

## Υπολογιστική Γεωμετρία & Εφαρμογές 3Δ Μοντελοποίησης

Αριθμός Εργαστηρίου	Απαλλακτική
Επώνυμο	Λινάρδος
Όνομα	Αθανάσιος
Έτος	5 <sup>ο</sup>
ΑΜ	1059293

### ΕΡΩΤΗΜΑΤΑ

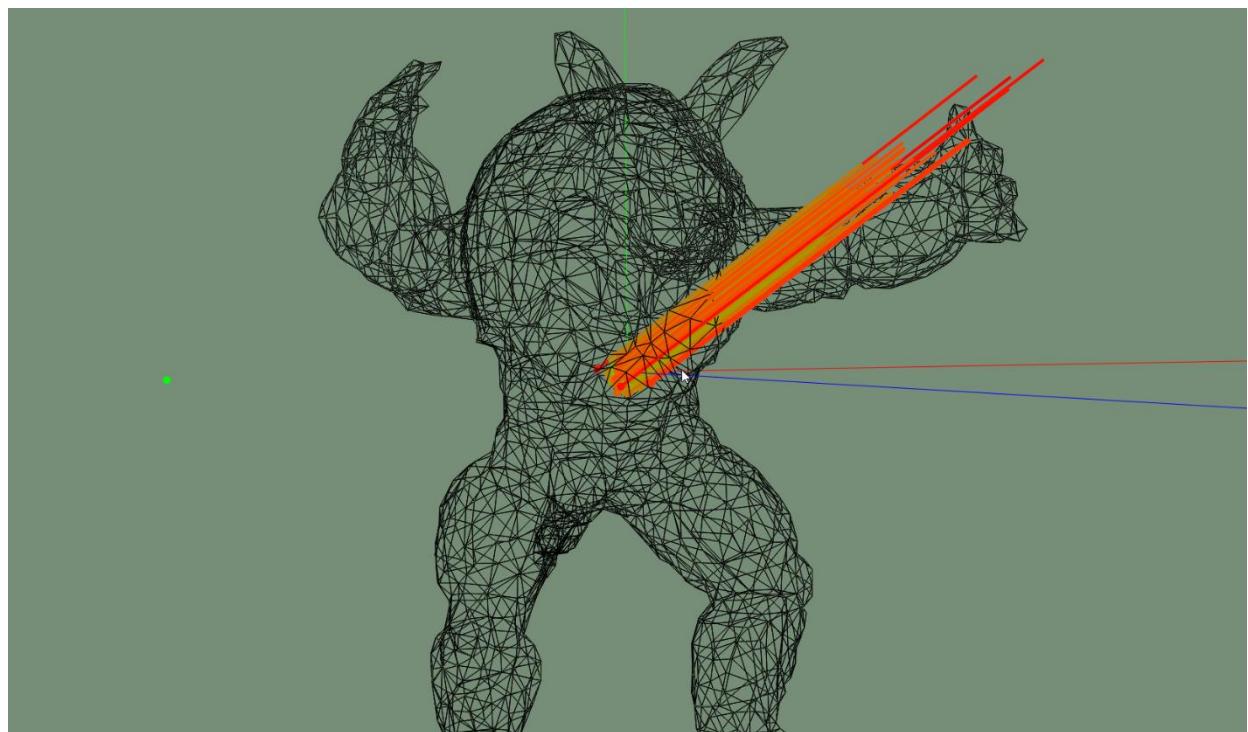
Τα παρακάτω ερωτήματα υλοποιήθηκαν σε περιβάλλον του vvr που μπορεί δυναμικά να επιλέγει ο χρήστης από τον φάκελο 'resources/obj' το mesh που θέλει να επεξεργαστεί. Με την χρήση των δοσμένων συντομεύσεων πληκτρολογίου μπορεί να εμφανίσει μόνο το πλέγμα του αντικειμένου , το εσωτερικό του , τους άξονες , να φορτώσει άλλο mesh ('r') , να ξεκινήσει την οριζόντια κίνηση του ( ' ) ή να τρέξει οποιοδήποτε από τα παρακάτω ερωτήματα , των οποίων το αποτέλεσμα παραμένει στην προσομοίωση μέχρι ο χρήστης να επιλέξει να τα κρύψει με την αντίστοιχη συντόμευση. Για τον υπολογισμό των πινάκων χρησιμοποιώ την βιβλιοθήκη «Eigen».

**A\_1)** Βρείτε τις διαφορικές συντεταγμένες του πλέγματος:

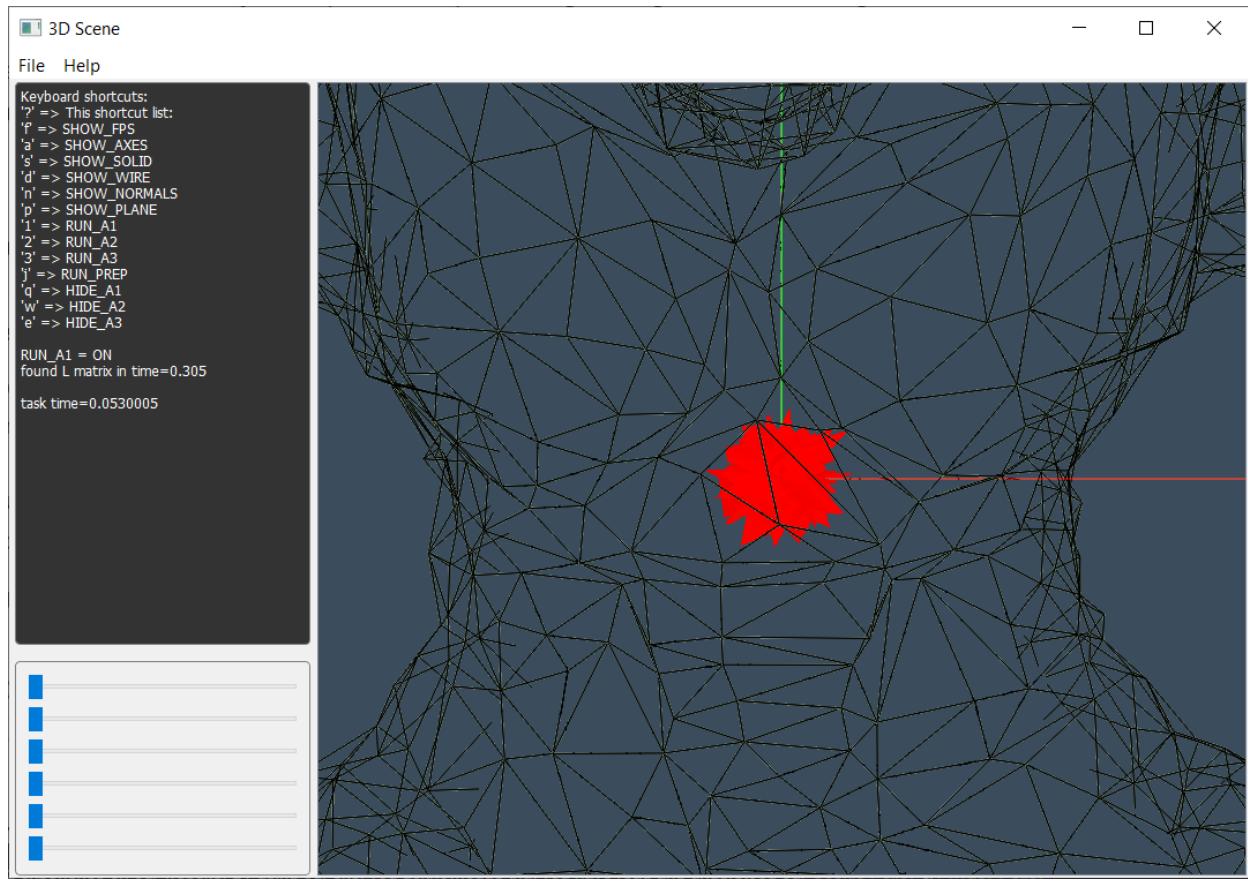
Για να υπολογίσω τις διαφορικές συντεταγμένες πρώτα υπολογίζω τους άμεσους γείτονες του κάθε σημείου του πλέγματος κατασκευάζοντας τον Laplace πίνακα. Τον βρίσκω από τη σχέση:  $L = I - D_{\text{inverse}} * A$  , όπου I είναι ο μοναδιαίος πίνακας ,  $D_{\text{inverse}}$  ο αντίστροφος του πίνακα  $D$ . Ο πίνακας  $D$  είναι ένας διαγώνιος που περιέχει στην διαγώνιό του τον αριθμό των συνδέσεων κάθε σημείου με τα υπόλοιπα , δηλαδή τον αριθμό των άμεσων γειτόνων του σημείου  $v_i$  στο  $dii$ . Η αντιστροφή του πίνακα αυτού γίνεται με την συνάρτηση  $SparseDiagonalInverse(D, D_{\text{inverse}}, verticesCount)$  που αντικαθιστά τις τιμές  $dii$  με  $1/dii$

. Ο πίνακας  $A$  έχει τιμή  $a_{ij} = 1$  αν είναι άμεσοι γείτονες τα σημεία  $v_i$  και  $v_j$  και 0 σε κάθε άλλη περίπτωση, ενώ και η διαγώνιος του είναι 0. Τελικά, οι διαφορικές συντεταγμένες υπολογίζονται από την σχέση:  $\text{Diff} = L * V$ , όπου  $V$  είναι ο πίνακας με τα σημεία του πλέγματος. Παρακάτω φαίνονται οι διαφορικές συντεταγμένες μέσα στο πλέγμα του αντικειμένου(τα κβαντισμένα μέτρα τους και οι συντεταγμένες τους αντίστοιχα) :

*Current Object:armadillo\_low\_low*



*Εικόνα 1*



Εικόνα 2

### A\_2) Πραγματοποιείστε Taubin smoothing(shrinking):

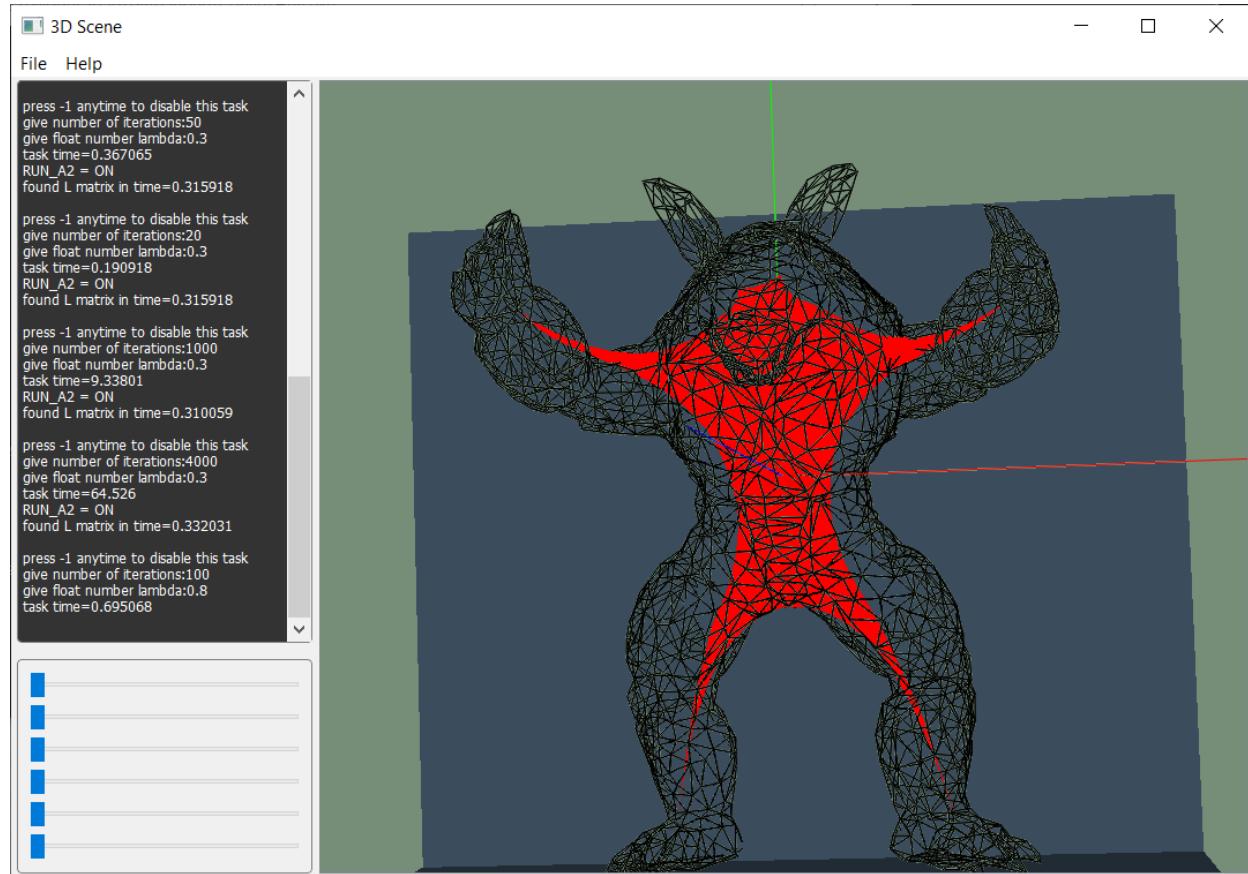
Για να πραγματοποιήσω εξομάλυνση του αντικειμένου αρκεί να φιλτράρω τον θόρυβο υψηλής συχνότητας από το σήμα που αναπαριστούν τα σημεία του πλέγματος. Αυτό το επιτυγχάνω χωρίς να περάσω στο πεδίο της συχνότητας με διαφοροποίηση πεπερασμένης διαφοράς των σημείων, που ισοδυναμεί με επαναληπτική πρόσθεση του πίνακα  $L$  στα σημεία σε κάθε διάσταση χρησιμοποιώντας μία παράμετρο  $\lambda$ :

$$P_i \leftarrow P_i + \lambda L(P_i), \text{ όπου } 0 < \lambda < 1$$

Για το  $\lambda$  επιλέγω μικρές τιμές, ώστε να μην γίνει τόσο εμφανής η συρρίκνωση του αντικειμένου, η οποία είναι αναπόφευκτη και για αρκετές επαναλήψεις τείνει η απεικόνιση του αποτελέσματος σε 1 σημείο. Με την χρήση πινάκων κατέληξα στην παρακάτω σχέση:

$$\mathbf{Diff}' = (\mathbf{I} - \lambda \mathbf{L}) * \mathbf{Diff} \text{ που εκτελείται επαναληπτικά και δίνει τις νέες συντεταγμένες των σημείων του πλέγματος στον πίνακα } \mathbf{Diff}.$$

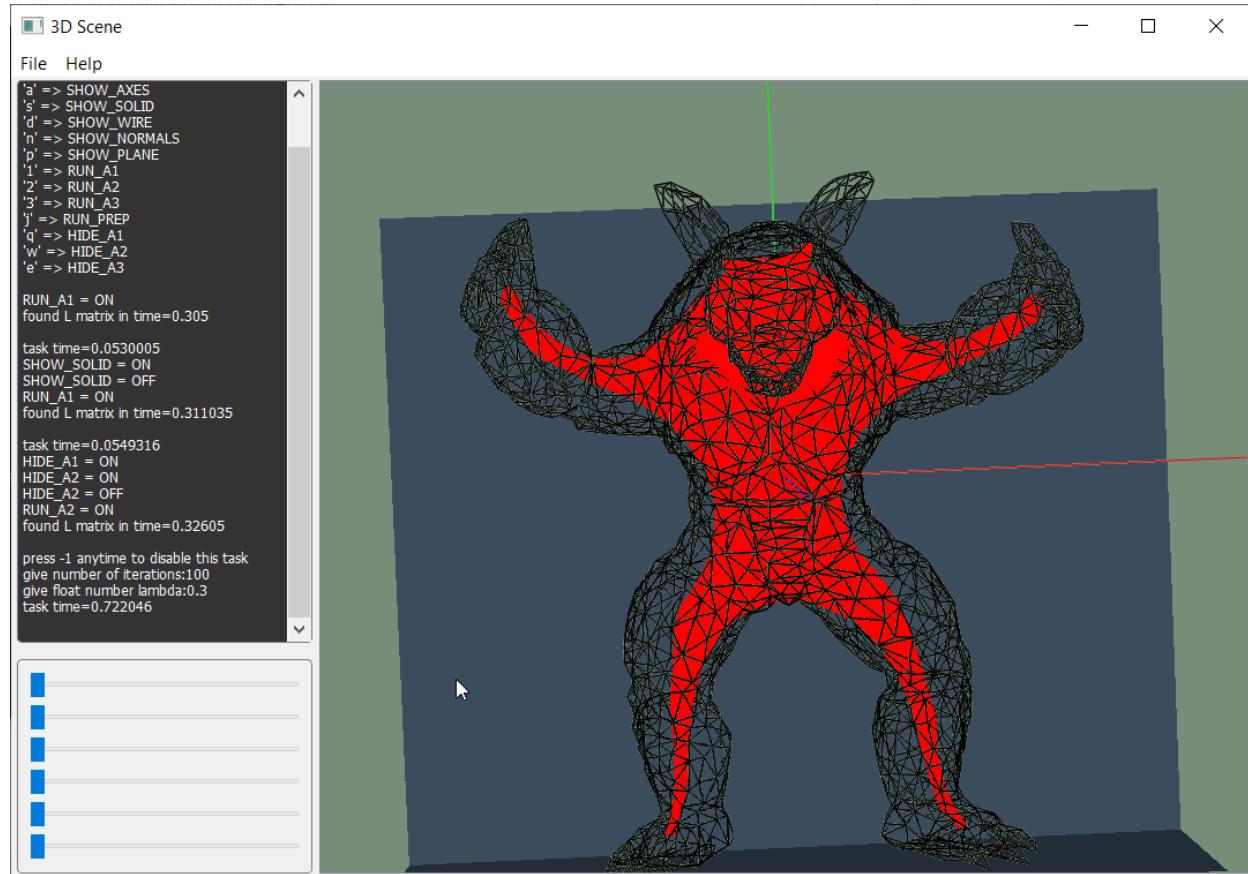
Για  $\lambda = 0.8$  και 100 επαναλήψεις παίρνω το παρακάτω αποτέλεσμα:



Εικόνα 3

Παρατηρώ ότι επειδή το  $\lambda$  είναι πολύ μεγάλο, πολύ γρήγορα το αντικείμενο συρρικνώνεται. Οπότε πρέπει να χρησιμοποιήσω μικρότερη τιμή για το  $\lambda$ .

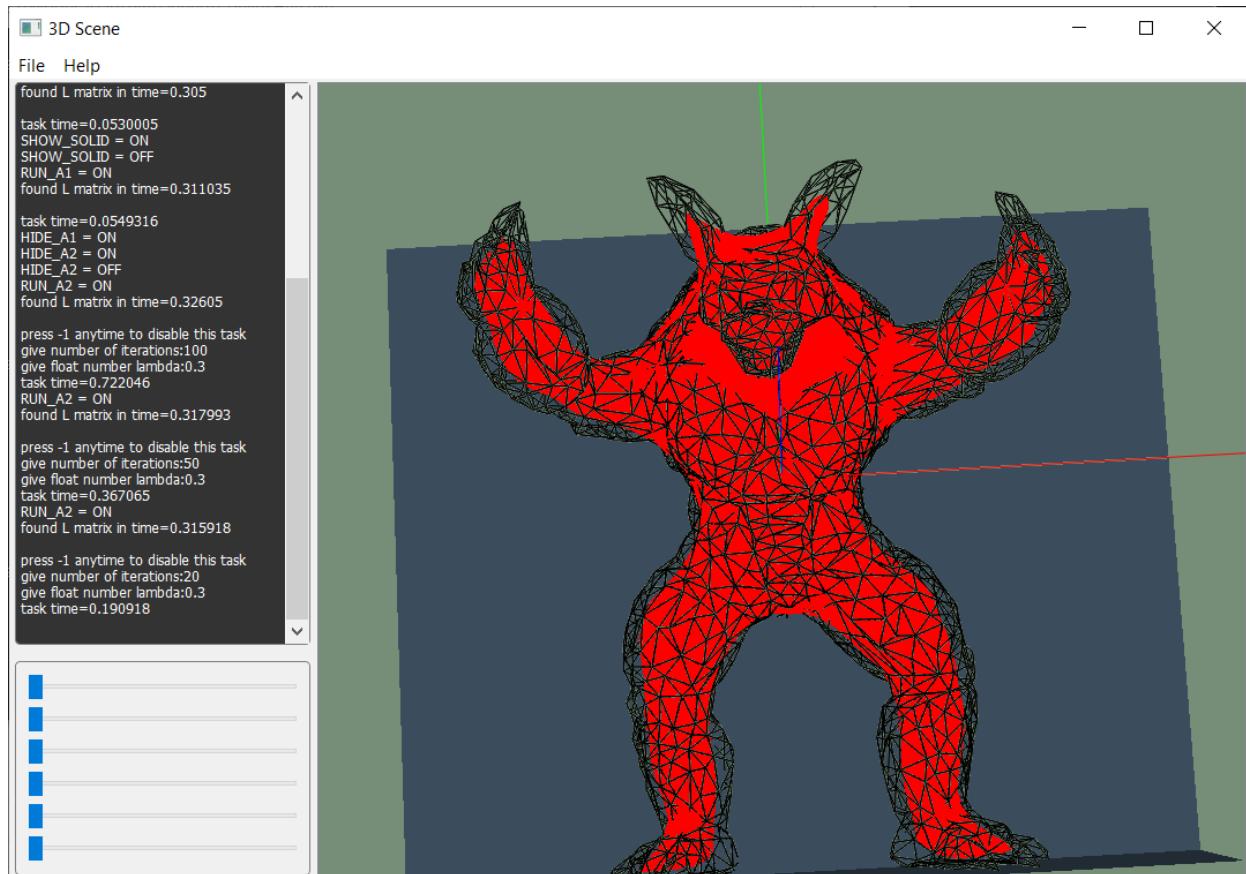
Για  $\lambda = 0.3$  και 100 επαναλήψεις παίρνω το παρακάτω αποτέλεσμα:



Εικόνα 4

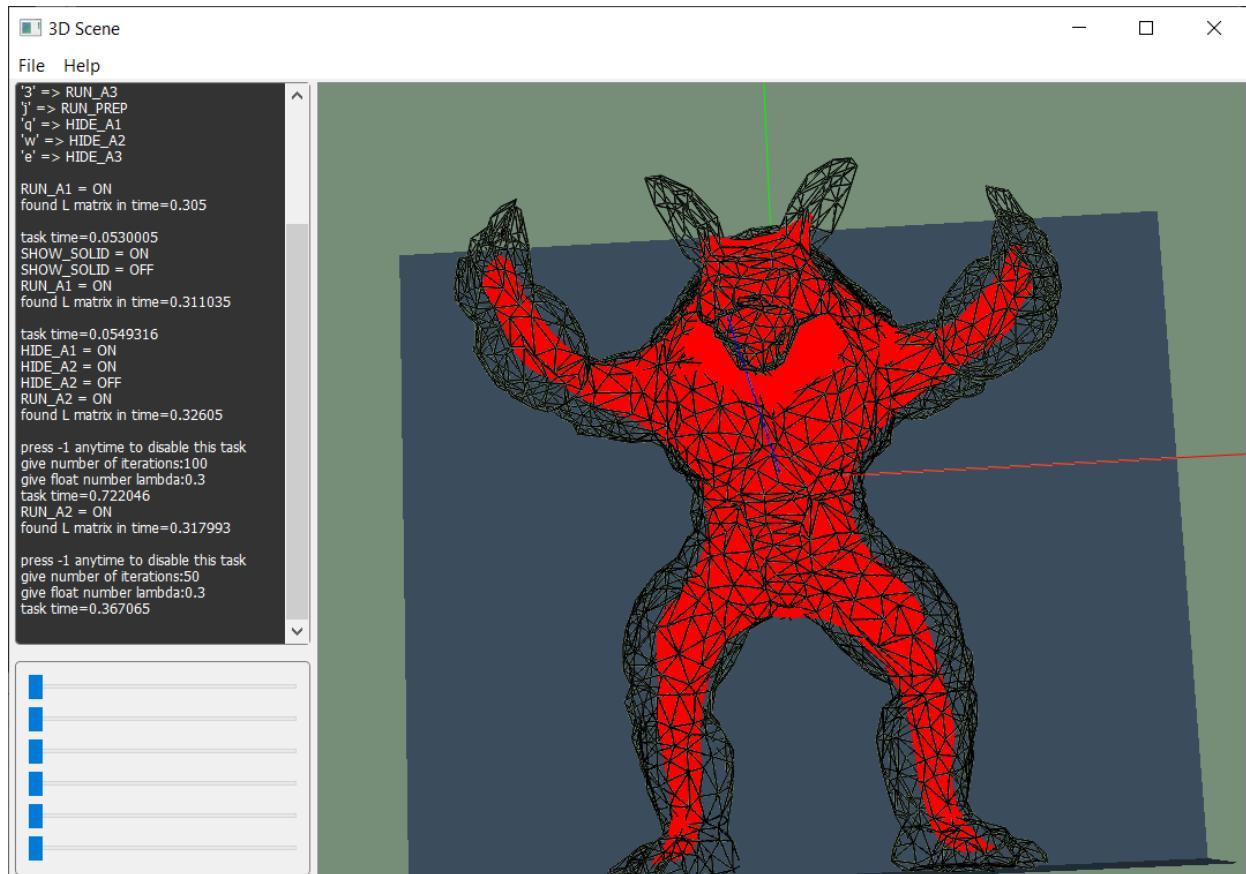
Είναι εμφανές ότι η συρρίκνωση του αντικειμένου είναι αρκετά μικρότερη. Μετά δοκίμασα διαφορετικό αριθμό επαναλήψεων για την ίδια τιμή του  $\lambda=0.3$ :

Για 20 επαναλήψεις:



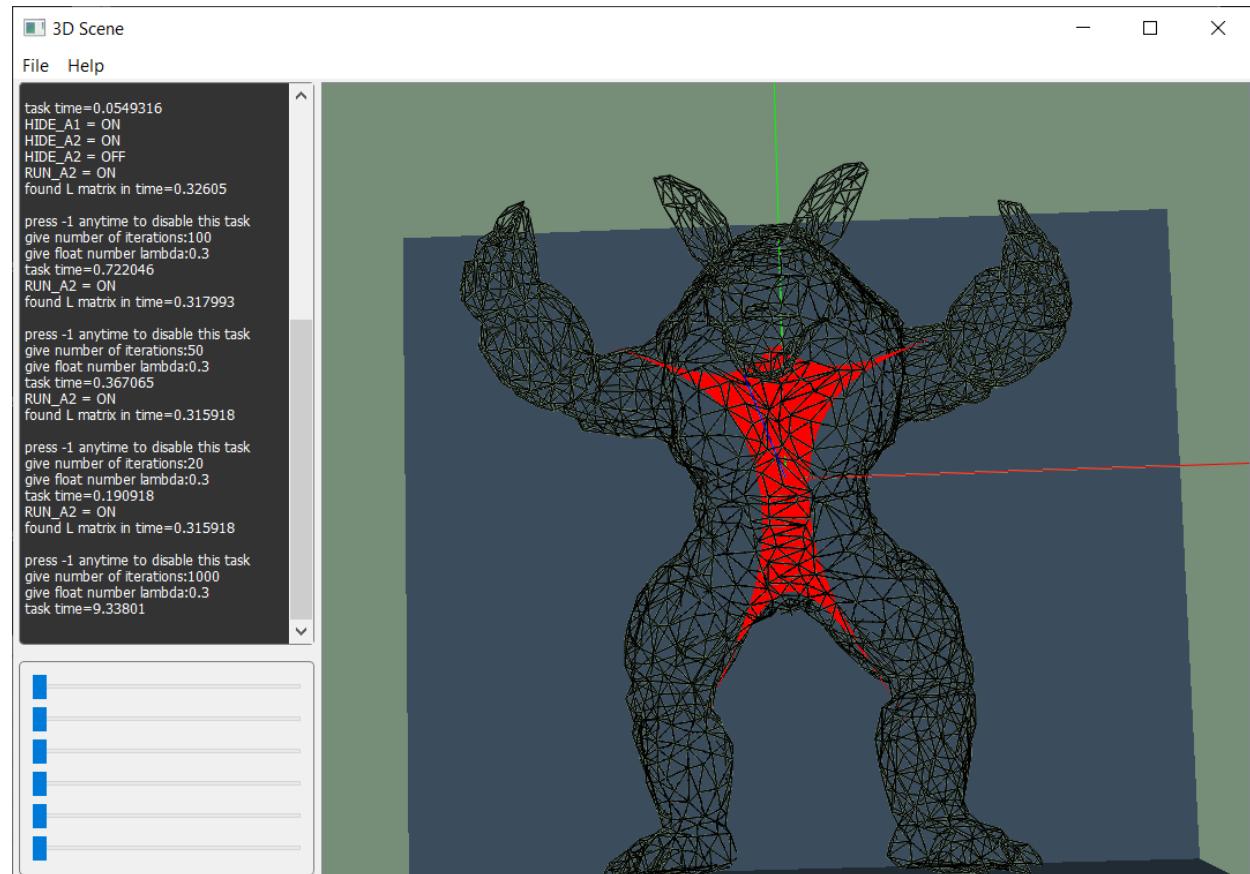
Εικόνα 5

Για 50 επαναλήψεις:



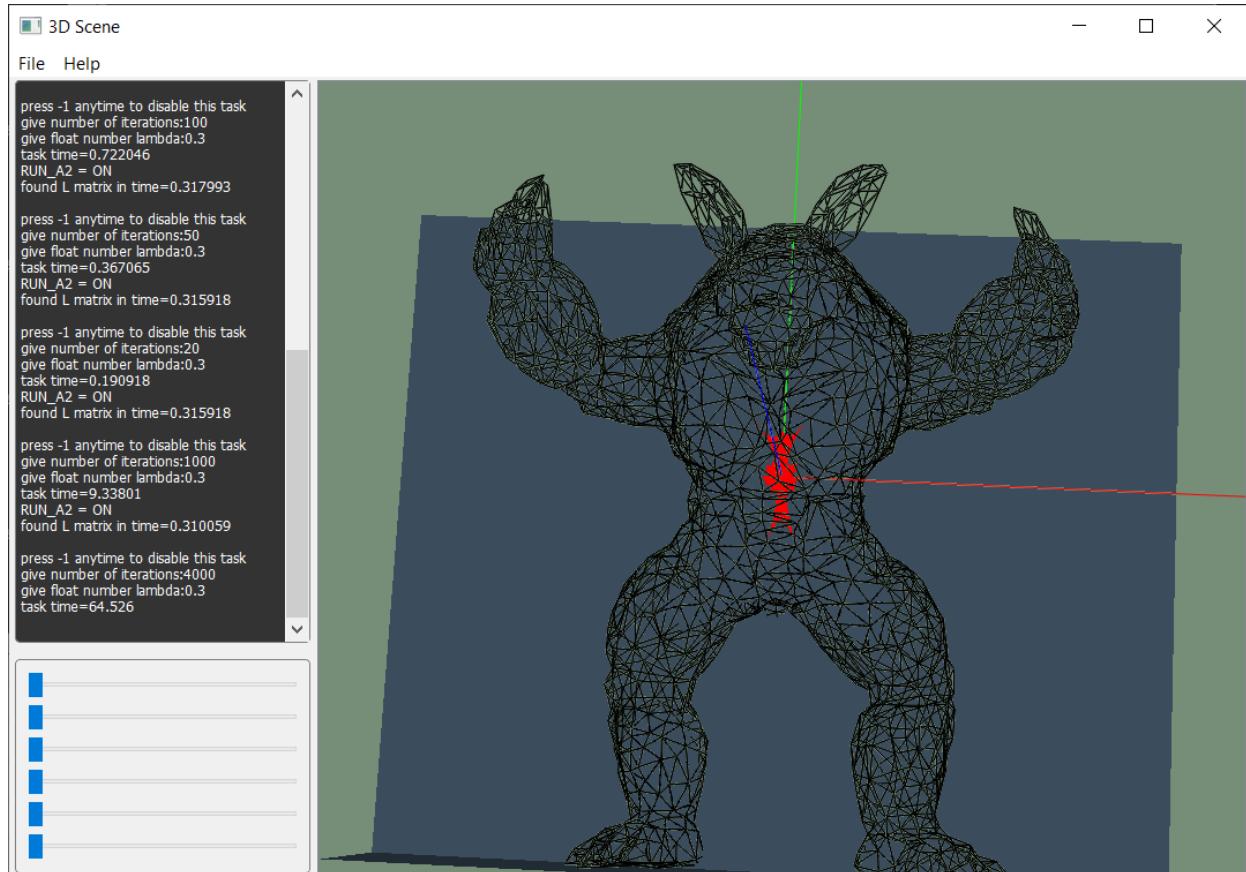
Εικόνα 6

Για 1000 επαναλήψεις:



Εικόνα 7

Για 4000 επαναλήψεις:



Εικόνα 8

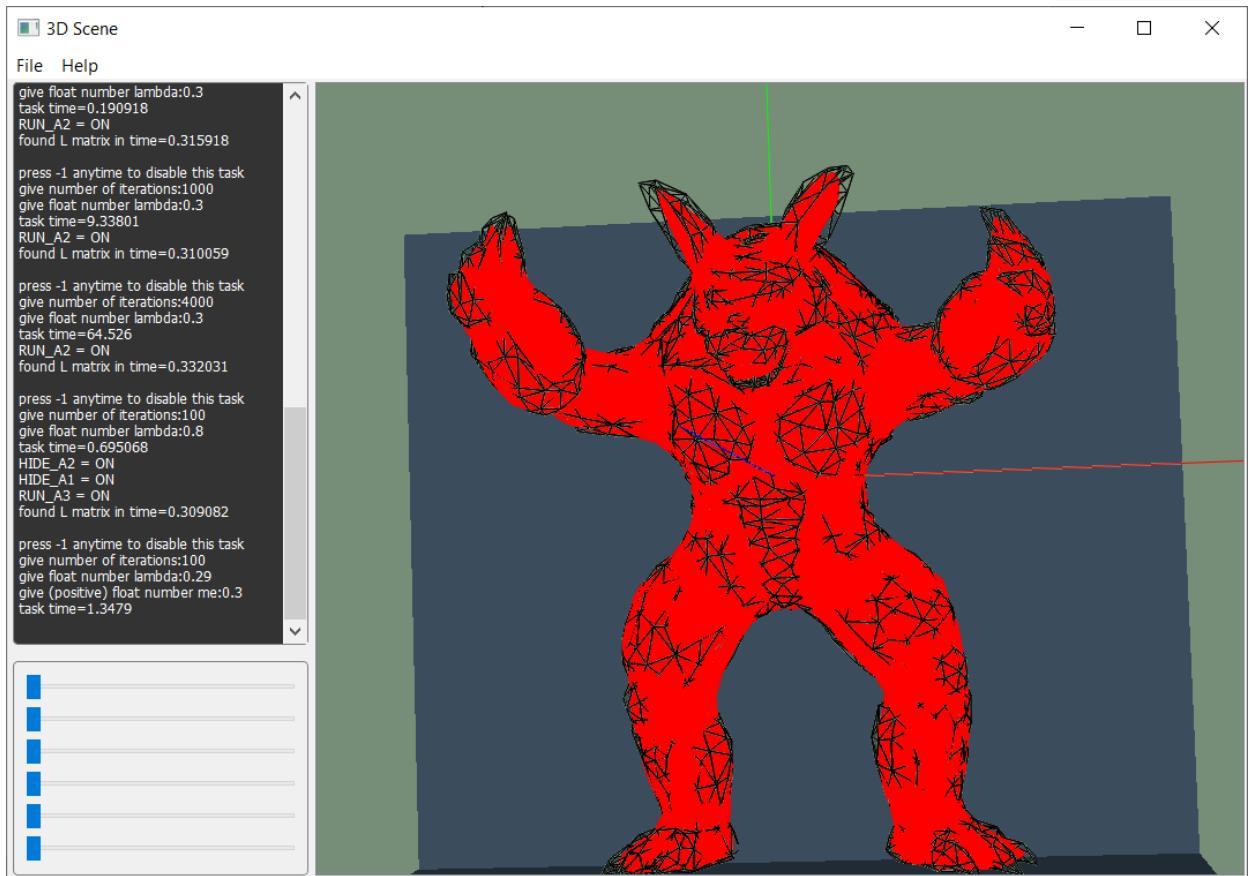
Από τα παραπάνω καταλήγω στο συμπέρασμα ότι ακόμα και με μικρότερο λ με αρκετές επαναλήψεις θα καταλήξει και πάλι το σχήμα να τείνει σε ένα σημείο, οπότε για αντικείμενα με περισσότερο θόρυβο που χρειάζονται πολλές επαναλήψεις δεν είναι κατάλληλη η μέθοδος που χρησιμοποιήσα.

### A\_3) Πραγματοποιείστε smart “Taubin inflation”:

Για να πετύχω την εξομάλυνση ενός πλέγματος χωρίς να γίνεται συρρίκνωσή του χρησιμοποίησα την λύση του Taubin, που υλοποιεί την ίδια μέθοδο με το προηγούμενο ερώτημα με την διαφορά ότι χρησιμοποιεί ένα παραπάνω βήμα επεξεργασίας των σημείων που φουσκώνει το αντικείμενο. Οπότε σε κάθε επανάληψη έχω ένα βήμα συρρίκνωσης με παράμετρο λ και ένα βήμα inflation με παράμετρο  $\mu < \theta$ , όπου  $|\mu| > \lambda$ . Η παράμετρος μ πρέπει να έχει λίγο μεγαλύτερο μέτρο από την λ, έτσι ώστε το αντικείμενο να διατηρεί το αρχικό του μέγεθος.

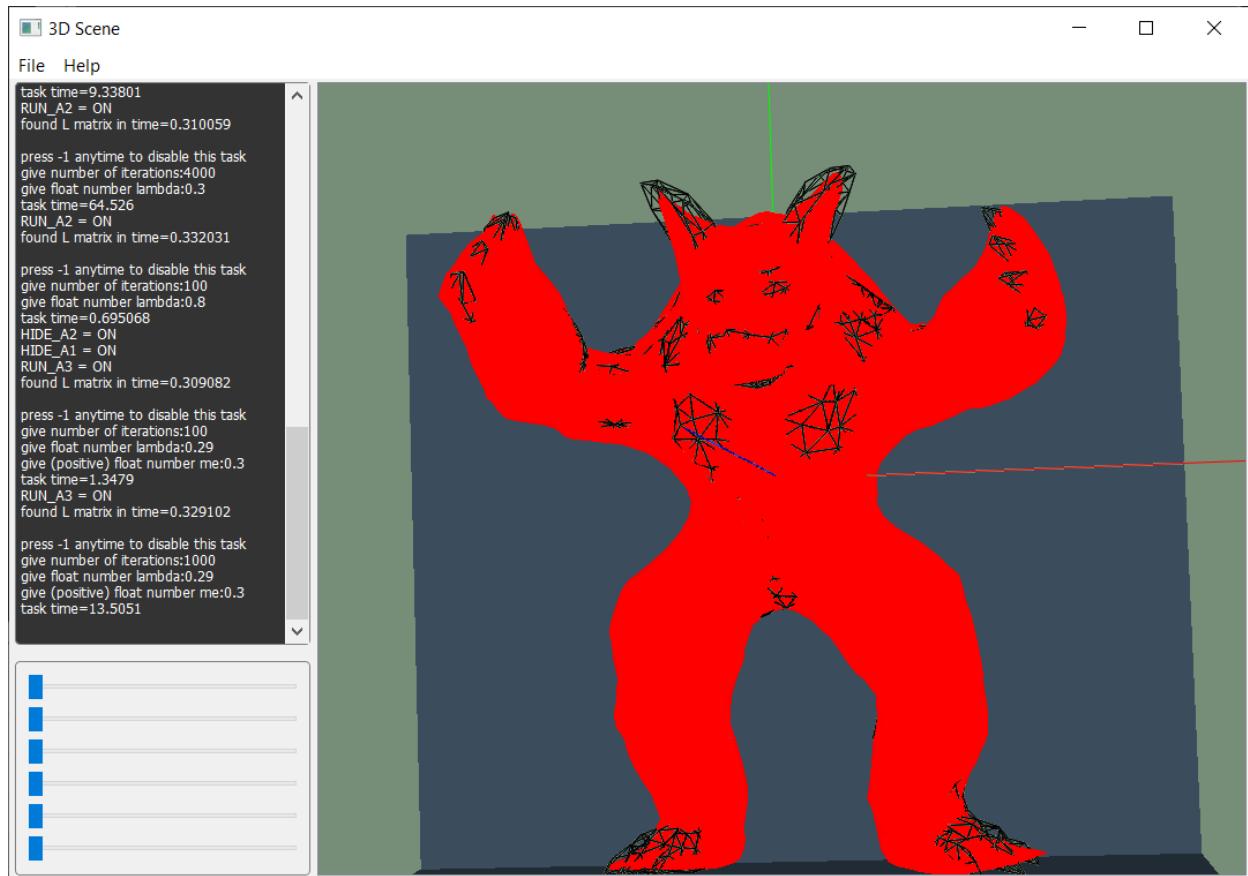
Επιλέγω  $\lambda=0.29$  και  $\mu=0.3$ :

→ για 100 επαναλήψεις:



Εικόνα 9

→ για 1000 επαναλήψεις:

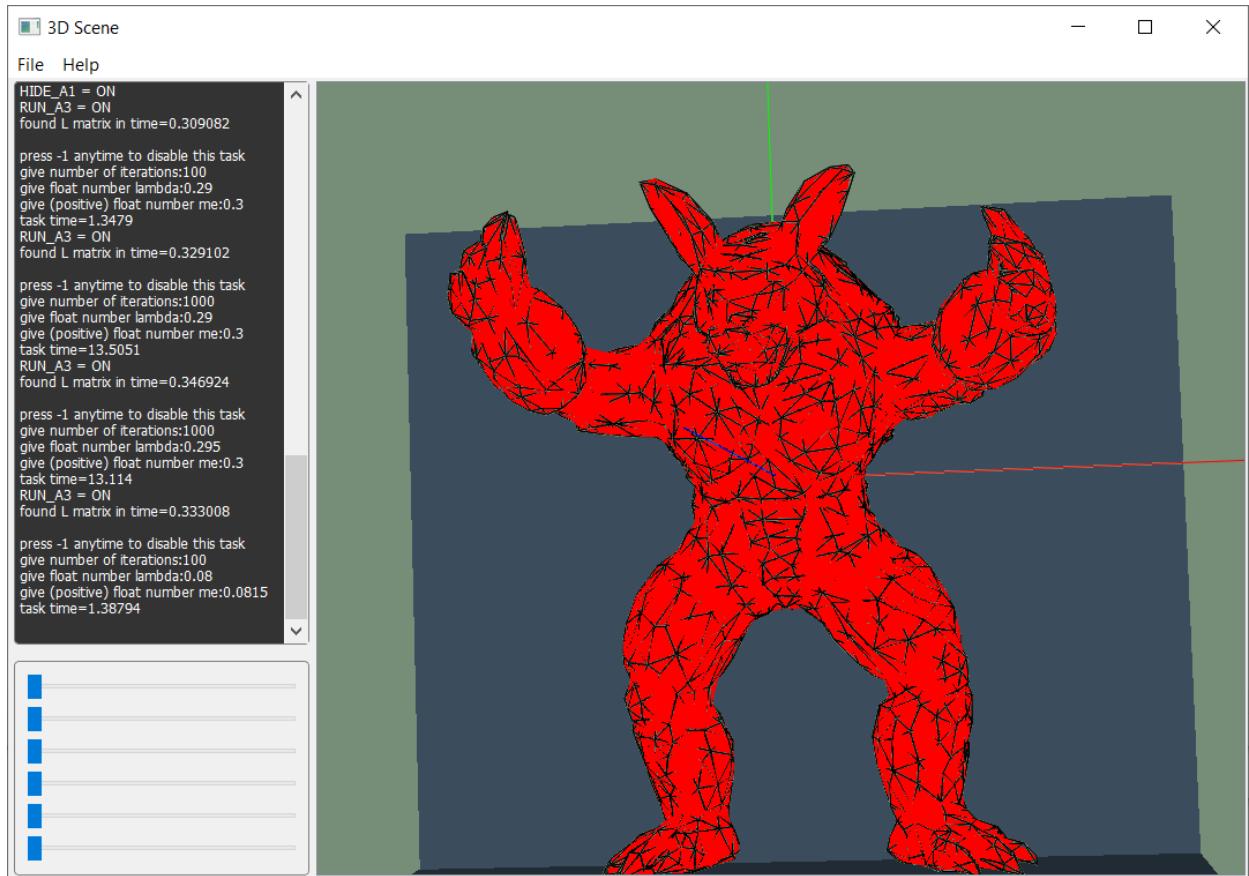


Εικόνα 10

Από τα παραπάνω φαίνεται ότι το αντικείμενο με την επεξεργασία αυτή ξεφεύγει λίγο από το μέγεθος του αρχικού και χάνεται περισσότερη λεπτομέρεια από το επιθυμητό. Άρα, πρέπει να χρησιμοποιήσω μικρότερες τιμές  $\lambda$ ,  $\mu$  και πιο κοντά μεταξύ τους.

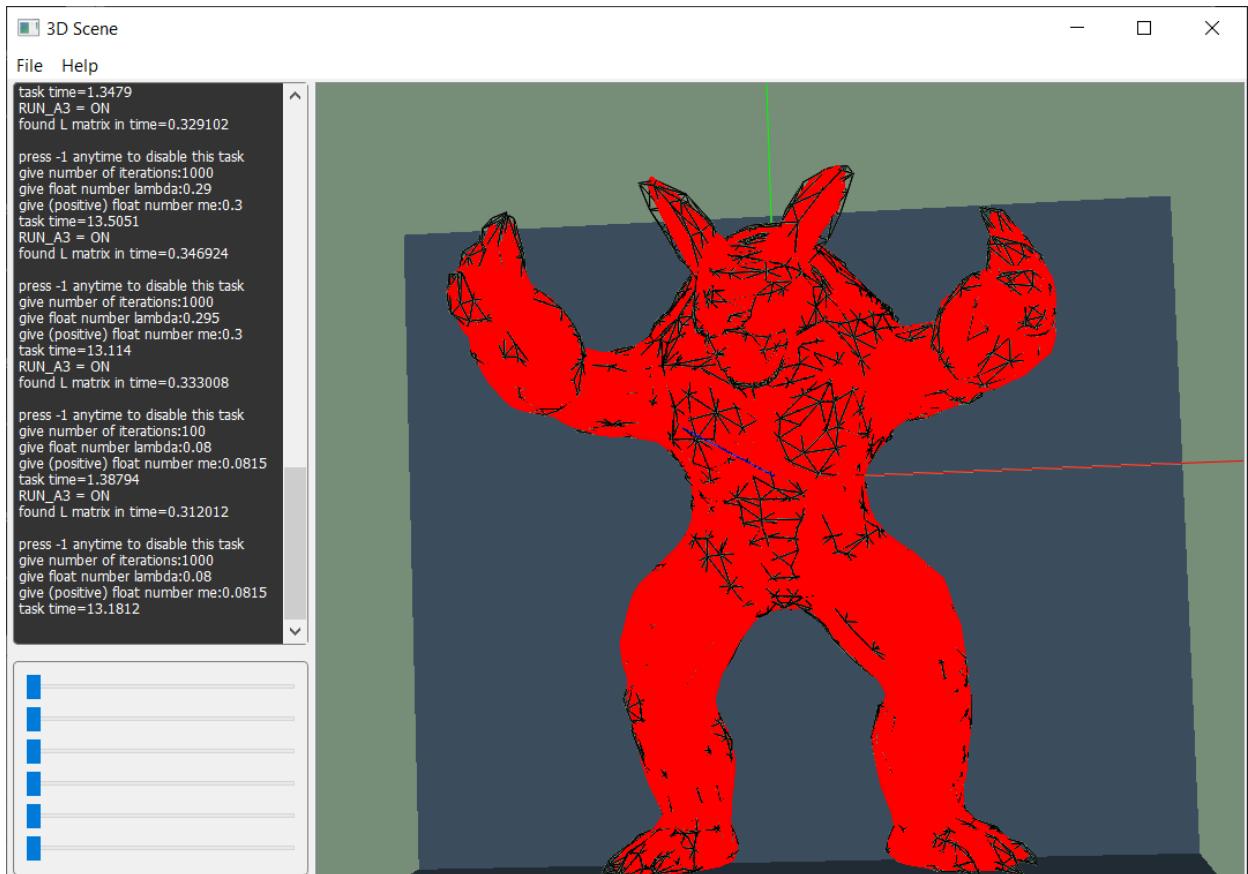
Επιλέγω  $\lambda=0.08$  και  $\mu=0.0815$ :

→ για 100 επαναλήψεις:



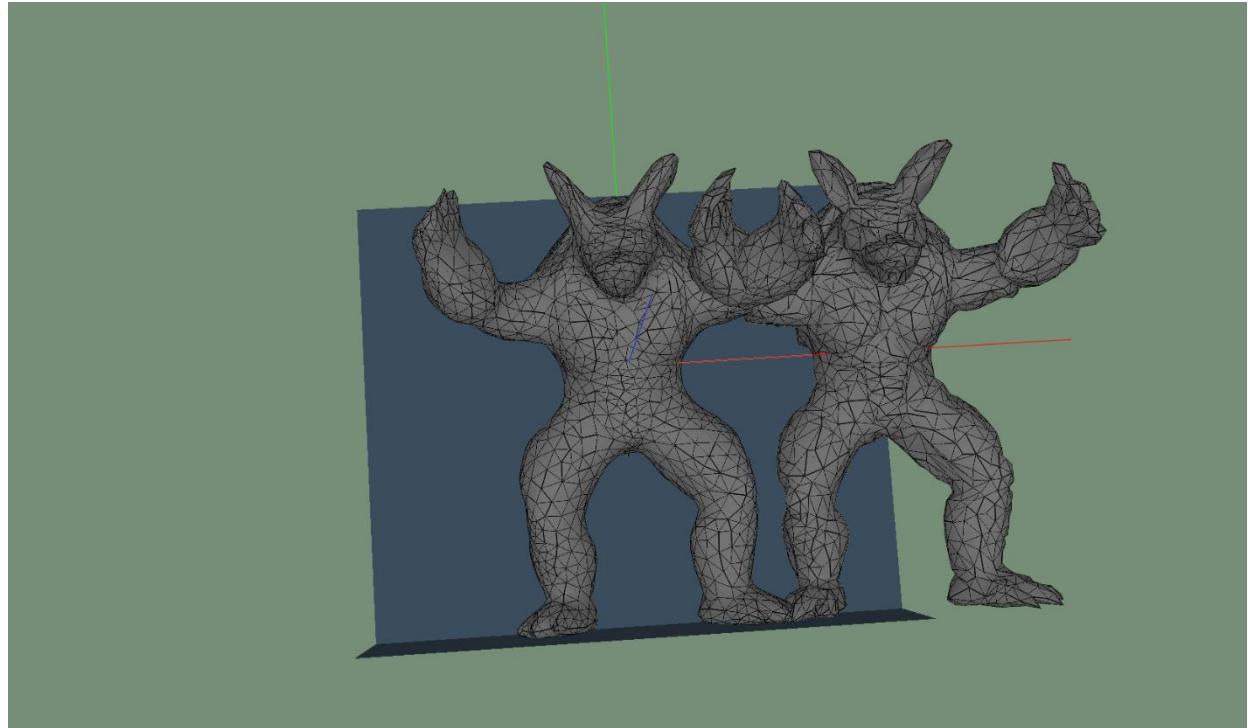
Εικόνα 11

→ για 1000 επαναλήψεις:

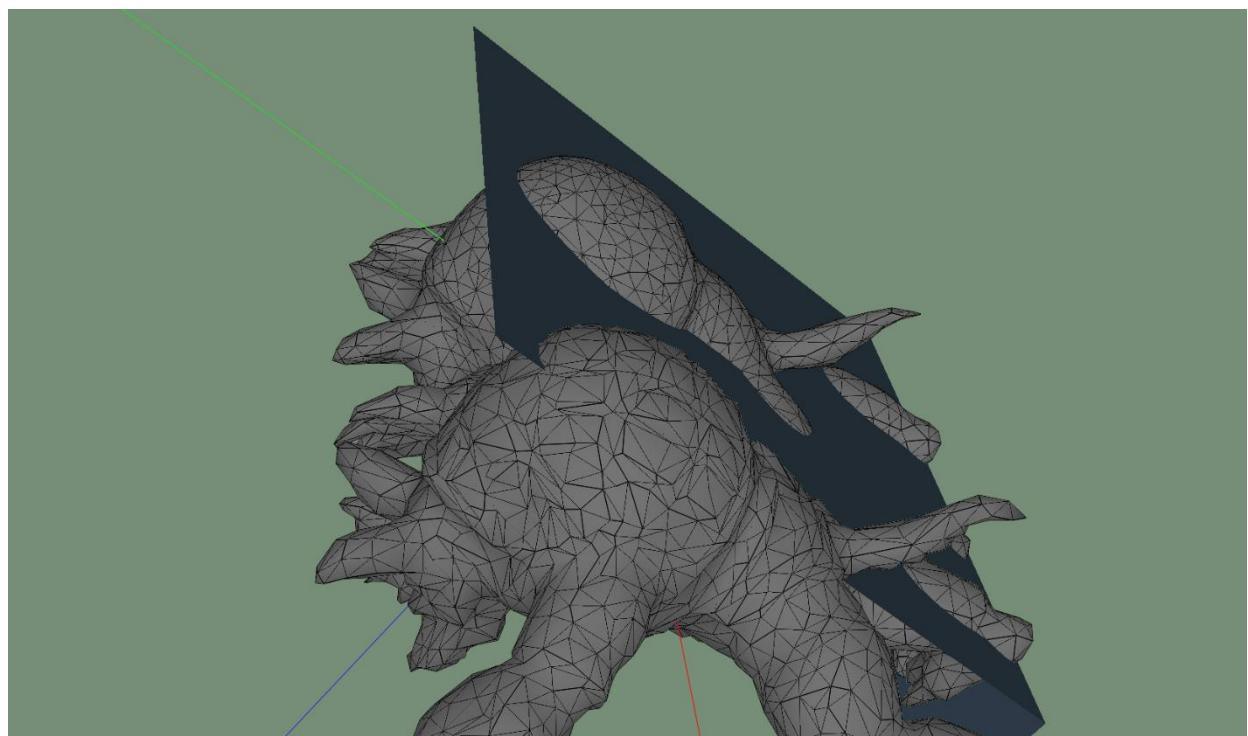


Εικόνα 12

Οι παραπάνω δοκιμές είναι αρκετά κοντά στο επιθυμητό αποτέλεσμα εξομάλυνσης του αρχικού αντικειμένου και αυτό φαίνεται απεικονίζοντας το αρχικό και το αποτέλεσμα δίπλα δίπλα:



Εικόνα 13



Εικόνα 14

Παρατηρώ στο πίσω μέρος του αντικειμένου ότι έχει πετύχει η εξομάλυνση χωρίς να συρρικνωθεί ή να μεγεθυνθεί η πλάτη του armadillo. Επίσης, τα άκρα του δεν έχουν χάσει πολύ λεπτομέρεια.

**B\_1)** Για να βρω το πεδίο/συνάρτηση της προσημασμένης απόστασης από ένα πλέγμα αρχικά χρησιμοποίησα την συνάρτηση *findSigned\_Brute* που υπολογίζει την βέλτιστη απόσταση του κάθε σημείου έναρξης από το κάθε σημείο του επιλεγμένου πλέγματος με έναν διπλό βρόχο. Μετά, για τον καθορισμό του πρόσημου της τιμής της απόστασης χρησιμοποιώ την συνάρτηση *meshContainsPoint* που ελέγχει αν το σημείο βρίσκεται εντός του πλέγματος, οπότε έχω θετικό πρόσημο και αλλιώς αρνητικό πρόσημο. Η συνάρτηση αυτή χρησιμοποιεί την μέθοδο ray tracing, όπου από ένα εξωτερικό του πλέγματος σημείο φέρνω μία ακτίνα στο σημείο για το οποίο γίνεται ο έλεγχος. Μετά, βρίσκω με την *lineIntersection* τα σημεία τομής της ακτίνας με το κάθε τρίγωνο του πλέγματος και αν ο αριθμός σημείων τομής είναι περιττός, τότε η ακτίνα εισέρχεται (σε κάθε περιττό σημείο τομής) στο εσωτερικό του πλέγματος χωρίς να εξέρχεται (σε κάθε άρτιο σημείο τομής). Η *lineIntersection* υπολογίζει τα σημεία τομής τριγώνου με γραμμή στις 3 διαστάσεις με τις παρακάτω εξισώσεις :

Για το τρίγωνο  $tr = (P_1, P_2, P_3)$  βρίσκω το normal του:

$$\overrightarrow{P_{12}} = P_2 - P_1 , \quad \overrightarrow{P_{13}} = P_3 - P_1 \Rightarrow \vec{n} = \overrightarrow{P_{12}} \times \overrightarrow{P_{13}}$$

Για την ακτίνα που διέρχεται από τα σημεία  $pl_1, pl_2$ , όπου  $pl1$  ένα εξωτερικό σημείο και η αρχή της ακτίνας και  $pl2$  το εξεταζόμενο σημείο:

$$\overrightarrow{u} = pl_2 - pl_1$$

Το σημείο τομής τους ανήκει στην ακτίνα οπότε:

$$P_i = pl_1 + t * \vec{u} \tag{1}$$

Και ανήκει στο τρίγωνο  $tr$  οπότε:

$$P_i = P_1 + u * \overrightarrow{P_{12}} + v * \overrightarrow{P_{13}} \tag{2}$$

Εξισώνοντας τις (1) και (2):

$$\begin{aligned} pl_1 + t * \vec{u} &= P_1 + u * \overrightarrow{P_{12}} + v * \overrightarrow{P_{13}} \Rightarrow \\ \Rightarrow u * \overrightarrow{P_{12}} + v * \overrightarrow{P_{13}} - t * \vec{u} &= pl_1 - P_1 \end{aligned}$$

Γράφοντας την παραπάνω με μορφή πινάκων :

$$\begin{bmatrix} \overrightarrow{P_{12}} \cdot x & \overrightarrow{P_{13}} \cdot x & -\vec{u} \cdot x \\ \overrightarrow{P_{12}} \cdot y & \overrightarrow{P_{13}} \cdot y & -\vec{u} \cdot y \\ \overrightarrow{P_{12}} \cdot z & \overrightarrow{P_{13}} \cdot z & -\vec{u} \cdot z \end{bmatrix} * \begin{bmatrix} u \\ v \\ t \end{bmatrix} = \begin{bmatrix} Pl_1 \cdot x - P_1 \cdot x \\ Pl_1 \cdot y - P_1 \cdot y \\ Pl_1 \cdot z - P_1 \cdot z \end{bmatrix}$$

Θεωρώ την ορίζουσα :  $D = \det \begin{pmatrix} \overrightarrow{P_{12}} \cdot x & \overrightarrow{P_{13}} \cdot x & -\vec{u} \cdot x \\ \overrightarrow{P_{12}} \cdot y & \overrightarrow{P_{13}} \cdot y & -\vec{u} \cdot y \\ \overrightarrow{P_{12}} \cdot z & \overrightarrow{P_{13}} \cdot z & -\vec{u} \cdot z \end{pmatrix}$

Με Cramer προκύπτει η λύση:

$$u = \frac{\det \begin{pmatrix} Pl_1 \cdot x - P_1 \cdot x & \overrightarrow{P_{13}} \cdot x & -\vec{u} \cdot x \\ Pl_1 \cdot y - P_1 \cdot y & \overrightarrow{P_{13}} \cdot y & -\vec{u} \cdot y \\ Pl_1 \cdot z - P_1 \cdot z & \overrightarrow{P_{13}} \cdot z & -\vec{u} \cdot z \end{pmatrix}}{D}$$

$$v = \frac{\det \begin{pmatrix} \overrightarrow{P_{12}} \cdot x & Pl_1 \cdot x - P_1 \cdot x & -\vec{u} \cdot x \\ \overrightarrow{P_{12}} \cdot y & Pl_1 \cdot y - P_1 \cdot y & -\vec{u} \cdot y \\ \overrightarrow{P_{12}} \cdot z & Pl_1 \cdot z - P_1 \cdot z & -\vec{u} \cdot z \end{pmatrix}}{D}$$

$$t = \frac{\det \begin{pmatrix} \overrightarrow{P_{12}} \cdot x & \overrightarrow{P_{13}} \cdot x & Pl_1 \cdot x - P_1 \cdot x \\ \overrightarrow{P_{12}} \cdot y & \overrightarrow{P_{13}} \cdot y & Pl_1 \cdot y - P_1 \cdot y \\ \overrightarrow{P_{12}} \cdot z & \overrightarrow{P_{13}} \cdot z & Pl_1 \cdot z - P_1 \cdot z \end{pmatrix}}{D}$$

Για να υπάρχει το σημείο τομής  $P_i$  πρέπει να ισχύουν τα εξής :

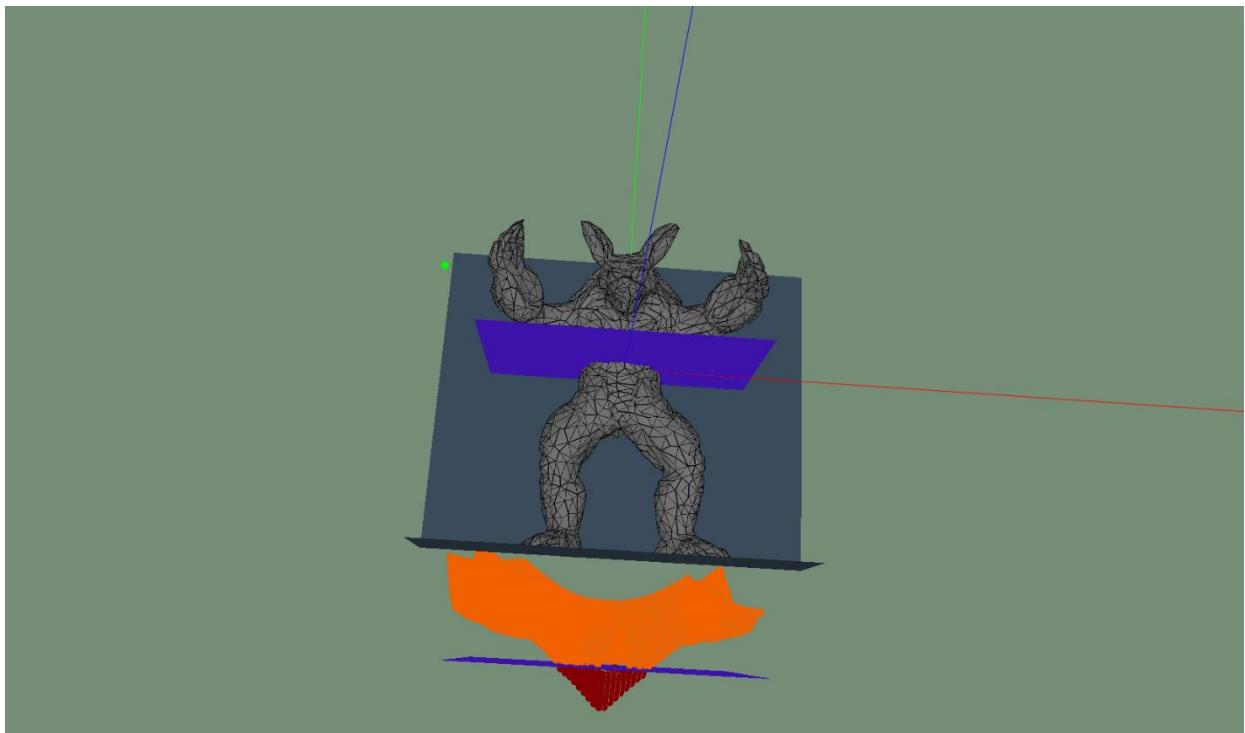
$$u \geq 0, v \geq 0, u + v \leq 1 \text{ και } -(\vec{n} * \vec{u}) > 0$$

Αν υπάρχει το σημείο τομής , τότε το υπολογίζω από την εξίσωση ευθείας για την παράμετρο  $t$  που υπολόγισα προηγουμένως:

$$P_i = Pl_1 + t * \vec{u}$$

Οπότε αν βρω περιττό αριθμό σημείων τομής για την ακτίνα του επιλεγμένου σημείου με τα τρίγωνα του πλέγματος , τότε βρίσκεται εντός του πλέγματος , αλλιώς είναι εκτός. Τα αποτελέσματα για την συνάρτηση signed distance τα αποθηκεύω σε έναν πίνακα με στοιχεία δομής *signed\_distance* , η οποία έχει την τιμή της συνάρτησης , το index του σημείου που έχει την μικρότερη απόσταση από το σημείο ελέγχου και το σημείο ελέγχου.

Από την εκτέλεση του παραπάνω για την απεικόνιση του πεδίου προσημασμένης απόστασης των σημείων ενός επιπέδου στο  $y=0$  με τα σημεία ενός πλέγματος με το flag *RUN\_B1\_PLANE* προκύπτει:



*Εικόνα 15*

Όπου οι αρνητικές αποστάσεις απεικονίζονται με κόκκινο και το δεύτερο επίπεδο στο  $y=-20$  δείχνει τους άξονες για μηδενική τιμή της απόστασης και η κάθε τιμή της απόστασης απεικονίζεται κάτω από το αντίστοιχο σημείο του επιπέδου.

Σε δεύτερο στάδιο χρησιμοποίησα δομή kd-tree που κατασκευάζεται για να περιέχει τα σημεία του πλέγματος και επιτρέπει την ταχύτερη εύρεση της βέλτιστης απόστασης με την μέθοδο του κοντινότερου γείτονα του κάθε σημείου ελέγχου. Για μεγάλο αριθμό σημείων αρχίζει και γίνεται αισθητή η διαφορά και φαίνεται από τα παρακάτω αποτελέσματα εκτέλεσης των συναρτήσεων:

*-armadillo\_low\_low.obj*

*Average time for brute force function = 2.659 sec*

*Average time for kd-tree function = 2.61 sec*

**B\_2)**Για να βρω το διανυσματικό πεδίο των normals ενός πλέγματος μόνο για αρνητικές τιμές του sdf , δηλαδή μόνο για σημεία εντός του πλέγματος , έφτιαξα την συνάρτηση *find\_Normals*. Για κάθε σημείο  $p=p3$  με αρνητική sdf ελέγχω για κάθε τρίγωνο του πλέγματος αν το σημείο είναι ένα από αυτά του τριγώνου. Στην περίπτωση που ισχύει αυτό , τότε από το  $p$  και τα άλλα 2 σημεία του τριγώνου  $p1,p2$  χρησιμοποιώ τις παρακάτω σχέσεις για να βρω το τμήμα  $j$  και το αντίστοιχο βάρος  $j$  του normal του για το κάθε τρίγωνο που ανήκει:

$$\overrightarrow{P_{12}} = P_2 - P_1 \quad , \quad \overrightarrow{P_{13}} = P_3 - P_1 \Rightarrow \vec{n}_j = \overrightarrow{P_{12}} \times \overrightarrow{P_{13}}$$

$\vec{n}_j = \vec{n}_j / |\vec{n}_j|$  που είναι το normal του κάθε τριγώνου στο 1o ring του σημείου

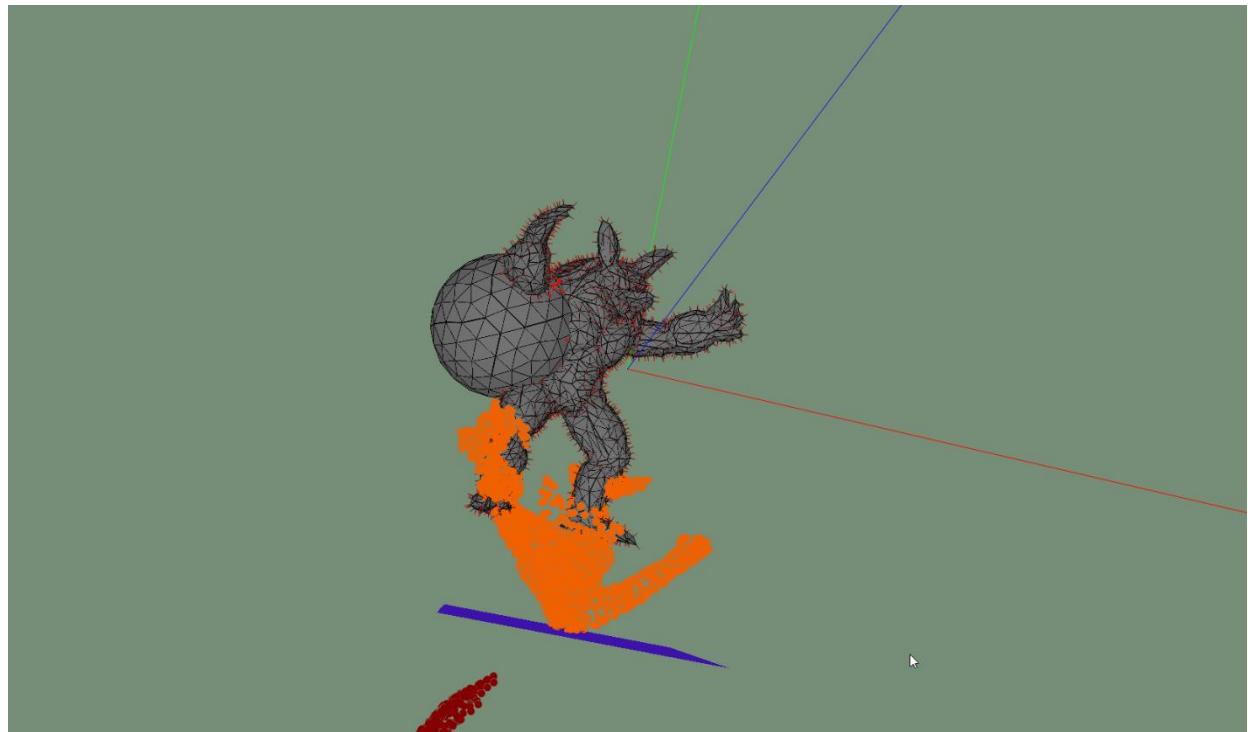
$\vec{P_{31}} = P_1 - P$  ,  $\vec{P_{32}} = P_2 - P \Rightarrow \theta = \widehat{\vec{P_{31}}, \vec{P_{32}}}$  η γωνία μεταξύ των πλευρών του κάθε τριγώνου που περιέχουν το σημείο

$w_j = \frac{\sin(\theta)}{|\vec{P_{31}}| * |\vec{P_{32}}|}$  το βάρος για το κάθε τρίγωνο με βάση την γωνία και το μήκος των πλευρών του (δηλαδή το εμβαδόν του κατά επέκταση)

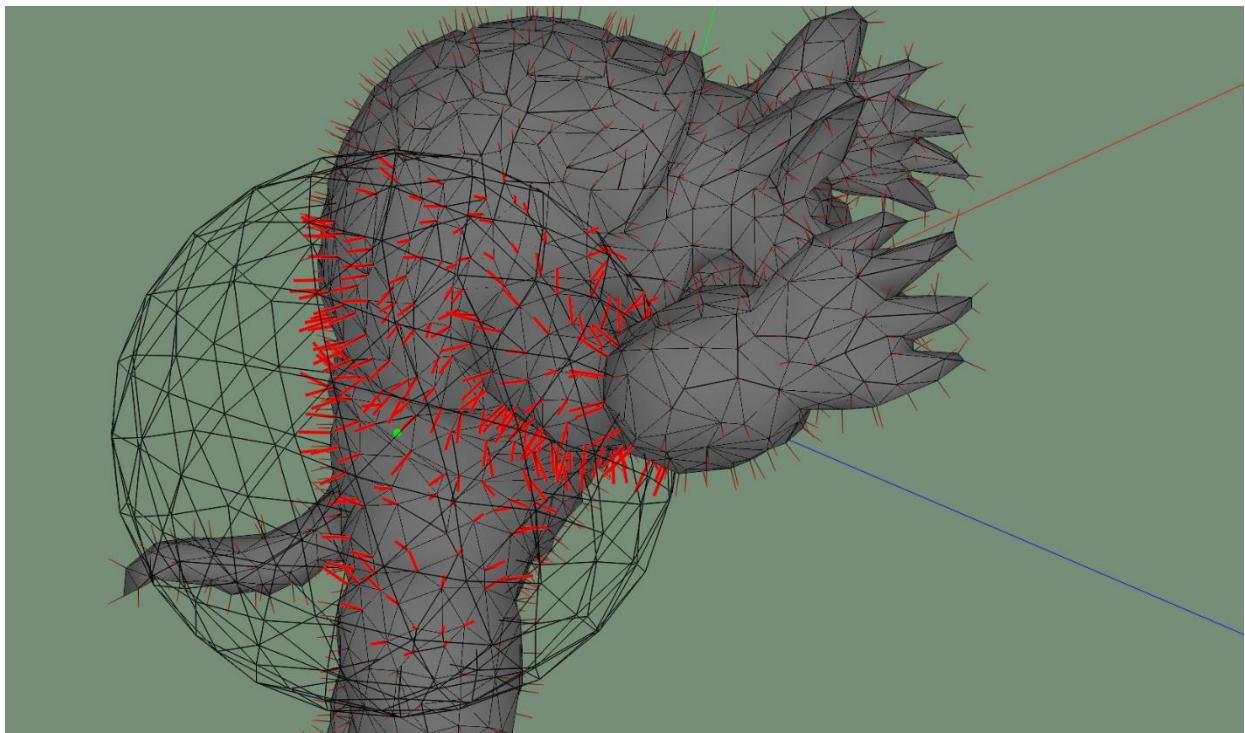
$\Rightarrow \vec{n} += w_j * \vec{n}_j, w += w_j$

Οπότε διαιρώντας στο τέλος με το άθροισμα των βαρών προκύπτει το normal:  $\vec{n}/= w$

Αν τα normal δείχνουν στο εσωτερικό του πλέγματος, τότε τα αντιστρέφω. Τελικά, τα αποθηκεύω με την δομή normal\_point και απεικονίζονται όπως φαίνεται παρακάτω με χρήση των flags *RUN\_B2*, *SHOW\_B1* για την sdf της απόστασης των σημείων του ενός πλέγματος από το άλλο και *SHOW\_B2* για τα normals του *m\_model* πλέγματος:



Εικόνα 16



Εικόνα 17

**B\_3)** Για να εκτελέσω εντοπισμό σύγκρουσης μεταξύ 2 πλεγμάτων πρέπει πρώτα να ορίσω την ταχύτητα και την γωνιακή ταχύτητα ίδιο-περιστροφής .

Για την ταχύτητα και γωνιακή ταχύτητα ενός πλέγματος χρησιμοποιώ τον  $3 \times 4$  ομογενή πίνακα μετασχηματισμού  $M$  που περιέχει στις 3 πρώτες στήλες 3 διανύσματα που καθορίζουν τους άξονες του αντικειμένου και στην 4<sup>η</sup> στήλη το σημείο αρχής του αντικειμένου που δηλώνει την θέση του σε σχέση με τους αρχικούς άξονες στο σημείο  $(0,0,0)$ . Ο πίνακας αυτός χρησιμοποιείται πολλαπλασιάζοντας τις 3 πρώτες στήλες του με το κάθε σημείο για να βρω τις συντεταγμένες του καινούριου μετασχηματισμένου σημείου και μετά προσθέτω στο αποτέλεσμα το σημείο αρχής από την 4<sup>η</sup> στήλη.

Λόγω της μορφής του πίνακα , η 4<sup>η</sup> στήλη καθορίζει το διάνυσμα μετατόπισης  $\Delta l = (\Delta x, \Delta y, \Delta z)$  του αντικειμένου για μία συγκεκριμένη χρονική στιγμή  $t$  και διάνυσμα ταχύτητας  $U$  από τον τύπο ευθύγραμμης κίνησης:

$$\vec{\Delta l} = \vec{U} \cdot t$$

Οπότε κατά την προσομοίωση της κίνησης του πλέγματος μπορώ να καθορίσω την μετατόπιση στον χώρο του αντικειμένου από την επιλεγμένη ταχύτητα και την τρέχουσα χρονική στιγμή εκτέλεσης της.

Για την γωνιακή ταχύτητα ίδιο-περιστροφής πρέπει να ορίσω τον πίνακα περιστροφής  $R$  , τον οποίο πολλαπλασιάζω με τον πίνακα μετασχηματισμού  $M$  (ο οποίος περιέχει τα διανύσματα για τους αρχικούς άξονες  $(1,0,0)$ ,  $(0,1,0)$  και  $(0,0,1)$ ) για να προκύψει ο  $M' = M^*R$  . Τον πίνακα  $M'$  τον εφαρμόζω στα σημεία του πλέγματος για να περιστραφεί το αντικείμενο κατά τις γωνίες  $\Delta thx$ ,  $\Delta thy$ ,  $\Delta thz$  που καθορίζονται από τις γωνιακές ταχύτητες

$\Omega_x$ ,  $\Omega_y$  και  $\Omega_z$  ως προς τον αντίστοιχο άξονα. Παρακάτω γίνεται ο υπολογισμός του πίνακα  $R$ :

$$R(\Delta thx, \Delta thy, \Delta thz) = R(\Delta thx) * R(\Delta thy) * R(\Delta thz), \text{ óπου:}$$

$$R(\Delta thx) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\Delta thx) & -\sin(\Delta thx) \\ 0 & \sin(\Delta thx) & \cos(\Delta thx) \end{bmatrix},$$

$$R(\Delta thy) = \begin{bmatrix} \cos(\Delta thy) & 0 & \sin(\Delta thy) \\ 0 & 1 & 0 \\ -\sin(\Delta thy) & 0 & \cos(\Delta thy) \end{bmatrix} \text{ και}$$

$$R(\Delta thz) = \begin{bmatrix} \cos(\Delta thz) & -\sin(\Delta thz) & 0 \\ \sin(\Delta thz) & \cos(\Delta thz) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Επίσης:  $\overrightarrow{\Delta th} = (\Delta thx, \Delta thy, \Delta thz) = \vec{\Omega} \cdot t$ , óπου  $\vec{\Omega} = (\Omega_x, \Omega_y, \Omega_z)$ .

Παρακάτω φαίνεται ένα παράδειγμα για την κίνηση ενός κύβου με ταχύτητα  $\overrightarrow{U1} = (\frac{4}{3}, 0, 0)$  και γωνιακή ταχύτητα  $\overrightarrow{\Omega} = (\frac{\pi}{8}, \frac{\pi}{8}, \frac{\pi}{8})$ :

[https://www.dropbox.com/s/x04affpf095ia7c/ConvexHull\\_IwkjN7AjGK.mp4?dl=0](https://www.dropbox.com/s/x04affpf095ia7c/ConvexHull_IwkjN7AjGK.mp4?dl=0)

Για τον καθορισμό της χρονικής στιγμής της σύγκρουσης 2 πλεγμάτων με πολλά σημεία ο υπολογισμός σε πραγματικό χρόνο είναι απαγορευτικά χρονοβόρος (αρκετά δευτερόλεπτα σε κάθε βήμα) και οδηγεί σε αποτέλεσμα όπου η προσομοίωση δεν προλαβαίνει να απεικονίσει το σωστό αποτέλεσμα της σύγκρουσης σε πραγματικό χρόνο. Άρα, επιλέγω να προσεγγίσω το σημείο σύγκρουσης εκ των προτέρων. Αυτό σημαίνει ότι συγκλίνω σε μία χρονική στιγμή, η οποία είναι όσο πιο κοντά γίνεται στην πραγματική και μετά ξεκινάω την προσομοίωση και την σταματάω μόλις φτάσει/ξεπεράσει την θεωρητική χρονική στιγμή που υπολόγισα.

Για την περίπτωση, όπου δεν έχω την *signed\_distance\_function* και εκτελώ την ανίχνευση σύγκρουσης μόνο με τα πλέγματα, χρησιμοποίησα έναν απλό αλγόριθμο που υπολογίζει επαναληπτικά τις καινούριες συντεταγμένες τους για σταθερό χρονικό βήμα και ελέγχει με την συνάρτηση *meshContainsPoint* εάν οποιοδήποτε σημείο του ενός πλέγματος βρίσκεται μέσα στο άλλο και στην περίπτωση που ισχύει αυτό σταματάει ο βρόχος και υπολογίζονται οι θεωρητικές τιμές για την χρονική στιγμή σύγκρουσης, το σημείο επαφής και το normal στο σημείο επαφής.

Για την περίπτωση, όπου έχω την *signed\_distance\_function*, χρησιμοποίησα έναν αλγόριθμο που υπολογίζει επαναληπτικά τις καινούριες συντεταγμένες τους για μεταβλητό χρονικό βήμα, που καθορίζεται από την ελάχιστη τιμή της sdf, το πρόσημό της και από την διαφορά μεταξύ των ταχυτήτων που έχουν τα 2 πλέγματα:

$$change = \frac{|min\_sdf|}{8 * |U1 - U2|}$$

Για την περίπτωση όπου περνάω το σημείο σύγκρουσης αλλάζει πρόσημο η ελάχιστη τιμή της sdf, είναι μεγαλύτερη η τιμή της από την επιθυμητή ακρίβεια και το μεταβλητό χρονικό βήμα έχει μεγαλύτερη τιμή από την επιθυμητή χρονική ακρίβεια: Τότε, καθορίζοντας ένα κάτω και ένα άνω όριο για την εκτιμώμενη στιγμή σύγκρουσης (από τον χρόνο που βρήκα στο προηγούμενο και προπροηγούμενο βήμα αντίστοιχα) βρίσκω το χρονικό βήμα από το μέσο του διαστήματος που ορίζουν τα 2 αυτά όρια:

$$change = \frac{(u_{bound} - l_{bound})}{2}$$

Μετά τον καθορισμό των ορίων, για κάθε αλλαγή ελέγχω αν το χρονικό βήμα φέρνει τον χρόνο εκτός του διαστήματος που ορίζουν και στην περίπτωση αυτή διαιρώ το βήμα δια 2.

Αν μία από τις παραμέτρους ακρίβειας που έχω ορίσει για την ελάχιστη sdf και το ελάχιστο χρονικό βήμα ξεπεραστεί τότε σταματάει ο βρόχος και υπολογίζω τις θεωρητικές τιμές για την χρονική στιγμή σύγκρουσης, το σημείο επαφής και το normal στο σημείο επαφής.

Για τις παρακάτω συγκρούσεις το επιλεγμένο αντικείμενο είναι το «armadillo\_low\_low.obj» που έχει 2002 σημεία και τρίγωνα και επιλέγω ταχύτητα  $\overrightarrow{U2} = (-\frac{4}{3}, 0, 0)$ , γωνιακή ταχύτητα  $\overrightarrow{\Omega2} = (0, \frac{\pi}{8}, 0)$  και αρχική θέση στο  $(0,0,0)$ .

-1<sup>η</sup> περίπτωση είναι η σύγκρουση σφαίρας με το επιλεγμένο αντικείμενο:

Η χρήση της σφαίρας που περιβάλλει το αντικείμενο για την ανίχνευση κρούσης έχει το πλεονέκτημα ότι δεν επηρεάζεται το αποτέλεσμα από τον προσανατολισμό των αντικειμένων και είναι αρκετά απλή, όμως αφήνει συνήθως αρκετό κενό χώρο, υποφέρει δηλαδή από ακρίβεια.

Για τον υπολογισμό της σφαίρας που περιβάλλει το αντικείμενο πήρα ως κέντρο το κέντρο μάζας του αντικειμένου και ως ακτίνα την μέγιστη απόσταση των σημείων του από το κέντρο μάζας. Για να ελέγχω αν ένα σημείο είναι εντός της σφαίρας αρκεί να συγκρίνω την απόσταση του σημείου αυτού από το κέντρο της με την ακτίνα της σφαίρας και με τον ίδιο τρόπο βρήκα την sdf της.

$$d = |\overrightarrow{PCentroid}| - radius$$

$d > 0 \Rightarrow$  έξω από την σφαίρα, αλλιώς μέσα

Α) Για την χρήση collision detection χωρίς sdf και σταθερό χρονικό βήμα = 1e-5:

Αποτελέσματα εκτέλεσης:

*mass1=0.0780372, mass2=251.729*

*TEMP\_SOLID = ON*

*SHOW\_AABB = ON*

*SHOW\_AABB = OFF*

*SHOW\_TEMP\_SPHERE = ON*

*RUN\_B3 = ON*

*choose mode to find collision: '1'=sphere, '2'=AABB, '3'=object*

*1*

*use sdf? (y/n)*

*n*

*running sphere with mesh collision detection without sdf*

*initial coll\_time=0*

*final coll\_time=4.51024*

*Found collision time in 4.112 seconds.*

*RUN\_B3 = OFF*

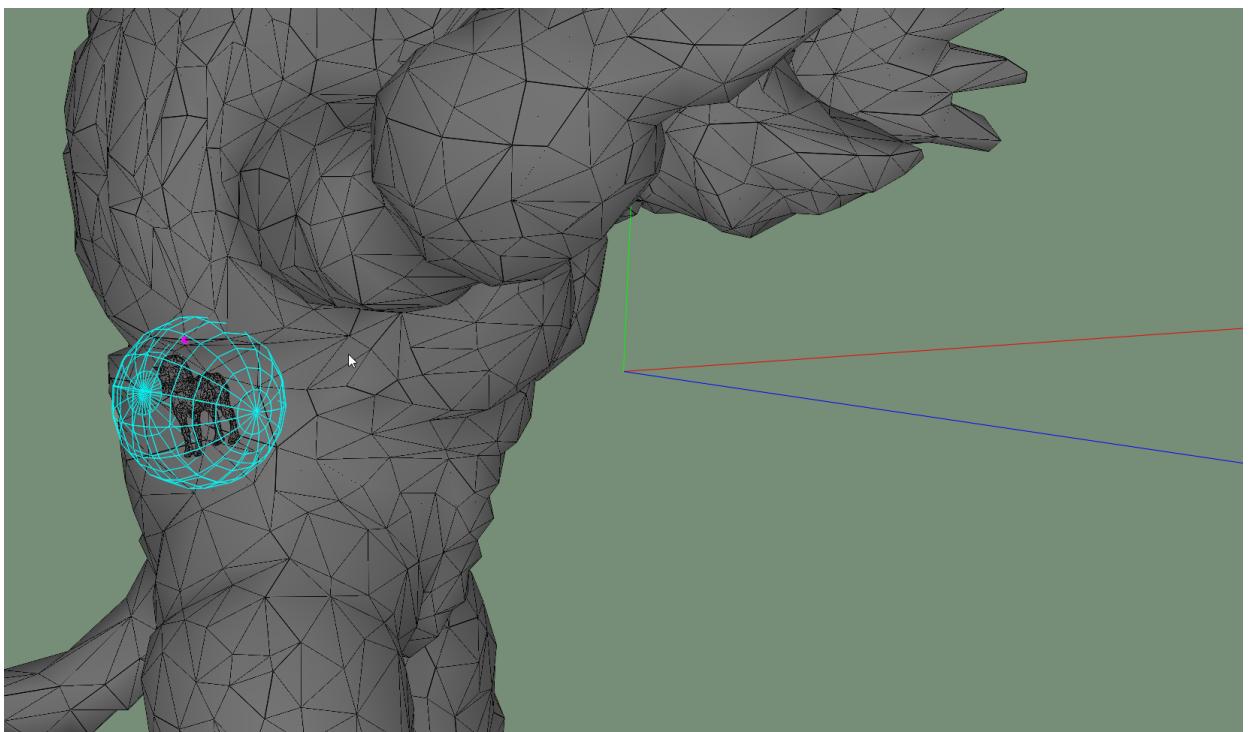
*MOVE\_B3 = ON*

*reached collision*

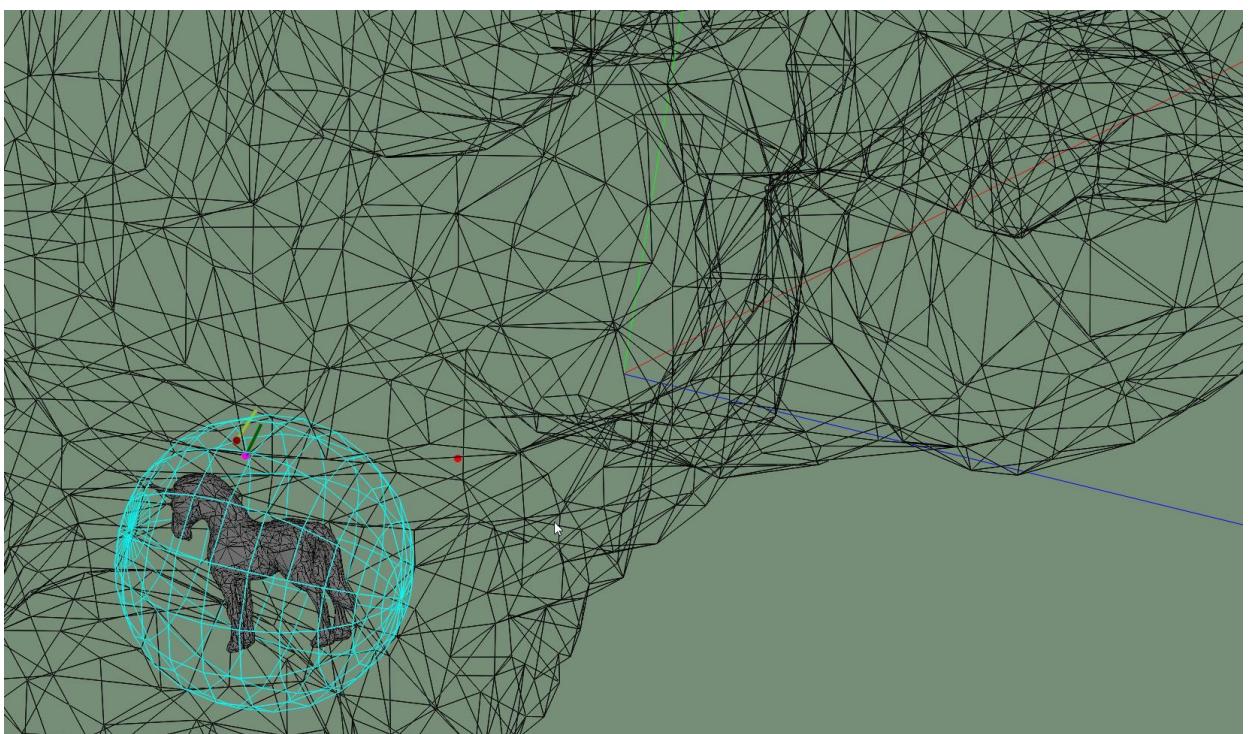
*time=4.901*

*theoretical time =4.51024*

Άρα, θα έχω καθυστέρηση προσομοίωσης = *time - theoretical time = 0.39 sec*, που οδηγεί σε μερική είσοδο της σφαίρας μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 18



Εικόνα 19

Β) Για την χρήση collision detection με sdf, ελάχιστη απόκλιση sdf από το  $0 = 0.001$  και ελάχιστο χρονικό βήμα  $= 1e-4$ :

*mass1=0.0780372, mass2=251.729*

*TEMP\_SOLID = ON*

*SHOW\_AABB = ON*

*SHOW\_AABB = OFF*

*SHOW\_TEMP\_SPHERE = ON*

*RUN\_B3 = ON*

*choose mode to find collision: '1'=sphere , '2'=AABB , '3'=object*

*1*

*use sdf? (y/n)*

*y*

*running sphere with mesh collision detection*

*initial coll\_time=2.6*

*min\_sdf\_final=0.0020026 , final\_change=9.38717e-05*

*theoretical contact index=1042 , x=-12.9719 , y=1.32017 , z=-0.672657*

*final coll\_time=4.48861*

*Found collision time in 0.825 seconds.*

*RUN\_B3 = OFF*

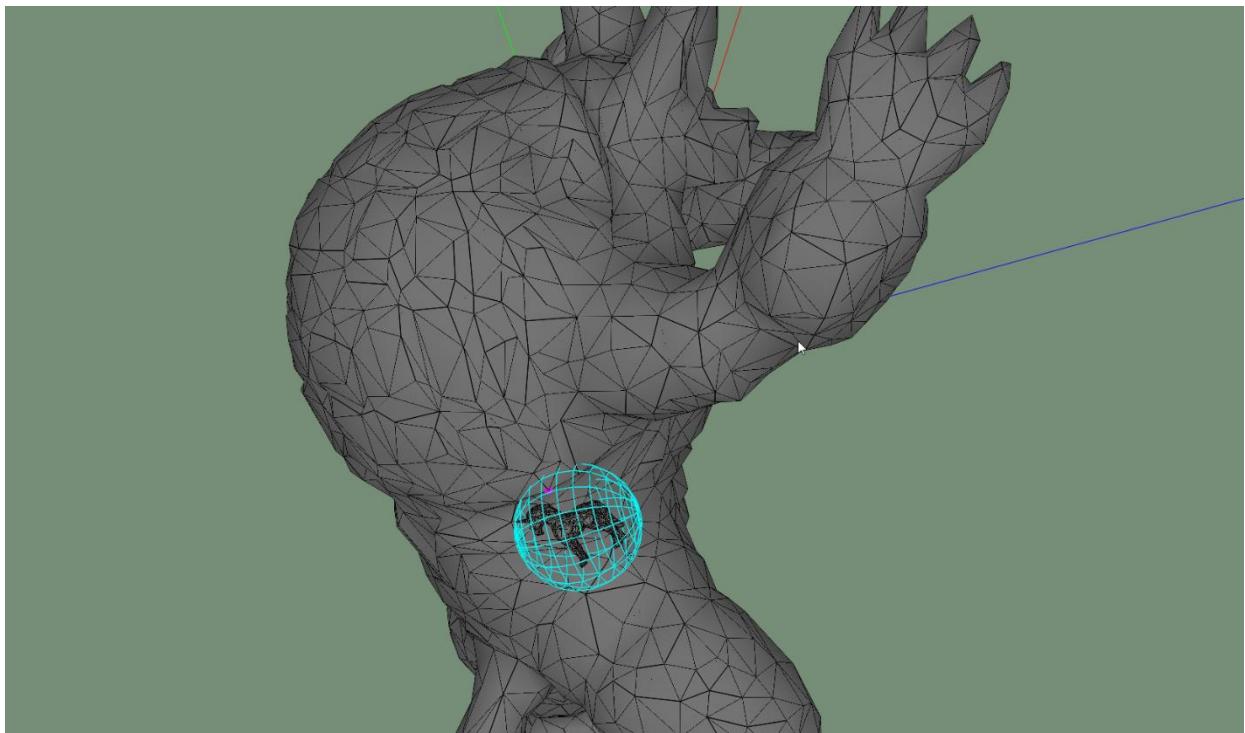
*MOVE\_B3 = ON*

*reached collision*

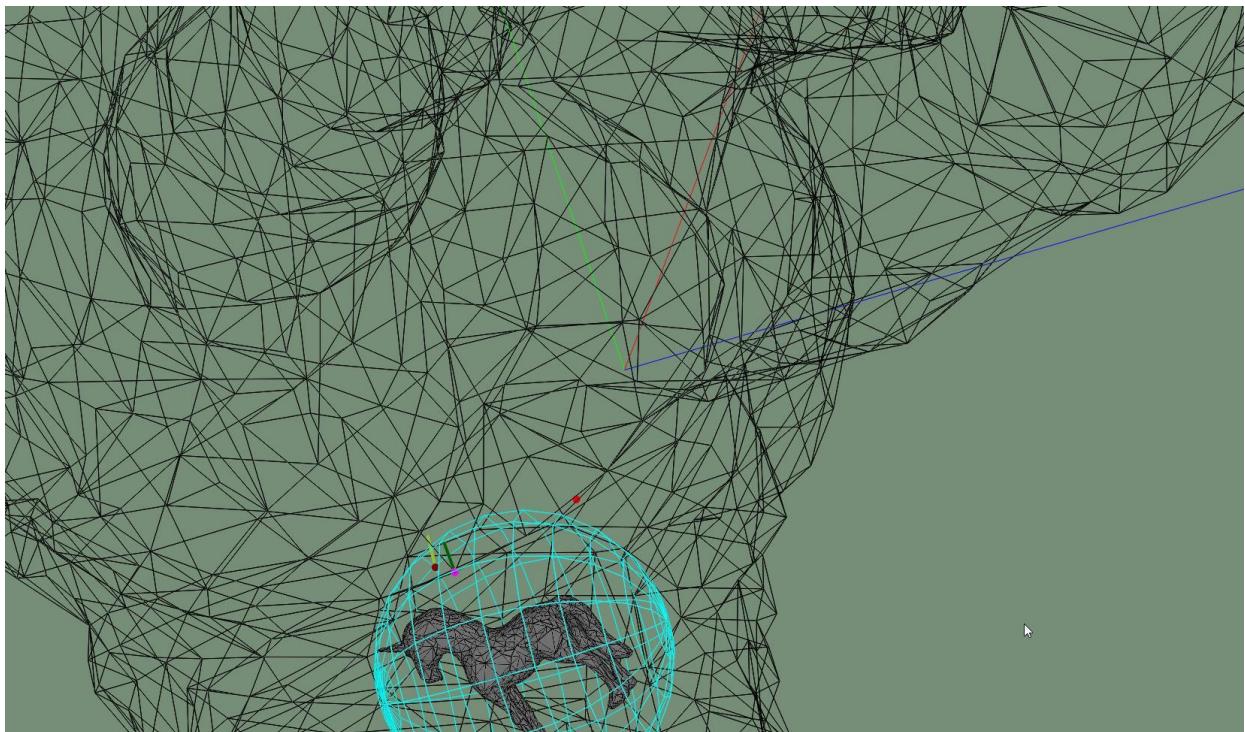
*time=4.846*

*theoretical time =4.48861*

Άρα , θα έχω καθυστέρηση προσομοίωσης = *time - theoretical time = 0.36 sec*, που οδηγεί σε μερική είσοδο της σφαίρας μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 20



Εικόνα 21

Παρατηρώ ότι πετυχαίνω λίγο καλύτερη ακρίβεια για πολύ μικρότερο χρόνο εκτέλεσης σε σύγκριση με τον αλγόριθμο χωρίς sdf.

-2<sup>η</sup> περίπτωση είναι η σύγκρουση του πλαισίου οριοθέτησης ευθυγραμμισμένου με τους άξονες(AABB) με το επιλεγμένο αντικείμενο:

To Axis Aligned Bounding Box ενός αντικειμένου αποτελεί μία από τις απλούστερες προσεγγίσεις που μπορούμε να κάνουμε για την σύγκρουση 2 αντικειμένων, έχει όμως το μειονέκτημα ότι δεν μας δίνει καλά αποτελέσματα για διαφορετικούς προσανατολισμούς των αντικειμένων κατά την κίνησή τους, ακόμα και όταν μεταβάλλεται δυναμικά το AABB με την ίδιο-περιστροφή και κίνηση του αντικειμένου. Το αντικείμενο για το οποίο χρησιμοποιώ το AABB έχει στα παρακάτω παραδείγματα εκτέλεσης ταχύτητα  $\overrightarrow{U1} = (\frac{4}{3}, 0, 0)$ , γωνιακή ταχύτητα  $\overrightarrow{\Omega1} = (\frac{\pi}{8}, \frac{\pi}{8}, \frac{\pi}{8})$  και αρχική θέση στο  $(-20, 0, 0)$ .

Για την εύρεση του AABB από τα σημεία του αντικειμένου χρησιμοποίησα την συνάρτηση *findAABB* που βρίσκει τις μέγιστες και ελάχιστες τιμές για τα x,y,z από τα σημεία και τις χρησιμοποιεί για να ορίσει το αντίστοιχο κουτί που προσαρμόζεται πάνω στο αντικείμενο κατά την μετακίνησή του, καθώς αλλάζουν τα σημεία του.

Η ποιο σημαντική επιτάχυνση σε αυτήν την περίπτωση είναι ο ταχύτερος έλεγχος για το αν τα σημεία βρίσκονται εντός ή εκτός του AABB με την συνάρτηση *AABBContainsPoint*. Για να το πετύχω αυτό πρώτα πηγαίνω στην περίπτωση που έχω μοναδιαίας πλευράς κύβο με κέντρο στο  $(0,0,0)$ , μετασχηματίζοντας το σημείο που ελέγχω από global coordinates σε local coordinates και μειώνοντας την κλίμακα χρησιμοποιώντας το μήκος της κάθε πλευράς του AABB:

$$p_{local} = p_{global} - AABB_{Centre}$$

$$a_x = edge_x / 2 = |MaxX - MinX|$$

$$a_y = edge_y / 2 = |MaxY - MinY|$$

$$a_z = edge_z / 2 = |MaxZ - MinZ|$$

$$x_{scaled} = x = p_{local_x} / a_x$$

$$y_{scaled} = y = p_{local_y} / a_y$$

$$z_{scaled} = z = p_{local_z} / a_z$$

Αν  $|x| \leq 1$  &  $|y| \leq 1$  &  $|z| \leq 1$  τότε το σημείο  $(x,y,z)$  είναι μέσα στον κύβο με σημεία  $(+1, +1, +1)$ , οπότε και το σημείο  $p_{global}$  είναι μέσα στο AABB.

Άρα, το πρόβλημα μετατρέπεται σε  $O(N)$  από  $O(N^2)$  που ήταν πριν με την χρήση της συνάρτησης *meshContainsPoint*.

Ομοίως, για την εύρεση της sdf αντικειμένου από το AABB, αρκεί να κάνω την ίδια μετατροπή και αλλαγή κλίμακας για το σημείο και μετά το πρόβλημα ανάγεται στην εύρεση της ελάχιστης απόστασης σημείου από κύβο πλευράς 1 στην αρχή των αξόνων. Για την επίλυση του προβλήματος αυτού χωρίζω τον χώρο σε 26 περιοχές που ορίζονται προεκτείνοντας την κάθε πλευρά του κύβου και στις 2 κατευθύνσεις. Ανάλογα σε ποια περιοχή ανήκει το σημείο, τότε η απόσταση του ορίζεται ως εξής:

Av  $|x| \leq 1 \& |y| \leq 1 \& |z| > 1$  τότε η αντίστοιχη τετραγωνική επιφάνεια ορίζει την απόσταση :

$$d = |z| - 1$$

Av  $|x| \leq 1 \& |y| > 1 \& |z| > 1$  τότε η αντίστοιχη πλευρά του κύβου ορίζει την απόσταση:

$$d = \sqrt{(|y| - 1)^2 + (|z| - 1)^2}$$

Av  $|x| > 1 \& |y| > 1 \& |z| > 1$  τότε το αντίστοιχο σημείο του κύβου ορίζει την απόσταση:

$$d = \sqrt{(|x| - 1)^2 + (|y| - 1)^2 + (|z| - 1)^2}$$

Και ομοίως στις υπόλοιπες περιπτώσεις.

A)Για την χρήση collision detection χωρίς sdf και σταθερό χρονικό βήμα = 1e-5:

τα αποτελέσματα εκτέλεσης:

*mass1=0.0780372, mass2=251.729*

*SHOW\_TEMP\_AABB = ON*

*TEMP\_SOLID = ON*

*RUN\_B3 = ON*

*choose mode to find collision:'1'=sphere, '2'=AABB, '3'=object*

*2*

*use sdf? (y/n)*

*n*

*running aabb with mesh collision detection without sdf*

*initial coll\_time=0*

*final coll\_time=5.11165*

*Found collision time in 3.631 seconds.*

*RUN\_B3 = OFF*

*MOVE\_B3 = ON*

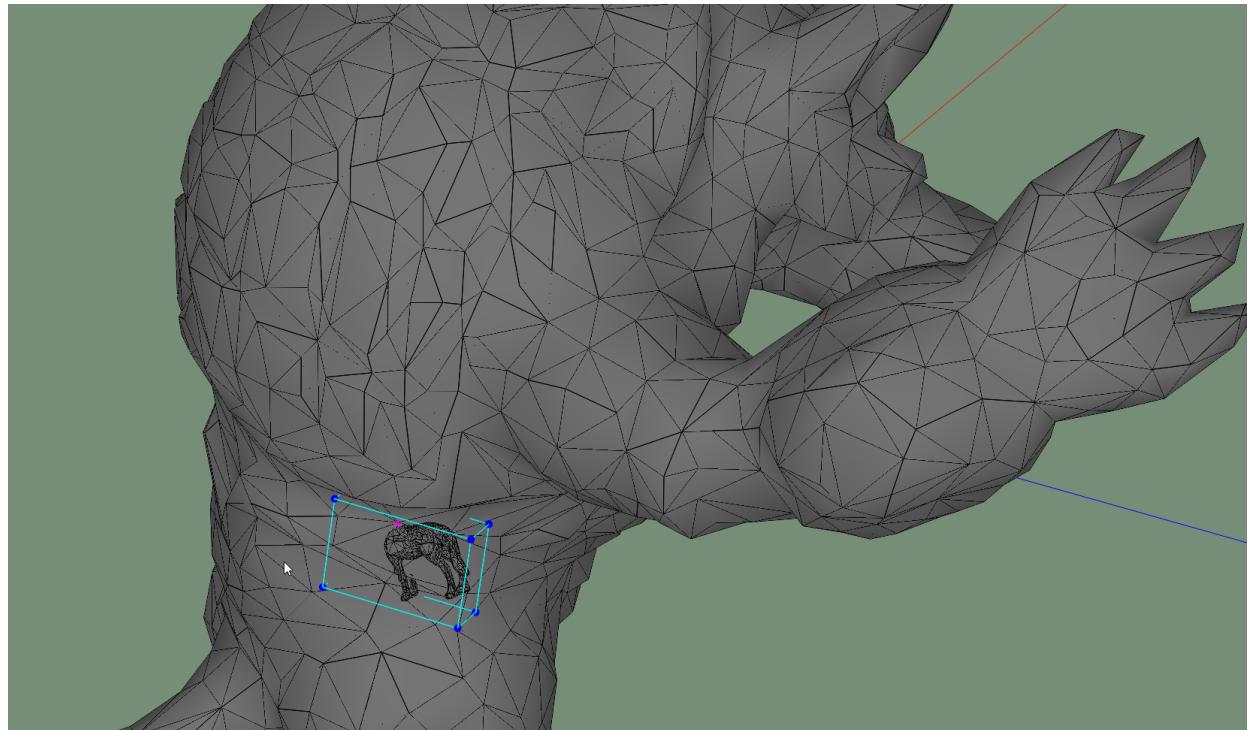
*SHOW\_SOLID = OFF*

*reached collision*

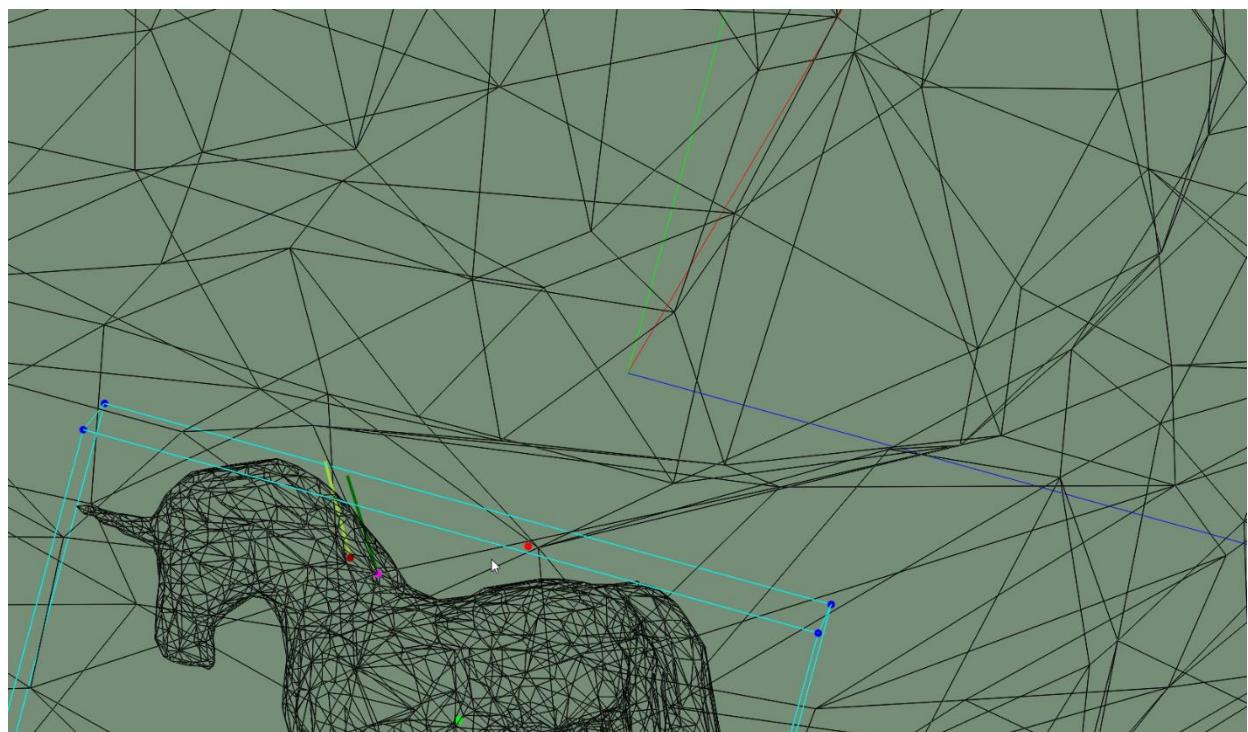
*time=5.26601*

*theoretical time =5.11165*

Άρα, θα έχω καθυστέρηση προσωμοίωσης = *time - theoretical time* = 0.154 sec, που οδηγεί σε μερική είσοδο του AABB μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 22



Εικόνα 23

Όπου το κόκκινο σημείο είναι το σημείο επαφής για θεωρητικό χρόνο και το αντίστοιχο normal διάνυσμα στο σημείο αυτό με κίτρινο και το μοβ σημείο είναι το σημείο επαφής για πραγματικό χρόνο εύρεσης της κρούσης με το αντίστοιχο normal με πράσινο. Παρατηρώ ότι η απόκλιση αυτή είναι εμφανής από την διαφορά μεταξύ των γωνιών των 2 normals και από το μέρος του κύβου που βρίσκεται μέσα στο αντικείμενο.

Για να αυξήσω την ακρίβεια εύρεσης του σημείου σύγκρουσης χρησιμοποιώ μικρότερο χρονικό βήμα = 1e-6:

*mass1=0.0780372, mass2=251.729*

*TEMP\_WIRE = ON*

*SHOW\_SOLID = OFF*

*RUN\_B3 = ON*

*choose mode to find collision:'1'=sphere, '2'=AABB, '3'=object*

*2*

*use sdf? (y/n)*

*n*

*running aabb with mesh collision detection without sdf*

*initial coll\_time=0*

*final coll\_time=5.08053*

*Found collision time in 35.291 seconds.*

*RUN\_B3 = OFF*

*SHOW\_TEMP\_AABB = ON*

*MOVE\_B3 = ON*

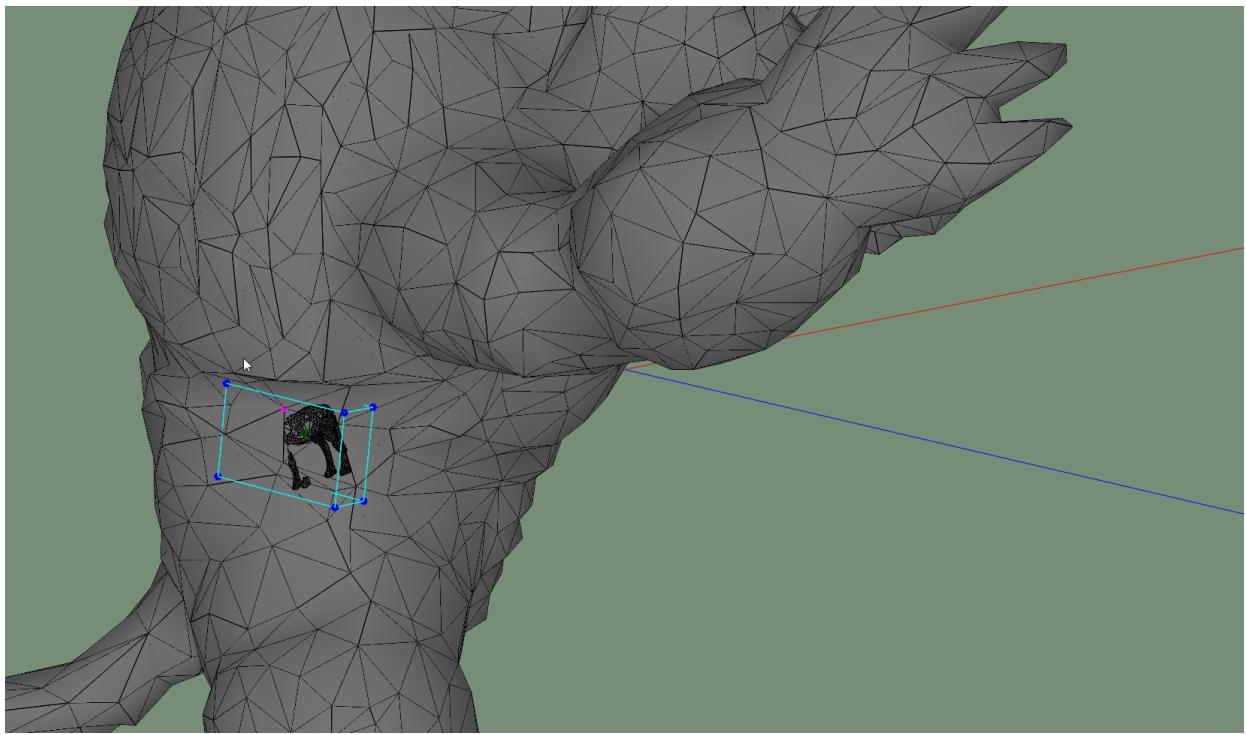
*SHOW\_SOLID = ON*

*reached collision*

*time=5.321*

*theoretical time =5.08053*

Άρα, θα έχω καθυστέρηση προσομοίωσης = *time - theoretical time = 0.24 sec*, που οδηγεί σε μερική είσοδο του κύβου μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 24

Παρατηρώ ότι , αυξάνεται η χρονική καθυστέρηση προσομοίωσης και αυξάνεται πολύ ο χρόνος εκτέλεσης του προγράμματος και δεν έχω κάποια διαφορά στην ακρίβεια της προσομοίωσης .

Β) Για την χρήση collision detection με sdf , ελάχιστη απόκλιση sdf από το  $0 = 0.001$  και ελάχιστο χρονικό βήμα  $= 0.0001$ :

*mass1=0.0780372, mass2=251.729*

*SHOW\_TEMP\_AABB = ON*

*TEMP\_SOLID = ON*

*RUN\_B3 = ON*

*choose mode to find collision:'1'=sphere , '2'=AABB , '3'=object*

*2*

*use sdf? (y/n)*

*y*

*running aabb with mesh collision detection*

*initial coll\_time=2.5*

*min\_sdf\_final=0.00212385 , final\_change=9.95555e-05*

*theoretical contact index=1439 , x=-12.8418 , y=0.524722 , z=-0.495295*

*final coll\_time=5.07679*

*Found collision time in 1.602 seconds.*

*RUN\_B3 = OFF*

*MOVE\_B3 = ON*

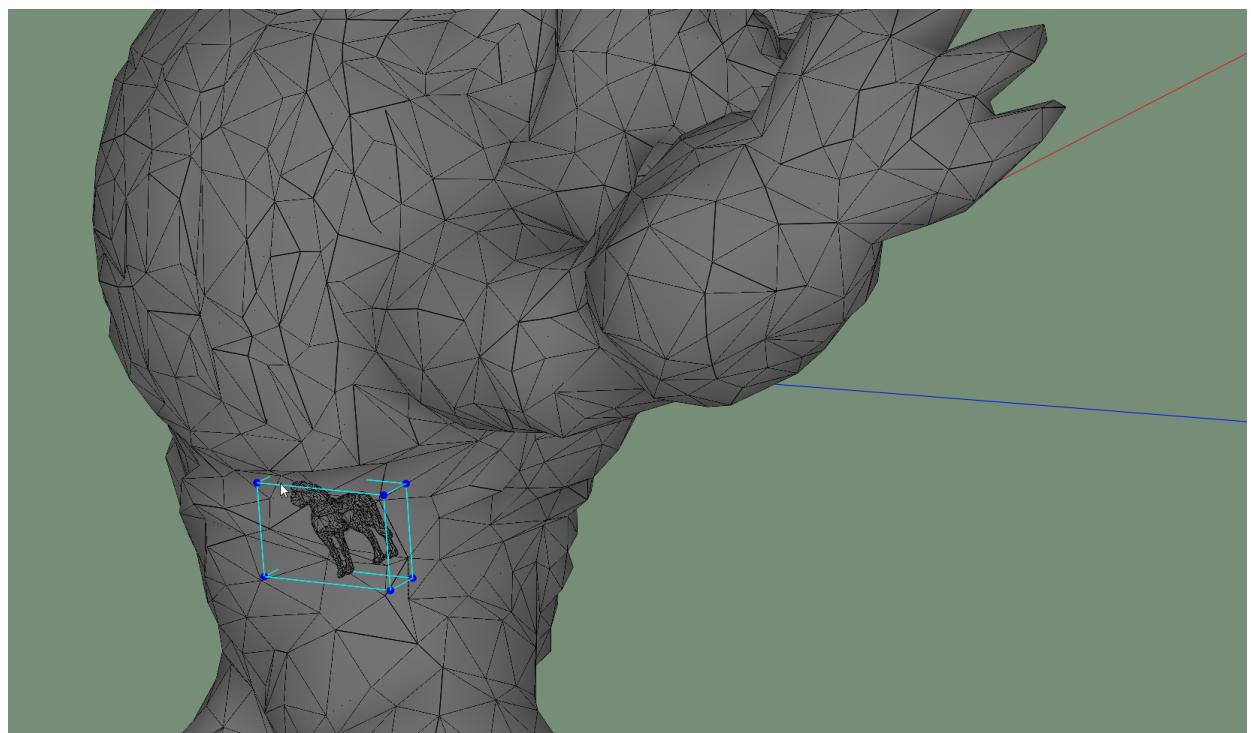
*SHOW\_SOLID = OFF*

*reached collision*

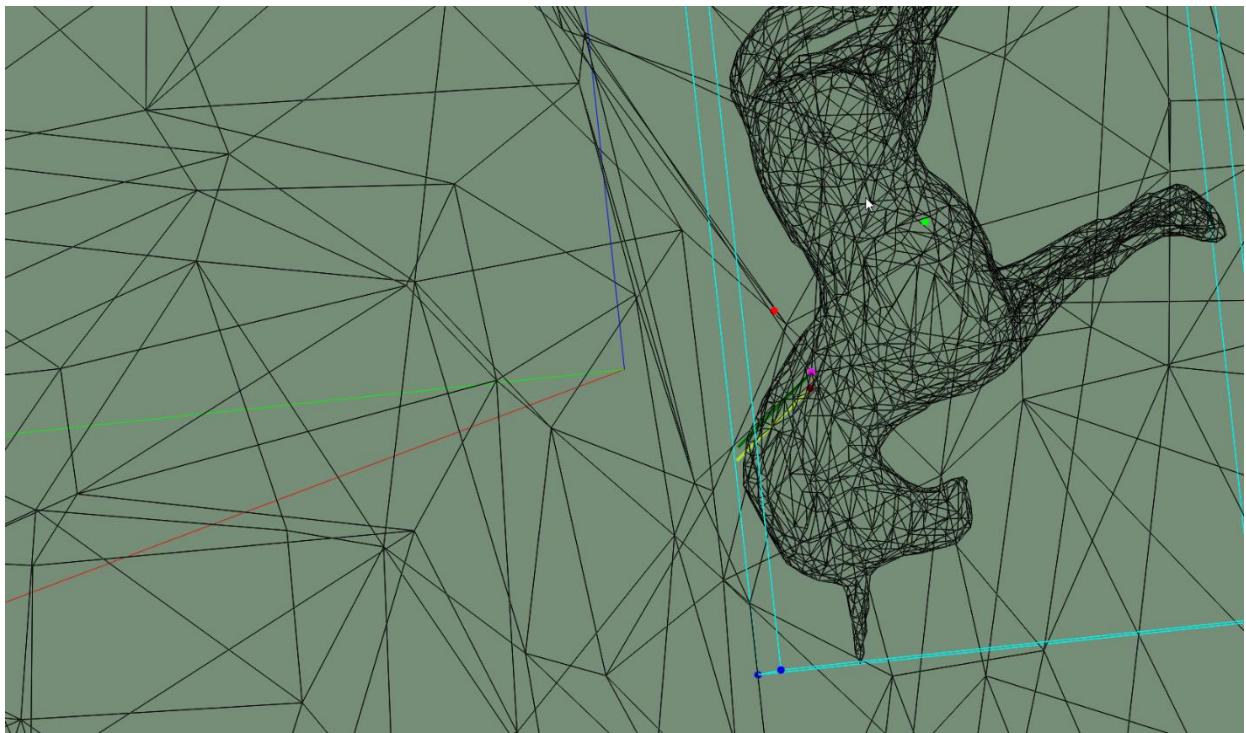
*time=5.156*

*theoretical time =5.07679*

Άρα, θα έχω καθυστέρηση προσομοίωσης = *time - theoretical time = 0.08 sec*, που οδηγεί σε μερική είσοδο του κουτιού μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στο παρακάτω σχήμα:



*Εικόνα 25*



*Εικόνα 26*

Παρατηρώ ότι με την χρήση της sdf το αποτέλεσμα είναι πολύ καλύτερο , αφού είναι πιο κοντά τα σημεία επαφής , το κουτί εισέρχεται κατά μικρό βαθμό στο αντικείμενο και ο χρόνος εκτέλεσης είναι πολύ μικρότερος για συγκρίσιμη ακρίβεια με τον προηγούμενο αλγόριθμο (1.6sec) .

-3<sup>η</sup> περίπτωση είναι η σύγκρουση απλού αντικειμένου με το επιλεγμένο αντικείμενο:

Η χρήση του απλού αντικειμένου βελτιώνει την ακρίβεια , αλλά αυξάνει ταυτόχρονα κατά πολύ τον χρόνο εκτέλεσης της ανίχνευσης σύγκρουσης.

A) Για την χρήση collision detection χωρίς sdf και σταθερό χρονικό βήμα = 0.0001:  
τα αποτελέσματα εκτέλεσης:

*mass1=1.41576, mass2=251.729*

*TEMP\_SOLID = ON*

*RUN\_B3\_NO\_SDF = ON*

*running collision detection without sdf*

*initial coll\_time=0*

*final coll\_time=5.00453*

*Found collision time in 84.614 seconds.*

*RUN\_B3\_NO\_SDF = OFF*

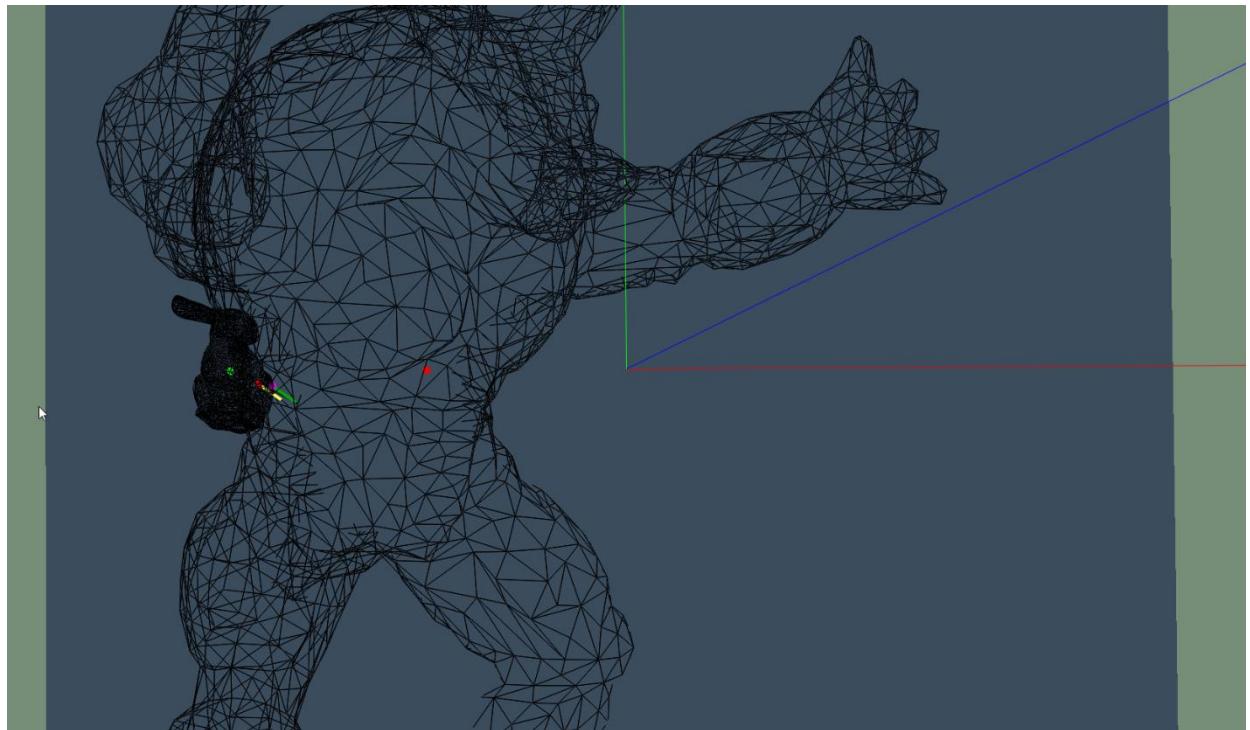
*MOVE\_B3 = ON*

*reached collision*

*time=5.055*

*theoretical time =5.00453*

Άρα, θα έχω καθυστέρηση προσομοίωσης = *time - theoretical time = 0.05 sec*, που οδηγεί σε μερική είσοδο του αντικειμένου μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στο παρακάτω σχήμα:



*Εικόνα 27*

Παρατηρώ ότι είναι αναγκαία και σε αυτήν την περίπτωση η περεταίρω μείωση του χρονικού βήματος στην τιμή  $1e-5$ :

*mass1=1.41576, mass2=251.729*

*TEMP\_SOLID = ON*

*RUN\_B3\_NO\_SDF = ON*

*running collision detection without sdf*

*initial coll\_time=0*

*final coll\_time=4.78669*

*Found collision time in 816.869 seconds.*

*RUN\_B3\_NO\_SDF = OFF*

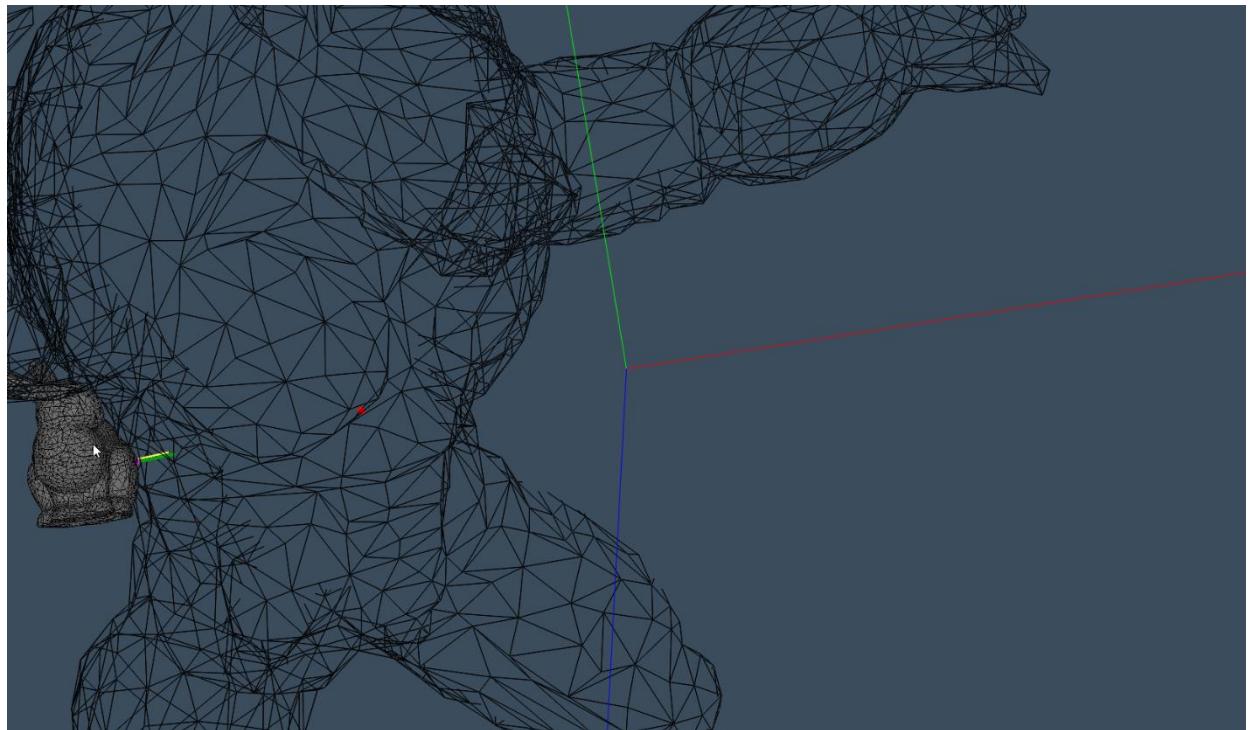
*MOVE\_B3 = ON*

*reached collision*

*time=4.86304*

*theoretical time =4.78669*

Άρα, θα έχω καθυστέρηση προσομοίωσης = time – theoretical time = 0.08 sec, που οδηγεί σε μερική είσοδο του αντικειμένου μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στο παρακάτω σχήμα:



*Εικόνα 28*

Παρατηρώ ότι σε αυτήν την περίπτωση ο χρόνος εκτέλεσης είναι πολύ μεγάλος για αυτό το χρονικό βήμα.

Β) Για την χρήση collision detection με sdf, ελάχιστη απόκλιση sdf από το 0 = 0.1 και ελάχιστο χρονικό βήμα = 0.001:

*mass1=1.41576, mass2=251.729*

*TEMP\_SOLID = ON*

*RUN\_B3 = ON*

*running collision detection*

*initial coll\_time=2.6*

*min\_sdf\_final=0.0908774, final\_change=0.00425988*

*final coll\_time=4.73182*

*Found collision time in 159.076 seconds.*

*RUN\_B3 = OFF*

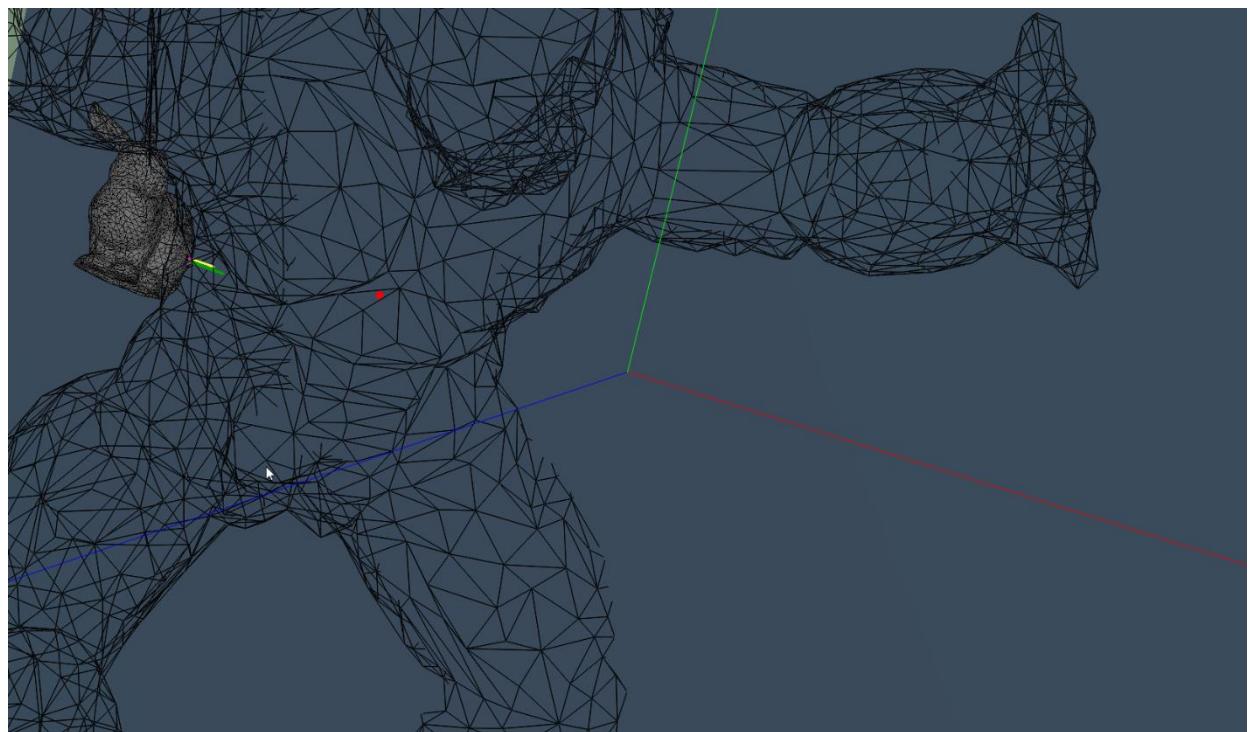
*MOVE\_B3 = ON*

*reached collision*

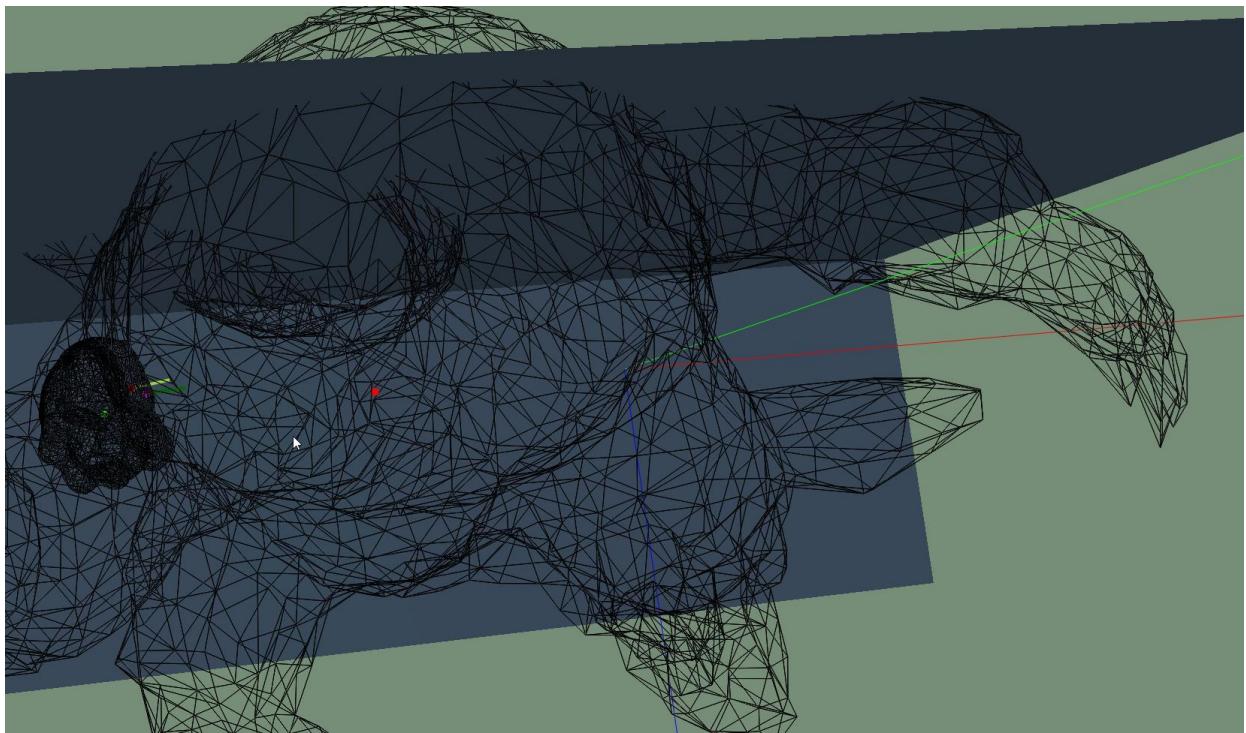
*time=4.97801*

*theoretical time =4.73182*

Άρα, θα έχω καθυστέρηση προσομοίωσης = *time - theoretical time* = 0.25 sec, που οδηγεί σε μερική είσοδο του αντικειμένου μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στα παρακάτω σχήματα:



*Εικόνα 29*



Εικόνα 30

Παρατηρώ ότι είναι απαραίτητη η αύξηση της ακρίβειας και ότι η εκτέλεση σταμάτησε λόγω της  $\text{min\_sdf\_final} = 0.09 < 0.1$ , άρα κρατάω το ίδιο όριο για το χρονικό βήμα 0.001 και μειώνω το όριο για την  $\text{min\_sdf}$  στο 0.01:

$\text{mass1}=1.41576, \text{mass2}=251.729$

$\text{TEMP\_SOLID} = \text{ON}$

$\text{RUN\_B3} = \text{ON}$

*running collision detection*

*initial coll\_time=2.6*

$\text{min\_sdf\_final}=0.0706481, \text{final\_change}=0.000419855$

*final coll\_time=4.76315*

*Found collision time in 212.291 seconds.*

$\text{RUN\_B3} = \text{OFF}$

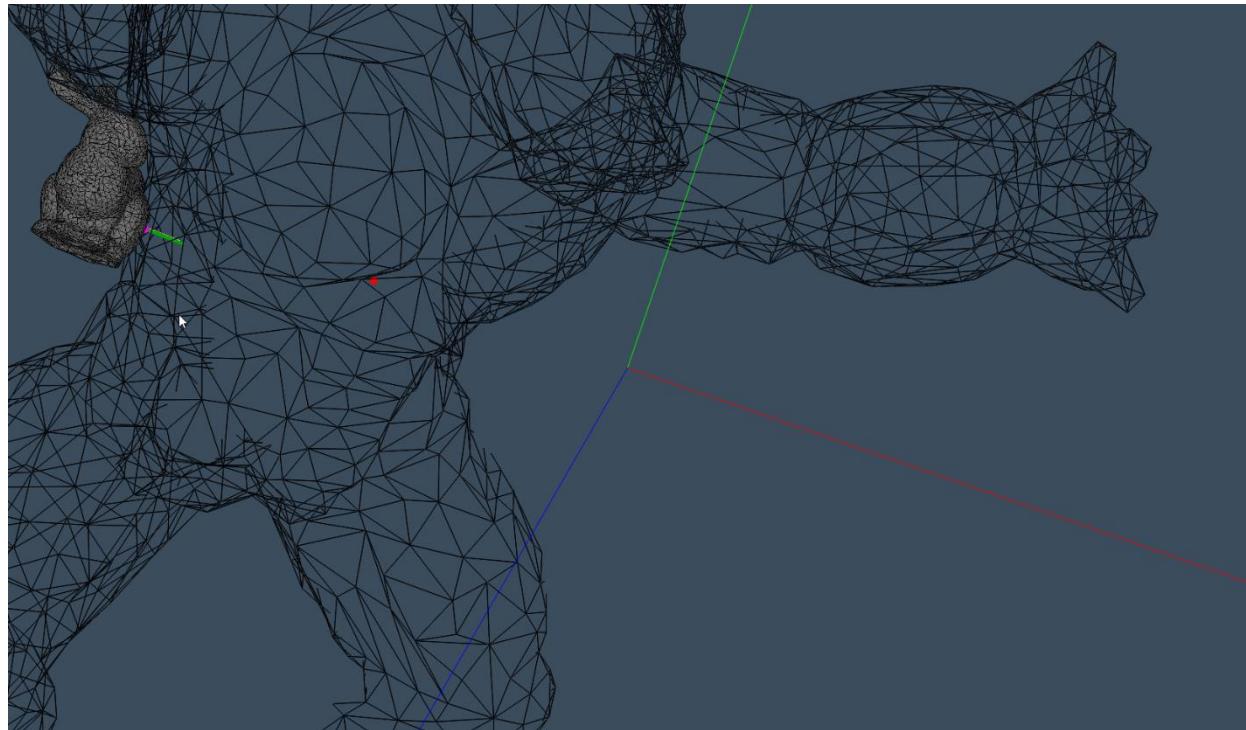
$\text{MOVE\_B3} = \text{ON}$

*reached collision*

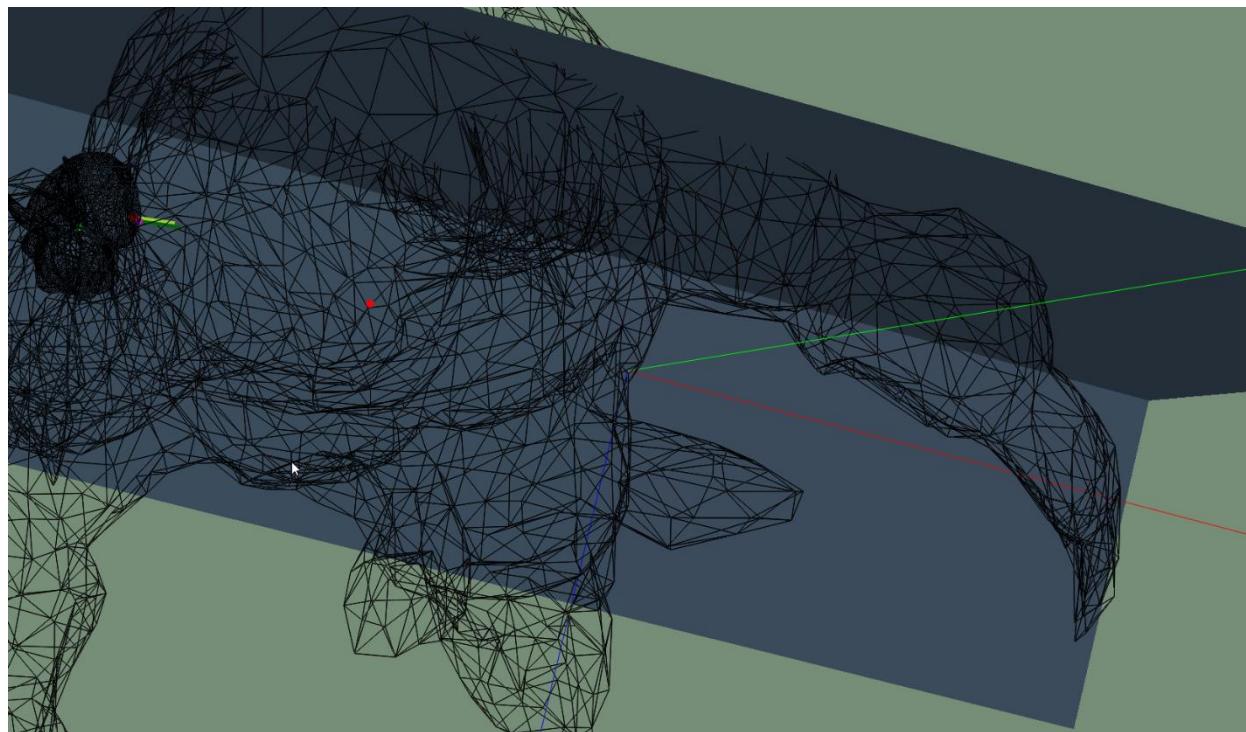
*time=4.854*

*theoretical time =4.76315*

Άρα, θα έχω καθυστέρηση προσομοίωσης = *time - theoretical time* = 0.09 sec, που οδηγεί σε μερική είσοδο του αντικειμένου μέσα στο άλλο αντικείμενο και απόκλιση του σημείου επαφής από την θέση του για θεωρητικό χρόνο όπως φαίνεται στα παρακάτω σχήματα:



Εικόνα 31



Εικόνα 32

Παρατηρώ ότι σε αυτήν την περίπτωση έχω πολύ καλή ακρίβεια και ταυτόχρονα ο χρόνος εκτέλεσης είναι 4 φορές μικρότερος από την εκτέλεση του αλγορίθμου χωρίς sdf για συγκρίσιμη ακρίβεια.

**B\_4)** Για τον υπολογισμό του διανύσματος της δύναμης απόκρισης σύγκρουσης και την προσομοίωση του αποτελέσματος πρέπει αρχικά να καθοριστούν η μάζα ενός πλέγματος και ορισμένα ακόμα μεγέθη.

Για τον καθορισμό της μάζας ενός πλέγματος πρέπει πρώτα να βρω τον όγκο του, το οποίο επιτυγχάνεται βρίσκοντας τον όγκο του κάθε τριγώνου στο πλέγμα και προσθέτοντάς τους. Για το κάθε τρίγωνο βρίσκω τον προσημασμένο όγκο του τετράεδρου που δημιουργούν τα σημεία του με το σημείο (0,0,0) από τον τύπο της διακρίνουσας του 3x3 πίνακα που περιέχει στις στήλες του τις συντεταγμένες των σημείων του τριγώνου και διαιρώντας με 6. Η διαίρεση με το 6 γίνεται επειδή η διακρίνουσα είναι στην πραγματικότητα ο όγκος του παραλληλεπίπεδου που δημιουργούν τα 3 σημεία ως προς το (0,0,0), το οποίο περιέχει 6 τετράεδρα. Επειδή ο όγκος του κάθε τριγώνου είναι προσημασμένος ανάλογα με την κατεύθυνση της κάθε έδρας του παραλληλεπίπεδου, τα σημεία των όγκων των τριγώνων που τέμνονται προσθαφαιρούνται.

$$V_i = \frac{1}{6} (p_1 \cdot p_2) \cdot p_3, V = \Sigma(V_i)$$

Ορίζοντας μία πυκνότητα density για το πλέγμα προκύπτει η ολική μάζα του από τον τύπο:

$$m = V * density$$

Η τοπική μάζα του κάθε σημείου του πλέγματος βρίσκεται διαιρώντας την ολική μάζα με τον αριθμό σημείων του πλέγματος θεωρώντας ομογενή κατανομή της μάζας στο κάθε σημείο του:

$$m_k = \frac{m}{number\_of\_points}$$

Μετά βρίσκω τον πίνακα για τον τανυστή της ροπής αδράνειας του κάθε πλέγματος όταν αυτό ορίζεται ως συμπαγές αντικείμενο αποτελούμενο από σημειακές μάζες  $m_k$ :

$$I = \begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{bmatrix}$$

$I_{ij} = \sum_{k=1}^N m_k (||r_k||^2 \delta_{ij} - x_i^k x_j^k)$ , όπου τα  $i, j$ , όταν είναι ίσα με 1,2 ή 3 αντιστοιχούν στον άξονα x, y ή z αντίστοιχα,  $r_k$  είναι το διάνυσμα από το κέντρο μάζας του αντικειμένου μέχρι το σημείο  $m_k$  και  $\delta_{ij} = 1$  μόνο για  $i = j$  και  $\delta_{ij} = 0$  σε κάθε άλλη περίπτωση. Τον πίνακα αυτόν τον αντιστρέφω και παίρνω τον  $I_{ij}^{-1}$ .

Το αποτέλεσμα της δύναμης αντίδρασης το αναπαριστώ με την στιγμιαία ορμή αντίδρασης που βρίσκεται από τον τύπο:

$$j_r = \frac{-(1+e)\vec{\nu}_r \cdot \hat{n}}{m_1^{-1} + m_2^{-1} + I_1^{-1}(\vec{r}_1 \times \hat{n}) \times \vec{r}_1 + I_2^{-1}(\vec{r}_2 \times \hat{n}) \times \vec{r}_2}$$

Όπου  $\vec{v}_r$  είναι η σχετική ταχύτητα των αντικειμένων στο σημείο επαφής που βρίσκεται από τον τύπο:

$$\vec{v}_r = \vec{v}_{p2} - \vec{v}_{p1} = \vec{v}_2 + \vec{\Omega}_2 \times \vec{r}_2 - \vec{v}_1 + \vec{\Omega}_1 \times \vec{r}_1$$

$\vec{r}_1$  και  $\vec{r}_2$  είναι τα διανύσματα απόστασης των κέντρων μάζας των αντικειμένων από το σημείο επαφής και είναι ο λόγος της σχετικής νέας ταχύτητας του σημείου επαφής πάνω στο normal του προς την σχετική ταχύτητα του σημείου επαφής πριν την σύγκρουση. Θεωρώ ελαστική κρούση, οπότε  $e=1$ .

Άρα, έχω το διάνυσμα της στιγμιαίας ορμής αντίδρασης:

$$\vec{j}_r = j_r \hat{n}$$

Από τους τύπους της κινηματικής βρίσκω τις καινούριες ταχύτητες και γωνιακές ταχύτητες:

$$\vec{v}_1' = v_1 - \frac{j_r}{m_1} \hat{n}$$

$$\vec{v}_2' = v_2 - \frac{j_r}{m_2} \hat{n}$$

$$\vec{\Omega}_1' = \vec{\Omega}_1 - j_r I_1^{-1} (\vec{r}_1 \times \hat{n})$$

$$\vec{\Omega}_2' = \vec{\Omega}_2 - j_r I_2^{-1} (\vec{r}_2 \times \hat{n})$$

Παράδειγμα σύγκρουσης αντικειμένου με σφαίρα:

-Βίντεο της σύγκρουσης:

[https://www.dropbox.com/s/wvsvaks5qxw0j8l/ConvexHull\\_QLRk2sV045.mp4?dl=0](https://www.dropbox.com/s/wvsvaks5qxw0j8l/ConvexHull_QLRk2sV045.mp4?dl=0)

-Αποτελέσματα εκτέλεσης:

*mass1=0.0780372, mass2=251.729*

*SHOW\_TEMP\_SPHERE = ON*

*TEMP\_SOLID = ON*

*RUN\_B3 = ON*

*choose mode to find collision: '1'=sphere, '2'=AABB, '3'=object*

*1*

*use sdf? (y/n)*

*y*

*running sphere with mesh collision detection*

*initial coll\_time=2.6*

*min\_sdf\_final=0.0020026, final\_change=9.38717e-05*

*theoretical contact index=1042, x=-12.9719, y=1.32017, z=-0.672657*

*final coll\_time=4.48861*

*Found collision time in 0.421 seconds.*

*RUN\_B3 = OFF*

*MOVE\_B3 = ON*

*SHOW\_SOLID = OFF*

*reached collision*

*time=4.639*

*theoretical time =4.48861*

*MOVE\_B3 = OFF*

*old speeds:U1=(1.333, 0.000, 0.000), U2=(-1.333, 0.000, 0.000)*

*old angular speeds:OMEGA1=(0.393, 0.393, 0.393), OMEGA2=(0.000, 0.131, 0.000)*

*Ur\*n=-2.09008, temp\*n=17.7118*

*new speeds:U1'=(-0.364, -0.318, 0.309), U2'=(-1.333, 0.000, -0.000)*

*new angular speeds:OMEGA1'=(0.328, 0.747, 6.375), OMEGA2'=(0.000, 0.131, 0.000)*

*jr\_magnitude=0.136919*

*jr\_angles with x,y,z = (14.6387,79.545,100.131)*

*jr=(0.132475,0.0248458,-0.0240838)*

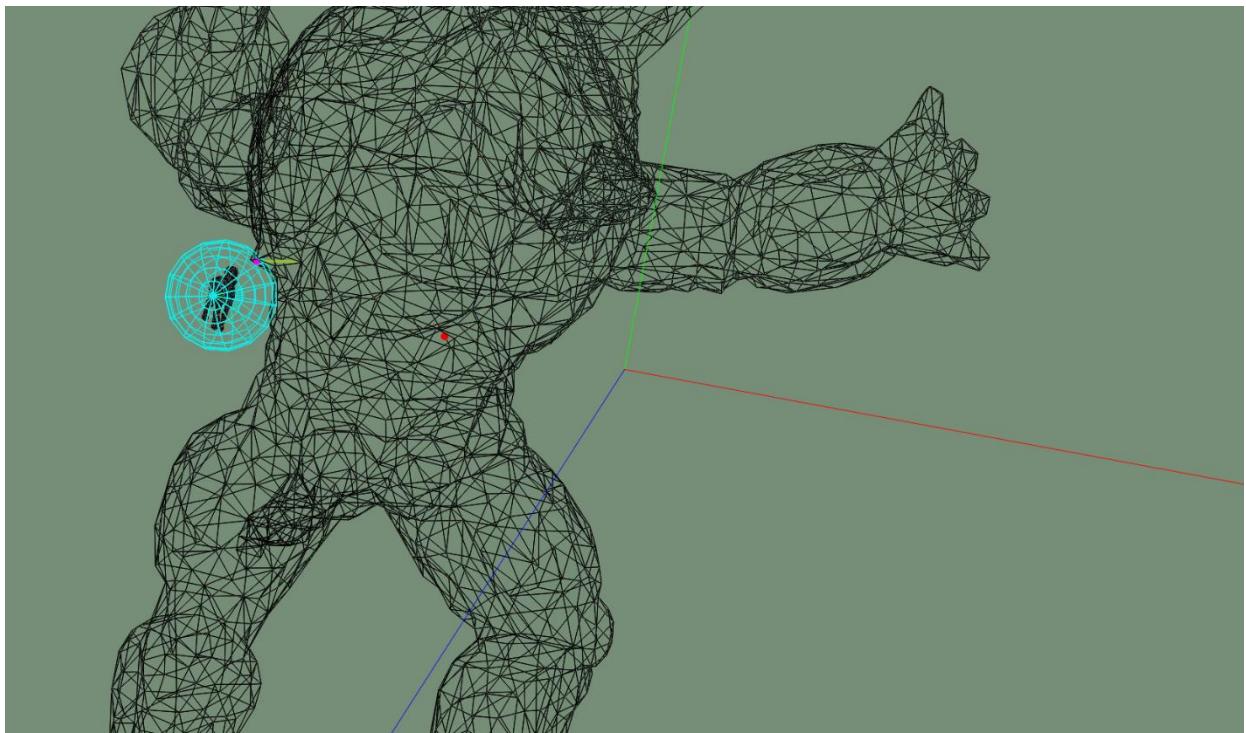
-Διάνυσμα  $\vec{J_r}$  :

$$j_{rx} = jr_{mag} * \cos(jr_{angle_x}) = 0.13247$$

$$j_{ry} = jr_{mag} * \cos(jr_{angle_y}) = 0.02485$$

$$j_{rz} = jr_{mag} * \cos(jr_{angle_z}) = , -0.02408$$

Άρα  $\vec{J_r} = 0.13247\hat{i} + 0.02485\hat{j}, -0.02408\hat{k}$  και απεικονίζεται στο παρακάτω σχήμα:



*Εικόνα 33*

Για την αντίστοιχη δύναμη αντίδρασης έχω:

$$j_r = \int_{t0}^{t1} f dt \Rightarrow f = \frac{j_r}{t1-t0}$$

Θεωρώ ότι η σύγκρουση κρατάει για χρονική διάρκεια  $t1 - t0 = 1ms$ , οπότε:

$$f = \frac{j_r}{0.001} = 1000 * j_r \quad (1)$$

Άρα, στο παράδειγμα αυτό έχω:  $f = 136.919\hat{n}$  (N)

Παράδειγμα σύγκρουσης αντικειμένου με AABB:

-Βίντεο της σύγκρουσης:

[https://www.dropbox.com/s/j0gbt3rg6hcyqqt/ConvexHull\\_S7tVxu8Yz0.mp4?dl=0](https://www.dropbox.com/s/j0gbt3rg6hcyqqt/ConvexHull_S7tVxu8Yz0.mp4?dl=0)

-Αποτελέσματα εκτέλεσης:

*mass1=0.0780372, mass2=251.729*

*TEMP\_SOLID = ON*

*SHOW\_TEMP\_AABB = ON*

*RUN\_B3 = ON*

*choose mode to find collision: '1'=sphere, '2'=AABB, '3'=object*

*use sdf? (y/n)*

*y*

*running aabb with mesh collision detection*

*initial coll\_time=2.5*

*min\_sdf\_final=0.00212385, final\_change=9.95555e-05*

*theoretical contact index=1439, x=-12.8418, y=0.524722, z=-0.495295*

*final coll\_time=5.07679*

*Found collision time in 1.034 seconds.*

*RUN\_B3 = OFF*

*MOVE\_B3 = ON*

*reached collision*

*time=5.276*

*theoretical time =5.07679*

*MOVE\_B3 = OFF*

*old speeds: U1=(1.333, 0.000, 0.000), U2=(-1.333, 0.000, 0.000)*

*old angular speeds: OMEGA1=(0.393, 0.393, 0.393), OMEGA2=(0.000, 0.131, 0.000)*

*Ur\*n=-2.36904, temp\*n=7.61561*

*new speeds: U1'=(-1.454, -0.716, 0.739), U2'=(-1.332, 0.000, -0.000)*

*new angular speeds: OMEGA1'=(-0.320, 2.331, 4.470), OMEGA2'=(0.000, 0.131, 0.000)*

*jr\_magnitude=0.231873*

*jr\_angles with x,y,z = (20.2573, 76.0597, 104.399)*

*jr=(0.217531, 0.0558605, -0.0576621)*

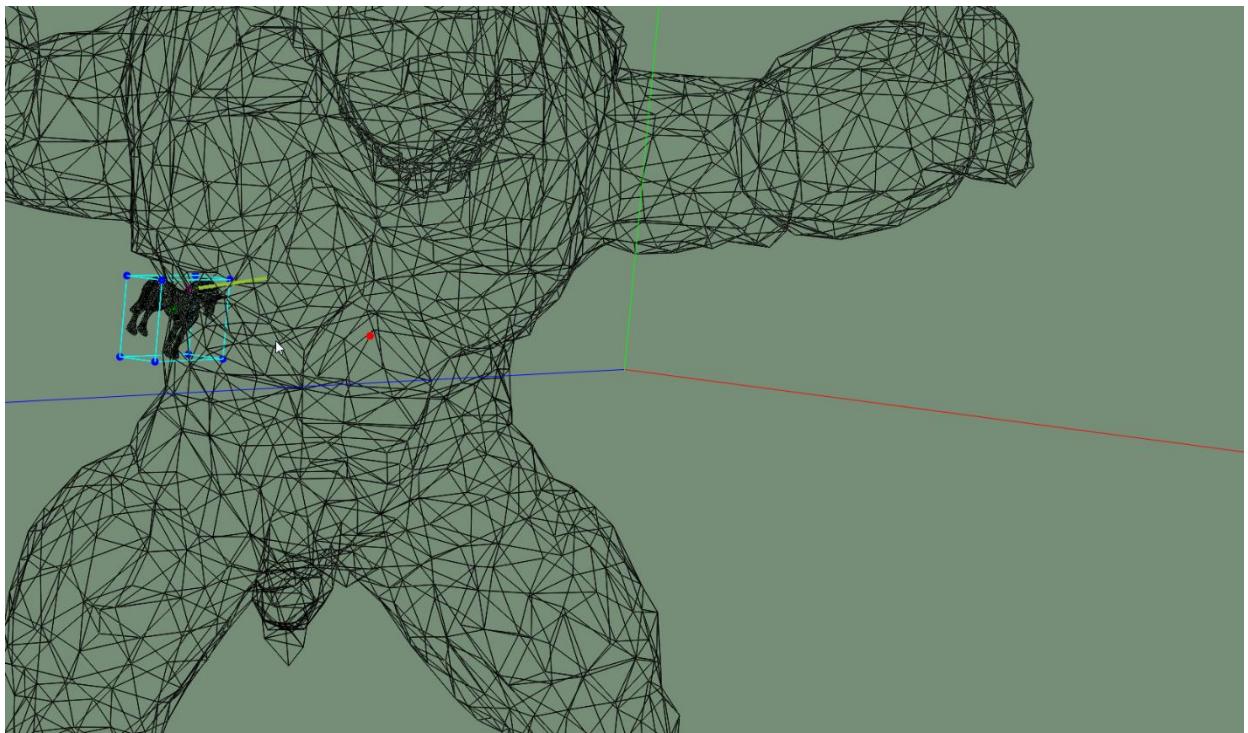
*-Διάνυσμα  $\vec{j}_r$ :*

$$j_{rx} = jr_{mag} * \cos(jr_{angle_x}) = 0.21753$$

$$j_{ry} = jr_{mag} * \cos(jr_{angle_y}) = 0.05586$$

$$j_{rz} = jr_{mag} * \cos(jr_{angle_z}) = -0.05766$$

*Άριθμος  $\vec{j}_r = 0.21753\hat{i} + 0.05586\hat{j} - 0.05766\hat{k}$  και απεικονίζεται στο παρακάτω σχήμα:*



Εικόνα 34

Από την (1) έχω:  $f = 231.873 \hat{n}$  (N)

Παράδειγμα σύγκρουσης αντικειμένου με άλλο αντικείμενο:

-Βίντεο της σύγκρουσης:

[https://www.dropbox.com/s/ljm9ab28iawx2yu/ConvexHull\\_GUfxd5UKjv.mp4?dl=0](https://www.dropbox.com/s/ljm9ab28iawx2yu/ConvexHull_GUfxd5UKjv.mp4?dl=0)

-Αποτελέσματα εκτέλεσης:

$mass1=1.41576, mass2=251.729$

$TEMP\_SOLID = ON$

$RUN\_B3 = ON$

*running collision detection*

*initial coll\_time=2.6*

*min\_sdf\_final=0.0706481, final\_change=0.000419855*

*contact index=379, x=-12.3153, y=-0.695812, z=-0.61498*

*length=2503*

*final coll\_time=4.76315*

Found collision time in 219.872 seconds.

*RUN\_B3 = OFF*

*MOVE\_B3 = ON*

*reached collision*

*time=4.911*

*theoretical time =4.76315*

*MOVE\_B3 = OFF*

*old speeds:U1=(1.333, 0.000, 0.000), U2=(-1.333, 0.000, 0.000)*

*old angular speeds: $\Omega_1=(0.393, 0.393, 0.393)$ ,  $\Omega_2=(0.000, 0.131, 0.000)$*

*new speeds:U1'=(-3.649, 0.302, 0.414), U2'=(-1.305, -0.002, -0.002)*

*new angular speeds: $\Omega_1'=(0.228, 1.519, -0.921)$ ,  $\Omega_2'=(0.000, 0.131, -0.000)$*

*jr\_magnitude=7.09084*

*jr\_angles with x,y,z = (5.8739, 93.4602, 94.7407)*

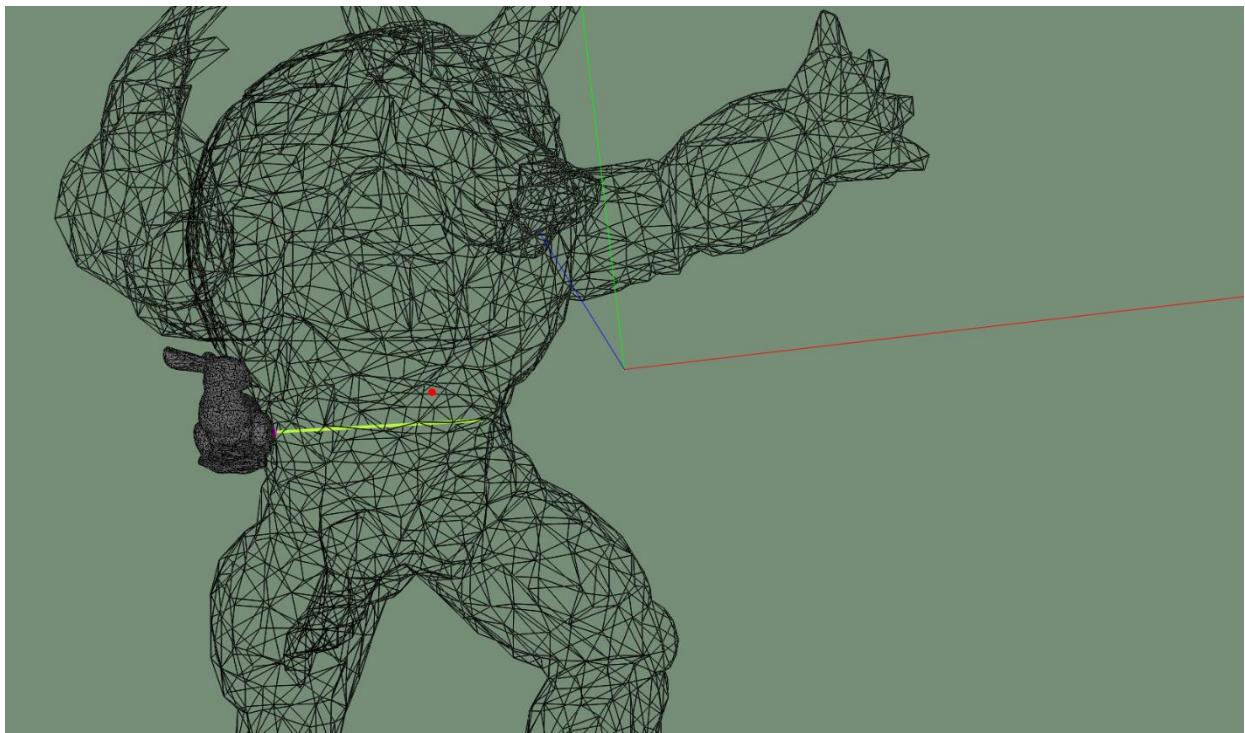
-Διάνυσμα  $\vec{J_r}$ :

$$j_{rx} = jr_{mag} * \cos(jr_{angle_x}) = 7.0536$$

$$j_{ry} = jr_{mag} * \cos(jr_{angle_y}) = -0.428$$

$$j_{rz} = jr_{mag} * \cos(jr_{angle_z}) = -0.586$$

Άρα  $\vec{J_r} = 7.0536\hat{i} - 0.428\hat{j} - 0.586\hat{k}$  και απεικονίζεται στο παρακάτω σχήμα:

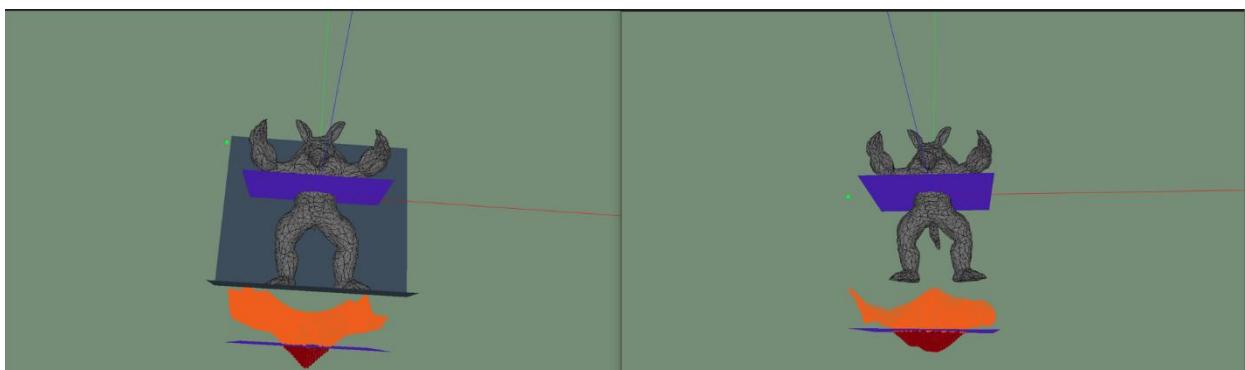


Εικόνα 35

Από την (1) έχω:  $f = 709.084\hat{n}$  (N)

**B\_5)** Για να παράγω την sdf μέσω ενός νευρωνικού δικτύου άμεσης αναπαράστασης με συναρτήσεις περιοδικής ενεργοποίησης που ονομάζεται siren χρησιμοποίησα κώδικα σε python που παίρνει ένα αντικείμενο σε τύπο αρχείου .xyz , το εκπαιδεύει με τον κώδικα του siren (επέλεξα το “armadillo\_low\_low” αντικείμενο για 1500 περίπου εποχές) και μετά δημιουργεί για την εποχή 1500 ένα μοντέλο που μπορεί να υπολογίσει την sdf για ένα δοσμένο νέφος σημείων. Παρατηρώντας το αντικείμενο που αναπαράγεται από την επαλήθευση της εκπαίδευσης , είναι μετατοπισμένο και σε μικρότερη κλίμακα από το αρχικό , άρα κάνω τις αντίστοιχες μετατροπές για το νέφος σημείων που αποθηκεύω στο αρχείο για επεξεργασία από το μοντέλο της sdf.

Παρακάτω βλέπω μια σύγκριση μεταξύ της sdf που βρήκα από το siren μοντέλο (δεξιά) και της sdf που βρήκα με την μέθοδο από το ερώτημα B\_1):



Εικόνα 36

Είναι εμφανές ότι έχω κάποιες απώλειες αλλά σε γενικές γραμμές ακολουθεί την ίδια μορφή με την μέθοδο του B\_1).

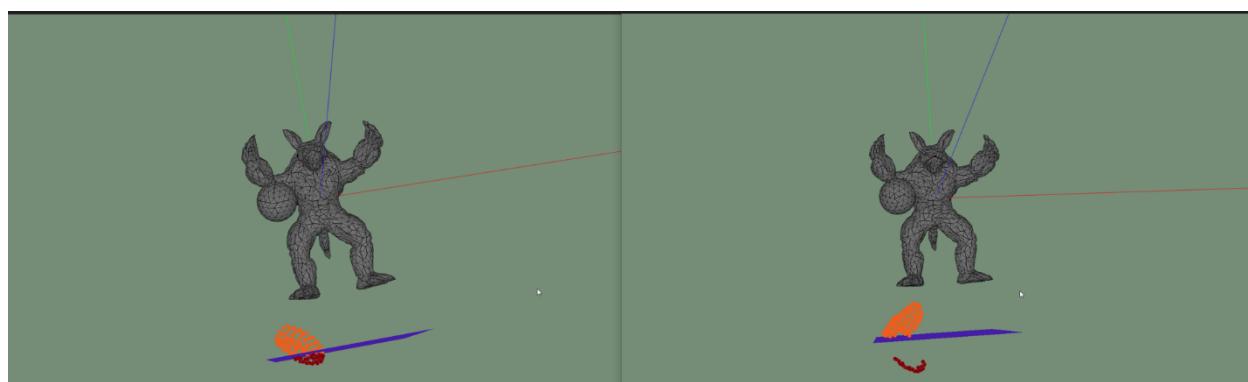
Όσον αφορά την ταχύτητα έχω τα παρακάτω αποτελέσματα εκτέλεσης:

*Time for neural function = 5.291*

*Time for B\_1) function = 6.005*

Βλέπω ότι υπάρχει εμφανής βελτίωση , η οποία μπορεί να μεγαλώσει αν μειωθούν οι καθυστερήσεις από την επικοινωνία του προγράμματος με το sdf\_model\_create.py μέσω αρχείων text. Αυτό φαίνεται και από το παρακάτω παράδειγμα , όπου για τον μικρό αριθμό σημείων της σφαίρας η siren μέθοδος είναι πιο αργή:

Παράδειγμα με σφαίρα αντί για επίπεδο (~160 σημεία):



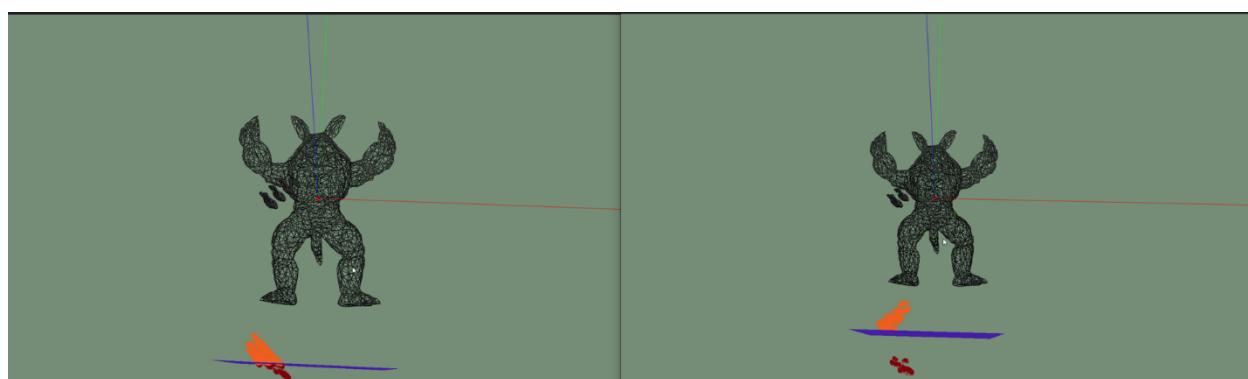
*Εικόνα 37*

Όσον αφορά την ταχύτητα έχω τα παρακάτω αποτελέσματα εκτέλεσης:

*Time for siren function = 5.022*

*Time for B\_1) function = 0.480999*

Παράδειγμα με κορίνες(14 χιλιαδες σημεία):



*Εικόνα 38*

Όσον αφορά την ταχύτητα έχω τα παρακάτω αποτελέσματα εκτέλεσης:

*Time for siren function = 9.856*

*Time for B\_1) function = 35.298*

Από τα παραπάνω συμπεραίνω ότι η μέθοδος siren στην τρέχουσα υλοποίησή της επιταχύνει την εύρεση της sdf μόνο για μεγάλο αριθμό σημείων (χιλιάδες) και παρουσιάζει εμφανή απόκλιση στις τιμές της sdf ακόμα και για 1500 εποχές , το οποίο δημιουργεί ανακρίβεια κατά την εκτέλεση σύγκρουσης.

*Βιβλιογραφία :*

- (1) [\*G. Taubin. "Curve and surface smoothing without shrinkage," Proceedings of IEEE International Conference on Computer Vision, 1995, pp. 852-857, doi: 10.1109/ICCV.1995.466848.\*](#)
- (2) [\*Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rossi and H. P. Seidel, "Differential coordinates for interactive mesh editing," Proceedings Shape Modeling Applications, 2004., 2004, pp. 181-190, doi: 10.1109/SMI.2004.1314505.\*](#)
- (3) [\*Bærentzen, Andreas & Aanæs, Henrik. \(2002\). Generating Signed Distance Fields From Triangle Meshes.\*](#)
- (4) [\*Angelo, Luca Di, and Paolo Di Stefano. "Experimental comparison of methods for differential geometric properties evaluation in triangular meshes." Computer-Aided Design and Applications 8.2 \(2011\): 193-210.\*](#)
- (5) [\*Mesh Volume\*](#)
- (6) [\*Density, wikipedia\*](#)
- (7) [\*https://en.wikipedia.org/wiki/Collision\\_detection\*](https://en.wikipedia.org/wiki/Collision_detection)
- (8) [\*http://www.cs.uu.nl/docs/vakken/ddm/2015-2016/Lecture%20-%20Collision%20Detection.pdf\*](http://www.cs.uu.nl/docs/vakken/ddm/2015-2016/Lecture%20-%20Collision%20Detection.pdf)
- (9) [\*https://developer.mozilla.org/en-US/docs/Games/Techniques/3D\\_collision\\_detection\*](https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection)
- (10)[\*https://en.wikipedia.org/wiki/Collision\\_response\*](https://en.wikipedia.org/wiki/Collision_response)
- (11)[\*https://en.wikipedia.org/wiki/Moment\\_of\\_inertia#Definition\*](https://en.wikipedia.org/wiki/Moment_of_inertia#Definition)
- (12)[\*https://en.wikipedia.org/wiki/Rotation\\_matrix#In\\_three\\_dimensions\*](https://en.wikipedia.org/wiki/Rotation_matrix#In_three_dimensions)
- (13)[\*https://www.computationalimaging.org/publications/siren/\*](https://www.computationalimaging.org/publications/siren/)