

# Implementation of a Distributed Rendering Environment for the TeraGrid

S. Lee Gooding, Laura Arns, Preston Smith, and Jenett Tillotson, *Envision Center for Data Perceptualization, Rosen Center for Advanced Computing, Purdue University*

**Abstract**—This paper discusses the implementation of a distributed rendering environment (DRE) utilizing the TeraGrid. Using the new system, researchers and students across the TeraGrid have access to available resources for distributed rendering. Previously, researchers at universities and national labs, using high end rendering software such as Renderman Compliant Pixie were often limited by the amount of time that it takes to calculate (render) their final images. For example, a short animation project may be about two minutes in length. At 30 frames per second (fps), this is 3600 frames. An average rendering time for a fairly simple animation can be approximately 2 minutes, resulting in a total of 120 hours just to render a simple 2 minute animation. The amount of time required to render introduces several potential complications in a research setting. For example:

1. Many researchers do not have access to an individual machine dedicated for rendering
2. During local rendering, machines become unavailable for other research use, often for extended periods of time
3. Researchers may potentially lose render data due to local hardware failure, or manual rebooting of their render machine
4. Due to the complexity of renders, they are prone to errors, often requiring multiple renders of the same material

In contrast, a typical animation studio has a render farm, consisting of a cluster of computers (nodes) used to render 3D images, known as a distributed rendering environment. By spreading the rendering across hundreds of machines, the overall render time is reduced significantly. For example: using a DRE of 200 computers to render the example animation above would take only 36 minutes for full render. Unfortunately, most researchers do not have access to a distributed rendering environment. Our university has been developing a DRE for local use. However, because we are a TeraGrid site, we recently modified our DRE implementation to make use of Open Source rendering tools and grid tools such as Condor, in order to make the DRE available to other TeraGrid users. By taking available

local university resources and sharing them with external researchers, the problems listed above can be addressed.

**Index Terms**—distributed rendering, TeraGrid, Condor, Renderman, Maya, 3D, animation, renderfarm

## I. INTRODUCTION & BACKGROUND

The next section describes the traditional rendering process and some of the difficulties that can be encountered, which provides motivation for constructing distributed rendering environments. The following sections provide details on the environment available at our university for distributed rendering, including the TeraGrid environment.

### A. Rendering

Rendering plays a vital role in the creation of computer generated animations and images. Several different rendering engines have been developed that perform complex calculations. Renderings can be created to imitate the visual aspect of reality (photorealistic renderings), or in a stylistic fashion as well (non-photorealistic renderings). Regardless of the style, renderings are done using algorithms optimized specifically for rendering. These algorithms can be quite complex. One example is ray tracing, where virtual rays follow backwards from the camera until light is reached in the virtual scene [1, p. 52]. Along with the obvious complexity of these algorithms is an increased time for calculations. With modern hardware a typical feature length film may take anywhere from several minutes to over 200 minutes for a single frame to be calculated [2]. An extreme example of the time taken for a render is some of the crowd scenes in Shrek 2 [3], which took up to 40 hours per frame to render [4]. In a university setting, students and researchers typically don't produce animation at that level of complexity, but it is possible.

There are several complications that can arise when rendering 3D animations, especially during students' initial learning stages, and researcher test renders. In the movie industry it is not uncommon to render a frame over 50 times [2]. Shadows, lights, animation, and textures are just a few examples of elements that could create undesirable results in the rendering process. The following paragraphs discuss in more detail some of the problems that can occur.

Light and shadow errors are difficult to diagnose before rendering a full animation. Some common problems include

Manuscript received February 8, 2006. This work was supported in part by the National Science Foundation through the Purdue TeraGrid Resource Provider grant SCI-0503992.

S. L. Gooding is with the Envision Center for Data Perceptualization at Purdue University, West Lafayette, IN 47906 USA phone: 765-496-7888; e-mail: goodings@purdue.edu).

L. L. Arns is with the Envision Center for Data Perceptualization at Purdue University, West Lafayette, IN 47906 USA (e-mail: arns@purdue.edu).

P. M. Smith is with the Rosen Center for Advanced Computing, Purdue University, West Lafayette, IN 47906 USA (e-mail: psmith@purdue.edu).

J. Tillotson is with the Rosen Center for Advanced Computing, Purdue University, West Lafayette, IN 47906 USA (e-mail: jtillots@purdue.edu)

flickering, jagged shadows, undesired light levels, etc. Test renders should be run at various stages of the animation to identify potential problems. This will help ensure that all parts of the scene are lit correctly. Materials are directly linked to the lighting in the scene. Their interaction cannot be seen properly until the image is rendered, thus many renderings will have to be done before the user is happy with the outcome. Each time an attribute of a material or light is changed, a new render must be done in order to see the results. It is difficult to determine what the final render will look like in motion, based on a series of static images. Tools used to quickly generate a sequence of images will help researchers fine tune their project, and identify potential errors before the project deadline. Prior to the DRE this was rarely possible.

Without a DRE setup, students and researchers are limited by the amount of time available to render their projects. Productivity decreases during renders because the researcher's machine has no free resources. Within a lab setting, other users expect machines to be available, which limit those machines for render usage.

The overall quality expected of an animation may also be lowered due to the lack of computing power available. Strict project time constraints make it difficult to address technical errors in the final render, resulting in a less than professional result.

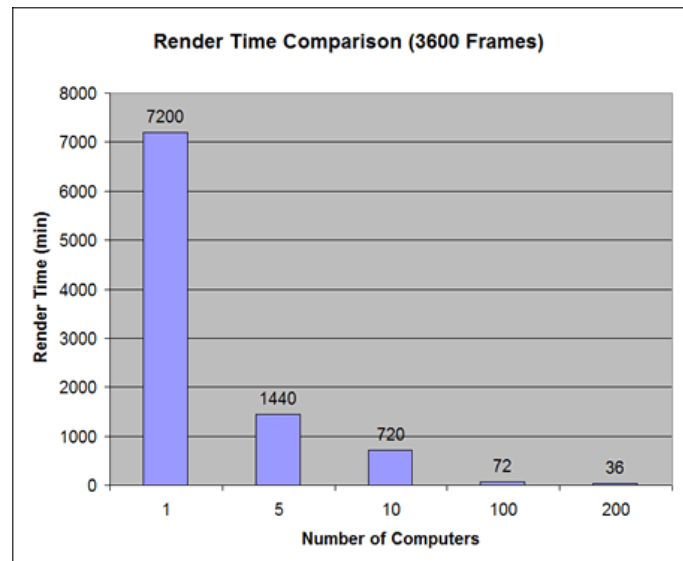
Complications that researchers and students encounter when using local machines for rendering, such as those mentioned above, prompted the creation of the Distributed Rendering Environment (DRE) at Purdue University. The DRE, utilizes machines across campus when they are idle. Using a combination of applications, render jobs are sent to a server that determines which machines are available and sends tasks to the available nodes, telling them what information to process. Rendered images are then uploaded to central network storage. Placing the rendered images on central storage reduces the possibility of losing data due to hardware failure. Additionally, the storage server is mirrored (backed up), further reducing the risk of potential data loss. The system has significantly decreased the amount of time taken to render animation projects. Render calculation times are significantly decreased, while end user computing resources are freed up for standard usage. The final rendered images can be downloaded at a later time by accessing the storage area.

Comparing the rendering environment for researchers and students prior to the DRE and after the DRE implementation, it becomes clear how significant the changes are by analyzing render times for typical uses. For example, Figure 1 (adapted from [5]) shows the approximate completion time for a 3600 frame render. It is clear from the figure that increasing the number of computers in the rendering cluster significantly decreases the total rendering time.

Figure 1: Impact of number of machines on total render time

### B. Available Resources

In order to implement a DRE, it is of course necessary to



have access to a large number of machines. Although academic departments often lack significant numbers of machines, the central IT organization for a university may have access to many more CPUs. The Rosen Center for Advanced Computing (RCAC) is a division of Information Technology at Purdue (ITaP), the central computing and telecommunications organization on the West Lafayette campus. Purdue's RCAC offers a wide variety of computational resources for computational scientists: Shared memory systems including a 320-processor IBM SP system, two 16-CPU IBM p690 Regatta systems, five 24-processor Sun F6800 systems, and two 56-processor Sun E10000 systems. Additionally, several different types of Linux clusters are available to serve Purdue's community.

### C. Recycled Clusters

In 2002, 650 recycled classroom workstations were repurposed into general-purpose Linux clusters available to all RCAC users. [6] These systems had previously served for three years in computer labs, but were still quite usable since only moving parts such as CD drives had worn out. By repurposing lab PCs at the end of their 3-year replacement cycle, the recycled clusters are able to be continuously expanded and upgraded. By summer 2004, over 1000 recycled PCs were having a new lease on life as recycled cluster nodes.

In previous work [5], we utilized the resources available through RCAC and the recycled clusters to implement a campus DRE. The system has been in operation for over a year and is often used by students and researchers. Because of the success of the initial DRE, we decided to expand the system to utilize more CPUs and to make it available to researchers at other facilities.

### D. Community Clusters

The current model for building high-performance clusters at Purdue is the community cluster concept [7]. Community clusters leverage federal and local investment in individual faculty led computing infrastructure and add value by providing centralized operations and higher system utilization. Rather than individual faculty purchasing small clusters, and

then needing to dedicate time or graduate student effort to installing and maintaining a cluster, community cluster users benefit from centrally provided maintenance of both hardware and software by a professional IT staff. Common hardware and software across large numbers of systems reduces duplication of effort.

Each member of the community clusters is provided with a dedicated PBS queue equal in size to their contribution to the cluster. Beyond their own queues, researchers are able to utilize idle cycles through a preemptible PBS queue, as well as through Condor [8]. The result of this system is high overall system utilization and additional resources for participants beyond what they have purchased.

### E. TeraGrid

In the fall of 2003, Purdue University, in a joint proposal with Indiana University, was awarded a \$3 million grant to join the TeraGrid [9], a grid computing project for building the world's largest, most comprehensive distributed infrastructure for open scientific research. On the TeraGrid, Purdue initially provided access both to its IBM SP system and recycled clusters. Now, Purdue's contribution to TeraGrid also includes a Data Portal that provides access to climate modeling data, satellite remote sensing, and weather radar data; both high-performance scratch storage (IBRIX), and archival mass storage (DXUL); a Condor-based distributed rendering engine; and a share of ownership in the newest community cluster, a 1024-processor, 64-bit Xeon system named "Lear". In August, 2005, RCAC also began to offer opportunistic access to all of its Linux clusters (2600 CPUs) to the TeraGrid through Condor.

Together, all of these resources combine to create an environment when a large scale DRE can be implemented and made available to researchers across the country.

## II. IMPLEMENTATION OF THE TERADRE

### A. Hardware

Having discussed the benefits and goals of constructing a distributed rendering environment, the following section discusses the technical hardware and software issues involved in constructing the system.

The core of the system is made up of several server computers that manage and exchange data with the execute nodes, which perform the actual rendering. The basic structure of the hardware can be seen in figure 2. The hardware is discussed next.

#### 1) Submission Server – TG-Gatekeeper

The submission server, TG-Gatekeeper, is responsible for maintaining communication with all execute (render) nodes within its pool.

The TG-Gatekeeper acts as the gate for users to access the TeraDRE. This is done by using authentication with a key through SSH. The TeraDRE scripts are located on the server, and the storage server is mounted on TG-Gatekeeper. Job status and control is done from this server

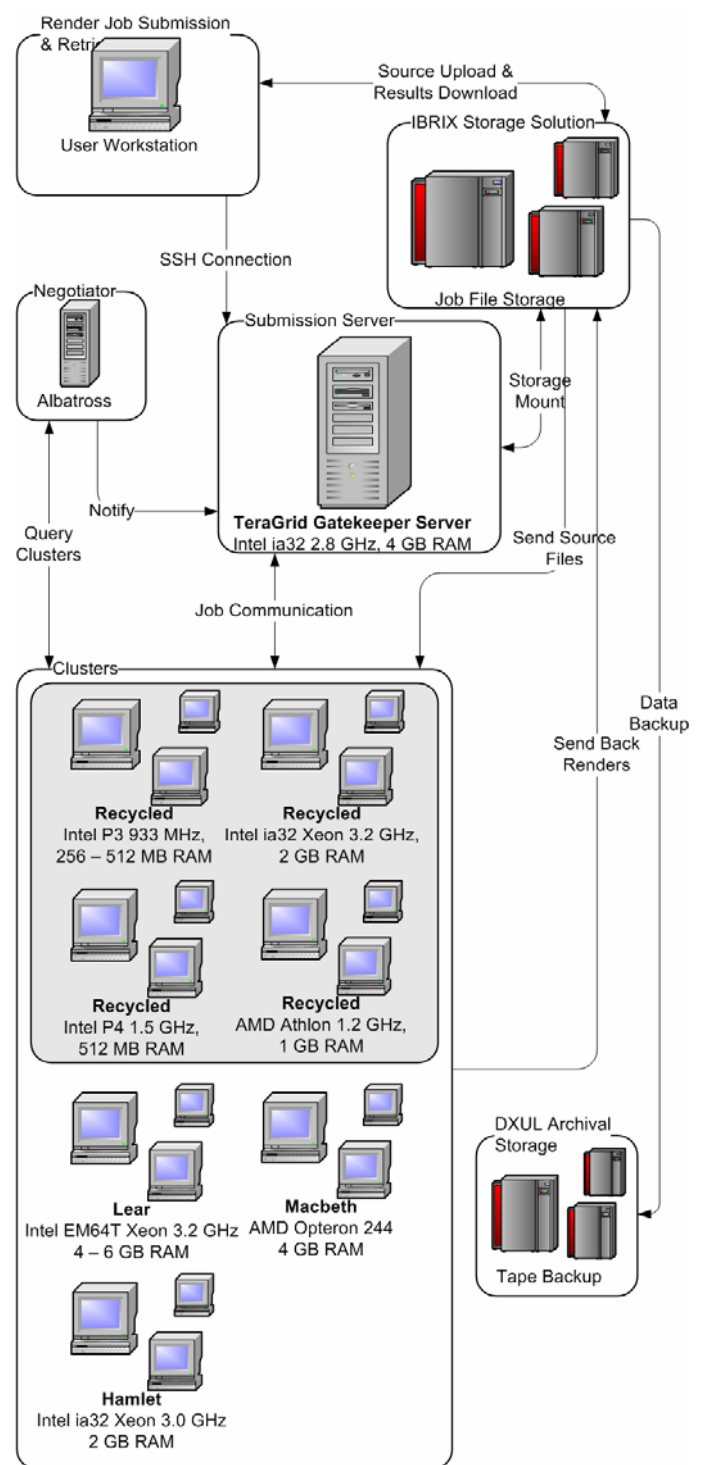


Figure 2: Structure of the TeraDRE

The submission server must be maintained by an administrator with the knowledge needed to implement the proper connections to other pools on the TeraGrid and Condor.

#### 2) Negotiator – Albatross

Albatross server acts as an information gatherer, and negotiates between render jobs and machines requesting new data for rendering. Albatross will find machines that are currently available for rendering, and notify TG-Gatekeeper

for submission to those machines. The machine also keeps track of user Job priority. [10]

### 3) *Storage Solution – IBRIX*

Essential for the network distribution of a job's files is a storage server that can be accessed by all the render nodes. A separate storage solution is integrated with the system for storage of source files and rendered images. The storage server carries several benefits, including fulfilling the goal of having central storage available for the TeraDRE. Stored on the server are the project source files, rendered output, and the scripts that need to be accessed to run the TeraDRE. IBRIX is mounted as scratch space on TG-Gatekeeper. This gives user access to the files using their TeraGrid account. Each user has a folder named based on their TeraGrid login.

The space available on this system must be very large, as images can range in size from a few kilobytes to several hundred megabytes (depending upon the format and resolution). Multiply that by the thousands of frames that will be rendered and the space can quickly fill. The system currently has 20 terabytes of storage capacity. IBRIX uses a custom file system that can handle the transfer of thousands of files from multiple sources [11]. The storage for the TeraDRE is highly scalable and designed to handle the strenuous demands of parallel file writing as done when thousands of frames are rendered and written to the same storage location.

### 4) *Execute Nodes – Clusters*

As discussed earlier, the computers used for the actual rendering are diverse. These machines are responsible for the actual calculations that are performed during the render. They have Condor installed to communicate with TG-Gatekeeper. This software identifies attributes of the machines, and starts the actual renders on the machines. All the required software is installed and configured on the render nodes. This includes Condor daemon services, `condor_startd` and `condor_starter`, which run in the background of the system [10]. `Condor_startd` sends the `condor_collector` (located on TG-Gatekeeper) information regarding the status of the render node. Obtaining information such as the amount of CPU and RAM available for processing is important in discerning the availability of a machine. `Condor_starter` receives commands from the submission server through `condor_negotiator` for remote execution as if they were run from the local render node.

### 5) *Archival Storage – DXUL*

It is very important to backup the images that have been rendered on the system. The DXUL archival storage server is used for backing up data on IBRIX to tapes.

### 6) *Submission and File Retrieval – User Workstation*

In order for any job to actually take place, machines must be setup to submit to the server, TG-Gatekeeper. The actual submission of a job takes place on any machine that has the proper software installed, as well as access to the TeraGrid.

This computer becomes known as the submission server. All source project files are initially located on this computer. Software run on this computer will prepare the researcher's project for submission to the TeraDRE. The server will also

initialize first contact with TG-Gatekeeper.

### 7) *Network Structure*

The TeraDRE system works most efficiently with high speed network connecting all the servers and render nodes together. The system will run on slower networks, but may result in delays of job submissions. This is primarily caused by lack of bandwidth for file transfers to the execute nodes. Condor configuration files can be modified to reduce overloading of bandwidth on slower networks and servers. A solid network foundation is essential for a reliable distributed computing environment. The RCAC Condor pools are connected with gigabit Ethernet, with 10-gigabit uplinks to the core campus backbone. From the core research network, RCAC's Condor pools can be accessed by a dedicated 10-gigabit link to the TeraGrid via I-light, the state of Indiana's high-speed research network.

## B. *Software*

Due to the heterogeneous nature of the machines in clusters, software selection is a very important part of the setup. It is important to select packages that are compatible with the hardware and packages that will be integrated. The software listed was chosen based on availability to the university and frequency of use in the classroom and research. The following are the core software packages used for implementing the TeraDRE:

### 1) *TeraDRE Framework Software*

#### a) *Condor*

Condor is the backbone of the TeraDRE. This software package is responsible for managing the workload of compute-intensive jobs run on a cluster. Condor takes care of job queuing and scheduling, priorities, along with resource monitoring and management. [<http://www.cs.wisc.edu/condor/description.html>].

Condor works by letting users submit their render jobs to the system where they are placed in a queue. Using Condor software, the system analyzes policies for the render, runs the job based on the policy, monitors progress, and supplies notification upon job completion. [12]

Condor was created at the University of Wisconsin-Madison, and is maintained by the Condor Research Project. [12]

#### b) *Linux*

One of the first things that must be considered when setting up the software side of a distributed rendering system is the operating system that will be running, and what role it will play in the system setup. A university environment often limits the decision of what operating system will be used.

Due to the very nature of a cluster, where hundreds or thousands of computers are used for computations, Linux has become a popular alternative to a commercial operating system. [13, p. 41] There are several reasons for the decision to use Linux as the operating system for clusters. First, Linux

is freely available. Second, it has superior performance compared to other desktop operating systems running in a clustered environment and has become the preferred operating system of many researchers.. [13, p. 41] Running Linux also gives system administrators easy remote access to the machine to make changes, tighter security control, and the ability to easily upgrade to other types of distribution software

Yet another important attribute to Linux is that while providing a UNIX environment, development tools that were written for Microsoft Windows can be supported. [14]

c) *GSISCP and GSISSH*

GSISCP and GSISSH are both utilized by the custom scripts written for the TeraDRE. In order to submit to the system both packages must be installed and located in the environment path. GSISCP is used for asset file transfer to the central storage location. GSISSH is used for directory creation, as well as to execute commands during submission on TG-Gatekeeper.

d) *Python*

Python was chosen as the core language for creation of the custom scripts that integrate the rendering software with Condor and the TeraGrid. Python is cross-platform, supports object oriented programming, modules, etc. It is also easy to read and maintain when compared to other scripting languages such as Perl.

Python scripts can also be packaged in a way that that does not require the user to have Python installed. This makes it very easy to distribute scripts that are useable on almost any system.

e) *BASH*

BASH, or Bourne Again Shell, is the primary command language interpreter for the GNU operating system. BASH also comes as a standard software install on most Linux systems [15]. BASH is used with the TeraDRE for its scripting capabilities and direct interaction with the Linux operating system.

2) *Rendering Software - Pixie*

A primary function of the TeraDRE is to distribute the frames needed to render research and educational animations and visualizations. Choosing the best software to perform the rendering is extremely crucial. Many factors were involved in the decision. Licensing issues were considered, compatibility, command line access, and popularity.

With the potential of a very large user base, it was important to identify and use a software package the many users would be able to generate data for. In 1988 Pixar published the RenderMan Interface Specification. The goal was to create something that would be adaptable to advances in modeling, animation, and rendering technology [1, P. 14]. Having an open specification has made it possible to create Renderman compliant rendering engines. With the specification being outdated now, these engines should be

rigorously tested for direct compatibility with Pixar's PhotoRealistic RenderMan, although they often use different algorithms [1].

The cost of the software also needs to be taken into consideration. The TeraGrid setup is a unique situation that may call for copies of the software to be installed all over the country. This could create several issues with software that requires a license to be purchased.

Due to the nature of the distribution system, any render software to be used must have command line access for rendering. Without command line access there is no way to execute the render.

With these considerations, Pixie, an open source (Gnu Public License) RenderMan compatible renderer was chosen for the initial setup of the TeraDRE. Pixie is being developed for graphics research, and users that cannot purchase expensive rendering packages. [16]

Most popular animation packages, such as Alias Maya, support some form of exportation to RenderMan Interface Bytestream (RIB), which is the standard file type for RenderMan renderers. This allows for a large number of users to utilize the TeraDRE.

### III. THE SUBMISSION PROCESS

The actual submission process to the TeraDRE is initiated from the user's computer. Many tasks have been automated, making the process very simple. Once submission is completed, job control is then done from TG-Gatekeeper. This is different from a typical Condor implementation, where the submission machine maintains control of the scheduling for the job. Segregating the client computer from Condor allows it to be shutdown, or used for other purposes without disrupting the render job. Before the first submission an ssh-key certificate with tg-gatekeeper.purdue.teragrid.org should be setup to eliminate the need to enter a password multiple times during submission.

The previous sections focused on the hardware and software requirements for the TeraDRE. This section describes the underlying execution of a typical rendering job submitted to the system. A typical 3d animation project contains a large number of assets that are required during the rendering process. Keeping these assets centralized and organized is essential for a functional distributed environment. Code was written to gather, organize, and maintain these assets for rendering. All client side scripts have been packaged and need to be extracted to a directory on the machine that will be submitting jobs to the TeraDRE.

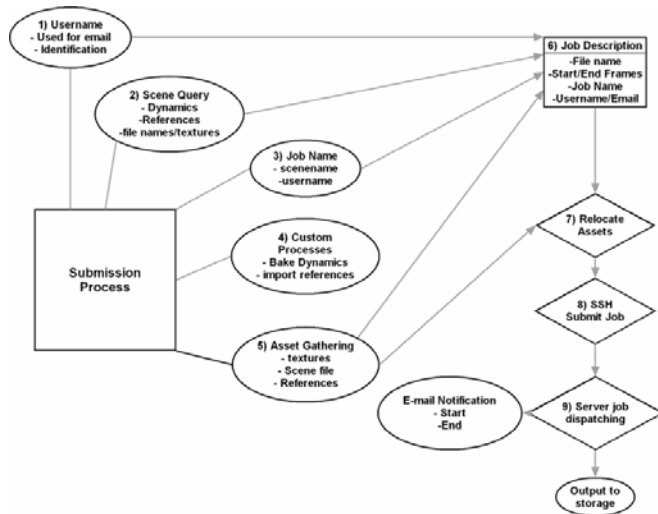


Figure 3: Flow chart of the submission process with custom scripts

### A. Organize Assets and Initialize

Before beginning submission, it is important to verify that the RIB files being used are compatible. It is required that RIB files be named with a frame pad size of 4 (i.e., `baseName.0001.rib`, `baseName.0002.rib`, ..... `baseName.0103.rib`). This is due to a limitation in the current filename parser. ASCII RIB files should also be checked for internal file paths. Paths must be relative or shaders and textures may not be properly found when rendered using the TeraDRE.

The first step in submitting a job is identifying the location of all the assets needed for calculating the scene. This would include textures, shaders, RIB files, etc. The full path to each file will be needed. Textures should also be converted to a RenderMan compatible format. A script has been created that will parse through a directory and run the proper command for converting all images.

The initialization script, `tDRE_init.py`, can now be run. This script manages all the global variables (through a class) passed between the other Python script modules. At the end of the submission process it initiates the job with Condor on the TG-Gatekeeper.

Key static variables stored in the init class are:

- `storageLoc`: this is the root location of IBRIX mounted on TG-Gatekeeper.
- `submitLoc`: web address of TG-Gatekeeper
- `condorLoc`: location of Condor on TG-Gatekeeper
- `linUserPath`: root location of user accounts on the IBRIX mount
- `batchExecPath`: location of a predefined batch executable file used by Condor to execute on render nodes
- `pyScriptsLoc`: path on TG-Gatekeeper pointing to the TeraDRE source files

Key variables passed from the command line:

- `renderer`: keyword designating the render engine to use for the job. This determines key commands placed in the job description file (discussed below)
- `sceneName`: actual name of the core RIB file to be rendered
- `startFrame`: the first frame in the animation
- `endFrame`: the last frame in the animation
- `username`: username registered for TeraGrid access

The init script is run from the command line within Linux

or Windows. The file will be in the directory where the client scripts were extracted to. Several flags are used to pass information required for proper file generation. The flags are as follows:

- u (TeraGrid user name)
- r (Renderer Used, pixie)
- n (baseName of scene file: `baseName.####.rib`, user would specify `-n baseName`)
- s (Start Frame)
- e (End Frame)
- f (baseName file1 file2 file45)

Example usage:

```
python.exe uDRE_init.py -u username -r renderer -n sceneName -s startFrame -e endFrame -f sourceFiles
```

Following the execution of the above command, several Python classes are instantiated within the main init script, preparing the job for actual submission to Condor on the TeraGrid.

### B. Job Name Generation

Vital to completing a successful submission to the TeraDRE is accurate identification of the current system user (Figure 3, step 1). This user's TeraGrid account name is passed to the init script. The username will be used internally by the scripts for running secure connections to TG-Gatekeeper. This login name enables the proper permissions to write to the storage server. When an admin views the queue he/she will be able to identify users based on this stored value.

Each job can be identified by the user that submits, but a unique name must be assigned to the actual job so that multiple jobs by the same user can be distinguished. The job name is based on a combination of the username, main RIB filename, and the current date/time.

The job name will also be used for folder structure generation, and naming of specific files generated during the submission process.

Generation is initiated by a class within the script module `tDRE_jobName.py`, which is instantiated from the init script. This will pass the unique job name back to the init script, making it available to other script modules.

### C. Directory Structure Creation

Before the job files are uploaded to IBRIX, a job specific directory structure must be created. User specific directories are mounted under `/scratch/ibrix1` on TG-Gatekeeper. Job specific directories will be created within the user's personal folder.

The `tDRE_createDirs.py` script opens a tunnel to a connection made with `gssissh`. `Mkdir` commands are then passed through the connection to TG-Gatekeeper, creating directories for the user. For example, the following directories will be created if they do not exist:

- `/scratch/ibrix1/username/condor_jobs/`
- `/scratch/ibrix1/username/condor_jobs/jobname`
- `/scratch/ibrix1/username/condor_jobs/jobname/source_files`

Rendered files, log files, and asset files will be placed within this directory structure. Condor will be aware of these directories, and access them throughout the rendering process.

#### D. Job Description Generation

One of the most important assets from Condor's perspective is the job description file. The job description file is an ASCII file with Condor job specifications. This file is unique for each job that is sent through the TeraDRE. The job description file communicates to Condor, on TG-Gatekeeper, all the information needed to distribute the job to all of the execute nodes. Information such as asset file locations, frames, render output locations, environment variables, log file location, policies, etc, are all identified in this file. Using the `tDRE_jobDesc.py` script, variables stored from the `tDRE_init.py` script are used in the generation of the job description file. This is where the actual render commands are defined for each frame in the job. The correct `condor_exec` file is enabled and Pixie specific render commands are defined. These commands are eventually passed to the `condor_exec` file. The `condor_exec` file is a batch file that contains Pixie render command (`/apps/01/Pixie-1.5.5/bin/rndr $@`). The file is located on TG-Gatekeeper. When a frame is sent to an execute node the `condor_exec` file specified in the job description file is copied over and executed by Condor with the render arguments being passed in. Below is an example job description file:

```
# liquidTextureAnim_perspShape.0001.rib
Universe = vanilla
Executable = /usr/rmt_share/scratch34/src/r5/condor_exec/pixie_exec
environment = PIXIE_TEXTURES_PATH=/usr/
log = crW.log
error = crW.$(Process).err
output = crW.$(Process).out
notification = Never
initialdir=/path/condor_jobs/116_liquidTextureAnim_perspShape.0001.rib-283/
should_transfer_files = ALWAYS
WhenToTransferOutput = ON_EXIT_OR_EVICT
Arguments = -p -t /path/condor_jobs/116_liquidTextureAnim_perspShape.0001.rib-283/source_files/liquidTextureAnim_perspShape.0001.rib.0001.rib
Queue
```

#### E. Asset Relocation

With the completion of the job description file all the prerequisites for submission have been fulfilled. The assets will now be copied to IBRIX storage. Having all of the necessary files on the central storage device serves several purposes. First, users should be able to logout of the local computer once submission is completed. Files necessary for the render cannot be stored on the local computer or they will become inaccessible when logged out. Placing the files on central storage provides a location that all render nodes can retrieve the data from for rendering. Second, a temporary backup of the work is now located on the drive. Lastly, in group project situations this provides an easy way for all team members to access the scene files used to render.

The Python init script accesses a class located in

`tDRE_moveFiles.py`. Using `gsiscp`, all files passed to the `tDRE_init.py` script at run time, including the job description file, will be transferred to the IBRIX mount on TG-Gatekeeper. These files will be located within the `jobname/source_files` directory

#### F. Condor Submit

All assets have been placed in identified locations and the TeraDRE job is ready for execution. The last command run by the Python scripts runs the `condor_submit` command on TG-Gatekeeper. `Condor_submit` is executed through a `gsissh` connection and is pointed to the job description file. Condor will output the status of the submission process and state success or failure. A job cluster number will also be given. This cluster number is the identifier used with Condor for controlling the job and checking its status.

It is possible to manually resubmit a job as well. This can be done by logging directly into TG-Gatekeeper and running the `condor_submit` command on the description file for the job needing resubmission.

#### G. Checking Job Status

Upon successful submission, users can login using `ssh`, or `gsissh`, directly into `tg-gatekeeper.purdue.teragrid.org`. Once logged in, the `condor_q [clusternumber]` command can be executed. Condor will output the status of the job. More detailed information can be retrieved. Detailed commands can be found in the Condor documentation. Depending upon the traffic currently running through the TeraGrid, execute nodes may not be immediately available for rendering.

### IV. RETRIEVING RENDERED FILES

Once a job is completed, the rendered files need to be removed from the storage server. Log files and rendered output will be located in following directory on TG-Gatekeeper:

```
/scratch/ibrix1/username/condor_jobs/jobname
```

The files should be retrieved as single frames. Using compositing software the images can be combined into a single video and compressed for dissemination.

### V. DISCUSSION AND FUTURE WORK

#### A. Speed Benefits and High Definition

High Definition (HD) video is becoming a standard for video and animation production. This means that renderings will have to be done at much higher resolutions, resulting in increased render times. This generates an even greater demand for the TeraDRE. The benefits of the TeraDRE are evident from several test renders that have been run. A T4-Bacteriophage Animation rendered for Michael Rossmann, at Purdue University, took approximately 17 minutes per frame at 4K (4096x3112) high-definition resolution on a 64-bit AMD computer. The entire animation was over 1500 frames, which would take over 17 days to render on the single

workstation used for rendering.

Along with the higher resolution comes the need for more storage space. With over 20 terabytes of information, the TeraDRE is ready for the transition to HD

### B. Adding Additional Render Engines

The TeraDRE was designed for easy integration of other rendering packages. Virtually any Linux compatible renderer with command line access can be implemented with the TeraDRE. By modifying the `tDRE_jobDescription.py` script, new rendering engines can be added to the TeraDRE. Users can add engines without having to load them on the execute nodes. By placing the new render engine in a location readable by execute nodes, a custom `condor_exec` file can be created to point to the new render executable.

### C. Expansion of Computing Resources

Multiple Condor pools can interact with each other. When one is filled, it is possible for jobs to spill-over and be distributed over another cluster. Integration with the TeraGrid allows for easy expansion to pools all across the country. This would possibly lead to the implementation of the world's largest distributed rendering environment through the TeraDRE and TeraGrid.

### D. Optimizing RIBS for Animation Rendering

ASCII RIB files can be quite large. When thousands of them are generated for rendering across a network compression should be used to minimize file size and reduce bandwidth across the network. Currently under development is the implementation of a method using diff, described by Ian Stephenson in 2002, resulting in compression of approximately two thirds [19].

Several BASH scripts and C programs are being created to automate the process and integrate it with the current TeraDRE setup.

### E. Multi-Distribution System Compatibility

One of the eventual goals of the DRE is to have a system developed that will plug into various distribution systems without having to alter the external scripts. With the complex environment that constructs the infrastructure of a university it is often difficult to keep up with the changes in software. The TeraDRE setup will allow easy migration to diverse distribution setups. Each script is written to receive commands that are custom for each application that plugs in.

### F. Web Based GUI

Submitting render jobs to the original DRE has been simplified on some lab machines. Specific lab machines have been setup with the proper software and configuration for use with the DRE. Although submission doesn't have to take place on these designated machines, there have been some customizations made to those systems that simplify the submission process. This is done through custom menus within software like 3D Studio Max(3) and Maya(4). Shown in Figure 4, these menus integrate DRE access directly into

the animation package interface.

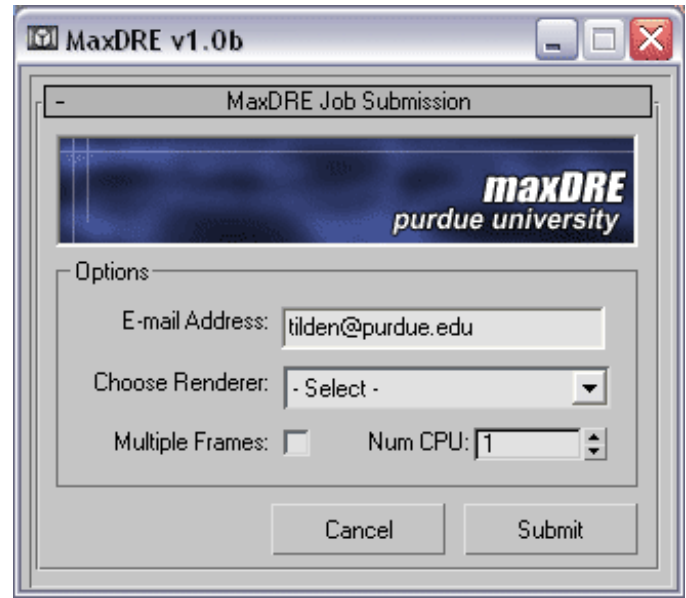


Figure 4: Custom menu accessed from within 3D Studio Max

Similarly, we would like to provide a graphical user interface (GUI) for accessing the TeraDRE. A web based GUI is an ideal method for submitting jobs to the TeraDRE. Implementation of a GUI will eliminate many redundant tasks, such as the generation of file lists, textures, and shaders. The GUI will also allow for a job history, that lets the user re-render previously submitted jobs, while specifying changes, new files, etc. This will reduce bandwidth by eliminating uploads of files that are already located on the user storage space.

## ACKNOWLEDGMENT

The authors wish to thank Gary Bertoline, Scott Meador, Sebastien Goasguen, Adam Piper, and Jonathan Tilden.

## REFERENCES

- [1] A. Apodaca, and L. Gritz, *Advanced Renderman: Creating CGI for Motion Pictures*. Academic Press. SD, CA. 2000
- [2] Pipelinefx, <http://www.pipelinefx.com/about-profile.php>, Pipelinefx.com, Verified 2/8/2006.
- [3] I. Lynch, "Greatest computer-generated movie sequences – Shrek 2," <http://www.vnunet.com/vnunet/features/2130027/greatest-computer-generated-movie-sequences-shrek>, Vnunet.com. Aug. 2004, Verified 2/8/2006.
- [4] B. Evangelista, "Shrek sequel – a picture that's worth 20 terabytes: Animators raise the bar of complexity in movie," *San Francisco Chronicle*. June 2004, p. E-1.
- [5] K. Madhavan, G. Bertoline, and L. Arns, "The Design of a Distributed Rendering Environment to Support Classroom Instruction in Animation and Scientific Visualization." *IEEE Computer Graphics and Applications*, vol 25, no 5, pp. 32-38, Sept/Oct 2005.
- [6] Purdue University, "Recycled computers make Purdue a powerhouse," <http://www.itap.purdue.edu/enablingthefuture/discovery/recycled.cfm>, Verified 2/8/2006.
- [7] Shuey, et al, "Local Community Clusters Experiences with Resource Sharing in International and National Grid Infrastructure." *NSF Infrastructure Experience 2005*, Urbana, IL.

- [8] UW-Madison Department of Computer Sciences, "Condor High Throughput Computing," <http://www.cs.wisc.edu/condor>, Dec. 2005, Verified 2/8/2006.
- [9] TeraGrid, "TeraGrid," <http://www.teragrid.org>, Verified 2/8/2006.
- [10] Condor Team, "Condor Administrator's Manual – The Different Roles a Machine Can Play," [http://www.cs.wisc.edu/condor/manual/v6.7/3\\_1Introduction.html#SECTION00411000000000000000](http://www.cs.wisc.edu/condor/manual/v6.7/3_1Introduction.html#SECTION00411000000000000000), Oct 2003, Verified 2/8/2006.
- [11] IBRIX, "Technology: Scaling File Systems to the Next Level," <http://www.ibrix.com/technology.php>, Verified 2/8/2006.
- [12] UW-Madison Department of Computer Sciences, "What is Condor?," <http://www.cs.wisc.edu/condor/description.html>, Verified 2/8/2006.
- [13] R. Buyya, *High Performance Cluster Computing: Architectures and Systems, Vol. 1*. Prentice Hall. Upper Saddle River, NJ. 1999.
- [14] H. Dietz, "Linux Parallel Processing HOWTO," <http://yara.ecn.purdue.edu/~pplinux/PPHOWTO/pphowto-5.html#ss5.1>, Jan. 1998, Verified 2/8/2006.
- [15] T. Kay, *Linux + Certification Bible. Hungry Minds. New York, NY. 2002*
- [16] O. Arikan, "Pixie," <http://www.cs.utexas.edu/~okan/Pixie/pixie.htm>, Verified 2/8/2006.
- [17] Discreet, <http://www.discreet.com>, Verified 2/8/2006.
- [18] Alias, <http://www.alias.com>, Verified 2/8/2006.
- [19] I. Stephenson, "Compressing RIB Sequences using diff source information," Presented at the PIXAR User Group Meeting, San Antonio, Siggraph, 2002.