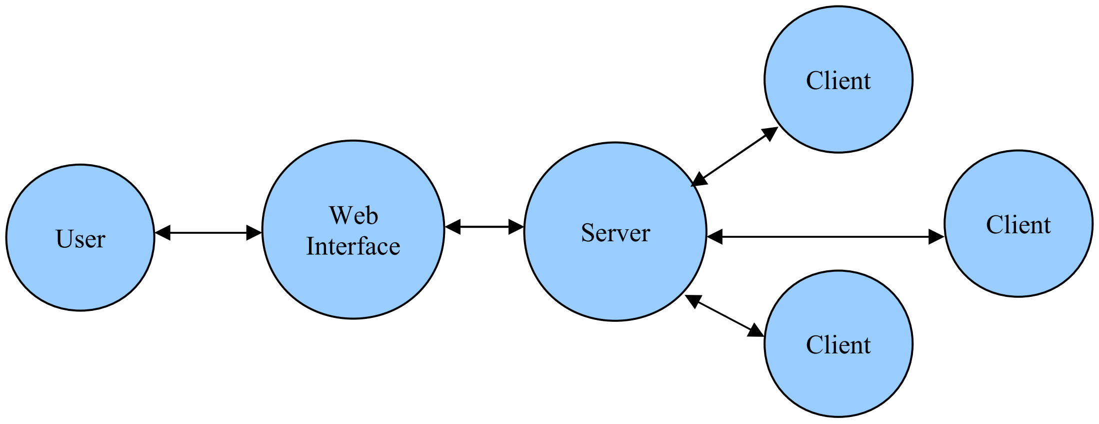
Thea Render Farm Automation

Revision	Author	Reason for Change
05/05/12	Ioannis Pantazopoulos	Initial version.
17/07/12	-//-	New Switches: -seed/-merge/-info/-licensefile

Introduction

Once network rendering has been setup in your computer farm, it is possible to automate the process of assigning a render job to the farm. At this point, we will not discuss the possibility of creating an intermediate application for batching render jobs, but rather how to start the application automatically. We discuss also various switches that can be used to parametrize the network rendering.

Please, take a look at figure 1; we can see how a render farm can be setup for render jobs coming through a web interface. The user interacts with the web interface computer, which accordingly triggers the server with a new render job. The server starts a network rendering with all clients co-operating on the render job. At the end, the machine running the web interface can detect the finished render job and send the results back to the user.



Launching the Clients

The clients running on each machine have to be launched before any render job arrives. This way, they can be involved from the very first render job; note that even if started with delay, the clients will still be able to contribute to a render job. We recommend adding the client application as a start up application, so that thea client starts immediately after logging in. Note that login is required, since thea client is an application and not a driver.

At this moment, the clients are already launched and polling for a new render job (refer to network rendering document, to see how to setup a client). We highly recommend putting the server computer and all client computers on the same (1 Gigabit or better) local area network. The web interface computer can be the same like thea server computer, but we also recommend separating these computers (and thus, separating these tasks), since the server may become occasionally overloaded with rendering and serving the clients.

What is left now, is to launch thea server with a new render job. This is something that will be done on per render job basis, i.e. when a new job comes in, thea server will be launched. A queuing system for the render jobs is necessary here, since we should not start a new render job before finishing the previous one, but we will not discuss this further.

Launching the Server

We can launch thea server whenever a new render job is available. Assuming that we can launch an application using a shell command, here is how our first command line will look like:

```
thea -load renderjob.pack.thea -server default -render -save output.png -exit
```

Here, we tell the application to launch, open a scene file, set default server options, render the scene, save the image after rendering and exiting at the end. We assumed that the application and scene file can be located, otherwise the full path to the executable as well to the scene (and probably the output image as well) are needed. If the full path contains spaces, then it is required to enclose it in double quotes, for example:

```
"/Program Files/Thea Render/thea" -load "PathToScene/renderjob.pack.thea" -server default -render -save "PathToOutput/output.png" -exit
```

Overriding Render Settings

It is possible to override the render settings with a custom preset of your choice. Render settings is considered to be a thea object that can be opened just like when opening a scene, i.e. using the same "-load" switch. To do this, you will need then to load the render settings just after opening a scene, for example:

```
thea -load renderjob.pack.thea -load mysettings.xml
```

Non-Private Render Farms

For commercial render farms or in general, for render farms that are not used for private rendering, a specific check must be realized to ensure that the user that submits a render job is a licensed one. This requires an internet connection to allow thea application to check the licensed user status by accessing our web server. The check can be made by the web interface computer. Here is the typical command line:

```
thea -farmcheck farmkey.txt userkey.txt
```

This will make thea perform a check with the web server, whether the user key is valid (note that the farm key is also required). These keys are the license keys that can be exported from thea once (from Help > License Form > Misc) and used afterward when submitting a new render job.

When the above command line runs, thea will exit immediately after checking with the web server. If everything is fine, it will just exit with return code 0. If not, it will display an error message box (containing a reason for the error) and will exit with non-zero return code. If you are automating the whole process and would like to avoid displaying the error message (which requires user intervention), then you simply add the -silent switch with the call:

```
thea -farmcheck farmkey.txt userkey.txt -silent
```

Checking on Render Job Finish

A usual way to check whether a render job has been finished is to have thea creating a file at the end (before exiting) while checking at the same time for the same file existence. Assuming that the file does not exist before starting thea, this is how we could modify the very first line for the render job:

thea -load renderjob.pack.thea -server default -render -save output.png -createfile finished.job -exit

The above line will create the file "finished.job" after rendering and saving have been completed. So, we are sure that all output is available. A typical pseudo-code on the web interface computer, checking for file existence would look like below:

```
// make sure file is removed before starting render.
RemoveFile("finished.job");

// issue command to launch render job.
ShellExecute("thea -load renderjob.pack.thea -server default -render -save output.png
-creatfile finished.job -exit");

while ( FileExists("finished.job") == false )
{
// wait for 4 seconds, before checking the file again.
    Sleep(4000);
}

// everything is now available.
```

Calculating Effective Computer Power

Calculating the computer power that has been used for a render job is quite important for (commercial) render farms, since this is directly related to the consumption and final cost for a render job. First of all, we have to make use of a specific command line switch that tells thea to save the network log in a text file, so that we are able to read it back and process it. Here is how we should change now our command line:

thea -load renderjob.pack.thea -server default -render -save output.png -savenetlog netlog.txt -createfile finished.job -exit

With the addition of -savenetlog switch we are able to save the network log information to a file that we can later process. This information is related to the server and clients contributing to a render, along with their render times. We can process this text file, as shown below[1], to calculate the effective power consumed by the render job (in GHz · sec).

```
float CalculateEffectivePower(const char *netlogfile)
{
    std::ifstream istr(netlogfile);
```

[1] The listings shown are provided only for clarification purposes.

```
    char netline[500];
    istr.getline(d); // first line is the header, so we can bypass it.

// the server line is enough to compute the effective overall power.
    std::string name,status,platform;
    float contribution;
    int eh,em,es,sh,sm,ss;
    int threads,revision;
    char c;
    istr >> name >> c >> status >> contribution >> c >> c;
    istr >> eh >> c >> em >> c >> es >> c;
    istr >> sh >> c >> sm >> c >> ss >> c;
    istr >> threads >> platform >> revision;

// compute total server running time in seconds.
    int servertime=(eh-sh)*3600+(em-sm)*60+(es-ss);

// total contribution in time.
    float totaltime=servertime*100.0f/contribution;

// compute total effective power put to the render job, we assume an i7 server.
    const int ServerCores=4;
    const bool ServerHyperthreading=true;
    const float ServerFrequency=3.0f; // in GHz
    const float ServerHyperFrequency=0.25f *  ServerFrequency; // +25%

    float ServerGHz;
    if (threads<=ServerCores)
        ServerGHz=ServerFrequency*threads;
    else
    {
        ServerGHz=ServerFrequency*ServerCores;
        if (threads<=2*ServerCores)
            ServerGHz+=ServerHyperFrequency*(threads-ServerCores);
        else
            ServerGHz+=ServerHyperFrequency*ServerCores;
    }

// finally, the effective power is just a multiplication.
    float power=ServerGHz*totaltime;
    return power;
}
```

Terminating Render Job Remotely

You may come at a time that you would like to terminate a render job that runs on the server but from another computer (or the same but using some sort of automation). This is possible with the -stopserver switch, that stops current render job (while continuing with saving any information as input in the original command line and exiting). Here is an example way to do this, assuming that thea server runs on the same machine:

thea -stopserver 127.0.0.1

Command Line on MacOSX

The command line calls above need to be altered when issuing them on a MacOSX system. Here is how a typical command line would look like then:

open -a '/Applications/Thea.app' –args -load renderjob.pack.thea -server default -render -save output.png -exit

More Switches for Network Rendering

We give here a list of switches that are useful to control network rendering. Note that for some of them, the order is not important but for many, it is important where to place them in the execution call.

-darkroom

With this switch, thea will run in darkroom mode, i.e. it will show only the darkroom panel omitting the viewport and other panels. This way, it will both start faster and consume less resources during rendering.

-client

With this switch, thea will run in client mode.

-licensefile filename

With this switch, thea will be forced using the given file, instead of searching in its Licenses folder for appropriate license file.

-load filename

This will tell thea to open the scene (or script or render settings or film buffer) indicated by the filename. Switch order is important.

-merge filename

This will tell thea to merge the scene (or film buffer) indicated by the filename (with the scene or buffer currently in memory. Merging a scene will take place with the default settings. Switch order is important.
When merging image buffers, these should have the same settings (i.e. same resolution, number of relight and channel images). Here is a command line example:
thea -load buffer1.img.thea -merge buffer2.img.thea -save final.img.thea -console -nosplash -exit

-render

This will tell thea to start rendering. Switch order is important.

-resume

This will tell thea to resume rendering (assuming scene and film buffer have been already loaded). Switch order is important.

-save filename

With this switch thea will save a scene, buffer, colimo project or an image indicated by the filename (the file type depends on the filename extension). Switch order is important.

-savenetlog filename

With this switch thea will save the network log file. Switch order is important.

-createfile filename

With this switch thea will create an empty file. Switch order is important.

-removefile filename

With this switch thea will remove the given file. Switch order is important.

-server [default]

With this switch thea will act as a server. If next argument is "default" then it will override settings with default distribution options (threads, priority, server port). Switch order is important.

-stopserver ipaddress

With this switch thea will issue a network message to the server given by the IP address to terminate its current render job.

-clientaccept ipaddress

With this switch thea server will accept only the given clients. It is possible to include multiple IP addresses, wildcard (*) and range (a-b). Examples:
-clientaccept 192.168.1.*
-clientaccept "192.168.1.* 170.125.1.1-5"

-clientreject ipaddress

With this switch thea server will reject any clients from the given IP address(es). Note that it is possible an IP address to be both accepted and rejected, in which case, the final decision depends on the range covered by the accept/reject IP addresses. For example, in the following command, 192.168.1.1 will be accepted but 192.168.1.3 will be rejected:

-clientaccept 192.168.1.1-5 -reject “192.168.1.3 192.168.1.*”

-maxclients number

With this switch thea server will accept the given maximum number of clients.

-serverport portnumber

With this switch thea will change the port used for network rendering (default value: 6200). Switch order is important.

-serverupload nominalminutes

With this switch thea will change the nominal upload period for clients to the server (default value: 1 min). Switch order is important.

-maxpasses passes

With this switch thea will change the Max Passes value for unbiased/progressive rendering. Switch order is important.

-maxsamples samples

With this switch thea will change the Max Samples value (i.e. targeted samples/pixel) for unbiased/progressive rendering. Switch order is important.

-timelimit minutes

With this switch thea will change the Time Limit (min) value for unbiased/progressive rendering. Switch order is important.

-threads number

With this switch thea will change the number of threads used in rendering for the server. Switch order is important.

-seed number

With this switch thea will use the given seed to override the default seed (automatically set based on internal or network counter). Default value -1 corresponds to automatic seed. In case of network rendering with 2 computers and given A seed for the first and B seed for the second, seed B must be greater than seed A + threads rendering on computer A, in order to avoid 2 or more threads giving the exact same contribution. Similarly for more than 2 computers; as a quick setting rule, for example, if your network computers do not have more than 16 cores, then seed of computer *k*, should be set to 16 $\cdot k$. Switch order is important.

-netlog filename

With this switch thea will be updating the network log every few seconds according to the render progress.

-info filename

With this switch thea will read from the given img.thea filename (image buffer) and create a file with the same name and .info extension, containing information about the image buffer. Switch order is important..

-farmcheck farmkey userkey

With this switch thea will check with the web server for the validity of farm and user exported keys.

-nosplash

With this switch thea will not show the splash image (slightly faster launch).