programozói kézikönyv

(a könyv a 2dfx v1.0.1 firmware és a v7-os (TVC: v1) áramköri változat alapján készült)

Tartalomjegyzék

Tartalomjegyzék (folytatás)

Tartalomjegyzék (folytatás)

Köszönetnyilvánítás

A 2dfx létrejöttéhez ezúton is nagyon köszönöm **Persa Noel** ***(geco)*** építő hozzászólásait, javaslatait, aki felhasználói/fejlesztői szemmel kísérte végig a projekt fejlődését az első mozgó kukac képernyőre kerülésétől a 2dfx komplex grafikai rendszerré válásáig.

Ugyanitt köszönöm az **OpenAI 5.5** és **OpenAI 5.6 Sol** modelljeinek ***(CsetPeti)***, hogy az ötletekből és az elképzelt architektúrából működő, a lehetőségekhez képest a végletekig optimalizált kódot gyártott a 2dfx-ben szolgáló mikrokontroller számára.

Az Enterprise és a Videoton TVC tervezőinek, kifejezetten **Nicholas Toop**nak és **David Allen Woodfield**nek köszönöm, hogy annak idején gondos előrelátással megnyitották a lehetőségét annak, hogy a két gép egyszer majd külső színbemeneteket használjon.

Köszönöm **apám**nak, hogy az 1988-as tavaszi BNV-n megvette nekem az Enterprise számítógépemet és **Szabó László**nak, aki közel huszonöt évig őrizte azt a padlásán, hogy aztán 2022-ben visszakerüljön hozzám.

Matusa István ***(Tutus)*** és **Németh Zoltán** ***(ZozoSoft)*** közvetett részvétele a projektben abban nyilvánult meg, hogy az elmúlt negyven évben életben tartották az Enterprise számítógép köré szerveződött klubot, újságot és mind a mai napig a motorjai ennek a kis, ámde lelkes és kitartó közösségnek, köszönöm nektek!

Nélkületek a 2dfx ma nem létezne.

2026. augusztus

Általános információk

FIGYELMEZTETÉS

A 2dfx-et csak saját felelősségre használd.

Igyekeztem úgy megtervezni a hardvert, hogy normál használat mellett ne lehessen vele különösebb bajt okozni, de ettől még egy közvetlenül a számítógép bővítőcsatlakozójára kötött hardverről van szó. Hibás csatlakoztatással, rossz irányban vagy rossz pozícióban bedugott kártyával vagy nem megfelelő más bővítőkártyával együtt használva komoly károkat lehet okozni.

Rossz esetben nemcsak a 2dfx, hanem maga a számítógép, másik bővítőkártya, vagy akár ezek közül több eszköz egyszerre is meghibásodhat. **Az ilyen jellegű károkért nem vállalok felelősséget.**

A legfontosabb szabály: **a 2dfx-et mindig kikapcsolt számítógéphez csatlakoztasd, illetve csak kikapcsolt állapotban húzd le róla.** Bekapcsolás előtt nézd meg még egyszer, hogy a csatlakozó jó irányban és a megfelelő pozícióban van-e.

Más, külső színvezérlést használó bővítőkártyák

A 2dfx az Enterprise, illetve a TVC külső színvezérlésére szolgáló **/EXTC** és **EC0-3** jeleket használja. Ezek a vonalak egyik gépben sem olyanok, mint egy hagyományos háromállapotú adatbusz, ahol több eszköz egyszerűen "elengedheti" a buszt, amikor éppen nincs dolga rajta. Inkább olyan jelekként érdemes rájuk gondolni, mint például a Z80 **/INT**, **/NMI** vagy **/WAIT** bemeneteire: ha több hardver is használni akarja ugyanazt a vonalat, akkor azoknak elektronikai szempontból együttműködésre alkalmas módon kell csatlakozniuk.

A 2dfx erre természetesen fel van készítve hardveresen. Az **/EXTC** és **EC0-3** jelek nem közvetlenül az MCU lábairól mennek a számítógép felé, hanem megfelelő open-collector jellegű meghajtón keresztül, felhúzó- és soros ellenállásokkal. Vagyis a 2dfx önmagában nem próbálja "erőből" meghajtani ezeket a vonalakat egy másik eszközzel szemben.

Ha a számítógépedben van más olyan kártya is, amely az **/EXTC**, **EC0-3** jeleket használja, akkor a két eszköz együttes használata előtt mindenképpen érdemes megnézni annak a kártyának a dokumentációját vagy kapcsolási rajzát, illetve megkérdezni az alkotóját, hogy az adott bővítés fel van-e készítve ezeknek a jeleknek a más hardverrel történő megosztására.

Ha egy másik kártya ezeket a vonalakat közvetlenül, hagyományos push-pull kimenetről hajtja, vagy például 5V-os jel kerülhet közvetlenül olyan alkatrészére, amely erre nem alkalmas, akkor azt a kártyát nem szabad a 2dfx-szel egyidejűleg használni.

Ilyen esetben ugyanis előfordulhat, hogy a két bővítés elektromosan egymás ellen dolgozik. Ennek a vége nem feltétlenül csak az lesz, hogy nem működik: valamelyik kártya akár fizikailag is tönkremehet. Ráadásul attól, hogy két bővítés külön-külön tökéletesen működik, még egyáltalán nem következik, hogy egymás mellett is biztonságosan használhatóak.

Ha tehát nem tudod biztosan, hogy egy másik, külső színvezérlést használó kártya megfelelően kezeli az **/EXTC** és **EC0-3** vonalak megosztását, akkor ne használd a két kártyát egyszerre.

Műszaki háttérinformációk

Az Enterprise és a Videoton TVC látható képét egyaránt dedikált videoáramkörök állítják elő. Közös bennük, hogy a kor néhány más számítógépével – például a Commodore 64-el, az MSX-szel vagy az Atarikkal – ellentétben nem rendelkeznek hardveres sprite-kezeléssel.

A tervezők ugyanakkor nem zárták le végleg ezt a lehetőséget. Mindkét gép bővítőcsatlakozójára kivezették a pixelórajelet, a vízszintes és függőleges szinkronjeleket, valamint kialakítottak öt bemenetet az úgynevezett külső színbemenetek számára.

A külső színbemenetekkel (**`/EXTC`**, **`EC0-3`**) a gép belső videoáramköre arra utasítható, hogy az éppen kirajzolt saját pixel helyett a külső bemeneteken érkező négybites értéknek megfelelő színt jelenítse meg. Ez tizenhat lehetséges színt jelent, feltéve, hogy az ötödik bemenet, a **`/EXTC`** alacsony aktív szinten van. Ennek a megoldásnak az igazi különlegessége az, hogy a külső szín a számítógép aktuális videomódjától és színmódjától függetlenül jelenik meg*. Más szóval: bármilyen üzemmódban dolgozik is a gép, az aktuális elemi pixel színe kívülről felülírható egy tizenhatos színpaletta valamelyik értékével.
** Videoton TVC esetében van egy megkötés (GRAPHICS 2), de erről még a későbbiekben szó lesz.*

Hogy éppen melyik pixel kerül sorra, azt külső hardverrel a szinkronjelek és a pixelórajel alapján pontosan követni lehet. Enterprise esetén egy tévésor 912 pixelórajelből áll, a TVC-nél pedig 800-ból. Ezek természetesen elméleti pixelszámok: a teljes tévésor nem látható, csak annak egy része, a látható képtartomány.

PAL rendszerben egy félkép 312,5 tévésorból áll, ezért a 2dfx számára fontos elméleti felbontás Enterprise esetén **`912×312`**, TVC esetén pedig **`800×312`**. Ezek a számok adják meg azt a koordinátarendszert, amelyben a 2dfx működik.

Mi a 2dfx?

A 2dfx olyan külső grafikus hardver, amely a számítógépből jövő pixelórajel és a szinkronjelek alapján, az **`/EXTC`** és az **`EC0-3`** bemenetek közvetlen vezérlésével képes az aktuális képernyő tetszőleges képpontjának színét megváltoztatni. Működési elve egyszerű: a 2dfx két számlálót vezet. Az egyik az aktuális sort (Y), a másik a soron belüli aktuális pixelt (X) követi. Az X számláló minden vízszintes szinkronimpulzusnál, az Y számláló pedig minden függőleges szinkronimpulzusnál indul újra. Az X számlálót a pixelórajel növeli egyesével, míg az Y számlálót a sorszinkron jel növeli, szintén egyesével. A 2dfx mikrokontrollere (MCU) ezek alapján tudja, hogy a számítógép a kép kirajzolásának melyik pontján jár épp, és ennek megfelelően állítja a külső színbemeneteket.

A 2dfx-ben működő mikrokontrollerben egy komplex szoftver fut, amely lehetővé teszi az Enterprise és a Videoton TVC grafikai képességeinek kibővítését, tekintve, hogy – hasonlóan egy mai GPU-hoz – rengeteg beépített algoritmus segítségével a pixelenkénti számítást a Z80-tól átveszi.

A hardverhez egy egyszerű, jól követhető programozói interfész tartozik. A számítógépen futó felhasználói programnak nem kell minden egyes pixelről külön-külön adatot küldenie minden egyes képkockában; ehelyett magasabb szintű parancsokkal és paraméterekkel írja le, hogy milyen grafikus elemeket szeretne megjeleníteni a képernyőn, amelyeket azután a 2dfx rendererje a megfelelő képpontok külső megváltoztatásával láthatóvá tesz.

A 2dfx ezen felül **8 MB** belső RAM-mal rendelkezik. Ez a felhasználó számára teljes egészében elérhető tárterület a **Resource RAM (RRAM)**. Ide kerülhetnek képadatok, szövegek, sprite-ok, csempék, kitöltőminták és minden olyan adat, amelyre a rajzoláshoz szükség van.

Ez a kézikönyv ennek a működésnek a megértéséhez és a 2dfx felhasználói interfészének teljes körű használatához ad segítséget.

A kézikönyv legfrissebb változatát, illetve a 2dfx-szel kapcsolatos egyebeket (firmware, online editorok, vásárlás) a **`https://2dfx.net`** weboldalon lehet megtalálni.

A 2dfx általános működése és korlátai

A nyolcvanas évekbeli mikroszámítógépek közül azokban, ahol bármilyen hardveres sprite-kezelést valósítottak meg, maga a hardver determinálta a felhasználó számára rendelkezésre álló lehetőségeket, így lett pl. egyidejűleg legfeljebb nyolc, 24×21 pixel méretű sprite kirakása a határ a Commodore 64 esetében.

A 2dfx nem egy hasonló jellegű konkrét hardvertermék, aminek adottak a működési szabályai, vagy amely minden egyes képkocka kirajzolásához – annak tartalmától függetlenül – ugyanazt a szoftvert ugyanannyi idő alatt futtatja le.

A 2dfx minden egyéb hardveres megoldástól eltérően óriási rugalmasságot kínál. A működésének legfőbb szabálya, hogy **bármennyi és bármilyen bonyolultságú grafikát elvégezhet, amit csak a felhasználó megálmodik, de ezeknek a grafikai objektumoknak a kiszámítására egy képfrissítésnyi (egy képkockányi) idő áll a rendelkezésére, ami a mi esetünkben 20 ms**. Ebbe az időkeretbe az összes grafikus megjelenítésnek bele kell férnie ahhoz, hogy szép, folyamatos hatású mozgó sprite-okat, egyéb grafikai elemeket kapjunk. Visszafordítva a rendelkezésre álló időt, **ez 50 fps-es képfrissítést jelent**, azaz a 2dfx-nek másodpercenként ötvenszer teljesen újra kell rajzolnia a grafikai objektumokat, szövegeket, sprite-okat. Ez elsőre meghökkentően soknak tűnhet a TVC-hez vagy az Enterprise-hoz szokott programozó számára, de ezt a sűrű képfrissítést a 2dfx-ben dolgozó mikrokontroller a legtöbb esetben könnyedén megoldja.

Általánosságban elmondható, hogy egy objektumnak minél nagyobb a kiterjedése, annál tovább tart megrajzolni, de a 2dfx-ben lévő sok-sok agyonoptimalizált rajzolási rutin miatt ez sem feltétlenül egyenesen arányos a megjelenített objektumok számával és/vagy méretével. Bizonyos feladatok úgynevezett fast-path-okon keresztül hajtódnak végre, amikben lényegesen kevesebb matematikai számításra van szükség, míg mások, jellegüknél fogva hosszabb számítási igényűek, azaz lassabbak.

Így a 2dfx-ben található legtöbb korlát valójában nem tényleges fizikai korlátot jelent; legfőképp a felhasználói interfész egyszerűsítése volt a cél bizonyos paraméterek értékeinek maximalizálásával. Így például a 2dfx "mindössze" 64 különböző sprite-ot tud kezelni és megjeleníteni egyidejűleg – és ezt az esetek döntő többségében meg is teszi. Ugyanakkor nem garantálható, hogy a 20 ms-os időintervallumban (renderidő) valójában mind a 64 ki is rajzolható akadásmentesen. Ez a korlát megintcsak abból fakad, hogy a sprite-ok vagy egyéb grafikai objektumok mérete és bonyolultsága milyen fokú. Szerencsére az alkalmazott MCU igen gyors, illetve a kód is számtalan optimalizáláson esett túl, így minden szokványos feladat bőven elvégezhető idejében, de adódhatnak különleges helyzetek, amikor ez mégsem tartható.

A 2dfx-en "a puding próbája az evés" elve működik: a felhasználó megálmodhat bármilyen bonyolultságú grafikát, hatalmas sprite-okat, teljes képernyős mozgó háttérképeket, stb. és majd programozás/tesztelés során dől el, hogy azok kirakása a renderidőbe tényleg belefér-e. Ehhez a 2dfx egyébként vizuális segítséget is nyújt: egyrészt egy dedikált kimenetét oszcilloszkópra kötve pontosan mérhető a tényleges ráfordított renderidő, másrészt a 2dfx képes megjelenítés közben figyelmeztetni a programozót a képernyő közepére kirakott hatalmas, villogó felkiáltójel mutatásával arról, hogy a renderelés a 20 ms-on túlcsordult. A közvetlen vizuális figyelmeztetés természetesen szoftveresen be- és kikapcsolható funkció. Renderidő-túlcsorduláskor sincs egyébként semmilyen probléma. A 2dfx nem fagy le, nem indul újra, nem omlik össze, nem rajzol összevissza, egész egyszerűen láthatóan akadozóvá válik a grafikák animálása vagy mozgatása a képernyőn, de csak addig, amíg a renderidő túllépése ténylegesen fennáll.

A könyv végén található Easter Eggek leírásánál feltüntettem a mért renderidőt is az adott eggre vonatkozóan mindkét géptípus esetében, ez adhat némi támpontot a 2dfx határainak feszegetésére irányuló próbálkozásokhoz.

Geometria

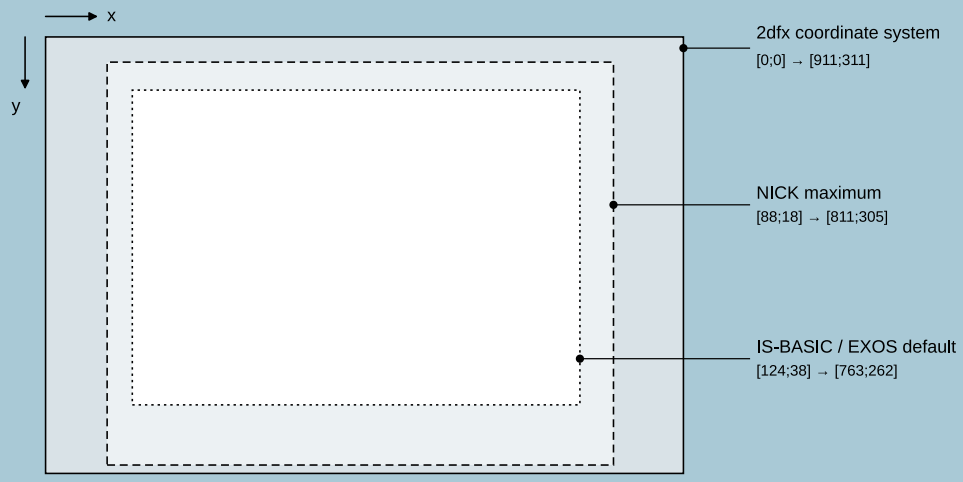
A 2dfx grafikus koordinátarendszerét a korábban említett elméleti pixelszám határozza meg – emlékeztetőül: Enterprise esetén **`912×312`**, TVC esetén pedig **`800×312`**. Mivel ez nem csak a látható képtartományt foglalja magában, létrehozhatunk olyan objektumokat is, amelyek részben vagy teljesen a képernyőn kívül helyezkednek el. Ez különösen hasznos például akkor, ha egy sprite-ot a látható területen kívülről szeretnénk beúsztatni, vagy ha pixelpontosságú szövegscrollt készítünk úgy, hogy a szöveg már a nem látható részen elkezdődik, majd a túloldalon fut ki.

Mindkét gépnél fontos geometriai sajátosság, hogy a pixelek aránya nagyjából **`2:1`** (a TVC-n valamivel még kevesebb). Ha a képernyőn valóban négyzetet szeretnénk látni, akkor annak vízszintes méretét pixelben kb. kétszer akkorára kell venni, mint a függőleges méretét.

Sem az Enterprise, sem a TVC nem ad vissza információt arról, hogy a képernyő éppen milyen felbontásban működik, mekkora a látható szélesség, hány sor látszik, vagy mekkora a border. A programozónak ezért a 2dfx által rajzolt tartalmak pozícióját mindig a gépen beállított képmérettel összhangban kell meghatároznia. A 2dfx a saját, fent leírt koordinátarendszere alapján dolgozik.

Enterprise geometria

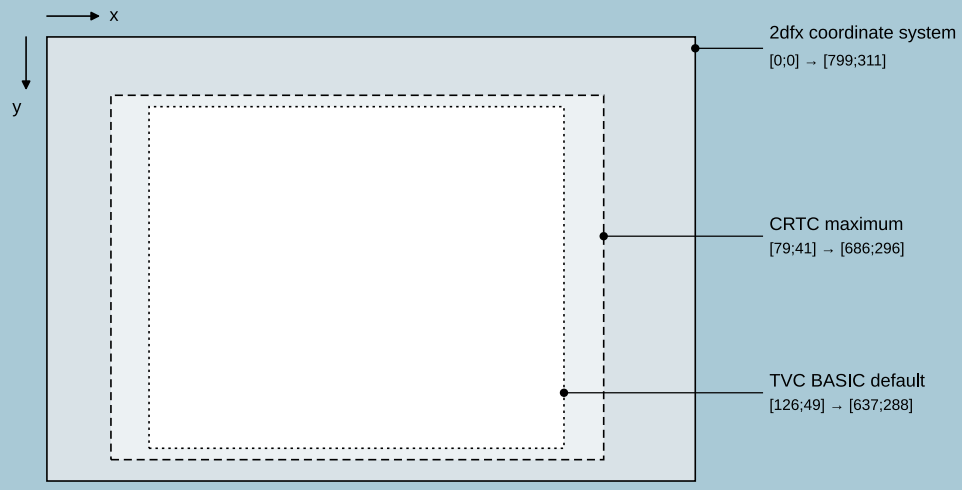
Enterprise esetén az alábbi ábra mutatja a koordinátarendszer gyakorlati felépítését:



Az Enterprise-ban a NICK videochip programozásával állítható be a látható képtartomány mérete és helyzete. A legnagyobb elérhető képernyőterület a fenti ábra szerinti koordináták között jelenik meg. Ha nem módosítjuk a NICK LPT-jét, hanem az EXOS / IS-BASIC alapértelmezett képernyőméretét használjuk, akkor a ténylegesen látható rész ennél kisebb tartományba esik (lásd az ábrán). A képernyőn kívüli terület a border része; ezt a NICK hardveresen nem engedi külső színbemenetekkel felülírni.

Videoton TVC geometria

A Videoton TVC koordinátarendszere:



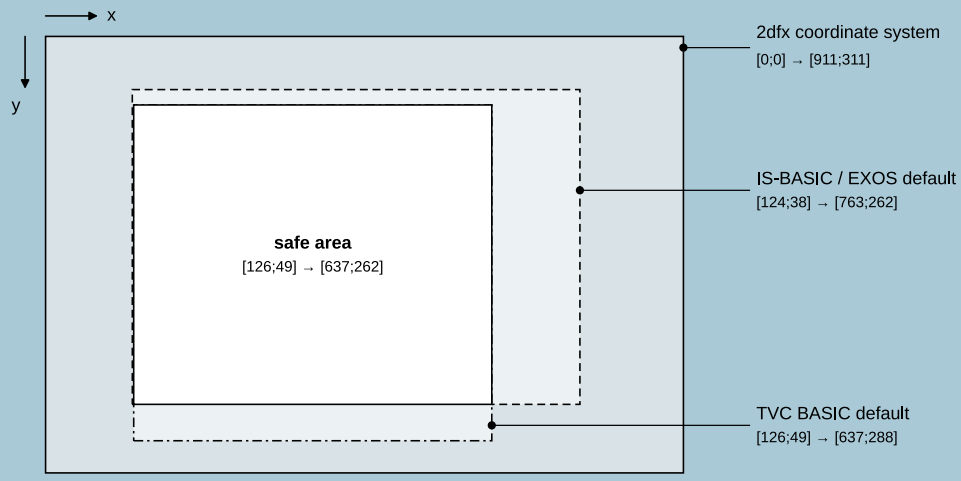
A Videoton TVC-ben a Motorola 6845 CRTC vezérli az alaplapi videoáramkörök időzítését. Ez az áramkör az Enterprise NICK chipjéhez képest egyszerűbb lehetőségeket ad; a TVC legnagyobb, még látható felbontása **608×256** pixel lehet.

TVC esetén különösen fontos, hogy a teljes vízszintes felbontás a gép alaplapi kialakítása miatt csak GRAPHICS 2 üzemmódban érhető el. Bármely más üzemmódban a TVC a külső színbemenetekről érkező elméleti pixelek közül csak minden másodikat veszi figyelembe, majd azt két pixel szélességben jeleníti meg. A 2dfx erről nem kap visszajelzést – a TVC nem szolgáltat ilyen információt –, ezért mindig ugyanabban az ütemben küldi az egymást követő pixeleket. A megjelenítés értelmezése ilyenkor magán a TVC-n múlik. Az Easter Eggek között van egy, ami kifejezetten ennek az alaplapi hardveres korlátozásnak a bemutatására szolgál.

A TVC-nél is igaz, hogy a látható képtartományon kívül eső terület border színű marad. A border ideje alatt a külső színbemenetekkel nem lehet megváltoztatni az ottani pixelek színét.

Keresztfejlesztés

Mivel a két gép alapbeállításai eltérőek, keresztfejlesztéskor hasznos lehet, ha olyan koordinátarendszert választunk, amely mindkét gépen az alaphelyzeti (IS-BASIC vagy TVC BASIC) látható területen belülre esik. Ugyan az előző két ábrából egy pillanat alatt meg lehet mondani azt a biztonságosan rajzolható területet, ahová érdemes a grafikáinkat elhelyezni, a számolgatások elkerülése érdekében itt egy ábra a biztonságos területről (safe-area):



Az is fontos, hogy a keresztfejlesztéskor elegendő egy géphez elkészíteni a 2dfx által kirajzolt grafikai elemeket, sprite-okat, bitmapeket, mert a 2dfx pontosan ugyanúgy fogja azokat kirajzolni az Enterprise-on is, mint a TVC-n és fordítva. A színek beállítására kell igazából odafigyelnünk: TVC-n a fix színpalettát le kell képeznünk az Enterprise oldalán a NICK megfelelő programozásával, ha pl. TVC-ről portolunk 2dfx-es alkalmazást Enterprise-ra.

Figyelem! A két gépen a pixelek aránya némileg eltérő a pixelórajelbeli különbségek miatt. Míg az Enterprise-on egy pixel ~70ns ideig tart, a TVC-nél ez pontosan 80ns. Ez a gyakorlatban azt jelenti, hogy ugyanazon a monitoron nézve egy 2dfx pixel TVC esetében mintegy 14%-kal szélesebb, mint Enterprise-on. Ettől függetlenül ez valószínűleg nem okoz érezhető aránybeli eltérést a két gép képét összehasonlítva.

Színkezelés

Mint már ismeretes, az Enterprise és a TVC esetében is öt bemeneti vonal áll rendelkezésre ahhoz, hogy egy képpont színét megváltoztassuk.

Az **EC0-3** vonalak egy négybites számot határoznak meg; azt a színindexet az aktuális 16-os színpalettából, amit meg kívánunk jeleníteni az adott pixelen.

Az **/EXTC** bemeneti vonal pedig azt vezérli mind az Enterprise, mind a TVC esetén, hogy az **EC0-3** bemeneteken érkező színindex valóban felülírja-e az adott pixelt, magyarán ez az adott pixelre vonatkozóan maga a "transzparencia kapcsoló". 0 esetén a gép az **EC0-3** vonalakról érkező színindexet rakja ki az adott pixelre, 1 esetén pedig a számítógép videoáramkörei által generált saját pixelt rakja ki a képernyőre.*

**Enterprise esetén létezik még ezen felül két másik, egzotikusabb beállítási lehetőség is a NICK képességei révén, de ezzel ebben a könyvben nem foglalkozom, a NICK-kel kapcsolatos leírásokból ez megtanulható; a 2dfx működési elvén nem változtat.*

A fenti öt bemeneti vezeték mindegyikét kell használnunk a vezérléshez – ez a 2dfx esetén pixelenként öt bit tárolását tenné szükségessé. Ugyan a kezdeti kísérletek ebből indultak ki, a tárolás bonyolultsága és az ezzel összefüggő renderelési többlet-időszükséglet miatt a 2dfx végleges verziójában úgynevezett hardveres komparátoros transzparenciavezérlést használok.

A gyakorlatban ez azt jelenti, hogy a 2dfx számára be tudunk (és be is kell) állítani a 16 lehetséges színindex közül egyet, amelyet aztán egy, a 2dfx áramköri paneljére tervezett, a mikrokontrollertől független hardverelem összehasonlít a 2dfx-ből éppen érkező színindexszel. Amennyiben a két érték eltér, a hardveres komparátor azt jelzi a számítógépnek az **/EXTC** bemeneti vonalon, hogy a 2dfx által generált színt jelenítse meg az adott pixelen, ellenkező esetben pedig arra utasítja a számítógépet, hogy az **EC0-3** vonalakon érkező színt hagyja figyelmen kívül, azaz a saját maga által generált pixelt jelenítse meg.

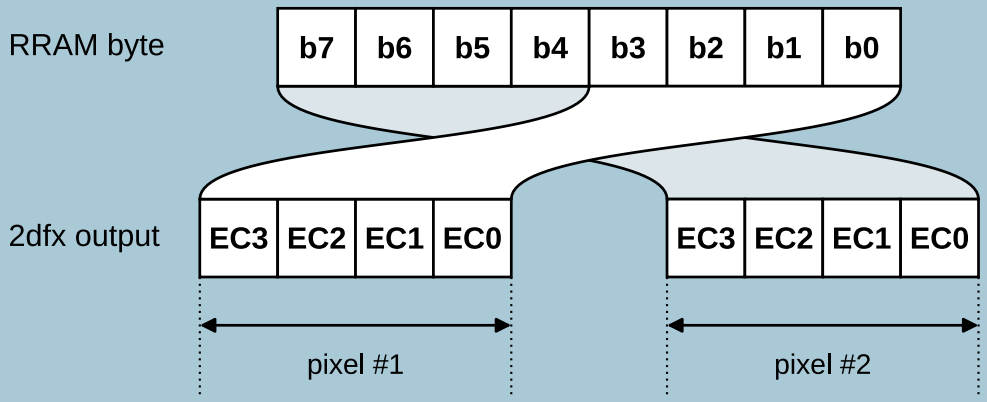
Ez azt vonja maga után, hogy **a 2dfx használatakor egy színt ki kell jelölnünk és ezzel együtt fel is kell áldoznunk azért, hogy a 2dfx transzparencia jelet tudjon biztosítani a számítógép számára.** TVC esetén ez praktikusan lehet a "második fekete" szín, Enterprise-nál pedig például a BIAS színek (8-15) valamelyike. A 2dfx működési profiljától (Enterprise vagy TVC) függetlenül a bekapcsoláskori alapérték a **nullás** színindex, ezt egyetlen parancs segítségével menet közben bármikor meg tudjuk változtatni, bármely időpillanatban.

A 2dfx ugyanakkor rendelkezik egy speciális ENGINE_ON_FORCED üzemmóddal is: ez a hardveres színkomparátort felülbírálja és folyamatosan 0 értéken tartja az **/EXTC** bemenetet. Ebben az üzemmódban le kell mondanunk a számítógépünk által generált grafikákról, szövegkijelzésekről, mivel folyamatosan arra utasítja a 2dfx a számítógépet, hogy a külső színbemenetekről érkező színinformációt jelenítse meg, a sajátját teljesen felülírva.

Tekintve, hogy a 2dfx igyekszik a grafikus feladatok lehető legszélesebb spektrumát lefedni a parancskészletével, valószínűsíthető, hogy a számítógépünk saját grafikai képességére csak minimálisan, vagy akár egyáltalán nem is lesz szükségünk.

A transzparencia megértése után magukról az **EC0-3** vonalakról kötelező beszélnünk azért, hogy a 2dfx-et programozó számára ne okozzon meglepetéseket a színek kezelése.

A 2dfx a saját memóriájában egy bájton két pixel színét tárolja, ami a következőképpen függ össze az **EC0-3** vonalak értékével – erről a Resource RAM használatánál is szó esik majd.



Látható, hogy bármelyik gépet is használjuk, a 2dfx saját RAM-jában két egymást követő pixel színindexe van tárolva és ezek változtatás nélkül kerülnek ütemezetten az **EC0-3** vonalakra. Ez annak érdekében került így kialakításra, hogy a mikrokontroller idejét feleslegesen ne raboljuk azzal, hogy bitátrendezéseket kelljen folyamatosan végeznie, mielőtt egy pixel színét megjeleníti az **EC0-3** kimeneteken. Ezáltal több erőforrás jut a renderelési feladatokra.

A 2dfx belső 4bpp pixelformátumában szándékosan az alsó négy bit tartalmazza a bal oldali, a felső négy bit pedig a jobb oldali pixel színindexét. Ez elsőre fordítottnak tűnhet a megszokott tárolási módokhoz képest, azonban közvetlenül a 2dfx kimeneti interfészének működéséhez igazodik: a framebuffer adatait 32 bites DMA továbbítja a PIO felé, amely jobbra léptetve először mindig az alsó négy bitet küldi ki az EC0-3 vonalakra. Így a framebuffer tartalma minden köztes félbájt átrendezés nélkül, közvetlenül jeleníthető meg, ezzel is javítva a feldolgozási sebességet.

A 2dfx online grafikus editorai eleve ebben a formátumban exportálják a rajzolt képi információkat.

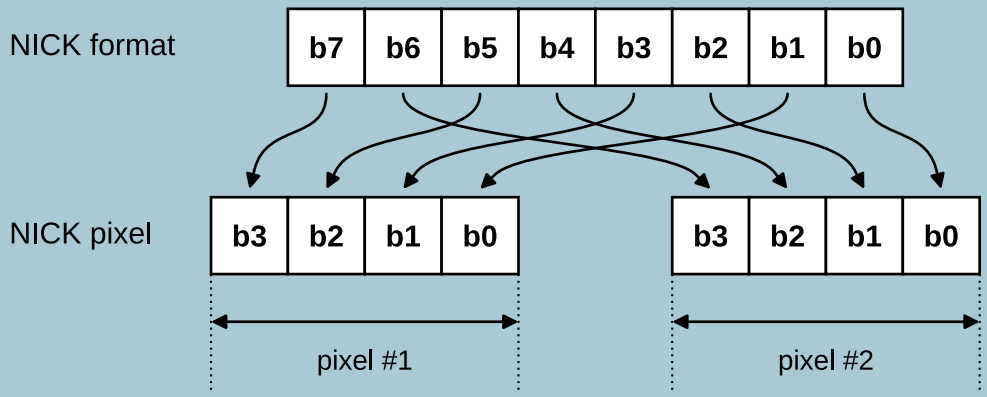
Enterprise színkezelés

Az Enterprise-ban a színek és az **EC0-3** vonalak kapcsolata meglehetősen egyszerű. Az Enterprise aktuális 16 színből álló palettája* egy az egyben meg van feleltetve az **EC0-3** vonalak értékének.

**a paletta alsó nyolc színe (0-7) szabadon definiálható a lehetséges 256 színből – akár soronként is, ha megfelelő LPT-t készítünk –, a paletta felső nyolc színe pedig a NICK BIAS értékének állításával változtatható az egész képre nézve.*

Azaz, ha a palettánk hetedik színére szeretnénk az adott pixelt megváltoztatni, akkor nincs más dolgunk, mint a 2dfx RAM-jában az adott pixel értékét **0111b**-re változtatni.

Ez a tárolási mód bármennyire is egyszerűen követhető, az Enterprise-on felnőtt programozók számára teljesen idegen, ugyanis a NICK sajátosságai miatt két pixel színe a következő módon van reprezentálva a NICK saját video RAM-jában 16 színű mód esetén:

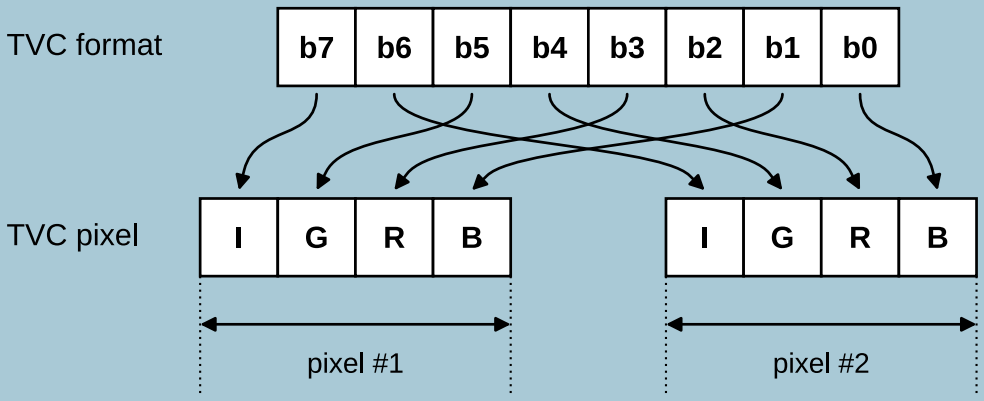


Így tehát a meglévő NICK formátumú sprite adatainkat vagy képeinket át kellene alakítanunk 2dfx belső formátumra annak érdekében, hogy azt a 2dfx korrekt módon meg tudja jeleníteni. Mielőtt a kedves programozó a szívéhez kapna, megnyugtatásul: a 2dfx ezen segít, ugyanis az **UPLOAD_RAW** (ez a 2dfx formátumban való feltöltés parancs neve) parancs mellett ismeri az **UPLOAD_NICK** parancsot is. Utóbbi éppen arra szolgál, hogy az eredetileg NICK formátumban előállított képi anyagot feltölthetővé teszi további felhasználói változtatás nélkül, majd a feltöltést követően a 2dfx maga végzi el a NICK→2dfx formátumkonverziót bájtról bájtra. Ezekről a parancsokról **A Resource RAM (RRAM) felépítése és használata** fejezetben található részletes információ.

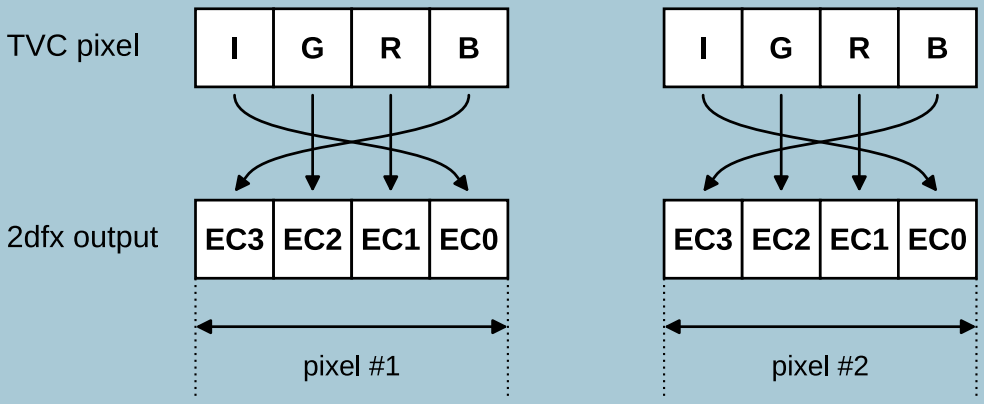
Videoton TVC színkezelés

Ugyan a TVC összesen tizenhat színnel rendelkezik, ráadásul ezek nem is változtathatóak, a tervezők a pixelek színeinek a saját video RAM-ban való tárolását ennél a gépnél is alaposan megvariálták, csakúgy, mint az Enterprise-nál.

16 színű módban az egymás melletti pixelek színeinek tárolása kísértetiesen hasonlít az Enterprise NICK formátumára, azaz:

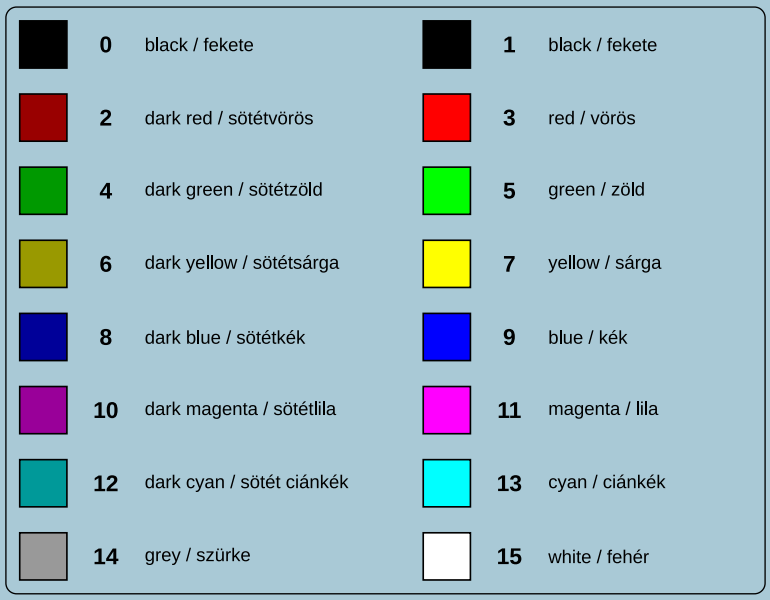


A keveredés itt azonban nem állt meg, mert a tervezők még az **EC0-3** vonalak értékeit is összezagyválták a már eleve rendezetlen video RAM-beli tároláshoz képest, íme:



Látható, hogy teljesen másmilyen módon kerültek kiosztásra az egyes értékekhez tartozó bitek a TVC video RAM-jához képest is, IGRB helyett BGRI-t vár az EC3-0 bemenet.

Ebből a mintázatból viszont fix elrendezés következik (a nem változtatható 16-os paletta miatt), amely alapján a TVC grafikákat az alábbi színtáblázat szerint kell előállítani és feltölteni a 2dfx RRAM-jába a korrekt megjelenítés végett:



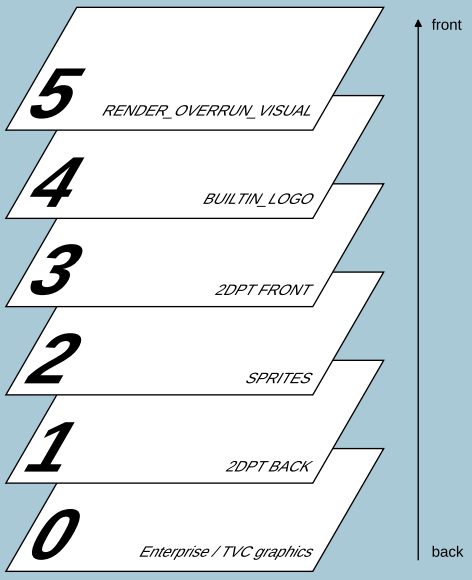
Pánikra itt sincs semmi ok, ugyanis a fenti formátum egyrészt csak az **`UPLOAD_RAW`** parancs használata esetén kötelező a színhelyes képgenerálás érdekében, másrészt pedig a 2dfx beépített **`UPLOAD_TVC`** parancsa a TVC kézikönyvében leírt 16 színű video RAM tárolási módjában elkészült grafikai anyagokat feltölti a 2dfx-be, majd automatikusan elvégzi a feltöltött adatokon a szükséges TVC→2dfx konverziót. Bővebben erről **A Resource RAM (RRAM) felépítése és használata** fejezet ad felvilágosítást a konkrét parancsokkal és azok használatával.

Mindemellett, ahogy már korábban is jeleztem, a 2dfx online grafikus szerkesztőprogramjai alapbeállításkor eleve 2dfx formátumban exportálják az elkészült grafikákat, így azok az **`UPLOAD_RAW`** paranccsal azonnal használhatók (lásd később).

A fenti színtáblázat ismerete ugyanakkor a transzparens szín kiválasztásánál fontos, nem mindegy, melyik színindexünket áldozzuk fel az átláthatóság biztosítására.

Rétegződés

A 2dfx a képet **öt** egymásra épülő rétegként állítja elő. A nagyobb sorszámú réteg mindig magasabb prioritást kap, vagyis az alatta lévő rétegek fölé kerül:



A **nulladik réteg** maga az Enterprise vagy a TVC által előállított kép: szöveg, grafika, border, minden, amit az adott gép önállóan kirajzolna. A 2dfx ezt az alapképet módosítja a külső színbemenetek használatával.

A további rétegek megértéséhez érdemes röviden végignézni, hogyan épült fel a 2dfx grafikus rendszere.

A fejlesztés eredeti célja a hardveres sprite-ok megjelenítése volt úgy, hogy ezzel a Z80 CPU-nak a lehető legkevesebbet kelljen foglalkoznia. A sprite-okat nem az alkalmazásnak kell időigényesen, szoftveresen kirajzolnia; elég a 2dfx megfelelő paramétereit beállítani. A sprite-ok között sorszám szerinti prioritás van: a 0-s sprite van legalul, a 63-as pedig legfelül. Ha két sprite látható, nem transzparens pixelei fedésbe kerülnek, akkor a magasabb sorszámú sprite pixele jelenik meg az adott helyen. A **második réteg** a sprite-oké.

A fejlesztés közben kiderült, hogy a 2dfx-ben dolgozó MCU ennél jóval többre is képes. Nem csak sprite-okat tud kezelni, hanem a gépek teljes elméleti vízszintes és függőleges felbontásában is képes grafikus tartalmat előállítani, akár pixelenként, a rendelkezésre álló tizenhat szín bármelyikével.

Így a sprite-ok mellé bekerült két 2D rajzolási réteg is: a *2DPT back* és a *2DPT front*. Mindkettő egy-egy, legfeljebb 256 elemi grafikai lépésből álló playlist alapján rajzolódik újra minden képkockánál. A 2DPT back a sprite-ok mögé kerül, a 2DPT front pedig a sprite-ok elé. A könyv későbbi fejezetei részletesen bemutatják, milyen rajzolási lehetőségeket adnak ezek a rétegek. A *2DPT back* így az **első réteg**, a *2DPT front* pedig a **harmadik réteg**.

Külön rétegként jelenhet meg a **`SHOW_BUILTIN_LOGO`** is. Ez a beépített 2dfx logót rajzolja ki, illetve tünteti el a képernyő tetszőleges pontján anélkül, hogy a felhasználónak külön logóadatot kellene feltöltenie. Játékok vagy demók menüjében például jól használható egy „powered by 2dfx” jelzéshez. Használata természetesen nem kötelező. A logó fix méretű: **`206×79`** pixel és a **negyedik réteg** egyetlen eleme.

A legmagasabb prioritású, **ötödik réteg** egy fejlesztés közben különösen hasznos visszajelzés. Ha a 2DPT back/front rajzolása és a sprite-ok feldolgozása nem fér bele az adott képkockára rendelkezésre álló időbe – 50 fps módban 20 ms-ba, 25 fps módban 40 ms-ba –, akkor a 2dfx egy folyamatosan változó színű felkiáltójelet jeleníthet meg a **`SET_RENDER_OVERRUN_VISUAL`** parancs bekapcsolása esetén. Ez azt jelzi, hogy a grafikai feladat meghaladja a renderer aktuális időkeretét, ami a gyakorlatban akadozóbb képváltást eredményezhet. A funkció használatáról később részletesebben is lesz szó.

Kommunikáció a 2dfx-szel

A 2dfx Enterprise és Videoton TVC esetén is mindössze két Z80 I/O porton keresztül kommunikál. Ezek mindkét gépen fixen az **F8h (248)** és az **F9h (249)** portok.

A parancsok és paraméterek a használt géptől függetlenül azonosak. Ez megkönnyíti az alkalmazások Enterprise és TVC közötti portolását: a 2dfx által rajzolt grafikus objektumok ugyanazokkal a parancsokkal és ugyanabban a koordinátarendszerben jelennek meg.

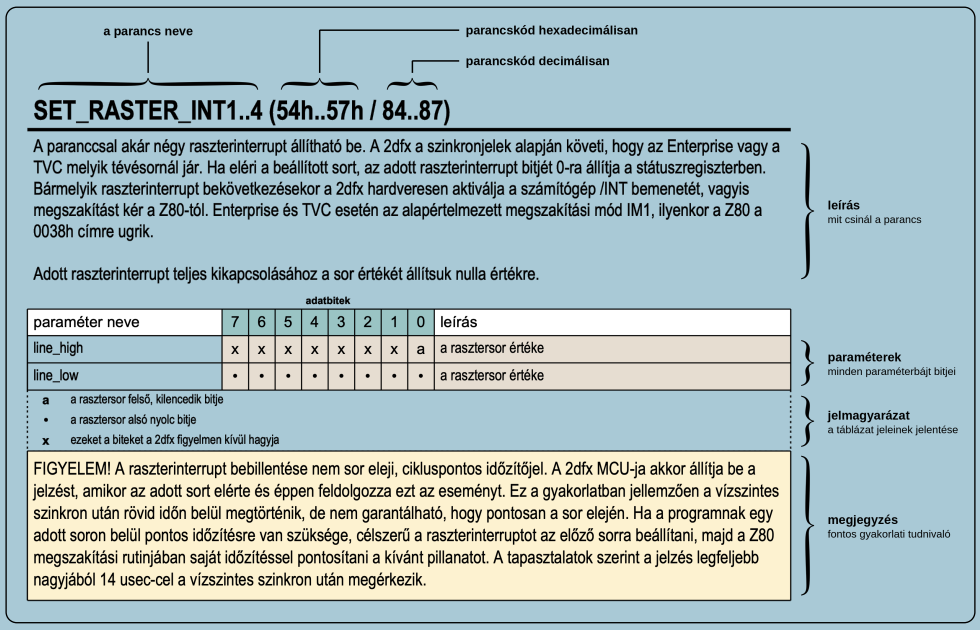
A portolásnál természetesen figyelni kell a két gép eltérő látható képtartományára, a kép kezdőpozíciójára, valamint a színek értelmezésére, erről a **Geometria** és a **Színkezelés** fejezetekben részletesen írtam.

Az **F8h (248)** port egyszerre *parancs- és státuszport*: írható és olvasható. Az **F9h (249)** az *adatport*, amely csak írásra szolgál, ezen keresztül kapja meg a 2dfx az adott parancs paramétereit. Egy háromparaméteres parancs elküldése például így néz ki:

```
OUT 248, aaa          ! aaa = a parancs kódja
OUT 249, bbb          ! bbb = az első paraméter értéke
OUT 249, ccc          ! ccc = a második paraméter értéke
OUT 249, ddd          ! ddd = a harmadik paraméter értéke
```

A sprite- és 2DPT-jellegű grafikai parancsok hatása mindig a következő képkockában jelenik meg. Ez megakadályozza, hogy renderelés közben félkész állapotú objektumparaméterek kerüljenek felhasználásra, és lehetővé teszi, hogy egy képkockányi időn belül – külön szoftveres időzítés nélkül – bármikor előkészítsük a következő képen látható változásokat. A változtatások a legközelebbi függőleges szinkronnál válnak aktívvá.

A továbbiakban a parancsok és paramétereik bemutatására az alábbi táblázatos formát használom:



A státuszregiszter

Az **F8h (248)** port olvasása a 2dfx státuszregiszterének értékét adja vissza. **Az egyes jelzések aktív állapotát a 0 (nulla) érték jelzi!** A regiszter bitjei az alábbi jelentéssel bírnak:

bit	elnevezés	leírás
bit7	COLLISION_DETECTED	Két aktív sprite-nak volt legalább olyan látható pixele az előző képkockában, amely fedésbe került. Az ütközésvizsgálat vízszintesen két, függőlegesen egy pixeles pontossággal történik.
bit6	2DFX_PRESENT	A 2dfx ezt a bitet az inicializálás befejezését követően fixen 0 értéken tartja. Ezzel a felhasználói program ellenőrizheti, hogy a 2dfx kártya jelen van-e a rendszerben. (lásd még: Inicializálás fejezet)
bit5	RENDER_OVERRUN	Aktív, ha a 2dfx nem tudta az adott renderelési feladatot a képkockára rendelkezésre álló időn belül elvégezni. Törölni a CLEAR_RENDER_OVERRUN paranccsal lehet.
bit4	RASTER_INT4	A negyedik raszterinterrupt bekövetkezett.
bit3	RASTER_INT3	A harmadik raszterinterrupt bekövetkezett.
bit2	RASTER_INT2	A második raszterinterrupt bekövetkezett.
bit1	RASTER_INT1	Az első raszterinterrupt bekövetkezett.
bit0	RENDER_TIME	A renderelési idő oszcilloszkópos mérésére szolgál. A 2dfx áramköri paneljén RTIME jelölésű forrasztási pontra is ki van vezetve. Aktív, mialatt a renderer egy képkocka adatait számolja. SYS_RESET parancs kiadása után addig 0 az értéke, amíg a parancs fut, utána 1-be áll.

A státuszport ezen felül a 2dfx áramköri paneljén nyolc zöld LED-re és ugyanennyi forrasztási pontra is ki van vezetve, megkönnyítve ezzel analizátoros vagy oszcilloszkópos mérések végzését.

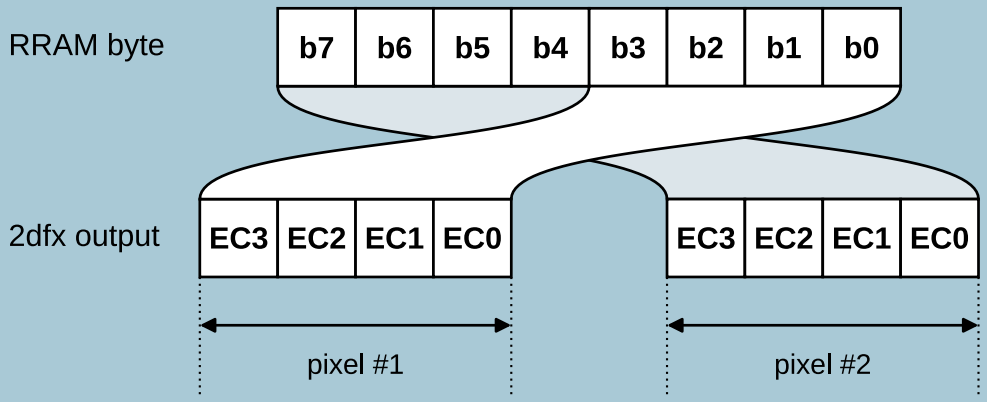
A státuszbitekről részletes tájékoztatás a velük összefüggő parancsok vagy események leírásánál olvasható.

A parancsok részletes ismertetése

A Resource RAM (RRAM) felépítése és használata

A 2dfx kártyán 8 MB RAM áll rendelkezésre a megjelenítéshez szükséges adatok tárolására. Ide kerülhetnek például sprite-bitképek, karakterkészletek, csempék, tilemap-térképek, kiírandó stringek, illetve bármilyen más adat, amelyet a rajzoló alrendszerek felhasználnak. Ezt a memóriaterületet nevezzük **Resource RAM**-nak, röviden **RRAM**-nak.

A hatékony megjelenítés érdekében a 2dfx belső pixelformátuma egy bájtban két egymás melletti pixel színindexét tárolja:



Vagyis az RRAM egy bájtja két szomszédos pixel színindexét tartalmazza: a kimeneti **`EC0-3`** jelek leendő értékét.

A betűkészletek (fontok) tárolása szintén az RRAM-ban történik. Mivel a fontoknál nem pixelenként tároljuk a színt, hanem csak azt, hogy az adott pixel látszik-e, 1bpp formátumot használunk. A 2dfx-szel kiírt FONT-alapú szövegek tehát csak egyféle színt használhatnak. Ez stringenként természetesen szabadon változtatható, de adott karakterképen belül nem.

Az RRAM címtartománya **`000000h-7FFFFFh`**. A teljes terület a felhasználó rendelkezésére áll, és bármely címtől kezdve szabadon tölthetők bele adatok. A címek 23 bitesek, és a 2dfx alrendszerei mindenhol három bájtban várják őket *little-endian* formátumban: először az alsó nyolc bitet, utána a középső nyolc bitet, végül a felső hét bitet. A harmadik bájt legfelső bitje 0 kell legyen.

Ha egy feltöltés túlfutna a **`7FFFFFh`** címen, a 2dfx automatikusan visszaugrik a **`000000h`** címre. A **`7FFFFFh`** után következő bájt tehát már az RRAM elejére kerül, és onnan folytatódik a címszámláló növelése *(address wrap)*.

Az **`UPLOAD_XX`** parancsokkal egy tranzakcióban legfeljebb 65536 bájt, azaz 64 kB tölthető fel. Ez nem korlátozza egy grafikai objektum teljes méretét; csak azt jelenti, hogy a nagyobb adatokat több egymást követő feltöltéssel kell az RRAM-ba juttatni.

A feltöltési parancsok hasonlóan működnek, mint a Z80 esetében az LDIR: amennyiben a feltöltendő bájtok száma **nulla**, úgy a parancs **65536** bájtnyi adatot vár. Ellenkező esetben pedig pontosan annyit, amennyit a kétbájtos hosszban átadtunk.

A 2dfx saját formátumában való feltöltéshez, fontadatok beküldéséhez, minden olyan egyéb adat (pl. string) 2dfx-be juttatásához, amely azt kívánja, hogy az RRAM-ban a feltöltött adatok bájtpontosan tárolódjanak le, az **UPLOAD_RAW** parancsot kell használnunk.

Az Enterprise NICK chipje és a különböző képkonvertálók a pixeladatokat ettől eltérő sorrendben tárolják, ezért a 2dfx-nek ezt a formátumot is kezelnie kell.

Ugyanez igaz a Videoton TVC-re is, ahol a tizenhat színű pixelinformációk tárolása szintén eltér a 2dfx belső formátumától. Ráadásul a TVC alaplapján az **EC0-3** színbemenetek bekötése sem egyezik meg a hagyományos videomódok logikájával: EC0 az I, EC1 az R, EC2 a G, EC3 pedig a B jelre kapcsolódik.

Az **UPLOAD_NICK** és **UPLOAD_TVC** parancsok ezért minden feltöltött adatbájtot a NICK, illetve TVC formátumából a 2dfx belső pixelformátumára alakítanak át feltöltés közben. Az **UPLOAD_RAW** ezzel szemben konverzió nélkül, bájtpontosan tárolja az adatokat az RRAM-ban.

UPLOAD_RAW (80h / 128)

Az RRAM-ba való feltöltésre szolgáló parancs. Paraméterként át kell adni az RRAM kezdőcímét, valamint a feltölteni kívánt bájtok számát, majd pontosan annyi bájtot, amennyi adatot fel kívánunk tölteni. Az RRAM-ba változtatás nélkül helyezi el a feltöltött adatokat.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
RRAM_address_low	•	•	•	•	•	•	•	•	az RRAM kezdőcíme bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	az RRAM kezdőcíme bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	az RRAM kezdőcíme bit22..16
length_low	•	•	•	•	•	•	•	•	a feltölteni kívánt bájtok száma bit7..0
length_high	•	•	•	•	•	•	•	•	a feltölteni kívánt bájtok száma bit15..8
payload_0000h	•	•	•	•	•	•	•	•	az első feltöltött adatbájt
payload_0001h	•	•	•	•	•	•	•	•	a második feltöltött adatbájt
payload_yyyyh	•	•	•	•	•	•	•	•	az yyyyh-ik feltöltött adatbájt

0 kötelezően nulla értékre kell beállítani

FIGYELEM! If the number of bytes to upload (length_low, length_high) is set to zero, the command expects exactly 65536 bytes of payload data.

UPLOAD_NICK (81h / 129)

Az RRAM-ba való feltöltésre szolgáló parancs. Paraméterként át kell adni az RRAM kezdőcímét, valamint a feltölteni kívánt bájtok számát, majd pontosan annyi bájtot, amennyi adatot fel kívánunk tölteni. Az RRAM-ba való beírás előtt elvégzi a NICK-féle bitátrendezést (lásd Színkezelés fejezet).

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
RRAM_address_low	•	•	•	•	•	•	•	•	az RRAM kezdőcíme bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	az RRAM kezdőcíme bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	az RRAM kezdőcíme bit22..16
length_low	•	•	•	•	•	•	•	•	a feltölteni kívánt bájtok száma bit7..0
length_high	•	•	•	•	•	•	•	•	a feltölteni kívánt bájtok száma bit15..8
payload_0000h	•	•	•	•	•	•	•	•	az első feltöltött adatbájt
payload_0001h	•	•	•	•	•	•	•	•	a második feltöltött adatbájt
payload_yyyyh	•	•	•	•	•	•	•	•	az yyyyh-ik feltöltött adatbájt

0 kötelezően nulla értékre kell beállítani

FIGYELEM! If the number of bytes to upload (length_low, length_high) is set to zero, the command expects exactly 65536 bytes of payload data.

UPLOAD_TVC (82h / 130)

Az RRAM-ba való feltöltésre szolgáló parancs. Paraméterként át kell adni az RRAM kezdőcímét, valamint a feltölteni kívánt bájtok számát, majd pontosan annyi bájtot, amennyi adatot fel kívánunk tölteni. Az RRAM-ba való beírás előtt elvégzi a TVC-féle bitátrendezést (lásd Színkezelés fejezet).

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
RRAM_address_low	•	•	•	•	•	•	•	•	az RRAM kezdőcíme bi7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	az RRAM kezdőcíme bi15..8
RRAM_address_high	0	•	•	•	•	•	•	•	az RRAM kezdőcíme bit22..16
length_low	•	•	•	•	•	•	•	•	a feltölteni kívánt bájtok száma bit7..0
length_high	•	•	•	•	•	•	•	•	a feltölteni kívánt bájtok száma bit15..8
payload_0000h	•	•	•	•	•	•	•	•	az első feltöltött adatbájt
payload_0001h	•	•	•	•	•	•	•	•	a második feltöltött adatbájt
payload_yyyyh	•	•	•	•	•	•	•	•	az yyyyh-ik feltöltött adatbájt

0 kötelezően nulla értékre kell beállítani

FIGYELEM! If the number of bytes to upload (length_low, length_high) is set to zero, the command expects exactly 65536 bytes of payload data.

A 2dfx beépített ZX1 kicsomagolóval is rendelkezik. Az **UPLOAD_RAW** paranccsal így előre tömörített ZX1 adatcsomagok tölthetők az RRAM-ba, majd a kívánt célterületre kicsomagolhatók. Az **UNZX1_RAW** bájtpontos kicsomagolást végez, míg az **UNZX1_NICK** és **UNZX1_TVC** a kicsomagolás után a megfelelő bitátrendezést is elvégzi az érintett RRAM-területen.

Például ha egy Enterprise-formátumú grafikai objektumot a fejlesztői gépen ZX1-gyel tömörítünk, a tömörített adatot **UPLOAD_RAW** paranccsal kell feltölteni. Ezután az **UNZX1_NICK** parancs kicsomagolja az adatot, majd elvégzi rajta a NICK→2dfx formátumátalakítást.

A lehetséges feltöltési és kicsomagolási variánsokról a **Feltöltési útmutató** fejezet ad átfogó, táblázatos képet.

ZX1-ben tömörített adat feltöltése után a kicsomagolást mindig külön el kell végezni. A 2dfx grafikai alrendszerei az RRAM adott területét nyers adatként értelmezik; nincs tudomásuk arról, ha ott valójában tömörített adatcsomag található.

Figyelem! UNZX1-kicsomagolás közben az aktuális renderelési munka felfüggesztésre kerül. A képernyőn addig az utolsó elkészült képkocka marad látható, amíg a művelet be nem fejeződik. Emiatt célszerű a nagyobb feltöltéseket/kicsomagolásokat az alkalmazás indításakor, pályák között, demókban pedig jelenetek közötti átmenetnél elvégezni.

UNZX1_RAW (83h / 131)

Az UPLOAD_RAW paranccsal előzőleg RRAM-ba feltöltött, ZX1-el tömörített adatokat csomagolja ki a célcímtől kezdődően. A kicsomagolás közben bitátrendezést nem végez.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
RRAM_src_addr_low	•	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit7..0
RRAM_src_addr_mid	•	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit15..8
RRAM_src_addr_high	0	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit22..16
RRAM_dst_addr_low	•	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit7..0
RRAM_dst_addr_mid	•	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit15..8
RRAM_dst_addr_high	0	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit22..16

0 kötelezően nulla értékre kell beállítani

FIGYELEM! ZX1-compressed data should be uploaded only with UPLOAD_RAW so that its contents are preserved byte-for-byte.

UNZX1_NICK (84h / 132)

Az UPLOAD_RAW paranccsal előzőleg RRAM-ba feltöltött, ZX1-el tömörített adatokat csomagolja ki a célcímtől kezdődően. A kicsomagolás közben NICK-féle bitátrendezést végez az adatbájt eltárolása előtt.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
RRAM_src_addr_low	•	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit7..0
RRAM_src_addr_mid	•	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit15..8
RRAM_src_addr_high	0	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit22..16
RRAM_dst_addr_low	•	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit7..0
RRAM_dst_addr_mid	•	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit15..8
RRAM_dst_addr_high	0	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit22..16

0 kötelezően nulla értékre kell beállítani

FIGYELEM! ZX1-compressed data should be uploaded only with UPLOAD_RAW so that its contents are preserved byte-for-byte. If the original source data (before compression) was in NICK format, use this command for decompression.

UNZX1_TVC (85h / 133)

Az UPLOAD_RAW paranccsal előzőleg RRAM-ba feltöltött, ZX1-el tömörített adatokat csomagolja ki a célcímtől kezdődően. A kicsomagolás közben TVC-féle bitátrendezést végez az adatbájt eltárolása előtt.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
RRAM_src_addr_low	•	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit7..0
RRAM_src_addr_mid	•	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit15..8
RRAM_src_addr_high	0	•	•	•	•	•	•	•	a tömörített adatok RRAM-beli kezdőcíme bit22..16
RRAM_dst_addr_low	•	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit7..0
RRAM_dst_addr_mid	•	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit15..8
RRAM_dst_addr_high	0	•	•	•	•	•	•	•	a kitömörítés RRAM-beli kezdőcíme bit22..16

0 kötelezően nulla értékre kell beállítani

FIGYELEM! ZX1-compressed data should be uploaded only with UPLOAD_RAW so that its contents are preserved byte-for-byte. If the original source data (before compression) was in TVC format, use this command for decompression.

Általános működést befolyásoló globális parancsok

A globális parancsok a 2dfx általános működését befolyásolják. Nem egy-egy sprite-hoz vagy rajzolási objektumhoz tartoznak, hanem a grafikus motor egészére hatnak: beállítják az üzemmódot, a transzparens színt, a renderelési ütemet, illetve a működéshez kapcsolódó jelzéseket.

SYS_RESET (8Fh / 143)

Alaphelyzetbe állítja a 2dfx alkalmazás- és grafikai állapotát, egyfajta soft resetet hajt végre. Törli az RRAM tartalmát, az SPB-ket, mindkét 2DPT-t és a DEFINE_XX resource-definíciókat, valamint alaphelyzetbe állítja többek között a SET_FPS, SET_TRANSPARENT_COLOR és raszterinterrupt-beállításokat. A kiválasztott engine módot (OFF/NORMAL/FORCED) nem változtatja meg. A SYS_RESET alatt a státuszregiszter nulladik bitje 0 értékű és 1-re vált, amikor a SYS_RESET befejeződött. Ezt mindenképpen várjuk ki, mielőtt újabb parancsot adunk ki!

a parancsnak nincs **F9h (249)** paramétere

SET_ENGINE_MODE (50h / 80)

A 2dfx három fő működési módja között választ.

ENGINE_OFF esetén a renderer nem indít új képkockákat, és a 2dfx nem jelenít meg saját grafikai tartalmat, de a kommunikáció továbbra is működik: a hardver fogadja és feldolgozza a parancsokat.
ENGINE_ON a normál működési mód. Ilyenkor a 2dfx csak ott veszi át a pixel színének meghatározását, ahol nem transzparens tartalmat rajzol; transzparens pixel esetén az Enterprise vagy TVC saját képe marad látható.
ENGINE_ON_FORCED módban a 2dfx folyamatosan aktívan tartja az /EXTC jelet, így a gép saját képe nem látszik át: minden pixel színét a 2dfx határozza meg.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
üzemmód	•	•	•	•	•	•	•	•	nyolcbites érték az üzemmód beállítására

- 0 = ENGINE_OFF
- 1…254 = ENGINE_ON
- 255 = ENGINE_ON_FORCED

SET_TRANSPARENT_COLOR (40h..4Fh / 64..79)

Ezzel a paranccsal adható meg azt a színindex, amelyet a 2dfx a továbbiakban átlátszónak tekint. Ha egy sprite vagy más, transzparenciavizsgálatot végző grafikai parancs ilyen színű pixelt tartalmaz, a 2dfx azt nem rajzolja ki, hanem az /EXTC jel inaktív állapotával jelzi a számítógépnek, hogy az adott helyen maradjon a gép saját képe. Sprite-oknál az ütközésvizsgálat is csak a látható, vagyis nem transzparens pixelek között történik. A parancskód utolsó négy bitje eleve meghatározza az átlátszó színindexet, így tehát a parancsnak további paraméter megadása nem szükséges.

a parancsnak nincs **F9h (249)** paramétere
A bekapcsoláskori alapérték a nullás színindex.

SET_FPS (51h / 81)

A 2dfx jelenetfrissítési ütemét állítja. Alapértéke 0, ami 50 fps működést jelent: a 2dfx minden függőleges szinkronnál új képkockát készít, így egy renderelésre 20 ms áll rendelkezésre. Ha egy program, játék vagy demó nem igényel ilyen gyors frissítést, vagy a grafikai feladat túl összetett a 20 ms-os időkerethez, a SET_FPS 1 értékkel 25 fps-re váltható. Ilyenkor egy renderelés legfeljebb 40 ms-ig tarthat, és az elkészült kép két 50 Hz-es videoképkockán keresztül marad látható.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
üzemmód	0	0	0	0	0	0	0	a	az üzemmód beállítására szolgál

a 0 = 50 fps
1 = 25 fps

0 kötelezően nulla értékre kell beállítani

SET_RASTER_INT1..4 (54h..57h / 84..87)

A paranccsal akár négy raszterinterrupt állítható be. A 2dfx a szinkronjelek alapján követi, hogy az Enterprise vagy a TVC melyik tévésornál jár. Ha eléri a beállított sort, az adott raszterinterrupt bitjét 0-ra állítja a státuszregiszterben. Bármelyik raszterinterrupt bekövetkezésekor a 2dfx hardveresen aktiválja a számítógép /INT bemenetét, vagyis megszakítást kér a Z80-tól. Enterprise és TVC esetén az alapértelmezett megszakítási mód IM1, ilyenkor a Z80 a 0038h címre ugrik.

Adott raszterinterrupt teljes kikapcsolásához a sor értékét állítsuk nulla értékre.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
line_high	0	0	0	0	0	0	0	a	a rasztersor értéke
line_low	•	•	•	•	•	•	•	•	a rasztersor értéke

a a rasztersor felső, kilencedik bitje

• a rasztersor alsó nyolc bitje

0 kötelezően nulla értékre kell beállítani

FIGYELEM! Triggering a raster interrupt is not a cycle-exact timing signal at the start of a line. The 2dfx MCU sets the indication when it has reached the configured line and is processing that event. In practice this normally happens shortly after horizontal sync, but it cannot be guaranteed to occur exactly at the beginning of the line. If a program needs precise timing within a particular line, it is advisable to configure the raster interrupt for the previous line and then use timing inside the Z80 interrupt routine to reach the required point. In practice, the indication arrives no later than roughly 14 usec after horizontal sync.

CLEAR_RASTER_INT (58h / 88)

Az összes épp aktív raszterinterrupt kérést törli, a státuszregiszterben mind a négy megszakításbitet 1-re állítja, ezáltal a Z80 /INT vonalát is inaktiválja.

a parancsnak nincs **F9h (249)** paramétere

FIGYELEM! The raster-interrupt indication is sticky. Once triggered, the Z80 /INT input remains active until the user program issues CLEAR_RASTER_INT.

SET_RENDER_OVERRUN_VISUAL (52h / 82)

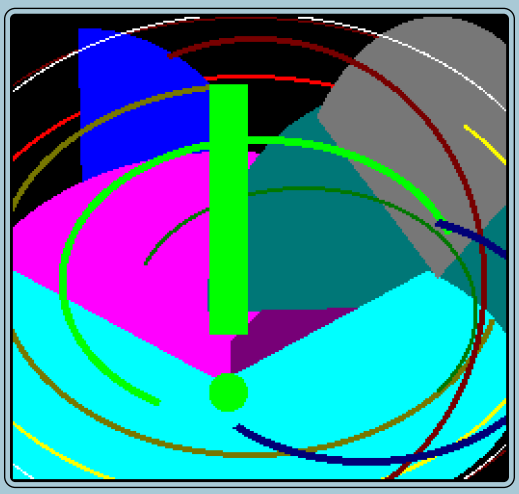
Főleg fejlesztés közben hasznos parancs. A 2dfx folyamatosan méri, hogy mennyi időbe kerül az adott képkocka előállítása. 50 fps módban erre 20 ms, 25 fps módban 40 ms áll rendelkezésre. Ha a renderelés akár csak egyszer túllépi az aktuális időkeretet – például sok nagy sprite és összetett 2DPT back/front rajzolás miatt –, a RENDER_OVERRUN státuszbit 0-ra vált és ott is marad (sticky flag). Ez a bit ugyan a felhasználói programból is kiolvasható a státuszregiszter olvasásával, de fejlesztés közben kényelmesebb lehet bekapcsolni a vizuális jelzést: ilyenkor a 2dfx a legfelső rétegen egy nagy, képkockánként változó színű felkiáltójelet rajzol ki.

Kiadott felhasználói programban a SET_RENDER_OVERRUN_VISUAL bekapcsolása általában nem javasolt. Indításkor a funkció alapértéke 0, vagyis kikapcsolt állapotú.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
üzemmód	0	0	0	0	0	0	0	a	az üzemmód beállítására szolgál

a 0 = kikapcsolva
1 = bekapcsolva

0 kötelezően nulla értékre kell beállítani



CLEAR_RENDER_OVERRUN (53h / 83)

A parancs két dolgot végez el: törli a RENDER_OVERRUN státuszbitet, vagyis visszaállítja 1-re, és ha a vizuális overrun-jelzés be van kapcsolva, eltünteti a felkiáltójelet a képernyőről. A törlés természetesen csak az addig bekövetkezett overrunra vonatkozik. Ha a következő képkocka előállítása ismét túllépi a render budgetet, a státuszbit újra 0-ra vált, és bekapcsolt SET_RENDER_OVERRUN_VISUAL mellett a felkiáltójel is újra megjelenik.

a parancsnak nincs **F9h (249)** paramétere

SHOW_BUILTIN_LOGO (90h / 144)

Egy fix, 206×79 pixel méretű, mérethelyes 2dfx logót rajzol ki a képernyő tetszőleges X/Y koordinátájára. A logó a negyedik rétegen jelenik meg, vagyis a 2DPT back, a sprite-ok és a 2DPT front fölött látszik. Háromféle logikai színt használ: transzparens hátteret, a 2dfx felirat fő színét és az ívelt X-szár kiemelő színét. Enterprise-on a 2dfx nem ismeri a NICK aktuális palettáját, ezért a parancsban meg kell adni, hogy a logó két látható színe melyik színindexet használja. TVC-n fix a 16 színű paletta, de a keresztplatformos parancsforma miatt a színindex bájtot ott is meg kell adni; a TVC-s környezet ezt figyelmen kívül hagyja.

A beépített logó jól használható játékok vagy demók menüjében, ha szeretnénk jelezni, hogy az adott program 2dfx-re készült.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
highbits	x	x	x	x	x	a	a	b	koordináták felső bitjei
X_low	•	•	•	•	•	•	•	•	X koordináta alsó nyolc bit
Y_low	•	•	•	•	•	•	•	•	Y koordináta alsó nyolc bit
color_mapping	c	c	c	c	d	d	d	d	a logó színindexei

a az X koordináta felső két bitje
b az Y koordináta felső bitje
c a 2dfx logo fehér színének színindexe
d a 2dfx logo piros színének színindexe
x ezeket a biteket a 2dfx figyelmen kívül hagyja

HIDE_BUILTIN_LOGO (91h / 145)

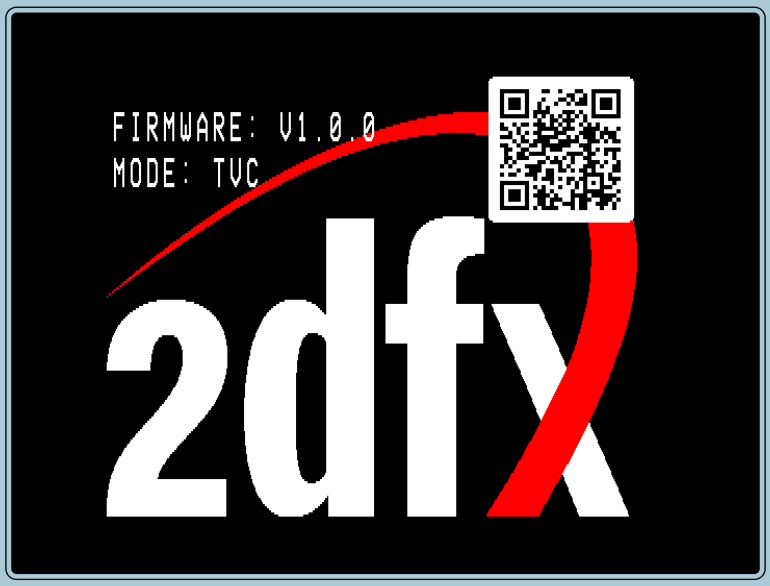
Eltünteti a korábban megjelenített 2dfx logót.

a parancsnak nincs **F9h (249)** paramétere

SHOW_FW (8Eh / 142)

A 2dfx-en futó aktuális firmware verzióját, a működési módot, valamint a firmware-ek letöltőlinkjét mutatja meg a képernyőn. A parancsból való kilépéshez resetelni kell a számítógépet.

a parancsnak nincs **F9h (249)** paramétere



A sprite alrendszer és használata

A 2dfx legalapvetőbb funkciója a hardveres sprite-ok kezelése. A rendszer egyszerre 64 sprite-ot tud megjeleníteni a képernyő látható vagy akár nem látható területén, transzparencia figyelembevételével.

Egy sprite maximális mérete `912×312` pixel lehet.

A sprite-okhoz egyszerű transzformációk is kérhetők renderelés közben: **Xflip** (vízszintes tükrözés), **Yflip** (függőleges tükrözés), **Xdouble** (vízszintes duplázás) és **Ydouble** (függőleges duplázás). Ezek a műveletek egymással szabadon kombinálhatók.

A 2dfx alapszintű ütközésjelzést is ad. Sprite-onként külön beállítható, hogy részt vegyen-e az ütközésvizsgálatban. Az eredmény a renderelés során kerül a státuszregiszter legfelső bitjébe, és a következő elkészült képkockáig stabilan ott marad. A vizsgálat nem befoglaló téglalap alapján működik: ütközés csak akkor történik, ha két, ütközésre engedélyezett sprite legalább egy-egy látható, nem transzparens pixele ténylegesen fedésbe kerül.

A sprite-ok leírói a kilenc bájtos **Sprite Parameter Block**ok, röviden **SPB**-k. A legtöbb 2dfx parancsnál pontosan meg kell adni az összes paramétert, az SPB-írás azonban ettől eltér: elegendő annyi bájtot elküldeni, amennyi a kívánt változtatáshoz minimálisan szükséges. Emiatt az SPB felépítése úgy lett kialakítva, hogy a gyakran módosított adatok – például az RRAM kezdőcíme, az X és Y pozíció – az elejére kerüljenek, a ritkábban változó értékek, például a szélesség és magasság pedig a végére. A módosított SPB-adatok a következő függőleges szinkronnál válnak aktívvá, így elkerülhető, hogy renderelés közben félállapotú paraméterek okozzanak képhibát.

A sprite kiválasztása magával a parancsbájttal történik. A **00h-3Fh**, vagyis **0-63** közötti parancsok a megegyező sorszámú sprite SPB-jének módosítását indítják el.

A sprite vízszintes mérete lehet páratlan pixelszám is. Ilyenkor az RRAM-ba úgy kell feltölteni az adatokat, hogy az utolsó pixelt tartalmazó bájt felső négy bitje már nem tartozik a sprite-hoz; ezt a renderer figyelmen kívül hagyja. Célszerű ide is a transzparens színindexet írni, még akkor is, ha a hardver számára ez az érték már nem számít.

A sprite-adatokat az RRAM-ban sorfolytonosan kell tárolni. A kezdőcím a sprite bal felső sarkának első két pixelét tartalmazó bájtra mutat, majd sorban következnek az első sor további pixelei, utána a második sor, és így tovább. Egy sprite adatigénye: ((W+1) / 2) * H bájt, páratlan szélességnél a sorvégi fél bájtot is figyelembe véve.

A 2dfx önmagában nem animálja a sprite-okat, de az SPB-ben szabadon módosítható a grafikai adat RRAM-beli kezdőcíme. Így ugyanaz a sprite a következő képkockában már másik képkockafázist jeleníthet meg. Ez ugyanaz a logika, amelyet a klasszikus nyolcbites hardveres sprite-rendszereknél is megszoktunk.

A sprite képernyőre úsztatása egyszerűen megoldható: a kirajzolást a látható tartományon kívül kezdjük, majd képkockánként változtatjuk az X és/vagy Y koordinátát. A 2dfx koordinátarendszere miatt a sprite ilyenkor természetesen akkor is létezhet, amikor még nem látszik.

Az SPB-k a 2dfx belső RAM-jában tárolódnak, nem foglalnak helyet az RRAM-ban.

A sprite alrendszer mindössze egyetlen parancsot használ.

ALTER_SPB (00h..3Fh / 0..63)

A parancsbájttal kijelölt SPB módosítására szolgál. Az egyetlen parancs a 2dfx-ben, amely nem követeli meg az összes paraméter mindenkori átadását, a paraméterbájtok a feltöltést követően azonnal beírodnak a 2dfx-ben tárolt SPB leíró megfelelő bájtjára.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
highbits	a	b	c	c	d	e	e	f	vezérlőbitek és az X,Y,W,H paraméterek felső bitjei
X_low	•	•	•	•	•	•	•	•	X koordináta alsó nyolc bit
Y_low	•	•	•	•	•	•	•	•	Y koordináta alsó nyolc bit
RRAM_address_low	•	•	•	•	•	•	•	•	RRAM kezdőcím bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	RRAM kezdőcím bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	RRAM kezdőcím bit22..16
transform_flags	g	h	i	j	0	0	0	0	transzformációs jelzők
W_low	•	•	•	•	•	•	•	•	W szélesség (pixelekben) alsó nyolc bit
H_low	•	•	•	•	•	•	•	•	H magasság (pixelekben) alsó nyolc bit

a SPRITE_ENABLE bit, 1 esetén a sprite megjelenik, 0 esetén nem

b COLLISION_ENABLE bit, 1 esetén a sprite részt vesz az ütközésdetektálásban

c az X koordináta felső két bitje

d az Y koordináta felső bitje

e a W szélesség felső két bitje

f a H magasság felső bitje

g Xdouble bit, 1 esetén a sprite minden pixele megduplázódik a renderelés során vízszintes irányban

h Ydouble bit, 1 esetén a sprite minden pixele megduplázódik a renderelés során függőleges irányban

i Xflip bit, 1 esetén a sprite vízszintesen tükrözésre kerül a renderelés során

j Yflip bit, 1 esetén a sprite függőlegesen tükrözésre kerül a renderelés során

0 kötelezően nulla értékre kell beállítani

A 2DPT bemutatása, működése

A 2dfx a sprite-ok mellett egyszerű és összetett 2D grafikai feladatok elvégzésére is alkalmas. Ezek az elemek megjelenhetnek a sprite-ok mögött (*2DPT back*) vagy a sprite-ok előtt (*2DPT front*).

A **2DPT** a **2D Parameter Table** rövidítése. Mind a *back*, mind a *front* réteghez tartozik egy-egy 256 elemű playlist. A playlist elemei a slotok, vagyis olyan fiókok, amelyekbe a felhasználó egy-egy elemi grafikai parancsot helyezhet. A slotok számozása **0-255** közé esik.

A renderer először a 2DPT back playlistet dolgozza fel a 0-s slottól legfeljebb a 255-ösig. Ezután következnek a sprite-ok, majd ugyanígy a 2DPT front playlist. A 2dfx a 2DPT-k tartalmát eltárolja, ezért nem kell minden képkockánál újra feltölteni a teljes táblázatot; elég csak azokat a slotokat módosítani, amelyek ténylegesen változnak.

A 2dfx a 2DPT-k és a sprite-ok között nem végez ütközésvizsgálatot.

A 2DPT-k segítségével teljes felbontású playfieldek, HUD-elemek, információs sávok és egyéb grafikus rétegek készíthetők az összes lehetséges színnel.

Minden 2DPT parancsnak két változata van: egy a back, egy pedig a front playlisthez. A parancskód legfelső bitje választja ki a célterületet: 0 esetén a 2DPT back, 1 esetén a 2DPT front módosul. A parancsok részletes ismertetésekor az első parancskód a back, a második parancskód a front 2DPT-beli használatot jelzi.

A 2DPT-k a 2dfx belső RAM-jában találhatók. A felhasználói program közvetlenül nem fér hozzájuk, és nem foglalnak helyet az RRAM-ban.

Inicializáláskor a 2DPT-k minden slotja **`RET`** paranccsal van feltöltve.

2DPT általános parancsok

A következő parancsok nem közvetlen rajzoló utasítások; egy adott általános feladatot látnak el és/vagy a 2DPT-ben rákövetkező slotokra vonatkozó általános utasításokat takarják.

2DPT - NOP (60h, E0h / 96, 224)

A NOP parancs az égvilágon semmit nem csinál. A 2DPT lejátszásakor a renderer folytatja a playlist következő slotjával a feldolgozást. Hasznos lehet későbbiekben feltöltött slotok helyének biztosítására.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)

2DPT - RET (7Fh, FFh / 127, 255)

A RET parancs megállítja a 2DPT további feldolgozását, azt jelezve a renderernek, hogy az aktuális 2DPT befejeződött, a további slotokat nem kell figyelembe venni.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)

az utolsó aktív 2DPT parancs után célszerű a RET használata, hogy elkerüljük esetleges régi slotokban lévő feladatok elvégzését

2DPT - SET_COLOR (61h, E1h / 97, 225)

A további 2DPT parancsokhoz beállítja a használni kívánt tinta és papírszín színindexét.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
színindex	a	a	a	a	b	b	b	b	lásd a magyarázatot

a a tintaszín színindexe

b a papírszín színindexe

az alapértéke 0Fh (15) színindex a tintaszínre és a transzparens színindex a papírszínre

2DPT - SET_XY_OFFSET (6Eh, EEh / 110, 238)

A soron következő rajzolási parancsok origóját (X/Y, X0/Y0 és CX/CY), illetve a LINE és az ERASE_LINE X1/Y1 értékeit a parancs paramétereinek megfelelő, előjeles egész számú pixellel eltolja. Ez az ofszet lehet negatív is.

A paranccsal a soron következő összes rajzolási művelet koordinátái eltolhatóak; hasznos lehet pl. többrétegű grafikai objektumcsoport egyidejű mozgatására. A parancs értelmezése a következő SET_XY_OFFSET parancs kiadásáig, ennek hiányában a 2DPT végéig érvényben marad.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
X_offset	•	•	•	•	•	•	•	•	a megadott előjeles egész szám (-128…127) szerinti pixellel azt ezt következő parancsok X koordinátáját eltolja (00h = 0; 7Fh = 127; 80h = -128; FFh = -1)
Y_offset	•	•	•	•	•	•	•	•	a megadott előjeles egész szám (-128…127) szerinti pixellel azt ezt következő parancsok Y koordinátáját eltolja (00h = 0; 7Fh = 127; 80h = -128; FFh = -1)

FIGYELEM! The initial SET_XY_OFFSET value is X=0 and Y=0 at the beginning of 2DPT playlist playback. The command does not use relative values: the parameters of two consecutive SET_XY_OFFSET commands are not added together. Each is always interpreted relative to the absolute X=0 and Y=0 origin, so a later command replaces the earlier offset.

2DPT - SKIP_NEXT (6Fh, EFh / 111, 239)

A paraméterként átadott számú következő 2DPT slotot átugorja, így azok nem vesznek részt a képernyőre rajzolásban. Gyakorlatilag a Z80 JR opcode megfelelője, de csak előre ugorhat.

Amennyiben a paraméterben megadott értékkel a 255-ik slot utánra ugrana, a 2DPT feldolgozását befejezi.

Különösen jól használható olyan grafikai elemeknél, amelyek csak bizonyos időben kell, hogy megjelenjenek, de az utána következő parancsok viszont minden esetben. Ilyenkor célszerű a 2DPT-t "elugratni", majd ha szükség van az ideiglenes parancsokra, akkor mindössze ezt a SKIP_NEXT slotot kell felülírni egy NOP parancsra.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
skip_cmd_no	•	•	•	•	•	•	•	•	a megadott értéknyi következő slot parancsot átugorja

2DPT grafikai primitívek

A grafikai primitívek pontok, vonalak, téglalapok, ellipszisek és ezek különböző változatainak rajzolására szolgálnak. Mindegyik primitív egyetlen 2DPT-slotot foglal el.

A grafikai primitívek listáját és részletes paraméterezését az alábbi táblázatok mutatják be. Minden 2DPT parancsnál pontosan annyi paramétert kell megadni, amennyit az adott parancs leírása előír.

2DPT - PLOT (62h, E2h / 98, 226)

Egy pixelnyi pontot rajzol az aktuális tintaszínnel a paraméterként átadott koordinátapárra.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	x	x	a	a	b	x	x	x	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	az X koordináta alsó nyolc bitje
Y_low	•	•	•	•	•	•	•	•	az Y koordináta alsó nyolc bitje

a az X koordináta felső két bitje
b az Y koordináta felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja

FIGYELEM! A known processing peculiarity of NICK on the Enterprise is that it cannot display a single isolated non-transparent pixel immediately after transparent pixels. ENGINE_ON_FORCED is an exception because in that mode transparency is never actually signalled to NICK.

2DPT - LINE (63h, E3h / 99, 227)

Egy megadott vastagságú vonalat rajzol az aktuális tintaszínnel a paraméterben átadott két koordinátapár között. Ferde vonal esetén a Bresenham-féle vonalalgoritmust használja a képpontok kiszámításához.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	a	a	b	b	c	d	d	e	lásd a magyarázatot
X0_low	•	•	•	•	•	•	•	•	a kezdőpont X koordinátájának alsó nyolc bitje
Y0_low	•	•	•	•	•	•	•	•	a kezdőpont Y koordinátájának alsó nyolc bitje
X1_low	•	•	•	•	•	•	•	•	a végpont X koordinátájának alsó nyolc bitje
Y1_low	•	•	•	•	•	•	•	•	a végpont Y koordinátájának alsó nyolc bitje

a a kívánt vonalvastagság - 1
b a kezdőpont X koordinátájának felső két bitje
c a kezdőpont Y koordinátájának felső bitje
d a végpont X koordinátájának felső két bitje
e a végpont Y koordinátájának felső bitje

2DPT - ERASE_LINE (64h, E4h / 100, 228)

Egy megadott vastagságú vonalat rajzol az aktuális papírszínnel a paraméterben átadott két koordinátapár között. Ferde vonal esetén a Bresenham-féle vonalalgoritmust használja a képpontok kiszámításához.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	a	a	b	b	c	d	d	e	lásd a magyarázatot
X0_low	•	•	•	•	•	•	•	•	a kezdőpont X koordinátájának alsó nyolc bitje
Y0_low	•	•	•	•	•	•	•	•	a kezdőpont Y koordinátájának alsó nyolc bitje
X1_low	•	•	•	•	•	•	•	•	a végpont X koordinátájának alsó nyolc bitje
Y1_low	•	•	•	•	•	•	•	•	a végpont Y koordinátájának alsó nyolc bitje

a a kívánt vonalvastagság - 1
b a kezdőpont X koordinátájának felső két bitje
c a kezdőpont Y koordinátájának felső bitje
d a végpont X koordinátájának felső két bitje
e a végpont Y koordinátájának felső bitje

2DPT - BUCKET_FILL (65h, E5h / 101, 229)

Az átadott X/Y koordinátán található színindexből kiindulva az összefüggő, azonos színű területet az aktuális tintaszínnel tölti ki. A kitöltés azokon a kapcsolódó pixeleken terjed tovább, amelyek színindexe megegyezik a kiindulási X/Y koordinátán talált színindexszel; ettől eltérő színű pixeleknél a kitöltés megáll. Tipikus felhasználása például egy zárt alakzat belsejének kiszínezése. A használata nagyobb vagy bonyolultabb területek esetében meglehetősen időigényes lehet, ezért téglalapok, lekerekített téglalapok vagy ellipszisek kitöltéséhez inkább a megfelelő dedikált FILL_XX parancsokat érdemes használni, amelyek kifejezetten gyors működésre vannak optimalizálva.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	x	x	a	a	b	x	x	x	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a kitöltés mintavételezési pontja X koordinátájának alsó nyolc bitje
Y_low	•	•	•	•	•	•	•	•	a kitöltés mintavételezési pontja Y koordinátájának alsó nyolc bitje

a a kitöltés mintavételezési pontja X koordinátájának felső két bitje
b a kitöltés mintavételezési pontja Y koordinátájának felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja

FIGYELEM! BUCKET_FILL uses an internal, finite work stack to keep track of areas that still need to be processed. Its size is deliberately limited, so the fill operation can never cause a stack overflow or another memory error in the MCU. On a highly fragmented fill area with many branches or regions connected by narrow passages, however, this internal work stack may become full. In that case the operation still terminates safely, but part of the area may remain unfilled. If this happens, simply issue BUCKET_FILL again at an X/Y coordinate inside one of the remaining unfilled regions.

2DPT - RECT (66h, E6h / 102, 230)

Téglalapot rajzol a megadott vonalvastagsággal. A téglalap körvonalai a SET_COLOR-ban meghatározott tintaszínnel rajzolódnak ki. Abban az esetben, ha a SET_COLOR-ban megadott papírszín nem az átlátszó szín, úgy a papírszínnel a téglalap belsejét kitölti. Átlátszó színindexű papírszín esetén a téglalapnak csak a körvonalát rajzolja meg.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	a	a	b	b	c	d	d	e	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a bal felső sarok X koordinátájának alsó nyolc bitje
Y_low	•	•	•	•	•	•	•	•	a bal felső sarok Y koordinátájának alsó nyolc bitje
W_low	•	•	•	•	•	•	•	•	a szélesség (W) értékének alsó nyolc bitje
H_low	•	•	•	•	•	•	•	•	a magasság (H) értékének alsó nyolc bitje

a a kívánt vonalvastagság - 1
b a bal felső sarok X koordinátájának felső két bitje
c a bal felső sarok Y koordinátájának felső bitje
d a szélesség (W) pixelei számának felső két bitje
e a magasság (H) pixelei számának felső bitje

a vonalvastagság figyelembe veszi a 2:1-es pixelarányt, így a függőleges oldalak vonalvastagsága kétszerese a vízszintesekének

2DPT - FILL_RECT (67h, E7h / 103, 231)

Téglalapot rajzol az aktuális tintaszínnel, amit utána a tintaszínnel ki is tölt.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	x	x	a	a	b	c	c	d	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a bal felső sarok X koordinátájának alsó nyolc bitje
Y_low	•	•	•	•	•	•	•	•	a bal felső sarok Y koordinátájának alsó nyolc bitje
W_low	•	•	•	•	•	•	•	•	a szélesség (W) értékének alsó nyolc bitje
H_low	•	•	•	•	•	•	•	•	a magasság (H) értékének alsó nyolc bitje

a a bal felső sarok X koordinátájának felső két bitje
b a bal felső sarok Y koordinátájának felső bitje
c a szélesség (W) pixelei számának felső két bitje
d a magasság (H) pixelei számának felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja

2DPT - ROUND_RECT (68h, E8h / 104, 232)

Lekerekített sarkú téglalapot rajzol a megadott vonalvastagsággal. A téglalap vonalai a korábban SET_COLOR parancsban meghatározott tintaszínnel rajzolódnak ki. Abban az esetben, ha a SET_COLOR-ban megadott papírszín nem az átlátszó szín, úgy a papírszínnel a téglalap belsejét kitölti. Átlátszó színindexű papírszín esetén a téglalapnak csak a körvonalát rajzolja meg. A sarkok egy-egy negyed ellipszisként jelennek meg, az ellipszis RX és RY rádiuszainak megfelelően.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	a	a	b	b	c	d	d	e	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a bal felső sarok X koordinátájának alsó nyolc bitje
Y_low	•	•	•	•	•	•	•	•	a bal felső sarok Y koordinátájának alsó nyolc bitje
W_low	•	•	•	•	•	•	•	•	a szélesség (W) értékének alsó nyolc bitje
H_low	•	•	•	•	•	•	•	•	a magasság (H) értékének alsó nyolc bitje
X_radius	•	•	•	•	•	•	•	•	a vízszintes sugár (RX) értéke
Y_radius	•	•	•	•	•	•	•	•	a függőleges sugár (RY) értéke

a a kívánt vonalvastagság - 1
b a bal felső sarok X koordinátájának felső két bitje
c a bal felső sarok Y koordinátájának felső bitje
d a szélesség (W) pixelei számának felső két bitje
e a magasság (H) pixelei számának felső bitje

2DPT - FILL_ROUND_RECT (69h, E9h / 105, 233)

Lekerekített sarkú téglalapot rajzol az aktuális tintaszínnel, amit utána a tintaszínnel ki is tölt. A sarkok egy-egy negyed ellipszisként jelennek meg, az ellipszis RX és RY rádiuszainak megfelelően.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	x	x	a	a	b	c	c	d	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a bal felső sarok X koordinátájának alsó nyolc bitje
Y_low	•	•	•	•	•	•	•	•	a bal felső sarok Y koordinátájának alsó nyolc bitje
W_low	•	•	•	•	•	•	•	•	a szélesség (W) értékének alsó nyolc bitje
H_low	•	•	•	•	•	•	•	•	a magasság (H) értékének alsó nyolc bitje
X_radius	•	•	•	•	•	•	•	•	a vízszintes sugár (RX) értéke
Y_radius	•	•	•	•	•	•	•	•	a függőleges sugár (RY) értéke

a a bal felső sarok X koordinátájának felső két bitje
b a bal felső sarok Y koordinátájának felső bitje
c a szélesség (W) pixelei számának felső két bitje
d a magasság (H) pixelei számának felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja

2DPT - ELLIPSE (6Ah, EAh / 106, 234)

Ellipszist rajzol a megadott vonalvastagsággal. Az ellipszis körvonala a SET_COLOR-ban meghatározott tintaszínnel rajzolódik. Abban az esetben, ha a SET_COLOR-ban megadott papírszín nem az átlátszó szín, úgy a papírszínnel az ellipszis belsejét kitölti. Átlátszó színindexű papírszín esetén az ellipszisnek csak a körvonalát rajzolja meg. Az átadandó paraméterek az ellipszis origója (CX/CY) és a vízszintes (RX), valamint a függőleges (RY) rádiusza.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	a	a	b	b	c	d	d	e	lásd a magyarázatot
CX_low	•	•	•	•	•	•	•	•	a középpont X koordinátájának alsó nyolc bitje
CY_low	•	•	•	•	•	•	•	•	a középpont Y koordinátájának alsó nyolc bitje
X_radius	•	•	•	•	•	•	•	•	a vízszintes sugár (RX) alsó nyolc bitje
Y_radius	•	•	•	•	•	•	•	•	a függőleges sugár (RY) alsó nyolc bitje

a a kívánt vonalvastagság - 1
b a középpont X koordinátájának felső két bitje
c a középpont Y koordinátájának felső bitje
d a vízszintes sugár (RX) értékének felső két bitje
e a függőleges sugár (RY) értékének felső bitje

2DPT - FILL_ELLIPSE (6Bh, EBh / 107, 235)

Ellipszist rajzol az aktuális tintaszínnel, amit utána a tintaszínnel ki is tölt. Az átadandó paraméterek az ellipszis origója (CX/CY) és a vízszintes (RX), valamint a függőleges (RY) rádiusza.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	x	x	a	a	b	c	c	d	lásd a magyarázatot
CX_low	•	•	•	•	•	•	•	•	a középpont X koordinátájának alsó nyolc bitje
CY_low	•	•	•	•	•	•	•	•	a középpont Y koordinátájának alsó nyolc bitje
X_radius	•	•	•	•	•	•	•	•	a vízszintes sugár alsó nyolc bitje
Y_radius	•	•	•	•	•	•	•	•	a függőleges sugár alsó nyolc bitje

a a középpont X koordinátájának felső két bitje
b a középpont Y koordinátájának felső bitje
c a vízszintes sugár (RX) értékének felső két bitje
d a függőleges sugár (RY) értékének felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja

2DPT - ARC (6Ch, ECh / 108, 236)

A megadott vonalvastagsággal ívet rajzol az aktuális tintaszínnel. Az átadandó paraméterek hasonlóak az ellipszishez: középpont (X/Y), vízszintes (RX) rádiusz, függőleges (RY) rádiusz, ezek egészülnek ki a start_angle és end_angle szögparaméterekkel.

A szögparaméterek egy bájtban egyenlő mértékben osztják fel a 360°-nyi szöget. A kiemelt szögek beosztása: 00h (0) – 0°, 12 óra; 40h (64) – 90°, 3 óra; 80h (128) – 180°, 6 óra; C0h (192) – 270°, 9 óra. A kiosztás miatt az FFh (255) nem egyezik meg a 0°-kal.

Azonos szögparaméterek esetén nem rajzol ívet.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	a	a	b	b	c	d	d	e	lásd a magyarázatot
CX_low	•	•	•	•	•	•	•	•	a középpont X koordinátájának alsó nyolc bitje
CY_low	•	•	•	•	•	•	•	•	a középpont Y koordinátájának alsó nyolc bitje
X_radius	•	•	•	•	•	•	•	•	a vízszintes sugár alsó nyolc bitje
Y_radius	•	•	•	•	•	•	•	•	a függőleges sugár alsó nyolc bitje
start_angle	•	•	•	•	•	•	•	•	kezdőszög értéke
end_angle	•	•	•	•	•	•	•	•	végszög értéke

a a kívánt vonalvastagság - 1
b a középpont X koordinátájának felső két bitje
c a középpont Y koordinátájának felső bitje
d a vízszintes sugár (RX) értékének felső két bitje
e a függőleges sugár (RY) értékének felső bitje

FIGYELEM! ARC is one of the most computationally expensive primitives. In some cases it is worth replacing it with a pre-drawn shape or bitmap uploaded to RRAM.

2DPT - PIE (6Dh, EDh / 109, 237)

Kördiagramot rajzol az aktuális tintaszínnel. Az átadandó paraméterek hasonlóak az ellipsziséhez: középpont (X/Y), vízszintes (RX) rádiusz, függőleges (RY) rádiusz, ezek egészülnek ki a start_angle és end_angle szögparaméterekkel. A szögparaméterekhez tartozó kezdő- és végpontokat összeköti az origóval, így hozva létre a kördiagramot, amit aztán a tintaszínnel ki is tölt.

A szögparaméterek egy bájtban egyenlő mértékben osztják el a szögeket. A kiemelt szögek beosztása: 00h (0) – 0°, 12 óra; 40h (64) – 90°, 3 óra; 80h (128) – 180°, 6 óra; C0h (192) – 270°, 9 óra. A kiosztás miatt az FFh (255) nem egyezik meg a 0°-kal.

Azonos szögparaméterek esetén nem rajzol.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	x	x	a	a	b	c	c	d	lásd a magyarázatot
CX_low	•	•	•	•	•	•	•	•	a középpont X koordinátájának alsó nyolc bitje
CY_low	•	•	•	•	•	•	•	•	a középpont Y koordinátájának alsó nyolc bitje
X_radius	•	•	•	•	•	•	•	•	a vízszintes sugár alsó nyolc bitje
Y_radius	•	•	•	•	•	•	•	•	a függőleges sugár alsó nyolc bitje
start_angle	•	•	•	•	•	•	•	•	kezdőszög értéke
end_angle	•	•	•	•	•	•	•	•	végszög értéke

a a középpont X koordinátájának felső két bitje
b a középpont Y koordinátájának felső bitje
c a vízszintes sugár (RX) értékének felső két bitje
d a függőleges sugár (RY) értékének felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja

FIGYELEM! PIE is one of the most computationally expensive primitives. In some cases it is worth replacing it with a pre-drawn shape or bitmap uploaded to RRAM.

2DPT resource-alapú parancsok

A 2DPT nemcsak közvetlenül rajzolt primitíveket támogat, hanem úgynevezett resource-alapú parancsokat is.

Ezek a parancsok összetettebb feladatokat végeznek, és abból indulnak ki, hogy a szükséges adatok már korábban bekerültek az RRAM-ba. Elsőre ez kicsit közvetettnek tűnhet, de a használat logikája egyszerű: először feltöltjük és definiáljuk az erőforrást, később pedig a 2DPT parancs már csak hivatkozik rá.

A 2dfx négyféle resource-alapú 2DPT parancsot támogat. Ezeket a következő alfejezetek mutatják be.

2DPT BLIT – bitmapek, háttérképek gyors kirakása

A **BLIT** egy előre definiált, téglalap alakú képrész gyors kirajzolására szolgál. Az RRAM megadott kezdőcímétől W szélességben és H magasságban tárolt képadatot másolja a képernyő megadott X/Y pozíciójára. Működése sokban hasonlít a sprite-hoz, de a sebesség érdekében egyszerűbb szabályokat követ: csak egész bájtokat, azaz páros számú vízszintes pixelt kezel, és a cél X koordinátát párosra igazítja, figyelembe véve a **SET_XY_OFFSET** parancsban beállított eltolást is. Emellett opcionálisan végez transzparenciavizsgálatot: vagy minden pixelt átmásol, azt is, amelyik a transzparens színindexet tartalmazza, vagy a sprite-okhoz hasonlóan végez transzparenciavizsgálatot és annak megfelelően másolja a bitmap-et. Természetesen az utóbbi egy lassabb folyamat az előzőhöz képest.

A **BLIT** legkézenfekvőbb felhasználása egy előre elkészített, RRAM-ba feltöltött kép vagy háttérelem gyors megjelenítése.

A **BLIT** használatához először a szokásos UPLOAD parancsokkal fel kell tölteni a képadatot az RRAM-ba. Ezután a **DEFINE_BITMAP** rendel hozzá egy logikai azonosítót, valamint megadja a bitmap kezdőcímét, szélességét és magasságát. A későbbi 2DPT **BLIT** parancs már erre az azonosítóra hivatkozik, és csak azt kell megadnia, hogy a bitmap hova kerüljön a képernyőn. Ha a **BLIT** páratlan X koordinátát kap, azt automatikusan párosra igazítja, vagyis az alsó bitet figyelmen kívül hagyja.

A **BLIT** parancs a 2DPT bármely slotjába beilleszthető. Egyszerre legfeljebb 256 bitmap definiálható, de menet közben bármelyik újrahasznosítható.

DEFINE_BITMAP (5Ch / 92)

Az RRAM-ba feltöltött bitmaphez egy logikai azonosítót rendel. A 2DPT BLIT parancsa ezt az azonosítót használja a bitmap RRAM-beli kezdőcímének, szélességének és magasságának meghatározásához. Egyszerre maximum 256 bitmap definíció létezhet, de bármelyikük újrafelhasználható működés közben.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
bitmap_id	•	•	•	•	•	•	•	•	a bitmap kívánt azonosítószáma (0..255)
RRAM_address_low	•	•	•	•	•	•	•	•	RRAM kezdőcím bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	RRAM kezdőcím bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	RRAM kezdőcím bit22..16
highbits	x	x	x	x	x	a	a	b	lásd a magyarázatot
W_low	•	•	•	•	•	•	•	x	a bitmap szélessége (W) alsó nyolc bit – az utolsó bitet a 2dfx ignorálja
H_low	•	•	•	•	•	•	•	•	a bitmap magassága (H) alsó nyolc bit

a a szélesség (W) pixelei számának felső két bitje
b a magasság (H) pixelei számának felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja
0 kötelezően nulla értékre kell beállítani

2DPT - BLIT (71h, F1h / 113, 241)

A DEFINE_BITMAP paranccsal létrejött azonosítójú, RRAM-ba előzőleg feltöltött bitmapet másolja a képernyő megadott (X/Y) koordinátájára. A BLIT transzparenciavizsgálatot csak opcionálisan végez, a pixeleket az eltárolt színindexek alapján másolja és írja felül a célterület pixeleit transzparenciavizsgálat kikapcsolt állapota esetén. Amennyiben a transzparenciavizsgálat be van kapcsolva, úgy a sprite-okhoz hasonlóan végzi a pixelek kirajzolását. Ebben az esetben a másolás némileg lassabb.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	a	x	b	b	c	x	x	x	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a bitmap bal felső sarkának X koordinátája a képernyőn; alsó nyolc bit
Y_low	•	•	•	•	•	•	•	•	a bitmap bal felső sarkának Y koordinátája a képernyőn; alsó nyolc bit
bitmap_id	•	•	•	•	•	•	•	•	a bitmap azonosítószáma (0..255) a DEFINE_BITMAP által összerendelve

a transzparenciavizsgálat bekapcsolása 1 esetén
b a bal felső sarok X koordinátájának felső két bitje
c a bal felső sarok Y koordinátájának felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja
0 kötelezően nulla értékre kell beállítani

FIGYELEM! For speed, BLIT copies only an even number of pixels and performs transparency testing only when explicitly requested. Otherwise every pixel of the bitmap is copied unchanged into its bounding rectangle.

2DPT FONT – szövegek, stringek kiírása

A 2dfx tetszőleges szöveg megjelenítésére is használható. A karakterek az aktuálisan beállított előtérszínnel, vagyis a **SET_COLOR** által meghatározott tintaszínnel rajzolódnak ki.

A 2dfx nem tartalmaz beépített karakterkészletet, de legfeljebb 64 különböző font definiálható. Egy font 128 karakteralakzatból állhat. A karakterek 1bpp mintázatát először az RRAM-ba kell feltölteni, a megfelelő UPLOAD paranccsal.

A fontok 1bpp bitmapjénél a legfelső bit a bal szélső pixelt, a legalsó bit a jobb szélső pixelt jelenti.

A karaktercellák 8×8 pixeles alapegységekből épülnek fel, és ennek vízszintes, illetve függőleges többszörösei is lehetnek. Fontos, hogy ez nem automatikus nagyítást jelent: a nagyobb cellaméret több ténylegesen tárolt pixelt igényel karakterenként. Így egy karakter akár 128×128 pixeles is lehet, ha a fontdefiníció ezt írja le.

Az RRAM-ban a fontadatok a nulladik karakter definíciójával kezdődnek, majd sorban következik az első, a második, egészen a 127-ik karakterig. Egy 8×8-as font teljes helyigénye 128 * 8 bájt, vagyis 1024 bájt, míg egy 64×32-es fonté 128 * 8 * 32 bájt, azaz 32768 bájt.

A **DEFINE_FONT** resource parancs az RRAM-ba feltöltött fontadatot rendeli hozzá egy logikai fontazonosítóhoz, és ugyanitt adható meg a karaktercellák mérete is. A parancsot célszerű közvetlenül a feltöltés után kiadni, így a későbbiekben a **DRAW_TEXT** már egyszerűen hivatkozhat a kész karakterkészletre.

Ahhoz, hogy szöveget írjunk ki, magát a karakterláncot is fel kell tölteni az RRAM-ba, mégpedig **UPLOAD_RAW** paranccsal. Egy string hossza maximum 255 karakter lehet, ennél rövidebb string esetén egy **80h (128)** értékű lezáró bájttal kell jelezni a string végét. A ténylegesen megjeleníthető karakterek a **00h-7Fh (0-127)** tartományba esnek. A string feltöltésére nincs külön parancs, és fontos, hogy RAW adatként kerüljön az RRAM-ba, vagyis a 2dfx ne rendezze át a bitjeit.

A **DRAW_TEXT** parancs bármely 2DPT-slotba beilleszthető. Paraméterként megkapja a használni kívánt font azonosítóját, az első karakter képernyőpozícióját, valamint a kiírandó string RRAM-beli kezdőcímét.

A **DRAW_TEXT** bájtról bájtra olvassa a stringet az RRAM-ból, és addig rajzolja a karaktereket, amíg 80h értékű lezáró bájtot nem talál, vagy el nem éri a 255 maximális karaktert.

DEFINE_FONT (5Bh / 91)

Az RRAM-ba feltöltött font bitmapet rendeli össze egy logikai fontazonosítóval. Egyszerre maximum 64 fontazonosító létezhet, de bármelyikük újradefiniálható működés közben. Az összerendelés mellett meghatározza a fontban egy karakterkép legnagyobb méretét (max. 128×128 pixel). Egy font 128 karaktert definiálhat.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
font_id	0	0	•	•	•	•	•	•	a font kívánt azonosítószáma (0…63)
RRAM_address_low	•	•	•	•	•	•	•	•	RRAM kezdőcím bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	RRAM kezdőcím bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	RRAM kezdőcím bit22..16
width_height_units	a	a	a	a	b	b	b	b	lásd a magyarázatot

a karakterszélesség nyolc pixeles egységekben – 1 (0 = 8 pixel, 1 = 16 pixel, stb.)
b karaktermagasság nyolc pixeles egységekben – 1 (0 = 8 pixel, 1 = 16 pixel, stb.)
0 kötelezően nulla értékre kell beállítani

2DPT - DRAW_TEXT (70h, F0h / 112, 240)

A fontazonosítóval megjelölt fonttal az RRAM-ba betöltött címtől kezdődő szöveget írja ki az adott (X/Y) koordinátától a képernyőre. A betűk kirakásához az aktuális tintaszínt használja. Amennyiben az aktuális papírszín nem a transzparens szín, úgy a karakterek mögötti területet a papírszínnel tölti fel. Az RRAM-ban tárolt szöveget az első 80h (128) értékű lezáró bájtig olvassa és írja ki, ennek hiányában 255 karakterig.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	x	x	a	a	b	x	x	x	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a bal felső sarok X koordinátájának alsó nyolc bitje
Y_low	•	•	•	•	•	•	•	•	a bal felső sarok Y koordinátájának alsó nyolc bitje
font_id	0	0	•	•	•	•	•	•	a kiválasztott font azonosítószáma
RRAM_address_low	•	•	•	•	•	•	•	•	a letárolt szöveg RRAM kezdőcíme bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	a letárolt szöveg RRAM kezdőcíme bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	a letárolt szöveg RRAM kezdőcíme bit22..16

a a kezdőpont X koordinátájának felső két bitje
b a kezdőpont Y koordinátájának felső bitje
x ezeket a biteket a 2dfx figyelmen kívül hagyja
0 kötelezően nulla értékre kell beállítani

2DPT PATTERN_FILL – téglalapok mintázatos kitöltése

A 2dfx segítségével téglalap alakú területeket ismétlődő mintázattal is kitölthetünk. Egy kitöltőminta (pattern) fixen 16×8 pixeles, 4bpp formátumú képelem, ezért pontosan 64 bájtot foglal. Több pattern egymás után, folyamatosan is elhelyezhető az RRAM-ban; ezek együtt egy mintatáblát alkotnak.

A használat első lépése tehát a minták képi adatainak feltöltése az RRAM-ba. Ezután a **DEFINE_PATTERN** resource paranccsal meg kell adni az adott mintatábla kezdőcímét, és hozzá kell rendelni egy table_id azonosítót. Egyszerre legfeljebb 64 különböző table_id definiálható.

Egy pattern tábla legfeljebb 256 egymást követő mintát tartalmazhat. A **PATTERN_FILL** parancsban ezért két dolgot választunk ki: először azt, hogy melyik table_id alatt definiált táblát használjuk, majd ezen belül a pattern_number értékkel a 0..255 közötti konkrét mintát. A kiválasztott pattern címe így a tábla kezdőcíméből és a mintázat sorszámából adódik; a programozónak nem kell minden egyes mintához külön resource definíciót létrehoznia.

A **PATTERN_FILL** a 2DPT back vagy front bármely slotjába kerülhet. A parancs a megadott X/Y koordinátától kezdődő, W szélességű és H magasságú téglalapot tölti ki a kiválasztott 16×8-as minta ismétlésével. A mintázat vízszintesen és függőlegesen is automatikusan ismétlődik mindaddig, amíg a teljes megadott területet ki nem tölti.

A kitöltendő téglalap méretének nem kell a 16×8-as minta egész számú többszörösének lennie. Ha a legutolsó ismétlés túlnyúlna a megadott W×H területen, a 2dfx csak a téglalapon belülre eső pixeleket rajzolja ki. A framebuffer szélén történő clipping sem változtatja meg a pattern fázisát: a mintázat továbbra is az eredetileg megadott X/Y kezdőponthoz igazodik.

A **PATTERN_FILL** nem használ transzparencia-vizsgálatot: a kiválasztott pattern minden pixelét kirajzolja.

DEFINE_PATTERN (5Dh / 93)

Az RRAM-ba feltöltött fix 16×8-as kitöltőmintákhoz egy logikai azonosítót rendel. Egy azonosító legfeljebb 256 darab különböző, az RRAM-ban egymást követő 16×8-as kitöltőmintát tartalmazhat. Egyszerre legfeljebb 64 pattern definíció létezhet, de bármelyikük újrafelhasználható működés közben.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
table_id	0	0	•	•	•	•	•	•	a kitöltőminta-tábla azonosítószáma
RRAM_address_low	•	•	•	•	•	•	•	•	a kitöltőminták RRAM kezdőcíme bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	a kitöltőminták RRAM kezdőcíme bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	a kitöltőminták RRAM kezdőcíme bit22..16

0 kötelezően nulla értékre kell beállítani

2DPT - PATTERN_FILL (72h, F2h / 114, 242)

A DEFINE_PATTERN paranccsal létrehozott azonosítójú, az RRAM-ba korábban feltöltött, kiválasztott kitöltőmintával az ebben a parancsban meghatározott téglalapot feltölti. A parancs sebességre optimalizált, így csak páros X koordináta és páros W szélesség érték engedélyezett, a két paraméter legutolsó bitjét a 2dfx ignorálja. A transzparenciát nem kezeli, pixelről pixelre másol az RRAM-ból.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	x	x	a	a	b	c	c	d	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a kitöltendő téglalap bal felső sarkának X koordinátája a képernyőn; alsó nyolc bit
Y_low	•	•	•	•	•	•	•	•	a kitöltendő téglalap bal felső sarkának Y koordinátája a képernyőn; alsó nyolc bit
W_low	•	•	•	•	•	•	•	x	a kitöltendő téglalap szélessége; alsó nyolc bit – az utolsó bitet a 2dfx ignorálja
H_low	•	•	•	•	•	•	•	•	a kitöltendő téglalap magassága; alsó nyolc bit
table_id	0	0	•	•	•	•	•	•	a kitöltőminta-tábla azonosítószáma
pattern_number	•	•	•	•	•	•	•	•	a használni kívánt kitöltőminta sorszáma

a a bal felső sarok X koordinátájának felső két bitje
b a bal felső sarok Y koordinátájának felső bitje
c a szélesség (W) pixelei számának felső két bitje
d a magasság (H) pixelei számának felső bitje
0 kötelezően nulla értékre kell beállítani
x ezeket a biteket a 2dfx figyelmen kívül hagyja

2DPT TILEMAP – csempealapú játéktér rajzolása

A **TILEMAP** elsőre talán a legösszetettebb, de egyben az egyik leghasznosabb 2DPT-parancsszerkezet. Előre definiált, ismétlődő mintákból – csempékből, azaz tile-okból – teljes pályákat vagy nagyobb játéktér-részleteket lehet vele felépíteni. A teljes pálya jóval nagyobb lehet, mint a látható képtartomány; a **TILEMAP** mindig ennek egy kiválasztott ablakát, az úgynevezett viewportot rajzolja ki. Ezt az ablakot egyetlen 2DPT-parancs paramétereinek módosításával lehet mozgatni.

A tilemap-rendszer három fő parancsból áll. Ezek közül kettő resource-alapú definíció, vagyis az aktuális rendereléstől függetlenül, globálisan állítja be a használandó adatokat.

A csempék fixen 16×8 pixeles képelemek. Egy csempe 64 bájtot foglal az RRAM-ban, a csempék pedig egymás után következnek. A **DEFINE_TILESET** megadja a csempekészlet logikai azonosítóját és azt az RRAM-címet, ahol a hozzá tartozó csempegrafikák kezdődnek. Egy TILESET legfeljebb 256 különböző csempét tartalmazhat, és egyszerre legfeljebb 64 TILESET definiálható. Egy pályaszerkezet egy adott TILESET-en belül tehát legfeljebb 256 csempetípussal dolgozik, de a képernyőn több TILESET is használható, ha több **TILEMAP** parancsot helyezünk el a 2DPT-ben.

A **DEFINE_TILESET** használata előtt a csempék képadatát fel kell tölteni az RRAM-ba, a szokásos UPLOAD/UNZX1 folyamat valamelyikével. A csempék a kezdőcímtől számítva fix lépésközzel következnek: a tile_id azonosítójú csempe címe tileset_base + tile_id * 64. A későbbi tilemap-adatok a csempékre 00h-FFh, vagyis 0-255 közötti sorszámmal hivatkoznak.

A **DEFINE_TILEMAP** egy teljes térképet ír le. Ez legfeljebb 256×256 csempéből állhat, vagyis tekinthetjük akár a teljes játéktérnek is. Használatához először az RRAM-ba kell tölteni a térkép celláit: minden bájt egy adott helyen lévő csempe sorszámát adja meg. A térképadatok W×H bájtot foglalnak, sorfolytonosan tárolva, ahol minden bájt az adott cellában használt csempe 0..255 közötti azonosítóját tartalmazza.

A **TILEMAP** maga a 2DPT-be kerülő rajzolóparancs. Ez határozza meg, hogy a korábban definiált térkép melyik része jelenjen meg a képernyőn. Meg kell adni a használni kívánt tilemap azonosítóját, a képernyőn megjelenő viewport bal felső X/Y koordinátáját, a térképen belüli bal felső forráscsempe X/Y pozícióját, valamint vízszintesen és függőlegesen a kirajzolandó csempék számát.

A munkafolyamat első lépése a csempegrafikák feltöltése az RRAM-ba. Ezután a **DEFINE_TILESET** paranccsal a feltöltött grafikai adat kezdőcímét egy logikai tileset-azonosítóhoz rendeljük.

Második lépésként fel kell tölteni az RRAM-ba a teljes térképet, ahol minden bájt a használt csempe sorszámát jelenti. Ezt a **DEFINE_TILEMAP** kapcsolja össze egy tilemap-azonosítóval; itt kell megadni a teljes térkép szélességét és magasságát is.

Az utolsó lépés a megjelenítés. A **TILEMAP** parancs bármelyik 2DPT-slotba beírható, és megadja, melyik tilemapből, melyik képernyőpozícióra, mekkora látható ablakot kell kirajzolni. A viewport méretét nem pixelekben, hanem csempék számában kell megadni vízszintes és függőleges irányban.

DEFINE_TILESET (5Eh / 94)

Egy tileset_id nevű azonosítóval rendeli össze az RRAM-ba korábban feltöltött, 16×8 pixel méretű csempekészletet. A csempekészlet 256 különböző mintázatú csempéből állhat.

A csempemintáknak az RRAM-ban egymást követően, folytonosan kell elhelyezkedniük. Minden egyes csempe 64 bájt, azaz egy teljes csempekészlet 16 kB méretű. Egyidejűleg legfeljebb 64 csempekészlet definiálható.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
tileset_id	0	0	•	•	•	•	•	•	a csempekészlet azonosító száma (0…63)
RRAM_address_low	•	•	•	•	•	•	•	•	a csempeképek RRAM kezdőcíme bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	a csempeképek RRAM kezdőcíme bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	a csempeképek RRAM kezdőcíme bit22..16

0 kötelezően nulla értékre kell beállítani

DEFINE_TILEMAP (5Fh / 95)

Legfeljebb 32 különböző, maximum 256×256 csempéből álló tilemap-et (térképet) definiálhatunk a parancs segítségével, amelyet tilemap_id-vel azonosítunk.

A térkép adatait előzőleg fel kell töltenünk az RRAM-ba, a maximális szélesség (W) és magasság (H) szorzatának eredménye számú bájt sorfolytonos feltöltésével, ahol minden bájt az adott térkép egy csempéjének az azonosítóját tartalmazza.

A térképet egyszerre egy csempekészlethez (TILESET) tudjuk hozzárendelni. Több tilemap-azonosító is használhatja ugyanazt a tileset_id-t.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
tilemap_id	0	0	0	•	•	•	•	•	a definiálni kívánt csempetérkép azonosítója (0..31)
RRAM_address_low	•	•	•	•	•	•	•	•	a térképadatok RRAM kezdőcíme bit7..0
RRAM_address_mid	•	•	•	•	•	•	•	•	a térképadatok RRAM kezdőcíme bit15..8
RRAM_address_high	0	•	•	•	•	•	•	•	a térképadatok RRAM kezdőcíme bit22..16
W_cell	•	•	•	•	•	•	•	•	a térkép vízszintes kiterjedése csempékben számolva – 1
H_cell	•	•	•	•	•	•	•	•	a térkép függőleges kiterjedése csempékben számolva – 1
tileset_id	0	0	•	•	•	•	•	•	a csempekészlet azonosító száma (0…63)

0 kötelezően nulla értékre kell beállítani

2DPT - TILEMAP (73h, F3h / 115, 243)

A megadott tilemap_id-vel megjelölt, előzőleg definiált és feltöltött tilemap adott részét rajzolhatjuk ki vele a képernyőre.

A parancsban meg kell adnunk a használni kívánt tilemap_id-t, a kirajzolás kezdőpontjának koordinátáit a képernyőn (X/Y), a bal felső sarok csempéjének a helyét az adott térképben (csempénként számítva) és a kirajzolni kívánt csempék számát mind vízszintes, mind függőleges irányban.

A csempék pixeleit a 2dfx transzparenciavizsgálat nélkül, bájtpontosan másolja a célterületre.

	adatbitek								
paraméter neve	7	6	5	4	3	2	1	0	leírás
slot_number	•	•	•	•	•	•	•	•	2DPT slot szám (0…255)
highbits	a	a	a	a	a	b	b	c	lásd a magyarázatot
X_low	•	•	•	•	•	•	•	•	a bal felső sarok X koordinátájának alsó nyolc bitje
Y_low	•	•	•	•	•	•	•	•	a bal felső sarok Y koordinátájának alsó nyolc bitje
X_source	•	•	•	•	•	•	•	•	az első csempe X pozíciója a térképben csempeszámban mérve
Y_source	•	•	•	•	•	•	•	•	az első csempe Y pozíciója a térképben csempeszámban mérve
C_draw	•	•	•	•	•	•	•	•	a rajzolni kívánt csempék száma vízszintes irányban
R_draw	•	•	•	•	•	•	•	•	a rajzolni kívánt csempék száma függőleges irányban

a a használt csempetérkép (tilemap_id) száma (0..31)

b a bal felső sarok X koordinátájának felső két bitje

c a bal felső sarok Y koordinátájának felső bitje

A 2dfx gyakorlati használata

Inicializálás

A 2dfx a számítógép áram alá helyezésekor, illetve a számítógép **RESET** gombjának megnyomásakor inaktív, alvó módba kerül. Egyik státuszbit sem ad vissza érvényes értéket, illetve a 2dfx hagyományos parancsokra nem reagál.

Alvó üzemmódban a 2dfx kizárólag a **SET_ENGINE_MODE** parancsot értelmezi és hajtja végre. A felhasználói szoftver futtatásakor tehát a 2dfx-et fel kell ébreszteni és munkára kell bírni az **ENGINE_ON** vagy az **ENGINE_ON_FORCED** üzemmódok valamelyikének bekapcsolásával:

```
OUT 248, 80
OUT 249, 1          ! ez a normál mód
```

vagy

```
OUT 248, 80
OUT 249, 255        ! ez a forced mód
```

A két üzemmód közötti eltérést a **SET_ENGINE_MODE** parancs leírása taglalja.

A 2dfx ébredés után végrehajtja a belső inicializálási rutinjait, ez **legrosszabb esetben két másodpercet vesz igénybe**.

Ezt követően a státuszregiszter kiolvasásával, annak hatodik bitjét lekérdezve (**AND 40h**) bizonyosodhatunk meg afelől, hogy a gépben van-e egyáltalán 2dfx kártya és az működőképes: a státuszbit értéke **nulla** lesz inicializált és működő 2dfx esetén.

A kb. 2 másodperces inicializálási idő alatt a 2dfx nem kommunikál, parancsokat nem hajt végre, mindenképpen várjunk a felhasználói programunkban időzítéssel vagy a státuszregiszter folyamatos olvasásával (megfelelő timeout-tal a végtelen ciklus elkerüléséhez, ha a gépben nincs 2dfx) és a hatodik bit nulla értékének ellenőrzésével!

Természetesen előfordulhat, hogy a 2dfx már aktív a programunk indításakor, mert egy előző felhasználói program már használta azt. Ezt a státuszregiszter (**F8h**) kiolvasásával, annak hatodik bitjének vizsgálatával (**AND 40h**) tudjuk eldönteni. Ha a bit értéke nulla, a 2dfx üzemképes állapotban van. Ilyen esetben annak érdekében, hogy az előző program által RRAM-ban, 2DPT-ben, SPB-kben hagyott adatait töröljük, adjuk ki a **SYS_RESET** parancsot és várjuk meg, amíg a státuszregiszter nulladik bitje (**AND 01h**) 1-re vált.

Javasolt workflow

A felhasználói program indításakor első lépésként hajtsuk végre az előző fejezet szerinti inicializálási lépéseket.

Ha a jelenlétellenőrzés sikeres, állítsuk be a szükséges globális paramétereket: a transzparens színt, az FPS-t, illetve minden olyan beállítást, amelyben el akarunk térni az induláskori alapértékektől. Ugyanígy célszerű a felhasználói programban a Z80 megszakításkezelő rutinját is kialakítani, hogyha raszterinterruptot tervezünk használni.

A következő lépésben célszerű feltölteni az RRAM tartalmát. Ez nagyobb adatmennyiségnél időigényes lehet, ezért érdemes közben valamilyen visszajelzést adni a felhasználónak. Ilyenkor például jól használható a beépített 2dfx logó megjelenítése (lásd: **SHOW_BUILTIN_LOGO** parancs), mert néhány paraméterbájton kívül nem igényel külön adatfeltöltést.

Az RRAM feltöltése után állítsuk be a resource-alapú parancsokhoz tartozó definíciókat: például a fontokat, bitmapeket, tilemapeket és minden egyéb **DEFINE_XX** jellegű beállítást. Így a későbbi 2DPT-parancsok már kész erőforrásokra hivatkozhatnak.

Ezt követheti a 2DPT back és front playlistek felépítése. Hasznos trükk, különösen BASIC-ből történő programozásnál, ha a nulladik slotot átmenetileg kihagyjuk, vagy csak legutoljára állítjuk be. Induláskor és **SYS_RESET** után minden 2DPT-slot **RET** parancsot tartalmaz. Márpedig ha a renderer a nulladik slotban **RET**-et talál, azonnal befejezi az adott playlist feldolgozását. Így elkerülhető, hogy a 2DPT feltöltése közben részleges, félkész grafika jelenjen meg. Miután a többi slotot beállítottuk, utolsó lépésként írjuk át a nulladik slotot **NOP**-ra vagy a kívánt első grafikai parancsra; a következő képkockában a renderer már a teljes playlistet fogja látni.

Hasonló logika alkalmazható az SPB-k előkészítésénél is. A programban használt sprite-ok SPB-it töltsük fel teljes, kilenc bájtos formában, de az első bájt SPRITE_ENABLE bitjét hagyjuk 0-n. Később így elegendő csak az első SPB-bájtot újraküldeni 1-re állított SPRITE_ENABLE bittel, és a sprite azonnal megjelenik a már előkészített paraméterekkel.

Ezzel az inicializálás nagy része elkészült. A felhasználói program ezután már a játékmenetre, demológikára vagy alkalmazáslogikára koncentrálhat; a grafikai munka jelentős részét a 2dfx végzi el.

A programunkból való rendes kilépéskor a korrektség kedvéért adjuk ki a **SET_ENGINE_MODE** parancsot nullás, azaz ENGINE_OFF paraméterrel.

Feltöltési útmutató

Az RRAM használata fejezetben olvasható, hogy a 2dfx többféle feltöltési és kicsomagolási módokat ismer. Ezek megfelelő kombinálásához segít az alábbi áttekintő táblázat.

A táblázat a lehetséges összes kombinálciót tartalmazza. **`TILEMAP`** térképadat; FONT karakterkészlet adat; szöveg/string **`DRAW_TEXT`**-hez kizárólag RAW adatok lehetnek.

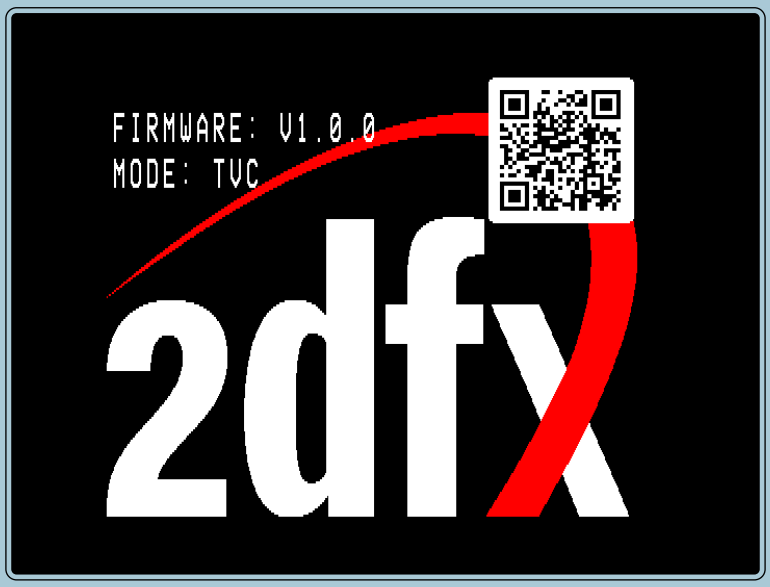
forrásadat jellege	forrás formátuma	tömörített?	feltöltési parancs	kicsomagolási parancs
képi adat sprite-hoz, BLIT-hez, PATTERN_FILL-hez; TILESET csempekép	2dfx belső	nem	UPLOAD_RAW	–
képi adat sprite-hoz, BLIT-hez, PATTERN_FILL-hez; TILESET csempekép	TVC	nem	UPLOAD_TVC	–
képi adat sprite-hoz, BLIT-hez, PATTERN_FILL-hez; TILESET csempekép	NICK	nem	UPLOAD_NICK	–
TILEMAP térképadat; FONT karakterkészlet adat; szöveg/string DRAW_TEXT-hez	2dfx belső	nem	UPLOAD_RAW	–
képi adat sprite-hoz, BLIT-hez, PATTERN_FILL-hez; TILESET csempekép	2dfx belső	igen	UPLOAD_RAW	UNZX1_RAW
képi adat sprite-hoz, BLIT-hez, PATTERN_FILL-hez; TILESET csempekép	TVC	igen	UPLOAD_RAW	UNZX1_TVC
képi adat sprite-hoz, BLIT-hez, PATTERN_FILL-hez; TILESET csempekép	NICK	igen	UPLOAD_RAW	UNZX1_NICK
TILEMAP térképadat; FONT karakterkészlet adat; szöveg/string DRAW_TEXT-hez	2dfx belső	igen	UPLOAD_RAW	UNZX1_RAW

Szoftverfrissítés

Bár a fejlesztés során a legtöbb időt a tesztelés vitte el, **a 2dfx garantáltan nem hibamentes**. Előfordulhatnak benne apróbb/nagyobb bugok, amiket természetesen igyekszem javítani a felhasználói visszajelzések alapján. Az előzőek természetesen igazak erre a kézikönyvre is.

A 2dfx működtetőszoftverének legutolsó verziója mindig a **`https://2dfx.net/firmware/latest`** címen érhető el.

A 2dfx az éppen futó szoftverének verzióját a **`SHOW_FW`** paranccsal jeleníti meg a képernyőn. A parancs kiadása után megjelenik a 2dfx logója, az aktuális firmware verzió, az üzemmód (TVC vagy Enterprise) és egy QR kód, ami a fenti letöltőlinkre mutat.



A **`SHOW_FW`** parancs kiadása után a rendes működéshez való visszatéréshez a TVC vagy Enterprise gépünket resetelni kell.

Mind a TVC-hez, mind az Enterprise-hoz készült 2dfx kártya ugyanazt a működtetőszoftvert futtatja (unified firmware), illetve ugyanazokat a rajzoló rutinokat használja, így a firmware-ből nincs külön TVC vagy Enterprise változat.

A **SHOW_FW** parancs által mutatott szoftververziót hasonlítsuk össze a weboldalon található kiírt verziószámmal. Amennyiben eltérést tapasztalunk, az alábbi, nagyon egyszerű módon tudjuk a 2dfx firmware-jét frissíteni:

A weboldalról az .uf2 kiterjesztésű fájlt töltsük le, majd a számítógépünket (PC vagy Mac) dugjuk össze egy hétköznapi USB kábellel a 2dfx USB-C bemenetével. Ezt követően kapcsoljuk ki a TVC-t vagy az Enterprise-t, majd kapcsoljuk be úgy, hogy közben nyomva tartjuk a 2dfx áramköri paneljén lévő egyetlen nyomógombot. Rövidesen a PC-nken megjelenik egy RP2350 nevű meghajtó, ekkor engedjük fel a nyomógombot. Az RP2350 meghajtóra csak át kell másoljuk a letöltött .uf2 fájlt, a szoftverfrissítés pedig az átmásolást követően automatikusan végbemegy. A művelet nem tart tovább néhány másodpercnél, ezután az RP2350 meghajtó eltűnik, a 2dfx pedig reseteli magát bekapcsoláskori állapotra, de már az új firmware-t futtatja, amit az inicializálási lépések után a **SHOW_FW** parancs kiadásával ellenőrizhetünk is.

Előfordulhat, hogy másolás közben hibajelzést kapunk a gépünktől, ebben az esetben ismételjük meg a ki-, majd bekapcsolás folyamatát és kíséreljük meg az .uf2 fájl újbóli átmásolását. A tapasztalatok szerint néhány tíz másodperc tétlenség után az RP2350 meghajtó nem mindig fogadja a másolást.

Összefoglaló parancstáblázat

RRAM parancsok

parancskód hex/dec	parancsnév	funkció
80h / 128	UPLOAD_RAW	adatok feltöltése RRAM-ba, utólagos feldolgozás nélkül, bájtpontosan
81h / 129	UPLOAD_NICK	adatok feltöltése RRAM-ba, feltöltés közbeni NICK szerinti bitátrendezéssel
82h / 130	UPLOAD_TVC	adatok feltöltése RRAM-ba, feltöltés közbeni TVC szerinti bitátrendezéssel
83h / 131	UNZX1_RAW	előzőleg RRAM-ba feltöltött adatok ZX1 kicsomagolása, utólagos feldolgozás nélkül
84h / 132	UNZX1_NICK	előzőleg RRAM-ba feltöltött adatok ZX1 kicsomagolása NICK bitátrendezéssel
85h / 133	UNZX1_TVC	előzőleg RRAM-ba feltöltött adatok ZX1 kicsomagolása TVC bitátrendezéssel

Sprite parancs

parancskód hex/dec	parancsnév	funkció
00h..3Fh / 0..63	ALTER_SPB	a parancskód szerinti sorszámú sprite SPB-jének módosítása

2dfx általános és rendszerparancsok

parancskód hex/dec	parancsnév	funkció
40h..4Fh / 64..79	SET_TRANSPARENT_COLOR	a parancskód alsó négy bitje meghatározza a transzparens színindexet – alapérték: 0
50h / 80	SET_ENGINE_MODE	meghatározza a 2dfx rendererének működését, az inicializálás első lépése!
51h / 81	SET_FPS	a 2dfx jelenetütemét állítja 50 fps / 25 fps között – alapérték: 50 fps
52h / 82	SET_RENDER_OVERRUN_VISUAL	bekapcsolja a renderidő túllépés vizuális (felkiáltójel) megjelenítését – alapérték: kikapcsolt
53h / 83	CLEAR_RENDER_OVERRUN	törli a bebillentett renderidő túllépés jelzését
54h..57h / 84..87	SET_RASTER_INTx	a négy lehetséges raszterinterrupt bekapcsolása vagy kikapcsolása – alapérték: mind a négy kikapcsolt
58h / 88	CLEAR_RASTER_INT	a bebillent raszterinterrupt(ok) törlése, magát az interruptbeállítást nem törli, csak a jelzést!
8Eh / 142	SHOW_FW	kiírja az aktuális firmware verziót, az aktuális üzemmódot – kilépni csak a számítógép resetelésével lehetséges!
8Fh / 143	SYS_RESET	melegreset; törli az RRAM-ot, SPB-ket, 2DPT back/front táblákat és a DEFINE_XX resource-definíciókat; alaphelyzetbe állítja a SET_FPS, SET_TRANSPARENT_COLOR és raszterinterrupt-beállításokat; az engine módot megőrzi
90h / 144	SHOW_BUILTIN_LOGO	kijelzi a 2dfx logóját adott X/Y koordinátára, fix 206×79 pixel méretben, minden rajzolási réteg fölött
91h / 145	HIDE_BUILTIN_LOGO	eltünteti az előző parancsban kirakott logót a képernyőről

2DPT resource definíciós parancsok

parancskód hex/dec	parancsnév	funkció
5Bh / 91	DEFINE_FONT	RRAM-ba feltöltött 1bpp fontadatokat font_id-hez rendel
5Ch / 92	DEFINE_BITMAP	RRAM-ba feltöltött bitmapet bitmap_id-hez rendel
5Dh / 93	DEFINE_PATTERN	RRAM-ba feltöltött kitöltőminta-készletet table_id-hez rendel
5Eh / 94	DEFINE_TILESET	RRAM-ba feltöltött csempealakzat-készletet tileset_id-hez rendel
5Fh / 95	DEFINE_TILEMAP	RRAM-ba feltöltött csempetérképet tilemap_id-hez rendel

2DPT back általános parancsok

parancskód hex/dec	parancsnév	funkció
60h / 96	NOP	nem csinál semmit, helykitöltő
61h / 97	SET_COLOR	kijelöli a továbbiakban használt tinta- és papírszínt a további parancsokhoz
6Eh / 110	SET_XY_OFFSET	a további 2DPT parancsok számára X/Y előjeles értéket hozzáad a kezdőkoordinátákhoz
6Fh / 111	SKIP_NEXT	a parancs paramétere szerinti következő x slotot a 2dfx átugorja, azokat nem hajtja végre
7Fh / 127	RET	a parancshoz érve a 2dfx befejezi a 2DPT feldolgozását függetlenül attól, hogy van-e további parancs a playlistben

2DPT back grafikai primitívek

parancskód hex/dec	parancsnév	funkció
62h / 98	PLOT	pontot rajzol tintaszínnel X/Y koordinátákon
63h / 99	LINE	vonalat rajzol tintaszínnel, megadott vonalvastagsággal, X0/Y0 és X1/Y1 között
64h / 100	ERASE_LINE	vonalat rajzol papírszínnel, megadott vonalvastagsággal, X0/Y0 és X1/Y1 között
65h / 101	BUCKET_FILL	alakzatot tölt ki tintaszínnel X/Y koordinátától, az abban lévő színű pixellel szomszédos pixelekkel addig, amíg más színbe nem ütközik
66h / 102	RECT	téglalapot rajzol tintaszínnel, majd azt papírszínnel kitölti, ha az nem a transzparens szín, megadott vonalvastagsággal, X/Y kezdőkoordinátától W szélességben és H magasságban
67h / 103	FILL_RECT	tintaszínű téglalapot rajzol, amit azzal ki is tölt, X/Y kezdőkoordinátától W szélességben és H magasságban
68h / 104	ROUND_RECT	RX és RY rádiuszú negyedellipszisekkel határolt, lekerekített sarkú téglalapot rajzol tintaszínnel, majd azt papírszínnel kitölti, ha az nem a transzparens szín, megadott vonalvastagsággal, X/Y kezdőkoordinátától W szélességben és H magasságban
69h / 105	FILL_ROUND_RECT	RX és RY rádiuszú negyedellipszisekkel határolt, lekerekített sarkú tintaszínű téglalapot rajzol, amit azzal ki is tölt, X/Y kezdőkoordinátától W szélességben és H magasságban
6Ah / 106	ELLIPSE	CX/CY origótól RX és RY rádiuszú ellipszist rajzol tintaszínnel, majd azt papírszínnel kitölti, ha az nem a transzparens szín, megadott vonalvastagsággal
6Bh / 107	FILL_ELLIPSE	CX/CY origótól RX és RY rádiuszú ellipszist rajzol tintaszínnel, amit azzal ki is tölt
6Ch / 108	ARC	ívet rajzol tintaszínnel, megadott vonalvastagsággal CX/CY origótól RX és RY rádiuszú ellipszis mentén start_angle és end_angle szögek között
6Dh / 109	PIE	kördiagramot rajzol CX/CY origótól RX és RY rádiuszú ellipszis mentén start_angle és end_angle szögek között, a kezdő és végpontokat az origóval összeköti, a belső részt tintaszínnel kitölti

2DPT back resource parancsok

parancskód hex/dec	parancsnév	funkció
70h / 112	DRAW_TEXT	a font_id-vel meghatározott karakterkészlettel az RRAM-ban eltárolt stringet (egyszerre max. 255 karakter) a megadott X/Y kezdőkoordinátától kirajzolja tintaszínnel, ha a papírszín nem a transzparens szín, akkor a mögöttes részt papírszínnel kitölti
71h / 113	BLIT	a bitmap_id-vel meghatározott bitmapet kirajzolja az X/Y koordinátától kezdve, transzparenciavizsgálattal vagy anélkül
72h / 114	PATTERN_FILL	a table_id-vel meghatározott kitöltőminta-készletből a megadott pattern_number szerinti mintával kitölti az X/Y koordinátától a W szélességű és H magasságú téglalapot
73h / 115	TILEMAP	a tilemap_id-vel meghatározott csempetérkép alapján X_source és Y_source csempétől kezdődően C_draw és R_draw számú térképrészletet rajzol adott X/Y kezdőkoordinátától

2DPT front általános parancsok

parancskód hex/dec	parancsnév	funkció
E0h / 224	NOP	nem csinál semmit, helykitöltő
E1h / 225	SET_COLOR	kijelöli a továbbiakban használt tinta- és papírszínt a további parancsokhoz
EEh / 238	SET_XY_OFFSET	a további 2DPT parancsok számára X/Y előjeles értéket hozzáad a kezdőkoordinátákhoz
EFh / 239	SKIP_NEXT	a parancs paramétere szerinti következő x slotot a 2dfx átugorja, azokat nem hajtja végre
FFh / 255	RET	a parancshoz érve a 2dfx befejezi a 2DPT feldolgozását függetlenül attól, hogy van-e további parancs a playlistben

2DPT front grafikai primitívek

parancskód hex/dec	parancsnév	funkció
E2h / 226	PLOT	pontot rajzol tintaszínnel X/Y koordinátákon
E3h / 227	LINE	vonalat rajzol tintaszínnel, megadott vonalvastagsággal, X0/Y0 és X1/Y1 között
E4h / 228	ERASE_LINE	vonalat rajzol papírszínnel, megadott vonalvastagsággal, X0/Y0 és X1/Y1 között
E5h / 229	BUCKET_FILL	alakzatot tölt ki tintaszínnel X/Y koordinátától, az abban lévő színű pixellel szomszédos pixelekkel addig, amíg más színbe nem ütközik
E6h / 230	RECT	téglalapot rajzol tintaszínnel, majd azt papírszínnel kitölti, ha az nem a transzparens szín, megadott vonalvastagsággal, X/Y kezdőkoordinátától W szélességben és H magasságban
E7h / 231	FILL_RECT	tintaszínű téglalapot rajzol, amit azzal ki is tölt, X/Y kezdőkoordinátától W szélességben és H magasságban
E8h / 232	ROUND_RECT	RX és RY rádiuszú negyedellipszisekkel határolt, lekerekített sarkú téglalapot rajzol tintaszínnel, majd azt papírszínnel kitölti, ha az nem a transzparens szín, megadott vonalvastagsággal, X/Y kezdőkoordinátától W szélességben és H magasságban
E9h / 233	FILL_ROUND_RECT	RX és RY rádiuszú negyedellipszisekkel határolt, lekerekített sarkú tintaszínű téglalapot rajzol, amit azzal ki is tölt, X/Y kezdőkoordinátától W szélességben és H magasságban
EAh / 234	ELLIPSE	CX/CY origótól RX és RY rádiuszú ellipszist rajzol tintaszínnel, majd azt papírszínnel kitölti, ha az nem a transzparens szín, megadott vonalvastagsággal
EBh / 235	FILL_ELLIPSE	CX/CY origótól RX és RY rádiuszú ellipszist rajzol tintaszínnel, amit azzal ki is tölt
ECh / 236	ARC	ívet rajzol tintaszínnel, megadott vonalvastagsággal CX/CY origótól RX és RY rádiuszú ellipszis mentén start_angle és end_angle szögek között
EDh / 237	PIE	kördiagramot rajzol CX/CY origótól RX és RY rádiuszú ellipszis mentén start_angle és end_angle szögek között, a kezdő és végpontokat az origóval összeköti, a belső részt tintaszínnel kitölti

2DPT front resource parancsok

parancskód hex/dec	parancsnév	funkció
F0h / 240	DRAW_TEXT	a font_id-vel meghatározott karakterkészlettel az RRAM-ban eltárolt stringet (egyszerre max. 255 karakter) a megadott X/Y kezdőkoordinátától kirajzolja tintaszínnel, ha a papírszín nem a transzparens szín, akkor a mögöttes részt papírszínnel kitölti
F1h / 241	BLIT	a bitmap_id-vel meghatározott bitmapet transzparenciavizsgálattal vagy anélkül kirajzolja az X/Y koordinátától kezdve
F2h / 242	PATTERN_FILL	a table_id-vel meghatározott kitöltőminta-készletből a megadott pattern_number szerinti mintával kitölti az X/Y koordinátától a W szélességű és H magasságú téglalapot
F3h / 243	TILEMAP	a tilemap_id-vel meghatározott csempetérkép alapján X_source és Y_source csempétől kezdődően C_draw és R_draw számú térképrészletet rajzol adott X/Y kezdőkoordinátától

Easter eggek, demók parancsai

parancskód hex/dec	parancsnév	funkció
C0h / 192	EGG_C0	firmware demó: "Minek nevezzelek?"
C1h / 193	EGG_C1	firmware demó: transzformációs benchmark
C2h / 194	EGG_C2	firmware demó: animált 2DPT bemutató
C3h / 195	EGG_C3	firmware demó: szövegek kezelése, megjelenítése
C4h / 196	EGG_C4	firmware demó: BLIT bemutató
C5h / 197	EGG_C5	firmware demó: PATTERN_FILL bemutató
C6h / 198	EGG_C6	firmware demó: mini TILEMAP bemutató
C7h / 199	EGG_C7	firmware demó: játszható klasszikus Pong
C8h / 200	EGG_C8	firmware demó: WOW pong
C9h / 201	EGG_C9	firmware demó: Kukacinvázió!
CAh / 202	EGG_CA	firmware demó: Boulder Dash by 2dfx
CBh / 203	EGG_CB	firmware demó: render overrun bemutató
CCh / 204	EGG_CC	firmware demó: diagnosztikai tesztek
CDh / 205	EGG_CD	firmware demó: teljes képernyős BLIT
CEh / 206	EGG_CE	firmware demó: A börtön
CFh / 207	EGG_CF	firmware demó: TVC külső szín mintavételi teszt
D0h / 208	EGG_D0	firmware demó: TVC képfrissítési stressзteszt
D1h / 209	EGG_D1	firmware demó: a látható képernyőterület koordinátái
D2h / 210	EGG_D2	firmware demó: animált 2DPT és SET_XY_OFFSET bemutató

Beépített firmware demók (Easter Eggs)

Beépített firmware demók (Easter Eggs)

A 2dfx-be eleinte csak kísérleti, mérési és demonstrációs céllal kerültek be demók vagy más néven Easter Eggek. Ezek valójában egy adott 2dfx funkció vagy funkciócsoport ellenőrzésére szolgáltak, de végül bennhagytam őket a kódban, mert akár csak ezeket végignézve is egész jó képet kaphatunk a 2dfx által nyújtott szolgáltatásokról, illetve arról is, hogy ezek mennyire számításigényesek a 2dfx számára; így lehet némi viszonyítási alapunk, hogy a 2dfx milyen mértékben terhelhető.

Az érdekesebb eggek/demók leírásában található egy mért renderidő mind Enterprise, mind TVC környezetben. A két érték valamelyest eltér egymástól, több okból: egyrészt az Enterprise esetén nagyobb a vízszintes felbontás, ezért több a látható(vá tehető) pixelek száma, másrészt pedig a demók jelentős részét adoptáltam a két gép eltérő alap (IS-BASIC vs. TVC BASIC) képernyőfelbontásához, ami eltérő számításigényt hozhat magával. A renderidőknél zárójelben feltüntetett százalékos érték a tényleges használatot jelenti a 20 ms időkeretből.

A renderidőknek akkor van minimum és maximum értéke, amikor a képernyőn látható tartalom változik, pl. a C2-es demóban a megjelenő körök mérete folyamatosan nő, illetve csökken, ez mérhető időbeli különbséget okoz. Jól megfigyelhető a renderidők változásával a 2dfx alapfilozófiája szerinti működés, amely szerint a rajzolási feladatok nem feltétlenül csak azok számától, hanem azok bonyolultságától is függenek.

A renderer üresjáratban (inicializálás vagy **`SYS_RESET`** után) mindkét gépen ~128 us időtartamot elfoglal a rendelkezésre álló 20 vagy 40 ms időkeretből. A demóknál feltüntetett renderidők ezzel együtt értendők.

A renderelés időtartamát egyébként oszcilloszkóp segítségével is követhetjük: a 2dfx nyomtatott áramköri paneljén egy dedikált RTIME forrasztási pont, valamint a mellette lévő GND forrpont szkópra kötésével magunk is láthatjuk a tényleges renderidőt: az adott képkocka előállításának kezdetekor az RTIME 0-ba vált (0V), majd a képkocka generálásának befejeztével 1-be (3,3V). A nullában eltöltött időtartam maga a képgenerálás ideje, azaz a renderidő.

A demók 50 fps-es módban, a maximális látható vízszintes felbontásban készültek, ezért Videoton TVC gépen való használatukat megelőzően (vagy akár közben) váltsunk GRAPHICS 2 módra. Enterprise esetén ez nem szükséges, ott minden video üzemmódban rendelkezésre áll a maximális vízszintes felbontás.

Az Enterprise IS-BASIC-je nem túl színgazdag. Ahhoz, hogy a firmware demók látványosabbak legyenek a képernyőn, célszerű átállítani az aktuális palettát a következők szerint, majd utána indítani az egg-et a megfelelő OUT paranccsal, de előtte ne feledkezzünk meg a 2dfx inicializálásáról!

```
SET £102:PALETTE 0,73,75,219,146,182,109,255
OUT 248, xxx  ! xxx = az egg parancskódja
```

Az eggek képernyőfotóit (a C0 kivételével) Videoton TVC-n készítettem.

EGG_C0 (C0h / 192) – "Minek nevezzelek?"

Ezt a demót kifejezetten a 2026. május 29-én megtartott Enterprise Klubtalálkozóra készítettem látványelem gyanánt: miután bemutattam a tesztpanelt és a működését, utolsó mozzanatként "adjunk nevet a gyereknek" okán készült ez el. Itt tudta meg a Tisztelt Közönség, hogy a panel neve: **2dfx**. Öt fő jelenetből áll a demó:

1. káosz – Itt az Enterprise felirat egyes betűi pattognak egymástól függetlenül a képernyőn mintegy másfél percig. A betűk mindegyike egy-egy sprite, viszont az RRAM-ban tárolva csak a különböző betűk képei vannak: E, N, T, R, P, I, S. A sprite-ok egyenként 52 pixel magasak és a szélességük változó: 14-34 pixelből állnak. A korrekt arányú megjelenítéshez minden sprite-nál be van kapcsolva az Xdouble bit, így minden pixel vízszintes irányban megduplázódik. Emellett látható hat színes csík, ezek szintén sprite-ok, 48x6 pixel méretben kerülnek a képernyőre (itt is be van kapcsolva az Xdouble bit).

A képernyő jobb és alsó széléhez érkezve az adott betű sprite egy 11 vízszintes és 11 függőleges, előre generált mozgási fázisból álló animáción megy keresztül, ilyenkor a sprite "összecsukódik", majd ugyanez a szekvencia fordított sorrendben megismétlődik, de úgy, hogy a helyzettől függően az adott sprite SPB-jében az Xflip vagy az Yflip transzformációs bitek is bekapcsolásra kerülnek és már a traszformált sprite folytatja a képernyőn az útját. Vagyis például a fejjel lefelé repülő betű sprite adata nincs külön eltárolva az RRAM-ban, a renderer menet közben végzi a transzformációt az SPB-ben beállított transzformációs bit alapján. Az animációs fázisok változtatása az adott sprite képadat RRAM-beli kezdőcímének menet közbeni átírásával történnek.

A jobb szélső színes csík és a T betű kapcsolatát is érdemes megfigyelni: ennél a két sprite-nál ugyanis be van kapcsolva az ütközésdetektálás. Amennyiben a két sprite-nak látható pixelei kerülnek fedésbe egymással (ütköznek), a képernyő alsó harmadában középen egy színes csík villan fel, ami addig marad a képernyőn, amíg az ütközési állapot fennál. Ez a csík is egy sprite, az egg belső logikája folyamatosan ellenőrzi az ütközés meglétét; ütközéskor mindössze a sprite SPB-jében a SPRITE_ENABLE bit kerül bekapcsolásra, ütközés végén pedig kikapcsolásra.

2. összeállás – Minden betű a színes csíkok fölé mozog, összeállítva az ENTERPRISE feliratot.

3. visszaforgatás – Azok a sprite-ok, amelyek az összeálláskor valamilyen Xflip vagy Yflip transzformációs fázisban voltak, azok a káoszban leírt animációs fázisokat megismételve a helyükön visszafordulnak. Valamelyik tengelyére szimmetrikus betűalakzatok (pl. "I" betű) esetén az adott visszaforgatás nem történik meg.

4. bemutatkozik a 2dfx – A harmadik jelenet után körülbelül tíz másodperccel gravitációs hatást és visszapattanást szimulálva a 2dfx logója lezuhan és adott Y pozícióban megáll. A logó mérete 515×198 pixel, külön kerül feltöltésre (nem a beépített **`SHOW_BUILTIN_LOGO`**).

5. szivárvány – Az Enterprise felirat alatti színes csíkok (sprite-ok) váltakozni kezdenek, ezt úgy érjük el, hogy a csíkok X pozícióit változtatjuk a vonatkozó SPB-kben.

Összefoglaló: látványos animációt lehet készíteni a 2dfx-szel mindössze SPB-k változtatásával. Miután az animációs fázisok letárolásra kerülnek az RRAM-ban, a tíz, egymástól függetlenül mozgó, finoman forduló betű, a hat színes csík, az ütközésjelzés és a nagy lezuhanó logó kizárólag a vonatkozó SPB-k módosításával történik.

Renderidők alakulása:

jelenet	renderidő EP	renderidő TVC
káosz	2,5 ms (12,5%)	2,42 ms (12,1%)
utolsó jelenet, minden a helyén	2,97 ms (14,85%)	2,93 ms (14,65%)

EGG_C1 (C1h / 193) – transzformációs benchmark

Ebben a demóban mind a 64 lehetséges sprite megjelenik a képernyőn, SPB-k használatával. A demó célja a sprite transzformációk bemutatása és az ehhez szükséges renderidő különbségek dokumentálása volt.

A két festményszerű (Piet Mondrianra és Victor Vasarely-re hajazó) téglalap előre, offline módon kiszámolt adatokból áll, amelyek ZX1-el tömörítésre kerültek. Méretük óriási: 540×230 pixel. Ennek ellenére viszonylag kevés helyet foglalnak tömörítve. Az 540×230 pixel mérethez 62100 bájtnyi adat eltárolására és feltöltésére volna szükség. Ezzel szemben a tényleges adatmennyisége a Mondrian téglalapnak 170 bájt (!), míg a Vasarely-nek 25851 bájt. A demó ezeket feltölti az RRAM-ba az **`UPLOAD_RAW`** parancs belső megfelelőjével (amely a vonatkozó F8h parancs API-ját használja, tehát pontosan úgy működik, mintha Enterprise-ról vagy TVC-ről töltöttük volna fel), majd az **`UNZX1_RAW`** segítségével tömöríti ki az RRAM-ba őket.

A 62 db. smiley SPB ugyanazt a 16×16 pixel méretű, RRAM-ba **`UPLOAD_RAW`** paranccsal feltöltött sprite-adatot használja és egyszer kerül feltöltésre. A 16×16 pixeles smiley mindössze 128 bájt helyet foglal az RRAM-ban, ezért ennél a kisméretű grafikánál szándékosan nem alkalmaztam ZX1 tömörítést. Érdekességképpen a tényleges smiley-adat ZX1-gyel az Enterprise változat esetében 76, a TVC változatnál 84 bájtra tömöríthető.

A demó három fő jelenetből áll:

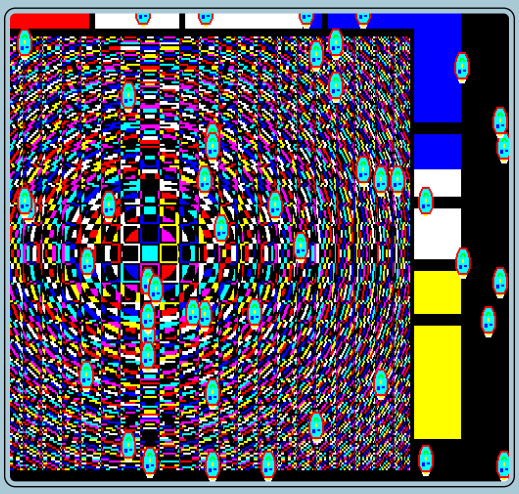
1. megérkeznek a sprite-ok – A képernyőn megjelenik először a két 540×230 pixeles festmény, majd egymás után a 62 db. 16×16-os smiley.

2. transzformációk bemutatása a smiley sprite-okkal

Itt az SPB-kben a különféle transzformációs flagek használatával módosítjuk a látható képadatot. Semmilyen más műveletet nem végzünk közben magával az RRAM-ban letárolt sprite-adattal. A két nagy sprite szándékosan nem kerül transzformálásra.

3. sprite-ok mozgatása – A két nagy sprite a képernyőn definiált területen belül pattog, a 62 db. smiley pedig véletlenszerűen meghatározott irányba és sebességgel mozog; ahol kiér a képernyőről, az ellenkező irányból belép. Lehet, hogy majd néhány sprite állni fog: ez konkrétan a véletlen műve, ugyanis a sprite-onkénti X és Y sebesség véletlen előállításakor mindkét érték nulla lett. Eközben a második jelenet transzformációs fázisai kerülnek alkalmazásra a smiley-kon, végtelenítve.

Összefoglaló: mind a 64 sprite megjelenhet a képernyőn, mozoghat, és különböző transzformációk egyidejű alkalmazásával sem feltétlenül éri el a kritikus 20 ms renderidőt.



Renderidők alakulása:

transzformáció	renderidő EP	renderidő TVC
normál – kikapcsolt transzformációs bitek	5,81 ms (29,05%)	4,52 ms (22,6%)
X double	6,32 ms (31,6%)	5 ms (25%)
Y double	6,12 ms (30,6%)	4,75 ms (23,75%)
X+Y double	7,28 ms (36,4%)	5,92 ms (29,6%)
X flip	5,92 ms (29,6%)	4,64 ms (23,2%)
Y flip	5,72 ms (28,6%)	4,44 ms (22,2%)
X+Y flip	8,82 ms (44,1%)	7,54 ms (37,7%)
X double + X flip	6,32 ms (31,6%)	5,02 ms (25,1%)
Y double + Y flip	8,82 ms (44,1%)	7,54 ms (37,7%)
X+Y double + X+Y flip	7,42 ms (37,1%)	6,04 ms (30,2%)

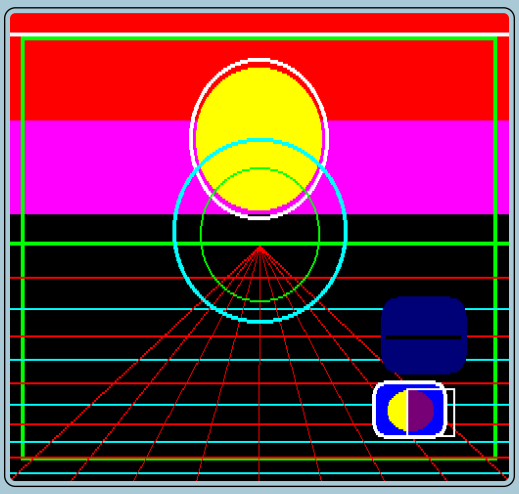
EGG_C2 (C2h / 194) – animált 2DPT bemutató

Ebben a demóban nincsenek sprite-ok egyáltalán, csak a 256-os 2DPT back playlistet használja, azon belül is a teljes parancskészletből az általános **SET_COLOR** és **RET** parancsokon kívül a grafikai primitíveket. Összesen 73 db. 2DPT slotból áll a képernyő megrajzolása, amelyből 36 db. **SET_COLOR**, 1 db. **RET**, a többi pedig **FILL_RECT**, **LINE**, **RECT**, **ELLIPSE**, **ROUND_RECT**, **FILL_ROUND_RECT**, **ERASE_LINE** és **BUCKET_FILL**.

Minden képfrissítést követően újraépíti a teljes 2DPT táblát – ugyan erre valójában semmi szükség, hisz elegendő volna csak kifejezetten azokat a slotokat felülírni, amelyekben a koordináták (pl. a mozgó vonal, vagy az alsó mozgó téglalap) és/vagy a rádiuszok (felső két kör). A renderer minden 20 ms-ban üres képernyővel indul, és végigmegy a 2DPT slotokon, amíg **RET** utasítást nem talál.

A renderelési idő változása annak köszönhető, hogy a rajzolt körök mérete változik menet közben.

Összefoglaló: sprite-ok és resource-alapú 2DPT parancsok nélkül is komplex pályaképet lehet készíteni, így pl. egy Elite-típusú játék teljes játékmezőjének megrajzolása mindössze a rajzolási paraméterek megváltoztatásával elvégezhető.



Renderidők alakulása:

paraméter	renderidő EP	renderidő TVC
minimum	0,95 ms (4,75%)	0,87 ms (4,35%)
maximum	1,18 ms (5,9%)	1,11 ms (5,55%)

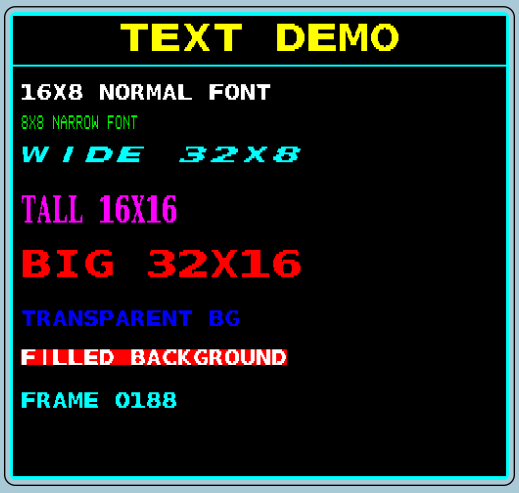
EGG_C3 (C3h / 195) – szövegek kezelése, megjelenítése

Ez a demó egyszerre használ öt komplett, 128 karakterképet ábrázoló fontkészletet a 2DPT resource-alapú **DEFINE_FONT** és **DRAW_TEXT** parancsaival: 8×8 pixeles keskeny font, 16×8 pixeles normál font, 32×8 pixeles széles font, 16×16 pixeles magas font és 32×16 pixeles nagy font.

Minden font offline módon van tárolva az MCU kódjában, ZX1 tömörítéssel. Ezt a demó induláskor feltölti az RRAM-ba az **UPLOAD_RAW** parancs segítségével és kicsomagolja az **UNZX1_RAW**-val, majd definiálja a 2dfx részére a **DEFINE_FONT** parancsokkal. Ezen felül szintén feltölti magukat a stringeket az RRAM-ba, majd megjeleníti azokat a **DRAW_TEXT** segítségével. Menet közben a demó mindössze a frame számláló által kiírt szövegrészletet módosítja az RRAM-ban (újat tölt fel), minden mást, így az egész 2DPT táblát változatlanul hagyja.

A 2DPT-ből 24 slotot használ, ami magát a képernyőt rajzolja meg (téglalap, benne a vonal, valamint **SET_COLOR**), valamint a **DRAW_TEXT** parancsot a különböző fontdefiníciókra és az RRAM-beli stringekre hivatkozva.

Összefoglaló: a 2dfx támogatásával szövegek kiírásához, vagy egy játékban feliratok, HUD-ok megjelenítéséhez a 2dfx részéről kevés erőforrást igényel a kirajzolás, illetve az Enterprise vagy TVC oldaláról a stringek kirakása is egy nagyon egyszerű folyamat.



Renderidők alakulása:

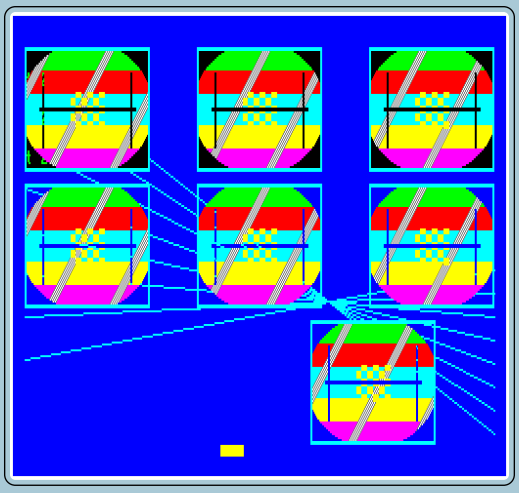
renderidő EP	renderidő TVC
1,32 ms (6,6%)	1,31 ms (6,55%)

EGG_C4 (C4h / 196) – BLIT bemutató

Ebben a demóban a **BLIT**-tel megvalósítható műveleteket látjuk. A demóban használt 128×64 pixeles bitmap objektumnak 64 darab, előre, offline módban legyártott animációs fázisa van. Ezek egyetlen ZX1-tömörített animációs bankként találhatók meg az MCU kódjában; a bank az **UPLOAD_RAW** és **UNZX1_RAW** műveletek megfelelőivel kerül az RRAM-ba, ahol kitömörítés után 256 kB területet foglal el. Ráadásul, hogy a betöltött adatok mennyisége még nagyobb legyen, ugyanezt a 64 fázist 16 fázissal eltolva még három további bankban is eltároljuk az RRAM-ban. Ennek gyakorlati haszna nincs, mindössze azt mutatja be, hogy egyszerűen tudunk nagy mennyiségű adatot betölteni a 2dfx-be. A négy bank együtt 1 MB RRAM-területet foglal.

A kirajzolás a 2DPT-vel történik kétféle módon: az első sorban a **BLIT** objektumok transzparenciavizsgálat nélkül kerülnek a képernyőre. A különböző források miatt jól látszik a mozgó csíkok egymáshoz képest eltolt helyzete. A második sorban ugyanezek az objektumok már transzparenciavizsgálattal kerülnek ki, míg az alsó sorban szintén transzparens **BLIT**-tel jelenik meg a vízszintesen mozgó példány.

A demó menet közben a 2DPT-ben lévő **BLIT** parancsokat nem változtatja, kivéve az alsó sorban mozgó példány X koordinátáját. Az animációt elsősorban úgy éri el, hogy képkockánként módosítja a **DEFINE_BITMAP** descriptorban a bitmap RRAM-beli kezdőcímét, így az mindig az aktuális 4 KiB-os animációs fázisra mutat. A látható animáció közben új bitmap-adat már nem kerül feltöltésre vagy kitömörítésre.



Renderidők alakulása:

renderidő EP	renderidő TVC
1,77 ms (8,85%)	1,63 ms (8,15%)

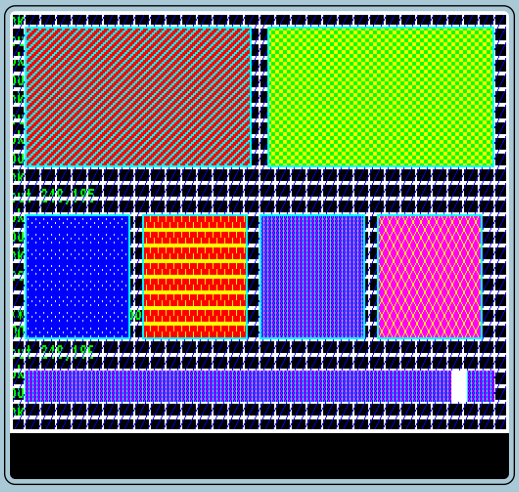
EGG_C5 (C5h / 197) – PATTERN_FILL bemutató

Ez a demó a 2DPT **PATTERN_FILL** parancsát mutatja be működés közben. A demó nyolc darab 16×8 pixeles kitöltőmintát készít és használ. A nagy háttérterület, a két széles téglalap, a három kisebb téglalap és a legalsó széles sáv mind ebből a nyolcféle mintából dolgozik. A képernyő alján egy kisebb, 16×16 pixeles **FILL_RECT** is folyamatosan mozog.

Működését tekintve a nyolc fix méretű minta egymás után, egyetlen 512 bájtos pattern bankban van tárolva az RRAM-ban, amelyet egyetlen **DEFINE_PATTERN** descriptor tesz elérhetővé a 2DPT számára. A különböző, egymástól független **PATTERN_FILL** parancsok ebből a bankból választják ki az aktuálisan használni kívánt mintát.

A nagy háttér mintázata lassabban, az alsó széles sáv mintázata gyorsabban változik, miközben a 16x16 pixeles tömör téglalap végighalad az alsó sávon. A demó minden frissítéskor újra felépíti a háttér 2DPT-listáját.

Összefoglaló: a mintázatos kitöltés nagyon jól használható például padlók, állapotkijelzők hátterének vagy egyéb ismétlődő felületek rajzolásához, miközben az ehhez szükséges grafikai erőforrás mérete, illetve számítási ideje is minimális.



Renderidők alakulása:

renderidő EP	renderidő TVC
2,16 ms (10,8%)	1,73 ms (8,65%)

EGG_C6 (C6h / 198) – mini TILEMAP bemutató

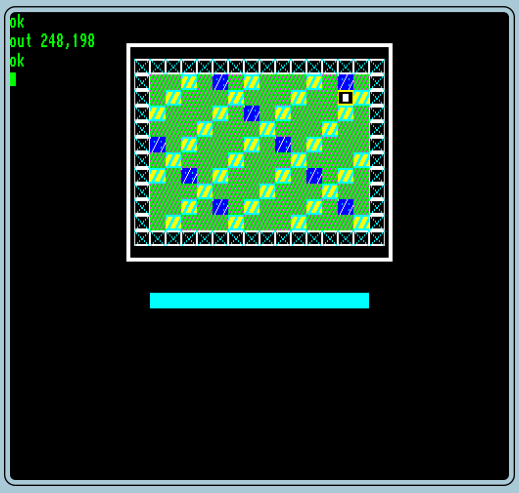
Itt a 2DPT **TILEMAP** parancsának működését láthatjuk. A megjelenített pálya 16×12 darab, egyenként 16×8 pixeles csempéből áll. A tilemap körül egy keret, alatta pedig egy színes sáv látható; ezeket egyszerű 2DPT grafikai primitívek rajzolják ki.

A demó összesen nyolc különböző 16×8 pixeles csempe számára foglal helyet a csempekészletben (tileset). Egy csempe 64 bájt, így a teljes nyolc csempéből álló csempekészlet 512 bájt RRAM-ot foglal.

A 16×12 cellás tilemapnál minden cellához mindössze egyetlen bájt tartozik, amely megadja, hogy az adott helyen melyik sorszámú csempe jelenjen meg, így a teljes pálya leírója 192 bájt az RRAM-ban.

A tilemap parancsszerkezetét ebben a demóban is a leírásban található három paranccsal működtetjük: magát a csempekészletet a **DEFINE_TILESET** paranccsal, a térképet (tilemap) egy **DEFINE_TILEMAP** paranccsal definiáljuk. Ezután a komplett 16×12-es TILEMAP-et egyetlen 2DPT slot paranccsal rakjuk a képernyőre.

A demó működés közben a három parancsot összesen egyszer állítja be. Az animáció közben a 192 bájtos pályaleíró készül el újra (az adott helyen módosuló csempe számának átírásával), majd kerül feltöltésre az RRAM-ba. Ezt is lehetne nyilván észszerűsíteni, hogy csak azt a néhány bájtot írjuk felül az RRAM-ban, ami ténylegesen változik, de az egyszerűség kedvéért mind a 192 bájtot újra feltöltjük. A következő rendereléskor ugyanaz az egyetlen 2DPT **TILEMAP** parancs már automatikusan az új tartalmat rajzolja ki.



Renderidők alakulása:

renderidő EP	renderidő TVC
0,47 ms (2,35%)	0,4 ms (2%)

EGG_C7 (C7h / 199) – játszható klasszikus Pong

Ez a demó egy egyszerű, játszható Pong játék, amely bemutatja, hogyan lehet néhány sprite és a 2DPT együttes használatával egy kvázi komplett játékot felépíteni. A két ütő és a labda három külön sprite, a játéktér, a középvonal és a pontszámok 2DPT parancsokkal készülnek.

A három sprite: az ütők 12×48 pixeles téglalapok, a labda pedig 12×6 pixeles téglalap. Ezek offline módon tárolva vannak, a demó induláskor az **UPLOAD_RAW** belső megfelelőjével tölti fel őket az RRAM-ba (összesen 612 bájt).

A sprite-adatok menet közben nem változnak, kizárólag az SPB-kben változtatom a kirajzolás helyét (X és Y koordináták).

A playfield néhány egyszerű 2DPT parancsból áll, **RECT** és **FILL_RECT** használatával. Maguk a pontszámkijelzések is **FILL_RECT** téglalapok a megfelelő helyeken az adott értéktől függően.

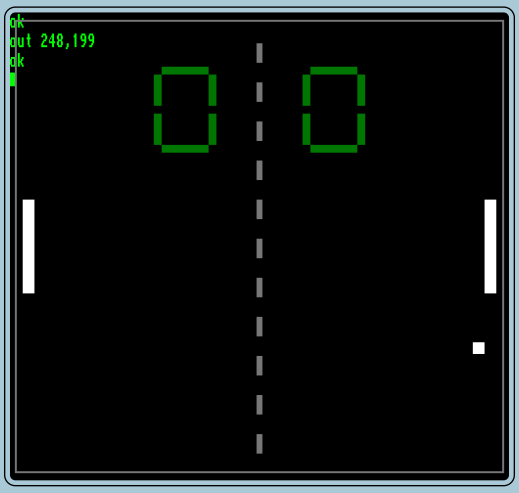
A demó minden képfrissítéskor kiszámolja a labda és az ütők új helyzetét és frissíti a három SPB-t, valamint újra felépíti a háttér teljes 2DPT playlistjét. Itt lehetne egyszerűsíteni, hogy csak az esetlegesen változó pontértékeknek megfelelő slotokat írjuk felül. A demó a 2dfx által nyújtott ütközésvizsgálatot nem használja, maga a logika számolja ki a téglalapok alapján az ütközést és változtatja meg a labda X/Y koordinátáit.

Amikor az egyik játékos pontot szerez (a labda a másik térfélen kiesik), akkor a playfieldet megrázzuk. Ezt a **SET_XY_OFFSET** állításával érjük el, sokkal egyszerűbb, mint az összes vonatkozó 2DPT slot X és Y értékeit újraszámolni és feltölteni. Mivel a **SET_XY_OFFSET** kizárólag a 2DPT playlistre vonatkozik, látható, hogy a sprite-ok nem rázkódnak együtt a pályával.

Rövid BASIC programmal Enterprise-on vagy TVC-n játszható is a Pong, ha a működése közben az F9h port adott bitjeinek értékét megváltoztatjuk, azzal jelezzük a demó felé, hogy új játékot indítunk vagy hogy az adott ütőt mely irányba akarjuk mozgatni. A mozgatás jelzését törölni kell, amikor felengedjük a billentyűt! Íme az Enterprise-on futó BASIC program, amivel az irányítást el tudjuk végezni:

```
100 PROGRAM "pong.bas"
110 SET KEY RATE 1
120 SET KEY DELAY 1
130 SET KEY CLICK OFF
140 SET £102:PALETTE 0,73,75,219,146,182,109,255
150 OUT 248,80
160 OUT 249,1
170 WAIT DELAY 2
180 OUT 248,199
190 LET A$=INKEY$
200 IF A$="q" THEN OUT 249,8
210 IF A$="a" THEN OUT 249,4
220 IF A$="p" THEN OUT 249,2
230 IF A$="l" THEN OUT 249,1
240 IF A$="n" THEN OUT 249,128
250 IF A$="" THEN OUT 249,0
260 GOTO 190
```

Összefoglaló: ez a demó egy egyszerű példa arra, hogy egy játék képernyőjének előállításához valójában milyen kevés adatot kell változtatni. Miután a három sprite-adat egyszer bekerült az RRAM-ba, játék közben gyakorlatilag csak néhány koordináta és a pontszám változik; a playfieldet a 2dfx minden képkockánál néhány 2DPT paranccsal állítja elő.



Renderidők alakulása:

renderidő EP	renderidő TVC
0,32 ms (1,58%)	0,31 ms (1,56%)

EGG_C8 (C8h / 200) – WOW pong

Ez a demó a C7-es klasszikus Pong vizuálisan felturbózott változata. A játékmenet megegyezik a klasszikus Ponggal, de szándékosan teli van rakva mindenféle egyéb, 2DPT-vel egyszerűen megvalósítható grafikai elemmel. Vizuális kísérletnek hívom, gyakorlatilag élvezhetetlenül csicsás, bár tekinthetjük úgy is, hogy advanced level pong…

Ebben a három sprite mellett fontot, tilemapet, patterneket, szövegeket és különböző 2D grafikai primitíveket is használunk és a C7-nél ismertetett BASIC programmal játszható is annak, aki szereti a kihívásokat…

A háttér egyik része egy animált tilemap. Enterprise-on ez 39×22, TVC-n 30×24 cellából áll; egy cella továbbra is egyetlen bájt, amely az adott helyen megjelenítendő csempe sorszámát tartalmazza. A hozzá tartozó csempekészlet (tileset) nyolc darab csempét használ. A tilemap tartalma minden képfrissítésnél újra elkészül. A keret, a vízszintes és függőleges "díszítőelemek" mellett bizonyos csempék helyzete is változik, ezért a háttér folyamatosan mozgónak tűnik. Magát a csempekészletet természetesen nem töltjük fel újra, csak a néhány száz bájtos térkép (tilemap) változik. A **DEFINE_TILESET** és **DEFINE_TILEMAP** descriptorok ugyanarra az RRAM-területre mutatnak végig.

A demó négy darab **PATTERN_FILL** mintát is használ. Ezekből készülnek a felső és alsó animált sávok, valamint a pontszámok körüli és középső panelek. A minták kiválasztása időben változik, így további mozgás látszik a háttérben anélkül, hogy új grafikai adatot kellene feltölteni menet közben.

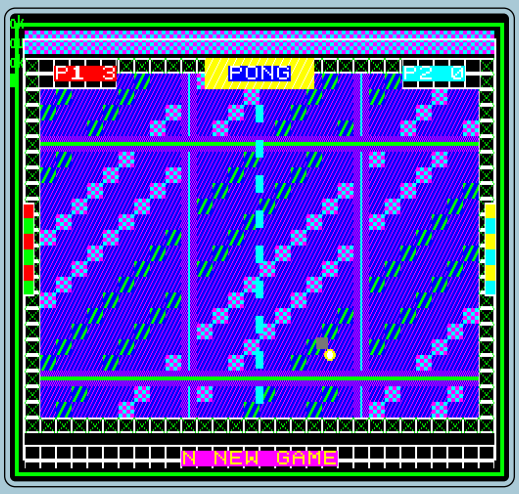
A feliratokhoz egy komplett, 16x8 pixeles fontkészlet kerül az RRAM-ba. Egy karakter nyolc bájtot foglal, így a teljes font mérete 1024 bájt. A „PONG”, „P1 0”, „P2 0” és „N NEW GAME” stringek külön RRAM-területen találhatók. Pontszerzéskor nem kell új **DRAW_TEXT** parancsot létrehozni a 2DPT-ben, a P1 vagy P2 stringben lévő számjegy karakterét írjuk át az RRAM-ban, ezzel a már 2DPT-ben lévő **DRAW_TEXT** ugyanarra a stringre hivatkozik mindig.

A 2DPT a **TILEMAP** és **PATTERN_FILL** parancsok mellett **RECT**, **FILL_RECT** és **DRAW_TEXT** utasításokból épül fel. Ezek rajzolják ki a playfield kereteit, a középvonalat, a feliratokat, egyebeket és a labda mögött látható egyszerű „utánhúzásos” effektet.

Pontszerzés után ez a demó is megrázza a képet, de a C7-nél valamivel látványosabb módon. A 2DPT-vel rajzolt háttér és a sprite-ok két külön rázási animációt kapnak. A háttér koordinátái egy közös eltolást kapnak **SET_XY_OFFSET** használatával, míg az ütők és a labda SPB-koordinátái ettől függetlenül, kissé eltérő értékekkel változnak. Emiatt az egész jelenet együtt rázkódik, mégsem merev képként mozog.

A játék minden 50 Hz-es frissítésnél kiszámolja az új játékállapotot, elkészíti és feltölti az aktuális tilemapet, frissíti szükség esetén a pontszám stringjét, újraépíti a háttér 2DPT-t, majd beállítja a három sprite aktuális SPB-jét. A nagyobb grafikai erőforrások – a sprite-ok, a font, a tileset és a pattern bank – eközben végig változatlanul az RRAM-ban maradnak.

Összefoglaló: a C7-hez képest itt jól látható, hogy ugyanaz a nagyon egyszerű játéklogika komplexebb képernyővel is körülvehető anélkül, hogy magának a számítógépnek képpontonként bármit is rajzolnia kellene.



Renderidők alakulása:

renderidő EP	renderidő TVC
1,56 ms (7,8%)	1,42 ms (7,1%)

EGG_C9 (C9h / 201) – Kukacinvázió!

Ez a demó kizárólag sprite-okat használ, viszont azokból majdnem rögtön az összeset. Először a képernyő bal, jobb és alsó szélén jelennek meg egymás után a kukacok fejei, majd rövid várakozás után mind a 63 megtámadja a képernyőt. A végén jobbról megérkezik a főkukac: ez használja a 64-ik sprite slotot, így ekkor már mind a 64 lehetséges sprite aktív.

Egy normál kukac 64×16 pixeles, vagyis egy animációs fázisa 512 bájt és nyolc különböző animációs fázisa van. Ezért egy teljes, adott színű animáció 4 kB-t foglal az RRAM-ban. A demó mind a 15 használható színhez elkészíti ugyanezt a nyolc fázist – a 0-s színindex továbbra is a transzparens –, ezért a normál kukacok teljes animációs tárigénye 60 kB lesz az RRAM-ban. Ez viszont nem azt jelenti, hogy a 63 sprite-nak külön saját grafikai adata van! Mind a 63 kukac ugyanazt a közös 15 féle színű animációs RRAM adatokat használja. Az egyes példányoknál csak az SPB-ben lévő RRAM kezdőcímet változtatjuk attól függően, hogy éppen melyik szín és melyik animációs fázis tartozik hozzá. Minden kukac véletlenszerűen kap egyet a 15 színből, és az animáció sebessége is eltérhet. A jobbról balra haladó kukacokhoz sincs külön tükrözött grafika eltárolva. Ugyanazt a képadatot használják, csak az SPB-ben bekapcsoljuk az Xflip transzformációt.

A 64-ik sprite-ként megjelenő főkukac ugyanezen animáció alapján készül, de az RRAM-ban már eleve négyszeres méretre nagyítva tároljuk. Egy fázis így 256×64 pixel, azaz 8192 bájt, a nyolc fázis összesen 64 kB. Megjelenítéskor ezen felül még az SPB-jének az Xdouble és Ydouble transzformációs bitjeit is bekapcsoljuk, ezért a ténylegesen látható sprite 512×128 pixeles lesz. Mivel a főkukac jobbról balra halad, így az SPB-jében az Xflip is be van kapcsolva.

Összefoglaló: a demó bemutatja a sprite-oknál használható erőforrás optimalizálást: 63 egymástól függetlenül mozgó, különböző színű, animációs sebességű és változó irányú objektumhoz nem kell feltétlen 63 külön animációs készlet. A képadatok közösek, menet közben lényegében csak az SPB-k cím-, pozíció- és transzformációs adatai változnak.

Renderidők alakulása:

jelenet	renderidő EP	renderidő TVC
hatvanhárom kukac rohangál	2,11 ms (10,55%)	2,07 ms (10,35%)
együtt a teljes csapat	4,21 ms (21,05%)	3,97 ms (19,85%)

EGG_CA (CAh / 202) – Boulder Dash by 2dfx

Ez a demó a C6-os egyszerű **TILEMAP** bemutató továbbfejlesztett változata: itt már egy, a látható képernyőnél jóval nagyobb pályán mozgunk folyamatos, pixelenkénti scrollozással. A játékos figurája középen marad, a komplett világ mozog mögötte.

A teljes pálya 160×100 darab csempéből áll, vagyis összesen 16 000 tilemap cellát tartalmaz. Mivel egy cella egyetlen bájt, maga a komplett pályaleírás 16 000 bájt RRAM-területet foglal. A csempekészlet 29 különböző 16×8 pixeles csempéből áll, a teljes csempekészlet 1856 bájt.

A grafikai objektumok valójában 2×2 db. csempéből épülnek fel, tehát egy pályaelem valójában 32×16 pixel. Így készül a föld, a fal, a tégla, a szikla, a gyémánt, a kijárat.

A komplett pálya (**DEFINE_TILEMAP**) a demó indulásakor egyszer kerül létrehozásra és feltöltésre az RRAM-ba. Scrollozás közben maga a tilemap-adat egyáltalán nem változik és nem kerül újra feltöltésre. Enterprise-on egyszerre 38×25, TVC-n 30×26 csempe alkotja a látható ablakot.

A finom scrollozás lényege az, hogy a kamera pozícióját (viewport) nemcsak csempékben, hanem pixelekben tartjuk nyilván. Az aktuális X és Y pozícióból kétféle adatot számolunk, egyrészt melyik csempe legyen a **TILEMAP** parancs első forráscellája, másrészt ezen a csempén belül hány pixellel vagyunk eltolódva.

Amikor még nem értünk el a következő 16 pixeles csempehatárig, ugyanaz marad a kezdő csempe, viszont a komplett **TILEMAP** kirajzolási X koordinátáját néhány pixellel balra toljuk. Amikor átlépjük a csempehatárt, eggyel nő a forrás tile X koordinátája, a pixelen belüli eltolás pedig újra indul. Ugyanez történik függőleges irányban is, csak ott a csempe magassága 8 pixel. Van itt egy apró trükk: ha a tilemapet például 15 pixellel balra eltoljuk, a jobb szélen már belóg a következő csempe. Ezért a demó mindig egy plusz oszlopot és egy plusz sort is kirajzol. Enterprise esetén tehát a 38×25-ös látható területhez 39×26, TVC-n a 30×26-os területhez 31×27 csempe kerül kirakásra.

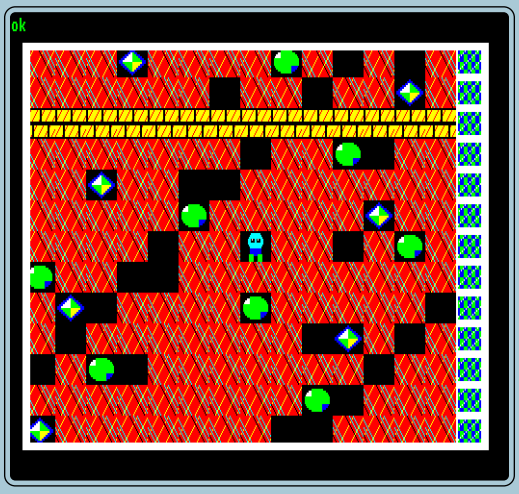
A fölöslegesen kirajzolt széleket négy 2DPT **FILL_RECT** slot maszkolja le a széleken, majd egy **RECT** slot rajzol keretet a látható ablak köré. Így kívülről csak a fix méretű viewportot látjuk, miközben mögötte a **TILEMAP** egy kicsit nagyobb területet rajzol és azon belül mozog.

A játékos egy külön, 32×16 pixeles sprite, mérete 256 bájt, és végig ugyanazon a képernyőpozíción marad; a mozgás látszatát valójában a mögötte scrollozó tilemap adja.

A demó minden 50 Hz-es frissítéskor újraépíti a háttér 2DPT-listáját, benne az aktuális **TILEMAP** forrás- és célkoordinátákkal. A **DEFINE_TILESET** és **DEFINE_TILEMAP** descriptorokat viszont csak egyszer, a demó indulásakor, mivel maguk az RRAM-beli adatok nem változnak. A lényeg, hogy a 16 kB-os pályaadathoz scrollozás közben egyáltalán nem kell hozzányúlni.

A kamera útvonala szándékosan a pálya jobb és alsó széle közelébe is elmegy, így az is látható, hogy ha a **TILEMAP**-et úgy állítjuk be, hogy az eredeti térképből kilógna a rajzolt terület, a helyén nem a pálya másik oldala vagy valamiféle szemetelés látszik, az MCU ezt a "kilógást" helyesen kezeli.

Összefoglaló: a demó legfontosabb tanulsága, hogy egy nagy tilemap finom scrollozásához nem kell magát a pályaadatot mozgatni vagy újratölteni. Elég a **`TILEMAP`** parancs forráscelláját és képernyőpozícióját módosítani, valamint egy plusz sort és oszlopot kirajzolni a széleken. Egy több ezer cellás pálya így végig változatlanul az RRAM-ban maradhat anélkül, hogy az Enterprise vagy TVC oldaláról ezt újra kellene tölteni minden egyes alkalommal.



Renderidők alakulása:

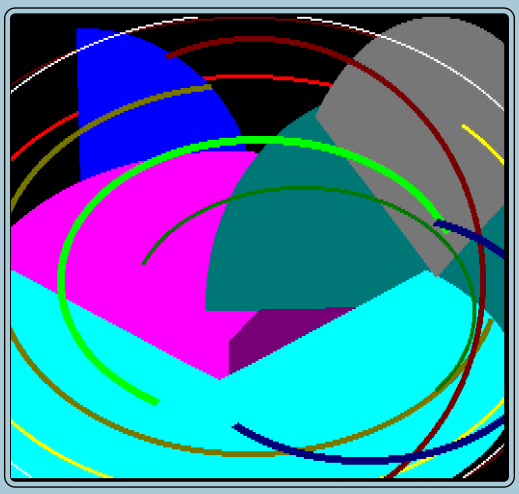
renderidő EP	renderidő TVC
1,64 ms (8,2%)	1,46 ms (7,3%)

EGG_CB (CBh / 203) – render overrun bemutató

Ez a demó szándékosan olyan 2DPT-t állít össze, amelynek renderelése 50 fps módban bőven túllépi a rendelkezésre álló 20 ms időkeretet. Ehhez két nagy **ELLIPSE** mellett hat kitöltött **PIE** és nyolc, különböző vonalvastagságú **ARC** primitívet rajzol egymásra – a két utóbbi a legszámításigényesebb primitív.

A cél elsősorban nem az **ARC** és **PIE** bemutatása, hanem annak demonstrálása, mi történik akkor, ha a renderer nem készül el időben egy képkockával.

Render overrun esetén a státuszregiszter ötödik bitje (**AND 20h**) nullába vált, és ebben az állapotban marad mindaddig, amíg a **CLEAR_RENDER_OVERRUN** paranccsal nem töröljük. Ha közben elérkezik a következő képfrissítés ideje, miközben az előző renderelés még folyamatban van, a már elkészült előző framebuffer marad a képernyőn.

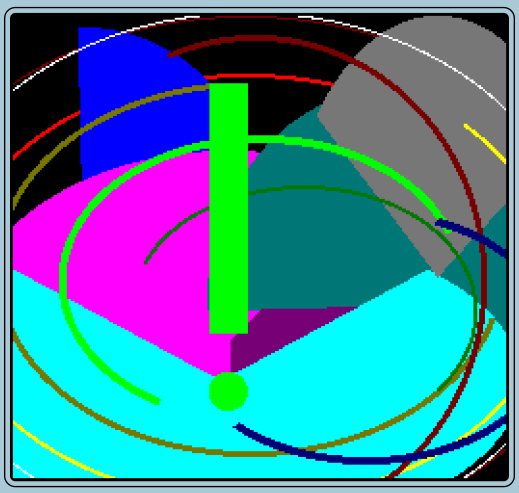


A CB demóval egyszerűen kipróbálható a **SET_RENDER_OVERRUN_VISUAL** parancs is a gyakorlatban.

A demó indítása után kapcsoljuk be a vizuális jelzést:

```
OUT 248, 82
OUT 249, 1
```

A képernyő közepén egy villogó, színét változtató felkiáltójel jelenik meg a normál grafikai rétegek fölött.



A sticky overrun állapot ekkor ugyan törölhető a **CLEAR_RENDER_OVERRUN** paranccsal, de mivel a demó fut és ugyanazt a képet rajzolja ki megállás nélkül, a következő képkockában újra megjelenik a felkiáltójel.

Renderidők alakulása:

renderidő EP	renderidő TVC
43,1 ms (215,5%)	43,1 ms (215,5%)

EGG_CC (CCh / 204) – diagnosztikai tesztek

Ez a demó öt, egyenként körülbelül 20 másodpercig futó diagnosztikai tesztből áll. Ezek nem látványdemók, hanem a renderer, az RRAM címzés, a sprite-transzformációk, a vágás (clipping) és a különböző resource-típusok helyes működésének ellenőrzésére szolgálnak. Itt a renderidőt nem jelölöm külön, ugyanis nincs jelentősége.

23 bites RRAM-címzés és cache ellenőrzése – az első jelenet különböző, szándékosan egymástól távoli RRAM-címeken elhelyezett sprite-okat jelenít meg, többek között a **000000h**, **010000h**, **100000h**, **200000h**, **400000h** és **7FF000h** tartományokból. Emellett több sprite-nál az RRAM-cím alsó bitjei azonosak, csak a felső címrészek különböznek. Ez annak ellenőrzésére szolgál, hogy a renderer és a belső sprite-cache valóban a teljes 23 bites címet használja, és nem keveri össze az egymástól akár megabájtokra lévő erőforrásokat.

Sprite transzformációk – ebben a jelenetben ugyanazt az irányjelző sprite-ot jeleníti meg tíz példányban, a C1-ben is használt X/Y double, X/Y flip és kombinált transzformációkkal. Mivel a grafika aszimmetrikus, bármilyen hibás tükrözés vagy nagyítás azonnal látható.

Vágás (clipping) – a harmadik szakasz egy 96×64 pixeles sprite-ot helyez részben a renderelhető terület bal, jobb, felső és alsó szélén kívülre, illetve egy Xdouble változatot is használ. A cél annak ellenőrzése, hogy a sprite clipping a normál és transzformált objektumoknál is helyesen működik-e, és a renderer nem olvas az objektumhoz tartozó RRAM területen kívülről.

Erőforrások kezelése – itt egyszerre működik **PATTERN_FILL**, **TILEMAP**, két **BLIT** és két sprite, amelyek közül az egyik X+Y double transzformációt kap. Arra szolgál, hogy egyetlen képkockában ellenőrizhessük a különböző resource descriptorok és renderelési módok egymás melletti működését.

ZX1 kitömörítés – az utolsó teszt egy ZX1-tömörített sprite-adatot tölt fel az RRAM egyik területére, a célterületet előtte nullázza, majd elindítja az **UNZX1_RAW** műveletet. A sprite SPB-je már eleve a kitömörítés célcímére mutat, ezért a kép akkor jelenik meg, amikor az MCU core1-en futó kitömörítési művelet elkészült. Ez egyszerre ellenőrzi a ZX1 kitömörítést és azt is, hogy a core1 által előállított adat helyesen válik láthatóvá a renderer számára.

EGG_CD (CDh / 205) – teljes képernyős BLIT

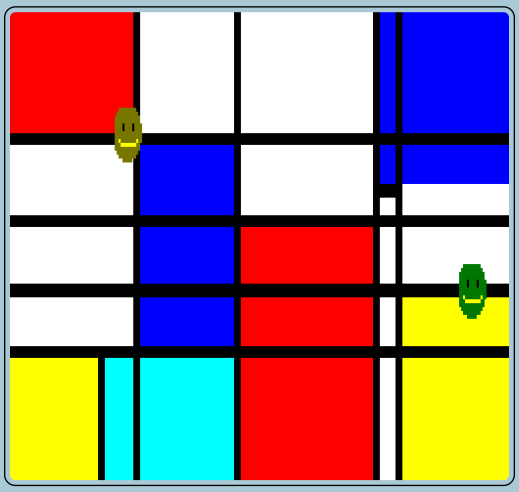
Ez a demó a **BLIT** renderelési sebességének mérésére készült. Egyetlen bitmap tölti ki szinte a teljes 2dfx renderterületet: Enterprise-on 744×288, TVC-n 704×256 pixel méretben. Ez Enterprise-on 107136 bájt, TVC-n 90112 bájt képadatot jelent.

A Mondrian-szerű bitmapet a demó induláskor egyszer generálja le és tölti fel az RRAM-ba. A kép nem használja a nullás, transzparens színindexet. Egy **DEFINE_BITMAP** parancs után egyetlen 2DPT **BLIT** parancsot ad ki. Menet közben sem maga a bitmap, sem a **BLIT** parancs nem változik.

A demó két, egyenként húsz másodperces jelenetből áll, amelyek váltogatják egymást:

Teljes képernyős **BLIT** – ebben a részben csak a nagy bitmap **BLIT**-je terheli a renderert, így közvetlenül mérhető egy közel teljes framebuffer méretű bitmap másolásának ideje.

Teljes képernyős **BLIT** + két sprite – itt a teljes képernyős **BLIT** változatlanul megmarad, de mellé megjelenik két 16×16 pixeles smiley sprite. Mindkettőn be van kapcsolva az Xdouble és Ydouble transzformáció, így 32×32 pixel méretben jelennek meg. Ezzel jól mérhető, mennyivel növeli meg a renderidőt a két sprite kirakása a már eleve nagy **BLIT** fölött.



Renderidők alakulása:

jelenet	renderidő EP	renderidő TVC
teljes képernyős BLIT	2,22 ms (11,1%)	1,86 ms (9,3%)
teljes képernyős BLIT + két sprite	2,29 ms (11,45%)	1,94 ms (9,7%)

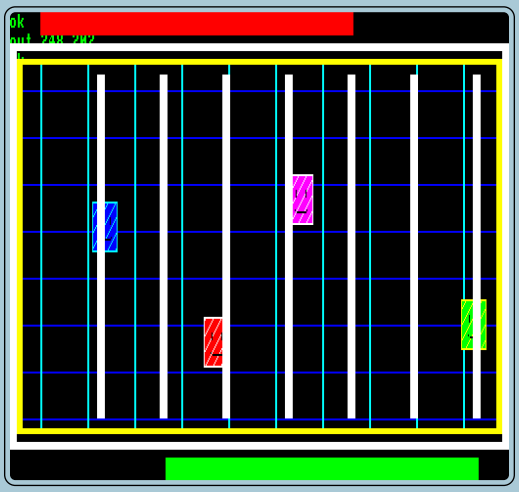
EGG_CE (CEh / 206) – A börtön

Ez a demó a 2dfx teljes rétegsorrendjét mutatja be: 2DPT back, sprite-ok, 2DPT front

A képernyőn négy darab 32×32 pixeles börtönlakó, ámde mosolygós sprite pattog. Mögöttük a 2DPT back rajzolja ki a rácsot és a keretet, előttük pedig a 2DPT frontban lévő függőleges rácsrudak, külső keret és két HUD-sáv jelenik meg.

A négy sprite képadatát a demó induláskor egyszer tölti fel az RRAM-ba. Menet közben csak az SPB-k X/Y koordinátái változnak, ahogy a sprite-ok pattognak a képernyőn. A 2DPT back szintén egyszer készül el, mivel a sprite-ok mögött látható rács statikus. A 2DPT front viszont minden képfrissítéskor újraépül, mert a függőleges rácsrudak folyamatosan mozognak. Amikor egy sprite egy ilyen rúd mögé kerül, az 2DPT front egyszerűen felülrajzolja, ugyanakkor a háttér rácsvonalai (2DPT back) mindig a sprite-ok mögött maradnak.

Összefoglaló: a demó megmutatja, hogy a sprite-ok fölé rajzolt grafikai elemekhez nincs szükség külön sprite-okra vagy a sprite-képek módosítására. Egy játékban ugyanilyen módon készülhet például HUD, ablakkeret, kapu, rács, előtérben lévő tereptárgy vagy bármilyen más maszkoló grafika.



Renderidők alakulása:

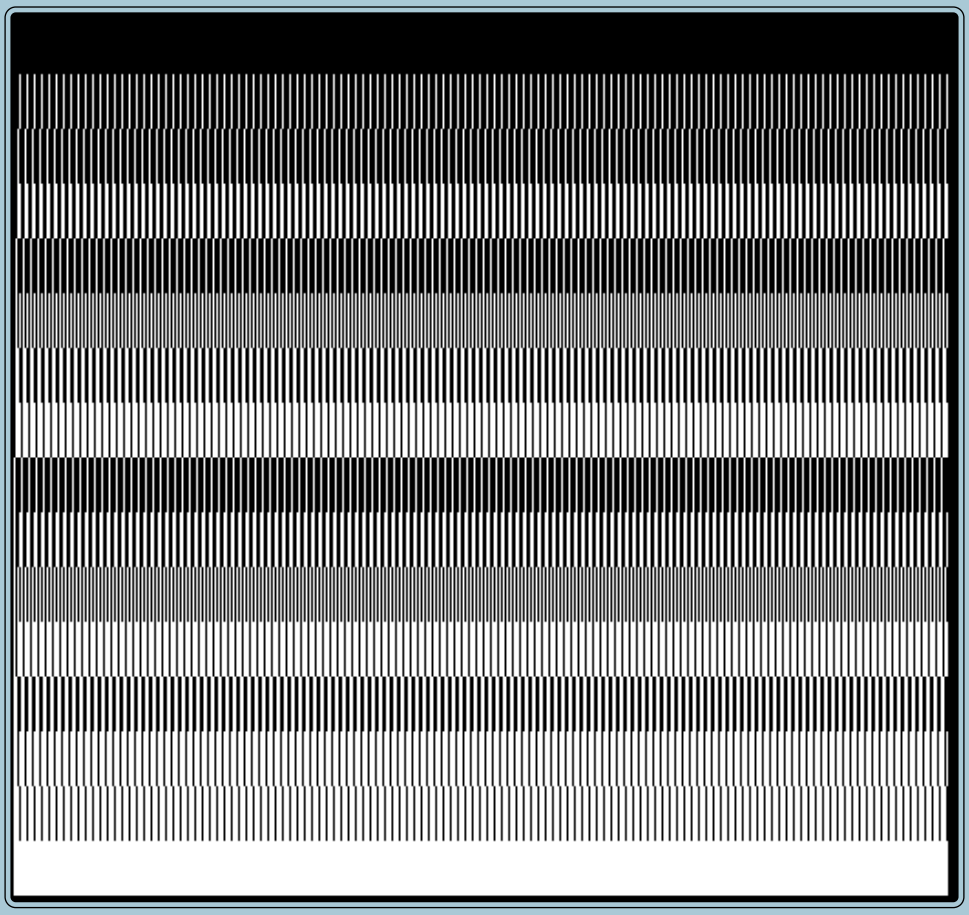
renderidő EP	renderidő TVC
1,04 ms (5,2%)	1 ms (5%)

EGG_CF (CFh / 207) – TVC külső szín mintavételi teszt

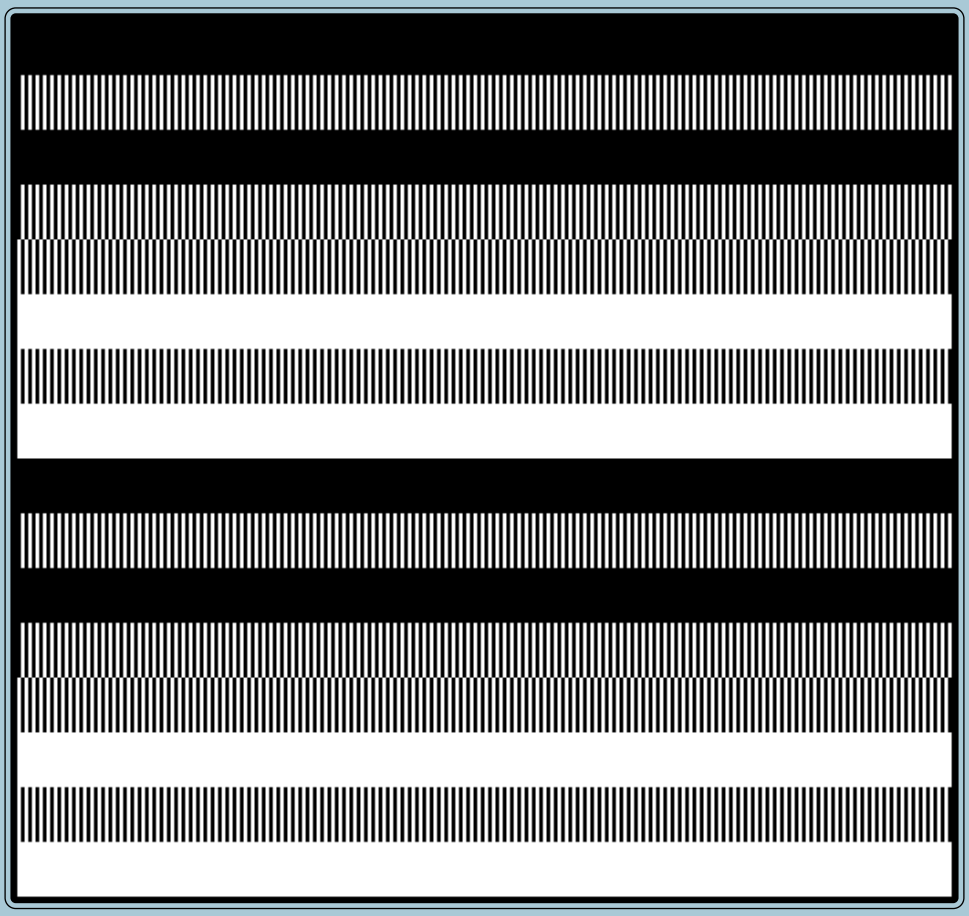
Ez a demó kizárólag TVC-n használható, és a TVC külső színbemenetei feldolgozásának fontos korlátját mutatja be, a renderidő itt nem fontos.

A 2dfx a TVC felé 12,5 MHz-es órajellel változtatja az EC0..EC3 külső színjeleket, ezzel érve el a teljes lehetséges vízszintes felbontást. A TVC viszont ezt a teljes sebességet csak GRAPHICS 2 módban mintavételezi. GRAPHICS 4 és GRAPHICS 16 módban a videorendszer az EC0..EC3 bemeneteket csak fele ilyen sűrűn, vagyis minden második 2dfx pixelnél veszi figyelembe.

A demó 16 vízszintes sávot jelenít meg, amelyek négy pixelenként eltérő bitmintát mutatnak 0-15 között, sávonként módosítva magát a bitmintát. Az első csíkban minden pixel 0 értékű, a második csíkban három 0 értékű és egy 15 értékű pixelcsoport követi egymást, és így tovább. GRAPHICS 2 módban mind a 16 sáv helyesen és egymástól jól megkülönböztethetően látható.



Az egyéb videomódokban viszont jól látszik a felezett mintavétel következménye. Egy olyan gyorsan váltakozó minta, mint például az 1010, a 2dfx oldaláról helyesen kerül kiküldésre, a TVC viszont csak minden második pixelt mintavételezi. Emiatt bizonyos sávok eredeti mintázata teljesen eltűnik, és például egy váltakozó 0101 minta tömör fehér 1111 blokknak látszódik.



A 604×240 pixeles tesztkép egy bitmapként készül el az RRAM-ban, majd egyetlen 2DPT **`BLIT`** paranccsal kerül a képernyőre.

A demóból világosan érzékelhető, miért kell 2dfx használatakor TVC-n GRAPHICS 2 módot választani, ha a maximális vízszintes felbontást szeretnénk kihasználni.

EGG_D0 (D0h / 208) – TVC képfrissítési stressztest

Ez egy egyszerű, kizárólag TVC-n használható stressztest, amely szándékosan minden egyes 50 Hz-es képfrissítésnél nagy területen változtatja meg a képernyő tartalmát, a renderelési idő irreleváns.

Két teszt váltakozik benne: az egyiknél a teljes 704×256 pixeles renderterület két eltérő szín között villog, a másiknál pedig egy fekete és fehér terület közötti függőleges határ mozog nagy sebességgel jobbra-balra.

A célom egyszerűen az volt, hogy egymást követő képkockák között minél látványosabb különbség legyen, így bármilyen képkockakeveredés vagy egyéb megjelenítési, illetve szinkronizációs probléma azonnal észrevehetővé váljon.

EGG_D1 (D1h / 209) – a látható képernyőterület koordinátái

Ez a demó a 2dfx X/Y koordinátarendszerének és a ténylegesen látható képernyőterületnek az összehasonlítására készült.

Először egy egy pixel széles függőleges vonal halad végig a teljes X tartományon, majd ugyanez visszafelé. Ezután egy vízszintes vonal járja végig az Y koordinátákat oda-vissza. A világos mérővonal mellett szomszédos vonalak segítik, hogy a pozíció televízión vagy más megjelenítőn könnyebben követhető legyen.

A képernyő közepén egy négyszámjegyű kijelző folyamatosan mutatja az éppen vizsgált pontos koordinátát a 2dfx koordinátarendszerében.

Minden koordináta négy képkockán keresztül marad a képernyőn, a végpontokon pedig rövid szünet van, így megfigyelhető, hogy egy adott X vagy Y érték hol jelenik meg, illetve mikor tűnik el a képernyőről.

A demó segítségével az Enterprise-on vagy a TVC-n egyénileg programozott látható képernyőterület a felhasználói program tervezésekor megmutatható, így a 2dfx számára már olyan grafikai koordinátákat tudunk megadni rajzoláskor, amelyek ténylegesen a látható képtartományon belül (vagy éppen kívül) vannak. Ezt a demót használtam a kézikönyv **Geometria** fejezetében megtalálható pontos látható koordináták rajzaihoz.

EGG_D2 (D2h / 210) – animált 2DPT és SET_XY_OFFSET bemutató

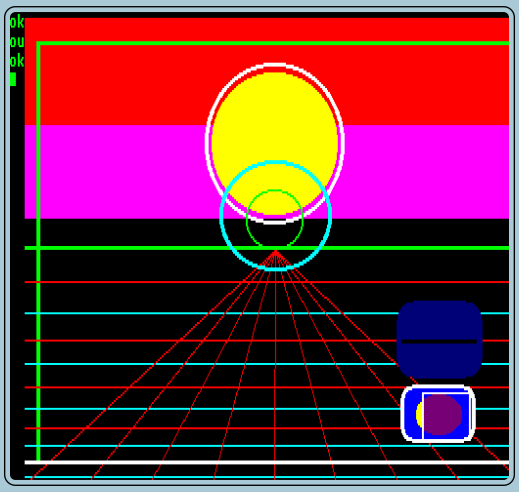
Ez a demó a C2-es 2DPT primitívek bemutatóját változatlan formában használja, azzal a különbséggel, hogy a teljes grafika folyamatosan mozog a képernyőn.

A 2DPT első eleme itt egy **SET_XY_OFFSET** parancs. Az ezt követő valamennyi pozíciófüggő 2DPT slot (vonalak, téglalapok, ellipszisek, kitöltések, stb.) automatikusan megkapja ugyanazt az X/Y eltolást. Maguknak az egyes grafikai parancsoknak a koordinátáit tehát nem módosítjuk.

Az X és Y offset egymástól függetlenül változik: vízszintesen +-24 pixel, függőlegesen +-10 sor közötti tartományban. A C2-ben látható belső animációk (például a mozgó vízszintes vonal, a változó méretű körök és az alsó mozgó téglalap) közben ugyanúgy működnek tovább. A **SET_XY_OFFSET** ezekre csak egy további közös eltolást tesz rá.

A C2 és D2 renderideje közvetlenül összehasonlítható, hiszen a két demó grafikai tartalma azonos; a D2-ben ehhez mindössze a **SET_XY_OFFSET** működése adódik hozzá. A táblázatból jól látható, hogy a **SET_XY_OFFSET** a renderidőhöz gyakorlatilag nem ad hozzá, szinte nulla többletidőt jelent a használata.

Összefoglaló: a **SET_XY_OFFSET** funkció jól használható például teljes képernyőrész vagy HUD mozgatására, kamera-rázásra, átmenetekre, illetve bármilyen olyan helyzetben, amikor több 2DPT elemet szeretnénk egyetlen közös koordináta-rendszerben mozgatni.



Renderidők alakulása a D2 demóban:

paraméter	renderidő EP	renderidő TVC
minimum	0,96 ms (4,8%)	0,87 ms (4,35%)
maximum	1,2 ms (6%)	1,11 ms (5,55%)

Renderidők alakulása a C2 demóban:

paraméter	renderidő EP	renderidő TVC
minimum	0,95 ms (4,75%)	0,87 ms (4,35%)
maximum	1,18 ms (5,9%)	1,11 ms (5,55%)

a hardvert tervezte: Vaczkó Károly
a szoftverachitektúrát, parancsinterfészt tervezte: Vaczkó Károly
az MCU-ban futó kódot a ChatGPT 5.5 Plus, majd a ChatGPT 5.6 Sol készítette, valamint
számtalan további ellenőrzésen ment át az Anthropic Fable 5 által
a kézikönyvet írta: Vaczkó Károly

2026. augusztus