ytp 弹幕算法说明

0 前言

本文只讨论整体的思想，不会涉及详细的代码实现，毕竟写代码是最基础的东西，重要的还是设计一个好的模型，以后扩展起来也方便。

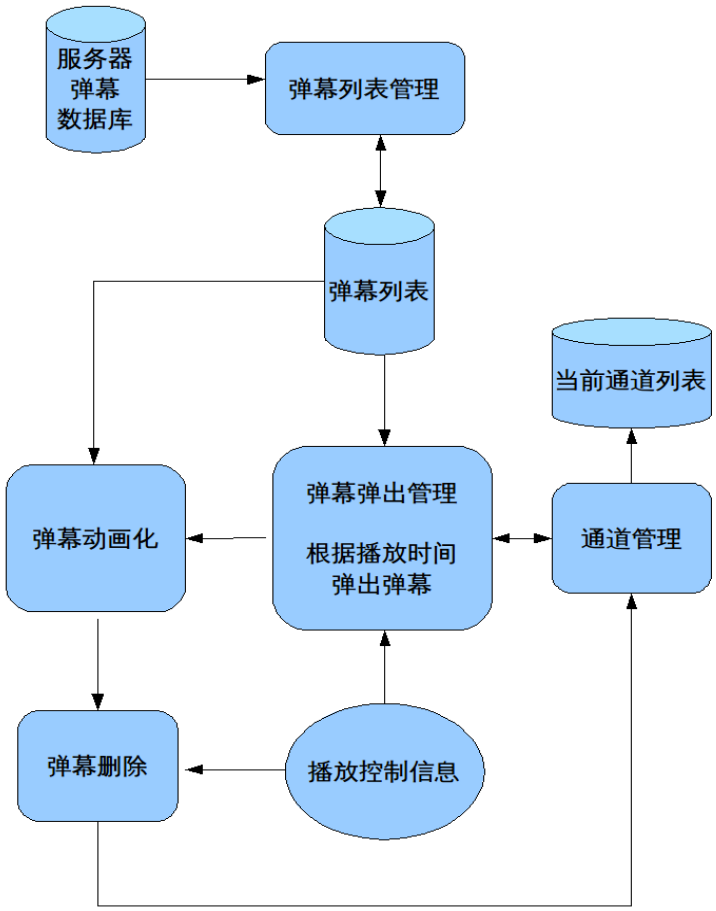
全文没有检查错别字，而且我自己写完以后也没有复看一遍，如果有错误或错别字的话还请谅解。

这里讨论的只是最基本的，也是最核心的弹幕效果。至于扩展之类的，如支持脚本，甚至支持 srt 字幕格式之类的，在这里不会讨论。其实要实现较好的扩展性，一个好的数据结构和框架是必不可少的。这篇文章所说明的算法在我看来应该还是比较容易扩展的，至少我现在想到的扩展都能比较容易地实现。

与本文对应的 ytp 的项目地址是 http://ytp.bbxy.net/，源代码也在这上面。在写这篇文的时候 SVN 源码的版本是 73，但这个版本里面的算法和数据结构以及程序框架都是我一点一点慢慢实现起来的，组织结构很乱，我也懒得去重写一遍了。这篇文章可以算是把程序框架重写了一遍吧，毕竟看这篇文章要比去看那些代码更容易整个算法。

1 算法整体过程

首先将屏幕划分为不同的通道，然后按照时间顺序为每一个弹幕分配通道，在分配通道的时候需要计算弹幕所使用的通道会不会和其他弹幕使用的通道相冲突。弹幕分配到通道以后，就将通道信息保存起来，然后将弹幕交由弹幕显示函数处理。弹幕显示函数根据弹幕类型和弹幕时间对弹幕应用动画效果。过程可以参考右图。



2 通道分配

2.1 通道划分

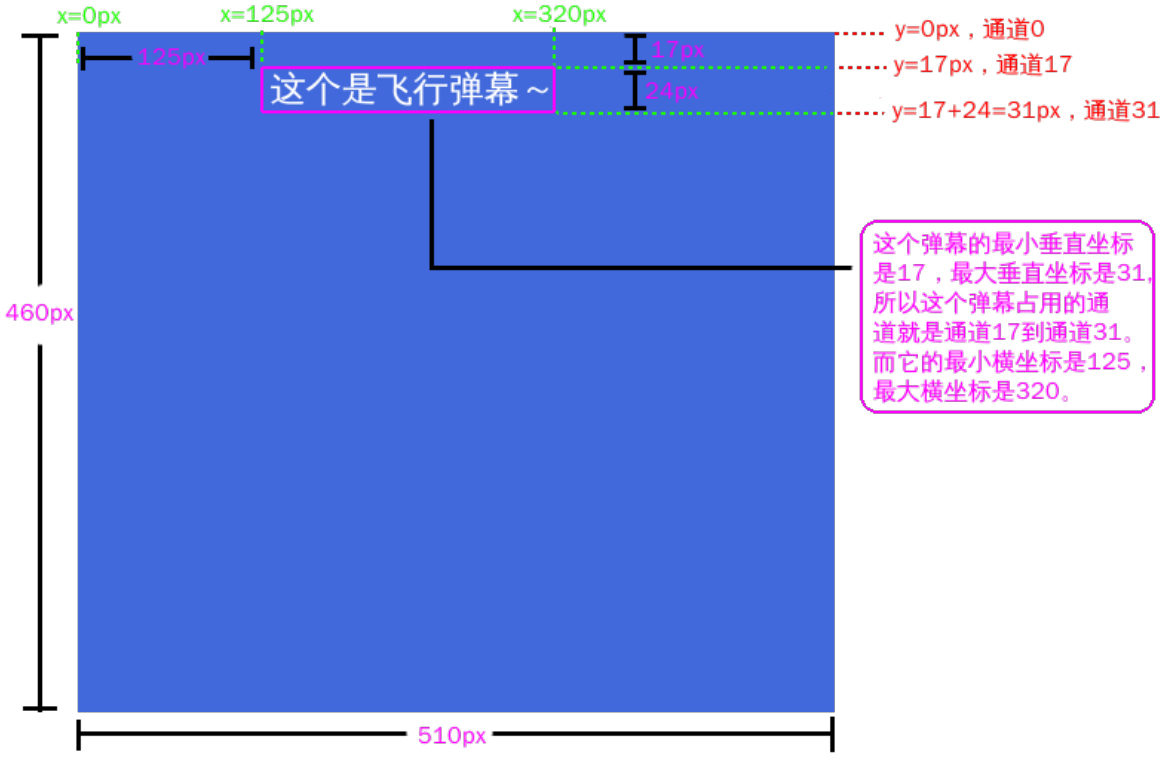
为了方便后期计算通道冲突，我们就将每一行像素作为一个通道，于是整个播放屏幕的通道数就是视频的高度。如果视频的高度是 600，那么我们就有 600 个通道。为了便于说明，我们还要给通道进行编号，编号从上到下，第一行是 1 号，第二行是 2 号，以此类推。

因为一个弹幕占据一行的高度不可能只有一个像素，所以一个弹幕将会占用多个连续的通道。具体占用

多少个像素，由弹幕的字体大小决定。如果一个弹幕的字体大小是 16px，那么这个弹幕就要占用 16 个通道。

2.2 记录弹幕占用的通道范围

我们要计算弹幕占用的通道是否互相冲突，于是就要保存每个弹幕占用的通道。因为弹幕的字体大小是固定的，所以我们只要记录弹幕占用的最小通道编号就可以了。假设一个弹幕占用的最小通道数是 n，弹幕字体大小为 s，那么这个弹幕占用的通道范围就是：[n, n+s-1]，也就是 n 号通道到 n+s-1 号通道。
为了方便计算通道冲突，我们需要保存每个通道被占用的情况，这可以利用一个二维数组来实现。定义一个数组 channel_used[channel_id, poptext_list]，第一个数据项是通道编号，第二个数据项保存占用了这个通道的弹幕编号列表。图示如下。



3 弹幕管理

一般在播放器加载影片的时候，需要同时从服务器中获取对应影片的弹幕列表，并将这些弹幕管理起来，方便调用。显然，我们应该用一个数组来保存弹幕数据。为了方便查找弹幕，我们把弹幕按照弹出时间（弹幕出现在屏幕上时影片播放的时间）升序排列。
为了方便管理，我们还应该给每个弹幕指定一个唯一的编号，我们可以直接使用弹幕在数据库中的编号作为弹幕的编号，这样就不必为每一个弹幕生成一个编号了。
设置一个名为 poptext_list 的一维数组，用来保存弹幕列表。

3.1 插入弹幕

用户在发表一个新的弹幕时，这个弹幕必须立即显示到影片上，这就要求我们立刻把这个弹幕插入到弹幕列表中。因为弹幕列表已经是顺序排列的了，我们可以使用二分法在 $\log_2(n)$ 的时间复杂度内找到弹幕应该插入的位置，然后就可以直接插入弹幕了。
应用弹幕动画
设置一个变量 next_poptext_index，指示下一个应该被显示的弹幕在数组中的下标。在播放器刚刚载入

的时候，该变量应该为 0。另外设置一个定时器，定时器定时执行函数 check_pop_poptext()函数，该函数主要过程如下：

1. 检查第 next_poptext_index 个弹幕的弹出时间是否大于当前影片播放的时间，如果是，则结束执行，否则执行下一步；
2. 对第 next_poptext_index 个弹幕应用动画效果，也就是调用 show_poptext()函数；
3. next_poptext_index++，然后跳到第 1 步继续执行。

在 ActionScript 中，定时器可以用 setTimeout()函数实现。

3.2 设置动画效果（show_poptext()函数）

这个函数是比较核心的函数，它要完成的工作有三个，分别是：

1. 使用 request_channel()函数为弹幕申请一个通道；
2. 申请一个 TextField，并将其放到刚刚申请的通道上，然后设置 TextFiled 的内容和样式（也就是弹幕内容和样式）；
3. 然后根据弹幕的类型（顶端、底端或飞行）调用具体的实现动画效果的函数（poptext_fly()、poptext_top()和 poptext_bottom()函数）。

TextField 的垂直坐标是已经确定了的，就是通道编号，我们要利用水平坐标来实现弹幕飞行的效果。

首先实现 poptext_top()和 poptext_bottom()函数，这两个函数只需要把 TextField 放到水平居中的位置，然后显示 TextField，最后再启动一个一次性的定时器，定时器时间长度和弹幕显示时间相同，定时器时间到后就删除这个 TextField。

为了实现水平居中，TextField 的水平坐标可以通过如下公式计算出来：

TextField._x = (VideoWidth – TextField.width) / 2

接下来要实现 poptext_fly()函数。在这个函数中需要设置一个定时器，定时器每次执行都会计算弹幕这个时候应该飞行到的位置的水平坐标，然后把这个坐标值赋给 TextField。定时器的时间可以设置为 50 毫秒，这样定时器在 1 秒内就会被执行 20 次，TextField 的水平位置也被刷新了 20 次，于是看起来的效果就是弹幕在飞行了，实际上也就是动画的原理。

已经飞出屏幕范围的弹幕是应该删除掉的，所以定时器在计算弹幕的水平坐标之后，应该首先判断弹幕是否已经飞出了播放器范围，如果是，则删除 TextField，并停止定时器。可以通过下面的逻辑表达式来判断弹幕是否已经飞出了屏幕范围（只考虑弹幕从右往左飞行的情况）：

TextField._x + TextField.width <= 0

无论弹幕是飞行的还是顶部的或是底部的弹幕，在删除 TextField 之后都应该立刻调用 release_channel()函数释放它所占用的通道。

4 通道分配和冲突检测

4.1 通道分配过程

按照目前常见的情况，飞行弹幕和顶部弹幕是从上到下排列的，而且在没有充满屏幕的情况下是不能重叠的，而底部弹幕是从下到上排列的，且在没有充满屏幕的情况下也是不能重叠的。鉴于两种不同的弹幕排列方式，我们的 request_channel()函数需要按照不同的弹幕类型使用不同的算法来分配通道。

4.2 通道分配函数 request_channel()

在下面的说明中，假设弹幕高度（也就是字体大小）为 h。

如果弹幕是顶部弹幕或飞行弹幕，request_channel()函数按照下面的过程执行：

1. 设置变量 n=1；
2. 判断[n,n+h-1]这个范围内的通道是否被顶部弹幕或飞行弹幕占用，如果没有被占用，则直接分配通道 n，结束执行，否则执行下一步；
3. 调用 poptext_conflict_top()函数判断如果使用该通道，是否会和现有的弹幕发生冲突（也就是弹幕发生重叠），如果不会造成冲突，则分配通道 n，结束执行，否则执行下一步；
4. 执行 n++，然后跳转到第 2 步继续执行。

如果弹幕是底部弹幕，request_channel()函数安装下面的过程执行：

1. 设置变量 n = VideoHeight – h + 1；
2. 判断[n-h+1,n]这个范围内的通道是否被其他底部弹幕占用，如果没有被占用，则直接分配通道 n，结束执行，否则执行下一步；
3. 执行 n–，然后跳转到第 2 步继续执行。

注意我们在查询可用通道时是依次检查所有的通道，这种方法效率很低。如果存在冲突的话，poptext_conflict_top()函数就会被调用很多次，而这个函数所做的工作是比较耗时间的，这样有可能造成性能下降，具体表现就是到“亿千万”的弹幕时浏览器会假死几秒钟。ytp 的 SVN73 版本已经使用一些方法跳过了期间大多数不需要检测的通道，这样就减少了很多 poptext_conflict_top()函数的调用。具体的方法虽然不复杂，但是繁琐，要在草稿纸上涂涂画画才能看出来，不过只要画出来以后情况就比较清楚了，于是在这里我也不详细说明具体方法的过程了，源代码中好像我也写了不少注释，看看源代码应该也能明白。

4.3 冲突检测函数 poptext_conflict_top()

从下到上排列的弹幕只有底部弹幕一种，所以不需要检测冲突。而飞行弹幕和顶部弹幕都是从上到下排列的，而且飞行弹幕不是固定的，于是有必要进行冲突检测。实际上不进行冲突检测的话，弹幕效果可能也不会差很多，进行冲突检测主要是为了尽量使弹幕集中在顶部，这样弹幕效果会更好。

申请通道的弹幕和已经占用当前通道的弹幕的组合有 4 种，其中只有 3 种可能会发生冲突，冲突情况如表所示。

申请通道的弹幕→ 是否可能发生冲突↘ ↓已经占用通道的弹幕	飞行	顶部
飞行	√ （情况 1）	√ （情况 3）
顶部	√ （情况 2）	╳ （情况 4）

所以我们只需要考虑前三种情况就可以了。为了方便叙述，下面将申请通道的弹幕称为“本弹幕”，已经占用了当前通道的弹幕称为“前弹幕”，显示弹幕的 TextField 也分别称为 PrevTextField（前弹幕）和 CurTextField（本弹幕）。

情况 1：

虽然两个弹幕都是飞行的，但是有可能前弹幕的飞行速度更快。如果是这种情况的话，本弹幕的弹出时间只要晚于前弹幕完整显示出来的时间就不会发生冲突。于是，只要同时满足下面两个条件，就不会发

生冲突（假设本弹幕弹出时间为 cpt，显示延续时间为 tst，前弹幕的显示延续时间为 pst）：

- (PrevTextField.width / PrevTextField.width + ViedoWidth) * pst <= cpt
- (PrevTextField.width + VideoWidth) / pst > (ThisTextField.width + VideoWidth) / tst

情况 2：

这种情况下，只要在前弹幕消失前，本弹幕的水平座标大于 PrevTextField 最右端的坐标就不会发生冲突，也就是满足下面的条件（假设本弹幕的弹出时间为 cpt，前弹幕的结束时间为 pet）：

VideoWidth - (CurTextField.width + VideoWidth) / tst * (pet – cpt) <= PrevTextField._x + PrevTextField.width

情况 3：

这种情况下，只要保证本弹幕显示的时候，PrevTextField 最右端的坐标小于本弹幕的水平座标就不会发生冲突，也就是满足下面的条件：

PrevTextField._x + PrevTextField.width < CurTextField._x

5 播放控制

弹幕和影片的播放时间是息息相关的，所以在对影片进行普通的播放控制的时候，也必须对弹幕进行操作，这样才能保证弹幕正确显示。

5.1 暂停

之前我们设置了一个用来定时执行 check_pop_poptext()函数的定时器。当暂停影片的时候，就应该停止这个定时器。虽然不停止也不会有什么问题，但是处于效率考虑，最好还是停用。

在显示顶部或底部弹幕的时候，我们已经设置了一个一次性的定时器用来删除这些弹幕。但如果暂停了影片，这些弹幕就不应该被删除。所以在删除这些弹幕之前，应该进一步判断这些弹幕应该被删除的时间是否已经超过了当前影片的时间，如果是才删除。如果不是，则应该再设置一个一次性定时器来定时删除弹幕，定时器的时间应该设置为当前影片时间与弹幕应该消失的时间的差。为了防止这个差值太小而导致频繁地重新创建定时器，可以限制定时器最小的时间为 100 毫秒。如果差值小于 100 毫秒，就忽略差值，而直接将定时器的时间设为 100 毫秒。当然你也可以根据实际情况设置得更快或更慢一些。

5.2 从暂停状态恢复播放

这时候应该启动用于执行 check_pop_poptext()函数的定时器。

5.3 定位影片

当用户拖动播放进度条的时候是最麻烦的。在这种情况下，首先要删除当前播放时间不应该存在的弹幕，然后要从弹幕列表中读取当前播放时间应该显示的弹幕，最后还要设置飞行弹幕的水平座标。

首先删除弹幕，这一步不复杂，只需要遍历当前显示的弹幕，删除其中弹幕消失时间小于当前影片时间的弹幕和弹出时间大于当前影片时间的弹幕就可以了。

接下来设置当前的飞行弹幕的水平坐标，这个计算很简单，这里就不说了。

最后应该从弹幕列表中读取应该出现的弹幕，然后对这个弹幕调用 show_poptext()函数。使用最简单的方法，我们应该从弹幕列表表头开始，依次判断每一个弹幕在当前播放时间是否应该显示，判断依据是弹幕的弹出时间小于当前播放时间且消失时间大于当前播放时间。假设当前影片时间为 CurrentClipTime，弹幕的弹出时间是 ps，消失时间是 es，则应该满足下面的条件：

```
ps < CurrentClipTime && es > CurrentClipTime
```

对满足条件的弹幕调用 show_poptext()函数显示出来。

为了便于遍历当前显示的弹幕，我们可以设置一个数组 current_poptext_list 来保存当前显示的弹幕。显示一个新的弹幕时，将弹幕的编号保存到数组中，删除弹幕的时候从数组中删除对应弹幕的编号。

5.4 停止播放

停止时的操作等同于暂停影片然后定位影片到 0 秒。

5.5 隐藏弹幕

隐藏弹幕时，应该停止用于执行 check_pop_poptext()函数的定时器，然后遍历 current_poptext_list 数组删除当前所有弹幕。

5.6 显示弹幕

显示弹幕操作和定位影片的最后一步操作一样，只需要依次查询弹幕列表中的弹幕，把应该在当前影片播放时间显示的弹幕显示出来就可以了。

其他需要考虑的问题

对于一个影片来说，其高度只有几百个像素，也就是说我们只有几百个通道可以使用，而一个弹幕就要占用十几个通道，这样几百个通道只能容纳大约十几个弹幕，显然这是不够的。不过，请注意在实施 request_channel()函数中查找可用通道的过程时，我们并没有指定通道编号的上下限，也就是说理论上我们可以无限制地查找通道，直到查找到一个可用通道。但是如果通道编号超过影片高度或者通道编号为负数的话，弹幕就看不到了，所以，我们还要对申请到的通道进行求模运算，模数就是影片高度，于是最终分配到的通道编号就是：

```
ActualChannelID = ActualChannelID % VideoWidth
```

这样处理的结果就是，当弹幕到达播放器边缘的时候，会从头开始排列。实际上现在一般的弹幕都是这样处理的，当整个屏幕容纳不下所有弹幕的时候，弹幕就会重叠起来。通过求模我们就可以实现这种效果。

6 结语与致谢

从 2008 年到现在 2009 年我一直在研究这个东西，到现在我竟然忘了我最初是为什么想研究弹幕算法了，不过有一点是可以确定的，就是虽然现在有很多弹幕网站，但是却没有人公布一个弹幕算法，这是原因之一，不过不是主要原因。不过没关系了，现在算法也弄出来了，管我最初的目的是什么呢。

虽然我觉得这个算法已经不错了，但是应该还有提升的空间，让算法过程更加结构化，数据结构更清晰，如果有宅感兴趣的话，就一起来做贡献吧。如果要联系我的话，请发邮件到 gs@bbxy.net。

在此感谢 cosechy 在我研究的时候帮助测试弹幕，yh 你提出的扩展真是太强大了，不过似乎强大得超出“生产关系”所能适应的“生产力”水平了，估计几年内不会有网站愿意这样做，就像某论坛一样。