CPU vs. GPU Based 3D Visualization of Algebraic Surfaces

Allan Gelman[1] , **Danny Gelman**[1] , Joseph Masterjohn[2], Victor Milenkovic[2], Nancy Newlin[3]
1:Undergraduate, MIT, 2: Department of Computer Science, University of Miami, 3: Undergraduate, University of Florida

Introduction

Manufacturing companies, engineers, and architects rely on modeling software. This software uses algebraic surfaces to visualize the various shapes or structures. Having software that depicts these surfaces as accurately and efficiently as possible is essential for the success of these aforementioned fields.

Our objective is to analyze the pros and cons, in terms of runtime and usability, of the CPU and the GPU approach to visualizing 3D surfaces. The CPU approach calculates on the CPU a set of triangles that approximate the portion of the surface currently in view and sends those triangles to the GPU for display. The GPU allows rapid changes in the position, orientation, and scale of the triangles, but a large change requires a new set of triangles for accuracy and completeness. The GPU approach calculates on the GPU a set of points on the surface corresponding to each pixel. Each change in view requires recomputing the set of points.

Computing triangles on the CPU is time-consuming and cannot be done in parallel. More than a five second delay would make the visualization hard to use. Computing points on the GPU is even more time-consuming but fortunately can be done in parallel. In this case, a delay of more than 0.2 seconds would be objectionable because a frame rate of less than five frames per second makes rotation, etc., too choppy.

Problem Statement

Our goal was to create a software that, in an efficient manner, guarantees the visualization is mathematically correct unless uncertainty is explicitly indicated. The software is to function as a debugging tool to test the accuracy of other algorithms that seek intersections of algebraic surfaces. We will compare the pros and cons of the CPU approach versus the GPU approach of this software.

Numerical Methods

Both methods apply to surfaces defined as the zero set of a trivariate polynomial. Two of the coordinates are substituted, yielding a univariate polynomial. Descartes rule of signs and subdivision yield a set of intervals that contain a single zero. Newton's method shrinks the interval. All calculations use interval arithmetic to ensure correctness. When a sign test is ambiguous in interval arithmetic, the CPU method increases the amount of precision used. Since there is no multiple precision library on the GPU, the GPU just marks the interval as ambiguous. We used an Intel(R) Xeon(R) CPU E5520 @ 2.27GHz and an 8GB 256-Bit GDDR6 1515 MHz NVIDIA(R) card with 2944 CUDA Cores.

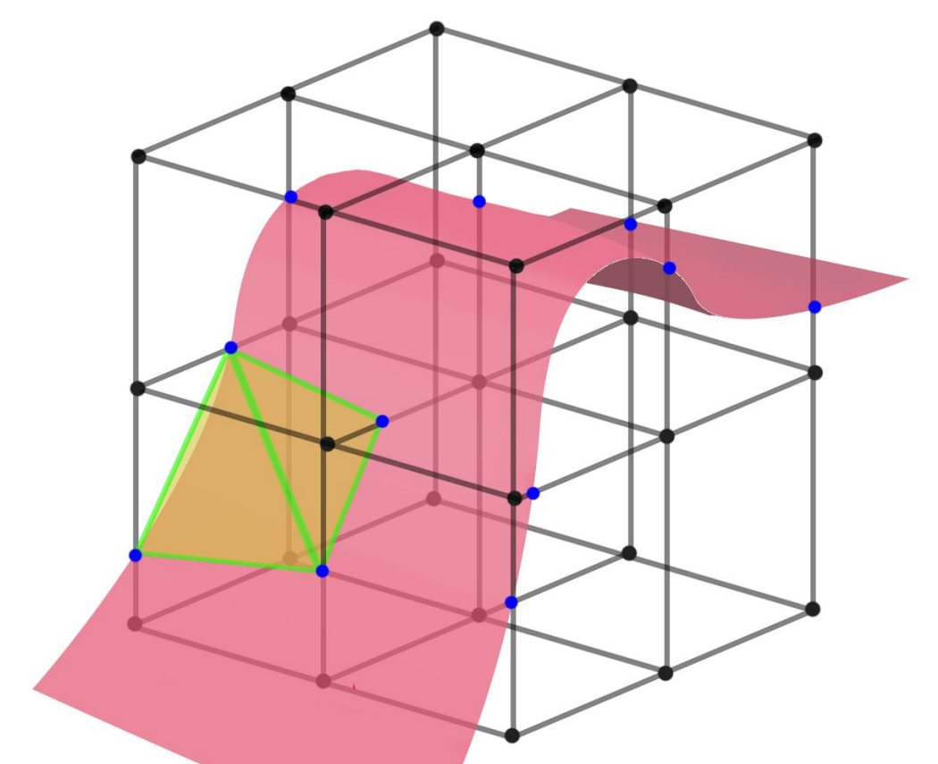
Acknowledgments

Masterjohn and Milenkovic are supported by NSF grant CCF-1526335, and the Gelman's were supported by an REU under that grant. Newlin was supported by an REU under NSF grant CNS-1659144. University of Miami Computer Science department equipment was used.

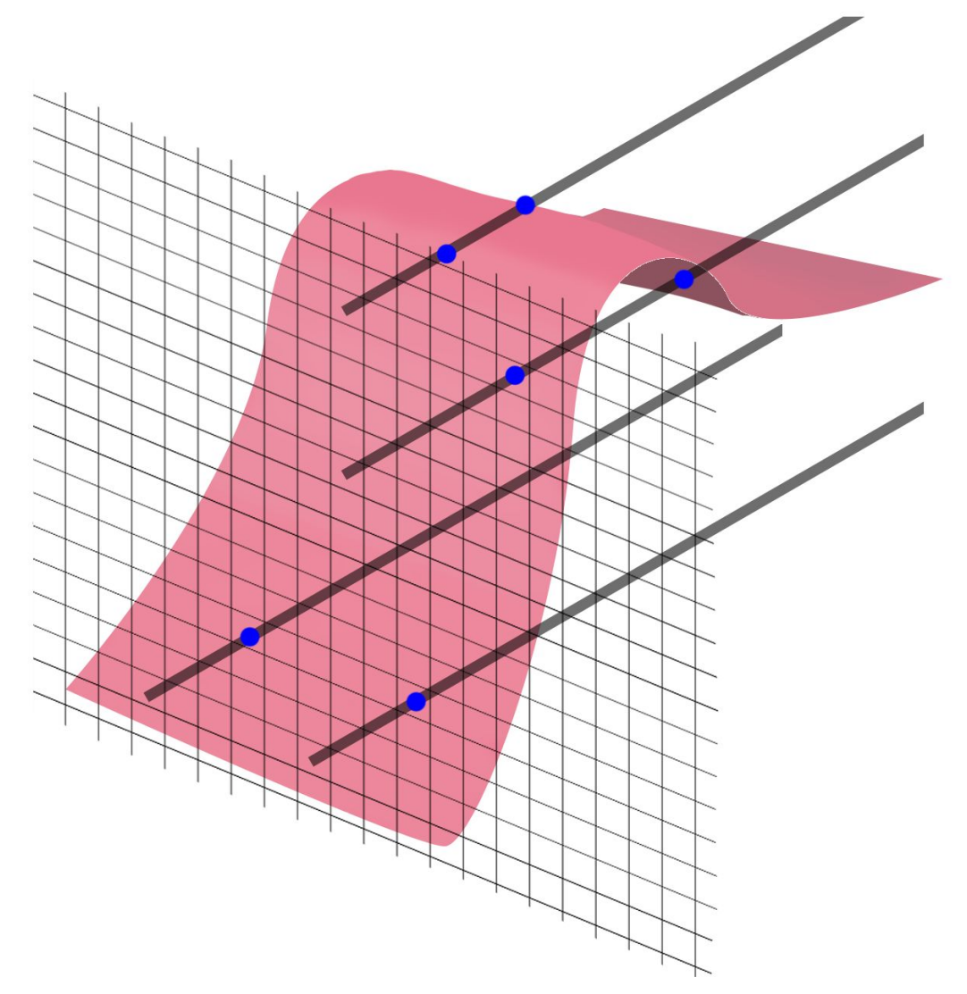
UNIVERSITY OF MIAMI
DEPARTMENT of
COMPUTER SCIENCE

NSF

Algorithms

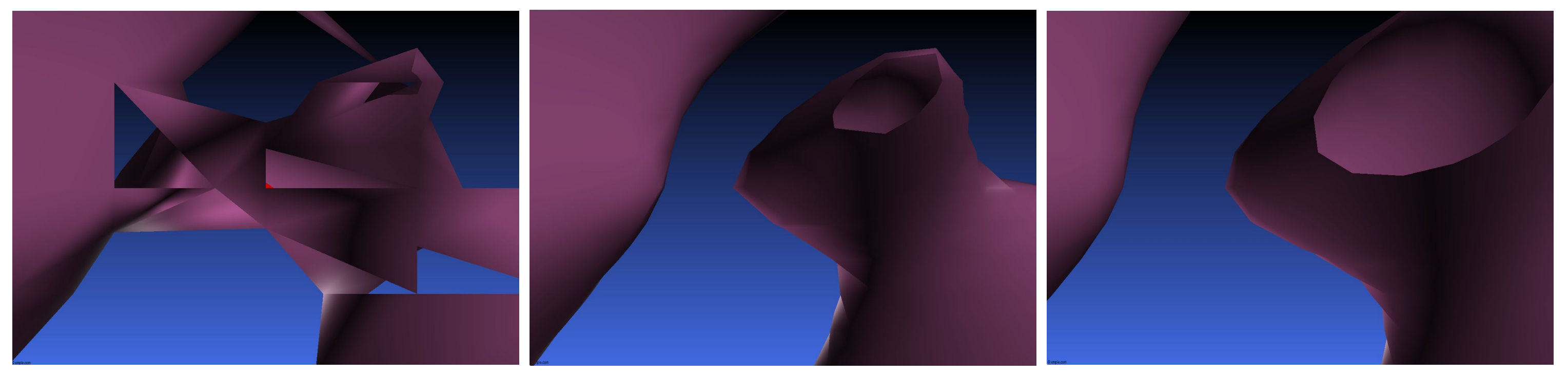


This figure shows how the CPU approach works. First, we find the intersection points between a grid in the viewing volume and a surface, as shown by the blue dots. Then the intersection points are connected together to form various polygons, and the polygons are triangulated. This is shown by the four points connecting to make a quadrilateral, which is then split into two triangles. All the triangles get sent to the GPU, at which point the user can have full control to rotate, scale, or translate this representation of the surface.



This figure shows how the GPU approach works. We calculate intersections between a surface and lines that come from every pixel of the viewing screen that span from z=-1 to z =1. The figure shows such lines for 4 random pixels and the corresponding intersection points, represented by blue dots. The first three intersections coming from every pixel are displayed. Here, you can rotate, scale, or translate the actual surface.

Regridding

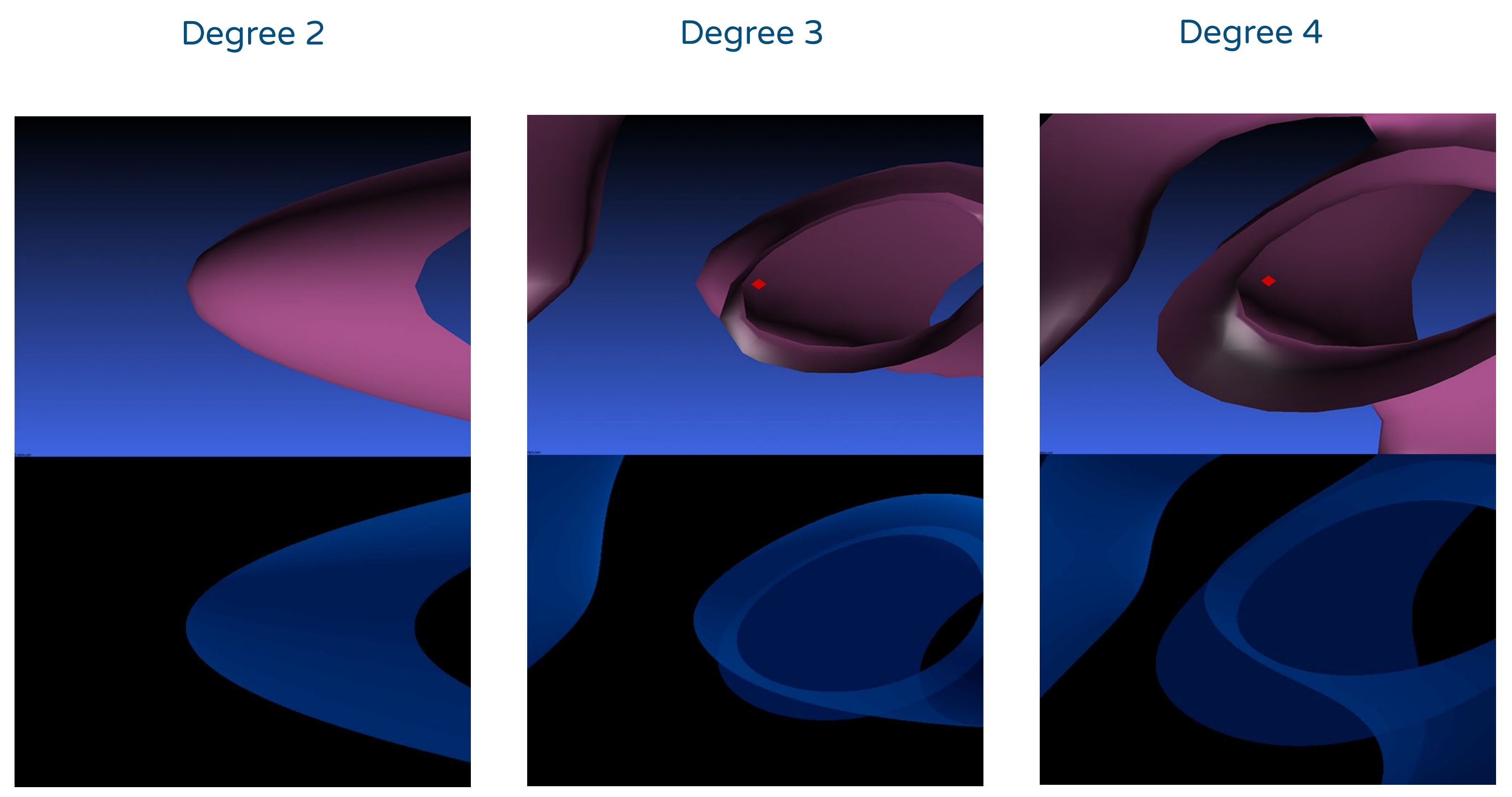


The CPU and GPU approaches are different in the ways they allow the user to view and explore surfaces of interest. The GPU approach allows users to transform the surface and see the part of the surface in the viewing volume in its entirety at all times. However, that is not the case with CPU version. The CPU version displays a subsection of the surface that was in the viewing volume at the time that it was triangulated. So when transforming the surface, the user will not see the edges expand to fill the screen, as one does with the GPU method, until regridding is requested. As the user zooms out and regrids, the finer details of the complex surface may not be accurately represented, as shown in the first image. This happens when the software is unable to properly connect the polygons because there is more than one intersection on an edge of the grid. Zooming in and regridding generates the more accurate versions of the surface, shown in the second and third images.

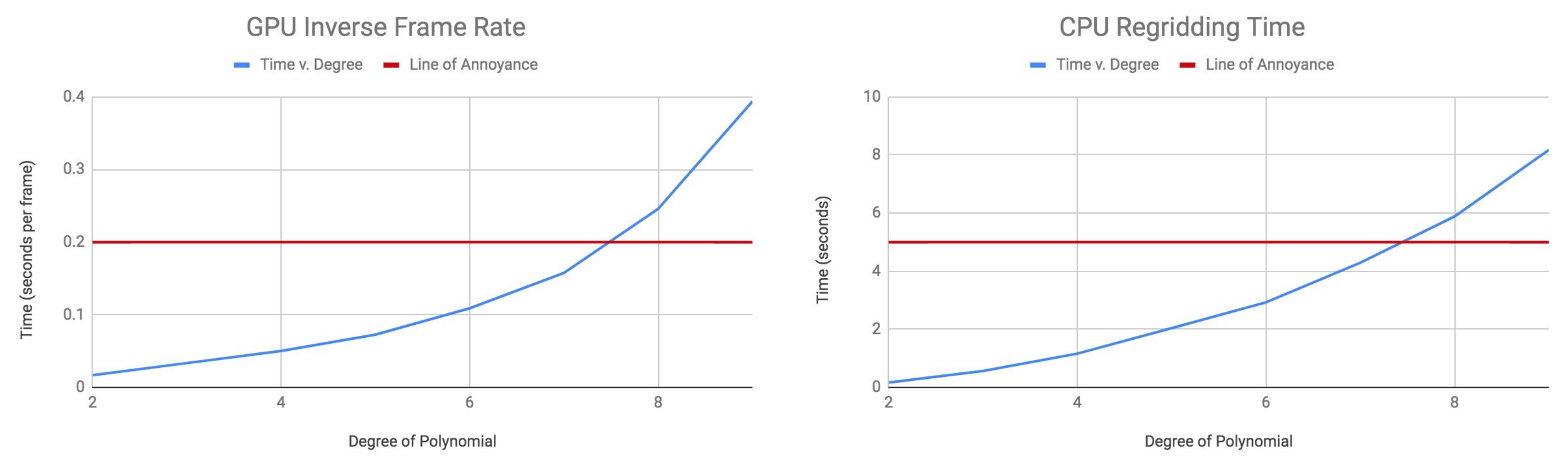
Conclusions and Future Work

Given a good graphics card, the GPU method is as fast as the CPU method and has complete pixel-level accuracy at all times. It also can provide translucency since the order of pixels in z is known. The CPU method is useful when a good graphics card is not available. Future work includes improving the CPU approach to be able to handle surfaces that have more than one intersection per edge of the grid. For both softwares, we want to enhance the user interface by improving the rendering and lighting. This will improve usability by more clearly displaying the surface intersections.

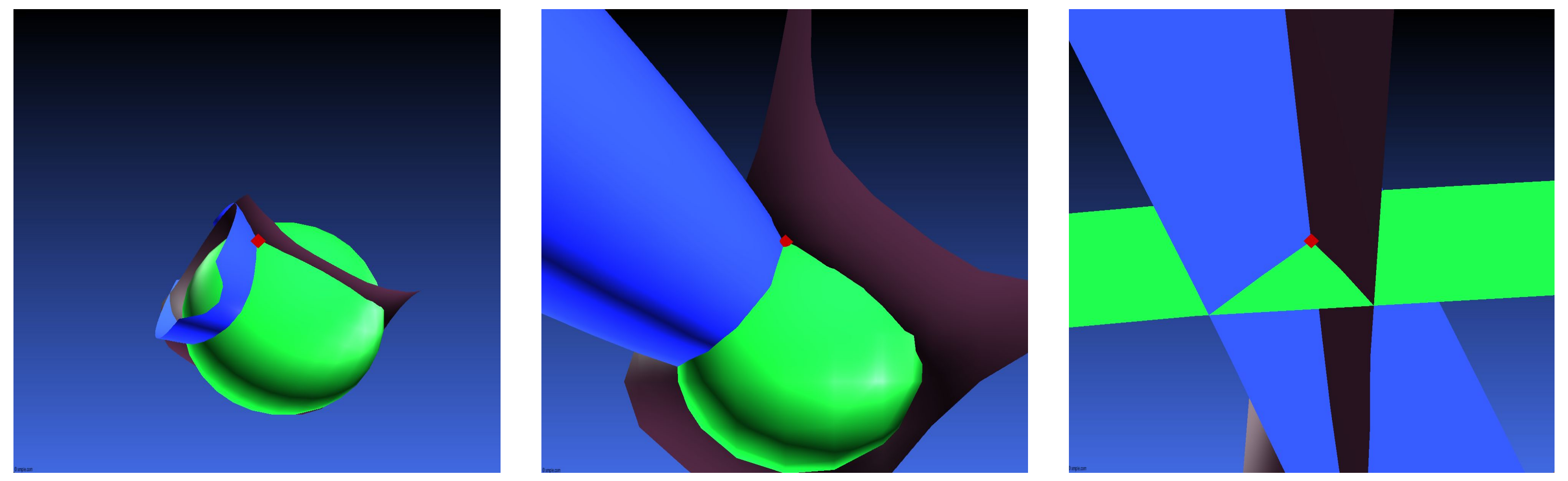
Results



These figures show various polynomials, of increasing degree, displayed in the CPU software (top images) and displayed in the GPU software (bottom images). The red dot represents the origin in the CPU images.



These graphs display how the two approaches' runtime is affected by displaying increasingly complex surfaces. We tested how each software performs when displaying polynomials of degree 2 to degree 9. For the CPU approach, we determined that the threshold at which the runtime becomes an inconvenience to the user, represented by the line of annoyance, is 5 seconds. For the GPU approach, we determined the line of annoyance is at 0.2 seconds per frame. Based on these graphs, it seems that the GPU and CPU approaches both handle displaying polynomials up to degree 7 well. Any higher degree results in a poor user experience.



Both approaches allow the user to view multiple surfaces at once and very closely see their intersection. The first image shows a sphere, paraboloid, and polynomial of degree four intersecting in the CPU version. The second and third images show how the user can progressively zoom in and regrid to get very close to the intersection point. The origin, represented by the red dot, helps the user navigate the surfaces to be able to zoom and regrid in the desired intersection..