OpenCV 学习笔记（18）双目测距与三维重建的 OpenCV 实现问题集锦（三）立体匹配与视差计算

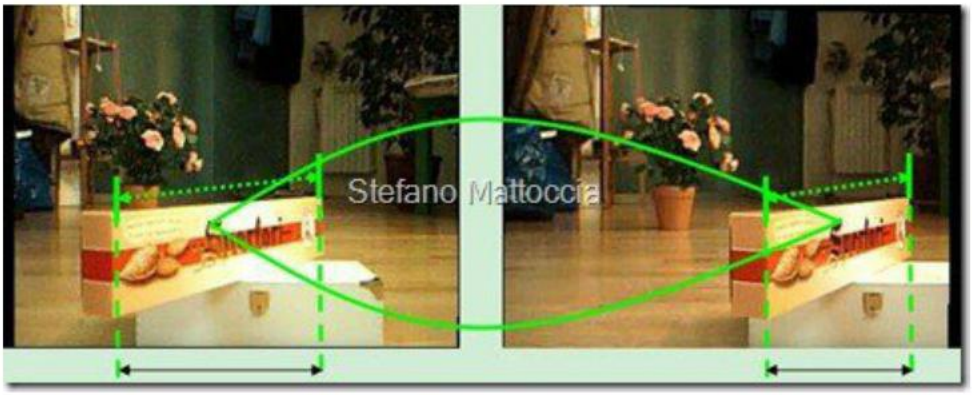
四、双目匹配与视差计算

立体匹配主要是通过找出每对图像间的对应关系，根据三角测量原理，得到视差图；在获得了视差信息后，根据投影模型很容易地可以得到原始图像的深度信息和三维信息。立体匹配技术被普遍认为是立体视觉中最困难也是最关键的问题，主要是以下因素的影响：

（1） 光学失真和噪声（亮度、色调、饱和度等失衡）

（2） 平滑表面的镜面反射

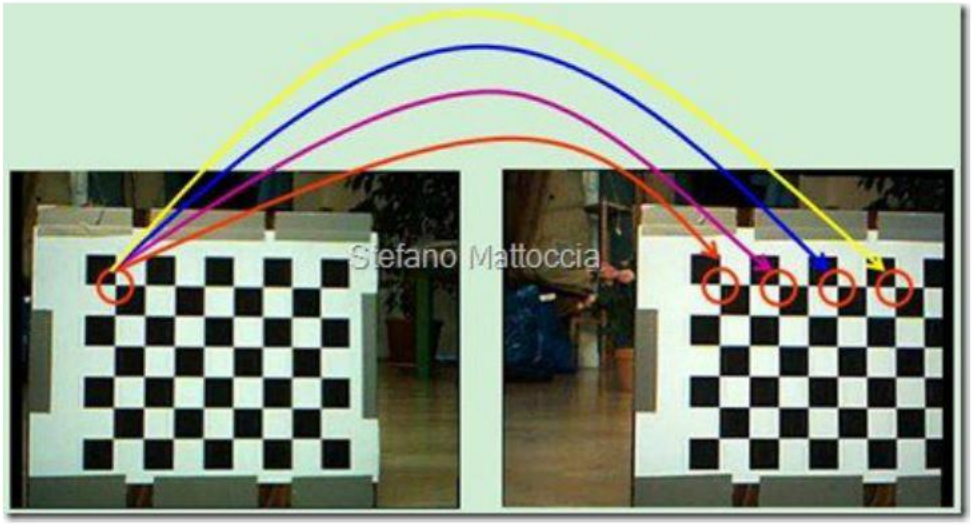
（3） 投影缩减（Foreshortening）



（4） 透视失真（Perspective distortions）

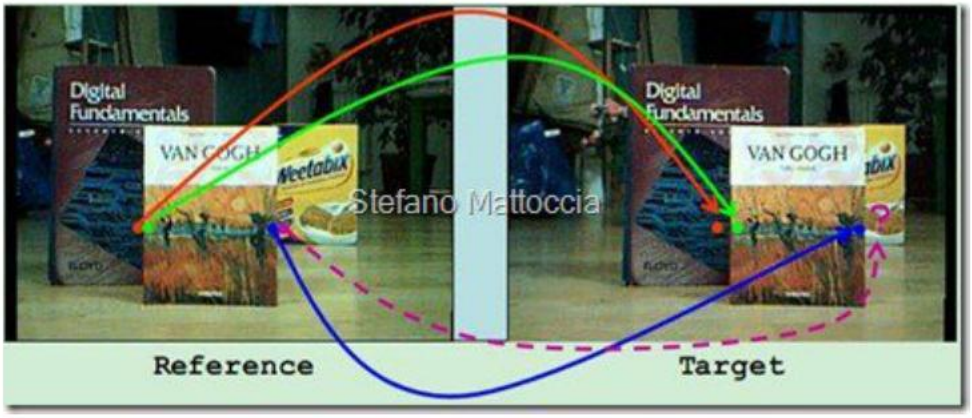
（5） 低纹理（Low texture）

（6） 重复纹理（Repetitive/ambiguous patterns）



（7） 透明物体

（8） 重叠和非连续



目前立体匹配算法是计算机视觉中的一个难点和热点，算法很多，但是一般的步骤是：

A、匹配代价计算

匹配代价计算是整个立体匹配算法的基础，实际是对不同视差下进行灰度相似性测量。常见的方法有灰度差的平方 SD（squared intensity differences），灰度差的绝对值 AD（absolute intensity differences）等。另外，在求原始匹配代价时可以设定一个上限值，来减弱叠加过程中的误匹配的影响。以 AD 法求匹配代价为例，可用下式进行计 算，其中 T 为设定的阈值。

$$C(x_i, y_i) = \begin{cases} |I_L(x_i) - I_R(y_i)|, & |I_L(x_i) - I_R(y_i)| < T \\ T, & |I_L(x_i) - I_R(y_i)| > T \end{cases}$$

图 18

B、 匹配代价叠加

一般来说，全局算法基于原始匹配代价进行后续算法计算。而区域算法则需要通过窗口叠加来增强匹配代价的可靠性，根据原始匹配代价不同，可分为：

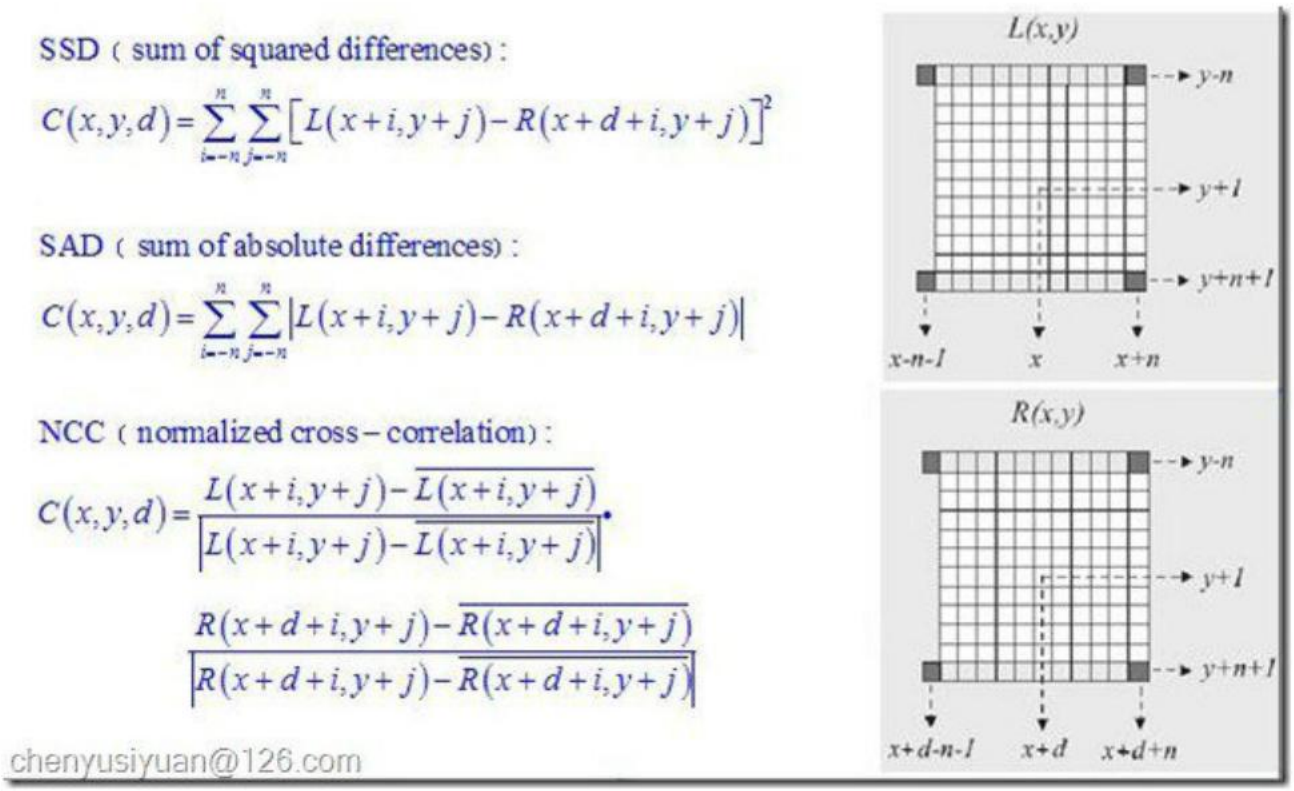


图 19

C、 视差获取

对于区域算法来说，在完成匹配代价的叠加以后，视差的获取就很容易了，只需在一定范围内选取叠加匹配代价最优的点（SAD 和 SSD 取最小值，NCC 取最大值）作为对应匹配点，如胜者为王算法 WTA（Winner-take-all）。而全局算法则直接对原始匹配代价进行处理，一般会先给出一个能量评 价函数，然后通过不同的优化算法来求得能量的最小值，同时每个点的视差值也就计算出来了。

D、视差细化（亚像素级）

大多数立体匹配算法计算出来的视差都是一些离散的特定整数值，可满足一般应用的精度要求。但在一些精度要求比较高的场合，如精确的三维重构中，就需要在初始视差获取后采用一些措施对视差进行细化，如匹配代价的曲线拟合、图像滤波、图像分割等。

有关立体匹配的介绍和常见匹配算法的比较，推荐大家看看 Stefano Mattoccia 的讲义 Stereo Vision: algorithms and applications，190 页的 ppt，讲解得非常形象详尽。

1． opencv2.1 和 opencv2.0 在做 stereo vision 方面有什么区别了？

2.1 版增强了 Stereo Vision 方面的功能：

(1) 新增了 SGBM 立体匹配算法（源自 Heiko Hirschmuller 的《Stereo Processing by Semi-global Matching and Mutual Information》）， 可以获得比 BM 算法物体轮廓更清晰的视差图（但低纹理区域容易出现横/斜纹路，在 GCstate->fullDP 选项使能时可消减这种异常纹路，但对应区域视差变为 0，且运行速度会有所下降），速度比 BM 稍慢， 352*288 的帧处理速度大约是 5 帧/秒；

(2) 视差效果：BM < SGBM < GC；处理速度：BM > SGBM > GC ；

(3) BM 算法比 2.0 版性能有所提升，其状态参数新增了对左右视图感兴趣区域 ROI 的支持（roi1 和 roi2，由 stereoRectify 函数产生）；

(4) BM 算法和 GC 算法的核心代码改动不大，主要是面向多线程运算方面的（由 OpenMP 转向 Intel TBB）；

(5) cvFindStereoCorrespondenceBM 函数的 disparity 参数的数据格式新增了 CV_32F 的支持，这种格式的数据给出实际视差，而 2.0 版只支持 CV_16S，需要除以 16.0 才能得到实际的视差数值。

2. 用于立体匹配的图像可以是彩色的吗？

在 OpenCV2.1 中，BM 和 GC 算法只能对 8 位灰度图像计算视差，SGBM 算法则可以处理 24 位（8bits*3）彩色图像。所以在读入图像时，应该根据采用的算法来处理图像：

```
int color_mode = alg == STEREO_SGBM ? 1 : 0;
//////////////////////////////////////////////////////////////////////////
// 载入图像
cvGrabFrame( lfCam );
cvGrabFrame( riCam );
frame1 = cvRetrieveFrame( lfCam );
frame2 = cvRetrieveFrame( riCam );
if(frame1.empty()) break;
resize(frame1, img1, img_size, 0, 0);
resize(frame2, img2, img_size, 0, 0);
// 选择彩色或灰度格式作为双目匹配的处理图像
if (!color_mode && cn>1)
{
cvtColor(img1, img1gray, CV_BGR2GRAY);
cvtColor(img2, img2gray, CV_BGR2GRAY);
img1p = img1gray;
img2p = img2gray;
}
else
{
img1p = img1;
img2p = img2;
}
```

3. 怎样获取与原图像有效像素区域相同的视差图？

在 **OpenCV2.0 及以前的版本中，所获取的视差图总是在左侧和右侧有明显的黑色区域**，这些区域没有有效的视差数据。视差图有效像素区域**与视差窗口（ndisp，一般取正值且能被 16 整除）和最小视差值（mindisp，一般取 0 或负值）相关**，视差窗口越大，视差图左侧的黑色区域越大，最小视差值越小，视差图右侧的黑色区域越大。其原因是为了保证参考图像（一般是左视图）的像素点能在目标图像（右视图）中按照设定的视差匹配窗口匹配对应点，**OpenCV 只从参考图像的第（ndisp－1＋mindisp）列开始向右计算视差，第 0 列到第（ndisp－1＋mindisp）列的区域视差统一设置为（mindisp－1）*16；视差计算到第 width＋mindisp 列时停止，余下的右侧区域视差值也统一设置为（mindisp－1）*16**。

```
00177 static const int DISPARITY_SHIFT = 4;
…
00411     int ndisp = state->numberOfDisparities;
00412     int mindisp = state->minDisparity;
00413     int lofs = MAX(ndisp - 1 + mindisp, 0);
00414     int rofs = -MIN(ndisp - 1 + mindisp, 0);
00415     int width = left->cols, height = left->rows;
00416     int width1 = width - rofs - ndisp + 1;
…
00420     short FILTERED = (short)((mindisp - 1) << DISPARITY_SHIFT);
…
00466     // initialize the left and right borders of the disparity map
00467     for( y = 0; y < height; y++ )
00468     {
00469         for( x = 0; x < lofs; x++ )
00470             dptr[y*dstep + x] = FILTERED;
00471         for( x = lofs + width1; x < width; x++ )
00472             dptr[y*dstep + x] = FILTERED;
00473     }
00474     dptr += lofs;
00475
00476     for( x = 0; x < width1; x++, dptr++ )

…
```

这样的设置很明显是不符合实际应用的需求的，它相当于把摄像头的视场范围缩窄了。因此，**OpenCV2.1** 做了明显的改进，不再要求左右视图和视差图的大小（size）一致，**允许对视差图进行左右边界延拓**，这样，虽然计算视差时还是按上面的代码思路来处理左右边界，但是视差图的边界得到延拓后，有效视差的范围就能够与对应视图完全对应。具体的实现代码范例如下：

```
//////////////////////////////////////////////////////////////////////////
// 对左右视图的左边进行边界延拓，以获取与原始视图相同大小的有效视差区域
copyMakeBorder(img1r, img1b, 0, 0, m_nMaxDisp, 0, IPL_BORDER_REPLICATE);
copyMakeBorder(img2r, img2b, 0, 0, m_nMaxDisp, 0, IPL_BORDER_REPLICATE);

//////////////////////////////////////////////////////////////////////////
// 计算视差
if( alg == STEREO_BM )
{
        bm(img1b, img2b, dispb);
        // 截取与原始画面对应的视差区域（舍去加宽的部分）
        disp1f = dispb.colRange(m_nMaxDisp, img1b.cols);
}
else if(alg == STEREO_SGBM)
{
        sgbm(img1b, img2b, dispb);
        disp1f = dispb.colRange(m_nMaxDisp, img1b.cols);
}
```

4. cvFindStereoCorrespondenceBM 的输出结果好像不是以像素点为单位的视差？

“@scyscyao：在 OpenCV2.0 中，BM 函数得出的结果是以 16 位符号数的形式的存储的，出于精度需要，所有的视差在输出时都扩大了 16 倍(2^4)。其具体代码表示如下：

```
dptr[y*dstep] = (short)(((ndisp - mind - 1 + mindisp)*256 + (d != 0 ? (p-n)*128/d : 0) + 15) >> 4);
```

可以看到，原始视差在左移 8 位(256)并且加上一个修正值之后又右移了 4 位，最终的结果就是左移 4 位。

因此，在实际求距离时，cvReprojectTo3D 出来的 X/W,Y/W,Z/W 都要乘以 16（也就是 W 除以 16），才能得到正确的三维坐标信息。”

在 **OpenCV2.1 中，BM 算法可以用 CV_16S 或者 CV_32F 的方式输出视差数据**，使用 32 位 float 格式可以得到真实的视差值，而 CV_16S 格式得到的视差矩阵则需要 除以 16 才能得到正确的视差。另外，OpenCV2.1 另外两种立体匹配算法 **SGBM 和 GC 只支持 CV_16S 格式的 disparity 矩阵**。

5. 如何设置 BM、SGBM 和 GC 算法的状态参数？

（1）StereoBMState

// 预处理滤波参数

- **preFilterType：预处理滤波器的类型**，主要是用于降低亮度失真（photometric distortions）、消除噪声和增强纹理等，有两种可选类型：CV_STEREO_BM_NORMALIZED_RESPONSE（归一化响应） 或者 **CV_STEREO_BM_XSOBEL（水平方向 Sobel 算子，默认类型）**，该参数为 int 型；
- **preFilterSize：预处理滤波器窗口大小**，容许范围是[5,255]，一般**应该在 5x5..21x21 之间**，参数必须为**奇数**值，int 型
- **preFilterCap：预处理滤波器的截断**值，预处理的输出值仅保留[-preFilterCap, preFilterCap]范围内的值，**参数范围：1 - 31（文档中是 31，但代码中是 63）**，int

// SAD 参数

- **SADWindowSize：SAD 窗口大小**，容许范围是[5,255]，一般**应该在 5x5 至 21x21 之间**，参数必须是**奇数**，int 型
- **minDisparity：最小视差，默认值为 0**，可以是负值，int 型
- **numberOfDisparities：视差窗口**，即最大视差值与最小视差值之差，窗口大小**必须是 16 的整数倍**，int 型

// 后处理参数

- **textureThreshold：低纹理区域的判断阈值**。如果当前 SAD 窗口内所有邻居像素点的 x 导数绝对值之和小于指定阈值，则该窗口对应的像素点的视差值为 0（That is, if the sum of absolute values of x-derivatives computed over SADWindowSize by SADWindowSize pixel neighborhood is smaller than the parameter, no disparity is computed at the pixel），该参数不能为负值，int 型
- **uniquenessRatio：视差唯一性百分比**， 视差窗口范围内最低代价是次低代价的(1 + uniquenessRatio/100)倍时，最低代价对应的视差值才是该像素点的视差，否则该像素点的视差为 0（the minimum margin in percents between the best (minimum) cost function value and the second best value to accept the computed disparity, that

is, accept the computed disparity d^ only if SAD(d) >= SAD(d^) x (1 + uniquenessRatio/100.) for any d != d*+/-1 within the search range ），该参数不能为负值，**一般 5-15 左右的值比较合适，** int 型

- **speckleWindowSize：检查视差连通区域变化度的窗口大小**，值为 0 时取消 speckle 检查，int 型
- **speckleRange：视差变化阈值**，当窗口内视差变化大于阈值时，该窗口内的视差清零，int 型

// OpenCV2.1 新增的状态参数

- **roi1, roi2：左右视图的有效像素区域**，一般**由双目校正阶段的 cvStereoRectify 函数传递**，也可以自行设定。一旦在状态参数中设定了 roi1 和 roi2，OpenCV 会通过 cvGetValidDisparityROI 函数计算出视差图的有效区域，在有效区域外的视差值将被清零。
- **disp12MaxDiff：左视差图（直接计算得出）和右视差图（通过 cvValidateDisparity 计算得出）之间的最大容许差异**。超过该阈值的视差值将被清零。该参数**默认为 -1**，即不执行左右视差检查。int 型。注意在程序调试阶段最好保持该值为 -1，以便查看不同视差窗口生成的视差效果。具体请参见《使用 OpenGL 动态显示双目视觉三维重构效果示例》一文中的讨论。

在上述参数中，对视差生成效果影响较大的**主要参数是 SADWindowSize、numberOfDisparities 和 uniquenessRatio 三个，一般只需对这三个参数进行调整，其余参数按默认设置即可**。

在 OpenCV2.1 中，BM 算法有 C 和 C++ 两种实现模块。

（2）StereoSGBMState

SGBM 算法的状态参数大部分与 BM 算法的一致，下面只解释不同的部分：

- **SADWindowSize：SAD 窗口大小**，容许范围是[1,11]，**一般应该在 3x3 至 11x11 之间**，参数必须是奇数，int 型
- **P1, P2：控制视差变化平滑性的参数**。P1、P2 的值越大，视差越平滑。P1 是相邻像素点视差增/减 1 时的惩罚系数；P2 是相邻像素点视差变化值大于 1 时的惩罚系数。**P2 必须大于 P1**。OpenCV2.1 提供的例程 **stereo_match.cpp 给出了 P1 和 P2 比较合适的数值**。
- **fullDP：布尔值**，当设置为 TRUE 时，运行双通道动态编程算法（full-scale 2-pass dynamic programming algorithm），会占用 O(W*H*numDisparities)个字节，对于高分辨率图像将占用较大的内存空间。**一般设置为 FALSE**。

注意 OpenCV2.1 的 SGBM 算法是用 C++ 语言编写的，没有 C 实现模块。与 H. Hirschmuller 提出的原算法相比，主要有如下变化：

1. 算法默认运行单通道 DP 算法，只用了 5 个方向，而 fullDP 使能时则使用 8 个方向（可能需要占用大量内存）。
2. 算法在计算匹配代价函数时，采用块匹配方法而非像素匹配（不过 SADWindowSize=1 时就等于像素匹配了）。
3. 匹配代价的计算采用 BT 算法（"Depth Discontinuities by Pixel-to-Pixel Stereo" by S. Birchfield and C. Tomasi），并没有实现基于互熵信息的匹配代价计算。
4. 增加了一些 BM 算法中的预处理和后处理程序。

（3）StereoGCState

GC 算法的状态参数只有两个：**numberOfDisparities 和 maxIters** ，并且**只能通过 cvCreateStereoGCState 在创建算法状态结构体时一次性确定**，不能在循环中更新状态信息。GC 算法并不是一种实时算法，但可以得到物体轮廓清晰准确的视差图，适用于静态环境物体的深度重构。

注意 **GC 算法只能在 C 语言模式下运行，并且不能对视差图进行预先的边界延拓**，左右视图和左右视差矩阵的大小必须一致。

6. 如何实现视差图的伪彩色显示？

首先要将 16 位符号整形的视差矩阵转换为 8 位无符号整形矩阵，然后按照一定的变换关系进行伪彩色处理。我的实现代码如下：

```
// 转换为 CV_8U 格式，彩色显示
dispLfcv = displf, dispRicv = dispri, disp8cv = disp8;
if (alg == STEREO_GC)
{
        cvNormalize( &dispLfcv, &disp8cv, 0, 256, CV_MINMAX );
}
else
{
        displf.convertTo(disp8, CV_8U, 255/(m_nMaxDisp*16.));
}
F_Gray2Color(&disp8cv, vdispRGB);
```

灰度图转伪彩色图的代码，主要功能是使灰度图中 亮度越高的像素点，在伪彩色图中对应的点越趋向于 红色；亮度越低，则对应的伪彩色越趋向于 蓝色；总体上按照灰度值高低，由红渐变至蓝，中间色为绿色。其对应关系如下图所示：

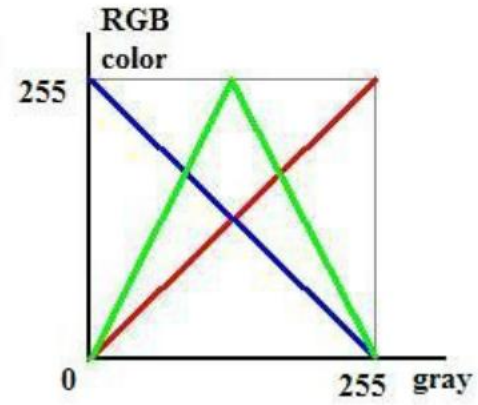


图 20

```
void F_Gray2Color(CvMat* gray_mat, CvMat* color_mat)
{
        if(color_mat)
                cvZero(color_mat);

        int stype = CV_MAT_TYPE(gray_mat->type), dtype = CV_MAT_TYPE(color_mat->type);
        int rows = gray_mat->rows, cols = gray_mat->cols;

        // 判断输入的灰度图和输出的伪彩色图是否大小相同、格式是否符合要求
        if (CV_ARE_SIZES_EQ(gray_mat, color_mat) && stype == CV_8UC1 && dtype == CV_8UC3)
        {
                CvMat* red = cvCreateMat(gray_mat->rows, gray_mat->cols, CV_8U);
                CvMat* green = cvCreateMat(gray_mat->rows, gray_mat->cols, CV_8U);
                CvMat* blue = cvCreateMat(gray_mat->rows, gray_mat->cols, CV_8U);
```

```
                CvMat* mask = cvCreateMat(gray_mat->rows, gray_mat->cols, CV_8U);

                // 计算各彩色通道的像素值
                cvSubRS(gray_mat, cvScalar(255), blue);       // blue(I) = 255 - gray(I)
                cvCopy(gray_mat, red);                  // red(I) = gray(I)
                cvCopy(gray_mat, green);                        // green(I) = gray(I),if
gray(I) < 128
                cvCmpS(green, 128, mask, CV_CMP_GE ); // green(I) = 255 - gray(I), if
gray(I) >= 128
                cvSubRS(green, cvScalar(255), green, mask);
                cvConvertScale(green, green, 2.0, 0.0);

                // 合成伪彩色图
                cvMerge(blue, green, red, NULL, color_mat);

                cvReleaseMat( &red );
                cvReleaseMat( &green );
                cvReleaseMat( &blue );
                cvReleaseMat( &mask );
        }
}
```

7. 如何将视差数据保存为 txt 数据文件以便在 Matlab 中读取分析？

由于 OpenCV 本身只支持 xml、yml 的数据文件读写功能，并且其 xml 文件与构建网页数据所用的 xml 文件格式不一致，在 Matlab 中无法读取。我们可以通过以下方式将视差数据保存为 txt 文件，再导入到 Matlab 中。

```
void saveDisp(const char* filename, const Mat& mat)
{
        FILE* fp = fopen(filename, "wt");
        fprintf(fp, "%02d\n", mat.rows);
        fprintf(fp, "%02d\n", mat.cols);
        for(int y = 0; y < mat.rows; y++)
        {
                for(int x = 0; x < mat.cols; x++)
                {
                        short disp = mat.at<short>(y, x); // 这里视差矩阵是 CV_16S 格式的，
故用 short 类型读取
                        fprintf(fp, "%d\n", disp); // 若视差矩阵是 CV_32F 格式，则用 float
类型读取
                }
        }
        fclose(fp);
}
```

相应的 Matlab 代码为：

```
function img = txt2img(filename)
data = importdata(filename);
r = data(1);    % 行数
c = data(2);    % 列数
disp = data(3:end); % 视差
vmin = min(disp);
vmax = max(disp);
disp = reshape(disp, [c,r])'; % 将列向量形式的 disp 重构为 矩阵形式
%  OpenCV 是行扫描存储图像，Matlab 是列扫描存储图像
%  故对 disp 的重新排列是首先变成 c 行 r 列的矩阵，然后再转置回 r 行 c 列
img = uint8( 255 * ( disp - vmin ) / ( vmax - vmin ) );
mesh(disp);
set(gca,'YDir','reverse');  % 通过 mesh 方式绘图时，需倒置 Y 轴方向
axis tight; % 使坐标轴显示范围与数据范围相贴合，去除空白显示区
```

显示效果如下：

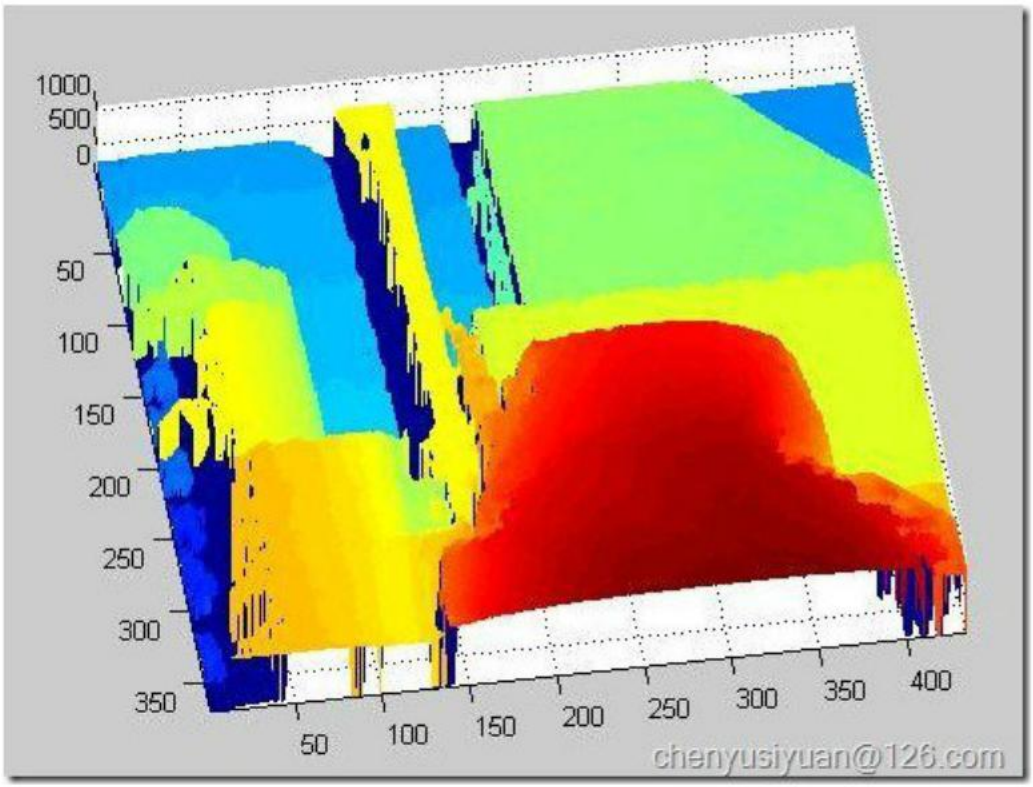


图 21

（待续……）

.

.