Major Project Documentation

Joe Withers

Contents

1 Pipeline Management

1.1 Requirements

- Reliably store all of the project data in such a way that is accessible to all team members.
- Provide artists with a tool that manages assets, allowing them to update, reference,
- Provide artists with a simple method for releasing new versions of assets.
- Provide artists with a simple method for gathering assets, so that they can be referenced into a scene.
- Manage versioning of assets, accompanied by information regarding each versions release date, author, and description.
- Automatically update references to assets whenever an asset or one of it's depenencies has been updated.
- Provide an automated method for caching the entire scene, with the aim of 'packaging' the project to make it suitable for rendering on different machines.

1.2 Limitations

Prior to developing the pipeline and associated tools, it was important to address the limitations imposed by the environment in which we would be working. The most important of these were:

- Lack of unified storage amongst users. Due to the way the university network is set up, it isn't possible to have a single network location for our shared data storage, without sacrificing one members allocated user storage.
- Lack of storage per user. The approximate storage limit per user is 50gb, which would quickly be hit in a complex production environment. It is therefore imperative that we are concious of the data that we keep hold of.

- Lack of storage space on the renderfarm. The approximate storage limit on the renderfarm is 30gb, meaning that all of the data required to render a scene or shot must fit well within this limit, as rendered frames are written to the same location.

1.3 Storage

For file storage we chose to use Resilio Sync, a peer-to-peer file synchronization service to store all of our working files. This ensured that each team member has their own local copy of the entire working directory, which is beneficial when creating backups. We chose Resilio Sync primarily because it is a free service that is compatible with the university computer network, however it does present us with some problems.

Due to being it peer-to-peer service, we often found that directories would fail to synchronize properly if not syncronized frequently with the other peers. This would not be a problem in a cloud hosted service as the directory state of the working directory would be reliably centralized, reducing the possibility of files becoming desynchronized, however these services are typically expensive and it was difficult to predict our exact storage requirements.

We also noticed a strange problem with Resilio Sync, in which the contents of files would be reduced to 0 bytes. Fortunately the data is usually not lost as it is sent to the 'Archive', which functions as a temporary recycle bin, though restoring these files manually each time it happened proved to become quite tedious. I decided to write a simple bash script to check through the working directory to identify any files with a size of 0 bytes, and to check if a matching file was present in the Archive. However, this wasn't particularly effective as often they would be missing from both the main working directory and the Archive, meaning I would have to search through backups to find the file to restore, which at times felt a bit like baby-sitting.

1.4 Asset Management

1.5 Asset Pipeline

2 Character Rigging

2.1 Requirements

For this project I was responsible for the rigging of both characters, as well as the rigging of any props that the characters interact with. Prior

to working on the character rigs, it was important to outline features that would be required to achieve an appealing animation. This ranged from features that would allow the animator (Yves Bedenikovic) to work with them more efficiently, to features that would improve the overall aesthetic of the animation such as cloth and hair simulation. The following features were found to be of highest importance:

- Controls should be familiar to the animator to allow them to work intuitively with the rig. This can be achieved by using previous rigs that the animator has used as reference when setting up controllers.
- Rigs should be capable of achieving the desired facial expressions and poses as dictated by the story.
- Rigs should include the necessary geometry and nodes to allow for cloth simulation to be applied to clothes.

2.2 Solution

With these features in mind, I decided to use an automated rigging system to speed up the creation of the character rigs. This allowed me to focus primarily on the features listed above, and let a tool automate the creation of the basic bipedal rig.

I first looked at Kraken[2], a rigging system included within Fabric Engine. This appealed to me as it was easily extensible through it's scripting language, which I thought I would find intuitive given that I had produced a basic automated rigging system for the specialism assignment in second year. Unfortunately, Fabric software went bankrupt at the beginning of the academic year so we were unable to get it working on the university computers.

I then found Advanced Skeleton 5[3], an extensive rigging tool for Autodesk Maya, which I found to be extremely capable and was more than adequate for my needs.

3 Groom

Xgen stuff

3.1 Pipeline

3.2 Results

4 Rendering

4.1 Optimisation

4.2 Distributed Rendering Tools

Due to inconsistencies with the renderfarm, our only other immediate option for rendering was to use the
[4] [1]

5 Compositing

References

[1] Ben Martin. Parallel ssh execution and a single shell to control them all, 2008.

[2] Fabric Software. Kraken - rigging framework, 2015. Accessed 23 December 2017.

[3] Animation Studios. Advanced skeleton, 2016. Accessed 23 December 2017.

[4] www.parallel ssh.org. Parallelssh - asynchronous parallel ssh client library., 2014. Accessed 8 October 2017.