Hannah Norman
SUNet ID: hnorman
Professor Fatahalian
CS 348K – Visual Computing Systems
6 June 2024

DyDef: A Visualization Tool for Dynamic Deformations on Shell Objects

Background

In order to fully appreciate the technical significance of my project, it's first important to understand what Finite Element Analysis (FEA) is. In the simplest of terms, FEA approximates solutions to complex problems by breaking down a large system into smaller, more manageable parts. These parts are called *finite elements*. FEA is most applicable in the engineering world across a variety of domains, whether it be structural, biomedical, aerospace, or mechanical engineering among many other disciplines. The technique utilizes a numerical method called the Finite Element Method (FEM) to solve these complex systems. FEM involves five main steps:

1. Take a geometric representation of your object—for example, a CAD model.
2. Define material properties (e.g., material density, Poisson's ratio, Young's modulus) and boundary conditions of your model.
3. Create a mesh of your model and discretize it into finite elements.
4. Solve the complex system of equations, typically via some sort of solver. In this step, forces are applied and values like displacements and strains/stresses are calculated.
5. Apply post-processing effects, such as Von Mises stress heatmaps, to the deformed model.

With this background in place, I will move onto the setup of my project.

Problem Setup

My project builds upon an existing software library called DeformFX and an accompanying application called HalfPipe. DeformFX is a highly parallel software library for nonlinear finite element simulation. HalfPipe is an application that utilizes DeformFX to simulate and render a half cylinder with a point force applied at the center-end. See Figure 1 (on the following page) for a screenshot taken from the original HalfPipe application.

The problem at hand is that the DeformFX library does not provide sufficient visualization tools. The inputs to HalfPipe are in the pre-existing codebase; the application takes a mesh with specified material properties and boundary conditions alongside defined forces and a force target. For output, HalfPipe displays a simulation run of what occurs when the forces are applied to the mesh at the given target. The original simulation runs in the HalfPipe application window over the course of 30 seconds at around 12 frames per second (fps). Once the simulation completes, the application must be completely restarted to run the simulation again. During the simulation, plenty of deficiencies become apparent, too. Most visibly, the mesh exhibits aliasing artifacts due to a lack of depth testing on the deformed mash. The remaining deficiencies encompass the features HalfPipe lacks. In particular, there is no way to interact with the mesh, there are no quantitative statistics displayed on screen for simulation analysis, and there are no available post-processing features, such as a Von Mises stress heatmap, as is typical of most FEA simulations.

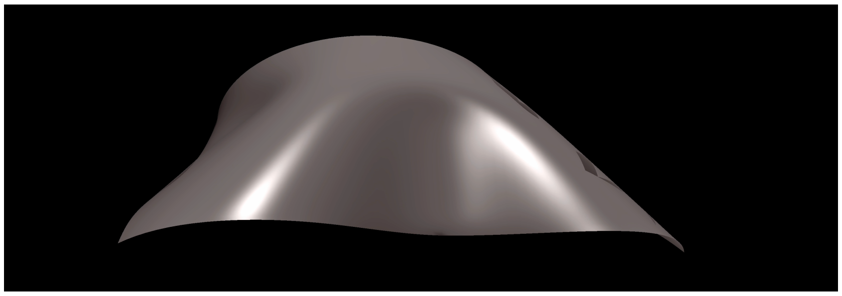


Figure 1. Original HalfPipe simulation run, about halfway through the deformation.

With so many unignorable deficiencies apparent, the goals for my project became clear quickly. Primarily, I wanted to create a tool that accurately visualizes and simulates dynamic deformations on this halfpipe shell while simultaneously providing useful information to the user. I had two users in mind: myself, the novice, who is new to FEA and is seeking an application that helps me better understand deformations; and the expert, such as my boss Justin Voo who initially developed this framework, and who is seeking an application that captures the critical elements of FEA, such as deformation analysis and post-processing abilities. In fact, since HalfPipe is a closed-source codebase and the only other eyes on it are Mr. Voo's, he helped me lay out four goals:

1. Get rid of graphics errors.
2. Display helpful quantitative statistics during runtime.
3. Provide some sort of pseudo real-time mesh interaction.
4. Add common FEA post-processing visualizations.

Alongside these goals, I also outlined several constraints with Mr. Voo's help.

1. HalfPipe must run at no less than its current framerate (12fps) and preferably faster.
2. The GUI should be intuitive and easy enough to use for both novices and experts.
3. Features must not be overly complex due to a lack of experience on my end with Objective-C and Apple's Metal API. This project marks my first foray into both.

Given this setup of goals and constraints, the crux of my problem became how to optimize HalfPipe to improve its visualization capabilities without sacrificing its current runtime efficiency.

Approach

As mentioned, I started with the existing HalfPipe codebase, as provided by Mr. Voo. The GitHub page for this project can be seen here: https://github.com/hnorm0629/cs348k-final-project. I will be turning it back to private after grades have been published for this class.

The visualization goals for my project aligned with a GUI, and thus DyDef was born. In order to implement DyDef, I first converted my goals into feasible, implementable features. This would provide structure for my project and give me an idea of when I crossed the figurative finish line. To begin, I wanted to clean up the mesh visualization, which involved implementing a depth buffer and integrating it into the rendering pipeline. With that out of the way, I moved onto the true GUI features. First, I surveyed the code base and learned that each frame of the simulation produced unique solution and error values, which can be important in FEA since low error and high solution indicate an accurate mesh deformation. I also determined I wanted to add total elapsed time and frame rate to the interface so the user can track progress throughout the simulation. Furthermore, and this was my vaguest goal, I wanted some way to interact with the mesh in a real-time manner. I decided that a slider which allowed scrubbing through the entire simulation, from start to finish, would best achieve this goal. In addition, I added gesture recognizers to the application to allow the user to visually interact with the mesh's position and orientation via pinching and rotating their fingers on a laptop trackpad. Finally, I knew I wanted to add a Von Mises stress heatmap to the GUI, and I decided a button toggle would maximize user control. During implementation, I decided I would also add another heatmap toggle for additional information, this time regarding nodal displacements of the mesh throughout the deformation.

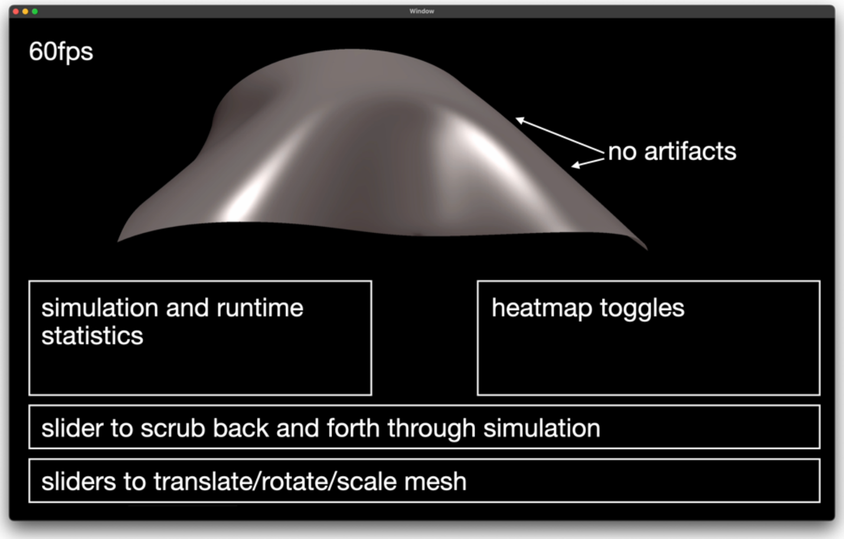

Window
60fps
no artifacts
simulation and runtime statistics
heatmap toggles
slider to scrub back and forth through simulation
sliders to translate/rotate/scale mesh

Figure 2. Early-stage implementation plan. Most of these features made it into the final GUI.

As for low-level methodology, I focused on three main files: SurfaceRenderer, where frame-by-frame drawing occurs; ViewController, where window customization occurs; and Model, where strain calculation occurs. Also, to support my pseudo real-time simulation scrubbing feature, I implemented the FrameData class, which encapsulates and tracks simulation data for each frame. Across 30 seconds of simulation, there are about 360 frames total. Since the mesh itself wasn't overly complex—it comprises around 5100 vertices—space constraints were not a large issue during implementation, though I still intended to prioritize space efficiency. Ultimately, for each frame, FrameData stores current rendering vertices, index data, stress calculations per vertex as well as runtime, frames per second, error, and solution data. When pairing the index of the FrameData array with the value on the scrubbing slider, HalfPipe gains the ability—after the initial simulation run where the data is stored—to return to any point during the simulation, while maintaining current mesh position and orientation as determined by the user.

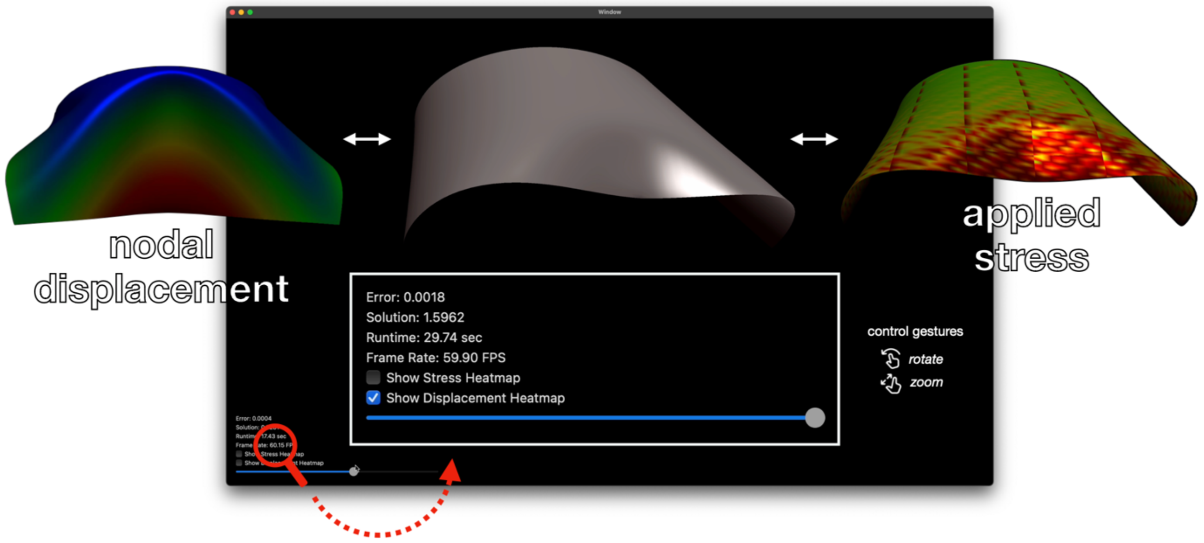
Results

I implemented each of the features outlined in my approach, and I arrived at the current iteration of DyDef. Figure 3 below highlights key features of the final application, but for a more comprehensive video demo, please see the following link: https://drive.google.com/file/d/1vd0rS6lsXAV9DAzWbj2S-spYk_C1OSi_/view?usp=sharing.

Evaluation

When planning out DyDef, I defined an overarching goal for myself: to create a tool that accurately visualizes and simulates dynamic deformations on this halfpipe shell while simultaneously providing useful information to the user. More than that, I wanted DyDef to be *good* tool, but how do I define good?

The goals I laid out for my project were four-fold and feature-focused. I wanted to a) remove rendering artifacts, b) add quantitative statistics to the application window, c) support some sort of real-time interaction with the mesh, and d) add post-processing visualizations. These are easily checked off by watching the final demo. For ease, Figure 3 below summarizes these features.



```
Error: 0.0018
Solution: 1.5962
Runtime: 29.74 sec
Frame Rate: 59.90 FPS
☐ Show Stress Heatmap
☑ Show Displacement Heatmap
```

Figure 3. Summary of DyDef features, including control gestures, quantitative statistics, two heatmap toggles, and simulation scrubbing slider.

I was able to implement every feature I had planned. Some bugs remain in the Von Mises stress heatmap (note that each finite element is visible due to a red stress pattern on their left borders), but it still effectively highlights applied stress throughout the deformation simulation. Still, success goes beyond checking off the implementation features. I returned to the question, *how do I determine if DyDef is a good tool*, and I offer three evaluative questions below:

1. Does the DyDef GUI effectively help users better understand the deformation simulations?
2. Is the interface intuitive enough to be used by both novice's and experts familiar with FEA?
3. Does the GUI run at least as efficiently as the original simulation on standard hardware? Does it improve efficiency at any point?

As to the first question, I believe it does. The slider, in particular, allows the user to scrub back to any point in the simulation and analyze what the mesh looked like, what the solution and error values were, and visualize displacements and Von Mises stresses. As opposed to the simple, un-interactive, start-to-finish simulation HalfPipe was originally, these features encourage and enable much more analysis of the deformation and can lead to users better understanding the simulation at hand. Regarding the second question, I do think the interface is intuitive, at least from a novice's perspective. Toggles, sliders, and trackpad gestures are common to many applications, though in retrospect, perhaps further labeling could clarify their use. Finally, for the third question, I was unable to achieve real-time execution during the initial simulation, though it never drops below the original 12fps. But as evidenced in Figure 3, once the slider and other control features become accessible to the user, the framerate increases 5x to 60fps, thereby achieving the real-time interaction goal I set out initially. By these standards and self-evaluation, I achieved success.

To round out this evaluation and expand beyond my own viewpoint, I sent my demo to Mr. Voo along with the three evaluative questions I provided above. DyDef represents somewhat of a challenge to effectively evaluate since the codebase is closed-source, but since Mr. Voo is the creator of HalfPipe and DeformFX, I thought he could offer valuable feedback. I will include his responses below, verbatim. Regarding the three questions:

1. *Does the DyDef GUI effectively help users better understand the deformation simulations?*
    a. "Yes. The ability to move forward and backward while changing the visualization overlays certainly helps provide new insights to the mechanics of the deformation which wouldn't be there before."
2. *Is the interface intuitive enough to be used by both novice's and experts familiar with FEA?*
    a. "Yes. Visualizing stresses in this manner (i.e. the Von Mises stress) is pretty standard and I think pretty intuitive in this GUI."
3. *Does the GUI run at least as efficiently as the original simulation on standard hardware? Does it improve efficiency at any point?*
    a. "Certainly, the recording/replay is going to render faster so, in that sense, it is indeed a performance increase."

In sum, he says, "I think you can claim you met your initial goals." He continues, "I think the most useful result of this is the ability to move forward and backwards through the simulation while changing the visualization; I think that can really give a user some insight into how the variables change during deformation. If that were the primary result of this work, I'd call it a success." Finally, regarding the runtime efficiency, Mr. Voo adds, "The 5x performance increase in the replay is totally valid, I think. The purpose is to help a user visualize the changes in the stress over the course of a given motion. This performance improvement does that, regardless of whether it happens in the simulation itself or in the post-processing."

In terms of future work, Mr. Voo also offered some thoughts: "Porting the simulation to AMX (on Mac) would probably have yielded some concrete performance improvements in the simulation itself. But, given your time constraints, I don't know if it would have been reasonable to do that alongside the GUI additions." I agree with this. When designing this project, I had considered optimizing current performance instead, but ultimately, I decided to go down the GUI path. Over the summer, perhaps, I can re-approach DyDef from a performance perspective and port to AMX next.

Team Responsibilities

N/A. I worked on this project individually, so I implemented each feature myself. Although, I would like to give fair acknowledgement and thanks to my boss, Justin Voo, who provided me with the HalfPipe application codebase, the accompanying DeformFX software library, and plenty of advice, support, and feedback during the development process.

References

Objective-C Documentation:
https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/
Metal By Example: https://metalbyexample.com/the-book/
AppKit Framework Documentation: https://developer.apple.com/documentation/appkit
"The Finite Element Method (FEM) – A Beginner's Guide":
https://www.jousefmurad.com/fem/the-finite-element-method-beginners-guide/
"Inverting Hooke's Law": https://www.finiteelements.org/hookeslaw.html
"Von Mises Stress": https://www.continuummechanics.org/vonmisesstress.html