Developing an Arcade Game

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

Lisa Mannel[*] Thomas Prinz[†] Alexander Schmidt[‡]

Figure 1: *Logo of our Game Arcade Racer*

Abstract

In our game ”Arcade Racer” the player controls a car driving through a natural environment. The aim is to reach a sequence of checkpoints as fast as possible to create a new highscore. The best highscore is represented by a ghost car, which mimics the behavior of the player who set up that score. There are several obstacles such as trees or mountains, as well as different ground types, that change the cars behavior or even destroy it. To keep the game interesting, new randomized maps can be created. (cf. Figure 1).

Keywords: game programming, arcade, game, racing

1 Gameplay

1.1 Map Randomization

The map consist of a plane with several objects placed above. It is surrounded by high mountains the car cannot pass through. There are different types of ground, each of which is rendered on its own plane object. This planes are then blended together according to information saved in a jpg-file, leading to the image of one plane with different ground types. Thus the distribution of the ground types on the map can be changed easily by changing the jpg-files with an image editor.

The checkpoints as well as the objects are placed randomly above the plane whenever a new map is created. They cannot be placed onto lava and some objects do not spawn on water either. To make sure each checkpoint can be reached, a randomized path is created between them, on which no object can be placed. Similar objects, like trees and trunks for example, form forests to create a more realistic world. The information about the map is then saved into a txt-file, such that exactly the same map can be reloaded. Only one map can be saved at the same time.

Some of the objects are self-made, either because no suitable model could be found in the Internet, or because the models found were to complicated and led to a bad performance. Some examples are the car, the wheels, the trees or the mountains. Most models found in the internet have been modified due to performance issues. (Which objects and/or textures exactly were created by us and which not is documented in the sources.txt file)

[*]lisa.mannel@rwth-aachen.de
[†]thomas.prinz@rwth-aachen.de
[‡]alexander.schmidt1@rwth-aachen.de

1.2 Main menu

In the main menu the player can choose whether to create a new map or whether to keep playing the old one. The players names can be entered and the highscores are displayed. A new game can be started or the program can be exited. The main menu is texture based except for dynamic information (e.g. highscore), which is realized via 2D text rendering.

1.3 Highscore and Ghost Car

If a player reaches a new highscore, the information about the cars behavior is saved in a txt-file. The next time the map is played this information is used to create a "ghost car" that acts exactly as the player did who set up the best highscore. Whenever a new map is created, this information is deleted.

2 Physics

2.1 Driving Physics

The physic quite realistically simulates different gear levels with acceleration and maximum velocity depending on current gear level. It takes different ground types into consideration. Steering is approximated because of its complexity. [Monster]

2.2 Collision Handling

The collision handling is realized by drawing a 2D bounding box around every object. A very simple bounding circle check discards any two objects that are too far to collide. Then the separating axis theorem is used to determine if two objects collide. This was refined to allow multiple bounding boxes per object to allow the car to drive below a bridge. As a result the car loses speed proportional to how perpendicular it collides with an object.

3 Graphics

3.1 Lighting & Shadows

3.1.1 Dynamic Lights

Our lightsources are dynamic and can move arbitrarily through the level creating dynamic shadows.

3.1.2 Shadow Mapping

We faced some problems creating visually attractive shadows for a map that is that huge. As a solution we implemented Cascaded Shadow Maps, which provide a higher effective resolution in the area near the camera and less resolution further away. To create smooth shadows we implemented Variance Shadow Mapping. This introduced 'Light Bleeding' as another problem. We tried to solve this problem as good as possible by saving how close the pixel is to the light and applying a reduction function with different strengths. We also added a dynamic day/night cycle with a linearly approximated sun which makes our sunlight shadows dynamic shadows. The Variance Shadow Mapping is based on chapter 8 of the GPU Gems series. [Nguyen 2007]

3.1.3 Phong Lighting

Phong Lighting is used for the sunlight. The headlights/backlights use a different approach in order to guarantee good visuals at night.

3.2 Motion Blur

We implemented Motion Blur as a post processing effect. While rendering we determine the screen space velocity of each pixel by comparing the last frames position with the current frames position. In a post processing step we blur the image along this velocity vector. To exclude certain object from blur entirely we weight the blurring with a factor proportional to how equal the two velocities are. The motion blur is loosely based on chapter 27 of the GPU Gems series except that we precompute the velocity instead of recovering it via z-buffer. [Nguyen 2007]

3.3 Particle System

We created a particle system that is used for weather effects like rain and snow as well as explosion effects and checkpoints. The particle system is capable of rendering animated particles by simply cycling through appropriate texture coordinates. The particles are rendered as points which are extended to 2D quads and billboarded using a Geometry Shader. This limits the data that is sent to the Vertex Shader to 4 floats (3 for position and 1 for lifetime of the particle). The particle physic is CPU-based.

3.4 Explosions

Using the particle system we added explosions as another particle type and graphical effect. Explosions occur as a death animation when you drive over lava.

3.5 Ground Normalmapping

To make the planar ground look more realistic and interesting, normal maps are used.

3.6 Skydome & Dynamic weather

A skydome has been implemented to display the skylike background. We also added a weather class which dynamically but randomly changes the current weather. Depending on daytime and weather the skydome fades between different textures. Different weather types are: Clear, Cloudy, Rainy and Snowy.

3.7 2D Text Rendering

For highscore and general user output (e.g. 'Game Over') we implemented a way of doing simple 2D text rendering. For text rendering we blend simple quads over the final frame. To save draw calls text rendering uses instancing to draw multiple letters at once. A pre-antialiased, dynamic width texture font is used.

3.8 MiniMap & Arrow

We added a perspective minimap which gives the player an overview of its position. This is simply realized by adding a camera that flies over the car and renders the scene. An arrow guides the player to the next checkpoint.

References

MONSTER, M. Car physics for games. `http://www.asawicki.info/Mirror/Car%20Physics%20for%20Games/Car%20Physics%20for%20Games.html`.

NGUYEN, H. 2007. *Gpu Gems 3*, first ed. Addison-Wesley Professional.