Image Processing

Why Do We Process Images?

- Enhancement and restoration
  - Remove artifacts and scratches from an old photo/movie
  - Improve contrast and correct blurred images
- Transmission and storage
  - Images and Video can be more effectively transmitted and stored
- Information analysis and automated recognition
  - Recognizing terrorists
- Evidence
  - Careful image manipulation can reveal information not present
  - Detect image tampering
- Security and rights protection
  - Encryption and watermarking preventing illegal content manipulation

Some examples

Compression

- Color image of 600x800 pixels
  - Without compression
    - 600*800 * 24 bits/pixel
      = 11 520K bits = 1.44M bytes
  - After JPEG compression (popularly used on web)
    - only 89K bytes
    - compression ratio ~ 16:1
- Movie
  - 720x480 per frame, 30 frames/sec, 24 bits/pixel
  - Raw video ~ 243M bits/sec
  - DVD ~ about 5M bits/sec
  - Compression ratio ~ 48:1

“Library of Congress” by M.Wu
(600x800)

Denoising

From X.Li
http://www.ee.princeton.edu/~lixin/denoising.htm

Deblurring

Blurred & noisy image

Restored image

From Mathworks

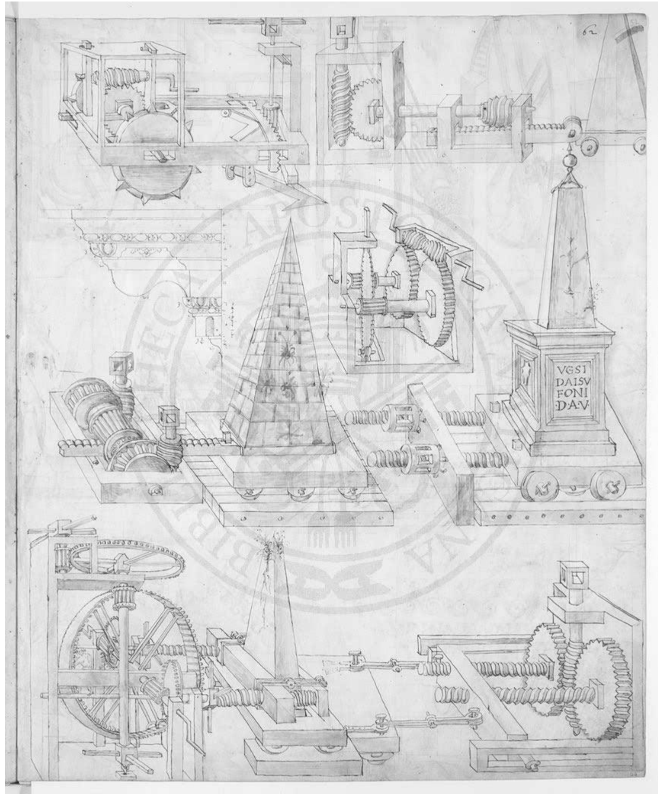
Visual Mosaicing

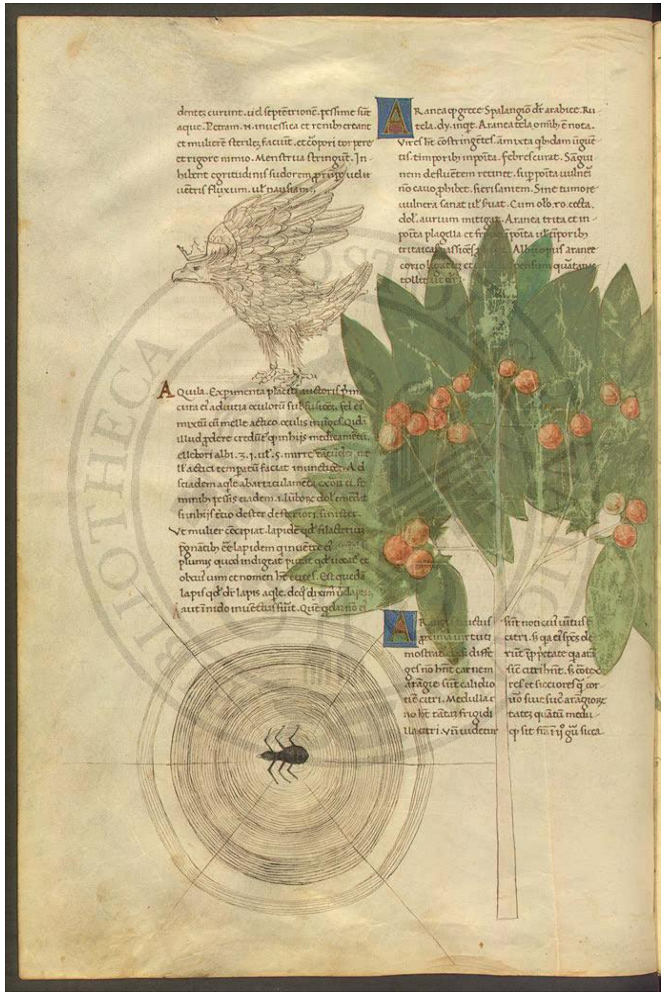
- Stitch photos together without thread or scotch tape

R.Radke – IEEE PRMI journal paper draft 5/01

Visible Digital Watermarks

- from IBM Watson web page
  “Vatican Digital Library”





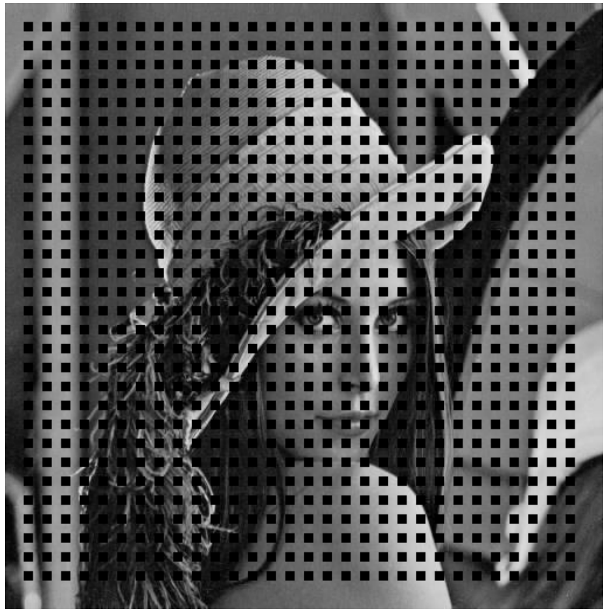
Invisible Watermark

- 1st & 30th Mpeg4.5Mbps frame of original, marked, and their luminance difference
- human visual model for imperceptibility: protect smooth areas and sharp edges

Error Concealment

(a) original Lenna image

(b) corrupted Lenna image



(c) concealed Lenna image

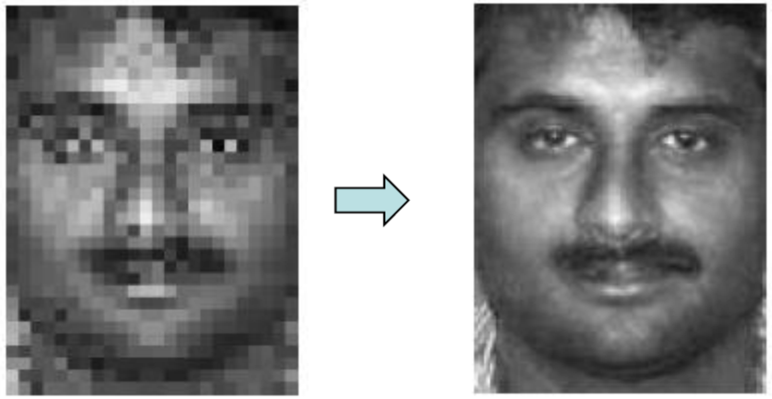
25% blocks in a checkerboard pattern are corrupted

corrupted blocks are concealed via edge-directed interpolation

Examples were generated using the source codes provided by W.Zeng.

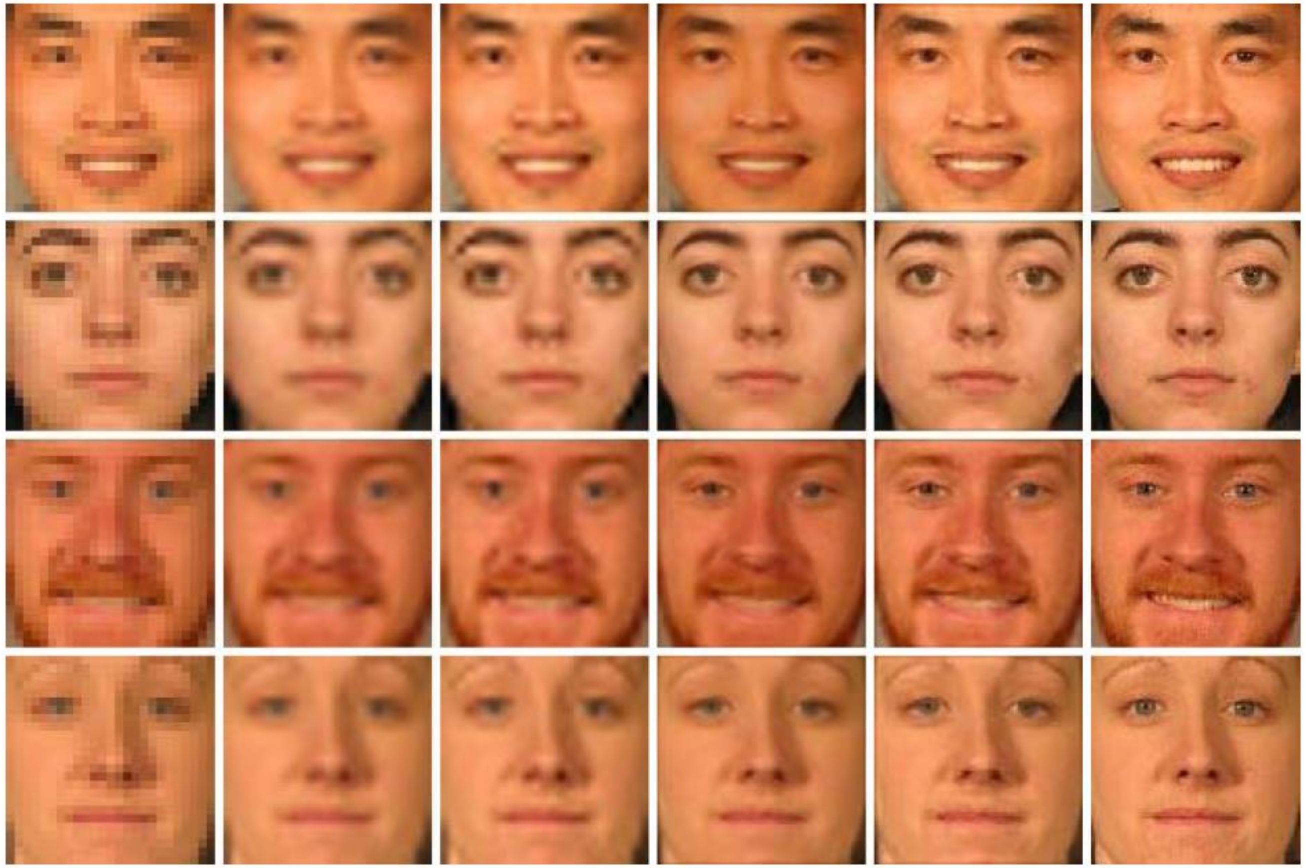
Image Super-Resolution

- • Super-Resolution(SR)
- Low resolution images ➡ High resolution images
- • Face Hallucination
  - Super-resolution on human faces
- • Constraints
- – Reconstruction constraints
  - Close to input image when blurred and down-sampled
- – Global structure constraints
  - Recovered image should be like a face
- – Sparsity assumption
  - Sparse coefficients are preserved during down-sampling



Courtesy of Jianchao Yang

Face Hallucination



Input image | Bicubic interpolation | Back projection | NMF | Sparse Coding | Original

Courtesy of Jianchao Yang

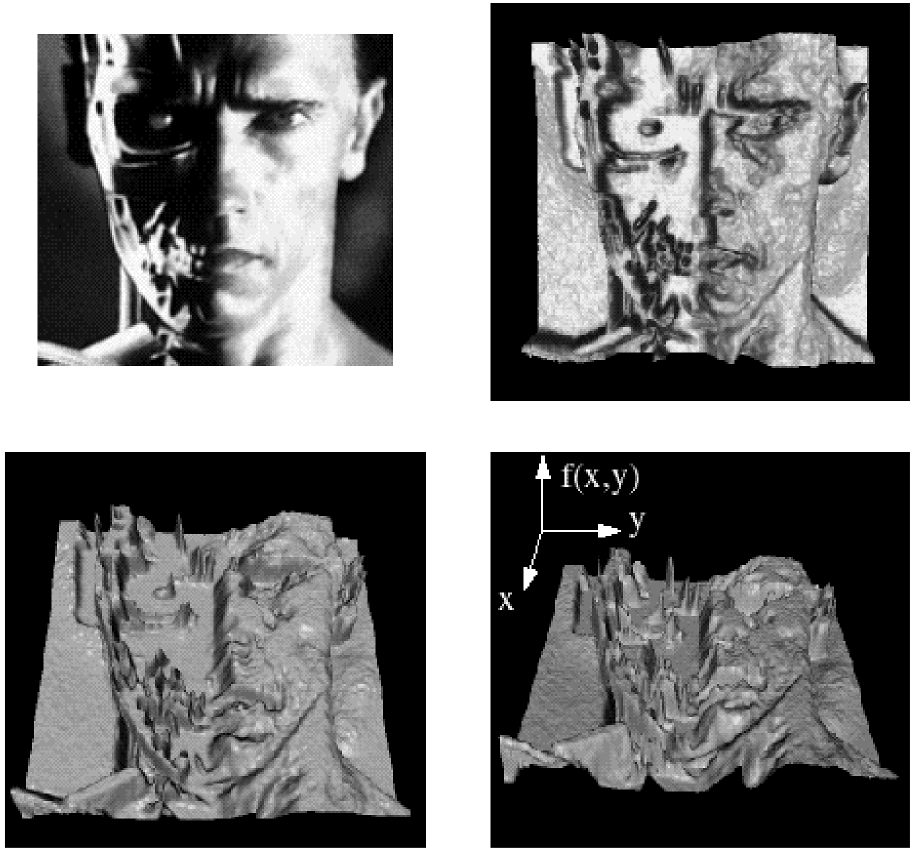
What is an Image?

What is an image?

- We can think of an image as a function, f:
  - f(x, y) gives the intensity at position (x, y)
  - Realistically, we expect the image only to be defined over a rectangle, with a finite range:
    - f: [a,b]x[c,d] → [0,1]
- A color image is just three functions pasted together. We can write this as a "vector-valued" function

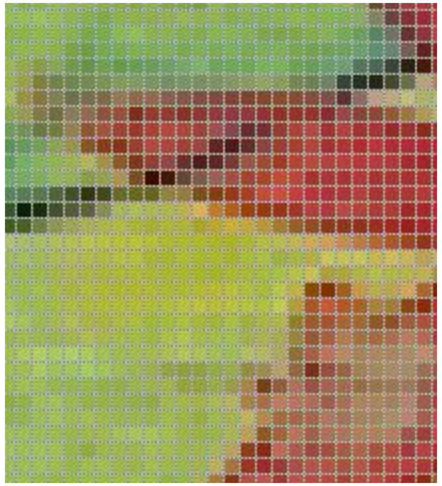
$$f(x,y)=\begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

Images as functions



Digital Image

- In digital imaging, a pixel (picture element) is the smallest piece of information in an image.
- The word pixel is based on a contraction of pix (for "pictures") and el (for "element")
- Each pixel is a sample of an original image, where more samples typically provide a more accurate representation of the original.
- The intensity of each pixel is variable; in color systems, each pixel has typically three or four components such as red, green, and blue, or cyan, magenta, yellow, and black.



http://en.wikipedia.org/wiki/Pixel
Google.com image search

What is a digital image?

- We usually operate on digital (discrete) images:
  - Sample the 2D space on a regular grid
  - Quantize each sample (round to nearest integer)
- The image can now be represented as a matrix of integer values

$j \longrightarrow$

$i \downarrow$

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

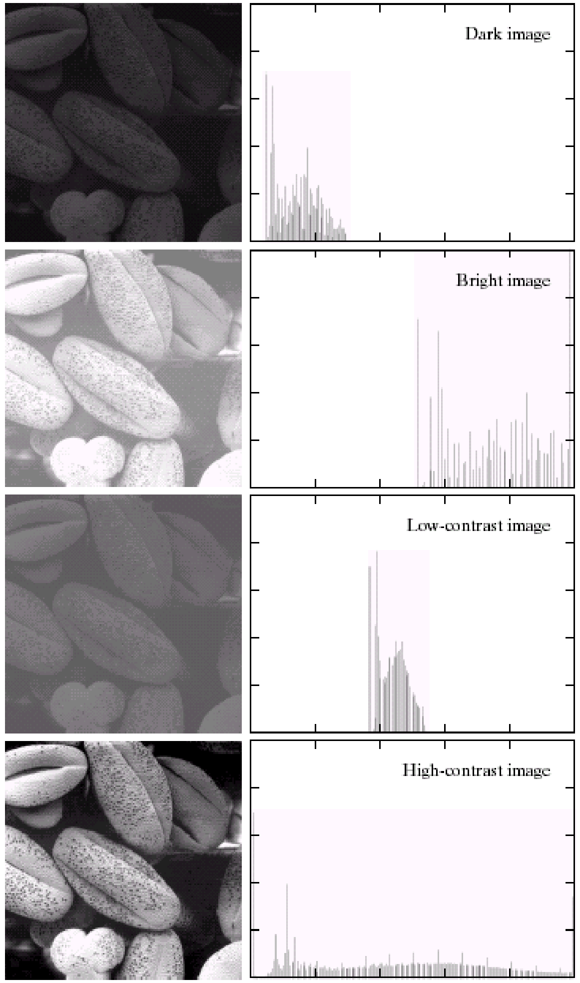
Other Applications

Image Enhancement

a b
c d

FIGURE 3.9 (a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 3.0, 4.0,$ and 5.0, respectively. (Original image for this example courtesy of NASA.)

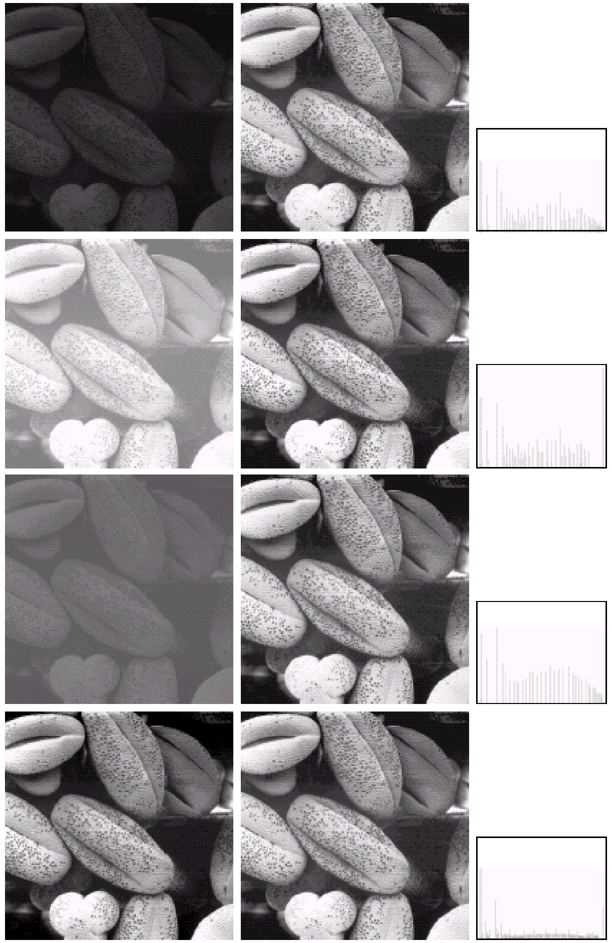
Image Histograms



a b

FIGURE 3.15 Four basic image types: dark, light, low contrast, high contrast, and their corresponding histograms. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)

Histogram Equalization



a b c

FIGURE 3.17 (a) Images from Fig. 3.15. (b) Results of histogram equalization. (c) Corresponding histograms.

How can we read images?

Image file Formats

- Image file formats are standardized means of organizing and storing digital images.
- Types of files
  - JPEG
  - PNG
  - BMP
  - GIF
  - PPM
  - PGM
  - Etc

How can we read these formats

- So many different formats, which one is the best one?
- Which one is the easiest to interpret?
- The answer to these questions is not easy.
- Best way to read these types of files is through libraries such as Cimg, libjpeg, OpenCV.

Simplest version

- portable pixmap format (PPM), the portable graymap format (PGM)
  - These formats don't use any sort of compression so they are big files but easy to read the data.

```
P2
# Shows the word "FEEP" (example from Netpbm man page on PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Simplest version

- In this PGM example
  - P2 means PGM
  - The next two numbers give the width and the height.
  - The next number represents the maximum value (numbers of grey between black and white)
    - Black is 0 and max value is white.
  - Then follows the matrix with the pixel values.
  - The PGM and PPM formats (both ASCII and binary versions)

```
P2
# Shows the word "FEEP" (example from Netpbm man page on PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Simple algorithm

- Brightness
  - Increase/Decrease the intensity of the pixels by a specific value
  - Lets look at the code

What is the Makefile

```
# environment
SM := 35

# compilers
CUDA_PATH ?= $(shell test -d /shared/apps/cuda7.0 && echo /shared/apps/cuda7.0)

ifeq ($(CUDA_PATH),)
CUDA_PATH := $(shell test -d /usr/local/cuda-7.0 && echo /usr/local/cuda-7.0)
endif

GCC := g++
NVCC := $(CUDA_PATH)/bin/nvcc
MPI  = mpiCC

# libraries
CUDA_LIB_PATH := $(CUDA_PATH)/lib

# Remove function
RM = rm -f

# Compiler flags:
# -g    debugging information
# -Wall turns on most compiler warnings
GENCODE_FLAGS := -gencode arch=compute_$(SM),code=sm_$(SM)
LIB_FLAGS := -lcudadevrt -lcudart
ifeq ($(OS), DARWIN)
	CCFLAGS := -stdlib=libstdc++
else
	CCFLAGS := -O3
endif
NVCCFLAGS :=
GccFLAGS = -fopenmp -O3
MPIFLAGS = -Wno-deprecated

debug: GccFLAGS += -DDEBUG -g -Wall
debug: MPIFLAGS += -DDEBUG
debug: NVCCFLAGS += -g -G
debug: all

# The build target executable:
TARGET = brightness

all: build

build: $(TARGET)

$(TARGET): lib/dlink.o lib/main.o lib/alg/$(TARGET).o lib/img/imghandler.o lib/alg/locationhandler.o
	$(NVCC) $(NVCCFLAGS) $^ -o $@ $(GENCODE_FLAGS) -link

lib/dlink.o: lib/alg/$(TARGET).o
	$(NVCC) $(NVCCFLAGS) $^ -o $@ $(GENCODE_FLAGS) -dlink

lib/main.o: lib/main.cpp lib/config.h
	$(GCC) $(GccFLAGS) -c $< -o $@

lib/alg/locationhandler.o: lib/alg/locationhandler.cpp lib/alg/locationhandler.h lib/config.h
	$(GCC) $(GccFLAGS) -c $< -o $@

lib/alg/$(TARGET).o: lib/alg/$(TARGET).cu lib/config.h
	$(NVCC) $(NVCCFLAGS) -dc $< -o $@ $(GENCODE_FLAGS)

lib/img/imghandler.o: lib/img/imghandler.cpp lib/config.h
	$(GCC) $(GccFLAGS) -c $< -o $@

clean:
	$(RM) $(TARGET) *.o lib/*.o *.tar* *.core*
```

What is the Makefile

- Makefiles are a simple way to organize code compilation.
- A makefile is a file containing a set of directives used with the make build automation tool from GNU.
- Incredibly useful when your project becomes bigger and bigger.

```makefile
# environment
SM := 35

# compilers
CUDA_PATH ?= $(shell test -d /shared/apps/cuda7.0 && echo /shared/apps/cuda7.0)

ifeq ($(CUDA_PATH),)
CUDA_PATH := $(shell test -d /usr/local/cuda-7.0 && echo /usr/local/cuda-7.0)
endif

GCC := g++
NVCC := $(CUDA_PATH)/bin/nvcc
MPI  = mpiCC

# libraries
CUDA_LIB_PATH := $(CUDA_PATH)/lib

# Remove function
RM = rm -f

# Compiler flags:
# -g    debugging information
# -Wall turns on most compiler warnings
GENCODE_FLAGS := -gencode arch=compute_$(SM),code=sm_$(SM)
LIB_FLAGS := -lcudadevrt -lcudart
ifeq ($(OS), DARWIN)
	CCFLAGS := -stdlib=libstdc++
else
	CCFLAGS := -O3
endif
NVCCFLAGS :=
GccFLAGS = -fopenmp -O3
MPIFLAGS = -Wno-deprecated

debug: GccFLAGS += -DDEBUG -g -Wall
debug: MPIFLAGS += -DDEBUG
debug: NVCCFLAGS += -g -G
debug: all

# The build target executable:
TARGET = brightness

all: build

build: $(TARGET)

$(TARGET): lib/dlink.o lib/main.o lib/alg/$(TARGET).o lib/img/imghandler.o lib/alg/locationhandler.o
	$(NVCC) $(NVCCFLAGS) $^ -o $@ $(GENCODE_FLAGS) -link

lib/dlink.o: lib/alg/$(TARGET).o
	$(NVCC) $(NVCCFLAGS) $^ -o $@ $(GENCODE_FLAGS) -dlink

lib/main.o: lib/main.cpp lib/config.h
	$(GCC) $(GccFLAGS) -c $< -o $@

lib/alg/locationhandler.o: lib/alg/locationhandler.cpp lib/alg/locationhandler.h lib/config.h
	$(GCC) $(GccFLAGS) -c $< -o $@

lib/alg/$(TARGET).o: lib/alg/$(TARGET).cu lib/config.h
	$(NVCC) $(NVCCFLAGS) -dc $< -o $@ $(GENCODE_FLAGS)

lib/img/imghandler.o: lib/img/imghandler.cpp lib/config.h
	$(GCC) $(GccFLAGS) -c $< -o $@

clean:
	$(RM) $(TARGET) *.o lib/*.o *.tar* *.core*
```

```
int main(int argc, char* argv[]) {

        // Files needed
        char* imageFile = NULL;

        // Load Intensity Image
        image<unsigned char>* intensityInput = loadPGM(imageFile);

        cpu.height = intensityInput->height();
        cpu.width  = intensityInput->width();

        cpu.gridXSize = 1 + (( cpu.width - 1) / TILE_SIZE);
        cpu.gridYSize = 1 + ((cpu.height - 1) / TILE_SIZE);

        int XSize = cpu.gridXSize*TILE_SIZE;
        int YSize = cpu.gridYSize*TILE_SIZE;

        cpu.size = XSize*YSize;

        cpu.intensity = new unsigned char[cpu.size];

        for (unsigned int y = 0 ; y < YSize ; y++){
                for (unsigned int x = 0 ; x < XSize ; x++){
                        unsigned int newLocation =
imageLocation (x, y, cpu.gridXSize);
                        if (x < cpu.width && y < cpu.height) {
                                cpu.intensity[newLocation] =
intensityInput->data[y*cpu.width + x];
                        } else{
                                // Necessary in case image size is
not a multiple of TILE_SIZE
                                cpu.intensity[newLocation] = 0;
                        }
                }
        }


        cpu.result = brightness(cpu.intensity,
                                cpu.height,
                                cpu.width);
```

```
        // Output RGB images
        char filename[64];
        sprintf(filename, "result.ppm");

        srand(1000);

        Color color;

        // Create output image
        image<Color> output = image<Color>(cpu.width, cpu.height,
true);
        image<Color>* im = &output;

        Color randomcolor = randomColor();
        for (unsigned int y = 0 ; y < cpu.height ; y++){
                for (unsigned int x = 0 ; x < cpu.width ; x++){
                        unsigned int newLocation = imageLocation
(x, y, cpu.gridXSize);

                        color.r = cpu.result[newLocation];
                        color.g = cpu.result[newLocation];
                        color.b = cpu.result[newLocation];
                        im->access[y][x] = color;
                }
        }
        savePPM(im, filename);

        // Free resources and end the program
        free(cpu.intensity);

        return 0;
}
```

```
unsigned char *brightness(unsigned char *intensity,
                unsigned int height,
                unsigned int width){

        int gridXSize = 1 + (( width - 1) / TILE_SIZE);
        int gridYSize = 1 + ((height - 1) / TILE_SIZE);

        int XSize = gridXSize*TILE_SIZE;
        int YSize = gridYSize*TILE_SIZE;

        // Both are the same size (CPU/GPU).
        gpu.size = XSize*YSize;

        // Allocate arrays in GPU memory

        checkCuda(cudaMalloc((void**)&gpu.intensity
, gpu.size*sizeof(char)));
        checkCuda(cudaMalloc((void**)&gpu.result
, gpu.size*sizeof(char)));

        // Allocate result array in CPU memory
        gpu.resultOnCPU = new unsigned char[gpu.size];

        checkCuda(cudaMemcpy(gpu.intensity,
                        intensity,
                        gpu.size*sizeof(char),
                        cudaMemcpyHostToDevice));

        checkCuda(cudaDeviceSynchronize());

        dim3 dimGrid(gridXSize, gridYSize);
        dim3 dimBlock(BLOCK_TILE_SIZE, BLOCK_TILE_SIZE);

        // Launch kernel to begin image segmenation
        brightnessAlgorithm<<<dimGrid,
dimBlock>>>(gpu.intensity,
                                                gpu.result,
                                                inc);

        checkCuda(cudaDeviceSynchronize());

        // Retrieve results from the GPU
        checkCuda(cudaMemcpy(gpu.resultOnCPU,
                        gpu.result,
                        gpu.size*sizeof(char),
                        cudaMemcpyDeviceToHost));

        // Free resources and end the program
        checkCuda(cudaFree(gpu.intensity));
        checkCuda(cudaFree(gpu.result));

        return(gpu.resultOnCPU);

}
```

```
__global__ void brightnessAlgorithm(unsigned char
*intensity,
                                    unsigned char
*result,
                                    unsigned int inc){

        int tx = threadIdx.x;
        int ty = threadIdx.y;
        int bx = blockIdx.x;
        int by = blockIdx.y;

        // Read Input Data
        int x = bx*TILE_SIZE+tx;
        int y = by*TILE_SIZE+ty;

        int location =  y*(gridDim.x*TILE_SIZE)+x;

        unsigned char value = intensity[location];

        // Algorithm
        if (value + inc > 255) result[location] =
255;
        else result[location] = value + inc;

}
```

How can we improve brightness?

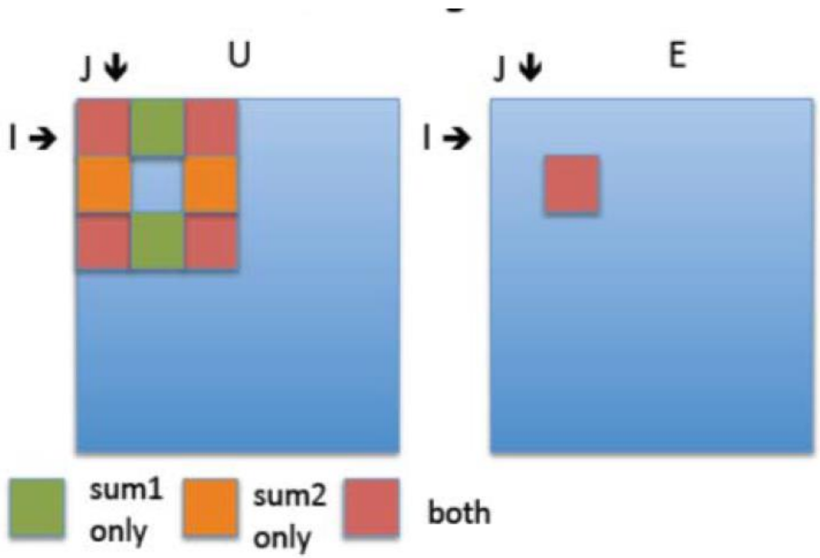
- Shared memory?
- Constant Memory?
- Texture memory?

How can we improve brightness?

- More processing per thread
  - Adding more work to each thread
- Bring in more data on each read

A more complex example

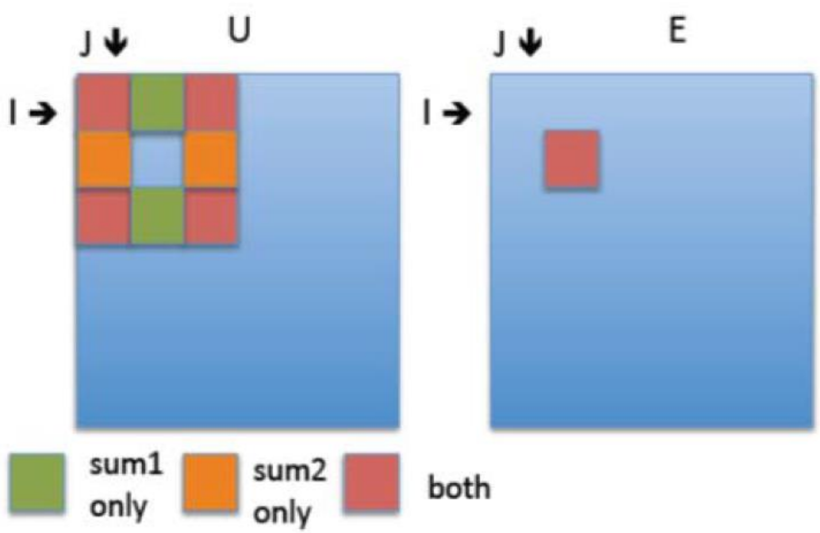
- Sobel algorithm
  - Sobel edge detection
    - Find the boundaries of the image where there is significant difference as compared to neighboring “pixels” and replace values to find edges.



Sobel Algorithm

- Looks for edges in both horizontal and vertical directions, then combine the information into a single metric.
- The masks are:

$$y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



Edge Magnitude =

$$\sqrt{x^2 + y^2}$$

Edge Direction = $\tan^{-1}\left[\frac{y}{x}\right]$

Sobel Algorithm

Input

Output

```
int main(int argc, char* argv[]) {

        // Files needed
        char* imageFile = NULL;

        // Load Intensity Image
        image<unsigned char>* intensityInput =
loadPGM(imageFile);

        cpu.height = intensityInput->height();
        cpu.width  = intensityInput->width();

        cpu.gridXSize = 1 + (( cpu.width - 1) / TILE_SIZE);
        cpu.gridYSize = 1 + ((cpu.height - 1) / TILE_SIZE);

        int XSize = cpu.gridXSize*TILE_SIZE;
        int YSize = cpu.gridYSize*TILE_SIZE;

        cpu.size = XSize*YSize;

        cpu.intensity = new unsigned char[cpu.size];

        for (unsigned int y = 0 ; y < YSize ; y++){
                for (unsigned int x = 0 ; x < XSize ; x++){
                        unsigned int newLocation =
imageLocation (x, y, cpu.gridXSize);
                    if (x < cpu.width && y < cpu.height) {
                        cpu.intensity[newLocation] =
intensityInput->data[y*cpu.width + x];
                    } else{
                        // Necessary in case image size
is not a multiple of TILE_SIZE
                        cpu.intensity[newLocation] = 0;
                    }
                }
        }


        cpu.result = sobel(cpu.intensity,
                           cpu.height,
                                                cpu.width);

        // Output RGB images
        char filename[64];
        sprintf(filename, "result.ppm");

        srand(1000);

        Color color;

        // Create output image
        image<Color> output = image<Color>(cpu.width, cpu.height,
true);
        image<Color>* im = &output;

        Color randomcolor = randomColor();
        for (unsigned int y = 0 ; y < cpu.height ; y++){
                for (unsigned int x = 0 ; x < cpu.width ; x++){
                        unsigned int newLocation = imageLocation
(x, y, cpu.gridXSize);

                        color.r = cpu.result[newLocation];
                        color.g = cpu.result[newLocation];
                        color.b = cpu.result[newLocation];
                        im->access[y][x] = color;
                }
        }
        savePPM(im, filename);

        // Free resources and end the program
        free(cpu.intensity);

        return 0;
}
```

```
unsigned char *sobel(unsigned char *intensity,
                unsigned int height,
                unsigned int width){

        int gridXSize = 1 + (( width - 1) /
TILE_SIZE);
        int gridYSize = 1 + ((height - 1) /
TILE_SIZE);

        int XSize = gridXSize*TILE_SIZE;
        int YSize = gridYSize*TILE_SIZE;

        // Both are the same size (CPU/GPU).
        gpu.size = XSize*YSize;

        // Allocate arrays in GPU memory
        checkCuda(cudaMalloc((void**)&gpu.intensity
, gpu.size*sizeof(char)));
        checkCuda(cudaMalloc((void**)&gpu.result
, gpu.size*sizeof(char)));

        // Allocate result array in CPU memory
        gpu.resultOnCPU = new unsigned
char[gpu.size];

        checkCuda(cudaMemcpy(gpu.intensity,
                intensity,
                gpu.size*sizeof(char),
                cudaMemcpyHostToDevice));

        checkCuda(cudaDeviceSynchronize());
```

```
        dim3 dimGrid(gridXSize, gridYSize);
        dim3 dimBlock(BLOCK_TILE_SIZE,
BLOCK_TILE_SIZE);

        // Launch kernel to begin image segmenation
        sobelAlgorithm<<<dimGrid,
dimBlock>>>(gpu.intensity,
gpu.result,threshold);

        checkCuda(cudaDeviceSynchronize());

        // Retrieve results from the GPU
        checkCuda(cudaMemcpy(gpu.resultOnCPU,
                gpu.result,
                gpu.size*sizeof(char),
                cudaMemcpyDeviceToHost));

        // Free resources and end the program
        checkCuda(cudaFree(gpu.intensity));
        checkCuda(cudaFree(gpu.result));


        return(gpu.resultOnCPU);

}
```

```
__global__ void sobelAlgorithm(unsigned char
*intensity,
                             unsigned char
*result,
                             unsigned int
threshold){

        int tx = threadIdx.x;
        int ty = threadIdx.y;
        int bx = blockIdx.x;
        int by = blockIdx.y;

        int x = bx*TILE_SIZE+tx;
        int y = by*TILE_SIZE+ty;

        int xsize = TILE_SIZE*gridDim.x;
        int ysize = TILE_SIZE*gridDim.y;

        if (y > 1 && y < ysize-1 && x > 1 && x
< xsize-1){
                int location =  y*xsize+x;
                int sum1 =  input[ xsize * (y-1) + x+1 ] -
                            input[ xsize * (y-1) + x-1 ] +
                        2 * input[ xsize * (y)   + x+1 ] -
                        2 * input[ xsize * (y)   + x-1 ] +
                            input[ xsize * (y+1) + x+1 ] -
                            input[ xsize * (y+1) + x-1 ];
                int sum2 = input[ xsize * (y-1) + x-1 ] +
                       2 * input[ xsize * (y-1) + x   ] +
                           input[ xsize * (y-1) + x+1 ] -
                           input[ xsize * (y+1) + x-1 ] -
                       2 * input[ xsize * (y+1) + x   ] -
                           input[ xsize * (y+1) + x+1 ];

                int magnitude =
sum1*sum1+sum2*sum2;

                if (magnitude > threshold)
                        result[location] =
255;
                else
                        result[location] = 0;
        }
}
```