Developing a Pinball Game

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

Benjamin Hohlmann* Matthias Möller† Laurin Scholz‡

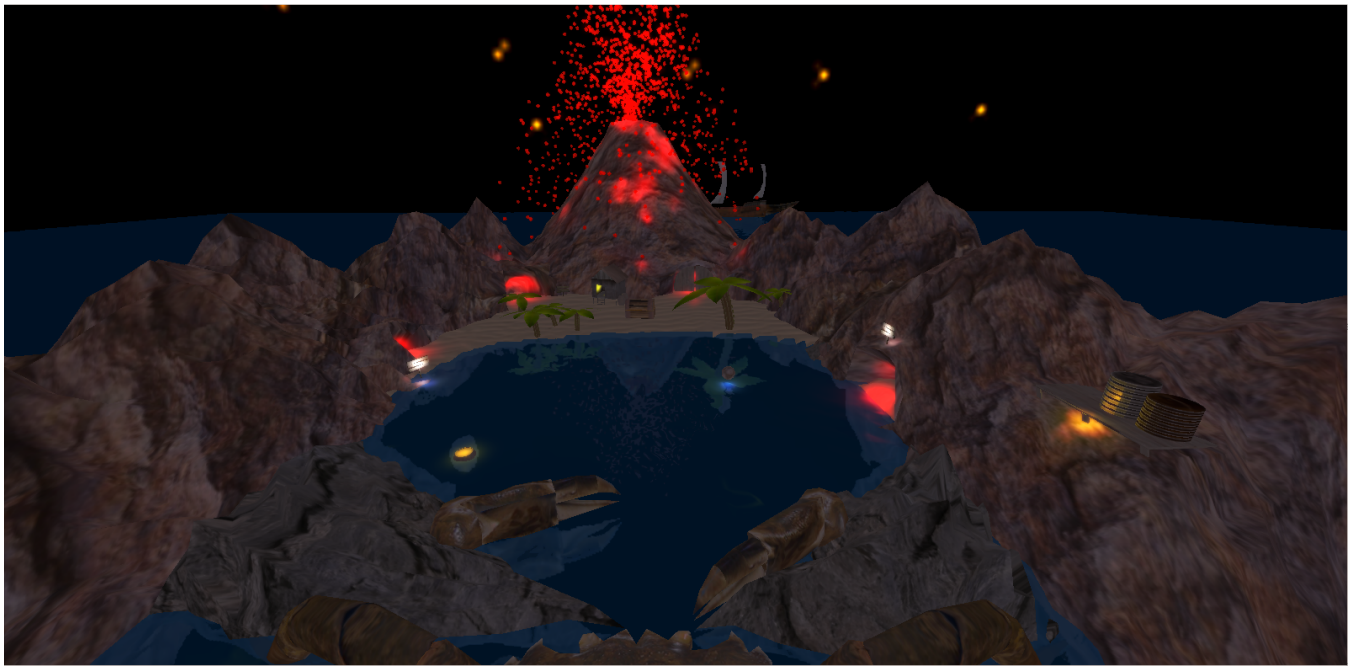


Figure 1: *The players view of the table. The crab-claws work as flippers.*

Abstract

The objective was to develop a small but fun pinball game with some modern graphics effects. We went for a peaceful tropical island scene, because it's something different then the usual pinball themes. We tried to fit everything into the scene, so the ball is launched by a cannon, points are displayed with coins and crab-claws work as flippers. (cf. Figure 1).

Our job was to create the game logic, graphical effects and models. We started from a simple ACGL scene using the Qt Framework, already including the physics engine bullet. Apart from the XML-file containing the scenegraph, the work was done in C++.

Keywords: game programming, pinball game, tropical

1 Gameplay

The gameplay is the same as known from other pinball games. It contains quests which are not described by text, but by lights with equal color. For instance if a cave is hit, the sign beside it stops to glow. After hitting three caves, the fourth is unlocked, allowing you to unleash the vulcano and achieve an additional ball.

In the game the palms work as bumpers, some kicking the ball in a predefined directions, some only kicking the ball away, but always in strength depending on the balls speed. The caves work as tunnels. When the ball enters one cave it's shot randomly out of one of the caves, whereupon some caves like the vulcano cave only work as entrances.

2 Content Creation

This part of the project includes creating the models and textures and an editor to create levels. However, we soon realized that we don't have the manpower to complete the editor and abandoned it. It's original intention was to create the scenegraph in XML format, but as we only have one pinball table, the overhead of creating the editor isn't worth the later benefit of not having to type the XML file directly. We still stayed with using XML as format, as loading the scenegraph from XML doesn't require rebuilding the project, so fine tuning the scene becomes a lot easier and faster.

Another idea we had in the beginning was to use a heightmap for the terrain of the isle. Although it is implemented, we only use it to separate the land from the water. The Mountains however are mere mesh objects, otherwise it's not possible to create caves, which make up for an important part of the gameplay. This way we also saved the work of adding the heightmap to the collision world.

For creating the models, we planned to use 3ds Max, but soon switched to Blender, because of a lot of minor problems we had with the former. We only exported simple .obj files and did not create the scene within blender. This is a lot easier than creating a plugin for Blender, which could have been an alternative to creating an editor.

*Benjamin.Hohlmann@rwth-aachen.de
†Matthias.Moeller@rwth-aachen.de
‡Laurin.Scholz@rwth-aachen.de

To simplify the placing of objects without an editor, we decided not to rotate the world coordinate system, including the heightmap and every single object, but instead rotate the gravity vector.

The game atmosphere could be improved greatly by adding a sound system, a nice intro and a menu, but we lacked of enough time for these optional specials at the end.

3 Game Logic

For the quests and other gameplay requirements like noticing when the ball is lost we needed an event system. Implementing this system was not easy, as it uses Bullet for the collision detection and the bullet documentation mostly consists of its code itself.

We also created an animation system which provides not only simple animations (e.g. linear movement/rotation) for the XML-file but also some more complex hardcoded animations like the ships movement and animation of the vulcano-fragments. Since our animations can only rotate around the middle of the object, we couldn't use it for the flippers, as those are rotated around one end. Again we had to deal with bullet, because the flippers are physic objects, what caused some trouble.

Built on the event system and the animation system is the quest system, which allows us to define multiple triggers that have to get activated first to unlock a major target with a defined unlock animation. When the major target is hit it's locked again with an animation and the quest gets reset. In the end we created only two quests due to lack of time, but with the system available it wouldn't have been hard to create more quests. The quest system is not available in the XML-file as time was running short and as a result of the low number of quests it was easier to create them manually.

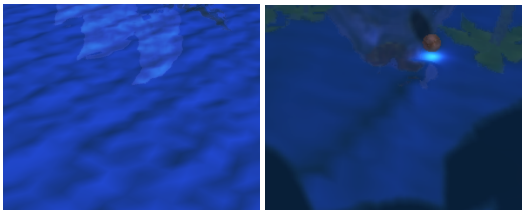
Because of bullets bad documentation, we also abandoned another function: Importing .obj models into the physics world. Although this function seems quite important, it turned out to be easier to built the physics world only out of bullet native objects like planes, boxes and cones. Another advantage of this approach is the simplicity of the physics world, which on the one hand makes the physics simulation very fast and on the other hand prevents edges that make the ball jump.

4 Graphics

While the tropical topic doesn't allow fast non realistic effects e.g. big glow, the rendering part was more concentrated in generating geometry, supporting also the content creation part, and the scene color.

An example for the scene color is the glow effect. It is a nice effect which supports the scene color, especially the beach. But the glow has to be nearly invisible, since we have not a Sci-Fi theme.

The water is one essential part of the game, since it is used for the background and for the lower part of the table. The background water is designed to be fast and not well detailed, so it is generated by some simple sinus curves via GPU. In contrast to the background, the table or foreground water should be a part of the game (See figure:2). Due to this, we chose the fast approximated Heightfield water simulation [Bridson and Müller-Fischer 2007] allowing the ball interacting with the water. Merging both sorts of water is done via shaders, since simulated water is computed by the CPU and background water by a vertex shader. As an extra, water supports real time water reflection, reflecting the scene and the skybox, which reflection is important creating a realistic color behaviour of the water during day and night.



(a) (b)

Figure 2: *(a) background water and (b) water ball interaction.*

Dynamic daytime is an effect which was simple to implement but hard to polish. Moving directional light, a synchronized camera with the directional light for the shadow map and a changing skybox with dawn were the only things to implement. Surprisingly, finding the right color for the sun and the ambient color, given a good light atmosphere at each time, consumed a lot of time. Also, it was a huge workload to correct the bias value for the shadow map getting the lowest flickering while moving and no shadow peter panning on objects. Another problem was the synchronization of the light direction and the shadow camera since the shadow camera needs always the whole scene, creating nice shadows. The solution was to use only a approximated synchronization, causing a lower degree of realism. The player's camera doesn't move, so we could define a static spline for the shadow camera depending on the daytime.

As a result of a non transparent water model, we decided using deferred shading, giving the possibility of adding a huge number of lights into the scene. This way, we could easily add a volcano eruption with lots of lights lowering the performance only slightly.

Instead of creating a complex particle system, what was not possible due to lack of time, we simply used many geometry objects in the scenegraph for the vulcano eruption. This was possible because there were enough resources left.

5 Conclusion

With the benefit of hindsight we can say that using a XML file for the scenegraph is a good idea, as it enhances the workflow from the object creation to ingame integration a lot.

As we are only a small group of three people, splitting the project into the three parts Content Creation, Game Logic and Graphics worked very well. This way, everyone can focus on his own part and whenever two parts overlap it is easy to sit down and implement the code together. Because we use the Model-View-Control pattern, the overlap between different parts is minimized, allowing us to work mostly independently. This however led us to lose sight of documenting our code, which was not that big problem, as everyone was easy reachable for the others to explain his code if needed.

Not implementing the editor in the first place could have saved enough time to add other optional, but nice functions, yet apart from this and a few other details, the overall approach worked out very well.

References

BRIDSON, R., AND MÜLLER-FISCHER, M. 2007. Fluid simulation. *SIGGRAPH 2007*.

FINCH, M., AND WORLDS, C. 2007. *GPU Gems*. Addison-Wesley.