Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

Texto de qualificação de Mestrado

Processamento de áudio em tempo real em dispositivos não convencionais

Autor: André Jucovsky Bianchi

Orientador: Prof. Dr. Marcelo Gomes de Queiroz

Texto apresentado ao programa de Pós Graduação em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo para Exame de Qualificação de Mestrado.

17 de agosto de 2011

Resumo

Este trabalho tem como objetivo estudar a realização de processamento de áudio digital em tempo real utilizando três plataformas com características computacionais fundamentalmente distintas porém bastante acessíveis em termos de custo e disponibilidade de tecnologia: Arduino, GPU e Android. Arduino é um dispositivo com licenças de hardware e software abertas, baseado em um microcontrolador com baixo poder de processamento, muito utilizado como plataforma educativa e artística para computações de controle e interface com outros dispositivos. GPU é uma arquitetura de placas de vídeo com foco no processamento paralelo, que tem motivado o estudo de modelos de programação específicos para sua utilização como dispositivo de processamento de propósito geral. Android é um sistema operacional para dispositivos móveis baseado no kernel do Linux, que permite o desenvolvimento de aplicativos utilizando linguagem de alto nível e possibilita o uso da infraestrutura de sensores, conectividade e mobilidade disponível nos aparelhos. Buscaremos sistematizar as limitações e possibilidades de cada plataforma através da utilização de técnicas de análise encontradas na literatura e da implementação de técnicas de processamento de áudio digital em tempo real em cada ambiente.

Sumário

Capítulo 1

Introdução

Este texto é um relatório apresentado para Exame de Qualificação no Programa de Mestrado em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo. A elaboração desta proposta vem ocorrendo desde o final de 2010 e ainda possui grande abertura a contribuições em termos de metodologia e direcionamento do estudo como um todo.

O processamento de sinais digitais em tempo real é uma área de pesquisa que interessa tanto a técnicos quanto a artistas. Para ampliar a apropriação da tecnologia pelos artistas, é interessante estudar soluções que sejam acessíveis em termos de custo e disponibilidade de tecnologia. Este estudo pretende explorar os limites e possibilidades de processamento de áudio digital em tempo real utilizando dispositivos que possuam características computacionais bastante distintas mas que sejam acessíveis para os usuários finais em termos de custo e tecnologia.

Esta abordagem nos leva na direção de propor a utilização de plataformas que não foram projetadas especificamente para o processamento de sinais digitais, mas que também podem ser utilizadas para este fim. Para a escolha dos dispositivos propostos, levamos em conta não só o valor de aquisição e disponibilidade no mercado (em comparação com soluções profissionais), mas também outros aspectos relevantes como licença de utilização e características do processo de produção. É importante notar que a realidade quanto à disponibilidade das plataformas é diferente em outros contextos com menos recursos, diferenças de legislação ou influência de outros fatores. Tendo em mente este sentido de disponibilidade, pretendemos quantificar e qualificar as possibilidades de processamento de áudio digital em tempo real para performances artísticas ao vivo que disponham de um mínimo de investimento de recursos ou que cujo contexto seja tal que estas tecnologias já estejam amplamente disponíveis como resultado de sua forte presença no mercado.

Existem outras características que são desejáveis para a escolha das plataformas de estudo, como sua capacidade de interação com outros sistemas computacionais, seu nível de obsolescência (é interessante conseguir utilizar um dispositivo mesmo depois que tenha sido substituído por um modelo mais novo no mercado) e adaptabilidade a diferentes cenários de utilização. Também é interessante que as plataformas possuam características bastante distintas uma da outra para motivar diferentes abordagens das técnicas de processamento e enriquecer o estudo.

Nas próximas seções deste capítulo, descreveremos as plataformas propostas para estudo neste trabalho, veremos de que forma estas plataformas já estão sendo utilizadas para processamento de sinais digitais em estudos relacionados e finalizaremos o primeiro capítulo com uma rápida descrição dos objetivos gerais deste estudo. No capítulo 2, revisaremos alguns estudos que fundamentarão a análise de cada plataforma. No capítulo 3, descreveremos em detalhes as plataformas e levantaremos algumas possibilidades de análise específicas para cada plataforma. Finalmente, no capítulo 4 apontaremos algumas direções possíveis para dar continuidade ao

trabalho e apresentaremos uma proposta de cronograma.

1.1 Processamento em tempo real na performance interativa

As possibilidades estéticas da utilização de instrumentos elétricos e eletrônicos surgem quando artistas entram em contato com ferramentas e conhecimento suficiente para realizar esta exploração. Alguns exemplos de uso artístico da tecnologia são a subversão de circuitos eletrônicos com o objetivo de criar novos instrumentos, como na prática denominada Circuit Bending[1], e o desenvolvimento de sistemas complexos dotados de autonomia musical e capacidade de interação, como os sistemas *Cypher* (ROWE, 1992a) e *Voyager* (LEWIS, 2000). Esse tipo de performance interativa, na qual artista e aparato tecnológico dividem o palco e interagem em tempo real, muitas vezes ocorre como fruto de pesquisa realizada em trabalhos acadêmicos.

São muitas as possibilidades de uso da tecnologia na produção artística sonora como, por exemplo, o desenvolvimento de controladores de dispositivos sonoros, a criação de instrumentos expandidos, nos quais a utilização da tecnologia ajuda a gerar novos sons ou novas formas de interação, e o desenvolvimento de certa inteligência musical, através da detecção de características de um conjunto de sinais e atuação no ambiente como resultado de raciocínio computacional. A forma de utilização de tecnologia que interessa a este trabalho é o processamento de sinais digitais realizado em tempo real, de modo que o artista possa utilizar o resultado da computação na performance ao vivo.

Em termos de proposta, os sistemas computacionais desenvolvidos para performance interativa se encontram entre dois extremos. Em um extremo estão os sistemas desenvolvidos com objetivo de auxiliar um trabalho de um artista específico. No outro extremo, estão sistemas dotados de alta flexibilidade que se propõe a auxiliar artistas com as mais diferentes concepções estéticas. O processamento de sinais digitais em tempo real figura como uma ferramenta disponível para artistas utilizarem em qualquer sistema de performance interativa. A utilização de plataformas altamente disponíveis para este tipo de processamento aproxima artistas de novas possibilidades na construção de sistemas musicais de qualquer tipo.

1.2 Plataformas consideradas para estudo de caso

Para estudar o processamento de sinais digitais em tempo real e avaliar as restrições e possibilidades em relação aos algoritmos e às tecnologias disponíveis hoje em dia, escolhemos três plataformas para estudo de caso. A escolha das plataformas foi feita levando em conta seu baixo custo em relação a plataformas comerciais, a alta disponibilidade para usuários comuns e a existência de diferenças fundamentais de projeto, propósito e características entre elas.

A primeira plataforma analisada é o Arduino, uma combinação de hardware e software com licença aberta que compreende um microcontrolador e uma interface que possibilita a captura, processamento e síntese de sinais analógicos e digitais. A segunda plataforma considerada é o circuito do tipo GPU, disponível em grande parte das placas de vídeo comercializadas atualmente, que permite processamento paralelo e pode atuar como coprocessador de dados em conjunto com a arquitetura tradicional dos computadores pessoais. A terceira plataforma estudada é o sistema operacional Android para dispositivos móveis, que permite grande flexibilidade e abstração no desenvolvimento por se tratar de um sistema operacional completo e com partes de seu código publicado sob licenças abertas, além de apresentar abundância de sensores e interfaces de conectividade.

[1] http://www.anti-theory.com/soundart/circuitbend/cb01.html

Nas próximas seções, revisaremos brevemente cada uma das plataformas escolhidas com o objetivo de dar uma visão sobre o histórico e as motivações do desenvolvimento e da escolha de cada uma delas para análise neste trabalho.

1.2.1 Arduino: pouca tecnologia, muita flexibilidade

O projeto **Arduino**[2] foi iniciado em 2005 com o objetivo de criar uma plataforma de desenvolvimento de projetos para estudantes mais barata dos que as disponíveis na época. Baseado nas idéias de prover uma estrutura minimal para interface com um microcontrolador, o Arduino veio de uma ramificação do projeto Wiring[3], uma plataforma de desenvolvimento criada em 2003 com o objetivo de unir designers e artistas ao redor do mundo para compartilhar idéias, conhecimento e experiência utilizando hardware e software com licenças abertas. O software utilizado pelo projeto Wiring, por sua vez, foi influenciado pela plataforma de desenvolvimento Processing[4], outro projeto aberto iniciado em 2001 por ex-integrantes do MIT Media Lab[5].

Tanto o hardware quanto o software do Arduino são publicados sob licenças Open Source[6]. Os projetos originais das placas do tipo Arduino estão publicados sob a licença Creative Commons Attribution Share-Alike 2.5[7], o que significa que o projeto do hardware não requer nenhuma permissão para o uso e permite trabalhos derivados tanto para uso pessoal quanto para uso comercial, contanto que haja crédito para o projeto oficial e que os projetos derivados sejam publicados sob a mesma licença. Já o software utiliza duas versões diferentes de licenças de software livre: o ambiente escrito em Java é liberado sob a licença GPL[8] e as bibliotecas em C/C++ do microcontrolador são liberados sob a licença LGPL[9]. Toda a documentação no sítio do projeto é publicada sob a licença Creative Commons Attribution-ShareAlike 3.0[10] e os trechos de código estão em domínio público. Atualmente, existem diversas empresas[11] e indivíduos fabricando o Arduino e, devido a este esquema de licenciamento, qualquer um com acesso às ferramentas adequadas pode construir uma placa a partir de componentes eletrônicos básicos.

A disponibilidade dos projetos das placas e do código fonte do compilador, somada a uma escolha adequada das licenças de distribuição, resultou em uma comunidade mundial que desenvolve plataformas de hardware e software que podem ser utilizados para as mais diversas aplicações (respeitadas, é claro, as limitações do poder computacional dos microcontroladores utilizados). Também decorre destas escolhas a possibilidade de fabricação, industrial ou caseira, por um preço acessível. Hoje existem muitos outros projetos baseados no Arduino que utilizam microcontroladores do mesmo tipo ou outros modelos de fabricantes diferentes e com características distintas, mas totalmente compatíveis no nível do software[12]. Também é possível encontrar na internet uma variedade muito grande de módulos conectáveis às placas Arduino, tanto para comprar quanto para construir. Estes módulos têm como objetivo prover outras funções: desde as mais básicas como implementação de relógios digitais que possibilitam um controle mais fino da passagem do tempo[13], até funções mais avançadas como interconexão sem fio com outras plataformas[14].

[2] http://arduino.cc/
[3] http://wiring.org.co/
[4] http://www.processing.org/
[5] http://www.media.mit.edu/
[6] http://www.opensource.org/docs/osd
[7] http://creativecommons.org/licenses/by-sa/2.5/
[8] http://www.gnu.org/licenses/gpl.html
[9] http://www.gnu.org/licenses/lgpl.html
[10] http://creativecommons.org/licenses/by-sa/3.0/
[11] http://www.arduino.cc/en/Main/Buy
[12] http://www.arduino.cc/playground/Main/SimilarBoards
[13] http://totusterra.com/index.php/2009/10/31/using-the-555-timer-as-an-external-clock
[14] http://arduino.cc/en/Main/ArduinoXbeeShield,http://jt5.ru/shields/cosmo-wifi/,http://jt5.ru/shields/cosmo-

Ao longo da existência do projeto Arduino, diversas versões do hardware foram desenvolvidas e publicadas[15]. Cada versão possui características diferentes como, por exemplo, tipo de conexão com o computador (USB, serial, FireWire), modelos diferentes do microcontrolador utilizado, formato, tamanho, facilidade de montagem (em termos de ferramentas e técnica necessárias), e até estética. Apesar disso, todos possuem a mesma interface básica e são compatíveis com os mesmos módulos e código.

O projeto Wiring, que deu origem ao projeto Arduino, possuía como objetivo o compartilhamento de idéias, experiências e conhecimento entre artistas e designers ao redor do mundo. O projeto Arduino, por sua vez, tem como objetivo, desde sua concepção, desenvolver uma plataforma educacional economicamente viável. A junção destas idéias resulta em uma característica fundamental do Arduino, que é a confluência de criatividade e técnica.

O arcabouço de desenvolvimento do projeto Arduino é distribuído livremente e a transferência do programa para o microcontrolador pode ser feita utilizando um cabo USB. Além disso, a existência de uma comunidade que dá suporte aliada a outros motivos estéticos, práticos, econômicos e técnicos, fizeram com que o Arduino se difundisse muito entre hobbystas e artistas[16,17]. Nos últimos anos, projetos utilizando o Arduino têm ocupado galerias, museus e diversos outros espaços, artísticos ou não (GIBB, 2010). Dada a grande disponibilidade de projetos compatíveis com Arduino (módulos de hardware e pedaços de código), a plataforma tem sido utilizada nos mais diversos contextos, como por exemplo para controlar os motores de impressoras tridimensionais[18], compor novos instrumentos musicais [19,20,21], controlar aeromodelos[22], entre muitos outros usos.

Finalmente, é importante comentar que as pessoas que mantém o projeto Arduino expressam uma preocupação com a desigualdade de condições de estudo e trabalho em diferentes partes do mundo. Um indivíduo ou empresa que queira utilizar o nome Arduino deve contribuir com o projeto de alguma forma: pagando taxas, liberando os projetos ou código desenvolvidos, documentando e dando suporte ao produto ou qualquer combinação dessas possibilidades. No sítio do projeto, é explicitado que uma parte das entradas de dinheiro são destinadas a fomentar a computação em lugares onde os custos de hardware são proibitivos. Além disso, para utilizar a marca, é pedido ao fabricante que faça todo o esforço para ter o hardware montado sob condições de trabalho justas. Estes procedimentos, frutos de reflexão crítica sobre o papel e o impacto do projeto na sociedade, são importantes não só porque levam em conta o sistema de produção (de bens materiais e simbólicos) como um todo, mas também porque atribuem à ferramenta didática uma potencialidade de transformação da realidade.

Através da experimentação com captura, processamento e produção de sinais analógicos e digitais de áudio utilizando o Arduino, pretendemos quantificar a capacidade computacional e qualidade do processamento de sinais digitais em tempo real da plataforma. Como se trata de uma plataforma com baixo poder de processamento, grande parte dos projetos desenvolvidos com Arduino o utilizam como ferramenta de automação, usando suas entradas e saídas para captura de sinais de sensores e emissão de sinais de controle baseados em instruções computacionalmente simples. Apesar disto, é possível encontrar trabalhos específicos sobre processamento de sinais utilizando Arduino[23], e sua natureza lúdica reafirma o interesse na exploração da plataforma

gsm/

[15] http://arduino.cc/en/Main/Hardware
[16] http://www.arduino.cc/playground/Projects/ArduinoUser
[17] http://www.studiobricolage.org/arduino-1
[18] http://wiki.makerbot.com/thingomatic
[19] http://www.surek.co.uk/trampoline/
[20] http://xciba.de/pumpbeats/
[21] http://servoelectricguitar.com/
[22] http://aeroquad.com/
[23] http://interface.khm.de/index.php/lab/experiments/arduino-realtime-audio-processing/

como ferramenta educacional e artística.

1.2.2 GPGPU: computação de propósito geral utilizando processadores gráficos

GPU é um acrônimo para **Graphics Processing Unit** (unidade de processamento de gráficos) e se trata de um tipo de circuito projetado especificamente para o processamento paralelo de imagens para a exibição na tela de um computador. O processamento de imagens, bi ou tridimensionais, possui uma série de características e necessidades específicas que foram determinantes para as escolhas feitas durante o desenvolvimento dos circuitos específicos para estes fins.

A computação paralela é fundamentada pela associação entre estruturas e operações matemáticas altamente paralelizáveis e os dispositivos capazes de realizar tais computações. O início do estudo formal e as primeiras propostas de circuitos ocorreram ainda na década de 1950 e resultaram tanto em propostas teóricas, como por exemplo a máquina universal sugerida por Holland (HOLLAND, 1959), quanto na construção dos primeiros computadores paralelos, projetados pela IBM[24] (multiprogramação com apenas um processador[25]) e UNIVAC[26] (uso de dois processadores em paralelo), produzidos em 1959.

Nos anos seguintes, houve avanços importantes em pesquisas relativas a paralelismo e processamento de sinais, como por exemplo o desenvolvimento da *Transformada Rápida de Fourier* (COOLEY; TUKEY, 1965) em 1965 e a descrição, em 1966, da taxonomia utilizada até hoje para classificação das possíveis relações entre instruções e dados nos circuitos de computação paralela (FLYNN, 1966), entre outros.

Naquele momento, toda infraestrutura computacional e pesquisas em andamento eram destinadas a aplicações científicas, militares e industriais e até o meio da década de 1960 os resultados de todas essas computações só podiam ser gravados em fita magnética, perfurados em cartões ou impressos em papel. Em 1967 surge o primeiro teletipo[27] que, através da simulação de um terminal de impressão, imprime os resultados em uma tela ao invés de imprimir no papel. Os primeiros computadores pessoais começaram a ser comercializados nos anos 1970 e em meados dos anos 1980 a indústria de hardware começou a produzir os primeiros modelos equipados com placas de vídeo, com instruções primitivas para auxiliar na produção de gráficos em duas dimensões[28,29,30].

No início da década de 1990, o desenvolvimento das placas de vídeo se alinha direta e definitivamente com as pesquisas na área da computação paralela. Ao longo de toda a década, intensifica-se o desenvolvimento de circuitos especializados para processamento de imagens tridimensionais com foco no mercado de usuários domésticos, impulsionado pelo mercado de jogos para computador. O modelo de *pipeline* desenvolvido para as placas de vídeo ao longo desta década apresenta, além de paralelismo de *dados*, paralelismo de *tarefas*. Os dados são divididos em blocos que são inseridos sequencialmente na pipeline e passam por uma série de estágios diferentes da computação (as tarefas). Nesta fila, a saída de um estágio é inserida diretamente na entrada do estágio seguinte. Na execução de cada tarefa sobre um bloco, a aplicação de uma mesma operação ocorre em paralelo em todos os dados do bloco. Além disso, a cada momento, toda a pipeline pode estar ocupada com blocos de dados em estágios diferentes da computação e todas estas tarefas são executadas também em paralelo (OWENS et al., 2008).

[24] http://en.wikipedia.org/wiki/IBM_7030
[25] http://www.cs.clemson.edu/~mark/stretch.html
[26] http://en.wikipedia.org/wiki/UNIVAC_LARC
[27] http://en.wikipedia.org/wiki/Datapoint_3300
[28] http://tinyurl.com/3k6ek2x
[29] http://en.wikipedia.org/wiki/Commodore_Amiga#Graphics
[30] http://en.wikipedia.org/wiki/8514_%28display_standard%29

Algumas características são especialmente importantes para compreender a estrutura dos circuitos GPU atuais. Uma primeira característica é o alto requerimento computacional para o processamento de gráficos tridimensionais em tempo real: milhões de pixeis têm que ser computados por segundo e a computação de cada um pode ser bastante complexa. Apesar deste alto requerimento computacional, existe também alto grau de paralelismo presente nas estruturas e operações matemáticas relacionadas. Uma transformação linear sobre um conjunto de vértices, por exemplo, é uma operação que pode ser realizada sobre vários elementos em paralelo. Outra característica ainda é a necessidade do resultado das computações ser recebido em tempo real, o que requer uma alta taxa de fluxo de dados. Estas três características levaram o desenvolvimento dos circuitos de processamento de gráficos para um caminho bastante distinto do das CPUs (OWENS et al., 2008).

As CPUs são otimizadas para alta performance de código sequencial, com muitos transistores dedicados à obtenção de paralelismo no nível das instruções. Por outro lado, a natureza altamente paralelizável da computação gráfica permite à GPU utilizar os recursos adicionais diretamente para computação, atingindo intensidade aritmética mais alta com o mesmo número de transistores. Por causa de diferenças fundamentais entre as arquiteturas, a velocidade de crescimento da capacidade computacional dos hardwares gráficos é muito mais alta que a das CPUs (OWENS et al., 2007). Em junho de 2011, quando da elaboração deste texto, a melhor placa gráfica comercializada pela NVIDIA atingia pico de processamento de mais de 500 GFLOPS[31], enquanto que a melhor CPU Intel disponível não ultrapassa 100 GFLOPS[32].

Em 1999 o termo GPU era utilizado pela primeira vez pelas fabricantes de placas de vídeo para designar um modelo de processamento paralelo e sua implementação em placas com circuitos especializados para o processamento de transformações lineares, cálculos de iluminação e renderização de gráficos tridimensionais.

No início da década de 2000, os primeiros modelos de GPU possuíam funções fixas em muitos de seus estágios de computação e os programadores tinham de gastar tempo desenvolvendo estratégias para adaptar a estrutura de seus problemas às possibilidades computacionais oferecidas pelo hardware. Ao longo da década, ocorreram grandes avanços nas pesquisas sobre programação de propósito geral com dispositivos gráficos, o que reafirmou o reconhecimento de sua aplicabilidade científica. Neste contexto, a arquitetura das GPUs foi sendo transformada, e desde então o desafio dos fabricantes tem sido atingir um bom balanço entre prover acesso de baixo nível ao hardware para aumento de performance e o desenvolver ferramentas e linguagens de alto nível para permitir flexibilidade e produtividade na programação. Hoje o que temos disponível é um dispositivo com unidades completamente programáveis que suportam operações vetoriais de ponto flutuante, acompanhadas de linguagens de alto nível e arcabouços de programação que abstraem e facilitam ainda mais a tarefa de programação para GPU.

O domínio numérico da computação gráfica é prioritariamente o de ponto flutuante e por isso os primeiros modelos de GPU não possuíam, por exemplo, suporte a aritmética de inteiros e operações do tipo *bitwise*. Uma outra dificuldade das primeiras gerações era a inflexibilidade do acesso aleatório para escrita na memória, pois a posição final de cada vértice na tela, que corresponde ao endereço na memória onde cada pixel resultante será gravado, era definida num estágio inicial da computação e não podia ser alterada posteriormente. Estas questões foram superadas nos últimos anos através da implementação do suporte a diversos tipos de operação e pesquisa científica sobre formas de utilizar o hardware gráfico. Atualmente, já existem implementações de algoritmos criptográficos complexos que rodam inteiramente em GPU fazendo uso intenso de aritmética de inteiros[33], além de estudos sobre a eficiência de operações paralelas

[31] http://www.nvidia.com.br/object/tesla_gpu_processor_br.html
[32] http://www.intel.com/support/processors/xeon/sb/CS-020863.htm
[33] http://http.developer.nvidia.com/GPUGems3/gpugems_ch36.html

como *gather* e *scatter* utilizando GPUs (HE et al., 2007).

Já fazem alguns anos que a maioria absoluta dos computadores pessoais novos (mais de 90% dos desktops e notebooks) e uma grande parte dos dispositivos móveis (PDAs, telefones celulares, etc) incluem um processador que atua exclusivamente na aceleração do processamento de gráficos integrado à placa mãe[34]. Grande parte desses dispositivos já estão sendo fabricados com GPU. Esta alta disponibilidade da GPU nos computadores modernos tem permitido enorme avanço nas pesquisas e sua utilização como um dispositivo genérico de coprocessamento, não apenas para aceleração do processamento de imagens, mas para propósitos mais gerais. É possível utilizar a infraestrutura da GPU para cálculos diversos, utilizando paradigmas de computação paralela (como MIMD e SIMD) através do modelo de processamento de fluxos de dados, como veremos na seção 3.4.2 (VENKATASUBRAMANIAN, 2003). Já não é novidade o uso da GPU para a construção de supercomputadores com milhares de processadores em paralelo a um custo acessível para utilização científica[35], e uma busca rápida revela uma quantidade muito grande de problemas mapeados para resolução nesta infraestrutura paralela.

Neste contexto, um novo termo aparece para designar a utilização da GPU para propósitos outros que não o processamento de vídeo para o qual a plataforma foi inicialmente desenvolvida. O termo **GPGPU**, acrônimo para *General Purpose Computation on GPU*, faz referência a este tipo de utilização de circuitos GPU para propósitos gerais. No campo da Computação Musical, já é possível encontrar menções à utilização da GPU para processamento de áudio tridimensional e auralização através de técnicas como HRTF[36], por exemplo (GALLO; TSINGOS, 2004).

Mais recentemente, outras soluções têm sido propostas com foco na integração entre as arquiteturas da GPU e CPU[37,38,39]. De qualquer forma, tanto as contribuições metodológicas trazidas ao longo do desenvolvimento da GPU quanto os arcabouços teóricos e computacionais desenvolvidos são de extrema valia para a computação paralela em geral. Além disso, as possibilidades computacionais trazidas pela GPU e sua curva de crescimento muito mais rápida do que a da CPU tradicional constituem ainda mais motivos para continuar o estudo deste tipo de plataforma.

A proposta deste trabalho em relação à GPU é estudar as possibilidades e limites do processamento de áudio digital em tempo real utilizando este tipo de plataforma. Para isto, pretendemos abordar modelos computacionais como o de processamento de fluxos de dados (que será descrito na seção 3.4.2) com o objetivo de aproximar a computação paralela de ferramentas já existentes para processamento de sinais digitais em tempo real. Pretendemos quantificar a complexidade computacional e possibilidades de paralelismo de alguns algoritmos de efeitos de áudio, implementar algumas soluções utilizando os arcabouços disponíveis (descritos na seção 3.4.3) e realizar uma pequena comparação com soluções comerciais em termos de custo/benefício. Além disso, pretendemos estudar a possibilidade de utilização da GPU com programas com licença livre bastante utilizados para processamento digital em tempo real como Pure Data[40] e CSound[41], através da avaliação de implementação de interface com os arcabouços disponíveis.

[34] http://computershopper.com/feature/the-right-gpu-for-you
[35] http://fastra.ua.ac.be/
[36] *Head Related Transfer Functions*
[37] http://www.nvidia.com/object/pr_nexus_093009.html
[38] http://www.amd.com/us/Documents/49282_G-Series_platform_brief.pdf
[39] http://software.intel.com/file/37300
[40] http://puredata.info/
[41] http://www.csounds.com/

1.2.3 Android: abstração, conectividade, sensores e mobilidade

Com a evolução da computação e da engenharia, temos à nossa disposição dispositivos computacionais altamente complexos que cabem na palma da mão. **Android** é o nome de um sistema operacional para dispositivos móveis, desenvolvido a partir de 2003 por empresa homônima. Em 2005, sistema e empresa foram comprados pela Google Inc., multinacional americana que atua em diversas áreas tecnológicas que formam base para sua atividade principal em termos de receita: o mercado de publicidade eletrônica.

O sistema operacional Android é composto por um kernel fortemente baseado no kernel do Linux[42], um conjunto de drivers para muitos modelos de aparelhos, uma série de programas utilitários para o gerenciamento do sistema operacional e uma vasta gama de aplicações para o usuário final (HALL; ANDERSON, 2009). Apesar de se tratar de um conjunto de programas que compõem um sistema operacional completo, o nome Android é também comumente utilizado para fazer referência aos dispositivos móveis distribuídos com este sistema operacional.

O projeto Android baseia seus aplicativos na linguagem Java, o que permite a utilização do mesmo código binário em diferentes arquiteturas. Também disponibiliza bibliotecas para interface com funcionalidades que estão se tornando cada vez mais comuns em dispositivos móveis, como processamento gráfico otimizado, conexão via bluetooth, 3G, GSM, WiFi, câmera, GPS, bússola e acelerômetro.

A licença principal utilizada no projeto Android é a *Apache Software License 2.0*[43], considerada pela Free Software Foundation[44] como uma licença de software livre[45] compatível com a versão 3 da GPL[46]. Apesar disto, outras licenças estão presentes no projeto, como no caso dos patches do Linux Kernel que têm que ser licenciados sob a GPL versão 2.0[47], por força do licenciamento do próprio kernel do Linux.

Discussões recentes têm levantado preocupações quanto à possível infração de alguns termos de licenças livres na utilização de código do kernel do Linux no sistema operacional Android. Alguns exemplos são a negação por parte da empresa Google da distribuição do código fonte do seu produto até o lançamento de versões mais novas do sistema[48,49] (inclusive colocando em cheque a "liberdade" da licença Apache como um todo) e a possível infração ao importar partes de arquivos de cabeçalho do kernel do Linux e licenciar o código derivado com licenças não previstas pela GPL[50].

Outros sistemas operacionais têm sido desenvolvidos como ramificações do Android, com o objetivo de priorizar outras questões. O projeto Replicant[51], por exemplo, é uma distribuição do sistema operacional Android com 100% do código publicado sob licenças livres. Também é possível instalar o sistema Debian Linux em dispositivos que suportam o Android[52], ou modificar o sistema para utilizar soluções de segurança como criptografia de disco[53], por exemplo.

Ao longo deste trabalho, sempre que falarmos em Android estaremos nos referindo a todas as plataformas que compreendem o ecossistema complexo descrito nesta seção. As técnicas e

[42] http://kernel.org/
[43] http://www.apache.org/licenses/LICENSE-2.0
[44] http://www.fsf.org/
[45] http://www.fsf.org/about/what-is-free-software
[46] http://www.gnu.org/licenses/license-list.html,
[47] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[48] http://www.theregister.co.uk/2011/05/10/android_ice_cream_sandwich/
[49] http://www.itworld.com/open-source/164153/how-google-can-delay-android-source-code-releases
[50] http://ebb.org/bkuhn/blog/2011/03/18/bionic-debate.html
[51] http://replicant.us/about/
[52] http://lanrat.com/android/debian
[53] https://github.com/guardianproject/LUKS/wiki

metodologias que pretendemos desenvolver ao longo deste trabalho não devem depender de um tipo de instalação específica, dos modelos dos dispositivos móveis ou do sistema operacional em si. Ao contrário, a idéia é analisar as possibilidades de uso de plataformas móveis em geral como dispositivos para processamento de áudio e contemplar sistemas nos quais seja possível capturar áudio, receber sinais de outros sensores e aparelhos, executar código desenvolvido pelo usuário, e retornar o resultado do processamento destes sinais através de saídas de áudio ou conexões do tipo WiFi, bluetooth, infravermelho ou quaisquer outras que estejam disponíveis.

Uma primeira proposta de utilização da plataforma é a implementação de bancos de filtros, a serem aplicados ao áudio capturado pelo microfone durante uma ligação, antes de sua transmissão pela rede de telefonia celular. A mesma técnica poderia ser utilizada em programas de comunicação que funcionam com outros tipos de tecnologia como VOIP, por exemplo. A integração com outros programas pode gerar aplicações tanto para fins artísticos, como mixagem colaborativa de áudio, quanto para fins práticos, como criptografia de voz.

1.3 Trabalhos relacionados

Como vimos na seção 1.2, a escolha das plataformas para este estudo foi feita tendo em mente algumas características em comum, como alta disponibilidade e (relativo) baixo custo, mas também considerando as particularidades de cada uma em termos de tecnologia computacional, bem como a riqueza de possibilidades de exploração resultante dessas particularidades. Talvez pela diferença de idade entre as três opções levantadas (20 anos de desenvolvimento de placas de vídeo aceleradas contra pouco mais de 6 anos dos projetos Arduino e Android), ou talvez por assimetrias no interesse acadêmico sobre cada uma, a impressão é de que há muito mais material disponível sobre GPGPU do que sobre as outras duas plataformas.

Enquanto a pesquisa em Arduino parece interessante no sentido lúdico, didático, experimental e artístico, para o Android já é possível encontrar uma gama maior de aplicações de processamento de sinais digitais. Já o processamento paralelo na GPU é um fato e uma tendência, tanto no âmbito acadêmico quanto no de mercado. Essas diferenças se refletem neste texto na medida em que o tema GPGPU ganha mais espaço por sua complexidade, idade, interesse científico e consequente abundância de material.

Nas próximas seções, veremos alguns estudos que abordam soluções relacionadas às propostas deste trabalho.

1.3.1 Processamento de sinais no Arduino

O Laboratório para Ciência da Computação Experimental da Academia de Artes Midiáticas de Colônia[54] possui um estudo (não publicado em periódico) sobre a utilização do Arduino para processamento de sinais em tempo real[55]. Pequenas adaptações do hardware são feitas para possibilitar a entrada e saída de sinais analógicos utilizando as conversões ADC e DAC disponíveis no microcontrolador do Arduino e alguns tipos de filtro são implementados. Este estudo é um bom ponto de partida para implementações como as que estamos propondo neste trabalho.

Outra iniciativa interessante é a da utilização do Arduino como placa de som, através da implementação de um driver para Linux utilizando a infraestrutura do projeto ALSA[56] (DIM-

[54] http://interface.khm.de/
[55] http://interface.khm.de/index.php/lab/experiments/arduino-realtime-audio-processing/
[56] http://www.alsa-project.org/

ITROV; SERAFIN, 2011b). O resultado fica aquém das placas de som comerciais, principalmente com relação à resolução dos sinais capturados e emitidos (8 bits no Arduino contra 32 bits de placas comerciais), mas o trabalho abre caminho para uma implementação completa e funcional de uma placa de som com licença aberta. Um estudo subsequente analisa questões relacionadas à entrada e saída de áudio em placas de som e propõe, como complementação do estudo anterior, um projeto de uma placa complementar para o Arduino que cuidaria exclusivamente da entrada e saída de sinais (DIMITROV; SERAFIN, 2011a).

1.3.2 Processamento de sinais em hardware gráfico

Uma das técnicas básicas de realização de processamento de sinais digitais é a Transformada Rápida de Fourier (*Fast Fourier Transform* – `FFT`) da qual falaremos com mais detalhes na seção 2.3.1. É possível encontrar estudos discutindo detalhes da implementação da `FFT` em placas gráficas do tipo GPU (MORELAND; ANGEL, 2003), incluindo comparações com implementações altamente otimizadas para a CPU tradicional, como a biblioteca `FFTW`[57], escrita em C e publicada sob licença livre. Um outro estudo implementa a Transformada Discreta do Coseno (*Discrete Cosine Transform* – `DCT`), outra técnica de análise, representação e compressão de sinais digitais, e deriva uma relação para a performance ótima na divisão de operações entre GPU e CPU (MOHANTY, 2009). Ainda neste caminho, encontramos também a implementação para GPU da Transformada Discreta de Wavelets (*Discrete Wavelet Transform* – `DWT`), mais uma ferramenta para análise de sinais digitais (WONG et al., 2007). Na direção de utilização da GPU como dispositivo para processamento de áudio, outro estudo avalia possibilidades de espacialização de áudio em tempo real, aplicando HRTFs com ajuda de funções nativas de reamostragem de texturas (GALLO; TSINGOS, 2004).

No sentido de utilizar a GPU como dispositivo para processamento de propósito geral, ou seja, para outros tipos de computação que não somente o processamento de gráficos tridimensionais, é possível encontrar estudos em diversas direções. Desde simples multiplicações de matrizes e implementações de solucionadores do problema de decidir a linguagem `3-SAT` (THOMPSON; HAHN; OSKIN, 2002), passando por análises de imagens médicas e simulações físicas e biológicas (OWENS et al., 2007; OWENS et al., 2008), até soluções para recuperação de informação (GOVINDARAJU et al., 2005), são muitas as iniciativas de adaptação de problemas para soluções utilizando GPU.

1.3.3 Processamento de sinais em dispositivos móveis

Com o objetivo de trazer para os dispositivos móveis uma ferramenta muito utilizada para o processamento de sinais digitais em computadores pessoais, o Grupo de Tecnologia Musical da Universidade Pompeu Fabra de Barcelona[58] realizou uma adaptação do Pure Data para a arquitetura PocketPC (GEIGER, 2003). Analisando a qualidade sonora, a capacidade computacional e a performance do sistema na arquitetura escolhida, este estudo abre caminho para a utilização de sistemas móveis no processamento em tempo real com flexibilidade comparável à dos computadores pessoais.

Um outro estudo aborda uma aplicação bastante distinta das propostas neste texto, mas utiliza técnicas que são de interesse para nosso trabalho. Visando a implementação de sistemas de vigilância em dispositivos móveis, um estudo do Departamento de Engenharia da Informação da Universidade de Modena e Reggio Emilia na Itália discute as possibilidades de processamento

[57] http://www.fftw.org/
[58] http://www.mtg.upf.edu/

de imagens e transmissão de sinais via internet utilizando dispositivos móveis (CUCCHIARA; GUALDI, 2010).

Finalmente, a discussão sobre as ferramentas, conceitos e atividades envolvidas na criação de música levou ao desenvolvimento de um aplicativo de mixagem de sons específico para a plataforma Android, chamado mixDroid (FLORES et al., 2010). Neste trabalho, o aplicativo serve como prova de conceito sobre associações entre conceitos da área de Ciências Humanas e a prática de fazer música com auxílio de ferramentas computacionais.

1.4 Objetivos

Tendo em mente as plataformas escolhidas e os trabalhos relacionados, listamos abaixo alguns objetivos gerais deste trabalho:

- Quantificar a capacidade computacional e os limites de processamento de sinais de áudio de cada plataforma proposta.
- Utilizar os arcabouços e metodologias encontrados na literatura para estudar e implementar algoritmos e técnicas de processamento de áudio digital nas plataformas escolhidas. Realizar uma comparação das soluções desenvolvidas para estas plataformas com soluções tradicionais. Os algoritmos específicos a serem implementados ainda não estão determinados, mas no fim desta seção serão apresentadas algumas opções concretas.
- Estudar as possibilidades de implementação de interface das plataformas entre si e com programas de processamento em tempo real de código aberto, como Pure Data e CSound.

Para poder realizar os objetivos propostos, é interessante levantar uma lista de algoritmos de processamento de sinais que poderiam implementados. A seguir, relacionamos algumas possibilidades de algoritmos para serem implementados para o estudo, com a ressalva de que, neste momento, ainda se tratam apenas de exemplos. Dividimos a lista em três classes, de acordo com o custo computacional de cada algoritmo.

Processamentos leves:

- Filtros básicos de suavização (realce de graves) ou de diferenças (realce de agudos).
- Equalização em contexto mais geral usando polos e zeros como descritores de regiões de ressonância e anti-ressonância em filtros IIR de ordem baixa.
- Efeitos simples (overdrive, phaser, wah-wah) (ZöLZER et al., 2002);
- Detecção de características sonoras de baixo nível (low level descriptors), como frequência fundamental, energia rms, centróide espectral ou MFCC, ou ainda descritores psicoacústicos, como brilho, harmonicidade ou ruidosidade (PEETERS, 2004).

Processamentos de dificuldade média:

- Efeitos diversos (pitch shifting, flanger, companders, vibrato, chorus) (ZöLZER et al., 2002).

- Reverberação/espacialização genéricas (bancos de filtros-pente + realimentação) (ZöLZER et al., 2002).

Processamentos computacionalmente pesados:

- Morphing em tempo real (aplicação da envoltória espectral de um sinal a outro sinal) (ZöLZER et al., 2002).
- Phase Vocoder (com análise/ressíntese em tempo real) (ZöLZER et al., 2002).
- Auralização usando respostas impulsivas medidas ou simulação através de modelos geométricos (MOORE, 1990).

No capítulo 3, veremos com algumas possibilidades de exploração específicas para cada uma das plataformas.

Capítulo 2

Fundamentação teórica

Nas próximas seções, revisaremos algumas referências teóricas relevantes para os estudos propostos neste trabalho. Primeiro abordamos a investigação de limites inferiores para a realização de processamento de sinais digitais em tempo real, que é interessante para estabelecer um patamar de análise para o Arduino enquanto ferramenta dotada de baixo poder computacional. Em seguida, analisamos a evolução de técnicas paralelas para processamento digital em geral para fundamentar o estudo da plataforma GPGPU. Por fim, levantamos algumas possibilidades de uso de aparelhos Android para o processamento em tempo real e terminamos o capítulo com algumas considerações sobre a comparação das três plataformas.

2.1 Evolução e exemplos de processamento em tempo real

A evolução dos circuitos especializados em processamento de sinais digitais se deu, em geral, através da inclusão de características que facilitam a computação dos algoritmos de processamento de sinais digitais. Um exemplo é a inclusão do produto interno como operação básica, o que permite a computação de filtros de resposta impulsiva finita de forma muito mais rápida. Outros fatores fundamentais para a qualidade e velocidade dos circuitos especializados para processamento de sinais digitais como os que estão disponíveis hoje são a associação de múltiplas unidades de execução (como por exemplo a possibilidade da execução do cálculo do produto interno em paralelo com o processamento lógico e aritmético), o esforço para atingir eficiência no acesso à memória, o desenvolvimento de formatos de dados específicos para obter fidelidade numérica, a utilização de estágios sequenciais de processamento (*pipelines*) e o desenvolvimento de conjuntos de instruções especializados (EYRE; BIER, 2000).

Enquanto a produção de hardware segue linhas industriais, a produção de software para processamento de áudio em tempo real pode ser dividida em duas frentes: técnica e experimental. De um lado, podem ser encontrados programas de propósito geral que implementam modelos computacionais para processamento em tempo real, como CSound (VERCOE; ELLIS, 1990), Pure Data (PUCKETTE, 1996), e outros. Do outro lado, encontram-se as produções artísticas que expressam concepções estéticas específicas, explorando idéias de acompanhamento ao vivo e interatividade na performance, como por exemplo os sistemas Voyager (LEWIS, 2000) e Cypher (ROWE, 1992b).

2.2 Limites inferiores para o processamento de sinais digitais em tempo real

Ao lidar com processamento de sinais digitais em tempo real, uma questão fundamental que surge é sobre o poder computacional necessário para realizar com efetividade o processamento em questão, de forma a obter os resultados dentro de um prazo razoável. Para isto, é necessário quantificar a velocidade e o desempenho do hardware utilizado e a relação destes com as necessidades de cada tipo de processamento proposto.

Por meio da transformação da representação esquemática de filtros digitais em grafos dirigidos e levando-se em conta o tempo de processamento de cada tipo de operação que é realizada no filtro, é possível determinar o período mínimo (ou a taxa máxima) de amostragem associado a uma certa classe de filtros digitais (RENFORS; NEUVO, 1981). Neste sentido, podemos também seguir o caminho inverso e nos perguntar qual é o poder computacional mínimo para permitir a aplicação de um certo filtro a um sinal digital, se a taxa de amostragem é dada. Em outras palavras, se desejamos uma taxa de amostragem fixa, qual é o máximo de tempo que uma certa plataforma pode gastar em cada operação para que o cálculo seja realizado sem diminuir a taxa de geração de amostras? E, se a taxa de geração de amostras for prejudicada, quais são os prejuízos sonoros levando em conta os detalhes, por exemplo, do tipo de conversão digital-analógica utilizada? Esta abordagem nos permite determinar, para uma plataforma com uma capacidade computacional determinada, os tipos de processamento de sinais digitais que podem ser implementados para utilização em tempo real.

Além de somente estudar o custo computacional associado a certas classes de filtros digitais, também existem iniciativas no sentido de projetá-los tendo como objetivo obter filtros com baixo custo computacional. Podemos encontrar desde trabalhos que fazem uso de técnicas de inteligência artificial e teoria dos grafos para desenvolver filtros de resposta impulsiva finita de baixa complexidade (REDMILL; BULL, 1997), até trabalhos que comparam diferentes formas de implementação de paralelismo no cálculo de filtros avaliando a performance de cada uma (DEEPAK; MEHER; SLUZEK, 2007).

Trabalhos como os apresentados nesta seção podem ser utilizados para ajudar na avaliação da capacidade de processamento em tempo real das plataformas propostas.

2.3 Paralelismo no processamento de sinais digitais

Muitas técnicas têm sido utilizadas para utilizar paralelismo no processamento de sinais, sempre explorando características paralelas inerentes a alguns tipos de estruturas de dados, equações e algoritmos. Dentre as plataformas escolhidas para análise, a GPU é a que desperta maior interesse em relação ao estudo do paralelismo no processamento de sinais, pois trata-se essencialmente de um processador paralelo. Dispositivos móveis mais recentes, inclusive alguns capazes de suportar o sistema operacional Android, também possuem circuitos do tipo GPU, apesar de que com capacidade computacional reduzida devido à necessidade de economia de energia.

Nesta seção, descreveremos alguns estudos sobre paralelismo no processamento de sinais, com o objetivo de desenvolver um ferramental que nos ajudará a analisar as possibilidades que uma plataforma de processamento paralelo pode trazer para o processamento de sinais digitais em tempo real.

2.3.1 Paralelismo na transformada de Fourier

No início do século XIX, Joseph Fourier sugeriu que a solução para a equação do calor em um meio sólido poderia ser expressa como uma combinação linear de soluções harmônicas (FOURIER, 1807). Mais de vinte anos depois, foi provado que a mesma idéia vale para uma classe mais geral de sinais (DIRICHLET, 1829). Desses trabalhos surgiu a "Transformada de Fourier", uma operação matemática inversível que decompõe um sinal em suas componentes de frequência.

Até a metade do século XX, os conceitos desenvolvidos a partir da transformada de Fourier já haviam ganhado aplicação nas mais diversas áreas do conhecimento científico, como por exemplo: cálculo da periodicidade da orientação do spin de certos cristais, monitoramento sísmico remoto (no contexto de facilitar acordos sobre banimento de testes nucleares após a segunda guerra mundial), melhoramentos da capacidade de detecção acústica de submarinos a distância, filtros digitais, processamento de fala, música e imagens, análise espectral de dados de interferômetro para determinação do espectro infravermelho de planetas, estudos na área de oceanografia, entre muitas outras (COOLEY, 1987). Este cenário propiciou o desenvolvimento de muitas variantes do algoritmo original de cálculo dos coeficientes da transformada. O algoritmo ingênuo consome tempo proporcional a N^2 se N é o tamanho da entrada, e o amadurecimento desta linha de pesquisa deu origem à `FFT` (Fast Fourier Transform), a transformada "rápida" de Fourier, que diminui o consumo de tempo para algo proporcional a $N.\log(N)$ (COOLEY; TUKEY, 1965).

Mais de 10 anos depois foi proposta uma adaptação da `FFT` para processamento paralelo em uma máquina desenvolvida especialmente para este propósito, de forma que a operação seja dividida em níveis e cada nível possua uma quantidade pequena de operações elementares que possam ser realizadas ao mesmo tempo (PEASE, 1968). O resultado é um algoritmo que realiza três tipos de operações paralelas (a diferença entre pares de elementos adjacentes da entrada, uma permutação "ideal" das linhas de uma matriz, e habilidade de multiplicar todos os elementos da entrada por um mesmo fator) e cuja execução é da ordem de duas vezes mais rápida do que a da `FFT` tradicional.

A manipulação do espectro de um sinal digital pode ser feita utilizando-se a Transformada de Fourier para trabalhar diretamente no domínio da frequência, ou através do cálculo da convolução do sinal digital com a resposta impulsiva de um certo filtro no domínio do tempo. O estudo do cálculo da convolução levou a uma outra abordagem bastante comum no processamento digital de sinais, que é a divisão do sinal de entrada em blocos de tamanho fixo e o desenvolvimento de algoritmos para realização de cálculos em blocos (OPPENHEIM; SCHAFER; BUCK, 1999).

Durante a década de 60 e início dos anos 70, muita discussão foi feita em torno da eficiência e estabilidade de algoritmos de cálculo de convolução em blocos. Esta discussão culminou no desenvolvimento de técnicas para a aplicação de filtros recursivos utilizando operações em blocos, que permitem a execução em paralelo do processamento de cada bloco (BURRUS, 1972).

A seguir, veremos como o paralelismo se desenvolve no contexto do hardware específico para processamento de sinais digitais.

2.3.2 Circuitos digitais para processamento paralelo

Como vimos na seção 2.1, o desenvolvimento dos projetos de processadores de sinais digitais tem sido motivado pelas características dos algoritmos disponíveis para realizar o trabalho. Com a implementação de paralelismo nas arquiteturas ocorre o mesmo. Existem diversos tipos de paralelismo possíveis de serem explorados em um sistema como, por exemplo, paralelismo

de dados, de instruções ou de tarefas. A combinação do nível de exploração de cada tipo de paralelismo deve levar em conta um balanço entre rapidez de hardware, flexibilidade de software e domínio de aplicação. Essa abordagem tem dado origem a diversas implementações distintas de circuitos paralelos para processamento de sinais em tempo real (SERNEC; ZAJC; TASIC, 2000).

Como veremos mais adiante na seção 3.4.2, a estrutura do processamento de sinais é convenientemente capturada pelo modelo de processamento de fluxos de dados. Este modelo surge para satisfazer a necessidade de especificação formal do paralelismo inerente a soluções algorítmicas para problemas em diversos domínios, tais como mineração de dados, processamento de imagens, simulações físicas, e muitos outros (OWENS et al., 2007). O processamento gráfico não só se encaixa neste modelo mas também é influenciado por ele. O fato de que o desenvolvimento das arquiteturas tem sido fortemente influenciado pelo modelo de processamento de fluxos de dados pode ser observado pelas escolhas estratégicas recentes das fabricantes de placas de vídeo. As arquiteturas mais novas mostram uma tendência a unificar as unidades programáveis, em direção a um modelo menos específico do que somente para processamento de imagens tridimensionais. A idéia do processamento de fluxos de dados é complementada pelo desenvolvimento de arcabouços que implementam este modelo, como veremos na seção 3.4.3.

2.4 Conectividade, sensores e interatividade na performance

A realização de processamento de áudio em tempo real em plataformas móveis com capacidade de interconexão abre muitas possibilidades de colaboração para produção artística. A conectividade, aliada à abundância de tipos de sensores diferentes (acelerômetro, giroscópio, luz, campo magnético, orientação, pressão, proximidade, temperatura, entre outros) resulta numa ferramenta bastante rica para exploração e experimentação.

Assim, o processamento de áudio em dispositivos móveis pode ser compreendido neste contexto como uma ferramenta tecnológica para produção artística dotada de capacidade de interconexão com outros dispositivos e interação com o ambiente através de sensores e atuadores. A flexibilidade trazida por plataformas como o Android nas quais é possível desenvolver programas utilizando linguagens de alto nível aproxima os artistas em potencial das possibilidades trazidas pelo instrumento.

No contexto artístico, é interessante compreender conectividade e mobilidade não só por suas características desejáveis, como a possibilidade de comunicação instantânea entre múltiplos pontos dispersos no espaço, mas também através das limitações inerentes a estas características. Se uma volta em torno da terra à velocidade da luz demora cerca de 130 milissegundos, então a latência na comunicação planetária é inevitável. Nesse sentido, encontramos estudos que, ao desconstruir a idéia da conectividade como solução tecnológica universal para todos os problemas, sugere que a latência e outras características normalmente compreendidas como indesejáveis na comunicação podem ser aproveitadas como material estético (SHROEDER et al., 2007).

Capítulo 3

Metodologia

3.1 Análise das plataformas em estudos de caso

Em termos de capacidade computacional, é possível encontrar algumas métricas quantitativas para comparação das três plataformas, como por exemplo número de operações de ponto flutuante por unidade de tempo (intensidade computacional) e tempo de resposta (latência). Para isto, algumas idéias são o estudo de algoritmos de processamento de sinais (filtros, análise, síntese, etc) e a possibilidade de representação de fórmulas em função da capacidade computacional disponível (cálculo da ordem máxima de filtros em função do número de amostras disponíveis, tamanho máximo de blocos de amostras para análise janelada, etc).

Apesar disso, por causa das diferenças fundamentais entre as plataformas escolhidas para análise, é muito difícil (e talvez até irrelevante) estabelecer uma comparação puramente quantitativa entre as três. Em um extremo, o Arduino desperta o interesse também por suas limitações de velocidade e capacidade de memória, enquanto que a GPU tem como objetivo fundamental a aceleração da computação através da utilização de paralelismo, além de dispor de diversos níveis de memória. Já a mobilidade e comunicação com outros dispositivos, ainda que por motivos diferentes, são características fundamentais do Arduino e do Android, mas não da GPU.

Nesse sentido, outras métricas qualitativas parecem ser de igual relevância, considerando também os papéis possíveis de cada plataforma na criação e na performance artística. Algumas comparações possíveis são quanto à possibilidade de extensão das funcionalidades de cada plataforma, seu custo e as licenças de uso associadas, as possibilidades de integração e/ou comunicação remota destas três plataformas com ferramentas tradicionais para processamento de som em tempo real, tais como Pure Data ou MAX/MSP.

A possibilidade da interação com outras ferramentas de processamento em tempo real aumenta o alcance e o interesse do uso das plataformas escolhidas. A comunicação remota do Pure Data com o Android, por exemplo, possibilitaria o intercâmbio tanto de sinais de áudio quanto de controle, permitindo a variação em tempo real dos parâmetros de algoritmos de processamento. Note que isto poderia ocorrer nos dois sentidos, ou seja, é possível que o Pure Data controle parâmetros do processamento no Android ou o contrário.

3.2 Processamento de sinais digitais nas plataformas escolhidas

Como vimos no capítulo 1, um dos objetivos deste estudo é analisar possibilidades de implementação e eficiência de técnicas de processamento em tempo real nas plataformas descritas.

Para isto, é interessante encontrar trabalhos que motivem a exploração de pontos relevantes de cada uma das plataformas.

Pode-se definir uma metodologia de exploração da limitação computacional do Arduino a partir da implementação de técnicas canônicas de processamento de sinais digitais como aquelas descritas pela bibliografia básica consolidada (OPPENHEIM; SCHAFER; BUCK, 1999; ZöLZER et al., 2002), o que permitirá a descrição e análise da plataforma e seus limites.

Para a GPU a situação é um pouco diferente. A literatura sobre paralelismo no processamento de propósito geral utilizando GPU já é muito extensa e inclui otimizações de hardware e software nos diferentes estágios de desenvolvimento dos circuitos do tipo GPU. Diversas aplicações já foram descritas utilizando mapeamento de problemas para o domínio do processamento gráfico nestes diferentes estágios. Neste momento, parece que a pesquisa deve seguir o caminho de consolidar a computação de fluxos de dados como modelo de programação para *pipelines* de placas gráficas, estudar a influência dos diferentes tipos de hardware existentes neste tipo de modelagem mais abstrata, e propor formas de aproveitamento destes recursos utilizando programas para processamento de sinais em tempo real com licenças abertas amplamente utilizados. Neste sentido, o estudo e implementação de técnicas básicas de paralelismo para o processamento digital, como por exemplo a Transformada de Fourier Paralela (descrita na seção 2.3.1), os bancos de filtros baseados em subconvoluções paralelas (GRAY, 2003) e outros filtros simples utilizando diferentes modelos de paralelismo, como foi feito para uma plataforma diferente em Deepak, Meher e Sluzek (2007), são caminhos interessantes a serem seguidos.

Já na consideração do sistema Android, uma primeira abordagem seria avaliar as possibilidades de processamento em tempo real, ou seja, quais sinais são possíveis de captar e através de quais interfaces é possível emitir sinais processados (entrada de microfone, P2 de entrada e/ou saída, áudio da chamada telefônica, transmissão de dados via TCP/IP, bluetooth, etc). Também é interessante determinar como a utilização da linguagem Java, geralmente associada à necessidade de altos recursos computacionais, pode influenciar o processamento em tempo real.

Uma consideração importante é que a escolha exata de quais algoritmos serão implementados e discutidos depende ainda de uma investigação um pouco mais profunda sobre as tecnologias propostas. Na seção 1.4, vimos algumas possibilidades de algoritmos a serem implementados para estudar cada plataforma. Nas próximas seções, veremos detalhes do funcionamento de cada uma das plataformas e algumas idéias mais específicas de como explorar cada uma. No capítulo final (seção 4.2), veremos como pretendemos seguir com o estudo e as implementações.

3.3 Estudo de caso: Arduino

A plataforma desenvolvida pelo projeto Arduino consiste em um conjunto de hardware e software que, juntos, compõem uma interface simplificada para interação com um microcontrolador. Existem diversos projetos para placas de Arduino, mas todos possuem as mesmas estrutura e as mesmas funcionalidades básicas. O hardware genérico é composto por um processador com memória flash programável e suporte a entradas e saídas analógicas e digitais. O software, por sua vez, é composto por um compilador e um ambiente de desenvolvimento integrado (IDE), desenvolvidos em Java[1] com bibliotecas em C, e um sistema de inicialização que roda na placa. O projeto da placa possui o mínimo necessário para alimentação e comunicação com o microcontrolador: reguladores, relógio de cristal, interface USB ou serial para conexão com um computador e uma interface de programação para substituir o sistema de inicialização.

Nas próximas seções, descreveremos com um pouco mais de detalhes cada um dos compo-

[1] http://www.oracle.com/technetwork/java/index.html

nentes básicos de hardware e software do Arduino, para em seguida avaliar suas possibilidades e limitações de uso e estabelecer parâmetros de análise.

3.3.1 Hardware básico e conexão com o computador

Um **microcontrolador** é um conjunto minimal de componentes para a execução de uma aplicação específica: processador, memória para armazenamento do programa (em geral flash), memória de acesso aleatório para armazenamento dos dados e interfaces de entrada e saída de dados. A diferença fundamental entre microcontroladores e microprocessadores (como, por exemplo, as unidades centrais de processamento dos computadores de mesa e notebooks) é que os microcontroladores possuem toda a estrutura necessária para a computação em um único chip, enquanto que microprocessadores utilizam memória de acesso aleatório externa ao chip para armazenamento de programa e dados. Além disso, microcontroladores possuem custo de produção mais baixo e menor consumo de energia pois, em geral, são menos flexíveis (em termos de aplicações), dispõem de menor capacidade computacional, além de serem desenvolvidos com tecnologias específicas, diferentes das do microprocessador[2].

A função básica do hardware do Arduino é disponibilizar uma interface mínima para a comunicação de um computador com um microcontrolador, e deste microcontrolador com outros módulos de hardware, sensores ou atuadores. Existem vários projetos diferentes de placas Arduino, que diferem principalmente no modelo de microcontrolador utilizado, interface de comunicação com o computador e disposição dos componentes na placa.

A escolha de uma marca e modelo de microcontrolador para um projeto deve levar em conta diversos fatores, como por exemplo a possibilidade de reprogramação (e a infraestrutura necessária para realizar este procedimento), as possibilidades de conexão com periféricos (USB, rede, módulos de PWM, de memória externa, etc), voltagem de alimentação suficiente para controlar diretamente LEDs e outros periféricos, apresentação (em termos de necessidades relativas a manipulação, transporte, proteção, etc) e limites de memória para o programa e para os dados (muitas famílias de microcontroladores possuem limites muito pequenos, veja a tabela 3.1 com os limites dos modelos do Arduino).

O objetivo do projeto Arduino é, desde seu início, criar uma plataforma para desenvolvimento de projetos com microcontroladores que seja acessível (financeira e tecnologicamente) para estudantes, artistas, hobbystas e curiosos em geral. Esta ambição cria algumas necessidades em termos de funcionalidade da plataforma, o que por consequência gera restrições para a estrutura do microcontrolador a ser utilizado pelo projeto. Necessitava-se de um microcontrolador que fosse facilmente reprogramável a partir de uma interface disponível em qualquer computador pessoal e possuísse um bom balanço entre flexibilidade e padronização na interface com periféricos.

O projeto Arduino escolheu os microcontroladores da marca Atmel[3] da série megaAVR, mais especificamente os modelos ATmega168, ATmega328, ATmega1280 e ATmega2560. Os modelos diferem um do outros na frequência de operação, capacidade de memória (tanto para programa quanto para dados), e número de pinos de entrada e saída digitais e analógicas. Uma comparação entre as características de Arduinos que utilizam cada modelo de microcontrolador pode ser vista na tabela 3.1. Um mesmo modelo de microcontrolador é utilizado por vários modelos diferentes de Arduino e a tabela mostra o valor máximo de cada característica encontrada entre os modelos que utilizam um mesmo microcontrolador.

A atualização do programa nos microcontroladores do Arduino é feita através da interface

[2] http://www.engineersgarage.com/microcontroller
[3] http://www2.atmel.com/

Tabela 3.1: Características dos processadores utilizados em diferentes modelos do Arduino

	ATmega168	ATmega328	ATmega1280	ATmega2560
Frequência de operação	16 MHz	16 MHz	16 MHz	16 MHz
Memória para programa	16 KB	32 KB	128 KB	256 KB
Memória para dados	1 KB	2 KB	8 KB	8 KB
Entradas/saídas digitais	14	14	54	54
Saídas com PWM	6	6	14	14
Entradas analógicas	8	8	16	16

de desenvolvimento, que é executada em qualquer computador capaz de rodar aplicações Java e se comunica com a placa através de algum tipo de conexão padrão. Os primeiros projetos de Arduino possuíam interface serial com o computador, mas com o declínio da produção de placas-mãe com esta interface e a popularização do USB, os modelos mais novos do Arduino possuem interface USB. Também estão disponíveis projetos de módulos USB para a adaptação deste tipo de interface a modelos antigos ou para a inclusão de mais uma porta de comunicação nos modelos mais novos.

O projeto Arduino não teria tanto alcance se não incluísse uma interface de desenvolvimento que simplifica a escrita, compilação e carga do programa no microcontrolador. É possível obter, a partir do sítio oficial do projeto[4], o código fonte e versões compiladas para Windows, Mac OS X e Linux do software que unifica o processo de desenvolvimento para o Arduino em torno de apenas uma ferramenta. Com uma placa de Arduino conectada ao computador via USB e um ambiente Java configurado corretamente, é necessário apenas um clique para compilar o código e transferi-lo para a área do programa no microcontrolador. Através da abstração dos detalhes mais árduos da interação com o microcontrolador obtida pela utilização do mesmo software para programação, compilação e transferência do programa para a placa, o projeto atinge um grande nível de usabilidade. A plataforma se utiliza de um *bootloader* (um gerenciador de carregamento do sistema) gravado nas primeiras centenas de bytes do chip que permite que a substituição do programa seja feita sem a necessidade de hardware específico[5].

3.3.2 Extensões do Arduino

Um Arduino pode ser estendido através de placas que podem ser acopladas à placa principal. Estes módulos levam o nome de *shields*[6] (escudos) e compreendem tanto a integração de diversas tecnologias padrão como Ethernet, Xbee, cartões SD, sensores de temperatura, WiFi e GSM, como outras funcionalidades úteis como controle de motores[7], reprodução de MP3[8], ou mesmo um único escudo que disponibiliza acelerômetro, alto falante, microfone, transmissor e receptor infravermelho, LED RGB, botões, potenciômetro e sensor de luz visível[9]. O sítio do Arduino aponta uma listagem extensa de escudos oficiais e não oficiais compatíveis com a plataforma[10].

Existem também muitas placas derivadas do Arduino, compatíveis apenas com o software. São clones genéricos, placas e interfaces para os mais diversos fins, desde implementações de pilotos automáticos para aeromodelos e tanques até versões que ocupam menos espaço ou podem

[4] http://arduino.cc/en/Main/Software
[5] http://arduino.cc/en/Tutorial/Bootloader
[6] http://arduino.cc/en/Main/ArduinoShields
[7] http://www.robotpower.com/products/MegaMoto_info.html
[8] http://www.roguerobotics.com/products/electronics/rmp3
[9] http://www.ruggedcircuits.com/html/gadget_shield.html
[10] http://shieldlist.org/

ser montadas em papel[11].

Deve-se observar que existe um balanço entre a técnica e infraestrutura disponível para a construção de uma placa, um escudo ou outros tipos de extensões e adaptações do Arduino e a qualidade do resultado do processamento, principalmente quando da captura e síntese de áudio. Variações de temperatura no ambiente ou ressonância com ondas de radiofrequência, por exemplo, podem interferir no resultado da conversão de sinais entre representações analógicas e digitais em montagens caseiras que não possuam cuidados especiais.

3.3.3 Entradas e saídas analógicas, frequência de operação e taxa de amostragem

Os microcontroladores da Atmel possuem algumas portas de entrada com conversores analógico-digitais com resolução de 10 bits (veja a tabela 3.1) que mapeiam voltagens entre dois valores de referência para inteiros entre 0 e 1023. A maioria dos modelos de microcontrolador do Arduino operam a uma taxa de 16 MHz, mas como a leitura da entrada analógica utilizando a função da biblioteca demora cerca de 100 microssegundos, a taxa de amostragem de uma porta analógica atinge cerca de 10.000 Hz[12]. Acessando-se o hardware diretamente é possível obter taxas de amostragem um pouco maiores, como veremos abaixo. Apesar disto, o valor não é muito mais alto e esta restrição representa não só um limite do espectro de frequências representadas após a digitalização mas também estabelece a necessidade de utilização de um filtro externo no sinal de entrada para cortar frequências acima da taxa de Nyquist para que não haja aliasing.

Com esta limitação na taxa de amostragem de um sinal de entrada, uma opção interessante para o processamento de áudio em tempo real é a reconfiguração dos relógios internos para diminuir a taxa de chamada da função de processamento e assim permitir um período maior de processamento entre a chegada de duas amostras consecutivas. Um projeto que implementa processamento de áudio em tempo real[13], por exemplo, realiza amostragem alternadamente em duas portas diferentes, uma com entrada de um sinal de áudio, e outra conectada a sinal de controle. Para obter um tempo razoável de processamento, os relógios internos são configurados de forma que a amostragem de cada sinal (de áudio e de controle) é feito com uma taxa efetiva de 15 KHz. Supondo que os sinais de controle não precisam ser amostrados com taxa igual ao sinal de áudio e que a computação pode ser feita em blocos, talvez seja possível melhorar estes resultados.

Também é possível gerar saídas analógicas com resolução de 8 bits através de circuitos PWM embutidos no processador. O sinal gerado por esta técnica pode ser utilizado para controle do nível de brilho de LEDs, controle de velocidade de motores e, com um pouco de boa vontade, gerar sons. A frequência máxima obtida de um sinal PWM no Arduino utilizando-se a função de escrita analógica da biblioteca é 500 Hz[14], bastante baixa em relação ao espectro de frequências audíveis. Utilizando-se de algumas funções específicas do microcontrolador e fazendo a escrita direto na porta de saída, é possível obter frequências de até 1.3 KHz[15].

Uma outra possibilidade de uso do PWM é gerar sinais com forma de ondas fixas. Realizando um controle fino da geração do sinal PWM, um experimento conseguiu gerar frequências de até 16 KHz , e obter uma relação sinal/ruído de por volta de 50 dB para as frequências mais baixas (até 3KHz), o que corresponde ao esperado para um conversor digital-analógico com resolução de 8 bits ($20 \times \log_{10}(2^8) \approx 48$ dB)[16].

[11] http://www.arduino.cc/playground/Main/SimilarBoards
[12] http://arduino.cc/en/Reference/AnalogRead
[13] http://interface.khm.de/index.php/lab/experiments/arduino-realtime-audio-processing/
[14] http://www.arduino.cc/en/Reference/AnalogWrite
[15] http://www.arcfn.com/2009/07/secrets-of-arduino-pwm.html
[16] http://interface.khm.de/index.php/lab/experiments/arduino-dds-sinewave-generator/

Apesar destas limitações, existem escudos do Arduino tanto para gravar[17] quanto para reproduzir som com maior resolução e frequência (12 bits e 22 KHz, respectivamente, neste exemplo[18]). Assim, se a baixa fidelidade não é uma escolha estética, sempre há a possibilidade de extensão da plataforma utilizando hardware específico de forma a atender as necessidades do trabalho de cada artista.

3.3.4 Estrutura de um programa e bibliotecas

Um programa que roda no microcontrolador de um Arduino deve ter ao menos duas funções definidas: `setup()` e `loop()`. A função `setup()` é chamada no início da execução do programa (após a alimentação com energia ou um reinício do sistema), e é usada para inicializar variáveis, configurar os modos dos pinos, inicializar bibliotecas, alterar os relógios, etc. A função `loop()` é então executada e é chamada novamente a cada interrupção gerada pelos relógios internos.

Uma série de bibliotecas estão disponíveis para possibilitar a interface com dispositivos de entrada e saída como interface *serial*, *Ethernet*, visores de cristal líquido, matrizes de LEDs, entre outros[19]. Também é possível encontrar bibliotecas para tarefas comuns como controle dos relógios, troca de mensagens com o computador, entre Arduinos ou com dispositivos de outras naturezas, e até mesmo para tarefas mais avançadas como servidores HTTP.

3.3.5 Propostas de análise

Aplicações de processamento de sinais em geral exibem alta intensidade computacional, ainda mais quando se trata de processamento em tempo real. Por causa de suas limitações de poder de processamento e flexibilidade de interface com outros dispositivos, o Arduino tem sido utilizado principalmente para computações de controle, para as quais as limitações do hardware não representam um empecilho.

Neste sentido, com o objetivo de compreender e quantificar as limitações da plataforma e por que essas limitações se aplicam à utilidade pretendida, propomos o seguinte:

- Verificar a viabilidade de realização de cálculos em blocos, acumulando amostras em um vetor temporário. Assim, pode-se dispor de mais tempo de processador sem interrupções de hardware para o processamento do sinal, além de permitir análise janelada do sinal de entrada, o que pode fornecer informações sobre como o espectro do sinal evolui com o tempo.
- Determinar o período mínimo de amostragem e assim a frequência máxima capturada pela amostragem do sinal de entrada, bem como os filtros necessários antes da entrada do sinal no aparelho para que não haja aliasing.
- Determinar a intensidade computacional máxima que não influencia a taxa de amostragem.
- Quantificar limitações no processamento adaptativo, ou seja, na frequência de atualização dos parâmetros de controle dos filtros utilizados. Essa questão não é muito relevante se cada amostra é calculada independentemente a partir de uma descrição explícita do filtro, mas pode ser importante no processamento em blocos, ou na utilização de estruturas de dados auxiliares para aumentar a eficiência do cálculo.

[17] http://shieldlist.org/seeedstudio/music
[18] http://www.ladyada.net/make/waveshield/faq.html
[19] http://arduino.cc/en/Reference/Libraries

Com isto, demos uma visão geral das características do Arduino e levantamos algumas possibilidades de exploração da plataforma. Na próxima seção, acompanharemos o desenvolvimento da GPU e veremos como o paralelismo pode ser utilizado para o processamento de sinais em tempo real.

3.4 Estudo de caso: GPGPU

Nesta seção, faremos uma breve descrição da arquitetura atual dos circuitos do tipo GPU e dos arcabouços de programação disponíveis, evidenciando formas diferentes de interagir com a plataforma. Em seguida, introduziremos o ferramental teórico do processamento de fluxos de dados que fundamenta a programação de propósito geral utilizando GPU, para então propor formas de avaliação das possibilidades de processamento de sinais digitais em tempo real.

Como vimos na seção 1.2.2, a evolução dos circuitos do tipo GPU ao longo de mais de 10 anos levou a utilização dos circuitos gráficos para a computação de propósito geral de uma situação inicial na qual as possibilidades eram bastante restritas para uma situação atual de maior flexibilidade. No início, as funções fixas para propósitos específicos em diversos estágios da *pipeline* forçavam a necessidade de adaptação da solução de um problema a uma estrutura bastante engessada. A utilização de hardware especializado para implementação de funções fixas específicas para o processamento de imagens tridimensionais é complementada com a implementação de paralelismo de dados e de tarefas. Assim, várias tarefas podem ser executadas ao mesmo tempo, cada uma realizando uma mesma operação em paralelo em todo um conjunto de dados. Isto resulta em uma arquitetura que consegue atingir alta intensidade computacional e alta taxa de fluxo de dados.

Com o amadurecimento do campo de pesquisa, as técnicas se tornaram mais sofisticadas e as comparações com os trabalhos fora do campo da GPU mais rigorosas. No nível do software, esta maturidade é evidenciada pela a construção de aplicações reais nas quais a GPU demonstra possuir grandes vantagens. No nível do hardware, pudemos observar a transformação da GPU em um processador paralelo totalmente programável com funções fixas adicionais para propósitos especiais, como por exemplo alguns tipos de transformação linear e funções trigonométricas. Os conjuntos de funcionalidades e de instruções dos processadores de vértices e de fragmentos têm aumentado e convergido. Por causa disso, a *pipeline* baseada em tarefas paralelas tem tido uma tendência a ser remodelada para uma estrutura baseada em uma única unidade programável unificada, de forma que é possível atingir níveis mais complexos de paralelismo de tarefas e de dados. Isso faz com que, cada vez mais, os programadores possam se concentrar em apenas uma unidade programável contando ainda com técnicas básicas de computação paralela como *map*, *reduce*, *scatter*, entre outras (OWENS et al., 2007; OWENS et al., 2008).

Mesmo com esses avanços, programar para a GPU não se trata somente de aprender uma nova linguagem, mas de utilizar um modelo de computação diferente do tradicionalmente utilizado na programação para CPU, de forma a se adaptar à arquitetura utilizada. Para entender os modelos de computação, é necessário antes ter uma idéia de como funciona a *pipeline* de uma GPU.

3.4.1 Processamento de fluxos na pipeline

O processamento de gráficos tridimensionais requer grande capacidade computacional paralela e alta taxa de fluxo de dados para exibição dos resultados da computação em tempo real. Para atender a esta demanda, a *pipeline* tradicional para processamento de gráficos possui uma série de estágios de computação nos quais funções fixas são aplicadas aos dados de entrada (tipicamente

conjuntos de triângulos e conjuntos de texturas) e no fim são geradas imagens para exibição na tela. Os estágios de computação da *pipeline* tradicional são:

- **Processamento de vértices:** Os vértices dados como entrada para a GPU são mapeados para uma posição na tela, de acordo com interações com as fontes de luz da cena.
- **Construção de primitivas:** A partir dos vértices, são construídos triângulos, as primitivas fundamentais suportadas pelo hardware das GPUs.
- **Rasterização:** É o processo de mapeamento dos triângulos para pixeis. Cada triângulo gera uma primitiva chamada **fragmento** para cada pixel que cobre na tela. A cor de cada pixel pode ser computada a partir de vários fragmentos.
- **Processamento de fragmentos:** Cada fragmento gerado é processado utilizando texturas armazenadas na memória para determinar sua cor final.
- **Composição:** Os fragmentos são combinados para determinar a cor final de cada pixel. Em geral, mantém-se a cor do fragmento mais próximo da tela.

Os estágios de processamento de vértices e de fragmentos são altamente paralelizáveis pois a computação do pixel da tela associado a um vértice não depende de outros pixeis, assim como a determinação da cor final de um fragmento não depende de outros fragmentos. Apesar disto, nas primeiras gerações de GPU, as operações disponíveis nesses estágios não eram programáveis, mas apenas configuráveis. Era possível, por exemplo, escolher a posição e cor dos vértices e fontes de luz, mas não o modelo de iluminação que determinava sua interação. O passo chave para a generalização do uso da GPU está na idéia da substituição das funções fixas nesses estágios por programas especificados pelo usuário para serem aplicados em cada vértice e fragmento. Enquanto as primeiras gerações de GPU podiam ser descritas como uma pipeline de funções fixas com adição de elementos programáveis, as novas gerações são melhores caracterizadas como *pipeline*s programáveis com suporte de unidades de funções fixas (OWENS et al., 2008).

No contexto da *pipeline* gráfica, o conjunto de vértices de entrada e a memória de texturas podem ser utilizados como fontes de dados e cada estágio de processamento altera esses dados. Por causa do paralelismo nas tarefas (veja seção 1.2.2), a performance da *pipeline* depende sempre da tarefa mais lenta. O resultado de uma passagem completa pela *pipeline* pode ser armazenado na memória de texturas, que pode então ser utilizado durante o processamento dos dados de entrada na próxima passagem. Por causa da natureza das operações da *pipeline* gráfica, foi possível atingir uma flexibilização no processamento de fragmentos muito mais alta do que no processamento de vértices. Apesar disso, a tendência observada é de convergência dos conjuntos de instruções e funcionalidades e unificação da unidade programável, como descrito na seção anterior.

O desenvolvimento deste tipo de arquitetura de *pipeline* para processamento gráfico e o aumento da flexibilidade das unidades programáveis motivou o desenvolvimento de um modelo de programação que tem como objetivo generalizar a expressão do paralelismo e das estruturas de comunicação presentes em estruturas computacionais similares à do processamento gráfico, como veremos na próxima seção.

3.4.2 Processamento de fluxos de dados (*stream processing*)

Num primeiro momento, o modelo de programação de propósito geral utilizando GPU era uma subversão do modelo de computação gráfica para GPU. O programador definia uma primitiva geométrica que cobrisse um domínio de computação de interesse e utilizava a estrutura da

pipeline para transformar os dados, tendo muitas vezes que se utilizar de truques para realizar operações que não eram suportadas ou cujo consumo de tempo não era otimizado. Com os avanços das arquiteturas e dos modelos de computação paralela, hoje o programador define o domínio de interesse da computação como uma grade estruturada de *threads* e realiza operações matemáticas e acesso aleatório de leitura e escrita em uma memória global com bastante flexibilidade (OWENS et al., 2008).

Como vimos na seção 3.4.1, a *pipeline* gráfica é tradicionalmente estruturada como uma sequência de estágios de computação conectados por um fluxo de dados que atravessa todos os estágios. Esta estrutura, consequente das propriedades matemáticas do processamento tridimensional e das possibilidades e limitações da construção de processadores e memória, é capturada por um modelo computacional chamado **processamento de fluxos de dados** (*stream processing*), que captura a localidade do processamento e expõe o paralelismo e alguns padrões de comunicação de uma aplicação (KAPASI et al., 2003).

No modelo de processamento de fluxos de dados, todos os dados são representados através de **fluxos** (*streams*), definidos como conjuntos ordenado de dados do mesmo tipo. A computação nos fluxos de dados é feita através de **núcleos** (*kernels*), funções que operam em um ou mais fluxos de entrada e que possuem um ou mais fluxos de dados como saída. A saída de um núcleo deve ser uma função somente de sua entrada. Além disso, dentro de um núcleo a computação sobre um elemento do fluxo não deve depender da computação de outros elementos. Este modelo permite que a computação de um núcleo sobre vários elementos de um fluxo de dados seja realizada em paralelo e que vários núcleos sejam concatenados formando uma cadeia de composição de funções sobre um ou mais fluxos (OWENS, 2005).

A estrutura da *pipeline* gráfica descrita na seção 3.4.1 pode ser vista como uma restrição do modelo de fluxo de dados e núcleos. Sob o modelo de processamento de fluxos de dados, a implementação de uma *pipeline* gráfica deste tipo envolveria simplesmente a escrita de um núcleo para cada estágio de processamento da *pipeline* e a conexão da saída de um núcleo com a entrada de outro, na ordem apresentada.

A eficiência do modelo de processamento de fluxos de dados está não só na possibilidade de um núcleo processar diversos elementos do fluxo em paralelo, mas também no fato de que diversos núcleos podem ser calculados em paralelo permitindo a implementação de paralelismo de tarefas. Além disso, um bom balanço entre núcleos com funções fixas (que dependem do domínio do problema e podem ser implementadas em hardware especializado) e núcleos completamente programáveis pode aumentar ainda mais a eficiência da computação.

O estudo do processamento de fluxos de dados é interessante ainda pois representa um modelo alternativo de computação que pode levar em conta as diferenças da velocidade de avanço das tecnologias de processadores e de memórias. Enquanto a eficiência de um processador pode ser medida em termos de capacidade computacional (operações lógicas ou aritméticas por unidade de tempo), a eficiência das memórias de acesso aleatório é medida em termos de banda (quantidade de dados transferida por unidade de tempo) e latência (tempo de percurso de um bloco de dados desde a origem até o destino). Enquanto a capacidade computacional dos processadores aumenta cerca de 71% a cada ano, a banda de transferência das memórias de acesso aleatório cresce somente 25%, e a latência diminui apenas 5%. As questões (1) computação *versus* comunicação, (2) latência *versus* banda e (3) consumo de energia são hoje questões centrais para o desenvolvimento de processadores (OWENS, 2005).

Estudos sobre a eficiência e complexidade de modelos de processamento de fluxos podem ser encontrados desde a década de 1980, quando do aparecimento das primeiras placas de vídeo com aceleração para o processamento de gráficos. Já naquela época foi mostrado, por exemplo, que ordenação é um problema difícil neste modelo computacional. Também foram estabelecidos li-

mites inferiores e superiores para diversos outros problemas além de ser dada uma caracterização do poder computacional em função do número de registros e operações disponíveis (FOURNIER; FUSSELL, 1988). Mais recentemente, foi mostrado também que o número de "passagens" para a computação da mediana em um modelo de processamento de fluxos de dados é proporcional a $O(\log N)$, em oposição à complexidade de $O(N)$ no modelo tradicional, se N é o tamanho da entrada (GUHA et al., 2003).

É interessante notar que a *pipeline* gráfica tradicional dos anos 90, que veio como solução para as necessidades do processamento de imagens tridimensionais (alta intensidade computacional, alto grau de paralelismo e alta taxa de fluxo de dados), acabou motivando esta abordagem mais abstrata através do modelo de processamento em fluxos de dados, que por sua vez voltou a influenciar o projeto e a produção das GPUs em direção à computação de propósito geral. O balanço entre o uso de funções fixas ou núcleos completamente programáveis, que também pode ser vista como um balanço entre o número de transistores utilizados para controle da computação ou para o processamento propriamente dito, continua sendo uma questão central no desenvolvimento da GPU. De qualquer forma, a abstração e eficiência trazidas pelo modelo de processamento de fluxos de dados tem permitido a adaptação de diversos problemas para o domínio de aplicação da GPU.

3.4.3 Arcabouços para o processamento de fluxos de dados

Junto com a flexibilização do hardware da GPU e a possibilidade de programação de algumas unidades de processamento da *pipeline*, as empresas fabricantes de placas de vídeo tornaram disponíveis as primeiras linguagens de programação de alto nível específicas para estas plataformas. Até hoje, a maioria das linguagens disponíveis para GPU partem do pressuposto que a infraestrutura da placa é utilizada para a geração de imagens, e portanto provém meios de compilar programas para o processamento de vértices e fragmentos, que são carregados nos respectivos estágios da *pipeline* para gerar a imagem descrita pelo programa (OWENS et al., 2007).

Essa especifidade de domínio de aplicação divide as iniciativas de desenvolvimento de linguagens de programação e ambientes de depuração para GPU em dois grupos distintos. O primeiro grupo é composto de linguagens como as descritas acima, que abstraem as capacidades da GPU permitindo a utilização de um ambiente familiar para programação, porém continuam centradas em torno da idéia de que a GPU é utilizada para geração de imagens. Por esta razão, estas linguagens são chamadas de *Shading Languages* (em referência ao processo de obtenção de níveis de luminosidade para representação de imagens tridimensionais em superfícies bidimensionais). Alguns exemplos são Cg[20], HLSL[21], OpenGL Shading Language[22], Sh[23] e Ashli[24]. Muitas vezes o modelo adotado por estas linguagens torna a programação para GPU demasiado complicada, pois força o desenvolvedor a pensar em termos de primitivas geométricas, texturas e fragmentos, quando muitas vezes seu domínio de aplicação pode ter características distintas.

Num outro extremo, encontramos linguagens de programação de alto nível desenvolvidas com o objetivo de prover funcionalidades de programação de propósito geral e ao mesmo tempo esconder dos usuários os detalhes específicos da arquitetura da GPU. Estas linguagens se baseiam em conceitos do processamento de fluxos de dados na estruturação de seus modelos computacionais. São exemplos as linguagens Brook (BUCK et al., 2004), Scout, Microsoft Accelerator, CGiS e Glift (OWENS et al., 2007; OWENS et al., 2008).

[20] http://http.developer.nvidia.com/Cg/Cg_language.html
[21] http://msdn.microsoft.com/en-us/library/bb509635%28v=VS.85%29.aspx
[22] http://www.opengl.org/documentation/glsl/
[23] http://libsh.org/
[24] http://developer.amd.com/archive/gpu/ashli/pages/default.aspx

Uma terceira opção é a possibilidade de terceirização de processamento para placas gráficas em ambientes de programação já existentes. A linguagem CUDA[25] da NVIDIA[26], por exemplo, é uma extensão de C que permite determinar quais trechos da computação serão realizados na CPU e quais trechos serão delegados à GPU. Outros softwares muito utilizados para modelagem e realização de cálculos como Matlab[27] e Octave[28] também possuem a possibilidade de integração com GPU[29,30,31,32].

3.4.4 Propostas de análise

Com todo o arcabouço descrito, quais são as possibilidades de processamento de sinais em tempo real que a GPU nos trás? Quais são as limitações? Para analisar as possibilidades de uso da arquitetura GPU e da programação de propósito geral utilizando GPU, propomos as seguintes abordagens:

- Descrever as arquiteturas disponíveis e tendências de evolução da GPU: número de instruções simultâneas, instruções especiais (cálculos vetoriais, funções trigonométricas, etc), quantidade e funcionamento das unidades programáveis e tendências de evolução da *pipeline*.
- Determinar a intensidade computacional máxima como função da quantidade, tamanho e velocidade das unidades programáveis. Relacionar a arquitetura de processamento com a modelagem como processamento de fluxos de dados.
- Estudar a modelagem de filtros como processamento de fluxos de dados e utilizar arcabouços de processamento de fluxos para a implementação de algoritmos comuns de processamento de sinais digitais em tempo real.
- Estudar as possibilidades de implementação de interface com ferramentas de processamento de sinais com licença livre como, por exemplo, Pure Data e CSound.

Agora que já descrevemos o cenário da programação de propósito geral utilizando GPU, passaremos para a próxima plataforma de interesse. Na seção seguinte, veremos como se organiza o sistema Android e como podemos utilizá-lo para o processamento de sinais em tempo real.

3.5 Estudo de caso: Android

Iniciaremos esta seção com uma descrição de como está organizado o sistema operacional Android e como esta organização permite que o sistema esteja disponível para diversos modelos diferentes de aparelhos móveis, com arquiteturas e características distintas. Em seguida, desenvolveremos uma metodologia de análise da plataforma como um todo para finalmente prosseguir com propostas para análise e obtenção de resultados sobre este estudo de caso.

[25] *Compute Unified Device Architecture*
[26] http://developer.nvidia.com/what-cuda
[27] http://www.mathworks.com/products/matlab/
[28] http://www.gnu.org/software/octave/
[29] http://gp-you.org/
[30] http://www.accelereyes.com/
[31] http://billbrouwer.wordpress.com/2009/09/30/cuda-octave/
[32] http://scinesur.wordpress.com/2009/11/12/cuda-on-octave-3-x-in-linux-64-bits/

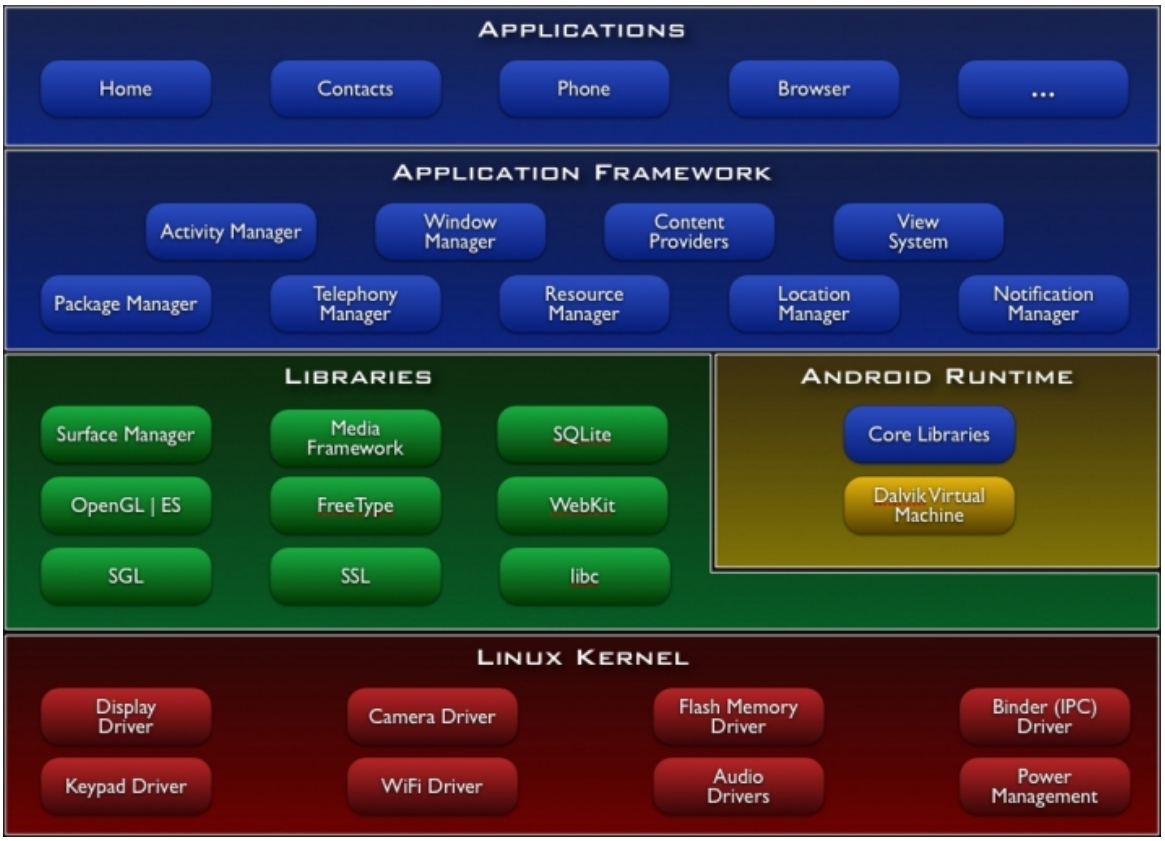

Figura 3.1: A organização do sistema operacional Android: uma pilha de software.

3.5.1 Organização em camadas

O sistema operacional Android é organizado em quatro camadas, como pode ser visto na figura 3.1: aplicações (*applications*); arcabouço para aplicações (*application framework*); bibliotecas e tempo de execução (*libraries and Android runtime*); e, finalmente, o kernel do Linux (*Linux kernel*).

Na camada mais próxima do usuário estão as aplicações (envio de mensagens, calendário, navegador, contatos, etc), escritas em Java e desenvolvidas de acordo com algumas convenções que permitem o intercâmbio de funcionalidades entre aplicações distintas. As aplicações são desenvolvidas utilizando um arcabouço de desenvolvimento em Java, conceitualmente posicionado numa camada imediatamente abaixo da camada de aplicações, composto por um conjunto de classes que provê interface com as funcionalidades do aparelho e com outras aplicações. As bibliotecas para desenvolvimento estão disponíveis para descarga na internet[33], bem como sua documentação[34]. Um desenvolvedor de aplicações para Android precisa, portanto, conhecer as convenções e o arcabouço de desenvolvimento disponíveis.

De acordo com as convenções de desenvolvimento para o Android, toda aplicação deve definir quais recursos do sistema está preparada para utilizar, como por exemplo entrada e/ou saída de som, internet, bluetooth, etc. No momento da instalação de uma certa aplicação no sistema, o usuário do aparelho é inquirido sobre a concessão de permissão para cada um dos recursos aos quais a aplicação dá suporte. O sistema Android se utiliza, então, da infraestrutura de processos e usuários fornecida pelo sistema Linux para garantir que cada aplicação seja

[33] http://developer.android.com/sdk/index.html
[34] http://developer.android.com/reference/packages.html

executada dentro de um contexto limitado, de forma que um eventual mal funcionamento da aplicação não comprometa nada além dela mesma. Cada aplicação no sistema é, portanto, executada dentro de um processo diferente, lançado por um usuário (de sistema) específico, criado para aquela aplicação no momento da instalação, com permissões para acessar apenas os recursos que o usuário (do aparelho) escolheu. Desta forma, a aplicação possui acesso a uma porção de memória limitada e não possui mais permissões do que o estritamente necessário para seu funcionamento (ou até eventualmente menos, se o usuário do aparelho assim o desejar).

Além de definir quais recursos do sistema está preparada para utilizar, uma aplicação também pode definir quais recursos disponibiliza para o sistema, de forma que outras aplicações possam utilizar esses recursos. Por exemplo, uma aplicação de processamento digital de sinais pode fornecer filtros de áudio e vídeo que podem ser utilizados por outras aplicações no momento em que forem processar conteúdo multimídia. Esta reutilização de recursos é gerenciada pelo mesmo arcabouço de desenvolvimento que descrevemos (representado pela segunda camada, de cima para baixo, na figura 3.1).

Descendo mais uma camada no modelo de organização do sistema Android, encontra-se um conjunto de bibliotecas que são utilizadas por diversos componentes do sistema e que também estão disponíveis para o desenvolvedor de aplicações através do arcabouço para desenvolvimento de aplicações da camada imediatamente acima. Estão incluídas bibliotecas de sistema, gravação e reprodução de mídia, processamento de imagens em duas e três dimensões, suporte a alguns tipos de banco de dados, entre outras funcionalidades.

Numa mesma camada conceitual, bibliotecas operam junto com código "em tempo de execução". Como as aplicações são desenvolvidas em Java, o código gerado não é específico para uma arquitetura e necessita de uma máquina virtual para ser executado. O sistema Android utiliza uma máquina virtual própria[35], otimizada para execução em aparelhos móveis, de forma que múltiplas instâncias podem rodar ao mesmo tempo de forma eficiente. Como descrevemos acima, cada aplicação é executada dentro de um processo próprio. Cada processo executa uma instância diferente desta máquina virtual, que depende da infraestrutura do kernel do Linux para gerenciamento de memória de baixo nível e criação de *threads* de execução.

Finalmente, no último nível se encontra o kernel do Linux, que funciona como uma camada de abstração entre o hardware e as outras camadas de software, provendo serviços para os outros níveis tais como gerenciamento de memória e de processos, segurança, rede e drivers.

Todas as aplicações e também o arcabouço de desenvolvimento para as aplicações são escritos em Java. As bibliotecas de sistema e a máquina virtual são escritas em C/C++. O kernel do Linux é escrito em C. Com exceção do Kernel do Linux (que é publicado sob a GPL versão 2.0), todo o resto do código fonte do Android é, em sua maioria, licenciado sob a Apache Software License 2.0[36] e pode ser descarregado da internet[37].

3.5.2 Desenvolvimento de aplicações

Nesta seção, descreveremos rapidamente os tópicos fundamentais para o desenvolvimento de aplicações no Android. Informações mais detalhadas podem ser encontradas no sítio do produto[38].

Como descrito anteriormente, as aplicações para Android são escritas em Java utilizando um arcabouço de bibliotecas específico. Um kit de desenvolvimento, disponível para descarga

[35] http://code.google.com/p/dalvik/
[36] http://www.apache.org/licenses/LICENSE-2.0
[37] http://source.android.com/
[38] http://developer.android.com/guide/topics/fundamentals.html

no sítio do Android[39], pode ser utilizado para compilar e empacotar os binários junto com outros arquivos eventualmente necessários para distribuição. Após instalada, cada aplicação é executada em um ambiente seguro e controlado, utilizando um usuário do sistema específico (um usuário diferente é criado para cada aplicação instalada), de forma que roda em seu próprio processo e máquina virtual, com o mínimo de permissões necessárias para seu funcionamento. As permissões para uso de recursos do sistema devem ser dadas pelo usuário do aparelho no momento da instalação.

As aplicações no sistema Android são divididas em **componentes**, definidos como "pontos através dos quais o sistema pode acessar a aplicação". Cada componente pode ser de um de quatro tipos: `atividade` (uma tela com interface de usuário), `serviço` (operações sem interface), `provedor de conteúdo` (gerenciamento de persistência de dados) ou `receptor de mensagens difundidas` (responde a mensagens do sistema). Qualquer aplicação pode iniciar componentes de outra aplicação e eventualmente receber de volta o resultado da execução daquele componente.

Uma aplicação não possui permissão para executar os componentes de outra aplicação diretamente, e portanto a conexão entre componentes de aplicações distintas deve ser intermediada pelo sistema. O acesso a componentes dos tipos `atividade`, `serviço` e `receptor de mensagens` são feitos através de `mensagens de intenção` assíncronas, enviadas pela aplicação ao sistema. O acesso a componentes do tipo `provedor de conteúdo` é feito através de um objeto específico, chamado "resolvedor de conteúdo".

Com o objetivo de prover para o sistema todas as informações necessárias para seu correto funcionamento, uma aplicação deve incluir um arquivo chamado **arquivo de manifesto**. O arquivo de manifesto declara para o sistema os componentes existentes na aplicação, identifica as permissões necessárias (como acesso à internet ou acesso de leitura aos contatos), declara a versão mínima das bibliotecas requeridas, as funcionalidades de hardware e software (como acesso ao microfone ou comunicação via bluetooth), outras bibliotecas necessárias, entre outras informações.

Por fim, uma aplicação pode ser constituída de mais do que simplesmente código em Java: pode conter, por exemplo, imagens, arquivos de áudio, vídeo, certificados criptográficos, etc. Uma aplicação Android permite que esses recursos sejam definidos separadamente do código e empacotados junto com a aplicação compilada[40]. Esta característica permite maior flexibilidade na manutenção de recursos (separada da manutenção do código), além de possibilitar a otimização dos recursos para mais de uma configuração de dispositivo.

3.5.3 Entrada e saída de áudio e interatividade

Existem diversas classes disponíveis no arcabouço disponibilizado pelo projeto Android que permitem a obtenção e reprodução de som utilizando arquivos de áudio e também microfone e alto-falante disponíveis no hardware[41]. Para poder utilizá-las para os fins deste trabalho, é necessário verificar a possibilidade de redirecionamento do fluxo de áudio para que passe por um ou mais estágios de processamento antes do envio para seu destino final, nem que para isto seja necessária a permissão do usuário no momento da instalação da aplicação.

Ao invés de simplesmente tocar o resultado de um processamento, uma outra opção é fazer a transmissão do sinal pela rede (*streaming*). É possível encontrar programas (pagos) para Android que realizam a transmissão de áudio utilizando protocolos proprietários[42]. Seria muito

[39] http://developer.android.com/sdk/index.html
[40] http://developer.android.com/guide/topics/resources/index.html
[41] http://developer.android.com/reference/android/media/package-summary.html
[42] http://xiialive.com/

interessante estudar a viabilidade de uma versão livre e grátis nos moldes de programas com licença livre utilizados atualmente para transmissão de áudio via internet, como por exemplo a dupla Icecast[43] (servidor) e Darkice[44] (cliente).

Uma outra opção ainda é a utilização de diversos dispositivos conectados para realizar a mixagem de áudio colaborativa e em tempo real. Questões como interfaces para comunicação (bluetooth, rede, etc), taxa de amostragem da interface de som de cada modelo de aparelho, taxa de amostragem dos arquivos de áudio utilizados, capacidade de transferência (banda) e latência devem ser consideradas para tornar viável esta idéia.

3.5.4 Propostas de análise

Algumas idéias para analisar a plataforma Android são:

- Determinar a intensidade computacional máxima como função dos modelos de aparelhos disponíveis e a curva de crescimento da capacidade de cada modelo.
- Determinar as possibilidades de interação entre dispositivos e a possibilidade de atuação conjunta em um mesmo ambiente sonoro virtual e real.
- Estudar as possibilidades de implementação de interface com ferramentas de processamento de sinais com licença livre como, por exemplo, Pure Data e CSound.

Isto conclui uma visão geral sobre o sistema Android e algumas propostas para exploração da plataforma. No capítulo seguinte, veremos uma proposta de cronograma de atividades para dar sequência ao projeto.

[43] http://www.icecast.org/
[44] http://code.google.com/p/darkice/

Capítulo 4

Trabalhos futuros

4.1 Testes preliminares

Desde o início da elaboração deste trabalho (no final de 2010), alguma familiaridade já foi obtida com as plataformas propostas. Temos em mãos um Arduino para testes e já foi possível realizar o processamento de áudio em tempo real, da forma descrita no trabalho do Laboratório para Ciência da Computação Experimental da Academia de Artes Midiáticas de Colônia (seção 1.3.1), e implementar alguns filtros passa-banda simples. Também possuímos uma placa gráfica com GPU disponível para testes e utilizando a linguagem CUDA (descrita na seção 3.4.3), já conseguimos transferir dados para a memória da placa e realizar operações em paralelo. No caso do Android, ainda não possuímos um equipamento exclusivo para testes, mas é possível utilizar o emulador disponível no sítio do projeto para iniciar o desenvolvimento[1].

Como dissemos na seção 3.2, a escolha do que será implementado ainda depende de uma avaliação mais profunda de cada plataforma. Para isto, propomos no cronograma um período de testes preliminares com objetivo de determinar o que seria possível de implementar e experimentar em cada plataforma. A idéia é, neste período de testes, expandir os experimentos com cada plataforma, estudando e utilizando aplicações e arcabouços como os descritos no capítulo 3. Assim, num primeiro momento pretendemos expandir os experimentos com as seguintes abordagens:

- **Arduino:** implementação de cálculos em blocos.
- **GPU:** implementação da delegação de cálculos de `FFT` para GPU no Pure Data
- **Android:** estudo do código de aplicações para processamento sonoro no Android, como por exemplo o aplicativo padrão chamado "Efeitos de Áudio", que interfere na reprodução de músicas, e o aplicativo mixDroid, descrito na seção 1.3.3.

4.2 Possibilidades de implementação

Como dissemos no capítulo 1, este trabalho ainda possui questões metodológicas em aberto e se beneficiará muito com as críticas e sugestões que forem trazidas no Exame de Qualificação.

A princípio tentaremos, para cada plataforma, implementar algoritmos de processamento como os listados na seção 1.4, de acordo com o nível de dificuldade computacional, na medida

[1] http://developer.android.com/guide/developing/tools/emulator.html

em que o avanço com as plataformas e o cronograma forem permitindo. A meta, lembramos, não é implementar toda a lista de algoritmos, mas caracterizar os limites de cada plataforma e ilustrar estes limites através de implementações de paradigmas algoritmicos. Assim, dois efeitos que tenham custo ou técnica de processamento parecidos não precisam ser ambos implementados para ilustrar que uma plataforma possui certa capacidade de processamento.

4.3 Cronograma proposto

Tendo em mente os comentários sobre os testes preliminares da seção 4.1 e sobre as possibilidades de implementação da seção 4.2, propomos um cronograma de atividades com planejamento de término do trabalho em Fevereiro de 2012, para defesa no mês seguinte. A idéia é interagir com as plataformas em paralelo, intercambiando técnicas e dividindo o tempo para cada tarefa do cronograma de forma igual.

	08/11	09/11	10/11	11/11	12/11	01/12	02/12	03/12
Testes preliminares								
Escolha dos algoritmos								
Implementações								
Testes finais e resultados								
Redação de artigos								
Redação do texto final								
Defesa								

Com isto, concluímos este relatório e esperamos que a banca convidada possa ter compreendido a proposta e os objetivos deste trabalho.

Referências Bibliográficas

BUCK, I. et al. Brook for gpus: stream computing on graphics hardware. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 23, p. 777–786, August 2004. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/1015706.1015800>.

BURRUS, C. Block realization of digital filters. *Audio and Electroacoustics, IEEE Transactions on*, v. 20, n. 4, p. 230 – 235, oct 1972. ISSN 0018-9278.

COOLEY, J.; TUKEY, J. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, v. 19, n. 90, p. 297–301, 1965.

COOLEY, J. W. How the fft gained acceptance. In: *Proceedings of the ACM conference on History of scientific and numeric computation*. New York, NY, USA: ACM, 1987. (HSNC '87), p. 97–100. ISBN 0-89791-229-2. Disponível em: <http://doi.acm.org/10.1145/41579.41589>.

CUCCHIARA, R.; GUALDI, G. Mobile video surveillance systems: An architectural overview. In: _____. *MOBILE MULTIMEDIA PROCESSING FUNDAMENTALS METHODS AND APPLICATIONS*. [S.l.]: Springer Berlin / Heidelberg, 2010. v. 5960, p. 89–109.

DEEPAK, G.; MEHER, P.; SLUZEK, A. Performance characteristics of parallel and pipelined implementation of fir filters in fpga platform. In: *Signals, Circuits and Systems, 2007. ISSCS 2007. International Symposium on*. [S.l.: s.n.], 2007. v. 1, p. 1 –4.

DIMITROV, S.; SERAFIN, S. An analog i/o interface board for audio arduino open sound card system. In: _____. *Proceedings of SMC 2011 - 8th Sound and Music Computing Conference*. [S.l.]: Padova University Press, 2011. p. 290–297.

DIMITROV, S.; SERAFIN, S. Audio arduino - an alsa (advanced linux sound architecture) audio driver for ftdi-based arduinos. In: _____. *NIME2011 Proceedings of the International Conference on New Interfaces for Musical Expression*. [S.l.: s.n.], 2011. p. 211–216. ISBN ISSN 2220-4792.

DIRICHLET, P. G. Sur la convergence des séries trigonométriques qui servent à représenter une fonction arbitraire entre des limites données. *Journal für die reine und angewandte Mathematik*, v. 4, p. 157–169, jun. 1829. Disponível em: <http://arxiv.org/abs/0806.1294>.

EYRE, J.; BIER, J. The evolution of dsp processors. *Signal Processing Magazine, IEEE*, v. 17, n. 2, p. 43 –51, mar 2000. ISSN 1053-5888.

FLORES, L. et al. *Musical Interaction Patterns: Communicating Computer Music Knowledge in a Multidisciplinary Project*. 2010.

FLYNN, M. Very high-speed computing systems. *Proceedings of the IEEE*, v. 54, n. 12, p. 1901 – 1909, dec. 1966. ISSN 0018-9219.

FOURIER, J. Mémoire sur la propagation de la chaleur dans le corps solides. *Noveau bulletin des sciences par la société philomatique de paris*, v. 6, n. 10, p. 215, 1807.

FOURNIER, A.; FUSSELL, D. On the power of the frame buffer. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 7, p. 103–128, April 1988. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/42458.42460>.

GALLO, E.; TSINGOS, N. Efficient 3d audio processing on the gpu. In: ACM. *Proceedings of the ACM Workshop on General Purpose Computing on Graphics Processors*. 2004. Disponível em: <http://www-sop.inria.fr/reves/Basilic/2004/GT04>.

GEIGER, G. Pda real time signal processing and sound generation on handheld devices (demo). In: . [S.l.: s.n.], 2003.

GIBB, A. M. *NEW MEDIA ART, DESIGN, AND THE ARDUINO MICROCONTROLLER: A MALLEABLE TOOL*. Tese (Doutorado) — Pratt Institute, 2010.

GOVINDARAJU, N. K. et al. Fast computation of database operations using graphics processors. In: *ACM SIGGRAPH 2005 Courses*. New York, NY, USA: ACM, 2005. (SIGGRAPH '05). Disponível em: <http://doi.acm.org/10.1145/1198555.1198787>.

GRAY, A. Parallel sub-convolution filter bank architectures. In: *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*. [S.l.: s.n.], 2003. v. 4, p. IV–528 – IV–531 vol.4.

GUHA, S. et al. Application of the two-sided depth test to csg rendering. In: *Proceedings of the 2003 symposium on Interactive 3D graphics*. New York, NY, USA: ACM, 2003. (I3D '03), p. 177–180. ISBN 1-58113-645-5. Disponível em: <http://doi.acm.org/10.1145/641480.641513>.

HALL, S. P.; ANDERSON, E. Operating systems for mobile computing. *J. Comput. Small Coll.*, Consortium for Computing Sciences in Colleges, USA, v. 25, p. 64–71, December 2009. ISSN 1937-4771. Disponível em: <http://portal.acm.org/citation.cfm?id=1629036.1629046>.

HE, B. et al. Efficient gather and scatter operations on graphics processors. In: *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*. [S.l.: s.n.], 2007. p. 1 –12.

HOLLAND, J. A universal computer capable of executing an arbitrary number of sub-programs simultaneously. In: *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*. New York, NY, USA: ACM, 1959. (IRE-AIEE-ACM '59 (Eastern)), p. 108–113. Disponível em: <http://doi.acm.org/10.1145/1460299.1460311>.

KAPASI, U. et al. Programmable stream processors. *Computer*, v. 36, n. 8, p. 54 – 62, aug. 2003. ISSN 0018-9162.

LEWIS, G. E. Too Many Notes: Computers, Complexity and Culture in "Voyager". 2000. Disponível em: <http://www.jstor.org/stable/1513376>.

MOHANTY, S. Gpu-cpu multi-core for real-time signal processing. In: *Consumer Electronics, 2009. ICCE '09. Digest of Technical Papers International Conference on*. [S.l.: s.n.], 2009. p. 1 –2.

MOORE, F. R. *Elements of computer music*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990. ISBN 0-13-252552-6.

MORELAND, K.; ANGEL, E. The fft on a gpu. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003. (HWWS '03), p. 112–119. ISBN 1-58113-739-7. Disponível em: <http://portal.acm.org/citation.cfm?id=844174.844191>.

OPPENHEIM, A. V.; SCHAFER, R. W.; BUCK, J. R. *Discrete-time signal processing (2nd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999. ISBN 0-13-754920-2.

OWENS, J. Streaming architectures and technology trends. In: *ACM SIGGRAPH 2005 Courses*. New York, NY, USA: ACM, 2005. (SIGGRAPH '05). Disponível em: <http://doi.acm.org/10.1145/1198555.1198766>.

OWENS, J. et al. Gpu computing. *Proceedings of the IEEE*, v. 96, n. 5, p. 879 –899, may 2008. ISSN 0018-9219.

OWENS, J. D. et al. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, Blackwell Publishing Ltd, v. 26, n. 1, p. 80–113, 2007. ISSN 1467-8659. Disponível em: <http://dx.doi.org/10.1111/j.1467-8659% -.2007.01012.x>.

PEASE, M. C. An adaptation of the fast fourier transform for parallel processing. *J. ACM*, ACM, New York, NY, USA, v. 15, p. 252–264, April 1968. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/321450.321457>.

PEETERS, G. *A large set of audio features for sound description (similarity and classification) in the CUIDADO project*. [S.l.], 2004.

PUCKETTE, M. Pure data: another integrated computer music environment. In: *in Proceedings, International Computer Music Conference*. [S.l.: s.n.], 1996. p. 37–41.

REDMILL, D.; BULL, D. Design of low complexity fir filters using genetic algorithms and directed graphs. In: *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1997. GALESIA 97. Second International Conference On (Conf. Publ. No. 446)*. [S.l.: s.n.], 1997. p. 168 –173. ISSN 0537-9989.

RENFORS, M.; NEUVO, Y. The maximum sampling rate of digital filters under hardware speed constraints. *Circuits and Systems, IEEE Transactions on*, v. 28, n. 3, p. 196 – 202, mar 1981. ISSN 0098-4094.

ROWE, R. Machine listening and composing with cypher. *Computer Music Journal*, v. 16, n. 1, 1992.

ROWE, R. Machine listening and composing with cypher. 1992.

SERNEC, R.; ZAJC, M.; TASIC, J. The evolution of dsp architectures: towards parallelism exploitation. In: *Electrotechnical Conference, 2000. MELECON 2000. 10th Mediterranean.* [S.l.: s.n.], 2000. v. 2, p. 782 – 785 vol.2.

SHROEDER, F. et al. Addressing the network: Perfomative strategies for playing apart. In: *Proceedings of International Computer Music Conference 2007*. International Computer Music Association, 2007. Disponível em: <http://eprints.bournemouth.ac.uk/9450/>.

THOMPSON, C. J.; HAHN, S.; OSKIN, M. Using modern graphics architectures for general-purpose computing: a framework and analysis. In: *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002. (MICRO 35), p. 306–317. ISBN 0-7695-1859-1. Disponível em: <http://portal.acm.org/citation.cfm?id=774861.774894>.

VENKATASUBRAMANIAN, S. The graphics card as a streaming computer. *CoRR*, cs.GR/0310002, 2003.

VERCOE, B. L.; ELLIS, D. P. Real-time csound: software synthesis with sensing and control. 1990.

WONG, T.-T. et al. Discrete wavelet transform on consumer-level graphics hardware. *Multimedia, IEEE Transactions on*, v. 9, n. 3, p. 668 –673, april 2007. ISSN 1520-9210.

ZöLZER, U. et al. *DAFX:Digital Audio Effects*. John Wiley & Sons, 2002. Hardcover. ISBN 0471490784. Disponível em: <http://www.worldcat.org/isbn/0471490784>.