6.1 Rendering Wet Sand

6.1.1 Goals

The primary goal of this project was to implement a method for rendering sand as a wet porous medium with water flowing into it from within PBRT. The final result was a video of sand being wetted by water particles flowing through it. The purpose of implementing this was to have a rendering technique available for rendering a multiphase flow of water water and sand.

6.1.2 Implementation

The rendering of wet sand was implemented in a pipeline as follows:

- A 2D fluid simulation which supported a mixed Darcy flow and free flow method was run to produce raw data of fluid being advected through two static blocks of sand.
- A python3 script which converted the raw data into data more readily parsable by pbrt, such as using particle positions to generate a density field and levelset.
- PBRT which rendered the fluid using a levelset rendering method to render fluid and a volumetric grid for rendering the sand blocks.
- Use exrtotiff to convert exr files to tiff.
- Use mencoder to stitch frames together into a video.

The primary reason for choosing an intermediate step between the fluid simulation and pbrt was to allow for rapid prototyping of parameters in the pbrt files generated without having to run the simulation multiple times and to extrapolate a 3D grid for the 2D grids generated by the 2D simulation. By having an intermediate frame description stored in the cluster it became trivial to express changes in the rendered scene between all frames of the simulation and parallelize the task of implementing those changes and rendering every frame through pbrt.

6.1.3 Fluid Simulation

The fluid simulation utilized a fairly naive 2D simulation of Darcy flow using particles as an extension of a 2D FLIP simulator for incompressible fluid. In the configuration used for the final render the sand is two static blocks of porous material that don't move throughout the simulation, which are represented by regions where the porosity is less than 1. The porosity $1 - \phi$ where ϕ is the volume of space taken by the porous material, so a porosity of 1 indicates that there is no porous material in a cell.

The fluid advects normally in regions when the porosity is precisely 1. When the fluid is in a porous. Per timestep the simulation outputted a file containing the particle radius, a list of particle positions and a grid of porosity values in a uniform grid. Since the porous material is static, the advection term in the Euler equations is zero, so advection in the solid blocks is ignored and the Darcy flux is simulated by adding a body force to the grid determined by the gradient of fluid density per cell.

Because the particle densities were not smeared across multiple grid cells very carefully, the changes in density after each timestep was discontinuous. The discontinuous changes in density leads to visible artifacts in the final renders but are not a definitive feature of the rendering method used.

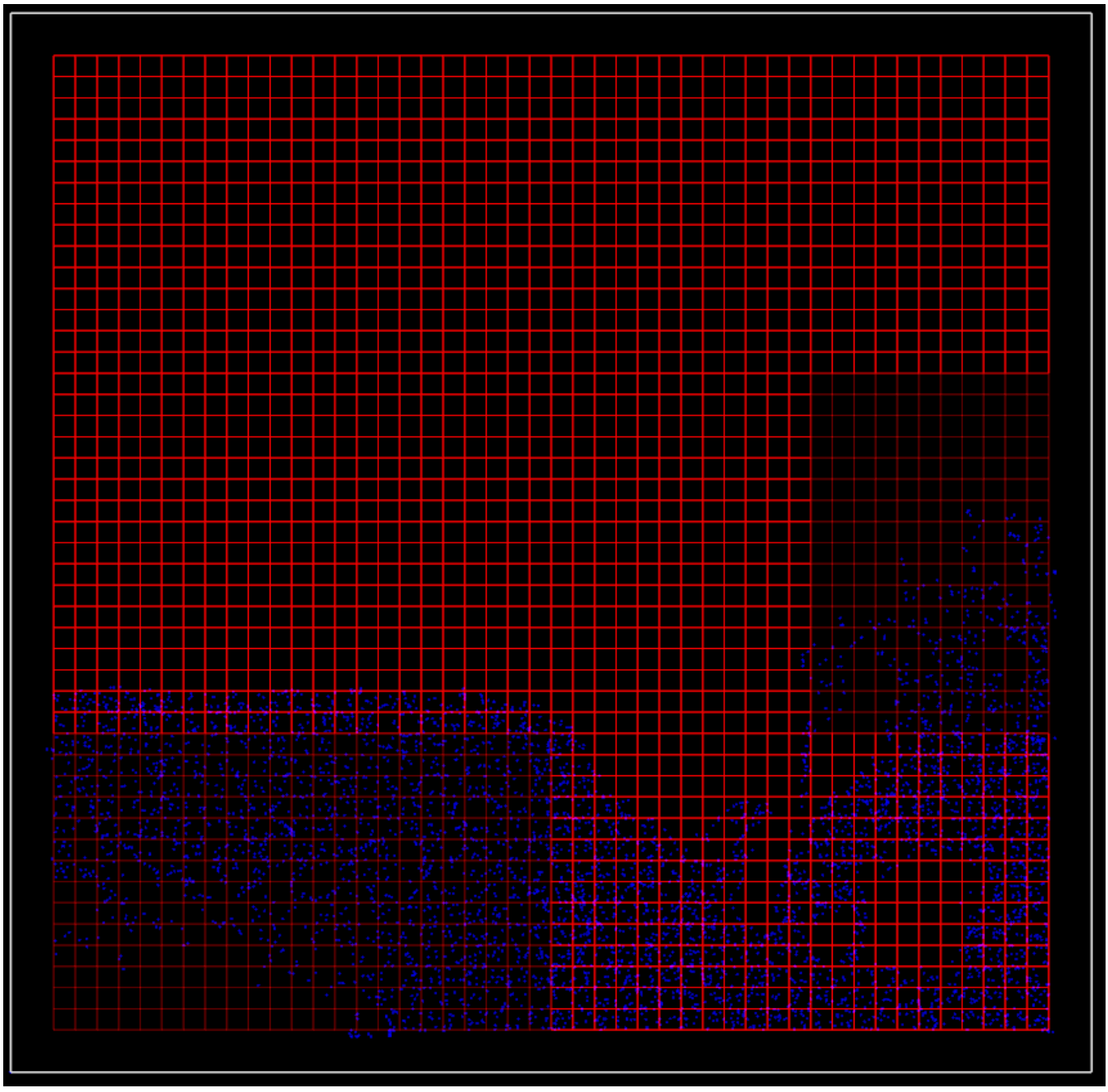

Figure 6.1: 2D fluid simulation with two blocks of porous material. Red signifies porosity and blue dots are fluid particles

6.1.4 Rendering

The rendering of a free surface fluid was done through an implicit grid shape plugin and the rendering of the wetted sand blocks was done through a volumetric grid plugin.

Free Surface Fluid

For rendering the free surface I updated the implicit grid shape plugin written by Robert Bridson, which was written for pbrt1. The method works by marching through cells in a 3D grid and extrapolates a cubic ray-implicit surface solve in each cell from 8 samples located on the vertices of each cell.

The main effort was in converting the extent of grid and various scales to match the parameters allowed by the volumegrid. Also, in order to simplify the usage of 3D grids in pbrt I implemented a simpler interface for loading them into pbrt. Instead of asserting integral "nx","ny","nz" and a float list of "<values>" one can assert a string "<values>_filename" which represents a file where the first line is the dimensions of the grid and the remaining lines are the values of the grid.

Figure 6.2: Rendering of a skull for testing the implicit grid module

Sand Volume

Dry sand doesn't allow for much light to pass through it and so its scattering is dominated by backscattering, which on a volumetric grid forces most of the light to bounce back from the surface. By increasing the amount of forward scattering the colouration of the sand becomes darker, which matches changes in physical colouration one usually sees in sand. The color choice for dry sand was determined by taking a mean RGB color value of an image of dry sand.

Following the discussion of [**?**] I decided to render sand as a volumetric field using the 2-term Henyey-Greenstein function, which is a linear interpolation between two Henyey-Greenstein functions , where one is bound to promote frontscattering and the other is bound to promote backscattering. More precisely, the standard Henyey-Greenstein function

$$p_{HG}(\cos\theta, g) = \frac{1-g^2}{(1-2g\cos\theta - g^2)^{1.5}}$$

is usually bound to $g \in [-1, 1]$, where negative g imply more backscattering and postiive g imply more frontscattering, while in the 2-term Henyey-Greenstein function one usually sees

$$p_{HG2}(\cos\theta, g1, g2, w) = wp_{HG}(\cos\theta, g1) + (1-w)p_{HG}(\cos\theta, g2)$$

where $g1 \in [0, 1]$ and $g2 \in [-1, 0]$. For this project I extended the pbrt volume density grid module to use the two-termed Henyey- Greenstein phase function with the weighting value computed by interpolating a grid of w values.

6.1.5 Raw Data to PBRT

The purpose of this part of the pipeline was to allow for rapid prototyping of the pbrt scene files derived from the fluid simulation without having to rewrite the simulation and to extrapolate the data from a 2D to 3D. The python3 script outputs a pbrt file, a levelset of the fluid outside of the porous material, a grid of porosity over the whole domain, and a grid of density values inside the porous regions.

The conversion from 2D to 3D was fairly trivial for the porous regions but depended on the grid type. For the porosity and density values it was sufficient to simply copy the grids several times. However, a bit of extra work was required to render the free surface fluid as a closed surface. In order to create a surface on the added dimension on the levelset for the 2D grid I created a zero crossing by sandwitching copies of the levelset grid with a grid of strictly positive values. In 1d analog we see something like this:

$$\begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Although density values are calculated through the fluid simulation, the density in the simulation of each particle only adds to the cell that the particle lies in. In order to alleviate the density issue and simulate the smeearing out of fluid particles in a porous medium I distribute the density of fluid particles in porous regions by a cubic spline normalized so that the peak density equals the local porosity.

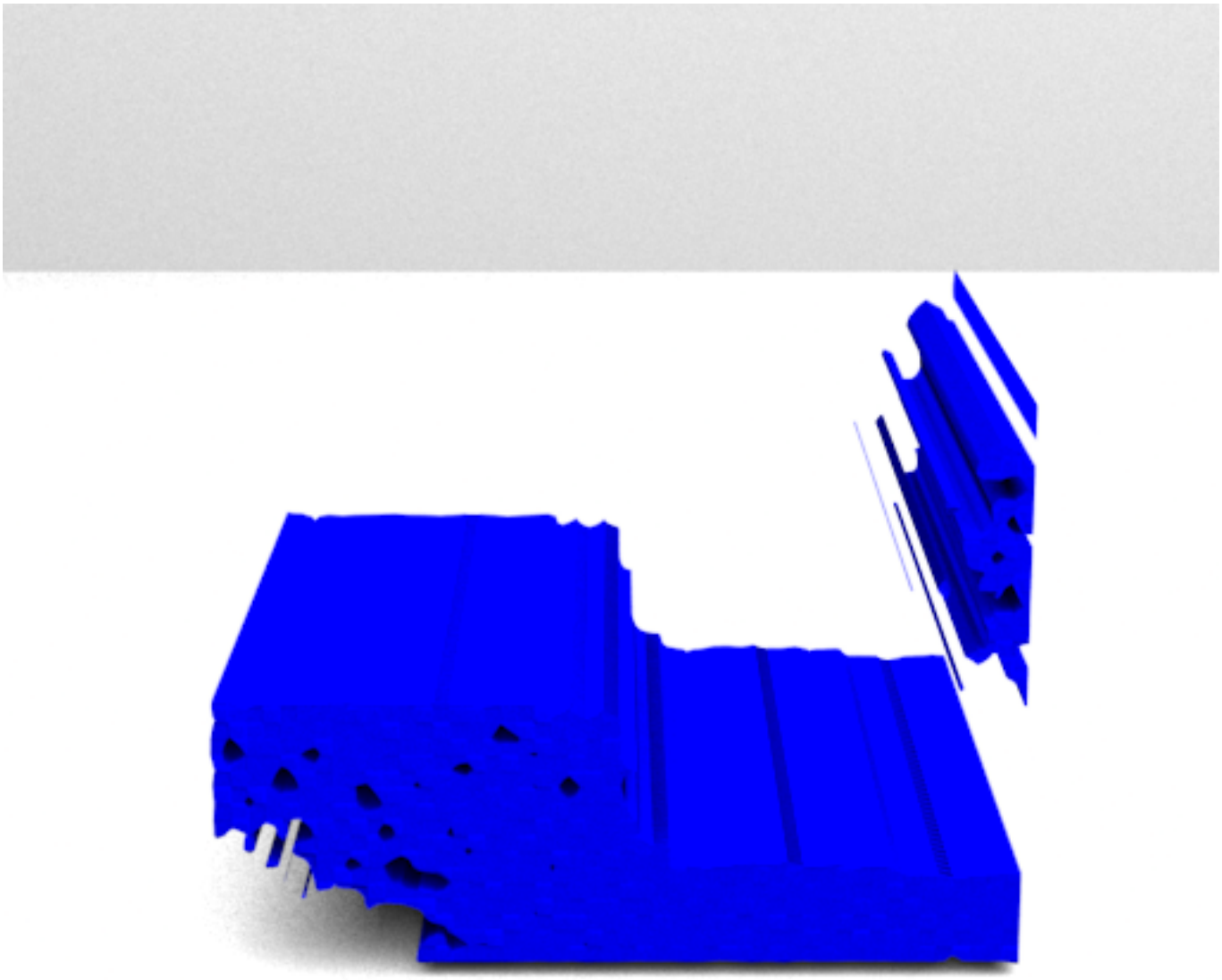

Figure 6.3: Rendering of 2D fluid from simulation using the implicit grid rendering module

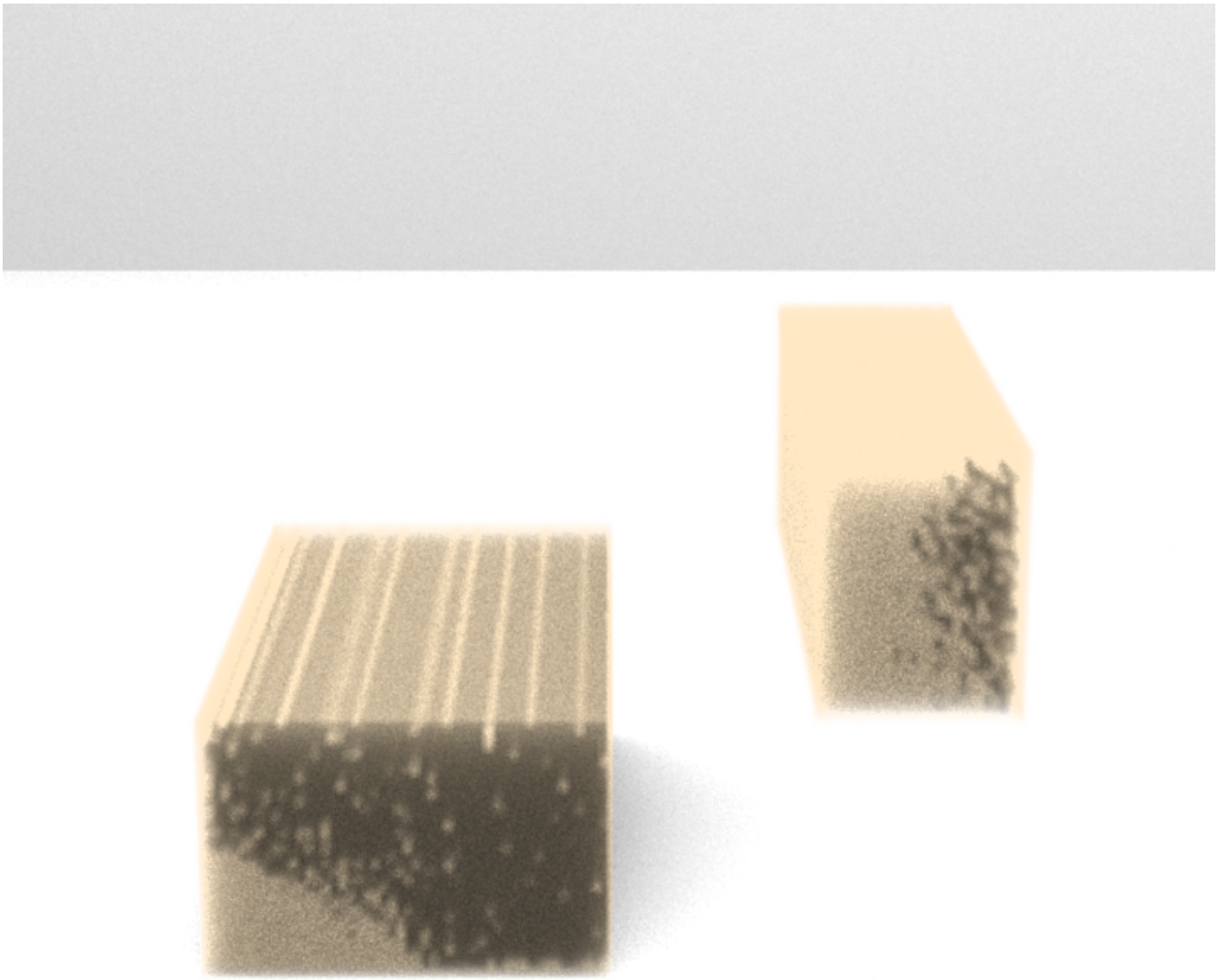

Figure 6.4: Rendering of wet sand volume

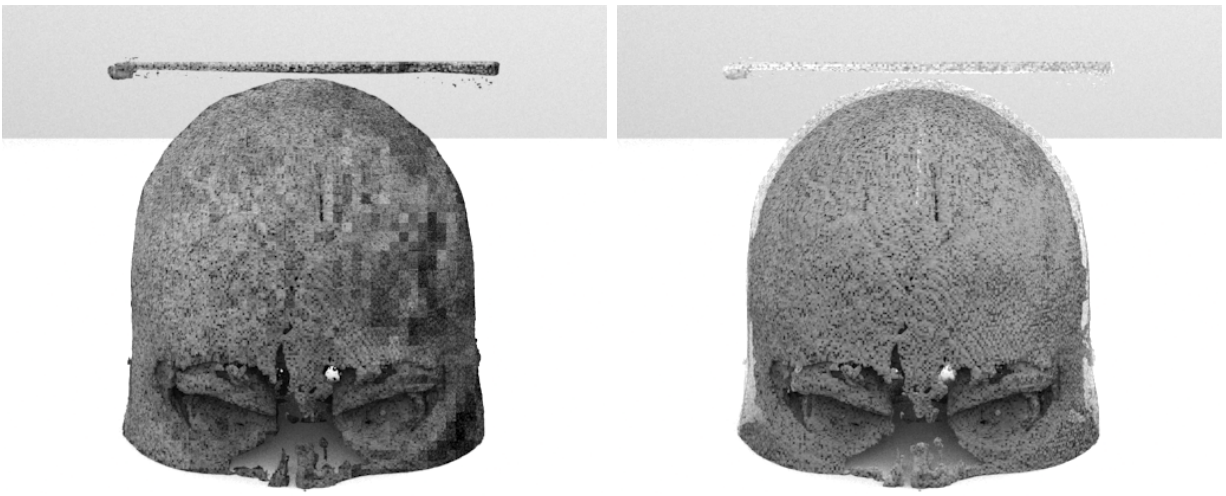

Figure 6.5: Left: skull rendered by pbrt built on gcc4.2. Right: skull rendered by pbrt built on clang3.1

6.1.6 Issues

Unfortunately there were some numerical issues with the implicit grid solver that became first visible when run on the cluster. The resulting renders had darkened squares on-axis with respect to screen space which were not appearing on my own workstation. The first thing noticeable for me was that the cluster was using a different compiler, so I installed clang3.1 on the cluster, which did remove the issues on the skull test grid. However when the implicit grid overlapped with the sand volume grid the free surface fluid simply turned black, with a few small bands of proper coloration. By tuning the rayEpsilon parameter I was able to extract more of the desired translucency at the cost of adding in black squares.

6.1.7 Conclusion

For this project I have implemented a method for rendering fluid from particles by generating an signed distance function for the fluid and then rendering it through an implicit grid module as well as implemented a method for rendering volumes of sand with variable wetness. By using what has been implemented I have been able to render fluid being advected in both a free surface and in a porous material.

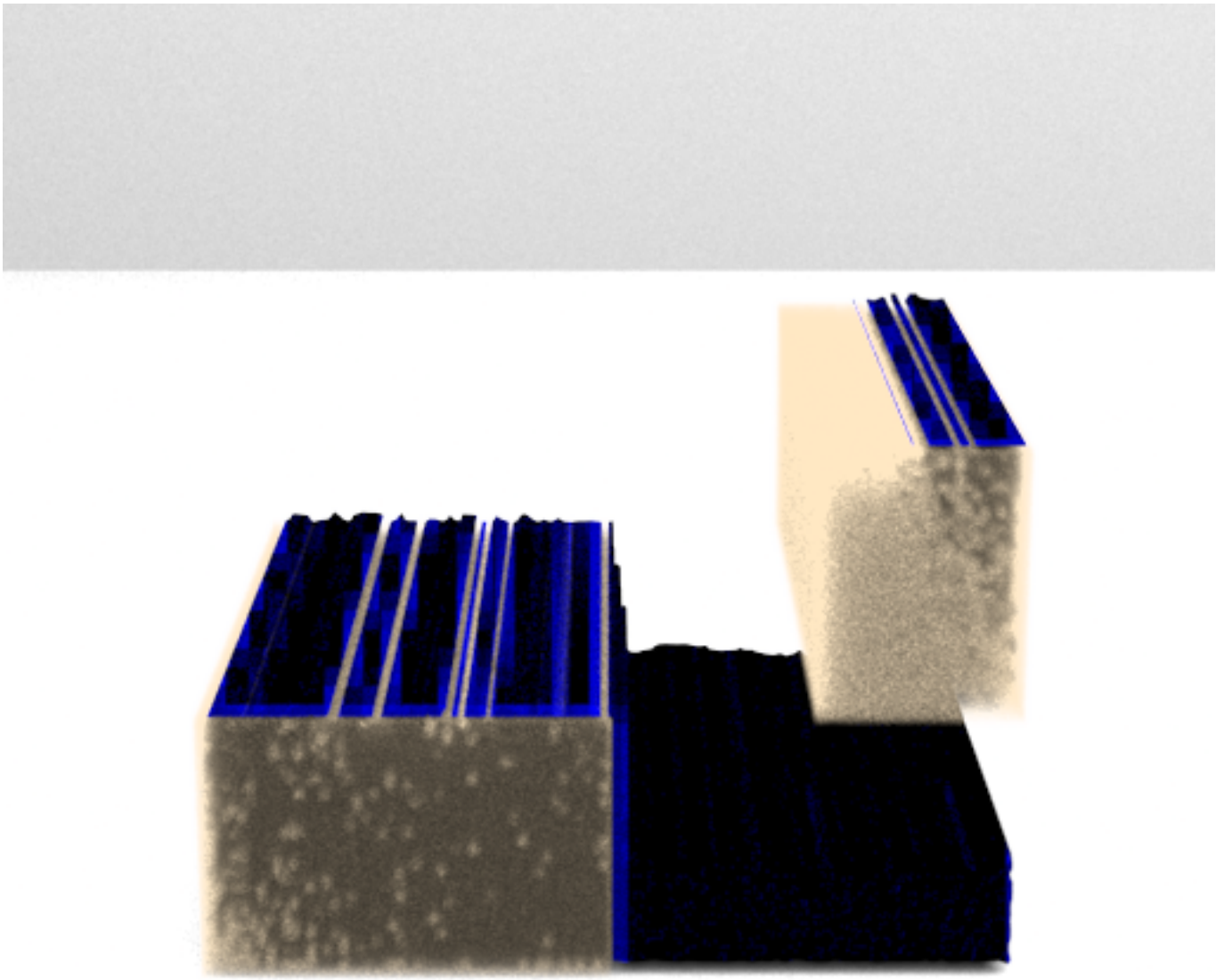

Figure 6.6: Free surface broken when sand volume grid added ontop