Universidade Federal do Rio de Janeiro
Escola Politécnica
Departamento de Eletrônica e de Computação

# Simulador de Boliche 3D com OpenGL

Autor: _____
Diego Ribeiro Bina

Orientador: _____
Prof. Sérgio Barbosa Villas-Boas, Ph. D.

Examinador: _____
Prof. Flávio Luis de Mello, D. Sc.

Examinador: _____
Prof. Edilberto Strauss, Ph. D.

DEL

Janeiro de 2010

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

## AGRADECIMENTO

Dedico este trabalho aos meus pais, que sempre me apoiaram e me ajudaram no que puderam. Em especial, à minha mãe, que sempre deu a vida por seus filhos. Também foram importantes meu irmão, um grande amigo, e meus amigos. Além deles, devo agradecer ao povo brasileiro, que possibilitou a minha formação nesta universidade pública.

**RESUMO**

Este trabalho descreve o meu projeto final do curso de Engenharia Eletrônica e de Computação da UFRJ. Ele utiliza conhecimentos de programação (C++) e computação gráfica para criar um jogo de boliche 3D. Seu objetivo é reunir vários conhecimentos adquiridos durante o curso para criar, com poucos recursos, um jogo que entretenha e divirta os jogadores. O fator diversão é o principal objetivo.

Palavras-Chave: computação gráfica, programação, jogo, ferramentas multi-plataforma, entretenimento.

## ABSTRACT

This work describes my Undergraduation Project for UFRJ's Electronic and Computer Engineering course. It uses programming skills (C++) and computer graphics knowledge to create a bowling 3D game. Its purpose is to gather the knowledge gained during the course in order to build, with limited resources, a game that can be entertaining and fun. The fun factor is the main goal.



Key-words: computer graphics, programming, game, cross-platform tools, entertainment

## SIGLAS

UFRJ – Universidade Federal do Rio de Janeiro

OpenGL – Open Graphics Layer

GUI – Graphic User Interface

API – Application Programming Interface

IA – Inteligência Artificial

PNG – Portable Network Graphics

JPEG – Joint Photographic Experts Group

L-GPL – GNU Lesser General Public License

HTML – HyperText Markup Language

ODBC – Open Data Base Connectivity

CAD – Computer-Aided Design

# Sumário

# Lista de Figuras

# Capítulo 1

# Introdução

## 1.1 – Tema

O trabalho tem como tema o desenvolvimento de jogos 3D para Windows utilizando OpenGL, uma API de computação gráfica já amplamente difundida, wxWidgets, uma renomada biblioteca open source multi-uso, e Bullet Physics, uma engine de física open source que vem ganhando espaço nos últimos anos. Neste sentido, pretende-se criar um jogo que gere diversão e entretenimento aos usuários.

Este jogo foi denominado BowlingGL em alusão à utilização da OpenGL nos gráficos 3D do jogo.

## 1.2 – Delimitação

O desenvolvimento restringe-se ao ambiente Windows, detentor da imensa maioria dos usuários de computador. Embora a linguagem de programação e as bibliotecas utilizadas não impeçam a ampliação do escopo para outros sistemas operacionais como Linux e Mac, este projeto não faz uso dessa capacidade, limitando-se a Windows. Também, este projeto não inclui plataformas de celular, vistas as limitações de hardware presentes e a incompatibilidade de algumas das ferramentas utilizadas.

## 1.3 – Justificativa

A indústria do entretenimento vem se desenvolvendo em grande escala nos últimos tempos. Em poucas décadas essa indústria deixou de ser vista como um coadjuvante e passou a ser encarada como um dos maiores setores da indústria mundial, recebendo investimentos maciços e gerando bilhões por ano.

Dentro dela, a indústria de jogos eletrônicos é uma das áreas que mais se desenvolveram. Em poucas décadas, seus produtos passaram de jogos de porão a megaproduções geradoras de milhões, constituindo parte do entretenimento pessoal, não apenas de crianças, mas também de jovens e adultos.

Uma área que evoluiu muito nos últimos anos e contribuiu muito para o desenvolvimento da indústria do entretenimento, principalmente da indústria de filmes e a de jogos eletrônicos, foi a Computação Gráfica. Aproveitando as contínuas melhorias de hardware, a Computação Gráfica chegou a um nível de realismo espantoso, simulando de forma quase perfeita o mundo real. Praticamente todos os filmes e jogos a utilizam de forma parcial ou total, criando por computador cenas inviáveis ou impossíveis de serem filmadas ou até mundos virtuais inteiros.

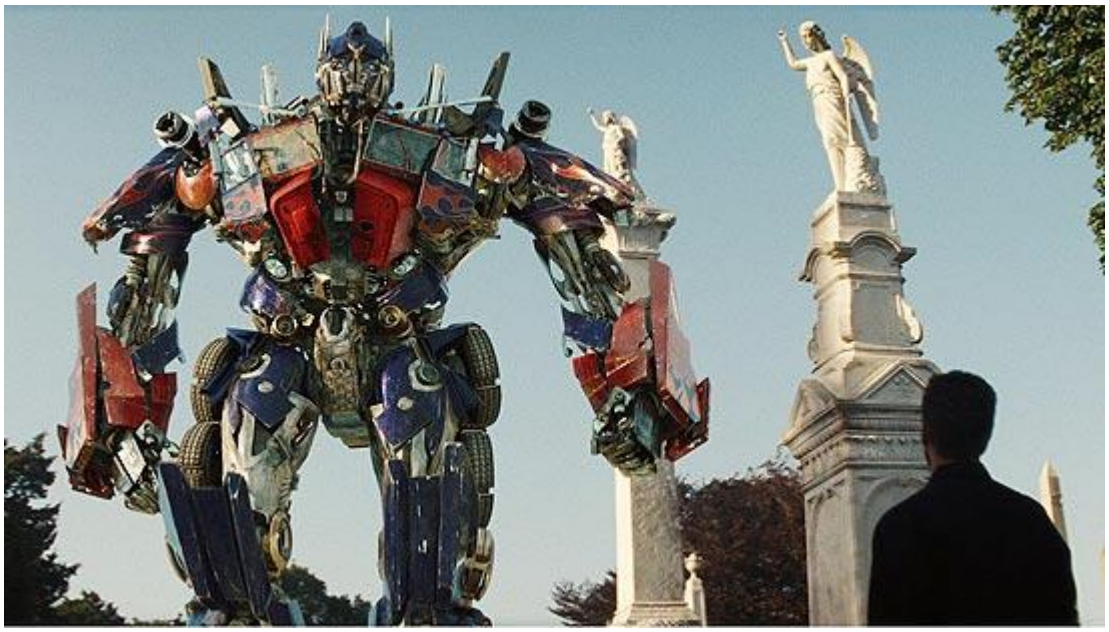

**Figura 1.1 - Cena do Filme Transformers**
**Esse filme utiliza amplamente efeitos de Computação Gráfica**

Dentro desse contexto, o presente projeto visa o estudo do desenvolvimento de jogos comerciais e da Computação Gráfica como partes importantes da indústria mundial.

## 1.4 – Objetivos

O objetivo geral é criar um jogo de boliche 3D em uma interface GUI para Windows, direcionado para o entretenimento. Desta forma, têm-se como objetivos específicos:

(1) Elaborar um jogo 3D de boliche que, logicamente, respeite as regras do esporte e utilize uma física compatível para o movimento dos objetos e colisões;

(2) Possibilitar o funcionamento do jogo em janelas GUI do Windows, adicionando menus e opções.

## 1.5 – Metodologia

Este projeto utiliza a linguagem de programação C++, sendo desenvolvido com Orientação a Objetos e tendo como base a biblioteca de gráficos OpenGL. A versão da biblioteca utilizada é a 2.8.9, a mais recente até a data do início do desenvolvimento.

Para desenvolver a interface gráfica, fez-se uso da biblioteca para C++ wxWidgets, a qual dispõe de uma classe chamada wxGLCanvas, que tem função exclusiva de fornecer um ambiente para o desenho de gráficos na janela do programa. A biblioteca OpenGL é usada dentro de um objeto dessa classe. Desta forma, através da wxWidgets pode-se aliar a renderização 3D da OpenGL às interfaces gráficas tradicionais do Windows (janelas, menus, etc). Outra biblioteca usada é a Bullet Physics. Ela fornece as simulações de física necessárias para as colisões e movimentos da bola e dos pinos.

Por opção estratégica, escolheu-se usar a OpenGL diretamente, para aprender melhor o seu uso, manter a portabilidade (Windows, Linux, Mac) e reforçar o conhecimento e a integração com a wxWidgets.

O presente projeto utilizará as bibliotecas multi-plataforma mencionadas acima para desenvolver um jogo de boliche 3D para Windows em janelas típicas desse sistema operacional, contendo menus e diversas opções.

## 1.6 – Descrição

No capítulo 2 será apresentada uma fundamentação para o projeto. Será mostrado o estado da arte do desenvolvimento de jogos, explicando conceitos como game engines e como o código de um jogo normalmente é estruturado. Também serão apresentadas as 3 grandes bibliotecas utilizadas neste projeto: wxWidgets, OpenGL e Bullet Physics, sendo dada uma breve introdução a cada uma delas.

O capítulo 3 falará da implementação do projeto, explicando como ele foi feito.

O capítulo 4 fornece uma análise dos resultados do projeto.

Por fim, o capítulo 5 apresenta as conclusões, com a verificação do atendimento aos objetivos deste projeto e as possíveis melhorias a serem feitas.

# Capítulo 2

# Fundamentação

## 2.1 – Desenvolvimento de Jogos

Atualmente, o desenvolvimento de jogos pode ser feito em diversas linguagens de programação e utilizando uma série de ferramentas auxiliares, dentre as quais, as Game Engines se destacam. Nesta seção, serão explicados os conceitos comuns do desenvolvimento de um jogo.

### 2.1.1 – Linguagens de Programação e Game Engines

Em geral, qualquer linguagem orientada a objetos pode ser usada, mas a principal é C++ (utilizada neste projeto). Também são utilizadas linguagens de script, como Lua e Python, principalmente quando se trabalha com game engines. Já as ferramentas utilizadas no desenvolvimento podem ser bibliotecas, API's ou game engines.

As API's são especificações de bibliotecas, tendo implementações diferentes dependendo do sistema em que são usadas. Entre elas, as mais populares para computadores são DirectX e OpenGL (utilizada neste projeto), usadas para computação gráfica.

As game engines, por sua vez, contêm uma grande variedade de opções e são mais direcionadas ao desenvolvimento de jogos, cada uma atendendo a diferentes necessidades do desenvolvedor. Elas podem ser divididas em vários tipos, cada um atendendo a uma parte do desenvolvimento do jogo: engines gráficas, físicas, sonoras, de IA, de Rede, etc. Algumas game engines englobam mais de um tipo, atuando em diferentes partes do jogo desenvolvido. Entre esses tipos, os que estão dentro do escopo deste projeto são as engines gráficas e as físicas, já que o jogo desenvolvido não faz uso das outras funcionalidades como sons, rede ou IA.

- Engines Gráficas

São utilizadas para computação gráfica, por esse motivo muitas vezes sendo desenvolvidas a partir das API's DirectX e OpenGL. Dentre as engines com foco em

gráficos 3D em C++ (parâmetro deste projeto), duas das mais utilizadas são: a Irrlicht, open source, intuitiva e fácil de usar; e a OGRE, também open source, porém mais complexa e com bastante material de consulta.

- Engines Físicas

São utilizadas para simular a física ou apenas para realizar detecção de colisão. Algumas dessas engines em C++ são: a Newton , engine de física muito completa e fácil de usar, de uso pago; a ODE, open source, simples e fácil de usar; e a Bullet Physics (utilizada neste projeto), também open source, simples e de fácil uso.

### 2.1.2 – Estrutura do Código

Os jogos em geral possuem um loop principal, que é o que executa o jogo. Quando este loop chega ao fim, o jogo também chega. O que ele faz é periodicamente verificar os comandos do usuário e realizar ações. O seu formato é o seguinte:

```
while(usuário não pede para sair)
        verificar input do usuário (através de teclado, mouse, controle, etc...)
        realizar IA
        movimentar objetos e verificar colisões
        renderizar gráficos num buffer
        realizar o swap do display buffer
        tocar sons
end while
```

No caso específico deste projeto, os passos “realizar IA” e “tocar sons” não são realizados, já que o jogo não possui nem IA nem sons.

Uma das preocupações que se deve ter é com o frame rate do jogo, isto é, o número de quadros que serão desenhados por segundo. Deve-se fazer uma projeção do desempenho e do uso de memória, a fim de evitar sobrecarregar o processador, a GPU ou a memória do computador. Para jogos, a taxa mínima indicada para o frame rate é em torno de 30 fps [1].

A programação orientada a objetos, amplamente utilizada no desenvolvimento de jogos, facilita o desenvolvimento por permitir ao programador a abstração de classes e subclasses (polimorfismo) e a utilização de classes como modelos para a criação de objetos. Algumas classes já podem vir embutidas dentro das API’s e game engines usadas, principalmente as classes que tratam imagens, sons, modelos 3D, etc. Já as classes que fazem parte do jogo em si são criadas pelo programador.

## 2.2 – Bibliotecas, API's e Game Engines Usadas no Projeto

O jogo desenvolvido neste projeto faz uso das API's wxWigdets e OpenGL e da game engine Bullet Physics. Cada uma será brevemente apresentada.

### 2.2.1 – wxWidgets

wxWidgets é uma extensa API multi-plataforma escrita em C++ e disponível para uso com C++, Python, Perl e C#. Ela permite a desenvolvedores criar aplicativos para Windows, Mac OS X, Linux e UNIX em arquiteturas 32-bit e 64-bit, assim como várias plataformas móveis incluindo Windows Mobile e iPhone. Ela também é open source, sob a licença L-GPL.

A wxWidgets fornece aparência e comportamento nativos do sistema aos aplicativos, pois usa a API nativa da plataforma, ao invés de emular a GUI. Dentre as funcionalidades fornecidas por essa biblioteca, estão:

- Desenvolvimento de interfaces gráficas (GUI);
- Manipulação de eventos;
- Playback de som e vídeo;
- Ambiente para renderização de gráficos 3D através da OpenGL;
- Suporte a internacionalização e a Unicode;
- Sockets;
- Multithreading;
- Arquivos e sistemas de arquivos virtuais, como manipulação de arquivos zip;
- Renderização HTML, útil para janelas de 'about' melhoradas, relatórios, etc;
- Containers como arrays e listas;
- Carregar, salvar, desenhar e manipular imagens;
- Timers e manipulação de datas;
- Logging de erros;
- Clipboard e drag and drop;
- Acesso a bancos de dados ODBC.

A wxWidgets é usada por muitas empresas, interessadas principalmente nas suas funcionalidades multi-plataforma. Alguns exemplos de aplicativos desenvolvidos com a wxWidgets são: AVG AntiVirus, Forte Agent, Audacity, iPodder, e Tortoise CVS.

Dentre os benefícios da wxWidgets, estão o funcionamento multi-plataforma, com uso de elementos GUI nativos do sistema, a grande rapidez (por ter sido desenvolvida em C++), a vasta lista de funcionalidades oferecidas, e, talvez a principal, o seu uso e distribuição do produto final sem custos ao desenvolvedor.

Já os contras são relacionados ao desenvolvimento. Há vários relatos de dificuldades de instalação no Windows, com a criação de makefiles e bakefiles, necessários na compilação da biblioteca, e com a conversão entre strings em C para strings Unicode.

### 2.2.2 – OpenGL

OpenGL é uma das API's mais utilizadas para desenvolver aplicações que utilizam gráficos 2D ou 3D. Ela é open source, de livre uso e multi-plataforma, isto é, funciona em diversos sistemas operacionais, como Windows, Linux e Mac.

Esta API foi desenvolvida originalmente pela Silicon Graphics Inc. e é amplamente usada, entre outros, em CAD, realidade virtual, simulação de vôo, além de jogos eletrônicos, onde compete com a Direct3D nas plataformas Windows.

A OpenGL é uma API procedural de baixo nível, necessitando que o programador dite os passos exatos para renderizar uma cena. Isso exige do programador um bom conhecimento de seu funcionamento, mas também fornece uma maior liberdade para implementar algoritmos de renderização próprios.

O funcionamento básico da OpenGL é aceitar primitivas geométricas como pontos, linhas e polígonos e convertê-los em pixels, para serem visualizados na tela. Isto é feito por um pipeline, mostrado simplificadamente a seguir. A maioria dos comandos da OpenGL envia primitivas para esse pipeline ou configura como ele processa essas primitivas.

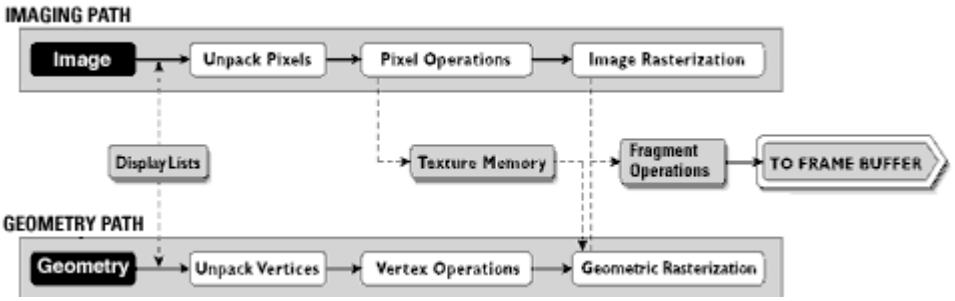


**Figura 2.1 - Pipeline Simplificado do Processo da OpenGL (Extraído da documentação da OpenGL)**

No caminho da imagem, a renderização ocorre a partir de imagens, operando-se com os seus pixels, e com a sua posterior rasterização, isto é a conversão das informações obtidas em pixels.

Já no caminho da geometria, a renderização ocorre a partir de primitivas geométricas, operando-se com vértices, por exemplo, e terminando na sua rasterização.

Após isso, ocorrem as operações de fragmentos, como, por exemplo, atualização de valores dependendo de valores de profundidade ou combinações de cores. Ao final, os dados são enviados ao frame buffer e são atualizados na tela com o método glSwapBuffers().

A OpenGL foi desenvolvida para ter apenas gráficos como saída. O núcleo da API não possui conceitos de janelas, áudio, teclado, mouse ou outros inputs. Isso permite que a renderização seja completamente independente do sistema operacional. Porém, alguma integração é necessária para permitir uma interação com o sistema hospedeiro. Isto é feito através de algumas API's adicionais: WGL (Windows), GLX (Linux) e CGL (Mac OS X). Neste projeto, no entanto, essa camada é delegada à wxWidgets, a qual fica encarregada de lidar com o sistema operacional onde a aplicação está rodando.

### 2.2.3 – Bullet Physics

A Bullet Physics é uma game engine C++ open source de física, disponível sob a licença ZLib [2], usada em jogos, simuladores e efeitos visuais de filmes. Ela é dividida em dois módulos, um para o sistema de detecção de colisões e outro para a dinâmica de corpos rígidos e flexíveis. Esses módulos são independentes entre si, o que possibilita ao

desenvolvedor substituir qualquer um deles por uma implementação que seja mais adequada ao seu sistema.

Essa engine fornece opções de detecções de colisão discreta ou contínua. Segundo Rejane Gomes [3], "[...] em algoritmos de detecção de colisão discreta, as consultas de colisão são realizadas em determinados intervalos de tempo enquanto existe movimento dos objetos. Contudo, o modelo pode mover-se muito rapidamente ou ser muito fino proporcionando facilmente uma falha na detecção de uma colisão, falha causada pelo intervalo de tempo insuficiente na chamada da execução do algoritmo". Na detecção de colisão discreta, "uma colisão só pode ser reportada depois que ela já aconteceu [...], não sendo possível reportar também o momento exato em que o contato ocorreu".

Já os algoritmos de detecção de colisão contínua "[...] consideram o movimento dos objetos e reportam o primeiro instante de contato entre dois modelos se uma colisão vier a acontecer. A grande importância desse tipo de abordagem reside no fato de que o resultado sempre garante a não interpenetração dos objetos. Porém, a maior limitação existente é que são, em geral, mais lentos do que métodos discretos".

O público alvo desta engine são desenvolvedores de jogos e de animação 3D que queiram fazer uso de suas simulações de física, além de físicos que queiram fazer experimentos com detecção de colisões e conceitos de dinâmica de corpos rígidos.

Algumas das funcionalidades da Bullet são:

- Simulações de corpos rígidos e flexíveis com detecções de colisão discreta e contínua;
- Formas de colisão (collision shapes) incluem meshes côncavos e convexos, além de todas as formas primitivas básicas;
- Corpos flexíveis incluem tecidos, cordas e objetos deformáveis;
- Plugins para Maya e Houdini, e integrado em Cinema 4D e Blender;
- Importação/exportação de conteúdo segundo a especificação de física COLLADA;
- Otimizações opcionais para Playstation 3 Cell SPU, CUDA e OpenCL;
- Versão C# que roda em XNA em Windows e Xbox 360;
- Versão multi-threaded disponível para público multi-core.

Entre os projetos que utilizam a Bullet, estão alguns famosos, como Grand Theft Auto 4 e 2012.

Entre os pontos positivos da Bullet, podemos citar:

- Suporta física de corpos flexíveis, e algumas funcionalidades mais avançadas como tecidos;
- Suporta formas complexas como meshes;
- Software livre;
- Muito leve, com pouco custo computacional. O tamanho dos arquivos das lib's somados não chega a 3 MB.

Entre os pontos negativos da Bullet, podemos citar:

- Ainda possui poucos usuários.
- Poucos tutoriais e explicações. Somado à pequena quantidade de usuários, os tutoriais existentes são praticamente apenas os da própria Bullet, os quais se baseiam mais nos demos que em explicações de fato. Se você planeja usar apenas funcionalidades básicas da biblioteca, pode se basear nos demos, mas se quer utilizar a biblioteca para algo mais complexo e específico, não tem por onde se orientar;
- Documentação fraca. Os atributos e métodos das classes não possuem explicações, tendo o próprio usuário que descobrir para que servem quando não são explicados nos demos ou nos escassos tutoriais;

# Capítulo 3

# Implementação

A implementação deste projeto não fez uso de nenhuma engine gráfica, utilizando diretamente a API OpenGL. Além disso, inicialmente, tentou-se a não utilização de engines físicas, mas devido aos resultados pouco satisfatórios, optou-se pelo uso da Bullet Physics.

O projeto pode ser dividido em 3 módulos: interface, gráficos 3D e física.

- A interface do jogo consiste nos menus e janelas, feitos através da API wxWidgets;

Essa API foi escolhida devido à grande variedade de opções que fornece e ao fato de ser de livre uso e multi-plataforma, possibilitando futuras versões para outros sistemas operacionais, se necessário, além de fornecer um ambiente de integração com a OpenGL

- O desenho dos gráficos 3D do jogo é feito com o auxílio da API OpenGL;

Essa API foi escolhida em detrimento da DirectX por ser multi-plataforma, de modo que o aplicativo não fique limitado a Windows, podendo ser produzido para outros sistemas operacionais, se necessário.

- A parte de física aplicada ao jogo é feita pela engine de física Bullet Physics, fornecendo os dados referentes a movimentos e colisões.

A Bullet foi escolhida devido à sua simplicidade e leveza. Como o projeto faz uso apenas de funcionalidades básicas de física (movimento e detecção de colisão de corpos rígidos), essa biblioteca já atende às necessidades. Esta engine foi a primeira testada e, como os resultados foram satisfatórios, não houve a necessidade de testar outra engine de física.

Tendo isso em mente, os diagramas de componentes e de classes do projeto são mostrados a seguir.

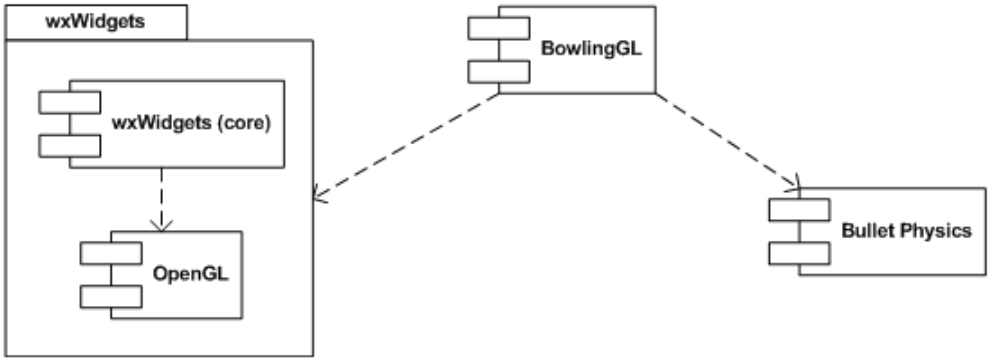


**Figura 3.1 - Diagrama de Componentes**

No diagrama de componentes acima são vistos o aplicativo e suas dependências, sendo elas a wxWidgets, a Bullet Physics e a OpenGL. A OpenGL, porém, tem suas funcionalidades encapsuladas numa dll da wxWidgets para serem usadas dentro da classe wxGLCanvas. Mais especificamente no caso deste projeto, elas serão usadas dentro da wxGLWindow, classe que herda da wxGLCanvas.

Um ponto a ser mencionado é que, devido ao pequeno tamanho de seus arquivos (2,11 MB no total), a Bullet Physics foi ligada de forma estática ao aplicativo, sendo as funcionalidades inseridas diretamente no arquivo do aplicativo, enquanto a wxWidgets foi ligada de forma dinâmica, tendo suas funcionalidades inseridas em arquivos externos, as dll's, com o aplicativo se ligando a elas quando necessário. A vantagem de ligação estática é facilitar a instalação e uso do programa, porém quando há muitas dll's envolvidas, como é o caso da wxWidgets, é recomendada a ligação dinâmica, para que o tamanho do aplicativo não fique muito grande.

E a seguir é mostrado o diagrama de classes do projeto:

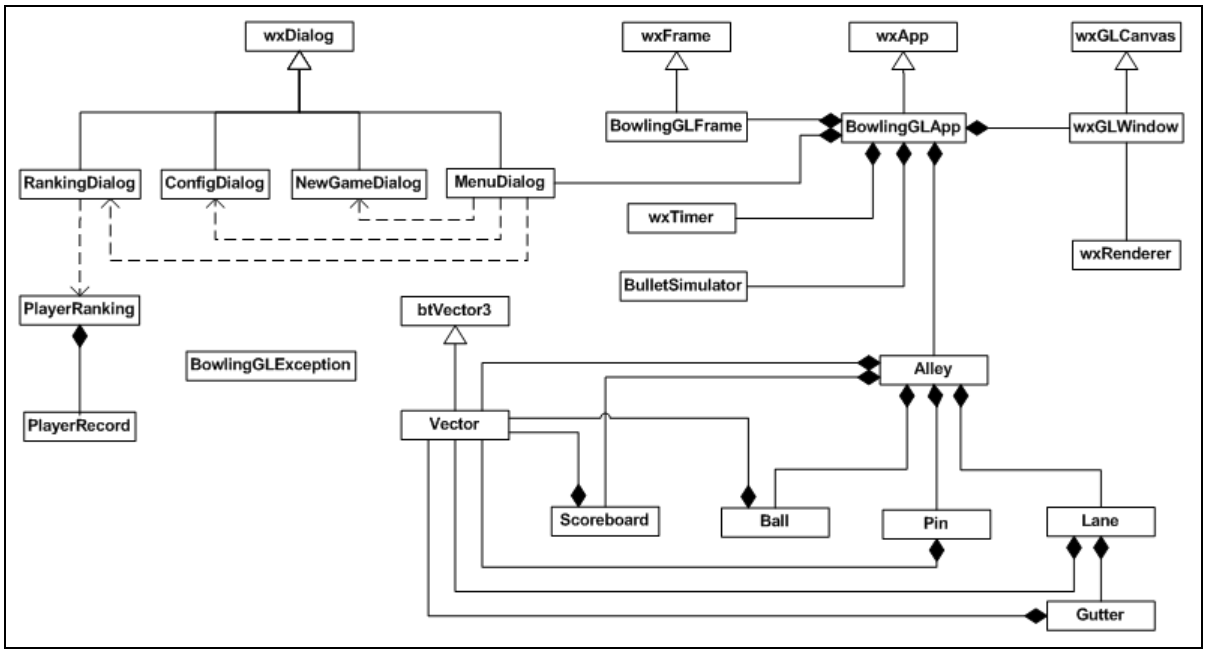


**Figura 3.2 - Diagrama de Classes**

No diagrama acima podem ser vistas as classes:

- BowlingGLApp - representa o aplicativo do jogo, contendo informações e referências para todas as outras classes, direta ou indiretamente;
- BowlingGLFrame - representa a janela principal do jogo;
- wxGLWindow - representa a parte da janela onde são desenhados os gráficos 3D. Esta classe é auxiliada pela wxRenderer, que cuida da renderização em si na wxGLWindow;
- Classes Dialog - são as subjanelas usadas para diversos fins na interface com o usuário. São eles: início de um novo jogo (NewGameDialog), configurações (ConfigDialog), ranking (RankingDialog) e menu principal (MenuDialog);
- BulletSimulator - classe responsável pelas simulações de física do jogo;
- PlayerRanking - ranking composto de vários PlayerRecord, que representam o placar de um jogador;
- Alley, Scoreboard, Ball, Lane, Pin e Gutter - classes que representam os objetos da galeria de boliche, com seus atributos e métodos;

- wxTimer - classe da wxWidgets usada para o loop do jogo;
- Vector - representa um vetor de 3 dimensões da Bullet Physics com a adição de algumas funcionalidades customizadas para o jogo;
- BowlingGLException - usada para tratamento de qualquer exceção durante o jogo.

Nos tópicos a seguir cada módulo do projeto será explicado em mais detalhes, mas primeiro será explicada a integração dos módulos, para dar uma idéia melhor do funcionamento do projeto como um todo.

## 3.1 – Integração dos Módulos

A integração dos módulos é feita por objetos das classes wxTimer, wxGLWindow (classe que herda de wxGLCanvas), wxRenderer e BulletSimulator, cada um pertencente a um dos módulos do projeto, explicados posteriormente em sua respectiva seção.

O objeto wxTimer é responsável pelo loop eterno do jogo. Ele conta continuamente um tempo determinado na sua inicialização e a cada trigger, chama o método Draw() do objeto wxGLWindow. Dentro desse método, chama-se o método Render() da classe wxRenderer. Neste, o método StepSimulation() do objeto BulletSimulator é chamado e faz as simulações físicas do jogo. Após isso, o método DrawScene() do Alley chama os métodos de desenho de cada um dos objetos do cenário. Esses objetos pegam as informações da simulação e as usam para se desenhar com a OpenGL. As informações pegas são vetores de posição e de rotação. Assim, esses valores são armazenados nos objetos para serem usados nos seus desenhos.

O modelo a seguir ilustra esse processo, mostrando as etapas do loop eterno do jogo:

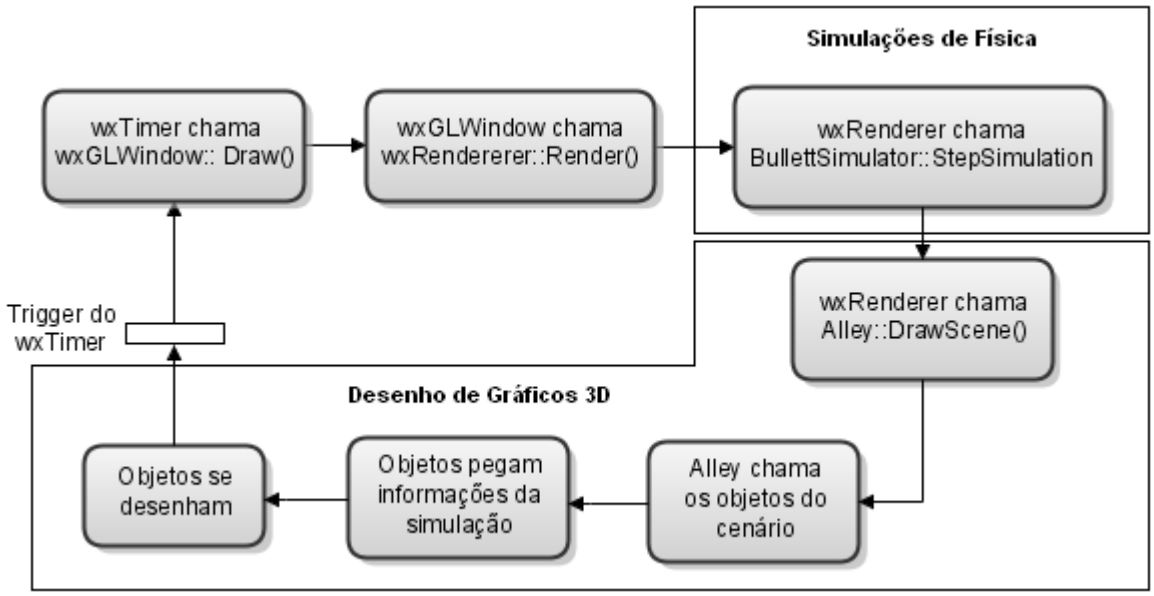


**Figura 3.3 - Modelo da Integração entre os Módulos**

## 3.2 – Interface

A interface do programa pode ser dividida em duas partes: a parte de menus e janelas e a parte dos controles do jogo. Ambas são implementadas através de funcionalidades fornecidas pela wxWidgets.

### 3.2.1 – Menus e Janelas

Esta parte diz respeito à escolha de opções dentro do jogo, realizada pelo usuário. Essas opções incluem jogo para um jogador, para dois jogadores, vista dos rankings dos placares e preferências de jogo.

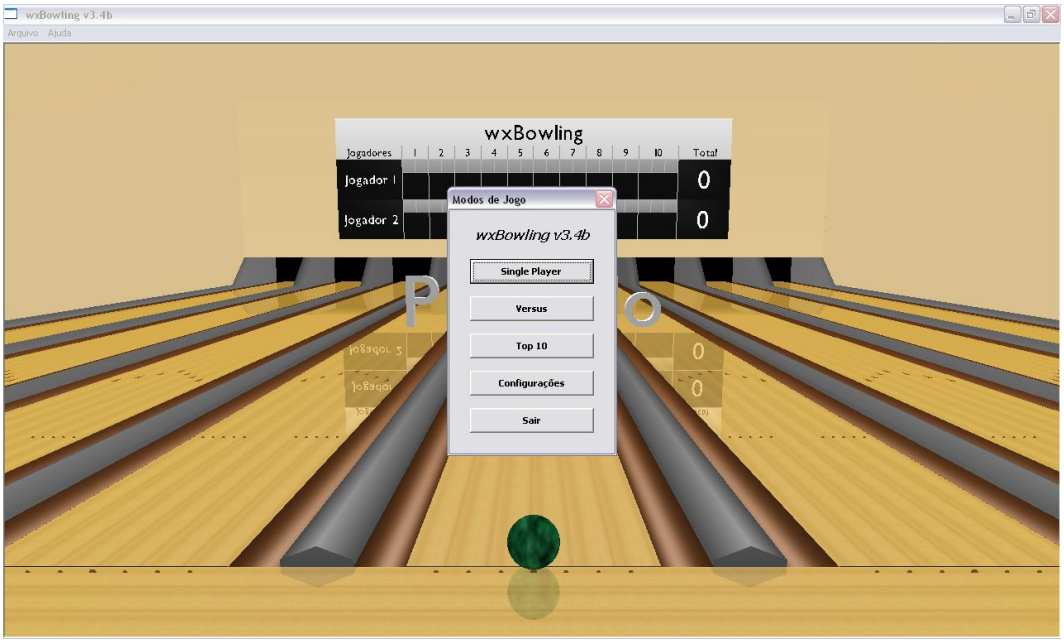


**Figura 3.4 - Menu Principal**

As opções do menu são as seguintes:

- Single Player

Jogo para um jogador. Ao escolher iniciar um jogo, o usuário digita seu nome, para ser usado no ranking.

- Versus

Jogo para dois jogadores, um contra o outro. Ao iniciar, cada jogador digita seu nome para o possível uso no ranking. Ambos podem entrar no ranking, não apenas o vencedor.

- Top 10

Ranking com os dez melhores placares de cada nível de dificuldade (fácil, normal e difícil). A vista dos rankings é dividida de acordo com o nível e eles podem ser apagados se o usuário assim desejar. Esses rankings são salvos nos arquivos binários ranking0, ranking1 e ranking2, um para cada nível.

- Configurações

São as preferências do jogo. Estas mostram algumas opções de customização: dificuldade e idioma. A dificuldade do jogo possui três níveis (fácil, normal e difícil), enquanto os idiomas disponíveis são português e inglês. Essas configurações são salvas num arquivo config.ini para persistirem após o fechamento do aplicativo. Assim, o idioma e a dificuldade escolhidos continuarão os mesmos quando o jogo for executado de novo.

Há ainda os menus da janela (Arquivo e Ajuda), que inclui opções para ir para o menu principal,  sair do jogo, consultar a ajuda, que explica os controles, e o about do jogo.

### 3.2.2 – Controles do Jogo

Após o início do jogo, o usuário pode utilizar o teclado ou o mouse para jogar. A equivalência entre os dois é simples: o botão esquerdo do mouse corresponde à tecla enter e o botão direito à barra de espaço.

Esses controles são explicados na opção Ajuda→Ajuda, mostrada a seguir:

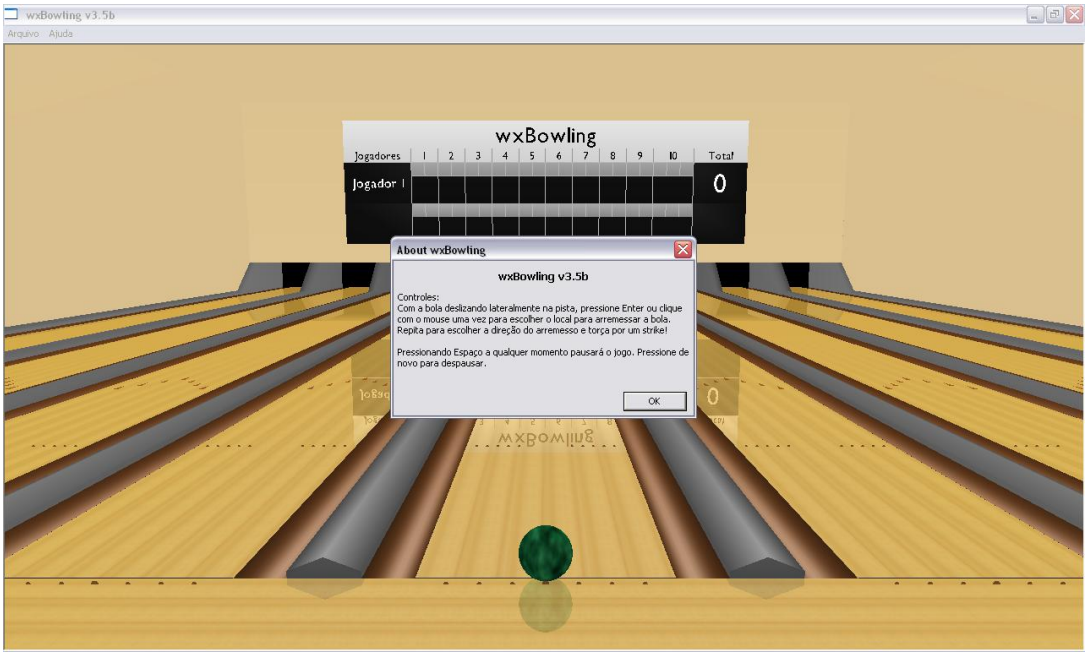


**Figura 3.5 - Janela de Ajuda do Jogo**

Deste modo, a dinâmica do jogo pode ser explicada pelo modelo a seguir:

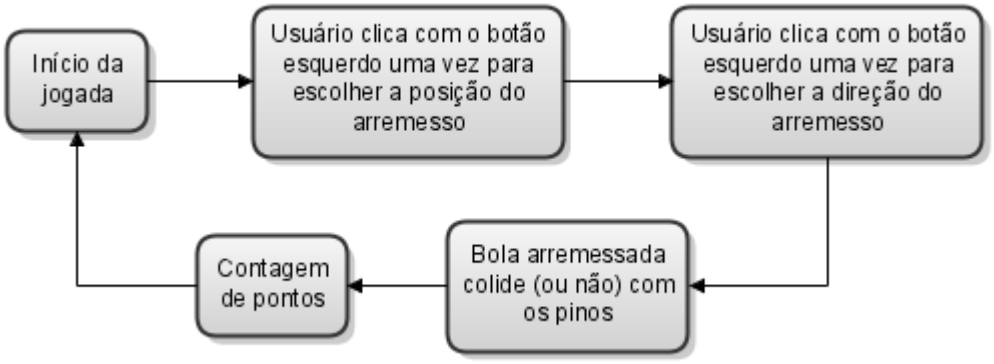


**Figura 3.6 - Modelo da Dinâmica do Jogo**

O usuário pode a qualquer momento pausar o jogo através do botão direito do mouse ou da barra de espaço.

Todos os controles são implementados através de classes da wxWidgets, sendo utilizados dentro do jogo para repassar as informações do usuário para as classes encarregadas da dinâmica do jogo (renderização de gráficos, lógica do jogo, etc).

## 3.3 – Gráficos 3D

Os gráficos 3D do jogo são todos desenhados utilizando a biblioteca OpenGL.


**Figura 3.7 - Gráficos 3D do Jogo**

As paredes, placar, divisórias e linhas das pistas são desenhadas normalmente, enquanto o piso e bola utilizam texturas fornecidas por arquivos png para um maior realismo. Já os pinos são desenhados através de um conjunto de cilindros, cones e uma meia esfera no topo.

As letras e números desenhados utilizam funções dos arquivos text3d.h e text3d.cpp [4], Essas funções são carregam de um arquivo (charset) modelos 3D e os desenham na tela.

Todos esses desenhos são feitos dentro das classes wxGLWindow e wxRenderer. Para desenhar os gráficos com essas classes, o seguinte procedimento foi usado:

- Criar uma classe derivada da wxGLCanvas, a wxGLWindow

Essa classe permite o desenho de gráficos 3D dentro da janela do programa e escuta os eventos vindos do usuário (mouse e teclado).

- Criar a classe wxRenderer

Essa classe é uma classe Singleton, ou seja, possui apenas uma instância. Ela é responsável pela renderização em si, sendo usada dentro da wxGLWindow.

- Implementar métodos para tratar os eventos de teclado e mouse na wxGLWindow

Esses eventos são responsáveis pela interação do usuário com o jogo. Para cada evento, é implementado um método para lidar com ele.

Assim, para um evento de teclado, se verifica qual a tecla pressionada e se realiza a ação correspondente. Para o mouse, os métodos para botão esquerdo e direito são separados, cada um implementando a sua funcionalidade.

- Delegar o desenho dos gráficos à classe wxRenderer

A cada loop de execução do jogo, o método Draw() da wxGLWindow é chamado. Esta classe delega essa função à wxRenderer, chamando o método Render(), passando a si mesma como argumento. Assim temos a chamada:

```
wxRenderer::GetInstance().Render(this);
```

- Testar inicialização da wxGLWindow

A cada chamada de Render(), a wxRenderer testa se a wxGLWindow passada como argumento foi inicializada. Se não foi, o método Init() é chamado, o qual seta alguns parâmetros da OpenGL como perspectiva, luz e normalização.

- Avançar simulação de física e desenhar objetos

Após o teste da inicialização, se o jogo não estiver pausado, a simulação de física é avançada em um passo. Então, os métodos de desenho dos objetos do cenário são chamados. Esses desenhos utilizam as funções da OpenGL, mas não são imediatamente mostrados na tela. Para diminuir o gasto computacional do desenho dos gráficos e para o usuário ver apenas o resultado final, eles são postos num buffer, o qual será desenhado na tela todo de uma vez.

- Desenhar resultados na wxGLWindow

Os desenhos feitos pelos objetos no item anterior são mostrados na tela com o método SwapBuffers() da wxGLWindow, herdado da wxGLCanvas().

Como o projeto foi feito utilizando Orientação a Objetos, cada objeto (galeria de boliche, pista, bola, pino, canaleta) é implementado em uma classe separada, possuindo o seu método de renderização. Desse modo, os objetos de pista, bola, pinos, etc pertencem à galeria, esta apenas chamando os métodos de cada objeto na hora devida.

Essa hora é definida por um objeto da classe wxTimer, que realiza um loop eterno que só é parado quando o usuário decide sair do jogo. Neste loop, é ordenado à wxGLWindow que desenhe os gráficos a cada intervalo de tempo definido pela macro REFRESH_RATE na inicialização do objeto wxTimer.

A macro REFRESH_RATE possui o valor de 20 ms na máquina de testes. Medindo o frame rate [5], encontrou-se que este varia entre 26 e 34 fps, tendo um valor médio de 30 fps. Os valores mais baixos foram registrados onde as cenas renderizadas eram mais estáticas como, por exemplo, quando o jogo está pausado. Já os maiores valores foram registrados nas cenas mais dinâmicas, mais especificamente, quando a câmera acompanha a bola enquanto esta percorre a pista.

O funcionamento deste módulo pode ser ilustrado pelo modelo a seguir:

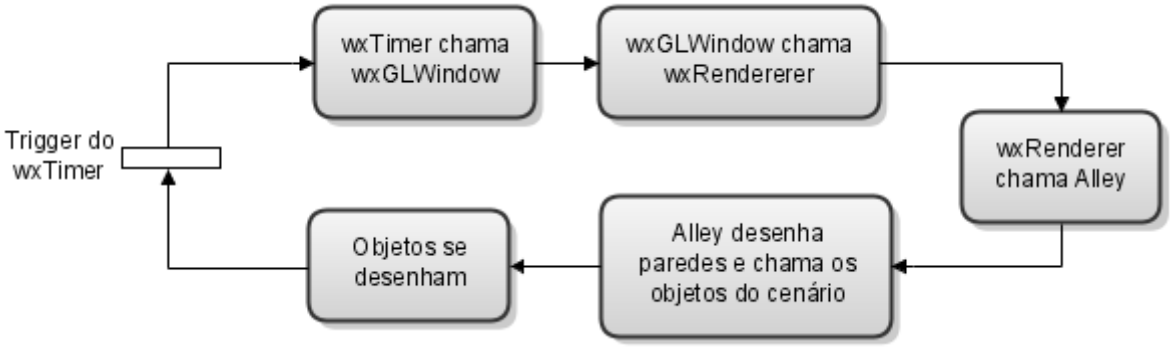


**Figura 3.8 - Modelo da Renderização dos Gráficos 3D**

## 3.4 – Física

Este módulo é responsável pelas simulações de física do jogo, realizadas pela engine de física, Bullet Physics. Estas simulações fornecem os dados das posições e das rotações destes objetos, isto é, bola e pinos.

Esses dados são utilizados para os desenhos dos objetos com OpenGL. Mais especificamente no caso deste jogo, a física presente envolve o movimento e as colisões entre pinos, bola e pista.

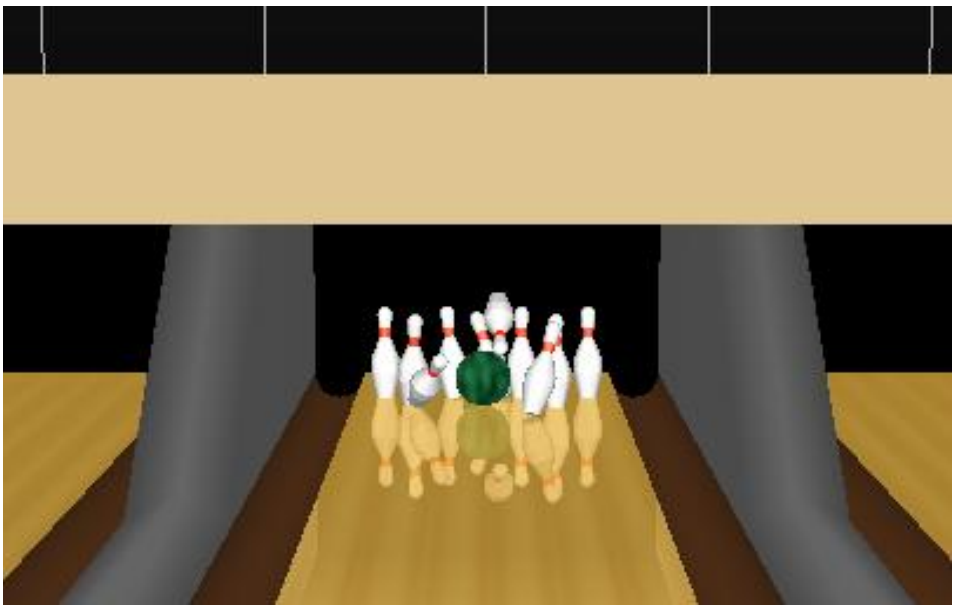

**Figura 3.9 - Colisão da Bola com os Pinos**

O funcionamento deste módulo é ilustrado pelo modelo a seguir:

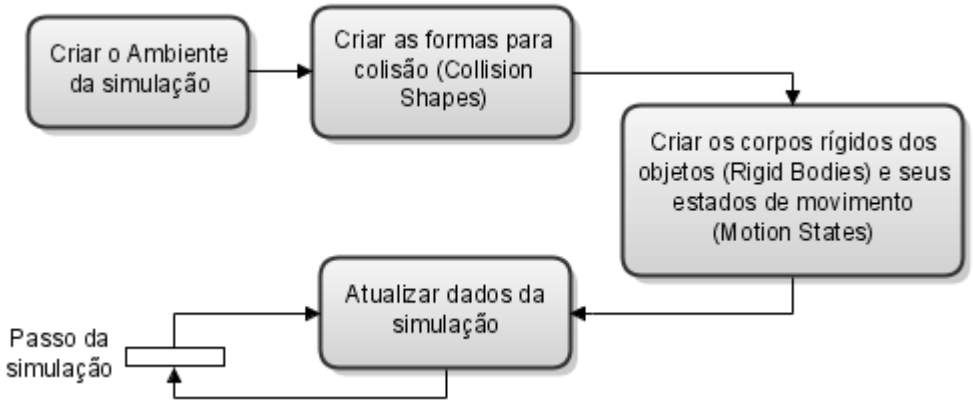


**Figura 3.10 - Modelo da Simulação de Física do Jogo**

Explicando um pouco mais cada etapa:

- Criar Ambiente da Simulação

Nesta etapa, são criados alguns objetos necessários à simulação como os responsáveis pelo tratamento das colisões e o próprio mundo da simulação, isto é, a forma e o tamanho do ambiente a ser considerado na simulação. Aqui também é definido o vetor gravidade do ambiente.

Isto é feito usando alguma das implementações da classe btDynamicsWorld. No caso, foi usada a btDiscreteDynamicsWorld. Esta classe é a responsável pelo gerenciamento de todos os objetos de física do ambiente e pelo seu update a cada frame.

- Criar as formas para colisão (Collision Shapes)

Collision Shapes são responsáveis pelas colisões. Elas não representam os objetos em si, são apenas formas usadas para o tratamento de colisões, tendo que ser atribuídas a um corpo rígido para poderem ser usadas. Porém, uma única Collision Shape pode ser usada para vários objetos diferentes, contanto que eles tenham a mesma forma. Esta abordagem poupa memória  no processamento das colisões.

Desse modo, no jogo são criadas formas para o chão da pista, as canaletas, as paredes, a bola e os pinos usando implementações da classe btCollisionShape. Contudo, para diminuir os custos da simulação, só foram criadas as formas relacionadas à pista

central, baseando essa decisão no fato de a bola e os pinos estarem sempre dentro desse espaço, seja na pista ou nas suas canaletas.

Assim, nesta etapa, são criadas as Collision Shapes para cada tipo de objeto no jogo: uma para a bola, uma para os pinos e outras para o chão e paredes da pista.

- Para a bola de boliche, escolheu-se a forma de uma esfera, representada pela classe btSphereShape;
- Para o chão e as paredes da pista central, escolheram-se paralelepípedos ou caixas, representados pela classe btBoxShape;
- Para os pinos, escolheram-se cilindros, representados pela classe btCylinderShape. Estes foram escolhidos por serem a forma básica mais próxima de um pino de boliche, não sendo justificável uma forma mais complexa por esta já apresentar excelentes resultados;
- Para a canaleta, não se escolheu nenhuma forma pela ausência de forma para um meio cilindro, forma característica de uma canaleta. A solução encontrada foi criar um "corredor" no lugar da canaleta. Com isso, se usa um paralelepípedo (btBoxShape) apenas para o chão da canaleta, e suas paredes laterais são definidas pelas formas do chão e da parede da pista central, deixando um espaço entre elas do diâmetro da bola. Dessa forma, já que a bola não pode se movimentar para os lados na canaleta, ela "escoa" pelo corredor como se fosse a forma de uma canaleta de verdade. O resultado final é mais do que satisfatório, não podendo ser notada qualquer anormalidade pelo usuário.

- Criar os corpos rígidos dos objetos (Rigid Bodies) e seus estados de movimento (Motion States)

Rigid Bodies, implementados pela classe btRigidBody, representam os objetos em si, cada um correspondendo a um objeto do cenário. Com isso, é preciso fornecer dados para a massa do objeto e propriedades do material, como atrito.

Além disso, um Rigid Body possui associada uma Collision Shape, de modo a tratar as colisões do objeto, e um Motion State, implementado pela classe btDefaultMotionState e responsável pela movimentação do objeto. Estes Motion States cuidam da sincronização do

objeto com a transformada do mundo da simulação, fornecendo os dados de posição e rotação usados para a renderização dos objetos.

Nesta etapa da simulação, é criado um Rigid Body para cada objeto do cenário, havendo distinção entre objetos de formas iguais, diferentemente do que ocorre com as Collision Shapes. Como cada objeto terá uma posição e uma rotação diferente dos outros, é necessário um Rigid Body (com um Motion State associado) para cada um.

- Iniciar e atualizar periodicamente dados da simulação

Por último, é definido o tempo do passo da simulação e esta é iniciada. A cada passo, as colisões e movimentos dos objetos são processados e suas posições são atualizadas. Isso é feito através do algoritmo a seguir:

```
void BulletSimulator::StepSimulation() {
m_dynamicsWorld->stepSimulation(5*REFRESH_RATE/1000.0f, 3*REFRESH_RATE/50.0f + 2);
Alley *alley = wxGetApp().m_curAlley;
alley->GetPlayerBall().SetPosition(GetBallPosition());
// Atualizar posições dos pinos
for (int k=0; k<10 ; k++) {
        alley->GetPin(k).SetPosition(GetPinPosition(k));
        alley->GetPin(k).SetRotationVector(GetPinRotationAxis(k));
        alley->GetPin(k).SetRotationAngle(GetPinRotationAngle(k));
...
```

O método stepSimulation acima é usado para dar o passo da simulação e é definido deste modo:

```
stepSimulation(
   btScalar timeStep,
   int maxSubSteps=1,
   btScalar fixedTimeStep=btScalar(1.)/btScalar(60.));
```

O parâmetro timeStep acima é definido em segundos e, como a macro REFRESH_RATE é definida em milissegundos, ela é dividida por 1000 para convertê-la para segundos. Os valores usados são empíricos e definidos com auxílio de sugestões da documentação da biblioteca.

Porém, a biblioteca apresentou um pequeno problema durante o seu uso. Os pinos, ao serem derrubados, de repente começavam a deslizar pela pista com um movimento acelerado, como se estivessem em uma rampa. Mas, ao contrário do que ocorre numa rampa, a aceleração ocorria para qualquer direção e sentido indiscriminadamente.

Após algumas tentativas em vão de consertar o problema, a solução encontrada foi um teste manual: se o pino for derrubado e não tiver saído da pista, a sua velocidade angular é diminuída manualmente, evitando o problema.

O código abaixo foi usado para a solução do problema. Ele é posicionado logo após o trecho de código mostrado acima.

```
…
if ((GetPinPosition(k).y() <= PIN_RADIUS)&&(fabs(GetPinPosition(k).x()) < LANE_WIDTH/2.0))
        //Manually stop pins from rolling indefinitely
        m_pinRigidBody[k]->setAngularVelocity(m_pinRigidBody[k]->getAngularVelocity()/2);
…
```

No código, a coordenada y do pino de número k (k é o contador do laço de repetição, variando de 0 a 9, um para cada pino) é testada. Se ela atingir o valor PIN_RADIUS, significa que o pino está deitado, ou seja, ele foi derrubado. Se, além disso, a posição x do pino for menor que LANE_WIDTH/2.0, isto é, se ele ainda estiver na pista e não na canaleta, a velocidade angular do pino é dividida por 2 a cada passo, até que passe de um limiar (definido pela biblioteca de física) e pare. Com isso, o problema foi corrigido.

# Capítulo 4

# Análise de Resultados

A jogabilidade do jogo é bastante razoável. Ela é simples e de fácil entendimento, o que ajuda no resultado final do jogo ao proporcionar diversão imediata, sem a necessidade de um grande aprendizado. Os menus são fáceis e elucidativos, não provocando dúvidas nos usuários, e os controles são simples, podendo ser aprendidos rapidamente. Doze usuários comuns testaram o aplicativo e nenhum usuário reclamou quanto aos menus e aos controles nem teve dificuldades com os mesmos. Porém, cinco dos doze (41,7%) reclamaram da ausência de controle de força ou efeito no arremesso. A implementação destes pode ser adicionada futuramente para aumentar a diversão do jogo.

A resposta dos controles do jogo é imediata, sem qualquer atraso. O tratamento de eventos (teclado e mouse) pela wxWidgets não apresentou nenhum problema e responde rapidamente durante a execução do jogo. O que se notou, porém, é que, no computador de testes [6], com outros aplicativos rodando e ocupando mais de cerca 90% da sua capacidade de processamento, há um pequeno atraso na renderização dos gráficos 3D, baixando o frame rate do jogo em 3 fps [5], variando de 23 a 31 fps . No entanto, até certo ponto, isso é esperado de um aplicativo que utilize gráficos 3D.

Em relação à renderização, o que se notou é que, ao retirar as texturas do jogo, a velocidade fica ligeiramente maior (1~2 fps [5]). Isso evidencia o custo computacional do uso de texturas. Porém, esse custo é aceitável em prol do realismo e da beleza do jogo.

O realismo do jogo, aliás, é bastante satisfatório, fato este devido na sua maior parte ao uso da Bullet Physics. A bola se desloca exatamente na direção escolhida pelo usuário, sempre caindo na canaleta quando sai da pista e se deslocando nela até o final. A bola também sempre gira na direção correta. As colisões da bola com os pinos e dos pinos entre si também é muito realista, não podendo ser notada qualquer anormalidade nesses processos. O processamento das colisões é rápido, não havendo qualquer atraso durante o mesmo. O único defeito notado foi o mencionado na seção Física do capítulo Implementação, o qual já foi consertado dentro do jogo.

# Capítulo 5

# Conclusão

## 5.1 – Avaliação

Ao final do projeto, o resultado obtido foi bastante satisfatório. Os objetivos enunciados na proposta do projeto foram atingidos:

- Elaborar um jogo 3D de boliche que, logicamente, respeite as regras do esporte e utilize uma física compatível para o movimento dos objetos e colisões

Objetivo atingido. O jogo apresenta gráficos 3D bonitos, respeita rigorosamente as regras do esporte e apresenta uma física totalmente compatível. No movimento dos objetos e nas colisões não foi notado nenhum defeito além do mencionado na seção Física do capítulo Implementação. Porém, este problema foi solucionado e tudo ocorre como esperado.

- Possibilitar o funcionamento do jogo em janelas GUI do Windows, adicionando menus e opções

Objetivo também atingido. O jogo funciona em Windows e possui menus e opções extras como rankings e níveis de dificuldade.

No aspecto da diversão, ponto chave de qualquer jogo, é possível afirmar que o jogo atinge as expectativas, proporcionando diversão para seus usuários. Os menus são simples e elucidativos, colaborando para a diversão do jogo.

## 5.2 – Adições Futuras

Para possíveis adições futuras para o jogo, podemos sugerir:

- Implementar força no arremesso

Necessário para aumentar o realismo do jogo. Com a adição da Bullet Physics, não deve dar muito trabalho, necessitando apenas de um método para pegar o input do usuário.

- Implementar efeito no arremesso

Situação semelhante ao item anterior. Também é necessário para aumentar o realismo do jogo. Porém, deve dar um pouco mais de trabalho para a definição do quanto a bola deve ser rotacionada após ser arremessada.

- Adicionar uma IA ao jogo, possibilitando jogos contra o computador

Esta adição, apesar de trabalhosa, aumentará muito o fator diversão do jogo ao fornecer ao usuário mais desafios ao jogar contra o computador.

- Adicionar áudio ao jogo

Esta adição também aumentará o fator diversão do jogo ao aumentar o realismo e a imersão do usuário no jogo. E quanto maior a imersão, maior a diversão do usuário.

- Implementar um frame rate ajustável para cada computador com suas especificações

No projeto atual, o frame rate foi ajustado manualmente para atingir um resultado satisfatório. Apesar de em todas as máquinas em que o jogo foi testado, o resultado ter sido satisfatório, num computador com um desempenho da placa de vídeo bem diferente (superior ou inferior) dos testados, pode haver diferenças na velocidade do jogo.

- Otimizar o desempenho

Este projeto não foi desenvolvido com grandes preocupações quanto ao desempenho. Apesar de o resultado ser bastante satisfatório e de os computadores de hoje disporem de muitos recursos como memória e capacidade de processamento, em computadores com menos recursos e rodando muitos aplicativos ao mesmo tempo, o jogo pode ficar lento.

Mesmo esse cenário não sendo o indicado ao se executar jogos 3D, alguma otimização no funcionamento do projeto pode ser útil.

- Portar o jogo para outras plataformas

Este projeto foi desenvolvido só para Windows, mas com alguns ajustes é perfeitamente capaz de rodar em sistemas como Mac e Linux devido à compatibilidade de seus componentes com outras plataformas.

Também seria interessante portar o jogo para plataformas exclusivas de videogames, como o PSP, por exemplo, além de plataformas de celular, como o iPhone. Porém, elas exigiriam um maior esforço, por ser necessário uma reformulação de pontos do projeto devido à incompatibilidade de seus componentes, além de, em alguns casos, como o das plataformas de celular, ser necessária também uma otimização do desempenho do jogo por causa das limitações de hardware existentes nos dispositivos móveis.

# Referências Bibliográficas


[1] Segundo texto de Roberto Tadeu Fauri *Técnicas empregadas no desenvolvimento de jogo*, disponível no site http://www.dinx.com.br/2009/08/tecnicas-empregadas-no-desenvolvimento-de-jogos-parte-1

[2] Segundo dados presentes na documentação da biblioteca, disponível no endereço http://www.continuousphysics.com/Bullet/BulletFull/index.html.

[3] Texto disponível em http://www2.joinville.udesc.br/~larva/santiago/Trabalhos/rejane.pdf

[4] Funções desenvolvidas por Bill Jacobs com auxílio do programa de modelagem 3D open source Blender (http://www.blender.org/) e disponibilizadas para livre uso em http://www.videotutorialsrock.com.

[5] Valores de frame rate medidos com o aplicativo D3DGear (http://www.d3dgear.com/)

[6] Macbook, processador Core 2 Duo, 3MB de Cache, 2 GB de memória RAM e placa de vídeo Intel Graphics Media Accelerator X3100 com memória de vídeo compartilhada (até 144 MB).

# Bibliografia


[1] *OpenGL, The Industry's Foundation for High Performance Graphics.* http://www.opengl.org/ (Primeiro acesso em Março de 2008).

[2] JACOBS, Bill, *OpenGL Tutorial.* http://www.videotutorialsrock.com/ (Primeiro acesso em Março de 2008).

[3] *OpenGL Software Development Kit 2.1 Reference Pages.* http://www.opengl.org/sdk/docs/man/ (Primeiro acesso em Abril de 2008).

[4] *wxWidgets, Cross-Platform GUI Library. http://www.wxwidgets.org/* (Primeiro acesso em Setembro de 2007).

[5] *wxWidgets 2.8 Reference Pages. http://docs.wxwidgets.org/stable/* (Primeiro acesso em Abril de 2008).

[6] Fischer, Thomas, *howto wxWidgets, glCanvas and openGL. http://thomasfischer.biz/?p=113* (Primeiro acesso em Janeiro de 2009).

[7] *Bullet Physics - Game Physics Simulation*. http://www.bulletphysics.com (Primeiro acesso em Dezembro de 2009)

[8] *Bullet Physics Documentation.* http://www.continuousphysics.com/mediawiki-1.5.8/index.php?title=Documentation (Primeiro acesso em Outubro de 2009).

[9] Pestana, Luis, *Introdução ao Desenvolvimento de Jogos.* http://www.triares.com.br/blog/?cat=8 (Primeiro acesso em Dezembro de 2009)

[10] *Wikipedia – Programação de Jogos Eletrônicos*. http://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_de_jogos_eletr%C3%B4nicos (Primeiro acesso em Dezembro de 2009)

[11] *Wikipedia – Motor de Jogo*. http://pt.wikipedia.org/wiki/Game_engine (Primeiro acesso em Dezembro de 2009)

[12] Fauri, Roberto Tadeu, *Técnicas empregadas no desenvolvimento de jogos*. http://www.dinx.com.br/2009/08/tecnicas-empregadas-no-desenvolvimento-de-jogos-parte-1 (Primeiro acesso em Dezembro de 2009)

[13] *Wikipedia – OpenGL*. http://en.wikipedia.org/wiki/OpenGL (Primeiro acesso em Dezembro de 2009)

[14] wxWidgets Development Team, *wxWidgets Overview*. http://www.wxwidgets.org/about/datasheets/wxWidgetsOverview.pdf (Primeiro acesso em Dezembro de 2009)

[15] Lira dos Santos, Artur, *CoReactive: Um Sistema de Colaboração para Ambientes Virtuais Distribuídos*. http://www2.fc.unesp.br/wrva/artigos/50462.pdf (Primeiro acesso em Dezembro de 2009)

[16] *Wikipedia – Bullet (software)*. http://en.wikipedia.org/wiki/Bullet_(software) (Primeiro acesso em Dezembro de 2009)

[17] *Irrlicht Engine*. http://irrlicht.sourceforge.net (Primeiro acesso em Dezembro de 2009)

[18] *OGRE – Open Source 3D Graphics Engine*. http://www.ogre3d.org (Primeiro acesso em Dezembro de 2009)

[19] *Newton Game Dynamics*. http://www.newtondynamics.com (Primeiro acesso em Dezembro de 2009)

[20] *Open Dynamics Engine*. http://www.ode.org (Primeiro acesso em Dezembro de 2009)

[21] *SGI*. http://www.sgi.com/ (Primeiro acesso em Dezembro de 2009)