Sri Lanka Institute of Information Technology

Shapeless

Project ID : 16-018

Software Requirement Specification

Student ID:

Name:

B.Sc. Special (Honours) Degree in Information Technology

Table of Contents

1 Introduction

1.1 Purpose

This document provides all the requirements for Shapeless, the soft body dynamics and fluid dynamics system. Described are details of how the system will be implemented including functional and non-functional requirements. All parts are intended primarily for the users of the plugin but may be of worth to game developers intending to improve the system.

1.2 Scope

This document covers the requirements for release 1.0 of Shapeless. The document will also cover features that future releases may contain. The proposed system incorporates fluid dynamics and soft body dynamics to provide a comprehensive deformation and fluid plug in.

This plugin is targeted at the Unity game engine to help independent developers include deformable solids in their games without having to resort to engine's implementations of non-deformable solids.

1.3 Definitions, Acronyms, and Abbreviations

Term	**Definition**
Metaball	A 3 dimensional particle that interacts with other particles in a fluid like manner.
Prefab	A prefabricated template object used to generate game objects.
Primitives	The basic 3D shapes, cubes, spheres and so on.
Rigid body	An object that is not subjected to deformations while interacting with the environment and other objects.
Shader	A computer program used to do shading, the production of appropriate level of colour and light of a model.
Render	The process (including all calculations) of producing a single (graphical) frame of a simulation.

1.4 Overview

Shapeless is a plugin for the Unity game engine. The distributed plugin will comprise of two major components, soft body deformation and fluid dynamics. The components will allow game developers to use these components to create non-rigid objects and fluids in their games.

This document will focus on the product requirements that the team needs to fulfil as well as constraints and targets the team need to deal with. Section 2 describes the product with the intended audience being the user. Section 3 focuses more on the development aspects of the product.

2 Overall Descriptions

Fluids will be implemented using an approximation to metaballs using Unity engine's primitive components. A metaball particle prefab will be created to be used in the fluid system. This will include the component that controls the particle's behaviour like surface tension and viscosity. A library of shaders will be developed to the user of the plugin. These shaders will include lit and unlit shaders for several types of fluids, some of them incorporating transparency and refraction.

A fluid particle factory (or tap) will be produced for developers to generate fluid particles in-game. The factory will itself be able to produce particles in different ways that games may require, with exposed functions to configure duration, rate of emission, pulsing emission and so on.

The soft body physics component via shape matching will aid the development of video games that employ deformable solids. The mass-spring system component will be developed to try to include some of its accuracy measures into the shape matching method, or see if it can render in real-time fast enough to be used in video games.

2.1 Product perspective

Below are comparisons of products that attempt to solve some of the problems identified for the development of Shapeless.

haVOK

Havok engine is a proprietary system for destruction in video games. It includes functionality for different deformations but is heavy and is generally used by AAA games.

JELLY MESH

JELLY MESH is a simple plugin for unity that allows soft body deformations. It lacks accuracy and is used for simple simulations.

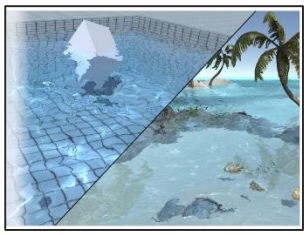
VertEx motion

VertEx motion is a decent plugin for unity that supports 1-dimensional and 2-dimensional deformation in meshes, which are hair and cloth.

Fluvio

Fluvio is a plugin for unity that handles fluid simulation. The plugin handles fluid behaviour well but falls short on realistic visuals.

Realistic Water



This is a plugin for unity that handles water simulation. The plugin provides limited interaction support for water, but handles rendering incredibly well.

Comparison

	Soft-body Deformation	Dimensions of Deformation	Fluid Simulation	Intractable Fluid Particles	Custom Fluid Shaders and Materials
Shapeless	✓	**3**	✓	✓	✓
Havok	✓	**3**			
VertEx Motion	✓	**1, 2**			
Jelly Mesh	✓	**3**			
Realistic Water			Only Water		✓
Fluvio			✓	✓	

Software interfaces

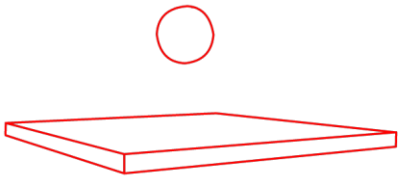
The system will be developed as a plugin for the Unity game engine. Therefore this plugin will require Unity to use; which allows compilation to all popular platforms.

2.2 Product functions

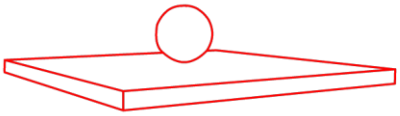
Soft body dynamics using shape matching

In video game engines, 3 dimensional objects are handled as-is, that is, engines do not change the object itself, or its points. This is done primarily because considering the object as a ‘rigid’ body allows numerous optimisations to be done for collisions and physics systems the engine is implementing.

A rigid body hitting the floor



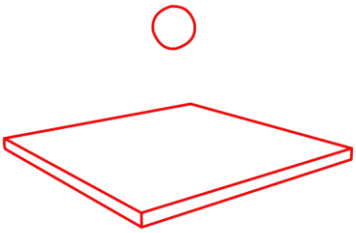
Rigid body before hitting



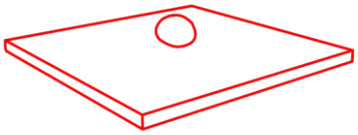
Rigid body after hitting

The soft body dynamics component will implement a soft body handler for the engine, so as to allow soft bodies to be simulated, which work more intuitively and naturally than rigid bodies.

A soft body hitting the floor



Soft body before hitting



Soft body after hitting

The shape matching function aims to models soft bodies using a ‘shape matching’ algorithm initially proposed by Muller [1].

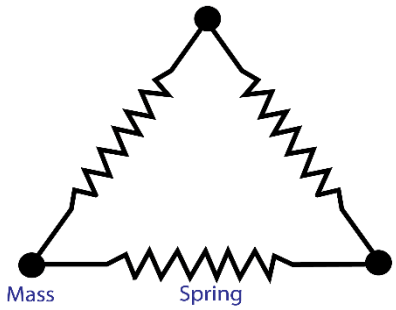
The main idea behind shape matching is to model the object as a set of particles with initial positions and masses, and in such a way that the particles of a mesh are not connected to each other and do not interact with each other.

The particles do however respond to external forces and collisions. After the reaction to an external force, the particles’ positions change relative to their starting position, relative to the centre of mass of the object. For each time step, the particles are moved to the goal position, which is calculated using its original and actual positions.

After the implementation of this component, a unity component will have been created, that users would only have to attach to their 3D meshes for them to work as soft bodies. The shape matching algorithm used promises fast rendering speeds at the expense of simulation accuracy.

Soft body dynamics using spring mass model

This system will use Verlet integration to implement the spring mass model for handling soft bodies.



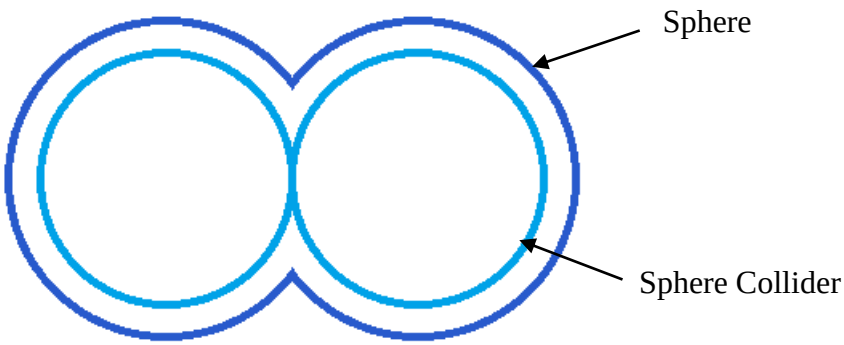
The mass-spring system works by modelling each particle or vertex as a point mass and the connections between particles as elastic springs. Dynamics are then computed with these assumptions.

After the development, here too a unity component will have been created, that users could attach to their 3D meshes for them to work as soft bodies. The spring mass algorithm however, tends to be slightly expensive but is accurate.

After the implementation of the two algorithms, optimised versions of either will be produced to reap the benefits of both algorithms while overcoming each other's drawbacks.

Fluid dynamics behaviour

In this plugin, fluid simulation is implemented using the concept of metaballs. Here, an approximation to metaballs is implemented through spheres in the game engine and sphere colliders. The engine's sphere colliders which typically bound the volume of spheres, are given a size slightly smaller than the, actual sphere. Hence, when two spheres get closer to each other, they will give an illusion of merging with each other due to their smaller colliders, which are actually touching each other. Through this, interaction between fluid particles is created.



For fluid behaviour, a component will be created to dictate the behaviour of fluid particles. Users would not typically engage with this component directly, but use other game objects that utilise the behaviour component.

A particle prefab will be available for users to base their own fluid particles off. A particle factory component will also be created, that users can attach to game objects of their choice, to spawn fluid particles.

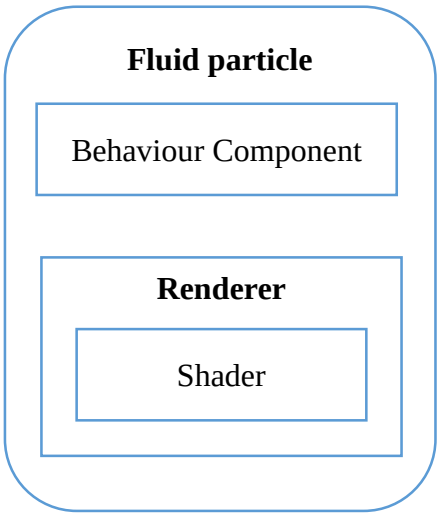
Fluid generation and visualisation

The particle factory creates metaball objects from the prefab. This component will be able to configure the spawned metaballs emitted. Settings include rate of emission, direction of emission, if the emissions should be in bursts, the lifetime of the metaballs and so on.

For rendering fluids, custom shaders will be written. These shaders will be available to users for their fluid particles to obtain a variety of fluid visuals.

Lit shaders will be written for fluids that are fluorescent, like magma or for a chemical in a game that should appear radioactive. Unlit shaders would comprise the rest of the shaders.

Transparent particles will use shaders that incorporate refraction to obtain a better sense of realism.



Architecture of the particle

2.3 User characteristics

The system is intended to be used by independent game developers and hobbyists.

2.4 Constraints

Shapeless simulations should operate on mobile devices, with the target device having 1.0 GHz processor, 1 GB RAM and a dedicated graphics processor. The test simulation could be fluid with 100 simultaneous particles or 10 primitive cubes with the deformation components.

2.5 Assumptions and dependencies

Assumptions

- The users of the system understand how to use a downloaded plugin for Unity
- All mobile phones and other computing devices running this plugin in games have dedicated graphics processors and CPUs with a speed of at least **1.0 GHz.**

Dependencies

- The system will require Unity version 5.1 onwards for installation.

2.6 Apportioning of requirements

The requirements described in section 2 are the primary specifications. The requirements in section 3 are the functional requirements. Initially the functional requirements will be implemented. These will be developed while making sure the non-functional requirements are met. The optional requirements may be developed as the developers see fit.

3 Specific requirements

The following are the main requirements of the system

- Creating a particle system for fluids that allows customization of fluid particles.
- Providing visualization options for fluid particles for developers.
- Create a comprehensive, customizable particle generating system.
- Create a fast soft body deformation system.
- Create an accurate soft body deformation system.

3.1 External interface requirements

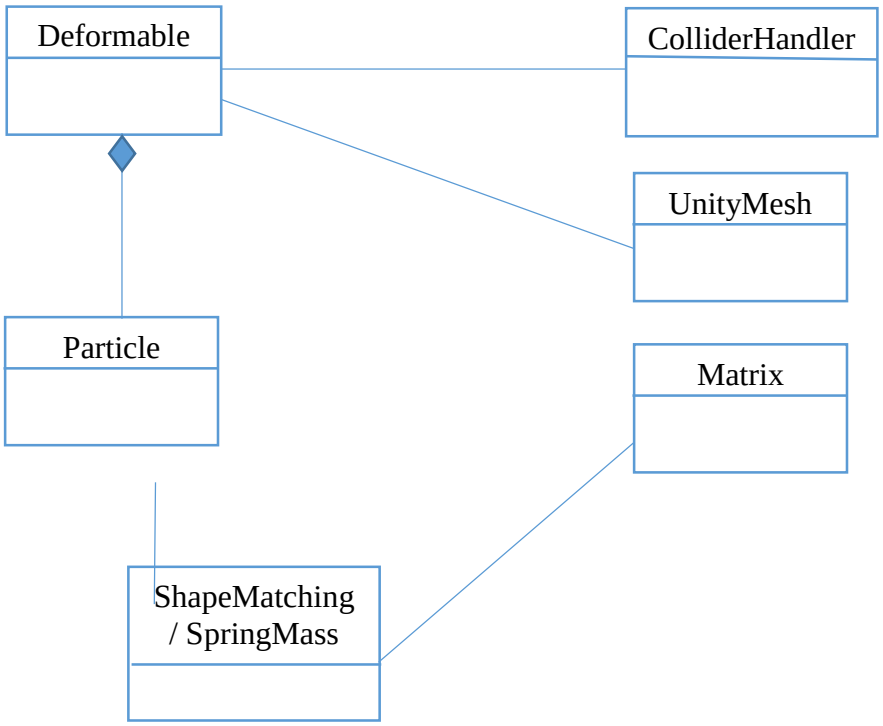
Software interfaces

- Unity and Mono Framework

Unity engine's primitives as well as its collider and collision components will be used for the development. This also improves performance by offloading collision detection to the engine. These components are mainly used for compatibility with Unity.

3.2 Classes/Objects

Soft Body Deformation



With this architecture of implementation, users will only have to attach the Shape-Matching or Spring-Mass component to their game object to add the deformation functionality.

The classes and objects the algorithms will be implemented with, haven't been elucidated here.

UnityMesh

The class that represents 3D meshes in Unity.

Deformable

A 3D mesh composed of several particles.

Particle

A single vertex or a point in a 3D mesh. Particles contain speed along with their x,y,z coordinates.

ColliderHandler

Changes the collider of the mesh in Unity to fit the new deformed mesh.

Matrix

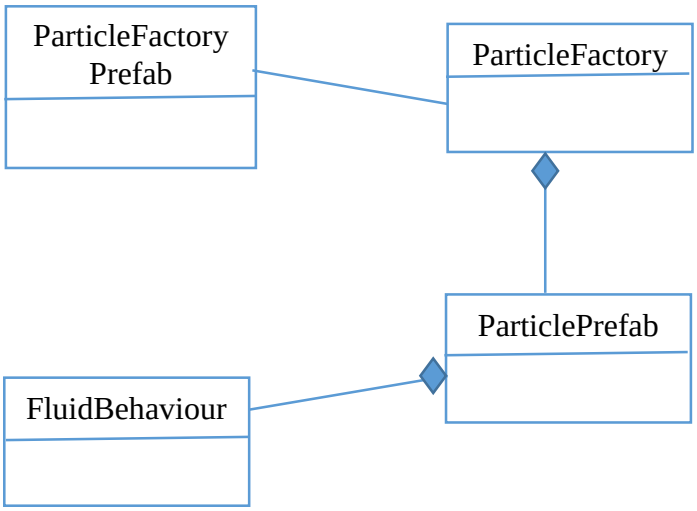
Does matrix operations.

ShapeMatching

Handles the shape matching algorithm for deformations.

SpringMass

Handles the spring-mass algorithm for deformations.



The user can configure the particle prefab and its behaviour according to their game. The particle factory prefab also accommodates configuration, after which the user can attach their particle prefab to it.

ParticleFactoryPrefab

The virtual object that generates particles.

ParticleFactory

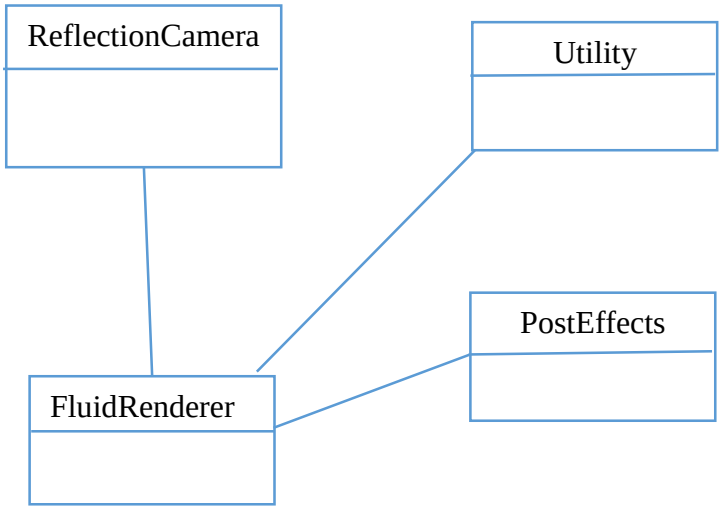
The class that handles particle generation.

ParticlePrefab

The virtual particle object

FluidBehavior

The class that handles fluid behaviour,that will be further extended by several classes for different functionalities.



Users will not directly interact with the renderer components besides deciding on the shaders to be used. Shaders will have exposed values for game developers to configure them as required. The rendering components will use an alternate camera among other components to achieve fluid effects.

FluidRenderer

The top level class that handles the reflection camera, shaders and additional effects.

ReflectionCamera

The camera object that is used to generate reflection details.

Utility

Has utility functions for rendering.

PostEffects

Handles additional effects for fluids.

3.3 Performance requirements

Render Time

- A scene should render in 60 fps, 10 primitive cubes with the soft body deformation component attached.
- A scene should render in 60 fps, 100 fluid particles with the fluid behavior component attached with the most resource-intensive shader (refraction).

3.4 Software system attributes

Maintainability

Due to the open source nature of the product it is vital that the code be extensible. Modifications should be simple to implement and future components added should work well with the base system.

The component based architecture of the plugin allows simply adding and removing features to the metaball prefab via components. This means extensions of the project can swap out even the sphere primitive used with a more advanced or effective mesh for different functionality.

References

[1] M. Müller, D. Charypar et al. “Particle-Based Fluid Simulation for Interactive Applications” - *Eurographics/SIGGRAPH*, San Diego, California, 2003.

[2] Y. Yu, H. Jung and H. Cho, "A new water droplet model using metaball in the gravitational field", Computers & Graphics, vol. 23, no. 2, pp. 213-222, 1999.

[3] R. Geiss, "Ryan's Guide to MetaBalls (aka Blobs)", *Geisswerks.com*, 2016. [Online]. Available: http://www.geisswerks.com/ryan/BLOBS/blobs.html. [Accessed: 14- Jan- 2016].

[4] S. Laster and K. Bailey, "Flocking Algorithms", Flocking Fish, 2011. [Online]. Available: https://flockproject.wordpress.com/2011/05/14/flocking-algorithms-2/. [Accessed: 07- Mar- 2016].

[5] Gourlay, Michael J. "Fluid Simulation For Video Games (Part 1) | IntelÂ® Developer Zone*". Software.intel.com*. N.p., 2012. Web. 1 Mar. 2016.

[6] K. Crane, I. Llamas, S. Tariq and H. Nguyen, *GPU Gems 3*. Boston, MA: Addison-Wesley Professional, 2007, pp. 633-675.

[7] J. Groff, "An intro to modern OpenGL. Chapter 2.2: Shaders", *Duriansoftware.com*, 2010. [Online]. Available: http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-2.2:-Shaders.html. [Accessed: 28- Feb- 2016].

[8] W. Engel, *Direct3D ShaderX*. Plano, Tex.: Wordware Pub., 2002.

[9] Terzopoulos, Demetri et al. "Elastically Deformable Models". *Siggraph.* Pasadena: N.p., 1987. Print.

[10] Terzopoulos Demetri, and Kurt Fleischer. "Modeling Inelastic Deformation". *Proceedings of the 15th annual conference on Computer graphics and interactive techniques - SIGGRAPH '88* (1988): n. pag. Web. 13 Feb. 2016.

[11] "Havok Destruction". *Havok*. N.p., 2016. Web. 6 Feb. 2016.

[12] Muller, Matthias et al. "Meshless Deformations Based On Shape Matching". *TOG* 24.3 (2005): 471. Web. 8 Feb. 2016.

[13] Parker, Eric G., and James F. O'Brien. "Real-Time Deformation And Fracture In A Game Environment". *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '09*

(2009): n. pag. Web. 12 Feb. 2016.

[14] Teschner, Matthias. "Modeling Dynamic Deformation With Mass Points And Springs". 2003. Presentation.

[15] Liu, Tiantian et al. "Fast Simulation Of Mass-Spring Systems". TOG 32.6 (2013): 1-7. Web. 7 Mar. 2016.

[16] A. Rivers and D. James. “Fast Lattice Shape Matching for Robust Real-Time Deformation” - *SIGGRAPH*, 2007.

[17] Sharma H. *Soft Body Deformation Dynamics Based On Shape Matching*. 1st ed. Dorset: N.p., 2013. Print.

[18] M. Desbrun, P. Schröder and A. Barr, in Interactive animation of structured deformable objects - *Proceedings of Graphics Interface*, Kingston Ontario, 1999.