UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

2009.1

# IMPLEMENTAÇÃO DO RANSAC COM ALGORITMO DE EIGHT-POINT EM GPU

## PROPOSTA DE TRABALHO DE GRADUAÇÃO

| | | |
|---|---|---|
| **Aluno** | André Vitor de Almeida Palhares | {avap@cin.ufpe.br} |
| **Orientador** | Djamel Fawzi Hadj Sadok | {jamel@cin.ufpe.br} |

01 de Março de 2009

# Índice

# 1. Contexto

## 1.1. Introdução

O problema da reconstrução em três dimensões de cenas a partir de duas imagens obtidas de diferentes ângulos pode ser tratado de diversas formas. Uma das suas etapas consiste em recuperar a matriz fundamental que leva os pontos de uma imagem aos seus correspondentes na outra. Uma das maneiras de encontrar a matriz fundamental é o algoritmo de *eight-point*, que requer um número mínimo de correspondências entre oito pontos. Porém, na busca de uma matriz fundamental que oferece uma transformação com boa precisão, é necessário o uso de diversas correspondências entre pontos, o que aumenta o grau de complexidade das computações que devem ser realizada.

As sessões que seguem introduzem melhor o problema que será abordado durante o desenvolvimento deste trabalho de graduação. Na próxima seção, mais idéias sobre o algoritmo de *eight-point* são introduzidas, e sua relação com o RANSAC, é delineada logo após. Depois será falado um pouco sobre GPU, que garantirá eficiência do algoritmo em questão e também exigirá uma nova maneira da organização do algoritmo em si. Por último, é dada uma introdução a CUDA, o ambiente de software no qual será desenvolvido o algoritmo.

## 1.2. Algoritmo de *Eight-point*

O *Eight-point* normalizado é um algoritmo utilizado em visão computacional na busca pela matriz essencial (ou pela matriz fundamental), uma ferramenta básica na análise de cenas que foram obtidas através de duas câmeras, e que é usada em etapas da reconstrução 3D da cena.

O *Eight-point* foi proposto em [2], um artigo que marcou a sua época. Também, foi descoberto que a normalização dos pontos das imagens antes da execução do algoritmo trás uma melhoria na precisão da matriz fundamental obtida.

Dadas duas imagens de uma mesma cena, obtidas a partir de duas câmeras, podemos definir um conjunto de pontos que se correspondem um a um, de uma imagem a outra. Dado um conjunto com oito ou mais correspondências entre pontos, podemos gerar a matriz fundamental fazendo-se uso do algoritmo de *Eight-point*.

Geralmente, na reconstrução de cenas, algum processo automatizado de localização de características (*features*) é utilizado para construção das correspondências entre as imagens. Assim, o uso do algoritmo de *Eight-point* pode perder precisão devido a erros inerentes cometidos por tais processos automatizados. Assim, é necessária a geração de diversas matrizes fundamentais a partir de diversas combinações das correspondências, e é feita então a escolha da melhor entre elas, utilizando-se alguma função de custo (soma dos quadrados dos erros da distância, por exemplo).

Como é computacionalmente custosa a geração de todas as matrizes fundamentais possíveis para uma cena, faz-se necessário o uso de um estimador que seja confiável.

## 1.3. RANSAC

O RANSAC (RANdom SAmple Consensus) é um estimador de parâmetros de modelos matemáticos robusto, inicialmente proposto em [1] que gera modelos a partir de um conjunto mínimo de dados e os avalia, utilizando o restante dos dados. A partir destes diferentes modelos e utilizando-se alguma condição de parada, se obtém o modelo mais representativo entre os avaliados.

O RANSAC é um estimador muito interessante para aplicações de análise de imagem automatizada[1]. Isto acontece porque ele é um estimador robusto, sendo assim, erros devido a uma coleta automática de características de uma imagem podem ser suavizados.

O *Eight-point* é utilizado em conjunção com o RANSAC para gerar vários modelos (matrizes fundamentais) que se adéquam a cena em questão e fazer a decisão de qual a melhor matriz, utilizando-se alguma condição de parada para o RANSAC (erro pequeno o bastante, número máximo de iterações, etc.).

## 1.4. GPU

GPU (Graphics Processing Unit, ou Unidade de Processamento Gráfico) [4], um tipo de microprocessador especializado no processamento de gráficos. Este dispositivo é encontrado em videogames, computadores pessoais e estações de trabalho, e pode estar situado na placa de vídeo ou integrado diretamente à placa mãe.

As GPUs modernas são responsáveis pela renderização e processamento de gráficos de um computador. A sua estrutura extremamente paralela permite que diversas operações sejam realizadas concorrentemente, realizando assim tarefas com grande eficiência, as quais seriam extremamente custosas para uma CPU (Central Processing Unit ou Unidade Central de Processamento) comum.

Com o advento de GPGPUs (General-Purpose computation on GPUs, ou computação de propósito geral em GPUs) [5], não apenas operações de origem estritamente gráficas são realizadas em GPUs. GPGPUs são capazes de realizar diversas tarefas dos mais variados tipos de aplicações que possam ser solucionadas através de algoritmos paralelizáveis com extrema eficiência.

## 1.5. CUDA

Um importante desafio no contexto de GPUs é o de como ocorrerá o desenvolvimento de softwares para tais tipos de sistemas. Surge assim a necessidade de se criar um ambiente no qual o programador possa acessar dos mais diversos tipos de ferramentas desses novos tipos de processadores, bem como criar aplicações paralelas de um modo transparente e escalável, independente do número de processadores.

CUDA (Compute Unified Device Architecture) [6] é um ambiente de software com um modelo de programação paralela que foi projetado para lidar com este problema. É baseado na linguagem C, facilitando assim o aprendizado por parte de programadores que tem conhecimentos nessa linguagem.

No ambiente CUDA, existem três conceitos chaves: uma hierarquia de grupos de threads; memórias compartilhadas; e barreiras de sincronização. Estes conceitos são disponibilizados ao programador a partir de um conjunto de extensões à linguagem C, e servem para gerenciar os aspectos de paralelização da plataforma.

## 2. Objetivos

Este trabalho de graduação tem por objetivo desenvolver uma pesquisa que dê base a uma implementação do algoritmo de *eight-point*, utilizando-se do RANSAC para a obtenção de um resultado robusto. A implementação será realizada em GPU, utilizando-se CUDA, buscando por partes dos algoritmos que podem ser paralelizadas ao máximo. Aliado ao alto grau de eficiência das GPUs, obtém-se um maior desempenho com relação a suas implementações genéricas.

Para tal, um cuidadoso estudo sobre as etapas dos algoritmos que possam ser paralelizados deve ser realizado. Geralmente, etapas em que diversas computações são essencialmente independentes umas das outras são de fácil paralelização.

Como parte do estudo realizado, também serão reunidas informações mais gerais sobre a recuperação da matriz fundamental, bem como de suas relações com outras etapas da reconstrução 3D e também das suas aplicações na área de visão computacional.

Como subprodutos não menos importantes deste trabalho, serão criadas também em GPU algumas funções mais gerais importantes que são ferramentas fundamentais para os algoritmos citados acima. Tais funções têm grande importância não somente na área de visão computacional. Como exemplo, temos o SVD(Singular Value Decomposition), necessário em algumas etapas do *eight-point*, e que é utilizado em diversas aplicações na área de processamento de sinais e estatística.

Além do desenvolvimento do projeto em GPU, serão realizadas comparações com implementações genéricas já existentes, não apenas dos algoritmos em geral, mas também de suas sub-funções importantes citadas no parágrafo anterior. Entre as principais bibliotecas que já possuem os algoritmos, temos a VXL [7], a qual será fundamentalmente usada como a base para a comparação.

## 3. Cronograma

O cronograma abaixo possui algumas datas que estabelecem quando as atividades principais do processo de desenvolvimento do trabalho de graduação serão realizadas.

| **ATIVIDADES** | **MARÇO** | | | | **ABRIL** | | | | **MAIO** | | | | **JUNHO** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Levantamento do material bibliográfico | | | | | | | | | | | | | | | | |
| Estudo sobre o algoritmo de *eight-point* | | | | | | | | | | | | | | | | |
| Estudo do RANSAC | | | | | | | | | | | | | | | | |
| Desenvolvimento do algoritmo em GPU | | | | | | | | | | | | | | | | |
| Comparações com outras implementações | | | | | | | | | | | | | | | | |
| Elaboração do relatório | | | | | | | | | | | | | | | | |
| Preparação da apresentação | | | | | | | | | | | | | | | | |

# 4. Referências


[1] Martin A. Fischler and Robert C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". Comm. of the ACM 24: 381–395.

[2] R. Hartley, “In defense of the eight-point algorithm,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 19, no. 6, pp. 580–593,1997.

[3] W. Chojnacki, M.J. Brooks, A. van den Hengel, D. Gawley, Revisiting Hartley's normalised eight-point algorithm. IEEE Trans. Pattern Analysis Machine Intelligence, 25, 9, 2003, 1172-1177.

[4] Luebke, D., Humphreys, G. How GPUs Work. Computer , vol.40, no.2, pp.96-100, Feb. 2007.

[5] General-Purpose GPU. URL: http://www.gpgpu.org. Visitado em Março, 2009.

[6] Compute Unified Device Architecture. URL: http://www.nvidia.com/cuda. Visitado em Março, 2009.

[7] VXL. URL: http://vxl.sourceforge.net/. Visitado em Março, 2009.


# 5. Possíveis Avaliadores

Veronica Teichrieb

## 6. Assinaturas

Djamel Fawzi Hadj Sadok
**Orientador**

André Vitor de Almeida Palhares
**Aluno**