Πανεπιστήμιο Πελοποννήσου

Σχολή Θετικών επιστημών και Τεχνολογίας

Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών

Πρόγραμμα Μεταπτυχιακών Σπουδών

Διπλωματική Εργασία

Απλοποίηση Τριγωνικών Μοντέλων στην GPU

Παπαγεωργίου Αλέξανδρος

ΑΜ: 2009021

Επιβλέπων καθηγητής
Πλατής Νικόλαος

Τρίπολη, Φεβρουάριος 2012

Περίληψη

Τα μοντέλα που αναπαριστούν τρισδιάστατα αντικείμενα, κυρίως με την μορφή των τριγωνικών πλεγμάτων, χρησιμοποιούνται σε πάρα πολλές εφαρμογές στα γραφικά υπολογιστών. Οι τρόποι παραγωγής τέτοιων μοντέλων επιτρέπουν τη δημιουργία τριγωνικών πλεγμάτων με μεγάλο βαθμό λεπτομέρειας, κάτι που όμως δεν είναι επιθυμητό για όλες τις χρήσεις αυτών. Για την επίλυση αυτού του προβλήματος έχουν δημιουργηθεί αλγόριθμοι απλοποίησης μοντέλων οι οποίοι παράγουν απλούστερα μοντέλα με βάση τα αρχικά, με τους περισσότερους να είναι υλοποιημένοι για την CPU. Τα τελευταία χρόνια ωστόσο παρατηρείται μια ραγδαία ανάπτυξη στις δυνατότητες των επεξεργαστών γραφικών (GPUs) σε σημείο που να έχουν ξεπεράσει τις CPUs σε επεξεργαστική ισχύ.

Αυτή η ραγδαία αύξηση της ισχύς των GPUs μαζί με την ανάπτυξη τεχνολογιών για την χρήση τους για υπολογισμούς γενικού σκοπού είναι που μας οδήγησε στην χρήση τους για απλοποίηση μοντέλων. Σε αυτή την εργασία παρουσιάζουμε έναν αλγόριθμο απλοποίησης μοντέλων που εκτελεί συρρικνώσεις ακμών οδηγούμενες από τη μετρική σφάλματος βασισμένη στα quadrics, ο οποίος εκμεταλλεύεται τις δυνατότητες των σύγχρονων καρτών γραφικών. Καθώς οι GPUs μπορούν να αντιμετωπιστούν σαν πολυεπεξεργαστικά συστήματα διαμοιρασμένης μνήμης, ο αλγόριθμος που υλοποιήσαμε χρησιμοποιεί παραλληλισμό δεδομένων όπως αυτός παρέχεται μέσω της τεχνολογίας OpenCL και δεν έχει σειριακά τμήματα στην κύρια επαναληπτική δομή του ώστε να χρησιμοποιεί πλήρως την επεξεργαστική ισχύ των GPUs. Ο αλγόριθμος απλοποιεί τριγωνικά πλέγματα που έχουν την ιδιότητα της πολλαπλότητας και η υλοποίησή του παράγει αποτελέσματα γρηγορότερα σε σχέση με αντίστοιχη σειριακή υλοποίηση.

Abstract

Models that represent three-dimensional objects, mostly in the form of triangular meshes, are used in many applications in computer graphics. The methods that produce such models allow the creation of triangular meshes with a high degree of detail, which is not always desirable for all of their uses. To solve this problem, simplification algorithms have been developed which produce simpler models based on the original ones; most of these algorithms are implemented on the CPU. In recent years, however, there has been a rapid development in the capabilities of graphics processors (GPUs) to the point that they exceed CPUs in processing power.

This rapid increase in the power of GPUs along with the development of technologies to use them for general purpose calculations led us to use them for model simplification. In this thesis we present a model simplification algorithm that performs edge contractions driven by a quadric based error metric, which takes advantage of modern graphics cards. As GPUs can be treated as shared memory multiprocessor systems, the algorithm we developed is using data parallelism as provided by OpenCL and has no serial segments in the main iterative structure in order to fully utilize the processing power of GPUs. The algorithm simplifies triangular meshes that possess the property of 2-manifold and its implementation produces results faster than the corresponding serial implementation.

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

Στα γραφικά υπολογιστών, η οπτικοποίηση και ο χειρισμός μοντέλων που αναπαριστούν τρισδιάστατα αντικείμενα αποτελεί μια βασική και πολλές φορές χρονοβόρα διαδικασία. Τα μοντέλα δημιουργούνται κυρίως με την χρήση σαρωτή (3D scanner) για την ψηφιοποίηση αντικειμένων του πραγματικού κόσμου, ή με την χρήση προγραμμάτων μοντελοποίησης, και μπορούν να προκύψουν μοντέλα με αρκετά μεγάλο βαθμό λεπτομέρειας. Για τις διάφορες χρήσεις των μοντέλων όμως αυτός ο βαθμός λεπτομέρειας δεν είναι πάντα ο κατάλληλος. Μπορεί ακόμα να είναι και επιβαρυντικός παράγοντας καθώς τα προγράμματα που θα τα χρησιμοποιήσουν πρέπει να διαχειρίζονται τα αντίστοιχα μεγάλα ποσά δεδομένων. Όταν εξετάζουμε ένα μοντέλο από πολύ κοντά, θέλουμε να έχουμε όσο το δυνατόν μεγαλύτερο βαθμό λεπτομέρειας, όταν όμως αρχίζουμε να απομακρυνόμαστε, αυτές οι λεπτομέρειες έχουν μικρότερη επίπτωση στην εμφάνιση του μοντέλου, μέχρι κάποιο σημείο όπου παύουν να είναι ορατές, και μεγαλύτερη σημασία να έχουν τα γενικότερα χαρακτηριστικά του. Το ιδανικό θα ήταν να υπάρχουν τα μοντέλα στον κατάλληλο βαθμό λεπτομέρειας που χρειαζόμαστε κάθε φορά.

Αυτό το πρόβλημα προσπαθεί να αντιμετωπίσει η απλοποίηση μοντέλων, να δημιουργήσει δηλαδή νέα μοντέλα βάσει των αρχικών, στο βαθμό λεπτομέρειας που απαιτείται ανάλογα με τη χρήση τους και τους διαθέσιμους υπολογιστικούς πόρους. Δεδομένων τέτοιων αρχικών λεπτομερών μοντέλων, έχουν αναπτυχθεί διάφοροι αλγόριθμοι απλοποίησης, οι οποίοι παράγουν μοντέλα-αποτελέσματα στον επιθυμητό βαθμό λεπτομέρειας. Αυτό στην βιβλιογραφία αποτελεί ένα θέμα που έχει μελετηθεί αρκετά, με τεχνικές όπως την συρρίκνωση ακμής και την απαλοιφή κορυφών να παράγουν αρκετά ποιοτικά αποτελέσματα σε αποδεκτό χρόνο. Ωστόσο οι περισσότεροι αλγόριθμοι είναι υλοποιημένοι κατά κύριο λόγο για την CPU. Με την εξέλιξη της τεχνολογίας όμως έχουμε φτάσει στο σημείο που οι κάρτες γραφικών (GPU) να έχουν ξεπεράσει τις CPU σε επεξεργαστική ισχύ.

Ο στόχος αυτής της διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός αλγόριθμου απλοποίησης που να εκμεταλλεύεται τις δυνατότητες των σύγχρονων καρτών γραφικών. Ο αλγόριθμος απλοποίησης που δημιουργήθηκε βασίζεται στην μετρική σφάλματος με την χρήση quadrics και υλοποιήθηκε με την χρήση της τεχνολογίας OpenCL.

Η δομή του κειμένου της εργασίας, όπως έχει αναπτυχθεί στα κεφάλαια που ακολουθούν, είναι η παρακάτω:

Κεφάλαιο 2 Παρουσιάζονται οι απαραίτητες έννοιες για την αναπαράσταση των μοντέλων και την απλοποίηση τους. Γίνεται μια παρουσίαση της μετρικής σφάλματος βα-

σισμένης σε quadrics που θα χρησιμοποιηθεί και κάποια βασικά στοιχεία σχετικά με τους παράλληλους αλγόριθμους που θα χρησιμοποιήσουμε στην συνέχεια της εργασίας.

Κεφάλαιο 3 Περιγράφεται η αρχιτεκτονική των GPU, αντιπαραθέτοντάς την με αυτή των CPU και παρουσιάζεται με λεπτομέρεια η τεχνολογία OpenCL που θα χρησιμοποιηθεί την εκμετάλλευση των δυνατοτήτων τους στην φάση της υλοποίησης.

Κεφάλαιο 4 Παρουσιάζεται και αναλύεται ο αλγόριθμος που αναπτύχθηκε στα πλαίσια της εργασίας. Εξηγούμε με επαρκή λεπτομέρεια τον βασικό αλγόριθμο που υλοποιήσαμε και αναλύουμε τα τμήματα που τον συνθέτουν καθώς και τις δομές δεδομένων που χρησιμοποιήσαμε.

Κεφάλαιο 5 Εξετάζονται τα ποιοτικά αποτελέσματα του αλγόριθμου χρησιμοποιώντας τα αποτελέσματα του προγράμματος που τον υλοποιεί και εξηγείται η επίδραση που έχουν οι παράμετροι του αλγόριθμου στο τελικό αποτέλεσμα.

Κεφάλαιο 6 Συγκρίνονται τα μοντέλα που παράγει ο παράλληλος αλγόριθμος που αναπτύχθηκε με αυτά αντίστοιχης σειριακής υλοποίησης του αλγόριθμου απλοποίησης

Κεφάλαιο 7 Συνοψίζεται η εργασία και παρουσιάζονται τρόποι με τους οποίους θα μπορούσε να συνεχιστεί στο μέλλον.

Κεφάλαιο 2

Απλοποίηση στα γραφικά υπολογιστών

Ο στόχος της απλοποίησης των μοντέλων στα γραφικά υπολογιστών, είναι να παρέχουμε έναν μηχανισμό για τον έλεγχο της πολυπλοκότητας τους. Πριν όμως αναφερθούμε στους τους διάφορους τρόπους απλοποίησης των μοντέλων, πρέπει αρχικά να αναφέρουμε τον τρόπο αναπαράστασης τους στον υπολογιστή και να δώσουμε κάποιους ορισμούς που θα χρησιμοποιήσουμε παρακάτω.

2.1 Αναπαράσταση μοντέλων

Για την αναπαράσταση μοντέλων υπάρχουν τρόποι που εντάσσονται σε δυο κύριες κατηγορίες, η πρώτη βασισμένη στον όγκο που καταλαμβάνει το μοντέλο και η δεύτερη λαμβάνοντας υπόψη μόνο την επιφάνειά του [TPPP08].

Στην πρώτη κατηγορία επεκτείνεται η ιδέα των pixels στις τρεις διαστάσεις, όπου υποδιαιρείται ο χώρος σε μικροσκοπικούς κύβους (voxels) οι οποίοι μπορούν να χρησιμοποιηθούν για την προσέγγιση του όγκου των αντικειμένων. Η αναπαράσταση αυτή χρησιμοποιείται για ημιδιαφανή αντικείμενα ή όταν μας ενδιαφέρει η εσωτερική δομή του αντικειμένου. Τα περισσότερα μοντέλα όμως είναι αδιαφανή και κατά την οπτικοποίησή τους το αποτέλεσμα καθορίζεται μόνο από την εξωτερική τους εμφάνιση χωρίς να μας ενδιαφέρει η εσωτερική τους δομή. Σε αυτές τις περιπτώσεις χρησιμοποιούμε τρόπους αναπαράστασης του μοντέλου που ανήκουν στην δεύτερη κατηγορία.

Για την αναπαράσταση των επιφανειών μπορούν να χρησιμοποιηθούν τρόποι που βασίζονται σε μαθηματικές περιγραφές αυτών, όπως επιφάνειες NURBS, επιφάνειες υποδιαίρεσης και γενικές παραμετρικές επιφάνειες. Αυτοί οι τρόποι δίνουν ακριβής αναπαραστάσεις των επιφανειών αλλά δεν μπορούν να περιγράψουν μοντέλα με τυχαία σχήματα. Έτσι χρησιμοποιούνται πολυγωνικά μοντέλα: σύνολα επίπεδων πολυγώνων που κατασκευάζονται μέσω συνόλων σημείων τα οποία χρησιμοποιούνται σαν κορυφές των πολυγώνων.

Στην πράξη τα πολύγωνα που χρησιμοποιούνται είναι τα τρίγωνα, οπότε έχουμε τριγωνικά μοντέλα (ή τριγωνικά πλέγματα), καθώς με τρία σημεία ορίζεται ένα επίπεδο στον τρισδιάστατο χώρο, οπότε τα χρησιμοποιούμενα πολύγωνα θα είναι πάντα επίπεδα. Επί-

σης κάθε πολύγωνο μπορεί να τριγωνοποιηθεί σε ένα στάδιο προεπεξεργασίας του μοντέλου, οπότε μπορούμε χωρίς απώλεια της γενικότητας να υποθέσουμε πως πάντοτε θα χειριζόμαστε τριγωνικά πλέγματα. Ο κυρίαρχος λόγος όμως που εστιάζουμε στα τριγωνικά πλέγματα είναι καθαρά πρακτικός: τα τριγωνικά πλέγματα είναι ευέλικτα και υποστηρίζονται σχεδόν από όλα τα συστήματα μοντελοποίησης. Η υποστήριξη υλικού για τέτοιες επιφάνειες είναι όλο και περισσότερο διαθέσιμη σε καταναλωτικό επίπεδο, με τις κάρτες γραφικών να γίνονται όλο και πιο ισχυρές και επί του παρόντος, για κανέναν άλλο τρόπο αναπαράστασης των μοντέλων δεν υπάρχει τόσο ευρεία υποστήριξη. Ο τρόπος αναπαράστασης των μοντέλων που θα χρησιμοποιήσουμε, για τους λόγους που αναφέρθηκαν παραπάνω, είναι με τριγωνικά πλέγματα.

Σαν μια πιο αυστηρή περιγραφή ενός τριγωνικού πλέγματος μπορούμε να πούμε ότι είναι το ζεύγος $\mathrm{M} = (\mathrm{V}, \mathrm{T})$ με V να είναι η λίστα κορυφών του μοντέλου και T η λίστα των τριγώνων που αποτελούν το μοντέλο. Η λίστα $\mathrm{V} = (\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_\mathrm{r})$ είναι μια διατεταγμένη ακολουθία, με κάθε σημείο να αναγνωρίζεται μοναδικά από έναν αριθμό i. Η λίστα των τριγώνων $\mathrm{T} = (\mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_\mathrm{n})$ είναι επίσης μια διατεταγμένη ακολουθία, με κάθε τρίγωνο να αναγνωρίζεται μοναδικά από έναν αριθμό. Κάθε σημείο $\mathrm{v}_\mathrm{i} = (\mathrm{x}_\mathrm{i}, \mathrm{y}_\mathrm{i}, \mathrm{z}_\mathrm{i})$ είναι ένα διάνυσμα στον Ευκλείδειο χώρο R^3 και κάθε τρίγωνο $\mathrm{t}_\mathrm{i} = (\mathrm{i}, \mathrm{j}, \mathrm{k})$ είναι μια διατεταγμένη λίστα τριών δεικτών στη λίστα των σημείων, τα οποία προσδιορίζουν τις τρεις κορυφές του τριγώνου.

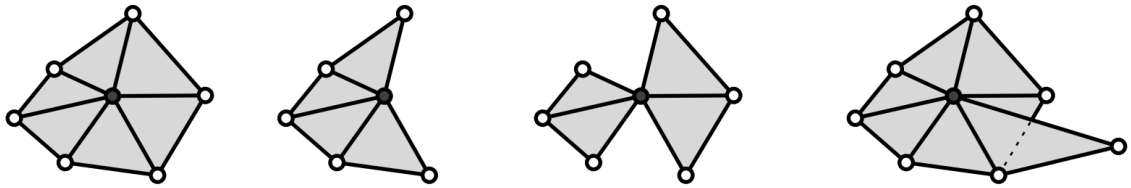
Υπάρχουν όμως και κάποιοι τρόποι αναπαράστασης μοντέλων που δεν μπορούμε να τους εντάξουμε στις παραπάνω κατηγορίες οι οποίοι περιληπτικά είναι οι εξής:

Κατασκευαστική στερεάς γεωμετρίας – Constructive Solid Geometry Ξεκινώντας από ένα σύνολο βασικών στερεών (παραλληλεπίπεδο, σφαίρα, κύλινδρος, κώνος), μπορούμε να συνθέσουμε πολύπλοκα αντικείμενα με την χρήση βασικών πράξεων συνόλων (ένωση, αφαίρεση, τομή).

Συστήματα Σωματιδίων – Particle systems Με σύνολα σωματιδίων μπορούν να δημιουργηθούν μοντέλα προσέγγισης φαινομένων που δεν έχουν μια σαφή γεωμετρική περιγραφή αλλά είναι γνωστοί οι φυσικοί κανόνες που διέπουν τα σωματίδιά τους (καπνός, φωτιά, σύννεφα)

2.2 Πολλαπλότητα

Με τον όρο πολλαπλότητα (manifold) διάστασης 2 εννοούμε μία επιφάνεια της οποίας κάθε σημείο έχει μία «γειτονιά» τοπολογικά ισοδύναμε με έναν επίπεδο δίσκο. Πολλαπλότητα με σύνορο είναι μία επιφάνεια της οποίας κάθε σημείο έχει μια γειτονιά που η τοπολογία του είναι ένας επίπεδος δίσκος ή ημιδίσκος. Μια τριγωνική επιφάνεια έχει την ιδιότητα της πολλαπλότητας αν κάθε ακμή μεταξύ δύο σημείων συμμετέχει σε κάποιο τρίγωνο, μπορεί να διαμοιράζεται μεταξύ ακριβώς δύο τριγώνων, ή να εμφανίζεται μόνο σε ένα τρίγωνο αν είναι συνοριακή ακμή. Στο σχήμα 2.1 από αριστερά προς δεξιά φαίνονται τριγωνικά πλέγματα από τα οποία το πρώτο έχει την ιδιότητα της πολλαπλότητας, το δεύτερο έχει την ιδιότητα της πολλαπλότητας με σύνορο και δυο παραδείγματα που δεν έχουν την ιδιότητα της πολλαπλότητας. Ειδικότερα για αυτά, στο ένα χάνεται η ιδιότητα της πολλαπλότητας καθώς η γειτονιά της κεντρικής κορυφής αποτελείται από περισσότερους του ενός ημιδίσκους και στο άλλο επειδή έχει μια ακμή που την μοιράζονται παραπάνω από δυο τρίγωνα.



Σχήμα 2.1: Πολλαπλότητα και μη πολλαπλότητα

Τα περισσότερα τριγωνικά πλέγματα που χρησιμοποιούνται στην πράξη έχουν την ιδιότητα της πολλαπλότητας και μάλιστα αρκετοί αλγόριθμοι χειρισμού τους, την απαιτούν.Η απαίτηση αυτή απλοποιεί τις δομές δεδομένων που χρησιμοποιούνται για την αναπαράσταση των μοντέλων και απλοποιεί το χειρισμό και την απλοποίησή τους.

2.3 Απλοποίηση

Ένα τριγωνικό πλέγμα έχει ένα δεδομένο αριθμό κορυφών και τριγώνων και έτσι παρέχει μια συγκεκριμένη ανάλυση για την αναπαράσταση ενός μοντέλου. Αυτός όμως ο βαθμός λεπτομέρειας δεν είναι πάντα κατάλληλος γα όλες τις εφαρμογές που θα χρησιμοποιηθεί το μοντέλο. Αν κοιτάμε το μοντέλο από πολύ κοντά θέλουμε στο σημείο που έχουμε εστιάσει όσο το δυνατόν περισσότερη λεπτομέρεια. Στην αντίθετη περίπτωση όμως που το μοντέλο βρίσκεται αρκετά μακριά, μια απλοποιημένη μορφή του μοντέλου θα ήταν πιο κατάλληλη καθώς με την αύξηση της απόστασης, μειώνεται και το ποσοστό της λεπτομέρειας που μπορούμε να αντιληφθούμε. Με την απλοποίηση των μοντέλων όπου μπορούμε, μειώνουμε την ποσότητα των δεδομένων που θα πρέπει να χειρίζεται ή και να οπτικοποιεί ο υπολογιστής και με την εξοικονόμηση υπολογιστικών πόρων που επιτυγχάνεται, μπορούμε να βελτιώσουμε την ανταπόκριση των προγραμμάτων που χειρίζονται τέτοιες επιφάνειες.

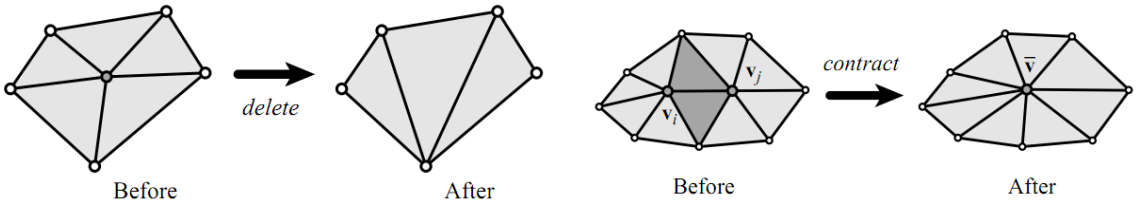
Υπάρχουν αρκετοί αλγόριθμοι για την απλοποίηση μοντέλων, με στόχο τη δημιουργία ενός νέου μοντέλου (συνήθως με την τροποποίηση του αρχικού). Στο νέο αυτό μοντέλο ο αριθμός των κορυφών και των τριγώνων που το αποτελούν γίνεται προσπάθεια να μειωθούν όσο γίνεται περισσότερο αλλά οι διαφορές του από το αρχικό μοντέλο να είναι όσο το δυνατόν μικρότερες.

Οι αλγόριθμοι απλοποίησης μοντέλων ανήκουν σε δύο μεγάλες κατηγορίες ανάλογα με τη στρατηγική του ακολουθούν. Έτσι έχουμε τη στρατηγική καθολικής απλοποίησης με την οποία χειριζόμαστε το μοντέλο στην ολότητά του, και τη στρατηγική της τοπικής απλοποίησης, με την οποία επαναληπτικά εκτελούμε μια ενέργεια τοπικής απλοποίησης του μοντέλου στα διάφορα μέρη του μέχρι αυτό να φτάσει στο επιθυμητό επίπεδο λεπτομέρειας. Οι βασικοί αλγόριθμοι απλοποίησης είναι οι παρακάτω:

Στρατηγική καθολικής απλοποίησης

Ομαδοποίηση κορυφών – Vertex clustering Χωρίζει τον χώρο που καταλαμβάνει το μοντέλο σε τμήματα και όλες οι κορυφές που βρίσκονται στο ίδιο τμήμα ενοποιούνται σε μια τελική κορυφή [LT97].

Προσέγγιση σχήματος – Shape approximation Ξεκινά με ένα πολύ απλό σχήμα στο οποίο προστίθεται λεπτομέρεια προσεγγίζοντας το μοντέλο στο επίπεδο



Σχήμα 2.2: Αριστερά: απαλοιφή κορυφής, Δεξιά: συρρίκνωση ακμής

λεπτομέρειας που θέλουμε [CSAD04].

Στρατηγική τοπικής απλοποίησης

Απλοποίηση διατήρησης εμφάνισης – Appearance preserving simplification Γίνεται απλοποίηση στο μοντέλο προσπαθώντας να διατηρήσει την οπτική απόκλιση στο ελάχιστο, συνδέοντας τα τρίγωνα με τα pixel στα οποία οπτικοποιούνται, και υπολογίζει το σφάλμα απλοποίησης βάσει της επίπτωσης που έχει η κάθε τροποποίηση του μοντέλου στο τελικό αποτέλεσμα, μια υπολογιστικά βαριά διαδικασία [COM98, LT00].

Απαλοιφή κορυφής – Vertex decimation Κάθε φορά αφαιρούμε μία κορυφή και τα τρίγωνα που τη χρησιμοποιούν, και ύστερα κλείνουμε την τρύπα που δημιουργείται με νέα τρίγωνα, λιγότερα απ' όσα χρησιμοποιούνταν πριν (βλ. σχήμα 2.2 (αριστερα)) [SZL+92].

Συρρίκνωση ακμής – Edge contraction Ο πιο χρησιμοποιούμενος τρόπος τοπικής απλοποίησης. Επιλέγεται μια ακμή την οποία «συρρικνώνουμε» και την μετατρέπουμε σε σημείο, διαγράφοντας το ένα ή και τα δυο τρίγωνα που τη χρησιμοποιούσαν (βλ. σχήμα 2.2 (δεξιά))[HDD+93, Gar99].

Σε κάποιες περιπτώσεις, η απαλοιφή κορυφής μπορεί να θεωρηθεί και σαν μια ειδική περίπτωση συρρίκνωσης ακμής. Στο παράδειγμα που φαίνεται στο σχήμα 2.2, η διαγραφή της κεντρικής κορυφής μπορεί να θεωρηθεί και σαν συρρίκνωση της ακμής που ενώνει την κεντρική κορυφή με αυτή που βρίσκεται από κάτω της. Στην γενική περίπτωση όμως κατά την απαλοιφή μιας κορυφής πρέπει να τριγωνοποιήσουμε το κενό που δημιουργείται και το οποίο περικλείεται από ένα αριθμό κορυφών.

Ο αλγόριθμος για τη συρρίκνωση ακμής είναι αρκετά πιο απλός και γρήγορος και συγχρόνως μπορεί να παράγει αρκετά καλά αποτελέσματα απλοποίησης. Αυτό τον αλγόριθμο χρησιμοποιήσαμε σαν βάση για την υλοποίηση που κάναμε. Με περισσότερη λεπτομέρεια ο επαναληπτικός αλγόριθμος συρρίκνωσης ακμής φαίνεται παρακάτω:

Μια συρρίκνωση ακμής που σημειώνεται ως $(\mathrm{v_i}, \mathrm{v_j}) \rightarrow \mathrm{v}$, τροποποιεί την επιφάνεια του μοντέλου σε τρία βήματα:

1. Μετακινούμε τις κορυφές $\mathrm{v_i}$ και $\mathrm{v_j}$ στην τελική θέση v.
2. Αντικαθιστούμε όλες τις εμφανίσεις της $\mathrm{v_j}$ στις κορυφές των τριγώνων με τη $\mathrm{v_i}$.
3. Αφαιρούμε την κορυφή $\mathrm{v_j}$ και όλα τα τρίγωνα που εκφυλίζονται (δύο κορυφές τους ταυτίζονται).

Στο πρώτο βήμα τροποποιούμε την γεωμετρία της επιφάνειας και στο δεύτερο τη συνδεσιμότητα του τριγωνικού πλέγματος. Στο τελευταίο βήμα αφαιρούμε τα στοιχεία που δεν χρειάζονται πια για την αναπαράσταση του μοντέλου.

Στον βασικό αλγόριθμο συρρίκνωσης ακμής υπεισέρχονται δύο βασικές παράμετροι οι οποίες δεν ορίζονται ρητώς και μπορούν να επηρεάσουν το τελικό αποτέλεσμα του αλγορίθμου. Αυτές οι παράμετροι είναι ο τρόπος με τον οποίο επιλέγεται κάθε φορά η ακμή πάνω στην οποία θα εφαρμόσουμε τον αλγόριθμο αλλά και η θέση του τελικού σημείου.

Για την τοποθέτηση του τελικού σημείου κάποιες προφανείς επιλογές είναι να χρησιμοποιήσουμε κάποιο από τα άκρα της ακμής ή το μέσο της, αλλά μπορεί να αποδειχθεί ότι αυτή η στρατηγική δεν παράγει τα βέλτιστα αποτελέσματα. Καλύτερη τοποθέτηση του τελικού σημείου μπορούμε να επιτύχουμε με τη χρήση μιας συνάρτησης που μας δείχνει το σφάλμα απλοποίησης που δημιουργείται κατά τη συρρίκνωση μιας ακμής, και να προσπαθούμε να τοποθετήσουμε την τελική κορυφή σε μια θέση που να ελαχιστοποιεί αυτό το σφάλμα. Ο αλγόριθμος απλοποίησης εξαρτάται από τη συνάρτηση σφάλματος και η σειρά με την οποία συρρικνώνονται οι ακμές εξαρτάται απόλυτα από τη συνάρτηση αυτή, καθώς θέλουμε κάθε φορά να επιλέξουμε την ακμή που θα παράγει το ελάχιστο σφάλμα. Η επιλογή της ακμής που θα συρρικνωθεί γίνεται με την χρήση μιας ουρά προτεραιότητας η οποία περιέχει πληροφορίες για όλες τις δυνατές συρρικνώσεις που μπορούν να γίνουν στο μοντέλο και ενημερώνεται ύστερα από κάθε συρρίκνωση ακμής.

Στη βιβλιογραφία προτείνονται αρκετοί τρόποι μέτρησης του σφάλματος και μέθοδοι ελαχιστοποίησης του. Μια τέτοια μέθοδος που παράγει αρκετά ποιοτικά αποτελέσματα με σχετικά μικρό υπολογιστικό κόστος σε σύγκριση με τις υπόλοιπες είναι η μέθοδος βασισμένη στα «quadric», όπως παρουσιάζεται στο [Gar99].

2.4 Απλοποίηση βασισμένη σε quadrics

Ο αλγόριθμος απλοποίησης με την χρήση quadrics αποτελεί έναν αλγόριθμο βασισμένο στη συρρίκνωση ακμής και με την χρήση μιας μετρικής σφάλματος βασισμένη σε επίπεδα.

2.4.1 Μετρική σφάλματος βασισμένη στα επίπεδα

Η ιδέα μιας τέτοιας μετρικής είναι ο υπολογισμός του σφάλματος που δημιουργείται κατά την απλοποίηση μέσω της τοπικής απόκλισης που έχει επιφάνεια του απλοποιημένου μοντέλου από την αρχική επιφάνειά του. Η τιμή που υπολογίζεται και χρησιμοποιείται σαν σφάλμα είναι το τετράγωνο της απόστασης των κορυφών του απλοποιημένου μοντέλου από την αρχική επιφάνεια. Για να είναι δυνατός ένας τέτοιος υπολογισμός πρέπει σε όλα τα στάδια του αλγορίθμου απλοποίησης να διατηρείται ένα ποσό πληροφορίας η οποία θα συνδέει κάθε κορυφή του απλοποιημένου μοντέλου με την επιφάνεια του αρχικού μοντέλου. Ο τρόπος που χρησιμοποιείται είναι ο εξής: Αρχικά σε κάθε κορυφή του μοντέλου αντιστοιχίζεται ένα σύνολο επιπέδων, τα οποία προσδιορίζονται από τα τρίγωνα στα οποία συμμετέχει η κάθε κορυφή. Κατά τη συρρίκνωση μιας ακμής, το σύνολο των επιπέδων που αντιστοιχούν στο προκύπτον σημείο δημιουργείται από την ένωση των συνόλων των δύο κορυφών της ακμής.

Η βασική εξίσωση αναπαράστασης ενός επιπέδου είναι η $\mathbf{n}^{\mathrm{T}}\mathbf{v} + \mathrm{d} = 0$, όπου $\mathbf{n} =$

$[a, b, c]^T$ είναι ένα κανονικό διάνυσμα (με $a^2 + b^2 + c^2 = 1$) και d ένας αριθμός. Δεδομένης αυτής της εξίσωσης, το τετράγωνο της απόστασης ενός σημείου $\mathbf{v} = [x, y, z]^T$ από το επίπεδο δίνεται από την εξίσωση

$$D^2(\mathbf{v}) = (\mathbf{n}^T\mathbf{v} + d)^2 = (ax + by + cz + d)^2 \tag{2.1}$$

Για κάθε κορυφή v που σχετίζεται με ένα σύνολο επιπέδων P, το σφάλμα ορίζεται στην κορυφή αυτή να είναι ίσο με το άθροισμα των τετραγώνων των αποστάσεών της από όλα τα επίπεδα με τα οποία αυτή σχετίζεται:

$$E_{plane}(\mathbf{v}) = \sum_i D_i^2(\mathbf{v}) = \sum_i (\mathbf{n}_i^T\mathbf{v} + d_i)^2 \tag{2.2}$$

Το πρόβλημα που υπάρχει κατά την χρήση αυτής της μετρικής από τον αλγόριθμο απλοποίησης είναι πως για την διατήρηση των συνόλων των επιπέδων που αντιστοιχούν σε κάθε κορυφή απαιτείται αρκετή μνήμη όπως και ότι ο υπολογισμός του σφάλματος μπορεί να γίνει αρκετά χρονοβόρος, ιδιαίτερα σε προχωρημένο στάδιο απλοποίησης όπου σε κάθε κορυφή μπορούν να αντιστοιχούν αρκετά επίπεδα.

2.4.2 Μετρική σφάλματος με quadrics

Η μετρική σφάλματος βασισμένη στα quadrics παρότι αποτελεί μια μετρική βασισμένη σε επίπεδα διαφοροποιείται αυτή στο ότι αντί να διατηρεί μια λίστα με τα επίπεδα που αντιστοιχούν σε κάθε κορυφή, χρησιμοποιεί έναν πιο συμπαγή τρόπο αναπαράστασης. Μέσω αυτής, ο τρόπος υπολογισμού του σφάλματος είναι αρκετά πιο απλός και η μνήμη που απαιτείται είναι σημαντικά μικρότερη. Αυτά τα χαρακτηριστικά της την κάνουν να είναι αρκετά αποδοτική, ενώ μάλιστα παράγει ποιοτικά αποτελέσματα.

Η βάση αυτής της αναπαράστασης προκύπτει από την εξίσωση της απόστασης ενός σημείου από ένα επίπεδο με τον υπολογισμό της να γίνεται ως εξής:

$$\begin{aligned} D^2(\mathbf{v}) &= (\mathbf{n}^T\mathbf{v} + d)^2 \\ &= (\mathbf{n}^T\mathbf{v} + d)(\mathbf{n}^T\mathbf{v} + d) \\ &= (\mathbf{v}^T\mathbf{n}\mathbf{n}^T\mathbf{v} + 2d\mathbf{n}^T\mathbf{v} + d^2) \\ &= (\mathbf{v}^T(\mathbf{n}\mathbf{n}^T)\mathbf{v} + 2(d\mathbf{n})^T\mathbf{v} + d^2) \end{aligned} \tag{2.3}$$

όπου $\mathbf{n}\mathbf{n}^T$ είναι ο πίνακας που προκύπτει από το εξωτερικό γινόμενο:

$$\mathbf{n}\mathbf{n}^T = \begin{bmatrix} a^2 & ab & ac \\ ba & b^2 & bc \\ ca & cb & c^2 \end{bmatrix}.$$

Έτσι ορίζουμε το quadric Q σαν την τριπλέτα

$$Q = (\mathbf{A}, \mathbf{b}, c)$$

όπου $\mathbf{A} = \mathbf{n}\mathbf{n}^T$ είναι ένας 3x3 πίνακας, $\mathbf{b} = d\mathbf{n}$ είναι ένα διάνυσμα τριών στοιχείων και $c = d^2$ ένας αριθμός. Το quadric Q αντιστοιχίζει μια τιμή $Q(\mathbf{v})$ σε κάθε σημείο $\mathbf{v}$ του χώρου με την εξίσωση δευτέρου βαθμού:

$$Q(\mathbf{v}) = \mathbf{v}^T\mathbf{A}\mathbf{v} + 2\mathbf{b}^T\mathbf{v} + c$$

Τα quadrics[1] μας παρέχουν μια εύχρηστη αναπαράσταση της τετραγωνικής απόστασης $D^2(\mathbf{v})$ ενός σημείου $\mathbf{v}$ από ένα συγκεκριμένο επίπεδο. Για ένα επίπεδο $\mathbf{n}^T\mathbf{v} + d = 0$, ορίζουμε το θεμελιώδες quadric Q να είναι

$$Q = (\mathbf{n}\mathbf{n}^T, d\mathbf{n}, d^2)$$

Με βάση την εξίσωση για το D^2 που δόθηκε νωρίτερα, μπορούμε να δούμε πως η τιμή αυτού του quadric είναι ίση με την τετραγωνική απόσταση $Q(\mathbf{v}) = D^2(\mathbf{v})$ ενός σημείου από ένα επίπεδο. Παρατηρούμε επίσης ότι τα quadrics μπορούν να προστεθούν άμεσα κατά τμήματα: $Q_i(\mathbf{v}) + Q_j(\mathbf{v}) = (Q_i + Q_j)(\mathbf{v})$, όπου $(Q_i + Q_j) = (\mathbf{A}_i + \mathbf{A}_j, \mathbf{b}_i + \mathbf{b}_j, c_i + c_j)$. Έτσι δεδομένου ενός συνόλου από θεμελιώδη quadrics, ορισμένα από ένα σύνολο επιπέδων, το σφάλμα E_Q ορίζεται πλήρως από το άθροισμα των quadrics Q:

$$E_Q(\mathbf{v}) = \sum_i D_i^2(\mathbf{v}) = \sum_i Q_i(\mathbf{v}) = Q(\mathbf{v})$$

όπου $Q = \sum_i Q_i$. Με άλλα λόγια, για να υπολογίσουμε το άθροισμα των τετραγωνικών αποστάσεων για ένα σύνολο επιπέδων, χρειαζόμαστε μόνο ένα quadric το οποίο είναι το άθροισμα των quadrics που ορίζονται από το κάθε επίπεδο στο σύνολο. Όταν γίνεται η συρρίκνωση της ακμής $(\mathbf{v}_i, \mathbf{v}_j)$, το quadric που προκύπτει είναι απλώς $Q = Q_i + Q_j$. Επιπρόσθετα, το κόστος της συρρίκνωσης $(\mathbf{v}_i, \mathbf{v}_j) \to \mathbf{v}$ είναι $Q(\mathbf{v}) = Q_i(\mathbf{v}) + Q_j(\mathbf{v})$.

Ωστόσο τα quadrics που δημιουργούνται κατά την απλοποίηση δεν ταυτίζονται ακριβώς με το σύνολο των επιπέδων που κανονικά θα δημιουργούνταν. Όταν δημιουργούμε την ένωση των δύο συνόλων επιπέδων $P_1 \cup P_2$, τα κοινά επίπεδα εμφανίζονται μόνο μια φορά στο τελικό σύνολο. Όταν όμως προσθέτουμε δύο quadrics $Q_1 + Q_2$, οι επαναλήψεις των επιπέδων παραμένουν στο αποτέλεσμα. Κάθε επίπεδο δύναται να προστεθεί μέχρι και τρείς φορές (μία φορά για κάθε κορυφή του τριγώνου που το ορίζει). Καθώς συρρικνώνονται οι ακμές, μια κορυφή μπορεί να συσσωρεύσει και τις τρείς εμφανίσεις ενός επιπέδου. Αυτή η πολλαπλή εισαγωγή θα μπορούσε να εξαλειφθεί χρησιμοποιώντας τον κανόνα εγκλεισμού-αποκλεισμού (inclusion-exclusion) κατά το άθροισμα των quadrics, κάτι που όμως θα περιέπλεκε έναν κατά τα άλλα αρκετά απλό αλγόριθμο. Σε κάθε περίπτωση η μετρική σφάλματος E_{plane}, η οποία χρησιμοποιεί σύνολα επιπέδων, αποτελεί ουσιαστικά μια ευριστική μέθοδο που χρησιμοποιούμε για την απλοποίηση. Το ότι η E_Q δεν είναι ακριβώς ίση με την E_{plane}, σημαίνει πως οι δύο αυτές ευριστικές μέθοδοι είναι ελαφρώς διαφορετικές.

2.4.3 Ομογενής παραλλαγή

Για τη χρήση ενός quadric Q, μπορούμε να το αντιμετωπίσουμε σαν τον ομογενή πίνακα όπου

$$\mathbf{Q} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^T & c \end{bmatrix}$$

[1] Ο όρος quadric χρησιμοποιείται επειδή οι ισοεπιφάνειες (isosurfaces) $Q(\mathbf{v}) = \varepsilon$ είναι quadric surfaces – επιφάνειες δευτέρου βαθμού οι οποίες περιλαμβάνουν ελλειψοειδείς, παραβολοειδείς, υπερβολοειδείς και επίπεδες επιφάνειες.

Δεδομένου αυτού του πίνακα μπορούμε να υπολογίσουμε το $\mathrm{Q}(\mathbf{v})$ χρησιμοποιώντας την εξίσωση

$$\mathrm{Q}(\mathbf{v}) = \mathbf{v}^{\mathrm{T}}\mathbf{Q}\mathbf{v}, \text{ όπου } \mathbf{v} = [\mathrm{x}, \mathrm{y}, \mathrm{z}, 1]^{\mathrm{T}}$$

Αυτή ή αναπαράσταση των quadrics κάνει εμφανή και άλλη μια ιδιότητά τους. Το μοντέλο μπορεί να μετασχηματισθεί χρησιμοποιώντας γραμμικούς μετασχηματισμούς, μετά τον αρχικό υπολογισμό των quadrics, και αυτά μπορούν να μετασχηματισθούν αντιστοίχως. Έστω $\mathbf{R}$ ένας γραμμικός μετασχηματισμός που εφαρμόζεται στο μοντέλο, και ότι τα quadrics Q έχουν υπολογισθεί πριν την εφαρμογή αυτού του μετασχηματισμού. Για χρησιμοποιήσουμε ένα quadric σε μια μετασχηματισμένη κορυφή, μπορούμε να προβάλλουμε (map) την κορυφή στον προ-μετασχηματισμένο χώρο

$$\mathrm{Q}(\mathbf{v}) = (\mathbf{R}^{-1}\mathbf{v})^{\mathrm{T}}\mathbf{Q}(\mathbf{R}^{-1}\mathbf{v})$$

το οποίο μπορούμε να το θεωρήσουμε ως το:

$$\mathrm{Q}(\mathbf{v}) = \mathbf{v}^{\mathrm{T}}((\mathbf{R}^{-1})^{\mathrm{T}}\mathbf{Q}(\mathbf{R}^{-1}))\mathbf{v}$$

Αυτό μας δίνει τον κανόνα μετασχηματισμού των quadrics

$$\mathbf{Q} \rightarrow (\mathbf{R}^{-1})^{\mathrm{T}}\mathbf{Q}(\mathbf{R}^{-1})$$

ο οποίος είναι ο στάνταρ κανόνας μετασχηματισμού τετραγωνικών επιφανειών.

Αυτός ο τρόπος αναπαράστασης των quadrics χρησιμοποιείται στην υλοποίηση του παράλληλου αλγόριθμου απλοποίησης καθώς το υλικό που στοχεύουμε παρέχει ειδική υποστήριξη για πράξεις μεταξύ πινάκων όπως και για πράξεις μεταξύ πινάκων και διανυσμάτων και έτσι μπορούμε να επιταχύνουμε περιφέρεται την διαδικασία απλοποίησης.

2.4.4 Πολιτικές τοποθέτησης κορυφής

Κατά την συρρίκνωση της ακμής $(\mathbf{v}_{\mathrm{i}}, \mathbf{v}_{\mathrm{j}}) \rightarrow \mathbf{v}$, χρειαζόμαστε κάποιον τρόπο για την επιλογή της τελικής θέσης του $\mathbf{v}$. Υπάρχουν δύο βασικές πολιτικές για την επιλογή της τελικής θέσης της κορυφής από τις οποίες μπορούμε να επιλέξουμε ανάλογα με την εφαρμογή που σχεδιάζουμε.

Η *τοποθέτηση αντικατάστασης* (subset placement) είναι η πιο απλή στρατηγική που μπορούμε να χρησιμοποιήσουμε. Απλώς επιλέγουμε μία από τις δύο κορυφές $\mathbf{v}_{\mathrm{i}}, \mathbf{v}_{\mathrm{j}}$, σαν τελική θέση ανάλογα με το ποιο από τα $\mathrm{Q}(\mathbf{v}_{\mathrm{i}}), \mathrm{Q}(\mathbf{v}_{\mathrm{j}})$ παράγει την μικρότερη τιμή. Με αυτή την πολιτική, όποια προσέγγιση ενός αρχικού μοντέλου παραχθεί, θα χρησιμοποιεί ένα υποσύνολο των κορυφών του, στις αρχικές τους θέσεις.

Συνήθως μπορούμε να παραγάγουμε καλύτερες προσεγγίσεις χρησιμοποιώντας την *βέλτιστη τοποθέτηση* (optimal placement). Η έννοια βέλτιστη αναφέρεται στο πλαίσιο της απλοποίησης με quadrics βασισμένη στην ελαχιστοποίηση του σφάλματος που παράγεται. Δεδομένου ενός quadric $\mathrm{Q} = (\mathbf{A}, \mathbf{b}, \mathrm{c})$, μπορούμε να προσπαθήσουμε να βρούμε το σημείο $\mathrm{v} = (\mathrm{x}, \mathrm{y}, \mathrm{z})$ στο οποίο η τιμή του $\mathrm{Q}(\mathbf{v})$ ελαχιστοποιείται. Καθώς το $\mathrm{Q}(\mathbf{v})$ είναι δευτέρου βαθμού, η εύρεση του ελαχίστου είναι ένα γραμμικό πρόβλημα, με το ελάχιστο να προκύπτει όπου $\frac{\partial \mathrm{Q}}{\partial \mathrm{x}} = \frac{\partial \mathrm{Q}}{\partial \mathrm{y}} = \frac{\partial \mathrm{Q}}{\partial \mathrm{z}} = 0$. Παίρνοντας τις μερικές παραγώγους μπορούμε να δούμε πως $\nabla \mathrm{Q}(\mathbf{v}) = 2\mathbf{A}\mathbf{v} + 2\mathbf{b}$.

Λύνοντας για $\nabla Q(\mathbf{v}) = 0$, βρίσκουμε πως η βέλτιστη θέση είναι η

$$\mathbf{v} = -\mathbf{A}^{-1}\mathbf{b}$$

με σφάλμα

$$Q(\mathbf{v}) = \mathbf{b}^{T}\mathbf{v} + c = -\mathbf{b}^{T}\mathbf{A}^{-1}\mathbf{b} + c$$

Μια μοναδική βέλτιστη μπορεί να μην υπάρχει. Αν ο $\mathbf{A}$ είναι ιδιάζων (singular), ο αντίστροφός του δεν υπάρχει και έτσι να μη μπορούμε να υπολογίσουμε το $\mathbf{v}$ χρησιμοποιώντας τις παραπάνω εξισώσεις. Σε αυτή την περίπτωση, το σύνολο των θέσεων στις οποίες ελαχιστοποιείται το $Q(\mathbf{v})$ σχηματίζει είτε μια γραμμή είτε ένα επίπεδο. Αυτό μπορεί να προκύψει αν για παράδειγμα όλα τα επίπεδα στο Q είναι παράλληλα μεταξύ τους. Σε αυτές τις περιπτώσεις μπορούμε να επιλέξουμε την βέλτιστη θέση πάνω στο ευθύγραμμο τμήμα $(\mathbf{v}_i, \mathbf{v}_j)$ ή αν αυτή δεν μπορεί να υπολογισθεί, το καλύτερο από τα δύο $\mathbf{v}_i, \mathbf{v}_j$ (subset placement).

2.5 Παράλληλη απλοποίηση επιφανειών

Καθώς η GPU είναι ένα εγγενώς παράλληλο σύστημα επεξεργασίας, όπως θα αναπτύξουμε αναλυτικότερα και στο επόμενο κεφάλαιο, θα εξερευνήσουμε και τις παράλληλες υλοποιήσεις για απλοποίηση μοντέλων. Αρχικά θα δώσουμε κάποιες βασικές έννοιες παραλληλισμού και μετά θα προχωρήσουμε στην παρουσίαση παράλληλων υλοποιήσεις απλοποίησης μοντέλων.

2.5.1 Μορφές παραλληλισμού

Υπάρχουν τρεις κύριες μορφές παραλληλισμού [Cro95] οι οποίες διαφοροποιούνται βάσει της οντότητας που κατανέμεται στους επεξεργαστές. Οι τρεις αυτές μορφές είναι: ο παραλληλισμός λειτουργιών, ο χρονικός παραλληλισμός και ο παραλληλισμός δεδομένων. Η επιλογή για το τι θα παραλληλοποιηθεί είναι το πρώτο πρόβλημα που πρέπει να λυθεί κατά τη σχεδίαση ενός παράλληλου αλγόριθμου. Κάθε πρόβλημα έχει μια ισχυρότερη οντότητα που μπορεί να παραλληλοποιηθεί, ή μία στη οποία είναι πιο εύκολο να γίνει σε σύγκριση με τις υπόλοιπες. Επιπλέον, υβριδικές λύσεις μπορούν να συνδυάσουν διάφορες μορφές παραλληλισμού.

Παραλληλισμός λειτουργιών (Functional parallelism)

Η συνολική διεργασία που εφαρμόζεται στα δεδομένα αναπαρίσταται σαν μια ταξινομημένη λίστα διακριτών λειτουργιών. Είναι αρκετό να ανατεθούν αυτές τις λειτουργίες ισόποσα (βάσει του υπολογιστικού κόστους) στις διαθέσιμες μονάδες επεξεργασίας και έτσι να δημιουργηθεί μια εφαρμογή τύπου pipeline. Ουσιαστικά σε κάθε επεξεργαστή ανατίθεται ένα στάδιο του pipeline και όλα μαζί εκτελούνται παράλληλα για να επιταχύνουν την συνολική διαδικασία. Η βασική λειτουργία κάθε σταδίου αποτελείται από μια επαναληπτική διαδικασία η οποία διαβάζει μια ποσότητα δεδομένων (που έχουν γίνει διαθέσιμα από το προηγούμενο ή από τα αρχικά δεδομένα για το πρώτο), τα επεξεργάζεται και τα προωθεί στο επόμενο. Μέσα στο pipeline σε κάθε στιγμή μπορούν να βρίσκονται

δεδομένα σε κάθε στάδιο της διαδικασίας τα οποία όλα επεξεργάζονται παράλληλα από τους αντίστοιχους επεξεργαστές. Ο αριθμός των βημάτων στο pipeline καθορίζει και τον βαθμό παραλληλισμού αυτής της προσέγγισης. Η ταχύτητα του όλου συστήματος καθορίζεται από το βραδύτερο τμήμα, το οποίο μπορεί να προκαλέσει άσκοπη σπατάλη της επεξεργαστικής ισχύς των επεξεργαστών που χειρίζονται τα άλλα τμήματα.

Χρονικός παραλληλισμός (Temporal parallelism)

Αυτή την μορφή παραλληλισμού έχουμε όταν πρέπει να παραχθεί συνεχής αλληλουχία των δεδομένων εξόδου. Τα τελικά τμήματα αντιστοιχίζονται από την εφαρμογή σε διακριτά χρονικά τμήματα (με βασικό παράδειγμα να είναι η αναπαραγωγή βίντεο). Σε αυτή την περίπτωση, ο παραλληλισμός επιτυγχάνεται με την αποσύνθεση του προβλήματος στον πεδίο του χρόνου και σε κάθε επεξεργαστή ανατίθενται τμήματα δεδομένων που αντιστοιχίζονται σε ένα ή περισσότερα χρονικά τμήματα.

Παραλληλισμός δεδομένων (Data parallelism)

Σε αυτή τη μορφή παραλληλισμού, τα δεδομένα χωρίζονται σε ένα αριθμό ρευμάτων, τα οποία ανατίθενται στους επεξεργαστές. Αυτή η μορφή παραλληλισμού μπορεί να κλιμακωθεί πολύ εύκολα για τη χρήση μεγάλου ποσού δεδομένων, με περιοριστικό παράγοντα να είναι ο αριθμός των επεξεργαστών. Βασικής σημασίας είναι η αρχιτεκτονική η οποία χρησιμοποιείται για την τροφοδότηση δεδομένων στους επεξεργαστές, τα χαρακτηριστικά της οποίας μπορεί να επηρεάσουν σημαντικά τις αποφάσεις κατά τον σχεδιασμό της εφαρμογής.

2.5.2 Έννοιες παράλληλων αλγορίθμων

Κάποιοι αλγόριθμοι παραλληλοποιούνται πολύ εύκολα, απαιτώντας πολύ μικρή επικοινωνία μεταξύ των επεξεργαστών ή πρόσθετη επεξεργασία των δεδομένων. Στους περισσότερους αλγόριθμους όμως υπεισέρχονται επιβαρύνσεις που δεν υπάρχουν στη σειριακή μορφή τους. Αυτές οι επιβαρύνσεις προκύπτουν από πολλές πηγές:

- Επικοινωνία μεταξύ των επεξεργαστών
- Άνισα κατανεμημένα ποσά εργασίας
- Περιττοί υπολογισμοί
- Αυξημένες ανάγκες μνήμης για πολλαπλά αντίγραφα των δεδομένων
- Βοηθητικά δεδομένα

Στις περιπτώσεις που οι επιβαρύνσεις είναι αρκετά μεγάλες, θα πρέπει να εξεταστεί και η περίπτωση του ανασχεδιασμού του αλγορίθμου ή και η χρήση ενός τελείως διαφορετικού για τον ίδιο σκοπό, ο οποίος να παραλληλοποιείται πιο εύκολα.

Κατά τη σχεδίαση ενός παράλληλου αλγόριθμου θα πρέπει να προσεχθούν τα παρακάτω βασικά ζητήματα σχεδίασης παράλληλων αλγορίθμων:

Συνάφεια δεδομένων (coherence) Με τη συνάφεια δεδομένων εννοούμε το κατά πόσο γειτονικά (στο χώρο ή και στο χρόνο) δεδομένα να έχουν παρόμοιες ιδιότητες.

Αποσύνθεση λειτουργιών/δεδομένων Ο βασικός σκοπός της αποσύνθεσης του προβλήματος είναι η ισόποση κατανομή του επεξεργαστικού κόστους στους επεξεργαστές. Η κατανομή αυτή όμως έχει επίδραση και στην επικοινωνία των επεξεργαστών.

Διακριτότητα (granularity) Σχετική έννοια με την αποσύνθεση λειτουργιών/δεδομένων, και αναφέρεται στο μέγεθος των λειτουργιών ή των τμημάτων δεδομένων που δημιουργούνται από την αποσύνθεση.

Δυνατότητα κλιμάκωσης Αναφέρεται στην ικανότητα μιας εφαρμογής να προσαρμόζεται σε κάθε μέγεθος προβλήματος και αριθμό επεξεργαστών. Διακρίνονται δύο είδη κλιμάκωσης:

Κλιμάκωση απόδοσης Η ικανότητα να επιτύχουμε μεγαλύτερη απόδοση για ένα πρόβλημα χρησιμοποιώντας περισσότερους επεξεργαστές

Κλιμάκωση δεδομένων Η δυνατότητα να χειριστούμε μεγάλες ποσότητες δεδομένων σε ένα σύστημα.

Ο παραλληλισμός μπορεί να χρησιμοποιηθεί κατά την σχεδίαση αλγορίθμων και ανάλογα με αποφάσεις που πρέπει να πάρουμε, οι οποίες καθοδηγούνται ή και καθορίζονται από την επιλογή του υλικού και την υλοποίηση των προγραμμάτων.

2.5.3 Υλοποιήσεις παράλληλων αλγόριθμων απλοποίησης

Για το πρόβλημα της απλοποίησης μοντέλων δύο είναι οι προφανείς τρόποι παραλληλοποίησης:

Χειρισμός κάθε κορυφής/ακμής/τριγώνου ανεξάρτητα Σε αυτή την προσέγγιση υπάρχουν τα προβλήματα της αλληλεξάρτησης των τμημάτων των δεδομένων που χρησιμοποιούνται και της δημιουργίας μεγάλων ούρων προτεραιότητας, τα οποία μπορούν να περιορίσουν σημαντικά τον σχεδιασμό αλλά και την απόδοση του αλγορίθμου απλοποίησης.

Διάσπαση του μοντέλου σε τμήματα Εδώ υπεισέρχεται το πρόβλημα της τμηματοποίησης του μοντέλου και σύνθεσης των τελικών τμημάτων, με το πρόβλημα της τμηματοποίησης να είναι από μόνο του ένα NP-πλήρες πρόβλημα.

Βεβαίως ανάλογα με την υλοποίηση μπορούν αν χρησιμοποιηθούν και υβριδικές μέθοδοι που σε κάποιο βαθμό (μικρό ή και μεγαλύτερο) συνδυάζουν τις δύο παραπάνω προσεγγίσεις.

Μια προσπάθεια επιτάχυνσης της διαδικασίας απλοποίησης με την χρήση παραλληλισμού είναι αυτή που περιγράφεται στο «GPU Accelerated Shape simplification for Mechanical-Based Applications» [HL07]. Σε αυτήν χρησιμοποιείται ουσιαστικά ένας σειριακός αλγόριθμος απλοποίησης στον οποίο αρχικά υπολογίζονται τα βάρη των κορυφών, και βάσει αυτών ταξινομούνται οι κορυφές του μοντέλου. Ύστερα βρίσκονται περιοχές της επιφάνειας που μπορούν να απλοποιηθούν ανεξάρτητα οι οποίες προωθούνται σε ένα

τμήμα εφαρμογής των απλοποιήσεων στο μοντέλο χρησιμοποιώντας παραλληλία πάνω στις ανεξάρτητες περιοχές που βρέθηκαν.

Μια άλλη υλοποίηση παράλληλου αλγόριθμου απλοποίησης είναι το «Parallel triangular mesh Decimation» [FS00]. Εδώ η βασική επαναληπτική δομή που εκτελείται μέχρι να επιτευχθεί ο στόχος απλοποίησης περιέχει τα παρακάτω:

1. Διάσπαση μοντέλου σε κομμάτια και κατανομή τους στους διαθέσιμους επεξεργαστές
2. Παράλληλος υπολογισμός της σπουδαιότητας της κάθε κορυφής
3. Συλλογή των αποτελεσμάτων και ταξινόμηση των κορυφών με την χρήση Quick Sort
4. Εύρεση ενός συνόλου ανεξάρτητων κορυφών στις οποίες μπορεί να εφαρμοστεί απλοποίηση
5. Διαίρεση του συνόλου σε τόσα τμήματα όσα και οι επεξεργαστές.
6. Ανάθεση ενός τμήματος σε κάθε επεξεργαστή για εφαρμογή των απλοποιήσεων

Σε αυτή την υλοποίηση τα βασικότερα σημεία περιορισμού είναι τα σειριακά τμήματα (η διάσπαση σε κομμάτια του μοντέλου και η ταξινόμηση της λίστας των κορυφών) τα οποία περιορίζουν την δυνατότητα κλιμάκωσης και την παραλληλοποίηση συνολικά του αλγόριθμου απλοποίησης.

Τέλος μια διαδικασία στην οποία εφαρμόζεται παραλληλισμός περιγράφεται στο «Real-time Mesh Simplification Using the GPU» [DT07] στην οποία χρησιμοποιείται ένας πίνακας στον οποίο αποθηκεύονται τα quadrics που υπολογίζονται και μια δομή probabilistic octree με την οποία τεμαχίζεται ο χώρος σε τμήματα. Με παραλληλία πάνω στα τρίγωνα, αυτά προστίθενται στις δύο αυτές δομές και εξάγεται το τελικό απλοποιημένο μοντέλο. Το αποτέλεσμα που παράγεται από αυτόν τον αλγόριθμο εξαρτάται άμεσα από το μέγεθος των δύο δομών που χρησιμοποιούμε. Λόγω της λογικής της καθολικής απλοποίησης με τη χρήση ομαδοποίησης κορυφών σε τμήματα που χώρου, η λεπτομέρεια που θα έχει το τελικό αποτέλεσμα εξαρτάται από το μέγεθος διαμέρισης του χώρου σε τμήματα (μέσω της δομής octree που χρησιμοποιείται).

Στις περισσότερες προσπάθειες επιτάχυνσης της διαδικασίας απλοποίησης με χρησιμοποίηση παραλληλισμού, υπάρχουν ένα ή περισσότερα τμήματα τα οποία θεωρούνται «εγγενώς» σειριακά όπως για παράδειγμα η διάσπαση του μοντέλου σε τμήματα, η ταξινόμηση των κορυφών και η ανάγκη για ένωση των απλοποιημένων τμημάτων που προκύπτουν. Αυτό κυρίως επιβάλλεται από την αρχιτεκτονική του συστήματος για το οποίο σχεδιάζεται ο αλγόριθμος, όπως για παράδειγμα ένα cluster υπολογιστών, ή από τις δομές δεδομένων που χρησιμοποιούνται για την απλοποίηση.

Κεφάλαιο 3

Χρήση του υποσυστήματος γραφικών για υπολογισμούς (GPGPU)

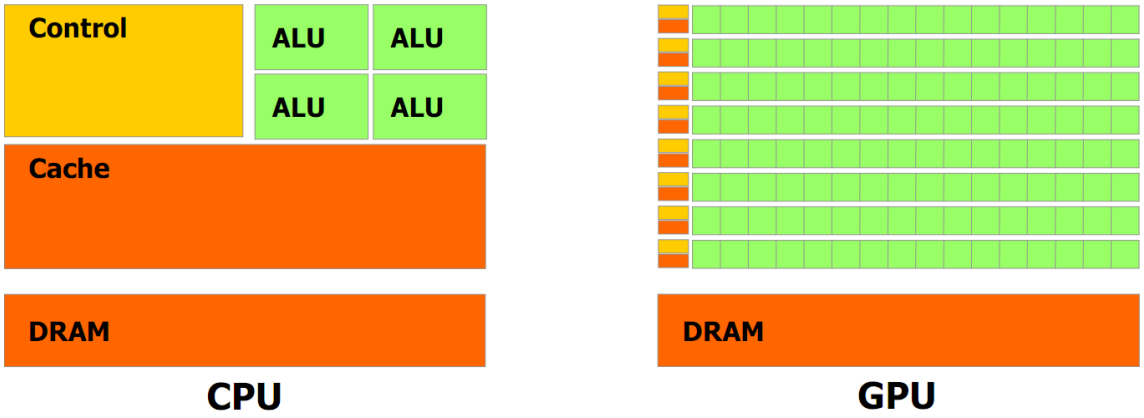
Σε αυτό το κεφάλαιο θα παρουσιαστούν οι λόγοι μου μας οδήγησαν στην χρησιμοποίηση των καρτών γραφικών (Graphics Processing Unit – GPU) για την εκτέλεση του αλγορίθμου απλοποίησης. Αρχικά θα παρουσιάσουμε τις διαφορές που έχουν από τους *κεντρικούς επεξεργαστές* (CPUs) και θα γίνει σύγκριση των της αρχιτεκτονικών τους. Ύστερα θα αναφέρουμε στους τρόπους με τους οποίους μπορεί να χρησιμοποιηθεί η GPU για υπολογισμούς γενικού σκοπού και τις τεχνολογίες που έχουν αναπτυχθει για να εξυπηρετήσουν αυτό τον σκοπό. Τέλος θα παρουσιαστεί η τεχνολογία OpenCL την οποία χρησιμοποιήσαμε για την υλοποίηση του αλγόριθμου απλοποίησης.

3.1 Σύγκριση CPU - GPU

Η CPU είναι το τμήμα του υπολογιστή το οποίο εκτελεί τις εντολές των προγραμμάτων, τις βασικές αριθμητικές και λογικές πράξεις και καθοδηγεί τις λειτουργίες εισόδου/εξόδου του συστήματος. Ο σχεδιασμός και η εξέλιξη των αρχιτεκτονικών που χρησιμοποιούνται για τις CPU γίνεται με τον στόχο ότι θα χρησιμοποιηθούν σαν επεξεργαστές γενικού σκοπού. Η εξέλιξή τους έχει οδηγήσει στην ανάπτυξη πολύπλοκων υπολογιστικών συστημάτων και παράλληλα με αυτά λειτουργικά συστήματα και πλήθους προγραμμάτων που εκμεταλλεύονται τις δυνατότητές τους.

Σε αντίθεση με τις CPUs, οι GPUs δημιουργήθηκαν για να εξυπηρετήσουν ένα συγκεκριμένο σκοπό, την επιτάχυνση της διαδικασίας απόδοσης γραφικών (rendering). Το περιορισμένο εύρος λειτουργιών που σχεδιάζονται να εκτελέσουν σε συνδυασμό με την συνεχή απαίτηση για αύξηση των επιδόσεών τους (καθοδηγούμενο κυρίως από την βιομηχανία ηλεκτρονικών παιχνιδιών) οδήγησε στην δημιουργία επεξεργαστών με αρκετά μεγάλες δυνατότητες υπολογισμών. Ειδικότερα τα τελευταία χρόνια έχουν μια ραγδαία ανάπτυξη στην υπολογιστική ισχύς τους η οποία συνεχώς αυξάνεται σε τέτοιο βαθμό όπου έχει ξεπεράσει αυτή των CPU και το χάσμα μεταξύ τους συνεχώς μεγαλώνει.

Ο λόγος της τόσο μεγάλης επεξεργαστικής ισχύς των GPU είναι η σχετικά απλή αρχι-



Σχήμα 3.1: Διαφορές αρχιτεκτονικής μεταξή CPU και GPU

τεκτονική τους που επιτρέπει επεξεργασία με *παραλληλία δεδομένων* (data-parallel processing). Οι εργασίες για τις οποίες η GPU παραδοσιακά χρησιμοποιείται (π.χ. μετασχηματισμό διανυσμάτων και χρωματισμό pixel) εγγενώς επιτρέπουν παραλληλισμό δεδομένων, με τα τμήματα των δεδομένων να έχουν πολύ λίγες ή και καθόλου εξαρτήσεις μεταξύ τους. Αυτή η ανεξαρτησία των δεδομένων επιτρέπει την παραλληλοποίηση των εργασιών που εκτελούνται στη GPU καθώς τα *νήματα* (threads) μπορούν να λειτουργήσουν ανεξάρτητα το ένα από το άλλο. Σαν αποτέλεσμα είναι εύκολο να αυξηθεί την απόδοση της GPU καθώς μεγαλύτερη απόδοση μπορεί να επιτευχθεί απλώς αυξάνοντας τον αριθμό των νημάτων που μπορούν να εκτελεστούν παράλληλα και αυτό μπορεί να γίνει με το να προσθέσουμε περισσότερους *πυρήνες* (cores) στη GPU.

Η CPU δεν είναι ικανή για το ίδιο μέγεθος παραλληλισμού όπως η GPU καθώς είναι σχεδιασμένη να εκτελεί ένα μεγαλύτερο εύρος εργασιών σε σχέση με τη δεύτερη. Οι εργασίες για τις οποίες είναι σχεδιασμένη η CPU συνήθως δεν γίνεται να έχουν ανεξαρτησία δεδομένων και από τη φύση τους δεν γίνεται να παραλληλοποιηθούν. Η γενικότητα των εργασιών που μπορεί να εκτελέσει η CPU έχει σαν αποτέλεσμα την πολυπλοκότητα της ίδιας και του υλικού που πρέπει να διαχειρίζεται. Επίσης έχει μεγάλα μέρη αφιερωμένα σε *προσωρινές μνήμες* (caches) μειώνοντας τον διαθέσιμο χώρο από *μονάδες υπολογισμού* (ALUs – Arithmetic Logic Units) και *μονάδες υπολογισμών για αριθμούς κινητής υποδιαστολής* (FPUs – Floating-point Units) σε σύγκριση με τη GPU η οποία έχει πολύ περισσότερες μονάδες υπολογισμού. Οι διαφορές αυτές του υλικού φαίνονται διαγραμματικά στο σχήμα 3.1.

Οι GPU επίσης έχουν ξεπεράσει τις CPU στο διαθέσιμο *εύρος ζώνης της μνήμης* (memory bandwidth). Μια τυπική GPU μπορεί να πετύχει ένα εύρος ζώνης γύρω στα 100 GB/s ενώ μια CPU κυμαίνεται γύρω στα 20 GB/s. Αυτή η αύξηση έχει επιτευχθεί με το να χρησιμοποιεί η GPU ένα πιο περιορισμένο μοντέλο μνήμης [Lef05] σε σχέση με τη CPU, η οποία πρέπει να ικανοποιήσει απαιτήσεις των συνήθων λειτουργικών συστημάτων. Στις GPU η δέσμευση και αποδέσμευση μνήμης δεν μπορεί να γίνει κατά την διάρκεια εκτέλεσης υπολογισμών και η αποθήκευση των αποτελεσμάτων (τουλάχιστον κατά την διαδικασία απόδοσης γραφικών) γίνεται σε προϋπολογισμένες διευθύνσεις.

Ένας παράγοντας που περιορίζει την απόδοση της GPU είναι η *καθυστέρηση της μνήμης* (latency). Ακόμα και αν η υπολογιστική ισχύς της GPU έχει αυξηθεί δραματικά τα τελευταία χρόνια, η ταχύτητα των μνημών δεν ακολουθεί αντίστοιχο ρυθμό αύξησης. Η υπολογιστική ισχύς των GPU αυξάνεται κατά 70% και το εύρος ζώνης κατά 25% κάθε

χρόνο ενώ η καθυστέρηση της μνήμης βελτιώνεται κατά 5%. Έτσι είναι σημαντικό να ελαχιστοποιούμε το πλήθος των προσπελάσεων που γίνονται στη μνήμη της GPU και να μεγιστοποιήσουμε τη δουλειά που γίνεται στα φορτωμένα δεδομένα ώστε να εκμεταλλευτούμε πλήρως της ισχύ της GPU.

Λόγω αυτών των διαφορών στην αρχιτεκτονική τους, η CPU και η GPU είναι κατάλληλες για διαφορετικού είδους προβλήματα:

- Η CPU υπερέχει σε προβλήματα που πρέπει να εκτελεστούν σειριακά, όπου ο έλεγχος ροής των διαδικασιών είναι αρκετά περίπλοκος με πολλές διακλαδώσεις και με μικρό μέγεθος δεδομένων.
- Η GPU αποδίδει καλύτερα σε προβλήματα που έχουν μεγάλο μέγεθος δεδομένων στα οποία πρέπει να γίνουν πολλοί αριθμητικοί υπολογισμοί και όπου η επεξεργασία των τμημάτων των δεδομένων είναι μεταξύ τους ανεξάρτητη. Τα προβλήματα που ανατίθενται στη GPU ιδανικά θα πρέπει να έχουν απλή ροή με όσο το δυνατόν λιγότερες διακλαδώσεις στη ροή του κώδικα που θα εκτελέσουν (καθώς κάθε νήμα χρειάζεται το δικό του σύνολο καταχωρητών).

3.2 GPGPU

Η μεγάλη επεξεργαστική ισχύς των GPUs είναι ο λόγος ανάπτυξης του GPGPU (General Purpose computing on Graphics Processing Unit), δηλαδή της χρήσης της GPU για υπολογισμούς γενικού σκοπού, άσχετους με την απόδοση γραφικών στην οθόνη.

Αυτή η χρήση των GPU ξεκίνησε με την εμφάνιση των προγραμματιζόμενων shaders το 2001, χρησιμοποιώντας τις γλώσσες προγραμματισμού για shaders όπως η GLSL και HLSL. Ο αρχικός σκοπός των shaders ήταν να γίνει δυνατή η τροποποίηση του τρόπου επεξεργασίας των γραφικών ώστε δημιουργηθούν οπτικά εφέ που δεν ήταν διαθέσιμα από τις μέχρι τότε υλοποιήσεις των GPU. Η χρήση των shaders για υπολογισμούς άσχετους με γραφικά γινόταν αρχικά με την τοποθέτηση των προς επεξεργασία δεδομένων σε υφές ή σε οποιοδήποτε διαθέσιμο buffer της κάρτας γραφικών (π.χ. z-buffer), να τα επεξεργαζόμαστε με ένα fragment shader και τα αποτελέσματα να αποθηκεύονται είτε σε υφές είτε στους buffers για γραφικά. Τέλος τα αποτελέσματα εξάγονταν από τις υφές ή τα buffers που είναι αποθηκευμένα και μετατρέπονται στην τελική τους επιθυμητή μορφή.

Η χρήση shaders για την επίλυση προβλημάτων γενικού σκοπού στην κάρτα γραφικών ήταν αρκετά δύσχρηστη και περιοριστική. Ο προγραμματισμός τους δεν είναι καθόλου διαισθητικός και εξανάγκασε τους προγραμματιστές να μάθουν να χρησιμοποιούν μια βιβλιοθήκη γραφικών (όπως OpenGL ή Direct3D) και ήταν αναγκασμένοι να χρησιμοποιούν μορφές δεδομένων σχετικές με γραφικά (όπως υφές) για την αποθήκευση των δεδομένων. Από αυτά τα προβλήματα αναδείχθηκε η ανάγκη για τη δημιουργία και χρήση GPGPU frameworks, με σκοπό να δοθεί στους προγραμματιστές ένας πιο άμεσος τρόπο προγραμματισμού της GPU για υπολογισμούς άσχετους με τα γραφικά αλλά και να τους απελευθερώσει από τη χρήση εννοιών βασισμένων στα γραφικά. Αυτά τα framework παρέχουν μια γλώσσα προγραμματισμού με δομές δεδομένων και έλεγχο ροής παρόμοιο με τη C προσθέτοντας διανυσματικές μορφές των απλών τύπων δεδομένων όπως int, float και double.

Ένα από τα πρώτα frameworks που εμφανίστηκαν ήταν το Close To Metal (CTM) [ATI]

της ATI (πλέον AMD) που παρουσιάστηκε το 2006 και παρείχε τρόπους για τον προγραμματισμό των GPU της ίδιας εταιρίας. Η διάρκεια ζωής του CTM ήταν αρκετά μικρή φτάνοντας μέχρι beta εκδόσεις και διάδοχός του είναι το Stream SDK. Η απάντηση της Nvidia στο CTM ήταν το CUDA [NVi10] το οποίο έγινε διαθέσιμο στο κοινό το 2007 και παρείχε μια αντίστοιχη διεπαφή για τον προγραμματισμό των καρτών γραφικών της. Άλλα αντίστοιχα framework είναι το BrookGPU [BFH+04] που αναπτύχθηκε από το πανεπιστήμιο του Stanford και το DirectCompute (μέρος του DirectX) [Boy08] της Microsoft.

3.3 OpenCL

Ένα κοινό πρόβλημα με τα GPGPU frameworks των δύο κυριότερων κατασκευαστών GPU, ATI και NVidia, ήταν ότι απαιτούσαν τη χρήση μιας GPU από την ίδια εταιρία. Αυτό έδωσε κίνητρα για να δημιουργηθεί ένα κοινό framework ανεξάρτητο από το υλικό και έτσι εμφανίστηκε η OpenCL.

Η OpenCL αναπτύχθηκε από την Apple (σε συνεργασία με ομάδες από τις AMD, IBM, Intel και Nvidia) η οποία ήθελε να εκμεταλλευτεί την υπολογιστική δύναμη των GPU στο λειτουργικό της Snow Leopard. Η Apple επέλεξε να δώσει τα δικαιώματα και τη διαχείριση της OpenCL στην ομάδα Khronos (`http://www.khronos.org/`) μετά την ολοκλήρωση της αρχικής πρότασης του framework έτσι ώστε να γίνει ένα ανοιχτό και χωρίς χρηματικές αξιώσεις (royalty-free) πρότυπο. Η ομάδα Khronos διαχειρίζεται και άλλα ανοιχτά πρότυπα όπως για παράδειγμα τα OpenGL, OpenKODE και Collada. Η πρώτη έκδοση έγινε διαθέσιμη το 2008 αλλά πέρασε αρκετός καιρός μέχρι να γίνουν διαθέσιμοι και οι αντίστοιχοι οδηγοί για τις κάρτες γραφικών που να υποστηρίζουν το νέο πρότυπο (περίπου στο τέλος του 2009). Η OpenCL έχει αρκετές ομοιότητες με την CUDA αλλά σε αντίθεση με αυτή, μπορεί να χρησιμοποιηθεί σχεδόν σε όλες τις κάρτες γραφικών των τελευταίων ετών και όχι μόνο σε αυτές της Nvidia. Ο κώδικας που γράφεται για την OpenCL μπορεί επίσης να τρέξει, εκτός από την GPU, και στην CPU ή σε άλλες συσκευές επεξεργασίας (π.χ. FPGA) δεδομένης της ύπαρξης κατάλληλων οδηγών, καθώς και σε συνδυασμούς αυτών ταυτόχρονα.

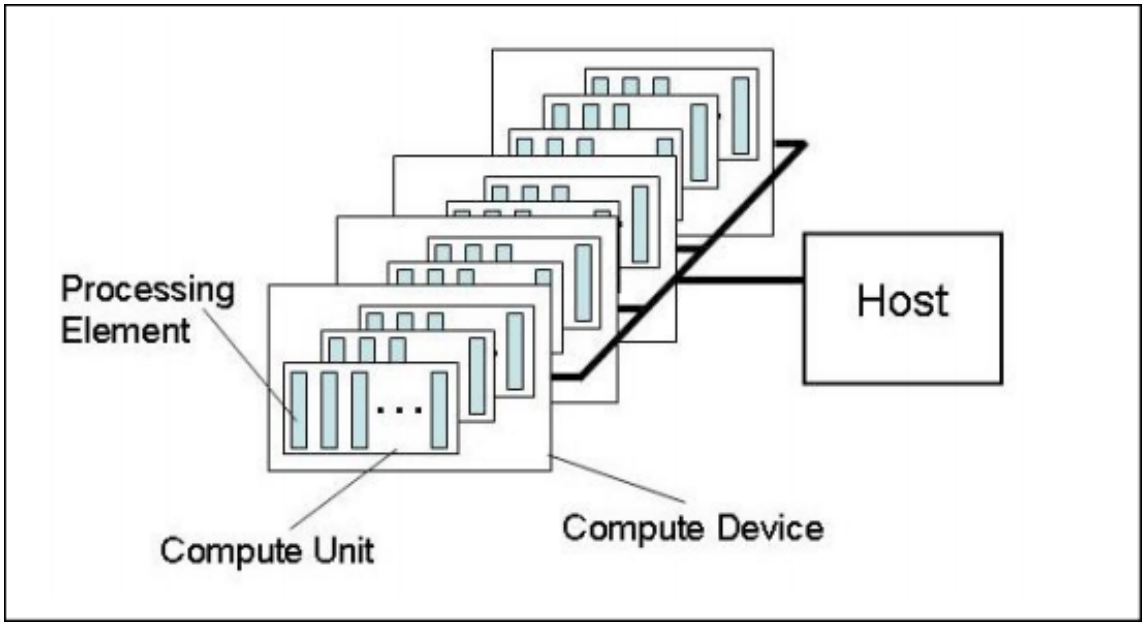
3.3.1 Μοντέλο πλατφόρμας

Η προδιαγραφή της OpenCL [Mun09] ορίζει μια πλατφόρμα στην οποία τρέχουν τα αντίστοιχα προγράμματα. Η πλατφόρμα αποτελείται από έναν υπολογιστή (host) (συνήθως ο υπολογιστής του χρήστη) ο οποίος είναι συνδεμένος με μία ή περισσότερες συσκευές της OpenCL. Μια εφαρμογή που εκτελείται στον host, μέσω της OpenCL καθορίζει ένα υποσύνολο συσκευών που θα χρησιμοποιήσει, από αυτές που είναι συνδεδεμένες με τον host, και υποβάλει αιτήματα εκτέλεσης κώδικα.

Μια συσκευή είναι ένας επεξεργαστής ικανός να εκτελέσει υπολογισμούς κινητής υποδιαστολής και είναι συνήθως μία GPU ή μια CPU αλλά μπορεί να είναι μια οποιαδήποτε συσκευή που είναι ικανή να εκτελέσει αντίστοιχους υπολογισμούς.

Η κάθε συσκευή αποτελείται από μία ή περισσότερες *μονάδες υπολογισμών* (compute units). Μια μονάδα υπολογισμών μπορεί να είναι για παράδειγμα ένας πυρήνας της CPU ή ένας Streaming Multiprocessor (SM)/συσκευή SIMD της GPU.

Μια μονάδα υπολογισμών αποτελείται από ένα ή περισσότερα *στοιχεία επεξεργασίας*



Σχήμα 3.2: Η αρχιτεκτονική της πλατφόρμας της OpenCL

(processing elements). Ένα στοιχείο επεξεργασίας είναι ένας εικονικός *βαθμωτός επεξεργαστής* (scalar processor) που μπορεί για παράδειγμα να είναι μία ALU ή ένας streaming processor. Σε αυτά τα στοιχεία επεξεργασίας εκτελούνται οι υπολογισμοί που δίνονται στην OpenCL από την εφαρμογή.

3.3.2 Μοντέλο εκτέλεσης

Μια εφαρμογή της OpenCL αποτελείται από δύο μέρη: ένα πρόγραμμα που εκτελείται στον host και *προγράμματα πυρήνων* (kernel programs) τα οποία εκτελούνται στις συσκευές που έχουν οριστεί από την εφαρμογή.

Πρόγραμμα στον host

Η εφαρμογή στον host ορίζει το *περιβάλλον* (context) των πυρήνων και διαχειρίζεται την εκτέλεσή τους. Το περιβάλλον που δημιουργεί η εφαρμογή στον host αποτελείται από τα παρακάτω τμήματα:

Συσκευές Ένα σύνολο από συσκευές οι οποίες είναι συνδεμένες με τον host.

Πυρήνες Οι συναρτήσεις που πρόκειται να εκτελεσθούν στις συσκευές.

Αντικείμενα προγραμμάτων Ο πηγαίος κώδικας και τα τελικά μεταγλωττισμένα εκτελέσιμα προγράμματα που υλοποιούν τις συναρτήσεις των πυρήνων.

Αντικείμενα μνήμης Ένας αριθμός από *αντικείμενα μνήμης* (memory objects) τα οποία είναι ορατά από τον host και τις συσκευές που έχουν δεσμευθεί από την εφαρμογή. Αυτά τα αντικείμενα μνήμης περιέχουν τα δεδομένα που μπορούν να χρησιμοποιήσουν οι πυρήνες κατά την εκτέλεσή τους.

Η εφαρμογή του host δημιουργεί και ύστερα διαχειρίζεται το περιβάλλον χρησιμοποιώντας συναρτήσεις που περιέχονται στη βιβλιοθήκη της OpenCL.

Η εφαρμογή του host χρησιμοποιεί μια ουρά εκτέλεσης έτσι ώστε να χειρίζεται την εκτέλεση των συναρτήσεων των πυρήνων στις συσκευές. Είναι δυνατό να έχουμε πολλαπλές ουρές εκτέλεσης για ένα περιβάλλον, αλλά συνήθως χρησιμοποιείται μόνο μία. Όλες οι ουρές εκτέλεσης που σχετίζονται με ένα περιβάλλον εκτελούνται ταυτόχρονα και ανεξάρτητα η μία από την άλλη.

Η εφαρμογή του host εισάγει εντολές στην ουρά εκτέλεσης οι οποίες μπαίνουν σε ένα χρονοδιάγραμμα για εκτέλεση στις συσκευές που βρίσκονται στο περιβάλλον. Υπάρχουν τριών ειδών εντολές που μπορούν να σταλούν στις συσκευές:

Εντολή εκτέλεσης πυρήνα Μια εντολή για να εκτελεστεί ένας συγκεκριμένος πυρήνας.

Εντολή μνήμης Εντολή για την μεταφορά από, προς ή και ανάμεσα στα αντικείμενα μνήμης που βρίσκονται στη μνήμη του host ή των συσκευών ή για τη δέσμευση και αποδέσμευση μνήμης από το χώρο διευθύνσεων του host.

Εντολή συγχρονισμού Εντολή που βάζει περιορισμούς στη σειρά εκτέλεσης των υπόλοιπων εντολών.

Οι εντολές που δίνονται σε μια συσκευή εκτελούνται ασύγχρονα σε σχέση με τον host. Ωστόσο, οι εντολές εκτέλεσης των πυρήνων και διαχείρισης μνήμης δημιουργούν *αντικείμενα γεγονότων* (event objects) που μπορούν να χρησιμοποιηθούν για τον συντονισμό της εκτέλεσης μεταξύ των συσκευών και του host. Γεγονότα επίσης μπορούν να χρησιμοποιηθούν για τον έλεγχο της σειράς με την οποία θα εκτελεσθούν οι εντολές. Αν ο προγραμματιστής δεν ορίσει ρητά τη σειρά εκτέλεσης των εντολών μέσω γεγονότων, αυτή εξαρτάται από δύο διαφορετικές καταστάσεις λειτουργίας:

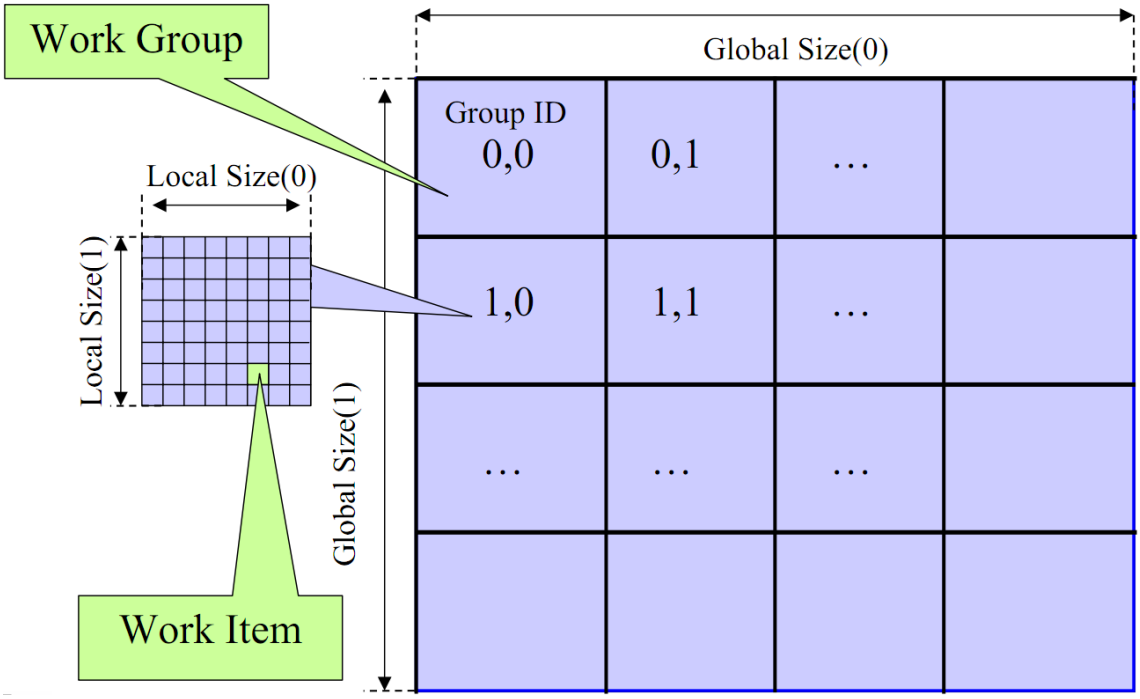
In-order εκτέλεση Οι εντολές εκτελούνται με τη σειρά που εμφανίστηκαν στην ουρά εκτέλεσης και ολοκληρώνονται με την ίδια σειρά. Αυτό σημαίνει ότι η εφαρμογή περιμένει την προηγούμενη εντολή να ολοκληρωθεί πριν δώσει την επόμενη.

Out-of-order εκτέλεση Οι εντολές εκτελούνται με τη σειρά που εμφανίζονται στην ουρά εκτέλεσης, αλλά η εφαρμογή δεν περιμένει την ολοκλήρωσή τους πριν προσθέσει και τις επόμενες εντολές στην ουρά. Οποιοσδήποτε συγχρονισμός μεταξύ των εντολών πρέπει να επιβληθεί από τον προγραμματιστή μέσω των εντολών συγχρονισμού.

Προγράμματα πυρήνων (Kernel programs)

Ένα πρόγραμμα πυρήνα είναι μια συλλογή από συναρτήσεις πυρήνων που εκτελούνται στις συσκευές ενός περιβάλλοντος. Μια συνάρτηση πυρήνα είναι μια συνάρτηση της οποίας εκτελούνται παράλληλα πολλά στιγμιότυπα (τα οποία αποτελούν τα στοιχεία εργασίας) στα στοιχεία επεξεργασίας της συσκευής.

Υπάρχουν δύο κατηγορίες από συναρτήσεις πυρήνα:



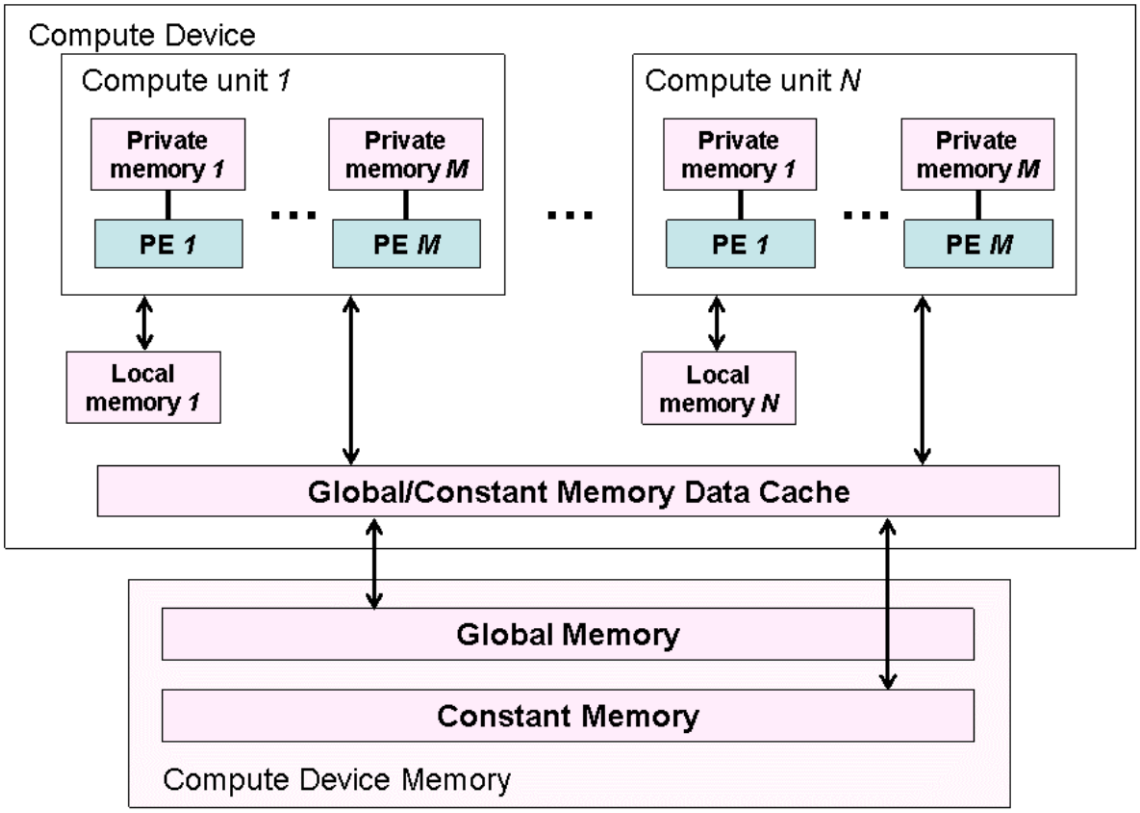
Σχήμα 3.3: Αντιστοίχιση αναγνωριστικών όταν χρησιμοποιείται εύρος δύο διαστάσεων

Πυρήνες της OpenCL Οι πυρήνες της OpenCL είναι γραμμένοι στη γλώσσα OpenCL C και μεταγλωττίζονται από τον αντίστοιχο μεταγλωττιστή της OpenCL. Κάποιες υλοποιήσεις μπορούν επίσης να παρέχουν άλλους τρόπους δημιουργίας των πυρήνων αυτών. Ένας πυρήνας της OpenCL υποστηρίζεται από όλες τις υλοποιήσεις της OpenCL. Αυτή η κατηγορία πυρήνων είναι και η πιο συχνά χρησιμοποιούμενη κατά τον προγραμματισμό σε OpenCL.

Εγγενείς πυρήνες (native kernels) Ένας εγγενής πυρήνας μπορεί να είναι μια συνάρτηση ορισμένη στον κώδικα της εφαρμογής στον host ή μια συνάρτηση που εισάγεται από μια βιβλιοθήκη. Οι εγγενείς πυρήνες γίνονται διαθέσιμοι στην OpenCL μέσω ενός δείκτη προς τη συνάρτηση που θα εκτελεστεί. Μοιράζονται τα αντικείμενα μνήμης με τους πυρήνες της OpenCL και μπαίνουν στην ουρά προς εκτέλεση μαζί τους. Η δυνατότητα εκτέλεσης εγγενών πυρήνων είναι προαιρετικό χαρακτηριστικό της OpenCL και η σημασιολογία τους ορίζεται από την υλοποίηση της OpenCL στις συσκευές. Το API της OpenCL περιέχει συναρτήσεις με τις οποίες μπορούμε να πάρουμε πληροφορίες από τις συσκευές για το αν υποστηρίζουν ή όχι εγγενείς πυρήνες.

Όταν μια συνάρτηση πυρήνα τίθεται προς εκτέλεση (βάζοντας μια εντολή για την εκτέλεση της συνάρτησης πυρήνα στη ουρά εκτέλεσης) από την εφαρμογή στον host, δίνονται μαζί και παράμετροι που ορίζουν το εύρος της συνάρτησης, το μέγεθος των ομάδων στοιχείων εργασίας που θα δημιουργηθούν και αντικείμενα γεγονότων για συγχρονισμό της λειτουργίας με άλλες συναρτήσεις πυρήνα που έχουν προστεθεί πιο πριν.

Το εύρος της συνάρτησης, το οποίο καλείται NDRange, καθορίζει ένα N-διάστατο χώρο ακεραίων, με N ίσο με 1, 2, ή 3. Για κάθε σημείο αυτού του χώρου δημιουργείται ένα στοιχείο εργασίας, οι συντεταγμένες του οποίου αποτελούν το *καθολικό αναγνωριστικό* (Global ID) του στοιχείου εργασίας. Με τον ορισμό του εύρους καθορίζεται ουσιαστικά το



Σχήμα 3.4: Εννοιολογική αρχιτεκτονική μιας συσκευής στην οποία παρουσιάζεται η δομή της μνήμης και τα στοιχεία επεξεργασίας της συσκευής

πλήθος των στοιχείων εργασίας που θα εκτελεστούν. Κάθε στοιχείο εργασίας εκτελεί τον ίδιο κώδικα πυρήνα, αλλά μπορεί να ακολουθήσει διαφορετική ροή στον κώδικα και τα δεδομένα που θα χειριστεί το καθένα μπορεί να διαφέρουν.

Τα στοιχεία εργασίας οργανώνονται σε *ομάδες εργασίας* (work groups) βάσει του μεγέθους ομάδας που έχει δοθεί σαν παράμετρος. Κάθε ομάδα εργασίας έχει το δικό της καθολικό αναγνωριστικό ομοίως με τα στοιχεία εργασίας, για την ταυτοποίησή τους στο εύρος της συνάρτησης. Επίσης τα στοιχεία εργασίας, εκτός από το καθολικό αναγνωριστικό έχουν και τοπικό αναγνωριστικό που δείχνει τη θέση τους μέσα στην ομάδα εργασίας που βρίσκονται (σχήμα 3.3).

Σε κάθε ομάδα εργασίας αντιστοιχίζεται μια μονάδα υπολογισμών και τα στοιχεία εργασίας στην ομάδα εκτελούνται ταυτόχρονα στα στοιχεία επεξεργασίας της μονάδας υπολογισμών. Η μνήμη μιας μονάδας υπολογισμών χρησιμοποιείται από κοινού από τα στοιχεία εργασίας της ομάδας. Τα στοιχεία εργασίας μιας ομάδας μπορούν να συγχρονίσουν την εκτέλεσή τους σε σχέση με τα άλλα στοιχεία εργασίας της ίδιας ομάδας, άλλα όχι με αυτά άλλης ομάδας.

3.3.3 Μοντέλο μνήμης

Υπάρχουν τέσσερις *χώροι διευθύνσεων* (address spaces) στην OpenCL (σχήμα 3.4):

Global Memory – Καθολική μνήμη Ένας χώρος διευθύνσεων που είναι προσβάσιμος

από όλα τα στοιχεία εργασίας όλων των ομάδων εργασίας. Τα στοιχεία εργασίας μπορούν να διαβάσουν και να γράψουν σε όλα τα αντικείμενα μνήμης που βρίσκονται στην καθολική μνήμη.

Constant memory – Σταθερή μνήμη Μια περιοχή της καθολικής μνήμης της οποίας δεν γίνεται να τροποποιηθούν τα περιεχόμενα. Ο host δεσμεύει και αρχικοποιεί τα αντικείμενα μνήμης που θα τοποθετηθούν στην σταθερή μνήμη.

Local Memory – Τοπική μνήμη Η τοπική μνήμη μιας συσκευής υλοποιείται είτε σαν μια μνήμη πάνω στη συσκευή αφιερωμένη γι' αυτό τον σκοπό είτε σαν ένα δεσμευμένο εύρος από τον καθολικό χώρο διευθύνσεων. Χρησιμοποιείται από κοινού από τα στοιχεία εργασίας μιας ομάδας εργασίας και είναι διαθέσιμη για ανάγνωση και εγγραφή. Σε κάποιες αρχιτεκτονικές, η τοπική μνήμη μπορεί να είναι σημαντικά γρηγορότερη από την καθολική μνήμη και σε αυτές τις περιπτώσεις μετακινώντας τα δεδομένα που θα επεξεργαστούμε στην τοπική μνήμη μπορούμε να έχουμε σημαντικά οφέλη από αυτή την ταχύτητα.

Private Memory – Ιδιωτική μνήμη Κάθε στοιχείο εργασίας έχει πρόσβαση σε ένα δικό του χώρο ιδιωτικής μνήμης. Η ιδιωτική μνήμη ενός στοιχείου εργασίας είναι προσβάσιμη μόνο από αυτό και μόνο αυτό μπορεί να διαβάσει και να γράψει σε αυτή.

Το μοντέλο της μνήμης του host και των συσκευών της OpenCL είναι κατά κύριο λόγο ανεξάρτητα. Αυτό προκύπτει από το ότι ο host είναι ορισμένος εκτός της OpenCL. Ωστόσο είναι απαραίτητη η μεταφορά δεδομένων από τη μνήμη του host στη μνήμη των συσκευών και αντίστροφα, κυρίως για την τροφοδότηση των αρχικών δεδομένων και την ανάκτηση των αποτελεσμάτων. Αυτό επιτυγχάνεται με τις εντολές μνήμης που αναφέρθηκαν νωρίτερα, οι οποίες προστίθενται στην ουρά εκτέλεσης. Υπάρχουν εντολές για την μεταφορά από και προς τα αντικείμενα μνήμης που υπάρχουν στη συσκευή όπως επίσης και για δέσμευση και αποδέσμευση περιοχών για τα αντικείμενα μνήμης. Οι εντολές μνήμης μπορεί να είναι blocking ή non-blocking, με τις πρώτες να επιστρέφουν μόλις έχει ολοκληρωθεί η αντίστοιχη λειτουργία και με τις δεύτερες να επιστρέφουν όταν η εντολή προστεθεί στην ουρά εκτέλεσης.

Συνέπεια μνήμης

Η OpenCL χρησιμοποιεί ένα *χαλαρό μοντέλο συνέπειας* (relaxed consistency model) στη διαχείριση μνήμης.

- Το μοντέλο δεν εγγυάται ότι η ορατή μνήμη από ένα στοιχείο εργασίας θα είναι πάντα συνεπής με όλα τα στοιχεία εργασίας.
- Η μνήμη ενός στοιχείου εργασίας έχει συνέπεια εγγραφής/ανάγνωσης.
- Η τοπική μνήμη σε μια ομάδα εργασίας είναι συνεπής μεταξύ των στοιχείων εργασίας της ομάδας στα *φράγματα* (barriers). Τα φράγματα είναι ρητώς ορισμένα σημεία στον κώδικα των πυρήνων στα οποία τα στοιχεία εργασίας μιας ομάδας συγχρονίζονται μεταξύ τους.
- Η καθολική μνήμη είναι συνεπής στα σημεία φράγματος για τα στοιχειά εργασίας μιας ομάδας. Ωστόσο δεν υπάρχουν εγγυήσεις συνέπειας της μνήμης μεταξύ των στοιχείων εργασίας διαφορετικών ομάδων εργασίας.

Η συνέπεια μνήμης, για τα αντικείμενα μνήμης που χρησιμοποιούνται από τις διάφορες εντολές που προστίθενται στην ουρά εκτέλεσης, επιβάλλεται από την OpenCL στα σημεία συγχρονισμού.

Συγχρονισμός

Υπάρχουν δύο τρόποι συγχρονισμού διαθέσιμοι από την OpenCL:

- Συγχρονισμός μεταξύ των στοιχείων μιας ομάδας εργασίας
- Συγχρονισμός μεταξύ των εντολών που προστίθενται στην ουρά εκτέλεσης για ένα περιβάλλον

Ο συγχρονισμός μεταξύ των στοιχείων εργασίας μιας ομάδας εργασίας γίνεται με τη χρήση φραγμάτων. Τα φράγματα συνήθως χρησιμοποιούνται για να επιτύχουμε συνέπεια της μνήμης, αλλά μπορούν να χρησιμοποιηθούν οπουδήποτε ο προγραμματιστής θέλει να συγχρονίσει τα στοιχεία εργασίας μιας ομάδας μεταξύ τους. Ένα φράγμα πρέπει να εκτελεστεί είτε από όλα τα στοιχεία εργασίας ή από κανένα. Αν ένα στοιχείο εργασίας που εκτελεί έναν πυρήνα φτάσει σε ένα σημείο φράγματος, όλα τα στοιχεία εργασίας της ομάδας πρέπει να εκτελέσουν το φράγμα που βρίσκεται στο ίδιο σημείο του κώδικα.

Ο συγχρονισμός μεταξύ των εντολών που προστίθενται στην ουρά εκτέλεσης ενός περιβάλλοντος γίνεται στα *σημεία συγχρονισμού* (synchronization points). Αυτά τα σημεία συγχρονισμού είναι τα:

Φράγματα ουράς εκτέλεσης Ένα φράγμα στην ουρά εκτέλεσης βεβαιώνει ότι όλες οι εντολές που έχουν προστεθεί πριν το φράγμα έχουν ολοκληρωθεί και έχει σταματήσει η τροποποίηση των αντικειμένων μνήμης πριν εκτελεσθούν περαιτέρω εντολές. Αυτός ο τύπος φράγματος μπορεί να χρησιμοποιηθεί για τον συγχρονισμό των εντολών που προστίθενται στην ίδια ουρά εκτέλεσης.

Αναμονή γεγονότων Όλες οι συναρτήσεις της OpenCL που προσθέτουν εντολές στην ουρά εκτέλεσης επιστρέφουν ένα *γεγονός* (event) όταν η εντολή προστεθεί στην ουρά, το οποίο προσδιορίζει την εντολή και τα αντικείμενα μνήμης που αυτή διαχειρίζεται. Κατά την προσθήκη μιας εντολής στην ουρά μπορούν να δοθούν σαν παράμετροι και τέτοια γεγονότα, και στην περίπτωση αυτή η εντολή είναι υποχρεωμένη να περιμένει την ολοκλήρωση αυτών των γεγονότων πριν ξεκινήσει τη λειτουργία της. Με αυτό τον τρόπο διασφαλίζεται ότι η εντολή που προστέθηκε θα συγχρονίσει τη λειτουργία της με αυτές που δημιούργησαν τα αντίστοιχα γεγονότα.

Κεφάλαιο 4

Παράλληλος αλγόριθμος απλοποίησης τριγωνικού πλέγματος για χρήση με την OpenCL

Όπως είδαμε και στην ενότητα 2.5.3, οι περισσότερες υλοποιήσεις για απλοποίηση τριγωνικών μοντέλων με τη χρήση παραλληλισμού έχουν ένα ή περισσότερα σειριακά τμήματα στην κύρια επαναληπτική δομή του αλγόριθμου που αποτρέπουν την πλήρη παράλληλη υλοποίησή του. Στην υλοποίηση που θα παρουσιαστεί σε αυτό το κεφάλαιο, περιγράφεται ένας αλγόριθμος απλοποίησης τριγωνικού πλέγματος βασισμένος σε συρρικνώσεις ακμών ο οποίος χρησιμοποιεί παραλληλισμό δεδομένων, όπως αυτός παρέχεται από την OpenCL, και με τέτοιο τρόπο ώστε να μην υπάρχουν σειριακά τμήματα στην κύρια επαναληπτική δομή του. Ο αλγόριθμος απλοποίησης καθοδηγείται από σφάλμα που υπολογίζεται με τα quadrics (βλ. ενότητα 2.4.2).

4.1 Σχεδιασμός αλγορίθμου

Ο βασικός αλγόριθμος απλοποίησης ο οποίος χρησιμοποιεί συρρικνώσεις ακμών και quadrics είναι αυτός που περιγράφεται στο [Gar99] και από τον τρόπο που έχει σχεδιαστεί είναι εν γένει σειριακός. Η λογική στην οποία έχει βασιστεί είναι ότι σε κάθε επανάληψη του αλγορίθμου εκτελείται μια συρρίκνωση ακμής, αυτή που βάσει του σφάλματος βασισμένο στα quadrics θα επιφέρει την μικρότερη τροποποίηση του μοντέλου. Υπενθυμίζεται ότι, όπως έχει αναφερθεί στην παράγραφο 2.3, ο αλγόριθμος απλοποίησης με συρρικνώσεις ακμών είναι εγγενώς σειριακός καθώς χρησιμοποιεί μια ουρά προτεραιότητας που περιέχει πληροφορίες για όλες τις συρρικνώσεις ώστε να επιλέξει την καταλληλότερη, και η οποία ενημερώνεται ύστερα από κάθε συρρίκνωση που εκτελείται.

Για να μπορέσει να παραλληλοποιηθεί η διαδικασία απλοποίησης χρειάζεται κάθε φορά να εκτελείται όχι μόνο μία συρρίκνωση ακμής αλλά όσες περισσότερες γίνεται διατηρώντας το απλοποιημένο μοντέλο όσο πιο κοντά γίνεται στο αρχικό. Επειδή όμως η κάθε συρρίκνωση ακμής τροποποιεί μια μικρή περιοχή του μοντέλου, δεν είναι δυνατόν να γίνουν δυο συρρικνώσεις ακμών αν οι περιοχές που τροποποιούν αλληλοκαλύπτο-

νται. Επομένως για να επιτευχθεί ο στόχος της παραλληλοποίησης πρέπει να βρεθούν ανεξάρτητες περιοχές του μοντέλου και από τις αντίστοιχες συρρικνώσεις που μπορούν να γίνουν να εκτελέσουμε αυτές που θα επιφέρουν την μικρότερη τροποποίηση στο μοντέλο. Κάθε τέτοια περιοχή του μοντέλου μπορεί να ταυτοποιηθεί μοναδικά από την κεντρική της κορυφή. Αφού βρούμε τις ανεξάρτητες περιοχές, κρατάμε τις αντίστοιχες κεντρικές κορυφές σε μια λίστα την οποία ταξινομούμε βάσει του σφάλματος απλοποίησης και εκτελούμε συρρικνώσεις σε αυτές που έχουν το μικρότερο σφάλμα. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να επιτευχθεί ο στόχος απλοποίησης, τον οποίο στην περίπτωσή μας ορίζουμε σαν τον αριθμό των κορυφών που θέλουμε να έχει το απλοποιημένο μοντέλο.

4.2 Περιγραφή αλγορίθμου

Σε κάθε βήμα αντιμετωπίζουμε το μοντέλο από διαφορετική σκοπιά με τον παραλληλισμό να εφαρμόζεται με βάση τα τρίγωνα, τις κορυφές ή ακόμα και με βάση το πλήθος των επεξεργαστών που παρέχονται. Αυτή η δυνατότητα μας δίνεται από την αρχιτεκτονική της πλατφόρμας που παρέχει η OpenCL και η οποία βασίζεται εν πολλοίς στην αρχιτεκτονική των καρτών γραφικών και περιγράφει ένα σύστημα που έχει τη δυνατότητα εκτέλεσης υπολογισμών με παραλληλισμό δεδομένων.

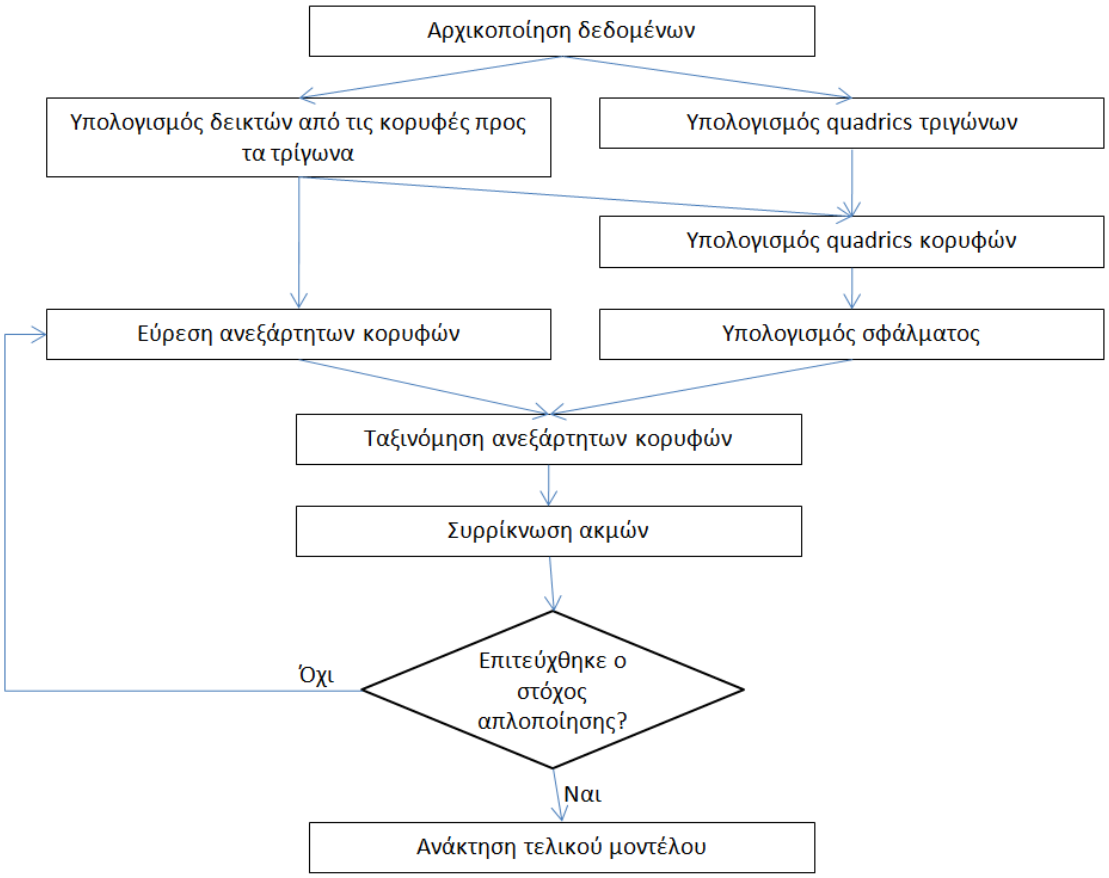
Τα δεδομένα εισόδου του αλγόριθμου είναι το μοντέλο που θέλουμε να απλοποιήσουμε και ο στόχος της απλοποίησης, που είναι ο τελικός αριθμός κορυφών που θέλουμε να έχει το απλοποιημένο μοντέλο. Σαν βασική ενέργεια απλοποίησης που εφαρμόζεται στο μοντέλο χρησιμοποιείται η συρρίκνωση ακμής, καθοδηγούμενη από τη χρήση των quadrics και με τη χρήση του αλγόριθμου βέλτιστης τοποθέτησης για την κορυφή που προκύπτει.

Για την διαδικασία της απλοποίησης χρειάζεται να ξέρουμε τα τρίγωνα στα οποία συμμετέχει η κάθε κορυφή. Αυτή η πληροφορία είναι έμμεσα διαθέσιμη αλλά για να εντοπίσουμε τα ζητούμενα τρίγωνα θα έπρεπε κάθε φορά να διατρέχουμε όλα τα τρίγωνα του μοντέλου. Για την αποφυγή αυτού του κόστους υπολογίζουμε αυτή την πληροφορία στην αρχή του αλγορίθμου και την τοποθετούμε σε μια δομή δεδομένων ώστε να είναι εύκολη η ανάκτηση και η διαχείρισή της στα επόμενα στάδια του αλγορίθμου. Παράλληλα με αυτή τη διαδικασία υπολογίζουμε τα αρχικά quadrics και το σφάλμα για κάθε κορυφή.

΄Υστερα έχουμε τη βασική επαναληπτική δομή του αλγόριθμου, μέχρι να φτάσουμε τον στόχο απλοποίησης, όπου σε κάθε επανάληψη βρίσκουμε ένα σύνολο από ανεξάρτητες κορυφές, τις ταξινομούμε βάσει του σφάλματος που αντιστοιχεί στην καθεμία και εκτελούμε συρρικνώσεις σε ακμές των οποίων το ένα από τα δύο άκρα τους είναι μία από τις κορυφές του συνόλου που υπολογίσαμε. Κατά την εκτέλεση της συρρίκνωσης των ακμών, τροποποιείται κατάλληλα η συνδεσιμότητα του τριγωνικού πλέγματος, σημειώνοντας ως διαγραμμένα όσα τρίγωνα και κορυφές δεν χρειάζονται πια, και υπολογίζονται νέες τιμές για τα quadrics και το αντίστοιχο σφάλμα στις κορυφές που προκύπτουν. Τέλος μόλις φτάσουμε τον στόχο απλοποίησης, ανακτούμε τους πίνακες κορυφών και τριγώνων και με ένα στάδιο «φιλτραρίσματος» παίρνουμε το τελικό απλοποιημένο μοντέλο.

Τα τμήματα που αλγορίθμου και οι εξαρτήσεις που υπάρχουν μεταξύ τους φαίνονται στο σχήμα 4.1.

Οι εξαρτήσεις ανάμεσα στα τμήματα έχουν την έννοια ότι για να εκτελεστεί κάποιο τμήμα πρέπει πρώτα να ολοκληρώσει την λειτουργία του κάποιο προηγούμενο που θα



Σχήμα 4.1: Η δομή του αλγόριθμου απλοποίησης

δημιουργήσει τα δεδομένα ή θα τα τροποποιήσει κατάλληλα ώστε να μπορούμε να τα χειριστούμε. Για παράδειγμα, η διαδικασία εύρεσης των ανεξάρτητων σημείων απαιτεί την ύπαρξη του δισδιάστατου πίνακα δεικτών από τις κορυφές στα τρίγωνα που περιγράψαμε προηγουμένως και έτσι δημιουργείται η αντίστοιχη εξάρτηση. Ομοίως για την ταξινόμηση των ανεξάρτητων σημείων πρέπει να ξέρουμε τα σημεία που θα χρησιμοποιήσουμε και το σφάλμα που αντιστοιχεί στο καθένα, και έτσι το τμήμα της ταξινόμησης εξαρτάται από τα τμήματα της εύρεσης των ανεξάρτητων σημείων και υπολογισμού του σφάλματος που αντιστοιχεί σε κάθε κορυφή. Αυτές οι εξαρτήσεις και η ροή δεδομένων που δημιουργείται, είναι που τελικά συνθέτουν τον συνολικό αλγόριθμο από τα αντίστοιχα τμήματα.

Συνολικά ο αλγόριθμος απλοποίησης αποτελείται από τα παρακάτω βήματα:

- Αρχικοποίηση δομών δεδομένων
- Υπολογισμός δεικτών από τις κορυφές προς τα τρίγωνα
- Υπολογισμός quadrics σε δύο βήματα
  - Υπολογισμός θεμελιώδους quadric για κάθε τρίγωνο
  - Υπολογισμός quadrics για κάθε κορυφή
- Αρχικός υπολογισμός σφάλματος
- Είσοδος στην βασική επαναληπτική δομή

  - Εύρεση ανεξάρτητων κορυφών
  - Ταξινόμηση ανεξάρτητων κορυφών
  - Συρρίκνωση ακμών
- Ανάκτηση απλοποιημένου μοντέλου

4.3 Δομές δεδομένων που χρησιμοποιήθηκαν

Ο τρόπος αποθήκευσης και χρήσης των δεδομένων αποτελεί κρίσιμο παράγοντα για την παραλληλοποίηση του αλγορίθμου. Οι αποφάσεις κατά τον σχεδιασμό του έγιναν έχοντας υπόψιν τα χαρακτηριστικά της OpenCL, της πλατφόρμας στην οποία θα εκτελεστεί. Τα κύρια χαρακτηριστικά που επηρέασαν τον σχεδιασμό είναι:

- Η εκτέλεση υπολογισμών με παραλληλισμό δεδομένων.
- Δεν υπάρχει η δυνατότητα δυναμικής διαχείρισης μνήμης. Όλες οι διαδικασίες εφαρμόζονται πάνω σε πίνακες σταθερού μεγέθους που ορίζονται από το πρόγραμμα στον host.

Οι πίνακες που χρησιμοποιούνται κατά την λειτουργία του αλγόριθμου είναι οι ακόλουθοι:

Πίνακες μοντέλου Περιέχουν τα δεδομένα του μοντέλου και είναι ο πίνακας κορυφών και ο πίνακας τριγώνων (βλ. παράγραφο 4.3.1).

Πίνακας quadrics Ένα quadric για κάθε κορυφή (βλ. παράγραφο 4.3.2).

Πίνακας σφάλματος Το σφάλμα βάσει των quadrics για κάθε κορυφή

Δομή δεικτών από τις κορυφές προς τα τρίγωνα (βλ. παράγραφο 4.3.3)

Πίνακας ανεξάρτητων κορυφών Οι κορυφές που θα χρησιμοποιηθούν σαν βάση για την απλοποίηση

Κατά την λειτουργία του αλγόριθμου χρησιμοποιούνται και οι παρακάτω βοηθητικοί πίνακες:

Πίνακας θεμελιωδών quadrics των τριγώνων Χρησιμοποιείται κατά τον υπολογισμό των αρχικών quadrics των κορυφών.

Πίνακας ένδειξης ότι μια κορυφή χρησιμοποιείται Χρησιμοποιείται κατά την διαδικασία εύρεσης των ανεξάρτητων κορυφών.

Τα δεδομένα που χειρίζεται ο αλγόριθμος τοποθετούνται σε παράλληλους πίνακες με τους δύο πρώτους, με την έννοια ότι τα στοιχεία του καθενός βρίσκονται σε αντιστοιχία με έναν από τους δύο βασικούς πίνακες του μοντέλου. Μια πιο καλά δομημένη σχεδίαση θα ήταν να χρησιμοποιούμε πίνακες δομών ώστε να έχουμε συγκεντρωμένα τα δεδομένα που χρειάζονται για κάθε κορυφή και τρίγωνο. Επιλέχθηκε η χρήση παράλληλων πινάκων καθώς στις υπάρχουσες υλοποιήσεις της OpenCL επιβάλλεται ένα μέγιστο όριο για το μέγεθος που μπορεί να έχει ένας πίνακας, το οποίο θα περιόριζε σημαντικά το μέγεθος των μοντέλων που θα μπορούσαμε να χειριστούμε. Στη συνέχεια θα παρουσιαστούν οι πίνακες δεδομένων που χρησιμοποιούνται κατά την απλοποίηση και η δομή του καθενός.

4.3.1 Πίνακες μοντέλου

Ο αλγόριθμος απλοποίησης χειρίζεται μοντέλα που αναπαριστώνται με την μορφή τριγωνικού πλέγματος. Αυτά τα τριγωνικά πλέγματα στερούνται κάποιας μορφής κανονικότητας στο τρόπο που ενώνονται οι κορυφές για να σχηματίσουν τα τρίγωνα και έτσι δεν είναι πολλές οι υποθέσεις που μπορούμε να κάνουμε. Η απαίτηση που έχουμε από τα μοντέλα που δίνονται προς απλοποίηση είναι να έχουν την ιδιότητα της πολλαπλότητας (βλ. ενότητα 2.2), η οποία ορίζει μια «σωστή» δομή για το μοντέλο και μας διευκολύνει αρκετά κατά τον χειρισμό τους.

Το μοντέλο δίνεται στον αλγόριθμο με την μορφή δύο πινάκων, έναν με $3\mathrm{V}$ αριθμούς κινητής υποδιαστολής (float) όπου V είναι ο αριθμός των κορυφών, και έναν με $3\mathrm{F}$ μη προσημασμένους ακεραίους (unsigned int) όπου F είναι ο αριθμός τριγώνων. Στον πρώτο πίνακα, κάθε τριάδα πραγματικών αριθμών $(3\mathrm{i}, 3\mathrm{i}+1, 3\mathrm{i}+2)$ με $0 \leq \mathrm{i} < \mathrm{V}$ αντιπροσωπεύει τις συντεταγμένες $(\mathrm{x}, \mathrm{y}, \mathrm{z})$ της κορυφής i του μοντέλου. Παρόμοια στον δεύτερο πίνακα, κάθε τριάδα ακεραίων $(3\mathrm{i}, 3\mathrm{i}+1, 3\mathrm{i}+2)$ με $0 \leq \mathrm{i} < \mathrm{F}$ αναπαριστά το τρίγωνο i με κορυφές $(\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3)$, με καθένα από τους τρεις αυτούς ακεραίους να είναι ο δείκτης της αντίστοιχης κορυφής στον πρώτο πίνακα. Το απλοποιημένο μοντέλο που επιστρέφεται έχει επίσης αυτή την δομή.

Κατά την πράξη της συρρίκνωσης ακμής πρέπει με κάποιο τρόπο να διαγράψουμε κορυφές και τρίγωνα από τους πίνακες του μοντέλου. Οι διαγραφές απαιτούν την μετακίνηση μεγάλων ποσοτήτων δεδομένων ή και τη δέσμευση ενός καινούριου πίνακα με παράλληλη αποδέσμευση του υπάρχοντος μετά την μετακίνηση των δεδομένων. Επειδή μια τέτοια διαδικασία είναι αρκετά χρονοβόρα και συνολικά μη αποδοτική, έχουμε ακολουθήσει μια διαφορετική πρακτική όπου αν θέλουμε να διαγράψουμε μια κορυφή, βάζουμε την τιμή FLT_MAX (όπου FLT_MAX είναι η μεγαλύτερη τιμή που μπορεί να αναπαραστήσει ο float) στην πρώτη από τις τρεις θέσεις που της αντιστοιχούν στον πίνακα των κορυφών σαν ένδειξη διαγραφής. Από αυτό το σημείο και ύστερα κατά την εκτέλεση του αλγόριθμου, όταν αντιμετωπίζουμε μια τέτοια κορυφή την αγνοούμε, και αν προκύψει ότι πρέπει να την χρησιμοποιήσουμε, σημαίνει ότι έχει γίνει κάποιο σφάλμα στον αλγόριθμο. Η ίδια πρακτική χρησιμοποιείται και με τα τρίγωνα όπου σαν δείκτης διαγραφής χρησιμοποιείται η τιμή UINT_MAX (η μεγαλύτερη τιμή που μπορεί να αναπαραστήσει ο unsigned int).

4.3.2 Πίνακας των quadrics

Το quadric, όπως έχουμε αναφέρει και στην ενότητα 2.4, είναι μια ποσότητα πληροφορίας η οποία συσχετίζει μια κορυφή με τα επίπεδα-τρίγωνα στα οποία ανήκει αρχικά και με αυτά που συσσωρεύει κατά τη διαδικασία της απλοποίησης. Η αναπαράσταση η οποία θα χρησιμοποιήσουμε στην υλοποίηση είναι η ομογενής παραλλαγή με τη μορφή ενός 4x4 πίνακα για κάθε quadric (βλ. ενότητα 2.4.3). Αυτό σημαίνει πως για κάθε quadric πρέπει να διατηρούμε 16 αριθμούς κινητής υποδιαστολής, αν είναι δυνατόν μορφής double για την ελαχιστοποίηση των σφαλμάτων που υπεισέρχονται στους υπολογισμούς. Σε αυτή την αναπαράσταση υπάρχει κάποιος πλεονασμός δεδομένων καθώς η βασική αναπαράσταση (βλ. ενότητα 2.4.2) απαιτεί μόνο 10 αριθμούς. Προτιμήσαμε την πρώτη καθώς για τον υπολογισμό των quadrics και του σφάλματος γίνονται πράξεις μεταξύ πινάκων και διανυσμάτων και το υλικό της GPU εκτελεί παράλληλα τέτοιες πράξεις.

Ο πίνακας των quadrics είναι ένας ενιαίος πίνακας αποτελούμενος από αριθμούς

double, με κάθε συνεχόμενη 16-άδα να αντιπροσωπεύει ένα quadric. Το quadric i που ξεκινά από τη θέση $16i$ του πίνακα αντιστοιχεί στην κορυφή i του μοντέλου.

Για τον αρχικό υπολογισμό των quadric των κορυφών, χρησιμοποιείται και ένας προσωρινός πίνακας με τα θεμελιώδη quadric που προκύπτουν από τα τρίγωνα. Αυτός ο πίνακας έχει την ίδια δομή με αυτόν των κορυφών, κρατώντας όμως ένα quadric για καθένα από τα τρίγωνα του αρχικού μοντέλου.

4.3.3 Δείκτες από τις κορυφές προς τα τρίγωνα

Σε όλα τα κύρια βήματα του αλγορίθμου, πλην αυτού της ταξινόμησης, χρειαζόμαστε την πληροφορία σε ποια τρίγωνα χρησιμοποιείται η κάθε κορυφή. Αυτή η πληροφορία είναι έμμεσα διαθέσιμη από τον πίνακα των τριγώνων αλλά επειδή τα τριγωνικά πλέγματα, στην γενική περίπτωση, δεν έχουν κάποια μορφή κανονικότητας στη συνδεσιμότητά τους, για να την ανακτήσουμε πρέπει κάθε φορά να διατρέξουμε όλο τον πίνακα των τριγώνων. Για να αποφύγουμε αυτή την επιβάρυνση δημιουργούμε την δομή που περιγράφεται παρακάτω και την χειριζόμαστε κατάλληλα κατά την εκτέλεση του αλγορίθμου απλοποίησης.

Για κάθε κορυφή χρειαζόμαστε ουσιαστικά ένα αριθμό από δείκτες προς τα αντίστοιχα τρίγωνα, δηλαδή μια λίστα ακεραίων. Ο αριθμός των δεικτών που χρειαζόμαστε για κάθε κορυφή εξαρτάται από την συνδεσιμότητα του μοντέλου στην περιοχή που βρίσκεται η κάθε κορυφή αλλά θα πρέπει να μεταβάλλεται δυναμικά ώστε να αντιπροσωπεύει το μοντέλο μετά από κάθε απλοποίηση. Η αρχιτεκτονική της OpenCL όμως δεν δίνει στους πυρήνες τη δυνατότητα δυναμικής διαχείρισης μνήμης και έτσι πρέπει να εργαστούμε πάνω σε σταθερού μεγέθους πίνακες. Για να ξεπεράσουμε αυτά τα προβλήματα η υλοποίηση που έγινε περιλαμβάνει δυο πίνακες, έναν πίνακα δεδομένων στον οποίο κρατάμε τους δείκτες και έναν πίνακα επικεφαλίδων, ο οποίος έχει μια εγγραφή για κάθε κορυφή και μας λέει σε ποιο σημείο του πίνακα δεδομένων βρίσκονται οι δείκτες για μια κορυφή και πόσοι είναι αυτοί. Στον ορισμό της επικεφαλίδας υπάρχει και ένα τρίτο στοιχείο, δείκτης ότι πρέπει να συνεχίσουμε να διαβάζουμε δείκτες από διαφορετικό σημείο του πίνακα. Ο συνδυασμός των τριών δεικτών στις επικεφαλίδες και του πίνακα δεδομένων σχηματίζει ουσιαστικά μια δομή unrolled linked list για κάθε κορυφή του μοντέλου, μιας συνδεδεμένης λίστας όπου κάθε κόμβος μπορεί να περιέχει παραπάνω από μια τιμές. Η ύπαρξη μιας τέτοιας δομής μας δίνει κάποιο βαθμό δυναμικής διαχείρισης της μνήμης πάνω σε πίνακες σταθερού μεγέθους και η ανάγκη ύπαρξης της θα φανεί κατά την ανάλυση της διαδικασίας συρρίκνωσης ακμών (ενότητα 4.4.5).

```
struct arrayInfo
{
    unsigned int position;
    unsigned int size;
    unsigned int continuesTo;
};
```

Σχήμα 4.2: Ορισμός δομής επικεφαλίδας

4.4 Ανάλυση κεντρικών σημείων του αλγορίθμου

Στην αρχή του κεφαλαίου περιγράψαμε σύντομα τα τμήματα του αλγορίθμου απλοποίησης με τη σειρά που εκτελούνται. Όμως για να επιταχυνθεί η διαδικασία της απλοποίησης, το κάθε τμήμα εσωτερικά χρησιμοποιεί παραλληλισμό δεδομένων. Για να μπορέσουμε να χρησιμοποιήσουμε στο μέγιστο αυτή τη μορφή παραλληλισμού, πρέπει τα τμήματα των δεδομένων που επεξεργάζονται να είναι όσο το δυνατόν πιο μικρά και συγχρόνως ανεξάρτητα μεταξύ τους. Για να επιτευχθεί αυτός ο στόχος, το κάθε τμήμα σχεδιάστηκε ξεχωριστά από τα υπόλοιπα, έχοντας υπόψη την γενικότερη δομή της διαδικασίας απλοποίησης, καθώς οι λειτουργίες που εκτελούν είναι εντελώς διαφορετικές, και ο παραλληλισμός εφαρμόστηκε με τρόπο κατάλληλο για το κάθε τμήμα. Σε αυτή την ενότητα θα περιγράψουμε τη λειτουργία των τμημάτων και τον τρόπο που εφαρμόζεται ο παραλληλισμός στο καθένα.

4.4.1 Υπολογισμός δεικτών από τις κορυφές προς τα τρίγωνα

Ο σκοπός αυτού του τμήματος είναι η δέσμευση κατάλληλου ποσού μνήμης και η αρχικοποίηση της μνήμης αυτής για την δομή των δεικτών από τις κορυφές προς τα τρίγωνα που περιγράφηκε στην ενότητα 4.3.3.

Η μνήμη που απαιτείται για τον πίνακα των επικεφαλίδων είναι ίση με το μέγεθος της δομής επικεφαλίδας επί τον αριθμό των κορυφών. Για τον πίνακα των δεδομένων, όμως, δεν μπορούμε να κάνουμε κάποια υπόθεση καθώς το απαιτούμενο ποσό μνήμης εξαρτάται από τη συνδεσιμότητα του τριγωνικού πλέγματος. Ο αριθμός των τριγώνων στα οποία χρησιμοποιείται μια κορυφή μπορεί να διαφέρει σημαντικά από μοντέλο σε μοντέλο, ή ακόμα και σε διαφορετικές περιοχές της επιφάνειας ενός μοντέλου. Το μέγεθος της μνήμης που θα χρησιμοποιήσουμε για τον πίνακα δεδομένων υπολογίζεται σε ένα στάδιο προεπεξεργασίας του μοντέλου, στο οποίο διατρέχουμε τον πίνακα των τριγώνων και βρίσκουμε τον μέγιστο αριθμό τριγώνων που εμφανίζεται μια κορυφή. Έτσι η μνήμη που δεσμεύουμε ισούται με τον αριθμό των κορυφών επί αυτό το μέγιστο που υπολογίσαμε. Με αυτό τον υπολογισμό δημιουργείται εννοιολογικά ένας δισδιάστατος πίνακας από τον μονοδιάστατο πίνακα που δεσμεύεται, όπου σε κάθε κορυφή αντιστοιχεί μια γραμμή του δισδιάστατου πίνακα, αρκετά μεγάλη ώστε να χωρέσει όλους τους δείκτες προς τα τρίγωνα που χρησιμοποιούν την συγκεκριμένη κορυφή.

Για να υπολογίσουμε τους δείκτες προς τα τρίγωνα, εργαζόμαστε με παραλληλισμό πάνω στον πίνακα των τριγώνων. Για κάθε κορυφή κάθε τριγώνου, αυξάνουμε κατά ένα (1) τον μετρητή των δεικτών για τη συγκεκριμένη κορυφή χρησιμοποιώντας ατομικές πράξεις, οι οποίες επιστρέφουν την τιμή του μετρητή πριν την αύξησή του, και χρησιμοποιούμε αυτή την τιμή μαζί με την θέση στην οποία ξεκινούν οι δείκτες για την συγκεκριμένη κορυφή στον πίνακα δεδομένων για να αποθηκεύσουμε τον ακέραιο που δείχνει στο τρέχον τρίγωνο. Έτσι μόλις διατρέξουμε όλα τα τρίγωνα, η δομή των δεικτών από τις κορυφές προς τα τρίγωνα θα αντιπροσωπεύει πλήρως το μοντέλο.

4.4.2 Υπολογισμός αρχικών quadrics και σφάλματος

Όπως είδαμε και στην παράγραφο 2.4.2, το quadric που αντιστοιχεί σε κάθε κορυφή είναι μια ποσότητα δεδομένων που εξαρτάται από τα τρίγωνα στα οποία αυτή συμμετέχει,

και είναι ίση με το άθροισμα των θεμελιωδών quadric που προκύπτουν από τα αντίστοιχα τρίγωνα. Έτσι για να γίνει αποδοτικά ο υπολογισμός τους, έχουμε διασπάσει τη διαδικασία σε δύο βήματα. Στο πρώτο βήμα υπολογίζεται το θεμελιώδες quadric που αντιστοιχεί σε κάθε τρίγωνο και το αποτέλεσμα του υπολογισμού τοποθετείται σε έναν ενδιάμεσο πίνακα, και στο δεύτερο βήμα χρησιμοποιείται ο πίνακας αυτός και η δομή δεικτών από τις κορυφές προς τα τρίγωνα για να υπολογισθεί το quadric που αντιστοιχεί σε κάθε κορυφή.

Στο πρώτο βήμα χρησιμοποιείται παραλληλισμός δεδομένων βάσει των τριγώνων, όπου καθένα εξετάζεται ανεξάρτητα από τα υπόλοιπα ώστε να γίνει ο υπολογισμός του θεμελιώδους quadric. Ο ενδιάμεσος πίνακας που χρησιμοποιείται αποτελείται από τόσες εγγραφές όσα και τα τρίγωνα του μοντέλου, και το αποτέλεσμα για το τρίγωνο i τοποθετείται στην αντίστοιχη θέση i του ενδιάμεσου πίνακα.

Στο δεύτερο βήμα χρησιμοποιείται παραλληλισμός δεδομένων βάσει των κορυφών του μοντέλου. Χρησιμοποιώντας την δομή δεικτών από τις κορυφές προς τα τρίγωνα που αυτές συμμετέχουν, η οποία έχει υπολογισθεί νωρίτερα, και τον ενδιάμεσο πίνακα από το προηγούμενο βήμα, υπολογίζουμε το quadric που αντιστοιχεί σε κάθε κορυφή με το να διατρέχουμε τους δείκτες και να αθροίζουμε τα θεμελιώδη quadric.

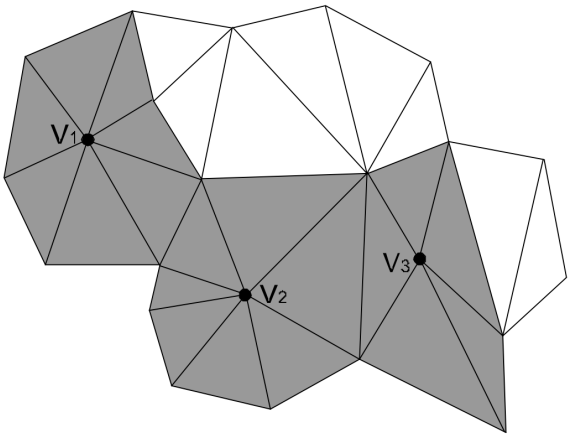
Μέσω αυτής της διαδικασίας των δύο βημάτων προκύπτει ένας πίνακας V θέσεων, με κάθε θέση i να περιέχει το quadric της αντίστοιχης κορυφής. Από αυτό τον πίνακα και με την χρήση της εξίσωσης $\mathrm{E_Q}(\mathbf{v})$ της ενότητας 2.4.2, υπολογίζουμε με παραλληλισμό βάσει των κορυφών, το σφάλμα που αντιστοιχεί σε κάθε μία. Το αποτέλεσμα αυτού του υπολογισμού τοποθετείται σε έναν πίνακα V θέσεων, μία για κάθε κορυφή.

Η διατήρηση ενός πίνακα που περιέχει το σφάλμα για κάθε κορυφή κρίθηκε απαραίτητη παρότι ο υπολογισμός τους είναι σχετικά άμεσος, καθότι αυτές οι τιμές είναι που καθοδηγούν την όλη διαδικασία της απλοποίησης και σε διαφορετική περίπτωση θα χρειαζόταν ο συνεχής επανυπολογισμός τους.

4.4.3 Εύρεση ανεξάρτητων κορυφών

Για να προχωρήσουμε στην απλοποίηση πρέπει να βρούμε περιοχές του μοντέλου τις οποίες κατά το βήμα της απλοποίησης να μπορούμε να χειριστούμε ανεξάρτητα, ώστε να μπορέσει να παραλληλοποιηθεί η διαδικασία. Στον αλγόριθμο που αναπτύξαμε κάθε τέτοια περιοχή προσδιορίζεται από μια κορυφή του μοντέλου (την κεντρική κορυφή της περιοχής) και όλες τις κορυφές που την περιβάλλουν. Για να προσδιορίσουμε τις ανεξάρτητες περιοχές κρατάμε μόνο την κεντρική κορυφή της κάθε μίας, καθώς με βάση αυτή και χρησιμοποιώντας τον πίνακα των τριγώνων και τους δείκτες από τις κορυφές προς τα τρίγωνα, μπορούμε να την προσδιορίσουμε μοναδικά. Για να βρούμε τις ανεξάρτητες περιοχές που αναζητούμε αντιμετωπίζουμε το μοντέλο σαν γράφο με κορυφές τις κορυφές του μοντέλου και ακμές που αντιστοιχούν στα τρίγωνα του μοντέλου. Οι κορυφές που ζητάμε αποτελούν μια μορφή *ανεξάρτητου συνόλου κορυφών* πάνω σε αυτό τον γράφο.

Από την θεωρία των γράφων γνωρίζουμε ότι η εύρεση του *μέγιστου ανεξάρτητου συνόλου* (maximum independent set) είναι NP-πλήρες πρόβλημα και δύσκολο να γίνει κάποια προσέγγισή του. Η εύρεση ενός *μεγιστικού ανεξάρτητου συνόλου* (maximal independent set), δηλαδή ενός συνόλου ανεξάρτητων κορυφών που δεν είναι υποσύνολο κάποιου άλλου συνόλου ανεξάρτητων κορυφών, δεν έχει μοναδική λύση. Ένα τέτοιο σύνολο μπορεί να βρεθεί σε πολυωνυμικό χρόνο χρησιμοποιώντας έναν άπληστο σειριακό αλγόριθμο και βοηθητική μνήμη.



Σχήμα 4.3: Οι κορυφές v_1, v_2, v_3 είναι ανεξάρτητες. Οι v_1 και v_3 είναι υπέρ-ανεξάρτητες.

Τα ανεξάρτητα σύνολα που ορίζονται από τη θεωρία των γράφων ορίζονται τέτοια ώστε οι κορυφές να έχουν απόσταση τουλάχιστον 2. Όμως οι κορυφές που ζητάμε εμείς απαιτούμε να έχουν απόσταση τουλάχιστον 3, που αλλιώς ονομάζονται *υπέρ-ανεξάρτητες* (super independent) [FS00] (σχήμα 4.3). Η απαίτηση για απόσταση 3 προκύπτει από το ότι η κεντρική κορυφή κάθε περιοχής θα είναι η μία από τις δύο κορυφές της ακμής πάνω στην οποία θα εφαρμοστεί η συρρίκνωση. Έτσι βεβαιώνεται ότι δυο ακμές που θα συρρικνωθούν δεν θα έχουν κάποια κοινή κορυφή αλλά και η απόσταση τους θα είναι τέτοια ώστε η τροποποίηση της συνδεσιμότητας του μοντέλου να είναι ανεξάρτητη μεταξύ δύο γειτονικών περιοχών.

Για την επίλυση αυτού του προβλήματος εξετάστηκαν τρεις αλγόριθμοι, ένας σειριακός και δύο παράλληλοι οι οποίοι παρουσιάζονται ακολούθως:

Σειριακός αλγόριθμος

Η λογική του σειριακού αλγόριθμου είναι αρκετά απλή. Διατρέχει σειριακά τις κορυφές και για κάθε μία εξετάζει όλες τις γειτονικές της. Αν η τρέχουσα κορυφή ή κάποια από τις γειτονικές της έχει σημειωθεί ότι χρησιμοποιείται, την αγνοούμε και συνεχίζουμε με την επόμενη. Αν περάσει τον αρχικό έλεγχο, σημειώνουμε την τρέχουσα κορυφή και όλες τις γειτονικές της στον βοηθητικό πίνακα ότι χρησιμοποιούνται, και προσθέτουμε την τρέχουσα κορυφή στον πίνακα ανεξάρτητων κορυφών.

Αυτός ο αλγόριθμος, αν και μας δίνει τα αποτελέσματα που ζητάμε, είναι σειριακός και έτσι μόνο ένας από τους διαθέσιμους επεξεργαστές χρησιμοποιείται. Θέλουμε να χρησιμοποιήσουμε όλους τους διαθέσιμους επεξεργαστές της κάρτας γραφικών για να επιταχύνουμε την διαδικασία το δυνατόν περισσότερο καθώς η εύρεση ανεξάρτητων σημείων είναι τμήμα που εκτελείται σε κάθε επανάληψη του αλγόριθμου απλοποίησης.

Ευριστικός παράλληλος αλγόριθμος βασισμένος στο σφάλμα απλοποίησης

Η επόμενη προσπάθεια που έγινε ήταν να χρησιμοποιήσουμε τα δεδομένα του πίνακα σφάλματος, ώστε να δημιουργηθεί μια ευριστική διαδικασία που βρίσκει ικανό αριθμό ανε-

ξάρτητων κορυφών. Η λογική του αλγόριθμου είναι να βρούμε τα τοπικά ελάχιστα έτσι ώστε όχι μόνο να βρούμε έναν αριθμό από ανεξάρτητες κορυφές αλλά αυτές να είναι και τοπικά βέλτιστες ως προς την διαδικασία απλοποίησης.

Ο αλγόριθμος χρησιμοποιεί παραλληλισμό δεδομένων βάσει των κορυφών του μοντέλου. Για κάθε κορυφή εξετάζει όλες τις κορυφές σε απόσταση μέχρι 2 συγκρίνοντας το σφάλμα της κεντρικής με αυτό των υπολοίπων. Αν η κεντρική κορυφή έχει το ελάχιστο σφάλμα, προστίθεται στον πίνακα των ανεξάρτητων κορυφών. Με τον έλεγχο για απόσταση μέχρι και 2 εξασφαλίζουμε ότι δύο ανεξάρτητες κορυφές θα έχουν απόσταση τουλάχιστον 3, καθώς αν ήταν μικρότερη θα ήταν είτε κεντρική κορυφή ή κάποια από αυτές της γειτονιάς απόστασης 2 όπου θα απορρίπταμε.

Το πρόβλημα με αυτόν τον αλγόριθμο είναι ότι οι κορυφές που θα βρεθούν εξαρτώνται απόλυτα από την κατανομή των σφαλμάτων. Αν σε κάποια περιοχή του μοντέλου δεν υπάρχει κάποιο τοπικό ελάχιστο, δεν θα επιλεγεί κάποια κορυφή, παρότι θα μπορούσαμε να βρούμε ανεξάρτητες κορυφές. Σε ακραίες περιπτώσεις και με κατάλληλη κατανομή των σφαλμάτων στις κορυφές, είναι δυνατό να βρούμε πολύ λίγες μέχρι ακόμα και μόνο μία κορυφή, ανεξάρτητα από τον αριθμό των κορυφών του μοντέλου. Στον κώδικα, για να αντιμετωπιστεί αυτό το φαινόμενο έχουμε εισαγάγει ειδικό έλεγχο αν επιστραφεί αρκετά μικρός αριθμός κορυφών, να εκτελούμε τον σειριακό αλγόριθμο. Αυτό όμως δεν αποτελεί λύση στο πρόβλημα, το οποίο εμφανίστηκε στην πράξη σε αρκετές από τις δοκιμές που κάναμε, και έτσι οδηγηθήκαμε στην ανάπτυξη του τρίτου αλγόριθμου.

Παράλληλος αλγόριθμος εύρεσης ανεξάρτητων κορυφών

Ο τρίτος και τελικός αλγόριθμος που υλοποιήσαμε λαμβάνει υπ' όψιν του το ότι θα εκτελεστεί σε ένα πολυεπεξεργαστικό σύστημα που προσφέρει παραλληλισμό δεδομένων, και ότι όλα τα δεδομένα είναι διαθέσιμα ανά πάσα στιγμή.

Ο αλγόριθμος εκτελείται σε τρία στάδια. Στο πρώτο αρχικοποιούμε τον βοηθητικό πίνακα με παραλληλισμό στα στοιχεία του. Στο δεύτερο στάδιο, χωρίζουμε τον πίνακα κορυφών του μοντέλου σε τμήματα ίσου μεγέθους, τόσα ώστε να καλύψουμε τους διαθέσιμους επεξεργαστές, και παράλληλα σε κάθε τμήμα εκτελούμε τον σειριακό αλγόριθμο με την διαφορά ότι για όλα τα στιγμιότυπά του υπάρχει ένας ενιαίος πίνακας (ο βοηθητικός πίνακας που μόλις αναφέραμε) για τον έλεγχο αν κάποια κορυφή χρησιμοποιείται. Για κάθε κορυφή εξετάζουμε σε απόσταση 2 αν κάποια από τις κορυφές χρησιμοποιείται ήδη και αν όχι, σημειώνουμε την κορυφή ότι χρησιμοποιείται. Τέλος στο τρίτο στάδιο γίνεται η συλλογή των αποτελεσμάτων που υπολογίστηκαν. Με παραλληλισμό πάνω στα στοιχεία του βοηθητικού πίνακα (που καθένα αντιστοιχεί σε μια κορυφή) ελέγχουμε αν μια κορυφή έχει σημειωθεί πως χρησιμοποιείται και αν είναι έτσι ελέγχουμε πάλι ότι δεν χρησιμοποιείται κάποια κορυφή σε απόσταση 2. Αν περάσουμε επιτυχώς τους ελέγχους, την προσθέτουμε στον πίνακα των ανεξάρτητων κορυφών. Ο επανέλεγχος των κορυφών στο τελευταίο στάδιο γίνεται για να αποφύγουμε την προσθήκη στο αποτέλεσμα κορυφών που βρίσκονται στα όρια των τμημάτων του μοντέλου και λόγω *συνθηκών ανταγωνισμού* (race conditions) μεταξύ των νημάτων εκτέλεσης έχουν σημειωθεί ότι χρησιμοποιούνται, ενώ δεν θα έπρεπε.

4.4.4 Ταξινόμηση κορυφών

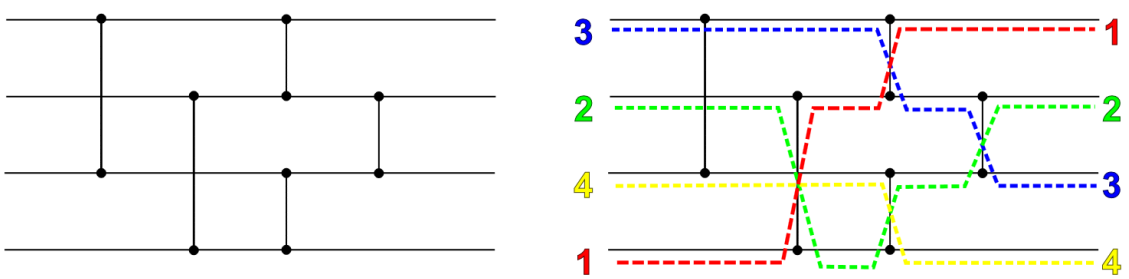
Το επόμενο βήμα του αλγόριθμου απλοποίησης αφορά την ταξινόμηση των ανεξάρτητων κορυφών που έχουν βρεθεί, βάσει του σφάλματος που αντιστοιχεί στην καθεμιά. Στους αλγόριθμους απλοποίησης που υπάρχουν στην βιβλιογραφία αυτό θεωρείται ως ένα εγγενώς σειριακό βήμα που εκτελείται από την CPU. Μια τέτοια υλοποίηση όμως θα απαιτούσε μεταφορά των δεδομένων από το σύστημα της OpenCL στον επεξεργαστή και πάλι πίσω με τις επιπτώσεις στο χρόνο εκτέλεσης και τις απαιτήσεις μνήμης που επιφέρει αυτή. Για να αποφύγουμε αυτό το κόστος όπως επίσης και για να παραλληλοποιήσουμε και να επιταχύνουμε την διαδικασία ταξινόμησης χρησιμοποιούμε την έννοια του δικτύου ταξινόμησης [KR88].

Δίκτυα ταξινόμησης

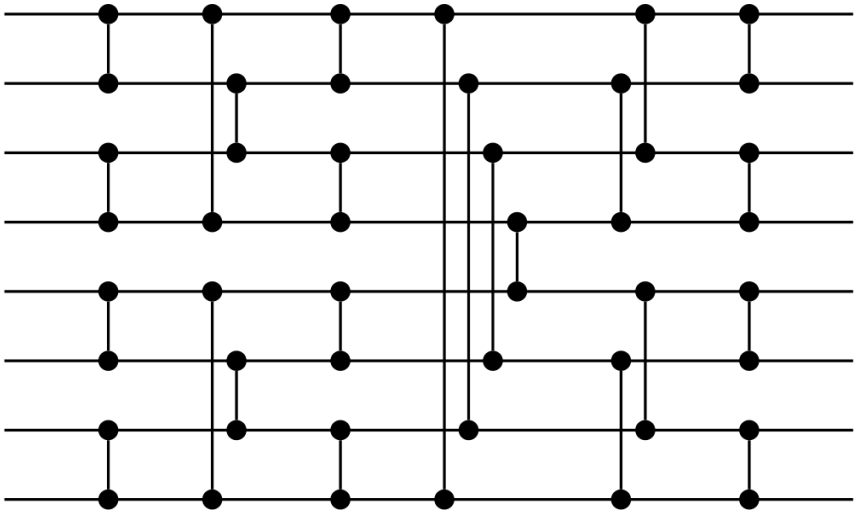
Ένα *δίκτυο ταξινόμησης* είναι ένα αφηρημένο μαθηματικό μοντέλο που αποτελείται από καλώδια και συγκριτές και χρησιμοποιείται για την ταξινόμηση μιας ακολουθίας αριθμών. Κάθε καλώδιο έχει μια τιμή. Ένας συγκριτής δέχεται σαν είσοδο δύο καλώδια, συγκρίνει την τιμή που αυτά έχουν και αν ικανοποιούν την διάταξη που αυτός ορίζει αφήνει τις τιμές ως έχουν αλλιώς ανταλλάσσει τις τιμές μεταξύ των δύο καλωδίων. Μια τέτοια κατασκευή από καλώδια και συγκριτές για να θεωρηθεί δίκτυο ταξινόμησης πρέπει τελικά να ταξινομεί οποιαδήποτε ακολουθία αριθμών αντιστοιχιστεί αρχικά στα καλώδια.

Το παράδειγμα που φαίνεται στο σχήμα 4.4 αποτελεί ένα δίκτυο ταξινόμησης με τέσσερα καλώδια και πέντε συγκριτές, των οποίων η κατεύθυνση είναι από πάνω προς τα κάτω, δηλαδή ο αριθμός στο πάνω καλώδιο μετά από έναν συγκριτή θα είναι μικρότερος από αυτόν στο κάτω καλώδιο, και συνολικά ταξινομεί τέσσερις αριθμούς κατά αύξουσα σειρά.

Κάθε αλγόριθμος ταξινόμησης που βασίζεται σε συγκρίσεις, μπορεί να αναπαρασταθεί από ένα δίκτυο ταξινόμησης, όπου κάθε καλώδιο κατά σειρά έχει ως τιμή την τιμή της αντίστοιχης θέσης του πίνακα δεδομένων, και οι συγκριτές αναπαριστούν τους ελέγχους και ανταλλαγές τιμών που γίνονται στις θέσεις του πίνακα. Για κάθε δίκτυο ταξινόμησης μπορούμε να κατασκευάσουμε έναν παράλληλο αλγόριθμο που να πραγματοποιεί ταυτόχρονα πολλές συγκρίσεις. Ο βαθμός παραλληλοποίησης ενός τέτοιου δικτύου προκύπτει από τις εξαρτήσεις που υπάρχουν μεταξύ των συγκριτών, όταν δύο συγκριτές χρησιμοποιούν το ίδιο καλώδιο. Λόγω του ότι η ταξινόμηση είναι βασισμένη σε συγκρίσεις, δεν μπορούμε να έχουμε λιγότερους από $\Omega(\mathrm{n}\log\mathrm{n})$ συγκριτές. Σε κάθε πέρασμα ενός παράλληλου αλγόριθμου, δεδομένης της ανεξαρτησίας των δεδομένων, μπορούμε να έχουμε το



Σχήμα 4.4: Παράδειγμα δικτύου ταξινόμησης και λειτουργίας του



Σχήμα 4.5: Δίκτυο ταξινόμησης που υλοποιείται από τον διτονικό ταξινομητή

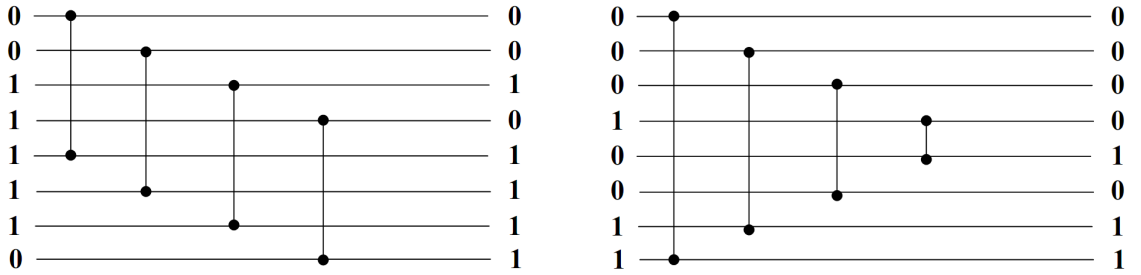
πολύ $\mathrm{n}/2$ συγκρίσεις και ανταλλαγές τιμών, άρα συνολικά ένας παράλληλος αλγόριθμος πρέπει να κάνει τουλάχιστον $\Omega(\log \mathrm{n})$ περάσματα πάνω στον πίνακα των δεδομένων.

Διτονικός ταξινομητής

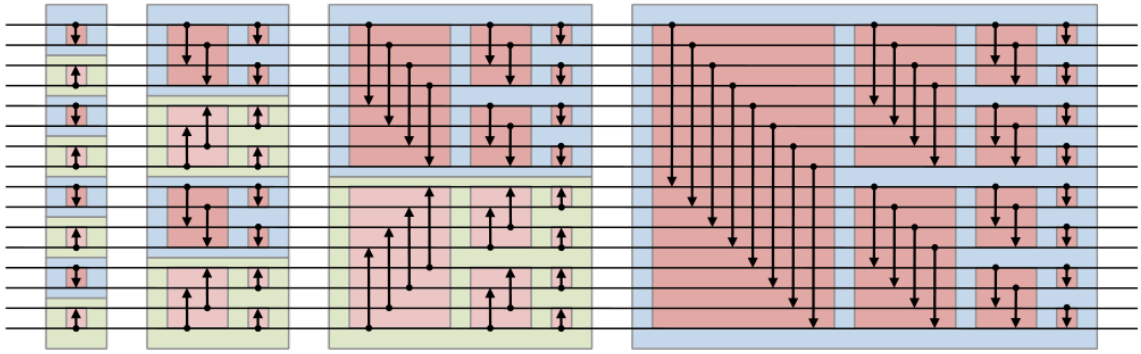
Ένα δίκτυο ταξινόμησης που μπορούμε να χρησιμοποιήσουμε για την υλοποίηση ενός παράλληλου αλγόριθμου ταξινόμησης είναι ο διτονικός ταξινομητής [Knu73]. Το δίκτυο ταξινόμησης στο οποίο βασίζεται αποτελείται από $\mathrm{O}(\mathrm{n}\log^2(\mathrm{n}))$ συγκριτές με καθυστέρηση $\mathrm{O}(\log^2(\mathrm{n}))$, δηλαδή αποτελείται από $\mathrm{O}(\log^2(\mathrm{n}))$ βήματα που στο καθένα γίνονται $\mathrm{O}(\mathrm{n})$ συγκρίσεις και πιθανώς ανταλλαγές τιμών. Η ανάπτυξη του αλγόριθμου βασίζεται στην έννοια της διτονικής ακολουθίας, η οποία είναι μια ακολουθία που είναι αρχικά μονοτονικά αύξουσα και στη συνέχεια μονοτονικά φθίνουσα (ή το αντίστροφο).

Η κατασκευή του δικτύου αυτού γίνεται χρησιμοποιώντας δύο βασικά δομικά στοιχεία, τον *ημι-καθαριστή* (half-cleaner) και το *συγχωνευτή* (merger) (σχήμα 4.6). Ο ημικαθαριστής χωρίζει μια διτονική ακολουθία σε δύο υποακολουθίες όπου κάθε στοιχείο της πρώτης υποακολουθίας είναι μικρότερο ή ίσο με κάθε στοιχείο της δεύτερης. Ο συγχωνευτής μαζί με τροποποιημένους ημικαθαριστές συνθέτουν το δίκτυο συγχώνευσης. Το δίκτυο συγχώνευσης παίρνει δύο ταξινομημένες υποακολουθίες και τις συγχωνεύει επιστρέφοντας μια συνολικά ταξινομημένη ακολουθία. Υπάρχουν δύο γενικές μορφές κατασκευής του δικτύου ταξινόμησης η πρώτη με τη χρήση ημικαθαριστών (σχήμα 4.7) και η δεύτερη με τη χρήση δικτύων συγχώνευσης (σχήμα 4.5).

Κατά την εκτέλεση του διτονικού ταξινομητή, αρχικά ταξινομούνται ακολουθίες μήκους 2, η πρώτη κατά αύξουσα σειρά, η δεύτερη κατά φθίνουσα, η τρίτη κατά αύξουσα κ.ο.κ. Στο επόμενο βήμα θεωρούνται οι διτονικές ακολουθίες μήκους 4 που έχουν δημιουργηθεί και ταξινομούνται οι μισές κατά αύξουσα και οι άλλες μισές κατά φθίνουσα σειρά. Η εκτέλεση συνεχίζεται με αυτή την λογική διπλασιάζοντας το μέγεθος της διτονικής ακολουθίας που εξετάζεται όπως έχει δημιουργηθεί από το προηγούμενο βήμα του αλγορίθμου. Παράδειγμα διτονικού ταξινομητή 16 αριθμών φαίνεται στο σχήμα 4.7 όπου για κάθε βήμα με γαλάζιο φόντο τονίζονται οι ακολουθίες που ταξινομούνται κατά αύξουσα σειρά και με πράσινο αυτές που ταξινομούνται κατά φθίνουσα σειρά.



Σχήμα 4.6: Ημι-καθαριστής και συγχωνευτής



Σχήμα 4.7: Δίκτυο διτονικού ταξινομητή 16 αριθμών υλοποιημένο με ημι-καθαριστές

Η λογική εκτέλεσης που χρησιμοποιείται από τον διτονικό ταξινομητή υπονοεί πως οι ακολουθίες που θα ταξινομηθούν πρέπει να έχουν μήκος κάποια δύναμη του δύο. Αν θέλουμε να ταξινομήσουμε έναν πίνακα με διαφορετικό αριθμό στοιχείων, θα πρέπει να του αυξήσουμε το μέγεθος μέχρι την επόμενη δύναμη του δύο, προσθέτοντας κατάλληλα στις κενές θέσης στοιχεία που είναι μεγαλύτερα από οποιοδήποτε στοιχείο του αρχικού πίνακα και μετά να χρησιμοποιήσουμε το δίκτυο ταξινόμησης. Αυτό μας επιβάλει είτε να χρησιμοποιήσουμε έναν ξεχωριστό πίνακα στον οποίο θα γίνει η ταξινόμηση είτε να έχουμε προβλέψει το μέγεθος του πίνακα να είναι μια δύναμη του δύο και με ένα βήμα επεξεργασίας να προσθέσουμε τα κατάλληλα στοιχεία στο τέλος του. Και με τους δύο τρόπους έχουμε μη αποδοτική χρήση μνήμης και ειδικά στον πρώτο επιβάρυνση με την μετακίνηση δεδομένων.

Η λύση που υλοποιήθηκε, και η οποία αναφέρεται στο [PSHL10], βασίζεται στα χαρακτηριστικά του διτονικού ταξινομητή. Δεν είναι ανάγκη η σειρά με την οποία είναι ταξινομημένες οι ακολουθίες να είναι αύξουσα, φθίνουσα, αύξουσα κ.ο.κ. και η σειρά αυτή να διατηρείται αυστηρά κατά την εκτέλεση του αλγορίθμου. Αυτό που μας ενδιαφέρει είναι οι δύο υποακολουθίες να συνθέτουν τελικά μια διτονική ακολουθία. Για να επιλέξουμε τον τρόπο που θα ταξινομηθούν οι ακολουθίες, προβάλλουμε τον πίνακά σε έναν πίνακα μεγέθους ίσο με την επόμενη δύναμη του δύο. Η ακολουθία στην οποία προβάλλεται το τελευταίο στοιχείο του αρχικού μας πίνακα επιλέγουμε να ταξινομηθεί με αύξουσα σειρά και όλες οι προηγούμενες με τέτοιο τρόπο ώστε για το επόμενο βήμα του αλγόριθμου να δημιουργηθούν διτονικές ακολουθίες. Μια τέτοια επιλογή είναι και μοναδική. Όσες ακολουθίες βρίσκονται έξω από τα όρια του πίνακα τις αγνοούμε. Η επιλογή για αύξουσα ταξινόμηση του τελευταίου τμήματος γίνεται ώστε να μην χρειαστεί να διαβάσουμε τιμές έξω από τα όρια του πίνακα δεδομένων, και όσες τιμές θα υπήρχαν στην προβαλλόμενη ακολουθία πέρα από τα όρια του αρχικού πίνακα, θεωρούμε πως έχουν τιμές μεγαλύτε-

ρες από τις ήδη υπάρχουσες και όσες μετακινήσεις αφορούν τέτοιες τιμές ουσιαστικά τις αγνοούμε.

4.4.5 Συρρίκνωση ακμών

Στο τμήμα αυτό γίνεται η χρήση των δεδομένων που έχουν προετοιμαστεί από τα προηγούμενα τμήματα και ουσιαστικά είναι αυτό στο οποίο γίνεται η απλοποίηση του μοντέλου. Η συρρίκνωση ακμής είναι η βασική ενέργεια απλοποίησης που εφαρμόζεται στο μοντέλο, με κάθε μια να λαμβάνει μέρος στα όρια μιας ανεξάρτητης περιοχής από αυτές που έχουν βρεθεί.

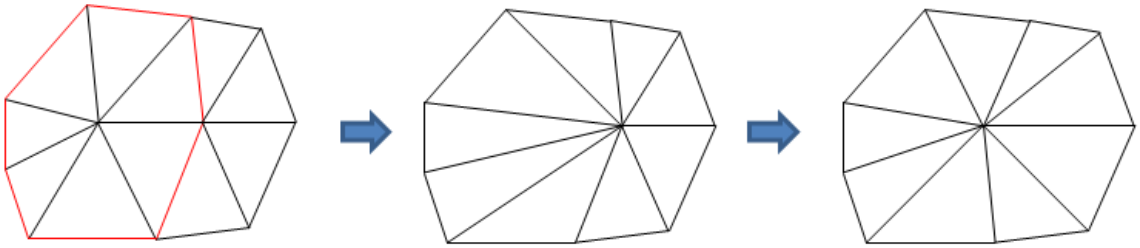
Κάθε περιοχή ορίζεται από μία κεντρική κορυφή, με βάση την οποία θα εκτελεστεί ο αλγόριθμος συρρίκνωσης ακμής. Η συρρίκνωση γίνεται σε μια από τις ακμές στις οποίες συμμετέχει η κεντρική κορυφή. Η επιλογή για το ποια ακμή θα χρησιμοποιηθεί γίνεται βάσει του ποια συρρίκνωση θα δημιουργήσει το μικρότερο σφάλμα αλλά και με τον περιορισμό η τελική κατάσταση του πλέγματος μετά τη συρρίκνωση να διατηρεί την ιδιότητα της πολλαπλότητας στην συγκεκριμένη περιοχή. Αφού επιλεγεί η ακμή την συρρικνώνουμε στο ένα άκρο της και ύστερα χρησιμοποιώντας τα quadrics των αρχικών κορυφών υπολογίζουμε την τελική θέση της κορυφής στην οποία την μετακινούμε. Αυτό που μένει να κάνουμε είναι να τροποποιήσουμε την συνδεσιμότητα των τριγώνων με την διαγραφή όποιων τριγώνων διαγράφονται και την ανανέωση των τιμών για τα quadric και το σφάλμα που αντιστοιχεί στην τελική κορυφή.

Τα τρίγωνα στα οποία θα συμμετέχει η τελική κορυφή είναι τα τρίγωνα στα οποία συμμετείχαν οι δύο κορυφές της ακμής που συρρικνώνεται, μείον τα δύο το πολύ τρίγωνα στα οποία συμμετέχει η ακμή (βασιζόμενοι στην ιδιότητα της πολλαπλότητας). Η πληροφορία σε ποια τρίγωνα χρησιμοποιείται κάθε κορυφή βρίσκεται σε δύο μέρη στα δεδομένα που χειριζόμαστε, στους δείκτες από τις κορυφές προς τα τρίγωνα, και στα ίδια τα τρίγωνα.

Το πρώτο βήμα που εκτελούμε είναι να τροποποιήσουμε την συνδεσιμότητα των τριγώνων. Έτσι διατρέχουμε τη λίστα των τριγώνων που χρησιμοποιούν την κεντρική κορυφή και αλλάζουμε τις αντίστοιχες εγγραφές στα τρίγωνα ώστε να δείχνουν στην δεύτερη κορυφή της ακμής πάνω στην οποία γίνεται η απλοποίηση. Μετά από αυτό το βήμα, κανένα τρίγωνο δεν θα έχει αναφορά στην κεντρική κορυφή, και στο τέλος της διαδικασίας συρρίκνωσης θα μπορέσουμε να την σημειώσουμε σαν διαγραμμένη.

Ο αριθμός των τριγώνων που συμμετέχει η κάθε κορυφή δεν είναι ο ίδιος για κάθε κορυφή και δεν παραμένει σταθερός κατά την απλοποίηση του μοντέλου. Με την OpenCL όμως έχουμε πίνακες σταθερού μεγέθους, οπότε πρέπει να τους χειριστούμε με τέτοιο τρόπο ώστε να έχουμε εγγραφές μεταβλητού μήκους. Οι εγγραφές που θα έχει η τελική κορυφή είναι αυτές που είχαν οι δύο αρχικές κορυφές. Ο διαθέσιμος χώρος που έχουμε για μια κορυφή στην γενική περίπτωση δεν είναι αρκετός για να κρατήσουμε τους δείκτες που χρειαζόμαστε. Τον χώρο που χρειαζόμαστε τον βρίσκουμε από την κορυφή που διαγράφουμε, με αυτό να γίνεται με τον τρόπο που έχουμε κατασκευάσει την δομή που κρατάμε τους δείκτες. Με το τρίτο πεδίο (continuesTo) κάθε εγγραφής του πίνακα επικεφαλίδων ορίζουμε ουσιαστικά μια συνδεδεμένη λίστα και έτσι συνδέουμε τις δύο αρχικές λίστες σε μία που περιέχει όλους τους δείκτες προς τα τρίγωνα που είχαν οι δύο αρχικές κορυφές. Αυτό που απομένει είναι να διατρέξουμε την τελική λίστα δεικτών και να αφαιρέσουμε τα τρίγωνα που θα διαγραφούν.

Κατά την διαγραφή των τριγώνων από τη λίστα δεικτών κάνουμε και μια συμπύκνωσή



Σχήμα 4.8: Συρρίκνωση ακμής: Αριστερά: Αρχικό τριγωνικό πλέγμα (με κόκκινο πλαίσιο τονίζεται η ανεξάρτητη περιοχή που έχει επιλεγεί). Μέση: Τροποποίηση συνδεσιμότητας. Δεξιά: Τοποθέτηση κορυφής στην τελική της θέση

της ώστε αρχικά να μην έχει κενά ενδιάμεσα αλλά και για να βελτιστοποιήσουμε την χρήση της μνήμης για κάθε κορυφή. Αν παραλείπαμε το βήμα της συμπύκνωσης, σε επόμενα στάδια απλοποίησης θα έπρεπε να διατρέχουμε αρκετούς κόμβους της λίστας, τόσους όσες και οι συρρικνώσεις που έχουν γίνει και έχει προκύψει η τελική κορυφή, με το αντίστοιχο κόστος στο χρόνο εκτέλεσης, όταν πραγματικά θα χρειαζόμασταν μόνο κάποιους από αυτούς τους κόμβους για να αποθηκεύσουμε τους δείκτες που θέλουμε. Επίσης με την αραιή κατανομή των δεδομένων στους πίνακες θα ακυρώναμε την λειτουργία των κρυφών μνημών που πιθανώς να υπάρχουν.

Τα τρίγωνα που θα διαγραφούν εμφανίζονται εκτός από τις λίστες δεικτών των κορυφών της ακμής που συρρικνώνεται, και στις λίστες των τρίτων κορυφών των δύο τριγώνων. Έτσι πρέπει και σε αυτές να κάνουμε την ίδια διαδικασία αφαίρεσης των δεικτών προς τα τρίγωνα που θα διαγραφούν ακολουθούμενη από την συμπύκνωση της λίστας. Η εύρεση αυτών των κορυφών είναι αρκετά εύκολη καθώς ήδη ξέρουμε ποια τρίγωνα θα διαγραφούν και ποιες είναι οι κορυφές που συμμετέχουν στην ακμή που συρρικνώνεται.

Το επόμενο βήμα της διαδικασίας είναι να τροποποιήσουμε την θέση της κορυφής που απομένει μετά τη συρρίκνωση. Με την χρήση των quadrics υπολογίζουμε την τελική θέση για την κορυφή και το σφάλμα που της αντιστοιχεί.

Τελευταίο βήμα της διαδικασίας είναι η διαγραφή των τριγώνων και της κορυφής που δεν χρειαζόμαστε πλέον. Όπως έχουμε αναφέρει και κατά την περιγραφή των δομών δεδομένων που χρησιμοποιούμε, η διαγραφή των τριγώνων και των κορυφών που δεν χρειαζόμαστε γίνεται με την τοποθέτηση μιας τιμής-δείκτη διαγραφής.

Συνολικά τα βήματα της διαδικασίας συρρίκνωσης, στη σειρά που εκτελούνται είναι τα παρακάτω:

- Εύρεση καλύτερης γειτονικής κορυφής.
- Εύρεση τριγώνων προς διαγραφή.
- Υπολογισμός βέλτιστης τοποθέτησης.
- Τροποποίηση συνδεσιμότητας τριγώνων
- Συνένωση των δύο λιστών δεικτών από τις κορυφές προς τα τρίγωνα
- Πακετάρισμα λιστών δεικτών ακραίων κορυφών και συγχρόνως αφαίρεση δεικτών προς τα τρίγωνα που θα διαγραφούν.

- Εύρεση των τρίτων κορυφών.
- Πακετάρισμα λιστών δεικτών των παραπάνω κορυφών και συγχρόνως αφαίρεση δεικτών προς τα τρίγωνα που θα διαγραφούν.
- Εφαρμογή βέλτιστης τοποθέτησης, ενημέρωση quadric και σφάλματος τελικής κορυφής.
- Διαγραφή τριγώνων από το μοντέλο
- Διαγραφή κεντρικής κορυφής από το μοντέλο

Κατά τη διαδικασία της συρρίκνωσης ακμής έχουμε επιλέξει να διαγράφεται η κορυφή που βρίσκεται στο κέντρο της ανεξάρτητης περιοχής. Η επιλογή αυτή έγινε ώστε να εξαλείψουμε τις εξαρτήσεις που υπάρχουν μεταξύ των περιοχών που εφαρμόζεται η διαδικασία απλοποίησης και να μπορέσουμε να εκτελέσουμε παράλληλα όλες τις συρρικνώσεις για ένα πέρασμα του αλγορίθμου απλοποίησης. Αυτό όμως δεν επηρεάζει το τελικό ποιοτικό αποτέλεσμα της διαδικασίας καθώς η κορυφή που προκύπτει τοποθετείται στον τρισδιάστατο χώρο χρησιμοποιώντας τα quadrics των δύο κορυφών της ακμής.

4.4.6 Ανάκτηση τελικού μοντέλου

Μετά το τέλος της επαναληπτικής διαδικασίας και αφού έχουμε απλοποιήσει το μοντέλο στο επίπεδο που επιθυμούμε, μένει μόνο να επιστρέψουμε το αποτέλεσμα. Οι πίνακες όμως των τριγώνων και των κορυφών περιέχουν αρκετά κενά καθώς περιέχουν κορυφές και τρίγωνα που έχουν σημειωθεί σαν διαγραμμένα. Η διαδικασία ανάκτησης του μοντέλου αποτελεί ένα στάδιο μεταεπεξεργασίας των πινάκων (που εκτελείται από την CPU) ώστε να τους φέρουμε στην μορφή που θέλουμε.

Η μετακίνηση των εγγραφών των τριγώνων ώστε να δημιουργήσουμε έναν πίνακα χωρίς κενά δεν επιφέρει από μόνη της κάποιο επιπλέον κόστος. Από την άλλη, για να μετακινήσουμε τις κορυφές στον πίνακα κορυφών, πρέπει να ενημερώσουμε και τα αντίστοιχα τρίγωνα. Για να επιτύχουμε το επιθυμητό αποτέλεσμα, η διαδικασία εκτελείται σε δύο βήματα, πρώτα συμπυκνώνουμε τον πίνακα κορυφών και ύστερα τον πίνακα τριγώνων, με παράλληλη χρήση ενός πίνακα όπου κρατά με τον αριθμό των θέσεων που έχει μετακινηθεί η κάθε κορυφή. Κατά τη συμπύκνωση του πίνακα κορυφών, μετακινούμε τις κορυφές τόσες θέσεις όσες και οι διαγραμμένες κορυφές που υπάρχουν πριν την τρέχουσα κορυφή. Με κάθε μετακίνηση σημειώνουμε στον πίνακα μετακίνησης την απόσταση από την αρχική θέση της κορυφής στον πίνακα κορυφών μέχρι τη θέση που προέκυψε. Κατά την συμπύκνωση των τριγώνων κάνουμε παρόμοια μετακίνηση των εγγραφών όπου συγχρόνως κάνουμε χρήση του πίνακα μετακίνησης για να διορθώσουμε τους δείκτες στις τελικές θέσεις των κορυφών.

Μετά και την εκτέλεση του τελικού αυτού βήματος μπορούμε να επιστρέψουμε το αποτέλεσμα, το τελικό απλοποιημένο μοντέλο που έχει προκύψει από τον αλγόριθμο απλοποίησης.

4.4.7 Παράμετροι αλγόριθμου

Κατά την εκτέλεση του αλγορίθμου απλοποίησης υπάρχουν κάποιες μεταβλητές που επηρεάζουν τον χρόνο εκτέλεσης αλλά και την ποιότητα του τελικού αποτελέσματος που παράγεται.

Η πρώτη είναι αυτή που καθορίζει το ποια διαδικασία από τις τρεις που έχουν υλοποιηθεί θα χρησιμοποιηθεί για την εύρεση των ανεξάρτητων κορυφών (βλ. ενότητα 4.4.3). Σαν προκαθορισμένη επιλογή έχουμε να χρησιμοποιείται ο τρίτος αλγόριθμος, η παράλληλη υλοποίηση εύρεσης ανεξάρτητων περιοχών χωρίς την χρήση του σφάλματος κάθε σημείου.

Επόμενη παράμετρος που χρησιμοποιείται αφορά το ποσοστό των ανεξάρτητων περιοχών που θα χρησιμοποιηθούν σε κάθε επανάληψη του αλγορίθμου απλοποίησης. Σε έναν αντίστοιχο σειριακό αλγόριθμο απλοποίησης θα χρησιμοποιούσαμε κάθε φορά μια κορυφή, αυτή με το ελάχιστο σφάλμα. Ο στόχος όμως που έχουμε είναι να δημιουργήσουμε έναν παράλληλο αλγόριθμο και να χρησιμοποιήσουμε όσο το δυνατόν περισσότερες. Σε κάθε επανάληψη εξετάζουμε όλο το μοντέλο για να βρούμε τις ανεξάρτητες κορυφές και τις ταξινομούμε με βάση το σφάλμα που αντιστοιχεί στην κάθε μία, διαδικασίες που επιφέρουν κάποιο κόστος κατά τον χρόνο εκτέλεσης και το οποίο για μεγάλα μοντέλα μπορεί να γίνει αρκετά υψηλό. Αν χρησιμοποιήσουμε όλες τις κορυφές που βρίσκουμε, θα εκτελέσουμε απλοποιήσεις που μπορεί να έχουν μεγάλο σφάλμα στο τελικό αποτέλεσμα. Αν χρησιμοποιήσουμε πολύ λίγες θα έχουμε πολλές επαναλήψεις του αλγορίθμου με το αντίστοιχο κόστος κατά τον χρόνο εκτέλεσης. Η παράμετρος αυτή μας δίνει τον έλεγχο για την ταχύτητα και την ποιότητα των αποτελεσμάτων του αλγόριθμου απλοποίησης. Η τιμή που δίνεται σαν προκαθορισμένη κατά την υλοποίηση είναι 0.85, η χρησιμοποίηση του 85% των περιοχών που βρίσκονται σε κάθε πέρασμα και έχουν το μικρότερο σχετικό σφάλμα, τιμή η οποία δίνει αρκετά καλά αποτελέσματα σχετικά γρήγορα.

Μια άλλη τιμή η οποία χειρίζεται εσωτερικά από τον παράλληλο αλγόριθμο εύρεσης ανεξάρτητων σημείων είναι ο αριθμός των νημάτων εκτέλεσης που θα δημιουργηθούν και συγχρόνως ο αριθμός των τμημάτων που θα σπάσει το μοντέλο για την εύρεση των ανεξάρτητων περιοχών. Για την μεγιστοποίηση της χρήσης των επεξεργαστών θέλουμε να έχουμε αρκετά νήματα ώστε να έχουμε την καλύτερη δυνατή χρήση τους. Αν όμως χρησιμοποιήσουμε πάρα πολλά, αυξάνουμε τις εξαρτήσεις μεταξύ των περιοχών που μπορούν να μειώσουν τον τελικό αριθμό των κορυφών που θα επιλεγούν ως ανεξάρτητες. Έτσι και εδώ πρέπει να υπολογίσουμε μια τιμή που μας δίνει όσο το δυνατόν καλύτερα αποτελέσματα. Στην τρέχουσα υλοποίηση, ο υπολογισμός του αριθμού των τμημάτων γίνεται αρχικά βάσει του πλήθους των μονάδων υπολογισμού έτσι ώστε να μην δημιουργούνται περισσότερα από 64 τμήματα (με τα αντίστοιχα νήματα εκτέλεσης) για την κάθε μία αλλά και κάθε τμήμα να έχει τουλάχιστον 100 κορυφές.

Τέλος υπάρχει και η παράμετρος που καθορίζει το μέγεθος του πίνακα των ανεξάρτητων κορυφών. Ο πίνακας αυτός δεσμεύεται κατά την πρώτη επανάληψη του αλγόριθμου απλοποίησης και επαναχρησιμοποιείται στις επόμενες. Το ιδανικό θα ήταν να υπολογίζουμε ακριβώς το μέγεθος που χρειαζόμαστε. Καθώς όμως η συνδεσιμότητα του τριγωνικού πλέγματος των μοντέλων δεν έχει κάποια κανονικότητα, το μόνο που μπορούμε να κάνουμε είναι να χρησιμοποιήσουμε μια προσεγγιστική τιμή η οποία θα καλύπτει τις ανάγκες μας. Η τιμή που χρησιμοποιήθηκε κατά την υλοποίηση είναι ίση με το 5% του πλήθους των αρχικών κορυφών του μοντέλου.

Κεφάλαιο 5

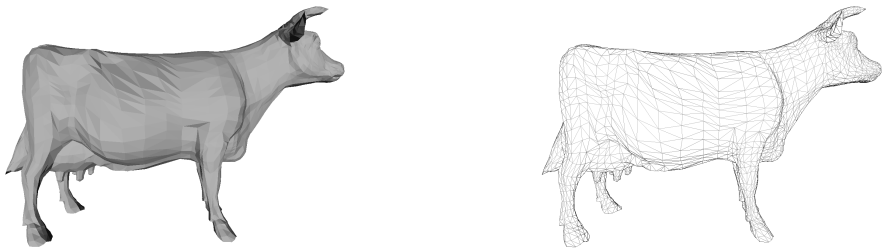
Αποτελέσματα απλοποίησης

Σε αυτό το κεφάλαιο θα εξετάσουμε τα ποιοτικά αποτελέσματα του αλγορίθμου απλοποίησης, το πώς επηρεάζονται τα μοντέλα κατά την απλοποίηση και κατά πόσο μοιάζουν με το αρχικό μοντέλο που δίνεται κάθε φορά. Θα εξετάσουμε τις περιπτώσεις όπου απλοποιείται ένα μοντέλο με σχετικά χαμηλό βαθμό λεπτομέρειας και ένα μοντέλο με υψηλό βαθμό λεπτομέρειας. Τέλος θα εξετάσουμε τον τρόπο που επηρεάζουν οι παράμετροι του αλγορίθμου απλοποίησης τα αποτελέσματά του.

5.1 Απλοποίηση μοντέλου που έχει χαμηλό βαθμό λεπτομέρειας

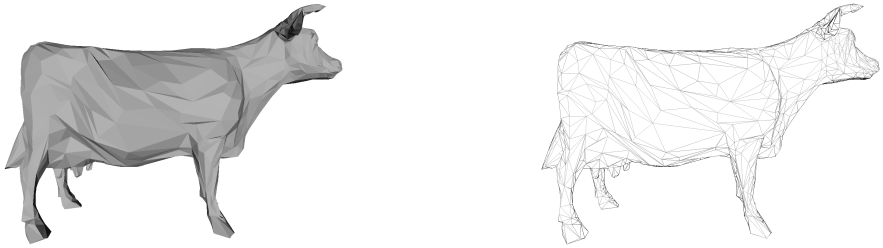
Το μοντέλο που θα εξετάσουμε είναι αυτό μιας αγελάδας στην αρχική του ανάλυση και ύστερα σε διάφορα στάδια απλοποίησης χρησιμοποιώντας τις προκαθορισμένες τιμές για τις παραμέτρους του αλγόριθμου απλοποίησης. Το αρχικό μοντέλο αποτελείται από 2903 κορυφές και 5804 τρίγωνα. Η ανάλυση του αρχικού μοντέλου είναι σχετικά χαμηλή και με την απλοποίηση αναμένουμε κάποια εμφανή διαφοροποίηση του μοντέλου με τροποποίηση κάποιων από τα χαρακτηριστικά του. Στα σχήματα 5.1 έως και 5.5 παρουσιάζεται το μοντέλο στην αρχική του μορφή, ακολουθούμενο από απλοποιημένες μορφές του οι οποίες αποτελούνται από 1451 κορυφές (2900 τρίγωνα), 500 κορυφές (≅1000 τρίγωνα), 150 κορυφές (≅300 τρίγωνα) και 51 κορυφές (≅100 τρίγωνα).

Στα δύο πρώτα επίπεδα που απλοποιήθηκε το μοντέλο, 25% και 8,6% αντίστοιχα, τα χαρακτηριστικά και το συνολικό του σχήμα διατηρούνται στο αποτέλεσμα. Πιο συγκεκριμένα στα κέρατα, στο σχήμα του κεφαλιού, τα πόδια και η ουρά, σημεία που χρειάζονται πιο πολύ λεπτομέρεια και προσδίδουν τα αναγνωριστικά χαρακτηριστικά του μοντέλου διατηρούνται, με το σώμα να απλοποιείται πιο πολύ. Στα επόμενα στάδια της απλοποίησης, στο 2,5% και 0,87% αντίστοιχα, ο αριθμός των κορυφών και τριγώνων έχουν μειωθεί δραματικά ώστε να μην μπορούν να αναπαραστήσουν πιστά το μοντέλο. Στην πρώτη περίπτωση παρατηρούμε ότι διατηρούνται ακόμα οι χαρακτηριστικές λεπτομέρειες του μοντέλου και το γενικό του σχήμα να παραμένει αναγνωρίσιμο παρότι έχουν απλοποιηθεί αρκετά τα χαρακτηριστικά του μοντέλου και έχουν αρχίσει να γίνονται εμφανείς παραμορφώσεις. Στην δεύτερη περίπτωση, με τον υπερβολικά χαμηλό αριθμό τριγώνων, το μοντέλο έχει αρκετές παραμορφώσεις και έχουν χαθεί τα περισσότερα χαρακτηριστικά του μοντέλου, αλλά ακόμα σε αυτό τον βαθμό απλοποίησης, στο τελικό μοντέλο υπάρ-



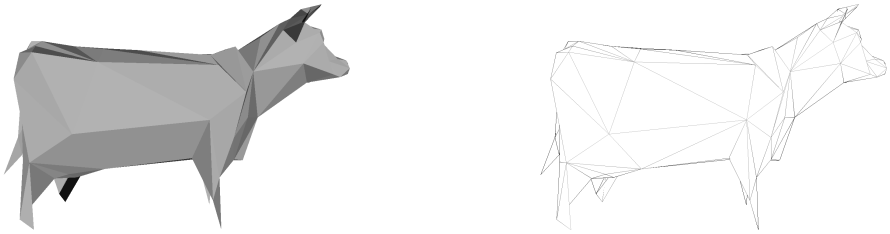
Σχήμα 5.1: Αρχικό μοντέλο αγελάδας με 2903 κορυφές

χει ακόμα το συνολικό σχήμα του μοντέλου όπου μπορούμε να ξεχωρίσουμε τις περιοχές που βρισκόντουσαν το κεφάλι, το σώμα, τα πόδια και κάποια τρίγωνα υπάρχουν ακόμα για να δείξουν κάποια από τα χαρακτηριστικά του όπως τα κέρατα, την ουρά και κάποια καμπυλότητα στο πρόσωπο που αναπαριστά το αρχικό σχήμα του κεφαλιού του μοντέλου.

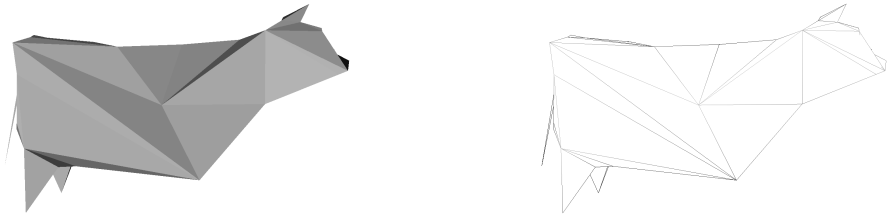


Σχήμα 5.2: Απλοποιημένο μοντέλο αγελάδας στις 1451 κορυφές

Σχήμα 5.3: Απλοποιημένο μοντέλο αγελάδας στις 500 κορυφές



Σχήμα 5.4: Απλοποιημένο μοντέλο αγελάδας στις 150 κορυφές



Σχήμα 5.5: Απλοποιημένο μοντέλο αγελάδας στις 50 κορυφές

5.2 Απλοποίηση μοντέλου που έχει υψηλό βαθμό λεπτομέρειας

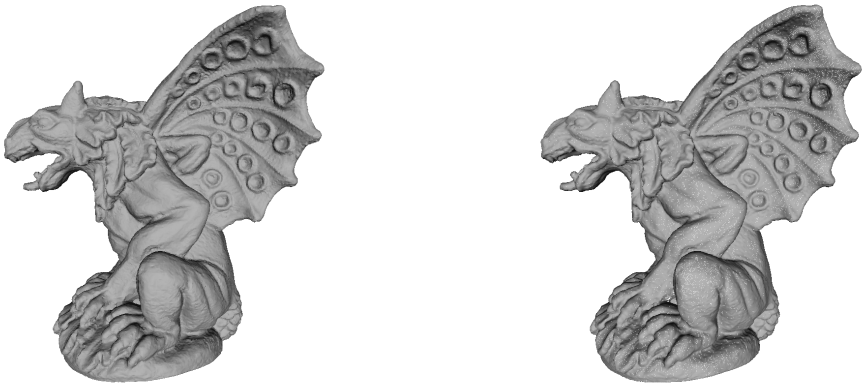
Σε αυτή την περίπτωση θα εξετάσουμε το μοντέλο ενός gargoyle. Το αρχικό μοντέλο είναι σχετικά μεγάλο και αποτελείται από 500.000 κορυφές και 1.000.000 τρίγωνα. Οι απλοποιήσεις που παρουσιάζονται αποτελούνται από 40000 κορυφές (80.000 τρίγωνα), 5000 κορυφές (10.000 τρίγωνα), 2000 κορυφές (4000 τρίγωνα) και 500 κορυφές (≅1000 τρίγωνα).

Το αρχικό μοντέλο (σχήμα 5.6) έχει αρκετά υψηλό επίπεδο λεπτομέρειας, με την επιφάνειά του να αποτελείται από υπερβολικό αριθμό μικρών τριγώνων που αναπαριστούν με πιστότητα το μοντέλο. Η λεπτομέρεια είναι τόσο μεγάλη όπου όταν εμφανίζουμε μόνο το πλέγμα του μοντέλου (wireframe) χωρίς να σκιάζουμε τα τρίγωνα, το μοντέλο και στις δύο περιπτώσεις φαίνεται συμπαγές χωρίς κενά. Αυτό όμως το επίπεδο λεπτομέρειας όμως είναι υπερβολικό αν θέλουμε μόνο να εμφανίσουμε συνολικά το μοντέλο χωρίς να εστιάσουμε σε κάποια συγκεκριμένη λεπτομέρεια.

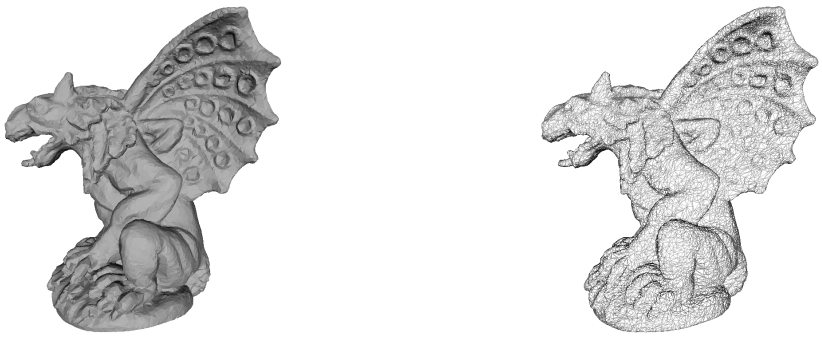
Στο πρώτο επίπεδο απλοποίησης που παρουσιάζεται (σχήμα 5.7), το οποίο χρησιμοποιεί μόνο το 8% των αρχικών κορυφών και τριγώνων του μοντέλου, δεν μπορούμε να δούμε κάποιες αξιοσημείωτες διαφορές, τουλάχιστον στο μέγεθος που εμφανίζεται. Αν παρατηρήσουμε από πιο κοντά το μοντέλο, θα παρατηρήσουμε πως κάποιες από τις επιφάνειες από λείες έχουν γίνει σχετικά αδρές και όπως για παράδειγμα στο χέρι και στο πόδι, και τα χαμηλότερα κυκλικά σχήματα στα φτερά δεν διαγράφονται με την τελειότητα που υπάρχουν όπως στο αρχικό μοντέλο. Και πάλι όμως έχουμε μια πολύ καλή αναπαράσταση του μοντέλου με στο 8% του αρχικού μοντέλου.

Με περαιτέρω απλοποίηση αρχίζουν να χάνονται κάποιες από τις λεπτομέρειες, όπου στο 1% του αρχικού μοντέλου (σχήμα 5.8), όπως στους κύκλους στα φτερά, στα δάχτυλα και γενικότερα η συνολική ποιότητα του αποτελέσματος να έχει αρχίσει να μειώνεται, με τις επιφάνειες να έχουν αρχίσει να γίνονται αδρές, με τις χαρακτηριστικές λεπτομέρειες στα πόδια και το πώς διαγράφονται τα μάτια να εξαφανίζονται. Παρόλα αυτά το μοντέλο παραμένει αναγνωρίσιμο, με αρκετές λεπτομέρειες ακόμα να διαγράφονται στο αποτέλεσμα. Η κατανομή των τριγώνων παρατηρούμε ότι ακολουθεί τα χαρακτηριστικά του μοντέλου, με ομοιόμορφη αραιή κατανομή στις σχετικά λείες περιοχές και μεγαλύτερη συγκέντρωση στα σημεία που υπάρχουν λεπτομέρειες ή και μεγαλύτερη καμπυλότητα.

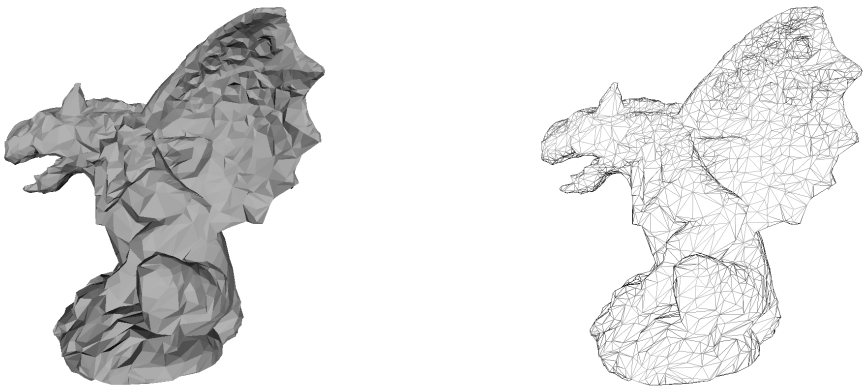
Σε ακόμα μεγαλύτερη απλοποίηση, όπου φτάνουμε στο 0,4% (σχήμα 5.9) και τελικά στο 0,01% (σχήμα 5.10) του αρχικού μοντέλου, οι λεπτομέρειες έχουν εξαφανιστεί τελείως, αλλά η συνολική δομή και το σχήμα διατηρούνται. Έχουν διατηρηθεί το κεφάλι με το ανοιχτό στόμα, τα αυτιά, τα ανοιχτά φτερά, ο κορμός και η βάση, στοιχεία που το κάνουν αναγνωρίσιμο ακόμα και με τέτοια θεαματική μείωση σε τρίγωνα.



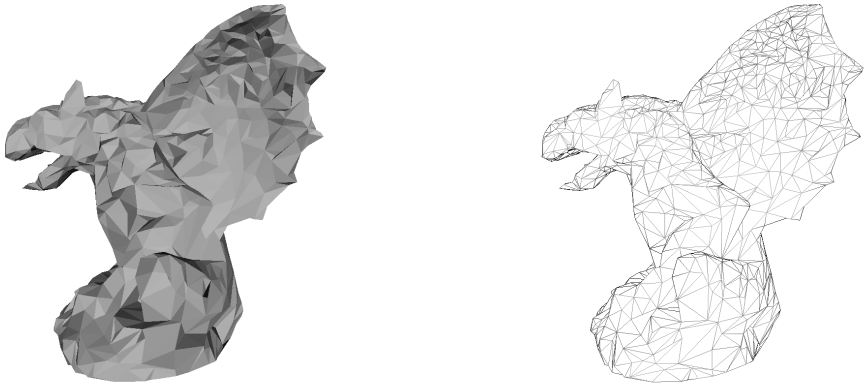
Σχήμα 5.6: Αρχικό μοντέλο gargoyle με 500.000 κορυφές



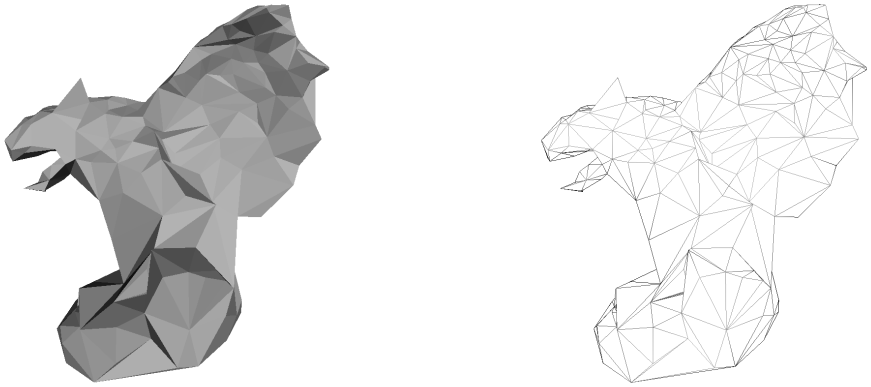
Σχήμα 5.7: Μοντέλο gargoyle απλοποιημένο στις 40000 κορυφές



Σχήμα 5.8: Μοντέλο gargoyle απλοποιημένο στις 5000 κορυφές



Σχήμα 5.9: Μοντέλο gargoyle απλοποιημένο στις 2000 κορυφές



Σχήμα 5.10: Μοντέλο gargoyle απλοποιημένο στις 500 κορυφές

5.3 Τρόπος που επηρεάζουν οι παράμετροι την διαδικασία απλοποίησης

Η απλοποίηση αποτελεί μια ευριστική διαδικασία και οι παράμετροι του αλγόριθμου μπορεί να έχουν σημαντική επίπτωση στο τελικό αποτέλεσμα. Η παράμετρος που ορίζει το ποσοστό των ανεξάρτητων περιοχών που θα χρησιμοποιηθούν από αυτές που βρίσκονται σε κάθε πέρασμα του αλγορίθμου μπορεί να επηρεάσει σημαντικά την διατήρηση των λεπτομερειών στο τελικό μοντέλο. Εδώ θα χρησιμοποιηθεί το μοντέλο ενός αλόγου όπου θα παρουσιαστεί στην αρχική του μορφή, και ύστερα απλοποιώντας το στον ίδιο αριθμό κορυφών, θα δούμε τον τρόπο που μπορεί να επηρεάσει αυτή η παράμετρος το τελικό ποιοτικό αποτέλεσμα, αλλά και τον χρόνο εκτέλεσης του αλγορίθμου.

Το αρχικό μοντέλο αποτελείται από 28485 κορυφές και 96966 τρίγωνα (σχήμα 5.11). Όλες οι απλοποιήσεις που θα παρουσιαστούν έγιναν με στόχο τον ίδιο αριθμό κορυφών και τριγώνων, τις 1500 κορυφές και 2996 τρίγωνα. Ο μόνος παράγοντας που αλλάζει την απλοποίηση είναι το ποσοστό των ανεξάρτητων περιοχών που θα χρησιμοποιηθούν σε κάθε πέρασμα του αλγορίθμου.

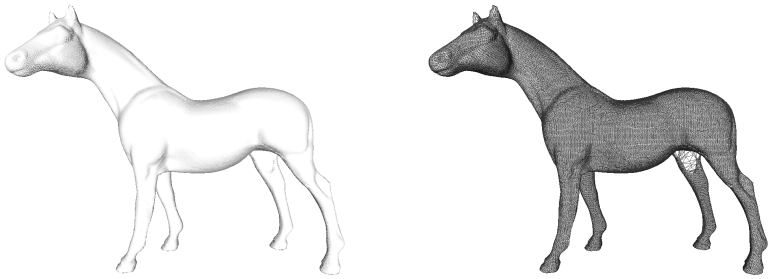
Το αρχικό μοντέλο μπορούμε να δούμε ότι έχει μια σχεδόν ομοιόμορφη κατανομή τριγώνων. Όταν χρησιμοποιούμε όλες τις περιοχές που βρίσκονται στο τελικό μοντέλο προκύπτει μια αντίστοιχη σχεδόν ομοιόμορφη κατανομή τριγώνων. Ουσιαστικά η διαδικασία ταξινόμησης και επιλογής των καλύτερων περιοχών ακυρώνεται και έτσι γίνεται μια επιθετική απλοποίηση. Επειδή όμως σε κάθε πέρασμα του αλγορίθμου χρησιμοποιούμε τον μέγιστο δυνατό αριθμό από τις διαθέσιμες περιοχές, πλησιάζουμε κάθε φορά πιο κοντά στον επιθυμητό αριθμό κορυφών και έτσι ο αλγόριθμος χρειάζεται να κάνει λιγότερες επαναλήψεις. Αυτή η ταχύτητα όμως έχει και επίπτωση στο τελικό ποιοτικό αποτέλεσμα. Όπως μπορούμε να δούμε και από το σχήμα 5.12, και σε σχέση με τα υπόλοιπα αποτελέσματα, οι λεπτομέρειες στο κεφάλι, στα πόδια και σε σημεία που υπάρχει αυξημένη καμπυλότητα στο μοντέλο δεν διατηρούνται. Θα θέλαμε σε αυτά τα μέρη του μοντέλου να δοθεί μεγαλύτερο ποσοστό του τελικού αριθμού των τριγώνων και λιγότερο στις επιφάνειες που δεν έχουν τόσο πολύ λεπτομέρεια, όπως στο κύριο σώμα του μοντέλου.

Όταν αρχίζουμε και μειώνουμε το ποσοστό των κορυφών που θα χρησιμοποιηθούν, αρχίζει να έχει ουσιαστικό ρόλο και η ταξινόμηση. Όσο μικρότερο ποσοστό χρησιμοποιούμε, τόσο περισσότερο σημασία έχει το σφάλμα που αντιστοιχίζεται σε κάθε περιοχή, καθώς για απλοποίηση επιλέγουμε τις περιοχές με το μικρότερο σφάλμα από αυτές που έχουν βρεθεί. Οι περιοχές με μεγάλο σφάλμα (και ουσιαστικά με περισσότερη λεπτομέρεια) βρίσκονται προς το τέλος του πίνακα των ανεξάρτητων κορυφών σε αντίθεση με αυτές που είναι πιο επίπεδες και δεν θα προκαλέσουν μεγάλη διαφορά στο μοντέλο, οι οποίες βρίσκονται προς τις πρώτες θέσεις του πίνακα. Έτσι μπορούμε να δούμε πως μειώνοντας την παράμετρο αυτή, το τελικό αποτέλεσμα αρχίζει να γίνεται καλύτερο μέχρι και την ακραία περίπτωση (σχημα 5.14) όπου χρησιμοποιούμε το 5% των κορυφών που βρίσκονται σε αυτό, μπορούμε να δούμε πως έχει διατηρηθεί, αν και απλοποιημένη σε κάποιο βαθμό, η λεπτομέρεια στο κεφάλι, στα πίσω πόδια και στο πίσω μέρος του αλόγου όπου υπάρχει μεγάλη καμπυλότητα, σε αντίθεση με το κύριο σώμα όπου έχει απλοποιηθεί πιο επιθετικά. Ουσιαστικά υπάρχει μια κατανομή των τριγώνων ανάλογη με την πολυπλοκότητα του μοντέλου σε κάθε τμήμα του. Ο χρόνος εκτέλεσης όμως σε αυτή την περίπτωση είναι αρκετά μεγάλος καθώς με τον μικρό αριθμό των περιοχών που χρησιμοποιούμε σε κάθε πέρασμα, πλησιάζουμε πιο αργά στον επιθυμητό αριθμό κορυφών και τριγώνων, με τον αλγόριθμο να πρέπει να κάνει περισσότερες επαναλήψεις.

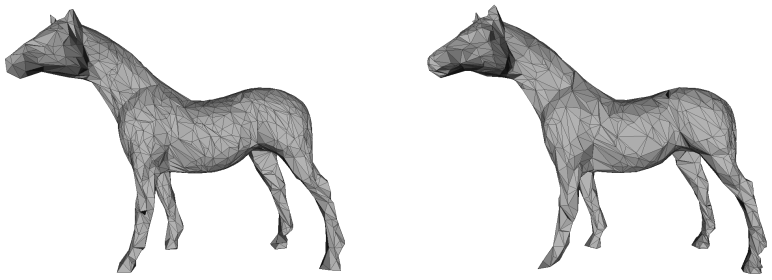
PointsPerPassFactor	Επαναλήψεις αλγορίθμου	Χρόνος εκτέλεσης (ms)
1.00	37	983
0.85 (Default)	43	1155
0.5	72	1950
0.25	144	3635
0.1	357	8174
0.05	731	15257

Πίνακας 5.1: Χρόνος εκτέλεσης για διάφορες τιμές της χρήσης ανεξάρτητων περιοχών

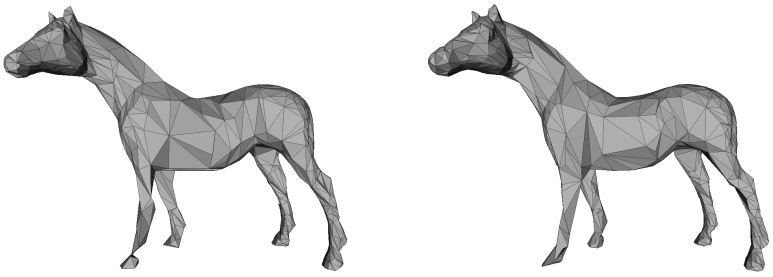
Ο χρόνος εκτέλεσης του αλγορίθμου και οι επαναλήψεις που πρέπει να κάνει στην κάθε περίπτωση, για να δημιουργηθούν τα μοντέλα που εμφανίζονται στα σχήματα 5.12 έως και 5.14, φαίνονται στον πίνακα 5.1.



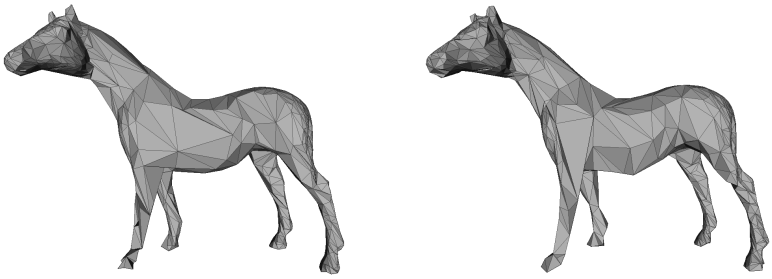
Σχήμα 5.11: Αρχικό μοντέλο αλόγου



Σχήμα 5.12: Χρήση 100% και 85% των ανεξάρτητων περιοχών



Σχήμα 5.13: Χρήση 50% και 25% των ανεξάρτητων περιοχών



Σχήμα 5.14: Χρήση 10% και 5 των ανεξάρτητων περιοχών

Κεφάλαιο 6

Σύγκριση με σειριακό αλγόριθμο

Σε αυτό το κεφάλαιο συγκρίνουμε τα μοντέλα που παράγει ο αλγόριθμος που αναπτύξαμε με αυτά αντίστοιχης σειριακής υλοποίησης του αλγόριθμου απλοποίησης. Η σειριακή υλοποίηση που χρησιμοποιήσαμε για την σύγκριση είναι αυτή που υπάρχει στο πρόγραμμα MeshLab [CCC+08] (έκδοση 1.3.0._64bit) και συγκεκριμένα το φίλτρο "Quadric Edge Collapse Decimation" της κατηγορίας "Remeshing, simplification and reconstruction". Η σύγκριση των δυο υλοποιήσεων γίνεται βάσει του χρόνου εκτέλεσης και της ποιότητας των αποτελεσμάτων που παράγουν. Οι μετρήσεις των ποιοτικών χαρακτηριστικών των μοντέλων έγιναν με την χρήση του προγράμματος PolyMeco [SMS05] (έκδοση 1.1.1).

Όλες οι μετρήσεις έγιναν σε υπολογιστή με επεξεργαστή Intel(R) Core(TM) i7 920 στα 2,67 GHz, 6134 MB DDR3 μνήμη και κάρτα γραφικών την Nvidia GeForce GTX 275 με 896 MB GDDR3 μνήμη. Οι τιμές των παραμέτρων που χρησιμοποιούμε στην υλοποίηση του παράλληλου αλγορίθμου (πρόγραμμα decimator) είναι αυτές που έχουν τεθεί ως προκαθορισμένες (βλ ενότητα 4.4.7). Για τις απλοποιήσεις που γίνονται με το MeshLab, η τιμή της παραμέτρου Quality threshold, είναι ορισμένη στην προκαθορισμένη τιμή 0,3 και οι επιλογές που χρησιμοποιούμε είναι η *Βέλτιστη τοποθέτηση των απλοποιημένων κορυφών* (Optimal position of simplified vertices) και το *Καθάρισμα του μοντέλου μετά την απλοποίηση* (Post-simplification clearing), η οποία διαγράφει μη χρησιμοποιούμενες κορυφές και τρίγωνα από το απλοποιημένο μοντέλο.

Για την σύγκριση των αλγορίθμων χρησιμοποιούμε δυο μοντέλα τα οποία απλοποιούμε, αυτό του αλόγου, μοντέλο χαμηλού βαθμού λεπτομέρειας, και αυτό του gargoyle, μοντέλο υψηλού βαθμού λεπτομέρειας. Η σειρά βημάτων που ακολουθούμε για την εξαγωγή των αποτελεσμάτων είναι αρχικά η απλοποίηση των μοντέλων σε διάφορα επίπεδα απλοποίησης μετρώντας τον χρόνο που απαιτείται κάθε φορά. Ύστερα εισάγουμε τα μοντέλα στο πρόγραμμα PolyMeco για την εκτέλεση μετρήσεων που αφορούν τα ποιοτικά χαρακτηριστικά των απλοποιημένων μοντέλων.

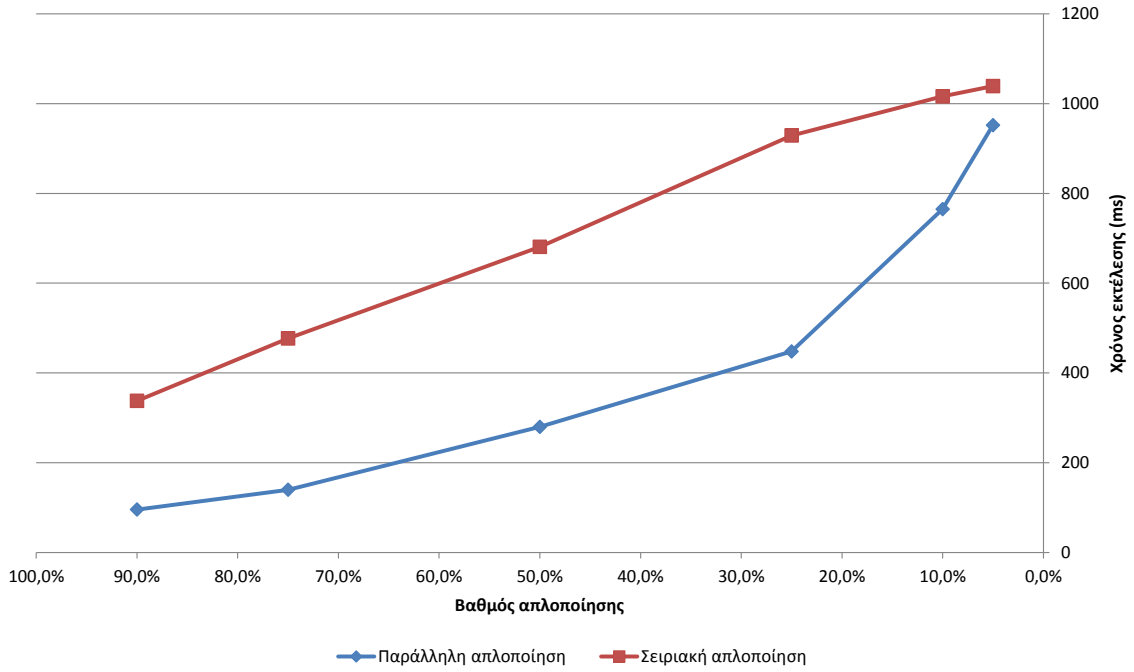
6.1 Μοντέλο χαμηλού βαθμού λεπτομέρειας

Το πρώτο μοντέλο που εξετάζουμε είναι αυτό του αλόγου, ένα μοντέλο με σχετικά χαμηλό βαθμό λεπτομέρειας, το οποίο αποτελείται από 45485 κορυφές και 96966 τρίγωνα.

Στον πίνακα 6.1 μπορούμε να δούμε τον αριθμό των επαναλήψεων που πρέπει να

Απλοποίηση	Κορυφές	Τρίγωνα	Παράλληλη απλοποίηση		Σειριακή απλοποίηση
			Επαναλήψεις	Χρόνος (ms)	Χρόνος (ms)
90%	43636	87268	3	96	338
75%	36363	72722	5	140	477
50%	24243	48488	10	280	681
25%	12121	24238	18	448	929
10%	4848	9692	29	765	1016
5%	2424	4844	37	952	1039

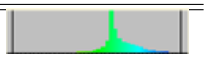
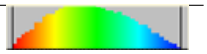
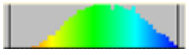
Πίνακας 6.1: Χρόνος που απαιτείται (σε miliseconds) για την απλοποίηση του μοντέλου του αλόγου



Σχήμα 6.1: Χρόνος εκτέλεσης για την απλοποίηση του μοντέλου του αλόγου από τις δύο υλοποιήσεις

εκτελεστούν από την παράλληλη υλοποίηση και τον χρόνο που χρειάζεται για διάφορους βαθμούς απλοποίησης, όπως και τον χρόνο που χρειάζεται η σειριακή υλοποίηση για τον ίδιο βαθμό απλοποίησης. Εδώ παρατηρούμε πως για απλοποίηση του μοντέλου μέχρι και στο 50% του αρχικού, η παράλληλη υλοποίηση χρειάζεται σημαντικά λιγότερο χρόνο, περίπου το 1/3 της σειριακής υλοποίησης. Για περαιτέρω απλοποίηση παρατηρούμε πως ο χρόνος που χρειάζεται η παράλληλη υλοποίηση, αν και μικρότερος αυτής της σειριακής, αρχίζει να αυξάνεται μέχρι το σημείο όπου όταν απλοποιούμε το μοντέλο στο 5% του αρχικού, η διαφορά να έχει μειωθεί σημαντικά. Αυτό συμβαίνει γιατί καθώς μειώνεται το μέγεθος του μοντέλου, το πλήθος των ανεξάρτητων κορυφών που υπάρχουν μειώνεται σημαντικά και έτσι ο παράλληλος αλγόριθμος να χρειάζεται να κάνει περισσότερες επαναλήψεις.

Όσον αφορά τα ποιοτικά χαρακτηριστικά των απλοποιημένων μοντέλων, τα αποτελέσματα των μετρήσεων που θα χρησιμοποιήσουμε από αυτές που είναι διαθέσιμες στο πρόγραμμα PolyMeco είναι:

Μοντέλο	Γεωμετρική απόκλιση	Απόκλιση καν. διαν.	Ομαλότητα	Γωνίες
Αρχικό	-	-	0.000163391	
Παράλληλη απλοποίηση 25%	3.02668E-005	0.0598309	0.000676380	
Σειριακή απλοποίηση 25%	1.45917E-005	0.0328475	0.000471769	
Παράλληλη απλοποίηση 10%	6.85628E-005	0.0965949	0.001191830	
Σειριακή απλοποίηση 10%	3.2593E-005	0.0567528	0.000787039	
Παράλληλη απλοποίηση 5%	1.28863E-004	0.1336120	0.001810930	
Σειριακή απλοποίηση 5%	5.75173E-005	0.0825911	0.001190420	

Πίνακας 6.2: Ποιοτικά χαρακτηριστικά μοντέλων-αποτελεσμάτων για το μοντέλο του αλόγου

Η μέση τιμή της *γεωμετρικής απόκλισης* (Geometric deviation), ένα μέτρο της γεωμετρικής απόστασης ανάμεσα στις κορυφές του απλοποιημένου μοντέλου και τις αντίστοιχες κορυφές του αρχικού μοντέλου. Κάθε κορυφή του απλοποιημένου μοντέλου αντιστοιχίζεται σε μια κορυφή του αρχικού μοντέλου, τέτοια ώστε να ελαχιστοποιείται η μεταξύ τους απόσταση. Η γεωμετρική απόκλιση μας δίνει μια ένδειξη του πόσο κοντά στο αρχικό μοντέλο βρίσκεται το απλοποιημένο μοντέλο που έχει δημιουργηθεί.

Η μέση τιμή της *απόκλισης των κανονικών διανυσμάτων* (Normal deviation), όπου μετριέται η διαφορά των κανονικών διανυσμάτων των αντίστοιχων κορυφών. Η απόκλιση των κανονικών διανυσμάτων είναι μια σημαντική μετρική καθώς τα κανονικά διανύσματα χρησιμοποιούνται για τον υπολογισμό του φωτισμού του μοντέλου και έτσι μια μεγάλη απόκλιση στο απλοποιημένο μοντέλο μπορεί να επιφέρει σημαντικές οπτικές αποκλίσεις όταν αυτό αποδίδεται στην οθόνη.

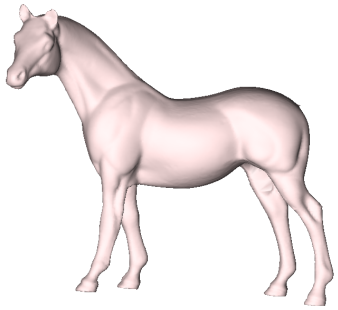
Η μέση τιμή της *ομαλότητας* (Smoothness) της επιφάνειας του κάθε μοντέλου, υπολογισμένη μέσω της απόστασης της κάθε κορυφής από το κέντρο βάρους των άμεσων γειτόνων της.

Η κατανομή των γωνιών που υπάρχουν στα τρίγωνα του μοντέλου μέσω της οποίας μπορούμε να κρίνουμε την ποιότητα των τριγώνων που παράγονται από τον κάθε αλγόριθμο. Αν στα τρίγωνα του μοντέλου υπάρχουν πολύ μικρές ή πολύ μεγάλες γωνίες, μπορεί να δημιουργηθούν προβλήματα στον περαιτέρω χειρισμό ή και στην απόδοση του.

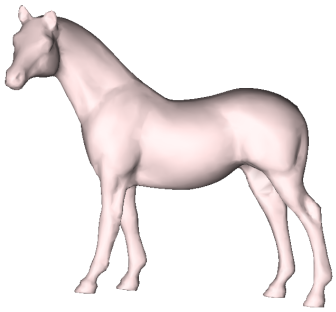
Στον πίνακα 6.2 υπάρχουν τα αποτελέσματα αυτών των μετρήσεων. Εδώ παρατηρούμε πως για κάθε απλοποίηση του μοντέλου στους βαθμούς απλοποίησης που παρουσιάζονται, η σειριακή υλοποίηση παράγει καλύτερα αποτελέσματα. Για την γεωμετρική απόκλιση και την απόκλιση των κανονικών διανυσμάτων η σειριακή υλοποίηση παράγει αποτελέσματα περίπου κατά 50% καλύτερα από την παράλληλη υλοποίηση. Η διαφορά στην ομαλότητα της επιφάνειας μεταξύ των δυο υλοποιήσεων παρότι δεν είναι τόσο μεγάλη όσο των δυο προηγούμενων μετρήσεων, παραμένει σχετικά μεγάλη. Συνολικά παρατηρούμε πως τα αποτελέσματα που παράγονται από την σειριακή υλοποίηση είναι συγκρίσιμα με αυτά που παράγονται από την παράλληλη υλοποίηση στο επόμενο επίπεδο

απλοποίησης. Παρότι υπάρχει αυτή η σχετική απόκλιση στα αποτελέσματα των μετρήσεων που λαμβάνουμε για τις μετρήσεις για τις δυο υλοποιήσεις, οι τιμές είναι της ίδιας τάξης μεγέθους και αρκετά μικρές ώστε να μην υπάρχουν σημαντικές οπτικές διαφορές στα τελικά μοντέλα.

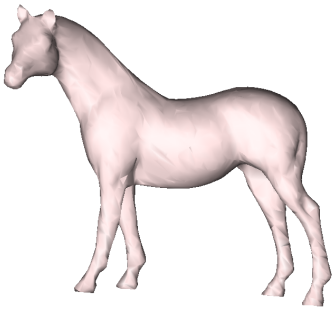
Η επίπτωση της απλοποίησης φαίνεται καλύτερα στο σχήμα 6.2 στο οποίο απεικονίζεται η γεωμετρική διαφορά και η διαφορά των κανονικών διανυσμάτων των απλοποιημένων μοντέλων σε σχέση με το αρχικό. Τα απλοποιημένα μοντέλα που χρησιμοποιήθηκαν είναι αυτά που παράγονται από τις δυο υλοποιήσεις για απλοποίηση στο 10% του αρχικού μοντέλου. Εδώ παρατηρούμε ότι η γεωμετρική απόκλιση που δημιουργείται από την σειριακή υλοποίηση επηρεάζει σχεδόν το σύνολο του μοντέλου σε αντίθεση με την παράλληλη υλοποίηση όπου η το μεγαλύτερο μέρος του μοντέλου προσεγγίζει καλύτερα το αρχικό αλλά με σημεία όπως τις οπλές του άλογου, στο εσωτερικό των πίσω ποδιών και στην άκρη των αυτιών να έχουν μεγαλύτερη διάφορα από την σειριακή υλοποίηση. Για την απόκλιση των κανονικών διανυσμάτων, αν και έχει την ίδια κατανομή και στα δυο μοντέλα, η παράλληλη υλοποίηση επιφέρει μεγαλύτερες αλλαγές στο μοντέλο από την σειριακή. Στις περιοχές όπου υπάρχει η μεγαλύτερη γεωμετρική απόκλιση (αυτιά, οπλές, εσωτερικά των πίσω ποδιών) παρατηρείται και η μεγαλύτερη απόκλιση των κανονικών διανυσμάτων.



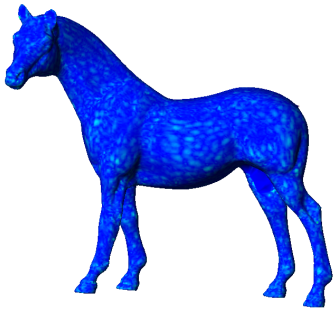
Αρχικό μοντέλο αλόγου

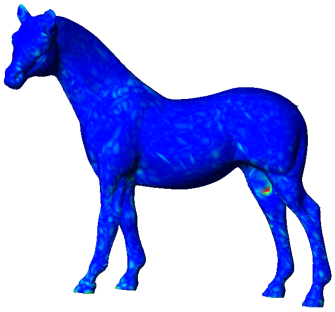


Σειριακή απλοποίηση 10%

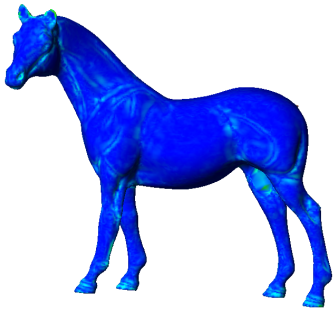


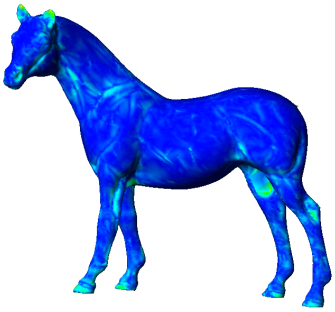
Παράλληλη απλοποίηση 10%





Γεωμετρική απόκλιση





Απόκλιση κανονικών διανυσμάτων

Σχήμα 6.2: Συγκριση αποτελεσμάτων απλοποίησης για το μοντέλο του αλόγου

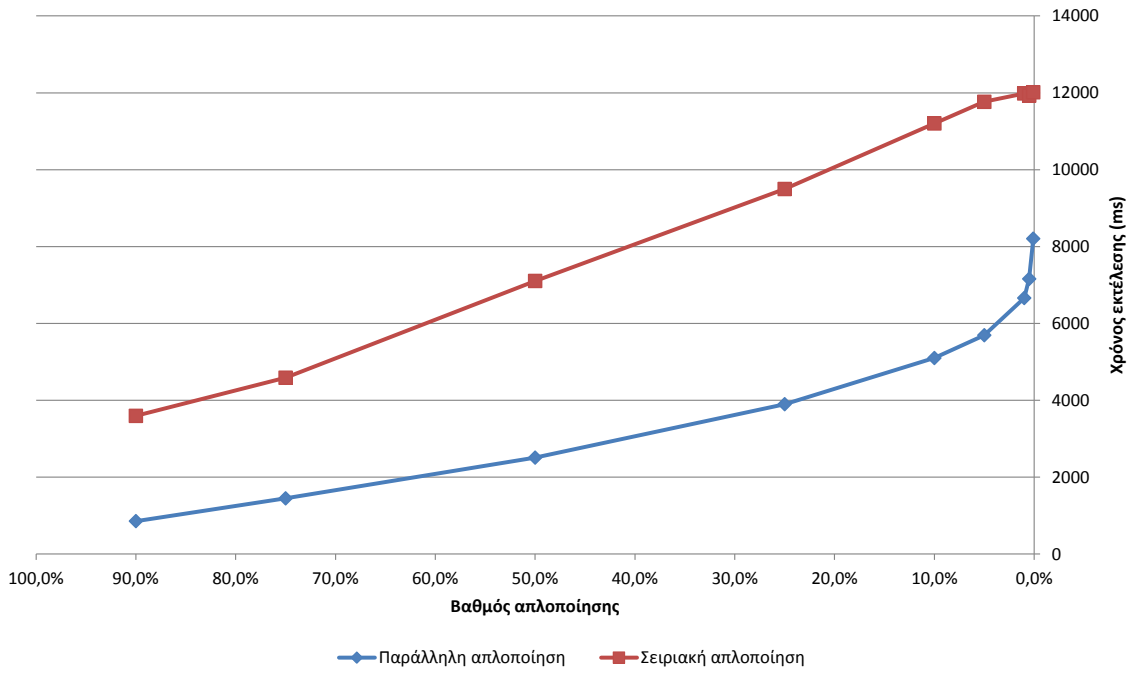
6.2 Μοντέλο υψηλού βαθμού λεπτομέρειας

Το δεύτερο μοντέλο που θα εξετάσουμε είναι αυτό του gargoyle, ένα μοντέλο με αρκετά μεγάλο βαθμό λεπτομέρειας, το οποίο αποτελείται από 500002 κορυφές και 1000000 τρίγωνα.

Στον πίνακα 6.3 παρουσιάζεται ο χρόνος εκτέλεσης των αλγορίθμων απλοποίησης. Ομοίως με το μοντέλο του αλόγου, παρατηρούμε ότι και για το μοντέλο του gargoyle ο χρόνος που χρειάζεται η παράλληλη υλοποίηση είναι σημαντικά μικρότερος για ελαφριά απλοποίηση του μοντέλου, σε αυτή την περίπτωση περίπου το 1/4 αυτού της σειριακής. Καθώς απλοποιούμε περισσότερο το μοντέλο η διαφορά στον απαιτούμενο χρόνο αρχίζει

Απλοποίηση	Κορυφές	Τρίγωνα	Παράλληλη απλοποίηση		Σειριακή απλοποίηση
			Επαναλήψεις	Χρόνος (ms)	Χρόνος (ms)
90%	450001	899998	4	858	3596
75%	375001	749998	6	1450	4588
50%	250001	499998	10	2510	7106
25%	125000	249996	18	3900	9498
10%	50002	100000	29	5101	11204
5%	25002	50000	37	5694	11766
1%	5006	10008	54	6662	11798
0.5%	2503	5002	63	7160	11946
0.1%	507	1010	81	8206	12015

Πίνακας 6.3: Χρόνος που απαιτείται (σε miliseconds) για την απλοποίηση του μοντέλου του gargoyle

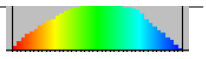
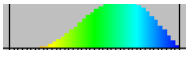
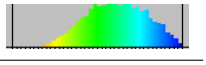


Σχήμα 6.3: Χρόνος εκτέλεσης για την απλοποίηση του μοντέλου του gargoyle από τις δύο υλοποιήσεις

να μικραίνει μέχρι το σημείο όπου για απλοποίηση στο 1% του αρχικού μοντέλου η παράλληλη υλοποίηση απαιτεί κάτι παραπάνω από το 1/2 του χρόνου που απαιτεί η σειριακή υλοποίηση. Από το αυτό το σημείο και μετά, η διάφορα αρχίζει να μικραίνει τόσο ώστε για απλοποίηση στο 0,1% του αρχικού μοντέλου η παράλληλη υλοποίηση να απαιτεί κάτι παραπάνω από τα 2/3 του χρόνου που απαιτεί η σειριακή υλοποίηση.

Σε αυτή την περίπτωση όπου απλοποιούμε ένα μοντέλο με μεγάλο βαθμό λεπτομέρειας και μέσω του διαγράμματος στο σχήμα 6.3 που προκύπτει από τον πίνακα 6.3 μπορούμε να παρατηρήσουμε καλύτερα την συμπεριφορά των δυο αλγορίθμων. Ο σειριακός αλγόριθμος ακολουθεί μια γραμμική σχέση όπου ο χρόνος απλοποίησης είναι σχεδόν απόλυτα εξαρτημένος από τον βαθμό απλοποίησης αλλά έχει αρκετά μεγάλο αρχικό κόστος. Στην παράλληλη υλοποίηση παρατηρούμε μια διαφορετική συμπεριφορά. Το αρχικό κόστος είναι αρκετά μικρότερο και ο χρόνος απλοποίησης είναι σχεδόν γραμμικά εξαρτημένος από τον βαθμό απλοποίησης μέχρι το σημείο όπου ο αριθμός των διαθέσιμων ανεξάρτητων περιοχών στο μοντέλο μειώνεται σημαντικά και ο απαιτούμενος χρόνος αυξάνει εκθετικά.

Οι μετρήσεις που αφορούν τα ποιοτικά χαρακτηριστικά των αποτελεσμάτων βρίσκονται στον πίνακα 6.4. Η συμπεριφορά του παράλληλου αλγόριθμου είναι η ίδια με αυτή που παρατηρήσαμε στην παράγραφο 6.1 όπου αναλύσαμε την συμπεριφορά του αλγορίθμου για την απλοποίηση του μοντέλου του αλόγου. Η σειριακή υλοποίηση παράγει καλύτερα αποτελέσματα από την παράλληλη με την γεωμετρική απόκλιση και την απόκλιση των κανονικών διανυσμάτων να είναι κατά 50% μικρότερη. Η διαφορά στην ομαλότητα των επιφανειών είναι μικρότερη και καθώς αυξάνεται ο βαθμός απλοποίησης η διάφορα των τιμών μικραίνει. Και εδώ όμως παρότι παρατηρείται μια σχετικά μεγάλη διαφορά μεταξύ των δύο υλοποιήσεων, οι τιμές είναι της ίδιας τάξης μεγέθους και αρκετά μικρές ώστε να μην υπάρχουν σημαντικές οπτικές διάφορες μεταξύ των αποτελεσμάτων των δυο αλγορίθμων. Όσον άφορα την ποιότητα των τριγώνων των μοντέλων, κρίνοντάς την μέσω των γωνιών τους, παρατηρούμε ότι η σειριακή υλοποίηση παράγει και εδώ καλύτερα αποτελέσματα. Η κατανομή των γωνιών της σειριακής υλοποίησης παρότι περιλαμβάνει περισσό-

Μοντέλο	Γεωμετρική απόκλιση	Απόκλιση καν. διαν.	Ομαλότητα	Γωνίες
Αρχικό	-	-	0.0803534	
Παράλληλη απλοποίηση 25%	0.0152437	0.10102	0.246738	
Σειριακή απλοποίηση 25%	0.00673284	0.0704322	0.170303	
Παράλληλη απλοποίηση 10%	0.0335727	0.157384	0.407931	
Σειριακή απλοποίηση 10%	0.014986	0.10919	0.284824	
Παράλληλη απλοποίηση 5%	0.0596091	0.208343	0.583822	
Σειριακή απλοποίηση 5%	0.0249861	0.148015	0.419823	
Παράλληλη απλοποίηση 1%	0.220996	0.348567	1.34098	
Σειριακή απλοποίηση 1%	0.0781319	0.273196	1.08336	

Πίνακας 6.4: Ποιοτικά χαρακτηριστικά μοντέλων-αποτελεσμάτων για το μοντέλο του gargoyle

τερες μικρές γωνίες, δεν έχει σχεδόν καθόλου αρκετά μεγάλες γωνίες. Από την αντίστοιχη κατανομή που παράγεται από την παράλληλη υλοποίηση, παρατηρούμε ότι η αν και η μέση τιμή πλησιάζει την ιδανική, η απόκλιση των τιμών από τη μέση τιμή είναι τέτοια ώστε να υπάρχουν αρκετές μικρές και μεγάλες γωνίες στα τρίγωνα του μοντέλου.

Στο σχήμα 6.4 παρουσιάζονται αρχικά τα απλοποιημένα μοντέλα και από τους δύο αλγορίθμους για απλοποίηση στο 10% του μοντέλου μοντέλου του gargoyle και στη συνέχεια η επίπτωση της απλοποίησης πάνω στο αρχικό μοντέλο και για τους δύο αλγορίθμους στον συγκεκριμένο βαθμό απλοποίησης. Τα μοντέλα στον συγκεκριμένο βαθμό απλοποίησης και στο μέγεθος που παρουσιάζονται φαίνονται σχεδόν πανομοιότυπα και με μικρές διαφορές από το αρχικό μοντέλο. Την παρατήρηση αυτή μπορούμε να την επιβεβαιώσουμε και από την χρωματική κωδικοποίηση της γεωμετρικής απόκλισης πάνω στο αρχικό μοντέλο και για τα δυο μοντέλα-αποτελέσματα. Και στις δυο περιπτώσεις η απόκλιση του έχουν τα τελικά μοντέλα από το αρχικό είναι πολύ μικρές, με τον παράλληλο αλγόριθμο όμως να παρουσιάζει κάποιο βαθμό απόκλισης σε σημεία με πολλές λεπτομέρειες. Όσον αφορά τα κανονικά διανύσματα παρατηρούμε ότι στα σημεία όπου υπάρχουν λεπτομέρειες, όπως τα φτερά, τον λαιμό και στα πόδια και οι δυο αλγόριθμοι επηρεάζουν το μοντέλο, με τον παράλληλο να επιφέρει περισσότερες αλλαγές στο μοντέλο. Παρά τις τροποποιήσεις που υφίσταται το αρχικό μοντέλο και από τους δυο αλγόριθμους, και όπως είδαμε και από τον πίνακα 6.4, οι αλλαγές που γίνονται είναι αρκετά μικρές ώστε να έχουν σημαντική επίπτωση στο τελικό οπτικό αποτέλεσμα.

Αρχικό μοντέλο gargoyle

Σειριακή απλοποίηση 10%

Παράλληλη απλοποίηση 10%

Γεωμετρική απόκλιση

Απόκλιση κανονικών διανυσμάτων

Σχήμα 6.4: Συγκριση αποτελεσμάτων απλοποίησης για το μοντέλο του gargoyle

Κεφάλαιο 7

Συμπεράσματα

Λαμβάνοντας υπόψιν τα χαρακτηριστικά των σύγχρονων καρτών γραφικών και με την χρήση της τεχνολογίας OpenCL που μας επιτρέπει την εκμετάλλευση των δυνατοτήτων τους, σχεδιάσαμε και υλοποιήσαμε έναν παράλληλο αλγόριθμο απλοποίησης ο οποίος βασίζεται στη συρρίκνωση ακμών και τη μετρική σφάλματος με quadrics. Με την χρήση της κάρτας γραφικών και του παράλληλου αλγόριθμου που αναπτύξαμε, επιτύχαμε την επιτάχυνση της διαδικασίας απλοποίησης τριγωνικών μοντέλων με τα τελικά αποτελέσματα να είναι ποιοτικώς συγκρίσιμα με αυτά αντίστοιχης σειριακής υλοποίησης.

7.1 Μελλοντικές κατευθύνσεις

Υπάρχουν αρκετοί τρόποι με τους οποίους η εργασία που έγινε να μπορέσει να συνεχιστεί, και από την άποψη της βελτίωσης του προγράμματος αλλά και από την εξέλιξη του αλγόριθμου απλοποίησης.

Στον αλγόριθμο που παρουσιάσαμε περιέχονται στην αρχή και το τέλος του σειριακά τμήματα τα οποία εκτελούνται από την CPU, τα οποία αφορούν τον υπολογισμό του μεγέθους του πίνακα δεικτών από τις κορυφές προς τα τρίγωνα και το φιλτράρισμα των πινάκων κορυφών και δεικτών για την ανάκτηση του απλοποιημένου μοντέλου. Μια πρώτη εξέλιξη θα ήταν να προσπαθήσουμε να τα μεταφέρουμε και αυτά στην κάρτα γραφικών. Ειδικότερα όσον αφορά τον υπολογισμό του απαιτούμενου χώρου για τους δείκτες προς τα τρίγωνα κάθε κορυφής θα μπορούσε να γίνει ίσως χρησιμοποιώντας μια τεχνική *παράλληλων αθροισμάτων* (parallel scan)[SHG08].

Όσον αφορά τον πίνακα δεδομένων στη δομή που κρατάμε τους δείκτες από τις κορυφές προς τα τρίγωνα, ο τρόπος με τον οποίο δημιουργείται και αρχικοποιείται έχει σαν αποτέλεσμα να υπάρχουν αρκετά κενά στον πίνακα που δεσμεύεται. Με μια βελτιστοποιημένη προσέγγιση με την οποία λαμβάνουμε υπόψιν το πλήθος των δεικτών κάθε κορυφής θα μπορούσαμε να μειώσουμε το απαιτούμενο ποσό μνήμης για τον πίνακα των δεδομένων.

Επίσης μπορεί να εξεταστεί η δυνατότητα ώστε πέραν του απλοποιημένου μοντέλου να παίρνουμε σαν αποτέλεσμα μια δομή progressive mesh [Hop96] η οποία επιτρέπει με κατάλληλο χειρισμό της την δυναμική δημιουργία απλοποιημένων μοντέλων στον επιθυμητό βαθμό λεπτομέρειας. Καθώς ο βασικός αλγόριθμος δημιουργίας της δομής

progressive mesh βασίζεται σε σειριακό αλγόριθμο απλοποίησης με συρρίκνωση ακμής, η μετατροπή του σε παράλληλο και η σύνδεσή του με τον υπόλοιπο αλγόριθμο απαιτεί επιπλέον τμήματα στον αλγόριθμο που έχουμε υλοποιήσει με ιδιαίτερη προσοχή στο συγχρονισμό των νημάτων εκτέλεσης και την ανεξαρτησία των δεδομένων. Μια τέτοια επέκταση θα μπορούσε να γίνει με την προσθήκη ενός ακόμα πίνακα στον οποίο θα καταγράφουμε τις απλοποιήσεις με την σειρά που εκτελούνται και τις ακμές/κορυφές/τρίγωνα που χρησιμοποιούνται σε κάθε μία. Με την χρήση μιας τέτοιας δομής κατά το τμήμα της συρρίκνωσης ακμής και στο τέλος με ένα τμήμα φιλτραρίσματος της θα μπορούσαμε να δημιουργήσουμε την δομή progressive mesh. Καθώς κατά την εκτέλεση του αλγορίθμου απλοποίησης εντοπίζουμε ανεξάρτητες περιοχές στο μοντέλο, θα μπορούσαμε ίσως να χρησιμοποιήσουμε αυτή την πληροφορία σε συνδυασμό με την δομή progressive mesh για εφαρμογές selective refinenemt (προσθήκη λεπτομέρειας επιλεκτικά σε περιοχές ενδιαφέροντος).

Τέλος, ο σχεδιασμός και η υλοποίηση του αλγορίθμου απλοποίησης έγινε με βάση τα γενικά χαρακτηριστικά των καρτών γραφικών μέσω της αρχιτεκτονικής που περιγράφεται από την OpenCL, αποκρύπτοντας κάποιες λεπτομέρειες για την πραγματική υλοποίηση τους. Αν γνωρίζουμε πως η εκτέλεση θα γίνει σε μια συγκεκριμένη κάρτα γραφικών, μπορούμε να εκμεταλλευτούμε τα ιδιαιτέρα χαρακτηριστικά της, όπως την αρχιτεκτονική της μνήμης και τον τρόπο προσπέλασής της, το μέγεθος και την θέση κρυφών μνημών και τον τρόπο που εκτελούνται οι πυρήνες από το υλικό, για επιταχύνουμε περαιτέρω την διαδικασία απλοποίησης.

Βιβλιογραφία

[ATI] ATI/AMD. Close to metal. `http://sourceforge.net/projects/amdctm/`.

[BFH+04] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for gpus: stream computing on graphics hardware. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 777–786. ACM, 2004.

[Boy08] C. Boyd. The directx 11 compute shader. *SIGGRAPH'08: ACM SIGGRAPH 2008 Classes, Beyond Programmable Shading: Fundamentals*, 2008.

[CCC+08] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Sixth Eurographics Italian Chapter Conference*, pages 129–136, 2008.

[COM98] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 115–122. ACM, 1998.

[Cro95] Thomas W. Crockett. Parallel rendering. Technical report ICASE 95-31, Institute for Computer Applications in Science and Engineering, 1995.

[CSAD04] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 905–914. ACM, 2004.

[DT07] Christopher DeCoro and Natalya Tatarchuk. Real-time mesh simplification using the GPU. In Bruce Gooch and Peter-Pike J. Sloan, editors, *Proceedings of the 2007 Symposium on Interactive 3D Graphics, SI3D 2007, April 30 - May 2, 2007, Seattle, Washington, USA*, pages 161–166. ACM, 2007.

[FS00] Martin Franc and Václav Skala. Parallel triangular mesh decimation. In *Proceedings of International Conference SCCG*, pages 164–171, 2000.

[Gar99] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, School of Computer Science Carnegie Mellon University 5000 Forbes Avenue Pittsburgh, PA 15213-3891, 1999.

[HDD+93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26. ACM, 1993.

[HL07] Jon Mikkelsen Hjelmervik and Jean-Claude Léon. GPU-accelerated shape simplification for mechanical-based applications. In *Shape Modeling International*, pages 91–102. IEEE Computer Society, 2007.

[Hop96] Huge Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM, 1996.

[Knu73] D.E. Knuth. *The art of computer programming, Vol. 3*, volume 109. Addison-Wesley, Reading, MA, 1973.

[KR88] R.M. Karp and V. Ramachandran. *A survey of parallel algorithms for shared-memory machines*. University of California, Berkeley, Computer Science Division, 1988.

[Lef05] Aaron Lefohn. Gpu memory model overview. *GPGPU: General-Purpose Computation on Graphics Hardware*, 2005.

[LT97] Kok-Lim Low and Tiow-Seng Tan. Model simplification using vertex-clustering. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 75–ff. ACM, 1997.

[LT00] P. Lindstrom and G. Turk. Image-driven simplification. *ACM Transactions on Graphics*, 19(3):204–241, 2000.

[Mun09] Aaftab Munshi. *The OpenCL specification version 1.0*. Khronos OpenCL Working Group, 2009.

[NVi10] NVidia. *NVIDIA CUDA C Programming Guide Version 3.2*. Nvidia, 2701 San Tomas Expressway, Santa Clara, CA 95050, 8 2010.

[PSHL10] Hagen Peters, Ole Schulz-Hildebrandt, and Norbert Luttenberger. Fast in-place sorting with cuda based on bitonic sort. *Parallel Processing and Applied Mathematics*, pages 403–410, 2010.

[SHG08] S. Sengupta, M. Harris, and M. Garland. Efficient parallel scan algorithms for gpus. *NVIDIA, Santa Clara, CA, Tech. Rep. NVR-2008-003*, 2008.

[SMS05] S. Silva, J. Madeira, and B.S. Santos. Polymeco-a polygonal mesh comparison tool. In *Information Visualisation, 2005. Proceedings. Ninth International Conference on*, pages 842–847. IEEE, 2005.

[SZL+92] William J. Schroeder, Jonathan A. Zarge, William E. Lorensen, et al. Decimation of triangle meshes. *COMPUTER GRAPHICS-NEW YORK-ASSOCIATION FOR COMPUTING MACHINERY-*, 26:65–65, 1992.

[TPPP08] Theoharis Theoharis, Georgios Papaioannou, Nikolaos Platis, and Nicholas M. Patrikalakis. *Graphics and Visualization: Principles & Algorithms*. AK Peters, Ltd., 2008.

Παράρτημα

Πρόγραμμα decimator

Το πρόγραμμα decimator υλοποιεί τον παράλληλο αλγόριθμο απλοποίησης που περιγράφεται στην εργασία. Η υλοποίηση του προγράμματος έγινε με προσοχή ώστε να μπορεί να μεταγλωττίζεται και να εκτελείται στα κυριότερα λειτουργικά συστήματα Windows και linux (και Mac Os X αν και δεν έχει δοκιμαστεί). Οι βιβλιοθήκες που χρησιμοποιούνται, πέραν της στάνταρ βιβλιοθήκης της C++, είναι οι παρακάτω:

- OpenGL (libGL libGLU)
- glut (libglut)
- GLEW (libGLEW)
- OpenCL (libOpenCL)

Ο κώδικας του προγράμματος υπάρχει στο συνοδευτικό CD της εργασίας. Μαζί με τον κώδικα του προγράμματος, παρέχονται και τα project files για το Visual Studio 2008 για την μεταγλώττιση του προγράμματος στα Windows και ένα makefile για την μεταγλώττισή του στο linux.

Οδηγίες για την μεταγλώττιση και την εκτέλεση του προγράμματος, αλλά και μια περιγραφή της δομής του, υπάρχουν στο αρχείο README που παρέχεται μαζί με τον κώδικα.