luofuchong

IT博客 首页 新随笔 联系 聚合 管理

随笔-59 评论-129 文章-4 trackbacks-0

< 2010年5月 >

日	一	二	三	四	五	六
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

常用链接

我的随笔
我的评论
我参与的随笔

留言簿(53)

给我留言
查看公开留言
查看私人留言

随笔档案

2012年1月 (1)
2011年11月 (1)
2011年10月 (3)
2011年8月 (1)
2011年5月 (1)
2010年8月 (2)
2010年5月 (2)
2010年3月 (1)
2010年2月 (1)
2009年10月 (2)
2009年9月 (1)
2009年7月 (2)
2009年4月 (2)
2009年3月 (2)
2008年10月 (3)
2008年9月 (1)
2008年8月 (2)
2008年5月 (1)
2008年4月 (4)
2008年3月 (1)
2008年1月 (2)
2007年12月 (1)
2007年11月 (1)
2007年10月 (1)
2007年9月 (5)
2007年8月 (8)
2007年7月 (2)
2007年1月 (5)

搜索

搜索

【转】Tslib主要滤波算法分析

Tslib解析

Author:Jiujin.hong
转载请说明出处：http://blog.csdn.net/hongjiujing or www.linuxforum.net嵌入式linux版块
tslib背景：

在采用触摸屏的移动终端中，触摸屏性能的调试是个重要问题之一，因为电磁噪声的缘故，触摸屏容易存在点击不准确、有抖动等问题。
Tslib是一个开源的程序，能够为触摸屏驱动获得的采样提供诸如滤波、去抖、校准等功能，通常作为触摸屏驱动的适配层，为上层的应用提供了一个统一的接口。

tslib插件：

pthres 为Tslib 提供的触摸屏灵敏度门槛插件；variance 为Tslib 提供的触摸屏滤波算法插件；dejitter 为Tslib 提供的触摸屏去噪算法插件；linear 为Tslib 提供的触摸屏坐标变换插件。
触摸屏驱动为与硬件直接接触部分，为上层的Tslib 提供最原始的设备坐标数据，并可以配置采样间隔、屏幕灵敏度等。采样间隔决定了单位时间内的采样数量，在其他参数不变的情况下，采样间隔越小意味着单位时间内的采样数量越多，也就意味着采样越逼真、越不容易出现采样信息丢失如输入法书写时丢笔划的情况，但因为噪声的影响，采样间隔越小同时也意味着显示出来的图形的效果越差。
Tslib 为触摸屏驱动和应用层之间的适配层，其从驱动处获得原始的设备坐标数据，通过一系列的去噪、去抖、坐标变换等操作，来去除噪声并将原始的设备坐标转换为相应的屏幕坐标。

tslib接口:
在tslib 中为应用层提供了2 个主要的接口ts_read()和ts_read_raw()，其中ts_read()为正常情况下的借口，ts_read_raw()为校准情况下的接口。

正常情况下，tslib 对驱动采样到的设备坐标进行处理的一般过程如下：
raw device --> variance --> dejitter --> linear --> application
module module module
校准情况下，tslib 对驱动采样到的数据进行处理的一般过程如下：
raw device--> Calibrate
由于各种相关期间的影响，在不同的硬件平台上，相关参数可能需要调整。以上参数的相互关系为：采样间隔越大，采样点越少，采样越失真，但因为信息量少，容易出现丢笔划等丢失信息情况，但表现出来的图形效果将会越好；去噪算法跟采样间隔应密切互动，采样间隔越大，去噪约束应越小，反之采样间隔越小，去噪约束应越大。去抖算法为相对独立的部分，去抖算法越复杂，带来的计算量将会变大，系统负载将会变重，但良好的去抖算法可以更好的去除抖动，在进行图形绘制时将会得到更好的效果；灵敏度和ts 门槛值为触摸屏的灵敏指标，一般不需要进行变动，参考参考值即可。

过滤插件分析：

Variance:触摸屏滤波算法

问题：一些触摸屏取样非常粗略，因此，即使你持着笔不放，样本可能不同，有时会大幅增加。最坏的情况是由于采样的时候电噪声的干扰，可大大脱离现实笔的位置不同，这会导致鼠标光标移动“跳”起来，然后返回回来。
解决方法：延迟一个时隙采样数据。如果我们看到最后采样读出来的数据太多的不同，我们将其标示为“可疑”。如果下一个采样读取的数据接近“可疑” 情况出现之前的数据，“可疑”数据将被丢弃。否则我们认为笔正在进行一个快速的笔移动动作，“可疑”数据的采样和出现”可疑”数据之后的采样都将通过。

重要算法分析：

```
static int variance_read(struct tslib_module_info *info, struct ts_sample *samp, int nr)
{
struct tslib_variance *var = (struct tslib_variance *)info;
struct ts_sample cur;
int count = 0, dist;

while (count < nr) {
```

如果采样数据被标记为“提交噪音”状态，将当前采样数据相关结构体赋予噪音状态，将清除标志位。

```
if (var->flags & VAR_SUBMITNOISE) {
cur = var->noise;
var->flags &= ~VAR_SUBMITNOISE;
} else {
```

如果如果采样数据没有被标记为“提交噪音”，继续采样数据。

```
if (info->next->ops->read(info->next, &cur, 1) < 1)
return count;
}
```

如果当前没有压力值，处于没有笔触摸或者笔释放状态，但是却收到笔按下消息，表明为收到噪音干扰，
所有当笔一释放就立即清除队列，否则之前的层将捕抓到笔起来的消息，但是已经太晚，如果info->next->ops->read()出现堵塞，将出现这种情况。

```
if (cur.pressure == 0) {
/* Flush the queue immediately when the pen is just
* released, otherwise the previous layer will
* get the pen up notification too late. This
* will happen if info->next->ops->read() blocks.
*/
if (var->flags & VAR_PENDOWN) {
var->flags |= VAR_SUBMITNOISE;
var->noise = cur;
}
/* Reset the state machine on pen up events. */
```

复位笔起来事件状态标记位

```
var->flags &= ~(VAR_PENDOWN | VAR_NOISEVALID | VAR_LASTVALID);
goto acceptsample;
```

通知接受采样数据

```
} else
var->flags |= VAR_PENDOWN;
```

通知笔按下
如果标记位与“VAR_LASTVALID"状态不同，进行下一个采样。

```
if (!(var->flags & VAR_LASTVALID)) {
var->last = cur;
var->flags |= VAR_LASTVALID;
continue;
}
```

如果为笔按下事件

```
if (var->flags & VAR_PEN DOWN) {
/* Compute the distance between last sample and current */
```

计算上一次的采样数据与当前采样数据的距离

```
dist = sqr (cur.x - var->last.x) +
sqr (cur.y - var->last.y);

if (dist > var->delta) {
```

如果误差大于默认值，比如30。

视之前的采样为噪音？可疑？

```
/* Do we suspect the previous sample was a noise? */
if (var->flags & VAR_NOISEVALID) {
```

但是如果之前的采样已经是可疑状态，视为快速的笔移动触发动作。

```
/* Two "noises": it's just a quick pen movement */
samp [count++] = var->last = var->noise;
var->flags = (var->flags & ~VAR_NOISEVALID) |
VAR_SUBMITNOISE;
} else
```

如果之前的采样并不是可疑状态，视为可疑状态.

```
var->flags |= VAR_NOISEVALID;
/* The pen jumped too far, maybe it's a noise ... */
var->noise = cur;
```

```
continue;
} else
var->flags &= ~VAR_NOISEVALID;采样的数据属于正常数据.
}

acceptsample:
#ifdef DEBUG
fprintf(stderr,"VARIANCE----------> %d %d %d\n",
var->last.x, var->last.y, var->last.pressure);
#endif
samp [count++] = var->last;
var->last = cur;
}

return count;
}
```

dejitter 去噪插件分析：

问题：一些触摸屏从ADC获取X/Y坐标采样值，他们的最低位带有很大的噪音干扰，这就导致了触摸屏输出值的抖动。

比如我们保持着按某一点，我们会得到许多的X/Y坐标采样，他们相近但是不相等。同时如果我们试图在一个画图程序里面去画一个直线，

我们将得到一个充满“毛刺“的直线。

解决：我们对最后几个值应用一个重量平滑滤波，从而去除输出“毛刺”。我们发现坐标发生重大变化，我们会重新设置笔位置的积压，从而

避免平滑不应该要平滑的坐标。当然，这些都是假设所有噪音都已经由底端过滤器滤波过了，例如variance模块。

工作原理：

该过滤器的工作原理如下：我们掌握最新的N样本轨道，我们不断跟踪最新的N个采样，根据一定的重量求平均。最旧的数据有最少的重量，最近的数据

有最大的重量。这有助于消除抖动，同时不影响响应时间,因为我们为每一个输入采样输出一个输出样本，笔移动会变得更加顺畅。

重要算法分析：

为了让事情简单（避免误差),我们确保SUM(重量）＝2次方。同时当我们有不到默认采样数量的时候,我们必须知道怎么去近似测试。

```
static const unsigned char weight [NR_SAMPHISTLEN - 1][NR_SAMPHISTLEN + 1] =
{
/* The last element is pow2(SUM(0..3)) */
{ 5, 3, 0, 0, 3 }, /* When we have 2 samples ... */
{ 8, 5, 3, 0, 4 }, /* When we have 3 samples ... */
{ 6, 4, 3, 3, 4 }, /* When we have 4 samples ... */
};

static void average (struct tslib_dejitter *djt, struct ts_sample *samp)
{
const unsigned char *w;
int sn = djt->head;
int i, x = 0, y = 0;
unsigned int p = 0;

w = weight [djt->nr - 2];找出与重量数组相对应的数据，例如如果是第一次采样就没有，如果是第二次采样，就对应{ 5, 3, 0, 0, 3 },依此类推。

for (i = 0; i < djt->nr; i++) {
x += djt->hist [sn].x * w [i];
y += djt->hist [sn].y * w [i];
p += djt->hist [sn].p * w [i];
sn = (sn - 1) & (NR_SAMPHISTLEN - 1);记录每一次采样的序号
}

samp->x = x >> w [NR_SAMPHISTLEN];求出平均值
samp->y = y >> w [NR_SAMPHISTLEN];
samp->pressure = p >> w [NR_SAMPHISTLEN];
#ifdef DEBUG
fprintf(stderr,"DEJITTER----------> %d %d %d\n",
samp->x, samp->y, samp->pressure);
```

```
#endif
}

static int dejitter_read(struct tslib_module_info *info, struct ts_sample *samp, int nr)
{
struct tslib_dejitter *djt = (struct tslib_dejitter *)info;
struct ts_sample *s;
int count = 0, ret;

ret = info->next->ops->read(info->next, samp, nr);
for (s = samp; ret > 0; s++, ret--) {
if (s->pressure == 0) {
/*
* Pen was released. Reset the state and 如果笔释放，复位状态标准，同时丢弃所有历史事
件。
* forget all history events.
*/
djt->nr = 0;
samp [count++] = *s;
continue;
}

/* If the pen moves too fast, reset the backlog. */ 如果笔移动太快，复位积压
if (djt->nr) {
int prev = (djt->head - 1) & (NR_SAMPHISTLEN - 1);
if (sqr (s->x - djt->hist [prev].x) +
sqr (s->y - djt->hist [prev].y) > djt->delta) { 如果之前的x的平方距离值与之前的y的平方距离
值加入门槛值，提示超过门槛值，丢弃，复位。
#ifdef DEBUG
fprintf (stderr, "DEJITTER: pen movement exceeds threshold\n");
#endif
djt->nr = 0;
}
}

djt->hist[djt->head].x = s->x;
djt->hist[djt->head].y = s->y;
djt->hist[djt->head].p = s->pressure;
if (djt->nr < NR_SAMPHISTLEN) 如果采样数小于默认采样数，继续执行
djt->nr++;

/* We'll pass through the very first sample since
* we can't average it (no history yet).
*/
if (djt->nr == 1) 如果这是第一次采样，没有历史或者旧采样数据，直接赋值。
samp [count] = *s;
else { 如果不是第一次采样，就执行平均函数，求得经过平均后的采样值。
average (djt, samp + count);
samp [count].tv = s->tv;
}
count++;

djt->head = (djt->head + 1) & (NR_SAMPHISTLEN - 1);记录采样的序号
}
return count;
}
```

总结：经过分析varience滤波模块插件和dejitter去抖模块插件，我们知道如下：
1：varience是最低层滤波插件，方差滤波器，试图做得最好，过滤掉由ADC采样过来的随机噪音，
通过限制某些采样的运动速度，例如：
笔不应该比一些门槛值快一些。
主要参数：门槛值delta
求出之前的采样点和当前的采样点的平方距离（X2-X1)^2 + (Y2-Y1)^2)，用来确定两个样本是
“近”还是“远”。如果以前和目前的样本之间的距离是'远'，
样品被标记为'潜在噪音'或者“可疑“,但这并不意味着它将被丢弃。如果下次的采样接近于它，我们将
视是一次普通的快速移动动作。同时如果“潜在噪音"之后的采样比之前讨论的
采样都“远”，也将认为出现了一次普通的快速移动动作。如果出现“潜在噪音”之后的采样和出现

“潜在噪音“之前的采样相近，我们将丢弃”潜在噪音 “这次数据，认为它为要过滤的噪音。
2:dejitter去抖模块插件
去除X/Y坐标的抖动,这是通过使用一个加权平滑滤波器实现的。最近的采样有最重的重量，早期的采样有重量轻的重量，这使得实现1：1的输入－输出速率。
主要参数：门槛值delta
两个采样之间的平方距离，（X2-X1)^2 + (Y2-Y1)^2),即定义了'快速运动'的门槛。如果笔移动快，平滑笔的动作是不合适的,另外，快速运动任何时候都不是准确的。所以如果检测到了快速运动，该过滤模块只是简单地丢弃积压和复制输入到输出。

另外有兄弟比较懂dejitter这个插件的，可以详细讲一下,先谢谢了！

Changelog:

1:Post initial version

posted on 2010-05-06 11:28 lfc 阅读(2099) 评论(0) 编辑 收藏 引用

标题 re: 【转】Tslib主要滤波算法分析

姓名

主页

验证码 * 6601

内容(提交失败后,可以通过“恢复上次提交”恢复刚刚提交的内容)

Remember Me?

提交 登录 使用高级评论 新用户注册 返回页首 恢复上次提交

[使用Ctrl+Enter键可以直接提交]

博客园首页随笔：

· 阿里云的背后故事（希望别被关了）
· 设计模式学习总结-外观模式（Facade Pattern)
· 内存的一些magic number和debug crt
· IHttpModule.Init方法被执行多次的原因
· WebCore渲染之一：基础
博客园 IT新闻 BlogJava 博客生活 C++博客 PHP博客