

# Web-based 3D animation software

**Indy Chang Liu**

*ThinkingCactus Ltd.*

indy.liu@thinkingcactus.com

## **Abstract**

We are building a web based application for 3D animation creation, similar to what Google sketchup is to 3D modelling, allowing users to enjoy animation creation without download, installation and concerns with graphics hardware.

Users customise 3D scenes and chronological events on the web front end, then the task is processed and rendered by the servers remotely. The resulting animation file is hosted online for viewing, sharing or downloading. In the text we will discuss the structure with emphasis on rendering processes.

## **1. Introduction**

To make life easier for people who has creative ideas for 3D animations but lack 3D modelling skills, our development guideline is around decoupling modeling from story telling and then decoupling story telling from rendering. The resulting system design consists of several major components:

- web interface
- task queuing server
- resource management
- rendering process
- video hosting

Interaction between components are illustrated in Figure 1.1. The majority of the components of this project are programmed in Python. The code is readable, clean, reusable and robust. It is easy to stay agile and to make modular improvements. Python-ogre facilitates pythonic object oriented code while running fast and stable by handling heavy lifting graphics processes using the underlying C++ code. The latest python-ogre features such as compositors, render to large textures and full anti-aliasing on those textures enables us to render high quality animations ready for TV and movie screens with rich effects.

## **2. Components**

### **2.1. web interface**

The web interface facilitates user interactions such as changing 3D character's face, mix costumes, changing dialogues and such. These user commands are encapsulated in an XML file which defines the 3D scene as well as a time line of actions with meta-data such as user ID, time information etc.. The XML file will be validated by

a server before being sent to the task queue manager. See Figure 2.1 for a sample excerpt.

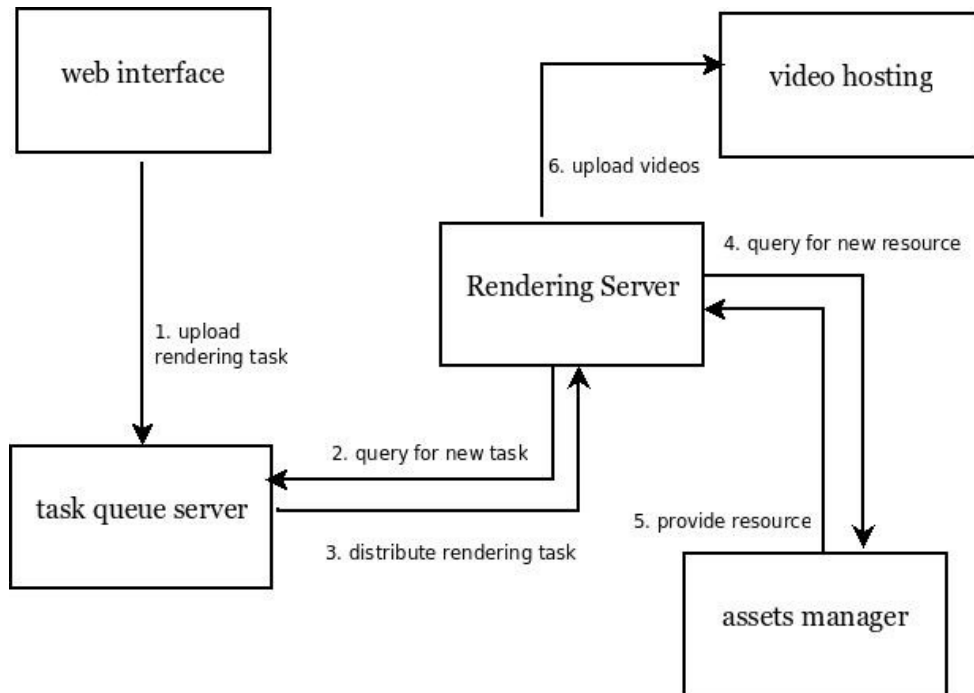


Figure 1.1. web-based animation interaction diagram

```

...
<actors>
  <actor id="actor1" name="Amy">
    <position x="-35.7" y="-10.1" z="-5.7" />
    <orientation qw="0.0" qx="0.10" qy="0.0" qz="0.9" />
    <attachments>
      <attachment id="actor1_f3_panties0" name="f3_panties0">
        <f3_panties0 b="1.0" g="0.501960813999" r="1.0" />
      </attachment>
      <attachment id="actor1_f3_tank_top0" name="f3_tank_top0">
        <f3_tank_top0 b="0.26" g="0.264" r="0.26" />
      </attachment>
    </attachments>
    <faceDefinition>
      <elements>
        <part cartoon="female_face_cartoon.svg" />
        <part face="female_face_02.svg" />
      </elements>
      <states />
    </faceDefinition>
  </actor>
</actors>
<actions>
  <actor id="actor1">
    <skeletonAnimation length="2.98" speed="1.0" time="0.0">
      <animation animationName="LB:air:somersault0" length="0.57" speed="1.0" time="0.0" />
    </skeletonAnimation>
    <skeletonAnimation length="2.0" speed="1.0" time="3.8">
      <animation animationName="LB:air:RL_kick0" length="2.09" speed="1.0" time="3.8" />
    </skeletonAnimation>
  </actor>
</actions>
...
  
```

Figure 2.1. Sample Scene XML

## 2.2. The task queue server

The task queue server is responsible for coordinating tasks among multiple remote rendering servers. The server keeps a queue of user uploaded scene scripts, ready to be rendered. When the next rendering server (worker) is idle, it will query the queuing server for a new job. The server then distributes the job to that rendering server and keeps track of the status and result of the process.

Celery (<http://celeryproject.org/>) is chosen as the task queue manager. It uses AMQP messaging to distribute tasks among multiple computers and multiple processes. It supports sub-task multiprocessing and task status feedbacks so we can have one rendering task divided into several sub-tasks and give user status reports along the way. Exception keeping, automatic worker computer replacement and fail-retry cycles make sure the process is fail safe.

## 2.3. The render server

Decoupling the creation process from the rendering process enabled us to easily switch between different rendering mechanisms and libraries. We chose python-ogre (<http://www.python-ogre.com/>), a python wrapping of the open source Ogre3D graphics engine including its essential plug-ins for the following reasons:

- cross platform - the server needs to be able to run on linux;
- shader support - the engine supports latest shader versions;
- community activity - measured by the contents on project home page, wiki page, community forums and mailing list;
- flexibility - plugin based structure to integrate with different physics, AI libraries as well as easily altering low level process such as space partition algorithm, rendering pipeline and render target customisation;
- maturity – it is used for many successful commercial projects.

As illustrated in Figure 2.3, Upon receiving a rendering task, the server will validate the task and make sure it has the latest assets, 3D models, textures, sound etc. required for the rendering. If additional resource is needed, the rendering server will query and stream them from the resource database. After which the 3D scene is constructed and rendered frame by frame to a texture. Each frame of texture will have its bitmap data converted to a binary string and piped down the video encoder. The video encoder will index, compress and multiplex the video stream with the audio stream and creates an animation file. Finally the file will be uploaded to the Amazon S3 server using the boto python-S3 interface (<http://code.google.com/p/boto/>) for the user to watch, share or download.

Each step along the way, if an exception is raised, it will be reported to the task manager and stored in a database. The task manager can then decide whether to retry executing the task for a number of times before aborting and reporting it the web front end and the admin.

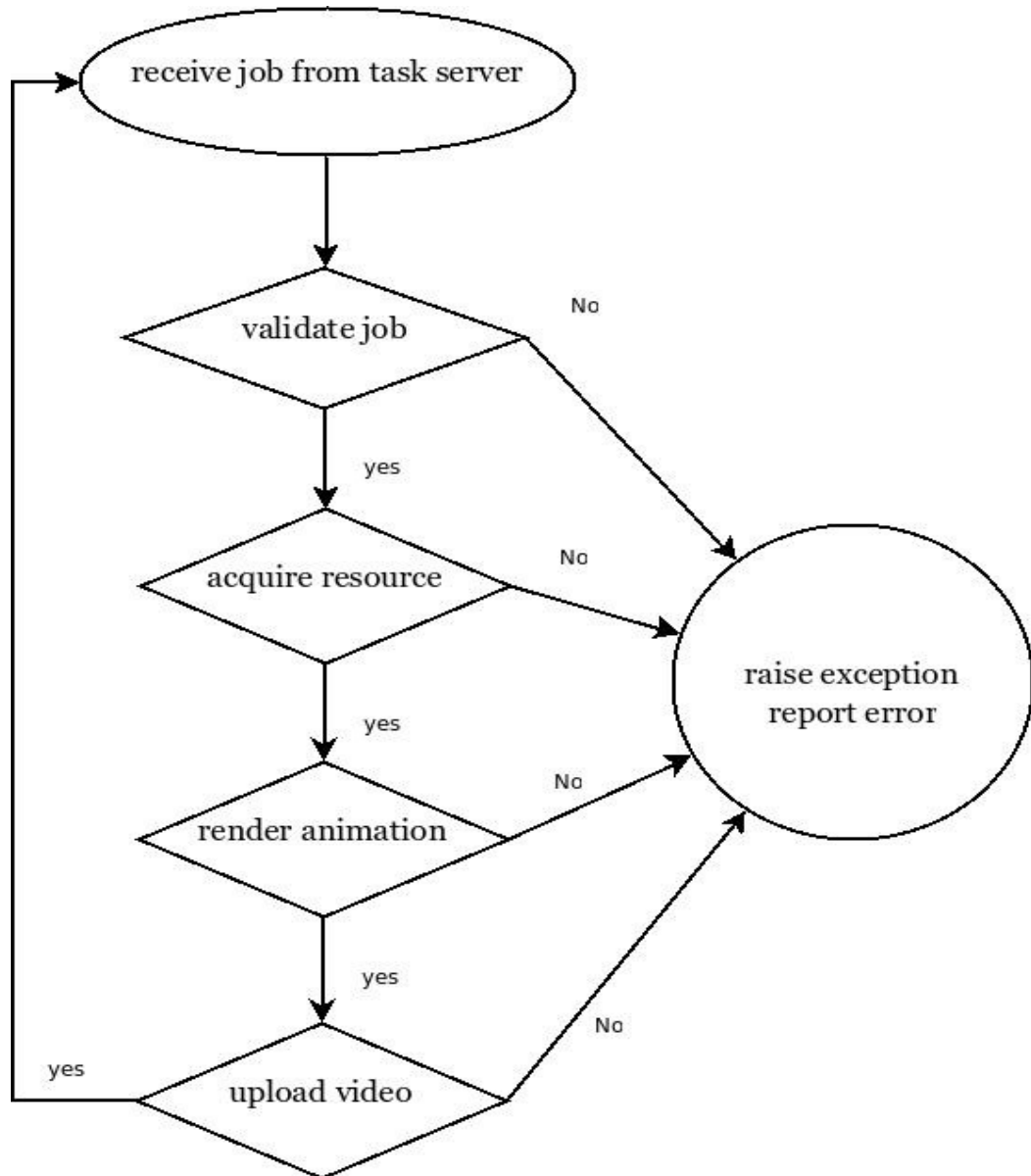


Figure 2.3. Render Server work flow

### 3. Discussions

The benefit of this structure, apart from being robust and easy to maintain, comes in three main aspects:

First of all, as long as the rendering server and the queuing server have consistent interface we can explore different technologies for the front end, satisfying users of a full spectrum of skill requirements and hardware capability. The advanced users will use a fully functional 3D scene editor and animator on their desktop. Other users will use a Flash or Javascript based web page to make changes to the scene. Basic HTML or text based user interface can be used to satisfy users of hand held devices or cellphone texting services. After creation, the rendering jobs will be sent through the same rendering process. And even cellphone users will be able to render

animations with true physics, particle systems and high end shader effects like texture mapping, cel-shading or high dynamic range rendering.

Secondly, the assets server is kept separate from the rendering and animation creation process. This allows seamless asset updates from contents suppliers and modelling artists. As long as the server can retrieve the new assets required for the render, we can keep improving the look and feel of the models and scenes. This way we facilitate agile, on-going maintenance and improvements of art contents. Which in turn helps us cater the changing taste and different needs of our users.

And finally, the back end rendering server can easily be upgraded as graphics technology progresses. Based on market feedback, we can change to using a popular commercial 3D engine such as CryEngine 2. A ray-tracing based engine can be implemented to render movie quality animations for corporate clients and animation studios.

#### **4. Reference**

Boto - <http://code.google.com/p/boto/>

celeryD - <http://celeryproject.org/>

Python-ogre - <http://www.python-ogre.com/>