Thraxion: Three-Dimensional Action Simulator

Justin Gerthoffer, Jon Studebaker, David Colborne
Jeff Stuart, Frederick C. Harris, Jr.

Department of Computer Science and Engineering
University of Nevada, Reno
1664 N. Virginia St.
Reno, NV 89557, USA

{gerthoff, jstudeba, colborne, stuart, fredh} @cse.unr.edu

Abstract: *The Thraxion program detailed in this paper is a user interactive 3D action simulator that models the behavior of objects in motion. Using a physics engine and collision detection, Thraxion is able to model the behavior of objects of varying mass, size, composition, and texture colliding against each other. The user is able to create objects, edit the force directions and magnitudes action upon the objects, and may then begin a simulation based on these parameters. The user may then save the initial conditions of the simulation, load different initial conditions, or create a new scenario. Because Thraxion is based on OpenGL and the Qt user interface libraries, it has multiple platform support and is being actively developed on Linux and Mac OS X.*

Keywords: collision detection, bounding sphere, three-dimensional simulation

1. Introduction

Thraxion is a software project designed to allow the user to easily design a three dimensional scene that can be simulated and rendered in real time. A rudimentary physics engine, already built by the authors, will be the basis for the physics controlling the graphics on the screen. Ease of use and making the existing engine more accurate are two of the major goals of this project.

The software will allow the building of scenes to display what would happen under a set of initial conditions set by the user. The environment will allow for spheres colliding with solid objects, distance constraints between objects and, in the future, solid object collision. It will allow for the saving and loading of three dimensional scenes for running at later times. The users will be able to choose pre-built objects and create almost any environment they wish with them.

Almost anyone should be able to benefit from the program since ease of use is a major goal. Target audiences include younger students who might wish to see what they have learned about basic Newtonian physics in action, as well as programmers looking for an easy-to-use three dimensional collision simulator for a graphics engine.

The program will be novel in that it is easy to use, deals only with fundamental collision objects, is fast, and will make it easy to create and manage complex three dimensional simulations. It will be perfect for quick demonstrations in 3D involving basic physics.

By modeling Thraxion using UML notation, such as explained in [5], the project has been better understood and the goals have been simpler to extrapolate based on the existing functionality. Planning of Thraxion has also proven to be simpler and more consistent by following UML and basic UP guidelines.

This paper, in its remaining part, is organized as follows: Section 2 presents the main functional and non-functional requirements, Section 3 includes details on use cases, Section 4 describes the architectural design of Thraxion, Section 5 describes the detailed design of Thraxion, Section 6 provides the current state and

future work of the project, and Section 7 provides the conclusion to the report.

2. Requirements Specification

Following standard software engineering guidelines, the main functional and non-functional requirements of Thraxion are presented below.

2.1 Functional Requirements

The most important functional requirements of Thraxion are:

1. The user shall be able to write, open and save scenes for demonstration purposes.
2. The user shall be able to add text comments to scenes.
3. The user shall be able to switch between various graphical modes, including wireframe mode and solid mode.
4. The user will have at least four views in the graphical user interface, including an isometric view and three plane-sliced views.
5. The user will be able to create scenes using a graphical user interface for the designer.
6. The user will have a set of pre-defined objects to build scenes with.
7. The user may switch between a scripted scenario editing mode and a purely graphical scenario editing mode.
8. The user may be able to move items around interactively during scenes.

2.2 Non-Functional Requirements

The most important non-functional requirements of Thraxion are:

1. The program shall run on most common Linux distributions.
2. The program shall be built using Qt 3.3.
3. The program shall require less than 128 megabytes of random access memory for proper functioning.
4. The program shall run all scenes with at least a 25% speed increase upon completion compared to the original speed of the program at the beginning of this project.
5. The program shall implement mouse and keyboard user interaction.
6. The program shall run 5 limbs with one solid at a rate equal to or greater than 30 frames per second using a Pentium 4 running at 2 GHz with 512 MB of RAM running Slackware 10.
7. The program will run on Mac OS 10.3 without using X11.
8. The program's source will compile without editing the Makefile on Linux.
9. The program will use a scripting language for scene editing instead of manual compilation of scene files.
10. The program may run on Windows XP.
11. The program may use a graphical user interface for scene editing, instead of requiring user knowledge of a scripting language.
12. The program may use a tree-based collision detection system to optimize the code.

3. Use Case Modeling

As part of the formal modeling process laid out in [5], the functionality of Thraxion has been defined using use cases and scenarios. The entire functionality of Thraxion is captured in the use case diagram shown in Section 3.1 at a high level of abstraction; this was done to help identify the mechanisms through which the user would interact with Thraxion. The use cases are compared to the requirements listed in Section 2 using the Requirements Traceability Matrix in Section 3.1.

3.1Detailed Use Cases

Presented below are Use Cases for Thraxion. A Use Case Diagram is presented in Figure 1.

UC1. Install Qt – Qt is the GUI front end for Thraxion. Depending on the system, installing Qt can be relatively easy or an extremely engaging and time-consuming task.

UC2. Build Source – Includes acquiring and building the source using qmake or an edited Makefile. Also includes certain platform-specific issues.

UC3. Create Ropes – Create ropes, around which particles and objects may be

grouped for certain effects (a necklace, for example).

UC4. Create Sphere – Creates a sphere, which is essentially a large particle.

UC5. Create Solid – Create a solid, which is composed of many particles.

UC6. Create Stationary Object – Create a stationary object, or an object with infinite mass, such that it does not move when objects or forces strike it.

UC7. Create Limbs – Creates an inflexible rope around which particles may be grouped around for certain effects (rigid body motion, for example).

UC8. Create Forces – Creates a vector force upon an object that can interact with a force (non-stationary objects).

UC9. Save Scene – Save a pre-created scene.

UC10. Open Scene – Load a saved scene.

UC11. Start Scene – Begin a created scene.

UC12. Stop Scene – Stop a running scene.

UC13. Edit Forces – Edit existing forces interacting with an object.

UC14. Zoom In – Zoom in on a scene, for a better view.

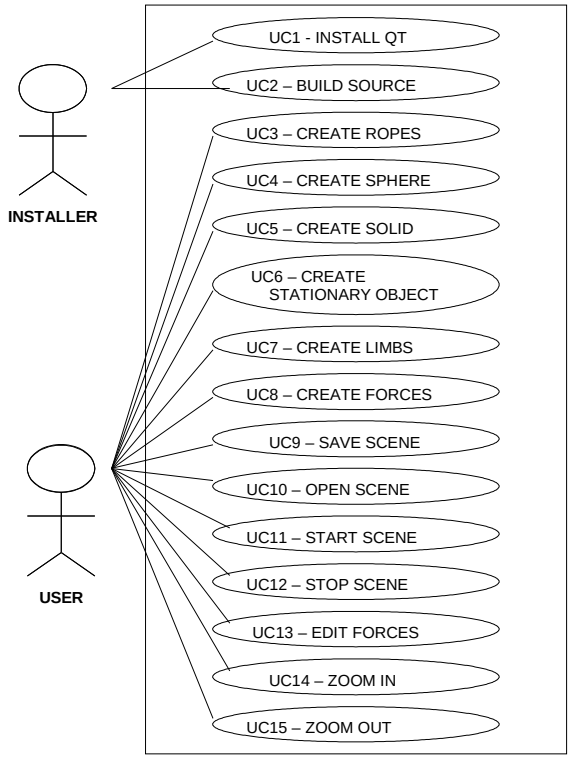

Figure 1: Use Case Diagram for Thraxion

UC15. Zoom Out – Zoom out on a scene, for a better view.

UC16. 3.2 Requirements Traceability Matrix

The Requirements Traceability Matrix detailed below shows how the use cases match up with the requirements listed in Section 2.

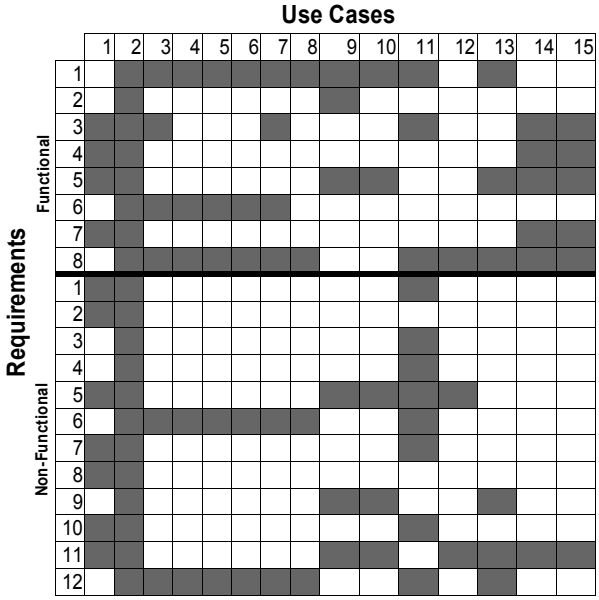

Figure 2: Requirements Traceability Matrix

4. Architectural Design

The layered architecture diagram is presented in Figure 3, and is a hierarchical generalization of the layers of the Thraxion program. A brief description of each subsystem is as follows:

C++ Libraries: All of Thraxion is implemented using C++; consequently, Thraxion makes heavy use of a number of C++ libraries.

OpenGL: The graphics rendering in Thraxion are handled through OpenGL[1].

Qt: The GUI and display for Thraxion are handled through Qt libraries and the Qt framework[2].

Collision Detection & Reaction: Collision detection and reaction to each collision is

currently handled through an all-to-all bounded spheres-based method. However, alternative approaches, such as [3] and [4] are being considered to increase the efficiency and capabilities of Thraxion.

Effects, Lights & Textures: Controls the look of objects, using OpenGL properties.

Particle System & Objects: Identifies the location of each object and the number and properties of each particle composing each object.

Playback Tab: Controls the pace and sequence of playback of a scene by interacting with the Canvas Manager interface, which manages the display of the results of collisions, effects, and the particle system.

Designer Tab: Used to switch between playback and designer mode; can edit the properties being managed by the Canvas Manager interface.

Main Window: Displays all pertinent information for the user.

Help System: Built-in series of documents designed to aid the user as they interface with Thraxion.

GUI: The entire interface system through which the user interacts with Thraxion; includes the Main Window and the Help System.

5. Detailed Design

The system activity chart for Thraxion, per the layout recommended in [5], is included as Figure 6.

The class diagram for Thraxion, laid out according to the specifications laid out in [5], is included in Figure 7, which lists all the classes in Thraxion as well as most of the major functions. Due to size constraints, certain trivial variables and functions have been omitted, as have Qt- and OpenGL-specific inherited functions that some classes implement. However, such classes are marked as having inherited from the appropriate Qt and OpenGL class where possible

6. Current Status and Future Work

Thraxion is currently fully functional on many major Linux distributions and currently exhibits approximately 25% improvement in speed as measured by frames per second compared to the original program on which Thraxion is based. The Macintosh distribution can be run using X11 and an X11-based Qt installation; work is continuing on allowing Thraxion to behave like a native Macintosh application to provide greater speed and lower overhead. A help system has been implemented, as has a graphical user interface, detailed in 6.1. Future work includes allowing Thraxion scenario files to be scripted, so that they do not need to be compiled before run, making the help system more comprehensive, further optimizing the code through the use of tree-based structures, such as those in [3] and [4], and lifting the current object quantity limitations once the collision detection code can handle it.

6.1 Screen Shots

Figure 4 shows the Designer Window in its current form. The Designer Window is the main interface used by the user to create new scenes. Figure 5 shows the Main Demonstration scene, in which the user may test the scenes created in the Designer Window or other scenes generated by other users. Figure 5 is a scene composed of a series of spheres and some solids organized around a rope. This helps illustrate the rope concept, as well as display how various objects interact with each other in Thraxion. The screen shots were created with the March 15, 2005 version of Thraxion; future releases may have changes to the interface.

7. Conclusion

The Thraxion action simulation software whose specification has been presented in this paper is an innovative, low cost and low overhead solution where users can experiment and interact with a basic physics and graphics engine. The focus of Thraxion has been on creating a multi-platform 3D simulation engine that is rich in functionality but low on total overhead in terms of memory and speed.

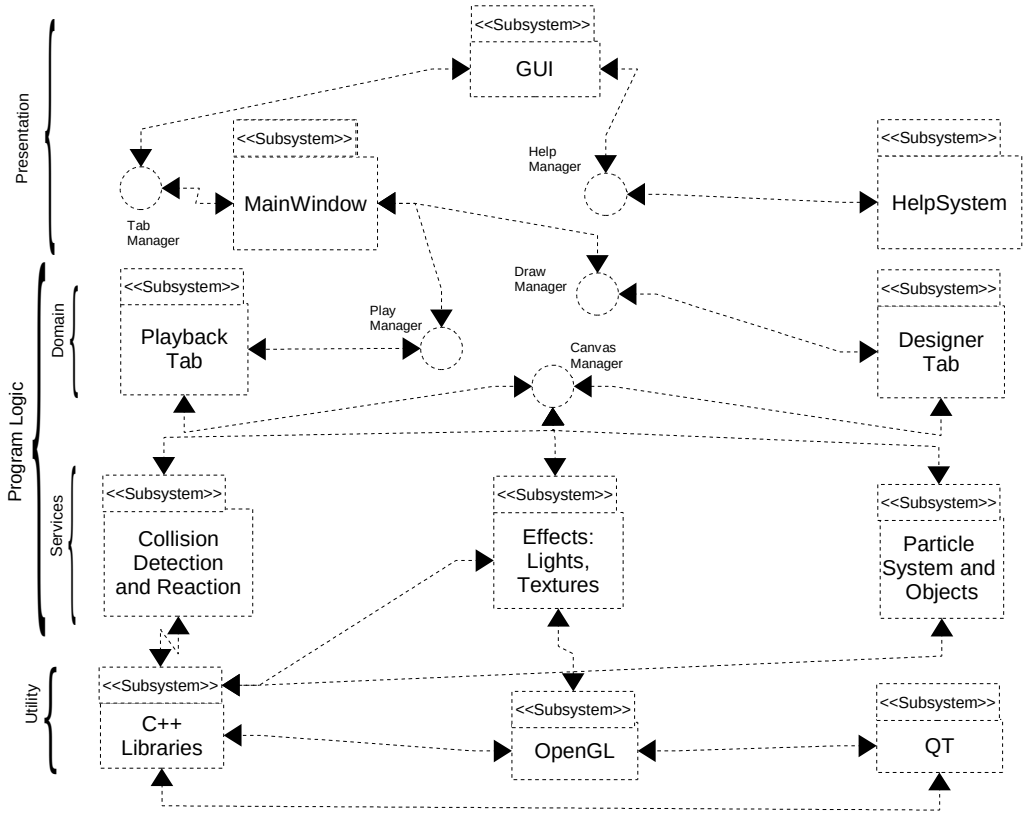

Figure 3: Thraxion Layered Architectural Diagram

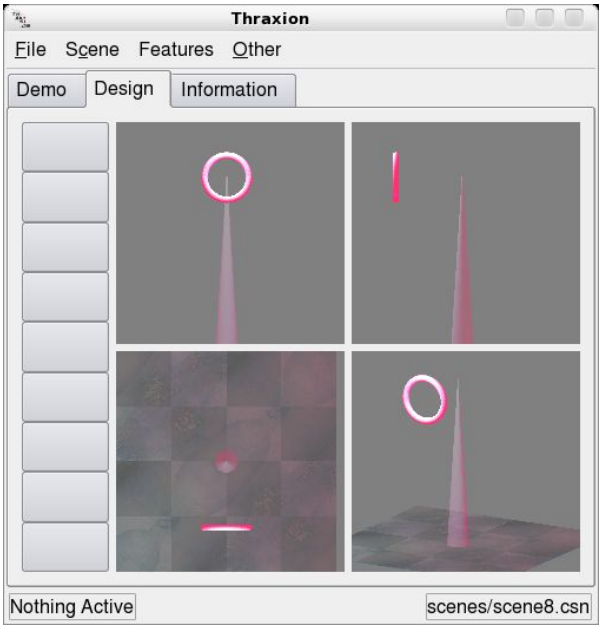

Figure 4: Designer Window

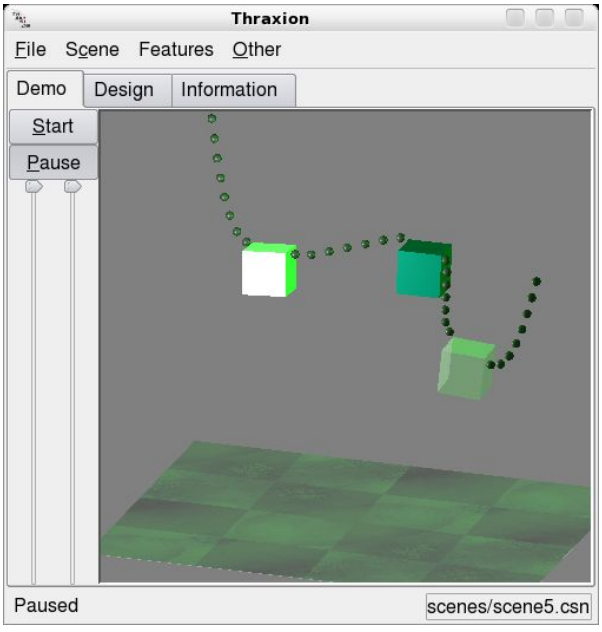

Figure 5: Main Demonstration scene

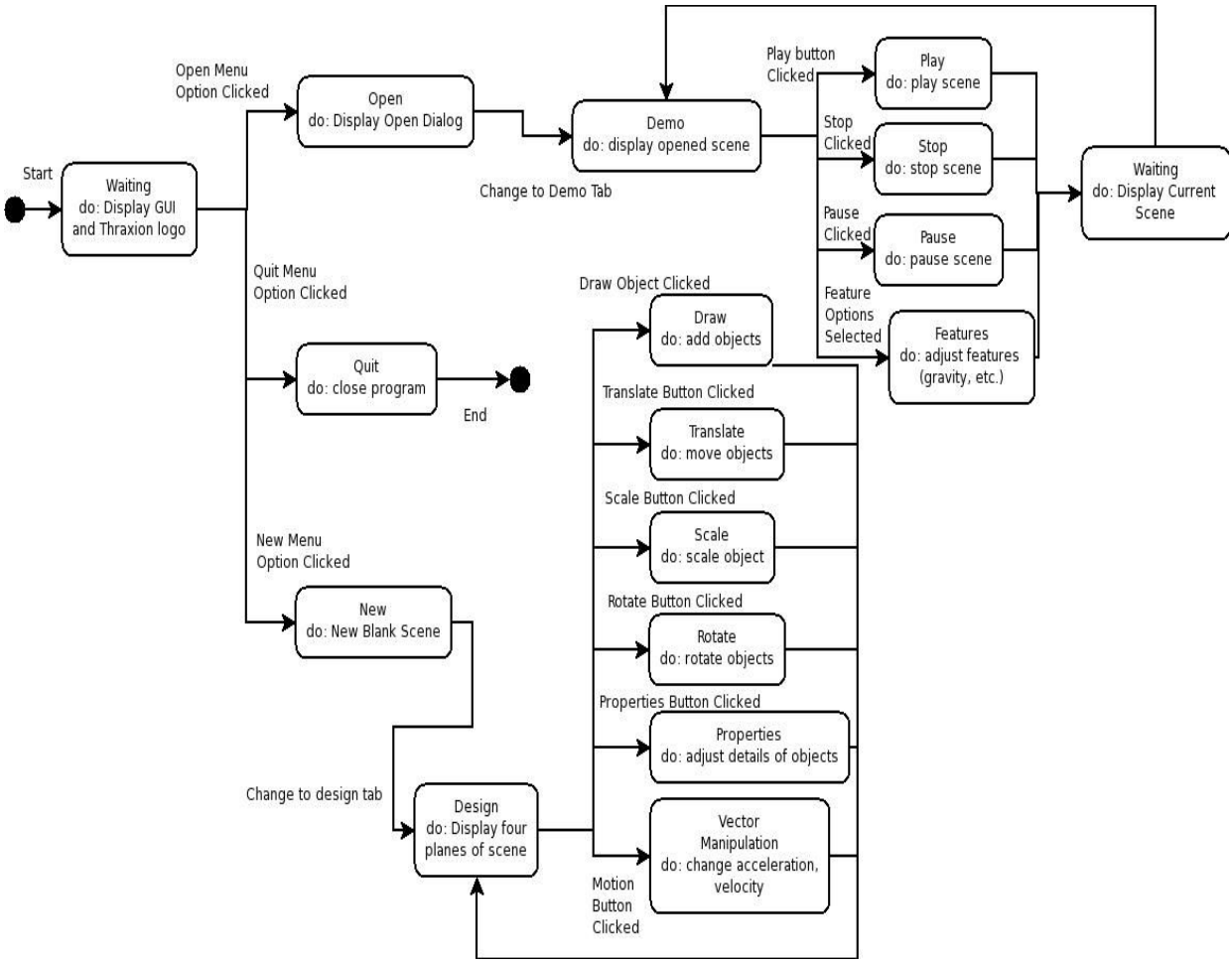

Figure 6: Thraxion State Diagram – Partial

The possibilities for enhancing our work are endless and could involve applications such as game design, computer animation, or even educational applications. For game design, Thraxion may serve as a basic collision detection engine that could be useful for simulating the effect of object collisions on background objects. Computer animation may benefit from having Thraxion handle the basic collisions and interactions between simple objects, leaving the complicated collisions and interactions to more complex programs that incur greater overhead. For educational applications, Thraxion could serve as an easy-to-use demonstration tool of basic Newtonian physics on multiple objects for the high school or early college level. Thraxion could also serve an educational purpose as a simple, working, non-theoretical example of how to implement a low-cost, low-overhead 3D collision engine. Consequently, further refinements in code and collision detection algorithms used would result in a stronger application for any of these domains.

8. References

[1] Angel, Edward, *OpenGL - A Primer*, Addison-Wesley, 2nd Ed., 2005

[2] Dalheimer, Matthias Kalle, *Programming with Qt*, O'Reilly, 2nd Ed., 2002

[3] Zachmann, Gabriel, "Minimal Hierarchical Collision Detection," *VRST'02*, November 11-13, 2002

[4] Otaduy, Miguel A; Lin, Ming C., "CLODs: Dual Hierarchies for Multiresolution Collision Detection," *Eurographics Symposium on Geometry Processing* (2003)

[5] J. Arlow and I. Neustadt, *UML and the Unified Process: Practical Object-Oriented Analysis & Design*, Addison-Wesley, 2002.

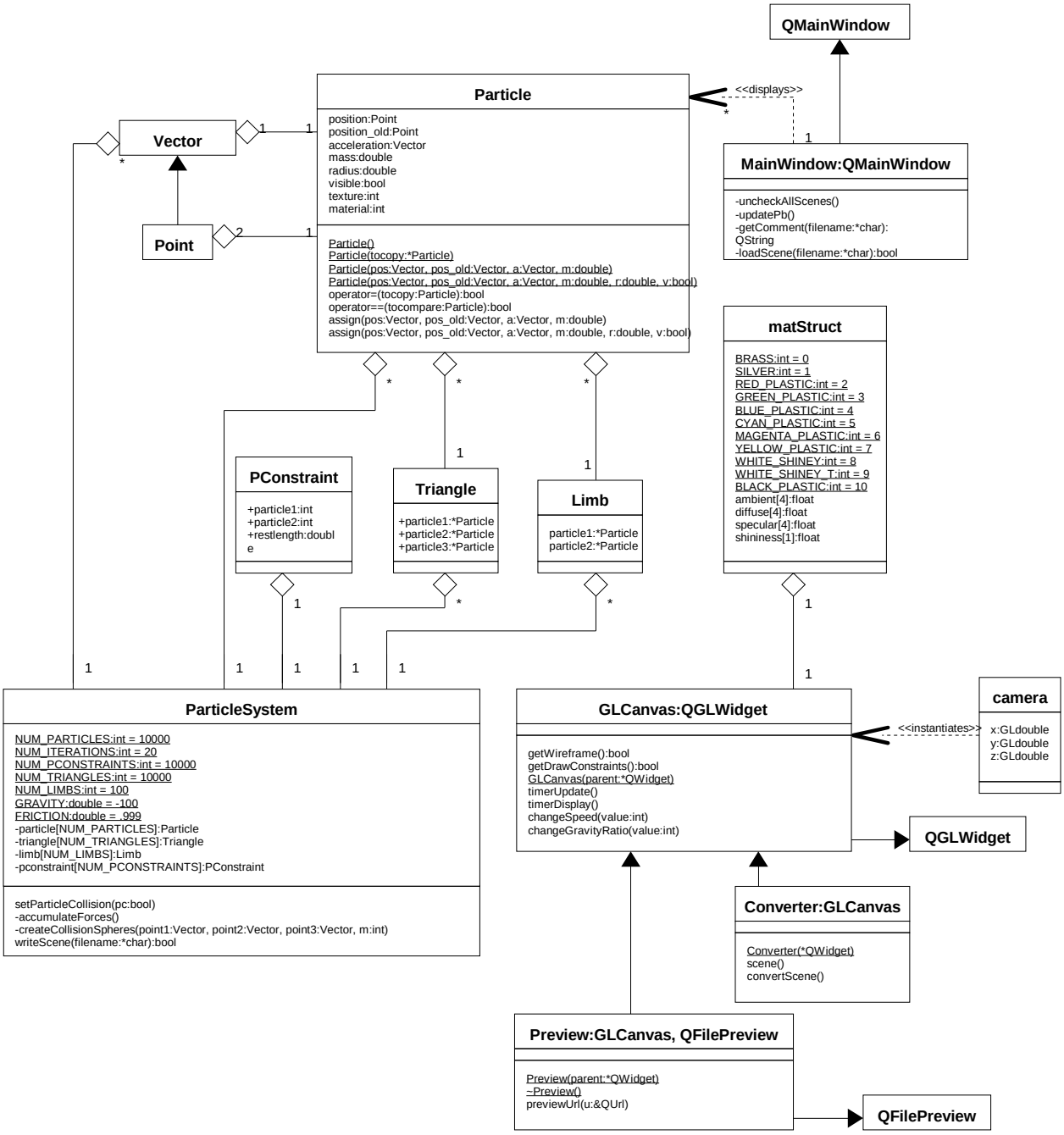

Figure 7: Thraxion Class Diagram