blenderCloud: Simulation baking and rendering in the Cloud

Dacre Capstick
Department of computer science
University of Bristol
Bristol, UK
Email: dc12335@my.bristol.ac.uk

Vong Panha Huot
Department of computer science
University of Bristol
Bristol, UK
Email: vh13294@my.bristol.ac.uk

Abstract*— This paper outlines *blenderCloud, a blender 3D plugin and render farm service that uses Google Compute Engine to provide users with a fully customizable and scalable solution to animation rendering and simulation baking. A series of screenshot instructions can be downloaded from safe. blenderCloud has a web console where the job progress and its detail can be monitored. The console can be accessed by this link: http://104.199.94.139:3000/

Due to the nature of blenderCloud, the target user tends to be a seasoned blender user to set up the render settings correctly. Hence, we have provided a file that is set up with all necessary features for download in safe. This is named 'ocean.blend'

Keywords—cloud computing; simulation; rendering; blender-3D; Google Compute Engine; Animation

I. INTRODUCTION

Rendering is an extremely compute heavy process. Full-length feature films consume far more CPU time rendering than the development of the films themselves. Disney Pixar's Monsters University took 29 hours to render each frame, even with the power of 24,000 cores in their own data center. All in all, it took over 100 million CPU hours to render the final film [9]. From this, it is clear to see that current render techniques take a vast amount of time to produce high-quality results.

Although we will not cover the full details of what is involved in rendering at production level quality, it is important to understand that it is a compute intensive process. Unlike real-time applications, such as video games, offline rendering uses ray tracing and simulates light in a highly physically accurate way where irradiance and energy conservation is adhered to. As with simulating anything from the real world, a vast amount of computation is needed. The rendering technique used is not the only difference between real-time and offline rendering. The content rendered offline tends to be far more complex, consisting of millions of triangles and often includes other simulations such as fabric, hair, and fluid.

II. MOTIVATION FOR USING CLOUD COMPUTING

A. What is cloud computing?

Cloud computing is a term used to describe the delivery of services over the internet. Cloud computing enables users to access compute resources such as virtual machines or storage, as a utility. Much like electricity, the amount of computing resources used will be logged and billed. Cloud resources are billed in a granular nature, meaning that users pay only for what they use. Cloud resources are, also, easily scalable with cloud providers even offering auto-scaling functionality meaning that when rented compute resources are saturated with work more compute resources are provisioned up to some maximum specified by the user. The flexibility of the cloud and the billing system simplifies server choices for fast growing startup companies and can save large e-commerce businesses money during less busy periods [10].

B. What are the benefits of cloud computing?

There are many benefits to cloud computing, the most obvious being the fact that users have access to immense compute resources without having to invest large amounts in capital expenditure to set up a data center themselves. Additionally, as mentioned previously, the flexibility of the cloud allows users to scale their computing resources up or down to meet demand, saving money in quiet periods and ensuring that all users of a system can be served during busy periods.

C. Why use the cloud for rendering?

The motivations for doing rendering in the cloud are numerous. An obvious motivation is speeding up batch rendering by distributing the work between many nodes, which is made possible by the embarrassingly parallel property of rendering. On top of this, computationally heavy processes can be offloaded onto Google's servers, leaving the user's PC free to use. Finally, as mentioned earlier, the nature of cloud computing allows small animation studios or even independent developers to render high quality animated content without having to invest vast sums of resources to build a render farm of their own. The granular nature of the way cloud computing is charged also ensures that users pay only for the compute resources used.

blenderCloud takes advantage of the scalability of the cloud by offering users complete control over how many instances they want to be running at one time and could be easily extended to allow users to choose what machine type these instances would be (this was not done to keep to the free tier of compute engine). This is also a key reason why compute engine was used over Google's app engine. The desire to control exactly the number of instances used and the type of machine used was a motivation driven by the obvious need for small studios and individual developers to have complete control over the cost of their cloud usage.

Due to time constraints, our application was rescoped to be a proof of concept on how blender's simulations could be

baked and rendered in the cloud. By this, we mean that we focused on adapting only one of blender's many simulations to work in the cloud. However, time permitting, this could be extended to all of blender's simulations. In this work, we focus on ocean water simulation. The simplicity of the algorithm used over that of the complex fluid simulation and cloth simulation algorithms offered by blender allowed us to develop a robust implementation in the short amount of time available.

D. Why Google Compute Engine?

The two leading cloud service providers Amazon and Google both have IaaS (Infrastructure as a Service) offerings in the form of Google Compute Engine and Amazon EC2 (Elastic compute cloud). IaaS offers users access to virtualized compute resources and an operating system, allowing users complete control over what is and is not installed on the virtualized instances which are exactly what is needed to set up the blenderCloud system. Google's compute engine was chosen over Amazon's EC2 due to the speed advantages associated with Google's instance creation over that of Amazon's EC2, providing up to 5 times speed. [1] This is an important feature because blenderCloud creates instances on job creation (based on the desired number of instances, specified by the user). Another motivating factor why Google's Compute Engine was chosen over Amazon's EC2 is that Google Cloud Storage offers performance benefits over that of Amazon's S3 in term of download and upload speed. This is particularly important in the case of blenderCloud due to the amount of data transfer that can be demanded from transferring hundreds of rendered image files as well as uploading large 3D application files. [2]

III. IMPLEMENTATION

In this section, we detail the design and implementation of our cloud render/fluid simulation baking farm as well as outline what tools and techniques were used in development. This will be covered in two distinct sections devoted to frontend and backend respectively.

A. Frontend

When designing the front-end of our application, it was important to offer users several key features: The ability to submit render jobs directly from blender's UI; Full control over the resources used to carry out the job; the ability to view the job's progress and the progress of the subsequent tasks within each job; the ability to terminate jobs; the ability to download the rendered images once the job was finished.

To offer these features it meant that two interfaces had to be developed: One for the blender plugin, allowing artists to unlock the power of Google's data centers directly from the 3D application that they already use; The other for the web interface where users can monitor their render jobs. To achieve this, several technologies were used:

- Blender python API enables access to data and settings within blender as well as standardized UI elements. This enables the ability to make a UI that is familiar to blender users as well as access data and settings information that can be pulled out and sent to GCE (Google Compute Engine) on job submission.
- AngularJS: A structural framework for dynamic web apps [4].
- Angular Material: a UI Component framework and a reference implementation of Google's Material Design Specification. Giving our web UI a familiar look and feel.

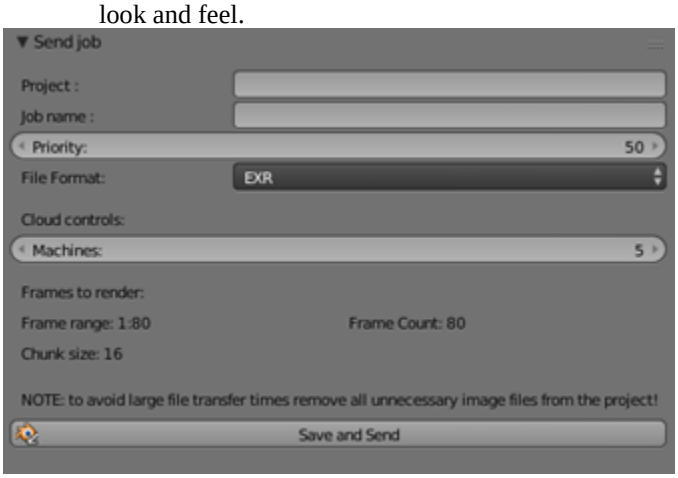

Figure 1: blenderCloud plugin for blender

Figure 1 illustrates the design of the blender plugin developed to allow users to submit jobs to the cloud from within blender. The benefits of doing this are many-sided. The most obvious benefit is that the user does not have to interact with a new and possibly confusing interface. Additionally, the blender python API enables the retrieval of information stored within the blend file meaning settings and parameters that are needed by our render farm can be retrieved on job submission. Hence artists can set up the file as if they were going to perform the rendering on their own machine and our cloud render farm will deliver the exact results they are expecting. Finally, this also allows the blend file to be packed, meaning that all images and external dependencies within the blender project are bundled into the file making it far more efficient and easy to manage when uploading to the cloud. The UI is simple and conforms to blenders standardized UI design ensuring that blender users are familiar with the controls. The only required information is the number of machines, project name, and file format. the frames to render, the render engine, and other settings are taken from the render settings within the file and the chunk size is worked out automatically based on these parameters. Due to quota limits on the free version of Google's compute engine, only 8 CPUs can be running at one time and hence our application is limited to 8 parallel machines, each machine only having one CPU. On clicking the 'save and send' button the blend file, as well as the information provided in the plugin interface are sent to the cloud via HTTP requests. The job is then created and starts being processed in one of Google's data centers.

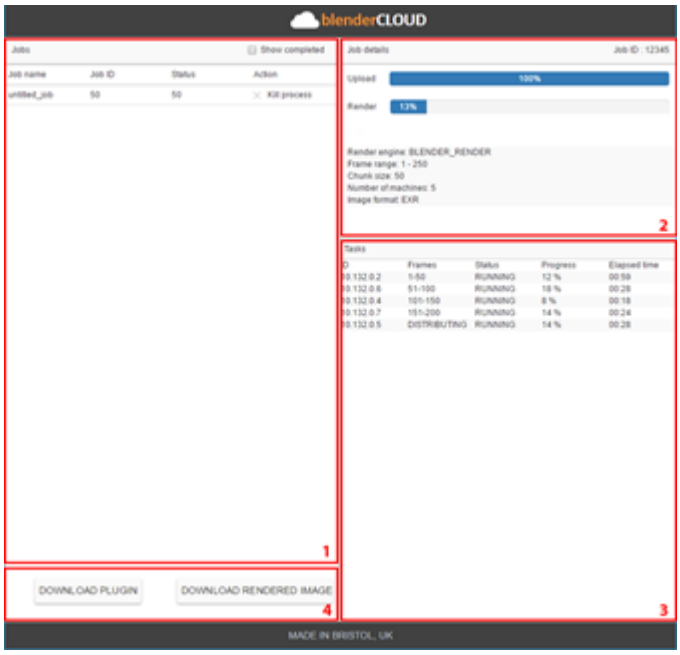

Figure 2: blenderCloud web interface

Figure 2 illustrates the web frontend in which users can monitor jobs that they have submitted. The web interface is split into 4 sections as annotated in the figure. Section 1 (top left) is the job list section, this gives a high-level overview of the jobs that have been submitted and the ability to terminate jobs. Clicking on a job row within this section reveals information about the job on the right-hand side. Section 2 (top right) gives a slightly more detailed overview of the job, giving the overall progress of the file upload and the render as well as displaying some of the render settings specified by the user. This was done so that users could review the settings that the job had been submitted with and quickly identify any issues that might mean the job should be terminated. Section 3 (bottom right) is the task section and gives information about the individual machines that are being used to process the job. Giving the status of the machine, information about the task in the form of the frames to be rendered, the progress of the render, and the compute time of the machine. Section 4 is the download section, where the user can download the latest blender cloud plugin as well as download a zip of the rendered images on completion of the render job.

B. *Backend*

To populate our frontend with the desired information, as outlined above, there needed to be some means of communication to the backend. Key features for the backend were: Real-time updates on the job progress; Real-time updates on the status and progress of the instances; Ability for users to terminate jobs and shutting down associated instances; ability to download a compressed file of the images. Three technologies were used to deliver these requirements:

Socket.io: A javascript library for real-time web applications. Enables real-time bidirectional event-based communication on every platform, device or browser. [5]. Enabled the display of real-time updated content on the front end of our web interface. This was used to update the status, progress and elapsed time for each task.

Express.js: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. [6]. Offered the primary means of communication between frontend and backend

Python requests: HTTP requests library for python. Used to submit jobs to our cloud system directly from the blender UI. Also used in a python script that runs alongside the render to report back information about the render's progress and status.

IV. CLOUD SYSTEM DESIGN

A. *Overview*

In this section, the design of our cloud system will be detailed making note of the cloud technologies utilized as well as the data flow through our system as outlined in figure 3

B. *Manager - Workers communication*

Our cloud design incorporated a manager-worker hierarchy where one virtualized instance, referred to as the 'manager' acts as a bridge between the front end and the power of GCE. Communications between the worker and manager entities are done via HTTP requests that follow RESTful API. Google's compute engine offers internal IP addresses that allow for communication between instance with ultra low latency on the same network.

C. *Role of Manager*

The manager plays an important role in our cloud system. The manager handles all communication between the user and the cloud as well as 'farming' out work to the worker instances. A full list of what the manager is responsible for, as detailed in figure 7, is detailed below:

- Retrieving the job configurations submitted by the user
- Retrieving the scene to be rendered in the form of a packed blend file
- Store the blend file in the GCS bucket to be accessed by worker instances
- Creating worker instances using an instance template image
- Distributing work between the workers and constructing each worker's render command
- Keeping track of the progress of each task to be fed to the front end as well as detect failures in the system
- Processing a chunk of the render itself, rather than wasting the virtual instance
- Serving a zipped version of the rendered images to the user

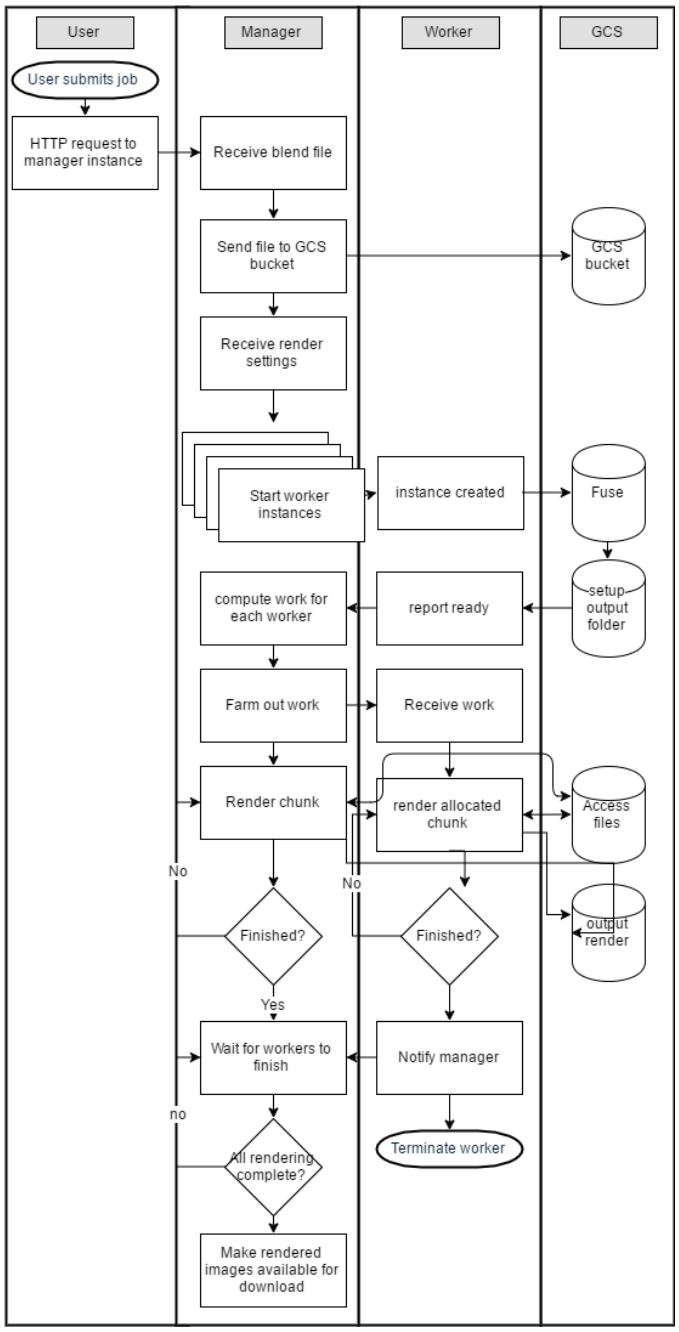

Figure 3: high-level overview of data flow through our cloud system

D. Life cycle of workers

When the manager creates the worker instances a number of startup scripts are run on the workers, readying them for processing. What the workers have to do, as outlined in figure 3, is outlined in list form below:

- Worker instances are created from an instance template image, which has all the required programs and dependencies installed
- Workers have to create a connection with the GCS bucket to ensure access to the blend file to be rendered as well as output the rendered images.
- Other low-level startup tasks are processed, for example, acquiring the instances internal IP address for identification when communicating between instances
- Task is received from the manager via HTTP request using internal IP address, offering ultra low latency
- Task is executed, running a python script that reports back to the manager the current status of the machine and the progress of the render over a socket.io listen server
- Once the task has been processed the worker will report to the manager and terminate itself

E. Storage

Google cloud storage is used to store objects that are accessed by both the worker and the manager instances. Anything that is accessed by only of one of these entities is stored within the persistent local storage offered by the virtual image.

Buckets are the basic containers that hold data in Google Cloud Storage (GCS). Unlike folders on a typical file system buckets cannot be nested and all buckets have the same namespace hence gaining access to objects can quickly become complicated.

To enable an instance to gain access to read and write to objects within a GCS bucket our system uses Cloud Storage FUSE. This is a FUSE adapter that allows users to mount Google Cloud Storage buckets as file systems in Linux or OS X systems. As well as provide ways for applications to upload and download google cloud storage objects using standard file system semantics [7]. Object storage names consist of a series of names (folder names/file name) and "/". GCS Fuse uses this and interprets the "/" character as the directory separator hence any objects with a common prefix are treated as if they are in the same folder. On startup of a new instance, the instance is ‘fused' to the GCS bucket as part of the start up script.

For storage of data that did not need to be shared between instances Google offers the ability to keep a persistent disk (the virtual disk of the server) which is maintained whilst the instances are offline. By doing this the time before the instances are ready to render is significantly decreased.

F. Scaling

01. Blender's render engines, blender render, and blender cycles are both compiled with OpenMP (Open Multi-Processing) which supports shared memory multi-processing. render frames are broken down into blocks, the size of which can be defined by the user, or automatically allocated by the render engine based on the hardware the engine is being run on. The number of blocks processed at any one time is defined by the number of cores present on the machine. Hence a CPU with 8 cores can provide an 8 times speed up over a single core CPU (In theory,

however, communication between cores could act as a bottleneck here). The key takeaway point from this is that no matter whether blender's render engines are being used on a single core or multi-core machine it will attempt to use 100% of the CPU if possible. This is the key reason why Google Compute Engine's auto scaling features were not desired for this project, as a single frame could result in the boot up of multiple VM instances with very little or even no speed up due to the communication needed between the instances when working on a single frame. Instead, a user defined scaling system was developed allowing the user to choose their desired amount of compute resources.

02. Compute engine's managed instance groups allow users to set up a group of identical servers. This allows the servers in the group to be easily referenced as well as allows for batch operations to be performed with minimal effort. Managed instance groups require an instance template from which the identical instances in the group are created. Given our system infrastructure design of a single manager instance and multiple identical worker instances, this made instance groups a perfect fit for our needs. Managed instance groups were created with the desired number of machines as submitted by the user with the use of the Gcloud command line tool.

03. Google Compute Engine uses operating system images to create the root persistent disk for instances in a GCE project. Images contain a boot loader, an operating system, and a root file system [5]. Both the worker and manager instances were derived from a ubuntu 14.4.5 image. Both have the latest version of blender installed and respective server code/dependencies.

V. RESULTS AND PERFORMANCE

In this section, the tests carried out on our system will be detailed and the results from these tests will be presented and discussed.

The "Animated Ocean Scene" from sketchfab [3] is used to test the performance of the blenderCloud system. 100 frames of animation will be simulated and rendered under four different job configurations, differing only the number of instances used to carry out the job. These configurations are 1, 2, 4 and 8 single core virtual instances.

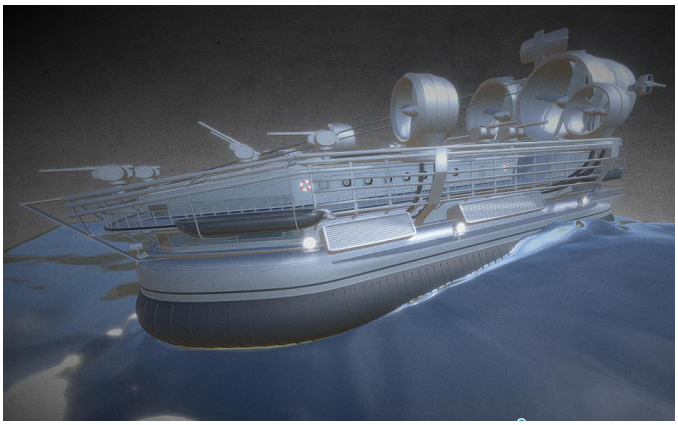

Figure 4: Single frames render from the "Animated Ocean Scene" *used in the testing of the scalability of our cloud system*

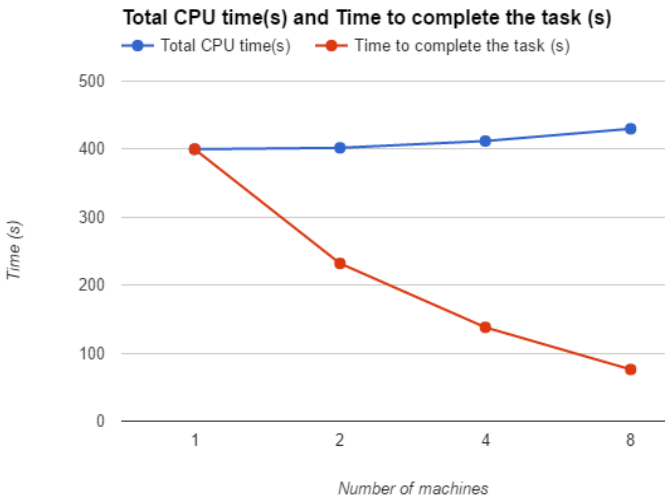

Figure 5: performance scalability

Figure 4 shows a preview of a single frame rendered on the blendCloud render farm. Figure 5 is the result of our testing. This graph displays the total amount of CPU time to carry out all of the tasks (the rendering of each distributed chunk) plotted against the time for the job to finish.

It is key to note that testing on the cloud is subject to a lot of variables that could cause inaccurate results being read for performance. For example, due to the virtualized nature of cloud instances, many 'virtual' machines are hosted on a single physical server. The machine that is assigned to the job might, therefore, be shared and affected by other users interactions with the machine. To mitigate the effect this might have on our results, each configuration was tested several times and an average was taken in an attempt to identify and exclude anomalous results.

From these results, it is clear to see that as the number of virtual instances used to process the job increases, the time taken to finish the job decreases. It can also be seen from

figure 5 that the time to complete the task does not decrease linearly. For example, if four times as many instances are used this does not result in a four times speed-up. This is due to a number of reasons, but it is important to note that during our testing it was seen that the manager instance was always the bottleneck. The manager instance, in all cases, took the longest to finish and thus was the determinant of the ‘time to complete the task'. This is due to the amount of work the manager has to deal with alongside the rendering of its allocated chunk. This work being the communication between instances as well as hosting the server and communicating with the front end. Thus, it is likely that not including the manager instance for processing could prove favorable in speeding up the system further. The motivation for using the manager was to cut costs by utilizing all unused compute power.

On starting new instances there is a period of time where startup scripts and resources are provisioned. During this time the instance can enter the ‘running' state and thus the CPU time is added to the overall count whilst not actually rendering. This is the cause of the increase in total CPU time when increasing the number of instances, as shown on the graph in figure 5. There is more of this redundant up time, however, the period of time is relatively small, on average around 10 seconds, which is comparatively small to the amount of time used to process a large render job.

VI. Conclusion and Future Developments

It is clear to see that doing simulation baking and rendering in the cloud is a very viable and cost effective way to significantly speed up computationally expensive CGI processes. Our system only scratches the surface of what is possible using the cloud for such tasks. There are a number of features that were desired in the blenderCloud system but due to time constraints could not be included. These could serve to further flesh out the system and make this into a viable commercial render farm:

- Expanding parallel simulation baking to all simulations offered by blender
- Queuing system that allows for large numbers of jobs to all be submitted at once and processed when resources are available
- Selecting and customizing the power of the instances used to process jobs.

REFERENCES

[1] JR. Narayanam, "Ten Features that make Google Compute Engine (GCE) better than AWS", 9-December-2013. [Online]. Available at: https://yourstory.com/2013/12/google-compute-engine-better-than-aws/. [Accessed: 15 - Dec - 2016]

[2] Z. Bjornson, "AWS S3 vs Google Cloud vs Azure: Cloud Storage Performance", 29-December-2015. [Online]. Available at: http://blog.zachbjornson.com/2015/12/29/cloud-storage-performance.html. [Accessed: 10 - Dec - 2016]

[3] 3D Haupt, "Animated Ocean Scene", 2016 [Online]. Available at: https://sketchfab.com/models/0ba0611f6e0e4c038ec676caf9ef5660. [Accessed: 02 - Jan - 2017]

[4] Docs.angularjs.org. (2017). *AngularJS*. [online] Available at: https://docs.angularjs.org/guide/introduction [Accessed 5 Jan. 2017].

[5] Socket.io. (2017). *Socket.IO*. [online] Available at: http://socket.io/ [Accessed 5 Jan. 2017].

[6] Expressjs.com. (2017). *Express - Node.js web application framework*. [online] Available at: http://expressjs.com/ [Accessed 5 Jan. 2017].

[7] Google Cloud Platform. (2017). *Cloud Storage FUSE | Cloud Storage Documentation | Google Cloud Platform*. [online] Available at: https://cloud.google.com/storage/docs/gcs-fuse [Accessed 5 Jan. 2017].

[8] Google Cloud Platform. (2017). *Images | Compute Engine Documentation | Google Cloud Platform*. [online] Available at: https://cloud.google.com/compute/docs/images [Accessed 5 Jan. 2017].

[9] Takahashi, D. (2017). *How Pixar made Monsters University, its latest technological marvel*. [online] VentureBeat. Available at: http://venturebeat.com/2013/04/24/the-making-of-pixars-latest-technological-marvel-monsters-university/ [Accessed 6 Jan. 2017].

[10] SearchCloudComputing. (2017). *What is cloud computing? - Definition from WhatIs.com*. [online] Available at: http://searchcloudcomputing.techtarget.com/definition/cloud-computing [Accessed 6 Jan. 2017].