GPU-based AI Vision

Research and Development
Jason S. Hardman & Subhir Rao

Description:

One of the main problems with AI-Vision is the high cost of scanning the AI's view for visible players. We know how powerful the GPU is at performing simple instructions, in parallel, at very high speeds, so we decided to use the GPU as a scanner to search the AI's view for visible opponents. The algorithm described below scans a texture (the AI's view) for visible opponents and then returns the target player's position to the CPU for the AI to use.

Algorithm:

Step 1: Render the scene from the AI's perspective to a 2D texture (Stage 1 shader). Everything in the scene is rendered with an RGBA value of 0.0f, except for the player opponents. The opponents are rendered with an RGB of 1.0f and the alpha channel is used to store the player's index (which we will need later). We first render all the opponents to the screen and then render all the visible objects in the scene, including the terrain, on top of this. This way, an opponent can be obscured by terrain or other objects.

Step 2: Depending on the size of the output texture from the previous step, create a vertex array or buffer. The output texture from the previous step and the vertex buffer need to have a one to one correlation, i.e., we have one vertex for each Texel. The reason for this is that the next step relies heavily on the vertex shader.

Step 3: Pass the above vertex buffer and the output texture from Step 1 to our Stage 2 shader. The Stage 2 shader, to put it very simply, is nothing more than a very very fast scanner. We are going to use the vertex shader to scan through the output texture. If our texture lookup gives us a value equal to 0.0f, we discard the vertex (set it's output position to -1.0f).

If we find any other value in our lookup, it means we have spotted someone. In this case, based on the player's index (which was stored in the alpha channel) we change the current vertices' position. We then retrieve the player's position by multiplying the texture coordinates with the inverse of the ViewProjection matrix from Stage 1. The player's position is stored in the RGB channel.

Step 4: Based on the number of opposing players, the output texture from the last step should be reasonably small (in our case, 4x4). All we need to do is scan the final output texture, look for non-zero values and store them in a list. Then we un-project these values and we've got a list of all visible opponents from the AI's perspective.