Visualizador de Panoramas

Hallison Oliveira da Paz

7 de março de 2013

Resumo

Este trabalho tem por finalidade apresentar as atividades desenvolvidas entre os meses de agosto de 2012 e janeiro de 2013 dentro do projeto Realidade Aumentada Móvel nos aspectos relativos ao Visualizador de Panoramas. Neste período, foi desenvolvido um aplicativo para dispositivos móveis sobre a plataforma Android, capaz de exibir imagens panorâmicas, utilizando os sensores de atitude do dispositivo como meio de visualização. Os sensores do dispositivo como acelerômetro, magnetômetro (bússola) e giroscópio são utilizados para que a visualização na tela do aparelho seja coerente com a movimentação (rotações) do usuário e orientação do dispositivo móvel.

Como parte da experiência desejada ao usuário, o aplicativo possui uma interface para leitura de QR Codes, capaz de utilizar a informação decodificada para requisitar e transferir da internet as imagens necessárias para utilização. Esta funcionalidade visa permitir que, em um ambiente previamente preparado para utilização, fossem utilizados QR codes para identificação do exato local em que o usuário se encontra, providenciando a visualização de imagens coerentes para uma experiência de realidade aumentada.

O aplicativo encontra-se em sua primeira versão, com todos os requisitos funcionais relacionados à visualização de imagens panorâmicas integrados. É necessário aprimorar....

Objetivo

Dentro do projeto Realidade Aumentada Móvel, almeja-se construir um aplicativo para dispositivos móveis, sobre as plataformas Android e iOS, que seja capaz de exibir imagens panorâmicas, utilizando os sensores de atitude do dispositivo como meio de visualização. Pretende-se exibir na tela do dispositivo uma imagem 360 graus do ambiente em que o usuário se encontra, que pode ser modificada para prover uma experiência de realidade aumentada. Os sensores do dispositivo como acelerômetro, magnetômetro (bússola) e giroscópio são utilizados para que a visualização na tela do aparelho seja coerente com a movimentação (rotações) do usuário e orientação do dispositivo móvel.

Para viabilizar esta experiência, o aplicativo deve, ainda, possuir um leitor de QR codes e interação com um banco de dados de imagens em um servidor na internet. Estas duas funcionalidades permitiriam que, em um ambiente previamente preparado para utilização, fossem utilizados QR codes para identificação do exato local em que o usuário se encontra e busca e visualização da imagem correta. Isto porque os meios de localização dos dispositivos móveis ainda não são eficazes ao nível que desejamos (descobrir em que cômodo de uma casa o usuário se econtra, por exemplo)

1 INTRODUÇÃO

Atualmente, o aplicativo está em sua primeira versão utilizável, possuindo um menu simples para seleção de imagens e permitindo a visualização de imagens panorâmicas de baixa resolução (1024 x 512 pixels). O menu inicial consta das opções "Web", "Local", "QRCode", "Configurações" e "Sair". A opção Web faz uma requisição a um servidor para que seja exibida uma lista de panoramas disponíveis on line. Para selecionar uma imagem, basta tocar sobre seu nome, que o aplicativo fará download da imagem e a armazenará em uma pasta de cache na memória secundária do dispositivo. Após isso, a imagem selecionada será carregada para exibição. Caso a requisição falhe, é exibida uma mensagem de erro ao usuário e nenhuma ação é efetuada.

A opção "Local" exibe o nome das imagens presentes na pasta "panoramas", que é suposta estar

na raiz do armazenamento do dispositivo. A pasta de cache é criada dentro da pasta "panoramas". Da mesma forma que na opção "Web", a imagem selecionada é carregada para exibição.

As opções "Configurações" e "QRCode" ainda não foram implementadas. Pretende-se utilizar QR codes para fazer uma seleção automática do panorama a ser exibido, de acordo com o local onde o usuário se encontra. Eventualmente, os QR codes podem ser utilizados para prover pequenas quantidades de informações necessárias para efetuar correções na visualização dos panoramas. Por último, a opção "Sair", finaliza a aplicação.

1.1 METODOLOGIA

O aplicativo foi testado em 3 dispositivos diferentes: um tablet da Nvidia, não comercial, específico para uso de desenvolvedores e testes de novas tecnologias, que utiliza a versão 4.03 do sistema Android; um tablet Samsung Galaxy S com versão do Android 2.3.3; e um tablet Sony Xperia S com versão 4.03 do Android. Neste estágio, a utilização dos dados dos sensores ainda é muito dependente da qualidade do hardware.

2 VISUALIZAÇÃO

O problema da visualização consiste, basicamente, em pegar uma imagem panorâmica e fazer com que seja exibida na tela do dispositivo apenas uma pequena parte desta imagem, compatível com o campo de visão humano e coerente com o que a maioria das câmeras desses dispositivos nos permitiria observar. Além disso, o trecho da imagem a ser exibido deve ser determinado pela orientação do aparelho no espaço, sendo sempre alterado pelos movimentos do usuário. Ou seja, se segurarmos o dispositivo tentando observar o teto através dele, em sua tela deve ser exibido o teto da imagem; se fizermos uma rotação em 360^{o} segurando o dispositivo, a imagem exibida deve corresponder da mesma forma.

É importante observar que as imagens panorâmicas possuem um único ponto de observação, chamado de centro de projeção. Devido a isso, a cena será sempre observada do ponto em que a imagem foi fotografada. Este trabalho utiliza panoramas no formato equirretangular, muito utilizado pelos profissionais de fotografia e facilmente obtido tanto a partir de composição de várias imagens, quanto pela utilização de câmeras omnidirecionais.

2.1 RENDERING DA CENA

As imagens no formato equiretangular possuem uma parametrização natural associada às coordenadas de latitude e longitude em uma esfera. A ideia base da visualização é, portanto, criar uma malha poligonal de uma esfera, em que as coordenadas de posição dos vértices no espaço cartesiano R^3 são dadas pelas equações 1.1, 1.2 e 1.3 em que $0 \leq \Theta \leq$ PI e $0 \leq \varphi \leq$ 2PI.

$$x = Raio\, sen(\Theta)\, sen(\varphi) \tag{2.1}$$

$$y = Raio\, cos(\Theta) \tag{2.2}$$

$$z = Raio\, sen(\Theta)\, cos(\varphi) \tag{2.3}$$

Desta forma, os vértices obtidos representam as intersecções entre os paralelos e meridianos que desejamos representar na esfera. De posse da malha poligonal, apenas precisamos definir a imagem que queremos visualizar como uma textura a ser aplicada na esfera e visualizada a partir de seu interior. As coordenadas de textura de cada vértice são valores entre 0 e 1 calculados utilizando-se as equações 1.4 e 1.5.

$$u = \varphi/2PI \tag{2.4}$$

$$v = \Theta/PI \tag{2.5}$$

Com estas coordenadas, conseguimos um meio de mapear pontos da imagem em pontos da esfera. Com uma escolha razoável de pontos, pode-se completar a visualização por interpolação. Após este mapeamento, teremos o centro de projeção da imagem exatamente no centro da esfera.

2.2 IMPLEMENTAÇÃO

A visualização foi implementada em OpenGL ES 2.0. Por questões de desempenho, optou-se por fazer acesso nativo ao hardware do dispositivo, implementando funções em C++ que interagem com os trechos em Java através da interface JNI (Java Native Interface).

Alguns detalhes devem ser observados para a correta visualização. O primeiro é que temos o encontro de todos os meridianos da esfera nos polos. Isso faz com que os vertices dos paralelos que passariam por essas regiões se degenerem em um único ponto em cada polo. Para evitar este fenômeno e manter uma forma padronizada de associar os vértices (não criar casos especiais), o ângulo Θ tem seu domínio reduzido iniciando a partir de um valor muito pequeno, porém próximo de 0 (zero) e terminando em um valor muito próximo, porém inferior a PI. O segundo detalhe a ser observado, é a necessidade de unir os pontos do início e do final da textura, para que possamos ter uma visualização realmente em 360 graus. Para resolver este problema, ao percorrer cada paralelo, o ponto final possui a mesma coordenada de posição do ponto inicial, porém coordenadas de textura diferente.

Por último, implementa-se uma câmera que permita a visualização do interior da esfera a partir de seu centro, para que o usuário (observador) tenha o ponto de vista da câmera que fotografou a cena. Portanto, na visualização de um panorama são considerados apenas os movimentos de rotação, visto que uma translação requiriria uma outra imagem, fotografada a partir do ponto transladado.

3 SENSORES E ORIENTAÇÃO DO DISPOSITIVO

Nosso objetivo nesta etapa é obter uma matriz que represente a transformação de um vetor no sistema de coordenadas global para um vetor no sistema de coordenadas do dispositivo.

Para o sistema Android, o sistema de coordenadas do dispositivo apresenta duas convenções: para os dispositivos com telas menores como os celulares, a direção Y está alinhada com a maior extensão do aparelho, a direção Z é ortogonal ao plano da tela do dispositivo e, consequentemente, a direção X fica no mesmo plano da tela, ortogonal à direção Y; já para os dispositivos com telas maiores como um *tablet*, a direção Y permanece no plano da tela do aparelho, porém é ortogonal à maior extensão, a direção Z permanece ortogonal, deixando a direção X também determinada. Essa diferença decorre da orientação em que é mais natural para o usuário utilizar cada aparelho.

3.1 IMPLEMENTAÇÃO

A API dos sistema Android facilita o acesso aos dados de diversos sensores presentes no aparelho. Podemos obter os dados puros de cada sensor individualmente ou o resultado de um algoritmo de fusão de dados implementado por cada fabricante do *hardware* que utiliza o sistema. É cada vez mais comum que os aplicativos tentem descobrir a orientação do aparelho em relação a um sistema de coordenadas global para que possam executar determinadas funções. Essa necessidade tornou o desenvolvimento desta etapa mais eficiente e eficaz para este aplicativo, na medida em que os fabricantes de hardware têm implementado métodos que retornam o resultado da fusão de dados entre os sensores disponíveis, com boa precisão.

Nosso objetivo é obter uma matriz de rotação, baseada na orientação do dispositivo, que possa ser utilizada em nosso código OpenGL como fator extrínsico da câmera. Foram implementadas duas abordagens para isto: a primeira, utiliza os dados puros do acelerômetro e do magnetômetro com um filtro passa-baixa e, a partir dos dados corrigidos, extrai a matriz de rotação; a segunda, utiliza um resultado de fusão de dados entre acelerômetro, bússola e giroscópio, que resulta em um vetor, cujas componentes são utilizadas (sem nenhum filtro) para extrair uma matriz de rotação.

4 RESULTADOS

A segunda abordagem para extração da matriz de rotação tem apresentado resultados melhores que a primeira. No tablet da Nvidia, os resultados foram muito bons, dentro do que se espera como padrão para uma experiência de qualidade com o aplicativo. No tablet da Sony, o resultado foi bom, porém há trepidações e outras inconstâncias na visualização que a deixam longe do ideal; acredita-se, no entanto, que esses problemas podem ser resolvidos com um tratamento adequado dos dados (por software). O tablet da Samsung, por não possuir giroscópio, apresentou atrasos, saltos e outras inconsistências que prejudicam a experiência do aplicativo. Parte dos probelmas de visualização observados neste último aparelho, podem estar relacionados também à implementação do algoritmo de fusão de dados, pois a versão do sistema utilizada é bem mais antiga em relação à dos outros dispositivos de teste.

APÊNDICE

6 NOÇÕES DE OPENGL

Esta seção descreve alguns conceitos básicos de OpenGL, mais especificamente a versão OpenGL ES 2.0, que foi utilizada neste trabalho. É importante que o leitor tenha algum conhecimento de conceitos de computação gráfica.

OpenGL é uma API (Aplication Programming Interface) utilizada na construção de aplicações gráficas 3D em desktop, sendo um padrão adotado nos sistemas Linux, Mac OS e Windows. Vários jogos como os da série Doom e Quake, além de aplicações de CAD (Computer Aided Design) utilizam esta API.

As primeiras versões do OpenGL, referenciadas como 1.x (para 1.0, 1.1, etc) abstraiam para o programador algumas etapas do pipeline de renderização da cena. Isto facilitava a programação, porém dava menos controle ao programador sobre alguns itens da cena. A partir da versão 2.0, o OpenGL introduziu o conceito de *shaders*, permitindo que algumas etapas fossem programadas, dando mais controle ao modo em que os elementos gráficos são manipulados antes de serem renderizados, ao custo do aumento da complexidade do código.

6.1 OPENGL ES

O grupo Khronos desenvolveu um padrão simplificado da API OpenGL denominado OpenGL ES (Embbeded System) voltado para o uso em dispositivos portáteis ou sistemas embarcados. A OpenGL ES possui algumas simplificações, em relação ao padrão comum, para melhor se enquadrar nas limitações de hardware dos dispositivos fim. Atualmente, as versões mais utilizadas do OpenGL ES são a 1.1 e a 2.0. A primeira é derivada da versão 1.5 da OpenGL, implementando, portanto, um pipeline com funções fixas. Já a versão ES 2.0 é derivada da versão 2.0 da OpenGL, com uso de Vertex Shaders e Fragment Shaders, recursos que permitem programar operações sobre vértices e sobre fragmentos, respectivamente.