Developing a Jump'n'Fly Game

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

Jonas Nagy-Kuhlen[*] Lukas Prediger[†] Adam Malatenski[‡]

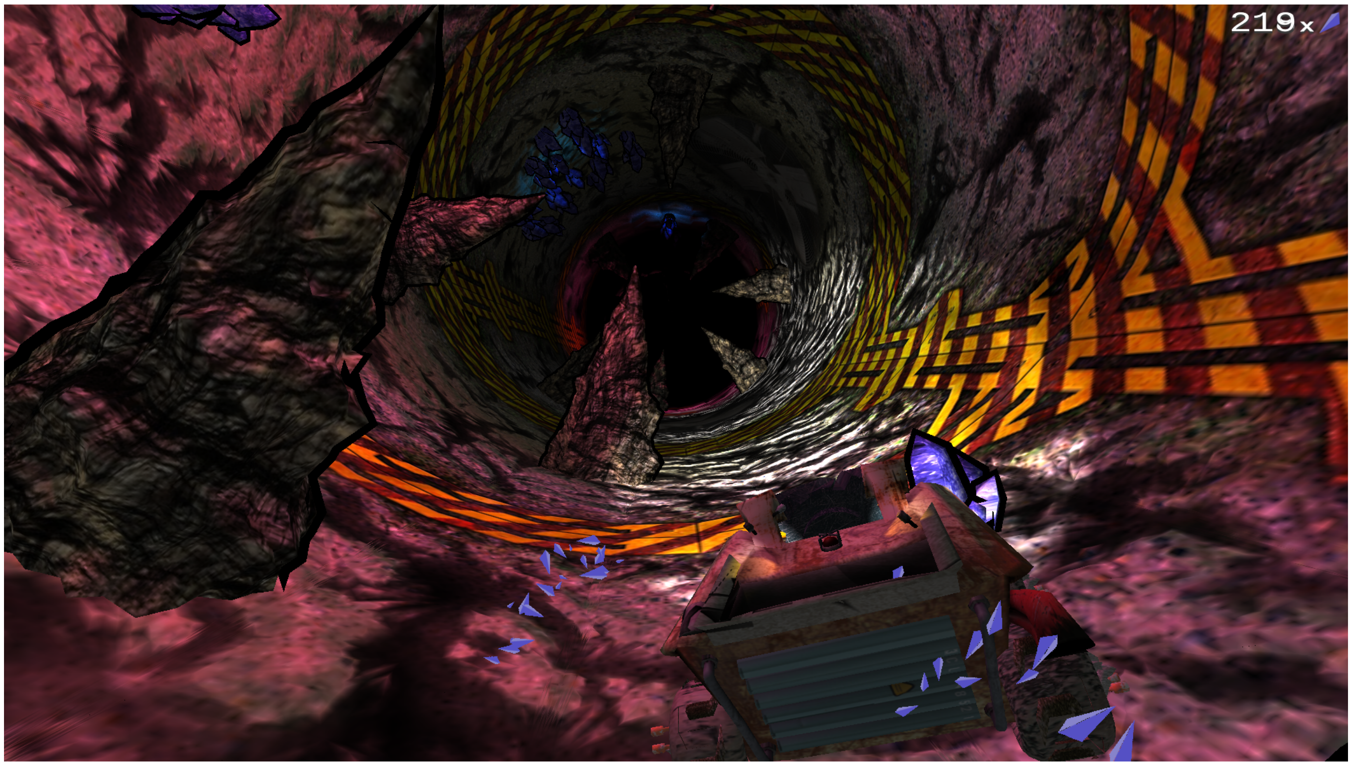

Figure 1: *A typical situation in the game. The camera is positioned behind the vehicle controlled by the player.*

Abstract

The objective of this pro-practical was to develop a jump'n'fly game in which the player controls a vehicle that follows a given track and tries to avoid several obstacles. We freely adapted this idea and abandoned the jump as well as the fly part.

The player controls a mining vehicle in a cylindrical tunnel. The vehicle is equipped with a drill to collect crystals while avoiding stalagmites which have crushed through the tunnel's walls. Due to special grip of the wheels, the vehicle can travel freely on the tunnel's surface but is not able to lose contact to the ground.

We decided to implement a comic-like look achieved by drawing black outlines around objects. Furthermore, we emphasized the cold and dark tunnel atmosphere by use of light and darkness. That is the reason why we applied dim ambient light and bright light spots on the vehicle as well as some local point lights along the track.

Keywords: game programming, jump'n'fly game

[*]jonas.nagy-kuhlen@rwth-aachen.de
[†]lukas.prediger@rwth-aachen.de
[‡]adam.malatenski@rwth-aachen.de

1 Gameplay

The goal of the game is to collect as many crystals as possible on a track of fixed length. When the vehicle collects a crystal by simply driving through it the player is rewarded with points (displayed in the upper right corner, cf. Figure 1). Collision with stalagmites results in a loss of points because some crystals fall out of the vehicle's crystal container. The current highscore is saved locally.

The vehicle accelerates automatically to its maximum speed. Steering behaviour is controlled by the player. Collision with stalagmites causes the vehicle to lose speed and bounce off the obstacle. If a collision with a stalagmite seems to be unavoidable, the player can activate the vehicle's drill which enables it to destroy any obstacle without losing points until the drill needs to cool down.

2 Graphics

We decided to use single pass rendering for lighting effects because a limited number of lights in the scene seemed to be sufficient for our purpose. For light calculations we implemented simple Phong shading. Despite the fact that atmospheric light effects had a high priority in our group, we went without dynamic shadows because the dominant light sources, the spot lights attached to the vehicle, were always aligned with the view direction and thus shadows would be occluded by the objects casting them anyway.

Another high priority was the presentation of the rough tunnel surface. Normal mapping was ineligible because of the flat angle between surface and camera direction. Hardware tessellation was a suitable alternative. In addition to the diffuse texture of the tunnel we created a height map which provides a height value for each point orthogonal to the surface. The tessellation shader makes use of this data to subdivide the tunnel mesh and raises or lowers the vertices according to the height map. Thus we were able to represent every detail of the rocky structure of the tunnel as geometry. Out of performance considerations, we introduced dynamic level of detail. Faces close to the camera are subdivided many times, whereas faces in the background are not subdivided at all.

To display the outlines of objects we considered edge detection in a post-processing pass. Anyway, considering performance, we chose a different approach. Objects with outlines are drawn twice. The second draw call inflates the object by moving all vertices along their normals by a constant value. This mesh is then drawn with reversed faces in black colour, resulting in a nice outline around the object. This approach works well for simple, approximately convex geometry. However, it is not applicable for complex geometry, especially if there are multiple vertices with different normals at the same location. This is the reason why the vehicle has currently no outline.

To further enhance visual experience, a particle system has been implemented. A lot of particles can be drawn at low costs because each particle is represented by a billboard constructed in the geometry shader, reducing the amount of data transferred to the GPU.

We designed the rendering of text in a similiar manner. The system is capable of drawing arbitrary strings at given positions in screen space using sprite fonts. Thus we were able to easily layout a menu and display the current score in the game.

3 Software Components

We put a lot of effort into designing a flexible, easy to use and extensible object-oriented architecture as the basis of the game. The goal was to provide components which would make the implementation of specific game logic easy and straightforward. Game logic should be focussed on controls and mechanics and not bother with rendering, collisions and other low-level details. We believe that we succeeded in reaching this goal. In the following the two most important conceptual components of the game are presented.

3.1 Scene System

The scene system forms the base of the game. It allows the creation of scene nodes which can be placed in a 3D scene and define a transformation. Scene nodes are hierarchically structured by parent-child relationships, where nodes define a reference frame for their descendants. Every high-level object like meshes, lights, particle emitters, cameras and collision nodes is a scene node. The hierarchy can be traversed by distinct visitors for different purposes, for example rendering of meshes or collision detection and resolution. Altogether, the scene system takes care of all 3D representation and rendering as well as collision resolution. It is completely independent of specific game applications and may be used for other games as well.

3.2 Level System

The tunnel is made out of level segments of fixed length which are placed in sequence. Thus an impression of an infinite tunnel is achieved. Each segment has a well-defined structure and specifies the positions of obstacles, crystals and lights as well as tunnel geometry. A random sequence of segments forms the tunnel. Management of the segments is the task of the level system. It loads segments from XML files and builds their representation in the scene. Due to performance considerations, only two segments are visible and active at the same time. When the vehicle reaches the end of a segment, this segment is discarded and a new segment is attached behind the remaing one.

4 Assets and Tools

The level segment XML files provide the opportunity to define complex objects consisting of meshes, textures, shaders and transformations. This makes easy level creation and extension of game content possible without need to rebuild the game. XML is easy to read and understand, which further simplifies the process. Unfortunately, the game currently offers only limited content because of limited resources. The major drawback of editing the XML files manually is that one has no visual feedback, especially concerning object transformation. That is the reason why we developed a level editor which allows to place objects on the unrolled tunnel surface and export the result as a XML level segment file.

5 Conclusion

We believe that at all we did a good job in developing a playable and enjoyable casual game which offers a solid gameplay experience. During the course we gained experience in computer graphics and were able to design and implement a complex software system. It was a good decision to put effort into a clean architecture as a base for the game. Developing of the game logic, even at the end of the development process, did not raise any difficulties. Anyway, some decisions were not optimal. Chosing the visitor pattern for data structure traversal made it difficult to extend the object hierarchy and possibly reduced performance. Furthermore, relying on a single pass lighting system caused performance issues and made light distribution limited and inflexible. Deferred rendering would have been a better choice for lighting purposes. We would have liked to provide more content and visual effects for the game, but unfortunately this was not possible to achieve with merely two active developers.

6 Distribution of Work

• Scene System:	Jonas Nagy-Kuhlen, Lukas Prediger
• Level System:	Lukas Prediger
• Physics and Game Logic:	Jonas Nagy-Kuhlen
• Tessellation:	Jonas Nagy-Kuhlen, Lukas Prediger
• Lighting:	Jonas Nagy-Kuhlen
• Particles:	Jonas Nagy-Kuhlen
• Font and Image Rendering:	Lukas Prediger
• Menu and Highscore:	Lukas Prediger
• Sprite Graphics:	Jonas Nagy-Kuhlen
• Models:	Jonas Nagy-Kuhlen, Adam Malatenski
• Video and Report:	Jonas Nagy-Kuhlen, Lukas Prediger