

桌面系統的混成器 簡史

目錄

Contents

- 早期的棧式窗口管理器
- NeXTSTEP 與 Mac OS X 中混成器的發展
- 插曲：曇花一現的 Project Looking Glass 3D
- Windows 中的混成器
- 這就結束了？Linux 桌面呢？

（原本是想寫篇關於 Wayland 的文章，後來越寫越長感覺能形成一個系列，於是就先把這篇背景介紹性質的部分發出來了。）

Linux 系統上要迎來 Wayland 了，或許大家能從各種渠道打聽到 Wayland 是一個混成器，替代 X 作為顯示服務器。那麼 **混成器** 是個什麼東西，桌面系統為什麼需要它呢？要理解為什麼桌面系統需要 **混成器**（或者它的

Compositing Window Manager

另一個叫法，混成窗口管理器），在這篇文章中我想回顧一下歷史，瞭解一下混成器出現的前因後果。

首先介紹一下混成器出現前主要的一類窗口管理

Stacking Window Manager

器，也就是 棧式窗口管理器 的實現方式。

本文中所有桌面截圖來自維基百科，不具有著作權保護。

早期的棧式窗口管理器

棧式窗口管理器的例子，Windows 3.11 的桌面



我們知道最初圖形界面的應用程序是全屏的，獨佔整個顯示器（現在很多遊戲機和手持設備的實現仍舊如此）。所有程序都全屏並且任何時刻只能看到一個程序的輸出，這個限制顯然不能滿足人們使用計算機的需求，於是就有了窗口的概念，有了桌面隱喻。

Desktop Metaphor

在桌面隱喻中每個窗口只佔用顯示面積的一小部分，有其顯示的位置和大小，可以互相遮蓋。於是棧式窗口管理器就是在圖形界面中實現桌面隱喻的核心功能，其實現方式大體就是：給每個窗口一個相對的“高度”或者說“遠近”，比較高的窗口顯得距離用戶比較近，會覆蓋其下比較低的窗口。繪圖的時候窗口管理器會從把窗口按高低排序，按照從低到高的順序使用畫家算法繪製整個屏幕。

這裏還要補充一點說明，在當時圖形界面的概念剛剛普及的時候，繪圖操作是非常“昂貴”的。可以想象一下 800x600 像素的顯示器輸出下，每幀 真彩色 位圖 就要佔掉 $(800 \times 600 \times 3 \approx 1.4 \text{ MiB})$ 的內存大小，30Hz 的刷新率（也就是30FPS）下每秒從 CPU 傳往繪圖設備的數據單單位圖就需要 $(1.4 \times 30 = 41 \text{ MiB})$ 的帶寬。對比一下當時的 VESA 接口 總的數據傳輸能力也就是 $(25 \text{ MHz}) \times 32 \text{ bits} = 100 \text{ MiB/s}$ 左右，而 Windows 3.1 的最低內存需求是 1MB，對當時的硬件而言無論是顯示設備、內存或是CPU，這無疑都是一個龐大的負擔。

於是在當時的硬件條件下採用棧式窗口管理器有一個巨大 **優勢**：如果正確地採用畫家算法，並且合理地控制重繪時 **只繪製沒有被別的窗口覆蓋的部分**，那麼無論有多少窗口互相遮蓋，都可以保證每次繪製屏幕的最大面積不會超過整個顯示器的面積。同樣因為實現方式棧式窗口管理器也有一些難以迴避的 **限制**：

1. 窗口必須是矩形的，不能支持不規則形狀的窗口。
2. 不支持透明或者半透明的顏色。
3. 爲了優化效率，在縮放窗口和移動窗口的過程中，窗口的內容不會得到重繪請求，必須等到縮放或者移動命令結束之後窗口纔會重繪。

以上這些限制在早期的 X11 窗口管理器比如 twm 以及 XP 之前經典主題的 Windows 或者經典的 Mac OS 上都能看到。在這些早期的窗口環境中，如果你拖動或者縮放一個窗口，那麼將顯示變化後的窗口邊界，這些用來預覽的邊界用快速的位置反轉方式繪製。當你放開鼠

標的時候纔會觸發窗口的重繪事件。雖然有很多方法或者說技巧能繞過這些限制，比如 Windows XP 上就支持了實時的 重繪事件和不規則形狀的窗口剪裁，不過這些技巧都是一連串の hack，難以擴展。

NeXTSTEP 與 Mac OS X 中混成器的發展

NeXTSTEP 桌面



轉眼進入了千禧年，Windows 稱霸了 PC 產業，蘋果爲重振 Macintosh 請回了 Jobs 基於 NeXTSTEP 開發 Mac OS X。

NeXTSTEP 在當時提供的 GUI 界面技術相比較於同年代的 X 和 Windows 有一個很特別的地方：拖動滾動條或者移動窗口的時候，窗口的內容是 **實時更新** 的，這比只顯示一個縮放大小的框框來說被認爲更直觀。而實現這個特性的基礎是在 NeXTSTEP 中運用了 Display PostScript (DPS) 技術，簡單地說，就是每個窗口並非直接輸出到顯示設備，而是把內容輸出到 (Display) PostScript 格式交給窗口管理器，然後窗口管理器再在需要的時候把 PostScript 用軟件解釋器解釋成位圖顯示在屏幕上。



比起讓窗口直接繪製，這種方案在滾動和移動窗口的時候不需要重新渲染保存好的 DPS，所以能實現實時渲染。到了實現 Mac OS X 的時候，爲了同時兼容老的 Mac 程序 API (carbon) 以及更快的渲染速度，以及考慮到 Adobe 對蘋果收取的高昂的 Display PostScript 授權費，Mac OS X 的 Quartz 技術在矢量圖的 PDF 描述模型和最終渲染之間又插入了一層抽象：



Mission Control



也就是說在 Mac OS X 中無論窗口用何種方式繪圖，都會繪製輸出成一副內存中的位圖交給混成器，而後者再在需要的時候將位圖混成在屏幕上。這種設計使得 2001年3月發佈的 Mac OS X v10.0 成為了第一個廣泛使用的具有軟件混成器的操作系統。

到了 Mac OS X v10.2 的時候，蘋果又引入了 Quartz Extreme 讓最後的混成渲染這一步發生在顯卡上。然後在 2003年1月公開亮相的 Mac OS X v10.3 中，他們公佈

了 Exposé (後來改名為 Mission Control) 功能，把窗口的縮略圖（而不是事先繪製的圖標）並排顯示在桌面上，方便用戶挑選打開的窗口。

由於有了混成器的這種實現方式，使得可能把窗口渲染的圖像做進一步加工，添加陰影、三維和動畫效果。這使得 Mac OS X 有了美輪美奐的動畫效果和 Exposé 這樣的方便易用的功能。或許對於喬布斯而言，更重要的是因為有了混成器，窗口的形狀終於能顯示為他夢寐以求的圓角矩形了！

插曲：曇花一現的 Project Looking Glass 3D

在蘋果那邊剛剛開始使用混成器渲染窗口的 2003

Sun Microsystems

年，昔日的昇陽公司則在 Linux 和 Solaris 上用 Java3D 作出了另一個炫酷到沒有朋友的東西，被他們命名為 Project Looking Glass 3D（縮寫 LG3D，別和 Google 的 Project Glass 混淆呀）。這個項目的炫酷實在難以用言語描述，好在還能找到兩段視頻展示它的效果。





LG3D



如視頻中展示的那樣，LG3D 完全突破了傳統的棧式窗口管理方式，在三維空間中操縱二維的窗口平面，不僅像傳統的窗口管理器那樣可以縮放和移動窗口，還能夠旋轉角度甚至翻轉到背面去。從視頻中難以體會到的一點是，LG3D 在實現方式上與 Mac OS X 中的混成器有一個本質上的不同，那就是處於（靜止或動畫中）縮放或旋轉狀態下的窗口是 **可以接受輸入事件** 的。這一重要區別在後面 Wayland 的說明中還會提到。LG3D 項目展示了窗口管理器將如何突破傳統的棧式管理的框架，可以說代表了窗口管理器的未來發展趨勢。

LG3D 雖然以 GPL 放出了實現的源代碼，不過整個項目已經停滯開發許久了。官方曾經放出過一個預覽版的 LiveCD。可惜時隔久遠（12年前了）在我的 VirtualBox 上已經不能跑起來這個 LiveCD 了……

更為可惜的是，就在這個項目剛剛公開展示出來的時候，喬布斯就致電昇陽，說如果繼續商業化這個產品，昇陽公司將涉嫌侵犯蘋果的知識產權（時間順序上來看，蘋果最初展示 Exposé 是在 2003 年 6 月 23 日的 Apple Worldwide Developers Conference，而昇陽最初展示 LG3D 是在 2003 年 8 月 5 日的 LinuxWorld Expo）。雖然和喬布斯的指控無關，昇陽公司本身的業務也着重於服務器端的業務，後來隨着昇陽的財政困難，這個項目也就停止開發並不了了之了。

Windows 中的混成器

Longhorn 中的 Wobbly 效果



上面說到，Windows 系列中到 XP 為止都還沒有使用混成器繪製窗口。看着 Mac OS X 上有了美輪美奐的動畫效果，Windows 這邊自然不甘示弱。於是同樣在 2003 年展示的 Project Longhorn 中就演示了 wobbly 效果的窗口，並且跳票推遲多年之後的 Windows Vista 中實現了完整的混成器 Desktop Window Manager (DWM)。整個 DWM 的架構和 Mac OS X 上看到的很像：



和 Mac OS X 的情況類似，Windows Vista 之後的應用程序有兩套主要的繪圖庫，一套是從早期 Win32API 就沿用至今的 GDI（以及 GDI+），另一套是隨着 Longhorn 計劃開發出的 WPF。WPF 的所有用戶界面控件都繪製在 DirectX 貼圖上，所以使用了 WPF 的程序也可以看作是 DirectX 程序。而對老舊的 GDI 程序而言，它們並不是直接繪製到 DirectX 貼圖的。首先每一個 GDI 的繪圖操作都對應一條 Windows Metafile (WMF) 記錄，所以 WMF 就可以看作是 Mac OS X 的 Quartz 內部用的 PDF 或者 NeXTSTEP 內部用的 DPS，它們都是矢量圖描述。隨後，這些 WMF 繪圖操作被通過一個 Canonical Display Driver (cdd.dll) 的內部組建轉換到 DirectX 平

面，並且保存起來交給 DWM。最後，DWM 拿到來自 CDD 或者 DirectX 的平面，把它們混合起來繪製在屏幕上。

值得注意的細節是，WPF 底層的繪圖庫幾乎肯定有 C/C++ 綁定對應，Windows 自帶的不少應用程序和 Office 2007 用了 Ribbon 之後的版本都採用這套繪圖引擎，不過微軟沒有公開這套繪圖庫的 C/C++ 實現的底層細節，而只能通過 .Net 框架的 WPF 訪問它。這一點和 OS X 上只能通過 Objective-C 下的 Cocoa API 調用 Quartz 的情況類似。

另外需要注意的細節是 DirectX 的單窗口限制在 Windows Vista 之後被放開了，或者嚴格的說是基於 WDDM 規範下的顯卡驅動支持了多個 DirectX 繪圖平面。在早期的 Windows 包括 XP 上，整個桌面上同一時刻只能有一個程序的窗口處於 DirectX 的 **直接繪製** 模式，而別的窗口如果想用 DirectX 的話，要麼必須改用軟件渲染要麼就不能工作。這種現象可以通過打開多個播放器或者窗口化的遊戲界面觀察到。而在 WDDM 規範的 Vista 中，所有窗口最終都繪製到 DirectX 平面上，換句話說每個窗口都是 DirectX 窗口。又或者我們可以認為，整個界面上只有一個真正的窗口也就是 DWM 繪製的全屏窗口，只有 DWM 處於 DirectX 的直接渲染模式下，而別的窗口都輸出到 DirectX 平面裏（可能通過了硬件加速）。

由 DWM 的這種實現方式，可以解釋為什麼 窗口模式下的遊戲總是顯得比較慢，原因是整個桌面有很多不同的窗口都需要 DWM 最後混成，而如果在全屏模式下，

只有遊戲處於 DirectX 的直接渲染方式，從而不會浪費對遊戲而言寶貴的 GPU 資源。

由於 DWM 實現了混成器，使得 Vista 和隨後的 Windows 7 有了 Aero Glass 的界面風格，有了 Flip 3D、Aero Peek 等等的這些輔助功能和動畫效果。這套渲染方式延續到 Windows 8 之後，雖然 Windows 8 還提出了 Modern UI 不過傳統桌面上的渲染仍舊是依靠混成器來做的。

這就結束了？Linux 桌面呢？

別急，我寫這些文章的目的是想聊聊 Linux 中的混成器，尤其是 X 下現有的混成器和 Wayland，這篇文章只是個背景介紹。關於 X 中混成器的實現方式和限制，且聽我下回分解。