Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Polygonální rendering metodou Sort-Last

Plzeň, 2007 Luboš Kolářík

Abstrakt

Sort-Last is a computer graphics technique for rendering extremely large data sets on clusters of computers. Sort-Last works by dividing the data set into even-sized chunks for parallel rendering and then composing the images to form the final result. A subimage is generated by each processing unit and the final image is obtained by compositing subimages in the proper order. Sort-last rendering requires the movement of large amounts of image data among cluster nodes. The network interconnecting the nodes becomes a major bottleneck. Sort-last system was implemented as a easy to use library. Experiments show that performance is mainly affected by the image composition time. Performance of this system showed up as not so good. Average FPS is on the edge of usability.

Prohlášení

Prohlašuji, že jsem bakalářskou/diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 13.5.2007 Luboš Kolářík

Obsah

1 Úvod

Paralelní rendering se snaží urychlit renderování složitých scén nebo objektů využitím dalších připojených stanic. Princip spočívá v rozdělení scény, její vykreslení na několika stanicích (clusteru) a následném složení výsledného obrazu. Cílem paralelního systému, může být zvýšení rozlišení výsledného obrazu, nebo možnost vykreslovat mnohem složitější scény a objekty. Zvýšení rozlišení je možné využít například pro velké displeje složené z několika monitorů. Naproti tomu zvýšení možné složitosti scény je využitelné v lékařství, pro prohlížení neskenovaných uměleckých děl či terénů. Jsou i scény, které se v dnešní době vůbec nevejdou do paměti počítače a není možné je v reálném čase vykreslovat bez použití více stanic.

Cílem této práce je navrhnout a realizovat systém, který by byl schopen vykreslovat právě takto složité scény. Mělo by jít o systém, ve kterém každá stanice vykreslí část scény a výsledný obraz pošle centrální stanici, která všechny složí do výsledku, který je vidět na monitoru stanice. Systém by se měl umět vypořádat s průhlednými plochami, což je u takovýchto systémů poměrně obtížně řešitelný problém.

V teoretické části jsou popsány různé principy pro paralelní vykreslování. Dále je detailněji popsán sort-last systém, který je využit pro realizaci práce. Jsou zde také rozebrány nejdůležitější problémy vznikající při implementaci systému a nástin jejich řešení. Jako poslední je zde popsán problém průhledných ploch.

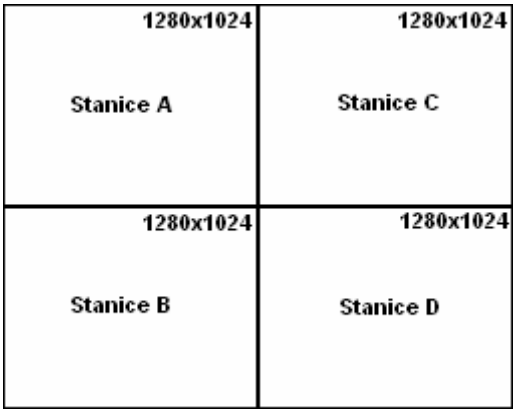
Ve třetí kapitole je detailně rozepsána realizace programu systému. Je zde popsáno fungování klientských (vykreslujících scénu) i serverové (skládající a zobrazující výsledek) aplikací. Dále jsou zde výsledky měření realizovaného systému.

2 Teoretická část

2.1 Druhy paralelních systémů

Vykreslení objektů složených z trojúhelníků probíhá ve dvou zásadních krocích. Jedním z nich je transformace geometrie a druhým je rasterizace. Po předání dat od aplikace jsou souřadnice vrcholů trojúhelníků transformovány do souřadnic vhodných pro vykreslení na obrazovku. Potom přijde na řadu rasterizace, kdy jsou vykresleny na obrazovku všechny pixely trojúhelníku. Systémy pro paralelní rendering se dělí na tři druhy, podle místa, kde dojde k rozdělení dat a jejich odeslání jednotlivým stanicím. Jsou to sort-first, sort-middle a sort-last.

Sort-first rozděluje data jednotlivým stanicím hned po jejich přijetí od aplikace, ještě před transformací. Sort-first systémy jsou používány pro vykreslování ve vysokém rozlišení u displejů složených z mnoha monitorů (tzv. tiled displejů – obr. 2.1.1.1).



2.1.1.1 – Realizace displeje s rozlišením 2560x2048 pixelů pomocí čtyřech displejů s rozlišením 1280x1024 a jejich rozdělení jednotlivým stanicím.

Každá stanice potom vykresluje jenom sobě přiřazenou výseč a zároveň jí zobrazuje na k sobě připojeném monitoru. Síla těchto systémů tkví právě v možnosti vykreslování velmi vysokých rozlišení. Protože data vykreslovaná jednou stanicí se mění s každým pohybem kamery (jejich výřez zobrazuje jinou část scény), jsou tyto systémy omezeny složitostí scény. Scéna musí být buď distribuována za běhu po síti, nebo musí být celá uložená na každé stanici. Dnešní hardware také provádí transformaci geometrie i pro objekty mimo výřez, takže složitost scény nemůže být o moc větší, než kdyby byla vykreslována na jedné stanici. Známý sort-first systém je například systém Chromium – (http://chromium.sourceforge.net/)

Sort-middle rozděluje data po jejich transformaci, ale ještě před rasterizací. Tento systém by byl ještě výhodnější pro tiled displeje než sort-first. Již transformovaná data je

totiž relativně jednodušší rozdělit mezi klienty. Stále zde platí, že data celé scény musí být distribuována na všech stanicích, takže je omezen velikostí scény stejně jako sort-first systémy. Bohužel na dnešním hardwaru zatím není rozšířená možnost zasahování do vykreslování, která by umožnila získat data mezi transformací a rasterizací. Proto se tento systém používá minimálně.

Sort-last rozděluje data ještě před započetím vykreslování podobně jako sort-first systém. Rozdíl oproti sort-first systému je v zobrazení výsledku. Stanice posílají data po síti zpět hlavní stanici, která všechny obrazy zkomponuje do sebe a vytvoří jeden, který následně zobrazí. Obraz musí pro správné složení obsahovat informace o hloubce pixelu. Při skládání se vybere od každé stanice pixel s nejmenší hloubkou a ten se uloží do výsledného obrázku. Z toho vyplývá, že úvodní rozdělení je zcela nezávislé na pozici objektů a může být prakticky náhodné. Proto je možné rozdělit scénu na části a každou část natrvalo přiřadit jedné stanici – stanice nepotřebuje mít v paměti celou scénu, takže její složitost není omezena přenosy přes síť, nebo pamětí stanic. Naproti tomu se přes síť musí přenášet výsledné obrázky, jejichž velikost narůstá se vzrůstajícím rozlišením. Proto se sort-last systémy používají pro velmi složité scény, vykreslované při nižším rozlišení. Existující sort-last systém je popsán například na http://www.evl.uic.edu/cavern/rg/20040411_kirihata/.

Hybridní systémy – Jsou založeny na kombinaci systémů. Využívají například zároveň sort-last systém, který umožňuje vykreslování složitých scén a sort-first, který zároveň umožňuje vysoké rozlišení. Příkladem je systém popsaný v [3].

2.2 Sort-last systém

2.2.1 Princip

Hned po přijetí dat z aplikace je musí roztřídit jednotlivým stanicím. Ty následně vykreslí svojí část scény s použitím funkce renderování do textury, kdy scéna není vykreslována do backbufferu (místo v paměti alokované pro výsledný obraz, z něhož se potom vykresluje na monitor), ale do určeného místa v paměti. Odtud se potom dají data poměrně snadno získat pro další použití. Protože nejpomalejším místem v systému je síť, kterou jsou jednotlivé stanice propojeny a výsledné obrázky jsou datově poměrně náročné, je nutné obraz, pokud možno bezeztrátově, zkomprimovat. Tato komprese musí být zároveň dost rychlá pro využití v reálném čase. Potom se teprve může obraz odeslat nadřízené stanici. Ta potom porovná hloubky u každého pixelu, vybere vždy pixel od klienta s nejmenší

hloubkou (ostatní jsou jím zakryty, protože jsou dál od kamery) a ten uloží do výsledného obrazu. Ten potom jednoduše vykreslí, nebo jinak předá uživateli systému.

2.2.2 Rozdělení geometrie

Veškerá geometrie musí být přiřazena jednotlivým stanicím. To je možné učinit za běhu programu, například pokud se počítá se změnami scény. Stejně tak je možné ji rozdělit při startu. Tady je nutné mít informace o všech vykreslovaných objektech. Existují dvě zásadní možnosti rozdělení.

První možností je nezávisle na pozici. Výhodou těchto metod je jednoduchá implementace. Velkou nevýhodou je nutnost přenášet spolu s obrazem i informace o hloubce pixelu. To je 16-32 bitů ke každému pixelu. Náročnost na síť potom vzroste pro 24-bitové barvy o dvě třetiny. Další nevýhodou je složitější (a tím pádem i pomalejší) skládání výsledného obrazu.

Další možností je rozdělit si vykreslovanou scénu na souvislé části. Každý klient potom kreslí jednu část prostoru scény. Nevýhodou je složitější implementace, kdy je nutné při startu programu scénu rozdělit pro aktuální počet klientů. Dále je nutné rozdělit objekty podle jejich pozice ve scéně. Je třeba také počítat se situací, kdy jeden objekt zasahuje do více částí scény. Na druhou stranu není nutné posílat informace o hloubce (pokud jsou části vytvořeny tak, že je lze seřadit podle vzdálenosti od kamery). Posílá se jenom informace o průhlednosti, na což stačí 1bit/pixel. Pokud systém pracuje s poloprůhledností, je nutné použít více bitů. Běžně se používá 1 byte. I tak je to polovina dat navíc oproti 16-tibitové hloubce. Potom se zjednodušuje i skládání, kdy stačí vykreslit obrázky od všech stanic od té, která má přiřazenu nejvzdálenější část po tu s nejbližší pře sebe s využitím alphablendingu.

Metody rozdělení scény:

Náhodně – Nejjednodušší metoda. Každý objekt se přiřadí náhodné stanici. Rozdělování je nezávislé na pozici. Asi nejjednodušší možná metoda. Není zaručeno optimální rozdělení zátěže na stanice (při nestejně složitých objektech)

Určeno uživatelem – Uživatel si sám určí stanici, která má objekt vykreslovat. Výhodou je variabilita, která umožňuje uživateli napsat si vlastní rozdělovací algoritmus a třeba i zohledňovat nestejný výkon stanic.

Rozdělení scény pomocí Quadtree, Octtree – Scéna je rozdělena pomocí těchto stromů. Nevýhodou může být složitost částí, které mohou vzniknout při rozdělování scény.

Při každém dalším dělení listu stromu vzniknou další 4 (8 u Octtree) části. Není zaručeno pro všechny počty stanic, že bude stejně částí jako listů stromu. Je nutné mít informace o celé scéně při vytváření stromu (nebo ho znovu generovat za běhu). Výhodou je poměrně jednoduché vytvoření vyváženého stromu a tím i rovnoměrné rozložení zátěže.

Rozdělení scény pomocí BSP stromu – Scéna je rozdělena BSP stromem. Asi nejsložitější na implementaci. Má všechny výhody i nevýhody Quad a Octtree. Výjimkou je možnost vygenerovat strom s jakýmkoliv počtem listů.

2.2.3 Kódování a přenos obrazu

Po vykreslení scény do textury a její zkopírování do přístupné paměti je nutné obraz přenést přes síť ke stanici, která zkompletuje výsledný obraz. To ovšem není tak jednoduché. Pokud budeme počítat rozlišení 1024*768 a 24 bitů (3 byty) na pixel, vyjde nám velikost jednoho obrázku $1024 \times 768 \times 3 = 2359296$b. To je 2,25MB. Pokud počítáme dnešní nejrozšířenější ethernetovou síť s rychlostí 100Mb/s, což je 12,5MB/s (a to je jenom teoretická rychlost, reálná propustnost je nižší) vyjde nám asi 5,5 snímku za sekundu. To je ještě použitelné minimum, při kterém je ale již znát výrazně trhaný pohyb. To je ovšem jenom barevná informace. Podle zvolené metody je nutné ke každému pixelu přidat 1(průhlednost), 2(16 bitů hloubky) nebo 3(24 bitů hloubky) byty. Maximální možné FPS (frames per second -počet snímků za sekundu) se sníží na 4,17 pro 4 byty, 3,33 pro 5 bytů a 2,78 pro 6 bytů na pixel. To jsou už nepoužitelné hodnoty. Tyto hodnoty také odpovídají jednomu klientu. Pokud bychom připojili druhého, bylo by nutno přijmout dvakrát tolik dat a teoretické hodnoty FPS by byly omezeny na polovinu. A dále by klesaly s připojováním dalších klientů.

Proto je nutno obraz komprimovat. Existuje mnoho ztrátových i bezeztrátových kompresních algoritmů používaných pro kompresi dat. Většinu jich ovšem pro tuhle situaci není možné použít, protože komprese celého obrazu musí proběhnout několikrát za vteřinu a složité algoritmy s dobrým kompresním poměrem jsou příliš pomalé.

Jedna nejjednodušších kompresních metod používaných pro tyto systémy je RLE (run length encoding). Tento algoritmus využívá toho, že více pixelů vedle sebe může mít stejnou barvu. Potom místo toho, aby byl odeslán každý pixel zvlášť, odešle se barva jenom jednou spolu s počtem opakování. Výhodou je velmi rychlá komprese a dekomprese, kterou by mělo být možné použít i pro kompresi a dekompresi obrázků v reálném čase. Nevýhodou je ne moc dobrý kompresní poměr, který může jít až do hodnot

větších než jedna, pokud je obrázek velmi různorodý (nevyskytují se vedle sebe stejné barvy, je často odesílána barva spolu s počtem jedna).

Další možností je zjišťovat u každého snímku jeho okraje, a neodesílat prázdná místa okolo. To je možné využít hlavně u systémů, které dělí scénu na bloky. Pokud je daný blok od pozorovatele daleko, zmenší se jeho velikost a tím i velikost přenášeného obrazu. Její využitelnost ovšem sráží současné použití RLE komprese, která jednobarevné pozadí (které by bylo normálně odříznuto) zkomprimuje s vysokou účinností.

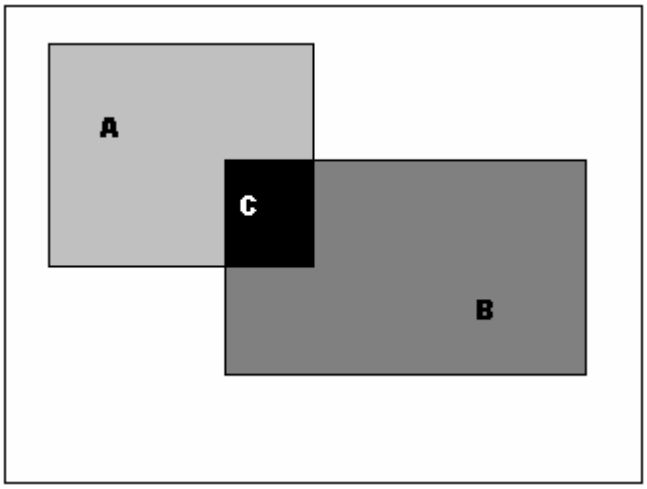
Pokud počítáme s tím, že scéna se bude měnit jen pomalu, lze použit ke zmenšení množství přenesených dat operaci XOR s předchozím snímkem. Předchozí snímek se uloží a při odesílání dalšího snímku se aplikuje XOR byte po bytu mezi těmito snímky. Po přijetí se provede stejná operace, čímž dostaneme původní obrázek. Po aplikaci XOR vzniknou data, kde 1 označuje bit, který se od minulého snímku změnil. Oproti tomu 0 označuje bit beze změny. Pokud by se snímky tedy jen minimálně měnily, většina takto upravených dat by byla 0, což by mělo zvýšit účinnost RLE komprese.

2.2.4 Skládání výsledného obrazu

Skládání obrazu je operace, při které se po získání částečných obrazů ode všech stanic sestaví výsledek pro konečné zobrazení. Základní verze sort-last systému počítá s náhodně rozloženými daty. Proto jsou ke každému obrazu přijaty také informace o hloubce pixelů. Vlastní složení je potom poměrně jednoduché. Pro každý pixel se ze všech přijatých obrazů vybere ten s nejmenší hloubkou. Ten je potom zapsán do výsledného obrazu. Ovšem rychlost může být problémem. Pro vyšší rozlišení je skládání náročnou operací, která se navíc musí provádět každý snímek. Pro každý pixel je nutné projít seznam pixelů patřících na jeho místo a najít ten s nejmenší hloubkou. Tím pádem i zvětšení počtu klientů může skládání zpomalovat. Proto je i tady nutné hledat různé urychlovací metody.
První a nejjednodušší možné zrychlení je využití toho, že jedna stanice nemusí vykreslovat geometrii přes celou plochu výsledného obrázku. Pokud víme, že v určitém místě obrazu není nic vykresleno, nemusí se pro toto místo vůbec data od této stanice procházet. Pokud známe ohraničující obdélníky od dvou stanic a chceme složit jejich částečné obrazy do jednoho, stačí porovnávat hloubky jenom v místě, kde mají ohraničující obdélníky společnou plochu (obr. 2.2.4.1). Tímto se plocha, kde je nutné porovnávat hloubky pixelů výrazně sníží.

Další možností je i skládání provést paralelně na více strojích. Stanice se rozdělí na páry. Každá stanice z páru potom složí polovinu obrazu. Každá stanice má v tomto bodě

složenou polovinu obrazu od dvou stanic. Následně je možné vytvořit z každých dvou párů stanic dva nové páry (pokud dělíme obraz na levou a pravou polovinu, spojíme stanice, které skládaly levé poloviny do nového páru, stejně tak stanice skládající pravé poloviny). Teď je možné opět rozdělit obraz napůl (dělíme už ale jenom polovinu, takže každá stanice skládá čtvrtinu velikosti obrazu) a zase každá stanice složí svou část obrazu. Takto je možné



2.2.4.1 – Provedení skládání jenom v místech, kde je to nutné. A – Plocha vykreslená jednou stanicí B – plocha vykreslená druhou stanicí C – plocha, kde je nutno provést

pokračovat pro libovolný počet stanic. Výsledný obraz je potom sestaven jakoby z dlaždic s velikostí závislou na počtu stanic.

Pokud je systém založen na dělení geometrie podle její pozice v prostoru (například pomocí stromů), je ovšem skládání scény mnohem jednodušší. Stačí poslat spolu s informací o barvě každého pixelu i informaci o jeho průhlednosti (s tím, že pozadí bude mít hodnotu průhlednosti 0 – úplně průhledná). Není nutné posílat informaci o hloubce pixelů. Při skládání se potom seřadí jednotlivé části prostoru podle vzdálenosti od kamery. Pomoci alphablendingu se potom jednoduše vykreslí částečné obrazy od nejvzdálenějšího k nejbližšímu. Pixely, která některá stanice nepoužila (nenakreslila do nich nic) jsou průhledné, a je skrz ně vidět na „vzdálenější“ částečné obrazy.

2.3 Průhlednost u paralelních systémů

Průhlednost je jedním z velkých problémů sort-last systémů. Geometrie může být vykreslována poloprůhledná pomocí tzv. alphablendingu. To je proces, při kterém se míchají právě vykreslované pixely s pixely již vykreslenými. Poměr v jakém se mají barvy míchat může být určen texturou, konstantou atd. Výsledná barva se počítá následovně:

$C = C_1 \times A + C_2 \times (1 - A)$, kde C je výsledná barva, C1 je nově nanášená barva, C2 je barva již uložená ve framebufferu a A je číslo určující průhlednost nanášeného pixelu (0 – zcela průhledný až 1 – zcela neprůhledný). Nově nanášená barva je tedy závislá na barvě, která již byla uložená v bufferu (objekt je částečně průhledný, takže musí být vidět, co je za ním). Tato operace je nekomutativní. Všechny objekty musí být pro správný výsledek naneseny od nejvzdálenějšího po nejbližší. Další důvod pro řazení průhledných objektů je z-buffer. Pokud napřed vykreslíme bližší průhledný objekt tak se do z-bufferu uloží jeho hloubka. Pokud se potom pokusíme vykreslit vzdálenější (který by měl být přes původní objekt vidět) pixely za již nakresleným objektem budou mít větší hloubku a nebudou vůbec vykresleny.

U sort-last systémů je tento problém ještě markantnější. Trojúhelníky jsou náhodně rozloženy po obrazovce. Proto je možné, že jedna stanice vykreslí poloprůhledný objekt a jiná stanice by měla vykreslit další objekt dál, při skládání se potom použije ten pixel, který je blíž a vzdálenější objekt nebude vidět. Při využití více stanic pro vykreslování tedy nestačí jenom seřadit průhledné objekty. Systém musí řešit další problém.

Jednou možností je pomatovat si všechny barvy, které byly nakresleny na jedno místo (pixel). Potom je seřadit podle vzdálenosti a spočítat z nich výslednou barvu. Tahle metoda je ale extrémně náročná. Znamená to posílat mnohem více obrazových dat. Při skládání by pak bylo nutné pro každý pixel seřadit několik těchto seznamů (pro každou stanici jeden) do jednoho a z toho pak spočítat výslednou barvu. Tuto metodu není možné pro realtimové systémy použít bez masivních optimalizací. Pokud chceme použít pro vykreslování grafickou kartu, prakticky je problém tyto seznamy získat, protože v průběhu vykreslování, do kterého se nedá moc zasahovat, se pixely neustále překreslují bližšími.

Další možnost je vykreslit průhledné objekty do jiného bufferu a posílat je zvlášť. Při skládání by se napřed složily průhledné a neprůhledné objekty zvlášť a z těchto dvou obrazů by se zase pomocí porovnávání hloubky pixelu (s tím, že pokud je průhledný pixel blíž než neprůhledný, spočítá se výsledná barva podle míry průhlednosti). Nevýhodou této metody je, že mohou vzniknout situace, kdy nebude přesná. Pokud například bude jedna stanice vykreslovat dva průhledné objekty přes sebe, složí jejich barvy dohromady. Pokud ale jiná stanice vykreslí mezi těmito objekty jiný objekt, výsledky budou zkreslené. Pokud bude neprůhledný, bude v daném pixelu barva spočítaná špatně, jako kdyby oba průhledné objekty byly před neprůhledným, který je ale mezi nimi.

Pokud používáme dělení prostoru na oblasti pro jednotlivé stanice, je možné zase využít zjednodušeného skládání. Všechny poloprůhledné objekty je možné vykreslit do

jednoho bufferu (po vykreslení všech neprůhledných objektů a seřazené podle vzdálenosti od kamery). Částečný obraz potom obsahuje informace o průhlednosti. Při vykreslování částečných obrazů seřazených podle vzdálenosti se barvy správně smíchají. Ani tato metoda ovšem nemusí být vždy zcela přesná. Pokud je na jedné stanici vykreslován průhledný objekt, za kterým už nic jiného není, použije se při alphablendingu barva pozadí, která se smíchá s barvou objektu. Pokud se potom při skládání ukáže, že jiná stanice vykreslila objekt za tímto objektem, barvy se sice smíchají správně podle jejich průhlednosti, ale v barvě průhledného objektu již je započítána barva pozadí – výsledná barva je jiná.

Pokud má systém být schopen řešit průhledné plochy je tedy nutno zvolit kompromis, mezi přesností výsledku a rychlostí zobrazování.

3 Realizační část

Systém se skládá ze dvou aplikací. Jedna se spouští ne všech klientech (stanice určené pro vykreslení částečných obrazů) – klient a druhá se spouští jenom na jedné stanici. Ta potom řídí vykreslování a zobrazování výsledného obrazu. Všechny části systému jsou tvořeny pomocí programovacího jazyka C# a multimediálního balíku DirectX. Managed DirectX a C# jsou velmi jednoduché na používání. Programy se dají psát rychleji než v nižších programovacích jazycích jako je třeba C/C++. Nevýhodou je však nižší výkon vytvořených aplikací.

3.1 Server

Server je program vytvořený uživatelem knihovny. Všechno vykreslování se provádí pomocí tříd z této knihovny (hlavní třídy používané serverem jsou znázorněny na obr. 3.1.1.1). Existují dvě třídy, přes něž je možno vykreslovat scénu.

StaticRenderer – třída zapouzdřující vykreslování statické scény. Uživatel při startu programu definuje celou scénu a za běhu upravuje jenom pozici kamery.

DynamicRenderer – třída zapouzdřující vykreslování dynamické scény. Tato metoda je nejvíce podobná standardnímu vykreslování pomocí Managed DirectX. Je možné vykreslovat pohyblivé objekty, měnit parametry vykreslování atd. Cenou za mnohem větší volnost při vykreslování scény je nižší výkon oproti statické scéně.

3.1.1 Síťová komunikace

Komunikace mezi serverem a jeho klienty je jedním ze základních kamenů tvorby systému.

V počáteční fázi výstavby systému bylo použito balíku DirectInput pro síťovou komunikaci. Bohužel s tímto balíkem se ani zdaleka nepodařilo využít maximální přenosovou kapacitu sítě (maximální rychlost přenosu byla mezi 1-1,2MB/s). Proto je v konečné verzi systém postaven na třídě Socket .NET frameworku. Pro komunikaci se až na výjimky využívá UDP protokol. Veškerá komunikace probíhá přes třídu zvanou Network. Ta má na starosti několik věcí

Správa klientů – Síťová vrstva naslouchá na stanoveném portu a očekává žádost o připojení nového klienta. Pokud taková žádost přijde, odešle potvrzení a přidá si IP adresu a port do seznamu připojených klientů. K novému klientovi také přiřadí jeho id, které je použito při odesílání zpráv. Podobně tato třída zajišťuje odpojení klienta.

Odesílání dat klientům – Třída Network obsahuje několik jednoduchých funkcí, které pošlou specifikované data klientovi určenému jeho id. Také je možno odeslat data všem připojeným klientům.

Přijímání dat od klientů – Ke každému klientu je možné přiřadit jednu metodu, která bude zavolána v případě, že jsou od klienta přijata nějaká data.

3.1.2 Načítání dat

Způsob načítání dat je rozdílný pro statickou a dynamickou verzi systému.

Statická verze – Načítání dat probíhá při startu vykreslování. Uživatel musí zadat popisy (ve struktuře ModelDescription) všech modelů ve scéně. Ty jsou potom najednou rozděleny jednotlivým klientům.

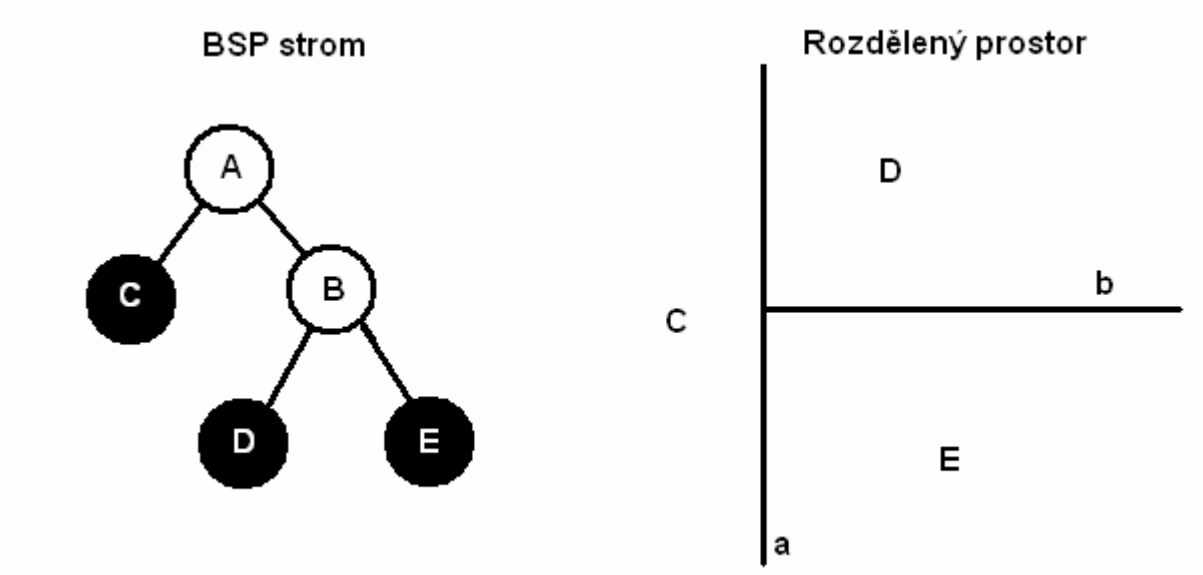
Dynamická verze – Při startu vykreslování je nutné podobně jako u statické verze předat systému seznam objektů použitých ve scéně. Rozdíl je ale v tom, že tento seznam je použit jenom pro rozdělení prostoru klientům (viz 3.1.3). Pomocí metod LoadMesh a LoadTexture je možné založit mesh nebo texturu do systému (doopravdy se načítají u klienta až pokud je klient potřebuje pro vykreslování za běhu programu). Tyto metody vracejí číslo, které se pak používá pro manipulaci s daty.

3.1.3 Rozdělení dat klientům

Systém využívá rozdělení scény na oblasti, kde jednu oblast vykresluje vždy jeden klient. Při startu programu je nutné zadat seznam objektů spolu s jejich pozicemi a

kolizními kvádry. Pomocí těchto dat je vytvořen BSP strom. BSP strom je struktura tvořená z uzlů stromu a listů. Každý uzel má přiřazenou plochu, podle které dělí prostor. Dále má přiřazeny dva podřízené uzly (pokud uzel nemá podřízené uzly, jde o poslední uzel stromu – list). Jeden představuje prostor před dělící plochou, druhý za dělící plochou. Tyto uzly mohou mít přiřazenou další dělící plochu a tím znovu dělit jim přiřazený prostor pro další dva podřízené uzly (viz. obr. 3.1.3.1).

Při použití pro dělení scény jeden list stromu potom odpovídá jednomu klientovi (pro dynamickou část se mohou pozice objektů změnit v průběhu vykreslování, vytvořený strom ale zůstane stejný). Strom je vytvářen s ohledem na složitost objektů pokud možno vyvážený. Mezi daty o každém objektu je i položka označující počet trojúhelníků, z nichž je složen. Strom je potom tvořen rekurzivně. Takto volaná funkce vždy obdrží seznam objektů náležících k dané větvi stromu. Najde nejvýhodnější další rozdělovací plochu (projde všechny plochy tvořené každou stranou každého kolizního kvádru a vybere tu, pro kterou je na obou



3.1.3.1 – Jednoduchý BSP strom a jím rozdělený prostor. Plocha a uzlu A dělí prostor na dvě poloviny. Jednu polovinu má přiřazenu list C a o druhou polovinu se dělí listy D a E

stranách přibližně stejně trojúhelníků). Potom vytvoří dva nové uzly a rekurzivně zavolá samu sebe pro tyto nové uzly. Toto je opakováno, dokud není vytvořen strom s požadovaným počtem listů.

3.1.4 Řízení vykreslování

Řízení vykreslování probíhá zcela odlišně pro statickou a dynamickou část.

Statická část – Po tom, co uživatel při startu zadá systému seznam objektů a jejich transformací, nemůže už jejich vykreslení vůbec ovlivnit. Jedinou možností je nastavení pozice a směru pohledu kamery. Není možné nastavit ani parametry vykreslovaní. Všechny objekty jsou vykresleny bez uživatelovy asistence. Uživatel může nastavit pouze použitá světla. Server při počátečním rozdělení přiřadí jednotlivé objekty klientům.

Dynamická část – Tady už se systém víc blíží k obvyklému vykreslování na jednom počítači. Uživatel má možnost si nastavit parametry pro vykreslování i světla. Podobně jako u obyčejných programů pro vykreslování se nastavují textury. Pro vykreslení objektu se používá metoda DrawMesh, která jako parametr očekává číslo identifikující model dříve načtený pomocí LoadMesh. Při každém vykreslení se zjistí, do kterých částí scény model zasahuje a podle toho se pošlou příkazy k vykreslení objektu jen klientům, kteří mají tyto části „na starosti“.

3.1.5 Skládání výsledného obrazu

Obraz se skládá pro obě verze systému shodně. Když jsou přijaty částečné obrazy, je zavolána metoda pro vykreslení výsledku. Při skládání je využito rozdělení scény na souvislé bloky. Pomocí BSP stromu se částečné obrazy seřadí podle vzdálenosti od pozorovatele. Pozadí částečných obrazů je nastaveno na zcela průhledné, takže výsledný obraz se zobrazí pouhým překreslením všech částečných obrazů pře sebe s využitím alphablendingu. Stejně se postupuje i pokud jsou ve scéně poloprůhledné objekty.

3.2 Klient

Klientská aplikace je postavena tak, aby v ní uživatel knihovny nemusel provádět žádné úpravy. Jedna aplikace je využita pro oba druhy scény.

3.2.1 Načtení dat

Pro načítání dat je využito .X formátu, který je podporován DirectX. Tento formát také může obsahovat informace o použitém materiálu a textuře.

Pro statickou scénu probíhá načtení dat při zahájení vykreslování. Program přijme od serveru seznam modelů, které má načíst. Výslednou cestu k souboru dostane následovně. Vezme cestu nastavenou po spuštění programu v příslušném okénku. Potom přidá „/meshes“ a nakonec jméno souboru přijaté od serveru. Z výsledné cesty se pak

pokusí načíst soubor s daty o modelu. Pokud model obsahuje materiál s připojenými texturami, pokusí se je načíst také. Výslednou cestu potom získává stejně jako pro soubor modelu, jenom místo „/meshes“ přidává „/textures“. Pokud jakýkoliv soubor není na disku nalezen, pokusí se o něj zažádat server.

Pro dynamickou scénu je načítání mnohem jednodušší. Cesta se použije přímo tak, jak přišla ze serveru ani se nenačítají textury přiřazené k modelu. Pokud soubor neexistuje, je o něj požádán server. Soubor je potom uložen na disk a otevřen stejně jako kdyby tam byl od začátku.

3.2.2 Vykreslení objektů

Pro statickou scénu je vykreslení objektů poměrně jednoduchou záležitostí. Objekty máme u každého klienta rozděleny do dvou seznamů pro neprůhledné a průhledné. Neprůhledné se tedy jednoduše vykreslí. K průhledným objektům je ještě navíc přiřazena jejich pozice. Podle této pozice jsou seřazeny podle vzdálenosti od kamery a vykresleny od nejvzdálenějšího.

U dynamické scény je všechno řízeno uživatelem. Proto má každý klient informace o všech objektech. Pokud přijde požadavek na vykreslení určitého objektu, program ho vzhledá podle jeho id (které je na serveru specifikováno uživatelem při voláni metody pro kreslení). Pokud objekt ještě není načten, načte se z disku (případně se přenese přes síť ze serveru). Potom už je vykreslen s použitím aktuálních nastavení.

3.2.3 Kódování obrazu

Po vykreslení snímku je nutné obraz zkopírovat z grafické paměti, zkomprimovat a odeslat serveru. Pro kompresi obrazu je použita RLE komprese a volitelně i XOR s předchozím snímkem. Pro zvýšení účinnosti je celý obraz převeden do YUV barevného prostoru. Protože ve většině obrázků se nejvíc mění jasová složka (Y), je v tomto prostoru vyvedena do samostatného kanálu. Druhé dvě složky by potom měly dosahovat lepšího kompresního poměru (hodnota barev se mění méně než světlost, takže účinnost RLE komprese se zvyšuje). Při kompresi obrazu bez převodu do YUV barevného prostoru nebyla RLE komprese účinná. Průměrný kompresní poměr byl dokonce vetší než 1, konkrétně asi 1,3. S využitím převodu se kompresní poměr zlepšil průměrně na číslo okolo 0,6. Tento převod je urychlen grafickou kartou.

Poté dojde na vlastní RLE kompresi. Komprimuje se každý kanál zvlášť.

3.2.4 Přenos dat

Pro přenos dat je využito UDP protokolu. Jeho nevýhodou je možnost přehození pořadí či dokonce ztráta paketu při přenosu. Také je limitována maximální velikost dat, které je možno přenést najednou. Proto se snímek přenáší po 1024 pixelech. Tento počet pixelů se zkomprimuje RLE kompresí a odešle serveru s umístěním prvního pixelu na obrazovce. Tímto je zaručeno, že i při výpadku paketu nemusí dojít k zastavení celého systému. Ovšem na jednoduchých LAN sítích je ztráta velmi nepravděpodobná.

3.3 Možnosti využití, omezení

3.3.1 Statická část

Verze systému pro statické scény je využitelná prakticky jenom pro urychlení vykreslování jednoduché scény. Uživatel může zasahovat do vykreslování jenom nastavením světel a upravováním pozice kamery. Tato aplikace je vhodná pro prohlížení složitých modelů, například neskenovaná umělecká díla. Výhodou je možnost zobrazování objektů, které se na jedné stanici ani nevejdou do paměti a bylo by nutné je každý snímek načítat z disku. Naproti tomu pro běžné využití jako počítačové hry je zcela nevyužitelná. Drtivá většina grafických efektů zde není možné realizovat.

3.3.2 Dynamická část

Verze systému pro dynamické scény je využitelná již pro více možností. Hlavním polem zájmu stále zůstávají extrémně složité scény. Navíc je v této verzi možné scénu upravovat za běhu programu. Je také možné zasahovat do nastavení parametrů vykreslování. Velkou nevýhodou tohoto systému je jeho nízký výkon v porovnaní se systémem pro statické scény. Využitelnost pro počítačové hry je sice na vyšší úrovni, ale pořád je většinu efektů nemožné implementovat. Například kvůli nemožnosti použit shadery, stencil buffer atd.

3.3.3 Možnosti úprav klienta

Pokud pomineme možnost úpravy zdrojového kódu, stále nám zůstává jedna možnost jak upravit funkčnost klientské aplikace. Po spuštění aplikace je možné vybrat DLL knihovnu, která obsahuje třídu později použitou pro načítání a zobrazení modelů. Tím pádem je možné bez zásahu do klientské aplikace změnit například načítaný formát

souboru na jiný, nebo i vygenerovat mesh programově. Změnou metody pro vykreslování je možné omezeně zasahovat do vykreslování modelů i u systému pro statické scény.

3.4 Testy systému

Pro otestování výkonu jsem použil pět verzí jednoduché scény. Každá verze byla složena z jiného počtu trojúhelníků. Protože clustery stanic nemá cenu používat pro jednoduché objekty, testované scény byly složeny z asi 356 000 – 1 780 000 trojúhelníků. Poslední modely dnešních grafických karet by pravděpodobně tyto počty zvládly bez problémů samy. Testy byly ale provedeny na starších modelech. Měřil jsem výsledky pro 2,3 a 4 klienty napojené na server. Použitá rozlišení byla 320x240, 640x480 a 800x600.
Testované stanice byly vybaveny procesory Intel Pentimu4 2,8GHz, 2GB RAM a grafickou kartou ATI FireGL T2.

3.4.1 Test jedné stanice

Pro srovnání jsem provedl test stejné scény při stejných rozlišeních bez použití urychlení pomocí dalších stanic.
V tabulce jsou naměřené průměrné FPS (frames per second - počet snímků za sekundu) v závislosti na složitosti scény a použitém rozlišení obrazu.

	356 000	712 000	1 068 000	1 424 000	1 780 000
320x240	17,679	8,97	5,77	4,26	3,48
640x480	16,9	8,55	5,68	4,28	3,44
800x600	15,96	8,54	5,61	4,27	3,42
1024x768	13,86	8,43	5,31	4,27	3,39
1280x1024	13,786	8,12	5,2	4,13	3,22

3.4.1.1 – Naměřené FPS pro měřenou scénu vykreslovanou pouze jednou stanicí bez použití Sort-Last systému

Z výsledků je jasně vidět, že rozlišení prakticky vůbec neovlivňuje naměřené výsledky. Tady se projevuje složitost scény. Grafická karta bez problémů stíhá rasterizovat trojúhelníky, ale nestíhá transformovat vertexy. Proto se naměřené FPS prudce snižují při zvýšení složitosti scény.

3.4.2 Rychlost zobrazování

Výsledky testů jsou v grafech 3.4.2.1 – 3.4.2.6. V nejnižším rozlišení 320x240 je paralelní systém už od zhruba 500 000 trojúhelníků rychlejší než jedna stanice. Zajímavý je vysoký výkon clusteru složeného ze tří stanic. Pravděpodobně je to pro tento systém ideální počet stanic. Cluster složený ze třech stanic dosahuje dvakrát vyšších hodnot než ostatní systémy. Při takto nízkém rozlišení je ještě poznat snižující se rychlost vykreslování při zvyšující se složitosti scény. U clusterů z více stanic se sice hodnoty snižují o něco pomaleji, ale domnívám se, že by rozdíl oproti jedné stanici mel být větší. To ukazuje na velikou režii paralelního systému a z toho ne moc dobré využití výpočetního výkonu clusteru.

V rozlišení 640x480 už se začíná projevovat náročnost systému při vysokých rozlišeních. Systém podává stejné nebo lepší výsledky při třech klientech až kolem 1 200 000 trojúhelníků ve scéně. Pro 2 a 4 klienty je to dokonce až kolem 1 500 000 trojúhelníků. Se vzrůstající scénou klesají hodnoty systému velmi pomalu. Pokud by to znamenalo vysoký výkon vykreslování, který má velkou rezervu ve složitosti scény, byl by to úspěch. Protože nižší rozlišení ovšem ukazuje poměrně silnou závislost na složitosti scény, musí jít o limitaci systémem. Existují dvě možnosti, kde by mohlo ke zpomalení dojít. Buď při přenosu přes síť, nebo při kódování a dekódování částečných obrazů. Další experimenty ukazují, že systém nejvíc zpomaluje RLE komprese částečných obrazů. Skládání výsledného obrazu by zpomalení nemělo způsobovat, protože výsledky se moc neliší v závislosti na počtu klientů. Použitý způsob skládání by měl být velmi rychlý, díky jeho provedení grafickou kartou.

Rozlišení 800x600 dopadlo velmi podobně jako nižší rozlišní 640x480. Výsledné hodnoty se snížily, ale stejně jako u 640x480 jsou jen minimálně závislé na složitosti scény, či počtu připojených stanic.

Výsledky systému pro dynamickou scénu jsou mnohem nižší. Vysoká volnost při rendering je vykoupena mnohem nižším výkonem. Hodnoty se také snižují s přibývajícími připojenými stanicemi. To zase ukazuje na limitaci výkonu systémem.

3.4.3 Využití sítě

Byl měřen počet bytů přijatých serverem průměrně na jeden vykreslený snímek pro měřené rozlišení a počet klientů. Testy byly provedeny na verzi systému pro statickou scénu.

	320x240	640x480	800x600
2 klienti	184387	722663	945683
3 klienti	226832	792792	1148416
4 klienti	228020	911129	1215575

1.1.3.1 – Průměrný počet bytů přijatých serverem na jeden snímek

Z tabulky je vidět, že s přibývajícími klienty se náročnost na síť zvyšuje jen minimálně. To je způsobeno RLE kompresí. Scéna je rozdělena na části. Pokud je tedy připojen další klient, vykresluje část scény, kterou předtím vykresloval jiný klient. Klient, který vykresloval tuto část původně má teď ale místo této části ve výsledku jen barvu pozadí, která se zkomprimuje do minima dat. Díky tomuto faktu je zátěž sítě při velkém počtu klientů menší než bylo očekáváno.

Pokud vezmeme maximální rychlost přenosu přes 100Mb/s síť (12,5MB/s) a vydělíme jí množstvím dat přijatým za jeden snímek, získáme teoretické maximum FPS na této síti. Tohoto čísla není možné nikdy dosáhnout, protože nikdy na této síti nedosáhneme jejího plného využití. Také nějaký čas zabere vykreslení a kódování snímků, stejně jako jejich skládání a zobrazení.

	320x240	640x480	800x600
2 klienti	31,16173	9,804621	7,014665
3 klienti	23,5561	6,743212	4,737668
4 klienti	19,89876	6,938681	3,830758

1.1.3.2 – Maximální možný počet snímků za vteřinu, který by bylo možné dosáhnout na 100Mb/s síti při využití její maximální teoretické rychlosti

Pokud porovnáme tuto tabulku a grafy naměřených FPS je vidět, že systém dosahuje tak maximálně poloviny teoretického maxima. Ukázalo se, že nejpomalejší operace v systému je RLE komprese. Zde je nutno projít celý obraz pixel po pixelu a to několikrát za vteřinu. Proto se také výkon tak prudce snižuje se vzrůstajícím rozlišením.

4 Závěr

Účelem práce bylo vytvořit Sort-Last systém pro paralelní rendering. Měly být možné dvě situace. Jedna, kdy je scéna uložena u klienta a server jen řídí a zobrazuje výsledky vykreslování (statická scéna). A druhá, kdy jsou všechna data uložena na serveru a ten je rozesílá dle potřeby klientům (dynamická scéna). Obě tyto možnosti jsou v systému implementovány. U dynamické scény je možné ve vysoké míře řídit i vykreslování dat. U statické scény je možné prakticky jenom nastavit kameru.

Systém měl být schopen zobrazovat i poloprůhledné objekty. To je možné u obou verzí systému. Bohužel může docházet k chybám ve vykreslování, kdy se do poloprůhledného objektu přimíchá na jedné stanici i barva pozadí, která by tam neměla být. Jde o barvu pozadí, která je přimíchána do poloprůhledného systému při vykreslování u klienta.(více o tomto problému viz 2.3 - poslední část).

Asi největším zklamáním systému je jeho výkon. Uspokojivé výsledky podává jenom do rozlišení 320x240. Ve vyšších rozlišeních už je jeho rychlost v řádu jednotek FPS, což je dost málo. Ačkoliv bylo počítáno s vlivem sítě propojující stanice se serverem, nakonec se nejužším místem systému s největší pravděpodobností stala komprese částečných obrazů. Což je paradoxně právě metoda pro snížení nároků na síť. Celá komprese je prováděna procesorem, takže systém je na něm velmi závislý, ačkoliv by nejvíc práce měla udělat grafická karta při vykreslování geometrie.

Systém má ještě mnoho možností dalšího rozvoje. Asi nejzákladnější pro praktické využití je jeho zrychlení. Jsem přesvědčen, že úpravami v systému je možno dosáhnout mnohem lepších výsledků. Další věc je rozšíření jeho možností pro rendering. V tuto chvíli je možné vykreslovat pouze pomocí fixed function pipelíne(FFP), což je nejjednodušší možnost vykreslování, v tuto dobu postupně nahrazovaná vertex a pixel shadery (v nových DirectX 10 už bude dokonce vypuštěna úplně). Možnost využití shaderu v systému by tedy byl další logický krok v rozvoji systému. K tomu i mnoho nastavení FFP pořád není možné v systému využít.

Použitá literatura

[1] Kirihata, Y., Leigh, J., Xiong, C., and Murata, T. A sort-last rendering system over an optical backplane. CITSA 2004, 2004.

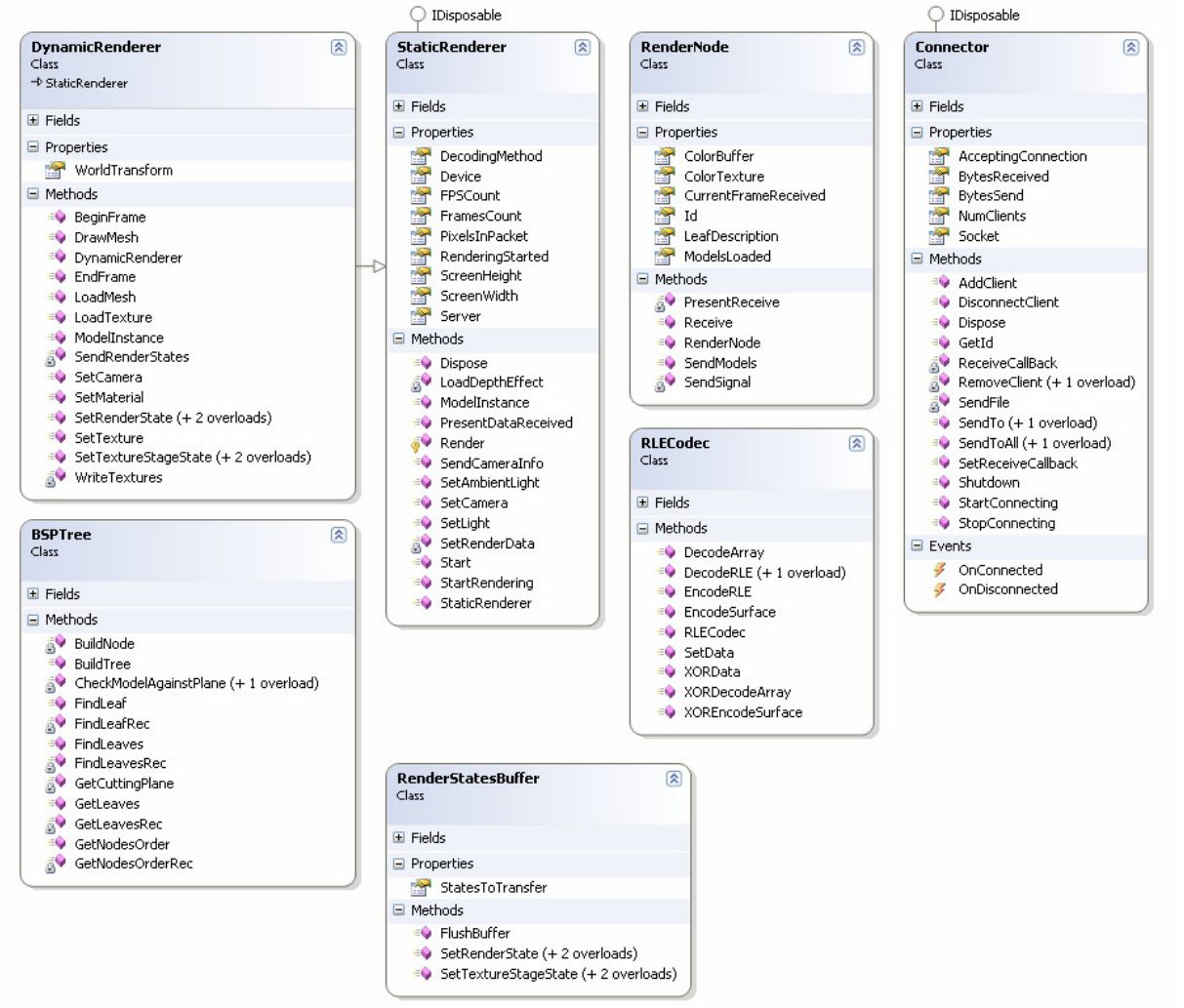
[2] James Ahrens and James Painter. Efficient sort-last rendering using compression-based image compositing. *Proceedings of the Second Eurographics Workshop on Parallel Graphics and Visualization*, 1997.

[3] K. L. Rudrajit Samanta, Thomas Funkhouser and J. P. Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. *Eurographics / SIGGRAPH Workshop on Graphics Hardware '00*, stránky 97–108. Addison-Wesley Publishing Company, Inc., 2000.

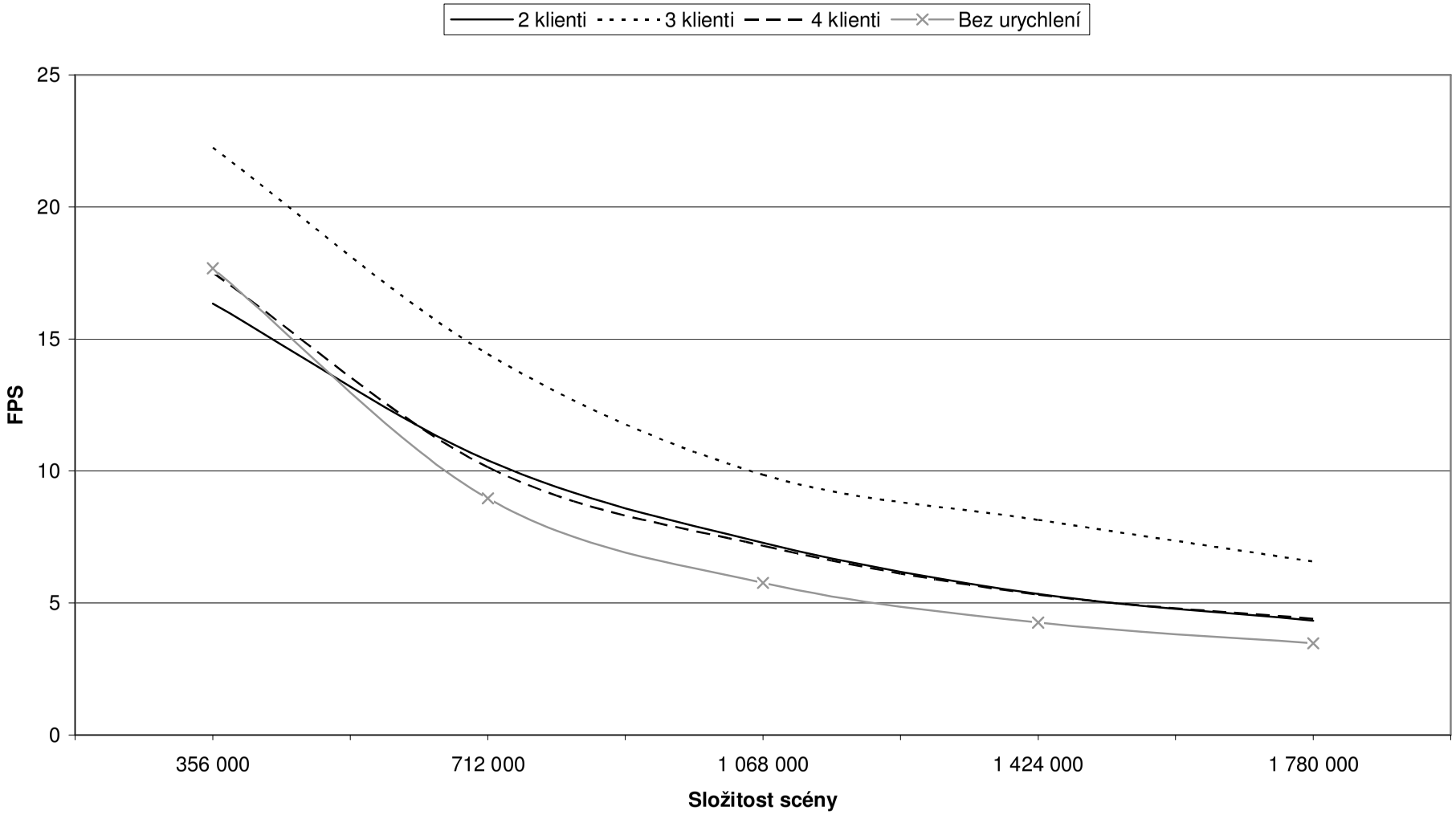
[5] Ma, K.-L., Painter, J.S., Hansen, C.D. and Krog, M.F., Parallel Volume Rendering Using Binary-Swap Compositing, *IEEE CG&A*, stránky 59-68, 1994.

PŘÍLOHY

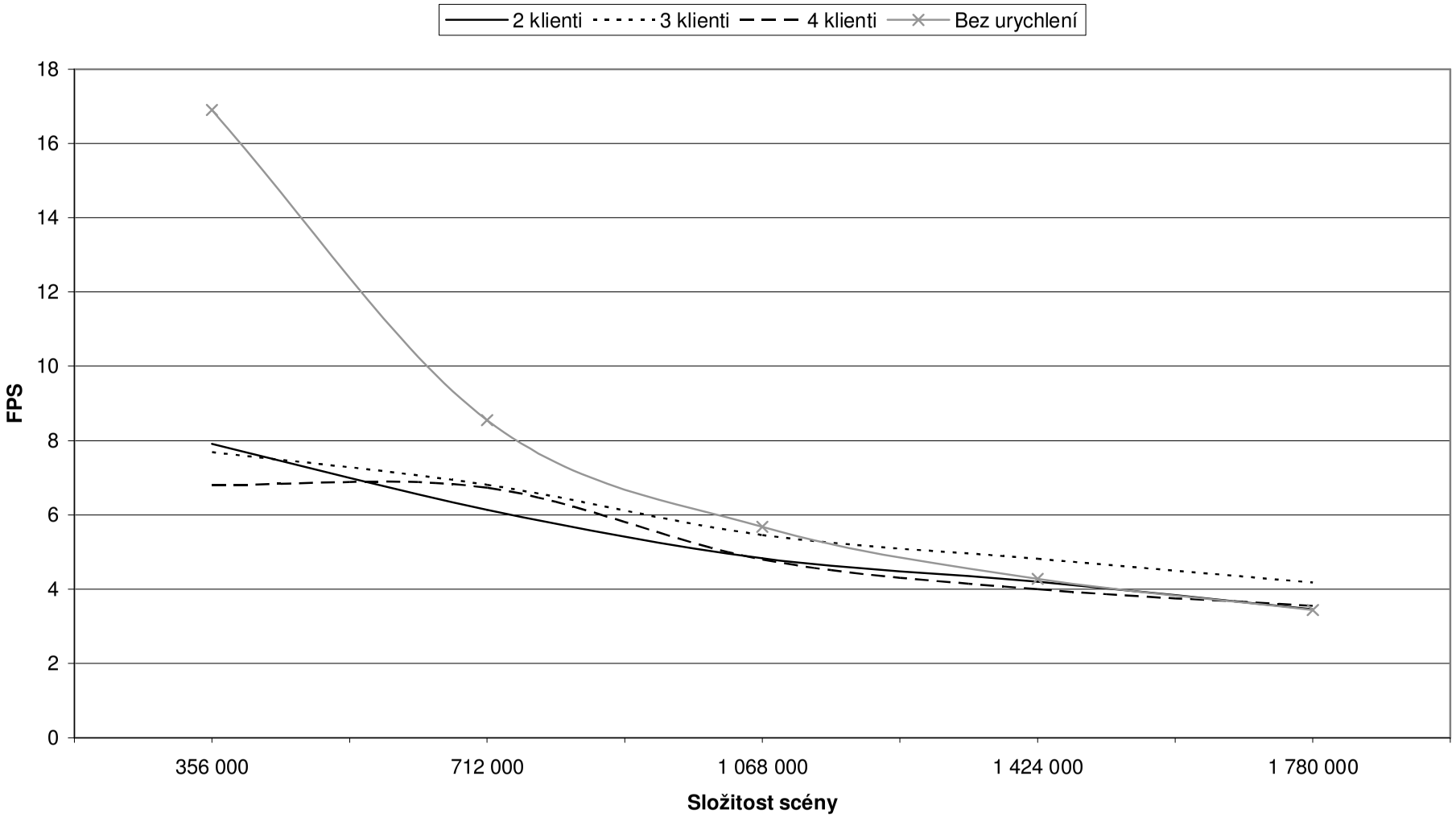
3.1.1.1 – Třídy využívané aplikací serveru



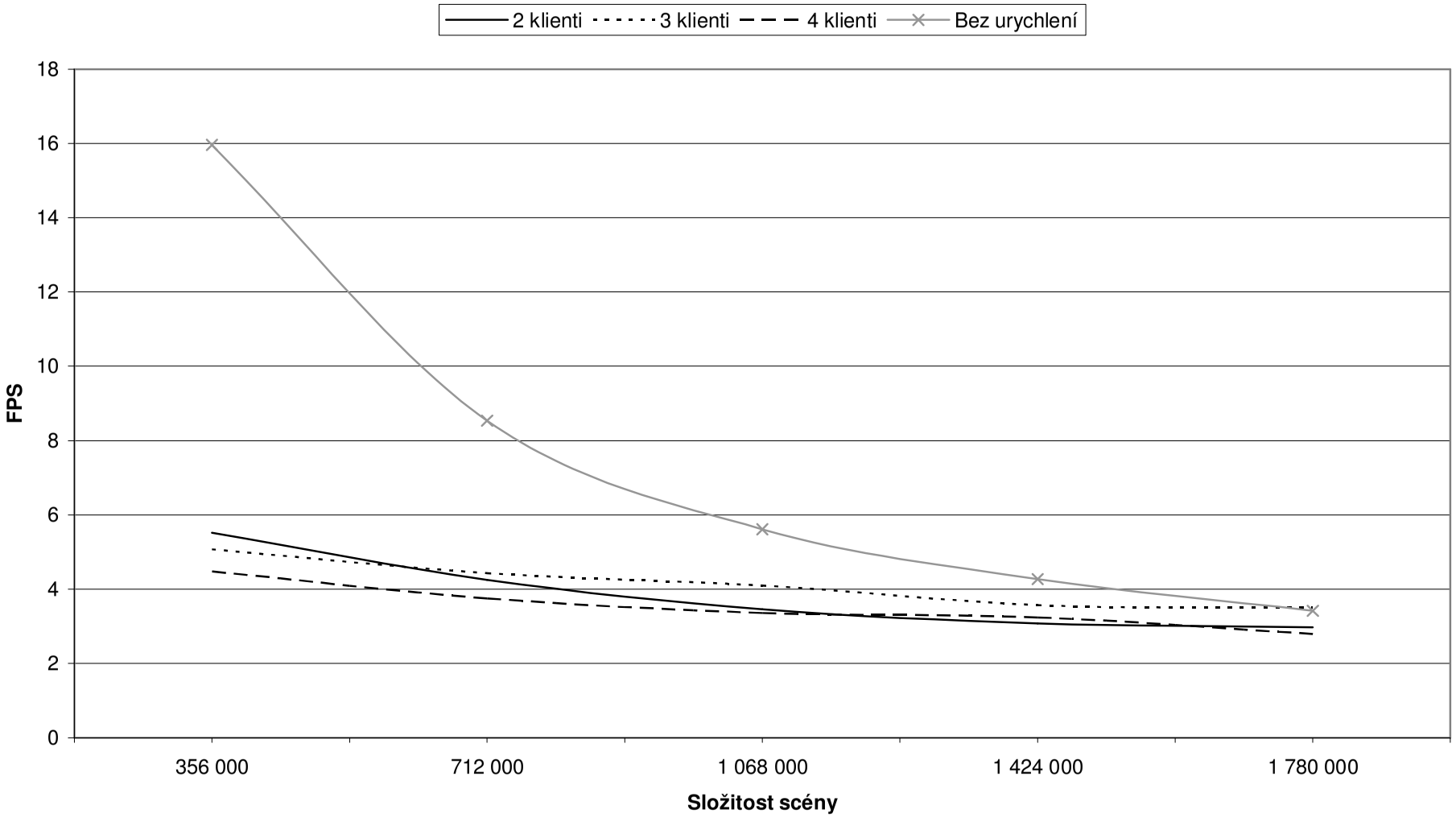
3.4.2.1 - Rychlost zobrazování statické scény - 320x240



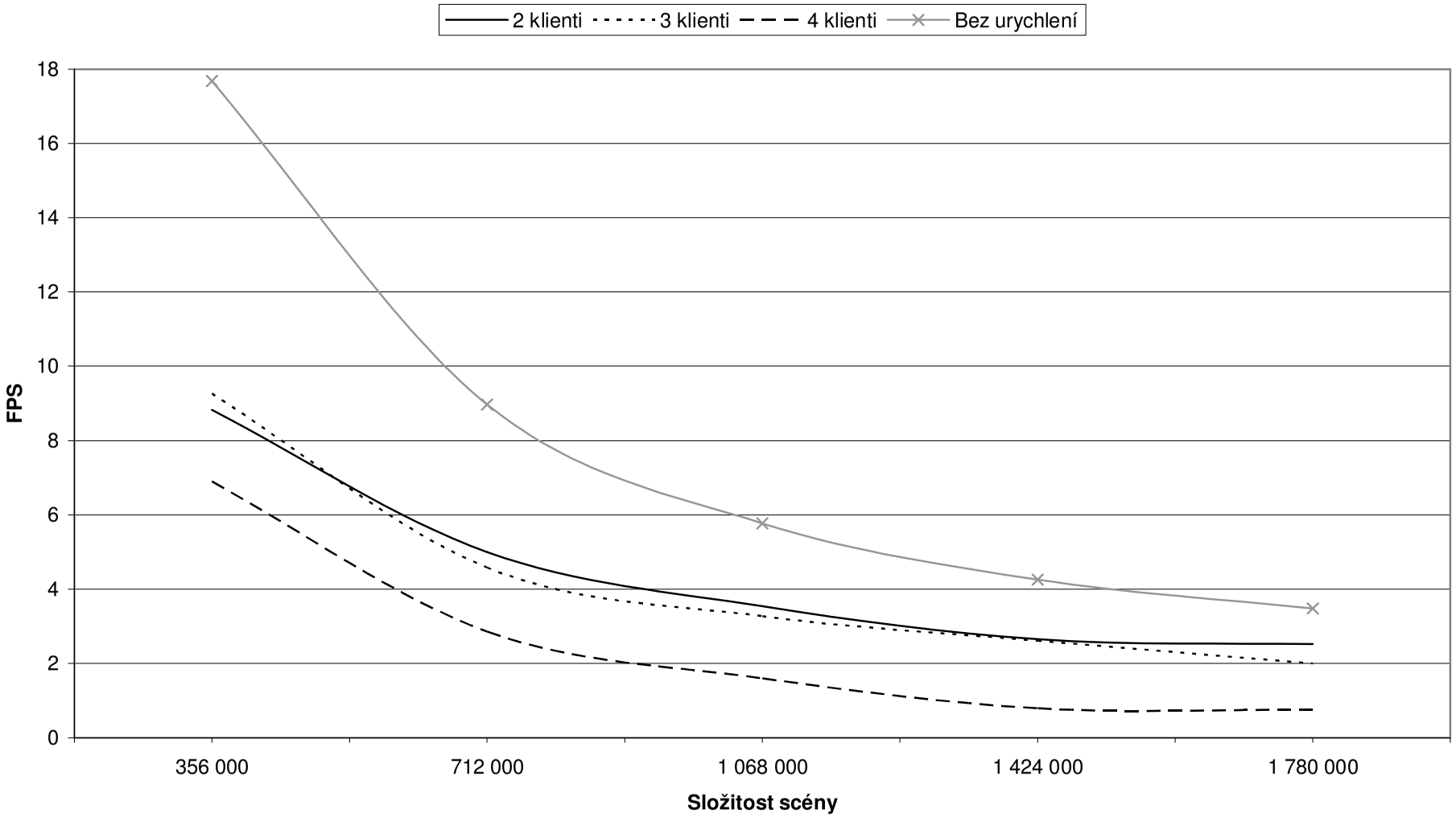
3.4.2.2 - Rychlost zobrazování statické scény - 640x480



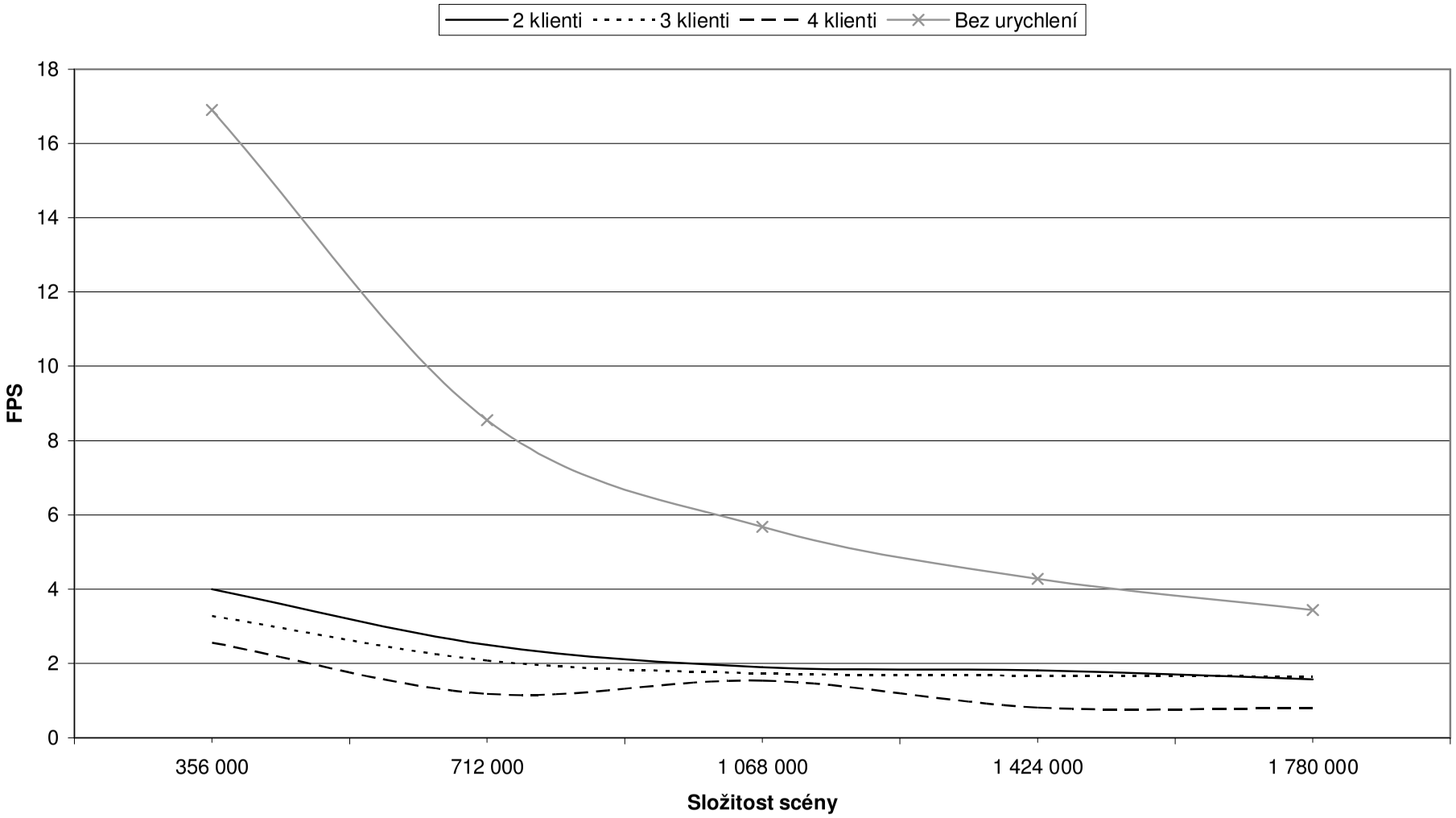
3.4.2.3 - Rychlost zobrazování statické scény - 800x600



3.4.2.4 - Rychlost zobrazování dynamické scény - 320x240



3.4.2.5 - Rychlost zobrazování dynamické scény - 640x480



3.4.2.6 - Rychlost zobrazování dynamické scény - 800x600