Laboratório 1 - Processamento de Imagens

Arthur Reichert Costa
Universidade de Brasília
Campus Darcy Ribeiro, Asa Norte
arreico@gmail.com

Natalia Oliveira Borges
Universidade de Brasília
Campus Darcy Ribeiro, Asa Norte
natioliveira97@hotmail.com

1. Objetivos

Esse projeto tem por objetivo usar o processamento de imagens para se familiarizar com a arquitetura Mips e sua linguagem em assembly. Além disso, também pretendemos fazer uma análise de desempenho para avaliar se a sulução foi boa e como é possível melhorá-la.

2. Introdução

A arquitetura MIPS foi desenvolvida em 1981 pelo pesquisador da Universidade de Stanford John Hennessy. Ela é baseada na arquitetura RISC (Reduced Instruction Set Computer) e tinha por objetivo aumentar o desempenho usando pipelines profundas para instruções.

A CPU MIPS é composta por 32 resgistradores de 32 bits e possui alguns de co-processadores que executam tarefas auxiliares, como operações com números em ponto flutuante. Seguindo a arquitetura Von Neumann, as instruções e os dados são armazenados na memória.

As instruções da linguagem assembly MIPS possuem 32 bits e podem ser de três tipos R, I, J. Instruções do tipo R operam o conteúdo de dois registradores e armazenam o resultado em um terceiro registrador. Instruções do tipo I operam entre um registrador e um operando imediato e armazenam o resultado em um registrador. Já instruções do tipo J são usadas para saltos no código e recebem o endereço para qual irá saltar.

Para executar esse projeto foi utilizada uma IDE para programação em MIPS assembly chamada MARS. Essa IDE foi desenvolvida para propósitos educacionais e ajuda a visualizar o que acontece durante a execução do código. Além disso possui algumas aplicações, como BitMap Display e Instruction Statistics, que serão abordadas posteriormente nesse relatório.

Para avaliar todo o aprendizado que tivemos em sala de aula, nesse projeto trabalharemos com processamento de imagens. Os três requisitos desse projeto são: leitura e escrita de imagens utilizando o formato BitMap 24 bits, aplicação de filtros na imagem (efeito de borramento, extração de bordas e segmentação por limiar) e, por fim, análise de desempenho dos métodos utilizados.

3. Materiais e Métodos

Para implementação desse projeto utilizamos o simulador MARS versão 4.5. Intalados em um MacBook Pro 13"com processador 2,7 GHz Intel Core i5 8GB RAM,com sistema operacional macOS High Sierra versão 10.13.6 e em um ASUS Rog Zephyrus m(gm501) 15.6"Intel Core i7-8750H, placa de vídeo NVidia GTX 1060 e 16GB RAM, com sistema operacional linux Mint 19 Cinnamon.

3.1. Requisito 1

3.1.1 Leitura

Devemos ser capazes de ler e escrever uma imagem no formato Bitmap 24 bits, e para isso devemos entender como esse formato é organizado.

O formato Bitmap 24 bit possui 54bytes de informações de cabeçalho seguido por n bytes de dados determinados pelo tamanho da imagem. Para cada pixel da imagem são armazenados 3 bytes referentes aos canais RGB da imagem

Para visualizar a imagem no MARS devemos usar a ferramenta Bitmap Display. Para essa ferramenta, um pixel é uma word de 4 bytes em que os bits 0-7 são referentes ao canal azul, os bits 8-15 são referentes ao canal verde e os bits 16-23 são referentes ao canal vermelho.

Dessa forma, temos que "transfomar"cada pixel de 3 bytes em uma word de 4 bytes preenchendo os 24 bits menos significativos.

Problemas enfrentados: Quando implementamos uma função que transfomava os 3bytes em 4 e copiava os pixels na ordem de leitura no endereço do Bitmap, a imagem ficava de ponta cabeça. E quando copiávamos na ordem inversa ficava, a imagem ficava espelhada.

Solução:

Para solucionar esse problema, tivemos que ler linha por linha do arquivo, e escrever no Bitmap Display começando pela última linha, para isso usamos os dados recebidos do header da imagem para manipular ponteiros. Dessa forma, a imagem é carregada de baixo pra cima no Bitmap Display.

Escolhemos como endereço de memória do Bitmap Display a memória heap 0x10040000.

3.1.2 Escrita

Para escrever a imagem processada em um arquivo, o programa pergunta ao usuário o nome que quer salvar, abre o arquivo e escreve o header. Em seguida e realiza o procedimento inverso da leitura, transformando os 4bytes em 3 e escrevendo linha por linha desde o fim do endereço do Biymap Display.

3.2. Requisito 2

3.2.1 Efeito de Borramento

3.3. Noção Geral

O processo de borramento se baseia na aplicação de uma convolução com um kernel que, para cada pixel, obtém o valor médio na vizinhança do mesmo, e atribui esse valor para o pixel central (centralizado em relação à vizinhança). Assim, pixels que possuem valores discrepantes em relação ao seu contorno são suavizados, ou seja, partes de frequência mais alta (maior variação em intervalo curto) são filtradas, por isso se fala que se trata de um filtro passa-baixa. De modo geral isso permite remover ruídos de elevadas frequências na imagem, mas em contrapartida diminui o contraste de alguns detalhes na imagem, que se tornam menos visíveis.

3.4. Convolução

O processo de convolução utilizado na imagem trata-se de uma aproximação de uma integral de convolução de funções no plano em um domínio contínuo, só que no domínio discreto. Consideramos a imagem como uma função e o núcleo (uma matriz quadrada que iremos aplicar) como outra função; as posições dos elementos na imagem e no núcleo são os domínios discretizados de ambas as funções (com um passo discreto fixo), e os valores dos elementos de ambas (núcleo e imagem) são os valores que as funções assumem. A convolução no domínio contínuo, para cada ponto (x,y) no domínio, é dada por

$$f(x,y) * h(x,y) = \iint_A f(x,y)h(x-\alpha, y-\beta)\, d\alpha\, d\beta \tag{1}$$

onde A é um plano que se estende ao longo do infinito. No entanto, assumindo que a imagem assume valor 0 além de uma determinada região retangular, o resultado da convolução será sempre 0 para pixels além dessa região. Discretizando ambos os domínios da imagem (f) e da função h, e assumindo que h também é não-nula apenas em uma pequena região retangular, podemos representar o domínio de h pelas posições em uma matriz, e os valores que h assume nesse domínio através dos valores dessa matriz. Usando assim uma matriz para representar a função h e uma matriz para representar o domínio e os valores da imagem, seja

$$\begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix} \tag{2}$$

a matriz que representa a função h, onde a posição (x,y) corresponde a linha 1 e coluna 1 na matriz, o resultado da convolução da imagem (f) pelo núcleo no ponto (x,y) é dado por:

$f(x,y) * h(x,y) = (Af(x-1,y-1) + Bf(x,y-1) + Cf(x+1,y-1) + Df(x-1,y) + Ef(x,y) + Ff(x+1,y) + Gf(x-1,y+1) + Hf(x,y+1) + If(x+1,y+1))$

3.5. A média como Borramento

Como discutido anteriormente, o objetivo do borramento é aplicar um filtro passa baixa na imagem, de modo a eliminar outliers. Para isso, precisamos antes de mais nada obter uma soma dos valores de intensidade em cada canal dos pixels numa vizinhança. Para isso, basta aplicarmos convolução com um núcleo onde todos os elementos possuem valor 1. Como exemplo, seja

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{3}$$

o núcleo utilizado, obteremos a soma de todos os pixels numa vizinhança quadrada de lado 1 após aplicar a convolução. Feito isso, basta dividirmos o resultado pelo número de elementos do núcleo para obtermos a média. Assim, o resultado do borramento para um pixel de posição (x,y) na imagem é

$$\frac{f(x,y) * h(x,y)}{n^2} \tag{4}$$

onde n é a dimensão da matriz que representa o núcleo.

3.5.1 Extração de Bordas

O objetivo da extração de bordas é detectar contornos na imagem. Antes de mais nada, precisamos caracterizar regiões onde se encontra um contorno e/ou as vizinhanças de um contorno. Para isso, vamos considerar um contorno como sendo uma região onde para uma pequena variação

espacial de posição ocorre uma variação abrupta de intensidade dos pixels, para alguma direção no espaço. Relembrando o conceito de gradiente do cálculo 3, $\begin{pmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{pmatrix}$, percebemos que segundo a definição dada anteriormente de contorno, o módulo do gradiente da imagem como função no plano terá maximos locais em contornos. Assim, começamos a delimitar como deverão ser os métodos para obtenção de contornos na imagem.

3.6. Métodos baseados em gradiente

Conforme foi visto na seção anterior, pode-se notar que uma das formas de detectar bordas é identificar valores altos de magnitude do gradiente, onde a magnitude do gradiente é dada por

$$\sqrt{\frac{df}{dx}^2 + \frac{df}{dy}^2} \tag{5}$$

No entanto, para simplificar o problema, aproximaremos a magnitude do gradiente por

$$\left|\frac{df}{dx}\right| + \left|\frac{df}{dy}\right| \tag{6}$$

Considerando uma vizinhança retangular 3x3, podemos aproximar a expressão acima por:

$|[f(x-1,y-1) + f(x-1,y) + f(x-1,y+1) - f(x+1,y-1) - f(x+1,y) - f(x+1,y+1)]| + |[f(x-1,y-1) + f(x,y-1) + f(x+1,y-1) - f(x-1,y+1) - f(x,y+1) - f(x+1,y+1)]|,$

o que equivale a

$$f(x,y) * h1(x,y) + f(x,y) * h2(x,y) \tag{7}$$

onde

$$h1 = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \tag{8}$$

e

$$h2 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \tag{9}$$

h1 e h2 são conhecidos como operadores de Prewitt priorizando as variações centrais, chegamos em

$$h1 = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \tag{10}$$

e

$$h2 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{11}$$

conhecidos como operadores de sobel, que foram os utilizados neste trabalho.

Por último aplicamos a máscara de segmentação por limiar para realçar o contraste excolhido pelo usuário, é possível realçar tanto bordas baixas, pouco contraste, quanto bordas altas, muito contraste.

3.6.1 Segmentação por Limiar

A segmentação por limiar consiste em limitar os valores pemitidos para cada canal de cor RGB e aplicar uma máscara, tornando branco os pixels que satisfazem os critérios estabelecidos e preto os que não satisfazem.

Como as imagens são armazenadas no formato RGB para aplicar esse filtro precisamos de seis parâmetros escolhidos pelo usuários. Valor mínimo de R, G e B, e valor máximo de R, G e B.

O resultado da operação é então:

$Branco[R = 255, \quad G = 255, \quad B = 255] \quad se:$

$Rmin \leq R \leq Rmax \quad e$

$Gmin \leq G \leq Gmax \quad e$

$Bmin \leq B \leq Bmax$

Ou

$Preto[R = 0, \quad G = 0, \quad B = 0] \quad se:$

$R > Rmax \quad ou \quad R < Rmin \quad ou$

$G > Gmax \quad ou \quad G < Gmin \quad ou$

$B > Bmax \quad ou \quad B < Bmin$

Para implementação, primeiramente carregamos a imagem original no Bitmap Display no endereço heap 0x10040000 e chamamos a função de segmentação limiar.

A função de segmentação por limiar percorre toda a área de memória do Bitmap Display e carrega a word que representa cada pixel. Analisa os 3 primeiros bytes da word, que correnpondem aos canais RGB, e realiza a operação descrita. Se o resultado for branco a função carrega os 3 bytes de valor 255 usando sll(shift left logical) e add (adição) em um registrador. Se for preto atribui o valor 0 para o registrador. Em seguida carrega o conteúdo desse registrador como uma word para o Bitmap Display.

3.7. Requisito 3

O requisito 3 será analizado na seção 4

4. Resultados

4.1. Requisito 1

A imagem foi aberta e lida corretamente e no Bitmap Display

Figura 1. Resultado de abrir imagem e mostrar no Bitmap Display

As imagens processadas também foram escritas corretamente de volta para um arquivo .bmp.

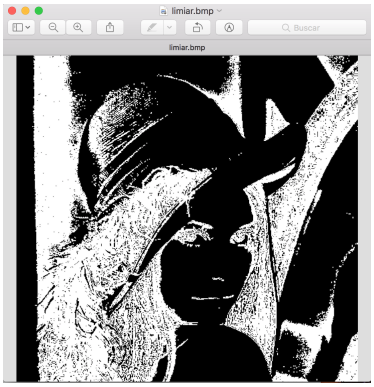

Figura 2. Resultado de escrever uma imagem processada de volta para um arquivo .bmp

4.2. Requisito 2

4.2.1 Efeito de Borramento

O efeito de borramento funcionou como esperado. Dependendo do tamanho do kernel escolhido pelo usuário a imagem fica mais ou menos embaçada.

Figura 3. Efeito de borramento com kernel 5x5

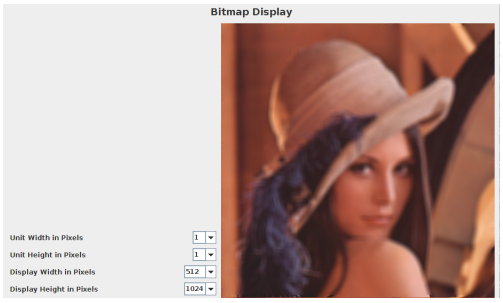

Figura 4. Efeito de borramento com kernel 11x11

4.2.2 Efeito de extração de bordas

O efeito de extração de bordas também funcionou como esperado. O programa realiza duas vezer um procedimento de convolução e pergunta ao usuário valores threshold para a máscara.

Para valores baixos de threshold a imagem detecta muitas bordas, já que pouco contraste já é considerado uma borda. Para valores mais altos de threshold a detecção é mais seletiva e mostra apenas as bordas mais evidentes.

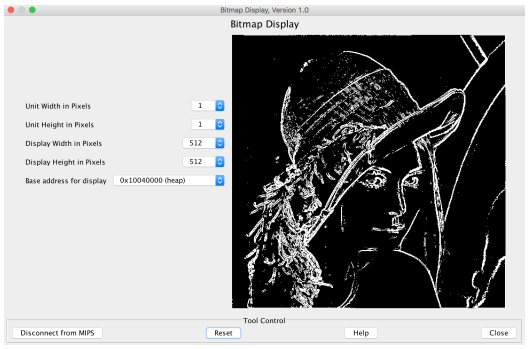

Figura 5. Resultado do detector de bordas para parâmetros de contraste min=15 e max=100

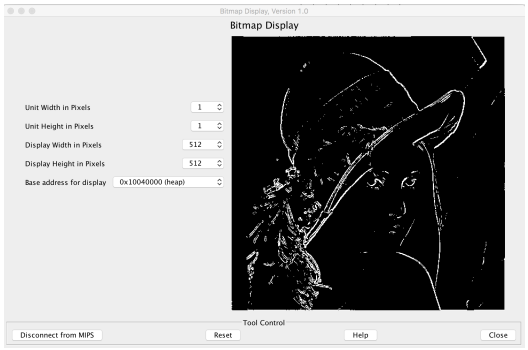

Figura 6. Resultado do detector de bordas para parâmetros de contraste min=30 e max=100

4.2.3 Efeito de segmentação por limiar

Nesse requisito, o usuário escolhe a faixa de cor de quer em cada canal, escolhendo seis parâmetros de comparação. Essa aplicação também funcionou como esperado.

Para pegar imagens de cor mais escura, devemos escolher valores baixos nos três canais.

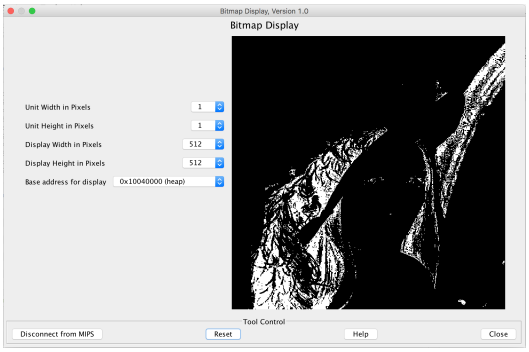

Figura 7. Resultado de segmentação por limiar para cores escuras Rmin = 0, Rmax = 50, Gmin = 0, Gmax = 50, Bmin = 0, Bmax = 50

Para pegar imagens de cor mais clara, devemos escolher valores altos nos três canais.

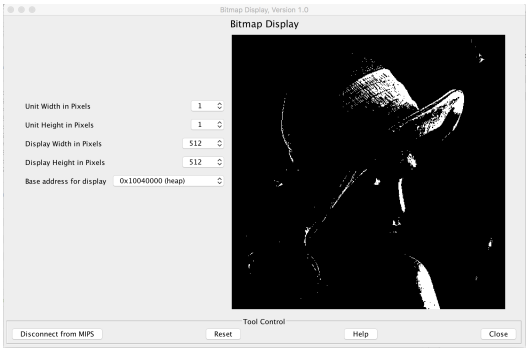

Figura 8. Resultado de segmentação por limiar para cores escuras Rmin = 150, Rmax = 255, Gmin = 150, Gmax = 255, Bmin = 150, Bmax = 255

4.3. Requisito 3

Para realizar esse procedimento utilizamos a ferramenta do MARS Instruction Statistics. Essa ferramenta mostra a porcentagem de instruções de ALU, jump, Branch, Memory e outras.

Essa ferramenta é útil para análise de desempenho de um programa, porém a sua execução deixa o programa extremamente lento. Uma tarefa que demora alguns segundos, como abrir a imagem, chega a demorar cerca de meia hora em paralelo com com a ferramenta.

- Abertura e leitura de imagem

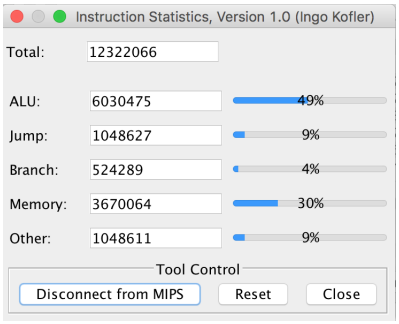

Figura 9. Resultado da estatística para abrir a imagem

Para abrir imagem 49% das operações foram lógicas e aritiméticas e 30% de acesso a memória. Essa estatística faz sentido, já que é necessário carregar cada pixel do arquivo, fazer operações lógicas para transformar os três bytes lidos em uma word de 4 bytes, e carregar novamente para o endereço do Bitmap. Como existe um duplo loop e algumas funções, 7% de jumps e 7% branchs.

Fazendo uma média por pixel, cada pixel exige 47 operações.

2. Escrita de imagem

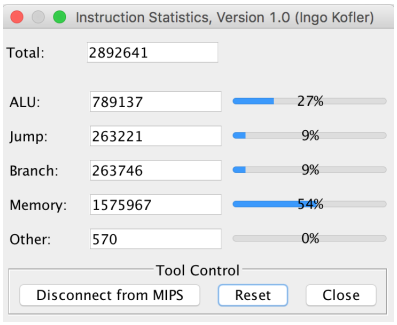

Figura 10. Resultado da estatística para escrever e salvar imagem

Para a escrita de imagens, a porcentagem maior foi para operações com memória, 54%.

Apesar de ser a operação do carregamento, a implementação dessa aplicação foi feita por um membro diferente da dupla, gerando resultados diferentes. Como também foi feita depois da função de leitura, já havia uma maior maturidade para implementação, evitando operações desnecessárias.

Para cada pixel, houve uma média de 11 operações. Cerca de 23% do total de operações para leitura da imagem.

3. Efeito de borramento

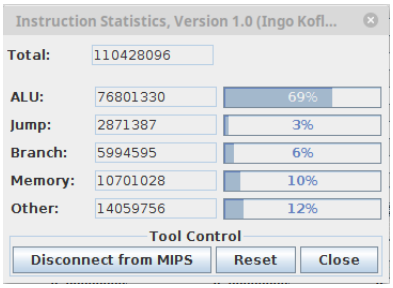

Figura 11. Resultado da estatística para efeito de borramento kernel 3x3

Para o efeito de borramento vimos que houve enorme número de intruções. Dentro dessa estatística estão incluidas as intruções para leitura da imagem, para a cópia da imagem, já que não podemos fazer as operações na imagem original, e para a aplicação da convolução. Resultando em uma média de 421 intruções por pixel.

O número de operações lógicas aritméticas é o mais expressivo, 69%. Esse número faz sentido já que a operação de convolução exige esse tipo de processamento.

4. Efeito de extração de bordas

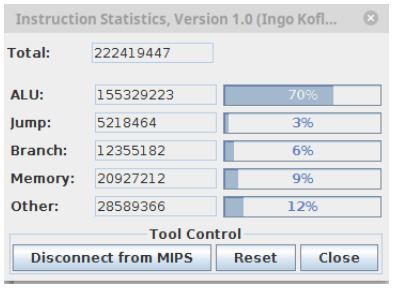

Figura 12. Resultado da estatística para extração de bordas

O número de instruções nessa aplicação foi o maoir, já que usamos duas convoluções para gerar melhores resultados e ainda aplicamos a segmentação por limiar. Gerando uma média de 848 operações por pixel.

O número de operações de lógica aritmética foi de 70%, semelhante ao do efeito de borramento, como também as outras estatísticas.

5. Segmentação por limiar

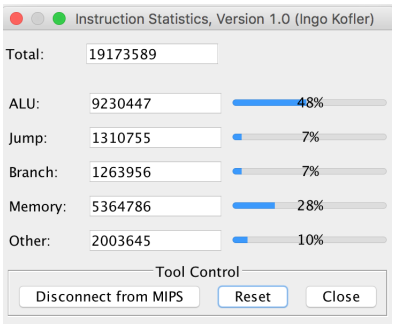

Figura 13. Resultado da estatística para segmentação por limiar

Nas estatisticas de segmentação por limiar estão também contabilizadas as estatísticas de leitura da imagem. Já que para aplicar o filtro é necessário carregar a imagem para a memória. Logo a contagem de intruções real para essa aplicação é 6851523, o que dá uma média de 26 operações por pixel. As proporções de intruções foram muito semelhates das de leitura de imagem.

5. Discussão e Conclusões

Pela análise das imagens obtidas, podemos concluir que o projeto foi bem implementado gerando bons resultados.

O efeito de borramente foi mais expressivo de acordo com o tamanho do kernel, porém quanto maior esse kernel, mais operações são necessárias e maior é o processamento.

O efeito de extração de bordas envolve duas convoluções, gerando um processamento considerável. Porém é constante e não aumenta de acordo com a entrada de threshold do usuário.

O efeito de segmentação por limiar foi o que teve o melhor desempenho, já que não envolve convolução. Também tem um processamento constante que independe dos parâmetros passados pelo usuário.

Vimos ao longo do projeto que houve um amadurecimento em relação ao conhecimento da linguagem e arquitetura MIPS. Principalmente com relação ao reconhecimento de erros. A característica da IDE de permitir parar a execução e verificar o valor de cada registrador ajudou bastante nesse quesito.

Com o decorrer da implementação também tentamos reduzir o número de instruções dentro de loops para aumentar o desempenho, como pode ser visto na implementação de escrita de imagens, que teoricamente exigiria o mesmo processamento da leitura, mas utilizou 23% do processamento.

Tentamos também reduzir o acesso a memória onde era possível. Como na aplicação de segmentação por limiar, em que a operação é feita em cima da matriz de entrada. Nas aplicações de borramento e extração de borda isso não foi possível, já que precisavamos da matriz original para realizar as operações.

Aprendemos muito com a realização desse projeto, tanto no quesito de processamento de imagens quanto no entendimento da arquitetura utilizada.

Referências

[1] P. F. A. Faria. Disciplina de processamento de imagens. Disponível em http://www.ic.unicamp.br/~ffaria/pi2s2015/class10/aula_segmentacao1.pdf.

[2] A. W. R. Fisher, S. Perkins and E. Wolfart. Sobel edge detector. Disponível em https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm.

[3] J. Roger L. Easton. Fundamentals of digital image processing. Disponível em https://www.cis.rit.edu/class/simg361/Notes_11222010.pdf.