Μεταφορά και βελτιστοποίηση εφαρμογής ray-tracing σε πολυπύρηνη πλατφόρμα–Porting and optimization of a ray-tracing application on a manycore platform

Πανεπιστήμιο Θεσσαλίας

Χρήστος Παπαιωάννου

Μάρτιος 2015

Περιεχόμενα

1 Εισαγωγή **3**

2 Ray Tracing:Περιγραφή Αλγορίθμου και Εισαγωγή στην Cuda **5**

2.1 Τι είναι το Ray tracing ; 5
2.2 Ο Αλγόριθμος Ray tracing 6
2.3 Διαφορές με το Rasterazation 7
2.4 Ετερογενή συστήματα - CUDA 9
2.4.1 Η ανάγκη για πολυπύρηνα συστήματα 9
2.4.2 Ετερογενή συστήματα 9
2.4.3 Το μοντέλο CUDA για Nvidia GPUS 10

3 RayTracing:Στην πράξη **15**

3.1 Εύρεση Primary Rays 15
3.2 Ray-Surface Intersections 16
3.2.1 Διασταύρωση Ακτίνας - Σφαίρας 17
3.2.2 Διασταύρωση Ακτίνας - Τριγώνου 17
3.3 Μετασχηματισμοί 18
3.4 Εφέ Γραφικών 19
3.4.1 Φωτισμός 19
3.4.2 Σκιές - Shadows 25
3.4.3 Αντανάκλαση - Reflection 26

4 Acceleration Structures **29**

4.1 Δημοφιλείς Δομές 29
4.2 Bounding Volume Hierarchy(BVH) 30
4.2.1 Κατασκευή Δέντρου-Tree contruction 31
4.2.2 Διαπέραση Δέντρου-Tree traversal 34

5 Ray Tracing στη GPU **36**
5.1 Υλοποίηση 36
5.2 Μετρήσεις 41

6 Επίλογος **47**

Κεφάλαιο 1

Εισαγωγή

Τα γραφικά υπολογιστών έχουν γίνει ένα μεγάλο κομμάτι της καθημερινής μας ζωής.Η πλειοψηφία των ταινιών υψηλού budget,διαφημίσεων ακομά και καρτούν που παράγονται σήμερα χρησιμοποιούν τεχνικές γραφικών για την απεικόνιση εφέ σε πολλές από τις σκηνές τους.Πολλά πλάνα μάλιστα παράγονται αποκλειστικά με τη χρήση υπολογιστών.Παρ'όλα αυτά είναι αρκετές οι περιπτώσεις που οποιοσδήποτε μπορεί να καταλάβει ότι σε ένα πλάνο υπάρχει έλλειψη ρεαλισμού και ότι η εικόνα έχει παραχθεί από έναν ηλεκτρονικό υπολογιστή.

Οι παράγοντες που παίζουν ρόλο στο ρεαλισμό μιας εικόνας γραφικών υπολογιστών είναι πολλοί αλλά οι κυριότεροι είναι τρεις: α) animation,β) modeling και γ) rendering.Το animation και το modeling βασίζονται κυρίως στις ικανότητες του καλλιτέχνη,αλλά το rendering είναι καθαρά κομμάτι των γραφικών υπολογιστών.Η απεικόνιση ρεαλιστικών εικόνων στα γραφικά είναι ακριβή και ο υπολογισμοί για το rendering απαιτούν αρκετό χρόνο.Για παράδειγμα, ο χρόνος απεικόνισης ενός καρέ μιας ρεαλιστικής πολύπλοκης σκηνής μπορεί να διαρκέσει μερικά λεπτά ή ώρες.Ανάλογα την εφάρμογη ο χρόνος απεικόνισης μπορεί να μην παίζει σημαντικό ρόλο,λόγου χάριν στα εφέ μιας ταινίας αφού το rendering μπορεί να γίνει offline. Ωστόσο,ο χρόνος απεικόνισης μπορεί να είναι απαγορευτικός για εφαρμογές πραγματικού χρόνου όπως video games ή εφαρμογές τύπου virtual reality στις οποίες υπάρχει διαδραστικότητα με το χρήστη.

Μια τεχνική γραφικών υπολογιστών για την απεικόνιση ρεαλιστικών ψηφιακών τρισδιάστων σκηνών είναι το ray tracing.Το ray tracing προσομοιώνει τον τρόπο που λειτουργεί το φως όταν οι ακτίνες μιας πηγής φωτός φτάσουν σε μια επιφάνεια και στη συνέχεια στο ανθρώπινο μάτι ή στο φακό μιας φωτογραφικής κάμερας.Η διαδικασία μέχρι σήμερα είναι αρκετά ακριβή υπολογιστικά και δεν χρησιμοποιείται σε εφαρμογές πραγματικού χρόνου στις οποίες κυριαρχεί η

τεχνική του rasterazation.Υπάρχουν βέβαια και άλλες παρόμοιες μέθοδοι παραγωγής ρεαλιστικών εικόνων,όπως είναι το path tracing,αλλά το ray tracing είναι σήμερα η πιο διαδεδομένη.

Στα πλαίσια της διπλωματικής εργασίας αναπτύχθηκε ο αλγόριθμος του ray tracing για μια πολυπύρηνη CPU και μεταφέρθηκε σε μια κάρτα γραφικών.Για την παράλληλη υλοποίηση στον επεξεργαστή χρησιμοποείται OpenMP [Ope11] και το προγραμματιστικό μοντέλο CUDA [NVI14] της Nvidia για την κάρτα γραφικών.Επιτεύχθηκε speedup μέχρι 246x σε σχέση με την πολυνηματική υλοποίηση του επεξεργαστή και ο χρόνος απεικόνισης μειώθηκε από δεκάδες λεπτά σε μερικά δευτερόλεπτα για μεγάλες και πολύπλοκες ψηφιακές σκηνές.Η βελτιστοποίηση στη CPU έγινε με τον profiler VTune [VTu] της Intel και τον Nvidia Visual Profiler [Nvi] για τη κάρτα γραφικών.

Επισκόπηση Κειμένου

Στο κεφάλαιο 2 του κειμένου εξηγείται η μέθοδος του ray tracing και γίνεται μια μικρή εισαγωγή στο προγραμματιστικό μοντέλο CUDA.Στο κεφάλαιο 3 παρουσιάζεται με λεπτομέρια η υλοποίηση του αλγορίθμου και δύο εφέ γραφικών,σκιά και αντανάκλαση.Στο κεφάλαιο 4 αναλύεται η δομή επιτάχυνσης BVH-tree για τη βελτίωση της επίδοσης της μεθόδου.Το κεφάλαιο 5 αναφέρεται στη μεταφορά και τη βελτιστοποίηση του αλγορίθμου σε μια Nvidia GPU χρησιμοποιώντας το προγραμματιστικό μοντέλο CUDA.Επίσης παρουσιάζονται μετρήσεις επιδόσεων που συγκρίνουν το χρόνο απεικόνισης μεταξύ GPU και τη βελτιστοποιημένη πολυνηματική υλοποίηση της CPU.

Σε όλη την έκταση του παρόντος κειμένου θεωρείται ότι ο αναγνώστης έχει βασικές γνώσεις γραμμικής άλγεβρας και γραφικών υπολογιστών.Ο κώδικας της διπλωματικής εργασίας είναι ανοικτός και βρίσκεται στο παρακάτω url: `https://github.com/ChRis6/Eos`

Κεφάλαιο 2

Ray Tracing:Περιγραφή Αλγορίθμου και Εισαγωγή στην Cuda

2.1 Τι είναι το Ray tracing ;

Το ray tracing είναι μια τεχνική γραφικών για την απεικόνιση τρισδιάστατων αντικειμένων μιας εικονικής ψηφιακής σκηνής.Στον πραγματικό κόσμο οι ακτινές φωτός μιας πηγής φτάνουν σε διάφορα αντικείμενα γύρω μας και ανακλούνται προς όλες τις κατευθύσεις,φτάνοντας στον αμφιβληστροειδή χιτώνα.Στη συνέχεια ο εγκέφαλος επεξεργάζεται το οπτικό σήμα και το μετατρέπει σε εικόνα.Με παρόμοιο τρόπο δουλεύει και ο αλγόριθμος του ray tracing, με τη διαφορά οτι ακολουθεί τις ακτίνες φωτός με την αντίθετη φορά,δηλαδή από την εικόνα προς τη σκήνη και στη συνέχεια προς τις πηγές φωτός.

Ο κύριος λόγος που ο αλγόριθμος του ray tracing ακολουθεί τις ακτίνες φωτός με αντίθετη φορά είναι η επίδοση.Για να προσομοιώσουμε τον τρόπο που δουλεύει η όραση στον πραγματικό κόσμο,πρέπει να ακολουθήσουμε άπειρες ακτίνες από την πηγή φωτός,από τις οποίες η συντριπτική πλειοψηφία δεν θα φτάσει ποτέ στο ανθρώπινο μάτι ή στο φακό μιας κάμερας, που σημαίνει οτι δεν συνεισφέρουν καθόλου στη δημιούργια εικόνας.Συνεπώς θα σπαταλήσουμε πολλούς υπολογιστικούς πόρους υπολογίζοντας πολλές ακτίνες χωρίς να είναι αναγκαίο.

2.2 Ο Αλγόριθμος Ray tracing

Ο αλγόριθμος του ray tracing παράγει ρεαλιστικές εικόνες που μπορούμε να τις αποθηκεύσουμε είτε σε αρχεία εικόνων, είτε σε buffers παραθύρων.Για κάθε pixel της εικόνας εκτοξεύουμε μια ακτίνα προς την κατεύθυνση της σκηνής που θέλουμε να απεικονίσουμε.Στην συνέχεια ακολουθείται η ακτίνα μέχρι να συγκρουστεί σε κάποιο αντικείμενο(σημείο σύγκρουσης) και έπειτα φτάσει σε κάποια πηγή φωτός.Εφόσον η ακτίνα φτάσει σε μια πηγή φωτός,μπορούμε εύκολα να υπολογίσουμε το χρώμα του σημείου σύγκρουσης σύμφωνα με τις ιδιότητες του αντικειμένου(χρώμα,materials κτλ) και το χρώμα της πηγής για τη συγκεκριμένη ακτίνα.Τέλος το μόνο που μένει είναι να αποθηκεύσουμε το χρώμα του σημείου σύγκρουσης της ακτίνας για το συγκεκριμένο pixel(συνήθως σε μορφή RGB).

Algorithm 1 Ray Tracing Algorithm

```
1: procedure RayTraceScene
2:     for all pixels i in Image do
3:         ray ← calculateRayForPixel(i)
4:         colorRGB ← raytrace(ray)
5:         Image[i] ← colorRGB
6:     end for
7: end procedure
```

Οι ακτίνες ξεκινούν από ένα σημείο στον εικονικό χώρο που στην ουσία αναπαριστά τη θέση του ματιού ή του φακού μιας ψηφιακής κάμερας και ονομάζεται σημείο έναρξης.Στο σημείο έναρξης δημιουργείται ένα δισδιάστατο πλέγμα από σημεία τα οποία αναπαριστούν τη θέση του κάθε pixel στην τελική εικόνα.Η τελική διεύθυνση μιας ακτίνας για το pixel k δίνεται εύκολα από τη διαφορά του σημείου X_k και του σημείο έναρξης X_{start}, δηλαδή:

$$\vec{d}_{ray} = nomalize(\vec{X}_k - \vec{X}_{start})$$

Επίσης το διάνυσμα διεύθυνσης κανονικοποιείται έτσι ώστε να έχει μέτρο μονάδα.Αυτό θα χρησιμεύσει στη συνέχεια όταν θα είναι αναγκαίο να βρεθεί η απόσταση μεταξύ των σημείων X_{start} και $X_{intersection}$,δηλαδή του σημείου σύγκρουσης της ακτίνας.Ο αλγόριθμος περιγράφεται σε μορφή ψευδοκώδικα στον αλγόριθμο 1

Τα αντικείμενα μιας ψηφιακής σκηνής συνήθως περιγράφονται ως ένα σύνολο από γεωμετρικά σχήματα(geometric shapes), όπως επίπεδα,τρίγωνα,σφαίρες

και κύλινδροι.Το πιο δημοφιλές γεωμετρικό σχήμα από αυτά είναι το τρίγωνο, επειδή έχει πολλές ιδιότητες που κάνουν την επεξεργασία πιο εύκολη.Για παράδειγμα, οι κορυφές ενός τριγώνου σχηματίζουν ένα και μοναδικό επίπεδο στο οποίο ανήκουν όλα τα ενδιάμεσα σημεία.Αυτή η ιδιότητα δίνει τη δυνατότητα σε πολλούς αλγορίθμους να βελτιστοποιήσουν πολλές πράξεις πάνω σε τρίγωνα,κάνοντας έτσι την επεξεργασία τους πολύ γρήγορη.

Το κομμάτι που έχει ενδιαφέρον στον παραπάνω αλγόριθμο είναι η μέθοδος που ακολουθεί το μονοπάτι μιας ακτίνας, δηλαδή η μέθοδος *raytrace*.Η μέθοδος *raytrace* θα επιστρέψει το χρώμα του κοντινότερου σημείου σύγκρουσης ακτίνας-επιφάνειας προς το σημείο έναρξης X_{start}.Είναι κρίσιμο να επιστραφεί το κοντινότερο σημείο,γιατί διαφορετικά στην τελική εικόνα θα ζωγραφιστούν αντικείμενα που υπο φυσιολογικές συνθήκες θα κρύβονταν πίσω από άλλα αντικείμενα.

Algorithm 2 Ray Trace

```
1: procedure RayTrace
2:     t ← MAX
3:     for all surfaces in Scene do
4:         s ← INTERSECT ray WITH surface
5:         t ← min(t, s)
6:     end for
7:     intersectionPoint ← calculateIntersection(ray,t)
8:     colorRGB ← shade(intersectionPoint)
9:     return colorRGB
10: end procedure
```

2.3 Διαφορές με το Rasterazation

Το rasterazation είναι μια άλλη τεχνική γραφικών και χρησιμοποιείται ευρέως σήμερα σε πολλά εργαλεία απεικόνισης γραφικών,ακόμα και σε ιδεο γαμες.Η τεχνική του rasterazation επεξεργάζεται τη ψηφιακή σκηνή ανά επιφάνεια(συνήθως τρίγωνα) και χρησιμοποιεί τεχνικές τύπου depth buffering για να λύσει το πρόβλημα του βάθους σε τρισδιάστατες σκηνές.Από την άλλη, το ray tracing επεξεργάζεται την ψηφιακή σκηνή ανα pixel ή πιο σωστά ανά ακτίνα.

Συνεπώς,η πολυπλοκότητα του rasterazation είναι γραμμική προς τον αριθμό των τριγώνων μιας σκηνής,ενώ στο ray tracing είναι γραμμική προς τον

Algorithm 3 Rasterazation

```
1: procedure RASTERIZE
2:     for all surfaces in Scene do
3:         pixels ← findPixels(surface)
4:         shade(pixels,surface.properties)
5:         Store pixels in Image
6:     end for
7: end procedure
```

αριθμό των pixels της τελικής εικόνας.Ουσιαστικά όμως και οι δύο αλγόριθμοι κάνουν την ίδια δουλειά αλλά με διαφορετικά βήματα εκτέλεσης.

Το rasterazation κατάφερε να γίνει δημοφιλές λόγω της απλότητας του αλγορίθμου και κυρίως λόγω της ταχύτητας επεξεργασίας.Τα τελευταία χρόνια αναπτύχθηκε ειδικά σχεδιασμένο hardware(GPUs) που υλοποιεί τον αλγόριθμο του rasterazation, το οποίο έφερε επανάσταση στο χώρο των γραφικών.Σήμερα μπορούμε να ζωγραφίζουμε εκατομμύρια τρίγωνα μιας ψηφιακής σκηνής κατά τη διάρκεια ενός καρέ λόγω της ταχύτητας που προσφέρει το rasteraztion.Κάτι τέτοιο είναι δυνατό να συμβεί και με τη τεχνική του ray tracing.Σύγχρονες προγραμματιζόμενες κάρτες γραφικών μας δίνουν τη δυνατότητα να υλοποιήσουμε αλγορίθμους απεικόνισης γραφικών όπως το ray tracing και να πάρουμε μεγάλη επίδοση σαν το rasterazation.Βέβαια ακόμη το ray tracing δεν έχει φτάσει την ταχύτητα που προσφέρει το rasterazation, αλλά καθημερινά δημοσιεύονται καινούριοι αλγόριθμοι και άρθρα που αποσκοπούν στην βελτίωση της επίδοσης του.

Παρ΄όλο που η επίδοση και η ταχύτητα επεξεργασίας του rasterazation είναι το μεγάλο πλεονέκτημα έναντι του ray tracing, είναι ταυτόχρονα και μεγάλο μειονέκτημα από άποψη ποιότητας απεικόνισης.Ακριβώς επειδή το rasterazation δουλεύει με κάθε τρίγωνο ξεχωριστά,είναι αδύνατο απεικονιστούν ρεαλιστικά εφέ.Για παράδεγμα,με το rasterazation είναι αδύνατο παράξουμε ρεαλιστικές σκιές και αντανακλάσεις σύμφωνα με τη γεωμετρία της ψηφιακής σκηνής και καταφεύγουμε σε λύσεις που δεν προσφέρουν ακρίβεια,όπως χάρτες σκιών (shadow maps).Από την άλλη τέτοια εφέ γραφικών είναι εύκολα να παραχθούν με τη τεχνική του ray tracing,πολλά από οποία περιγράφονται στο επόμενο κεφάλαιο.

2.4 Ετερογενή συστήματα - CUDA

2.4.1 Η ανάγκη για πολυπύρηνα συστήματα

Μέχρι τα μέσα του 2000 η ωμή επίδοση ήταν ο στόχος των κατασκευαστών επεξεργαστών.Τα συστήματα ενός πυρήνα κυριαρχούσαν στην αγορά και όπως είναι φυσικό η πλειοψηφία του software που κυκλοφορούσε ήταν γραμμένο και ραμμένο στα μέτρα τέτοιων συστημάτων.Στις περιπτώσεις βαριών επεξεργαστικά υπολογισμών δύο επιγολές είχε κανείς: α) να βελτιστοποιήσει αλγοριθμικά τον υπολογισμό της εφαρμογής είτε, β) να περιμένει μερικούς μήνες μέχρι να βγει καινούριος επεξεργαστής που να είναι σημαντικά πιο γρήγορος.Όταν δεν ήταν δυνατό να βελτιωθεί αλγοριθμικά ο υπολογισμός,κάτι που είναι συχνό ακόμα και σήμερα,η λύση ήταν η αναμονή για καινούριο σύστημα.

Σχεδόν κάθε 18 μήνες έβγαινε καινούριο μοντέλο επεξεργαστή με έξυπνες βελτιώσεις στην αρχιτεκτονική,σύμφωνα με τον νόμο του Moore [Sch97]. Ο κύριος παράγοντας βελτίωσης των επιδόσεων ενός επεξεργαστή ήταν η αύξηση του ρυθμού ρολογιού του συστήματος.Αυτό απαιτούσε την ανάγκη για μεγαλύτερη προσφορά ενέργειας στο τσιπ του επεξεργαστή που με τη σειρά του ζητούσε επαρκή ψύξη λόγω της απαγωγής θερμότητας των κυκλωμάτων.Έτσι διαπιστώθηκε οτι το ποσό απαιτούμενης ενέργειας λειτουργίας ενός επεξεργαστή με πολλά τρανζίστορ και μεγάλο ρυθμό ρολογιού θα είναι απαγορευτική τα επόμενα χρόνια,αφού θα έφτανε σε σημαντικά υψηλά επίπεδα.Τα ρολόγια των επεξεργαστών θα έφταναν σε ένα άνω όριο με αποτέλεσμα να μην ανεβαίνει με ανάλογο τρόπο και η επίδοση.

Έτσι το 2005 βγήκαν μαζικά στην αγορά τα πρώτα πολυπύρηνα σύστηματα από τις intel και amd.Συγκεκριμένα τα μοντέλα amd athlon 64 X2 και intel pentium D είχαν δύο όμοιους πυρήνες το καθένα.Με αυτό τον τρόπο οι κατασκευαστές κατάφεραν να μειώσουν τον ρυθμό ρολογιού του επεξεργαστή και ταυτόχρονα να κρατήσουν ίδιες επιδόσεις.Με άλλα λόγια,η επίδοση παρέμενε στα ίδια επίπεδα αφού πλέον κάθε πυρήνας μπορούσε να εκτελεί υπολογισμούς παράλληλα.

2.4.2 Ετερογενή συστήματα

Σήμερα πολλά σύγχρονα υπολογιστικά συστήματα διαθέτουν εκτός από CPUs πολλά ακόμη υπολογιστικά εξαρτήματα,όπως οι κάρτες γραφικών(GPUs).Οι GPUs σχεδιάστηκαν ως ειδικά σχεδιαμένα συστήματα υπολογισμού για απεικόνιση γραφικών υπολογιστών.Μια σύγχρονη κάρτα γραφικών,όπως είναι η nvi-

dia GTX980 διαθέτει σχεδόν 2000 υπολογιστούς πυρήνες.Συνεπώς είναι επιθυμητό να μπορέσουμε να χρησιμοποιήσουμε έναν τόσο μεγάλο αριθμό πυρήνων όχι μονο για γραφικά υπολογιστών αλλά και για παράλληλους υπολογισμούς γενικού σκοπού,που μέχρι σήμερα γίνονταν μόνο στην CPU.Οι κατασκευαστές συνειδητοποίησαν την παραπάνω ανάγκη και το μέγεθος της ζήτησης της αγοράς και δημιούργησαν προγραμματιστικά μοντέλα για την εκμετάλευση τέτοιων υπολογιστικών συστημάτων.

Ένα προγραμματιστικό μοντέλο για κάρτες γραφικών είναι η CUDA [NVI14].Η CUDA σχεδιάστηκε από την nvidia για τις κάρτες γραφικών της εταιρίας και δίνει στους προγραμματιστές τα κατάλληλα εργαλεία για να μπορούν να αξιοποιούν κάρτες γραφικών της Nvidia κάνοντας υπολογισμούς γενικού σκοπού.

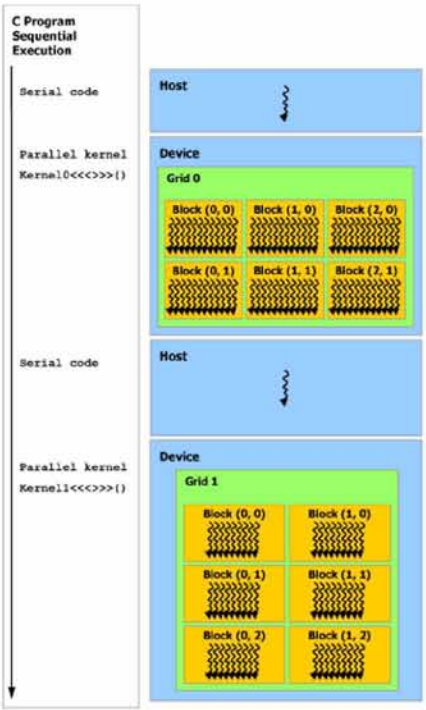
2.4.3 Το μοντέλο CUDA για Nvidia GPUS

Οι σύγχρονες κάρτες γραφικών αποτελούν συστήματα μαζικών παράλληλων υπολογισμών.Μια τυπική GPU διαθέτει παραπάνω από 1000 πυρήνες για υπολογισμούς γενικού σκοπού.Το προγραμματιστικό μοντέλο CUDA παρέχει ότι χρειάζεται ένας προγραμματιστής για να αναπτύξει εφαρμογές που εκτελούν υπολογισμούς σε μια κάρτα γραφικών.Ο κώδικας που γράφεται για τη GPU γράφεται σε CUDA C και αποτελεί επέκταση της γλώσσας C.

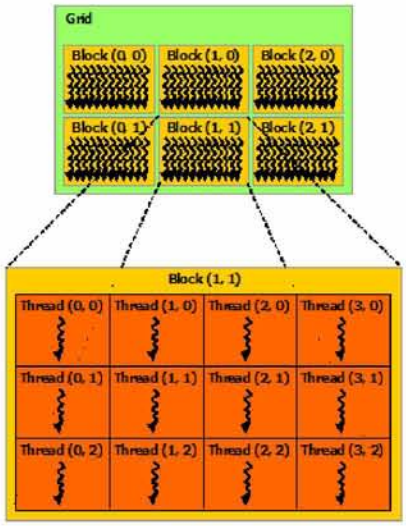
Μια συνηθισμένη ροή εκτέλεσης ενός υπολογισμού είναι αυτή που φαίνεται στην εικόνα 2.1.Ο επεξεργαστής(CPU) του συστήματος συνήθως αρχικοποιεί τον υπολογισμό,καλεί τον κατάλληλο κώδικα που τρέχει στην κάρτα γραφικών και παίρνει τα αποτελέσματα.Λόγω της σχεδίασης της αρχιτεκτονικής μιας κάρτας γραφικών,είναι αναγκαίο να χωρίζουμε το σειριακό κομμάτι του υπολογισμού από το παράλληλο και να κατανέμεται το σειριακό στον επεξεργαστή και το παράλληλο στην κάρτα γραφικών.

Αρχιτεκτονική

Στο επίπεδο του hardware η αρχιτεκτονική μιας NVIDIA CUDA-ENABLED GPU χωρίζεται σε ένα σύνολο από πολυεπεξεργαστικά κομμάτια που ονομάζονται Streaming multiproccessors(SM).Όταν μια εφαρμογή στη CPU ζητήσει η εκτέλεση να μεταφερθεί στην GPU, δηλαδή να εκτελέσει ένα kernel grid,ο hardware scheduler κατανέμει τον υπολογισμό στα SMs ανάλογα την διαθεσιμότητα εκτέλεσης κάθε SM.Κάθε νήμα εκτέλεσης που θα τρέξει στην κάρτα γραφικών ομαδοποιείται σε blocks από νήματα.Τα νήματα ενός block εκτελούνται παράλληλα σε ένα SM και ένας SM μπορεί να τρέξει πολλά thread blo-



Σχήμα 2.1: Τυπική Ροή ενός υπολογισμού στην cuda [NVI14]



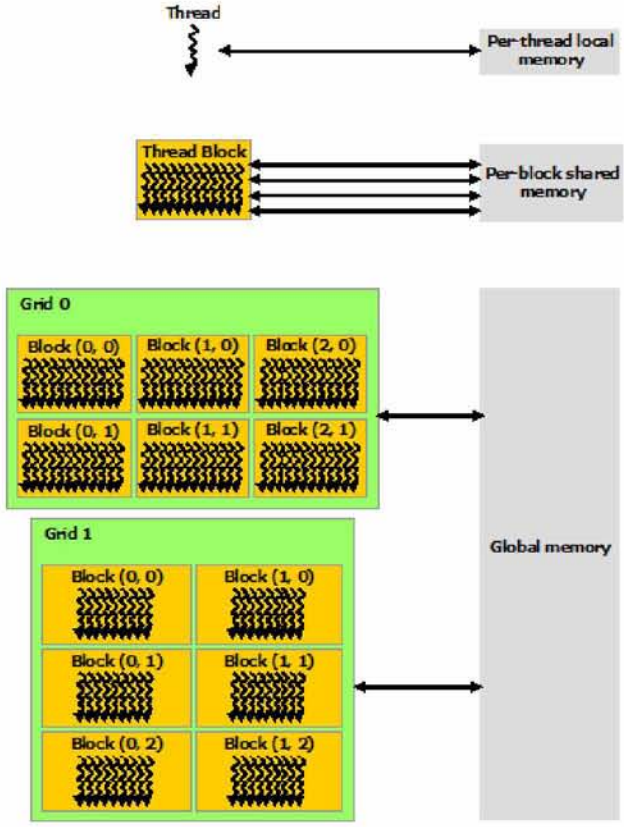
Σχήμα 2.2: Οργάνωση νημάτων σε blocks [NVI14]

cks.Όταν ένα block από threads τελειώσει την εκτέλεση που του αναλογεί,ένα καινούριο block ανατίθεται στο αντίστοιχο SM μέχρις ότου να μην υπάρχουν άλλα διαθέσιμα blocks προς εκτέλεση,δηλαδή με άλλα λόγια να έχει τελειώσει ο kernel.

Το μοντέλο εκτέλεσης

Τα νήματα στην κάρτα γραφικών οργανώνονται και τρέχουν σε γκρουπ των 32 που ονομάζονται warps.Κάθε warp μοιράζεται το ίδιο Control Unit και κάθε νήμα διαθέτει το δικό του APU.Γιάυτό το λόγο είναι σημαντικό τα νήματα που βρίσκονται στο ίδιο warp να εκτελούν την ίδια εντολή μηχανής.Σε περιπτώσεις διακλαδώσεων που ένας αριθμός νήματων από τα 32 αποφασίσει να εκτελέσει διαφορετικό κομμάτι κώδικα παρατηρείται warp divergence.Τα νήματα που πρέπει να εκτελέσουν διαφορετική διακλάδωση θα περιμένουν τα υπόλοιπα νήματα του warp να τελειώσουν τη δική τους διακλάδωση,προτού τρέξουν το δικό τους κομμάτι κώδικα.

Είναι σημαντικό ο προγραμματιστής να προσέξει κάθε νήμα ενός warp να εκτελεί ανα πάσα στιγμή το ίδιο κομμάτι κώδικα.Ο κύριος λόγος είναι η επίδοση.Αν για παράδειγμα τα μισά νήματα ενός warp αποφασίσουν να εκτελέσουν την διακλάδωση A και τα υπόλοιπα 16 την διακλάδωση B,τότε αυτό το warp θα χρησιμοποιήσει μόνο τη μισή επεξεργαστική δύναμη της GPU.Αν αυτό ισχύει



Σχήμα 2.3: Οργάνωση μνήμης σε μια nvidia GPU [NVI14]

για όλα τα warps όλων των blocks ενός kernel,τότε ο κώδικας χρησιμοποιεί μόνο το 50% της GPU.Στην ακραία περίπτωση που κάθε νήμα εκτελεί διαφορετική διακλάδωση,ο κώδικας χρησιμοποιεί μόνο το 3% της επεξεργαστικής ικανότητας της κάρτας γραφικών.

Το παραπάνω ισχύει μόνο στις περιπτώσεις που τα νήματα του ίδιου warp πάρουν διαφορετικό μονοπάτι.Αν τα νήματα του warp 1 εκτελέσουν τη διακλάδωση A και τα νήματα του warp 2 την διακλάδωση B,τότε δεν υπάρχει divergence.Warp divergence παρατηρείται μόνο στα νήματα του ίδιου warp,δηλαδή στα νήματα που ανήκουν στην ίδια 32αδα.

Το μοντέλο μνήμης

Στην εικόνα 2.3 φαίνεται το μοντέλο μνήμης μιας Nvidia κάρτας γραφικών.Τα κύρια μέρη της ιεραρχίας μνήμης είναι: α)registers, β)shared memory, γ) global memory.

Registers: Οι καταχωρητές είναι ιδιωτικοί για κάθε νήμα.

Local Memory: Η τοπική μνήμη είναι ιδιωτική μνήμη για κάθε νήμα.Συνήθως στην τοπική μνήμη αποθηκεύονται δεδομένα που δεν γίνεται να μπουν σε καταχωρητές για κάθε νήμα,όπως για παράδειγμα τοπικοί πίνακες αλλά και τοπικές μεταβλητές που δεν αποθηκεύονται σε καταχωρητές λόγω περιορισμού καταχωρητών.Σε μια τέτοια κατάσταση γίνεται λόγος για register spilling.

Shared memory: Η shared memory είναι κοινή μνήμη για όλα τα νήματα που βρίσκονται στο ίδιο block.Έχει χαμηλό latency πρόσβασης και συχνά χρησιμοποιείται για τη συνεργασία νημάτων του ίδιου block.

Global memory: Η καθολική μνήμη είναι η κύρια μνήμη της κάρτας γραφικών.Όλα τα νήματα ενός kernel έχουν πρόσβαση στη global memory.Στις σύγχρονες GPU η πρόσβαση στη κύρια μνήμη είναι cached.

Texture Memory: Χρησιμοποιείται για την πρόσβαση textures σε ένα kernel.Είναι cached.

Constant Memory: Μνήμη σταθερών.Είναι χρήσιμη στις περιπτώσεις που κάθε νήμα διαβάζει από την ίδια διεύθυνση.Είναι cached.

Host (CPU) memory: Βρίσκεται εκτός κάρτας γραφικών και δεν είναι προσβάσιμη από τη GPU.

Κεφάλαιο 3

RayTracing:Στην πράξη

Σε αυτό το κεφάλαιο παρουσιάζεται η υλοποίηση του αλγορίθμου ray tracing για την απεικόνιση ψηφιακών τρισδιάστατων,οι βασικοί αλγόριθμοι διασταύρωσης ακτίνας - επιφάνειας, μερικά εφέ γραφικών ,τεχνικές φωτισμού καθώς επίσης και μετασχηματισμοί επιφανειών.

3.1 Εύρεση Primary Rays

Μια ακτίνα περιγράφεται γεωμετρικά ως μια ημιευθεία,δηλαδή έχει ένα αρχικό σημείο $\vec{o}$ ένα διάνυσμα διεύθυνσης $\vec{d}$.Η παραμετρική εξίσωση μιας ακτίνας είναι η παρακάτω:

$$r = \vec{o} + t * \vec{d}, t > 0 \tag{3.1}$$

Η παράμετρος t είναι βαθμωτή και περιγράφει το μήκος της ακτίνας.Ο περιορισμός για $t > 0$ υπάρχει έτσι ώστε η ακτίνα να εκτείνεται μόνο προς μια κατεύθυνση.Όταν η παράμετρος t είναι αρνητική,τότε το σημείο βρίσκεται ¨πίσω' από το αρχικό σημείο $\vec{o}$ της ακτίνας.Στις περισσότερες περιπτώσεις μας ενδιαφέρει μόνο μια κατεύθυνση της ακτίνας,γιάυτό υπάρχει και ο περιορισμός.Για παράδειγμα,όταν υπολογίζονται οι πρώτες ακτίνες από την κάμερα,μας ενδιαφέρει μόνο μια κατεύθυνση,η κατεύθυνση που είναι στραμένη η κάμερα.

Στον αλγόριθμο του ray tracing, η ψηφιακή κάμερα περιγράφεται από τέσσερα διανύσματα:

- $\vec{origin}$:Το σημείο της κάμερας στον ψηφιακό τρισδιάστατο χώρο.
- $\vec{up}$: Το διάνυσμα που δείχνει προς το πάνω μέρος μιας κάμερα.

- $\vec{right}$: Το διάνυσμα που δείχνει προς τα δεξιά της κάμερας.
- $\vec{view}$: Το διάνυσμα που δείχνει την κατεύθυνση που στρέφεται η κάμερα.

Γνωρίζοντας τα παραπάνω διανύσματα μπορούμε να δημιουργήσουμε ακτίνες για κάθε ένα pixel της τελικής εικόνας.Τα διανύσματα της κάμερας ορίζουν το δισδιάστατο πλέγμα που αντιστοιχεί σημεία στον τρισδιάστατο χώρο στα pixels της εικόνας.

Για να είναι ακτινές ανεξάρτητες από την ανάλυση της εικόνας που ζωγραφίζουμε,είναι αναγκαίο τα δείγματα μιας εικόνας να μετατραπούν από το 2D πλέγμα $(0,0)-(WIDTH,HEIGHT)$ στο $(-0.5,-0.5)-(0.5,0.5)$.Άρα το pixel (i,j) μετατρέπεται στις παραπάνω συντεταγμένες,που ονομάζονται κανονικές συντεταγμένες (normalized coordinates) με τον παρακάτω τρόπο:

$$(i_{norm},j_{norm})=(\frac{i-\frac{width}{2}}{width},\frac{j-\frac{height}{2}}{height})$$

Έπειτα η διεύθυνση της ακτίνας για το δείγμα (i,j) υπολογίζεται σύμφωνα με τα διανύσματα χώρου της κάμερας:

$$\vec{d}=i_{norm}*\vec{right}+j_{norm}*\vec{up}+\vec{view} \tag{3.2}$$

Τέλος το διάνυσμα $\vec{d}$ κανονικοποιείται έτσι ώστε να έχει μέτρο μονάδα.Είναι σημαντικό το μέτρο να είναι μονάδα,γιατί στη συνέχεια που θα χρησιμοποιηθεί για τη διασταύρωση ακτίνας επιφάνειας,η παράμετρος t της εξίσωσης 3.1.Σημειώνεται ότι όλες οι ακτίνες έχουν το ίδιο σημείο έναρξής $\vec{o}$,που ισοδυναμεί με το σημείο της ψηφιακής κάμερας στον τρισδιάστατο εικονικό χώρο.Το διάνυμα $\vec{d}$ της εξίσωσης 3.2 είναι διαφορετικό για κάθε pixel της τελικής εικόνας.

3.2 Ray-Surface Intersections

Σε αυτό το κομμάτι παρουσιάζονται αλγόριθμοι για τη διασταύρωση ακτίνας επιφάνειας.Οι αλγόριθμοι είναι γενικού σκοπού και δεν χρησιμοποιούνται μόνο στο ray tracing.Κάθε μέθοδος ελέγχει αν η ακτίνα ¨ακουμπά' την επιφάνεια που εξετάζεται και επιστρέφει δεδομένα που χρειάζονται στη συνέχεια του αλγορίθμου.Τέτοια δεδομένα είναι συνήθως η απόσταση d του αντικειμένου από το αρχικό σημείο μιας ακτίνας,το κανονικοποιημένο διάνυσμα που είναι κάθετο στην επιφάνεια (normal vector) που χρησιμοποιείται στη φάση του φωτισμού

της επιφάνειας και το υλικό της επιφάνειας(material) που ορίζει τις ιδιότητες της(χρώμα,ποσοστό ανάκλασης,ποσοστό διάθλασης).Είναι κρίσιμο οι μέθοδοι διασταύρωσης να είναι υπολογιστικά αποδοτικοί,γιατί αποτελούν την καρδιά του ray tracing.

3.2.1 Διασταύρωση Ακτίνας - Σφαίρας

Μια σφαίρα περιγράφεται από ένα διάνυσμα που αναπαριστά το κέντρο της σφαίρας $\vec{p_c}$ και μια βαθμωτή μεταβλητή για την ακτίνα:

$$(\vec{p} - \vec{p_c}) * (\vec{p} - \vec{p_c}) - r^2 = 0 \tag{3.3}$$

Αντικαθιστώντας την εξίσωση της ακτίνας(3.1) στην εξίσωση της σφαίρας παίρνουμε:

$$(t * \vec{d} + \vec{o} - \vec{p_c}) * (t * \vec{d} + \vec{o} - \vec{p_c}) - r^2 = 0$$

Λύνουμε ως προς t και έχουμε:

$$\vec{d} * \vec{d} * t^2 + 2 * \vec{d} * (\vec{o} - \vec{p_c}) * t + (\vec{o} - \vec{p_c}) * (\vec{o} - \vec{p_c}) - r^2 = 0$$

Η παραπάνω είναι δευτεροβάθμια εξίσωση με άγνωστο την παράμετρο t.

$$a = \vec{d} * \vec{d} \tag{3.4}$$

$$b = 2 * \vec{d} * (\vec{o} - \vec{p_c}) \tag{3.5}$$

$$c = (\vec{o} - \vec{p_c}) * (\vec{o} - \vec{p_c}) - r^2 \tag{3.6}$$

Μια δευτεροβάθμια εξίσωση έχει λύση όταν η διακρίνουσα είναι μεγαλύτερη ή ίση του μηδενός.Αν η διακρίνουσα είναι αρνητική,τότε η ακτίνα δεν διασταυρώνεται με τη σφαίρα.Αν υπάρχει λύση,τότε επιστρέφεται η μικρότερη από τις δύο με την προυπόθεση οτι είναι μεγαλύτερη από το μηδέν.Αρνητική ρίζα σημαίνει ότι το t της ακτίνας είναι αρνητικό,που έχει ως αποτέλεσμα η σφαίρα να βρίσκεται ¨πίσω' από το αρχικό σημείο της ακτίνας.

3.2.2 Διασταύρωση Ακτίνας - Τριγώνου

Μια πολύ γνωστή μέθοδος διασταύρωσης ακτίνας - τριγώνου είναι η αυτή των Möller–Trumbore [Tru97].Θεωρείται μέχρι σήμερα η ταχύτερη μέθοδος διασταυρώσεως ακτίνας - τριγώνου και εύκολα μπορεί να μετατραπεί έτσι ώστε να είναι αποδοτική στη GPU,δηλαδή χωρίς να έχει διακλαδώσεις.

3.3 Μετασχηματισμοί

Οι μετασχηματισμοί είναι ένα κρίσιμο κομμάτι των γραφικών υπολογιστών.Χωρίς μετασχηματισμούς δεν θα ήταν δυνατό να έχουμε δυναμικές σκηνές,animation και γενικότερα η αλλελεπίδραση των αντικειμένων θα ήταν σημαντικά μειωμένη.Σε αυτό το κομμάτι παρουσιάζεται μια μέθοδος που προσφέρει αποδοτικούς μετασχηματισμούς για τον αλγόριθμο του ray tracing.

Μια συνηθησμένη μέθοδος μετασχηματισμών είναι αυτή που χρησιμοποιεί η OpenGL [Gro].Η CPU ορίζει τους πίνακες μετασχηματισμών των σημείων προς απεικόνιση και τους στέλνει στο κατάλληλο σημείο του pipeline.Οι μετασχησματισμοί στην OpenGL γίνονται συνήθως στον vertex shader,δηλαδή στο κομμάτι του pipeline που επεξεργάζεται κάθε κορυφή ξεχωριστά,προτού γίνει αποκοπή(clipping).Άρα είναι φανερό οτι στην παραπάνω περίπτωση οι μετασχηματισμοί εφαρμόζονται για κάθε κορυφή ξεχωριστά.

Η παραπάνω προσέγγιση δεν είναι αποδοτική για το ray tracing.Θυμηθείτε η μέθοδος raytrace(Αλγόριθμος 2) ελέγχει για διασταύρωση κάθε επιφάνεια ξεχωριστά για κάθε ακτίνα.Με άλλα λόγια,θα πρεπει για κάθε δείγμα της εικόνας να μετασχηματιστούν όλες οι επιφάνειες,δηλαδή όλες οι κορυφές της σκηνής.Αν ο αριθμός των κορυφών είναι αρκετά μεγαλός,για παράδειγμα μια τυπική σκηνή περιέχει κορυφές της τάξης του εκατομμύριου,η παραπάνω μέθοδος δεν είναι καλή προσέγγιση.

Συνεπώς σκοπός είναι να αποφύγουμε με κάποιο τρόπο να μετασχηματίσουμε κάθε κορυφή της σκηνής.Αν μετασχηματίσουμε την ακτίνα αντί για τις πραγματικές κορυφές της σκηνής με τον αντίστροφο μετασχηματισμό,τότε θα πετύχουμε το ίδιο αποτέλεσμα.Μια ακτίνα μεταχηματίζεται αν εφαρμόσουμε τον αντίστροφο μετασχηματισμό των κορυφών στο αρχικό σημείο και τη διεύθυνση της ακτίνας.

$$rayOrigin_{local} = M^{-1} * rayOrigin_{camera} \tag{3.7}$$

$$rayDirection_{local} = M^{-1} * rayDirection_{camera} \tag{3.8}$$

Με τον παραπάνω τρόπο οι ακτίνες μετασχηματίζονται στις τοπικές συντεταγμένες της επιφάνειας.Συνεπώς μπορούμε να χρησιμοποιήσουμε μια από τις παραπάνω μεθόδους διασταυρώσεως ακτίνας-επιφάνειας,αφού η ακτίνα και η επιφάνεια προς εξέταση βρίσκονται στο ίδιο σύστημα συντεταγμένων.

3.4 Εφέ Γραφικών

Στην παρακάτω ενότητα παρουσιάζονται διάφορα εφέ γραφικών που υλοποιούνται σύμφωνα με τον αλγόριθμο του ray tracing.Τα κύρια σημεία είναι η παραγωγή σκιών,αντανακλάση,διάθλαση και το σημαντικότερο ο φωτισμός επιφανειών που ακολουθεί.

3.4.1 Φωτισμός

Ο φωτισμός επιφανειών ανέκαθεν αποτελούσε ζήτημα συζητήσεων και ερευνών.Είναι το σημαντικότερο κομμάτι οποιουδήποτε αλγορίθμου απεικόνισης γραφικών γιατί αποτελεί την καρδιά της απεικόνισης.Ο φωτισμός μιας επιφάνειας καθορίζει κατα πόσο ρεαλιστική θα είναι η απεικόνιση μιας ψηφιακής εικονικής σκηνής γραφικών υπολογιστών.Χωρίς φωτισμό η απεικόνιση θα ήταν ανιαρή και το κυριότερο δεν θα ήμασταν σε θέση να διακρίνουμε ορθά τα αντικείμενα της σκηνής.

Αρχικά πρέπει να οριστεί μια πηγή φωτός στην ψηφιακή σκηνή που θα φωτίσει και θα δώσει χρώμα στις επιφάνειες.Μια ψηφιακή σκηνή που δεν διαθέτει πηγή φωτός θα είχε ως αποτέλεσμα στον αλγόριθμο του ray tracing μια εντελώς μαύρη εικόνα.Μια πηγή φωτός αναπαρίσταται από μια τρισδιάστατη κορυφή που δηλώνει τη θέση της στον τρισδιάστο χώρο.Ανάλογα με τη μορφή της πηγής χρειάζονται και διαφορετικοί παράμετροι που να την περιγράφουν.Οι πιο γνωστές μορφές πηγής φωτός είναι οι παρακάτω:

- Πηγή Σημείου.Αυτή η πηγή περιγράφεται από ένα μοναδικό σημείο και στέλνει τις ακτίνες της προς όλες τις κατευθύσεις στο χώρο.Μια λάμπα εσωτερικού χώρου περιγράφεται με μια τέτοια σκηνή.

- Πηγή Κατεύθυνσης.Αυτή η πηγή διαθέτει μόνο ένα διάνυσμα κατεύθυνσης και στέλνει όλες τις ακτίνες της παράλληλα προς αυτήν την κατεύθυνση.Ο ήλιος του ηλιακού συστήματος μπορεί θεωρηθεί μια τέτοια πηγή.

- Πηγή κέντρου προσοχής.Ένας ηλεκτρικός φακός περιγράφεται από μια τέτοια πηγή.

Η απλούστερη μορφή πηγής για τα γραφικά υπολογιστών είναι η πηγή κατεύθυνσης.Κατα τη διάρκεια υπολογισμού του φωτισμού μιας επιφάνειας,όπως περιγράφεται παρακάτω,ο σημαντικότερος παράγοντας είναι η κατεύθυνση των ακτίνων φωτός.Στην υλοποίηση του ray tracing χρησιμοποίησα μόνο πηγές σημείου.

Για τον υπολογισμό του φωτισμού μιας επιφάνειας χρειάζονται οι εξής πληροφορίες: α) το κανονικοποιημένο διάνυσμα (normal vector) που είναι κάθετο στην επιφάνεια,β) το σημείο διασταύρωσης ακτίνας-επιφάνειας, γ) το σημείο της πηγής και δ) οι ιδιότητες του υλικού της επιφάνειας(material).Το σημείο διασταύρωσης ακτίνας επιφάνειας υπολογίζεται εύκολα από την εξίσωση της ακτίνας (3.1) αφού ο παράγοντας t είναι γνωστός και έχει υπολογιστεί από τη μέθοδο διασταύρωσης ακτίνας-επιφάνειας.Το κάθετο διάνυσμα υπολογίζεται εύκολα για τις παρακάτω επιφάνειες:

- Σφαίρα.Το κανονικό διάνυμα υπολογίζεται ως:

$$\vec{normal}_{sphere} = normalize(\vec{p_{intersction}} - \vec{origin}_{sphere})$$

- Τρίγωνο.

$$\vec{normal}_{tri} = normalize((\vec{v2} - \vec{v1}) \times (\vec{v3} - \vec{v1}))$$

  Τα διανύσματα $\vec{v1}$,$\vec{v2}$,$\vec{v3}$ είναι οι κορυφές του τριγώνου.Με τον παραπάνω τρόπο όμως κάθε σημείο της επιφάνειας του τριγώνου θα έχει το ίδιο normal vector κατά τη φάση του φωτισμού.Σε αυτή την περίπτωση γίνεται λόγος για flat shading.Αν δίνεται και ένα normal vector για κάθε κορυφή του τριγώνου(συνήθως από το αρχείο εισόδου),τότε μπορούμε να χρησιμοποιήσουμε γραμμική παρεμβολή σύμφωνα με τις barycentric coordinates του τριγώνου για να βρουμε το κανονικό διάνυσμα του σημείου διασταύρωσης ακτίνας-τριγώνου:

$$\vec{normal}_{tri} = normalize((1 - u - v) * \vec{n_1} + u * \vec{n_2} + v * \vec{n_3})$$

  Τα u,v είναι βαθμωτές παράμετροι.Τα διανύσματα $\vec{n_1}, \vec{n_2}, \vec{n_3}$ απότελούν τα κανονικά διανύσματα κάθε κορυφής του τριγώνου.

Προσοχή χρειάζεται ο υπολογισμός του κανονικού διανύσματος και του σημείου διασταύρωσης ακτίνας-επιφάνειας όταν έχει μετασχηματιστεί η ακτίνα.Για να είναι σωστοί οι υπολογισμοί φωτισμού πρέπει όλα τα διανύσματα να βρίσκονται στον ίδιο χώρο.Εφόσον έχει εφαρμοστεί ο αντίστροφος μετασχηματισμός στην ακτίνα, τότε το σημείο διασταυρώσεως και το κανονικό διάνυσμα βρίσκονται στον χώρο που ορίζεται από τον αντίστροφο μετασχηματισμό,δηλαδή στις τοπικές συντεταγμένες της επιφάνειας.Για να τα επαναφέρουμε στο χώρο συντεταγμένων που ορίζονται από τον μετασχηματισμό της επιφάνειας πρέπει απλά να μετασχηματίσουμε σύμφωνα ξανά σύμφωνα με τον πίνακα μετασχηματιμού:

$$\vec{p_{intersection}} = M * \vec{p_{local}}$$

Το κανονικό διάνυσμα χρειάζεται μεγαλύτερη προσοχή.Στην ουσία αποτελεί διάνυσμα διεύθυνσης και όχι διάνυσμα σημείου.Τα διανύσματα διεύθυνσης θα μετασχηματιστούν ως εξής:

$$\vec{normal} = normalize((M^{-1})^{\tau} * \vec{normal}_{surface}$$

Phong Shading

Μια δημοφιλής και αποδοτική τεχνική φωτισμού είναι η μέθοδος phong [Pho75].Η μέθοδος καθορίζει το χρώμα μιας επιφάνειας ανάλογα τις ιδιότητες της και το χρώμα και θέση της πηγής φωτός.Η μέθοδος χωρίζει τις ιδιότητες του φωτός μιας επιφάνειας σε 3 διαφορετικές συνιστώστες:

- Ambient:Το χρώμα που δίνεται στην επιφάνεια από ανακλάσεις ακτίνων που δεν φτάνουν απευθείας από την πηγή φωτός αλλά από άλλα αντικείμενα στη σκηνή.Μπορεί να είναι μια απλή σταθερά χρώματος RGB με μικρή ένταση ή να βρίσκεται από πολύπλοκους υπολογισμούς που κάνουν το φωτισμό πιο ρεαλιστικό,όπως είναι για παράδειγμα τεχνικές τύπου ambient occlusion(κάνε ρεφ).

- Diffuse:Το χρώμα που δίνεται από ακτίνες που φτάνουν απευθείας στην επιφάνεια.Αυτή η συνιστώσα είναι η κυριότερη γιατί απεικονίζει το βασικό χρώμα της επιφάνειας.

- Specular: Το χρώμα που δίνεται στις επιφάνειες που είναι γυαλιστερές.

Το τελικό χρώμα φωτισμού της μεθόδου είναι το άθροισμα των τριών συνιστωσών χρωμάτων.

Ambient Color

Η ambient συνιστώσα που χρησιμοποιήσα είναι απλή υπολογιστικά και περιγράφεται από μια βαθμωτή μεταβλητή μικρής θετικής τιμής(0.15).Το diffuse χρώμα του υλικού της επιφάνειας σε RGB πολλάσιάζεται με την ambient μεταβλητή για να πάρουμε την ambient συνιστώσα.

$$\overrightarrow{ambientColor} = ambientFactor * \overrightarrow{Diffuse_{material}}$$

Diffuse Color

Το diffuse χρώμα έχει το μεγαλύτερο ενδιαφέρον από τις τρεις συνιστώσες,γιατί καθορίζει σημαντικά το χρώμα της επιφάνειας που θα αποθηκευτεί στη συνέχεια στην τελική εικόνα.Για να υπολογίσουμε τον diffuse παράγοντα χρειάζομαστε όπως φαίνεται στην παρακάτω εικόνα(πες ποια εικόνα):

- Το κανονικό διάνυσμα της επιφάνειας για το σημείο διασταύρωσης ακτίνας-επιφάνειας(normal vector).
- Το σημείο διασταύρωσης ακτίνας-επιφάνειας.
- Το σημείο θέσης της πηγής φωτός.
- Το χρώμα των ακτίνων φωτός της πηγής.
- Το diffuse χρώμα της επιφάνειας.Συνήθως περιγράφεται από τις ιδιότητες του υλικού της επιφάνειας(material).

Το normal vector και το σημείο διασταύρωσης βρίσκεται από τις μεθόδους διασταύρωσης ακτίνας-επιφάνειας,όπως περιγράφεται παραπάνω.Η θέση και το χρώμα της πηγής φωτός συνήθως δίνοται στην αρχικοποίηση της σκηνής.Το τελικό diffuse χρώμα δίνεται από την παρακάτω εξίσωση:

$$\overrightarrow{diffuse_{color}} = diffuse_{factor} * \overrightarrow{diffuse_{material}} * \overrightarrow{diffuse_{light}} \tag{3.9}$$

Το γινόμενο $\overrightarrow{diffuse_{material}} * \overrightarrow{diffuse_{light}}$ καθορίζει το τελικό diffuse χρώμα της επιφάνειας. Προσέξτε οτι το γινόμενο είναι πολλαπλασιασμός ανα συνιστώσα δύο διανυσμάτων και όχι εσωτερικό γινόμενο.Με αυτό τον τρόπο υπολογίζεται μόνο το χρώμα που αντανακλάται από την επιφάνεια περιορίζοντας το από το χρώμα του φωτός.Για παράδειγμα,αν ρίξουμε μπλε φως σε μια ντομάτα,στα μάτια μας θα φανεί μαύρη γιατί η ντομάτα αντανακλά μόνο το κόκκινο χρώμα.Αντίστοιχα,αν μια επιφάνεια έχει χρώμα σε RGB $surface_{color} = (0.0, 1.0, 0.0)$ ενώ η πηγή φωτός $light_{color} = (0.0, 0, 0, 1.0)$,τότε ο πολλασιασμός των δύο διανυσμάτων θα είναι το μαύρο χρώμα $black = (0.0, 0.0, 0.0)$.Για αυτό το λόγο συνήθως το χρώμα της πηγής φωτός είναι λευκό $white = (1.0, 1.0, 1.0)$ έτσι ώστε να αντακλούνται όλα τα χρώματα μιας επιφάνειας,εκτός κι αν ο σκοπός είναι να πετύχουμε κάποιο διαφορετικό εφέ.

Το μεγαλύτερο ενδιαφέρον στην εξίσωση 3.9 είναι η βαθμωτή μεταβλητή $diffuse_{factor}$.Η μεταβλητή καθορίζει το ποσοστό του χρώματος που δίνεται

στην επιφάνεια,κυμαίνεται στο διάστημα $(0, 1)$ και εξαρτάται από τη γωνια θ μεταξύ του normal vector $\vec{N}$ και του διανύσματος $\vec{L}$.Ο υπολογισμός του $\vec{L}$ είναι απλός και βρίσκεται:

$$\vec{L} = normalize(\overrightarrow{Light_{position}} - \overrightarrow{P_{intersection}})$$

Από το εσωτερικό γινόμενο των δύο διανυσμάτων μπορούμε να βρούμε το συνημίτονο της φωνίας θ,αφού:

$$dot = |\vec{A}| * |\vec{B}| * \cos\theta$$

Το σύνολο τιμών του συνημιτόνου είναι το διάστημα $[-1, 1]$.Αρνητική τιμή σημαίνει οτι τα διανύσματα έχουν γωνία μεγαλύτερη από 90 μοίρες,που στην περίπτωση του φωτισμού συνεπάγεται ότι η θέση της πηγής βρίσκεται ¨πίσω' από την φορά που δείχνει το κανονικό κάθετο διάνυσμα.Συνεπώς η επιφάνεια έχει ¨πλάτη' την πηγή φωτός και άρα πρέπει να μην φωτιστεί ή ισοδύναμα να πάρει το μαύρο χρώμα.Γιάυτό το λόγο περιορίζουμε τις τιμές του συνημιτόνου στο διάστημα $(0, 1)$.Άρα ο diffuse παράγοντας υπολογίζεται ως:

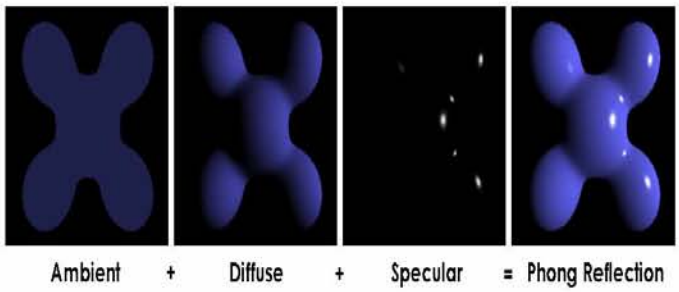
$$diffuse_{factor} = max(0.0, \vec{N} \cdot \vec{L}) \tag{3.10}$$

Επίσης το συνημίτονο έχει μια ακόμη ιδιότητα που είναι χρήσιμη στην περίπτωση του φωτισμού:Αν η γωνία των διανυσμάτων $\vec{N}$ και $\vec{L}$ είναι μηδέν,δηλαδή είναι παράλληλα,τότε το συνημίτονο δίνει τιμή 1.Άρα το συγκεκριμένο σημείο θα φωτιστεί με πλήρες χρώμα σύμφωνα με την εξίσωση 3.9.Αν τα διανύσματα είναι κάθετα,τότε οι ακτίνες της πηγής οριακά δεν πετυχαίνουν την επιφάνεια και το συνημίτονο δίνει τιμή μηδέν,δηλαδή το χρώμα που θα δωθεί θα είναι μαύρο.Για τις ενδιάμεσες γωνίες το χρώμα της επιφάνειας καθορίζεται ομάλα σύμφωνα με το ποσοστό που ορίζει το συνημίτονο.

Specular Color

Το specular χρώμα υπολογίζεται με παρόμοιο τρόπο.Σε αυτήν την περίπτωση χρειαζόμαστε και τη θέση της κάμερας στον τρισδιάστο χώρο και έναν ακόμη παράγοντα specularity που ορίζει κατα πόσο μεγάλη είναι η γυαλάδα της επιφάνειας στην εικόνα.Το specular χρώμα υπολογίζεται ως:

$$\overrightarrow{Specular_{color}} = Spec_{factor} * \overrightarrow{Specular_{surface}} * \overrightarrow{Specular_{light}} \tag{3.11}$$



Σχήμα 3.1: Φωτισμός κατά Phong [Wik]

Ξανά ενδιαφέρον στην παραπάνω εξίσωση έχει ο παράγοντας $Spec_{factor}$ που καθορίζει κατα πόσο γυαλίζει μια επιφάνεια.Το διάνυσμα $\vec{L}$ είναι το ίδιο που χρησιμοποιείται για τον diffuse παράγοντα.Το διάνυσμα $\vec{C}$ δείχνει τη κατεύθυνση από το σημείο διασταυρώσεως προς την κάμερα:

$$\vec{C} = normalize(\overrightarrow{Camera_{pos}} - \overrightarrow{P_{intesection}})$$

Το διάνυσμα $\vec{R}$ δείχνει την κατεύθυνση ανάκλασης που θα ακουληθήσει η ακτίνα αφότου φτάσει στην επιφάνεια.Προσοχή,το specular χρώμα δεν υπολογίζει το χρώμα των ακτίνων λόγω αντανάκλασης του υλικού.Η αντανάκλαση επιφάνειας περιγράφεται σε επόμενη παράγραφο.Αντίστοιχα η γωνία θ μεταξύ των διανυσμάτων $\vec{C}$ και $\vec{R}$ βρίσκεται χρησιμοποιώντας εσωτερικό γινόμενο.

$$specCos = max(0.0, \vec{C} \cdot \vec{R}) \tag{3.12}$$

Ο τελικός παράγοντας για το specular χρώμα σύμφωνα με την τεχνική phong υπολογίζεται ως:

$$Spec_{factor} = specCos^{specularity} \tag{3.13}$$

Το τελικό χρώμα που δίνεται στην επιφάνεια για το συγκεκριμένο σημείο διασταυρώσεως είναι το άθροισμα των παραγώντων,δηλαδή:

$$\overrightarrow{color} = \overrightarrow{ambientColor} + \overrightarrow{diffuseColor} + \overrightarrow{specularColor}$$

Ο φωτισμός υλοποιείται στη μέθοδο shade του αλγορίθμου 2 του κεφαλαίου 2.

3.4.2 Σκιές - Shadows

Ένα εφέ γραφικών που είναι δύσκολο να απεικονιστεί από αλγορίθμους τύπου rasterazation είναι οι σκιές.Ο κυριότερος λόγος που το εφέ είναι δύσκολο να απεικονιστεί είναι γιατί το rasterazation επεξεργάζεται κάθε επιφάνεια ξεχωριστά και όταν έρχεται η στιγμή του φωτισμού για τη συγκεκριμένη επιφάνεια δεν υπάρχει πληροφορία της ¨γειτονικής' γεωμετρίας,δηλαδή των υπόλοιπων επιφανειών της σκηνής.Οι σκιές που υπολοποιούνται σε τέτοιους αλγορίθμους βασίζονται κυρίως σε shadow maps(κάνε ρεφ) ή παραπάνω από ένα rendering pass.

Στο ray tracing το εφέ των σκιών είναι πολύ εύκολο να υλοποιηθεί γιατί ανα πάσα στιγμή κάθε ακτίνα έχει πρόσβαση σε όλη τη γεωμετρία της σκηνής.Προτού κληθεί ο υπολογισμός του φωτισμού για το σημείο διασταυρώσεως για τις πηγές φωτός της σκηνής,ξεκινάμε μια καινούρια ακτίνα που ονομάζεται ακτίνα σκιάς(shadow ray με αρχικό σημείο το σημείο διασταυρώσεως που έχει ήδη βρεθεί από τις αρχικές ακτίνες της κάμερας και με κατεύθυνση το διάνυσμα που δείχνει προς τη πηγή φωτός. Έπειτα ελέγχεται αν η ακτίνα διασταυρώνεται με κάποια επιφάνεια της σκηνής.Αν βρεθεί επιφάνεια διασταυρώσεως,τότε σημαίνει οτι αυτή η επιφάνεια ¨κρύβει' τις ακτίνες της συγκεκριμένης πηγής για το σημείο διασταυρώσεως των ακτίνων της κάμερας.Σε διαφορετική περίπτωση,δηλαδή αν δεν βρεθεί σημείο διασταυρώσεως,τότε οι ακτίνες της πηγής φωτός ¨βλέπουν' το σημείο διασταυρώσεως της πρώτης ακτίνας και ο υπολογισμός προχωράει με στη φάση του φωτισμού.Ο αλγόριθμος 2 τροποποιείται για να υποστηρίζει και το εφέ σκιών:

Με μερικές απλές αλλαγές ο αλγόριθμος 4 δείχνει πως υλοποιούνται οι σκιές στον αλγόριθμο του ray tracing αλλά και πόσο εύκολο είναι να υπολογίσουμε το φωτισμό για πολλές πηγές φωτός σε μια εικονική σκηνή.Προσέξτε οτι η μέθοδος *ιντερσεστΐσιβιλιτψ* απλά χρειάζεται να επιστρέφει *τρυε* ή *φαλσε*,δηλαδή αν η ακτίνα σκιας διασταυρώνεται με οποιαδήποτε επιφάνεια της σκηνής.Σε αντίθεση με τις διασταυρώσεις των πρώτων ακτίνων κάμερας,η αναζήτηση δεν συνεχίζεται έτσι ώστε να βρεθεί η κοντινότερη επιφάνεια στο αρχικό σημείο της shadow ray.Οποιαδήποτε επιφάνεια διασταύρωσης αρκεί για να αποφασίσουμε αν πρέπει να εφαρμόσουμε φωτισμό στο αρχικό σημείο διασταυρώσεως για τη συγκεκριμένη πηγή φωτός.

Κλείνοντας είναι απαραίτητο να αναφερθεί ότι το αρχικό σημείο της shadow ray πρέπει να μετακινηθεί κατά ένα μικρό παράγοντα προς τη κατεύθυνση του κανονικού διανύσματος(normal vector) της επιφάνειας για να αποφύγουμε την

Algorithm 4 Ray Trace

```
1: procedure RayTrace
2:     t ← MAX
3:     colorRGB ← Black
4:     for all surfaces in Scene do
5:         s ← INTERSECT ray WITH surface
6:         t ← min(t, s)
7:     end for
8:     intersectionPoint ← calculateIntersection(ray,t)
9:     for all lights do
10:        shadowRay ← calcRay(intersectionPoint,directionToLight)
11:        intersectionFound ← intersectVisibility(shadowRay)
12:        if ¬intersectionFound then
13:            colorRGB ← colorRGB + shade(intersectionPoint,Light)
14:        end if
15:    end for
16:    return colorRGB
17: end procedure
```

εύρεση διασταύρωσης με την ίδια την επιφάνεια που ξεκινάει η ακτίνα,δηλαδή:

$$\overrightarrow{shadow_{origin}} = \overrightarrow{Point_{intersection}} + \epsilon * \overrightarrow{Normal_{intersection}}$$

3.4.3 Αντανάκλαση - Reflection

Η αντανάκλαση είναι ένα άλλο εφέ γραφικών που είναι δύσκολο να υπολογιστεί σωστά σύμφωνα με την γειτονική γεωμετρία της σκηνής σε αλγορίθμους τύπου rasterazation.Στο ray tracing αυτή η πληροφορία είναι διαθέσιμη για κάθε ακτίνα ξεχωριστά και έτσι είναι σχετικά απλό να βρούμε σωστές ανταναακλάσεις επιφανειών.

Αρχικά πρέπει η επιφάνεια διασταυρώσεως της ακτίνας της κάμερας να επιτρέπει την ανάκλαση ακτίνων,όπως είναι για παράδειγμα ένας καθρέπτης ή η επίφανεια μιας λίμνης.Η διεύθυνση της ανάκλασης περιγράφεται από τον παρακάτω τύπο,όπως περιγράφεται με λεπτομέρια στη δημοσίευση του bran de grieve [Gre06]:

$$\vec{R} = \vec{I} - 2(\vec{I} \cdot \vec{N})\vec{N} \tag{3.14}$$

Το διάνυσμα $\vec{I}$ είναι η κατεύθυνση προσπίπτουσας ακτίνας και $\vec{N}$ είναι το normal vector της επιφάνειας.Η κατεύθυνση ανάκλασης περιγράφεται από την εξίσωση 3.14.

ο αλγόριθμος του ray tracing εύκολα μετατρέπεται έτσι ώστε να υποστηρίζει και το εφέ της αντανάκλασης.Οι αλλαγές που χρειάζονται φαίνονται παρακάτω:

Algorithm 5 Ray Trace

```
1:  procedure RayTrace
2:      t ← MAX
3:      colorRGB ← Black
4:      for all surfaces in Scene do
5:          s ← INTERSECT ray WITH surface
6:          t ← min(t, s)
7:      end for
8:      intersectionPoint ← calculateIntersection(ray,t)
9:      for all lights do
10:         shadowRay ← calcRay(intersectionPoint,directionToLight)
11:         intersectionFound ← intersectVisibility(shadowRay)
12:         if ¬intersectionFound then
13:             colorRGB ← colorRGB + shade(intersectionPoint,Light)
14:         end if
15:     end for
16:     if Surface is Reflective then
17:         reflectedRay = reflect(ray)
18:         colorRGB ← colorRGB + fresnel * raytrace(reflectedRay)
19:     end if
20:     return colorRGB
21: end procedure
```

Όπως φαίνεται στον αλγόριθμο 5,η μέθοδος raytrace γίνεται αναδρομική.Στο σημείο διασταυρώσεως δημιουργείται μια νέα ακτίνα με αρχικό σημείο το σημείο διασταυρώσεως της επιφάνειας και κατεύθυνση που ορίζει η εξίσωση 3.14.Έπειτα ακολουθείται η καινούρια ακτίνα καλώντας την raytrace, η οποία με τη σειρά της θα επιστρέψει το χρώμα της επιφάνειας που διασταυρώνεται.Με αυτό τον τρόπο στο χρώμα του αρχικό σημείου δισταυρώσεως θα προστεθεί το χρώμα της επιφάνειας ανάκλασης.

Ένα ερώτημα που προκύπτει από τον αλγόριθμο 5 είναι πότε σταματάει η αναδρομή·Υποθετικά μια ακτίνα θα μπορούσε να προσπίπτει συνέχεια πάνω

σε επιφάνειες την που αντανακλούν.Σε αυτή την περίπτωση ο υπολογισμός θεωρητικά δεν θα τελείωνε ποτέ και πρακτικά θα έσκαγε το πρόγραμμα.Για την αντιμετώπιση του προβλήματος υπάρχουν δύο τρόποι: α) Να οριοθετίσουμε τον αριθμό πιθανών αναδρομικών κλήσεων ή β) να δημιουργήσουμε ένα μοντέλο φυσικής απεικόνισης των ακτίνων δίνοντας τες ένα ποσό ενέργειας κατά την κατασκευή τους και μειώνοντας το κάθε φορά που διασπάται σε περισσότερες μέχρις ότου η ενέργεια μιας ακτίνας είναι σημαντικά μικρή για να κάνει διαφορά στο τελικό χρώμα.Στην υλοποιήση που έφτιαξα διάλεξα τον πρώτο τρόπο.

Τέλος ένας σημαντικός παράγοντας που συνεισφέρει σημαντικά στον φωτορεαλισμό των αντανακλάσεων είναι η βαθμωτή μεταβλητή *fresnel*. Διαφορετικά υλικά αντανακλούν διαφορετικό ποσοστό χρώματος σε μια πραγματική επιφάνεια,όπως για παράδειγμα ένας καθρέπτης με ένα διαμάντι. Ο παράγοντας fresnel ορίζει το ποσοστό του χρώματος που θα αποδοθεί στην επιφάνεια από την επιφάνεια αντανάκλασης.

Κεφάλαιο 4

Acceleration Structures

Σε αυτό το κεφάλαιο παρουσιάζονται διάφορες δομές επιτάχυνσης του αλγορίθμου ray tracing για τη βελτίωση της επίδοσης.

4.1 Δημοφιλείς Δομές

Παρατηρώντας τον αλγόριθμο 2 του κεφαλαίου 2 φαίνεται οτι το υπολογιστικά βαρύ κομμάτι είναι η αναζήτηση διασταυρώσεων της ακτίνας με μια επιφάνεια της σκηνής.Συχνά οι επιφάνειες μιας ψηφιακής σκηνής είναι της τάξης των εκατοντάδων χιλιάδων,ακόμη και εκατομμυρίων.Είναι φανερό οτι η μέθοδος της απλής σειριακής αναζήτησης διασταυρώσεων δεν είναι αποδοτική.Με απλή σειριακή αναζήτηση κάθε ακτίνα διασταυρώνεται ξεχωριστά με κάθε επιφάνεια για την εύρεση της κοντινότερης επιφάνειας,αφού κάθε ακτίνα έχει ουσιαστικά διαφορετική κατεύθυνση.

Μια γνωστή δομή για τη μείωση του αριθμού διασταυρώσεων είναι το ομοιόμορφο τρισδιάστατο πλέγμα(uniform grid).Ο τρισδιάστατος ψηφιακός χώρος που καλύπτει η σκηνή υποδιαιρείται ομοιόμορφα σε AABB (axis aligned bounding box).Αντί να ελέγχουμε κάθε επιφάνεια ξεχωριστά,ελέγχουμε τα AABB του πλέγματος προς την κατεύθυνση της ακτίνας και αν στο AABB περικλείωνται επιφάνειες της σκηνής, τότε ελέγχονται για διασταύρωση.Με αυτό τον τρόπο διασταυρώνονται μόνο οι επιφάνειες που βρίσκονται στην κατεύθυνση της ακτίνας χωρίς να χρειάζεται να εξεταστούν οι υπόλοιπες.

Το ομοιόμορφο πλέγμα έχει όμως ένα μειωνέκτημα που μπορεί να επηρεάσει αρνητικά τελικά την επίδοση του ray tracing.Όταν το μέγεθος των επιφανειών είναι δυσανάλογο,δηλαδή άλλες είναι μεγάλες ενώ άλλες είναι σηματικά μι-

κρές,τότε εύκολα παραλείπονται οι μικρές επιφάνειες.Ένα κλασσικό παράδειγμα σκηνής αποτελεί η κούπα σε στάδιο(teapot in stadium).Για να διασταυρώσουμε μια ακτίνα με την κούπα,θα χρειαστεί να σταυρώσουμε πολλές άλλες επιφάνειες του γηπέδου για να ζωγραφιστεί τελικά η κούπα.Μπορούμε να μειώσουμε το μέγεθος κάθε AABB του πλέγματος έτσι ώστε να μειώσουμε τον αριθμό διασταυρώσεων με επιφάνειες του γηπέδου,αλλά έτσι αυξάνουμε σημαντικά τον αριθμό διασταυρώσεων ακτίνων με τα AABB του grid που ουσιαστικά δεν αποτελούν επιφάνειες της σκηνής.Αν από την άλλη αυξήσουμε το μέγεθος των AABB,τότε ναι μεν μειώνουμε τον αριθμό διασταυρώσεων με τα AABB,αλλά ταυτόχρονια αυξάνεται ο αριθμός διασταυρώσεων επιφανειών που τελικά δεν χρωματιστούν από τη φάση του φωτισμού.Έτσι είναι φανερό οτι η επιλογή του μεγέθους του AABB του ομοιόμορφου πλέγματος πρέπει να είναι ανάλογη με το μέγεθος των επιφανειών και φυσικά οι επιφάνειες μεταξύ τους να έχουν όμοιο μέγεθος.

Μια άλλη κατηγορία δομών επιτάχυνσης είναι οι δομές ιεραρχίας.Στην ουσία αποτελούν δέντρα που είτε χωρίζουν χωρικά τη σκηνή(spatial subdivision),όπως είναι τα Kd Trees,είτε υποδιαιρούν τις επιφάνειες μεταξύ τους όπως είναι τα BVH Trees(Bounding Volume Hierarchy).Στην υλοποίηση που έκανα στα πλαίσια του ray tracing χρησιμοποιήσα BVH δέντρα.

4.2 Bounding Volume Hierarchy(BVH)

Η BVH είναι μια δομή δέντρου πάνω σε ένα σύνολο επιφανειών.Κάθε επιφάνεια της σκηνής περικλείεται μέσα σε ένα σχήμα όγκου οριοθέτησης(bounding volume) που αποτελούν τα φύλλα του δέντρου.Κάθε δομή οριοθέτησης αναδρομικά περικλείει όλες τις διαθέσιμες επιφάνειες που με τη σειρά τους περικλείονται από δομές οριοθέτησης μέχρις ότου υπάρχει μια και μοναδική δομή οριοθέτησης που αποτελεί τη ρίζα του δέντρου.Στόχος είναι να μειώσουμε συνολικά τον αριθμό διασταυρώσεων ακτίνας-επιφάνειας του αλγορίθμου και να τις αντικαταστήσουμε με γρήγορες μεθόδους διασταυρώσεως ακτίνας-δομής οριοθέτησης.Η βασική ιδέα είναι ότι αν μια ακτίνα δεν διασταυρώνεται με τη δομή οριοθέτησης,τότε σίγουρα δεν θα διαταυρώνεται και με τις επιφάνειες που περικλείωνται.Με αυτό τον τρόπο μπορούμε εύκολα να αποφανθούμε αν χρειάζεται να διασταυρώσουμε μια ακτίνα με ένα σύνολο από επιφάνειες(trivial surface rejection).

Η επιλογή της δομής οριοθέτησης είναι κρίσιμη.Η δομή οριοθέτησης πρέπει να περικλείει όλες τις επιφάνειες που δίνονται με τον μικρότερο δυνατό όγκο και ταυτόχρονα η μέθοδος διασταύρωσης με μια ακτίνα να είναι γρήγορη.Συνήθως

η ταχύτητα διασταυρώσεως είναι αντιστρόφως ανάλογη με τον όγκο της δομής οριοθέτησης.Οι δομές που περικλείουν ένα σύνολο επιφανειών με μικρό όγκο συχνά χρειάζονται αρκετούς υπολογισμούς κατά τη φάση της διασταύρωσης,ενώ δομές που είναι πιο ανεκτές στον όγκο,έχουν ταχύτερη μέθοδο διασταυρώσεως (συνήθως χρειάζονται και λιγότερη μνήμη).Το AABB (axis aligned bounding box) προσφέρει ικανοποιητική σχέση ταχύτητας-όγκου.Στην υλοποίηση μου χρησιμοποίησα AABB.

4.2.1 Κατασκευή Δέντρου-Tree contruction

Η κατασκευή της ιεραρχίας αποτελεί κρίσιμο κομμάτι της τελικής επίδοσης της δομής.Μια αφελής κατασκευή δέντρου BVH θα είναι σημαντικά πιο γρήγορη από την απλή σειριακή αναζήτηση αλλά υστερεί έναντι μιας προσεγμένης.Η ποιότητα της κατασκευής καθορίζει ουσιαστικά την επίδοση της δομής κατά τη διάρκεια της απεικόνισης.Ο τρόπος διαχωρισμού των επιφανειών,όπως εξηγείται παρακάτω, αποτελεί το κλειδί της ποιότητας της δομής.

Συνήθως τα BVH δέντρα είναι δυαδικά,αλλά δεν είναι ο κανόνας.Δέντρα με μεγαλύτερο αριθμό παιδιών έχουν μικρότερο ύψος που σημαίνει οτι η αναζήτηση γρήγορα φτάνει στα φύλλα,δηλαδή στις επιφάνειες της ψηφιακής σκηνής(τρίγωνα,σφαίρες κτλ),από την άλλη αυτό απαιτεί μεγαλύτερο χρόνο επεξεργασίας στους ενδιάμεσους κόμβους.Στην υλοποιήση μου χρησιμοποίησα δυαδικά δέντρα.

Οι κύριοι μέθοδοι για την κατασκευή ενός δέντρου BVH χωρίζονται σε δύο κατηγορίες.Από τη μια μεριά υπάρχουν οι offline αλγόριθμοι που απαιτούν όλες οι επιφάνειες της σκηνής να είναι γνωστές εκ των προτέρων και από την άλλη οι online που δεν χρειάζονται πληροφορία για όλη τη γεωμετρία,η οποία μπορεί να αλλάζει ακόμη και δυναμικά.Οι offline είναι οι πιο απλοί στην υλοποίηση και χωρίζονται σε δύο υποκατηγορίες:

- Top-down:Η κατασκευή γίνεται από πάνω προς τα κάτω.Αρχικά η ρίζα του δέντρου περιέχει κάθε επιφάνεια της γεωμετρίας.Έπειτα αναδρομικά οι επιφάνειες χωρίζονται σε δύο σύνολα,ένα για κάθε παιδί του δέντρου μέχρις ότου ένας κόμβος γίνει φύλλο,δηλαδή περιέχει το πολύ N επιφάνειες.

- Bottom-up:Κατασκευή από κάτω προς τα πάνω.Αρχικά όλες οι επιφάνειες θεωρούνται ως τα φύλλα του δέντρου και ομαδοποιούνται ανά δύο ή περισσότερες σε μια δομή οριοθέτησης που κατασκευάζεται και αποτελεί

ενδιάμεσο κόμβο του δέντρου.Η κατασκευή σταματά όταν δεν υπάρχει άλλος κόμβος προς ομαδοποίηση,δηλαδή φτάσουμε σε ένα μοναδικό κόμβο,τη ρίζα του δέντρου.

Η πιο απλή και γρήγορη μέθοδος είναι η top-down κατασκευή.Όπως αναφέρθηκε,πρέπει όλες οι επιφάνειες να είναι γνωστές προτού αρχίσει η κατασκευή του δέντρου.Ο αλγόριθμος (κανε ρεφ) δείχνει σε μορφή ψευδοκώδικα τον top-down τρόπο κατασκευής της BVH. Ο αλγόριθμος σε κάθε αναδρο-

Algorithm 6 BVH construction

Require: surfaces,start,end

```
1:  procedure buildTopDown
2:      node = buidNode()
3:      node.bounding = computeBoundingVolume(surfaces,start,end)
4:      addToTree(node)
5:      numNodes = end − start
6:      if numNodes < MAX then
7:          createLeaf(node, surfaces,start,end)
8:      else
9:          splitIndex = partition(surfaces,start,end)
10:         buildTopDown(node.leftChild, start,splitIndex)
11:         buildTopDown(node.rightChild,splitIndex,end
12:         makeInterNode(node)
13:     end if
14: end procedure
```

μική κλήση δημιουργεί έναν καινούριο κόμβο.Κάθε κόμβος περιέχει μια δομή οριοθέτησης που περιλαμβάνει όλες τις επιφάνειες που ορίζονται από τα indices start και end.Έπειτα ελέγχεται αν το πλήθος των επιφανειών για τον καινούριο κόμβο του δέντρου είναι μικρότερο από μια σταθερά MAX και ο κόμβος μετατρέπεται σε φύλλο.Η σταθερά καθορίζει το μέγιστο αριθμό επιφανειών κάθε φύλλου.Μικρές τιμές σταθεράς κάνουν το δέντρο πιο ψηλό,ενώ μεγάλες χρειάζονται μεγαλύτερο χρόνο επεξεργασίας φύλλου,αφού πρέπει να εξεταστούν όλες οι επιφάνειες προς διασταύρωση.Σε διαφορετική περίπτωση,δηλαδή όταν το πλήθος επιφανειών είναι μεγάλύτερο από τη σταθερά MAX,το σύνολο των επιφανειών χωρίζεται σε δύο σύνολα και αναδρομικά δημιουργούνται τα δύο παιδιά του ενδιάμεσου κόμβου.

Το κομμάτι του αλγορίθμου που καθορίζει την ποιότητα του δέντρου είναι η γραμμή εννέα του αλγορίθμου 6.Ένα bvh δέντρο θεωρείται καλό όταν προσφέρει υψηλή επίδοση κατά τη διάρκεια απεικόνισης του ray tracing.Ένα απρόσεκτο partitioning του συνόλου επιφανειών για τον τρέχων κόμβο είναι δυνατό να επηρεάσει σημαντικά την επίδοση κατά τη διαπέραση στη φάση της απεικόνισης.Για παράδειγμα, αν αποφασίσουμε το δέντρο να είναι ισοσταθμισμένο,τότε κάθε φορά χωρίζουμε τις επιφάνειες σε δύο σύνολα ίσου πλήθους.Αυτή η προσέγγιση είναι καλή μόνο στις περιπτώσεις που οι επιφάνειες της σκηνής βρίσκονται σχετικά ¨κοντά' μεταξύ τους και έχουν όμοιο μέγεθος.Αν οι επιφάνειες είναι διαφορετικού μεγέθους και δεν καλύπτουν ομοιόμορφα το χώρο,τότε κατά τη διαπέραση,όπως περιγράφεται στην επόμενη παράγραφο του κειμένου,η ακτίνα θα επισκεπτεί πολλούς κόμβους του δέντρου άδικα,χωρίς να βρεθεί δηλαδή μια επιφάνεια διασταυρώσεως.Έτσι είναι φανερό ότι η partitioning μέθοδος πρέπει να λαμβάνει υπόψιν και το χώρο που τελικά καταλαμβάνουν οι επιφάνειες στη τρισδιάστατη σκηνή.

Μια γνωστή και γρήγορη μέθοδος για τον διαχωρισμό των επιφανειών είναι ο διαχωρισμός χωρικής διαμέσου(spatial median split [WK06]).Σύμφωνα με τη μέθοδο spatial median split αρχικά υπολογίζεται ο άξονας της δομής οριοθέτησης του κόμβου(X,Υ,Z) με το μεγαλύτερο μήκος.Έπειτα οι επιφάνειες ταξινομούνται σύμφωνα με τον άξονα που επιλεχτηκε.Το index διαχωρισμού των επιφανειών υπολογίζεται με βάση το μέσο σημείο της δομής οριοθέτησης στον επελεγμένο άξονα.Με αυτό τον τρόπο οι επιφάνειες χωρίζονται χωρικά σε δύο διαφορετικά σύνολα.Ο τρόπος διαχωρισμού είναι μεν σχετικά γρήγορος και δημιουργεί bvh δέντρα καλής ποιότητας,από την άλλη δεν είναι ισοσταθμισμένα.Η ταχύτητα απεικόνισης όμως είναι σημαντικά πιο γρήγορη σε σχέση με τον απλό και αφελή διαχωρισμό ισοσταθμισμένου δέντρου.

Μια εναλλακτική μέθοδος είναι η SAH(surface area heuristic) [Wal07].Η μέθοδος υπολογίζει μια συνάρτηση κόστους με βάση την επιφάνεια που καλύπτει η δομή οριοθέτησης.Το ελάχιστο υπολογίσιμο κόστος καθορίζει το index διαχωρισμού των επιφανειών,δηλαδή τα σύνολα επιφανειών κάθε παιδιού ενός ενδιάμεσου κόμβου του δέντρου.Η συνάρτηση κόστους είναι η παράκατω:

$$SAH_{cost} = C_{Tr} + N_r * C_I * \frac{A_r}{A} + N_l * C_I * \frac{A_l}{A} \tag{4.1}$$

όπου C_{Tr} και C_I είναι σταθερές της υλοποίησης και περιγράφουν το κόστος διαπέρασης και το κόστος διασταυρώσεως της ακτίνας από τον συγκεκριμένο κόμβο αντίστοιχα.Οι παράμετροι N_r και N_l αναφέρονται στο πλήθος των επιφανειών για το δεξί και αριστερό παιδί του κόμβου αντίστοιχα. Η A είναι η surface

area της δομής οριοθέτησης του τρέχον κόμβου ενώ A_r και A_l η surface area των παιδιών.

Η SAH κατασκευάζει πιο γρήγορα δέντρα σε σχέση με τη μέθοδο spatial median split,αλλά παίρνει αρκετό χρόνο για να κατασκευάσει το δέντρο.Στις περιπτώσεις που οι επιφάνειες της σκηνής είναι της τάξης των χιλιάδων,η κατασκευή μπορεί να διαρκέσει αρκετά λεπτά.Σε αντίθεση,η spatial median split κατασκευάζει πολύ γρήγορα τα δέντρα ακόμα κι αν ο αριθμός των επιφάνειών είναι μεγάλος.Στην υλοποίηση μου χρησιμοποιώ την spatial median split για τους πρώτους κόμβους του δέντρου για να μειώσω τον αριθμό επιφάνειων στα πρώτα υποδέντρα και στη συνέχεια αλλάζω τη διαχώριση σε SAH.Ο παραπάνω τρόπος δεν παράγει τόσο ποιοτικά δέντρα όσο η SAH,αλλά η επίδοση απεικόνισης είναι συγκρίσιμη και η κατασκευή διάρκεί μόνο μερικά δευτερόλεπτα.

4.2.2 Διαπέραση Δέντρου-Tree traversal

Η διαπέραση του δέντρου είναι ένα ακόμη σημαντικό κομμάτι που χρειάζεται ιδιαίτερη προσοχή.Κατά τη διάρκεια της απεικόνισης κάθε ακτίνα,είτε είναι ακτίνα κάμερας,είτε είναι δευτερεύουσα ακτίνα(σκιάς,αντανάκλασης),πρέπει να διαπεράσει ένα τουλάχιστον ένα κομμάτι του δέντρου,προτού λάβει την πληροφορία που χρειάζεται στη συνέχεια ο υπολογισμός.Οι ακτίνες κάμερας και αντανάκλασης αναζητούν την κοντινότερη επιφάνεια διασταυρώσεως,ενώ οι ακτίνες σκιάς αναζητούν μια οποιαδήποτε επιφάνεια.Για τις δεύτερες η διαπέραση είναι μικρής διάρκειας αφού αρκεί να φτάσει σε ένα οποιοδήποτε φύλλο του δέντρου,ενώ οι πρώτες συνήθως σπαταλούν περισσότερο χρόνο.

Ο αλγόριθμος 7 παρουσιάζει σε μορφή ψευδοκώδικα τη διαπέραση μιας ακτίνας από ένα BVH δέντρο.Αρχικά ο υπολογισμός ξεκινάει από την ρίζα του δέντρου και ελέγχεται για διασταύρωση η δομή οριοθέτησης.Αν βρεθεί διασταύρωση,ο υπολογισμός συνεχίζεται αναδρομικά προς τα δύο παιδιά.Όταν μια ακτίνα φτάσει σε φύλλο του δέντρου,τότε διασταυρώνεται με όλες τις επιφάνειες που περικλείονται στη δομή οριοθέτησης που είναι στην ουσία γεωμετρικές επιφάνειες της τρισδιάστατης σκηνής.Αν χρειαζόμαστε την κοντινότερη διασταύρωση,δηλαδή η ακτίνα είναι είτε ακτίνα κάμερας είτε ακτίνα αντανάκλασης,τότε γίνεται ανανέωση της κοντινότερης επιφάνειας και ο υπολογισμός γίνεται στους υπόλοιπους κόμβους του δέντρου.Αν η ακτίνα είναι ακτίνα σκιάς,τότε μπορούμε να σταματήσουμε τις αναδρομικές κλήσεις μόλις φτάσουμε στο πρώτο φύλλο του δέντρου και βρούμε την πρώτη διασταύρωση ακτίνας-επιφάνειας.

Η αναζήτηση διασταυρώσεως γίνεται σημαντικά πιο γρήγορη με τον αλγόριθμο 7.Η ακτίνα ακολουθεί μόνο τις χωρικά υποψήφιες επιφάνειες της σκη-

Algorithm 7 BVH Traversal

Require: BVH root,ray

```
1: procedure TRAVERSE RAY
2:     boundingVolume ← getBV(node)
3:     intersection ← intersectBV(ray, boundingVolume
4:     if intersection == Found then
5:         if node == Leaf then
6:             intersectRayWithSurfaces(ray,node)
7:         else
8:             TraverseLeftChild(node)
9:             TraverseRightChild(node)
10:        end if
11:        updateMinDistanceSurface()
12:    end if
13:    return minDistanceSurface
14: end procedure
```

νής με αντίτιμο το χρόνο υπολογισμού διασταυρώσεως ακτίνας-δομής οριοθέτησης που σε αρκετές περιπτώσεις είναι γρήγορος.Με αυτόν τον τρόπο αποφεύγονται άσκοποι υπολογισμοί και διασταυρώσεις με επιφάνειες που ουσιαστικά δεν θα επηρεάσουν την απεικόνιση στη συνέχεια του ray tracing.

Η επίδοση του αλγορίθμου 7 μπορεί να βελτιωθεί με δύο μικρές αλλαγές.Αντί να ελέγχεται αναδρομικά πρώτα ο αριστερός κόμβος και ύστερα ο δεξιός,η αναδρομική κλήση να γίνεται πρώτο στο κοντινότερο παιδί,αφού έτσι είναι πιο πιθανό να βρεθεί διασταύρωση ακτίνας-επιφάνειας.Επίσης μπορούμε να αποφύγουμε την κατάβαση σε ένα παιδί αν η απόσταση από το αρχικό σημείο της ακτίνας είναι μεγαλύτερη από την ήδη υπολογισμένη ελάχιστη απόσταση επιφάνειας.Με αυτές τις αλλαγές ο χρόνος διαπέρασης της ακτίνας γίνεται μικρότερος,γιατί απορίπτουμε κομμάτια του δέντρου που εγγυημένα δεν θα συνεισφέρουν ουσιαστική πληροφορία.

Κεφάλαιο 5

Ray Tracing στη GPU

Σε αυτό το κεφάλαιο παρουσιάζεται η υλοποίηση του ray tracing σε μια Nvidia GPU.Περιλαμβάνει όλες τις αναγκαίες αλλαγές που χρειάζεται η υλοποίηση στη CPU ώστε να είναι φιλική και αποδοτική στην αρχιτεκτονική της κάρτας γραφικών.Οι πιο κρίσιμες αλλαγές είναι η τοποθέτηση των δεδομένων στη μνήμη της GPU και η αποδοτική διαπέραση της BVH από ένα warp.Τέλος παρουσιάζονται μετρήσεις και σημειώνονται οι διαφορές επιδόσεων με τη CPU.

5.1 Υλοποίηση

Ο αλγόριθμος του ray tracing μπορεί να θεωρηθεί embarrassingly parallel.Οι ακτίνες του αλγορίθμου είναι ανεξάρτητες μεταξύ τους και το χρώμα που επιστρέφεται για κάθε ακτίνα βασίζεται μόνο στην επιφάνεια διασταυρώσεως της σκηνής.Κατά τη διάρκεια της απεικόνισης μπορούν να δημιουργηθούν εκατομμύρια ακτίνες,οπότε μια καλή ιδέα είναι ο παραλληλισμός να γίνει στο επίπεδο της ακτίνας.Κάθε νήμα της GPU αναλαμβάνει τον υπολογισμό που αντιστοιχεί σε μια ακτίνα και αποθηκεύει το κατάλληλο χρώμα στην εικόνα.

Ένα νήμα της κάρτας γραφικών που αναπαριστά μια ακτίνα μπορεί να ζητήσει τις ιδιότητες οποιασδήποτε επιφάνειας της σκηνής.Οι πιο συνηθισμένες ιδιότητες είναι η θέση της στο χώρο,το υλικό της και το χρώμα της επιφάνειας.Συνεπώς είναι αναγκαίο ολόκληρη η ψηφιακή σκηνή να χωράει στη κύρια μνήμη της κάρτας γραφικών.Σε διαφορετική περίπτωση πρέπει η γεωμετρία της σκηνής να χωριστεί σε δύο ή περισσότερα κομμάτια και η απεικόνιση να γίνει σε περισσότερα από ένα passes.Στην υλοποίηση μου υποθέτω οτι η ψηφιακή σκηνή χωράει στη μνήμη της GPU.Επίσης δεν έχει υλοποιηθεί το εφέ

της αντανάκλασης,όπως περιγράφεται στο κεφάλαιο 3,λόγω περιορισμού αναδρομικών συναρτήσεων στην αρχιτεκτονική των GPU.

Coalesced Memory Access

Ένα σημαντικό και κρίσιμο κομμάτι της υλοποίησης σε μια GPU είναι η ορθή τοποθέτηση των δεδομένων στη μνήμη.Οι μνήμες cache των καρτών γραφικών είναι αρκετά μικρότερες σε μέγεθος από τις αντίστοιχες των σύγχρονων επεξεργαστών.Η απρόσεκτη και αφελής πρόσβαση μνήμης σε έναν kernel της κάρτας γραφικών είναι δυνατόν να μειώσει σημαντικά την επίδοση.Μια κάρτα γραφικών μπορεί να τρέχει χιλιάδες νήματα εκτέλεσης ανα πάσα χρονική στιγμή και το κάθε νήμα προκειμένου να εκτελέσει το κομμάτι υπολογισμού που του αναλογεί,μπορεί να ζητήσει αρκετά bytes από την κύρια μνήμη της κάρτας γραφικών.Για να είναι αποδοτική η πρόσβαση στη μνήμη για κάθε νήμα,είναι αναγκαίο κάθε νήμα να βρίσκει τα δεδομένα που χρειάζεται ¨κοντά' του,δηλαδή σε μια μνήμη με μικρό latency όπως είναι οι μνήμες cache και η shared memory.

Η πρώτη μου προσέγγιση ήταν να τοποθετήσω τα δεδομένα στη κάρτα γραφικών παρόμοια με τη CPU.Στο host τα δεδομένα ήταν τοποθετημένα σε πίνακες από structs (array of structs).Αυτός ο τρόπος δεν δούλεψε καλά.Τα structs είχαν μεγάλο μέγεθος και η πρόσβαση στη μνήμη για κάθε νήμα δεν ήταν coalesced.Αυτό είχε σαν αποτέλεσμα να μην αξιοποιείται πλήρως το bandwidth της κύριας μνήμης και το κυριότερο τα cache misses ήταν πολλά για κάθε thread.

Η λύση ήρθε με τη χρήση struct of arrays.Κάθε πρόσβαση νήματος σε ένα array της μνήνης ήταν coalesced,αφού γειτονικά νήματα ζητούσαν γειτονικές θέσεις μνήμης. Έτσι,όπως είναι αναμενόμενο, αυξήθηκαν ακομά και τα cache hits της μικρής κρυφής μνήμης της κάρτας γραφικών.

BVH στη κάρτα γραφικών

Το κρισιμότερο κομμάτι στην απεικόνιση του ray tracing είναι η διαπέραση της BVH.Κάθε ακτίνα της σκηνής πρέπει να διαπεράσει τουλάχιστον μια φορά τη δομή BVH για να συνεχιστεί ο υπολογισμός.Με άλλα λόγια κάθε thread της GPU ουσιαστικά θα διαπεράσει το δέντρο.Πρέπει λοιπόν η διαπέραση να είναι αποδοτική από τη μεριά της μνήμης και φυσικά να σέβεται το μοντέλο εκτέλεσης της GPU.Υπενθυμίζεται ότι τα νήματα μιας κάρτας γραφικών τρέχουν σε ομάδες των 32 που ονομάζονται warps.Για κάθε warp υπάρχει μόνο ένας unit controller που σημαίνει οτι και τα 32 νήματα πρέπει σε κάθε κύκλο να εκτελούν την ίδια εντολή.Διαφορετικά παρατηρείται divergence,όπως περιγράφεται στο κεφάλαιο

2.

Το BVH δέντρο για να μεταφερθέι στη κάρτα κάρτα γραφικών και να είναι η πρόσβαση αποδοτική,πρέπει πρώτα να μετατράπεί.Στη μνήμη του host,δηλαδή στη μνήμη που έχει πρόσβαση η CPU,το δέντρο αποθηκεύεται σε κόμβους και κάθε κόμβος ξεχωριστά δεικτοδοτείται με pointers.Αυτή η προσέγγιση δεν θα δουλέψει καλά στη μνήμη της κάρτας γραφικών,αφού τίποτα δεν μας εγγυάται οτι η μνήμη των κόμβων του δέντρου θα βρίσκονται σε συνεχόμενες θέσεις μνήμης.Συνεπώς το δέντρο,προτού μεταφερθεί στη GPU,μετατρέπεται πρώτα σε πίνακα από structs.Κάθε θέση του πίνακα είναι ένας κόμβος του αρχικού δέντρου.Οι δείκτες προς τα αριστερά και δεξιά παιδιά των κόμβων αντικαθιστούντε από indices στις αντίστοιχες θέσεις κόμβων στον πίνακα.Η διαπέραση της BVH κατά τη μετατροπή γίνεται κατα βάθος (depth-first),αφού η διαπέραση των νημάτων στη GPU θα γίνει επίσης κατα βάθος.Τέλος γίνεται μια ακόμη μετατροπή και το δέντρο από array of structs γίνεται struct of arrays για να είναι αποδοτική η πρόσβαση μνήμης.

Η διαπέραση της BVH,όπως περιγράφεται από τον αλγόριθμο 7 του προηγούμενου κεφαλαίου,είναι αναδρομική.Η αναδρομή είναι περιορισμένη στην αρχιτεκτονική των GPU,αφού μόνο GPUS με compute capability 2.0 και πάνω υποστηρίζουν αναδρομικές συναρτήσεις.Συνεπώς η διαπέραση πρέπει να γίνει επαναληπτικά με τη χρήση στοίβας.Κάθε thread εκτέλεσης της κάρτας γραφικών διαθέτει τη δική του ιδιωτική στοίβα.

Ο αλγόριθμος 8 παρουσιάζει σε μορφή ψευδοκώδικα την επαναληπτική μέθοδο διαπέρασης.Αρχικά ελέγχεται προς διασταύρωση η ρίζα του δέντρου.Αν η ακτίνα δεν διασταυρώνεται με τη ρίζα,τότε εγγυημένα η ακτίνα δεν θα πετύχει καμία επιφάνεια της σκηνής.Έπειτα επαναληπτικά ελέγχονται τα δύο παιδιά κάθε κόμβου.Το κοντινότερο παιδί ανατίθεται ως ο επόμενος κόμβος προς εκτέλεση και το άλλο παιδί μπαίνει στη στοίβα.Στην περίπτωση που μόνο ένα παιδί διασταυρώνεται με την ακτίνα,τότε δεν χρειάζεται να προσθέσουμε τίποτα στη στοίβα.Αν κανένα παιδί δεν διασταυρώνεται με την ακτίνα, τότε γίνεται pop από την στοίβα και εξετάζεται ο νέος κόμβος.Όταν η διαδικασία φτάσει σε φύλλο του δέντρου,τότε ελέγχονται όλες οι επιφάνεις της σκηνής που περικλείονται στο συγκεκριμένο φύλλο,ανανεώνεται η ελάχιστη απόσταση επιφάνειας όπως και πριν και γίνεται pop ο επόμενος κόμβος προς εξέταση από τη στοίβα.Η διαδικασία σταματά όταν η στοίβα αδειάσει,δηλαδή δεν ύπάρχει άλλος κόμβος του δέντρου προς εξέταση.

Με την επαναληπτική διαπέραση του δέντρου με χρήση στοίβας αποφεύγονται αναδρομικές κλήσεις και αναγκάζονται όλα τα νήματα ενός warp να εκτελούν το ίδιο κομμάτι κώδικα.Οι διακλαδώσεις που παρουσιάζονται όμως στην

Algorithm 8 BVH stack Traversal

```
Require: Stack,Ray
 1: procedure TraverseStack
 2:     Stack ← Empty
 3:     currNode ← Root
 4:     if !ray Intersects Root then
 5:         return false
 6:     end if
 7:     while currNode ≠ NULL do
 8:         if currNode = BvhNODE then
 9:             intersect both children
10:             if intersection Not Found then
11:                 currNode ← stack pop
12:             end if
13:             currNode ← Nearest Child
14:             stack push far child
15:         else
16:             intersectRayWithLeaf()
17:             updateMinDistance()
18:             currNode ← stack pop
19:         end if
20:     end while
21: end procedure
```

επαναληπτική διαπέραση προκαλούν ένα μεγάλο πρόβλημα επίδοσης,το divergence.Υπενθυμίζεται ότι divergence παρατηρείται όταν τα νήματα του ίδιου warp εκτελέσουν διαφορετικό branch κώδικα.Η διαπέραση,όπως περιγράφεται στον αλγόριθμο 8,δεν λαμβάνει υπόψιν το θέμα του divergence.Αρχικά είναι πολύ πιθανό πολλά νήματα να μην διασταυρώνοται με τη ρίζα του δέντρου και ο υπολογισμός τους να τελειώσει πολύ γρήγορα,ενώ ένα υποσύνολο του warp να διασταυρώνεται και να χρειάζεται να διαπεράσει ένα μεγάλο κομμάτι του δέντρου.Επίσης στον αλγόριθμο 8 δεν λαμβάνεται υπόψιν ότι όλες οι ακτίνες,δηλαδή όλα τα νήματα,έχουν ουσιαστικά διαφορετική κατεύθυνση και είναι δυνατόν να επιλέξουν διαφορετικό επόμενο κόμβο προς εξέταση.Σε αυτήν την περίπτωση τα πράγματα είναι πολύ άσχημα,αφού ένα υποσύνολο του warp μπορεί να βρίσκεται σε φύλλο και να εξετάζει διασταυρώσεις με επίφανειες της σκηνής,ενώ τα υπόλοιπα είτε να βρίσκονται σε κάποιον ενδιάμεσο κόμβο της BVH ή να έχουν τελειώσει εντελώς τη διαπέραση.

Είναι φανερό οτι πρέπει με κάποιο τρόπο τα νήματα του ίδιου warp να συγχρονιστούν και να αποφασίσουν από κοινού ποιος θα είναι ο επόμενος κόμβος προς εξέταση.Ευτυχώς η CUDA προφέρει τρόπους συγχρονισμού νημάτων ακόμα και σε επίπεδο warp με τη χρήση warp voting συναρτήσεων.Δύο τέτοιες συναρτήσεις είναι οι *_all(int predic)* και *_any(int predic)*. Η *_all()* επιστρέφει μη μηδενική τιμή όταν όλα τα νήματα ψηφίσουν με μη μηδενικό predicate,διαφορετικά επιστρέφει μηδέν,ενώ η *_any()* επιστρέφει μη μηδενική τιμή αν έστω και ένα νήμα ψηφίσει με μη μηδενικό predicate. Με την *_all()* μπορούμε να αναγκάσουμε όλα τα νήματα του warp να διαπέρασουν το δέντρο αν έστω και ένα διασταυρώνεται με τη ρίζα,ενώ με την *_any()* να συγχρονίσουμε τους ενδιάμεσους κόμβους,δηλαδή αν έστω και ένα νήμα αποφασίσει να επισκεφτεί έναν κόμβο,τότε όλα τα νήματα του warp θα τον επισκεφτούν.

Οι warp voting συναρτήσεις όμως έχουν ένα σημαντικό μειονέκτημα.Κάθε φορά που αποφασίζεται ποιος θα είναι ο επόμενος κόμβος προς εξέταση,θα υπάρχουν νήματα στο ίδιο warp που δεν θα χρειάζεται να τον επισκεφτούν είτε επειδή η απόσταση της δομής οριοθέτησης είναι μεγαλή είτε κυρίως γιατί η ακτίνα που τους αντιστοιχεί δεν διασταυρώνεται με τη δομή οριοθέτησης.Με αυτόν τον τρόπο τα νήματα του warp σπαταλάνε χρόνο υπολογίζοντας ουσιαστικά σκουπίδια αφού ¨αναγκάζονται' να επισκεφτούν κόμβους που δεν θα επηρεάσουν την απεικόνιση για την συγκεκριμένη ακτίνα.Παρ΄όλα αυτά η επίδοση που παρατηρείται λόγω συγχρονισμού μέσω warp voting συναρτήσεων είναι μεγαλύτερη.Ο χρόνος που σπαταλάται από την GPU λόγω divergence είναι συκριτικά αρκετά μεγαλύτερος από τον αφελή υπολογισμό σκουπιδιών για κάθε νήμα.Συνεπώς η χρήση warp voting μεθόδων βελτιώνει το χρόνο υπολογισμού της διαπέρασης

της ακτίνας από την BVH που έχει με τη σειρά του ως αποτελέσμα μικρότερο χρόνο απεικόνισης για κάθε frame του ray tracing.

5.2 Μετρήσεις

Σε αυτήν την παράγραφο παρουσιάζονται μετρήσεις επιδόσεων της μεθόδου ray tracing για την CPU και GPU.Η CPU που χρησιμοποιήθηκε είναι η AMD phenom 955 και διαθέτει τέσσερα physical cores χωρίς SMT και η GTX 760 GPU της Nvidia.Η μέθοδος εξετάζεται σε διάφορες τρισδιάστατες ψηφιακές σκηνές που αποτελούνται από τρίγωνα,δηλαδή δεν υπάρχουν άλλου είδους επιφάνειες όπως σφαίρες κτλ,σε πολλαπλές αναλύσεις εικόνων από 720p μέχρι 4K.Κατά τη διάρκεια των μετρήσεων χρησιμοποιήθηκαν μόνο primary rays,δηλαδή ακτίνες που ξεκινούν από την κάμερα.

Οι ψηφιακές σκηνές μετρήσεων επιλέχτηκαν με διαφορετικό μέγεθος πολυπλοκότητας για την καλύτερη εξαγωγή συμπερασμάτων επίδοσης μεταξύ CPU και GPU.Οι σκηνές είναι οι παρακάτω:

- 3D monkey Suzanne.Μικρή σκηνή.Αριθμός τριγώνων:968 [Ble]
- Bunny.Μεσαία σκηνή.Αριθμός τριγώνων: 69.666 [mod]
- Fairy Forest.Μεγάλη σκηνή.Αριθμός τριγώνων: 172.669 [ani]

Όλες οι σκηνές που εξετάζονται είναι στατικές,διαθέτουν δύο πηγές φωτός και δεν υπάρχει καθόλου animation.Επίσης το BVH δέντρο οριοθετείται σεύψος 20 για κάθε σκηνή.

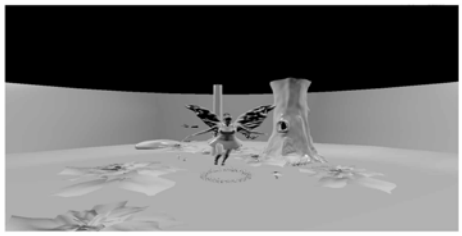
Σχήμα 5.1: Suzanne

Σχήμα 5.2: Bunny

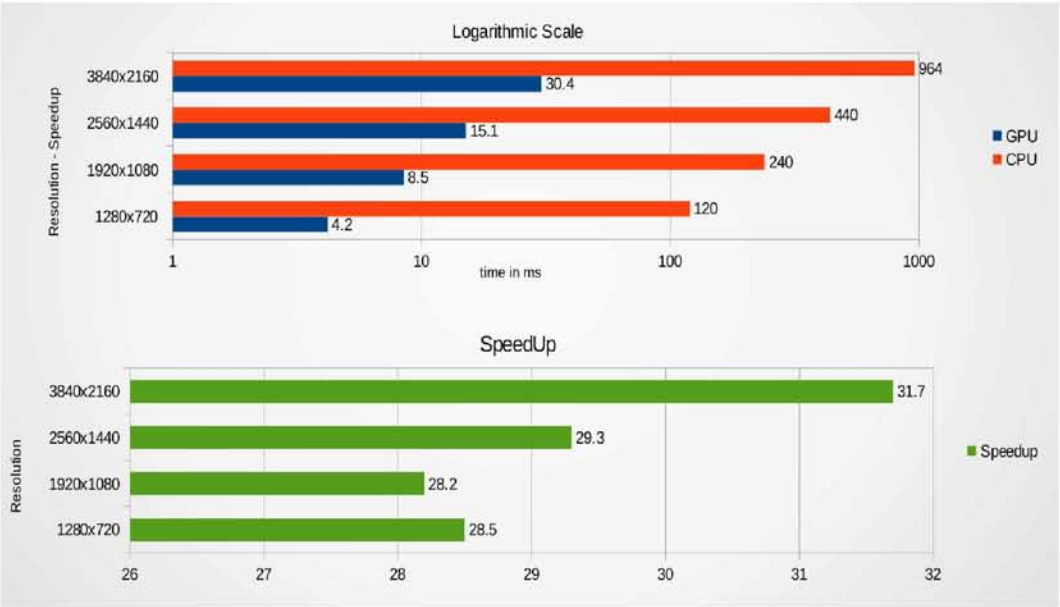
Οι χρόνοι που παρουσιάζονται περιλαμβάνουν την απεικόνιση ενός μόνο καρέ.Δεν λαμβάνεται υπόψιν ο χρόνος κατασκευής του BVH δέντρου.Ο υπολογισμός στη CPU γίνεται με OpenMP.Ο χρόνος της GPU είναι το άθροισμα του χρόνου μεταφοράς της τελικής εικόνας στο host και του χρόνου απεικονισης.Το λειτουργικό σύστημα που χρησιμοποιήθηκε είναι το Ubuntu 14.04.Ο υπολογισμός γίνεται σε ένα mega-kernel.

Παρατηρήσεις και Περιορισμοί

Στη μικρή σκήνή (Suzanne) το speedup παραμένει σχεδόν σταθερό σε κάθε ανάλυση.Φυσικά ο χρόνος υπολογισμού διαφέρει αφού η απεικόνιση γίνεται κάθε

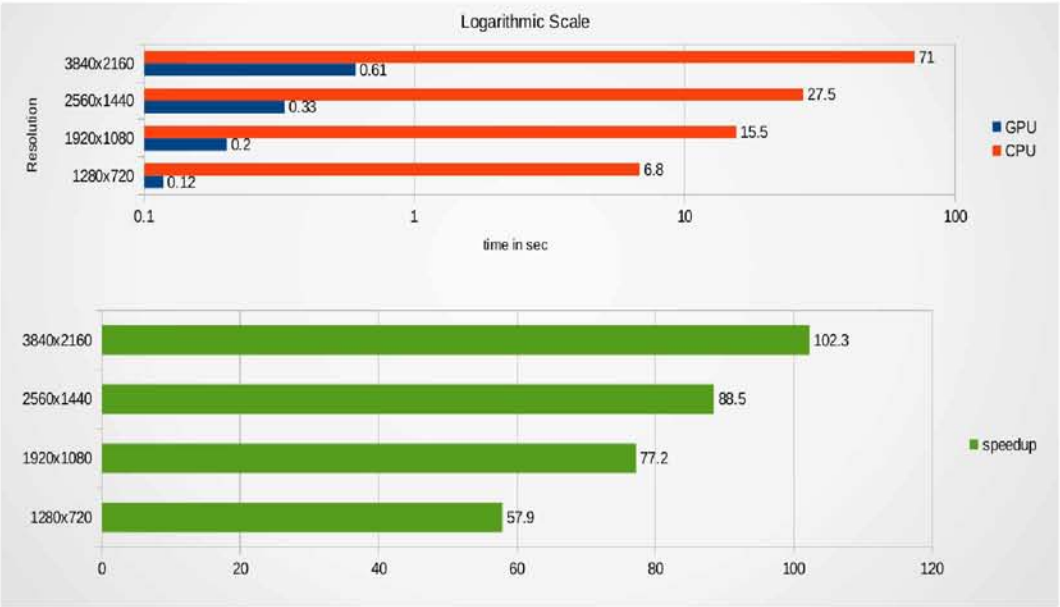


Σχήμα 5.3: Fairy Forest



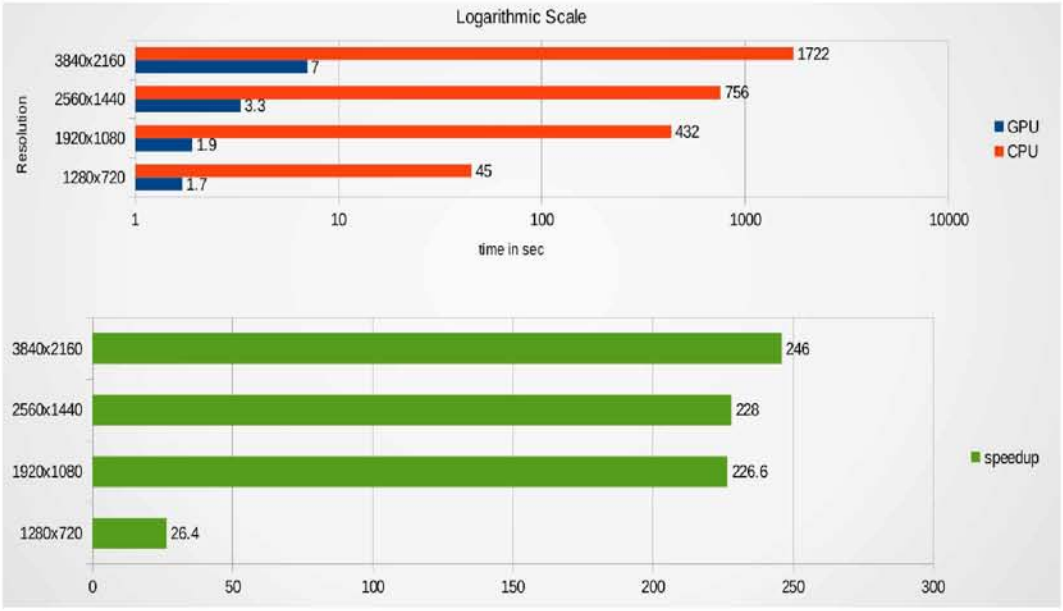
Σχήμα 5.4: Επίδοση για τη σκηνή:Suzanne

φορά σε μεγαλύτερη ανάλυση.Μέχρι και την ανάλυση 4K το speedup παραμένει κοντά στο 30x.Το BVH δέντρο στη μικρή σκηνή έχει λίγους κόμβους και η διαπέραση για κάθε νήμα(CPU ή GPU thread) διαρκεί μικρό χρονικό διάστημα.Η φάση του φωτισμού δεν είναι computetionaly intensive και δεν παίζει σημαντικό παράγοντα στον τελικό χρόνο απεικόνισης που παρατηρείται στη CPU και τη GPU.



Σχήμα 5.5: Επίδοση για τη σκηνή:Bunny

Το παραπάνω δεν παρατηρείται στις υπόλοιπες σκηνές.Κάθε φορά που μεγαλώνουμε την ανάλυση,το speedup αυξάνεται σημαντικά.Μεγαλώνοντας την ανάλυση ουσιαστικά αυξάνουμε και τον παραλληλισμό του υπολογισμού,αφού ο παραλληλισμός γίνεται σε επίπεδο ακτίνας,δηλαδή με άλλα λογια σε επίπεδο sample/pixel της τελικής εικόνας.Οι GPUs είναι σχεδιασμένες για βαριούς παράλληλους υπολογισμούς και αυτό ταιριάζει στη φύση του προβλήματος.Ειδικά σε μεγάλες σκηνές όπως fairy η διαφορά επίδοσης σε υψηλές αναλύσεις είναι τεράστια.Φυσικά μεγαλώνοντας την ανάλυση απεικόνισης,ο υπολογισμός διαρκεί και περισσότερο χρόνο,ωστόσο το speedup μεγαλώνει.



Σχήμα 5.6: Επίδοση για τη σκηνή:Fairy Forest

Το BVH δέντρο στις μεγάλες σκηνές διαθέτει αρκετές επιφάνειες(οι συγκεκριμένες αποτελούνται μόνο από τρίγωνα) και η διαπέραση μιας ακτίνας ακόμα και από ένα φύλλο είναι ακριβή για ένα νήμα της CPU.Η GPU κλιμακώνει καλύτερα όμως τον υπολογισμό γιατί η αρχιτεκτονική της προσφέρει χιλιάδες ενεργά threads ανα πάσα χρονική στιγμή σε αντίθεση με τη CPU στην οποία σε κάθε thread αντιστοιχούν χιλιάδες ακτίνες.Συνεπώς,λόγω της παράλληλης φύσης του υπολογισμού της μεθόδου ray tracing,περισσότερα νήματα προσφέρουν καλύτερη επίδοση.

Ένας τρόπος για να κρύβει η GPU latencies λόγω πρόσβασης μνήμης είναι οι εναλλαγές νημάτων.Το κόστος εναλλαγής νημάτων σε ένα multiproccessor της GPU είναι σχεδόν μηδενικό και επιλέγουμε να δημιουργούμε μεγάλες γεωμετρίες νημάτων έτσι ώστε να υπάρχουν κάθε στιγμή διαθέσιμα νήματα προς εκτέλεση.Για να τρέξει ένα γρουπ από νήματα σε έναν multiproccessor είναι αναγκαίο να είναι επαρκής οι παρακάτω πόροι για την εκτέλεση τους: α) registers και β) shared memory.Όταν δεν είναι επαρκής οι καταχωρητές για όλα τα νήματα ή η shared memory για το thread block, τότε το συγκεκριμένο γρουπ νημάτων δεν είναι δυνατόν να εκτελεστεί προτού οι παραπάνω πόροι γίνουν διαθέσιμοι.

Ο kernel για το ray tracing στις παραπάνω μετρήσεις περιορίζεται από τους

καταχωρητές.Κάθε νήμα της GPU χρειάζεται πολλούς καταχωρητές, πάνω από 64,με αποτέλεσμα να μην είναι δυνατό να κρυφτούν εύκολα τα cache misses και η καθυστέρηση πρόσβασης στη global memory λόγω έλειψης γκρουπ νημάτων που διαθέτουν τους αναγκαίους πόρους.Ο kernel δεν χρησιμοποιεί καθόλου shared memory,οπότε μόνο οι καταχωρητές παίζουν ρόλο.Μάλιστα επειδή κάθε νήμα χρειάζεται παραπάνω από 64 καταχωρητές,παρατηρείται register spilling.Register spilling έχουμε όταν το πλήθος των διαθέσιμων καταχωρητών δεν επαρκεί για την εκτέλεση του υπολογισμού και ο μεταγλωτιστής αναγκάζεται να χρησιμοποιεί την τοπική μνήμη των νημάτων για την προσωρινή αποθήκευση τους. Αυτό είναι συχνό φαινόμενο στη CPU,αφού οι διαθέσιμοι καταχωρητές είναι πάντα λιγότεροι από όσους χρειάζεται ένας υπολογισμός και το αντίτιμο σε χρόνο εκτέλεσης είναι σχετικά μικρό.Στη GPU όμως το πρόβλημα είναι μεγάλο,αφού έτσι όχι μόνο αυξάνεται ο αριθμός προσβάσεων στη τοπική μνήμη των νημάτων μεγαλώνοντας έτσι το χρόνο υπολογισμού,αλλά επίσης μειώνεται το occupancy,δηλαδή το ποσοστό των ενεργών νημάτων προς το τον μέγιστο αριθμό νημάτων που μπορούν να τρέξουν ταυτόχρονα.Έτσι είναι δύσκολο η GPU να κρύψει με εναλλαγές νημάτων την πρόσβαση στη μνήμη που έχει ως αποτέλεσμα ο υπολογισμός να διαρκεί περισσότερο.

Κεφάλαιο 6

Επίλογος

Το ray tracing είναι μια απαιτητική υπολογιστικά μέθοδος απεικόνισης ρεαλιστικών γραφικών υπολογιστών.Η παράλληλη φύση της κάνει τον αλγόριθμο απεικόνισης embarrassingly parallel και ο υπολογισμός απλοϊκά χωρίζεται σε παράλληλα νήματα εκτέλεσης.Οι σύγχρονες κάρτες γραφικών αποτελούν many-core συστήματα για υπολογισμούς γενικού σκοπού στις οποίες μπορούμε να τρέξουμε μαζικά χιλιάδες νήματα.Δυστυχώς οι GPU είναι σχεδιασμένες για data parallel προβλήματα ενώ το ray tracing είναι task parallel και η επίδοση που αναμένεται δεν είναι υψηλή.Ωστόσο,το speedup που παρατηρείται στις μετρήσεις που παρουσιάζονται στο κεφάλαιο 5 σε σχέση με μια σημερινή CPU φτάνουν ακόμη και το 246x για μεγάλες σκηνές.Παρ΄όλα αυτά ο χρόνος απεικόνισης ενός frame είναι απαγευρετικός για εφαρμογές πραγματικού χρόνου και πολύπλοκες τρισδιάστατες ψηφιακές σκηνές.

Μελλοντικές Επεκτάσεις

Αυτό που δεν δοκιμάστηκε στη διπλωματική εργασία είναι η επίδοση της μεθόδου σε πολλαπλές GPU και να εξεταστεί η συμπεριφορά του αλγορίθμου σε ένα τέτοιο σύστημα.Από τα αποτελέσματα και τις μετρήσεις από μια μόνο GPU θεωρείται ότι το speedup θα είναι γραμμικό με τον αριθμό των καρτών γραφικών που θα χρησιμοποιηθούν.Επίσης να προστεθούν στην απεικόνιση περισσότερα εφέ γραφικών,όπως soft shadows, glossy reflections και depth of field.

Bibliography

[ani] animrep. *The Utah 3D Animation Repository*. URL: http://www.sci.utah.edu/~wald/animrep/.

[Ble] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Blender Institute, Amsterdam. URL: http://www.blender.org.

[Gre06] Bram de Greve. "Reflections and Refractions in Ray Tracing". In: (2006).

[Gro] Khronos Group. *The Industry's Foundation for High Performance Graphics*. URL: https://www.khronos.org/opengl/.

[mod] Stanford models. *The Stanford 3D Scanning Repository*. URL: http://graphics.stanford.edu/data/3Dscanrep/.

[Nvi] Nvidia. *Visual Profiler*. URL: https://developer.nvidia.com/nvidia-visual-profiler.

[NVI14] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide*. Aug. 2014.

[Ope11] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 3.1*. 2011. URL: http://www.openmp.org/mp-documents/OpenMP3.1.pdf.

[Pho75] Bui Tuong Phong. "Illumination for Computer Generated Pictures". In: *Commun. ACM* 18.6 (June 1975), pp. 311–317. ISSN: 0001-0782. DOI: 10.1145/360825.360839. URL: http://doi.acm.org/10.1145/360825.360839.

[Sch97] Robert R. Schaller. "Moore's Law: Past, Present, and Future". In: *IEEE Spectr.* 34.6 (June 1997), pp. 52–59. ISSN: 0018-9235. DOI: 10.1109/6.591665. URL: http://dx.doi.org/10.1109/6.591665.

[Tru97] Thomas Moller - Ben Trumbore. "Fast, Minimum Storage Ray/Triangle Intersection". In: *Journal of Graphics Tools* (1997).

[VTu] Intel VTune. *Performance Analyzer*. URL: `http://software.intel.com/en-us/intel-vtune/`.

[Wal07] Ingo Wald. "On fast construction of SAH-based bounding volume hierarchies". In: *Proceedings of the 2007 IEEE/EG Symposium on Interactive Ray Tracing. IEEE*. 2007.

[Wik] Wikipedia. *Phong reflection model*. URL: `http://en.wikipedia.org/wiki/Phong_reflection_model`.

[WK06] Carsten Wächter and Alexander Keller. "Instant Ray Tracing: The Bounding Interval Hierarchy". In: (2006). URL: `http://people.cs.clemson.edu/~dhouse/courses/405/papers/bounding-interval-WK06.pdf`.