Realtime Per-Pixel Displacement Mapping on Arbitrary Geometry

Robin Lobel, December 2004

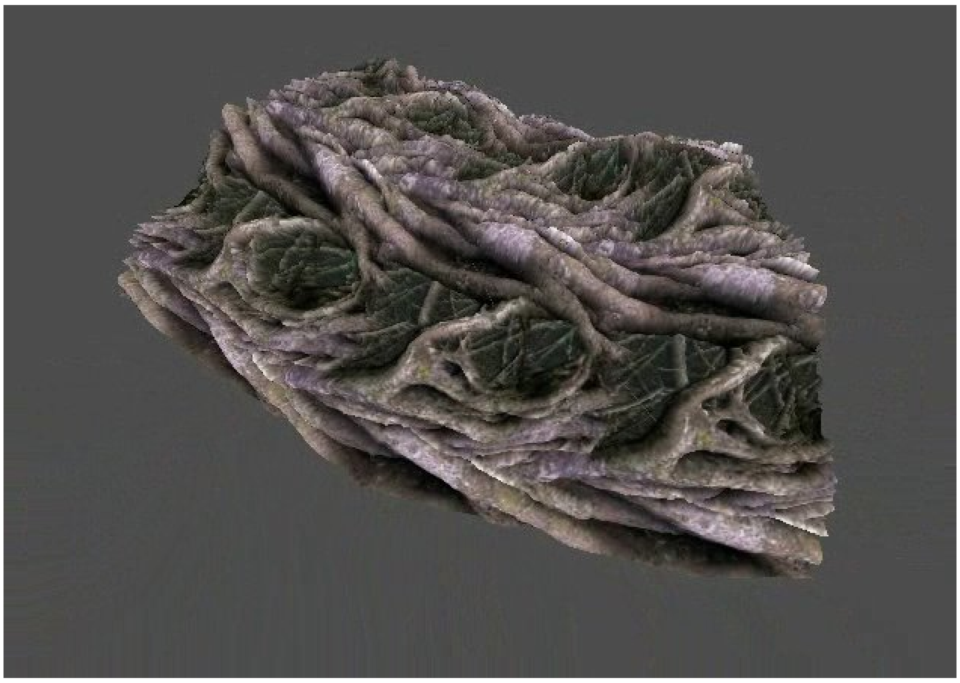

Fig 1. Realtime per-pixel displacement mapping applied on arbitrary model

Abstract

This paper present a method to achieve realtime per-pixel displacement mapping on arbitrary geometry, with object sides rendering.

1. Introduction and related works

Displacement Mapping is a way to add details on a polygonal surface.
Until 2000, it was mostly an offline technique, due to the large amount of computation needed and insufficient processing power [1].
The idea to use pixel shaders (as found in GPUs) to compute displacement first appeared in 2001 with Parallax Mapping [2].
Precise realtime per-pixel displacement mapping can also be achieved using the technic presented here.

2. The idea behind realtime per-pixel displacement mapping

Since GeForce FX are available, the possibilities for shader programming have greatly evolved.
More texture sampling and conditionnal sampling allows faster searchs for intersection through the displacement volume.
Because GPU are faster we can also consider real displacement volumes for each triangle, which means rendering using bottom triangle, top triangle, and 3 sides.

2.1 Using displacement volume

In this implementation we do not render top triangle, only bottom and sides (which mean 2 triangles per side=6 triangles for all sides), with per-pixel rejection where needed (ray leaving displacement volume with no intersection).
Each part of the displacement volume envelope is rendered with its own 3D displacement coordinates at each vertex.

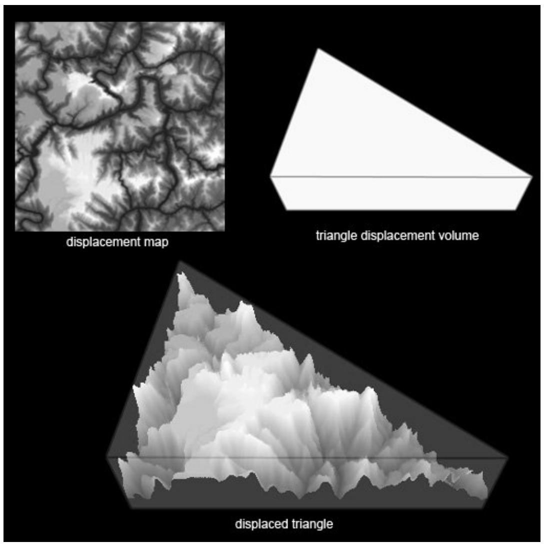

Fig 2. Displacement volume

Because displacement does not have to be uniform, displacement vectors can have different directions. The consequence is that sides are not necessarily flat.
In this implementation, sides are divided in 2 triangles each. To avoid geometry conflicts, it has to be decided how each side is divided for the whole geometry (sides are shared and rendered twice for 2 side-by-side displacement volumes).

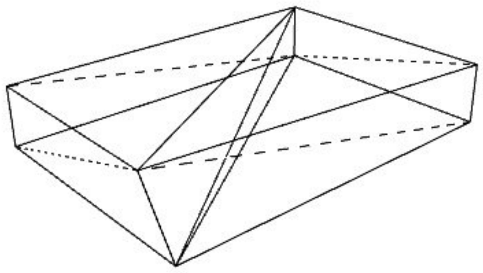
Fig 3. Non uniform displacement

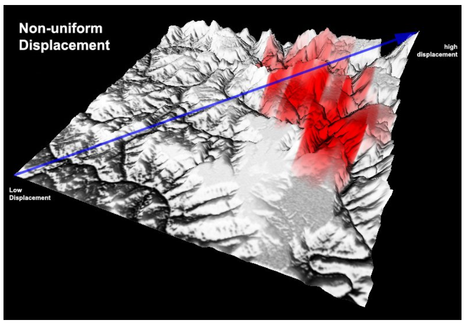

Fig 4. Another example of non unform displacement

2.2 Intersection search algorithm

We precompute each displacement map by transforming each texel in angles.
Theses angles represent an upside-down cone whose vertex sticks to the displacement, while the base is on the top of displacement volume. This cone represent the maximum empty space the ray can go through, with no intersection.

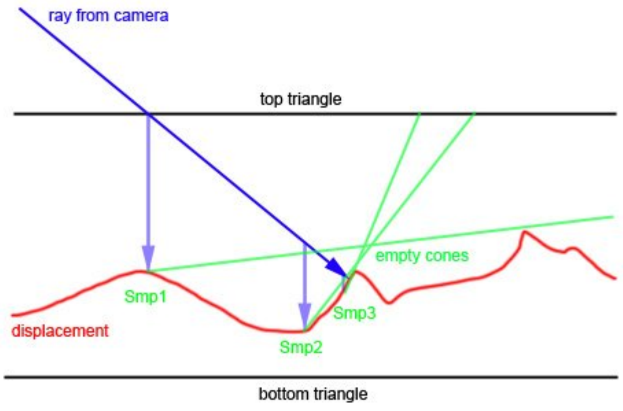

Fig 5. Intersection algorith

As shown if Fig 5., this algorithm allows pretty fast while accurate marching through displacement volume. 8 samples are enough to handle most cases.

To make the cone algorithm even more effective, we can use RGBA displacement map with A as the displacement, and RGB as 3 different cone angles, covering each 120 degrees.
This way we need even less samples to go through the displacement volume (6 instead of 8).

3. Results

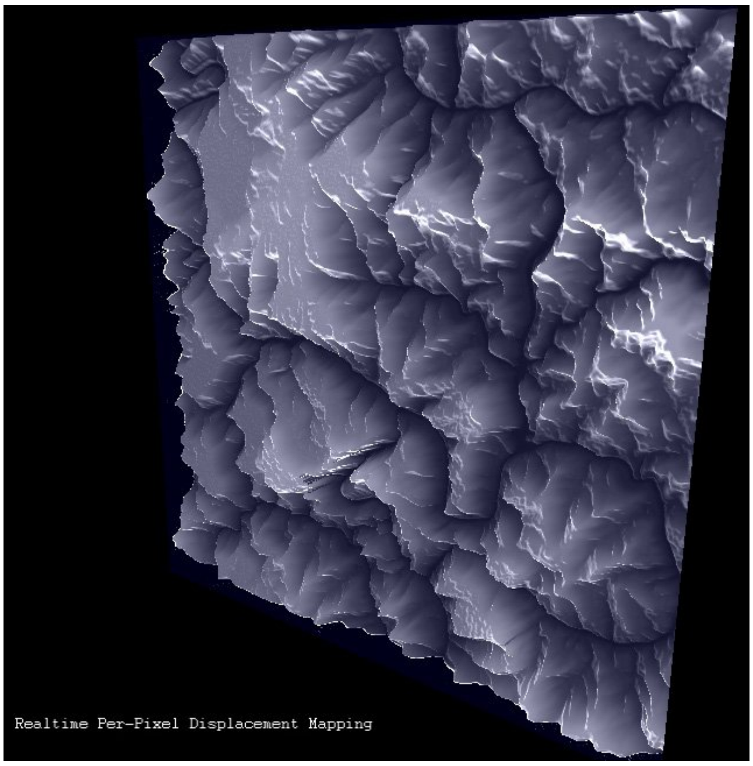

Fig 6. Displacement on a single quad

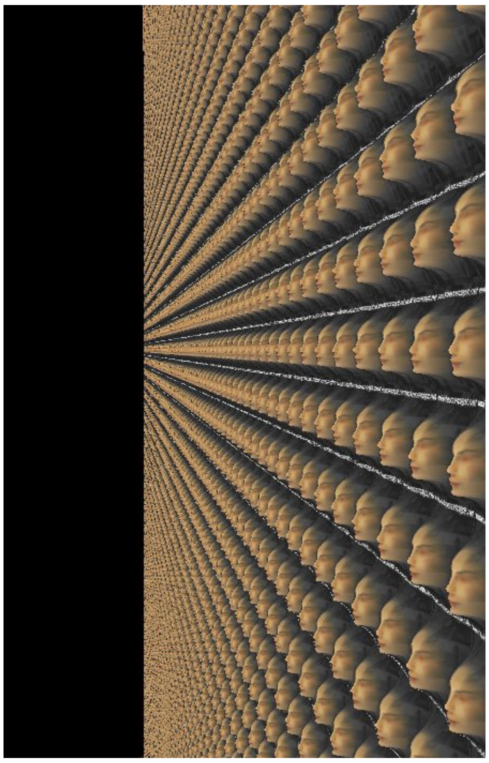

Fig 7. 100million triangles equivalent on a single displaced quad

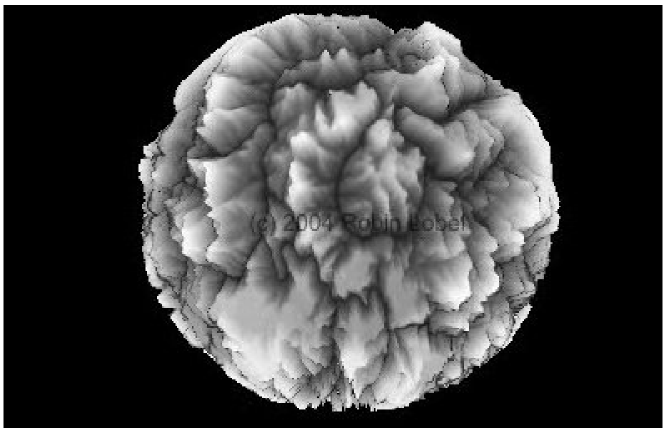

Fig 8. Displacement on a sphere

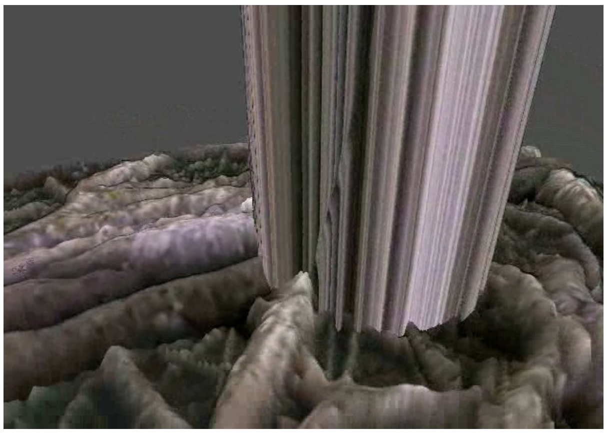

Fig 9. Displacement with Z-write support

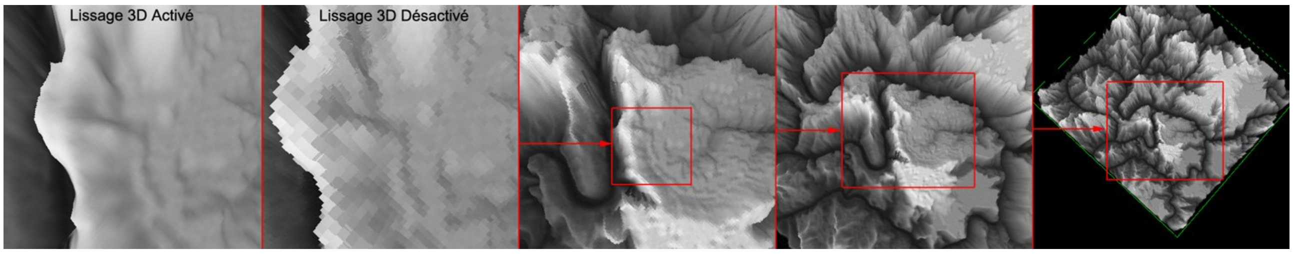

Fig 10. Cone algorithm supports bilinear filtering

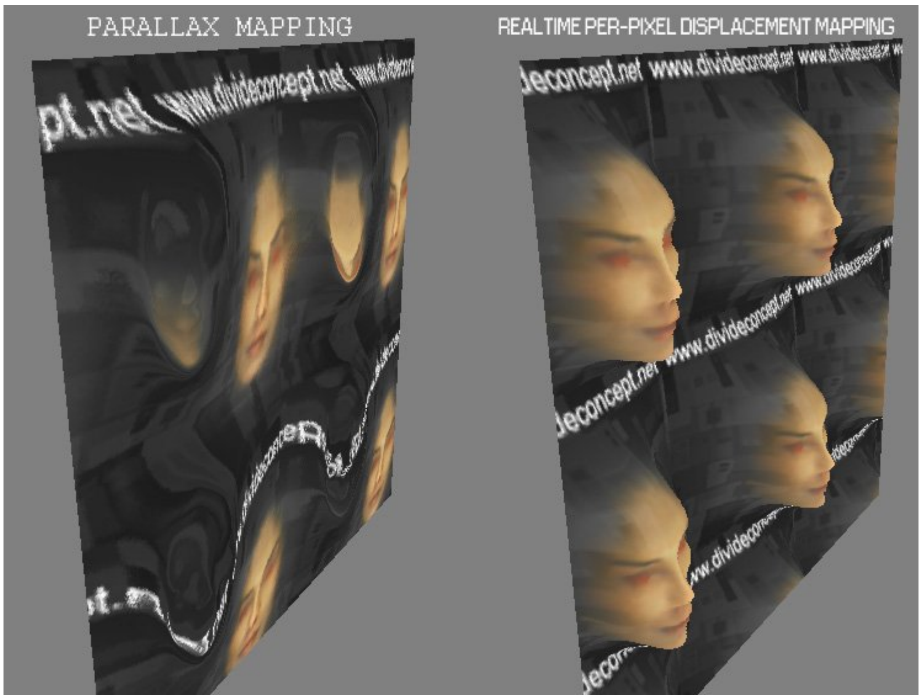

Fig 11. Parallax Mapping vs Realtime Per-Pixel Displacement Mapping on extreme case

4. Limitations and future development

If you don't sample enough, some artifacts may occurs, such as the border effect.

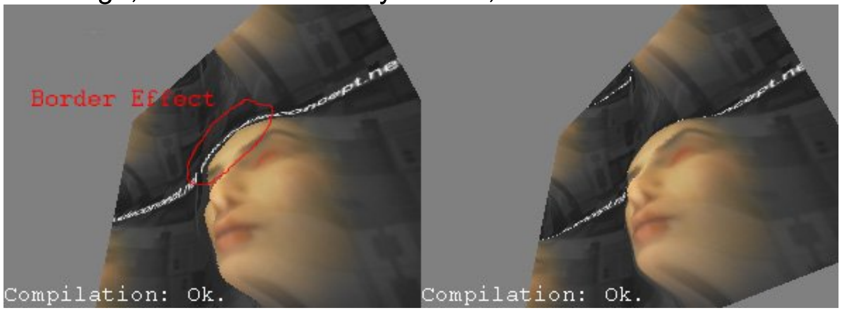

Fig 12. Not enough sampling creates artifacts

Conditionnal (indirect) sampling is slower than direct sampling on current hardware (GeForce FX). It will probably be faster on future generation.

References

[1] http://en.wikipedia.org/wiki/Displacement_mapping
[2] http://en.wikipedia.org/wiki/Parallax_mapping