
    posicionarMalha( pontos, estrutura, parametros )
    desenharPontos( estrutura, pontos, malha, .1 )
    if ( parametros.mode != 'manual' ) {
        desenharGradiente( estrutura, malhaCelulas, malha, matrizes, .01 ,
parametros.visibilidade )
        agentes = initAgentes( agentes, estrutura, parametros, malha, malhaCelulas,
parametros.visibilidade )
    }
    parametros.rodar.x = .003
    parametros.rodar.y = 0
    parametros.rodar.z = 0

}

function desenharMalhaDeformada() {
    pontos = deformarPontos( pontos )
    conexoes = initConexoes( pontos )
    colorirMalha()
    posicionarMalha( pontos, estrutura, parametros )
    desenharVetores( estrutura, pontos, malha )
    desenharConexoes( estrutura, conexoes )
    desenharPontos( estrutura, pontos, malha, .4 )
    //refinamentoCelulas( esferas, celulas, parametros.refinamento, matrizes, malha )
}

```

```

function rebubinarMalha() {
    conexoes = initConexoes( pontos )
    if ( parametros.mode != 'manual' ) {
        celulas = initCelulas( malha )
        celulas = addConexoesCelulas( malha, pontos, celulas )
        celulas = mapearVerticesCelulas( celulas )
        malhaCelulas = refinamentoCelulas( esferas, celulas, parametros.refinamento, matrizes,
malha )
    }

}

function rebubinarGrupos() {
    group = new THREE.Group
    cena.add(group)
    estrutura = new THREE.Group
    group.add(estrutura)
    esferas = new THREE.Group()
    estrutura.add( esferas )
}

function colorirMalha() {
    cena = initCena()
    initIluminar( cena )
    rebubinarGrupos()
}

function cfgRefinamentoGradiente() {
    parametros.refinamento.x = parametros.dimensoes.x <= 3 ? 10 : (parametros.dimensoes.x *
2 + 3 )
    parametros.refinamento.y = parametros.dimensoes.y <= 3 ? 10 : (parametros.dimensoes.y *
2 + 3 )
    parametros.refinamento.z = parametros.dimensoes.z <= 3 ? 10 : (parametros.dimensoes.z *
2 + 3 )
}

```

```
function cfgRefinamentoFluxo() {
    parametros.refinamento.x = 4
    parametros.refinamento.y = 4
    parametros.refinamento.z = 4
}

// Configurações do ambiente 3d
function initEnv() {
    console.log( '2 - instanciando Ambiente' )
    canvas = initCanvas()
    camera = initCamera( parametros ).camera
    cameraZ = initCamera( parametros ).cameraZ
    cena = initCena()
    initIluminar( cena )
    renderer = initRenderer( canvas )
}
function render() {
    renderer.render( cena, camera )
    //console.log( '10 - atualizando o Renderizador' )
}
function resize() {
    renderer.setSize( window.innerWidth, window.innerHeight )
    camera.aspect = window.innerWidth / window.innerHeight
    camera.updateProjectionMatrix()
    console.log( '7 - ajustando Tela' )
}

// carregar interface
function initInterface( body, parametros ) {
    navLinks = initNav( body, parametros )
    initSliders( body, parametros )
    initGradienteInputs( body, parametros, navLinks )
}
```

```
    next = initNext( body, matrizes, navLinks, parametros.rodar )
    before = initBefore( body, navLinks, parametros.rodar )
}
```

```
function initNext( body ) {
    let next = createElement( 'div', { class: 'next' } )
    next.append( '➤ ' )
    next.addEventListener( 'click', event => executarNext() )
    body.append( next )
    return next
}

function executarNext() {
    if ( matrizes == null ) {
        calcularMalha()
        navegarPontos( navLinks[1].a, navLinks )
    } else if ( parametros.rodar.x == .002 ) {
        cfgRefinamentoGradiente()
        desenharMalhaCalculada()
        navegarPontos( navLinks[2].a, navLinks )
    } else if ( parametros.rodar.x == .0025 ) {
        cfgRefinamentoFluxo()
        desenharFluxo()
        navegarPontos( navLinks[3].a, navLinks )
    } else if ( parametros.rodar.x == .003 ) {
        desenharMalhaDeformada()
        navegarPontos( navLinks[4].a, navLinks )
    }
}

export {
    ajustarMalha,
    cfgRefinamentoGradiente,
    cfgRefinamentoFluxo,
    calcularMalha,
```

```
desenharMalhaCalculada,  
desenharFluxo,  
desenharMalhaDeformada  
}
```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE K - COORDENADAS GEOMETRIA DOMO

Quadro 36 - Pontos Domo 132 Elementos.

pontos	
pontos = [	
{	
'identificador': 0,	
'posicionamento': {	
'x': 0,	
'y': 179.022,	
'z': 0	
},	
'conexoes': {	
'identificador': [	
1,	
2,	
3,	
4,	
5,	
6,	
7,	
8,	
9,	
10,	
11,	
12,	
13,	
14,	
15,	
16,	
17,	
18,	
19,	

```
        20,
        21,
        22,
        23,
        24
    ]
},
'vetor': null,
'cor': null
},
{
    'identificador': 1,
    'posicionamento': {
        'x': 0,
        'y': 156.947,
        'z': 86.7
    },
    'conexoes': {
        'identificador': [
            0,
            2,
            24,
            25,
            26
        ]
    },
    'vetor': null,
    'cor': null
},
{
    'identificador': 2,
    'posicionamento': {
        'x': 22.4396,
```

```
'y': 156.947,  
'z': 83.7458  
},  
'conexoes': {  
  'identificador': [  
    0,  
    1,  
    3,  
    26,  
    27  
  ]  
},  
'vetor': null,  
'cor': null  
},  
{  
  'identificador': 3,  
  'posicionamento': {  
    'x': 43.95,  
    'y': 156.947,  
    'z': 75.0844  
  },  
  'conexoes': {  
    'identificador': [  
      0,  
      2,  
      4,  
      27,  
      28  
    ]  
  },  
  'vetor': null,  
  'cor': null
```

```
    ]  
  },  
  'vetor': null,  
  'cor': null  
},  
{  
  'identificador': 6,  
  'posicionamento': {  
    'x': 83.7458,  
    'y': 156.947,  
    'z': 22.4396  
  },  
  'conexoes': {  
    'identificador': [  
      0,  
      5,  
      7,  
      30,  
      31  
    ]  
  },  
  'vetor': null,  
  'cor': null  
},  
{  
  'identificador': 7,  
  'posicionamento': {  
    'x': 86.7,  
    'y': 156.947,  
    'z': 0  
  },  
  'conexoes': {
```

```
'identificador': [  
    0,  
    6,  
    8,  
    31,  
    32  
]  
,  
'vetor': null,  
'cor': null  
,  
{  
    'identificador': 8,  
    'posicionamento': {  
        'x': 83.7458,  
        'y': 156.947,  
        'z': -22.4396  
    },  
    'conexoes': {  
        'identificador': [  
            0,  
            7,  
            9,  
            32,  
            33  
        ]  
    },  
    'vetor': null,  
    'cor': null  
},  
{  
    'identificador': 9,  
    'posicionamento': {
```

```
'x': 75.0844,  
'y': 156.947,  
'z': -43.95  
,  
'conexoes': {  
  'identificador': [  
    0,  
    8,  
    10,  
    33,  
    34  
  ]  
,  
'vetor': null,  
'cor': null  
},  
{  
  'identificador': 10,  
  'posicionamento': {  
    'x': 61.3062,  
    'y': 156.947,  
    'z': -61.3062  
  },  
  'conexoes': {  
    'identificador': [  
      0,  
      9,  
      11,  
      34,  
      35  
    ]  
  },  
  'vetor': null,
```

```
'cor': null
},
{
  'identificador': 11,
  'posicionamento': {
    'x': 43.95,
    'y': 156.947,
    'z': -75.0844
  },
  'conexoes': {
    'identificador': [
      0,
      10,
      12,
      35,
      36
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 12,
  'posicionamento': {
    'x': 22.4396,
    'y': 156.947,
    'z': -83.7458
  },
  'conexoes': {
    'identificador': [
      0,
      11,
      13,
```

```
        36,  
        37  
    ]  
    },  
    'vetor': null,  
    'cor': null  
},  
{  
    'identificador': 13,  
    'posicionamento': {  
        'x': 0,  
        'y': 156.947,  
        'z': -86.7  
    },  
    'conexoes': {  
        'identificador': [  
            0,  
            12,  
            14,  
            37,  
            38  
        ]  
    },  
    'vetor': null,  
    'cor': null  
},  
{  
    'identificador': 14,  
    'posicionamento': {  
        'x': -22.4396,  
        'y': 156.947,  
        'z': -83.7458  
    },  
}
```

```
'conexoes': {
  'identificador': [
    0,
    13,
    15,
    38,
    39
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 15,
  'posicionamento': {
    'x': -43.95,
    'y': 156.947,
    'z': -75.0844
  },
  'conexoes': {
    'identificador': [
      0,
      14,
      16,
      39,
      40
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 16,
```

```
'posicionamento': {
  'x': -61.3062,
  'y': 156.947,
  'z': -61.3062
},
'conexoes': {
  'identificador': [
    0,
    15,
    17,
    40,
    41
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 17,
  'posicionamento': {
    'x': -75.0844,
    'y': 156.947,
    'z': -43.95
  },
  'conexoes': {
    'identificador': [
      0,
      16,
      18,
      41,
      42
    ]
  ]
},
```

```
'vetor': null,
'cor': null
},
{
  'identificador': 18,
  'posicionamento': {
    'x': -83.7458,
    'y': 156.947,
    'z': -22.4396
  },
  'conexoes': {
    'identificador': [
      0,
      17,
      19,
      42,
      43
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 19,
  'posicionamento': {
    'x': -86.7,
    'y': 156.947,
    'z': 0
  },
  'conexoes': {
    'identificador': [
      0,
      18,
```

```
    20,
    43,
    44
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 20,
  'posicionamento': {
    'x': -83.7458,
    'y': 156.947,
    'z': 22.4396
  },
  'conexoes': {
    'identificador': [
      0,
      19,
      21,
      44,
      45
    ]
  }
},
'vetor': null,
'cor': null
},
{
  'identificador': 21,
  'posicionamento': {
    'x': -75.0844,
    'y': 156.947,
    'z': 43.95
```

```
    },
    'conexoes': {
      'identificador': [
        0,
        20,
        22,
        45,
        46
      ]
    },
    'vetor': null,
    'cor': null
  },
  {
    'identificador': 22,
    'posicionamento': {
      'x': -61.3062,
      'y': 156.947,
      'z': 61.3062
    },
    'conexoes': {
      'identificador': [
        0,
        21,
        23,
        46,
        47
      ]
    },
    'vetor': null,
    'cor': null
  },
  {
```

```
'identificador': 23,
'posicionamento': {
  'x': -43.95,
  'y': 156.947,
  'z': 75.0844
},
'conexoes': {
  'identificador': [
    0,
    22,
    24,
    47,
    48
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 24,
  'posicionamento': {
    'x': -22.4396,
    'y': 156.947,
    'z': 83.7458
  },
  'conexoes': {
    'identificador': [
      0,
      1,
      23,
      25,
      48
    ]
  ]
}
```

```
    },
    'vetor': null,
    'cor': null
  },
  {
    'identificador': 25,
    'posicionamento': {
      'x': -23.5684,
      'y': 94.083,
      'z': 179.02
    },
    'conexoes': {
      'identificador': [
        1,
        24,
        26,
        48,
        49,
        72
      ]
    },
    'vetor': null,
    'cor': null
  },
  {
    'identificador': 26,
    'posicionamento': {
      'x': 23.5684,
      'y': 94.083,
      'z': 179.02
    },
    'conexoes': {
      'identificador': [
```

```
    1,
    2,
    25,
    27,
    49,
    50
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 27,
  'posicionamento': {
    'x': 69.0991,
    'y': 94.083,
    'z': 166.8201
  },
  'conexoes': {
    'identificador': [
      2,
      3,
      26,
      28,
      50,
      51
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 28,
```

```
'posicionamento': {  
  'x': 109.9209,  
  'y': 94.083,  
  'z': 143.2517  
},  
'conexoes': {  
  'identificador': [  
    3,  
    4,  
    27,  
    29,  
    51,  
    52  
  ]  
},  
'vetor': null,  
'cor': null  
},  
{  
  'identificador': 29,  
  'posicionamento': {  
    'x': 143.2517,  
    'y': 94.083,  
    'z': 109.9209  
  },  
  'conexoes': {  
    'identificador': [  
      4,  
      5,  
      28,  
      30,  
      52,  
      53
```

```
]
},
'vetor': null,
'cor': null
},
{
  'identificador': 30,
  'posicionamento': {
    'x': 166.8201,
    'y': 94.083,
    'z': 69.0991
  },
  'conexoes': {
    'identificador': [
      5,
      6,
      29,
      31,
      53,
      54
    ]
  }
},
'vetor': null,
'cor': null
},
{
  'identificador': 31,
  'posicionamento': {
    'x': 179.02,
    'y': 94.083,
    'z': 23.5684
  },
  'conexoes': {
```

```
      'identificador': [
        6,
        7,
        30,
        32,
        54,
        55
      ]
    },
    'vetor': null,
    'cor': null
  },
  {
    'identificador': 32,
    'posicionamento': {
      'x': 179.02,
      'y': 94.083,
      'z': -23.5684
    },
    'conexoes': {
      'identificador': [
        7,
        8,
        31,
        33,
        55,
        56
      ]
    },
    'vetor': null,
    'cor': null
  },
  {
```

```
'identificador': 33,
'posicionamento': {
  'x': 166.8201,
  'y': 94.083,
  'z': -69.0991
},
'conexoes': {
  'identificador': [
    8,
    9,
    32,
    34,
    56,
    57
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 34,
  'posicionamento': {
    'x': 143.2517,
    'y': 94.083,
    'z': -109.9209
  },
  'conexoes': {
    'identificador': [
      9,
      10,
      33,
      35,
      57,
```

```
]
},
'vetor': null,
'cor': null
},
{
  'identificador': 35,
  'posicionamento': {
    'x': 109.9209,
    'y': 94.083,
    'z': -143.2517
  },
  'conexoes': {
    'identificador': [
      10,
      11,
      34,
      36,
      58,
      59
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 36,
  'posicionamento': {
    'x': 69.0991,
    'y': 94.083,
    'z': -166.8201
  },
  },
```

```
'conexoes': {  
  'identificador': [  
    11,  
    12,  
    35,  
    37,  
    59,  
    60  
  ]  
},  
'vetor': null,  
'cor': null  
},  
{  
  'identificador': 37,  
  'posicionamento': {  
    'x': 23.5684,  
    'y': 94.083,  
    'z': -179.02  
  },  
  'conexoes': {  
    'identificador': [  
      12,  
      13,  
      36,  
      38,  
      60,  
      61  
    ]  
  },  
  'vetor': null,  
  'cor': null  
},
```

```
{
  'identificador': 38,
  'posicionamento': {
    'x': -23.5684,
    'y': 94.083,
    'z': -179.02
  },
  'conexoes': {
    'identificador': [
      13,
      14,
      37,
      39,
      61,
      62
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 39,
  'posicionamento': {
    'x': -69.0991,
    'y': 94.083,
    'z': -166.8201
  },
  'conexoes': {
    'identificador': [
      14,
      15,
      38,
      40,
```

```
        62,  
        63  
    ]  
    },  
    'vetor': null,  
    'cor': null  
},  
{  
    'identificador': 40,  
    'posicionamento': {  
        'x': -109.9209,  
        'y': 94.083,  
        'z': -143.2517  
    },  
    'conexoes': {  
        'identificador': [  
            15,  
            16,  
            39,  
            41,  
            63,  
            64  
        ]  
    },  
    'vetor': null,  
    'cor': null  
},  
{  
    'identificador': 41,  
    'posicionamento': {  
        'x': -143.2517,  
        'y': 94.083,  
        'z': -109.9209
```

```
    },  
    'conexoes': {  
      'identificador': [  
        16,  
        17,  
        40,  
        42,  
        64,  
        65  
      ]  
    },  
    'vetor': null,  
    'cor': null  
  },  
  {  
    'identificador': 42,  
    'posicionamento': {  
      'x': -166.8201,  
      'y': 94.083,  
      'z': -69.0991  
    },  
    'conexoes': {  
      'identificador': [  
        17,  
        18,  
        41,  
        43,  
        65,  
        66  
      ]  
    },  
    'vetor': null,  
    'cor': null
```

```
    },  
    {  
      'identificador': 43,  
      'posicionamento': {  
        'x': -179.02,  
        'y': 94.083,  
        'z': -23.5684  
      },  
      'conexoes': {  
        'identificador': [  
          18,  
          19,  
          42,  
          44,  
          66,  
          67  
        ]  
      },  
      'vetor': null,  
      'cor': null  
    },  
    {  
      'identificador': 44,  
      'posicionamento': {  
        'x': -179.02,  
        'y': 94.083,  
        'z': 23.5684  
      },  
      'conexoes': {  
        'identificador': [  
          19,  
          20,  
          43,
```

```
        45,
        67,
        68
    ]
},
'vetor': null,
'cor': null
},
{
    'identificador': 45,
    'posicionamento': {
        'x': -166.8201,
        'y': 94.083,
        'z': 69.0991
    },
    'conexoes': {
        'identificador': [
            20,
            21,
            44,
            46,
            68,
            69
        ]
    },
    'vetor': null,
    'cor': null
},
{
    'identificador': 46,
    'posicionamento': {
        'x': -143.2517,
        'y': 94.083,
```

```
'z': 109.9209
},
'conexoes': {
  'identificador': [
    21,
    22,
    45,
    47,
    69,
    70
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 47,
  'posicionamento': {
    'x': -109.9209,
    'y': 94.083,
    'z': 143.2517
  },
  'conexoes': {
    'identificador': [
      22,
      23,
      46,
      48,
      70,
      71
    ]
  },
  'vetor': null,
```

```
'cor': null
},
{
  'identificador': 48,
  'posicionamento': {
    'x': -69.0991,
    'y': 94.083,
    'z': 166.8201
  },
  'conexoes': {
    'identificador': [
      23,
      24,
      25,
      47,
      71,
      72
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 49,
  'posicionamento': {
    'x': 0,
    'y': 0,
    'z': 265.72
  },
  'conexoes': {
    'identificador': [
      25,
      26,
```

```
        50,
        72
    ]
},
'vetor': null,
'cor': null
},
{
    'identificador': 50,
    'posicionamento': {
        'x': 68.7734,
        'y': 0,
        'z': 256.6658
    },
    'conexoes': {
        'identificador': [
            26,
            27,
            49,
            51
        ]
    },
    'vetor': null,
    'cor': null
},
{
    'identificador': 51,
    'posicionamento': {
        'x': 132.86,
        'y': 0,
        'z': 230.1203
    },
    'conexoes': {
```

```
'identificador': [  
    27,  
    28,  
    50,  
    52  
]  
,  
'vetor': null,  
'cor': null  
,  
{  
    'identificador': 52,  
    'posicionamento': {  
        'x': 187.8924,  
        'y': 0,  
        'z': 187.8924  
    },  
    'conexoes': {  
        'identificador': [  
            28,  
            29,  
            51,  
            53  
        ],  
        'vetor': null,  
        'cor': null  
    },  
    {  
        'identificador': 53,  
        'posicionamento': {  
            'x': 230.1203,  
            'y': 0,
```

```
'z': 132.86
},
'conexoes': {
  'identificador': [
    29,
    30,
    52,
    54
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 54,
  'posicionamento': {
    'x': 256.6658,
    'y': 0,
    'z': 68.7734
  },
  'conexoes': {
    'identificador': [
      30,
      31,
      53,
      55
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 55,
```

```
'posicionamento': {  
  'x': 265.72,  
  'y': 0,  
  'z': 0  
},  
'conexoes': {  
  'identificador': [  
    31,  
    32,  
    54,  
    56  
  ]  
},  
'vetor': null,  
'cor': null  
},  
{  
  'identificador': 56,  
  'posicionamento': {  
    'x': 256.6658,  
    'y': 0,  
    'z': -68.7734  
  },  
  'conexoes': {  
    'identificador': [  
      32,  
      33,  
      55,  
      57  
    ]  
  },  
  'vetor': null,  
  'cor': null
```

```
    },  
    {  
      'identificador': 57,  
      'posicionamento': {  
        'x': 230.1203,  
        'y': 0,  
        'z': -132.86  
      },  
      'conexoes': {  
        'identificador': [  
          33,  
          34,  
          56,  
          58  
        ]  
      },  
      'vetor': null,  
      'cor': null  
    },  
    {  
      'identificador': 58,  
      'posicionamento': {  
        'x': 187.8924,  
        'y': 0,  
        'z': -187.8924  
      },  
      'conexoes': {  
        'identificador': [  
          34,  
          35,  
          57,  
          59  
        ]  
      }  
    }  
  ]  
}
```

```
    },
    'vetor': null,
    'cor': null
  },
  {
    'identificador': 59,
    'posicionamento': {
      'x': 132.86,
      'y': 0,
      'z': -230.1203
    },
    'conexoes': {
      'identificador': [
        35,
        36,
        58,
        60
      ]
    },
    'vetor': null,
    'cor': null
  },
  {
    'identificador': 60,
    'posicionamento': {
      'x': 68.7734,
      'y': 0,
      'z': -256.6658
    },
    'conexoes': {
      'identificador': [
        36,
        37,
```

```
        59,  
        61  
    ]  
    },  
    'vetor': null,  
    'cor': null  
},  
{  
    'identificador': 61,  
    'posicionamento': {  
        'x': 0,  
        'y': 0,  
        'z': -265.72  
    },  
    'conexoes': {  
        'identificador': [  
            37,  
            38,  
            60,  
            62  
        ]  
    },  
    'vetor': null,  
    'cor': null  
},  
{  
    'identificador': 62,  
    'posicionamento': {  
        'x': -68.7734,  
        'y': 0,  
        'z': -256.6658  
    },  
    'conexoes': {
```

```
'identificador': [  
    38,  
    39,  
    61,  
    63  
]  
,  
'vetor': null,  
'cor': null  
,  
{  
    'identificador': 63,  
    'posicionamento': {  
        'x': -132.86,  
        'y': 0,  
        'z': -230.1203  
    },  
    'conexoes': {  
        'identificador': [  
            39,  
            40,  
            62,  
            64  
        ],  
        },  
        'vetor': null,  
        'cor': null  
    },  
    {  
        'identificador': 64,  
        'posicionamento': {  
            'x': -187.8924,  
            'y': 0,
```

```
'z': -187.8924
},
'conexoes': {
  'identificador': [
    40,
    41,
    63,
    65
  ]
},
'vetor': null,
'cor': null
},
{
  'identificador': 65,
  'posicionamento': {
    'x': -230.1203,
    'y': 0,
    'z': -132.86
  },
  'conexoes': {
    'identificador': [
      41,
      42,
      64,
      66
    ]
  },
  'vetor': null,
  'cor': null
},
{
  'identificador': 66,
```

```
'posicionamento': {  
  'x': -256.6658,  
  'y': 0,  
  'z': -68.7734  
},  
'conexoes': {  
  'identificador': [  
    42,  
    43,  
    65,  
    67  
  ]  
},  
'vetor': null,  
'cor': null  
},  
{  
  'identificador': 67,  
  'posicionamento': {  
    'x': -265.72,  
    'y': 0,  
    'z': 0  
  },  
  'conexoes': {  
    'identificador': [  
      43,  
      44,  
      66,  
      68  
    ]  
  },  
  'vetor': null,  
  'cor': null
```

```
    },  
    {  
      'identificador': 68,  
      'posicionamento': {  
        'x': -256.6658,  
        'y': 0,  
        'z': 68.7734  
      },  
      'conexoes': {  
        'identificador': [  
          44,  
          45,  
          67,  
          69  
        ]  
      },  
      'vetor': null,  
      'cor': null  
    },  
    {  
      'identificador': 69,  
      'posicionamento': {  
        'x': -230.1203,  
        'y': 0,  
        'z': 132.86  
      },  
      'conexoes': {  
        'identificador': [  
          45,  
          46,  
          68,  
          70  
        ]  
      }  
    }  
  ]  
}
```

```
    },
    'vetor': null,
    'cor': null
  },
  {
    'identificador': 70,
    'posicionamento': {
      'x': -187.8924,
      'y': 0,
      'z': 187.8924
    },
    'conexoes': {
      'identificador': [
        46,
        47,
        69,
        71
      ]
    },
    'vetor': null,
    'cor': null
  },
  {
    'identificador': 71,
    'posicionamento': {
      'x': -132.86,
      'y': 0,
      'z': 230.1203
    },
    'conexoes': {
      'identificador': [
        47,
        48,
```

```
        70,  
        72  
    ]  
    },  
    'vetor': null,  
    'cor': null  
},  
{  
    'identificador': 72,  
    'posicionamento': {  
        'x': -68.7734,  
        'y': 0,  
        'z': 256.6658  
    },  
    'conexoes': {  
        'identificador': [  
            25,  
            48,  
            49,  
            71  
        ]  
    },  
    'vetor': null,  
    'cor': null  
}  
]
```

Fonte: Elaborado pelo autor, 2021.

APÊNDICE L - FLUXO

Quadro 37 - Função initAgente.

initAgente
<pre>function initAgente() { //console.log("Criando agente...") let agente = { pos: null, cor: null, vetor: null, path: [] } //console.log("... o agente foi criado") return agente } export default initAgente</pre>

Fonte: Elaborado pelo autor, 2021.

Quadro 38 - Função initAgentes.

initAgentes
<pre>import initAgente from './initAgente.js' import moverAgente from './moverAgente.js' import desenharAgente from './desenharAgente.js' function initAgentes(agentes, estrutura, parametros, malha, malhaCelulas , visibilidade) { let dx, dy, dz, d dx = parametros.discretizador.x dy = parametros.discretizador.y dz = parametros.discretizador.z let rx, ry, rz, r rx = parametros.refinamento.x ry = parametros.refinamento.y rz = parametros.refinamento.z d = Math.sqrt(dx * dx + dy * dy + dz * dz) r = Math.sqrt(rx * rx + ry * ry + rz * rz) let n = (dx * dy * dz * r * 4) > 1000 ? 1000 : (dx * dy * dz * r * 4) agentes = []</pre>

```

for ( let item = 0; item < n; item++ ) {
    let agente = initAgente()
    let passo = Math.floor(Math.random() * 90 + 10)
    for ( let i = 0; i <= passo ; i++) { agente = moverAgente( parametros, agente, malha,
malhaCelulas ) }
    agentes.push( agente )
}
for ( let agente of agentes ) { agente = desenharAgente( estrutura, agente, visibilidade ) }

console.log( 'Dados malhaCelulas' )
return agentes
}

export default initAgentes

```

Fonte: Elaborado pelo autor, 2021.

Quadro 39 - Função moverAgente.

```

moverAgente
import compararPosicionamento from './compararPosicionamento.js'
import verificarPosicionamentoAgente from './verificarPosicionamentoAgente.js'

function moverAgente( parametros, agente , malha, malhaCelulas ) {
    //console.log( "Movendo agente..." )
    let dx, dy, dz
    dx = malha.dimensoes.totais.x
    dy = malha.dimensoes.totais.y
    dz = malha.dimensoes.totais.z

    if ( agente.pos == null ) {
        agente.pos = { x: Math.random() * dx, y: Math.random() * dy, z: Math.random() * dz }
    }
    // armazenando a posição inicial do agente
    let posi = Object.assign( {}, agente.pos )

```

```

// coletando vetores da malha | agente.vector = {}, agente.color = {}
agente = compararPosicionamento( parametros, agente, malha, malhaCelulas )
// movendo o agente
agente.pos.x += agente.vetor.u
agente.pos.y += agente.vetor.v
agente.pos.z += agente.vetor.w
// armazenando a posição final do agente
let posf = Object.assign( {}, agente.pos )
let cor = Object.assign( {}, agente.cor )
// verificar se o agente saiu dos limites
agente = verificarPosicionamentoAgente( posi, posf, cor, agente, malha )

//console.log( "... o agente foi movido" )
return agente
}
export default moverAgente

```

Fonte: Elaborado pelo autor, 2021.

Quadro 40 - Função compararPosicionamento.

compararPosicionamento

```

function compararPosicionamento( parametros, agente, malha, malhaCelulas ) {
  //console.log( "O agente está coletando dados da malha...", parametros, malha )
  let lx, ly, lz
  lx = malha.dimensoes.discretas.x / (parametros.refinamento.x*2)
  ly = malha.dimensoes.discretas.y / (parametros.refinamento.y*2)
  lz = malha.dimensoes.discretas.z / (parametros.refinamento.z*2)
  let maxdist = Math.sqrt( lx * lx + ly * ly + lz * lz )/2
  //console.log( maxdist, malha.dimensoes.discretas, lx, ly, lz )
  for ( let celula of malhaCelulas ) {

    let dx, dy, dz, d
    dx = agente.pos.x - celula.posicionamento.x
    dy = agente.pos.y - celula.posicionamento.y

```

```

    dz = agente.pos.z - celula.posicionamento.z
    d = Math.sqrt( dx * dx + dy * dy + dz * dz )
    if ( d < maxdist ) {
        let unitario = Object.assign( {}, celula.vetor.unitario )
        agente.vetor = unitario
        agente.vetor.u *= .2
        agente.vetor.v *= .2
        agente.vetor.w *= .2
        agente.cor = celula.cor
    }
}
//console.log( "... o agente coletou dados da malha" )
return agente
}

```

export default compararPosicionamento

Fonte: Elaborado pelo autor, 2021.

Quadro 41 - Função verificarPosicionamentoAgente.

verificarPosicionamentoAgente

```

function verificarPosicionamentoAgente( posi, posf, cor, agente, malha ) {
    //console.log( "Comparando posição do agente com os limites da malha..." )
    let dx, dy, dz
    dx = malha.dimensoes.totais.x
    dy = malha.dimensoes.totais.y
    dz = malha.dimensoes.totais.z
    if ( posf.x <= 0 || posf.x >= dx || posf.y <= 0 || posf.y >= dy || posf.z <= 0 || posf.z >= dz ) {
        //console.log( "ForaDosLimites!, resetando posição do agente..." )
        agente.pos = null
        //console.log( "... a posição do agente foi resetada." )
    } else {
        //console.log( "Atualizando o Caminho percorrido pelo agente..." )
        agente.path.push( { posi: posi, posf: posf, cor: cor } )
    }
}

```

```

    //console.log( "... o Caminho percorrido pelo agente foi atualizado" )
  }

  //console.log( "... foi comparada a posição do agente com os limites da malha.")
  return agente
}

export default verificarPosicionamentoAgente

```

Fonte: Elaborado pelo autor, 2021.

Quadro 42 - Função desenharAgente.

```

desenharAgente

import * as THREE from '../three.module.js'

function desenharAgente( cena, agente, visibilidade ) {
  //console.log( 'Desenhando agente...' )

  let v1 = visibilidade.v1
  let v2 = visibilidade.v2
  let v3 = visibilidade.v3
  let v4 = visibilidade.v4
  let v5 = visibilidade.v5
  let v6 = visibilidade.v6

  let index = 0
  let posicionamentoConexoes = new Float32Array( agente.path.length * 6 )
  let coresConexoes = []

  for ( let i in agente.path ) {
    posicionamentoConexoes[ ( 6 * i ) + 0 ] = agente.path[i].posi.x
    posicionamentoConexoes[ ( 6 * i ) + 1 ] = agente.path[i].posi.y
    posicionamentoConexoes[ ( 6 * i ) + 2 ] = agente.path[i].posi.z

```

```
posicionamentoConexoes[ ( 6 * i ) + 3 ] = agente.path[i].posf.x  
posicionamentoConexoes[ ( 6 * i ) + 4 ] = agente.path[i].posf.y  
posicionamentoConexoes[ ( 6 * i ) + 5 ] = agente.path[i].posf.z
```

```
let t = agente.path[i].cor.t  
let intervalo = 1 / 6
```

```
if ( t >= 0 && t < 1 * intervalo ) {  
  if ( v1 == true ) {  
    coresConexoes.push( agente.path[i].cor.r * 255 )  
    coresConexoes.push( agente.path[i].cor.g * 255 )  
    coresConexoes.push( agente.path[i].cor.b * 255 )  
  
    coresConexoes.push( agente.path[i].cor.r * 255 )  
    coresConexoes.push( agente.path[i].cor.g * 255 )  
    coresConexoes.push( agente.path[i].cor.b * 255 )  
  } else {  
    coresConexoes.push( 0 )  
    coresConexoes.push( 0 )  
    coresConexoes.push( 0 )  
  
    coresConexoes.push( 0 )  
    coresConexoes.push( 0 )  
    coresConexoes.push( 0 )  
  }  
}  
  
if ( t >= 1 * intervalo && t < 2 * intervalo ) {  
  if ( v2 == true ) {  
    coresConexoes.push( agente.path[i].cor.r * 255 )
```

```

coresConexoes.push( agente.path[i].cor.g * 255 )
coresConexoes.push( agente.path[i].cor.b * 255 )

coresConexoes.push( agente.path[i].cor.r * 255 )
coresConexoes.push( agente.path[i].cor.g * 255 )
coresConexoes.push( agente.path[i].cor.b * 255 )
} else {
coresConexoes.push( 0 )
coresConexoes.push( 0 )
coresConexoes.push( 0 )

coresConexoes.push( 0 )
coresConexoes.push( 0 )
coresConexoes.push( 0 )
}

}

if ( t >= 2 * intervalo && t < 3 * intervalo ) {
if ( v3 == true ) {
coresConexoes.push( agente.path[i].cor.r * 255 )
coresConexoes.push( agente.path[i].cor.g * 255 )
coresConexoes.push( agente.path[i].cor.b * 255 )

coresConexoes.push( agente.path[i].cor.r * 255 )
coresConexoes.push( agente.path[i].cor.g * 255 )
coresConexoes.push( agente.path[i].cor.b * 255 )
} else {
coresConexoes.push( 0 )
coresConexoes.push( 0 )
coresConexoes.push( 0 )

coresConexoes.push( 0 )
coresConexoes.push( 0 )

```

```

        coresConexoes.push( 0 )
    }

}

if ( t >= 3 * intervalo && t <= 4 * intervalo ) {
    if ( v4 == true ) {
        coresConexoes.push( agente.path[i].cor.r * 255 )
        coresConexoes.push( agente.path[i].cor.g * 255 )
        coresConexoes.push( agente.path[i].cor.b * 255 )

        coresConexoes.push( agente.path[i].cor.r * 255 )
        coresConexoes.push( agente.path[i].cor.g * 255 )
        coresConexoes.push( agente.path[i].cor.b * 255 )
    } else {
        coresConexoes.push( 0 )
        coresConexoes.push( 0 )
        coresConexoes.push( 0 )

        coresConexoes.push( 0 )
        coresConexoes.push( 0 )
        coresConexoes.push( 0 )
    }
}

if ( t > 4 * intervalo && t <= 5 * intervalo ) {
    if ( v5 == true ) {
        coresConexoes.push( agente.path[i].cor.r * 255 )
        coresConexoes.push( agente.path[i].cor.g * 255 )
        coresConexoes.push( agente.path[i].cor.b * 255 )

        coresConexoes.push( agente.path[i].cor.r * 255 )
        coresConexoes.push( agente.path[i].cor.g * 255 )
        coresConexoes.push( agente.path[i].cor.b * 255 )
    } else {

```

```

coresConexoes.push( 0 )
coresConexoes.push( 0 )
coresConexoes.push( 0 )

coresConexoes.push( 0 )
coresConexoes.push( 0 )
coresConexoes.push( 0 )
}

}

if ( t > 5 * intervalo && t <= 6 * intervalo ) {
    if ( v6 == true ) {
        coresConexoes.push( agente.path[i].cor.r * 255 )
        coresConexoes.push( agente.path[i].cor.g * 255 )
        coresConexoes.push( agente.path[i].cor.b * 255 )

        coresConexoes.push( agente.path[i].cor.r * 255 )
        coresConexoes.push( agente.path[i].cor.g * 255 )
        coresConexoes.push( agente.path[i].cor.b * 255 )
    } else {
        coresConexoes.push( 0 )
        coresConexoes.push( 0 )
        coresConexoes.push( 0 )

        coresConexoes.push( 0 )
        coresConexoes.push( 0 )
        coresConexoes.push( 0 )
    }

}

}

```

```

let conexoesMaterial, conexoesGeometria, conexoesMesh

```

```

    conexoesGeometria = new THREE.BufferGeometry()
    conexoesGeometria.setAttribute(                                'position',                                new
THREE.Float32BufferAttribute( posicionamentoConexoes, 3 ) )
    conexoesGeometria.setAttribute(                                'color',                                new
THREE.Uint8BufferAttribute( coresConexoes, 3, true ) )
    conexoesGeometria.setDrawRange( 0, agente.path.length * 6 )
    conexoesMaterial = new THREE.LineBasicMaterial({
        vertexColors: true,
        blending: THREE.AdditiveBlending,
        transparent: true,
        opacity: .7
    })
    conexoesMesh = new THREE.LineSegments( conexoesGeometria, conexoesMaterial )
    agente.mesh = conexoesMesh
    agente.passo = Math.floor( Math.random() * 5 )
    agente.way = 0
    cena.add( agente.mesh )

    //console.log( '... Agente desenhado' )

    return agente
}

export default desenharAgente

```

Fonte: Elaborado pelo autor, 2021.