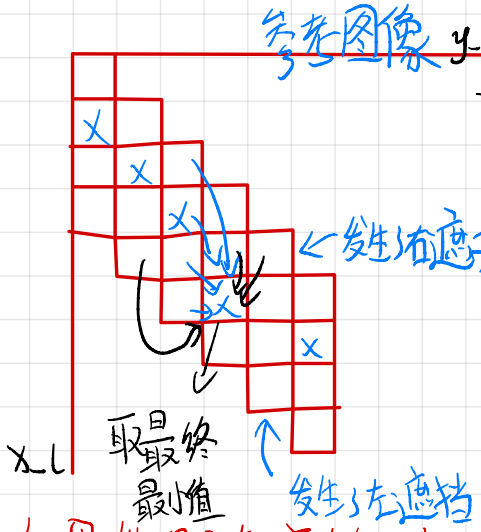
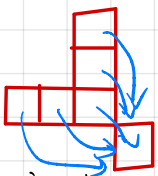


以右图为参考图像

	y=1	2
x=1	d_{1-1}	∞
2	d_{1-2}	$\min(d_{1-1}+d_{2-2}-2\cdot k_r,\ d_{1-2}+d_{2-2}+1\cdot k_{occ},\ d_{1-3}$和$d_{1-4}$同理)
3	d_{1-3}	⋮
4	d_{1-4}	⋮
5	∞	⋮
6	∞	∞

视差变化必然没有遮挡，因此无需惩罚

$-1\cdot k_r$



问题是多大的视差变化才算是遮挡呢？

假设变化1个就罚吧，但是要分左→右，右→左？

如果出现了左遮挡，则

右图 scan_line会有很多连续位置出现视差缺失，

反之同理

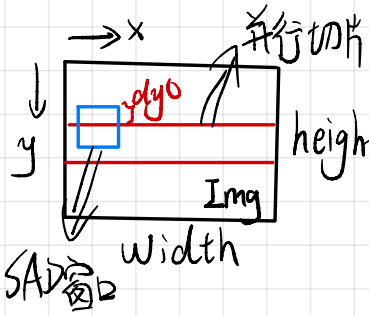
但是好像以右图像为参考无法判断是否有左遮挡，可以先竖直DP再横向DP，即先以R为参考，再以L为参考

	3
x=1	∞
2	∞
3	$\min(C_{2-2}+d_{3-3}+1\cdot k_r,$
4	
5	
6	

C_{2-1}, C_{2-2}, b_{1-2}

如果仍处于同一块遮挡 k_{occ}不需要累加

以上为 standard DP



Wsz2：SAD半窗

dyo：在切片之外的信息（opencv采用并行计算）

dyl：同dyo，只是在结束行

sstep：图像一行指针偏移量

dstep：视差矩阵 列数 × ndisp

cstep：切片中计算SAD高 × ndisp

= 切片高 + dyo + dyl

(int*) sad → sad[-1]

看是否标志位

内存长度 ndisp + dyo*ndisp ⇒ 仅为设定视差搜索范围的长度

(int*) hasdo

(height + dyl)* ndisp + Wsz2 + 1 ⇒ 列乘以视差的长度

= ndisp + dyo*ndisp + 1 + sad

(int*) htext

height + Wsz2 + 1 + dyo*ndisp ⇒ 一列的长度

= hsado + (height + dyl)*ndisp + Ws2 + 2

(uchar*) cbufo

(Wsz+1)* cstep（暂存上次求取的AD，即右面一列要减去的值）

= htext + height + Wsz2 + 2 + dyo*ndisp

原则上只需要 2*cstep，但为便编程

初始化内存

```
memset( hsad0 - dy0*ndisp, 0, (height + dy0 + dy1)*ndisp*sizeof(hsad0[0]) );
memset( htext - wsz2 - 1, 0, (height + wsz + 1)*sizeof(htext[0]) );
```

初始化[hsad] ⇒ 表数组
初始化[htext]
} 因为这两个后面都是累加的

```
for( x = -wsz2-1; x < wsz2; x++ )
{
    hsad = hsad0 - dy0*ndisp; cbuf = cbuf0 + (x + wsz2 + 1)*cstep - dy0*ndisp;  // 因为视差范围已经规定了必须要是
16 的整数倍，所以减去是不会影响指针对齐的
    lptr = lptr0 + std::min(std::max(x, -lofs), width-lofs-1) - dy0*sstep;
    rptr = rptr0 + std::min(std::max(x, -rofs), width-rofs-ndisp) - dy0*sstep;  // 中间部分是为了防止越界，正常的都为
x，因为一般不会越界，，
    for( y = -dy0; y < height + dy1; y++, hsad += ndisp, cbuf += ndisp, lptr += sstep, rptr += sstep )  // 这里后面的为地
    {
        int lval = lptr[0];
        d = 0;
        for( ; d < ndisp; d++ )
        {
            int diff = std::abs(lval - rptr[d]);
            cbuf[d] = (uchar)diff;
            hsad[d] = (int)(hsad[d] + diff);  //
        }
        htext[y] += tab[lval];  // 纹理通过查表
而且仅仅是左图的纹理
    }
}
```

先计算一个窗口列数的SAD，然后自构机式前进，边吃边拉

hsad实际长度为(dy0+height+dy1)*ndisp

越界保护

索取并行切片外，但SAD计算需要的

红=黑+绿-蓝

没有重叠的(只吃不拉) ⇒ 存储要拉多少，可不能随便乱拉啊

SAD计算(第一次只吃不拉)

纹理先映射再累加(也是边吃边拉，映射表相当于卷积)

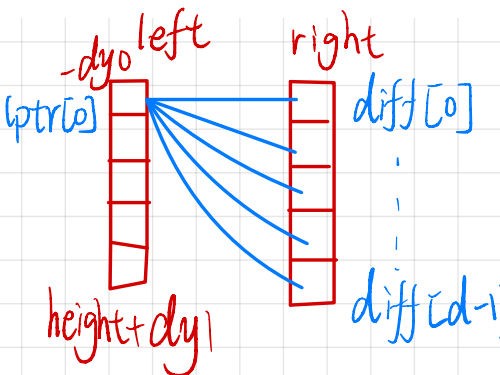
```
for( x = 0; x < width1; x++, dptr++ )
{
    int* costptr = cost.data ? cost.ptr<int>() + lofs + x : &costbuf;
    int x0 = x - wsz2 - 1, x1 = x + wsz2; // x0 表示窗口的左边缘，x1 表示窗口的右边缘
    const uchar* cbuf_sub = cbuf0 + ((x0 + wsz2 + 1) % (wsz + 1))*cstep - dy0*ndisp; // 中间
吗？而且为什么不直接用 x 表示，因为 x0+wsz2+1 不就等于 x 吗，，
    cbuf = cbuf0 + ((x1 + wsz2 + 1) % (wsz + 1))*cstep - dy0*ndisp; // cbuf0 已经初始化过
    hsad = hsad0 - dy0*ndisp;
    lptr_sub = lptr0 + MIN(MAX(x0, -lofs), width-1-lofs) - dy0*sstep;
    lptr = lptr0 + MIN(MAX(x1, -lofs), width-1-lofs) - dy0*sstep;
    rptr = rptr0 + MIN(MAX(x1, -rofs), width-ndisp-rofs) - dy0*sstep;

    for( y = -dy0; y < height + dy1; y++, cbuf += ndisp, cbuf_sub += ndisp,
        hsad += ndisp, lptr += sstep, lptr_sub += sstep, rptr += sstep )
    {
        int lval = lptr[0];
        d = 0;
        for( ; d < ndisp; d++ )
```

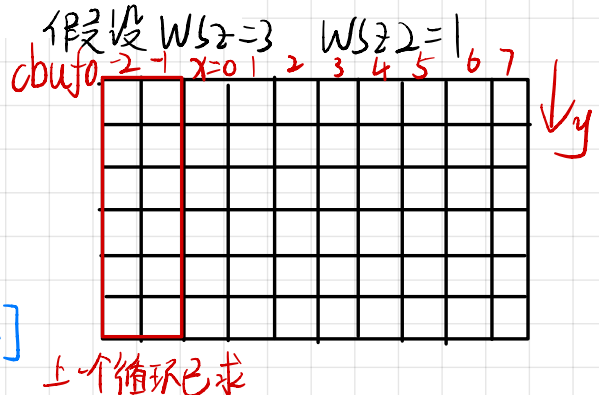
只计算信息重叠部分

(左右-致性检查的)

SAD窗口上边缘

下边缘





```
    {
        int diff = std::abs(lval - rptr[d]);
        cbuf[d] = (uchar)diff;
        hsad[d] = hsad[d] + diff - cbuf_sub[d];
    }
    htext[y] += tab[lval] - tab[lptr_sub[0]];
}

// fill borders
for( y = dy1; y <= wsz2; y++ )
    htext[height+y] = htext[height+dy1-1];
for( y = -wsz2-1; y < -dy0; y++ )
    htext[y] = htext[-dy0];

// initialize sums
for( d = 0; d < ndisp; d++ )
    sad[d] = (int)(hsad0[d-ndisp*dy0]*(wsz2 + 2 - dy0));
hsad = hsad0 + (1 - dy0)*ndisp;
for( y = 1 - dy0; y < wsz2; y++, hsad += ndisp )
{
    d = 0;
    for( ; d < ndisp; d++ )
        sad[d] = (int)(sad[d] + hsad[d]);
}
int tsum = 0;
for( y = -wsz2-1; y < wsz2; y++ )
    tsum += htext[y];

// finally, start the real processing
for( y = 0; y < height; y++ )
{
    int minsad = INT_MAX, mind = -1;
    hsad = hsad0 + MIN(y + wsz2, height+dy1-1)*ndisp;
    hsad_sub = hsad0 + MAX(y - wsz2 - 1, -dy0)*ndisp;
    d = 0;
    for( ; d < ndisp; d++ )
    {
        int currsad = sad[d] + hsad[d] - hsad_sub[d];
        sad[d] = currsad;
        if( currsad < minsad )
        {
            minsad = currsad;
            mind = d;
        }
    }
    tsum += htext[y + wsz2] - htext[y - wsz2 - 1];
    if( tsum < textureThreshold )
    {
        dptr[y*dstep] = FILTERED;
        continue;
    }

    if( uniquenessRatio > 0 )
    {
        int thresh = minsad + (minsad * uniquenessRatio/100);
        for( d = 0; d < ndisp; d++ )
        {
            if( (d < mind-1 || d > mind+1) && sad[d] <= thresh)
                break;
        }
        if( d < ndisp )
        {
            dptr[y*dstep] = FILTERED;
            continue;
        }
    }

    sad[-1] = sad[1];
    sad[ndisp] = sad[ndisp-2];
    int p = sad[mind+1], n = sad[mind-1];
    d = p + n - 2*sad[mind] + std::abs(p - n);
    dptr[y*dstep] = dispDescale<mType>(ndisp - mind - 1 + mindisp, p-n, d);

    costptr[y*coststep] = sad[mind];
}
```