Developing a Marble game

Results of a practical course at the Chair for Computer Graphics and Multimedia
(RWTH Aachen University, Germany)

Jasper Veit Manousek* Steffen Fündgens† Author 3‡ Author 4§

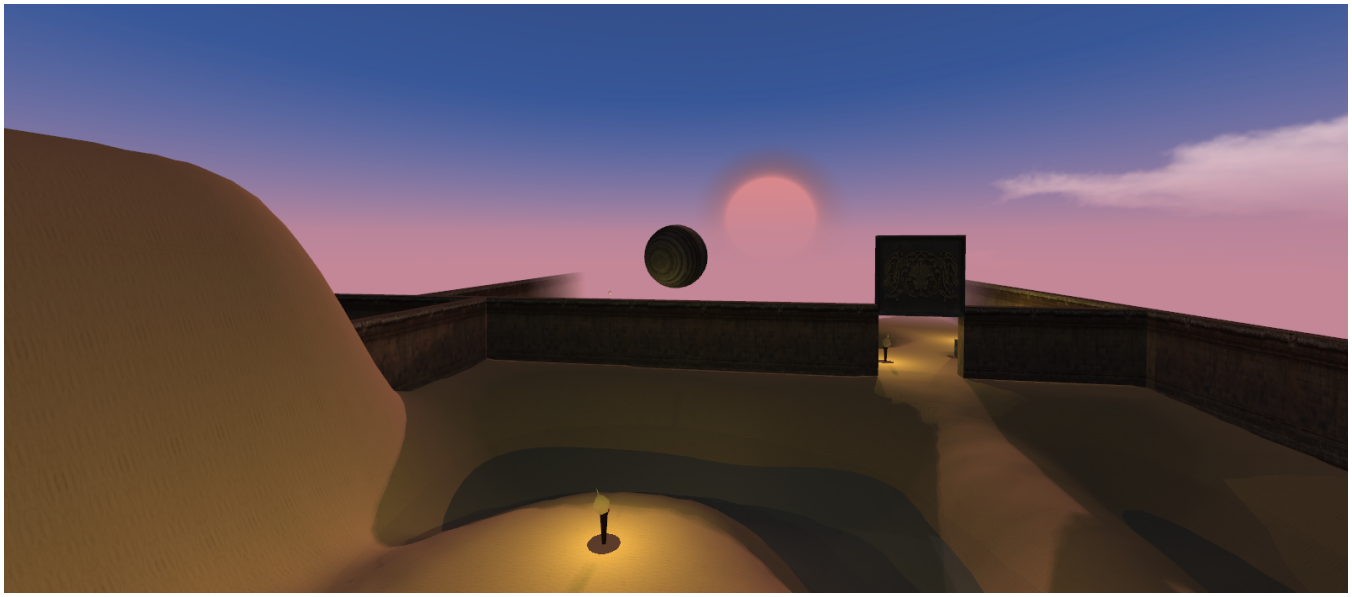


Figure 1: *Teaser*

Abstract

Marble racing games are a sub-genre of games where the player controls a marble to either run races, solve puzzles or complete some other tasks. In Saxum the player must travel through an open level completing some basic switch tasks and a simple puzzle to help the sun rise again.

1 Basic Layout

Our game is designed around the main level class. In it we load all the data from our XML files as needed. From here we also have access to the other two main classes, the physics and the graphics classes. Additionally we have the list of all our objects here.

2 Details

2.1 Loading

Our game plays mainly on a height-map. We load the height-map from a greyscale png and pass the height-data to a framebuffer, so ACGL can render it. We also pass the height-data to *Bullet Physics*, to create a colision shape.

On the height-map we place objects that we load from two XML files. The compositions file defines some properties for classes of objects, like the names of the obj file and the texture file, and parameters for lighting and for the physics. It also defines compositions that are made up of multiple objects that can be scaled, rotated and translated individually. One compositions file could get used for multiple levels. The level XML file defines which compositions are placed where. This part of the file gets generated by our converter. In addition to that, the compositions can also be scaled and rotated manually here.

The converter is a separate executable that takes the path to a PNG file as input. In the PNG file, compositions are placed as pixels. We decided to let the red value of the pixel identify which kind of composition it is. The green and blue values are written by the converter and used to identify single compositions. This way manual changes in the generated level XML can be kept when the converter is run again.

2.2 Triggers

Because a lot of the gameplay in Saxum is focused on solving challenges and activating events, we decided to integrate the scripting language LUA to make our triggers customizable. In the level XML we can add triggers to objects. We define a region in global space and when the object enters or leaves the region a LUA script is called. The script can then activate different events like opening a door or letting the sun rise at the end of the level.

2.3 Graphics

Some techniques are a little bit more advanced, so that you can even use subsections.

*Jasper.Manousek@rwth-aachen.de
†Steffen.Fuendgens@rwth-aachen.de
‡author3@rwth-aachen.de
§author4@rwth-aachen.de

2.4 Physics

The physics is based on Bullet Physics, with a simple callback function used to relay the data to the graphics pipeline. We decided to use a btvhTerrainShape for the terrain. This is much more effective than using a triangle mesh. Additionally we have spheres, boxes, and a few other basic primitives. Finally we implemented btvhTriangleMesh shapes and concave triangle meshes, which are more strenuous for the system, but allow for moveable triangle meshes.

Unfortunately Bullet Physics proved unable to handle many forms of constraints that we required. For this we created two instances of spring constraints. The first is a spring constraint which creates a force attempting to keep two rigid bodies a set distance apart. The second was a derivation of the first, generating a force in an attempt to confine the rigid body to a certain position. This constraint is what we used to create switches that could be embedded within the terrain by filtering out the collision between the two bodies. We also used a similar constraint to create a physics based camera. The camera attempts to follow the player at an angle specified by the player. Finally we used the constraints to get the player to float in the air. To do this a constraint is added via the LUA script, allowing the player get an improved view of the sunrise.

Additionally the respawn animation was created using the physics engine. It lets the ball continue to rotate, while changing certain properties of it, especially how it will move and that it will ignore the collision with all other objects. After it sinks a certain distance it will reappear in a similar fashion before its normal physic state is restored.

The physics came with many challenges, partially stemming from our integration into a new field, but also partially due to the nature of Bullet Physics itself. If bullet does not recieve enough updates the simulation becomes jagged and unrealistic. To solve this it is suggested to recall the world multiple times per frame to allow a more accurate simulation. Unfortunately, especially with the introduction of the physics based camera, problems arose. To solve this we not only recalled the complete update method within the physics, but also recall the complete update method in the level to stabilize the experience.

Figure 2: *Use one image.*

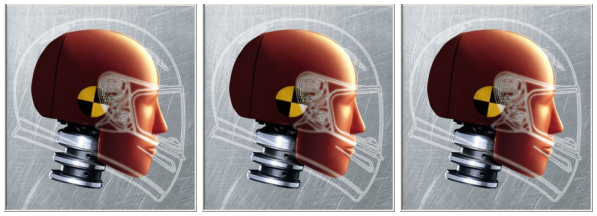
2.5 Content Creation

3 Conclusion

Our group had no prior experience with such large projects. Though our lack of previous experience hampered our progress, we have managed to amass a great deal of experience in game design, project management and teamwork, as well as knowledge in the underlying framework of games.

Our game is an interesting twist on an old concept, and our play tests have shown that we manage to achieve a notable awe factor for those who complete it.

Through the experience we have gained we are confident that all our future projects, especially those in the discipline of game programming will benefit greatly.



(a) (b) (c)

Figure 3: *Use several images.*