INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE - IFSUL, CÂMPUS PASSO FUNDO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Roger....

Título do trabalho

Passo Fundo
2023

Roger....

Título do trabalho

Projeto de pesquisa submetido como requisito parcial para a aprovação na disciplina de Trabalho de Conclusão I do Curso de Ciência da Computação, do Instituto Federal Sul-Rio-Grandense, Câmpus Passo Fundo.

Orientador: Me. José Antônio Oliveira de Figueiredo

Coorientador: Me. Juscelino ????

Passo Fundo

2023

SUMÁRIO

1 TEMA

Simulação Computacional

1.1 Delimitação do tema

Detecção de colisão entre objetos em simulações computacionais

2 PROBLEMA

Estudar e avaliar os mecanismos para detecção de colisão em uma simulação computacional que busca apresentar objetos de forma mais realística possível.

3 OBJETIVOS

3.1 Objetivo Geral

3.2 Objetivos Específicos

1. Conhecer as principais etapas de uma simulação física computacional.
2. Avaliar o funcionamento dos algoritmos sort and sweep e octree na fase de detecção de possíveis colisões em uma etapa de simulação.
3. Avaliar o funcionamento de algoritmos para detecção de colisão entre objetos de geometrias específicas.
4. Avaliar o método Impulse para a resolução de colisão entre dois objetos em uma etapa de simulação.

4 JUSTIFICATIVA

Segundo Bourg e Bywalec (2013), "a detecção de colisão é um problema de geometria computacional que determina se ocorreu, e onde ocorreu, uma ou mais colisões entre objetos" da simulação. Com base nisto, podemos afirmar que existem diversos algoritmos para detecção de colisão entre objetos de geometrias específicas, e também diferentes técnicas para otimizar esta detecção tornando-a mais eficiente, rápida e robusta.

Ericson (2004) afirma que apesar de hoje em dia os computadores serem extremamente rápidos, a detecção de colisão continua sendo um problema fundamental. Isso ocorre porque jogos e outras simulações são frequentemente construídos em um ambiente 3D, com centenas de milhares de polígonos, exigindo assim, algoritmos e estruturas de dados mais sofisticados para operar, em tempo real, com conjuntos de dados dessa magnitude.

Por ser uma área ainda em expansão, pouco explorada e conhecida, este estudo se justifica pois novos métodos poderão surgir, havendo possibilidades para vários trabalhos de pesquisa e desenvolvimento.

5 REFERENCIAL TEÓRICO

5.1 Simulação Computacional

Segundo Durán (2018) A simulação computacional é uma técnica amplamente utilizada para o estudo de sistemas complexos que por diversos fatores não podem ser facilmente reproduzidos devido a custos elevados , ou pelo tempo necessário para que se realize os experimentos para a obtenção de resultados precisos.

A simulação,permite que um pesquisador avalie diversos cenários diferentes para sua pesquisa apenas alterando suas variáveis de controle reduzindo a sim os custos e os riscos de um experimento físico. Essa técnica pode ser aplicada em diversas áreas como biologia, engenharia, física, química entre outras.

Para construir uma simulação, torna-se necessária a representação matemática ou computacional do comportamento do sistema a ser simulado. Isso inclue mas não se limita a variáveis de controle que podem alterar o ambiente simulado, e os objetos que serão simulados durante algumas etapas, ou por um período de tempo.

A simulação é uma poderosa ferramenta que contribue na descoberta de novas tecnologias, ou no aprimoramento das existentes.

5.1.1 Representando Formas Geométricas Computacionalmente

Na computação, não existe maneira correta ou incorreta de representar a geometria. A forma usada dependerá do nível de detalhamento desejado. É possível fazer esta representação de diversas maneiras, por exemplo: forma vetorial, paramétrica ou de forma poligonal.

Neste estudo será adotada a representação vetorial, devido a sua simplicidade e fácil compreensão. O trabalho será desenvolvido com 3 geometrias principais, que são: esferas, caixas e poliedros.

Para representação dessas formas geométricas, são utilizados pontos de coordenadas cartesianas, definidos em uma estrutura de vetor tridimensional. O código a seguir demonstra essa estrutura.

Algoritmo 1 – Código de exemplo

```
class vector3d{
float x;
float y;
```

```
float z;

vector3d(float x, float y, float z){
this->x=x;
this->y=y;
this->z=z;
}
};
```

Onde x representa o eixo que vai da esquerda para a direita; y representa o eixo perpendicular ao eixo x; E z é o eixo que indica a profundidade apontando para cima.

Lembrando também que os vetores tem diversas operações relacionadas como soma, subtração, multiplicação, divisão, Comprimento ou módulo, produto escalar, produto cruzado e produto triplo escalar que são relevantes para este estudo.

5.1.2 Esferas

As esferas são simples de representar computacionalmente, e também incrivelmente fáceis de se computar. Para representarmos uma esfera, tudo o que precisamos é de um ponto que representará o centro da esfera, e um raio, que indica o quanto a esfera se estende em todas as direções a partir do centro.

Algoritmo 2 – Código de exemplo

```
class sphere3d {
public:
vector3d center;
float radius;
sphere3d(const vector3d& center, float radius)
{
this->center=center;
this->radius=radius;
}
};

sphere3d s({0.0f, 0.0f, 0.0f}, 5.0f);
```

A sim temos uma esfera cujo o centro está posicionado exatamente na origem de nosso sistema cartesiano, e que se estende por um raio de 5 em todas as direções.

5.1.3 Caixas

As caixas são trivialmente representáveis de diversas formas. Com elas podemos representar formas cuboides facilmente. Ela normalmente é representada por dois vetores, onde um indica o canto inferior esquerdo, e o outro o canto superior direito. No livro Real Time Collision Detection, Christer Ericson demonstra várias maneiras de se representar AABBS, que nada mais são que caixas. Outras formas populares de se representar caixas são: Ponto central mais meias larguras que se estenderão do centro até as bordas da caixa. Ou a forma que usaremos que consiste em um ponto mínimo e um vetor representando a largura, altura e comprimento da caixa respectivamente. A seguir as três formas descritas são representadas. Ambas fornecem vantagens e desvantagens dependendo do contexto em que forem utilizadas.

Caixa representada pelo ponto mínimo e máximo:

Algoritmo 3 – Código de exemplo

```
class box3d_1{
vector3d min;
vector3d max;
};
```

Caixa com meia larguras

Algoritmo 4 – Código de exemplo

```
class box3d_2
{
vector3d center;
vector3d radius;
};
```

Caixa com ponto mínimo e medidas completas:

Algoritmo 5 – Código de exemplo

```
class box3d_3 {
vector3d min;
vector3d measures;
};
```

Todas elas servem para representar a mesma coisa, mas podem ser utilizadas em contextos diferentes dependendo o requisito. A box3d$_1$ é *excelente para aabbs porque já tem seus limites bem de*

Os 3 exemplos abaixo criam 3 caixas exatamente com os mesmos pontos finais.

Algoritmo 6 – Código de exemplo

```
box3d_1 b1({10,10,10}, {20,20,20});
box3d_2 b2({15,15,15}, {5,5,5});
box3d_3 b3({10,10,10}, {10,10,10});
```

Todas as 3 representações terão seus limites entre 10 e 20.

5.1.4 Poliedros

Os poliedros nada mais são que uma nuvem de pontos que representam uma geometria específica. Os poliedros certamente são a maneira mais precisa de representar geometria, porém eles são uma faca de dois gumes. Quanto mais detalhado o poliedro, mais caro será para computar colisões, rotações e translações. Existem dois tipos de poliedros. Poliedros convexos e poliedros côncavos. Poliedros convexos são poliedros que são coplanares ou que possuem suas faces voltadas para fora. Por outro lado, poliedros côncavos possuem pelo menos uma de suas faces voltada para o interior. Neste trabalho não usaremos poliedros pois eles utilizam algoritmos mais sofisticados para operarem sobre nuvens de pontos para detectar colisões. Alguns dos algoritmos mais famosos para isto incluem o GJK(GilbertJohnsonKeerthi), ou o Teorema dos eixos separadores, na sigla em inglês (SAT).

5.2 Estruturas para delimitação de volumes

Em uma simulação,temos os mais variados objetos interagindo uns com os outros. Como por exemplo, uma esfera colidindo contra um triângulo representado de forma poligonal. Na simulação, teremos diversos objetos e precisamos testar a colisão entre um dos objetos contra todos os outros com seus algoritmos e métodos específicos, o que pode ser computacionalmente caro e custoso.

É para resolver esse problema de executar algoritmos caros a todo instante que surgiram os volumes delimitadores. É escolhida uma forma arbitrária para representar um volume delimitador para encapsular a geometria específica.

Existem vários tipos de volume delimitadores como esferas, retângulos, e cascas convexas. Todas elas tem suas desvantagens e vantagens como por exemplo, um volume delimitador como uma esfera é extremamente simples e rápido de se calcular, porém, ela é autamente imprecisa para detectar colisões gerando diversos falsos positivos.

A forma mais precisa é uma casca convexa, porém, ela é cara de se computar. Então um retângulo fornece um bom equilíbrio entre precisão e velocidade sendo a sim, a forma preferida de volume delimitador.

Depois de escolhida a forma do volume delimitador que aqui foi escolhida o retângulo,

surge outro problema. Que tipo de volume delimitador queremos? Um AABB, ou um OBB? Novamente, ambos tem suas vantagens e desvantagens que são explicadas a seguir:

5.2.1 AABB

Aligned Axis Bounding Box é uma caixa retangular alinhada aos três eixos do sistema de coordenadas, por tanto, ela não pode sofrer rotações. Então, caso a geometria contida por este volume delimitador sofra uma rotação, o volume delimitador deve ser reconstruído para acomodar a geometria rotacionada. Isso pode ser especialmente custoso de se computar em poliedros com muitos vértices.

No entanto, a vantagem é que o teste de colisão é eficiente e relativamente simples de se computar.

Algoritmo 7 – Código de exemplo

```
class AABB
{
public:
vector3d min;
vector3d max;
GeometricShape* shape;
};

sphere3d* s=new sphere3d({5, 5, 5}, 5);
AABB* aabb=new AABB(s);
```

A bounding box desta esfera será 2 vetores que envolverão completamente em um cubo. Seus vetores respectivamente serão: min=0,0,0 max=10,10,10

5.2.2 OBB

Um Oriented Bounding box tem a mesma função de um AABB, porém, ele não está preso a nenhum eixo e é invariável a rotação dos objetos pois ele também é capaz de rotacionar acompanhando a rotação do objeto contido. Ele é representado de forma diferente de um aabb com um ponto central, as meia medidas para os eixos, e uma matriz de rotação para servir como orientação do obb. É possível representar a matriz de rotação como ângulos de Euler para economizar memória, porém, mais tarde ao atualizar a rotação, terá que converter os ângulos em uma matriz. Como Christer Ericson comenta em seu livro, Obbs e conversões de ângulos é surpreendentemente caro e complicado.

Como tudo na computação tem seus prós e contras, obbs são geralmente mais precisos que AABBS, porém, são mais caros de se computar, e também mais difíceis de se implementar.

Algumas representações típicas de obbs se seguem:

Algoritmo 8 – Código de exemplo

```
class obb_1
{
public:
vector3d center;
vector3d measures;
matrix3x3 orientation;
};
```

Algoritmo 9 – Código de exemplo

```
class obb_2
{
public:
vector3d center;
vector3d measures;
vector3d angles;
};
```

Para realizar um teste de sobreposição obb obb é necessário utilizar o teorema dos eixos separadores que diz que se existir ao menos um plano que divida os dois objetos, eles não estão colidindo.

5.2.3 Corpo Rígido

5.2.4 O que é um corpo rígido?

Segundo Bourg e Bywalec (2013, tradução nossa) Um corpo rígido é um conjunto de partículas que permanecem a distâncias fixas umas das outras sem nenhum tipo de translação ou rotação entre elas. Em outras palavras, um corpo rígido não mudará de forma enquanto se move, ou sua deformação é tão insignificante que pode ser desprezada.

Um corpo rígido é frequentemente representado como tendo uma posição, uma orientação e dimensões para representar seu volume. Estes são requisitos mínimos para a sua definição, porém, dependendo do nível de precisão desejada, pode-se adicionar mais atributos como tensores de inércia, velocidades lineares e angulares, coeficiente de restituição, e o que mais for necessário para que se atinja os objetivos desejados.

5.2.5 Diferença entre corpo dinâmico e corpo estático

Existem 3 classes principais de corpos que podem ser representadas em uma simulação: Corpos rígidos, que não se deformam enquanto se movem, ou caso se deformem,

são capazes de voltar a sua forma original;

Corpos macios, que podem deformar enquanto se movem ou por ação de forças ou eventos externos;

Ou corpos estáticos, que nada mais é que uma sub-classificação de um corpo rígido ou um corpo macio, mas com a ausência de massa, ficando a sim, estacionário na posição em que se encontra...

5.3 Detecção de Colisão

Segundo Bourg e Bywalec (2013), a detecção de colisão é um problema de geometria computacional que determina se e onde, dois ou mais objetos colidiram.

Uma colisão pode ser detectada de diversas maneiras dependendo da natureza dos objetos envolvidos como esferas e caixas.

Para algumas situações, uma colisão pode ser detectada se a distância entre os dois objetos está a baixo de uma determinada tolerância. Em outros casos, os objetos podem estar se sobrepondo em um ou mais pontos.

Após uma colisão ser detectada, informações devem ser coletadas para a resolução da colisão posteriormente.

Quais informações coletar depende muito da natureza do simulador. Um simulador de física por exemplo pode precisar do ponto, ou pontos de contato que fizeram os objetos colidirem, além de sua velocidade relativa no instante da colisão.

Outras informações importantes para se coletar durante a detecção de colisão inclue o vetor normal, que é um vetor unitário que apontará na direção em que a colisão aconteceu.

E por último, a profundidade de sobreposição, ou de penetração dos objetos. Esta informação é necessária para separar os objetos para que eles apenas se toquem, e não se sobreponham.

5.4 Fases da Detecção de Colisão

A detecção de colisão é dividida em duas áreas principais que são:

5.4.1 Fase Ampla

A fase ampla é responsável por determinar rapidamente pares de objetos que possivelmente estejam colidindo, e descartar aqueles objetos que não estão.

Nesta fase, os AABBS das formas geométricas são testados pois possuem algoritmos simplificados e eficientes para esta tarefa.

Quando uma colisão entre dois AABBS é detectada, a sua geometria contida pode não estar colidindo com a geometria do segundo AABB. Então estes dois objetos são agrupados para uma análise mais detalhada em uma fase posterior.

5.4.2 Fase Estreita

A fase estreita é responsável pelo trabalho mais detalhado e preciso da detecção de colisão.

É nesta fase onde os algoritmos e técnicas específicas são aplicadas para determinar se de fato, os pares de objetos suspeitos de colisão estão colidindo.

Caso estejam, informações como normal, ponto de contato e profundidade de penetração devem ser calculados para uso posterior.

5.5 Algoritmos para Fase Ampla

Um problema da detecção de colisão é que em uma simulação podem existir muitos objetos. Alguns distantes, e outros próximos.

Quando pegamos um objeto de forma aleatória deste conjunto de objetos queremos saber com quais outros objetos existe a chance de colisão. Em uma situação desta natureza,a complexidade dos testes normalmente é de N^2, $oquecomputacionalmenteémuitodispendioso, porqueos$

Para acelerar este processo, existem diversas estruturas de dados e técnicas de compartimentação espacial. A lógica destas técnicas é agrupar os objetos que estão próximos uns dos outros e interromper a busca quando o restante dos objetos estão fora de alcance.

Algumas das estruturas mais populares atualmente são as árvores octrees, grades uniformes e a divisão espacial por hash.

5.5.1 Octrees

Uma octree é uma estrutura de dados hierárquica que organiza um espaço tridimensional em células menores recursivamente. Dado um nó raiz, este nó é dividido em 8 filhos, que também podem ser subdivididos em mais 8 filhos até que um critério de parada seja satisfeito. Este critério pode ser uma profundidade máxima para a árvore, ou o tamanho do nó que será dividido atingiu um limite mínimo.

Os objetos adicionados nesta estrutura tentarão ser adicionados no menor nó que seja capaz de conter todo o volume do objeto. Com esta compartimentação espacial, os objetos só precisam verificar a possibilidade de colisão com seus irmãos, e com os nós mais profundos acelerando assim o processo.

5.5.2 Ordenar e Varrer

5.6 Resolução de Colisão

6 METODOLOGIA

A proposta de desenvolvimento deste projeto é um programa gráfico que executará simulações de movimento em ambientes 3d, para analisar os mecanismos de detecção de colisão. Este programa será implementado com algumas leis básicas da física, para que os objetos simulados reajam de forma mais realista, tornando assim, a simulação mais dinâmica. O usuário terá a opção de definir as variáveis de seu ambiente de simulação, como limites espaciais cujo os objetos podem percorrer, definir obstáculos, aplicar a força da gravidade, definir o passo de tempo da simulação, bem como configurar individualmente cada objeto presente na mesma. Os objetos simulados terão os seguintes parâmetros configuráveis: massa, posição, velocidade, coeficiente de restituição e forma. Os parâmetros serão configuráveis por meio de um painel. O programa será organizado em 3 partes principais: Tela principal: Será onde a simulação será desenhada, mostrando ao usuário o andamento da simulação. A tela principal também terá um menu permitindo o controle de diversos aspectos da simulação. Configuração do ambiente: Esta tela será responsável por configurar o ambiente de simulação. Propriedades como limites do ambiente, passo de tempo, e se a gravidade deve ou não ser aplicada são exemplos de propriedades configuráveis. Também nesta tela, terá a lista de objetos que poderão ser controlados. Engine de simulação: Esta parte do programa será subdividida em simulação do movimento e detecção de colisão. Simulação do movimento: Esta etapa é responsável pelos cálculos que modelam o movimento dos objetos presentes no ambiente da simulação, utilizando para isto as leis de Newton. No entanto, a única restrição existente é que os objetos estáticos não devem ser movimentados. Pois, eles modelam obstáculos ou a geografia do ambiente. Detecção de colisão: A detecção de colisão é a etapa responsável por aplicar algoritmos específicos nos objetos a fim de determinar se existe uma sobreposição, ou se a distância entre 2 objetos é inferior a uma determinada tolerância. Esta etapa também é responsável por resolver a colisão caso exista. Modelagem preliminar do simulador:

Nesta cessão, a modelagem preliminar da engine de simulação será discutida apresentando seus principais componentes.

Algoritmo 10 – Código de exemplo

```
classe vetor3d
{
eixo_x
eixo_y
eixo_z
magnitude()
```

```
normalizar()
inverso()
produto_escalar()
produto_vetorial()
produto_triplo_escalar()
};
```

A classe vetor3d representará as coordenadas cartesianas tridimensionais no espaço. Existem 3 componentes fundamentais: eixo$_X$: *Representaadistânciadopontoemrelaçãoaumaorigemd*
Representaverticalmenteadistânciadopontoatéaorigem.eixo$_z$: É*oeixoperpendicularaoeixoxeeixoy.*

O vetor também tem diversas operações associadas como soma, subtração, multiplicação e divisão. Também possui métodos específicos como normalização, magnitude, inverso e seus produtos como produto vetorial e escalar.

Algoritmo 11 – Código de exemplo

```
classe FormaGeometrica
{
tipo
recuperar_centroide()
calcular_suporte()
transladar(vetor v)
escalar(s)
rotacionar(orienta  o)
converter_string()
};
```

A classe de forma geométrica é uma base para as próximas implementações de esferas e caixas. Ela declara padrões que devem ser seguidos pelas classes filhas.

Algoritmo 12 – Código de exemplo

```
classe esfera estende FormaGeometrica
{
ponto_central
raio
};
```

A classe esfera tem 2 atributos principais que são: Ponto$_c$*entral* : É*umvetorquerepresentaascoo*
Representaoquantoaesferaseestendeemtodasasdireçõesapartirdocentro.

Algoritmo 13 – Código de exemplo

```
classe caixa  estende FormaGeometrica
```

```
{
m nimo
arestas
};
```

A classe caixa tem duas propriedades que também são importantes: Mínimo: É um vetor que representa o canto inferior esquerdo da caixa. Arestas: É o quanto as arestas se estendem nos 3 eixos a partir do canto inferior esquerdo...

Algoritmo 14 – Código de exemplo

```
classe AABB
{
min
max
FormaGeometrica
transla  o(vetor v)
escala(s)
escala(vetor v)
computar_volume_delimitador()
};
```

Esta classe representa uma caixa alinhada ao eixo (AABB) Ela é utilizada para envolver completamente uma forma geométrica, seja ela esfera, caixa ou poliedro. A classe possui 3 atributos principais: min: É um vetor representando o canto inferior esquerdo do aabb... Max: É um vetor representando o canto superior direito do aabb... FormaGeometrica: É a forma geométrica que o aabb está contendo... Também possui métodos para redimensionar e transladar o AABB.

Algoritmo 15 – Código de exemplo

```
classe colisao_info
{
ponto_de_colisao
normal_da_colisao
profundidade
objeto_1
objeto_2
};
```

A classe CollisionInfo é responsável por conter informações sobre a colisão que foi detectada como ponto de colisão, normal, profundidade, e os objetos envolvidos na colisão.

Algoritmo 16 – Código de exemplo

```
classe colisao
{
esfera_esfera(s1, s2, info)
esfera_caixa(s, c, info)
caixa_caixa(c1, c2, info)
esfera_poliedro(s, p, info)
caixa_poliedro(c, p, info)
poliedro_poliedro(p1, p2, info)
};
```

A classe de colisão irá conter os métodos específicos de colisão entre as formas geométricas específicas como esferas x esferas, caixas x caixas e poliedros x poliedros.

Algoritmo 17 – Código de exemplo

```
classe corpo_rigido_interface
    {
massa
restitui  o
nome
aabb
posi  o
    };
```

Esta interface define alguns atributos básicos que será necessário ao modelar o objeto. Se precisarmos de atributos mais específicos, ou se quisermos expandir nossa simulação futuramente, basta sub-classificarmos esta interface e realizar as mudanças necessárias.

Algoritmo 18 – Código de exemplo

```
classe fase_ampla
{
escanear(lista_de_objetos, lista_de_possiveis_colis es)
};
```

Esta interface representa o nosso algoritmo de broadphase. Como existem diversos algoritmos, optou-se de criar uma interface e realizar a implementação separadamente. O método scan recebe um vetor de corpos rígidos para verificar, e também uma lista onde será armazenado informações sobre uma possível colisão...

Algoritmo 19 – Código de exemplo

```
classe fase_estreita
    {
detectar_colis es(lista_de_possiveis_colis es , lista_de_colis es)
    };
```

Esta classe implementa o algoritmo de fase estreita da mesma forma que a interface de fase ampla. Este algoritmo recebe uma lista de possíveis colisões e sua tarefa é aplicar os testes de detecção de colisão entre geometrias específicas e avaliar se de fato, o par testado está colidindo.

Algoritmo 20 – Código de exemplo

```
classe solucionador_de_colis o
    {
resolver(lista_de_colis es)
resolver_par(objeto_1 , objeto_2 , info)
    };
```

Esta classe irá resolver as colisões detectadas. Ela age separando os corpos se existir sobreposição, e aplicando impulso...

Algoritmo 21 – Código de exemplo

```
classe integrador_num rico
{
mover(lista_de_objetos , tempo)
mover_objeto(objeto , tempo)
};
```

Esta classe implementa o algoritmo que fará os objetos se movimentarem na simulação. Para isso serão utilizadas equações das leis de Newton.

Algoritmo 22 – Código de exemplo

```
classe octree
{
raiz
adicionar_objeto(objeto)
remover_objeto(objeto)
fase_ampla(lista_de_objetos , lista_de_poss veis_colis es)
fase_ampla(objeto , lista_de_poss veis_colis es)
}

A classe octree    utilizada na fase ampla, mas como ela    relativam
```

```
se construir, ela    utilizada em objetos est ticos como obst culos
A classe tem 2 m todos de fase ampla. O primeiro, recebe uma lista d
que queremos que sejam testados, e a sa da    uma lista com todas as
detectadas. O segundo m todo, recebe um objeto e uma lista de sa da

\begin{lstlisting}[frame=single,caption=C digo de exemplo\label{codi
class ambiente_de_simula  o estende AABB
{
tempo_atual
gravidade
lista_de_objetos
fase_ampla
fase_estreita
solucionador_de_colis es
integrador_num rico
octree
adicionar_objeto(objeto)
remover_objeto(objeto)
passo_de_simula  o(tempo)
};
```

Esta classe é responsável pela simulação do ambiente. Para isso recebe e manipula todos os objetos instanciados pelas classes anteriormente descritas. A classe permite adicionar e remover objetos, e também definir os algoritmos utilizados para realizar as tarefas. O método de atualização será chamado sempre que um passo na simulação for avançado. Ele primeiramente movimentará os objetos, e então fará a fase ampla, fase estreita, e resolverá as colisões...

Requisitos do Projeto

Compilador compatível com c++20 ou superior; Compilador compatível com C2017 ou posterior; wxwidgets para a criação das telas e controles como listas, botões e demais controles... opengl para a renderização gráfica; wxglade para construir a interface gráfica do usuário (GUI) fmod e bass para indicações sonoras;

6.1 Cronograma

Agosto: primeira quinzena:

Configuração do ambiente de desenvolvimento e modelagem das classes do simulador.

segunda quinzena: Revisão e testes das classes desenvolvidas Registros e documen-

tação parcial para composição do artigo final

Setembro primeira quinzena:

Desenvolvimento da janela principal do aplicativo e seus diálogos.

segunda quinzena: Desenvolvimento da animação gráfica dos objetos Registro sobre o andamento do projeto para a composição do artigo final

Outubro primeira quinzena Refinamento de todo o projeto desenvolvido e realizar a integração dos componentes.

segunda quinzena Finalização do protótipo para testes e correções de bugs Registro do progresso para a composição do artigo final

Novembro Primeira quinzena:

Testes do protótipo Escrita do artigo final

Segunda quinzena:

Término do artigo final Dezembro primeira quinzena: Finalização e entrega do artigo final Defesa do Trabalho de conclusão de curso

REFERÊNCIAS

BOURG, D. M.; BYWALEC, B. *Physics for Game Developers: Science, math, and code for realistic effects*. [S.l.]: "O'Reilly Media, Inc.", 2013. Citado na página 3.

DURÁN, J. M. *Computer simulations in science and engineering: Concepts-Practices-Perspectives*. [S.l.]: Springer, 2018. Citado na página 4.

ERICSON, C. *Real-time collision detection*. [S.l.]: Crc Press, 2004. Citado na página 3.