X 中的混成器与 Composite 扩展

目录

在上篇文章「桌面系统的混成器简史」中我介绍了其它桌面系统中的混成器的发展史和工作原理，话题回到我们的正题 Linux 系统上，来说说目前 X 中混成器是如何工作的。这篇文章将比上一篇深入更多技术细节，不想看太多细节的可以直接跳过看结论。

原始的 X 的绘图模型

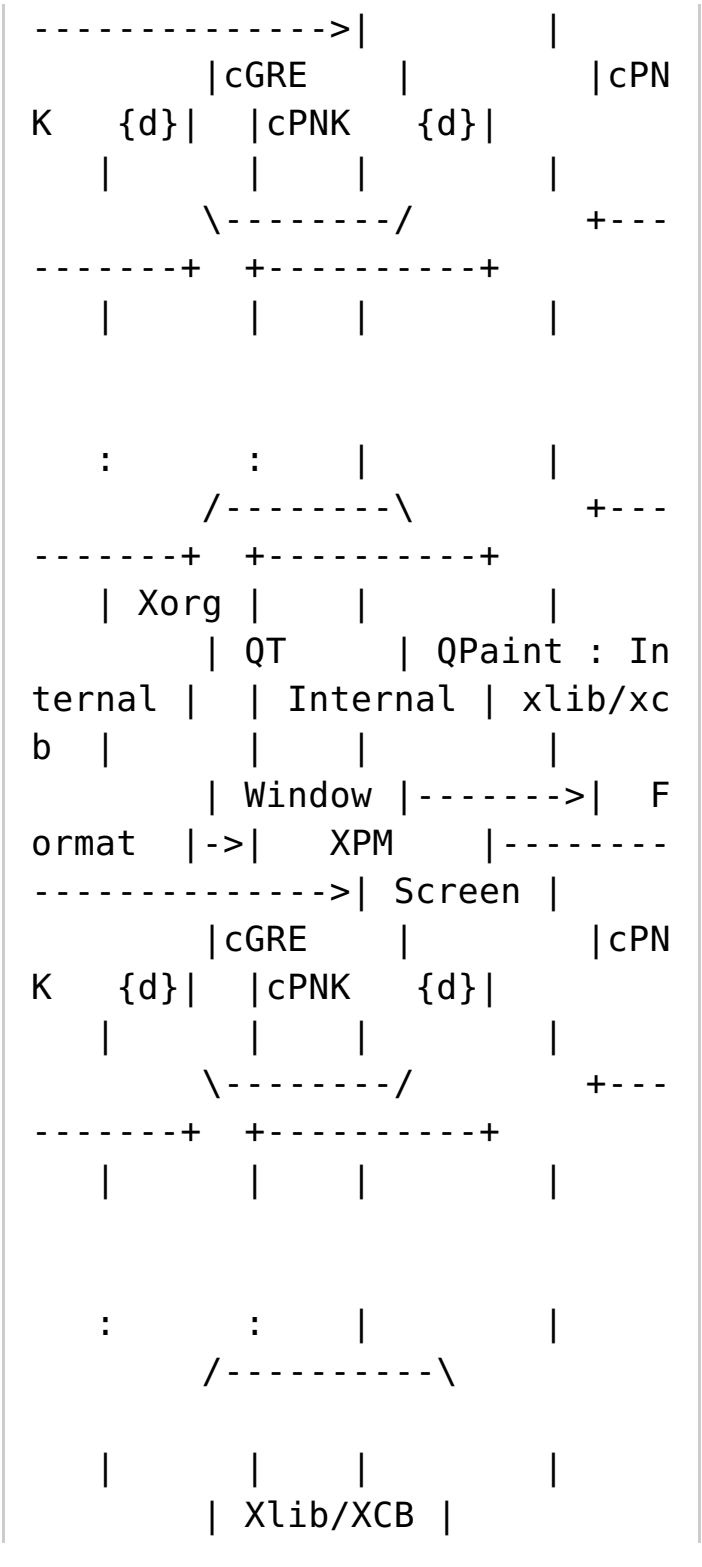
首先，没有混成器的时候 X 是这样画图的：

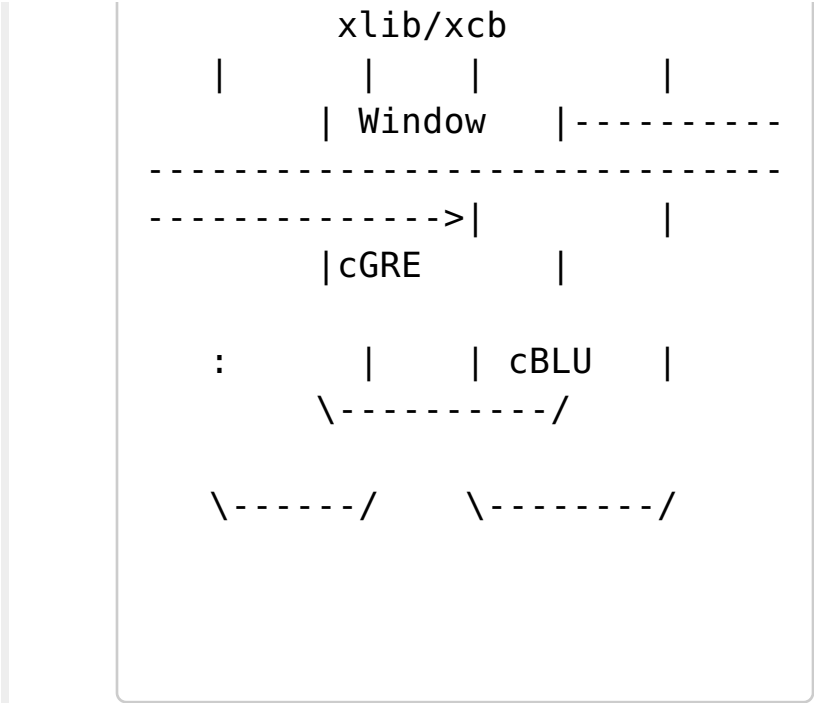
System Message: ERROR/3 (/home/travis/build/farseerfc/farseerfc/content-X-and-compositext.zhs.rst, line 23)

Error in "ditaa" directive:

```
.. ditaa::

        /--------\        +---
-------+  +----------+
   /------\    /--------\
        | GTK    | Cairo  : In
ternal |  | Internal | xlib/xc
b  :      |    |        |
        | Window |------->|
PDF    |->|   XPM    |--------
```


|cGRE |
|cPNK {d}|
|cPNK {d}|
| Xorg |
| QT |
| QPaint : Internal | xlib/xcb |
| Window |
| Format |
| XPM |
| Screen |
|cGRE |
|cPNK {d}|
|cPNK {d}|
| Xlib/XCB |



X 的应用程序没有统一的绘图 API 。GTK+ 在 3.0 之后统一用 Cairo 绘图， 而 Cairo 则是基于 PDF 1.4 的绘图模型构建的， GTK 的 2.0 和之前的版本中也有很大一部分的绘图是用 Cairo 进行， 其余则通过 xlib 或者 xcb 调用 X 核心协议提供的绘图原语绘图。 QT 的情况也是类似，基本上用 QPaint 子系统绘制成位图然后交给 X 的显示服务器。 显示服务器拿到这些绘制请求之后，再在屏幕上的相应位置绘制整个屏幕。 当然还有很多老旧的不用 GTK 或者 QT 的程序，他们则直接调用 X 核心协议提供的绘图原语。

值得注意一点是 X 上除了没有统一的绘图模型，也没有统一的矢量图格式。 X 核心协议的绘图原语提供的是像素单位的绘图操作，没有类似 GDI+ 或者 Quartz 提

Device Independence
供的 设备无关 的「点」的抽象。所以只用 X 的绘图原语的话，我们可以把 (1,1) 这个像素点涂黑，但是不能把 (0.5, 0.5) 这个点涂黑，这一设计缺陷在 Unix Hater's Handbook 中已经被吐槽过了。因为这个缺陷，所以直接用 X 绘图原语绘制的图像不能像 矢量图那样进行无损缩放。同样的缺陷导致 X 绘图原语绘制的字符不能做到 subpixel-level anti-aliasing 子像素级 抗锯齿 （这解释了默认配置下的 xterm 和 urxvt 中的字体渲染为什么难看 ）。相比之下 GDI 有对应的 WMF 矢量图格式， Quartz 有对应的 PDF 矢量图格式， 而 X 中没有这样的格式对应。因为没有统一的矢量图格式，所以无论是 Cairo 、QPaint 还是没有用这些绘图库但是同样在意字体和曲线渲染效果的程序（比如 Firefox 和 Chromium）都需要首先渲染到内部的 XPixMap 位图格式，做好子像素渲染和矢量缩放，然后再把渲染好的位图转交给 X 图形服务器。

通过 Composite 扩展重定向窗口输出

2004年发布的 X11R6.8 版本的 Xorg 引入了 Composite 扩展 。这个扩展背后的动机以及前因后果在一篇文章 The (Re)Architecture of the X Window

System 中有详细的表述。Composite 扩展允许某个 X 程序做这几件事情：

1. 通过 `RedirectSubwindows` 调用将一个窗口树中的所有窗口渲染重定向到 内部存储 (off-screen storage)。重定向的时候可以指定让 X 自动更新窗口的内容到屏幕上或者由混成器手动更新。
2. 通过 `NameWindowPixmap` 取得某个窗口的内部存储。
3. 通过 `GetOverlayWindow` 获得一个特殊的用于绘图的窗口，在这个窗口上绘制的图像将覆盖在屏幕的最上面。
4. 通过 `CreateRegionFromBorderClip` 取得某个窗口的边界剪裁区域（不一定是矩形）。

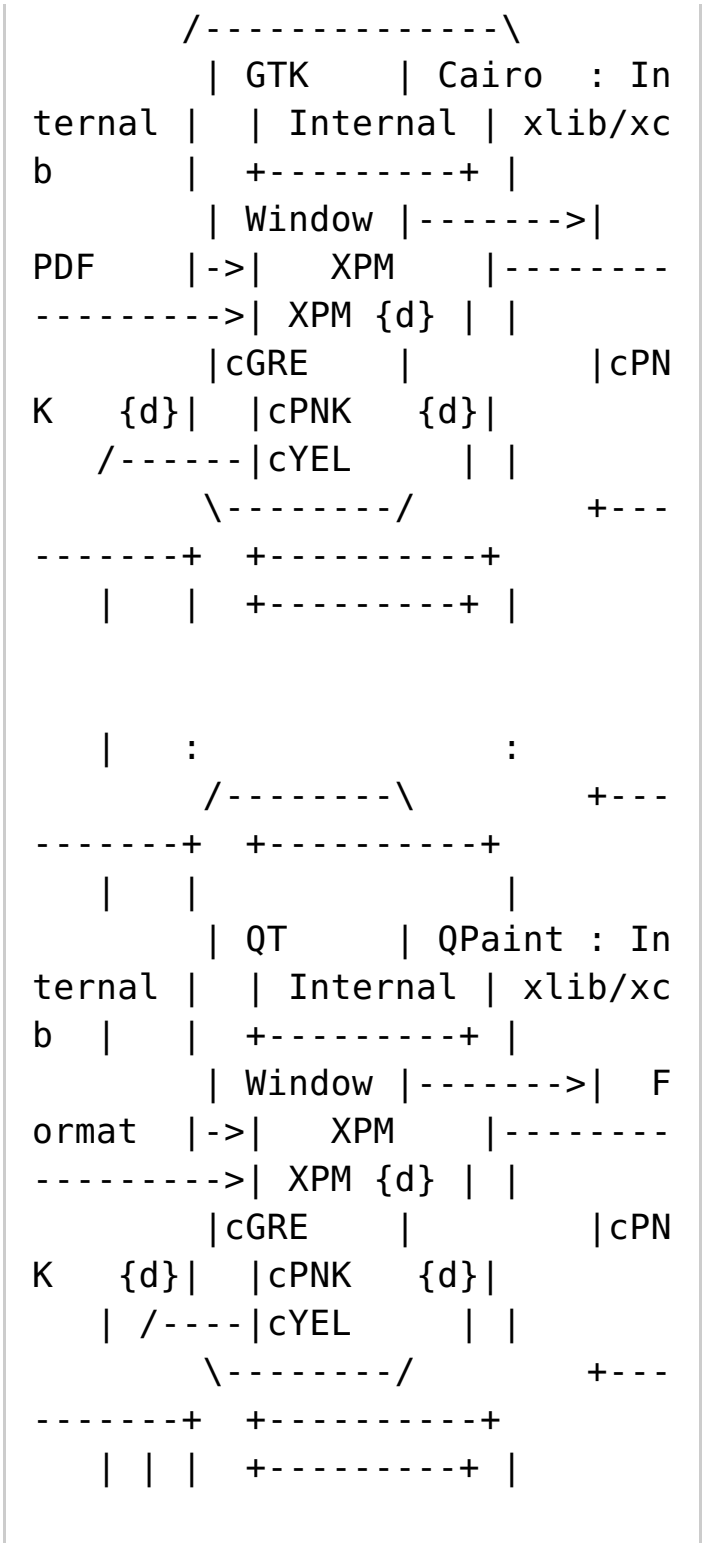
有了 Composite 扩展，一个 X 程序就可以调用这些 API 实现混成器。这里有篇 教学解释如何使用 Composite 扩展。开启了混成的 X 是这样绘图的：

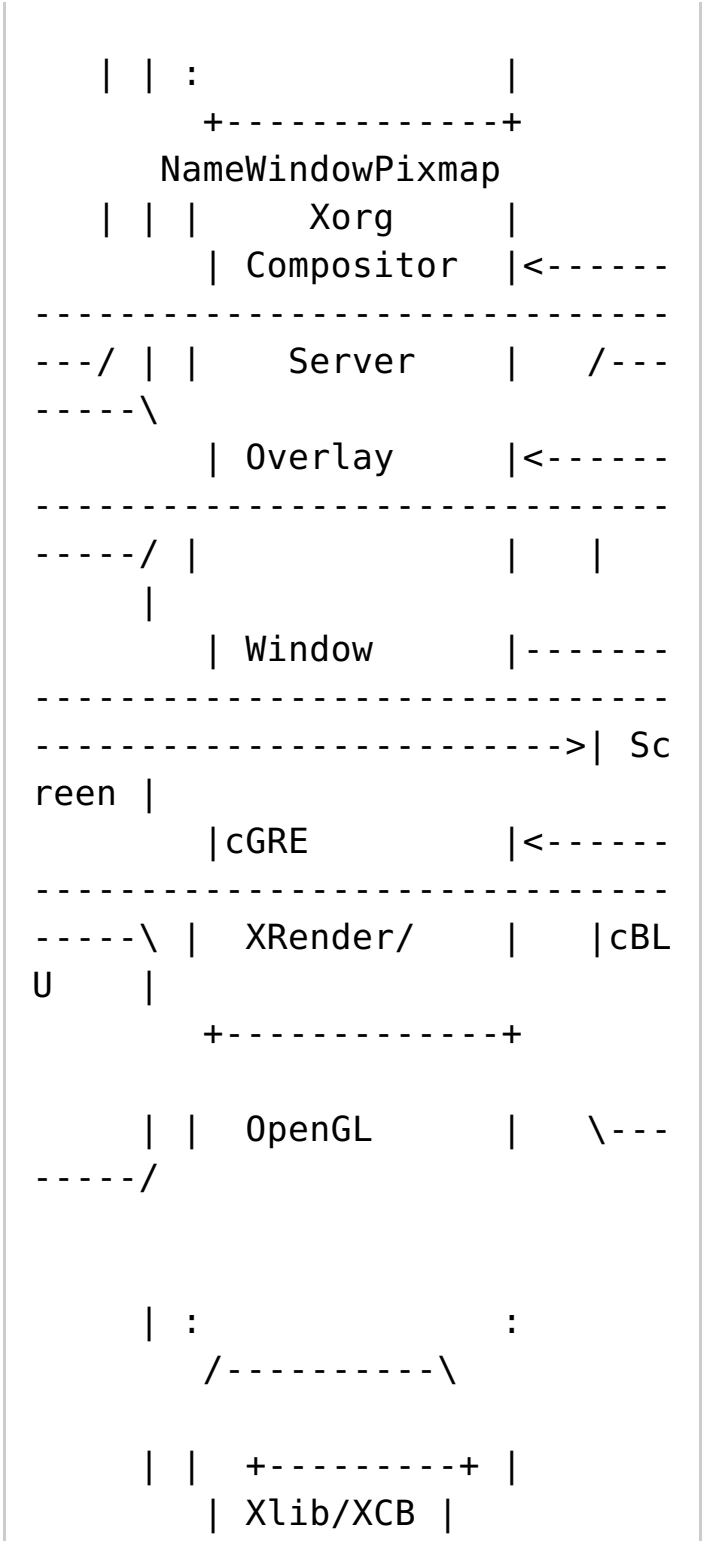
System Message: ERROR/3 (/home/travis/build/farseerfc/farseerfc/content-X-and-compositext.zhs.rst, line 92)

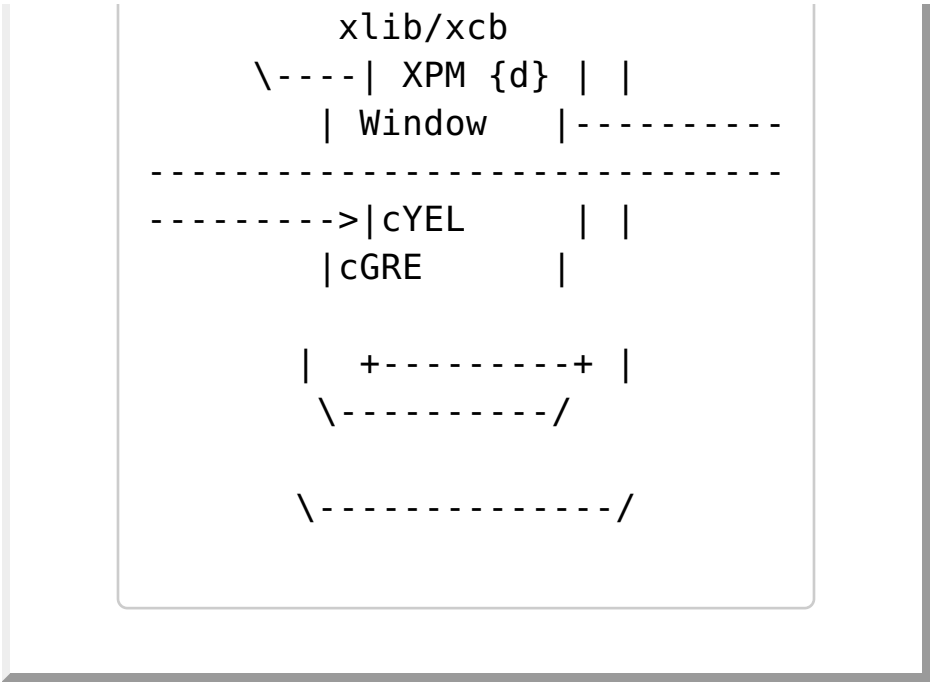
Error in "ditaa" directive:

```
.. ditaa::

        /--------\         +---
-------+  +----------+
```


/--------------\
| GTK | Cairo : In
ternal | | Internal | xlib/xc
b | +---------+ |
| Window |------->|
PDF |->| XPM |--------
---------->| XPM {d} | |
|cGRE | |cPN
K {d}| |cPNK {d}|
/------|cYEL | |
\--------/ +---
-------+ +----------+
| | +---------+ |
| : :
/--------\ +---
-------+ +----------+
| | |
| QT | QPaint : In
ternal | | Internal | xlib/xc
b | | +---------+ |
| Window |------->| F
ormat |->| XPM |--------
---------->| XPM {d} | |
|cGRE | |cPN
K {d}| |cPNK {d}|
| /----|cYEL | |
\--------/ +---
-------+ +----------+
| | | +---------+ |


| | :
+-------------+
NameWindowPixmap
| | | Xorg |
| Compositor |<------
---/ | | Server | /---
-----\
| Overlay |<------
-----/ | | |
|
| Window |-------
>| Sc
reen |
|cGRE |<------
-----\ | XRender/ | |cBL
U |
+-------------+
| | OpenGL | \---
-----/
| : :
/----------\
| | +---------+ |
| Xlib/XCB |



整个 X 的混成器模型与 Mac OS X 的混成器模型相比，有如下几点显著的区别：

1. 混成的部分是交由外部的程序完成的，对混成的绘制方式和绘制普通窗口一样。 出于效率考虑，绝大多数 X 上的混成器额外使用了 XRender 扩展或者 OpenGL/EGL 来加速绘制贴图。不过即使如此，还是不能避免同样的位图（内容不一定完全一致， 比如 X 可以在窗口交给它的位图上加上边框然后再返还给混成器） **在不同的三个程序之间来回传递** 。
2. RedirectSubwindows 调用针对的是一个窗口树，换句话说是一个窗口 及其全部子窗口，不同于 Mac OS X 中混成器会拿到全部窗口的输出。 这个特点其实并不算是限制，因为 X 中每个虚拟桌面都有一个根窗口，只要指定这个根窗口 就可以拿

到整个虚拟桌面上的全部可见窗口输出了。 反而这个设计提供了一定的自由度，比如我们可以用这个调用实现一个截图程序， 拿到某个特定窗口的输出，而不用在意别的窗口。

3. 为了让窗口有输出，窗口必须显示在当前桌面上，不能处于最小化 状态或者显示在别的虚拟桌面，

   mapped

   用 X 的术语说就是窗口必须处于 被映射 的状态。因此直接用上述方法 **不能得到没有显示的窗口的输出** ，比如不能对最小化的窗口 直接实现 Windows 7 中的 Aero Peak 之类的效果。这个限制可以想办法绕开， 比如在需要窗口输出的时候临时把窗口映射到桌面上，拿到输出之后再隐藏起来， 不过要实现这一点需要混成器和窗口管理器相互配合。

4. 不像 Mac OS X 的基于 OpenGL Surface 的绘图模

   device independent　　device dependent

   型是 设备无关 的，这里 X 的绘图模型是 设备相关的。 这既是优点也是缺点。从缺点方面而言，显示到 X 的位图输出因为设备相关性， 所以严格对应显示器的点阵，并不适合作为文档格式打印出来。当然无论是 Cairo 还是 QPaint 都提供了到 PostScript 或者 PDF 后端的输出，所以实用层面这个并不构成问题。 设备相关这一点的优点在于，绘制到 XPM 位图的时候，程序和绘图库是能拿到输出设备（显示器） 的特殊属性的，从而绘图库能考虑不同的色彩、分辨率、 DPI 或者

   subpixel　layout

   子像素布局 这些属性以提供最好的渲染效果。

Mac OS X 10.4 在设计的时候也曾考虑过提供无极缩放的支持，而这种支持到了 Mac OS X 10.5 中就缩水变成了 Retina 的固定 2 倍缩放。这种局面在 X 上没有发生正是因为 X 的绘图模型的这种设备相关性，而 Mac OS X 的混成器采用的 OpenGL Surface 则无视了这些设备相关的属性。

输入事件的重定向，这可能做到么？

通过上述 Composite 扩展提供的 API ，混成器可以把窗口的 **输出** 重定向到自己的窗口上。 但是仅仅重定向输出，整个 X 还不处于可用状态，因为 **没有重定向输入** 。 考虑一下用户试图用鼠标点击某个按钮或者文本框，这时鼠标处于的位置是在 OverlayWindow 上绘制的位置，这个鼠标事件会交给 OverlayWindow ，而用户期待这个事件被发送给他看到的按钮上。

需要重定向的事件主要有键盘和鼠标事件两大类（暂时先不考虑触摸屏之类的额外输入）。 由于 Composite 扩展并没有直接提供这方面的重定向 API ，这使得输入事件处理起来都比较麻烦，

假设要重定向键盘事件，混成器需要效仿输入法框架（fcitx, ibus, scim） 那样处理一部分按键事件并把其余事件转给具有输入焦点的程序。 看看现有的输入法框架和诸多程序间的问题，我们就能知道这里的坑有多深。 于是 **大部分 X 的混成器都不处理键盘事件重定向** 。再来看重定向鼠标事件，这边的坑比重定向键盘事件的坑更多， 因为不像重定向窗口输出那样只需要考虑 top-level 顶层 窗口， 重定向鼠标输入的时候要考虑所有子窗口（它们有独立的事件队列）， 以及要准确记录输入事件事件发生时的键盘组合键状态，还要正确实现 ICCCM/EWMH 中描述的转交窗口焦点的复杂规则，所有这些都已经在 X 中实现过的事情需要重新实现一遍。

由于坑太多难以实现，所以所有 X 下的混成器的实现方式都是直接忽略这个繁重的任务， **不重定向输入事件** 而把它交给 X 处理。具体的实现方式就是通过 XFixes 扩展提供的 `SetWindowShapeRegion` API 将 OverlayWindow 的 **输入区域** `ShapeInput` 设为空区域，从而忽略对这个 OverlayWindow 的一切鼠标键盘事件。 这样一来对 OverlayWindow 的点击会透过 OverlayWindow 直接作用到底下的窗口上。

因为选择了不重定向输入事件， X 下的混成器通常会处于以下两种状态：

1. 选择状态下可以缩放窗口的大小，扭曲窗口的形状，并且可以把窗口绘制在任意想要绘制的位置上（并不是移动窗口的位置）， **但是不能让用户与窗口的内容交互** 。

2. 正常状态下可以让用户与窗口的内容交互，但是**绘制的窗口位置、大小和形状必须严格地和 X 记录的窗口的位置、大小和形状保持一致**。持续时间短暂的动画效果可以允许位置和形状稍有偏差，但是在动画的过程中如果用户点击了 变形缩放过的窗口，那么鼠标事件将发往错误的（X 记录中的而非显示出的）窗口元素上。

可以发现这两种状态就直接对应了 Gnome 3 的普通 Activity 状态和缩略图状态（点击 活动 或者戳画面左上角之后显示的状态），这也解释了为什么尽管 Gnome 3 的窗口有硕大的关闭按钮，但是在缩略图状态下 Gnome 3 仍然需要给窗口加上额外的关闭按钮：**因为处于缩略状态下的窗口只是一张画而不能点**。

Composite 扩展的这些限制使得 X 下的混成器目前只能实现 Mac OS X 那样的 Exposé 效果，而不能实现 LG3D 那样直接在 3D 空间中操纵窗口内容。

解决重定向问题曾经的一缕曙光是 升阳公司 Sun Microsystems 在开发 LG3D 的过程中同时提议过另一个 X 扩展叫做 Event Interception 或者简称 XEvIE，这个扩展的设计目的就是提供 API 让某个程序接收并操纵全部的键盘和鼠标事件。可惜这个扩展随着升阳公司本身的陨落而 处于无人维护的状态，这一点也在它的官方网页上说明了：

> It has been suggested that this extension should not be used because it is broken and maintainerless.

Composite 扩展的不足

通过上面的介绍，我们就已经可以看到 Composite 扩展的不足之处了。 总结起来说，主要有两大不足：

1. 绘图效率低。因为同样的位图从应用程序传到 Xorg ，再从 Xorg 传到混成器， 最后从混成器再绘制到屏幕上，绕了一个大弯。这就是为什么 Wayland 的开发者在他的slide the real story behind Wayland and X 里这么说：

> and what's the X server?
> really bad IPC
>
> 那么 X 服务器到底做了什么呢？ 非常糟糕的进程间通讯

2. 没有重定向输入事件。如果我们要在 X 的混成器里做这个事情， 基本上我们要全部重写一遍 X 已经写好的窗口事件分发逻辑。

既然同样要重写，为什么不直接重写一遍 X 呢，扔掉那些历史负担，扔掉那些无用的 API ，重新设计可扩展的 API ，做好快速安全的 IPC —— 嗯，重写 X 就是 Wayland 的目的。

不过这么重写了的 Wayland 还是我们熟悉可爱的 X 么？它有哪些地方变样了？ 这将是我下一篇文章的内容。

附录：扩展阅读

我自己没有写过窗口管理器，没有写过混成器，没有写过 Wayland 程序，以上说的都是我从互联网上看到的整理出来的内容。写下本文的过程中我参考了这些文章：

The (Re)Architecture of the X Window System 这篇2004年写的文章描述了 Composite 扩展出现的动机和历史，介绍了绘图库的实现情况，涉及了上面所说的那些 X 扩展被用到的情况和可能。 同时这篇文章还展望了很多现在的 X 已然实现了的功能，比如 OpenGL 和 X 的

结合方面我们有了 GLX 和 AIGLX，比如内核的显卡支持方面我们有了 DRI 和 KMS。总之这是一篇描述 Linux 桌面未来的发展轨迹的非常有阅读价值的历史文献。

so you want to build a compositor 这是一篇 2008 年写的博文，介绍如何用 Clutter 实现一个最简单的混成器。

Composite tutorial 这是另一篇介绍如何实现一个简单的混成器的博文，用 Qt 实现，但是同样很底层。

unagi 这是一个可用的（但是已经长期没有开发的）类似 xcompmgr 的混成器。这个项目貌似 是一位研究生的硕士毕业设计，同时他公开了硕士学位的毕业论文 Master thesis: Writing an X compositing manager 其中也对实现一个简单的混成器做了详尽描述，包括介绍了相关的 X 扩展和调用。