Google Summer of Code 2013 – Abstraction Layer Library for PhysX

Preston Hamlin – prestonwhamlin@gmail.com

An abstraction layer library works on top of an established library of code to provide a simplified coding experience and often more navigable source code. By creating generalized presets and more intuitive object interaction, the programmer is able to quickly write repetitive or frequently used code in a more concise manner. The abstraction layer also serves to hide much of the finer details and features, which may otherwise have required explicit attention from the programmer, while still providing said features for those who desire to make use of them. In this manner, the barrier to entry for the field is lessened.

About Preston

- Solid mathematics and physics background
  - Calculus III, Differential Equations, University Physics II
- Primarily code with C++ and Python
  - Majority of coursework thus far in C++ (OOP, Data Structures & Algorithms...)
- Full time student at Florida State University
  - Pursuing BS in Computer Science

Interests

My interests include Artificial Intelligence, Apparent Intelligence, Computer Architecture and Computer Graphics. I have several self-education projects that I fall back on when I have some free time, which is not very often. These include teaching myself OpenGL, WinAPI, HTML5 (primarily the canvas and drawing to it) and the basics of various languages (Ruby/Rails, C Lisp, Java). For entertainment and a challenge, I work on building a functional CPU in Minecraft.

Why Now?

I have been wanting to contribute to open-source projects for several years now. However, whenever I had found a project that interested me, the problems the project faced were well beyond what I was comfortable with. For example, a couple years ago I was wanted to try and contribute to the Eclipse project. When I looked into the bug tracking and feature requests, they were all concerning the fundamental structure of the compiler and required substantial knowledge of both the project design and internals to even understand how the issue arose, let alone resolve it.

Developing an abstraction layer would allow me to see a project from concept to fruition and provide a great deal of insight into successive improvement of code via peer review and rigorous testing. I find the collaboration and amount of detail put into open-source projects to be a wondrous thing and very much desire to participate.

Availability

I will be spending the summer assisting one of my professors with a research project he is working on, so I will be available somewhat over half-time. When the Fall semester begins in mid-August I will be available roughly half-time but may have drops in availability when course registration begins.

Future Plans

I should hope that with sufficient documentation, the code provided would be clear to understand. Especially since this would be an abstraction meant to improve the fluidity of code. However, this cannot be guaranteed for all persons. As such, I would gladly answer questions regarding the project after the Summer is done. Since the PhysX library continues to advance, chances are that the code will need infrequent maintenance. I do not know if a future GSoC student would be interested in advancing the project, but I will certainly keep up with Nvidia's releases and continue to contribute periodically as my studies allow. This would also be an educational experience, dealing with forward and backward compatibility of code.

Ratings

C++ - 4/5
While I am not familiar with all the obscure facets of the language, I am confident in my ability with and understanding of what I do know.

STL - 3/5
Having taken a class all but dedicated to the STL, I am quite familiar with the containers, and the algorithms are a bit further off.

Boost - 2/5
Unfortunately the majority of my coursework requires explicitly writing all source code, only allowing for STL headers. As such, hands-on application of Boost libraries has been lacking.

SVN - 2/5
I am familiar with the utilities and concepts for both SVN and GIT and have used both. I rank it low because the only practical usage has been with other students for small projects.

Utilities

Development	I have used Visual Studio, Eclipse (with MinGW) and Makefiles/g++ for C++ coding. I primarily use Visual Studio or a combination of Notepad++ and g++ depending on what I am coding.
Documentation	I am most familiar with Doxgen for generating HTML manuals.

Proposal

I wish to develop a library of code to rest atop the PhysX 3.2 SDK for Linux (and possibly other platforms). This entails designing and implementing classes and methods that utilize existing SDK code to provide a simplified coding experience. Whereas currently one might write

```
std::vector<PxRigidActor*> physics_actors;

static PxDefaultErrorCallback default_error_callback;
static PxDefaultAllocator default_allocator_callback;
PxFoundation* foundation = PxCreateFoundation(PX_PHYSICS_VERSION,
                                              default_error_callback,
                                              default_allocator_callback);
PxPhysics* physics = PxCreatePhysics(PX_PHYSICS_VERSION, *foundation,
                                     PxTolerancesScale(), true, PZoneManager);

PxSceneDesc scene_description;
scene_description.filterShader       = &gDefaultFilterShader;
PxScene* scene                       = mPhysics->createScene(scene_description);
Renderer* renderer = getRenderer();

PxReal density          = ___;
int radius              = ___;
PxVec3 position         = ___;
PxMaterial* material    = scene->getMaterialFromIndex(___);

PxRigidDynamic* sphere = PxCreateDynamic(*physics, PxTransform(position),
                                         PxSphereGeometry(radius),
                                         *material, density);

SetupDefaultRigidDynamic(*sphere);
scene->addActor(*sphere);
physics_actors.push_back(sphere);
createRenderObjectsFromActor(sphere, material);
```

I would endeavor to simplify it down to

```
physics          physics_a1(___);
scene            scene_a1(physics);
sphere           sphere1(radius);

sphere1->set_material(___);
sphere1->set_velocity(___);
scene->add_actor(sphere1);
```

Where the material, positron, velocity, rotation, rotational velocity, scale and other attributes all have default values. An object would be instantiated by supplying arguments to a constructor and/or calling mutator functions. To create a joint between two objects one could do so with a single function:

```
sphere1.join(cube3, FIXED);
```

Where FIXED is part of an enumeration of joint modes. In addition to the various primitives, a 3D ngon could be produced from a container of points in the Cartesian coordinate system.

Tentative Roadmap

Week 1	•Map class interactions	•Develop pseudocode, parent classes	
Week 2	•Map PxPhysics interaction	•Develop first drafts of primitives	
Week 3, 4	•Map PxScene interaction	•Develop PxPhysics abstraction	
Week 5, 6	•Map rigid body interaction	•Develop PxScene abstraction	
Week 7	•Analyze materials	•Determine enumerations	•Determine exceptions
Week 8	•Finalize a build	•Analyze and discuss results	
Week 9	•Implement cooking	•Test	
Week 10	•Implement friction	•Test	
Week 11	•Implement particles	•Test	
Week 12	•Implement advanced collision	•Test	
Week 13	•Finalize build	•Determine To-Do list	