**NPhysics Engine**

At the beginning of this project, we had proposed to work on either a small Rust game or contribute to Rust libraries by porting over standard game libraries. We were looking for a project that would impact tfhe future of Rust, so we looked to the wiki (<https://github.com/mozilla/rust/wiki/Computer-Graphics-and-Game-Development>) for areas of active growth. From there, the obvious choice was the kiss3d graphics engine for Rust (<https://github.com/sebcrozet/kiss3d>) because it combined our dual interests in games and graphics, and would provide a platform for future Rust developers.

From the start, we realized that kiss3d was very much still in active development and trying to keep up with the latest Rust updates. After looking through sebcrozet’s projects, we found the nphysics engine, a young library using kiss3d to provide an engine for 2-D and 3-D graphics. Naturally, our first step was to contact the author sebcrozet and ask where he thought our efforts would be best focused. His response pointed out that there were a number of features we could approach – the one we found most relevant was the parallelization of the collision detection feature. After our initial planning and a few more emails with sebcrozet, we realized that the state of nphysics’ dependencies had left us in a difficult position. It had to do with disparities between the versions of Rust, nphysics, and the other libraries, but in short, we had to depend on the update timeline of sebcrozet and other primary contributors. Without those updates, we were unable to test the base build of nphysics or any of our proposed updates. Later in the report, we will explain our attempts to fix these issues and the results.

At this time, we were rather far along into the project, with significant time put towards developing the proper (often changing) environments for nphysics development. Additionally, we had also looked at most of sebcrozet’s proposed ideas and considered potential approaches. Since it looked like the libraries would not be updated in time for project submission, the natural solution was to document our ideas, hoping that future developers would be able to easily build them into the working library.

**Areas of Work**

This section contains a number of brief explanations of current areas for improvement in nphysics and short thoughts on how we had planned to approach them. We dedicated most of our time to describing parallelization as we thought it was most relevant and useful for the class.

*Additional joints*

Currently, nphysics contains two types of joints: BallInSocket and Fixed joints. The first is suitable for ragdoll physics whereas the second is more useful for locking the translational and relative motion of two objects. As we understand it, when speaking of the relative motion of a set of two objects, there are six types of relative motion (3 translational, 3 rotational). A useful joint to add would be a hinge joint, which restricts motion to a single degree of freedom – that of one object rotating around the other. Implementing the hinge joint would involve developing a system to generate a set of linear equations to solve the Mixed Linear Complementarity Problem, given a single degree of freedom for rotation in one axis.

*Add support for triangle meshes*

Triangle meshes are the most typically used geometric representation of static objects in games. More often referred to as the “polygons” that make up graphics, triangle meshes compose a geometry that simplifies computation by grouping several triangles into a mesh structure. This reduces computations from three per triangle to a single computation per vertex, which may include several triangles. Currently, the collision detector for nphysics does not support the use of triangle meshes. However, triangle meshes are similar to the existing CompoundAABB shape, which is a similar idea for convex geometries. As a result, collision detection is present for arbitrary convex objects, so the majority of the mathematical work is already complete. Implementing triangle meshes would involve creating a separate structure for the collision engine. The key portions of this structure would be including attributes for the engine to detect the proper boundary volumes of any meshes and modifying the collision detection dispatcher to recognize that. For an efficient implementation, it would be best to find a memory-efficient way of representing triangles.

*Add dedicated box v. box collision detection algorithm*

Currently, nphysics uses the Gilbert-Johnson-Keerthi (GJK) distance algorithm to calculate collisions between two objects. This is a powerful algorithm because it allows calculations for two objects of arbitrary convex geometries. The issue is that this can act as a calculation bottleneck for the engine, slowing it down in areas where less generic approaches will suffice. For box v. box collisions, the problem can be reduced to discrete geometry. Implementing a solution for this problem would involve essentially finding or developing an algorithm that provides noticeable speedup in test cases for a large set of cubes. As none of us have extensive graphics/algorithm experience nor would we be able to test our results, we did not put as much thought into an implementation for this problem.

-------------------------------------------------------------------------------------------------------

**Collision detection using parallel pipelines**

Collision detection has already been done, and is available in the ncollide repository by the same author (<https://github.com/sebcrozet/ncollide>). Ncollide, a dependency of Nphysics, is able to detect collisions of the following pairs: ball vs ball, plane vs convex object, convex object vs convex object. Additionally, it supports ray casting and impact time computation for irrotational objects.

In addition to Ncollide, NPhysics uses Nalgebra for vector/matrix math.

Nphysics utilizes the ncollide library and attempts to parallelize the collision detections using Rust tasks. However, it is currently inefficient due to several limitations. A typical physics pipeline parallelization update the broad phase that is not parallelized, builds collision islands (connected components of the collision graph given by the broad phase), and runs the collision detection and compute forces and integrates each island in parallel. However, currently the island computations are done by an independent component (the IslandActivationManager) which cannot communicate with the collision detector and the constraints solver, rendering efficient collision impossible. To solve this problem, the only solution currently would be to rewrite the World class as described below.

**How our plans compare vs. his code + other engines**

After studying some physics engines in other languages and conversing with the author, we decided that it would be necessary to write a new environment that is parallelization-aware (removing the dependency on islandActivationManager, etc) to fix this issue. According to the author, this kind of environment is implemented in Bullet, a real-time multiphysics open source library that provides collision detection, soft body and rigid body dynamics in 3D (<http://bulletphysics.org/wordpress/>). To implement it in Nphysics may require a complete revamp of the code due to its dependency on IslandActivationManager. After some research, we decided not to pursue in the interest of time. Instead we focused on the other bugs and fixes within nphysics library in its parallelization scheme.

**Problems Encountered:**

As we worked with the nphysics engine, we encountered several problems that heavily affected our progress. While working on the engine, Rust-nightly was released and various repositories relevant to nphysics were either out of date or up to date with the latest rust. We initially thought the problem had to do with our local environment but after a lot of debugging and some guidance from the #rust-gamedev irc, we were told the problem was because some libraries were using rust-nightly and others were using rust-0.8. This created a unique situation where we were able to build certain libraries but not others. During this period of transition, certain libraries changed from a status of build-passing to build-failing, indicating that the library itself was unbuildable. We did manage to fix one of the rust-nightly errors regarding the kiss3d engine (also written by sebcrozet) through some debugging but other build errors affected us.

The nphysics engine was compilable afterwards but the examples we needed to test the engine were not compilable. This is largely because the examples depended on the 2D graphics engine, Rust-sfml. The 2D graphics engine was build-failing for a period of time but was only recently set to build-passing. We git cloned the nphysics engine as suggested by sebcrozet for the easiest way to install dependencies. Unfortunately, building the examples still failed when we tried to make the example dependencies. Debugging the problem was even harder because we each had different error messages. For example, Mark had a Rust-sfml fail error while trying to build nphysics examples and Eric had failures within the diagnostic.rs file. We tried cloning the Rust-sfml library and building the library ourselves but we also ran into errors building the Rust-sfml library. Many of the errors that we occurred were fixed over time by various people working on the Rust Computer Graphics and Game Development libraries.

Ultimately, we set out to improve the nphysics engine made by sebcrozet. During our journey, we encountered many problems regarding nphysics. Sebcrozet himself has said the documentation is lacking so we wanted to document our progress and the issues that occurred. Using our documentation, we hope developers who want to contribute to the nphysics engine will have a clear understanding of the current state of the engine and avoid many of the pitfalls we encountered.

\*Both users “Bjz” and “Sebcrozet” were especially helpful throughout our project and consistently gave us sound advice through the #rust-gamedev irc channel and email. Sebcrozet is the repository owner of both Kiss3d and nphysics while Bjz is the repository owner of glfw-rs.

\*Rust-nightly was just updated recently again (12/16) and all the libraries are going through various updates again, changing between build-failing to build-passing