Syzygy: A Physics Simulation Language and Visualizer
CSDS 395 Final Report
Spring 2025
Hayden Caldwell, Callum Curtis, Jacob Leider, and Jonas Muhlenkamp

Project Summary
Syzygy is a general-use particle simulation software designed to take user input via a simple, custom-built programming language. By using easy-to-learn commands to create particles and forces, this software will fill a gap in the available simulation systems, making complex simulation more accessible to interested amateurs and helping unify the diverse set of systems used in technical research. Syzygy will provide users with a wide array of options for customizing and visualizing their simulated environment.

Background
Physics simulation is an important tool both for education and research. Simulations allow students to better visualize the physical concepts that they learn about in class, helping to firmly engrave them in their minds, while also serving as proof-of-concept for many advanced research applications in astronomy, particle physics, biophysics, and other fields. Current options for such simulators, however, are either primarily targeted toward the video game industry (e.g. Unity) or custom-built for single use cases within academia. Furthermore, these simulators either use a complex user interface for configuring systems or require their users to learn fully-fledged programming languages like C# or Python. This presents an unnecessarily steep barrier of entry for novices interested in creating their own physics simulations, whether for teaching others or for personal investigation.

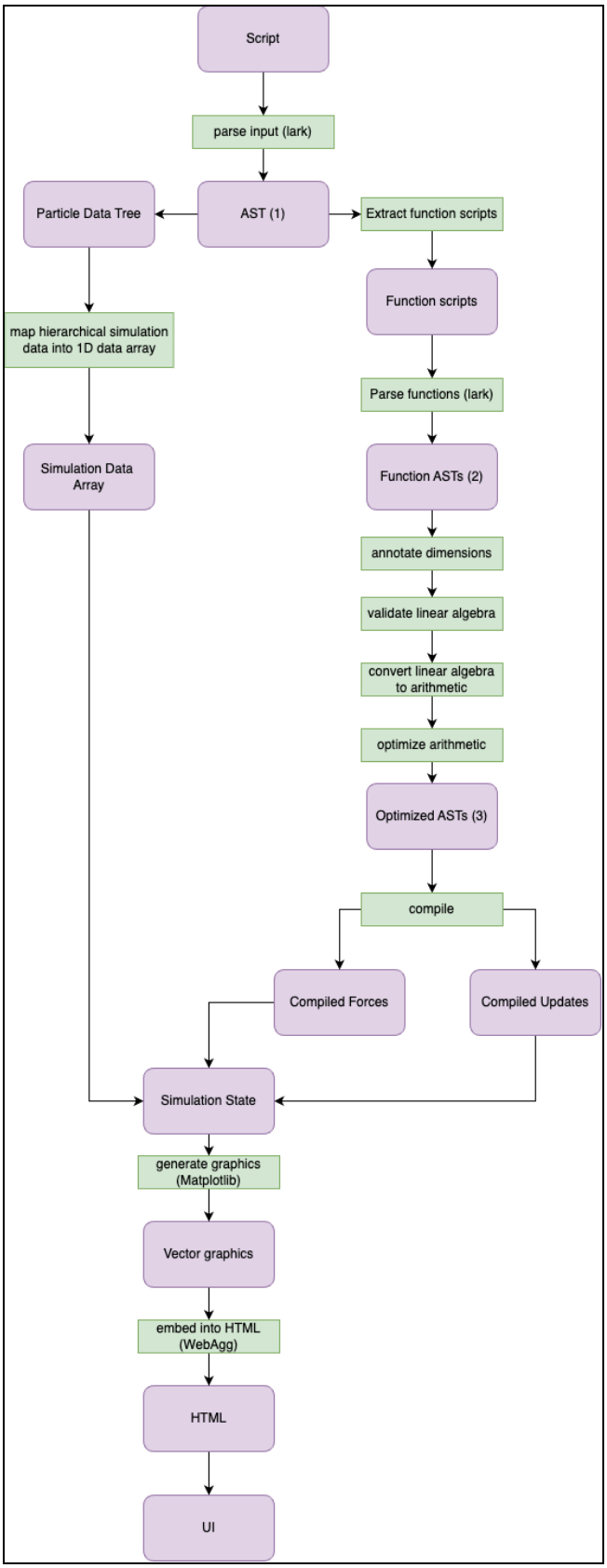
The fact that many physics simulations are custom-built for specific use cases leaves a hole that asks to be filled - why is there not an industry standard package or software for these simulations? By creating such a system and streamlining it compared to existing video game physics engines, we can a) increase educational access to advanced physics concepts and b) successfully fill this need for a general-use simulator.

The system of Syzygy uses a highly focused custom language to create structures within the simulated environment. The user will be able to implement their own custom point particles and force fields within the environment, all of which will be systematically updated over time in whatever way the user desires.

Technical Specifications

As a complete application, Syzygy has several necessary components. Aside from implementing each component, a challenge lies in transferring data between components, and designing efficient interfaces when components communicate. Components are enumerated below, alongside a brief summary of their requirements.

Application Component	Component Specifications
Scripting Language	● Employs a quasi-functional language paradigm ● Clear, concise syntax ● Balance of high applicability with low complexity
Language Parser	● Extended Backus-Naur Form defined for the language ● Parsing package Lark creates a syntax tree according to EBNF
Syntax Tree Input Converter	● Transfers the data stored in the syntax tree output by the parser to data structures maintained by the system solver ● Maps particles and their properties to array offsets in the simulations global data array
Force/Update Rule Compiler	● Compile parsed user defined functions (ASTs at this point) into executable code
System Solver	● Iteratively solves a system of differential equations governing an n-body, m-force simulation ● Outputs intermittent updates of the system's state
Graphics Renderer	● Redraws the simulation from data output by the system solver ● Offers 2D and 3D visuals



The final processing pipeline for Syzygy.

Objectives
Our product software has three main components: A language, a system solver, and a renderer. The goal of this language is to appeal to a wide audience of hobbyists and recreational users who may not have experience with common programming languages.

Technical Requirements and Dependencies
For the entire project, we are using Python 3, due to its widespread support, robust standard library, and powerful string manipulation capabilities. For transferring information from the language interpreter to the system solver, we are using JSON files. For rendering, we are using the Python package matplotlib for its 3-D animation capabilities. This information is laid out in the table below.

Application Component	Planned Dependencies
Language Parser	• Python, powered by Lark
Information Transfer	• Abstract Syntax Tree (AST) data structure within Python
Syntax Tree-to-Solver Input Converter	• Python
System Solver	• Python
Graphics Renderer	• Matplotlib
User Interface	• Webagg

Syntax
The syntax for the language is straightforward. Points, forces, and the update rules for them will all be created as single functions, which can be spread over as many lines as desired for readability, and look as follows:

```
point(pos=[X, Y, Z], vel=[Vx, Vy, Vz], acc=[Ax, Ay, Az], mass=M, charge=E,
        name=”Point”)
force(name=”Force”, in=[p, q], *out=r, func=”p + q”) *out parameter is optional
update(name=”Update”, in=[p, q], out=r, func=”p + q”)
```

Parser Functionality Summary
The parser uses the Python package Lark to process the Syzygy script. It converts the script function calls into objects within an abstract syntax tree (AST) by following a specifically defined Extended Backus-Naur Form that the syntax above adheres to.

Functions of forces and update rules, written as strings in the syntax, are indicated by their appropriate parameter label (‘func’). These parameters are split out from the rest of the syntax

processing as they need an additional step to be verified and configured into a state that can be conveniently used by the simulation state. The functions are processed independently by Lark into their own independent ASTs, at which point they are algebraically verified and mathematical functions like dot(a, b) and norm(a) are expanded into computer-interpretable algebra structures. The final ASTs for functions and points are then ultimately compiled into the simulation state.

Simulation State (SimState) Manager Summary

To facilitate the simulation, the simulation data needs to be mapped back and forth from a global data array, the functions (forces and updates) need to be mapped to the properties that are updated (net_force in the case of forces), and the simulation needs to be "stepped" by computing all forces, then updating the property values of all affected particles. The "DataLayout," "FuncHandler," and "SimState" classes deal with these requirements respectively.

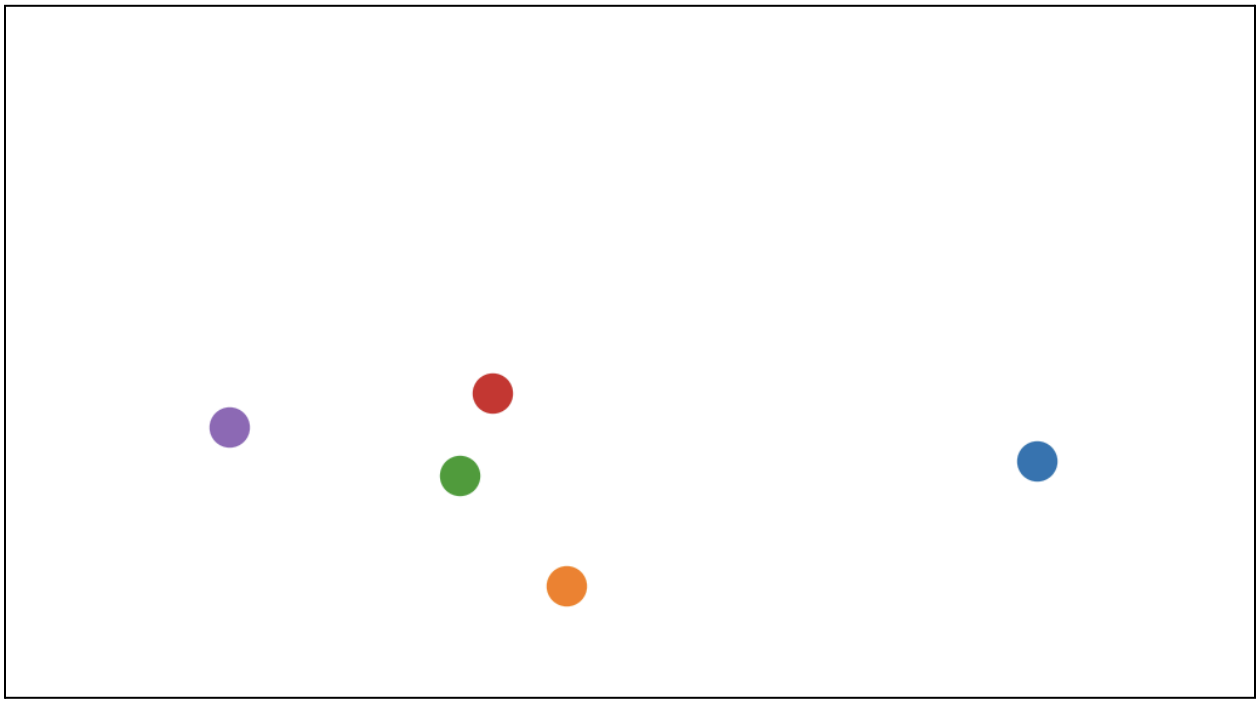
Two members of the SimState class interface with the frontend: "step" and "positions." The former updates the simulation's internal state, and the latter outputs the current positions of the simulation's particles.

User Interface Summary

As shown in the chart on page 4, we implemented the user interface for Syzygy through various elements compatible with our MatPlotLib graphics. Once the simulation state is represented visually, it can be embedded into an HTML page through WebAgg, a service compatible with MatPlotLib. The WebAgg implementation of Syzygy allows for simulation states to be rendered in real time through the user's Web browser, paving the way for interactive instances of Syzygy running online. Our current implementation opens the simulation in a browser window running through the active port, complete with a set of default visualization controls included with WebAgg.

Through our current interface, the user can rotate or move the viewing window of the simulation, zoom in or out, and save a point in the simulation as an image file. In the future, this interface could be improved upon by adding ways to edit the speed of the simulation or skip to a certain step/frame, but this was difficult to implement in our time window because it involved directly editing the default WebAgg HTML structure rather than anything already included on our own end. An additional vision for this interface in the future is allowing for an in-browser command line to connect to our back end and parse syntax/update the simulation directly from a Syzygy-embedded page.

Example Rendering



An example of the output of the program, each point generated by its own command.

The simulation output effectively generates the points and their force trajectories within the animation environment, representing them as standard points from the matplotlib rendering.

Meeting Design Specifications

The final result of this project successfully achieved our goal of serving as a simple, flexible programming language that can create useful basic physics simulations. The commands of the language are clear and concise, allowing users to place points in the simulation wherever they like and to define the forces governing those points as they see fit for their purposes. The animation output is readily visible, interpretable, and manipulatable in the viewer. These constitute the basic design specifications that we set for our project.

Notably we did need to revise our specifications from our initial proposal, where we downsized the product from incorporating large structures to simply including basic points. This improvement could be handled with more time, especially with the language apparatus successfully built, but simply was not possible in a semester.

In Conclusion

Syzygy successfully serves as an effective functional programming proto-language capable of creating highly custom physics simulation spaces of points and forces. By using the Python packages Lark and matplotlib, as well as webagg implementation for matplotlib, the script input by the user is effectively processed through the Syzygy system to create the animation in a webpage viewer.

Past Report Updates:

Midterm Project Update

A detailed list of the tasks that have been completed are shown below on the updated project timeline. So far, we have completed a parser and an interpreter that convert commands in our unique language into a JSON file, as well as a tool that creates an animation using a JSON file. Although the two tools have not been fully connected yet, they are functioning in their separate branches and close to being merge compatible.

In order to ensure that our progress would not reach a bottleneck at any point during the development process, we have split the developmental pipeline into three separate stages which will all connect to create the final product:

- The user command to JSON file phase. This consists of taking a command in our CLI syntax and transitioning it to a JSON file containing the necessary data to create the simulation state. The syntax of the language consists of points, forces, and update rules in a comma separated format. Jonas has primarily taken the lead on this side of the project, and overall it is about 75% complete in the current state. Formal creation of force and update rule processing remains to be done.
- The JSON file to simulation state phase. This consists of taking a JSON with the necessary information about the interacting bodies and transferring it into a simulation state that can be visualized by the user. Jacob has been the primary lead for this section of the project, and it is also about 75% complete in the current state.
- The simulation state to animation phase. This consists of the visualization and user interface tools for viewing the simulation. This is where the bulk of the work on the project will be required post-midterm, and is only about 5% complete currently. All group members will work on this phase going forward, with Hayden and Callum focusing specifically on implementing an intuitive UI. The animation is working, but not interactive from the frontend yet. As of right now, this is done through matplotlib, but may change in future development.

Adjustments in Project Management

Approaching the halfway point of the project timetable, as well as in consultation with the TA's from our project proposal, we have made some adjustments to the scope of our project to ensure we can deliver a complete product for our presentation at Intersections. The primary consideration we decided to eliminate at this stage is the support for solids in our systems, leaving the product as a simulator for individual particles. Since this task felt more like an add-on and less essential for the minimum viable product, we decided this was the most practical aspect to cut. Although we are maybe a half step behind our proposed timeline from the beginning, we are very close to what we needed to finish with regard to everything else we planned in the project proposal.

Remaining Work

As mentioned in the project update, most of the remaining work for our project consists of developing an animation framework and user interface for the animation. Once we connect the command line interface to the animator, the primary focus of the rest of the semester will be

to flesh out the animator. By adding UI options to the animator and improving the simulation state, our application will grow from something that can be used by experts to something that is much more easily scalable to a wider audience.

Project Management Timeline

We have still been meeting weekly, which continues to be the plan after spring break. Below is the updated timetable following the conclusion of spring break:

Deadline	Tasks
Weeks 1-3 (ending 1/31)	☑ ~~Formulate project idea + write proposal~~
Week 4 (ending 2/7)	☑ ~~Start prototyping visualization system independent of scripting language: Matplotlib solar system visualization~~ ☑ ~~Determine needed commands in scripts: Points, Forces, and Updates~~ ☑ ~~Prepare basic parser for simple points and forces Compile commands to json, compile json to anim~~
Week 5 (ending 2/14)	☑ ~~Complete prototype of visualization system Prototype remains in matplotlib~~ ☑ ~~Test compatibility of parser with visualizer Parser pairs with visualizer~~
Week 6 (ending 2/21)	☑ ~~Continue work on frontend designs~~
Week 7 (ending 2/28)	☑ ~~Continue work on merging parser and visualizer~~
Week 8 (ending 3/7)	☑ ~~Create demonstration video (due 3/7)~~
Week 9 (ending 3/14) (spring break)	☑ ~~Spring break~~
Week 10 (ending 3/21)	☑ ~~Begin advanced particle simulation~~ ☑ ~~Build developed documentation resource~~ ☑ ~~Pair parser with visualizer completely~~ ☑ ~~Add in more granular control past first script run (i.e. adjust simulation with simple controls) Frontend team, post break~~
Week 11 (ending 3/28)	☑ ~~Continue advanced particle simulation~~
Week 12 (ending 4/4)	☑ ~~Complete advanced particle simulation~~
Week 13 (ending 4/11)	☑ ~~Start Intersections poster~~ ☑ ~~Final touches on software upgrades~~

Week 14 (ending 4/18)	☑ ~~Intersections poster finalization~~ ☑ ~~Intersections presentation on 4/18~~
Week 15 (ending 4/25)	☑ ~~Write group report (due 4/28)~~ ☑ ~~Prepare final presentation~~ ☑ ~~Deliver final presentation and group report~~