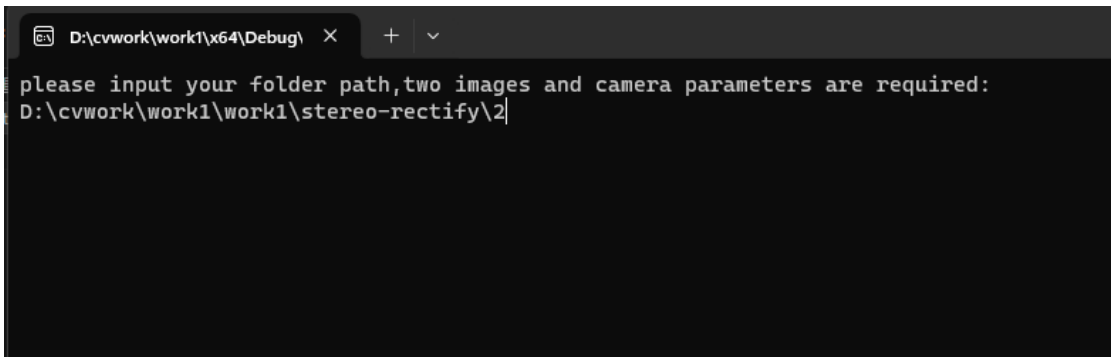
实验报告：双视图立体匹配与重建

一、 功能描述

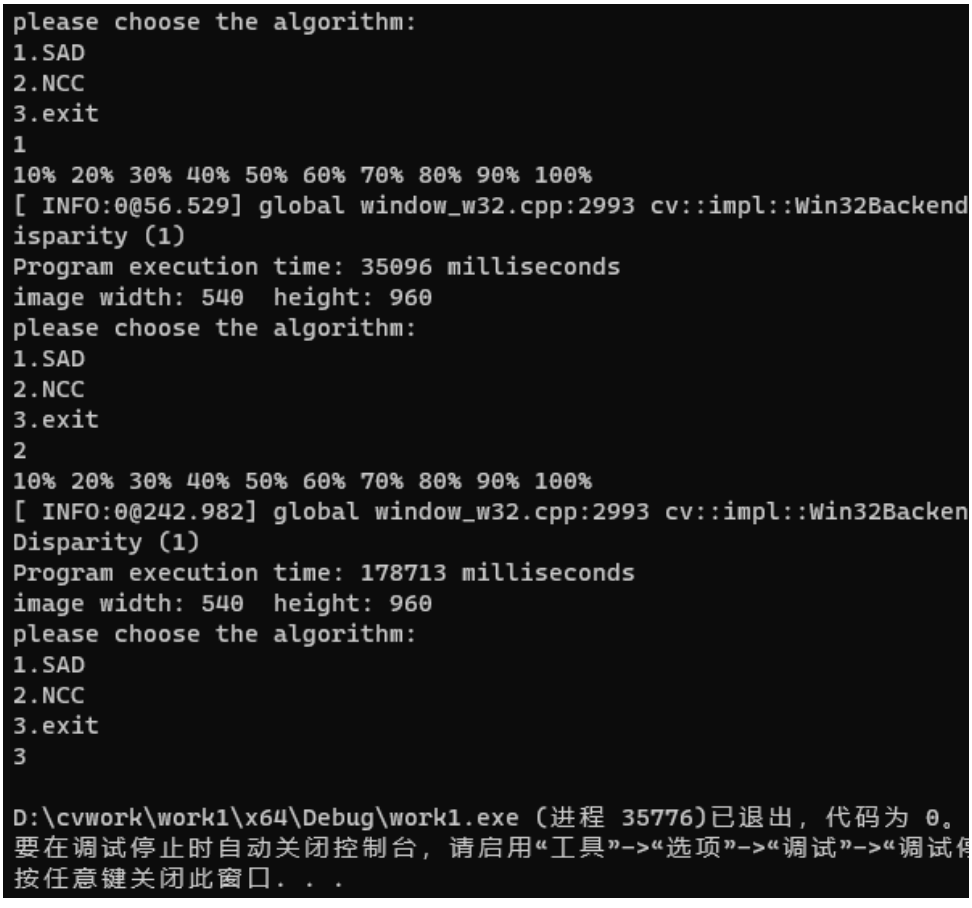
1. 交互部分：

a） 允许用户选择输入的图片



b） 允许用户选择立体匹配算法

c） 用户可重复选择算法进行对比，可自行选择退出程序



2. 程序输出：

a) 原始图像、矫正图像以及匹配视差图，

b) 控制台会输出程序执行时间、图像大小，同时会实时显示程序运行进度

二、 算法实现

1. 立体矫正

使用函数 stereorectify 来完成立体矫正，函数根据用户提供的相机参数（存储在 camerapara 类中），左右眼图片，通过 opencv 提供的 stereoRectify 函数完成矫正。

输出矫正好的左右眼图像（cv::Mat& rectified_left, cv::Mat& rectified_right）。

```
//input camera parameter,two image; output rectified_left,right_image
bool stereorectify(const camerapara& para, const cv::Mat& left_image, const cv::Mat& right_image, cv::Mat& rectified_left, cv::Mat& rectified_right) {

    //build extrinsic matrix
    cv::Mat left_rotation_matrix = para.leftRTMat(cv::Rect(0, 0, 3, 3)).clone();
    cv::Mat left_translation_vector = para.leftRTMat(cv::Rect(3, 0, 1, 3)).clone();

    cv::Mat right_rotation_matrix = para.rightRTMat(cv::Rect(0, 0, 3, 3)).clone();
    cv::Mat right_translation_vector = para.rightRTMat(cv::Rect(3, 0, 1, 3)).clone();

    //build camera matrix
    cv::Mat K_left = (cv::Mat_<double>(3, 3) << para.fx, 0, para.cx,
        0, para.fy, para.cy,
        0, 0, 1);
    cv::Mat K_right = K_left;

    cv::Mat R = right_rotation_matrix * left_rotation_matrix.t();
    cv::Mat T = -R * left_translation_vector + right_translation_vector;


    int width = left_image.cols;
    int height = right_image.rows;
    //stereoRectify
    cv::Mat R1, R2, P1, P2, Q;
    cv::Rect validROI[2];
    cv::stereoRectify(K_left, cv::Mat(), K_right, cv::Mat(),
        cv::Size(width, height), R, T,
        R1, R2, P1, P2, Q, cv::CALIB_ZERO_DISPARITY, 0, cv::Size(width, height), &validROI[0], &validROI[1]);

    cv::Mat map_left_x, map_left_y, map_right_x, map_right_y;
    cv::initUndistortRectifyMap(K_left, cv::Mat(), R1, P1, cv::Size(width, height), CV_32FC1, map_left_x, map_left_y);
    cv::initUndistortRectifyMap(K_right, cv::Mat(), R2, P2, cv::Size(width, height), CV_32FC1, map_right_x, map_right_y);

    cv::remap(left_image, rectified_left, map_left_x, map_left_y, cv::INTER_LINEAR);
    cv::remap(right_image, rectified_right, map_right_x, map_right_y, cv::INTER_LINEAR);

    return true;

}
```

2. SAD

SAD 算法（sum of absolute differences）在 SAD 类中的 computerSAD 函数中实现，函数接收左右眼标准立体视图，输出视差图。

SAD 的基本思想是对经行对准后的左右视图图像的对应像素块的对应像素的差的绝对值进行求和。这里使用大小为 winSize*winSize 的邻域窗口作为像素点的匹配窗口，设置大小为 DSR 的搜索区域，在搜索区域中找到与另一幅图中对应像素最匹配的像素（即 sad 值最小的像素）。然后计算其视差填入视差矩阵中。

sad 计算公式：

$$C(x,y,d)=\sum_{i<n}^{n}\sum_{j<n}^{n}|L(x+i,y+j)-R(x+d+i,y+j)|$$

实现代码：

```
//sum of absolute differences
Mat SAD::computerSAD(Mat& L, Mat& R) {
    int Height = L.rows;
    int Width = L.cols;
    Mat Kernel_L(Size(winSize, winSize), CV_8U, Scalar::all(0));
    Mat Kernel_R(Size(winSize, winSize), CV_8U, Scalar::all(0));
    Mat Disparity(Height, Width, CV_8U, Scalar(0)); //Parallax

    double complete = 0.1;

    for (int i = 0; i < Width - winSize; i++)
    {
        for (int j = 0; j < Height - winSize; j++)
        {
            Kernel_L = L(Rect(i, j, winSize, winSize));
            Mat MM(1, DSR, CV_32F, Scalar(0)); //

            for (int k = 0; k < DSR; k++)
            {
                //Calculate the sum of the absolute values of the difference between pixels in the kernel
                int x = i - k;
                if (x >= 0)
                {
                    Kernel_R = R(Rect(x, j, winSize, winSize));
                    Mat Dif;
                    absdiff(Kernel_L, Kernel_R, Dif);
                    Scalar ADD = sum(Dif);
                    float a = ADD[0];
                    MM.at<float>(k) = a;
                }

            }
            //The pixel with the lowest SAD value is the best match
            Point minLoc;
            minMaxLoc(MM, NULL, NULL, &minLoc, NULL);

            int loc = minLoc.x;//Parallax value

            Disparity.at<char>(j, i) = loc * 16;

        }
        if (i > complete* Width) {
            cout << complete * 100 << "% ";
            complete += 0.1;
        }
    }
```

3. NCC

NCC 算法（Normalized Cross-Correlation）在 NCC 类中的 computerNCC 函数中实现，函数接收左右眼标准立体视图，输出视差图。

同 SAD 一样需要设置邻域窗口大小和搜索范围，计算对应的 ncc 值，ncc 值在[-1,1]之间，越接近 1 表示越匹配：

ncc 计算公式：

$$NCC(p,d)=\frac{\sum_{(x,y)\in W_p}(I_1(x,y)-\overline{I1}(p_x,p_y))\bullet(I_2(x+d,y)-\overline{I}_2(p_x+d,p_y))}{\sqrt{\sum_{(x,y)\in W_p}(I_1(x,y)-\overline{I}_1(p_x,p_y))^2\bullet\sum_{(x,y)\in W_p}(I_2(x+d,y)-\overline{I}_2(p_x+d,p_y))^2}}$$

实现代码：

```
Mat NCC::computerNCC(Mat in1, Mat in2, bool add_constant = false)
{
    int width = in1.size().width;
    int height = in1.size().height;

    Mat left = bgr_to_grey(in1);
    Mat right = bgr_to_grey(in2);

    if (add_constant)
    {
        right += 10;
    }

    Mat depth(height, width, 0);
    vector< vector<double> > max_ncc; // store max NCC value

    for (int i = 0; i < height; ++i)
    {
        vector<double> tmp(width, -2);
        max_ncc.push_back(tmp);
    }

    double complete = 0.1;
    for (int offset = 1; offset <= max_offset; offset++)
    {
        Mat tmp(height, width, 0);

        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < offset; x++)
            {
                tmp.at<uchar>(y, x) = right.at<uchar>(y, x);
            }

            for (int x = offset; x < width; x++)
            {
                tmp.at<uchar>(y, x) = right.at<uchar>(y, x - offset);
            }
        }

        // calculate each pixel's NCC value
        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
```

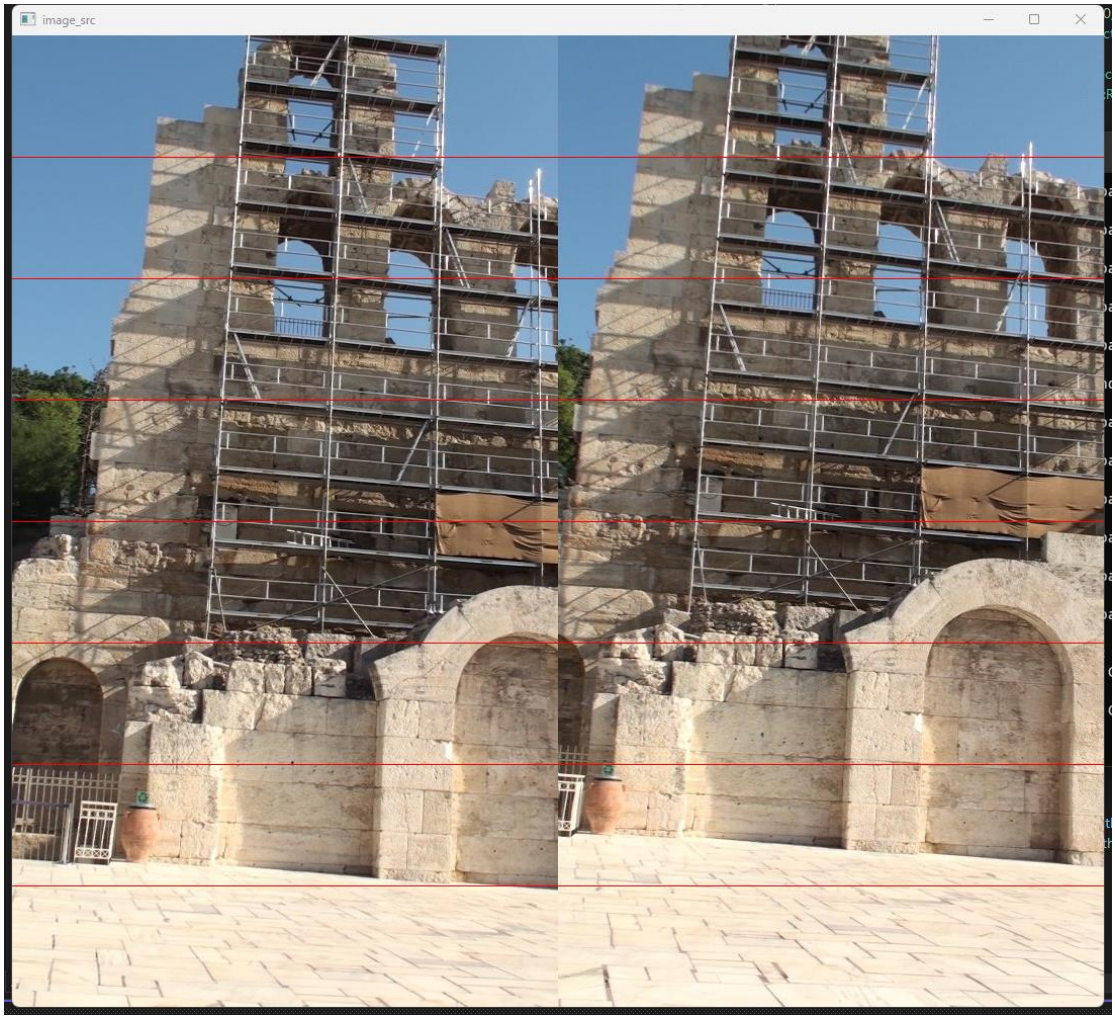
三、 实验结果

使用了提供的三组未矫正的立体图像对程序进行测试，结果如下：

1. 第一组：

图像大小：宽 540，高 960

a） 原始图像：



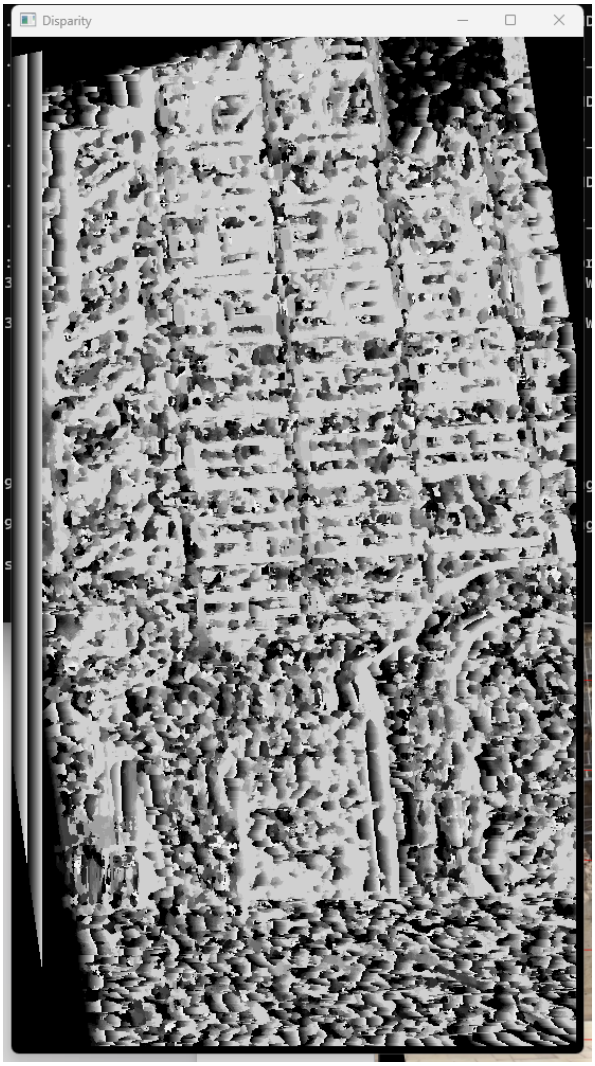
b） 矫正后图像：

c） SAD 算法

运行时间：66368 毫秒

```
Program execution time: 66368 milliseconds
```

视差图：

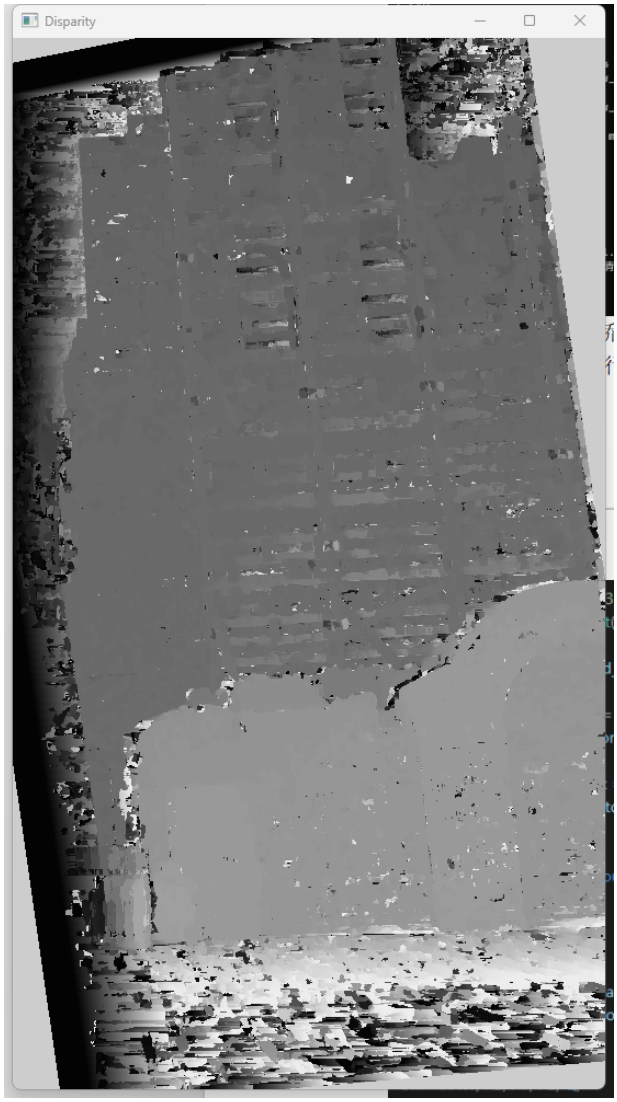


d) NCC

运行时间：228085 毫秒

```
Program execution time: 228085 milliseconds
image width: 540  height: 960
```

视差图

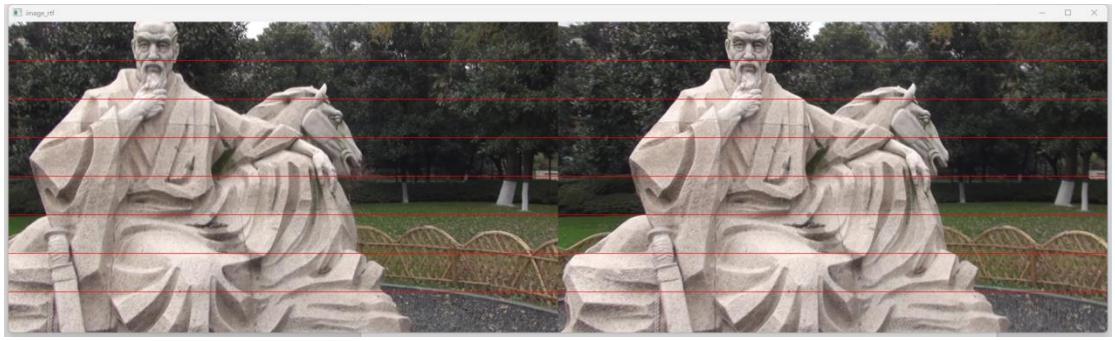


2. 第二组：

图像大小：宽 960，高 540

a) 原始图像：

b) 矫正后图像：

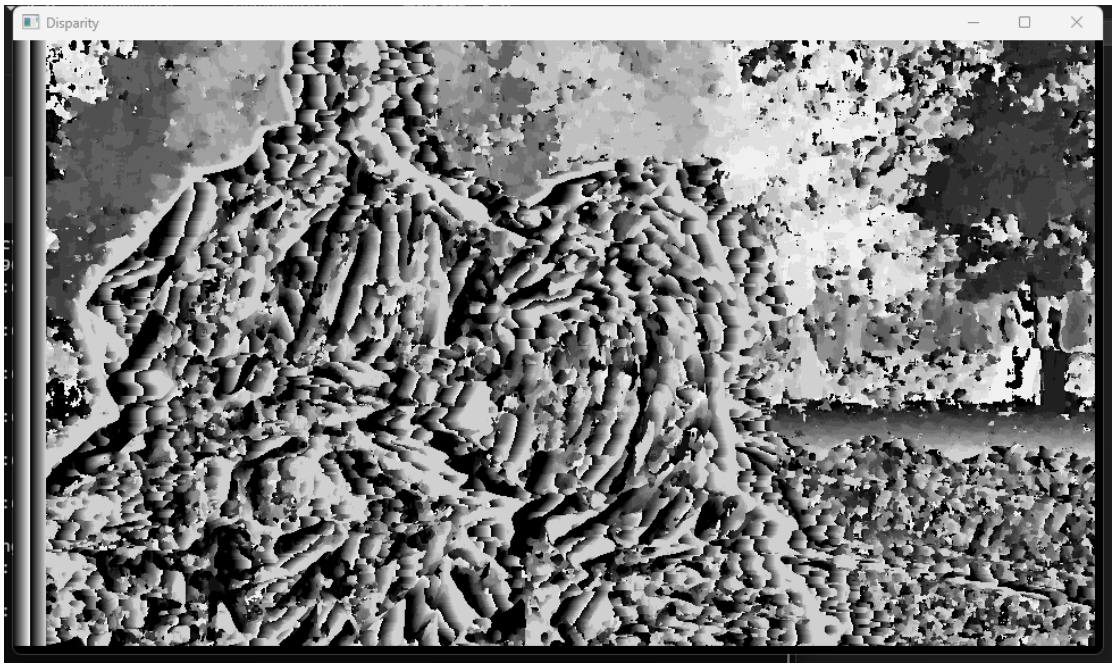


c) SAD

运行时间：35475 毫秒

```
Program execution time: 35475 milliseconds
```

匹配视差图

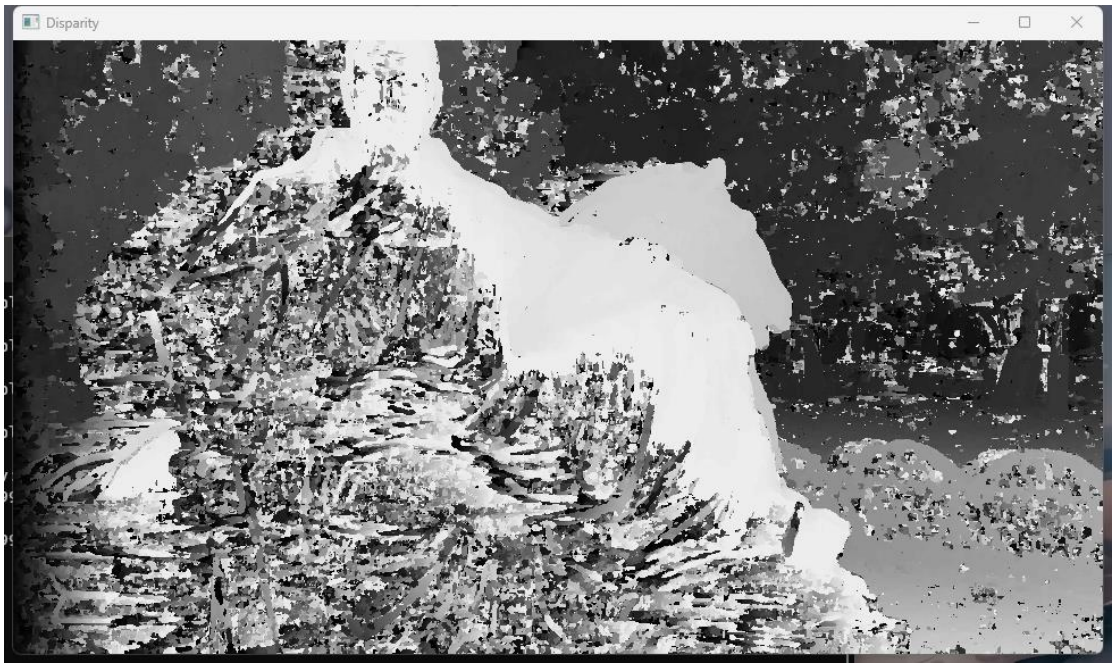


d) NCC

运行时间：184137 毫秒

```
Program execution time: 184137 milliseconds
```

匹配视差图

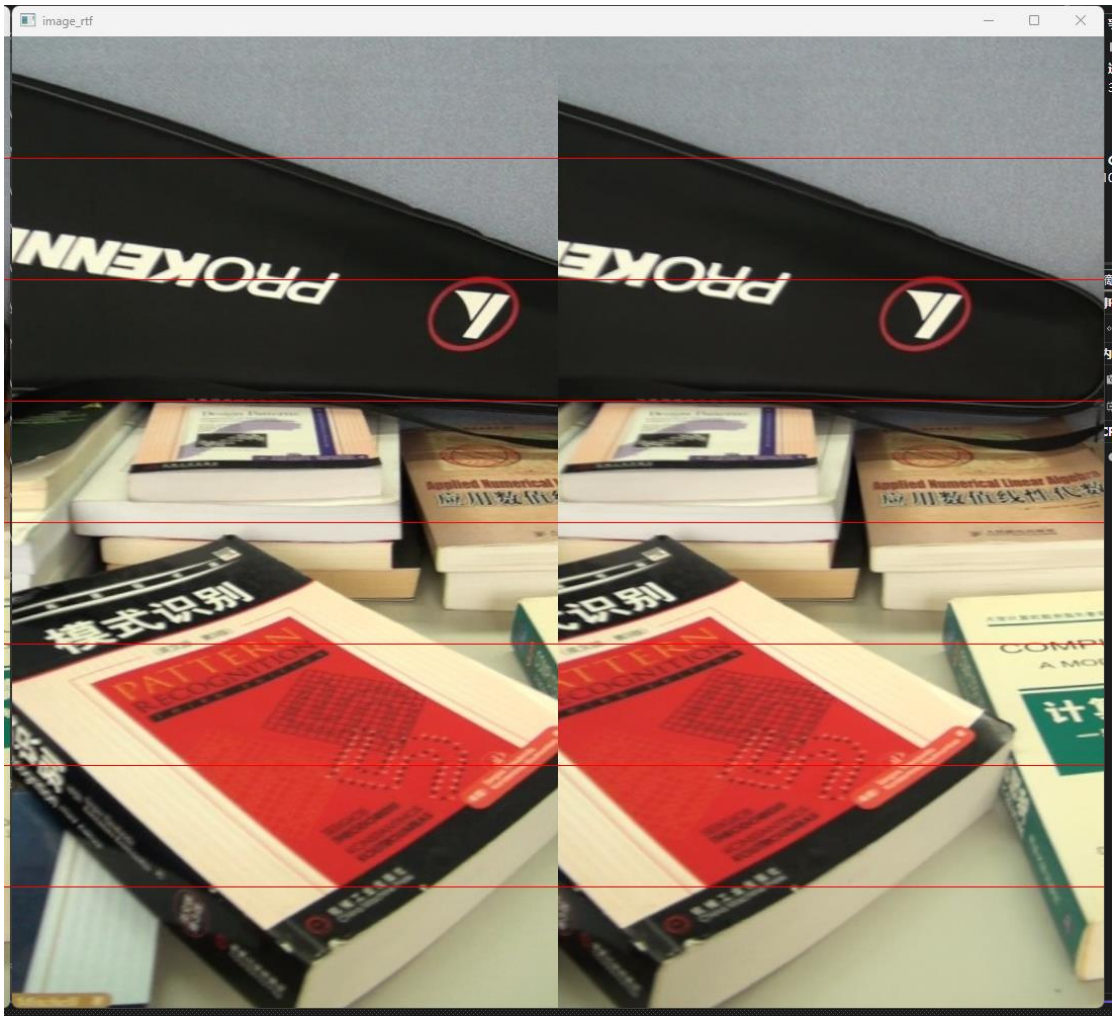


3. 第三组

图像大小：宽 960，高 540

a） 原始图像：

b） 矫正后图像：

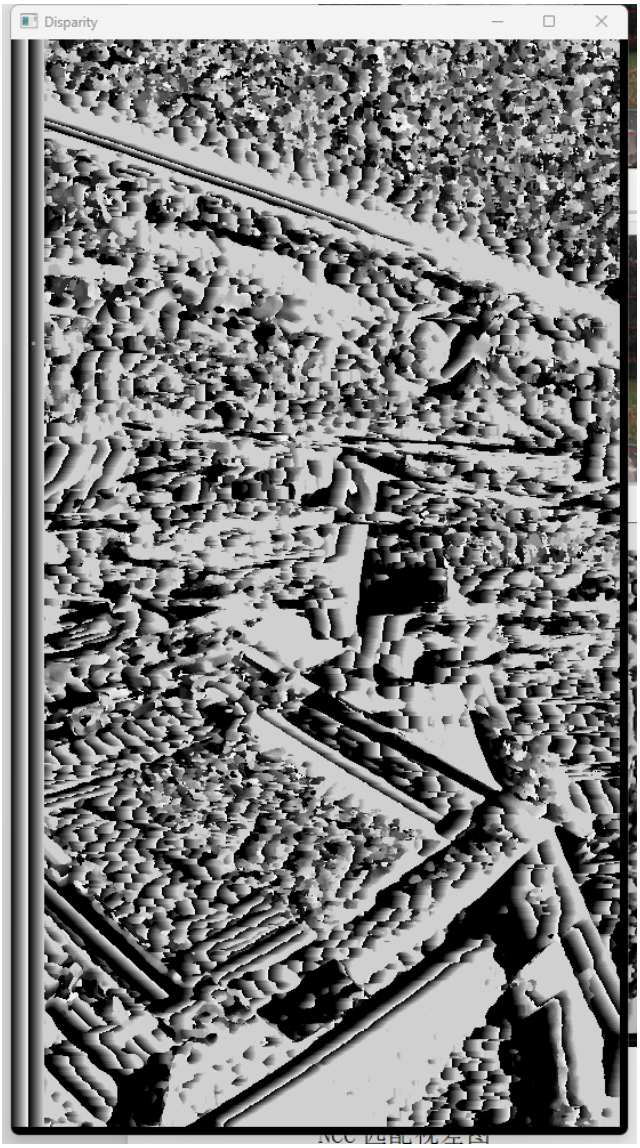


c） SAD

运行时间：35952 毫秒

```
Program execution time: 35952 milliseconds
```

匹配视差图：



d) NCC

运行时间：

```
Program execution time: 181522 milliseconds
```

匹配视差图：