Previous Lecture:
- Working with images

Today’s Lecture:
- Another example on images: edge finding
- Another example on characters and strings
- Introduction to cell arrays

Announcements:
- Discussion this week in classrooms (Hollister or Phillips)
- Project 4 due Thurs 4/7 at 11pm
- Make use of tutoring—sign up on CMS next week
- Any feedback on consulting? Feedback link available in consulting area (Carpenter Hall ACCEL Green Rm)
- Check website “Staff” link for updates on office hours

What is an edge?

Near an edge, grayness values change abruptly

<table>
<thead>
<tr>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

The Rate-of-Change-Array

Suppose \( A \) is a 2-d image array; it has the type \( \text{uint8} \).
Let \( B(i,j) \) be the maximum difference between \( A(i,j) \) and its eight neighbors; \( B \) is the rate-of-change array.
So \( B(i,j) \) is the maximum value in

\[
A(\max(1,i-1):\min(m,i+1),\ldots,\max(1,j-1):\min(n,j+1)) - A(i,j)
\]

General plan for showing the edges in in image

- Identify the “edge pixels”
- Highlight the edge pixels
  - make edge pixels white; make everything else black

Built-in function to convert to grayscale. Returns 2-d array.

function Edges(jpgIn,jpgOut,tau)
% jpgOut is the “edge diagram” of image jpgIn.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be an edge.
A = rgb2gray(imread(jpgIn));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
  for j = 1:n
    B(i,j) = A(i,j) - A(max(1,i-1):min(m,i+1),
                              max(1,j-1):min(n,j+1))
    if B(i,j) > tau
      B(i,j) = 255; % B is rate-of-change array
    else
      B(i,j) = 0;
    end
  end
end
Rate-of-change example

<table>
<thead>
<tr>
<th>90</th>
<th>81</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>60</td>
<td>59</td>
</tr>
<tr>
<td>56</td>
<td>57</td>
<td>58</td>
</tr>
</tbody>
</table>

Rate-of-change at middle pixel is 30

Be careful! In "uint8 arithmetic" 57 - 60 is 0

Recipe for rate-of-change $B(i,j)$

% The 3-by-3 subarray that includes $A(i,j)$
% and its 8 neighbors (for an interior pixel)
Neighbors = $A(i-1:i+1,j-1:j+1)$;

% Subtract $A(i,j)$ from each entry
Diff = abs(double(Neighbors) - double(A(i,j)));

% Compute largest value in each column
colMax = max(Diff);

% Compute the max of the column max's
$B(i,j) = max(colMax)$;

function Edges(jpgIn,jpgOut,tau)
% jpgOut is the "edge diagram" of image jpgIn.
% At each pixel, if rate-of-change > tau then the pixel is considered an edge.
A = rgb2gray(imread(jpgIn));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
  for j = 1:n
    Neighbors = A(max(1,i-1):min(i+1,m), ... max(1,j-1):min(j+1,n));
    B(i,j) = max(max(abs(double(Neighbors) - double(A(i,j)))));
    if B(i,j) > tau
      B(i,j) = 255;
    else
      B(i,j) = 0;
    end
  end
end
imwrite(B,jpgOut,'jpg')

“Edge pixels” are now identified; display them with maximum brightness (255)

A

|  2 3 2 1 1 1 |
|  2 1 1 1 1 1 |
|  1 1 1 90 90 90 |
|  1 1 90 90 90 90 |
|  1 90 90 90 90 90 |
|  1 90 90 90 90 90 |

B(i,j)

|  2 1 1 1 0 0 |
|  1 2 89 89 89 |
|  1 1 89 89 0 0 |
|  1 1 89 89 0 0 |
|  1 89 89 0 0 0 |
|  1 89 89 0 0 0 |

Edge finding: Effect of edge threshold, $\tau$
Matlab types: char, double, uint8, logical

- **a** is a 1-d array with type char components. We call a a "string" or "char array"
- **b** is a 1-d array with type double components. double is the default type for numbers in Matlab. We call b a "numeric array"
- **c** is a 1-d array with type uint8 components. We call c a "uint8 array"
- **d** is a scalar of the type logical. We call d a "boolean value"

Example: censoring words

```matlab
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A, regardless of case.
% A is a matrix of characters.
% str is a string. Assume that str is never split across two lines.
% D is A with X's replacing the censored string str.

D= A;
B= lower(A);
s= lower(str);
ns= length(str);

% Build a string of X's of the right length
Xs= char( zeros(1,ns));
for k= 1:ns
    Xs(k)= 'X';
end

% Traverse the matrix to censor string str
for r= 1:nr
    for c= 1:nc-ns+1
        if  strcmp( s , B(r, c:c+ns-1) )==1
            D(r, c:c+ns-1)= Xs;
        end
    end
end

Returns an array of type double
Changes the type to char
```

Array vs. Cell Array

- **Simple array**
  - Each component stores one scalar. E.g., one char, one double, or one uint8 value
  - All components have the same type

- **Cell array**
  - Each cell can store something "bigger" than one scalar, e.g., a vector, a matrix, a string (vector of chars)
  - The cells may store items of different types

1-d and 2-d examples ...

Cell array: individual components may contain different types of data
Cell Arrays of Strings

C = { 'Alabama', 'New York', 'Utah'}

C = [ 'Alabama' 'New York' 'Utah']

Use braces {} for creating and addressing cell arrays

Matrix

Cell Array

Create

Addressing

Example: Represent a deck of cards with a cell array

D(1) = 'A Hearts';
D(2) = '2 Hearts';

D(13) = 'K Hearts';
D(14) = 'A Clubs';

D(52) = 'K Diamonds';

But we don’t want to have to type all combinations of suits and ranks in creating the deck... How to proceed?

Make use of a suit array and a rank array...

suit = {'Hearts', 'Clubs', ...
         'Spades', 'Diamonds'};

rank = {'A', '2', '3', '4', '5', '6', ...
        '7', '8', '9', '10', 'J', 'Q', 'K'};

Then concatenate to get a card. E.g.,

str = [rank{3} ' ' suit{2} ];
D{16} = str;

So D{16} stores '3 Clubs'

To get all combinations, use nested loops

% Given cell arrays rank and suit
i = 1; % index of next card
for k= 1:4
    % Set up the cards in suit k
    for j= 1:13
        D{i} = [ rank{j} ' ' suit{k} ];
        i = i+1;
    end
end

See function CardDeck