Real Time Ray Tracing of Implicit Surfaces Utilizing GPGPU Computing

Raymond L. Imber
Department of Computer Science, University of Nevada Las Vegas
Dr. Donald Spickler
Department of Mathematics, Salisbury University
Dr. Enyue Lu
Department of Computer Science, Salisbury University

Abstract

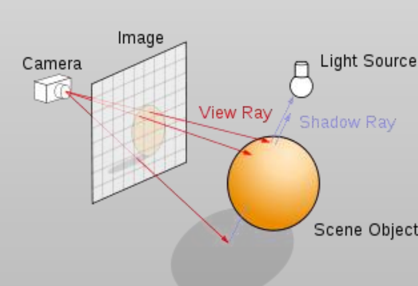
Real time ray tracing using the GPU has recently become possible and reasonable, but one of the largest bottlenecks associated with GPU ray tracing is the movement of model data onto the hardware for processing. Implicit Surfaces can be represented very compactly, making them ideal for GPU architecture.

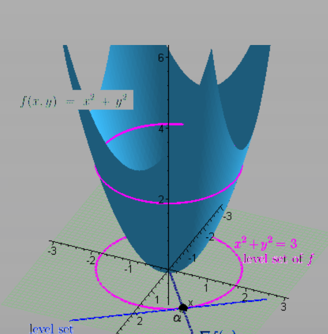
Real time ray tracing of general implicit surfaces is widely applicable, but very little work has been done regarding the rendering of general implicit surfaces on the GPU. Singh[2] is one of the only published research on the subject. Singh uses a pixel shader based approach to great success, but with limitations.

I propose using NVIDIA CUDA as an alternative to pixel shaders, to allow for much greater flexibility. I also propose using an alternative root finding method known as Ridders' Method, to add robustness to the ray tracing algorithm.

Ray TracingTheory

Ray Tracing produces images by simulating the path of light rays through a scene. This process is accomplished by starting from the image plane and "tracing" the path of a light ray back into a scene. The process of tracing the ray involves checking if the ray has intersected any object in the scene. Such an intersection implies that the ray was reflected by that object. Color and other light information are then applied to the ray based on the reflective and refractive properties of the object. This process is repeated for a large sampling of rays to produce a final image.



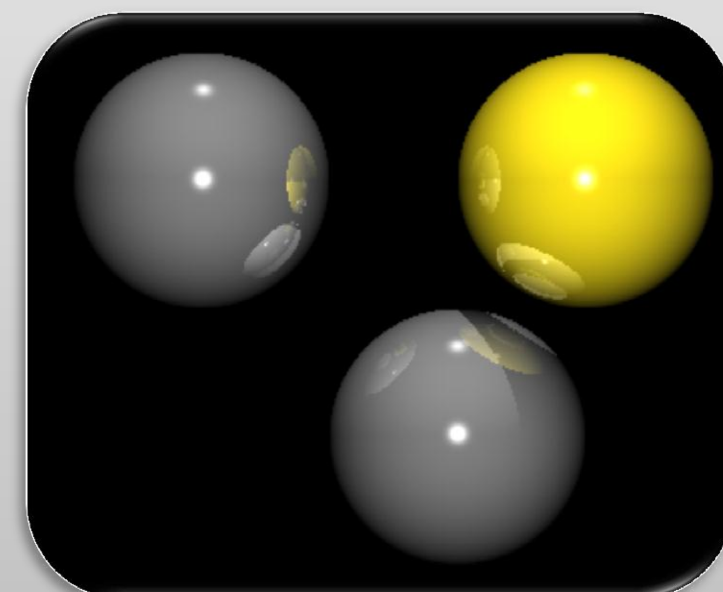


A Surface in $\mathbb{R}^3$ can be described implicitly by the set of all points where a function, $f(x,y,z)$, is equal to 0. This set is known as a level set, or level surface, of $f(x,y,z)$.

Checking for an intersection between a ray and a surface in the scene can be reduced to finding the root of the function describing the surface, when projected onto $r(t)=(x,y,z)$, the parametric function of the ray. $f(r(t))=0$. This implies the ray has intersected the level surface of $f(x,y,z)$.

A useful colliery of this result is the gradient of $f(x,y,z)$ at a point is perpendicular to the level set of $f(x,y,z)$ at that point. This is the definition of the normal vector at a point for a surface, which is necessary for computing shading information for a surface.

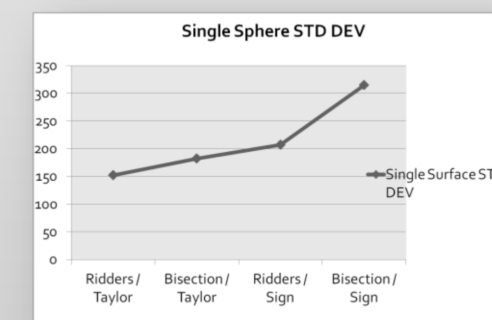
Results

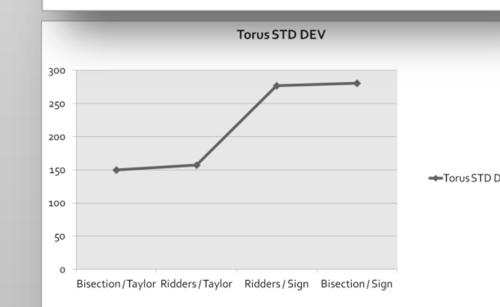


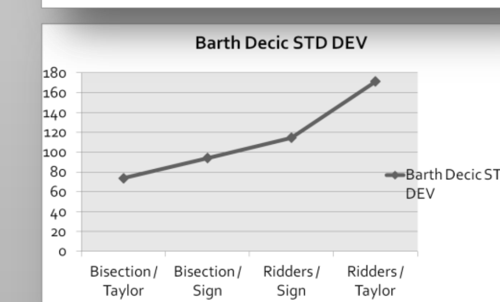
(a) 3 Sphere Scene, analytically solved

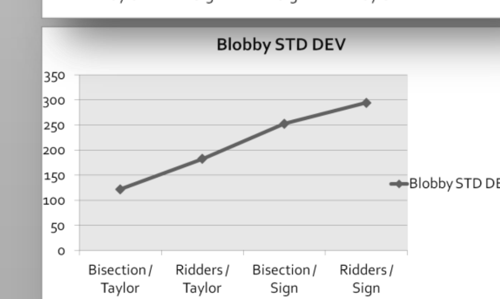
(b) Torus, Ridders' / Taylor method
step size 0.3

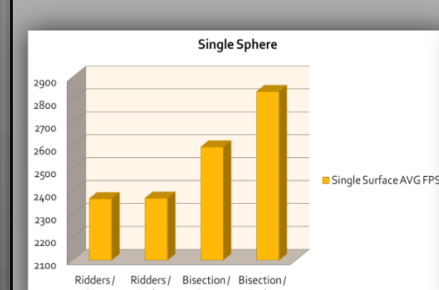
(c) Non-Algebraic Blobby described by the level surface:
$x^2+y^2+z^2+sin(4x)-cos(4y)+sin(4z)-1=0$
Left: Ridders' / Taylor method, step size: 0.5
Middle: Ridders' / Taylor method with invalid roots highlighted
Right: Bisection / Taylor method, step size 0.5

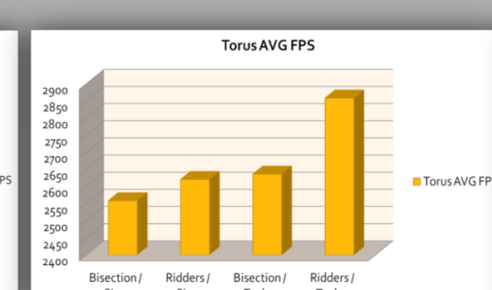


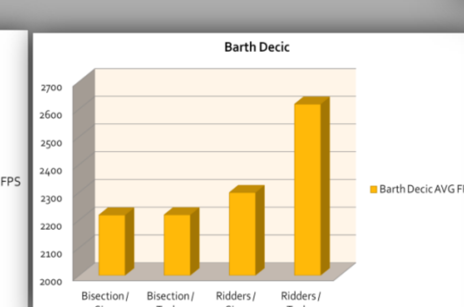


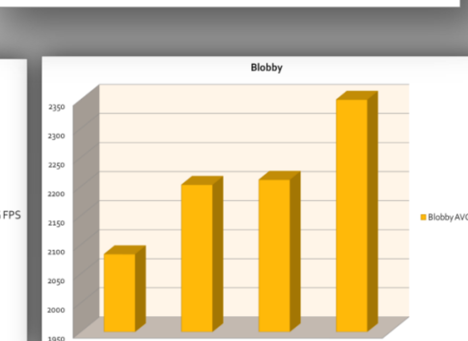












Ray Marching

Sing[2] suggests a numerical approximation algorithm called ***ray marching***. Ray marching consists of two steps. First, "march" down the ray, evaluating the function at the end points of each march interval. An interval extension test can be used to determine if a root lies between the interval. If it is determined that a root exists in the interval, proceed to step two. Step two uses numerical approximation methods to approximate the root between a set interval.

There are two interval extension methods that were tested in this experiment:

1.) **The Sign Test:** This test checks for a difference in sign between the values of the function at each end point. A difference in sign implies the existence of a root by the intermediate value theorem. It may return a false negative if there are an even number of roots in the interval.

2.) **The Taylor Test:** This test checks for not only a change in the sign at the end points, but also checks for a difference in sign of the first order Taylor Series expansion centered on each end point and extended to the mid point. This makes the method much more robust.

There are two numerical approximation methods that were tested in this experiment:

1.) **Bisection Method:** This method successively bisects the interval, using the midpoint as the approximation of the root at each iteration.

2.) **Ridders' Method[1]:** This method uses the two end points and the midpoint of the interval to transform the function at the three points to a line. The root of this line is then used as the approximation of the root at each approximation.

Conclusions and Future Work

The experiment measured all four possible configurations of interval tests and numerical approximation methods. Both average frames per second and standard deviation were measured. Frames per second was interpreted as the raw performance of the algorithm, while standard deviation was used to interpret the stability of the algorithm.

The success of a marching method is highly dependant on the surface being rendered, but certain patterns did begin to emerge. The interval extension test used seemed to have the greatest impact on both performance and stability of the algorithm. The Taylor test consistently produces higher frame rates and lower standard deviation, showing the Taylor test to be both fast and robust for use with the GPU architecture.

Ridders' method did not influence the speed or stability as much as hypothesized, but visual inspection of the non-algebraic blobby(c), seems to indicate that Ridders' method produces more accurate roots. This should be examined further with more rigorous accuracy tests.

The implementation itself was fairly naïve and the load would often crash the GPU. Optimization of the code should produce greatly improved program stability.

References

[1] Ridders, C. (1979). "A new algorithm for computing a single root of a real continuous function". *IEEE Transactions on Circuits and Systems* **26**: 979–980. doi:10.1109/TCS.1979.1084580

[2] Jag Mohan Singh, P.J. Narayanan, "Real-Time Ray Tracing of Implicit Surfaces on the GPU," IEEE Transactions on Visualization and Computer Graphics, vol. 99, no. RapidPosts, pp. 261-272, , 2009