Horizon Bending for Stylized Spherical Worlds

Christoph Kubisch, Per Normann Abrahamsen

Otto-von-Guericke Universität Magdeburg
3 Point Studios
Email: kubisch@isg.cs.uni-magdeburg.de
perna@3pointstudios.com

Abstract

We present a technique that creates the illusion of a spherical world, while the source scene description is from a flat world. The latter allows us to use standard techniques of planar terrain rendering and physics with a single directional gravity. This is less complex than having an actual spherical world, where different techniques and scene editors are used. Our deformation effect allows us to change the curvature at runtime, which can be used to adjust the appearance of the world or add dramatic effects. Another benefit using this technique is that it allows to limit view distance. In a planar world the view distance can be quite high, which may require level-of-detail techniques. Using a modified visibility test according to the world's curvature, we benefit from drawing less objects, while assuming most detail lies on the ground and the view direction is mostly parallel to ground.

1 Introduction

In virtual worlds for computer games, certain exaggerations of natural phenomena can help to create a distinct style. Varying the scale of proportions is one such a frequently used method. Recent titles such as EA Maxis' *Spore* and Nintendo's *Mario Galaxy* have successfully used spherical worlds in their designs. The majority of games, however, still use planar worlds. Such planar worlds can be rendered and animated using many standard techniques, whilst spherical worlds have a higher complexity, e.g [1] and [2]. That complexity exists in tools for creating the worlds, rendering them and physics simulation.

2 Our Approach

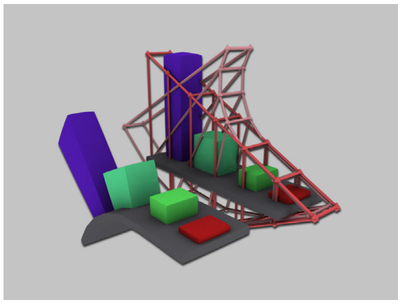

Figure 1: The deformed world geometry is shown left. Right objects illustrate the inversely deformed visibility frustum.

The deformation effect is similar to ([4]). However they focussed on making more terrain visible based on context, while we want to see less to get additional culling and stylization of world's curvature. We decided to implement this effect as on-the-fly deformation of all vertices depending on distance to camera (illustrated in figure 1 left) using a vertex program. At runtime the minimum and maximum range of the effect, as well as the angle and base elevation can be altered. The bending is length-preserving at the base level. Geometry below it should be avoided, as it could create distortions by self-overlaps. Although the top level geometry is also distorted, this is less obvious due to the geometry rotating away from the viewer. The effect of the deformation is visible in figure 2 top. The bending axis lies in the base level and is perpendicular to the direction (camera to vertex) on the base level.

Furthermore the curvature can help to reduce the

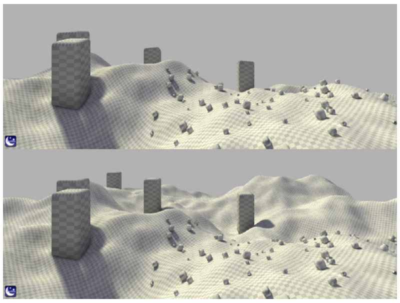

Figure 2: The effect applied to a test scene with 90 ° bending (top) and no bending (bottom).

amount of visible objects. To benefit from this we modified the visibility test system. The camera's frustum is split into multiple sections, which are inversely deformed as seen in figure 1 right. Applying this multi-frustum reduces the drawn objects. We assume that in a typical world scenery the majority of detail will be on the ground, and the sky will contain less objects.

2.1 Results

As the effect is vertex driven, the quality depends on the tessellation level of the scenes' geometry. On current generation hardware rendering a few more triangles of the same state settings (textures, shaders,...) does not have severe performance penalties ([3]). Therefore objects can benefit from higher resolution meshes to minimize vertex deformation artifacts. We tested our method on a `GeForce 9600 GT 512 MB, Core2Duo 2.3Ghz`. The scene consists of around $200,000$ triangles and $150,000$ vertices. To simulate common fragment-program load, a dummy program with 56 instructions was used.

We tested different configurations for their relative speed:

- **Off**: No bending is performed. This yields to the highest amount of visible fragments and the most simple vertex-program. 100%.
- **Conditional**: Bending is performed for vertices beyond the minimum range. The vertex-program has 47 additional instructions and makes use of conditional branching. 92%.
- **On**: Bending is performed for all vertices. The vertex-program has 53 additional instructions. Performs faster than conditional branching as the branching overhead outweighs the arithmetic costs. 98%.
- **Frustum**: As above but with improved frustum culling. Depending on scene can reduce processed meshes quite a lot. 170%

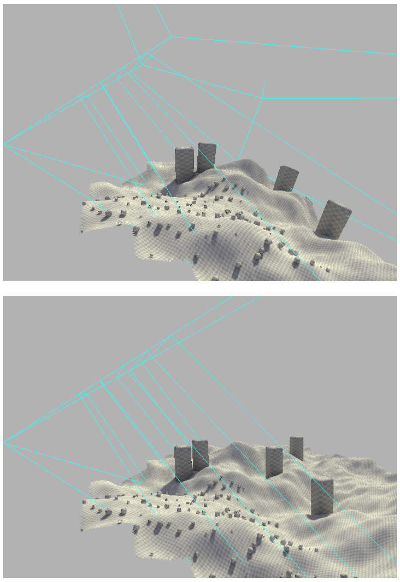

Figure 3: The scene of figure 2 is shown from a different angle and with the camera's cyan frustum hull.

References

[1] K. Compton and J. Grieve and E. Goldman and O. Quigley and C. Stratton and E. Todd and A. Willmott. Creating Spherical Worlds. In *ACM Siggraph Sketches*, 82, 2007.

[2] M. Klasen and H.C. Hege. Terrain Rendering using Spherical Clipmaps. In *Eurographics / IEEE VGTC Symposium on Visualization*, 91–98, 2006.

[3] M. Wloka. Batch, Batch, Batch: What does it really mean? In *Game Developers Conference*, 2003.

[4] S. Möser and P. Degener and R. Wahl and R. Klein. Context Aware Terrain Visualization for Wayfinding and Navigation In *Computer Graphics Forum 27/7*,1853–1860, 2008.