

桌面系统的混成器 简史

目录

- 早期的栈式窗口管理器
- NeXTSTEP 与 Mac OS X 中混成器的发展
- 插曲：昙花一现的 Project Looking Glass 3D
- Windows 中的混成器
- 这就结束了？Linux 桌面呢？

（原本是想写篇关于 Wayland 的文章，后来越写越长感觉能形成一个系列，于是就先把这篇背景介绍性质

的部分发出来了。)

Linux 系统上要迎来 Wayland 了，或许大家能从各种渠道打听到 Wayland 是一个混成器，替代 X 作为显示服务器。那么 **混成器** 是个什么东西，桌面系统为什么需要它呢？要理解为什么桌面系统需要 **混成器**（或者它的另一个叫法，Compositing Window Manager 混成窗口管理器），在这篇文章中我想回顾一下历史，了解一下混成器出现的前因后果。

首先介绍一下混成器出现前主要的一类窗口管理器，也就是 Stacking Window Manager 栈式窗口管理器 的实现方式。

本文中所有桌面截图来自维基百科，不具有著作权保护。

早期的栈式窗口管理器

栈式窗口管理器的例子，Windows 3.11 的桌面



我们知道最初图形界面的应用程序是全屏的，独占整个显示器（现在很多游戏机和手持设备的实现仍旧如此）。所有程序都全屏并且任何时刻只能看到一个程序的输出，这个限制显然不能满足人们使用计算机的需求，于是就有了窗口的概念，有了桌面隐喻。

Desktop Metaphor

在桌面隐喻中每个窗口只占用显示面积的一小部分，有其显示的位置和大小，可以互相遮盖。于是栈式窗口管理器就是在图形界面中实现桌面隐喻的核心功能，其实现方式大体就是：给每个窗口一个相对的“高度”或者说“远近”，比较高的窗口显得距离用户比较近，会覆盖其下比较低的窗口。绘图的时候窗口管理器会从把窗口按高低排序，按照从低到高的顺序使用画家算法绘制整个屏幕。

这里还要补充一点说明，在当时图形界面的概念刚刚普及的时候，绘图操作是非常“昂贵”的。可以想象一下 800x600 像素的显示器输出下，每帧 真彩色 位图就要占掉 $(800 \times 600 \times 3 \approx 1.4 \text{ MiB})$ 的内存大小，30Hz 的刷新率（也就是30FPS）下每秒从 CPU 传往绘图设备的数据单单位图就需要 $(1.4 \times 30 = 41 \text{ MiB/s})$ 的带宽。对比一下当时的 VESA 接口 总的数据传输能力也就是 $(25 \text{ MHz} \times 32 \text{ bits} = 100 \text{ MiB/s})$ 左右，而 Windows 3.1 的最低内存需求是 1MB，对当时的硬件而言无论是显示设备、内存或是CPU，这无疑都是一个庞大的负担。

于是在当时的硬件条件下采用栈式窗口管理器有一个巨大 **优势**：如果正确地采用画家算法，并且合理地控制重绘时 **只绘制没有被别的窗口覆盖的部分**，那么无论有多少窗口互相遮盖，都可以保证每次绘制屏幕的最大面积不会超过整个显示器的面积。同样因为实现方式栈式窗口管理器也有一些难以回避的 **限制**：

1. 窗口必须是矩形的，不能支持不规则形状的窗口。
2. 不支持透明或者半透明的颜色。
3. 为了优化效率，在缩放窗口和移动窗口的过程中，窗口的内容不会得到重绘请求，必须等到缩放或者移动命令结束之后窗口才会重绘。

以上这些限制在早期的 X11 窗口管理器比如 twm 以及 XP 之前经典主题的 Windows 或者经典的 Mac OS 上都能看到。在这些早期的窗口环境中，如果你拖动或者缩放一个窗口，那么将显示变化后的窗口边界，这些用来预览的边界用快速的位图反转方式绘制。当你放开鼠

标的时候才会触发窗口的重绘事件。虽然有很多方法或者说技巧能绕过这些限制，比如 Windows XP 上就支持了实时的重绘事件和不规则形状的窗口剪裁，不过这些技巧都是一连串 hack，难以扩展。

NeXTSTEP 与 Mac OS X 中混成器的发展

NeXTSTEP 桌面



转眼进入了千禧年，Windows 称霸了 PC 产业，苹果为重振 Macintosh 请回了 Jobs 基于 NeXTSTEP 开发 Mac OS X。

NeXTSTEP 在当时提供的 GUI 界面技术相比较于同年代的 X 和 Windows 有一个很特别的地方：拖动滚动条或者移动窗口的时候，窗口的内容是 **实时更新** 的，这比只显示一个缩放大小的框框来说被认为更直观。而实现这个特性的基础是在 NeXTSTEP 中运用了 Display PostScript (DPS) 技术，简单地说，就是每个窗口并非直接输出到显示设备，而是把内容输出到 (Display) PostScript 格式交给窗口管理器，然后窗口管理器再在需要的时候把 PostScript 用软件解释器解释成位图显示在屏幕上。



比起让窗口直接绘制，这种方案在滚动和移动窗口的时候不需要重新渲染保存好的 DPS，所以能实现实时渲染。到了实现 Mac OS X 的时候，为了同时兼容老的 Mac 程序 API (carbon) 以及更快的渲染速度，以及考虑到 Adobe 对苹果收取的高昂的 Display PostScript 授权费，Mac OS X 的 Quartz 技术在矢量图的 PDF 描述模型和最终渲染之间又插入了一层抽象：



Mission Control



也就是说在 Mac OS X 中无论窗口用何种方式绘图，都会绘制输出成一副内存中的位图交给混成器，而后者再在需要的时候将位图混成在屏幕上。这种设计使得 2001年3月发布的 Mac OS X v10.0 成为了第一个广泛使用的具有软件混成器的操作系统。

到了 Mac OS X v10.2 的时候，苹果又引入了 Quartz Extreme 让最后的混成渲染这一步发生在显卡上。然后在 2003年1月公开亮相的 Mac OS X v10.3 中，他们公布

了 Exposé (后来改名为 Mission Control) 功能，把窗口的缩略图（而不是事先绘制的图标）并排显示在桌面上，方便用户挑选打开的窗口。

由于有了混成器的这种实现方式，使得可能把窗口渲染的图像做进一步加工，添加阴影、三维和动画效果。这使得 Mac OS X 有了美轮美奂的动画效果和 Exposé 这样的方便易用的功能。或许对于乔布斯而言，更重要的是因为有了混成器，窗口的形状终于能显示为他梦寐以求的圆角矩形了！

插曲：昙花一现的 Project Looking Glass 3D

在苹果那边刚刚开始使用混成器渲染窗口的 2003 年，昔日的 Sun Microsystems 升阳公司则在 Linux 和 Solaris 上用 Java3D 作出了另一个炫酷到没有朋友的东西，被他们命名为 Project Looking Glass 3D（缩写 LG3D，别和 Google 的 Project Glass 混淆呀）。这个项目的炫酷实在难以用言语描述，好在还能找到两段视频展示它的效果。





LG3D





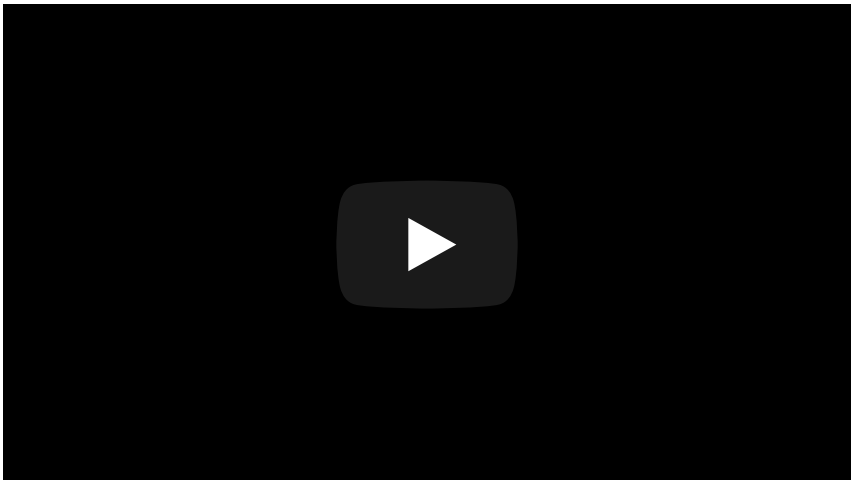
如视频中展示的那样，LG3D 完全突破了传统的栈式窗口管理方式，在三维空间中操纵二维的窗口平面，不仅像传统的窗口管理器那样可以缩放和移动窗口，还能够旋转角度甚至翻转到背面去。从视频中难以体会到的一点是，LG3D 在实现方式上与 Mac OS X 中的混成器有一个本质上的不同，那就是处于（静止或动画中）缩放或旋转状态下的窗口是 **可以接受输入事件** 的。这一重要区别在后面 Wayland 的说明中还会提到。LG3D 项目展示了窗口管理器将如何突破传统的栈式管理的框架，可以说代表了窗口管理器的未来发展趋势。

LG3D 虽然以 GPL 放出了实现的源代码，不过整个项目已经停滞开发许久了。官方曾经放出过一个 预览版的 LiveCD。可惜时隔久远（12年前了）在我的 VirtualBox 上已经不能跑起来这个 LiveCD 了……

更为可惜的是，就在这个项目刚刚公开展示出来的时候，乔布斯就致电升阳，说如果继续商业化这个产品，升阳公司将涉嫌侵犯苹果的知识产权（时间顺序上来看，苹果最初展示 Exposé 是在 2003 年 6 月 23 日的 Apple Worldwide Developers Conference，而升阳最初展示 LG3D 是在 2003 年 8 月 5 日的 LinuxWorld Expo）。虽然和乔布斯的指控无关，升阳公司本身的业务也着重于服务器端的业务，后来随着升阳的财政困难，这个项目也就停止开发不了了之了。

Windows 中的混成器

Longhorn 中的 Wobbly 效果



上面说到，Windows 系列中到 XP 为止都还没有使用混成器绘制窗口。看着 Mac OS X 上有了美轮美奂的动画效果，Windows 这边自然不甘示弱。于是同样在 2003 年展示的 Project Longhorn 中就演示了 wobbly 效果的窗口，并且跳票推迟多年之后的 Windows Vista 中实现了完整的混成器 Desktop Window Manager (DWM)。整个 DWM 的架构和 Mac OS X 上看到的很像：



和 Mac OS X 的情况类似，Windows Vista 之后的应用程序有两套主要的绘图库，一套是从早期 Win32API 就沿用至今的 GDI（以及 GDI+），另一套是随着 Longhorn 计划开发出的 WPF。WPF 的所有用户界面控件都绘制在 DirectX 贴图，所以使用了 WPF 的程序也可以看作是 DirectX 程序。而对老旧的 GDI 程序而言，它们并不是直接绘制到 DirectX 贴图的。首先每一个 GDI 的绘图操作都对应一条 Windows Metafile (WMF) 记录，所以 WMF 就可以看作是 Mac OS X 的 Quartz 内部用的 PDF 或者 NeXTSTEP 内部用的 DPS，它们都是矢量图描述。随后，这些 WMF 绘图操作被通过一个 Canonical Display Driver (cdd.dll) 的内部组建转换到 DirectX 平

面，并且保存起来交给 DWM。最后，DWM 拿到来自 CDD 或者 DirectX 的平面，把它们混合起来绘制在屏幕上。

值得注意的细节是，WPF 底层的绘图库几乎肯定有 C/C++ 绑定对应，Windows 自带的不少应用程序和 Office 2007 用了 Ribbon 之后的版本都采用这套绘图引擎，不过微软没有公开这套绘图库的 C/C++ 实现的底层细节，而只能通过 .Net 框架的 WPF 访问它。这一点和 OS X 上只能通过 Objective-C 下的 Cocoa API 调用 Quartz 的情况类似。

另外需要注意的细节是 DirectX 的单窗口限制在 Windows Vista 之后被放开了，或者严格的说是基于 WDDM 规范下的显卡驱动支持了多个 DirectX 绘图平面。在早期的 Windows 包括 XP 上，整个桌面上同一时刻只能有一个程序的窗口处于 DirectX 的 **直接绘制** 模式，而别的窗口如果想用 DirectX 的话，要么必须改用软件渲染要么就不能工作。这种现象可以通过打开多个播放器或者窗口化的游戏界面观察到。而在 WDDM 规范的 Vista 中，所有窗口最终都绘制到 DirectX 平面上，换句话说每个窗口都是 DirectX 窗口。又或者我们可以认为，整个界面上只有一个真正的窗口也就是 DWM 绘制的全屏窗口，只有 DWM 处于 DirectX 的直接渲染模式下，而别的窗口都输出到 DirectX 平面里（可能通过了硬件加速）。

由 DWM 的这种实现方式，可以解释为什么 窗口模式下的游戏总是显得比较慢，原因是整个桌面有很多不同的窗口都需要 DWM 最后混成，而如果在全屏模式下，

只有游戏处于 DirectX 的直接渲染方式，从而不会浪费对游戏而言宝贵的 GPU 资源。

由于 DWM 实现了混成器，使得 Vista 和随后的 Windows 7 有了 Aero Glass 的界面风格，有了 Flip 3D、Aero Peek 等等的这些辅助功能和动画效果。这套渲染方式延续到 Windows 8 之后，虽然 Windows 8 还提出了 Modern UI 不过传统桌面上的渲染仍旧是依靠混成器来做的。

这就结束了？Linux 桌面呢？

别急，我写这些文章的目的是想聊聊 Linux 中的混成器，尤其是 X 下现有的混成器和 Wayland，这篇文章只是个背景介绍。关于 X 中混成器的实现方式和限制，且听我下回分解。