**3D Game Programming Final Project Report, NCTU**

Name/ID : 許筑淨 / 0116062

Name/ID : 曾亮齊 / 0116057

Name/ID : 王秋玄 / 0110714

THIS MUST BE YOUR GROUP’S OWN WORK! **Yes !**

**Introduction**

This is an obstacle-avoidance game that the player should avoid hitting the obstacles by clicking different keyboard combination, for example, ‘space’ for jumping, ‘z’ for sliding, and ‘x’, ‘c’ are for the shooting buttons. If the player hits the obstacles, the game will be over.

**Game Story**

In this game, the player will have the chance to help our lovely heroine, Rin cross through everything that hinders her on her way to school. If the player successfully helps Rin not to be late for school, Rin maybe fall in love with our player.

**Game Type & Game Target**

It is an obstacle-avoidance game. The player scores will continuously increase as long as player lasts in the game. The player can also gain extra scores by collecting stars.

Players can have our heroine to jump, slide and shoot blue/red bullets by pressing “space”, “z”, “x”, “c”. If the obstacle is a standing wall, the player should press “space” to jump over the standing wall. When about to hit a car, the player can choose to either jump or slide under through the car depends on the height of the car. When facing a red/blue tall wall, the player should shoot red/blue bullets corresponding to the color of the wall to wreck the wall in order to keep moving. Red bullets can be shot by pressing “c”, and Blue bullets are shot by pressing “x”.

**Game Platform**

Game Engine – Ogre3D, the version of Boost, Ogre, and .NET are all the same with the homework’s version. Therefore, the system that can run the homework program should be able to run our game as well.

Addition 3rd party Libraries – There are many functions that we rely on other libraries.

Physical Engine – Bullet3, OgreBullet

Game Scene Design – DotSceneLoader, TinyXML

**Game Design**

Inspiration –

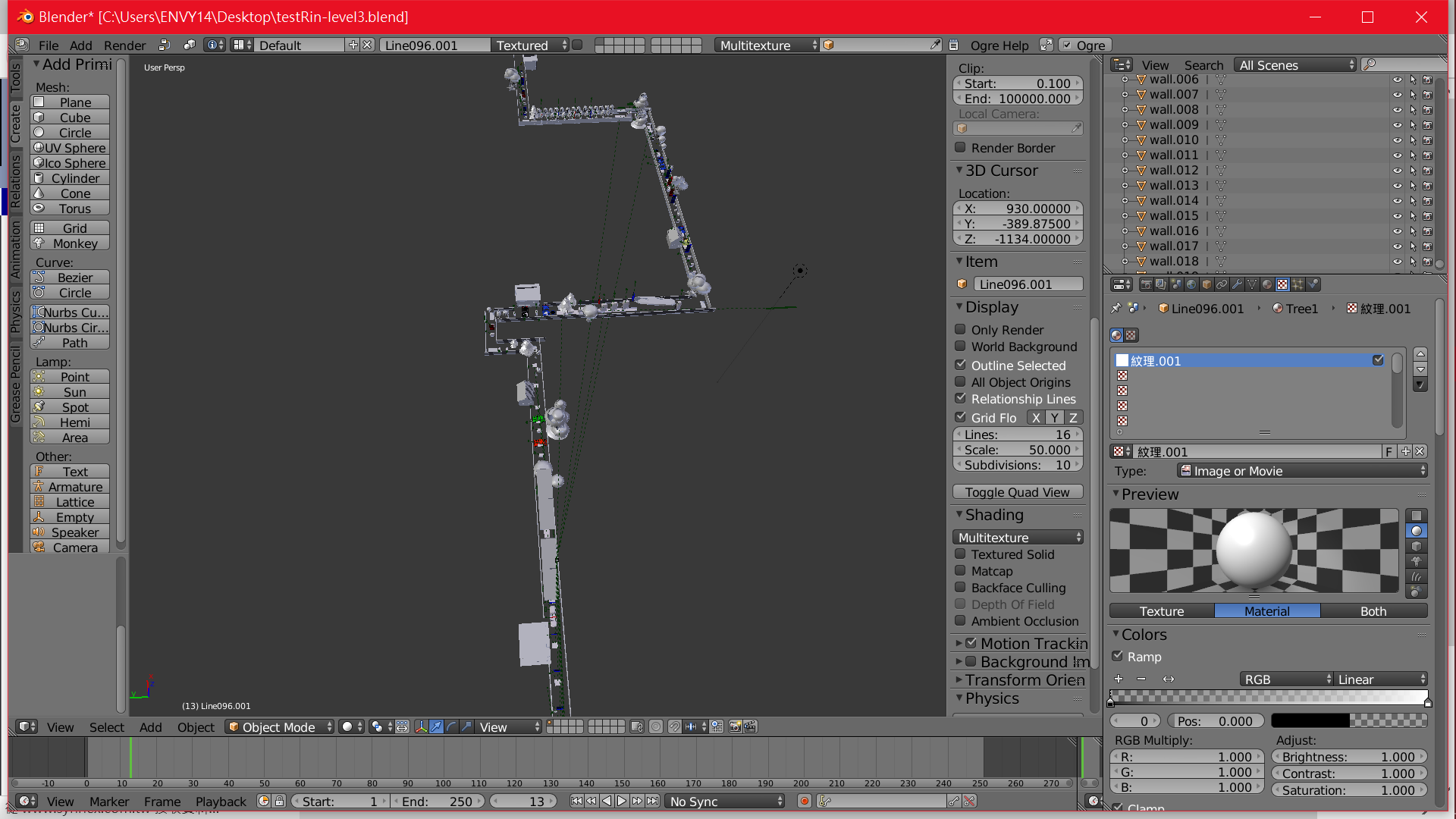
This game is inspired from two games, Geometry Dash and Temple Run. Geometry Dash is a 2D Rhythm game that the player follows the rhythm and tap to avoid fatal obstacles. Temple Run is also a well-known smartphone game that player uses multiple action to avoid fatal obstacles and survive.

Game and level Design –

This game is a 3D camera following action game. The basic playing concept is music and avoidance, and a cute heroine (player’s model), since everyone loves cute high school girl.

The main scene design and setting use Blender to create scene related files for scene loader to load materials, models, so basically all the design things are done in the Blender. The following picture is a pure model scene without material rendering.

The whole game scene will be a single direction walkway with occasional turns. There are trees, houses and fences as decorations. The main obstacles will be brick walls, cars, and colored high walls. There are also supporting appliances in this game, such as trampolines.



The following picture is the completed scene with material rendering.



The game has three main levels. The first level “Level 1” will be the simplest for the player, and the last level “Level 3” will be the most difficult one for the player to survive through. The difficulties players will face when challenging this game will be the increasing number of actions that players will have to control as the level increases.

In “Level 1” players would only have to deal with “jump” and “turn” action, and the number of fatal obstacles would also be the least.

In “Level 2” players would need to control the “jump”, “turn”, “slide” and more fatal obstacles precisely to survive through level 2.

The “Level 3” is the most difficult, including the action listed above and “shooting”. When shooting players would also pay extra attention to the color of the target wall, therefore, to speak honestly there are two actions players need to spare more focus on - the Blue bullet shooting and the Red bullet shooting. As level third has been tested the most time, it could honestly to be said the most difficult and distracting design.

**Game Analysis**

Practical Hardness (platform) –

In this game we use Ogre3D Development kit to practice. Compare to others who use Unity developing Tool kit, we spent almost half- third of our time dealing with basic system level problems, including physical system, graphics user interface, audio library and scripting system. Frankly, it took us about ten times longer than other group to handle the functions we’ve spoken above. Take physical system for example, it took nearly fifty hours integrating the whole system. Due to limit time, it is extraordinary difficult to be done in both delicate and complicate way.

Difficulties (for player) –

It will be a very hard game for ordinary players even in the simplest Level 1. All the controlling in the game must be precise enough to get through the facing obstacles.

The game does not provide any save/load function, which means it either requires players to have immediate and correct reaction toward the upcoming obstacles or to learn from multiple failure experience.

Characteristic:

First of all, we have a very adorable heroine as our main character. In our opinion, whether the main character is lovable or not is important, since that’s one of the significant impressions of the game.

Also, the game control is not as easy as you see in the first impression. Because the player should be aware of the obstacle to figure it out what they should do next in short time, and the different keyboards combination and obstacles increases the difficulty. It’s a game that requires the player has enough skills to pass even in the first stage.

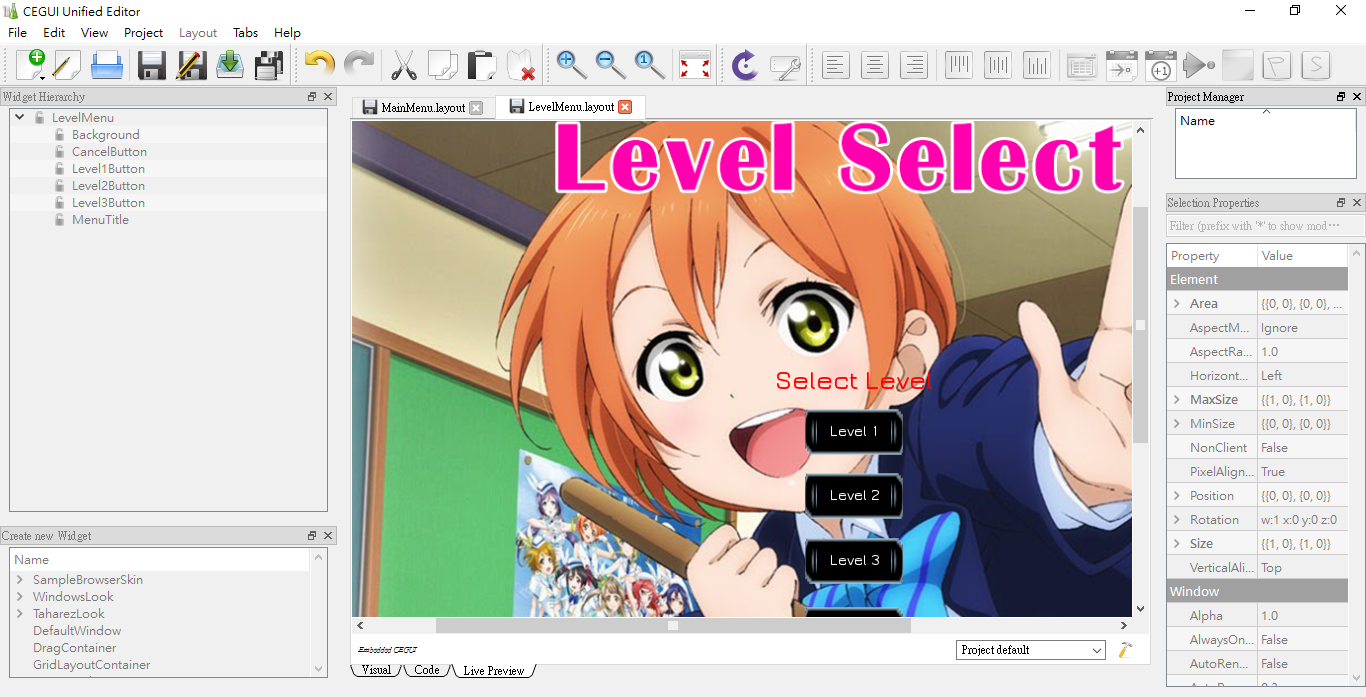
Delicacy

For the screen delicacy, we achieved the basic requirement, for example, the total scene design, the particle system, the model animation, and the sound effects. And for the game delicacy, in order to increase the difficulty and diversity of the method that player get points, we give several kinds of method for the player to gain the score and even decrease the points as well. First of all, the score increases as the game progresses. Secondly, after we shoot the right obstacle, the player can gain higher scores than the first way. On the other hand, it will be a reduction if the player shoots obstacle with the wrong bullets. Also, if the player presses the sliding button too often, the increasing speed of the score will become slower for the reason of not allowing the player slides when he/she does not need to.

**Technical Document**

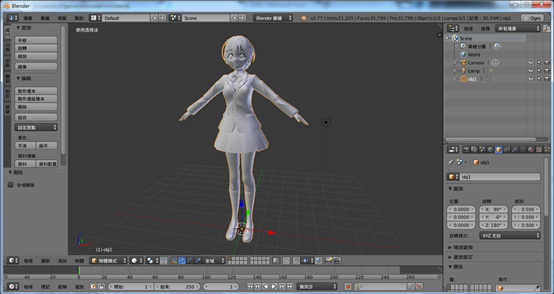
GUI (CEGUI, XML)

1. Because the built-in GUI of Ogre3D is not so easy to use, we employ a third party GUI library: the CEGUI. It is also a library written in C++, so it is pretty easy to integrate with our program. The CEGUI library also provides API for integrating with Ogre3D. In our project, we implement many window classes for different GUI interface. By using inheritance, the code is pretty easy to understand.
2. Another good thing for CEGUI is that it provides an external GUI editor called CEED. We can design the layout of GUI window and export them into XML files. After that, we load these files info program and attach event handlers for them. The following picture is the snapshot of CEED editor.



3D model Animation

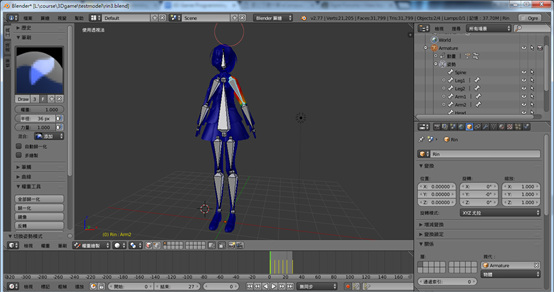
1. We have our model without any weight and animation in the beginning. (The model is converted from pmx file to obj file)



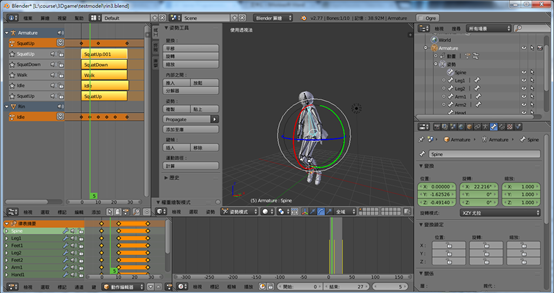
2. First of all, we have to create the armature corresponding to the model. However, there is still some bug after you link the model to the armature with automatic weight.



3. So we should modify the weight of the model manually and ensure all of the model’s surfaces are painted otherwise those that aren’t painted will stretch to the origin. This is a bug caused by the conversion between Blender and Ogre3D.



4. Create the animation by moving each of the armature. After an animation is created, add an nlastrip, so that this animation can be used in Ogre3d by programming.



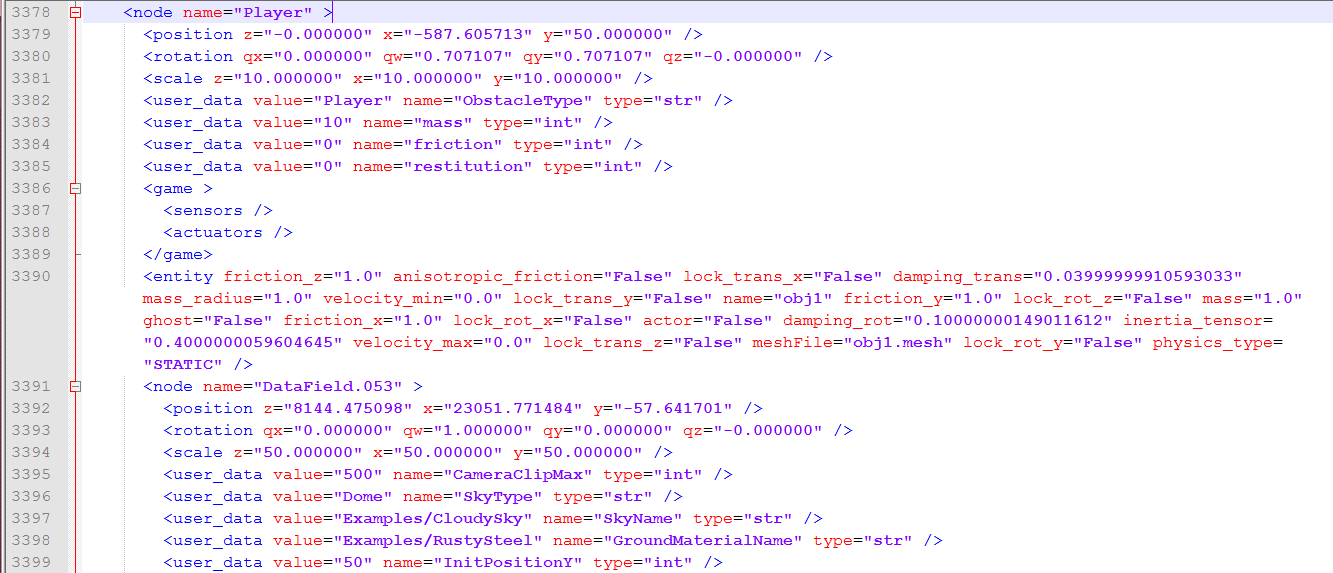
Physical system (Bullet Physics)

* We integrated the third-party physical library - The Bullet Physics System into our game project. The Bullet Physics System is a platform independent library written in C++. Any program written in C++ can use it to provide dynamic world. Because the Bullet system does not bind to rendering system, it has its own root object to manage all objects with physical effect. To link the Bullet system with the Ogre3D, there is an extra Ogre3D plugging called OgreBullet. It is a simple wrapper that allows creating an Ogre3D object with corresponding Bullet object. However, the wrapper is too simple to provide every function we needs, we spent a lot of time to modify the wrapper so that we can implement more features.
* We have mentioned before that the Ogre3D meshes and Bullet object are internally independent. They only share the same position and orientation information. The following snapshot shows the Bullet object associated with meshes. The Bullet objects are drawn in green wireframe.



DotSceneScript (XML file loader)

* Because it is no sense to hard code the game level in source code, we must employ external scene file loader. After the scene file has been successfully loaded into program, we than parse the content and build the scene according to the file. We use the DotScene(stands for .scene file extension) file loader as the scene file parser, it loads scene from XML files. The scene itself is built within Blender. After everything is ready, we export the scene using Blender Ogre Exporter. This tool converts all objects in the scene to the Ogre3D mesh format (.mesh).
* The following picture is the snapshot of DotScene file.



Sound Effect (OpenAL)

* Unlike Unity, there is no built-in sound library for Ogre3D. We use the open source library - OpenAL as our sound library. We implement a simple wrapper for OpenAL so that we can easily play sound effect in out game.

**SWOT Analysis**

SWOT for game (in the vision of selling our product)

Strengths – In this game, we content a lot of 3rd party libraries, so it can be a good use of tutorial of Ogre3D and even for the game designing.

Weakness – Honestly, there’s not much people write a game with Ogre3D currently, it can be figured out by searching internet resource. However, there are not many resources for it, so it might be more difficult for the programmer to write a game in Ogre3D than other engine, for example, Unity.

Opportunities – It’s a good prototype for the use of tutorial. For the one who starts learning programming, it better for them to write program with dealing lower level problems.

Threats – Just like the weakness, it take more time and effort to finish a game in Ogre3D, including searching the resource. So, it might be weeded out by times.

**Man power**

王秋玄 :

Keyboard Library –

Create a class library that includes the function that the program needs. For example, you have to write more segments to implement detecting the trigger of one button usually, and it will be more unreadable as well. So it’s better to implement a library that provide the function “isKeyTriggered()”.

Character’s 3D model –

It includes the creation of the armature, weight painting of the model and the animation. Also, ensure the model can do the right animation in Ogre3D, and it includes the animation’s creation part and Ogre3D programming part. Since it may looks differently in Ogre3D because of the conversion.

許筑淨 :

Camera Control –

Build class for camera control. This will allow the looking direction and the position of the camera be attached to our player, and hence follow our player.

Game stage Design –

Base on the expecting difficulty to design and deploy the obstacles, supporting appliances, and decoration in the game scene.

GUI design and 2D picture design –

Using GUI Toolkit CEED and photo shop to design and complete the art concept of the game.

曾亮齊：

Physical Engine –

Integrates the Bullet Physics system into Ogre3D. After doing this, we can easily create a Ogre3D object with physics system embedded in it.

DotSceneLoader –

Integrates and improves the third-party DotSceneLoader for Ogre3D. Fix some bugs and add extra functions to create complex scene.

GUI engine –

Integrates CEGUI system into our project so that we can create lots of GUI windows in the game.

Sound effects –

Integrates OpenAL in the game and provides easy API for playing sound clips.

Game testing.

**Milestones**

March:

Apply the 3rd party libraries’ physical engine.

Finish the Keyboard Library.

Finish the Camera Controller.

April:

Finish the combination of DotSceneLoader and physical engine.

Start learning our scene editor: Blender, and star the game stage design with it.

May:

Game GUI.

Main character’s 3D model design.

Game’s sound effect.

June:

3D model’s animation.

Game’s 2D picture design.

Game testing.

**Discussion**

As long as we decided to do this project with Ogre3D, we are already acknowledged that we need to spend more time to finish this whole project than others who do not use Ogre3D. During process of developing we did face a lot of difficult technical problems, such as integrating third party library, self-made model animation and GUI (graphic user interface). None of our group member has any game developing experience, and we could only count on ourselves to solve problems we faced. We used internet to look for documents and instructions, we tried and error countless time during the developing process. In the end, though it may not seem to be perfect, still by efforts of all the group members and schedule management , we have created a quite completed, playable game.

**Conclusions**

In this term project we‘ve learned a lot about designing of low level and basic concepts. Since Ogre 3D does not provide as many convenient and completed existing game design system for developers as Unity does, we have to study lots of study and research papers and uses many third party library to complete our game. So far the game can neither be said perfect nor delicate. However, we did spend a lot of time learning and practicing on most of the basic game concepts, which is kindly what we have expected to learn and practice when taking this course.

**References**

[Blender] <https://www.blender.org/>

[Bullet Physics] <http://bulletphysics.org/wordpress/>

[Ogre Bullet] <http://www.ogre3d.org/tikiwiki/OgreBullet>

[Ogre DotScene] <http://www.ogre3d.org/tikiwiki/DotScene>

[TinyXML] <http://www.grinninglizard.com/tinyxml/>

[Blender Ogre Exporter] <http://www.ogre3d.org/forums/viewtopic.php?f=4&t=61485>

[Obj Model] <https://www.cgtrader.com/>

[Animation] <http://www.ogre3d.org/forums/viewtopic.php?f=8&t=79616>