Uma nova técnica para criação de sistemas de videoconferência em dispositivos móveis Android

Giancarlo Rampanelli, Felipe Cecagno, André Schulz, Valter Roesler
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Rio Grande do Sul, Brasil
{grampanelli, fcecagno, aschulz, roesler}@inf.ufrgs.br

RESUMO

Este artigo tem o objetivo de apresentar uma nova estratégia para criação de videoconferências em dispositivos móveis com sistema operacional Android. É apresentado um estudo sobre as classes multimídia padrão do kit de desenvolvimento de software do Android, detalhando as limitações que dificultam o desenvolvimento de sistemas multimídia em tempo real sem a utilização de bibliotecas externas. Várias soluções propostas em trabalhos anteriores são comparadas, ressaltando prós e contras, e conduzindo a necessidade da utilização de uma nova estratégia, detalhada neste artigo, que é baseada principalmente em bibliotecas alternativas. A ideia proposta é então validada num sistema Android real, mostrando suas vantagens.

ABSTRACT

This paper aims to present a novel strategy for implementation of real-time interactive multimedia systems on Android-powered mobile devices. We will present a study about the standard multimedia classes of Android's software development kit, detailing the limitations that hinder the development of real time multimedia systems without using external libraries. Some alternative solutions proposed in previous works will be presented, driving to the need of using a new strategy, detailed in this article, which is based primarily on alternative libraries. The proposed idea is then validated in a real Android system, showing its advantages.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; J.7 [**Computers in other systems**]: Real time; I.6 [**Simulation and modeling**]: Applications; H.4 [**Information systems applications**]: Communications ApplicationsComputer conferencing, teleconferencing, and videoconferencing

General Terms

Design, Experimentation

Keywords

Interação multimídia de tempo real, Vídeo, JNI

1. INTRODUÇÃO

Sistemas de transmissão de vídeo interativo em tempo real são complexos de serem desenvolvidos, uma vez que, além da necessidade de áudio e de vídeo serem capturados, transmitidos, recebidos e exibidos de forma contínua por todos os participantes, é necessário que esse processo ocorra com baixa latência. Se o intervalo de tempo entre a captura e a exibição nos pontos remotos ultrapassar 400 milissegundos, o resultado pode ser uma interação altamente frustrante [16].

É um desafio desenvolver com qualidade tais sistemas pois as redes comerciais atuais, em geral, possuem largura de banda restrita. O desafio se torna ainda maior na área de dispositivos móveis, que tipicamente possuem hardware limitado, o que torna a manipulação de vídeo um ponto crítico do sistema. Além disso, smartphones e tablets se conectam à Internet através de redes 3G ou Wi-Fi, ambas caracterizadas por altas taxas de perda de pacotes e baixas velocidades [14].

Um sistema de interação por vídeo pode ser dividido em dois módulos:

- Um módulo origem, composto de:
  - um sub-módulo que captura áudio e vídeo não comprimidos do microfone e da câmera, respectivamente;
  - um sub-módulo que codifica áudio e vídeo;
  - um sub-módulo que envia pela rede os dados codificados;
- Um módulo destino, composto de:
  - um sub-módulo que recebe da rede dados de áudio e vídeo codificados;
  - um sub-módulo que decodifica áudio e vídeo;
  - um sub-módulo que exibe os dados decodificados para o usuário.

Neste trabalho será apresentada uma nova estratégia para criação de sistemas de videoconferência em dispositivos móveis com sistema operacional Android. Para justificar as escolhas feitas e a utilização de bibliotecas alternativas, primeiramente será feita uma análise da Software Development Kit (Kit de Desenvolvimento de Software - SDK) do Android para multimídia, seguida de uma comparação de possíveis estratégias propostas por trabalhos anteriores para desenvolvimento de sistemas de interação por vídeo em tempo real com a utilização apenas das classes padrão do Android, e de suas limitações. Por fim, para validar a estratégia, são realizadas implementações e os devidos resultados são apresentados.

2. ANÁLISE DA SDK DO ANDROID PARA MULTIMÍDIA

O Android é um sistema operacional de código aberto voltado principalmente para dispositivos móveis, desenvolvido numa colaboração entre Google e outros membros da Open Handset Alliance, e baseado no núcleo do Linux. Em comparação com outros sistemas operacionais existentes para dispositivos móveis, o Android é caracterizado por aplicações de vídeo mais simples. Isso se deve às limitações das APIs disponibilizadas pelo Android para programação de aplicativos de vídeo e, alternativamente, à grande complexidade para criar tais aplicativos utilizando APIs alternativas. A dificuldade de se encontrar bons aplicativos de vídeo para Android se torna ainda mais marcante na área de vídeo interativo.

Aplicativos para Android são escritos na linguagem de programação Java e executam sobre a máquina virtual Dalvik. O desenvolvimento de aplicativos é feito com o auxílio da SDK do Android, que fornece, além das plataformas para compilação, uma variedade de ferramentas úteis no desenvolvimento, como depuradores de código e emuladores [11].

É possível ainda desenvolver módulos de aplicativos nas linguagens C e C++, visto que o Android executa sobre o núcleo do Linux [12]. Para isso, é necessário utilizar a Native Development Kit (Kit de Desenvolvimento Nativo - NDK) para Android. Entretanto, não é possível executar um aplicativo totalmente desenvolvido utilizando a NDK - é necessário que o ponto de partida do aplicativo seja escrito em Java. A interação entre o código Java e o código nativo é feita através da JNI (Java Native Interface), interface que permite que um código que está executando em uma máquina virtual interaja com um código nativo [12].

As classes de referência da SDK do Android que podem ser importantes para o desenvolvimento de um sistema de interação através de áudio e vídeo podem ser classificadas como segue:

- Para o módulo de origem: MediaRecorder, Camera e AudioRecord;
- Para o módulo de destino: MediaPlayer e AudioTrack.

Existem outras classes relacionadas, porém estas apresentam pequenas extensões ou encapsulamentos das classes acima, sem importância para este trabalho. Também existem classes relacionadas ao envio e recebimento de dados genéricos pela rede que podem ser utilizadas em conjunto com as classes acima. A seguir as classes supracitadas serão detalhadas, e com base nas características de cada uma serão apresentadas estratégias de desenvolvimento de um sistema de vídeo interativo bem como suas limitações.

2.1 Classe MediaRecorder

A classe MediaRecorder realiza a captura de áudio e vídeo (através do microfone e da câmera) e codifica os dados utilizando o hardware disponível no aparelho. Os dados codificados podem ser salvos em arquivo ou transmitidos pela rede por meio de sockets. A classe também permite a exibição de uma pré-visualização local da captura para o usuário. A utilização do MediaRecorder acontece da seguinte forma [6]:

1. Declarar uma instância da classe

```
MediaRecorder mMediaRecorder = new MediaRecorder();
```

2. Escolher as fontes de áudio e vídeo:

```
mMediaRecorder.setAudioSource(<fonte_de_audio>);
mMediaRecorder.setVideoSource(<fonte_de_video>);
```

3. Escolher o destino (arquivo ou socket)

```
mMediaRecorder.setOutputFile(<destino>);
```

4. Iniciar a captura:

```
mMediaRecorder.prepare();
mMediaRecorder.start();
```

5. Exibir o preview da captura:

```
mMediaRecorder.setPreviewDisplay(<superficie_local>);
```

É importante ressaltar que, apesar do funcionamento da classe ser bastante simples e intuitivo, não existe nenhum mecanismo de acesso direto aos dados codificados, sem a necessidade de um passo intermediário. Essa característica impossibilita que os dados sejam encapsulados em um protocolo de aplicação antes de serem enviados. Para realizar essa tarefa usando a classe MediaRecorder, é necessário que um socket local seja aberto e intermedie os dados entre a classe e o receptor local, que no caso poderia ser um encapsulador.

Além disso, a classe está totalmente vinculada aos codificadores disponíveis no dispositivo, que são poucos. Não tendo acesso aos dados decodificados, não é possível aplicar sobre os mesmos um processo de codificação diferente do padrão, inviabilizando o uso da classe para este fim.

2.2 Classe Camera

A classe Camera realiza a captura de vídeo (a partir da câmera) e permite, bem como a calsse MediaRecorder, uma pré-visualização local da captura para o usuário. Ela pode ser utilizada de duas formas: para tirar fotografias (ou seja, capturar um quadro, codificá-lo e salvá-lo em arquivo) ou para configurar um callback que é executado a cada quadro capturado, e que recebe dados não codificados. A utilização da classe acontece como segue [6]:

1. Obter acesso à câmera:

```
Camera mCamera = Camera.open();
```

2. Escolher a superfície de preview:

```
mCamera.setPreviewDisplay(<superficie_local>);
```

3. Configurar o callback (opcional):

```
mCamera.setPreviewCallback(<contexto>);
```

4. Iniciar a captura:

```
mCamera.startPreview();
```

5. Tirar foto (opcional):

```
takePicture(<Camera.ShutterCallback>,
    <Camera.PictureCallback_raw>,
    <Camera.PictureCallback_jpeg>);
```

6. Implementar o callback que recebe o quadro não codificado (opcional):

```
@Override
public void onPreviewFrame (byte[] data, Camera camera){}
```

2.3 Classe AudioRecord

A classe AudioRecord realiza a captura de áudio (a partir do microfone), e os dados capturados não codificados são colocados em um buffer principal. Para utilizar os dados capturados pode-se fazer uma cópia do buffer principal para um buffer secundário. O funcionamento da classe AudioRecord segue as seguintes etapas [6]:

1. Declarar uma instância da classe:

```
AudioRecord mAudioRecord = new AudioRecord(
    MediaRecorder.AudioSource.MIC,
    <taxa_de_amostragem>, <numero_de_canais>,
    <bits_por_amostra>, <tamanho_do_buffer_principal>);
```

2. Iniciar a captura:

```
mAudioRecord.startRecording();
```

3. Transferir o conteúdo do buffer principal (quando desejado) para um buffer secundário:

```
record.read(<buffer_secundario>,
    <indice_inicial_do_buffer_secundario>,
    <numero_de_bytes_a_transferir>);
```

2.4 Classe MediaPlayer

A classe MediaPlayer fornece funções de alto nível para a exibição de vídeo e áudio. Permite a escolha de origem entre arquivo local, servidor web e streaming por RTSP [13]. Configurada a origem do vídeo, a classe decodifica o áudio e o vídeo utilizando os recursos de hardware disponíveis e exibe-os para o usuário. As etapas necessárias são as seguintes [6]:

1. Declarar uma instância da classe:

```
MediaPlayer mMediaPlayer = new MediaPlayer();
```

2. Escolher a origem do vídeo:

```
mMediaPlayer.setDataSource(<origem_do_video>);
```

3. Iniciar o processo de leitura (local ou remota), decodificação e renderização:

```
mMediaPlayer.prepare();
mMediaPlayer.start();
```

2.5 Classe AudioTrack

A classe AudioTrack serve para renderizar áudio, e funciona de maneira similar à classe AudioRecord - para renderizar uma amostra de áudio, deve-se transferí-la de um buffer secundário para o buffer principal da classe. As etapas necessárias são as seguintes [6]:

1. Declarar uma instância da classe:

```
mAudioTrack = new AudioTrack(<tipo_de_stream>,
    <taxa_de_amostragem>,
    <numero_de_canais>,
    <bits_por_amostra>,
    <tamanho_do_buffer_principal>,
    <modo_(estatico_ou_stream)>);
```

2. Caso o modo utilizado seja o stream, chamar o método play():

```
mAudioTrack.play();
```

   Nada será renderizado pois o buffer principal está vazio.

3. Transferir o conteúdo do buffer secundário (quando desejado) para o buffer principal:

```
mAudioTrack.write(<buffer_secundario>,
    <indice_inicial_do_buffer_secundario>,
    <numero_de_bytes_a_transferir>);
```

4. Caso o modo utilizado seja **stream**, cada novo dado recebido no buffer principal será reproduzido sem a necessidade de chamar o método play();

5. Caso o modo utilizado seja **estático**, chamar o método play() para renderizar os dados (quando desejado):

```
mAudioTrack.play();
```

3. ESTRATÉGIAS PARA VÍDEO INTERATIVO COM AS CLASSES PADRÃO

3.1 Estratégia 1

É a técnica implementada em [15].

3.1.1 Na origem

Essa estratégia se baseia na classe MediaRecorder para realizar captura de áudio e vídeo, e salva pequenas amostras do conteúdo capturado codificado (2 segundos) em arquivos temporários. Durante a captura os dados do arquivo vão sendo enviados ao ponto remoto através de uma classe do Android para transmissão de dados genéricos. Ao finalizar a gravação da captura no arquivo temporário, a classe MediaRecorder insere no início do arquivo um cabeçalho e metadados do contêiner escolhido (3GPP ou MPEG-4). Esses dados, por sua vez, também são enviados para o ponto remoto, fechando o ciclo e reiniciando o processo.

3.1.2 No destino

O ponto remoto, por sua vez, recebe os dados de áudio e vídeo provenientes da origem e armazena-os em um arquivo temporário. Quando o cabeçalho e os metadados chegam, o ponto remoto finaliza o arquivo, e nesse instante é possível reproduzir o seu conteúdo através da classe MediaPlayer. O novo arquivo temporário deve ter nome diferente do anterior, pois o início do recebimento acontecerá antes do término da reprodução do arquivo recebido.

3.1.3 Análise da estratégia

Segundo o autor, a classe MediaRecorder leva em média 1,3 segundos para ser inicializada, e não é possível inicializar um novo MediaRecorder antes de finalizar o anterior. Logo, cada 2 segundos de conteúdo capturado será seguido de 1,3 segundos sem conteúdo. No destino, a classe MediaPlayer leva em média 0,83 segundos para ser inicializada, o que também representaria um intervalo de tempo sem conteúdo. Entretanto, os tempos de inicialização não se somam como atraso, mas se sobrepõem, como mostra a figura 1.

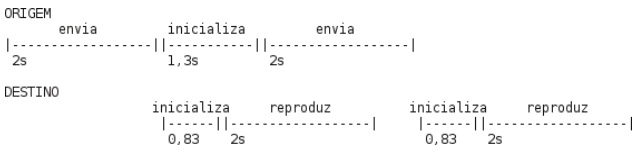

Figura 1: Estratégia 1 - Linha do tempo

Uma análise rápida revela um atraso mínimo de 2,83 segundos, que é o tempo de captura do arquivo temporário na origem somado ao tempo de inicialização do MediaPlayer no destino. O autor reconhece que a técnica não serve para interação em tempo real, mas afirma que é o melhor resultado possível utilizando apenas as classes padrão do Android.

3.2 Estratégia 2

É a técnica implementada em [9]. Esta técnica utiliza a capacidade da classe MediaPlayer em reproduzir conteúdo ao vivo utilizando o protocolo RTSP.

3.2.1 Na origem

A classe MediaRecorder é utilizada para capturar vídeo. Os dados codificados são colocados diretamente em um socket local, que por sua vez é lido por um outro módulo que implementa o protocolo RTSP. Dessa forma os dados são enviados para o destino.

3.2.2 No destino

A classe MediaPlayer é inicializada e configurada para receber o conteúdo de um fluxo RTSP.

3.2.3 Análise da estratégia

O método resulta em um sistema de vídeo ao vivo, mas não em um sistema de interação em tempo real, pois a implementação do RTSP no Android não serve para tempo real. Tal afirmação foi comprovada através do seguinte experimento:

1. Uma câmera foi conectada a um PC, tendo seu conteúdo capturado através do software VideoLAN Media Player (VLC) [10];

2. O fluxo de dados proveniente do VLC foi direcionado para um servidor de streaming RTSP (Darwin Streaming Server [2]) especificado por um arquivo SDP (Session Description Protocol);

3. O fluxo RTSP foi recebido em um smartphone Motorola Milestone (Android 2.0.1) utilizando a classe MediaPlayer.

O atraso médio obtido durante o experimento foi de 10 segundos. Ao receber o mesmo fluxo em um VLC instalado em outro computador da mesma rede, o atraso foi de aproximadamente 1 segundo. Além do experimento proposto, também foi realizada uma comparação do RTSP do Android com o RTSP do VLC utilizando diversos links RTSP públicos na Internet. O mesmo resultado anterior foi verificado.

Comparado ao método anterior, nesta estratégia não ocorrem paradas na exibição. Porém, segundo os testes de desempenho realizados com a classe MediaPlayer, o atraso é maior do que no método anterior, devido ao baixo desempenho do RTSP utilizado juntamente com a classe MediaPlayer do Android.

3.3 Estratégia 3

Ao invés de utilizar a classe MediaRecorder para realizar a captura de áudio e vídeo, utiliza-se as classes Camera e AudioRecord. Ambas as classes permitem acesso direto aos dados não codificados. No entanto, não há nenhuma classe padrão do Android que codifique um fragmento de áudio ou vídeo a partir de dados não codificados, e quadros não codificados são grandes demais para serem transmitidos pela rede em tempo viável para vídeo interativo.

Poderia ser implementado um codificador/decodificador em Java. No entanto, codificação e decodificação são processos que exigem um uso intensivo de CPU, ou seja, possuem um alto custo computacional, o que torna a solução inviável, uma vez que a implementação executaria sobre uma máquina virtual em um dispositivo com baixo poder computacional.

3.4 Estratégia 4

3.4.1 Na origem

Utilizar a função de foto da classe Camera, que retorna um quadro codificado em JPEG, e enviar cada quadro independentemente pela rede para o destino.

3.4.2 No destino

Receber cada quadro e exibi-lo com alguma classe padrão do Android que decodifique e exiba imagens JPEG.

3.4.3 Análise da estratégia

Codecs de vídeo são tipicamente mais eficazes do que codecs de imagens, visto que um vídeo é um conjunto de imagens relacionadas entre si, e a compressão leva em consideração a relação entre essas imagens. O codec JPEG apenas comprime uma imagem sem relacioná-la com nenhuma outra, portanto tem menos poder de compressão. Além disso, a função de foto do Android não foi projetada para este uso e é pouco eficiente. Um dos motivos é que quando a função foto é chamada, o preview do vídeo capturado é interrompido. Para tirar uma nova foto é preciso reiniciar o preview.

O método em si não foi testado pois, conforme justificado anteriormente, a estratégia é inviável independente da implementação. Outro problema da estratégia é não ter um correspondente para o áudio - somente o vídeo seria transmitido.

3.5 Estratégia 5

Como o Android possui código aberto, esta estratégia propõe copiar as bibliotecas nativas (C/C++) internas do sistema operacional que interagem com as classes MediaPlayer e MediaRecorder e com o hardware dos aparelhos, inserindo-as em um novo projeto compilável através da NDK. Desse modo teria-se acesso aos quadros codificados e decodificados pelo hardware e uma classe de transmissão de dados genéricos do Android poderia ser utilizada para realizar a transmissão e recepção dos dados multimídia. A implementação poderia ser organizada da seguinte forma:

- Captura: câmera e microfone seriam acessados diretamente pelo código nativo, utilizando as bibliotecas internas do Android;
- Codificação e decodificação: em hardware, utilizando as bibliotecas internas do Android;
- Transmissão e recepção: alguma classe padrão do Android para transmissão e recepção de dados genéricos;
- Renderização: tela e alto falante seriam acessados diretamente pelo código nativo, utilizando as bibliotecas internas do Android.

Esta seria a solução mais eficiente possível, visto que todas as tarefas com alto custo computacional seriam realizadas em hardware, e os dados trafegariam pela rede comprimidos. Contudo, as bibliotecas nativas são dependentes de plataforma, ou seja, cada dispositivo tem particularidades de hardware que são generalizadas através da implementação de suas bibliotecas nativas. Utilizando a estratégia descrita, seria necessário manter diversas versões da mesma biblioteca, uma pra cada dispositivo. Por esse motivo, a manipulação de bibliotecas nativas internas do Android é desencorajado pelos desenvolvedores do Android. Além disso, essa solução é limitada à utilização dos codecs suportados por padrão pelos dispositivos.

4. SOLUÇÃO ADOTADA

Pela análise acima pôde-se concluir que, até o momento, não é possível construir um sistema de interação por vídeo em tempo real utilizando apenas as classes padrão do Android. A solução proposta utiliza algumas das classes padrão do Android apresentadas, mas se baseia principalmente em implementações alternativas.

4.1 Na origem

A classe Camera é utilizada para realizar a captura de vídeo, conforme citado na seção 2.2, através da configuração de um callback que recebe os quadros não codificados. Através dessa classe é possível configurar os parâmetros desejados, tais como resolução e número de quadros por segundo. Os quadros não codificados são apresentados no formato YUV420SP, também conhecido como NV21, único formato de captura suportado por todos os dispositivos Android.

Os quadros não codificados são enviados para a interface em C++ através da JNI para a realização das tarefas de conversão e codificação. Esse envio é feito através do método *setPreviewCallbackWithBuffer*, somente possível com a utilização de Java Reflection em versões do Android anteriores a 2.2. Esse método, em comparação com o método *setPreviewCallback*, reduz pela metade o custo computacional da operação, permitindo assim o dobro da taxa de quadros por segundo, se desejado. No método *setPreviewCallback*, o Garbage Collector interrompe a execução do aplicativo a cada quadro.

A captura de áudio é feita através da classe AudioRecord, conforme citado na seção 2.3. As amostras de áudio são capturadas no formato não codificado RAW (PCM), e o mesmo procedimento descrito acima é utilizado para enviar os dados para a interface em código nativo.

Como Android executa sobre um núcleo do Linux, foi possível gerar uma compilação do FFmpeg através da NDK [12]. O FFmpeg [3] é um projeto que implementa diversos codificadores e decodificadores em software, escrito na linguagem C, e pequenas adaptações possibilitam sua compilação para o Android. Dessa forma aproveita-se o desempenho melhor do código nativo, em comparação com o código Java, para executar operações com alto custo computacional.

Entretanto, o FFmpeg exige para diversos codecs que o quadro cru de entrada esteja no formato YUV420P, diferentemente do formato de saída da classe Camera. A conversão intermediária necessária entre formatos é realizada em software através do SwScale, que é parte do FFmpeg.

O quadro codificado com o codec desejado é então enviado para o ponto remoto. Isso pode ser feito tanto através do código nativo, via Berkeley Sockets, quanto através do código Java, via Java Sockets. Entretanto, a segunda opção é mais custosa, uma vez que os dados devem voltar pro contexto Java via JNI. Antes do envio, ainda, é possível encapsular os dados no protocolo desejado (como RTP ou RTMP).

4.2 No destino

No ponto remoto, o processo inverso é aplicado. Primeiramente os dados de áudio e vídeo devem ser recebidos através de sockets, que estar tanto no contexto Java quando no contexto de código nativo. Se os dados estiverem encapsulados em algum protocolo, o processo inverso deve ser aplicado a fim de isolar os dados de cada quadro de vídeo ou amostra de áudio. Os dados são então decodificados com o FFmpeg compilado para código nativo. Visto que não há como acessar o hardware de som do dispositivo diretamente do código nativo, a reprodução de som é realizada através da classe AudioTrack, detalhada na seção 2.5.

A troca de contexto dos dados entre código nativo e código Java é feita através da JNI, porém, não diretamente. Invocando um método Java a partir do contexto nativo, ou seja, utilizando a JNI diretamente, o Garbage Collector é executado a cada quadro, causando uma enorme queda no desempenho (100ms de interrupção, em média). Alternativamente foi utilizada uma outra técnica da JNI que apresentou melhor desempenho. A técnica consiste na criação de um *array* em Java que representa um quadro, e um *array* de mesmo tamanho em código nativo. No código nativo invoca-se funções da JNI que associam os dois *arrays* à mesma posição de memória. Dessa forma, cada atualização do *array* no código

nativo com novos dados decodificados é propagada ao *array* do Java, que reproduz os dados de áudio conforme citado anteriormente. Assim, sem cópias ou alocação de memória adicional, a solução mostrou-se a mais eficiente para a realização da tarefa proposta.

De maneira similar, não há como acessar o hardware da tela diretamente do código nativo para renderizar os quadros de vídeo. Adicionalmente, não há uma classe do Android que receba dados de vídeo não codificados e renderize-os. Por isso, optou-se por utilizar o OpenGL ES [7] para a realização da tarefa. O OpenGL ES é a versão do OpenGL para sistemas embarcados, e possibilita acesso aos recursos de hardware gráfico diretamente do contexto nativo.

Através do OpenGL ES, desenha-se um retângulo com as dimensões desejadas e aplica-se uma textura ao retângulo contendo o quadro de vídeo. No entanto, para que a textura seja exibida na tela, é necessário que a função de renderização seja chamada de dentro do método *onDrawFrame* da classe GLSurfaceView.Render e retorne para o método a cada quadro. Além disso, o OpenGL ES só permite aplicação de texturas no formato RGB, e o FFmpeg, por sua vez, só decodifica quadros de vídeo para o formato YUV420P, sendo necessária uma conversão intermediária.

Caso a resolução do vídeo recebido seja diferente das dimensões da região de pintura da tela, utiliza-se o próprio OpenGL ES para a realização do redimensionamento. Isso porque foi verificado experimentalmente que o OpenGL ES possui melhor desempenho do que o FFmpeg para realizar tal operação, até porque no OpenGL ES o redimensionamento é feito em hardware na versão 1.1 e posterior.

Outra limitação do OpenGL ES é a capacidade de, em algumas versões, somente suportar aplicação de texturas sobre retângulos de dimensões potência de dois. Ou seja, para desenhar um vídeo com resolução 320x240, deve-se utilizar uma área de desenho de pelo menos 512x256. Dessa forma que a limitação foi superada: cria-se uma região com dimensões potência de dois imediatamente maiores do que o tamanho do vídeo, e aplica-se a textura de qualquer dimensão sobre parte dessa região.

5. ANÁLISE DA TÉCNICA E RESULTADOS

A fim de validar experimentalmente a estratégia proposta, foram realizados alguns testes em dispositivos com Android disponíveis no mercado - o HTC Magic, o Motorola Milestone e o Samsung Galaxy S. As especificações técnicas dos dispositivos relevantes para os experimentos são apresentadas na tabela 1, e especificações detalhadas podem ser encontradas em [4].

5.1 Experimento 1

O objetivo do teste foi verificar experimentalmente a capacidade máxima de decodificação de um dispositivo móvel com Android. Utilizou-se um HTC Magic e um arquivo de vídeo local com resolução 176x144. A taxa de decodificação alcançada foi de 227 quadros por segundo, ou seja, tempo médio por quadro de 4,41ms.

5.2 Experimento 2

Tabela 1: Dispositivos utilizados nos experimentos

Modelo	Processador	Android	Resolução	OpenGL ES
HTC Magic	528MHz	1.6	320x480	1.1
Motorola Milestone A853	500MHz	2.0.1	480x854	1.0
Samsung Galaxy S GT-I9000B	1000MHz	2.1-update1	480x800	1.1

Foi implementado um aplicativo de videoconferência entre dispositivos móveis aplicando a técnica apresentada. Para verificar experimentalmente o funcionamento do aplicativo, foi realizada uma conversa por vídeo entre o Motorola Milestone e o Samsung Galaxy S.

Os parâmetros de codificação de vídeo utilizados no teste foram:

- Resolução: 320x240
- Codec: MPEG-4
- Taxa de bits: 256kbps
- GOP: 12
- Taxa de quadros: 15

Os parâmetros de codificação de áudio utilizados no teste foram:

- Codec: MP2
- Taxa de bits: 64kbps
- Frequência: 22050Hz
- Bits por amostra: 16
- Canais: 1

O vídeo foi ampliado localmente com o OpenGL ES para a resolução máxima do aparelho. A tabela 2 exibe os resultados do teste.

Tabela 2: Resultados do experimento 2

Dispositivo	Tipo de medida	Taxa de quadros	Tempo médio por quadro
Motorola	Captura do vídeo	15	1,16ms
Samsung	Captura do vídeo	15	0,72ms
Motorola	Renderização do vídeo	15	16,03ms
Samsung	Renderização do vídeo	14	2,88ms

Enquanto a captura do vídeo teve desempenho semelhante nos dois dispositivos, a renderização do vídeo teve um melhor desempenho no Samsung Galaxy S. A diferença de performance é justificado pela versão do OpenGL ES do Samsung - conforme citado anteriormente, a partir da versão 1.1 o redimensionamento é realizado em hardware, enquanto nas versões anteriores tal tarefa é realizada em software.

5.3 Experimento 3

Realizamos uma transmissão de vídeo do Samsung Galaxy S para o Motorola Milestone. Os parâmetros de codificação do áudio e do vídeo utilizados foram os mesmos do experimento 2, porém com a taxa de quadros do vídeo igual a 30. A tabela 3 exibe os resultados do teste.

Tabela 3: Resultados do experimento 3

Dispositivo	Tipo de medida	Taxa de quadros	Tempo médio por quadro
Samsung	Captura do vídeo	30	0,58ms
Motorola	Renderização do vídeo	29	12,77ms

Uma taxa de quadros de vídeo igual a 30 por segundo representa um tempo de 33ms disponível para processamento de cada quadro. Considerando o tempo de decodificação médio do experimento 1, cada quadro tomou 17,18ms de processamento para ser exibido no Motorola Milestone, ou seja, o conteúdo não apresentou atraso cumulativo.

5.4 Experimento 4

O IVA é um sistema de videoconferência para ensino a distância com suporte a transmissão de áudio e vídeo em alta qualidade [17]. Utilizando a estratégia proposta neste trabalho, foi implementado um aplicativo para Android que permite a interação de usuários a partir de dispositivos móveis por áudio e vídeo com a videoconferência do IVA.

Neste experimento, o aplicativo Android recebeu um fluxo de dados codificado em MPEG-4 (vídeo) e MP2 (áudio) gerado a partir de um PC e distribuído para os participantes remotos pelo IVA. Os parâmetros de codificação do vídeo foram: taxa de bits de 1400kbps, resolução de 720x480 e taxa de quadros de 15 por segundo. O áudio foi codificado a uma taxa de 128kbps. O Motorola Milestone executando o aplicativo recebeu e exibiu esta transmissão mantendo a taxa de 15 quadros por segundo, com atraso levemente superior à exibição no PC. A figura 2 mostra o teste em execução.

Como o sistema IVA é desenvolvido na linguagem C++, e o grupo possui acesso ao seu código fonte, diversos módulos do sistema puderam ser reusados no aplicativo para Android. Dentre os módulos reusados, os mais importantes foram as bibliotecas que encapsulam o FFmpeg para codificação e decodificação, a biblioteca de filas de alta performance com múltiplos consumidores e a biblioteca que implementa os protocolos de transmissão de dados multimídia e de sinalização utilizados pelo sistema. Esse experimento comprovou a flexibilidade que o Android dá aos desenvolvedores, que podem incluir códigos e bibliotecas Java em seus projetos,

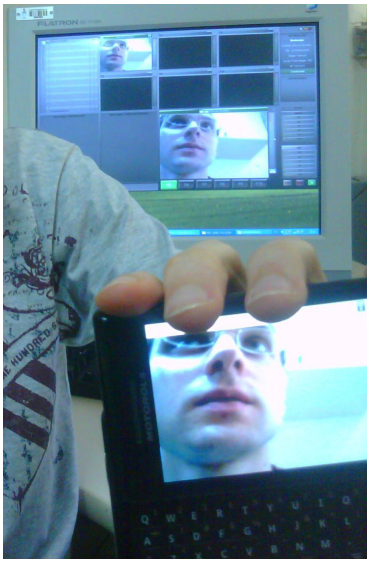

Figura 2: Integração ao sistema IVA

bem como código nativo em C e C++ compilado através da NDK.

5.5 Experimento 5

A estratégia também foi integrada ao sistema de videoconferência na web chamado BigBlueButton [1]. O BigBlueButton é um sistema de código aberto que permite a interação por vídeo e áudio, bem como bate papo, compartilhamento de tela e apresentações. O sistema utiliza a tecnologia Flash para transmissão de conteúdo, e o protocolo utilizado para interação entre clientes e servidor é o RTMP [8]. Os codecs utilizados pelo sistema são o Sorenson H.263 para vídeo e o μ-law para áudio. As resoluções possíveis vão de 160x120 até 1280x720, sendo o padrão 320x240.

O aplicativo de integração com o BigBlueButton foi implementado no âmbito do projeto GT-Mconf [5]. Apesar da tecnologia Flash ser suportada em dispositivos Android com versão 2.2 ou posterior, a decisão do projeto foi implementar um aplicativo nativo para Android, compatível com as versões mais antigas.

A figura 3 apresenta um vídeo sendo exibido no aplicativo Android, proveniente da videoconferência do BigBlueButton. A estratégia de implementação foi um pouco diferente da adotada no experimento 4, em que as bibliotecas de implementação dos protocolos de rede eram escritas em C++. Uma biblioteca de implementação do protocolo RTMP, escrita em Java, foi utilizada para a comunicação com o servidor, e tanto áudio quando vídeo eram recebidos no contexto do Java e enviados ao contexto de código nativo através da JNI.

6. CONCLUSÃO

Este artigo apresentou uma solução para implementação de sistemas de vídeo interativo em tempo real para Android, sendo consistente com a versão 1.5 do Android ou posterior. Os aspectos inovadores da estratégia são, principalmente, a codificação/decodificação de áudio e vídeo em software

Figura 3: Vídeo no Mconf: Integração ao sistema BigBlueButton

(com o FFmpeg compilado para o Android), a utilização da JNI para manipular os dados de áudio e de vídeo entre a camada nativa e a camada Java e a utilização de OpenGL ES para renderizar o vídeo. Todas essas escolhas são justificadas através de um estudo sobre as classes padrão do Android e suas limitações, seguida de uma comparação de estratégias propostas por trabalhos anteriores e seus resultados. Os testes da implementação da solução são apresentados, e seus resultados validam a utilização da técnica na implementação desse tipo de sistema.

O principal aspecto positivo da técnica é a flexibilidade dada ao desenvolvedor e ao sistema. A solução não envolve a utilização de um protocolo de transmissão específico ou de um codec - qualquer biblioteca de transmissão, escrita tanto em C/C++ quanto em Java, pode ser utilizada no sistema, e o mesmo não fica limitado aos codecs oferecidos pelo hardware do dispositivo. Adicionalmente, ao contrário dos trabalhos mencionados, a estratégia proposta não apenas realiza codificação, decodificação e transmissão de áudio e vídeo, mas também faz essa transmissão em tempo real, permitindo uma comunicação interativa por vídeo.

7. TRABALHOS FUTUROS

O objetivo do grupo para o futuro é desenvolver um framework que encapsule todos os pontos complexos da implementação desta técnica. O framework permitirá que seja possível implementar aplicativos de interação por vídeo com a estratégia acima de modo simples e rápido, e ao mesmo tempo oferecer flexibilidade ao programador com relação a escolha de codecs e protocolo de transmissão de dados. Também será possível utilizar o framework para reprodução de arquivos de vídeo locais que estão codificados com codecs ou formatos não suportados em hardware pelo Android, visto que o FFmpeg suporta, em software, um grande variedade de codecs.

8. REFERÊNCIAS

[1] Bigbluebutton. Disponível em http://bigbluebutton.org/. Acessado em maio de 2011.

[2] Darwin streaming server. Disponível em http://dss.macosforge.org/. Acessado em maio de 2011.

[3] Ffmpeg. Disponível em http://www.ffmpeg.org/. Acessado em maio de 2011.

[4] Gsmarena.com - gsm phone reviews, news, opinions, votes, manuals and more... Disponível em http://www.gsmarena.com/. Acessado em maio de 2011.

[5] Gt-mconf: Sistema de multiconferência para acesso interoperável web e móveis. Disponível em http://www.inf.ufrgs.br/prav/gtmconf. Acessado em maio de 2011.

[6] Guia de referência para desenvolvedores do android. Disponível em http://developer.android.com/reference/packages.html. Acessado em maio de 2011.

[7] Opengl es: The standard for embedded accelerated 3d graphics. Disponível em http://www.khronos.org/opengles/. Acessado em maio de 2011.

[8] Real-Time Messaging Protocol (RTMP) Specification. Disponível em `http://www.adobe.com/devnet/rtmp.html`. Acessado em maio de 2011.

[9] Sipdroid: Free sip/voip client for android. Disponível em http://sipdroid.org. Acessado em maio de 2011.

[10] Videolan media player. Disponível em http://www.videolan.org/vlc/. Acessado em maio de 2011.

[11] F. Ableson, C. Collins, and R. Sen. *Unlocking Android*. Manning Publications Co., Greenwich, CT, USA, 2009.

[12] X. Fu, X. Wu, M. Song, and M. Chen. Research on audio/video codec based on android. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on*, pages 1–4. IEEE, 2010.

[13] S. Y. Hashimi and S. Komatineni. *Pro Android*. Apress, Berkely, CA, USA, 1st edition, 2009.

[14] Q. Huynh-Thu and M. Ghanbari. Temporal aspect of perceived quality in mobile video broadcasting. *Broadcasting, IEEE Transactions on*, 54(3):641–651, 2008.

[15] D. Janssen. Implementierung und evaluierung von ip-video-streaming unter android. Master's thesis, Fachhochschule Köln University of Applied Sciences Cologne, Cologne, Germany, Mar. 2010.

[16] J. F. Kurose and K. W. Ross. *Computer networking - a top-down approach featuring the internet*. Addison-Wesley-Longman, 2001.

[17] V. Roesler, R. Husemann, and C. H. Costa. A new multimedia synchronous distance learning system: the IVA study case. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1765–1770, New York, NY, USA, 2009. ACM.