Hessian-based Robust Ray-Tracing of Implicit Surfaces on GPU

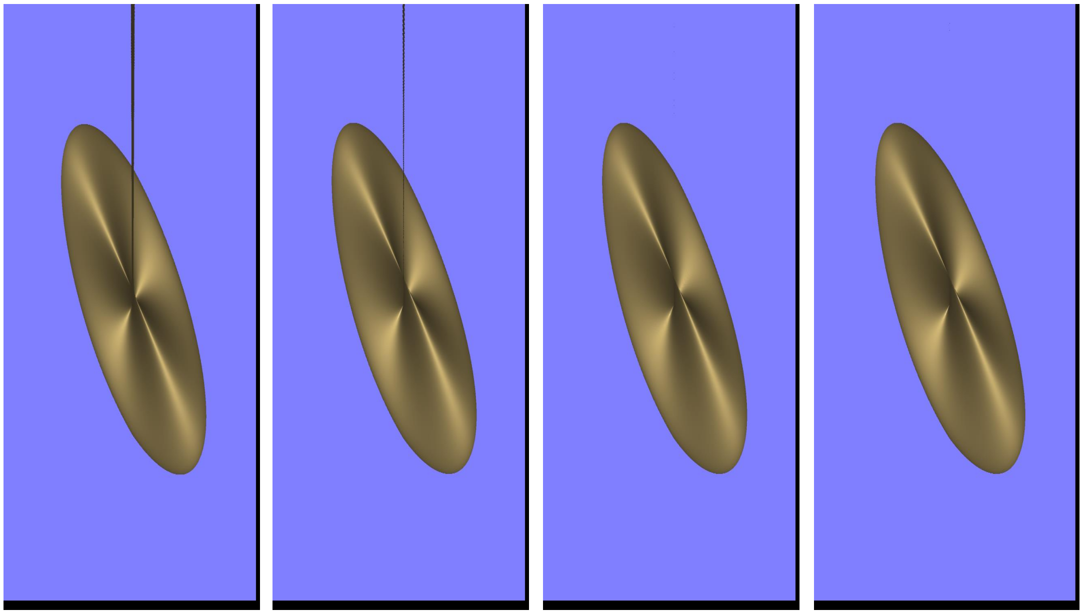

Figure 1: *(Ray-Traced Cross-Cap Surface) Left to Right Algorithms: Taylor Test, Taylor Test with Mean Curvature, Taylor Test with Approximate Hessian, and Taylor Test with Exact Hessian.* **The lines are self-intersecting artifacts, which reduce significantly on using Hessian.**

Abstract

In recent years, the Ray Tracing of Implicit Surfaces on a GPU has been studied by many researchers. However, the existing methods have challenges that mainly includes solving for self-intersecting surfaces. General solutions for Ray Tracing suffer from the problem of false roots, and robust solutions are hard to generalize. In this paper, we present a robust algorithm based on Extended Taylor-Test Adaptive Marching Points, which allows a robust rendering of Self-Intersecting Implicit Surfaces on a GPU. We are using the Second Order Taylor Series expansion to alleviate the problem of double-roots in Self-Intersecting Implicit Surfaces. Our approach is simple to implement and is based on the Hessian Matrix of the Implicit Surface that can be attributed to the Hessian Matrix can be used to obtain second-order Taylor Series expansion for the uni-variate ray-equation. We compare our results using the simulated ground-truth with the smallest step-size possible with proposed algorithm, and our proposed algorithm gives the best visual results as well as highest SSIM percentage than other approaches.

1 Introduction

Implicit Surfaces are an important category in Computer Graphics (CG) as they are compact and can be evaluated quickly. In general, an Implicit geometry is defined by an equation $S(x, y, z) = 0$ and different forms of $S(x, y, z)$ are possible. For example, an algebraic form which can be represented by a polynomial equation and transcendental for which uses trigonometric functions. The basic idea is to reduce the surface $S(x, y, z) = 0$ to the form an equation, $F_f(t) = 0$ by substituting the ray-equation in the implicit surface, which results in a uni-variate equation. The uni-variate equation needs to be solved fragment-wise. Each fragment can then solve for t and perform per-pixel lighting based on the point of intersection and a normal at the point of intersection. The root-finding for ray-surface intersection is usually done in two steps, i.e., root-bracketing and root-isolation for General implicit surfaces [Hart 1996; Singh and Narayanan 2010; Knoll et al. 2007]. The problem becomes difficult when the implicit surface is self-intersecting, and then root bracketing becomes complex and might fail. In this paper, we specifically address the problem of the rendering of Self-Intersecting implicit surfaces on a GPU.

In the rest of the paper, we discuss the Related Work in Section **??** and specifically the method using Mean-Curvature in Section 1. We describe the details of the proposed method using the Hessian Matrix in Section 3. We finally discuss results and future-work in 4.

2 Related Work

In [Hart 1996] author has introduced a concept of Geometric distance which is hard to compute and not easy to generalize. However, it can help to obtain robust results as indicated in [Hart 1996]. In [Knoll et al. 2007; Mitchell 1990] authors have proposed robust approaches which give better results, but the methods are computationally expensive and hard to generalize. The approach discussed in [Mitchell 1990] isolates the root using repeated bisections till the interval in t contains a single root. Reliable interval-extensions, however, are difficult to compute for large intervals in the domain of complex functions. In [Singh and Narayanan 2010] authors present a Taylor Test for root bracketing which performs decently well and is easy to compute with some false roots. Furthermore, [Singh 2017] uses a differential geometry concepts to alleviate the problem of false roots as they use the sign of Mean Curvature to decide if more or less sampling needs to be done for the root bracketing. The authors chose the base step-size by dividing the ray into steps of Δt and defined the base stepsize as $2.0/\mu(S(t))$. Further, the formulation for Mean Curvature is used from [Goldman 2005] as

H = $-\nabla \cdot \nabla S$ without division by magnitude of the normal as that does not change the sign of H, where H is used to decrease the step size by a factor of 0.5 when negative and increase by a factor of 2 when a positive sign is obtained. Finally, they use Taylor test for root bracketing [Singh 2017] without user-defined constants $\tau 1$ and $\tau 2$. This results in partial alleviation of the problem as the method still shows false-roots as we show in the results later.

2.1 Related Work: Adaptive Marching Points Algorithm with Taylor Test Overview

This section provides the detailed review of the method proposed by authors [Singh and Narayanan 2010] i.e., the Adaptive Marching Points Algorithm. The parametric form of 3D ray from a pixel or a fragment(f) is $p(t) = O + tD_f$, where t is the ray parameter, O the camera center, and D_f the direction of the ray. Substituting the 3D coordinate x, y, z from the ray equation into the surface equation $S(x, y, z) = 0$, we obtain the Equation 1 given as:

$$F_f(t) = 0. \tag{1}$$

Furthermore, the authors are interested in computing the smallest positive t as the object which is considered opaque, and it is the point of intersection. Hence, each pixel needs to solve Equation 1 independently and find the corresponding root. Once, the root is found we need to do shading for which the normal is a gradient $\vec{\nabla} S(x, y, z)$ and can be used for lighting and shadows. The following algorithm describes the steps involved in the related approach in [Singh and Narayanan 2010]:

Algorithm 1 Adaptive Marching Points (f, b)

1: Find the intersections t_{near} and t_{far} of the ray for fragment f with the near and far planes.
2: Initialize $delta$ to the base step size b; t to starting point t_{near}
3: **while** $t < t_{far}$ **do**
4: Set the stepsize δ using Equation 2.
5: **if** rootExistsIn $(t, t + \delta)$ **then**
6: Goto step 11 with $[t, t + \delta]$ as the isolated interval
7: **end if**
8: $t = t + \delta$
9: **end while**
10: If we have no interval as isolated, then we discard the pixel.
11: Perform 10 steps of Newton-bisections of the isolated interval.

Finally, the authors perform two kinds of adaptations on the base Algorithm 1:

1. **Distance Adaptation:** The magnitude of $|S(x, y, z)|$ is used as a *proximity measure*, further which is used as an approximation to algebraic distance. Furthermore, the base step-size is doubled when algebraic distance is greater than τ_2 and halved when algebraic distance is less than τ_1. The thresholds (τ_1 and τ_2) are defined as recommended by the authors.
2. **Silhouette Adaptation:** Silhouettes are the regions close to the boundary of the surface and hence is important to increase the sampling rate as argued by the authors. They use the magnitude of the derivative $F'_f(t) = \vec{\nabla} S(x, y, z) \cdot D_f$, which serves as *horizon measure* and is close to zero, near internal and external silhouettes of the even complex implicit surfaces. Thus, $|F'_f(t)| \leq \tau_3$ is satisfied and they use it for further reduction of the step-size.

Combining the distance and silhouette adaptation, the authors fix the step size in each iteration using the formula given by Equation 2:

$$\delta = \begin{cases} b/4 & \text{if} \quad |S(p(t))| \leq \tau_1 \text{ and } |\vec{\nabla} S(p(t)) \cdot D_f| \leq \tau_3 \\ b/2 & \text{if} \quad |S(p(t))| \leq \tau_1 \\ 2b & \text{if} \quad |S(p(t))| > \tau_2 \\ b & \quad\quad \text{otherwise} \end{cases} \tag{2}$$

where b is the base step-size and τ_1, τ_2 and τ_3 are the user-defined thresholds. Once, the step-size is fixed one needs to do the root-containment test (Step 5, Algorithm 1). This is done by the authors in the following two ways:

Sign test: This test is simple and root exists if the function changes sign between the end points of the step, i.e., if $(S(p(t_i)) * S(p(t_{i+1})) < 0)$. However, it can miss the roots and produce false-roots.

Taylor test: The authors take first order Taylor series approximation of the function at the mid-point of an interval, and evaluated the series from both endpoints. It works well for moderate length of intervals. Furthermore, authors define the interval extension called Taylor-extension as follows: The extension of F in the interval $[t_i, t_{i+1}]$ is defined as $\tilde{F}([t_i, t_{i+1}]) = [\min\{p, q, r, s\}, \quad \max\{p, q, r, s\}]$, where

$$\begin{aligned} q &= F(t_i) + F'(t_i)\tfrac{(t_{i+1}-t_i)}{2}, \qquad & p &= F(t_i), \\ r &= F(t_{i+1}) - F'(t_{i+1})\tfrac{(t_{i+1}-t_i)}{2}, \qquad & s &= F(t_{i+1}) \end{aligned} \tag{3}$$

However, this test is slower than the sign test as more computations are required for obtaining the derivatives. Further, it can produce false-roots too.

3 Proposed Approach: Hessian-based Robust Ray-Tracing of Implicit Surfaces

In this section, we describe the proposed approach based on the Hessian-based Taylor-Test for Robust Ray-Tracing of an Implicit Surface on the GPU. This, when compared to Mean-Curvature based approach from [Singh 2017] results in much cleaner formulation and results, are considerably better.

The definition of Hessian Matrix for an Implicit Surface S(x,y,z) is :

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 S}{\partial^2 x} & \frac{\partial^2 S}{\partial^1 x \partial^1 y} & \frac{\partial^2 S}{\partial^1 x \partial^1 z} \\ \frac{\partial^2 S}{\partial^1 y \partial^1 x} & \frac{\partial^2 x}{\partial^2 y} & \frac{\partial^2 S}{\partial^1 y \partial^1 z} \\ \frac{\partial^2 S}{\partial^1 z \partial^1 x} & \frac{\partial^2 S}{\partial^1 z \partial^1 y} & \frac{\partial^2 S}{\partial^2 z} \end{pmatrix}$$

The Extended-Taylor extension of F in the interval $[t_i, t_{i+1}]$ is defined as follows $\tilde{F}([t_i, t_{i+1}]) = [\min\{p, q, r, s\}, \quad \max\{p, q, r, s\}]$, where

$$\begin{aligned} &p = F(t_i), \\ &q = F(t_i) + F'(t_i)\tfrac{(t_{i+1}-t_i)}{2} + F''(t_i)\tfrac{(t_{i+1}-t_i)^2}{2}, \\ &r = F(t_{i+1}) - F'(t_{i+1})\tfrac{(t_{i+1}-t_i)}{2} + F''(t_i)\tfrac{(t_{i+1}-t_i)^2}{2}, \\ &s = F(t_{i+1}), \\ &F'(ti) = \bigtriangledown S(x, y, z) \bullet D_f \\ &F''(ti) = (D_f, (H(O + ti * D_f) \star D'_f)) \end{aligned} \tag{4}$$

A sign-change in the proposed Hessian-based Taylor-Test will show the presence of the roots. This results in a simpler formulation and provides a better Interval Extension as compared to Mean-Curvature based approach [Singh 2017].

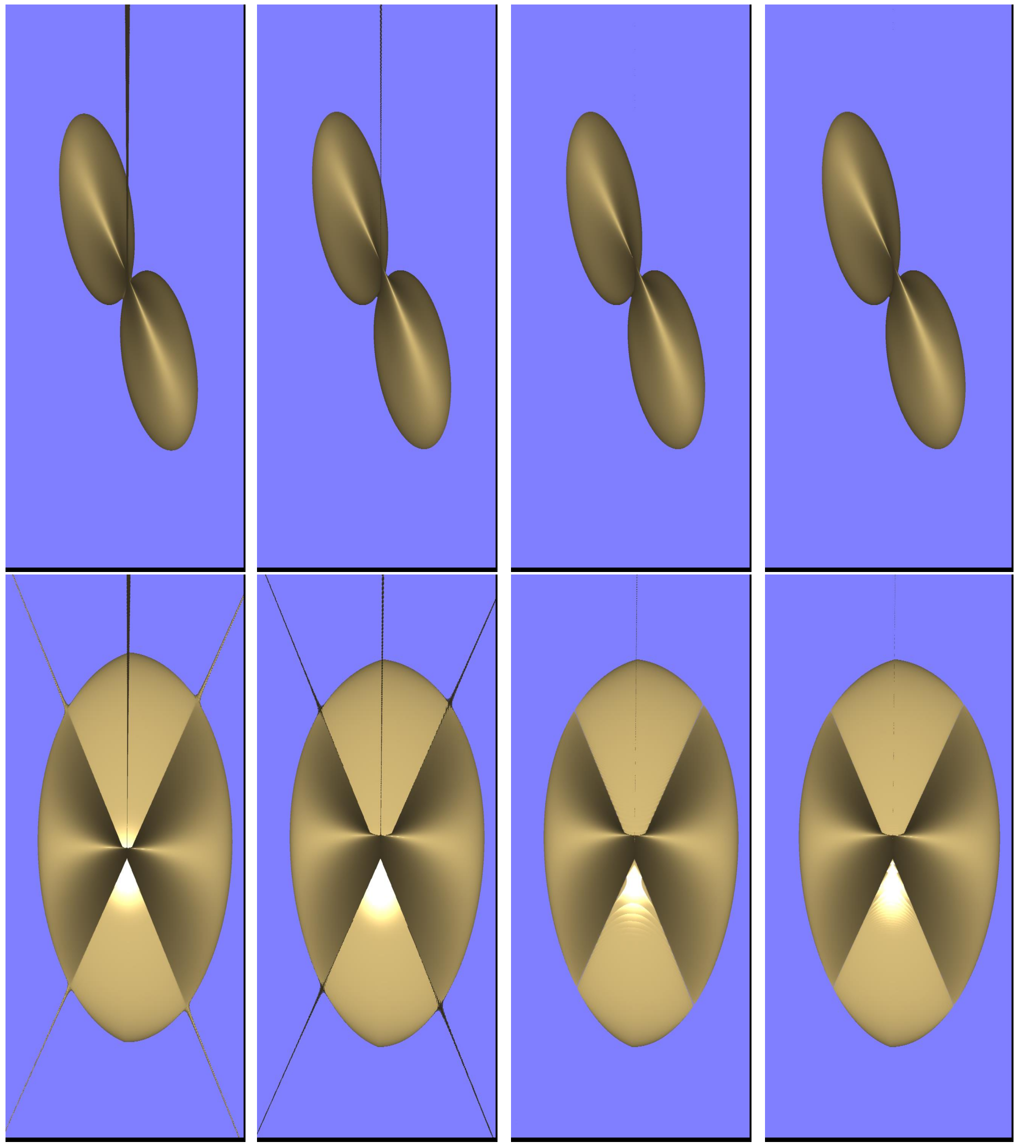

Figure 2: *(Ray-Traced Surfaces) Left to Right Algorithms: Taylor Test, Taylor Test with Mean Curvature, Taylor Test with Approximate Hessian, and Taylor Test with Exact Hessian.* **The lines are self-intersecting artifacts, which reduce significantly on using Hessian except for Steiner.**

4 Results & Future Work

We present our result on the three Self-Intersecting Implicit Surfaces which are Steiner Surface, Cross-Cap Surface and Miter Surface whose equations are given by the Equation 5, 6 and 7 respectively. Table 1 shows comparative results on NVidia Quadro P5000 system with our technique. We are using GLSL for the implementation. We just need additional computation of Hessian Matrix and we get state-of-the-art results using it. Figure 2 and Figure 1 show results where we get the fewest false-roots against compared algorithms.

We compute simulated Ground Truth by increasing the number of steps and reducing the step-size to smallest possible with the Hessian-based approach. Ground-Truth was computed by Hessian-based approach because Mean-Curvature and Taylor-Test approaches were still giving false-roots even at very small step-sizes.

Since the Hessian Matrix is defined for General Implicit Surfaces except for the non-differentiable ones. We would like to check the generalization on differentiable Implicit Surfaces as part of future work.

Following are the equations of the Implicit Surfaces used in this work:

Steiner Surface

$$x^2y^2 - x^2z^2 + y^2z^2 - xyz = 0. \tag{5}$$

Cross-Cap Surface

$$4x^2(x^2 + y^2 + z^2 + z) + y^2(y^2 + z^2 - 1) = 0. \tag{6}$$

Miter Surface

$$4x^2(x^2 + y^2 + z^2) - y^2(1 - y^2 - z^2) = 0. \tag{7}$$

Surface	Taylor Test	Mean Curvature	Hessian Approximate	Hessian Exact
Cross-Cap[4]	592(96.28%)	150(99.64%)	152(99.91%)	23
Miter [4]	580(95.74%)	154(99.57%)	158(99.91%)	24
Steiner [4]	416(92.14%)	151(96.67%)	88(98.78%)	25

Table 1: *Frame rates for self-intersecting Implicit Surfaces for a 1024×1024 window on an NVidia Quadro P5000. The order of algebraic surfaces appears within square brackets. The step size for Taylor Test, Mean Curvature, and Approximate Hessian is 0.01, and Exact Hessian is 0.002.* $\tau_1 = \tau_2 = \tau_3 = 0.01$.

References

GOLDMAN, R. 2005. Curvature formulas for implicit curves and surfaces. *Comput. Aided Geom. Des. 22*, 7 (Oct.), 632–658.

HART, J. C. 1996. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10, 527–545.

KNOLL, A., HIJAZI, Y., HANSEN, C., WALD, I., AND HAGEN, H. 2007. Interactive ray tracing of arbitrary implicits with simd interval arithmetic. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, IEEE Computer Society, RT '07, 11–18.

MITCHELL, D. P. 1990. Robust ray intersection with interval arithmetic. In *Proceedings on Graphics Interface '90*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 68–74.

SINGH, J. M., AND NARAYANAN, P. J. 2010. Real-time ray tracing of implicit surfaces on the GPU. *IEEE Transactions on Visualization and Computer Graphics 16* (March), 261–272.

SINGH, J. M., 2017. Robust ray-tracing of implicit surfaces on the gpu. http://www.highperformancegraphics.org/2017/program/.