Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Αρχιτεκτονική, σχεδιασμός και υλοποίηση ενός επεκτάσιμου, ανεξάρτητου υποσυστήματος φυσικής πραγματικού χρόνου για εικονικά περιβάλλοντα, παιχνίδια και εξομοιώσεις**
Ονοματεπώνυμο Φοιτητή	**Γεράσιμος Ράπτης**
Πατρώνυμο	**Άλκης**
Αριθμός Μητρώου	**ΜΠΠΛ/09006**
Επιβλέπων	**Θεμιστοκλής Παναγιωτόπουλος, Καθηγητής**

Τριμελής Εξεταστική Επιτροπή

Θεμιστοκλής Παναγιωτόπουλος Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Χαράλαμπος Κωνσταντόπουλος Λέκτορας

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ....4
1. Περίληψη – Abstract....7
1.1. Περίληψη....7
1.2. Abstract....7
2. Εισαγωγή....8
2.1. Εικονικά περιβάλλοντα και προσομοιώσεις....8
2.1.1. Ακρίβεια στην προσομοίωση....9
2.1.2. Πιστευτότητα και Ρεαλισμός....10
2.2. Φυσική στα εικονικά περιβάλλοντα και τις προσομοιώσεις....11
2.2.1. Τα παιχνίδια ως εικονικά περιβάλλοντα – Φυσική σε παιχνίδια....12
2.3. Μοντελοποίηση φυσικών νόμων....15
2.3.1. Δυνατότητες μοντελοποίησης φυσικών νόμων....15
2.3.2. Είδη μοντελοποίησης φυσικών νόμων κατά την ακρίβεια και το χρόνο 16
2.3.3. Μοντελοποίηση σωμάτων....18
2.4. Τρόποι χρήσης σε λογισμικό – Μηχανές φυσικής....19
2.4.1. Απουσία φυσικής, ή προ-υπολογισμένα σενάρια φυσικών νόμων....19
2.4.2. Ενσωματωμένη μηχανή φυσικής....20
2.4.3. Ανεξάρτητες Μηχανές / υποσυστήματα φυσικής....23
2.5. Φυσικοί νόμοι σε μηχανές φυσικής πραγματικού χρόνου....24
2.5.1. Ανίχνευση και αντιμετώπιση συγκρούσεων....24
2.5.2. Κινηματική και Μηχανική στερεού σώματος (rigid-body mechanics)....26
2.5.3. Μηχανική παραμορφώσιμου σώματος (deformable-body mechanics)....26
2.5.4. Επαφές, σύνδεσμοι και περιορισμοί....26
2.5.5. Άλλοι νόμοι....27
2.6. Εφαρμογή των φυσικών νόμων σε μηχανή φυσικής πραγματικού χρόνου27
2.6.1. Εμπρόσθιο Δυναμικό Πρόβλημα....28
2.6.2. Πιθανά προβλήματα....29
2.6.3. Μέθοδοι ολοκλήρωσης....30
2.6.4. Ικανοποίηση περιορισμών – Αντίστροφο δυναμικό πρόβλημα....33
3. Σύλληψη / Ανάλυση στόχων....35
3.1. Στόχοι του σχεδιασμού....35
3.2. Βασικά σημεία στις επιλογές τεχνολογίας....36
3.3. Βασικά σημεία στη δομή των νόμων....37
3.4. Βασικά σημεία στη δομή των φυσικών ιδιοτήτων....37
4. Επιλογές υλοποίησης και γλώσσας προγραμματισμού....41
4.1. Επιλογές ως προς την προσέγγιση....41
4.1.1. Κλασσική αλγοριθμική προσέγγιση ενός νήματος....41
4.1.2. Πολυνηματική αλγοριθμική προσέγγιση....41
4.1.3. Διανυσματοποίηση (Vectorization)....42
4.1.4. Επεξεργασία γενικής χρήσης GPU (Shaders, OpenCL, nVidia Cuda)....43
4.2. Επιλογές ως προς τη γλώσσα....44
4.2.1. Γλώσσες που μεταγλωττίζονται σε κώδικα μηχανής (native)....44
4.2.2. Γλώσσες που διερμηνεύονται (interpreted)....45

4.2.3. Γλώσσες που χρησιμοποιούν περιβάλλον εκτέλεσης (Runtime environment)....45
4.3. Τελική επιλογή....46
5. Σχεδιασμός και υλοποίηση του υποσυστήματος φυσικής Vesper3D....48
5.1. Διάγραμμα τάξεων του Vesper3D....50
5.2. Επεξήγηση του διαγράμματος / Περιγραφή των τάξεων....51
5.2.1. class PhysicsLaw (abstract):....51
5.2.2. struct PhysicalPropertyDescription:....52
5.2.3. class PhysicalPropertyProvider - (abstract):....52
5.2.4. class NewtonsLawOfMotion (implements PhysicsLaw):....53
5.2.5. class PhysicsEngine (implements PhysicalPropertyProvider):....53
5.2.6. class BasePhysicalPropertyProvider – (implements PhysicalPropertyProvider)....55
5.3. Βοηθητικές τάξεις και αρχεία....56
5.3.1. ResizeableArray <T>....56
5.3.2. Vector4<T>....57
6. Χρήση του Vesper3D....59
6.1. Προϋποθέσεις χρήσης....59
6.2. Βασικοί χρήστες και απαιτούμενη ανάπτυξη λογισμικού....59
6.2.1. Εφαρμογή Πελάτης....59
6.2.2. Διεπαφή μεταξύ Πελάτη και Vesper3D....59
6.2.3. Κεντρική μηχανή του Vesper3D....60
6.2.4. Φυσικοί Νόμοι....60
6.3. Πρότυπα φυσικών νόμων....60
6.3.1. Βασικό πρότυπο (float/4Dhomogenous)....61
6.3.2. Πλεονεκτήματα από τη χρήση Προτύπου....62
6.3.3. Παράδειγμα χρήσης προτύπου....62
7. Οδηγίες χρήσης του Vesper 3D (How-to)....64
7.1. Περιεχόμενα της διανομής του Vesper3D....64
7.1.1. Κυρίως μηχανή....64
7.1.2. Φυσικοί νόμοι....64
7.1.3. Πρότυπα....64
7.2. Συγγραφή νόμων φυσικής....64
7.3. Συγγραφή της διεπαφής....65
7.4. Χρήση της μηχανής φυσικής Vesper3D από το πρόγραμμα-πελάτη....66
7.5. Αρχεία κεφαλίδων....67
8. Μελέτη περιπτώσεων (Case Study)....68
8.1. Μελέτες επιπρόσθετου κόστους....68
8.1.1. Εκτέλεση κενού σεναρίου χωρίς τη μηχανή....68
8.1.2. Εκτέλεση σεναρίου με κενή μηχανή φυσικής....68
8.1.3. Εκτέλεση σεναρίου με μηχανή φυσικής με κενές φυσικές ιδιότητες....69
8.1.4. Εκτέλεση πλήρους απλού σεναρίου....69
8.1.5. Συμπεράσματα Μελετών επιπροσθέτου κόστους:....70
8.2. Μελέτες χρηστικότητας – Custom Μηχανή γραφικών OpenGL....70
8.2.1. Σύνδεση φυσικών ιδιοτήτων με απεικονιζόμενα αντικείμενα σε custom μηχανή γραφικών OpenGL....70
8.2.2. Ευθύγραμμη Ομαλή Κίνηση....71

8.2.3. Προσθήκη βαρυτικού πεδίου στην κατακόρυφη βολή....72
8.2.4. Πλάγια βολή υπό την επίδραση βαρυτικού πεδίου....73
8.2.5. Μάζες συνδεδεμένες με ελατήρια....76
8.2.6. Φασματογράφος μάζας – Χρήση ορισμών πεδίων....77
8.3. Μελέτη αλληλεπίδρασης με εικονικό περιβάλλον (Επικοινωνία με jReve)85
8.3.1. Βασικά για την διαγλωσσική επικοινωνία....85
8.3.2. Βασικό παράδειγμα....86
8.3.3. Ίσες μάζες χωρίς αρχική ταχύτητα υπό την επίδραση ελατηρίου με απόσβεση 86
8.3.4. Ίσες μάζες με αρχική συμμετρική ταχύτητα υπό την επίδραση ελατηρίου με απόσβεση 86
8.3.5. Ίσες μάζες χωρίς αρχική ταχύτητα υπό την επίδραση ελατηρίου με απόσβεση 87
8.3.6. Άνισες μάζες με αρχική ασύμμετρη ταχύτητα υπό την επίδραση ελατηρίου με απόσβεση87
8.3.7. Μάζες εκτελούν βαλλιστική κίνηση υπό την επίδραση βαρυτικού πεδίου87
8.3.8. Ίσες μάζες με αρχική ταχύτητα προς τα άνω υπό την επίδραση ελατηρίου με απόσβεση και βαρύτητας....88
8.3.9. Ίσες μάζες με αρχική ταχύτητα προς τα άνω υπό την επίδραση ελατηρίου με απόσβεση και βαρύτητας. Μείωση σκληρότητας ελατηρίου....88
8.4. Μηχανική παραμορφώσιμου σώματος με δομή – Ανίχνευση συγκρούσεων88
8.4.1. Οριζόντιο επίπεδο – σκληρό δάπεδο – σκληρό σώμα....90
8.4.2. Οριζόντιο επίπεδο – μέτρια σκληρό δάπεδο – μαλακό σώμα....91
8.4.3. Οριζόντιο επίπεδο – μαλακό δάπεδο – μέτρια σκληρό σώμα....92
8.4.4. Κεκλιμένο επίπεδο – μέτρια σκληρό δάπεδο – σκληρό σώμα – υψηλή τριβή 93
8.1.1. Κεκλιμένο επίπεδο – μέτρια σκληρό δάπεδο – μέτρια σκληρό σώμα – χαμηλή τριβή....94
8.4.5. Κεκλιμένο επίπεδο – σκληρό δάπεδο – πολύ μαλακό σώμα – υψηλή τριβή 95
8.5. Μελέτη απόδοσης - αλληλεπίδρασης με ειδικές τεχνολογίες (OpenCL). 96
8.5.1. Γενικά για το OpenCL....97
8.5.2. Εκτέλεση με διανυσματικές βιβλιοθήκες....100
8.5.3. Εκτέλεση με OpenCL....101
8.5.4. Εκτέλεση με συνεργασία τεχνολογιών....102
8.5.5. Εκτέλεση με OpenCL με βελτιστοποίηση μεταφοράς δεδομένων....103
9. Συμπεράσματα και αξιολόγηση σχεδιασμού και υλοποίησης του Vesper3D.....104
9.1. Αξιολόγηση.....104
9.2. Συμπεράσματα - Περίληψη.....106
10. Βιβλιογραφία.....108

1. Περίληψη – Abstract

1.1. Περίληψη

Διαπραγματευόμαστε την ανάλυση απαιτήσεων, τον σχεδιασμό και την υλοποίηση ενός υποσυστήματος επεξεργασίας νόμων φυσικής πραγματικού χρόνου, το οποίο θα χρησιμοποιεί ένα υψηλό επίπεδο αφαίρεσης. Θέλουμε να επιτύχουμε τη δυνατότητα χρήσης από διαφορετικά συστήματα όπως εικονικά περιβάλλοντα, παιχνίδια και προσομοιώσεις, και ταυτόχρονα να διατηρήσουμε επεκτασιμότητα. Η αφαίρεση συνίσταται στην δόμηση των φυσικών μεγεθών σε δομές προγραμματιστικά ανεξάρτητες από το αντικείμενο όπου ανήκουν, τη δόμηση των φυσικών νόμων σε αυτοπεριεχόμενα τμήματα και στην παρουσία ενός κεντρικού επίπεδου που ενορχηστρώνει τις οντότητες αυτές. Ο χρήστης/προγραμματιστής του συστήματος φυσικής θα έχει θα έχει τη δυνατότητα να σχεδιάζει, προσθέτει και αφαιρεί φυσικούς νόμους στο υποσύστημά του κατά βούληση, με τη μορφή ανεξαρτήτων τμημάτων, χωρίς να απαιτείται αλλαγή στο κυρίως τμήμα. Ταυτόχρονα σε όλο το σχεδιασμό διατηρείται η ικανότητα βελτιστοποίησης κώδικα σε υψηλό επίπεδο(αρχιτεκτονικής) ή σε χαμηλό επίπεδο (κώδικα), και η δυνατότητα χρήσης διαφορετικών τεχνολογιών προγραμματισμού. Το όλο υποσύστημα προσθέτει ελάχιστη επιβάρυνση στην απόδοση του συστήματος.
Το παραπάνω σύστημα θα υλοποιηθεί στο σύνολό του και θα υλοποιηθούν ενδεικτικά φυσικοί νόμοι για τον έλεγχο της λειτουργίας του.

1.2. Abstract

We are negotiating the requirements analysis, design and implementation of a real time physics law processing subsystem which will be using a high level of abstraction. We wish to make it possible to be used by different systems such as virtual environments, games and simulations, and at the same time keep it extensible. The abstraction mainly consists of the structuring of physical properties in entities independent to the objects to which they refer, the design of physics laws as independent modules, and the presence of a central layer that orchestrates the interaction of the above entities. The programmer/ user will be able to design, add and remove physics laws to his simulation as independent modules without need to alter the core engine. At the same time throughout the entire design close attention will be paid not to hinder optimization opportunities that would be otherwise possible, either at a high level (architecture) or at low level(code), and that the physics laws will be able to use different technologies. The whole subsystem will be adding minimum overhead to the performance of the client system.
The above system will be completely implemented, together with a few physics laws that will serve to demonstrate its functionality.

Keywords: Software engineering, Physics, Physics Engine, Physics Layer, Virtual Environments, Physics Simulation

2. Εισαγωγή

2.1. Εικονικά περιβάλλοντα και προσομοιώσεις

Τα εικονικά περιβάλλοντα αποτελούν σήμερα ένα τομέα υψηλού ενδιαφέροντος σε ακαδημαϊκούς και εμπορικούς τομείς. Στον ακαδημαϊκό και ερευνητικό χώρο χρησιμοποιούνται σε προσομοιώσεις ως πλατφόρμες έρευνας, ως αναπαραστάσεις φυσικών περιβαλλόντων, στην τεχνητή νοημοσύνη ως περιβάλλοντα αλληλεπίδρασης αναπαραστάσεων φυσικών προσώπων (avatars) και ευφυών πρακτόρων και παρουσιάζουν στη σύγχρονη επιστήμη ιδιαίτερο ενδιαφέρον.
Γενικά καθίσταται δύσκολο να βρεθεί ένας κοινά αποδεκτός ορισμός για το **εικονικό περιβάλλον**. Ένας κοινός ορισμός που μπορεί να βρεθεί από τον τυπικό ενδιαφερόμενο μετά από σύντομη διαδικτυακή αναζήτηση (*dictionary.com*) είναι ***μια δημιουργημένη από υπολογιστή τρισδιάστατη απεικόνιση κάποιου χώρου, μέσα στην οποία (απεικόνιση) ο χρήστης αντιλαμβάνεται ότι βρίσκεται και στην οποία λαμβάνουν χώρα αλληλεπίδρασεις.***
Θεωρούμε τον παραπάνω ορισμό περιοριστικό: εύκολα μπορεί κάποιος να φανταστεί ότι, επί παραδείγματι, μια απεικόνιση θα μπορούσε να είναι όχι τρισδιάστατη, αλλά δισδιάστατη, και να αποτελεί εικονικό περιβάλλον. Όμως, αποδεχόμενοί τον ως επαρκή, καθίσταται προφανές ότι σε αυτόν εμπίπτουν πολυάριθμες εφαρμογές, και ιδιαίτερα πολλές κατηγορίες παιχνιδιών για υπολογιστή.
Στον εμπορικό χώρο ο τομέας των παιχνιδιών σε ηλεκτρονικούς υπολογιστές γνωρίζει τεράστια άνθηση, η οποία σε πολλές περιπτώσεις αποτελεί κινητήρια δύναμη για την βιομηχανία υπολογιστών – η εμπορικότητα δημιουργεί την ανάγκη να αναπαρασταθούν με ολοένα και μεγαλύτερο επίπεδο ρεαλισμού και πιστότητας φυσικοί ή φανταστικοί κόσμοι με σκοπό την βελτίωση της εμπειρίας του χρήστη, ώστε πιέζει την αιχμή της τεχνολογίας.
Αν το παραπάνω φαίνεται παράδοξο, ας παρατηρηθεί άνευ αποδείξεως ότι η αναπαράσταση ενός συστήματος με μεγάλο βαθμό ρεαλισμού μπορεί να χρησιμοποιήσει τεράστιους υπολογιστικούς πόρους, με αποτέλεσμα – τουλάχιστον μέχρι στιγμής – η τεχνολογία των παιχνιδιών να είναι πάντοτε και ανά πάσα στιγμή σε θέση να απορροφήσει οσουσδήποτε υπολογιστικούς πόρους θέτει στην διάθεσή της η βιομηχανία υλικού (hardware) υπολογιστών.
Προσομοίωση εδώ είναι η ***αναπαράσταση της συμπεριφοράς ενός συστήματος μέσω της χρήσης ενός άλλου συστήματος, ιδιαίτερα της περίπτωσης ενός προγράμματος ηλεκτρονικού υπολογιστή.***
Σε ακαδημαϊκούς, ερευνητικούς και γενικά επιστημονικούς χώρους, υλοποιούνται προσομοιώσεις διαφόρων φυσικών συστημάτων για ερευνητικούς ή εμπορικούς σκοπούς, όπως η προσομοίωση του γαλαξία, του ηλιακού συστήματος ή άλλων περιοχών του σύμπαντος για αστροφυσική έρευνα, συστημάτων σωματιδίων για την προσομοίωση υποατομικών αλληλεπιδράσεων, προσομοίωση στατική και δυναμικής συμπεριφοράς φυσικών κατασκευών στην Δομοστατική. Γενικώς θεωρείται αυταπόδεικτο ότι υπάρχουν πολυάριθμες χρήσεις σε πολλούς τομείς για την προσομοίωση φυσικών συστημάτων.
Ειδικεύοντας τον παραπάνω ορισμό στο τμήμα που αφορά την επιστήμη της Πληροφορικής μπορούμε να πούμε: ***Προσομοίωση είναι η αναπαράσταση της***

συμπεριφοράς ενός συστήματος μέσω της χρήσης ενός προγράμματος ηλεκτρονικού υπολογιστή.

Ο τομέας της προσομοίωσης μέσω υπολογιστή είναι πολύ μεγάλης σημασίας στο σύγχρονο πολιτισμό διότι *επιτρέπει την μελέτη συστημάτων έξω από τα συστήματα αυτά.* Με τον τρόπο αυτό, δίνονται πολυάριθμες δυνατότητες που διαφορετικά θα ήταν ιδιαίτερα δύσκολες ή αδύνατες: συστήματα των οποίων η απευθείας μελέτη μπορεί να ήταν χρονοβόρα (συστήματα με πολύ μεγάλο χρόνο ζωής, όπως το σύμπαν), επικίνδυνη ή απρόσιτη(επιφάνεια του ηλίου, εσωτερικό της γης), είτε μη πρακτική διότι απαιτείται εκ των προτέρων γνώση της συμπεριφοράς του συστήματος (μηχανική των κατασκευών – δομοστατική).

Μερικοί από τους τομείς που κατά κόρον χρησιμοποιούν προσομοιώσεις για τη μελέτη συστημάτων είναι η κοσμολογία, η αστροφυσική, η μοριακή βιολογία, η φαρμακοτεχνική, η υποατομική φυσική, η μηχανική (δομοστατική), η σεισμολογία και πολλές άλλες.

Στην περίπτωση της παρούσας εργασίας, ασχολούμαστε με την προσομοίωση φυσικών συστημάτων, και συγκεκριμένα μας ενδιαφέρουν προσομοιώσεις οι οποίες έχουν σκοπό την προσομοίωση φυσικής συμπεριφοράς ή φυσικών νόμων, ή που χρησιμοποιούν την προσομοίωση φυσικών νόμων για την επιτυχία προσομοίωσης κάποιου άλλου συστήματος.

Από τις παραπάνω περιγραφές των εικονικών περιβαλλόντων και των προσομοιώσεων μπορούμε άμεσα να συμπεράνουμε ότι σε πολλές περιπτώσεις θα απαιτείται η προσομοίωση νόμων φυσικής ώστε να προσεγγισθούν με κάποιο ρεαλισμό τα συστήματα αυτά. Συμπεραίνουμε επίσης ότι η προσομοίωση φυσικών νόμων μπορεί να αποτελεί αυτοσκοπό (όπως στην περίπτωση εξομοίωσης ενός φυσικού συστήματος), ή να αποτελεί ένα μόνο επίπεδο στην επιτυχία του ρεαλισμού (όπως στην περίπτωση ενός παιχνιδιού).

2.1.1. Ακρίβεια στην προσομοίωση

Για να μετρηθεί, ή τουλάχιστον για να εκτιμηθεί, η επιτυχία μιας προσομοίωσης, τμήμα ή όχι ενός εικονικού περιβάλλοντος, μπορούμε να απομονώσουμε κάποιες σημαντικές παραμέτρους.

Ένα σημαντικό κοινό σημείο στα εικονικά περιβάλλοντα και στις προσομοιώσεις είναι η ακρίβεια.

Έστω μια καθεαυτώ προσομοίωσης ενός φυσικού συστήματος (θα αναφέρεται εφεξής ως *προσομοίωση*), με πιθανό σκοπό την προσέγγιση ή πρόβλεψη της συμπεριφοράς του φυσικού συστήματος αυτού (θα το αναφέρουμε εφεξής ως *αντικείμενο* σύστημα).

Δεχόμαστε ως εξ ορισμού δεδομένο ότι **η προσομοίωση επιθυμούμε να έχει ακριβώς, ή να προσεγγίσει σε κάποιο σχετικά μεγάλο βαθμό, τη συμπεριφορά του αντικείμενου συστήματος.** Ακρίβεια (*accuracy*) εδώ θα θεωρήσουμε την ομοιότητα του αντικείμενου συστήματος με την προσομοίωση. Θεωρούμε λοιπόν ότι μια προσομοίωση η οποία εμφανίζει «πανομοιότυπη» συμπεριφορά με το αντικείμενο σύστημα είναι απόλυτα ακριβής. Κατά τα άλλα, στην περίπτωση που η προσομοίωση προσεγγίζει αλλά δεν έχει απόλυτα τη συμπεριφορά του αντικείμενου συστήματος, τη διαφορά ανάμεσα στη συμπεριφορά θα ονομάσουμε *ανακρίβεια(inaccuracy)*.

Σε μια προσομοίωση, η ακρίβεια βρίσκεται σε άμεση σύνδεση με τη καθεαυτώ χρησιμότητα της προσομοίωσης, οπότε αποτελεί αυτοσκοπό : είναι προφανές ότι η προσομοίωση είναι τόσο χρησιμότερη όσο πιο κοντά είναι αν τα αποτελέσματά της στο αντικείμενο σύστημα, τουλάχιστον στο βαθμό που ο βαθμός της ανακρίβειας είναι

μετρήσιμος. Αντίστροφα, εάν η προσομοίωση έχει μεγάλη διαφορά από το αντικείμενο σύστημα, χάνεται το νόημά της διότι τελικά η προσομοίωση «δεν προσομοιάζει» στο αντικείμενο σύστημα.
Αυτό ισχύει είτε σε περίπτωση που μας ενδιαφέρει η ποσοτική εκτίμηση κάποιων μεγεθών (όπου ανακρίβεια αυξάνει την διασπορά των αποτελεσμάτων από την πραγματικότητα) είτε η ποιοτική συμπεριφορά (όπου μεγάλες αποκλίσεις μπορεί να αλλάξουν και τη φύση των αποτελεσμάτων).

2.1.2. Πιστευτότητα και Ρεαλισμός

Αντίστοιχα και αντίστροφα, στα εικονικά περιβάλλοντα η ακρίβεια επηρεάζει (αλλά δεν ταυτίζεται με) την προβλεψιμότητα (predictability) και το ρεαλισμό (realism) του, με άμεσο αντίκτυπο στην πιστευτότητα (believability) του, η οποία αποτελεί συχνά βασικό στόχο σε ένα εικονικό περιβάλλον.
Η πιστευτότητα επηρεάζει την εμπειρία του χρήστη μέσω του μηχανισμού που ο ποιητής και φιλόσοφος Samuel Taylor Coleridge ονόμασε Διακοπή της Δυσπιστίας(Suspension Of Disbelief): ο χρήστης / θεατής επιτρέπει στον εαυτό του να «ξεχάσει» ότι κοιτάζει, επί παραδείγματι, μια οθόνη που εμφανίζει ένα φανταστικό κόσμο, και επιτρέπει στον εαυτό του προσωρινά να αντιμετωπίσει την αναπαράσταση ως κόσμο πραγματικό και τα γεγονότα που συμβαίνουν σε αυτό σαν πραγματικά παρά το (πιθανώς) «απίστευτο» πλαίσιό τους.
Παρατηρούμε λοιπόν πως στην περίπτωση αυτή της αναπαράστασης **δεν αποτελεί πλέον αυτοσκοπό η ακρίβεια**, παρά αποτελεί σκοπό κυρίως στο βαθμό που επηρεάζει την πιστευτότητα του εικονικού κόσμου, ακριβώς διότι σε πολλές περιπτώσεις **απαιτείται για την επιτυχία μόνο ένα *πιστευτό* και όχι ένα *ακριβές* αποτέλεσμα.**
Αντίστοιχα, έστω ένα σύστημα το οποίο φέρεται να προσομοιάζει ένα πραγματικό, και υπάρχει πολύ μεγάλη ανακρίβεια (inaccuracy) στην προσομοίωση – δηλαδή η συμπεριφορά δεν μοιάζει στο αντικείμενο σύστημα ούτε καν ποιοτικά, ή υπάρχει ασυνέπεια (inconsistency) στη συμπεριφορά του. Η συμπεριφορά του δεν θα είναι προβλέψιμη, και θα φαίνεται ως απουσία νομοτέλειας, με αποτέλεσμα ο χρήστης πολύ δύσκολα να μπορεί να το πιστέψει. Ως παράδειγμα, ας υποθέσουμε το παρακάτω σενάριο.

- Μια ελαστική σφαίρα (μπαλάκι) βάλλεται πάνω σε ένα κατακόρυφο τοίχο. Ο χρήστης αναμένει, ποιοτικά, το μπαλάκι να αναπηδήσει στον τοίχο και να αντιστραφεί η οριζόντια ταχύτητά του με κάποια απώλεια. Μια μηχανή φυσικής υπολογίζει το αποτέλεσμα της κρούσης του εφαρμόζοντας κάποια δύναμη στο μπαλάκι.
    - Αν η μηχανή φυσικής υπολογίσει ακριβώς τη δύναμη που θα πρέπει να ασκηθεί στο μπαλάκι, αυτό θα εμφανιστεί να αναπηδά ακριβώς με τον τρόπο που θα αναπηδούσε στην πραγματικότητα για ένα «ακριβές» αποτέλεσμα.
    - Αν η μηχανή φυσικής έχει κάποια προσέγγιση που έχει ως αποτέλεσμα την εφαρμογή δύναμης διαφορετικής από αυτήν που θα έπρεπε για την τέλεια προσομοίωση, θα υπάρχει κάποια διαφορά με την συμπεριφορά που θα αναμενόταν να έχει το μπαλάκι, όπως αυτό μετά την αναπήδηση να έχει μικρότερη ή μεγαλύτερη ταχύτητα από αυτή που θα «έπρεπε». Το αποτέλεσμα όμως είναι ποιοτικά σωστό (το μπαλάκι αναπηδά) και ο

χρήστης δεν αντιλαμβάνεται καμία διαφορά διότι δεν έχει στην πραγματικότητα τη δυνατότητα να μετρήσει τη διαφορά.

- Αν η μηχανή φυσικής δεν εφαρμόσει αρκετή δύναμη, ή την εφαρμόσει με λανθασμένη φορά, το μπαλάκι μπορεί να περάσει μέσα από τον τοίχο. Ο χρήστης ξαφνιάζεται και αντιλαμβάνεται αμέσως την ασυνέπεια λόγω της ποιοτικής διαφοράς με την αναμενόμενη συμπεριφορά: το μπαλάκι περνά μέσα από τον τοίχο ενώ έπρεπε να αναπηδήσει, και η «ποιοτική» αυτή ασυνέπεια είναι άμεσα ορατή.

Έτσι, ένας από τους παράγοντες που μπορεί να επηρεάσουν άμεσα την *πιστευτότητα, τη συνέπεια* και το *ρεαλισμό* ενός εικονικού περιβάλλοντος με φυσική είναι η προσομοίωση της φυσικής του κόσμου αυτού.
Εδώ, **φυσική του εικονικού κόσμου** εννοούμε την **προσέγγιση και προσομοίωση ενός υποσυνόλου των πραγματικών φυσικών νόμων** ώστε αυτοί είτε να προσδίδουν **πιστευτότητα** στο σύστημά ή/και να **συμμετέχουν στη λογική συμπεριφορά** του.
Η διαφορά των περιπτώσεων αυτών είναι ότι στην πρώτη περίπτωση μας ενδιαφέρει περισσότερο η πιστευτότητα και ενδεχομένως ο ρεαλισμός, ενώ συνήθως στη δεύτερη περίπτωση ενδεχομένως μας ενδιαφέρει και η ακρίβεια.
Αντίστοιχα, φυσική της προσομοίωσης είναι ένα υποσύνολο των φυσικών νόμων το οποίο προσπαθούμε να προσομοιώσουμε για την αναπαράσταση άλλου συστήματος. Εδώ συνήθως μας αφορά μόνο η ακρίβεια του συστήματος και αποτελεί αυτοσκοπό, χωρίς να λαμβάνουμε υπ' όψη την πιστευτότητα του συστήματος.

2.2. Φυσική στα εικονικά περιβάλλοντα και τις προσομοιώσεις

Σε ένα εικονικό κόσμο, η φυσική συχνά αποτελεί τμήμα της ***λογικής*** του: Τμήματα που αποτελούν το φυσικό κόσμο εμφανίζουν συμπεριφορά που φαίνεται συμβατή με τη φυσική του πραγματικού κόσμου (πέφτουν όταν αφήνονται, λυγίζουν ή σπάνε όταν τους εφαρμόζεται δύναμη, συγκρούονται και γενικά εμφανίζονται να ακολουθούν φυσικούς νόμους), αλλά και επηρεάζουν την εμπειρία του χρήστη διότι π.χ. του επιτρέπουν να περπατά ή πετά, εμποδίζουν την κίνησή του μέσα από φαινομενικώς στέρεα αντικείμενα και γενικώς επιτρέπουν ή απαγορεύουν διάφορες δυνατότητες και κινήσεις του.
Σε ένα εικονικό περιβάλλον μπορεί να προσομοιώνονται φυσικοί νόμοι οι οποίοι ενώ δεν έχουν αναγκαστικά σύνδεση με τη φυσική του πραγματικού κόσμου, και στο κατάλληλο πλαίσιο φαίνονται πιστευτοί (τηλεμεταφορά, όπλα σωματιδίων και λέιζερ υψηλής ισχύος που καταστρέφουν αντικείμενα, διαστημική πτήση που δεν προκύπτει από την φυσική του πραγματικού κόσμου). Στην περίπτωση αυτή γίνονται κάποιες παραδοχές, ρητές η όχι, οι οποίες επιτρέπουν στο σύστημα να παραμένει πιστευτό ασχέτως εάν η χρησιμοποιούμενη φυσική απαντά στον πραγματικό κόσμο.
Σε κάθε περίπτωση, οι φυσικοί νόμοι που μοντελοποιούνται και εφαρμόζονται σε ένα εικονικό κόσμο είναι πάντοτε ένα μικρό υποσύνολο των πραγματικών – όταν βέβαια προκύπτουν από αυτούς – το οποίο επιλέγεται με κάποια κριτήρια. Για να εφαρμοσθούν οι νόμοι αυτοί, σχεδιάζεται ένας τρόπος μοντελοποίησης και ενσωμάτωσής τους στο περιβάλλον αυτό. Το λογισμικό που προκύπτει από τη μοντελοποίηση αυτή ονομάζεται συνήθως μηχανή φυσικής.

Όπως συχνά συμβαίνει στην επιστήμη των υπολογιστών, ο τρόπος με τον οποίον θα μοντελοποιηθούν οι φυσικοί νόμοι δεν είναι μοναδικός, αντίθετα μπορεί να εφαρμοστούν οσαδήποτε μοντέλα για την προσέγγιση τους.
Ως γνωστόν, υπάρχουν περισσότεροι φυσικοί νόμοι από αυτούς που θα μπορούσαν να απαριθμηθούν εδώ, πολύ περισσότεροι μάλιστα αν συμπεριλάβουμε αυτούς που δεν έχουν ακόμα ανακαλυφθεί ή διατυπωθεί.
Γενικά όμως, μέσα σε ένα εικονικό περιβάλλον και κατά το χρόνο εκτέλεσής του, ενδιαφέρον έχουν φυσικοί νόμοι που αφορούν συστήματα που μεταβάλλονται συν τω χρόνω, και που έχουν μακροσκοπικά ορατό αποτέλεσμα. Διαφορετικά, σε συστήματα χωρίς χρονική μεταβολή η φυσική επεξεργασία μπορεί να γίνει στατικά, από ένα εξωτερικό σύστημα που δεν έχει τους περιορισμούς πραγματικού χρόνου και εκτέλεσης του περιβάλλοντος, δηλαδή πιθανότατα θα χρησιμοποιηθούν πριν την εκκίνηση του εικονικού περιβάλλοντος ή ανεξάρτητα από αυτό.
Ας πάρουμε το παράδειγμα ενός τοπίου: θεωρούμε ότι δεν μας ενδιαφέρει να ενσωματωθούν φυσικοί νόμοι μέσα σε ένα εικονικό περιβάλλον από τους οποίους θα προκύψει η **δημιουργία** ενός αμετάβλητου τοπίου (το τοπίο αυτό αφού θα είναι αμετάβλητο στη διάρκεια της προσομοίωσης), διότι μετά την εκκίνηση της προσομοίωσης οι νόμοι αυτοί θα είναι ανενεργοί. Πιθανότερο είναι ότι το τοπίο αυτό θα μπορούσε να υπολογισθεί εκ των προτέρων χρησιμοποιώντας μια μηχανή όχι πραγματικού χρόνου και συνεπώς χωρίς τους περιορισμούς που θα έχει μια τέτοια μηχανή, και όχι κατά τη διάρκεια της εκτέλεσης του εικονικού περιβάλλοντος. Ακόμα και στην περίπτωση που ο μηχανισμός αυτός αποτελούσε τμήμα του προγράμματος, πιθανότατα θα ακολουθούσε διαφορετικούς κανόνες από τη φυσική που θα εμφανιζόταν στο εικονικό περιβάλλον κατά τη διάρκεια εκτέλεσής του. Τελικά, η καθεαυτώ προσομοίωση φυσικής του εικονικού περιβάλλοντος θα μπορούσε να ασχοληθεί μόνο με την περεταίρω μεταβολή του εν λόγω τοπίου συν τω χρόνω.
Αντιθέτως, στις προσομοιώσεις πολλές φορές μπορεί να έχει σημασία η μοντελοποίηση συστημάτων χωρίς χρονική μεταβολή, επί παραδείγματι η μοντελοποίηση συγκεκριμένων καταστάσεων ατόμων ή μορίων μέσω της λύσης των εξισώσεων των δυνάμεων που ασκούνται ανάμεσά τους βάσει μοντελοποιημένων φυσικών νόμων.
Σε κάθε περίπτωση, ο Μηχανικός καλείται να επιλέξει τον τρόπο με τον οποίο θα μοντελοποιηθούν οι συγκεκριμένοι φυσικοί νόμοι, και αντίστοιχα με τους νόμους αυτούς θα πρέπει να υπάρχει η αντίστοιχη μοντελοποίηση των Σωμάτων που υπάρχουν στο περιβάλλον της προσομοίωσης.

2.2.1. Τα παιχνίδια ως εικονικά περιβάλλοντα – Φυσική σε παιχνίδια

Όπως αναφέρθηκε, στον χώρο των παιχνιδιών εμφανίζονται πολύ συχνά εικονικά περιβάλλοντα.
Η κατηγορία των παιχνιδιών Πρώτου Προσώπου (First Person, FP) είναι πάντοτε ένα εικονικό περιβάλλον εξ' ορισμού.
Στα παιχνίδια, η έννοια πρώτο πρόσωπο αναφέρεται στην γραφική απεικόνιση - στο βιβλίο Fundamentals Of Game Design (Andrew Rollings, Ernest Adams – 2006) αναφέρεται ότι τα παιχνίδια με προοπτική πρώτου προσώπου είναι γενικώς βασισμένα σε «**avatars**» (ελεύθερα, η οντότητα την οποία ελέγχει ο χρήστης του παιχνιδιού), όπου το παιχνίδι εμφανίζει ό,τι θα έβλεπε ο παίκτης αν βρισκόταν στη θέση και με τις ικανότητες του avatar του. Συνεπώς, συνήθως ο παίκτης δεν μπορεί να δει το σώμα του avatar αλλά πιθανώς να μπορεί να δει τμήματά του, όπως τα χέρια.

Δηλαδή, το «Πρώτο Πρόσωπο» αυτόματα σημαίνει ότι ο χρήστης «υποδύεται» κάποιο ρόλο, κοιτάζοντας μέσα από τα μάτια του avatar.
Εξ' ορισμού καταλαβαίνουμε ότι όταν κοιτάζουμε «από τα μάτια» του ρόλου, αυτόματα παρατηρούμε ένα συγκεκριμένο περιβάλλον – ένα εικονικό περιβάλλον. Για να καταλάβουμε την σημασία αυτής της οπτικής, ας την αντιθέσουμε επί παραδείγματι με ένα παιχνίδι καρτών ή «κουίζ», όπου ο χρήστης δεν παρατηρεί κάποιον συγκεκριμένο κόσμο, παρά παίζει το παιχνίδι με μια εντελώς «αφηρημένη» έννοια, χωρίς ένα συγκεκριμένο περιβάλλον που προσομοιώνει κάποιον κόσμο.
Αντίστοιχα, επίσης στα παιχνίδια Τρίτου Προσώπου αναφερόμαστε εξ' ορισμού σε ένα εικονικό περιβάλλον, το οποίο ο χρήστης παρατηρεί όχι μέσα από τα μάτια του avatar του, αλλά από ένα άλλο σημείο του χώρου, μέσω μιας ή περισσοτέρων εικονικών «καμερών» που βρίσκονται ή κινούνται στον εικονικό χώρο. Στην περίπτωση αυτή, ο χρήστης παρατηρεί το «avatar» – όταν αυτό υπάρχει – και δεν βρίσκεται «μέσα» σε αυτό.
Στα παιχνίδια που λαμβάνουν χώρα ως εικονικά περιβάλλοντα, η φυσική, απλούστερη ή πιο περίπλοκη, παίζει γενικά σημαντικό ρόλο στην ποιότητα της εμπειρίας και την ευχαρίστηση που αντλεί ο χρήστης. Για να έχουμε μια καλύτερη εικόνα της κατάστασης ας δούμε παραδείγματα παιχνιδιών από ένα υποσύνολο των κατηγοριών, και ας μελετήσουμε εξωτερικά την χρήση κανόνων φυσικής σε αυτά.
Ας τηρηθεί εδώ η επιφύλαξη ότι, εφ' όσον πρακτικά όλος ο κόσμος που μας περιβάλλει είναι αποτέλεσμα της φυσικής, αναφερόμαστε σε νόμους που έχουν κάποια απ' ευθείας μοντελοποίηση στο παιχνίδι – ακόμα και το χρώμα του ουρανού είναι αποτέλεσμα φυσικής αλλά δεν θεωρούμε εδώ ότι ο γαλάζιος ουρανός ή το κόκκινο ηλιοβασίλεμα θεωρούνται μοντελοποίηση φυσικής.

- **First Person Shooters** (FPS)(«Σκοποβολή» ή «Πυροβολισμοί» Πρώτου Προσώπου) και γενικώς FP –παιχνίδια. Στα παιχνίδια αυτά ο τρόπος παιχνιδιού περιστρέφεται γύρω από τη μάχη με πυροβόλα όπλα ή γενικώς όπλα με βλήματα (Doom, Wolfenstein 3D, Half Life, Counterstrike, Operation Flashpoint, Call Of Duty, Crysis κ.α.) Αν και δεν είναι απαραίτητα το πρώτο, το παιχνίδι που αποτέλεσε τη βάση για την επιτυχία του είδους θεωρείται το Doom. Η ελάχιστη φυσική που χρησιμοποιείται στην πλειοψηφία τους αναφέρεται τουλάχιστον σε μια υπονοούμενη μάλλον παρά μοντελοποιημένη βαρύτητα που κρατά το χρήστη στο έδαφος και κάποια ανίχνευση συγκρούσεων που προσομοιώνουν το περπάτημα και σχεδόν πάντα τα άλματα και τις πτώσεις (Doom). Σε νεώτερα από αυτά υλοποιούνται νόμοι φυσικής που αναφέρονται στις Βολές υπό βαρύτητα και γενικώς Βαλλιστική για την προσομοίωση ρεαλιστικότερων τροχιών βλημάτων (Operation Flashpoint), ή κίνηση και συμπεριφορά οχημάτων (Halo). Πέραν τούτου άλλοι νόμοι φυσικής μπορεί να υλοποιούνται για την επιτυχία ρεαλισμού της απεικόνισης, όπως άνεμος που κινεί δέντρα, φυσική «πανιού» για σημαίες και άλλα αντικείμενα, και υλοποιήσεις διαφόρων φυσικών νόμων που περιλαμβάνονται στον όρο-ομπρέλα «Φυσική Οχημάτων». Μεγάλης σημασίας και ακανθώδους πολυπλοκότητας για τέτοια παιχνίδια επίσης αποτελεί το καταστρέψιμο περιβάλλον.
- **First Person Role Playing Games** (FPRPG) (Παιχνίδια Ρόλων Πρώτου Προσώπου), και FP-RPG. Τα παιχνίδια αυτά συχνά αλλά όχι αποκλειστικά είναι επέκταση των FPS, όπου όμως δίνεται επιπρόσθετη σημασία στην αλληλεπίδραση του χρήστη με το περιβάλλον και άλλους χρήστες. Η φυσική

τους περιλαμβάνει ότι και στα FPS, αλλά η πολυπλοκότητά τους συχνά συμπαρασύρει και άλλα φαινόμενα που μπορεί να μοντελοποιηθούν μέσω φυσικής, ή προκαλεί τάση για μοντελοποίηση περισσότερων φυσικών νόμων για τη ρεαλιστικότερη απεικόνιση του κόσμου, συχνά με τη μορφή αντικειμένων με ρεαλιστικότερη συμπεριφορά.

- **(First Person) Vehicle Simulation.** Παιχνίδια όπου ο χρήστης υποδύεται τον οδηγό ή πιλότο ενός οχήματος. Το όχημα μπορεί να είναι πρακτικά οτιδήποτε, όπως αυτοκίνητο, μοτοσυκλέτα, αεροπλάνο, ελικόπτερο ή κάποια φανταστική υλοποίηση διαστημοπλοίου. Στην κατηγορία αυτή συμπεριλαμβάνεται και η κατηγορία παιχνιδιών μάχης με αεροπλάνα ή διαστημόπλοια. Η φυσική αυτών των παιχνιδιών, ειδικά της προσομοίωσης αεροπλάνων, μπορεί να γίνει οσοδήποτε λεπτομερής και απαιτητική. Η προσομοίωση της κίνησης ενός οχήματος και η επιτυχία ρεαλισμού είναι μια τέχνη και επιστήμη η οποία συνεχώς εξελίσσεται. Έτσι, στα αεροπλάνα (παιχνίδια όπως η οικογένεια του Falcon, τα USAF και USNF και άλλα) υπάρχει η ανάγκη προσομοίωσης περίπλοκων νόμων βαρύτητας, αεροδυναμικής, κινηματικής και δυναμικής, ορμής και άλλα, ώστε να προσομοιωθεί στο δυνατό βαθμό η ρεαλιστική συμπεριφορά ενός αεροπλάνου. Στα αυτοκίνητα επίσης απαιτείται μοντελοποίηση αντίστοιχων νόμων, με μεγάλη σημασία στην κινηματική/δυναμική, ορμή τριβές/πρόσφυση και ενδεχομένως αεροδυναμική. Αντίθετα, στα διαστημικά παιχνίδια η εικόνα αλλάζει αφού – εφ' όσον δεν υπάρχει ένα φυσικό αντίστοιχο μαχητικού διαστημοπλοίου που πρέπει να προσομοιωθεί – ο σχεδιαστής έχει μεγάλη ελευθερία στην συμπεριφορά που θέλει να επιτύχει. Ας σημειωθεί όμως ότι τα περισσότερα αποτελούν αυστηρό υποσύνολο της φυσικής των αεροπλάνων (X-Wing, Tie Fighter, Wing Commander), ουσιαστικά χρησιμοποιώντας ένα αρκετά απλοποιημένο μοντέλο φυσικής η που προσομοιώνει μια απλούστατη απεικόνιση ατμοσφαιρικής πτήση, κάποια όμως ξεφεύγουν εντελώς χρησιμοποιώντας μακροσκοπική φυσική με βαρύτητα των αστρικών σωμάτων (Elite 2: Frontier) και άλλα.
- **Third Person Shooter/Rpg/Strategy**. Αντίστοιχα, στα παιχνίδια όπου, όταν υπάρχει, το avatar του χρήστη παρατηρείται μαζί με το περιβάλλον, οι απαιτήσεις φυσικής αλλάζουν αρκετά. Σε «**Shooters**» (Resident Evil series) συνήθως η βαλλιστική δίνει τη θέση της σε ευθύγραμμες τροχιές ή αυτόματη στόχευση. Φυσικά υπάρχουν κάποιες περιπτώσεις όπου η φυσική γίνεται σημαντικό τμήμα της εμπειρίας του παίκτη (TP – Vehicle Simulation, και παιχνίδια με λεπτομερές καταστρεφόμενο περιβάλλον: Red Faction Guerrilla). Στις περισσότερες όμως από αυτές τις περιπτώσεις το «Third Person» τμήμα δεν είναι αυστηρό και αποτελεί ουσιαστικά μια συμπεριφορά ημι-πρώτου προσώπου όπου η κάμερα ακολουθεί το avatar και η οπτική γωνία είναι παρόμοια με του πρώτου προσώπου. Στα Strategy η φυσική συνήθως λαμβάνει χώρα ως μέσο επιτυχίας ρεαλισμού και όχι ως τμήμα της «λογικής» του παιχνιδιού, και χρησιμοποιείται με μάλλον απλό τρόπο για την ενίσχυση της συμπεριφοράς του κόσμου.
- **Παιχνίδια φυσικής**. Αποτελεί μια υπό - κατηγορία παιχνιδιών που συνήθως είναι «μάλλον απλά» και ανήκουν στην κατηγορία των λεγόμενων «απλών παιχνιδιών» (Casual Games). Εδώ όμως έχουμε συχνά την αρκετά λεπτομερή μοντελοποίηση κάποιων φυσικών νόμων ως αυστηρά αναπόσπαστο τμήμα του

παιχνιδιού (World Of Goo, angry birds, διάφορα internet games). Δεν μπορεί να δοθεί μια συγκεκριμένη περιγραφή, αλλά γενικά ας παρατηρηθεί το γεγονός ότι επιλέγονται κάποιοι φυσικοί νόμοι (World of Goo: Μηχανική Παραμορφώσιμου Σώματος με ελατήρια και απόσβεση, angry birds: βαλλιστική, ιξώδη ελατήρια και τριβές) και χτίζεται ένα παιχνίδι γύρω από αυτούς.

2.3. Μοντελοποίηση φυσικών νόμων

2.3.1. Δυνατότητες μοντελοποίησης φυσικών νόμων

Μπορούμε εύκολα να καταλάβουμε ότι η μοντελοποίηση ενός νόμου μπορεί να γίνει με πολλούς διαφορετικούς τρόπους και τεχνικές, και η επιλογή ενός συγκεκριμένου τρόπου οφείλει να γίνει με βάση τις συγκεκριμένες απαιτήσεις. Ας σκεφτούμε ένα απλό παράδειγμα μιας πλάγιας βολής από ένα κανόνι. Ανάλογα με το σκοπό της μοντελοποίησης αυτής, θα μπορούσε να συγγραφεί πρόγραμμα που να υπολογίζει τα στοιχεία της βολής αυτής με άπειρους τρόπους. Χάριν επεξήγησης, αναφέρονται οκτώ από αυτούς:

1. Θα μπορούσε να υπολογισθούν εκ των προτέρων με αναλυτικό τρόπο οι εξισώσεις τροχιάς, και να χρησιμοποιηθούν απ' ευθείας ώστε να σχεδιαστεί η τροχιά αυτή στην οθόνη
2. Θα μπορούσαν να χρησιμοποιηθούν εξισώσεις της βολής με βάση το χρόνο που να δείχνουν στην οθόνη ανά τακτά χρονικά διαστήματα τη θέση του σώματος
3. Θα μπορούσε να μοντελοποιηθεί η βαρύτητα ως σταθερό πεδίο επιτάχυνσης με φορά προς τα κάτω και μέγεθος ($9.81 m/sec^2$) και να εφαρμοστεί στο σώμα ως επιτάχυνση, να μοντελοποιηθούν οι νόμοι της κινηματικής του Νεύτωνα, να εφαρμόζονται μέσω ολοκλήρωσης ανά τακτά χρονικά διαστήματα, και μέσω των αποτελεσμάτων τους να δείχνουν τη θέση του σώματος στην οθόνη.
4. Θα μπορούσαν να μοντελοποιηθούν τα παραπάνω, αλλά επιπρόσθετα το σώμα να διαθέτει μάζα, και να μοντελοποιηθεί η αντίσταση του αέρα ως δύναμη που επιδρά πάνω στο σώμα
5. Θα μπορούσε να εφαρμοσθεί το παραπάνω, αλλά να μοντελοποιηθεί η βαρύτητα ως πεδίο δύναμης (9.81N), να εφοδιαστεί το σώμα με μάζα, να εφαρμοσθεί η δύναμη πάνω του.
6. Θα μπορούσε να μοντελοποιηθεί η Γη με τη μάζα και την ακτίνα της, να τοποθετηθεί επάνω της το βλήμα και να υπολογιστεί έτσι η δύναμη που του ασκεί η Γη, με ή χωρίς την παραπάνω αντίσταση του αέρα.
7. Θα μπορούσαν να μοντελοποιηθούν οι εξισώσεις της Γενικής Σχετικότητας του Einstein και να υπολογιστεί η τροχιά του βλήματος με βάση την καμπύλωση του χώρου που προκαλεί η Γη
8. Θα μπορούσε σε κάποια από τις προηγούμενες περιπτώσεις να προστεθούν μεταβλητές όπως ο καιρός, η υγρασία ή οτιδήποτε άλλο μπορούμε να φανταστούμε που θα επηρέαζε τη βολή αυτή.

Αν και κάποια από αυτά φαίνονται ακραία για μια συνήθη περίπτωση (7), ανάλογα με τις απαιτήσεις, οτιδήποτε από αυτά μπορεί να είναι μια δόκιμη αντιμετώπιση: τα (2), (3), (4), (5) θα μπορούσαν να χρησιμοποιηθούν σε κάποιο παιχνίδι, τα (1), (3), (4), (5) θα μπορούσε να είναι κατάλληλα για κάποια στρατιωτική εφαρμογή, ενώ τα (5), (6),

(7), (8) θα μπορούσαν να χρησιμοποιηθούν για την επαλήθευση κάποιας θεωρίας ή των εξισώσεων κάποιου άλλου μοντέλου φυσικής.

2.3.2. Είδη μοντελοποίησης φυσικών νόμων κατά την ακρίβεια και το χρόνο

Συνεπώς, βλέπουμε διάφορα επίπεδα ακρίβειας, πιστότητας και ταχύτητας της προσομοίωσης. Στο παραπάνω παράδειγμα θα μπορούσαμε να μοντελοποιήσουμε όσο το δυνατόν περισσότερους νόμους μπορούμε, με όσο μεγαλύτερη ακρίβεια μπορούμε, να θέσουμε στο πρόγραμμα το ερώτημα της τροχιάς σε ένα τεράστιο επίπεδο πολυπλοκότητας, και να επιτρέψουμε στο πρόγραμμα να τρέξει για λεπτά, ώρες ή και περισσότερο ώστε να πάρουμε μια απάντηση. Ή θα μπορούσαμε να απαιτήσουμε ταχύτατα μια απάντηση σε σχεδόν πραγματικό χρόνο (μερικές δεκάδες χιλιοστά του δευτερολέπτου), αναγκαζόμενοι αντίστοιχα να χρησιμοποιήσουμε λιγότερο περίπλοκη μοντελοποίηση για να έχουμε μια απάντηση το ταχύτερο δυνατόν.
Εδώ μπορούμε έστω και διαισθητικά να καταλάβουμε ότι 1) η *ακρίβεια και η πιστότητα της προσομοίωσης,* 2) η *ποσότητα των φυσικών νόμων* και 3)ο *απαιτούμενος χρόνος επεξεργασίας* θα είναι μεγέθη ανταγωνιστικά μεταξύ τους.
Πραγματικά, η ανταγωνιστικότητα αυτή των μεγεθών μας καταλήγει σε μια πρώτη κατηγοριοποίηση της μοντελοποίησης φυσικών νόμων

- **Μοντέλα υψηλής ακρίβειας, μη πραγματικού χρόνου**

Εδώ επιλέγεται να μοντελοποιηθούν κάποιοι νόμοι με μεθόδους που θα δώσουν αποτελέσματα πολύ μεγάλης ακρίβειας (*precision*). Πιθανώς να γίνουν αναλυτικές επιλύσεις των εξισώσεων ή προσεγγιστικές λύσεις με πολύ μεγάλης ακριβείας προσομοιώσεις, μεθόδους ελέγχου σφαλμάτων και μείωση σφάλματος μέσω επ' άπειρον αυτοεπανάληψης μέχρι να καταλήξουν σε σταθερό συμπέρασμα κ.ο.κ.
Τα μοντέλα αυτά έχουν πολύ μεγάλη αξία σε επιστημονικές έρευνες, στρατιωτικές και διαστημικές εφαρμογές, μηχανική ανάλυση κατασκευών και άλλες χρήσεις που απαιτούν την *_επαρκώς ακριβή_* προσομοίωση ενός φυσικού συστήματος σχεδόν ανεξάρτητα από τον απαιτούμενο υπολογιστικό χρόνο.
Το πλεονέκτημα των μεθόδων αυτών είναι η επιτυχία οσοδήποτε μεγάλου βαθμού ακρίβειας, συνήθως με αντίστοιχη αύξηση του χρόνου υπολογισμού στις προσεγγιστικές λύσεις (κατά το βαθμό πολυπλοκότητας του συστήματος), ή η εύρεση της απολύτως ακριβούς (αναλυτικής) λύσης.
Μειονέκτημα των μοντέλων αυτών είναι προφανώς ο χρόνος επεξεργασίας που απαιτείται, ο οποίος τις κάνει ακατάλληλες για κάθε είδους χρήση που απαιτεί απάντησης σε πρακτικά πραγματικό χρόνο. Στην περίπτωση της αναλυτικής επίλυσης, υπάρχουν πολλά προβλήματα τα οποία δεν διαθέτουν καθόλου αναλυτική λύση.
Στο προηγούμενο παράδειγμα της βολής, τέτοια μοντέλα θα ήταν πιθανότατα τα (1), (7) και (8)

- **Ποιοτικά μοντέλα**

Στα μοντέλα αυτά δεν απαιτούμε την ακριβή απεικόνιση κάποιων φυσικών νόμων αλλά την ποιοτική απεικόνισή τους με μοναδικό σκοπό συνήθως την πιστευτότητα. Χρησιμοποιούνται πολύ συχνά σήμερα για την απεικόνιση πολύπλοκων από πλευράς φυσικής αλληλεπιδράσεων, οι οποίες όμως για το σκοπό της προσομοίωσης δεν προκαλούν ανάγκη ακριβείας, παρά μόνο μια ποιοτική εκτίμηση. Το σώμα που αφήνεται πρέπει να πέσει χωρίς να μας ενδιαφέρει ιδιαίτερα η τιμή της επιτάχυνσης,

στην κρούση πρέπει να αναπηδά κ.ο.κ. Με τον τρόπο αυτό αποφεύγονται πολύπλοκοι υπολογισμοί ενώ διατηρείται κάποιος βαθμός ρεαλισμού (Marc Cavazza, Simon Hartley, Jean-Luc Lugrin, Mikael Le Bras 2004). Φαινόμενα που είναι συχνά κατάλληλα για τέτοια μοντελοποίηση είναι περίπλοκα φαινόμενα, συχνά αποτέλεσμα αλληλεπίδρασης πολλών νόμων, που όμως έχουν απλά, απτά αποτελέσματα: Ρευστομηχανική (γέμισμα ενός ποτηριού), μεταφορά θερμότητας (βρασμός νερού), βιολογία (θάμβωση από το φως), εσωτερική αντοχή (καταστροφή αντικειμένων).
Τα μοντέλα αυτά λειτουργούν συχνά μαζί με ακριβέστερα μοντέλα φυσικής, αλλά τις περισσότερες φορές αποτελούν τμήμα της λογικής μιας προσομοίωσης και όχι τμήμς της μηχανής φυσικής του, ώστε να προσομοιώσουν μεμονωμένα ή ειδικά γεγονότα.

- **Μοντέλα πραγματικού χρόνου, προσεγγιστικά.**

Στις περιπτώσεις αυτές χρησιμοποιούνται μοντέλα φυσικών νόμων που είναι σε θέση να υπολογίζονται σε «σχεδόν πραγματικό χρόνο», δηλαδή ουσιαστικά σε κάποια κλάσματα δευτερολέπτου ανάλογα με τις απαιτήσεις μας.
Ο γενικός κανόνας που μπορεί να μας δώσει τον ορισμό είναι:
Σε ένα σύστημα που εξελίσσεται με το χρόνο, ένα μοντέλο που μπορεί να υπολογίσει την επόμενη κατάσταση κάποιων απαιτούμενων παραμέτρων για την εμφάνιση στο επόμενο καρέ σε χρόνο ίσο ή μικρότερο από το χρονικό διάστημα που μεσολαβεί ανάμεσα στα καρέ αυτά, μπορεί να χρησιμοποιηθεί ως μοντέλο πραγματικού χρόνου.
Απλούστερα στο μοντέλο πραγματικού χρόνου ο επεξεργαστής πρέπει να προλαβαίνει να εκτελέσει όλους τους απαιτούμενους υπολογισμούς φυσικής ανάμεσα στα καρέ, καταναλώνοντας ένα ποσοστό μόνο του απαιτούμενου υπολογιστικού χρόνου.
Τα μοντέλα αυτά συνήθως προκύπτουν από κάποια *μέθοδο ολοκλήρωσης* που εφαρμόζει τους φυσικούς νόμους για μικρά χρονικά διαστήματα.
Επί παραδείγματι αντί για την αναλυτική εξίσωση τροχιάς του βλήματος, ένα προσεγγιστικό μοντέλο θα διάβαζε τη θέση, την ταχύτητα και την επιτάχυνση του βλήματος, θα θεωρούσε τα μεγέθη σταθερά για τα διαστήματα αυτά, και θα εφάρμοζε διαδοχικά το δεύτερο νόμο του Νεύτωνα για τα διαστήματα αυτά (ολοκλήρωση ως προς το χρόνο). Η διαδικασία αυτή είναι πολύ απλή – στις περισσότερες περιπτώσεις τετριμμένα απλή – για τον επεξεργαστή, εν αντιθέσει με την αναλυτική επίλυση μιας εξίσωσης τροχιάς.
Τα μοντέλα πραγματικού χρόνου χρησιμοποιούνται κατεξοχήν στα εικονικά περιβάλλοντα με φυσική και στα παιχνίδια – εκεί η ταχύτητα εκτέλεσης και ο χρόνος είναι κεφαλαιώδους σημασίας, και η φυσική δεν αποτελεί αυτοσκοπό, αλλά τμήμα ενός μεγαλύτερου συστήματος. Η φυσική εκεί θα πρέπει να συμμετάσχει στους υπολογισμούς χωρίς να μονοπωλήσει τους διαθέσιμους πόρους του συστήματος. Έτσι τα θέματα επιδόσεως του συστήματος περνάν σε πρώτο ρόλο και η ακρίβειά τους περιορίζεται: αρκεί, συνήθως, να είναι τόσο μόνο ακριβή ώστε να είναι *_προβλέψιμα_*, *_ρεαλιστικά_* και *_πιστευτά_*.
Ο *χρόνος ολοκλήρωσης* (dt) μπορεί να είναι σταθερός ή δυναμικός. Στην περίπτωση δυναμικού dt γενικά χρησιμοποιείται σε κάθε καρέ ο χρόνος που έχει διέλθει από το προηγούμενο καρέ, ή μια συγκεκριμένη αναλογία του. Δηλαδή γενικώς έχουμε, σε κάθε καρέ, ένα «βήμα επεξεργασίας» της μορφής

- Σε κάθε καρέ (χρόνος t_1), αν το προηγούμενο καρέ εμφανίστηκε στο t_0
  - Επεξεργάσου φυσική για χρόνο $dt=t_1-t_0$

Στα μοντέλα με σταθερό βήμα αντίστροφα, επεξεργαζόμαστε τη φυσική σε ένα βήμα χρόνου dt_c που είναι σταθερό και ανεξάρτητος του dt, όσες φορές απαιτείται για να καλυφθεί ο χρόνος dt. Ο υπολογισμός αυτός μπορεί να πάρει κάποιες από τις παρακάτω μορφές:
Όταν το dt είναι σταθερό και γνωστό εκ των προτέρων:

- Σε κάθε καρέ
    - Επεξεργάσου φυσική για χρόνο dt_c

Όταν απαιτείται σταθερό βήμα αλλά το dt δεν είναι γενικά γνωστό εκ των προτέρων (επαναληπτική διαδικασία):

- Σε κάθε καρέ
    - Επεξεργάσου φυσική για χρόνο dt_c
    - Αφαίρεσε το dt_c από το dt
    - Επανέλαβε για όσο το dt είναι μεγαλύτερο από 0

Γενικά, στα μοντέλα πραγματικού χρόνου που χρησιμοποιούνται σε εικονικά περιβάλλοντα συναντάμε κάποιους νόμους πολύ συχνότερα από άλλους, και έχουν να κάνουν με την μοντελοποίηση φαινομένων και νόμων που συναντάμε στην καθημερινότητα και βοηθούν στην πιστευτότητα του περιβάλλοντος. Συχνότεροι νόμοι που συναντάμε είναι της κινηματικής (Νευτώνεια μηχανική), μηχανικής και δυναμικής (μηχανική στερεού σώματος, δυνάμεις επαφής) μηχανικής παραμορφώσιμου σώματος (δυνάμεις ελατηρίου και απόσβεσης) αλλά οποιοσδήποτε φυσικός νόμος με χρονική εξέλιξη μπορεί να απαιτηθεί σε συχνότερες η σπανιότερες περιπτώσεις (μηχανική παραμορφώσιμου σώματος, στατική, αντίσταση του αέρα, ρευστομηχανική κ.α.).
Στη γενική περίπτωση, υλοποιούνται φυσικοί νόμοι με κάποιο «άμεσο» αποτέλεσμα. Σε κάποιες επαναληπτικές περιπτώσεις μπορεί επίσης να δούμε να μοντελοποιούνται κάποιες αρχές οι οποίες απαιτείται να ικανοποιούνται ως περιορισμοί (αρχή διατήρησης της ορμής, αρχή διατήρησης της ενέργειας κ.α.).
Σε μια τέτοια περίπτωση, το μοντέλο προσεγγιστικά θα φαινόταν ως εξής:
Φυσικό μέγεθος-στόχος έστω ενέργεια στο προηγούμενο καρέ = E_0)

- Στο νέο καρέ
    - Υπολόγισε την ενέργεια που προέκυψε λόγω των αλλαγών (E_1)
    - Εάν το $E_0 <> E_1$
    - (Με κάποιον αλγόριθμο), αλλοίωσε κάποια φυσικά μεγέθη ώστε το E_1 να προσεγγίσει το E_0 (ένα ή περισσότερα βήματα, αναλυτικά ή αριθμητικά)
    - (Επανέλαβε εάν απαιτείται)

Η διαδικασία αυτή της – επαναληπτικής ή όχι – ικανοποίησης περιορισμών είναι σημαντική για να μοντελοποιηθούν κάποιες συμπεριφορές που με την προηγούμενη, εμπρόσθια επίλυση είναι μάλλον δύσκολη.

2.3.3. Μοντελοποίηση σωμάτων

Η προσομοίωση συστημάτων φυσικής δεν είναι νέος κλάδος, παρά γίνεται εκτεταμένη έρευνα από το 1980 περίπου και μετά. Οι Δημήτρης Τερζόπουλος και Andrew Witkins, 1987 χρησιμοποιούν το διαχωρισμό «κινηματικής» και «δυναμικής» αλληλεπίδρασης, όπου κινηματική είναι στέρεα σώματα που κινούνται με συγκεκριμένες ταχύτητες και επιταχύνσεις ώστε να δώσουν ένα βασικό βαθμό πιστευτότητας, και δυναμικές αλληλεπιδράσεις, όπου εμφανίζονται δυνάμεις που επιτρέπουν στα σώματα να

αλληλεπιδρούν, συγκρούονται και εμφανίζουν γενικά πολύ πιο πολύπλοκη και φυσική συμπεριφορά, με ιδιαίτερη έμφαση στην παραμορφωσιμότητα και την μοντελοποίηση σωμάτων από διαφόρων ειδών τμήματα που αλληλεπιδρούν.
Μια αρκετά απλή χρήσιμη μοντελοποίηση σωμάτων είναι χρησιμοποιώντας μια Μηχανή Σωματιδίων (Particle Engine), όπου τα σώματα μοντελοποιούνται μέσω σημειακών μαζών που συνδέονται με ελατήρια. Μέσω του απλού αυτού μοντέλου έχουμε την δυνατότητα μοντελοποίησης αρκετά πολύπλοκων σωμάτων, τα οποία συντίθενται από πολλές απλές σημειακές μάζες, με χρήσεις σε εικονικά περιβάλλοντα στην Δομοστατική (πεπερασμένα στοιχεία) και άλλους τομείς (Kirk Martini, 2005). Σε αντίστοιχες εργασίες (Liliya Kharevych and Rafi (Mohammad) Khan, 2002) χρησιμοποιείται σε συνδυασμό με αλγορίθμους ανίχνευσης συγκρούσεων για την προσομοίωση σωμάτων που λειτουργούν ως ελαστικά σύνθετα αντικείμενα. Η παρούσα εργασία θα χρησιμοποιήσει για κάποιες από τις περιπτώσεις την μοντελοποίηση αυτή για την αναπαράσταση φυσικών σωμάτων.

2.4. Τρόποι χρήσης σε λογισμικό – Μηχανές φυσικής

Σε επίπεδο εφαρμογής (λογισμικού), οι φυσικοί νόμοι μπορούν να υλοποιηθούν και προσαρτηθούν με πολυάριθμους τρόπους. Γενικά, ο κύριος παράγοντας που μας ενδιαφέρει εδώ είναι εάν οι νόμοι φυσικής αποτελούν τμήμα η όχι του λογισμικού, και κατά πόσον είναι «πραγματικές» προσομοιώσεις φυσικών νόμων ή απλά «σενάρια» (scripts) που μοιάζουν μόνο με αυτούς.

2.4.1. Απουσία φυσικής, ή προ-υπολογισμένα σενάρια φυσικών νόμων.

Υπάρχει, φυσικά, η τετριμμένη περίπτωση της απουσίας φυσικής από ένα εικονικό περιβάλλον. Εκεί, το περιεχόμενο και η συμπεριφορά είτε δεν υπακούουν καθόλου σε φυσική (π.χ. παιχνίδια που δεν διαδραματίζονται σε εικονικό περιβάλλον) είτε εμφανίζουν συμπεριφορά που θα προέκυπτε από φυσική χωρίς να υπολογίζουν όμως την φυσική αυτή.
Η βασική περίπτωση που εμφανίζει ενδιαφέρον από πλευράς φυσικής είναι όταν φυσικά φαινόμενα που θα απεικονισθούν έχουν εκ των προτέρων υπολογισθεί και εφαρμόζονται αλγοριθμικά μέσα στο εικονικό περιβάλλον.
Στα παιχνίδια, αυτό μεταφράζεται στην δημιουργία προ-υπολογισμένων (τυποποιημένων, canned) συμπεριφορών και τμημάτων κίνησης αντικειμένων, οι οποίες ενεργοποιούνται από γεγονότα στο παιχνίδι (*nVidia PhysX FAQ*).
Τέτοιες περιπτώσεις εμφανίζονται συνεχώς, ακόμα και σε περιπτώσεις που διαθέτουν μηχανή φυσικής, για λόγους είτε απλότητας, μείωσης κόστους επεξεργασίας ή ευκολίας ανάπτυξης: Δεν θα άξιζε τον κόπο να υλοποιηθούν οι πλήρεις κανόνες της οπτικής για να επιτύχουμε τον γαλάζιο ουρανό στο παιχνίδι!
Θα παρατεθούν εδώ για χάρη κατανόησης κάποια παραδείγματα γεγονότων τα οποία προσομοιάζουν φυσικά φαινόμενα χωρίς τη μοντελοποίηση νόμων φυσικής.

- Προσθήκη ενός επιπρόσθετου texture (decal) μιας οπής σε ένα τοίχο όταν το χτυπήσει μια σφαίρα (Πολυάριθμα παιχνίδια FPS – Call Of Duty, Counterstrike, STALKER). Στην περίπτωση αυτή, εφόσον ανιχνευθεί σύγκρουση ενός βλήματος με ένα αντικείμενο – τοίχο, προστίθεται μια προκαθορισμένη υφή (texture) στον τοίχο αυτό. Να παρατηρηθεί ότι η ίδια η κίνηση και η τροχιά της σφαίρας συχνά **είναι** το αποτέλεσμα μοντελοποίησης φυσικής, η κρούση με αντικείμενα όμως πολύ σπάνια.

- Προ-υπολογισμένη κίνηση μιας γεωμετρίας πανιού για να ομοιάζει κίνηση στον άνεμο. Σε ένα παιχνίδι το οποίο δεν μοντελοποιεί καθόλου την έννοια του αέρα, της κατεύθυνσής του και γενικά των στοιχείων του, προστίθεται ένας αλγόριθμος όπου τα σημεία που αποτελούν μια σημαία μετακινούνται ώστε να δώσουν την εντύπωση «αεροδυναμικού» λικνίσματος.
- Βάδισμα χαρακτήρων (Σχεδόν το σύνολο των παιχνιδιών που εμφανίζουν τρισδιάστατα avatars). Στην περίπτωση αυτή μοντελοποιείται το έδαφος και οι χαρακτήρες, και οι χαρακτήρες εμφανίζονται να πατούν στο έδαφος. Η **πτώση** ενός χαρακτήρα στο έδαφος γενικά ακολουθεί κάποιον – έστω και απλοϊκό – νόμο φυσικής, το βάδισμα σχεδόν ποτέ. Στην γενική περίπτωση, διατηρείται αλγοριθμικά μια προαποφασισμένη απόσταση από το έδαφος.
- Βροχή (εμφάνιση σωματιδίων που μοιάζουν να πέφτουν τυχαία) (Σχεδόν το σύνολο των παιχνιδιών που εμφανίζουν καιρό). Στη γενική περίπτωση είναι ένα οπτικό αποτέλεσμα που προκύπτει από αντικείμενα που εμφανίζονται να κινούνται κατακόρυφα ή υπό γωνία προς τα κάτω χωρίς στην πραγματικότητα να γίνεται εφαρμογή οποιουδήποτε φυσικού νόμου.

Στο περιβάλλον της VRML, όπου δίνεται η δυνατότητα προσομοίωσης χαρακτήρα που «βαδίζει» σε προεπιλεγμένο έδαφος, ενώ δεν υπάρχουν νόμοι φυσικής για την βαρύτητα που θα «έσπρωχναν» το χαρακτήρα στο έδαφος.
Σε όλες τις παραπάνω περιπτώσεις, η εμφάνιση φυσικής στο πρόγραμμα περιλαμβάνει την μοντελοποίηση ενός συγκεκριμένου γεγονότος, ώστε όταν η το γεγονός αυτό λάβει χώρα, να εμφανιστεί ένα φυσικά πιστευτό αποτέλεσμα.
Πλεονεκτήματα:

- Δυνατότητα χειρισμού με οσηδήποτε λεπτομέρεια συγκεκριμένων περιπτώσεων
- Δυνατότητα επιλογής ανά συγκεκριμένη περίπτωση του πιο αποδοτικού τρόπου για την επιτυχία του (οπτικού ή άλλου) αποτελέσματος που επιθυμούμε
- Δυνατότητα εφαρμογής οσησδήποτε ακρίβειας, πιστευτότητας ή ρεαλισμού ανάλογα με την «ευκολία» του γεγονότος και την προσπάθεια που καταβάλλεται για τη μοντελοποίηση.

Μειονεκτήματα:

- Οποιοδήποτε γεγονός δεν μοντελοποιηθεί ρητά, δεν λαμβάνει χώρα ή δεν εμφανίζει φυσική συμπεριφορά (παράδειγμα: αντικείμενα που όταν ο χρήστης τα αφήσει, αυτά μένουν στον αέρα αντί να πέσουν).
- Ασυνέπεια. Γεγονότα που στον πραγματικό κόσμο προκύπτουν από τους ίδιους φυσικούς νόμους δεν συνδέονται λογικά, με αποτέλεσμα τις ασυνέπειες (στο παραπάνω παράδειγμα, δυο αντικείμενα μπορούν να πέφτουν με διαφορετική ταχύτητα διότι ακολουθούν διαφορετικά σενάρια)
- Κακή επεκτασιμότητα. Όσο προστίθενται γεγονότα, τόσο απαιτείται συγγραφή σεναρίων για την εκτέλεσή τους.
- Ακρίβεια. Εφ’ όσον δεν προσομοιάζεται φυσικός νόμος, είναι μικρή η πιθανότητα τα σενάρια που θα γραφούν να ταιριάζουν επαρκώς με αυτόν. Βέβαια αυτό δεν είναι υποχρεωτικό, καθ’ ότι θεωρητικά τα σενάρια θα μπορούσαν να ομοιάζουν σε φυσικούς νόμους χωρίς να τους μοντελοποιούν.

2.4.2. Ενσωματωμένη μηχανή φυσικής

Στην περίπτωση αυτή, η φυσική αποτελεί αναπόσπαστο τμήμα του λογισμικού, όπως συμβαίνει συχνά σε εξειδικευμένες προσομοιώσεις φυσικών συστημάτων (όπου η

μοντελοποίηση των φυσικών νόμων αποτελούν αυτοσκοπό), και σε απλά (casual) παιχνίδια που απαιτούν πολύ συγκεκριμένους φυσικούς νόμους που αποτελούν αναπόσπαστο τμήμα της λογικής τους.
Η περίπτωση αυτή καλύπτει ιστορικά μεγάλο φάσμα παιχνιδιών, και αποτελεί ουσιαστικά τη γέννηση της φυσικής σε παιχνίδια.
Γενικά, η διαδικασία εφαρμογής συνίσταται στην επιλογή κάποιων φυσικών νόμων και στην ένταξή τους στην εφαρμογή ως τμήμα της λογικής τους.
Ένα από τα παλαιότερα ιστορικά παραδείγματα είναι το Asteroids (Atari, 1979). Στο παιχνίδι αυτό, ο χρήστης κινεί σε δισδιάστατο χώρο ένα τρίγωνο που αντιπροσωπεύει ένα διαστημόπλοιο το οποίο μπορεί να στρίψει, να πυροβολήσει ή να εφαρμόσει ώθηση. Εδώ μοντελοποιούνται απλοποιημένοι ο Πρώτος και Δεύτερος Νόμος του Νεύτωνα ($x=x_0+v*dt$, $F=ma$), και κάποια μορφή τριβής. Η εφαρμογή της ώθησης εφαρμόζει επιτάχυνση στο σώμα, το οποίο καλύπτει απόσταση ανάλογη της ταχύτητάς του, και όταν σταματήσει η εφαρμογή της ώθησης, το διαστημόπλοιο προοδευτικά σταματά υπό την επίδραση τριβής. Παρατηρούμε λοιπόν ότι αν αφαιρέσουμε τους φυσικούς νόμους, θα ήταν αδύνατο να παιχθεί το παιχνίδι αυτό.

- **Η περίπτωση του Frontier: Elite 2**

Μια αρκετά μεταγενέστερη και πολύ ενδιαφέρουσα περίπτωση είναι το παιχνίδι Frontier:Elite 2. Το παιχνίδι αυτό, μέσα σε άλλες του λειτουργίες, είναι μια προσομοίωση διαστημικής πτήσης. Το σύμπαν του παιχνιδιού αποτελεί μια περίπλοκη και ενδιαφέρουσα αναπαράσταση του Γαλαξία, με τον παίκτη ελεύθερο να επισκεφθεί, λιγότερο ή περισσότερο κατά βούληση, ηλιακά συστήματα με έναν η περισσότερους ήλιους, κατοικημένους η μη πλανήτες, δορυφόρους, αστεροειδείς, διαστημικούς σταθμούς και άλλα αντικείμενα.
Από πλευράς φυσικής όμως το εξαιρετικό του ενδιαφέρον είναι η λεπτομερής, αρκετά ακριβής και πιστευτή προσομοίωση της βαρύτητας και της κίνησης σωμάτων στο κενό. Σε παιχνίδια διαστημικής πτήσης είναι πολύ συχνή μια απλοποιημένη μοντελοποίηση η οποία προσομοιάζει την ατμοσφαιρική πτήση : αύξηση της πρόωσης οδηγεί σε αύξηση της ταχύτητας και όχι σε αύξηση της επιτάχυνσής(X-wing, Tie Fighter, Descent, Wing Commander, Freelancer κ.α.), σαν να υπήρχε ατμοσφαιρική τριβή. Το μοντέλο αυτό οδηγεί σε ένα μη ρεαλιστικό, αλλά ευχάριστο και οικείο για τους περισσότερους παίκτες μοντέλο πτήσης.
Στο Frontier, το διαστημόπλοιο του παίκτη έχει συγκεκριμένα τεχνικά χαρακτηριστικά όπως η ισχύς των προωθητήρων του και η μάζα του. Από αυτά τα τεχνικά χαρακτηριστικά του κάθε διαστημοπλοίου **προκύπτουν** οι δυνατότητες του διαστημοπλοίου με βάση τους φυσικούς νόμους της Νευτώνειας Μηχανικής, και η πτήση του ακολουθεί σχεδόν πλήρως τους νόμους της Νευτώνειας κινηματικής και τον Νόμο της Παγκόσμιας έλξης.
Δηλαδή, σε επίπεδο ηλιακού συστήματος – όπου και διαδραματίζεται το κυρίως μέρος του παιχνιδιού – το διαστημόπλοιο του παίκτη δέχεται βαρυτικές δυνάμεις από τα αστρικά σώματα στο ηλιακό σύστημα. Οι προωθητήρες του διαστημοπλοίου προσθέτουν δύναμη προς την συγκεκριμένη κατεύθυνση η οποία ανταγωνίζεται τις δυνάμεις αυτές και δεν εφαρμόζονται τριβές : η επιτάχυνση να παραμένει σταθερή όσο εφαρμόζεται η δύναμη, οπότε το διαστημόπλοιο επιταχύνει συνεχόμενα. Το διαστημόπλοιο έλκεται από τα ουράνια σώματα ανάλογα με τη μάζα τους, και προστίθεται επιτάχυνση προς αυτά. Αποτέλεσμα αυτών είναι ότι η συμπεριφορά του κόσμου είναι μεν πολύπλοκη, αλλά ταυτόχρονα είναι **ακριβής**, **πιστευτή** και

προβλέψιμη, έως ένα βαθμό. Ταυτόχρονα, αν ο χρήστης επιλέξει να μην χρησιμοποιήσει κάποιον αυτοματοποιημένο τρόπο χειρισμού του διαστημοπλοίου (αυτόματος πιλότος), ο χειρισμός γίνεται αρκετά απαιτητικός.
Η εξαιρετικά ακριβής, ιδιαίτερα για τα δεδομένα της εποχής, μοντελοποίηση της φυσικής αποτελεί μια καινοτομία που **επιτρέπει στον παίκτη ακόμα και να χρησιμοποιήσει την ιδιαίτερη γνώση του πραγματικού κόσμου στον φανταστικό** – ένας παίκτης μπορεί να τοποθετήσει το διαστημόπλοιό του σε τροχιά ή και σε γεωστατική τροχιά, να εκτελέσει ταχείες προσεγγίσεις στους πλανήτες και να αποφύγει επικείμενες συγκρούσεις *χρησιμοποιώντας την γνώση του για τη φυσική του πραγματικού κόσμου***.**
Φυσικά, η μοντελοποίηση αυτή δεν είναι απόλυτη. Με σκοπό να δούμε τι είδους παραδοχές θα ήταν χρήσιμες στην σχεδίαση ενός εικονικού περιβάλλοντος, μπορούμε να απομονώσουμε κάποια χαρακτηριστικά, αφού θα ήταν απλά αδύνατο για τα σημερινά δεδομένα, και πιθανώς για κάποια αρκετά μελλοντικά, να μοντελοποιηθεί ολόκληρος ο γαλαξίας και να υπολογίζονται σε πραγματικό χρόνο οι αλληλεπιδράσεις μεταξύ όλων των σωμάτων. Θα ήταν επίσης εντελώς περιττό για τους σκοπούς ενός παιχνιδιού.
Περιγράφοντας δια αποτελέσματος, βλέπουμε τα εξής στοιχεία:

- Το ηλιακό σύστημα είναι το μέγιστο τμήμα του κόσμου στο οποίο λειτουργεί η βαρύτητα. Δεν μοντελοποιούνται κινήσεις και έλξεις μεταξύ των άστρων σε διαφορετικά συστήματα. Η παραδοχή αυτή γίνεται προφανής αν συνειδητοποιήσουμε ότι στα χρονικά πλαίσια του παιχνιδιού (τάξη μεγέθους λίγων ετών από την αρχή μέχρι τη λήξη του παιχνιδιού), τα αποτελέσματα των αλληλεπιδράσεων μεταξύ των άστρων είναι αμελητέα και ο γαλαξίας δείχνει στατικός (εκτός από κάποιες περιοχές στο κέντρο του, οι οποίες όμως δεν μοντελοποιούνται)
- Η βαρύτητα που μοντελοποιήθηκε φαίνεται να λειτουργεί όχι μεταξύ οσωνδήποτε σωμάτων (NxN, N-Body Gravity), αλλά ιεραρχικά, μεταξύ δηλαδή του βαρύτερου κοντινότερου αντικειμένου το οποίο ασκεί δύναμη στα ελαφρύτερα κοντινά του (1xN, 2-Body Gravity). Έτσι, βλέπουμε μια «ποιοτική» ανακρίβεια στην μοντελοποίηση η οποία επηρεάζει την εμπειρία του χρήστη αλλά όχι σε εξαιρετικό βαθμό. Δηλαδή, όταν η απόσταση του διαστημοπλοίου από αστρικά σώματα είναι «μεγάλη» (στις παρυφές ενός ηλιακού συστήματος), το σκάφος του χρήστη επηρεάζεται από το μεγαλύτερο σώμα (τον ήλιο). Όσο πλησιάζει σε άλλα σώματα (πλανήτες), μετά από κάποια απόσταση, θεωρείται ότι το σκάφος μπήκε στην «επικράτεια» του σώματος, σταματά να υπολογίζεται η βαρυτική αλληλεπίδραση με τον ήλιο και αρχίζει να υπολογίζεται σε σχέση με τον πλανήτη - η κίνηση γίνεται πλέον σχετική στον πλανήτη και όχι στον ήλιο. Η ιεραρχική αυτή κίνηση συνεχίζεται και για μικρότερα σώματα (δορυφόρους, διαστημικούς σταθμούς, αστεροειδείς), με την «επικράτεια» του καθενός να είναι μικρότερη, ανάλογα με τη μάζα του.
- Παρά τις παραπάνω παραδοχές, φαίνεται οι δυνάμεις που υπολογίζονται κάθε φορά να είναι ακριβείς και αναμενόμενες με βάση τη φυσική του πραγματικού κόσμου.
- Δημιουργείται όμως ασυνέχεια στο σημείο αλλαγής της επικράτειας.

Με τον τρόπο αυτό βλέπουμε ότι μια ενσωματωμένη μηχανή φυσικής μπορεί εύκολα να αποτελέσει και τμήμα της **λογικής του παιχνιδιού**(Game Logic) εκτός από μέσον

αύξησης του ρεαλισμού του. Παρατηρούμε εδώ εύκολα ότι αν αφαιρούταν η φυσική από το παιχνίδι Frontier, δεν θα μπορούσε να παιχθεί – δεν θα μπορούσε να κινηθεί το διαστημόπλοιο!. Αντίστροφα, σε ένα παιχνίδι που η φυσική είναι μόνο τρόπος βελτίωσης της εμπειρίας (π.χ. βαρύτητα στα αντικείμενα ώστε ότι αφήνεται να πέφτει), αν αφαιρεθεί η φυσική μπορεί να φαίνεται μη-ρεαλιστικό (τα αντικείμενα που αφήνονται αιωρούνται), δεν είναι όμως υποχρεωτικό ότι δεν μπορεί να παιχθεί.

2.4.3. Ανεξάρτητες Μηχανές / υποσυστήματα φυσικής

Τα παραπάνω παραδείγματα σε συνδυασμό με τις σύγχρονες πρακτικές σχεδίασης λογισμικού μπορούν να μας οδηγήσουν στη σκέψη ότι με τον ένα η τον άλλο τρόπο, ο υπολογισμός της φυσικής αποτελεί, ή θα μπορούσε/έπρεπε να αποτελεί, ένα **επίπεδο** *(Layer)* στο πρόγραμμα (παιχνίδι ή εικονικό περιβάλλον) που σχεδιάζουμε.
Ακόμα και αν αποτελέσει τμήμα της λογικής, ο κάθε φυσικός νόμος είναι γενικά ένα σύνολο εξισώσεων που συνδέουν σταθερές και φυσικά μεγέθη που πρέπει να ικανοποιούνται, ή (συχνά ταυτόσημα) ένα σύνολο συναρτήσεων που μας δίνουν την τιμή κάποιων φυσικών μεγεθών με βάση κάποια άλλα.
Γενικά, αγνοώντας προς στιγμήν τις ιδιαιτερότητες που κάθε φορά θα έχει το προς σχεδίαση σύστημα όμως, γίνεται φανερό ότι το παραπάνω θα μπορούσε στη γενική να υλοποιηθεί ως ένα σύνολο συναρτήσεων ή υποπρογραμμάτων που έχουν ως εισόδους και εξόδους φυσικά μεγέθη και σταθερές.
Απομονώνοντας και γράφοντας με γενικό τρόπο ώστε να μπορούν να χρησιμοποιηθούν από παραπάνω του ενός προγράμματος, τα παραπάνω θα μας οδηγούσαν σε ένα υποσύστημα, ένα ενδιάμεσο πρόγραμμα (middleware) ή βιβλιοθήκη(library), το οποίο μοναδικό σκοπό έχει τον υπολογισμό φυσικών μεγεθών με την εφαρμογή μοντελοποιημένων φυσικών νόμων.
Ο ευέλικτος αυτός τρόπος μοντελοποίησης κατόπιν θα λειτουργεί μέσω επικοινωνίας με το κυρίως λογισμικό παρέχοντάς του ως λειτουργία ή υπηρεσία τη φυσική.
Το μοντέλο αυτό απαντά όλο και συχνότερα στα εικονικά περιβάλλοντα και συγκεκριμένα στα παιχνίδια, όπου ανεξάρτητοι παράγοντες (εταιρίες) δημιουργούν πλήρεις μηχανές φυσικής οι οποίες χρησιμοποιούνται από τα παιχνίδια για την προσομοίωση φυσικών φαινομένων (Havok, nVidia PhysX, ODE, Newton, Bullet).
Παρά την απατηλή φαινομενική γενικότητα της περιγραφής που δόθηκε, προφανώς η κάθε μηχανή φυσικής έχει διαφορετικές ιδιότητες, πιθανές χρήσεις, δυνατότητες, αδυναμίες και περιορισμούς.
Όμως, τα πλεονεκτήματα και μειονεκτήματα της προσέγγισης αυτής είναι πολυάριθμα και χρήζουν ιδιαίτερης μελέτης.

- Αφού το υποσύστημα φυσικής έχει ανεξάρτητο κύκλο ζωής από το λογισμικό από το οποίο χρησιμοποιείται, έχουμε μείωση του κόστους (οικονομικού αλλά και από πλευράς πόρων και προσπάθειας), διότι δεν θα χρησιμοποιηθεί μόνο από ένα πρόγραμμα – όπως θα συνέβαινε στην ενσωματωμένη φυσική – αλλά μπορεί να επαναχρησιμοποιηθεί από πολλά προγράμματα.
- Έμμεσα προκύπτει από το παραπάνω ότι μπορούμε να έχουμε ευεργετικά αποτελέσματα στην ποιότητα της φυσικής, διότι το λογισμικό μπορεί να συνεχίσει να εξελίσσεται και να βελτιώνεται ώστε να μένει εμπορικά ή ακαδημαϊκά ανταγωνιστικό, και μπορεί να φτάσει σε πολύ υψηλό βαθμό ωριμότητας σε σχέση με την «μια – και – έξω» χρήση μιας μηχανής φυσικής

που χρησιμοποιείται από μια μοναδική εφαρμογή, όπου και ο κύκλος ζωής της θα έληγε με το πέρας της σχεδίασης του προγράμματος.

- Οι χρήστες ενός τέτοιου συστήματος είναι δυνατόν κάποιες φορές διαφανώς να λαμβάνουν τα πλεονεκτήματα από ανεξάρτητες ενημερώσεις του υποσυστήματος. Δηλαδή, εφ' όσον σχεδιαστεί σαν ένα ανεξάρτητο επίπεδο με συγκεκριμένη **διεπαφή προγραμματισμού εφαρμογής** (*API - Application Programming Interface*), η ανάπτυξη και η βελτίωσή του, όπως και η επέκτασή του, μπορούν να συνεχίσουν ανεξάρτητα από τα λογισμικά που το χρησιμοποιούν. Δηλαδή, θα μπορούσε υπό κάποιους περιορισμούς ένα πρόγραμμα να χρησιμοποιεί μια έκδοση του υποσυστήματος φυσικής η οποία στο μέλλον βελτιώνεται, και να αναβαθμιστεί το υποσύστημα φυσικής που χρησιμοποιείται με θετικά αποτελέσματα στον τελικό χρήστη χωρίς να χρειάζεται ανασχεδιασμός ή αναβάθμιση του κυρίως προγράμματος.
- Όμως, όπως και στις περισσότερες περιπτώσεις διαχωρισμού των εφαρμογών σε επίπεδα, χάνουμε μέχρι κάποιο βαθμό την ειδικότητα. Ακόμα και με τη χρήση ενός γενικού υποσυστήματος φυσικής, θα απαιτούνται εξειδικεύσεις και μετατροπές για να καλυφθούν συγκεκριμένες απαιτήσεις συγκεκριμένων προγραμμάτων.
- Επίσης, σε πολλές περιπτώσεις κάποιο υποσύστημα απλά δεν θα μπορεί να υποστηρίξει τις απαιτήσεις μιας συγκεκριμένης εφαρμογής λόγω σχεδιαστικών, προγραμματιστικών, φυσικών ή άλλων ιδιαιτεροτήτων – δεν θεωρούμε ότι μπορεί ένα πρόγραμμα να αποτελέσει την πανάκεια της φυσικής σε εικονικά περιβάλλοντα.

2.5. Φυσικοί νόμοι σε μηχανές φυσικής πραγματικού χρόνου

Για να μελετήσουμε τις μηχανές φυσικής που απαντούν στο χώρο της πληροφορικής, είναι χρήσιμο να δώσουμε κάποιες κατηγοριοποιήσεις και τομείς φυσικών νόμων και μηχανών φυσικής τα οποία απαντώνται πολύ συχνά. Οι «κλασσικές» μηχανές φυσικής που συναντάμε συχνά μπορούν να χωριστούν σε διάφορους τομείς.

2.5.1. Ανίχνευση και αντιμετώπιση συγκρούσεων

Ο τομέας αυτός της **ανίχνευσης συγκρούσεων** (*Collision Detection*) είναι κεφαλαιώδους σημασίας για την αλληλεπίδραση αντικειμένων στα εικονικά περιβάλλοντα, και θα λέγαμε ότι αποτελεί μια επιστήμη από μόνος του. Με μια «αφελή» αντιμετώπιση, ακόμα και στην απλούστατη περίπτωση όπου όλα τα σώματα θεωρούνται σφαίρες, το πρόβλημα «**ποια αντικείμενα από δεδομένο σύνολο αντικειμένων συγκρούονται σε μια δεδομένη χρονική στιγμή**» είναι τετραγωνικού χρόνου $O(n^2)$. Ο χρόνος αυτός μπορεί, σε αντίστοιχες αφελείς υλοποιήσεις, να γίνει τετραγωνικός ανά τον αριθμό των σημείων των αντικειμένων όταν αυτά γίνονται στερεά πλέγματα (meshes).
Με κατάλληλους αλγορίθμους η επεξεργασία γίνεται ευμενέστερη, και σε περιπτώσεις μπορεί να πλησιάσει στον γραμμο-λογαριθμικό O(nlogn) ή και τον γραμμικό O(n) ή χρόνο.
Αναφορικά και μόνο, ο απλούστερος και «αφελέστερος» αλγόριθμος είναι:

- Αντιμετωπίζονται όλα τα σώματα ως σφαίρες με συγκεκριμένη διάμετρο.
- Για κάθε σώμα

- Για κάθε σώμα που δεν έχει ελεγχθεί ήδη στο προηγούμενο βήμα
  - Αν το άθροισμα των διαμέτρων των δύο σωμάτων είναι μεγαλύτερο από την απόστασή τους, υπάρχει σύγκρουση.
- Επανέλαβε για όλα τα σώματα που δεν ελέγχθηκαν στο πρώτο βήμα

- Επανέλαβε για όλα τα σώματα

Όπως είναι προφανές, ο παραπάνω αλγόριθμος είναι τετραγωνικός ($O(n^2)$) ως προς τον αριθμό των σωμάτων, και αποτελεί την «απλούστατη» υλοποίηση.
Για την μείωση της πολυπλοκότητας της χρησιμοποιούνται διάφορες τεχνικές προσομοίωσης των σωμάτων, ιεράρχησής τους και διαχωρισμών του χώρου, όπως η ιεράρχηση του χώρου σε παραλληλεπίπεδα ευθυγραμμισμένα στους άξονες (Axis Aligned Bounding Boxes - AABB Trees, Gino Van De Bergen 1998), η Δυαδική Διαίρεση Χώρου (Binary Space Partitioning – BSP), Ευθυγραμμισμένα στα αντικείμενα παραλληλεπίπεδα(OBB Trees) και άλλες τεχνικές.
Οι τεχνικές αυτές γενικά ακολουθούν τη λογική της ιεράρχησης του χώρου, ώστε να αποφευχθούν έλεγχοι ζευγών σωμάτων που δεν έχουν πιθανότητα να συγκρούονται (σε γενικές γραμμές, περιέχονται πλήρως σε διαφορετικά διαμερίσματα του χώρου και συνεπώς δεν θα μπορούσαν να συγκρούονται.
Πολλές εργασίες διερευνούν την κατάσταση και συγκρίνουν τις τεχνικές στο θέμα της ανίχνευσης συγκρούσεων (Ming C. Lin & Stefan Gottschalk 1988, Matthew Moore and Jane Wilhelms, 1999)
Λόγω της πολυπλοκότητας του χώρου αυτού δεν θα αναλυθούν οι διάφορες τεχνικές σε λεπτομέρεια εδώ. Ας αναφερθεί όμως ότι είναι αρκετά σπάνια η περίπτωση προσομοίωσης φυσικού περιβάλλοντος χωρίς τη χρήση κάποιας μορφής ανίχνευσης συγκρούσεων, διότι διαφορετικά δεν μπορούν με κάποιο γενικό τρόπο να μοντελοποιηθούν δυνάμεις επαφής και κρούσεις.
Επίσης, το παραπάνω μπορούμε να το θεωρήσουμε ή όχι ακριβώς τμήμα της φυσικής του κόσμου – ίσως καλύτερα θα μπορούσαμε να το χαρακτηρίσουμε ως προαπαίτηση για την εφαρμογή κάποιων νόμων φυσικής: Για να εφαρμοσθούν φυσικοί νόμοι που αφορούν τη σύγκρουση, θα πρέπει κάπως το υποσύστημα να έχει πρώτα την πληροφορία ότι έγινε σύγκρουση. Ο παραπάνω διαχωρισμός όμως είναι μάλλον φιλοσοφικός. Ο κανόνας είναι ότι για να **λειτουργήσουν πολλοί χρήσιμοι νόμοι φυσικής,** όπως νόμοι σχετικά με κρούσεις ή νόμοι για την επαφή σωμάτων**, θα πρέπει το υποσύστημα φυσικής να διαθέτει κάποιον αλγόριθμο ανίχνευσης συγκρούσεων, είτε κάποια διεπαφή για να λαμβάνει εξωτερικά την πληροφορία για συγκρουόμενα σώματα.**
Αντίθετα, η **επίλυση/αντιμετώπιση συγκρούσεων**, δηλαδή η εκτέλεση ειδικών ενεργειών πάνω σε σώματα για τα οποία ανιχνεύθηκε σύγκρουση, **αποτελεί ακριβώς τμήμα του υποσυστήματος φυσικής**. Η επίλυση της σύγκρουσης είναι η απάντηση στο πρόβλημα «**ποιοι νόμοι εφαρμόζονται όταν δυο σώματα συγκρούονται**», και είναι σαφέστατη έκφραση της φυσικής του κόσμου.
Η κάθε σύγκρουση μπορεί να αντιμετωπισθεί με οσουσδήποτε τρόπους ανάλογα με τη γενικότερη μοντελοποίηση της φυσικής. Όπως θα δούμε παρακάτω, μπορεί π.χ. να αντιμετωπισθεί ως ελαστική κρούση με τη μηχανική στερεού σώματος και την αρχή διατήρησης ορμής και ενέργειας, να αντιμετωπισθεί με μηχανική παραμορφώσιμου σώματος, ή και άλλες παραμέτρους.

Στην παρούσα εργασία η αντιμετώπιση και επίλυση συγκρούσεων θα αντιμετωπισθεί ως μια «ομπρέλα» η οποία περιλαμβάνει όλους τους φυσικούς νόμους οι οποίοι ενδεχομένως αφορούν κάποια περίπτωση σωμάτων σε επαφή.

2.5.2. Κινηματική και Μηχανική στερεού σώματος (rigid-body mechanics)

Ο τομέας της **κινηματικής** είναι ο συχνότερος που συναντάμε ακόμα και στα πιο απλοϊκά συστήματα φυσικής. Αναφέρεται στην περιγραφή της κίνησης σωμάτων χωρίς να λαμβάνονται υπ' όψιν οι δυνάμεις που εφαρμόζονται σε αυτά. Σπανίως απαντά μόνος του σε ένα υποσύστημα φυσικής, αλλά σχεδόν πάντοτε αποτελεί ένα υποσύνολό της φυσικής και της λογικής το οποίο λαμβάνει υπ' όψιν θέσεις, ταχύτητες και επιταχύνσεις για να υπολογίσει την κίνηση σωμάτων.
Ο τομέας της δυναμικής αρχίζει να γίνεται πραγματική φυσική αφού ξεκινά να υπολογίζει και να λαμβάνει υπ' όψιν του δυνάμεις. Ένα βήμα πριν τη μηχανική στερεού σώματος βρίσκουμε τις **μηχανές σωματιδίων** (*particle engines*), που αφορούν σώματα χωρίς δομή (σημειακά), με εφαρμογές όπως οι βολές και η μοντελοποίηση βαρύτητας N-σωμάτων(N-Body simulation).
Στη μηχανική στερεού σώματος αρχίζουμε να έχουμε αντικείμενα με διαστάσεις και συνδέσμους μεταξύ τους, και δίνεται η δυνατότητα και για εφαρμογές όπως μοντελοποίηση σώματος – ανδρείκελου (*rag-doll physics*).
Στον τομέα της μηχανικής στερεού σώματος έχουν γίνει εκτεταμένες έρευνες και εργασίες, διότι σε μεγάλο ποσοστό των περιπτώσεων μπορεί να προσφέρει ικανοποιητικά και ρεαλιστικά αποτελέσματα, χωρίς να απαιτεί την κατά πολύ μεγαλύτερη πολυπλοκότητα της μηχανικής παραμορφώσιμων σωμάτων.
Σήμερα, χάρη στην άπλετη επεξεργαστική ισχύ που είναι πλέον διαθέσιμη «στο σπίτι» μέσω της προόδου της Τεχνολογίας Υπολογιστών, είναι εφικτή η μοντελοποίηση εξαιρετικά πολύπλοκων αλληλεπιδράσεων μεταξύ στερεών σωμάτων σε έναν «απλό» προσωπικό υπολογιστή. Για παράδειγμα, οι Eran Guendelman, Robert Bridson, και Ronald Fedkiw το 2003 παρουσίασαν εργασία με 2500 αντικείμενα που πέφτουν σε σωρό και μετά από λίγο ηρεμούν. Στο αντίστοιχο demo της nVidia για πράξεις στην κάρτα γραφικών μέσω Cuda ή OpenCL, σε ένα από τα παραδείγματα εμφανίζεται η σε πραγματικό χρόνο βαρυτική αλληλεπίδραση 2500 σωμάτων (nVidia Cuda/OpenCL Demos: N-Body Simulation).

2.5.3. Μηχανική παραμορφώσιμου σώματος (deformable-body mechanics)

Ο τομέας αυτός απαντάται λιγότερο συχνά από τη μηχανική στερεού σώματος, εκτός από κάποιες υποκατηγορίες του όπως τα ελατήρια. Μπορεί επίσης να αφορά μοντελοποίηση σωμάτων, μηχανική σκοινιού (rope physics), υφάσματος (cloth simulation), μοντελοποίηση οχημάτων (vehicle physics) και άλλα (Terzopoulos, Watkins 1987, 1988) (Liliya Kharevych and Rafi (Mohammad) Khan, 2002).
Η απλούστερη μοντελοποίηση που χρησιμοποιείται χωρίς να καταφύγουμε σε πολύπλοκα συστήματα και νόμους, είναι η μοντελοποίηση σημειακών μαζών συνδεδεμένων με αρκετά άκαμπτα ελατήρια, ώστε τα σώματα να αποτελούν χωροδικτυώματα.

2.5.4. Επαφές, σύνδεσμοι και περιορισμοί

Γενικώς αφορά σε όλους τους παραπάνω τομείς, αλλά έχει ιδιαίτερο ενδιαφέρον και χρησιμότητα και για αυτό αναφέρεται ξεχωριστά.

Αναφέρεται στην μοντελοποίηση συνδέσμων (joints), ελατηρίων και αποσβεστήρων (springs and dampers), περιορισμών (constraints) και γενικά συνδέσεων μεταξύ άλλων οντοτήτων.
Όπως θα δούμε παρακάτω, οι σύνδεσμοι και οι περιορισμοί αποτελούν ιδιαιτέρου ενδιαφέροντος περιπτώσεις: πολλές φορές για να μοντελοποιηθούν δεν αρκεί η εφαρμογή φυσικών νόμων εμπρόσθια στο χρόνο, αλλά απαιτείται η εύρεση λύσης εξισώσεων αντίστροφα, πράγμα που περιπλέκει ιδιαίτερα την μοντελοποίηση. Αντίστροφα όμως, η καλή μοντελοποίηση συνδέσμων και περιορισμών συμβάλλει μέγιστα στην ρεαλιστική απεικόνιση.

2.5.5. Άλλοι νόμοι

Σε απλή αναφορά ας πούμε ότι τα παραπάνω απλώς αναφέρουν κάποια σημεία ενδιαφέροντος που απαντούν συχνά στα εικονικά περιβάλλοντα και δεν πρέπει σε καμία περίπτωση να θεωρηθεί ότι αποτελούν εξαντλητική κατηγοριοποίηση. Οποιοσδήποτε νόμος ή οικογένεια νόμων φυσικής, οσοδήποτε εξωτικός, μπορεί να απαιτηθεί να μοντελοποιηθεί με βάση τις εκάστοτε απαιτήσεις. Επί παραδείγματι στην παρούσα εργασία θα δείξουμε κάποια παραδείγματα μοντελοποίησης ηλεκτρομαγνητισμού και δυναμικών πεδίων.

2.6. Εφαρμογή των φυσικών νόμων σε μηχανή φυσικής πραγματικού χρόνου

Όπως αναφέρθηκε, οι πιθανές μοντελοποιήσεις κάθε νόμου είναι πρακτικά άπειρες. Όμως, μπορούμε γενικά να παρατηρήσουμε έναν κοινό αλγόριθμο που – συχνά – ακολουθούν οι μηχανές πραγματικού χρόνου, και γενικά μπορεί να αποτελέσει – τουλάχιστον χονδροειδώς – το εφαλτήριο για το σχεδιασμό μιας τέτοιας μηχανής με συγκεκριμένα βήματα.

1. Μοντελοποίηση των «φυσικά ενεργών» σωμάτων: Σχεδιασμός των σωμάτων που θα συμμετέχουν στην προσομοίωση.
2. Μοντελοποίηση των φυσικών ιδιοτήτων: Για κάθε σώμα στο οποίο θα επιδράσει η μηχανή φυσικής, τηρούνται δομές δεδομένων με τις φυσικές του ιδιότητες, οι οποίες θα διαβάζονται, επεξεργάζονται, χρησιμοποιούνται και μεταβάλλονται από τη μηχανή φυσικής ακολουθώντας τους φυσικούς νόμους.
3. Κεντρικός κύκλος επεξεργασίας:
    a. Εκτέλεση των φυσικών υπολογισμών για κάθε καρέ (frame) της προσομοίωσης (εμπρόσθιο δυναμικό πρόβλημα)
        i. Εύρεση των δυνάμεων που δρουν με βάση τις φυσικές ιδιότητες και τις αλληλεπιδράσεις των σωμάτων ανάλογα με την κατάστασή τους στο προηγούμενο καρέ
            1. Μεταβολή της επιτάχυνσης με βάση τις δυνάμεις
            2. (Ή) μετατροπή της δύναμης σε επιτάχυνση
        ii. Μεταβολή της ταχύτητας των σωμάτων με βάση τις δυνάμεις αυτές για το χρόνο ανάμεσα στο προηγούμενο και το παρόν καρέ(dt)
            1. (ή και) Μεταβολή άλλων ιδιοτήτων των σωμάτων που επηρεάζονται από τις δυνάμεις
        iii. Μετακίνηση των σωμάτων με βάση τις ταχύτητές τους για dt και γενικά εφαρμογή των αποτελεσμάτων των φυσικών νόμων

   b. *Ικανοποίηση περιορισμών*, αν υπάρχουν (αντίστροφο δυναμικό πρόβλημα)*:*
      i. Εύρεση των «εξαναγκασμών» που επιδρούν στο σύστημα (περιορισμοί, σύνδεσμοι κ.α.)
      ii. «*Με κάποια μέθοδο*» (επίλυση εξισώσεων κ.α.), μετατροπή των προηγούμενων αποτελεσμάτων ώστε να ικανοποιηθούν οι περιορισμοί αυτοί
4. Χρήση από το κυρίως λογισμικό κάποιων από τις νέες τιμές των ιδιοτήτων αυτών (γενικά της θέσης) για την απεικόνιση των σωμάτων
5. Εμφάνιση του καρέ

Στον κεντρικό κύκλο υπάρχουν πολυάριθμες παραλλαγές που έχουν να κάνουν τόσο με τη σειρά, όσο και με τον τρόπο εφαρμογής των βημάτων, περισσότερα ή λιγότερα βήματα.
Γενικά όμως, η εφαρμογή των περισσότερων φυσικών νόμων για μια προσεγγιστική μηχανή πραγματικού χρόνου συνίσταται στην επίλυση του **εμπρόσθιου δυναμικού προβλήματος** (*forward dynamic problem)*. Το πρόβλημα αυτό είναι η **εύρεση της κατάστασης ενός δυναμικού συστήματος με δεδομένη την προηγούμενη κατάστασή του**.
Σε γενικές γραμμές τα υπό-βήματα 3.a.x αποτελούν μια ολοκλήρωση(*integration*) των φυσικών νόμων για συγκεκριμένο χρονικό διάστημα (dt).
Τα βήματα αυτά μπορούν να εφαρμοσθούν με διάφορους τρόπους, λαμβάνοντας υπ' όψιν ένα ή περισσότερα από τα προηγούμενα καρέ, να εκτελέσουν ή όχι ενδιάμεσους υπολογισμούς, να χρησιμοποιήσουν και να εφαρμόσουν διάφορες τεχνικές ανάλογα με το επιθυμητό αποτέλεσμα.
Από τη διαδικασία αυτή υπολογισμού του **εμπρόσθιου δυναμικού προβλήματος** μπορούμε να καταφύγουμε στην ολοκλήρωση στην πλειοψηφία των περιπτώσεων, δηλαδή να προσεγγίσουμε όλες τις κινήσεις και αλληλεπιδράσεις με σταθερά διανύσματα ή μεγέθη (ευθύγραμμα τμήματα στη γραφική παράσταση ως προς το χρόνο) για το κάθε τμήμα χρόνου dt του βήματος
Αντίστοιχα, η ικανοποίηση περιορισμών απαιτεί συνήθως την επίλυση του **αντίστροφου δυναμικού προβλήματος** (reverse dynamic problem). Το πρόβλημα αυτό συχνά καταλήγει πολυσύνθετο και καταφεύγουμε σε διάφορες τεχνικές για την απλοποίησή του: Αναφέρεται στην κατάσταση ότι γνωρίζουμε εκ προοιμίου πως πρέπει να λειτουργήσει ένας περιορισμός (επί παραδείγματι, μια άρθρωση), και πρέπει να καταστρωθούν και λυθούν εξισώσεις που δείχνουν τι θα πρέπει να συμβεί στο σύστημα ώστε να ικανοποιηθεί ο περιορισμός αυτός.
Αυτό μπορεί να επιτύχει συχνά με τη χρήση επαναλήψεων οι οποίες «αλλοιώνουν» το αποτέλεσμα των υπολοίπων φυσικών νόμων έως ότου επιτευχθεί αποτέλεσμα συμβατό με τον περιορισμό. Στην παρούσα εργασία δεν θα αναλυθούν σε λεπτομέρεια οι περιορισμοί και οι απαιτήσεις τους.

2.6.1. Εμπρόσθιο Δυναμικό Πρόβλημα

Η διαδικασία αυτή του τεμαχισμού του χρόνου (ή άλλου μεγέθους ως προς το οποίο άλλα μεγέθη μεταβάλλονται) σε ίσα τμήματα τα οποία θεωρούνται αμετάβλητα είναι η γνωστή Αριθμητική Ολοκλήρωση (Numerical Integration).
Από τη θεωρία γνωρίζουμε ότι αν τα τεμάχια αυτά είναι απείρως μικρά, το αποτέλεσμα είναι ακριβές για τις περιπτώσεις ολοκληρώσιμων μεγεθών.

Όμως, και στην περίπτωση που το χρονικό τμήμα δεν είναι απείρως μικρό, μπορούμε να πάρουμε αρκετά «ακριβή αποτελέσματα», και σε σχεδόν κάθε περίπτωση τόσο ακριβέστερα όσο ομαλότερη και πιο κοντά στην οριζόντια ευθεία είναι η συνάρτηση που ολοκληρώνουμε.
Στην ολοκλήρωση για επίλυση του Εμπρόσθιου Δυναμικού Προβλήματος, ανάλογα με τη σειρά ή τον τρόπο εκτέλεσης των παραπάνω βημάτων, προκύπτει ή χρησιμοποιείται διαφορετική μέθοδος ολοκλήρωσης, με διαφορετικά πλεονεκτήματα και μειονεκτήματα.
Ακολουθεί μια συνοπτική αναφορά στις πιο κοινές μεθόδους ολοκλήρωσης που απαντούν σε προσομοιώσεις φυσικών νόμων πραγματικού χρόνου, όπου για λόγους απλότητας ο μόνος στόχος είναι ο υπολογισμός της επόμενης θέσης του σώματος.
Ισχύουν δηλαδή σε κάθε περίπτωση:
F = f(x,(v)) (η δύναμη στο τρέχον καρέ είναι συνάρτηση της τρέχουσας θέσης και πιθανώς ταχύτητας ή άλλων ιδιοτήτων)
a = F/m (η επιτάχυνση ευθέως ανάλογη της δύναμης)
a = x'' (η επιτάχυνση είναι η δεύτερη παράγωγος της θέσης).
v = x' (η ταχύτητα είναι η πρώτη παράγωγος της θέσης).
Από τα παραπάνω, εφοδιασμένοι με αρχικές θέσεις, ταχύτητες και γενικά φυσικές ιδιότητες, μπορούμε, θεωρώντας σταθερά τα μεγέθη ανάμεσα σε οποιαδήποτε διαδοχικά καρέ, να τα προσεγγίσουμε μέσω ολοκλήρωσης.

Παρατηρούμε ότι εφόσον όλα τα μεγέθη θεωρούνται σταθερά για τη διάρκεια του καρέ, μπορούμε να υπολογίσουμε την νέα θέση (και άλλα φυσικά μεγέθη) με διάφορους τρόπους.

2.6.2. Πιθανά προβλήματα

Τα κυριότερα προβλήματα που έχουν να αντιμετωπισθούν σε μια μηχανή φυσικής έχουν να κάνουν με την ακρίβεια που επηρεάζει τη σταθερότητα και τη διατήρηση ενέργειας, και με την πολυπλοκότητα που επηρεάζει το κόστος υπολογισμού. Δηλαδή:

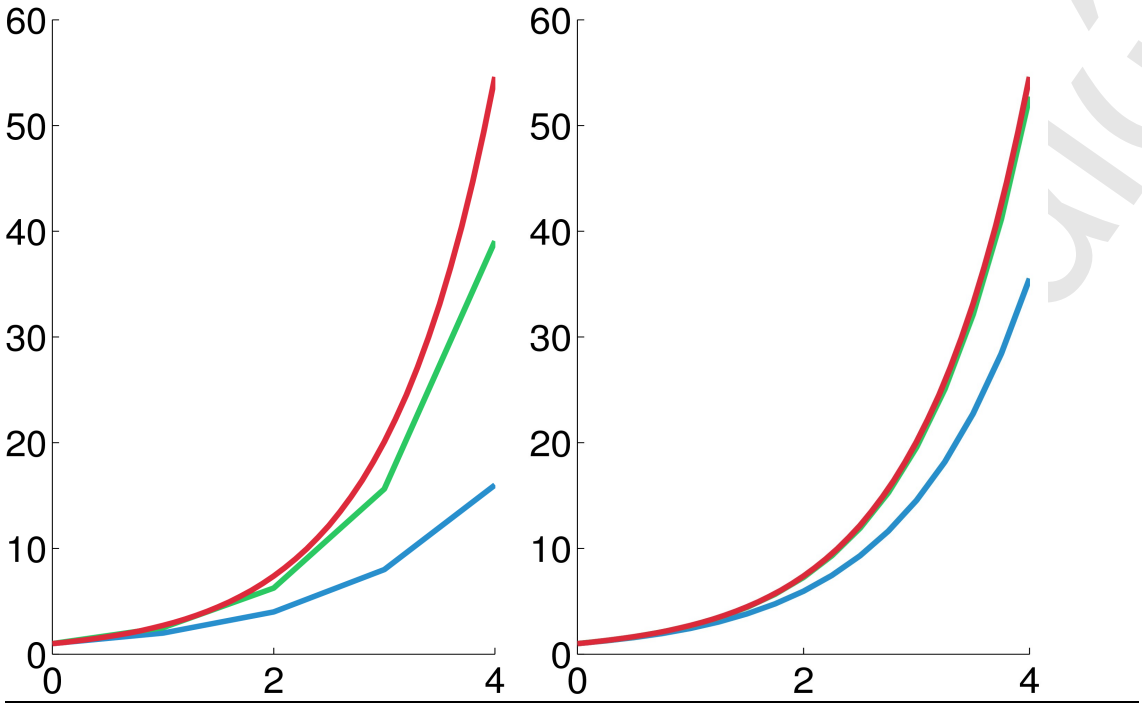
- Προσπαθούμε κυρίως να μοντελοποιήσουμε «συνεχή» μεγέθη, και όχι μεγέθη που παίρνουν μόνο διακριτές τιμές. Έτσι, γενικά χρησιμοποιούνται πράξεις κινητής υποδιαστολής. Οι αριθμοί κινητής υποδιαστολής έχουν συγκεκριμένη ακρίβεια σε σημαντικά ψηφία, οπότε στη συντριπτική πλειοψηφία των περιπτώσεων έχουμε λάθη στρογγυλοποίησης. Όταν εκτελούνται πράξεις – ιδίως προσθαφαίρεσης – ανάμεσα σε αριθμούς με πολύ μεγάλη διαφορά κλίμακας, τα λάθη αυτά μπορεί να γίνουν πάρα πολύ σημαντικά. Παράδειγμα:
    - Σε αυστηρά μαθηματικά, 100,000,000+1 = 100,000,001
    - Σε αριθμούς κινητής υποδιαστολής κατά IEEE 754 (32 bit : sign 1/mantissa 8/significand 23), λόγω της μεγάλης διαφοράς σημαντικών ψηφίων, 100,000,000 + 1= 100,000,000, δηλαδή η πράξη απαλείφεται πλήρως λόγω στρογγυλοποίησης.
- Λόγω αυτού, όταν μοντελοποιούνται φαινόμενα με πολύ μεγάλη διαφορά κλίμακας μεγεθών, μπορεί να εμφανισθούν πολύ σημαντικά σφάλματα.
- Επίσης, υπάρχει περιορισμός και στο απόλυτο μέγεθος των μεγεθών
- Αντίστοιχα, όταν τα μεγέθη θα εμφάνιζαν σημαντικές μεταβολές **κατά τη διάρκεια του καρέ**, θα εμφανισθούν σφάλματα. Επί παραδείγματι, σε μια κρούση που διαρκεί πιθανώς λιγότερο από ένα καρέ, αν η υπολογιζόμενη

δύναμη κρούσης εφαρμοσθεί για ολόκληρο το καρέ, θα προσδώσει στο σώμα ταχύτητα διαφορετική από αυτή που θα έπρεπε.

- Για να μειωθούν τα προβλήματα αυτά, συχνά εισάγονται πολυπλοκότεροι υπολογισμοί, οι οποίοι αυξάνουν την ακρίβεια, διορθώνουν ή μειώνουν τα σφάλματα, αλλά ταυτόχρονα αυξάνουν το υπολογιστικό κόστος. Παράδειγμα είναι οι επαναλαμβανόμενοι αλγόριθμοι, η υλοποίηση περιορισμών και άλλα.

2.6.3. Μέθοδοι ολοκλήρωσης

Στα παρακάτω ονομάζουμε x_0 την τιμή της θέσης στο προηγούμενο (υπό εξέταση) καρέ, με x_t η τιμή στο παρόν (προς υπολογισμό) καρέ, και $dt = t-t_0$ το χρονικό διάστημα πού έχει διέλθει από το προηγούμενο στο παρόν καρέ.
Με τον όρο Σταθερότητα παρακάτω αναφερόμαστε στην ικανότητα της προσομοίωσης υπό διάφορες συνθήκες να προσεγγίζει την πραγματικά αναμενόμενη συμπεριφορά (που θα προέκυπτε αν τα μεγέθη υπολογιζόταν με ακριβή, αναλυτική επίλυση), όσο το dt μεγαλώνει.
Η σταθερότητα έχει μέγιστη σημασία διότι όσο μικρότερο απαιτείται να είναι το dt για την επιτυχία επιθυμητού αποτελέσματος, τόσο περισσότεροι υπολογισμοί απαιτείται να γίνουν για την εφαρμογή στο κάθε καρέ.
Στην περίπτωση προσέγγισης φυσικών νόμων μπορούμε ποσοτικά να ορίσουμε τη σταθερότητα μέσω της διαφορά της συνολικής Ενέργειας του συστήματος στην προσομοίωση από τη συνολική ενέργεια που θα είχε το σύστημα με μια ακριβή επίλυση (δΕ). Όταν το δΕ αυτό έχει την τάση να αυξάνεται από καρέ σε καρέ τότε θα έχουμε ένα ασταθές σύστημα. Το ασταθές σύστημα πιθανώς θα «εκραγεί», δηλαδή θα σταματήσει να έχει έστω και ποιοτικά τη συμπεριφορά που αναμένουμε. Αντίστροφα, όταν η ενέργεια αυτή τείνει να μένει σταθερή ή αυξάνεται ελάχιστα στο χρόνο της προσομοίωσης ή παλινδρομεί γύρω από μικρές τιμές, η προσομοίωση θα είναι σταθερή, δηλαδή οι όποιες ανακρίβειες θα τείνουν να μην επηρεάζουν την ποιοτική συμπεριφορά του συστήματος.
Ας παρατηρηθεί εδώ ότι τα παρακάτω αφορούν κυρίως στην περίπτωση όπου κύριο μέλημά μας είναι η ακριβής περιγραφή της θέσεως ενός σώματος.



*Διαφορά της **πραγματικής τιμής** ενός μεγέθους (**κόκκινη γραμμή**) με την υπολογιζόμενη ποσότητα μέσω **ολοκλήρωσης Explicit Euler (μπλε γραμμή)** και **Midpoint (πράσινη γραμμή)** για βήμα 1(αριστερό σχήμα) ή 0.25 (δεξιό σχήμα).Το παραπάνω αναφέρεται σε απλή προσέγγιση της εξίσωσης y(t)=y'(t), y(0)=(1), όπου ακριβής λύση είναι η $y=e^t$*

- Μέθοδος Explicit Euler:

Η απλούστερη από τις μεθόδους ολοκλήρωσης, και η ασταθέστερη από αυτές (πολύ εύκολο να αλλάξει η ενέργεια του συστήματος λόγω ανακριβειών).
Εφαρμογή των νόμων
$a_t = f(x_0)$
$x_t = x_0 + v_0 dt$
$v_t = v_0 + a_0 dt$
Επεξήγηση:

- Υπολογίζονται οι επιδράσεις των φυσικών νόμων με βάση τις προηγούμενες τιμές των φυσικών ιδιοτήτων.
- Υπολογίζονται οι νέες θέσεις με βάση την προηγούμενη θέση και ταχύτητα.
- Υπολογίζονται οι νέες τιμές των υπόλοιπων φυσικών ιδιοτήτων με βάση τις προηγούμενες τιμές τους και τις αλληλεπιδράσεις.

Η Explicit Euler είναι γενικώς πάρα πολύ εύκολο στο να εφαρμοσθεί. Επίσης, επειδή ανά πάσα στιγμή απαιτείται η γνώση των φυσικών ιδιοτήτων μόνο του προηγούμενου καρέ, έχει τις ελάχιστες δυνατές απαιτήσεις σε μνήμη.
Όμως, από ανάλυση της συγκεκριμένης μεθόδου προκύπτουν μεγάλα σφάλματα πρώτης τάξης, με αποτέλεσμα την αρκετή αστάθεια της μεθόδου, ιδιαίτερα σε περιπτώσεις όπου στο σύστημα εφαρμόζονται δυνάμεις δεύτερης τάξης (εξαρτώμενες από την πρώτη παράγωγο της θέσης: την ταχύτητα).

- Semi – Implicit Euler

$a_t = f(x_0)$
$v_t = v_0 + a_t dt$

$x_t = x_0 + v_t dt$

Απλή και αποδοτική βελτίωση της Explicit Euler χωρίς ιδιαίτερο επιπρόσθετο κόστος.
Επεξήγηση:

- Υπολογίζονται οι επιδράσεις των φυσικών νόμων με βάση τις προηγούμενες τιμές της θέσης και των άλλων φυσικών ιδιοτήτων.
- Υπολογίζονται οι νέες τιμές των φυσικών ιδιοτήτων και η επίδραση στην ταχύτητα με βάση τις επιδράσεις αυτές.
- Υπολογίζονται οι νέες θέσεις με βάση τη νέα ταχύτητα.

Η Semi – Implicit Euler είναι σταθερότερη από την Explicit Euler, αλλά είναι ακόμα σχετικά ασταθής, ιδίως σε προβλήματα με δυνάμεις δευτέρας τάξεως. Η διαφορά είναι ότι πρώτα υπολογίζεται η νέα ταχύτητα με βάση τις δυνάμεις του προηγούμενου καρέ και κατόπιν υπολογίζεται η θέση. Το μέγιστο πλεονέκτημά της είναι ότι, στην πλειοψηφία των περιπτώσεων, αποτελεί ουσιαστικά μια «δωρεάν» βελτίωση.

- Implicit Euler

$a_t = f(x_t)$
$v_t = v_0 + a_t dt$
$x_t = x_0 + dt(v_t)$

Ιδιαίτερα σταθερή μέθοδος. Η μέθοδος αυτή όμως δεν μπορεί να εφαρμοστεί με απλό αλγόριθμο: Όλα τα μεγέθη λύνονται ως διαφορικές εξισώσεις με βάση τις τιμές τους στο νέο καρέ.
Δηλαδή, η Implicit Euler συνίσταται στο να ικανοποιηθούν ταυτόχρονα όλες οι εξισώσεις, άρα να βρεθεί αναλυτική επίλυση των φυσικών μεγεθών, αφού βλέπουμε ότι το a_t με το x_t αλληλοεξαρτώνται και πρέπει να υπολογισθούν ταυτόχρονα. Για το σκοπό αυτό σπάνια συναντάται σε μηχανές πραγματικού χρόνου για την επίλυση του εμπρόσθιου δυναμικού προβλήματος.

- Midpoint

$x_{0+dt/2} = x_0 + dt/2(v_0 + a_0 dt)$
$a_{0+dt/2} = f(x_{0+dt/2})$
$v_{0+dt/2} = v_o + a_{o+dt/2} dt/2$
$x_t = x_0 + dt (v_o + a_{o+dt/2} dt)$

Επεξήγηση:

- Υπολογίζεται μια ενδιάμεση θέση μέσω explicit Euler (η θέση η οποία θα προέκυπτε εάν το dt ήταν το μισό από αυτό που είναι)
- Υπολογισμός των τιμών των επιδράσεων για την ενδιάμεση αυτή θέση
- Υπολογισμός των τιμών των φυσικών ιδιοτήτων του επομένου καρέ με ολοκλήρωση *με βάση τις επιδράσεις της ενδιάμεσης θέσης*
- Υπολογισμός των νέων τιμών της θέσης με βάση τις νέες τιμές των φυσικών ιδιοτήτων και των επιδράσεων της ενδιάμεσης θέσης.

Η Midpoint Method είναι ο πιο δημοφιλής και αποδοτικός συμβιβασμός που χρησιμοποιείται στην πράξη ως βελτίωση από τη Semi-Implicit Euler. Το σφάλμα της είναι δεύτερης τάξης και όχι πρώτης όπως στις Euler, με αποτέλεσμα να δίνει πολύ σταθερά αποτελέσματα για συνήθεις προσομοιώσεις με δυνάμεις πρώτης και δεύτερης τάξης ως προς τη θέση (π.χ. βαρυτικές δυνάμεις, ηλεκτροστατικές δυνάμεις, ελατήρια). Προσθέτει όμως ένα επιπρόσθετο κόστος υπολογισμών για τις τιμές των μεγεθών της

ενδιάμεσης θέσης, και αντίστοιχη αύξηση της χρήσης μνήμης διότι απαιτεί ταυτόχρονη διατήρηση των επιδράσεων σε δυο θέσεις μνήμης

- Verlet, Velocity Verlet, Newton – Raphson κ.α.

Άλλες μέθοδοι που χρησιμοποιούνται κατά περίπτωση. Αναλόγως με τους εφαρμοζόμενους νόμους, τις πληροφορίες που ούτως ή άλλως μπορεί να διατηρεί το σύστημα ανάμεσα στα καρέ, με τις απαιτήσεις σε ακρίβεια και ταχύτητα, και γενικά με το σχεδιασμό και τις απαιτήσεις, αλλάζει η χρηστικότητα των μεθόδων αυτών. Δηλαδή, ο κάθε αλγόριθμος αποτελεί μια προσπάθεια αντιμετώπισης των προβλημάτων που αναφέραμε νωρίτερα, δηλαδή γενικώς, προστίθεται υπολογιστικό κόστος για να αυξηθεί ταυτόχρονα η ακρίβεια.

2.6.4. Ικανοποίηση περιορισμών – Αντίστροφο δυναμικό πρόβλημα

Εάν διατίθεται επαρκής επεξεργαστική ισχύς ώστε να εφαρμοσθεί εξαιρετικά μικρό χρονικό βήμα υπολογισμού, και χρησιμοποιούνται αριθμοί οι οποίοι έχουν επαρκή ακρίβεια για να αντιμετωπίσουν προβλήματα με συνεχή συμπεριφορά (λείες συναρτήσεις), το εμπρόσθιο δυναμικό πρόβλημα μπορεί να αρκεί για την μοντελοποίηση των νόμων. Σε πολλές εφαρμογές (όπως όλα τα παραδείγματα της εργασίας αυτής), επαρκεί πλήρως η εφαρμογή αυτή.
Όμως, υπάρχουν πολλές περιπτώσεις στις οποίες **μπορεί το εμπρόσθιο δυναμικό πρόβλημα να μην επαρκεί**. Μια από τις σημαντικότερες περιπτώσεις αυτές είναι οι **περιορισμοί**.
Εδώ αναφερόμαστε σε καταστάσεις όπου εκ των προτέρων γνωρίζουμε κάποιες συμπεριφορές τις οποίες απαιτούμε να ικανοποιηθούν, πρακτικά ασχέτως από την δυναμική που θα εφαρμοστεί.
Χαρακτηριστικά παραδείγματα είναι το έμβολο, που κινείται πάντοτε σε ευθύγραμμη τροχιά, οι μεντεσέδες μιας πόρτας, ή μια σιδηροτροχιά. Στην περίπτωση αυτή, απαιτούμε το τρένο να κινείται πάνω στις σιδηροτροχιές αυτές, οι οποίες διατηρούνται ακλόνητες. Θα μπορούσε κάποιος να υλοποιήσει όλες τις δυνάμεις που ασκούν οι σιδηροτροχιές στο τρένο, είναι όμως πολύ πιθανόν ότι κάτι τέτοιο λόγω των ανακριβειών μπορεί να οδηγήσει σε λανθασμένα αποτελέσματα, δεδομένων μάλιστα των τεράστιων δυνάμεων που μπορεί να φτάσουν να ασκούνται για ελάχιστα χρονικά διαστήματα. Ομοίως, μπορεί να υπολογίζονται οι κρούσεις ως δυνάμεις επιφανειακών ελατηρίων. Όμως, όταν ο χρόνος που διαρκεί το φαινόμενο αρχίζει να πλησιάζει ή και να γίνεται μικρότερος από το χρόνο του καρέ, τα σφάλματα γίνονται πολύ μεγάλα και η προσομοίωση μπορεί να καταρρεύσει.
Μια τεχνική που χρησιμοποιείται για την αποφυγή του προβλήματος αυτού είναι ο **περιορισμός**. Στην περίπτωση αυτή, υπολογίζεται η κίνηση π.χ. του τρένου με βάση την κινηματική του, και κατόπιν **απαιτείται** η θέση του να βρίσκεται πάνω στις ράγες, δηλαδή **περιορίζεται** πάνω σε μια συγκεκριμένη τροχιά – *η θέση και η ταχύτητά του αλλοιώνονται σαν να δεχόταν το τρένο τη δύναμη των ραγών*. Η συμπεριφορά αυτή επίσης μπορεί να χρησιμοποιηθεί για την υλοποίηση τροχών, εμβόλων, αξόνων κ.α.
Όταν περιορισθεί η τροχιά, μένουμε με το αντίστροφο πρόβλημα από άυτό που είχαμε νωρίτερα : ***Δεδομένων των συγκεκριμένων δυνάμεων που δέχεται το τρένο στο συγκεκριμένο καρέ*, και *δεδομένου ότι πρέπει να παραμείνει στην συγκεκριμένη τροχιά*, ποιες θα έπρεπε να είναι οι δυνάμεις που θα του ασκηθούν για να συμβεί**

αυτό, και πώς αλλάζει το αποτέλεσμα τον υπολοίπων δυνάμεων που του ασκούνται;

Στη γενική περίπτωση το πρόβλημα αυτό απαιτεί επίλυση και όχι απλά υπολογισμό. Δηλαδή, αναπτύσσεται κατά περίπτωση συγκεκριμένος τρόπος με τον οποίον ικανοποιούνται οι περιορισμοί αφού υπολογισθεί το εμπρόσθιο δυναμικό πρόβλημα. Μια αρκετά συνήθης προσέγγιση που ακολουθείται είναι όταν ανιχνεύεται μια κατάσταση που απαιτεί την εφαρμογή των περιορισμών, εφαρμόζονται μικρότερα βήματα υπολογισμού ώστε ο περιορισμός να ικανοποιηθεί σε μικρότερα βήματα και να μην επηρεάσει σημαντικά την ακρίβεια.

3. Σύλληψη / Ανάλυση στόχων

Στην παρούσα μεταπτυχιακή διατριβή θα παρουσιαστεί ο σχεδιασμός και η υλοποίηση ενός υποσυστήματος φυσικής, του Vesper3D.
Με βάση αυτά που αναφέρθηκαν, το Vesper3D, θα ανήκει στην κατηγορία μηχανών φυσικής πραγματικού χρόνου και θα καλύπτει τις απαιτήσεις και στόχους που περιγράφονται παρακάτω.

3.1. Στόχοι του σχεδιασμού

α) Γενικότητα φυσικών νόμων: Το Vesper3D θα έχει ως στόχο τη λειτουργία με διαφορετικούς φυσικούς νόμους, και όχι με κάποιο προκαθορισμένο σύνολό τους: Σκοπός εδώ δεν είναι η συγκεκριμένη ανάπτυξη ενός συστήματος που υλοποιεί ένα συγκεκριμένο φυσικό νόμο για ένα εικονικό περιβάλλον: θα πρέπει να μπορούν να υλοποιηθούν και προστεθούν στο σύστημα οσοιδήποτε φυσικοί νόμοι, σε δεύτερο χρόνο, και αυτοί να είναι σε θέση να λειτουργήσουν σε αυτό και να συνεργαστούν με τους υπόλοιπους.
Δηλαδή, θέλουμε να επιτύχουμε την αποσύνδεση (decoupling) του ορισμού των σωμάτων και της προσομοίωσης από την εφαρμογή των φυσικών νόμων, με σκοπό να διατηρήσουμε το υποσύστημα γενικό, ανεξάρτητο από συγκεκριμένο εικονικό περιβάλλον
β) Επεκτασιμότητα: Σε συνάρτηση με το α), θα πρέπει να είναι δυνατή η προσθήκη/αφαίρεση διαφορετικών νόμων φυσικής ως ανεξάρτητα modules τα οποία να μπορούν να ενεργοποιούνται/απενεργοποιούνται με απλή παραμετροποίηση, χωρίς να απαιτείται επαναμεταγλώττιση του υπόλοιπου κώδικα. Οι ίδιοι οι νόμοι θα είναι προϊόν ανάπτυξης και μεταγλώττισης (compiled code) και όχι σενάρια (scripts) για λόγους απόδοσης. Οι νόμοι όμως θα πρέπει να μπορούν να γραφτούν με απλό τρόπο (ανάλογα φυσικά και με τη δική τους πολυπλοκότητα) ταυτόχρονα όμως να μην υπάρχει τεχνικός περιορισμός για τη σύνταξη μεγαλύτερων και πιο σύνθετων νόμων.
γ) Γενικότητα χρήσεως: Θα πρέπει το σύστημα να παρέχει την απλούστερη δυνατή διεπαφή, ώστε να είναι δυνατό να χρησιμοποιηθεί με απλό τρόπο από διαφορετικής φύσεως προγράμματα και εικονικά περιβάλλοντα, ως ένα ***ενδιάμεσο πρόγραμμα***(***Middleware***), με την ελάχιστη δυνατή προσπάθεια από πλευράς του προγραμματιστή του.
δ) Γενικότητα φυσικών σωμάτων: Θα πρέπει να είναι δυνατή η τροφοδοσία του Vesper3D με διαφορετικά δεδομένα φυσικών σωμάτων / ιδιοτήτων, και το σύστημα να μπορεί να τα διαχειριστεί. Εδώ να προστεθεί ότι στο πνεύμα της σχεδίασης, θα πρέπει να διατηρείται μια λογική ανεξαρτητοποίησης της εκτέλεσης από την τροφοδοσία με γνωστά σώματα: Θα πρέπει το σύστημα να είναι σε θέση να διαχειριστεί ότι γνωρίζει και να αγνοήσει χωρίς σφάλματα ότι δε γνωρίζει.
ε) Ελάχιστες δυνατές παραδοχές (Minimum-assumption): Στο πνεύμα της γενικότητας και της επεκτασιμότητας, θα πρέπει να μη γίνουν υποθέσεις πάνω στο τι μπορεί και τι δεν μπορεί να χρειαστεί ο πιθανός χρήστης της εφαρμογής, παρά θα πρέπει να μένουν ανοιχτές όσο το δυνατόν περισσότερες δυνατότητες υλοποίησης. Παράδειγμα «απαράδεκτης» παραδοχής θα ήταν π.χ. ο περιορισμός του χρήστη σε μια συγκεκριμένη μέθοδο ολοκλήρωσης ή μια δομής δεδομένων για τα υλικά σώματα.
στ) Μικρό επιπρόσθετο κόστος εκτέλεσης (Minimum-*Overhead*): Θα πρέπει το σύστημα να είναι σχεδιασμένο με τρόπο που να μην προκαλεί “μεγάλο” επιπρόσθετο

φόρτο κατά τη φάση λειτουργίας πραγματικού χρόνου (δεν μας απασχολεί όμως αυτό κατά τις φάσεις εκκίνησης/αρχικοποίησης/κλεισίματος). Για το σκοπό αυτό θα πρέπει η δομή να είναι τέτοια ώστε το ενεργοποιημένο αλλά «κενό» σύστημα (χωρίς φυσικούς νόμους/ιδιότητες) να προκαλεί ελάχιστο φόρτο στο σύστημα με το οποίο είναι συνδεδεμένο.
ζ) Δυνατότητα υψηλής απόδοσης (*High performance*): Ενώ το Vesper3D από μόνο του δεν έχει μια συγκεκριμένη απόδοση (αφού δεν θα αποτελείται από συγκεκριμένο αριθμό φυσικών νόμων και ιδιοτήτων), θα πρέπει ο σχεδιασμός του να μην αποτρέπει τη χρήση τεχνικών και μεθόδων υψηλής απόδοσης, και γενικά να μην εμποδίζει στην απόδοση των φυσικών νόμων.

Παρατηρούμε εδώ ότι ανάμεσα στα δ, ε και στ, ζ θα υπάρξει ανταγωνισμός:
Η υψηλή απόδοση εξαρτάται πάρα πολύ από τη χρήση συγκεκριμένων δομών, διατάξεων στη μνήμη, τα είδη πράξεων, μέχρι και τις συγκεκριμένες εντολές που θέλουμε να εισάγουμε. Αν θέλουμε να σχεδιάσουμε ένα πραγματικά υψηλής απόδοσης σύστημα, σίγουρα θα αναγκαστούμε να εισάγουμε κάποιους περιορισμούς στο χρήστη, κάποια «κανάλια» μέσα στα οποία θα κινηθεί ώστε να μπορέσουμε να επιτύχουμε υψηλούς βαθμούς βελτιστοποίησης. Μπορούμε από τώρα να προβλέψουμε ότι ένας σημαντικός περιορισμός που θα τεθεί, θα είναι πάνω στα είδη δομών δεδομένων που θα υποστηρίζονται, αφού από τους σημαντικότερους παράγοντες απόδοσης σήμερα είναι η διάταξη της μνήμης σε συνεχή μπλοκ, και η αποδοτική χρήση της λανθάνουσας μνήμης. Ενδεικτικά αναφέρουμε ότι μια ανάγνωση από την κύρια μνήμης σε έναν υπολογιστή Intel Core 2 είναι περί τους 200 κύκλους επεξεργαστή, ενώ η αντίστοιχη ανάγνωση κοστίζει περί τους 15 κύκλους από το 3^{ο} επίπεδο λανθάνουσας, περί τους 7 από το δεύτερο, και περί τους 2 κύκλους από τον πρώτο. Συνεπώς, μια άτακτη διάταξη μνήμης μπορεί να καταστρέψει πλήρως οποιαδήποτε άλλη προσπάθεια βελτιστοποίησης, οπότε θα άξιζε ιδιαίτερη μνεία στην διαχείριση της μνήμης.
Αντίστοιχα, οι τελευταίες τεχνικές Γενικού Προγραμματισμού στη Μονάδα Επεξεργασίας Γραφικών (GPGPU programming) μέσω τεχνολογιών όπως τα OpenCL (Amd) και CUDA (nVidia) θα πρέπει να είναι δυνατόν να χρησιμοποιηθούν.

3.2. Βασικά σημεία στις επιλογές τεχνολογίας

Όσον αφορά τη γλώσσα προγραμματισμού, υπάρχουν επίσης περιορισμοί που θα πρέπει να τηρηθούν. Κατ' αρχάς, η γλώσσα συγγραφής επηρεάζει τη δυνατότητα χρήσης της μηχανής από άλλες γλώσσες προγραμματισμού.
Δηλαδή, επειδή ο σκοπός εδώ δεν είναι η συγγραφή ολοκληρωμένης εφαρμογής, αλλά ενδιάμεσου λογισμικού, αυτόματα απορρίπτονται όλες οι επιλογές που βασίζονται σε συγκεκριμένη πλατφόρμα που καταλήγει ενδιάμεσο κώδικα, όπως η Java και η C#, και όχι σε τελικό κώδικα της πλατφόρμας-στόχου.
Ο λόγος για αυτό είναι ότι από τη μία είναι εγγυημένο ότι σχεδόν όλες οι γλώσσες προγραμματισμού έχουν τη δυνατότητα να επικοινωνήσουν με τελικό (native) κώδικα μηχανής. Αντίστροφα, δεν δίνεται πάντα η δυνατότητα να επικοινωνήσουν μεταξύ τους : θα μπορούσε να αποδειχθεί ιδιαίτερα δυσχερές κώδικας Java να επικοινωνήσει με κώδικα C# ή κώδικας Matlab με Lua.
Επιπροσθέτως, η απαίτηση του ελαχίστου κόστους οδηγεί επίσης σε απόρριψη τις γλώσσες αυτές διότι δημιουργεί επιπρόσθετο κόστος η Εικονική Μηχανή (Virtual

Machine) που είναι απαραίτητο να φορτωθεί για να εκτελεστούν οσοδήποτε μικρά προγράμματα ή βιβλιοθήκες.
Συνεπώς, απαιτείται γλώσσα προγραμματισμού που καταλήγει σε κώδικα μηχανής (native), ώστε να μπορούμε να βεβαιώσουμε τη χρήση του υποσυστήματος φυσικής από άλλες γλώσσες προγραμματισμού.
Όπως θα αναλυθεί στο επόμενο κεφάλαιο (υλοποίηση), στη συγκεκριμένη υλοποίηση χρησιμοποιήθηκε το περιβάλλον προγραμματισμού Microsoft Visual Studio 2010 με το μεταγλωττιστή της Microsoft Visual C++ 2010. Άλλος εξ' ίσου κατάλληλος υποψήφιος ήταν ο μεταγλωττιστής gcc++. Θα μπορούσε με διαφορετική υλοποίηση και αν δεν μας ενδιέφερε η χρήση αντικειμενοστραφούς μοντέλου να χρησιμοποιηθεί η C++.

3.3. Βασικά σημεία στη δομή των νόμων

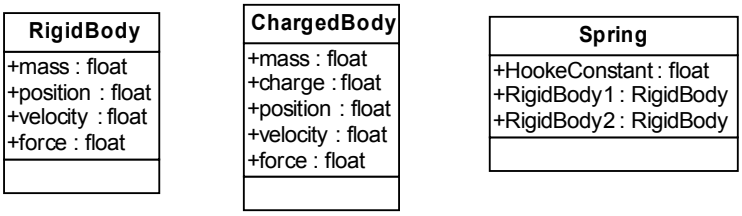
Επιλέχθηκε οι νόμοι να αποτελούν αυτοτελή, μεταγλωττισμένα τμήματα που φορτώνονται και καλούνται από κεντρικό υποσύστημα. Οι νόμοι θα είναι

- **Μεταγλωττισμένοι,** ώστε να μπορούμε να καλύψουμε τις απαιτήσεις υψηλών επιδόσεων.
- **Αυτοτελείς**, ώστε να μπορούμε να σχεδιάσουμε το κεντρικό υποσύστημα με τρόπο ώστε αυτοί να φορτώνονται δυναμικά και ανεξάρτητα μεταξύ τους.
- **Συνεπείς,** ώστε ο χρήστης-προγραμματιστής να μπορεί να γράψει τους νόμους που θέλει, να τους μεταγλωττίσει και να τους προσαρτήσει στο πρόγραμμά του. Για το σκοπό αυτό θα πρέπει να παρέχουμε στο χρήστη ένα απλό C++ Header File, και αυτός να γράψει όλες τις λεπτομέρειες υλοποίησης του συγκεκριμένου φυσικού νόμου.

Έχουμε λοιπόν ορίσει **αφηρημένη τάξη** (Abstract Class) PhysicsLayer::PhysicsLaw, (στην C++ αυτό είναι το αντίστοιχο της UML **διεπαφής**(*Interface*)), την οποία πρέπει να κληρονομήσουν όλοι οι φυσικοί νόμοι που γράφονται. Στο ίδιο Header File δίνεται και η δήλωση μίας συνάρτησης – Εργοστάσιο η οποία θα πρέπει να επιστρέφει αντικείμενο (instance) του φυσικού νόμου. Ο πλήρης κώδικας του Header File δίνεται στο παράρτημα, όπως και παράρτημα νόμου που τον χρησιμοποιεί.
Όπως θα δούμε στη δομή του κεντρικού συστήματος, για την απρόσκοπτη λειτουργία της διαδικασίας αυτής χρησιμοποιείται μια παραλλαγή του προτύπου σχεδιασμού *(design pattern)* των Gang Of Four "**Abstract Factory**", που μας επιτρέπει να δημιουργούμε εύκολα αντικείμενα που υλοποιούν τη διεπαφή PhysicsLaw από ανεξάρτητες βιβλιοθήκες dll και να τα χρησιμοποιούμε στον κώδικά μας χωρίς να γνωρίζουμε απολύτως τίποτα για την τελική τους υλοποίηση.

3.4. Βασικά σημεία στη δομή των φυσικών ιδιοτήτων

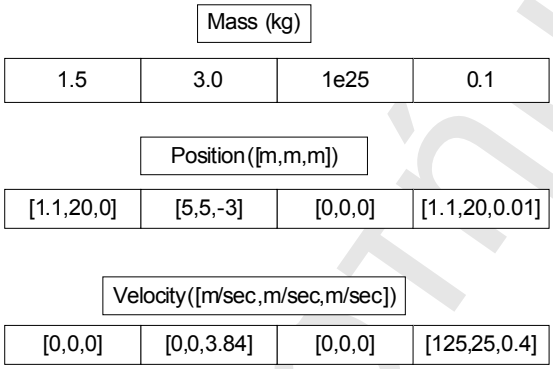
Κομβικό σημείο του σχεδιασμού ήταν η επιλογή της δομής των φυσικών ιδιοτήτων. Συνήθης δομή των αντικειμένων είναι να δίνονται οι φυσικές ιδιότητές τους στον ορισμό των τάξεών τους, και να περιέχονται εκεί τα δεδομένα τους. Δηλαδή, το κάθε αντικείμενο περιέχει «Οριζοντίως» τη θέση, την ταχύτητα, τη μάζα και όποιες άλλες πιθανές ιδιότητες περιέχει. Στην περίπτωση αυτή οι φυσικοί νόμοι επιδρούν πάνω στο αντικείμενο αυτό, διαβάζοντας τις ιδιότητες που αυτά διαθέτουν, είτε με τη χρήση interfaces που υλοποιούν τα συγκεκριμένα αντικείμενα.



Μια τέτοια όμως δομή θα μας δημιουργούσε πολυάριθμα προβλήματα στο σχεδιασμό, ιδιαίτερα στο να καλύψουμε την ad hoc προσθήκη και αφαίρεση νόμων και φυσικών ιδιοτήτων. Με τον τρόπο αυτό, θα έπρεπε ο κάθε φυσικός νόμος να επενεργεί πάνω σε συγκεκριμένα είδη αντικειμένων, εν δυνάμει οδηγώντας σε πολύπλοκες δομές οι οποίες δεν μπορούν να ενεργοποιηθούν / απενεργοποιηθούν / αλλάξουν κατά βούληση.

Αποφασίστηκε να χρησιμοποιηθεί μια εντελώς αντίστροφη δομή από αυτήν:

Όσον αφορά το Vesper3D, δεν θα υπάρχει η έννοια αντικειμένου. Το υποσύστημα θα διαβάζει φυσικές ιδιότητες οι οποίες αλληλεπιδρούν σε αντιστοιχία.
Δηλαδή, ως παράδειγμα, συντάσσεται ένα array από float, και αποτελεί τις μάζες (*mass*), ένα array από διανύσματα που αποτελεί τις ταχύτητες (*velocity)*, και ένα array από διανύσματα που αποτελεί τις θέσεις (position). Όταν εκτελεστεί ο αντίστοιχος νόμος, θεωρεί ότι τα στοιχεία στην ίδια θέση του array ανήκουν στο ίδιο σώμα. (π.χ. το mass[12] είναι η μάζα του σώματος 12, το velocity[12] η ταχύτητά του κ.ο.κ.



Αυτό από μόνο του όμως θα είχε μια μεγάλη ατέλεια στην πολύ συχνή περίπτωση που φυσικοί νόμοι μοιράζονται την ίδια ιδιότητα.
Ας σκεφτούμε την παρακάτω περίπτωση:
Σύστημα στο οποίο λειτουργεί νόμος της κινηματικής (Νεύτωνα), συν το νόμο ηλεκτροστατικών δυνάμεων του Coulomb.
Η κινηματική απαιτεί τα παρακάτω μεγέθη:

- Μάζα (ανάγνωση)
- Θέση (ανάγνωση/ μεταβολή)
- Ταχύτητα (ανάγνωση/ μεταβολή)
- Δύναμη (ανάγνωση)

Ο νόμος του Coulomb απαιτεί τα παρακάτω μεγέθη

- Θέση (ανάγνωση)
- Φορτίο (ανάγνωση)

- Δύναμη (μεταβολή)

Έχουμε 1000 τέτοια σώματα, από τα οποία όμως τελικά μόνο 40 έχουν φορτίο.
Ο νόμος του Νεύτωνα λειτουργεί αποδοτικά: εφαρμόζεται μία φορά, όσες δηλαδή απαιτούνται, για καθένα από τα 1000 αυτά σώματα.
Στην περίπτωση όμως του νόμου του Coulomb, θα αναγκαστούμε να ερευνήσουμε 1000*999/2 ζεύγη στην «αφελή» υλοποίηση, ή, με μια ελαφρά βελτιστοποίηση, στην καλύτερη περίπτωση 40*960 ζεύγη (όπου θα μελετάμε ζεύγη σωμάτων μόνο όταν το πρώτο έχει φορτίο διάφορο του μηδενός). Τελικά όμως πιθανότητα να αλληλεπιδράσουν έχουν μόνο 40*39/2 από αυτά. Αυτό προκύπτει διότι, ως περιγράφηκε, το αντικείμενο 440 θα πρέπει να έχει ένα φορτίο στη θέση **charge**[440].
Πέρα από τους άχρηστους υπολογισμούς που απαιτούνται, υπάρχει αδικαιολόγητη σπατάλη μνήμης, αφού για απαίτηση 40 θέσεων μνήμης θα δεσμεύσουμε 1000.
Παρατηρούμε δηλαδή ότι θα προκύπτουν συχνά περιπτώσεις όπου θα απαιτούντο η χρήση **αραιών πινάκων** (*sparse arrays*) με την επακόλουθη σπατάλη μνήμης.
Το πρόβλημα αυτό λύθηκε με την εισαγωγή των πεδίων που θα χρησιμοποιούν **πυκνούς πίνακες** με ευρετήρια (**indexed**). Τα ευρετήρια εδώ είναι πίνακες με δείκτες.
Στην περίπτωση αυτή θεωρούμε εκ των προτέρων ότι κάποιες φυσικές ιδιότητες δεν θα είναι παρούσες σε όλα τα σώματα (μάλιστα σε πολλές περιπτώσεις θα μας ενδιαφέρει η ελευθερία προσομοίωσης και όχι τόσο η απόδοση, οπότε θα χρησιμοποιούμε αυτήν την παραδοχή σε όλες τις ιδιότητες).
Έτσι, θα είναι στην ευχέρεια του προγραμματιστή είτε να θεωρήσει ότι μια ιδιότητα την έχουν όλα τα «ομοειδή» αντικείμενα, είτε μόνο κάποια από αυτά. Όταν μια ιδιότητα ορίζεται ως indexed, η μηχανή θα πρέπει να δημιουργήσει αυτόματα έναν επιπλέον πίνακα με όνομα *ιδιότητα_Indexes*, η κάθε θέση του οποίου θα «δείχνει» σε ποιο αντικείμενο αντιστοιχεί η ιδιότητα αυτή. Στην παραπάνω περίπτωση θα φροντίζαμε η ιδιότητα **charge** να χρησιμοποιεί δείκτες. Έτσι, θα λέγαμε ότι υπάρχουν 40 charges, και 40 δείκτες που δείχνουν σε ποιο σώμα αντιστοιχεί αυτή.
Η περίπτωση αυτή θα πρέπει να λαμβάνεται υπ' όψιν πάντοτε κατά τη σύνταξη των φυσικών νόμων, ότι δηλαδή η κάθε ομάδα φυσικών ιδιοτήτων μπορεί να είναι ή όχι indexed.
Δηλαδή, ο νόμος του Νεύτωνα θα επενεργούσε στα σώματα 0->999, αλλά ο νόμος του Coulomb θα διερευνούσε μόνο τα 40 φορτία, και διαβάζοντας από τον πίνακα charge_Indexes θα διάβαζε τη θέση και θα εφάρμοζε τη δύναμη στο σωστό αντικείμενο κάθε φορά.
Απαιτείται μόνο από τον προγραμματιστή να αναφέρει στον ορισμό της ιδιότητας ότι αυτή είναι indexed, και τότε θα τηρηθεί επιπρόσθετος πίνακας με τους δείκτες της από το PhysicsEngine.

Επίσης, τίποτα δεν μας εμποδίζει να έχουμε τελείως διαφορετικές φύσεις σωμάτων που οδηγούν σε ομάδες ιδιοτήτων που είναι «ανεξάρτητες μεταξύ τους».
Παράδειγμα, σχεδιάζουμε τους νόμους κινηματικής του Νεύτωνα για σημειακές μάζες, και χρησιμοποιούμε θέση, ταχύτητα, μάζα και δύναμη (ο νόμος χρησιμοποιεί τα παραπάνω arrays position, velocity,mass, force). Αντίστοιχα το νόμο του Hooke για ελατήρια, και πάντοτε τα ελατήρια αυτά έχουν ως άκρα σημειακές μάζες (οπότε ο νόμος χρησιμοποιεί τις έννοιες «position» και «mass» για τις σημειακές μάζες, μια

ιδιότητα «άκρα» («endpoints») που υποδηλώνει ποιες μάζες συνδέει το ελατήριο, μια ιδιότητα «φυσικό μήκος» («length») και μια ιδιότητα «σκληρότητας» (hardness). Βλέπουμε εδώ ότι, όσον αφορά το νόμο αυτό, το ελατήριο δεν έχει δική του «μάζα», ή δική του «θέση», η δική του «ταχύτητα», και δεν θα μπορούσε να έχει στο παράδειγμα αυτό. Αντίστροφα, οι έννοιες «άκρα ελατηρίου», «φυσικό μήκος ελατηρίου», «σκληρότητα hooke» (όσον αφορά τα ελατήρια) δεν έχουν ποτέ νόημα για σημειακές μάζες. Στο συγκεκριμένο δηλαδή παράδειγμα, έχουμε δύο είδη σωμάτων: σημειακά και δισδιάστατα, τα οποία είναι πλήρως ανεξάρτητα. Έτσι, θα μπορούσαμε χωρίς να χρησιμοποιήσουμε δείκτες και καταλόγους, να έχουμε δυο ομάδες από φυσικές ιδιότητες, αυτές που αναφέρονται σε σημειακά και αυτές που αναφέρονται σε δισδιάστατα σώματα. Έτσι, θα μπορούσαμε να μιλάμε για τα σημειακά σώματα 1,2,3… n και τα δισδιάστατα σώματα 1,2…m και όχι για τα σώματα 1,2,3…m+n και να αποφύγουμε τους δείκτες.

Το παράδειγμα αυτό λειτουργεί για να δείξει τον μεγάλο βαθμό ελευθερίας που αφήνουμε στον προγραμματιστή, όπου τελικά η κάθε ομάδα φυσικών νόμων του επιτρέπει να ορίσει το δικό του «σύμπαν» της προσομοίωσης με βάση τις δικές του απαιτήσεις.

Τελικά, αυτή η «κατακόρυφη» δόμηση των ιδιοτήτων μας παρέχει τα παρακάτω πλεονεκτήματα:

- Προσθήκη κατά βούληση φυσικών ιδιοτήτων στα σώματα, σε πρώτη φάση ανεξάρτητα από το αν υπάρχουν φυσικοί νόμοι να τα χρησιμοποιούν
- Κανένας επιπρόσθετος περιορισμός στις φυσικές ιδιότητες που έχει το κάθε σώμα πέρα από τους εγγενείς περιορισμούς των φυσικών νόμων
- Δυνατότητα να επιτρέπεται (για λόγους ελευθερίας) ή όχι (για λόγους επιδόσεων) η χρήση των δεικτών
- Ελευθερία στη σύνθεση των σωμάτων
- Εγγύηση, λόγω της χρήσεως των arrays, ότι τα μεγέθη που θα χρησιμοποιεί ο επεξεργαστής όσο εκτελεί τις εντολές κάποιου φυσικού νόμου θα βρίσκονται όσο το δυνατόν πιο «κοντά» και «συμπυκνωμένα» στη μνήμη, αφού η χρήση C++ arrays αντίστοιχα εγγυάται συνεχή τοποθέτηση στη μνήμη όλων των τιμών κάθε φυσικής ιδιότητας (βελτίωση του Spatial Locality of Reference)

Αντίστοιχα, βλέπουμε να μας επιβάλλεται περιορισμός στην ανάθεση μνήμης: δεν υπάρχει απόλυτη ελευθερία στο χρήστη/σχεδιαστή να τοποθετήσει στη μνήμη τις ιδιότητες που επιθυμεί. Θα πρέπει να δομήσει τα arrays των ιδιοτήτων, και είτε να χρησιμοποιήσει τους πίνακες ως έχουν, είτε να σχεδιάσει τις τάξεις του με δείκτες στις θέσεις μνήμης που θα χρησιμοποιεί το PhysicsEngine, είτε, στη χειρότερη περίπτωση, να δημιουργεί αντίγραφα των ιδιοτήτων που θέλει όταν πρέπει να τους χρησιμοποιήσει.

4. Επιλογές υλοποίησης και γλώσσας προγραμματισμού

4.1. Επιλογές ως προς την προσέγγιση

Σε αδρές γραμμές, μπορούμε να ξεχωρίσουμε κάποιες από τις βασικές τεχνικές που μπορούμε να χρησιμοποιήσουμε και θα επηρεάσουν την τελική λογική των υλοποιήσεων.
Γενικά, ο σχεδιασμός μας έχει γίνει με την αντικειμενοστραφή προσέγγιση, και για αυτό τα παρακάτω αφορούν μόνο τις γλώσσες με τέτοιες δυνατότητες. Ειδικότερα, έχουμε προσανατολιστεί προς τις γλώσσες που χρησιμοποιούν το μοντέλο κληρονομικότητας της Simula (C++, C#, Java), χωρίς όμως αυτό να αποτελεί απόλυτο περιορισμό. Το μοντέλο αυτό χρησιμοποιεί τις ονομαζόμενες **εικονικές κλήσεις** (*virtual function calls*) όπου για όλες τις μεθόδους ενός αντικειμένου στις οποίες επιτρέπεται κληρονομικότητα, στην αρχή του κάθε αντικειμένου διατηρείται ένας πίνακας που επιλέγει την συνάρτηση που θα κληθεί (*vtable*).

4.1.1. Κλασσική αλγοριθμική προσέγγιση ενός νήματος

Στην προσέγγιση αυτή (singlethreaded, algorithmic) οι μέθοδοι υλοποιούνται με τον «απλό» αλγοριθμικό τρόπο, όπου τα πάντα τρέχουν διαδοχικά σε ένα μοναδικό νήμα εκτέλεσης, και παράλληλες πράξεις εκτελούνται σε βρόχους. Η μέθοδος αυτή οδηγεί σε πολύ απλές υλοποιήσεις, και αφήνει ανοιχτές πρακτικά όλες τις γλώσσες προγραμματισμού ως επιλογές. Σε μια τέτοια προσέγγιση, οι φυσικές ιδιότητες θα μπορούσαν να αποτελούνται αποκλειστικά από πίνακες αριθμών κινητής υποδιαστολής. Τα διανύσματα θα μπορούσαν να είναι τριάδες στον ίδιο πίνακα, ή και να είναι σπασμένα σε παραπάνω από ένα πίνακες ανά συνιστώσα, κατά την επιθυμία του σχεδιαστή.
Τα πλεονεκτήματα είναι:

- Απόλυτη ελευθερία επιλογής γλώσσας υλοποίησης (ανεξάρτητο από γλώσσα)
- Μη περιορισμός σε συγκεκριμένα συστήματα (ανεξάρτητο από σύστημα)
- Γενικά, απλός και ευανάγνωστος κώδικας
- Πολλές επιλογές στη δόμηση των φυσικών ιδιοτήτων

Τα μειονεκτήματα είναι:

- Ελάχιστες επιλογές για βελτιστοποίηση της απόδοσης
- Μικρότερη απόδοση λόγω χρήσης μοναδικού νήματος(thead) σε σχέση με άλλες επιλογές.

4.1.2. Πολυνηματική αλγοριθμική προσέγγιση

Εδώ βρίσκουμε τη φυσική εξέλιξη της μονονηματικής εκτέλεσης. Και πάλι χρησιμοποιούμε απλές δομές που απαντούν σε πρακτικά όλες τις γλώσσες προγραμματισμού που μας ενδιαφέρουν. Η βασικότερη διαφορά είναι ότι στην **πολυνηματική**(*multithreaded*) προσέγγιση, όπου αυτό είναι δυνατόν, οφείλουμε να δημιουργούμε επιπρόσθετα Νήματα εκτέλεσης, τα οποία θα τρέχουν παράλληλα για να αξιοποιηθούν περισσότεροι του ενός επεξεργαστικοί πυρήνες. Με τον τρόπο αυτό επίσης εκμεταλλευόμαστε καλύτερα τις πιθανές καθυστερήσεις αναμονής σε κάποια νήματα ακόμα και σε έναν πυρήνα επεξεργασίας.
Πλεονεκτήματα

- Μεγάλη ελευθερία επιλογής γλώσσας υλοποίησης (σχεδόν ανεξάρτητο από γλώσσα)
- Μικρός περιορισμός σε συγκεκριμένα συστήματα (σχεδόν ανεξάρτητο από σύστημα)
- Αύξηση της απόδοσης σε σχέση με το ένα νήμα (με μέγιστο όριο ταχύτητας λίγο μεγαλύτερο από 100% επί τον αριθμό πυρήνων του συστήματος, στην απόλυτα ιδανική περίπτωση)

Μειονεκτήματα

- Πιθανή ανάγκη επιβολής περιορισμών σε κεντρικό επίπεδο της μηχανής
- Μεγάλη αύξηση της πολυπλοκότητας κώδικα, λόγω της ανάγκης διαχείρισης των νημάτων και της χρήσης δομών συγχρονισμού
- Ανάγκη χρήσης συγχρονισμένων δομών δεδομένων για ταυτόχρονη προσπέλαση δεδομένων

4.1.3. Διανυσματοποίηση (Vectorization)

Η διαδικασία αυτή αφορά στην εκμετάλλευση συγκεκριμένων δυνατοτήτων των σύγχρονων επεξεργαστών. Στις σύγχρονες οικογένειες επεξεργαστών Intel x86/ Amd64, (Pentium και εξής), επεκτείνονται συνεχώς τα σύνολα εντολών (MMX, SSE, AMD 3D Now!, SSE 2, SSE3, SSE4, SSE5 κ.α.). Σχεδόν αυθαίρετα μπορούμε να πούμε ότι κάθε σύγχρονος επεξεργαστής υλοποιεί τουλάχιστον το σύνολο εντολών SSE2, και πιθανώς κάποιο μεταγενέστερο. Οι νεότεροι επεξεργαστές intel της τρέχουσας γενιάς (Intel Core i7) υλοποιούν μέχρι το SSE5.3 σύνολο εντολών.
Οι επεξεργαστές που υλοποιούν το SSE2 διαθέτουν ειδικούς καταχωρητές (registers) με μέγεθος 128 bit (αντίθετα από τους «κλασσικούς» καταχωρητές των 32 ή 64 bit αναλόγως της αρχιτεκτονικής του υπολογιστή). Οι καταχωρητές αυτοί είναι εκμεταλλεύσιμοι σε επίπεδο κώδικα μηχανής ή συμβολικής γλώσσας (assembly) με διάφορους τρόπους.
Στην περίπτωσή μας, μας ενδιαφέρει η χρήση τους για παράλληλες πράξεις διανυσμάτων:
Ο κάθε καταχωρητής μπορεί να φορτωθεί με τετράδες αριθμών πλάτους 32 bit (ακεραίων ή κινητής υποδιαστολής), και να εκτελεστούν πράξεις παράλληλα στους τέσσερις αριθμούς. Μπορούν έτσι να εκτελεσθούν όλες οι κλασσικές αριθμητικές και bitwise πράξεις (+-*/|&^), και κάποιες πρόσθετες ανάλογα την αρχιτεκτονική. Σε κάποια υποσύνολα δίνεται και η δυνατότητα εκτέλεσης ιδιαίτερα σύνθετων πράξεων (π.χ. εσωτερικό γινόμενο).
Πλεονεκτήματα

- Με κατάλληλη υλοποίηση και σωστές συνθήκες, δίνεται δυνατότητα μεγάλης αύξηση της απόδοσης ακόμα και σε ένα μοναδικό νήμα εκτέλεσης (ανάλογη της υλοποίησης, τυπικό νούμερο μπορεί να είναι το 200%-300%),

Μειονεκτήματα

- Πολύ μικρή ελευθερία επιλογής γλώσσας υλοποίησης (λίγες επιλογές στη γλώσσα προγραμματισμού), αφού πρέπει είτε να χρησιμοποιηθεί κώδικας συμβολικής γλώσσας, είτε ο μεταγλωττιστής να υποστηρίζει συγκεκριμένες αρχιτεκτονικές, και οι πράξεις γράφονται σχεδόν σε συμβολική γλώσσα μέσω ειδικών εντολών του μεταγλωττιστή (compiler intrinsics). Στην περίπτωσή μας

πέφτουμε στη δεύτερη περίπτωση αφού η MSVC++2010 υποστηρίζει intrinsics για την εκτέλεση εντολών SSE.

- Ανάγκη για χρήση ιδιαίτερων δομών δεδομένων που φορτώνονται στους καταχωρητές SSE (array of float, array of int et.c.)
- Κώδικας περιορισμένος σε συγκεκριμένες οικογένειες επεξεργαστών (που όμως καλύπτουν τη συντριπτική πλειοψηφία των επεξεργαστών προσωπικών ηλεκτρονικών υπολογιστών σήμερα)
- Περιορισμός στις γλώσσες προγραμματισμού που υποστηρίζουν τις παραπάνω τεχνικές.

4.1.4. Επεξεργασία γενικής χρήσης GPU (Shaders, OpenCL, nVidia Cuda)

Από τις πλέον σύγχρονες μεθόδους που μπορούν να χρησιμοποιηθούν σήμερα είναι η χρήση της κάρτας γραφικών για την εκτέλεση πράξεων γενικής χρήσης. Η μέθοδος αυτή γνωρίζει πρωτοφανή άνθηση στις σύγχρονες μηχανές φυσικής και ιδιαίτερα στη μηχανή της nVidia (PhysX).
Οι σύγχρονοι επεξεργαστές γραφικών (κάρτες γραφικών) ακολουθούν αρχιτεκτονική μεγάλης παραλληλοποίησης για να καλύψουν τις τεράστιες ανάγκες διανυσματικών πράξεων που απαιτούν τα τρισδιάστατα γραφικά. Οι πράξεις αυτές γενικά επιτρέπουν σε μεγάλο βαθμό την παράλληλη εκτέλεσή τους, οπότε οι κάρτες γραφικών έχουν ακολουθήσει αντίστοιχα με την υλοποίηση πολλών παράλληλων επεξεργαστών. Έτσι, μια τυπική, σύγχρονη κάρτα γραφικών περιέχει δεκάδες ή εκατοντάδες επεξεργαστικές μονάδες, καθεμία από τις οποίες είναι ένα υποσύνολο μιας κλασσικής κεντρικής μονάδας επεξεργασίας.
Οι επεξεργαστές αυτοί για να καλύψουν τις σύγχρονες ανάγκες γραφικών υλοποιούν ιδιαίτερα αποδοτικές εντολές για την εκτέλεση παράλληλων, διανυσματικών εντολών και ταυτόχρονα να εκτελούν πολλές τέτοιες εντολές ταυτόχρονα.
Η ταχύτητα εκτέλεσης ενός προγράμματος που κάνει χρήση της κάρτας γραφικών για διανυσματικές πράξεις μπορεί να ξεφύγει εντελώς από τις δυνατότητες του αντίστοιχου προγράμματος υλοποιημένου για επεξεργαστή γενικής χρήσεως. Η τεράστια παραλληλοποίηση που μπορεί να εκτελέσει η κάρτα γραφικών, όταν χρησιμοποιείται σε δεδομένα που επιτρέπουν παράλληλη εκτέλεση (π.χ. πράξεις μεταξύ πινάκων) επιτρέπει τυπικά αύξηση απόδοσης της τάξεως του 50 000% (50 φορές αύξηση ταχύτητας) και μπορεί να φτάσει (σε αλγορίθμους πλήρως κατάλληλους για παραλληλοποίηση και μεγάλες ποσότητες δεδομένων, σε συστήματα με δυσανάλογα ισχυρές κάρτες γραφικών σε σχέση με επεξεργαστή) σε αριθμούς όπως το 500 000% (500 φορές αύξηση ταχύτητας).
Η χρήση των τεχνικών αυτών όμως απαιτεί σχεδόν πάντα τη χρήση ειδικών γλωσσών προγραμματισμού. Οι κλασσικότερες μέθοδοι είναι η συγγραφή ειδικών πυρήνων επεξεργασίας (*kernels* - μικρές συναρτήσεις που εκτελούνται παράλληλα κατά το μοντέλο SIMD – Single Instruction Multiple Data) σε γλώσσα για προγραμματισμό σκιαστών (shaders), και τα τελευταία χρόνια εμφανίστηκαν και δυο γλώσσες που γράφτηκαν ειδικά για αυτή τη λειτουργία: Το Cuda της nVidia (υποστηρίζει μόνο κάρτες γραφικών nVidia), και το OpenCL της Amd (υποστηρίζει όλες τις κάρτες γραφικών που παρέχουν υλοποίηση για αυτό, και όλους τους επεξεργαστές που είναι συμβατοί με SSE2).
Πλεονεκτήματα:

- Πιθανή αύξηση κατά πολλές τάξεις μεγέθους της απόδοσης του συστήματος

- Πλέον σύγχρονη μέθοδος υλοποίησης

Μειονεκτήματα:

- Ειδικοί περιορισμοί στο υλικό (συγκεκριμένες κάρτες γραφικών ή συγκεκριμένοι επεξεργαστές)
- Επιλογή από κάποιες γλώσσες προγραμματισμού που υποστηρίζονται από το εκάστοτε σύστημα
- Συγκεκριμένη γλώσσα προγραμματισμού των πυρήνων ανά υλοποίηση (GLSL, HDSL, Cuda, OpenCl)
- Ιδιάζων τρόπος γραφής για την χρήση των σχετικών προγραμμάτων
- Αναγκαστική συγγραφή των πυρήνων εκτέλεσης σε άλλη γλώσσα προγραμματισμού από τη γλώσσα της συγγραφής του προγράμματος και ανάγκη χρήσης ειδικής διεπαφής μεταξύ τους
- Ανάγκη δόμησης του προγράμματος με τρόπο που να υποστηρίζει αυτού του είδους την παραλληλοποίηση

4.2. Επιλογές ως προς τη γλώσσα

Όσον αφορά τη γλώσσα προγραμματισμού, έχουμε να επιλέξουμε ανάμεσα σε δυο βασικές κατηγορίες, αφού γλώσσες σεναρίων (scripting) απορρίπτονται ως ακατάλληλες για τη συγκεκριμένη αντικειμενοστραφή υλοποίηση και τις απαιτήσεις απόδοσης.

4.2.1. Γλώσσες που μεταγλωττίζονται σε κώδικα μηχανής (native)

C, C++, Fortran, Turbo Basic, Turbo Pascal, Visual Basic, Objective-C κ.α.
Οι γλώσσες αυτές κατά τη μεταγλώττιση παράγουν απευθείας εκτελέσιμο αρχείο για μια συγκεκριμένη αρχιτεκτονική επεξεργαστή και ένα συγκεκριμένο λειτουργικό σύστημα.
Το γεγονός δίνει διάφορα χαρακτηριστικά:

- Μη μεταφερσιμότητα. Το εκτελέσιμο που κάθε φορά θα παραχθεί είναι χρησιμοποιήσιμο από συγκεκριμένες κατηγορίες συστημάτων. Ένα εκτελέσιμο για Microsoft Windows XP σε Intel 64 bit επεξεργαστές είναι μη εκτελέσιμο σε Linux, ή και σε 32 bit σύστημα με Windows. Έτσι, θα πρέπει να παραχθούν εκτελέσιμα για όλα τα συστήματα στα οποία θα τρέχει το πρόγραμμα.
- Ικανότητα μεταγλώττισης σε πολλά συστήματα. Μπορούν να επιλεγούν γλώσσες με πολύ μεγάλο εύρος αποδοχής, για τις οποίες παρέχονται μεταγλωττιστές που παράγουν εκτελέσιμα για πάρα πολλούς διαφορετικούς επεξεργαστές και λειτουργικά συστήματα.
- Ικανότητα επικοινωνίας με άλλες γλώσσες. Αντίστροφα με τη μεταφερσιμότητα σε επίπεδο συστήματος, τα απευθείας εκτελέσιμα (προγράμματα ή βιβλιοθήκες), υποστηρίζονται για να κληθούν από σχεδόν όλες τις γλώσσες προγραμματισμού.
- Ειδίκευση. Επειδή κάθε φορά η μεταγλώττιση γίνεται σε συγκεκριμένο σύστημα, μπορούν να χρησιμοποιηθούν δομές και λειτουργίες οι οποίες παρέχουν συγκεκριμένες υπηρεσίες που εκμεταλλεύονται πολύ χαμηλού επιπέδου ικανότητες του συστήματος. (Επί παραδείγματι, πολλοί μεταγλωττιστές C++ παρέχουν λέξεις-κλειδιά οι οποίες εκμεταλλεύονται απ' ευθείας τις εντολές SSE2 των επεξεργαστών που τις υποστηρίζουν).

- Απόδοση. Η βελτιστοποίηση του προγράμματος σε επίπεδο μεταγλωττιστή μπορεί να γίνει απ' ευθείας για το σύστημα για το οποίο προορίζεται.
- Συμβιβασμοί απόδοσης: Αντίστροφα, όσο μεγαλύτερη γκάμα συστημάτων θέλουμε να υποστηρίζει το πρόγραμμα, τόσο λιγότερες από τις παραπάνω δυνατότητες μπορούμε να χρησιμοποιήσουμε.

4.2.2. Γλώσσες που διερμηνεύονται (interpreted)

Basic, Pascal, Python, Perl, Php, Lisp, Ruby, Smalltalk
Οι γλώσσες αυτές δεν μας ενδιαφέρουν στη συγκεκριμένη διατριβή, διότι η διαδικασία της διερμηνείας δεν επιτρέπει τους βαθμούς απόδοσης που μας ενδιαφέρουν και δεν επιτρέπουν πάντα εύκολη διεπαφή με άλλες γλώσσες της ιδίας οικογένειας.

4.2.3. Γλώσσες που χρησιμοποιούν περιβάλλον εκτέλεσης (Runtime environment)

Java, C#, F#, Visual Basic .Net, C++/CLR
Οι σύγχρονες αυτές γλώσσες χρησιμοποιούν ένα ενδιάμεσο επίπεδο εκτέλεσης: Εγκαθίσταται στον υπολογιστή ένα συγκεκριμένο **Περιβάλλον Εκτέλεσης(Runtime Environment),** το οποίο παρέχει μια **Εικονική Μηχανή(Virtual Machine)**. Η εικονική αυτή μηχανή έχει ένα δικό της κώδικα μηχανής που ονομάζεται συνήθως **ενδιάμεση γλώσσα (Intermediate Language),** ή αλλιώς **bytecode, IL** κ.λπ.
Αντίστοιχα, ο μεταγλωττιστής αυτών των γλωσσών δεν παράγει κώδικα μηχανής για το σύστημα προορισμού του, αλλά για τη συγκεκριμένη εικονική μηχανή.
Κατά την εκτέλεση ενός τέτοιου προγράμματος, καλείται ο ενδιάμεσος κώδικας. Τότε, φορτώνεται από το λειτουργικό σύστημα το αντίστοιχο περιβάλλον εκτέλεσης, το οποίο μεταγλωττίζει ή διερμηνεύει τον ενδιάμεσο κώδικα σε κώδικα μηχανής, και τον εκτελεί.
Οι γλώσσες αυτές παρέχουν πολυάριθμα και σημαντικά πλεονεκτήματα αλλά και αντίστοιχα πολυάριθμα και σημαντικά μειονεκτήματα.

- Μεταφερσιμότητα αρχιτεκτονικής. Εφόσον το σύστημα προορισμού διαθέτει το αντίστοιχο περιβάλλον εκτέλεσης, ο μεταγλωττισμένος κώδικας μπορεί να χρησιμοποιηθεί αυτούσιος από όλα τα διαφορετικά συστήματα ασχέτως αρχιτεκτονικής
- Μεταφερσιμότητα λειτουργικού συστήματος. Θεωρητικά, εφόσον το σύστημα προορισμού διαθέτει το αντίστοιχο περιβάλλον εκτέλεσης, ο κώδικας μπορεί να χρησιμοποιηθεί ασχέτως λειτουργικού συστήματος. Πρακτικά, δεν υπάρχουν πάντα υλοποιήσεις εικονικών μηχανών για όλα τα λειτουργικά (επί παραδείγματι, κατά τη στιγμή της συγγραφής το περιβάλλον CLR της Microsoft υπάρχει μόνο για Microsoft Windows και μόλις γίνονται τα πρώτα βήματα μιας υλοποίησης για Linux).
- Μεγάλο αρχικό κόστος εκτέλεσης. Το κάθε πρόγραμμα για να εκτελεστεί θα πρέπει πρώτα να φορτωθεί η αντίστοιχη εικονική μηχανή. Κατόπιν, θα πρέπει το πρόγραμμα είτε να διερμηνεύεται (μεγάλο κόστος απόδοσης κατά την εκτέλεση), είτε να μεταγλωττιστεί επί τόπου *(just-in-time)* (μεγάλο αρχικό κόστος).
- Μεγάλο αποτύπωμα στη μνήμη (memory footprint): Ασχέτως του πόσο απλό είναι ένα πρόγραμμα, θα πρέπει να φορτωθεί το περιβάλλον εκτέλεσης για να

εκτελεσθεί. Στον υπολογιστή του συντάκτη (Microsoft Windows 7 / .Net 4.0), το βασικό πρόγραμμα "Hello World" για C#/.Net απαιτεί περί τα 2 MB μνήμης για την εικονική μηχανή CLR (*Common Language Runtime*), ενώ αντίστοιχα απαιτεί περί τα 300 KB σε MS VC++, ενώ περί τα 30bytes σε assembly).

- Ικανότητα επί τόπου βελτιστοποίησης. Το περιβάλλον εκτέλεσης έχει θεωρητικά απόλυτη γνώση των δυνατοτήτων του συστήματος όπου τρέχει, οπότε μπορεί θεωρητικά να τρέξει επί τόπου βελτιστοποιήσεις κατάλληλες για αυτό, και να εκμεταλλευθεί πλήρως την αρχιτεκτονική του. Όμως, η ικανότητα αυτή είναι άρρηκτα συνδεδεμένη με την υλοποίηση της συγκεκριμένης εικονικής μηχανής, και είναι εντελώς έξω από τον έλεγχο του προγραμματιστή. Συνεπώς ο προγραμματιστής σε πολύ μεγάλο βαθμό μεταφέρει στην εικονική μηχανή την ευθύνη βελτιστοποίησης του προγράμματός του.
- Κακή επικοινωνία με άλλες γλώσσες. Πρακτικά όλες οι γλώσσες που δεν καταλήγουν σε εκτελέσιμο κώδικα μπορούν παρόλα αυτά να εκτελέσουν κλήσεις σε αυτόν. Όμως, σπάνια παρέχεται η δυνατότητα για το αντίστροφο και την επικοινωνία μεταξύ τους : Είναι δυνατόν να εκτελεστούν κλήσεις από Java προς κώδικα μηχανής και από C#/.Net προς κώδικα μηχανής, αλλά πάρα πολύ δύσκολο να εκτελεσθούν κλήσεις από Java σε C#/.Net και αντίστροφα.

4.3. Τελική επιλογή

Όσον αφορά το θέμα της γενικής προσέγγισης που αναφέρθηκε παραπάνω, θα πρέπει να διατηρηθούν κατά το δυνατόν ανοιχτές οι επιλογές του προγραμματιστή των φυσικών νόμων.
Στο επίπεδο της μηχανής φυσικής, δεν θεωρείται απαραίτητη η εσωτερική εφαρμογή πολυνηματικής εκτέλεσης (μελετάται όμως για μελλοντική εισαγωγή). Ο λόγος είναι ότι επειδή ο κάθε προγραμματιστής/χρήστης του Vesper3D θα κάνει τις δικές του επιλογές όσον αφορά τον κώδικα των φυσικών νόμων, δεν θα πρέπει η μηχανή να του επιβάλλει εκ προοιμίου συγκεκριμένους περιορισμούς συγχρονισμού στην προσπέλαση των φυσικών ιδιοτήτων.
Αντίστροφα, στο επίπεδο φυσικού νόμου θα πρέπει να υποστηρίζεται η πολυνηματική εκτέλεση. Επειδή όμως δεν είναι ο αρχικός συντάκτης της μηχανής αυτός που θα υλοποιήσει τελικά τους εκάστοτε φυσικούς νόμους, αρκεί εδώ απλά να μην αποτραπεί η χρήση τους. Τελικά δηλαδή, ο συγχρονισμός εδώ θα πρέπει να γίνεται σε επίπεδο φυσικού νόμου και όχι σε επίπεδο μηχανής.
Αντίστοιχα, θα πρέπει να υποστηρίζεται πλήρως η διανυσματοποίηση των μεγεθών, συνεπώς (ομοίως με πριν) η χρήση ή όχι προγραμματισμού GPGPU επίσης θα πρέπει να γίνει σε επίπεδο φυσικού νόμου, ενώ η μηχανή θα πρέπει ούτε να την αποτρέπει ούτε να την επιβάλλει.

Σε επίπεδο γλώσσας, απορρίφθηκαν οι διερμηνευόμενες γλώσσες προγραμματισμού για λόγους ταχύτητας.
Επιλέχθηκε να γίνει η υλοποίηση σε γλώσσα που καταλήγει σε εκτελέσιμο κώδικα. Ο παράγοντας που βάρυνε περισσότερο στην επιλογή είναι η διαγλωσσική επικοινωνία και οι ανοιχτές δυνατότητες χρήσης τεχνολογιών. Θέλουμε το υποσύστημα φυσικής Vesper3D να μπορεί να χρησιμοποιηθεί από άλλα συστήματα ασχέτως της γλώσσας

που γράφτηκαν. Αντίστροφα, το να απαιτείται μια επιπρόσθετη μεταγλώττιση ανά υποστηριζόμενο σύστημα δεν κρίνεται ως ιδιαίτερα αποτρεπτικό.
Επίσης διατηρήθηκε η ίδια γλώσσα με μεταγλωττιστή στη σύνταξη των φυσικών νόμων – κρίθηκε ότι με αυτόν τον τρόπο ο προγραμματιστής θα μπορεί να αντλήσει τη μέγιστη απόδοση από αυτήν, και δε θα περιοριστεί από τις δυνατότητες μιας συγκεκριμένης εικονικής μηχανής.

Τελικά, επιλέχθηκε η γλώσσα C++, και η συγκεκριμένη υλοποίηση πραγματοποιήθηκε σε Microsoft Visual C++ 2010. Εκτός από τα γενικά πλεονεκτήματα όλων των μεταγλωττιζόμενων γλωσσών, η C++ έχει επιπλέον την ιδιαιτερότητα ότι είναι ταυτόχρονα πλήρως αντικειμενοστραφής, αρκετά χαμηλού επιπέδου, και έχει την καλύτερη δυνατή υποστήριξη από API γραφικών (OpenGL, DirectX κ.α.), πράγμα που καλύπτει σε μεγάλο βαθμό τι ανάγκες του πιθανού χρήστη του Vesper3D για παιχνίδια και προσομοιώσεις. Επιπλέον, υπάρχει πληθώρα μεταγλωττιστών υψηλής ποιότητας (Microsoft, Gnu, Intel, Borland), και ανάμεσα σε αυτούς πάρα πολλοί υποστηρίζουν επεκτάσεις χρήσης διανυσματικών πράξεων SSE χωρίς να καταφύγουμε σε συμβολική γλώσσα, μέσω των λεγόμενων compiler intrinsic functions που υποστηρίζουν απ' ευθείας τους 128bit registers και τις εντολές SSE2 (Intel C++, Microsoft Visual C++, gcc++), δυνατότητα που όπως θα δούμε παρακάτω επιτρέπει την ανάπτυξη φυσικών νόμων υψηλής απόδοσης με αρκετά εύκολη υλοποίηση.

5. Σχεδιασμός και υλοποίηση του υποσυστήματος φυσικής Vesper3D

Λόγω της κεφαλαιώδους σημασίας που παίζει η υποστήριξη της κατακόρυφης δομής των φυσικών ιδιοτήτων, το υποσύστημα φυσικής ονομάστηκε **Vesper3D**: **VE**rtical **S**tructure **P**hysics lay**ER 3D.**
Οι λειτουργίες του Vesper3D που δεν έχουν περιγραφεί ακόμα αφορούν στις παρακάτω λειτουργίες:

- Αρχικοποίηση του συστήματος
  - Φόρτωμα των φυσικών νόμων
  - Φόρτωμα και περιγραφή των φυσικών ιδιοτήτων
- Χρήση του συστήματος
  - Κλήση των σχετικών συναρτήσεων (Χρήση των φυσικών νόμων)
  - Επικοινωνία με το αρχικό πρόγραμμα
    - Προσθήκη νέων φυσικών ιδιοτήτων
    - Ανάκτηση ανανεωμένων τιμών φυσικών ιδιοτήτων
- Βοηθητικά αρχεία και κώδικας

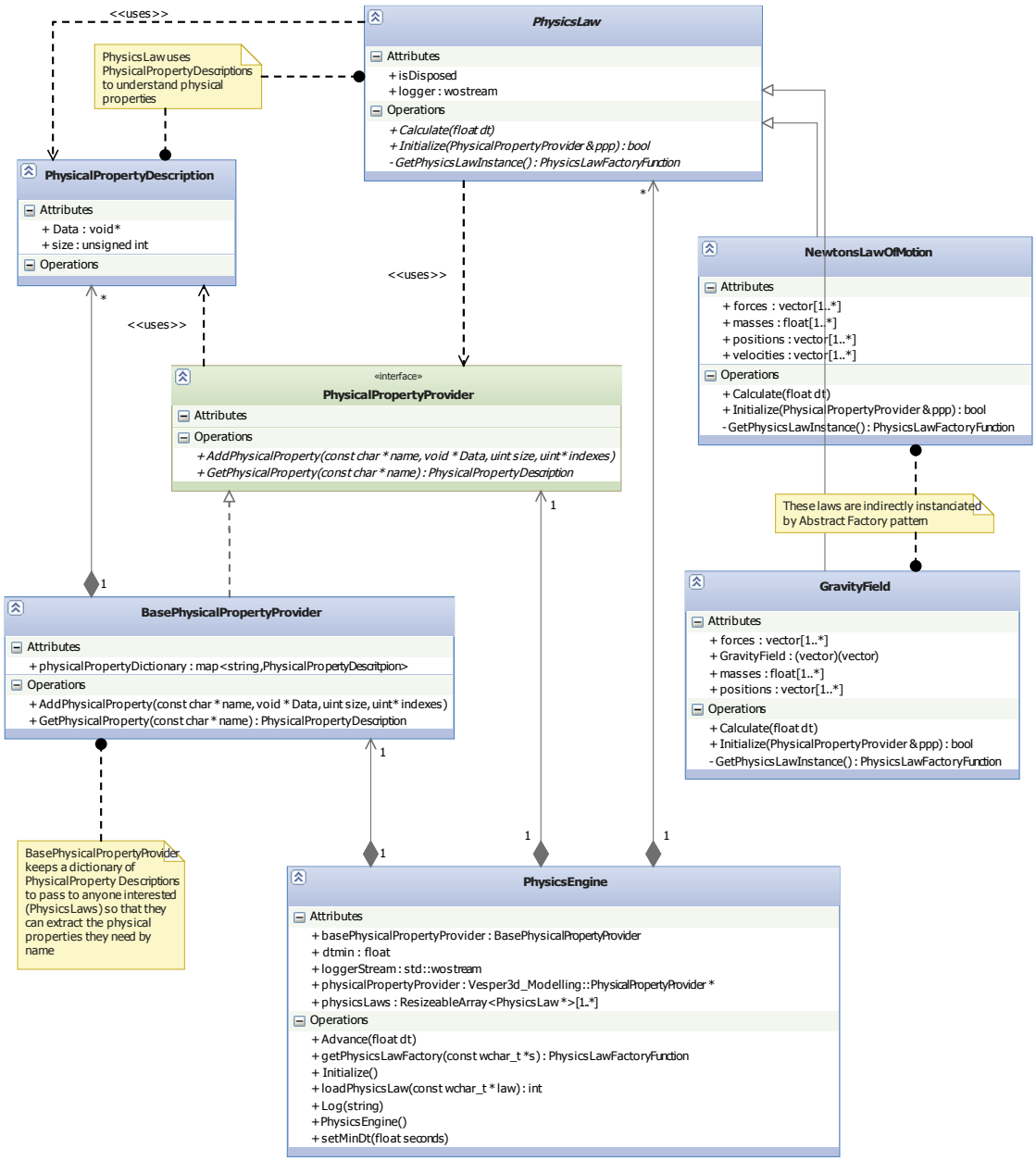
Η δομή του συστήματος περιγράφεται στο παρακάτω UML διάγραμμα Τάξεων. Στο διάγραμμα αυτό φαίνονται και δυο υλοποιήσεις φυσικών νόμων, όπου πλέον τα PhysicalPropertyDescriptions έχουν λάβει την «κανονική», χρησιμοποιήσιμη υπόστασή τους ως *arrays* που ο νόμος γνωρίζει πώς να χρησιμοποιήσει.

Εδώ αξίζει να σημειωθεί μια σημαντική ιδιαιτερότητα.
Η κατακόρυφη δομή ιδιοτήτων αποτελεί πρόταση και παρακίνηση της υλοποίησης, όχι εξαναγκασμό. Η σχεδίαση αυτή μπορεί να χρησιμοποιηθεί με «εκφυλισμό» της σε μοναδική φυσική ιδιότητα με τον παρακάτω τρόπο:
Αν ορισθεί δομή (*struct*) αντικειμένου με «οριζόντια» δομή ιδιοτήτων, δηλαδή αντικείμενο που περιέχει τις φυσικές ιδιότητές του, μπορεί δείκτης της δομής αυτής να περαστεί ως PhysicalProperty στον PhysicalPropertyProvider. Με τον τρόπο αυτό, θα μπορούσε επίσης να γραφτεί «φυσικός νόμος» ο οποίος να χρησιμοποιεί κάποιες ή όλες αυτές τις ιδιότητες, πιθανότατα συγχωνεύοντας άλλους φυσικούς νόμους. Αντίστοιχα, θα μπορούσαν να ορισθούν περισσότεροι φυσικοί νόμοι που να εκτελούνται αντίστοιχα πάνω στις ίδιες ή άλλες δομές, χωρίς να υπεισέρχεται κάποιος περιορισμός στη διαδικασία αυτή.
Σε ιδιαίτερες περιπτώσεις μια τέτοια χρήση μπορεί να αποβεί χρήσιμη, αλλά γενικά θα λειτουργήσει ενάντια στην επεκτασιμότητα του συστήματος. Παρ' όλα αυτά, η δυνατότητα αυτή δεν έχει αφαιρεθεί, αφού επιτρέπεται η χρήση οποιουδήποτε αντικειμένου ως *Data* της κάθε φυσικής ιδιότητας.
Παρατηρούμε δηλαδή ότι η «φυσική ιδιότητα» ως όρος είναι λίγο ελαστικός. Οποιαδήποτε δεδομένα χρειάζονται για τη χρήση κάποιου φυσικού νόμου μπορούν να ονομαστούν φυσική ιδιότητα, και μάλιστα σε κάποιες ιδιαίτερες περιπτώσεις, οι ιδιότητες αυτές μπορεί να αποτελούν συνθέσεις του προγραμματιστή για συγκεκριμένο σκοπό (παράδειγμα, θα μπορούσε να ορισθεί φυσική ιδιότητα «GravityData» που να είναι ένα C++ float[] που να περιέχει τη θέση ενός σώματος στις πρώτες 3 θέσεις και τη μάζα στην τέταρτη, ή και μια εντελώς σύνθετη ιδιότητα «RigidBodyData» που να περιέχει *struct* ορισμένο από τον προγραμματιστή που να περιέχει όλες τις φυσικές

ιδιότητες των σωμάτων σε ένα *struct*. Η μοναδική απαίτηση σε κάθε περίπτωση είναι ο κάθε φυσικός νόμος να γνωρίζει ακριβώς πως θα χρησιμοποιήσει τις πληροφορίες αυτές.

Στο διάγραμμα τάξεων UML απεικονίζονται με πράσινο οι διεπαφές (interfaces). Όμως, η γλώσσα C++ δεν υποστηρίζει ακριβώς τις διεπαφές : η λειτουργικότητα αυτή υλοποιείται μέσω μιας αφηρημένης (abstract) τάξης η οποία περιέχει μόνο – κατά το ιδίωμα της C++ – «pure virtual» (αμιγώς εικονικές) μεθόδους, δηλαδή μεθόδους χωρίς υλοποίηση. Στον κώδικα εμφανίζονται σαν κανονικές τάξεις που περιέχουν μέλη της μορφής:
return_type **virtual** method_name(*parameters)* **= 0;**
Η σύνταξη αυτή εκφράζει ότι η συγκεκριμένη μέθοδος είναι ***εικονική*** (μπορεί *να παρακαμφθεί (override) σε υποτάξεις της*) και ***αφηρημένη*** (*δεν διαθέτει απτή υλοποίηση στην παρούσα τάξη*). Κάθε τάξη που περιέχει τέτοιες μεθόδους θεωρείται από τη C++ ***αφηρημένη****(abstract)*. Αν μια τέτοια τάξη δεν περιέχει καθόλου πεδία, αποτελεί μια, κατά την ορθόδοξη αντικειμενοστραφή σχεδίαση, ***διεπαφή(interface).***

5.1. Διάγραμμα τάξεων του Vesper3D



5.2. Επεξήγηση του διαγράμματος / Περιγραφή των τάξεων

5.2.1. class PhysicsLaw (abstract):

Αφηρημένη τάξη, η οποία περιγράφει τη δομή που πρέπει να υπακούουν όλοι οι φυσικοί νόμοι.

Πεδία

bool *isDisposed*:
Λογική τιμή που περιγράφει αν ο νόμος είναι έτοιμος να τρέξει. Τιμή true σημαίνει ότι ο νόμος είναι μη αρχικοποιημένος, τιμή false περιγράφει ότι ο νόμος είναι αρχικοποιημένος. Αντίστοιχα, σε περίπτωση νόμων που πρέπει να κάνουν απελευθέρωση πόρων συστήματος, τιμή true σημαίνει ότι είναι απελευθερωμένοι οι πόροι, τιμή false ότι είναι σε χρήση.

std::wostream * *logger*:
Δείκτης σε ρεύμα bytes που γενικά αναμένεται να παρέχει στο νόμο το αντικείμενο PhysicsEngine που θα τον αρχικοποιήσει, και θα χρησιμοποιείται από το νόμο για την αποστολή γενικών πληροφοριών κειμένου (logging).

Λειτουργίες

bool *Initialize* (PhysicalPropertyProvider & ppp) – *(abstract)*
Η μέθοδος αυτή, όταν υλοποιηθεί σε κάποια τάξη που κληρονομεί το PhysicsLaw, εκτελείται από αντικείμενο PhysicsEngine για να πραγματοποιήσει την αρχικοποίηση του νόμου.
Ο τύπος PhysicalPropertyProvider είναι η διεπαφή που επιτρέπει στο νόμο την ανάσυρση των ιδιοτήτων που χρειάζεται. Όπως περιγράφεται παρακάτω, PhysicalPropertyProvider είναι ένα αντικείμενο που διατηρεί λεξικό φυσικών ιδιοτήτων.

void *Calculate*(float dt) – *(abstract)*
Η μέθοδος αυτή είναι ουσιαστικά η καρδιά της μηχανής φυσικής Vesper3D. Υλοποιείται στις τάξεις - παιδιά του PhysicsLaw, και καλείται σε κάθε καρέ από το PhysicsEngine ώστε ο νόμος να επενεργήσει πάνω στις φυσικές ιδιότητες.

extern "C" PhysicsLaw* GetPhysicsLawInstance() *(declaration only)*
Ορίζεται ως απλή συνάρτηση ώστε το PhysicsEngine να μπορεί να εκτελέσει απευθείας δυναμική (runtime) κλήση της από τη δυναμική βιβλιοθήκη dll που περιέχει το φυσικό νόμο (μέσω της κλήσης GetProcAddress στα Windows ή αντίστοιχης για άλλα λειτουργικά συστήματα). Παρ' ότι «θεωρητικά» δεν αποτελεί τμήμα του PhysicsLaw, λειτουργεί ως και οφείλει να αντιμετωπίζεται ως μια στατική μέθοδος-εργοστάσιο της τάξης, για αυτό και αποτελεί αναπόσπαστο τμήμα του header file με το φυσικό νόμο. Η συνάρτηση αυτή επιτρέπει τη δυναμική φόρτωση των νόμων ασχέτως υλοποίησης. Η σύνταξη extern "C" ακολουθείται ώστε να μην είναι απαραίτητη η χρήση του C++ «διακοσμημένου» (*decorated*) ονόματός της στον κώδικα. Η κλήση αυτή πρέπει να επιστρέφει ένα νέο αντικείμενο PhysicsLaw του συγκεκριμένου φυσικού νόμου για τη χρήση του PhysicsEngine.

5.2.2. struct PhysicalPropertyDescription:

Δομή (struct) που «τυλίγει» τις απαραίτητες πληροφορίες για την αποθήκευση μιας φυσικής ιδιότητας.

Πεδία

void * *Data*:
Στο πεδίο αυτό περιέχεται ένας μοναδικός δείκτης στη δομή δεδομένων (συνήθως array ή buffer) που θα περιέχει τις τιμές μιας φυσικής ιδιότητας. Δεν υπάρχει κάποιος πρακτικός περιορισμός στο σε τι δεδομένα δείχνει το Data, αλλά σε γενικές γραμμές περιμένουμε να είναι δείκτης σε array of float (βαθμωτά φυσικά μεγέθη), array of vectors (διανυσματικά φυσικά μεγέθη), array of integers (indexes για άλλα μεγέθη), ή πιθανώς κάποιος δείκτης σε συνάρτηση (function pointer) για χρήση σε πεδία δυνάμεων.
Εδώ αξίζει να σημειωθεί ότι οι παραπάνω πληροφορίες του είδους δεδομένων που περιέχονται στο Data, διότι δεν υπάρχει τελικά νόημα σε αυτό: Όταν ο φυσικός νόμος ζητά με το όνομά της μια ιδιότητα, οφείλει να γνωρίζει ακριβώς τη μορφή στην οποία θα την παραλάβει. Διαφορετικά δεν θα υπήρχε ούτως η άλλως τρόπος να τη χρησιμοποιήσει.
unsigned int *size*:
Στο πεδίο αυτό περιέχεται ο αριθμός δεδομένων που περιέχει το Data. Προσοχή, το πεδίο δεν αναφέρεται στον αριθμό των μεταβλητών (floats/integers), αλλά στον αριθμό ιδιοτήτων που αυτά συνθέτουν. Δηλαδή, αν η δομή δεδομένων είναι τρισδιάστατο διάνυσμα από float, και έχουμε 6 float, το size θα να είναι 2, όχι 6.

5.2.3. class PhysicalPropertyProvider - (abstract):

Ορισμός διεπαφής (interface, που εκφράζεται στην C++ μέσω αφηρημένης τάξης χωρίς υλοποιημένα μέλη), που απαιτείται από το PhysicsLaw για να προμηθεύονται φυσικές ιδιότητες οι φυσικοί νόμοι. Αποτελεί το μόνο «αντίστροφο» κανάλι επικοινωνίας του PhysicsLaw με το PhysicsEngine, και είναι απαραίτητο ώστε το PhysicsLaw να γνωρίζει τον τρόπο με τον οποίο θα ζητά από το PhysicsEngine την προμήθεια φυσικών ιδιοτήτων.
PhysicalPropertyDescription *GetPhysicalProperty*(const char * *name*) *(override)*:
Υλοποίηση της διεπαφής (αφηρημένης τάξης) PhysicalPropertyProvider.
Τη μέθοδο αυτή καλούν οι φυσικοί νόμοι κατά την αρχικοποίησή τους για να λάβουν τις φυσικές ιδιότητες.
Η χρήση του PhysicalPropertyProvider είναι το σημείο επαφής του PhysicsLaw με το PhysicsEngine. Η διεπαφή PhysicalPropertyProvider επιτρέπει στο PhysicsLaw να χρησιμοποιήσει αντικείμενο οποιασδήποτε τάξης υλοποιεί την αφηρημένη τάξη/διεπαφή PhysicalPropertyProvider για να πάρει φυσικές ιδιότητες. Στη συνήθη περίπτωση το PhysicsEngine χρησιμοποιεί ένα BasePhysicalPropertyProvider που κατασκευάζεται από προεπιλογή.
void *AddPhysicalProperty*(const char * *name*, void* *data*, unsigned int *size*, unsigned int* *indexes*=0) – (abstract)
Η λειτουργία αυτή καλείται εξωτερικά ώστε να προστεθούν φυσικές ιδιότητες στο *PhysicalPropertyProvider*. Οι παράμετροι είναι

- *name* : C –string με το οποίο θα αναγνωρίζεται η σειρά. Χρησιμοποιήθηκε *const char* * (C-string) και όχι C++ *string* γιατί η κλήση αυτή θα περνά τα όρια της βιβλιοθήκης και απαιτείται μέγιστη συμβατότητα των ειδών δεδομένων.
- *data* : *Δείκτης* σε οτιδήποτε (void *). Δείχνει στα δεδομένα της συγκεκριμένης φυσικής ιδιότητας. Μπορεί να περιέχει οτιδήποτε, αλλά γενικώς αναμένεται περισσότερο να είναι κάτι από τα παρακάτω:
  - δείκτης σε μοναδικό *float* (κάποια παγκόσμια σταθερά)
  - float [ ] (κάποιο βαθμωτό φυσικό μέγεθος)
  - Vector4<float> (v4f)(κάποιο διανυσματικό φυσικό μέγεθος)
  - Δείκτης σε συνάρτηση v4f ->v4f ή v4f -> float (κάποιο δυναμικό πεδίο)
  - Δείκτης σε CL Buffer (αποθήκη δεδομένων κατά OpenCL)
- *size* :Ο αριθμός των φυσικών μεγεθών που περιέχονται στο *data*.
- *indexes* : Για φυσικά μεγέθη με *δείκτες*, όπως αναφέρθηκε νωρίτερα, εδώ θα περάσει ένα unsigned int [ ], που θα αποτελεί το ευρετήριο για το συγκεκριμένο φυσικό μέγεθος. Η παράμετρος αυτή είναι προαιρετική, και έχει προκαθορισμένη τιμή NULL (0), που υποδηλώνει ότι η φυσική ιδιότητα δεν είναι indexed.

5.2.4. class NewtonsLawOfMotion *(implements PhysicsLaw)*:

Παράδειγμα υλοποίησης φυσικού νόμου.
Υπάρχουν φυσικά τα κληρονομημένα πεδία του PhysicsLaw isDisposed και logger, και οι υλοποιημένες πλέον μέθοδοι Initialize και Calculate.
(Κώδικας για αυτά μπορεί να βρεθεί στο παράρτημα.)
Επιπροσθέτως, βλέπουμε τα εσωτερικά πεδία στα οποία ο ίδιος ο νόμος αποθηκεύει τις φυσικές ιδιότητες:
masses (float *), velocities (vector *), forces (vector *) και externalforces(vector *). Τα πεδία αυτά γεμίζονται κατά την κλήση της Initialize ως εξής:
Καλείται η Initialize με παράμετρο αναφορά στον τρέχοντα PhysicalPropertyProvider. Εκεί, ο φυσικός νόμος καλεί μέσω την getPhysicalProperty του PhysicalPropertyProvider με παράμετρο “mass”, οπότε του επιστρέφεται PhysicalPropertyDescription για τις μάζες. Κάνει «cast» το μέλος Data του PhysicalPropertyDescription σε float *, και το αποθηκεύει στο μέλος του masses. Ομοίως για όλα τα υπόλοιπα forces, positions, velocities, externalforces. Αν η διαδικασία αυτή γίνει με επιτυχία, θέτει isDisposed=false και επιστρέφει true. Σε αντίθετη περίπτωση, γράφει αντίστοιχο μήνυμα σφάλματος στο logger, και επιστρέφει false. Εφόσον δεν έχει τεθεί disposed = false, οφείλουμε (τουλάχιστον σε debug builds) να ελέγξουμε αν disposed=true στην αρχή κάθε κλήσης calculate.

5.2.5. class PhysicsEngine (implements PhysicalPropertyProvider):

Η τάξη αυτή αποτελεί την καρδιά του Vesper3D. Αποτελεί το σημείο εισόδου και επαφής των εξωτερικών προγραμμάτων, τον «τροχονόμο» των φυσικών νόμων, τον «ιδιοκτήτη» των (κατά τα άλλα public) αντικειμένων PhysicalPropertyProvider, οπότε και το σημείο αποθήκευσης των περιγραφών φυσικών ιδιοτήτων, και γενικότερα τον οδηγό, τον «τροχονόμο» ολόκληρου του υποσυστήματος φυσικής.

Πεδία

PhysicalPropertyProvider * physicalPropertyProvider

Δείκτης στο κεντρικό αντικείμενο physicalPropertyProvider που θα χρησιμοποιείται για την επικοινωνία με τους φυσικούς νόμους, και το οποίο θα χρησιμοποιούν και οι εξωτερικοί χρήστες του PhysicsEngine.
Κατά την υλοποίηση του αντικειμένου PhysicsEngine, ο δείκτης αυτός δείχνει στο στατικό BasePhysicalPropertyProvider.

BasePhysicalPropertyProvider basePhysicalPropertyProvider
Στατικός BasePhysicalPropertyProvider που δημιουργείται από προεπιλογή για ευκολότερη χρήση του PhysicsEngine. Προέρχεται από στατική ανάθεση μνήμης, οπότε ο χρήστης δεν χρειάζεται να ασχοληθεί μαζί του αν επιθυμεί να χρησιμοποιήσει διαφορετικό PhysicalPropertyProvider. Ο χρήστης του PhysicsEngine μπορεί να θέσει στον δείκτη physicalPropertyProvider άλλα αντικείμενα που επιθυμεί χωρίς να χρειάζεται να ασχοληθεί με τον – ανενεργό πλέον – basePhysicalPropertyProvider.

wostream * *loggerStream*:
Ρεύμα συμβόλων που γενικά παρέχεται από τον εξωτερικό χρήστη. Χρησιμοποιείται για τη γενική καταγραφή πληροφοριών κατάστασης για νόμους που φορτώθηκαν και άλλες πληροφορίες καταγραφής συμβάντων. Περνιέται επίσης στους φυσικούς νόμους για δική τους καταγραφή.

ResizeableArray<PhysicsLaw *> *physicsLaws*:
ResizeableArray είναι μια βοηθητική δομή δεδομένων που έχουμε φτιάξει, η οποία λειτουργεί όπως ένα κανονικό C++ *array,* με κάποιες επεκτάσεις (περιγράφονται παρακάτω).
Η συγκεκριμένη δομή αποθηκεύει *δείκτη* για κάθε φυσικό νόμο *PhysicsLaw* που φορτώνεται από το *PhysicsEngine* ώστε ο νόμος να εκτελείται σε κάθε καρέ.

float *dtmin*:
Υποστήριξη εκτέλεσης σταθερού ή μεταβλητού χρονικού βήματος.
Όπως περιγράφηκε, το PhysicsEngine εκτελείται με βάση το χρόνο σε κάθε καρέ.
Όταν το dtmin είναι 0, η εκτέλεση των νόμων εκτελείται μία φορά ανά κλήση της Advance(dt), με παράμετρο dt.
Όταν το dtmin είναι μεγαλύτερο του μηδενός, η Advance εκτελεί τον κάθε φυσικό νόμο με παράμετρο dtmin, για τόσα χρονικά διαστήματα dtmin όσα απαιτούνται για να υπερκαλύψουν το dt. Η διαφορά που δημιουργείται διατηρείται και προστίθεται στο επόμενο καρέ, ώστε να μην προκύπτουν συσσωρευόμενα σφάλματα.

<u>Λειτουργίες</u>
PhysicsLawFactoryFunction *getPhysicsLawFactory*(const wchar_t * law):
Η μέθοδος αυτή είναι εσωτερική (private), και αφορά στο μηχανισμό φορτώματος των φυσικών νόμων.
Η παράμετρος law είναι το όνομα της βιβλιοθήκης dll του νόμου σε Unicode συμβολοσειρά.
Η συνάρτηση αυτή θα καλέσει τη ενσωματωμένη συνάρτηση των Windows LoadLibrary, και θα φορτώσει τη βιβλιοθήκη με όνομα αρχείου **law.dll**. Κατόπιν, πάνω στη βιβλιοθήκη αυτή θα εκτελέσει τη συνάρτηση GetProcAddress με παράμετρο

«GetPhysicsLawInstance» ώστε να λάβει δείκτη στη συνάρτηση – εργοστάσιο του φυσικού νόμου αυτού και να τον επιστρέψει. Σε περίπτωση αποτυχίας επιστρέφει 0(NULL).

void *LoadPhysicsLaw*(const wchar_t * *law*):
Η λειτουργία αυτή καλείται εξωτερικά ώστε να φορτωθεί φυσικός νόμος από dll. Η συνάρτηση αυτή καλεί την *getPhysicsLawFactory* για να λάβει δείκτη στη συνάρτηση-εργοστάσιο του φυσικού νόμου, την οποία και εκτελεί για να υλοποιήσει το αντικείμενο του νόμου.
Εδώ μπορούμε να παρατηρήσουμε τη υλοποίηση του εξαιρετικά χρήσιμου προτύπου σχεδίασης (design pattern) **Abstract Factory**. Το πρότυπο αυτό εδώ μας επιτρέπει να υλοποιούμε και να χρησιμοποιούμε αντικείμενα φυσικών νόμων χρησιμοποιώντας μια αφηρημένη συνάρτηση – εργοστάσιο χωρίς να γνωρίζουμε για το αντικείμενο τίποτε εκτός από τη βασική διεπαφή.
Η παράμετρος law είναι το όνομα του αρχείου της βιβλιοθήκης του νόμου.
Η εκτέλεση του *PhysicsLawFactoryFunction* που μας επέστρεψε η getPhysicsLawFactory μας επιστρέφει δείκτη στο τελικό σχηματισμένο αντικείμενο του φυσικού νόμου, το οποίο και προσθέτουμε στη δομή *physicsLaws*.

void *Log*(…):
Βοηθητική συνάρτηση που καταγράφει πληροφορίες του προγράμματος.

void *Initialize*() - *(override)*:
Η συνάρτηση αυτή καλεί διαδοχικά τις αντίστοιχες συναρτήσεις *Initialize*(…) όλων των φυσικών νόμων που βρίσκονται στο *physicsLaws*, και καταγράφει λεπτομερώς τα αποτελέσματα της διαδικασίας.
void *Advance*(float dt) - *(override)*:
Η συνάρτηση αυτή είναι ο λόγος για τον οποίο υπάρχει όλη αυτή η σχεδίαση: καλείται εξωτερικά σε κάθε καρέ για να επεξεργαστεί τη φυσική του συστήματος. Αναλόγως της τιμής του *dtmin* καλεί διαδοχικά, μία ή περισσότερες φορές, τη συνάρτηση Calculate των φυσικών νόμων. Εδώ παρατηρούμε ένα άλλο σημαντικό πρότυπο σχεδίασης, το **Delegation Pattern** (Πρότυπο αντιπροσώπου). Το PhysicsEngine μεταθέτει τη δουλειά των φυσικών υπολογισμών που του ζητείται, στα PhysicsLaws.

5.2.6. class BasePhysicalPropertyProvider – (implements PhysicalPropertyProvider)

Η τάξη αυτή έχει ως μοναδικό σκοπό την υλοποίηση του PhysicalPropertyProvider. Είναι πιθανώς ο απλούστερος δυνατός PhysicalPropertyProvider.
Κάθε νέο PhysicsEngine δημιουργεί ένα τέτοιο αντικείμενο, και αποθηκεύει τη θέση του στον δείκτη physicalPropertyProvider. Ασχέτως αν τελικά χρησιμοποιηθεί κάποιος άλλος physicalPropertyProvider, η θέση μνήμης του BaseAttributeProvider ελευθερώνεται στον καταστροφέα της.

Πεδία
std::map<std::string, PhysicalPropertyDescription> *physicalPropertyDictionary*:

Η δομή αυτή αποτελεί την αποθηκευτική καρδιά του αντικειμένου *PhysicalPropertyProvider*, που με τη σειρά του είναι η αποθηκευτική καρδιά του PhysicsEngine. Είναι ένας *χάρτης*, δηλαδή μια δομή δεδομένων που αποθηκεύει ζεύγη κλειδιών/τιμών. Εδώ τα κλειδιά είναι κλασσικά C++ std::*string*, και οι τιμές είναι PhysicalPropertyDescription. Με τον τρόπο αυτό έχουμε μια απευθείας αντιστοίχιση της κάθε φυσικής ιδιότητας με ένα όνομα. Έτσι, ο κάθε νόμος μπορεί να επιλέξει τις φυσικές ιδιότητες που χρειάζεται μέσω των αντίστοιχων λειτουργιών ανάθεσης/ανάκτησης.

Λειτουργίες

void *AddPhysicalProperty*(const char * *name*, void* *data*, unsigned int *size*, unsigned int* *indexes*=0) – *(override)*

Η λειτουργία αυτή υλοποιεί την αντίστοιχη του PhysicalPropertyProvider. Η μέθοδος αυτή δημιουργεί αντικείμενο PhysicalPropertyDescription και αποθηκεύει σε αυτό το *data* και το *size*. Το αντικείμενο αυτό το εισάγει στο χάρτη physicalPropertyDictionary με όνομα *name*.
Κατόπιν ελέγχει αν η παράμετρος indexes υπάρχει. Αν υπάρχει, δημιουργεί δεύτερο αντικείμενο PhysicalPropertyDescription, εισάγει το indexes και το *size*, και εισάγει το νέο αντικείμενο στο χάρτη με όνομα *name*_Indexes.

PhysicalPropertyDescription *GetPhysicalProperty*(const char * *name*) *(override)*:
Υλοποίηση της διεπαφής (αφηρημένης τάξης) PhysicalPropertyProvider.
Ανακτά το PhysicalPropertyDescription με όνομα name από το χάρτη, και το επιστρέφει. Αν δεν βρεθεί, επιστρέφει 0(NULL).

5.3. Βοηθητικές τάξεις και αρχεία

Στο κεφάλαιο αυτό περιγράφονται συνοπτικά άλλα αρχεία και δομές δεδομένων που αναπτύχθηκαν ανεξάρτητα και χρησιμοποιούνται από το Vesper3D χωρίς να αποτελούν τμήμα του. Απαιτούνται για τη μεταγλώττισή του αλλά όχι για τη χρήση του.

5.3.1. ResizeableArray <T>

Η δομή αυτή χρησιμοποιείται συνεχώς στην διατριβή αυτή.
Χρησιμοποιεί την τεχνική των C++ Templates για να υλοποιήσει μια αποθηκευτική δομή δεδομένων πολλών χρήσεων, που αποθηκεύει κάθε απλό τύπο δεδομένων (POD – Plain Old Data). Ουσιαστικά, τη χρησιμοποιούμε σε κάθε περίπτωση που θα χρησιμοποιούσαμε ένα απλό array και δυναμική ανάθεση μνήμης (το ResizeableArray τοποθετείται πάντοτε στο **σωρό** (*heap*) και δεν εκτελεί ποτέ στατική ανάθεση στη **στοίβα** (*stack*)). Το ResizeableArray έχει τις παρακάτω ιδιαιτερότητες:

- Εκτελεί λειτουργίες **αυτόματης ανάθεσης και απελευθέρωσης μνήμης**, για την αύξηση και σμίκρυνση μεγέθους , είτε αυτόματα είτε με συγκεκριμένες εντολές. Συνεπώς, παίρνει από τον κώδικά μας την ευθύνη ανάθεσης νέας μνήμης και μεταφοράς δεδομένων όταν αυτό απαιτείται, μέσω των κλήσεων resize(…),reserve(). Αντίστοιχα, με κλήση στο trim() μπορεί να φέρει το μέγεθος του array ακριβώς στον αριθμό των στοιχείων του πίνακα. Με τον τρόπο αυτό λειτουργεί ως std::vector της STL (Standard Template Library).
- Περιλαμβάνει εντολές λειτουργίας ***Λίστας****(List)* :insert(…), remove(…), first(), last(). Με τις εντολές αυτές αυτοματοποιεί την εισαγωγή και αφαίρεση

αντικειμένων από το array δίνοντας την *ψευδαίσθηση* διασυνδεδεμένης λίστας. Με τον τρόπο αυτό λειτουργεί σαν **ArrayList** της Java ή **List** της C#.

- Περιλαμβάνει τις εντολές **Στοίβας**(*stack*): push(…), pop().
- Διατηρεί **βοηθητικά μέλη** για την διευκόλυνση του προγραμματιστή (size: αριθμός αντικειμένων, capacity: δεσμευμένη μνήμη), οπότε επιτρέπει ανά πάσα στιγμή να γνωρίζουμε πληροφορίες όπως ο αριθμός των αντικειμένων που περιέχει για την εύκολη προσπέλασή της.
- Διαθέτει **iterators** (ιδίωμα C++) για την υποστήριξη της εντολής for_each.
- Η μνήμη του είναι πάντοτε ευθυγραμμισμένη στα 16 bytes. Με τον τρόπο αυτό υποστηρίζει άριστα τις δομές __m128 (που είναι αυτές που φορτώνονται στους καταχωρητές 128bit των επεξεργαστών SSE2), και επίσης, σε γενικές γραμμές, βοηθά την απόδοση με κόστος κάποια κενά μνήμης.
- *Προσοχή θέλει το γεγονός ότι δεν πρέπει να διατηρούμε απευθείας δείκτες σε περιεχόμενά της ανάμεσα σε πιθανά resize(), διότι η μνήμη θα έχει ενδεχομένως μετακινηθεί.*
- Η ανάκτηση δεδομένων είναι σχεδόν όσο ταχεία θα ήταν σε ένα τυπικό array, διότι ως βιβλιοθήκη περιέχεται μόνο σε αρχείο header, οπότε ο κώδικάς της εξ' ορισμού **εκτελείται σε σειρά** με τον καλούν κώδικα (*inline*).

5.3.2. Vector4<T>

Η δομή αυτή έχει υλοποιηθεί ώστε να πραγματοποιήσουμε ένα βασικό Πρότυπο (βλέπε επόμενο κεφάλαιο) για την ανάπτυξη φυσικών νόμων. Στο πρότυπο αυτό όλα τα βαθμωτά μεγέθη αποτελούνται από *float (κατά IEEE 7534-2008)*, και όλες οι συνιστώσες τρισδιάστατων διανυσματικών μεγεθών είναι επίσης διανύσματα float.

Η δομή Vector4<T> και οι συνοδευτικές της δομές (Vector3<T>, Matrix4<T>, Quaternion<T>) αποτελούν βασικές δομές διανυσματικής άλγεβρας (Διάνυσμα, Μήτρα) και το Τετράνιο που είναι δομή πολύτιμη στα τρισδιάστατα γραφικά.
Οι δομές αυτές δεν έχει νόημα να περιγραφούν πλήρως εδώ, αλλά να αναφερθεί ότι αποτελούν την συνηθέστερα χρησιμοποιούμενη δομή δεδομένων από την παρούσα διατριβή για την έκφραση διανυσματικών μεγεθών. Τα βασικά σημεία είναι:

- Η δομή επιτρέπει τη χρήση όλων των βασικών αριθμητικών τύπων (integer, float, double, long, long long, short, char), και γενικά τύπους που υποστηρίζουν τις βασικές αριθμητικές πράξεις.
- Η γενική υλοποίηση είναι απλή, αλγοριθμική και χωρίς ιδιαίτερες βελτιστοποιήσεις.
- **Παρέχεται ειδίκευση προτύπου (template specialization) ειδικά για το Vector4<float>** (διάνυσμα τεσσάρων αριθμών κινητής υποδιαστολής), για το οποίο έχει εκπονηθεί μεγάλης κλίμακας προσπάθεια. Η ειδίκευση αυτή **χρησιμοποιεί τους 128bit SSE καταχωρητές (τύπος __m128 στη Visual C++) για την αποθήκευση διανυσμάτων 4 συνιστωσών, σε ομογενείς συντεταγμένες. Οι υλοποιήσεις των συναρτήσεων χρησιμοποιούν τις βελτιστοποιημένες συναρτήσεις διανυσματικών πράξεων (compiler intrinsics) του μεταγλωττιστή MS Visual C++ για πράξεις ανάμεσα σε διανύσματα __m128.** Η δομή αυτή έχει ψευδώνυμο (alias/typedef τη λέξη **v4f**)

- Με τον τρόπο αυτό και εκτεταμένη χρήση υπερφόρτωσης πράξεων (operator overloading), έχουν υλοποιηθεί όλες οι κλασσικές πράξεις διανυσμάτων σε ομογενείς συντεταγμένες 4 διαστάσεων : Πρόσθεση, αφαίρεση, πολλαπλασιασμός κατά μέλη, διαίρεση κατά μέλη, εσωτερικό γινόμενο, εξωτερικό γινόμενο, μήκος, απόσταση, κανονικοποίηση, ομογενοποίηση κ.α.
- **Έτσι, εάν χρησιμοποιηθεί η βιβλιοθήκη αυτή δίνεται στον προγραμματιστή ένα εύκολο και γρήγορο εργαλείο για την υλοποίηση φυσικών νόμων με τη χρήση διανυσματοποίησης που αναφέρθηκε παραπάνω**

Παρατήρηση:
Οι ομογενείς συντεταγμένες που περιγράφονται συνίστανται στην έκφραση ενός τρισδιάστατου διανύσματος (x,y,z) με τέσσερις συντεταγμένες (x,y,z,w) με μοναδικό περιορισμό ότι δεν είναι και οι τέσσερις συντεταγμένες 0.
Το κάθε διάνυσμα x,y,z,w αντιπροσωπεύει το σημείο (x/w,y/w,z/w) του ευκλείδειου χώρου. Όταν w=0, το x,y,z,w αντιπροσωπεύει το αντίστοιχο σημείο στο άπειρο.
Η απεικόνιση αυτή είναι η απεικόνιση που χρησιμοποιείται από όλες τις σύγχρονες κάρτες γραφικών για τα πρότυπα DirectX και OpenGL ώστε να αντιμετωπίζονται όλες οι πράξεις με διανύσματα και μήτρες με τον ίδιο τρόπο χωρίς να απαιτείται ιδιαίτερος χειρισμός για τα σημεία στο άπειρο, και επιτρέπει να εκφρασθούν όλοι οι απαραίτητοι μετασχηματισμοί διανυσμάτων (αλλαγή κλίμακας, μετατόπιση, περιστροφή, προβολή κ.λπ.) ως μήτρες 4x4.

6. Χρήση του Vesper3D

6.1. Προϋποθέσεις χρήσης

Από όλα τα παραπάνω έχει προκύψει η βασική προϋπόθεση χρήσης του Vesper3D:
Η βασική προϋπόθεση χρήσης του Vesper3D συνίσταται στην ύπαρξη δυνατότητας κλήσης στη μηχανή σε κάθε καρέ με παράμετρο το χρονικό διάστημα που διήλθε από το προηγούμενο καρέ, σε μονάδες συμβατές με τους φυσικούς νόμους που χρησιμοποιήθηκαν.
Η άλλη προϋπόθεση χρήσης είναι να μπορούν οι φυσικές ιδιότητες να φτάσουν σε μορφή κατανοητή από τη μηχανή.

6.2. Βασικοί χρήστες και απαιτούμενη ανάπτυξη λογισμικού

Η μηχανή Vesper3D θεωρεί ως δεδομένη την αλληλεπίδραση των παρακάτω ατόμων και προγραμμάτων για την τελική χρήση:

6.2.1. Εφαρμογή Πελάτης

Το εικονικό περιβάλλον ή η προσομοίωση που θα εκμεταλλευτεί το Vesper3D για τον υπολογισμό των αποτελεσμάτων φυσικής. Θεωρείται ότι σε μια «ορθόδοξη» υλοποίηση, πέρα από μια βασική διεπαφή ανάμεσα στον πελάτη και το Vesper3D, δεν θα πρέπει να απαιτείται ανάπτυξη κώδικα στον ίδιο τον πελάτη, ακόμα και αν απαιτείται προσθήκη ή η αφαίρεση φυσικών νόμων και ιδιοτήτων. Θα πρέπει η μηχανή να είναι σε θέση να *«καταναλώσει»* οποιοδήποτε αριθμό και είδος φυσικών ιδιοτήτων, ακόμα και όταν δεν έχει φυσικούς νόμους να τις χρησιμοποιήσουν.
Το πρόγραμμα Πελάτης γενικά αναμένεται ότι:

- Θα ορίζει τις φυσικές ιδιότητες με τρόπο ανεξάρτητο από το Vesper3D
- Θα αναγιγνώσκει τις φυσικές ιδιότητες οι οποίες επηρεάζουν την συμπεριφορά του (επί παραδείγματι τις θέσεις σωμάτων όπως αυτές έχουν τροποποιηθεί από την επίδραση των φυσικών νόμων που διαχειρίζεται το Vesper3D).
- Θα εγγράφει και τροποποιεί, κατά τρόπο ανεξάρτητο από το Vesper3D, τις φυσικές ιδιότητες τις οποίες επιθυμεί (π.χ. επίδραση εξωτερικών δυνάμεων κατά τη «λογική» της προσομοίωσης, ή αυθαίρετη «τηλεμεταφορά» σωμάτων)

Παράδειγμα: ReveWorlds/jReve. Θα ορίζονται φυσικές ιδιότητες χωρίς κάποια ιδιαίτερη μνεία στη χρήση τους από το υποσύστημα Vesper3D.

6.2.2. Διεπαφή μεταξύ Πελάτη και Vesper3D

(συνήθως PhysicalPropertyProvider)
Το τμήμα αυτό κώδικα θα πρέπει να αναπτυχθεί κατά περίπτωση, σχεδόν για κάθε Πελάτη. Είναι το τμήμα του κώδικα για το οποίο θα απαιτηθεί προγραμματιστική εργασία ώστε να χρησιμοποιηθεί το Vesper3D από κάποιον πελάτη.
Πάντοτε θα περιέχει τμήμα της υλοποίησης στη γλώσσα συγγραφής του Πελάτη, και τμήμα της υλοποίησης σε C++ ή γλώσσα διεπαφής ανάμεσα στη γλώσσα του πελάτη και C++ (JNI κ.λπ.).
Το τμήμα αυτό έχει την ευθύνη της μεταφοράς ή αντιγραφής ή γενικά της επικοινωνίας ανάμεσα στις φυσικές ιδιότητες – όπως αυτές είναι ορισμένες στον πελάτη – σε μορφή κατανοητή από τους φυσικούς νόμους. Ουσιαστικά υλοποιεί ένα ***Πρότυπο*** (όπως αυτό

θα επεξηγηθεί στο 6.2)

Η Διεπαφή αναμένεται ότι:

- Θα δημιουργεί ή/και αρχικοποιεί ή/και καταστρέφει την **έκφανση** (instance) του Vesper3D σε χρόνο κατάλληλο για τον Πελάτη
- Θα ορίζει ή/και μετασχηματίζει ή/και αντιγράφει ή/και αναθέτει μνήμη για τις φυσικές ιδιότητες, κατά τις απαιτήσεις του Πελάτη και του Προτύπου για τα οποία συνετάχθη, ώστε να επιτραπεί η χρήση τους από το Vesper3D
- Θα ζητά τη φόρτωση φυσικών νόμων από το Vesper3D σε χρόνο κατάλληλο για τον Πελάτη
- Πιθανώς να εκτελεί τη δημιουργία ή/και καταστροφή των σωμάτων

Όταν υλοποιείται σε C++, συνήθως το επίπεδο αυτό της διεπαφής ή τουλάχιστον τμήμα του θα υλοποιηθεί μέσω της *διεπαφής* **PhysicalPropertyProvider**, δηλαδή θα είναι το υπεύθυνο τμήμα για τη χρήση των φυσικών ιδιοτήτων.
Σε περιπτώσεις χρήσης γλωσσών υψηλότερου επιπέδου, μπορεί να χρησιμοποιείται ως βάση η default υλοποίηση **BasePhysicalPropertyProvider** που πάντοτε περιέχεται στην κεντρική μηχανή, και η διαχείριση θα μεταφέρεται στη γλώσσα υψηλού επιπέδου ώστε να ικανοποιηθούν οι συγκεκριμένες ιδιαιτερότητες και απαιτήσεις της στη χρήση μνήμης (Garbage Collection, απαιτήσεις μνήμης κ.λπ.).
Εδώ γίνεται φανερό ότι ο όγκος της ανάπτυξης που θα απαιτηθεί για να χρησιμοποιηθεί το υποσύστημα Vesper3D βρίσκεται στη διεπαφή αυτή.

6.2.3. Κεντρική μηχανή του Vesper3D

Το τμήμα PhysicsEngine της μηχανής Vesper3D. Σε καμία περίπτωση δεν πρέπει να απαιτηθεί απευθείας ανάπτυξη λογισμικού πάνω στο PhysicsEngine για να επιτραπεί η απλή χρήση της μηχανής από συγκεκριμένο πελάτη. Κάτι τέτοιο υπονοεί αποτυχία της σχεδίασης.
Η μόνη περίπτωση που η ζητούμενη ανάπτυξη θεωρείται δόκιμη, είναι όταν θα έδινε μια δυνατότητα που όλοι οι πιθανοί πελάτες της μηχανής θα μπορούσαν εν δυνάμει να χρησιμοποιήσουν (Παράδειγμα: θα μπορούσε να υλοποιηθεί δυνατότητα οι διάφοροι φυσικοί νόμοι να χρησιμοποιούν διαφορετικά μεταξύ τους “dt” όταν χρησιμοποιείται σταθερό βήμα χρόνου. Στην περίπτωση αυτή αναφερόμαστε όμως σε επέκταση ή προσθήκη δυνατότητας στη μηχανή,(feature) και όχι σε εξειδίκευσή της απλά για να επιτραπεί η βασική της χρήση.

6.2.4. Φυσικοί Νόμοι

Γενικά, οι φυσικοί νόμοι θεωρείται ότι θα αναπτύσσονται κατά περίπτωση.
Όμως, μπορούν να συνταχθούν Πρότυπα τα οποία θα «ομαδοποιούν» τους φυσικούς νόμους σε ομοειδή σύνολα. Κατ’ αυτόν τον τρόπο, μπορούν «ομάδες» φυσικών νόμων να υλοποιηθούν και να επαναχρησιμοποιηθούν από διαφορετικούς πελάτες, αρκεί η Διεπαφή να ακολουθήσει το Πρότυπό τους.

6.3. Πρότυπα φυσικών νόμων

Πρότυπο εδώ θα ονομάζουμε κάποιους κανόνες σύνταξης φυσικών νόμων και ιδιοτήτων, ήτοι:

- Δομή και είδος δεδομένων για Βαθμωτά φυσικά μεγέθη (π.χ. float [])
- Δομή και είδος δεδομένων για Διανυσματικά φυσικά μεγέθη.
- Δομή / είδος / τρόπος αλληλεπίδρασης / περιγραφή των συναρτήσεων που εκφράζουν Δυναμικά Πεδία. Στη συνήθη περίπτωση τα δυναμικά πεδία οφείλουν να εκφράζονται ως συναρτήσεις (Callbacks) που λαμβάνουν ως παράμετρο Διάνυσμα (ως έχει ορισθεί ανωτέρω) και επιστρέφουν επίσης Διάνυσμα.
- Ιδιαιτερότητες διάταξης

6.3.1. Βασικό πρότυπο (float/4Dhomogenous),

Το βασικό Πρότυπο που αναπτύχθηκε, υλοποιήθηκε και χρησιμοποιείται από όλα τα case studies της παρούσας εργασίας το ονομάζουμε πρότυπο Vesper3D/float/4D, και θεωρούμε ότι σε πραγματικές συνθήκες θα κάλυπτε το 90% των απαιτήσεων 3D εφαρμογών.
Για την υλοποίησή του έχει σχεδιαστεί τάξη που περιγράφει διανυσματικά μεγέθη (Vector4<float> ή αλλιώς v4f). Η τάξη αυτή περιγράφεται παρακάτω.
Περιγράφεται ως:

- Βαθμωτά φυσικά μεγέθη: **float []** (Αριθμός κινητής υποδιαστολής IEEE754 (C/ C#/Java float) σε πίνακα)
- Διανυσματικά φυσικά μεγέθη: **v4f []** με σειρά διάταξης στη μνήμη x, y, z, w. (Τετραδιάστατα διανύσματα από float σε Ομογενείς συντεταγμένες, τα οποία μπορούν να αποθηκεύονται ως Vector4<float>[] v4f ή float[])
- Δυναμικά πεδία: Συναρτήσεις ή Στατικές Μέθοδοι τάξεων, που δέχονται ως παράμετρο το παραπάνω ορισμένο διάνυσμα και επιστρέφουν το παραπάνω ορισμένο διάνυσμα (v4f<float> *name*(v4f<float>)
- Χρήση δεικτών (indexes) ανά περίπτωση.

Το Πρότυπο αυτό υλοποιήθηκε ώστε να μπορεί να διευκολύνει κάποιες λειτουργίες των σύγχρονων υπολογιστών και API's τα οποία αναμένουμε να χρησιμοποιηθούν (OpenGL, DirectX, OpenCL, Cuda)

- Οι αριθμοί κινητής υποδιαστολής διευκολύνουν ιδιαίτερα στην αναπαράσταση φυσικών μεγεθών λόγω της δυνατότητάς τους να παραστήσουν μεγέθη με αρκετά διαφορετικές κλίμακες (τάξη μεγέθους από $10^{-38} – 10^{-38}$)
- Όπως αναφέρθηκε και νωρίτερα, στην τρέχουσα γενιά των επεξεργαστών Intel και Amd για προσωπικούς υπολογιστές, αναπτύσσονται συνεχώς τεχνολογίες για την επιτάχυνση των υπολογισμών πράξεων κινητής υποδιαστολής. Έτσι, πέρα από τη γενική επιτάχυνση των πράξεων ανάμεσα σε αριθμούς κινητής υποδιαστολής, τα πρότυπα SSE 2 και εξής που υποστηρίζονται από όλους τους σχετικά νέους Intel και Amd επεξεργαστές (Pentium 4 και εξής και Amd Athlon XP και εξής), παρέχουν απευθείας υποστήριξη για παράλληλες διανυσματικές πράξεις ανάμεσα σε τετραδιάστατα διανύσματα από float. Οπότε, μπορούν να συνταχθούν ή να ληφθούν βιβλιοθήκες για πράξεις ανάμεσα σε τέτοια διανύσματα με πολύ υψηλή απόδοση και ποιότητα (έχει συνταχθεί τέτοια βιβλιοθήκη στην παρούσα εργασία. Άλλα παραδείγματα είναι η aMath vector library, Microsoft Xna math κ.α.)
- Οι σύγχρονες κάρτες γραφικών χρησιμοποιούν εσωτερικά σχεδόν αποκλειστικά τετραδιάστατα ομογενή διανύσματα από float για την εσωτερική απεικόνιση

θέσεων σωμάτων στον τρισδιάστατο χώρο. Το γεγονός αυτό επιτρέπει την άμεση χρήση των δομών των θέσεων σωμάτων του Προτύπου αυτού από APIs γραφικών όπως το OpenGL και το DirectX σε πολλές περιπτώσεις χωρίς απολύτως κανένα μετασχηματισμό. Η σημασία της δυνατότητας μεταφοράς αυτούσιων δομών δεδομένων στην κάρτα γραφικών για απεικόνιση δεν πρέπει να υποεκτιμηθεί διότι μπορεί να μας σώσει από μεταφορά πολύ μεγάλης ποσότητας δεδομένων.

- Εύκολη χρήση από τους περισσότερους καλούντες. Με εξαίρεση την ιδιαιτερότητα που αναπόφευκτα θα προκαλέσει η κοινή χρήση μνήμης ανάμεσα στο Vesper3D και σε Γλώσσες με Συλλέκτη Απορριμμάτων (Garbage Collected), πρακτικά όλοι οι καλούντες μπορούν εύκολα από κώδικα να χειριστούν απλές δομές δεδομένων που αποτελούνται από float.

6.3.2. Πλεονεκτήματα από τη χρήση Προτύπου

Στη γενική περίπτωση δεν «απαιτείται» η σύνταξη ενός προτύπου. Μπορούν να συνταχθούν φυσικοί νόμοι που να λαμβάνουν αυθαίρετες δομές δεδομένων, και οι δομές αυτές να εξηγηθούν ώστε να παρασχεθούν ως έχουν από τη Διεπαφή και τον Πελάτη.
Αν όμως συνταχθεί ένα Πρότυπο, γίνεται κατόπιν φανερή η ισχύς της επεκτασιμότητας του υποσυστήματος Vesper3D.
Μπορούν να συνταχθούν φυσικοί νόμοι οι οποίοι να ακολουθούν το προσυμφωνηθέν Πρότυπο, και μπορούν ανεξαρτήτως να ορισθούν από τον Πελάτη οι φυσικές ιδιότητες που θα χρησιμοποιηθούν από τέτοιους Φυσικούς Νόμους, αρκεί φυσικά ο Πελάτης και ο συντάκτης των Φυσικών Νόμων να ακολουθούν «κοινή λογική» και γνώση των κανόνων Φυσικής πάνω στα μεγέθη που χρησιμοποιούν. Το μοναδικό σημείο όπου πρέπει κάποια ιδιαιτερότητα να γεφυρωθεί, είναι η ονομασία των ιδιοτήτων – αυτό όμως μπορούμε πλέον να το θεωρήσουμε τετριμμένο.

6.3.3. Παράδειγμα χρήσης προτύπου

Ο Πελάτης:
Ορίζει για τα σώματά του φυσικές ιδιότητες:
*Μάζα, Θέση, Ταχύτητα, Συνισταμένη Δύναμη, Εξωτερικές Δυνάμεις, Φορτίο, Όγκος, Μαγνητική Επαγωγή, Σημείο Βρασμού, Ειδική Συχνότητα, Σκληρότητα (*ελατηρίου*), Απόσβεση (*ελατηρίου*), Άκρα(*ελατηρίου*).*
Ορίζει συναρτήσεις (callbacks) για τρία δυναμικά πεδία:
Βαρυτικό, το *Μαγνητικό* και το *Ηλεκτροστατικό*.
Ορίζει σταθερές:
Σταθερά Παγκόσμιας Έλξης, Σταθερά Coulomb, Σταθερά Planck.

Ανεξάρτητα, **ο συντάκτης των Φυσικών Νόμων** ορίζει τους παρακάτω φυσικούς νόμους:
Νόμος Παγκόσμιας Έλξης Νεύτωνα:
Χρησιμοποιεί τις ιδιότητες *Θέση*, *Μάζα* όσων σωμάτων τις διαθέτουν και τη *Σταθερά Παγκόσμιας Έλξης* για να επηρεάσει την *Συνισταμένη Δύναμη* τους.
Ηλεκτροστατική Έλξη Coulomb:

Χρησιμοποιεί τις ιδιότητες *Θέση*, *Φορτίο* όσων σωμάτων τις διαθέτουν και τη *Σταθερά Coulomb* για να επηρεάσει την *Συνισταμένη Δύναμη* τους.
Ηλεκτρομαγνητικό Πεδίο Maxwell:
Χρησιμοποιεί τις ιδιότητες *Θέση*, *Ταχύτητα, Φορτίο* όσων σωμάτων τις διαθέτουν, το πεδίο *Ηλεκτρομαγνητικό Πεδίο* και τη *Σταθερά Coulomb* για να επηρεάσει την *Συνισταμένη Δύναμη* τους.
Δυνάμεις Ελατηρίων Hooke με Ιξώδη Απόσβεση:
Χρησιμοποιεί τις ιδιότητες *Μήκος(ελατηρίου),Σκληρότητα(ελατηρίου), Απόσβεση(ελατηρίου), Άκρα(ελατηρίου),* και τις ιδιότητες Θέση, Ταχύτητα όσων σωμάτων αναφέρονται στις ιδιότητες *Άκρα(ελατηρίου)*, ώστε να επηρεάσει την *Συνισταμένη Δύναμη* τους.
Αύξηση Μάζας Αδρανείας της Ειδικής θεωρία της Σχετικότητας
Χρησιμοποιεί τις *ιδιότητες Θέση, Ταχύτητα, Μάζα Ηρεμίας* και τη σταθερά *Ταχύτητα του Φωτός στο Κενό* για να επηρεάσει τη *Μάζα* τους.
Νευτώνεια κίνηση:
Χρησιμοποιεί τις *ιδιότητες Θέση, Ταχύτητα, Συνισταμένη Δύναμη, Εξωτερικές Δυνάμεις* για να υπολογίσει την επόμενη θέση του σώματος.

Παρατηρούμε από τις παραπάνω λίστες τα εξής:
Οι νόμοι Νόμος Παγκόσμιας Έλξης Νεύτωνα, Ηλεκτροστατική Έλξη Coulomb, Ηλεκτρομαγνητικό Πεδίο Maxwell, Νευτώνεια κίνηση θα εφαρμοστούν κανονικά χωρίς κάποια ιδιαιτερότητα, και θα έχουν τα αναμενόμενα αποτελέσματα.

Όμως, παρατηρούμε ότι:
Για τις Δυνάμεις Ελατηρίων Hooke με Ιξώδη Απόσβεση θα χρησιμοποιούσε τη φυσική ιδιότητα Μήκος (ελατηρίου) η οποία δεν έχει ορισθεί, και η Αύξηση Μάζας Αδρανείας της Ειδικής θεωρία της Σχετικότητας χρησιμοποιεί την ιδιότητα Μάζα Ηρεμίας και τη σταθερά Ταχύτητα του Φωτός στο Κενό που επίσης δεν έχουν ορισθεί.
Οι νόμοι αυτοί θα καταγράψουν (log) σφάλμα κατά την αρχικοποίησή τους, και θα αγνοηθούν κατά την εκτέλεση.
Αντίστροφα, οι φυσικές ιδιότητες *Όγκος, Μαγνητική Επαγωγή, Σημείο Βρασμού, Ειδική Συχνότητα,* τα πεδία *Βαρυτικό και Ηλεκτροστατικό* και η *σταθερά Planck*, δεν χρησιμοποιούνται από κάποιο φυσικό νόμο και θα αγνοηθούν.
Εδώ όμως βλέπουμε την αξία της συγκεκριμένης αρχιτεκτονικής:
Εάν προστεθεί (από τον τελικό χρήστη!) η ιδιότητα Μήκος(ελατηρίου), αμέσως θα ξεκινήσει να λειτουργεί κανονικά ο νόμος Δύναμη Ελατηρίου με Ιξώδη Απόσβεση χωρίς καμία ενέργεια από πλευράς του συντάκτη Φυσικών Νόμων, και χωρίς ανάπτυξη λογισμικού.
Αντίστροφα, όταν θα συγγραφεί ο νόμος Δύναμη Βαρυτικού Πεδίου (χρησιμοποιεί Μάζα, Θέση, Βαρυτικό Πεδίο, Συνισταμένη Δύναμη), θα ξεκινήσουν κανονικά όλα τα σώματα με τις αντίστοιχες ιδιότητες να υπακούουν στο νόμο αυτό χωρίς καμία ενέργεια από πλευράς του Πελάτη.
Παρατηρούμε λοιπόν ότι τα Πρότυπα επιτρέπουν την σχεδόν πλήρη (εκτός δηλαδή από την ονοματολογία) αποσύνδεση του Πελάτη από το Φυσικό Νόμο, και την ελεύθερη επέκταση του συστήματος ανεξάρτητα στις δυο πλευρές.

7. Οδηγίες χρήσης του Vesper 3D (How-to)

7.1. Περιεχόμενα της διανομής του Vesper3D

Η επιθυμητή αρχιτεκτονική (32/64bit) όλων των βιβλιοθηκών πρέπει να είναι ίδια ανάμεσα στη μηχανή, τους φυσικούς νόμους και το πρόγραμμα-πελάτη ή την εικονική μηχανή του.

7.1.1. Κυρίως μηχανή

Η κυρίως βιβλιοθήκη της μηχανής περιέχεται στα παρακάτω 3 αρχεία:

- Το αρχείο **Vesper3D32.dll ή Vesper3D64.dll** (ανάλογα με την επιθυμητή αρχιτεκτονική) περιέχει τον μεταγλωττισμένο κώδικα του Vesper3D και θα πρέπει να είναι παρόν στο σύστημα κατά την εκτέλεση του προγράμματος-πελάτη.
- Το αρχείο **Vesper3D32.lib ή Vesper3D64.lib** είναι η βιβλιοθήκη εισαγωγής (import library) και θα πρέπει να είναι παρούσα κατά τη μεταγλώττιση της διεπαφής ή/και του προγράμματος – πελάτη ανάλογα με τον τρόπο υλοποίησης της διεπαφής.
- Τα αρχεία κεφαλίδων **(Vesper3D)\PhysicsEngine.h και (Vesper3D)\PhysicsLaw.h** περιέχουν τις δηλώσεις των τάξεων και συναρτήσεων της μηχανής και θα πρέπει να είναι παρόντα κατά τη μεταγλώττιση της διεπαφής (ή/και του προγράμματος – πελάτη ανάλογα με τον τρόπο υλοποίησης της διεπαφής).

7.1.2. Φυσικοί νόμοι

Οι φυσικοί νόμοι συγγράφονται κατά περίπτωση και τους δίνεται (κατά σύμβαση) όνομα της μορφής **V3dLaw_***ΟνομαΝόμου***XX.dll**, όπου XX η αρχιτεκτονική της μηχανής(32 ή 64 bit).

7.1.3. Πρότυπα

Επίσης, αν πρόκειται να χρησιμοποιηθεί κάποιο πρότυπο, απαιτείται επίσης το αρχείο κεφαλίδας του (το προεπιλεγμένο βρίσκεται στο **Vesper3DDefaultPrototype.h**) και οποιεσδήποτε άλλες εξαρτήσεις του (το προεπιλεγμένο πρότυπο απαιτεί τα αρχεία VectorMath.h και τα αρχεία (VectorMath)*.h)

7.2. Συγγραφή νόμων φυσικής

Για τον κάθε φυσικό νόμο απαιτείται να συγγραφεί ένα αρχείο κώδικα και να μεταγλωττιστεί σε ανεξάρτητη βιβλιοθήκη. Θα πρέπει να υλοποιηθεί η διεπαφή που περιγράφεται στην κεφαλίδα **PhysicsLaw.h**, δηλαδή:

- Να συμπεριληφθεί η κεφαλίδα PhysicsLaw.h
  - (#include <Vesper3D\PhysicsLaw.h>)
- (Προαιρετικά) Να συμπεριληφθεί η κεφαλίδα του προτύπου
  - (#include <Vesper3D\Vesper3DDefaultPrototype.h)
- Να ορισθούν δείκτες στις δομές τις οποίες θα χρησιμοποιήσει για τις φυσικές ιδιότητες που απαιτεί (π.χ. float * masses;v4f * positions ; κ.λπ.).

- Να ορισθούν άλλες απαραίτητες πληροφορίες και μεταβλητές ανάλογα με τις απαιτήσεις (π.χ. . int masses; κ.λπ.)
- Να υλοποιηθεί η μέθοδος Initialize η οποία θα λαμβάνει αναφορά σε PhysicalPropertyProvider, και μέσω της μεθόδου GetPhysicalProperty που αυτή υλοποιεί, θα αποθηκεύει τις πληροφορίες φυσικών ιδιοτήτων στους δείκτες που έχει προβλέψει. Σε περίπτωση επιτυχίας επιστρέφει TRUE (1), διαφορετικά επιστρέφει FALSE(0).
- Να υλοποιηθεί η μέθοδος Calculate με παράμετρο float, και η οποία εκτελεί όλους τους υπολογισμούς για χρονικό βήμα dt.
- Να υλοποιηθεί συνάρτηση GetPhysicsLawInstance, η οποία θα δημιουργεί αντικείμενο (instance) του φυσικού νόμου και θα επιστρέφει δείκτη (pointer) σε αυτό.,
- Σε κάθε περίπτωση σφάλματος, να γράφει στο OutputStream *logger* που προβλέπει η διεπαφή αν αυτό είναι παρόν(π.χ. if (logger) logger<<"Error initializing forces";)

7.3. Συγγραφή της διεπαφής

Η συγγραφή της διεπαφής αποτελεί την κυρίως ανάπτυξη λογισμικού που θα απαιτείται για τη χρήση του Vesper3D. Δεν μπορεί να περιγραφεί 100% εδώ, διότι θα πρέπει να υλοποιηθεί κατά περίπτωση και κατά απαίτηση. Θα περιγραφούν όμως οι λειτουργίες που θα πρέπει να επιτελέσει. Υπενθυμίζεται ότι σε περιπτώσεις (ιδιαίτερα όταν ο πελάτης είναι γραμμένος σε C++), η διεπαφή μπορεί να είναι απλώς τμήμα του προγράμματος-πελάτη, ή να αποτελεί κάποιο ξεχωριστό τμήμα του.
Το βασικό τμήμα της διεπαφής θα είναι γενικά μια τάξη η οποία επεκτείνει τη βασική τάξη PhysicalPropertyManager. Να παρατηρηθεί ότι η τάξη αυτή είναι πλήρως εξωτερική από τη μηχανή φυσικής: είναι σχεδιασμένη για να χρησιμοποιηθεί από αυτήν χωρίς να αποτελεί τμήμα της. Αντίστροφα, έχει γνώση του interface PhysicalPropertyProvider, και κάνει χρήση του ώστε να προσθέσει φυσικές ιδιότητες στη μηχανή φυσικής μέσω της AddPhysicalProperty, και αλλαγές μέσω της GetPhysicalProperty.
Η τάξη αυτή θα παρέχει τουλάχιστον μέθοδο GetPhysicalAttribute η οποία θα χρησιμοποιείται από τον πελάτη για να λάβει τις πληροφορίες φυσικών ιδιοτήτων που απαιτεί, και πιθανότατα θα εκτελεί και διαφορετικές λειτουργίας.
Οι υπόλοιπες λειτουργίες θα πρέπει να δοθούν ανά περίπτωση.
Συχνά αυτές θα είναι:

- Ανάγνωση των αντικειμένων από κάποιου είδους πηγή πληροφοριών (π.χ. αρχείο)
- Ανάθεση και ελευθέρωση μνήμης για τα αντικείμενα αυτά
- Στην περίπτωση που αυτό απαιτείται (ιδιαίτερα αν ο πελάτης είναι γραμμένος σε κάποια γλώσσα προγραμματισμού εικονικής μηχανής), «μετάφραση» των ιδιοτήτων ανάμεσα στον πελάτη και τη μηχανή.

Ως παράδειγμα αναφέρεται η τάξη ObjectProcessorFile που έχει συνταχθεί για την παρούσα εργασία, η οποία αναγιγνώσκει αρχείο που περιέχει αντικείμενα σε προκαθορισμένη μορφή, και αναθέτει ResizeableArray<*datatype*> ανάλογα με τα είδη που βρίσκει.

Αναφορικά, οι μέθοδοι που εκθέτει είναι

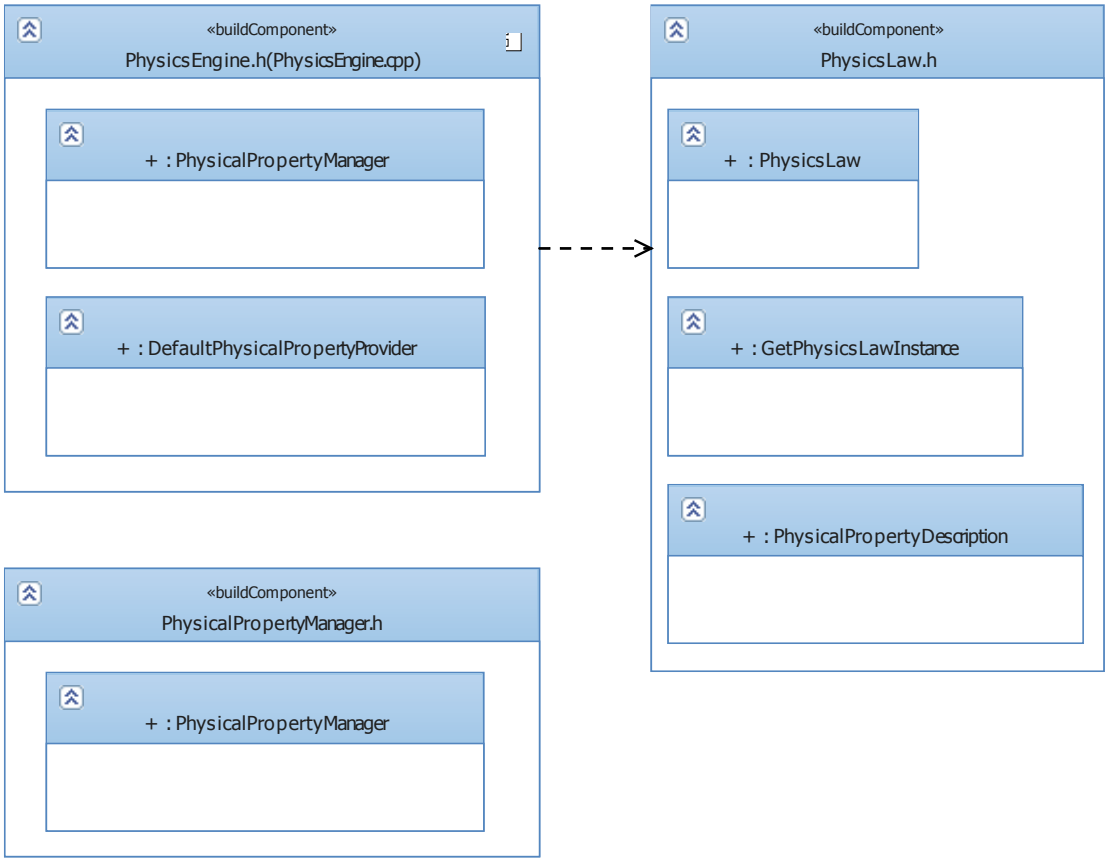
- void * GetPhysicalAttribute(const char * *name*) Επιστρέφει δείκτη στη δομή δεδομένων της φυσικής ιδιότητας με όνομα *name*
- void * GetPhysicalAttributeIndexes(const char * *name*) Επιστρέφει δείκτη στη δομή δεδομένων *των δεικτών* της φυσικής ιδιότητας με όνομα *name*
- void insertObjects(std::istream & w) Διαβάζει πλήρως το input stream w (ρεύμα εισόδου, συνήθως αρχείο), αναγιγνώσκει και αναγνωρίζει (parse) τα αντικείμενα που με συγκεκριμένη δομή περιγράφει αυτό, και τα τοποθετεί εσωτερικά σε δομές δεδομένων κατανοητές από το Vesper3D
- int getSize(const char * name) Επιστρέφει τον αριθμό αντικειμένων που φέρουν την ιδιότητα *name*
- bool addLoadedPropertiesToEngine(PhysicalPropertyProvider & pe) Προσθέτει όλες τις φυσικές ιδιότητες που έχει φορτώσει στον PhysicalPropertyProvider pe ώστε αυτός να χρησιμοποιηθούν από τη μηχανή φυσικής του.

7.4. Χρήση της μηχανής φυσικής Vesper3D από το πρόγραμμα-πελάτη

- Εισαγωγή των κεφαλίδων και της κυρίως βιβλιοθήκης του Vesper3D (π.χ. #include Vesper3D\PhysicsEngine.h)
- Εισαγωγή των απαραίτητων αρχείων της διεπαφής (π.χ. #include Vesper3D\ObjectProcessorFile.h)
- Δημιουργία του αντικειμένου PhysicsEngine (π.χ. PhysicsEngine pe;)
- Ορισμός και Διαχείριση των φυσικών ιδιοτήτων σε δομές δεδομένων
    - Δημιουργία/ Χρήση του αντικειμένου Διεπαφής (π.χ. ObjectProcessorFile opf;) και
    - Δημιουργία/ανάγνωση/χρήση των φυσικών ιδιοτήτων μέσω της διεπαφής

    ή
    - Ορισμός των δομών δεδομένων όπως αυτές απαιτούνται από το απαραίτητο πρότυπο (π.χ. σε float[] ή v4f[]) και προσθήκη τους στη μηχανή και
    - Απευθείας φόρτωσή τους στο PhysicalPropertyProvider της μηχανής
- Φόρτωμα των απαραίτητων φυσικών νόμων (π.χ. pe.LoadPhysicsLaw("Vesper3D_NewtonsLawOfMotion64.dll");
- Μετά τη φόρτωση των νόμων και ιδιοτήτων στη μηχανή, εκτέλεση της Initialize().
- Σε κάθε καρέ, εκτέλεση της Calculate(dt) του PhysicsEngine
- Ενδεχομένως αλλαγή κάποιων φυσικών ιδιοτήτων
- Ενδεχομένως ανάγνωση των ανανεωμένων τιμών κάποιων ιδιοτήτων

7.5. Αρχεία κεφαλίδων



Στο διάγραμμα αυτό βλέπουμε τα απαραίτητα αρχεία κεφαλίδων του Vesper3D.
Το αρχείο κεφαλίδων PhysicsEngine.h (και το αρχείο κώδικα PhysicsEngine.cpp) περιέχει την δήλωση της τάξης Vesper3D. Έχει εξάρτηση από το αρχείο PhysicsLaw.h.
Το αρχείο κεφαλίδων PhysicsLaw.h περιέχει τη δήλωση της διεπαφής φυσικών νόμων, της τάξης PhysicalPropertyProvider της τάξης PhysicalPropertyDescription και της συνάρτησης – εργοστασίου.
Το αρχείο κεφαλίδων PhysicalPropertyManager.h περιγράφει την ελάχιστη διεπαφή που πρέπει να παρέχει μια διεπαφή ώστε να διαχειρίζεται φυσικές ιδιότητες. Δεν εξαρτάται από άλλο αρχείο και κανένα άλλο αρχείο δεν εξαρτάται από αυτό, οπότε και ουσιαστικά αποτελεί οδηγία στον συντάκτη και όχι προαπαιτούμενο.

Για να χρησιμοποιηθεί η μηχανή φυσικής από κάποιο πρόγραμμα (είτε κάποια διεπαφή είτε απευθείας ο πελάτης) απαιτείται το αρχείο κεφαλίδων PhysicsEngine.h και η αντίστοιχη βιβλιοθήκη εισαγωγής PhysicsEngine.lib. Για την εκτέλεση απαιτείται η δυναμική βιβλιοθήκη PhysicsEngine.dll.
Για να συγγραφούν και μεταγλωττισθούν φυσικοί νόμοι απαιτείται η αντίστοιχη κεφαλίδα PhysicsLaw.h. Για τη χρήση τους απαιτείται μόνο η αντίστοιχη δυναμική βιβλιοθήκη (dll).

8. Μελέτη περιπτώσεων (Case Study)

8.1. Μελέτες επιπρόσθετου κόστους

Με τις μελέτες αυτές θα δείξουμε ότι η μηχανή προκαλεί μηδαμινό επιπρόσθετο κόστος εκτέλεσης (overhead) σε ένα σύστημα γραφικών που αναμένεται να τρέχει σε λογικές ταχύτητες (20-120 fps) οπότε η ταχύτητα εκτέλεσης θα εξαρτάται μόνο από την υλοποίηση των συγκεκριμένων φυσικών νόμων.
Τα συστήματα εκτέλεσης είναι:
Σύστημα εκτέλεσης (1)
Επεξεργαστής Intel i5 4.8Ghz
Φυσική Μνήμη 8GB
Κάρτα γραφικών AMD Radeon HD4870 1GB DDR3.
Λειτουργικό σύστημα Microsoft Windows 7 64 bit.

Σύστημα εκτέλεσης (2)
Επεξεργαστής Intel Core 2 Duo 3.8Ghz
Φυσική Μνήμη 4GB
Κάρτα γραφικών AMD Radeon HD6450 1GB DDR3.
Λειτουργικό σύστημα Microsoft Windows 7 64 bit.

8.1.1. Εκτέλεση κενού σεναρίου χωρίς τη μηχανή

Η πλατφόρμα γραφικών που χρησιμοποιείται έχει συγγραφεί από το συντάκτη της παρούσας εργασίας.
Αποτελείται από ένα βασικό βρόχο που η ταχύτητά του εξαρτάται μόνο από τις διαδοχικές κλήσεις SwapBuffers του OpenGl, δηλαδή από την κλήση στη σχεδίαση ανά καρέ. Δεν εκτελούνται άλλες λειτουργίες.
Σε κάθε καρέ, εκτελείται κλήση σε συνάρτηση Update,
Η Update που καλείται από την πλατφόρμα σε κάθε καρέ δεν περιέχει τίποτα.

```
void Update(float dt){ }
```

Στο κενό σενάριο αυτό το σύστημα 1 έδωσε 1348 fps (χρόνος καρέ 0.7418 msec)
Στο κενό σενάριο αυτό το σύστημα 2 έδωσε 405 fps (χρόνος καρέ 2.469 msec)

8.1.2. Εκτέλεση σεναρίου με κενή μηχανή φυσικής

Στην ίδια πλατφόρμα γραφικών, θα προστεθεί μηχανή στην οποία θα γίνονται κλήσεις σε κάθε καρέ, αλλά η μηχανή δεν θα έχει φυσικούς νόμους να εκτελέσει. Θα μετρηθούν τα καρέ ανά δευτερόλεπτο, και θα πρέπει οι κλήσεις να επιστρέψουν ταχέως για να μην προκληθεί μεγάλο επιπρόσθετο φορτίο.
Ήτοι, το αρχείο Laws.ini που χρησιμοποιεί η εφαρμογή για τη φόρτωση φυσικών νόμων είναι κενό.

```
void Update(float dt){
    objects->Update(dt*updateFactor);
}
```

Σύστημα 1 : 1332 fps (χρόνος καρέ 0.7507 msec)
Σύστημα 2 : 395 fps (χρόνος καρέ 2.531 msec)

Από τα δυο παραπάνω σενάρια συμπεραίνουμε ότι η βασική κλήση της update εμπεριέχει επιβάρυνση (overhead) :
Συστημα 1: 0,0089 msec = 8.9 μsec =~1.18% του επεξεργαστικού χρόνου
Συστημα 2: 0,062 msec = 62 μsec = ~0,1569% του επεξεργαστικού χρόνου

Σε μια τυπική εφαρμογή που τρέχει στα 60fps, η επιβάρυνση αυτή θα ισούταν με:

Συστημα 1: 0.0089/(1/60) /1000 = 0,000534 =0.0534%, λιγότερο δηλαδή από μισό τις χιλίοις, πρακτικά μηδέν
Συστημα 2: 0.062/(1/60)/1000 = 0,00372 =0.372%, περίπου δηλαδή εν τρίτο τοις εκατό.

8.1.3. Εκτέλεση σεναρίου με μηχανή φυσικής με κενές φυσικές ιδιότητες

Θα προστεθούν κάποιοι φυσικοί νόμοι στη μηχανή, αλλά οι φυσικές ιδιότητες θα είναι κενές. Μετρήθηκαν τα καρέ ανά δευτερόλεπτο, και θα πρέπει οι κλήσεις να επιστρέψουν ταχέως για να μην προκληθεί μεγάλο επιπρόσθετο φορτίο.

Σύστημα 1 : 1332 fps (χρόνος καρέ 0.7507 msec)
Σύστημα 2 : 394 fps (χρόνος καρέ 2.538 msec)

Παρατηρούμε μη μετρήσιμη επιβάρυνση σε σχέση με το προηγούμενο παράδειγμα

8.1.4. Εκτέλεση πλήρους απλού σεναρίου

Στο σενάριο αυτό προστίθενται οι ιδιότητες 4 επιφανειών που αποτελούν τετράεδρο συν μια ακλόνητη επιφάνεια (επιφάνεια με σώματα μηδενικής μάζα ως κορυφές). Οι μάζες στις κορυφές του τετραέδρου συνδέονται με ελατήρια για τον ορισμό στερεού σώματος. Το τετράεδρο πέφτει υπό την επίδραση βαρύτητας και συγκρούεται με την ακλόνητη επιφάνεια.

Οι φυσικοί νόμοι που χρησιμοποιήθηκαν είναι:

- NewtonsLawOfMotion (υλοποίηση απλών νόμων της κινηματικής και της αδράνειας).
    - Χρησιμοποιούμενες φυσικές ιδιότητες: mass(float, μάζα), position(v4f, θέση),velocity(v4f, ταχύτητα), force(v4f, συνισταμένη δύναμη),externalforce(v4f, εξωτερικές δυνάμεις)
- GravityField (Υλοποίηση βαρυτικού πεδίου, εδώ με ένταση 9.81m/sec^2 και φορά κατά τον αρνητικό z άξονα
    - Χρησιμοποιούμενες φυσικές ιδιότητες: mass(float) μάζα, force(v4f, δύναμη), GravityField(συνάρτηση v4f/v4f, πλήρης περιγραφή του βαρυτικού πεδίου για τη δεδομένη θέση)
- Springs (Υλοποίηση δυνάμεων ελατηρίου)
    - Χρησιμοποιούμενες φυσικές ιδιότητες: endpoints(int) άκρα του ελατηρίου), hookeConstant(float) σκληρότητα, damping(float, συντελεστής ιξώδους απόσβεσης), length (float, μήκος ηρεμίας ελατηρίου)

Σύστημα 1 : 1332 fps (χρόνος καρέ 0.7507 msec)
Σύστημα 2 : 394 fps (χρόνος καρέ 2.538 msec)

Παρατηρούμε επίσης πρακτικά μηδενική επιβάρυνση σε σχέση με το προηγούμενο παράδειγμα.

8.1.5. Συμπεράσματα Μελετών επιπροσθέτου κόστους:

Αποδεικνύεται από τα παραπάνω παραδείγματα ότι η μηχανή σε απλές εφαρμογές σε ένα σύγχρονο υπολογιστή δεν προκαλεί σοβαρό επιπρόσθετο κόστος.
Το συμπέρασμα που μπορούμε να εξάγουμε δεν είναι φυσικά ότι η προσθήκη φυσικής θα είναι «δωρεάν» όσον αφορά την απόδοση.
Όμως, τελικά, αφού το **επιπρόσθετο** κόστος (*overhead*) της μηχανής είναι πρακτικά μηδενικό, η απόδοση του συστήματος είναι πλέον αποκλειστικά υπό τον έλεγχο και την ευθύνη του σχεδιαστή των φυσικών νόμων, και εμείς δεν τον επιβαρύνουμε πρακτικά καθόλου με επιπρόσθετο κόστος. **Διεκδικούμε συνεπώς το δικαίωμα να αναφερόμαστε στο Vesper3D ως μια πάρα πολύ ελαφριά από πλευράς κόστους εκτέλεσης μηχανή φυσικής.**

8.2. Μελέτες χρηστικότητας – Custom Μηχανή γραφικών OpenGL

Μια βασικότατη προϋπόθεση για τη χρηστικότητα του Vesper3D είναι η ικανότητα να απεικονίζονται τα σώματα που περιγράφει και διαχειρίζεται.

8.2.1. Σύνδεση φυσικών ιδιοτήτων με απεικονιζόμενα αντικείμενα σε custom μηχανή γραφικών OpenGL

Ως πρόγραμμα – πελάτης θα χρησιμοποιηθεί μια απλή μηχανή γραφικών OpenGL η οποία σε κάθε καρέ εκτελεί κλήση σε συνάρτηση που δημιουργήθηκε για το σκοπό αυτό (Update). Μέσα από αυτή τη συνάρτηση εκτελείται η κλήση στην Update του Vesper3D ώστε να εκτελεστούν οι υπολογισμοί του καρέ.
Στην οικογένεια αυτή παραδειγμάτων θα συνδεθούν ιδιότητες της μηχανής φυσικής με απεικονιζόμεντα artifacts του OpenGL.
Για την διαχείριση των φυσικών ιδιοτήτων θα χρησιμοποιηθεί η διεπαφή ObjectProcessorFile που δημιουργήθηκε για να διαβάζει τα αντικείμενα από αρχείο. Η διεπαφή αυτή δίνει τη δυνατότητα να ορίζονται τα αντικείμενα σε αρχείο .ini με σχετικά απλό τρόπο ώστε να μπορούν εύκολα να ορισθούν τα παραδείγματα που θέλουμε να εκτελέσουμε.
Η διεπαφή αυτή ορίζει – καθαρά για λόγους ευκολίας – αυτόματα κάποιες από τις φυσικές ιδιότητες για όλα τα σημειακά σώματα, όπως την ιδιότητα **force** την οποία πολυάριθμοι φυσικοί νόμοι χρησιμοποιούν για να αποθηκεύσουν τη δύναμη που υπολογίζεται ότι ασκούν σε σώματα, αλλά δεν είναι κατάλληλη για να χρησιμοποιηθεί από το πρόγραμμα-πελάτη.
Σε άλλες διεπαφές η παραδοχή αυτή δεν ισχύει και όλες οι ιδιότητες πρέπει να ορισθούν ρητά και να τους δοθούν αρχικές τιμές.
Για την απεικόνιση των ιδιοτήτων ως σώματα:

Θα δημιουργηθεί **απευθείας σύνδεση της φυσικής ιδιότητας Θέση (position)** που χρησιμοποιείται από πολυάριθμους νόμους φυσικής **με την εμφάνιση σημείων στο OpenGL.**
Επίσης, **θα συνδεθεί η φυσική ιδιότητα Endpoints** (που χρησιμοποιείται από το φυσικό νόμο springs για να ορισθούν τα άκρα ενός ελατηρίου στον πίνακα positions) **για την εμφάνιση γραμμών** με άκρα τις θέσεις στις οποίες αναφέρεται**.**
Τέλος, **θα συνδεθεί η φυσική ιδιότητα Vertices** (που χρησιμοποιείται από το φυσικό νόμο SurfaceCollisionDetector για τον ορισμό επιφανειών) **για να εμφανισθούν επίπεδα** με άκρα τις θέσεις στις οποίες αναφέρονται**.**
Με τον παραπάνω τρόπο εμφάνισης κατανοούμε ότι στα παραδείγματά μας θα εμφανίζονται τα «σώματα» ως σημεία, οι «σύνδεσμοι» ως γραμμές και οι «επιφάνειες» ως επίπεδα.
Τα διάφορα χρώματα που εμφανίζονται είναι «καρφωτά στον κώδικα» hardcoded.
Ας συνειδητοποιήσουμε εδώ ότι φυσικά οι παραπάνω παραδοχές δεν είναι υποχρεωτικές. Κατά πάσα πιθανότητα, σε μια τελική χρήση του Vesper3D από μια πλήρη μηχανή παιχνιδιού, η φυσική ιδιότητα position θα μπορούσε π.χ. να συνδεθεί με το διάνυσμα που εκφράζει τη θέση ενός ολόκληρου σώματος και όχι απλά ενός σημείου, ή με οποιοδήποτε άλλο αριθμό του φυσικού κόσμου που απεικονίζεται.
Θα πρέπει εδώ να συνειδητοποιήσουμε ότι για να γίνει η παραπάνω παραδοχή, τελικά οι φυσικές ιδιότητες που δεν απαιτείται να χρησιμοποιηθούν από φυσικούς νόμους για αν απεικονισθούν. **Απαιτείται μόνο να έχουν ορισθεί και τροφοδοτηθεί στη μηχανή φυσικής** (κλήση AddProperty(xxxx)). Θα μπορούσε π.χ. να ορισθεί μια διανυσματική ιδιότητα Color η οποία να εκφράζει το χρώμα κατά RGBA ενός αντικειμένου, και με τη σειρά της να αναγιγνώσκεται από τον Πελάτη για την εμφάνιση σωμάτων έγχρωμα κατά αντικείμενο, παρ' ότι η ιδιότητα αυτή δεν χρησιμοποιείται από φυσικούς νόμους και δεν είναι αυστηρά «φυσική».
Το Vesper3D συνεπώς έχει την δυνατότητα– και είναι σαφώς θεμιτό– να λειτουργεί ως ένας γενικότερος διαχειριστής ιδιοτήτων αντικειμένων.
Θα εκτελεστούν σενάρια με διάφορους φυσικούς νόμους και σε διαφορετικές πλατφόρμες για να αξιολογηθεί η χρηστικότητα της εφαρμογής.

8.2.2. Ευθύγραμμη Ομαλή Κίνηση

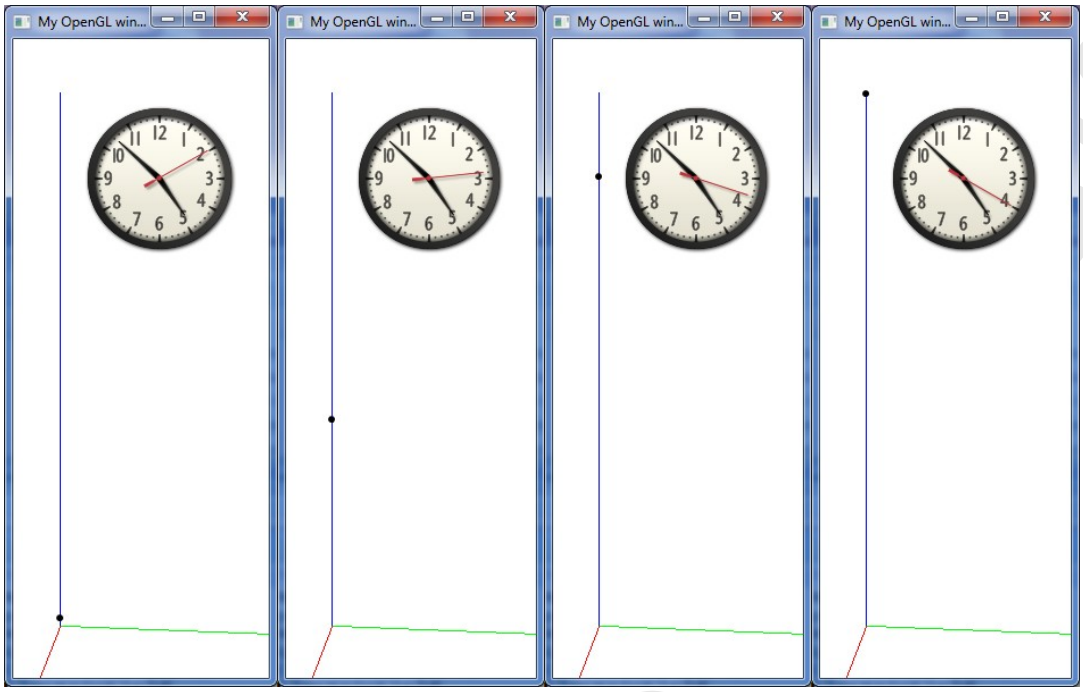
Το σενάριο αυτό απαιτεί ένα σώμα με μάζα και κάποια αρχική ταχύτητα. Απουσία άλλων φυσικών νόμων, το σώμα αυτό οφείλει να κινείται ευθύγραμμα και με σταθερή ταχύτητα.
Απαιτούμενοι φυσικοί νόμοι:

1) NewtonsLawOfMotion (Νόμοι νευτώνιας μηχανικής)

Απαιτούμενες φυσικές ιδιότητες:

1) Position (Θέση του σώματος, v4f)
2) Velocity (Ταχύτητα του σώματος, v4f)
3) Mass (Μάζα, float)
4) Force (Συνισταμένη δύναμη. Ορίζεται αυτόματα από την μηχανή (ObjectProcessorFile) διότι δεν προορίζεται να χρησιμοποιείται από τον πελάτη.

Μέσω του αρχείου Laws.ini φορτώνουμε αποκλειστικά το νόμο NewtonsLawOfMotion.
Στο αρχείο CaseStudy11-LinearMotion.ini ορίζουμε [Body] με μάζα 1kg \, αρχική θέση 0,0,0 και αρχική ταχύτητα 0,0,1. Αναμένουμε να διανύσει τον άξονα Z (10m) σε 10 δευτερόλεπτα.



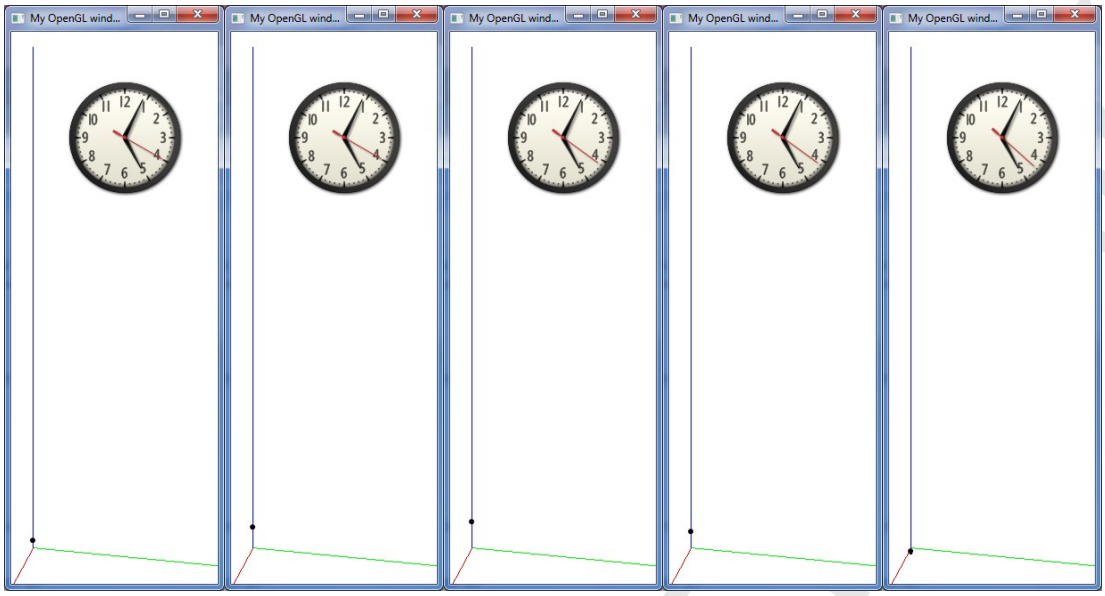
Αποτέλεσμα: Επιτυχές.

8.2.3. Προσθήκη βαρυτικού πεδίου στην κατακόρυφη βολή

Προστίθεται επιτάχυνση της βαρύτητας 9.81m/sec^2 προς τα κάτω και επαναλαμβάνεται το παράδειγμα. Αναμένεται σε λίγο περισσότερο από 1 δευτερόλεπτο να έχει ακινητοποιηθεί η μάζα και σε λίγο περισσότερο από 2 να έχει κατέβει κάτω από το σημείο εκτόξευσής της.
Απαιτούμενος επιπρόσθετος νόμος: GravityField. Το ομαλό βαρυτικό πεδίο στη συγκεκριμένη περίπτωση υλοποιείται ως συνάρτηση στον κώδικα του παραδείγματος (επειδή το πεδίο είναι ομαλό, σε κάθε σημείο του χώρου επιστρέφει τιμή (0,0,-9.81).

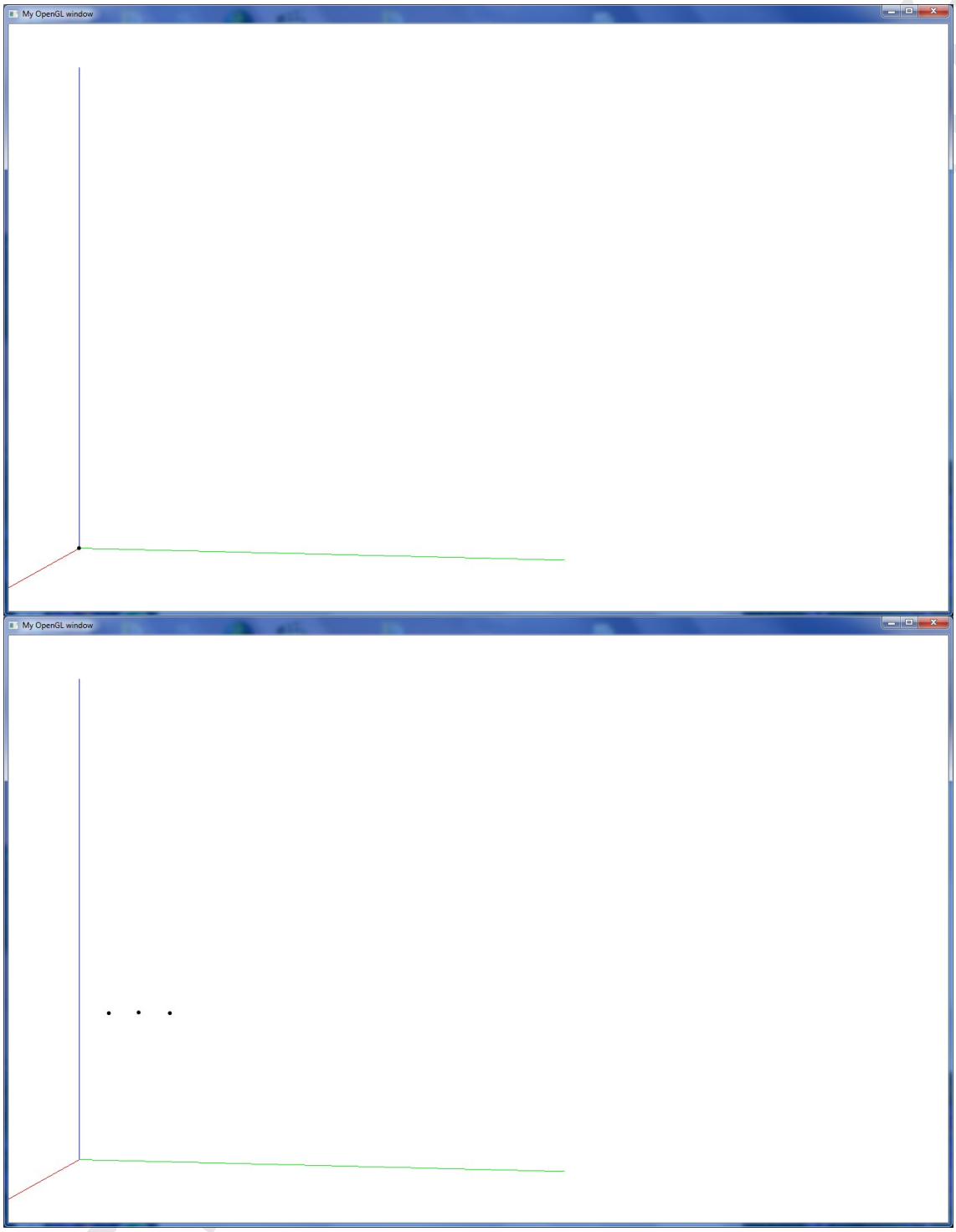
```
inline v4f GravityField(v4f & pos)
{
   return v4f(0,0,-9.81f);
}
```

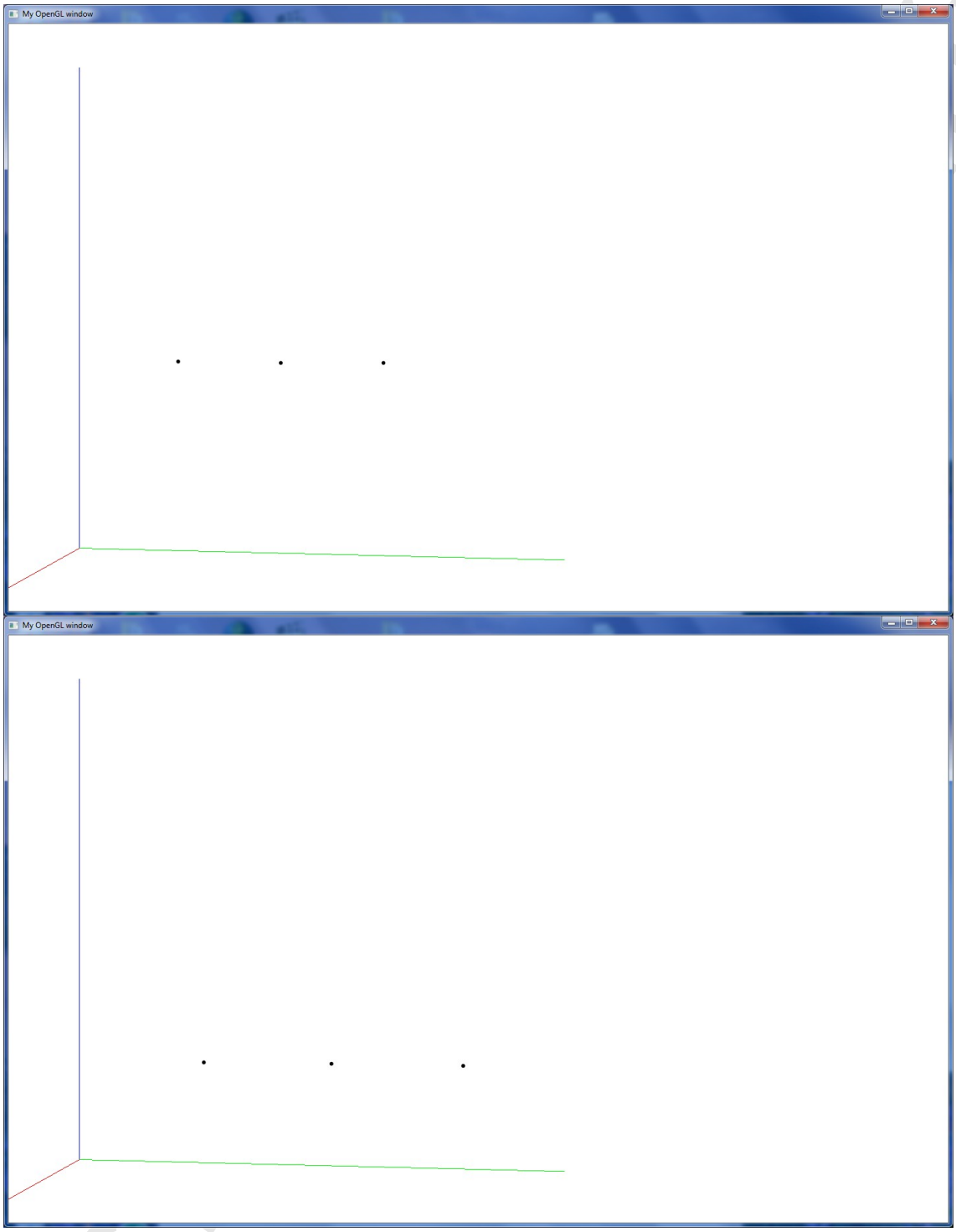
Με τον ίδιο ακριβώς τρόπο θα μπορούσαμε να υλοποιήσουμε οποιαδήποτε μορφή βαρυτικού πεδίου επιθυμούμε.

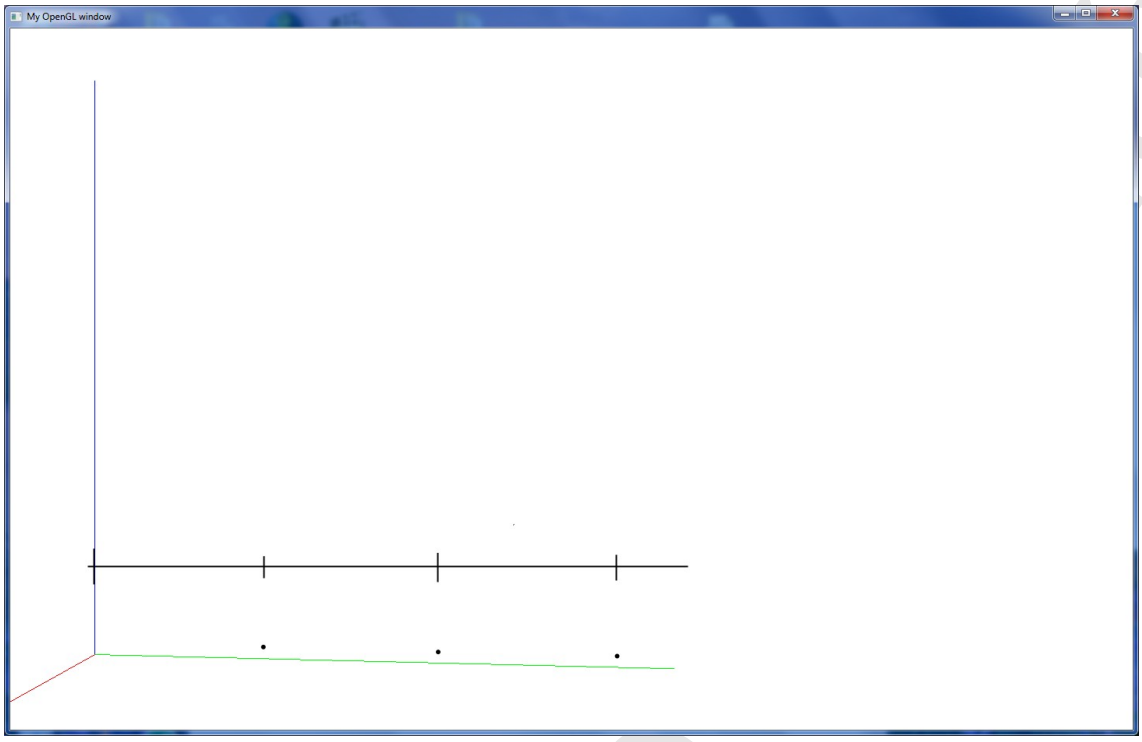


8.2.4. Πλάγια βολή υπό την επίδραση βαρυτικού πεδίου

Το σενάριο αυτό απαιτεί ένα βαρυτικό πεδίο ισχύος 9.81m/sec^2 και μια μάζα που εκτοξεύεται με κάποια αρχική ταχύτητα και εκτελεί παραβολική τροχιά υπό το πεδίο αυτό.
Η βολή θα εκτελεσθεί για 3 σώματα ταυτόχρονα με ίδια συνιστώσα ταχύτητας κατά z (30, αντίστροφη του βαρυτικού πεδίου), και 3 διαφορετικές οριζόντιες ταχύτητες (5,10,15)
Ποιοτικά, η τροχιά των σωμάτων σώματα θα πρέπει να έχει σε κάθε χρονική στιγμή την ίδια κατακόρυφη συνιστώσα και οι οριζόντιες συνιστώσες να έχουν ανά δύο και από τον άξονα τον x την ίδια απόσταση μεταξύ τους.



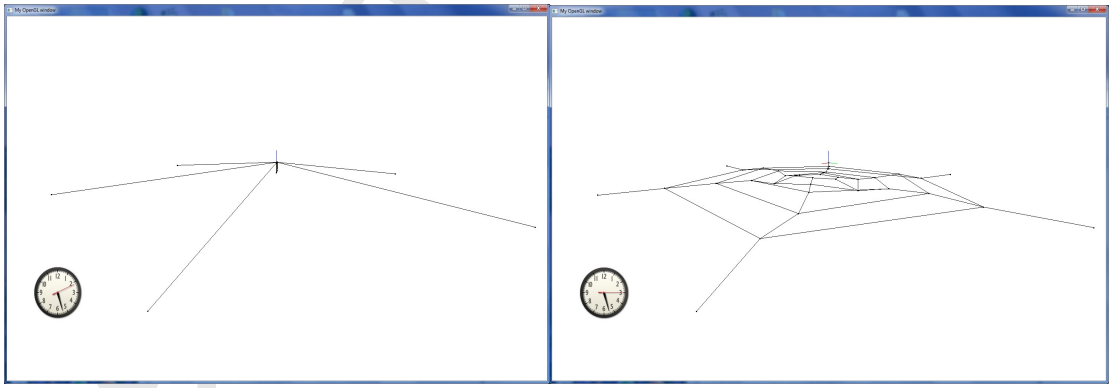


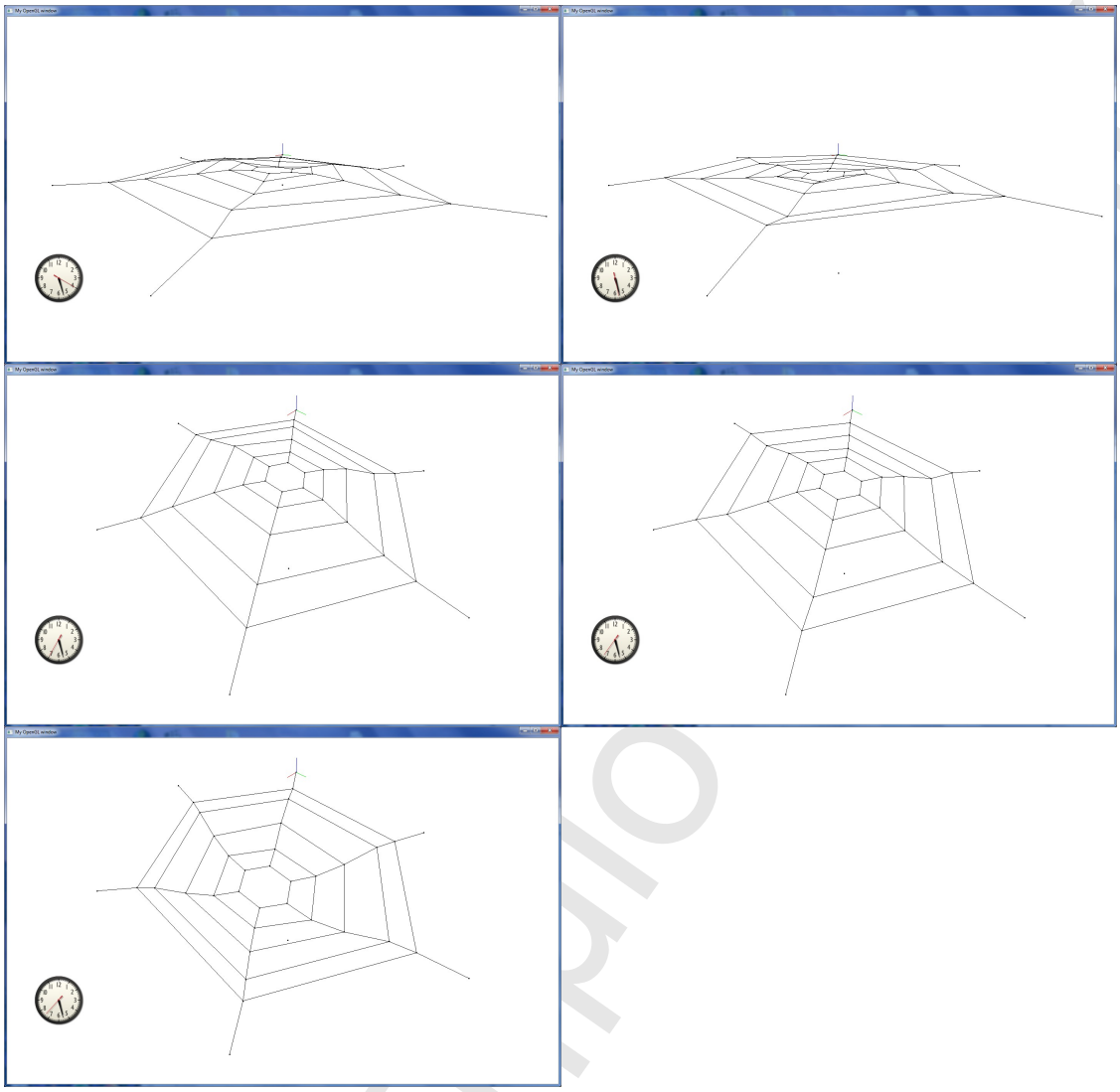


Παρατηρούμε επ' ακριβώς την αναμενόμενη συμπεριφορά.

8.2.5. Μάζες συνδεδεμένες με ελατήρια

Θα προστεθούν φυσικοί νόμοι για δυνάμεις ελατηρίων. Θα ορισθεί σχετικά μεγάλος αριθμός μαζών (20) οι οποίες θα συνδεθούν με τρόπο που προσομοιάζει ιστό αράχνης. Τα ελατήρια θα είναι μετρίως ελαστικά και με σχετικά μικρή απόσβεση.
Θα δοθούν αρχικές θέσεις με τυχαίες μετατοπίσεις ώστε να δημιουργηθεί μια ασύμμετρη, φυσική κίνηση που προσομοιάζει ιστό αράχνης.





8.2.6. Φασματογράφος μάζας – Χρήση ορισμών πεδίων

Για το παράδειγμα αυτό θα προστεθούν 2 πεδία, ηλεκτρικό και μαγνητικό που θα επενεργούν σε συγκεκριμένες περιοχές του χώρου ώστε να λειτουργήσει το παράδειγμα.
Θα αφεθούν θετικά ηλεκτρικά φορτισμένα σωματίδια μέσα σε περιοχή του χώρου όπου θα βρίσκεται ηλεκτροστατικό πεδίο. Ο λόγος που επιλέγεται όλα τα σωματίδια να φέρουν το ίδιο φορτίο είναι διότι διαφορετικά θα επιταχυνθούν προς αντίθετες διευθύνσεις, πράγμα που δεν εξυπηρετεί στο συγκεκριμένο παράδειγμα.
Τα φορτισμένα σωματίδια συνεπώς θα δεχθούν δύναμη φοράς ίδιας με του ηλεκτρικού πεδίου και γνωστής τιμής κατά την σχέση **F**=q***E** όπου F η δύναμη, q το φορτίο και E το ηλεκτρικό πεδίο.
Το ηλεκτρικό πεδίο θα είναι σταθερής τιμής διεύθυνσης, και θα βρίσκεται σε συγκεκριμένη περιοχή του χώρου, όπου θα λειτουργήσει ως επιταχυντής, αφού θα προσδώσει ταχύτητα στα φορτισμένα σωματίδια.
Τα φορτισμένα σωματίδια θα βγουν από την περιοχή του χώρου του ηλεκτρικού πεδίου και θα εισέλθουν σε περιοχή του χώρου με μαγνητικό πεδίο.

Το μαγνητικό πεδίο θα είναι επίσης περιορισμένων ορίων και σταθερής τιμής. Η φορά του θα είναι κάθετη στο επίπεδο παρατήρησης.
Τα φορτισμένα σωματίδια **οφείλουν** αν έχουν υλοποιηθεί σωστά τα παραπάνω, να δεχθούν δύναμη πάντοτε κάθετη στην ταχύτητά τους και το μαγνητικό πεδίο, κατά το νόμο F=q***B**x**u** (όπου x η διανυσματική πράξη εξωτερικό γινόμενο). Για κάθετα μεταξύ τους πεδία αυτό καταλήγει σε τιμή δύναμης F=q*B*u.
Εφ' όσον η δύναμη είναι σταθερής τιμής και πάντοτε κάθετη στην ταχύτητα, οφείλουν τα σωματίδια να εκτελέσουν κυκλικές τροχιές.
Επιλύοντας μεταξύ τους όλες τις παραπάνω εξισώσεις, θα βρούμε ότι η ακτίνα της τροχιάς εξαρτάται άμεσα από την αναλογία m/q των σωματιδίων που εκτοξεύουμε.

Οι φυσικοί νόμοι που είναι απαραίτητοι για το πείραμα αυτό είναι οι: NewtonsLawOfMotion, ElectricField, MagneticField.

Στον κώδικα, ορίζονται οι παρακάτω συναρτήσεις. Δείκτες συναρτήσεων (Function Pointers σε αυτές) τροφοδοτούνται ως φυσικές ιδιότητες **"ElectricField"** και **"MagneticField"** στο Vesper3D. Οι συναρτήσεις αυτές έχουν την παρακάτω απλή υλοποίηση:

```
v4f ElectricField(v4f & pos)
{
   //Square Prism shaped electric field bounded at (-50,0),(-50,-100),(50,-100),(50,0) With a direction
of (0,.5,0) ("Right")
   if (pos.x>-50 && pos.x<50 && pos.y>-100 && pos.y<0)
   {
      return v4f(0,.5,0);
   }
   else return v4f::zero();
}

v4f MagneticField(v4f & pos)
{
   //Square Prism shaped electric field bounded at (-1000,0),(-1000,inf),(1000,inf),(1000,0) with a
direction of (0,0,1) ("Up")
   if (pos.x>-1000 && pos.x<1000 && pos.y>0)
   {
      return v4f(0,0,.02f);
   }
   else return v4f::zero();
}
```

Σε ψευδοκώδικα, αυτές οι δυο συναρτήσης θα διαβαζόταν:
Ηλεκτρικό πεδίο:

- Για δεδομένο **διάνυσμα** θέσης:
  - Αν το διάνυσμα βρίσκεται σε περιοχή του χώρου ανάμεσα στα x:50,x:-50, y:100,y:-100 και για οποιοδήποτε z, επέστρεψε ηλεκτρικό πεδίο εντάσεως 0.5 N/Cb με φορά προς το θετικό y (πράσινος άξονα)
  - Διαφορετικά, επέστρεψε μηδενικό διάνυσμα

Μαγνητικό πεδίο:

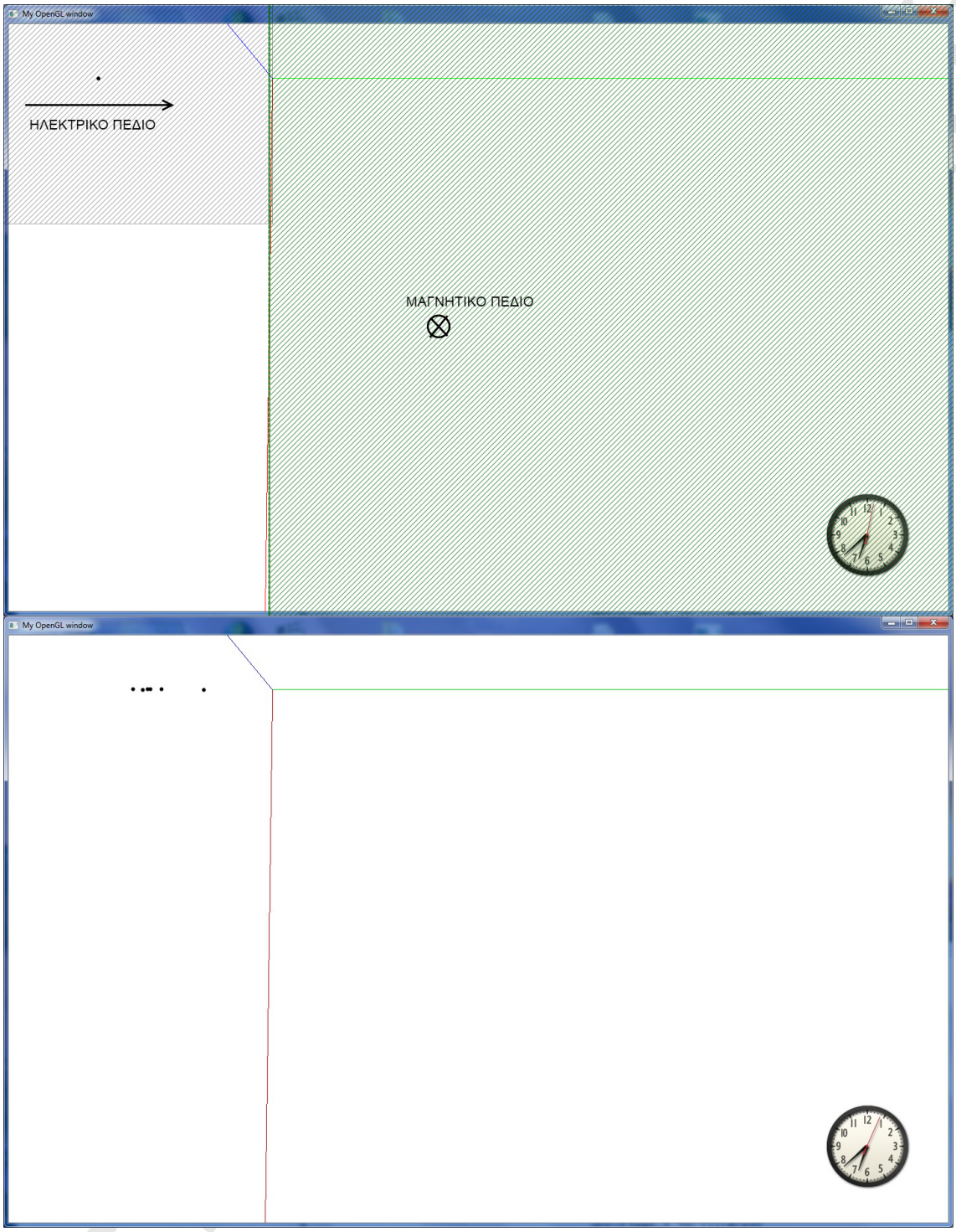
- Για δεδομένο **διάνυσμα** θέσης:
  - Αν το διάνυσμα βρίσκεται σε περιοχή του χώρου ανάμεσα στα x:1000,x:-1000, για οποιοδήποτε θετικό y και για οποιοδήποτε z,

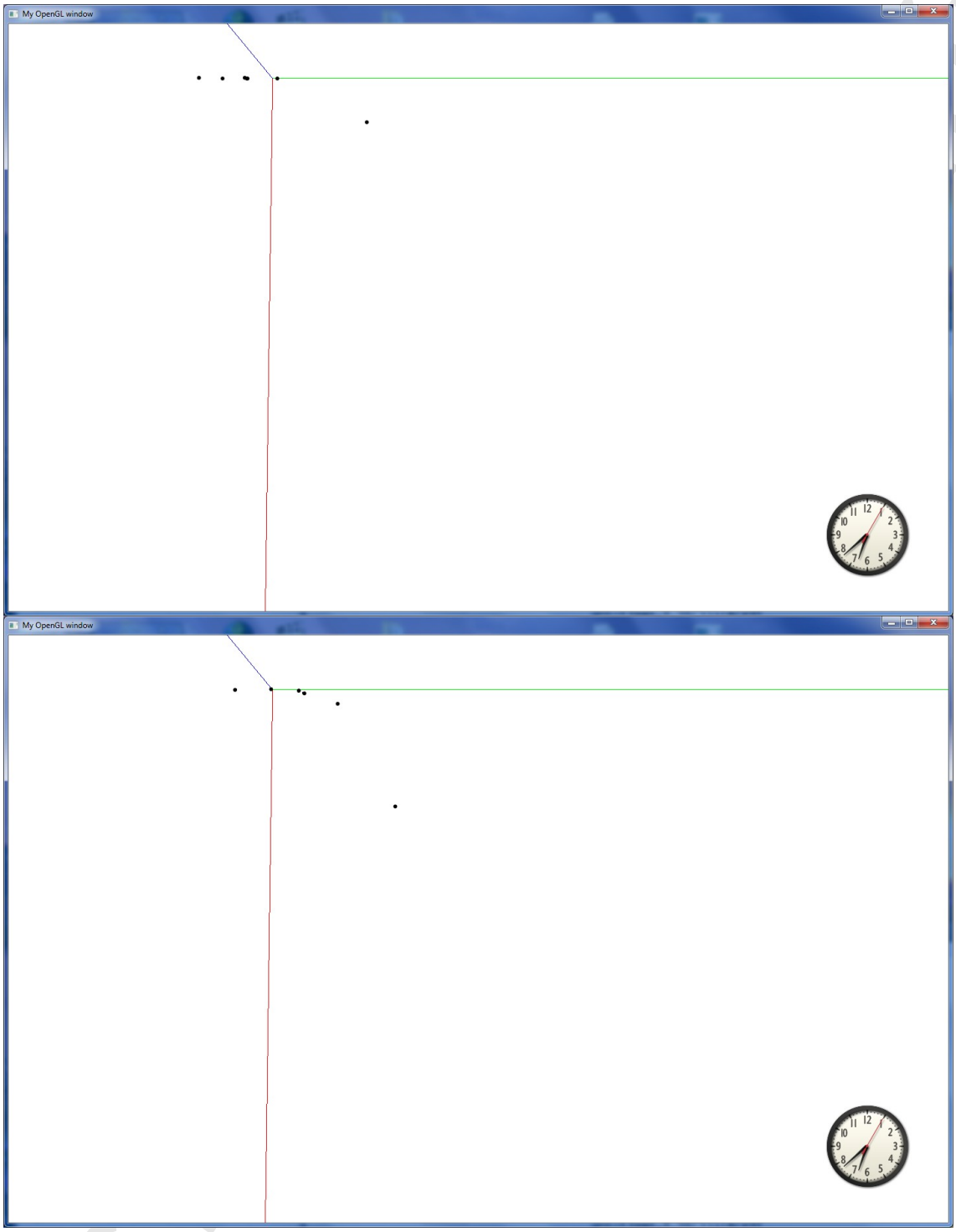
επέστρεψε διάνυσμα μαγνητικού πεδίου εντάσεως 0.02 Henry με φορά προς το θετικό z (μπλε άξονας)
- Διαφορετικά, επέστρεψε μηδενικό διάνυσμα

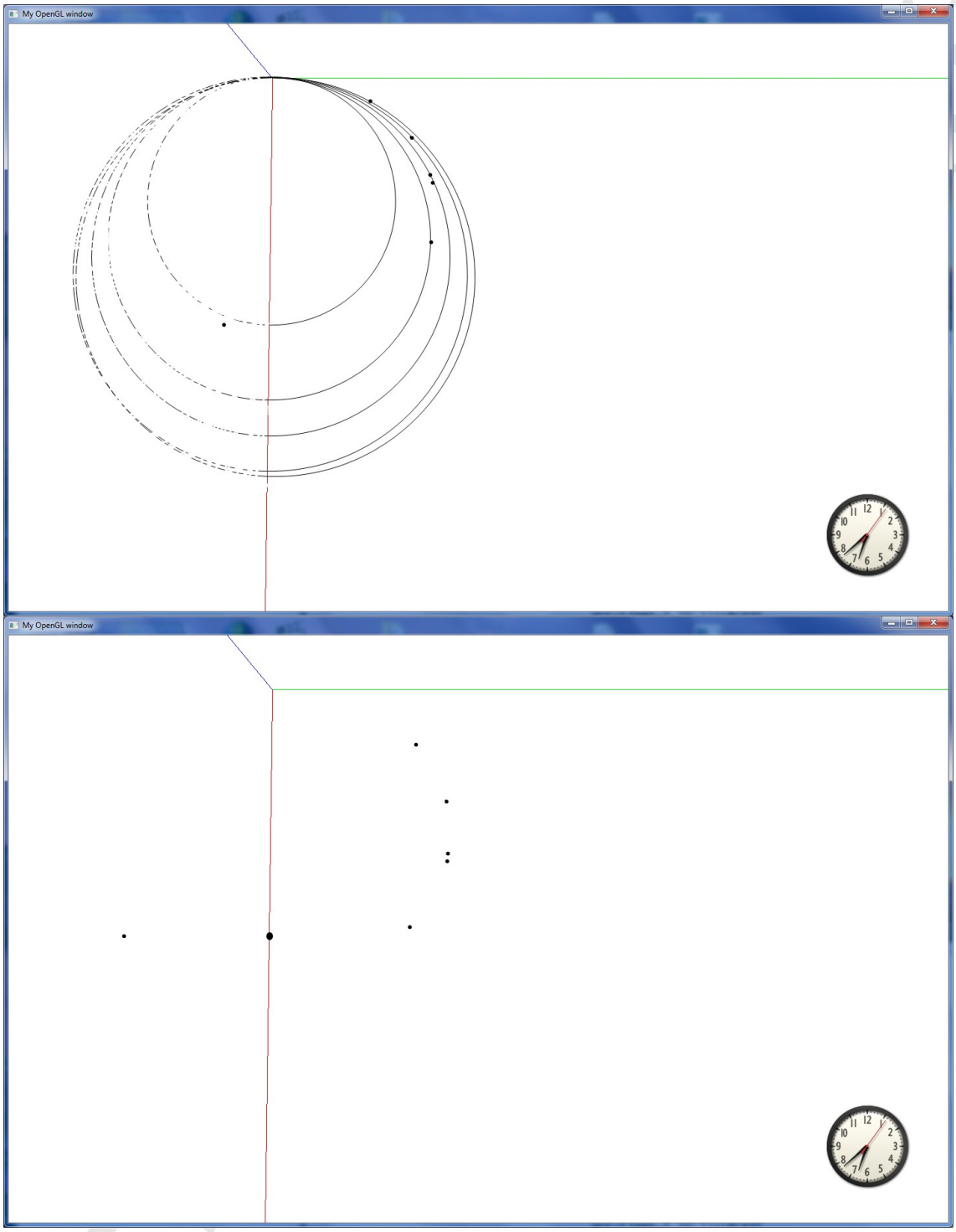
Εισάγονται 6 σώματα με τα παρακάτω στοιχεία:

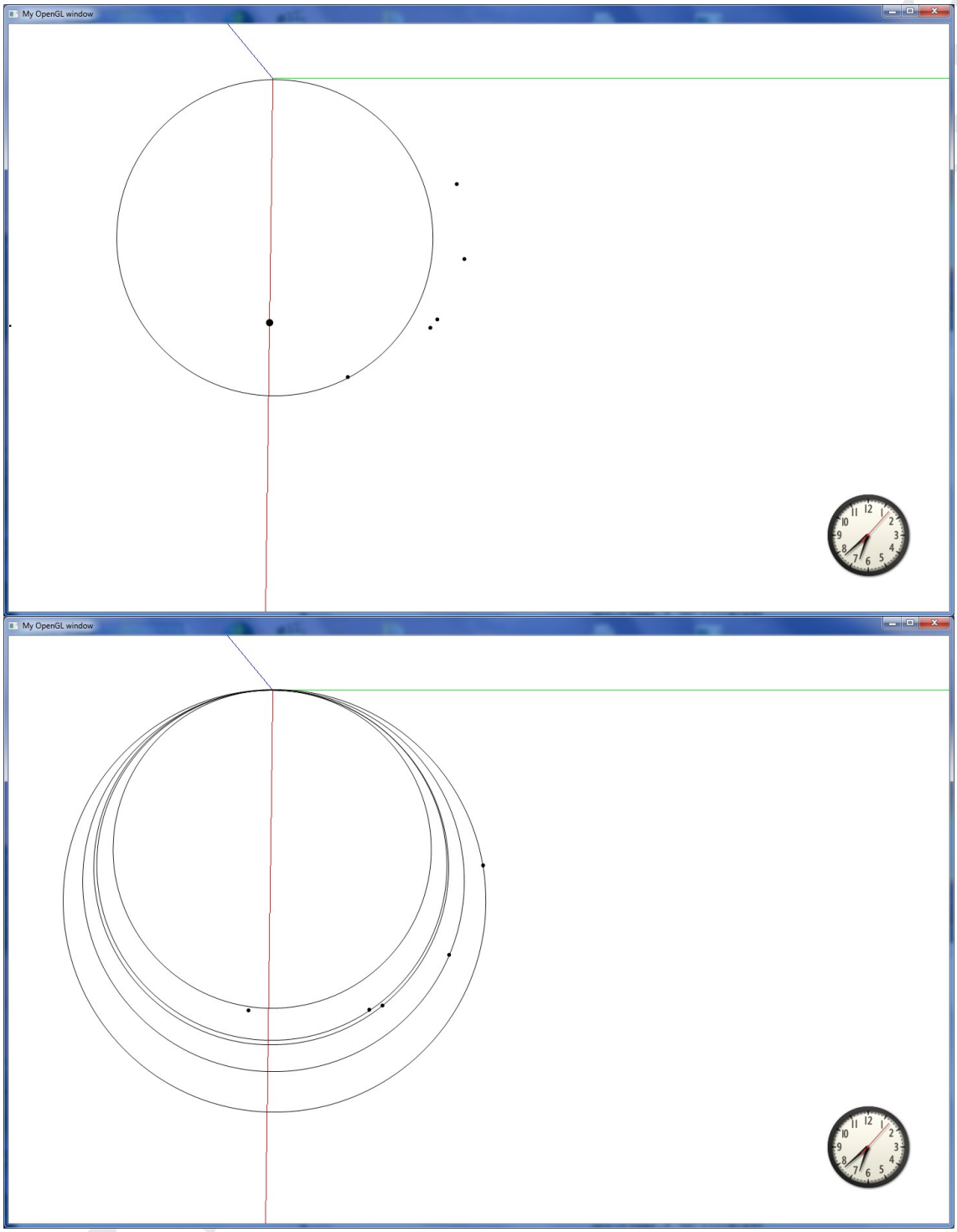
Μάζα	Φορτίο	Λόγος q/m
.0002	.02	100
.0002	.012	60
.0002	.01	50
.00041	.02	48.7
.00024	.01	41.6
.0003	.01	33,3

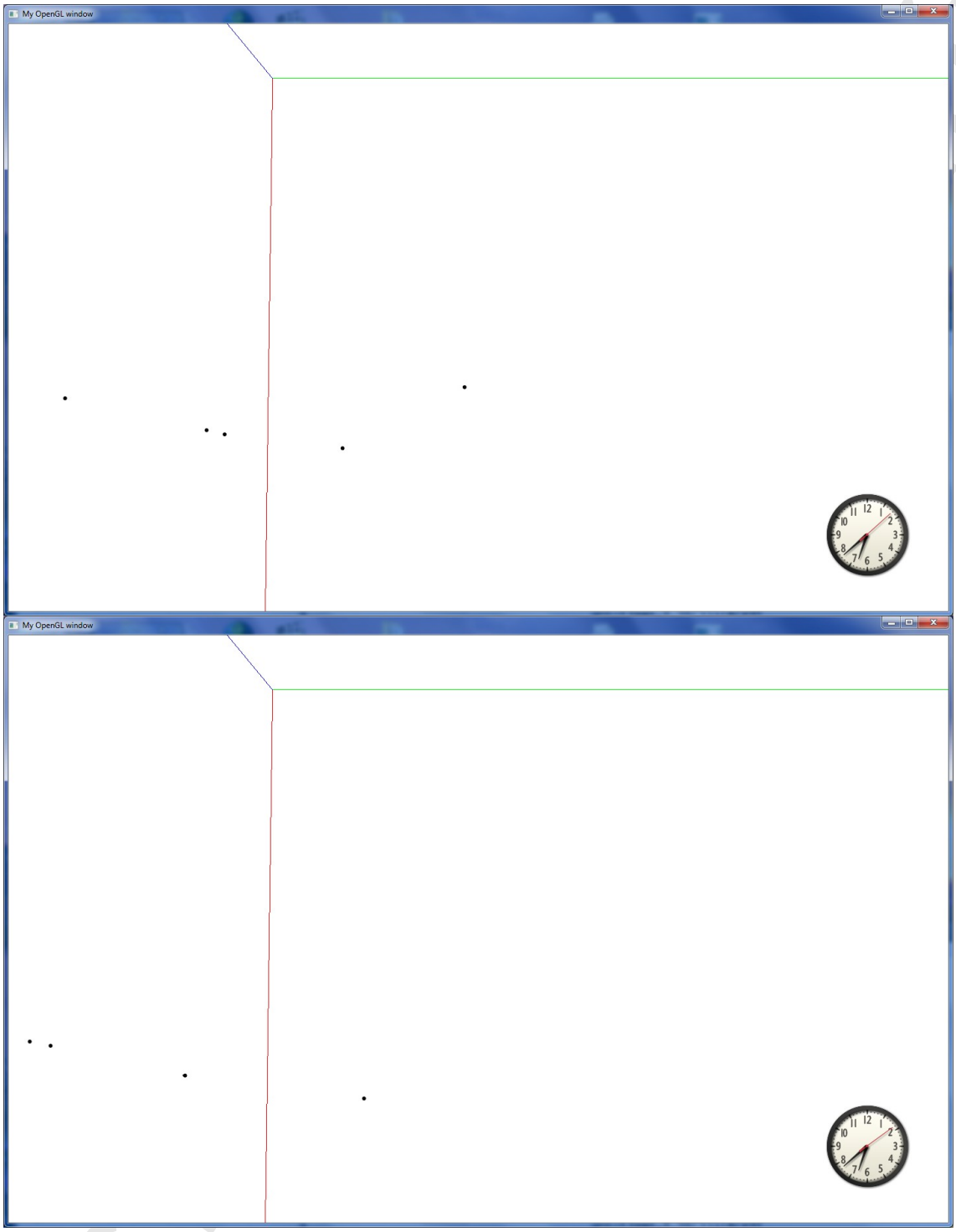
Ποιοτικά, θα έχουμε 1 σώμα το οποίο ταχέως θα προσπεράσει τα υπόλοιπα και θα ακολουθήσει τροχιά «μικρού» κύκλου, και 5 σώματα τα οποία θα είναι σχετικώς κοντά με μια μικρή διασπορά μεταξύ τους με μια σχετικά καλή διασπορά.
Επίσης, για το συγκεκριμένο πείραμα θα αφαιρεθεί το βαρυτικό πεδίο ώστε να μην είμαστε υποχρεωμένοι να λειτουργήσουμε με μεγέθη κοντά στην πραγματικότητα: Σε πραγματικό φασματογράφο μάζας, η βολή διαρκεί ελάχιστα microsecond ή και λιγότερο, πράγμα που αποτρέπει τη μακροσκοπικής κλίμακος επιτάχυνση της βαρύτητας από το να επηρεάσει τα αποτελέσματα της βολής.
Ενώ λοιπόν τίποτα δεν μας αποτρέπει από το να χρησιμοποιήσουμε πραγματικά μεγέθη και κάποια αναλογία του χρόνου ώστε να καταστούν οι βολές παρατηρήσιμες, το συγκεκριμένο πείραμα μας ενδιαφέρει ποιοτικά μόνο, οπότε θα αφαιρέσουμε την βαρυτική δύναμη και θα χρησιμοποιήσουμε μακροσκοπικές τιμές για τις φυσικές ιδιότητες που μας ενδιαφέρουν.
Οι παραπάνω τιμές έχουν επιλεγεί αυθαίρετα με μοναδικό σκοπό την δημιουργία ενός παραδείγματος το οποίο θα μπορεί να εκτελεστεί με «βολικές» τιμές μεγεθών που θα μπορούν να παρατηρηθούν.


My OpenGL window
ΗΛΕΚΤΡΙΚΟ ΠΕΔΙΟ
ΜΑΓΝΗΤΙΚΟ ΠΕΔΙΟ
My OpenGL window


My OpenGL window
My OpenGL window


My OpenGL window
My OpenGL window


My OpenGL window
My OpenGL window


My OpenGL window
My OpenGL window

8.3. Μελέτη αλληλεπίδρασης με εικονικό περιβάλλον (Επικοινωνία με jReve)

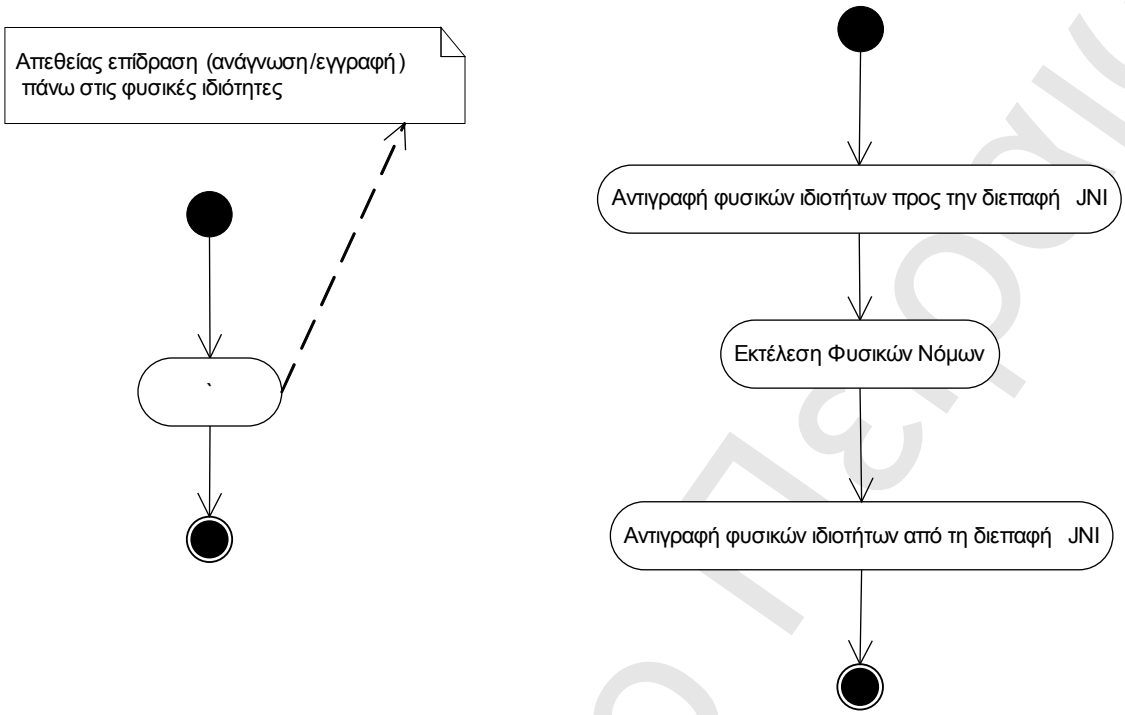
Θα συγγραφεί κώδικας για την προσαρμογή του Vesper3D στο εικονικό περιβάλλον jReve, και θα εκτελεσθούν σενάρια στο περιβάλλον αυτό για να δειχθεί η ικανότητα χρήσης του Vesper3D από ανεξάρτητο εικονικό περιβάλλον.

8.3.1. Βασικά για την διαγλωσσική επικοινωνία

Για την επικοινωνία του Vesper3D με το jReve απαιτούνται κάποια βασικά βήματα. Κατ' αρχάς απαιτείται μια διεπαφή η οποία επιτελεί κάποιες βασικές λειτουργίες βασισμένη σε κάποιες παραδοχές.

- Στην πλευρά του καθεαυτώ Vesper3D (PhysicsEngine) δεν απαιτείται απολύτως καμία αλλαγή ή ειδική version.
- Στην ενδιάμεση διεπαφή απαιτείται η σύνταξη μιας τάξης μέσω της τεχνολογίας JNI (Java Native Interface), η οποία θα εκθέσει σε κώδικα Java τις μεθόδους της κυρίως μηχανής του Vesper3D (v3d::PhysicsEngine). Δηλαδή θα ορισθούν στην πλευρά του JNI μέθοδοι που θα «τυλίγουν» τις μεθόδους του PhysicsEngine ώστε να μπορούν να κληθούν από τον κώδικα Java. (Initialize, AddPhysicalProperty, LoadPhysicsLaw, Calculate)
- Απαιτείται η χρήση δομών δεδομένων οι οποίες να επιτρέπουν την χρήση τόσο από περιβάλλον Java όσο και από native κώδικα. Η δομές αυτές υπάρχουν στην γλώσσα Java με την ονομασία DirectBuffer. Οι δομές αυτές μπορούν να χρησιμοποιούνται και από τις δυο πλευρές (PhysicsEngine, Java). Οπουδήποτε ο κώδικας του Vesper3D θα χρησιμοποιούσε πίνακες σε float ή int, ο κώδικας Java περνά δείκτες στις δομές αυτές, και οι φυσικοί νόμοι τις χρησιμοποιούν σαν να ήταν απλοί πίνακες.
- Απαιτείται, στην πλευρά της Java μια τάξη η οποία καλεί τις μεθόδους JNI. Με τον τρόπο αυτό έχουμε μεταφέρει ουσιαστικά το API του v3d::PhysicEngine σε Java.
- Στην πλευρά του jReve, απαιτείται η χρήση ενός Validator. Το αντικείμενο αυτό αναλαμβάνει την ανά καρέ συμπεριφορά, δηλαδή:
    - Τη μεταφορά των φυσικών ιδιοτήτων από το jReve προς τα DirectBuffers της διεπαφής με το Vesper3D
    - Την ανά καρέ κλήση στο Vesper3D με την παράμετρο του χρόνου
    - Την ανάκτηση των φυσικών ιδιοτήτων που χρειάζεται να γνωρίζει το Vesper3D προς το jReve (positions) ώστε να εμφανίσει τα σώματα στις νέες τους θέσεις.

Η διαδικασία συνεπώς συνοψίζεται στο παρακάτω διάγραμμα:



8.3.2. Βασικό παράδειγμα

Ορίζονται στο jReve δυο σώματα και ένα ελατήριο.
Για τα σώματα αυτά ορίζονται οι φυσικές ιδιότητες Μάζα, Ταχύτητα, Θέση, Σκληρότητα Ελατηρίου και Απόσβεση.
Στην ενδιάμεση διεπαφή ορίζονται άνευ δυνατότητα αλλαγής οι ιδιότητες Δύναμη (δεν εκφράζει εδώ κάτι απαιτούμενο), Εξωτερική Δύναμη (δεν χρησιμοποιείται σε αυτό το παράδειγμα εξωτερική δύναμη), Επιτάχυνση της Βαρύτητας(0,0,-9.81m/sec^2).
Σε κάθε παράδειγμα θέτονται διαφορετικές τιμές των ιδιοτήτων, και φορτώνονται διαφορετικοί φυσικοί νόμοι.

8.3.3. Ίσες μάζες χωρίς αρχική ταχύτητα υπό την επίδραση ελατηρίου με απόσβεση

Αρχικές θέσεις	(-10,5,0)	(10,5,0)
Ταχύτητες	(0,0,0)	(0,0,0)
Μάζες	(1)	(1)
Σκληρότητα ελατηρίου	(100)	
Απόσβεση	(10)	

Οι μάζες ταλαντώνονται υπό την επίδραση του ελατηρίου και μετά από κάποια δευτερόλεπτα ηρεμούν. (Η συνολική ενέργεια μηδενίζεται λόγω της απόσβεσης)

8.3.4. Ίσες μάζες με αρχική συμμετρική ταχύτητα υπό την επίδραση ελατηρίου με απόσβεση

Αρχικές θέσεις	(-10,5,0)	(10,5,0)

Ταχύτητες	**(0,-10,0)**	**(0,10,0)**
Μάζες	(1)	(1)
Σκληρότητα ελατηρίου	(100)	
Απόσβεση	(10)	

Οι μάζες ταλαντώνονται υπό την επίδραση του ελατηρίου και ταυτόχρονα εκτελούν περιστροφική κίνηση. Μετά από κάποια δευτερόλεπτα η ταλάντωση αποσβεννύεται και οι μάζες περιστρέφονται η μία γύρω από την άλλη κυκλικά χωρίς συνολική μετατόπιση (διατήρηση ενέργειας στο τμήμα της κίνησης που δεν υφίσταται τριβές).

8.3.5. Ίσες μάζες χωρίς αρχική ταχύτητα υπό την επίδραση ελατηρίου με απόσβεση

Αρχικές θέσεις	(-10,5,0)	(10,5,0)
Ταχύτητες	**(0,0,0)**	(0,10,0)
Μάζες	(1)	(1)
Σκληρότητα ελατηρίου	(100)	
Απόσβεση	(10)	

Οι μάζες ταλαντώνονται υπό την επίδραση του ελατηρίου και ταυτόχρονα εκτελούν περιστροφική κίνηση και το σύστημα μετατοπίζεται προς το θετικό x. Μετά από κάποια δευτερόλεπτα η ταλάντωση αποσβεννύεται και οι μάζες περιστρέφονται η μία γύρω από την άλλη κυκλικά και το κέντρο μάζας του συστήματος εκτελεί ευθύγραμμη ομαλή κίνηση με ταχύτητα u=(0,5,0) (διατήρηση ενέργειας στο τμήμα της κίνησης που δεν υφίσταται τριβές).

8.3.6. Άνισες μάζες με αρχική ασύμμετρη ταχύτητα υπό την επίδραση ελατηρίου με απόσβεση

Αρχικές θέσεις	(-10,5,0)	(10,5,0)
Ταχύτητες	**(0,3,0)**	**(0,00,0)**
Μάζες	**(10)**	(1)
Σκληρότητα ελατηρίου	(100)	
Απόσβεση	(10)	

Οι μάζες ταλαντώνονται υπό την επίδραση του ελατηρίου. Η μπλε ταλαντώνεται και περιστρέφεται γύρω από την κόκκινη και η κόκκινη ταλαντώνεται και περιστρέφεται γύρω από την μπλε. Η μπλε εκτελεί ταλαντώσεις μεγαλύτερου πλάτους από την κόκκινη. Το κέντρο μάζας ταλαντώνεται και μετακινείται. Μετά από κάποια δευτερόλεπτα η ταλάντωση αποσβεννύεται και το σύστημα μετατοπίζεται ενώ οι μάζες περιστρέφονται η μια γύρω από την άλλη με την μπλε να κινείται περισσότερο από την κόκκινη.

8.3.7. Μάζες εκτελούν βαλλιστική κίνηση υπό την επίδραση βαρυτικού πεδίου

Αρχικές θέσεις	(-10,5,0)	(10,5,0)
Ταχύτητες	**(10,0,30)**	**(-20,0,20)**
Μάζες	**(1)**	**(1)**
Βαρυτικό πεδίο	**(0,0,-9.81)**	

Οι μάζες εκτελούν παραβολικές τροχιές ανεξάρτητα μεταξύ τους με βάση την κλασσική βαλλιστική.

8.3.8. Ίσες μάζες με αρχική ταχύτητα προς τα άνω υπό την επίδραση ελατηρίου με απόσβεση και βαρύτητας

Αρχικές θέσεις	(-10,5,0)	(10,5,0)
Ταχύτητες	**(0,0,15)**	**(0,0,15)**
Μάζες	(1)	(1)
Σκληρότητα ελατηρίου	(100)	
Απόσβεση	(10)	
Βαρυτικό πεδίο	(0,0,-9.81)	

Οι μάζες εκτοξεύονται προς τα άνω, ενώ ταυτόχρονα ταλαντώνονται υπό την επίδραση του ελατηρίου. Όλο το σύστημα εκτελεί βολή. Μετά από κάποια δευτερόλεπτα η ταλάντωση σταματά ενώ το σύστημα πέφτει.

8.3.9. Ίσες μάζες με αρχική ταχύτητα προς τα άνω υπό την επίδραση ελατηρίου με απόσβεση και βαρύτητας. Μείωση σκληρότητας ελατηρίου.

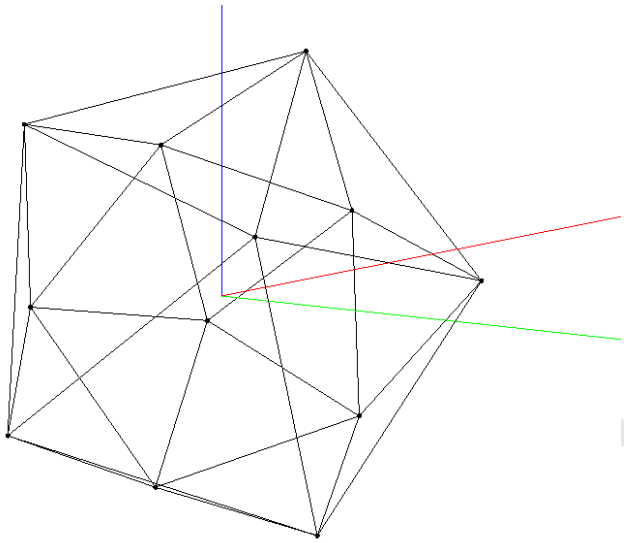
Αρχικές θέσεις	(-10,5,0)	(10,5,0)
Ταχύτητες	(0,0,15)	(0,0,15)
Μάζες	(1)	(1)
Σκληρότητα ελατηρίου	**(30)**	
Απόσβεση	(10)	
Βαρυτικό πεδίο	(0,0,-9.81)	

Ίδια συμπεριφορά με το προηγούμενο παράδειγμα. Μείωση της συχνότητας ταλάντωσης. Μείωση του χρόνου στον οποίο αποσβεννύεται η ταλάντωση.

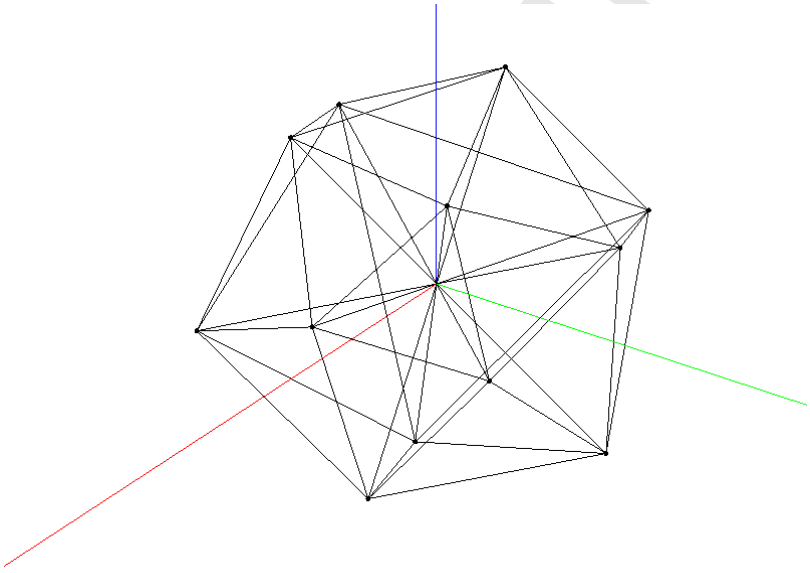
8.4. Μηχανική παραμορφώσιμου σώματος με δομή – Ανίχνευση συγκρούσεων

Το παράδειγμα αυτό χρησιμοποιεί αρχικά τους ίδιους νόμους με τον ιστό αράχνης. Υλοποιείται επιπροσθέτως ένας φυσικός νόμος ο οποίος εκτελεί ανίχνευση και αντιμετώπιση συγκρούσεων.
Ο νόμος αυτός υλοποιείται σχετικώς απλοϊκά, η σχεδίαση όμως του Vesper3D του επιτρέπει σε δεύτερο χρόνο να επανασχεδιαστεί και βελτιστοποιηθεί εφ' όσον επιθυμείται.
Ο φυσικός νόμος αυτός λειτουργεί αποκλειστικά με ανίχνευση συγκρούσεων επιφάνειας προς επιφάνεια, και μοντελοποιείται με επιφανειακά ελατήρια, και διείσδυση σώματος σε σώμα για την εφαρμογή δυνάμεων. Δηλαδή, η δύναμη διαμοιράζεται στις 3 κορυφές της κάθε επιφάνειας που διεισδύει στην άλλη επιφάνεια και είναι ανάλογη της διείσδυσης των επιφανειών, οι οποίες διαθέτουν επίσης πάχος (thickness), σκληρότητα (hardness) (σκληρότητα hooke ανά μονάδα επιφανείας), και επιφανειακή απόσβεση (surfaceDamping) (ιξώδης απόσβεση ανά μονάδα επιφανείας).
Οι φυσικοί νόμοι αυτοί μας επιτρέπουν να προσομοιάσουμε όχι μόνο ένα σώμα με εσωτερική δομή, αλλά και μια επιφάνεια πάνω στην οποία μπορεί να προσκρούσει το σώμα.
Θα ορισθεί ένα «σώμα» ως κανονικό εικοσάεδρο. Το σώμα αυτό δεν έχει υπόσταση όσον αφορά το Vesper3D – ορίζονται 11 μάζες (κορυφές) 100 κιλών, σε κατάλληλες θέσεις κανονικού εικοσαέδρου, 30 ελατήρια με κατάλληλο μήκος (~5.2 m) ως ακμές

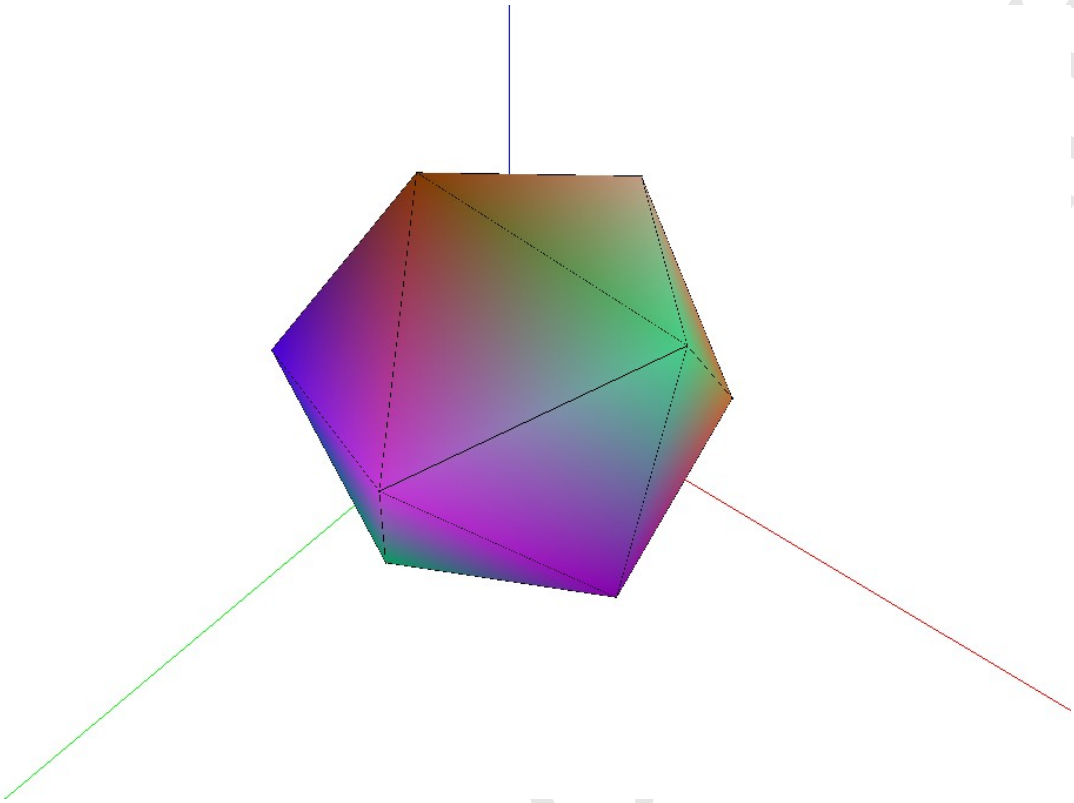
ανάμεσα στα σώματα τα οποία συγκρατούν, και 20 επιφάνειες (έδρες) οι οποίες θα χρησιμεύσουν για την ανίχνευση συγκρούσεων.



Στα περισσότερα παραδείγματα θα προστεθεί επίσης ένα 13ο σώμα στο κέντρο του εικοσαέδρου, με ελατήρια κατάλληλου μήκους και ίσης σκληρότητας με τα υπόλοιπα, με σκοπό να ενισχύσει την εσωτερική δομή του.



Το εικοσάεδρο «ενδύεται» με τις επιφάνειες, οι οποίες χρωματίζονται τυχαία για να φαίνεται ο όγκος του.

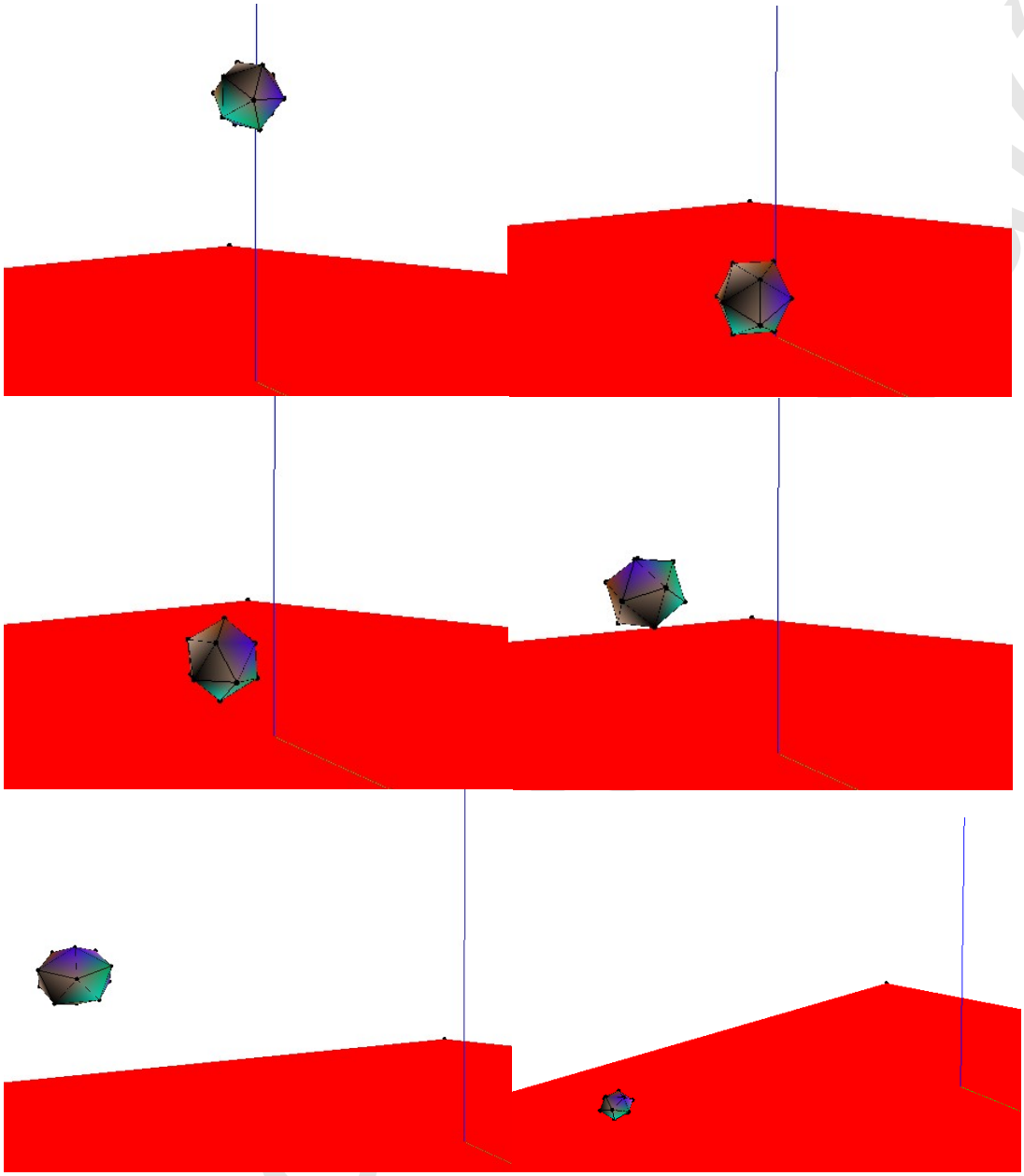


Τα παραδείγματα που θα ακολουθήσουν συνίστανται στην τοποθέτηση του εικοσαέδρου αυτού 100m από επίπεδο που ορίζεται από 3 ακλόνητες μάζες (m=0), και την εκτέλεση πτώσεις με διάφορες παραμέτρους για την μελέτη διαφορετικών συμπεριφορών.

8.4.1. Οριζόντιο επίπεδο – σκληρό δάπεδο – σκληρό σώμα

Σκληρότητα δαπέδου: 1,500,000
Σταθερά ελατηρίου ακμής εικοσαέδρου: 5,000,000

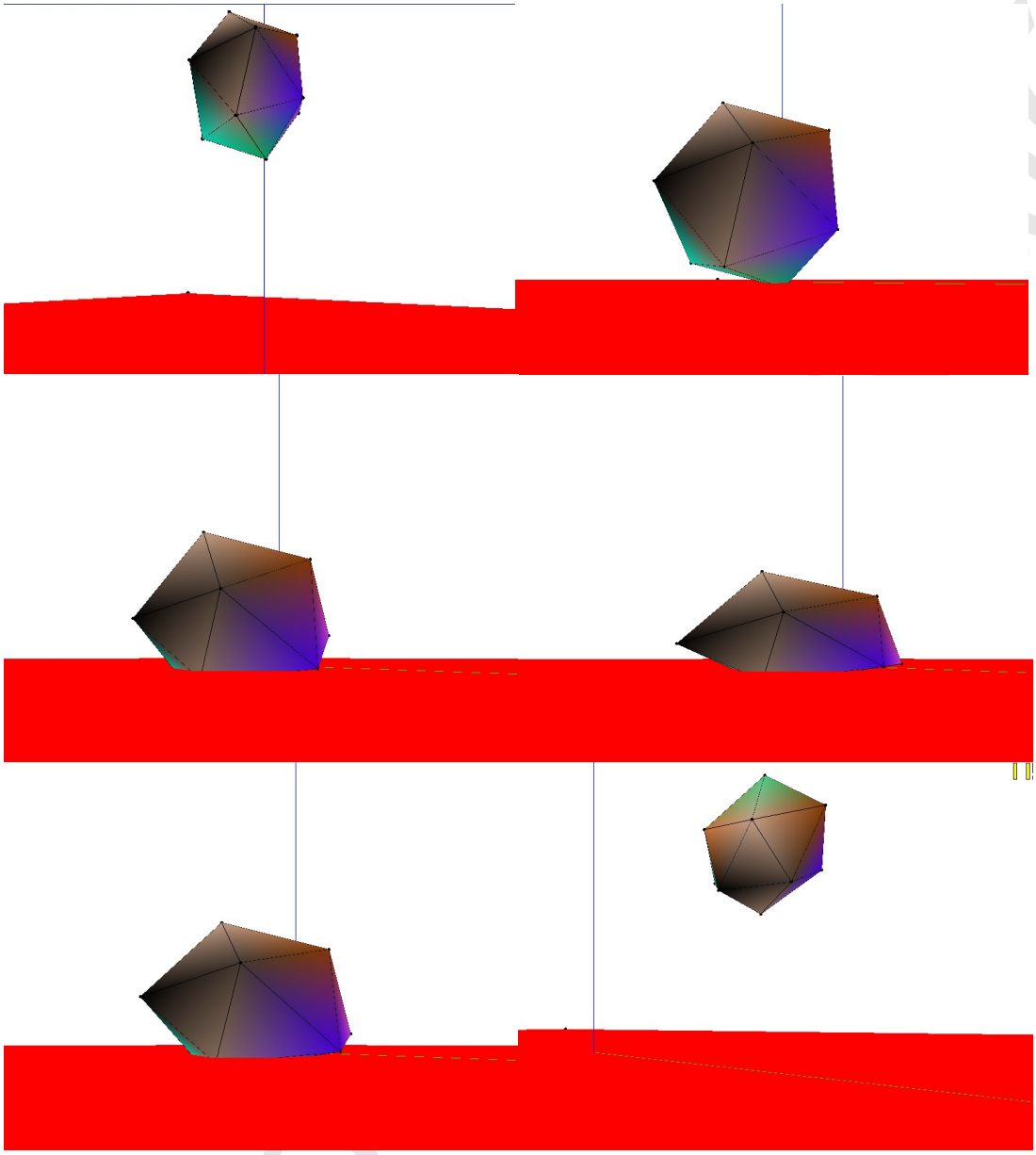
Συμπεριφορά μπάλας που αναπηδά



Το εικοσάεδρο υφίσταται μικρή παραμόρφωση και αναπηδά από το δάπεδο αρκετές φορές πριν ηρεμήσει πάνω σε κάποια από τις έδρες του.

8.4.2. Οριζόντιο επίπεδο – μέτρια σκληρό δάπεδο – μαλακό σώμα

Σκληρότητα δαπέδου: 100,000
Σταθερά ελατηρίου ακμής εικοσαέδρου: 50,000
Συμπεριφορά σαν «ζελέ».

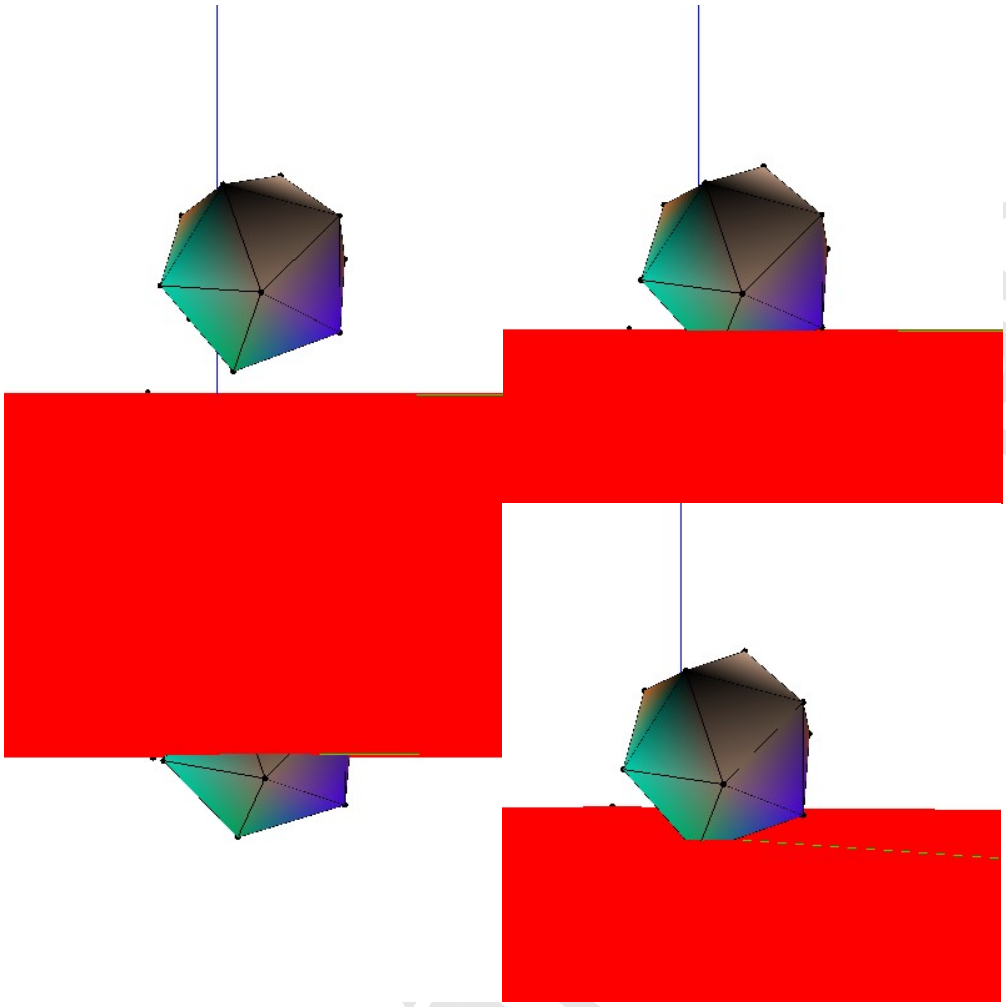


Το σώμα υφίσταται μεγάλη παραμόρφωση κατά την αναπήδηση και ταλαντώνεται ορατά.

8.4.3. Οριζόντιο επίπεδο – μαλακό δάπεδο – μέτρια σκληρό σώμα

Σκληρότητα δαπέδου: 5000
Σταθερά ελατηρίου ακμής εικοσαέδρου: 500,000

Το δάπεδο ομοιάζει σε συμπεριφορά με τεντωμένο λάστιχο

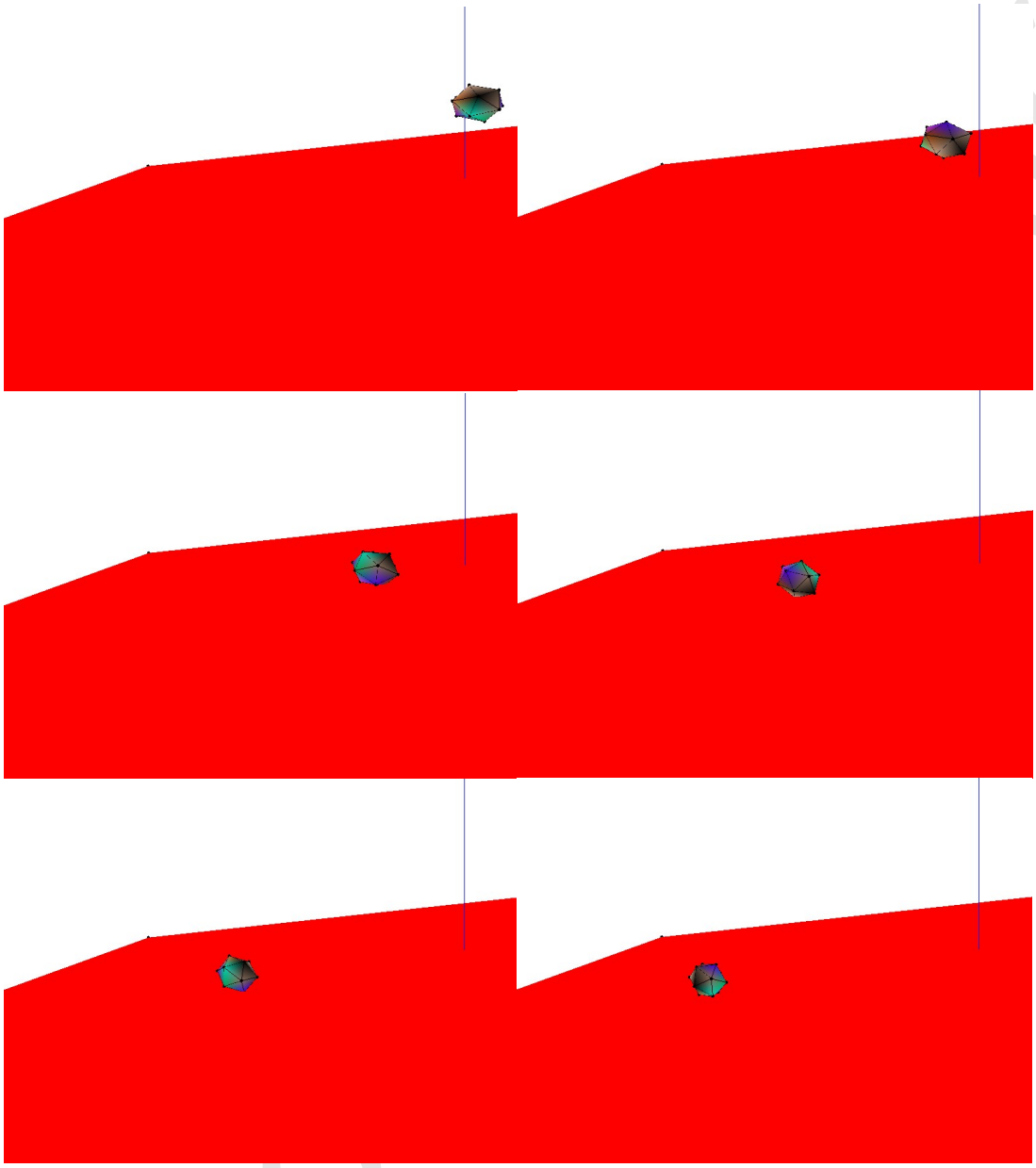


Το σώμα ηρεμεί με τμήμα του ορατά μέσα στην επιφάνεια, «βουλιάζει» ελαφρώς σε αυτήν.

8.4.4. Κεκλιμένο επίπεδο – μέτρια σκληρό δάπεδο – σκληρό σώμα – υψηλή τριβή

Σκληρότητα δαπέδου: 50,000
Σταθερά ελατηρίου ακμής εικοσαέδρου: 5,000,000
Συντελεστής τριβής εικοσαέδρου-δαπέδου: 0.5 * 0.5 = 0.25

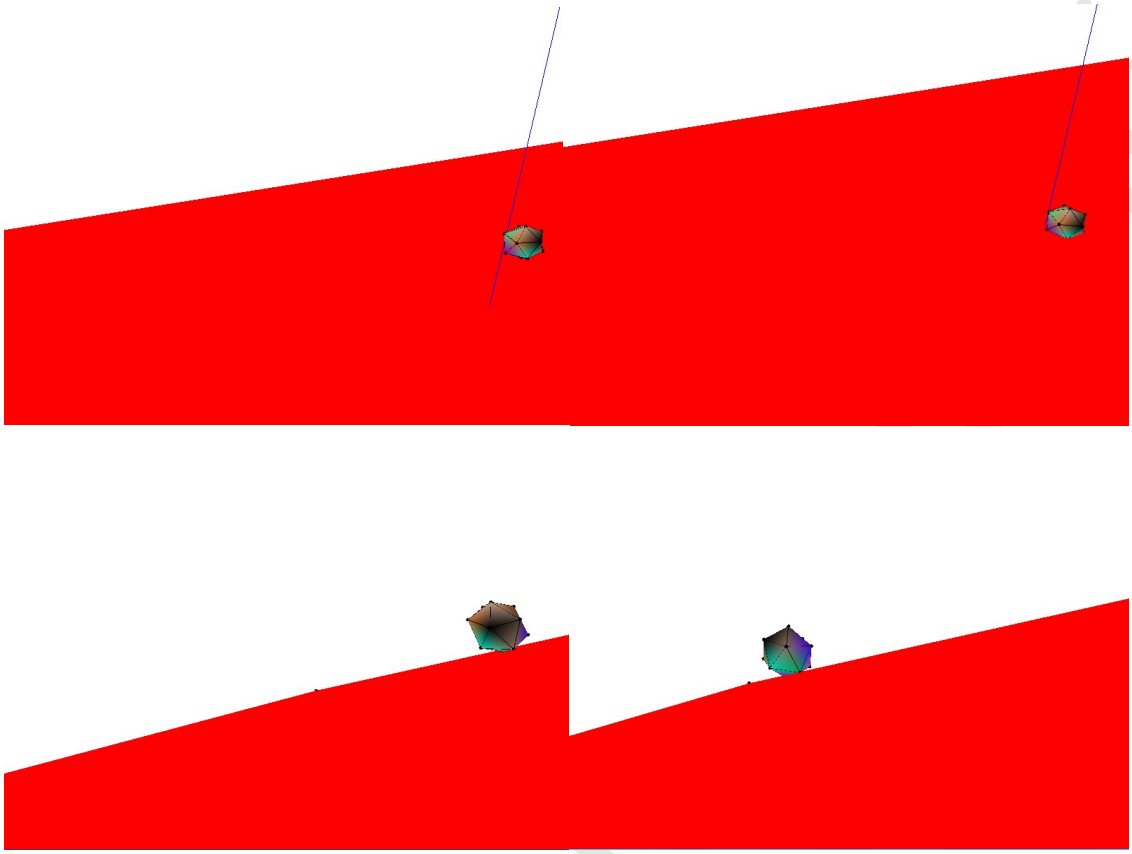
Το σώμα αναπηδά μερικές φορές και κατόπιν **κυλά** στο κεκλιμένο επίπεδο



8.1.1. Κεκλιμένο επίπεδο – μέτρια σκληρό δάπεδο – μέτρια σκληρό σώμα – χαμηλή τριβή

Σκληρότητα δαπέδου: 100,000
Σταθερά ελατηρίου ακμής εικοσαέδρου: 500,000
Συντελεστής τριβής εικοσαέδρου-δαπέδου: 0.5 * 0.01 = 0.005

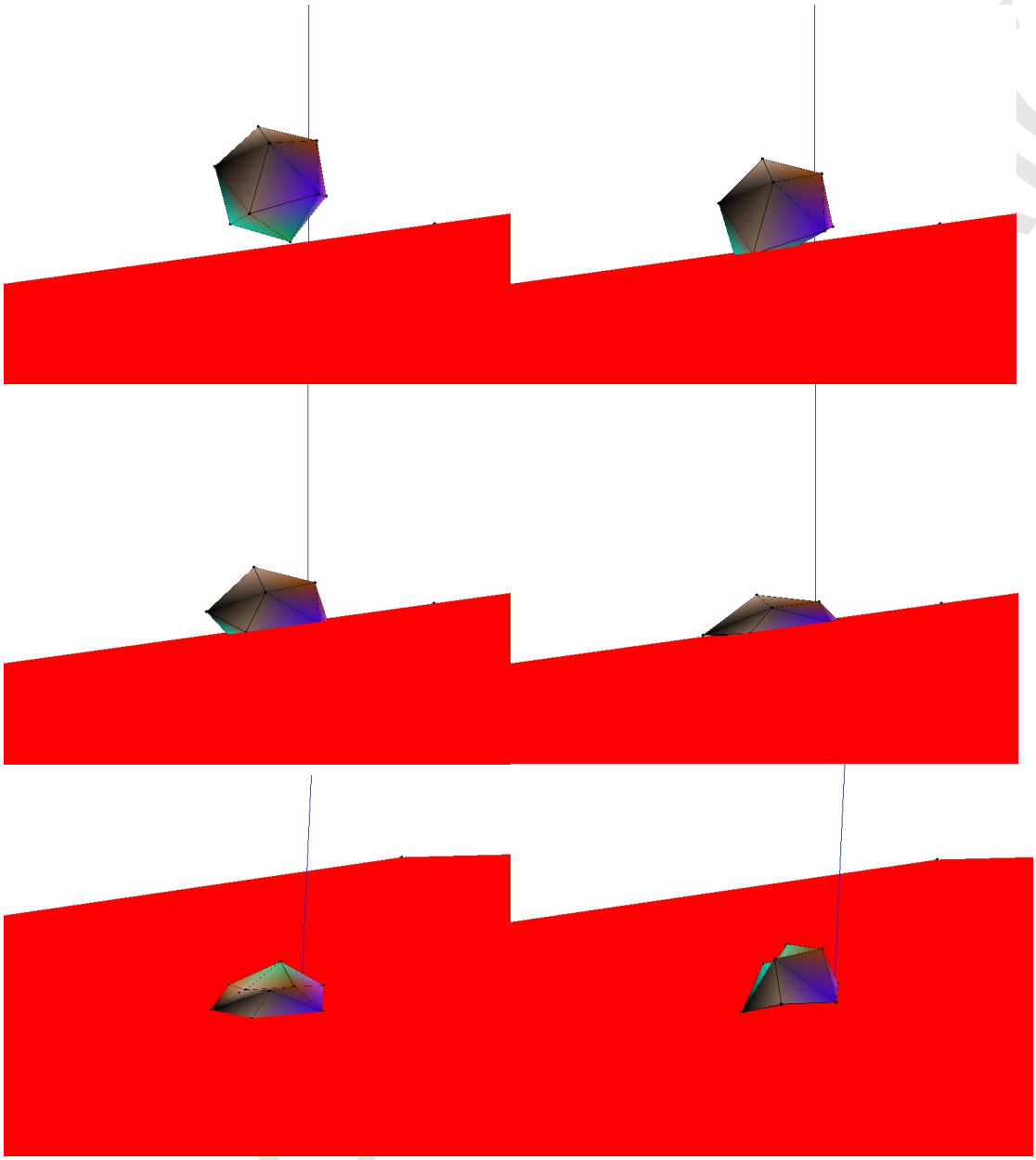
Το σώμα αναπηδά μερικές φορές και κατόπιν **σύρεται** στο κεκλιμένο επίπεδο



8.4.5. Κεκλιμένο επίπεδο – σκληρό δάπεδο – πολύ μαλακό σώμα – υψηλή τριβή

Σκληρότητα δαπέδου: 100,000
Σταθερά ελατηρίου ακμής εικοσαέδρου: 10,000

Το σώμα παραμορφώνεται μόνιμα από την κρούση με το δάπεδο. Η μισή του καμπυλότητα αναστρέφεται και παίρνει το σχήμα κενού ημισφαιρίου



Εδώ, ενώ μαθηματικά δεν έχουμε «καταστροφή» του σώματος (πρόκειται για ακριβώς τα ίδια σώματα με ακριβώς τους ίδιους συνδέσμους), η κρούση με το δάπεδο μετακίνησε τα ελατήρια πέρα από τα σημεία που ισορροπούσαν στο σχήμα εικοσαέδρου και αυτά πήραν πλέον νέο, μη κυρτό σχήμα.

8.5. Μελέτη απόδοσης - αλληλεπίδρασης με ειδικές τεχνολογίες (OpenCL)

Για τα παραδείγματα αυτά, ορίζεται ένα δεύτερο πρότυπο, και συντάσσεται για το Vesper3D μια οικογένεια φυσικών νόμων διαφορετική από αυτούς που χρησιμοποιήθηκαν έως τώρα.
Οι φυσικοί αυτοί νόμοι θα χρησιμοποιούν από το πρόγραμμα-πελάτη οι ίδιες φυσικές ιδιότητες που χρησιμοποιήθηκαν έως τώρα.

Οι φυσικοί νόμοι αυτοί δεν θα αποτελούνται από απλές αλγοριθμικές πράξεις μεταξύ διανυσμάτων τύπου v4f : θα χρησιμοποιούν την τεχνολογία **OpenCL** για να εκτελέσουν τους απαιτούμενους υπολογισμούς στην κάρτα γραφικών εκμεταλλευόμενοι την εξαιρετική ικανότητα παράλληλων υπολογισμών.
Για το παράδειγμα θα υλοποιήσουμε πρόγραμμα που υπολογίζει την κίνηση σωμάτων υπό την επίδραση των μεταξύ τους βαρυτικών δυνάμεων (*N-Body problem*).
Θα ισχύουν, κατά τα άλλα, όλες οι αρχές και παραδοχές που χρησιμοποιήσαμε έως τώρα.
Οι φυσικές ιδιότητες θα αποθηκεύονται όπως μέχρι τώρα: Οι βαθμωτές ιδιότητες σε μεταβλητές float, και οι διανυσματικές ιδιότητες σε τετράδες float με σειρά x,y,z,w. Δηλαδή, οι νόμοι αυτοί θα είναι συμβατοί με το βασικό πρότυπο που ορίσαμε.
Για την μελέτη αυτή θα συγκριθεί η απόδοση για 4 μεθόδους εκτέλεσης στις οποίες έγινε νωρίτερα μνεια.
Πρώτα, θα χρησιμοποιηθεί για τον υπολογισμό των δυνάμεων η βιβλιοθήκη διανυσμάτων με πράξεις SSE που έχει συγγραφεί και χρησιμοποιείται κατά κόρον στην παρούσα εργασία (NBodyGravity, NewtonsLawOfMotion).
Κατόπιν, θα χρησιμοποιηθούν για τον υπολογισμό των δυνάμεων και τον θέσεων νόμοι που χρησιμοποιεί το OpenCL (NBodyGravityOpenCL, NewtonsLawOfMotionOpenCL)
Τρίτον, θα δείξουμε ότι μπορούμε να εκμεταλλευτούμε τη σωστής δομής του Vesper3D, είναι δυνατή η ανάμειξη εντελώς ανεξάρτητων και διαφορετικής φύσεως μεταξύ τους τεχνολογιών, όπου θα εκτελούμε τον υπολογισμό N- δυνάμεων μέσω OpenCL μέσω του NBodyGravityOpenCL, και τον υπολογισμό των θέσεων μέσω του NewtonsLawOfMotion που χρησιμοποιήθηκε σε όλα τα προηγούμενα παραδείγματα
Τέλος, θα δείξουμε ότι ομοίως εκμεταλλευόμενοι τη σωστή δομή του Vesper3D, έχουμε μεγάλες δυνατότητες βελτιστοποίηση του κώδικα ώστε να επιτύχουμε εξαιρετική απόδοση στα παραδείγματα που τρέχουμε.

8.5.1. Γενικά για το OpenCL

Η τεχνολογία OpenCL είναι μια Διεπαφή Προγραμματισμού Εφαρμογών (*API, Application Programming Interface*) που επιτρέπει εξαιρετική παραλληλοποίηση υπολογισμών μέσω της λογικής SIMD (Single Instruction Multiple Data), όπου δηλαδή μια σειρά υπολογισμών επαναλαμβάνεται παράλληλα για «μεγάλο» όγκο δεδομένων. Η λογική αυτή είναι ιδιαίτερα κατάλληλη για υλοποιήσεις όπου μεγάλο ποσοστό των υπολογισμών είναι όμοιο αλλά εφαρμόζεται σε διαφορετικά δεδομένα εισόδου, οπότε μπορεί να εκτελείται ανεξάρτητα για κάθε ομάδα δεδομένων εισόδου.
Για το παρόν παράδειγμα υπολογίζονται δυνάμεις «για κάθε» σωματίδιο, και με βάση αυτές ταχύτητες και θέσεις επίσης «για κάθε» σωματίδιο. Οι υπολογισμοί αυτοί μπορούν να παραλληλοποιηθούν.

Κατά την αρχικοποίησή της, η τεχνολογία αυτή απαιτεί ιδιαίτερες παραδοχές και δομές μνήμης. Σε αδρές γραμμές η διαδικασία που ακολουθείται είναι η παρακάτω:

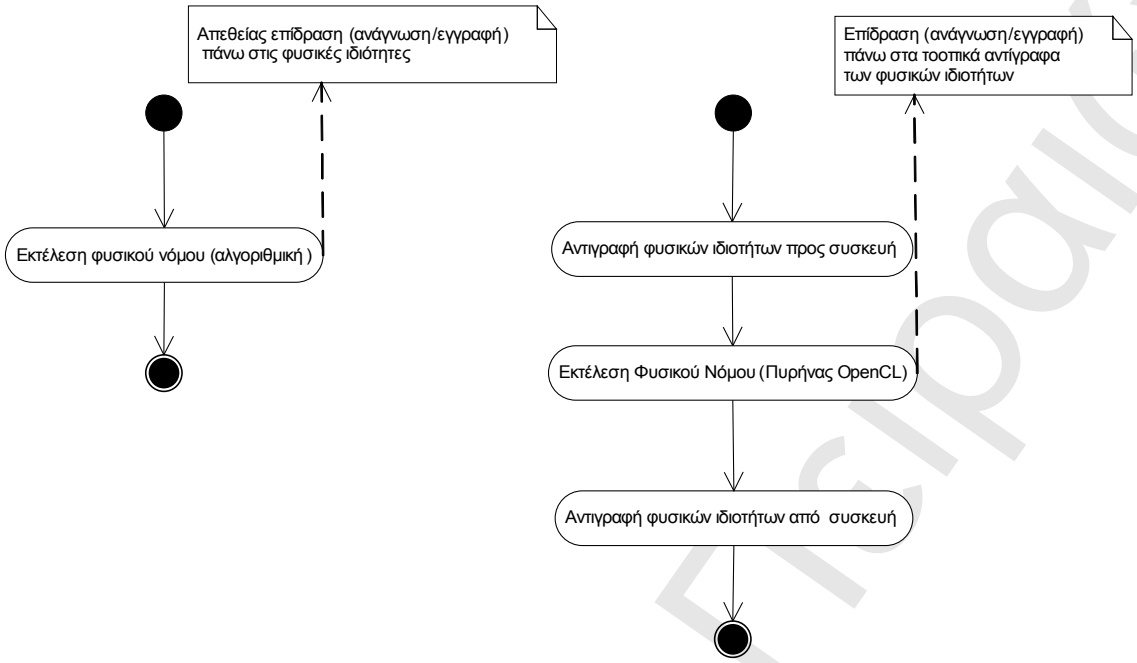
- Εύρεση και επιλογή / ορισμός κατάλληλης πλατφόρμας OpenCL (cl::Platform) από το σύστημα (δηλαδή, κατάλληλων συσκευών κάρτας γραφικών και οδηγοί (*drivers*) που υποστηρίζουν την τεχνολογία OpenCL.

- Επιλογή κατάλληλης συσκευής (cl::Device) για τη συγκεκριμένη πλατφόρμα (κάρτα γραφικών, επεξεργαστής ή άλλη συσκευή που υποστηρίζει την επιλεγμένη πλατφόρμα)
- Ορισμός κατάλληλου αντικειμένου «πλαισίου» OpenCL(cl::Context)
- Συγγραφή των προγραμμάτων **πυρήνα** (kernel) - σε εμάς αυτοί είναι οι φυσικοί νόμοι) στη γλώσσα πυρηνών OpenCL (παραλλαγή της κλασσικής γλώσσας C), όπου χρησιμοποιούνται ως παράμετροι απλές τιμές ή δείκτες σε πίνακες. Οι πυρήνες είναι οι αλγόριθμοι που θα εφαρμοστούν σε κάθε ομάδα δεδομένων εισόδου
- Ορισμός αντικειμένου Πυρήνα (cl::Kernel) για το συγκεκριμένο πλάισιο εκτέλεσης και συσκευή, για την συγκεκριμένη συνάρτηση που θα κληθεί
- Μεταγλώττιση των πυρήνων κατά την εκτέλεση του προγράμματος (runtime) για τη συγκεκριμένη πλατφόρμα και συσκευή
- Ορισμός αντικειμένου Ουράς Εντολών (cl::CommandQueue) για το συγκεκριμένο πλάισιο εκτέλεσης και συσκευή.

Κατά την εκτέλεσή της, η τεχνολογία αυτή απαιτεί τα παρακάτω βήματα για τον κάθε πυρήνα (εδώ, πυρήνας είναι ο κάθε φυσικός νόμος):

- Ορισμός ιδιαίτερων αντικειμένων αποθήκευσης των απαιτούμενων δεδομένων (cl::Buffer) που θα λειτουργήσουν ως παράμετροι εισόδου και εξόδου στους πυρήνες
- Αντιστοίχιση των δομών δεδομένων και κάθε άλλης πληροφορίας που απαιτείται ως παράμετρος στον συγκεκριμένο πυρήνα (Kernel::SetArg)
- Αντιγραφή των απαιτούμενων δεδομένων φυσικών ιδιοτήτων που θα αναγνωστούν από τον πυρήνα (εδώ, πίνακες από float που αφορούν το συγκεκριμένο φυσικό νόμο) από τη μνήμη του υπολογιστή στην μνήμη της συσκευής μέσω κλήσης στην ουρά εκτέλεσης (CommandQueue::EnqueueWriteBuffer)
- Προσθήκη στην ουρά εκτέλεσης μιας εκτέλεσης του συγκεκριμένου πυρήνα (CommandQueue::EnqueueNDRangeKernel)
- Αντιγραφή των απαιτούμενων δεδομένων φυσικών ιδιοτήτων που ο πυρήνας έχει αλλάξει από τη μνήμη της συσκευής στη μνήμη του υπολογιστή (CommandQueue::EnqueueReadBuffer)

Εφ' όσον εκτελεσθούν σε κάθε καρέ, για κάθε φυσικό νόμο, με τη σωστή σειρά, τα παραπάνω βήματα, ουσιαστικά αναμένουμε την σωστή συμπεριφορά που είχαμε πριν. Για να καταλάβουμε καλύτερα τα επιπρόσθετα βήματα, ας συγκρίνουμε τα παρακάτω UML διαγράμματα δραστηριότητας:

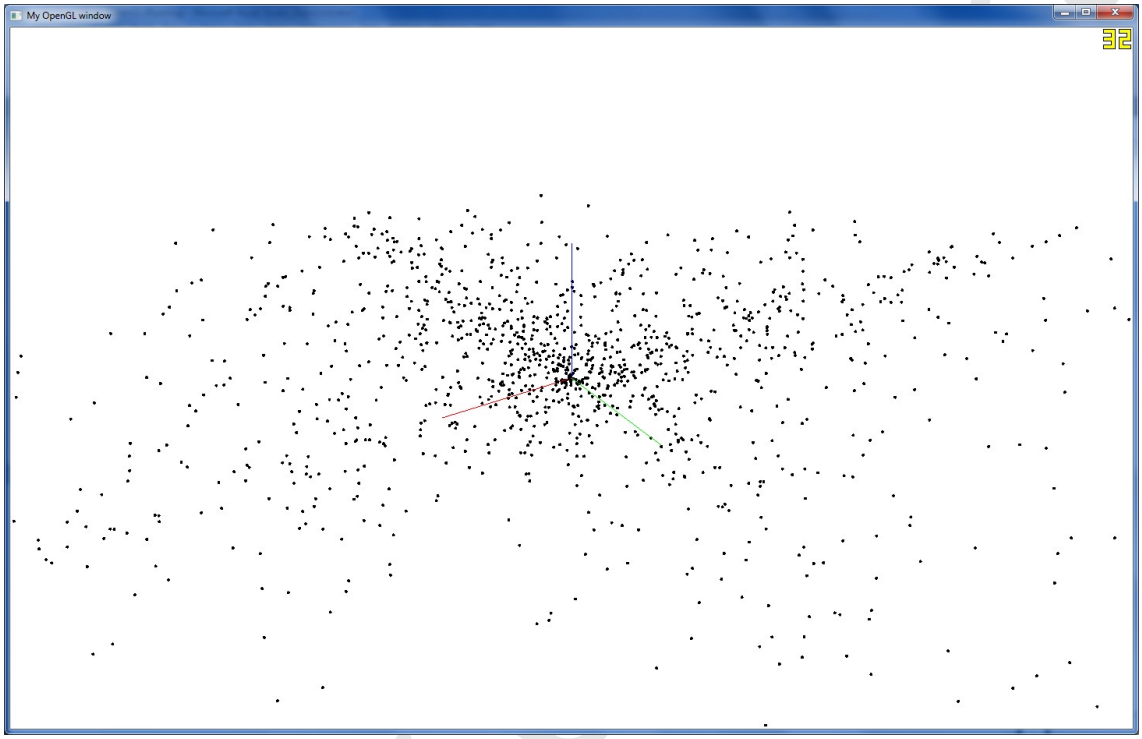


Ως αναφέρθηκε, στο παρόν παράδειγμα θα υλοποιηθεί παράδειγμα βαρυτικής αλληλεπίδρασης N- Σωμάτων, δηλαδή τυχαίος αριθμός σωμάτων που ανά δυο αλληλεπιδρούν με δυνάμεις βαρύτητας.
Το πρόβλημα αυτό στη βασική αλγοριθμική του μορφή είναι τετραγωνικής πολυπλοκότητας $O(n^2)$, και συνεπώς έχει νόημα η προσπάθεια βελτιστοποίησης της απόδοσής του.
Οι δομές των φυσικών ιδιοτήτων στην κάρτα γραφικών δέον να επαναχρησιμοποιούνται ανάμεσα στους φυσικούς νόμους. Για το σκοπό αυτό, θα χρησιμοποιηθεί συγκεκριμένη ονοματολογία τους, και θα αποθηκευθούν ως «φυσικές ιδιότητες» ώστε να μπορούν να επικοινωνηθούν ανάμεσα στους φυσικούς νόμους. Δηλαδή, το cl::Buffer που θα ορίζει την περιοχή μνήμης της κάρτας γραφικών που θα περιέχει το αντίγραφο του πίνακα της φυσικής ιδιότητας “mass”, θα ορισθεί κατά την αρχικοποίηση του πρώτου νόμου που θα το απαιτεί, και θα αποθηκευθεί στον κεντρικό PhysicalPropertyProvider με όνομα “massClBuffer” ώστε να χρησιμοποιηθεί από κάθε τυχόν επόμενο φυσικό νόμο που μπορεί να το απαιτήσει. Ομοίως και για τις υπόλοιπες φυσικές ιδιότητες.

Για το παράδειγμα αυτό, λόγω της φύσης του, δεν θα χρησιμοποιηθεί το ObjectProcessorFile για την ανάγνωση σωμάτων από αρχείο. Επειδή για να δείξουμε τα πλεονεκτήματα της χρήσης του OpenCL θα απαιτηθεί μεγάλος αριθμός σωμάτων, αυτά θα δημιουργηθούν μέσω κώδικα από το συγκεκριμένο σενάριο. Το σενάριο αυτό θα τοποθετεί σώματα σε τυχαίες θέσεις γύρω από κεντρικό σώμα μεγαλύτερης μάζας και με κατάλληλες υπολογιζόμενες τροχιακές ταχύτητες, ώστε να παραχθεί μια ενδιαφέρουσα εικόνα που μοιάζει με ένα γαλαξία όπου ήλιοι περιστρέφονται γύρω από τον πυρήνα του γαλαξία.
Στο γαλαξία θα τοποθετηθούν 1,536 ήλιοι. Η ποσότητα των υπολογισμών συνεπώς είναι οι συνδυασμοί 1,536 ανά 2, ήτοι ο υπολογισμός 1,178,880 ζευγών δυνάμεων.

8.5.2. Εκτέλεση με διανυσματικές βιβλιοθήκες

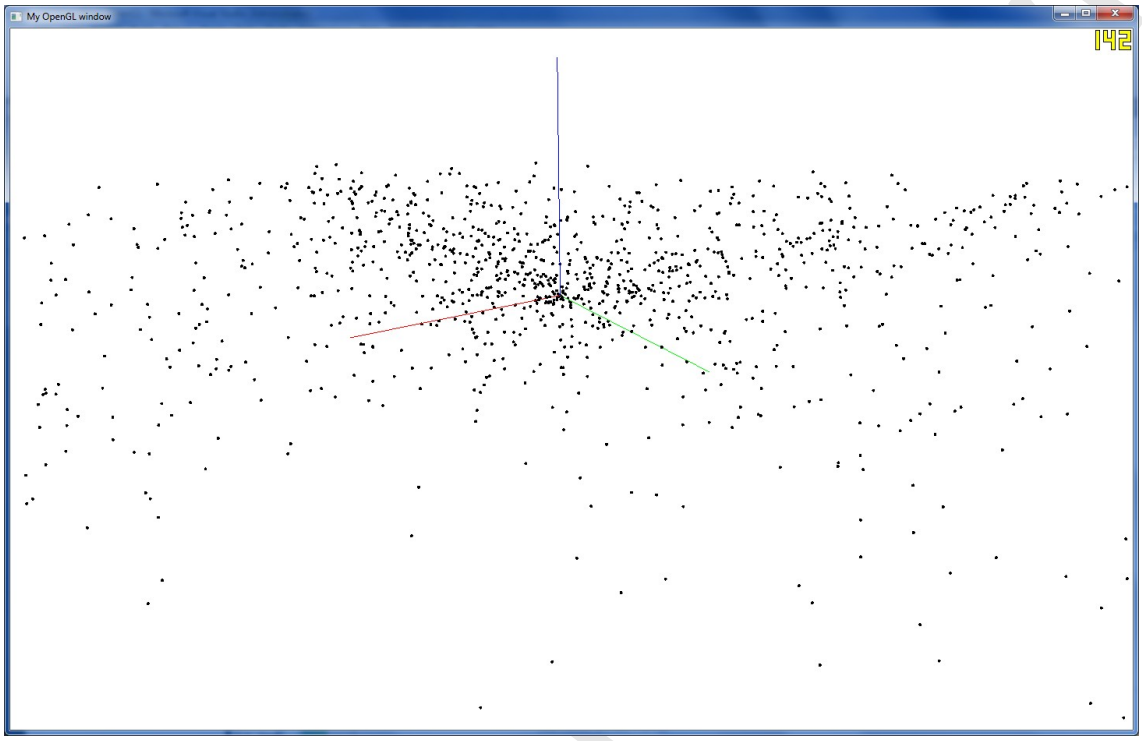
Το σενάριο αρχικά θα το τρέξουμε με απλό αλγοριθμικό νόμο υπολογισμού n-σωμάτων (NBodyGravity) με τον κλασσικό διανυσματικό νόμο της νευτώνειας κίνησης (NewtonsLawOfMotion).



Για το σενάριο αυτό, με τους κλασσικούς διανυσματικούς υπολογισμούς παίρνουμε απόδοση 32 καρέ ανά δευτερόλεπτο (fps) στο σύστημα 1 και 29 fps στο σύστημα 2.

8.5.3. Εκτέλεση με OpenCL

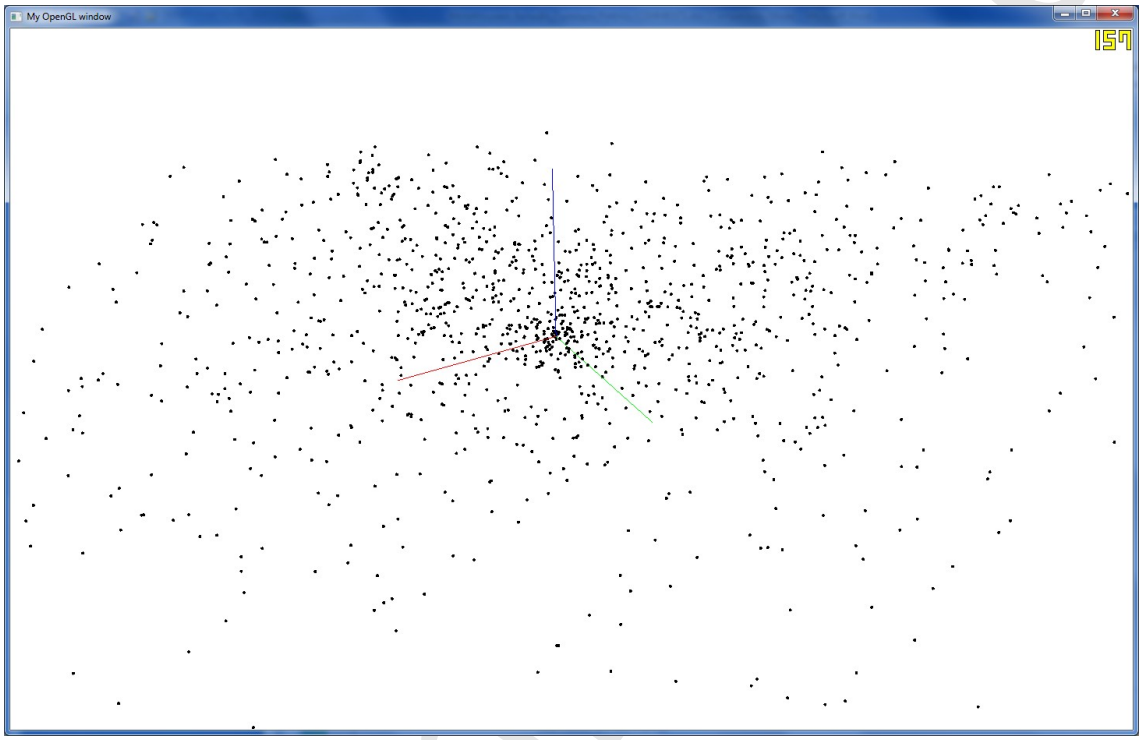
Θα το τρέξουμε το σενάριο των 1536 ήλιων με τον νόμο n-σωμάτων που χρησιμοποιεί την υλοποίηση OpenCL (NBodyGravityOpenCL).



Στο σενάριο αυτό μετράμε 142fps. Πρόκειται για εντυπωσιακότατη αύξηση της απόδοσης 443%, δηλαδή πάνω από τετραπλασιασμός της αποδόσεως του συστήματος.

8.5.4. Εκτέλεση με συνεργασία τεχνολογιών

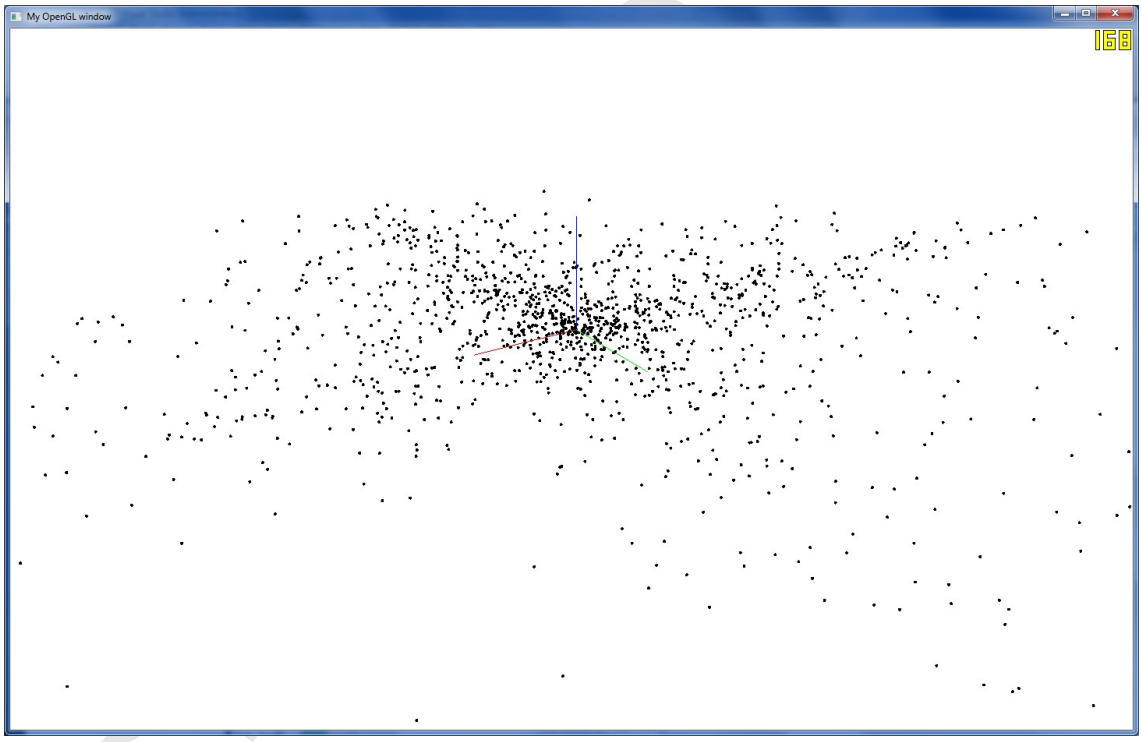
Επόμενο σενάριο είναι η χρήση συνδυασμού τεχνολογιών. Εδώ θα υπολογισθούν οι δυνάμεις N-σωμάτων μέσω του OpenCL και ο υπολογισμός των μετατοπίσεων μέσω του NewtonsLawOfMotion με τις διανυσματικές βιβλιοθήκες. Κύριος σκοπός του σεναρίου αυτού είναι να αποδείξει την ευελιξία του συστήματος Vesper3D στην ενορχήστρωση διαφορετικής φύσεως συστημάτων.



Στο σενάριο αυτό παρατηρούμε τη μέγιστη απόδοση μέχρι στιγμής: 157 fps (500% σε σχέση με το απλό διανυσματικό).
Χωρίς να μπούμε σε ιδιαίτερες λεπτομέρειες υλοποίησης, μια ταχεία ανάλυση των γεγονότων που εκτελούνται σε κάθε κύκλο, μας οδηγεί στην πιθανή υπόθεση η αύξηση απόδοσης οφείλεται στο γεγονός ότι ο νόμος της νευτώνειας κίνησης με τις διανυσματικές βιβλιοθήκες πραγματοποιεί πολύ μικρό αριθμό υπολογισμών συγκριτικά με τις δυνατότητες ενός σύγχρονου επεξεργαστή, πολλώ δε περισσότερο σε σχέση με την κάρτα γραφικών (έχει γραμμική πολυπλοκότητα O(n)).
Έτσι, η επιβάρυνση που προσθέτει η αντιγραφή των ιδιοτήτων από και προς την κάρτα γραφικών πριν και μετά από κάθε εκτέλεσή του, και η χρήση της ουράς εκτέλεσης προσθέτει τελικά επιβάρυνση μεγαλύτερη από τον συνολικό χρόνο εκτέλεσης του NewtonsLawOfMotion ανά καρέ.

8.5.5. Εκτέλεση με OpenCL με βελτιστοποίηση μεταφοράς δεδομένων

Θα αποπειραθούμε να βελτιστοποιήσουμε πλήρως την απόδοση του συστήματος με το OpenCL. Για να το επιτύχουμε θα ελαχιστοποιήσουμε τη μεταφορά δεδομένων από και προς την κάρτα γραφικών με βάση αυτά που αναφέραμε παραπάνω.
Δηλαδή, θα θεωρήσουμε ότι μεταφορά δεδομένων προς το πρόγραμμα πελάτη θα γίνεται μόνο μέσω της φυσικής ιδιότητας positions και δεν θα ανανεώνονται οι φυσικές ιδιότητες ταχύτητας, μάζας και δυνάμεων, αφού ούτως η άλλως δεν χρησιμοποιούνται από το πρόγραμμα πελάτη. Επίσης δεν θα γίνεται καμία επιπρόσθετη μεταφορά δεδομένων από το πρόγραμμα-πελάτη στη μηχανή.
Συνεπώς, αφαιρούμε οπό τον κώδικα των φυσικών νόμων τα τμήματα κώδικα που αφορούν στην αντιγραφή δεδομένων από και προς τους πίνακες φυσικών ιδιοτήτων, με εξαίρεση τον πίνακα positions : Η μοναδική αντιγραφή που λαμβάνει χώρα γίνεται από το cl:buffer προς τον πίνακα positions. Η παραδοχή αυτή περιορίζει το πρόγραμμα πελάτη στην πρόσβαση σε ανανεωμένες τιμές μόνο για τη θέση, και επίσης μας περιορίζει στην χρήση αποκλειστικά νόμων που χρησιμοποιούν το OpenCL. Όμως, αν τα προηγούμενα συμπεράσματά μας είναι σωστά, αναμένουμε να επιτύχουμε εξαιρετική απόδοση.



Στο σενάριο αυτό μετράμε 168 fps στο σύστημα 1. Δηλαδή, πρόκειται για τη μέγιστη αύξηση απόδοσης που επιτύχαμε σε σχέση με το σύστημα αναφοράς: 522%, δηλαδή πάνω από πενταπλάσιο από την απόδοση του αρχικού συστήματος, και αύξηση απόδοσης στο 118.5%, δηλαδή 18.5% καλύτερη απόδοση σε σχέση με την αρχική υλοποίηση OpenCL.

9. Συμπεράσματα και αξιολόγηση σχεδιασμού και υλοποίησης του Vesper3D

9.1. Αξιολόγηση

Για να αξιολογηθεί η σχεδιασμένη και υλοποιημένη εργασία, μπορούμε να συγκρίνουμε τους σκοπούς που αναφέρθηκαν στο κεφάλαιο 3 με την υλοποιημένη στα κεφάλαια 5 και 6 λύση, και να μελετήσουμε τα αποτελέσματα των σεναρίων του κεφαλαίου 8.

α) Γενικότητα φυσικών νόμων:

Δόθηκε η δυνατότητα σε έναν ανεξάρτητο προγραμματιστή να γράψει σε κώδικα C++ οποιοδήποτε φυσικό νόμο, να αναφέρει με απλό τρόπο ποιές φυσικές ιδιότητες απαιτεί αυτός και με ποιά μορφή θα βρίσκονται αυτές, και να συγγράψει οσοδήποτε απλό η περίπλοκο κώδικα αυτός απαιτεί για να υλοποιήσει το νόμο αυτό. Απαιτείται από τον προγραμματιστή η συγγραφή ενός συγκεκριμένου αρχείου κώδικα με δυο μόνο μεθόδους δεδομένης τάξης που δίνεται ως διεπαφή, και η μεταγλώττισή του κώδικα. Κατόπιν ο φυσικός αυτός νόμος είναι έτοιμος για χρήση, αρκεί το πρόγραμμα-πελάτης να παρέχει ονομαστικά τις απαραίτητες φυσικές ιδιότητες.

Η απαίτηση αυτή θεωρούμε ότι καλύφθηκε πλήρως.

β) Επεκτασιμότητα:

Ο χρήστης μπορεί να φτιάξει οποιοδήποτε αριθμό φυσικών νόμων οποιασδήποτε φύσεως χωρίς κάποιο περιορισμό από την εφαρμογή. Παρατηρώντας τις μελέτες χρηστικότητας και ιδιαίτερα τον φασματογράφο μάζας (Κεφαλαιο 8.2.6), παρατηρούμε την ευκολία με την οποία κάποιος χρήστης μπορεί από τα πρώτα παραδείγματα με τους νόμους της απλής κινηματικής και της βαρύτητας να προσθέσει νόμους ηλεκτρομαγνητικών πεδίων και να δημιουργήσει ένα παράδειγμα πολύ διαφορετικής φύσεως πάνω στο ίδιο ακριβώς πρόγραμμα-πελάτη.

Φυσικοί νόμοι μπορούν να προστεθούν και αφαιρεθούν κατά βούληση χωρίς κανένας περιορισμός να τίθεται από το Vesper3D.

Η απαίτηση αυτή θεωρούμε ότι καλύφθηκε *Βέλτιστα*.

γ) Γενικότητα χρήσεως

Ο προγραμματιστής μπορεί από κώδικά C++ του προγράμματος-πελάτη, να λάβει τα απαραίτητα αρχεία του Vesper3D και να δηλώσει από τον κώδικά του ένα αντικείμενο της τάξης Physics Engine, να φορτώσει όσους νόμους θέλει με μια κλήση τον καθένα, και, αν οι ιδιότητες είναι δομημένες σε απλά arrays, μπορεί να το χρησιμοποιήσει άμεσα.

Σε περίπτωση χρήσης του συστήματος από κάποια μη – native γλώσσα θα απαιτηθεί η σχεδίαση και υλοποίηση μιας διεπαφής, και κατόπιν θα μπορεί το Vesper3D να χρησιμοποιηθεί άμεσα (***Case Study – jReve***).

Σε περίπτωση που οι φυσικές ιδιότητες δεν είναι δομημένες σε arrays, θα πρέπει h διεπαφή που θα συνταχθεί διεπαφή να αναλαμβάνει τη μετάφραση τον φυσικών ιδιοτήτων ανάμεσα στον πελάτη και το Vesper3D (***Case Study – jReve***).

Θεωρείται ότι αυτή η απαίτηση καλύφθηκε πολύ ικανοποιητικά.

δ) Γενικότητα φυσικών σωμάτων:

Δώσαμε στο πρόγραμμα-πελάτη τη δυνατότητα να ορίσει οποιοδήποτε αριθμό φυσικών ιδιοτήτων, με οποιαδήποτε σημασιολογία αυτός απαιτεί. Όταν οι ιδιότητες αυτές

συμπίπτουν με αυτές που αναμένει να δεχτεί κάποιος φυσικός νόμος, αυτές χρησιμοποιούνται αυτόματα.
Προκύπτει μόνο κάποια πολυπλοκότητα όταν υπάρχει ανάγκη μεγάλης μερικής επικάλυψης μεταξύ ιδιοτήτων, αλλά η χρήση δεικτών επιτρέπει τη συγγραφή σχετικών φυσικών νόμων.
Με τον τρόπο αυτό μπορεί να ορισθεί, πρακτικά, οποιοδήποτε σώμα οποιασδήποτε σημασιολογίας. Η απαίτηση αυτή θεωρούμε ότι καλύφθηκε πολύ ικανοποιητικά.

ε) Ελάχιστες δυνατές παραδοχές (*Minimum assumption*):

Δίνεται στο χρήστη η δυνατότητα να μοντελοποιήσει οποιοδήποτε χρονικά εξαρτώμενο σύστημα φυσικής. Απαιτείται μια κλήση ανά καρέ, συνεπώς κάθε σύστημα που χρησιμοποιεί το Vesper3D θα πρέπει να εμφανίζει συμπεριφορά ανά καρέ (per-frame behaviour).
Δίνεται στο χρήστη η δυνατότητα χρήσης φυσικών μεγεθών ή και συναρτήσεων callbacks στον εξωτερικό κώδικα, με οποιονδήποτε τρόπο ορισμού, αριθμό φυσικών νόμων και σειρά εκτέλεσής τους, με δυνατότητα ενεργοποίησης και απενεργοποίησή τους.
Ο μοναδικός περιορισμός που τίθεται είναι στη δόμηση των φυσικών ιδιοτήτων, οι οποίες θα πρέπει πριν την τροφοδοσία στο Vesper3D να βρίσκονται απαραίτητα σε ομοειδείς πίνακες.
Η δυνατότητα προσθήκης και καταστροφής σωμάτων κατά την εκτέλεση (runtime) θα πρέπει να υποστηρίζεται σε επίπεδο φυσικού νόμου.
Χάρη στη χρήση των δεικτών (indexes), μπορούμε να ορίσουμε τα φυσικά μεγέθη των αντικειμένων με αρκετή ελευθερία. Σε σχέση με την υποστήριξη μόνο συγκεκριμένων φυσικών μεγεθών που θα παρείχε μια αρχιτεκτονική με «οριζόντια» δόμηση αντικειμένων, θεωρούμε ότι η απαίτηση καλύφθηκε πολύ ικανοποιητικά.

στ) Μικρό επιπρόσθετο κόστος εκτέλεσης (*Minimum overhead*):

Η κλήση του κενού συστήματος (χωρίς κανένα φυσικό νόμο φορτωμένο) συνίσταται σε μια μοναδική κλήση συνάρτησης και μια μοναδική εντολή σύγκρισης.
Κάθε φυσικός νόμος προσθέτει μια μοναδική εικονική κλήση συνάρτησης (virtual function call) πέρα από τον κατεξοχήν δικό του κώδικα.
Παρατηρήσαμε στις μελέτες επιπρόσθετου κόστους (Case Studies 8.1.x– Εκτέλεση χωρίς νόμους, εκτέλεση με νόμους χωρίς αντικείμενα) ότι το επιπρόσθετο κόστος που προσθέτει το Vesper3D είναι ουσιαστικά μηδενικό, συνεπώς η τελική απόδοση εξαρτάται ουσιαστικά μόνο από το πρόγραμμα πελάτη και τους καθεαυτώ υπολογισμούς των φυσικών νόμων.
Θεωρείται ότι η απαίτηση αυτή καλύφθηκε *Βέλτιστα* (optimally).

ζ) Δυνατότητα υψηλής απόδοσης (*High performance*):

Η απόδοση του συστήματος εξαρτάται πλήρως – προφανώς και κατ' ανάγκη – από το σχεδιασμό των καθεαυτώ φυσικών νόμων.
Όμως, η κάλυψη της προηγούμενης απαίτησης ελαχίστου κόστους εκτέλεσης δίνει ένα εξαιρετικό αρχικό σημείο.
Σε συνδυασμό με αυτό, η δυνατότητα που δόθηκε μέσω της παρεχόμενης βιβλιοθήκης διανυσματικών πράξεων και πράξεων με μήτρες και τετράνια, δίνει στο χρήστη τη δυνατότητα συγγραφής κώδικα υψηλής απόδοσης με ελάχιστη προσπάθεια από μέρους του.
Η κατακόρυφη δόμηση των αντικειμένων έχει επίσης ευεργετικά αποτελέσματα στη χρήση της μνήμης :

Στην οριζόντια δόμηση (*array of structs*), το κάθε αντικείμενο φορτώνεται στην λανθάνουσα μνήμη ανεξάρτητο, ή σε ομάδες. Αν δεν υπάρχει αρκετός χώρος στη λανθάνουσα για όλα τα αντικείμενα, κατά τη διαδοχική εκτέλεση των νόμων κάποια από αυτά θα εκτοπίζονται για να χρειαστεί να ξαναφορτωθούν αργότερα. Αυτό συμβαίνει διότι ο κάθε φυσικός νόμος προφανώς επενεργεί σε υποσύνολο των φυσικών ιδιοτήτων κάθε αντικειμένου. Αντίστοιχα, εάν δεν φορτώνονται αντικείμενα στη λανθάνουσα μνήμη, αλλά μόνο ιδιότητες, χάνεται η δυνατότητα χρήσης της διαδοχικής (sequential) μεταφοράς.
Αντιθέτως, με την κατακόρυφη δόμηση των ιδιοτήτων (*structure of arrays*), η κάθε ιδιότητα που χρησιμοποιείται από κάθε νόμο, είναι εγγυημένο ότι θα φορτωθεί διαδοχικά στη λανθάνουσα μνήμη και θα χρησιμοποιηθεί πλήρως, διότι ο κάθε φυσικός νόμος φορτώνει μόνο τις ιδιότητες που απαιτεί.
Σχετικές μετρήσεις, ενώ δεν είναι απόλυτες, έχουν δείξει ότι όσο αυξάνεται το συνολικό μέγεθος τόσο η κατακόρυφη (*structure of arrays*) δόμηση υπερτερεί στη χρήση της λανθάνουσας μνήμης σε σχέση με την οριζόντια (*array of structs*).
Έτσι επιτυγχάνουμε βέλτιστη αξιοποίηση της λανθάνουσας μνήμης στην περίπτωση που έχουμε πολλά αντικείμενα τα οποία θα χρησιμοποιηθούν από πολλούς φυσικούς νόμους.
Οι μελέτες απόδοσης (8.4.x) αποδεικνύουν την δυνατότητα χρήσης της αιχμής της τεχνολογίας για τη συγγραφή νόμων μέσω του OpenCL API.
Βλέπουμε ότι η δομή που χρησιμοποιήσαμε βεβαιώνει ότι δεν επιβάλλεται τεχνικός περιορισμός στο χρήστη, και του επιτρέπεται να χρησιμοποιήσει διαφορετικές τεχνολογίες στον κώδικά του, με εξαιρετικά αποτελέσματα στην απόδοση.
Υπενθυμίζουμε την αύξηση απόδοσης 520% που το τελευταίο παράδειγμα (8.4.4) πέτυχε σε σχέση με την ήδη καλογραμμένη υλοποίηση αναφοράς.
Περεταίρω ανάλυση των δυο τελευταίων παραδειγμάτων (8.4.3,8.4.4) αποδεικνύει ότι με την δομή που σχεδιάσαμε διατηρούμε αναλλοίωτη στον προγραμματιστή τη δυνατότητα βελτιστοποίησης της απόδοσης και της ανάμειξης μη ομοειδών τεχνολογιών, αφού το σύστημα Vesper3D μπορεί να διευκολύνει την επικοινωνία μεταξύ συστημάτων ανεξάρτητα από την φύση τους.
Θεωρείται ότι η απαίτηση αυτή καλύφθηκε *Βέλτιστα* (optimally).

9.2. Συμπεράσματα - Περίληψη

Η παρούσα εργασία είχε σκοπό την απόδειξη ότι μπορεί να συνταχθεί σύστημα επεξεργασίας φυσικών νόμων το οποίο να δύναται να αποτελέσει τη βάση ανάπτυξης του τμήματος φυσικής για ανεξάρτητα και ανομοιογενή μεταξύ τους συστήματα.
Γίνονται κάποιες αρχικές παραδοχές, η βασικότερη από της οποίες είναι ότι η μηχανή Vesper3D δεν απαιτεί και δεν αναγνωρίζει φυσικά σώματα ως τέτοια, παρά επενεργεί σε θεωρητικώς ανεξάρτητες μεταξύ τους φυσικές ιδιότητες.
Οι φυσικές ιδιότητες με πολύ χαλαρό μόνο τρόπο συνδέονται με σώματα, χωρίς όμως τα σώματα αυτά να διαθέτουν αναγκαστικά κάποιου είδους δομή ή σχέση μεταξύ τους.
Όμως, με τον τρόπο αυτό η μηχανή Vesper3D δεν επιβαρύνεται με ιδιότητες οι οποίες δεν έχουν σχέση με το σκοπό της, και αντίστοιχα δεν επιβαρύνει το πρόγραμμα – πελάτη με την ανάγκη να διατηρεί μη ομοειδείς ιδιότητες στα βασικά του αντικείμενα, ενώ ταυτόχρονα γίνεται απλό να προσπελασθούν από τον κάθε φυσικό νόμο οι ιδιότητες που απαιτεί.

Η παραδοχή αυτή οδηγεί σε μερική λύση του συνδέσμου μεταξύ των φυσικών ιδιοτήτων του Vesper3D και του προγράμματος – πελάτη.
Η παραδοχή αυτή επεκτάθηκε ώστε να σχεδιαστεί ένα μοντέλο στο οποίο οι φυσικοί νόμοι επενεργούν απευθείας σε φυσικές ιδιότητες και όχι σε σώματα, ακολουθώντας το ίδιο μοντέλο όπου οι φυσικοί νόμοι χρησιμοποιούν μόνο τις ιδιότητες που απαιτούν και δεν υπάρχει η ανάγκη να επενεργούν σε εκ των προτέρων ορισμένα σώματα.
Έτσι, το υποσύστημα Vesper3D παίζει το ρόλο του διαχειριστή των φυσικών ιδιοτήτων σε εσωτερικές ή εξωτερικές δομές δεδομένων, τις οποίες διαμοιράζει κατά απαίτηση ανάμεσα στους φυσικούς νόμους και στο πρόγραμμα – πελάτη.
Κατόπιν έγιναν συγκεκριμένες επιλογές για την υλοποίηση του Vesper3D, εκπονήθηκε η υλοποίηση και αναλύθηκε.
Η υλοποίηση βρέθηκε πλήρως λειτουργική. Εκτελέσθηκαν συγκεκριμένα παραδείγματα, που καταδεικνύουν τη χρηστικότητά της, και καθιστούν σαφή την επιτυχία της υλοποίησης αυτής.

10. Βιβλιογραφία

Andrew Rollings: “Fundamentals of Game Design”
(Book) Prentice Hall, 2006

Millington, I: Game Physics Engine Development
(Book) Elsevier 2004, Morgan Kaufmann

Eberly, D; Shoemake,k: “Game Physics”
(Book) Elsevier 2007, Morgan Kaufmann

Colin B. Price: “The usability of a commercial game physics engine to develop physics educational materials: An investigation”
Simulation Gaming 2008, Sage Publishing

Terzopoulos, D.; Witkin, A.: “Physically based models with rigid and deformable components”
Computer Graphics and Applications vol8 (nov 1988, p41-51)

Terzopoulos, D.; Witkin, A.: Kass,M: “Constraints on deformable models: Recovering 3D shape and nonrigid motion”
Artificial Intelligence vol36 issue1 (aug 1988, p91-123)

Terzopoulos, D.; Platt, J.; Barr, A.; Fleischert, K.: “Elastically Deformable Models”, Computer Graphics vol21 (Jul 1987, no.4)

Den Bergen, G.V. : “Efficient Collision Detection of Complex Deformable Models using AABB Trees”, J. Graphics Tool vol2 (1998)

Jacobsen, T.: “Advanced Character Physics”
Game Developers Conference 2001

Catto, E.: “Reinventing The Spring: Soft Constraints”
Game Developers Conference 2001

Moore, M.;Wilhelms, J.: “Collision Detection and Response for Computer Animation”, Computer Graphics vol22 (Aug 1988, no.4)

Faloutsos, P; Yeh, T.; Reinman,G.: “Enabling real-time physics simulation in future interactive entertainment”
Sandbox ’06 Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames

Gamma, E;Helm, R.;Johnson, R; Vlissides,R (Gang of Four): “Design Patterns: Elements of Reusable Object-Oriented Software”
(Book) Addison-Wesley 1995

Joselli, M.;Clua, E.; Montenegro, A.;Conci, A.; Pagliosa, P.: “A new physics engine with automatic process distribution between CPU-GPU”
Sandbox ’08 Proceedings of the 2008 ACM SIGGRAPH symposium on Videogames

Boeing, A.; Braunl, T.: “Evaluation of real-time physics simulation systems”
GRAPHITE 2007 5th international conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia

Price, C.: “Learning physics with the Unreal Tournament engine”
Physics Education, Volume 42 (May 2008, no.3)

Cavazza, M.; Hartley, S.; Lugrin, J.L.; Le Bras, M.: “Qualitative Physics In Virtual Environments”
IUI '04 Proceedings of the 9th international conference on Intelligent user interface

Liliya Kharevych, Rafi (Mohammad) Khan, David Mount “3D Physics Engine for Elastic and Deformable Bodies”, University of Maryland, 2002

Guendelman, E.; Bridson, R.; Fedkiw, R.: “Nonconvex rigid bodies with stacking”
SIGGRAPH ’03 ACM SIGGRAPH 2003 Papers

Biship, B.; Kelliher,T.P.“Specialized Hardware for Deformable Object Modelling”
Circuits and Systems for Video Technology, Vol 13 issue 11(nov 2003, p.1074-1079)

Α. Παράρτημα – Σημαντικά αρχεία κώδικα

PhysicsLaw.h

Αρχείο δηλώσεων (Header file) του νόμου φυσικής, για Visual Studio 2010. Το αρχείο αυτό θα πρέπει να συμπεριληφθεί στη μεταγλώττιση κάθε νόμου φυσικής, και αποτελεί επίσης το βασικό οδηγό για τον προγραμματιστή του νόμου

```
#pragma once
#define WIN32_LEAN_AND_MEAN
#include <Windows.h>
#include <vectorMath.h>

#ifdef PHYSICSLAW_EXPORTING
#define PHYSICSLAW_API __declspec(dllexport)
#else
#define PHYSICSLAW_API __declspec(dllimport)
#endif

namespace Vesper3d{
   //Force fields are functions (callbacks) that laws can call to get the value of a field
   //for a specific point in space during a particular frame.
   typedef v4f (__fastcall *ForceField)(v4f&);

   //PhysicalPropertyDescription carries all data for a specific physical property:
   //A pointer to the actual data, the number of items in the data,
   //and a pointer to the array of indexes if this specific property is indexed.
   //Data can be
   //a) a tightly packed array of scalar floats
   //b) a tightly packed array of v4f vectors
   //c) a tightly packed array of integers that serve as indexes to other bodies(i.e. spring endpoints)
   //d) a funtion pointer to a field function
   //e) a float
   //f) a user-defined pod type
   struct PhysicalPropertyDescription {
      void * Data;
      unsigned int Size;
   };

   //The PhysicalPropertyProvider is a class that manages physical property storage, by adding and getting properties
   //Giving AddPhysicalProperty access to laws gives the possibility of lazy initialization of shared, internal properties.
   //For example, if "forces" is not provided by any program, the first law to need them might create them, and the rest
   //would find them pre-initialized.
   class __declspec(align(16)) PhysicalPropertyProvider
   {
   public:
      virtual PhysicalPropertyDescription GetPhysicalProperty(const char * name) =0;
      virtual bool AddPhysicalProperty(const char * name, void* data, unsigned int size, unsigned int* indexes = 0) =0;
   };

   class PhysicsLaw
   {
   public:
      //Setting disposed=true skips execution of the law in the normal circle of calling the laws.
      bool disposed;
      //Output stream: Use this to communicate logging information. This will be set by the Physics Engine.
      std::wostream * Logger;
      //Override this function to load all propertied that are needed by this law. Use the
      //PhysicalPropertyProvider to get all the relevant properties in your code by calling GetPhysicalProperty.
      virtual bool Initialize(PhysicalPropertyProvider & ppp)=0;
      //This will be called every frame. Perform your Physics logic here.
      virtual void Calculate(float dt)=0;

      PhysicsLaw():Logger(0),disposed(true){}
#include <Snippets/SNIPPET_aligned_new_delete.h>
   };
   //Just return a new concrete object here : return new MyNewPhysicsLaw();
   extern "C" PHYSICSLAW_API PhysicsLaw* __stdcall GetPhysicsLawInstance();
};
```

NewtonsLawOfMotion.cpp

Παράδειγμα νόμου φυσικής. Το αρχείο αυτό είναι το σύνολο του ορισμού ενός φυσικού νόμου – από τον προγραμματιστή απαιτείται μόνο η μεταγλώττιση ενός τέτοιου αρχείου σε .dll και η προσάρτησή του στο PhysicsEngine με την αντίστοιχη κλήση, και φυσικά ο ορισμός των αντίστοιχων φυσικών ιδιοτήτων στο πρόγραμμα.
Για τη χρήση του απαιτούνται τα παρακάτω βήματα:

- το συγκεκριμένο αρχείο να γίνει compile με όνομα PhysicsLaw_NewtonsLawOfMotion32.dll
- Να ζητηθεί από το PhysicsEngine να φορτώσει το νόμο NewtonsLawOfMotion
- Να δημιουργηθεί arrays με float για τις μάζες (μία ανά σώμα),
- Να δημιουργηθούν arrays με float για τις θέσεις, ταχύτητες, δυνάμεις, εξωτερικές δυνάμεις (4 παράμετροι σε ομογενείς συντεταγμένες ανά σώμα : x,y,z,w(=1))
- Να περαστούν στο PhysicsEngine τα παραπάνω arrays με ονόματα “mass”, “position”, “velocity”, “force”,”externalForce”

```
#define PHYSICSLAW_EXPORTING
#include <Vesper3d/PhysicsLaw.h>
namespace Vesper3d
{
   // Extend the PhysicsLaw class to create your law
   class __declspec(align(16)) NewtonsLawOfMotion: public PhysicsLaw
   {
   public:
      // Declare pointers to the arrays of physical Properties you will use
      float * Masses;
      v4f * ExternalForces;
      v4f * Forces_Accelerations;
      v4f * Positions;
      v4f * Velocities;
      unsigned int size;

      bool Initialize(PhysicalPropertyProvider & ppp) override;
      void Calculate(float dt) override;

      NewtonsLawOfMotion();
   };

   //Constructor - Initialize everything to 0/NULL
   NewtonsLawOfMotion::NewtonsLawOfMotion():size(0),Positions(0),Velocities(0),
      Masses(0),Forces_Accelerations(0),ExternalForces(0){}

   //Initialize: The Physics Engine will call this to initialize the law. A reference to
   //a PhysicalPropertyProvider is needed to retrieve the Physical Properties.
   bool NewtonsLawOfMotion::Initialize(PhysicalPropertyProvider & ppp) {
      PhysicalPropertyDescription a;

      //This call retrieves the PhysicalPropertyDescription for “mass”,
      a = ppp.GetPhysicalProperty("mass");
      if (!a.Data) {
         //If the engine has provided a logger, use it before returning failure
         if(Logger) *Logger << L"NewtonsLawOfMotion: Property \"mass\" not loaded\n";
         return false;
      }
      //All clear! Set the masses to your structure, and set the number of masses you have
      //Data contains a pointer to a float (array) here. This should be of course known in
      //advance, since the application must create that data.
      Masses = (float*)a.Data;
      size=a.Size;
      //Same thing to retrieve the positions. Here, Data contains a pointer to a
      //Vector4<float> (v4f), a very data structure with SSE2 for vector math
      a = ppp.GetPhysicalProperty("position");
      if (!a.Data) {
```

```
            if(Logger) *Logger << L"NewtonsLawOfMotion: Property \"position\" not loaded\n";
        }
        Positions = (v4f*)a.Data;

        a = ppp.GetPhysicalProperty("velocity");
        if (!a.Data) {
            if(Logger) *Logger << L"NewtonsLawOfMotion: Property \"velocity\" not loaded\n";
            return false;
        }
        Velocities = (v4f*)a.Data;

        a = ppp.GetPhysicalProperty("force");
        if (!a.Data) {
            if(Logger) *Logger << L"NewtonsLawOfMotion: Property \"force\" not loaded\n";
            return false;
        }
        Forces_Accelerations= (v4f*)a.Data;

        a = ppp.GetPhysicalProperty("externalForce");
        if (!a.Data) {
           if(Logger) *Logger << L"NewtonsLawOfMotion: Property \"externalForce\" not loaded\n";
            return false;
        }
        ExternalForces=(v4f*)a.Data;

        //If we reached this point, the law was initialized correctly. Signal the all clear!
        //If you forget to set disposed to false, the law will not run!
        disposed=false;
        return true;
    }

    // This sample uses forces to calculate velocities according to semi –  implicit Euler, and
    // all objects move according to first and second law of Newton together.
    void NewtonsLawOfMotion::Calculate(float dt)
    {
        //Don't say I didn't warn you… Set disposed to false as soon as you initialize.
        if (disposed) return;
        //Broadcast dt to a v4f for efficient Simd calculations
        v4f dtf(dt,dt,dt,1);
        for (unsigned int i = 0; i<size;++i)
        {
            //Support for immovable bodies.
            if (Masses[i]==0)
            {
                Velocities[i]=v4f(0,0,0);
                continue;
            }
            //OPTIMIZATION: Since this is always the last law to be applied, we transform
            //force to acceleration without extra memory.
            Forces_Accelerations[i]/= Masses[i];

            //Then we use the acceleration to apply x = x0 + u*dt
            Velocities[i]+=Forces_Accelerations[i] * dtf;
            Positions[i]+=Velocities[i] * dtf;
        }
        //External Forces is the starting point for all forces each frame. All other laws will
        //add their forces on top of these, on array "forces".
        memcpy(Forces_Accelerations, ExternalForces, sizeof(v4f)*size);
    }

    //THE FACTORY - this is the only entry point of the dll that returns the instance of the law
    extern "C" __declspec(dllexport) PhysicsLaw* __stdcall GetPhysicsLawInstance(){
        return new NewtonsLawOfMotion();
    }
}
```

PhysicsLaw.h

Το αρχείο αυτό είναι το API που πρέπει να καταναλωθεί από το πρόγραμμα-πελάτη, άμεσα ή μέσω ενδιάμεσου επιπέδου, για να χρησιμοποιήσει το Vesper3D.

```
// The following ifdef block is the standard way of creating macros which make exporting
// from a DLL simpler. All files within this DLL are compiled with the PHYSICSMIDDLEWARE_EXPORTS
// symbol defined on the command line. This symbol should not be defined on any project
// that uses this DLL. This way any other project whose source files include this file see
// VESPER3D_API functions as being imported from a DLL, whereas this DLL sees symbols
// defined with this macro as being exported.
#pragma once
#ifdef VESPER3D_EXPORTS
#define VESPER3D_API __declspec(dllexport)
#else
#define VESPER3D_API __declspec(dllimport)
#endif

#define WIN32_LEAN_AND_MEAN             // Exclude rarely-used stuff from Windows headers
#include <windows.h>
#include <vectorMath.h>
#include <Vesper3d/PhysicsLaw.h>
#include <string.h>
#include <MyUtils/ResizeableArray.h>
#include <map>

namespace Vesper3d{

// Default implementation for the PhysicalPropertyProvider that manages the physical properties
   class __declspec(align(16)) DefaultPhysicalPropertyProvider:public PhysicalPropertyProvider
   {
   private:
      std::map<std::string, PhysicalPropertyDescription> physicalPropertyDictionary;
   public:
      PhysicalPropertyDescription GetPhysicalProperty(const char * prop) override;
      bool AddPhysicalProperty(const char * name, void* data, unsigned int size, unsigned int* indexes = 0) override;
      DefaultPhysicalPropertyProvider():physicalPropertyDictionary(){}
   };

// THE ENGINE. It is exported by Vesper3D.dll
   class VESPER3D_API __declspec(align(16)) PhysicsEngine  {
   private:
      myext::ResizeableArray<PhysicsLaw *> physicsLaws;
      PhysicsLawFactoryFunction getPhysicsLawFactory(const wchar_t *s);
      void __stdcall Log(const std::wstring s);
      void __stdcall Log(double s);
      DefaultPhysicalPropertyProvider basePhysicalPropertyProvider;
      float dtmin;

   public:
      PhysicsEngine();
      virtual ~PhysicsEngine();

      //The client uses this pointer to set or access the PhysicalPropertyProvider to use
      PhysicalPropertyProvider * physicalPropertyProvider;
      std::wostream * loggerStream;

      int __stdcall LoadPhysicsLaw(const wchar_t * law);
      void __stdcall Initialize();
      void __fastcall Advance(float dt);
      void __stdcall setFixedDt(float seconds);
      float __stdcall getFixedDt();
      void __stdcall clearFixedDt();
   };
};
```