Dive into Mobile VR/AR

In the following document we explain the challenges we faced for the development of our iPhone VR game "Boo Hunters"

Game Concept

The concept of the game is to catch invisible ghosts using a scanner. The scanner slowly makes a detected ghost visible. When the ghost is fully visible, it is catched.
In the following a few techniques we used to achieve some of the game functions are presented.

Importing the Blender Model into the game

Importing the Blender Model into the game was challenging for different reasons:

First of all, the number of vertices in the model was way too large for the game to run smoothly. Decimating the model in Blender and exporting it again solved this problem. We had to find the balance between a manageable number of vertices and a visually appealing look of the model, since taking too many vertices would result in the destruction of many furniture structures.

For importing the model into Xcode we exported different .obj files from Blender and used a simple Parser that we programmed ourselves which turned these files into .h and .c files that contained the data of the vertices positions, the texels, the normals and the materials, all necessary information to work with the model in OpenGL.

Ghost movement

Knowing how to implement the movement of the ghosts in the game was also challenging. As the ghosts have a predefined path that they repeat over and over again we decided the best way to implement the movement was to define for every ghost an offset as a global variable and a sign (+1 or -1) and to create a function called by a timer every 0.1 seconds. This function should move the vertices of the ghost in the sign direction till the offset is reached. When this happens, the sign is multiplied by -1, so that the direction the function is going to move the vertices gets inverted. This way the ghosts move in a defined path infinitely.
To make it seem like the ghosts were floating we also implemented up and down movement. This was done by adding to the z-Position of each vertice of each ghost a sine function over the current absolute time of the system. This way the sine function runs between 1 and -1 and an up and down movement is achieved.

Collision Detection

For detecting the collision of the Player with the environment we went with a two-dimensional approach: We take the 3D-coordinates of our player (x,y,z-data). Only two of these are relevant for our form of collision detection, the x and the z value. Both these values can be used to specify the location of our player from a top-down-perspective. This is important when the next piece of our collision detection comes in play: the collision map.
The collision map is a two-dimensional picture that was created using Blender to capture the top-down-view on our map. The resulting picture was then edited with different colors. Red represents the area that the player can’t pass through, whereas green color represent the area in which the player is free to walk.
The game then takes the player’s 2D-coordinates (the x and the z value) and transforms them, so that they represent the player’s position on the collision map. This has to be done since the in-game map and the collision map aren’t the same size. Now that we have the 2D-coordinates of the player on the collision-map, the actual collision detection can take place.
When the player tries to move now, the game checks the color of the pixels around the player’s position on the collision map. When the color that’s returned is red, the movement is rejected. If the returned color is green, the movement is valid.

Scanner Ghost Detection

The Scanner Detection was the next challenging part. We wanted the scanner to be accurate, but there should be a little extra margin to the area the scanner is scanning, so detecting the ghosts doesn’t take perfect precision from the player.
As with the collision detection, we used a two-dimensional approach for this aswell. We take the 2D-coordinates of the player that we used in the collision detection (the x and z values) and add the 2D-coordinates of the ghosts to the equation. The final data that is needed for the scanner detection is the horizontal camera angle. That is because we use the tangent function to detect if the player is looking at an invisible ghost with his scanner: We calculate a right-angled triangle from the player’s 2D-position to the ghosts’ 2D-positions. Now if the opposite line divided by the adjacent line is equal to the tangent of the horizontal camera angle, the player is looking in the direction of a ghost. As said above, we added a little margin of error to the equation so the player doesn’t have to be too precise with his scanning.
When the scanner is on and the player is looking in a ghost’s direction, that ghost is marked as “currently scanning”. It starts to become visible, until it’s fully visible. At that point the ghost is captured.