OULU UNIVERSITY OF
APPLIED SCIENCES

Joni Mikkola

# COLLADA IMPORTER FOR REALXTEND

# COLLADA IMPORTER FOR REALXTEND

Joni Mikkola

Bachelor's thesis

November 2011

Degree Programme in Information Technology and Telecommunications

Oulu University of Applied Sciences

# ABSTRACT

OULU UNIVERSITY OF APPLIED SCIENCES
Degree programme
Information Technology and Telecommunications

Line
Software developer

Commissioned by
CIE – Center for Internet Excellence

ABSTRACT
Thesis Pages + appendices
B.Sc. 59 + 4

Date
09 October 2011

Author
Joni Mikkola

Thesis title
Collada Importer for RealXtend

Keywords
collada, virtual worlds, 3d, animation, opengl, ogre

The purpose of the thesis was to implement a Collada importer for the realXtend virtual world platform. Collada is an interchange file format for interactive 3D applications, hence being a reasonable format for realXtend. The implementation should be separated as much as possible from the core code of realXtend in the way that it works as an individual component.

To achieve this, the Open Asset Import library was used for handling Collada data. In addition to Collada, Open Asset Import library offers support for many other 3D formats, too.

The resulting Collada importer was able to handle Collada among other formats. The main components imported from the Collada format are meshes, skeletons, materials and in some cases textures. From these components a complete Avatar could be formed, for example.

# TIIVISTELMÄ


OULUN SEUDUN AMMATTIKORKEAKOULU — TIIVISTELMÄ

Koulutusohjelma: Tietotekniikan koulutusohjelma
Opinnäytetyö: B.Sc. Sivut + liitteet: 59 + 4

Suuntautumisvaihtoehto: Ohjelmistojen kehitys
Aika: 09.10.2011

Työn tilaaja: CIE – Center for Internet Excellence
Tekijä: Joni Mikkola

Työn nimi: Collada Importer for RealXtend

Avainsanat: collada, virtual worlds, 3d, animation, opengl, ogre

Työn tavoitteena oli toteuttaa Collada-tuki realXtend-alustalle. Collada on formaatti interaktiivisille 3D-ohjelmille ja tästä syystä järkevä valinta realXtend-alustalle. Toteutus pitäisi erottaa realXtendin ydinkoodista niin paljon kuin mahdollista siten, että Collada-toteutus toimisi itsenäisenä komponenttina.

The Open Asset Import -kirjastoa käytettiin Collada-tiedostojen käsittelyyn. Kirjasto tukee Colladan lisäksi lukuisia muitakin formaatteja.

Tuloksena Tundra osaa käsitellä muun muassa Collada-tiedostoja. Meshit, skeletonit, materiaalit ja tekstuurit olivat pääkomponentit, joita Collada-formaatista ladattiin. Näistä komponenteista pystyttiin muodostamaan tarvittaessa vaikka Avatar.

# SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| 2D / 3D | 2 / 3 Dimensions |
| Alpha channel | Alpha channel defines the transparency of an image |
| API | Application Programming Interface |
| AssImp | Open Asset Import Library |
| Avatar | Graphical representation of a user in virtual world |
| Blender | Free and open-source 3D computer graphics software |
| C/C++ | General purpose programming languages |
| Collada | Interchange file format for interactive 3D applications |
| GLUT | OpenGL Utility Toolkit |
| Indice | 3D vector containing position or a point for x, y and z axes |
| IRC | Internet Relay Chat |
| JPG | Image format, short of Joint Photographic Experts Group |
| LLUDP | Low Level User Datagram Protocol, internet protocol |
| Ogre | Object-Oriented Graphics Engine |
| OpenGL | Open Graphics Library |
| PNG | Image format, short of Portable Network Graphics |
| RAM | Random Access Memory |
| RGB | Red, green and blue color components |
| Redmine | Project management tool |
| XML | Extensible Markup Language |

# CONTENTS

# 1 INTRODUCTION

The purpose of the Bachelor's thesis is to create support for a Collada file format in the Tundra platform of the realXtend project. However, the 3D file format support of Tundra is very limited because of the tight integration to Ogre, and therefore there is a need for a format that is an open standard and includes all elements necessary for creating a virtual world, such as Collada is.

The focus is to create the required functionality in the way that all data processing of the Collada files is transparent to the end-user. Some parts of the implementation are written to the very core of realXtend and therefore require much reviewing and testing.

First, a little insight is given of the realXtend project and how it differs from other existing virtual world platforms. Then, an insight of the tools used in this work is provided by describing them shortly as well as the structure of Tundra. There is great amount of terminology behind 3D, which is explained in order to help the reader follow the text.

From the basics the thesis continues to the Collada implementation part where the problems encountered during the project are explained. In addition, the methods for solving these problems are represented.

The goal of this thesis is to provide a fully working Collada importer for Tundra.

# 2 DEVELOPMENT PROCESS

In chapter 2, the development process is explained in order to give the reader an easier understanding of the steps taken for achieving this implementation.

The first step of this thesis was to create a task of the thesis to the Redmine. Redmine is used at CIE as a project management tool. After each day of working, the work done during the day was updated into the Redmine. If the work did not fit under any existing tasks, a new task was created. (Figure 1.)

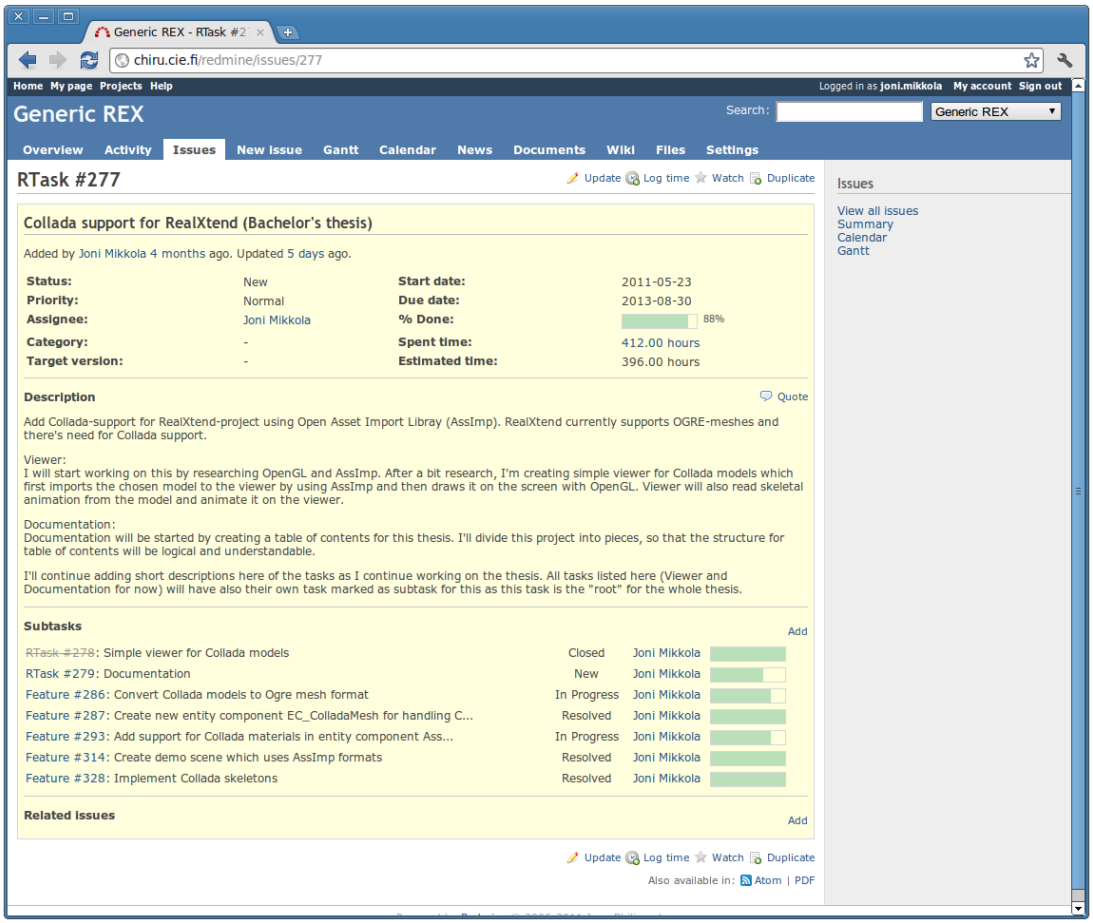


*FIGURE 1. Redmine project management tool*

The initial research started by figuring out how OpenGL and the Open Asset Import library work. This was done because the Open Asset Import module

will be used in Tundra later in order to achieve the Collada implementation. The information related to AssImp and OpenGL was searched from Google.

When the basics of AssImp and OpenGL were understood, a Simple OpenGL Viewer was created, which took advantage of both AssImp and OpenGL. The Viewer was done with a QT Creator. The purpose of this was to get some insight into how Collada files are practically converted and then rendered with OpenGL.

Furthermore, when the Viewer had been done, it was decided that the actual implementation work could be started as the theory part was somewhat figured out. Tundra already had a module called Open Asset Import, which was done by Rex Community some time earlier.

Initially, the Tundra and Open Asset Import modules were completely separated from each other, in other words there was no implementation between them. In addition, the Open Asset Import module was much outdated; it did not have support for materials, compressed textures and skeletons (figure 2).


*FIGURE 2. Tundra and Open Asset Import module without implementation*

Tundra is a fairly large project, containing over 100 000 lines of code and, in addition, it does not contain any well done documentation for developers. For this reason, some researching of Tundra had to be done, too, to determine for which part of the Tundra the implementation should be done.

Naturally, the whole process required a large amount of testing. The testing was conducted constantly. Each time some part of the code was changed, Tundra was started and tested if the code worked properly.

The implementation was then done in the Asset API, which does all the file handling in Tundra. In short, if Collada file is loaded in Tundra, Tundra sends the file over to Open Asset Import module, which then converts the data into mesh, materials, textures and skeletons, and then the converted data is transferred back to Tundra (figure 3).

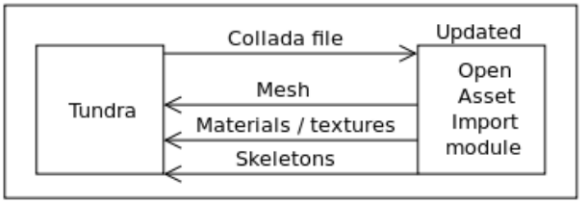


*FIGURE 3. Tundra and Open Asset Import module implementation*

Throughout the thesis, documenting, researching and testing was done. The figure 4 below represents how the time was divided between the tasks in the thesis.

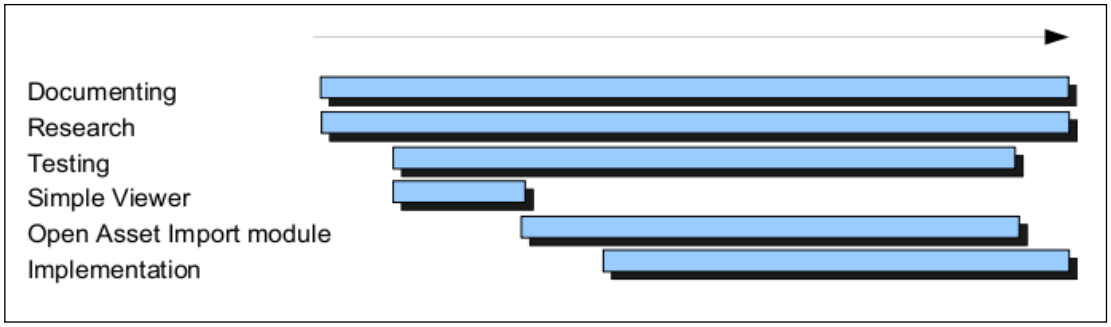


*FIGURE 4. How time was divided between the tasks*

# 3 VIRTUAL WORLDS

Some most commonly used virtual world platforms and technologies used in the thesis are presented in this chapter. The realXtend virtual world platform is presented more thoroughly, because it's the target platform in the thesis.

There are many definitions for describing virtual worlds, for example by Mark K. Bell: "A virtual world is a 'synchronous, persistent network of people, represented as avatars, facilitated by networked computers'". (27.)

In the future, virtual worlds could be used for numerous different purposes, such as educational, gaming, medical or meeting other people virtually. Technology is progressing rapidly enabling more and more diverse possibilities for creating real life simulators. Thus, it is frequently predicted that virtual worlds could simulate life within x years. However, there are problems to be solved before reaching that point.

Some online games can also be considered as virtual worlds. Consider World of Warcraft, for example. It offers a persistent network with millions of players simultaneously connected to the world. Each player has created an avatar for themselves which then can interact with each other. The definition of the virtual world does not take into account whether the world scenery is fixed or not, as is the case with the World of Warcraft.

In this chapter Tundra, Second Life and Sirikata virtual world platforms are presented as well as the realXtend project overall. Tundra and Sirikata platforms are open-source whereas Second Life is proprietary. Second Life offers more fixed world editing thus being much more limited in sense of dynamicity than Tundra or Sirikata (8).

## 3.1 RealXtend Project

RealXtend Foundation was originally started by Juha Hulkko, a private investor in Finland in 2007 after he had seen the business possibilities it offered.

He thought that if virtual worlds someday are going to be important, they should not be controlled by a single company, but rather be a free open source solution. (6.)

In 2007, the OpenSim compatible client server solution was developed. OpenSim is an open-source server platform for hosting virtual worlds. It is compatible with Second Life client. (7.)

Then realXtend association was founded in April 2011. The Association coordinates the collaboration and develops interest in realXtend. For example, the Association can suggest projects to the Foundation and the Foundation then can collect money from big companies and fund projects. (6.)

RealXtend is a common name for a platform for creating open-source virtual worlds. The realXtend project consists of two centerpieces: 3D virtual world viewer, codename Naali, and an integrated suite of virtual world servers, codenamed Taiga. Taiga and Naali are the first outcomes of the realXtend foundation. The next generation virtual world product of realXtend is Tundra, being both server and client. (10.)

### 3.1.1 Naali, Virtual World Viewer

Naali is the open source cross-platform virtual world viewer of realXtend. Naali is intended to be used with Taiga (10).

The link between Naali and Taiga is the low-level LLUDP protocol, which is a proprietary protocol developed by Linden Labs for virtual world communications (11).

### 3.1.2 Taiga, Virtual World Server

Taiga is a set of interoperating servers designed to be viewed with the Naali viewer. Taiga is an unmodified OpenSim server, with certain additional modules preconfigured to add the realXtend functionality. (10.)

### 3.1.3 Tundra, Virtual World Server and Client

Tundra is currently the main focus in the realXtend project (figure 5). Tundra shares the same codebase with Naali. Tundra is a branch diverged from Naali and it implements both a server and a client, instead of being only a client. (5.)

Tundra is an application development framework. Instead of providing a single end-user application like Naali, Tundra aims to be an SDK/engine for developers. This means that no in-world functionality is assumed or hardcoded (world sizes, scene content, client appearance or behavior, user interface), but the server which it is connected to, defines all these aspects. (5.)

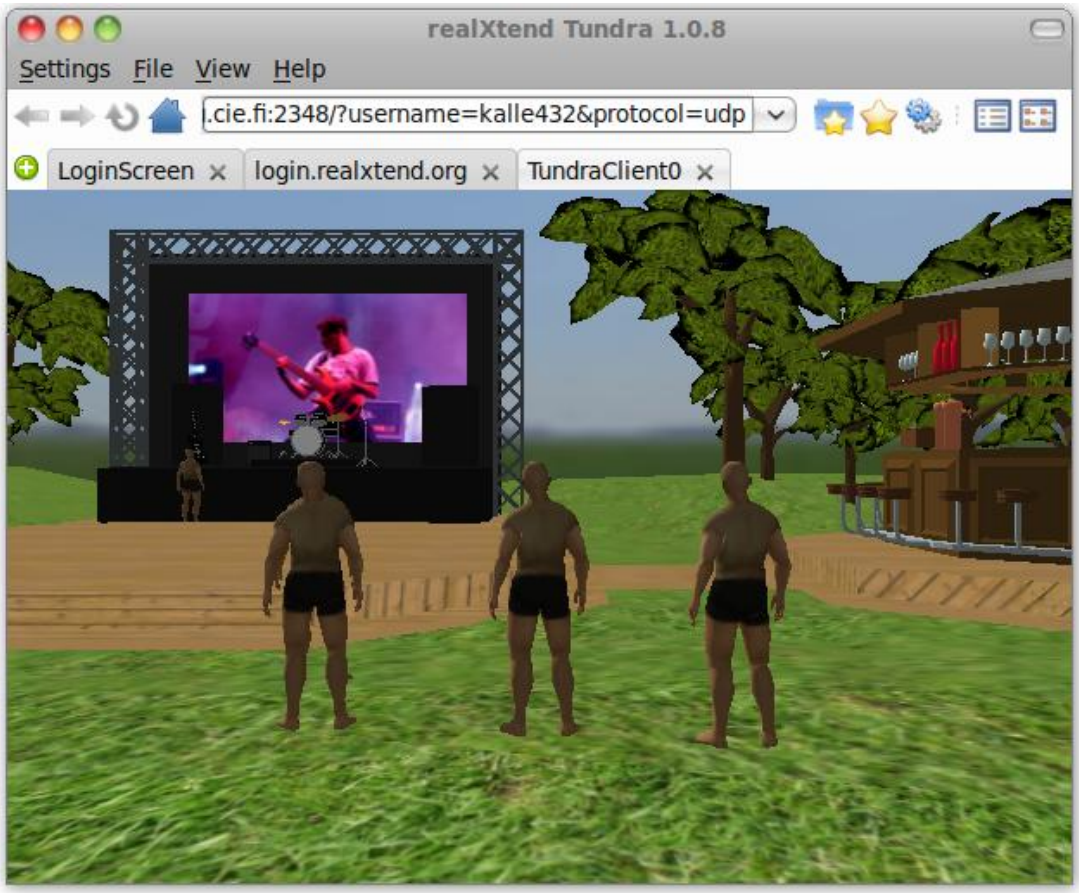


*FIGURE 5. realXtend Tundra v1.0.8 client running*

**Structure of Tundra**

Tundra consists of a core and modules. The core handles the low level calculations, providing a solid base for higher level modules to be built on (figure 6). All the modules are implemented in top of the core. This provides a more maintainable way for creating and modifying features. In addition, the entity components also provide a more diverse amount of functionality. (25).

The main third party libraries that Tundra is dependent on are QT, Bullet Physics, Ogre, Python, kNet and Boost. (25).

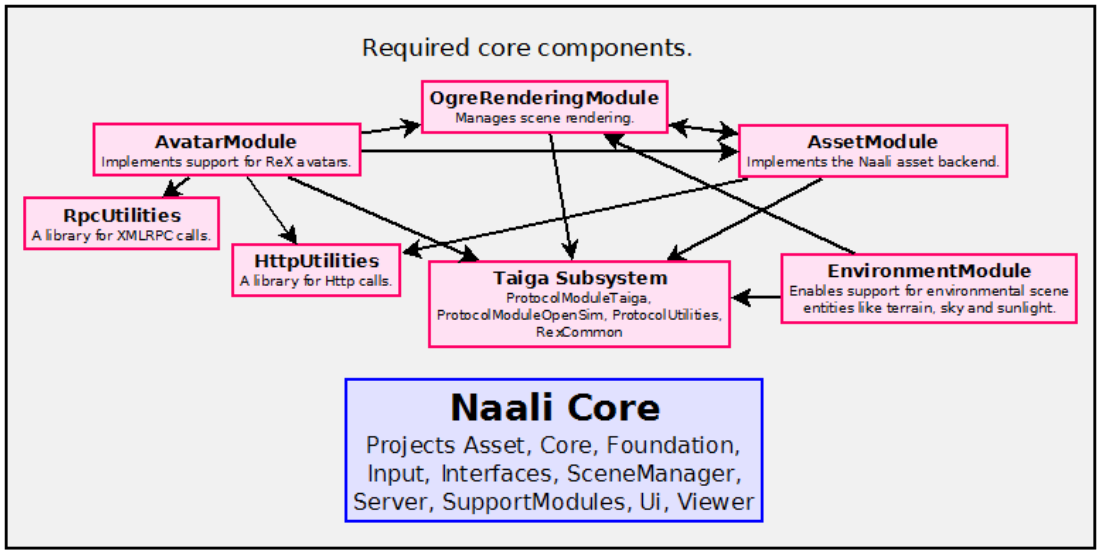


*FIGURE 6. Core components of Tundra*

For a developer it is optional to choose which modules to use with the build, thus limiting the necessary dependencies. Though, there are some modules that are required to be used with the core.

Modules for their part have their own dependencies, i.e. modules are dependent on some third party libraries, and some modules can even be dependent on some other modules making it a fairly complicated network of dependencies.

When a developer starts to implement some kind of functionality in Tundra, entity components or modules are the correct way to do it. The core code should not be compromised, as the core should already provide the necessary API for achieving the needed functionality. The core implements all the basic functionality, such as file transferring, rendering, scene handling etc.

In this thesis, the Collada conversion part of the implementation was done in the Open Asset Import module which is an optional module and is dependent on AssImp library. After that, the real implementation was done in Asset API, which is one of the core elements, hence making the core element dependent on the Open Asset Import module and AssImp library.

**Entity Components**

Entity components are used in Tundra for building extensible scenes. The entities are unique identities, with no data or typing. They aggregate components, which can be of any type and store arbitrary data. (8.)

The entities are individual components in the scene. An entity consists of components which define the data and the behavior for the entity. The components contain attributes, which define the essential behavior and in the end create the presence of the entity. The entities can contain any number of components of any type, even several instances from same component. It is up to the component to make sure this causes no problems. (12.)

For example an avatar to function avatar-like, it needs to catch events from the mouse and keyboard, play skeletal animations and move as a human would. In order to achieve all before-mentioned, several components have to be added. All necessary components for acquiring an Avatar-like entity are represented in the next chapter. See figure 7 for an Avatar entity structure.

**Avatar Example**

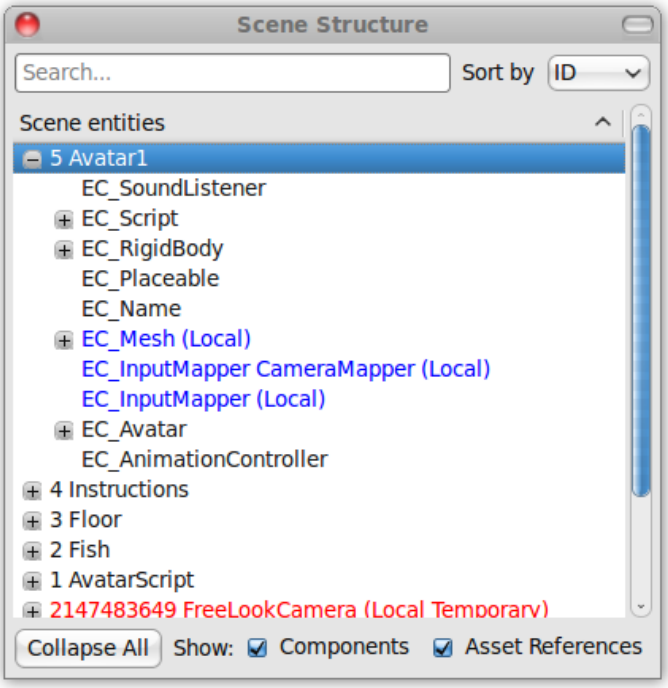


*FIGURE 7. Avatar entity with components*

- EC_Avatar defines the basic structure for an avatar. Mesh, materials, skeleton and animation elements are provided by an XML file which EC_Avatar references to.

- EC_SoundListener provides a sound listener position for the in-world 3D audio.

- EC_Script provides a mechanism for adding and running scripts in entities within Python or JavaScript languages. In this Avatar entity, the script chooses which animation to play when a scripted button is pressed and where to move the avatar etc. The real functionality is defined in the scripts.

- EC_RigidBody provides a rigid body for enabling collisions. The collisions use the Bullet Physics library.

- EC_Mesh defines the Avatar mesh, skeleton and materials.

- EC_Placeable provides a proper in-world transformation, position, orientation and scale.

- EC_Name defines the name of an avatar.

- EC_InputMapper and EC_InputMapper CameraMapper provide an event handler for the keyboard and mouse inputs. CameraMapper in this context is giving a third-person view from behind the avatar mesh. The first EC_InputMapper handles the input for the avatar mesh, walk and run, and the second EC_InputMapper moves the camera as the user gives an input.

- EC_AnimationController provides tools for playing the animation data from the Ogre skeleton file.

### 3.1.4 Community, user-oriented and developer-oriented channels

There are user-oriented and developer-oriented channels. Communication happens through mailing lists and IRC. The developers of realXtend are spread throughout the whole world, although most of them are located in Finland. Discussions of bigger topics are being written in the mailing lists, but in IRC there is a continuous discussion about the development of realXtend. Because IRC is providing such a fast way of contacting realXtend developers, it is a good place to ask small questions concerning realXtend in hope of a quick response.

As with the thesis, some feedback has occasionally been asked of the Collada importer through the IRC channel #realxtend-dev. The developers and community of the Open Asset Import have also been contacted by their forum during the thesis, seeking help for the usage of the library.

## 3.2 Second Life, Virtual World Platform

Second Life is the most popular virtual world platform reaching over 20 million users. See the Second Life client figure 8. The SL platform is largely predefined and hardcoded whereas Tundra aims for a more dynamic content creation. Second Life is a proprietary software and thus does not offer source code for developers. (8.)

Despite the fact that Second Life is proprietary, open source clones are also provided, realXtend being one of them (7). These implementations always lag behind due to the fact that Second Life is a closed code and thus the implementations required reverse engineering of the Second Life binaries which take some time and are never exact.


*FIGURE 8. Second Life*

## 3.3 Sirikata, Virtual World Platform

Sirikata is a BSD-licensed open source platform for creating virtual worlds. The Sirikata platform has grown out of several years' research at Stanford University, and the current ambition is to expand it into a fully community run open source project. (13).

KataSpace is a client running on the Sirikata platform. KataSpace uses WebGL and HTML5 for rendering and interactions (Figure 9).

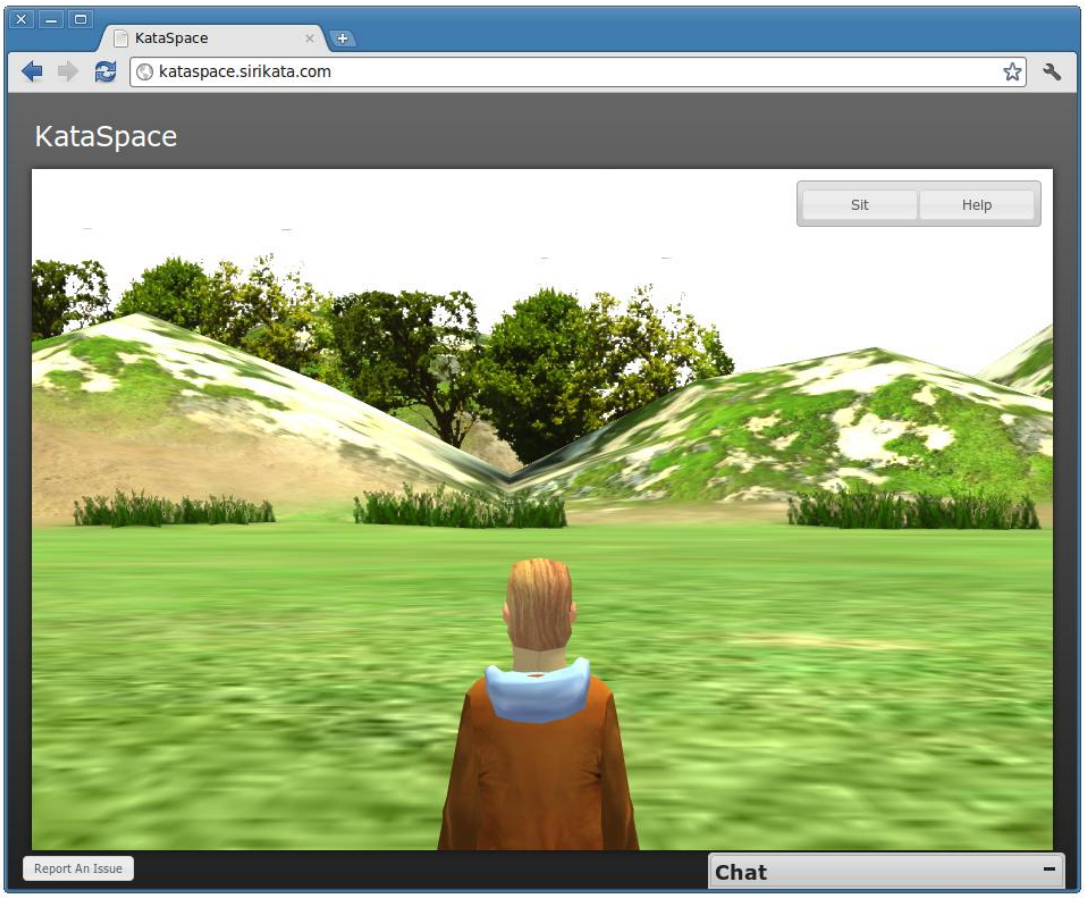


*FIGURE 9. KataSpace ran on Sirikata virtual world platform*

# 4 TECHNOLOGIES

## 4.1 Khronos Group, open standards maintainer

Khronos Group is a non-profit organization which maintains various different open standards. It aims for providing open standards for graphics and parallel computing. Some of the maintained standards are OpenGL, OpenCL, Collada, OpenVG and OpenMAX. The two most important standards from the perspective of this thesis are OpenGL and Collada. (14.)

The Khronos group was originally founded in 2000 by a number of media centric companies, including ATI Technologies, Discreet, Evans & Sutherland, Intel Corporation, NVIDIA, Silicon Graphics (SGI) and Sun Microsystems. (15.)

### 4.1.1 Collada

Collada is maintained by Khronos Group. Collada comes from the words Collaborative design activity. Collada defines an open standard XML-based schema for making it easier to transport 3D assets between applications. (9.)

Collada was originally created at Sony Computer Entertainment. Afterwards it became a property of the Khronos Group, a member funded industry consortium which now shares the copyright with Sony. The standard has been adopted by dozens of commercial game studios. (9.)

### 4.1.2 OpenGL

OpenGL is also maintained by Khronos Group like Collada. OpenGL is a widely supported open standard for portable and interactive 2D and 3D graphics applications. (15.)

RealXtend uses OpenGL for rendering graphics under Linux and OpenGL ES in MeeGo devices.

Direct3D could be counted as OpenGL's rival, but it is only available for a Windows platform. Nowadays almost all games use Direct3D as a 3D renderer for its performance and because it has many advanced features supported by graphics cards. However, the newest OpenGL offers the same capability as DirectX 11 primarily due to the advanced tessellation of the OpenGL v4. (16.)

The terms material, texture, mesh and animation are brought up numerous times further in the thesis, and therefore these terms are explained below. Although animation is not really a part of OpenGL, it is also explained.

**Material**

Material defines the visual appearance of the rendered model. Material information can contains such data as colors, lightning and reference to the texture.

**Texture**

Texture is an image that is mapped on the surface of a shape or polygon. Texture knows its position by reading UV coordinates from the surface. The letters UV are the letters that describe the 2D mesh because x, y and z are already in use for describing the position of the mesh in the 3D space.

An example of how to create a cube is illustrated in figure 10. The image on the right represents a cube positioned in top of a texture. In the image in the middle, texture reads its UV positions and wraps itself around the cube. Finally in the image on the left, the cube is fully wrapped by the texture, hence being ready to be pushed in the rendering pipeline.

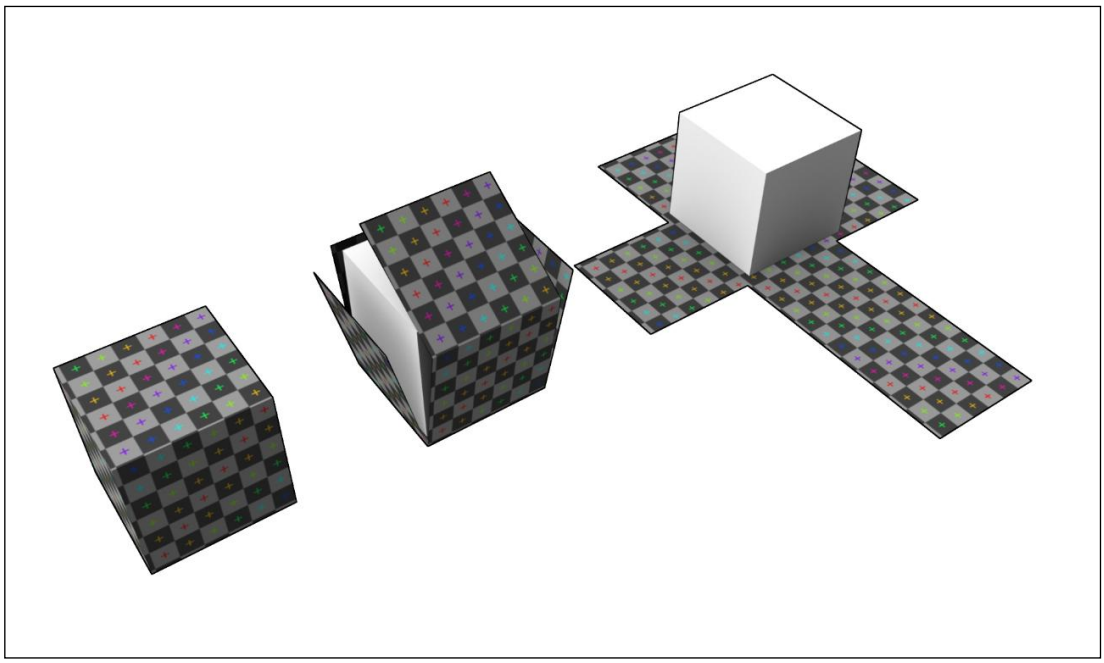

*FIGURE 10. UV mapped cube*

**Mesh**

Mesh is a collection of vertices, edges and faces that defines the geometric shape of an object (figure 11). The faces of the mesh usually consist of triangles and quadrilaterals.

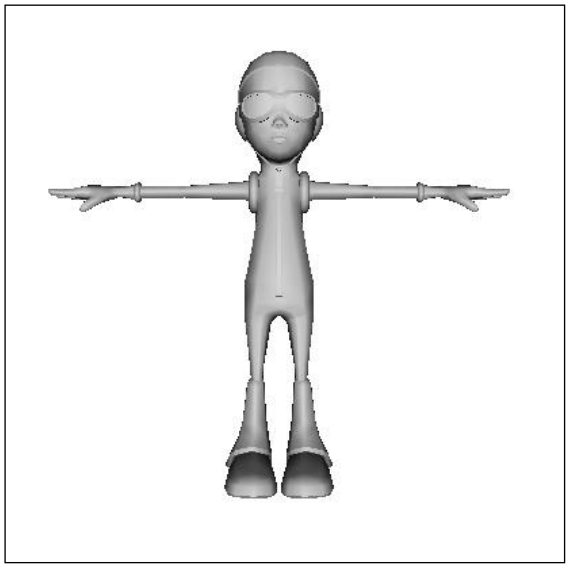

*FIGURE 11. Boy mesh in basic pose*

**Animation**

Animation in this context means a vectorized information that contains animation information, such as walking and running. In order to make the animation functional, mesh has to have a set of bones connected to each other and a skeletal movement for each animation set, which then change their transformation over a certain time period (figure 12).

*FIGURE 12. Skinned boy mesh with textures*

## 4.2 Google 3D Warehouse

Google 3D Warehouse is a service which allows users to upload, download and share 3D models. Today the 3D Warehouse offers dozens of models for free and even for commercial use. (17.)

When a model is uploaded, the user has to accept a license which transfers all rights to Google related to the model. One question Google has had in mind when creating the Warehouse is that users can easily fill the Google Earth service with content from the Warehouse. (17.)

For modeling there are many different kinds of software available, but most of them are not free and the learning curve is big due to the complex nature of the software. To make the modeling easier, Google has provided the modeling tool called Google Sketchup for giving users an easier access to the world of 3D. (17.)

The models which are uploaded to 3D Warehouse can be added to Google Earth by setting the option Google Earth Ready, which verifies that the building is real, current and correctly located. After some time it continues to a model review through several Google employees who then approve it if it meets the set criteria. (17.)

# 5 TOOLS AND LIBRARIES

In chapter 5 the tools and libraries that were used during the work are explained from a more technical aspect. Some of the technologies were already familiar and thus giving good starting point for the thesis.

## 5.1 Open Asset Import Library

Open Asset Import Library is a library for importing various well-known 3D model formats in a uniform manner, later referenced as AssImp. AssImp is an open source and has bindings for various programming languages. AssImp can also be used for exporting data, but it currently supports only Collada and 3ds Max formats for this purpose. In the thesis there is no need for an exporter, as all the imported data are converted to the Ogre mesh format internally and then transferred to the Tundra rendering core. (18.)

After the 3D format is imported using AssImp, all data is stored uniformly in the scene structure of AssImp which then provides an easy handling for the imported 3D data. After the importer has processed the 3D data, the importer parses the animations, cameras, lights, materials, meshes and textures into the corresponding data objects.

AssImp is written in C++ and it is licensed under the BSD license. There is also C API as well as bindings to various other languages, including Python and D. Assimp aims at providing a full asset conversion pipeline for the use of game engines and real time rendering systems of any kind. (18.)

Though the purpose of the thesis is to create a Collada importer, AssImp allows all other file formats that it supports to be available in Tundra, too. Some of the AssImp's supported 3d extensions are dae, blend, 3ds, obj, ms3d, lwo, cob and ase. Dae is the extension for Collada and stands for Digital Asset Exhange.

Collada was chosen as the 3D format for this work because it is an open standard and in the future it will probably be adapted by numerous other developers. Google's 3D Warehouse currently offers models in the following file formats: Google Sketchup, Google Earth and Collada. 3D Warehouse has also influenced the selecting of Collada as the 3D format.

Collada files are in XML format. XML defines the data structure of the Collada format, meaning that there are data for all the elements a scene could contain, such as animations, lightning and meshes (9). There is no reason to go deeply into the formation of XML, as all the data is parsed with the Open Asset Import Library hence doing all the file data handling.

As the thesis has progressed, numerous people have complained about the Collada standard being overly clever. It allows nearly everything, hence making it difficult for example for the Open Asset Import library to keep up.

## 5.2 OpenGL, environment for 2D/3D applications

OpenGL is an apparent selection for rendering 3D graphics in Ubuntu Linux, because it is the only compatible graphics pipeline offered for Linux. The OpenGL implementation as such is an open source and is supported on several platforms. However, usually a GPU vendor specific driver software implements the lower part of the acceleration.

**GLUT**

GLUT is a simple window system independent toolkit for writing OpenGL programs. It implements a windowing application programming interface for OpenGL. GLUT provides a portable API, and therefore it works across all PCs taking no account of the platform. (3.)

## 5.3 Ogre, 3D graphics engine

Ogre is an open source 3D graphics engine. Ogre is very widely used and has a very active community behind it. Ogre being a 3D graphics engine, it provides tools for rendering and handling 3D data. (19.)

Ogre wraps the most of the lower level OpenGL code in itself and as a result provides an easier high level graphics API for the developer, therefore hiding unnecessary OpenGL specific details. (19.)

## 5.4 OgreAssImp

OgreAssImp is a wrapper made for converting data processed through Open Asset Import Library to Ogre data. The source codes can be freely downloaded from the Google Project page of the OgreAssImp. (20.)

OgreAssImp can be used for collecting data from all formats the Open Asset Import Library supports. Data then can be exported to Ogre mesh, Ogre skeleton and Ogre material.

Although OgreAssImp included all the basic functionality, it took time to figure out how it worked since it needed some modifications. There were some big and small modifications that had to be done in order for OgreAssImp to work with Tundra. Modifications done to OgreAssImp are presented in chapter 6.

## 5.5 FreeImage, image conversion library

FreeImage is a library for handling image conversions and the library is aimed for developer use. FreeImage is an open source library with support for various image formats. (21.)

In this research project, Simple OpenGL Viewer, it was needed to convert images from the jpg and png formats to the RGB format. RGB means in this

context a 32-bit image where each of the four bytes represent channels red, green, blue and alpha. OpenGL does not recognize any compressed formats such as jpg or png directly, and that is why conversion must be done beforehand.

## 5.6 Git, version control system

Git is a completely free distributed revision control system. It has initially been designed and developed by Linus Torvalds for the Linux kernel development and it is currently being actively developed. The benefits of using Git are speed and independency of a network access or a central server. (22.)

GitHub is a web-based hosting service which hosts many kinds of software development projects, utilizing the GIT technology (figure 13).

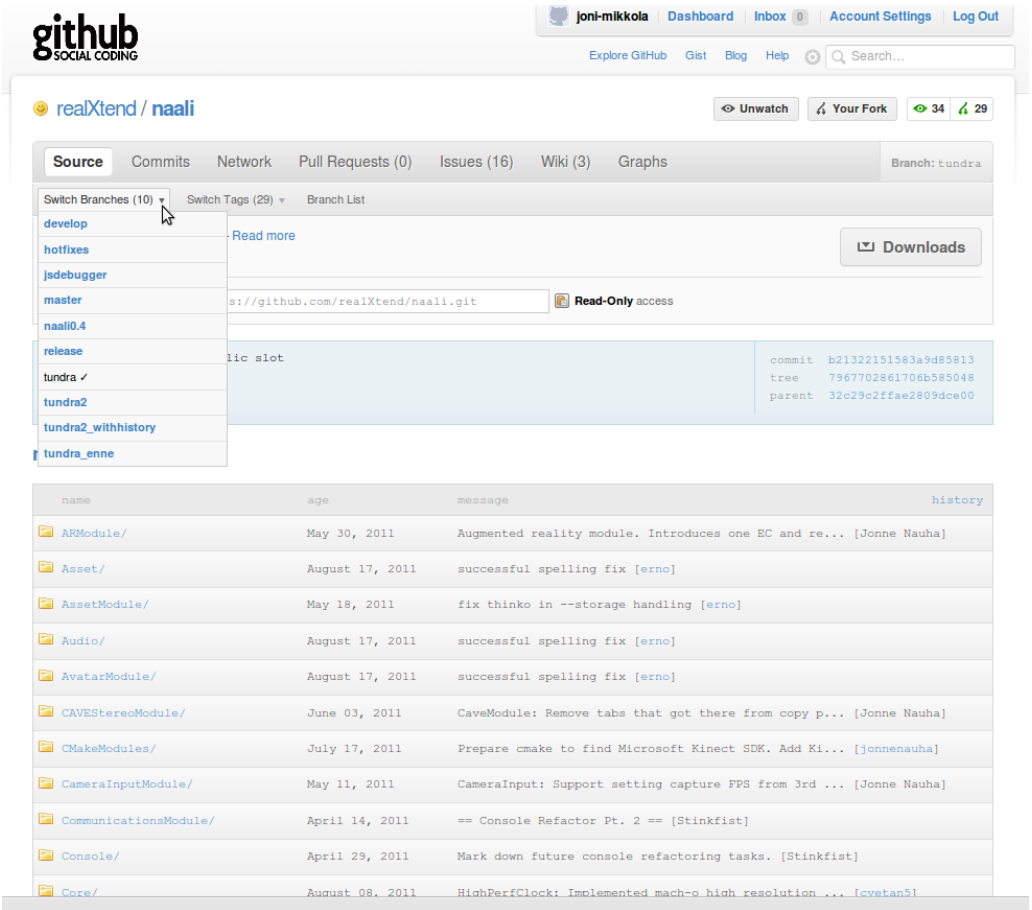


*FIGURE 13. Github showing realXtend project page*

**Merging to Tundra branch**

The development of the realXtend platform takes place under different branches in GitHub where each of them serves a different purpose. For example, the tundra2 branch stands for the upcoming Tundra 2 where development goes on rapidly.

In order to get feedback during the work, the code of Collada importer had to be uploaded somewhere where the superiors and members of the REX community could find and evaluate it.

It was not safe to start the coding straight into the Tundra branch since there was a big possibility for accidentally uploading a broken code, which then would have caused harm for other developers.

For avoiding such cases, GitHub offers fork functionality which allows contributing to someone else's project but within your own sandbox. The fork creates a clone of the selected project, and the developing work can be started in the forked project. When coding in the forked repository, the code is completely isolated from the original branch, and thus it does not disturb the development of others in any way.

What this isolation causes is that the fork eventually starts to lag behind the original branch as the original branch is still under development, getting new commits daily. Therefore it is advisable to merge the fork from time to time with the original branch to avoid consistence problems.

## 5.7 QT Framework

QT is a framework which aims for providing a generic platform for various operating systems (figure 14). QT was originally developed by a Norwegian company called Trolltech and currently it is produced by Nokia's Qt Development Frameworks division. (23.)

RealXtend is built on the QT framework. QT offers an easier portability for Windows, Linux, MeeGo and several other operating systems, that is why it is a fairly obvious choice for this kind of a project.

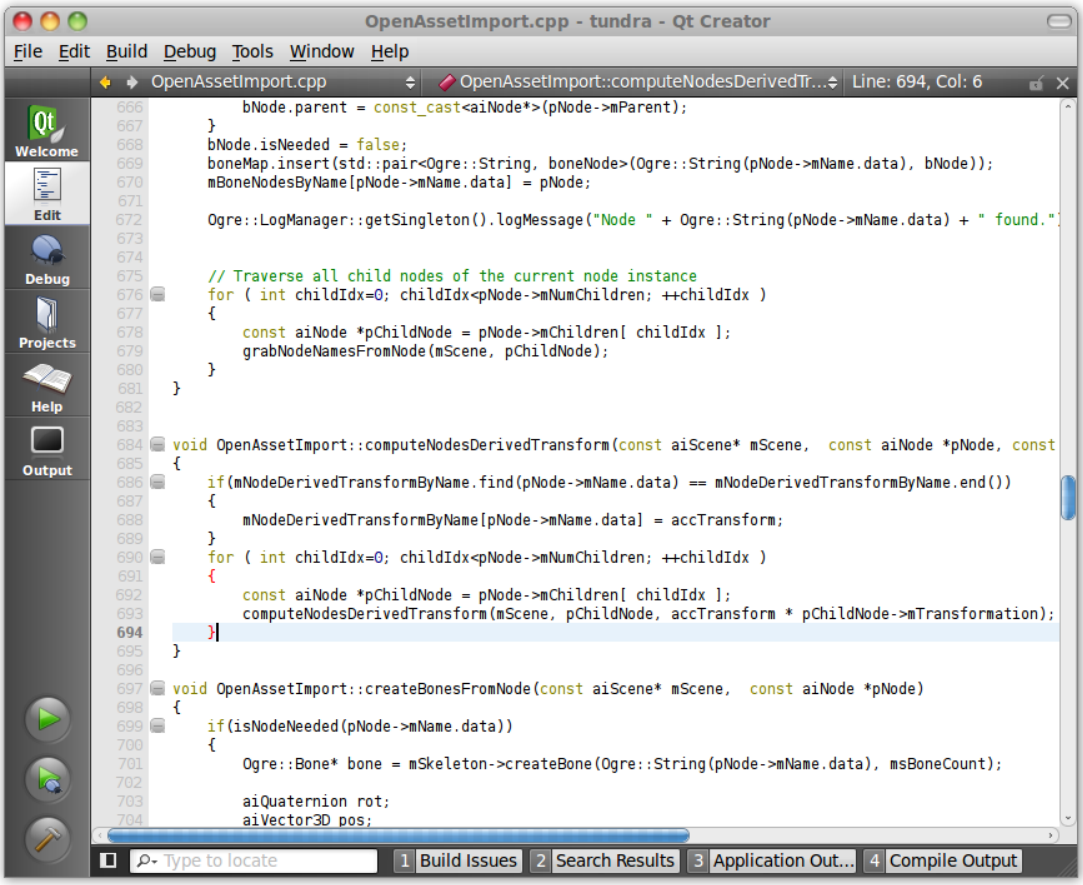


*FIGURE 14. Developing with QT Creator*

# 6 RESEARCH

The initial starting point for this project was to familiarize with the OpenGL and Open Asset Import Library to understand the basics of reading the Collada file and how to render it on the screen with textures and animations.

It was decided that two different projects are done for each research case. The first one was for rendering the Collada model downloaded from Google 3D Warehouse and the second one was for rendering the Collada model which contained animation data.

## 6.1 Creating Simple OpenGL Viewer

The viewer creation started by installing all required tools and libraries, QT Creator, GLUT and OpenGL libraries which were available in Ubuntu's package repository. Open Asset Import had to be downloaded from its homepage and then built and installed.

Open Asset Import contained a very suitable sample code for starting the experimenting with the library. This was very helpful for a beginner in the 3D graphics. The sample code included a window creation and a mesh rendering, and therefore texturing, animating and mouse rotations were the only primary components that were missing.

### 6.1.1 Importing file

The basic Collada file importing was very straightforward giving the file path for the first and the import flags for next parameter. AssImp has about 40 different import flags, when even all the additional properties are counted that can be set. Therefore, some experimenting had to be done for finding the balance between the importing settings. After a successful import had been done, the full file structure was provided by the aiScene class.

### 6.1.2 Textures and materials

Some research had to be done to figure out the way the textures are wrapped onto a face of a polygon. There was much discussion on the internet about this subject, thus helping in the search for solution. See sample code for texturing and material loading in appendix 1. Below there is a sample image of a Simple OpenGL Viewer rendering a model downloaded from Google's 3D Warehouse (figure 15).

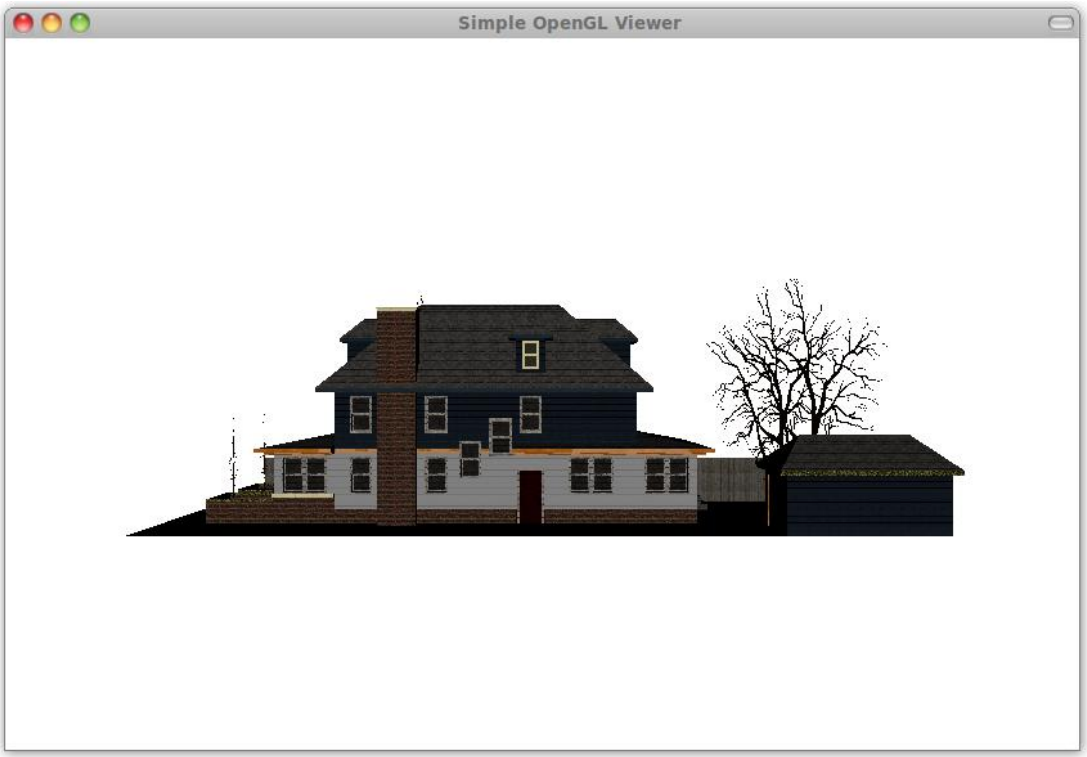


*FIGURE 15. "House" Collada model rendered with Simple OpenGL Viewer*

### 6.1.3 Skeletal animation

Getting the animation data works somewhat in the same manner as getting the texture data. After a successful file load, a pointer to aiScene is received consisting of the animation data.

An animation in this context is a set of keyframe sequences where each sequence describes the orientation of a single node in the hierarchy over a limited time span. Animations of this kind

> are usually used to animate the skeleton of a skinned mesh, but there are other uses as well. (1.)
>
> The aiAnimation consists of a series of aiNodeAnim's. Each bone animation affects a single node in the node hierarchy only, the name specifying which node is affected. For this node the structure stores three separate key sequences: a vector key sequence for the position, a quaternion key sequence for the rotation and another vector key sequence for the scaling. (1.)

Each animation has animation channels determining which node the animation channel is affecting. Thus, in this case there is one animation and all the nodes (hands, feet, torso, head etc) have their own channels. In short the operation goes as follows: load mesh, select animation, loop through channels in the animation and update position, rotation and scaling factors for each node.

For each update a trilinear interpolation is also applied. A short example use case of the linear interpolation could be a case where the first frame of animation has a human head mesh with the mouth fully open and in the next frame the mouth is closed. What trilinear interpolation does is that it calculates the data points between the mouth open and the mouth closed, hence forming a smooth animation. The usage of interpolations also reduces the size of the animation in bytes due to the decreased amount of frames. (2, p. 522.)

Though a transformation for the bones is updated, the actual mesh has not changed at all. There are still bone variations to apply to the mesh by calculating the bone weight for each part of the mesh.

# 7 IMPLEMENTING OPEN ASSET IMPORT

The most difficult part of the implementing was the fact that Tundra consists of over 100 000 lines of code and there was no good documentation for developers. Thus, it took many hours to get an idea of how different kinds of assets were transferred between the classes. Therefore, when the initial code was written in Tundra, it was moved many times before it found its final location.

## 7.1 Modifying OgreAssImp

When OgreAssImp eventually started to work together with Tundra, the experimenting was started to see how the Collada models are rendered with Tundra.

About a half of the tested models from Google 3D Warehouse seemed to be working and the other half had problems where vertexes and normals were disorganized, resulting in an incorrect rendering of the models. Some of the mesh's polygons were invisible, for example a part of the roof of the house was missing. In addition, there was a problem where a model would show up as too large making it difficult to start building a scene. Furthermore, there were skeletal animations which did not play correctly. Eventually, after long-lasting studies and experiments, these problems were broken down into four main questions.

### 7.1.1 Linear scaling

RealXtend does not take a stand in which sizes models are brought into the scene. Some models, i.e. a bridge was so big that on some occasions it could not be fully seen and sometimes only a slight fading from it in the horizon. That was the reason why it was essential for such models to be scaled down in size.

From the end-user's perspective, it is much easier to start creating a scene if all the meshes are scaled down to a size where they can be seen as a whole. It was discussed that it might distort the proportion the original modeler desired. Even though that was true, there will also be a later implemented feature which allows the referencing in the way that the original proportions will be preserved.

### 7.1.2 Lines and points

Some of the 3D formats contain unnecessary geometric primitive information, lines and points, which have to be removed in some formats, such as Collada. The lines and points were not really well documented, and therefore it is unclear in which formats this removal has to be done.

When the line and point primitives were included in the conversion, it caused the mesh to be rendered strangely as can be seen in figure 16.

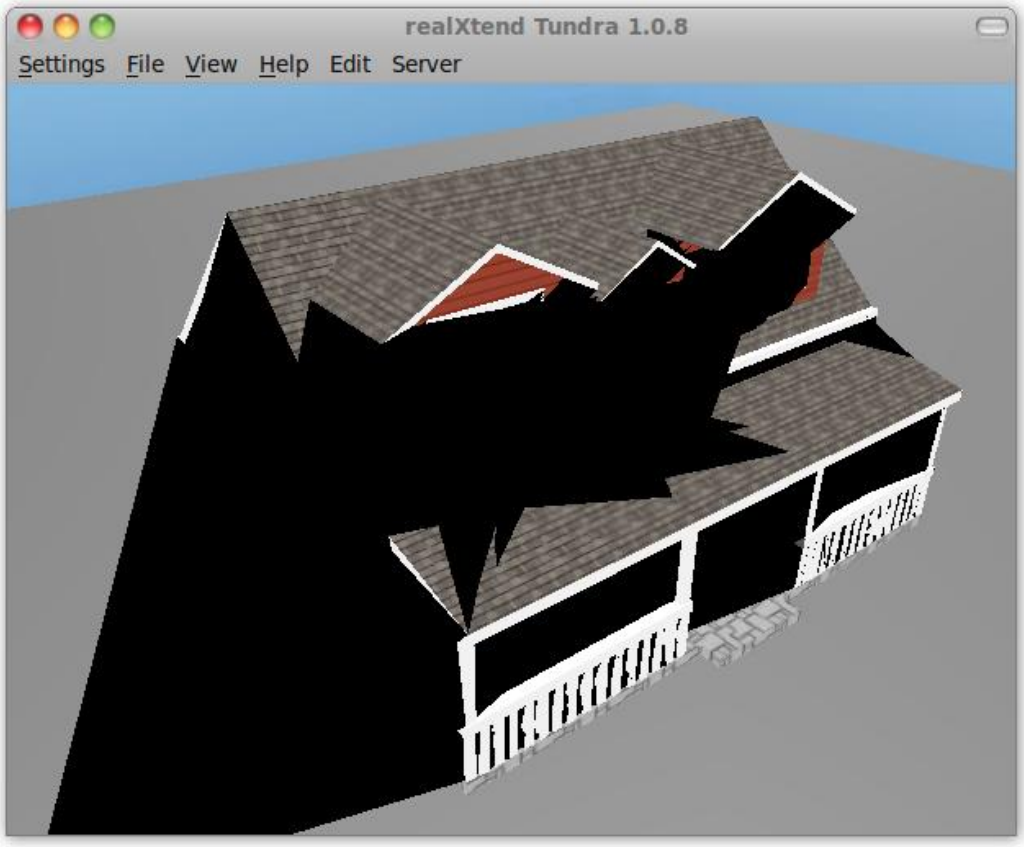


*FIGURE 16. Line and point primitives enabled in conversion*

Once the lines and points were removed in the conversion, the results were very good. (Figure 17.)

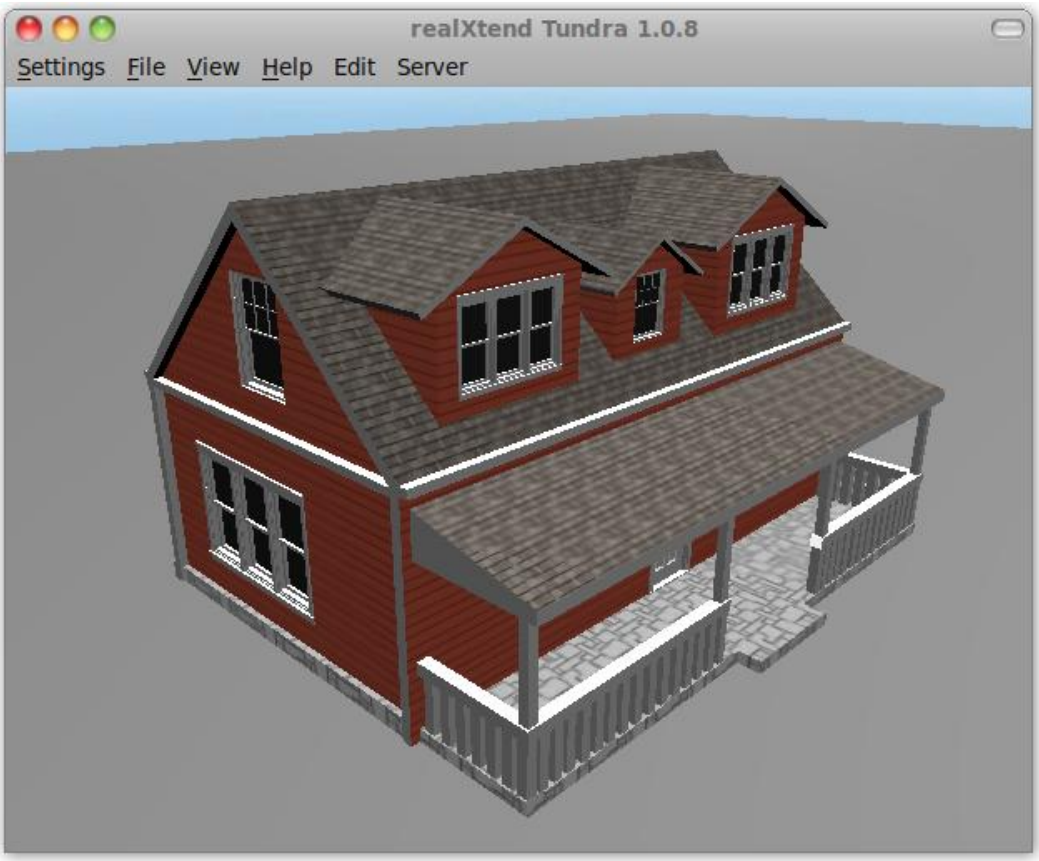


*FIGURE 17. Line and point primitives disabled in conversion*

### 7.1.3 Mesh memory allocation

The Collada model consists of a main mesh, which has sub meshes, which then can contain sub meshes and so on. It is a hierarchical structure of a root mesh containing child meshes. When the Collada model is processed through OpenAssetImport, there is a hierarchical structure which has to be converted into a single Ogre mesh. OgreAssImp converts this structure into the Ogre mesh, mesh by mesh. If some of these meshes contain more than 65335 indices, it causes a buffer overrun and some portion of the mesh is then dismissed from the conversion process (See figure 18).

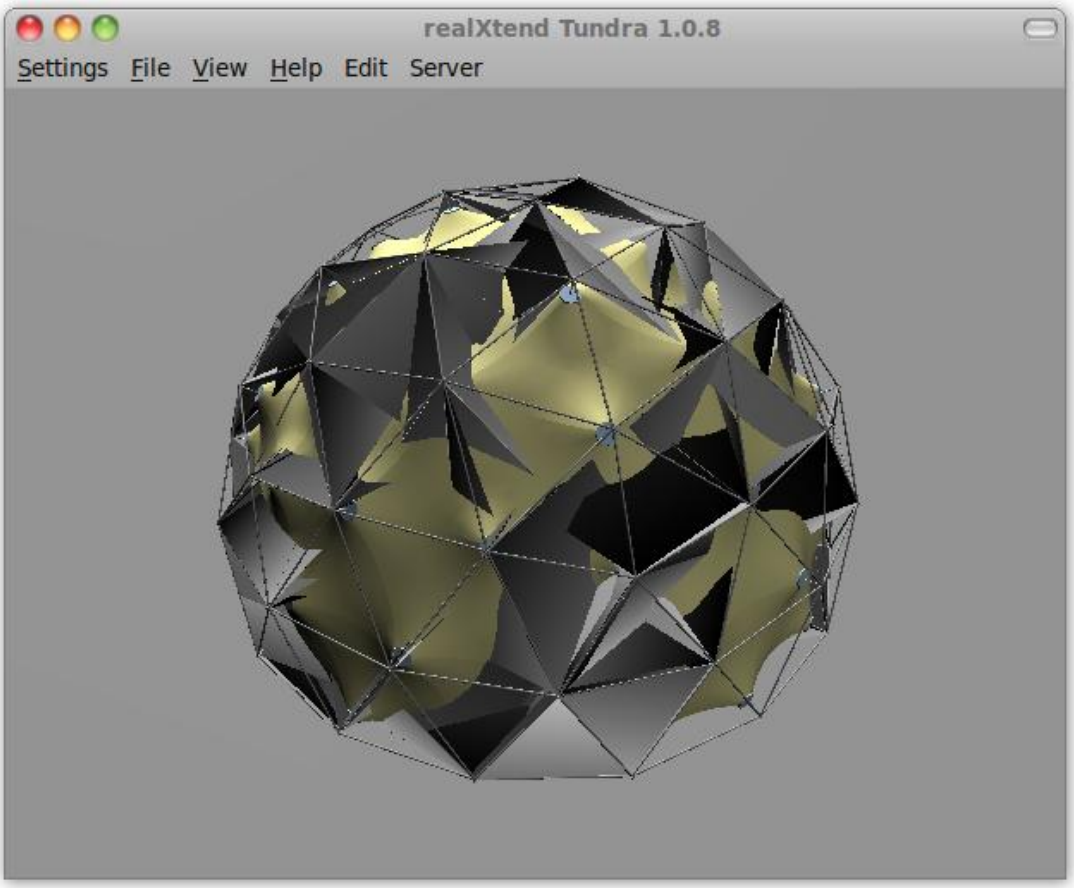


*FIGURE 18. Incorrect memory handling*

The reason for such a behavior is eventually pinpointed to the memory handling. When the mesh is converted by using a graphics card, a certain memory buffer for indices is needed to be allocated. The indice memory buffer was mistakenly allocated for only 16-bit. It was easy to change it to the 32-bit allocation, but for some reason Tundra still seemed to crash in the conversion bit.

Ultimately this was fixed by limiting each mesh size with AssImp (figure 19). If there were too many indices, the mesh was split into smaller parts.

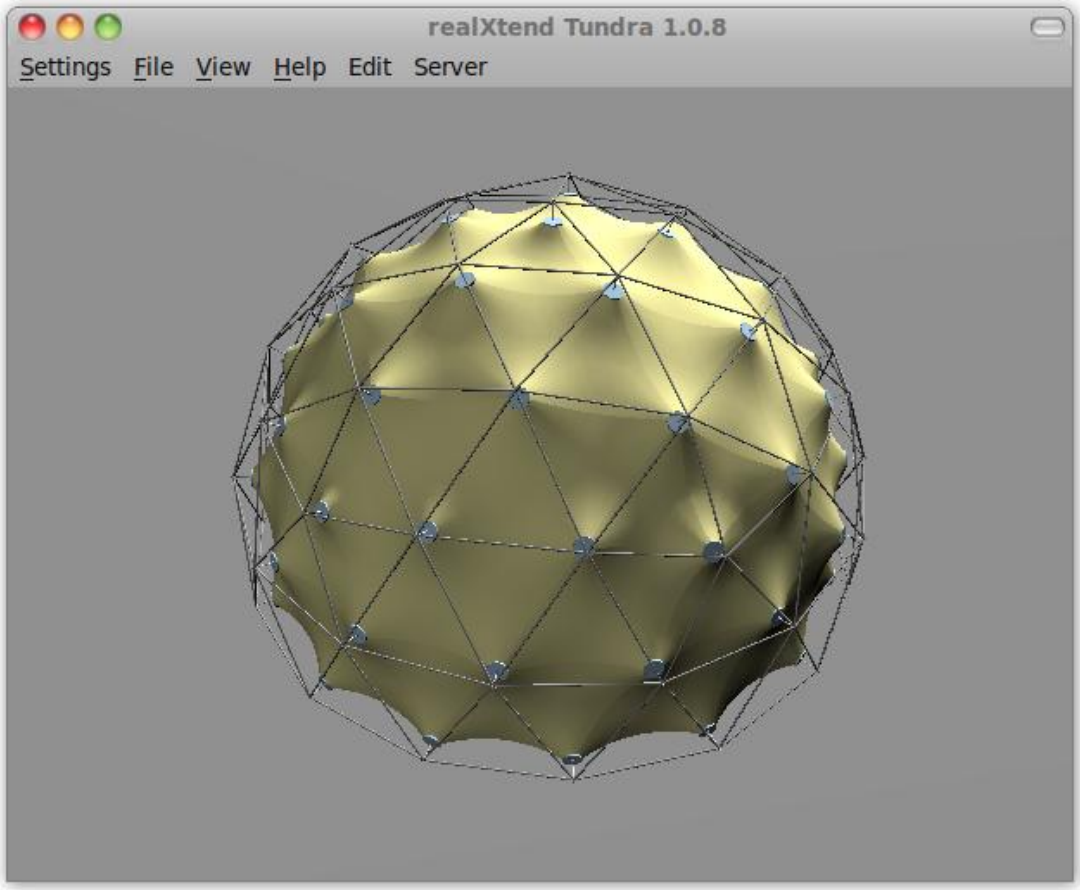


*FIGURE 19. Correct memory handling*

### 7.1.4 Back-face culling

The back-face culling defines whether a polygon is visible or not. The back-face culling in this context means a problem where some of the faces of a polygon are not rendered towards the camera and as a result those parts of the model are missing.

The direction of a face of a mesh is measured by calculating the cross product of the three edges of a triangle. Back-face culling is a method for dropping out the unneeded faces from being rendered, hence speeding up the rendering time. (24.)

In the mesh creation phase of OgreAssImp, the following material information for each mesh was read: diffusion, specularity, emission, ambience and shininess. However, it lacked the check for two-sided materials (figure 20).

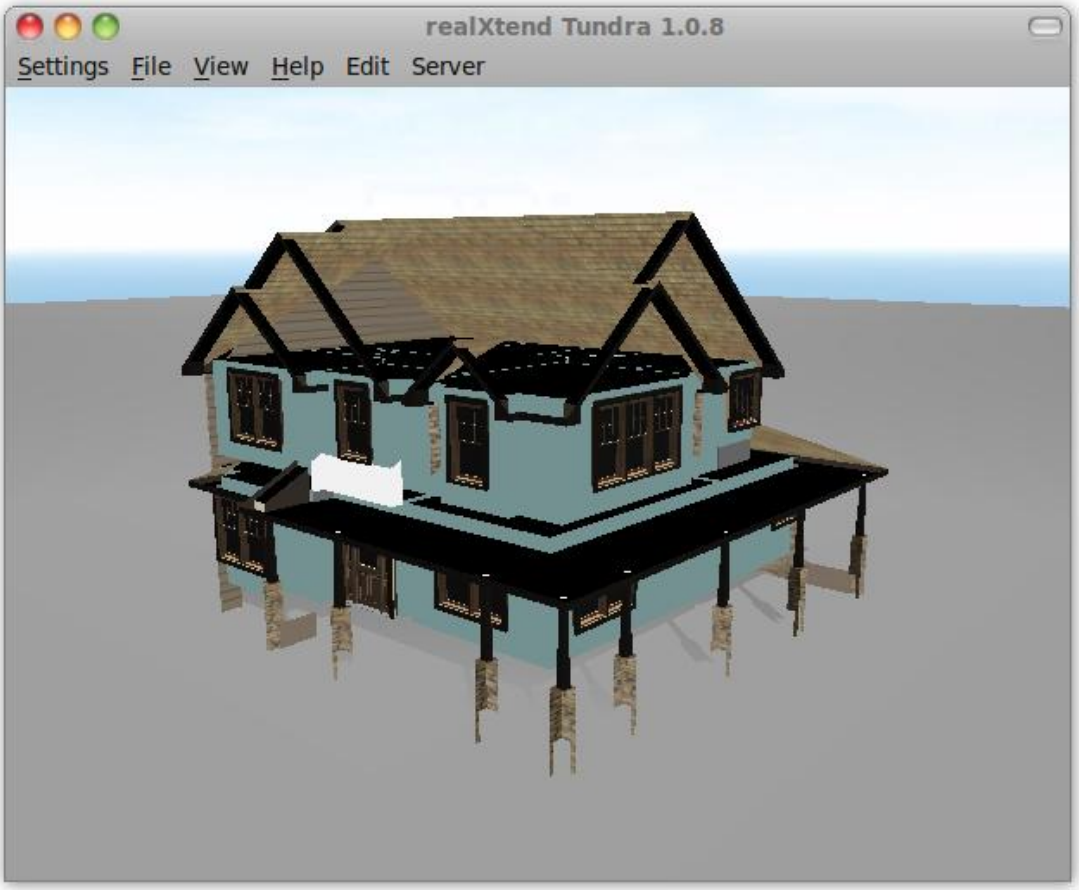


*FIGURE 20. Face culling set wrong*

Furthermore, when some Ogre documentation was consulted, the correct parameters were found for fixing the problem. This fixed a great number of culling problems, although some of the polygons were still unseen on some Google 3D Warehouse models (figure 21). It was eventually decided that these problems occurred as a result of a bad modeling and of the modeler's inability to set the facets correctly.

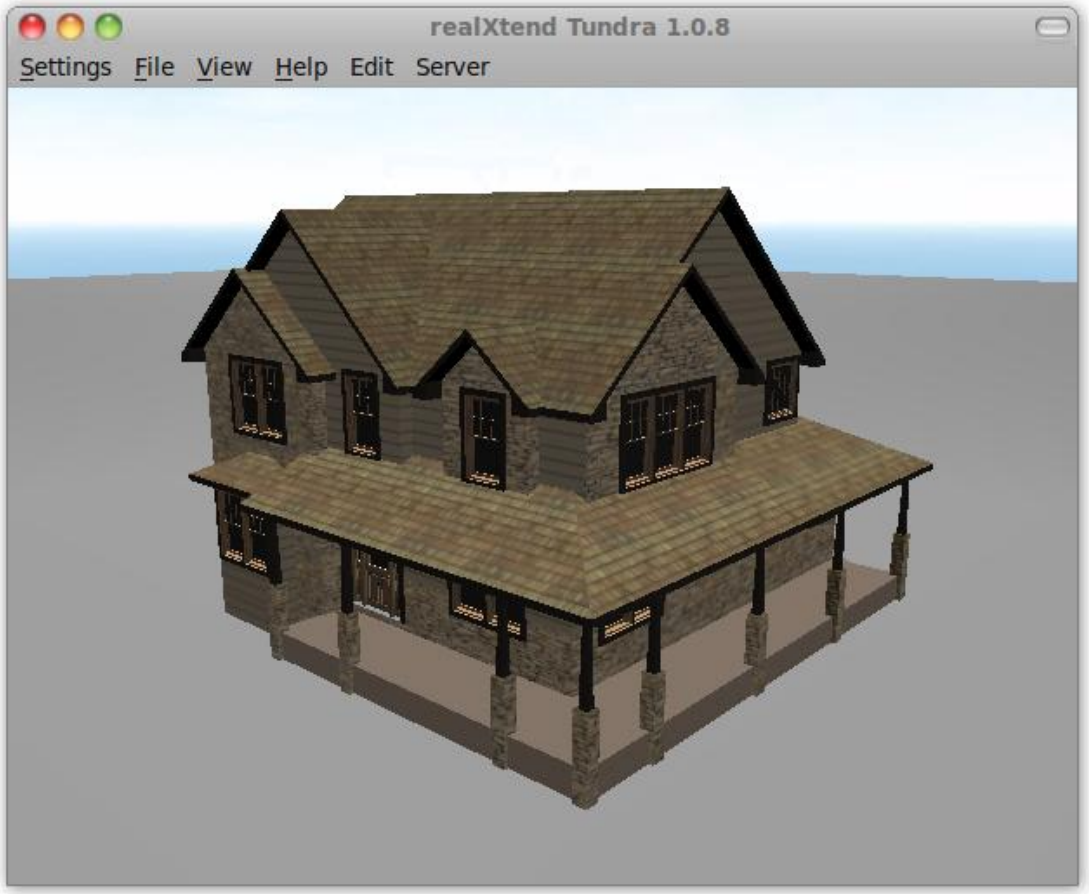


*FIGURE 21. Faces set correctly*

## 7.2 Mesh conversion

Initially, a few good pointers were given by the superiors, considering which code components of the realXtend should be modified in order to get the mesh conversion done. The Ogre mesh is currently only a 3D model format that realXtend supports, due to the tight integration of Tundra core with the Ogre rendering system. It was discussed that the conversion should be done in such a way that realXtend's renderer stayed agnostic about the 3D format.

When the asset was being transferred in realXtend's asset mechanisms and identified as Collada, Collada was instantly converted to the Ogre mesh (appendix 2) and the mesh was then put to an asset buffer, where it continued to the Ogre Rendering Module (appendix 4).

## 7.3 Skeletal animation

The support for skeletal animation was the last phase done for this thesis. Animations are needed in virtual worlds in many use cases, but the most important use case is the avatar. Whether the avatar is walking, running or sitting, the corresponding animation should be rendered on the screen. The initial base code for skeleton creation was included with the code of OgreAssImp, and thus there was no need to start from zero.

Since there was not enough time to implement the skeletons totally, the support for avatars was left out, although an initial support for the skeletons was done. As the animated Collada file was loaded into realXtend, the skeleton file for that model was created and then loaded and played by the user.

The code OgreAssImp provided for the skeleton creation was initially good. One half of the correctly formed Collada animations seemed to work well but the other half did not. In this case the correctly formed means Collada files that Blender can open. The Internet seems to be full of non-working Collada files which are not formed in the way that the Collada standard says. In some cases, one could try to get the skeletons work on a file that in the sense of Collada standard is messed up, although it is only waste of time.

The mesh seemed to get incorrect transformations as can be seen in figure 22. As the mathematics that lie in the field of 3D were somewhat a mystery, it was not easy to start debugging the cause for the problem.

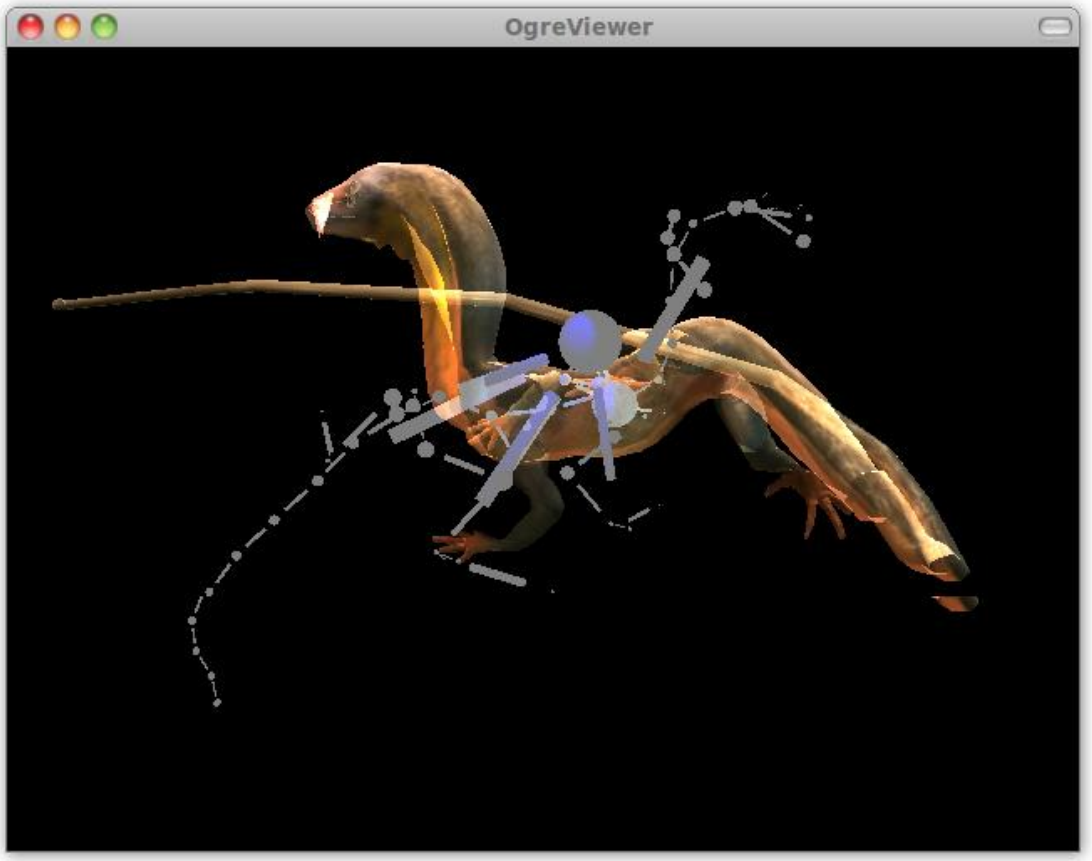


*FIGURE 22. Mesh transformation is wrong*

Finally, after hard work, an Ogre Viewer was decided to be created to render the skeleton and mesh data, which then could be used for debugging purposes, as the figure 22 shown above shows. Before the Ogre Viewer was created, probably 40 hours of work was spent on coding with the trial and error method.

The Ogre Viewer finally revealed that the mesh was not aligned correctly with the skeleton. It was actually assumed before but now the problem was verified, hence reducing the number of possible causes in the code. With the help of Tero Pihlajakoski, a 3D expert, the problem was pinpointed to a wrong transformation of the mesh. Ultimately the problem was solved by commenting out one useless matrix multiplication and as a result the mesh was correctly aligned with the skeleton. (Figure 23.)

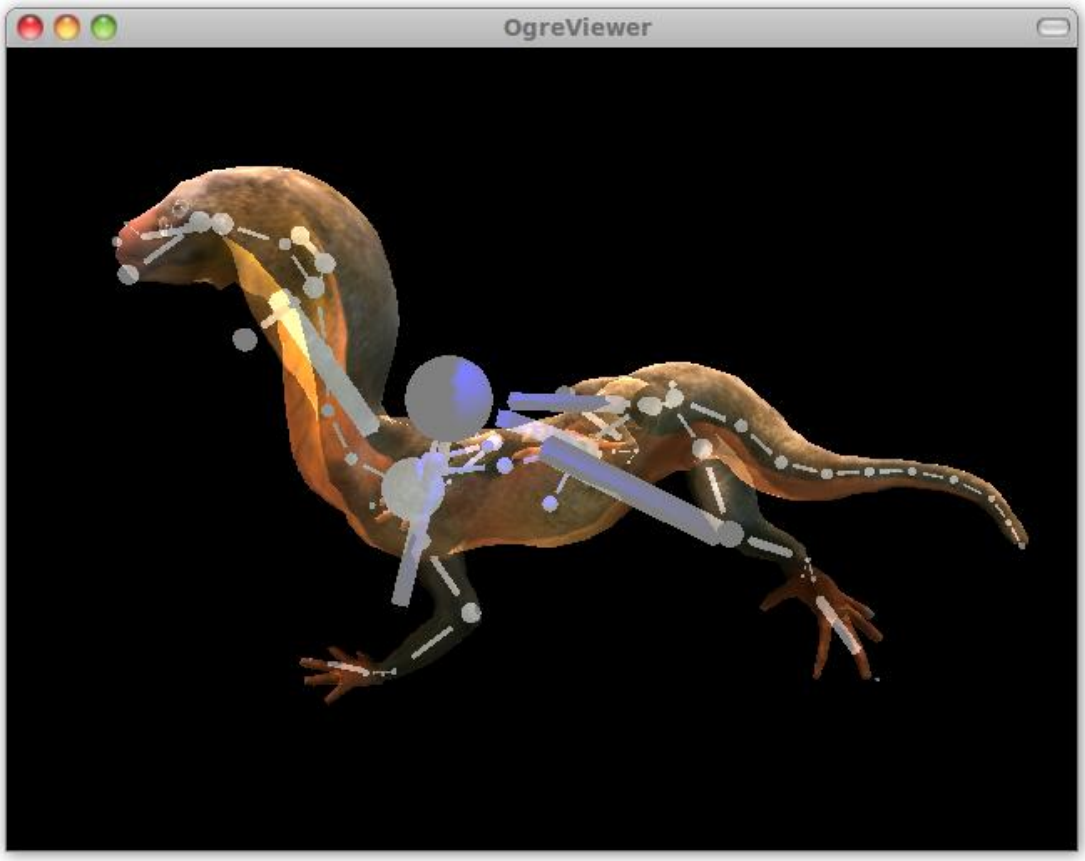


*FIGURE 23. Correctly aligned mesh*

## 7.4 Material references

In Tundra, each Ogre mesh is a component called EC_Mesh. EC_Mesh holds such information as reference to mesh itself, its materials and skeleton file (figure 24). Consequently material references are files located in the hard-drive, which are added as separate references.

Because the Collada files include material information stored in them, there is no need for adding them separately. All material data can be fetched from the Collada model while the conversion is happening in the Open Asset Import module.

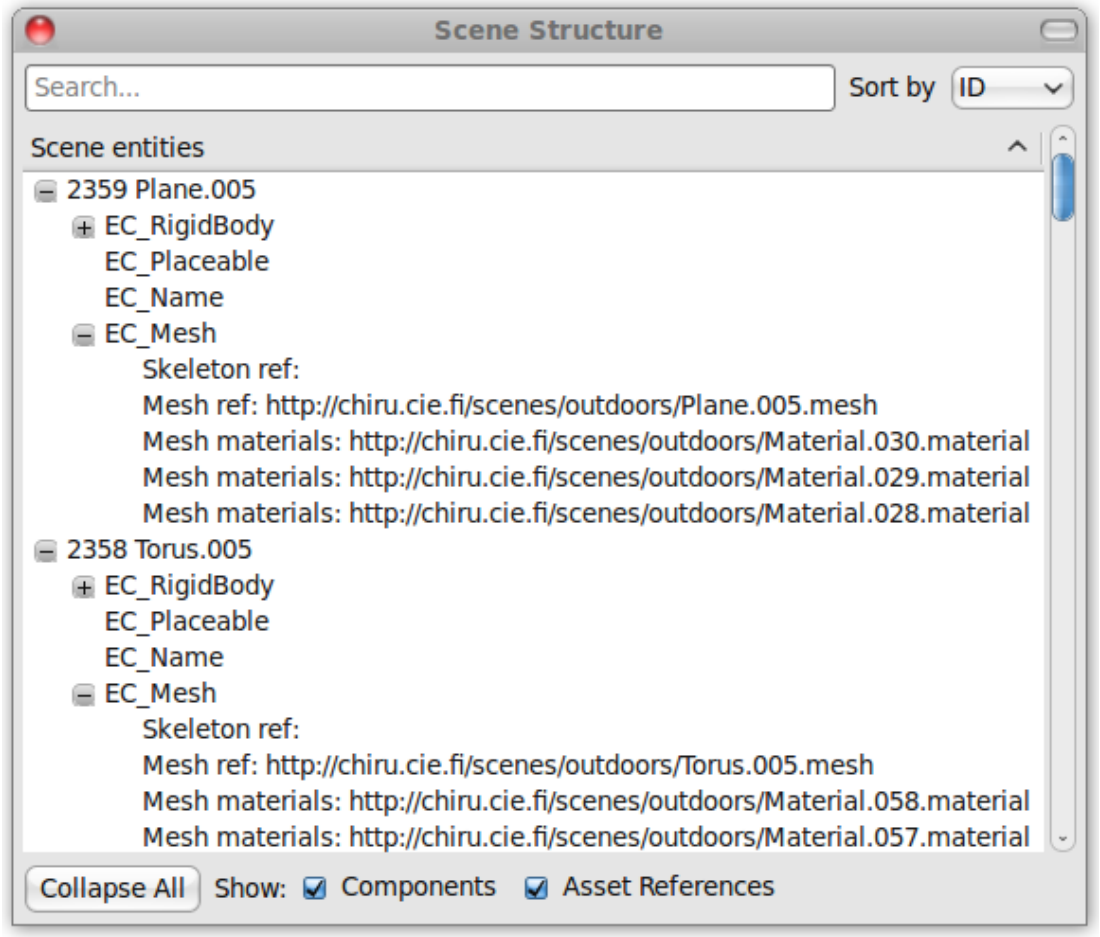


*FIGURE 24. References are files*

It was discussed that the Collada materials should be referenced as follows: local://house.dae#roof.material, where string after # points to sub-asset inside the requested Collada model (figure 25).

It was later discussed that Tundra needs a uniform way of referencing the sub-asset of a file. As it happened, there was not enough time to create a uniform support for the sub-assets within the scope of this thesis.

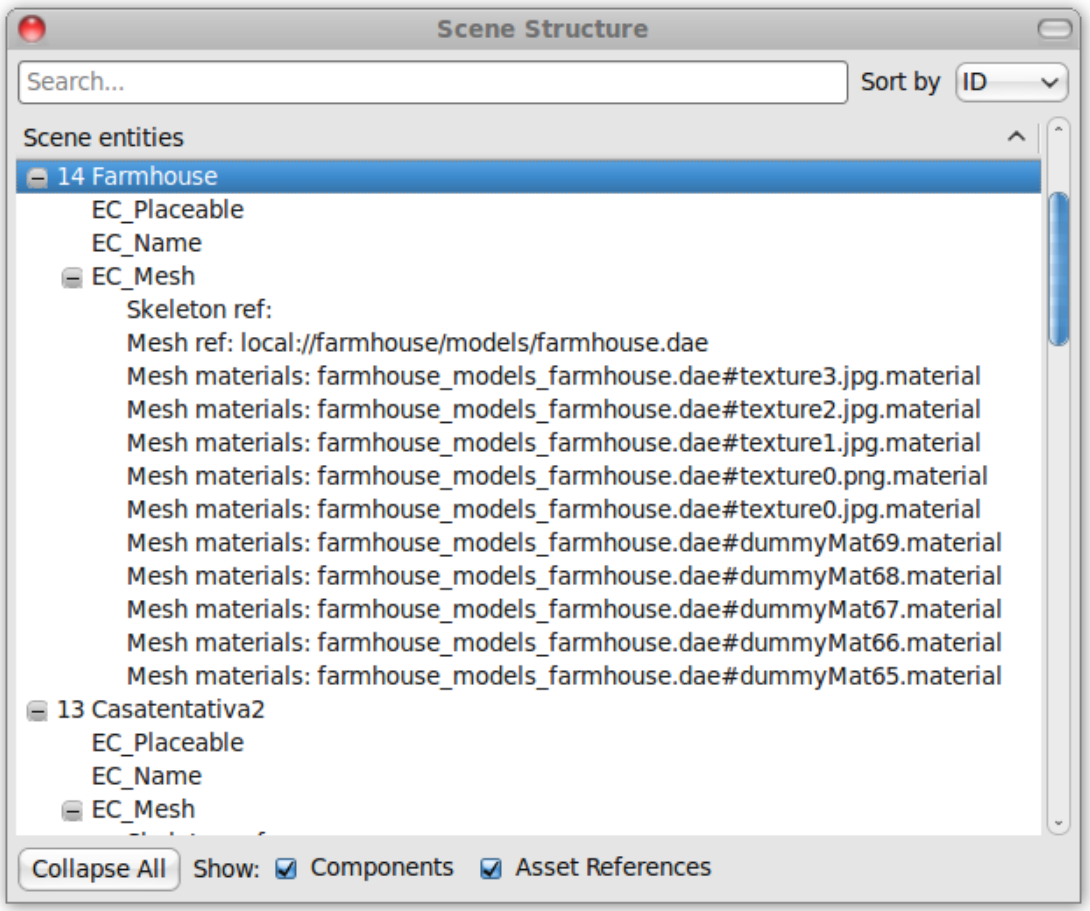


*FIGURE 25. Material references as sub-assets*

## 7.5 Compressed Blend textures

Though the thesis is about importing files in the Collada format, the Collada implementation also support various other formats, such as .blend, which is the format the Blender 3D modeling tool uses. Many of the models that the modelers have made for Tundra, are in .blend format. Blend models are useful for virtual worlds, especially for representing a scenery, as they can contain texture data within them thus being a handy format with all information stored in one file.

OgreAssImp did not initially have any support for compressed textures nor it had support for passing the textures using Ogre's ResourceManager. There were two options for loading the compressed textures.

The first option was to create a file from each texture the .blend file contained, and then let Tundra load them as it normally did, by a texture file reference in the material definition. The second option was to pass each texture using ResourceManager of Ogre, which was more time sparing, as then there was no need to write temporary files in a hard-drive. Instead, the data was passed via RAM buffers. The material file references to a texture that was located in Ogre's Resource Manager, not in a local file.

The second option was chosen, since it was the faster and cleaner solution, as then the whole process was internal. Therefore, there was no need to write anything temporary on a hard-drive.

## 7.6 Structure

The whole conversion system is explained shortly in this section. In figure 26 the system is divided into five steps and a result. Each step is described shortly.

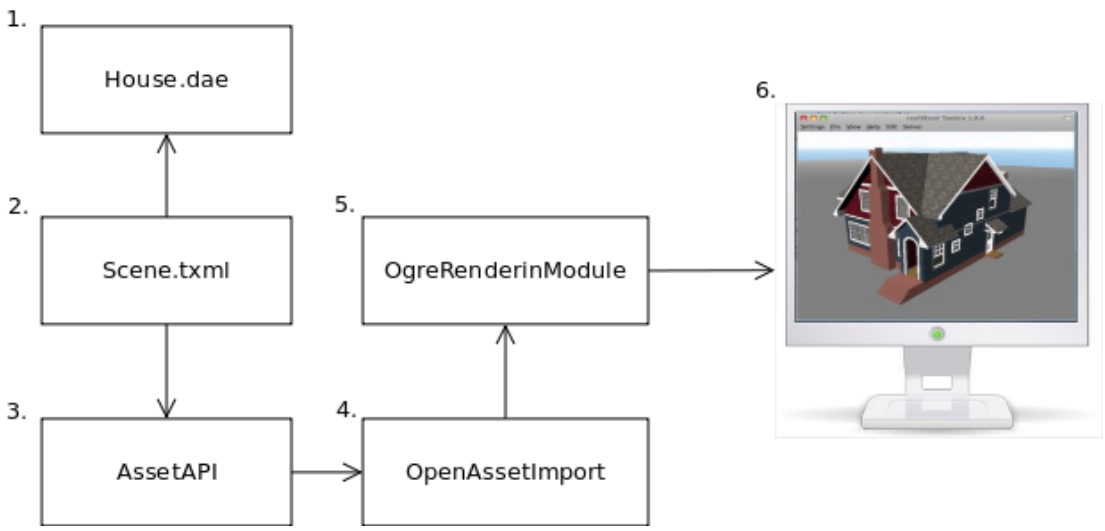


*FIGURE 26. Simple data flow diagram*

1. House.dae is the Collada file that is going to be added on the scene.

2. Scene.txml is the declarative file containing all the information related to the scene and the entity components it contains. A reference to House.dae file is located in this file.

3. AssetAPI handles all the assets that are run in Tundra. Because Scene.txml contains a reference to House.dae, the file is passed through AssetAPI as a result. AssetAPI identifies the file being in Collada format and therefore, passes it to the OpenAssetImport.

4. OpenAssetImport converts the Collada data to the Ogre format, including meshes, materials and skeletons.

5. The converted data is in the Ogre format, and therefore Ogre can render the data.

6. In figure 26, the House.dae model is being rendered with materials.

## 7.7 Results

It was planned that Tundra needs a dialog for browsing through the 3D Warehouse models (figure 27). This dialog was done by a co-worker. This does not really concern the topic of the thesis but is worth mentioning because it provides the final user experience.

When a user downloads the chosen Collada model from the Warehouse, it appears to the list on the right (figure 27). From the list the user can select a zip file and then click on “Add to scene”, which then adds the Collada file inside the zip as an asset. As the Collada file goes through Tundra's asset mechanism, at one point it is identified as being .dae. After that, the OgreAssImp does the conversion and passes the converted data forward to the renderer (appendix 4).

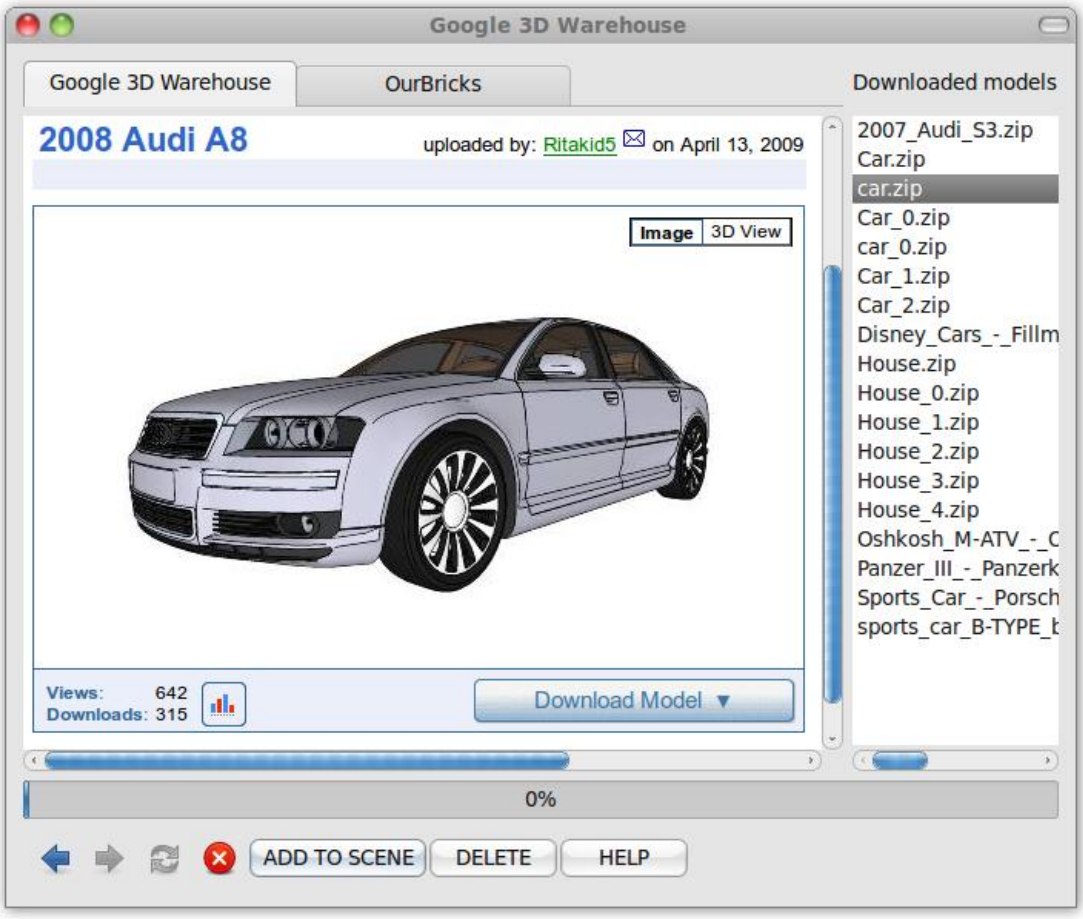


*FIGURE 27. 3D Warehouse dialog in Tundra*

A demoscene was included with the Collada importer to display how the importer performs. See figure 28, which shows the demoscene that is loaded in Tundra.

The materials and textures started to work as planned. The compressed blend files also started to work well. The skeletal animations seemed to start working for about a half of the models. The animations were tested in an avatar-like scenarios with Tundra's Animation Controller component.

However, the implementation could have been better, if there had been more time for implementing a more logical way of handling the sub-assets. Because the Asset Bundle mechanism was missing, the core components of Tundra had to be modified which was undesired. As a result, the real imple-

mentation between the Collada files and Tundra was not implemented in the final merge operation.

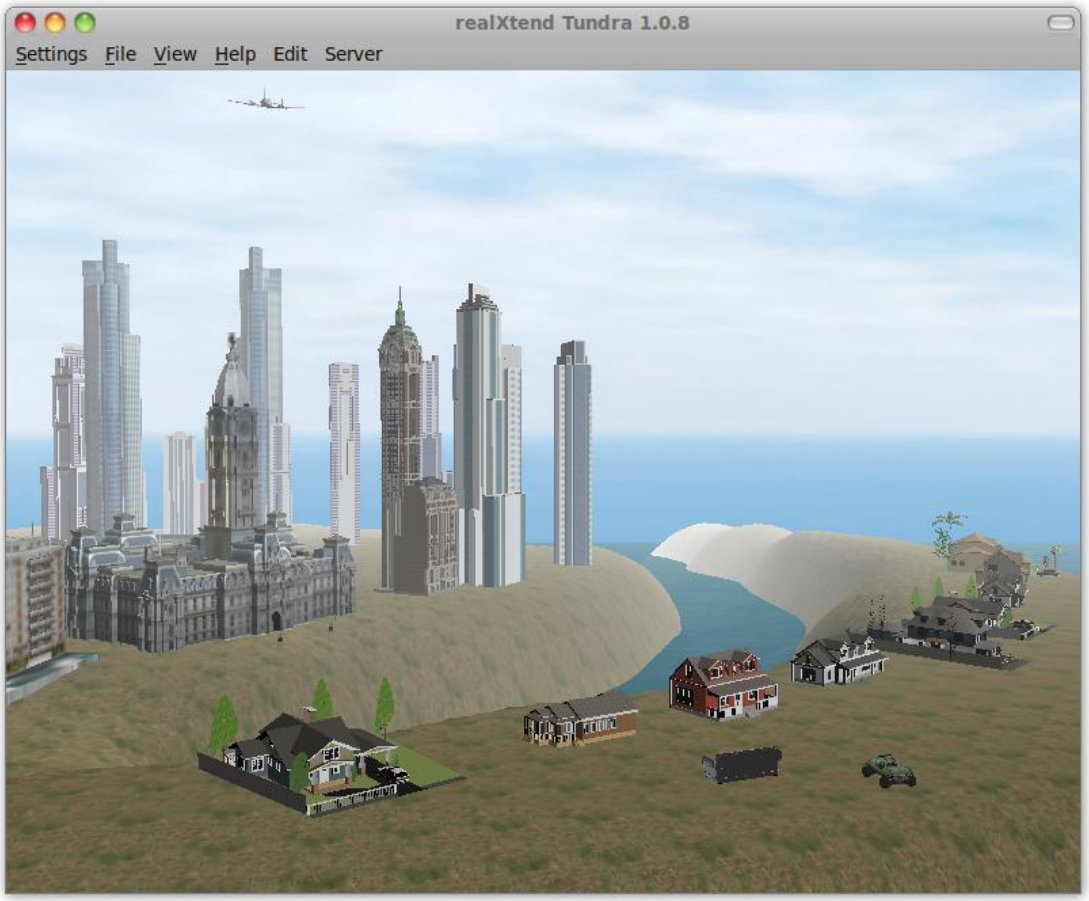


*FIGURE 28. Tundra scene filled with Collada files*

## 7.8 Merging Collada implementation

In this phase, all the work was done in the specific fork located in GitHub. As the Collada implementation finally did what was originally planned, it was time to merge the implementation to the original Tundra branch.

Initially, the intention was to merge the OpenAssetImport, the Google 3D Warehouse module and the implementation between them in the Tundra branch. Eventually the core touching code was dropped from the implementation, as it would have compromised the stability of the Tundra.

As a result, only the OpenAssetImport and Google 3D Warehouse modules were merged to the Tundra branch without integration. These modules were safe to merge as they were isolated from the Tundra core.

Although only the OpenAssetImport module and Google 3D Warehouse module were merged to Tundra, they give a good initial set for creating an implementation in a proper way later on, hopefully by the Rex community.

# 8 TESTING

The testing had to be done continuously, because in order to merge the Collada implementation to the Tundra branch, it was essential that the Collada implementation did not compromise the stability of Tundra.

Due to the fact that the implementation is done in the core elements of Tundra with a somewhat hack code, the core code is exposed to the implementation and as a result has to be thoroughly tested.

A developer should always consider easier ways for conducting the testing. Tundra is very large project and therefore the compiling takes time. Every time some functionality is changed in the code, the depending parts of the Tundra are built again. Sometimes even creating an application for testing purposes seems to be a good idea, if the problem offers such method for solution. It could eventually save much time by decreasing the compile time enormously, and in this case it did.

## 8.1 Google 3D Warehouse dialog

After the Collada Implementation was finished for the mesh and material conversion part and the linkage between the Warehouse dialog and Tundra was created, the testing started for the Warehouse dialog. The testing was mostly done by a developer, who designed and developed the dialog.

The testing method was simply testing every possible task a user can do with the dialog. Each time the dialog crashed or did something unusual, the developer was notified about the problem.

While the developer tested the dialog by downloading various models from http://www.ourbricks.com, http://sketchup.google.com/3dwarehouse/ and adding them to the scene, the testing was also done for the Collada implementation as it always did the conversion when “Add to scene” was clicked. Whenever a model rendered incorrectly, the developer notified of it. In case

of an incorrect model, it was tested in the way it is explained later in the thesis.

## 8.2 Scenes

Originally the Tundra scenes did not support any 3D formats other than the Ogre mesh. The test scene was set up, where models from various formats (3ds, dae, lwo, obj, etc.) were rendered on the screen, forming a nice scenery where each of the models varied from one another, thus creating a good testing environment.

Two scenes were created for testing, first for local and second by http references. The test scenes consisted of some houses, cars, sofa etc (figure 29). The test scene was tested locally and remotely having the same data located on a hard-drive and on the internet.


*FIGURE 29. Demo scene used for testing purposes*

## 8.3 Models

One problem with the models was that the internet seems to be full of broken models which do not follow the standards correctly. Thus, if a broken model is taken as a reference for testing, time is wasted when trying to get it rendered correctly as the original problem lies in the erroneous formation of the file.

Therefore, whenever a disfigured model was encountered, it was verified either as being broken or correct by using Blender. It was assumed that Blender followed the standards correctly, hence it was the obvious choice for such testing, being very actively developed and the only potential free 3D software for Ubuntu.

## 8.4 Animations

Animations were somewhat slow to test, because there were some of steps that had to be done in Tundra to get the animation played. In addition, the compiling after each modification took some time.

For speeding up the animation testing process, JavaScript script was created for mesh creation and animation playing (appendix 3). When this script was executed in the scene loading, the animation started playing and it could instantly be seen whether it was correct or not. At some point it was noted that the cause for the messed up animation could be the misaligned mesh or skeleton.

Tundra does not provide any good debugging solutions for skeletons, and for that reason it was suggested that the Ogre viewer should be programmed for showing if the skeleton or mesh has offset or some other problems, thus giving good advice where the problem could be located in the code. The viewer for such debugging purposes did not seem to be available on the internet. Eventually, it is planned that the Ogre viewer and the codes are uploaded on the GitHub or a similar project hosting site.

# 9 CONCLUSION

The goal of the thesis was to implement a Collada importer for Tundra. The initial objective was reached, as the importer can currently handle the Collada format the way it was planned. However, there are still many problems that need to be overcome in order to get the implementation done in a proper way.

Currently the core code of Tundra is exposed to the implementation, hence compromising the stability and being the reason why the core touching code was left out from the merge. One way of getting rid of it is to build an AssetBundle mechanism to the core.

Apart from the implementation, a well working Open Asset Import module is now provided for Tundra. The problems with Ogre mesh, Ogre skeleton and Ogre materials are now solved and they are fully functional and ready to use now. It is only the implementation with Tundra that can be made better.

Nevertheless, the OpenAssetImport module is now open and merged to the main branch of Tundra, giving it to the hands of developers, who hopefully start developing it further at some point.

# REFERENCES


1. Assimp: Data Structures. Available at: http://assimp.sourceforge.net/lib_html/data.html. Date of data acquisition 31 August 2011.

2. Heckbert, P. 1994, Graphics Gems IV. Boston: Apress.

3. GLUT – The OpenGL Utility Toolkit. Available at: http://www.opengl.org/resources/libraries/glut. Date of data acquisition 31 August 2011.

4. Thelin, J. 2007. Foundations of Qt development. New York: Apress.

5. Getting Started with Tundra. Available at: http://wiki.realxtend.org/index.php/Getting_Started_with_Tundra. Date of data acquisition 31 August 2011.

6. Presentation of realXtend.org by Toni Alatalo at NVWN Monthly Meeting. Available at: http://nordicworlds.net/2011/04/16/presentation-of-realxtend-org-by-toni-alatalo-at-nvwn-monthly-meeting. Date of data acquisition 1 September 2011.

7. RealXtend Association formed. Available at: http://www.hypergridbusiness.com/2011/04/realxtend-association-formed. Date of data acquisition 1 September 2011.

8. An Entity-Component Model for Extensible Virtual Worlds. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5928309&tag=1. Date of data acquisition 1 September 2011.

9. COLLADA – Digital Asset Schema Release 1.5.0. Available at: http://www.khronos.org/files/collada_spec_1_5.pdf. Date of data acquisition 1 September 2011.

10. Rex Community wiki. Available at: http://wiki.realxtend.org/index.php. Date of data acquisition 14 September 2011.

11. Naali viewer 0.4, Tundra SDK 1.0: Available at: http://groups.google.com/group/realxtend-dev/browse_thread/thread/c08923957f94264d. Date of data acquisition 14 September 2011.

12. Tundra Entity. Available at: http://www.realxtend.org/doxygen/_scene_model_page.html. Date of data acquisition 14 September 2011.

13. What is Sirikata?. Available at: http://www.sirikata.com/. Date of data acquisition 14 September 2011.

14. The Khronos Group Inc. Available at: http://www.khronos.org. Date of data acquisition 9 October 2011.

15. Khronos Group. Available at: http://en.wikipedia.org/wiki/Khronos_Group. Date of data acquisition 14 September 2011.

16. OpenGL 4.0: Competes With DirectX 11. Available at: http://techie-buzz.com/foss/opengl-4-0-compete-directx-11.html. Date of data acquisition 14 September 2011.

17. Frequently asked questions. Available at: http://www.google.com/sketchup/3dwh/faqs.html. Date of data acquisition 14 September 2011.

18. ASSIMP – Open asset Import Library. Available at: http://assimp.sourceforge.net/lib_html/index.html. Date of data acquisition 14 September 2011.

19. What OGRE is and what you can do with it. Available at: http://www.ogre3d.org/tikiwiki/Getting+Started. Date of data acquisition 14 September 2011.

20. AssImp Loader for Ogre3D. Available at: http://code.google.com/p/ogreassimp/. Date of data acquisition 14 September 2011.

21. What is FreeImage?. Available at: http://freeimage.sourceforge.net/. Date of data acquisition 14 September 2011.

22. Loeliger, J. 2009. Version control with Git. O'Reilly Media: New York.

23. Fitzek, P. - Mikkonen, T. - Torp, T. 2010. Qt for Symbian. John Wiley & Sons, Ltd: United Kingdom.

24. Backface Culling. Available at: http://content.gpwiki.org/index.php/Backface_culling. Date of data acquisition 14 September 2011.

25. Naali and Tundra Architecture Overview. Available at: http://www.realxtend.org/doxygen/_tundra_architecture_overview.html Date of data acquisition 19 September 2011.

26. Redmine – Wiki. Available at: http://www.redmine.org. Date of data acquisition 29 September 2011.

27. Towards a Definition of “Virtual Worlds”. Available at: http://journals.tdl.org/jvwr/article/download/283/237. Date of data acquisition 2 November 2011.

# APPENDICES

Appendix 1. Loading Textures

Appendix 2. Converting Collada to Ogre

Appendix 3. JavaScript code for animation testing in Tundra

Appendix 4. Dataflow diagram

## LOADING TEXTURE

```
// Get the image format
FREE_IMAGE_FORMAT imageFormat = FreeImage_GetFileType("boy_10.jpg",0);
// Load using the jpg decompressor
image = FreeImage_Load(imageFormat, filename.c_str());
if (image)
{
    image = FreeImage_ConvertTo32Bits(image);
    // Get the image dimensions
    int w = FreeImage_GetWidth(image);
    int h = FreeImage_GetHeight(image);
    // Allocate memory for image pixel data
    char * imageData = new char[w*h*4];
    // Convert image to RGB-data and fill imageData
    FreeImage_ConvertToRawBits((BYTE*)imageData, image, FreeI
mage_GetPitch(image), 32, FI_RGBA_RED, FI_RGBA_GREEN, FI_RGBA_BLUE, TRUE);
    GLuint numTextures = 1, *textureIds;
    // Generate texture name
    glGenTextures(numTextures, textureIds);
    // Bind a named texture to a texture target
    glBindTexture(GL_TEXTURE_2D, *textureIds);
    // Linear Filtering
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    // Enable texture on a graphics pipeline
    glTexImage2D(GL_TEXTURE_2D, 0, 3, w, h, 0, GL_BGRA, GL_UNSIGNED_BYTE,
imageData);
    delete imageData;
}
```

## CONVERTING COLLADA TO OGRE

```
if (IsAssimpSupported(transfer->asset->DiskSource()))
{
    OpenAssetImport import;
    QString filepath = transfer->asset->DiskSource();
    bool success;
    QString parsedRef = fileLocation.remove(0, filepath.lastIndexOf("/") +
1);
    if (parsedRef.startsWith("http"))
        success = import.convert(filepath, true, parsedRef);
    else
        success = import.convert(filepath, true, filepath);
    if (!success)
    {
        LogError("AssImp failed to load the file " + fileLoca-
tion.toStdString());
        return;
    }
    // Store all the material stuff into a map
    this->materialMap = import.GetMaterialList();
    // Vector is needed for keeping indexes in correct order for each ma-
terial
    this->materialIndexMap[filepath] = import.GetMaterialNames();
    Ogre::MeshSerializer serializer;
    QString tempFilename = "tmp.mesh";
    // Serialize data into Ogre Mesh
    serializer.exportMesh(import.GetMesh(), tempFilename.toStdString());
    // Fill transfer->rawAssetData with “tmp.mesh"
    LoadFileToVector(tempFilename.toStdString().c_str(), transfer-
>rawAssetData);
    // Delete the temporary file we used for serialization.
    QFile::remove(tempFilename);
}
```

## JAVASCRIPT CODE FOR ANIMATION TESTING IN TUNDRA

```
var anim = CreateThing("boy", GetVector(1,1,1), GetVector(0,0,-5));
var anim2 = CreateThing("gugianim", GetVector(0.05,0.05,0.05), GetVec-
tor(20,15,-10) );
var anim3 = CreateThing("model", GetVector(0.1,0.1,0.1), GetVector(40,0,-
5));
var cameraEntity = scene.CreateEntityRaw(scene.NextFreeId(),
["EC_Script"]);
// Hook to tick update for continuous rotation update
frame.Updated.connect(ServerUpdate);


function CreateThing(ref, scale, pos) {
    var entity = scene.CreateEntityRaw(scene.NextFreeId(), ["EC_Mesh",
"EC_Placeable", "EC_AnimationController"]);
    entity.SetName(ref);
    entity.SetTemporary(true);
    var r = entity.mesh.meshRef;
    r.ref = "local://" + ref + ".mesh";
    entity.mesh.meshRef = r;
    r.ref = "local://" + ref + ".skeleton";
    entity.mesh.skeletonRef = r;
    var t = entity.mesh.nodeTransformation;
    t.scale = scale;
    t.pos = pos;
    entity.mesh.nodeTransformation = t;
    return entity.animationcontroller;
}


function ServerUpdate(frametime) {
    anim.EnableAnimation("Animation0", true);
    anim2.EnableAnimation("animation", true);
    anim3.EnableAnimation("Animation0", true);
}


function GetVector(x, y, z) {
    var vec3 = new Vector3df();
    vec3.x = x;
    vec3.y = y;
    vec3.z = z;
    return vec3;
}
```

## DATAFLOW DIAGRAM