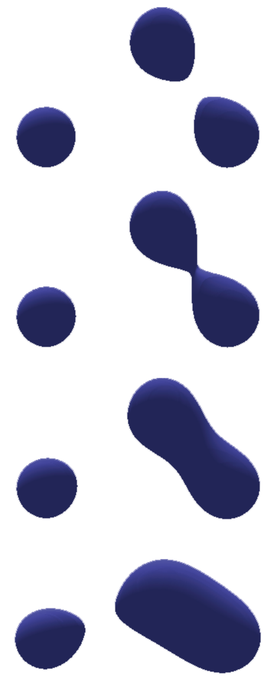
GPU-Accelerated Ray Casting of Node-Based Implicit Surfaces

Christian Lessig*, Derek Nowrouzezahrai*, Karan Singh* – Dynamic Graphics Project – University of Toronto

Introduction

Node-based implicit surfaces are a popular surface representation for modeling and animation, especially for organic shapes and natural phenomena. In this poster we present a novel GPU-based ray casting engine for the direct and efficient visualization of these surfaces. Because our approach is based on ray object intersection tests, no intermediate surface representation is necessary. This reduces the memory requirements on the GPU and the computational costs on the CPU. Additionally, ray casting also has the advantage that the results are guaranteed to be pixel-accurate. For small to medium-sized scenes our technique provides real time frame rates. For scenes with many nearby nodes we approximate the contribution of most of the nodes by an ambient field. This provides interactive frame rates and smooth surfaces even for complex scenes. We demonstrate the applicability of our method by visualizing a Lagrangian fluid simulation, a problem for which existing techniques are more complex and do not provide our pixel-accurate results.



Solution

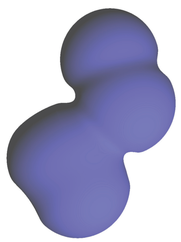


Figure 1: Blending of nodes with Equation 2 as field function.

The surfaces we want to visualize are defined as iso-value contours of a field function

$$S_F = \sum_i K(\bar{x}, \bar{x}_i) \qquad (1)$$

The function S_F is the superposition of local, monotonic decreasing kernel functions $K(x,x_i)$ which are defined at a finite number of nodes x_i. Iterative techniques must be employed for the intersection test, in contrast to previous GPU based ray casting engines where closed form solutions were available. We use Newton Iteration for the ray object intersection test because of its quadratic convergence and algorithmic simplicity.

Modern GPUs are designed to efficiently display triangle-based geometry using rasterization. This makes it necessary to map the ray casting algorithm onto the pipeline stages of current graphics hardware (Figure 3). It is well known that each eye ray is used to compute the color for one screen pixel. Therefore, the ray object intersection test can be computed in the fragment shader. This is efficient because

- the fragment processor is currently the unit with the most processing power,
- the independence of the intersection tests allows us to exploit the parallelism of the fragment processor,
- the ray setup can be performed in the rasterization unit by mapping ray directions to texture coordinates.

Similar to many GPGPU algorithms, polygonal geometry has to be rendered to invoke the computations. Rendering a screen sized quad and intersecting every object with every ray is inefficient, as most objects cover only a small portion of the screen. Therefore we render polygonal convex hulls for the scene entities. For each object this generates a tight but complete set of candidate rays which are likely to intersect the object.

For general node-based implicit surfaces it is non-trivial to compute a convex hull. Equation 1 can only be computed efficiently for certain applications such as physical simulations where a proximity constraint on the nodes exists. We therefore propose a modified field function

$$\hat{S}_F = \frac{\sum_i K(\bar{x}, \bar{x}_i)^2}{\sum_i K(\bar{x}, \bar{x}_i)} \qquad (2)$$

for which the surface is bound by the superposition of the convex hulls of the local field functions (Figure 1).

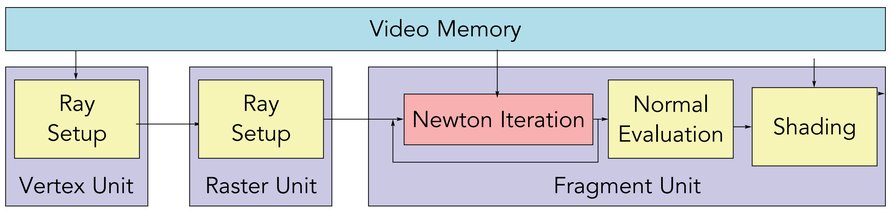


Figure 3: Mapping of the ray casting algorithm onto a GPU

Depending on the surface, often more than 100 nodes have to be taken into account when evaluating a surface point. This is very expensive and, due to hardware limitations, impossible to implement on current graphics hardware. However, omitting many summands may lead to non-smooth surfaces. We therefore sum up only the K largest terms and approximate the contribution of the remaining nodes by an ambient field. This field is precomputed on the CPU and stored in a volume texture. The resolution of the texture can be used to trade performance for quality. The implementation of the Newton Iteration and the generation of the initial guesses follow those proposed in [1]

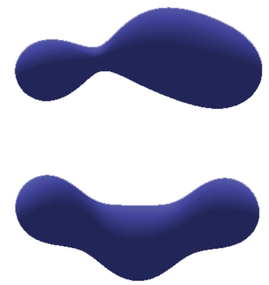


Figure 4: Blending between three nodes under the influence of attraction and repulsion forces and gravity, rendered at 45 fps

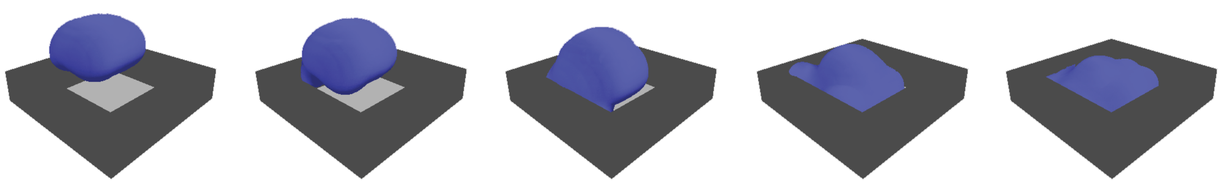


Figure 2: Liquid flowing into a tank, together simulation and rendering run at 5 fps with 100 nodes.

Results

Experimental Setup

C++, OpenGL and GLSL •
Pentium D 3.2 GHz •
1 Gbyte of RAM •
Nvidia 7800 GT •
512 x 512 image resolution •

To demonstrate the applicability of our ray casting engine we employ it to visualize a Lagrangian fluid simulation, similar to the one proposed by Müller et al. [2]. This is a particularly attractive example because Lagrangian simulations are node-based and lend naturally to an implicit surface. For scenes with a limited number of nodes where no ambient field is needed, we achieve real-time frame rates (Figure 4), for more complex scenes where the ambient field is used we achieve interactive frame rates (Figure 2).

Currently, the two main limitations are

- the use of the ambient field which has to be computed on the CPU and which is only an approximation,
- the not perfect convergence of the Newton Iteration.

We believe that the next generation of graphics hardware will provide the feature set and compute power to discard the ambient field and significantly improve the performance.

Avoiding pixel artifacts caused by failing intersection tests will be one of the main objectives of our future work; improving the initial guesses for the Newton Iteration is one possibility; an alternative is to employ root finding algorithms which are guaranteed to converge. Another avenue for future work is to explore other applications for our algorithm, such as modeling systems or blending polygons with implicit surfaces [3]. We also plan to implement the Lagrangian simulation on the GPU so that physical simulation and visualization run on the GPU.

References

[1] J. F. Blinn, *A Generalization of Algebraic Surface Drawing*, SIGGRAPH 1982.
[2] M. Müller, D. Charypar, M. Gross, *Particle-Based Fluid Simulation for Interactive Applications*, SCA 2003
[3] Karan Singh, R. Parent, *Joining Polyhedral Objects using Implicitly Defined Surfaces*, The Visual Computer 17, 2001

* {lessig,derek,karan}@dgp.toronto.edu