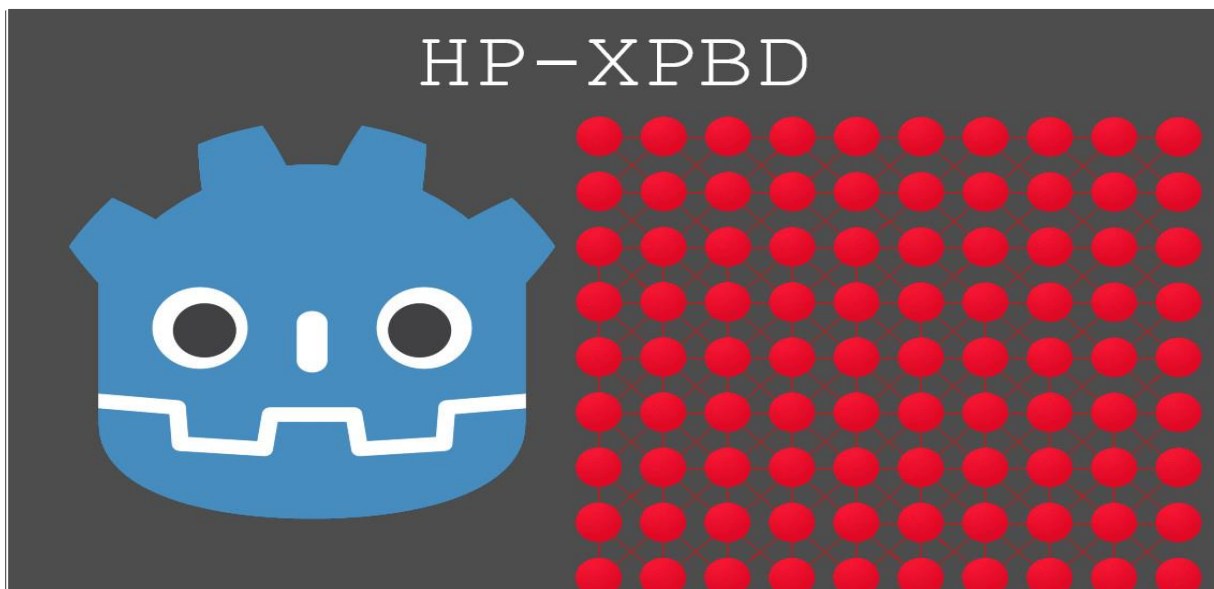


ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Αλγόριθμος Ιεραρχικής Προτεραιοποίησης
Περιορισμών για Extended Position-Based Dynamics»



Του φοιτητή
Γιώργου Περάκη
Αρ. Μητρώου: 2021129

Επιβλέπων:
Παπαδοπούλου Μαρία
Αναπληρωτής Καθηγητής

26/08/2026

Τίτλος Δ.Ε. Αλγόριθμος Ιεραρχικής Προτεραιοποίησης Περιορισμών για Extended Position-Based Dynamics

Κωδικός Δ.Ε. 26178

Ονοματεπώνυμο φοιτητή Γιώργος Περάκης

Ονοματεπώνυμο εισηγητή Παπαδοπούλου Μαρία

Ημερομηνία ανάληψης Δ.Ε. 13/03/2026

Ημερομηνία περάτωσης Δ.Ε. 26/8/2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Γεώργιου Περάκη που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραίτητως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Η παρούσα διπλωματική εργασία έχει ως αντικείμενο τη σχεδίαση και την υλοποίηση ενός plugin προσομοίωσης μαλακών σωμάτων (Soft Body) σε πραγματικό χρόνο για τη μηχανή παιχνιδιών Godot, βασισμένου στον αλγόριθμο Extended Position-Based Dynamics (XPBD). Ένα μεγάλο μέρος της εργασίας αποτελείται από την παρουσίαση ενός νέου επιλυτή, ο οποίος δίνει την δυνατότητα επίλυσης του πλέγματος με την χρήση προτεραιοτήτων. Έχοντας παρακολουθήσει κατά τη διάρκεια των σπουδών μου στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων μαθήματα που άπτονται των Γραφικών Υπολογιστών και του Προγραμματισμού Συστημάτων, διαπίστωσα ότι η μηχανή Godot, παρά τη ραγδαία ανάπτυξή της ως open-source εναλλακτική, στερείται ενός αξιόπιστου συστήματος προσομοίωσης παραμορφώσιμων σωμάτων σε δύο διαστάσεις.

Σκοπός του plugin είναι να προσφέρει έναν προσαρμοσμένο κόμβο (SoftBody2D) που ενσωματώνεται απρόσκοπτα στο περιβάλλον της Godot μέσω του συστήματος GDExtension, επιτρέποντας στον χρήστη να δημιουργεί μαλακά σώματα ορίζοντας παραμέτρους όπως οι διαστάσεις του πλέγματος, η δυσκαμψία (stiffness), ο αριθμός υποβημάτων (substeps) και ο τρόπος επίλυσης. Ο προτεινόμενος ιεραρχικός επιλυτής αξιοποιεί την παρατήρηση ότι σε ένα πλέγμα σωματιδίων οι περιορισμοί δεν συνεισφέρουν ισοδύναμα στη συνολική σταθερότητα: οι δομικοί περιορισμοί (οριζόντιοι και κατακόρυφοι) διατηρούν τη γενική μορφή του σώματος, ενώ οι διατμητικοί (διαγώνιοι) αντιστέκονται στην παραμόρφωση διάτμησης. Με την ταξινόμησή τους σε ομάδες προτεραιότητας και τη διαδοχική επίλυσή τους, ο ιεραρχικός επιλυτής εξασφαλίζει ότι οι κρίσιμοι δομικοί περιορισμοί ικανοποιούνται πρώτοι, μειώνοντας τις αλληλεπιδράσεις μεταξύ ανταγωνιστικών διορθώσεων και βελτιώνοντας τη σύγκλιση του συστήματος. Επιπλέον, το plugin διαθέτει πλήρες σύστημα καταγραφής απόδοσης, χρόνου εκτέλεσης και σφάλματα περιορισμών, εξάγοντας τα αποτελέσματα σε αρχεία CSV και παράγοντας αυτόματα συγκριτικά γραφήματα μεταξύ των δύο επιλυτών. Η επιστημονική σύγκριση μεταξύ των δύο αυτών μεθόδων θα βοηθήσει στην κατανόηση της ιεραρχικής μεθόδου ενώ παράλληλα με βάση επιστημονικά συμπεράσματα θα κριθεί εάν είναι αξιόπιστη μέθοδος επίλυσης.

Το όφελος από την εκπόνηση της παρούσας εργασίας είναι εμφανές. Από τη μία πλευρά, με βοήθησε προσωπικά να ακονίσω τις ικανότητες μου σε προσομοιώσεις φυσικών σωμάτων αλλά και στην ανάπτυξη ελεύθερου λογισμικού. Από την άλλη πλευρά, όπως προκύπτει από τις συγκριτικές μετρήσεις απόδοσης ο ιεραρχικός επιλυτής παρουσιάζει μετρήσιμα πλεονεκτήματα σε σφάλμα σύγκλισης και σταθερότητα σε σχέση με τον Flat XPBD, ενώ το plugin στο σύνολό του αποτελεί ένα χρήσιμο και λειτουργικό εργαλείο για προγραμματιστές παιχνιδιών, ερευνητές ή φοιτητές που επιθυμούν να ενσωματώσουν ρεαλιστικές παραμορφώσεις μαλακών σωμάτων στα 2D projects τους στη Godot.

Περίληψη

Οι τεχνικές προσομοίωσης βασισμένες σε θέσεις (Position-Based Dynamics) εφαρμόζονται ολοένα και περισσότερο στην ανάπτυξη παιχνιδιών και διαδραστικών εφαρμογών για την αναπαράσταση παραμορφώσιμων σωμάτων σε πραγματικό χρόνο. Ωστόσο, η μηχανή παιχνιδιών Godot, παρά τη μεγάλη βιβλιοθήκη οντοτήτων για άμεση χρήση, δεν διαθέτει προσομοιώσεις μαλακών σωμάτων.

Στην παρούσα διπλωματική εργασία, παρουσιάζεται η σχεδίαση και υλοποίηση ενός plugin μαλακών σωμάτων για τη Godot, βασισμένου στον αλγόριθμο XPBD, με κεντρική συνεισφορά την πρόταση ενός νέου Ιεραρχικού Επιλυτή Περιορισμών (Hierarchical Constraint Solver). Ο προτεινόμενος επιλυτής ταξινομεί τους περιορισμούς σε ομάδες προτεραιότητας: δομικούς (structural) σε υψηλή προτεραιότητα και διατμητικούς (shear) σε χαμηλότερη, και τους επιλύει διαδοχικά, εξασφαλίζοντας ότι οι κρίσιμοι δομικοί περιορισμοί ικανοποιούνται πρώτοι, μειώνοντας τις αλληλεπιδράσεις μεταξύ ανταγωνιστικών διορθώσεων και βελτιώνοντας τη συνολική ταχύτητα εκτέλεσης σε σχέση με τον συμβατικό Flat XPBD. Το plugin υλοποιείται σε C++ ως GDExtension, προσφέροντας έναν προσαρμοσμένο κόμβο SoftBody2D με πλήρη παραμετροποίηση μέσω του Inspector της Godot, διαστάσεις πλέγματος, δυσκαμψία, αριθμό υπο-βημάτων, επαναλήψεις επιλυτή και επιλογή αλγορίθμου.

Επιπλέον, το plugin ενσωματώνει σύστημα μετρήσεων απόδοσης που καταγράφει χρόνους εκτέλεσης βήματος και σφάλματα περιορισμών, εξάγει τα δεδομένα σε μορφή CSV και παράγει αυτόματα συγκριτικά γραφήματα μεταξύ των δύο επιλυτών. Το σύστημα αξιολογήθηκε μέσω πειραματικών μετρήσεων σε πλέγματα σωματιδίων κλιμακούμενα έως 10.000 σωματίδια. Η σύγκριση έγινε μεταξύ του αλγόριθμου XPBD και του προτεινόμενου HP-XPBD ως προς τον χρόνο βήματος, τον μέσο χρόνο βήματος, τον μέγιστο χρόνο βήματος και το σφάλμα σύγκλισης περιορισμών.

«Hierarchical Constrain Prioritization Algorithm For Extended Position-Based Dynamics»

«George Perakis»

Abstract

Position-Based Dynamics (PBD) simulation techniques are increasingly being applied in the development of games and interactive applications for the real-time representation of deformable bodies. However, despite its extensive library of readily usable entities, the Godot game engine does not provide soft-body simulations.

This thesis presents the design and implementation of a soft-body plugin for Godot based on the XPBD algorithm, with the main contribution being the proposal of a novel Hierarchical Constraint Solver. The proposed solver classifies constraints into priority groups: structural constraints at high priority and shear constraints at lower priority, solving them sequentially. This ensures that critical structural constraints are satisfied first, reducing interactions between competing corrections and improving overall step time compared to conventional Flat XPBD. The plugin is implemented in C++ as a GDEExtension and provides a customizable SoftBody2D node with full configuration through the Godot Inspector, including mesh dimensions, stiffness, number of substeps, solver iterations, and algorithm selection.

Additionally, the plugin incorporates a performance measurement system that records simulation step execution times and constraint errors, exports the data in CSV format, and automatically generates comparative graphs between the two solvers. The system was evaluated through experimental measurements on particle grids scaling up to 10,000 particles. The XPBD algorithm and the proposed HP-XPBD algorithm were compared in terms of step time, average step time, maximum step time, and constraint convergence error.

Περιεχόμενα

Πρόλογος	iii
Περίληψη	iv
Abstract	v
Περιεχόμενα	vi
Κατάλογος Σχημάτων	viii
Κατάλογος Πινάκων	viii
Συντομογραφίες	ix
Κεφάλαιο 1ο: Θεωρητικό υπόβαθρο	1
1.1 Εισαγωγή	1
1.2 Position-Based Dynamics	2
1.2.1 Βασικές αρχές PBD	2
1.2.2 Περιορισμοί.....	3
1.3 Extended Position-Based Dynamics	4
1.3.1 Μαθηματική θεμελίωση	4
1.3.2 Πολλαπλασιαστής Lagrange και συμμόρφωση.....	5
1.3.3 Σύγκριση PBD και XPBD.....	7
1.4 Μαλακά σώματα σε πραγματικό χρόνο	8
1.4.1 Μοντελοποίηση με σωματίδια και ελατήρια.....	8
1.4.2 Ιεραρχική επίλυση περιορισμών	9
1.5 Σχετικές εργασίες.....	11
1.5.1 Μέθοδοι βασισμένες σε θέση.....	11
1.5.2 Μέθοδοι πεπερασμένων στοιχείων.....	11
1.5.3 Ενοποιημένα πλαίσια σωματιδίων	12
1.5.4 Υλοποιήσεις σε μηχανές παιχνιδιών.....	12
Κεφάλαιο 2ο: Αρχιτεκτονική και Σχεδίαση του Plugin.....	13
2.1 Εισαγωγή	13
2.2 Μηχανή Godot και GDExtension	14
2.2.1 Αρχιτεκτονική GDExtension	14
2.2.2 Σύνδεση C++ με Godot	15
2.3 Δομή του plugin.....	16
2.3.1 Κλάση SoftBody2D.....	17

2.3.2 Δομή Particle.....	18
2.3.3 Δομή Distance_Constraint	19
2.3.4 Βοηθητική κλάση Vec2.	19
2.4 Σύστημα Build	20
Κεφάλαιο 3ο: Υλοποίηση του XPBD επιλυτή	23
3.1 Αλγόριθμος προσομοίωσης	23
3.1.1 Βρόχος υποβημάτων.....	23
3.1.2 Πρόβλεψη θέσεων	24
3.1.3 Προβολή περιορισμών.	25
3.2 Flat XPBD επιλυτής.....	26
3.3 Ιεραρχικός επιλυτής	27
3.4 Δημιουργία πλέγματος σωματιδίων	28
3.5 Ανίχνευση σύγκρουσης με δάπεδο.....	28
3.6 Συνολική ροή	29
Κεφάλαιο 4ο: Μετρήσεις απόδοσης και αξιολόγηση	31
4.1 Εισαγωγή	31
4.2 Σύστημα μετρήσεων.....	31
4.2.1 Δειγματοληψία και καταγραφή.	31
4.2.2 Εξαγωγή σε CSV.....	33
4.3 Πειραματική διάταξη.....	34
4.3.1 Πειράματα δοκιμών.	34
4.3.2 Περιβάλλον εκτέλεσης	38
4.4 Αποτελέσματα.....	38
4.4.1 Σύγκριση επιλυτών (Ομάδα Α)	39
4.4.2 Κλιμάκωση μεγέθους (Ομάδα Β).....	47
4.4.2 Υποβήματα ενάντι επαναλήψεων (Ομάδα Γ)	54
4.6 Συζήτηση αποτελεσμάτων	59
Κεφάλαιο 5ο: Συμπεράσματα και μελλοντική εργασία	61
5.2 Συνεισφορά της εργασίας.....	61
5.3 Προτάσεις μελλοντικής εργασίας	61
ΒΙΒΛΙΟΓΡΑΦΙΑ	62

Κατάλογος Σχημάτων

Σχήμα 2.2.2: Εξατομικευμένη κλάση μέσα στον inspector της Godot	16
Σχήμα 2.3: Δομή οντότητας <i>SoftBody2D</i>	17
Σχήμα 2.4: Οθόνη επιλογών βοηθητικού script	24
Σχήμα 3.6: Διάγραμμα ροής λογικής plugin	30
Σχήμα 4.4.1 (1): Αποτελέσματα πειράματος A1	39
Σχήμα 4.4.1 (2): Αποτελέσματα πειράματος A2	40
Σχήμα 4.4.1 (3): Αποτελέσματα πειράματος A3	41
Σχήμα 4.4.1 (4): Αποτελέσματα πειράματος A4	42
Σχήμα 4.4.1 (5): Αποτελέσματα πειράματος A5	43
Σχήμα 4.4.1 (6): Αποτελέσματα πειράματος A6	44
Σχήμα 4.4.1 (7): Αποτελέσματα πειράματος A7	45
Σχήμα 4.4.2 (1): Αποτελέσματα πειράματος B1	47
Σχήμα 4.4.2 (2): Αποτελέσματα πειράματος B2	48
Σχήμα 4.4.2 (3): Αποτελέσματα πειράματος B3	49
Σχήμα 4.4.2 (4): Αποτελέσματα πειράματος B4	50
Σχήμα 4.4.2 (5): Αποτελέσματα πειράματος B5	51
Σχήμα 4.4.2 (6): Αποτελέσματα πειράματος B6	52
Σχήμα 4.4.2 (7): Αποτελέσματα πειράματος B7	53
Σχήμα 4.4.3 (1): Αποτελέσματα πειράματος Γ1	54
Σχήμα 4.4.3 (2): Αποτελέσματα πειράματος Γ2	55
Σχήμα 4.4.3 (3): Αποτελέσματα πειράματος Γ3	56
Σχήμα 4.4.3 (4): Αποτελέσματα πειράματος Γ4	57
Σχήμα 4.4.3 (5): Αποτελέσματα πειράματος Γ5	58

Κατάλογος Πινάκων

Πίνακας 4.2.1: Τύποι μετρήσεων	32
Πίνακας 4.3.1 (1): Πείραμα A1	34
Πίνακας 4.3.1 (2): Πείραμα A2	35
Πίνακας 4.3.1 (3): Πείραμα A3	35
Πίνακας 4.3.1 (4): Πείραμα A4	35
Πίνακας 4.3.1 (5): Πείραμα A5	36
Πίνακας 4.3.1 (6): Πείραμα A6	36
Πίνακας 4.3.1 (7): Πείραμα A7	36
Πίνακας 4.3.1 (8): Πειράματα ομάδας B	37
Πίνακας 4.3.1 (9): Πειράματα ομάδας Γ	38
Πίνακας 4.3.2: Στοιχεία συστήματος	38
Πίνακας 4.4.1: Αποτελέσματα πειραμάτων ομάδας A	45
Πίνακας 4.4.2 (1): Αποτελέσματα πειραμάτων μικρών πλεγμάτων	50
Πίνακας 4.4.2 (2): Αποτελέσματα πειραμάτων μεγάλων πλεγμάτων	53
Πίνακας 4.4.3 (1): Αποτελέσματα πειραμάτων υποβημάτων	56
Πίνακας 4.4.3 (2): Αποτελέσματα πειραμάτων μεγάλων πλεγμάτων	58

Συντομογραφίες

PBD	Position Based Dynamics
XPBD	Extended Position Based Dynamics
HP-XPBD	Hierarchical Prioritization-Extended Position Based Dynamics
FEM	Finite Element Method
KKT	Karush-Kuhn-Tucker
SPH	Smoothed Particle Hydrodynamics

Κεφάλαιο 1ο: Θεωρητικό υπόβαθρο

1.1 Εισαγωγή

Η προσομοίωση φυσικών φαινομένων αποτελεί πυλώνα των σύγχρονων εφαρμογών, των βιντεοπαιχνιδιών και των εικονικών περιβαλλόντων. Ανάμεσα στις πολλές κατηγορίες φυσικής προσομοίωσης, η αναπαράσταση παραμορφώσιμων ή μαλακών σωμάτων (soft bodies) παρουσιάζει ιδιαίτερο ενδιαφέρον, διότι επιτρέπει τη ρεαλιστική απεικόνιση αντικειμένων που αλλάζουν σχήμα ανάλογα με τις παραμέτρους του περιβάλλοντος και την επίδραση εξωτερικών δυνάμεων [1]. Σε αντίθεση με τα άκαμπτα σώματα (rigid bodies), τα οποία διατηρούν σταθερή γεωμετρία, τα μαλακά σώματα απαιτούν αλγορίθμους που επιλύουν συνεχώς τις εσωτερικές δυνάμεις μεταξύ των στοιχείων τους, γεγονός που αυξάνει σημαντικά το υπολογιστικό κόστος.

Για την αντιμετώπιση του προαναφερθέντος προβλήματος, δημιουργήθηκαν διάφορα μοντέλα προσομοίωσης για την ψηφιακή αναπαράσταση μαλακών αντικειμένων. Τα πιο συνηθισμένα μοντέλα βασίζονται στη «Μέθοδο Πεπερασμένων Στοιχείων» (FEM), τα οποία είναι εξαιρετικά ακριβή, αλλά είναι δύσκολο να εκτελεστούν με μεγάλη ταχύτητα λόγω του όγκου των υπολογισμών που απαιτούνται για κάθε προσομοίωση [2]. Οι Müller et al [3] πρότειναν μια νέα διαφορετική προσέγγιση για τη δημιουργία προσομοίωσης μαλακών σωμάτων χρησιμοποιώντας μεθόδους βασισμένες στη θέση, όπου δεν επίλυαν τις δυνάμεις που ασκούνται στα σωματίδια, αλλά αντίθετα επιλύουν ελέγχοντας άμεσα τη θέση κάθε σωματιδίου με περιορισμούς, έτσι ώστε το σύστημα να είναι σταθερό και αρκετά γρήγορο για να χρησιμοποιηθεί σε εφαρμογές πραγματικού χρόνου.

Παρά τα πλεονεκτήματά της, η κλασική PBD παρουσιάζει ένα σημαντικό μειονέκτημα: η δυσκαμψία του προσομοιούμενου υλικού εξαρτάται από τον αριθμό των επαναλήψεων του επιλυτή και το μέγεθος του χρονικού βήματος, καθιστώντας δύσκολη τη ρύθμιση των παραμέτρων από τον χρήστη [3]. Η μέθοδος Extended Position-Based Dynamics (XPBD), που προτάθηκε από τους Macklin et al [4], αντιμετωπίζει αυτό το πρόβλημα εισάγοντας πολλαπλασιαστές Lagrange και την έννοια της σύγκλισης, αποσυνδέοντας τις φυσικές ιδιότητες του υλικού από τις αριθμητικές παραμέτρους του επιλυτή.

Στο πλαίσιο της παρούσας εργασίας, μελετώνται οι παραπάνω μέθοδοι ως θεωρητική βάση για την υλοποίηση ενός plugin προσομοίωσης μαλακών σωμάτων σε δύο διαστάσεις. Πέρα από την υλοποίηση του κλασικού XPBD, η εργασία προτείνει έναν νέο Ιεραρχικό Επιλυτή, ο οποίος οργανώνει τους περιορισμούς σε επίπεδα προτεραιότητας αξιοποιώντας την παρατήρηση ότι δεν συνεισφέρουν όλοι οι περιορισμοί ισοδύναμα στη συνολική σταθερότητα ενός πλέγματος σωματιδίων [5].

1.2 Position Based Dynamics

Η μέθοδος Position Based Dynamics (PBD) παρουσιάστηκε από τους Müller et al. το 2007 [3] ως μια εναλλακτική προσέγγιση στην προσομοίωση παραμορφώσιμων σωμάτων, σχεδιασμένη εξ αρχής για εφαρμογές πραγματικού χρόνου. Η κύρια ιδέα της PBD μεθόδου, έχει να κάνει με την διαφορετική προσέγγιση όσον αφορά την προσομοίωση μαλακών σωμάτων. Αντί να ολοκληρώνονται οι δυνάμεις που δρουν πάνω στα σώματα για να υπολογιστεί η θέση τους, η μέθοδος PBD, αναπαριστά κάθε σώμα ως ένα πλέγμα σωματιδίων. Υπολογίζει απευθείας τις καινούργιες θέσεις των σωματιδίων του πλέγματος ενώ παράλληλα εκπληρώνει ένα σύνολο κανόνων που ονομάζονται περιορισμοί. Ο κάθε

περιορισμός αντιστοιχεί σε ένα φυσικό φαινόμενο όπως διατήρηση συνολικού όγκου, απόστασης μεταξύ σωματιδίων κ.α. Χωρίς περιορισμούς, η προσομοίωση δεν θα μπορούσε να σταθεί μόνη της και ολόκληρο το πλέγμα θα κατέρρεε.

Η προσέγγιση αυτή προσφέρει ένα θεμελιώδες πλεονέκτημα: τη μη υπό συνθήκη σταθερότητα. Το πρόβλημα που προκύπτει από τις κλασικές μεθόδους υπολογισμού δυνάμεων είναι η συσσώρευση ενέργειας σε κάθε βήμα υπολογισμού. Καθώς προστίθεται ολοένα και περισσότερο αυξανόμενη ενέργεια, το σύστημα γίνεται όλο και πιο ασταθές και σε κάποιες περιπτώσεις εκρήγνυται. Η PBD αποφεύγει εντελώς αυτό το πρόβλημα, διότι οι διορθώσεις θέσεων είναι εγγενώς φραγμένες: κάθε σωματίδιο μετακινείται μόνο κατά το ποσό που απαιτείται για την ικανοποίηση του περιορισμού, χωρίς να συσσωρεύεται ενέργεια στο σύστημα.

1.2.1 Βασικές αρχές PBD

Ο αλγόριθμος PBD ακολουθεί μια καθαρή δομή ανά χρονικό βήμα Δt η οποία μπορεί να συνοψιστεί στα εξής στάδια:

Αρχικά, γίνεται η πρόβλεψη των επόμενων θέσεων των σωματιδίων. Για κάθε σωματίδιο i με θέση x_i και ταχύτητα v_i , υπολογίζεται μια προβλεπόμενη θέση p_i λαμβάνοντας υπόψη τις εξωτερικές δυνάμεις, κυρίως τη βαρύτητα:

$$v_i \leftarrow v_i + \Delta t \cdot f_{ext} \cdot w_i$$

$$p_i \leftarrow x_i + \Delta t \cdot v_i$$

Γίνεται χρήση του αντίστροφου βάρους w_i , διότι όσο μεγαλύτερη μάζα έχει ένα σωματίδιο, τόσο λιγότερο θέλουμε να επηρεάζεται από τους περιορισμούς. Το f_{ext} αναφέρεται σε όλες τις εξωτερικές δυνάμεις που επιδρούν πάνω στο σωματίδιο. Κάθε σωματίδιο με $w_i = 0$, θεωρείται ακίνητο και δεν μπορεί να επηρεαστεί από καμία δύναμη.

Έπειτα ακολουθεί η επίλυση περιορισμών. Σε αυτό το βήμα του αλγορίθμου, οι προβλεπόμενες θέσεις των σωματιδίων τροποποιούνται με κάθε επανάληψη του βρόγχου. Εφόσον έχουν ικανοποιηθεί όλοι οι περιορισμοί του συστήματος, μπορούμε να προχωρήσουμε στο επόμενο βήμα.

Το τελευταίο βήμα είναι η ενημέρωση των ταχυτήτων κάθε σωματιδίου. Δεδομένου ότι έχουν γίνει όλες οι απαραίτητες διορθώσεις, μπορεί να γίνει η ενημέρωση έτσι ώστε κάθε σωματίδιο στο πλέγμα να αντικατοπτρίζει την πραγματική μετακίνηση που υπέστη:

$$v_i \leftarrow \frac{p_i - x_i}{\Delta t}$$

$$x_i \leftarrow p_i$$

Σε αυτό το σημείο παρατηρείται ο πιο σημαντικός μηχανισμός που τον διαφοροποιεί από άλλους εκείνη την περίοδο. Η ταχύτητα κάθε σώματος υπολογίζεται από τη μεταβολή της θέσης του. Επομένως, πρώτα υπολογίζουμε την προβλεπόμενη θέση, και έπειτα δρα η ταχύτητα πάνω στο σώμα. Οποιαδήποτε θέση

που καταρρίπτει τους επιβαλλομένους περιορισμούς διορθώνεται. Για αυτόν τον λόγο προκύπτει η σταθερότητα. Πλέον το σύστημα δεν προσθέτει επιπλέον ενέργεια με κάθε βήμα λόγω λάθος υπολογισμών, αλλά γίνεται η διόρθωση από πριν.

Εξίσου σημαντική είναι η χρήση υποβημάτων Δt . Κάθε βήμα Δt , υποδιαιρείται σε μικρότερα βήματα όπου ισχύει. Κάθε υποβήμα, ολοκληρώνει έναν κύκλο πρόβλεψης, επίλυσης και ενημέρωσης. Το αποτέλεσμα είναι η σημαντική βελτίωση της προσομοίωσης, χωρίς όμως να αλλοιώνεται το συνολικό χρονικό πλαίσιο στο οποίο παίρνει μέρος.

Η PBD υιοθετήθηκε ταχύτατα τόσο στην ακαδημαϊκή κοινότητα όσο και στη βιομηχανία βιντεοπαιχνιδιών, χάρη στην απλότητα υλοποίησής της και στην ικανότητά της να παρέχει οπτικά πειστικά αποτελέσματα. Μηχανές φυσικής όπως οι NVIDIA PhysX και Havok ενσωματώνουν παραλλαγές της PBD για την προσομοίωση υφασμάτων, μαλακών σωμάτων και σχοινιών [5].

1.2.2 Περιορισμοί

Όπως προαναφέρθηκε σε προηγούμενα κεφάλαια, οι περιορισμοί αποτελούν τον πυρήνα της προσομοίωσης. Κάθε περιορισμός που μπορεί να εφαρμοστεί, απεικονίζεται ως μία συνάρτηση $C(p_1, p_2, \dots) = 0$, όπου εκφράζει την επιθυμητή σχέση μεταξύ δυο σωματιδίων. Η δουλειά του επιλυτή είναι να μεταβάλει κάθε θέση σωματιδίου του πλέγματος, έτσι ώστε το C να συγκλίνει προς το 0. Ο πιο θεμελιώδης και ευρέως χρησιμοποιούμενος περιορισμός είναι ο περιορισμός απόστασης, ο οποίος εξασφαλίζει ότι δύο σωματίδια a και b διατηρούν μια σταθερή απόσταση d_0 μεταξύ τους:

$$C(p_a, p_b) = \|p_b - p_a\| - d_0$$

Όταν $C > 0$, τα σωματίδια έχουν απομακρυνθεί περισσότερο από την απόσταση ηρεμίας και πρέπει να πλησιάσουν. Όταν $C < 0$, έχουν πλησιάσει υπερβολικά και πρέπει να απομακρυνθούν. Ο τρόπος με τον οποίο αυτή η διόρθωση κατανέμεται μεταξύ των δύο σωματιδίων εξαρτάται από τα αντίστροφα βάρη τους: ελαφρύτερα σωματίδια μετακινούνται περισσότερο, βαρύτερα λιγότερο. Η κλίση του περιορισμού δίνεται από:

$$\nabla_{p_a} C = -\frac{p_b - p_a}{\|p_b - p_a\|}, \nabla_{p_b} C = \frac{p_b - p_a}{\|p_b - p_a\|}$$

και η διόρθωση θέσης κατανέμεται ανάλογα:

$$\Delta p_a = -\frac{w_a}{w_a + w_b} \cdot C \cdot \hat{n}, \Delta p_b = \frac{w_b}{w_a + w_b} \cdot C \cdot \hat{n}$$

όπου $\hat{n} = \frac{(p_b - p_a)}{\|p_b - p_a\|}$ το μοναδιαίο διάνυσμα κατεύθυνσης. Επομένως, με το που βρεθεί η κλίση στην οποία πρέπει να κατευθυνθεί κάθε σωματίδιο, μετακινείται σε αυτόν τον άξονα ανάλογα με το μέγεθος της διόρθωσης. Για λόγους χρονικών περιορισμών και με σκοπό τη διατήρηση της μελέτης σε διαχειρίσιμο επίπεδο πολυπλοκότητας, στην παρούσα έρευνα θα εξεταστεί αποκλειστικά ο συγκεκριμένος περιορισμός.

1.3 Extended Position Based Dynamics

Η μέθοδος PBD, ήταν εξαιρετικά δημιουργική και αποδοτική για την περίοδο στην οποία δημοσιεύτηκε. Υιοθετήθηκε από την επιστημονική κοινότητα μαζί και με άλλους κλάδους σχεδόν ακαριαία, ωστόσο παρά την σταθερότητα και απλότητα που προσέφερε είχε σημαντικές αδυναμίες. Το μεγαλύτερο μειονέκτημα της μεθόδου έχει να κάνει με την άμεση σχέση μεταξύ της ιδιότητας του υλικού και τις παραμέτρους της προσομοίωσης. Λόγω της λογικής των βημάτων του επιλυτή, κάθε περιορισμός δεν μπορεί να λυθεί με απόλυτη ακρίβεια. Όσο λιγότερα είναι τα βήματα του επιλυτή, τόσο πιο εύκαμπτο γίνεται ένα σώμα με τις ίδιες παραμέτρους. Αυτό το πρόβλημα αναγνωρίστηκε νωρίς στη βιβλιογραφία: οι Goldenthal et al. [6] πρότειναν ήδη από το 2007 τη χρήση πολλαπλασιαστών Lagrange για την επίλυση περιορισμών σε υφάσματα, ενώ οι Tournier et al. [7] παρουσίασαν ένα πλαίσιο σταθερής δυναμικής υπό περιορισμούς που συνέδεε ρητά τη μηχανική ενέργεια με τη διαδικασία προβολής. Σε αυτό ακριβώς το θεωρητικό υπόβαθρο στηρίχθηκαν οι Macklin et al για τη διατύπωση της XPBD, μιας μεθόδου που διατηρεί την αλγοριθμική δομή της PBD, αλλά θεμελιώνεται πλέον σε μια αυστηρή αρχή.

Η NVIDIA ενσωμάτωσε XPBD στο σύστημα FleX [8] για ενοποιημένη προσομοίωση σωματιδίων σε πραγματικό χρόνο, ενώ η μεταγενέστερη ανασκόπηση των Bender et al. [5] τοποθέτησε την XPBD ως τη σύγχρονη αναφορά για οποιαδήποτε για μέθοδο βασισμένη σε θέσεις.

1.3.1 Μαθηματική θεμελίωση

Η μέθοδος XPBD προσπαθεί να αλλάξει την έννοια των περιορισμών. Αντί να τους αναπαριστά ως γεωμετρικές προβολές, εντάσσονται σε πρόβλημα ελαχιστοποίησης ενέργειας. Σε κάθε βήμα, επιλύεται το παρακάτω σύστημα εξισώσεων:

$$M(x - \tilde{x}) = \Delta t^2 \nabla C(x)^T \lambda$$

$$C(x) + \tilde{\alpha} \lambda = 0$$

Το M αναφέρεται σε διαγώνιο πίνακα που περιέχει όλες τις μάζες των σωματιδίων ενώ $\tilde{x}$ είναι οι προβλεπόμενες θέσεις αφού έχουν επηρεαστεί από τις εξωτερικές δυνάμεις. Η δεύτερη εξίσωση μας εισάγει στις δύο πιο σημαντικές μεταβλητές του αλγορίθμου. Το λ αναφέρεται στον πολλαπλασιαστή Lagrange ενώ το $\tilde{\alpha}$ είναι χρονικά κανονικοποιημένη ‘συμμόρφωση’, δηλαδή το πόσο επιτρέπεται σε έναν περιορισμό να υποχωρήσει. Το $\tilde{\alpha}$ εκφράζεται από την παρακάτω σχέση:

$$\tilde{\alpha} = \frac{\alpha}{\Delta t^2} = \frac{1}{k \cdot \Delta t^2}$$

Τώρα μπορεί να γίνει κατανοητή η ερμηνεία της πρώτης εξίσωσης. Εκφράζει τον 2ο νόμο του Newton κατά τον οποίο:

$$F = ma$$

Ωστόσο στην προκειμένη περίπτωση αναφερόμαστε σε μια διακριτοποιημένη μορφή. Η μετακίνηση ενός σωματιδίου από την προβλεπόμενη θέση του οφείλεται στις δυνάμεις των περιορισμών, οι οποίες εκφράζονται μέσω του γινομένου $\nabla C(x)^T \lambda$. Η μέθοδος δεν απαιτεί όλοι οι περιορισμοί να μηδενιστούν αυστηρά. Γίνεται επιτρεπτή μία ελεγχόμενη παραβίαση που αναλογεί στο γινόμενο $\tilde{\alpha} \lambda$.

Η δυσκαμψία του υλικού εκφράζεται πλέον από τη μεταβλητή k ανάλογα με τον τύπο του περιορισμού. Το αντίστροφο της δυσκαμψίας είναι η ενδοτικότητα.

$$\alpha = \frac{1}{k}$$

Η κανονικοποίηση με την μεταβλητή Δt^2 , επιτρέπει την μεταβλητή να έχει την ίδια ερμηνεία ανεξαρτήτως από το χρονικό βήμα που ακολούθησε. Χωρίς αυτή, θα υπήρχαν διαφορετικά αποτελέσματα ανάλογα με το χρονικό βήμα που έχει οριστεί στην προσομοίωση.

1.3.2 Πολλαπλασιαστής Lagrange και συμμόρφωση

Το προαναφερόμενο σύστημα εξισώσεων είναι μη γραμμικό. Οι περιορισμοί $C(x)$, εξαρτώνται από τις άγνωστες θέσεις των σωματιδίων. Η επίλυση του συστήματος στη μέθοδο είναι επαναληπτική, μια διαδικασία πολύ παρόμοια με την μέθοδο Newton. Σε κάθε επανάληψη, για κάθε περιορισμό j ανάμεσα σε σωματίδια a και b , υπολογίζεται:

$$\Delta \lambda_j = \frac{-C_j(x) - \tilde{\alpha}_j \lambda_j}{w_a + w_b + \tilde{\alpha}_j}$$

Στην συγκεκριμένη συνάρτηση, w_a και w_b είναι τα αντίστροφα βάρη των δύο σωματιδίων, δηλαδή:

$$w_a = \frac{1}{m_a}, w_b = \frac{1}{m_b}$$

Ο παρονομαστής της προαναφερόμενης εξίσωσης $w_a + w_b + \tilde{\alpha}_j$ εκφράζει τρία στοιχεία. Τα πρώτα δύο είναι τα αντίστροφα βάρη των σωματιδίων, δηλαδή πόσο μεγάλη δυνατότητα μετακίνησης έχουν. Το τρίτο έχει να κάνει με την ενδοτικότητα του περιορισμού. Μπορεί να καθορίσει πόσο ‘μαλακός’ είναι ένας περιορισμός. Στον αριθμητή συναντάται η τρέχουσα παραβίαση περιορισμού $C_j(x)$ μαζί με τον πολλαπλασιαστή Lagrange. Ο πολλαπλασιαστής έχει μια μοναδική ιδιότητα η οποία ξεχωρίζει τη μέθοδο PBD σε σχέση με την μέθοδο XPBD. Ενημερώνεται συσσωρευτικά:

$$\lambda_j \leftarrow \lambda_j + \Delta\lambda_j$$

Η εξάρτηση του σε σχέση με όλους τους προηγούμενους πολλαπλασιαστές είναι αυτή που τον βοηθάει να συγκλίνει σωστά και ‘ρεαλιστικά’, σε αντίθεση με το να ξεκινάει τον υπολογισμό $\Delta\lambda_j$ από την αρχή σε κάθε βήμα όπως η μέθοδος PBD. Οι θέσεις των σωματιδίων διορθώνονται αναλογικά με τα αντίστροφα βάρη:

$$x_a \leftarrow x_a - w_a \Delta\lambda_j \hat{n}_j$$

$$x_b \leftarrow x_b + w_b \Delta\lambda_j \hat{n}_j$$

Όπου

$$\hat{n}_j = \frac{\nabla C_j}{|\nabla C_j|}$$

είναι η κανονικοποιημένη κατεύθυνση της κλίσης του περιορισμού. Η συσσωρευτική ενημέρωση λ επιτρέπει στον επιλυτή να λαμβάνει υπόψη τις διορθώσεις που έχουν ήδη πραγματοποιηθεί στα προηγούμενα υποβήματα. Δηλαδή ο πολλαπλασιαστής λειτουργεί σαν ‘μνήμη’ του συστήματος. Επειδή αντιπροσωπεύει τη συσσωρευτική δράση του περιορισμού, επιτρέπει στον επιλυτή να λάβει υπόψη της ήδη υπάρχουσες διορθώσεις που έχουν πραγματοποιηθεί. Ωστόσο, δεν είναι επιθυμητή η λύση της προηγούμενης κατάστασης και για αυτό τον λόγο σε κάθε χρονικό βήμα μηδενίζεται.

$$\lambda_j = 0,$$

Έπειτα, ξεκινάει εκ νέου η διαδικασία διόρθωσης στο επόμενο βήμα, έχοντας την προσωρινή ‘μνήμη’ του πολλαπλασιαστή σε κάθε υποβήμα για ένα ‘ρεαλιστικό’ αποτέλεσμα που μας επιτρέπει να διαφοροποιήσουμε τον αριθμό βημάτων από την υλική υπόσταση του αντικειμένου που προσομοιώνουμε.

Οι Macklin et al. [4] έδειξαν ότι η διαδικασία επίλυσης της XPBD μπορεί να ερμηνευθεί ως εφαρμογή Gauss-Seidel στο σύστημα KKT του αντίστοιχου προβλήματος βελτιστοποίησης. Η ερμηνεία αυτή παρέχει ένα αυστηρότερο θεωρητικό υπόβαθρο σε σχέση με την απλή γεωμετρική προβολή της PBD. Ωστόσο, η πρακτική σύγκλιση εξακολουθεί να εξαρτάται από τον αριθμό των βημάτων και των υποβημάτων. Το βασικό πλεονέκτημα της XPBD είναι ότι η φυσική παράμετρος της δυσκαμψίας διατηρεί την ίδια ερμηνεία, ακόμη και όταν αλλάζουν οι αριθμητικές παράμετροι της επίλυσης [4][5].

1.3.3 Σύγκριση PBD και XPBD

Μπορεί να γίνει αντιληπτή η διαφορά μεταξύ των δύο μεθόδων με ένα πρακτικό παράδειγμα. Έστω ένας περιορισμός απόστασης ο οποίος αναπαριστά ένα ελατήριο. Το ελατήριο έχει σταθερά δυσκαμψίας.

Στη μέθοδο PBD, η δυσκαμψία κλιμακώνει την διόρθωση των θέσεων των σωματιδίων με κάθε επανάληψη

$$\Delta x_a = + \frac{w_a}{w_a + w_b} \cdot k' \cdot C_j \cdot \hat{n}_j$$

$$\Delta x_b = - \frac{w_b}{w_a + w_b} \cdot k' \cdot C_j \cdot \hat{n}_j$$

Όπως αναφέρθηκε σε προηγούμενα κεφάλαια, η σταθερά αυτή δεν αντιστοιχεί σε πραγματικό φυσικό μέγεθος. Η δυσκαμψία που εφαρμόζεται στο σύστημα εξαρτάται από τον αριθμό επαναλήψεων και τα χρονικά υποβήματα. Το αποτέλεσμα είναι ένα σύστημα βαριά βασιζόμενο στη ευρετική μέθοδο με μηδενική μεταφορά των παραμέτρων μεταξύ διαφορετικών σεναρίων επίλυσης.

Ωστόσο, στη μέθοδο XPBD, η δυσκαμψία μεταφράζεται άμεσα σε φυσικές μονάδες (για περιορισμούς απόστασης) και η χρονική κανονικοποίηση ενσωματώνεται στον επιλυτή μέσω της μεταβλητής ενδοτικότητας

$$\tilde{\alpha} = \frac{1}{k \Delta t_{\text{sub}}^2}$$

Η διόρθωση υπολογίζεται ως:

$$\Delta \lambda_j = \frac{-C_j - \tilde{\alpha}_j \lambda_j}{w_a + w_b + \tilde{\alpha}_j}$$

Ο παρονομαστής είναι ο 'ρυθμιστής' του συστήματος διότι καθορίζει το μέγεθος διόρθωσης περιορισμών. Όταν ο αριθμός των επαναλήψεων ή υποβημάτων υπέστη αλλαγή, αντικατοπτρίζεται στην εξίσωση. Ο επιλυτής μπορεί να μην έχει συγκλίνει τελείως, ωστόσο η ερμηνεία της μεταβλητής k παραμένει σταθερή γιατί το σύστημα εξαρτάται από την ενδοτικότητα και όχι από τον αριθμό βημάτων για να επιβάλει την σωστή ποσότητα ευκαμψίας.

Η μέθοδος PBD θα μπορούσε να θεωρηθεί μια οριακή περίπτωση της μεθόδου XPBD. Όταν η ενδοτικότητα τείνει στο μηδέν ($\alpha \rightarrow 0$, δηλαδή $k \rightarrow \infty$), ο όρος $\tilde{\alpha}$ εξαφανίζεται από τον παρονομαστή και η εξίσωση ανάγεται σε:

$$\Delta\lambda_j \approx \frac{-C_j}{w_a + w_b}$$

που αντιστοιχεί στην κλασική PBD προβολή με $k' = 1$. Η XPBD γενικεύει αυτή τη συμπεριφορά, επιτρέποντας ενδοτικούς περιορισμούς με φυσικά ερμηνεύσιμες παραμέτρους. Παρέχει μια σαφέστερη σύνδεση μεταξύ των αριθμητικών διορθώσεων και των φυσικών δυνάμεων μέσω της ερμηνείας Gauss-Seidel στο σύστημα KKT [4][5].

1.4 Μαλακά σώματα σε πραγματικό Χρόνο

Η προσομοίωση μαλακών σωμάτων σε πραγματικό χρόνο θεωρείται από τα πιο απαιτητικά προβλήματα στο τομέα της υπολογιστικής φυσικής. Η κύρια διαφορά τους με τα άκαμπτα σώματα έχει να κάνει με την προσδιόριση τους. Τα άκαμπτα σώματα χαρακτηρίζονται από θέση και προσανατολισμό του αντικειμένου. Μια τέτοια προσομοίωση δεν είναι αρκετή για τα μαλακά σώματα. Λόγω της περίπλοκης συμπεριφορά τους, έχουν ανάγκη από δυναμικές μεταβλητές που καθορίζουν τους εσωτερικούς βαθμούς ελευθερίας κίνησης τους. Ωστόσο, η συγκεκριμένη προσέγγιση αυξάνει σε μεγάλο βαθμό τον αριθμό πολυπλοκότητας και κατ' επέκταση το υπολογιστικό κόστος.

Στο πλαίσιο των μεθόδων βασισμένων σε θέσεις, κάθε σώμα μοντελοποιείται από ένα δίκτυο σωματιδίων που συνδέονται με περιορισμούς. Πολλές μέθοδοι αναπαράστασης των συγκεκριμένων σωματιδίων έχουν καταφέρει να εισάγουν προβλεψιμότητα και σταθερότητα στην δυναμική συμπεριφορά των παραμορφώσιμων αντικειμένων

1.4.1 Μοντελοποίηση με σωματίδια και ελατήρια πλέγματος

Μια από τις πιο διαδεδομένες αναπαραστάσεις σωματιδίων στον χώρο των σωμάτων βασισμένα σε θέσεις είναι το μοντέλο μάζας-ελατηρίου [9]. Το σώμα διακριτοποιείται σε ένα σύνολο N σωματιδίων, καθένα από τα οποία χαρακτηρίζεται από τη θέση x_i , την ταχύτητα v_i και την αντίστροφη μάζα. Τα σωματίδια διατάσσονται σε κανονικό πλέγμα με απόσταση s μεταξύ γειτονικών κόμβων.

Για να διατηρηθεί η δομική ακεραιότητα του πλέγματος, επιβάλλεται ο περιορισμός απόστασης. Για κάθε ζεύγος συνδεδεμένων σωματιδίων (a, b) , ο περιορισμός ορίζεται ως:

$$C_j(x) = |x_b - x_a| - d_j$$

όπου d_j είναι το μήκος ηρεμίας της σύνδεσης. Αναπαριστά την απόσταση στην αρχική, μη παραμορφωμένη κατάσταση. Σε ένα δυσδιάστατο πλέγμα που αποτελείται από $W \times H$ σωματίδια, μπορούμε να διακρίνουμε δύο κατηγορίες περιορισμών βάσει της γεωμετρικής τους φύσης.

Οι δομικοί περιορισμοί, συνδέουν κάθε σωματίδιο με τους 4 άμεσους γείτονες του. Οποιαδήποτε κατακόρυφη ή οριζόντια σύνδεση οφείλεται σε αυτούς. Το μήκος ηρεμίας τους είναι $d = s$ και ο αριθμός τους είναι:

$$N_{\text{struct}} = W(H - 1) + H(W - 1)$$

Αυτοί οι περιορισμοί είναι υπεύθυνοι για την διατήρηση των βασικών αναλογιών του σώματος και μπορούν να αντισταθούν σε εφελκυστική και θλιπτική παραμόρφωση.

Από την άλλη πλευρά, συναντιούνται οι διατμητικοί περιορισμοί. Συνδέουν κάθε σωματίδιο με τον διαγώνιο γείτονα του. Το μήκος ηρεμίας τους είναι $d = s\sqrt{2}$ και ο αριθμός τους είναι:

$$N_{\text{shear}} = 2(W - 1)(H - 1)$$

Χωρίς αυτούς τους περιορισμούς, το σώμα θα κατέρρεε διαγωνίως λόγω της έλλειψης αντίστασης.

Ο συνολικός αριθμός όλων των περιορισμών μπορεί να αναλυθεί για ένα πλέγμα $W \times H$ ως εξής:

$$N_c = N_{\text{struct}} + N_{\text{shear}} = W(H - 1) + H(W - 1) + 2(W - 1)(H - 1)$$

Σε κάθε χρονικό βήμα, η προσομοίωση ακολουθεί πιστά τον αλγόριθμο XPBD. Αρχικά υπολογίζονται οι προβλεπόμενες θέσεις των σωματιδίων με ολοκλήρωση symplectic Euler

$$\tilde{x}_i = x_i + v_i \Delta t_{\text{sub}} + g \Delta t_{\text{sub}}^2$$

όπου g είναι η βαρυτική επιτάχυνση, και στη συνέχεια οι θέσεις διορθώνονται μέσω επαναληπτικής επίλυσης των περιορισμών. Μετά τη σύγκλιση, οι ταχύτητες ενημερώνονται με τις καινούργιες

$$v_i = \frac{x_i - x_i^{\text{prev}}}{\Delta t_{\text{sub}}}$$

Η διαδικασία αυτή επαναλαμβάνεται για κάθε υποβήμα, εξασφαλίζοντας ότι η χρονική κανονικοποίηση αντικατοπτρίζει σωστά τη φυσική κλίμακα χρόνου.

1.4.2 Ιεραρχική επίλυση περιορισμών

Η σειρά με την οποία ο επιλυτής επεξεργάζεται τους περιορισμούς έχει άμεση επίπτωση στο αποτέλεσμα και στην ποσότητα σύγκλισης της προσομοίωσης. Όταν προσομοιώνεται ένα σύστημα μαλακού σώματος, δεν έχουν όλοι οι περιορισμοί την ίδια βαρύτητα. Στην συγκεκριμένη μοντελοποίηση πλέγματος της προηγούμενης ενότητας, οι δομικοί περιορισμοί εξασφαλίζουν την βασική συνοχή του σώματος ενώ οι διατμητικοί αντιστέκονται στις παραμορφώσεις. Η μέθοδος ιεραρχικής επίλυσης HP-XPBD, εκμεταλλεύεται αυτή τη λεπτομέρεια, εκτελώντας κάθε περιορισμό ανάλογα με την βαρύτητα που του αντιστοιχεί.

Ο κάθε περιορισμός j αντιστοιχίζεται σε ένα επίπεδο προτεραιότητας $p_j \in \{0,1,2, \dots\}$, όπου $p = 0$ αντιστοιχεί στην υψηλότερη προτεραιότητα. Στην περίπτωση του δισδιάστατου πλέγματος:

- **Επίπεδο 0:** Οριζόντιοι και κατακόρυφοι περιορισμοί
- **Επίπεδο 1:** Διαγώνιοι περιορισμοί

Σε κάθε βήμα του επιλυτή, οι περιορισμοί επιλύονται διαδοχικά ανά επίπεδο

for $p = 0, 1, \dots, p_{\max}$: solve all j with $p_j = p$

Η προσέγγιση αυτή εξασφαλίζει ότι οι δομικοί περιορισμοί, οι οποίοι αποτελούν τον κύριο μηχανισμό μεταφοράς δυνάμεων μέσα στο σώμα, ικανοποιούνται πρώτοι σε κάθε επανάληψη. Οι διατμητικοί περιορισμοί εφαρμόζονται στη συνέχεια, λαμβάνοντας υπόψη τις θέσεις που έχουν ήδη διορθωθεί από τους δομικούς. Με αυτόν τον τρόπο, μειώνεται η αλληλεπίδραση μεταξύ περιορισμών διαφορετικής φυσικής σημασίας.

Σε αντίθεση, η επίπεδη επίλυση επεξεργάζεται όλους τους περιορισμούς σε μία ενιαία ακολουθία, χωρίς διάκριση προτεραιότητας:

for $j = 0, 1, \dots, N_C - 1$: solve j

Η σειρά εξαρτάται αποκλειστικά από τη σειρά εισαγωγής στη δομή δεδομένων. Αυτό μπορεί να οδηγήσει σε καταστάσεις όπου ένας διατμητικός περιορισμός «αναιρεί» εν μέρει τη διόρθωση ενός δομικού που μόλις εφαρμόστηκε, επιβραδύνοντας τη σύγκλιση. Αυτή η συνθήκη οδηγεί σε διαγωνισμό πόρων.

Η ιεραρχική προσέγγιση δεν αλλάζει τη μαθηματική θεμελίωση της μεθόδου XPBD ο τύπος υπολογισμού του Δl_j παραμένει ο ίδιος. Αυτό που μεταβάλλεται είναι η σειρά επίσκεψης στη μέθοδο Gauss-Seidel, η οποία επηρεάζει τον ρυθμό σύγκλισης αλλά όχι το σημείο ισορροπίας. Θεωρητικά, με άπειρες επαναλήψεις, και οι δύο στρατηγικές θα συνέκλιναν στην ίδια λύση. Στην πράξη όμως, με περιορισμένο αριθμό βημάτων, η ιεραρχική επίλυση μπορεί να προσφέρει καλύτερη διατήρηση του σχήματος και μειωμένο σφάλμα παραβίασης περιορισμών, ιδιαίτερα σε σενάρια μεγάλων παραμορφώσεων όπου η δομική ακεραιότητα απειλείται [10].

1.5 Σχετικές εργασίες

Εδώ και αρκετές δεκαετίες, η προσομοίωση παραμορφώσιμων σωμάτων σε πραγματικό χρόνο, εξετάζεται ενεργά από την επιστημονική κοινότητα. Σε γενικό πλαίσιο, οι μελέτες που ερευνούν

παραμορφώσιμα σώματα, μπορούν να κατηγοριοποιηθούν σε τέσσερις κύριους άξονες: μεθόδους βασισμένες σε θέση, μεθόδους πεπερασμένων στοιχείων, ενοποιημένα πλαίσια σωματιδίων, και υλοποιήσεις σε μηχανές παιχνιδιών.

1.5.1 Μέθοδοι βασισμένες σε θέση

Η πλέον θεμελιώδης έρευνα του Müller et al. [3] εισήγαγε τη μέθοδο Position-Based Dynamics. Η συγκεκριμένη μέθοδος αντικατέστησε τα συστήματα που έκαναν χρήση της επίλυσης δυνάμεων των σωματιδίων, με την επαναληπτική προβολή γεωμετρικών περιορισμών. Η μέθοδος υιοθετήθηκε αρκετά γρήγορα από την επιστημονική κοινότητα, ωστόσο επιβαρυνόταν διότι η αποτελεσματική δυσκαμψία ενός σώματος αποτελούταν από την αριθμό βημάτων της προσομοίωσης. Για να αντιμετωπιστεί αυτό το πρόβλημα, η μέθοδος XPBD εισήγαγε τον συσσωρευτικό πολλαπλασιαστή Lagrange και την ενδοτικότητα στις παραμέτρους του συστήματος. Με αυτόν τον τρόπο, εξασφαλιζόταν η ακεραιότητα της παραμέτρου δυσκαμψίας και μπορούσε να διατηρηθεί μια σταθερή ερμηνεία ανεξάρτητα από τον προεπιλεγμένο αριθμό βημάτων.

Πριν από την XPBD, οι Goldenthal et al. [6] είχαν ήδη προτείνει τη χρήση πολλαπλασιαστών Lagrange για την αυστηρή ικανοποίηση περιορισμών σε προσομοιώσεις υφάσματος, αποδεικνύοντας ότι η ακριβής επίλυση ήταν εφικτή σε πραγματικό χρόνο. Οι Tournier et al. [7] διεύρυναν αυτή την κατεύθυνση, παρουσιάζοντας ένα πλαίσιο σταθερής δυναμικής υπό περιορισμούς (stable constrained dynamics) που συνέδεσε ρητά τη μηχανική ενέργεια με τη διαδικασία προβολής. Οι δύο αυτές εργασίες αποτέλεσαν βασικές επιρροές για τη μαθηματική θεμελίωση της XPBD.

1.5.2 Μέθοδοι πεπερασμένων στοιχείων

Σε αντίθεση με τις μεθόδους βασισμένες σε θέσεις, η μέθοδος πεπερασμένων στοιχείων FEM, παρέχει θεμελίωση μέσω της διακριτοποίησης των μερικών διαφορικών εξισώσεων της ελαστικότητας. Οι Sifakis και Barbič [2] έχουν παρουσιάσει ολοκληρωμένο οδηγό. Σε αυτόν τον οδηγό αναγράφονται στρατηγικές προσομοίωσης τρισδιάστατων παραμορφώσιμων σωμάτων, καλύπτοντας θέματα θεωρίας, διακριτοποίησης και μοντελοποίησης. Όμως, η μέθοδος FEM, απαιτεί παραπάνω υπολογιστικούς πόρους σε σχέση με άλλες μεθόδους, και έτσι καθιστάτε κατώτερη σε θέματα προσομοίωσης μαλακών σωμάτων σε πραγματικό χρόνο.

Μια απόπειρα γεφύρωσης μεταξύ των δύο μεθόδων έγινε από τους Müller et al. [12], οι οποίοι κατάφεραν να υπολογίσουν βέλτιστη άκαμπτη μετατόπιση για κάθε σωματίδιο βάσει της γειτονιάς του. Η μέθοδος αυτή αποφεύγει εντελώς τον ορισμό πλέγματος και περιορισμών, αλλά η φυσική ακρίβειά της είναι περιορισμένη σε σύγκριση με τη FEM και την XPBD.

Από την άλλη, η μέθοδος Projective Dynamics από τους Bouaziz et al. [13], επιχειρεί να μετασχηματίσει το πρόβλημα ελαχιστοποίησης σε σταθερό πίνακα, επιτρέποντας την γρήγορη επίλυση σε κάθε χρονικό βήμα.

1.5.3 Ενοποιημένα πλαίσια σωματιδίων

Η NVIDIA ενσωμάτωσε XPBD στο σύστημα FleX [8], ένα ενοποιημένο πλαίσιο προσομοίωσης σωματιδίων που χειρίζεται ταυτόχρονα άκαμπτα σώματα, μαλακά σώματα, υφάσματα και ρευστά μέσω ενός κοινού επιλυτή. Η εργασία των Macklin et al. [8] έδειξε ότι η Position-Based προσέγγιση μπορεί να κλιμακωθεί σε σύνθετα σενάρια πολλαπλών φυσικών φαινομένων, αξιοποιώντας τον παραλληλισμό της GPU.

Στον τομέα των ρευστών, οι Bender και Koschier [14] παρουσίασαν την εφαρμογή XPBD σε Smoothed Particle Hydrodynamics (SPH), αποδεικνύοντας ότι η μέθοδος μπορεί να διαχειριστεί περιορισμούς ασυμπίεστου (incompressibility constraints) σε προσομοιώσεις ρευστών πραγματικού χρόνου. Η εργασία αυτή υπογραμμίζει τη γενικότητα του πλαισίου XPBD πέρα από τα μαλακά σώματα.

1.5.4 Υλοποιήσεις σε μηχανές παιχνιδιών

Η ενσωμάτωση φυσικής μαλακών σωμάτων σε μηχανές παιχνιδιών παρουσιάζει ιδιαίτερες προκλήσεις. Η Nealen et al. [1] κατέγραψαν την ευρεία ανασκόπηση των φυσικά βασισμένων μοντέλων παραμόρφωσης στα γραφικά υπολογιστών, αναδεικνύοντας το χάσμα μεταξύ ακαδημαϊκών μεθόδων και πρακτικών υλοποιήσεων. Σύγχρονες μηχανές όπως η Unreal Engine και η Unity παρέχουν υποστήριξη για μαλακά σώματα κυρίως μέσω εξωτερικών βιβλιοθηκών (π.χ. Chaos, Havok), οι οποίες χρησιμοποιούν ποικίλες μεθόδους από FEM μέχρι PBD.

Η μηχανή Godot, στην οποία αναπτύσσεται η παρούσα εργασία, δεν παρέχει ενσωματωμένη υποστήριξη για μαλακά σώματα σε δύο διαστάσεις. Η υλοποίηση ενός XPBD solver ως GDExtension plugin καλύπτει αυτό το κενό, προσφέροντας φυσικά παραμετροποιήσιμη προσομοίωση μαλακών σωμάτων απευθείας μέσα στο περιβάλλον ανάπτυξης. Η αρχιτεκτονική GDExtension, βασισμένη στη βιβλιοθήκη godot-cpp [15], επιτρέπει τη σύνταξη κώδικα C++ που εκτελείται με native ταχύτητα ενώ διατηρεί πλήρη πρόσβαση στο API της μηχανής.

Κεφάλαιο 2ο: Αρχιτεκτονική και σχεδίαση του plugin

2.1 Εισαγωγή

Παρόλο που πλέον έχει γίνει μαθηματική θεμελίωση των μεθόδων XPBD και HP-XPBD, η πλήρη υλοποίηση τους είναι εξίσου μεγάλη πρόκληση. Απαιτούνται διάφορες αρχιτεκτονικές αποφάσεις που καθορίζουν αν το τελικό αποτέλεσμα θα είναι ένα λειτουργικό, εύκολο στην χρήση και επαναχρησιμοποιούμενο εργαλείο. Οι πιο σημαντικές απορίες που πρέπει να λυθούν είναι οι εξής:

- Ποιο σύστημα θα φιλοξενήσει την προσομοίωση
- Με ποιόν τρόπο θα ενσωματωθεί ο κώδικας
- Τι γλώσσα προγραμματισμού θα χρησιμοποιηθεί
- Πως θα οργανωθούν τα δεδομένα
- Πως και ποιοι παράμετροι θα εκτεθούν στον χρήστη
- Πως θα υπάρξει ισορροπία μεταξύ έρευνας και χρηστικού λογισμικού

Στην παρούσα εργασία, ως πλατφόρμα ανάπτυξης επιλέχθηκε η μηχανή Godot (έκδοση 4.5), μια ανοιχτού κώδικα μηχανή παιχνιδιών που υποστηρίζει τόσο δυσδιάστατα όσο και τρισδιάστα γραφικά. Η μηχανή Godot προσφέρει ένα σύστημα επεκτάσεων, το GDExtension, που επιτρέπει τη σύνδεση εξωτερικού κώδικα C++ χωρίς τροποποίηση του πυρήνα της μηχανής, προσφέροντας ταυτόχρονα πρόσβαση στο πλήρες API κόμβων, σκηνών και πόρων. Η επιλογή αυτή καθορίστηκε από τρεις παράγοντες

- α) Η ανοιχτή αρχιτεκτονική της Godot καθιστά τη μηχανή ιδανική για ακαδημαϊκή έρευνα,
- β) Το GDExtension εξασφαλίζει απόδοση native C++ χωρίς binding overhead
- γ) Η κοινότητα της Godot αναπτύσσεται ταχύτατα, γεγονός που αυξάνει τη δυνητική χρησιμότητα του plugin

Για λόγους ευχρηστίας, το plugin, υλοποιήθηκε ως μία ενιαία κλάση. Η κλάση ονομάζεται *Softbody2D*, και κληρονομεί από τον κόμβο *Node2D* της Godot. Ο κόμβος *Node2D* παρέχει πλήρες ενσωμάτωση με την λογική της μηχανής και διαθέτει όλα τα απαραίτητα εργαλεία έτσι ώστε ο χρήστης να έχει μια ομαλή εμπειρία χωρίς να χρειαστεί να χρησιμοποιήσει την μηχανή με διαφορετικό τρόπο. Η κλάση περιέχει σωματίδια και περιορισμούς. Μέσα στον ίδιο τον επιλυτή, υπάρχει η δυνατότητα επίλυσης με δύο διαφορετικούς τρόπους. Τον κλασικό (Flat XPBD) και τον ιεραρχικό (HP-XPBD). Επιπλέον, διατίθεται ένα σύστημα καταγραφής και σύγκρισης δεδομένων μεταξύ των δύο τρόπων επίλυσης. Οι παράμετροι των μεθόδων, εκτίθενται στο χρήστη μέσω του Inspector.

Ο κώδικας είναι γραμμένος σε C++ και ενσωματώθηκε μέσω του μηχανισμού GDExtension[15]. Μέσω των διεπαφών του, επιτρέπεται η δημιουργία καινούργιων κόμβων χωρίς αλλαγή στον πυρήνα της μηχανής. Η γλώσσα C++, εξασφαλίζει υψηλή υπολογιστική απόδοση που απαιτεί ο επιλυτής σωματιδίων, ενώ παράλληλα το GDExtension προσφέρει αυτόματη ενσωμάτωση.

2.2 Μηχανή Godot και GDExtension

Η μηχανή Godot, είναι ανοιχτού κώδικα (άδεια MIT). Αναπτύσσεται ενεργά από το 2014, και μπορεί να υποστηρίξει δυσδιάστατα αλλά και τρισδιάστατα γραφικά. Σε αντίθεση με άλλες εμπορικές μηχανές, η Godot, παρέχει πλήρη πρόσβαση στον πηγαίο κώδικα χωρίς να επιβάλει οποιοδήποτε κόστος για την χρήση της. Η αρχιτεκτονική της, χτίζεται πάνω στο σύστημα κόμβων και σκηνών. Οποιοδήποτε αντικείμενο στην ιεραρχία της Godot, αναπαριστάτε από έναν κόμβο. Ανάλογα με το είδος του κόμβου, κάθε αντικείμενο έχει και ξεχωριστές ιδιότητες. Για παράδειγμα οι κόμβοι δύο διαστάσεων (Node2D), δεν περιέχουν λειτουργικότητα στον άξονα Z, σε αντίθεση με τους κόμβους τριών διαστάσεων (Node3D). Κάθε κόμβος, συνδέεται με οποιονδήποτε άλλον κόμβο σε ιεραρχία δέντρου. Μια σκηνή, αποτελείται από ιεραρχίες δέντρων φτιαγμένες από κόμβους.

Για τον προγραμματισμό, η Godot υποστηρίζει πολλαπλές γλώσσες: GDScript που είναι ειδικά σχεδιασμένη για τη μηχανή, C#, και μέσω του GDExtension οποιαδήποτε γλώσσα μπορεί να παράγει δυναμικές βιβλιοθήκες (C, C++, Rust, κ.ά.).

2.2.1 Αρχιτεκτονική GDExtension

Η παλαιότερη έκδοση του GDExtension, δεν επέτρεπε δημιουργία νέων τύπων κόμβων, χωρίς την τροποποίηση ή μεταγλώττιση του πυρήνα από τη μηχανή. Η σημερινή αρχιτεκτονική ωστόσο, παρέχει τα απαραίτητα εργαλεία για ομαλή δημιουργία και επεξεργασία κόμβων μέσω API. Η διεπαφή GDExtension γραμμένη στη γλώσσα προγραμματισμού C, ορίζει τον τρόπο επικοινωνίας μεταξύ της μηχανής και της εξωτερικής βιβλιοθήκης. Μέσω αυτής, η βιβλιοθήκη δηλώνει ποιες κλάσεις, μεθόδους και ιδιότητες θέλει να εγγράψει στη μηχανή. Η βιβλιοθήκη παρέχει επιπλέον μακροεντολές καταχώρησης τιμών. Με αυτές τις εντολές, υπάρχει η δυνατότητα δημιουργίας εξατομικευμένων κλάσεων και συναρτήσεων, που ενσωματώνονται άμεσα στη μηχανή. Τέλος, το αρχείο ρυθμίσεων `.gdextension`, ορίζει το σύμβολο εισόδου, τη κατώτατη συμβατή έκδοση, και τα μονοπάτια των μεταγλωττισμένων βιβλιοθηκών ανά πλατφόρμα και αρχιτεκτονική. Στην περίπτωση του plugin:

```
[configuration]
```

```
entry_symbol = "xpbdd_softbodies_init"  
compatibility_minimum = "4.5"
```

```
[libraries]
```

```
windows.debug.x86_64 = ". /bin/windows/xpbdd_softbodies.windows.template_de-  
bug.x86_64.single.dll"  
linux.debug.x86_64 = ". /bin/linux/libxpbdd_softbodies.linux.template_debug.x86_64.sin-  
gle.so"
```

Κατά τη φόρτωση, η Godot εντοπίζει το αρχείο *.gdextension*, φορτώνει τη βιβλιοθήκη για την τρέχουσα πλατφόρμα, και καλεί τη συνάρτηση εισόδου. Από εκεί και πέρα, οι καταχωρημένες κλάσεις είναι διαθέσιμες σαν να ήταν ενσωματωμένες στη μηχανή, εμφανίζονται στον Inspector, μπορούν να προστεθούν σε σκηνές, και αλληλοεπιδρούν με το υπόλοιπο σύστημα κόμβων.

2.2.2 Σύνδεση C++ με Godot

Η πρακτική σύνδεση μεταξύ κώδικα C++ και μηχανής Godot γίνεται σε δύο σημεία: τη δήλωση κλάσης και την εγγραφή μεθόδων/ιδιοτήτων.

Η δήλωση μιας νέας κλάσης απαιτεί κληρονομικότητα από μια υπάρχουσα κλάση της Godot και τη μακροεντολή *GDCLASS*

```
class SoftBody2D : public godot::Node2D {
    GDCLASS(SoftBody2D, godot::Node2D)
    // ...
};
```

Η μακροεντολή *GDCLASS* δημιουργεί αυτόματα τον απαραίτητο κώδικα (RTTI, virtual κλήσεις, binding metadata) ώστε η μηχανή να αναγνωρίζει τον τύπο κατά τη διάρκεια εκτέλεσης.

Η εγγραφή μεθόδων και ιδιοτήτων γίνεται στη στατική μέθοδο *_bind_methods()*, η οποία καλείται αυτόματα κατά την αρχικοποίηση:

```
void SoftBody2D::_bind_methods() {
    ClassDB::bind_method(D_METHOD("set_grid_stiffness", "value"),
        &SoftBody2D::set_grid_stiffness);

    ClassDB::bind_method(D_METHOD("get_grid_stiffness"),
        &SoftBody2D::get_grid_stiffness);

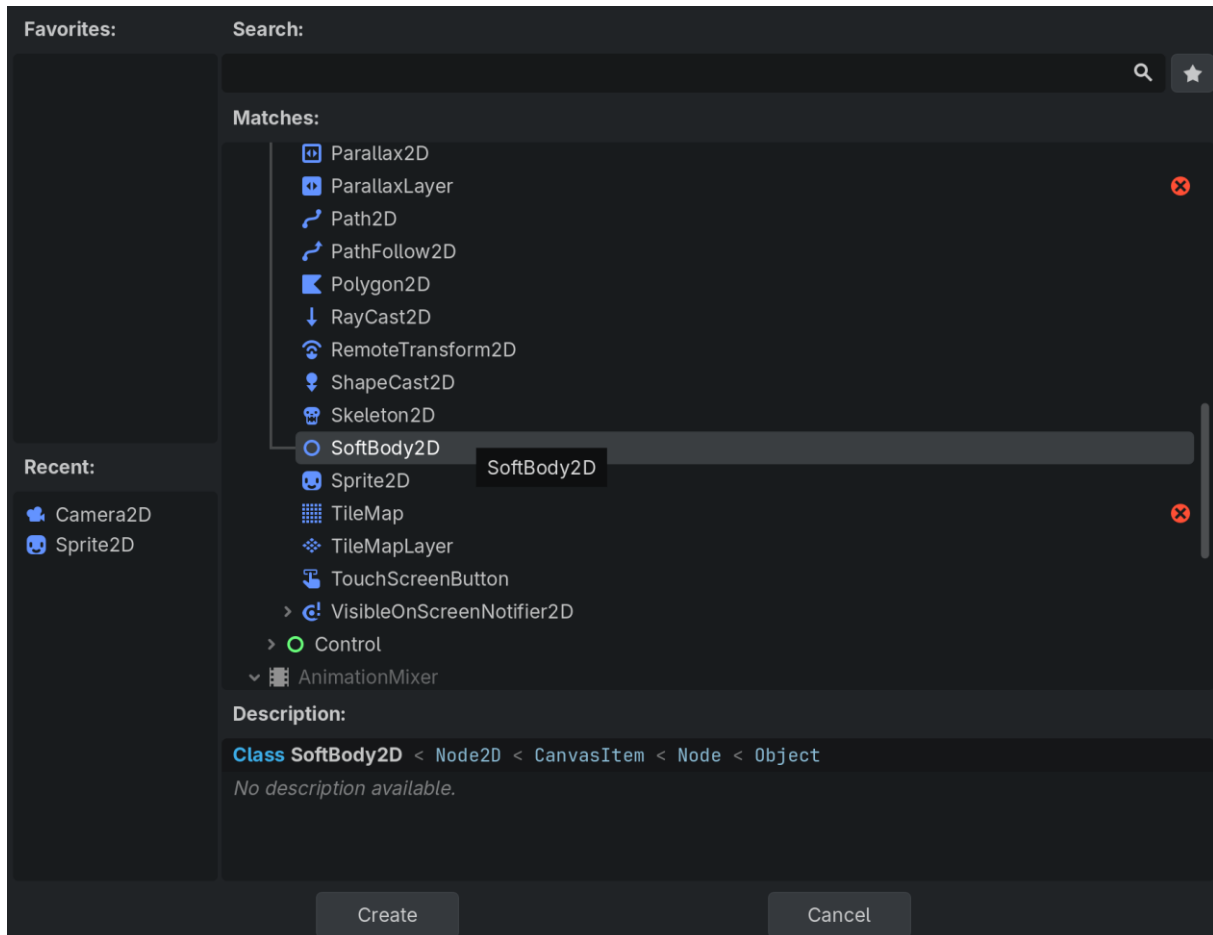
    ADD_PROPERTY(make_property_info(Variant::FLOAT, "grid_stiffness"),
        "set_grid_stiffness", "get_grid_stiffness");
}
```

Το μοτίβο getter/setter με *ADD_PROPERTY* είναι ο μηχανισμός που εκθέτει τις μεταβλητές C++ στον Inspector. Η μηχανή καλεί τον getter για εμφάνιση και τον setter όταν ο χρήστης αλλάξει μια τιμή. Αυτό εξασφαλίζει ότι η εσωτερική κατάσταση του solver ενημερώνεται πάντα μέσω ελεγχόμενων σημείων εισόδου, αποτρέποντας μη έγκυρες τιμές (π.χ. αρνητικά substeps).

Τέλος, η συνάρτηση εισόδου καταχωρεί τις κλάσεις στη μηχανή

```
void initialize_gdextension_types(ModuleInitializationLevel p_level) {
    if (p_level != MODULE_INITIALIZATION_LEVEL_SCENE) return;
    GDREGISTER_CLASS(SoftBody2D)
}
```

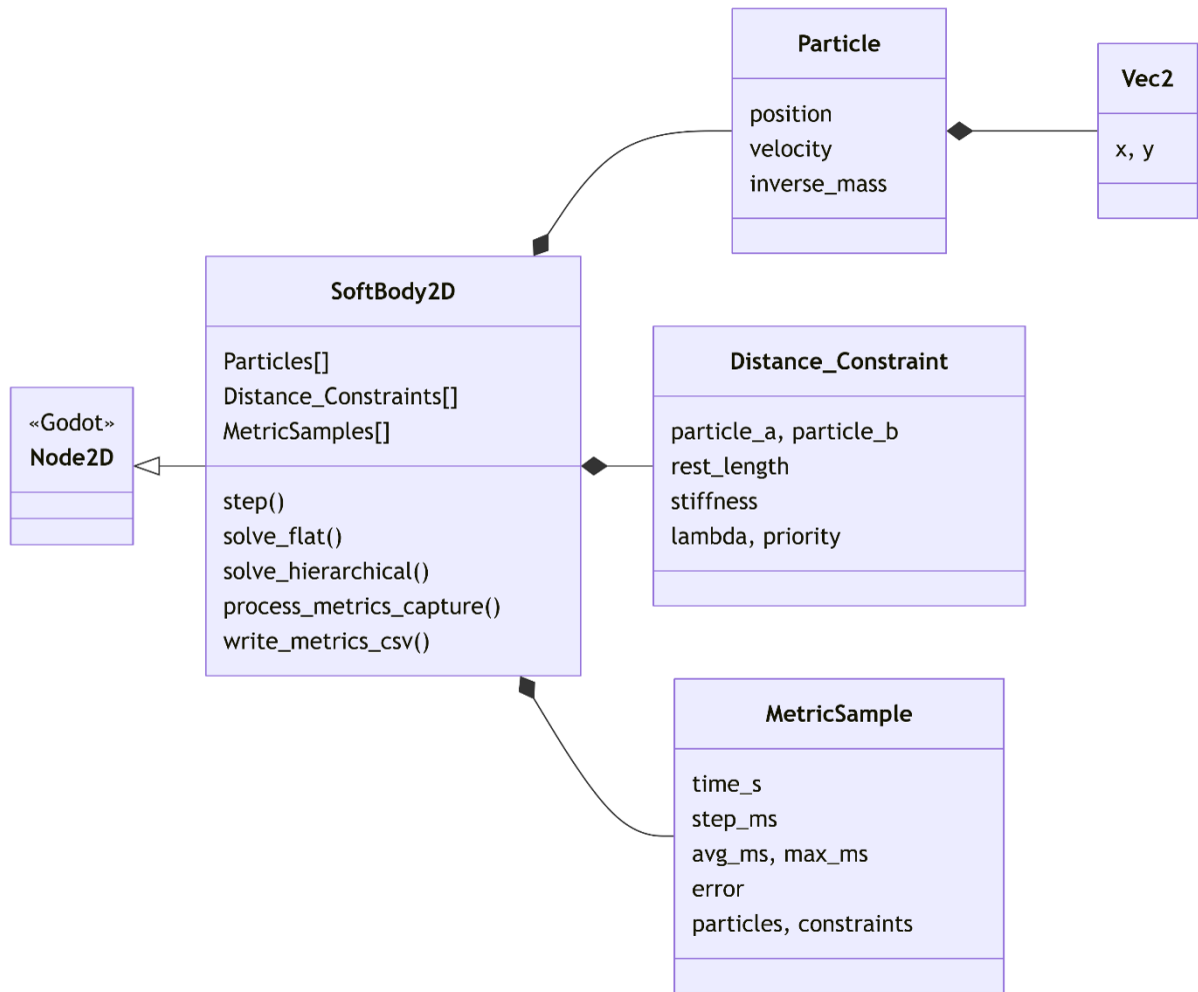
Μετά από αυτή τη διαδικασία, ο κόμβος *SoftBody2D* εμφανίζεται στη λίστα διαθέσιμων κόμβων της Godot και μπορεί να χρησιμοποιηθεί σε οποιοδήποτε project χωρίς επιπλέον ρύθμιση, αρκεί το αρχείο *.gdextension* και η αντίστοιχη δυναμική βιβλιοθήκη να βρίσκονται στον φάκελο του project.



Σχήμα 2.2.2: Εξατομικευμένη κλάση μέσα στον inspector της Godot

2.3 Δομή του plugin

Η οργάνωση του plugin είναι εντελώς επίπεδη. Δηλαδή, μια κεντρική κλάση είναι υπεύθυνη για ολόκληρη την ροή της προσομοίωσης, ενώ τρεις δομές αναπαριστούν όλα τα δεδομένα που χρειάζεται η προσομοίωση. Εξάλλου, τα πολύπλοκα μοτίβα σχεδίασης, θα πρέπει να εισάγονται σε λογισμικό, μόνο όταν άμεσα βελτιστοποιούν την αποδοτικότητα και την αναγνωσιμότητα [16].



Σχήμα 2.3: Δομή οντότητας *SoftBody2D*

Η κλάση *SoftBody2D* διαχωρίζεται σε τρία αρχεία. Αυτό το γεγονός, δεν αλλάζει την λογική δομή, εξακολουθεί να είναι μία οντότητα. Η επιλογή αυτή έγινε για ευκολότερη πλοήγηση από προγραμματιστές διότι η προσομοίωση, η αλληλεπίδραση με τη μηχανή Godot και η καταγραφή μετρήσεων βρίσκονται σε ξεχωριστά αρχεία.

2.3.1 Κλάση *SoftBody2D*

Ο πυρήνας του plugin βρίσκεται στη κλάση *Softbody2D*. Κληρονομεί από τη κλάση *Node2D* και περιέχει ολόκληρο τον κύκλο ζωής της προσομοίωσης, μαζί και με την τωρινή κατάσταση του.

```

class SoftBody2D : public godot::Node2D {
    GDCLASS(SoftBody2D, godot::Node2D)
private:
    std::vector<Particle> Particles;
    std::vector<Distance_Constraint> Distance_Constraints;
    std::vector<std::vector<int>>> priority_groups;
    int solver_mode = SOLVER_FLAT_XPBD;
}
    
```

```

int num_substeps = 10;
int solver_iterations = 1;
// ...
};

```

Όπως και με τους υπόλοιπους κόμβους στη Godot, έτσι και το *Softbody2D*, ακολουθεί τον ίδιο κύκλο εκτέλεσης με κάποιες διαφορές. Η συνάρτηση *_ready()*, είναι μια global συνάρτηση που καλείται στην αρχικοποίηση οποιουδήποτε κόμβου. Σε αυτή την συνάρτηση, αρχικοποιούνται όλοι οι παράμετροι, δημιουργείται το πλέγμα σωματιδίων και ξεκινάει η καταγραφή δεδομένων.

Στην global συνάρτηση *_physics_process(delta)*, εκτελείται ο επιλεγμένος επιλυτής του κόμβου. Αυτή η συνάρτηση καλείται κάθε καρέ της προσομοίωσης (τυπικά 60 φορές το δευτερόλεπτο), και εκτελεί τη μέθοδο *_step(delta)* η οποία αντιστοιχεί σε ένα βήμα της προσομοίωσης. Έπειτα από αρκετά βήματα εκτελείται ένας ολόκληρος κύκλος και ξεκινάει η εκτέλεση της συνάρτησης από την αρχή. Τέλος, η global συνάρτηση *_draw()* ευθύνεται για την εικονική αναπαράσταση των σωματιδίων και των περιορισμών που αποτελούν ένα σώμα *SoftBody2D*.

Η λειτουργία του επιλυτή επιλέγεται μέσω ενός enum

```

enum SolverMode {
    SOLVER_FLAT_XPBD = 0,
    SOLVER_HIERARCHICAL = 1,
};

```

Αυτό επιτρέπει εναλλαγή μεταξύ αλγορίθμων χωρίς αλλαγή στον υπόλοιπο κώδικα. Η συνάρτηση *step()* ελέγχει απλώς την τιμή του *solver_mode* και καλεί την αντίστοιχη επιλογή επιλυτή.

2.3.2 Δομή Particle

Κάθε σωματίδιο αναπαρίσταται από τη δομή *Particle*

```

struct Particle {
    Vec2 position;
    Vec2 prev_position;
    Vec2 velocity;
    double inverse_mass;
};

```

Κάθε πεδίο έχει μεταφραστεί ένα προς ένα από την μέθοδο XPBD. Η τρέχουσα θέση *position*, τροποποιείται κατά τις φάσεις πρόβλεψης και προβολής περιορισμών. Η θέση πριν την πρόβλεψη *prev_position* χρησιμεύει στον υπολογισμό ταχύτητας μετά το πέρας του υποβήματος. Η ταχύτητα *velocity* χρησιμοποιείται μόνο στη φάση πρόβλεψης για τον υπολογισμό της αρχικής μετατόπισης και τέλος το αντίστροφο βάρος *inverse_mass*. Η χρήση αντιστρόφου αντί του βάρους αποφεύγει διαιρέσεις κατά τον υπολογισμό διορθώσεων, και επιτρέπει σωματίδια με $w = 0$ (άπειρη μάζα = ακίνητα σημεία).

2.3.3 Δομή Distance_Constraint

Κάθε περιορισμός αναπαρίσταται από τη δομή *Distance_Constraint*

```
struct Distance_Constraint {
    int particle_a;
    int particle_b;
    double rest_length;
    double stiffness;
    double lambda;
    int priority;
};
```

Όπως και πριν, η δομή είναι μια πιστή μετάφραση της μεθόδου XBPD. Ωστόσο, υπάρχουν δύο αξιοσημείωτες λεπτομέρειες σε αυτή τη δομή. Αρχικά, σε κάθε περιορισμό αντιστοιχούν δύο ακέραιοι δείκτες οι οποίοι αποθηκεύουν τη θέση του επομένου ή προηγούμενου σωματιδίου. Οι δείκτες αυτοί είναι αναγκαίοι γιατί σε περίπτωση ανακατάταξης της μνήμης, χρειαζόμαστε τη διεύθυνση των σωματιδίων που επηρεάζονται άμεσα από τον περιορισμό. Επιπλέον, υπάρχει η ιδιότητα προτεραιότητας *priority*. Η συγκεκριμένη ιδιότητα εφαρμόζεται μόνο στον αλγόριθμο ιεραρχικής επίλυσης και αποθηκεύει το μέγεθος της προτεραιότητας κάθε περιορισμού.

2.3.4 Βοηθητική κλάση Vec2

Η δομή *Vec2* είναι μια minimal υλοποίηση δισδιάστατου διανύσματος με υπερφόρτωση τελεστών

```
struct Vec2 {
    double x, y;
    Vec2 operator+(const Vec2& other) const;
    Vec2 operator-(const Vec2& other) const;
    Vec2 operator*(double scalar) const;
    Vec2 operator/(double scalar) const;
};
```

Παρόλο που η Godot έχει ήδη υπάρχουσα υλοποίηση *Vector2* κατά τη διάρκεια της ανάπτυξης της εφαρμογής παρουσιάστηκαν δύο προβλήματα. Αρχικά, οι ήδη υπάρχον υπερφορτώσεις δεν ήταν αρκετές και επομένως έπρεπε διαρκώς να δημιουργώ δικές μου για να είναι συμβατές με τον κώδικα και επιπλέον η δομή *Vector2* της Godot, χρησιμοποιεί μεταβλητή τύπου float 32-bit. Η δική μας είναι ωστόσο double 64-bit, και επιτρέπει πολύ μεγαλύτερη αριθμητική ακρίβεια κατά τη σώρευση μικρών διορθώσεων σε πολλές επαναλήψεις μια κρίσιμη ιδιότητα για τη σύγκλιση του XPBD επιλυτή.

2.4 Σύστημα Build

Για να αναπτυχθεί ένα GDExtension, είναι αναγκαίο να συνδεθεί ο πηγαίος κώδικας με την βιβλιοθήκη godot-cpp. Δεν αρκεί μια απλή μεταγλώττιση των αρχείων C++. Η δυναμική βιβλιοθήκη περιέχει χιλιάδες αρχεία header για κάθε κλάση ενώ ταυτόχρονα πρέπει να μεταγλωττιστεί με συγκεκριμένα compiler flags, σωστή σειρά σύνδεσης και ονοματολογία που αντιστοιχεί στις καταχωρήσεις του αρχείου. Αν δεν υπήρχε ένα τέτοιο σύστημα build, θα ήταν αναγκαία η χειροκίνητη διαχείριση όλων αυτών των αρχείων από τον προγραμματιστή, πράγμα μη εφικτό.

Η βιβλιοθήκη godot-cpp ωστόσο, μας παρέχει με το δικό της σύστημα build που ονομάζεται SConstruct. Το συγκεκριμένο σύστημα, ρυθμίζει αυτόματα τον compiler ανά πλατφόρμα, ορίζει τα απαραίτητα include paths, και παράγει τη στατική βιβλιοθήκη libgodot-cpp. Η χρήση εναλλακτικών συστημάτων όπως CMake ή Meson θα απαιτούσε επιπλέον ρύθμιση χωρίς καθαρό όφελος.

Υπάρχουν αρκετά θετικά με την χρήση SConstruct. Αρχικά, μόνο μια εντολή αρκεί για την μεταγλώττιση χωρίς παραπάνω βήματα. Επιπλέον, γίνεται αυτόματη ανίχνευση για αλλαγές. Κάθε φορά που ξεκινάει το build, χτίζονται μόνο τα αρχεία που έχουν αλλάξει και τίποτα άλλο εφόσον έχει χτιστεί η εφαρμογή ήδη μια φορά. Αυτό μας εξυπηρετεί ιδιαίτερα διότι σε κάθε αλλαγή το build διαρκεί μόνο δευτερόλεπτα. Τέλος, το SConstruct υποστηρίζει cross-platform ιδιότητες. Μπορεί να τρέξει το συγκεκριμένο plugin σε οποιαδήποτε πλατφόρμα έχει επιλεχθεί. Ολόκληρη η διαδικασία μεταγλώττισης γίνεται με μία εντολή

```
scons platform=windows target=template_debug
```

Εσωτερικά, υποστηρίζονται και οι ρυθμίσεις build. Με κάθε διαφορετική ρύθμιση, μεταγλωττίζονται διαφορετικές κατηγορίες κλάσεων. Για παράδειγμα, στο παρόν plugin το οποίο είναι δισδιάστατο, χρειάζονται μόνο οι δισδιάστατες κλάσεις, επομένως θα επιλεχθεί το αντίστοιχο προφίλ.

Πέρα από όλη την διαδικασία μεταγλώττισης και build, έχει αναπτυχθεί και βοηθητικό script. Το *compile_debug_build*, αυτοματοποιεί ολόκληρη την διαδικασία. Εκτελείται η εντολή Scons με όλες τις απαραίτητες παραμέτρους και αντιγράφει όλες τις μεταγλωττισμένες βιβλιοθήκες κατευθείαν στον τρέχον project. Με αυτόν τον τρόπο, ο προγραμματιστής δεν χρειάζεται να κάνει τίποτα παραπάνω, πέρα από το να τρέξει το βοηθητικό script, και να επιλέξει τις σωστές παραμέτρους. Στο τέλος όλων των διαδικασιών, εκκινεί αυτόματα τη μηχανή Godot, και όλες οι αλλαγές είναι έτοιμες για δοκιμή.

```
Current Plugin Name: xpb_d_softbodies
Current Targeted Godot Version: 4.5

Choose an option:
1. Change Godot Target Version
2. Change Build Profile
3. Rename Plugin
4. Compile Plugin Debug Build
5. Generate Missing XML Documentation Files
Enter your choice (1-5), or 'q' to quit:
█
```

Σχήμα 2.4: Οθόνη επιλογών βοηθητικού script

Κεφάλαιο 3ο: Υλοποίηση του επιλυτή

3.1 Αλγόριθμος προσομοίωσης

Έχει ήδη παρουσιαστεί η δομή και η αρχιτεκτονική του plugin, ωστόσο η υλοποίηση σε πραγματικό περιβάλλον και η ένταξη με τη μηχανή Godot είναι διαφορετικές προκλήσεις. Η υλοποίηση ακολουθεί πιστά τις προαναφερόμενες μαθηματικές θεμελιώσεις και διατυπώσεις, ωστόσο η μετάβαση από εξισώσεις σε κώδικα δεν είναι τετριμμένη. Υπάρχουν πολλές παράμετροι που επηρεάζουν το τελικό αποτέλεσμα όπως πρακτικές αποφάσεις για διαμοιρασμό πόρων, οργάνωση της λογικής ή ακόμα και η σειρά εκτέλεσης. Οι περισσότερες αποφάσεις έχουν ληφθεί με την εξυπηρέτηση της κεντρικής μεθόδου XPBD ως το κύριο κίνητρο.

Ο αλγόριθμος XPBD, εκτελείτε στην μέθοδο *step(delta)* που αντιπροσωπεύει το βήμα της μεθόδου. Η μέθοδος αυτή καλείται αυτόματα από την Godot σε κάθε βήμα υπολογισμού. Είναι σημαντική η διαφορά μεταξύ physics και process frames. Τα process frames, αναφέρονται στα βήματα που καλούνται κάθε καρέ. Σε περίπτωση που εκτελούνταν η λογική μέσα σε αυτά, ανάλογα τα χαρακτηριστικά υλισμικού, θα υπήρχε διαφορετικό αποτέλεσμα. Η παράμετρος *delta* που λαμβάνει η μέθοδος, αντιστοιχεί στον χρόνο που μεσολάβησε από το προηγούμενο καρέ. Η *delta* βοηθάει στην κανονικοποίηση των αποτελεσμάτων, έτσι ώστε να υπάρχει το ίδιο αποτέλεσμα ανεξαρτήτως υλισμικού κάθε χρήστη.

3.1.1 Βρόχος υποβημάτων

Η παράμετρος *delta* δεν χρησιμοποιείται ως ένα ενιαίο βήμα, αλλά υποδιαιρείται σε *n* υποβήματα

$$\Delta t_{sub} = \frac{\delta}{n}$$

```
double sub_dt = delta / num_substeps;

for (int substep = 0; substep < num_substeps; substep++) {
    // 1. Πρόβλεψη θέσεων
    // 2. Μηδενισμός λ
    // 3. Επίλυση περιορισμών
    // 4. Ενημέρωση ταχυτήτων
    // 5. Ανίχνευση σύγκρουσης
}
```

Η τεχνική αυτή, βελτιώνει την σύγκλιση του επιλυτή, χωρίς όμως να αυξάνει τα συνολικά βήματα που πρέπει να εκτελέσει. Κάθε υποβήμα, εκτελεί έναν ολόκληρο κύκλο ζωής της μεθόδου XPBD, ώστε οι ταχύτητες να ενημερώνονται μετά από κάθε μικρό βήμα αντί μία φορά στο τέλος. Η προεπιλεγμένη τιμή *num_substeps = 10* σημαίνει ότι σε κάθε frame εκτελούνται 10 πλήρεις κύκλοι XPBD.

3.1.2 Πρόβλεψη θέσεων

Για την πρόβλεψη των θέσεων των σωματιδίων με βάση τη βαρύτητα και οποιεσδήποτε εξωτερικές δυνάμεις έχουν υποστεί, αρχικά αποθηκεύεται η τρέχουσα θέση και έπειτα υπολογίζεται η προβλεπόμενη.

```
for (int i = 0; i < Particles.size(); i++) {
    Particle &p = Particles[i];
    p.prev_position = p.position;
    p.position = p.position + p.velocity * sub_dt
                + Vec2(0.0, GRAVITY * sub_dt * sub_dt);
}
```

Η εξίσωση που αποθηκεύεται στο ίδιο το σωματίδιο, είναι η εξίσωση θέσης που επεξηγήθηκε σε προηγούμενα κεφάλαια

$$x_i \leftarrow x_i + v_i \cdot \Delta t_{sub} + g \cdot \Delta t_{sub}^2$$

Η εύρεση σταθεράς βαρύτητας η οποία αντιστοιχεί σε αληθοφανές αποτέλεσμα έχει να κάνει με την ανάλυση του αλγορίθμου. Δεδομένο ότι τρέχει η προσομοίωση σε 1080p ο όρος GRAVITY αντιστοιχεί σε $g = (0,980) \text{ pixels/s}^2$. Ο όρος $g \cdot \Delta t_{sub}^2$ προκύπτει από την κινηματική εξίσωση $\Delta x = \frac{1}{2} g \cdot t^2$, αλλά ο συντελεστής $\frac{1}{2}$ παραλείπεται σκοπίμως. Έχει ήδη απλοποιηθεί διότι συνδυάσαμε δύο βήματα από την μέθοδο symplectic Euler

$$v \leftarrow v + \Delta t \cdot g$$

$$x \leftarrow x + \Delta t \cdot v_{new}$$

Επομένως,

$$x \leftarrow x + \Delta t \cdot (v_{old} + \Delta t \cdot g)$$

$$x \leftarrow x + v_{old} \cdot \Delta t + g \cdot \Delta t^2$$

Ακριβώς μετά από αυτό το βήμα, ακολουθεί ο μηδενισμός των πολλαπλασιαστών Lagrange κάθε περιορισμού

```
for (int i = 0; i < Distance_Constraints.size(); i++) {
    Distance_Constraints[i].lambda = 0.0;
}
```

3.1.3 Προβολή περιορισμών

Σε αυτό το κομμάτι κώδικα, βρίσκεται το πιο υπολογιστικά απαιτητικό σημείο. Για κάθε βήμα, διατρέχονται όλοι οι περιορισμοί που αποτελούν το σώμα, και εφαρμόζεται η διόρθωση. Ωστόσο, πριν την εκτέλεση του κύριου βρόγχου, γίνεται η επιλογή μεταξύ XPBD και HP-XPBD

```
if (solver_mode == SOLVER_HIERARCHICAL) {
    solve_hierarchical(sub_dt);
} else {
    solve_flat(sub_dt);
}
```

Ανεξαρτήτως τον επιλυτή που έχει επιλεγεί, η λογική προβολή περιορισμών παραμένει η ίδια. Για κάθε περιορισμό j μεταξύ σωματιδίων a και b , ο κώδικας υλοποιεί ακριβώς τις εξισώσεις XPBD

```
Vec2 difference = Particles[c.particle_b].position
                  - Particles[c.particle_a].position;
double distance = std::sqrt(difference.x * difference.x
                             + difference.y * difference.y);

if (distance < 1e-7) continue;

double C = distance - c.rest_length;
Vec2 C_gradient = difference / distance;

double w_a = Particles[c.particle_a].inverse_mass;
double w_b = Particles[c.particle_b].inverse_mass;
double alpha_tilde = (1.0 / c.stiffness) / (sub_dt * sub_dt);
double delta_lambda = -(C + alpha_tilde * c.lambda)
                      / (w_a + w_b + alpha_tilde);

c.lambda += delta_lambda;

Particles[c.particle_a].position -= C_gradient * (delta_lambda * w_a);
Particles[c.particle_b].position += C_gradient * (delta_lambda * w_b);
```

Αρχικά, γίνεται ο υπολογισμός διαφοράς απόστασης μεταξύ των δύο γειτονικών σωματιδίων. Σε ορισμένες περιπτώσεις λόγω αριθμητικών σφαλμάτων ή ακραίες παραμορφώσεις, υπάρχει πιθανότητα η απόσταση μεταξύ των σωματιδίων να πλησιάσει το 0. Επειδή ο υπολογισμός του περιορισμού περιλαμβάνει διαίρεση με την απόσταση, μια τόσο μικρή τιμή μπορεί να οδηγήσει σε αριθμητική αστάθεια ή πρακτικά σε διαίρεση με μηδέν. Για τον λόγο αυτό, όταν , ο περιορισμός παραλείπεται προσωρινά. Πρόκειται για μια σπάνια αλλά πιθανή κατάσταση που μπορεί να εμφανιστεί σε ακραίες παραμορφώσεις του συστήματος.

Η μεταβλητή C είναι υπεύθυνη για την αποθήκευση παραβίασης κάθε περιορισμού. Η διαφορά μεταξύ τρέχουσας απόστασης και απόστασης ηρεμίας έχει διατυπωθεί μαθηματικά με τον παρακάτω τύπο

$$C_j = \|x_b - x_a\| - d_0$$

Η αρνητική τιμή αντιστοιχεί σε απομάκρυνση των σωματιδίων, ενώ η θετική υποδηλώνει έλξη. Το μοναδιαίο διάνυσμα κατεύθυνσης $\hat{n}$ αποτυπώνεται με την μεταβλητή *C_gradient* και δείχνει προς ποια κατεύθυνση πρέπει να κινηθούν τα σωματίδια.

Η ενδοτικότητα, δηλαδή το πόσο εύκολα μετακινείται ένας περιορισμός αντιστοιχεί στην μεταβλητή *alpha_tilde*. Είναι σημαντικό να σημειωθεί πως είναι αναγκαία η κανονικοποίηση, για αυτό εξάλλου γίνεται και η διαίρεση. Η διόρθωση του πολλαπλασιαστή Lagrange *delta_lamda*, εξαρτάται τόσο από την τρέχουσα παραβίαση *C* όσο και από τη σωρευτική διόρθωση $\tilde{\alpha}$ που έχει ήδη εφαρμοστεί. Τέλος, ο πολλαπλασιαστής ενημερώνεται συσσωρευτικά έτσι ώστε η έννοια της ‘μνήμης’ να παραμένει άθικτη.

3.2 Flat XPBD επιλυτής

Στον flat επιλυτή της μεθόδου XPBD, όλοι οι περιορισμοί λύνονται σειριακά, δηλαδή με την σειρά που βρίσκονται αποθηκευμένοι στο διάνυσμα *Distance_Constraints*

```
void SoftBody2D::solve_flat(double sub_dt) {
    for (int i = 0; i < Distance_Constraints.size(); i++) {
        Distance_Constraint &c = Distance_Constraints[i];

        // ... XPBD διόρθωση
    }
}
```

Ο όρος ‘Flat’ υποδηλώνει πως είναι η παραδοσιακή μέθοδος επίλυσης. Κάθε είδος περιορισμού αντιμετωπίζεται ισάξια. Ανεξαρτήτως από το αν είναι δομικός ή διατμητικός, έχει την ίδια ακριβώς πιθανότητα να επιλυθεί από την μέθοδο. Η σειρά επίλυσης τους καθορίζεται πολύ απλά από την σειρά δημιουργίας της συνάρτησης *spawn_grid* η οποία είναι υπεύθυνη για τον σχηματισμό του πλέγματος.

Στην πράξη ωστόσο, η συνάρτηση *spawn_grid* ακολουθεί ένα μοτίβο. Αρχικά δημιουργείται ο οριζόντιος περιορισμός του σωματιδίου, έπειτα ο κάθετος και μετά οι διατμητικοί. Το αποτέλεσμα παραμένει το ίδιο. Οι δομικοί και οι διατμητικοί εναλλάσσονται μέσα στο διάνυσμα χωρίς καμία εγγύηση προτεραιότητας. Παρόλο που αυτός ο αλγόριθμος έχει πολλά θετικά, όπως προβλεψιμότητα και χαμηλότερη υπολογιστική ισχύει διότι δεν χρειάζεται κάποιου είδους ταξινόμηση, υπάρχει μια σημαντική λεπτομέρεια που παραλείπει. Η μέθοδος Gauss-Seidel που χρησιμοποιεί η XPBD δεν είναι ανεξάρτητη από τη σειρά επίλυσης. Οι περιορισμοί που λύνονται πρώτοι τείνουν να ικανοποιούνται καλύτερα σε σχέση με τους μεταγενέστερους οι οποίοι μπορεί να παραβιαστούν εκ νέου λόγω των προηγούμενων διορθώσεων.

Σε ένα πλέγμα σωματιδίων, ένας ένα δομικός περιορισμός διορθώσει την θέση ενός σωματιδίου και έπειτα ένας διατμητικός το μετακινήσει σε διαφορετική θέση, χάνει την αποτελεσματικότητα της

διόρθωσης. Αυτό μπορεί να επιλυθεί με παραπάνω βήματα του επιλυτή, ωστόσο σε κρίσιμες εφαρμογές πραγματικού χρόνου που οι πόροι είναι περιορισμένοι, είναι αναγκαία η αποτελεσματικότητα.

3.3 Ιεραρχικός επιλυτής

Η ιεραρχική επίλυση περιορισμών ομαδοποιεί τους περιορισμούς σε επίπεδα προτεραιότητας και τους επιλύει διαδοχικά. Μεγαλύτερη προτεραιότητα έχουν οι δομικοί (προτεραιότητα 0) και έπειτα ακολουθούν οι διατμητικοί (προτεραιότητα 1). Όταν χρειαστεί να εφαρμοστεί ένας διατμητικός περιορισμός, οι θέσεις έχουν ήδη διορθωθεί από όλους τους υπόλοιπους δομικούς περιορισμούς από το τρέχων βήμα.

$$\underbrace{j \in \text{group}[0]}_{\text{δομικοί πρώτοι}}, \underbrace{j \in \text{group}[1]}_{\text{διατμητικοί δεύτεροι}}, \dots$$

Κατά την αρχικοποίηση καλείται η `build_priority_groups()`, η οποία δημιουργεί τον πίνακα δεικτών ανά επίπεδο

```
void SoftBody2D::build_priority_groups() {
    priority_groups.clear();
    int max_pri = 0;
    for (auto &c : Distance_Constraints) {
        if (c.priority > max_pri) max_pri = c.priority;
    }
    priority_groups.resize(max_pri + 1);
    for (int i = 0; i < (int)Distance_Constraints.size(); i++) {
        priority_groups[Distance_Constraints[i].priority].push_back(i);
    }
}
```

Η δομή `priority_groups` περιέχει τους δείκτες j με $p_j = p$. Η κατασκευή γίνεται σε $O(N_C)$ και εκτελείται μόνο μία φορά. Η κύρια διαφορά σε σχέση με τον flat solver είναι ο εξωτερικός βρόχος *for level*. Αντί να διατρέχει τους περιορισμούς κατά σειρά εισαγωγής, τους ομαδοποιεί ανά προτεραιότητα. Η μαθηματική πράξη ανά περιορισμό παραμένει η ίδια εφαρμογή XPBD

```
void SoftBody2D::solve_hierarchical(double sub_dt) {
    for (int level = 0; level < (int)priority_groups.size(); level++) {
        for (int idx : priority_groups[level]) {
            // ... XPBD διόρθωση
        }
    }
}
```

Η σειρά επίσκεψης επηρεάζει τον ρυθμό σύγκλισης της Gauss-Seidel αλλά όχι το σημείο ισορροπίας. Με άπειρα βήματα, και οι δύο επιλυτές συγκλίνουν στην ίδια λύση. Η ιεραρχική σειρά μειώνει τις αντικρουόμενες διορθώσεις σε σενάρια μεγάλων παραμορφώσεων, όπου ένας διατμητικός περιορισμός θα μπορούσε να «αναιρέσει» μερικώς μια δομική διόρθωση.

3.4 Δημιουργία πλέγματος σωματιδίων

Η μέθοδος *spawn_grid()* κατασκευάζει ολόκληρη τη δομή του μαλακού σώματος με βάση πέντε παραμέτρους που εκτίθενται στον Inspector. Αυτή η παραμετροποίηση επιτρέπει πλήρη έλεγχο από την κατασκευή του. Η εκτεθειμένες μεταβλητές είναι οι διαστάσεις του πλέγματος, η απόσταση μεταξύ των σωματιδίων, η αντίστροφη μάζα κάθε σωματιδίου και η δυσκαμψία κάθε σωματιδίου.

```
Vector2 origin = get_global_position();
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        Particles.push_back(Particle(
            Vec2(origin.x + x * spacing, origin.y + y * spacing),
            1.0 / mass
        ));
    }
}
```

Κάθε σωματίδιο αποθηκεύεται με αντίστροφη μάζα, ώστε ο solver να εκτελεί πολλαπλασιασμό αντί διαίρεσης σε κάθε iteration. Η διευθυνσιοδότηση είναι γραμμική: το σωματίδιο στη στήλη x και γραμμή y έχει δείκτη $idx = y \cdot W + x$. Ο δεύτερος βρόχος κατασκευάζει τους περιορισμούς απόστασης. Για κάθε κόμβο ελέγχεται αν υπάρχει γείτονας στην εκάστοτε κατεύθυνση, αποφεύγοντας πρόσβαση εκτός ορίων

```
int idx = y * width + x;

if (x < width - 1)
    Distance_Constraints.push_back(
        Distance_Constraint(idx, idx + 1, spacing, stiffness, 0));

if (y < height - 1)
    Distance_Constraints.push_back(
        Distance_Constraint(idx, idx + width, spacing, stiffness, 0));

double diag = std::sqrt(2.0) * spacing;
if (x < width - 1 && y < height - 1)
    Distance_Constraints.push_back(
        Distance_Constraint(idx, idx + width + 1, diag, stiffness, 1));
if (x > 0 && y < height - 1)
    Distance_Constraints.push_back(
        Distance_Constraint(idx, idx + width - 1, diag, stiffness, 1));
```

3.5 Ανίχνευση σύγκρουσης με δάπεδο

Η μόνος μη δυναμικός περιορισμός είναι η σύγκρουση με το δάπεδο. Αφού έχουν ενημερωθεί οι ταχύτητες ενός σωματιδίου, επιβάλλεται ένας απλός περιορισμός χώρου που εμποδίζει τα σωματίδια να διαπεράσουν το δάπεδο. Αντί να υπολογίζονται δυνάμεις αντίδρασης, η θέση προβάλλεται απευθείας

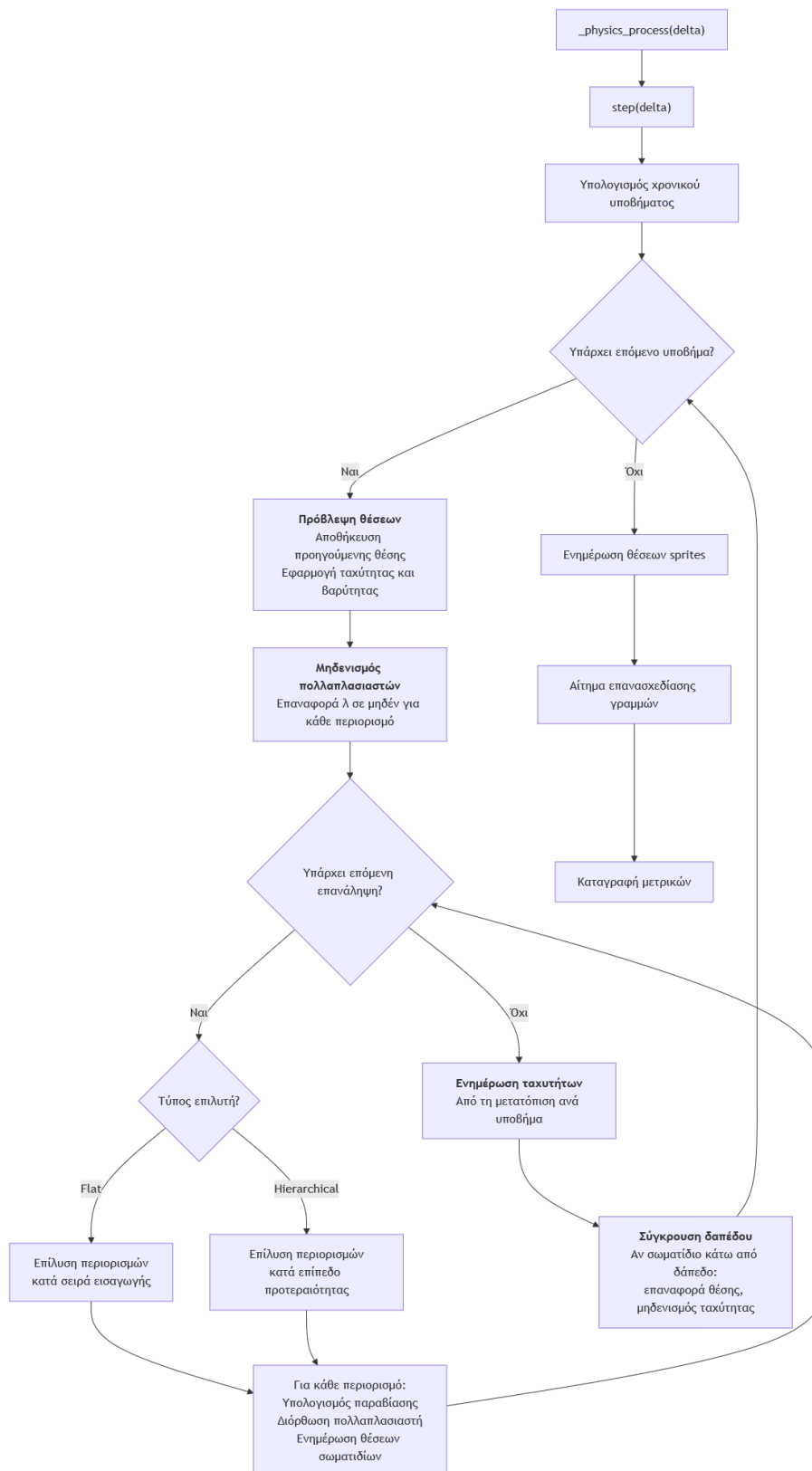
στο όριο του επιτρεπτού χώρου. Αν ένα σωματίδιο βρεθεί κάτω από το δάπεδο ($y_i > y_{\text{floor}}$ στο σύστημα συντεταγμένων της Godot όπου ο άξονας y αυξάνεται προς τα κάτω), εφαρμόζονται δύο διορθώσεις. Αρχικά η θέση στον ένα άξονα γίνεται $y_i = y_{\text{floor}}$ και έπειτα, μηδενίζεται η κατακόρυφη συνιστώσα ταχύτητας.

```
for (int i = 0; i < Particles.size(); i++) {
    Particle &p = Particles[i];
    if (p.position.y > floor_y) {
        p.position.y = floor_y;
        p.velocity.y = 0.0;
    }
}
```

Αυτή η τοποθέτηση εξασφαλίζει ότι η διόρθωση σύγκρουσης αντικατοπτρίζεται στην ταχύτητα του σωματιδίου πριν το επόμενο υποβήμα. Αφού η θέση τροποποιείται μετά τον υπολογισμό, ο μηδενισμός του v_y αποτρέπει τη συσσώρευση κατακόρυφης ορμής σε διαδοχικά υποβήματα.

3.6 Συνολική ροή

Ολόκληρη η ροή του προγράμματος μπορεί να συνοψιστεί στο ακόλουθο διάγραμμα ροής. Κάθε physics frame, η Godot καλεί τη `_physics_process()`, η οποία εκκινεί τον κύκλο προσομοίωσης. Το χρονικό βήμα χωρίζεται σε υποβήματα, και για κάθε υποβήμα εκτελείται η πλήρης ακολουθία XPBD: πρόβλεψη θέσεων, μηδενισμός πολλαπλασιαστών, επαναληπτική επίλυση περιορισμών, ενημέρωση ταχυτήτων και ανίχνευση σύγκρουσης. Μετά την ολοκλήρωση όλων των υποβημάτων, οι τελικές θέσεις μεταφέρονται στα οπτικά στοιχεία της σκηνής.



Σχήμα 3.6: Διάγραμμα ροής λογικής plugin

Κεφάλαιο 4ο: Μετρήσεις απόδοσης και αξιολόγηση

4.1 Εισαγωγή

Έχει πλέον παρουσιαστεί η μαθηματική θεμελίωση των μεθόδων XPBD και HP-XPBD. Επίσης, έχουν γίνει γνωστοί όλοι οι μηχανισμοί του plugin ως GDExtension. Στο παρόν κεφάλαιο θα αξιοποιηθεί πρακτικά το plugin για την πειραματική μελέτη μεταξύ των δύο αλγορίθμων.

Μέσω των μετρήσεων και μετρικών θα απαντηθούν μια πληθώρα ερωτημάτων. Ερωτήματα όπως είναι η προσομοίωση εφικτή σε πραγματικό χρόνο για πλέγματα πρακτικού μεγέθους; Πώς κλιμακώνεται ο χρόνος εκτέλεσης με τον αριθμό σωματιδίων και περιορισμών; Υπάρχει μετρήσιμη διαφορά μεταξύ του flat και του hierarchical επιλυτή ως προς τη σύγκλιση και την υπολογιστική επιβάρυνση; Για τον σκοπό αυτό, χρησιμοποιείται το ενσωματωμένο σύστημα μετρικών που καταγράφει χρόνο βήματος, μέγιστο χρόνο, και σφάλμα περιορισμών σε μορφή CSV, επιτρέποντας αυτοματοποιημένη σύγκριση και οπτικοποίηση.

4.2 Σύστημα μετρήσεων

Παράλληλα με την εκτέλεση των αλγορίθμων, ενσωματώνεται το σύστημα μετρικών. Η απόδοση της προσομοίωσης επηρεάζεται σχεδόν μηδαμινά από τις μετρήσεις. Η καταγραφή ενεργοποιείται αυτόματα κατά την εκκίνηση της σκηνης αν *auto_start_capture = true* ή χειροκίνητα μέσω κλήσης της μεθόδου *start_metrics_capture(duration_s, sample_ms)*.

4.2.1 Δειγματοληψία και καταγραφή

Σε κάθε physics frame, η *process_metrics_capture()* προσθέτει τον χρόνο delta σε έναν συσσωρευτή. Όταν ο συσσωρευτής ξεπεράσει το διάστημα δειγματοληψίας αποθηκεύεται ένα δείγμα με τις τρέχουσες τιμές

```
void SoftBody2D::process_metrics_capture(double delta_ms) {
    if (!capture_active) return;

    capture_elapsed_ms += delta_ms;
    sample_accumulator_ms += delta_ms;

    if (sample_accumulator_ms >= sampling_interval_ms) {
        MetricSample s;
        s.time_s = capture_elapsed_ms / 1000.0;
        s.step_ms = metric_step_time_ms;
        s.avg_ms = metric_avg_step_time_ms;
        s.max_ms = metric_max_step_time_ms;
        s.error = metric_constraint_error;
        s.particles = metric_particle_count;
        s.constraints = metric_constraint_count;
        metric_samples.push_back(s);
        sample_accumulator_ms -= sampling_interval_ms;
    }
}
```

```

    }

    if (capture_elapsed_ms >= capture_duration_s * 1000.0) {
        capture_active = false;
        write_metrics_csv();
    }
}

```

Σε κάθε δείγμα βρίσκονται πέντε μετρήσεις.

step_ms	Χρόνος εκτέλεσης τελευταίου step()	ms
avg_ms	Τρέχων μέσος όρος	ms
max_ms	Μέγιστος χρόνος από την αρχή	ms
error	Αθροιστικό σφάλμα	C _j
particles, constraints	Μέγεθος συστήματος	—

Πίνακας 4.2.1: Τύποι μετρήσεων

Ο χρόνος βήματος *step_ms*, μετράει τον πραγματικό χρόνο που πέρασε καθώς έπαιρναν μέρος οι υπολογισμοί για ένα physics frame. Περιλαμβάνει όλη τη διαδικασία όπως υποβήματα, επαναλήψεις, συγκρούσεις ωστόσο δεν περιλαμβάνει τον σχεδιασμό γραφικών στην οθόνη. Εφόσον η προσομοίωση τρέχει σε ρυθμό ανανέωσης 60Hz, αν η τιμή υπερβαίνει τα 16.6ms, αυτό σημαίνει πως η προσομοίωση δεν πρόλαβε να ολοκληρωθεί μέσα σε ένα frame.

Ο μέσος χρόνος *avg_ms* είναι ο αθροιστικός μέσος όρος χρόνο που πέρασε από την αρχή της προσομοίωσης. Αφότου σταθεροποιηθεί μετά από μερικά δευτερόλεπτα, μπορεί να δώσει αξιόπιστη εκτίμηση της τυπικής επιβάρυνσης.

Ο μέγιστος χρόνος *max_ms* μπορεί να διατυπώσει την χειρότερη δυνατή περίπτωση κατά την διάρκεια της προσομοίωσης. Αυτή η μέτρηση έχει ιδιαίτερο βάρος, διότι ακόμα και αν όλες οι υπόλοιπες είναι χαμηλές, μία αύξηση μεγαλύτερη από 16.6ms υποδηλώνει χάσιμο καρέ. Σημαντική πληροφορία για εφαρμογές real-time.

Τέλος, το σφάλμα περιορισμών *error* είναι το αθροιστικό άθροισμα των απόλυτων παραβάσεων όλων των περιορισμών

$$\text{error} = \sum_{j=0}^{N_C-1} ||x_b - x_a| - d_j|$$

Η μετρική αυτή υποδηλώνει το μέγεθος της σύγκλισης κάθε μεθόδου. Αν ο επιλυτής είχε συγκλίνει τέλεια, κάθε $|C_j|$ θα ήταν μηδέν. Στην πράξη, με πεπερασμένα βήματα, παραμένει ένα υπόλειμμα.

Μικρότερο *error* σημαίνει καλύτερη ικανοποίηση περιορισμών δηλαδή το μαλακό σώμα διατηρεί πιστότερα το σχήμα του.

4.2.2 Εξαγωγή σε CSV

Εφόσον τελειώσει ολόκληρη η διαδικασία καταγραφής η `write_metrics_csv()` αποθηκεύει τα δείγματα σε αρχείο CSV. Κάθε εκτέλεση δημιουργεί αυτόματα νέο φάκελο με χρονοσφραγίδα

```
metrics/metrics_20260731_143108/metrics.csv
metrics/metrics_20260731_143134/metrics.csv
```

Η δομή αυτή εξασφαλίζει ότι καμία εκτέλεση δεν αντικαθιστά προηγούμενη και πως όλα τα δεδομένα διατηρούνται για αργότερη σύγκριση. Κάθε CSV αρχείο, ξεκινάει με γραμμές σχολίων οι οποίες περιέχουν τις παραμέτρους εκτέλεσης για το τρέχον αρχείο

```
# solver_mode=flat_xpbd
# particles=100
# constraints=261
# substeps=10
# grid=10x10
# stiffness=1000
time_s, step_time_ms, avg_ms, max_ms, error, particles, constraints
0.100000, 0.245000, 0.312000, 0.512000, 3.450000, 100, 261
0.200000, 0.231000, 0.298000, 0.512000, 2.180000, 100, 261
...
```

Τα δεδομένα αυτά, επιτρέπουν στα scripts ανάλυσης να αναγνωρίσουν αυτόματα τις συνθήκες κάθε εκτέλεσης χωρίς χειροκίνητη σημείωση. Μετά την εγγραφή του CSV, η μέθοδος καλεί αυτόματα το script `compare_solvers.py`

```
std::ostringstream cmd;
cmd << "py ../tools/compare_solvers.py " << "metrics";
int rc = std::system(cmd.str().c_str());
```

Το script εντοπίζει το πιο πρόσφατο CSV για κάθε τύπο επιλυτή, και αν υπάρχουν και τα δύο, παράγει συγκριτικά γραφήματα στον φάκελο: `metrics/comparison_latest/`. Όλες οι αυτοματοποιήσεις ελαχιστοποιούν πολύ τα χειροκίνητα βήματα που πρέπει να εκτελέσει ο χρήστης. Το μόνο που χρειάζεται από τον ίδιο, είναι να τρέξει την σκηνή δύο φορές. Μία με τον ιεραρχικό επιλυτή και μία με τον flat. Εφόσον έχουν τρέξει επιτυχώς και έχουν καταγραφεί τα δεδομένα, δημιουργούνται αυτόματα διαγράμματα σύγκρισης των δύο.

4.3 Πειραματική διάταξη

Για τη σωστή και ακριβή σύγκριση μεταξύ των δύο μεθόδων, σχεδιάστηκε μία σειρά πειραμάτων που ακολουθούν την τεχνική της ελεγχόμενης μεταβολής [17]. Με κάθε διαφορετικό πείραμα, μεταβάλλεται μία παράμετρος την φορά, ενώ οι υπόλοιπες παραμένουν σταθερές. Με αυτή την μέθοδο, είναι εύκολος ο εντοπισμός αίτιων, παρά απλώς συσχετίσεων. Τα πειράματα που θα ακολουθήσουν, οργώνονται σε τρεις ξεχωριστές ομάδες:

- **Ομάδα Α: Σύγκριση επιλυτών.** Στη παρούσα ομάδα, η ανεξάρτητη μεταβλητή είναι ο τύπος επιλυτή. Θα εκτελεσθούν πέντε ζεύγη πειραμάτων υπό διαφορετικές συνθήκες. Ανάλογα το πείραμα θα εξεταστεί αν η ιεραρχική σειρά προσφέρει μετρήσιμο πλεονέκτημα σε σφάλμα ή χρόνο, και αν αυτό εξαρτάται από τη δυσκαμψία, το μέγεθος πλέγματος, ή τον αριθμό επαναλήψεων. Η μεθοδολογία σύγκρισης ακολουθεί τις πρακτικές αξιολόγησης που χρησιμοποιούν οι Macklin et al. [4] και Bender et al. [5] στις αντίστοιχες εργασίες τους.
- **Ομάδα Β: Κλιμάκωση μεγέθους.** Η ανεξάρτητη μεταβλητή είναι τα μεγέθη των σωματιδίων και των περιορισμών. Τα πειράματα θα εκτελέσουν τις μεθόδους ξεκινώντας με πλέγματα 5x5 και θα φτάσουν μέχρι πλέγματα 100x100. Η συγκεκριμένη ομάδα, αποκαλύπτει τη σχέση μεταξύ μεγέθους συστήματος και χρόνου εκτέλεσης, εντοπίζοντας το πρακτικό όριο για real-time λειτουργία [18] στα 60 Hz.
- **Ομάδα Γ: Υποβήματα έναντι επαναλήψεων.** Ανεξάρτητη μεταβλητή στη παρούσα ομάδα είναι η κατανομή του υπολογιστικού κόστους μεταξύ χρονικής ανάλυσης (υποβήματα) και ακρίβειας σύγκλισης (επαναλήψεις). Εκτελέσεις με ίδιο συνολικό αριθμό περσμάτων αλλά διαφορετική αναλογία δείχνουν ποια στρατηγική ελαχιστοποιεί το σφάλμα για δεδομένο υπολογιστικό προϋπολογισμό.

Η διάρκεια κάθε καταγραφής είναι 10 δευτερόλεπτα με δειγματοληψία ανά 100 ms, παράγοντας ~100 δείγματα ανά εκτέλεση τα οποία είναι αρκετά για στατιστική αξιοπιστία. Τα πρώτα δευτερόλεπτα κάθε καταγραφής περιλαμβάνουν τη φάση πτώσης λόγω βαρύτητας και τη σύγκρουση με το δάπεδο η οποία είναι η πιο απαιτητική φάση. Τα τελευταία αντιπροσωπεύουν τη σταθερή κατάσταση μετά την ηρεμία του σώματος.

4.3.1 Πειράματα δοκιμών

Στην ομάδα Α, κάθε πείραμα συμβάλει στην καθαρή σύγκριση μεταξύ των δύο αλγορίθμων.

Παράμετρος	Τιμή
Πλέγμα	10×10 (261 περιορισμοί)
Υποβήματα	10

Επαναλήψεις	1
Δυσκαμψία	1000

Πίνακας 4.3.1 (1): Πείραμα A1

Αυτό το πείραμα είναι η βάση αναφοράς. Οι παράμετροι αντιπροσωπεύουν πραγματικές συνθήκες για προσομοίωση πραγματικού χρόνου. Αρκετά υποβήματα για σταθερότητα, μία επανάληψη ανά υποβήμα (όπως συνιστούν οι Macklin et al. [11]), και μέτρια δυσκαμψία. Τα αποτελέσματα χρησιμεύουν ως σημείο σύγκρισης για τα υπόλοιπα πειράματα.

Παράμετρος	Τιμή
Πλέγμα	10×10 (261 περιορισμοί)
Υποβήματα	3
Επαναλήψεις	1
Δυσκαμψία	1000

Πίνακας 4.3.1 (2): Πείραμα A2

Οι ευκαιρίες σύγκλισης για τον επιλυτή είναι πολύ λίγες λόγω του ποσού των υποβημάτων. Το χρονικό υποβήμα γίνεται μεγαλύτερο (αντί), οδηγώντας σε μεγαλύτερες αρχικές παραμορφώσεις και εντονότερες διορθώσεις. Σε αυτές τις συνθήκες, η σειρά επίσκεψης αποκτά μεγαλύτερη σημασία: αν ένας διατμητικός περιορισμός αναιρέσει μερικώς μια δομική διόρθωση, δεν υπάρχουν αρκετά υποβήματα για αποκατάσταση.

Παράμετρος	Τιμή
Πλέγμα	20×20 (1121 περιορισμοί)
Υποβήματα	10
Επαναλήψεις	1
Δυσκαμψία	1000

Πίνακας 4.3.1 (3): Πείραμα A3

Το πλέγμα έχει γίνει σαφώς μεγαλύτερο. Κάθε διόρθωση θα πρέπει να περάσει από κάθε σημείο για να φτάσει από την μία άκρη στην άλλη. Οι πιθανότητες αντικρουόμενων διορθώσεων αυξάνονται πολύ περισσότερο όσο περισσότεροι είναι και οι περιορισμοί.

Παράμετρος	Τιμή
Πλέγμα	10×10 (261 περιορισμοί)

Υποβήματα	10
Επαναλήψεις	1
Δυσκαμψία	5000

Πίνακας 4.3.1 (4): Πείραμα A4

Σε αυτή την περίπτωση το πλέγμα έχει γίνει σχεδόν άκαμπτο. Κανένας περιορισμός δεν θέλει να ενδώσει και επομένως κάθε διόρθωση μετακινεί τα σωματίδια σε πλήρες μήκος ηρεμίας. Ο δείκτης ενδοτικότητας γίνεται , η οποία είναι μια πολύ μικρή τιμή.

Παράμετρος	Τιμή
Πλέγμα	10×10 (261 περιορισμοί)
Υποβήματα	10
Επαναλήψεις	5
Δυσκαμψία	1000

Πίνακας 4.3.1 (5): Πείραμα A5

Τα περάσματα του επιλυτή στο σύνολο όλων των περιορισμών είναι αυξημένα με πέντε επαναλήψεις. Θα εξεταστεί αν αρκούν οι πέντε επαναλήψεις για να εξαλειφθεί η διαφορά μεταξύ XPBD και HP-XPBD.

Τέλος, θα ακολουθήσουν ένα ζευγάρι από πειράματα που έχουν να κάνουν με την φυσική υπόσταση του πλέγματος, δηλαδή της μάζας των σωματιδίων του, και της απόστασης μεταξύ τους.

Παράμετρος	Τιμή
Πλέγμα	10×10 (261 περιορισμοί)
Υποβήματα	10
Επαναλήψεις	5
Δυσκαμψία	1000
Μάζα	0.1
Απόσταση	50

Πίνακας 4.3.1 (6): Πείραμα A6

Παράμετρος	Τιμή
Πλέγμα	10×10 (261 περιορισμοί)

Υποβήματα	10
Επαναλήψεις	5
Δυσκαμψία	1000
Μάζα	1
Απόσταση	150

Πίνακας 4.3.1 (7): Πείραμα A7

Αυτό το ζεύγος πειραμάτων εξετάζει την επιρροή της υλικής υπόστασης στο ποσό διορθώσεων μεταξύ των δύο μεθόδων. Τα πιο ελαφριά σωματίδια και η μεγαλύτερη απόσταση μεταξύ τους δημιουργούν μεγαλύτερες παραμορφώσεις στο πλέγμα, και ως αποτέλεσμα απαιτούν μεγαλύτερο αριθμό διορθώσεων.

Έπειτα ακολουθεί η ομάδα Β. Ο σκοπός αυτών των πειραμάτων είναι η μελέτη κλιμάκωσης και η μέτρηση της σχέσης μεταξύ μεγέθους συστήματος και χρόνου εκτέλεσης. Η αναμενόμενη κλιμάκωση είναι, δεδομένου ότι ο solver εκτελεί σταθερό αριθμό πράξεων ανά περιορισμό. Αποκλίσεις από τη γραμμικότητα μπορεί να οφείλονται σε cache effects ή memory allocation overhead σε μεγαλύτερα πλέγματα.

Πείραμα	Πλέγμα	Σωματίδια	Δυσκαμψία	Περιορισμοί
B1	5×5	25	1000	56
B2	10×10	100	1000	261
B3	15×15	225	1000	616
B4	20×20	400	1000	1121
B5	30×30	900	3000	2581
B6	50×50	2500	3000	7301
B7	100×100	10000	3000	29301

Πίνακας 4.3.1 (8): Πειράματα ομάδας Β

Τα τρία τελευταία πειράματα εξετάζουν ακραίες περιπτώσεις. Σκοπός τους είναι να βρουν το όριο πλέγματος για real-time εφαρμογές εντοπίζοντας το σημείο στο οποίο η παραλληλοποίηση ή η GPU επιτάχυνση γίνεται αναγκαία. Μία σημαντική λεπτομέρεια είναι πως ο βαθμός δυσκαμψίας αυξάνεται στην ομάδα μεγάλων σωμάτων. Χωρίς αυτή την αλλαγή, το σώμα θα κατέρρεε από το ίδιο του το βάρος και η προσομοίωση δεν θα παρείχε ακριβή αποτελέσματα.

Τέλος, η ομάδα Γ, διερευνάει την διαφορά μεταξύ υποβημάτων και επαναλήψεων. Θα εξεταστεί αν είναι αποδοτικότερο το σύστημα να επενδύσει σε μεγαλύτερη χρονική ανάλυση ή σε ακρίβεια σύγκλισης.

Πείραμα	Υποβήματα	Επαναλήψεις	Περάσματα
Γ1	5	1	5
Γ2	10	1	10
Γ3	20	1	20
Γ4	5	4	20
Γ5	10	2	20

Πίνακας 4.3.1 (9): Πειράματα ομάδας Γ

4.3.2 Περιβάλλον εκτέλεσης

Μια θεμελιώδη προϋπόθεση για πειραματικές μελέτες είναι η αναπαραγωγιμότητα. Για αυτόν τον λόγο, όλες οι εκτελέσεις πραγματοποιήθηκαν στο ίδιο σύστημα, υπό σταθερές συνθήκες, και καταγράφηκαν αναλυτικά οι παράμετροι του περιβάλλοντος.

Στοιχείο	Περιγραφή
CPU	AMD Ryzen 5 2600
RAM	16 GB, DDR4, 2133 MT/s
GPU	AMD Radeon RX 6600
Λειτουργικό σύστημα	Windows 10
Μηχανή	Godot 4.5
Γλώσσα υλοποίησης	C++ μέσω GDExtension
Μεταγλώττιση	Release (template_release)
Ρυθμός φυσικής	60 Hz
Νήματα	Single-threaded

Πίνακας 4.3.2 : Στοιχεία συστήματος

Η μεταγλώττιση σε τύπο release, εξασφαλίζει πως οι βελτιστοποιήσεις του μεταγλωττιστή είναι (O2). Κατά τη διάρκεια των μετρήσεων δεν εκτελούνταν άλλες εφαρμογές υψηλής υπολογιστικής επιβάρυνσης, ώστε να ελαχιστοποιηθεί ο θόρυβος στις χρονομετρήσεις.

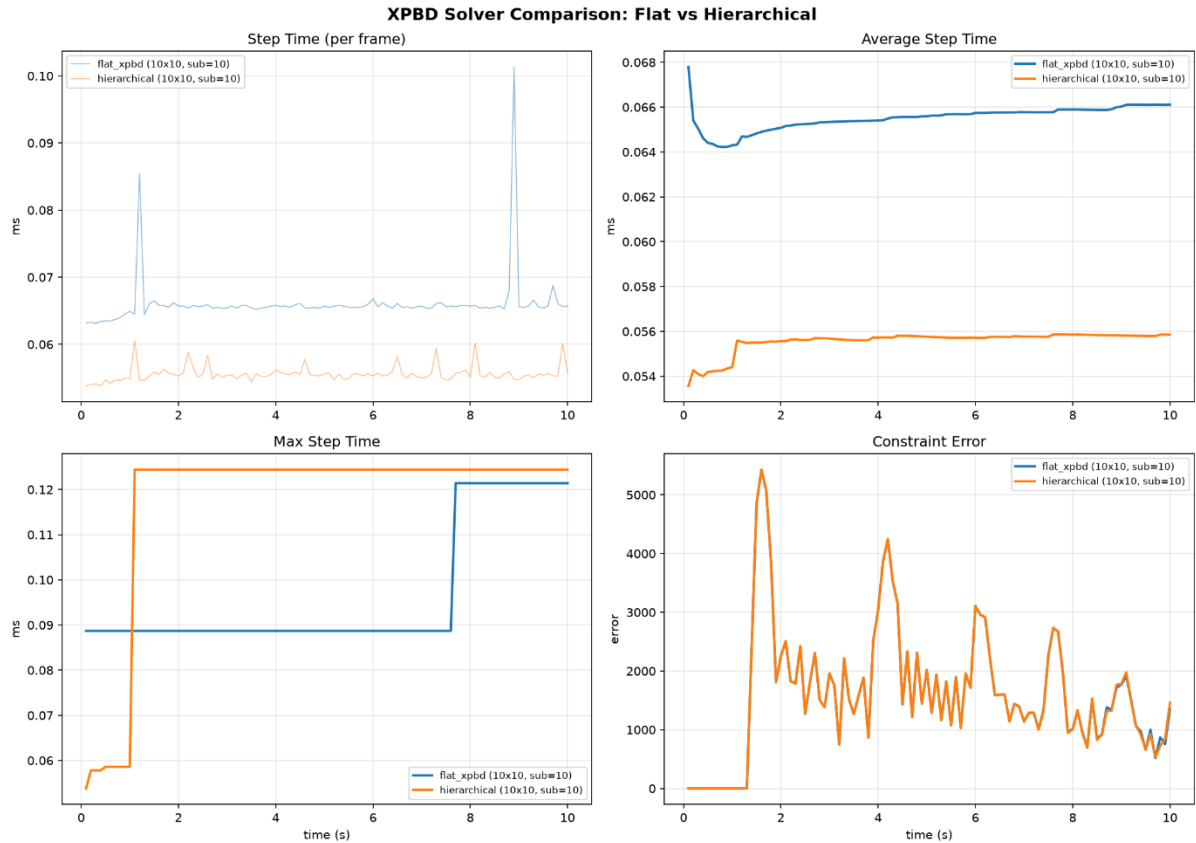
4.4 Αποτελέσματα

Σε αυτή την ενότητα θα παρουσιαστούν τα αποτελέσματα των προαναφερόμενων πειραμάτων. Σε κάθε αποτέλεσμα αντιστοιχεί και ένα γράφημα. Σε κάθε γράφημα απεικονίζεται με μπλε χρώμα η μέθοδος

XPBD και με πορτοκαλί η μέθοδος HP-XPBD. Ο οριζόντιος άξονας αντιστοιχεί στον χρόνο προσομοίωσης σε δευτερόλεπτα ενώ ο κάθετος άξονας στο μέτρο σύγκρισης.

4.4.1 Σύγκριση επιλυτών (Ομάδα Α)

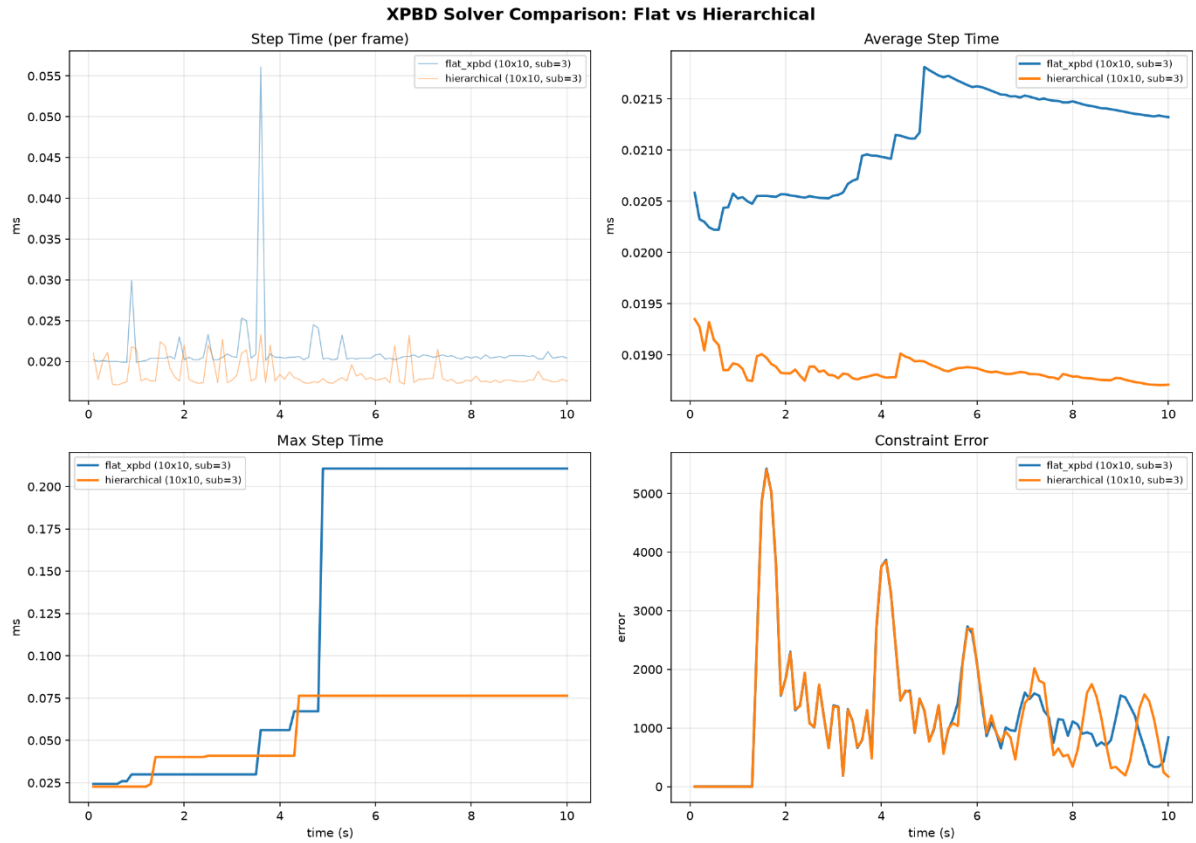
Το πρώτο πείραμα εξυπηρετεί το σημείο αναφοράς. Οι παράμετροι είναι μέσες τιμές που θα μπορούσαν να εφαρμοστούν οπουδήποτε.



Σχήμα 4.4.1 (1): Αποτελέσματα πειράματος Α1

Πείραμα Α1 (10×10, sub=10, iter=1, k=1000):

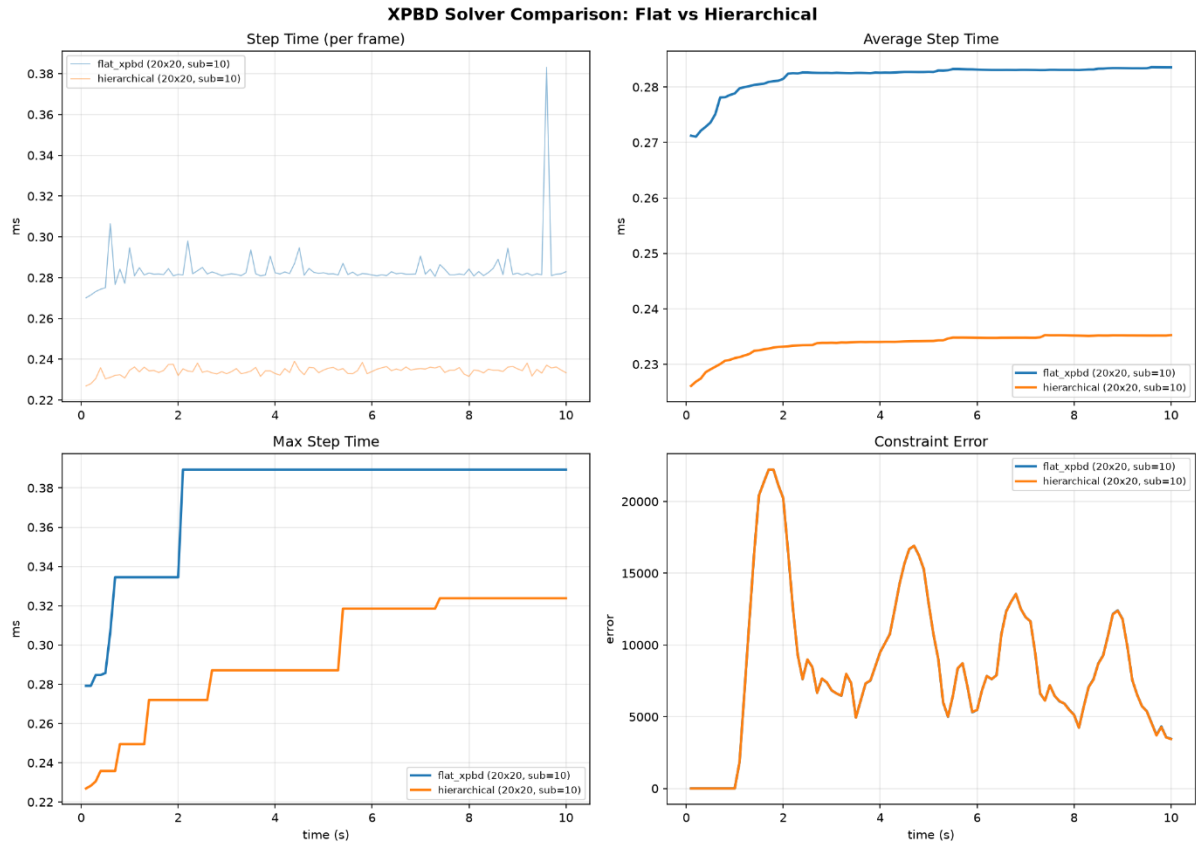
Στην ιεραρχική μέθοδο, ο χρόνος βήματος φαίνεται να είναι σταθερά ταχύτερος. Η flat μέθοδος κυμαίνεται περίπου στα 0.065ms, ενώ η ιεραρχική στα 0.055, μια βελτίωση περίπου 15%. Οι κορυφές του διαγράμματος έχουν να κάνουν με το περιβάλλον στο οποίο προσομοιώνονται οι μέθοδοι. Σε ένα λειτουργικό σύστημα, δεν είναι εφικτό να σταματήσει οποιαδήποτε άλλη λειτουργία και να τρέξει μόνο η προσομοίωση και για αυτό τον λόγο παρουσιάζεται θόρυβος. Το μέσο βήμα παρουσιάζει παρόμοια αποτελέσματα με σταθερή βελτίωση κατά 15% περίπου. Ωστόσο το μέγιστο βήμα εμφανίστηκε νωρίτερα στην ιεραρχική μέθοδο και είναι μεγαλύτερο κατά 0.005ms. Μπορούμε να παρατηρήσουμε παρόμοιο σφάλμα περιορισμών και στις δύο μεθόδους



Σχήμα 4.4.1 (2): Αποτελέσματα πειράματος A2

Πείραμα A2 (10×10 , sub=3, iter=1, k=1000):

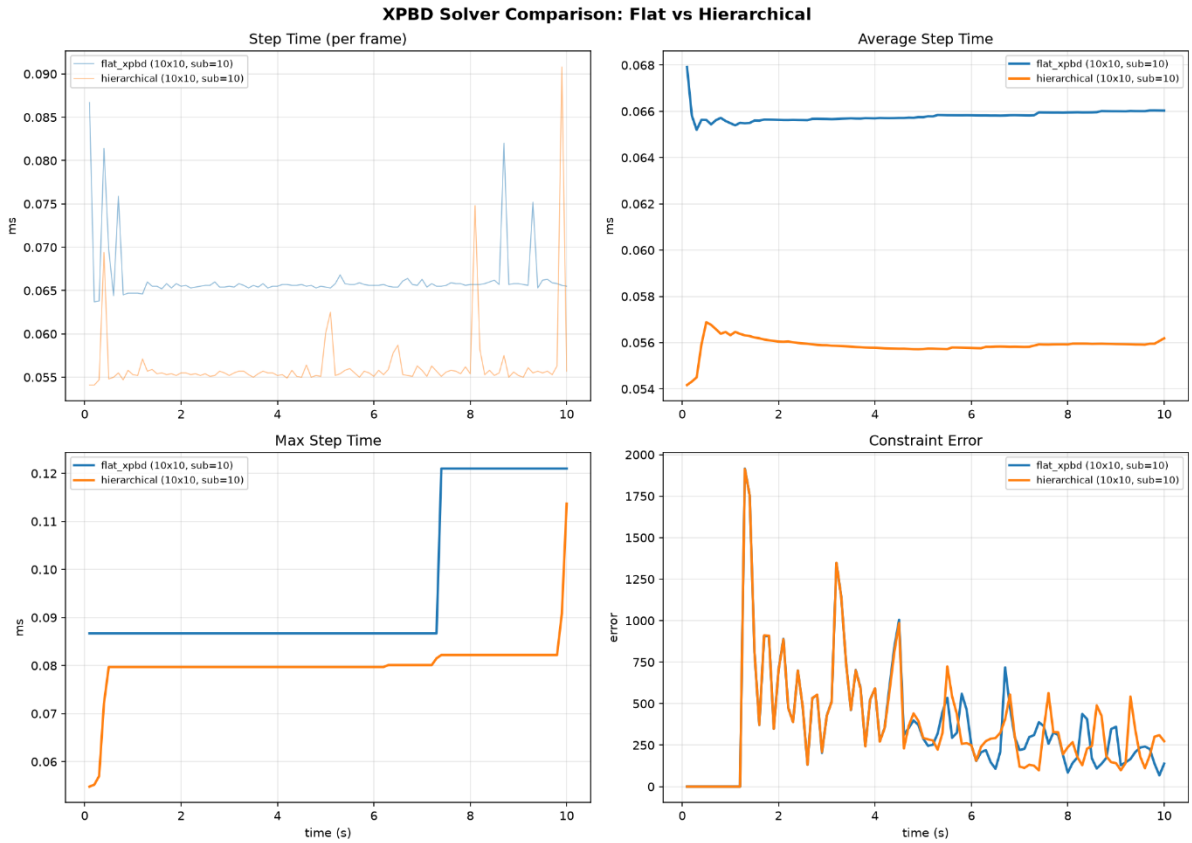
Η πτώση σε αριθμό υποβημάτων επέφερε μεγαλύτερη ταχύτητα σε χρόνους βημάτων. Η ταχύτητα της προσομοίωσης αυξήθηκε περίπου 60%. Όπως μπορούμε να δούμε, οι τιμές πλέον κυμαίνονται γύρω στα 0.02ms, σε σχέση με τα 0.06 του προηγούμενου πειράματος. Ωστόσο, παρόλο που η ιεραρχική μέθοδος εξακολουθεί να είναι πιο γρήγορη, η διαφορά τους μειώθηκε αισθητά. Το μέγιστο βήμα της flat μεθόδου, είναι ακόμα μεγαλύτερο σε σχέση με το πείραμα A1, και η ιεραρχική μέθοδος το μείωσε σε μεγάλο βαθμό. Όπως και πριν, το σφάλμα περιορισμών παραμένει σχεδόν ίδιο και στις δύο μεθόδους.



Σχήμα 4.4.1 (3): Αποτελέσματα πειράματος A3

Πείραμα A3 (20×20, sub=10, iter=1, k=1000):

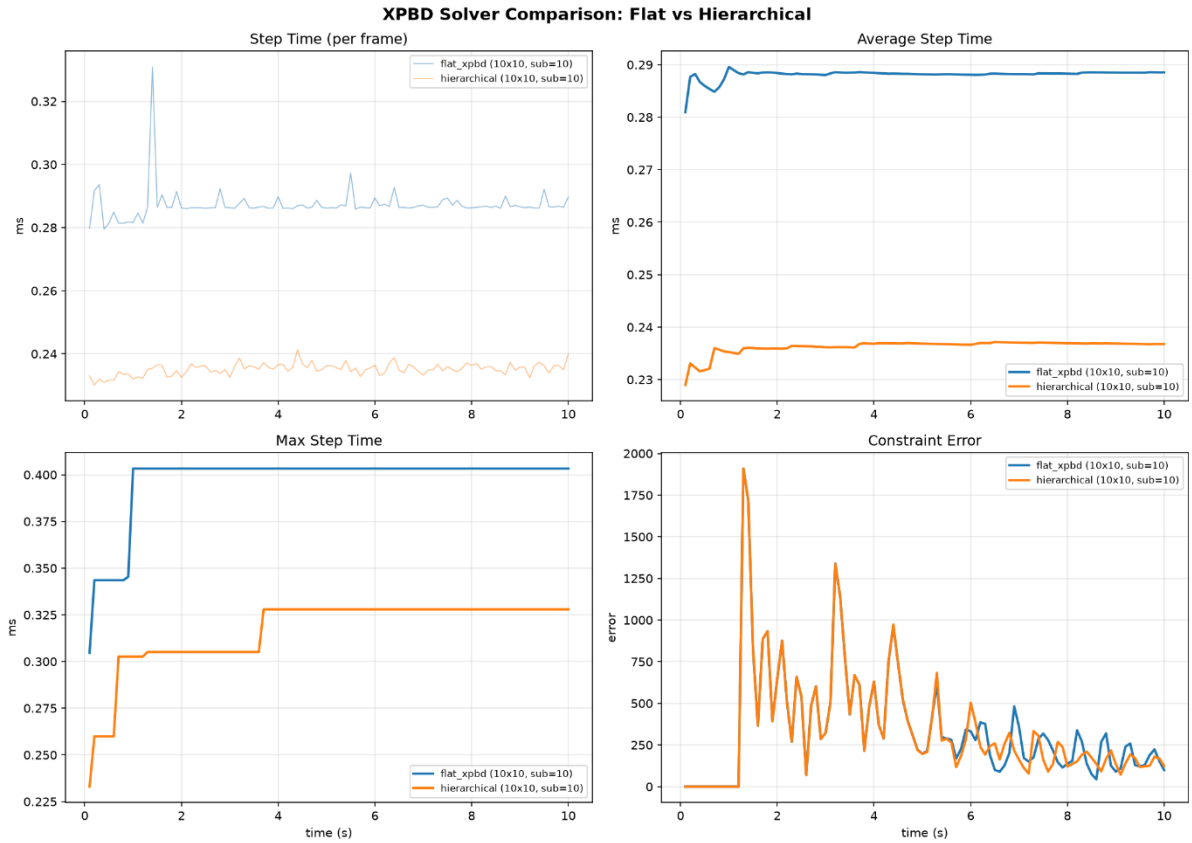
Η διαφορά χρόνου μεγαλώνει σημαντικά σε σχέση με τα μικρότερα πλέγματα. Η flat μέθοδος κινείται στα 0.28ms περίπου ενώ η ιεραρχική στα 0.23ms περίπου. Η διαφορά παραμένει περίπου 18%, η οποία σε απόλυτους αριθμούς (0.05ms) είναι τώρα μεγαλύτερη σε σχέση με το πείραμα A1 (0.01ms). Με τετραπλάσιους περιορισμούς, η σταθερή ποσοστιαία διαφορά μεταφράζεται σε μεγαλύτερο απόλυτο κέρδος.



Σχήμα 4.4.1 (4): Αποτελέσματα πειράματος A4

Πείραμα A4 (10×10, sub=10, iter=1, k=5000):

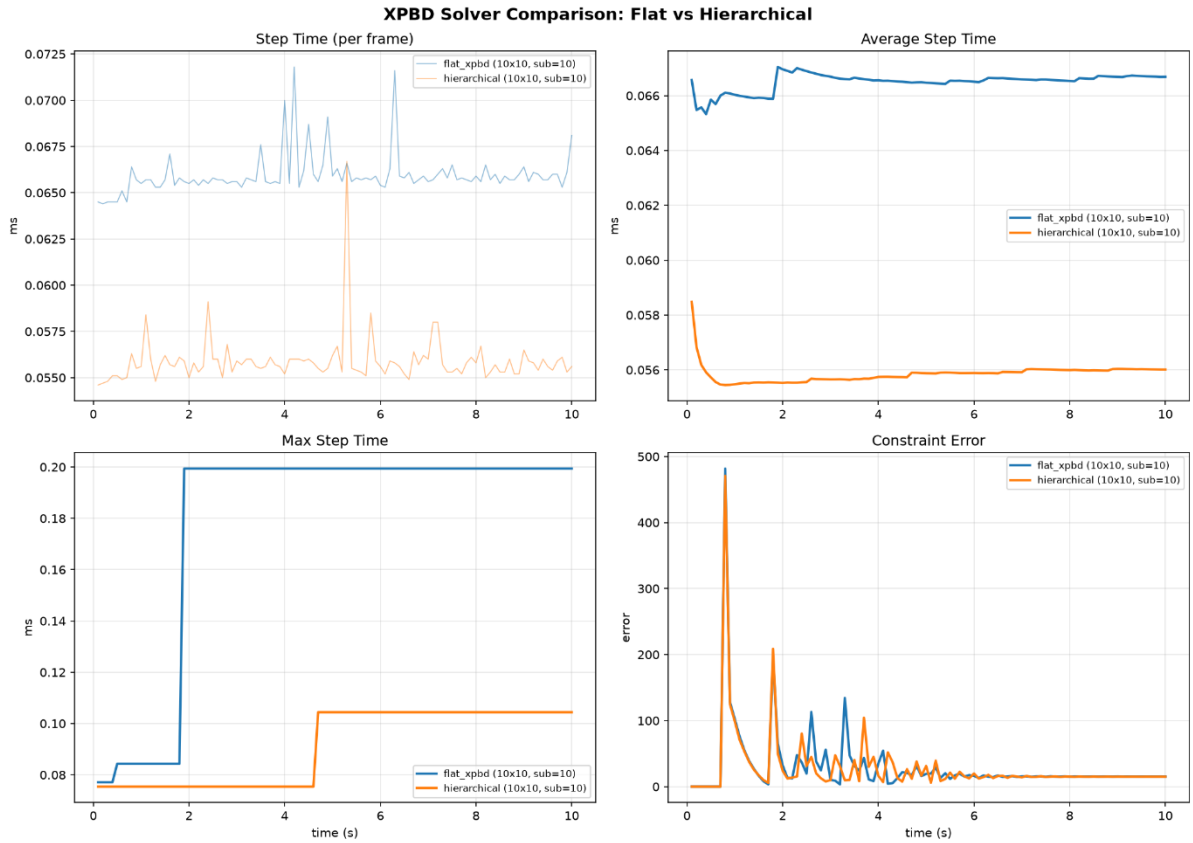
Φαίνεται πως η αύξηση της δυσκαμψίας των σωματιδίων δεν αυξάνουν την ταχύτητα εκτέλεσης. Όλες οι τιμές που έχουν να κάνουν με βήματα είναι πρακτικά αμετάβλητες, και κυμαίνονται στην ίδια περιοχή τιμών σε σχέση με το πείραμα A1. Ωστόσο, πλέον το σφάλμα περιορισμών έχει μειωθεί αισθητά, από 1000 έως 5000 σε 100 με 1900. Εφόσον το υλικό είναι δύσκαμπτο, η προσομοίωση δεν χρειάζεται να παραμορφώσει το πλέγμα όσο θα χρειαζόταν για τα προηγούμενα πειράματα και ως αποτέλεσμα υπάρχουν λιγότερα σφάλματα.



Σχήμα 4.4.1 (5): Αποτελέσματα πειράματος A5

Πείραμα A5 (10×10, sub=10, iter=5, k=1000):

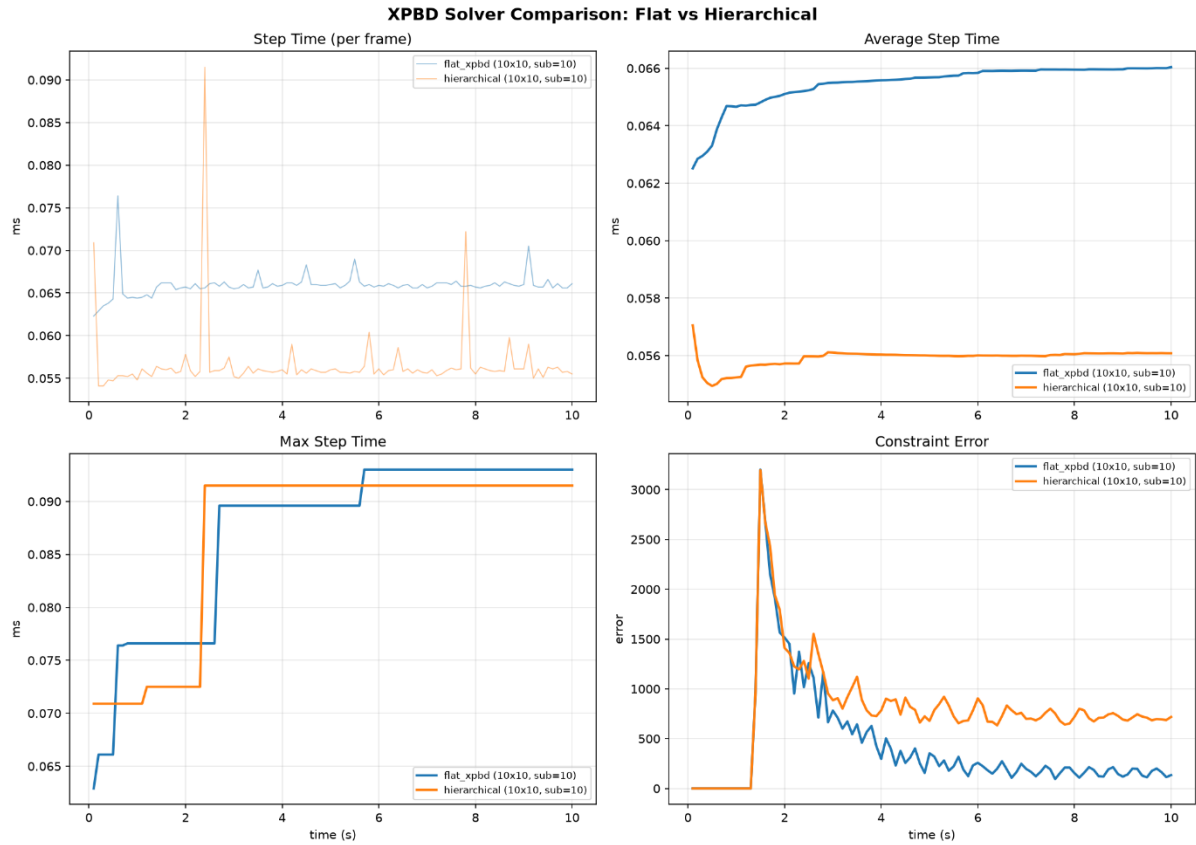
Καθώς αυξάνεται ο αριθμός επαναλήψεων, αυξάνεται και ο χρόνος υπολογισμού βήματος και αυτό αντικατοπτρίζεται και στις δύο μεθόδους. Παρόλα αυτά, το κέρδος σε χρόνο από την ιεραρχική μέθοδο έναντι την flat, είναι ακόμα πολύ αισθητό. Για την ακρίβεια, η διαφορά μεταξύ τους πλέον έχει φτάσει περίπου 20%, η μεγαλύτερη απόκλιση που έχουμε δει μέχρι στιγμής. Ωστόσο, όπως και με τα προηγούμενα πειράματα, η σύγκλιση που υποδηλώνεται από το σφάλμα περιορισμών παραμένει σχεδόν ίδια.



Σχήμα 4.4.1 (6): Αποτελέσματα πειράματος A6

Πείραμα A6 (10×10, sub=10, iter=5, k=1000, mass=0.1, spacing=50):

Οι χρόνοι βήματος για ένα ελαφρύ και πιο πυκνό πλέγμα, φαίνεται να είναι ίδιοι με το πείραμα A1, ακόμα και στο ποσοστό κέρδους της ιεραρχικής μεθόδου σε σχέση με την flat. Ωστόσο, μπορούμε να παρατηρήσουμε πως η ιεραρχική μέθοδος όχι μόνο έκοψε τον χρόνο του μέγιστου βήματος στη μέση, αλλά και πως παρουσιάστηκε αρκετά αργότερα στην προσομοίωση. Επιπλέον, η ιεραρχική μέθοδος φαίνεται να σταθεροποιείται με λιγότερες διακυμάνσεις σε σχέση με την flat μετά τις δύο πρώτες κρούσεις του πλέγματος με το δάπεδο.



Σχήμα 4.4.1 (7): Αποτελέσματα πειράματος A7

Πείραμα A7 (10×10, sub=10, iter=5, k=1000, mass=2, spacing=150):

Όπως και πριν, πάλι παρατηρούνται παρόμοιοι χρόνοι στα βήματα και στις διαφορές μεταξύ των μεθόδων. Το μέγιστο βήμα επίσης, φαίνεται να είναι παρόμοιο και στις δύο μεθόδους όσον αφορά τον χρόνο άφιξης και το μέγεθος του. Το σφάλμα περιορισμών όμως αποτυπώνει ένα ενδιαφέρον εύρημα. Σε περιπτώσεις που το πλέγμα είναι βαρύ και αραιό, η ιεραρχική μέθοδος φαίνεται να ζορίζεται να κρατήσει την δομή του διότι οι περιορισμοί έχουν πολύ μεγαλύτερο εύρος κίνησης, και επηρεάζονται πιο πολύ από την βαρύτητα. Το αποτέλεσμα είναι μεγαλύτερο σφάλμα περιορισμών σε σχέση με τη flat μέθοδο, η οποία επιλύει με σειρά εισαγωγής και δεν χρειάζεται να κρατήσει καμία δομή του πλέγματος ορθή.

Εάν συγκρίνουμε όλα τα αποτελέσματα ανά πείραμα μεταξύ τους, μπορούν να προκύψουν ενδιαφέρον ευρήματα.

Πείραμα	Avg Flat (ms)	Avg Hier (ms)	Διαφορά	Max Flat (ms)	Max Hier (ms)	Σφάλμα
A1	0.066	0.056	15%	0.12	0.12	~1000–5000

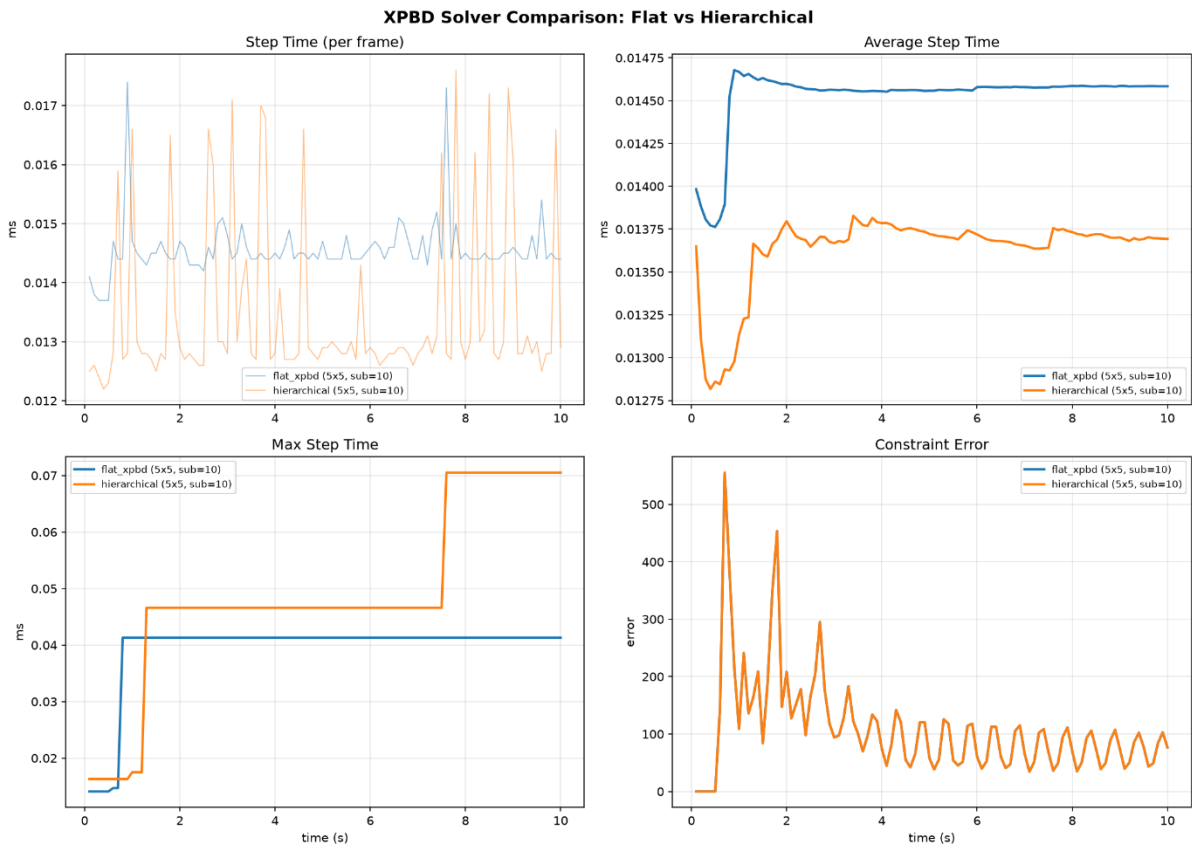
A2	0.021	0.019	10%	0.21	0.075	~500–5000
A3	0.29	0.24	17%	0.39	0.32	~5000–22000
A4	0.066	0.057	14%	0.12	0.11	~100–1900
A5	0.29	0.24	20%	0.4	0.33	~100–1900
A6	0.067	0.056	16%	0.2	0.1	<50
A7	0.066	0.057	14%	0.093	0.092	Flat ~200, Hier ~700

Πίνακας 4.4.1 : Αποτελέσματα πειραμάτων ομάδας A

Μπορούμε να παρατηρήσουμε πως η ιεραρχική μέθοδος έχει ταχύτερους χρόνους σε κάθε σενάριο. Όσο αυξάνεται ο αριθμός επαναλήψεων, τόσο μεγαλύτερη είναι η διαφορά κέρδους μεταξύ των δύο. Επιπλέον, πέρα από το πείραμα A7, το σφάλμα περιορισμών είναι ταυτόσημο και στις δύο μεθόδους. Η χειρότερη περίπτωση επίσης ευνοεί την ιεραρχική εκτός από πειράματα στα οποία και οι δύο τιμές συγκλίνουν.

4.4.2 Κλιμάκωση μεγέθους (Ομάδα Β)

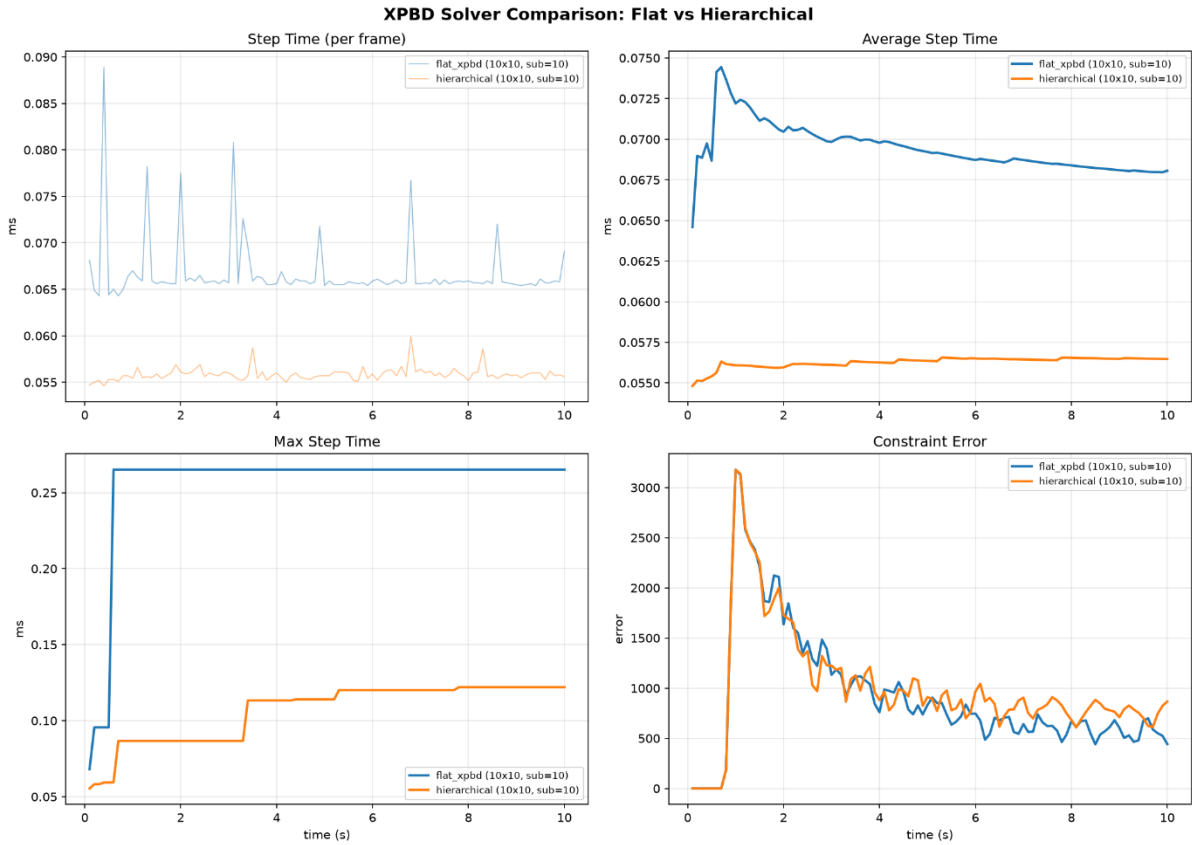
Η συγκεκριμένη ομάδα εξετάζει την κλιμάκωση μεταξύ των δύο μεθόδων κρατώντας σταθερές όλες τις υπόλοιπες παραμέτρους και αυξάνοντας το μέγεθος του πλέγματος.



Σχήμα 4.4.2 (1): Αποτελέσματα πειράματος B1

Πείραμα B1 (5×5, sub=10, iter=1, k=1000):

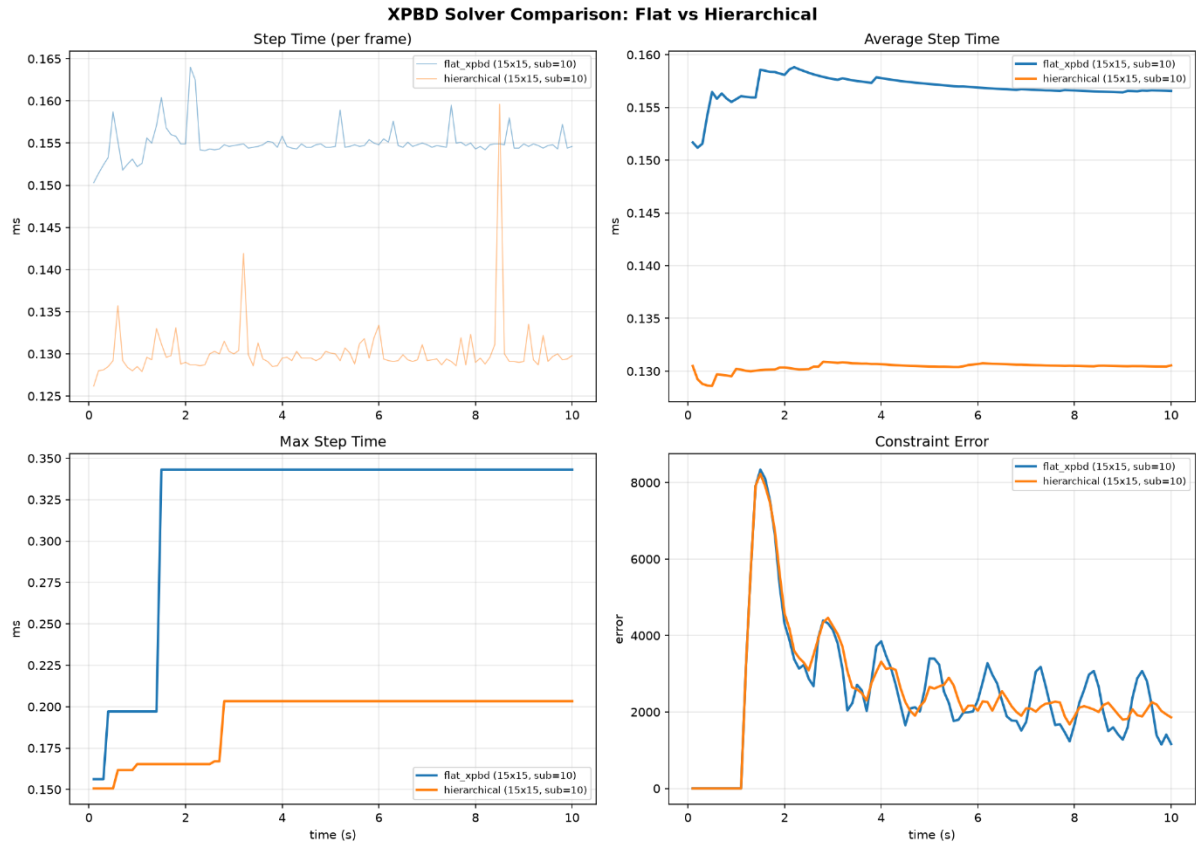
Σε αντίθεση με την προηγούμενη ομάδα, δεν είναι αναγκαίο το πείραμα αναφοράς, σκοπός είναι η σύγκριση μεταξύ όλων των πειραμάτων επομένως κάθε πείραμα έχει την ίδια βαρύτητα. Σε μικρό πλέγμα 5x5, συνεχίζει να φαίνεται η ίδια διαφορά μεταξύ των δύο μεθόδων με μειωμένους χρόνους, όπως είναι αναμενόμενο. Ακόμα και το σφάλμα περιορισμών παραμένει το ίδιο.



Σχήμα 4.4.2 (2): Αποτελέσματα πειράματος B2

Πείραμα B2 (10×10 , sub=10, iter=1, $k=1000$):

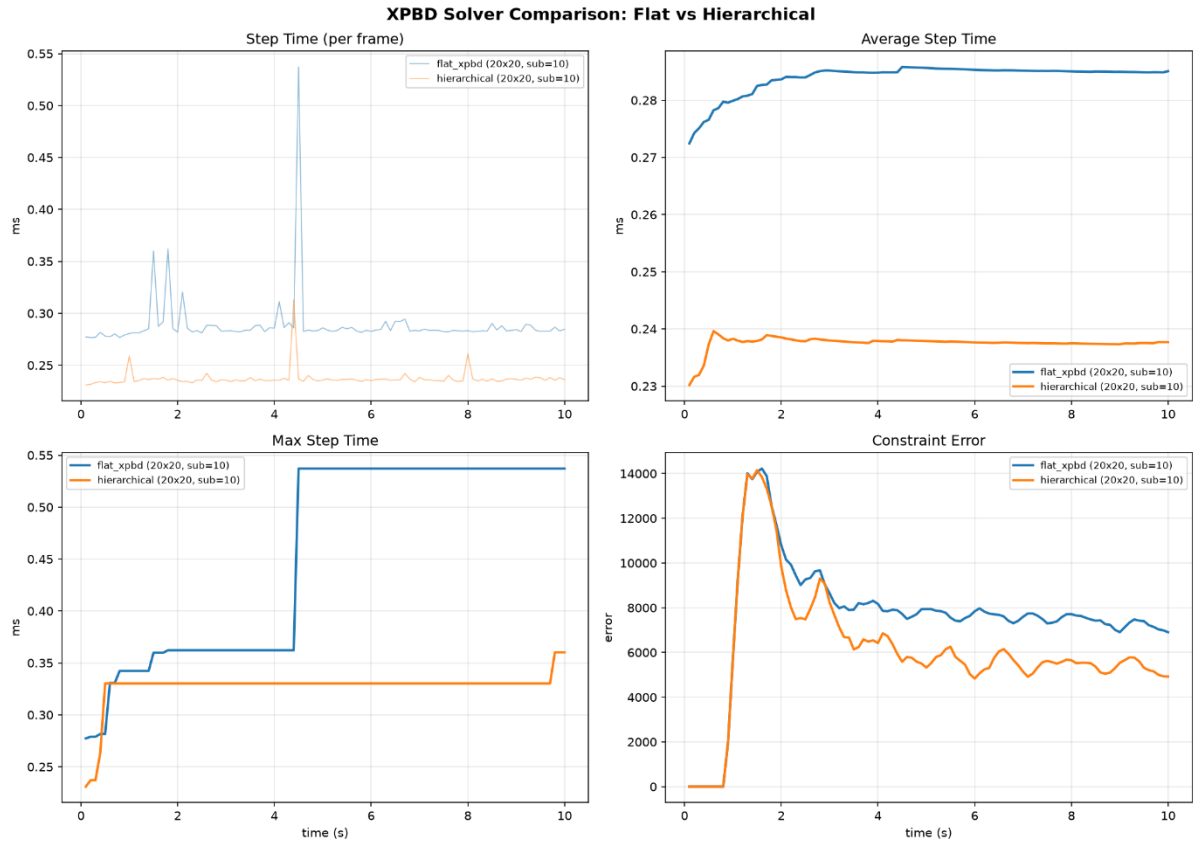
Το παρόν πείραμα είναι ίδιο με το πείραμα A1, ωστόσο με την εκτέλεση του ξανά, παρατηρούμε πως το μέγιστο βήμα είναι αντεστραμμένο σε σχέση με την προηγούμενη φορά. Στο B2, πρώτα εμφανίστηκε στην flat μέθοδο, και έπειτα στην ιεραρχική.



Σχήμα 4.4.2 (3): Αποτελέσματα πειράματος B3

Πείραμα B3 (15×15, sub=10, iter=1, k=1000):

Εδώ ξεκινάει να εμφανίζεται μια κλιμάκωση μεταξύ περιορισμών, χρόνου και σφαλμάτων περιορισμών. Φαίνεται πως η αναλογία κόστους ανά περιορισμό είναι σταθερή και γραμμική, δηλαδή $O(n)$.



Σχήμα 4.4.2 (4): Αποτελέσματα πειράματος B4

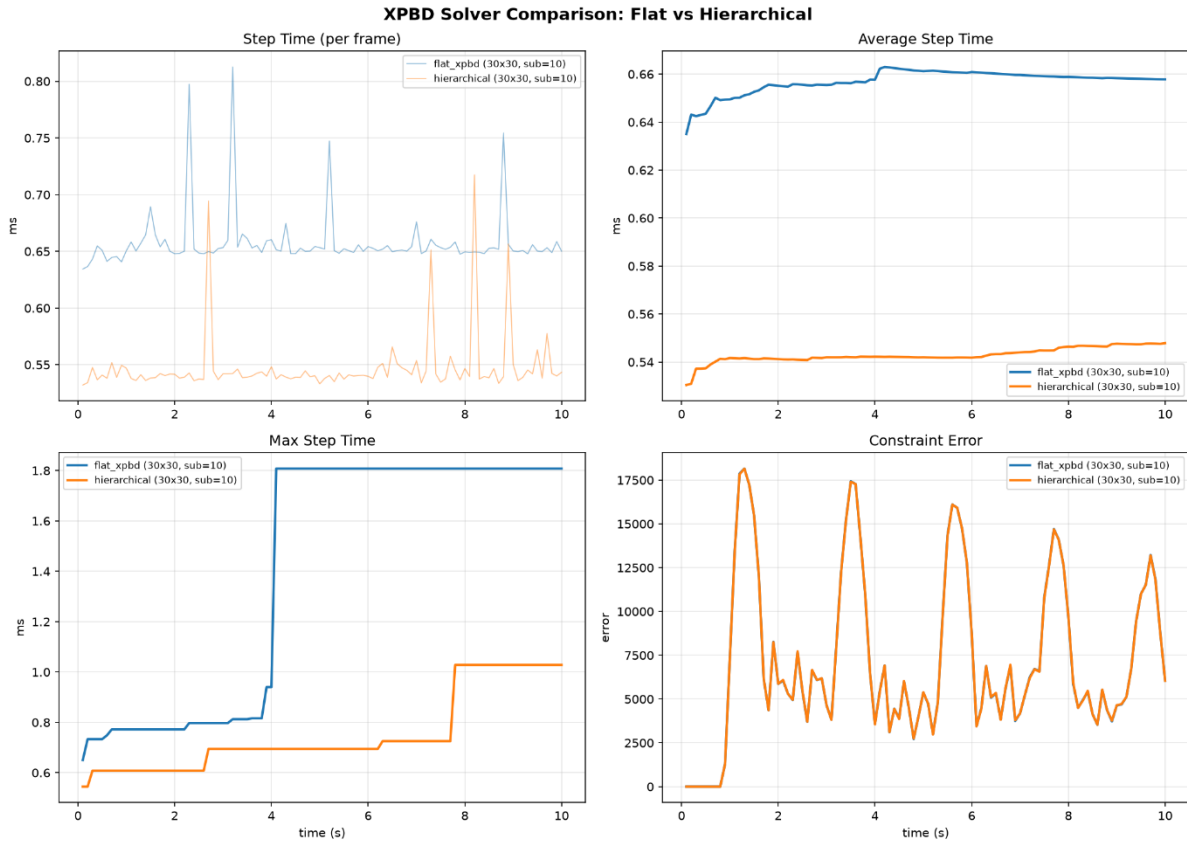
Πείραμα B4 (20×20, sub=10, iter=1, k=1000):

Το συγκεκριμένο πείραμα επιβεβαιώνει την γραμμική σχέση κλιμάκωσης που προαναφέρθηκε. Εάν συγκρίνουμε τα αποτελέσματα μεταξύ τους, η σχέση είναι καθαρή.

	B1	B2	B3	B4
Περιορισμοί	56	261	616	1121
Avg (Flat)	0.015	0.068	0.158	0.29ms
Avg (Hier)	0.014	0.057	0.130	0.24ms
Κορυφή σφάλματος	~500	~3200	~8000	~14000

Πίνακας 4.4.2 (1): Αποτελέσματα πειραμάτων μικρών πλεγμάτων

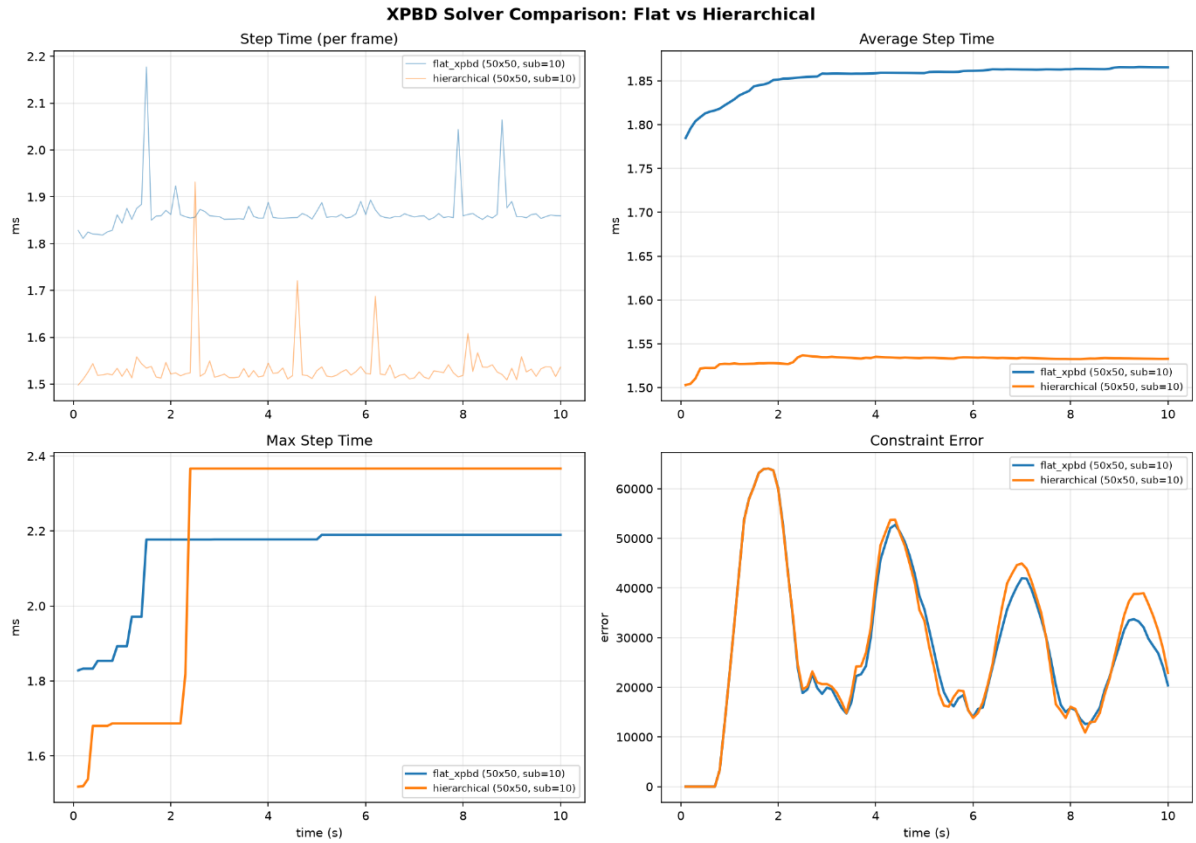
Η γραμμικότητα συνεχίζεται και πλέον επιβεβαιώνεται σε τέσσερα σημεία. Ένα αξιοσημείωτο γεγονός είναι πως η ιεραρχική μέθοδος παρουσιάζει μειωμένο σφάλμα στο παρόν πείραμα σε σχέση με τη flat μέθοδο. Πιθανόν να έχει να κάνει με το μέγεθος τους σώματος σε σχέση με το βαθμό δυσκαμψίας.



Σχήμα 4.4.2 (5): Αποτελέσματα πειράματος B5

Πείραμα B5 (30×30 , sub=10, iter=1, k=3000):

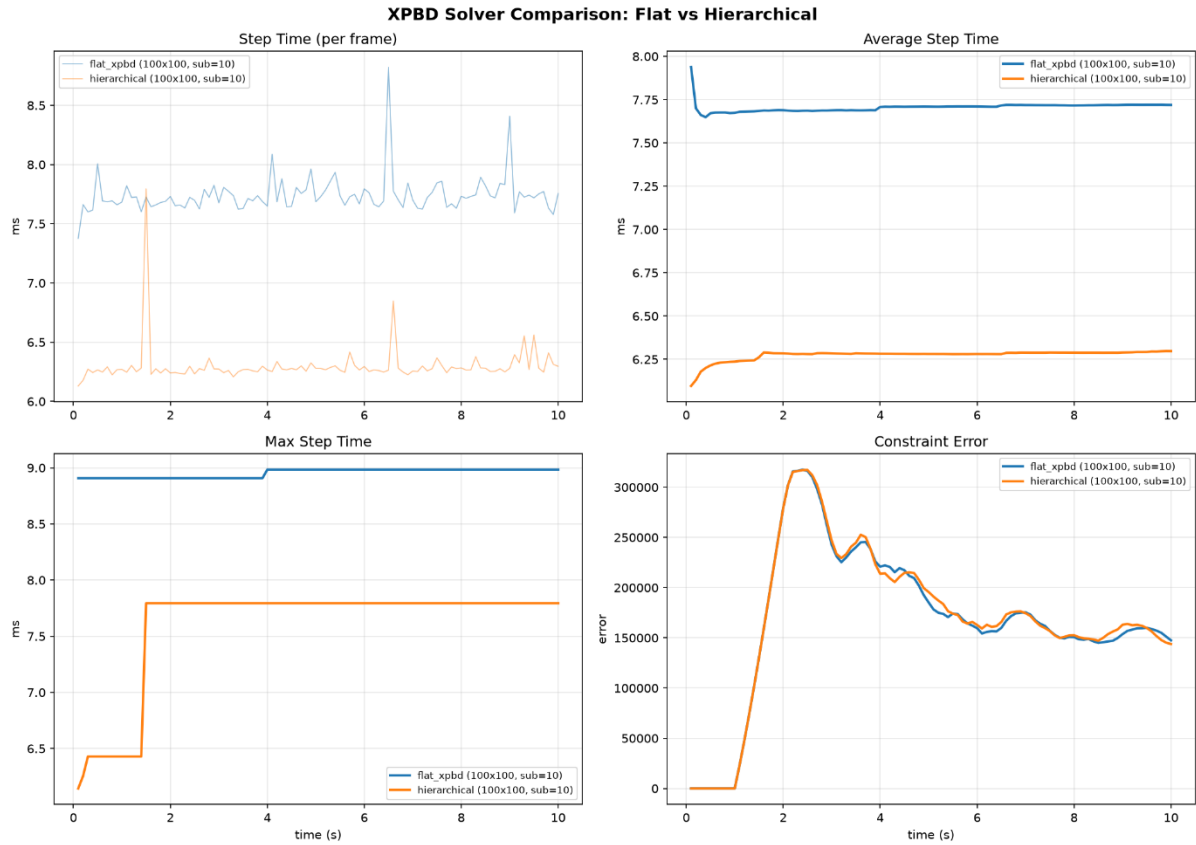
Πλέον ξεκινάει η δειγματοληψία για μεγάλα πλέγματα. Για άλλη μια φορά η διαφορά παραμένει σταθερή σε όλους τους τομείς και επιπλέον, η flat μέθοδος εμφανίζει νωρίτερα το μέγιστο βήμα. Είναι αξιοσημείωτο πως με την αλλαγή της δυσκαμνίας για την υποστήριξη τόσο μεγάλου σώματος, το σφάλμα περιορισμών είναι ακριβώς το ίδιο.



Σχήμα 4.4.2 (6): Αποτελέσματα πειράματος B6

Πείραμα B6 (50×50, sub=10, iter=1, k=3000)

Όπως και πριν, οι αναλογίες των τιμών παραμένουν ίδιες, ωστόσο πλέον το μέγιστο σφάλμα είναι μεγαλύτερο στην ιεραρχική μέθοδο. Παρόλα αυτά, δεν φαίνεται να έχουν σημαντική διαφορά αλλά ξεκινάνε να εμφανίζονται ασynchρονισμοί μεταξύ των δύο μεθόδων στο σφάλμα περιορισμών, γεγονός που υποδεικνύει πως αν είναι σωστά ρυθμιζόμενη η δυσκαμψία, και οι δυο μέθοδοι μπορούν να έχουν ακριβώς το ίδιο σφάλμα.



Σχήμα 4.4.2 (7): Αποτελέσματα πειράματος B7

Πείραμα B7 (100×100, sub=10, iter=1, k=3000):

Πλέον το πλέγμα έχει γίνει μη ρεαλιστικό για προσομοιώσεις πραγματικού χρόνου ωστόσο μπορεί να παρέχει χρήσιμα δεδομένα.

Πείραμα	Περιορισμοί	Avg Flat (ms)	Avg Hier (ms)	Διαφορά	Max Flat (ms)	Max Hier (ms)	Frame Budget Flat	Frame Budget Hier
B5	2581	0.66	0.54	15%	1.80	1	4.00%	3.30%
B6	7301	1.87	1.54	18%	2.19	2.37	11%	9.30%
B7	29301	7.70	6.30	18%	8.90	7.80	46%	38%

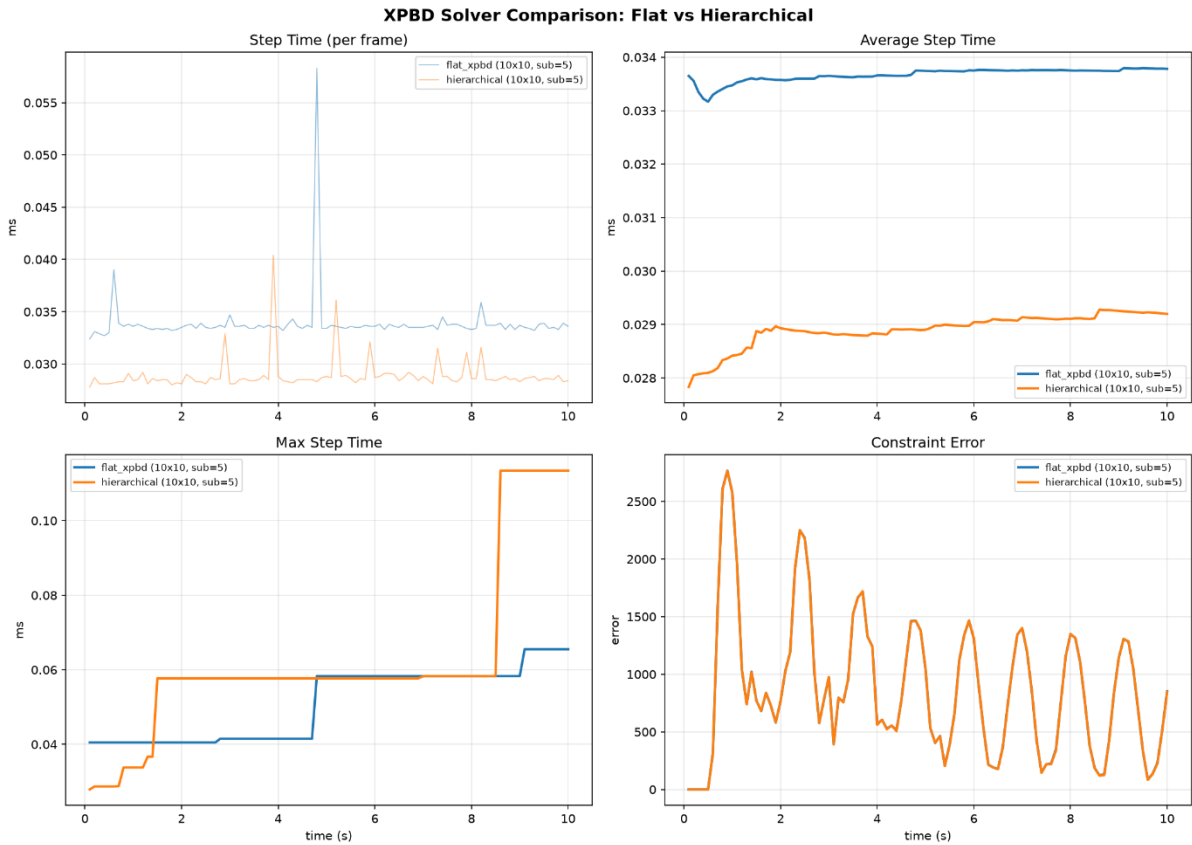
Πίνακας 4.4.2 (2): Αποτελέσματα πειραμάτων μεγάλων πλεγμάτων

Αρχικά, πλέον γνωρίζουμε πως ακόμα και αν το πλέγμα αυξηθεί σε μεγάλο βαθμό, η διαφορά ταχύτητας μεταξύ των δύο μεθόδων θα παραμείνει γύρω στην περιοχή ποσοστών 15% - 20%. Επιπλέον, η τάση για γραμμική κλιμάκωση φαίνεται να ισχύει ακόμα με τον χρόνο και τους περιορισμούς να πολλαπλασιάζονται x11. Το frame budget αναφέρεται στο διαθέσιμο ποσό χρονοθυρίδας που έχει κάθε

μέθοδος, και σύμφωνα με τα δεδομένα, υπάρχει σταθερό κέρδος σε κάθε πείραμα. Στο πείραμα B7, μεταφράζεται σε περίπου 1.4ms σε σωσμένο χρόνο.

4.4.3 Υποβήματα έναντι επαναλήψεων (Ομάδα Γ)

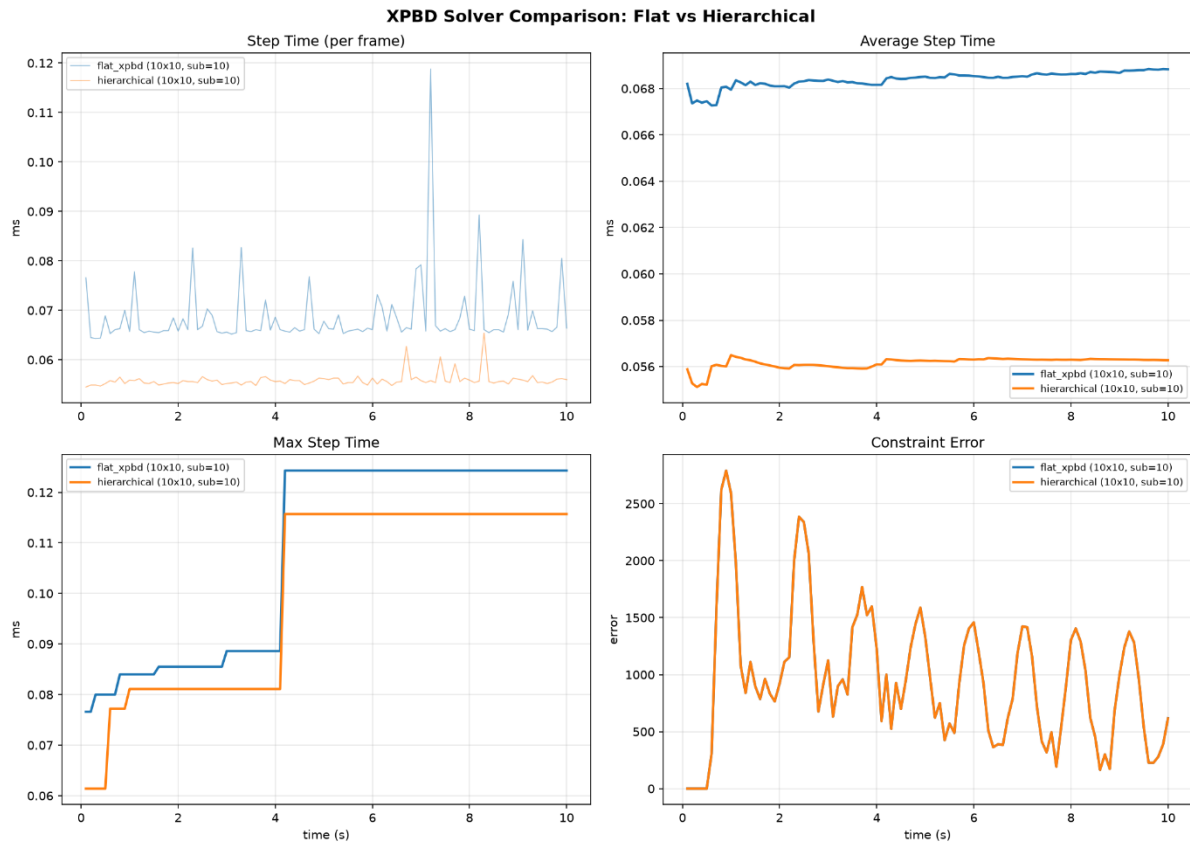
Με την ομάδα πειραμάτων Γ, θα φανερωθεί αν είναι βέλτιστη η αύξηση των επαναλήψεων ή των υποβημάτων σε κάθε μία από τις μεθόδους.



Σχήμα 4.4.3 (1): Αποτελέσματα πειράματος Γ1

Πείραμα Γ1 (10×10, sub=5, iter=1):

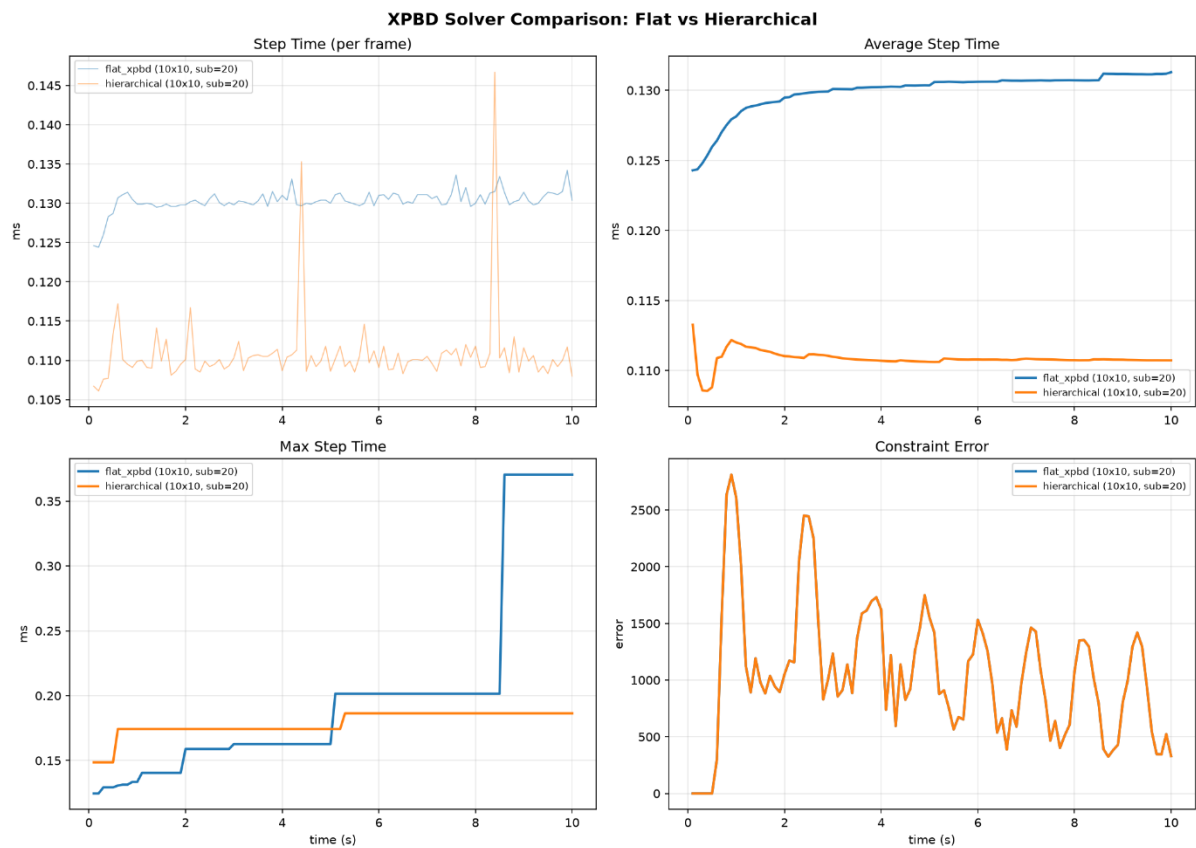
Όπως έγινε και στο πείραμα A2, με την μείωση την υποβημάτων αναλογεί και η μείωση χρόνου. Είναι λογικό συμπέρασμα εφόσον ο επιλυτής δεν χρειάζεται να δαπανήσει παραπάνω χρόνο κάνοντας διορθώσεις.



Σχήμα 4.4.3 (2): Αποτελέσματα πειράματος Γ2

Πείραμα Γ2 (10×10 , sub=10, iter=1):

Με το που διπλασιάσαμε τον αριθμό υποβημάτων, διπλασιάστηκε και ο χρόνος. Ωστόσο, το σφάλμα περιορισμών παραμένει το ίδιο. Ίσως η αύξηση των υποβημάτων, δεν βοηθάει στην μείωση σφαλμάτων περιορισμών σε καμία από τις δύο μεθόδους.



Σχήμα 4.4.3 (3): Αποτελέσματα πειράματος Γ3

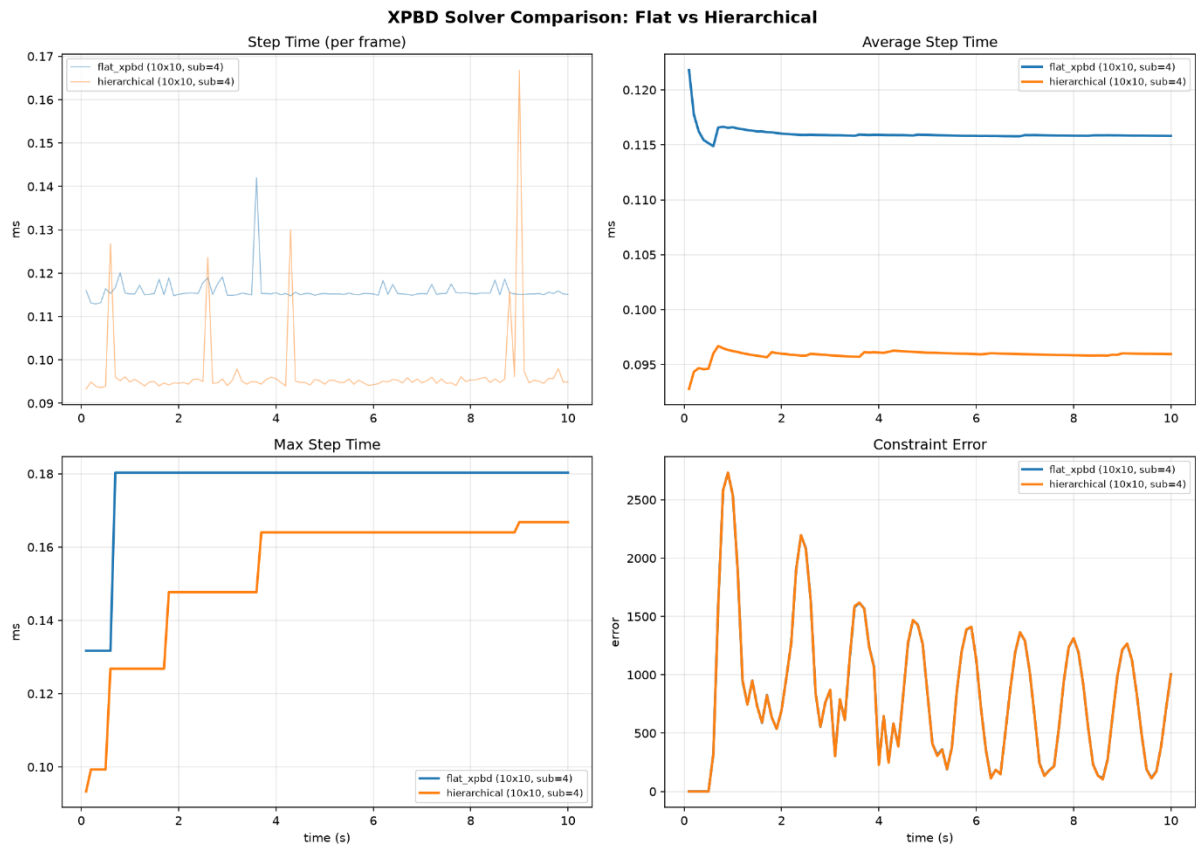
Πείραμα Γ3 (10×10, sub=20, iter=1):

Πράγματι, παρόλο που πλέον έχουν διπλασιαστεί τα υποβήματα ξανά, φαίνεται ο χρόνος να αυξάνεται, χωρίς κανένα όφελος στο σφάλμα περιορισμών.

Πείραμα	Substeps	Περάσματα	Avg Flat (ms)	Avg Hier (ms)	Σφάλμα (σταθ.)
Γ1	5	5	3%	3%	~500–1600
Γ2	10	10	7%	6%	~500–1600
Γ3	20	20	13%	11%	~500–1600

Πίνακας 4.4.3 (1): Αποτελέσματα πειραμάτων υποβημάτων

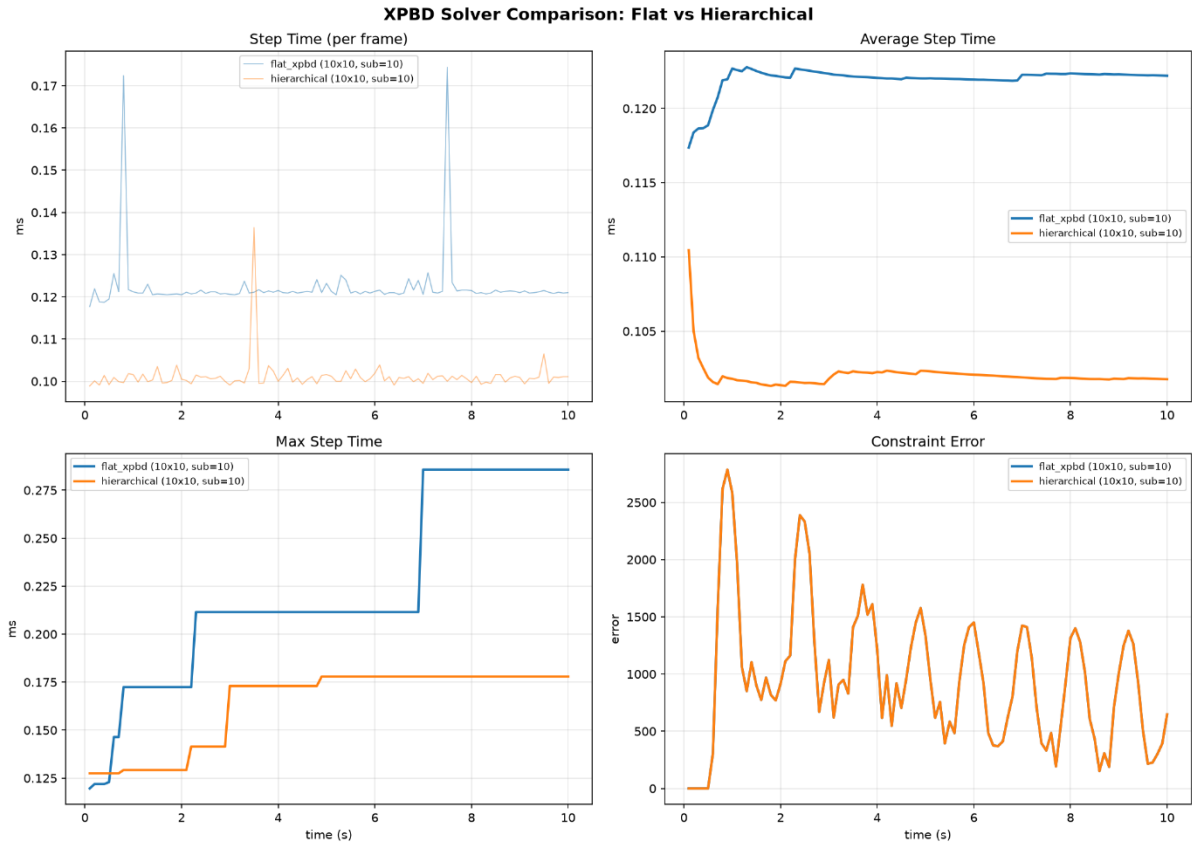
Το σφάλμα παραμένει σταθερό και στις δύο μεθόδους, πράγμα που σημαίνει ότι τα υποβήματα δεν εξυπηρετούν καμία από τις δύο.



Σχήμα 4.4.3 (4): Αποτελέσματα πειράματος Γ4

Πείραμα Γ4 (10×10 , sub=5, iter=4):

Με 20 περάσματα αντί 5, παρατηρείται μια οριακή βελτίωση στο σταθεροποιημένο σφάλμα (~1400 έναντι ~1600), κατά τα άλλα, όλα τα υπόλοιπα αποτελέσματα είναι αναμενόμενα.



Σχήμα 4.4.3 (5): Αποτελέσματα πειράματος Γ5

Πείραμα Γ5 (10×10, sub=10, iter=2):

Εδώ η κατανομή είναι πιο ισορροπημένη 10 υποβήματα με 2 επαναλήψεις. Τα μικρότερα χρονικά υποβήματα σημαίνουν μικρότερες αρχικές παραμορφώσεις ανά υποβήμα, αλλά μόνο 2 ευκαιρίες σύγκλισης ανά υποβήμα αντί 4. Ο μέσος χρόνος είναι 0.123ms της flat μεθόδου και 0.101ms της ιεραρχικής μεθόδου με ταυτόσημο σφάλμα.

Πείραμα	Υποβήματα	Επαναλήψεις	Περάσματα	Avg Flat (ms)	Avg Hier (ms)	Σφάλμα κορυφής	Σφάλμα (σταθ.)
Γ4	5	4	2000%	12%	0.096	~2700	~500–1400
Γ5	10	2	2000%	12%	0.101	~2600	~500–1400

Πίνακας 4.4.3 (2): Αποτελέσματα πειραμάτων επαναλήψεων

Παρόλο που τα δύο πειράματα εκτελούν τον ίδιο αριθμό περασμάτων και παράγουν πρακτικά ίδιο σφάλμα, το Γ4 είναι ~6% ταχύτερο. Οι επιπλέον επαναλήψεις δεν έχουν αυτό το κόστος απλώς ξαναδιατρέχουν τους περιορισμούς. Συνεπώς, αν ο στόχος είναι η μέγιστη ακρίβεια ανά χιλιοστό του δευτερολέπτου, χρήση λιγότερων υποβημάτων και περισσότερων επαναλήψεων είναι προτιμότερη.

4.5 Συζήτηση αποτελεσμάτων

Με βάση όλα τα διεξαγμένα πειράματα, μπορούμε να καταλήξουμε σε βασικές θεωρίες. Αρχικά όσον αφορά την ομάδα Α, τα κέρδη ταχύτητας είναι φανερά. Σε οποιοδήποτε πείραμα παρατηρείται σταθερό κέρδος ~15%, μια σημαντική βελτίωση σε σχέση με τον παραδοσιακό τρόπο επίλυσης. Η ομάδα Β φανερώνει πως εμφανίζεται μια κλιμάκωση μεταξύ περιορισμών, χρόνου και σφαλμάτων περιορισμών με γραμμικότητα $O(n)$ ανεξάρτητα από την μέθοδο επίλυσης. Ακόμα και σε ακραίες περιπτώσεις ωστόσο, η ιεραρχική μέθοδος καταλαμβάνει μικρότερο ποσοστό της χρονοθυρίδας. Τέλος η ομάδα Γ αποκαλύπτει πως και στις δύο μεθόδους η επιλογή περισσότερων επαναλήψεων αντί υποδημάτων είναι επιθυμητή. Σε γενικά πλαίσια η σύγκλιση και το σφάλμα περιορισμών ήταν ταυτόσημα μεταξύ των δύο μεθόδων, ωστόσο υπήρχε μια εξαίρεση στην ομάδα Α όσον αφορά την δυσκαμψία που χρήζει περισσότερη μελέτη.

Είναι σημαντική επίσης η αναφορά των περιορισμών της διεξαγωγής πειραμάτων. Αρχικά, τα δεδομένα που παράχθηκαν από τις προσομοιώσεις δεν είναι ετερογενή. Όλες οι ομάδες πειραμάτων πήραν μέρος στο ίδιο λειτουργικό σύστημα με το ίδιο υλισμικό. Δεν μπορούμε να είμαστε σίγουροι για την συμπεριφορά σε άλλου είδους μηχανήματα. Επιπλέον, η μελέτη εξέτασε μόνο δισδιάστατα πλέγματα με έναν μοναδικό περιορισμό που ήταν συγκεκριμένα δομημένος. Είναι πολύ πιθανό τα αποτελέσματα να ήταν εντελώς διαφορετικά αν τα είδη περιορισμών ήταν διαφορετικά (όπως περιορισμός όγκου κ.α.), ή η δομή της ιεραρχίας ήταν φτιαγμένη με άλλον τρόπο.

Κεφάλαιο 5ο: Συνεισφορά και μελλοντική εργασία

5.1 Συνεισφορά της εργασίας

Η ιδέα της προτεραιοποίησης περιορισμών σε ένα XPBD σύστημα δεν έχει μελετηθεί εκτενώς στη βιβλιογραφία. Η εργασία προτείνει μια απλή αλλά αποτελεσματική δομή δύο επιπέδων (δομικοί πρώτοι, διατμητικοί δεύτεροι) και τεκμηριώνει πειραματικά τα οφέλη και τους περιορισμούς της.

Το πλήρες, ανοιχτού κώδικα plugin για τη Godot προσφέρει ιδιαίτερη χρηστικότητα σε χρήστες της μηχανής ή ερευνητές. Το plugin είναι έτοιμο προς χρήση: ο χρήστης προσθέτει έναν κόμβο SoftBody2D, ρυθμίζει παραμέτρους μέσω του Inspector, και εναλλάσσει μεταξύ επιλυτές χωρίς μεταγλώττιση. Η αρχιτεκτονική GDExtension εξασφαλίζει τοπική απόδοση C++ χωρίς τροποποίηση του πυρήνα της μηχανής.

Τα 19 πειράματα τριών ομάδων καλύπτουν ένα ευρύ φάσμα παραμέτρων και παρέχουν ποσοτικά δεδομένα για τη σχέση ταχύτητας, ακρίβειας και κλιμάκωσης δεδομένα που μπορούν να αξιοποιηθούν ως σημείο αναφοράς για μελλοντικές εργασίες.

5.2 Προτάσεις μελλοντικής εργασίας

Υπάρχουν πολλές ανοιχτές κατευθύνσεις για βελτίωση και περεταίρω έρευνα. Η εισαγωγή της τρίτης διαστάσεις, αντιμετωπίζει πολύ μεγαλύτερη ποσότητα σωματιδίων και περιορισμών, ενώ παράλληλα παρουσιάζει καινούργιους περιορισμούς.

Επιπλέον, η παραλληλοποίηση και επιτάχυνση GPU είναι αναγκαία για την βέλτιστη αξιοποίηση του υλισμικού οποιουδήποτε υπολογιστή. Μέσω αυτών, θα ήταν εφικτή η προβολή πλέγματος 100x100 σε πραγματικό χρόνο.

Τέλος, η συγκριτική έρευνα θα μπορούσε να επεκταθεί και να συμπεριλάβει άλλες μεθόδους όπως Projective Dynamics [13] ή η Chebyshev επιτάχυνση. θα παρείχε μια πληρέστερη εικόνα της θέσης του στο τοπίο των σύγχρονων solvers.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[17] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, Springer, 2012.

[18] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2010.

Internet Site

[15] Godot Engine, "godot-cpp: C++ bindings for the Godot script API," GitHub repository, <https://github.com/godotengine/godot-cpp>, 2024.

Paper in Conference Proceedings

[2] E. Sifakis and J. Barbič, "FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction," in *ACM SIGGRAPH 2012 Courses*, Los Angeles, CA, 2012, pp. 1–50.

[4] M. Macklin, M. Müller, and N. Chentanez, "XPBD: Position-based simulation of compliant constrained dynamics," in *Proc. 9th International Conference on Motion, Interaction and Games (MIG '16)*, San Francisco, CA, 2016, pp. 49–54.

[9] X. Provot, "Deformation constraints in a mass-spring model to describe rigid cloth behavior," in *Proc. Graphics Interface '95*, Quebec, Canada, 1995, pp. 147–154.

[10] M. Müller, "Hierarchical position based dynamics," in *Proc. Workshop on Virtual Reality Interactions and Physical Simulations (VRIPhys)*, Grenoble, France, 2008, pp. 1–10.

[11] M. Macklin, K. Storey, M. Lu, P. Terdiman, N. Chentanez, S. Jeschke, and M. Müller, "Small steps in physics simulation," in *Proc. 18th ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '19)*, Los Angeles, CA, 2019, pp. 1–7.

[14] J. Bender and D. Koschier, "Divergence-free smoothed particle hydrodynamics," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '15)*, Los Angeles, CA, 2015, pp. 147–155.

Journal Articles

[1] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," *Computer Graphics Forum*, vol. 25, no. 4, pp. 809–836, Dec. 2006.

[3] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 109–118, Apr. 2007.

[5] J. Bender, M. Müller, and M. Macklin, "A survey on position-based simulation methods in computer graphics," *Computer Graphics Forum*, vol. 36, no. 6, pp. 228–251, Sep. 2017.

- [6] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun, "Efficient simulation of inextensible cloth," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 26, no. 3, Article 49, Jul. 2007.
- [7] M. Tournier, M. Nesme, B. Gilles, and F. Faure, "Stable constrained dynamics," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 34, no. 4, Article 132, Jul. 2015.
- [8] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 33, no. 4, Article 153, Jul. 2014.
- [12] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 24, no. 3, pp. 471–478, Jul. 2005.
- [13] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly, "Projective dynamics: Fusing constraint projections for fast simulation," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 33, no. 4, Article 154, Jul. 2014.
- [16] O. Astrachan, "OO Overkill: When Simple Is Better Than Not," *SIGCSE Bulletin*, vol. 33, no. 1, pp. 230–234, 2001.
- [19] D. H. Bailey, "Twelve ways to fool the masses when giving performance results on parallel computers," *Supercomputing Review*, vol. 4, no. 8, pp. 54–55, 1991.