# Thick Origami Kinematics Simulator

Jacob Pavelka, Anvay Pradhan, Noah Zambrano

November 2023

## Background and Motivation

### Introduction

Kinematic Simulation is an important tool for many problems. For linkage systems, it allows one to study the degrees of freedom of a system or its range of motion. For computer graphics, simulating kinematics is important for virtual reality training, video games, rapid digital prototyping, and robotics simulation [1]. For structural systems it allows one to study whether or not a system is stable. For example, understanding the kinematic properties of thick origami is important if it is used structurally. A kinematic simulation would show which forces would cause the origami to deform without resistance [2].

*For our project, we want to develop a kinematic simulation package for thick, rigid origami so we can become more familiar with kinematic simulation methods and develop performant software that others can use.* To test our package, we will simulate kinematic folding motions for a simple hinge made of two objects, a Miura fold, and a cube folded from flat. This project will incorporate course themes including scripting, linear algebra and libraries, object-oriented programming (OOP), workflows, and unit testing.

### Background

Add github issues background here.

The kinematics of an origami object can be simulated by first defining the origami's mesh based on pre-defined nodes, then imposing constraints on those nodes' degrees of freedom based on the defined mesh, and finally, solving for the mesh's kinematically admissible motion based on the constraints [4].

To simulate the kinematics of an object, it first needs to be represented mathematically, wich can be done by creating a mesh of the object. There are many methods for creating meshes, but we will use the one described in [3] because it creates a mesh of edges and nodes that is automatically stable.

This algorithm begins with a list of nodes that describe the important features of a convex hull, such as any corners. From here, four nodes are chosen randomly such that they create a tetrahedron, and links are defined between them. Then, another point is chosen randomly and connected to three existing points such that they all form another tetrahedron. Links are created between the three existing nodes and the new node. This process is repeated for all remaining nodes. Figure **??** illustrates this. This process will create the mesh for one rigid object, but to have objects that can develop complex motion, one or more objects will to be connected. This will be taken care of in later steps.

Once all points are connected, a connectivity matrix can be formed, and from this, the constraints can be formed. The general constraint that will be made for each connection is shown in Eq. 1.

$$l^2 = (\vec{x_i} - \vec{x_j}) \cdot (\vec{x_i} - \vec{x_j}) \tag{1}$$

Where $x_i$ and $x_j$ are the position vectors of connected nodes, and $l$ is the original distance between the two nodes.

Implementing this constraint ensures the distance between nodes remains constant to enforce rigid body kinematics. Compiling these constraints into one matrix and linearizing them with respect to time, a new relationship is defined in Eq. 2. When this matrix is compiled, adjacent nodes of connected bodies should also be constrained by Eq. 1

$$0 = C\dot{x} \tag{2}$$

Where $\dot{x}$ is a vector of each degree of freedom's velocity.

This equation can be solved, but it has multiple solutions. A better approach to finding kinematically admissible velocities is to project a trial velocity vector onto the null space of the constraint matrix as shown in Eq. 3.

$$\dot{x} = [I - C^{+}C]\dot{x}_0 \tag{3}$$

Ultimately, the goal of the simulator is to find the new node locations $x_{s+1}$ for each timestep in terms of the original node locations $x_s$ under applied loads. To do so, we can represent the new node locations in the following manner:

$$x_{s+1} = x_s + dx_s \tag{4}$$

where $dx_s$ is a vector of constraints linearized with respect to time to produce infinitesimal admissible motions (see Eq. 5). These can be integrated over time to achieve the full motion. Many methods may be used to integrate the velocities and higher-order terms over time, but we will use Eq. 5 in our implementation unless another is warranted.

$$dx_s \approx \dot{x}dt + \ddot{x}\frac{dt^k}{k!} \tag{5}$$

# Work Performed

In this project, we developed a library using other libraries and C++ in an object oriented manner. We used github to collaborate. We used Cmake to make builidng our project easier. We use programming development concepts with the help of unified modeling language to help us desing the library. This section will outline the work that was done for each of these tasks.

## Library Planning and Design

Collaboration is very beneficial but it takes a lot of planning upfront to implement it. To start out the project, we needed a clear idea of the problem we were trying to solve. From that problem, we developed high-level requirements the developed software should have, and then the functional requirements of the software. With these ideas in place, we were poised to have more efficient collaboration in the future.

Equally giving out work is another difficult aspect of collaboration. This is where github became handy. Using github, we were able to make well defined issues to define a finite amount of work that had a clear objective and relation back to the high level and functional requirements of the project. These clear issues helped our team members work on their tasks efficently. Github also has features that help with project management. Specifically, the project timeline feature helped keep all our task on-time. Jacob was the one repsonsible for setting up and mainainting the github repository, so when tasks were coming close to due, he would message the other team members as a reminder.

## Results and Discussion

What did we accomplish

## Conclusion

How is what we accomplished meaninful. Did it achieve the goals we set out for in the introduction?

To develop this simulation software, we will incorporate github for collaboration, cmake to help build the project, and a waterfall development workflow, where we will define the requirements of the software, then its architecture, through the rest of the steps, and finish with testing. The architecture will be designed with the Object Oriented Programming paradigm and the unified modeling language. The actual software will be developed using C++ and will integrate any performant linear algebra or scientific computing packages needed (blas,lapack,petsc). Testing, visualization, and example cases will be developed for this software too, and the example cases will be ran on Great Lakes.

Table 1: Tasks with Person In Charge, Descriptions, and Deliverables

| # | Task/Topic | PIC | Description | Deliverable |
|---|---|---|---|---|
| 1 | Github (Workflows) | Jacob | Update issues and review tasks, push code | Completed and reviewed tasks, repo link |
| 2 | Code Architecture (C++ OOP) | Jacob | Develop code structure by defining classes | UML diagram |
| 3 | Set up project file structure and compilation tools (CMAKE, Tools of the Trade) | Anvay | Define a storage system for the elements of the software and begin making the tools to configure,build, and install the software, where the remaining portions of the cmake tools will be defined as the code is developed | Project File structure developed and initial programs compile |
| 4 | Integrate Linear Algebra or Scientific Computing Libraries (Linear Algebra and Libraries) | Noah | Link linear algebra library (e.g. openBLAS) to aid in linear algebra operations, specifically the matrix solve and matrix-operations | Functioning code that successfully utilizes library |
| 5 | Meshing Implementation (C++ OOP) | Jacob | Write code to generate mesh from input file containing points of interest on convex hull. Update CMAKE, github with changes. | Functioning code that passes tests |
| 6 | Define Constraints (C++ OOP) | Noah | Implement methods to generate the constraint matrix. Update CMAKE, github with changes. | Functioning code that passes tests |
| 7 | Kinematics Implementation (C++ OOP) | Anvay | Implement methods to compute kinematically admissible deformations. Update CMAKE, github with changes. | Functioning code that passes tests |
| 8 | Unit Testing, Verification Testing, and Examples (Workflows/C++ OOP/-Great lakes) | Noah | Write tests for each method, run required tests and example cases on Great Lakes, and confirm results are correct | At least one test is defined for every method and they all pass |
| 9 | Visualization (Scripting) | Anvay | Implement a python script to produce animations of the simulation | Video files that can be included in our presentation |
| 10 | Presentation | All | Write slides for presentation | Powerpoint file |
| 11 | Report | All | Write sections of report | Completed report PDF and links to repo |

# References


[1] Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH 2022 Courses*, SIGGRAPH '22, New York, NY, USA, 2022. Association for Computing Machinery.

[2] Evgueni T Filipov, Tomohiro Tachi, and Glaucio H Paulino. Toward optimization of stiffness and flexibility of rigid, flat-foldable origami structures. In *The 6th International Meeting on Origami in Science, Mathematics and Education*, page 121, 2015.

[3] Tianhao Zhang and Ken'ichi Kawaguchi. Folding analysis for thick origami with kinematic frame models concerning gravity. *Automation in Construction*, 127:103691, 2021.

[4] Yi Zhu, Mark Schenk, and Evgueni T Filipov. A review on origami simulations: From kinematics, to mechanics, toward multiphysics. *Applied Mechanics Reviews*, 74(3):030801, 2022.