Developing a Narrative Game

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

Deniz Bicer[*] Raheel Yawar[†] Ali Arslan[‡]

Abstract

The purposed of the project was to create a narrative based walk-though simulator virtual reality game. The game used the Dive VR gear with an iPhone as a platform. The control input was through a standard bluetooth controller. The game progressed by player interaction with specific objects in the virtual environment. Audio narrations provided the player with clues on how to proceed. Graphical effects such as shadows, lights, bloom and anti-aliasing were implemented. They were chosen according to the needs of the interactive story in the virtual environment.

Keywords: game programming, narrative game

1 Content Creation

Blender was used as for the three dimensional modeling. Low-polygon meshes were used to optimize the rendering because the target platform was a mobile device. We wrote a custom obj format exporter for blender to support custom written object loader in the engine. Photoshop was used to texture the models. Voice actors were employed for the narrative dialogues. The dialogues were recorded and pieced together by an audio recording software.

2 Graphics

2.1 Lighting

Phong shading is used for lighting where light positions are defined in geometry and our object loader finds and put them in a collection containing lights. Due to split screen view, matrices for left and right eye are calculated separately and loaded to shaders as uniform for further Phong lighting calculations. In fragment shader (shader.fsh) diffuse, specular and ambient contributions are calculated, where texture color is used in the calculation of diffuse term.

2.2 Shadows

Shadow mapping is used for creating dynamic shadows. In first pass we attach shadow texture as GL-DEPTH-ATTACHMENT using frame buffer object. Instead of view matrix for eye this time we send light direction, hence resulting in a shadow texture map. In second pass we have to compare depth stored in texture map recorded in last pass and depth in this pass. To calculate depth in second pass we pass lightModelViewProjection matrix as uniform and multiply it with position to calculate point in light space. This point is sent to vertex shader, here as it is not being perspective transformed, we divide it with w component and convert it to NDC coordinated. If this depth (z component) is greater than depth from shadow texture then we multiply the light with a predefined shadow factor.

[*]deniz.bicer@rwth-aachen.de
[†]raheel.yawar@rwth-aachen.de
[‡]ali.arslan@rwth-aachen.de

2.3 Blur

Gaussian blur effect has been used where we first save the scene in fbo then pass its captured fbo texture to vertical Gaussian filter which is written in fragment shader (blur.fsh). The resulting fbo captured texture is passed through horizontal Gaussian filter.

2.4 Bloom/Glow

To achieve Bloom effect we specific locations by the help of texture alpha channel. We adjust alpha of glow regions to a particular value. For bloom effect, at first we apply Blur effect by the method explained in last section. The resulting texture attachment of fbo is being blended with normal scene. The blending is done in another extra pass in separate fbo. Another shader (blend.fsh) was written to achieve desired effect.

2.5 Antialiasing

We are using Fast Approximate Anti-Aliasing (FXAA) technique for which we have added functions in our fragment shader (post.fsh). This way all the edges, which result from alpha blended textures and pixel shaders are being smoothed.

3 Game Logic

In the part of game logic, features which make our game functional and intractable, are implemented. Overall what has been done are; loading objects that are exported from blender, sorting the objects out to the related data structures. In addition the interaction of the game via a game controller and the head movements of the player are implemented. After that the sequencing and narration of the game is included so that the player can be guided through a story line.

3.1 Object Loader

Our object loader accepts one object file which has all the objects of the game. The objects are sorted out according to a name convention. They all have their object type and name, where the type says what that object is and the name says in which room that object is. According to that we categorize the objects according to rooms. This helps us during collision detection and rendering the objects for different texture atlases. The types are important for intractable objects, where we need to instantiate a special object like doors and triggers, which are extended from object class.

3.2 Collision Detection

We needed collision detection for a feel of a real simulation of an environment where you cannot go through walls or objects. We used Axis Aligned Bounding Box (AABB). This part took a lot of time because of confusions at the coordinate systems. First the view matrixes were used to find out where the player is, but it was hard to keep up since the position of player and objects were changing all the time. After failure of this approach, we trace our own player position at a variable. Next step was to consider the room transitions. We only take account the objects which are at the current room, so we need to change the object set every time we switch to a new room. Here we use the categorized objects which come from the object loader. Door frames are extended to be used as an in between room. When the player leaves the door frame, we check which room they are in currently and according to that object set is switched to the current room.

3.3 Audio

We first used OpenAL to load the audios and play them when they are needed. It was convenient since OpenAL is similar to OpenGL, so learning the logic was easier. Also it provides lots of features to enhance the players sound experience; like the position of the sound, stereo audios, etc. We had to stop using OpenAL because it only works with the audio files which are in .caf format, and this format stores the audio in a 10 times bigger memory space than mp3. Therefore we transited to AVAudioPlayer which can work with mp3 files. Every level beginning, the related audios are loaded and then when their time comes they are played. When the level ends the audios are deleted from the memory.

3.4 Level Sequencing

Next step was obtaining the sequence of the levels, and learn what to do one after another. We used a XML file to have a flexible structure and avoid a hard coded level structure. We parse the XML file and store the sequence of each level at an array. Then the arrays for the levels are collected to have an array of levels. This data structure than used by level controller, to bind the objects and sequences with the correct levels.

3.5 Game Interaction

Interaction of the game is done by a game controller and the movements of the players head. Game controller works as a hardware keyboard and sends the button pressing and releases as a unique character for each button. At the program we handled the events and stored the state of the buttons to simulate continues movement. When the head rotation is implemented, we had problems with the modelview matrix, because the position of the player was always set wrong after the multiplications. This is solved by having the offset of the initial position of the player all the time and translating the player according to this offset.

References