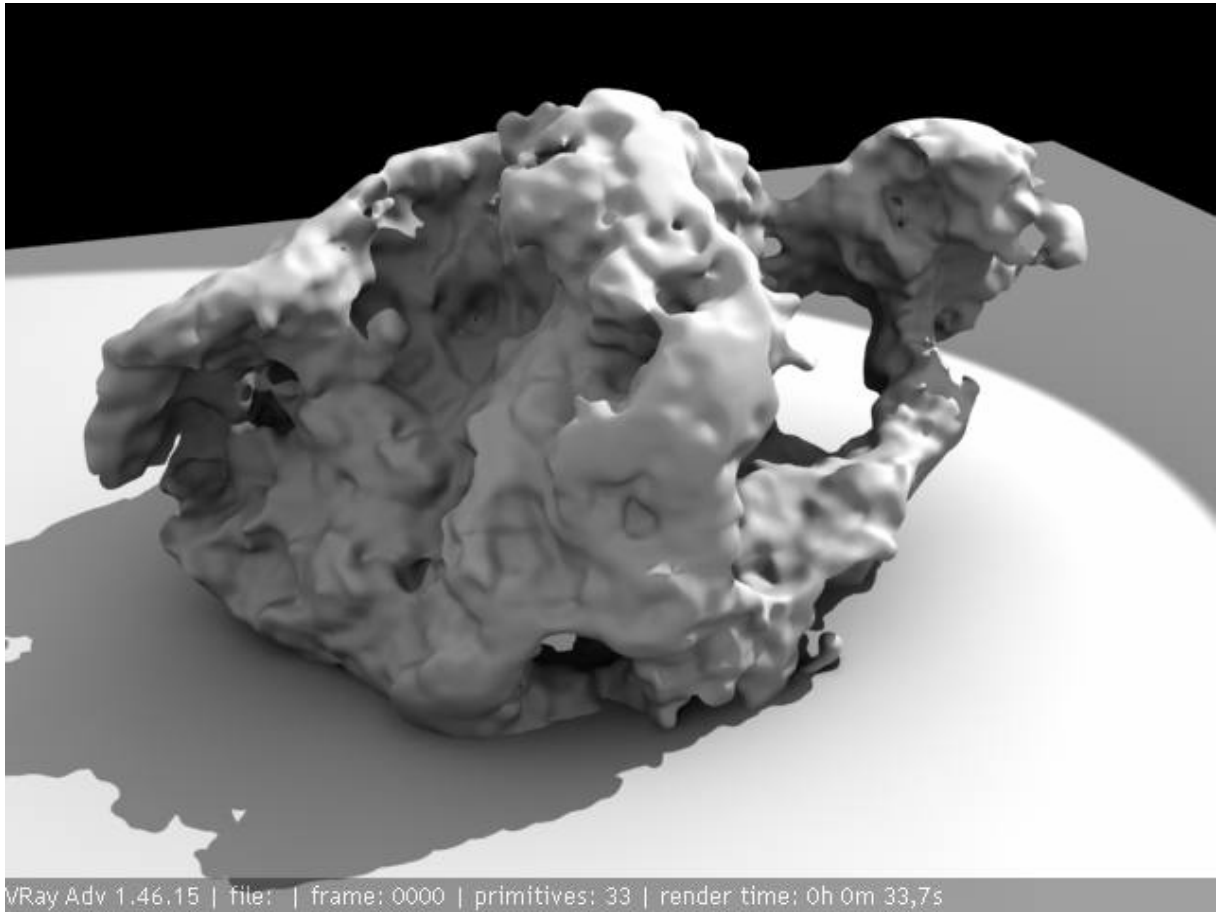


Direct Raytracing of Implicit Surfaces in 3dsmax with V-Ray



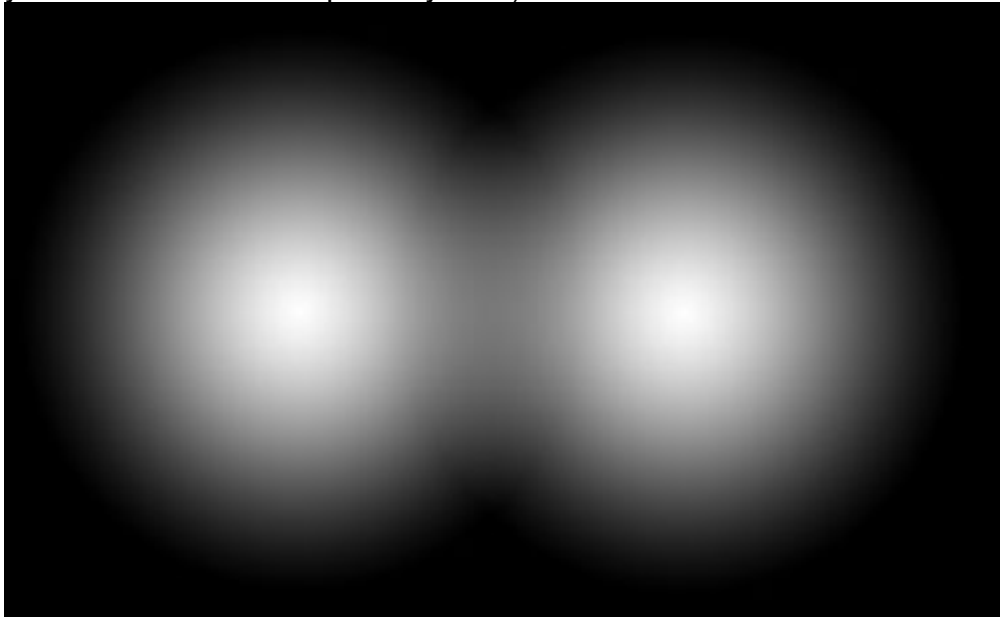
A small introduction to the theory of implicit surfaces

Dieter Morgenroth 2005

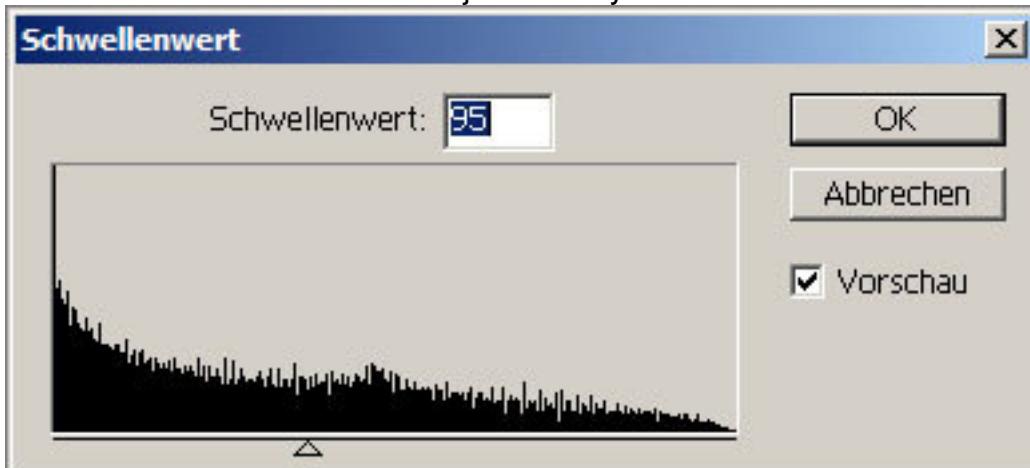
What are implicit surfaces?

Implicit surfaces or isosurfaces are defined by density fields and a threshold. Those points in space that share the same density can be visualized as a surface. Metaballs are an excellent example of this. If you put a falloff density field on every vertex of your basemesh and add them you get a blob surface. This can easily be visualized in 2D in Photoshop.

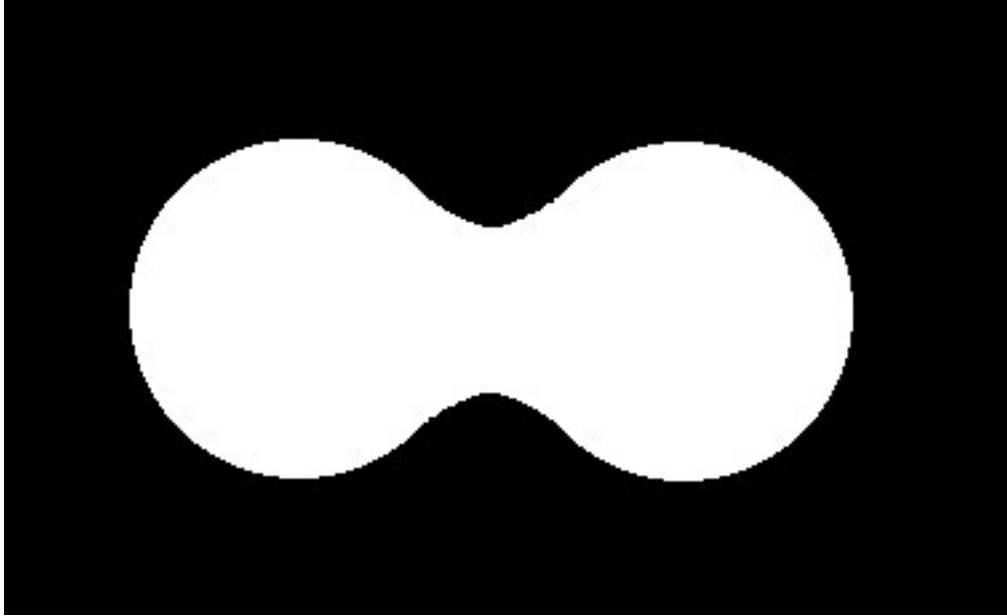
Take two circular gradients as the density fields. (Use two layers and add them so you can move them separately later)



You can now use a threshold adjustment layer on this.



This leads to a 2D “implicit surfaces”:



We can now introduce some additional noise from a procedural texture.



And multiply this to our Gradient:



This leads to a more irregular implicit surface:



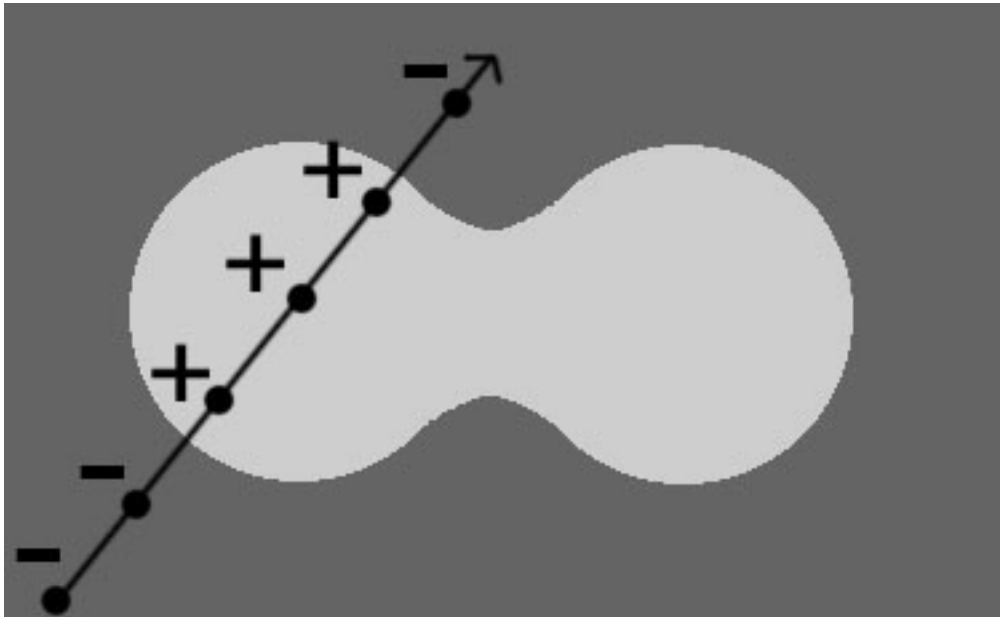
Until now this is in theory all procedural and resolution independent. For each point in space you can calculate the density field by adding the distance to each basevertex. The noise texture uses a noise function.

Feel free to try this yourself in Photoshop. If you use adjustment layers for the threshold you can even move the fields interactively!

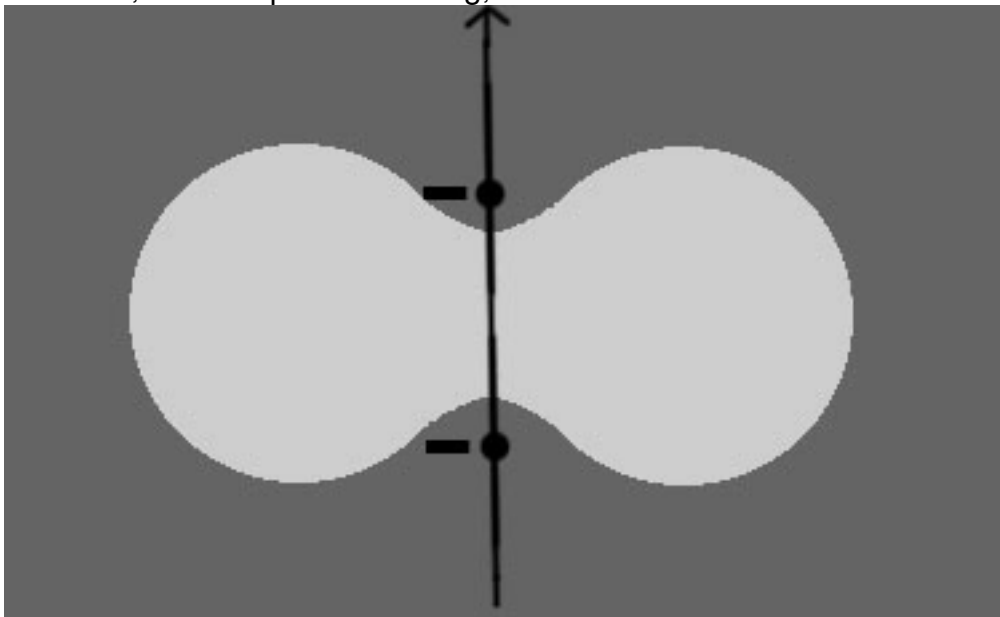
The same rules of course apply to 3D implicit surfaces.

Raytracing Isosurfaces

By raymarching into this field you can directly raytrace this isosurface. The density is evaluated along the ray at discrete steps. By subtracting the threshold from the density value, finding surface intersections is simple. If two consecutive samples have different signs the surface is intersected. To keep it simple we dump the noise texture for a moment and look at the density field only.

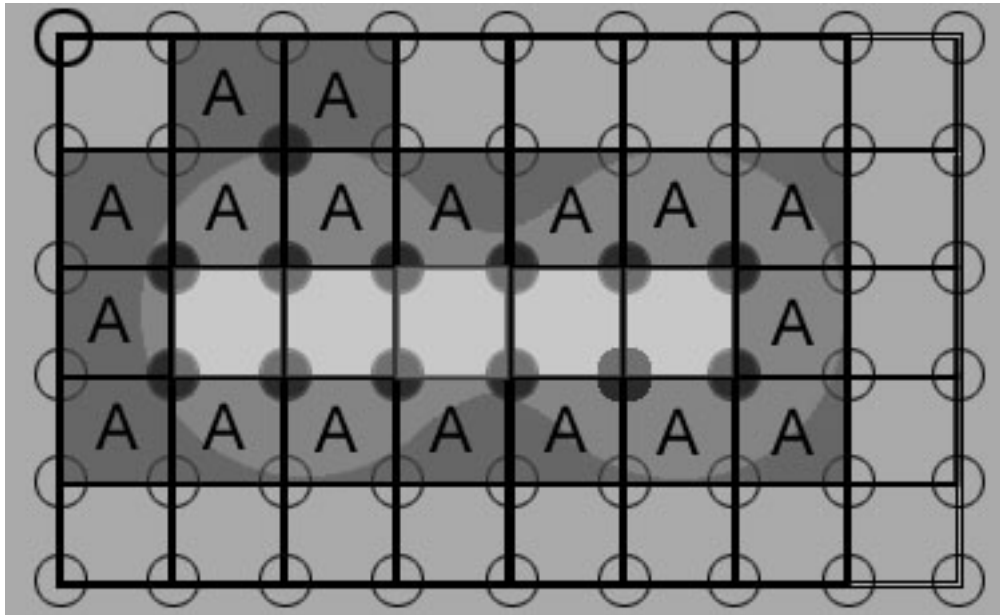


The intersection point can be refined with a binary root search in the interval that is an intersection candidate. This converges towards the exact intersection with a few iterations. This approach is considered as an efficient method in [1]. However, if the stepsize is too big, the surface can be missed.



Especially in Animations this leads to flickering edges. To avoid this the stepsize must be small enough to sample even small features. If noise textures are used, features can get very small! This of course leads to high computation times.

My approach to the problem is to divide the space with a coarse grid into voxelcells. In a first pass the corners of the cells are evaluated. If all the corners of a cell do not have the same sign, the cell is tagged as "alive".



To speed up the preparation phase I use a derivate of the fast marching method from the paper from Sethian². I start computing the density fields at the cells containing the vertices of the basemesh. Then I calculate the density of the neighbours of those cells. Until a threshold is reached I continue to calculate neighbouring cells of already calculated cells. This way I don't have to calculate the entire grid.

In the renderingphase I traverse the Voxelgrid with a 3D DDA Line Algorithm. This Algorithm is described in³. If a cell is reached that is "alive" this cell is raymarched with a small stepsize. For raymarching I don't have to calculate the density field again. I use the stored densities at the corners of the actual cell and perform a trilinear interpolation.

This method also misses certain areas, the advantage is, that these areas are always missed. As a consequence they do less flicker.

¹ "Interactive Ray Tracing for Isosurface Rendering" Parker et al.,

² "A Fast Marching Level Set Method for Monotonically Advancing Fronts" Sethian, 1995, Berkeley

³ "A Fast Voxel Traversal Algorithm for Ray Tracing" Amanatides and Woo, Toronto