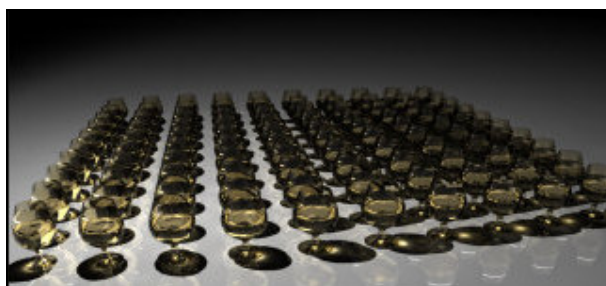


Distribuované 3D renderování



*Autor Johan Thorngren
([Brazil Rendering system](#))*

Autor: Jan Kříž
Datum: 9.1.2003
e-mail: jan.kriz@a-architectus.com

OBSAH

1	Úvodem.....	3
2	Bližší pohled na distribuované renderování.....	3
2.1	Snímková koherence.....	4
2.2	Rozdělení dat mezi uzly.....	5
3	Příklad distribuovaného animačního <i>render</i> systému.....	7
3.1	Teoretická koncepce renderovací farmy.....	7
4	Implementace distribuovaného renderování v současných 3D aplikacích.....	9
4.1	Používané moduly.....	9
4.2	Externí (3rd party) renderovací moduly.....	11
5	Závěr.....	13
6	Příloha.....	13
7	Literatura.....	15

1 Úvodem...

Hned na úvod zmíníme vlastní definici renderování či renderingu.

Rendering = proces, kdy 3D animační/renderovací software za pomoci počítače přebírá definované charakteristiky a informace 3D geometrie, světla, materiálů na objektech, prostředí, odrazů/lomů světla aj. a převádí tyto charakteristiky na základě předem definovaných stínovacích algoritmů na výsledný 2D obraz. Nyní bych zmínil co vlastně umožňuje práci v animační/renderovací oblasti výrazně usnadnit. Vzhledem k tomu, že tato lidská činnost je charakteristická nejen tvořivostí, ale i značným iterativním postupem, rychlá odezva výsledků práce je nanejvýš žádoucí a platí zde pravidlo: Čím více počítačů v síti, tím lépe. Dostáváme se tak k definici **síťového**, čili **distribovaného renderování**:

Tím rozumíme renderování za použití více než jednoho samostatného počítače, obvykle několik desítek, které jsou spojeny do společné sítě, aby vykonaly renderovací úlohu v co nejkratším čase a to s cílem distribuovat výpočetně náročnou úlohu mezi jednotlivé uzly. Často je tento postup vhodný při požadovaném výstupu až v stovkách tisíců snímků či oken (frames).

Je zřejmé, že právě dnešní doba, kdy hardware jde ve vývoji tak rychle dopředu, dohání běžné stolní počítače výkon silnějších serverů a lze tak sledovat stále větší množství amatérských, ale i profesionálních celovečerních filmů, které by bez distribuovaného renderování vyžadovaly desítky let tvorby.

2 Bližší pohled na distribuované renderování

Vysoce kvalitní počítačové animace vyžadují intenzivní výpočetní sílu. Dnes existují dva možné způsoby pro snížení výpočetní doby nutné k produkci 3D digitálního obsahu. Jedním z nich je neustálé vylepšování a vymyšlení sofistikovanějších renderovacích algoritmů, které budou těžit ze základních vlastností obrázků (čili vlastností pixelů – základní bod 2D grafiky) a budou schopny proces výstupu obrazu urychlit nebo z použití více počítačů v síti, spoléhající na pokročilý hardware jako například multiprocessorové platformy nebo grafické karty s rychlými VPU (**V**isual **P**rocessing **U**nit). Dále bych zmínil kombinaci obou směrů, jež vede k rapidnímu snížení výpočetního času. Ne vždy je však rozdělení dat mezi procesory různých pracovních stanic nejefektivnější (*záleží totiž i na způsobu rozdělení*). Vhodnými testy k prověření efektivnosti síly počítačů v síti je tzv. **Raytrace algoritmus a Radiozita**

(*Raytracing* – zpětné sledování paprsků - funguje tak, že sleduje světelný paprsek od konce dopadu na povrch objektu zpět ke světelnému zdroji. Každý počítaný paprsek pak bude jistě přispívat k podobě závěrečného obrázku. Naopak by to bylo náročné a neefektivní - vyžadovalo by to velmi mnoho paprsků vycházející ze zdrojového objektu. Raytracing se tak vlastně zabývá jen paprsky majícími vliv na konečný vzhled obrázku. Hovoří se tedy někdy o **efektivním Raytracingu**.

Výstižným přirovnáním k *Radiozité* by mohlo být tzv. "globální osvětlení" neboť se snaží o celkové šíření světla v rámci celé scény. Jde vlastně o odražené světlo (*všesměrové*) mezi objekty navzájem. Vzhledem k tomu, že způsob napodobení Radiozity Raytracingem při osvětlení komplexní scény by vyžadovalo mnoho času (*počítač by musel generovat mnoho paprsků a propočítat je společně s mnoha objekty*), byla vynalezena právě metoda Radiozity. Ta spočívá v uchování hodnot osvětlení při šíření světla přímo na povrchu objektů. Vynalezena byla tzv. **Stochastická paprsková metoda**, která vystřídala dřívější *Monte Carlo metodu*, mající za úkol simulovat radiozitu. Sofistikovanější tzv. **Galerkinova** metoda, počítá již s různou radiozitou u odlišných částí povrchu. Právě tyto algoritmy se často používají při vytváření fotorealistických vysoce kvalitních snímků pro různé účely. Rendering několikasekundové animace tak může trvat hodiny i dny. U komplexních animací to může být nepřekonatelný problém.

Základním bodem je pak paralelizace algoritmu snímkové koherence v síti (určitá souvislost mezi po sobě jdoucími obrazovými daty) a s tím spojené rozložení problému a jeho distribuce mezi pracovní stanice. Vyrovňování zátěže by mělo probíhat automaticky, s předpokladem snahy snížit náklady na komunikaci v ethernetu, která bude nižší ve srovnání s propojenou komunikací mezi multiprocessorovými stanicemi (pro pokročilé uživatele viz **integrace mental ray v 3ds max 6**).

2.1 Snímková koherence

V této kapitole bych se stručně zmínil o způsobu implementace tohoto algoritmu. Vzhledem k minimální změně následujícího snímku od toho předchozího (porovnáním pixelů snímků) lze zjistit, že velká část pixelů se nemění. Z toho plyne důsledek, že tyto pixely zřejmě nebude nutné znovu přepočítávat do dalšího snímku. Můžeme tak usoudit na prediktivní povahu tohoto algoritmu. Porovnáním snímků tedy zjistíme, které pixely se vůbec nezměnily a ty budeme do dalšího snímku považovat za neměnné, a tudíž nebudou přepočítány. Aby bylo možné dosáhnout tohoto úkolu, je objektový 3D prostor rozdělen na tzv. *voxely (krychle)* za

pomocí stejnoměrného prostorového rozdělení. Jakmile jsou ve 3D scéně použity světelné paprsky, algoritmus koherence zaznamenává, kterými voxely paprsek prochází. Je dobré si uvědomit, že také daným pixelem může procházet více paprsků (odražené, lomené, paprsky stínu atd.). Jestliže je u konkrétního voxelu zaznamenána změna (*například objekt se pohybuje*), všechny pixely, jejichž paprsky procházejí tímto voxellem, **budou** aktualizovány. V následujícím odstavci se podíváme na jednom z možných principů, jak tento algoritmus aplikovat na paralelní zpracování více uzly.

Pozn.: V této části se počítá se statickou kamerou, pohybují se jen objekty ve scéně (viz [Rozdělení dat mezi uzly](#))

2.2 Rozdělení dat mezi uzly

Cílem tohoto bodu je nalezení metody pro dekompozici problému mezi uzly. Ústředním bodem této metody je operace s pojmem *Intenzita pixelu*, která je rovna součtu 3 složek:

- Lokální intenzity pixelu způsobené přímým osvětlením (přímým paprskem)
- Součinu konstanty odrazivosti materiálu (reflektivita) a hodnoty odraženého světla od objektu s daným materiálem
- Součinu konstanty lámání světla v materiálu (refrakce) a hodnoty přeneseného světla skrze objekt .

Pak se postupuje tak, že se rozdělí jeden snímek na části s danou velikostí, které jsou vypočítávány **paralelně**. O tento proces se stará **master procesor (server)**, stejně jako o sběr informací o pixelech a jejich ukládání do externího souboru. Komunikace pak probíhá směrem od master uzlu k **slave** uzlům a zpět.

Je tedy jasné, že důležitým krokem je sladění využití snímkové koherence společně s rozdělením výpočetních úloh mezi různé uzly a využít tak i paralelní zpracování. Při práci s kamerou se její pohyb rozdělí na sekvence, na něž je možné aplikovat jak algoritmus koherence tak distribuci úloh. Paralelní zpracování těchto sekvencí – **rozdělení sekvencí** – zahrnuje přerozdělení *celých* snímků mezi dostupné procesory, takže každý dostane k výpočtu určitou podsekvenci celé animace. Aby byla snímková koherence plně využita, je nutná pevně daná následnost (*posloupnost*) snímků jeden po druhém (*tedy nikoli například renderování jen lichých nebo sudých snímků*). Nevýhodou této metody může být situace, kdy každému procesoru je přiřazen pevný (*statický*) počet snímků, což může vést k nevyrovnané meziprocessorové zátěži. Je totiž možné, že CPU mají různé rychlosti – což je povětšinou – a také, že každá jednotka může pracovat na snímku s různou složitostí (více raytrace paprsků,

více práce se světly apod.). Tento problém se dá vyřešit tak, že ukončí-li procesor svou činnost na podsekvenci dříve než ostatní, přiřadí se mu část další podsekvence jiného procesoru.

Další metoda spočívá v rozdělení jednoho snímku na podoblasti – **rozdělení snímku** - , které budou přiděleny různým procesorům. (*Narozdíl od předchozí metody, kde byly jednotlivým procesorům přiděleny celé snímky*). I zde však platí, že ne všechny snímky jsou vypočítávány stejně dlouho, vzhledem k různé náročnosti podoblastí snímků. Jako v předchozím případě je možné pro nečinné procesory přidělit další část (podoblast) snímku, na níž pracuje jiný procesor. Výhodou této metody je menší náročnost na paměť, neboť její nutně využitelná část je úměrná velikosti podoblasti snímku, na níž procesor v daném okamžiku pracuje. (Čím více procesorů, tím menší část snímku připadá na jedno CPU). Z toho lze usoudit, že rozdělení snímků na menší podoblasti, kterých je méně než počet procesorů (*například z obrázku v rozlišení 1024x768 na podoblasti o velikosti 64x48 při osmi procesorech, tj. na každou případnou dvě tyto podoblasti*), vede k lepšímu vyrovnání zátěže. Dokončí-li rychlejší procesor svou podčást, požádá o další.

Existují samozřejmě i další metody, avšak jde vesměs jen o různé kombinace výše zmíněných. Četné výsledky však dokazují, že doba nutná k vyhotovení animace při použití raytrace algoritmu, je nejnižší při vhodné kombinaci paralelního zpracování a algoritmu koherence snímků.

Pozn.: k multi-threadingu

- Softwarové balíky pro 3D animaci mají v sobě zabudovaný vlastní renderovací systém, který však nebývá co do kvality a rychlosti srovnatelný s přídatnými moduly pro rendering dostupnými na trhu. I když většinou při renderování je spuštěn jen jeden master render proces, který obhospodařuje celý rendering, k vytváření výsledného digitálního obsahu lze využít každý zapojený procesor. Zde je nutné zmínit ekonomickou stránku věci. Často je licence externího renderovacího systému odvozována právě od počtu počítačů. Například velmi výkonný server **Brazil rendering system v1.0**, určený pro animační software **3ds max** stojí kolem 1200USD (k lednu 2003) a licence je pro tři počítače (*jeden master workstation s grafickým rozhraním pro ovládání procesu, tj. přidávání, odebírání CPU do/z renderovacího procesu, včetně přidělování priorit atd. a dva renderovací pomocné počítače*). Každý další počítač zapojený do procesu vyžaduje dodatečnou licenci. Další nevýhodou může být zvýšený nárok na RAM paměť. Čím více procesorů je ve stejném okamžiku v procesu zapojeno, tím více RAM paměti je využito. Často jako mezník v počtu procesorů je osm. Například 32 bitový renderovací stroj **Mental Ray** (použitý již v několika úspěšných filmových sekvencích a ve verzi 3ds max 6 plně integrovaný) dovede při větším počtu procesorů (8+) využít celý

adresní prostor, a tak například při 16 procesorech vede virtuální prostor o 2GB k nedostatku velikosti zásobníku každého z CPU bez ohledu na RAM paměť.

3 Příklad distribuovaného animačního *render* systému

Některé společnosti zabývající se tvorbou digitálního obsahu vytvářejí pro své účely tzv. *Renderfarms* (něco jako „farma na renderování“). Tou se rozumí rozsáhlý prostor počítačů schopných zapojit se do pomocného výpočetního procesu a rozdělit tak výpočet snímků mezi počítače. V širším smyslu slova se pak cílem těchto projektů stává produkce Internetově založených renderovacích sítí - „*renderfarmy*“ - s použitím Java komponent a zpřístupnění tak nevyužitých počítačů na Internetu.

3.1 Teoretická koncepce renderovací farmy

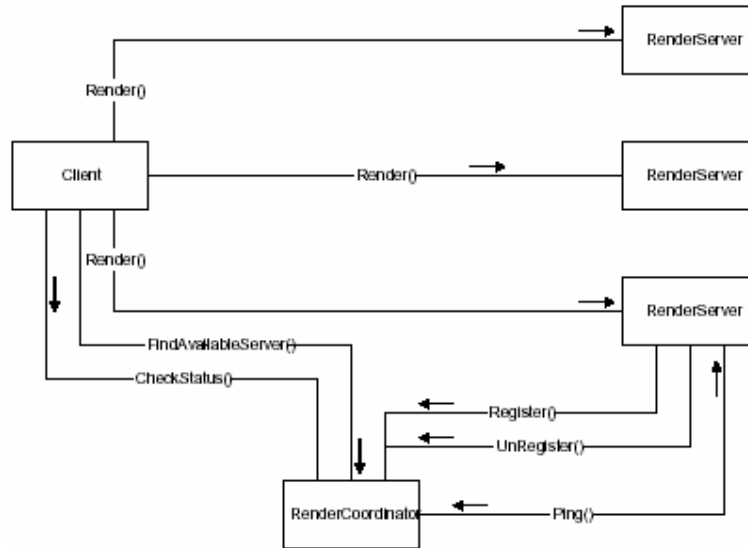
Renderovací farma (dále jen **RF**) se skládá ze tří logických fází: Shromažďování, renderování a animování. První fáze – **shromažďování** - je typická tím, že uživatel vytvořil sérii souborů, které definují snímky animace ve formátu nějakého *raytrace* enginu nebo renderovacího balíku (POV-Ray, Pixar's Renderman, Mental Ray, Brazil render...). Poté co uživatel v přidruženém klientském programu definuje náležitosti výsledného obrazu (rozlišení, počet snímků za sekundu, související soubory), je možné začít fázi **renderování**.

Tato fáze obsahuje následující komponenty:

- RenderCoordinator – ten spravuje a udržuje seznam všech dostupných render serverů, které mohou být poskytnuty klientovi při požadavku na výpočet určitých snímků animace.
- RenderServer – ten je umístěn na stroji s renderovacím softwarem. Spravuje konfiguraci a plní požadavky úloh ve frontě pro render na seznamu úloh ve formátu FIFO.

Je-li možné začít fázi renderování, upozorní klient *RenderCoordinatora* požadavkem na zajištění všech dostupných serverů. Ten mu vrátí „certifikát“ o volných serverech v síti. Klient pak metodou *Render(definice_scény)* volá příslušné servery s požadavkem na začátek procesu. Není-li daný server dostupný, vrátí klient koordinátorovi certifikát o neúspěšném volání (jako parametr je nedosažený *RenderServer*). *RenderCoordinator* se pak sám snaží kontaktovat daný server. Nepovede-li se to, odstraní jej ze svého seznamu. Obdrží-li *RenderServer* požadavek, začne Renderovací proces. Může dále shromažďovat další požadavky a to ve formě fronty FIFO, tj. požadavek je zařazen na konec fronty a čeká až budou vyřízeny požadavky před ním. Render pokračuje ve své činnosti až do vyprázdnění

fronty. Při požadavku uživatele na ukončení činnosti *RenderServeru*, je právě rozpracovaný požadavek dokončen a další požadavky ve frontě jsou předány ostatním běžícím serverům. Následující obrázek popisuje obecnou koncepci renderování (dle M.K.Novi – *Distribuovaný renderovací systém*)



obr č.1 – Fáze renderování

Následující etapa je **Animace**, kdy vyhotovené obrázky jsou předány úkolu pro zpracování sestavení obrázků dohromady. Tato etapa předpokládá úspěšné dokončení předchozí fáze.

Stejně jako fáze renderu, obsahuje i animační etapa obdobné komponenty, tj.:

AnimCoordinator, AnimServer. Jejich smysl je podobný - *AnimCoordinator* spravuje seznam všech dostupných *AnimServerů* a zajišťuje, aby jednotlivé snímky náležící k jedné animaci došly právě k jednomu serveru. *AnimServer* má pak za úkol jednotlivé snímky poskládat v ucelenou animaci (výsledným video formátem může být *.AVI*) za použití programu jako je *Targa Animator*

Na začátku tedy *RenderServer* předá zabalené obrázky s příslušnými informacemi *AnimCoordinatorovi*. Po vyhotovení certifikátu předá *AnimCoord.* příslušné soubory na zpracování dostupnému *AnimServeru*, který k sobě obrázky poskládá (má samozřejmě informace o vzájemné posloupnosti jednotlivých obrázků). Musí také počkat, než všechny nutné obrázky náležící k jednomu projektu dojdou na server. Pak je úloha zařazena obdobně jako u fáze renderu do fronty a čeká na vyhotovení.

Nevýhodu těchto systémů lze spatřovat v nevyvážené zátěži. Zatímco jeden server může být dramaticky zatížen, druhý nemusí pracovat vůbec. Záleží pak na použití algoritmu vyrovnání zátěže (*spíše než náhodné přidělování úkolů*), kterým *Coordinator* distribuuje úkoly jednotlivým serverům. V dnešní době však některá studia od takto distribuovaného zpracování (*alespoň u fáze animace*) upouští, neboť poskládání obrázků není tak výpočetně náročným úkolem jako je například renderování. Vyplatí se pak obrázky zpracované v renderovacím procesu ručně zařadit do fronty v příslušném programu.

V tomto odstavci šlo tedy o popis jakési teoretické koncepce, která se s postupným vylepšováním algoritmů pro vyrovnávání zátěže může stát novým standardem. Při snaze o automatizaci a začlenění algoritmů pro vyrovnávání zátěže uzlů, dle nichž se úkoly stanicím přidělovaly, se autoři často potýkali s problémy deadlocků (*a jak víme, z důvodu obtížné algoritimizovatelnosti preventivních opatření je tento problém zatím otevřený*). Zatím se tedy nejčastěji používá interních podnikových sítí s předem nadefinovanými IP adresami počítačů využitelných v síti. Dynamický proces přidělování úloh, jakým byla RF v tomto odstavci popsána, je již v některých 3D aplikacích zabudován .

4 Implementace distribuovaného renderování v současných 3D aplikacích

Teorie distribuce dat a úkolů při renderování je řešena v různých aplikacích odlišně. Samozřejmě je ale nutné podotknout, že jádro zůstává stejné a větší část implementace je pro aplikace totožná. V následující části bych se zaměřil na to, jak současní výrobci 3D animačního softwaru tuto problematiku řeší.

Proces začíná tím, že úloha distribuovaného renderování (*poté, co je scéna připravena, nasvícena, obohacena o textury... a spuštěn povel Render*) je softwarem rozdělena mezi renderovací servery a snímek je po částech – dle smyslu kapitoly [Bližší pohled na distribuované renderování](#), přidělen dostupným serverům. Dokončený výstup v podobě inkrementálně očíslovaných snímků bývá uložen ve sdíleném adresáři. Pro tuto činnost se využívá funkce základních renderovacích modulů.

4.1 Používané moduly

Část softwaru, často nazývaný **network manager**, má na starost kromě běžného přidělování úloh všem pracovním uzlům a řízení priorit procesů také zjišťování zátěže uzlů. Jsou

monitorovány nevyužité kapacity a práce je tak přerozdělována všem dostupným stanicím s cílem zapojení maximálního počtu uzlů s maximálním využitím. Tento *manažer* kromě kontroly zpracování úlohy také komunikuje prostřednictvím **klienta monitorování fronty** (*Queue monitor client*) za účelem:

- plánování úloh
- konfigurace serverů.

Klient je jakýmsi grafickým rozhraním uživatele k ovládání síťového renderování a lze jím kontrolovat stav zpracování renderingu i z kteréhokoli počítače připojeného k Internetu.

Mezi jeho hlavní úkoly patří: Aktivace/Deaktivace úloh, reorganizace nebo odstranění úloh a totéž lze provést i se servery.

Dalším modulem bývá **server modul** (*běžící jako služba na pozadí pro lepší odezvu a postup renderovací úlohy*), jenž se spouští na stanici, která má být použita jako renderovací server (*tím může být i manažer, avšak z důvodu rychlosti manažera to nebývá časté*). Ten odesílá svou IP adresu *síťovému manažeru*, který ji zaregistruje jako dostupný uzel při renderování. V daný okamžik registrovaný uzel už jen „očekává“ příkazy *manažera* pro vykonání úlohy. Ty jsou (*jak již bylo řečeno*) po dokončení v podobě snímků odesílány do společného, sdíleného adresáře.

Posledním modulem je vlastní **jádro aplikace** (*spustitelný soubor 3D aplikace*). To je v okamžik začátku renderování spuštěno v každém uzlu lokálně. Můžeme jej tedy považovat za „*trigger*“ zpracování úloh v daném uzlu poté, co je přijat podnět k vykonání od *manažera*. V softwaru určeném pro renderování bývá často zabudován modul pro měření výkonnosti jednotlivých serverů. Tato pomůcka – **Index výkonnosti** (*Performance index*)- je užitečná v případě, kdy chceme zjistit, které servery přispěly pro finální výpočet nejvíce a které naopak nejméně. Například škálou <0-1> lze označit od nejrychlejších strojů (1) po nejpomalejší (0). Stroje bývají hodnoceny na základě toho, jakou dobu spotřebovaly na daný snímek. Kumulovaná hodnota této doby dělená počtem snímků pak dává průměrnou dobu výpočtu na jeden snímek.

Pozn.: Ne vždy je nejrychlejší procesor zárukou nejvyšší hodnoty indexu výkonnosti. Přístupují-li servery k nezbytným souborům scény (mapy, textury, obrázky atd.), pak záleží také na propustnosti sítě. Jsou-li přenášeny veliké soubory – mpeg, avi, jakožto podčásti animace využitě ve scéně – pak největší výhodu (i index výkonnosti) může mít server, který tato data nahrává z lokálního disku, neboť ostatní stanice budou podstatnou dobu nahrávat tyto soubory, kdežto server s lokálními daty může již delší dobu renderovat..

Dalším dynamicky běžícím modulem, který lze ovládat za běhu je **Dialog přidělování úloh**.

Jeho úkolem je podávat při renderování informace o stavu zaneprázdněnosti jednotlivých serverů. Můžeme během stavu zpracování zjišťovat, který server pracuje, jak je vytížen, odebírat mu úlohy a přidělovat je méně zatíženému serveru nebo přidávat nové stanice a ty zapojit do procesu.

S tím úzce souvisí otázka priorit. Ta bývá koncipována tak, že čím nižší číselná hodnota, tím vyšší priorita. Vezměme do úvahy například 2 procesy (úlohy). Proces A s prioritou 10 je již nějakou dobu postoupen renderování a proces B s prioritou 5 je předán síťovému manažeru. V tento okamžik je proces A suspendován a čeká na dokončení procesu B. Po jeho vykonání proces A pokračuje dále. Je-li proces ohodnocen jako kritický, je poslán na začátek fronty automaticky.

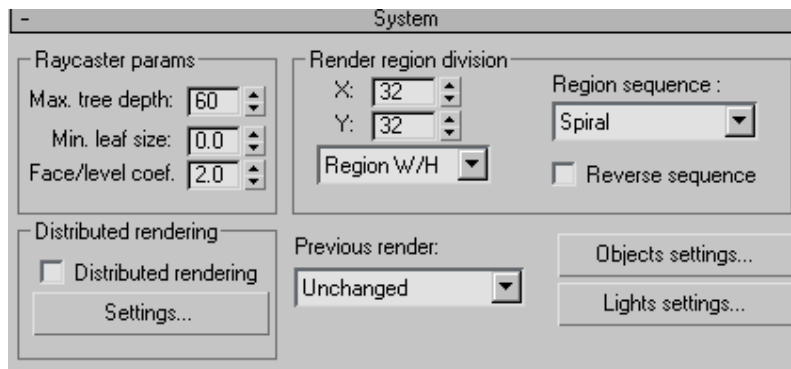
4.2 Externí (3rd party) renderovací moduly

V tomto pojednání jsem již zmínil, že pro kvalitní fotorealistický výstup se dnes používají externí pluginy (zásuvné moduly), které vzhledem k otevřenému rozhraní 3D aplikací lze používat společně s touto aplikací a nahradit nativně zabudovaný renderer, který nebývá pro profesionální výstupy nejvhodnější (*at' už z důvodu rychlosti nebo kvality obrazu*).

V následujícím odstavci bych popsal způsob a základní technologie využití distribuovaného renderování v těchto modulech. Jako příklad využití distribuce požadavků mezi síťové servery bych uvedl dva systémy - **Vray** a **Brazil rendering system**. Neexistuje samozřejmě jediný způsob jak tento problém vyřešit, ale u těchto systému (*a nejen těch*) se často hovoří o obecně využitelném způsobu: Rozdělit snímek na malé oblasti „*buckets*“ (*z anglického bucket=kelík, kyblík...*), které jsou podle určitého algoritmu rozděleny mezi renderovací servery. Poté je výsledek poskládán do výsledného obrazu. Odtud pojem bucket rendering. Základní myšlenkou je rozdělení správy distribuce mezi servery a klienty.

- **Render klient** – počítač, který uživatel používá při přípravě práce na renderování. Rozděluje snímek na „*buckety*“ a rozesílá je mezi servery. Klient kontroluje stav zpracování, posílá další potřebné informace (*soubory map, textur*) a přijímá výsledky. Lze zde spatřovat analogii s *Network manažerem* z [předchozí kapitoly](#) o interně zabudovaných modulech 3d aplikací určených pro distribuci požadavků. Je-li *bucket* dokončen a poslán zpět ke klientovi, je zobrazen na klientském displeji a poslán další požadavek na zpracování jiného *bucketu* na serveru.
- **Render server** – z předchozího je zřejmé, že server přijímá požadavky od klienta a plní je v podobě renderovaných *bucketů*, které jsou zasílány zpět ke klientovi. Naplňuje se tak základní představa klient-server modelu.

Následující obrázek ukazuje možnosti nastavení v systému *Vray*.



obr č.2 – kontrola systému *Vray* v rámci aplikace *3ds max*

Za zajímavost stojí zmínit sekci *Render_region_division*. Ústřední část distribuovaného renderovacího systému *Vray* je bucket, jehož vlastnosti lze nastavit právě zde. Jeho tvar – obdélníková část renderovaného snímku – jakožto nejmenší část posílaná uzlům v síti, je nastavitelná počtem pixelů v souřadnicích X(šířka), Y(výška). *Buckety* je možné poslat nečinným stanicím v LAN nebo rozdělit mezi CPU v jednom serveru. Jak jsem zmínil v kapitole [Rozdělení dat mezi uzly](#), nastavení velikosti *bucketu* je podstatné pro efektivní využití počítačů v síti. Příliš velké oblasti (málo bucketů) vedou k nevyužití výpočetní síly. Rozdělíme-li totiž snímek na více menších oblastí, může pracovat více procesorů na jednom snímku a urychlit tak celý proces renderování. Na druhou stranu rozdělíme-li snímek na příliš velký počet *bucketů*, mohou režírní náklady v podobě času spotřebovaného na nastavení či přenos *bucketů* po síti převýšit dobu renderování s nižším počtem bucketů (tj. menším počtem CPU zapojených do procesu). Pojem **RegionW/H (width/height)** pak určuje velikost oblasti dle šířky/výšky v pixelech, další možnost je *Region count*, kdy definujeme počet oblastí v snímku. V sekci **Distributed rendering** se setkáme se známými vlastnostmi jako přidání, odebrání serverů z renderování nebo nastavování priorit úloh (nízká / pod normálem / normální / nad normálem / vysoká / realtime).

Všechny renderovací systémy se v sobě snaží zahrnout 2 základní aspekty:

Rychlost a kvalitu. Tyto požadavky kladou na vývoj systémů obrovské nároky. Rutiny jsou často zpracovány v assembleru pro co možná nejrychlejší a nejefektivnější algoritmy.

Hledáme-li tedy v současných podmínkách to nejlepší řešení, znamená to zkombinovat velmi dobrý renderovací systém s velkým počtem počítačů v síti. A porovnávat lze opravdu

intenzivně. Rozdíly mezi systémy v době nutné pro zpracování některých raytrace snímků jsou rámcově i hodinové.

5 Závěr

Oblast animačních technologií jde velmi rychle kupředu. Konkurenční boj roste, což těší zákazníky v podobě novinek zabudovaných do 3D aplikací. Ceny HW také klesají velmi rychle. Důkazem toho mohou být stále častější animované snímky i celovečerní CGI filmy. Neoddělitelná součást těchto aktivit je vývoj v oblasti distribuce úloh v rámci renderovací farmy a neustálá snaha o snižování doby zpracování. Děje se tak jednak v oblasti vylepšování algoritmů pro renderovací systémy a jednak v oblasti hardwaru a sítí. Výkonné SGI servery jsou pomalu ale jistě doháněny běžně dostupnými dvouprocesorovými osobními počítači a je tak umožněno i menším studiím využít síly distribuovaného renderování. Tato oblast tedy jistě není uzavřená a domnívám se, že časem lze očekávat jen a jen lepší výsledky.

6 Příloha

Následující obrázky byly vytvořeny za použití *Brazil rendering system* a dokazují, že vytváření fotorealistických snímků za pomoci distribuovaného renderování již dnes není problém.



OBR.3. fotorealistický raytrace, autor: *Johan Thorngren*



OBR.4. autor: Al Barranco



OBR.4 autor: Jorge Seva & Sergio Miruri



OBR.4 autor: Al Barranco
více viz na www.splutterfish.com

7 Literatura

1. [<http://www.splutterfish.com> - Brazil rendering system]
2. [<http://www.chaosgroup.com> - Vray]
3. [Timothy A. Davis, Edward W. Davis - Rendering computer animations on a network]
4. [3ds max 5 – Users guide]
5. [<http://www.cgchannel.com> – Entropy rendering system]
6. [G. Humphreys, I. Buck, M. Eldridge, P. Hanrahan – Distributed Rendering for Scalable displays (část o Bucket rendering)]
7. [M.K. Novi – Distributed animation rendering system]
8. [<http://www.pixar.com> – Pixar's Renderman]

Pozn.: Brazil Rendering system, Vray, 3ds max, SGI, Silicon graphics, Pixar's Renderman, Mental Ray a všechny ostatní zmíněné obchodní značky (*pokud nebylo uvedeno jinak*) a jména produktů náleží jejich příslušným držitelům.
Obrázky lze nalézt na www.splutterfish.com a náleží jejich autorům.