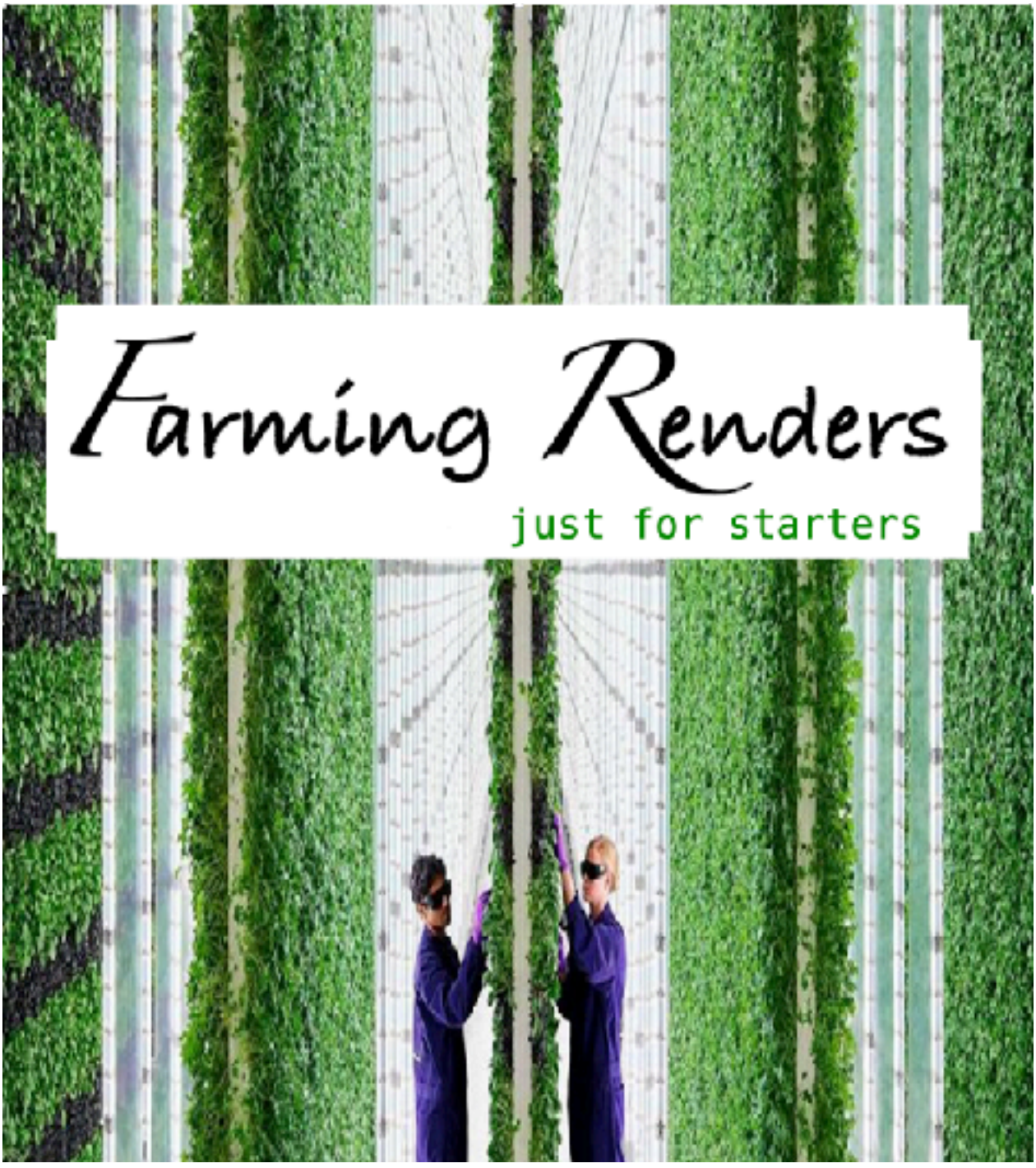


A guide for Blender users to construct an affordable, self-contained render farm that can be operated from a domestic studio.

THE CHALLENGE

It's an unescapable fact that if you want to produce quality 3D animation of any length, it's going to take a lot of raw computer power to do the image rendering. Even a high performance workstation will struggle if tasked to simultaneously perform animation tasks and rendering tasks.

Those artists who are determined to bridge the gap between several frames of great animation to several thousands of frames are faced with some challenges not the least of which is that the processing power also has to be paid for. There are no free lunches or free renders. One way or another you will have to fund the rendering. Another challenge is that the infrastructure needed for rendering is technically complex. The sheer amount of processing needed is well beyond a single PC but a proven approach is to harness the combined power of several PC's in what's called a 'render farm'.

Any one who is motivated enough and meets the challenge of building their own render farm is in a position to operate independently. More good news is that it is affordable. To put it in perspective, a small render farm with enough power to render several minutes of quality animation can be built for around the same price as a good leather lounge or a 4k television set.

IT professionals who build commercial render farms use advanced techniques however much of the infrastructure is aimed at achieving a profitable business model and not always at an animators use cases. So the real challenge is to build a cost/effective, self contained render farm using techniques that are not too complex and are aimed at the render requirements of a studio artist. Thats why this guide, or handbook was written.

The handbook has four parts, a primer, a host build guide, a software installation instruction and an operators manual. The main subject is the software installation guide. The other parts are there by way of completeness.

The primer exposes the technical environment and solution development. 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 purpose of the host build guide is to suggest ways to acquire and construct a cost-effective, flexible rendering platform that can be operated from a studio with access to only limited domestic electrical power. The platform is suitable to install the essential software for both CPU and GPU rendering using the Blender 3D animation package and its standard render engines, Cycles, EEVEE and Workbench. The guide does not aim to produce a platform with the highest possible performance characteristics.

Starting from a bare hardware platform the software installation instruction provides detailed procedures to install Linux, the needed utilities, Blender and render automation scripts. It should first be read carefully until it is clearly understood and double checked if any content is uncertain.

The operators manual includes the workflow of a basic render process.

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

Wayne (Wazza) McGrath
of Bonny Hills, New South Wales

SYSTEM REQUIREMENTS

Operating System and Applications

All host machines must be capable of supporting Linux Ubuntu 20.04 LTS Focal Fossa and Blender 2.83 LTS

Motherboard

Micro ATX form factor preferably ATX 12V 2.0 capacity, all hosts

Storage and RAM

Either a SATA, mSATA or preferable a M.2 channel to support an SSD boot drive with 500Gb for master host and 128 Gb for render hosts

Two DDM3 or DDM4 slots to support a minimum of 20 Gb for a render hosts and 8 Gb for the master host

I/O and Peripherals

Three USB 2 or USB 3 edge connectors, all hosts

A Gigabyte Ethernet device and connector compatible with Focal Fossa, all hosts

Rendering Capacity

A multi-core CPU with a minimum of eight high performance logical cores (threads) is essential for render hosts. A multi-core CPU with four high performance cores for master host

One PCIe x16 slot to support a GPU, all hosts

Power Supply

High efficiency 550 Watt with connectors to power motherboard, SATA device and 6-8 pin PCIe device. Preferably SFX form factor for render hosts. High efficiency 300 Watt supply for master host.

Communications

An 8 way plus a four way Gigabyte Ethernet switch, or a 10 or 16 way switch.

8 x 2m and 2 x 5m cat 6 Ethernet cables

Control and Monitoring

Two AV2000 2 port Ethernet extenders

A Wifi or Bluetooth enabled power line monitor and a smoke detector

Housing

A cabinet with nine removable host/ancillaries racks

Part 1 - Primer

Render Farm Overview

Design Guidelines

The design goal is a cost effective, reliable, easy to administer rendering solution with limited power consumption. Achieving these goals requires customisation of a standard 'out of the box' hardware and software. Administrative overhead is achieved by using a standard software configuration. Host machines are constructed to a common build to minimise variations in performance and quality. Peripheral costs are reduced by using remote access without the need for monitors and keyboards on each host.

This render farm 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 in 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 requires a CPU with fast single core operations while still allowing sufficiently fast multi-core operations. It also benefits from 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 for what they intend. 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.

THE ROLE OF LINUX IN CUSTOMISATION

A render farm is a systematic application of special purpose 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 requirements.

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. This opportunity is not provided in most other operating systems.

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.

A Cautionary Word for New Linux Users

Linux is a two-edged sword. Access to source code enables flexibility in providing solutions but at the same time it enables variability in functionality from one installation to the next. This variability is evident in the disparate results of a search for what command to use, how to configure a service or from where to install a utility. Human perceptions of quality can be influenced by how much variability there is in products from the same source. For example, the successful marketing of Japanese 'quality' products depended on a zero defects program that aimed to remove even the most minute differences between one product and the next. In that sense Linux is the antithesis of a mainframe operating system that remains stable over many years. Even though Linux functional differences may be necessary and well implemented, they can be perceived as bugs. For these reasons, it is recommended that all render farm hosts are configured to be as identical as possible.

CHOOSING A LINUX DISTRO

Distos are classified by stability criteria and features criteria. The versions with the latest features are often more buggy and less stable than those with older features. The features candidates are a ‘light’, ‘standard’ or ‘server’ distro. The light distros require expert customisation due to insufficient device drivers and significant differences to the base version. The server distros will be the most stable but lack a GUI so they are not an ideal starting point for most users.

Most all standard versions will have good support for routine customisation via their on-line repositories. A distro that includes ‘closed’ or non-open source device drivers from a commercial source can be a real plus. Some developers hold firmly to an ‘all open source’ configuration but others support closed device drivers. A few distros limit the number of end-user applications, however they mostly include a desktop GUI user interface that consumes significant memory and processor resources.

The kernel has evolved over time and development has forked into numerous distributions or ‘distros’ resulting in potential for incompatibility. The light, standard and server distros are 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 are customised for older machines and may not support all the services needed for more recent high performance devices. The standard versions are full-featured with a 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 render host. Server versions are preferred by operators of large scale installations but they are ‘headless’, meaning they don’t have a desktop GUI and are optimised to support server 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.

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 select a version with access to the relevant repositories and the tools to make installation a straight forward task.

MANAGING LINUX CUSTOMISATION

A thorough search is unlikely to find a distro that can be downloaded and installed ready for rendering so customisation is inevitable. Installing additional software is straight forward however removing unneeded services to free up resources for rendering requires expert knowledge. Successful customisation is more likely if a suitable version of Linux is used to begin with. The desktop GUI is often what influences the choice of distro however it is not significant for render farm customisation.

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. The architecture 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.

Although not unique to Linux, additional utilities will need to be installed and kernel dependancies may need to be updated prior to a compilation or installation of new software, so to avoid configuration problems, it is necessary to spend time and effort determining what dependancies are installed and compatible with any added utilities. Not managing the configuration and updating it when needed can result in a failure that is difficult to recover from.

Given that there is less technical risk in halting 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 begin customising with a proven, current desktop version with support for closed device drivers and an option for a minimal installation. A stable version with Long Term Support (LTS) will reduce the risk even more.

Network Rendering

A render farm 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 Independant 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 IT administrative 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 home-alone sole user.

The cloud provides functional transparency of the rendering process. The user uploads a prepared .blend file via a web browser and waits for notification that the finished output is available for download. The preparations are often to meet the requirements of the render farm. No knowledge of, or control over the rendering process is necessary on the part of the user. By definition there s no spare capacity in a render farm so what infrastructure is used must be paid for.

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 part of the operation as required by the master controller. It possible that processing power may leak.

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 allowances 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 installed on 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 initiated from a command 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 allocate all hosts to a single job or alternatively allocate them among concurrent processes such as test jobs, production jobs and rework jobs.

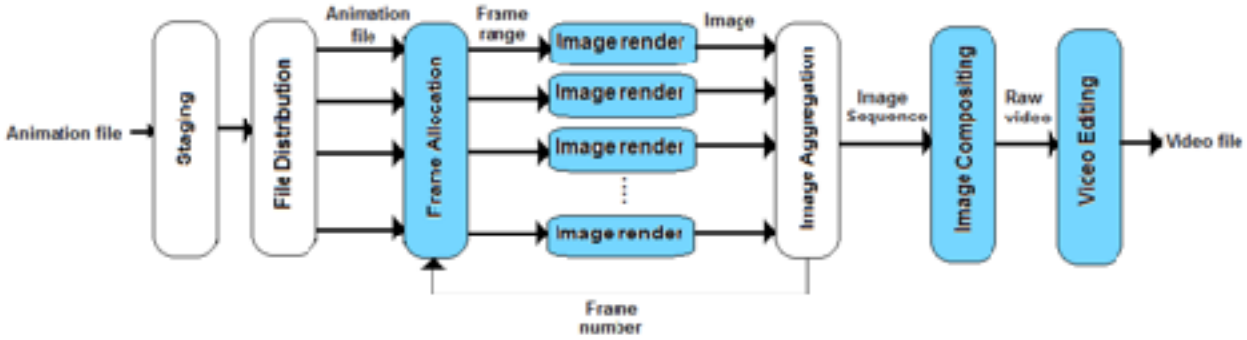
The larger the animation the more difficult it will become to assess how long the render will take. An estimate will be useful in deciding if some adjustments will need to be made or a contingency to be utilised. To meet the requirement the drop-box function should provide a job assessment.

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 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.

A Network Rendering 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 (SSH). By generating an encryption key on the master host and storing it on each render host, the master host can remotely login to a render host and issue commands 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). Once generated, the encryption key can be transferred manually using USB media or across the network using a special SSH copy command.

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 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 arising 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 it is practical to use GPUs from different vendors on the same render job by using the hosts Blender 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 prefix to the beginning of the file name so it can be clearly identified for rendering. The labelling is useful to ensure that any subsequent rework can be 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 needed to switch between CPU device or GPU device and avoids problems that can occur if 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 original Blender settings may be different 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 host's Blender as a contingency however these settings cannot be saved to the central 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 consistent 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 separate 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. frame12.png and frame12.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 number of logical cores on the host CPU. This model is for allocating sufficient threads to a render while still 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 is 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 factor in getting the most use of the render farm with least 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 (or a good guess). 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 / 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. Process monitoring tools are installed to identify where delays and bottlenecks occur due to resource contention with unneeded processes. Different tools may be needed depending on the render device in use.

Archiving

Archiving is based on Network Attached Storage. FreeNAS uses a Linux distro and has detailed installation instructions. An ex-corporate/government PC with FreeNAS and additional SATA hard disk storage is a good option as the fileserver host. The fileserver must be accessible on the same local network as the workstation(s) and master host but ideally not connected via the same Ethernet switch as the master and render hosts to avoid network contention.

Potted History

The basis of render engine functionality is the ray trace software algorithm. Ray tracing algorithms perform many geometric and trigonometric operations that calculate the behaviour of individual rays of light emanating from multiple light sources and reflections. 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 was a server that can run multiple processes in support of many general-purpose users, not one special application of parallel floating point calculations.

The first graphics cards were used for video display and provided only Z buffer (or frame buffer) functions. The earliest devices were referred to as a graphics adaptor (GA) and developed into the enhanced graphics adaptor(EGA), colour graphic adaptor (CGA) and video graphics array (VGA). Rendering algorithms such as the Bresenham raster algorithm were originally coded in application software but were eventually abstracted out along with the ray trace algorithm into specialised render engine middleware. Over time, specialised high performance hardware processors like numerical processing units, digital signal processing units, floating point arrays and gate arrays were developed for special near real time applications. Game console manufactures notably Nintendo, developed the techniques for integrating these processors to operate in a parallel. Hardware parallel processing models were combined with video adaptors and the modern graphics processing unit (GPU) was the result. The next phase of development critical to animation was the development of GPUs that emulated the software ray trace algorithm. These came in two basic forms, hardware accelerators such as CUDA cores used on Nvidia GPUs and software shaders used on AMD GPUs. Currently models are available with a few hundred accelerators up to 10,000 or more.

Although there remain many algorithms used in animation that require a powerful CPU the performance of a GPU with ray trace emulation 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 drivers for high performance GPUs are artificial intelligence, science based applications such as weather forecasting and financial based applications such as crypto coin mining. The objective for ray trace emulation on a GPU is not the highest possible image quality but near real time rendering of video game graphics with complex spacial content.

Neither multi-core CPUs or GPUs are ideal for rendering images for animation. One is too slow, the other has limited memory 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 in a middleware role. 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.

The GPU rendering model accesses hardware functions via an abstraction layer that hides the details of its ‘metal’ operation. Advanced GPUs have multiple modes of operation, e.g. raster operations, integer and floating point numerics, ray trace tensors 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 challenging. 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.

Render Engine APIs and GPU Parallel Processing Models

Application software accesses the functions of rendering middleware via a type of contract called an Application Programming Interface (API). Essentially the contract specifies what functions the middleware will perform if given specific commands and data from the software. The middleware in turn uses drivers and abstraction layers to access a hardware device such as a GPU.

Technically a parallel processing model is distinct from an API and refers to the type of computations that are supported by hardware and how they are controlled. As the software determines what processing models are supported the term API will be used as a convenience to refer to an interface with processing models and hardware abstraction.

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. 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 software, render engine software and GPU processing model can be at odds. Careful analysis of API version requirements is essential to prevent wasted time and money.

APIs generally have a longer development time than a GPU architecture and will likely be around for the release of several GPU models.

GPU Selection by Shortlisting

Choosing a standard GPU may be achieved by performing an informed 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 processing models, or GPU type are the main determinants of suitable 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.

Next compile a list of GPU models that supports both sets of API. 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 not 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. Multicore CPUs manufactured for the server market may not have efficient ALUs as they are not critical whereas multicore CPUs for the PC market need efficient ALUs. A four core PC CPU may out perform a 12 core server CPU of the same generation. 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 (threads) the better. Fewer than 8 is not recommended.

Current State of Parallel Processing Standards

In order to use a combination of different CPU cores and 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 is no equivalent standards organisation. Some use distinct channels for CPU instructions vs GPU instructions and others use 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.

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.

GPU APIs and Parallel Processing Models

OpenGL is widely supported by GPU vendors as a graphics display API however only later versions may be compatible with current application software. There are several competing parallel processing models 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. Nvidia capability has gradually progressed and is defined by Compute Capability. Blender, or more specifically Cycles requires Compute Capability 3.0 or above to perform CUDA rendering.

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 processing models. Also, it is generally acknowledged that Nvidia CUDA and Optix technologies in particular, perform better (faster) on Linux when using proprietary (closed source) drivers.

What to Choose

There is endless debate about which GPU model is superior. Clearly the latest generations are faster and more capable however within any generation there are pros and cons that fuel the debate. To add to the mix there are several alternative render engines, some of which are compatible with Cycles. For example AMD Radeon has developed the free Prorender engine with device drivers for all major operating systems, however it uses its own material library. There are also commercial alternatives like V-Ray and Renderman.

GPU capability is predominantly a function of the number of transistors on a chip however it is simplistic to say render engine capability is a function of lines-of-code. There are two main classes of ray trace algorithm, unbiased and biased. In simple terms, unbiased means the algorithm attempts to achieve photorealism by faithfully simulating the physics of light. Biased means the algorithms will omit steps that do not result in a noticeable difference or use alternatives to simulating ray paths. GPU rendering is more akin to a biased ray trace algorithm. Once again there are pros and cons that fuel an endless debate about which is better. At any rate Cycles is a useful unbiased render engine that is good for photorealistic animation and easy to use. It is somewhat slow compared to a biased render engine but comparable to other unbiased engines. The catch is that you have to use a GPU or render engine to discover its capabilities and find out if they are what you require. In a commercial operation those requirements my be mandated by a client.

In summary, because OpenGL is a standard available on all GPUs the current situation means a choice has to be made between either Nvidia GPUs with CUDA/ Optix with Compute Capability 3.0 or AMD Radeon GPUs with GCN2+. As in most situations the choice will not be a simple matter of rational selection and a measure of experience will help.

HOST IDENTIFICATION

Efficient network operation will require that each host machine is quickly identified by their host names, IP addresses and 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 host 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. Each host can maintain a lookup table of the other hosts on the network by recording their hostname in a special file. This method is similar to a more generalised naming convention 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 used to locate resources used by the render farm 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 use 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 allocated by the ISP for use in the Internet and another allocated from a local subnet for use in the local area network (LAN). The local subnet address range is 192.168.0.1 thru 192.168.0.255 and the gateway is allocated the first 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 of 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 service 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 own subnet IP address 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 web browser or 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 arguments that are passed to the program to modify its behaviour. They can be issued from from a desktop GUI 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 that the up arrow and down arrow functions are very useful to recall previous commands and make the job easier. Just to be clear, a terminal is a monitor and keyboard used to access a mainframe computer. A headless console has a PC terminal emulator program but no desktop GUI.

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 arguments to modify program behaviour. An example is the command to initiate Blender, $ blender. To be clear, during execution of a program internal functions can passed variables called parameters but variables passed externally to a program on startup are referred to as arguments.

Scripts

A scripting language is a text based 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). Even more restrictive, some commands can only be issued by an administrator with access to the file system root directory (# 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. All the commands can be executed with one command based on the shell script name. In Linux it is not possible to code shell commands in a text file and immediately execute it. To protect against inadvertent 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 executing 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.

USING ALTERNATIVE COMMANDS

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.

Using 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 organise them in folders for 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.

Using Shell Scripts

Shell scripts are typically used to automate admin functions. They can be stored anywhere within the file system however 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).

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 cores are allocated to the render task the kernel will need to continually interrupt the processes and obtain control of the 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, e.g. top that periodically extract data from usage logs recorded by the kernel and averages the values to report on resource consumption. In a multi-user system these reports are sufficient to identify a user that is consuming resources at the expense of others, however in a special purpose system with multiple CPU cores, the averaging approach can hide short peaks of usage that may be the main cause of contention problems. Other useful tools are available and easy to install, e.g. htop and CoreFreq.

Normally there are numerous system processes running in parallel along with user application 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. The relationship between the scheduler and logical cores is referred to as affinity. Affinity is manageable using process pinning and isolation. If the render engine has allocated threads to all available cores, the scheduler will direct interrupts across those cores with resulting render performance hits. Performance enhancement tools, e.g. Taskset are available to modify the affinity so some cores can be dedicated (pinned) to the render process without being interrupted. Pinning a process in itself does not prevent other processes from using those cores also. Settings are needed to isolate system and other processes to specific cores so the render process has 100% utilisation of the cores they are pinned to and the related cache memory.

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.

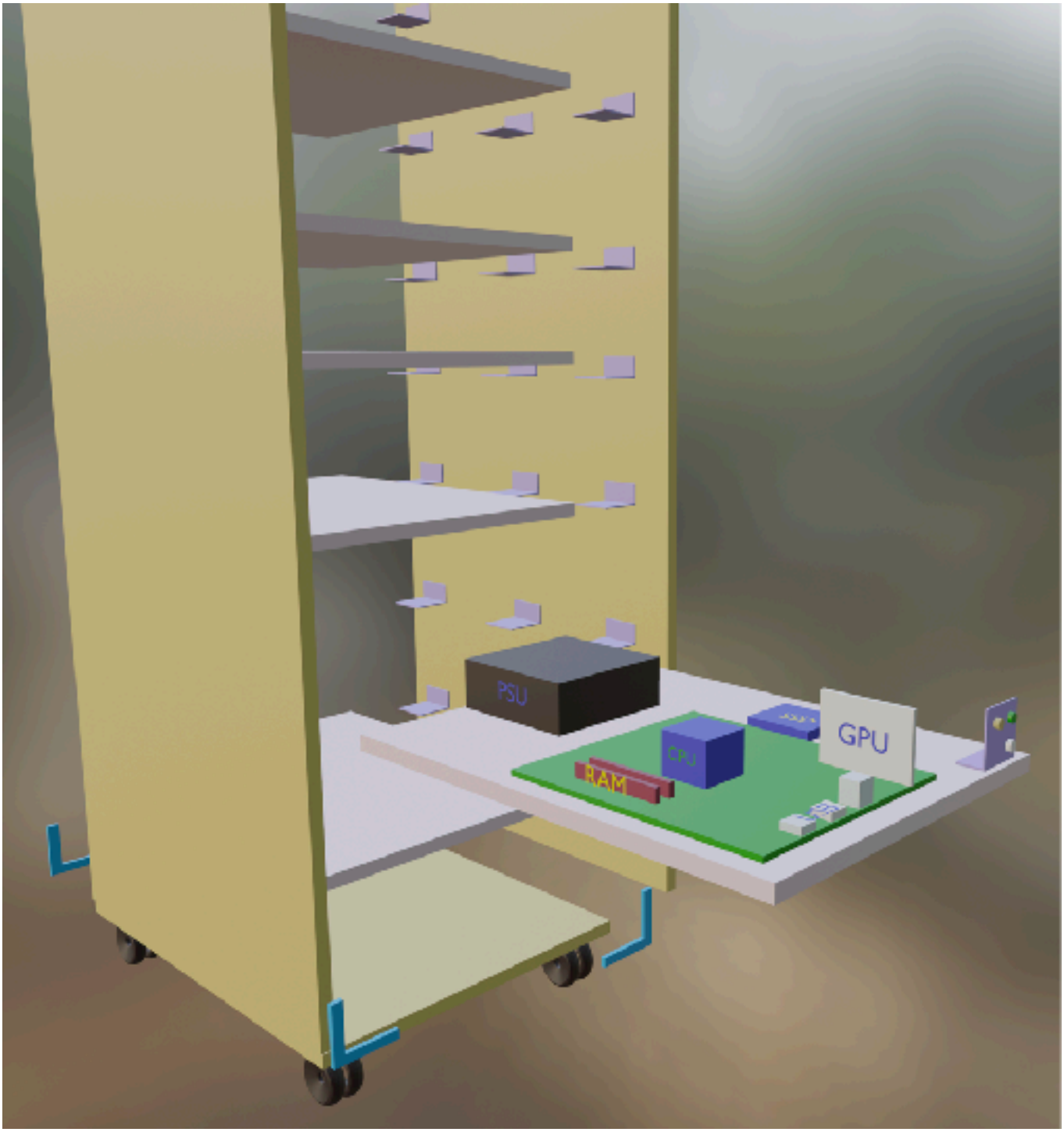
PERFORMANCE LIMITATIONS

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, however fast memory channels and multiple memory channels can measurably improve single core 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 a real time gaming experience with high power consumption and a risk of overheating. A host with modest power consumption and a prolonged duty cycle is necessary even at the expense of rendering speed.

Electrical power is consumed when transistors change logic state. The more transistors and the faster they switch state, the higher the power consumption. The electrical power used by a gaming PC can be in excess of 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 moderately fast devices with low power consumption is not wasted in a domestic render farm.

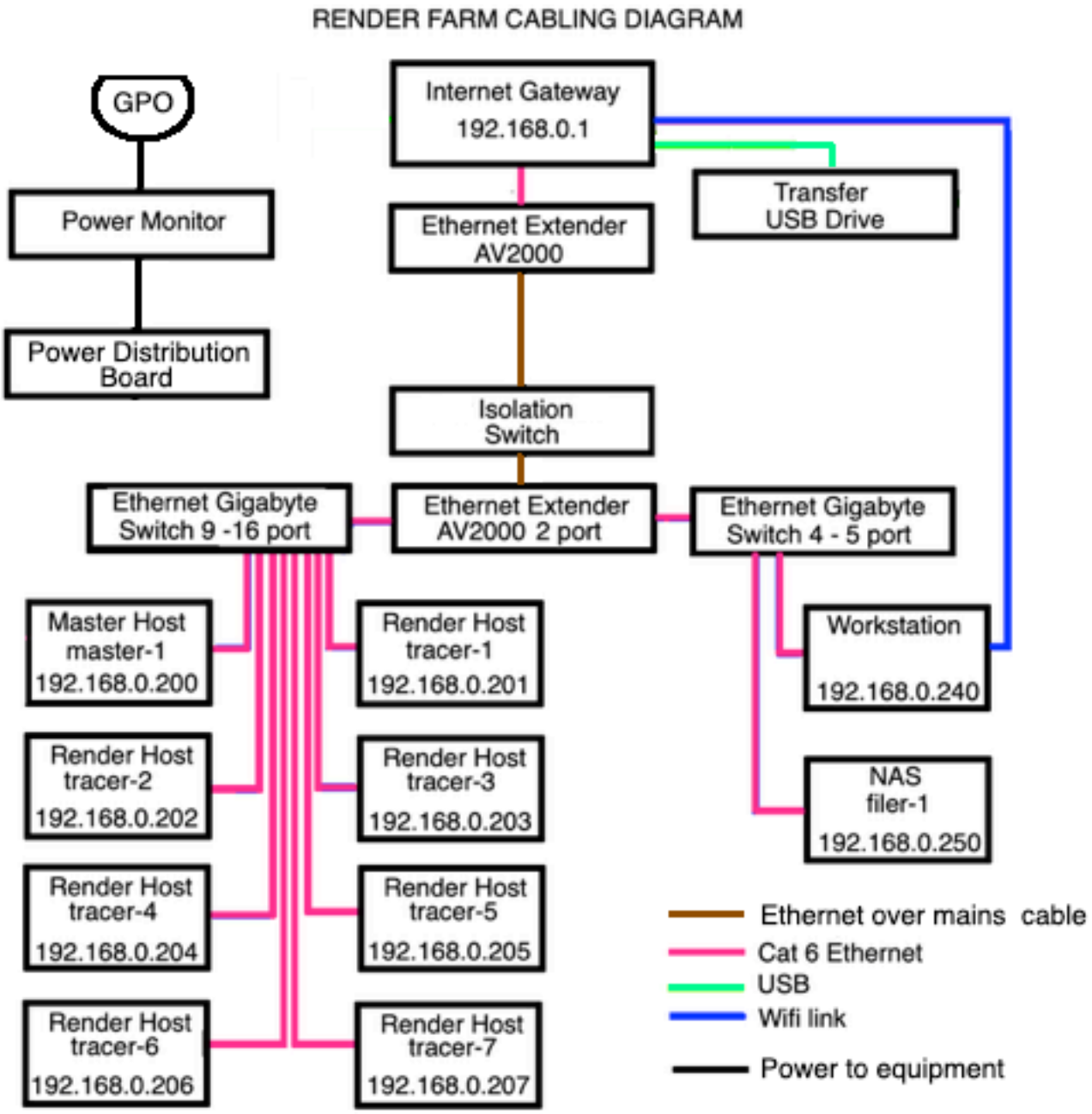
Given similar hardware capability the API used on a GPU can make a significant difference for certain render tasks. Closed source device drivers have a performance advantage over reverse engineered open source equivalents. Currently Ubuntu Linux is preferred as it supports a range of GPU technologies. 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 may be on the same circuit as general purpose outlets. (A render farm works just as well in the dark.) Calculate a realistic Watt budget to aid in making build decisions.

Reliability

Electronic components 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 have both a multiple core CPU and a ray trace capable GPU. All hosts need to be mounted in a cabinet with 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 Sourcing

It is highly recommended that new components are used to construct a host. If only recycled components can be afforded 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.

Ex-corporate/government PCs are a source of inexpensive motherboards and higher spec CPU's, RAM and SATA SSDs can be acquired separately and assembled into a reasonable host. If the motherboard uses non-standard PSU connections some re-engineering may be required. For example, 24 pin to 10 pin conversion kits have a loop from pin 16 to pin 17 to emulate the PS-ON signal. The loop is needed to turn the PSU on but then it can't be turned off. This can be overcome by cutting the loop in the middle and using a double pole momentary switch to power on. One pole connects to the normal power-on header, the other to each end of the cut loop. This arrangement will function normally. Retailers with stocks of new 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.

The initial GPU can be an inexpensive model with a few hundred shaders or ray-trace accelerators. Higher performance 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 higher Wattage to cater for later upgrades to CPU and/or GPU.

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 32 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.

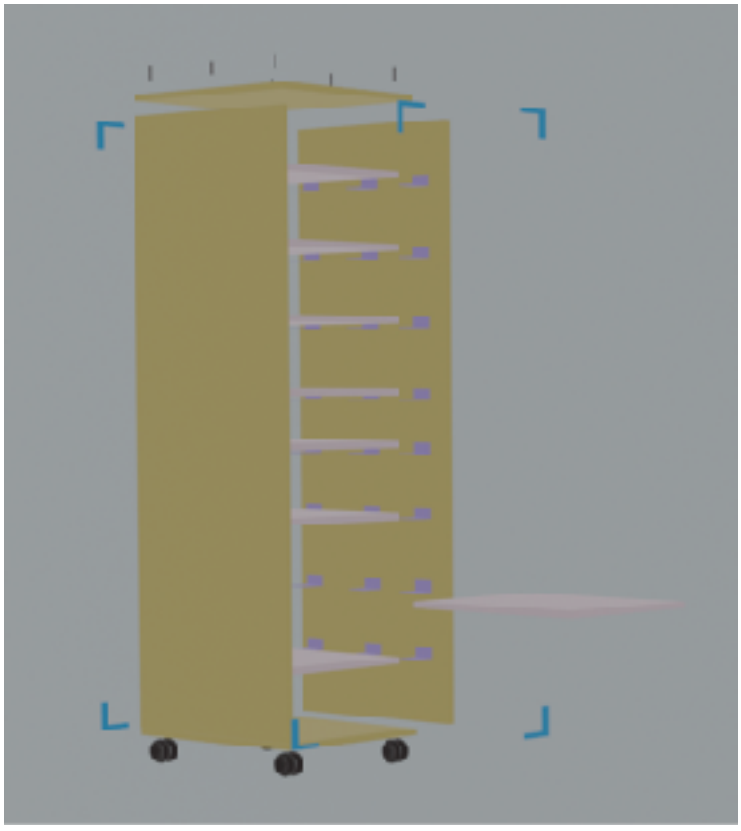
Carefully insert all host platforms and switch platform into the case.

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

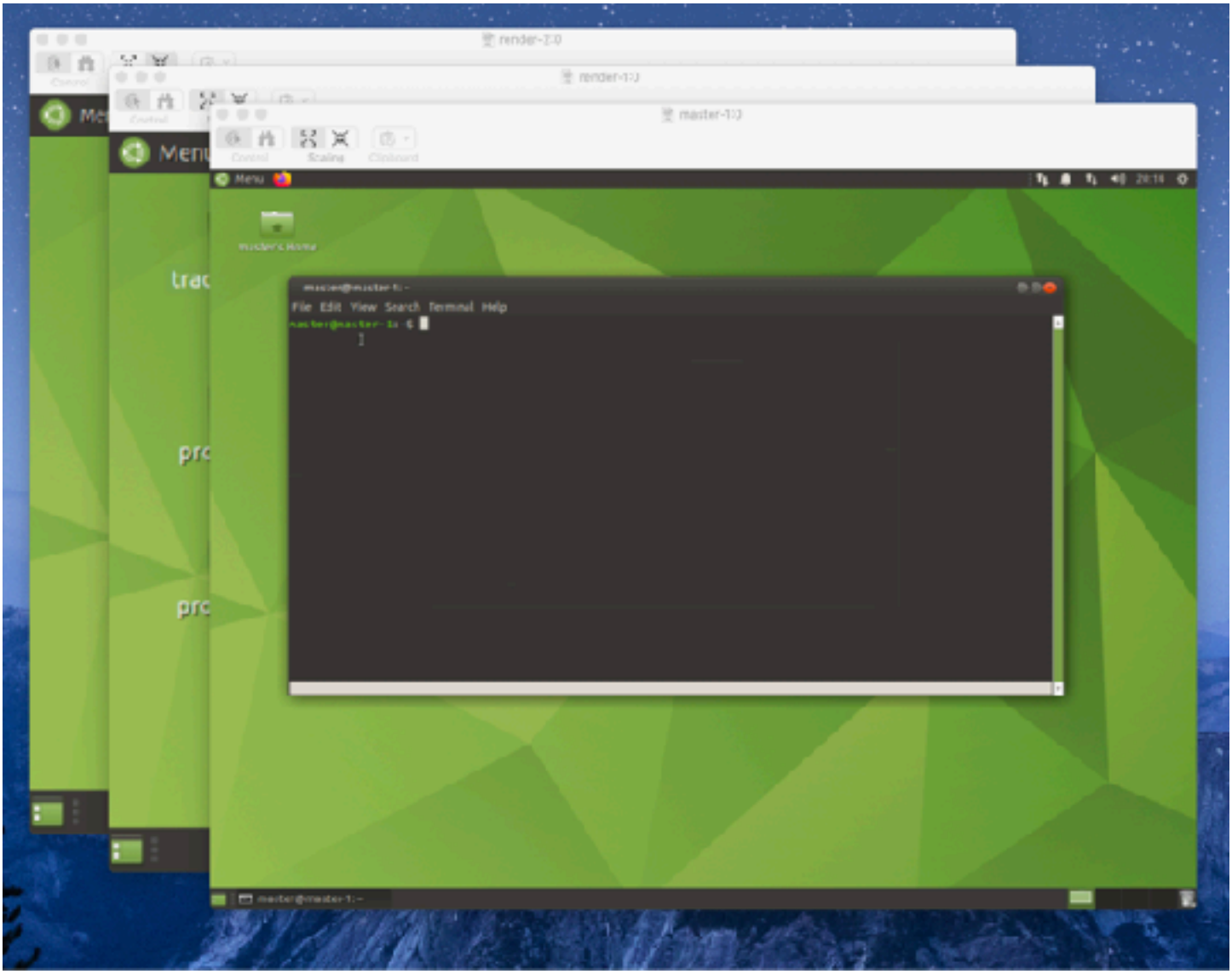
Connect all PSU and LAN switches to power board.

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

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 created 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 a 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 verbatim 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 command line shortcuts however using the up and down arrows to recall previously used commands is extremely useful. Copying a text version of this instruction to the host desktop and cutting and pasting commands and settings is possible.

In some instances the installation of Linux is interrupted and errors are reported. Just repeat the installation. Also, the auto-update feature may place locks on dependancies while installing applications or utilities. Just wait until the lock is released.

Gateway/modem Configuration

Modem/Router Info

Model: e.g. Telstra Smartmodem

Network Name: e.g. 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: 192.168.0.199

DLNA Sever enabled.

RESERVE STATIC IP ADDRESSES

Hostname	MAC Address (Ethernet)	Static IP Address
master-1	tba	192.186.0.200
render-1	tba	192.168.0.201
render-2	tba	192.186.0.202
render-3	tba	192.168.0.203
render-4	tba	192.168.0.204
render-5	tba	192.168.0.205
render-6	tba	192.168.0.206
render-7	tba	192.168.0.207
workstation-1	tba	192.168.0.240
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

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 Focal Fossa 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 Mate 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.

INSTALL UBUNTU MATE 20.04

Install Ubuntu on all hosts SSD drive:

Preliminary to installing the OS the host BIOS may need to be to be set up to use a USB device as the default boot media. This is achieved by inserting the boot USB and restarting the machine while holding down the delete key or other function key used to access the BIOS, e.g F2, F10 or F12 . 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 the boot process to commence.

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 Ubuntu 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` **# It is essential to update the installation before proceeding as the installer package may not be fully up to date.**

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 Ubuntu 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

SET 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 MAC ADDRESS

The Ethernet MAC address is needed to register a static IP address with the Gateway/router. To display only the ethernet MAC address. Take care not to obtain a WiFi address if one exists.

1 - `$ ifconfig | grep ether`

SET STATIC IP LAN 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` # 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

Note: Driver selection depends on installed device. May have a problem with dependencies for some drivers due to a Ubuntu bug. if not successful try install using Preferences -> Additional drivers.

5 - $ `sudo apt install nvidia-utils-460` # this driver installs

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

7 - $ `sudo ubuntu-drivers install`

8 - $ `sudo reboot`

9 - $ `nvidia-smi` # depends on a successful driver installation

10 - $ `sudo apt install nvtop` # if file held issue, install by compile

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 # Step 1 required for Intel processors only

```
$ 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 # Remaining steps for Intel and AMD processors

```
$ sudo apt-get install linux-headers-`uname -r`  # Find ` key on top left under ~

$ 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 group for networking

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 tracer 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 tracer on render-x

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. Linux defaults to using DNS so changes need to be made to default to the hosts file. Also, because a minimal installation was used, the dependancies for using hosts need to be installed. In this solution only the master host will need to resolve host names. This step is optional for render hosts.

Login as admin user.

Install hostname resolution dependancies

1 - $ sudo apt-get install libnss-mdns

Edit hosts file

1 - $ `cd /etc`

2 - $ `sudo nano hosts`

3 - Add entries for all machines on render farm

```
192.168.0.200     master-1
192.168.0.201     render-1
192.168.0.202     render-2
192.168.0.203     render-3
192.168.0.204     render-4
192.168.0.205     render-5
192.168.0.206     render-6
192.168.0.207     render-7
192.168.0.240     workstation-1
192.168.0.250     fileserver-1
```

ctl x

Change hostname resolution order

1 - $ `sudo nano /etc/nsswitch.conf`

edit host: entry to read

```
hosts:  files mdns4_minimal [NOTFOUND=return] dns mdns4
```

ctl x

2 - $ `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` # from admin user

2 - $ `sudo systemctl status smbd`

Create Shared Samba Configuration and Shared Directories

The Samba configuration file is used to create shared directories for network operation including drop-box, pickup-box, render file input, image output, and video output. Directories are created for production and test renders to keep the processes separate. Non-shared directories are created for workflow operations.

Create directories to be shared or used for local workflow

Login as master non-admin 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 resources`

14 - $ `mkdir drop_box`

15 - $ `mkdir pickup_box`

15 - $ `mkdir job_tickets`

16 - $ `mkdir scheduling`

17 - $ `mkdir for_archive`

18 - $ `mkdir local_in`

19 - $ `mkdir local_in/local_out`

20 - $ `mkdir tmp`

21 - Delete unneeded directories e.g. /templates

Set local $PATH

From master non-admin user login.

1 - Open hidden .profile file

$ `nano .profile`

2 - append new line and save

```
export PATH="$PATH:/home/master/render_bin"
```

Create Samba Configuration File

Login as 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 an SSH Client is Installed on Master Host

In ssh terminology the master host is an ssh client and the render hosts are servers. An SSH client should be installed along with Ubuntu Mate. To check the installation...

1 - $ ssh –V

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

$ sudo apt install openssh–client

Generate an Encryption Key Pair

SSH has an option for a client (the master host) to connect to a server (a render host) without a login password by using a private/public encryption key pair. The key pair is generated on the master host and then the public key only is copied to a special .ssh directory on each render host. Login to the master user 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` # directories beginning with a '.' are hidden but can be accessed as normal

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

Note: All render hosts will use the same copy of the master host public key. The ssh-keygen process is also run on render hosts to create the .ssh directory needed to store the master host key and to generate render host keys that can be transferred to the master host known_hosts file. Two methods key transfer are available, a special ssh-copy method or by manual transfer using USB media. The ssh-copy method is more convenient for existing hosts that may be physically inaccessible. The USB media method is more convenient when installing the host software.

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 libreoffice`

1 - $ `sudo apt-get install pinta`

RENDER HOSTS ONLY

Install CIFS-Utils

The CIFS utility mounts shared directories as part of the local file system. Login as admin.

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 tracer non-admin user and create a local directories for mounting shares and local workflow.

1 - $ `mkdir prod_render`

2 - $ `mkdir test_render`

3 - $ `mkdir prod_render/prod_images`

4 - $ `mkdir test_render/test_images`

5 - $ `mkdir local_in`

6 - $ `mkdir local_in/local_out`

7 - $ `mkdir render_bin`

8 - $ `mkdir redirect_logs`

9 - Delete the directories created by the Linux installer, e.g. Templates, Videos, Music...

Set local $PATH

From tracer non-admin user login.

1 - Open hidden .profile file

$ `nano .profile`

2 - append new line and save

```
export PATH="$PATH:/home/tracer/render_bin"
```

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.

Login as admin

1 - $ `cd /etc`
2 - $ `sudo nano fstab`

```
# Shared directories

//192.168.0.200/prod_render  /home/tracer/prod_render
cifs  username=<admin>, password=<password>,
vers=3.0,iocharset=utf8,file_mode=0777,dir_mode=0777 0 0


//192.168.0.200/test_render  /home/tracer/test_render
cifs  username=<admin>,password=<password>,
vers=3.0,iocharset=utf8,file_mode=0777,dir_mode=0777 0 0


//192.168.0.200/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

//192.168.0.200/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, making sure the master host shares are configured and running.

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 or <hostname>.local

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 on Render Hosts

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 tracer user

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 master host authorized_keys file and copy it to the tracer .ssh directory.

Test SSH

1 - Reboot master host and tracer hosts

2 - Login to master-1 as master user

3 - $ `ssh tracer@<`*`tracer ip`*`>` *or* **4** - $ `ssh tracer@render-x.local`

On the first connection the render host password is needed and a warning message will appear. Type 'yes'. The render host credential will be added to the known-hosts file then a command prompt at tracer@render-x should appear. The process is repeated for both type of host address but further connections will proceed without this step. Note: Using the IP address is more reliable than hostname which may not always 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/blenderxxx"

ctl x, y  # save and exit
```

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)

Render Tests

Tests will confirm the software installation. The tests require a master host and a render host. The first test is a local GUI render (requires a Blender compatible GPU to be installed). The second test is a local background render. The third test is a network background render. The fourth test is a SSH session test. The local tests will use local directories and the network test will use shared directories. Vnc remote connections may be used otherwise connect peripherals as required.

Prepare Test Blender File

In preparation for render tests select a render host and ensure both master and render hosts are fully configured up to the completion of the Blender installation. On a workstation:

1 - prepare a simple animation with 5 or 6 frames. Name it '5test.blend'

2 - in the Output properties->Out tab, set Overwrite to unchecked and Placeholders to checked

3 - copy the file to a USB media labelled 'Blender Tests' under a directory 'Test_files'

Transfer the test file to the Master Host and a Render Host

1 - Make sure both master host and render host are shut down

2 - Start the master host and wait until boot is complete and shares are available

3 - Login to the master host as master user

4 - Copy the 5test.blend file to home/test/test_render # use a share for network test

5 - Start the render host and wait until boot is complete and shares are mounted

6 - Login to the render host as tracer user.

7 - Copy the 5test.blend file to home/tracer/local_in

Test 1 - Local GUI Render Initiated from Render Host

1 - `$ cd ~` # change to /home/tracer directory

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

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

4 - Set Properties

Render Properties

Render Engine > Cycles
Device > CPU

Output Properties

Output location: /home/tracer/local_in/local_out

File Format > JPEG

5 - Initiate render

Render ->Render Animation (Blender should render test frames to local_out)

6 - Close Blender

7 - Check there are rendered images in local_out

Clean Up

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

Test 2 - Local Background Render Initiated from Render Host

1 - `$ cd ~` # change to /home/tracer directory

2 - `$ blender -b  ~/local_in/5test.blend  -o //local_out/test_images## -a`

3 - Check there are rendered images in local_out

Clean Up

1 - Delete test images in local_out

Test 3 - Network Background Render Initiated from from Render Host

Note: Input and output directories are shares on master host but render is initiated from render host.

1 - `$ cd ~` # change to /home/tracer directory

2 - `$ blender -b  ~/test_render/5test.blend  -o //test_images/test_images## -a`

3 - Check there are rendered images in home/master/test/test/render/test_images

Clean Up

1 - Delete test images in /home/master/test/test/render/test_images

2 - Delete 5test.blend from /home/master/test/test_render

Write Shell Scripts for SSH Session Test

Test shell scripts are for routine testing of a render initiated from a SSH session on the master host.
On a workstation:

4 - open a text editor and write a shell script as follows:

```
#!/bin/bash

# local_test.sh – Initiates a local background render session
```

```
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 ~/local_in/$infile –o //local_out/$outfile –a
```

5 - save script as local_test.sh

6 - open a text editor and write a shell script as follows:

```
#!/bin/bash

# network_test.sh – Initiates a network background render session

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 –F PNG –x 1 \
             –a
```

7 - save script as network_test.sh

8 - copy the shell scripts to Blender Tests USB media under a directory ‘Test_scripts'

9 - Login to render host as tracer user

8 - Copy the local_test.sh and network_test.sh files to home/tracer/render_bin

9 - Change permissions of local_test.sh and network_test.sh file to executable

```
$ chmod +x /render_bin/local_test_.sh      # make shell script executable
```

```
$ chmod +x /render_bin/network_test_.sh
```

Test 4 - SSH Session Test

1 - Make sure both master host and render host are shut down

2 - Start the master host and wait until boot is complete and shares are available

3 - Login to the master host as master user

4 - Copy the 5test.blend file to home/test/test_render # use a share for network test

5 - Start the render host and wait until boot is complete and shares are mounted

6 - `$ cd ~` # change to /home/master directory

7 - `$ ssh tracer@192.168.0.20x` # start SSH session on /home/tracer

Note: If hosts file is working use `$ ssh tracer@render-x.local`

8 - `$ cd render_bin`

9 - `$ sh network_test.sh –i 5test.blend –o test_images##`

10 - Check there are rendered images in /home/master/test/test_render/test_images

Clean Up

1 - Delete test images in /home/master/test/test_render/test_images

2 - Delete 5test.blend from /home/master/test/test_render

Test Scripts

Used for routine testing render host.

local_test.sh

```
#!/bin/bash

# local_test.sh – Initiates a local background render session

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 ~/local_in/$infile -o //local_out/$outfile -a
```

network_test.sh

```
#!/bin/bash

# network_test.sh - Initiates a network background render session

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 -F PNG -x 1 \
             -a
```

Render Process Automation

Shell scripts and Python scripts need to be coded for each automation model and and installed on the hosts. Note: Ubuntu Linux uses dash with the sh command so some shell scripts will not run as expected with bash. Use the ./ command where indicated.

AUTOMATION SCRIPT INSTALLATION SCHEDULE

Scripts	Installed Machines	Directory
Preprocessing		
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
max_cpu.sh	master host	home/master/render_bin
Model 1 Render		
prod_ren.sh	all render hosts	home/tracer/render_bin
test_ren.sh	all render hosts	home/tracer/render_bin
set_ren_fa.py	all render hosts	home/tracer/render_bin
Model 2 Render		
cpu_prod_ren.sh	all render hosts	home/tracer/render_bin
cpu_test_ren.sh	all render hosts	home/tracer/render_bin
alloc_cpu_rt.py	all render hosts	home/tracer/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_gpu_rp.py  - set GPU render properties
#    sets device settings for a host with 1 GPU
#    writes out .blend file for a gpu 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 settings for host with 1 GPU
# set render device to use gpu for all scenes in infile
# set optimum cpu core usage for 1 GPU host

def set_gpu_rp() :

    threads = 0
    opt_gpu_tile_x = 256
    opt_gpu_tile_y = 256


    for scene in bpy.data.scenes:
        scene.cycles.device = 'GPU'
        scene.render.tile_x = opt_gpu_tile_x
        scene.render.tile_y = opt_gpu_tile_y
        scene.render.threads_mode = 'FIXED'
        scene.render.threads = threads

    return

set_fa()
set_gpu_rp()

# write file for cpu render

outfile = "g_" + 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 PNG -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()
```

RENDER PROCESS SCRIPT INSTALLATION SCHEDULE

Scripts	Installed Machines	Directory
Process Control		
render_control.sh	master host	home/master/render_bin
file_report.sh	master host	home/master/render_bin
preprocess.sh	master host	home/master/render_bin
max_cpu.sh	master host	home/master/render_bin
move_images.sh	master host	home/master/render_bin
archive_render_file.sh	master host	home/master/render_bin
archive_images	master host	home/master/render_bin

SCRIPT CODE

render_control.sh

```
#!/bin/bash

# render_control.sh – main control of render process

#   run using ./ not sh

# declare constants, variables and arrays


declare –r rbn="/home/master/render_bin";
declare –r tmp="/home/master/tmp";

# Initialise variables
declare –a process_steps;
inf="Information";
fls="Files";
pre="Preprocess";
rnd="Render";
mvi="Move–Images";
afl="Archive–File";
aim="Archive–Images";

# set colour variables
green="\033[0;32m"
yellow="\033[0;33m"
blue="\033[0;34m"
purple="\033[0;35m"
clear="\033[0m"


# define user functions

# function error_exit checks return code and exits
error_exit () {
  echo
  echo "$1"
  echo –e ${yellow};
  echo "Exiting ..........";
  echo –e ${clear};
  sleep 3;
  exit 1;
}

# function contin prompts user to continue to next
contin() {
  echo
```

```
  read -p "Continue (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo
    echo -e ${green};
    echo "Exiting ..........";
    echo -e ${clear};
    sleep 3;
    exit 0;
  fi
}

# function info - displays infomation about each step

info() {
  clear
  echo -e ${purple};
  echo
  echo "    Render Process Information";
  echo "    --------------------------";
  echo
  echo "    Control process menu options:"
  echo
  echo "    Files - reports on the files currently in the render farm
directories"
  echo "          - removes hidden (.) files and directories during the report"
  echo
  echo "    Preprocess - selects a file from the dropbox directory"
  echo "               - applies render settings fequired for frame allocation"
  echo "               - optionally applies render settings fequired for quality
control"
  echo "               - optionally applies render settings for optimal CPU
usage"
  echo "               - moves the file to the staging directory"
  echo
  echo "    Render - selects a file from staging directory"
  echo "           - moves the file to the render directory"
  echo
  echo "    Move-Images - checks if the compositig directory is available"
  echo "                - selects an image sequence from the image directory"
  echo "                - moves all images in the sequence to the compositing
directory"
  echo
  echo "    Archive-File - selects a file from the render directory"
  echo "                 - moves the file to ths for-archive directory"
  echo
  echo "    Archive-Images - checks if the compositing directory has an image
sequence"
  echo "                   - compresses the image sequence into a tarball"
  echo "                   - moves the tarball to the for-archive directory"
  echo "                   - removes all files from the compositing directory"
  echo
```

```
  echo
  echo -e ${clear};
  contin
}


###
# Display run notice
###

clear
echo -e ${blue};
echo
echo "     ******************"
echo "     * RENDER CONTROL *"
echo "     ******************"
echo
echo "     Main control for render process."
echo
echo
echo -e ${clear};

contin

cd ~/render_bin;

# display menu of render process steps


title="Render Process Options:"
options=("$fls" "$pre" "$rnd" "$mvi" "$afl" "$aim" "$inf")

do_menu=0

while [ $do_menu = 0 ]; do
  clear
  echo
  echo "$title"
  echo
  select opt in "${options[@]}" "Quit"; do
    case $opt in
      "$fls")
        ./file_report.sh
        echo
        break
        ;;
      "$pre")
        ./preprocess.sh
        echo
        break
        ;;
```

```
      "$rnd")
        ./max_cpu.sh
        echo
        break
        ;;
      "$mvi")
        ./move_images.sh
        echo
        break
        ;;
      "$afl")
        ./archive_render_file.sh
        echo
        break
        ;;
      "$aim")
        ./archive_images.sh
        echo
        break
        ;;
      "$inf")
        echo "$inf selected"
        info
        echo
        break
        ;;
      Quit)
        do_menu=1
        break
        ;;
      *)
        echo "No option $REPLY"
        ;;
    esac
  done
done

echo -e ${green};
echo "Exiting ..........";
echo -e ${clear};
sleep 3;

exit 0
```

file_report.sh

```
#!/bin/bash

# file_report.sh – displays file disposition in key directories

#   run using ./ not sh

# declare constants and arrays
# Note: Defined paths are used to prevent inadvertant access to system
directories

declare –r unk="unknown";
declare –r ntf="not_found";
declare –r fil="files";
declare –r emp="empty";
declare –r rbn="/home/master/render_bin";
declare –r tmp="/home/master/tmp";
declare –r dbx="/home/master/drop_box";
declare –r stg="/home/master/staging";
declare –r prd="/home/master/prod/prod_render/"
declare –r pim="/home/master/prod/prod_render/prod_images";
declare –r com="/home/master/compositing";
declare –r pbx="/home/master/pickup_box";

declare –a img_seqs;

# set colour variables
green="\033[0;32m"
yellow="\033[0;33m"
blue="\033[0;34m"
clear="\033[0m"


# define user functions

# function error_exit checks return code and exits
error_exit () {
  echo
  echo "$1"
  echo –e ${yellow};
  echo "Exiting ..........";
  echo –e ${clear};
  sleep 3;
  exit 1;
}

# function contin prompts user to continue to next
contin() {
  echo
  read –p "Continue (Y/y)? " –n 1 –r;
```

```
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo
    echo -e ${green};
    echo "Exiting ..........";
    echo -e ${clear};
    sleep 3;
    exit 0;
  fi
}


# function remove_hidden - removes all hiddent files and directories

remove_hidden() {
  cd $1;
  if [[ -n $(find . -mindepth 1 -name '.*') ]]; then
    echo "Hidden files found";
    find . -mindepth 1 -name '.*';
    echo
    echo "Removing hidden files";
    rm -rf .* 2> /dev/null;
  else
    echo
    echo "No hidden files found";
  fi
  cd $rbn
}

# function check empty checks there are no files or directories present
check_empty() {
  if [ -n "$(find $1 -maxdepth 0 -empty 2> /dev/null)" ]; then
    status=$emp;
  else
    status=$fil;
  fi
}

report_files () {
  cd $1
  file_list=`ls *.* 2> /dev/null`
  file_count=0;
  for file in $file_list; do
    file_list+=("$file");
    file_count+=1;
  done
  if [ $file_count -gt 0 ]; then
    echo
    echo "Files in $1:"
    for file in $file_list; do
      echo $file
    done
```

```
  else
    echo
    echo "No files found"
  fi
  cd $rbn
}


###
# Display run notice
###

echo -e ${blue};
clear
echo
echo "     ***************"
echo "     * FILE REPORT *"
echo "     ***************"
echo
echo "     Displays the disposition of files in render process directories."
echo "     and deletes all hidden files";
echo
echo -e ${clear};

contin

cd ~/render_bin

###
# Report disposition of directories
###

# Report disposition of drop box

echo
echo "Reporting on Drop box ..........";

remove_hidden $dbx
report_files $dbx
contin
echo

# Report disposition of staging

echo
echo "Reporting on Staging ..........";

remove_hidden $stg
report_files $stg
contin
echo
```

```
# Report disposition of prod render

echo
echo "Reporting on Render ..........";

remove_hidden $prd
report_files $prd
contin
echo

# Report disposition of images

echo
echo "Reporting on Images ..........";

remove_hidden $pim
check_empty $pim
if [[ "$status" == "$fil" ]]; then
# compile list of image sequences
  last_seq=" ";
  last_frm=" ";
  cd $pim
  ls *.png | sort -t'-' -k1,1 -k2,2 | while read imgfile; do
    seq=$(echo $imgfile | cut -d'-' -f 1)
    frm=$(echo $imgfile | cut -d'-' -f 2)
    if [[ "$seq" == "$last_seq" ]]; then
      last_frm=$frm
    else
      last_seq=$seq
      echo "$seq-####" >> /home/master/tmp/seqs.txt
    fi
  done
# Recover sequences from external file
  tmp_seqs="$tmp/seqs.txt"
  while read line; do
     img_seqs+="${line} ";
  done < $tmp_seqs
  rm "$tmp/seqs.txt";

# display pick list
  echo
  echo "Image sequences in $pim:"
  for file in $img_seqs; do
    echo $file
  done
else
  echo
  echo "No files found"
fi
contin
```

```
echo

# Report disposition of compositing

echo
echo "Reporting on Compositing ..........";

remove_hidden $com
check_empty $com
if [[ "$status" == "$fil" ]]; then
# compile list of image sequences
  last_seq=" ";
  last_frm=" ";
  cd $com
  ls *.png | sort -t'-' -k1,1 -k2,2 | while read imgfile; do
    seq=$(echo $imgfile | cut -d'-' -f 1)
    frm=$(echo $imgfile | cut -d'-' -f 2)
    if [[ "$seq" == "$last_seq" ]]; then
      last_frm=$frm
    else
      last_seq=$seq
      echo "$seq-####" >> /home/master/tmp/seqs.txt
    fi
  done
# Recover sequences from external file
  tmp_seqs="$tmp/seqs.txt"
  while read line; do
     img_seqs+="${line} ";
  done < $tmp_seqs
  rm "$tmp/seqs.txt";

# display pick list
  echo
  echo "Image sequences in $com:"
  for file in $img_seqs; do
    echo $file
  done
else
  echo
  echo "No files found"
fi
contin
echo

# Report disposition of Pickupbox

echo
echo "Reporting on Pickup box ..........";

remove_hidden $pbx
report_files $pbx
```

```
contin
echo

echo
echo "All render process directories reported.";
echo
echo –e ${green};
echo "Exiting ..........";
echo –e ${clear};
sleep 3;

exit 0
```

preprocess.sh

```
#!/bin/bash

# preprocess.sh - applies mandatory and options render settings

#   run using ./ not sh

# declare constants
declare -r unk="unknown";
declare -r fil="files";
declare -r emp="empty";
declare -r ntf="not_found";
declare -r prd="/home/master/prod/prod_render";
declare -r trd="/home/master/test/test_render";
declare -r dbx="/home/master/drop_box";
declare -r stg="/home/master/staging";
declare -r org="Original";
declare -r hiq="High quality";
declare -r frt="Fast render";

# Declare arrays

declare -a blender_files;
declare -i file_count;
declare -x infile;
declare -x outfile;
declare -a options=("$org" "$hiq" "$frt")

# set colour variables
green="\033[0;32m"
yellow="\033[0;33m"
blue="\033[0;34m"
clear="\033[0m"

# define user functions

# function error_exit checks return code and exits
error_exit () {
  echo
  echo "$1"
  echo -e ${yellow};
  echo "Exiting ..........";
  echo -e ${clear};
  sleep 3;
  exit 1;
}

# function confirm prompts user to confirm input or selection
confirm() {
  echo
```

```
  read -p "Confirm (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    error_exit "No confirmation";
  fi
}

# function contin prompts user to continue to next
contin() {
  echo
  read -p "Continue (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo
    echo -e ${green};
    echo "Exiting ..........";
    echo -e ${clear};
    exit 0;
  fi
}

# function check empty checks there are no files or directories present
check_empty() {
  if [ -n "$(find $1 -maxdepth 0 -empty 2> /dev/null)" ]; then
    status=$emp;
  else
    status=$fil;
  fi
}

# funtion menu displays a pick list of items in an array
menu() {
  select item ; do
  if [ 1 -le "$REPLY" ] && [ "$REPLY" -le $# ]; then
    echo "$item selected";
    echo
    break;
  else
    echo "Select by a number from 1-$#";
  fi
done
}

# function move_file uses cp-rm - moves file (1) from source (2) to dest (3)
move_file() {
  if [ -f "$3/$1" ]; then
    error_exit "Duplicate file $infile";
  else
    if cp "$2/$1" "$3"; then
      rm "$2/$1";
    fi
  fi
}
```

```
remove_hidden() {
  cd $1;
  if [[ -n $(find . -mindepth 1 -name '.*') ]]; then
    echo "Hidden files found";
    find . -mindepth 1 -name '.*';
    echo
    echo "Removing hidden files";
    rm -rf .* 2> /dev/null;
  else
    echo
    echo "No hidden files found";
  fi
  cd $2;
}

# function delete_bakfile - deleted blender .blend1 backup file
delete_1file() {
 bakfile="${1}1";
 if test -f "/home/master/staging/$bakfile"; then
   echo  "deleting $bakfile"
   rm "/home/master/staging/$bakfile";
 fi
}

###
# Display run notice
###

echo -e ${blue};
clear
echo
echo "     *****************"
echo "     * PREPROCESSING *"
echo "     *****************"
echo
echo "     Applies frame allocation and quality control render settings."
echo
echo -e ${clear};

# Delete any spurious hidden files

echo
echo "Deleting spurious and hidden files ..........";
echo

remove_hidden $dbx $rbn;

remove_hidden $stg $rbn;
```

```
###
# Get Blender files from drop_box directory
###

cd $rbn;

status=$unk

check_empty $dbx

if [ "$status" = "$emp" ]; then
  error_exit "$dbx status is $status";
else
  echo "$dbx status is $status";
fi

cd $dbx

file_list=`ls *.blend`

cd $rbn

# Copy files from string to array

file_count=0;
for file in $file_list; do
  blender_files+=("$file");
  file_count+=1;
done

if [ "$file_count" -eq 0 ]; then
  error_exit "No Blender file to preprocess.";
fi

###
# Pick Blender file to be preprocessed
###

# Display available Blender files and confirm render

echo
echo "Blender files available for preprocessing:";
echo "------------------------------------------";
for file in "${blender_files[@]}"; do
  echo $file;
done

contin

# Display a pick list of render files
```

```
echo
echo
echo "Select input file:";
echo "-------------------";

menu "${blender_files[@]}"

infile=$item;

###
# Move selected file from drop_box to staging
###

move_file $infile $dbx $stg

###
# Apply render settings
###

# Apply mandatory frame allocation settings

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

# delete any blender backup file

delete_1file $infile

echo
echo "$infile has Frame Allocation settings";

# Display a pick list of optional settings

echo
echo "Select optional settings:";
echo "-------------------------";

menu "${options[@]}"

set=$item;

# Apply selected settings

case $set in

  $org )
  echo
  echo "$infile has $org settings"
  ;;

  $hiq )
  temp_file=$infile
```

```
  blender -b ~/staging/$infile -P ~/render_bin/set_hiq_rp.py
  infile="h_$temp_file";
  if [ -f "$stg/$temp_file" ]; then
    rm "$stg/$temp_file";
    if [ -f "$stg/$temp_file"1 ]; then
       rm "$stg/$temp_file"1;
    fi
  fi
  echo
  echo "$infile has $hiq settings"
  ;;

  $frt )
  temp_file=$infile
  blender -b ~/staging/$infile -P ~/render_bin/set_frt_rp.py
  infile="f_$temp_file";
  if [ -f "$stg/$temp_file" ]; then
    rm "$stg/$temp_file";
    if [ -f "$stg/$temp_file"1 ]; then
      rm "$stg/$temp_file"1;
    fi
  fi
  echo
  echo "$infile has $frt settings"
  ;;
esac

delete_1file $infile

# Prompt for optimal CPU settings

echo
read -p "Apply optimal CPU settings (Y/y)? " -n 1 -r;
if [[ $REPLY =~ ^[Yy]$ ]]; then
  temp_file=$infile
  blender -b ~/staging/$infile -P ~/render_bin/set_cpu_rp.py
  infile="c_$temp_file";
  if [ -f "$stg/$temp_file" ]; then
    rm "$stg/$temp_file";
    if [ -f "$stg/$temp_file"1 ]; then
      rm "$stg/$temp_file"1;
    fi
  fi
  echo
  echo "$infile has optimal CPU settings"
fi

delete_1file $infile

echo
echo "$infile is ready for rendering";
```

```
echo
echo –e ${green};
echo "Exiting ..........";
echo –e ${clear};
sleep 3;

exit 0;
```

max_cpu.sh

```
#!/bin/bash

# max_cpu.sh - initiates a cpu render on all available render hosts

#   run using ./ not sh

# Declare constants, variables and arrays

declare -r unk="unknown";
declare -r fil="files";
declare -r emp="empty";
declare -r ntf="not_found";
declare -r prd="/home/master/prod/prod_render";
declare -r stg="/home/master/staging";
declare -a blender_files;
declare -i file_count;
declare -x infile;
declare -x outfile;
declare -a render_hosts;
declare -a active_hosts;
declare -i active_count;


# Initialise array of all render host IP addresses

render_hosts[0]="192.168.0.201";
render_hosts[1]="192.168.0.202";
render_hosts[2]="192.168.0.203";
render_hosts[3]="192.168.0.204";
render_hosts[4]="192.168.0.205";
render_hosts[5]="192.168.0.206";
render_hosts[6]="192.168.0.207";

# set colour variables
green="\033[0;32m"
yellow="\033[0;33m"
blue="\033[0;34m"
clear="\033[0m"

# define user functions

# function error_exit checks return code and exits
error_exit () {
  echo
  echo "$1"
  echo -e ${yellow};
  echo "Exiting ..........";
  echo -e ${clear};
  sleep 3;
```

```
  exit 1;
}

# function confirm prompts user to confirm input or selection
confirm() {
  echo
  read -p "Confirm (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    error_exit "No confirmation";
  fi
}

# function contin prompts user to continue to next
contin() {
  echo
  read -p "Continue (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo
    echo -e ${green};
    echo "Exiting ..........";
    echo -e ${clear};
    exit 0;
  fi
}

# function check empty checks there are no files or directories present
check_empty() {
  if [ -n "$(find $1 -maxdepth 0 -empty 2> /dev/null)" ]; then
    status=$emp;
  else
    status=$fil;
  fi
}

# funtion menu displays a pick list of items in an array
menu() {
  select item ; do
  if [ 1 -le "$REPLY" ] && [ "$REPLY" -le $# ]; then
    echo "$item selected";
    echo
    break;
  else
    echo "Select by a number from 1-$#";
  fi
done
}

# function remove_hidden - removes all hiddent files and directories

remove_hidden() {
  cd $1;
```

```
  if [[ -n $(find . -mindepth 1 -name '.*') ]]; then
    echo "Hidden files found";
    find . -mindepth 1 -name '.*';
    echo
    echo "Removing hidden files";
    rm -rf .* 2> /dev/null;
  else
    echo
    echo "No hidden files found";
  fi
  cd $2
}

###
# Display run notice
###

echo -e ${blue};
clear
echo
echo "     **************"
echo "     * CPU RENDER *"
echo "     **************"
echo
echo "     Renders the selected Blender file using all active hosts."
echo
echo -e ${clear};
contin

# Delete any spurious hidden files

echo
echo "Deleting spurious and hidden files ..........";
echo

remove_hidden $stg $rbn;

remove_hidden $prd $rbn;


echo
echo "Searching network for active hosts";
echo "-----------------------------------";

# Ping hosts and if active add to active hosts array
active_count=0
for ip in ${render_hosts[@]}; do
  echo
  echo $ip;
  echo "-----------------------------------";
  if ping -c 1 -W 1 $ip; then
```

```
    echo "$ip is alive";
    active_hosts[${#active_hosts[@]}]="${ip}";
    active_count+=1;
  else
    echo
    echo "$ip is down";
  fi
done
echo
if [ "$active_count" -eq 0 ]; then
  error_exit "No active render hosts.";
fi

# list active hosts

echo
echo "Active hosts:";
echo "-------------";

for val in ${active_hosts[@]}; do
  echo "$val ready to render";
done

confirm

###
# Get Blender files from staging directory
###

# check if staging directory is empty

check_empty $stg

if [ "$status" = "$emp" ]; then
  error_exit "$stg status is $status";
else
  echo "$stg status is $status";
fi

# Read all staged Blender files into a string

cd $stg

file_list=`ls *.blend 2> /dev/null`

cd $rnd

# Copy files from string to array

file_count=0;
for file in $file_list; do
```

```
  blender_files+=("$file");
  file_count+=1;
done

if [ "$file_count" -eq 0 ]; then
  error_exit "No Blender file to render.";
fi

###
# Pick Blender file to be rendered and get output image filespec
###

# Display available Blender files and confirm render

echo
echo "Blender files available for render:";
echo "-----------------------------------";
for file in "${blender_files[@]}"; do
  echo $file;
done

contin

# Display a pick list of render files

#clear;
echo
echo "Select input file:";
echo "------------------";

menu "${blender_files[@]}"

infile=$item;

# Promt for output file

echo "Enter job number:";
echo "-----------------";
echo
echo "Job number is used as the output image file-spec."
echo
jobnum=""
while :
do
  echo
  read -p "Job number:" jobnum;
  if [ -z "$jobnum" ]; then
    error_exit "Image file not specified.";
  else
    if [[ $jobnum =~ "-" ]]  || [[ $jobnum =~ "#" ]] || [[ $jobnum =~ " " ]];
then
```

```
      echo "Job number can't have ' ','-' or '#'.  Reserved for file spec.";
    else
      break;
    fi
  fi
done

outfile="${jobnum}-####"

echo
echo "Input file is $infile";
echo
echo "Output file is $outfile";

confirm

# Copy infile to production render directory

if test -f "/home/master/prod/prod_render/$infile"; then
  error_exit "Duplicate $infile";
else
  cp ~/staging/"$infile" ~/prod/prod_render/;
  rm ~/staging/"$infile";
fi

###
# Get target render device type
###

render_device="cpu";

# set standard host user and bin directory

host_user="tracer";
bin_dir="/home/tracer/render_bin";

cd /home/master/render_bin

###
# Choose render script
###

#render_script="network_test.sh";
#render_script="prod_ren.sh";
render_script="cpu_prod_ren.sh";

# Set redirect outputs and remote host command

redirect_out="${render_device}_render.out"
redirect_err="${render_device}_render.err"
```

```
redirect="> ~/redirect_logs/$redirect_out 2> ~/redirect_logs/$redirect_err </dev/
null &"
#render_command="sh $render_script -i $infile -o $outfile \
# > ~/redirect_logs/$redirect_out 2> ~/redirect_logs/$redirect_err </dev/null &"
render_command="sh ${render_script} -i ${infile} -o ${outfile} ${redirect}"

###
# Initiate render on all available render hosts
###

echo "Initiating $render_device remote render.....................";
echo

for ip in ${active_hosts[@]}; do
  render_host="${host_user}@${ip}";
  echo
  echo "Starting  $render_host ................";
  echo
  ssh $render_host << EOF
    cd /home/tracer/redirect_logs
    [[ -f $redirect_out ]] && rm $redirect_out
    [[ -f $redirect_err ]] && rm $redirect_err
    cd /home/tracer/render_bin
    nohup $render_command
    disown -h
    exit
EOF
done

echo
echo "Render job $jobnum initiated at " `date`;
echo
echo "Blender output is redirected to /home/tracer/redirect_logs/$redirect_out"
echo
echo -e ${green};
echo "Exiting ..........";
echo -e ${clear};
sleep 3;

exit 0;
```

move_images.sh

```
#!/bin/bash

# move_images.sh – moves an image sequence to compositing directory

#   run using ./ not sh

# declare constants and arrays

declare –r unk="unknown";
declare –r fil="files";
declare –r emp="empty";
declare –r ntf="not_found";
declare –r rbn="/home/master/render_bin";
declare –r com="/home/master/compositing";
declare –r pim="/home/master/prod/prod_render/prod_images";
declare –r tim="/home/master/test/test_render/test_images";
declare –r tmp="/home/master/tmp";

declare –a img_seqs;

# set colour variables
green="\033[0;32m"
yellow="\033[0;33m"
blue="\033[0;34m"
clear="\033[0m"

# define user functions

# function error_exit checks return code and exits
error_exit () {
  echo
  echo "$1"
  echo –e ${yellow};
  echo "Exiting ..........";
  echo –e ${clear};
  sleep 3;
  exit 1;
}

# function confirm prompts user to confirm input or selection
confirm() {
  echo
  read –p "Confirm (Y/y)? " –n 1 –r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    error_exit "No confirmation";
  fi
}
```

```
# function contin prompts user to continue to next
contin() {
  echo
  read -p "Continue (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo
    echo -e ${green};
    echo "Exiting ..........";
    echo -e ${clear};
    exit 0;
  fi
}

# funtion menu displays a pick list of items in an array
menu() {
  select item ; do
  if [ 1 -le "$REPLY" ] && [ "$REPLY" -le $# ]; then
    echo "$item selected";
    echo
    break;
  else
    echo "Select by a number from 1-$#";
  fi
done
}

# function check empty checks there are no files or directories present
check_empty() {
  if [ -n "$(find $1 -maxdepth 0 -empty 2> /dev/null)" ]; then
    status=$emp;
  else
    status=$fil;
  fi
}

# function delete_spurious - deletes spurious files from a directory
# Note1: Spurious files are files that will have a determinetal effect on Blender
# when loading images for compositing.  Spurious files include empty files
# and hidden .DS_Store files generated by file manager applications such as
Finder.
# Note2: Although potentially dangerous the '*' method is necessary because
# suffixes are added to multiple files e.g ".DS_Store" "._.DS_Store"...

delete_spurious () {
cd $1
`find . -name '*.DS_Store' -type f -delete`;
cd $2;
}

# function remove_hidden - removes all hiddent files and directories
```

```
remove_hidden() {
  cd $1;
  if [[ -n $(find . -mindepth 1 -name '.*') ]]; then
    echo "Hidden files found";
    find . -mindepth 1 -name '.*';
    echo
    echo "Removing hidden files";
    rm -rf .* 2> /dev/null;
  else
    echo
    echo "No hidden files found";
  fi
  cd $2
}

# function move_file uses cp-rm - moves file (1) from source (2) to dest (3)
move_file() {
  if [ -f "$3/$1" ]; then
    error_exit "Duplicate file $infile";
  else
    if cp "$2/$1" "$3"; then
      rm "$2/$1";
    fi
  fi
}

###
# Display run notice
###

echo -e ${blue};
clear
echo
echo "     ***************"
echo "     * MOVE IMAGES *"
echo "     ***************"
echo
echo "     Moves an image sequence to compositing directory."
echo "     and deletes all hidden files"
echo
echo -e ${clear};
contin

###
# Check status of source and destination directories
###

cd ~/render_bin;

# Delete any spurious hidden files
```

```
echo
echo "Deleting spurious and hidden files ..........";
echo

delete_spurious $pim $rbn;

remove_hidden $pim $rbn;

delete_spurious $com $rbn;

remove_hidden $com $rbn;

echo
echo "Checking compositing directory status ..........";
echo

# Check if compositing directory is empty

status="$unk";

check_empty "$com";

if [ "$status" = "$fil" ]; then
  error_exit "$com status is $status";
else
  echo "$com status is $status";
fi

echo
echo "Checking image directory status ..........";
echo

# Check if image directory has files

status="$unk";

check_empty "$pim";

if [ "$status" = "$emp" ]; then
  error_exit "$pim status is $status";
else
  echo "$pim status is $status";
fi


###
# Display a pick lst of image sequences
###

echo
```

```
echo "Compiling list of image sequences.........";
echo

last_seq=" ";
last_frm=" ";

cd $pim
ls *.png | sort -t'-' -k1,1 -k2,2 | while read imgfile; do
  seq=$(echo $imgfile | cut -d'-' -f 1)
  frm=$(echo $imgfile | cut -d'-' -f 2)
  if [[ "$seq" == "$last_seq" ]]; then
    last_frm=$frm
  else
    last_seq=$seq
    echo "$seq-####" >> /home/master/tmp/seqs.txt
  fi
done

# Recover sequences from external file

tmp_seqs="$tmp/seqs.txt"
while read line; do
  img_seqs[${#img_seqs[@]}]="${line}";
done < $tmp_seqs

rm "$tmp/seqs.txt";

# display pick list

echo
echo "Select image sequence for transfer:";
echo "-----------------------------------";

menu "${img_seqs[@]}"

seq=$item;

###
# Transfer image sequence
###

# Confirm transfer

echo
echo "Transfer image sequence $seq";

confirm

# Transfer image sequence

jobnum=$(echo $seq | cut -d'-' -f 1)
```

```
echo
echo
echo "Transfering image files ..........";
echo

###
# Move selected image sequence from image directory to compositing
###

ls $jobnum*.* | while read img_file; do
  move_file $img_file $pim $com;
done
echo "Image count is: "; ls $com | wc -l;
echo
echo
echo "$seq ready for compositing.";
echo
echo -e ${green};
echo "Exiting ..........";
echo -e ${clear};
sleep 3;

exit 0;
```

archive_render_file.sh

```
#!/bin/bash

# archive_render_file.sh - moves render files to for_archive directory

#   run using ./ not sh

# declare constants, variables and arrays

declare -r unk="unknown";
declare -r fil="files";
declare -r emp="empty";
declare -r ntf="not_found";
declare -r rbn="/home/master/render_bin";
declare -r tmp="/home/master/tmp";
declare -r prd="/home/master/prod/prod_render/"
declare -r far="/home/master/for_archive";
declare -a blender_files;

# set colour variables
green="\033[0;32m"
yellow="\033[0;33m"
blue="\033[0;34m"
clear="\033[0m"

# define user functions

# function error_exit checks return code and exits
error_exit () {
  echo
  echo "$1"
  echo -e ${yellow};
  echo "Exiting ..........";
  echo -e ${clear};
  sleep 3;
  exit 1;
}

# function confirm prompts user to confirm input or selection
confirm() {
  echo
  read -p "Confirm (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    error_exit "No confirmation";
  fi
}

# function contin prompts user to continue to next
contin() {
  echo
```

```
  read -p "Continue (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo
    echo -e ${green};
    echo "Exiting ..........";
    echo -e ${clear};
    exit 0;
  fi
}

# funtion menu displays a pick list of items in an array
menu() {
  select item ; do
  if [ 1 -le "$REPLY" ] && [ "$REPLY" -le $# ]; then
    echo "$item selected";
    echo
    break;
  else
    echo "Select by a number from 1-$#";
  fi
done
}

# function check empty checks there are no files or directories present
check_empty() {
  if [ -n "$(find $1 -maxdepth 0 -empty 2> /dev/null)" ]; then
    status=$emp;
  else
    status=$fil;
  fi
}

# function delete_spurious - deletes spurious files from a directory
# Note1: Spurious files are files that will have a determinetal effect on Blender
# when loading images for compositing.  Spurious files include empty files
# and hidden .DS_Store files generated by file manager applications such as
Finder.
# Note2: Although potentially dangerous the '*' method is necessary because
# suffixes are added to multiple files e.g ".DS_Store" "._.DS_Store"...

delete_spurious () {
cd $1
`find . -name '*.DS_Store' -type f -delete`;
cd $2;
}

# function remove_hidden - removes all hiddent files and directories

remove_hidden() {
  cd $1;
  if [[ -n $(find . -mindepth 1 -name '.*') ]]; then
```

```
    echo "Hidden files found";
    find . -mindepth 1 -name '.*';
    echo
    echo "Removing hidden files";
    rm -rf .* 2> /dev/null;
  else
    echo
    echo "No hidden files found";
  fi
  cd $2
}

# function move_file uses cp-rm - moves file (1) from source (2) to dest (3)
move_file() {
  if [ -f "$3/$1" ]; then
    error_exit "Duplicate file $infile";
  else
     if cp "$2/$1" "$3"; then
       rm "$2/$1";
     fi
fi
}

###
# Display run notice
###

echo -e ${blue};
clear
echo
echo "     ************************"
echo "     * ARCHIVE RENDER FILE *"
echo "     ************************"
echo
echo "     Moves selected Blender file to for_archive directory."
echo "     and deletes all hidden files"
echo
echo -e ${clear};
contin

cd ~/render_bin;

# Delete any spurious hidden files

echo
echo "Deleting spurious and hidden files ..........";
echo

remove_hidden $prd $rbn;

remove_hidden $far $rbn;
```

```
###
# Check status of source directory
###

echo
echo "Checking Render directory status ..........";
echo

status="$unk";

check_empty "$prd";

if [ "$status" = "$emp" ]; then
  error_exit "$prd status is $status";
else
  echo "$prd status is $status";
fi

###
# Display of a pick list of render files
###


# read files in render directory

cd $prd

file_list=`ls *.blend 2> /dev/null`;

cd $rbn

# Copy files from sting to array

file_count=0;
for file in $file_list; do
  blender_files+=("$file");
  file_count+=1;
done

if [ "$file_count" -eq 0 ]; then
  error_exit "No Blender files to move";
fi

# Display available Blender files and confirm archive

echo
echo "Blender files available for archive:";
echo "-----------------------------------";
for file in "${blender_files[@]}"; do
  echo $file;
```

```
done

contin


# pick file to move

echo
echo
echo "Select file to move:";
echo "--------------------";

menu "${blender_files[@]}"

mov_file=$item;

# move selectd file

move_file $mov_file $prd $far

echo
echo "$mov_file moved to $far";
echo
echo -e ${green};
echo "Exiting ..........";
echo -e ${clear};
sleep 3;

exit 0
```

archiv_images.sh

```
#!/bin/bash

# archive_images.sh - consolidates, ompresses and moves image files to
for_archive directory

#   run using ./ not sh

# declare constants, variables and arrays

declare -r unk="unknown";
declare -r fil="files";
declare -r emp="empty";
declare -r ntf="not_found";
declare -r rbn="/home/master/render_bin";
declare -r com="/home/master/compositing";
declare -r tmp="/home/master/tmp";
declare -r far="/home/master/for_archive";
declare -a blender_files;

# set colour variables
green="\033[0;32m"
yellow="\033[0;33m"
blue="\033[0;34m"
clear="\033[0m"

# define user functions

# function error_exit checks return code and exits
error_exit () {
  echo
  echo "$1"
  echo -e ${yellow};
  echo "Exiting ..........";
  echo -e ${clear};
  sleep 3;
  exit 1;
}

# function confirm prompts user to confirm input or selection
confirm() {
  echo
  read -p "Confirm (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    error_exit "No confirmation";
  fi
}

# function contin prompts user to continue to next
contin() {
```

```
  echo
  read -p "Continue (Y/y)? " -n 1 -r;
  if [[ ! $REPLY =~ ^[Yy]$ ]]; then
    echo
    echo -e ${green};
    echo "Exiting ..........";
    echo -e ${clear};
    exit 0;
  fi
}

# function check empty checks there are no files or directories present
check_empty() {
  if [ -n "$(find $1 -maxdepth 0 -empty 2> /dev/null)" ]; then
    status=$emp;
  else
    status=$fil;
  fi
}

# function delete_spurious - deletes spurious files from a directory
# Note1: Spurious files are files that will have a determinetal effect on Blender
# when loading images for compositing.  Spurious files include empty files
# and hidden .DS_Store files generated by file manager applications such as
Finder.
# Note2: Although potentially dangerous the '*' method is necessary because
# suffixes are added to multiple files e.g ".DS_Store" "._.DS_Store"...

delete_spurious () {
cd $1
`find . -name '*.DS_Store' -type f -delete`;
cd $2;
}

# function remove_hidden - removes all hiddent files and directories

remove_hidden() {
  cd $1;
  if [[ -n $(find . -mindepth 1 -name '.*') ]]; then
    echo "Hidden files found";
    find . -mindepth 1 -name '.*';
    echo
    echo "Removing hidden files";
    rm -rf .* 2> /dev/null;
  else
    echo
    echo "No hidden files found";
  fi
  cd $2
}
```

```
# function move_file uses cp-rm - moves file (1) from source (2) to dest (3)
move_file() {
  if [ -f "$3/$1" ]; then
    error_exit "Duplicate file $infile";
  else
     if cp "$2/$1" "$3"; then
       rm "$2/$1";
     fi
fi
}

# function tar_files - compresses all files in a directory to a tar archive
tar_files() {
  cd $1
  seq_name=`ls -1 | head -n1`;
  tar_file=$(echo $seq_name | cut -d'-' -f1);
  tar -zcvf "${tar_file}.tar.gz" *;
  tar_file="${tar_file}.tar.gz";
  cd $rbn
}

# function clear_directory - deletes all files from a directory

clear_directory() {
  cd /home/master/compositing;
  rm -r $tar_name*.*;
  cd $rbn
}

###
# Display run notice
###

echo -e ${blue};
clear
echo
echo "     ******************"
echo "     * ARCHIVE IMAGES *"
echo "     ******************"
echo
echo "     Deletes all hidden files in compositing and for_archive directories,"
echo "     Consolidates (tar) all files,"
echo "     compresses (gz) all files,";
echo "     moves compressed file to for-archive directory,"
echo "     and deletes all iage files from compositing directory"
echo
echo -e ${clear};
contin

cd ~/render_bin;
```

```
# Delete any spurious hidden files

echo
echo "Deleting spurious and hidden files ..........";
echo

remove_hidden $com $rbn;

remove_hidden $far $rbn;

###
# Check status of source directory
###

echo
echo "Checking Compositing directory status ..........";
echo

status="$unk";

check_empty "$com";

if [ "$status" = "$emp" ]; then
  error_exit "$com status is $status";
else
  echo "$com status is $status";
fi

###
# Compress all files in compositing directory
###

tar_files $com;

# move compressed file

move_file $tar_file $com $far;

# clear out compositing directory

clear_directory;

echo
echo "Moved $tar_file to $far";
echo
echo -e ${green};
echo "Exiting ..........";
echo -e ${clear};
sleep 3

exit 0
```

Set Permissions

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

1 - `$ chmod +x <scriptname>`

Deployment Options

DEDICATED NETWORK, USB STICK

Dedicated Network

The ideal platform for the render farm is a dedicated local area network, consisting of identical or at least comparable hosts, each with a high performance CPU and GPU. The software could be manually installed on each host using the software installation guide. An alternative to manual installation is to use an image cloning utility such as FOG. After an initial manual installation of a master host and a render host, the cloning utility can be used to create an image of each installation. Additional hosts can be configured by writing the image to a hosts storage media and then manually changing the hostname and IP address to be unique.

'Home-alone Render - the progressive building of processing power on a network of dedicated hosts.'

USB Stick

A less than ideal but useful platform for the render farm is to use a bootable Linux USB to utilise non-dedicated machines on a local area network. Most PCs can have their boot sequence changed to initially look for a USB with a boot sector. If present, the PC will boot the operating system on the USB which will then have control of the CPU, memory and devices. The USB can also serve as the main storage media and bypass the built-in storage devices. An Internet search for "bootable Ubuntu Linux USB" will result in several options for creating a bootable USB with Ubuntu 20.4. Once created, the USB can be used to boot any available PC and the render farm software for a master host and a render host can be manually installed. Additional hosts can be configured by cloning the USB and then manually changing the hostname and IP address to be unique. Clearly there are limitations. All PCs must be on the same LAN segment but USB 3 is fast enough to be viable and a high capacity USB media will be sufficient as a storage device. Only Blender background CPU based rendering will be viable in many cases.

'Coup Render - the quick and successful overtake of the processing power on a network of general purpose hosts.'

Workstation Configuration

Workstations need to access the drop box and pickup box shared directories on the master host to submit jobs and collect results. To do this they must be in the RENDERFARM workgroup and use an SMB client compatible with Samba. It's useful to be able to 'ping' all machines that use the render farm or identify their LAN activity by hostname. A static IP address simplifies this. The IP address and MAC address can be registered with the Gateway/router and included in hosts files. For a Linux workstation, use the same instructions as the hosts, i.e. Set Hostname, Obtain MAC Address and Set Static IP Address. Use Nautilus file manager to connect. Detailing the configuration for all potential workstation operating systems is beyond the scope of this instructions however MAC OSX is compatible and the basic steps are provided. Windows is able to connect however some compatibility issues may occur.

OSX

The following example is the steps for a MAC workstation 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 - Set to `workstation-1`

Set Group Name

6 - Set to RENDERFARM

WINDOWS

The details may differ from one Windows version to the next so only the basic steps are provided.

Configure Static IP Address

1 - Control panel -> Network and Internet -> Change Adapter Settings - Local Area -> Properties

Obtain MAC Address

2 - Start menu -> Cmd -> ipconfig /all

Install TightVNC VNC Server

3 - Download TightVNC installer

4 - Install TightVNC

Set Hostname

5 - Control panel -> Change Settings -> Computer name

Enable SMB/CIFS

5 - Control panel -> Programs -> Turn Windows Features On -> SMB 1.0/CIFS -> check SMB/CIFS client

Map Network Drive to Samba Shares

6 - File Explorer -> This PC -> Computer -> Map network drive
choose a drive letter
browse available shares or enter share drive address i.e. \\192.168.0.3\drop_box ..pickup_box
check Reconnect at sign-in (logon) and Connect using different credentials

Set Group Name

7 - Control panel -> System -> Advanced system settings -> Computer name - > Change -> Workgroup change to RENDERFARM

Acceptance Test

PRELIMINARY

All hosts builds complete

All software installation complete

VNC service, Samba shared directories, Samba client and SSH connections tested and working.

Test Harness

Prepare a Blender animation file ready for rendering with at least 20 animation frames and with Output Properties->Overwrite 'checked' and Output Properties->Placeholder 'unchecked'. All image and texture resources must be packed into .blend file.

Save 5test.blend with 5 frames to render.
Save 10atest.blend with 10 frames to render.
Save 10btest.blend with 10 frame to render.
Save 10ctest.blend with 10 frames to render.
Save 10dtest.blend with 10 frames to render
Save 20test.blend with 20 frames to render.

Copy files to USB media.

Process Directories

Prepare master host and render host directories.

Master: All master scripts in /home/master/render_bin and all other process directories empty.

Render: All render scripts in /home/tracer/render_bin and all other process directories empty.

Installation Test

Activate the master host and copy test files to master host /home/master/drop_box directory using USB media.

Open terminal and cd to render_bin.

Test 1 - render control menu

1. run script ./render_control.sh
2. continue to menu
3. select 'Quit' to exit

exit with no action

Test 2 - files - continues on from Test 1

1. run script ./render_control.sh
2. continue to menu
3. select 'Files' option
4. continue to file report
5. continue through all reports

6. continue to exit

exit with hidden files removed in all process directories

Test 3 - preprocess - continues on from Test 2

1. run script ./render_control.sh
2. continue to menu
3. select 'Preprocess' option
4. continue to preprocessing
5. select 5test.blend
6. select 'original'
7. bypass 'CPU'

exit with 5test.blend moved to staging with frame allocation settings

8 repeat with following optional settings

- 10atest.blend with 'original' and 'CPU' settings -> c_10atest.blend
- 10btest.blend with 'high quality' settings -> h_10btest.blend
- 10ctest.blend with 'fast render' settings -> f_10ctest.blend
- 10dtest.blend with 'high quality' and 'CPU' settings -> c_h_10dtest.blend
- 20test.blend with 'original settings -> 20test.blend

Test 4 - render - continues on from Test 3

Activate all available render hosts

1. run script ./render_control.sh
2. continue to menu
3. select 'Render' option
4. continue to render
5. confirm available render hosts
6. select 20test.blend and enter job no. 'j123'

exit with active render and wait for completion

6. repeat with 10atest.blend and enter job no. j345

Test 5 - move images - continues on from Test 4

1. run script ./render_control.sh
2. continue to menu
3. select 'Move-Images' option
4. continue to move images
5. select j123 image sequence
6. continue to exit

exit with j123 seq moved to compositing

Test 6 - archive render file - continues on from Test 5

1. run script ./render_control.sh
2. continue to menu
3. select 'Archive-File' option

4. continue to archive file
4. select 20test.blend
5. continue to exit

exit with 20test.blend moved to for_archive

Test 7 - archive images - continues on from Test 6

1. run script ./render_control.sh
2. continue to menu
3. select 'Archive-Images' option
4. continue to archive images
5. observe compression of j123 sequence
5. continue to exit

exit with j123.tar.gz moved to for_archive and compositing directory empty

Test 8 - information - run any time

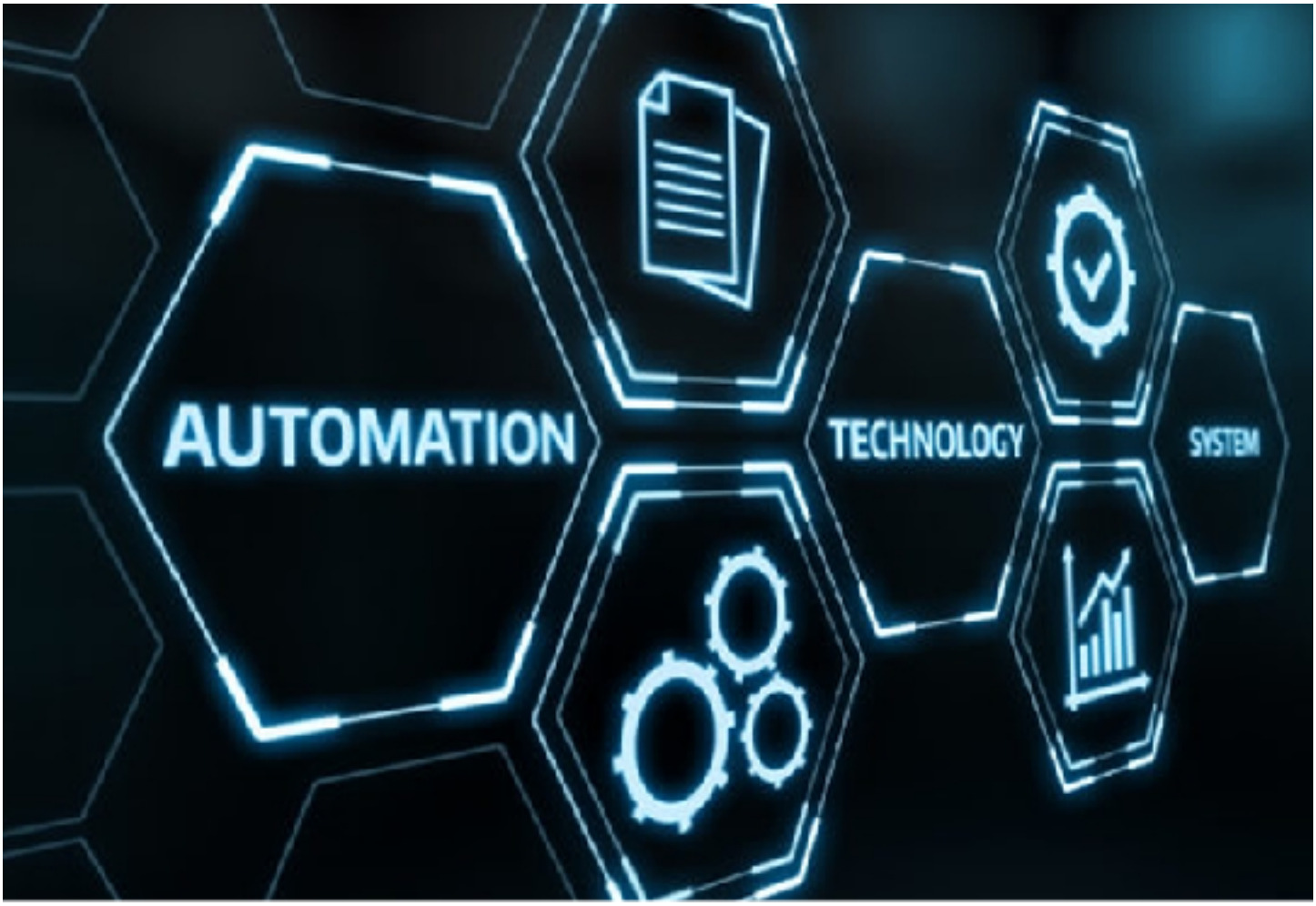
1. run script ./render_control.sh
2. continue to menu
3. select 'Information' option
4. read information
5. continue to exit

exit with no action

Recovery

Following the test, recover the process directories by deleting contents of all process directories except render_bin.

Part 4 - Operations



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

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

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

Operations Readiness Check

Automation Scripts

Current automation scripts install as per installation 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 fileserver-1

Share Directories

All share directories vacant and ready.

Render Farm Ops Guides

Ops guides provide a logical description of the order in which tasks occur.

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 pro-forma in drop_box and make a copy

2 - Rename to the next job number, enter information

3 - Save 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 <infile>`

Else If High Image Quality (HIQ)

3 - `$ sh hiq_rp.sh –i <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_tickets 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

Running a Render Job

Ad hoc

Shell scripts can be run 'as needed' by referring to the Ops Guides and initiating one-off tasks. A good understanding of what each script is programmed to achieve is essential.

Scheduled Run

The normal way to render is to plan a major render job and administer it via a menu system that controls the sequence of tasks in a reliable end to end process. E.g. from the master host, render_bin directory.

1 $ ./render_control.sh

IMAGE RENDERING

Image rendering may be performed on the master host.

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 and high quality audio output that are more suited to a workstation.

ARCHIVING

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 procedures without allowing for quick and easy recall.

ROUTINE MAINTENANCE

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

Backing up file 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 software installation and configuration. Ideally they should not be used for storage or any other application.

INITIATING CONCURRENT CPU AND GPU RENDERS

A VNC session using the client 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 initiate 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 cores for the operating system and a GPU process. Alternatively multiple command lines can be opened on the master with one SSH host shell per terminal.

PROCESS MONITORING

The 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 maintenance 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 and a longer render duty cycle is possible by halting (not removing) services that are not needed however great care is needed. Once all services and startup applications are discovered and potential candidates are identified, it is advisable to consult a Linux or Ubuntu forum although system programmers have a strong bias toward constantly updating the configuration. Any actions to halt services should be tested on a proxy installation that can easily be rebuilt, i.e. an old laptop.

For starters, open the Control Centre -> Startup Applications and check the 'Show hidden' box. Scroll down to find Screensaver and uncheck it. This will prevent the screensaver from interfering with remote access. Other start-up applications are not needed on a render host. Now go to Control

Centre -> Software & Updates -> Updates tab. Change 'Automatically check for updates' to every two weeks. This may prevent locking of dependancies. Other updating activities can be halted or changed to be less frequent. E.g. there is no need for an automatic Snap update if it has not been used to install any software.

When certain that a systemd service is never used and it can be halted without consequence to the render operation, halt it with:

```
$ sudo systemctl disable <application>.service
```