MST Stereo Matching Algorithm

What we need at beginning:
Filename of left image
Filename of right image
Max disparity (disparity range)
Scale (used to enhance the final disparity image usually scale = 256 / max_disparity)

Input: Left & Right (RGB) image (*.ppm)

unsigned char*** left, ***right; // Range: left[0~height-1][0~width-1][0~2]
left[y][x][0], left[y][x][1] and left[y][x][2] represent R, G, B value (0~255) of pixel in yth row xth line individually.

Function: qx_ppm.h

unsigned char*** loadimage_ppm_u(char* filename, int &h, int&w);

Usage

int h, w;
char* filename = “left.ppm”;
unsigned char*** left = loadimage_ppm_u(filename, h, w);

Output: disparity (gray-scale) image (*.pgm)

unsigned char** disparity_left; // Range: disparity_left[0~height-1][0~width-1];
disparity_left[y][x] represents the disparity value of the pixel in yth row xth line. (0~255).

Function: qx_ppm.h

void saveimage_pgm(char* filename, unsigned char** image, int h, int w, int scale);
// scale is used to enhance the gray-scale image, making it clear to recognize.

Stereo Matching Algorithm

1st step: construct minimum spanning tree of left image / right image

Node: every pixel

Edge: edges that connect neighboring nodes.

Weight: for neighboring node p, q in the left image:

Left Tree:

```
    dR = abs(left[yp][xp][0] – left[yq][xq][0]);
    dG = abs(left[yp][xp][1] – left[yq][xq][1]);
    dB = abs(left[yp][xp][2] – left[yq][xq][2]);
    W = max(dR, dG, dB);
```

Use unsigned char* **weight_left** to store W between the node and its parent.
(weight_left[node_id] = w)

Weight: for neighboring node p, q in the left image:

Right Tree:

```
    dR = abs(right[yp][xp][0] –right [yq][xq][0]);
    dG = abs(right [yp][xp][1] –right [yq][xq][1]);
    dB = abs(right [yp][xp][2] –right [yq][xq][2]);
    W = max(dR, dG, dB);
```

Use unsigned char* **weight_right** to store W between the node and its parent.

2nd step: compute matching cost for each pixel at disparity level d.

(Do for left image and right image at the same time)

Input: unsigned char*** left, unsigned char*** right, **float**left_gradient, float** right_gradiant;**

        // range: left_gradient[0~height-1][0~width-1]

// As for how to compute left_gradient, right_gradient,

// please check **qx_nonlocal_cost_aggregation.h**

// void compute_grandient(float** gradient, unsigned char*** image)

// Usage: compute_gradient(left_gradient, left);

//   compute_gradient(right_gradient, right);

Output: double* cost_vol_left, ***cost_vol_right;**

// range: cost_vol_left[0~height-1][0~width-1][0~max_disparity-1]

// do for cost_vol_left

```
for d = 0 to max_disparity - 1
    for y = 0 to height – 1
```

```
        for x = 0 to width – 1
            for c = 0 to 2
                double cost = 0;
                cost += abs (left[y][x][c] – right[y][max(0, x-d)][c]);

            cost = min(cost_vol_left[y][x][d] / 3, max_color_difference);
            // double max_color_difference = 7.0

            double cost_gradient =
                    min ((double)abs(gradient_left[y][x] - gradient_right[y][max(0,x-d)]),
                         max_gradient_color_difference);
            // double max_gradient_color_difference = 2.0
            cost_vol_left[y][x][d] = weight_on_color * cost +
                                  (1 – weight_on_color) * cost_gradient;
            // weight_on_color = 0.11

// update cost_vol_right
    for y = 0 to height – 1
        for x = 0 to width – max_disparity - 1
            for d = 0 to max_disparity_range
                cost_vol_right[y][x][d] = cost_vol_left[y][x+d][d];

        for x = width – max_disparity to width – 1
            for d = 0 to max_disprity_range
                if ( x + d < width) cost_vol_right[y][x][d] = cost_vol_left[y][x+d][d];
                else cost_vol_right[y][x][d] = cost_vol_right[y][x][d-1];
```

3rd: update aggregated matching cost

*Left image: aggregate matching cost **cost_vol_left** on the **left_tree***

Input: cost_vol_left (double*) // Obtained from 2nd step**

Output: cost_vol_left (double***)

```
Unsigned double*** cost _backup =
memcpy(cost_backup, cost_vol_left, sizeof(double) * height * width * max_disparity);

// 1st part: from leaf node to the root node
for all node p in the left_tree
    for all children node q of p
        double w = e^ ((double) -1 * weight_left[q] / (MAX_CHAR * sigma));
        // MAX_CHAR = 255
        // sigma = 0.1
        // weight_left is used in the tree construction.
        for d = 0 to max_disparity – 1
            double value_p = cost_backup[yp][xp][d];
            double value_q = cost_backup[yq][xq][d];
            value_p += w * value_q;
            cost_backup[yp][xp][d] = value_p;

// 2nd part: from root node to the leaf node
for all node q in the left_tree
    for the parent node p of q
        double w = e^ ((double) -1 * weight_left[q] / (MAX_CHAR * sigma));
        for d = 0 to max_disparity – 1
            double value_q_current = cost_backup[yq][xq][d];
            double value_p = cost_vol_left [yp][xp][d]; //Not cost_backup!!!

            cost_vol_left[yq][xq][d] = w * (value_p – w * value_q_current) + value_q_current;

Return cost_vol_left
```

*Left image: aggregate matching cost **cost_vol_right** on the **right_tree***

Input: cost_vol_right (double*)**

```
Output: cost_vol_right (double***)
// Same as aggregating matching cost on left_tree.
// Use cost_vol_right, cost_backup, weight_right

Reture cost_vol_right
```

4th: find disparity for each pixel

```
unsigned char** disparity_left_tmp, unsigned char** disparity_left,
Left Image:
for y = 0 to height – 1
    for x = 0 to width – 1
        current_min = cost_vol_left[y][x][0];
        disparity_left_tmp[y][x] = 0;
        for d = 1 to max_disparity – 1
            if (cost_vol_left[y][x][d] < current_min)
                disparity_left_tmp[y][x] = d;
                current_min = cost_vol_left[y][x][d];
ctmf(disparity_left_tmp[0], disparity_left[0], width, height, width, width, 2.0, 1, height * width);
// As for ctmf, please check ctmf.h & ctmf.cpp

usigned char** disparity_right_tmp, unsigned char** disparity_right,
```

Right Image:

```
for y = 0 to height – 1
    for x = 0 to width – 1
        current_min = cost_vol_right[y][x][0];
        disparity_ right _tmp[y][x] = 0;
        for d = 1 to max_disparity – 1
            if (cost_vol_left[y][x][d] < current_min)
                disparity_ right _tmp[y][x] = d;
                current_min = cost_vol_ right[y][x][d];
ctmf(disparity_right_tmp[0], disparity_right[0], width, height, width, width, 2.0, 1, height * width);
```