Developing a Pinball Game

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

Matthias Heinrichs[*] Maxim Jourenko[†] Aivar Kripsaar[‡] Florian Langel[§] Felix Rath[¶]

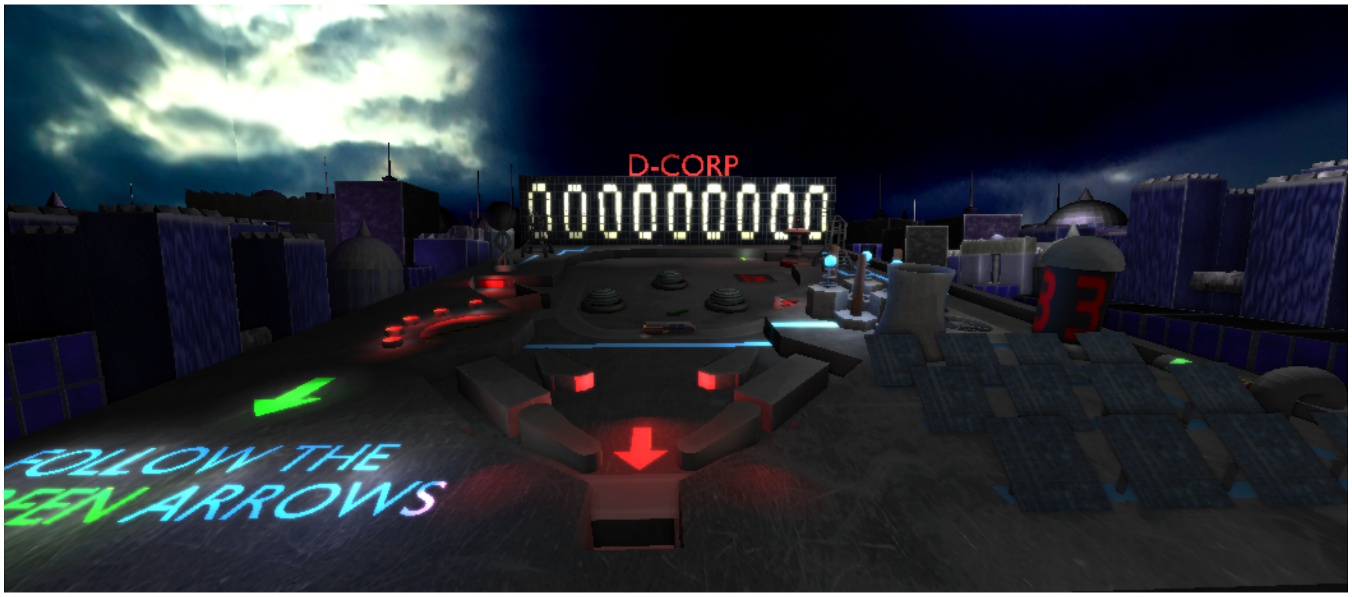

Figure 1: *An overview of the pinball table.*

Abstract

Our objective was to build a pinball game in a 3D graphics engine, employing various methods and tricks to create an experience that is visually compelling, while also being entertaining from a gameplay perspective.

While a few gameplay events served to make the game less one-note, due to being restricted to pinball, most of the focus fell on making the visual presentation stand out. To this end, multiple graphics effects were used to create the image of an industrial district in a slightly dystopian futuristic city. Of prime importance were various lighting techniques, such as glow. Animated objects and sounds were then used on top of that to breathe a bit of life into an otherwise bleak environment.

Keywords: game programming, pinball game

1 Gameplay

Figuring there wasn't a lot we could do with the base structure of the game while staying true to what pinball is, we decided to stick to the basics, while spicing the game up with interactive and non-interactive events.

We planned on including a multiple stage boss fight for our game, mostly involving hitting particular objects on the game table with the ball. Due to time constraints, we had to abandon the boss fight, with the only surviving larger gameplay event being a relatively straightforward rocket launch, which is achieved by reaching a certain minimum score and activating a couple of collision events with the ball. All in all, we managed to create a pinball table that feels like it reacts to the actions of the player.

2 Graphical Presentation

For our game we settled on a futuristic setting, and decided to model our pinball table after a modern/futuristic city, with a dark, unfriendly and gloomy looking atmosphere. In order to achieve this, we mainly focused on playing around with lights and shadows.

We employed a simple version of SSAO to darken most of our scene, as our camera position was static and we found that further techniques for shadowing (e.g. Shadow Mapping) were not required. This was especially apparent in our background, which consists of multiple rows of skyscrapers.

After deciding that most of our scene should be rather dark, we wanted to create a futuristic and modern feeling by distributing a lot of lights throughout our scene, some of which should move, while others would be static. Thus we decided to use deferred rendering, as it allowed us to use a high number of lights without without huge drops in performance. From the beginning we did not plan for any transparent objects to be in our final scene, so we chose not to focus on transparency, which requires quite a bit of extra effort to add, when using deferred rendering.

Before implementing deferred shading, we implemented a glow for our game, based on a brightpass shader. This gave us the possibility to brighten up some of our surfaces, which was crucial to creating

[*]matthias.heinrichs@rwth-aachen.de
[†]maxim.jourenko@rwth-aachen.de
[‡]aivar.kripsaar@rwth-aachen.de
[§]florian.langel@rwth-aachen.de
[¶]felix.rath@rwth-aachen.de

the believable image of a futuristic looking city. Our glow also performed well later on when we decided to add a power plant and energy conducting lanes to the base of our table. The glow made the bright texture look as if it was energy being transmitted giving off a natural glow.

For our background we chose a simple skybox with a dark cubemap as our texture, as it fit the general style of our scene. As another feature for our futuristic city, we planned for a few objects in our scene to have reflective surfaces. Implementing reflections proved to be a very difficult challenge, which cost us a lot of time. The final result was was far from ideal, with only one surface reflecting our scene, but not our skybox. Even with a few setbacks, we managed to create an atmospheric scene filled with bright lights, yet also possessing a touch of gloom, which is what we set out to do in the beginning.

We also implemented normal mapping, however its effects are very subtle in the final scene. Additionally, only a few objects received normal maps.

3 Additional Presentation

3.1 Audio

Phonon was used to implement the sound system. Including phonon was been quite simple and it works quite well on most Linux systems we tested on because it is a Qt-library. We had problems with it on Windows systems, but we still decided to use it, since Windows compatibility was not a priority, and Phonon fulfilled most of our requirements.

However, there were also issues with the sound system. Stopping and deleting sound objects had a huge overhead which resulted in an unstable framerate on systems with low hardware specs. Not stopping and deleting the sound objects resulted in a stable framerate, but lead to memory leaks. To solve that problem, we resorted to multithreading, giving each sound its own thread. Whenever the thread terminates the allocated memory gets freed.

3.2 Animations

A rather simple animation system is used to animate the objects. It uses a combination of primitive animations: rotation, translation, and scaling. Furthermore, it uses parameters, for instance, which axis the transformation takes place on and which sort of transformation it has to be, i.e. linear, constant, cosine. It is also possible to specify whether the transformation is temporary or permanent.

Although the system is quite simple to implement and useful for simple animations, it is not suited for complex animations, like a train moving along a large circuit. However, we decided not to put effort into the development of another animation system. We managed to cover all of our needs with what we already had in place.

4 Game Logic

4.1 Physics

Simulating a pinball game requires simulating physics. The BulletPhysics library was used to handle that simulation. The most essential parts in interacting with BulletPhysics were adding physical representations of each object, which is called a physics rigid body, to the physics world, moving the flippers, and handling queries for whether two rigid bodies collide, which is essential for triggering events.

The basic physics simulation had been running at an quite early stage of the project, however there were severe issues with the ball tunneling through objects. Tunneling occurs whenever the ball moves too much during one frame which leads to undetected collisions, because the objects just pass by at each other. Also, tunneling seemed to occur more often the lower the framerate got. BulletPhysics, however, has several approaches to avoid tunneling which require setting parameters for functions and rigid bodies. Unfortunately, the BulletPhysics manual and it's documentation do not explain the respective parameters sufficiently so a trial-and-error approach became necessary to solve the tunneling issues. Therefore we had to adjust those parameters constantly in the course of the project, especially when the average framerate dropped due to some new feature. Still, we were successful in preventing tunneling in the later versions of our game.

4.2 Events

Events are used to react to the player's actions. Basically they are handled by using triggers, as for instance queries for collisions, time stamps, comparing certain values as the score. Additionally those triggers call callback methods which handle the requested events, as for instance, applying animations, applying impulses on the ball, updating the score, switching lights on and off.

5 Assets & Tools

During our planning stages, we decided to create 3D models in Blender and use a custom-built editor to create our level. Basic functionality of the editor included the ability to import & export levels, and move assets around the level easily. While a lot of effort was put into the editor, and a lot of the features had already been implemented to some degree, it was decided at some point that work on the editor should be discontinued in favor of concentrating on the creation of game assets, due to time constraints.

One part of the editor managed to survive the cut: the importer. The level was described by an XML-file, which would be converted into a scene graph by the importer. Although support for events and animations in the XML was intended, it was never realized due to our feverish focus on getting as many assets completed as possible. This resulted in our intendedly dynamic system becoming a semi-rigid level, with animations and events being hard coded, yet referring to objects being dynamically loaded from the XML-file. It's an unpleasant mess that could've been avoided with more experience and time.

6 Conclusion

What we learned is that task and time management is of monumental importance in a group project. Lack of experience leads to many frustrating problems which eat away a lot of time. Rigid structures should be avoided as much as possible, due to the ever-changing nature of such projects. Working together at one place or at least keeping up communication with all group members also proved to be vital.

Doing graphics programming for the first time turned out to be a big challenge, and it cost us a lot of time to understand the underlying mathematics and to implement the features we had in our final game.

Most implemented features seemed to work quite well at first glance. However, a lot of time needed to be invested into fine-tuning and debugging to get the respective features stable. We learned a lot in the process of development.