Developing a Mini Racing Game

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

Christian Bormann* Daniel Peters† Dominik Studer‡ Jörn-Michael Miehe§ Michael Anhuth¶

Abstract

Miniature Madness is a arcade racing game. The player is controlling a mini racing car on a map in the setting of a children's room. In the racing mode the player tries to complete the map in the shortest time possible. There is also a free driving mode, where one can explore the map. To provide the player with the right feeling, our focus was on creating graphics effects supporting the atmosphere of the game. The game uses the chair's ACGL framework. For our game's physics - especially the vehicle handling - we used the bullet engine. The scene is loaded from a XML file. Each object in the scene was created in Blender.

Keywords: game programming, mini racing game

1 Graphics

In our scene there is only one fixed point light source, which is arranged below the ceiling. For the lighting we use the Phong shading and the phong lighting model with different parameters for the different materials.

Since our setting is a childrens room, we wanted the scene to be realistic, but not too serious. So we placed many little and colorful objects in the scene and chose the glow effect as post process to achieve a lax and even so realistic atmosphere. The glow effect is used in order to fluroresce the bright parts of the scene (cf. Figure 1 (c)). It is implemented as a brightness filter to catch only the bright part and a gaussian filter is used to blur it. On the one hand we needed a big gaussian kernel to see the effect, on the other hand it is very expensive in computation. So we decided to blur first horizontically and then vertically to have an linear instead of quadratic complexity.

(a) (b) (c)

Figure 1: *our graphic effects.*

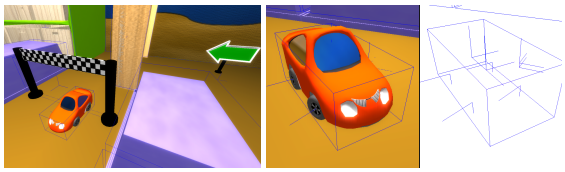
Then we implemented some effects to have a realistic looking scene. The first effect we implemented was the bumpmapping (cf. Figure 1 (a)). This effect is used to fake a bumpy surface of objects, by using the normals of the surface. In our scene we used it for the grain of the wooden objects, the surface of the wallpaper and the tire profile.The normal maps, generated by blender, aren't perfect, but it is sufficient to have a nice looking effect. The second effect we implemented was the environment mapping (cf. Figure 1 (b)). This effect displays reflections on specular surfaces. Since there are different techniques to implement the environment mapping, we chose the cube mapping. A cubemap consists of six textures, one for every direction, to represent the environment. We used it in our scene to display the reflections of the car windows.

*christian.bormann@rwth-aachen.de
†daniel.peters1@rwth-aachen.de
‡dominik.studer@rwth-aachen.de
§joern-michael.miehe@rwth-aachen.de
¶michael.anhuth@rwth-aachen.de

Finally we wanted to have a trace of tires effect as a typical racing game effect. We started trying to render the track into the texture under the car, e.g. the floor, but in order to some texture issues, we changed the implementation to a particle effect. Implementing a particle effect is easier, because of the independence from the underground, but also more ressource hungry. Unfortunately the effect still doesn't work allright. Probably there are unsolved errors with the management of array buffers and vertex buffer objects.

2 Physics

The gameplay relies heavily on the Bullet physics engine and especially its vehicle simulation. We decided to use a predefined model for vehicles, the bulletRayCastVehicle. In this model instead of simulating the car consisting of a chassis and several wheels, we use rays and check for intersection below the car to simulate car-like behaviour. This way all the basic forces that are known in physics can be considered but the simulation remains simple enough to allow modification of the vehicle behaviour quite easily (cf. Figure 2 (b)).
One of the biggest problems was to determine the road behaviour of the vehicle - it didn't feel right to use an arcade-like vehicle but it should not be to punishing as well. Thus we had to play around with the friction of the wheels and especially the increments of the steering quite a bit to achieve a good road behaviour that forces you to use the breaks or at least stop accelerating to not lose control in some situations.
In general the physics world is created independently from the .obj files that are used for the graphics to avoid unneccessary complexity as most of the models can be approximated by very few primitives known to bullet. Thus, when importing objects to the game a physical representation in form of bullet primitives has to be given.



(a) (b)

Figure 2: *physical representation.*

3 Gameplay

The game offers 2 modes, a free driving mode in which you can freely explore the track and a racing mode in which you have to complete a specified number of laps. To offer the player some feedback regarding his performance the current lap time as well as overall time and number of laps are presented in a GUI.
For each lap the player has to pass through a number of checkpoints to avoid cheating but also grant the ability to reset the car to the last checkpoint in case of a crash (cf. Figure 2 (a)).
A Trigger-system to control certain game events and especially control said checkpoints is part of the engine.
We decided to use pair tests between the car and every trigger instead of checking for all collisions with a trigger for better performance as a collision between triggers is not possible for our game.

4 Scene editing and loading

Editing and loading scenes for display consists of three components:
The Storage Container layer, the ACGL Decorator layer and the QtGUI Decorator layer.
The underlying container layer only serves as data storage for instances to be put in scene. For every possible type of object (e.g. materials or physical representations) there is a class using basic Qt features to hold its full information. This layer uses QtDOM to load/store the contents of the containers from/to a set of XML files. Using this Storage Container layer, the QtGUI Decorator layer [QtFoundation 2011] can create corresponding displayable forms for every item. To keep the GUI simple, MDI subwindowing techniques are used, and for consistency reasons, Qt's signal/slot-system is an intuitive way to go.
Likewise relying on the Storage Containers, the ACGL/3D Decorator contains factory mechanisms for creation of the corresponding ACGL/Bullet objects to be directly weaved into a fully featured Scenegraph.
To avoid memory leaks and to speed up Scenegraph rendering, each layer relies on a Factory/Singleton combination for object creation and access:

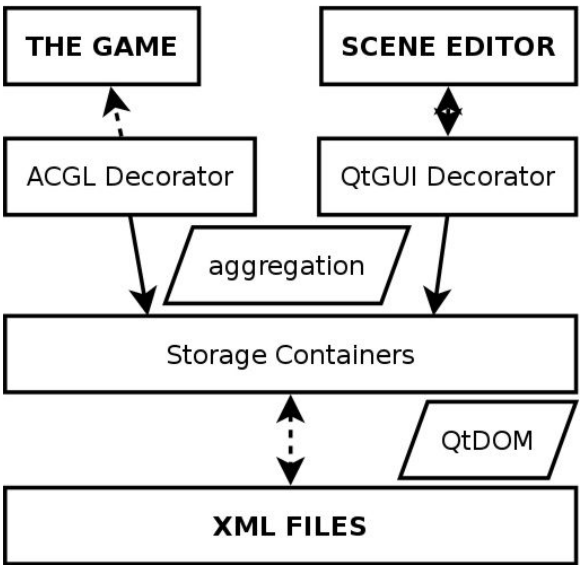


Figure 3: *UML diagram.*

5 Conclusion

When designing our game at the start of the practical course, we decided to choose the theme of a children's room for our mini racing game. We then thought about how to create the right feeling for the player. During the process of development we strongly recognized, that planning, prioritizing and documenting our work makes the development process much easier. As none of our group members had any experiences with graphics programming, most problems occured in this part of our work. But we were still able to get the features done, that support the atmosphere of the game the most. Here communication between our group members was important, so that we could help out each other.

References

ENGEL, C., 2010. Blender open material repository. Website. http://matrep.parastudios.de/.

QTFOUNDATION, 2011. Qt4.8 documentation. Website. http://qt-project.org/doc/qt-4.8.