Valósidejű háromdimenziós grafika beágyazott környezetben

SZÁNTÓ PÉTER

BME, Villamosmérnöki és Informatikai Kar, Méréstechnika és Információs Rendszerek Tanszék
szanto@mit.bme.hu

Reviewed

Kulcsszavak: 3D megjelenítés, szegmens alapú feldolgozás, FPGA, System-On-Chip

A beágyazott környezetekkel szemben támasztott tipikus követelmények (alacsony ár, kis fogyasztás) következményeként minél hatékonyabb architektúrák kialakítására van szükség, s ez természetesen igaz a megjelenítést végző hardverre is. Jelen cikk egy lehetséges módszert mutat be a háromdimenziós grafikai megjelenítés hatékonyságának növelésére, a szükséges memória sávszélesség csökkentésére.

1. Bevezetés

A felhasználói igények hatására a közeljövőben várhatóan számos újabb helyen merül fel valósidejű és valósághű háromdimenziós grafikai megjelenítés igénye. Ilyen eszközök lehetnek például az egyre nagyobb felbontású és színmélységű megjelenítővel rendelkező mobiltelefonok és digitális személyi asszisztensek (PDA), önálló, TV-vel összeköthető digitális eszközök (set-top-box) vagy akár járművek fedélzeti számítógépei. A 3D megjelenítés hatalmas számításigénye miatt ezekben a viszonylag alacsony – bár rohamosan növekvő – teljesítményű processzorral rendelkező rendszerekben a szoftveres megoldások hamar teljesítőképességük határára érnek; a komplex, valós idejű alkalmazások esetében dedikált, hardveres egységre van szükség.

Azonban nem a CPU teljesítménye jelenti az egyetlen korlátot: a 3D grafika szempontjából a megengedett fogyasztás és az ezzel szorosan összefüggő külső memória sávszélesség szűkös volta talán a legnagyobb megszorítás. Míg asztali számítógépekbe szánt grafikus egységeknél nem ritka a többszáz MHz-es órajelen működő, igen széles memória buszrendszer sem, addig beágyazott rendszerek esetében ennek töredéke áll rendelkezésre a teljes rendszer számára.

Igen fontos tehát egyrészről a rendelkezésre álló erőforrások optimális kihasználása, másrészről pedig egy jól skálázható architektúra kialakítása – hiszen a különböző területre szánt rendszerek teljesítményigénye meglehetősen különböző lehet. További kritériumként merülhet fel a jelenleg elterjedt szoftveres fejlesztői módszerekkel, lehetőségekkel történő kompatíbilitás megtartása, amely az alkalmazások hatékony és gyors adaptációját segíti elő.

2. A 3D megjelenítés alapjai

A 3D megjelenítés alapeleme – akár valós idejű, akár előre renderelt – a háromszög. A virtuális világot leíró modellek saját koordinátarendszerükben, háromszöghálóval közelítve adottak. A felhasznált háromszögek számát a modellezni kívánt objektum bonyolultsága, illetve a megjelenítés finomsága szabja meg: egyszerű esetben (mint például egy kocka) igen kisszámú háromszög felhasználásával tökéletes eredmény érhető el, a bonyolultabb felületek közelítése azonban igen nagyszámú háromszöget is igényelhet.

A megjelenítés [1,2] első lépése – a *transzformáció* – pozíciójuknak és orientációjuknak megfelelően elhelyezi a virtuális világ koordinátarendszerében a lokális koordinátarendszerben definiált objektumokat, majd a kamera helyének és látószögének függvényében a teret a képernyő koordinátarendszerébe transzformálja (a kamera az origóba kerül és a pozitív Z irányba néz). Ugyancsak a csúcspontokon végzendő műveletek közé sorolható a csúcspont alapú *megvilágítás*: ennek során a világban adott fényforrások hatását a háromszögek csúcspontjaira határozzuk meg.

A transzformációt és megvilágítást a *raszterizáció* követi, melynek során a háromszögeket a képernyőt alkotó pixelekre képezzük le, azaz két dimenzióban egyenletesen mintavételezzük.

A *láthatósági vizsgálat* során minden egyes képernyő-pixelre meghatározzuk az azon látható háromszöget, tehát azt, amelyik az adott képernyő pontban a kamerához legközelebb található.

A legutolsó lépés a pixelek színének meghatározása, az *árnyalás* (shading). Ehhez egyrészt felhasználhatók a csúcspont alapú megvilágításnál kiszámított szín értékek: a háromszög belső pontjaiban érvényes szín például a csúcspontokban adott értékek lineáris interpolációjával állítható elő (Gouraud shading). A megjelenítés részletgazdagsága textúrák alkalmazásával tovább növelhető. A legegyszerűbb esetben a textúra az objektum felületi mintájának „fényképe”, amelyet mintegy ráfeszítünk az objektumot meghatározó háromszög vázra. Általánosabban a textúra bármely felületi jellemzőt tároló egy-, két- vagy háromdimenziós tömb, melyet úgy rendelünk a háromszöghöz, hogy annak csúcspontjaiban megadjuk az ott érvényes textúra koordinátákat.

3. Szegmens alapú feldolgozás

Tipikus hardveres megjelenítő egységekben (úgynevezett Immediate Mode Renderer (IMR) megjelenítők) a feldolgozás háromszögről háromszögre haladva történik, a láthatósági vizsgálathoz pedig a Z-buffer algoritmust alkalmazzák. Ennek lényege, hogy minden egyes képernyő pixelhez tartozik egy elem az úgynevezett Z-bufferben (vagy mélységi bufferben), amely az adott időpillanatig feldolgozott, és az adott pixelt fedő háromszögek mélységi (Z) koordinátája közül a minimálist tartalmazza. A Z-buffer – méreténél fogva – a külső memóriában helyezkedik el. Újabb háromszög feldolgozásakor a háromszöget fedő összes pixelre meghatározzuk annak színét, valamint Z értékét (utóbbi is lineárisan interpolálható a csúcspontokban adott Z koordinátákból). A kiszámított mélységi értéket összevetve a Z-bufferben található megfelelő értékkel megállapítható, hogy az új háromszög közelebb van-e a kamerához, mint az eddig feldolgozottak. Amennyiben igen, úgy a Z-buffert és a pixel színeket tároló buffert is frissíteni kell az új adatokkal.

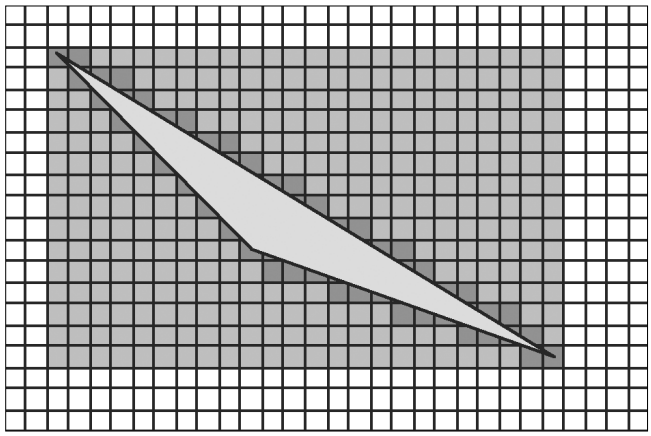
A vázolt algoritmusnak két alapvető problémája van. Egyrészt rendkívül sok memória művelettel jár a pixelenkénti Z-buffer olvasások és esetleges Z-buffer és szín-buffer írások miatt. Másrészt sok felesleges számítást igényelhet, hiszen ha a képet alkotó háromszögek közül a kamerához közelebb levők később érkeznek, akkor sok, már kiszámított pixel színét felülírják. A bemutatott egyszerű algoritmus hatásfoka természetesen nagyban növelhető különböző optimalizációkkal (például Early Z Test, ATI HyperZ [6]), ám ezek továbbra is nagyban függnek a háromszögek feldolgozási sorrendjétől.

Ezzel ellentétben a szegmentált feldolgozás [3] sorrend-független előnyökkel jár. A módszer alapja a kép feldarabolása kis téglalapokra (szegmensek), majd e téglalapok egymástól független feldolgozása. A szegmensek kis mérete lehetővé teszi a Z-buffer áramkörön belül történő megvalósítását, ami egyrészt a külső memóriához fordulások számát jelentősen csökkenti, másrészt igen nagy feldolgozási sebesség elérését teszi lehetővé, hiszen IC-n belül igen széles adatbuszok alakíthatók ki. Nagy teljesítményt igénylő vagy elosztott rendszerek esetén további előny hogy a szegmensek egymástól függetlenül, párhuzamosan feldolgozhatók.

A láthatósági vizsgálat árnyalás előtti elvégzése lehetővé teszi az árnyaló egység hatékony kihasználását, hiszen csak a valóban látható pixelek színét kell meghatározni (a sorrend ilyetén megválasztására IMR esetben is van lehetőség, ami növeli a hatékonyságot, de nem garantálja a 100%-t).

Természetesen e módszernek is vannak hátrányai. A kép első szegmensének feldolgozásakor már rendelkeznünk kell a képet alkotó összes transzformált háromszöggel, hiszen csak így határozható meg biztosan a látható objektum – ez pedig egy képidőnyi késleltetést jelent. Ezen kívül a hatékonyság megőrzéséhez szükség van egy, az IMR megoldásoknál szükségtelen hardver modulra.

Képzeljünk el ugyanis egy kicsi (néhány szegmens nagyságú) háromszöget. Ha ez minden egyes, a képet alkotó több száz szegmensben feldolgozásra kerül, az jelentős mennyiségű felesleges munkát jelent. Célszerű tehát a raszterizáció előtt egy új feldolgozási lépést beiktatni, amely minden egyes szegmensre meghatározza az abban legalább egy pixelt lefedő háromszögeket – így később, a szegmensek feldolgozásakor már csak ezekkel kell foglalkozni. A szegmentálás során két különböző megközelítést alkalmazhatunk: megelégedhetünk a fedett szegmensek körülbelüli meghatározásával, vagy minden háromszögre pontosan meghatározhatjuk a fedett szegmenseket.



1. ábra Szegmentálási módszerek

Előbbi megvalósítható például a háromszög befoglaló téglalapjának felhasználásával; azonban mint az *1. ábrán* látható, aránytalan háromszögek esetén ez a módszer igen rosszul működik (az ábrán például a befoglaló téglalap 370 szegmenst tartalmaz, míg a háromszög csupán 76-ban fed pixeleket).

4. Belső pontok megállapítása

Mielőtt a hardver implementáció részleteivel foglalkoznánk, érdemes a fedés megállapításának kérdéséről szólni, hiszen ez több egység esetében felmerül. A cél tehát egy háromszög belső pontjainak meghatározása.

Definiáljunk ehhez minden egyes háromszög oldalhoz egy, az oldal explicit egyenletéből képzett változót:

$$A(x,y) = (x - x_i) * \Delta y - (y - y_i) * \Delta x \qquad (1)$$

Ez a változó 0 értékű az egyenesen, negatív az egyik és pozitív az egyenes által meghatározott másik félsíkon.

A *2. ábra* a három oldal-változó előjelét mutatja abban az esetben, amikor a csúcspontok *y* koordinátájuk szerint növekvő sorrendbe vannak rendezve, és az (1)-ben szereplő Δ értékeket úgy képezzük, hogy a nagyobb sorszámú csúcspont *x/y* koordinátájából vonjuk ki a kisebb sorszámúét. Ekkor a belső pontokra teljesül az alábbi kifejezés:

$$(s(A_0(x,y)) \; XOR \; s(A_1(x,y))) \; AND \; (s(A_1(x,y)) \; XOR \; s(A_2(x,y))), \quad (2)$$

ahol *s(A(x,y))* az adott oldal *A(x,y)* változójának előjelét jelenti (0 nem-negatív esetben, 1 egyébként). Jól látható az is, hogy a képen *x* irányban lépve az *A(x,y)* változó Δy, míg y irányban lépve Δx mértékben változik.

5. Hardver architektúra

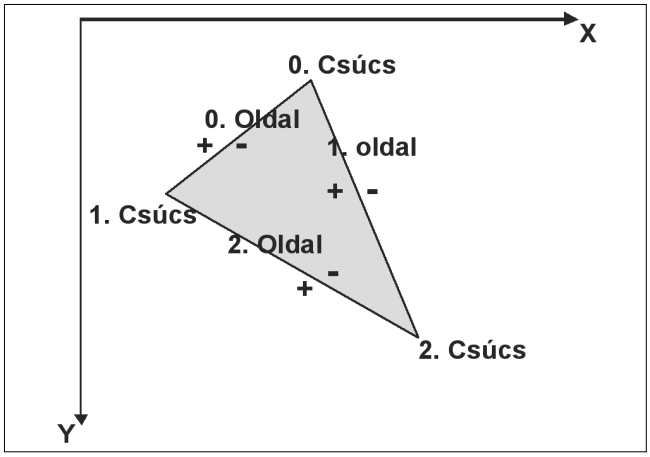
A *3. ábra* egy lehetséges, egyetlen áramkörön belüli System-On-Chip (SOC) rendszer vázlatát mutatja. Ennek része a központi vezérlő szerepét betöltő mikrokontroller, a külső memóriával kapcsolatot tartó memóriavezérlő és a megjelenítésért felelős egység. Természetesen lehetséges a processzor buszra egyéb perifériákat is kapcsolni. A megjelenítő egység ismertetett moduljai a fejlesztés során egy Xilinx XC2V6000 FPGA-ban kerültek megvalósításra, melynek részleteit a hatodik fejezet ismerteti.

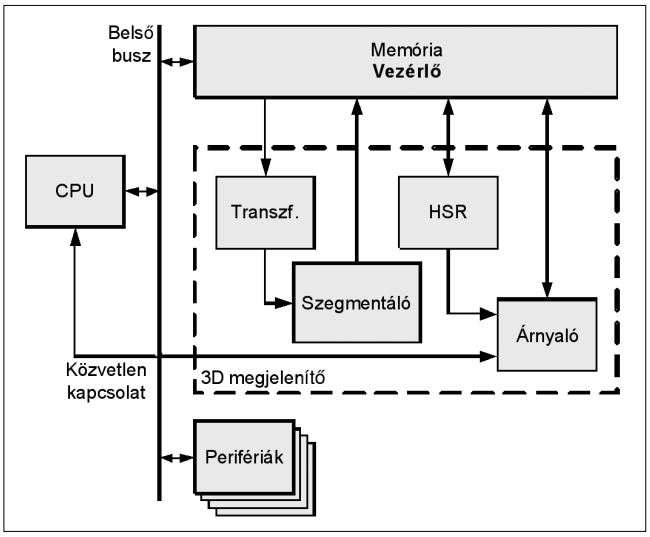
Maga a megjelenítő négy fő részből áll, melyek az egyes megjelenítési lépések elvégzéséért felelősek: a transzformációs *(Transzf)* egység a csúcspontokkal kapcsolatos műveleteket, a *Szegmentáló* a háromszögek szegmensekbe osztását, a *HSR* (Hidden Surface Removal) egység a láthatósági vizsgálatot, míg az *Árnyaló* a pixelek színének kiszámítását végzi.

5.1. Szegmentáló egység

A feldolgozási sorrendben ez az egység közvetlenül a transzformáció után következik. Bemenetei a transzformált csúcspontok *x* és *y* koordinátái, míg kimenete minden szegmenshez egy lista, amely az adott szegmensben levő háromszögek sorszámát tárolja. A kimeneti lista nagysága a megjeleníteni kívánt kép bonyolultságától függ, viszonylag nagy mérete miatt külső memóriában kapott helyet, így fontos kritérium hogy a feldolgozási láncban következő, láthatósági vizsgálatot végző HSR egység részéről ez a lista minél egyszerűbben feldolgozható legyen.

2. ábra Belső pontok megállapítása





3. ábra SOC rendszer

A hatékonyság maximalizálása érdekében a szegmens mérete képkockák között változtatható, ennek kezelésére mind a szegmentáló, mind pedig a HSR egység képes. A minimális szegmens méret 32x16 pixel, mind vízszintes, mind pedig függőleges irányban ezen méret többszörösei választhatók, egymástól függetlenül.

A szegmentáló három részből áll. A bemeneti fokozat végzi a bemeneti *x, y* koordinátákból a szegmens generátor számára szükséges adatok kiszámítását (Δ értékek, *A(x,y)* változók kezdeti értéke). A szegmens generátor egység a háromszög által fedett szegmenseken lépdel végig, s órajelenként generál egy érvényes szegmens koordináta kimenetet. A kimeneti fokozat a szegmens generátor kimeneti adatait felhasználva memóriacímeket és vezérlőjeleket állít elő a kimeneti lista felépítéséhez.

5.1.1. Bemeneti Fokozat

A bementi egység két nagyobb feldolgozó láncra bontható. A transzformációs egységgel történő kiegyensúlyozás megvalósítására a bemeneti adatok (háromszög csúcspont *x, y* koordináták) egy FIFO-ba kerülnek.

Az első feldolgozó lánc ezekből a csúcspontokból generál érvényes háromszögeket: ehhez 1, 2 vagy 3 új csúcspontra van szükség (például diszkrét háromszögek esetén új háromszöget három új csúcspont határoz meg, míg háromszög-szalag esetén elegendő egyetlen új csúcspont). Az érvényes háromszögek csúcspontjait *y* koordinátájuk szerint sorba rendezi.

A második feldolgozó lánc egyrészt az (1)-ben szereplő Δ értékeket határozza meg, másrészt kiszámítja a három oldalegyenes *A(x,y)* változóit a kezdő szegmens (amelyikben a háromszög 0. csúcspontja van) két felső csúcspontjában. A szegmens határok a pixel középpontok között találhatók. A pipeline 9 fokozata a következő funkciókat valósítja meg:

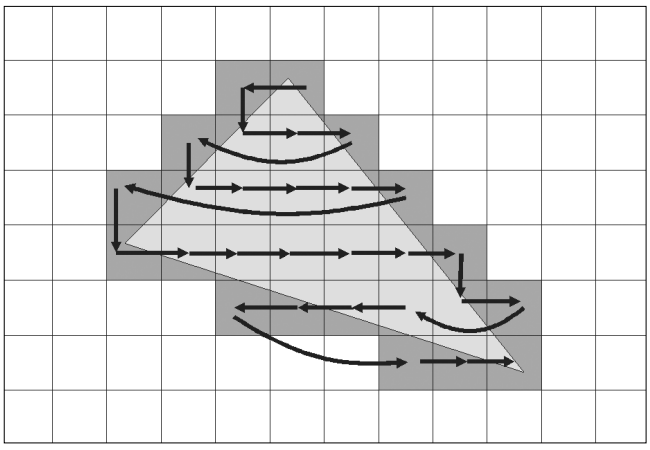
– bemenet multiplexálása;
– csúcspont szegmensének meghatározása (2 fokozat);

– háromszög-csúcspontok távolságának meghatározása a szegmens bal felső csúcsától, (1)-ben szereplő Δ értékek kiszámítása;
– (1)-ben szereplő részszorzatok meghatározása (2 fokozat);
– Δ értékek szorzása szegmens mérettel, *A(x,y)* változók meghatározása a bal felső szegmens csúcspontban;
– *A(x,y)* változók kiszámítása a jobb felső szegmens csúcspontban.

A hardver erőforrások csökkentésének érdekében a pipeline egyszerre egy oldal adatait képes feldolgozni (ezért került multiplexer az első pipeline fokozatba), tehát egy háromszög összes adatának előállítása három órajelet vesz igénybe. A szorzás műveletek az órajel frekvencia maximalizálása érdekében két ütem alatt történnek.

5.1.2. Szegmens generátor

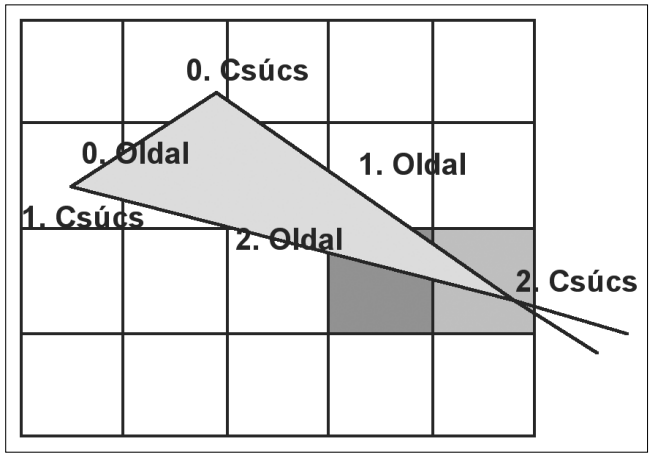
A fedett szegmensek koordinátáit előállító egység végiglépdel a feldolgozás alatt levő háromszög által fedett szegmenseken. Első megfontolásra durva felbontású raszterizáló egységnek hihetnénk, de mint látható lesz, ez sajnos nem így van. A feldolgozás a 0. háromszög-csúcspont szegmensében kezdődik, és mint minden szegmens-sorban, a jobbra lépéssel indul: az egység addig halad jobbra, amíg szükséges. A jobbra lépés befejeztével két eset lehetséges. Amennyiben a kezdő szegmenstől balra is található fedett szegmens, az algoritmus a kezdő szegmenstől eggyel balra található szegmensre ugrik, és a szükséges ideig lépdel balra. Ha jobbra lépés után nem kell ugrani, vagy befejeződött a balra lépés is, a következő szegmens-sor feldolgozása következik. Új szegmens-sorba lépésnél az algoritmus garantáltan a háromszög által fedett szegmensbe lép. Az elmondottakat szemlélteti a *4. ábra*.



4. ábra Szegmentálás

A jobbra lépések triviális esete amikor a szegmens jobb oldali csúcspontjainak egyike a háromszög belső pontja (pl. (6,4) szegmens). A további esetek figyelembevételéhez metszéspont értékeket generálunk minden szegmens-oldal és minden háromszög-oldal kombinációjával. Ezeknek előállítására ugyancsak az (1)-ben definiált *A(x,y)* értékek használhatók, hiszen egy háromszög akkor metsz egy szegmens-oldalt, ha a szegmens-oldalt meghatározó szegmens-csúcspontokban különbözik az adott oldal *A(x,y)* értékének előjele.

Az *5. ábra* a hasonló esetek egyikét mutatja. A sötétszürkével jelölt szegmensből jobbra kell lépni, annak ellenére, hogy a megfelelő szegmens-csúcspontok nem belső pontjai a háromszögnek. Ugyanakkor az 1. és 2. háromszög-oldalaknak van metszéspontja a jobb oldali szegmens-oldallal, tehát ennek alapján a lépési döntés meghozható. Ugyanezek a metszéspontok azonban a világosszürke szegmens esetében is megtalálhatók, itt mégsem szabad jobbra lépni; mégpedig azért mert az algoritmus elérte azt a szegmenst, amely tartalmazza a megfelelő (jelen esetben 2.) háromszög-csúcspontot.



5. ábra Jobbra lépés

Mint már említettük, a jobbra lépés befejezése után balra lépések következnek, amennyiben szükséges. A szükségesség eldöntéséhez a kezdő szegmens szolgál információval: amennyiben ott balra lépés is szükséges, akkor van szükség a jobbra lépés utáni visszaugrásra. A visszaugrás a kezdő szegmenstől balra található szegmensre történik, s innen folytatódik az esetleges balra lépési sorozat.

A jobbra és balra lépések végeztével a következő szegmens-sor feldolgozása kezdődik meg. Annak érdekében, hogy az algoritmus sor-lépésnél mindig biztosan háromszög által fedett szegmensbe kerüljön, a vízszintes irányú lépdelés során folyamatosan vizsgálja, hogy az adott szegmens megfelelő sor-lépési pont-e. Pozitív eredmény esetén a szegmens alsó csúcspontjaiban érvényes *A(x,y)* értékek elmentésre kerülnek, így ezek a későbbi sor-lépéshez felhasználhatók. Annak eldöntésére, hogy egy szegmens megfelelő-e sor-lépéshez, ugyancsak az *A(x,y)* értékekre van szükség: amennyiben az alsó szegmens-csúcspontok belső pontok, vagy az alsó szegmens-oldalnak metszéspontja van a megfelelő háromszög-oldalakkal, akkor a szegmens jó lépési pont. A háromszög feldolgozása akkor ér véget, amikor a legalsó háromszög-csúcspont szegmens-sorában járunk, és sem jobbra, sem pedig balra nem kell már lépni.

5.1.3. Kimeneti fokozat

A kimeneti fokozat minden egyes szegmenshez egy, a külső memóriában tárolt, 32 szavas tömbökből álló láncolt listát állít elő (szegmens lista). Egy 32 szavas blokk első 31 eleme háromszög sorszámokat tárol, míg az utolsó elem a következő 32 szavas tömb memória címét mutatja. A szegmensekhez tartozó első tömb fix címen található, míg a további tömböket dinamikusan foglaljuk, a szegmensben található háromszögek számának függvényében. A lista ilyetén kialakítása lehetővé teszi a HSR egység részéről a nagyobb egységekben történő beolvasást, de mégsem bánik túlságosan pazarlóan a memóriával.

A lista előállításához szükség van egy belső memóriára is, amely minden szegmenshez tárolja a következő (külső memória) írási címet. Új kép feldolgozásának megkezdésekor e címeknek a szegmensekhez tartozó első tömb első elemére kell mutatniuk; erről két, speciális háttér-háromszög gondoskodik, melyek feldolgozása alatt az összes cím alaphelyzetbe állítható. Ezen kívül csak a következő szabad 32 szavas blokk memória címét szükséges tárolni, amely új kép esetén a képen található szegmensek száma szorozva a tömb mérettel (32), és minden egyes új blokk foglalásakor inkrementálódik.

A külső memória foglaltsága esetén természetesen nem lehetséges az adatok kiírása, így ebben az esetben a kimeneti egység a memória felszabadulásáig leállíthatja a szegmens generátor működését.

5.2. HSR egység

A HSR egység a láthatósági és stencil tesztek elvégzéséért felelős, szegmensről szegmensre haladva dolgozza fel a képet. Az egyes szegmensekben azok a háromszögek kerülnek feldolgozásra, amelyek megtalálhatók a szegmenshez tartozó szegmens listában. A HSR egység minden háromszög esetén a szegmens összes pixelét megvizsgálja, függetlenül attól, hogy az a háromszög által fedett-e, vagy sem. Ez természetesen rontja a feldolgozás hatékonyságát, ugyanakkor determinisztikus működési időt eredményez, ami mind a cella, mind pedig a bemeneti fokozat vezérlését egyszerűsíti, valamint lehetővé teszi utóbbi egység erőforrás-igényének csökkentését.

A HSR egység egy bemeneti egységből és több, ugyanolyan felépítésű cellából áll. Alapesetben (minimális szegmens méret) a szegmens egy-egy sora van az egyes cellákhoz rendelve. A vízszintes irányú szegmens méret növekedésével ez nem változik, míg függőleges méret növekedése esetén N cella esetén minden N-edik szegmens sort az N-edik cella dolgoz fel.

5.2.1. Bemeneti egység

A bemeneti egység több műveletvégzőből áll, melyek alkalmasak a szükséges bemeneti adatok előállítására. Ezek egyrészt az Szegmentáló esetén már ismertetett, a háromszög oldalaihoz tartozó oldalegyütthatók és Δ értékek, másrészt a mélységi (Z) értékek számításához szükséges változók. A belső pontokra a Z értékeket a csúcspontokban adott értékekből lineáris interpolációval határozzuk meg, felhasználva a sík egyenletét:

$$z(x,y) = E_z * x + F_z * y + G_z = \frac{A_z}{C_z} * x + \frac{B_z}{C_z} * y + \frac{D_z}{C_z}, \qquad (3)$$

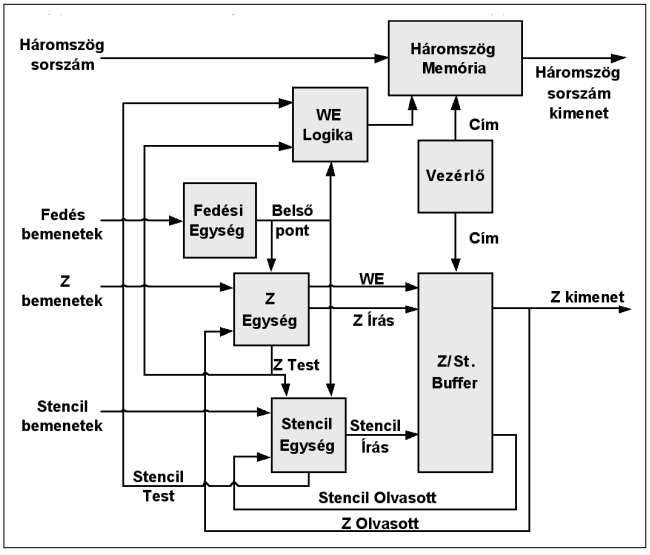
ahol a megfelelő együtthatók a csúcspont értékekből származtathatók:

$$\begin{aligned} A_z &= (z_1 - z_2) * (y_1 - y_0) - (y_1 - y_2) * (z_1 - z_0) \\ B_z &= (x_1 - x_2) * (z_1 - z_0) - (z_1 - z_2) * (x_1 - x_0) \\ C_z &= (x_1 - x_2) * (y_1 - y_0) - (y_1 - y_2) * (x_1 - x_0), \\ D_z &= A_z * x_0 + B_z * y_0 + C_z * z_0 \end{aligned} \qquad (4)$$

Mivel a cellák a bemeneti egységhez képest kétszeres órajellel járnak, és a minimális szegmens szélesség (azaz egy cella minimális feldolgozási ideje) 32 órajel, így a bemeneti egységnek 16 órajel áll rendelkezésre a szükséges adatok előállításához.

5.2.2. HSR cella

A tényleges feldolgozást a cellák végzik: a hozzájuk rendelt pixeleken végighaladva egyrészt megvizsgálják, hogy az belső pont-e, kiszámítják a megfelelő Z értéket (ez *x* és *y* irányokban haladva mindössze egy-egy összeadást igényel), és elvégzik a Z, valamint stencil tesztet. Egy cella vázlatát az *6. ábra* mutatja.



6. ábra HSR Egység

A feldolgozó modulok mellett minden cellához két, független memória is tartozik. Az egyikben a pixelenkénti Z és stencil értékeket tároljuk (Z/St Buffer), míg a másik kimeneti memóriaként funkcionál: minden pixelhez az azon látható háromszög sorszámát tárolja (Háromszög Memória).

Z egység

Az egységnek két különböző működési módja van az átlátszatlan, és az áttetsző háromszögek esetére. A már említett Z-buffer memóriában minden pixelhez két

érték tartozik (egymást követő páros és páratlan címeken), melyekből átlátszatlan esetben csak a páros címek kerülnek felhasználásra, míg áttetsző háromszögek feldolgozásakor a teljes memóriára szükség van.

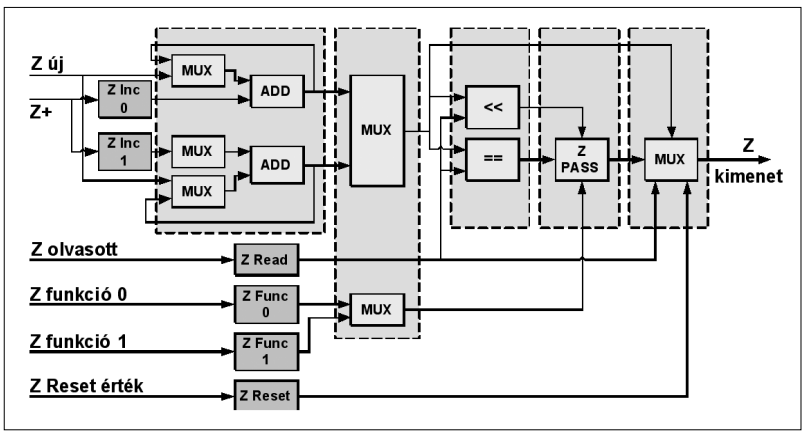
Átlátszatlan esetben a már ismertetett Z-buffer algoritmus kerül megvalósításra, azaz a feldolgozás alatt levő háromszög Z értékeit a Z-buffer tartalmával kell összehasonlítani. A szegmensek feldolgozása az átlátszatlan háromszögekkel kezdődik.

Az áttetsző háromszögek feldolgozása ([4]) ugyanakkor – lévén az áttetszőség sorrend-függő művelet – sorba rendezést igényel. A Z egység képes ezen háromszögek több menetben történő rendezésére és feldolgozására, pixelhelyes átlátszóságot érve el (a tipikus megjelenítők nem rendelkeznek hasonló funkcióval, ott a körülbelüli – nem pixelhelyes – sorba rendezés a CPU-ra hárul, ami a tipikus SOC rendszerek igencsak véges számítási kapacitását figyelembe véve nem feltétlenül tehető meg). Minden egyes feldolgozási menetben meghatározzuk azt a háromszöget, amely a kamerától a legmesszebb, de a már kiszínezett háromszögnél közelebb helyezkedik el. Ehhez – a legegyszerűbb megvalósítást tekintve – minden menetben fel kell dolgozni az összes áttetsző háromszöget.

Konkrét esetet (első menet) vizsgálva, a Z-buffer páros címein az adott pixelen látható átlátszatlan háromszög Z értéke található. Az áttetsző háromszögeket feldolgozva tehát keressük az ennél kisebb Z értékűeket, ami egy komparálást jelent pixelenként. Megszokott funkciójára felhasználva a Z-buffer páratlan című helyeit egy további komparálással lehetőség nyílik az áttetsző háromszögek közül a legtávolabbi kiválasztására. A Z teszt eredménye tehát akkor lesz igaz értékű, ha mindkét komparálás eredménye igaz. A következő menetben a Z-buffer egy pixelhez tartozó értékei funkciót cserélnek, hiszen ekkor a páratlan helyeken találjuk az első (már feldolgozott) áttetsző háromszög Z értékét, míg a páros címek felhasználhatók az újabb maximumkeresésre.

A Z egység vázlatos felépítését az *7. ábra* mutatja. A megfelelő működési frekvencia eléréséhez – csakúgy, mint a cella többi egysége – a Z Egység is pipeline szervezésű. Az áttekinthetőség érdekében az ábra csak az egyes fokozatok funkcionalitását mutatja (szaggatott téglalapok), a bennük levő regisztereket nem.

7. ábra Z egység



Egy háromszög feldolgozásának kezdetekor a háromszöghöz tartozó kezdeti Z értéket (*Z új*), az interpolációhoz szükséges értéket (*Z+*) és a vezérlő jeleket (Z funkció, reset érték) töltjük be. Az interpolálásra két, párhuzamosan működő összeadó szolgál (első fokozat), melyek két órajelenként, de egymáshoz képest egy órajellel eltolva írják saját regiszterüket.

Átlátszatlan esetben a két egység két, egymást követő pixel Z értékeit tartalmazza, míg áttetsző esetben ugyanazt a Z értéket. A következő fokozat multiplexere órajelenként váltakozva választ az előző fokozat két kimenete közül, majd a komparálási fokozatban megtörténik a Z-bufferből kiolvasott megfelelő értékkel történő összehasonlítás. Az összehasonlítás eredménye a beállított komparálási függvénynek (mindig igaz, soha nem igaz, kisebb, kisebb-egyenlő, nagyobb, nagyobb-egyenlő) megfelelően megadja hogy a Z teszt igaz-e. Ennek, valamint a fedési és stencil teszt eredményének felhasználásával már eldönthető, hogy milyen értéket kell a Z-bufferbe visszaírni:

– a feldolgozás alatt levő háromszög új értékét, amennyiben mindhárom teszt igaz;
– a reset értéket,
  – ha a Z teszt hamis, de belső pontról van szó és resetelni kell a Z-buffert,
  – nem belső pontról van szó és resetelni kell a Z-buffert;
– a kiolvasott értéket minden egyéb esetben.

A Z-buffer resetelésére (ami tipikusan a lehető legnagyobb értékkel való feltöltést jelenti, de ez nem megkötés) minden egyes új kép feldolgozásának kezdetekor szükség van.

Átlátszatlan esetben tehát a Z egység órajelenként képes egy pixelt feldolgozni, míg áttetsző esetben két órajel szükséges egy pixel feldolgozásához.

Stencil egység

A stencil funkció pixelek maszkolását teszi lehetővé, aminek megvalósítására egy pixelenkénti érték szolgál. Amennyiben a stencil teszt eredménye hamis egy adott pixelen, úgy annak színe nem módosul. Eme funkció segítségével számos különböző effektus megvalósítására nyílik lehetőség. Kompozíció során például több, különböző kameraállásból készített 2D vagy 3D képet illeszthetünk össze; a Stencil-buffer felhasználásával az újabb részleteket egy jól definiált, maszkolt területre helyezhetjük be. Így egyszerűen megvalósítható például szövegek megjelenítése, de ez a funkció megfelelő, például visszapillantó tükrök megjelenítésére is.

Az utóbbi esetben a vezető nézete az egyik 3D kép, míg a visszapillantó tükör pozíciójából készített kép a Stencil-buffer által kijelölt részre kerül. További, bonyolultabb effektusok (például árnyékok ([5]), tükröződések, objektum körvonalak) is meg-

valósíthatók a Stencil-buffer segítségével, ám ezeknek részletesebb ismertetése túlmutat jelen cikk keretein.

A stencil teszt során használt értékek:

- Stencil-buffer értéke (StBuff))
- Stencil referencia érték (StRef)
- Olvasási maszk (RMask)
- Írási maszk (WrMask)
- Komparálási funkció (COMP)
- Művelet (OP)

A fenti, zárójelben megadott jelölésekkel élve a Stencil-teszt eredménye a következő (& bitenkénti ÉS műveletet jelöl):

$$(StRef \ \& \ RMask) \ COMP \ (StBuff \ \& \ RMask) \qquad (5)$$

A komparálási funkció a Z egységnél már felsoroltak valamelyike lehet.

A Stencil-bufferbe írandó értéket az alábbi összefüggés adja meg:

$$(StBuff \ \& \sim WrMask) \mid (WrMask \ \& \ OP(StBuff)), \qquad (6)$$

ahol & bitenkénti ÉS, | bitenkénti VAGY, ~ pedig bitenkénti negálás műveletet jelent. Ellentétben a Z-bufferrel, a Stencil-buffer új értékkel történő írása nem csak abban az esetben lehetséges, ha a teszt eredménye igaz. Lehetőség van ugyanis külön művelet beállítására az alábbi esetekre:

- Stencil teszt hamis,
- Stencil teszt igaz, Z teszt hamis,
- Stencil teszt igaz, Z teszt igaz.

A lehetséges, a Stencil-bufferből olvasott értéken végrehajtható műveletek (OP):

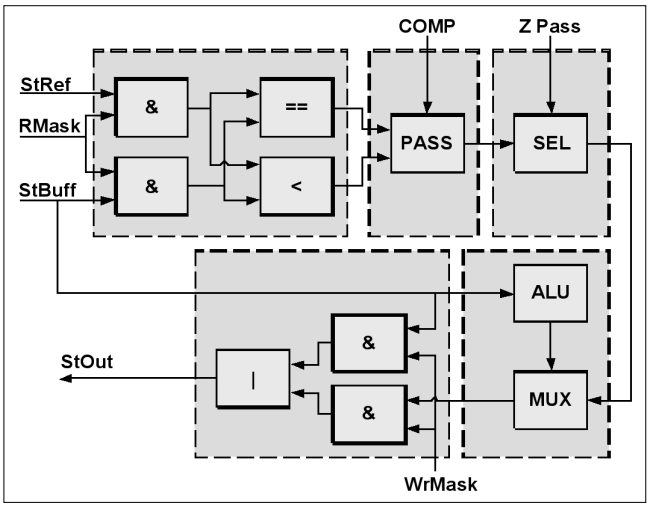
- nulla beírása,
- olvasott érték megtartása,
- bitenkénti invertálás,
- inkrementálás,
- inkrementálás szaturációval,
- dekrementálás,
- dekrementálás szaturációval.

A szintén pipeline felépítésű stencil egység blokkvázlatát a *8. ábra* mutatja, a szaggatott téglalappal körbekerített részek egy-egy fokozatot jelentenek.

Működése megfelel az eddig leírtaknak, talán csak annyi megjegyzés szükséges, hogy a választható műveletek eredményeit az ALU egység párhuzamosan generálja, s ezekből egy multiplexer választja ki a megfelelőt. A multiplexer vezérlőjelét a megelőző pipeline fokozat állítja elő, a stencil, Z és fedési teszteknek megfelelően.

6. Eredmények

A hardver modulok realizálása Xilinx XC2V6000-4 típusú, 6 millió kapu bonyolultságú FPGA (Field Programmable Gate Array) eszközön történt. A rendelkezésre álló, FPGA-ban implementálható processzor (Xilinx MicroBlaze) 80-100 MHz körüli órajelet képes elérni ezen eszközben, így a tervezett moduloknál is ez az órajel tartomány volt a cél.



8. ábra Stencil Egység

Az Indexelő egység összes modulja képes elérni a 100 MHz-es órajelet, így maximális számítási kapacitása 100 millió szegmens másodpercenként. A másodpercenként feldolgozható háromszögek száma a háromszögek által fedett szegmensek számától függ, a csúcsteljesítmény 33 millió háromszög másodpercenként (ekkor minden háromszög három vagy kevesebb szegmensben fed pixelt, és a szegmentáló bemeneti fokozata a limitáló tényező), míg az elméleti minimum 390 ezer háromszög másodpercenként (32x16 pixeles szegmens, 640x480 pixel felbontású kép és fél képernyőt fedő háromszögek).

A HSR Egység bemeneti egysége 100 MHz-es órajelen működik, míg maguk a cellák 200 MHz elérésére képesek. A jelenlegi, 8 cellát tartalmazó implementációban ez 1600 millió átlátszatlan pixel, és 800 millió áttetsző pixel feldolgozását jelenti másodpercenként. Mivel a szegmenseken belül a feldolgozás ideje a fedett pixelek számától független, az effektív kitöltési sebességet (a feldolgozott háromszög-pixelek számát) a szegmensekben átlagosan fedett pixelek aránya szabja meg, ami megfelelő szegmens méret megválasztásával maximalizálható.

Irodalom

[1] A. Watt: 3D Computer Graphics, Addison-Wesley, 2000.
[2] Szirmay-Kalos László: Számítógépes grafika, ComputerBooks, 2001.
[3] H. Holten-Lund: Design for scalability in 3D computer graphics architectures, Ph.D. Thesis. Technical University of Denmark, 2001.
[4] P. Diefenbach: Pipeline Rendering: Interaction and Realism Through Hardware-Based Multi-Pass Rendering, Ph.D. Thesis. University of Pennsylvania, 1996.
[5] Kilgard, M. J., Everitt C.: Optimized Stencil Shadow Volumes. Game Developer Conference, 2003.
[6] ATI Technologies, Performance Optimization Techniques for ATI Graphics Hardware with DirectX 9.0 ATI Radeon SDK, http://www.ati.com/developer/