RENATO DE SOUZA GOMES

PESQUISA DE BIBLIOTECAS MULTIPLATAFORMA PARA PROGRAMAS MULTIMÍDIA

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Bruno de Oliveira Schneider

LAVRAS
MINAS GERAIS – BRASIL
2000

RENATO DE SOUZA GOMES

PESQUISA DE BIBLIOTECAS MULTIPLATAFORMA PARA PROGRAMAS MULTIMÍDIA

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 03 de julho de 2001.

_____

Prof. Joaquim Quinteiro Uchôa

_____

Prof. José Monserrat Neto

_____

Prof. Bruno de Oliveira Schneider
UFLA
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

RESUMO

Devido à quase total substituição das interfaces em linhas de comando pelas interfaces gráficas e, também, pela crescente aceitação de sistemas operacionais alternativos ao Windows (mesmo em computadores de uso pessoal) há uma maior necessidade de esforço por parte dos programadores para fazer programas competitivos no mercado. O presente trabalho destina-se a ser uma referência a programadores (principalmente programadores de jogos) que queiram desenvolver aplicativos com interface gráfica e recursos multimídia e que sejam multiplataforma (pelo menos Linux e Windows). Estudou-se três bibliotecas de programação para tal fim, as quais estão disponíveis na Internet, são gratuitas e com código fonte aberto. Das três estudadas, uma não apresentou o rendimento esperado, enquanto que as outras duas se mostram em um estágio de desenvolvimento bastante satisfatório e dão suporte a várias plataformas.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1 - Introdução

1.1 Interfaces Gráficas

A forma que os programas de computador são apresentados ao usuário tem sofrido diversas modificações desde os primórdios da computação. Sendo uma das mais recentes a substituição quase que por completa das interfaces em linha de comando pelas interfaces gráficas, podendo ser observada principalmente a partir do início dos anos noventa. Essa transformação acarretou em uma grande aceitabilidade dos computadores por parte da população, uma vez que esta nova forma de serem apresentados torna seu uso mais intuitivo e fácil.

Como conseqüência dessa mudança, tem-se uma necessidade de maior esforço por parte dos programadores. Pois, além de terem que se preocupar com o algoritmo do programa, também têm que se preocupar cada vez mais com a interface que ele será apresentado.

Outro fator que também tem influenciado os programas é o uso cada vez maior de recursos multimídia, tais como objetos tridimensionais, sons e figuras. Tal uso é vindo, principalmente, do avanço dos dispositivos de hardware como, por exemplo, processadores mais velozes, placas de vídeos com aceleradores gráficos, placas de som com alta qualidade.

Isso acarreta em outro esforço por parte dos programadores que, no intuito de tornar seu programa competitivo, têm que se preocupar com os recursos que vão utilizar e como vão utilizá-los.

1.2 Independência de Plataforma

Atualmente pode-se observar no mercado de sistemas operacionais que o padrão de mercado é o Windows. Porém, a aceitação de outras plataformas tem crescido bastante, até mesmo para uso pessoal. Isso faz com que não se possa prever como estará o mercado futuro. Não é possível nem mesmo falar se alguma das plataformas atualmente utilizadas ainda existirá daqui a algum tempo.

Então, garantir que o programa assuma a característica de ser multiplataforma faz com que ele não somente tenha um mercado mais amplo mas, também, tenha uma 'vida' maior.

Isso traz um novo desafio ao programador, que é o de fazer seu programa de modo que este possa ser compilado nas mais diversas plataformas. Este não seria um grande desafio se o programador não tivesse que se preocupar com a interface gráfica. As interfaces gráficas e os outros recursos multimídia são o grande problema de se fazer um programa multiplataforma.

Cada sistema operacional oferece formas bem distintas de se usar os recursos multimídia do computador, tornando dispendiosa e difícil a tarefa de portar um programa escrito em uma plataforma para outra qualquer.

1.3 Possíveis Soluções

1.3.1 *Interface Development Environment* (IDE)

Uma possível solução para o desenvolvimento de interfaces gráficas seria o uso de IDEs. Porém, apesar de serem de qualidade e eficiência reconhecidas, são geralmente de custo elevado e, normalmente, destinam-se a uma única plataforma.

1.3.2 *Applications Programming Interface* (API)

Uma outra solução seria a partição do programa em elementos gráficos e não-gráficos. Para o desenvolvimento dos elementos gráficos, o programador usaria a API nativa do sistema operacional. Essa é uma boa solução, uma vez que elimina *overheads* (elementos incluídos inutilmente em um programa, uma vez que não são utilizados), freqüentemente associados ao uso de bibliotecas e também dá ao programa uma melhor performance em cada plataforma que aquele irá ser executado [3].

Essa solução, apesar de ser boa, exige muito mais dos programadores, tanto no desenvolvimento quanto nas futuras manutenções. Eles devem também aprender como escrever código para todas as plataformas escolhidas, o que não é uma tarefa trivial.

1.3.3 *Plataform-Independent Graphical User Interface* (PIGUI)

Um *kit* PIGUI é uma biblioteca de software que um programador usa para produzir códigos de interfaces gráficas para diferentes sistemas computacionais. Ele apresenta funções e/ou objetos que são independentes de qual sistema operacional o programador tem como objetivo. O *kit* não necessariamente fornece quaisquer características adicionais de portabilidade [3].

Há uma grande quantidade de *kits* PIGUI disponíveis na Internet, dos quais uma grande parte pode ser encontrada no endereço indicado em [8].

Dentre essas bibliotecas já se havia pesquisado algumas em estudos anteriores com relação à manipulação de janelas, menus, botões, etc. Tal estudo, feito pelo então Grupo de Desenvolvimento de Jogos da UFLA, serviu como base para a escolha das bibliotecas a serem pesquisadas. No total foram três as bibliotecas escolhidas, uma vez que é bastante complexo aprender a escrever código para cada uma, principalmente por não haver padronizações entre as bibliotecas.

Mais adiante será descrita cada uma delas e o porque de sua inclusão no estudo.

Capítulo 2 - Jogos por Computador

2.1 Definição de um Jogo por Computador

Um jogo por computador pode ser definido como um sistema composto de três partes básicas: **enredo, motor e interface interativa**. O sucesso de um jogo está associado à combinação perfeita destes componentes [2].

O enredo define o tema, a trama, o objetivo do jogo, o qual através de uma série de passos o usuário deve se esforçar para atingir. A definição da trama não envolve só criatividade e pesquisa sobre o assunto a ser enfocado pelo jogo, mas também a interação com pedagogos, psicólogos e especialistas em tal assunto.

A interface interativa controla a comunicação entre o motor e o usuário, reportando graficamente um novo estado do jogo. O desenvolvimento da interface envolve aspectos artísticos, cognitivos e técnicos. O valor artístico está na capacidade que ela tem de valorizar a apresentação do jogo, atraindo usuários e aumentando a sua satisfação ao jogar. O aspecto cognitivo está relacionado à correta interpretação gráfica pelo usuário. Note-se que em termos de jogos educacionais a interface deverá obedecer a critérios pedagógicos. O aspecto técnico envolve performance, portabilidade e a complexidade dos elementos gráficos.

O motor do jogo é o seu sistema de controle, o mecanismo que controla a reação do jogo em função de uma ação do usuário. A implementação do motor envolve diversos aspectos computacionais, tais como, a escolha apropriada da linguagem de programação em função de sua facilidade de uso e portabilidade, o desenvolvimento de algoritmos específicos, o tipo de interface com o usuário, etc.

2.2 Importância dos Jogos por Computador

2.2.1 Mercadológica

Atualmente, jogos por computador têm sido largamente explorados em termos comerciais. Dados da *Interactive Digital Software Association* (IDSA), a associação que organiza a *Electronic Entertainment Expo* (E3) demonstram que em 1999 foram comercializados em torno de US$ 6,1 bilhões em software de entretenimento só nos Estados Unidos. Em 1996, este volume era de US$ 3,7 bilhões. No Brasil, jogos disponíveis em CD têm uma venda estimada em duzentas mil unidades por ano, o que corresponde a 6% do mercado total de venda de CDs. Isto sem considerar a pirataria [2].

2.2.2 Educacional

Detalhes comerciais à parte, um ponto importante a ser considerado nesta área é a possibilidade de se combinar entretenimento com educação. Infelizmente, os jogos de maior sucesso comercial atualmente são os que melhor combinam violência com efeitos visuais sofisticados. Estima-se que menos de 20% dos jogos disponíveis têm algum tipo de enfoque educacional, sendo que, em geral, os jogos mais utilizados só servem para desenvolver rapidez de raciocínio e reflexo. Logo, há um grande campo para pesquisa nessa área e, considerando a possibilidade de se disponibilizar jogos educacionais via Internet, pode-se pensar em integrar este tipo de software a programas educacionais [2].

2.2.3 Capacitação Técnica

A capacitação técnica advém do fato de que os conceitos envolvidos na implementação de um jogo são multidisciplinares, abrangendo múltiplas subáreas computacionais, tais como, linguagens, sistemas operacionais, computação gráfica, inteligência artificial, etc., bem como as áreas de psicologia e pedagogia. Ressalte-se que jogos por computador estão caminhando para se tornar filmes

interativos para múltiplos usuários com acesso via Internet, assim, a tecnologia empregada no desenvolvimento de um jogo de ação também pode ser empregada na implementação de qualquer software de interação sofisticada na Internet, em especial, aqueles que envolvam educação [2].

2.2.4 Nicho de Mercado

Outro aspecto importante observado sobre jogos por computador é que eles, ao contrário de pacotes tradicionais, como, por exemplo, formatadores de texto, compiladores, sistemas gráficos, etc., são potencialmente mais competitivos em termos de mercado. Este fato se justifica porque cada jogo é um produto em particular e o seu sucesso não está totalmente relacionado à sua sofisticação computacional, mas sim aos atrativos lúdicos que ele fornece aos usuários. Neste sentido, a tão propalada "imaginação do brasileiro" pode ser um fator bastante positivo no desenvolvimento deste tipo de produto [2].

Capítulo 3 - Objetivos

Com base nas necessidades dos programadores de jogos, o objetivo deste trabalho é proporcionar a eles uma comparação entre algumas das bibliotecas multiplataforma para desenvolvimento de interfaces gráficas e manipulação de dispositivos multimídia (*kits* PIGUI). Tais bibliotecas, além de serem multiplataforma, também são gratuitas e com código fonte aberto.

A comparação é baseada em testes entre as bibliotecas selecionadas, para verificar a capacidade de cada uma em manipulação de imagens, som e outros quesitos passíveis de serem utilizados em jogos.

A partir dos testes realizados, gerar um *ranking* entre as bibliotecas. Neste *ranking* estarão presentes a posição de cada uma das bibliotecas e o porque desta posição.

A importância deste ranking é o fato de que um programador que o consulte terá como referência qual das bibliotecas selecionadas é mais adequada para sua aplicação ou, até mesmo, se nenhuma delas é aconselhável.

Capítulo 4 - Bibliotecas Selecionadas

Existem várias bibliotecas do tipo PIGUI disponíveis na Internet. Os critérios para seleção foram que a biblioteca fosse multiplataforma (pelo menos Linux e Windows), gratuita e tivesse código fonte aberto.

4.1 *Simple Direct-Media Layer* (SDL)

A biblioteca SDL é desenvolvida para facilitar produção de jogos que executam em Linux, *free*BSD, MacOS, Win32 e BeOS usando vários meios de interface nativos de alta-performance (para vídeo, áudio, etc.) e apresentando um único código fonte para sua aplicação. SDL é quase uma API de nível baixo, mas usando-a, aplicações completamente portáveis podem ser escritas com uma grande flexibilidade [4].

A SDL é composta de oito subsistemas: Áudio, CD-ROM, Manipulação de Eventos, Entrada e Saída em Arquivos, Manipulação de *Joystick*, *Threading*, Temporizadores e Vídeo.

Sua inclusão no estudo se deve ao fato de que ela é usada em várias versões para Linux de jogos de grande sucesso (*Civilization: Call to Power*, *Sim City 3000*, *Heretic II*, etc.).

4.2 wxWindows

A wxWindows é uma ferramenta de trabalho que fornece *Graphical User Interface* (GUI) e outras facilidades em mais de uma plataforma. A versão 2 atualmente suporta MS Windows (16-bit, Windows 9X e Windows NT), Unix com GTK+, Unix com Motif e MacOS. O suporte a OS/2 está em progresso [5].

A wxWindows foi desenvolvida para providenciar um modo barato e flexível de maximizar o investimento em desenvolvimento de aplicações GUI. Enquanto algumas classes de bibliotecas comerciais já existem para desenvolvimento multiplataforma, nenhuma segue os seguintes critérios:

- Gratuidade;
- código fonte disponível;
- simplicidade de programação;
- suporte a uma grande quantidade de compiladores.

Sua inclusão se deve ao fato de que ela é atualmente utilizada na UFLA em disciplinas do curso de Ciência da Computação.

4.3 *Fast Light Toolkit* (FLTK)

A FLTK é um *kit* de ferramentas C++ de desenvolvimento de interface gráfica para X (Unix), OpenGL e Microsoft Windows NT4.0, 95 ou 98. Ela foi inicialmente desenvolvida por Bill Spitzak e é atualmente reparada por um pequeno grupo de desenvolvedores através do mundo com matriz nos Estados Unidos [6].

Foi desenvolvida para ser ligada estaticamente. Isto foi feito dividindo-a em muitos pequenos objetos e desenvolvendo isso de maneira tal que as funções que não são usadas não têm apontadores a elas nas partes que são usadas, então não são ligadas. Isso permite a você fazer um programa fácil de instalar, ou modificar a FLTK para os requerimentos exatos de sua aplicação, sem se preocupar com emendas. A FLTK trabalha bem como uma biblioteca compartilhada e começou a ser incluída nas distribuições do Linux (por isso sua inclusão nos testes).

Capítulo 5 - Testes Realizados

Foram realizados quatro testes com cada uma das bibliotecas. São eles: mapeamento de bitmaps, animação de *sprites*, som e manipulação de objetos tridimensionais utilizando a OpenGL.

Em todos os testes foram verificadas facilidades de implementação, manipulação de eventos, orientação a objetos, velocidade dos aplicativos, linhas de código e tamanhos dos executáveis gerados.

5.1 Mapeamento de Bitmaps

A partir de um mapa, composto de vários bitmaps e que não cabe por completo na tela, mostra-se um pedaço dele e pode deslizar sobre ele sem que seja notado o redesenhamento da tela.

O objetivo deste teste é verificar a capacidade de cada uma em manipular grandes quantidades de pixels na tela da forma mais rápida o possível, uma vez que tal recurso é muito utilizado em alguns tipos de jogos. Outras características que também foram observadas com este teste foram a capacidade da biblioteca em utilizar o modo *fullscreen* e se ela era capaz de utilizar a resolução desejada pelo programador.

5.2 Animação de *Sprites*

Um *sprite* é uma imagem (ou conjunto de imagens) que é (são) mostrada(s) na tela sem deixar marcas por onde passa. A animação de um *sprite* é o ato de mostrar essa(s) figura(s) na tela numa seqüência de modo que pareça com um desenho animado.

O objetivo deste teste é verificar a capacidade de cada biblioteca em lidar com transparência ou máscaras aplicadas a figuras. Outro objetivo é o de implementar essa técnica, a qual é muito usada em jogos por computador.

5.3 Som

O objetivo deste teste é o de verificar a capacidade e a facilidade de cada biblioteca em manipular arquivos de som. Os principais tipos visados neste teste foram os do tipo WAVE, MIDI e MP3, por serem os mais utilizados tanto em jogos como em aplicativos de música.

Uma breve descrição sobre os tipos será dada a seguir:

5.3.1 Wave

Popularizou-se em função do grande número de plataformas com o Windows e seus aplicativos. O Wave é atualmente a base do áudio digital, sendo largamente utilizado em efeitos sonoros. Existem vários tipos de ferramentas que manipulam este formato, permitindo a geração dos mais variados tipos de efeitos. Dependendo da qualidade utilizada para um arquivo Wave, ele pode exigir uma taxa de transmissão de 40 MB por minuto, o que o torna muito pesado para operar em computadores de pequeno porte [2].

5.3.2 *Musical Instrument Digital Interface* (MIDI)

Define uma linguagem de transmissão de dados digitais entre sistemas computacionais, sintetizadores e instrumentos musicais. A comunicação pode ser feita com base no protocolo ou em arquivos MIDI. Por exemplo, quando uma nota é tocada em um teclado, a porta de comunicação MIDI envia para um outro instrumento ou computador conectado a ela, todas as informações pertinentes, tais como a nota, a sua velocidade, tipo, etc. O MIDI não é áudio digital, portanto não há possibilidade de se gravar vozes ou efeitos sonoros, ele somente mantém a seqüência de notas tocadas e as armazena com um tipo de som pré-definido, sendo eles variados, como guitarra, teclado, gaita, etc. Sua característica mais interessante é o pouco espaço em disco que ele ocupa, por exemplo, 1

minuto de som com só uma trilha e sem nenhum evento a ele ligado requer 3KB, o que comparado com o formato Wave e outros é muito pequeno [2].

5.3.3 *Moving Pictures Experts Group Áudio Layer 3* (MP3)

Tem como objetivo armazenar som compactado. Sua definição é baseada em um modelo psico-acústico, ou seja, o ouvido humano não é capaz de ouvir todas as freqüências, há um limite entre 20Hz e 20KHz e ele é mais sensitivo entre 2KHz e 4KHz. Assim, um algoritmo elimina grande parte das freqüências que um ouvido humano possa não escutar e ainda algumas que possam ser retiradas sem que haja perda na qualidade sonora. Ressalte-se que esta compactação é destrutiva, a compactação do MP3 elimina informações que nunca mais poderão ser recuperadas. Ele pode ter qualidade igual à do CD e tamanho até 12 vezes menor que um arquivo igual no formato Wave [2].

5.4 Manipulação de Objetos Tridimensionais Utilizando a OpenGL

A *OpenGL Utility Toolkit* (GLUT) é um grande exemplo de como software pode realmente servir as pessoas e se tornar um padrão somente por ser genial. A ferramenta de Mark Kilgard tem ganhado uma grande popularidade. Devido a ela ser completamente com código fonte aberto, o software tem sido sintonizado e arrumado a ponto de que você seria bobo em não usá-lo se você está fazendo uma aplicação multiplataforma [1].

O objetivo deste teste é verificar a capacidade de cada uma das bibliotecas em interagir com a biblioteca OpenGL, uma vez que ela é uma das melhores (se não for a melhor) para manipulação de objetos tridimensionais e também por ser multiplataforma, gratuita e com código fonte aberto.

Capítulo 6 - Resultados

Com os quatro testes já realizados em cada uma das bibliotecas e com os desempenhos apresentados por cada uma, foi possível verificar resultados bastante interessantes.

Os resultados relativos a cada teste serão expostos a seguir em forma de tabelas. Mas algumas observações gerais podem ser feitas a cada uma das bibliotecas:

SDL – foi de grande destaque na medida em que foi capaz de utilizar recursos como *fullscreen*, dar a possibilidade do usuário escolher entre utilizar memória de vídeo ou do sistema, utilizar telas com a resolução desejada (desde que possível). Porém ela não é orientada a objetos e possui poucas funções prontas para serem utilizadas, ou seja, ela proporciona a capacidade de, por exemplo, utilizar a placa de som, mas a maioria do código deve ser escrita pelo programador. Gera executáveis pequenos e bastante velozes. Outro problema é que os programas gerados com ela necessitam de uma de um arquivo chamado SDL.dll para poderem ser executados em Windows. Mas esse arquivo já vem em sua distribuição. Foi a de mais fácil instalação em Windows.

wxWindows – totalmente orientada a objetos e com uma grande quantidade de recursos, não só para multimídia mas também para rede e outros, já prontos. Porém gera executáveis bem maiores e mais lentos que os da SDL. Seus recursos para desenhar são bons, mas um pouco complicados de serem entendidos e implementados. De todas é a que tem a melhor documentação.

FLTK – gera executáveis pequenos e velozes. Porém, seus recursos para desenho são muito fracos e ela não tem funções prontas para utilizar tipos de imagens, tais como BMP, JPG e GIF. Também não é capaz de manipular sons.

6.1 Mapeamento de Bits

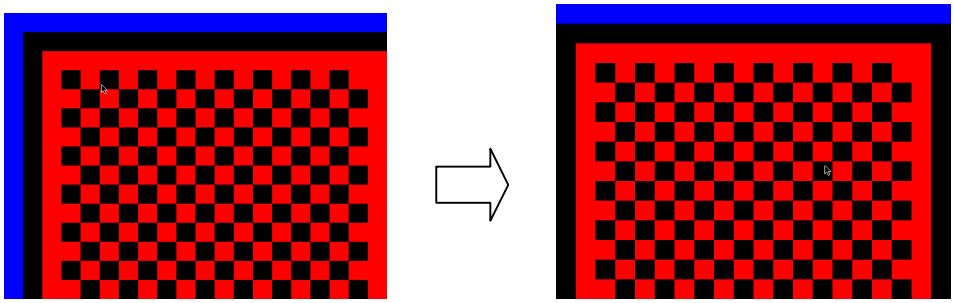

Figura 1 – Resultado obtido no teste Mapeamento de Bits com a SDL.

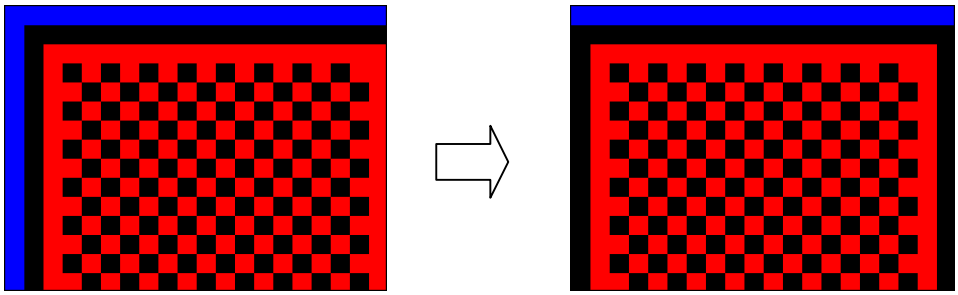

Figura 2 – Resultado obtido no teste Mapeamento de Bits com a wxWindows.

Tabela 1 – *Ranking* baseado no desempenho de cada uma das bibliotecas no teste de Mapeamento de Bits. O tamanho dos executáveis, como indicado na tabela, foi medido no sistema operacional Windows.

Posição	Nome da Biblioteca	Nro. de linhas de código	Tamanho do executável	Tamanho do executável com *strip*
1º	SDL	170	442.374 B (432 KB)	100.352 B (98 KB)
2º	WxWindows	158	1.929.854 B (1,83 MB)	1.018.368 B (994 KB)
-	FLTK	-	-	-

As figuras 1 e 2 mostram o mapa como ele é apresentado na tela assim que o aplicativo é posto para executar e a segunda tela que é apresentada assim que o cursor do mouse chega ao lado direito da tela. Apesar delas serem iguais,

elas não mostram o fato de que a Figura 1 foi captada em tela cheia e a Figura 2 não.

Pode-se notar, com base na tabela 1, que não há um *ranking* para a biblioteca FLTK. Isso se deve ao fato de que ela não fornece suporte aos principais arquivos de imagem e, mesmo tentando implementar com um tipo que ela suporta, não foi possível desenvolver um aplicativo à altura dos outros.

Quanto ao fato da SDL vir em primeiro lugar é conseqüência das facilidades que ela proporciona na manipulação de imagens. Apesar de não fornecer suporte a alguns tipos de imagens, há pacotes (também gratuitos, multiplataforma e com código fonte aberto) que a auxiliam. Ela também fornece uma função de *blit* (cópia de uma parte de memória para outra que está sendo mostrada) bem simples de ser entendida e usada.

A wxWindows veio em segundo pelo fato de não poder modificar a resolução do vídeo e também porque sua função de *blit* não é tão boa quanto a da SDL.

O número de linhas de código foi observado nos códigos fonte sem comentários e linhas em branco.

Strip é um programa para retirar os pontos de *debug* de um outro programa, ele vem junto com a distribuição do gcc (compilador c/c++ utilizado). Com isso reduz-se o tamanho do programa.

6.2 Animação de *Sprites*

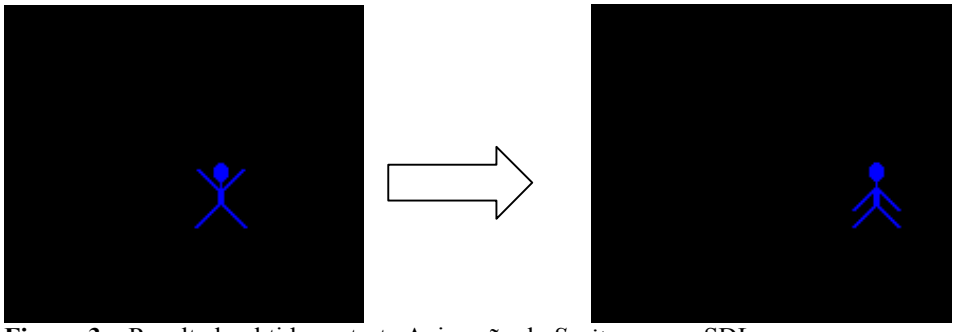

Figura 3 – Resultado obtido no teste Animação de *Sprites* com a SDL.

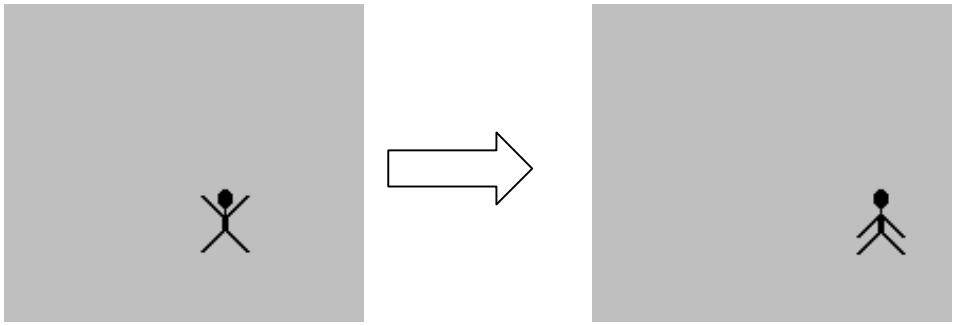

Figura 4 – Resultado obtido no teste Animação de *Sprites* com a wxWindows.

Tabela 2 – *Ranking* baseado no desempenho de cada uma das bibliotecas no teste de Animação de *Sprites*. O tamanho dos executáveis, como indicado na tabela, foi medido no sistema operacional Windows.

Posição	Nome da Biblioteca	Nro. de linhas de código	Tamanho do executável	Tamanho do executável com *strip*
1º	SDL	299	410.341 B (400 KB)	84.480 B (82,5 KB)
2º	WxWindows	124	1.925.575 B (1,83 MB)	1.015.296 B (991 KB)
-	FLTK	-	-	-

Existem várias técnicas para se implementar *sprites*. A escolhida foi a de seguir os passos abaixo:

- Num vetor (ou uma lista) guarda-se as imagens que vão compor a animação;
- Recupera-se o fundo do local onde o *sprite* está sendo mostrado
- Salva-se o fundo do local onde a imagem vai ser desenhada para que este possa ser recuperado;
- Desenha-se a imagem na posição desejada (local que foi salvo anteriormente);
- Alterna para a próxima imagem a ser mostrada pelo sprite.

Neste teste foram usadas duas imagens para compor a animação. O meio de se saber qual delas desenhar foi ter uma variável (do tipo inteiro) que se estivesse em 1 mostraria a primeira imagem, caso fosse –1 mostraria a segunda. Para alterna-la bastava multiplica-la por –1 (figuras 3 e 4).

Observando a tabela 2, pode-se verificar que novamente a FLTK não foi classificada devido à sua debilidade em manipular imagens.

Apesar da grande quantidade de linhas de código (muitas das quais eram somente para assegurar a boa funcionalidade do aplicativo e que poderiam ser excluídas) a SDL se destacou pela facilidade e precisão que ela proporciona ao programador. Uma pequena observação cabe aqui com relação à adição de transparência à imagem utilizada no *sprite*, a qual tem que ser feita pelo programador. Porém essa tarefa é contornada com uma função de quatro linhas de código.

Dois motivos levaram a wxWindows a ficar em segundo novamente. Primeiro porque mesmo atualizando apenas o retângulo necessário para desenhar o *sprite* foi possível, algumas vezes, observar o redesenhamento daquele retângulo. Segundo porque em alguns pontos o *sprite* deixava umas pequenas

marcas de sua cor. Um fator positivo dela foi a existência de uma função (wxIcon) que já atribui transparência ao bitmap utilizado.

6.3 Som

Tabela 3 - *Ranking* baseado no desempenho de cada uma das bibliotecas no teste de Som. O tamanho dos executáveis, como indicado na tabela, foi medido no sistema operacional Windows.

Posição	Nome da Biblioteca	Nro. de linhas de código	Tamanho do executável	Tamanho do executável com *strip*
1º	SDL	78	395.393 B (386 KB)	76.800 B (75 KB)
2º	WxWindows	60	1.923.566 B (1,83 MB)	1.015.808 B (991 KB)
-	FLTK	-	-	-

Novamente a biblioteca SDL veio em primeiro. Isso se deve ao fato de que, apesar de ser um pouco complicado seu entendimento (agravado pela falta de informações mais específicas em seu manual de referência), ela pelo menos fornecia a capacidade de se pausar a reprodução. A wxWindows, mesmo oferecendo uma função pronta para utilizar arquivos Wave, essa função permite apenas iniciar e parar a reprodução e também não está muito bem documentada.

Nenhuma delas foi capaz de utilizar os outros tipos de arquivos (MIDI, e MP3). Isso pode ser frustrante, uma vez que, para muitos jogos, utilizar-se desses arquivos é mais vantajoso que a utilização dos arquivos Wave.

6.4 Manipulação de Objetos Tridimensionais Utilizando a OpenGL

Neste teste não foi gerada uma tabela comparativa, pois aqui se pretende somente verificar a capacidade de cada biblioteca em utilizar a OpenGL para

desenhar objetos tridimensionais. Não foi implementado nenhum teste para verificar a facilidade em integrar as bibliotecas com a OpenGL.

Para ilustrar cada biblioteca utilizando a OpenGL, mostra-se a seguir exemplos disponibilizados por cada uma (Figuras 5, 6 e 7).

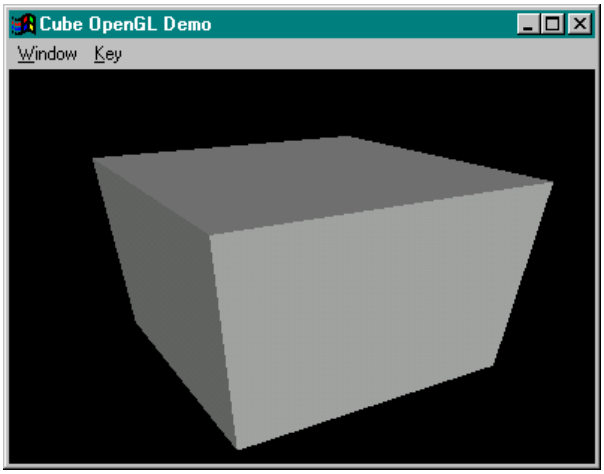

Figura 5 – Exemplo da biblioteca wxWindows utilizando a OpenGL para fazer um desenho tridimensional.

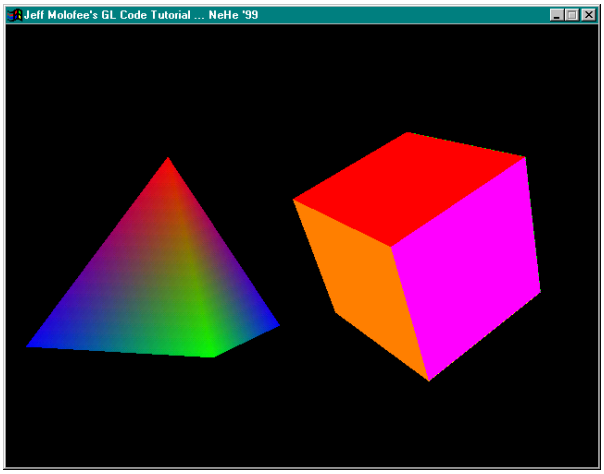

Figura 6 – Exemplo da biblioteca SDL utilizando a OpenGL para fazer um desenho tridimensional.

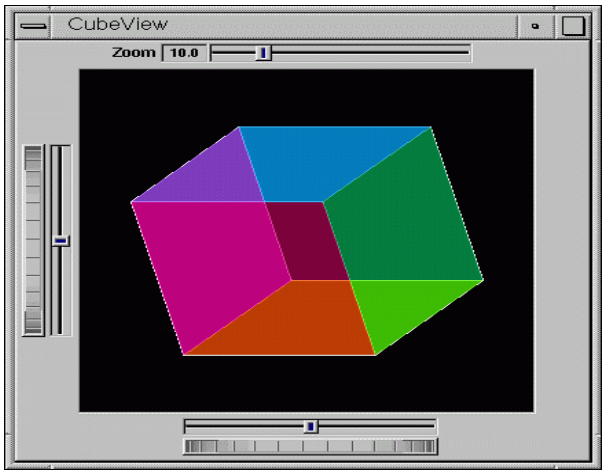

Figura 7 – Exemplo da biblioteca FLTK utilizando a OpenGL para fazer um desenho tridimensional.

Capítulo 7 - Conclusão

Com exceção da FLTK, com a qual não foi possível realizar os três primeiros testes, as demais bibliotecas se apresentaram aptas a desenvolver programas com grande qualidade.

Muito interessante também foi o fato de não ser necessário modificar uma linha de código sequer para que o programa pudesse ser compilado tanto em Linux como em Windows. Mas, mesmo assim, alguns recursos que estão disponíveis em uma plataforma não estão em outras. Isto pode ser contornado com diretivas de pré-compilação, as quais vão garantir que um recurso disponível, por exemplo, somente para Windows não seja usado em outra plataforma e dê resultados não desejados pelo programador.

Quanto ao uso das bibliotecas para desenvolver a interface gráfica e outros recursos multimídia de um jogo recomenda-se o uso da SDL. Pois, além de ter sobressaído nos testes em relação às outras, há também vários pacotes, que a utilizam como base e que podem ser utilizados em conjunto com ela, que fornecem ao usuário funções já prontas para manipular diferentes tipos de imagem, som e outros recursos que em sua distribuição original não são fornecidos.

A biblioteca SDL se saiu muito bem pela facilidade de se utilizar recursos que muitas vezes são desejados em jogos. Exemplos disso são o modo em tela cheia (*fullscreen*) e resolução gráfica configuráveis pelo programa. Seu estilo de utilizar superfícies como janelas (ou simplesmente como um espaço na memória, no qual se pode desenhar) é muito simples e fácil de ser utilizado.

A biblioteca wxWindows é uma boa opção para um programador que queira desenvolver um aplicativo que requeira botões, menus, caixas de texto, etc. Pois ela já trás funções que criam automaticamente tais objetos (e os criam de acordo com o padrão da plataforma em que o programa foi utilizado). A SDL em sua distribuição original não fornece tais facilidades.

Apesar das debilidades da FLTK, ela também pode ser uma boa opção quando se tratar de aplicativos mais simples, que não requeiram uso de imagens ou de som. Os objetos gerados com suas funções têm boa qualidade visual (de acordo com a plataforma em que o programa é compilado).

No geral pode-se dizer que investir em tal pesquisa pode ser de grande proveito para universidades, pois podem ser utilizadas pelos acadêmicos para desenvolverem seus programas com uma qualidade melhor, sem se prender a esta ou aquela plataforma e sem ter que utilizar recursos financeiros da entidade em que estudam para adquirir ferramentas disponíveis no comércio. Atente-se ao fato de que os programas gerados com tais bibliotecas não são precisamente de caráter gratuito ou de qualidade pior com relação aos desenvolvidos com uma ferramenta paga. Eles podem ter qualidade muito boa se desenvolvidos com a biblioteca certa. E ainda podem ter um bom valor no mercado se desenvolvidos corretamente. Vale ainda ressaltar que o seu mercado será ainda maior, dependendo de quantas plataformas a biblioteca suporta.

REFERÊNCIAS

[1] DELOURA, M. *Game Programming Gems*. Charles River Media, 2000.

[2] BATTAIOLA, A.L. Jogos por Computador. Em Anais da XIX Jornada de Atualização em Informática, Curitiba, 2000.

[3] McKAY, R. *Plataform Indepent FAQ*. 1997. url: http://www.zeta.org.au/~rosko/pigui.htm

[4] LANTINGA, S. *Simple Direct-Media Layer*. 2001. url: http://www.libsdl.org

[5] ROEBLING, R. *wxWindows, Cross-Plataform Development for Unix/ Windows/MacOS*. 2001. url: http://www.wxwindows.org

[6] SPITZAK, B.; et al. *The Fast Light Toolkit Home Page*. 2001 http://www.fltk.org

[7] APIKI, S. *Paths to Plataform Independence*. 1994. url: http://www.byte.com/art/9401/sec9/art1.htm

[8] TAI, L.C. *The GUI Toolkit*, *Framework Page*. 2001. url: http://www.geocities.com/SiliconValley/Vista/7184/guitool.html

ANEXOS

Anexo A – Trechos dos Códigos do Teste de *Sprites* com as Bibliotecas SDL e wxWindows.

- SDL:

```
//variáveis globais
SDL_Surface *screen; //janela principal
vector <SDL_Surface*> sprite(2);//sprite com  duas
                                //imagens
SDL_Surface *background;   //background do sprite
SDL_Rect spt_pos;     //posição atual do sprite
int spt_visible;      //1 -> mostra primeiro sprite
                      //-1 -> mostra segundo sprite
enum //constantes
{
      CIMA = 0,
      BAIXO,
      ESQUERDA,
      DIREITA
};
void LoadSprite()
{
      SDL_Surface *temp;
      //Carregar a primeira imagem do sprite
      temp = SDL_LoadBMP("spt1.bmp");
      //Configurar para pixel transparente como o
      //pixel em (0,0)
      if (temp->format->palette)
      {
            SDL_SetColorKey(temp, SDL_SRCCOLORKEY,
            *(Uint8*) temp-> pixels);
      }
      //Converter temp para formato vídeo
      sprite[0] = SDL_DisplayFormat(temp);
      SDL_FreeSurface(temp);

      //Para carregar a segunda imagem do sprite,
      //segue-se os mesmos passos da primeira

      //salvando o background
      temp = SDL_CreateRGBSurface(SDL_SWSURFACE,
      sprite[0]->w, sprite[0]->h, 8, 0, 0, 0, 0);
```

```
	//Converter background para o formato de vídeo
	background = SDL_DisplayFormat(temp);
	SDL_FreeSurface(temp);
	//configurar os flags
	spt_visible = 1;
	spt_pos.x = 0;
	spt_pos.y = 0;
	spt_pos.h = 32;
	spt_pos.w = 32;
	//transferir o retângulo para o background
	SDL_BlitSurface(screen, &spt_pos,background,
	NULL);
	//Mostrar o primeiro sprite em screen
	SDL_BlitSurface(sprite[0], NULL, screen,
	&spt_pos);
	SDL_UpdateRects(screen, 1, &spt_pos);
}
void MoveSprites(int dir)
{
	//transferir o background para screen
	SDL_BlitSurface(background, NULL, screen,
	&spt_pos);
	SDL_UpdateRects(screen, 1, &spt_pos);
	//Mudar para que a próxima figura seja mostrada
	spt_visible *= -1;
	switch(dir)
	{
		case CIMA:
			spt_pos.y -= 32;
			//transferir o novo background para
			//screen
			SDL_BlitSurface(screen, &spt_pos,
			background, NULL);
			// transferir o sprite de acordo com
			//spt_visible
			if(spt_visible > 0)
			{
				SDL_BlitSurface(sprite[0],
				NULL, screen, &spt_pos);
			}
			else
			{
				SDL_BlitSurface(sprite[1], NULL,
```

```
                    screen, &spt_pos);
               }
               break;
          case BAIXO:
          //repete-se os passos anteriores, só muda o
          //retângulo
          case ESQUERDA:
          //repete-se os passos anteriores, só muda o
          //retângulo
          case DIREITA:
          //repete-se os passos anteriores, só muda o
          //retângulo
     }
     //mostrar o sprite na nova posição
     SDL_UpdateRects(screen, 1, &spt_pos);
}
```

- wxWindows:

```
canvas::canvas(wxWindow *parent) : wxPanel(parent, -1,
     wxPoint(0,0),parent->GetSize(),wxSIMPLE_BORDER,
     "canvas")
{
     //iniciando as variáveis e as figuras do sprite
     wxIcon bmpAux;
     bmpAux.LoadFile("spt1.bmp", wxBITMAP_TYPE_ICO);
     m_sprite.push_back(bmpAux);
     bmpAux.LoadFile("spt2.bmp", wxBITMAP_TYPE_ICO);
     m_sprite.push_back(bmpAux);

     m_spt_visible = 1;
     m_destx = 0;
     m_desty = 0;
     //atribuindo o tamanho da janela principal às
     //variáveis telaw e telah
     parent -> GetSize(&m_telaw, &m_telah);
     parent -> Refresh();  //redesenhando a janela
                           //principal
}
void canvas::OnPaint(wxPaintEvent &evento)
{
     wxPaintDC dc(this);
     if (m_spt_visible > 0)
```

```
            dc.DrawIcon(m_sprite[0], m_destx, m_desty);
      else
            dc.DrawIcon(m_sprite[1], m_destx, m_desty);
}
void canvas::teclado(wxKeyEvent &evento)
{
      switch(evento.GetKeyCode())
      {
            case WXK_ESCAPE:
                  GetParent() -> Close(TRUE);
                  break;
            case WXK_UP:
                  if (m_desty > 0)
                  {
                        wxRect rect(m_destx, m_desty-32,
                        32, 64);
                        m_desty -= 32;
                        m_spt_visible *= -1;
                        GetParent() -> Refresh(TRUE,
                        &rect);
                  }
                  break;
            case WXK_DOWN:
            //repete-se os passos anteriores, só muda o
            //retângulo
            case WXK_LEFT:
            //repete-se os passos anteriores, só muda o
            //retângulo
            case WXK_RIGHT:
            //repete-se os passos anteriores, só muda o
            //retângulo
      }
}
```

Anexo B – Trechos dos Códigos do Teste de Mapeamento de Bitmaps com as Bibliotecas SDL e wxWindows.

- SDL:

```
//var. globais
SDL_Surface *tela;    //tela aonde será mostrado o
                      //mapa
vector <vector <char> > matriz; //matriz contendo
                                //informações sobre o
                                //mapa
int x, y;  //posições da matriz do canto sup. esq. que
           //deve aparecer
int w, h;  //tamanhos dos bmp's
void desenhaMapa()
{
    SDL_Surface *imagemBMP, *imagem;
    SDL_Rect destino;
    //primeiro lugar que vai aparecer
    destino.x = 0;
    destino.y = 0;
    destino.h = h;
    destino.w = w;
    //calcular a quantidade de imagens que aparecerão
    int contx, conty;
    contx = (int)(tela->w)/w + x;
    conty = (int)(tela->h)/h + y;
    if(contx > matriz[1].size())
        contx = matriz[1].size();
    if(conty > matriz.size())
        conty = matriz.size();
    // carregando as imagens numa superfície
    for(int i = y; i < conty; i++)
    {
       destino.x = 0;
       for(int j = x; j < contx; j++)
       {
            if (matriz[i][j] == 'w')
                imagemBMP =
                    SDL_LoadBMP("blacksquare.bmp");
            else if (matriz[i][j] == 'b')
```

```
                imagemBMP =
                    SDL_LoadBMP("bluesquare.bmp");
            else
                imagemBMP =
                    SDL_LoadBMP("redsquare.bmp");
            // Converter a imagem para o formato do
            //video (mapas de cores)
            imagem = SDL_DisplayFormat(imagemBMP);
            SDL_FreeSurface(imagemBMP);
            // acrescentar a imagem às outras
            SDL_BlitSurface(imagem, NULL, tela,
                &destino);
            SDL_FreeSurface(imagem);
            // deslocar a posição para a direita
            destino.x += w;
        }
        // deslocar a posição para baixo
        destino.y += h;
    }
    // Atualizar a tela
    SDL_UpdateRect(tela, 0,0,0,0);
}
void comandos()
{
    bool sair = false;     //para detectar fim de
                           //programa
    bool mudou = false;    //para detectar se a tela
                           //será redesenhada
    SDL_Event evento;      //para decifrar eventos
                           //ocorridos
    int i, j;   //para detectar as posições do mouse
    SDL_EnableUNICODE(1);
    while(!sair)
    {
        SDL_GetMouseState(&i, &j);
        if(i == 0)
        {
            if(x > 0)
            {
                mudou = true;
                x -= 1;
            }
        }
```

```
        if(i == ((int)tela->w - 1))
        {
            if(x < (matriz[1].size() - tela->w/w))
            {
                mudou = true;
                x += 1;
            }
        }
        if(j == 0)
        {
            if(y > 0)
            {
                mudou = true;
                y -= 1;
            }
        }
        if(j == ((int)tela->h - 1))
        {
            if(y < (matriz.size() - tela->h/h))
            {
                mudou = true;
                y += 1;
            }
        }
        if(mudou)
        {
            desenhaMapa();
            mudou = false;
        }
    }
}
```

- wxWindows:

```
//variáveis globais
vector <vector <char> > matriz; //matriz
                                //representando o
                                //mapa
int x, y;  //posição x,y da matriz que estão no canto
           //sup. esq. da tela
int w, h;  //largura e altura dos bitmaps
```

```
void canvas::deslizar(wxMouseEvent &evento)
{
    long mousex, mousey;   //posições x e y do cursor
                           //do mouse
    int telaw, telah; //largura e altura da janela
    bool mudou = false;    //detecta se a tela deve
                           //ser redesenhada
    mousex = evento.GetX();
    mousey = evento.GetY();
    GetSize(&telaw, &telah);
    if((mousex >= 0) && (mousex <= 4) && (x > 0))
    {
        mudou = true;
        x -= 1;
    }
    if((mousex >= (telaw - 4))&&(mousex <= telaw)&&(x
        < (matriz[1].size() - telaw/w -1)))
    {
        mudou = true;
        x += 1;
    }
    if((mousey >= 0) && (mousey <= 4) && (y > 0))
    {
        mudou = true;
        y -= 1;
    }
    if((mousey >= (telah - 4))&&(mousey <= telah)&&(y
        < (matriz.size() - telah/h -1)))
    {
        mudou = true;
        y += 1;
    }
    if((mousey < (telah - 4))&&(mousey > 4)&&(mousex <
        (telaw - 4))&&(mousex > 4))
        mudou = false;
    if(mudou)
    {
        Refresh(FALSE);    //FALSE para que não limpe
                           //a tela antes de
                           //redesenhar
        mudou = false;
    }
}
```

```
void canvas::desenha(wxDC *dc)
{
    wxMemoryDC temp_dc;    //local onde serão
                           //armazenados os bitmaps
    wxBitmap bitmap;
    //calcular a quantidade de imagens que aparecerão
    int contx, conty;
    int telaw, telah;
    GetParent() -> GetSize(&telaw, &telah);
    contx = (int) telaw / w + x;
    conty = (int) telah / h + y;
    if(contx > matriz[1].size())
        contx = matriz[1].size();
    if(conty > matriz.size())
        conty = matriz.size();
    wxBitmap bmpAux(contx * w, conty * h);
    temp_dc.SelectObject(bmpAux);
    //carregando todos os bitmaps
    int destx, desty = 0;
    for(int i = y; i < conty; i++)
    {
        destx = 0;
        for(int j = x; j < contx; j++)
        {
            if(matriz[i][j] == 'w')
                bitmap.LoadFile("blacksquare.bmp",
                    wxBITMAP_TYPE_BMP);
            if(matriz[i][j] == 'b')
                bitmap.LoadFile("bluesquare.bmp",
                    wxBITMAP_TYPE_BMP);
            if(matriz[i][j] == 'r')
                bitmap.LoadFile("redsquare.bmp",
                    wxBITMAP_TYPE_BMP);
            temp_dc.DrawBitmap(bitmap, destx, desty,
                FALSE);
            destx += w;
        }
        desty += h;
    }
    //mostrando tudo de uma só vez
    dc -> Blit(0, 0, telaw, telah, &temp_dc, 0, 0,
        wxCOPY, FALSE);
}
```

Anexo C – Trechos dos Códigos do Teste de Som com as Bibliotecas SDL e wxWindows.

- SDL:

```
//var. globias
struct
{
      SDL_AudioSpec spec;
      Uint8 *sound;   //ponteiro para o dado do wave
      Uint32 soundlen;      //tamanho do dado do wave
      int soundpos;   //Posição corrente que está
                      //tocando
} wave;
void audio_callback(void *unused, Uint8 *stream, int
      len)
{
      Uint8 *waveptr;
      int    waveleft;
      //Configurar os ponteiros
      waveptr = wave.sound + wave.soundpos;
      waveleft = wave.soundlen - wave.soundpos;
      //Tocar!
      while ( waveleft <= len )
      {
            SDL_MixAudio(stream, waveptr, waveleft,
                  SDL_MIX_MAXVOLUME);
            stream += waveleft;
            len -= waveleft;
            waveptr = wave.sound;
            waveleft = wave.soundlen;
            wave.soundpos = 0;
      }
      SDL_MixAudio(stream, waveptr, len,
            SDL_MIX_MAXVOLUME);
      wave.soundpos += len;
}
```

- wxWindows:

```
bool apl::OnInit()
{
    //criando um frame
    frame *meuFrame = new frame();
    meuFrame->Show();
    SetTopWindow(meuFrame);
    m_wave.Create("teste.wav");
    return true;
}
void frame::toca(wxCommandEvent &evento)
{
    if (wxGetApp().m_wave.IsOk())
    {
        wxGetApp().m_wave.Play(false);
    }
    else
    {
        wxMessageDialog *erro = new
            wxMessageDialog(this, "Erro ao abrir
            arquivo", "Message box", wxOK,
            wxDefaultPosition);
        erro -> ShowModal();
    }
}
```