Ausarbeitung zu

Netzwerkrendering & Distributed Rendering

Rendering in verteilten Systemen

Vorlesung Verteilte Systeme / WS07/08

Valentin Schwind (vs016)

Inhalt

1 Einleitung

1.1 Abstrakt

Die Vernetzung von Computern brachte schon seit ihren Anfängen Anwendungsentwickler auf die Idee, Infrastrukturen verteilter Systeme für die kollektive oder kollabierende Berechnung von Ergebnissen zu verwenden. Mit der Zeit etablierte sich das verteilte Rechnen und erreichte die verschiedensten Branchen. Grid-Computing wird z.B. von der Forschung zur gemeinschaftlichen, koordinierten Nutzung von Ressourcen innerhalb eines Netzwerks (oft sogar im Internet) verwendet. Verteiltes Rechnen wird bei physikalischen Simulationen (z.B. Wettervorhersagen, Bewegung von Galaxienclustern oder Verhalten von Flutwellen), für die Belegung oder Widerlegung mathematischer Theorien (z.B. die Suche nach Primzahlen) oder zum Design zukünftiger Medikamente verwendet. Die Wirtschaft nutzt verteiltes Rechnen zum Testen und Simulieren neuer Baustoffe (Aero-Grid), für die Analyse von Börsenkursen oder gar als Plattform für Softwaretechnologien, die sich generell auf die Lösung von massiven Berechnungen spezialisieren (BOINC, X-Grid).

Auf das Rendering von Computergrafiken in verteilten Systemen werde ich in dieser Ausarbeitung genauer eingehen. Sie behandelt die Ideen, die Prinzipien und die Techniken, die zur Verteilung beim Rendern notwendig sind, und zeigt anhand des Software-Bundle „Backburner“ und „3ds max“ der Firma Autodesk die Funktionsweise vom Verteilten Rendering.

1.2 Wie kommt man auf die Idee, verteilt zu rechnen?

Trotz der ungeheuren Rechenleistung heutiger Computer, reicht die Geschwindigkeit einzelner Systeme bei Weitem nicht aus, um schnell, massive Datenmengen zu berechnen. Dieses Problem wird nur in manchen Fällen durch die Einführung schnellerer Hardwarekomponenten (z.B. die CPU) tatsächlich gelöst. Der Grund hierfür liegt in der Software, den Entwicklern und an den Benutzern selbst, die durch schnellere Einzelsysteme dazu verleitet werden, die Qualität und Genauigkeit ihrer Berechnungen immer weiter zu erhöhen. Damit steigen natürlich wieder die Anforderungen an das neue System. In der Praxis bedeutet das, dass z.B. eine große Mehrheit von Mathematikern auch die Möglichkeiten der neuen, eingeführten 64-Bit Technologie nutzen möchte, um genauere Ergebnisse zu erhalten, oder ein 3D-Grafiker globale Beleuchtungseffekte bei der Berechnung der Szenen aktivieren möchte, um realistischere Bilder zu erzeugen. Immer mehr Anforderungen der Software sorgen mit der Zeit dafür, dass die „gefühlte“ Leistung und Geschwindigkeit der Hardware eines Einzelplatzsystems gleich bleibt (natürlich auch die effektive Arbeitszeit der Anwender). Nur zusätzliche Rechensysteme, die durch ein Netzwerk miteinander verbunden sind, und sich gemeinsam an der Berechnung einer Aufgabe beteiligen, beheben diesen Effekt nachhaltig, zeitsparend und verhältnismäßig kostengünstig.

1.3 Was sind die Vorteile von Berechnungen auf verteilten Systemen gegenüber Einzelcomputern?

Verteilte, parallele Berechnungen haben sich mit der Zeit als sehr viel effektiver und wirtschaftlicher erwiesen, als zentralisierte Computersysteme. Hier einige Gründe:

Ausfallsichereit
Der Ausfall eines einzelnen Systems führt zum Stopp aller laufenden Berechnungen. Je nach Art der Anwendung (und Stand der letzten Speicherung) führt dies zu einem Neustart und Wiederholung der Jobs von einem bestimmten Punkt an. Dies kostet Zeit und Geld. Ein verteiltes System bietet dagegen zu einem gewissen Grad Ausfallsicherheit, denn fällt eines der Systeme aus, rendern alle anderen unverändert weiter. Unerledigte Aufgaben des ausgefallenen Systems werden automatisch an andere delegiert.

Skalierbarkeit
Im laufenden Betrieb können Systeme an das Netzwerk angebunden werden. Bei jedem System altert die Hardware und muss irgendwann heruntergefahren oder ausgetauscht werden (ungeachtet der Tatsache, dass ein Austausch von Hardware in kritischen Projektabschnitten ungeahnte Folgen haben kann!). Grund hierfür kann auch der Ausfall einer Hardwarekomponente sein, oder leistungshungrigere Software. Oft ist die Aufrüstung teurer als der Kauf eines komplett neuen Systems. Die veraltete Hardware wird in einem verteilten System nicht durch die neue ersetzt, sondern kann das verteilte System weiterhin unterstützen.

Konsistenz
Ist ein isoliertes System fehlerhaft (oder es werden fehlerhafte Berechnungen auf einem durchgeführt) und wird dies vom Benutzer oder Entwickler eines Einzelplatzsystems nicht erkannt, kann der Beteiligte ohne Vergleich (oft) nicht erkennen, dass es sich bei seinen Ergebnissen um inkorrekte Daten handelt. Erst in einem verteilten System gibt es ein Vergleichsmodell oder eine „Mehrheit“, die dem Anwender die Fehlerhaftigkeit des Einzelsystems mitteilt.

Preis
Einzelcomputer im Verbund sind sehr viel günstiger, als ein vergleichbar schnelles Hochleistungszentrum (Mainframe). Zudem sind alle Komponenten problemlos erhältlich und können auch anderweitig oder nach Projektabschluss weiter dienlich sein (z.B. als Workstation oder Webserver).

1.4 Fazit

Für wirklich redundantes und schnelles Rechnen benötigt man ein verteiltes System. Ein solcher Systemverbund wird meistens als Computercluster, bei Berechnungen durch Grafikanwendungen auch als Renderfarm bezeichnet. Die Wirtschaftlichkeit hat sich für die Industrie bezahlt gemacht. Firmen bieten z.B. gegen eine Gebühr die Rechenleistung solcher Renderfarmen als Dienstleistung an. Schwerpunkt dieser Ausarbeitung ist das Rendern von Computergrafiken in verteilten Systemen. Was es mit dem Rendern auf sich hat, will ich im nächsten Kapitel erläutern.

2 Über das Rendern

2.1 Was ist Rendern?

Schon vor dem Informationszeitalter haben Menschen versucht aus Rohdaten, genaue (vor allem konsistente) Ergebnisse zu berechnen, die schließlich auf ein bestimmtes Medium gebracht werden können. Früher war es nur mit Taschenrechner, Stiften und Papier oder gar nur mit Kreide an der Tafel möglich technische Zeichnungen anzufertigen. Gebäude mussten z.B. vom Architekten stets als Modell vorkonstruiert werden. Filme konnten nur mit komplizierten Belichtungsverfahren nachbearbeitet werden. Heute geschieht alles mithilfe von Computern.

Vor dem Rendern existieren Rohdaten. Eine Anwendung errechnet durch sie Mediendaten, die nicht immer auch das Zielmedium sein müssen, sondern auch als Zwischenstufe dienen können. Es gibt unterschiedliche Arten von Mediendaten, die gerendert werden können. Die Verfahren der Berechnung und die Zielformate hängen von der Anwendung ab. Das Ergebnis einer solchen Berechnung bezeichnet man als Rendering. Die Zeit, die eine Anwendung zur Berechnung eines Renderings benötigt, nennt man Renderzeit, die von der Geschwindigkeit (CPU, GPU, RAM) oder Anzahl der beteiligten Systeme abhängig sind.

Wichtig ist, an dieser Stelle zu erwähnen, dass es sich beim Rendern um keine Konvertierung/Umwandlung von Daten handelt. Daten werden zwar in einem bestimmten Format codiert, waren aber davor als solches nicht vorhanden. Rendern bedeutet, dass neue Datenstrukturen/Mediendaten erzeugt werden, die davor nur in Form der Rohdaten vorhanden waren.

2.2 Ausgabeformate

Bilder
Ziel der Berechnung bei der Bilddatengewinnung sind zweidimensionale (komprimierte) Bitmaps (siehe PNG, JPG, BMP) oder Vektordateien (siehe PDF-Generierung). Man kann aus 3D-Daten 2D-Daten gewinnen (z.B. 3ds max, Maya, Softimage), aus 2D-Daten Bitmaps rendern (z.B. Photoshop, Illustrator, Photopaint) oder aus Videomaterial heraus Einzelbilder erzeugen (Premiere Pro, Avid XPress). In Programmen wie Illustrator und CorelTrace gibt es die Möglichkeit aus Bitmaps wiederrum Vektor-Daten zu rendern (Vektor-Rasterisierung). Bei jedem Rendering-Prozess gibt es unterschiedliche Renderzeiten (von Millisekunden bis mehrere Tage) und ist von der Größe und Bittiefe der Bitmaps abhängig.

Videos
Videos zu rendern kennt jeder, der einmal ein Urlaubsvideo geschnitten hat. Hier werden die Rohdaten der Kassette aus einer Videokamera übertragen (häufig noch nachbearbeitet) und schließlich als ein neues Video in ein Zielformat gerendert. Die Renderzeit bei z.B. einem Spielfilm ist von der Codierung, Länge und Auflösung abhängig und beträgt bei guter Auflösung ca. 12 Stunden.

Audio
Bei Audiodaten werden unterschiedliche Spuren zusammengemischt („Dubbing“ von WAVE-Dateien), künstliche MIDI-Instrumente hinzugefügt (VST-PlugIns) und Effekte (Kompressor, Hall, Reverb) beigegeben, die schließlich in eine neue Audiodatei gerendert werden. Bei vielen Audioanwendungen geschieht dies schneller als in Echtzeit.

Interaktive Anwendungen
Authoring-Tools (z.B. für DVD-, BlueRay-Medien) oder Frameworks für interaktive Webseiten (Flash, Flex, Silverlight) erzeugen interaktive Menüs oder Clips, auf deren Ablauf der Benutzer Einfluss nehmen kann. Solche Daten werden ebenfalls gerendert (oder teilweise gleichzeitig mit in Maschinen- oder Bytecode compiliert).

Binär-Daten
Simulations-Ergebnisse, die meist auch nur von speziellen Anwendungen (z.B. RealFlow zur Flüssigkeitssimulation) erzeugt und wiedergegeben werden können, werden oft in Binärdateien abgespeichert.

2.3 Computergrafiken

Das Rendering von Bilddaten durch eine 3D-Anwendung wird in der Computergrafik in zwei Teilbereiche unterschieden:

Echtzeitrenderings
Durch die Grafikkarte werden Bilddaten mit meist mit sehr hoher Geschwindigkeit (der sogenannten Framerate) gerendert und über den Monitor dargestellt. Solche Daten werden in der Regel nicht gespeichert, sondern dienen allein dem Gefühl, dass der Benutzer sich in Echtzeit z.B. durch eine 3D-Welt bewegt. Echtzeitrenderings werden hauptsächlich in Spielen, Demos, Benchmarks oder Machinimas (Videos mit Echtzeitgrafiken) eingesetzt.

Framerenderings
Nicht die Grafikkarte, sondern die CPU, berechnet Bilddaten und legt sie in einzelne Dateien (Frames) ab. Spielt man die Einzelbilder mit mehr als 24 Bilder pro Sekunde ab, entsteht wieder der Eindruck eines flüssigen Bewegungsablaufs. Gerenderte Frames, die nacheinander abgespielt werden können sind Animationen.

Das Framerendering von Animationen war der ausschlaggebende Faktor für die Nutzung von verteilten Systemen zur Berechnung von Grafiken. Denn alle Frames einer Animation sind trotz komplexester Abläufe durch die Rohdaten der 3D-Szenen unabhängige, parallel-berechenbare Einheiten, die getrennt gespeichert und durchnummeriert werden können. 3D-Programme müssen aber garantieren, dass es sich bei jedem Framerendering um eine konsistente Berechnung handelt. D.h. das Einzelbildrenderings sind eine Möglichkeit Animationen über Netzwerke zu rendern. Dabei muss jedoch sichergestellt werden, dass Rohdaten stets auch gleich gerendert werden! Geschieht des nicht, wird dies in der Animation bemerkbar, weil Frames unterschiedlich aussehen. Dies führt

zu Flackern oder zu Artefakten. Vollständige (encodierte) Animationen, Videos oder Echtzeitrenderings können auf heutigen Systemen noch nicht (oder nur mit ungeheurem Aufwand) verteilt gerendert werden! Die andere Methode des verteilten Renderings ist das „Distributed Rendering", über das wir später mehr erfahren werden. Doch bevor wir uns der Funktionsweise des Netzwerkrenderings widmen, möchte ich aufzeigen, warum Rendern so ungeheuer aufwendig ist. Zudem möchte ich die wichtigsten Rendertechniken vorstellen, die sich mit der Zeit im Grafikbereich etabliert haben.

2.4 Wieso ist Rendern so aufwendig?

Damit aus den Rohdaten einer 3D-Szene Bitmaps werden, muss der Renderer einer Anwendung ungeheure Datenmenge verarbeiten. Hierbei werden Matrizen umgewandelt, Integrale aufgelöst, Differenzierung gemacht und jede Menge Dateien in den Arbeitsspeicher kopiert. Für einen Menschen wären diese Berechnungen unvorstellbar aufwendig (aber theoretisch lösbar!). Grafiker wollen für ihre Renderings natürlich allerhand Effekte und Features nutzen, damit ein immer realistischerer Eindruck ihrer 3D-Modelle entsteht. Szenen werden hier mit unterschiedlichen Methoden und Verfahren berechnet, die immer mehr CPU-Leistung erforderlich machen. Ich habe eine Liste der wichtigsten Verfahren gemacht, und zeige grob auf, was bei einem Renderdurchgang alles berechnet werden kann:

Geometrie
Volumen, Vertex- oder Dreieckssberechnungen sind der einzig notwendige Bestandteil eines Renderers (sonst wird nichts gerendert).

Texturen
Bitmaps werden als Texturen auf die Oberfläche von Modellen projiziert. Aufgrund hoher Auflösungen müssen Texturen ebenfalls hohe Auflösungen haben. Große Texturen verbrauchen sehr viel Arbeitsspeicher.

Kantenglättung/Supersampling
Wichtige Grundlagen für Renderings sind die Kantenglättung für Geometrie und das Supersampling (ebenfalls ein Glättungsverfahren) für Texturen.

Schatten
Lichter werfen Schatten, die über unterschiedliche Verfahren (Raytrace, Shadow-Maps) erzeugt werden. Dieser Teil der Berechnung kann unterschiedlich präzise sein. Generell gilt: je genauer, desto aufwendiger, desto besser die Animation.

Globale Belechtung/Radiosity
Da es in der Wirklichkeit keine „Lichter“ gibt, sondern nur Objekte, die wegen ihrer Temperatur leuchten, wurden die GI (Global Illumination)-Renderer entwickelt, die nicht nur Lichter, sondern auch Objekte selbst leuchten und Schatten werfen lassen können. Hierbei wird sehr viel mehr Renderzeit benötigt, als bei den klassischen Beleuchtungsverfahren, da ein Lichtstrahl bei reflektierenden Objekten abprallen und seine Farbe ändern kann.

Reflexion/Refraktion
Diese Features sind sehr rechenintensiv, da sie die Brechung des Lichts, Beschaffenheit der Oberflächen und das Medium berücksichtigen, durch das ein Lichtstrahl geht.

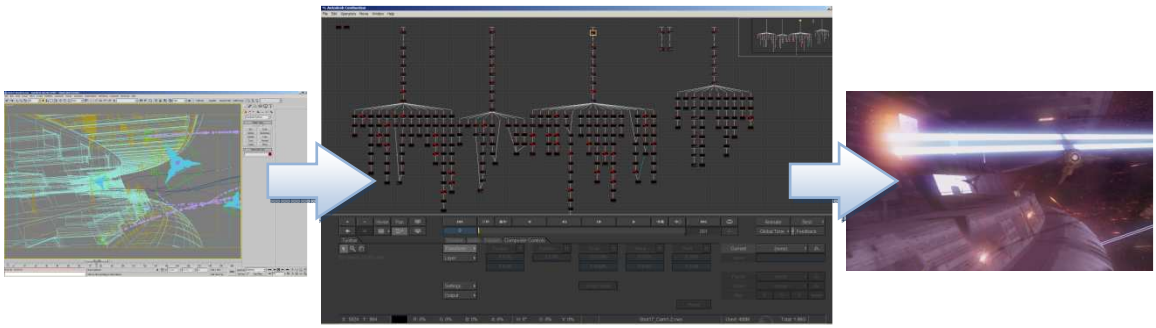
Caustics
Tritt ein Lichtstrahl aus einem Medium aus, erzeugen diese eine neue Lichtquelle, die wieder Strahlenbündel wirft (z.B. ein Licht, dass durch ein Weinglas geht und es gebündelt auf einen Tisch wirft).

Transluzenz/Sub-Surface-Scattering
Hier wird Licht innerhalb eines Objekts absorbiert, wodurch es heller erscheint (z.B. eine Taschenlampe hinter einer Hand).

Nimmt man alle Berechnungen zusammen, kombiniert diese mit sehr vielen Iterationsschritten und „Passes“, die nötig sind um eine flackerfreie Animation zu berechnen, erhält man sehr hohe Renderzeiten, die in ihrer Summe nur noch von mehreren Rechnern bewältigt werden können.

2.5 Rendern in Layern

Bei modernen Render-Verfahren werden (unter anderem) Effekte in sogenannten Layern gerechnet. Das bedeutet, dass jeder beliebige der obengenannten Features oder auch beliebig viele Strukturen innerhalb einer Szene (z.B. Hintergründe, ober einzelne Objekte) als separate Einzelbildreihe gerendert werden, sodass die Grafiker nachträglich im Compositing die Kontrolle über die einzelnen Objekte & Effekte behalten können, die sie beim Rendering einer einzelnen Animation verlieren würden. Diese Technik lässt erahnen welch ungeheurer Aufwand und Unmengen von Daten bei der Berechnung von komplexen Computeranimationen entstehen können. In Compositings werden die separat gerenderten Layers (Einzelbildreihen) bearbeitet und zu einem Rendering zusammengefügt.



3D-Szene Compositing der Layer Final

Compositing & Framerenderings werden derzeit noch über die CPUs eines Systems berechnet, da sie (im Gegensatz zu Grafikkarten) auch bei unterschiedlichen Herstellern und Familien konsistente Ergebnisse liefern. Eine Ausnahme bilden hier jedoch die 32- und 64-Bit Plattformen, die voraussetzen, dass man sich für eine der beiden Architekturen entscheiden muss. Da Prozessoren bei ihren Berechnungen vollständig ausgelastet werden (es sei denn man weißt den einzelnen Prozessen weni-

ger CPUs oder Prioritäten zu), wird ein System während des Renderns aufgrund seiner Auslastung für weitere Anwendungen faktisch nicht bedienbar.

2.6 Kanäle

In jedem Rendering gibt es zusätzlich noch die Option Kanäle zu verwenden. Für normale Bitmaps sind dies die jeweiligen Kanäle für Rot-, Grün- und Blau-Werte. Damit Renderings übereinandergelegt werden können, erlauben viele Datei-Formate (PNG oder TGA) einen Alpha-Kanal, der die Transparenz jedes Pixels festlegt. Komplexe Formate, wie RPF oder RLA, ermöglichen es noch sehr viel mehr Kanäle zu verwenden. Da aber all diese Kanäle beim Rendern sowieso berechnet werden kann der Anwender entscheiden, ob oder in welchem Format er die Kanäle mit gespeichert haben will. Da die Kanäle keinen Einfluss auf die Renderzeit haben und automatisch mit in jede Datei gespeichert werden, will ich in dieser Ausarbeitung nicht weiter auf Kanäle eingehen. Es bleibt noch zu erwähnen, dass jeder Kanal wiederrum in eine Datei/Einzelbildreihe gerendert werden kann.

2.7 Aktuelle Renderer

Derzeit gibt es eine ganze Reihe von Renderer, die sich durch die Palette unterstützter 3D-Anwendungen, durch ihre Verfahren und über ihre Features voneinander unterscheiden. Externe Renderer werden als PlugIn in eine 3D-Anwendung integriert. Interne Renderer sind standardmäßig implementierte Lösungen. Das Prinzip der Renderer bleibt immer gleich: 3D-Daten (Dreiecke & deren Oberflächen) müssen auf ein Pixelraster projiziert werden. Verbreitete Renderer in 3D-Programmen sind:

- vray (hochwertiger Renderer, mit der besten Balance zwischen Performance & Qualität)
- mental ray (verbreitetester Renderer, nahezu in allen 3D-Applikationen vertreten)
- maxwell (physikalisch korrekter Renderer mit irrsinnig hohen Renderzeiten)
- final render (flexibler Renderer mit komplexen Einstellungsmöglichkeiten)
- brazil (hochwertiger Renderer, Entwicklung wurde leider eingestellt)
- renderman (entwickelt von Pixar, wird nur in der Filmindustrie verwendet)
- sowie die Standardrenderer der jeweiligen 3D-Software (Scanline Renderer)

2.8 3D-Applikationen

Natürlich benötigt man zum verteilten Rendern auch eine entsprechende Grafikanwendung. Bis auf Pixar's renderman, benötigen alle derzeitigen Renderer eine Software, die als Plattform für das Rendering dient. Diese Anwendung muss daher auch auf jedem der beteiligten Systeme installiert sein. Da ich persönlich gerne mit 3ds max von Autodesk arbeite, möchte ich anhand dieser Software erklären, wie verteiltes Rendering genau funktioniert. Wichtig ist zu bemerken, dass dieses Verfahren in anderen 3D-Applikationen ähnlich (oder genauso) gehandhabt wird.

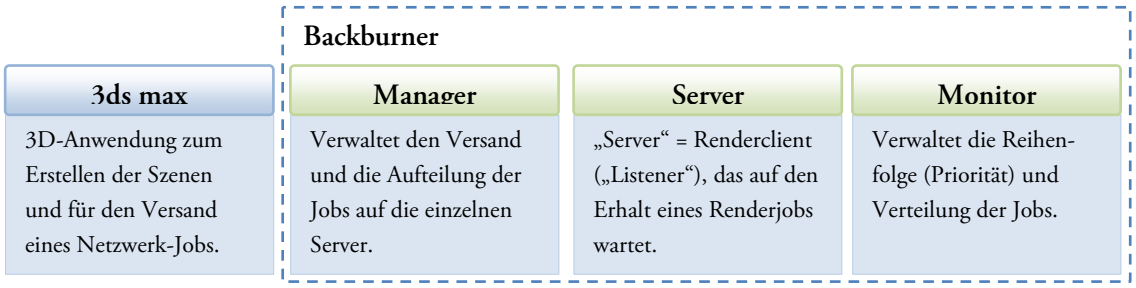
3 Netzwerkrendering

3.1 3ds max und Backburner

Wie erwähnt, nutze ich 3ds max von Autodesk als 3D-Anwendung. Dieses Tool ist in der 3D-Branche sehr weit verbreitet, unterstützt zahlreiche Standards und Dateiformate, bietet Einsteigern und Fortgeschrittenen genügend Funktionen und besitzt eine breite Auswahl an Features, um ausgereifte Computeranimationen zu produzieren. Eines der wichtigsten Features von 3ds max ist dabei die Fähigkeit nicht nur lokal, sondern auch im Netzwerk zu rendern.

Zum Netzwerkrendern gibt es bei 3ds max ein Tool-Paket, dass einen Manager für die Rendertasks, einen Monitor zur Verwaltung bietet und einen „Listener"-Tool mitbringt, das 3ds max startet und rendern lässt, wenn ein Job erhalten wurde. Diese Programm-Reihe heißt Backburner und wird bei einer 3ds max-Installation standardmäßig mit installiert, sodass alle im Netzwerk befindlichen Rechner alle Programme besitzen, die später für das Rendern im Netzwerk notwendig sind.

Die Tools im Überblick:



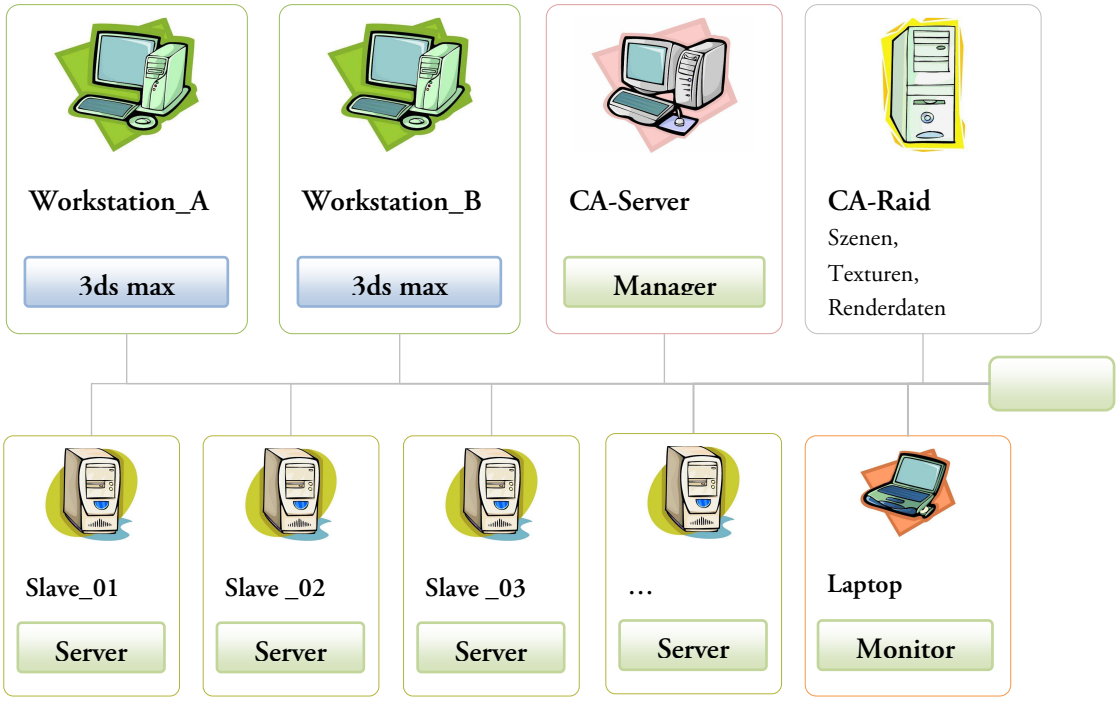
3ds max wird beim Netzwerkrendern zwar nicht als normale 3D-Anwendung offen gehalten, aber zwangsläufig müssen alle verwendeten PlugIns und Renderer, die in einer Szene verwendet werden, auch auf dem Clienten vorhanden sein, da sonst Dateien nicht geöffnet und gerendert werden können.

Darüberhinaus sollten sämtliche Programme und PlugIns dieselben Versionen und Services Packs besitzen. Es ist auch von Vorteil alle Systeme mit dem gleichen Betriebssystem auszustatten. Bei der Wahl einer 32- oder 64-Bit Architektur ist es zwingend erforderlich die beteiligten Systeme des Netzwerkrenderings homogen zu halten. Der Grund hierfür liegt in den Anwendungen, die bei 32- und 64-Bit-Architekturen unterschiedliche Ergebnisse berechnen (dazu später noch ein Beispiel).

Wollen wir uns nun überlegen, wie wir die Produkt-Familie von Autodesk geschickt auf ein verteiltes System bringen.

3.2 Ein Beispiel-Netzwerk

Um auf verteilten Systemen zu rendern, müssen wir uns zunächst eine Methode überlegen, wie wir am geschicktesten „verteilen“ und welche Rechner für welchen Job geeignet sind. Dies bedeutet dabei auch, dass es bei der Verteilung nicht nur darauf ankommt, dass alle beteiligten Systeme gleichmäßig ausgelastet werden, sondern auch, dass das Netzwerk unter der Flut von Datenströmen nicht zusammenbricht.



Ein typisches Firmennetzwerk besteht meist aus mehreren Workstations (A,B), einem Server (CA-Server), aus einem Raid-System für die Projektdaten (CA-Raid) und aus einer Renderfarm (Slave). Diese sind meist mit einem Switch verbunden. Zusätzlich kann noch ein mobiles Endgerät ans Netzwerk geschlossen werden, dass die Renderjobs überwacht.

Die obige Darstellung ist nur ein Beispiel von Entities. Wichtig ist zu erkennen, dass es (zunächst) egal ist, auf welchem System gerendert wird, wenn die beschriebenen Backburner-Prozesse laufen. Theoretisch ist es somit möglich alle Backburner-Prozesse auf nur einem System zu starten und somit die obige Darstellung des Netzwerks auf nur einen Computer zu reduzieren. Workstations können (solange nicht gearbeitet wird) ebenfalls mit rendern. Es empfiehlt sich CA-Server und CA-Raid keine Renderjobs zuzuweisen, da diese dadurch sehr langsam reagieren würden. Würden die Slaves in großem Mengen Daten anfordern käme es zu sehr langen Reaktionszeiten im gesamten Netzwerk.

Ich habe in meinem Beispiel CA-Server und CA-Raid getrennt. In der Praxis bilden diese beiden Systeme oft ebenfalls eine Einheit. Ich möchte jedoch verdeutlichen, dass die Projekt-Daten nicht unbedingt auf dem CA-Server liegen müssen. Zu erwähnen ist auch, dass es sich beim Raid um ein sehr performantes System mit zusätzlichen Backup-Funktionen handeln sollte (RAID5). Oftmals

wird (z.B. in der HdM-Stuttgart) das Raid-System in zwei Systeme unterteilt. Während Projektdateien (Szenen, Texturen) sehr schnell und redundant gehalten werden müssen, können Renderdaten reproduziert werden, benötigen aber weitaus mehr Speicher und werden daher oft in einem anderen System ausgelagert. In unserem Beispiel bleiben die Renderdaten jedoch auf dem Raid.

Die Topologie des Netzwerks selbst ist beim verteilten Rendern uninteressant. Zu beachten gilt, dass nach Erhalt eines Renderjobs enorme Datenströme von CA-Raid wegfließen. Es gibt daher mehrere Dinge, die ich beim verteilten Rendern empfehlen würde:

- Hohe Datenraten bevorzugt (am besten 1 Gbit-, 10 Gbit-Ethernet oder FibreChannel)
- Switch als LAN-Knoten
- Kein WLAN verwenden!
- Webserver für Job-Monitoring im Internet (beim Backburner mitgeliefert)

Anmerkung: Früher fand man oft verteilte Systeme, in denen zusätzlich Maps & Texturen auf andere Systeme ausgelagert wurden. Da es sich in der Computergrafik durchgesetzt hat, Texturen nur für bestimmte Szenen zu entwickeln (z.B. für die Schatten bestimmter Lichtsituationen auf Texturen der jeweiligen Objekte), wird im Allgemeinen kein zusätzliches System mehr benötigt, da solche Maps traditionell im Szenenordner gespeichert werden.

3.3 Konfiguration

Alle beteiligten Rechner sollten nach Möglichkeit eine feste IP-Adresse und einen eindeutigen Hostnamen besitzen. Es empfiehlt sich den CA-Server ins dasselbe Subnetz der Workstations und Slaves zu legen, ansonsten findet die automatische Subnetz-Suche von 3ds max beim Abschicken keinen Manager. Befindet sich der CA-Server in einem anderen Subnetz, muss er direkt adressiert werden.

3ds max vergibt standardmäßig ein TimeOut von 600 Minuten für jedes Frame eines Renderjobs. Backburner bricht einen Job ab, wenn dieses Limit erreicht wurde. Dieses sollte also manuell erhöht werden, wenn es sich um ein sehr lang andauerndes Rendering handelt.

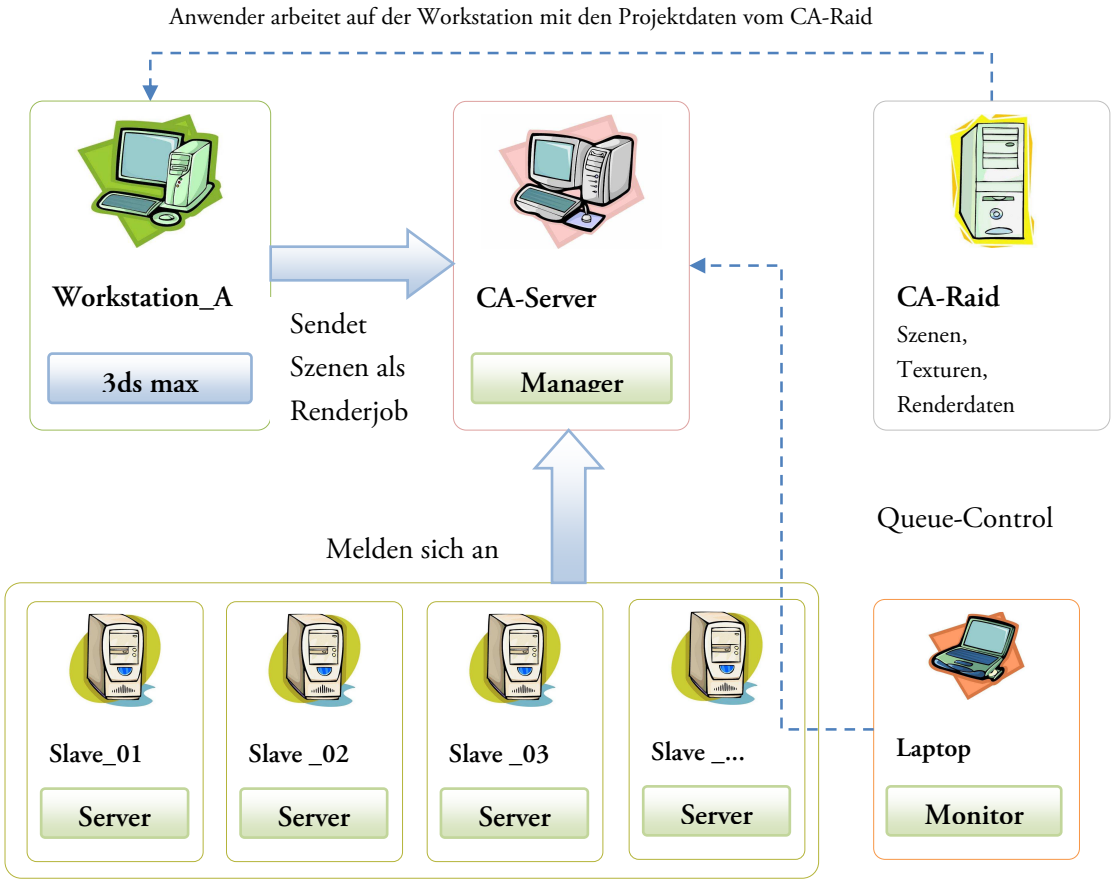
3.4 Abschicken eines Renderjobs

Was passiert nun, wenn wir eine Szene fertig gemodelt und animiert haben und diese nun im Netzwerk rendern möchten?

Das Abschicken von einem (oder mehreren) Jobs geschieht aus 3ds max heraus. Das Programm selber hat nur die Fähigkeit einige Optionen anzupassen, danach hat es keinen Zugriff mehr auf den Job. Beim Abschicken wird jede Szene (MAX-Datei) im Backburner-Ordner des CA-Servers abgelegt. Die Datei wird als ZIP komprimiert und zusätzlich mit XML-Metadaten versehen, die der User beim Abschicken angegeben hat (Name, Max-Version, Priorität, alternative Texturpfade, TimeOuts, etc.). Diese Einheit bezeichnen wir als „Job“ und wird vom Backburner-Manager als solches erkannt und registriert.

3.5 Verteilung

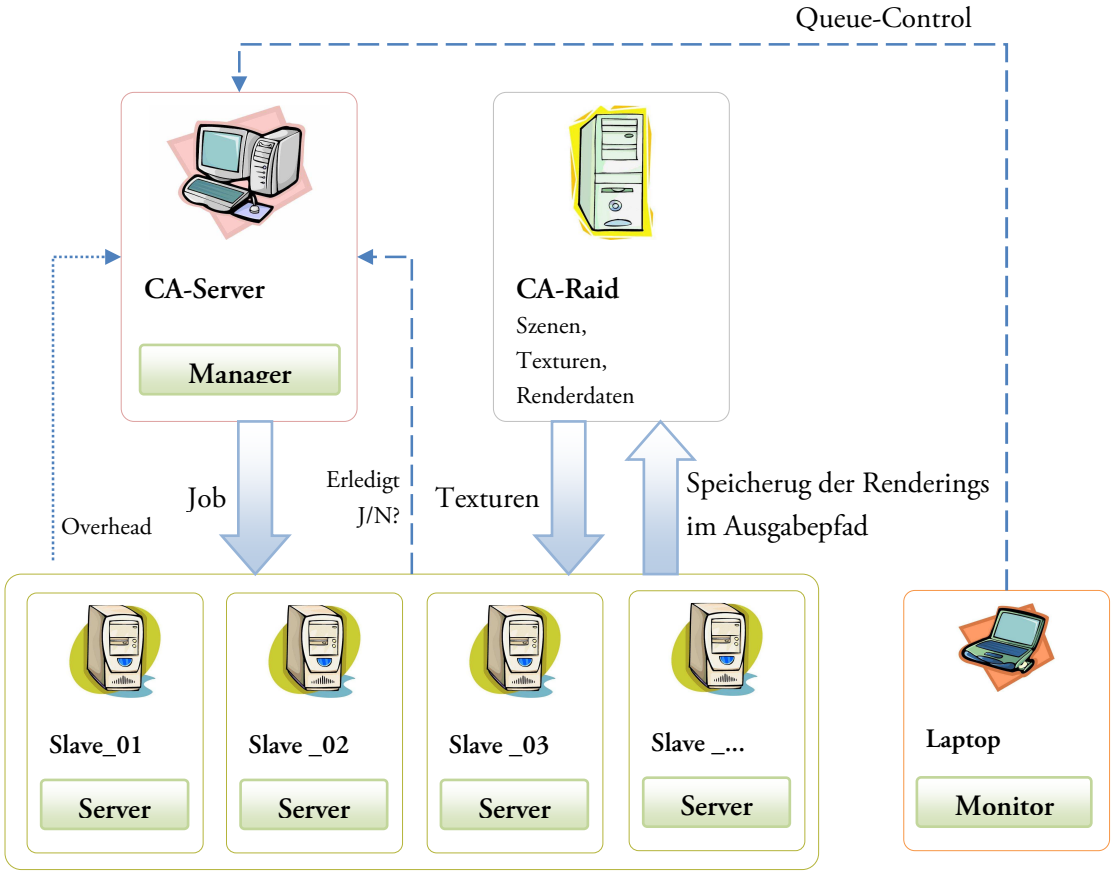
Erhält der Backburner Manager einen (oder mehrere Batch-Renderjobs), listet er diese automatisch nach Priorität und Datum. Wurde ein Job nicht als „Suspended“ markiert, wird er automatisch gestartet. Bevor ein Job an die Slaves geschickt wird, müssen sich diese beim Manager angemeldet haben. Sobald dies geschehen ist, wird der Job zugewiesen.



Der Rendermanager aktualisiert die Log-Datei und überprüft die Job-Metadaten. Ein beliebiges System kann sich die Job-Informationen mithilfe des Backburner Monitors ansehen. Hierbei sucht der Monitor nach einem Manager im Subnetz und registriert sich als „Queue-Controller“. Der Queue-Controller ist in damit in der Lage:

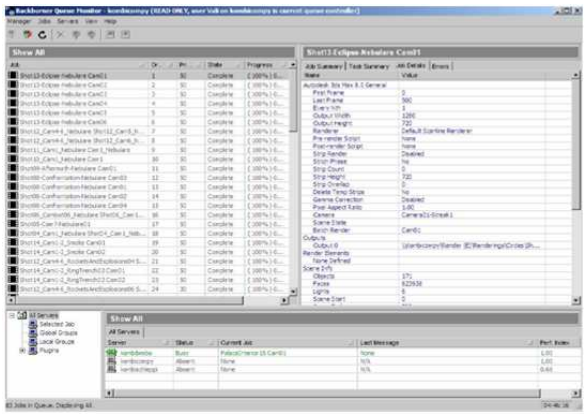
- Jobs anzuhalten (Suspend)
- Jobs neu zu starten (Restart)
- Jobs zu löschen (Delete)
- Die Priorität zu ändern (Priority: 0-99, Standard: 50)

Ist die Anwendung kein Queue-Controller, kann sie nur in die Jobübersicht einsehen. Beim Backburner gibt es keine Rechteverwaltung, kein User-LogIn und damit auch keine Administrator-Verwaltung. Damit kann jedes System die Queue-Control für die Renderjobs verwalten.



Nach Erhalt der Jobs werden alle ZIPs auf die einzelnen Server geschickt. Hier erkennt man den Flaschenhals eines verteilten Systems, da dabei enorme Datenmengen über das Netzwerk versendet werden. Es empfiehlt sich deswegen CA-Server und CA-Raid als getrennte Systeme zu behandeln, so dass 3D-Szenen und Texturen von getrennten Systemen bezogen werden. Nach Erhalt der komprimierten Szenen, entpacken diese die Archive und starten mit dem Öffnen der MAX-Datei. Beim „Lauching“ wird automatisch die eigentliche Applikation (3ds max) gestartet.

Im Backburner Monitor sehen wir welche Jobs mit welchem Fortschritt rendern. Es gibt eine Zusammenfassung für die gerenderten Frames, eine Übersichtstabelle der Systeme und deren Gruppe. Ohne manuelle Änderungen rendern alle beteiligten Systeme automatisch am selben Job bis dieser fertig ist.



3.6 Tasks & Blocks

Prinzipiell bedeutet Netzwerkrendern, dass ein Job verschickt wird, die Frames verteilt gerendert und die Renderings wieder zentral abgelegt werden. Das bedeutet auch, dass die Slaves gar nicht wissen (müssen), dass es sich bei den Einzelbildern eigentlich um Animationen handelt, die später zusammengesetzt werden. Auf den Slaves werden die Jobs lokal abgearbeitet und auf dem CA-Raid abgespeichert. Ein einzelnes Frame wird in der Regel auch als „Task" eines Jobs bezeichnet.

Werden aber auf einem schnellen System sehr viele Frames gerendert, kann der CA-Server, um Overhead bei der Kommunikation mit den Slaves zu sparen, und um den CA-Raid beim Speichern der Renderings zu entlasten, Tasks in sogenannte „Blocks" zusammenfassen. Erkennt der Manager, dass ein Slave z.B. sehr viele Frames hintereinander rendert, betraut er ihn mit einer bestimmten Anzahl von Frames (einem Block), die er dann als Gesamtpaket rendert und nach Abschluss eines Tasks auf dem CA-Raid ablegt und erst dann auch als „completed" markiert.

3.7 Notifikationen

Es gibt beim Netzwerkrendern natürlich eine ganze Reihe von Nachrichten, die gesendet werden können. Wir wollen uns aber gar nicht mit den Unmengen von Fehlermeldungen beschäftigen, die im Backburner so erscheinen können, sondern nur erwähnen, dass Notifikationen über Sockets an den Manager versendet werden. Denn während die übliche Kommunikation über XML-Metadaten läuft, werden die Registrierung und die Notifikationen (Errors, Events, Debugging) geloggt und direkt an den Manager gesendet.

3.8 Zusammenfassung

Beim Netzwerkrendern wird stets die lokale Anwendung ausgeführt, die nur auf das Rendern von Frames eingestellt sind. Zusätzlich zu 3ds max unterstützt Backburner auch Combustion für Compositing-Renderings.

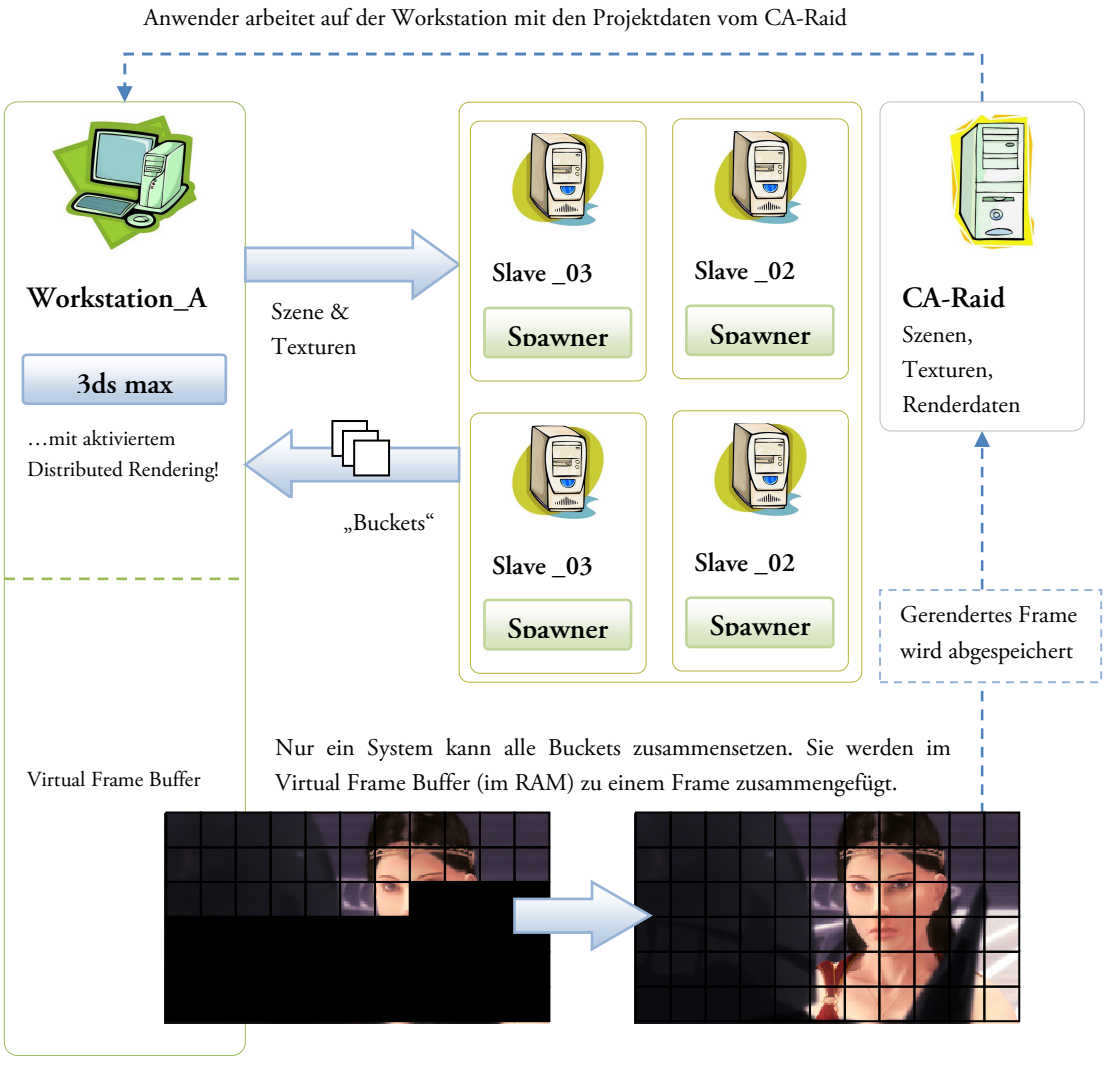
Netzwerkrendering bedeutet, dass es sich im eigentlichen Sinne keine echte Verteilung handelt. Dies ist der große Nachteil beim Netzwerkrendering, denn wenn ein System sehr langsam ist und mit einem Task nicht fertig wird, hat der Anwender keine vollständige Einzelbildreihe und ist davon abhängig, dass dieses langsame System fertig wird, wobei ihn die anderen (theoretisch) unterstützen oder ablösen könnten. Jedes System kann auch immer nur ein Bild berechnen, wodurch eine Renderfarm nutzlos wird, wenn der Grafiker nur ein einzelnes Motiv in Plakatgröße rendern möchte und darauf lange warten muss, weil es nur auf einem System rechnet.

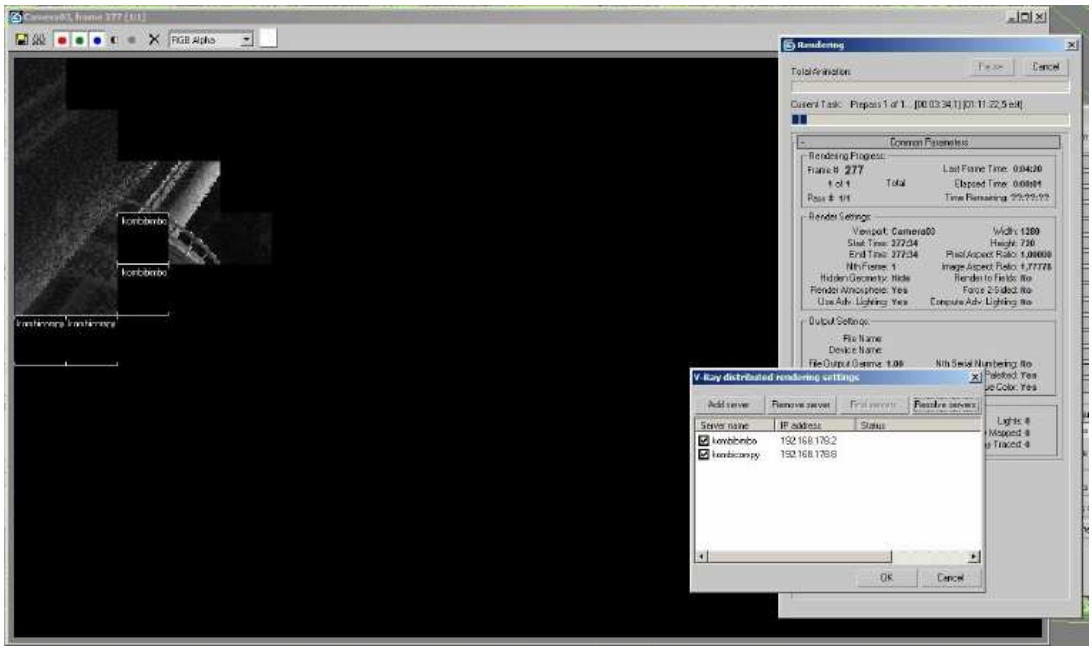
Die Lösung für die Verteilung von Frames nennt sich „Distributed Rendering" und wird im nächsten Kapitel erörtert.

4 Distributed Rendering

4.1 Wie funktioniert Distributed Rendering?

Wirklich verteiltes Rendern bedeutet, dass sich mehrere Systeme an der Berechnung eines einzelnen Frames beteiligen können. Dadurch wird alle verfügbare Rechenleistung im Netz gleichmäßig und voll ausgelastet. Wie beim Netzwerkrendering müssen beim Distributed Rendering einige Voraussetzungen gegeben sein. Distributed Rendering muss vom Renderer unterstützt werden (derzeit für 3ds max: vray, final render und mental ray) und es muss ein sogenannter „Spawner" auf den Slaves aktiviert sein, der die Szene verarbeiten und die errechneten Ergebnisse an die Workstation zurückschicken kann. Im Unterschied zum Netwerkrendering kommunizieren die Slaves weder mit dem Raid, noch mit einem CA-Server, sondern nur mit dem System, auf der der Job zusammengesetzt wird.





In der Praxis werden im Distributed Rendering-Menü des Renderers alle aktiven Systeme und deren Adressen eingetragen. Startet man ein Rendering erscheint der Virtual Frame Buffer und darauf das Ergebnis der einzelnen Buckets aller beteiligten Systeme. Ein Bucket steht für die CPU eines Systems. Fällt eines während des Renderings aus, wird der restliche Teil automatisch von anderen Systemen übernommen.

4.2 Tücken

- Bei vielen beteiligten Systemen entsteht ein großer Overhead im Netzwerk. Hier bietet es sich an wenig starke, anstatt viele schwache Systeme am Rendering zu beteiligen.
- Pre-Processing-Images (d.h. Bilder bei denen Informationen im Voraus berechnet werden müssen, z.B. Lichtinformationen) sollten auf nur einer Maschine berechnet werden, da die Verteilung voraussetzt, dass jeder Slave die Ergebnisse des anderen kennt. Dieses Problem ist in heutigen Renderern noch nicht ausreichend gelöst.
- Die Fertigstellung des Frames obliegt der Verantwortung der Workstation, die alles zusammensetzt. Fällt sie aus, ist das Frame verloren.
- Beim Distributed Rendering dürfen sich auf keinen Fall 32- und 64-Bit-Systeme gleichzeitig an einem Frame oder an einer Animation beteiligen. Die Hintergrundarchitektur muss homogen gehalten werden, damit keine Fehler im zusammengefügten Bild entstehen.
- Viele PlugIns unterstützen kein Distributed Rendering und können auch nicht verwendet werden, selbst wenn alle beteiligten Systeme das PlugIn installiert haben.

5 Schluss & Quellen

Grundlage dieser Ausarbeitung sind meine Arbeiten mit 3ds max, Combustion und Backburner. Ich habe keine Online-Quellen oder Bücher für meine Texte verwendet. Für die Grafiken standen freie Cliparts von www.barrysclipart.com zur Verfügung. Sonstige Bilder sind von mir und können nach Belieben weiterverwendet werden.

Ich hoffe hiermit einen Überblick über die Funktionsweise vom Rendering in verteilten Systemen verschafft zu haben und bedanke mich für Ihre Aufmerksamkeit.

Valentin Schwind

Online: www.valisoft.com
Nightfall-Grafiken: nightfall.valisoft.com
E-Mail: mail@valisoft.de