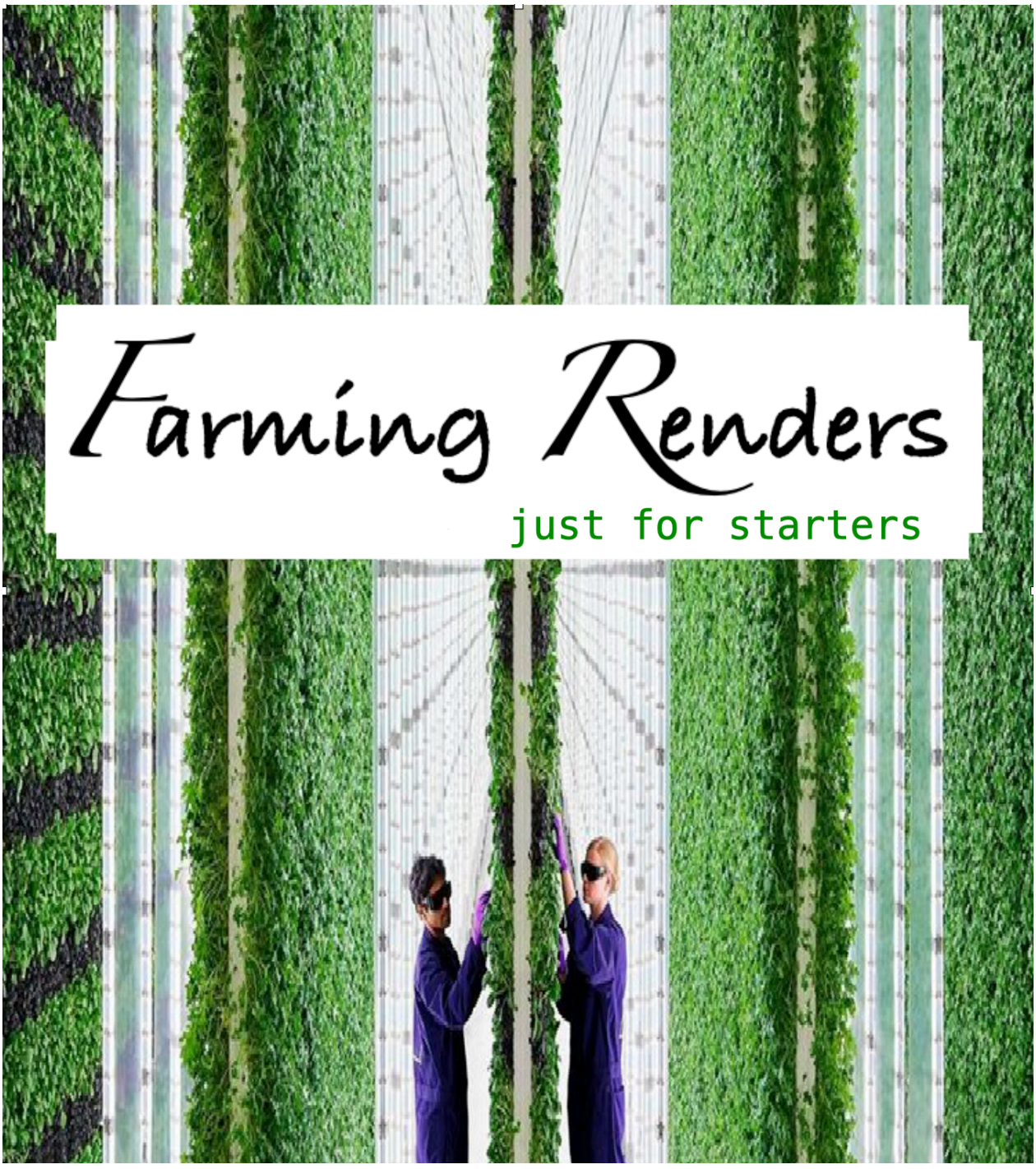


A guide for Blender users to construct and operate an affordable, small-scale, self-contained render farm that can be operated from a domestic studio.

FOREWORD

This handbook has four parts, a primer, a host build guide, a software installation instruction and an operators manual.

The primer exposes the technical background and development methods on which the solution is based. It is intended for newbies to Linux and local area networks who want to clarify some doubtful points so they can focus on building the render farm.

The host build guide does not aim to achieve the high performance of a commercial installation. Its goal is to establish an easy to operate, cost-effective, flexible rendering platform that can be operated from a studio with access to only limited domestic electrical power. The hardware build is designed to enable the installation of the essential software for using both CPU and GPU rendering.

The software installation instruction contains detailed procedures starting from a bare hardware platform to a working render farm. It should first be read carefully until it is clearly understood and double checked if any content is uncertain.

The operators manual includes the basic render process and routine maintenance.

As the title suggests, this is a starting point from which improvements in harware and software are posssible however it is a workable, if highly manual configuration.

Wayne (Wazza) McGrath

Part 1 - Primer

Render Farm Overview

Design Guidelines

The design and configuration goals were aimed at limiting administrative overhead. This is achieved by using a standard software configuration. Host machines are constructed to a minimal technical spec as there are advantages to minimising variations in build. Costs are reduced by using remote access without the need for individual monitors and keyboards on each host. Achieving these goals requires customisation above what is needed for a standard installation 'out of the box'.

This render farm design utilises four classes of host machine:

animation (artist) workstation - used for modelling, sculpting, animating, audio editing and producing a completed animation file ready for rendering.

master render host - used to configure settings on the animation file to be rendered, initiate render tasks, accumulate rendered images and perform image rendering and compositing tasks,

r**ender host(s)** - used for the single purpose of reading an animation file, rendering frames and returning rendered images,

Network Access Storage host (NAS) - used to store animation resources and archive completed animation, audio and video files.

An animation workstation will require a CPU architecture with fast single core operations while still allowing sufficiently fast multi-core operations. It will also require GPU support and multiple fast memory channels and storage devices. A workstation that will not struggle with complex animation scenes needs to be built to a performance specification. That said, an artist may use their preferred platform as long as it is viable. However to meet requirements for efficient rendering and flexible networking, the master, render hosts and NAS will use Linux in order to minimise cost and use the inherent capacity of Linux for customisation.

A remote desktop connection is used to minimise the need for individual monitors and keyboards. To achieve fast LAN communications, the master and render machines are net-worked using a Gigabyte Ethernet switch. The switch can be isolated from all other networks during a render operation.

LINUX UPSIDE

A render farm is a special purpose application of hardware and software technologies including multi-core CPUs and GPUs, highly efficient interprocess communications and optimised device drivers and task scheduling. Windows and Mac provide general purpose solutions that can be used as host machines but at a premium. Linux is free, reliable and is more flexible due to free and easy access to software and tools. There is an advantage in having the option to develop custom solutions to specific issues.

Customisation sometimes depends on access to source code and Linux is founded on the principle that a solution provider should have code access to de-bug it, improve it, expand it, remove security flaws or integrate it with other software. Linux provides the opportunity to easily add additional features and to some extent, remove unneeded functionality.

Software bugs are inevitable and getting them fixed can make a big difference. Commercial systems are generally not responsive to reported bugs and fixing a bug is subject to commercial considerations however Linux developers value their reputation and may be prepared to correct a problem and release an update.

LINUX DOWNSIDE

The kernel has evolved over numerous versions and development has forked into numerous distributions or 'distros' resulting in potential for incompatibility. Adopting Linux assumes sufficient knowledge to perform tasks by accurately entering commands via a terminal. This can be risky as not all commands work the same way on all versions. The shell used to interpret commands and the package managers (dpkg, RPM, SNAP) used to install applications may vary. This means some 'how to' guides could be misleading. Care is needed to ensure they are relevant.

Although not unique to Linux, additional utilities may need to be installed and kernel dependancies may need to be updated prior to a compilation or installation of new software, so it is necessary to determine what dependancies are currently installed to avoid problems.

Distros are categorised as 'light', 'standard' or 'server' with each being optimised for different roles. A good choice of distro is one that provides only the essential kernel and application services required to operate as a system but all of the device drivers, utilities and tools needed to do rendering. Unfortunately the 'light' versions of Linux are customised for older machines and may not support all the services needed for more recent high performance devices. There are standard versions with a full-feature desktop GUI and an excellent range of end-user applications installed, however the GUI consumes resources needed for rendering and most applications are unneeded on a ender host. Server versions are preferred by operators of large scale render farms but they are 'headless', meaning they don't have a desktop GUI and are optimised to support large-scale software such as databases and web servers. All that is required is a 'host' that is optimised to support one application, Blender and a few desktop utilities so a desktop GUI, albeit a scaled down one, is necessary.

A thorough search is unlikely to find a distro that can just be downloaded, installed and then is ready for rendering. Customisation is inevitable. Installing additional software is straight forward however removing unneeded services to free up resources for rendering can't be achieved without expert knowledge of services, utilities, kernel dependencies and boot processes.

LINUX CUSTOMISATION STRATEGY

Successful customisation is more likely if a suitable version of Linux is used to begin with. This does not imply any particular version is not useful, just less suitable in the context of rendering. The desktop GUI is often what influences the choice of distro however it is not significant for render farm customisation.

Although a gross simplification Linux can be thought of as a software architecture consisting of a kernel that controls the basic operation, a suite of application support services, a user interface and a set of end-user applications. Linux has developed into several major variants. Within those variants are a wide range of versions available from 'distro' developers. A distro will be based on the kernel of one of the major variants but will offer alternative configurations of services, user interface and end-user applications. A distro is often aimed at meeting the needs of a niche user group, e.g. virtualised environments, office workers, specialist applications, home users or gamers.

The candidates for a starting configuration are a 'light' , 'standard' or 'server' distro. The light and server distros require solving difficult customisation problems due to insufficient device drivers or lack of a GUI. Customisation is mainly achieved by adding additional services, removing un-needed services or compiling additional modules into the kernel. The services on standard Linux can be easily enabled, started, halted, restarted and disabled and installing a required service from a supported repository is usually a straight forward administrative task but compiling a module into the kernel, for example a device driver, may be hit and miss without an intimate knowledge of the process.

Most all standard versions will typically have good support for custom services via their on-line repositories. A distro that incorporates 'closed' or non-open source device drivers can be a real plus but some developers hold firmly to an 'all open source' configuration. Some have support for closed device drivers and options to limit the number of end-user applications, however they mostly include a desktop GUI user interface that consumes significant memory and processor resources.

Given that there is less risk in managing services on a desktop version than in installing additional services on a light version, and that a X-windows GUI will be needed for remote connections, a practical approach is to select a proven, current desktop version with support for closed device drivers and an option for a minimal installation.

HOST IDENTIFICATION

Efficient network operation will require that each host machine is quickly identified by assigning host names, allocating IP addresses and recording MAC addresses. Hostnames can be assigned as a unique name. IP addresses are allocated from a pool of available address numbers. MAC addresses are serialised by an industrial organisation, allocated to manufacturers and used to electronically label an installable device.

Hostname

There are several mechanisms used to identify a specific machine or group of machines but no one single identifier can be used in all situations. Groups of machines may be identified by the name of the network they are connect to or a workgroup name assigned to several machines that share resources like files and printers. Hardware devices installed in a machine have built in identifiers but they could fail and be replaced. A single machine may have more than one network address if it is connected to more than one network (e.g. Ethernet and Wifi). So a mechanism is needed to uniquely and permanently identify a specific host machine. The hostname is a human readable name given to the operating system within a host machine. Host means 'host to one or more applications'. It must be unique within the network(s) the machine is connected and is used as the basis of a resource locater to identify and access resources on the host such as a shared file, e.g. // hostname/directory/filename. Each host on a network can maintain a lookup table of other hosts by recording their hostname in a special file. A more generalised naming convention is used to identify hosts on the Internet. An Internet host name is recorded in a Domain Name Service (DNS) and its resources are identified by a universal resource locator (URL). A Hostname lookup table is required to locate resources used by the render operation however a DNS is not required.

The Hostname must be alpha-numeric and can be dot separated elements up to 253 characters in length however a single element of up to 63 characters will be sufficient.

IP Address

An IP (Internet Protocol) address is a number, like a post-box number, that is the primary means of routing communication messages between hosts in a network. Each host transmits and receives messages via an internal memory address called a communications socket. The external socket identifier consists of a port number plus an address number, the IP address. Port numbers identify a communications protocol and are allocated by an organisation that controls Internet technical standards. IP addresses are normally allocated dynamically to a host by a Gateway/router for a limited period of time (lease) after which it expires and is returned to the allocation pool. This means it is not possible to be absolutely sure of a hosts IP address at any given time and in turn socket to socket communications is not reliable. The issue is resolved by using a network protocol that broadcasts a request message to the entire network and records all responses in a lookup table that associates the allocated IP address with the permanent Hostname.

To avoid spending valuable time searching for IP addresses, the hosts in a render farm must have a permanently leased IP address. The permanently leased or static address is registered with the Gateway/ router to ensure it is not reallocated and registered with all other hosts for efficient communication. IP addresses are also recorded by the Ethernet switch to optimise connections and transmission rates.

An IP address is represented by four dot separated numbers each in the range of 0-255. The protocol ports are represented by a four digit number.

MAC Address

A media access control (MAC) address is a unique hexadecimal number that identifies a network interface card/device (NIC). If the NIC is moved to a different host, the MAC address goes with it. Separate MAC addresses are used for an Ethernet NIC and a Wifi NIC. The ethernet MAC address is used by the network switch to enable hardware level switching and also used on the Gateway/Router to associate a NIC with a statically allocated IP address. Care must be taken to obtain the ethernet MAC address and not a Wifi address.

A MAC address is represented by six, colon separated hexadecimal numbers.

Gateway and DNS Addresses

A host on a LAN is not able to communicate directly with servers on the Internet and must pass requests through a router device that connects to an Internet Service Provider (ISP) via a modem. The combination of router and modem is called a gateway. The gateway also provides other services to hosts on the LAN such as allocating IP addresses (DHCP service) and hosting sharable USB file storage.

The gateway has two permanent IP addresses, one for use in the Internet allocated by the ISP and one for use in the local area network allocated from a local subnet. The local subnet range is 192.168.0.1 thru 192.168.0.255 and the gateway is allocated the first address in the range, 192.168.0.1 Each host on a LAN needs to register the gateways local IP address to use when communicating with the Internet. An external Internet IP address is not required by the hosts on the LAN. The gateway uses its Internet address on behalf on local hosts and so the same local subnet range can be used on all local area networks. In the Internet, a LAN host is effectively identified by two IP addresses, the gateway address and its local area address.

A DNS services for locating domain names is not available on a LAN, but a host on a LAN can use any available external DNS service by recording its Internet IP address along with its IP adress and the gateway address. If a preferred DNS address is not recorded on a host, the gateway will provide a default server address usually one preferred by the ISP.

Render farm hosts do not need a DNS service as might a gaming machine, so the DNS address should be left to default to the Gateway IP address. It may be more secure to avoid well known DNS servers that potentially could probe the communication ports of a LAN based host.

ISSUING COMMANDS

An administrators job is primarily issuing commands (accurately and precisely). A command is a directive to execute a program. The command may have optional parameters that are passed to the program to modify its behaviour. They can be issued from from a desktop environment via a graphical launcher program however in Linux, commands are typically executed from a command prompt ($) within a desktop program referred to as a terminal emulator. Commands may be issued directly from the console on a headless host that has no graphical desktop environment. Note: The up arrow and down arrow functions are very useful to recall previous commands and make the job easier.

Programs

A program command is used to initiate an executable program. An executable is a file of machine instructions with a prefix of information that allows them to be loaded into memory and fetched by the CPU for execution. The command is the name of the executable program file and may have parameters to modify program behaviour. An example is the command to initiate Blender, $ blender.

Scripts

A scripting language is a specialised programming language that can be embedded and interpreted within an executable. A script can be used by a web browser to add additional functionality to a web page or can be used within Blender to add additional functionality such as an add-on. Numerous scripting languages are available. Some are specialised for web page functionality e.g. JavaScript. Others are general purpose languages that can run independently or within an executable, e.g. Python.

Shell Commands

The commands used to direct the kernel and manage the file system are entered into a command interpreter program called the shell (usually the bash shell). Shell commands can be executed from any place in the file system. Some commands can only be issued by a system administrator (sudo commands). Shell commands often have optional parameters that modify the behaviour of the command.

Shell Scripts

Multiple shell commands can be grouped together in a text file called a shell script and executed with one command based on the file name. In Linux it is not possible to code shell comands in a text file and immediatly execute it. To protect against inadvertant mistakes, the text file permissions have to be changed to make it recognisable as executable commands (chmod). Shell scripts are typically used to automate admin functions. As well as shell commands they can be used to execute programs, script languages and other shell scripts.

Directives

A form of command that is not issued via a shell or executable is a configuration directive. Directives are stored in a text file with a specific format that is interpreted by a kernel program during the boot process or during background system operation. The most comprehensive set of directives is available via by a module called systemd.

COMMAND ALTERNATIVES

Linux has a history of accumulating numerous single purpose utilities in keeping with a philosophy of writing self contained functions. (Faster to develop, easier to maintain and programers can stay out of each others

way.) The downside is there are a lot of commands to learn and sometimes it is uncertain which utilities are present on a host. The need to standardise on a multi-function module now dominates the need to develop multiple alternatives. This can be confusing on a system configured with a multi-function module plus numerous single purpose modules that perform similar roles. Nonetheless the systemd kernel module is a ubiquitous tool for controlling many aspects of how Linux functions, so if in doubt use a systemd command.

Systemd Units

Systemd utilises directives in a configuration text file called a unit file. There is no absolute requirement for where a unit file may be stored, however a good management practice is to store them in a folder structure that organises them into service units that define a service, drop-in units that can overwrite aspects of a service and run units that execute once only. Typically service units are stored in /lib/systemd/system, drop-in units are stored in /etc/systemd/system and run units are stored in /run/systemd/system. If these conventions are observed a software package update process will be more reliable as the package manager can easily locate updatable service units.

Shell Scripts

Shell scripts are typically used to automate admin functions. They can be stored anywhere within the file system however a good management practice is to default to a particular folder depending on who needs to use the scripts. Scripts used by just one user may be stored in their home directory i.e. /<username>/home/bin. Scripts available to all users may be stored in the folder structure for user applications i.e. /usr/local/bin. Scripts used only by a system administrator may be stored in a special folder in the user folder structure i.e. /usr/sbin (bin implies binary code).

Potted History

The core of render engine functionality is the ray trace algorithm, its variants and enhancements. Ray tracing algorithms perform many geometric and trigonometric calculations. They were originally coded to use a single CPU core but refactored to use threads when multi-core CPUs became available. Although multi-core CPUs are utilised for parallel threads the commercial driver for developing multi-core CPUs is a server that can support multiple general-purpose users, not one special application of parallel floating point calculations.

The first GPUs were used for video display and provided only Z buffer functions. Rendering algorithms such as the Bresenham raster algorithm were still coded in application software. Over time, specialised high performance processors like numerical processing units, digital signal processing units, large scale floating point arrays and gate arrays were developed for special near real time applications. Game console manufactures notably Nintendo, developed the techniques for integrating the units to operate in parallel. The processing units and integration techniques were combined with the video raster devices and the modern GPU was born. The next phase of development was to develop ray trace processors that made massively parallel concurrent floating point operations possible. The processors are supported by parallel processing models on which ray trace algorithms are practical.

Although there remain many algorithms used in animation that require a powerful CPU the performance of a GPU with ray trace processors can greatly outperform a multi core CPU, although there are arguments that the true ray trace algorithm and therefore image quality is compromised. The commercial driver for high performance GPUs is not image rendering but artificial intelligence, science based applications such as weather forecasting and financial based applications such as crypto coin mining. The objective for ray trace capability on GPUs is not the highest possible image quality but near real time rendering of video game displays with complex spacial content.

It may be misleading to regard either multi-core CPUs or GPUs as ideal for rendering images for animation. One is too slow, the other has questionable quality or imposes specialist technical knowledge on creative artists. Older or low end GPUs will not have ray trace capability. Ray trace capable GPUs are identifiable by their support for a parallel processing model and the API libraries and device drivers that enable them.

CPU vs GPU Concurrency

The CPU rendering model defines computational threads but relies on the operating system kernel to create processes that allocate the threads to CPU cores. A render engine can utilise many CPUs concurrently by using the kernel scheduler module as middleware. The inputs of each concurrent operation are all the same and the output quality of each operation is consistent with the inputs. This fact, plus the capacity to render very large hi-poly 3D scenes is reason for retaining CPU render capacity.

GPU middleware accesses hardware functions via an abstraction layer that hides the details of its ‘metal’ operation. GPUs also have multiple modes of operation, e.g. raster, floating point numerics, ray trace tensors, digital signal processing and artificial intelligence inferencing. As each requires its own processor and abstraction layer, GPUs may not have identical implementations of an API. Concurrency has two levels of complexity, concurrency within a GPU's devices and concurrency among different GPUs.

Currently there is no heterogeneous parallel processing standard for render engine developers to adopt. Programming different GPUs to work concurrently to the same quality is difficult or impossible. It may be

misleading to say that GPUs are superior to CPUs. There are tradeoffs between throughput, image quality, power consumption, software development and ease of use.

GPU and Render Engine APIs

Application software accesses the functions of a hardware device via a type of contract called an Application Programming Interface (API). Essentially the contract specifies what functions the device will perform if given specific commands and data by the software.

The most common video display APIs are OpenGL and DirectX. Fortunately most vendors support OpenGL as the standard video API, however an end-user application like Blender may require a particular version of the OpenGL standard and cannot use an older GPU that does not support it. In other cases the GPU may be compatible with Blenders OpenGL graphical display but incompatible with the API used by the render engine to access a GPU. Application sofware, render engine software and GPU middleware can be at odds. Careful analysis of API version requirements is essential to prevent wasted time and money.

Parallel processing models, middleware and APIs have a longer development time than a GPU device model and will likely be around for the release of several GPU models.

GPU Selection by Shortlisting

Choosing a standard CPU or GPU may be achieved by performing a rational selection exercise. Shortlisting will avoid ending up with incompatible software and hardware. It may also assist with understanding GPU technologies and their upgrade pathways.

The selection process begins with the animation application package and all potential render engines it might use as these items contain the subject matter that require the most investment in learning. In this case Blender has been selected along with Cycles and potentially other render engines that are compatible with Cycles. Graphical display APIs and image rendering APIs/models are the main determinants of GPUs for a render farm.

Start the list by noting what graphics APIs are supported by the application and what rendering APIs are supported by the render engines. Where there are multiple supported APIs, research into their relative merits can be used to note the preferred APIs. As used here, API includes processing models, middleware and hardware abstraction layers.

Next compile a list of GPU models that support both sets of APIs. GPUs are often manufactured in a series beginning with a base model followed by variants with increasing capabilities but all based on the same architecture. APIs are typically standardised to operate on the base model and subsequent devices in the series, so a list of GPU models is useful to identify APIs and potential GPU upgrades. Once it is clear what APIs are viable the preferred APIs can be revised and the GPU list reduced to the models that support preferred APIs.

Finally the GPUs that have efficient Linux device drivers are determined and then the most affordable and powerful GPU can be selected.

CPU Selection by Comparative Performance

Selecting a CPU is less complicated however there are distinctions that can be made between a suitable processor and one that is no so suitable. All CPUs will have a base clock speed and core count that are the first features to directly compare. The CPUs in a render farm must not overheat so over-clocking is irrelevant as a selection feature. Some CPUs have additional circuits that enable more than one software thread per core, these are referred to as logical cores or threads (threads are really a software thing). Most of the instructions will be computational so a CPU core with a superior Arithmetic Logic Unit (ALU) will perform faster. Older multicore CPUs manufactured for server operation may not have efficient ALUs as they were not critical. CPUs have internal cache memory (LU) used to store the data most likely needed for the next instruction and the larger the LU size the better the performance. Chip fabrication processes are measured in nanometers. The smaller the nanometer technology the faster the chip can operate. The more logical cores the better. Fewer than 8 is not recommended.

Parallel Processing Standards

In order to use a combination of CPU cores and multiple GPUs, a parallel processing standard is required. The Internet is viable due to the software standards issued by the W3C, however parallel processing architectures are still under development and there are no equivalent standards organisation. Some use distinct channels for CPU instructions vs GPU instructions and others using a single channel for both. Understandably, software developers prefer an API that offers a single channel accessible with a single module. This narrows

the choices of API down from the wider concerns of parallel computing to the APIs supported by gaming GPU vendors and render engine developers.

GPU API Standards

OpenGL is widely supported by GPU vendors as a graphics display API however only the later versions may be compatible with current application software. There are several competing APIs for GPU image rendering via a software render engine. The most prevalent are Metal, CUDA, Optix and OpenCL. OpenCL is standards based but primarily supported by AMD Radeon. Metal is specific to AMD Radeon GPUs on Apple Mac and CUDA and Optix are specific to Nvidia GPUs beginning in the GTX and RTX models. Vendors usually don't participant in the open source industry to provide parallel processing device drivers and as a consequence, open source alternative drivers while a worthy effort, are reverse engineered and often inefficient when compared to a closed source option from the manufacturer.

OpenCL is a standard supported by AMD and nominally by Nvidia, however AMD have been active in assisting the development of device drivers for Linux and as a result AMD GPUs are compatible with most Linux distros. Nvidia drivers are included on only some Linux distros e.g. Ubuntu. Currently pop OS is regarded as the benchmark solution provider for using Nvidia GPU technology. Although AMD GPU drivers can be readily installed on Linux their use is limited by the fact that current versions of the Cycles render engine only supports AMD cards with GCN Next 2 or later hardware architectures. Also, it is generally acknowledged that Nvidia CUDA and Optix technologies in particular, perform better (faster) on Linux when using proprietary (closed source) drivers.

To add to the mix there are several alternative render engines, some of which are compatible with Cycles, with potential to perform faster due the use of compiled languages and optimised object code. AMD Radeon has developed the Prorender engine with device drivers for all major operating systems, however it uses its own material library.

In summary, the current situation means either a choice has to be made between Nvidia GPUs with CUDA/ Optix technologies and AMD GPUs with GCN2+ technologies. Otherwise the render farm software must include custom scripts for detecting and utilising whatever GPU is installed, without compromising quality or the render parameters set by an animator.

Network Rendering

A render farm the builder is spoiled for choice of operating system, motherboard, CPU and GPU, however the list of choices for a small scale network rendering controller is thin.

A Short Digression

A puzzle for archaeologists is that ancient deities from Sumeria and other parts of the globe are depicted as carrying a hand bag and they ponder what could be in the bag. A software architect might not be so perplexed and immediately quip that it is obviously 'functionality'. They would know this because software is just a bag of functionality. They also know that over time adding more features to the bag can lead to functional bloat. A software architects job is to determine what functionality is needed and where and how it can be deployed. At some point non-essential functionality must be redeployed to make way for new essential functions. Users rely on software architects to make decisions about what is essential for animation functionality but what is in the deities bag is essential management and control functionality. Modern software architecture separates the requirements for control from those of subject-matter or data. Like the winged deity, render farm operators must make their own arrangements for their bag of control and management functions.

Adopting an Operating Model

Operators of an animation studio must determine what degree of control they want over the render process and how much capability they have for implementing their management requirements on top of their subject-matter and creative requirements. Rendering capability can be implemented with three basic models, the cloud, the crowd and the sole user.

The cloud provides functional transparency of the rendering process. The user uploads a .blend file via a web browser and waits for notification that the finished output is available for download. No knowledge of, or control over, the rendering process is necessary on the part of the user.

The crowd model is an aggregation of several users with local hosts using the the internet to inter-connect them. The functionality is not completely transparent. The end-to-end process is managed by a third party master controller but each user must install slave software on all their participating hosts and control their operation.

The sole user model is an aggregation of hosts on a local area network that operates independently of the internet and third party controllers. Control of the entire end-to-end process must be implemented and managed by the operator using a commercial render farm package or a custom solution.

The network rendering solution that follows is based on the sole user model. It does not preclude the use of the cloud or the crowd but makes no particular allowance for them.

End-to-end Process Requirements

Process requirements need be clearly articulated as the basis for a technical solution.

A drop-box is needed so an artist can upload an animation file for processing and go back to work without being concerned about the render farm operation. The drop-box is also needed to post a job ticket containing job related information such as wether the job is production, rework or test, the frames to be rendered and any render options offered by the render farm.

A work directory is needed to set render properties. Although properties can be set on the workstation, the devices and render engine may be different to those on the render farm. There is more flexibility and consistency if some settings are targeted at the devices and render engines present render hosts.

A central distribution point is needed to guarantee access to read render input files for the duration of the render process.

A central aggregation point is needed to guarantee access to write output files for the duration of the render process.

The implementation of the distribution and aggregation points should not determine how files are distributed or aggregated. They should function as shared storage locations that support multiple rendering models.

The implementation of the rendering task should not determine how the render is initiated. A render task should be able to be initated from a comand line, a GUI or even voice activation and remote control.

Some requirements are not-so-obvious.

Does the farm need to support concurrent render processes?

Is there a need to estimate render duration time?

Is there a need to halt the render process for an unspecified period and then resume?

Is there a need for rework to be rendered to the same settings and quality as the original.

Is there a need for a status report?

There are obvious disadvantages if an artist can't continue to work during a long render so offloading a test render to the render farm is advantageous. Also, parts of a previous render may need to be reworked. To meet these requirements the distribution function should be able to to allocate all hosts to a single job or alternatively allocate them among concurrent processes such as test jobs, production jobs and rework jobs.

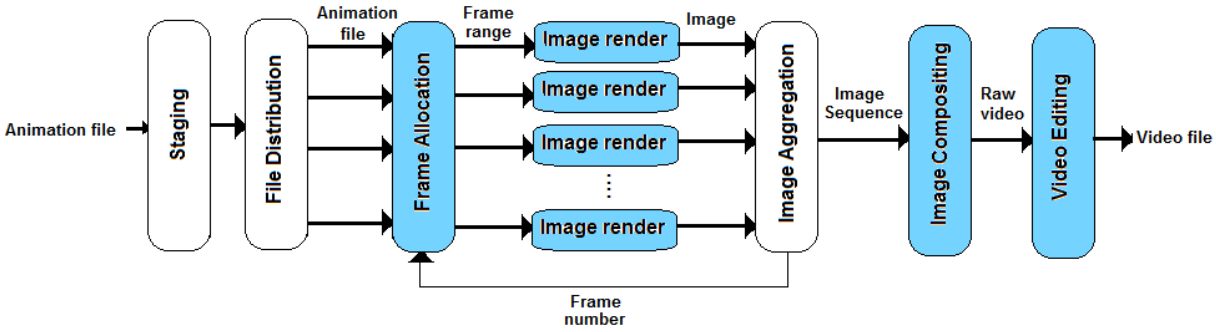
The larger the animation the more difficult it will become to assess how long the render will take. An estimate will be useful in deciding if some adjustments will need to be made or a contingency to be utilised. To meet the requirement the drop-box function should provide a job assessment.

As the farm will nominally operate from a domestic premises there may be occasions when the process is interrupted by external factors. A power outage can be managed by using a UPS however that won't deal with all possible hazards. Any problems can be somewhat managed if the render does not have to be restarted from the beginning. There is a requirement to establish way points in all stages of the process that can be used to restart a partially completed job.

It is self evident that a completed job, over-due job or failed job needs to be reported at the earliest. There is a requirement for a master controller to log job status and provide a status report on request, and for notification facility to alert of a failure or notify completion.

Network Render Process

The main stages of the render process are shown below. Fortunately Blender 2.83 LTS retains features that can be utilised to enable networked rendering and much of the entire process, as shown in blue. Data oriented tasks such as rendering, compositing and video editing are performed by Blender however staging, distribution and aggregation are process control functions.



END-TO-END NETWORK RENDER PROCESS

Centralised Distribution and Aggregation

A basic requirement for a render farm is for a render host to access the input file from a central distribution point and write the rendered output image files back to a central aggregation point.

Process Control

Another basic requirement is for a flexible means to control the render process. The command line provides several options for issuing commands ranging from fully manual control, distributed shell files to centralised shell files or execution of commands via a programming language application.

The Network Render Solution

Remote Desktop Access

Access to the desktop of any host can be implemented using a remote access utility such as VNC. This solves the need for interactive and manual control of hosts from a central point, i.e. the master host or a workstation.

Remote Commands

Commands can be issued from the master host across the network to a render host using the Secure Shell utility. The master host effectively logs on to the render host and commands may be issued as if from the render host command line. This includes initiating shell scripts and programs. Secure shell also supports piping the commands in a local (client) shell script to execute on the target host (server).

Network File Sharing

Centralised points for storing files are called network shared directories or just “shares”. Connection to a shared directory is achieved via a network protocol that exchanges well defined formatted messages as blocks of characters and multi-media content. The Server Message Blocks (SMB) protocol and its variants the Common Internet File System (CIFS) are implemented on Linux as Samba. It employs a request/response protocol and has two software components, a client that issues requests and a server that issues responses. Despite some drawbacks the Samba implementation of SMB is a workable choice for setting up shared directories on networks with Linux, Unix and Windows hosts as many NAS and file servers use it and there is good support from the Linux community. The standard Unix/Linux protocol of NFS remains available as a contingency for file sharing.

Shared files are implemented on the render farm by installing a Samba service as a server on the master host and creating shared directories that are accessible by render hosts as SMB clients. A Samba client does not need the Samba service to be installed. Clients can access shared directories located on a server using the CIFS utility, smbclient module or the Nautilus file manager. The CIFS utility can be used to to permanently mount the shares on each render host at boot-up. The mounted shares appear in the file system as a local directory. Nautilus file manager is equivalent to Windows Explorer and is useful for interactively browsing the network.

Seperate shares will be configured for test renders and production renders as well as shares for submitting animation files and retrieving completed video files.

Background Processing

Blender can run as a background task and be passed command line arguments including the name of the input file and the name spec for output files.

Render Automation

Automation includes preprocessor functions such as setting Blender properties and render task initiation functions. Automation shells will be run from the master host, either locally to perform preprocessor tasks or using secure shell to run background rendering tasks on the render hosts. Alternative automation models will be implemented to cater for maximum discretion of the animator subject to resolving any problems arrising if the frame allocation method is not set or when the animation workstation has different device types to the render hosts.

Frame Allocation

The Blender Output Properties Tab has options to specify the start and end frames and options to switch on a Placeholder property to indicate a frame is currently being rendered and to switch off an Overwrite property to indicate any completed frame should be skipped over. Using a central output shared directory and these properties and/or command line arguments Blender can effectively be used to co-ordinate the allocation of frames among render hosts.

CPU and GPU Rendering

Both CPU and GPU rendering will be implemented. GPU rendering is faster than CPU rendering however a CPU render will have more system memory to process very large files. To some extent is practical to use GPUs from different vendors on the same render job by using the hosts Blenders system preferences to permanently set the type of GPU device (CUDA, Optix, OpenCL) and using a Python script to set local device properties prior to a render. To achieve consistent image quality, a standard GPU model is recommended.

The render farm does not preempt the intentions of an animator so some settings that will impact on render time, e.g. baking, will remain the prerogative of the animator. Each preprocessor will save a new version of the original and add a label to the beginning of the file name so it can be clearly identified for rendering. The labelling is useful to endure that any subsequent rework is rendered to the same quality. The preprocessed files are retained with the original for archiving. The build has multiple cores but only one GPU to limit power consumption. An advantage of having one GPU is that the render hosts Blender user preferences can be set to the relevant render type, i.e. None, Cuda, Optix or OpenCL. This simplifies the python script to switch between CPU device or GPU device and avoids problems that can occur in the wrong combination of type and device are set. Options will be available to perform a CPU render, GPU render or both.

Preprocessing

Preprocessing is non optional. Its essential function is to switch off the Overwrite property and switch on the Placeholder property so the frame allocation method will work. All preprocessor scripts must perform this function on the animation file and the settings maintained when archived. Animators should change their default settings to enable the method however as the standard Blender settings are the opposite to what is essential, all files must be preprocessed before rendering to avoid a disaster. The function will also be performed locally on all render hosts Blender as a contingency however these settings cannot be saved to the animation file. Other preprocessor scripts set settings for quality, render speed and render device.

Model 1 Render

The animator has options. Prior to rendering the settings in the submitted animation file may, at the request of the animator, be over-written using shell scripts with Python preprocessor functions. This is to achieve consitent quality, opt for the fastest render time (usually for a test) or render with the original settings. The quality options available are High Image Quality (HIQ) or Fast Render Time (FRT). The animator can also request either a GPU or CPU render and the preprocessor will set optimal render settings for either. The file is preprocessed in a staging directory sepate from the shared directories. CPU thread allocation is set to auto, leaving scope for the operating system to manage concurrent render processes. The image sequence type also has to be standardised as there is potential for duplication of the same sequence name with different extensions, e.g. fram12.png and fram12.jpg.

Model 2 Render

The animator has the same options as Model 1 and GPU processing will be as per Model 1 however CPU thread allocation will be based on a formula. The number of threads allocated depends on the numer of logical cores on the processor. This model is for allocating the maximum threads to a render while leaving threads for the operating system and other processes.

Image Render and Compositing

The output of all renders will be a sequence of images that need to be rendered into a video file. A non-compression video format is highly recommended for editing as an mp4 codec will reduce the quality on every edit. If mp4 is required it should be the last conversion. Image rendering can be performed on the master host. Compositing maybe performed on the master host using its Blender installation. A directory will be created on the master host for shell .blend files with prepared compositing nodes. Compositing will be a manual process but image rendering could be automated with a Python script. Audio editing should be performed on a workstation with access to the required audio applications.

Render Planning and Scheduling

A simple text job ticket can accompany the animation file at submission, to specify render options. A folder for storing job tickets will be created on the master host. A spread sheet can be used to plan and allocate render jobs to hosts. The spread sheet can be a simple paper pro-forma or a spread sheet application can be installed on the master host. Estimation of render times can be automated using python scripts to collect sizing data from an animation file (e.g. number of vertices and number of frames) and calculated by the scheduling spreadsheet. Alternatively the estimate can be calculated by the animator and included on the job ticket. There are potential for scheduling errors if the animator does not provide a reasonable estimate or if they calculate it based on the wrong render settings or a different render type than used in the render farm. However there are advantages in calculating the estimate at the workstation. Although it is somewhat complicated, calculating good estimates is a major factor in getting the most use of the render farm with lest wasted processing time.

The animator can set Thread mode to Fixed and threads to 1, then render a single frame using the CPU to obtain the duration per frame (in seconds). This value can be regarded as the Likely Thread Duration. Then metrics such as the number of vertices can be used to calculate a second per thread duration estimate. A third estimate can be calculated from a history of similar renders. The longest duration of these becomes the Worst Case Thread Duration and the shortest the Best Case Thread Duration. These values are submitted on the job ticket along with the number of frames and used by the scheduling spreadsheet to calculate a weighted average, the Expected Per-thread Frame Duration . This value can be multiplied by a GPU accelerator factor to obtain the Expected Per-GPU Frame Duration. These values can be saved for history based estimates then used to calculate the Expected CPU Render Duration and Expected GPU Render Duration. The Expected CPU Render Time is calculated as the Expected Per-thread Frame Duration x number of allocated threads x number of frames. The Expected GPU Render Time is calculated as Expected Per-GPU Frame Duration x number of frames.

Process Monitoring

Controlling the infrastructure so that it applies the maximum resources toward the render process implies halting other unnecessary processes and redirecting the freed resources. Infrastructure resources include CPU cores and threads, RAM, storage, input/output channels and network traffic. Process monitoring tools are installed to identify where delays and bottlenecks occur due to resource content with unneeded processes. Different tools may be needed depending on the render device in use.

Performance Optimisation

Performance optimisation aims to reduce resource contention and direct the maximum machine resources to the render engine. Resources under contention include CPU cores, lu memory, RAM, cache, storage devices, data bus, control bus and communication channels. A process may have to wait on hold until shared resources become available so reducing the waiting period is beneficial. This may be achieved by halting all non-essential services or adjusting their priority and resource settings (e.g. swapiness). Performance monitoring tools are available to identify bottlenecks.

In some cases the contention for resources may occur between the kernel and the render process. If all CPU threads are allocated to the render task the kernel will need to continually interrupt the processes and obtain control of cores to perform critical machine functions. The interrupt process is time consuming and improved performance may result from dedicating separate cores to the kernel process and the render process. Special techniques are needed to manage CPU/thread affinity and scheduler context switching. In other cases the render engine settings for the number of tiles may make a significant difference.

Render performance may also be enhanced by hardware optimisations such using the RAM type recommended for a CPU. SSD memory can be accessed directly from some CPUs and measurable improvements can be made by providing seperate SSD devices and memory access channels for application data and the operating system.

OPTIMISATION TARGETS

Linux can operate in several modes (run levels), with each providing a different level of services. Rendering with Blender requires multi-user mode (run level 3). Unfortunately multi-user mode provides many services and applications that are not needed for the render task. Examples of non-essential applications and services are:

- unnecessary autostart user applications
- automatic notifications services
- automatic application software update services
- automatic operating system updates services
- unused application level services
- unused network services

Services are part of the operating system and managed by the systemd module. Applications are part of the desktop system and managed by a startup manager.

PERFORMANCE MONITORING

Linux has built-in tools that periodically extract data from usage logs recorded by the kernel and average the values to report on resource consumption. In a multi-user system these reports are sufficient to identify a particular user process that is over using resources at the expense of others, however in a special purpose system with multiple CPU cores, the averaging approach may hide short peaks of usage that may be the main cause of contention problems. Additional high sampling rate tools are useful.

Normally there are numerous system processes running in parallel with user processes. The kernels scheduler module can't wait for user processes to conclude before initiating system processes and uses a software technique called an interrupt. The interrupt halts the process, stores all the data and program values currently in the CPU and initiates the system process. When the system process is concluded the data and CPU values from the original process are reloaded and the process continues to run.

A single user render engine can allocate render execution threads to as many cores as are available. Even though the render engine has allocated threads to all available cores, the scheduler may still direct interrupts across all cores with resulting render performance hits. The relationship between the scheduler and execution threads is referred to as affinity. Performance monitoring tools are available to modify the affinity so some cores can be dedicated to the render process without being interrupted.

Some end user applications, utilities, services and shell scripts are started automatically at boot-up by a desktop system launcher program. The startup manager can be used to add, edit and delete auto-startups. CRON is a scheduler used to automate repetitive system admin tasks but may also be used by applications to do house-keeping tasks that free up resources. Systemd is the primary module for controlling kernel services.

HOST BUILD LIMITATIONS

Artist workstations require CPUs with multiple cores but also good single core performance as many modelling and sculpting workloads can only be performed on a single core. Fast memory channels and multiple memory channels measurably improve performance. While interactive performance is important a few minutes or even a couple of hours difference on a background render over several days may not be significant enough to warrant the cost of technologies that are designed for real time gaming experience with a risk of overheating. A host with a prolonged duty cycle is essential.

The electrical power used by a gaming PC can be in excess of a 400 Watts but a domestic power circuit is fused so it can supply limited continuous power without the risk of electrical fires. Assuming the render farm has sole use of a fused power supply circuit, this still places a limit on power consumption of around 2.8kW to 3.7 kW. The objective is to run six render hosts plus one master host within this limit. With a safety margin and scope to add a workstation and ancillary devices a render host will need to consume 400 Watts or less. A high end GPU may use 200 Watts at maximum capacity so the CPU, memory and storage are limited to 200 Watts or less. Effort toward identifying the fastest devices with the lowest power consumption is not wasted in a domestic render farm.

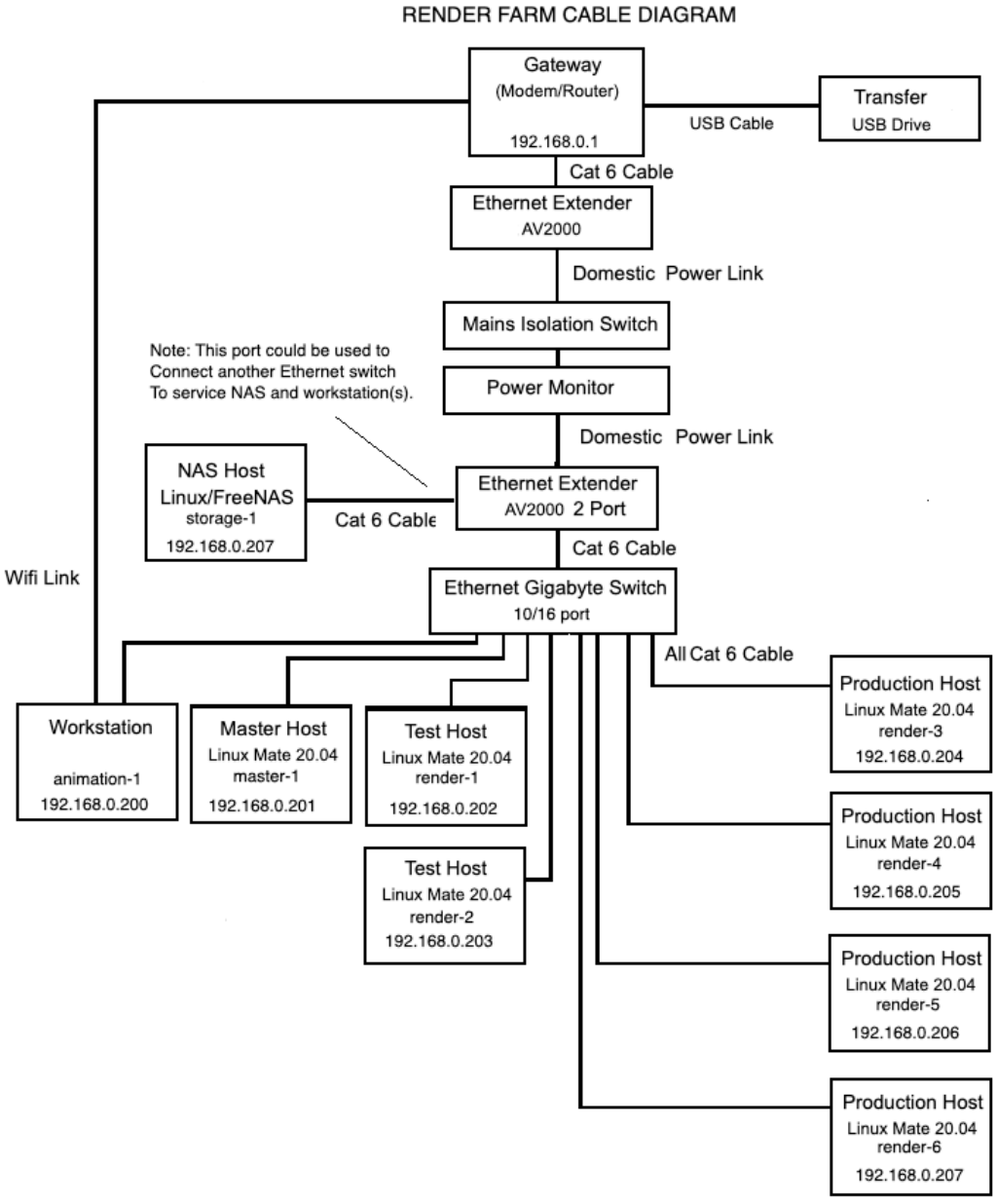
Given similar hardware capability the API used on a GPU can make a significant difference for certain render tasks. Closed source device drivers have a performance advantage over reverse engineered open source equivalents. Currently Ubuntu Linux is preferred as it supports a wide range of GPU technologies. pop OS is an alternative with both Nvidia and AMD driver support.

Stability, predictability and maintenance support is a factor for long running processes so Blender 2.83 LTS is preferred.

Part 2 - Hardware Build

Local Area Network

The heart of the network is an 8, 10 or 16 port Ethernet switch depending on the number of hosts. The hosts need to be connected to the Internet for software installation but must be isolated from all external networks during operation however it is convenient if a workstation remains connected to the Internet via a modem/ router through a wifi link. A power line extender is a practical means of locating the render farm at the best location and serving as an isolation control. The second port on the extender could be used to connect a 4 port switch to service workstation(s) and NAS, further reducing network traffic on the render host switch while maintaining network connectivity. Cat 6 cable is essential to maximum LAN speed.



Host Builds

The minimum facilities required for software installation and networked operation are specified. Any additional facilities on motherboards such as WiFi and overclocking will not be utilised.

Limiting Factors

Apart from a $ budget that may be eased over time, the build is also subject to a Watt budget that will be difficult to increase. The build assumes, and is limited to all the electrical power available from a domestic power circuit. Depending of the power distribution it may be possible to utilise the power from two circuits however it is critically import to determine which circuits(s) will be used and what other appliances will be sharing them. Avoid sharing kitchen, laundry and media room circuits and note that lighting is usually on the same circuit as general purpose outlets. Calculate a realistic Watt budget to aid in making build decisions.

Reliability

Electronic components such are can run continuously when there is adequate cooling. Power supplies with high efficiency ratings are more reliable under high load. Motherboards with more voltage regulators and heat sinks are less likely to have power related problems. Motherboards with high quality discrete components such as capacitors and resistors have longer usable lifetimes. Protection from electrical supply spikes and surges will prevent processor outages, component destruction and increase useful lifetime of components.

Flexibility

The aim is for each host to both a multiple core CPU and a ray trace capable GPU. All hosts need to be mounted in a cabinet with GPU external connectors readily accessible. An ATX micro size motherboard is a conveniently small form factor and sufficient for the required features. Low profile GPU cards and cooling fans may allow more hosts per cabinet. Beginning the construction with a cabinet in mind and choosing the components to suit may be unnecessarily restrictive. A modular cabinet design that can be adjusted to accomodate the components after the fact, may be more flexible.

Component Availability and Sourcing

If funds are no obstacle there are sources of custom built machines using the latest components however if building on a budget there are some traps and that could waste limited funds. High performance multi-core CPU's have been available in the PC market only since late 2014 so anything older may disappoint. Acquiring failed or questionable machines in the hope they can be repaired, may be a side track leading to a dead end.

Markets with volume sales may be where to look. The market for ex-corporate/government PCs is a source of inexpensive working machines that are suitable for a fileserver but all major components need to be replaced in order to build a practical render host. There is a market for motherboards from these same machines and they can be acquired in numbers. It will be cost effective to acquire the components separately and assemble them into a working host. Higher spec CPU's, RAM, SSDs and GPUs are available from the same sources. A source of new components is the market for current generation -1 components. Retailers with stocks of superseded components often sell them in volume at discounted prices. Though more expensive than ex-corporate components they can have the advantage of offering a CPU upgrade path not available in the older motherboards.

If funds are very tight it is feasible to omit the GPUs and use only CPU rendering. GPUs can be installed over time as funds become available. There is not a great difference in cost between a 300 Watt power supply and a 500 Watt supply so it is recommended to acquire the power supplies as specified. The specified Wattage is calculated to cater for the initial build, later upgrades to CPU and/or GPU and remain within the Watt budget.

Master Host Minimum Build

Motherboard - designed for continuous operation over extended period with support for at least 16 Gb of RAM, a PCI 2 x16 extension slot or better, at least 1 SATA 3 memory channel, 1 on board M.2 SSD slot or better, 3 USB channels and an Ethernet channel. Wifi is not required and can be omitted.

Power supply - a minimum of 300 Watts with 24 split 8, split PCI and Sata outputs for flexibility.

CPU - multiple cores with higher than average L1 and L2 memory. Minimum of 4 cores/8 threads at the top of the performance range. Over or high rated cooler fan.

RAM - minimum of 16 Gb as recommended by motherboard manufacturer.

OS/Application - 250 Gb SATA 3 external SSD.

GPU - minimum OpenGL 4.2 support, minimum 2 Gb on board RAM, DVI and HDMI output. - Geforce GT 710 or equivalent.

Render Host Minimum Build

Motherboard - designed for continuous operation over extended period with support for at least 32 Gb of RAM, a PCI 2 x16 extension slot or better, at least 1 SATA 3 memory channel, 1 on board M.2 SSD slot or better, 3 USB channels and an Ethernet channel. Wifi is not required and can be omitted.

Power supply - a minimum of 500 Watts with 24 split 8, split PCI and Sata outputs for flexibility.

CPU - multiple cores with higher than average L1 and L2 memory. Minimum of 4 cores/8 threads at the top of the performance range or 6 cores/12 threads at the middle of the performance range. Over or high rated cooler fan.

RAM - minimum of 12 Gb as recommended by motherboard manufacturer.

OS Storage - 250 Gb SATA 3 external SSD or 125 Gb on board M.2 SSD.

Application Storage - 125 Gb on board NVMe or M.2 SSD.

GPU - minimum OpenGL 4.2 support, minimum 2 Gb on board RAM, minimum 900 CUDA cores preferably 1500 CUDA cores, DVI and HDMI output. - GTX 1650/1660 or equivalent.

Fileserver Host

Motherboard - designed for continuous operation over extended period with support for at least 8 Gb of RAM, at least 4 SATA 3 connectors, 3 USB channels and an Ethernet channel. Wifi is not required and can be omitted. PCI slot optional.

CPU - Minimum of 2 cores/2 threads. Over or high rated cooler fan.

RAM - minimum of 4 Gb as recommended by motherboard manufacturer.

OS Storage - 125 Gb SATA 3 external SSD.

Archive Storage - 1 Tb SATA 3 HDD.

Onboard or integrated graphics. Low end graphics card optional.

Note: Fileserver may be a suitable used small form factor (SFF) PC.

Peripherals

A basic USB keyboard and mouse and an inexpensive, small (around 21.5 in) full HD monitor to share among hosts during software installation and dedicate to the master host during farm operation.

Note: The software solution will include a remote desktop utility to access all hosts during operation but so the animation workstation(s) is not interrupted by farm operation, the master host will need basic peripherals.

Cabinet

A simple design to accomodate 7 hosts and network switch is a hollow case with removable platforms on which the hosts are mounted. Indicative panel sizes are calculated based on the dimensions of a ATX micro motherboard, 500 Watt PSU and low profile fan CPU cooler. Oil coolers will require an additional 20mm (7/8in) case hight per host.

(from hardware shop)
Panels made from 16mm (11/16in) particle board
8 x host platform size 455mm x 350mm (18 in x 13 3/4 in)
2 x case side 1400mm x 500mm (57 1/4in x 19 11/16in)
2 x case top and bottom 500mm x 390mm (19 11/16in x15in)
12 x 30 mm (1 1/4in) countersunk Phillips drive particle board screws
100 16 mm (9/16in) 6g pan head self tap screws
100 12mm (7/16in) 4g countersunk Phillips drive timbre screws
15 L shaped mending plates to suit particle board (for case corners and backstop)
50 metal reinforcing brackets 90 degree to support host panels
all purpose glue
4 x castors
spray paint and plastic corner mould (if desired)
(from electronics shop)
50 10 mm (6/16in) untaped nylon motherboard spacers
7 x LEDS, power-on switch and header wiring if needed
(cut from small plastic chopping board or milk container)
miscellaneous plastic mounting strips to secure external drives and PSU

For each host make a 16mm particle board platform to hold all the components. Arrange host components with:

-PSU mounted securely to the rear flush with rear edge of panel
-motherboard secured to the front-left with spacers and self tap screws, connectors facing out with leading edge of ATX motherboard 425mm (16 3/4in) from rear edge of panel
-external storage mounted to right side of the motherboard

Construct a tally plate for the power-on switch and indicator LEDS out of a metal reinforcing bracket and mount on the front-right on the leading edge of the platform away from the motherboard and GPU connections.

Make a same size platform for mounting the Ethernet switch and ancillary equipment. Securely Mount the switch on the front right edge and 10 way surge protected power board along the left rear side. Position power monitor and smoke alarm.

Complete all PSU, storage and tally plate connections and bench test each host to power-on.

Make a hollow case of four particle board panels cut to the required height, depth and width. But side panels to top and bottom panels, clamp (e.g. picture frame clamps) and make rigid with particle board screws, glue and L shaped mending plates. Mount on sturdy castors.

In the interior of the case, both sides front and rear, measure off 4 host spaces (180mm) from the bottom and rule a line from front to rear a a guide for to positioning mounting brackets. Measure off the switch space (135mm) and rule positing lines. Measure off and rule guideline for remaining three host paces. Screw on six brackets, both sides front middle and rear, for each platform using timbre screws.

Carefully insert all host platforms and switch platform into the case.

Install L shaped mending plates on one side at the rear to act as a stop for each host platform.

Connect all PSU and LAN switch to power board.

Connect all hosts, workstation and power line extender to LAN switch with cat 6 patch cables.

The resulting cabinet is open at the front and rear to maximise airflow but exposes the electronics. If desired a plastic mesh screen can be cut to fit, with cutouts for access to power switch and external connectors and mounted so it can easily be removed.

Part 3 - Software Installation

Conventions

HOSTNAME, USERNAME AND GROUPNAME CONVENTIONS

Naming conventions are used for convenience. Hosts that perform the same function are given a common name appended by a dash and a serial number. The names used for a machine:

- used by an artists/animators is referred to as workstation,
- that provides file sharing and control is referred to as master,
- that is dedicated to performing render tasks is referred to as render, and
- that stores archived files and animation resources is referred to as fileserver.

Each workstation has normal logins independently of the render-farm. Each machine in the render-farm has an administrators account and a user account. The administrator account is used to manage the machine and the user account is used to perform render-farm functions. The administrators name and password are chosen by the administrator. The user name and password are as follows:

- master host: username = master, password = master
- render host: username = tracer, password = tracer
- fileserver host: username = filer, password = filer

All machines, including the workstations must be in the workgroup “RENDERFARM”

All actions that need to be taken to perform task, whether entering a command at the command prompt or entering settings via a GUI are numbered in the order they must be performed.

Commands that are to be entered verbatim are printed in mono type so spaces can be discerned. The command prompt (either $ for user login or # for root login) is included before a verbatum command. When multiple machines use the same installation, some commands include a hostname with a numerical suffix. Rather than repeat all hostnames an ‘x’ is used to indicate a number sequence, e.g. render-x stands for render-1, render-2 etc and master-x stands for master-1 etc. Some commands apply to both render login and master login. Rather than repeat the command ‘master or render’ is used. Enter only the host login that is relevant at the time.

When it is more convenient to place comments on the same line as a command, a ‘#’ is used in case the comment is inadvertently copied along with the command.

Contingencies

In addition to the main render farm hosts, disused laptops may be put back into service as platforms for familiarising master host and render host software installation steps and subsequently used as a backup master host or for trialing performance optimisation options. A laptop may also be useful to test including an external Windows machine into the workgroup.

Helpful Hints

Using the command line requires letter perfect accuracy. There are many shortcuts however using the up and down arrows to recall previously used commands is extremely useful. Copying this instruction to the host desktop and cutting and pasting commands and settings is often possible.

Gateway/modem Configuration

Modem/Router Info

Model: Telstra Smartmodem

Network Name: E86569

Host name: mymodem

Local IP Address: 192.168.0.1

Net mask: 255.255.255.0

DHCP Address: 192.168.0.0

DHCP Start: 192.168.0.2

DHCP End: 192.168.0.254

DLNA Sever enabled.

RESERVE STATIC IP ADDRESSES

Hostname	MAC Address (Ethernet)	Static IP Address
workstation-1	tba	192.186.0.200
master-1	tba	192.186.0.201
render-1	tba	192.168.0.202
render-2	tba	192.186.0.203
render-3	tba	192.168.0.204
render-4	tba	192.168.0.205
render-5	tba	192.168.0.206
render-6	tba	192.168.0.207
fileserver-1	tba	192.168.0.250

1 - Access Gateway/modem settings from browser at http://192.168.0.1/home.lp

2 - Add new static leases under Advanced tab - Local Network dialog Software Installation and Customisation Guide

Workstation Configuration

The following example is for a MAC with OSX**.**

Configure Static IP Address

1 - Open System Preferences -> Network

Highlight Ethernet (from connection options)
Select Configure IPv4: Manually
Enter IP Address: 192.168.0.200
Enter Subnet Mask: 255.255.255.0
Enter Router: 192.168.0.1
Close

Obtain Ethernet Mac Address

The MAC address is used in several communication processes however it is needed later in the installation to record a static IP lease on the Gateway/Router.

2 - Open System Preferences -> Network

Highlight Ethernet
Click Advanced
Click Hardware
Close

Add VNC User Group

3 - Add user group for VNC access - Open System Preferences -> Users & Groups.

Click padlock icon open
Enter ‘admin password’
Click + (to add new group in dropdown dialog)
Select New Account: Group
Enter Full Name: render
Click Create Group (check group appears under Group arrow)
Click padlock icon closed
Close

Enable VNC Client/Viewer

4 - Open System Preferences -> Sharing.

Check Screen Sharing
Click Computer Settings
Check VNC viewers may control screen with password:
Enter password ‘render’
Check Allow access for: Only these users:
Click + (to add new group from dropdown dialog)
Select render
Close

Set Hostname

5 - `workstation-1`

Set Group Name

6 - RENDERFARM

Windows 7 Laptop Configuration

A Windows laptop may be useful as a contingency and testing.

CONFIGURE HOSTNAME, STATIC IP ADDRESS AND INSTALL TIGHTVNC VNC SERVER

1 - Configure contingency laptop as Hostname laptop1

2 - Configure allocated static IP address

3 - Download the TightVNC version compatible with host OS

4 - Install TightVNC on master-01

5 - Test Hostname, IP address and VNC server

OBTAIN MAC ADDRESS

The MAC address is used in several communication processes however it is needed later in the installation to record a static IP lease on the Gateway/Router.

1 - Obtain MAC address

Linux Master and Render Host Configuration

DOWNLOAD ISO IMAGE

Ubuntu has several versions ranging from a server version a, studio version for animators and artists (including Blender), standard Gnome desktop version to a light weight version Lubuntu. Ubuntu Mate 20.04 will be used as the standard OS.

1 - Download Ubuntu Mate 20.04 iso file from the Ubuntu to an available Windows or Mac host.

DOWNLOAD BELENA ETCHER MEDIA WRITER

The Etcher media writer can used to create a USB boot media from a Windows or Mac platform. The media can be used to boot a Windows or Mac host with Linux and/or install Linux over the current OS. Other media writers may also work.

1 - Download Etcher media writer to host and install.

PREPARE BOOT USB

1 - Insert a USB Thumb-drive labeled as ‘Ubuntu “version” 20.4 Boot USB’

2 - Run the media writer and select the Ubuntu iso file to write to USB.

Note1: **Take care to select the USB** and not the host hard drive as the target.

3 - Wait for image to be written to USB and eject

Note: The USB Thumb-drive will be formatted for Linux ext4 file system and will not be readable on Mac or Windows however Linus can read FAT32 and NTFS formats.

The USB can now be used to Live boot any PC with Linux and then optionally install it. Host Configuration

INSTALL UBUNTU MATE 20.04

Install Ubuntu on hosts SSD drive:

Preliminary to installing the OS the host BIOS has to be configured to use a USB device as the default boot media. This is achieved by inserting a boot USB and restarting the machine while holding down the delete key or other function key used for the purpose. When the bios editor appears select Boot devices tab, Boot settings, Hard drives and promote the USB device to the 1st boot position by using the + key. Save and exit (F10) and wait for boot process.

1 - Connect machine to LAN switch and ensure the Gateway is accessible

2 - Insert the boot USB, power on the host and wait for Ubuntu to load.

3 - Select English as the language then double click the Install Lubuntu icon.

4 - Wait for 'Welcome' page - click Next at bottom of screen to enter preferences.

select install Ubuntu 20.04 Mate
select minimal installation, download updates, install additional graphics
select erase disk
set time zone - click on Sydney
enter user information
Your full name: administrators name
Name of computer: `render-x, master-x, fileserver-x`
User name: administrators login name
Password: administrators password
Note. Don't select the Login automatically button for admin user
Finish - continue. (will take several minutes)

5 - Restart and remove USB boot media when prompted

6 - Wait for boot from SSD to complete and login

7 - `$ sudo apt update` # essential to update packages before proceeding

8 - Uncheck Open welcome screen

Post install commands useful to check devices, modules and drivers (ls commands):

```
$ lspci          # shows pci, usb, SATA, SMB, IDE, Audio, Ethernet, VGA etc

$ lsblk          # shows storage mounts and RAM

$ lscpu          # shows details of cpu
```

Commands useful to check system status and performance and individual processes:

```
$ sudo systemctl status  # shows state of system

$ df -h          # shows disk space rounded
```

```
$ free –m          # shows installed memory and usage in Mbytes
$ uptime           # shows number users and load average for last 1, 5, 15 min
$ top              # shows users and their resource consumption in real time, top user indicated
$ sudo ps –a                                  # list current active processes
$ sudo ps –aux                                # list recent history of processes
$ cat /proc/<pid>/status                      # shows detailed information
$ ps –ef | grep <pid> | grep –v “grep”  # shows details of a process
$ kill –9 <pid>   # *** forces termination of process with potential data loss
$ systemctl –type=service                     # lists all loaded services
$ systemctl status <logfile>                  # shows status of a log file in /var/log/
```

CHECK UBUNTU 20.04 LTS IS INSTALLED

A Long Term Support (LTS) version is needed for stability.

1 - `$ lsb_release –a`

CHECK DETAILS OF GRAPHICS CARD AND DRIVER VERSION

1 - `$ sudo lshw –class display`

2 - Menu -> Control Center -> Hardware -> Additional Drivers

INSTALL NET-TOOLS

Net-tools are depreciated on Lubuntu in favour of iproute2 but they are often used in tutorials needed to manage IP addresses and other communications.

1 - Ensure there is an active Internet connection

2 - `$ sudo apt install net–tools`

3 - Wait for package download and installation

CHECK HOSTNAME

Some operations of the render farm need to identify each host by a unique hostname. The hostname identifies a host operating system, whereas a MAC address identifies a particular NIC (different for Ethernet and WIFI). Enter the following.

1 - `$ hostname`

If Hostname is incorrect, set it by:

2 - $ `sudo hostnamectl set-hostname render-x` **or** `tracer-x`

3 - $ `ifconfig -a` # confirm new Hostname

OBTAIN ETHERNET MAC ADDRESS

The MAC address is needed register a static IP address with the Gateway/router. To display only the ethernet MAC address.

1 - $ `ifconfig | grep ether`

SET STATIC IP LAN ADDRESS AND GATEWAY ADDRESS

Preliminary to setting static IP addresses a range of available IP addresses from the top half of the subnet space has to be identified and allocated to each host. On a small network it is usually safe to allocate static address beginning at 192.168.0.200.

To set a static IP address:

1 - $ `ifconfig -a` # display and note current IP and ethernet device name

2 - Menu -> Preferences -> Advanced Network Configuration

3 - Under Ethernet, highlight active service eg. Wired Connection 1
and click cog icon at bottom of dialog box to edit settings

4 - Ensure the ethernet device name is as noted and click IPv4 Settings

Change Method to Manual
Click Add Address
Enter Address e.g. 192.168.0.20x (as allocated on Gateway)
Enter Subnet mask e.g. 255.255.255.0
Enter Gateway e.g.192.168.0.1
Enter DNS Servers e.g. 192.168.0.1
Click Save

5 - Reboot

6 - $ `ifconfig -a` **or** $ `ip addr show` # display and check new settings

7 - $ `ping 192.168.0.1` # check connection with Gateway/modem

INSTALL LIGHTDM

LightDM is a display manager that works with the X11 authentication required for x11 vnc remote access. A gdm3 or sddm display manager will not work. If prompted, select lightdm.

1 - $ `sudo apt-get install lightdm`

INSTALL VNC SERVER

VNC is a remote desktop connection. It will be used to access to hosts from a central point and reduce the need for individual video monitors on all machines. Screen locking must be disabled to prevent interference with vnc communication.

1 - Deactivate screen locking

Menu -> Control Center -> Power Management -> OnAC Power
Actions..never, Display…never

Menu -> Control Center -> Screensaver
uncheck Activate screensaver and Lock screen

2 - `$ sudo apt update` # only necessary if update was not done during installation

3 - $ `sudo apt install x11vnc`

4 - $ `sudo nano /lib/systemd/system/x11vnc.service`

```
[Unit]
Description=x11vnc service
After=display-manager.service network.target syslog.target

[Service]
Type=simple
ExecStart=/usr/bin/x11vnc -forever -display :0 -auth guess -passwd RENDERFARM
-geometry 1024x768
ExecStop=/usr/bin/killall x11vnc
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Save with ctl x, y (save and exit)

5 - $ `systemctl daemon-reload`

6 - $ `systemctl enable x11vnc.service`

7- $ `systemctl start x11vnc.service`

8 - $ `systemctl status x11vnc.service`

9 - $ `reboot`

10 - $ `systemctl status x11vnc.service` # check vnc service started at boot

INSTALL ETHTOOL

Optimal performance of the render farm requires all machines transmit/receive data at the maximum possible. Ethtool is a useful monitoring tool that provides details of ethernet transmissions.

1 - $ `sudo apt-get install ethtool`

CHECK LAN COMMUNICATION SPEED

If all machines in the render farm are connected via a Gigabyte switch, each machine should be operating at the maximum 1000 Mb/s in full duplex mode.

1 - $ `ifconfig` # obtain ethernet device name e.g. enp2s0 or eth0

2 - $ `dmesg | grep <ethernet device name>` # list status and speed

3 - $ `ethtool <ethernet device name>` # lists details of ethernet connection

INSTALL PERFORMANCE MONITORING TOOLS

Linux has built-in system calls that can be used by performance monitoring tools.
RAM and cache are the main targets and some tools are installed by default including Top. Other useful tools need to be installed:

1 - $ `sudo apt install sysstat` # includes iostat, vmstat, pidstst, mpstat and sar

Note: Sysstat is available but the repository may not be configured in Software and Upgrades. Check Preferences -> Software Upgrades and ensure all 4 options are checked.

2 - $ `sudo apt-get install htop`

3 - $ `sudo apt-get install nmon`

4 - $ `sudo apt-get install dstat`

For hosts with NVIDIA GPU

5 - $ `sudo apt install nvidia-utils-460`

6 - $ `sudo ubuntu-drivers devices` # list will show recommeded driver

7 - $ `sudo ubuntu-drivers install`

8 - $ `sudo reboot`

9 - $ `nvidia-smi`

10 - $ `sudo apt install nvtop`

COMPILE AND INSTALL PRECISION MONITORING TOOLS

Monitoring tools that access data more frequently and with high precision need to be compiled into the kernel by an administrator. **Warning**. CoreFreq may not be compatible with all hardware. Given the same version of Ubuntu it may install into the kernel for some hardware but hang on others during the insmod operation. If this happens restore the grub file settings then a manual reboot will be necessary and the benefits of the tool foregone.

CoreFreq

1 - Disable NMI watchdog by editing grub file

```
$ sudo nano /etc/default/grub

....
GRUB_CMDLINE_LINUX="nmi_watchdog=0"

ctl x, y     # save file

$ sudo update-grub

$ reboot   # reboot to admin
```

2 - Install CoreFreq

```
$ sudo apt-get install git dkms build-essential libc6-dev libpthread-stubs0-dev

$ sudo git clone https://github.com/cyring/CoreFreq.git

$ cd CoreFreq

 $ sudo make
```

3 - Install the kernel module

```
$ sudo insmod corefreqk.ko

$ lsmod | grep corefreq  # reports if CoreFreq is installed

$ sudo dmesg | grep CoreFreq  # reports if recognised by the processor
```

When needed, start the module,

```
$ sudo ./corefreqd -i &
```

and then the client

```
$ ./corefreq-cli
```

CREATE NON-ADMIN USER

A non-admin user is needed for render farm operations and an auto-login and the user needs to be added to the RENDERFARM workgroup.

1 - $ `sudo adduser master` **or** `tracer`

full name: `master` **or** `tracer`
password: `master` **or** `tracer` # typing will not show, but then enter

Note. Leave other user info as null

2 - $ `sudo groupadd RENDERFARM` # add a user

3 - $ `sudo usermod –a –G RENDERFARM master` **or** `tracer` #add user to group

4 - $ `users` # lists all users - also use cat /etc/passwd for system users

5 - $ `groups master` **or** `tracer` # lists groups user is in - also use $ id <username>

ENABLE AUTO-LOGIN FOR NON-ADMIN USER

Auto-login is needed for render farm remote access and automation. The auto login is executed for the user specified in lightdm.conf.

1 - $ `cat /etc/X11/default–display–manager` # check that display manager is lightdm

2 - $ `sudo nano /etc/lightdm/lightdm.conf` #open for editing - may be new file

```
[Seat:*]
autologin–user=master or tracer
autologin–user–timeout=0
```

`ctl x, y` # save and exit

3 - reboot and ensure automatic login to master of render user

4 - Deactivate screen locking

Menu -> Control Center -> Power Management -> OnAC Power
Actions..never, Display…never

Menu -> Control Center -> Screensaver
uncheck Activate screensaver and Lock screen

5 - From another machine with a vnc client, check vnc is operating for render

Note. To access administrator account, logout from render and login as admin.

CREATE HOSTS FILE

On the Internet IP addresses and host names are found using a Domain Name Service (DNS). As there is no DNS on a LAN, a Hosts file is an alternative means of retrieving IP addresses and host names.

Switch to admin user.

1 - $ `cd /etc`

2 - $ `sudo nano hosts`

3 - Add entries for all machines on render farm

```
192.168.0.200     workstation-1
192.168.0.201     master-1
192.168.0.202     tracer-1
192.168.0.203     tracer-2
192.168.0.204     tracer-3
192.168.0.205     tracer-4
192.168.0.206     tracer-5
192.168.0.207     tracer-6
192.168.0.250     fileserver-1
```

ctl x

4 - $ `cat hosts`

5 - $ `reboot`

Master Host Only

INSTALL SAMBA

Samba is an Open Source implementation of the Server Message Blocks file-sharing protocol.

1 - $ `sudo apt install samba`

2 - $ `sudo systemctl status smbd`

Create Shared Samba Configuration and Shared Directories

The Samba configuration file is used to define shared directories for network operation, including drop-box directory, render input directories (render), render output directories (render_images), a video output directory and a pick-up directory. Separate render, images and video directories are created for production and test. The render output directories are sub-directories of the input directories so they can be located using a relative path. A naming convention is to ensure uniqueness within the user.

Create directories to be shared or used for workflow

Login as master user (share directories to be created under master home directory)

1 - $ `cd /home/master`

2 - $ `mkdir prod`

3 - $ `mkdir prod/prod_render`

4 - $ `mkdir prod/prod_render/prod_images`

5 - $ `mkdir prod/prod_video`

6 - $ `mkdir test`

7 - $ `mkdir test/test_render`

8 - $ `mkdir test/test_render/test_images`

9 - $ `mkdir test/test_video`

10 - $ `mkdir render_bin`

11 - $ `mkdir staging`

12 - $ `mkdir compositing`

13 - $ `mkdir compositing/presets`

14 - $ `mkdir drop_box`

15 - $ `mkdir pickup_box`

15 - $ `mkdir job_tickets`

16 - $ `mkdir scheduling`

17 - $ `mkdir for_archive`

18 - Delete unused standard install directories i.e. Videos, Documents, Templates…

Create Samba Configuration File

Login as master admin

1 - `cd /etc/samba`

2 - `$ sudo mv smb.conf smb.conf.bak` # wont use original conf - take a backup

3 - `$ sudo nano smb.conf` # create a conf file from scratch

```
[global]
server string = master-1 host
server role = standalone server
wins support = yes
name resolve order = host wins bcast
workgroup = RENDERFARM
security = user
map to guest = Bad User
usershare allow guests = yes
hosts allow = 192.168.0.0/16
hosts deny = 0.0.0.0/0

#Share for all render hosts to read production animation file
[prod_render]
path =  /home/master/prod/prod_render
force user = smbuser
force group = smbgroup
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
public = yes
writable = yes

#Share for all render hosts to read test animation file
[test_render]
path =  /home/master/test/test_render
force user = smbuser
force group = smbgroup
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
public = yes
writable = yes

# Share for all render hosts to write production rendered images
[prod_render_images]
path = /home/master/prod/prod_render/prod_images
force user = smbuser
force group = smbgroup
```

```
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
public = yes
writeable = yes
#read only = No
#guest ok = Yes
#write list = render-1 render-2 render-3 render-4 render-5

# Share for all render hosts to write test rendered images
[test_render_images]
path = /home/master/test/test_render/test_images
force user = smbuser
force group = smbgroup
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
public = yes
writeable = yes

# Share for compositing process to write production video
[prod_video]
path = /home/master/prod/prod_video
force user = smbuser
force group = smbgroup
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
public = yes
writeable = yes

# Share for compositing process to write test video
[test_video]
path = /home/master/test/test_video
force user = smbuser
force group = smbgroup
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
public = yes
writeable = yes

# Share for staging inputs
[drop_box]
path = /home/master/drop_box
force user = smbuser
force group = smbgroup
```

```
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
public = yes
writeable = yes

# Share for staging outputs
[pickup_box]
path = /home/master/pickup_box
force user = smbuser
force group = smbgroup
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
public = yes
writeable = yes

ctl x
```

4 - $ `testparm` # test conf parameters are ok

5 - $ `sudo systemctl restart smbd`

Create user and group to apply permissions

1 - $ `sudo groupadd –system smbgroup`

2 - $ `sudo useradd –system –no–create–home –group smbgroup –s /bin/false smbuser`

3 - $ `cat /etc/group` **4** - $ `cat /etc/passwd` # check created ok

Change shared directory permissions

Login as admin, invoke superuser (sudo -i) and then change directory to /home/master

1 - # `chown –R smbuser:smbgroup prod`

2 - # `chmod –R g+w prod`

3 - # `ls –l` **4** - # `ls –l prod` # check permissions

5 - # `chown –R smbuser:smbgroup test`

6 - # `chmod –R g+w test`

7 - # `ls –l` **8** - # `ls –l test` # check permissions

9 - # `chown –R smbuser:smbgroup drop_box`

10 - # `chmod –R g+w drop_box`

11 - # `ls –l` **12** - # `ls –l drop_box` # check permissions

13 - # `chown –R smbuser:smbgroup pickup_box`

14 - # `chmod –R g+w pickup_box`

15 - # `ls –l` **16** - # `ls –l pick_up box` # check permissions

17 - # `usermod –a –G smbgroup master` # give master user permits to delete files

18 - # `reboot`

Useful SAMBA Commands

`$ findsmb` # lists IP address, Netbios name and groupname

`$ nmblookup __SAMBA__` # lists IP address of all SAMBA servers on network

`$ nmblookup –S __SAMBA__` #lists all SMB servers

`$ nmblookup –S RENDERFARM` #lists all server members of a workgroup

Check SSH Client is Installed

The master host will request a connection to each render machine that will in turn respond to action the request. In ssh terminology the master host is an ssh client and the render hosts are servers.

SSH client should be installed with Ubuntu Mate. Check the installation.

Note: the command to install an ssh client from the admin logon is

`$ sudo apt install openssh–client`

1 - `$ ssh –V`

Generate an Encryption Key Pair

SSH has an option for a client to connect to a server without a login password by using encryption private/ public key pair. The key pair is generated on the client and copied to a special ssh file on each server. Login to master home directory.

1 - `$ ssh–keygen`

….Generating a public/private rsa key pair.

Press enter to accept default file to save the key

….Created directory ‘/home/master/.ssh/id_rsa’

Press enter to bypass passphrase (twice)

….The key fingerprint is…..

2 - $ `cd ssh`

3 - $ `ls –lh`

4 - Copy the file id_rsa.pub to a USB media directory ‘SSH’ for transfer to all render hosts.

5 - On USB media, make a copy of id_rsa.pub and rename it to authorized_ keys

Install VNC Client and SSH Client

X11VNC server is installed but does not have a viewer so alternatives are TigerVNC viewer and vinagre remote desktop application. Vinagre supports several connection protocols. The clients will appear in the Internet tab on the desktop menu. Login as admin.

1 - $ `sudo apt-get install –y tigervnc-viewer`

2 - $ `sudo apt install vinagre`

Install LibraOffice and Pinta

A spreadsheet, database and image editor may be useful for managing operations.

1 - $ `sudo apt install install libreoffice`

1 - $ `sudo apt-get install pinta`

RENDER HOSTS ONLY

Install CIFS-Utils

The CIFS utility is installed to support mounting s shared directory as part of the local file system.

1 - $ `sudo apt-get install cifs-utils`

Mount the Render Input Shared Directories on the Local File System at Boot-up

A permanent mount at boot-up requires a local directory to mount to, root username and password and an entry in the fstab file. For security, the root username and password can be stored in a hidden file (begins with .) The format of an fstab entry is:

```
//<server IP>/<share name>   /<path to local directory>
cifs   /<credentials>,<smb ver>,<format options>,<mode options> 0 0
```

Login as render user and create a local directories for mounting.

1 - $ `mkdir prod_render`

2 - $ `mkdir test_render`

3 - $ `mkdir prod_render/prod_images`

4 - $ `mkdir test_render/test_images`

5 - $ `mkdir staging`

6 - Delete unused directories

Switch user to admin

Edit fstab file to create cifs mounts

Note1: The username and password used are that of the administrator and could be easily discovered by non-admin users. They can be hidden in a password file but are included in the cifs entries as it is the most reliable implementation.

Note2: All cifs entries are on one line each but shown separated below for clarity.

1 - $ `cd /etc`

2 - $ `sudo nano fstab`

```
# Shared directories

//192.168.0.201/prod_render  /home/tracer/prod_render
cifs  username=<admin>, password=<password>,
vers=3.0,iocharset=utf8,file_mode=0777,dir_mode=0777 0 0


//192.168.0.201/test_render   /home/tracer/test_render
```

```
cifs  username=<admin>,password=<password>,
vers=3.0,iocharset=utf8,file_mode=0777,dir_mode=0777 0 0


//192.168.0.201/prod_render_images    /home/tracer/prod_render/prod_images
cifs   username=<admin>,password=<password>,
vers=3.0,iocharset=utf8,file_mode=0777,dir_mode=0777 0 0

//192.168.0.201/test_render_images    /home/tracer/test_render/test_images
cifs   username=<admin>,password=<password>
vers=3.0,iocharset=utf8,file_mode=0777,dir_mode=0777 0 0
```

Test the mounts

1 - $ `sudo mount –a`

If any problems type:

2 - $ `dmesg`

Alternative hidden credentials file

Login as admin and create credentials file

1 - $ `sudo nano .share_creds`

```
username=<admin name>
password=<admin password>
```

ctl x

In all cifs entries, replace “`username=<admin>,Password=<password>`” with “`credentials= /home/<admin>/.share_creds`”

Install Samba Client

The smbclient is installed to support command line access to shared directories.

1 - `sudo apt install smbclient`

2 - $ `reboot`

How to Access Shared Directories from From the Nautilus File Manger

Click on Browse Network to display shared directories.

How to Access Shared Directories From the Command Line

```
$ smbclient //<IP address>/<sharename> –U <username>%<password>

 smb: \> # smb prompt
 smb: \>help
 smb: \>exit
```

Note: Use the IP address instead of the hostname (as shown in tutorials) as the is an apparent problem with resolving the hostname.

How to Temporarily Mount a Shared Directory on the Local File System

Create a local directory to mount to:

1 - `$ sudo mkdir /<full path>/<tmpshare>`

2 - `$ sudo mount –t cifs //<shareserver IP>/<path to share>  /<path to tmpshare> –o username=user`

or

```
$ mount –t cifs –o username=none,password=none //<servername>/<path to share> /
<path tmpshare>
```

Alternative with uid and gid

The uid and gid are used to determine what resources a user can access so there may be advantages to including the uid and gid of render in the mount. Find uid gid with

```
$ whoami  # gives username

$ id <username>
```

For temporary mount

```
$ mount –t  cifs –o username=<username>,password=<password>,uid=xx,gid=xx,
rw,nounix,iocharset=utf8,file_mode=0777,dir_mode=0777
 //192.168.1.201/<path to share>  /<path to temp>
```

For permanent fstab entry

```
//192.168.1.120/<path to share>   /<path to temp>
 cifs    credentials=/<path to admin>/.smbcredentials,uid=33,gid=33,
rw,nounix,iocharset=utf8,file_mode=0777,dir_mode=0777
```

Test Network File Shares

Test the following configurations and services:

- all machines can auto boot to user

- the master host mounts all shared directories

- any workstation can access the drop box and pickup box shared directory

- all render hosts can access the blend and images shared directories

- master host and render host can be accessed via vnc

Install SSH Server

Each render host will be a ssh server to the master host ssh client.

Login as admin.

1 - $ `sudo apt-get install openssh-server`

2 - $ `sudo systemctl status ssh.service` # check ssh is running

Login as render

3 - $ `ssh-keygen`

….Generating a public/private rsa key pair.

Press enter to accept default file to save the key

….Created directory '/home/master/.ssh/id_rsa'

Press enter to bypass passphrase (twice)

….The key fingerprint is…..

4 - Insert USB media with the authorized_keys file and copy it to the .ssh directory.

Test SSH

1 - Reboot master host and tracer hosts

2 - Login to master-1 as master user

3 - $ `ssh tracer@<`*`tracer ip`*`>`

On the first connection a warning message appears. Type 'yes', then a command prompt at tracer@render-x should appear. Further connections will connect immediately.

Note: Using the IP address is more reliable than hostname which may not resolve.

Blender Application Installation

INSTALL BLENDER 2.83.4

Blender can be installed from a zip file to install a specific version that may not be available from a repository. The standard directory for a locally installed application so it is accessible by any user is /usr/local/bin.

Manually Install Blender on Master Host and All Render Hosts

1 - Download the Blender zip file for Linux (e.g blender-2.83.4-linux.tar.xz)

2 - Copy the zipped image file to a USB media formatted for FAT32

3 - Login to the host as admin and then insert the USB media

4 - Copy the tar file to /home and rename to blender.tar.xz. (use File Manager)

5 - Move the tar file to /usr/local/bin (use command line - need admin permits)

```
$ sudo mv blender.tar.xz /usr/local/bin
```

6 - Change to /usr/local/bin directory

```
$ cd /usr/local/bin
```

7 - Extract all files to a blender folder (use command line - need admin permits)

```
$ sudo tar –xf blender.tar.xz
```

8 - Check blender directory exists and rename to a shorter name

```
$ ls          # display current name

$ sudo mv blender–x.xx.x–linux64 blender283  # i.e. shorter but still has version

$ ls         # check new name
```

9 Add blender to global PATH environment variable (recommended method)

```
$ cd /etc/profile.d              # any .sh script found in this directory runs at login

$ sudo nano blender_path.sh      # create a script file to append blender path

export PATH=$PATH:/usr/local/bin/blender*xxx*

ctl x, y
```

10 - `$ sudo reboot` # to non-admin user terminal

11 - `$ echo $PATH` # check to see blender has been appended to PATH

12 - For each individual host, open Blender and set the Edit-> Preferences -> System -> Cycles Render Device to support the GPU installed GPU, i.e. 'None', 'CUDA', 'Optix', 'OpenCL'. (Very important)

TEST BLENDER INSTALLATION AND NETWORK OPERATION

In preparation for initial tests:

- configure a workstation for VNC remote operation and access to RENDERFARM group shared directories (see above instructions)

- prepare a simple animation with 5 frames. Name it '5copy.blend' and copy the file to a USB media labelled 'Test Files'. Ensure there are no leading spaces in the file name.

- open a test editor and write a shell script as follows:

```
#!/bin/bash

# gorender.sh – Initiates background render session using shared directories

while getopts "i:o:" flag
do
    case "$flag" in
        i) infile="$OPTARG";;
        o) outfile="$OPTARG";;
    esac
done
echo "Input file: $infile";
echo "Output file: $outfile";

cd ~
pwd

exec blender –b ~/test_render/$infile –o //test_images/$outfile –a
```

- save script as gorender.sh

- copy the shell script to a USB media under a directory 'Render_scripts. Labeled the media 'Render Farm Shell Scripts'

Create Directories for Local Blender Test on Master Host and All Render Host

Logon as user.

1 - `$ mkdir blender_in`

2 - `$ mkdir blender_in/blender_out`

3 - `$ mkdir render_test_files`

Copy Test File to Host

1 - Copy the test file to ~/render_test.

Test Blender Runs, Loads a File and Renders Images

1 - `$ cd ~`

2 - `$ blender` #Blender should load with splash screen

3 - $ File ->Open ->blender_in ->5test.blend. #Blender should load 5test.blend

4 - Set Properties

Render Properties

Render Engine > Cycles
Device > CPU

Output Properties

Output location: /home/tracer/blender_in/blender_out/test_images

File Format > JPEG

5 - Initiate render

Render ->Render Animation (Blender should render 5 frames to blender_out)

6 - Close Blender

Clean Up

1 - Delete test images in blender_out (directory must be empty for next test)

Test Blender Renders in Background Mode

1 - `$ blender -b ~/blender_in/5test.blend -o //blender_out/test_images -a`

Clean Up

1 - Delete test images in blender_out

2 - Copy /blender_in/5test.blend to /blender_test_files

Create Directories for Network Render on All Render Hosts

1 - `$ mkdir render_bin`

Install Render Shell Script on all Render Hosts

1 - Copy gorender.sh to /render_bin

2 - `$ chmod +x /render_bin/gorender.sh` # make shell script executable

Copy Test File to Master Shared Directory

1 - Login the master host as master user

2 - Copy the test file '5test.blend' to the ~/test/test_render directory.

Host Startup Sequence

Ensure master host and all render hosts are shutdown as Samba and CIFS mounts won't function properly if connected in the wrong sequence. Using the master host and two render hosts are sufficient for the test.

1 - Start the master host and wait until fully booted

2 - Start render hosts and wait until fully booted.

Initiate Render Process

1 - Open VNC remote sessions to bother render hosts. (This may be done on the workstation figured for VNC access.)

Perform the following commands on both render hosts via VNC

2 - Open a terminal session

3 - `$ cd render_bin`

4 - `$ sh gorender.sh –i 5test.blend –o test_images####`

5 - Observe the terminal output of Blender - each host should be rendering a frame

6.- Wait until all 5 frames have been rendered and 'Blender quit' is output on terminals.

7 - Using the file manager, open the test_blender/test_images directory on the master

Five rendered jpeg images should accumulate as the render proceeds.

At this point there should be a functioning render farm.

Clean Up

1 - Delete all images files from the master test_images directory.

2 Close VNC sessions **3** Shutdown hosts

Render Process Automation

Shell scripts and Python scripts need to be coded for each automation model and and installed on the hosts.

AUTOMATION SCRIPT INSTALLATION SCHEDULE

Scripts	Installed Machines	Directory
Preprosessing		
fa.sh	master host	home/master/render_bin
set_fa.py	master host	home/master/render_bin
hiq_rp.sh	master host	home/master/render_bin
set_hiq_rp.py	master host	home/master/render_bin
frt_rp.sh	master host	home/master/render_bin
set_frt_rp.py	master host	home/master/render_bin
gpu_rp.sh	master host	home/master/render_bin
set_gpu_rp.py	master host	home/master/render_bin
cpu_rp.sh	master host	home/master/render_bin
set_cpu_rp.py	master host	home/master/render_bin
Model 1 Render		
prod_ren.sh	all render hosts	home/render/render_bin
test_ren.sh	all render hosts	home/render/render_bin
set_ren_fa.py	all render hosts	home/render/render_bin
Model 2 Render		
cpu_prod_ren.sh	all render hosts	home/render/render_bin
cpu_test_ren.sh	all render hosts	home/render/render_bin
alloc_cpu_rt.py	all render hosts	home/render/render_bin

SCRIPT CODE

fa.sh

```
#!/bin/bash

# fa.sh – Runs Python script to set critical Overwite and Placeholder properties
render properties

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file"
   echo -e "\t-i Name of animation file including the file extension"
   exit 1 # Exit script after printing help
}


while getopts "i:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo "\tInput animation file: $infile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/staging/$infile -P ~/render_bin/set_fa.py
```

set_fa.py

```
#####################################################################
#  set_fa.py  - set sets critical Overwrite and Placeholder render properties
#    opens .blend file
#    sets Overwrite to False and Placeholder to True render properties
#    saves .blend file with settings
#####################################################################

import bpy

# get the name of this .blend file

infile=bpy.path.basename(bpy.context.blend_data.filepath)

# check if images are packed in

# set global output properties for all scenes in infile

def set_fa() :

    for scene in bpy.data.scenes:
        scene.render.use_overwrite = False
        scene.render.use_placeholder = True

    return

set_fa()

# save infile with essential settings


bpy.ops.wm.save_as_mainfile(filepath="/home/master/staging/" + infile)
```

hiq_rp.sh

```
#!/bin/bash

# hiq_rp.sh - Runs Python script to set high image quality render properties

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file"
   echo "\t-i Name of animation file including the file extension"
   exit 1 # Exit script after printing help
}


while getopts "i:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo -e "\tInput animation file: $infile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/staging/$infile -P ~/render_bin/set_hiq_rp.py
```

set_hiq_rp.py

```
#####################################################################
#  set_hiq_rp.py  - set high image quality render properties
#    opens .blend file
#    sets standard render properties
#    saves .blend file with settings
#####################################################################

import bpy

# get the name of this .blend file

infile=bpy.path.basename(bpy.context.blend_data.filepath)

# check if images are packed in

# set global output properties for all scenes in infile

def set_fa() :

    for scene in bpy.data.scenes:
        scene.render.use_overwrite = False
        scene.render.use_placeholder = True

    return

# declare standard production render property values and
# set global output properties for all scenes in infile

def set_hiq_rp() :

    res_x = 1920
    res_y = 1080
    percent = 100
    aspect_x = 1
    aspect_y = 1


    for scene in bpy.data.scenes:
        scene.render.resolution_x = res_x
        scene.render.resolution_y = res_y
        scene.render.resolution_percentage = percent
        scene.render.pixel_aspect_x = aspect_x
        scene.render.pixel_aspect_y = aspect_y

    return

set_fa()
set_hiq_rp()
```

```
# save infile with standard settings

outfile = "h_" + infile
bpy.ops.wm.save_as_mainfile(filepath="/home/master/staging/" + outfile)
```

frt_rp.sh

```
#!/bin/bash

# frt_rp.sh - Runs Python script to set fastest render time render properties

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file"
   echo "\t-i Name of animation file including the file extension"
   exit 1 # Exit script after printing help
}


while getopts "i:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo "\tInput animation file: $infile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/staging/$infile -P ~/render_bin/set_frt_rp.py
```

set_frt_rp.py

```
#####################################################################
#  set_frt_rp.py  - set fast render time render properties
#    opens .blend file
#    sets standard render properties
#    saves .blend file with settings
#
#####################################################################

import bpy

# get the name of this .blend file

infile=bpy.path.basename(bpy.context.blend_data.filepath)

# check if images are packed in

# set global output properties for all scenes in infile

def set_fa() :

    for scene in bpy.data.scenes:
        scene.render.use_overwrite = False
        scene.render.use_placeholder = True

    return

# declare standard production render property values and
# set global output properties for all scenes in infile

def set_frt_rp() :

    res_x = 1280
    res_y = 720
    percent = 90
    aspect_x = 1
    aspect_y = 1

    for scene in bpy.data.scenes:
        scene.render.resolution_x = res_x
        scene.render.resolution_y = res_y
        scene.render.resolution_percentage = percent
        scene.render.pixel_aspect_x = aspect_x
        scene.render.pixel_aspect_y = aspect_y

    return

set_fa()
set_frt_rp()
```

```
# save infile with standard settings

outfile = "f_" + infile
bpy.ops.wm.save_as_mainfile(filepath="/home/master/staging/" + outfile)
```

gpu_rp.sh

```
#!/bin/bash

# gpu_rp.sh - Runs Python script to set GPU render properties

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file"
   echo "\t-i Name of animation file including the file extension"
   exit 1 # Exit script after printing help
}


while getopts "i:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo "\tInput animation file: $infile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/staging/$infile -P ~/render_bin/set_gpu_rp.py
```

set_gpu_rp.py

```
#################################################################
#  set_frt_rp.py  - set fast render time render properties
#    opens .blend file
#    sets standard render properties
#    saves .blend file with settings
#
#################################################################

import bpy

# get the name of this .blend file

infile=bpy.path.basename(bpy.context.blend_data.filepath)

# check if images are packed in

# set global output properties for all scenes in infile

def set_fa() :

    for scene in bpy.data.scenes:
        scene.render.use_overwrite = False
        scene.render.use_placeholder = True

    return

# declare standard production render property values and
# set global output properties for all scenes in infile

def set_frt_rp() :

    res_x = 1280
    res_y = 720
    percent = 90
    aspect_x = 1
    aspect_y = 1

    for scene in bpy.data.scenes:
        scene.render.resolution_x = res_x
        scene.render.resolution_y = res_y
        scene.render.resolution_percentage = percent
        scene.render.pixel_aspect_x = aspect_x
        scene.render.pixel_aspect_y = aspect_y

    return

set_fa()
set_frt_rp()
```

```
# save infile with standard settings

outfile = "f_" + infile
bpy.ops.wm.save_as_mainfile(filepath="/home/master/staging/" + outfile)
```

cpu_rp.sh

```
#!/bin/bash

# cpu_rp.sh - Runs Python script to set CPU render properties

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file"
   echo "\t-i Name of animation file including the file extension"
   exit 1 # Exit script after printing help
}


while getopts "i:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo "\tInput animation file: $infile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/staging/$infile -P ~/render_bin/set_cpu_rp.py
```

set_cpu_rp.py

```
#####################################################################
#  set_ cpu_rp.py  - set cpu render properties
#    sets device settings for a host with CPU cores
#    writes out .blend files for cpu render
#####################################################################

import bpy

# get the name of the input file .blend file

infile=bpy.path.basename(bpy.context.blend_data.filepath)

# set global output properties for all scenes in infile

def set_fa() :

    for scene in bpy.data.scenes:
        scene.render.use_overwrite = False
        scene.render.use_placeholder = True

    return

# declare render device tile size settings
# set render device type to CPU
# set tile size
# set Threads mode as Auto-detect


def set_cpu_rp() :

    opt_cpu_tile_x = 32
    opt_cpu_tile_y = 32

    for scene in bpy.data.scenes:
        scene.cycles.device = 'CPU'
        scene.render.tile_x = opt_cpu_tile_x
        scene.render.tile_y = opt_cpu_tile_y
        scene.render.threads_mode = 'AUTO'

    return

set_fa()
set_cpu_rp()

# write file for cpu render

outfile = "c_" + infile
bpy.ops.wm.save_as_mainfile(filepath="/home/master/staging/" + outfile)
```

prod_ren.sh

```
#!/bin/bash

# prod_ren.sh - Initiates a production background render using users settings

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file -o Output-file"
   echo "\t-i Name of animation file including the file extension"
   echo "\t-o Name of output image sequence with optional # spec"
   exit 1 # Exit script after printing help
}


while getopts "i:o:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
       o) outfile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ] || [ -z "$outfile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo "\tInput animation file: $infile"
echo "\tOutput image sequence: $outfile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/prod_render/$infile \
             -o //prod_images/$outfile -F -x 1 \
             -P set_ren_fa.py \
             -a
```

test_ren.sh

```
#!/bin/bash

# test_ren.sh - Initiates a test background render using users settings

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file -o Output-file"
   echo "\t-i Name of animation file including the file extension"
   echo "\t-o Name of output image sequence with optional # spec"
   exit 1 # Exit script after printing help
}


while getopts "i:o:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
       o) outfile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ] || [ -z "$outfile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo "\tInput animation file: $infile"
echo "\tOutput image sequence: $outfile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/test_render/$infile \
             -o //test_images/$outfile -F PNG -x 1 \
             -P set_ren_fa.py \
             -a
```

set_ren_fa.py

```
#####################################################################
#  set_ren_fa.py  - set sets critical Overwrite and Placeholder
#  render properties on render host
#    opens .blend file
#    sets Overwrite to False and Placeholder to True render properties
#####################################################################

import bpy


# set global output properties for all scenes in infile

def set_fa() :

    for scene in bpy.data.scenes:
        scene.render.use_overwrite = False
        scene.render.use_placeholder = True

    return

set_fa()
```

cpu_prod_ren.sh

```
#!/bin/bash

# cpu_prod_ren.sh - Initiates a production background render using cpu cores

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file -o Output-file"
   echo "\t-i Name of animation file including the file extension"
   echo "\t-o Name of output image sequence with optional # spec"
   exit 1 # Exit script after printing help
}


while getopts "i:o:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
       o) outfile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ] || [ -z "$outfile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo "\tInput animation file: $infile"
echo "\tOutput image sequence: $outfile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/prod_render/$infile \
             -o //prod_images/$outfile -F PNG -x 1 \
             -P alloc_cpu_rt.py \
             -a
```

cpu_test_ren.sh

```
#!/bin/bash

# cpu_test_ren.sh - Initiates a test background render using cpu cores

helpFunction()
{
   echo ""
   echo "Usage: $0 -i Input-file -o Output-file"
   echo "\t-i Name of animation file including the file extension"
   echo "\t-o Name of output image sequence with optional # spec"
   exit 1 # Exit script after printing help
}


while getopts "i:o:" opt
do
   case "$opt" in
       i) infile="$OPTARG" ;;
       o) outfile="$OPTARG" ;;
   esac
done

# Print helpFunction in case parameters are empty
if [ -z "$infile" ] || [ -z "$outfile" ]
then
   echo "Missing arguement or dash";
   helpFunction
fi

# Begin script in case all parameters are correct
echo "Render job with..."
echo "\tInput animation file: $infile"
echo "\tOutput image sequence: $outfile"
echo "Initiate from..."
cd ~
pwd
start_time=$(date +"%c")
echo "Started at: $start_time"

# Blender background mode command line

exec blender -b ~/test_render/$infile \
             -o //test_images/$outfile -F PNG -x 1 \
             -P alloc_cpu_rt.py \
             -a
```

alloc_cpu_rt.py

```
#############################################################################
#  alloc_cpu_rt.py  - allocate CPU render threads
# set max cpu core usage based on leaving 1 thread to kernel use
# cpu_count() returns logical cores not physical cores i.e. total threads
#############################################################################

import bpy
from multiprocessing import cpu_count

def set_fa() :

    for scene in bpy.data.scenes:
        scene.render.use_overwrite = False
        scene.render.use_placeholder = True

    return

def alloc_cpu_rt() :

    available_threads = cpu_count()
    cpu_render_threads = max(1, (available_threads - 1))

    for scene in bpy.data.scenes:
        scene.render.threads_mode = 'FIXED'
        scene.render.threads = cpu_render_threads

    return

set_fa()
alloc_cpu_rt()
```

Set Permissions

On all hosts, set all schell scripts in render_bin to executable

1 - $ `chmod +x <scriptname>`

Part 4 - Operations

Pro forma

JOB TICKET

A simple job ticket is a useful means of passing render information from animator to render farm and provides a reference for analysis of results.

```
                                JOB TICKET

Job:

Job description:

Input filename:

Output image sequence:

Render engine type:

Render device type:

Render option:

Estimated render time:

Aide memoire

Job consists of a single letter prefix followed by a unique identifier (number).
i.e. (P) for production, (T) for test and (R) for rework.

Job description is a short description of the animation for later reference.

Input filename is the full filename with extension, of the animation file.

Output image sequence is the specification for the labelling of each image.
It must include the hashes needed to number the sequence e.g. ‘<job> + ####’.

The render engine type is the render engine used by the animator
i.e. Cycles, EEVEE, Workbench. The render farm may use a compatible alternative.

Render device type is either 'C' for CPU, 'G' for GPU or '*' for either.

Render option is the the quality settings, either 'HIQ' for high image quality,
'FRT' for fastest render time or 'O' for leave original settings

Estimated render time is preferably the Per-thread Frame Duration in seconds.
```

UTILISATION SHEET

A simple spreadsheet is a useful means of planning renders and analysing render farm utilisation and performance.

UTILISATION SHEET

For:.....

Hourly Slots	Master-1		Render-1		Render-2		Render-3		Render-4		Render-5		Render-6	
	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														

Insert Job # in all planned slots

Operations Readiness Check

Automation Scripts

Current automation scripts install as per instalation schedule.

Job Ticket

Blank Job Ticket available in drop_box share directory

Scheduling

Blank utilisation sheet available in master scheduling directory

Current utilisation sheet available in master scheduling directory

Archiving

Any files marked for archive to be transferred to Filserver-1

Share Directories

All share directories vacant and ready.

HOST BOOT SEQUENCE

Boot master-1 and wait until fully booted (share directories are active)

Boot required render hosts in numerical order

Job Submission

From workstation

1- Access job ticket proforma in drop_box and make a copy

2 - Rename to the next job number, enter information

3 - Save to to drop_box along with animation file.

Job Scheduling

1 - Inspect drop_box for new content

2 - Move animation file and job ticket to staging directory

3 - Inspect current utilisation sheet and calculate Expected Render Time

4 - Schedule job on Utilisation sheet

Preprocessing

From master-1 master login

1 - `$ cd render_bin`

If Original settings (O)

2 - `$ sh fa.sh –i staging/<infile>`

Else If High Image Quality (HIQ)

3 - `$ sh hiq_rp.sh –i staging/<infile>`

Else If Fastest Render Time (FRT)

4 - `$ sh frt_rp.sh –i <infile>`

If GPU render

5 - `$ sh gpu_rp.sh –i <infile>`

If CPU render

6 - `$ sh cpu_rp.sh –i <infile>`

If Production or Rework type

7 - Move animation file(s) from staging to prod_render directory

Else

8 - Move animation file(s) from staging to test_render directory

9 - Move job ticket to job_ticketss directory

10 - Identify render model

Initiate Model 1 Render

From master-1 master login

1 Determine if type is Production, Rework or Test

For each render host

2 `$ ssh tracer@<host IPaddress>`

3 `$ cd render_bin`

If Production or Rework type

4 `$ sh prod_ren.sh –i <infile> –o  <image_sequence>`

e.g. $ sh prod_ren.sh -i myfile.blend -o myfram####

Else

5 `$ sh test_ren.sh –i <infile> –o  <image_sequence>`

6 On completion move all frames to compositing directory

`$ mv <type>_images/<seqname>*.png compositing`

Initiate Model 2 Render

From master-1 master login

For each render host

1 $ ssh tracer@<*host IPaddress*>

2 $ cd render_bin

If GPU render

If Production or Rework type

3 $ sh prod_ren.sh –i <*infile*> –o <*image_sequence*>

e.g. $ sh prod_ren.sh -i myfile.blend -o myfram####

Else

4 $ sh test_ren.sh –i <*infile*> –o <*image_sequence*>

If CPU render

If Production or Rework type

5 $ sh cpu_prod_ren.sh –i <*infile*> –o <*image_sequence*>

e.g. $ sh cpu_prod_ren.sh -i myfile.blend -o myfram###

Else

6 $ sh cpu_test_ren.sh –i <*infile*> –o <*image_sequence*

7 On completion move all frames to compositing directory

$ mv <*type*>_images/<*seqname*>*.png compositing

8 Render image sequence

COMPOSITING

Compositing may be performed on the master host.

VIDEO EDITING AND AUDIO EDITING

Video editing could be performed on the master host or workstation. Audio editing will require audio applications,high quality audio output and closed earphones that are more suited to a workstation.

ARCHIVING

FreeNAS uses the same Linux distro and has detailed installation instructions. An ex-corporate/government PC with additional storage is a good option as the fileserver host.

1 - Move animation files from composting directory to for_archive directory

2 - Perform archival process

3 - Ensure any files required for rework or future reference can easily be recalled bask to the staging directory. A common mistake is to design archival and backup procedues without allowing for quick and easy recall.

ROUTINE MAINTENANCE

Periodic actions are needed to maintain the hosts at top performance. Maintenance includes:

Backing up fie storage

Cleaning up temporary files

Checking storage devices are not full or nearing full

Clearing out log files

It is important to maintain all render hosts to the same softwere installation and configuration. Ideally they should not be used for storage or any are application.

CONCURRENT PROCESSES

Using SSH is a simple means to access all render hosts from a single command line but therefore a single shell. A VNC session sing te vlient on the master host can be initiated for any render hosts to access its desktop GUI. This provides a means to open two separate command lines and initate a render process from each, resulting in concurrent processes. Although two concurrent CPU renders are possible the farm is designed to support concurrent CPU and GPU processes. The CPU process allocates for the operating system and a GPU process.

PROCESS MONITORING

The VNC master host VNC client can be used to run monitoring tools

PROCESS DISCOVERY

An ongoing process to optimise host performance begins with discovering all the services and applications, in particular those that are started at boot time and remain active.

Listing all systemd service units:

```
$ systemctl list-units --all --type=service --no-pager
```

```
Listing all units of all types and disposition:
```

```
$ systemctl list-unit-files --no-pager
```

Listing all active units

```
$ systemctl list-units --all --type=service --no-pager | grep running
```

List enabled or disabled services

```
$ systemctl list-unit-files | grep enabled
```

```
$ systemctl list-unit-files | grep disabled
```

In addition to systemd, shell scripts and programs can be initiated from logon profiles, CRON and the startup program on the Lightdm desktop. CRON is a job scheduler often used to run routine maintance automation. If not used it is one of the processes that can (should) be halted. The programs started by the desktop start-up program are accessible from Control Centre - Startup Applications. The program can be used to autostart shell scripts and applications, some of which are not needed by the render process. There are hidden startup applications that are not immediately discovered through the console. To un-hide them:

```
$  sudo sed –i ‘s/NoDisplay=true/NoDisplay=false/g’ /etc/xdg/autostart/*.desktop
```

OPTIMISATION

Increased performance is possible by halting services that are not needed however great care is needed as it is a task for experts. Once all services and startup applications are discovered and potential candidates are identified, it would be wise to consult a Linux or Ubuntu forum. Any actions to halt services should be rested on a proxy installation that can easily be rebuilt, i.e. an old laptop.