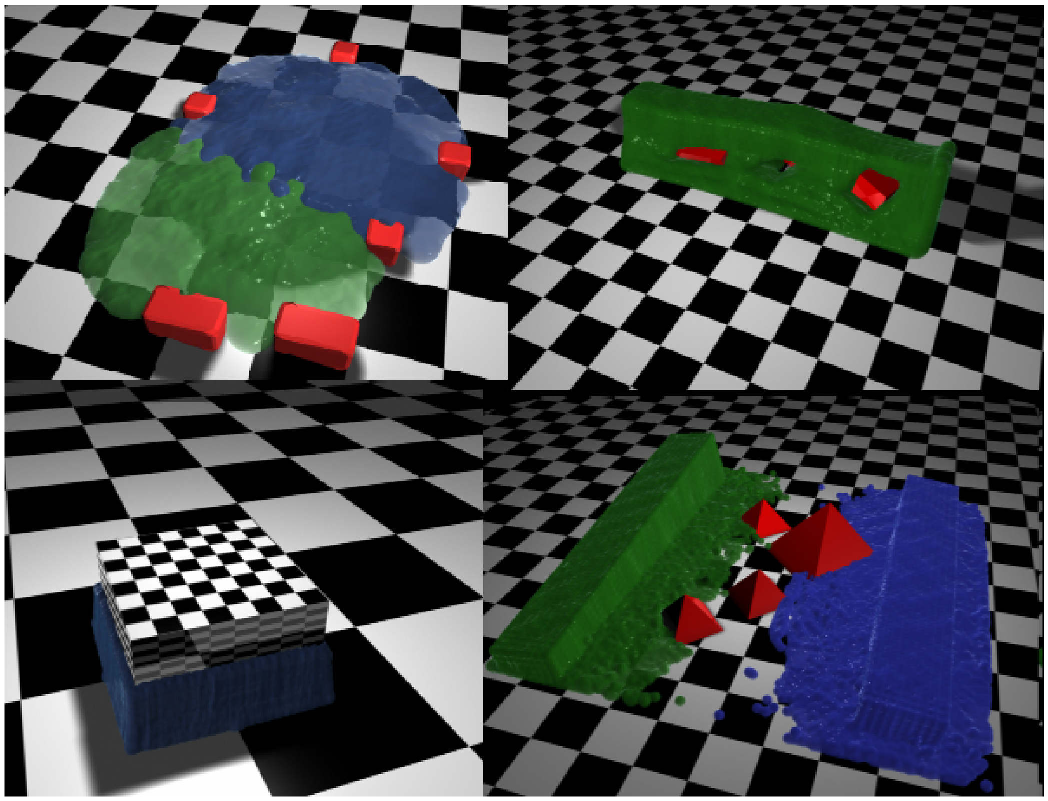
INTUITIVE AND FAST SIMULATION FOR SOFT AND VISCOUS SUBSTANCES

Vincent Bonnet

August 16, 2010



Msc Computer Animation and Visual Effects
Master Project
-
N.C.C.A BOURNEMOUTH UNIVERSITY

Abstract

This report demonstrates a framework to create soft and viscous substances. The system is based on a particle system where every particle has a potential and interacts with repulsive/attractive forces, friction and drag force to mimic the fluid viscosity. The fluid motion is not based on Navier-Stokes equations or other fluid theories, but on function sets to create a fast and intuitive tool to generate soft and viscous substances. The fluid simulation can be imported into Maya thanks to a particle cache writer developed for the project. The final result provides a larger range of effects going beyond the study of soft and viscous substances. A theoretical and practical approaches are covered in this paper.

Keywords: computer animation, fluid modelling, soft substances, meshless, molecular dynamics

Contents

List of Figures

1 Introduction

1.1 Motivation

Rigid body dynamics and fluid simulation play an important role in visual effects and computer graphics. Another type of object called soft body and deformable object is also a vast subject and cover fracturing objects, melting objects, soft substances, viscoelastic fluids etc. Those techniques try to mimic natural phenomena in computer graphics and are categorized into two type of algorithms.

The first type is called mesh-based method and is based on a grid or a mesh (spring-mass system). The mesh-based technique is a very popular algorithm in cloth simulation and jelly objects. Vertices are linked together by springs which causes the swing effect and gives a believable motion.

The meshless technique is based on particles and is used to simulate several effects such as highly deformable objects, fluids, soft substances (clay dough or mud), etc. Particles can interact among one another and with the environment. In visual effects, the physics doesn't matter as long as the result looks appealing. Real-time and fast algorithms help the artists to test if a concept is promising or not. Refining an animation which takes few minutes to compute is always dramatical in the industrial environment.

This study describes an intuitive and innovative concept to develop believable substance motions. The main goal is to provide an up-to-date report for researchers and developers to create fluid motions using mathematical functions.

1.2 Related work

Before 1980, most of the established modelling techniques were developed for engineering components such as phone, cars etc. Between 1980 and 1990, it's the innovative period in computer graphics where scientists tried to create natural phenomena by developing algorithms dealing with none-solid objects.

In 1983, Reeves developed the first particle system for LucasFilm to model fuzzy objects such as fire, clouds and water [1]. Particle systems offer a flexible and intuitive approach to track the movement of particles with positions, velocities and colours. His work will lead to new ways for object modelling techniques.

In 1986, G. Wyvill and colleagues[2] introduced the concept of soft objects when most of the modelling techniques were for solid geometries, they gave the first technique to model and animate soft objects including fabrics, cushions, living forms, mud and water.

In 1989, G. Miller and A. Pearce extended the algorithm by creating interactions among the particles for powder, fluid and viscous substance [3]. Those inter-particle relationships were based on a modified Lennard-Jones potential which is originally a mathematical model to describe the potential between a pair of neutral atoms or molecules. This technique is also suitable for breakable and melting objects.

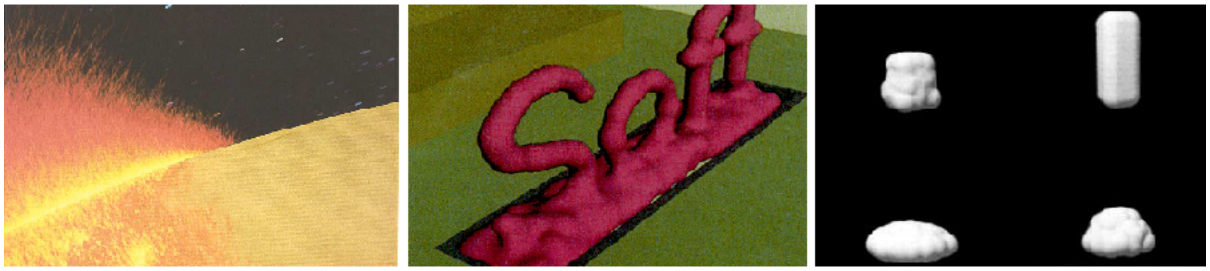


Figure 1.2.1: Reeve's particle system, Wyvill's soft object and Miller's viscous fluid

From 1995 to 1997, M.Desbrun and M.Gascuel released a range of meshless based algorithms for soft substances and soft objects extending and improving previous algorithms such as volume conservation and collisions.

In 1995 and 1997 [4][5], they introduced a system to animate soft substances using a particle system where the collision detection for deformations and contact forces were handled by the smooth isosurface defined by the particle system itself. They also improved the fusion and separation of a same substance.

In 1996 [6], M. Desbrun and colleagues explored the simulation for highly deformable objects and used a more advanced physical approach to describe the behaviour of a material. Desbrun used the smoothed particle hydrodynamics (SPH) which was previously introduced by physicists for accurate simulation of fluid dynamics.

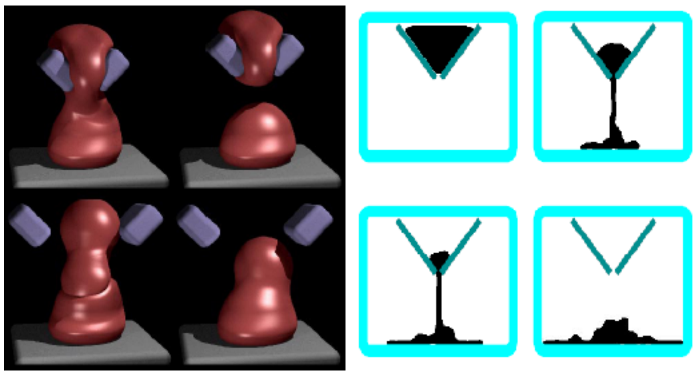


Figure 1.2.2: M.Desbrun and M.Gascuel's works

More recently, other techniques have been explored to obtain a larger range of effects.

In 2003, Muller and colleagues developed a particle-based fluid for interactive applications by using SPH to simulate fluids [7]. This technique is an extension of the SPH-based algorithm developed by Desbrun and Gascuel and it can deal with up to 5000 particles

In 2004, Muller [8] described another technique to simulate different range of materials such as plastic, elastic and melting objects. During the same year, Steele and colleagues[9] demonstrated a similar concept to create highly viscous liquids such as honey, molasses and syrup. The momentum and volume preservation have been the two main improvements and the interactions between particles are managed by mathematical functions.

In 2005, S.Cavet [10] and colleagues demonstrated a technique to simulate viscoelastic fluid by adding springs with varying rest length between particles. This algorithm supports elasticity, plasticity and viscosity effects.

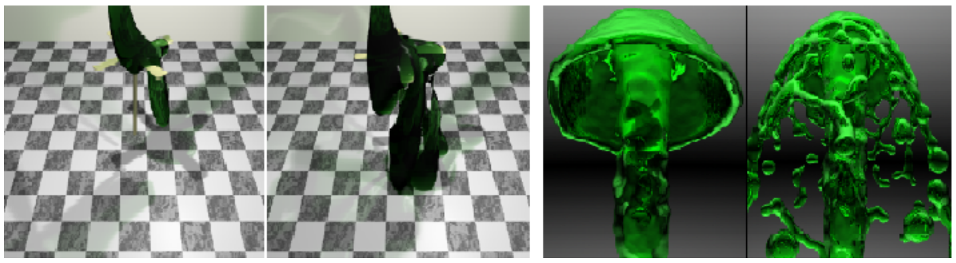


Figure 1.2.3: Steele's viscous liquid and Cavet's viscoelastic fluid

The interaction between a fluid and an object uses particulized objects and consists in converting an object into a list of particles across its surface or by filling its volume.

In 2005, this technique has been used by Bell [11] to detect collisions between granular materials (e.g. sands) and geometries. They have decided to cover rigid bodies of particles and the complex problem of collision detection between a fluid and an object was simplified to a more efficient and intuitive particle-particle collision.

In 2008, Takahiro Harada [12] released a paper for NVIDIA to demonstrate the concept of rigid body dynamics on the GPU. The collision detection was based on particles instead of geometries.

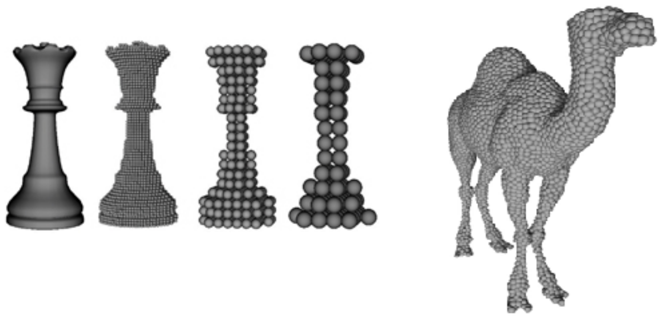


Figure 1.2.4: NVIDIA Chess representation and Bell's camel

1.3 Contribution

This project demonstrates a unified framework for interactive particles. The user/artist has the control and can choose any type of interactions. This new approach guarantees an intuitive and fast way for the artists to control their fluids and soft substances by "tweaking" and manipulating a small amount of parameters to achieve a realistic and natural effect.

2 Theory : Modelling entities with particles

2.1 Concept and Objectives

The concept is based on meshless technique which consists in representing assets as a list of particles for the simulation of physical phenomena. For instance a fluid is modelled as a group of particles interacting with one another according to a set of predefined formulae. The fluid can occupy any position offering a less restrictive system than the Eulerian model where the fluid is confined between boundaries.

The meshless concept is applied for a wide range of effects such as fluid, rigid body dynamics and soft bodies. This project is a proof of concept demonstrating a valid method to create particle interactions using a dictionary of functions to mimic believable soft substance motions.

Those following points are the requirements of the application :

- Develop framework for particle interactions based on functions.
- Good level of abstraction to support other type of objects in the future.
- Develop a pipeline and plugin for Maya to create and render a scene.
- Artist-friendly plugin.
- The final motion needs to be believable but not physically correct.
- The particles are not limited in boundaries.
- The system has to be robust and stable.

2.2 Object representation

In this project, objects are treated as a list of particles. For a particle-based fluid the particle representation is straight forward. The problem arises for rigid bodies when geometries have to be represented and converted into particles. There are several ways to tackle this problem, first a brief history of the concept is introduced and next the algorithm and implementation are unveiled.

2.2.1 Concept

A technique demonstrated by N.Bell in 2005 [11] and extended on the GPU for rigid body simulation by NVidia [12] describes an object as a list of particles. This approach was primarily used for granular material to detect collision between a solid object and sands. This technique has two main advantages; the particularized object is a cloud of particles which is auto-generated, moreover the sphere-sphere intersection is the most efficient algorithm of collision detection, hence object/object, fluid/fluid or object/fluid collision are simplified into a problem of particle/particle collision with different type of interactions.

2.2.2 Event

Fluids or rigid bodies are constituting of particles describing their shapes. All the computations are done at the particle level and to send back specific computations taking place at the object level (fluid or rigid body) a link has to exist to keep their dependencies. The concept of *event* is the link between an object and its particles. A pointer on the IParticulazedObject which is the abstract base class for a particulized entity is stored in each particle.

The figure below shows the particle interaction manager sending an event to a particle which passes another event to its owner. This concept is useful for rigid bodies when an impulse is triggered to the particle but the final computation of the transformation has to take place at the object level.

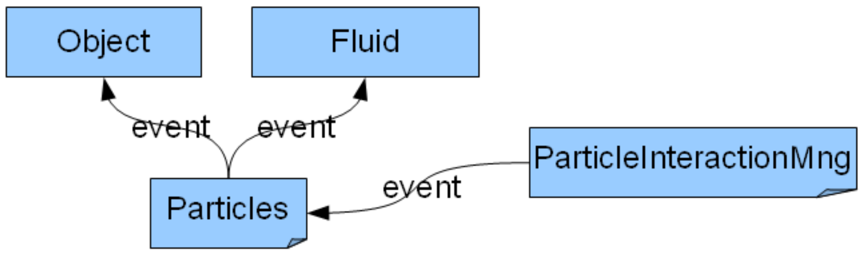


Figure 2.2.1: Particle/Object event

2.2.3 Algorithm

The generation of particles to cover an arbitrary geometry is less intuitive than across a basic primitive and requires the implementation of a generic algorithm.

This ”geometry to particles” conversion is a native feature in Houdini (”points from volume” Houdini sop), unfortunately, the project pipeline is based on Maya and no such implementation exists. A command has been developed to convert an object into particles.

The conversion algorithm consists in generating a voxel grid where the size of the voxel is the diameter of the particle. In order to place particles into this regular grid, rays are sent from an arbitrary grid face to the opposite grid face. Every time a ray hits the surface of the mesh, a counter keeping track of the number of intersections increases its value. Meanwhile, each time the ray crosses the middle of a voxel, it checks its counter. An odd number means the voxel is inside the object, hence a particle is created. On the other hand, an even number means the voxel is outside this object.

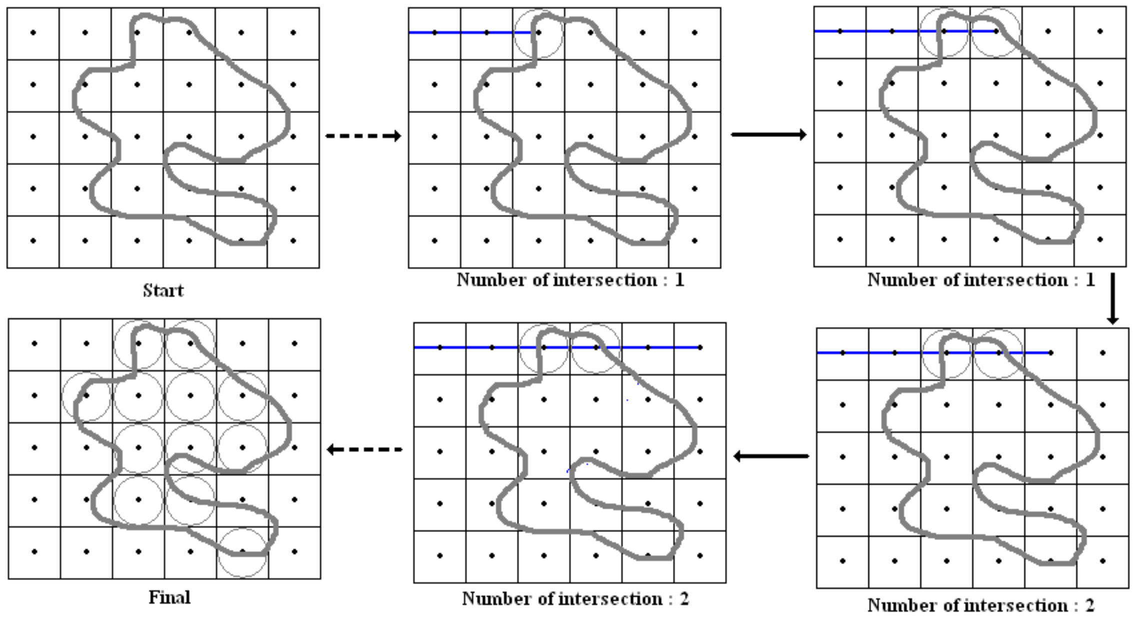

Start
Number of intersection : 1
Number of intersection : 1
Final
Number of intersection : 2
Number of intersection : 2

Figure 2.2.2: Convert to particle

The figure above demonstrates the different steps of the conversion. The top-left image is the object in the regular grid. The image 2 and 3 show the ray hitting the middle of a voxel with intersection counter equals 1, hence a particle is created. The image 4 and 5, show the ray hitting the following voxels with an intersection counter which is an even number (2), by consequence particles are not created.

2.2.4 Limitations

This type of representation is useful for collision detection but it implies a limitation in the transformation of an object. In fact the object containing those particles should have an uniform scale along every axes. For a non-uniform scale the algorithm sphere-sphere intersection is broken because the radius is not the same along every axis. This problem doesn't occur for fluids because their particles are already in world space by consequence no transformation is required for a fluid. The second limitation comes from the amount of memory required by meshless technique. For instance, a realistic fluid simulation has to be produced with high resolution which involves a large number of particles and memory.

2.3 Dynamics

2.3.1 Fluid Dynamics

As explained in the previous chapter the fluid is made of particles emitting an attractive/repulsive potential. The instantaneous force on a single particle is the sum of all the forces related to the surrounding particles plus the gravity. Two different forces describe the motion of a particle in the fluid; One is a function based on Lennard-Jones potential which is the attractive/repulsive force between neighbour particles. The second force is called the shear or viscous force and prevents the motion of a particle in the fluid. Those two forces are respectively explained in the two following subsections.

$$F(p) = \sum(f(pi)) + g$$

2.3.1.a Attractive/Repulsive force

The attractive/repulsive force is described as a Lennard-Jones potential. Sometimes called L-J potential, this function is a mathematical model initially used to simulate the interaction between two pair of neutral atoms or molecules. This model has been proposed in 1924 by the mathematician John Lennard-Jones.

This simple model uses two parameters. The first parameter is called depth of the potential well defining the smallest negative value of the function. The second parameter is the finite distance where the inter-particle force is equal to zero. A system of two particles is in equilibrium when their distance of separation is the same as the finite distance. This attractive/repulsive force is sampled into a list of linear functions because it is constant during the simulation (see 3.4 Optimisations). The function below is the L-J potential where r is the separation between two particles, o is the finite distance and e stands for the potential-well depth.

$$F(r) = 4e[(\frac{o}{r})^{12} - (\frac{o}{r})^{6}]$$

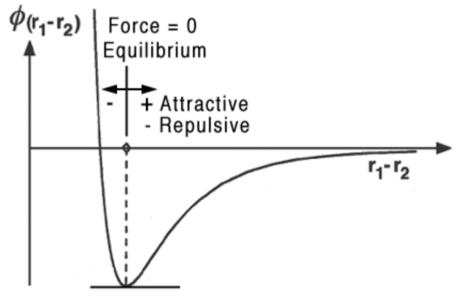


Figure 2.3.1: Lennard-Jones interaction

2.3.1.b Viscosity force

This extra force is the viscosity of the fluid and it limits the shear motion of the particles into the fluid. This force is modelled has a damping spring by reducing and damping the velocity of the particles. The function below describes the viscosity force where $\dot{p}2$ is the velocity of the neighbour, $\dot{p}1$ is the velocity of the current particle and $damping$ is the damping coefficient.

$$F(s) = (\dot{p}2 - \dot{p}1)damping$$

2.3.2 Rigid Body Dynamics

Rigid body dynamics is the study of object's motion, it describes the way objects fall and collide against one another. The next paragraphs give a quick introduction how rigid body simulation works and for more information, see the project *Vincent Bonnet, 2010 - "Fast rigid body dynamics with pre-fracturing"*.

2.3.2.a Linear and Angular motion

The computation of the new position and new velocity with an explicit euler integrator is given by those formulae below, where $s(t)$ and $s(t + dt)$ stand for the current and next position, $v(t)$ and $v(t + dt)$ stand for the current and next velocity and $a(t)$ is the acceleration.

$$s(t + dt) = s(t) + v(t).dt$$

$$v(t + dt) = v(t) + a(t).dt$$

Finally, the angular velocity is updated by this formula where $r(t + dt)$ and $r(t)$ describe the current and new orientation and $w(t)$ is the angular velocity.

$$r(t + dt) = r(t) + w(t).dt$$

The three following parts explain the collision response system based on impulses for rigid bodies.

2.3.2.b Impulse vs Force

The collision response models how an object bounces or slides against other objects. It exists different collision response algorithms and in this project, the collision reaction is computed by impulses. In their papers, Bell [11] and Takahiro [12] use a penalty based technique where the repulsive force is modelled by a spring and a damping force. It works fine but a force is not the best way to simulate collision response because it acts on the acceleration instead of the velocity. A collision lasts a fraction of second and applies a strong force on both of the objects involved in the collision. The impulse is more suitable for collision response and changes the velocity instantaneously.

2.3.2.c Impulse calculation

The amplitude of the impulse calculated without rotation is given by the formula below, where $Vrel$ is the relative velocity between two objects, $\widehat{N}$ is the normal contact (it should be normalized), e is the coefficient of restitution and $m1$, $m2$ are the respective mass of the two objects in contact.

$$J = \frac{-(1+e)Vrel.\widehat{N}}{\frac{1}{m1}+\frac{1}{m2}}$$

The amplitude of the impulse taking into account the rotation is given by the following formula where $I1^{-1}$ and $I2^{-1}$ are the respective inverse inertia tensors of the two objects, $\widehat{r1}$ and $\widehat{r2}$ are the vectors from the center of mass to the contact point.

$$J = \frac{-(1+e)Vrel.\widehat{N}}{\frac{1}{m1}+\frac{1}{m2}+(I1^{-1}(\widehat{r1}\times\widehat{n})\times\widehat{r1}+I2^{-1}(\widehat{r2}\times\widehat{n})\times\widehat{r2}).\widehat{n}}$$

Finally, the impulse vector is given by multiplying the amplitude of the impulse by the normal contact. The friction is ”faked” by scaling the impulse vector along the tangent of the contact. This scalar is called tangent attenuation.

$$\widehat{J} = J\widehat{N}$$

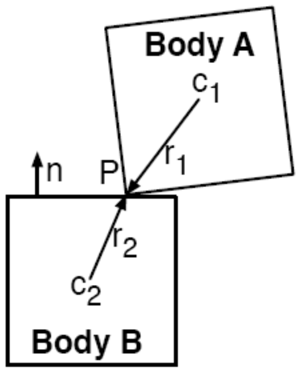


Figure 2.3.2: Collision between two bodies

2.3.2.d Applying an impulse

After calculation, the impulses are accumulated to the rigid body state of the object to be integrated. An impulse changes the velocity and the angular velocity of the object. The velocity is changed by using this formula.

$$V' = V + \frac{\widehat{J}}{m}$$

The angular velocity is more complex and involved the inverse of the inertia tensor. The inertia tensor is computed in body space and changes according to the orientation of the body. In order to transform the inverse of the inertia tensor, the similarity transformation is computed where $R(t)$ is the rotation matrix and $I(0)^{-1}$ is the inverse of the inertia tensor in body space. Moreover a rotation matrix is an orthogonal matrix which means $R^T = R^{-1}$ by consequence it is possible to simplify the similarity transformation by replacing the inverse rotation matrix by the transpose of the rotation matrix.

$$I(t)^{-1} = R(t)I(0)^{-1}R(t)^{-1} = R(t)I(0)^{-1}R(t)^T$$

Finally, the new angular velocity is given by the formula below, where $I(t)^{-1}$ is the inverse inertia tensor in the world space, $\widehat{J}$ is the impulse and $\widehat{r}$ is the vector between the center of mass and the contact point.

$$w' = w + I^{-1}(\widehat{J} \times \widehat{r})$$

2.3.3 Interactions

An interaction describes the motion function between two particles belonging to two different entities. For instance two different fluids are considered as two different entities. A particle object and particle fluid belong to two separate entities too. This paper focuses on two different types of objects (fluid / rigid body) which means four possibilities of arrangements exist: Fluid/Fluid, Object/Object, Object/Fluid and Fluid/Object. The two last interactions are similar and have the same effect. The table below resumes the rules and differences between the interactions.

Interaction table				
Fluid / Object	**Fluid / Fluid**		**Object / Object**	
Different entities	Same fluids	Different fluids	Same objects	Different objects
Global function	Fluid property function	Global function	-	Object property

2.3.3.a Fluid-Object interaction

The simplest case is the fluid-object interaction which is managed by a global function shared across the whole scene. An improvement can be done by creating a dictionary of interactions to deal with different functions to create different effects. The function is a pure repulsive force modelled by a linear function. This approach has been proposed by Steele in 2005 [9]. The figure below is the default static function representation of the fluid-object interaction. In this example, the values are clamped between the maximum (5) and the minimum (0), if the distance between the two tested particle exceeds 1.5, there is no force.

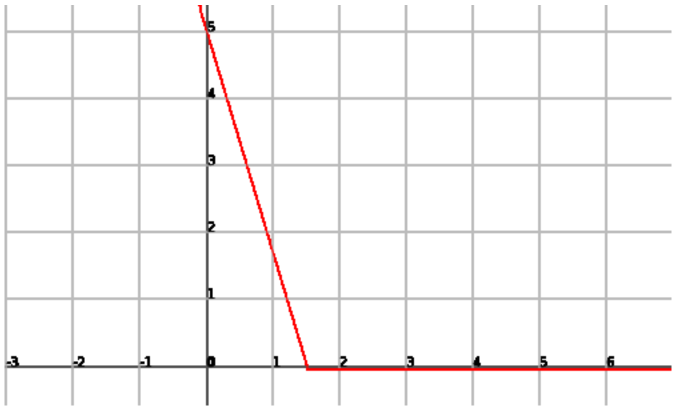


Figure 2.3.3: Fluid-Object interaction

2.3.3.b Fluid-Fluid interaction

The fluid-fluid interaction is the force exerted on each particle from two different fluids. The function between two particles from the same fluid has been treated in *2.3.1 Fluid Dynamics*. The figure below is the default function for global fluid-fluid interaction. In this example, the values are clamped between the maximum (5) and the minimum (0), if the distance between the two tested particle exceeds 1, there is no force.

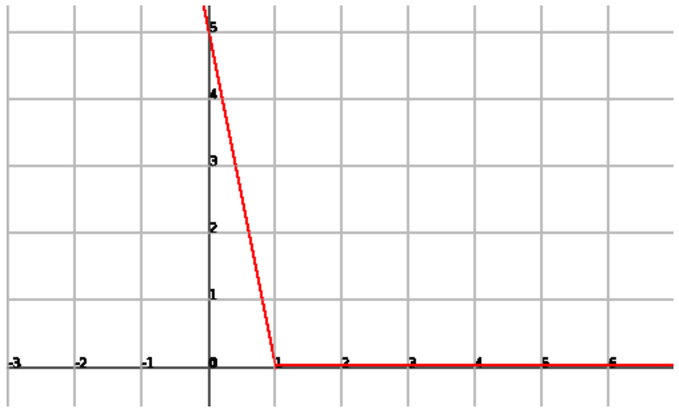


Figure 2.3.4: Fluid-Fluid interaction

2.3.3.c Object-Object interaction

The object-object interaction happens only when two particles belong to different objects. The collision response is based on impulses and this subject has been explained in the previous section called *2.3.2 Rigid Body Dynamics*.

2.3.3.d Improvements

The attractive and repulsive forces between different materials and objects are managed by global functions. A substance behaves differently when mixing with other fluids and the fluid-fluid interaction or fluid-object interaction may be unique for every type of particles. For instance the pair honey-oil flows differently than honey-water. As explained in the previous part, a dictionary of interactions can assign for each possibilities a specific function. This dictionary has to be created by the artist but this system is tedious to maintain because for every new material, it should assign a specific function for existing materials.

Interactions among different matters are accomplished by using constant repel forces. Unfortunately this concept doesn't take into account the relative velocity of each object. The relative velocity is the vector difference between the velocities of two entities. Two approaches are possible to fix this problem. One is to use impulses and consider particle fluid has a rigid body dynamic. The second option consists in adding a proportional amount of force according to the relative velocity of the two entities.

2.3.4 Numerical Integration

Explicit euler integration is not stable with large steps but it gives an accurate result with a small time step. For rigid body, it is ”tricky” to use verlet integration because the collision system is based on impulses to change the velocity but verlet doesnt have an explicit velocity which is expressed by the subtraction between the current and the previous position divided by the time step.

In this project, the time step is small and constant and by running a trial and error test, the computation of 8 sub-steps per frame gives a good result in most of the cases. The time step is related to the speed of the fastest object and of its size. In fact, an object going to fast may go out of the boundaries or go through other objects. The time step is kept small and RK2 (mid-point), RK4 or other higher order methods are expensive to integrate the system, therefore, the simple explicit euler integration can guarantee a fast and accurate computation, hence it is the integration used in the project.

2.4 Rendering

2.4.1 Implicit surfaces

The use of implicit surfaces with particle system is common and popular in computer graphics [3]. Implicit surfaces use implicit functions for their representations. Metaballs or blobby surfaces are implicit objects where every point emits fields which are sumed up into a volume of data. Finally, the marching cubes algorithm constructs an isosurface from this volume of data. The figure below shows four particles wrapped into their common isosurface.

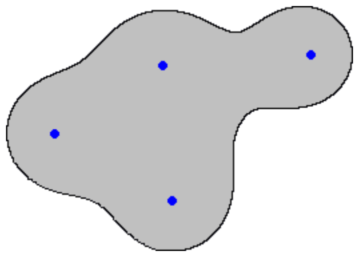


Figure 2.4.1: Implicit surfaces

2.4.2 Volume preservation

As explained in the previous section, the substance surface is an isosurface, unfortunately, an implicit function or surface can significantly alters the variation of its volume. In Desbrun's paper called "Animating soft substances with implicit surfaces", the local volume control is adjusted to maintain a constant volume across the whole surface.

The example below demonstrates the problem. During the fusion of two particles the global volume increases significantly. The first picture describes the initial positions, the second picture shows an 41% increase of the volume after merging and finally the third image with volume preservation shows a constant volume.

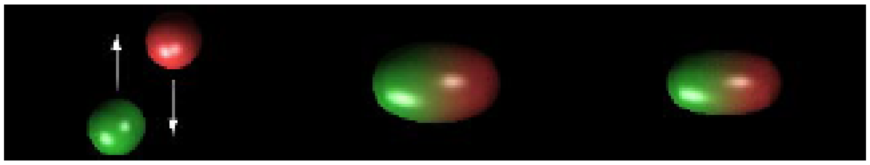


Figure 2.4.2: Volume control - Desbrun's example

The volume control is maintained by the addition of an extra value on the field function for each particle describing the isosurface. The extra value is proportional to the distance between the pair of particles and decreases according to their separation. It results into a reduction of the field intensity and by consequence the isosurface shrinks to obtain an approximation of the desired intensity. The decreasing function needs to be chosen carefully to balance and offset the unneeded volume.

3 Practical : Design and Implementation

This chapter explains the design, data structures and flows involved in the implementation of the project. It also discusses and give a range of options to develop this type of system.

3.1 Different options

Different options and approaches exist to develop such system as a maya plugin. The technique needs to be flexible enough to access the various variables and able to work across a network / renderfarm for future improvement.

3.1.1 nCloth and nNucleus

The concept of the Maya Nucleus framework is based on simple computational models and treats the interaction of elements as a system of particles that collide and exert forces on one another. This idea is exactly the purpose of this paper and creates an unified framework for particle-particle interactions.

Duncan's nCloth is a perfect example of using nCloth for simulating water. The explanations of his work are available on this website : *http://area.autodesk.com//blogs//duncan//water_using_ncloth_part_1_of_2*. It's also possible to customize solvers for the particle motion.

Unfortunately several disadvantages arise from Maya API and its limitations for such implementation. The problem comes from having a clear and reusable code and the rendering issue for Maya. By consequence, this option has not been considered for the final implementation.

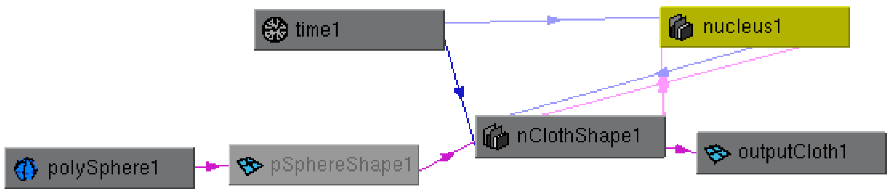


Figure 3.1.1: Nucleus

3.1.2 MPxFieldNode

The second approach consists in using MPxFieldNode class which is the base class for user defined fields. It inputs positions and velocities from particles and apply a force on each particle. There is no constraint the way forces are calculated and it's possible to access any object in the scene such as geometries, emitters, etc.

It is easy to implement and the customized field is obviously compatible with Maya's particle system. For instance, the creation of a field may be useful to set forces for every particle to keep them apart and mimic soft substances. Moreover all the physics and integrations are already implemented as part of Maya package by consequence there is no need to develop integrators or other features related to dynamics.

Unfortunately, there is no way to alter particle's data such as position or velocity in direct access. If a particle has to end up at a specific position, it's necessary to calculate the force which will be needed to have the particle at that position the next frame. Finally, this technique can be used for a quick solution but it is not suitable for this study.

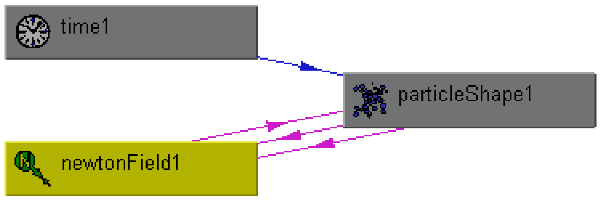


Figure 3.1.2: MPxFieldNode

3.1.3 External application

Another possibility is to create and render the scene in Maya, with the simulation happening into an external application. There is no limitation on the access of the data and a translator will be the bridge between Maya and this application. Moreover, creating an application is less restrictive, easier to debug and implement. This solution is the one selected for the project.

3.2 Design

3.2.1 OpenGL Framework design

The framework is the core of the system and manages assets by a set of managers for geometries, objects, textures, rendering etc. It's not related to any dynamics and is the reusable part of the application for future application. This framework has been used for hair simulation, rigid body dynamics and soft bodies.

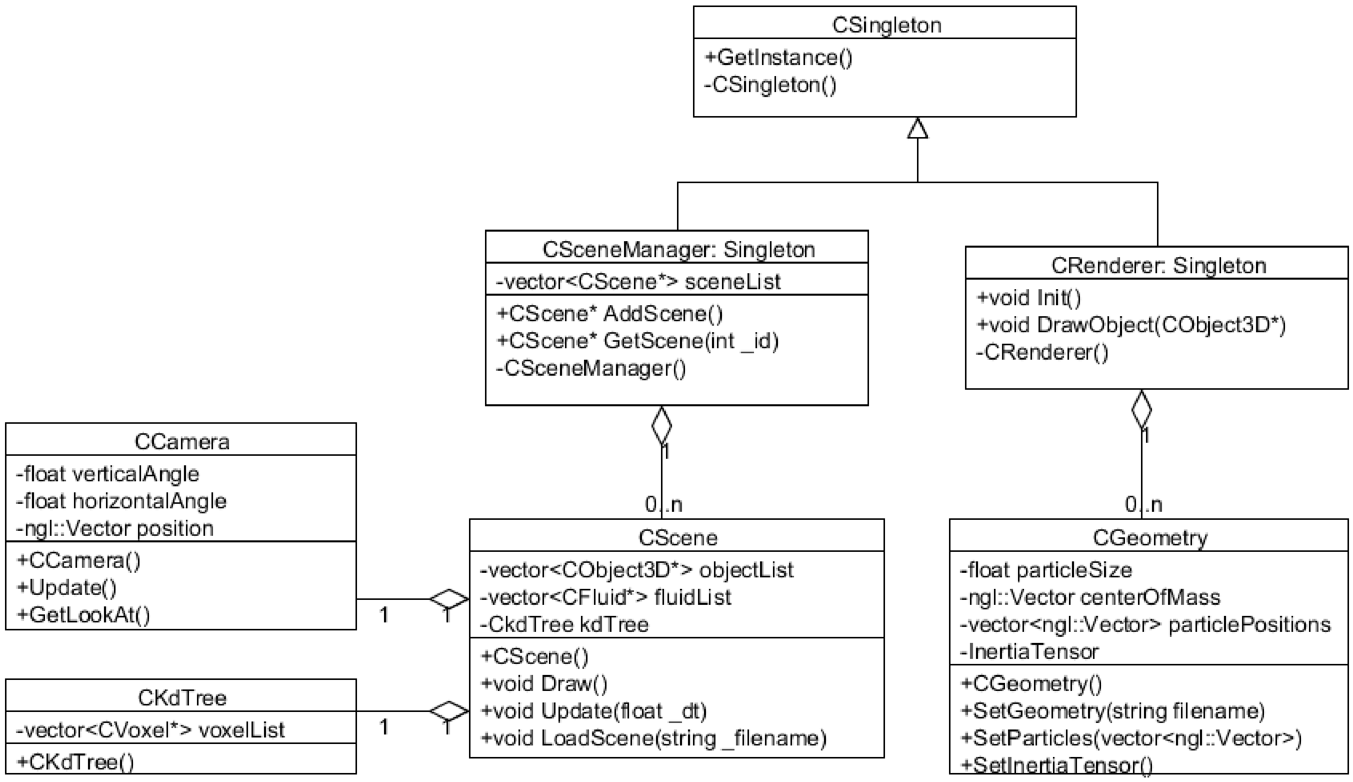


Figure 3.2.1: OpenGL framework 1

- *CSingleton* is the abstract class to develop a singleton class.
- *CScene* class is in charge of the lifetime of the objects and fluids. It creates, stores, updates and deletes the elements in the scene.
- *CRenderer* renders a scene. Instead of storing the geometry into the object, a dictionary of geometries manages the meshes. This technique allows to share a same mesh among different objects.
- *CGeometry* class stores a geometry mesh and the type of shape (sphere, cone, cylinder or box) to describe the distribution of the mass across the volume for dynamics purpose. Those information are used to compute an approximation of the inertia tensor for an object. The inertia tensor is not calculated into the geometry because its related to the mass and the size of the object. An object with a geometry, an uniform scale and a mass, can computes its inertia tensor. NCCABinMesh class has been used to speed-up the loading of the geometries. Those bin files are created if there is no related bin file to the obj file. The second time the application is running the geometry is using the bin file and not the obj.

This second part of the OpenGL framework is related to dynamics and optimizations.

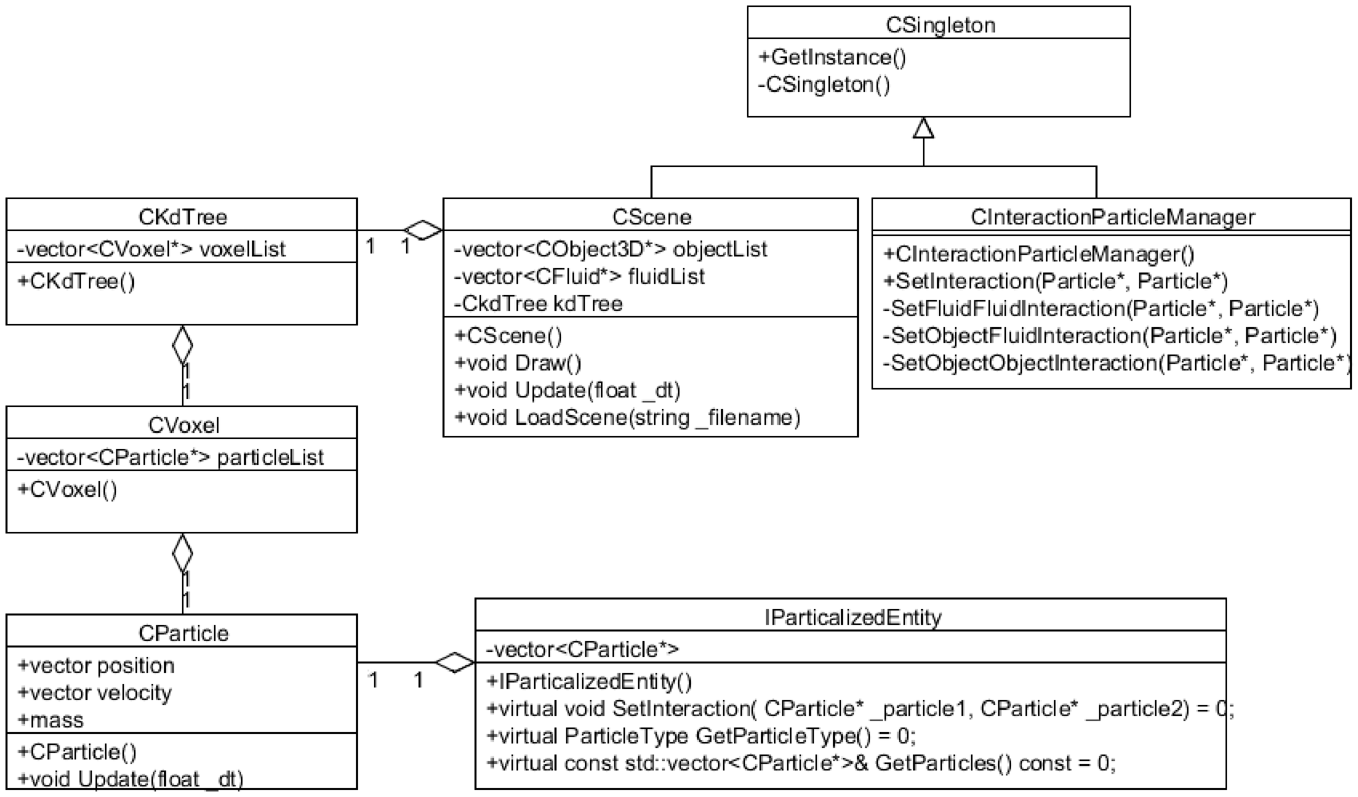

CSingleton
+GetInstance()
-CSingleton()
CKdTree
-vector<CVoxel*> voxelList
+CKdTree()
CScene
-vector<CObject3D*> objectList
-vector<CFluid*> fluidList
-CkdTree kdTree
+CScene()
+void Draw()
+void Update(float _dt)
+void LoadScene(string _filename)
CInteractionParticleManager
+CInteractionParticleManager()
+SetInteraction(Particle*, Particle*)
-SetFluidFluidInteraction(Particle*, Particle*)
-SetObjectFluidInteraction(Particle*, Particle*)
-SetObjectObjectInteraction(Particle*, Particle*)
CVoxel
-vector<CParticle*> particleList
+CVoxel()
CParticle
+vector position
+vector velocity
+mass
+CParticle()
+void Update(float _dt)
IParticalizedEntity
-vector<CParticle*>
+IParticalizedEntity()
+virtual void SetInteraction(CParticle* _particle1, CParticle* _particle2) = 0;
+virtual ParticleType GetParticleType() = 0;
+virtual const std::vector<CParticle*>& GetParticles() const = 0;
1
1

Figure 3.2.2: OpenGL framework 2

- *CKdTree* is a data structure for neighbour search optimizations. It stores a map of voxels identified by an unique identifier.
- *CVoxel* stores particles and defines a specific area in three-dimensional world space.
- *IParticalizedEntity* is an abstract class which is a base for any entity made of particles. The class CObject3D and CFluid inherit from IParticalizedEntity.
- *CParticle* is defined by its mass, position and velocity. The initial state (starting position and velocity) is also attached to the particle for restarting the simulation. The particle data structure also stores data related to rendering such as colour, size etc.
- *CInteractionDictionnary* is a singleton storing the various interactions between different type of materials. Two global functions for fluid/fluid interactions and object/fluid interactions are embedded into the class. This class is the ”hub” for the creation of new particle/particle interactions.

3.2.2 Rigid body design

The physics part for rigid body dynamics is in charge of moving the objects and integrating the system according to the gravity, external forces, impulses and object's materials.

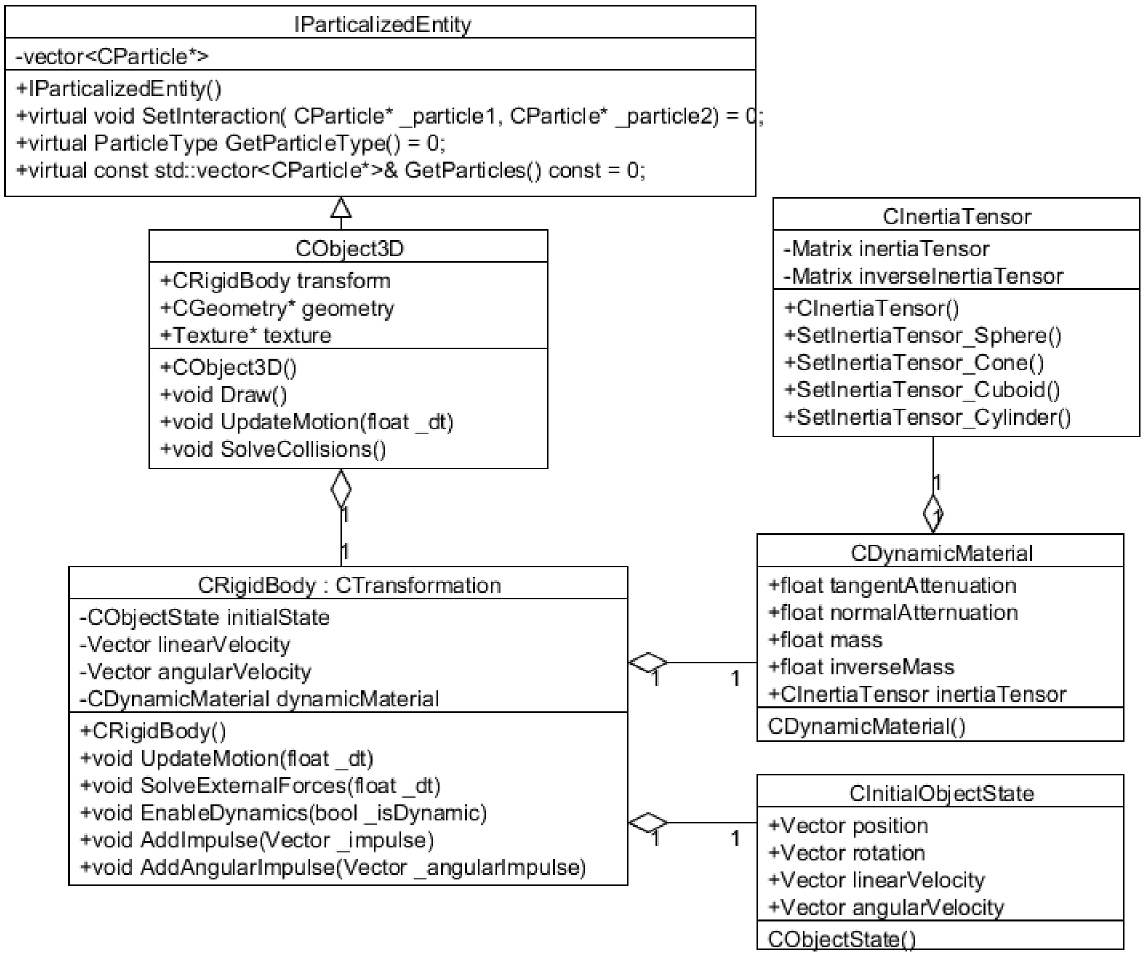

IParticalizedEntity
-vector<CParticle*>
+IParticalizedEntity()
+virtual void SetInteraction(CParticle* _particle1, CParticle* _particle2) = 0;
+virtual ParticleType GetParticleType() = 0;
+virtual const std::vector<CParticle*>& GetParticles() const = 0;
CObject3D
+CRigidBody transform
+CGeometry* geometry
+Texture* texture
+CObject3D()
+void Draw()
+void UpdateMotion(float _dt)
+void SolveCollisions()
CInertiaTensor
-Matrix inertiaTensor
-Matrix inverseInertiaTensor
+CInertiaTensor()
+SetInertiaTensor_Sphere()
+SetInertiaTensor_Cone()
+SetInertiaTensor_Cuboid()
+SetInertiaTensor_Cylinder()
CRigidBody : CTransformation
-CObjectState initialState
-Vector linearVelocity
-Vector angularVelocity
-CDynamicMaterial dynamicMaterial
+CRigidBody()
+void UpdateMotion(float _dt)
+void SolveExternalForces(float _dt)
+void EnableDynamics(bool _isDynamic)
+void AddImpulse(Vector _impulse)
+void AddAngularImpulse(Vector _angularImpulse)
CDynamicMaterial
+float tangentAttenuation
+float normalAtternuation
+float mass
+float inverseMass
+CInertiaTensor inertiaTensor
CDynamicMaterial()
CInitialObjectState
+Vector position
+Vector rotation
+Vector linearVelocity
+Vector angularVelocity
CObjectState()
1
1
1
1
1
1
1
1

Figure 3.2.3: Rigid body dynamics

- *CRigidBody* class inherits from the transformation class and defines the dynamic transformation of a rigid body. It contains a linear and angular velocity, initial state (state at the beginning of the simulation) and information about its material.
- *CInertiaTensor* class describes the inertia tensor of an object. It is approximated by using the inertia tensor of simple primitives (sphere / cone / cuboid and cylinder).
- *CDynamicMaterial* class represents the material of an object for a dynamic purpose. It contains the coefficient of restitution (normal attenuation), the coefficient of friction (tangent attenuation) and information related to the mass such as mass, the inverse of the mass, the inertia tensor and the inverse of the inertia tensor.
- *CObjectState* Class represents the initial state of an object (position, rotation, velocity and angular velocity).

3.2.3 Fluid design

The physics part for fluids is in charge of simulating an unique substance according to the gravity and the fluid property.

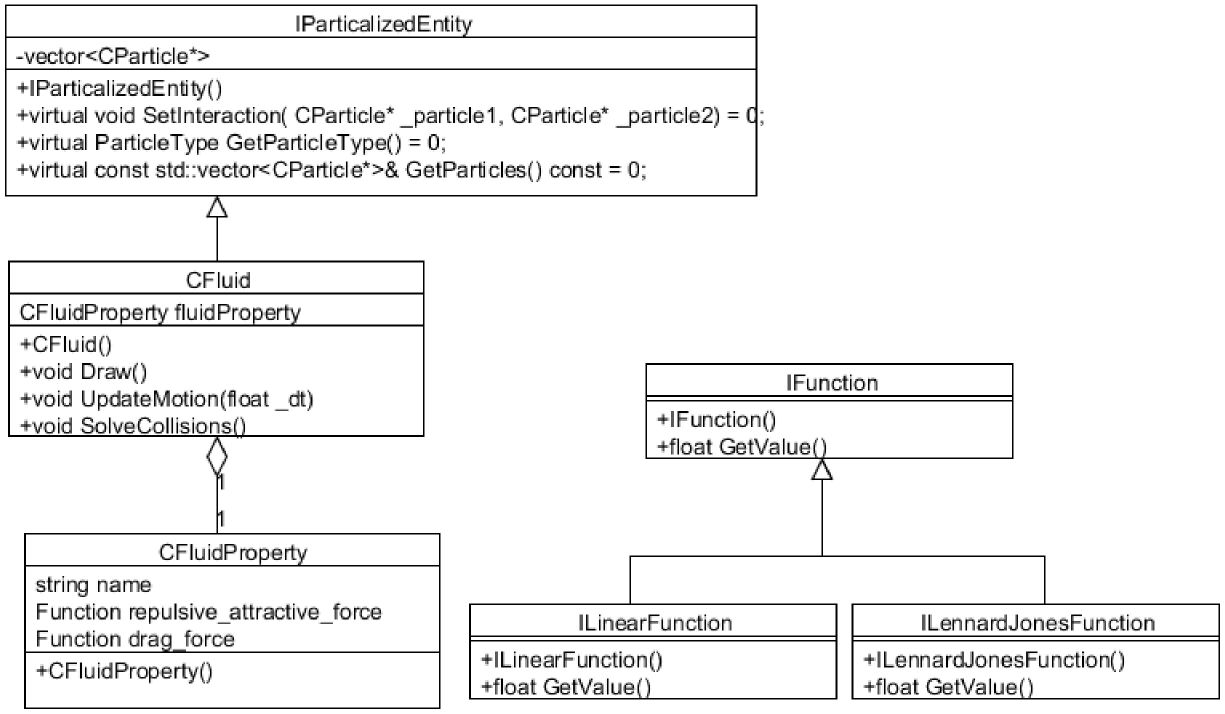


Figure 3.2.4: Fluid and particles

- *CFluid* inherits from IParticalizedEntity and describes a specific volume of an unique substance.
- *CFluidProperty* defines the different forces involved inside the fluid itself with the repulsive/attractive function and the viscosity function. It doesnt describe the interaction with external objects or fluids.
- *IFunction* is an abstract class to develop mathematical functions.
- *ILennardJonesFunction* and *ILinearFunction* inherit from IFunction and respectively implement the Lennard-Jones potential and linear function.

3.3 Simulation flow

The algorithms is based on 5 levels.

The first step consists in integrating the objects and fluids motions without taking into account collisions. This system is a post-collision which means collision responses are calculated when entities (objects / fluids) are already inter-penetrating.

The second step is the creation of the KdTree and builds the hierarchy to store all the particles to optimize the neighbour search process. This part is explained in the following section *3.4.1 Space partitioning*.

The third step parses the particles in the KdTree and cumulates the external forces from their neighbours. The interaction particle manager is in charge of calculating a force according to the position and the type of two particles.

The fourth and fifth steps integrate impulses, forces and fix rigid body inter-penetrations.

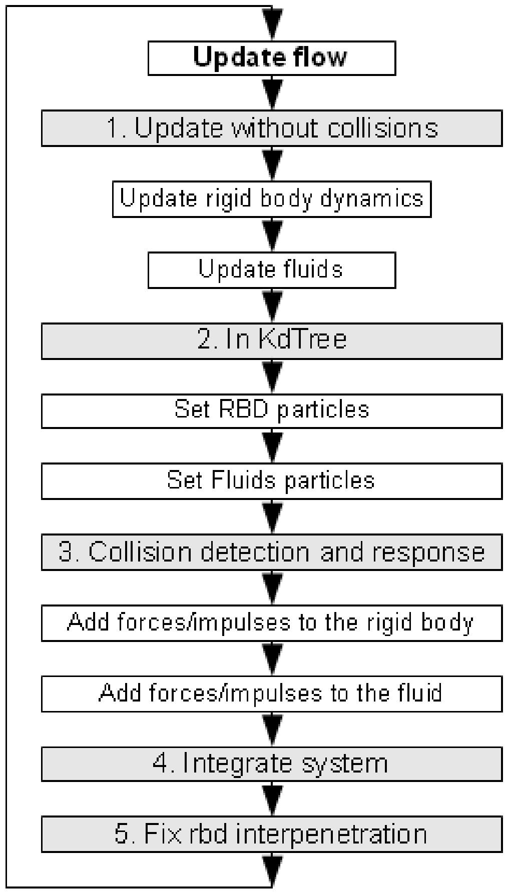

Update flow
1. Update without collisions
Update rigid body dynamics
Update fluids
2. In KdTree
Set RBD particles
Set Fluids particles
3. Collision detection and response
Add forces/impulses to the rigid body
Add forces/impulses to the fluid
4. Integrate system
5. Fix rbd interpenetration

Figure 3.3.1: Simulation flow

3.4 Optimisations

A fast simulation with many particles cannot be achieved without optimized structures and ”tricks”. Three main optimizations respectively dealing on the structure and the force calculation have been implemented to guarantee the speed of the simulation.

3.4.1 Space partitioning

The main optimisation comes from the 3D spatial partitioning where every voxels (or grid cells) are indexed into a KdTree for a fast neighbour search. The detection of the neighbours is a critical point since most of the computation for a single particle uses its surrounded particles. A brute force algorithm without optimizations has a O(n^2) complexity.

Fortunately, particles have a limited range of influence, this property allows the voxelising of the space into buckets or voxels. A regular grid is not the best choice because it creates an important memory overhead by allocating unused space. In 2004, Steels demonstrates a nearest neighbour search technique by using a hash table to split the space [9]. A similar technique has been developed into this project by allocating only voxels where the particles exist.

The size of a voxel has to be twice the size of the largest area of influence. The KdTree is resized every frame to confine all the particles into its boundaries. The voxel index $i = (ix, iy, iz)$ of the particle is defined by : $i = \frac{PositionWs - min}{VoxelSize}$ where $PositionWs$ is the position of the particle in the world space, min is the lowest corner of the KdTree and $VoxelSize$ is the voxel size. During the neighbour search of a specific particle, only the voxel holding this particle and its 26 neighbours are tested, the complexity drops from O(n^2) to O(n).

The figure below shows nine allocated voxels (grey colour) with a tested particle (blue colour) and its neighbour (red colour).

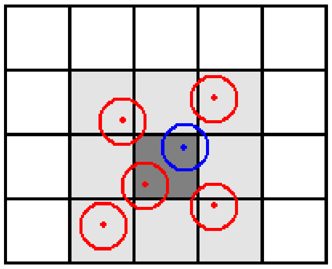


Figure 3.4.1: Regular grid and particles

The comparison table below matches with the theoretical expected results. In fact without spatial subdivision it takes 65 more times to compute a frame with 10 times more particles. With optimizations it takes 10.83 more times to compute a scene with 10 times the complexity, hence with spatial structure, the algorithm has a linear complexity.

-	With KdTree	Without KdTree	speed factor
1000 particles	0.012 sec	0.017 sec	x 1.42
5000 particles	0.07 sec	0.29 sec	x 4.14
10000 particles	0.13 sec	1.12 sec	x 8.61

3.4.2 Function sampling

Mathematical functions for forces are pre-calculated and sampled into an array. It reduces the computational cost of the forces by converting functions into a list of linear functions. The Lennard-Jones function with a power of 12 and a power of 6 benefits the most from this optimization and increases the overall performances of the application.

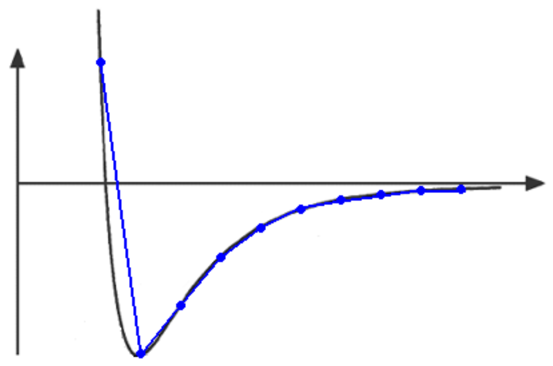


Figure 3.4.2: Sampling a function

The tests below demonstrate the efficiency and gain of performance for 1000, 5000 and 10.000 particles. The gain of performance increases with the number of particles. With 1000 particles the simulation speed is increased by 1.17, with 10.000 particles the speed is improved by 1.31.

-	With Sampling	Without Sampling	speed factor
1000 particles	0.012 sec	0.014 sec	x 1.17
5000 particles	0.07 sec	0.083 sec	x 1.19
10000 particles	0.13 sec	0.17 sec	x 1.31

3.4.3 Third Newton's law

"For every action there is an equal and opposite reaction." - Third Newton's law.

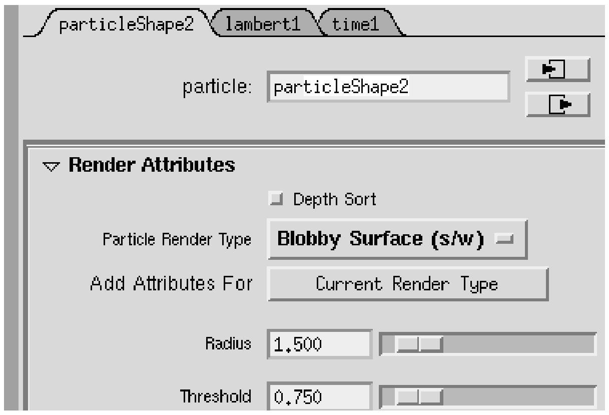
In physics-based animation, it's possible to take advantages of the third law of Newton and it will divide by two the computational cost of the simulation. In this study, particles are interchanging forces, hence when a particle A acts on a particle B with a force F, the particle B acts on the particle A with an opposite force -F. This technique avoids a redundant calculations and double the efficiency of the application.

3.5 Maya - Fluid rendering

The rendering of the fluid is done by using the default blobby surface provided by Maya's particle system.

3.5.1 Blobby surface

A blobby surface is an iso-surface defined by the function f(x,y,z) = c, where x,y and z are the three-dimensional coordinates and c is the force of the field. The original idea was to develop the blobby surface in Renderman, but for the time being, the native blobby surface of Maya has been the final choice.



particle: particleShape2

Depth Sort

Particle Render Type: Blobby Surface (s/w)

Add Attributes For: Current Render Type

Radius: 1.500

Threshold: 0.750

Figure 3.5.1: Maya - Blobby surface

The iso-surface is only used for rendering and is not involved in collision detection or other aspects of the simulation. In fact, particles are pre-visualised in OpenGL as simple spheres, by consequence it's important to have an iso-surface which fits perfectly with the particle surface. Maya blobby surfaces easily fulfil this requirement by giving a blobby surface radius 50% bigger than the particle diameter and a threshold twice smaller than the blobby surface. For instance, a particle diameter of 1, the blobby surface radius should be 1.5 and the threshold equals to 0.75. This rule gives a convincing result and has been followed by all the renders.

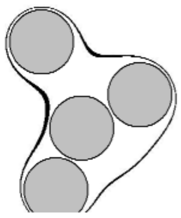


Figure 3.5.2: Particles and isosurface

3.5.2 Shading

For rendering, the native Maya ocean shader provided an excellent result after changing few default parameters. It is possible to obtain a good-looking substance with a subtle transparency and the enabling of the ray-tracing for refraction or reflection effects. Moreover a slightly waving effect add significant details especially for a fluid in low resolution with large particle size.

4 Pipeline

The pipeline is based on Maya and this chapter explains the different components and classes implemented for the plugin and the entire pipeline.

4.1 Global pipeline

The pipeline is composed of three phases.

1. After creating a scene in Maya with particles and rigid body objects, the scene is saved into a scene file (.scene). This file is compatible with the external application simulating the animation. *See 4.3.2 File Formats*.

2. The application simulates every frame and backups the animation into a set of simulated files (.sim). *See 4.3.2 File Formats*.

3. Finally, the scene file is loaded into Maya and the entire simulation is imported by keyframing rigid body objects and writing the particle cache disk files (.pdc). *See 4.4.3 File translator*.

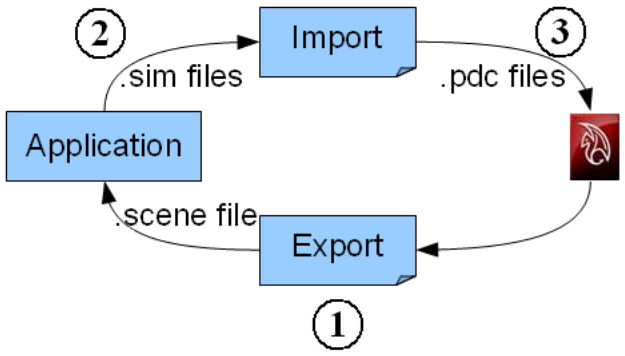


Figure 4.1.1: Pipeline flow

4.2 User interface

The user interface of the plugin is split into two parts, the shelf and the main window. After loading the plugin, the ”SoftSubstances” shelf appears with new functionalities. This section describes the various user interface embedded in the plugin to create and test a scene.

4.2.1 Shelf

The shelf is composed of five features :

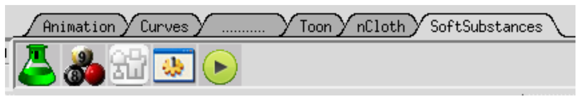


Figure 4.2.1: User interface : shelf

The selected particles are converted into soft substances by adding extra variables in the particleShape.

The selected geometries are converted into rigid body by adding extra variables.

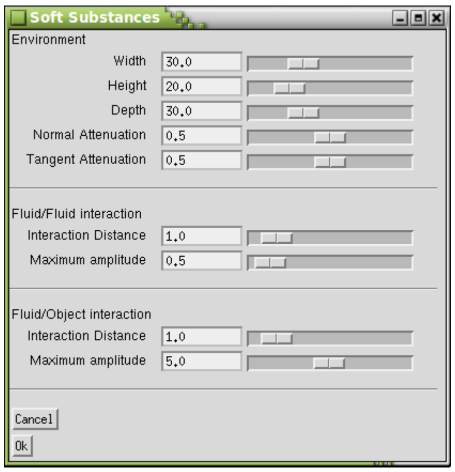
This module generates a regular set of particles that fill the selected rigid body objects.

Opens the options for the scene configuration such as environment size, global interaction functions etc.

Run the simulation on the external application. *This command doesn't work due to incompatibility with libstdc++ of Maya and the external application !*

4.2.2 Global parameters

The environment settings are available in the main user interface. Environment size and common dynamic variables are displayed. The two global interaction functions for fluid-fluid and fluid-object are accessible from this window. By pressing the cancel button, the latest configuration is restored.



Soft Substances

Environment

Width	30.0
Height	20.0
Depth	30.0
Normal Attenuation	0.5
Tangent Attenuation	0.5

Fluid/Fluid interaction

Interaction Distance	1.0
Maximum amplitude	0.5

Fluid/Object interaction

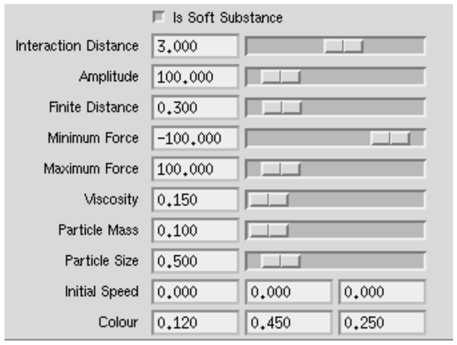
Interaction Distance	1.0
Maximum amplitude	5.0

Cancel
Ok

Figure 4.2.2: User interface : global parameters

4.2.3 Substances parameters

A particle system is considered as a soft substance by its additional variables into its shape. The boolean parameter called *"Is soft substance"* guarantees the Maya particles to be a substance. The five first variables refer to the Lennard-Jones function in charge of keeping particles apart. The viscosity damps the velocity to limit the motion of a particle into its fluid. The colour and size of the particle are only used for the OpenGL pre-visualization and will not affect the final rendering.



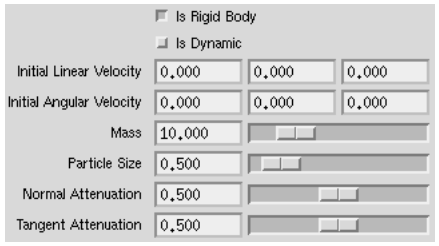
Is Soft Substance

Interaction Distance	3.000		
Amplitude	100.000		
Finite Distance	0.300		
Minimum Force	-100.000		
Maximum Force	100.000		
Viscosity	0.150		
Particle Mass	0.100		
Particle Size	0.500		
Initial Speed	0.000	0.000	0.000
Colour	0.120	0.450	0.250

Figure 4.2.3: User interface : substance parameters

4.2.4 RBD parameters

An object behaves as rigid body (static/dynamics) when it owns extra variables related to RBD. The boolean called *"Is rigid body"* defines the rigid body dynamics is enabled. The other variables store the initial state (linear and angular velocity) and dynamic material (mass, normal attenuation, tangent attenuation). The initial transformation of the object is extracted from its transformation.



Is Rigid Body

Is Dynamic

Initial Linear Velocity	0.000	0.000	0.000
Initial Angular Velocity	0.000	0.000	0.000
Mass	10.000		
Particle Size	0.500		
Normal Attenuation	0.500		
Tangent Attenuation	0.500		

Figure 4.2.4: User interface : RBD parameters

4.3 File System

4.3.1 Folders

Two different folders are automatically created :

The geometry folder named : *sceneName + "Geo"* stores the geometries of the scene. It contains the obj files written by the plugin and the NCCAbin files created by the application during the loading of a scene.

The simulation folder named *sceneName + "Sim"* stores the simulation files. It contains one simulation file per recorded frame and is generated by the external application when the simulation is completed.

For instance with a scene file named masterScene.scene, the simulation and geometry folders are respectively called masterSceneSim and masterSceneGeo. The example below shows two scene files with their associated folders. This hierarchy allows different scenes to exist in a same folder.

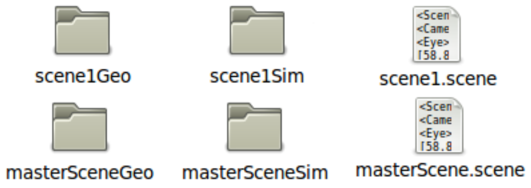


Figure 4.3.1: File system : Folders

4.3.2 File Formats

Two different file formats are used in the pipeline :

The first file format defines a scene, where objects and fluids are placed in space, the way they interact between each other etc. Its extension is ".scene". *See Appendix - Scene format*.

The second file format has an extension ".sim". A file is created per recorded frame and contains the keyframed transformations. For an object, it stores an uniform scale, position and rotation, for a particle system only the particle's positions are stored. *See Appendix-Simulation format*.

4.4 Plugin

The plugin is developed in C++ and provides new functions to set fluids, rigid body dynamics, saves and loads a scene. This section describes the design and implementation of the maya plugin developed for the pipeline of the project.

4.4.1 Design

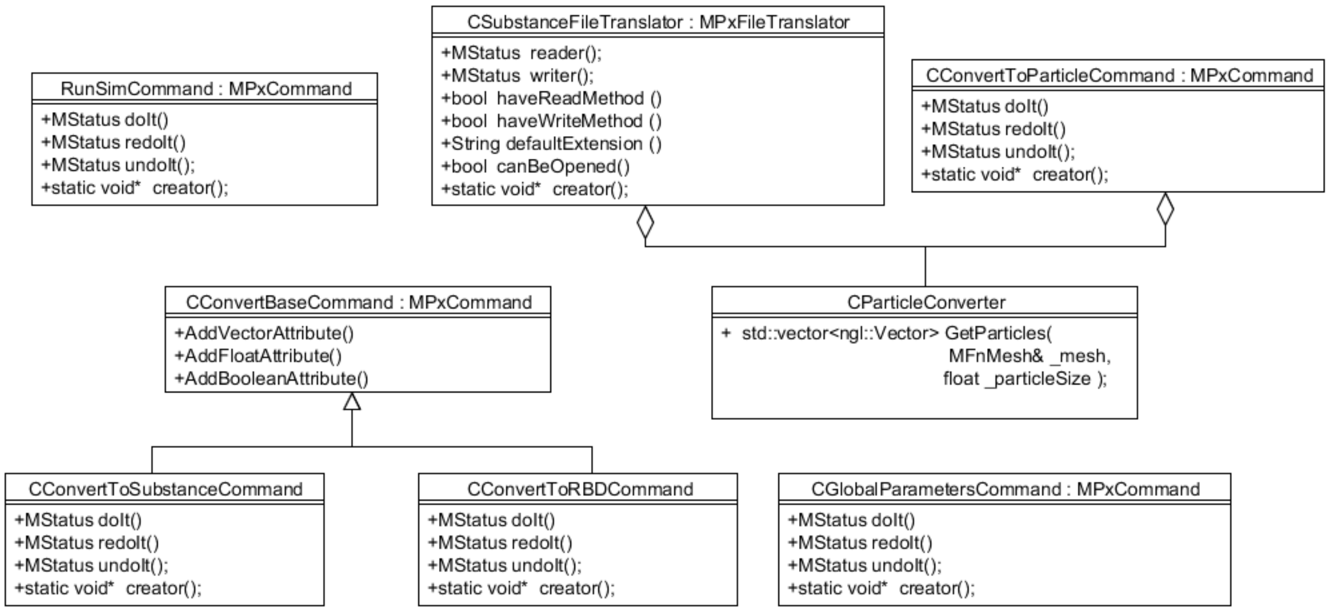


Figure 4.4.1: Plugin - design

4.4.2 Commands

It exists fives different commands:

- *CConvertBaseCommand* is the base command to add extra attributes in Maya nodes. Three functions respectively create vector, float and boolean attributes.
- *CConvertToSubstanceCommand* inherits from CConvertBaseCommand and calls mel script functions to add extra attributes to the selected particle systems.
- *CConvertToRBDCommand* inherits from CConvertBaseCommand and add extra attributes to the selected geometries to set rigid body objects.
- *CGlobalParametersCommand* is a command to open the window of the scene settings. The user interface is implemented in melscript. Getters have been developed in melscript to access variables from the Maya plugin. Since 2010, Maya integrates support for QT which offer a better and cleaner way to develop user interface in Maya.
- *CParticleConverter* is a static class which generates a regular set of particles that fill a given mesh according to specified particle size.
- *RunSimCommand* is a command to execute the external application and run the simulation.

4.4.3 File translator

A file translator has been created to load and save .scene file. It's a xml file format containing the information of a scene, which can be read back in the external application for the simulation.

During the saving process, the plugin writes the scene file by parsing all the selected objects and particle systems which have been previously converted into rigid bodies or soft substances. Geometries are triangulated on fly and saved into the geometry folder as .obj files. For the substances, data are extracted from the particle systems and written into the xml scene file.

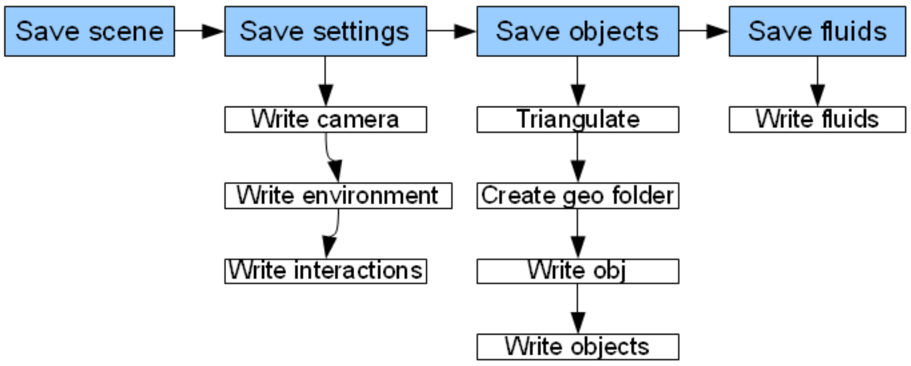


Figure 4.4.2: Saving process flow

During the loading process, the plugin checks the existence of the simulation folder. If the folder exists, each simulation file representing a single frame is opened one at a time. The objects are keyframed every frame. The loading of the fluid is more involved; PDC files, created during the preparation of the particle cache, are overwritten by the PCDWriter class. A PDC file represents a particle system at a specific frame and the naming convention is : particleShapeName + (250 * FrameID). For instance, a particle shape called particle1Shape at a frame 10 is called "particle1Shape_2500.pdc". *See Appendix - Maya PDC format*.

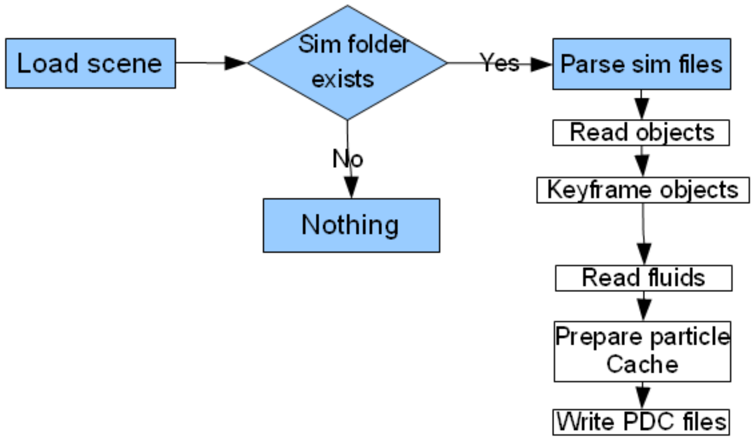


Figure 4.4.3: Loading process flow

4.5 Guide and instructions

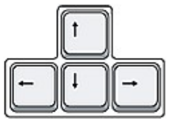
4.5.1 Executable

The executable accepts various arguments for running a simulation. The table below describes the arguments and default values available for the application.

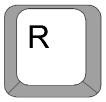
Arguments	Default value	Description
-f	scene1.scene	Scene file to load.
-record	1	Boolean to record the simulation or not.
-updateNum	8	Number of update per frame.
-drawPerFrame	1	Draw every frame or every update.
-stopRecordingAt	-1	Stop recording after a specific frame.

4.5.2 Keys

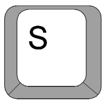
This section explains how to use the application which computes the simulation. By default the application is in *record mode*, when the calculation of a frame is completed, its state is stored in memory. By pressing the key *R*, the animation writes every frame in simulation file.


The camera is moved using the arrow keys. The up and down keys respectively move the camera forward and backward. The right and left keys turn the camera on the right or left side.

By pressing the left mouse key and moving the mouse, it gives a similar result than pressing the right and left keys but it is more handy.



The key R restarts the simulation. It resets the scene at the initial frame.



If the application is in *record mode* the simulation files are created after pressing the key S.

5 Results and Discussion

After the complete implementation, the application has been tested with a wide range of simulations to analyse the possibilities offered by this new approach. The tests gave convincing results for soft substances and highly viscous fluid but the application also offers good results for viscous-less fluid and other type of materials which were not considered at the beginning of the study.

Interactions: During this study, various experiments have been carried out and the substance motion is believable and provides a fast pre-visualisation. Since the treatment of inter-particle interactions are handled by basic functions, it's really easy to test and extend the technique. Moreover the small amount of parameters is an asset for the developer, in fact by playing with few parameters it is possible to achieve a wide range of effects from soft substance, viscous fluid to plasticity effect. The example below is running with 1000 particles and demonstrates the different combinations for inter-particle interactions (fluid/fluid and object/fluid).

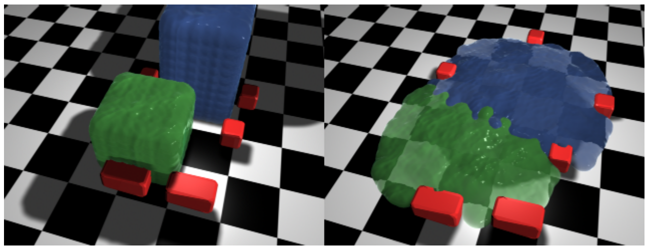


Figure 5.0.1: Result : Several interactions

Highly viscous substances: Other tests have been carried out on the "solidity" and viscosity of a substance. The strength is defined by the area of influence, larger is the influence area, more particle neighbours are taken into account and by consequence more compact the substance will be. On the other hand, if the influence area is small, it results into less interactions which, gives a less compact substance. The second parameter is the damping of the particle velocity into its own fluid, in this application this variable is called "viscosity". In the following example three substances made of 5000 particles are dropped, the blue one is smashed against the two others and the red posts surrounding them.

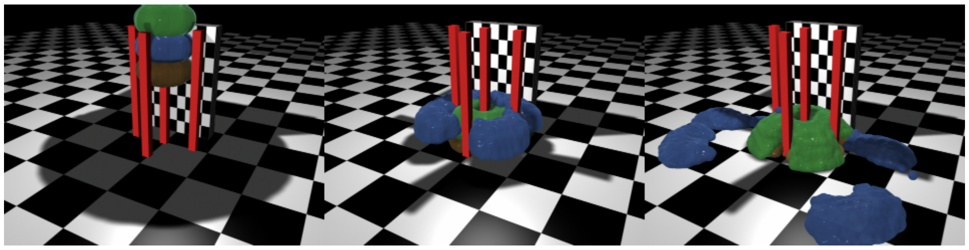


Figure 5.0.2: Result : Highly viscous substances

Large deformable object: Other experiments have been completed to create ”objects” which deforms according to the environment. The figure below is a perfect example, where a wall is deformed by high speed objects going toward it. The simulation is fast and only 5000 particles constitute the wall.

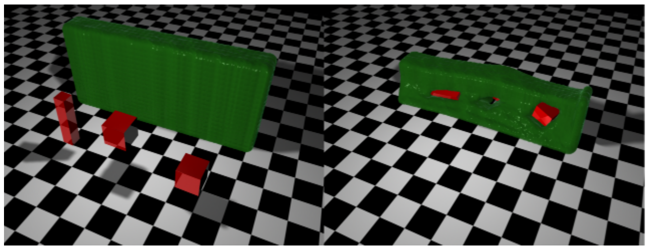


Figure 5.0.3: Result : Large deformation object

Beyond the study: This technique is not only limited to soft and viscous substances but can be extended for a high range of effect such viscous-less fluid, plasticity or solid objects. Those two following examples demonstrate the possibility for the application to deal with a large number of particles (50.000 and 200.000 particles). Moreover, substances are much more fluid-like than the rest of the previous examples. The damping and the area of influence have been kept low to limit the motion of the particles inside their own fluid.

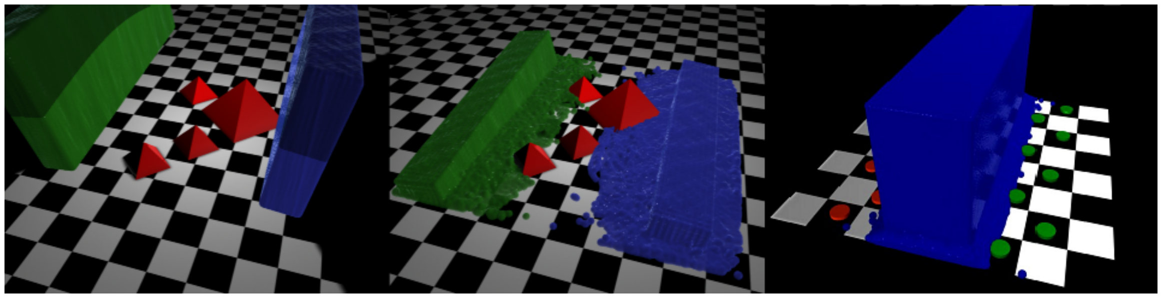


Figure 5.0.4: Result : Other effects

Performance and stability: The performance of the technique obviously depends on the number of particles which are running and the parameters. Due to the small amount of computation per particles, this new approach gives a new tool for the artists to create highly viscous substances. The stability is also dependent to the simulation. For instance a high damping to simulate viscous fluids needs more sub-steps per frame to maintain the stability. The figure below demonstrates the stability of the application, a fluid resisting deformation has been created to hold an object on its top. Four updates were calculated per frame and the simulation went wrong after 10 seconds because the involved forces were to high. Except in this specific case, the application is stable.

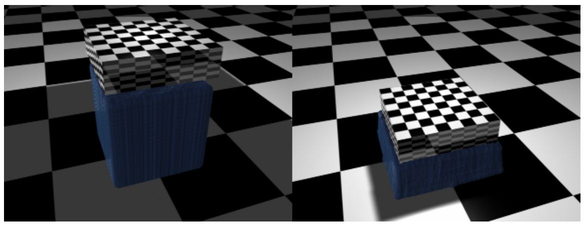


Figure 5.0.5: Result : Stability

6 Conclusion

All this project has been an experiment and explore new ways to create effects using functions describing a behaviour between two particles. Fluid simulation in visual effects and games can benefit from this concept. This study draws a better picture of ”function-based simulation” techniques and describes a general concept which can be extended in the future. This technique is not suitable for physical correct simulation but can be highly considered if the ultimate goal is the visual look and not the accuracy of the values. This technique can be recommended for high viscous fluid, which guarantee a stable and simple way to start.

6.1 Comparisons

This approach is not based on physics then the accuracy of the physics simulation cannot be compared with classic particle-based fluid (for instance smoothed-particle hydrodynamics) relying on Navier-Stokes equations or other fluid theories which offer an accurate dynamic model to track the trajectories of fluid particles. This technique is not suitable to create water-like fluid where splashes can occurs. This technique is suitable for substances such as dough, mud, jelly etc. Moreover, on the contrary to other simulation heavily based on physics, the parameters are easier to change and more intuitive because the interactions are only based on functions representing forces. This technique is also much faster than classic fluid solvers especially when a fluid is made of a large number of particles.

6.2 Future improvements

Flocking system, fluid simulation, hair simulation, rigid body have been a long study in computer graphics, this concept can be a new area of discovery for computer graphics in digital entertainment.

Some parts of the application can greatly be improved:

- Rotation : The rotations of the dynamic objects are causing problems when interacting with other objects. This part is related with rigid bodies dynamics.
- Relative velocity : As mentioned previously, the forces between two entities are based on functions and not on the relative velocity. The addition of a scalar to take into account the relative velocity can greatly helps the realism of the simulation.
- Multi-threading and GPGPU : All the objects can be updated independently to each other, this algorithm can easily be multi-threaded on CPU, GPU or across a renderfarm/network. For instance, after the creation of the KdTree, the structure can be sent across a network to compute forces on a range of particles. By taking advantages of the latest hardware and library such as OpenCL and CUDA, it is possible to de-multiply the efficiency at a relative low cost.
- Maya Dynamics - emitters and fields : The current system doesn’t offer a dynamic way to add particles. This limitation reduces the possible effects. In the future, a default Maya emitter could pour dynamically soft substances. This implementation can offer a larger range of effects. In this study, external forces such as air, wind, turbulences have not been considered but the integration of Maya fields into the system would help to reach new possibilities.

To conclude, this project demonstrates a framework for particle/particle interaction, investigations on new functions, controllability, speed and full integration on Maya Dynamics (fields, emitters) can offer an unique tool for artists to develop fast and stunning effects.

Appendix

Maya PDC format

The Particle Disk Cache(PDC) file is used by Maya for particle caching and startup cache. The description below is from Autodesk Documentation and a simplified version of PDC can be found at this address. *http://www.sdsc.edu//us//visservices//software//maya//pdc_table.html*

PDC file header

Description	Byte Count	Default
Four Characters indicating that this is a PDC file.	4	'P', 'D', 'C', ' '
One Integer indicating the file format version number.	4	1
One Integer holding bit information if the file are big endian or little endian.	4	1
One Integer holding extra information 1	4	0
One Integer holding extra information 2	4	0
One Integer indicating the number of particles	4	1
One Integer indicating the number of attributes (e.g. position)	4	1

PDC file single record

Description	Byte Count	Default
One Integer indicating the length of the attribute's name.	4	8
One String indicating the name of the attribute	L	position

PDC file : Swapping to support the byte order

Maya only supports big endian which means the biggest value in the byte sequence is stored at the lowest storage address. In other words, numerical values (except string) written into the PCD file need to swap their bytes. The following C++ function converts a little endian value into a big endian value.

```
int integerValue;
double doubleValue;

swap_value((char*) &integer, sizeof(integerValue));
swap_value((char*) &doubleValue, sizeof(doubleValue));

void swap_value (char* value_array, int size)
(
 char swap;
 for(int i=0;i < (size/2);i++)
 (
  swap = value_array[(size-1)-i];
  value_array[(size-1)-i]=value_array[i];
  value_array[i]=swap;
 )
)
```

Appendix

Scene format

Two personal formats to store the scene and the simulation. The scene file contains the entire scene, it is created by the translator in the plugin and read in the external application.

File structure			
Object tag	Parameters	Type	Description
Camera	Eye	vector	Position of the camera
	VerticalAngle	float	Vertical angle of the camera
	HorizontalAngle	float	Horizontal angle of the camera
Environment	Width	float	Width of the container
	Height	float	Height of the container
	Depth	float	Depth of the container
	Reflective	boolean	Reflectivity of the floor
	NormalAttenuation	float	Normal attenuation / coefficient of restitution
	TangentAttenuation	float	Tangent attenuation / friction
Object	UniformScale	float	Uniform scale
	InitialPosition	vector	Initial position of the object
	InitialRotation	vector	Initial rotation of the object
	InitialLinearVelocity	vector	Initial linear velocity of the object
	InitialAngularVelocity	vector	Initial angular velocity of the object
	NormalAttenuation	float	Normal attenuation / coefficient of restitution
	TangentAttenuation	float	Tangent attenuation / friction
	Mass	float	Mass of the object
	IsDynamic	boolean	Static or dynamic objects
Fluid	Name	string	Name of the particle system
	InteractionDistance	float	Maximum distance of interaction
	Amplitude	float	Amplitude of the function
	FiniteDistance	float	Distance splitting repulsive/attractive force
	MinimumForce	float	Minimum force of interaction
	MaximumForce	float	Maximum force of interaction
	Viscosity	float	Damping force to limit motion in the fluid
	InitialSpeed	vector	Initial speed of the fluid
	ParticleSize	float	Size of the particle
	Colour	vector	Colour of the particle
	NumberOfParticles	integer	Number of particles
	Particles	array of vectors	Positions for every particles

Appendix

Simulation format

The simulation file describes the object transformations and particle positions at a specific frame. It is created in the application and read by the plugin.

File structure			
Object tag	Parameters	Type	Description
Object	UniformScale	float	Uniform scale
	Position	vector	Position of the object at frame t
	Rotation	quaternion	Rotation of the object at frame t
	Filename	string	Filename of the object's geometry
Fluid	Name	string	Name of the particle system
	ParticleNumber	integer	Number of particle
	Particles	array of vectors	Particles's position

Appendix

Inertia tensors

This list of moment of inertia tensors given for different primitives.

Primitive	Inertia tensors
Solid sphere with radius r and mass m	$I = \begin{bmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{bmatrix}$
Solid cone with radius r, height h and mass m	$I = \begin{bmatrix} \frac{3}{5}mh^2 + \frac{3}{20}mr^2 & 0 & 0 \\ 0 & \frac{3}{5}mh^2 + \frac{3}{20}mr^2 & 0 \\ 0 & 0 & \frac{3}{10}mr^2 \end{bmatrix}$
Solid cuboid with width w, height h, depth d and mass m	$I = \begin{bmatrix} \frac{1}{12}m(h^2 + d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2 + d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2 + h^2) \end{bmatrix}$
Solid cylinder with radius r, height h and mass m	$I = \begin{bmatrix} \frac{1}{12}m(3r^2 + h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2 + h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{bmatrix}$

References

[1] WT Reeves. Particle systemsa technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics (TOG)*, 2(2):108, 1983.

[2] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The visual computer*, 2(4):227–234, 1986.

[3] G. Miller and A. Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers & Graphics*, 13(3):305–309, 1989.

[4] M. Desbrun and M.P. Gascuel. Animating soft substances with implicit surfaces. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 287–290. ACM, 1995.

[5] M. Desbrun. Animation of deformable models using implicit surfaces. 1997.

[6] M. Desbrun and M.P. Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation*, volume 96, pages 61–76. Citeseer, 1996.

[7] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, page 159. Eurographics Association, 2003.

[8] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151. Eurographics Association, 2004.

[9] K. Steele, D. Cline, P.K. Egbert, and J. Dinerstein. Modeling and rendering viscous liquids. *Computer Animation and Virtual Worlds*, 15(34):183–192, 2004.

[10] S. Clavet, P. Beaudoin, and P. Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, page 228. ACM, 2005.

[11] Yizhou Yu Bell, Nathan and Peter J. Mucha. Particle-based simulation of granular materials. In Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation, 2005.

[12] Takahiro Harada. Real-time rigid body simulation on gpus. GPU Gems 3, Chapter 29, 2008. `http://http.developer.nvidia.com/GPUGems3/gpugems3_ch29.html`.