Computer Science 1510

Lecture 33

April 1, 2016

Lecture Outline

- Special topics: Image processing

Final exam:
Wednesday, April 13
12 – 2 pm
EN-1001

Image processing

- Image processing involves applying signal processing techniques to compute characteristics of an image, or to intelligently modify an image.

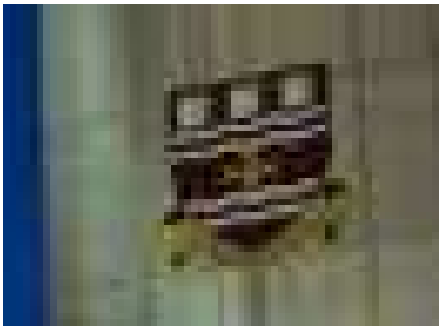
- The image is the input, while the output could be a set of characteristics of the image or another image.

- Applications of image processing may include:
  - Image enhancement
  - Image restoration
  - Segmentation - dividing the image into regions of uniform characteristics
  - Object classification
  - Image compression

Images

- An image is simply a grid of squares, where each square contains a single colour.

- For colour images each square (or "pixel") contains three values (RGB), while for black and white images each square contains one value.



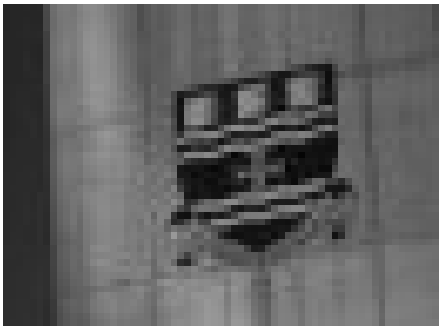


Image representation

- We can represent a black and white image as a matrix of intensity values (between 0 and 255).

- For example,

$$\begin{bmatrix} 2 & 3 & 5 & \cdots & 9 \\ 6 & 8 & 10 & \cdots & 15 \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ 2 & 3 & 5 & \cdots & 9 \end{bmatrix}$$

Edge detection

- A common image processing technique is edge detection, that is, to determine the boundaries between objects in an image.

- Edges occur where there is a distinct change in the intensity value.

- A threshold value can be used to select edges of a given strength. Requiring the change to be large may result in missing some edges, while a too small change may be too sensitive to small differences in intensity.

Edge detection (imageType.h)

```
#ifndef IMAGE_TYPE_H
#define IMAGE_TYPE_H

typedef struct imageType_s {
   unsigned int width;
   unsigned int height;
   unsigned int nchans;
   unsigned char *data;
} Image_t;
#endif
```

Edge detection (main.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ppmio.h"
#include "imageType.h"

void makeBigEnough(Image_t *img) {
   img->data=(unsigned char*)realloc(img->data,
         img->width*img->height*img->nchans*sizeof(unsigned char));
   return;
}

void makeSameSize(Image_t parent, Image_t *child) {
   child->width=parent.width;
   child->height=parent.height;
   child->nchans=parent.nchans;
   makeBigEnough(child);
}

void clearImage(Image_t *img) {
   size_t size=img->width*img->height*img->nchans;
   memset(img->data,0,size);
}

// Allocates sufficient memory if required;
void convertToGrayscale(Image_t color, Image_t *gray) {
   int i;
   int j;
   gray->width=color.width;
   gray->height=color.height;
   gray->nchans=1;
   makeBigEnough(gray);
   /* loop over all pixels */
   for (i=0;i<color.width*color.height;i++) {
      int avg=0;
```

```
      /* loop over channels */
      for (j=0;j<color.nchans;j++) {
         avg+=color.data[i*color.nchans+j];
      }
      gray->data[i]=avg/color.nchans;
   }
}

/* assumes images are grayscale */
int edgeDetect(Image_t src, Image_t *dst) {
   int i,j;
   int dx,dy;
   int diff;
   if (src.nchans !=1) {
      return -1;
   }
   makeSameSize(src,dst);
   clearImage(dst);
   for (j=1;j<src.height;j++) {
      for (i=1;i<src.width;i++) {
         dx=src.data[j*src.width+i]-src.data[j*src.width+i-1];
         dy=src.data[j*src.width+i]-src.data[(j-1)*src.width+i];
         diff=(dx>0?dx:-dx)+(dy>0?dy:-dy);
         if (diff>255) diff=255;
         dst->data[j*src.width+i]=(unsigned char)diff;
      }
   }
   return 0;
}

void threshold(Image_t src, Image_t *dst, unsigned char thresh) {
   makeSameSize(src,dst);
   int i;
   for (i=0;i<src.width*src.height*src.nchans;i++) {
      if (src.data[i]>thresh) dst->data[i]=255;
      else dst->data[i]=0;
   }
   return;
```

```
}

/* add two images together, result goes to dst */
void superimpose (Image_t A, Image_t B, Image_t *dst) {
   int i,j;
   int bj;
   int tot=0;
   makeSameSize(A,dst);
   for (i=0;i<A.width*A.height;i++) {
      for (j=0;j<A.nchans;j++) {
         bj=(j<B.nchans?j:0);
         tot=A.data[i*A.nchans+j]+B.data[i*B.nchans+bj];
         dst->data[i*A.nchans+j]=(tot<=255?tot:255);
      }
   }
}

int main(int argc, char *argv[]) {
   int th=0;
   Image_t color={0,0,0,NULL};
   Image_t gray={0,0,0,NULL};
   Image_t edges={0,0,0,NULL};
   Image_t thresh={0,0,0,NULL};
   Image_t super={0,0,0,NULL};
   if (argc!=2) {
         fprintf(stdout,"Usage: %s threshold(0-255)\n",argv[0]);
         return 0;
   }
   th=atoi(argv[1]);
   readPPM("in.ppm",&color);
   convertToGrayscale(color,&gray);
   writePPM("gray.ppm",gray);
   edgeDetect(gray,&edges);
   writePPM("edges.ppm",edges);
   threshold(edges,&thresh,th);
   writePPM("thresh.ppm",thresh);
   superimpose(color,thresh,&super);
   writePPM("super.ppm",super);
```

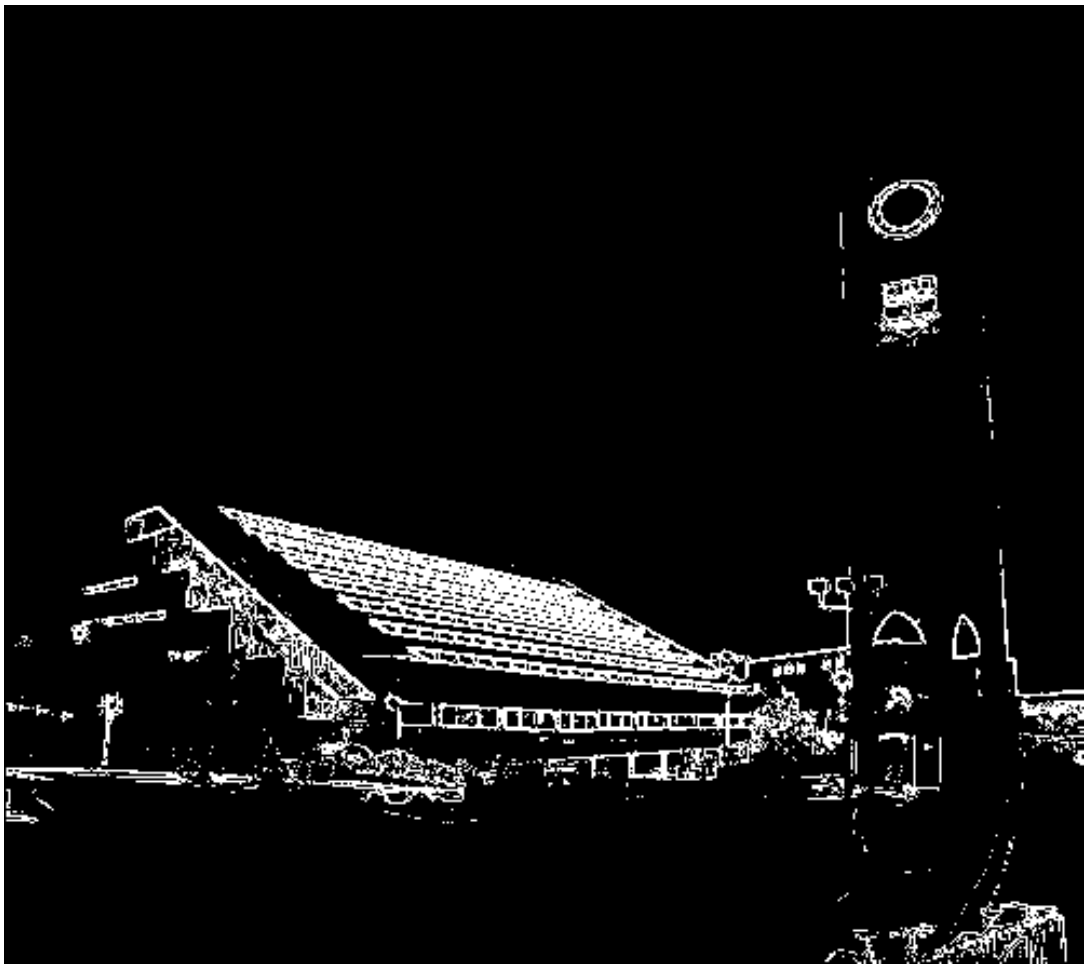
```
   return 0;
}
```

Original image

Grayscale image

Edges image

Thresholded image



Superimposed image