In this exercise we will implement stereo reconstruction. You have the choice of using the alpha-expansion algorithm or the alpha-beta swap algorithm. You can implement both for an extra 15 point credit.

Brief overview

The setting is as follows. A pair of (gray-level) images, \textbf{im1} and \textbf{im2}, that forms a rectified stereo pair of a static scene is given. The goal is to find the pixel-to-pixel correspondence, that is, given a pixel \((i,j)\) in \textbf{im1}, find the corresponding pixel in \textbf{im2}. The target pixel should be searched for along the corresponding epipolar line. Denoting by \(f(p)\) the disparity (shift) of a pixel \(p\), the problem can be formulated the following way:

\[
(*) \quad f^* = \arg\min_f \left( \sum_{p \in P} D_p (f(p)) + \sum_{\{p, q\} \in N} V_{p,q}(f(p), f(q)) \right)
\]

where:

- \(f(p)\), the \textbf{label} - the disparity of pixel \(p\).
- \(D_p(z)\), the \textbf{data} term, reflects how well pixel \(p\) in image \textbf{im1} fits its counterpart (pixel \(p\) shifted by \(z\)) in image \textbf{im2}.
- \(P\) is the set of pixels in \textbf{im1}.
- \(V_{p,q}(x, y)\), the \textbf{smoothness term} - the penalty of assigning disparity \(x\) to pixel \(p\) and disparity \(y\) to pixel \(q\).
- \(N\) - the set of interacting (neighboring) pixels, \(N \subseteq P \times P\).
The approximate minimization of an energy function of such a form can be done using either the alpha-expansion or the alpha-beta swap algorithm, presented in Boykov et al. ICCV 1999.

There are several forms for the smoothness and data terms.

**Data term**

In this exercise we use the following form for $D$:

$$ (***) D_p(f(p)) = (\min \{ D_{fwd}(p, f(p)), D_{rev}(p, f(p)), 20 \})^2 $$

where

$$ D_{fwd}(p, f(p)) = \min_{f(p)-0.5 \leq x \leq f(p)+0.5} |im1(p) - im2(p + x)| $$

$$ D_{rev}(p, f(p)) = \min_{p-0.5 \leq x \leq p+0.5} |im1(x) - im2(p + f(p))| $$

Where $im(x)$ is the image value at pixel location $x$ along the epipolar line. Fractional image locations are computed using linear interpolation. This is basically the square of the truncated absolute value of the difference between the pixels in both images allowing a compensation for the "discreteness of the image".

**Smoothness term**

We will be using smoothness terms that **do not depend on pixel values** but on the disparities alone. Such terms are called **spatially constant**. You will investigate two possibilities for the smoothness term:

1. **Potts model**:

   $$ V(x, y) = \begin{cases} K & \text{for } x \neq y \\ 0 & \text{for } x = y \end{cases} $$

   where $K=20$. Note that this data term is uniform across all possible disparities.

2. **Truncated L1**:

   $$ V(x, y) = 15 \times \min(3, |x - y|) $$
Coding Instructions:
This is a Matlab programming assignment.
- It is very important that you thoroughly document your code. In particular each function should begin with a short description of what it does and what the input and output are.
- Remember that Matlab is efficient when the code is vectorized, so try to avoid using loops.
- You may use the code for computing min-cut / max-flow graph cuts found at the course web site. This code will run on Windows 32-bit Matlab.
- In this exercise all images are already rectified, so the epipolar lines on the two images are simply the horizontal lines at the same height. (This affects the direction of the scan in the data term).

Question 1
In this question you are required to code the functions that perform a one-time computations needed for the iterative optimization.

a. Write a function called \( DT = \text{computeDataTerm}(\text{im1}, \text{im2}, \text{d}) \)
where:
- \( \text{im1}, \text{im2} \) – gray-level images of size MxN pixels (values range from 0 to 255).
- \( \text{d} \) – a 1xNd vector of possible disparities.
- \( \text{DT} \) – a MNxNd array of the data term for each pixel and for each possible disparity value (the 1\(^{\text{st}}\) index of DT is the linear indexing of im1 produced by column-stack).

Compute DT, according to equation (**)

b. Write two functions called:

\[
\text{[spatiallyConstCost]} = \text{computeSmoothnessTermPotts(d)}
\]
\[
\text{[spatiallyConstCost]} = \text{computeSmoothnessTermTruncL1(d)}
\]
Where:

1. \( \text{d} \) – a 1xNd vector of possible disparities
2. \( \text{spatiallyConstCost} \) – the spatially constant cost (the smoothness term).
   This is an array of size Nd x Nd with the cost of any two neighboring pixels having different disparities
c. Write function called:

```python
[edgeList] = generateEdges4Connected(M,N)
```

Where:

- $M, N$ – scalars (the size of the image)
- `edgeList` – Hx2 array representing neighboring pixels in the image, when each non-boundary pixel is 4-connected. H is the number of edges. An entry $(p,q)$ in `edgeList` establishes a link between the pixels $p$ and $q$ (in column-stack notation)

**Question 2**

In this question you’re required to code the routines needed for the iterative optimization.

a. Write a function called:

```python
[E] = computeEnergy(f, dataTerm, edgeList, smoothTerm)
```

Where:

- `dataTerm` – MNxN$_d$ array, output of `computeDataTerm`
- `edgeList` – Hx2 array representing neighboring pixels in the image, output of `generateEdges4Connected`
- `smoothTerm` – an array of size N$_d$xN$_d$, output of `computeSmoothnessTerm`* (see question 1, b)
- $f$ – MNx1 vector. **the indices** to `dataTerm` and `smoothTerm` **representing the current assignment**. That is: `dataTerm(k, f(k))` is the value of `dataTerm` of pixel $k$ when it is assigned the disparity $d(f(k))$, where $d$ is the vector of **real disparities** used to compute `dataTerm` with the `computeDataTerm` function. Similarly for `smoothTerm(f(p),f(q))`.

The function computes the total energy of a certain assignment $f$ according to equation (*)
b. Write one of the following functions, or both for extra credit:

\[
\text{nextAssignment} = \ldots
\]

\[
\text{alphaBetaSwap}(f, \text{dataTerm, edgeList, smoothTerm, alpha, beta})
\]

\[
\text{nextAssignment} = \ldots
\]

\[
\text{alphaExpand}(f, \text{dataTerm, edgeList, smoothTerm, alpha})
\]

Where:

- \( f \) - a MNx1 vector, the indices of the current assignment, as before
- \( \alpha, \beta \) (alphaBetaSwap) - scalars, labels to swap in the current move
- \( \alpha \) (alphaExpand) - scalar, label to expand in the current move
- \( \text{nextAssignment} \) – a MNx1 vector, the indices of the next assignment
- The rest of the parameters are as before

These functions perform a single iteration of alpha-expansion or alpha-beta swap. Use the min-cut / max flow code found at the course web site. This is a code in C precompiled to run on a pc 32-bit (but not linux64 bit).

maxflow description

Let \( G = (V,E) \) be a graph and let \( s \) and \( t \) be two new vertices. Define a new graph by connecting each node in \( V \) to \( s \) and to \( t \).

function \([\text{energy}, \text{binary_assignment}] = \text{maxflow} (\text{edges,tweights})\)

inputs

- \( \text{edges} \) – \(|E|\times4\) matrix, the first and second elements in each row are the indices of two vertices in \( V \) which are connected by an edge. The third and fourth entries are the weights on this edge. (In our case the graph is undirected so the third and fourth entries are identical)

- \( \text{tweights} \) – \(|V|\times3\) matrix, the first element in each row is the node index (so the first column in \( \text{tweights} \) is just the node indices \((1:|V|)'\)). The second entry in each row includes the weight on the edge from that node to \( s \) and the third includes the weight on the edges from that node to \( t \).

outputs

- \( \text{energy} \) – You can ignore this output.

- \( \text{binary_assignment} \) – \(|V|\times1\) vector with the value 1 for nodes that are connected to \( s \), and 2 for nodes that are connected to \( t \) in the minimum-energy cut.
Use the following figures as guidelines. If you need more details you can look at the paper by Boykov et al.

### Alpha-Beta Swap

### Alpha Expansion

<table>
<thead>
<tr>
<th>edge</th>
<th>weight</th>
<th>for</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^\alpha_p$</td>
<td>$D_p(\alpha) + \sum_{q \in N_p\backslash P_{\alpha \beta}} V_{{p,q}}(\alpha, f_q)$</td>
<td>$p \in P_{\alpha \beta}$</td>
</tr>
<tr>
<td>$t^\beta_p$</td>
<td>$D_p(\beta) + \sum_{q \in N_p\backslash P_{\alpha \beta}} V_{{p,q}}(\beta, f_q)$</td>
<td>$p \in P_{\alpha \beta}$</td>
</tr>
<tr>
<td>$e_{{p,q}}$</td>
<td>$V_{{p,q}}(\alpha, \beta)$</td>
<td>${p,q} \in N, p \in P_{\alpha \beta}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>edge</th>
<th>weight</th>
<th>for</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^\alpha_p$</td>
<td>$\infty$</td>
<td>$p \in P_{\alpha}$</td>
</tr>
<tr>
<td>$t^\beta_p$</td>
<td>$D_p(f_p)$</td>
<td>$p \notin P_{\alpha}$</td>
</tr>
<tr>
<td>$t^\alpha_p$</td>
<td>$D_p(\alpha)$</td>
<td>$p \in P$</td>
</tr>
<tr>
<td>$e_{{p,q}}$</td>
<td>$V_{{p,q}}(f_p, \alpha)$</td>
<td>${p,q} \in N, f_p \neq f_q$</td>
</tr>
<tr>
<td>$e_{{p,q}}$</td>
<td>$V_{{p,q}}(\alpha, f_q)$</td>
<td>${p,q} \in N, f_p = f_q$</td>
</tr>
</tbody>
</table>
c. Write one of the following functions, or both for extra credit:

```
[finalAssignment] = minimizeEnergyUsingSwap( ... 
    f0, dataTerm, edgeList, smoothTerm, labels)
```

```
[finalAssignment] = minimizeEnergyUsingExpansion( ... 
    f0, dataTerm, edgeList, smoothTerm, labels)
```

Where:

- \( f_0 \) – a MNx1 vector of initial guess for indices of the disparities
- \( \text{labels} \) - a 1xN_d vector of possible indices of the disparities
- \( \text{finalAssignment} \) – a MNx1 vector of indices of the final disparities
- The rest of the parameters are as before

These functions should iteratively call \( \text{alphaBetaSwap} / \text{alphaExpand} \), in the following form:

1. Start with an arbitrary labeling
2. Alpha-beta swap: For each pair of labels perform a swap move or
   Alpha expansion: For each label perform an expansion move
3. If the move improved the previous assignment (lower energy), keep the new assignment.
4. If any of these moves improved the assignment, repeat from step 2.
Question 3

a. Download the Tsukuba dataset found on the course web site. The dataset consists of 2 images and ground truth disparity. Run minimizeEnergyUsingSwap or minimizeEnergyUsingExpansion from question 2 (both if you wrote both) on this dataset with an arbitrary initial guess (e.g. all disparities are 0). Use the set of possible disparities (labels) [0:14]. Use the Potts model for the smoothness term. Save your result to q3aTsukubaSwap / q3aTsukubaExpansion variable. Compare your result to ground truth and save the comparison to q3aTsukubaDiffSwap / q3aTsukubaDiffExpansion variable. Produce a 1x3 image array named q3adisplaySwap / q3adisplayExpansion which consists of the disparity maps:

- Left image – your result
- Center image – the difference between your result and ground truth
- Right image – ground truth

Save this image to a q3adisplaySwap.png / q3adisplayExpansion.png file.

When producing the results, you can omit 14 pixels from the right side of the disparity map.

Compute the % of errors when comparing to ground truth:

error1Q3aSwap / error1Q3aExpansion = #pixels in result which differ from ground truth / #pixels in image

and also compute:

error2Q3aSwap / error2Q3aExpansion = #pixels in result which differ from ground truth by more than 1 / #pixels in image

ignore pixels in the boundary or at which the ground truth values are zero.

If you wrote both swap and expansion algorithms, are there differences in the results?

b. Download the cones dataset found at the course web site. The dataset consists of 2 images and ground truth disparity. Run minimizeEnergyUsingSwap or minimizeEnergyUsingExpansion from question 2 (both if you wrote both) on this dataset with an arbitrary initial guess (e.g all disparities are 0). Use the set of possible disparities (labels) [0:54]. Use both Potts and truncated L1 model. Save your results to q3bConesPottsSwap / q3bConesPottsExpansion and q3bConesL1Swap / q3bConesL1Expansion variables. Produce a 2x3 image array named q3bdisplaySwap / q3bdisplayExpansion which consists of 2 images:
Top row:
  - Left image – Potts model result
  - Center image – the difference between your result and ground truth
  - Right image – ground truth

Bottom row – the same for truncated L1 model

Save this image to a `q3bdisplaySwap.png / q3bdisplayExpansion.png` file.

When producing the results, you can omit 54 pixels from the right side of the disparity map.

Compute the % of errors as in paragraph a, save results to 4 or 8 files of the form `error[1/2]Q3b[Potts/L1][Swap/Expansion]`, i.e. `error1Q3bPottsSwap`.

Which smoothness model performs better?

If you wrote both swap and expansion algorithms, are there differences in the results?

**Submission instructions:**

Submit one `.zip` archive called:

`CVFall2010_ex3_Name1_FamilyName1_Name2_FamilyName2.zip`

by sending an email to nimrod.dorfman@weizmann.ac.il with the subject "CVFall2010 ex3". This zip archive should contain:

1. A brief discussion of the results in pdf, ps or rtf format. The 1st page should include the names of the submitters and your ID numbers.
2. A mat file called `ex3.mat` which includes the disparity maps of question 3
3. All the png images from question 3, comparing your results to the ground truth.
4. All m files used for implementation of functions in questions 1,2
5. All m files used for testing your code in question 3
6. Please submit a hard copy of your code and report to the "computer vision" mailbox.

Good luck!