atomicubes

Objects created from cubes which appear when approached.

# 1 Abstract

Entities in a game world are typically created with a solid mesh that is always visible. One example of this is a bridge which consists of a singular hexahedral mesh stretched over a chasm along one of its axes. This need not be the only way to represent objects though. By creating the mesh but not rendering it, the object can still participate in physics while letting another object can stand in, visually. We will explore this concept by creating a bridge in Unity as described above but replacing the visible part of the mesh with an array of cubes which form platforms beneath objects as they are approached.

# 2 Methodology

To create the atomicube-bridge a false-bridge was first created. This false-bridge consisted of two parallel walls of the same length, width and height. Additionally, the line which traveled through the starting point of each wall was perpendicular with the walls themselves, in other words: the starting points and ending points were aligned. Two steps led to the top of these walls, allowing a player to reach the top of the bridge. Between the walls a hexahedron filled the space creating a platform which the player could walk across. Figure 1, below, details this setup.

|  |
| --- |
|  |
| **Figure 1**: The false-bridge |

With the false-bridge in place, the atomicube-bridge could easily be created from it. First the hexahedron filling the top of the bridge was made invisible by removing its MeshRenderer. After this two cubes were created and attached to the same transform, we will call this the cube-transform. One of the cubes then had its MeshRenderer removed. After this the cube-transform was scaled, deactivated and placed in the center of the hexahedron. Figure 2, below, shows what the setup looked like at this point.

|  |
| --- |
|  |
| **Figure 2**: The cube-bridge |

Now, all that remained was creating the cube lattice to fill the bridge and getting the cubes to fly in. Instead of creating more cubes in the Unity editor, a script was applied to the hexahedron which cloned the cube-transform and placed the clones in a lattice. This script will be described in the following section, Code. Figure 3, below, shows the setup with the cube-lattice.

|  |
| --- |
|  |
| **Figure 3**: The cube-lattice bridge |

At this point cube-lattice bridge was finished but the cubes had not yet been arranged to fly in. This was done with the use of another script which translated the rendered part of the cube-transforms according to parametric equations when the non-rendered parts entered a trigger. This script will be described in the following section, Code. After the script was applied the cube-lattice bridge was finished. Figure 4, below, shows the final result.

|  |
| --- |
|  |
| **Figure 4**: The complete cube-lattice bridge |

# 3 Code

The cube-lattice bridge can be implemented with a minimal amount of code. Indeed, some of the scripts used within our particular implementation of the bridge were included to simplify tasks which would be unpleasant to perform with the unity editor. What this means that is that our implementation could have required even less code. Nonetheless, these scripts, as well as any others which might be of interest to someone attempting to implement their own cube-lattice bridge, will be described here. The classes subsection, below, will provide a listing of the classes created and their descriptions. The computation subsection, which follows the classes subsection, will provide a description of how all of these classes interact at an architectural level.

## 3.1 Classes

## 3.2 Computation

# 4 Results

# 5 Additional Resources