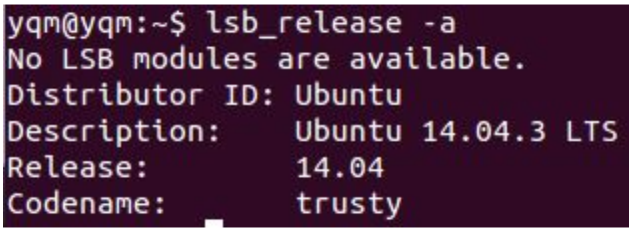
Stereo Matching

运行环境：



```
yqm@yqm:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.3 LTS
Release:        14.04
Codename:       trusty
```

Linux version 3.19.0-25-generic
Opencv 3.0
cmake version 2.8.12.2
gcc version 4.8.4
g++ (Ubuntu 4.8.4-2ubuntu1~14.04) 4.8.4

运行指令：

cmake .
make

运行说明：

实验把测例图片所在文件夹命名为 images，与源码放在同一根目录下

运行后，程序生成文件夹 resultImages，存放运行结果

文件说明：

运行的结果都储存“result”文件夹中，其中 resultImages_v0 是“ASW：size=11*11，rc=7,rp=36; SSD&NCC:size=5*5”跑出来的图片，resultImages_v1 是“ASW：size=11*11，rc=36, rp=7; SSD&NCC:size=5*5”跑的图片，然后文本文档是各部分运行的结果坏点率。

2.1 Basic Task

1.实现一个估计视差图坏像素率的函数

a) 在 disparity.h 中，声明估计函数

```
void Evaluate(Mat standard, Mat myMap);
```

b) 在 disparity.cpp 中实现函数，遍历标准图和自己生成的视差图作比较

```
for (int i = 0; i < standard.rows; i ++) {
        for (int j = 0; j < standard.cols; j ++) {
                ans = standard.ptr<uchar>(i)[j] - myMap.ptr<uchar>(i)[j];
                if (ans > 3 || ans < -3) count ++;
        }
}
```

c) 最后保留两位小数输出

```
cout <<setiosflags(ios::fixed);
cout <<setprecision(2) << result * 100 << '%' << endl;
```

2.用“Sum of Squared Difference (SSD)”来计算匹配代价，做视差匹配

a) 视差匹配

i. 匹配代价叠加

$$D_L(x,y) = \underset{d\in\{0,1,\ldots,d_{\max}\}}{\arg\min} \operatorname{dist}\left(F_L(x,y), F_R(x-d,y)\right)$$

部分代码实现：

设定一个 5*5 的滑块，当匹配左眼图的视差图 DL 时，对每一个 d 的取值都计算滑块方框内的左右视图同一水平，垂直位置相差 d 的像素差值的平方，与对应的 d 值一起存放到 vector 中，最后取最小 SSD 值对应的 d 值做为 DL (x, y) 的值。(最后还需要乘上三，增强显示效果)

函数：

把图片转成灰度图：

```
Mat turnIntoGray(Mat input)
```

取三通道平均值做为灰度值

```
int temp = 0;
for (int k = 0; k < 3; k ++)
		temp += input.at<Vec3b>(i, j)[k];
temp /= 3;
```

生成左右视差图：

```
Mat SSD(Mat Left, Mat Right, bool LeftOrRight, int size)
```

其中，Left、Right 分别时左右视图，布尔值 LeftOrRight=true 则返回左视差图，反之返回右视差图，size 表示滑动窗口 patch 的大小。

b) 不同的灰度值转换算法，优化

i. RGB 取平均值转为灰度图

Aloe 左视差图坏像素率为：24.83%

ii. 用 opencv 自带 API 实现彩色图转灰度图

```
Mat disparityLeft;
cvtColor(Left, disparityLeft, CV_BGR2GRAY);
```

Aloe 左视差图坏像素率为：24.68%

iii. 利用一个转化公式

Gray = R*0.299 + G*0.587 + B*0.114

Aloe 左视差图坏像素率为：24.72%

坏点率和灰度转化方式的关系不大。

c) 保存图片和坏点率到固定文件夹中 e.g.“resultImage”

i. 创建一个 string 数组保存文件夹目录

```
//文件夹下的目录
string name[FileNumber]= {"Aloe", "Baby1", "Baby2", "Baby3", "Bowling1",
"Bowling2", "Cloth1", "Cloth2", "Cloth3", "Cloth4", "Flowerpots",
"Lampshade1", "Lampshade2", "Midd1", "Midd2", "Monopoly",
"Plastic", "Rocks1", "Rocks2", "Wood1", "Wood2"};
```

ii. 对于每一个测例的处理 :生成一个对应的文件夹 ,把 disparity map

存入，并且计算坏点率

```
//在目的文件夹中创建相应的文件夹，以便存入图片
string str = "resultImages/" + name[i];
const char * dir = str.c_str();
mkdir(dir, S_IRWXU);
```

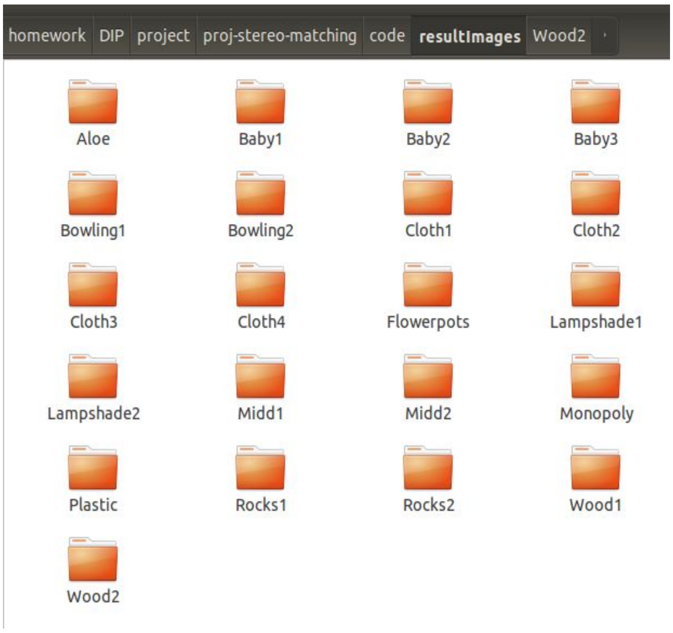
```
//get the disparity map and save it
Mat resultLeft = SSD(disparityLeft, disparityRight, true, 5);
imwrite( "resultImages/" + name[i] + "/" + name[i] + "_disp1_SSD.png", resultLeft );
Mat resultRight = SSD(disparityLeft, disparityRight, false, 5);
imwrite( "resultImages/" + name[i] + "/" + name[i] + "_disp5_SSD.png", resultRight );
```

```
//evaluate the quality of the disparity maps
Mat standardLeft = imread("images/" + name[i] + "/disp1.png", -1);
Evaluate(standardLeft, resultLeft);
Mat standardRight = imread("images/" + name[i] + "/disp5.png", -1);
Evaluate(standardRight, resultRight);
```

iii. 最后的坏点率

```
yqm@yqm:~
24.72%
25.78%
33.65%
33.31%
38.97%
39.25%
43.15%
43.08%
69.00%
69.41%
50.76%
50.46%
10.35%
8.47%
31.50%
31.86%
15.92%
14.38%
22.86%
24.85%
59.34%
59.46%
61.43%
60.47%
73.89%
73.35%
62.81%
62.50%
70.74%
71.26%
67.26%
65.99%
82.16%
82.99%
24.52%
```

```
23.80%
23.66%
21.92%
41.51%
40.64%
50.02%
50.35%
```



最后跑完全部测例大概 5 mins

3.Normalized Cross Correlation (NCC)计算匹配代价，做视差匹配

a）原理

$$NCC(x,y,d)=\frac{\sum_{i=-n}^{n}\sum_{j=-m}^{m} I_R(x+i,y+j)\,I_L(x+i,y+d+j)}{\sqrt{\sum_{i=-n}^{n}\sum_{j=-m}^{m} I_R^2(x+i,y+j)\sum_{i=-n}^{n}\sum_{j=-m}^{m} I_L^2(x+i,y+d+j)}}$$

b）函数实现

i. 函数原型

```
Mat NCC(Mat Left, Mat Right, bool LeftOrRight, int size);
```

Left：左视图灰度图

Right：右视图灰度图

LeftOrRight：为真，计算 DL；反之，计算 DR

Size：patch 窗口大小

ii. 套用公式计算

1. 计算分子

```
for (int x = i - size_; x <= i + size_; x ++) {
	for (int y = j - d - size_; y <= j - d + size_; y ++) {
		if (x < 0 || y < 0 || x >= row || y >= col || y + d < 0 || y + d >= col)continue;
		double temp1;
		double temp2;
		if (LeftOrRight) {
			temp1 = static_cast<double>(Right.ptr<uchar>(x)[y]);
			temp2 = static_cast<double>(Left.ptr<uchar>(x)[y + d]);
		}
		else {
			temp1 = static_cast<double>(Right.ptr<uchar>(x)[y + d]);
			temp2 = static_cast<double>(Left.ptr<uchar>(x)[y]);
		}
		tempZ += temp1 * temp2;
	}
}
```

2. 计算分母

```
double temp1 = 0;
double temp2 = 0;
for (int x = i - size_; x <= i + size_; x ++) {
	for (int y = j - d - size_; y <= j - d + size_; y ++) {
		if (x < 0 || y < 0 || x >= row || y >= col || y + d < 0 || y + d >= col)continue;
		if (LeftOrRight) {
			temp1 += pow(static_cast<double>(Left.ptr<uchar>(x)[y + d]), 2);
			temp2 += pow(static_cast<double>(Right.ptr<uchar>(x)[y]), 2);
		}
		else {
			temp1 += pow(static_cast<double>(Left.ptr<uchar>(x)[y]), 2);
			temp2 += pow(static_cast<double>(Right.ptr<uchar>(x)[y + d]), 2);
		}
	}
}
tempM = sqrt(temp1 * temp2);
```

3. 最后把结果和 d 一起存入 vector，取最大值作为 DL/DR 在该点的像素值，最后强度*3

```
double result = tempZ / tempM;
if (!LeftOrRight)d = -d;
v.push_back(make_pair(result, d));
```

4. 当 LeftOrRight=false 的时候表示计算右视图的视差图，此时 d 要取反

c） 运行结果储存在 resultImages 中：全部测例运行约 15 mins

d） 把 SSD 和 NCC 一起跑一遍

总耗时：15 mins 左右

坏点率，总体情况比 SSD 好：

Aloe_Left_SSD: 24.72%
Aloe_Left_NCC: 23.64%
Aloe_Right_SSD: 25.78%
Aloe_Right_NCC: 24.20%
Baby1_Left_SSD: 33.65%
Baby1_Left_NCC: 20.70%
Baby1_Right_SSD: 33.31%
Baby1_Right_NCC: 20.78%
Baby2_Left_SSD: 38.97%
Baby2_Left_NCC: 28.33%
Baby2_Right_SSD: 39.25%
Baby2_Right_NCC: 29.19%
Baby3_Left_SSD: 43.15%
Baby3_Left_NCC: 29.24%
Baby3_Right_SSD: 43.08%
Baby3_Right_NCC: 29.34%
Bowling1_Left_SSD: 69.00%
Bowling1_Left_NCC: 44.21%
Bowling1_Right_SSD: 69.41%
Bowling1_Right_NCC: 44.25%
Bowling2_Left_SSD: 50.76%
Bowling2_Left_NCC: 32.08%
Bowling2_Right_SSD: 50.46%
Bowling2_Right_NCC: 32.29%
Cloth1_Left_SSD: 10.35%
Cloth1_Left_NCC: 10.37%
Cloth1_Right_SSD: 8.47%

Cloth1_Right_NCC: 8.50%
Cloth2_Left_SSD: 31.50%
Cloth2_Left_NCC: 19.39%
Cloth2_Right_SSD: 31.86%
Cloth2_Right_NCC: 19.87%
Cloth3_Left_SSD: 15.92%
Cloth3_Left_NCC: 14.80%
Cloth3_Right_SSD: 14.38%
Cloth3_Right_NCC: 13.04%
Cloth4_Left_SSD: 22.86%
Cloth4_Left_NCC: 18.96%
Cloth4_Right_SSD: 24.85%
Cloth4_Right_NCC: 20.97%
Flowerpots_Left_SSD: 59.34%
Flowerpots_Left_NCC: 39.52%
Flowerpots_Right_SSD: 59.46%
Flowerpots_Right_NCC: 40.28%
Lampshade1_Left_SSD: 61.43%
Lampshade1_Left_NCC: 48.94%
Lampshade1_Right_SSD: 60.47%
Lampshade1_Right_NCC: 48.93%
Lampshade2_Left_SSD: 73.89%
Lampshade2_Left_NCC: 50.19%
Lampshade2_Right_SSD: 73.35%
Lampshade2_Right_NCC: 50.67%
Midd1_Left_SSD: 62.81%
Midd1_Left_NCC: 58.65%
Midd1_Right_SSD: 62.50%
Midd1_Right_NCC: 58.60%
Midd2_Left_SSD: 70.74%
Midd2_Left_NCC: 59.35%
Midd2_Right_SSD: 71.26%
Midd2_Right_NCC: 58.87%
Monopoly_Left_SSD: 67.26%
Monopoly_Left_NCC: 51.81%
Monopoly_Right_SSD: 65.99%
Monopoly_Right_NCC: 48.26%
Plastic_Left_SSD: 82.16%
Plastic_Left_NCC: 71.18%
Plastic_Right_SSD: 82.99%
Plastic_Right_NCC: 70.66%
Rocks1_Left_SSD: 24.52%
Rocks1_Left_NCC: 18.64%
Rocks1_Right_SSD: 23.80%

Rocks1_Right_NCC: 18.06%
Rocks2_Left_SSD: 23.66%
Rocks2_Left_NCC: 17.95%
Rocks2_Right_SSD: 21.92%
Rocks2_Right_NCC: 16.43%
Wood1_Left_SSD: 41.51%
Wood1_Left_NCC: 24.38%
Wood1_Right_SSD: 40.64%
Wood1_Right_NCC: 21.78%
Wood2_Left_SSD: 50.02%
Wood2_Left_NCC: 26.82%
Wood2_Right_SSD: 50.35%
Wood2_Right_NCC: 27.21%

4. Add a small constant amount of intensity (e.g. 10) to all right eye images, and re-run the above two methods.

a) 实现

b) 强度增加 5

i. Result

Aloe_Right_SSD: 32.88%
Aloe_Right_NCC: 24.31%
Baby1_Right_SSD: 60.14%
Baby1_Right_NCC: 21.17%
Baby2_Right_SSD: 58.17%
Baby2_Right_NCC: 29.60%
Baby3_Right_SSD: 69.26%
Baby3_Right_NCC: 30.17%
Bowling1_Right_SSD: 80.66%
Bowling1_Right_NCC: 44.73%
Bowling2_Right_SSD: 67.33%
Bowling2_Right_NCC: 32.88%
Cloth1_Right_SSD: 9.53%
Cloth1_Right_NCC: 8.53%
Cloth2_Right_SSD: 27.43%
Cloth2_Right_NCC: 20.49%
Cloth3_Right_SSD: 20.97%
Cloth3_Right_NCC: 13.19%
Cloth4_Right_SSD: 30.55%
Cloth4_Right_NCC: 24.97%
Flowerpots_Right_SSD: 79.68%

Flowerpots_Right_NCC: 41.67%
Lampshade1_Right_SSD: 81.28%
Lampshade1_Right_NCC: 49.50%
Lampshade2_Right_SSD: 83.64%
Lampshade2_Right_NCC: 51.42%
Midd1_Right_SSD: 70.54%
Midd1_Right_NCC: 59.01%
Midd2_Right_SSD: 64.57%
Midd2_Right_NCC: 59.47%
Monopoly_Right_SSD: 61.00%
Monopoly_Right_NCC: 48.36%
Plastic_Right_SSD: 82.50%
Plastic_Right_NCC: 71.16%
Rocks1_Right_SSD: 38.83%
Rocks1_Right_NCC: 18.30%
Rocks2_Right_SSD: 31.33%
Rocks2_Right_NCC: 16.64%
Wood1_Right_SSD: 63.96%
Wood1_Right_NCC: 22.28%
Wood2_Right_SSD: 80.38%
Wood2_Right_NCC: 27.56%

c) 强度增加 10

i. Result

Aloe_Right_SSD: 42.73%
Aloe_Right_NCC: 24.46%
Baby1_Right_SSD: 82.48%
Baby1_Right_NCC: 21.71%
Baby2_Right_SSD: 78.84%
Baby2_Right_NCC: 30.20%
Baby3_Right_SSD: 84.03%
Baby3_Right_NCC: 31.33%
Bowling1_Right_SSD: 90.46%
Bowling1_Right_NCC: 46.32%
Bowling2_Right_SSD: 81.67%
Bowling2_Right_NCC: 33.80%
Cloth1_Right_SSD: 16.46%
Cloth1_Right_NCC: 8.54%
Cloth2_Right_SSD: 60.61%
Cloth2_Right_NCC: 22.14%
Cloth3_Right_SSD: 43.07%
Cloth3_Right_NCC: 13.52%
Cloth4_Right_SSD: 50.16%

Cloth4_Right_NCC: 29.13%
Flowerpots_Right_SSD: 94.22%
Flowerpots_Right_NCC: 43.82%
Lampshade1_Right_SSD: 91.23%
Lampshade1_Right_NCC: 50.16%
Lampshade2_Right_SSD: 94.76%
Lampshade2_Right_NCC: 54.29%
Midd1_Right_SSD: 80.01%
Midd1_Right_NCC: 59.77%
Midd2_Right_SSD: 78.87%
Midd2_Right_NCC: 60.41%
Monopoly_Right_SSD: 57.41%
Monopoly_Right_NCC: 48.58%
Plastic_Right_SSD: 91.24%
Plastic_Right_NCC: 71.86%
Rocks1_Right_SSD: 59.16%
Rocks1_Right_NCC: 18.81%
Rocks2_Right_SSD: 56.84%
Rocks2_Right_NCC: 16.89%
Wood1_Right_SSD: 88.92%
Wood1_Right_NCC: 22.98%
Wood2_Right_SSD: 91.86%
Wood2_Right_NCC: 28.30%

d) 强度增加 20

i. Result

Aloe_Right_SSD: 79.63%
Aloe_Right_NCC: 28.20%
Baby1_Right_SSD: 95.23%
Baby1_Right_NCC: 23.29%
Baby2_Right_SSD: 93.84%
Baby2_Right_NCC: 34.08%
Baby3_Right_SSD: 95.11%
Baby3_Right_NCC: 34.43%
Bowling1_Right_SSD: 95.24%
Bowling1_Right_NCC: 52.17%
Bowling2_Right_SSD: 93.45%
Bowling2_Right_NCC: 36.86%
Cloth1_Right_SSD: 66.16%
Cloth1_Right_NCC: 8.61%
Cloth2_Right_SSD: 91.11%
Cloth2_Right_NCC: 24.42%

Cloth3_Right_SSD: 80.49%
Cloth3_Right_NCC: 15.86%
Cloth4_Right_SSD: 90.09%
Cloth4_Right_NCC: 40.31%
Flowerpots_Right_SSD: 96.57%
Flowerpots_Right_NCC: 48.61%
Lampshade1_Right_SSD: 96.70%
Lampshade1_Right_NCC: 55.43%
Lampshade2_Right_SSD: 97.61%
Lampshade2_Right_NCC: 59.49%
Midd1_Right_SSD: 87.14%
Midd1_Right_NCC: 62.31%
Midd2_Right_SSD: 86.96%
Midd2_Right_NCC: 62.72%
Monopoly_Right_SSD: 79.35%
Monopoly_Right_NCC: 49.33%
Plastic_Right_SSD: 96.06%
Plastic_Right_NCC: 73.44%
Rocks1_Right_SSD: 87.18%
Rocks1_Right_NCC: 20.43%
Rocks2_Right_SSD: 86.72%
Rocks2_Right_NCC: 17.88%
Wood1_Right_SSD: 95.84%
Wood1_Right_NCC: 25.18%
Wood2_Right_SSD: 95.17%
Wood2_Right_NCC: 30.74%

结论：在增加强度时，SSD 算法处理的视差图质量下滑严重，坏点率增加非常明显，NCC 算法的处理效果会好一些，坏点率前后对比增加幅度不大，但是也有所增加。

5.ASW 算法的实现

a) 算法实现

i. 原理

$$E(p,\bar{p}_d)=\frac{\sum_{q\in N_p,\bar{q}_d\in N_{\bar{p}_d}} w(p,q)w(\bar{p}_d,\bar{q}_d)e_0(q,\bar{q}_d)}{\sum_{q\in N_p,\bar{q}_d\in N_{\bar{p}_d}} w(p,q)w(\bar{p}_d,\bar{q}_d)}$$

$$d_p = \arg\min_{d \in S_d} E(p, \bar{p}_d)$$

where $S_d = \{d_{min}, \cdots, d_{max}\}$ is a set of all possible disparity values.

$$e_0(q, \bar{q}_d) = \sum_{c \in \{r,g,b\}} |I_c(q) - I_c(\bar{q}_d)|$$

$$w(p, q) = k \cdot \exp\left(-(\frac{\Delta c_{pq}}{\gamma_c} + \frac{\Delta g_{pq}}{\gamma_p})\right)$$

Δc pq and Δg pq represent the color difference and the spatial distance between pixel p and q.
Δc pq represents the Euclidean distance between two colors, cp = [Lp , ap , bp] and cq = [Lq , aq , bq],in the CIELab color space

$$\Delta c_{pq} = \sqrt{(L_p - L_q)^2 + (a_p - a_q)^2 + (b_p - b_q)^2}$$

γp is determined empirically

γc is typically 7

因为 K 最后会在表达式中约去，所以可以忽略

ii. 代码实现

1. 函数原型

```
Mat ASW(Mat Left, Mat Right, bool LeftOrRight, int size, int rc, int rp);
```

Left：三通道左视图

Right：三通道右视图

LeftOrRight :true ,返回 left disparity map ;false， 返回 right disparity map

Size : size of patch

rc\rp:表达式中的参数

2． 处理

a） 先转成 Lab 彩色模型

```
//change into Lab mode
Mat LeftLab;
cvtColor(Left, LeftLab, CV_BGR2Lab);
Mat RightLab;
cvtColor(Right, RightLab, CV_BGR2Lab);
```

这里需要注意使用的时 CV_BGR2Lab,而不是 CV_RGB2Lab,因为

Mat 默认的彩色图片三通道顺序为 B、G、R

b） 表达式的处理

对于每一隔 patch 内：

```
double Cpq = 0;
double Cp_q_ = 0;
double Gpq = 0;
double Gp_q_ = 0;
double Wpq = 0;
double Wp_q_ = 0;
double e0qq_ = 0;
```

```
//calculate Cpq\Cp_q_
for (int temp = 0; temp < 3; temp ++) {
	double sum1;
	double sum2;
	if (LeftOrRight) {
		sum1 = static_cast<double>(LeftLab.at<Vec3b>(i, j)[temp] - LeftLab.at<Vec3b>(x, y + d)[temp]);
		sum2 = static_cast<double>(RightLab.at<Vec3b>(i, j - d)[temp] - RightLab.at<Vec3b>(x, y)[temp]);
	}
	else {
		sum1 = static_cast<double>(RightLab.at<Vec3b>(i, j)[temp] - RightLab.at<Vec3b>(x, y + d)[temp]);
		sum2 = static_cast<double>(LeftLab.at<Vec3b>(i, j - d)[temp] - LeftLab.at<Vec3b>(x, y)[temp]);
	}
	Cpq += pow(sum1, 2);
	Cp_q_ += pow(sum2, 2);
}
Cpq = sqrt(Cpq);
Cp_q_ = sqrt(Cp_q_);
```

```
//calculate Gpq\Gp_q_
Gpq = sqrt(pow((x - i), 2) + pow((y + d - j), 2));
Gp_q_ = sqrt(pow((x - i), 2) + pow((y - j + d), 2));
```

```
//gain value of Wpq\Wp_q_
Wpq = exp( - (Cpq / rc + Gpq / rp));
Wp_q_ = exp( - (Cp_q_ / rc + Gp_q_ / rp));
```

```
//calculate e0pq\e0p_q_
for (int temp = 0; temp < 3; temp ++) {
	if (LeftOrRight)
		e0qq_ += fabs(static_cast<double>(Left.at<Vec3b>(x, y + d)[temp] - Right.at<Vec3b>(x, y)[temp]));
	else
		e0qq_ += fabs(static_cast<double>(Right.at<Vec3b>(x, y + d)[temp] - Left.at<Vec3b>(x, y)[temp]));
}
```

i. 分子

```
tempZ += Wpq * Wp_q_ * e0qq_;
```

ii. 分母

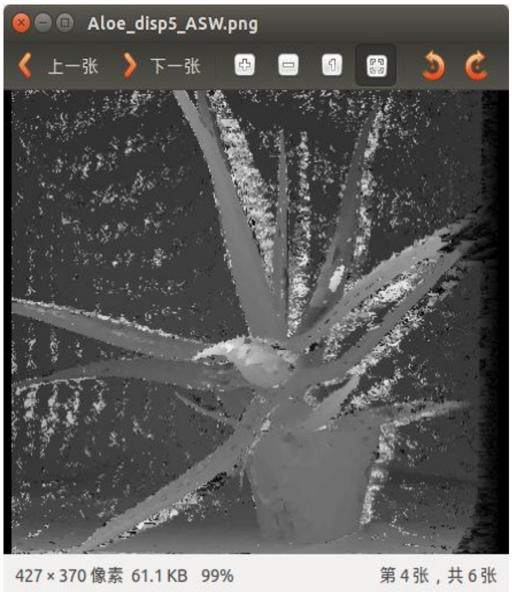
```
tempM += Wpq * Wp_q_;
```

b) 单个测例 Aloe 效果

ods [18], are shown in Figs. 7–8. The proposed algorithm is run with a constant parameter setting across all four images: the size of a support window = (33 × 33), $\gamma_c = 7$, $\gamma_p = 36$. As shown in Figs. 7–8, the proposed method

为了可以比较之前的结果，size=5， rc=7, rp=36

```
duration = 206 seconds
Aloe_Left_ASW: 31.33%
Aloe_Right_ASW: 31.39%
```



这是跑两张图的效果，确实很慢…

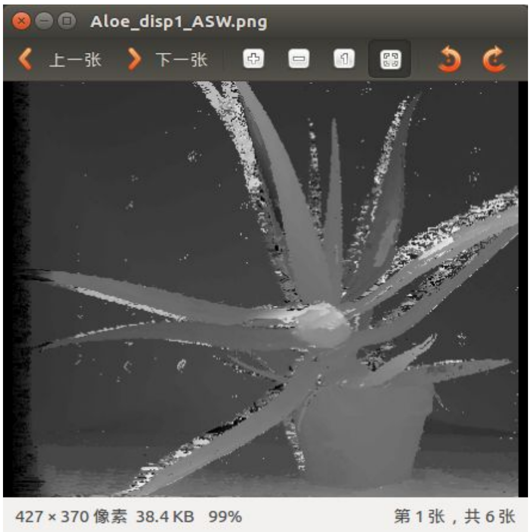
Size=11, rc=7, rp=36:

```
duration = 1050 seconds
Aloe_Left_ASW: 24.06%
Aloe_Right_ASW: 24.72%
```

试跑 size=33， rc=7, rp=36：

跑了 2 个多小时….效果提升得不是很明显…如下图..

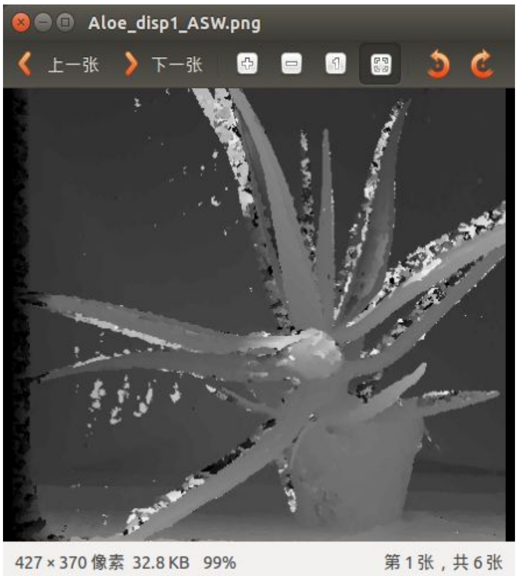
```
duration = 9123 seconds
Aloe_Left_ASW: 22.16%
Aloe_Right_ASW: 23.08%
```



把 rc\rp 的值交换，再取：

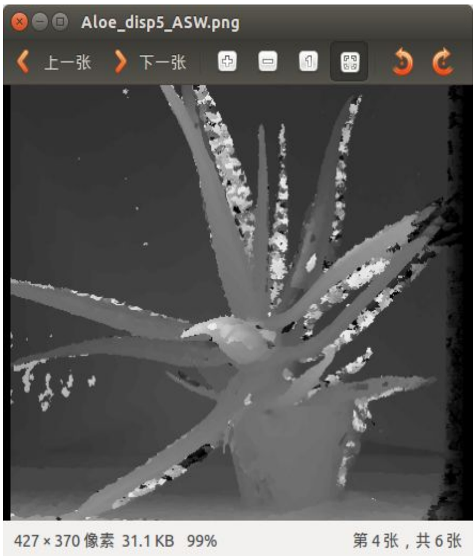
size=5，rc=36, rp=7:(坏点率相对 NCC 有所下降)

```
duration = 225 seconds
Aloe_Left_ASW: 21.95%
Aloe_Right_ASW: 23.13%
```



Size=7,rc=36,rp=7：（坏点率下降不明显）

```
duration = 433 seconds
Aloe_Left_ASW: 21.00%
Aloe_Right_ASW: 22.25%
```

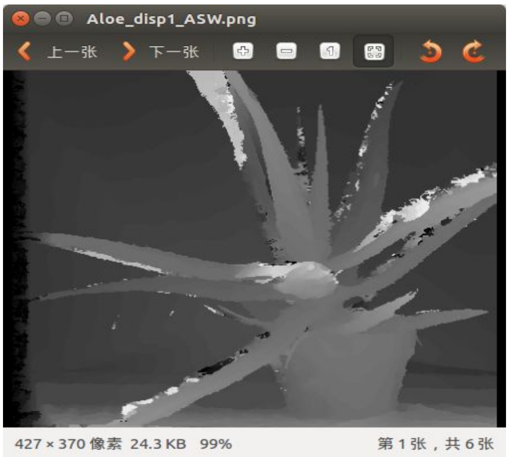


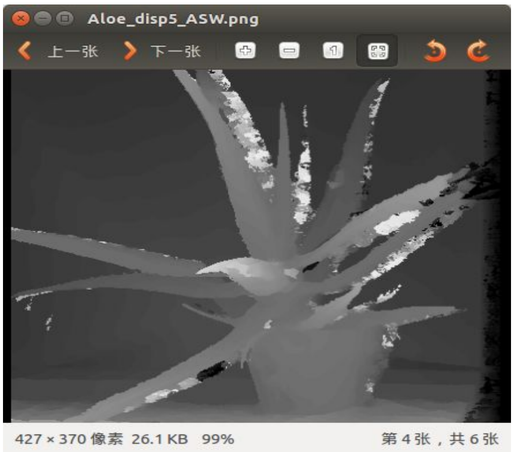
Size=11,rc=36,rp=7：

```
duration = 1045 seconds
Aloe_Left_ASW: 20.19%
Aloe_Right_ASW: 21.55%
```

Size=33*33，rc=36, rp=7

```
duration = 9242 seconds
Aloe_Left_ASW: 19.44%
Aloe_Right_ASW: 20.75%
```





由此可知，ASW 的效果在 11*11 以内的效果都是不明显，当 patch 的大小>=11*11 时，会的到比较好的效果。

c) 全部测例比较

i. Test1

参数设置：

SSD&NCC:size = 5*5
ASW:size = 11*11, rc = 7, rp = 36

Aloe_Left_ASW: 24.06%
Aloe_Left_SSD: 24.72%
Aloe_Left_NCC: 23.64%
Aloe_Right_ASW: 24.72%
Aloe_Right_SSD: 25.78%
Aloe_Right_NCC: 24.20%
Baby1_Left_ASW: 34.32%
Baby1_Left_SSD: 33.65%
Baby1_Left_NCC: 20.70%
Baby1_Right_ASW: 34.82%
Baby1_Right_SSD: 33.31%
Baby1_Right_NCC: 20.78%
Baby2_Left_ASW: 41.70%
Baby2_Left_SSD: 38.97%
Baby2_Left_NCC: 28.33%
Baby2_Right_ASW: 41.85%
Baby2_Right_SSD: 39.25%
Baby2_Right_NCC: 29.19%

Baby3_Left_ASW: 46.33%
Baby3_Left_SSD: 43.15%
Baby3_Left_NCC: 29.24%
Baby3_Right_ASW: 44.66%
Baby3_Right_SSD: 43.08%
Baby3_Right_NCC: 29.34%
Bowling1_Left_ASW: 64.82%
Bowling1_Left_SSD: 69.00%
Bowling1_Left_NCC: 44.21%
Bowling1_Right_ASW: 65.79%
Bowling1_Right_SSD: 69.41%
Bowling1_Right_NCC: 44.25%
Bowling2_Left_ASW: 44.85%
Bowling2_Left_SSD: 50.76%
Bowling2_Left_NCC: 32.08%
Bowling2_Right_ASW: 44.62%
Bowling2_Right_SSD: 50.46%
Bowling2_Right_NCC: 32.29%
Cloth1_Left_ASW: 14.02%
Cloth1_Left_SSD: 10.35%
Cloth1_Left_NCC: 10.37%
Cloth1_Right_ASW: 12.38%
Cloth1_Right_SSD: 8.47%
Cloth1_Right_NCC: 8.50%
Cloth2_Left_ASW: 30.65%
Cloth2_Left_SSD: 31.50%
Cloth2_Left_NCC: 19.39%
Cloth2_Right_ASW: 30.72%
Cloth2_Right_SSD: 31.86%
Cloth2_Right_NCC: 19.87%
Cloth3_Left_ASW: 17.12%
Cloth3_Left_SSD: 15.92%
Cloth3_Left_NCC: 14.80%
Cloth3_Right_ASW: 15.82%
Cloth3_Right_SSD: 14.38%
Cloth3_Right_NCC: 13.04%
Cloth4_Left_ASW: 24.18%
Cloth4_Left_SSD: 22.86%
Cloth4_Left_NCC: 18.96%
Cloth4_Right_ASW: 25.81%
Cloth4_Right_SSD: 24.85%
Cloth4_Right_NCC: 20.97%
Flowerpots_Left_ASW: 57.87%
Flowerpots_Left_SSD: 59.34%

Flowerpots_Left_NCC: 39.52%
Flowerpots_Right_ASW: 57.61%
Flowerpots_Right_SSD: 59.46%
Flowerpots_Right_NCC: 40.28%
Lampshade1_Left_ASW: 57.70%
Lampshade1_Left_SSD: 61.43%
Lampshade1_Left_NCC: 48.94%
Lampshade1_Right_ASW: 56.86%
Lampshade1_Right_SSD: 60.47%
Lampshade1_Right_NCC: 48.93%
Lampshade2_Left_ASW: 69.81%
Lampshade2_Left_SSD: 73.89%
Lampshade2_Left_NCC: 50.19%
Lampshade2_Right_ASW: 70.14%
Lampshade2_Right_SSD: 73.35%
Lampshade2_Right_NCC: 50.67%
Midd1_Left_ASW: 60.68%
Midd1_Left_SSD: 62.81%
Midd1_Left_NCC: 58.65%
Midd1_Right_ASW: 60.63%
Midd1_Right_SSD: 62.50%
Midd1_Right_NCC: 58.60%
Midd2_Left_ASW: 70.45%
Midd2_Left_SSD: 70.74%
Midd2_Left_NCC: 59.35%
Midd2_Right_ASW: 70.89%
Midd2_Right_SSD: 71.26%
Midd2_Right_NCC: 58.87%
Monopoly_Left_ASW: 78.42%
Monopoly_Left_SSD: 67.26%
Monopoly_Left_NCC: 51.81%
Monopoly_Right_ASW: 78.99%
Monopoly_Right_SSD: 65.99%
Monopoly_Right_NCC: 48.26%
Plastic_Left_ASW: 79.93%
Plastic_Left_SSD: 82.16%
Plastic_Left_NCC: 71.18%
Plastic_Right_ASW: 81.46%
Plastic_Right_SSD: 82.99%
Plastic_Right_NCC: 70.66%
Rocks1_Left_ASW: 26.06%
Rocks1_Left_SSD: 24.52%
Rocks1_Left_NCC: 18.64%
Rocks1_Right_ASW: 25.37%

Rocks1_Right_SSD: 23.80%
Rocks1_Right_NCC: 18.06%
Rocks2_Left_ASW: 26.27%
Rocks2_Left_SSD: 23.66%
Rocks2_Left_NCC: 17.95%
Rocks2_Right_ASW: 24.86%
Rocks2_Right_SSD: 21.92%
Rocks2_Right_NCC: 16.43%
Wood1_Left_ASW: 35.25%
Wood1_Left_SSD: 41.51%
Wood1_Left_NCC: 24.38%
Wood1_Right_ASW: 33.72%
Wood1_Right_SSD: 40.64%
Wood1_Right_NCC: 21.78%
Wood2_Left_ASW: 41.13%
Wood2_Left_SSD: 50.02%
Wood2_Left_NCC: 26.82%
Wood2_Right_ASW: 40.52%
Wood2_Right_SSD: 50.35%
Wood2_Right_NCC: 27.21%

比较结果，似乎 ASW 处理过后的视差图质量仅比 SSD 的稍微好一点，

而且也是普遍地比 NCC 差，决定在做一个 Test2

ii. Test2

参数设置：

SSD&NCC:size = 5*5
ASW:size = 11*11, rc = 36, rp = 7

```
duration = 25281 seconds
```

Aloe_Left_ASW: 20.19%
Aloe_Left_SSD: 24.72%
Aloe_Left_NCC: 23.64%
Aloe_Right_ASW: 21.55%
Aloe_Right_SSD: 25.78%
Aloe_Right_NCC: 24.20%
Baby1_Left_ASW: 23.75%
Baby1_Left_SSD: 33.65%
Baby1_Left_NCC: 20.70%
Baby1_Right_ASW: 23.23%

Baby1_Right_SSD: 33.31%
Baby1_Right_NCC: 20.78%
Baby2_Left_ASW: 30.75%
Baby2_Left_SSD: 38.97%
Baby2_Left_NCC: 28.33%
Baby2_Right_ASW: 31.75%
Baby2_Right_SSD: 39.25%
Baby2_Right_NCC: 29.19%
Baby3_Left_ASW: 36.20%
Baby3_Left_SSD: 43.15%
Baby3_Left_NCC: 29.24%
Baby3_Right_ASW: 35.19%
Baby3_Right_SSD: 43.08%
Baby3_Right_NCC: 29.34%
Bowling1_Left_ASW: 61.11%
Bowling1_Left_SSD: 69.00%
Bowling1_Left_NCC: 44.21%
Bowling1_Right_ASW: 61.72%
Bowling1_Right_SSD: 69.41%
Bowling1_Right_NCC: 44.25%
Bowling2_Left_ASW: 40.55%
Bowling2_Left_SSD: 50.76%
Bowling2_Left_NCC: 32.08%
Bowling2_Right_ASW: 40.05%
Bowling2_Right_SSD: 50.46%
Bowling2_Right_NCC: 32.29%
Cloth1_Left_ASW: 9.87%
Cloth1_Left_SSD: 10.35%
Cloth1_Left_NCC: 10.37%
Cloth1_Right_ASW: 8.01%
Cloth1_Right_SSD: 8.47%
Cloth1_Right_NCC: 8.50%
Cloth2_Left_ASW: 23.81%
Cloth2_Left_SSD: 31.50%
Cloth2_Left_NCC: 19.39%
Cloth2_Right_ASW: 23.80%
Cloth2_Right_SSD: 31.86%
Cloth2_Right_NCC: 19.87%
Cloth3_Left_ASW: 12.69%
Cloth3_Left_SSD: 15.92%
Cloth3_Left_NCC: 14.80%
Cloth3_Right_ASW: 10.95%
Cloth3_Right_SSD: 14.38%
Cloth3_Right_NCC: 13.04%

Cloth4_Left_ASW: 19.59%
Cloth4_Left_SSD: 22.86%
Cloth4_Left_NCC: 18.96%
Cloth4_Right_ASW: 21.38%
Cloth4_Right_SSD: 24.85%
Cloth4_Right_NCC: 20.97%
Flowerpots_Left_ASW: 55.40%
Flowerpots_Left_SSD: 59.34%
Flowerpots_Left_NCC: 39.52%
Flowerpots_Right_ASW: 55.14%
Flowerpots_Right_SSD: 59.46%
Flowerpots_Right_NCC: 40.28%
Lampshade1_Left_ASW: 53.89%
Lampshade1_Left_SSD: 61.43%
Lampshade1_Left_NCC: 48.94%
Lampshade1_Right_ASW: 52.80%
Lampshade1_Right_SSD: 60.47%
Lampshade1_Right_NCC: 48.93%
Lampshade2_Left_ASW: 66.81%
Lampshade2_Left_SSD: 73.89%
Lampshade2_Left_NCC: 50.19%
Lampshade2_Right_ASW: 67.17%
Lampshade2_Right_SSD: 73.35%
Lampshade2_Right_NCC: 50.67%
Midd1_Left_ASW: 56.32%
Midd1_Left_SSD: 62.81%
Midd1_Left_NCC: 58.65%
Midd1_Right_ASW: 56.19%
Midd1_Right_SSD: 62.50%
Midd1_Right_NCC: 58.60%
Midd2_Left_ASW: 65.54%
Midd2_Left_SSD: 70.74%
Midd2_Left_NCC: 59.35%
Midd2_Right_ASW: 65.59%
Midd2_Right_SSD: 71.26%
Midd2_Right_NCC: 58.87%
Monopoly_Left_ASW: 62.96%
Monopoly_Left_SSD: 67.26%
Monopoly_Left_NCC: 51.81%
Monopoly_Right_ASW: 62.80%
Monopoly_Right_SSD: 65.99%
Monopoly_Right_NCC: 48.26%
Plastic_Left_ASW: 76.73%
Plastic_Left_SSD: 82.16%

Plastic_Left_NCC: 71.18%
Plastic_Right_ASW: 77.13%
Plastic_Right_SSD: 82.99%
Plastic_Right_NCC: 70.66%
Rocks1_Left_ASW: 21.75%
Rocks1_Left_SSD: 24.52%
Rocks1_Left_NCC: 18.64%
Rocks1_Right_ASW: 21.06%
Rocks1_Right_SSD: 23.80%
Rocks1_Right_NCC: 18.06%
Rocks2_Left_ASW: 20.86%
Rocks2_Left_SSD: 23.66%
Rocks2_Left_NCC: 17.95%
Rocks2_Right_ASW: 18.88%
Rocks2_Right_SSD: 21.92%
Rocks2_Right_NCC: 16.43%
Wood1_Left_ASW: 28.97%
Wood1_Left_SSD: 41.51%
Wood1_Left_NCC: 24.38%
Wood1_Right_ASW: 26.91%
Wood1_Right_SSD: 40.64%
Wood1_Right_NCC: 21.78%
Wood2_Left_ASW: 38.27%
Wood2_Left_SSD: 50.02%
Wood2_Left_NCC: 26.82%
Wood2_Right_ASW: 37.63%
Wood2_Right_SSD: 50.35%
Wood2_Right_NCC: 27.21%

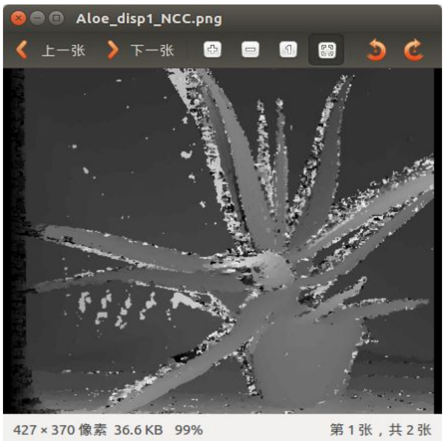
从上面可以看出，11*11 的 ASW 处理效果还不是很理想，虽然总体上处理效果优于 SSD，并且一部分的图像处理效果优于 NCC，但也有很多测例处理效果比 NCC 差的。其实很想跑一下 33*33 的测例，看看整体效果，但是估计了一下时间，大概需要 2-3 days，太慢了，ASW 的算法需要五个循环嵌套计算，暂时还想不出该怎么优化。

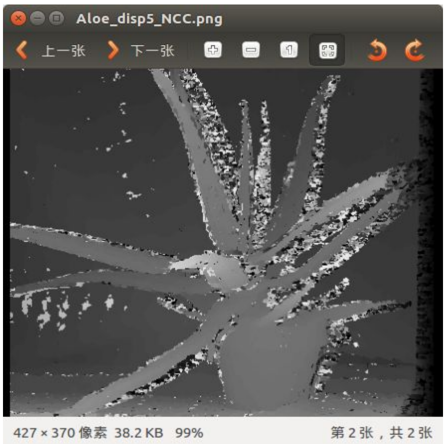
6.与 NCC 之间比较

A）NCC

i. 5*5

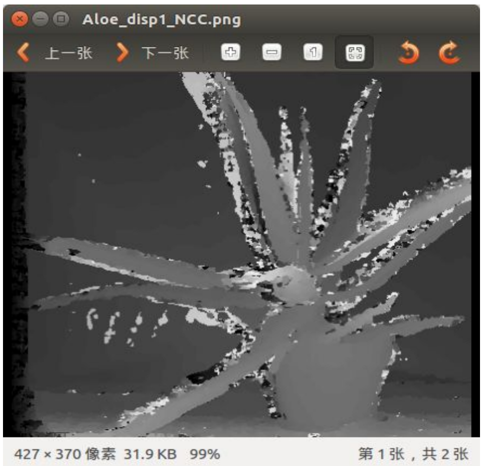
```
duration = 29 seconds
Aloe_Left_NCC: 23.64%
Aloe_Right_NCC: 24.20%
```

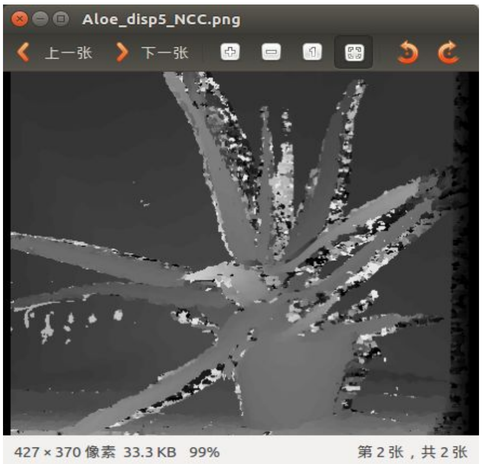




ii. 7*7

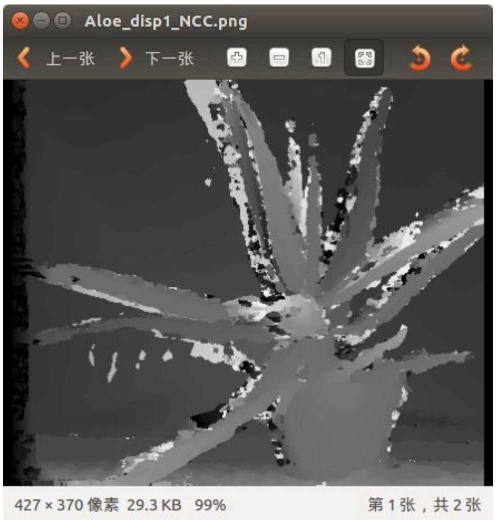
```
duration = 57 seconds
Aloe_Left_NCC: 24.67%
Aloe_Right_NCC: 25.34%
```





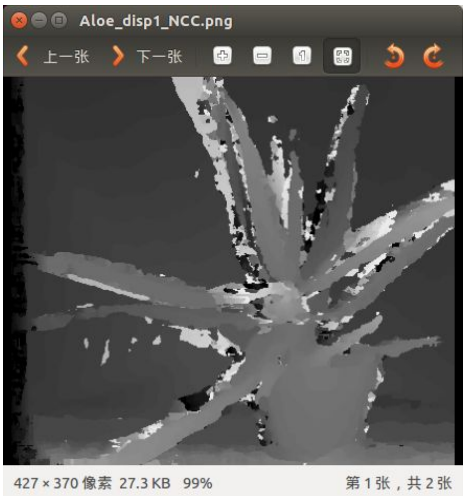
iii. 9*9

```
duration = 89 seconds
Aloe_Left_NCC: 26.08%
Aloe_Right_NCC: 26.79%
```



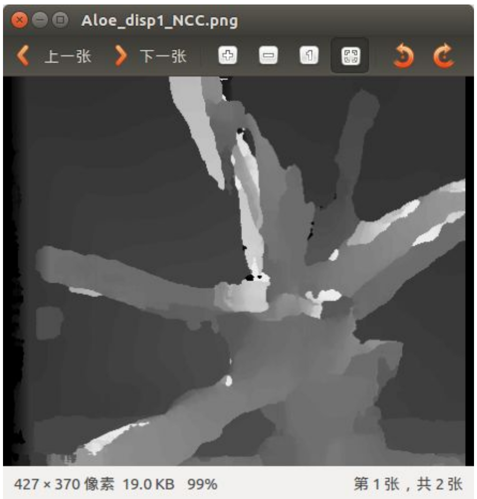
iv. 11*11

```
duration = 133 seconds
Aloe_Left_NCC: 27.35%
Aloe_Right_NCC: 28.08%
```



v. 33*33

```
duration = 1107 seconds
Aloe_Left_NCC: 37.25%
Aloe_Right_NCC: 36.87%
```



B）结论

从以上的比较可以看出，NCC 的主要优点在于，强度增加时并不明显影响视图匹配的效率和质量，但是 patch 窗口增大的时候，视差匹配图的效果就越来越差，坏点率也一直增加。对比 ASW 算法的匹配，小窗口下的 ASW 匹配并不如 NCC、SSD 的效果好，如 3*3、5*5，随着 patch 窗口增大，其处理的视察匹配结果越好，到了 33*33 时，aloe 测例的图形坏点率能达到 20%以下，但是，事实上窗口大小过了 11*11 之后，处理结果的质量上升得也不明显了，对比 11*11、33*33 的结果其实相差不大。

最后的总结：

1．这次 project 因为对性能有要求，需要比较，所以增加了一些 API 来实现这些功能，用 time（）计数，测试各 SSD、NCC、ASW 的性能，同事也有用到 mkdir（dir,mode）函数来创建文件夹，存储测试结果。

2．关于坏点率的得到，因为 evaluate 函数是有直接输出结果的，可以利用重定向直接写入目标文档。