Budapesti Műszaki és Gazdaságtudományi Egyetem

Virtualizációs technológiák és alkalmazásaik (VIMIAV89)

Grafikus alrendszerek virtualizációja

Házi feladat

Írta : Heisenberger Viktor

2009.

Tartalomjegyzék

1 Bevezetés

A feladat a grafikus alrendszerek virtualizációjának lehetőségeit, módszereit részletesen megvizsgálni. Ehhez kiindulásként feldolgoztam egy a témához kapcsolódó, VMware mérnökök által publikált tudományos cikket. Ezután megkerestem a lehető legtöbb információt a különböző virtualizációs megoldásokban használt módszerekről, és ki is próbáltam ezek közül a számomra elérhető megoldásokat különböző grafikát igénylő szoftverekkel.

A következő részben röviden áttekintem a grafikus rendszerek működését annak érdekében, hogy a témában kevésbé jártas olvasók könnyebben megérthessék ezen rendszerek virtualizálásának problémakörét és lehetséges megoldásait. Azután rátérek a grafikus rendszerek virtualizációs lehetőségeire, végül pedig ismertetem az egyes rendszereket, leírom a velük kapcsolatos tapasztalataimat.

2 Grafikus rendszerek működése

Ebben a szakaszban ismertetésre kerül a grafikus rendszerek alapvető működése és felépítése. Mind a kétdimenziós, mind a háromdimenziós grafikát tárgyaljuk, hiszen néhány évvel ezelőtt elkezdődött a 3D rendszerek virtualizálásának megvalósítása, amit ma már követelményként kezdenek támasztani a virtualizációs rendszerekkel szemben.

2.1 2D grafikus rendszerek

A kétdimenziós grafikus rendszerek már jó ideje léteznek az informatikában. Azóta főleg a támogatott maximális felbontás növekedett jelentősen, azonban az alapelveik megmaradtak. A következő szakaszban a framebuffer eszközök kerülnek ismertetésre, azután pedig az overlay technika.

2.1.1 Framebuffer eszközök

A framebuffer eszközök a ma igen elterjedt személyi számítógépek első példányaiban már jelen voltak. Ide tartoznak a Hercules, CGA (Color Graphics Adapter), EGA (Enhanced Graphics Adapter), VGA (Video Graphics Array), SVGA (Super Video Graphics Array) és VESA[1] (Video Electronics Standards Association) eszközök. A működési elvük egyszerű : memóriába ágyazottan vezérelhető perifériaként állítható a megjelenített kép, továbbá szoftveres megszakítással parancsok adhatók ki nekik, ezek az úgynevezett VGA BIOS rutinok. Többféle üzemmódban működhetnek, amelyek között a méret és az egyszerre megjeleníthető színek száma tér el. A programozott megszakítással meghívott utasításokkal nemcsak a rajzolás állapotai állíthatók, hanem a monitornak küldött analóg jel időzítései is, hiszen ezeket az eszközöket analóg katódsugár-csöves monitorok vezérlésére tervezték.

A modernebb eszközök támogatnak alapvető 2D megjelenítéshez hardveres gyorsítást, a leggyakoribb gyorsított műveletek a vonalrajzolás és a blit[2].

A virtualizációs keretrendszereknek ismerniük kell ezeket az üzemmódokat, hiszen alapértelmezetten a modern grafikus operációs rendszereket ezeket próbálják használni általuk ismeretlen videohardver észlelésekor.

2.1.1.1 Megjelenítési üzemmódok

A framebuffer eszközök többféle üzemmódban működhetnek. Ezeket két fő csoportra lehet osztani : szöveges üzemmódok és grafikus üzemmódok.

A grafikus üzemmódokban a megjelenített kép egy négyzetrácsból áll, amelynek minden eleme a lehetséges színek egyikét veheti fel, egy cellát nevezünk pixelnek. EGA és annál újabb hardverek esetén nagyobb színmélység érhető el, mint amit a maximálisan egyszerre

[1] A VESA egy szabványosítási szervezet, itt most az általuk szabványosított interfészen keresztül programozható SVGA videokártyákról lesz szó (amelyeket VESA-kompatibilis videokártyáknak szokás nevezni). Szükség volt a szabványra, hiszen az SVGA nem egy szabvány (ellentétben a többiekkel), és minden gyártó valamelyest eltérő módon terjesztette ki a VGA képességeket.

[2] A blit művelet grafikus memóriablokkok együttes másolása. Neve az Amiga típusú számítógépekben az erre specializálódott segédprocesszortól ered, amelynek neve *blitter* (Block Image Transmitter). Az Amiga blittere képes a blokkmásoláson kívül vonalrajzolásra és sokszögek kitöltésére is.

megjeleníthető színek tárolására szolgáló bitek száma engedne, ezt paletta használatával éri el. A kiadott színkód valójában egy tömböt indexel (ez a paletta), amelyben a valóban megjelenített színek RGB kódja szerepel.

A szöveges üzemmódokban a megjelenített képet szintén egy rácsként képzelhetjük el, amelyben minden cellába egy karakter kerülhet. A karakterek kinézete állítható, ugyanis többféle betűtípus támogatott (ezek egy ROM-ban tároltak), illetve van módosítható karakterkészlet is, amelyben minden karaktert pixelenként adhatunk meg (ugyanis a tényleges megjelenítés a grafikus módokkal egyenértékű raszter-alapú).

A grafikus memória a rendszermemória `A000:0000`…`B000:FFFF` címtartományán érhető el, ez 128kB méretű. A videokártya DMA segítségével fogja ezt átmásolni a saját memóriájába, amelyből D/A konverterek segítségével állítja elő az analóg videojelet a monitor számára. Üzemmódtól függően értelmezett az egyes területek jelentése a tartományon belül. Karakteres módban a `B800:0000` címen kezdődik a karakteres adat sorfolytonosan, és minden karaktert két byte ír le. A felső byte a karakter és hátterének színét határozza meg, míg az alsó a kiválasztott karaktert. Grafikus módban az `A000:0000` címen kezdődik a képadat (64kB van fenntartva a képnek, a másik 64kB más adatok, pl. paletta, tárolására fenntartott), szintén sorfolytonosan, azonban színmélységtől függően más-más szervezésben. Monokróm kép esetén egy pixel színét egy bit határozza meg, így például az `11001100` bináris értékű byte-ot az `A000:0000` címre írva a legfelső sorban balról az első, második, ötödik, és hatodik pixeleket gyújtjuk ki. 16 színű üzemmódban (4 bit pixelenként) úgynevezett bitplane szervezés van, azaz a monokróm képszervezés mintájára minden pixelhez egy-egy bit tartozik a sorfolytonos képen (ez a kép a bitplane), de négy ilyen képet kell elhelyezni egymás után a memóriában, így minden kép első bitjét összeolvasva egy négybites értékként kapjuk az első pixel színét, mindegyik kép második bitjét összeolvasva kapjuk a második pixel színét, és így tovább. 256 színű üzemmódban minden pixel színét 8 biten kell tárolni, így az `A000:0000` címtől kezdve minden byte egy pixelt jelent, nincsenek bitplane-ek. Magas felbontás és színmélység esetén (pédául 640x480 pixel, 256 szín) már nem elég nagy a képnek fenntartott 64kB-os szegmens a teljes kép tárolására. Ezért az SVGA és VESA kártyákon tömbkapcsolást alkalmaznak : a videomemória azonos méretű[3] szegmensekre van osztva (ezeket *bank*nak nevezik angolul), és kapcsolhatjuk, hogy a memóriába írt adat melyik szegmensre kerüljön, azaz a kép melyik részét képezze.

A következő táblázatban néhány üzemmód adatai láthatók (1. táblázat).

1. táblázat Néhány videokártya üzemmód

Felbontás	Színek	Típus	Kód
80x25 karakter	16	szöveges	0x03
640x480 pixel	2	grafikus (VGA)	0x11
320x200 pixel	256	grafikus (VGA)	0x13
320x200 pixel	4	grafikus (CGA)	0x05
640x480 pixel	16777216	grafikus (VESA)	0x112[4]

[3] Jellemzően 64kB méretűek, azonban léteznek más mérettel dolgozó kártyák.
[4] Az SVGA és VESA üzemmódok két byte hosszúak, míg a korábbi szabványoké csak egy byte-os.

2.1.1.2 VGA BIOS rutinok

Programozott megszakítással (`int 10h`) lehet utasításokat kiadni ezeknek a videokártyáknak. A paraméterátadás a CPU regiszterein keresztül történik. A két legalapvetőbb utasítás a „Set video mode" (kód : `00h`) és „Get current video mode" (kód : `0Fh`) utasítások, előbbi beállítja a megadott kódú módot, míg utóbbi lekérdezi a jelenleg használt módot. A következő példában beállítjuk a VGA 13-as videomódot (1. kódrészlet).

```
mov ah, 00h {AH regiszterbe az utasítás kódja}
mov al, 13h {AL regiszterbe a beállítandó üzemmód}
int 10h     {utasítás kiadása}
```

1. kódrészlet VGA 13-as mód beállítása (320x200 pixel, 256 szín)

A VESA szabvány szerinti SVGA kártyák kiterjesztették a VGA rutinkészletet, de a hívási mechanizmusuk ugyanaz. A kiterjesztett rutinkészletet nevezik VBE-nek (VESA BIOS Extensions).

2.1.2 Overlay technikák

Egy modern grafikus, multitaszkos operációs rendszerben a felhasználói felület képének megalkotása nem triviális feladat. Az egyes ablakok egymást fedhetik, akár teljesen, akár részlegesen, egyesek ritkán frissülnek, egyesek pedig gyakran. Nem lehet egy-egy ablak frissítésénél újraszámolni a teljes képet, főleg akkor nem, ha az egyik ablak tartalma gyorsan változik. Jelentős erőforrásokat igényel ennek a feladatnak a megoldása, ha egy közös területre írja a képét az összes ablak, hiszen folyamatosan ellenőrizni kell, hogy melyik ablak takarja melyiket, és mennyire. Az adminisztrációs költségek csökkentésére született az overlay technika. Lényege, hogy az alkalmazás kap egy saját memóriaterületet (gyakran a videomemóriában helyezkedik el), ahova rajzolhatja az ablakának a képét, a közös területre csak a területét jelöli meg (például egy egyedi színnel kitölti). Megjelenítéskor pedig az egyedi azonosító alapján mindig a saját memóriaterületről a közös kép megfelelő helyére kerül az ablak tartalma.

Már jó ideje képesek a grafikus kártyák az overlayt hardveresen megoldani, ráadásul képesek a helyettesítő képet transzformálni (például átméretezés, színtér konverzió[5]), mindezt a CPU terhelése nélkül. Ma már igen sok alkalmazás támaszkodik az overlay alapú megjelenítésre, főként médialejátszók. A virtualizációs keretrendszerekben tehát meg kell oldani ennek a funkciónak a támogatását.

2.2 3D grafikus rendszerek

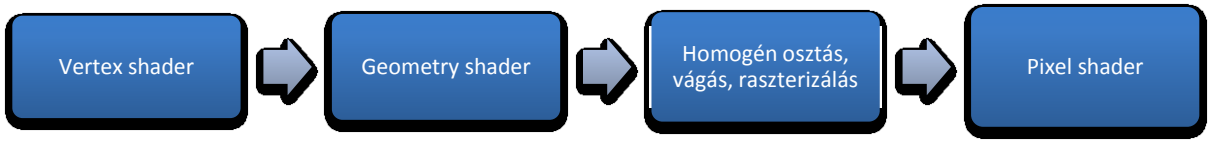
A háromdimenziós valós idejű számítógépes grafika az elmúlt évtizedben óriási fejlődésen ment keresztül. Eljutottunk az alacsony felbontású, korlátozott képességű szoftveres rendereléstől a hardveresen gyorsított inkrementális képszintézishez, amelyet teraFLOPS-ban mérhető teljesítményű GPU-k szolgálnak ki. A számítógépes játékokat és tervezőrendszereket

[5] Itt a színtér színek digitális ábrázolását jelenti. Például egy gyakori konverzió az RGB és YUV színterek közötti, amikor tunerkártya képét nézzük.

mára kinőtte a technológia, és olyan új alkalmazásokkal rendelkezik, mint az általános célú számítások (GPGPU) vagy a modern grafikus felhasználói felületek gyorsítása (hardware compositing). Egy nagyon új felhasználási terület pedig a GPU-alapú cloud computing (NVIDIA RealityServer, AMD Fusion Render Cloud). Ha virtuális gépekben is elérhető lenne hardveres gyorsítás, akkor az azt igénylő alkalmazásoknál is élvezhetők lennének a virtualizáció előnyei (például legacy alkalmazások futtatása modernebb rendszereken, könnyebb szoftverfejlesztés és tesztelés, izoláció, …), továbbá a ma már virtualizáción alapuló üzletágak kiterjeszthetik portfóliójukat nagyteljesítményű szolgáltatásokkal (például High Performance Computing szolgáltatások bérbe adása).

2.2.1 Elvi működés

A modern 3D gyorsítók az inkrementális képszintézisnek nevezett képalkotási technikát támogatják. Ez egy pipeline szervezésű rendszer, amelynek kezdetben minden fokozata fix funkcióval rendelkezett, a programozó csak előre definiált kapcsolókat állíthatott rajtuk. Ma már nemcsak programozhatóvá vált a legtöbb fokozat, hanem új fokozatok is bevezetésre kerültek. A hardveresen gyorsított 3D grafika programozására mára két API terjedt el : a platformfüggetlen OpenGL és a csak Windowson elérhető Direct3D. A következő ábrán látható a pipeline egyszerűsített változata (1. ábra).



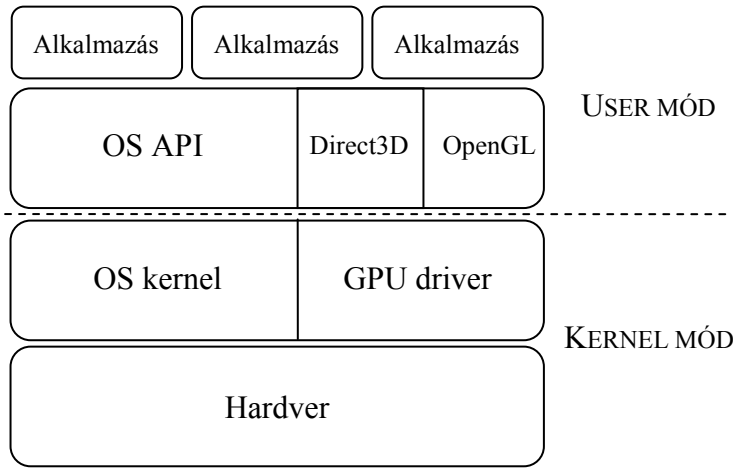
1. ábra Grafikus pipeline egyszerűsített változata

A pipeline háromszöghálókkal és textúrákkal dolgozik. Először a háromszöghálókat transzformálja a kamera és az objektum elhelyezkedése alapján a végső helyére, ezt végzi a vertex shader. Ezután következik[6] a nem kötelező geometry shader, amely módosítani képes a kapott geometrián. Eddig háromszögekkel dolgozott a csővezeték, a homogén osztás és vágás segítségével a képernyőn nem látható háromszögek kerülnek levágásra, majd a raszterizáló minden háromszögre eldönti, hogy a képernyő melyik pixeleit fedik le, és ezt az információt küldi csak tovább. A pixel shader felel minden egyes pixel színének a meghatározásáért.

Az OpenGL és Direct3D API-kat bármely modern programozási nyelvből elérhetjük, a shaderekhez azonban ezek nem ideálisak, így azok programozására külön nyelveket készítettek. Az OpenGL testülete a GLSL (GL Shading Language) nyelvet alkotta API-jukhoz, Direct3D-ben a HLSL (High Level Shading Language) használatos, míg az Nvidia Cg nyelve mindkét API-ban használható. Mindegyik erősen hasonlít a többire, szintaktikájukat a C nyelvből vették, amit kiterjesztettek vektortípusokkal és a grafikában gyakran használatos függvényekkel. A grafikus processzorok fejlődésével ezeket a nyelveket is kiterjesztették az újabb funkciók kihasználására. A grafikus processzorok képességeit kategóriákba lehet sorolni aszerint, hogy a shaderek specifikációjának melyik verzióját támogatják. Direct3D terminológiában ezeket hívják „*Shader Model*nek”.

[6] A Direct3D 11 által bevezetett tesszelációs szolgáltatások három új nem kötelező állomást vezettek be a vertex shader és a geometry shader közé.

Fontos még tudni az ezen API-kra épülő programok, maga az API-k, és az operációs rendszer együttes architektúráját. Ez egy rétegzett architektúra, amelyben legfelül vannak a felhasználói 3D-s grafikát megjelenítő programok, közvetlenül alattuk a grafikus API, amely alatt a kernel módban futó grafikus kártya driver[7] található. Ezt szemlélteti a következő ábra (2. ábra).



2. ábra 3D alkalmazások vázlatos architektúrája

A Direct3D és OpenGL API hívások a grafikus hardver driveréig jutnak, amely elvégzi a kártyán a kívánt műveleteket. A shaderprogramokat a driver lefordítja a gyártó és GPU-specifikus reprezentációjára, amelyet a GPU már képes futtatni.

Virtualizáció során (akár hosted, akár bare-metal, a problémakör azonos) az ábrán az „Alkalmazás" helyén virtuális gépek is vannak, amelyekben ez a teljes rétegszerkezet megtalálható, így a feladat vagy a külső driver réteget „bejuttatni" a virtuális gépbe, vagy a virtuális gépben lévő API réteget „kihozni" a gazda rendszerbe.

Fontos megjegyezni, hogy a grafikus kártya drivere, illetve maga a grafikus hardver nem rendelkezik publikus dokumentációval, pontos belső szerkezetüket és működésüket gyártóik cégtitoknak tekintik, ami egyes virtualizációs megoldási lehetőségek megvalósításának komoly gátat szab.

2.2.2 Hardver megvalósítások

Kezdetben a grafikus processzorok felépítése követte a logikai pipeline szervezését. Fizikailag is pipeline architektúrájú processzorok voltak, amelyek minden egyes állomáshoz tartalmaztak specializált feldolgozóegységet. Az újabb fejlesztések a nagyobb teljesítményt az órajelek és a feldolgozóegységek számának növelésével, továbbá a gyorsítótárak optimalizálásával érték el.

Ez a korszak zárult le az első Direct3D 10 képességű GPU-k megjelenésével (NVIDIA GeForce 8 sorozat, később pedig az ATI Radeon HD 2000 sorozat). Ezek a processzorok és utódaik már az úgynevezett SIMT (Single Instruction Multiple Thread) architektúrát követik.

[7] Windows Vista és újabb Windowsokban a grafikus driver egy része user módban fut, a stabilitás érdekében. Ezekben a rendszerekben a 2. ábra is kicsit más.

Az architektúra lényege, hogy kis, egyszerű SIMD feldolgozókból sokat alkalmaznak egyszerre. Ezeket blokkokba szervezik, minden blokk rendelkezik saját ütemezővel, gyorsítótárral, továbbá van egy kitüntetett, chipszintű ütemező. A raszterizálásra és raszteres műveletekre továbbra is specializált feldolgozókat alkalmaznak. Ez a processzor képes ugyanazt a kódot sok példányban párhuzamosan futtatni több szálon, amelyben az egyes példányok lefutása eltérhet egymástól (például eltérő elágazási utak, ciklus iterációk száma). Látható, hogy ez egy sokkal rugalmasabb architektúra, amely ráadásul lehetővé tette a GPGPU gyors fejlődését.

A virtualizáció szempontjából ez az architektúra sajnos nagyon kedvezőtlen, ugyanis nem arra tervezték, hogy menthető és visszatölthető állapotú legyen, hagyományos értelemben vett kontextus-váltásra képtelen (az aktuális kontextus folyamatban lévő számítását teljesen be kell fejeznie, mielőtt kontextust válthatna, egyszerre csak egy aktív kontextus lehet[8], még akkor is, ha vannak szabad erőforrások).

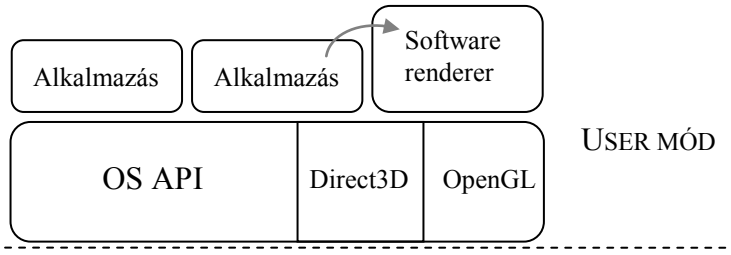
[8] Az NVIDIA Fermi kódnevű új fejlesztése lesz az első GPU, amely támogatja több kontextus párhuzamos futtatását, és az erőforrások dinamikus elosztását közöttük.

3 Virtualizációs lehetőségek

Ebben a szakaszban mutatom be a grafikus rendszerek virtualizálásának lehetőségeit, kitérve mindegyik megoldás előnyeire és hátrányaira, alkalmazásának feltételeire. Ahol nincs szabványos elnevezés, ott az elnevezések a VMware terminológiáját követik.

3.1 Szoftveres képalkotás (Software rendering)

Szoftveres képalkotásnál a CPU-n futó program (angol neve *software renderer*) felel a kép kiszámításáért. Ez a szoftverkomponens kapja meg az összes szükséges információt a kép megalkotásához. Ezt a módszert alkalmazták a hardveres 3D gyorsítás előtti 3D játékok, ma pedig jellemzően az off-line fotorealisztikus képszintézisnél alkalmazzák (például filmtrükkök, animációs filmek készítéséhez). Ilyenkor a képalkotást kérő szoftver tud róla, hogy egy szoftver számolja ki neki. Ezt illusztrálja a **3. ábra**.



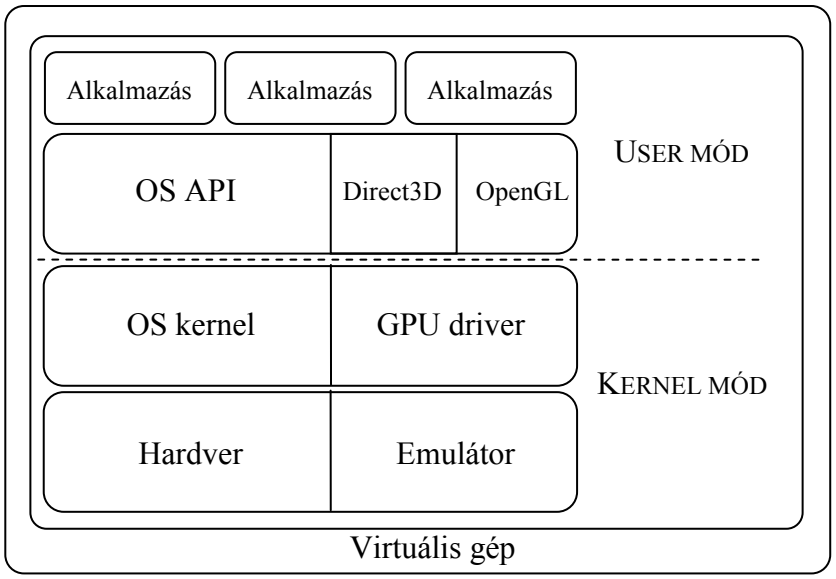
3. ábra Szoftveres képalkotás illusztráció

Előnyei: Ezzel a módszerrel a képminőség tökéletes, azaz pont azt számolja ki, amit a vendég gép akart. Egyszerre nyújtható grafikai szolgáltatás több futó virtuális gépnek. Lehetséges olyan grafikai szolgáltatásokat nyújtani a vendégnek, amelyeket a gazda szoftvere és hardvere nem lenne képes nyújtani. Rugalmas, hiszen a szoftverkomponens könnyen cserélhető, kevésbé hajlamos kompatibilitási problémákat okozni. Támogatott a futás felfüggesztése, az állapotának mentése és visszatöltése, továbbá a live migration is működik vele.

Hátrányai: Nagyon gyenge teljesítmény jellemzi, nagy erőforrásigénnyel (processzor és memória egyaránt). Nem könnyű implementálni.

3.2 Tiszta emuláció (Device emulation)

A tiszta emuláció alapelve, hogy egy szoftver szimulálja egy hardver működését. Amikor az emulált hardver szolgáltatásait egy szoftver igénybe venné, akkor nem tud róla, hogy valójában egy szoftverkomponens végzi el a munkát, és nem egy hardverelem. Emulációt akkor szokás használni, amikor más platformra vagy rendelkezésre nem álló hardverre írt szoftvert szeretnénk futtatni. A következő ábra szemlélteti a felépítését (**4. ábra**).



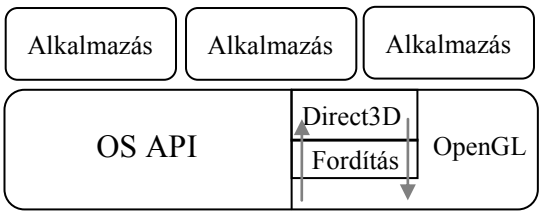
4. ábra Tiszta emulációra példa

Előnyei: A tökéletesen megírt emulátor azonos képminőséget ad az általa emulált hardverhez viszonyítva. Minden olyan szoftver fut vele, ami a fizikai hardveren is futna. A futása felfüggeszthető, állapota menthető és visszatölthető, live migrationt is támogat. Olyan szolgáltatásokat is lehet nyújtani, amelyekre a futtató gép hardvere képtelen.

Hátrányai: Nincs tökéletesen megírt emulátor. A leglassabb megoldás, csak a futtató hardverhez képest lassú hardvert érdemes emulálni, nagy erőforrásigény jellemzi. Nagyon nehéz implementálni, ugyanis az emulálni kívánt hardver pontos, mély ismerete és jól dokumentáltsága szükséges (GPU-k esetében a dokumentáció a gyártó cég titka). A hardver hibáinak ismerete is szükséges.

3.3 API fordítás

API fordítás során egy API szolgáltatásait nyújtjuk, de ezek implementációja egy másik, azonos szolgáltatásokat nyújtó API felhasználásával történik. Például a Linux világban ismert Wine és Cedega Windowsra készített programok futtatását végzi Linuxon. Ezt úgy valósítják meg, hogy a Windowsra írt programok számára a Windows API szolgáltatásait nyújtják, de valójában a futó Linux API hívásait használják fel, „lefordítják” rá a Windows API hívásokat. Ezt a megoldást nem lehet egyedül használni, ha virtuális gépben akarunk hardveres 3D gyorsítást, a használt API gyorsítását valahogy meg kell oldani. Az 5. ábra szemlélteti az API fordítást.



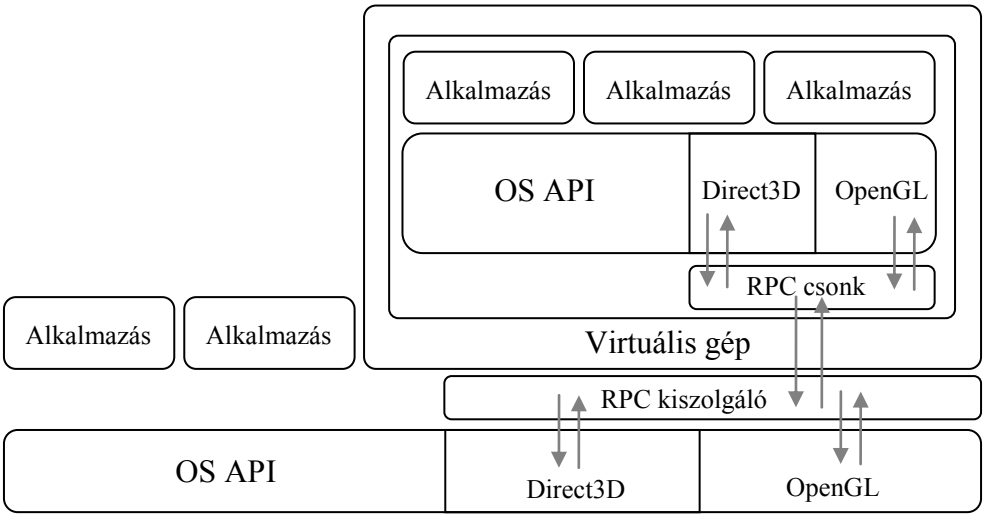
5. ábra API fordítás példa (Direct3D fordítása OpenGL-re)

Előnyei: A „lefordított” API-ra épülő szoftverek is működnek.

Hátrányai: Hosszadalmas implementáció (a „lefordított” API minden függvényét meg kell valósítani), a két API alapos ismerete szükséges.

3.4 API távoli eljáráshívás (API remoting)

Ebben a megoldásban a virtuális gépben lévő API belépési pontok által meghívott implementációt kicserélik, hogy a gazda operációs rendszerbe továbbítsák az API hívásokat távoli eljáráshívás jelleggel. A virtualizációs keretrendszer felügyeli a kommunikációt, és esetleg módosít a meghívott függvények paraméterein. Használatának előfeltétele, hogy a gazda operációs rendszer képes legyen hardveresen gyorsított 3D használatára. Ezt a technikát szemlélteti a 6. ábra.



6. ábra API távoli hívás

Előnyei: Jó teljesítményt nyújt. A képminőség megközelíti a natívan futó alkalmazásokét (ideális esetben azonos minőség érhető el). Egyszerre több virtuális gépnek nyújtható hardveres gyorsítás, felfüggeszthető a virtuális gép futása.

Hátrányai: Hosszadalmas implementáció, ugyanis minden API függvényt át kell írni. Nem menthető és visszaállítható a virtuális gép futásának pillanatnyi állapota (ezáltal a live migration sem lehetséges), ugyanis az API kontextus objektum és az összes állapotleíró a gazda gépen van.

3.5 „Fix átengedés” (Fixed pass-through)

Ennek a módszernek az alapötlete az, hogy rendeljünk fizikai hardvert közvetlenül virtuális géphez. Ezzel a gazda operációs rendszertől elvesszük, és a vendég operációs rendszer úgy látja, mintha a virtuális hardvernek a része lenne a fizikailag létező hardverelem. Ehhez a futtató fizikai számítógépnek rendelkeznie kell úgynevezett IOMMU (I/O Memory Management Unit) technológiával[9] (az Intel „Virtualization Technology for Directed I/O-

[9] Intel vonalon a Q35, Q45, X38, X48, X58, 3200 sorozatú, 5500 sorozatú lapkakészletek képesek a VT-d technológia használatára, azonban az alaplap gyártója letilthatja (akár szoftveresen a BIOS-ban, akár

nak", röviden VT-d-nek, az AMD IOMMU-nak nevezi). Ez a technológia a DMA-kat, megszakításokat, és I/O műveleteket átirányítja, így a vendég operációs rendszer biztonságosan használhatja az átadott hardvert, nem veszélyeztetve a gazda operációs rendszer működését.

Előnyei: Ez közelíti meg legjobban a gazda gépen történő futtatás teljesítményét, és azonos képminőséget ad. A GPU gyártó driverén nem kell módosítani, ugyanúgy kezeli a kártyát, mintha gazda operációs rendszeren lenne.

Hátrányai: Egy hardverelemet egyszerre csak egy futó virtuális géphez lehet rendelni, ha egyszerre több futó virtuális gépnek kell hardveres gyorsítás, akkor annyi grafikus kártyát kell a futtató számítógépbe szerelni, ahány egyszerre futó virtuális gép van, plusz egyet a gazda operációs rendszernek. Nem lehet felfüggeszteni a virtuális gépek futását, elmenteni és visszatölteni pillanatnyi állapotot, live migrationt végezni. Eltérő grafikus hardverű gazda gépre átmásolva a kikapcsolt állapotú virtuális gépet, a drivert le kell cserélni. A gazda operációs rendszertől és az alkalmazott hardvertől függően speciális driverre lehet szükség a gazda operációs rendszerben[10].

3.6 „Közvetett átengedés" (Mediated pass-through)

Ennek a megoldásnak a célja a „fix átengedés" teljesítményének jó megközelítése, de ugyanakkor rugalmasabb működés elérése. Az ötlet, hogy legyenek kontextus objektumok, amelyeken lehet elérni a grafikus kártyák funkcióit, és minden virtuális gép rendelkezik egy vagy több ilyennel. A VMM gondoskodik ezen kontextusok és a grafikus memória menedzseléséről. A nagy teljesítményigényű feladatokat a „fix átengedés" technikájával oldja meg a rendszer (egy kontextuson belül ez megtehető), a lassú műveleteket pedig más módszerrel (például emulációval). Ehhez elengedhetetlen, hogy a GPU-k képesek legyenek ezen kontextusok kezelésére. Szükséges továbbá egy grafikus driver a VMM-nek és a vendég operációs rendszereknek, amelyek közül egyik sem lehet olyan, mint egy szokványos egyetlen futó operációs rendszerrel rendelkező számítógép esetében. A VMM drivere pusztán menedzseli az erőforrásokat (kontextusok, memória), a vendégek drivere pedig hol hagyományos driverként viselkedik, hol másképp működik (például emulál). Látható, hogy a vendég drivere erősen paravirtualizált.

Előnyei: Nagyon jó teljesítmény. Nem kell vendég gépenként egy grafikus kártya. Menthető és visszatölthető a vendégek pillanatnyi állapota, live migration elvileg lehetségesnek tűnik azonos típusú gazda gépek esetén.

Hátrányai: Vendég gépek mozgatása eltérő grafikus kártyájú gazda gépek között jelenleg nem tűnik megoldhatónak. Módosítani kell a GPU-k működését (bonyolultabb

---

hardveresen is tilthatja). AMD vonalon nem találtam konkrét információt, csak annyit, hogy egyes szerverekbe szánt lapkakészletek ismerik a technológiát.

[10] Egyes NVIDIA Quadro FX grafikus kártyák képesek Quadro SLI Multi-OS technológiára, amely az NVIDIA és a Parallels szerint az SLI-be kötött Quadro FX kártyák közül egy kivételével mindet képes átadni a vendég gépeknek, és az átadási beállítások módosíthatók a gazda újraindítása nélkül (de az érintett vendég gépeket le kell állítani).

kontextuskezelés), és a drivereket (a VMM-nek és a vendégeknek). Amennyiben nem tökéletes a kontextusok közötti izoláció, úgy biztonsági kockázatokat jelenthet ez a megoldás.

4 Gyakorlati megvalósítások

Ebben a szakaszban ismertetem, hogy a különböző virtualizációs keretrendszerek milyen módon oldják meg a grafika virtualizációját, mikre képesek, és mit tapasztaltam ezekkel kapcsolatban.

A rendszereket a következő számítógépen próbáltam ki :

- Gazda operációs rendszer : Microsoft Windows Vista SP2 + platform update
- Processzor : Intel Core 2 Duo T9300
- RAM : 2GB DDR2
- Grafikus kártya : NVIDIA Quadro NVS 140M (16 stream processzor, 256MB dedikált RAM)

4.1 VMware Workstation 7.0

A VMware cég terméke, amelynek a legfrissebb verziója a nemrég megjelent 7.0. A kétdimenziós grafikát emulációval valósítja meg, míg a 3D gyorsítást API távoli hívással és API fordítással együtt. A dokumentáció szerint 3D gyorsításhoz a gazda operációs rendszer Windows XP, Windows Vista, Windows 7 vagy Linux lehet, és csak az előbb említett Windowsok lehetnek a vendégek. Windows Vista és Windows 7 vendégek esetén az Aero felület támogatott. A vendégben Direct3D9 API támogatott gyorsítással, Shader Model 3-ig, OpenGL szoftveres képalkotással megoldott. Ezt úgy valósítják meg, hogy az API-t lefordítják a vendégben egy saját szabványos protokollra, amit SVGA3D-nek hívnak, majd ezt átadja a gazda gépnek osztott memória segítségével, végül pedig a gazda lefordítja az elérhető gyorsított API-ra (Windows esetén Direct3D, Linux esetén OpenGL).

A hardveres 3D gyorsítás engedélyezéséhez a következő lépéseket kell megtenni :

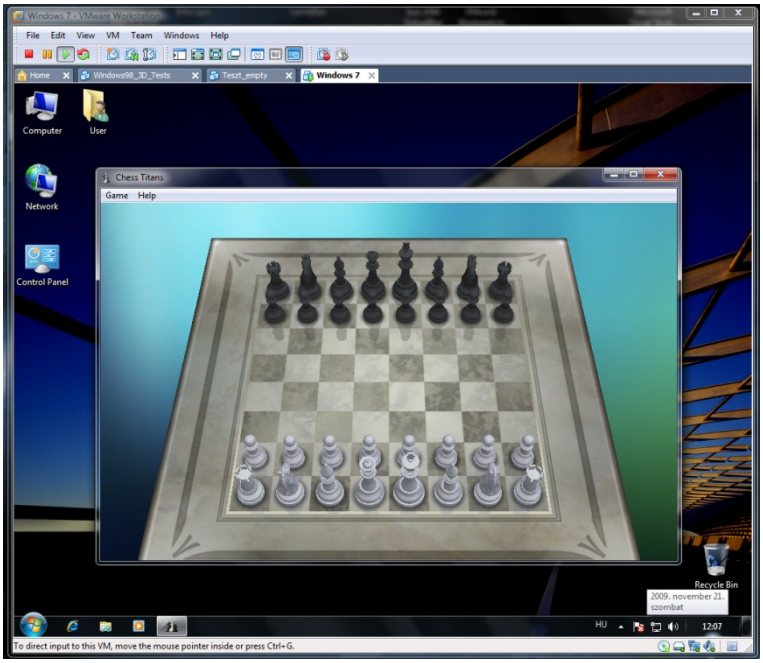
1. Ellenőrizni, hogy a gazda operációs rendszeren van hardveres 3D gyorsítás.
2. A virtuális gép beállításainál, a „Hardware” lapon, a „Display” kiválasztásával jobb oldalt az „Accelerate 3D graphics” opciót be kell pipálni.
3. A vendég gépre fel kell telepíteni a VMware Tools csomagot.

Kipróbáltam a Windows 7 operációs rendszert vendégként, és valóban működik mind az Aero, mind a 3D gyorsítás, ezt szemlélteti a 7. ábra.

Kipróbáltam három játékot Windows XP vendég gépen. Fontos tudni, hogy teljes képernyős játékok használatakor megszűnik a kurzor integráció, manuálisan kell a Ctrl+Alt billentyűkombinációval visszaszerezni, ha a gazda operációs rendszerben szeretnénk valamit csinálni játék közben.

Az első az Oni című (8. ábra), OpenGL-re írt akciójáték, amely nem használ shadereket. A sebessége megfelelő volt, azonban akadtak kompatibilitási problémák, véletlenszerűen hibával kilépett a játék. Ezen felül egyes grafikai elemek nem jelentek meg (az ajtók melletti

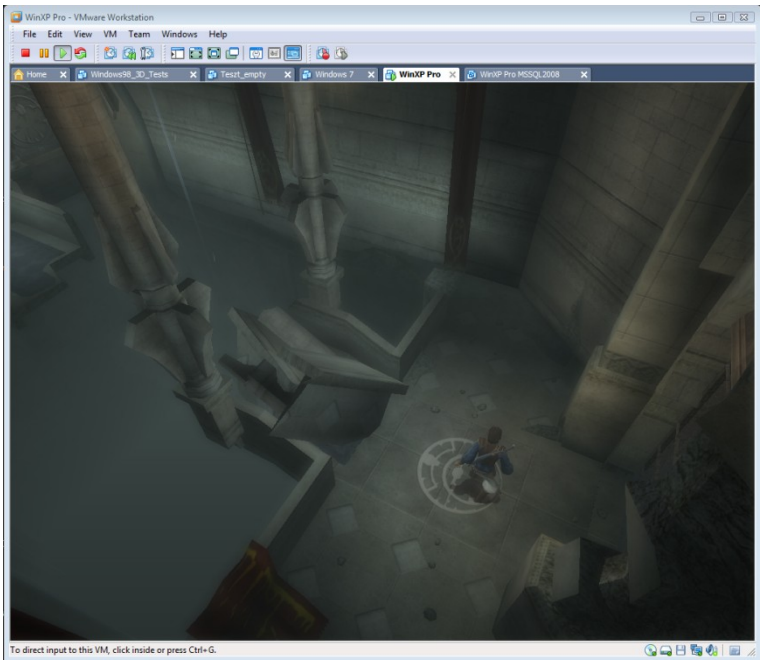
jelzőfény, amely jelzi, hogy az adott ajtó nyitva van-e vagy sem). Sajnos az egér mozgása nem volt egyenletes, ami zavaró, és megnehezíti a játékot.



7. ábra Windows 7 vendég VMware Workstation 7.0-ban, Aero bekapcsolva, és futó Chess Titans 3D sakkjáték

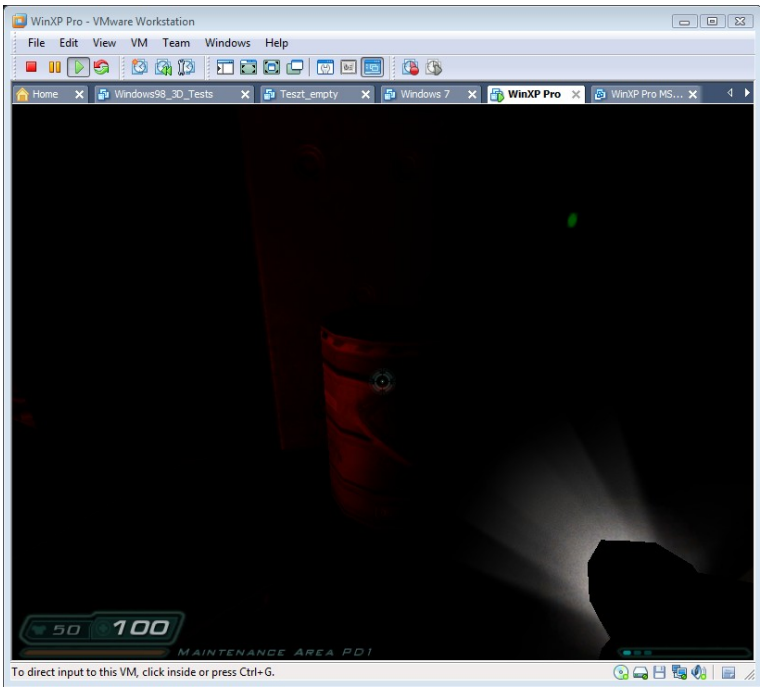
8. ábra Oni VMware-en futó Windows XP vendégben

A második játék a Prince of Persia : The Sands of Time volt (9. ábra). Ezt Direct3D9-re írták, és Shader Model 1-et vagy 2-t használ beállítástól függően (minden grafikai beállítást maximumra kapcsoltam). Itt is megjelent az egér nem egyenletes mozgása, de ezt leszámítva a játékmenet tökéletes volt. Egy másik problémát viszont okozott : a játékból kilépéskor leállt a virtuális gép (mintha „kihúznánk a konnektorból”). Gyanítom, hogy a másolásvédelmi mechanizmus a felelős.



9. ábra Prince of Persia VMware Windows XP vendégben

A harmadik játék a Doom3. Azért választottam, mert OpenGL-re írták, és használ shadereket (Shader Model 2-vel ekvivalens képességűeket). Itt meglepetésemre teljesen egyenletes volt az egér mozgása. Sajnos itt is voltak megjelenítési problémák, méghozzá az elemlámpa nem világította meg a megvilágítandó tárgyakat, ami ebben a sötétben játszódó játékban elég kellemetlen tud lenni (10. ábra).

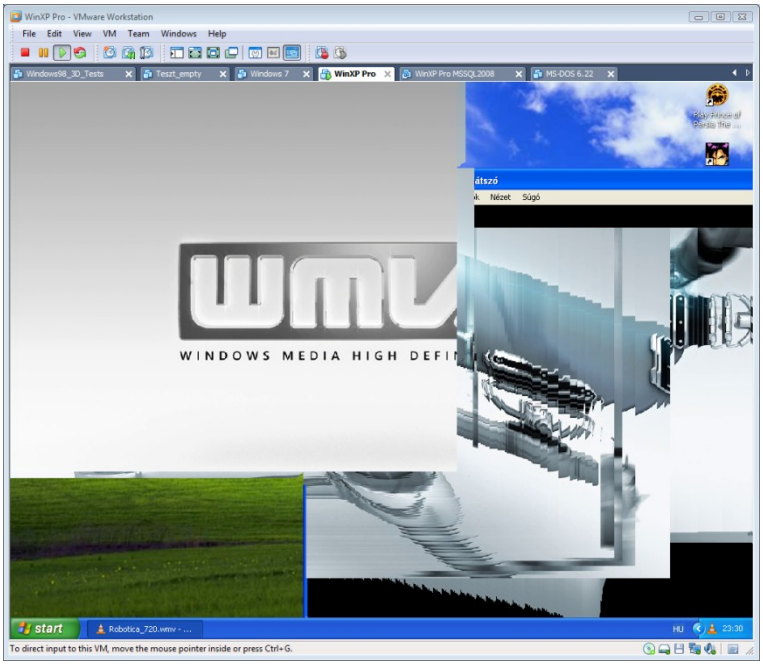


10. ábra Doom3 VMware Windows XP vendégben, látható az elemlámpa hibája

Mindhárom játék alatt igen erős volt a processzor terhelése, amelynek legnagyobb részét a VMware kernel módú folyamata, a *vmware-vmx.exe*, tette ki. Ezeknél sokkal újabb játéknak

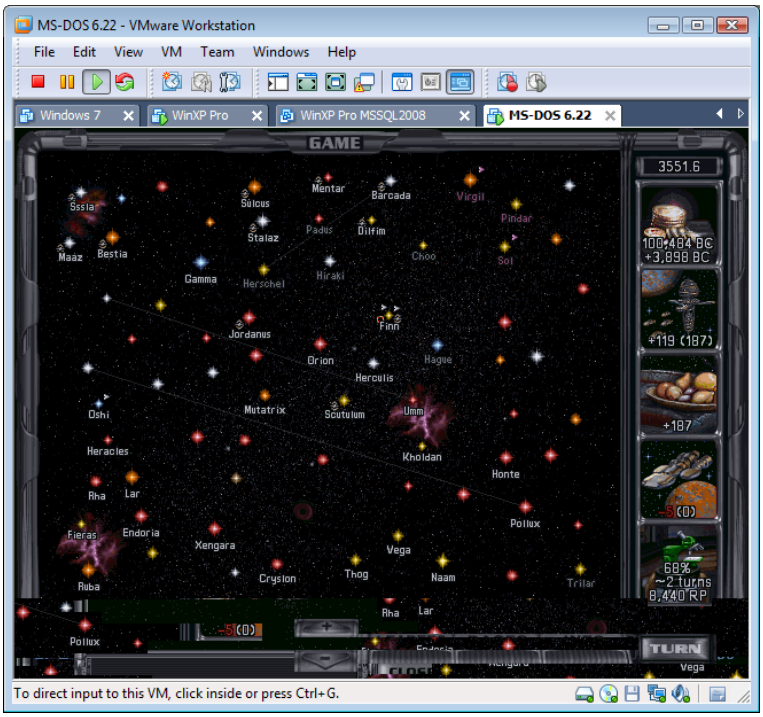
nem láttam értelmét kipróbálni, hiszen azok már a gazda operációs rendszeren futva sem tökéletesen játszhatók.

Kipróbáltam, hogy az overlayt támogatja-e. Igen, támogatja, azzal a megkötéssel, hogy a videó nem érhet ki a vendég gép „képernyőjén” túl. Ekkor ugyanis a videó mása megjelenik a virtuális gép „képernyőjének” bal felső sarkában, és az őt elvileg tartalmazó ablak mozgatásával együtt mozog (de nem azonos irányba), és szemetet hagy maga után a „képernyőn” (11. ábra).



11. ábra VMware Workstation 7.0 overlay kezelési hiba (Robotica című WMV HD bemutató videó)

Feltelepítettem egy MS-DOS 6.22 operációs rendszert is, amelyben szintén három játékot próbáltam ki. Az Alley Cat című játék CGA grafikát használ (mindkét támogatott palettával), és PC speakeren „zenél”. Ez a játék apróbb hanghibáktól eltekintve tökéletesen futott, bár túl gyorsan. A Screamer című játék szoftveresen számolt 3D grafikát használ, amelyet SVGA üzemmóddal jelenít meg, többféle hangkártyát képes használni, köztük Sound Blastereket is. Hang nem volt benne, de a grafika tökéletes volt, de ez a játék is túl gyorsan futott. A harmadik a Master of Orion 2 című űrstratégia, amely 2D grafikát jelenít meg, de magas felbontás és színmélység használatával. Hang itt sem volt, és ráadásul a grafika is hibás volt (egy téglalapnyi terület hibásan jelent meg, és csak abban tudott mozogni a kurzor, ezt nyilvánvalóan a framebuffer eszköz hibás emulálása okozza), ami játszhatatlanná tette (12. ábra, alján látható a hiba).



12. ábra VMware-ben Master of Orion 2, alul képhibával

4.2 Sun xVM VirtualBox 3.1.0

A VirtualBox nevű szoftvert az Innotek nevű cég fejlesztette, amíg a Sun meg nem vásárolta. Képességei nagyon hasonlítanak a VMware Workstationéhez, bár több ponton kevesebbet nyújt. A legfrissebb a 3.1.0 verziójú.

A hardveres OpenGL 2.0 gyorsítás[11] minden támogatott gazda operációs rendszer esetén elérhető (én csak Windows Vistán próbáltam), vendégnek pedig Windows XP vagy újabb Windows, Linux és Solaris a támogatottak. Windows vendég esetén Direct3D 8 és 9 támogatás is van, továbbá a hardveres overlay támogatás is megjelent a 3.1.0 verzióban (szintén csak Windows vendég esetén). Az OpenGL gyorsítást Chromium[12] segítségével valósítja meg, míg a Direct3D-t API fordítással (a WineD3D szoftverkomponens segítségével) még a vendégben OpenGL hívásokká alakítja, ahonnan már azonos módon megy tovább a gazdához. A hardveres overlay is API távoli hívással működik (DirectDrawt továbbít).

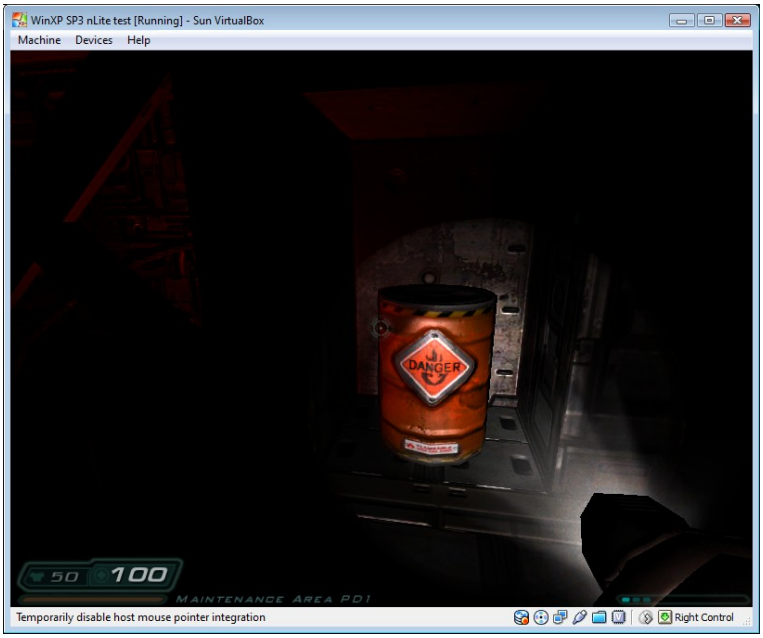
A hardveres 3D támogatáshoz a következő műveleteket kell elvégezni :

1. Meg kell győződni, hogy a gazda operációs rendszeren elérhető a gyorsítás.
2. A virtuális gép beállításainál a „Display” pontot kiválasztva, a jobb oldalon be kell pipálni az „Enable 3D acceleration” opciót (Windows vendég esetén az „Enable 2D video acceleration” opcióval engedélyezhető az overlay gyorsítás).

[11] A VirtualBox fejlesztői mindennemű hardveres 2D és 3D gyorsítást kísérleti stádiumban lévőnek tekintenek.
[12] A Chromium egy olyan szoftveres eszköz, amely OpenGL adatfolyamot képes módosítani és átirányítani, akár hálózaton keresztül is. VirtualBoxban utóbbi képességét használják ki, csak nem hálózaton küldik a gazdának, hanem valamilyen osztott memóriás megoldással (erről nem találtam bővebb információt).

3. A vendégben fel kell telepíteni a Guest Additions csomagot (Windows alatt **csökkentett módban kell**, mert a Windows védelme alatt álló Direct3D DLL-eket felülírja).
4. Játékok esetén manuálisan ki kell kapcsolni az egérkurzor integrációt (jobb klikk az egér ikonra a virtuális gép ablakának jobb alsó sarkában, majd „Disable” opcióra kattintás), különben teljesen rosszul működik.

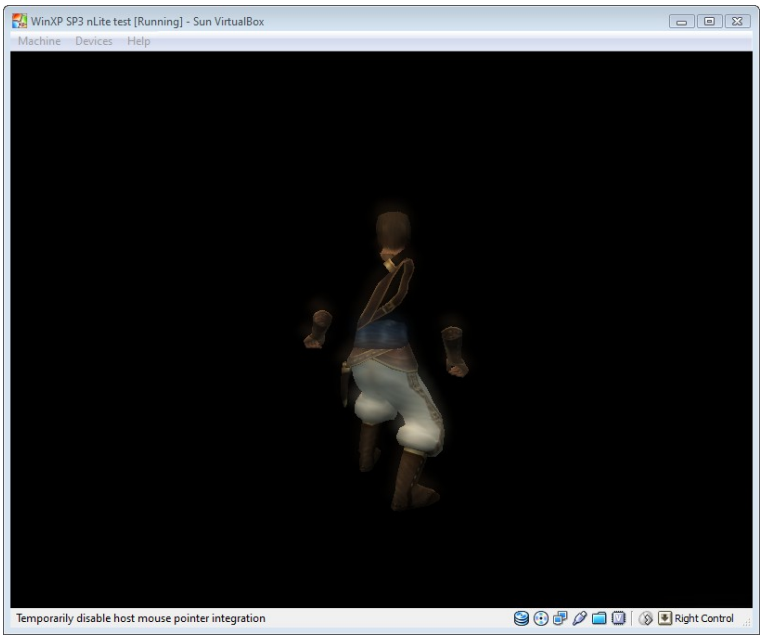
A három kipróbált játék közül a két OpenGL alapú teljesen hibátlanul működött, még az egérmozgások is teljesen egyenletesek voltak (13. ábra, 14. ábra).



13. ábra Doom3 VirtualBoxban futó Windows XP vendégben

14. ábra Oni VirtualBoxban futó Windows XP vendégben

A Direct3D alapú Prince of Persia esetén már a menüben sem jelent meg minden grafikus elem, a kurzor szaggatott, és játék indításakor a virtuális gépet leállította. A következő próbálkozásnál indításkor felkínálta a „safe mode” működést, amit elfogadtam. Ekkor már nem állította le a virtuális gépet, azonban a köd effekt túl erős volt, alig lehetett látni a tárgyakat. A köd effekt kikapcsolása után kiderült, hogy nem ez a gond, hanem az, hogy alig egy-két objektum jelenik meg. Az ezt követő próbálkozáskor le akartam venni a grafikát minimumra, de meglepve tapasztaltam, hogy a menü hibátlanul jelent meg, merészen kipróbáltam a játékot, de ismét leállt tőle a virtuális gép. Végül tényleg kipróbáltam minimumra állított grafikával, de a legjobb eredmény a menü tökéletes megjelenése volt, játékba lépéskor leállt a virtuális gép. Csak „safe mode” esetben sikerült a játékba lépni, ekkor a hercegnek csak egy részhalmaza jelent meg, ez látható a következő ábrán (15. ábra).



15. ábra Prince of Persia VirtualBoxban futó Windows XP vendégben, grafikai hibával

Az overlay ugyanúgy működik, mint a VMware Workstation esetében. Amíg a videó belefér a vendég ablakába, jól jelenik meg, mihelyst le kell vágni belőle, szemetel.

Az MS-DOS operációs rendszerre írt játékok mind grafikai hiba nélkül futottak, az Alley Cat kicsit lassú volt, a Screamer túl gyors, a Master of Orion 2 pedig megfelelő sebességű. A VMware Workstationnel ellentétben PC speaker emuláció nincs VirtualBoxban, viszont van Sound Blaster 16, amely MIDI módban nem működött, és wave kimenetet is csak Screamerben produkált (hangzásra hibátlan volt, de ha kilépéskor hallatszott hang, akkor az ismétlődött a virtuális gép kikapcsolásáig). El is vártam a helyes működést (a hangot nem), hiszen ha a processzor real módban van, akkor a VirtualBox a QEMU felhasználásával emulál.

4.3 Parallels Workstation 4.0 Extreme

Ezt a terméket a Parallels nagyteljesítményű munkaállomásokra szánja. Képességei az elérhető információk alapján nagyjából a VMware Workstationével vetekszenek (beleértve az API távoli híváson alapuló hardveres 3D gyorsítást), azzal a kivétellel, hogy a Parallels

megoldása hivatalosan támogat Intel VT-d technológiát. Grafikus kártyák átadásához több megkötés is van : hivatalosan csak NVIDIA Quadro FX kártyák támogatottak, továbbá a gazda operációs rendszerre olyan drivert kell hozzájuk telepíteni, amely támogatja a Quadro SLI Multi-OS nevű technológiát.

Ebből a termékből tudomásom szerint nem létezik kipróbálható változat, így nem volt lehetőségem tesztelni a képességeit (ezen kívül a megfelelő hardverrel sem rendelkezem).

4.4 Xen és VMware ESXi 4.0

A VMware ESXi 4.0 elvileg nagyjából arra képes, amire a Workstation 7.0, de a Xennel együtt képesek Intel VT-d technológiát használni. Megfelelő hardver hiányában az Intel VT-d technológiát nem tudtam kipróbálni, azonban YouTube-on megtekinthető néhány videó, amelyeken egy felhasználó Xen alatt képes átadni a videokártyáját egy Windows XP vendégnek (link, link, link). A videókból az is kiderül, hogy nem tökéletes még a technológia. A VMware a VMworld 2008-as konferencián demonstrálta az ESX VT-d képességét azzal, hogy egy 10Gbps sebességű hálózati kártyát adott át egy vendég operációs rendszernek, grafikát ők nem demonstráltak. Az AMD azonban a saját IOMMU-ját ugyanezen a konferencián demonstrálta (AMD FirePro grafikus kártya átadásával), szintén ESX 4.0 rendszeren, ez azonban még hivatalosan nem támogatott.

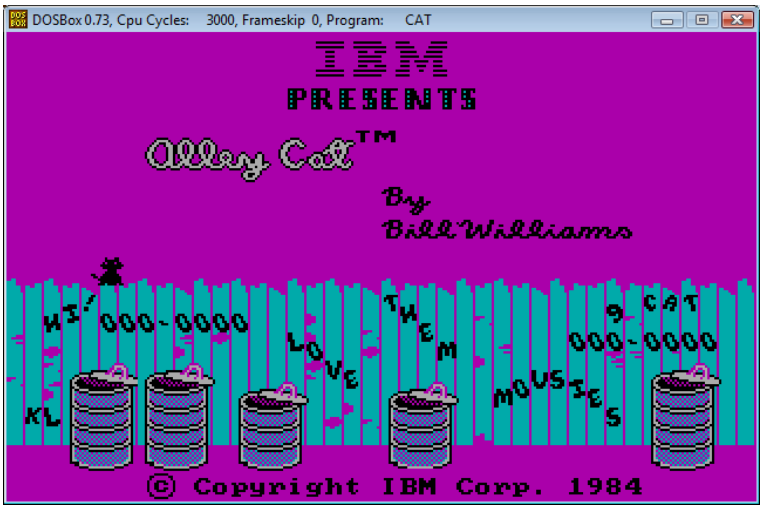
4.5 DOSBox 0.73

A DOSBox kilóg a sorból, ugyanis ezt a szoftvert kifejezetten régebbi, MS-DOS operációs rendszerre írt játékok futtatására fejlesztik. A CPU utasításokat többféle módban képes kezelni (emuláció és Just-In-Time fordítás), továbbá megoldja a többi rendszer túl gyors futtatási hibáját azzal, hogy a végrehajtás sebessége állítható. Az MS-DOS korszak legelterjedtebb hardvereit képes emulálni (Hercules, CGA, EGA, VGA, SVGA videokártyák, több típusú Sound Blaster hangkártya, Gravis Ultrasound hangkártya, IPX protokollú hálózatot emulál tunneling alkalmazásával TCP/IP felett[13], stb…). Hardveres 3D grafikára természetesen nem képes[14]. Továbbá teljes egészében user módban fut, nincs kernel módú komponense.

A három kipróbált játék természetesen hibátlanul futott DOSBoxban, az alábbi képen (16. ábra) az Alley Cat című játék látható.

[13] Így lehetséges IPX hálózatot támogató többjátékos móddal rendelkező játékokat játszani TCP/IP alapú helyi hálózaton, vagy akár interneten keresztül.

[14] MS-DOS operációs rendszerben az egyetlen elérhető 3D gyorsítás a 3Dfx cég Voodoo kártyáival volt lehetséges, amelyek a saját Glide API-t használták. 3Dfx Voodoo kártyák emulálása nincs a DOSBox fejlesztési ütemtervében, de léteznek API fordítók, amelyek Direct3D vagy OpenGL API-ra fordítanak, ezeket „*Glide wrapper*” néven lehet megtalálni.



16. ábra Alley Cat játék DOSBoxban (vegyük észre, hogy IBM termék ☺)

Ezen a linken egy videó található a futásáról : link.

5 Összegzés

Látható, hogy a különböző virtualizációs megoldásokat szállító cégek aktívan foglalkoznak a minél tökéletesebb grafika virtualizáción. Tapasztalataim szerint sok képesség jól működik, sok funkció gondmentesen elérhető virtuális gépekben, azonban még rengeteg továbbfejlesztési lehetőség nyitva áll. Nem minden API támogatott, és a támogatott API-k sem tökéletesek. Vegyük például a GPGPU API-kat. Virtuális gépből nem érhető el GPGPU szolgáltatás, azonban a mai cloud computing divat, és annak GPU-val történő gyorsítása kellő hajtóerőt adhat ezen technológia kifejlesztéséhez.

Úgy gondolom, hogy a grafikus rendszerek virtualizációja az elkövetkezendő években gyorsabb fejlődésnek indul, érdemes figyelni a fejleményeket.

6 Irodalomjegyzék

[1] VMware Workstation 7.0 User's Manual

[2] Sun VirtualBox User Manual

[3] Sun VirtualBox Developer Documentation

[4] DOWTY, MICAH, SUGERMAN, JEREMY : GPU Virtualization on VMware's Hosted I/O Architecture, WIOV 2008

[5] LÁSZLÓ, JÓZSEF : A VGA-kártya programozása Pascal és Assembly nyelven, Computerbooks kiadó, Budapest, 1995