# An Analysis of Interactive Deformable Solid Object Modeling

Shrirang Yardi
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
yardi@vt.edu

Benjamin Bishop
Department of Computing Sciences
University of Scranton
Scranton, PA
bishop@cs.uofs.edu

Thomas P. Kelliher
Department of Mathematics and Computer Science
Goucher College
Baltimore, MD
kelliher@bluebird.goucher.edu

***Abstract*— Interactive deformable object modeling is important for a number of application areas. Unfortunately, computational complexity quickly grows beyond the capabilities of consumer systems as model detail increases. Our first contribution is to analyze this application, and present details that might aid in optimizing existing algorithms or in developing new ones. Our second contribution is to provide source code that should be useful in future research.**

## I. INTRODUCTION

Interest exists in deformable object modeling for a number of interactive applications. These applications include: surgical simulation, electronic entertainment, wargame simulation and others. Interactive simulation is difficult because the simulation must be advanced under real-time contraints. Current systems available to consumers offer far less performance than required for simulating detailed objects.

Our research focuses on accelerating these simulations through the use of specialized hardware systems. In order to acheive this goal, we have completed a number of simulations to better understand the computational characteristics of these algorithms[1]. In this paper, we present the results of our simulations. Additionally, we provide open source code for our implementation of the deformable object simulation. Although these algorithms are described elsewhere, we have found implementation to be non-trivial, and we have not found any very general public source code for an existing implementation.

[1]We discuss the rationale for choosing the particular algorithm in later sections

## II. PRIOR WORK

A number of methods exist for representing and animating deformable objects. [1] provides a survey for some of the more mature techniques used in graphics. However, modeling deformation is not unique to graphics. It is intuitive to consider similar techniques as those used to model structural deformation (of bridges for example). However, the requirements of interactive graphics may differ significantly from that of structural engineering. In interactive simulation, we wish to guarantee stability and performance, usually at the expense of accuracy.

It is fairly trivial to develop a deformable object simulation using explicit integration such as the forward Euler method [2]. However, this approach is not well-suited for interactive simulation. If large uniform time steps are used, real-time performance can be assured, but the simulation is very likely to diverge. If an adaptive step-size is used, the simulation may be stable, but we cannot guarantee that we will meet our real-time constraints.

In [3], Baraff et al. describe a simulation method especially suited for cloth simulation. Cloth tends to be difficult to simulate efficiently due to object stiffness against stretch forces. Their approach is to apply implicit integration in order to allow for stability over large time steps. Since the goal is to produce visually pleasing results, accuracy may be sacrificed.

## III. Current Work

In our work[2], we have adopted the algorithms described in [3]. The justification for using these algorithms is supplied in that paper. The model is a series of mass points connected by deformable springs. Applying implicit integration, and converting the system of equations as in [3], we are left with a large sparse linear system. We then apply the Conjugate Gradient method [4] to produce our solution.

In order to analyze the simulation, we must also generate reasonable deformable objects to model. We have produced two models that are intended to be representative of the range of models that one might wish to simulate. The "cloth" model is a simple two dimensional mesh. The "cube" model is generated from a three dimensional uniform grid of points. Each point is connected with springs to the 26 points bordering it (except for boundary points). Models are generated automatically, which allows for an arbitrary level of detail to be supplied by the user.

These models are representative in the sense that they establish bounds on the expected structure of realistic models to be simulated. We expect that there are few useful models that are more simplistic (in terms of spring topology) than our cloth. Even with cloth, we may wish to add springs in order to model forces opposing folding. We expect complex three dimensional objects to behave similarly to our "cube" model. For these objects, an intuitive approach would be to model them as a mass-spring system by fitting the complex object to a three dimensional mesh (similar to the cube). Mesh points contained in the complex object would be retained, others would be discarded. We believe that this approach allows us to approximate the behavior of complex three dimensional objects using our "cube" model. Figures 1(a) and 1(b) show the structure of the "cloth" and "cube" models respectively. Figures 2(a) and 2(b) show the resulting linear system sparsity patterns for the same models.

A relatively simple gravity-driven simulation was used. For the "cloth" simulation, the position of one edge of the cloth was constrained, and no collision detection was applied. For the "cube" simulation, simplistic collision detection relative to a static floor was used. Collisions were resolved using control points – additional points connected to model points by zero-length stiff springs. Sample graphic output from simulating the "cloth" model is presented in Figure 3.

## IV. Results

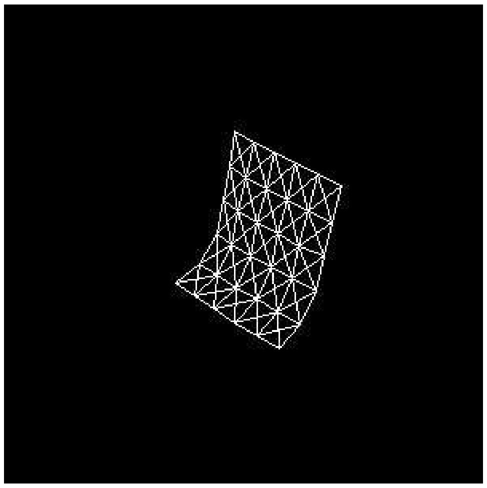
Each of our simulations was run using a range of model complexity. This allows us to draw conclusions about how the simulation complexity scales with object detail.

One of the important factors that we considered was the amount of memory that the computation requires. This is important because delay increases rapidly as additional levels of the memory hierarchy are accessed. Figure 4(a) shows the memory footprint for various models. Figure 4(b) shows the same figure per spring of the model. From figure 4(a) we can see that the growth in memory utilization is highly dependent on the type of model. Our "cube" model grows in complexity much more rapidly than the "cloth" model. However, figure 4(b) shows that memory required is almost constant per spring (with some startup overhead that becomes less important as the number of springs grows).

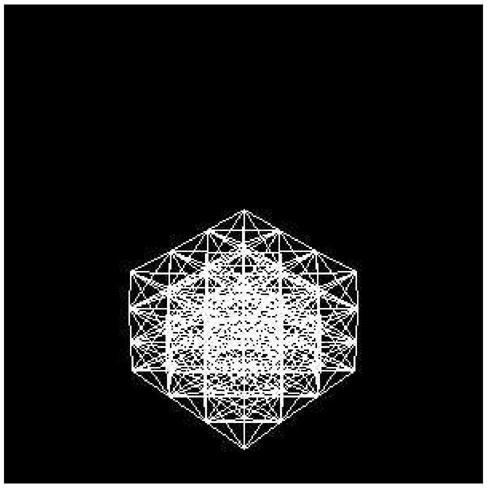
Another important result is the number of floating point operations (FLOPS) required for each iteration of the simulation (which includes many iterations of our iterative linear system solver). Figure 5(a) gives the number of floating point operations per iteration. Figure 5(b) gives the same figure per spring in the model. As with our memory figures, we see that computational complexity is highly dependent on model complexity. The number of floating point operations becomes roughly constant per spring as complexity increases, as shown in figure 5(b).

We have also produced figures showing the breakdown of floating point instructions used in these calculations. However, these figures are based on the Instruction Set Architecture (ISA) for our particular specialized processor. The results may be slightly different for other ISA's. Figure 6 shows these results.

[2]Our source code is available under the BSD License at `http://www.cs.uofs.edu/~bishop/spring.tar.Z`

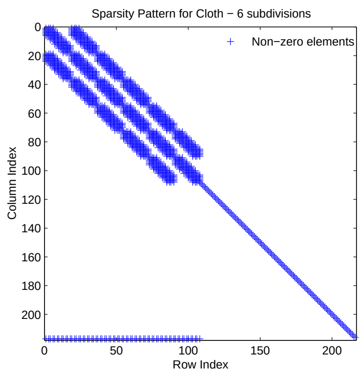


(a) Cloth model with 6 subdivisions (6x6 mesh)

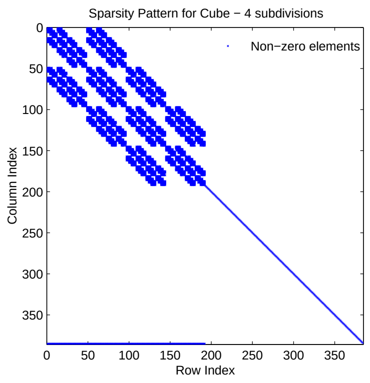


(b) Cube model with 4 subdivisions (4x4x4 mesh)

Fig. 1. Simulated models



(a) Cloth model with 6 subdivisions (6x6 mesh)



(b) Cube model with 4 subdivisions (4x4x4 mesh)

Fig. 2. Sparsity patterns for models

## V. Conclusion

Our results show that the computational complexity (in terms of FLOPS required) of the simulations remains constant per spring. However, computational complexity of the entire simulation grows rapidly with model complexity. This problem is further compounded by the growth in the memory requirements of the simulation as detail increases. As a result, we conclude that executing these simulations interactively will remain impractical unless other approaches are discovered.

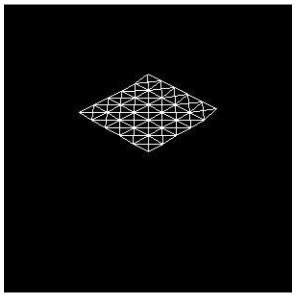
## References

[1] J. Foley, A. van Dam, S. Feiner, J. Hughes, *Computer Graphics Principles and Practice*, Addison-Wesley Publishing Company, pages 1039-1043, 1996.

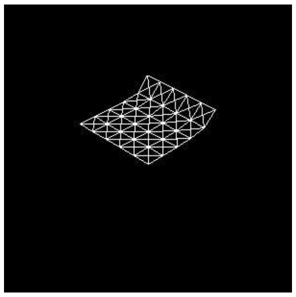
[2] W. Press, S. Teukolsky, W. Vetterling, B. Flannery, *Numerical Recipes in C, 2nd Edition*, Cambridge University Press, page 710, 1992.

[3] D. Baraff, A. Witkin, "Large Steps in Cloth Simulation", *ACM SIGGRAPH, Annual Conference Series*, pages 43-54, 1998.

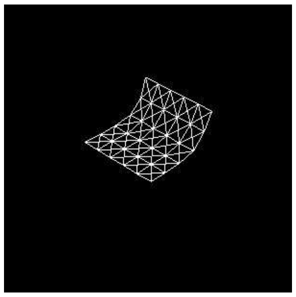
[4] J. Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain", `http://www.cs.cmu.edu/˜quake-papers/painless-conjugate-gradient.pdf`, August 1994.



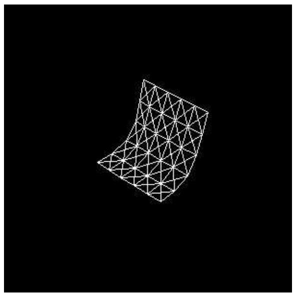
(a) frame 10



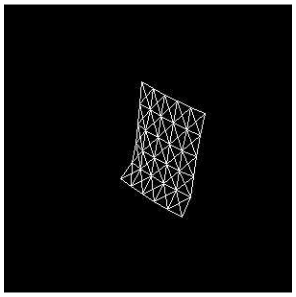
(b) frame 20



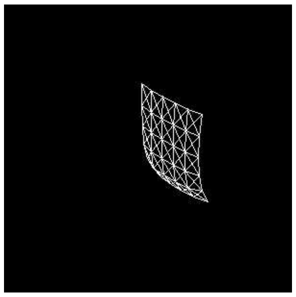
(c) frame 30



(d) frame 40

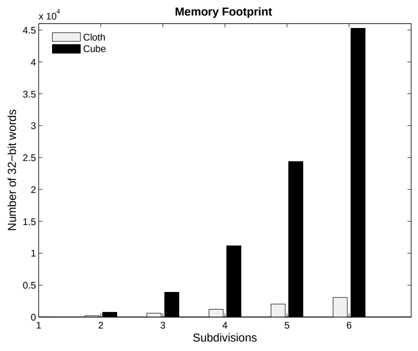


(e) frame 50

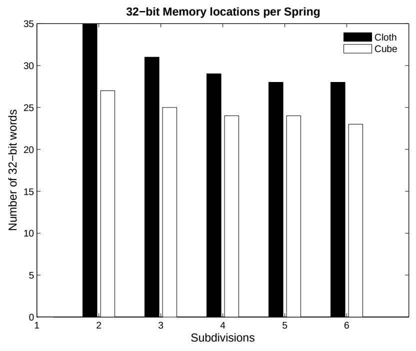


(f) frame 60

Fig. 3. Example simulation

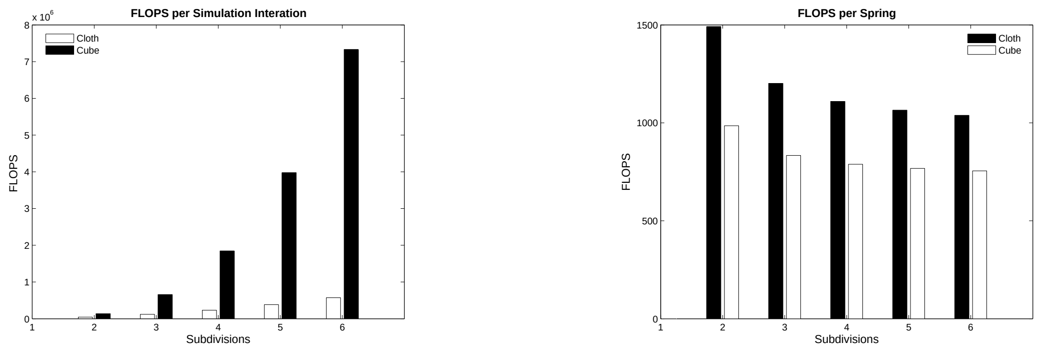


(a) Footprint



(b) Footprint/spring

Fig. 4. Memory utilization



(a) Flops/iteration (b) Flops/iteration (per spring)

Fig. 5. Floating point operations

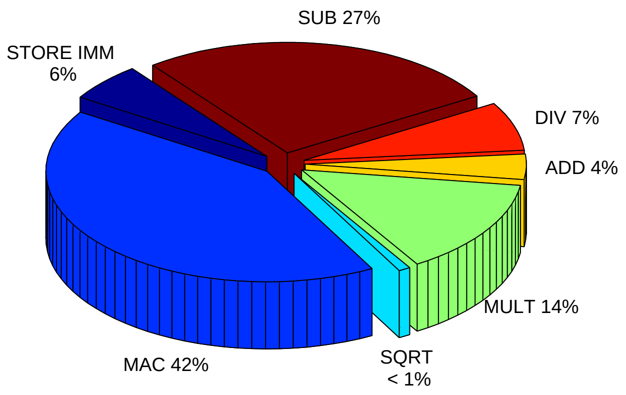


Fig. 6. Instruction Mix