Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών & Δικτύων

Διπλωματική Εργασία

Τομέας

Γραφικά Υπολογιστών

Θέμα

Μελέτη Τεχνικών Υφής (Texture Mapping)

Υπεύθυνος Καθηγητής: *κ. Ηλίας Χούστης*

Επιβλέπων: *κ. Αθανάσιος Γκαϊτατζής*

Ανδρεάδης Ανθούσης

Περιεχόμενα

1. Πρόλογος

Τα γραφικά υπολογιστών (computer graphics) αποτελούν έναν διαρκώς αναπτυσσόμενο κλάδο της πληροφορικής, με εφαρμογές σε πολλούς και διαφορετικούς τομείς όπως η εικονική πραγματικότητα, τα πολυμέσα, το λογισμικό παιχνιδιών, οι ιατρικές εφαρμογές κ.α. Ένα σημαντικό τμήμα αυτού του τομέα αποτελεί η δημιουργία εικόνων από τρισδιάστατα μοντέλα. Ένα μοντέλο για παράδειγμα, μπορεί να είναι η περιγραφή ενός φανταστικού κόσμου με πολυγωνικές επιφάνειες. Για την παραγωγή των εικόνων χρησιμοποιούνται τεχνικές που εκφράζουν τον τρόπο αλληλεπίδρασης του φωτός με τα αντικείμενα του μοντέλου. Πολλές από αυτές τις τεχνικές έχουν αναπτυχθεί ειδικά για τον τομέα των γραφικών, ενώ άλλες προέρχονται από τους τομείς των μαθηματικών και της φυσικής.

Η παρούσα διπλωματική εργασία, αφορά τις τεχνικές υφής (Texture Mapping). Η διαδικασία του Texture Mapping ουσιαστικά αποτελεί μια μέθοδο που προσδίδει ρεαλισμό στα αντικείμενα του τρισδιάστατου μοντέλου, δρώντας ανεξάρτητα από το μοντέλο φωτισμού. Χωρίς την διαδικασία του Texture mapping θα αντιμετωπίζαμε μεγάλο πρόβλημα στην απεικόνιση ρεαλιστικών αντικειμένων. Αυτό συμβαίνει γιατί στην πράξη τα περισσότερα αντικείμενα του περιβάλλοντός μας δεν αποτελούνται από αμιγή χρώματα ή από λεία στην επιφάνεια υλικά χωρίς διαταραχές. Συνεπώς, αν θέλαμε να χρησιμοποιήσουμε ένα απλό μοντέλο φωτισμού για να αποδώσουμε με ρεαλισμό ένα αντικείμενο, θα έπρεπε να διαιρέσουμε κάθε τμήμα του σε μικρά στοιχειώδη υπο-τμήματα, όπου το καθένα θα είχε το δικό του χρώμα, έτσι ώστε το συνολικό αποτέλεσμα να προσεγγίζει μια επιφάνεια με πολύπλοκη υφή. Ωστόσο η παραπάνω διαδικασία είναι υπολογιστικά ασύμφορη καθώς απαιτεί την δημιουργία και την επεξεργασία ενός μεγάλου αριθμού υπο-τμημάτων, από τα οποία αποτελείται το αρχικό αντικείμενο.

Για την απόδοση υφής στα αντικείμενα χρησιμοποιούνται διάφορες τεχνικές που υπάγονται στην κατηγορία μεθόδων απεικόνισης υφής (texture mapping). Αν έχουμε παραστήσει την υφή ως μια συνάρτηση ορισμένη σε ένα N-διάστατο χώρο (χώρος υφής) T^N, ο οποίος είναι υποσύνολο του $\Re^N$, τότε μπορούμε να θεωρήσουμε την απεικόνιση υφής σαν μια απεικόνιση $\mathrm{E}^3 \to \mathrm{T}^N$

ή $\Re^3 \to \mathrm{T}^N$, όπου E^3 και $\Re^3$ ο τρισδιάστατος χώρος των σημείων και των διανυσμάτων αντίστοιχα. Δηλαδή, κάθε σημείο των αντικειμένων του τρισδιάστατου χώρου μπορεί να χρωματιστεί σύμφωνα με την αντίστοιχη τιμή που βρίσκεται στο χώρο T^N, όταν μετασχηματίσουμε αυτό το σημείο του E^3 ή το αντίστοιχο διάνυσμα του $\Re^3$ στον χώρο της υφής T^N. Έτσι με αυτόν τον μετασχηματισμό μπορούμε να προσδώσουμε σε κάθε σημείο της επιφάνειας των αντικειμένων τη λεπτομέρεια που θέλουμε, χωρίς να πρέπει να μεταβάλλουμε τη μορφολογία του αντικειμένου. Η παραπάνω διαδικασία, αποτελεί το γενικό πλάνο του texture mapping, το οποίο μπορεί να τροποποιείται ανάλογα με τις απαιτήσεις του εκάστοτε μοντέλου.

Η διαδικασία του texture mapping δεν αντικαθιστά το μοντέλο φωτισμού. Για κάθε σημείο υπολογίζεται η υφή με τον παραπάνω τρόπο, η οποία στη συνέχεια σκιάζεται σε ποσοστό που επιβάλλει ο αλγόριθμος φωτισμού που χρησιμοποιείται. Κατά αυτόν τον τρόπο μπορούμε να ελέγξουμε τη διαφάνεια, την τραχύτητα του ανάγλυφου, τον ενδογενή φωτισμό, την ανακλαστικότητα και το συντελεστή διάθλασης των αντικειμένων ενός σκηνικού, ώστε να πάρουμε εικόνες που να προσεγγίζουν την πραγματικότητα με τον καλύτερο δυνατό τρόπο.

Η χρησιμότητα και η αναγκαιότητα των τεχνικών υφής έγκειται στο γεγονός ότι μειώνουν την επεξεργασία που χρειάζεται για την παραγωγή μοντέλων. Για παράδειγμα, με την δημιουργία μιας σφαίρας και την εφαρμογή στην επιφάνεια αυτής ενός texture προσώπου, γλιτώνουμε την επεξεργασία που θα ήταν απαραίτητο να εφαρμοστεί στο γεωμετρικό μοντέλο της σφαίρας ώστε να παραχθούν τα σχήματα των ματιών, της μύτης κ.τ.λ. Αυτός είναι και ο λόγος που ολόκληρη η βιομηχανία παιχνιδιών, έχει στραφεί σε αυτές τις τεχνικές για την απεικόνιση των «ακριβών» υπολογιστικά μοντέλων που χρησιμοποιούνται.

Οι τεχνικές υφής μπορούν να χωριστούν σε δύο βασικές κατηγορίες: τις image mapping και procedural mapping.

- **Image Mapping:** Η απεικόνιση χάρτη υφής σε αντικείμενα είναι η πιο συνηθισμένη τεχνική, η οποία και αναφέρεται ως η παλαιότερη μέθοδος απεικόνισης υφής. Η βασική ιδέα στην οποία αυτή στηρίζεται είναι η ύπαρξη

μιας εικόνας δύο διαστάσεων η οποία εκφράζεται ως ένα φραγμένο διάστημα ενός παραμετρικού χώρου T^2 με παραμέτρους u και v. Αυτό που ουσιαστικά συμβαίνει είναι να ορίζονται συναρτήσεις από τον E^3 χώρο, όπου και βρίσκονται τα αντικείμενά μας, στο φραγμένο διάστημα του T^2. Το τελικό αποτέλεσμα είναι το «τύλιγμα» ενός αντικειμένου του τρισδιάστατου χώρου με την δυσδιάστατη εικόνα. Ο τρόπος με τον οποίο θα γίνει το «τύλιγμα» εξαρτάται από τον μετασχηματισμό που επιλέγουμε για να προβάλουμε τα σημεία του τρισδιάστατου χώρου E^3 στον παραμετρικό χώρο T^2.

Υπάρχουν πολλοί μετασχηματισμοί με τους οποίους μπορούμε να ελέγξουμε τον τρόπο που θα προβληθούν τα σημεία ενός αντικειμένου στο χάρτη υφής που θέλουμε να χρησιμοποιήσουμε. Αυτοί οι μετασχηματισμοί, είναι γνωστοί ως συναρτήσεις απεικόνισης. Μια συνάρτηση απεικόνισης συνδέει τις παραμέτρους της υφής (u,v) με τα σημεία ενός αντικειμένου σε δύο στάδια. Αρχικά, οι παράμετροι u, v συνδέονται με τις παραμέτρους μιας ενδιάμεσης υποθετικής επιφάνειας στον τρισδιάστατο χώρο. Στο δεύτερο στάδιο τα σημεία του αντικειμένου απεικονίζονται πάνω στην υποθετική επιφάνεια. Οι βασικές συναρτήσεις απεικόνισης είναι οι εξής:

- ***Planar Mapping (Επίπεδη Συνάρτηση Απεικόνισης)***: Η ενδιάμεση επιφάνεια απεικόνισης είναι ένας επίπεδος προβολέας ο οποίος «φωτίζει» με μια παράλληλη δέσμη ακτινών το αντικείμενο. Χρησιμοποιείται πολύ συχνά και δίνει καλύτερα αποτελέσματα αν εφαρμοστεί σε σχετικά επίπεδες επιφάνειες.
- ***Cylindrical Mapping (Κυλινδρική Συνάρτηση Απεικόνισης):*** Το ενδιάμεσο στάδιο απεικόνισης είναι ο μοναδιαίος κύλινδρος ο οποίος είναι συμμετρικός ως προς τον άξονα Υ. Προτιμάται σε περιπτώσεις όπου έχουμε επιμήκη αντικείμενα, εξαιτίας της ομοιόμορφης απεικόνισης του χάρτη υφής γύρω από ένα άξονα και σε όλο το μήκος αυτού.
- ***Spherical Mapping (Σφαιρική Συνάρτηση Απεικόνισης):*** Το ενδιάμεσο στάδιο απεικόνισης είναι ο μοναδιαίος κύκλος. Χρησιμοποιείται όταν προσπαθούμε να αποδώσουμε υφή σε κοίλα αντικείμενα, όπως είναι τα σφαιρικά αντικείμενα.

- ***Box Mapping (Κυβική Συνάρτηση Απεικόνισης):*** Η συνάρτηση αυτή χρησιμοποιεί ως ενδιάμεσο στάδιο απεικόνισης έξι επίπεδες απεικονίσεις του ίδιου χάρτη υφής, τις οποίες και διατάσσει σαν πλευρές ενός κύβου.

- **Procedural Mapping:** Οι τεχνικές αυτής της κατηγορίας απεικονίσεως υφής, ακολουθούν μια διαφορετική μεθοδολογία που στηρίζεται σε αλγοριθμική απόδοση υφής σε κάθε σημείο του E^3. Βασικό χαρακτηριστικό της συναρτησιακής υφής είναι ότι ορίζεται μια απευθείας αντιστοίχηση οποιουδήποτε σημείου του E^3 σε μια μοναδική και συγκεκριμένη τιμή. Οι συναρτήσεις υφής είναι συνήθως συνεχής συναρτήσεις, ορισμένες σε ολόκληρο τον τρισδιάστατο χώρο. Η ανάθεση της υφής σε ένα σημείο γίνεται βάση του ακόλουθου προτύπου:

$$I = f_{tex}\left(\overline{P}\right),$$

όπου $\overline{P}$ είναι ένα σημείο του E^3, I η τιμή της υφής στο $\overline{P}$ και f_{tex} η συνάρτηση υφής.

Τα βασικά χαρακτηριστικά της συναρτησιακής υφής είναι:

- Ορίζεται σε ολόκληρο τον τρισδιάστατο χώρο
- Δεν υποφέρει από φαινόμενα pixelization
- Δεν υποφέρει από φαινόμενα γεωμετρικής παραμόρφωσης, καθώς δεν υπάρχει στάδιο μετασχηματισμού
- Χαρακτηρίζει ολόκληρο τον όγκο των αντικειμένων και όχι μόνο την επιφάνειά τους.

Αν και η τεχνική του procedural mapping εμφανίζει πολλά πλεονεκτήματα σε σύγκριση με το image mapping, δεν μπορεί να το αντικαταστήσει, καθώς η μία τεχνική συμπληρώνει την άλλη. Για τον παραπάνω λόγο δεν είναι δυνατή η απεικόνιση προκαθορισμένων σχεδίων με χρήση μόνο του procedural mapping, όπως και δεν είναι δυνατή η απεικόνιση ρεαλιστικών ανάγλυφων χρησιμοποιώντας μόνο image mapping.

Στα πλαίσια εκπόνησης της παρούσας διπλωματικής εργασίας, πραγματοποιήθηκε μελέτη Texture Mapping τεχνικών που έχουν ως στόχο την ορθή αναπαράσταση ανάγλυφης υφής σε αντικείμενα (αποτελούν

συνδυασμό των image και procedural mapping). Οι τεχνικές που μελετήθηκαν και οι οποίες υλοποιήθηκαν σε OpenGL Shader Language (εκτός της Emboss Bump Mapping) παρουσιάζονται στο 3^{ο} Κεφάλαιο και είναι οι εξής:

- Bump Mapping
- Emboss Bump Mapping
- Normal Mapping
- Parallax (Offset) Mapping
- Relief Mapping
- Vertex Displacement Mapping

2. OpenGL Shading Language

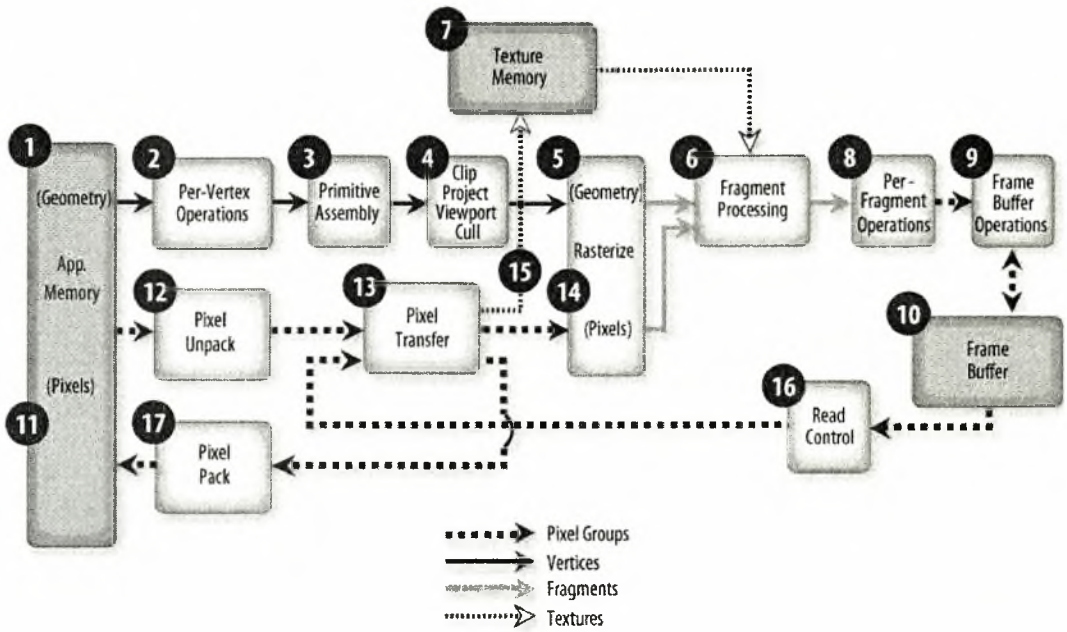
2.1 OpenGL και εισαγωγή στην OpenGL Shading Language

Το OpenGL αποτελεί ένα API (Application Program Interface) που έχει ως στόχο την παροχή πρόσβασης στις δυνατότητες του υλικού γραφικών. Είναι σχεδιασμένο με τέτοιο τρόπο, ώστε να αποτελεί τη χαμηλότερη δυνατή διασύνδεση για πρόσβαση στο υλικό, χωρίς όμως να εξαρτάται από αυτό. Το OpenGL έχει υλοποιηθεί για διάφορα λειτουργικά περιβάλλοντα, όπως τα Macs, τα PCs και τα UNIX-based, ενώ υποστηρίζεται από διάφορες αρχιτεκτονικές υλικού, είτε αυτές παρέχουν μικρή είτε μεγάλη επιτάχυνση στον τομέα των γραφικών.

Βασική διεργασία του OpenGL αποτελεί η μετατροπή δεδομένων που παρέχονται από κάποια εφαρμογή σε δεδομένα που εμφανίζονται στην οθόνη. Αυτή η διεργασία είναι γνωστή ως *Rendering*. Τυπικά οι υπολογισμοί του Rendering επιταχύνονται από ειδικά σχεδιασμένο υλικό (κάρτες Γραφικών), αλλά μερικοί ή και όλοι από αυτούς, μπορούν να πραγματοποιηθούν και από κάποια υλοποίηση λογισμικού που εκτελείται στην CPU *(Central Processing Unit)*. Το OpenGL παρέχει διαφάνεια επεξεργασίας και έτσι δίνει την δυνατότητα στους χρήστες, να αγνοήσουν τον τρόπο που γίνεται ο διαχωρισμός μεταξύ των υπολογισμών που αναλαμβάνει η CPU και η GPU *(Graphics Processing Unit)*. Το σημαντικό είναι ότι τα αποτελέσματα του Rendering πληρούν τις προϋποθέσεις που επιβάλλονται από το specification του API.

Το υλικό που είναι υπεύθυνο για την απεικόνιση των γραφικών και την διατήρηση των δεδομένων της οθόνης, ονομάζεται *«Επιταχυντής Γραφικών»*. Οι επιταχυντές γραφικών, έχουν τυπικά μια περιοχή μνήμης που είναι αφιερωμένη στην διατήρηση των δεδομένων της οθόνης. Έτσι κάθε pixel αναπαρίσταται στον επιταχυντή με κάποια bytes μνήμης. Αυτή η περιοχή μνήμης ονομάζεται *«Display Memory»*. Πέρα από αυτήν, κάθε επιταχυντής γραφικών, διαθέτει τυπικά και μια περιοχή μνήμης που ονομάζεται «*Offscreen Memory»,* η οποία δεν απεικονίζεται και η οποία περιέχει τα δεδομένα που

δεν είναι στην οθόνη. Τέλος, η περιοχή της μνήμης που αλλάζει από την διαδικασία του OpenGL Rendering, ονομάζεται *«Frame Buffer»*.



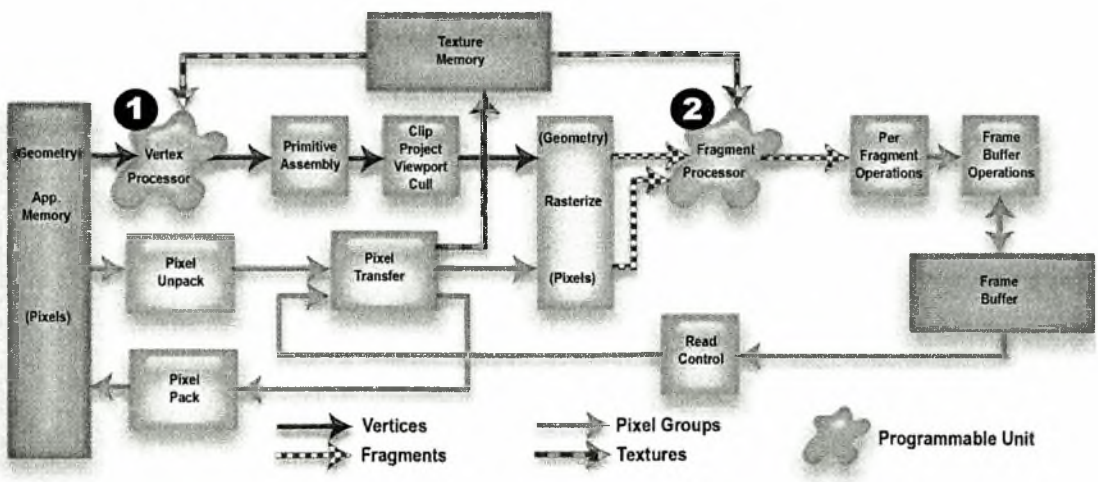
Εικόνα 1: OpenGL Fixed Functionality

Τα τελευταία χρόνια, έχει αλλάξει η γενική κατεύθυνση που ακολουθούνταν στον τομέα του υλικού των γραφικών. Συγκεκριμένα, έχουν αντικατασταθεί οι σταθερές λειτουργικότητες με προγραμματιζόμενες, κυρίως σε τομείς όπου είχε αυξηθεί πολύ η πολυπλοκότητά τους. Δύο από αυτούς τους τομείς, είναι η επεξεργασία κορυφών (vertex processing) και η επεξεργασία εικονοστοιχείων (fragment-pixel processing). Η διαδικασία του Vertex Processing, περιέχει τις λειτουργίες που εκτελούνται σε κάθε κορυφή των αντικειμένων, οι οποίες είναι κυρίως μετασχηματισμοί και υπολογισμοί φωτισμού. Τα Fragments, είναι δομές δεδομένων ανά εικονοστοιχείο, που δημιουργούνται από τον Rasterizer (Καμβάς) (Βλέπε Εικόνα 1). Ένα εικονοστοιχείο, περιέχει όλα εκείνα τα δεδομένα που είναι απαραίτητα για την ανανέωση μιας συγκεκριμένης θέσης του frame buffer. Η διαδικασία του Fragment Processing περιέχει εκείνες τις λειτουργίες που εκτελούνται ανά εικονοστοιχείο. Αυτές οι λειτουργίες είναι κυρίως η ανάγνωση από την μνήμη υφής και η εφαρμογή της αναγνωσθείσας τιμής.

Με την OGSL (OpenGL Shading Language), η σταθερή λειτουργικότητα των Vertex Processing και Fragment Processing έχει επαυξηθεί με την χρήση προγραμματιζόμενων μερών τα οποία επιτυγχάνουν ότι και η σταθερή λειτουργικότητα αλλά και πολλά περισσότερα. Η γλώσσα αυτή δίνει την δυνατότητα στους προγραμματιστές εφαρμογών να καθορίσουν οι ίδιοι την επεξεργασία που μπορεί να πραγματοποιηθεί σε αυτά τα προγραμματιζόμενα σημεία του OpenGL Pipeline.

Ο OGSL κώδικας που θα χρησιμοποιηθεί σε κάποιο από τα παραπάνω προγραμματιζόμενα στάδια, ονομάζεται «Shader». Υπάρχουν δύο ειδών Shaders: οι *«Vertex Shaders»* και οι *«Fragment Shaders»*. Το OpenGL παρέχει μηχανισμούς για την μεταγλώττιση των shaders και την διασύνδεσή τους ώστε να σχηματιστεί ο εκτελέσιμος κώδικας που ονομάζεται *«Program»*. Ένα program περιέχει ένα ή περισσότερα εκτελέσιμα που μπορούν να εκτελεστούν στις προγραμματιζόμενες μονάδες επεξεργασίας του OpenGL.

Η OpenGL Shading Language, έχει τις ρίζες της στην γλώσσα προγραμματισμού C, ενώ διαθέτει ένα πλούσιο σύνολο από τύπους δεδομένων οι οποίοι περιλαμβάνουν Vectors και Matrix τύπους δεδομένων, που χρησιμοποιούνται σε μεγάλο βαθμό στον μαθηματικό υπολογισμό των 3D γραφικών. Η γλώσσα αυτή περιέχει και μηχανισμούς της C++, όπως τη δήλωση μεταβλητών εκεί όπου χρειάζονται και όχι μόνο στην αρχή του κάθε μπλοκ. Πέραν αυτών των χαρακτηριστικών, η OGSL περιέχει ένα εκτεταμένο σύνολο built-in συναρτήσεων που βοηθούν στην κατασκευή αλγορίθμων σκίασης.



Εικόνα 2: OpenGL Functionality with OGSL Programmable Units

Μερικές από τις δυνατότητες που παρέχει η OGSL στους προγραμματιστές, είναι οι παρακάτω:

- Ρεαλιστικές υφές υλικών, όπως μέταλλο, ξύλο, πέτρα κ.α.
- Ρεαλιστικά εφέ φωτισμού, όπως spot lights και σκίαση
- Ρεαλιστικά φαινόμενα φωτιάς, καπνού, νερού κ.α.
- Προηγμένες τεχνικές φωτοσκίασης, όπως το ray tracing.
- Δυναμικές συναρτησιακές υφές
- Τεχνικές επεξεργασίας εικόνας, όπως unsharp masking, complex blending κ.α.
- Τεχνικές antialiasing προγραμματιζόμενες από τον χρήστη

Αρκετές από τις παραπάνω τεχνικές, ήταν διαθέσιμες και στο παρελθόν, όμως μόνο μέσω software υλοποιήσεων. Το γεγονός ότι οι τεχνικές αυτές υλοποιούνται πλέον με hardware επιτάχυνση, αυξάνει δραματικά την απόδοση του Rendering, ενώ την ίδια στιγμή η CPU απαλλάσσεται από αυτούς τους υπολογισμούς και μπορεί να χρησιμοποιηθεί για άλλες διεργασίες, όπως είναι η A.I (Τεχνητή Νοημοσύνη).

2.2 Τύποι επικοινωνίας με το εξωτερικό περιβάλλον

Η OGSL εισάγει τρεις νέους ειδικούς τύπους δεδομένων που αποτελούν τη διασύνδεση των shaders με το εξωτερικό τους περιβάλλον.

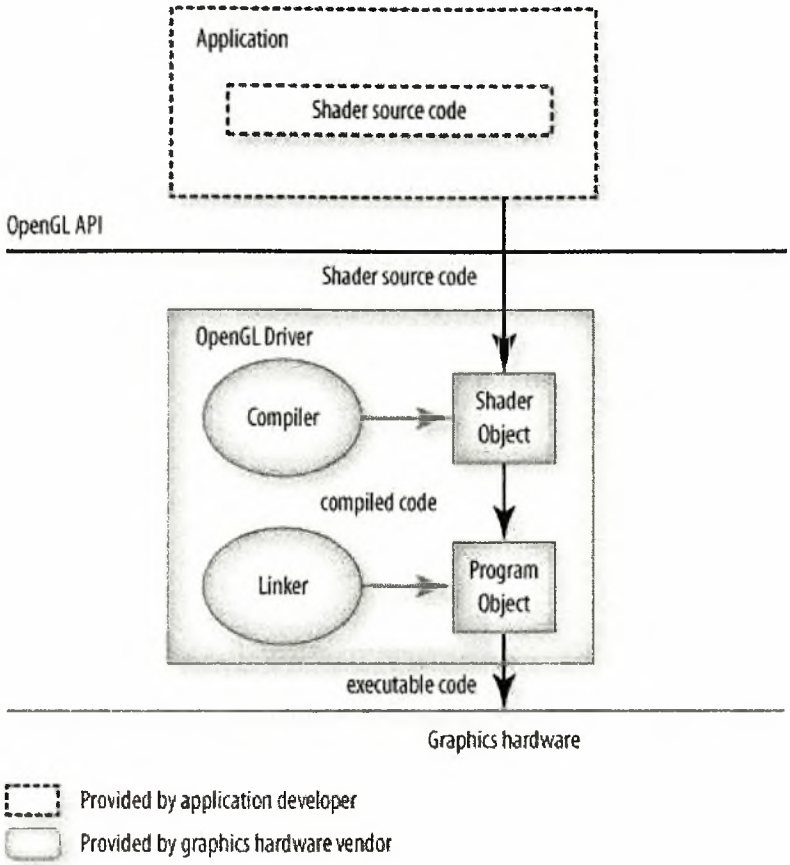
- **Attribute:** Οι μεταβλητές αυτές αναγκάζουν μια εφαρμογή να περνάει συχνά μεταβαλλόμενα δεδομένα στον vertex shader. Η μέγιστη συχνότητα ανανέωσης είναι μία φορά ανά κορυφή. Πέραν από τις attribute μεταβλητές που θα ορίσει ο χρήστης, υπάρχουν και κάποιες built-in όπως είναι οι gl_Vertex και gl_Normal τις οποίες όμως ο χρήστης μπορεί να μετονομάσει. Τα attributes περιορίζονται σε **floating-point scalars** και **floating-point matrices**. Ο ορισμός **integer** και **boolean** attributes δεν είναι δυνατός. Επίσης δεν μπορούν να οριστούν attributes τύπου **struct** και **array**. Τα attributes δεν μπορούν να μεταβληθούν σε έναν shader, ενώ δεν μπορούν να οριστούν attributes στον fragment shader.

- **Uniform:** Τα uniform, όπως και τα attributes, δεν μπορούν να μεταβληθούν μέσα σε shader, ενώ χρησιμοποιούνται για δεδομένα που μεταβάλλονται λιγότερο συχνά. Οι μεταβλητές αυτές μπορούν να τροποποιηθούν το πολύ μία φορά για κάθε primitive (οτιδήποτε μπορεί να σκιαστεί: points, lines, polygons, bitmaps και images). Όλοι οι τύποι δεδομένων, όπως και πίνακες αυτών μπορούν να χρησιμοποιηθούν για uniform μεταβλητές. Όλοι οι vertex και fragment shaders που αποτελούν ένα μοναδικό *"Program"*, μοιράζονται κοινά uniforms.

- **Varying:** Οι μεταβλητές τύπου varying, αποτελούν την μοναδική δίοδο επικοινωνίας των vertex shaders με τους fragment shaders. Η χρησιμότητά τους, βασίζεται στο γεγονός ότι για συγκεκριμένα χαρακτηριστικά κάποιου primitive, κάθε κορυφή μπορεί να έχει διαφορετική τιμή και οι τιμές αυτές χρειάζεται να παρεμβληθούν γραμμικά στα εικονοστοιχεία που βρίσκονται μεταξύ των κορυφών. Ο vertex shader, αποθηκεύει την τιμή κάθε κορυφής σε μία varying μεταβλητή. Όταν ο fragment shader διαβάσει αυτήν την μεταβλητή, θα πάρει την τιμή που του αντιστοιχεί από την γραμμική παρεμβολή. Δεν επιτρέπεται να αποθηκευτεί τιμή σε varying μεταβλητή από τον fragment shader.

2.3 Μοντέλο εκτέλεσης των OpenGL Shaders

Στην εικόνα 2 βλέπουμε τον τρόπο χειρισμού των shaders στο περιβάλλον εκτέλεσης του OpenGL. Οι εφαρμογές επικοινωνούν με το OpenGL, μέσω της κλήσης συναρτήσεων που ανήκουν στο API του. Μια νέα συνάρτηση, η ***glCreateShader***, επιτρέπει στις OpenGL εφαρμογές να δεσμεύσουν τις δομές δεδομένων που είναι αναγκαίες για την αποθήκευση των shaders. Αυτές οι δομές ονομάζονται *«Shader Objects»*. Μετά την δημιουργία ενός Shader Object, η εφαρμογή μπορεί να παρέχει τον πηγαίο κώδικα για τον Shader καλώντας τη συνάρτηση ***glShaderSource***. Αυτή η συνάρτηση παρέχει στο OpenGL τον πηγαίο κώδικα του shader σε μορφή string (συμβολοσειράς).



Εικόνα 3: Διάγραμμα εκτέλεσης των OpenGL Shaders

Όπως φαίνεται από την εικόνα 3 ο μεταγλωττιστής της OGSL, είναι στην πραγματικότητα μέρος του περιβάλλοντος οδηγού του OpenGL. Αυτή είναι και η κύρια διαφορά της OGSL σε σχέση με τις υπόλοιπες shading languages, όπως είναι οι HLSL της Microsoft και η Cg της NVIDIA. Σε αυτές τις γλώσσες ο μεταγλωττιστής της shading language βρίσκεται πάνω από το API των γραφικών και μεταγλωττίζει την υψηλού επιπέδου shading language σε κάτι που μπορεί να χρησιμοποιηθεί από το API των γραφικών που βρίσκεται σε χαμηλότερο επίπεδο.

Η μεταγλώττιση του κώδικα γίνεται μέσω της εντολής ***glCompileShader***, αφού πρώτα αυτός φορτωθεί στον οδηγό του OpenGL. Οι εφαρμογές πρέπει να προσαρτήσουν τα Shader Objects σε δομές δεδομένων που ονομάζονται *«Program Objects»*. Η προσάρτηση γίνεται μέσω της εντολής ***glAttachShader***. Έπειτα από την προσάρτηση των δομών δεδομένων, ο μεταγλωττισμένος shader μπορεί να συνδεθεί με το υπάρχον

OpenGL πρόγραμμα, χρησιμοποιώντας την εντολή ***glLinkProgram***. Μέσω αυτής της εντολής, εντοπίζονται οι εξωτερικές αναφορές μεταξύ των shaders, ελέγχεται η συμβατότητα του fragment με τον vertex shader και ανατίθενται οι περιοχές μνήμης για τις uniform μεταβλητές. Το αποτέλεσμα είναι ένα εκτελέσιμο που μπορεί να εγκατασταθεί με την εντολή ***glUseProgram*** ως ένα μέρος της τρέχουσας κατάστασης του OpenGL.

2.4 Παράδειγμα Χρήσης OpenGL Shader

Ο παρακάτω κώδικας θα μπορούσε να αποτελέσει το μέρος μιας OpenGL εφαρμογής. Παρουσιάζεται η διαδικασία της δέσμευσης των δομών των shaders, η φόρτωση του πηγαίου κώδικα, η μεταγλώττιση του OGSL κώδικα, η προσάρτηση του Program Object, η σύνδεσή του καθώς και η χρήση του. Τέλος, παρουσιάζεται η διαδικασία της παροχής των uniform μεταβλητών στο Program Object.

```
int installTempShaders(const GLchar *tempVertex, const GLchar *tempFragment)
{
        // handles to objects
        GLuint tempVS, tempFS, tempProg;

        // status values
        GLint vertCompiled, fragCompiled, linked;

        // Create a vertex shader object and a fragment shader object
        tempVS = glCreateShader(GL_VERTEX_SHADER);
        tempFS = glCreateShader(GL_FRAGMENT_SHADER);

        // Load source code strings into shaders
        glShaderSource(tempVS, 1, &tempVertex, NULL);
        glShaderSource(tempFS, 1, &tempFragment, NULL);

        // Compile the temp vertex shader, and print out
        // the compiler log file.
        glCompileShader(tempVS);
        glGetShaderiv(tempVS, GL_COMPILE_STATUS, &vertCompiled);
        printShaderInfoLog(tempVS);

        // Compile the temp fragment shader, and print out
        // the compiler log file.
        glCompileShader(tempFS);
        glGetShaderiv(tempFS, GL_COMPILE_STATUS, &fragCompiled);
```

```
	printShaderInfoLog(tempFS);

	//Jandling Error in Compiling Process
	if (!vertCompiled || !fragCompiled)
		return 0;

	// Create a program object and attach the two compiled shaders
	tempProg = glCreateProgram();
	glAttachShader(tempProg, tempVS);
	glAttachShader(tempProg, tempFS);

	// Link the program object and print out the info log
	glLinkProgram(tempProg);
	glGetProgramiv(tempProg, GL_LINK_STATUS, &linked);
	printProgramInfoLog(tempProg);

	//Handling Error in Linking Process
	if (!linked)
		return 0;

	// Install program object as part of current state
	glUseProgram(tempProg);

	// Set up initial uniform values
	glUniform3f(getUniLoc(tempProg, "tempColor"), 1.0, 0.3, 0.2);
	glUniform2f(getUniLoc(tempProg, "tempSize"), 0.30, 0.15);
	glUniform3f(getUniLoc(tempProg, "LightPosition"), 0.0,0.0,4.0);
	return 1;
}
```

Συνάρτηση για την εύρεση της τοποθεσίας των uniform μεταβλητών:

```
GLint getUniLoc(GLuint program, const GLchar *name)
{
	GLint loc;
	loc = glGetUniformLocation(program, name);
	if (loc == -1)
	printf("No such uniform named \"%s\"\n", name);
	printOpenGLError(); // Check for OpenGL errors
	return loc;
}
```

Η παρακάτω συνάρτηση τυπώνει τα μηνύματα πληροφοριών των shaders:

```
void printShaderInfoLog(GLuint shader)
{
	int infologLength = 0;
```

```
    int charsWritten = 0;
    GLchar *infoLog;

    // Check for OpenGL errors
    printOpenGLError();

    glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &infologLength);

    // Check for OpenGL errors
    printOpenGLError();
    if (infologLength > 0)
    {
            infoLog = (GLchar*)malloc(infologLength);
            if (infoLog == NULL)
            {
                    printf("ERROR: Could not allocate InfoLog buffer\n");
                    exit(1);
            }
            glGetShaderInfoLog(shader, infologLength,
            &charsWritten, infoLog);
            printf("InfoLog:\n%s\n\n", infoLog);
            free(infoLog);
    }

    // Check for OpenGL errors
    printOpenGLError();
}
```

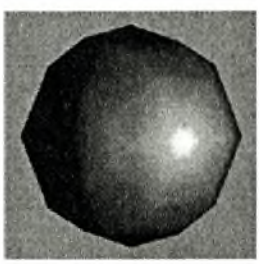
3. Ανάλυση Texture Mapping Τεχνικών

Σε αυτό το Κεφάλαιο παρουσιάζονται οι τεχνικές υφής που μελετήθηκαν καθώς και η υλοποίησή τους σε OpenGL Shading Language.

3.1 Per Pixel Lighting (Phong Lighting Model)

Η τεχνική του per-pixel lighting, δεν αποτελεί μια ξεχωριστή τεχνική Texture Mapping. Ο λόγος της παρουσίασής της, αποτελεί μια απλή επίδειξη των δυνατοτήτων που δίνουν στους προγραμματιστές οι Shading Languages, σε σχέση με την fixed functionality για per-Vertex lighting που παρέχει το OpenGL, αλλά και επειδή αυτό το μοντέλο φωτισμού είναι που χρησιμοποιείται στις υλοποιήσεις.

Η στατική λειτουργικότητα που παρέχει το OpenGL, αποτελείται από το μοντέλο φωτισμού Blinn – Phong, που εφαρμόζεται σε κάθε κορυφή των πολυγωνικών μοντέλων, ενώ οι τιμές όλων των ενδιάμεσων εικονοστοιχείων (Pixels), υπολογίζονται με γραμμική παρεμβολή. Με τα συγκεκριμένα μοντέλα φωτοσκίασης, είναι δυνατόν να προκύψει το οπτικό αποτέλεσμα που επιθυμείται από την εκάστοτε εφαρμογή. Αν όμως ο αριθμός των κορυφών του προς απεικόνιση αντικειμένου είναι μικρός, ο φωτισμός θα είναι «φτωχός». Αυτό θα μπορούσε να διορθωθεί μόνο με την αύξηση του αριθμού των πολυγώνων του αντικειμένου. Ωστόσο αυτή η αύξηση θα είχε ως αποτέλεσμα και την αύξηση της επεξεργαστική ισχύς της εφαρμογής.

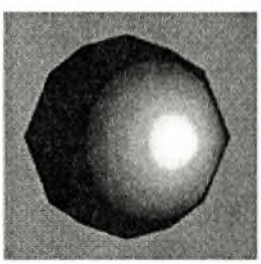


Εικόνα 4: Ένα αντικείμενο με λίγα πολύγωνα, όπου εφαρμόζεται per-vertex φωτισμός.

Η λύση στο παραπάνω πρόβλημα δόθηκε από την NVIDIA και συγκεκριμένα με την GeForce 2 όπου για πρώτη φορά εμφανίστηκε η τεχνική του NVIDIA Shading Rasterizer που επέτρεπε την προσθήκη τεχνικών εφέ

ανά εικονοστοιχείο. Ακριβέστερα, μέσω του Shading Rasterizer, οι παρακάτω μέθοδοι μπορούσαν πλέον να εκτελεστούν και να υποστηριχθούν ανά εικονοστοιχείο:

- Base Texture
- Bump Mapping
- Diffuse Lighting
- Specular Lighting
- Colored Fog
- Ambient Light
- Alpha Transparency



Εικόνα 5: Το ίδιο αντικείμενο με λίγα πολύγωνα όπου εφαρμόζεται per-pixel φωτισμός.

Στην τεχνική αυτή, εφαρμόζεται το μοντέλο του Phong, όπου ο φωτισμός αποτελείται από τρεις συνιστώσες: το φωτισμό περιβάλλοντος που είναι σταθερός (ambient light), το διάχυτο φωτισμό (diffuse light) και τον κατοπτρικό φωτισμό (specular light).

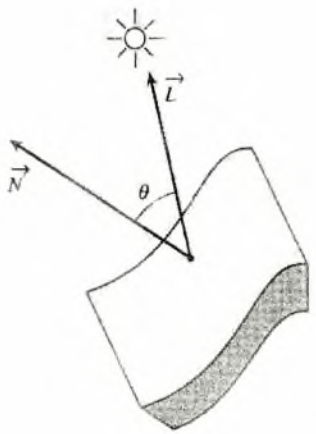
Η διάχυτη συνιστώσα (I_d) είναι η ιδανική διάχυτη συνιστώσα ανάκλασης και για αυτήν ισχύει ο νόμος συνημίτονου του Lambert:

$$I_d = I_i \cdot k_d \cdot \cos\theta \quad \textbf{(Σχέση 1)}$$

όπου I_i είναι η ένταση μιας σημειακής φωτεινής πηγής, $\vec{L}$ η ακτίνα πρόσπτωσης, $\vec{N}$ το κανονικό διάνυσμα της επιφάνειας, θ η γωνία μεταξύ της $\vec{L}$ και του $\vec{N}$ (Εικόνα 6) και k_d ο συντελεστής της διάχυτης ανάκλασης που εξαρτάται από το μήκος κύματος του φωτός και το υλικό της επιφάνειας.

Ξαναγράφουμε τη ***Σχέση 1*** με χρήση των $\vec{L}$, $\vec{N}$ και προσθέτοντας έναν έλεγχο για την εξάλειψη των αρνητικών γωνιών.

$$I_d = k_d \cdot I_i \cdot \max(0, \vec{L} \bullet \vec{N}) \quad \textbf{(Σχέση 2)}$$



Εικόνα 6: $\vec{N}$ και $\vec{L}$

Η ***Σχέση 2*** αποτελεί την μαθηματική φόρμουλα υπολογισμού της διάχυτης ανάκλασης στο μοντέλο φωτισμού του Phong. Η φόρμουλα αυτή μπορεί να γενικευτεί και για περισσότερες της μίας φωτεινής πηγής:

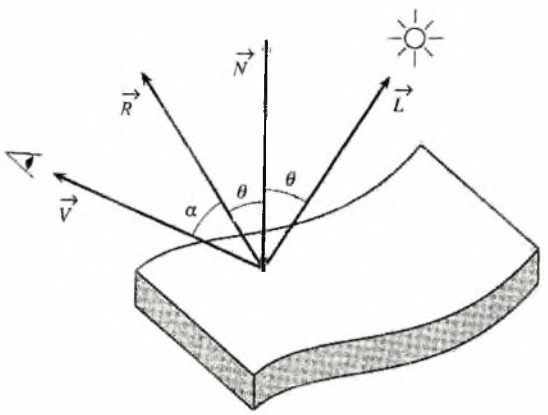
$$I_d = k_d \cdot \sum_j I_{i,j} \cdot \max(0, \vec{L} \bullet \vec{N}) \quad \textbf{(Σχέση 3)}$$

Η κατευθυνόμενη ανάκλαση που είναι η δεύτερη συνιστώσα στο μοντέλο του Phong, ακολουθεί το νόμο του καθρέφτη. Είναι συνάρτηση της γωνίας α μεταξύ της κατευθυνόμενης ανάκλασης $\vec{R}$ και της κατευθυνόμενης παρατήρησης $\vec{V}$ (Εικόνα 7).

$$I_s = I_i \cdot k_s \cdot \cos^n \alpha = I_i \cdot k_s \cdot (\vec{R} \bullet \vec{V})^n \quad \textbf{(Σχέση 4)}$$

όπου τα $\vec{R}$ και $\vec{V}$ είναι μοναδιαία διανύσματα, το n αντιστοιχεί στην αδρότητα της επιφάνειας και k_s είναι ο συντελεστής κατευθυνόμενης ανάκλασης που

κυμαίνεται μεταξύ 0 και 1. Η ***Σχέση 4*** είναι η φόρμουλα υπολογισμού της κατευθυνόμενης ανάκλασης στο μοντέλο του Phong.



Εικόνα 7: $\vec{N}$, $\vec{L}$, $\vec{R}$ **και** $\vec{V}$

Κανονικά, αντί του συντελεστή k_s, θα έπρεπε να ληφθεί υπόψη κάποια συνάρτηση $w(\theta,\lambda)$, όπου θ η γωνία πρόσπτωσης και λ το μήκος κύματος του προσπίπτοντος φωτός. Αυτή η συνάρτηση εξαρτάται από τη συμπεριφορά του γυαλιού. Έτσι για $\theta = 0^o$ δεν έχουμε καμιά ανάκλαση ενώ για $\theta = 90^o$, έχουμε ολική ανάκλαση. Για πολλά υλικά ισχύει $w(\theta,\lambda) = k_s$, όμως η μη εξάρτηση της συμπεριφοράς του μοντέλου από τη γωνία θ καθιστά το μοντέλο του Phong μια μη αμφίδρομη συνάρτηση κάτι που αποτελεί και ένα από τα κυριότερα μειονεκτήματά του.

Μέχρι στιγμής δεν έχει αναφερθεί ο τρόπος υπολογισμού του διανύσματος αντανάκλασης του φωτός, $\vec{R}$. Ο γεωμετρικός του υπολογισμός βασίζεται στην παρατήρηση ότι τα $\vec{L}, \vec{N}$ και $\vec{R}$ είναι συνεπίπεδα και οι γωνίες μεταξύ των ζευγών $(\vec{L},\vec{N})$ και $(\vec{N},\vec{R})$ είναι ίσες. Γίνεται η υπόθεση ότι τα $\vec{N}$ και $\vec{L}$ είναι μοναδιαία. Έστω $\vec{R}_1$ η προβολή του $\vec{L}$ στον άξονα του $\vec{N}$ (Εικόνα 8). Έχουμε:

$$\vec{R}_1 = \vec{N}\cos\theta = \vec{N}\left(\vec{N} \bullet \vec{L}\right) \quad \textbf{(Σχέση 5)}$$

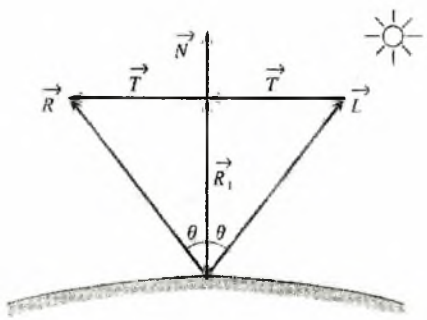
Επίσης ισχύει:

$$\vec{R} = \vec{R}_1 + \vec{T} \text{ και } \vec{T} = \vec{R}_1 - \vec{L}$$

Αντικαθιστώντας στη ***Σχέση 5,*** έχουμε

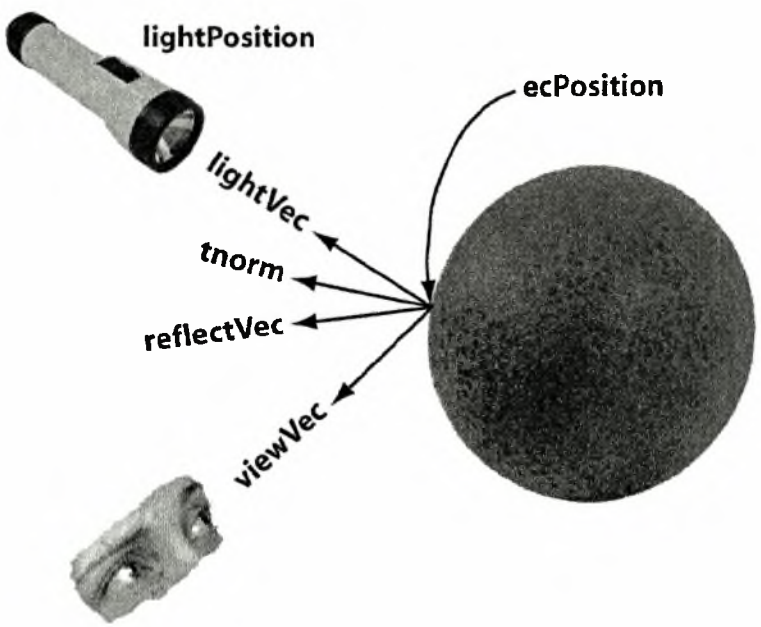
$$\vec{R} = 2\vec{R}_1 - \vec{L} = 2\vec{N}\left(\vec{N} \bullet \vec{L}\right) - \vec{L} \quad \textbf{(Σχέση 6)}$$

Η μέθοδος αυτή απαιτεί 6 πολλαπλασιασμούς και 5 προσθέσεις.



Εικόνα 8: Γεωμετρική μέθοδος υπολογισμού του $\vec{R}$

OGSL Shader



Εικόνα 9: Τα ονόματα των μεταβλητών στους κώδικες που παρουσιάζονται, προέρχονται από αυτό το σχήμα για διευκόλυνση του αναγνώστη

lightPosition: Η θέση της πηγής φωτός

ecPosition: Το σημείο παρατήρησης

lightVec: Το διάνυσμα του φωτός που αντιστοιχεί στο σημείο παρατήρησης

tnorm: Το υπολογιζόμενο κανονικό διάνυσμα της επιφάνειας που αντιστοιχεί στο σημείο παρατήρησης (σε κάποιες από τις μεθόδους που θα παρουσιαστούν, δεν διαφέρει από το αρχικό κανονικό διάνυσμα)

viewVec: Το διάνυσμα παρατήρησης

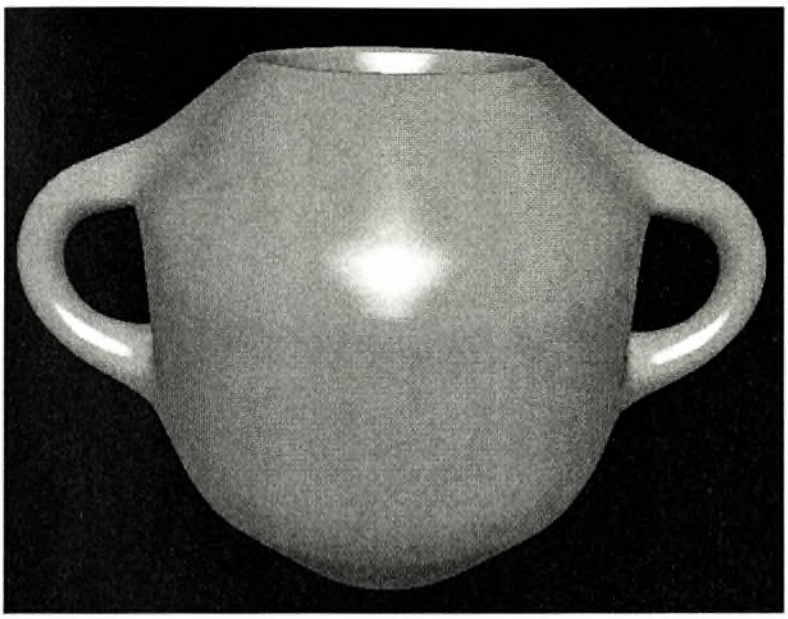
reflectVec: Το διάνυσμα κατευθυνόμενης ανάκλασης

Vertex Shader

anthousis@gmail.com

Fragment (Pixel) Shader

anthousis@gmail.com



Εικόνα 10: Phong Per Pixel Lighting Shader

3.2 Simple Texture Mapping

Αυτή η τεχνική είναι και η πιο απλή μέθοδος υφής με χρήση texture. Αποτελεί καθαρά μία τεχνική image mapping, καθώς πέραν την χρήσης του χάρτη υφής δεν γίνεται καμία διεργασία για την προσέγγιση του ανάγλυφου. Κάθε συνιστώσα της επιφάνειας του αντικειμένου, αντιστοιχίζεται σε μια συνιστώσα του δυσδιάστατου texture και παίρνει την ανάλογη χρωματική τιμή χωρίς κανέναν επιπλέον υπολογισμό. Για τον φωτισμό χρησιμοποιείται το μοντέλο του Phong, του οποίου η περιγραφή έγινε παραπάνω.

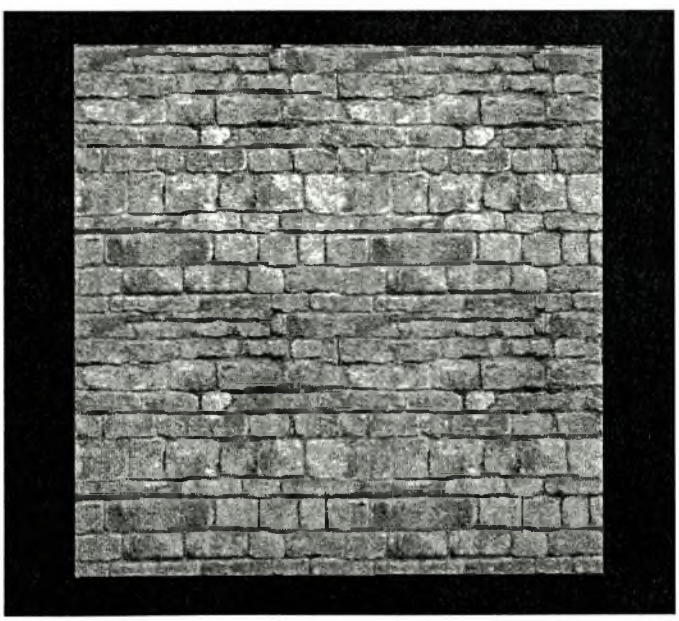
OGSL Shader

Vertex Shader

anthousis@gmail.com

Fragment (Pixel) Shader

anthousis@gmail.com

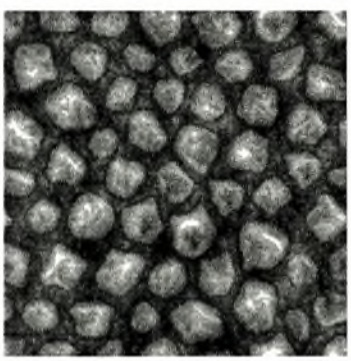


Εικόνα 11: Simple Texture Mapping Shader

3.3 Bump Mapping

Το 1978 στην δημοσίευση «*Simulation of Wrinkled Surfaces*», ο James Blinn παρουσίασε εκείνη την μέθοδο που σήμερα είναι γνωστή ως bump mapping (χαρτογράφηση προσκρούσεων). Η τεχνική αυτή προσομοιώνει τις «προσκρούσεις» (bumps) ή τις «ρυτίδες» (wrinkles) σε μια επιφάνεια χωρίς να πραγματοποιεί γεωμετρικές αλλαγές στο πρωτότυπο αντικείμενο. Το κανονικό διάνυσμα μιας δοθείσας επιφάνειας διαταράσσεται σύμφωνα με ένα χάρτη προσκρούσεων (bump map). Το διαταραγμένο κανονικό διάνυσμα είναι αυτό που χρησιμοποιείται στη συνέχεια για την σκίαση της επιφάνειας, σύμφωνα με την τεχνική του Phong. Το αποτέλεσμα αυτής της μεθόδου είναι η εμφάνιση των προσκρούσεων στην επιφάνεια.

Οι χάρτες ύψους, είναι στην ουσία 8-bit gray-scale εικόνες. Σε αυτές, το άσπρο αντιπροσωπεύει τις ανυψωμένες περιοχές και το μαύρο τις χαμηλές. Δηλαδή έχουμε 256 διαφορετικά επίπεδα ανύψωσης.



Εικόνα 12: Height Map

Η παρατήρηση του Blinn ήταν ότι δεν είναι ανάγκη να διαταράξουμε τις κορυφές του μοντέλου για να υπολογίσουμε τα κανονικά διανύσματα της επιφάνειας. Διαταράσσοντας μόνο τα κανονικά διανύσματα που θα χρησιμοποιηθούν στην σκίαση, κατορθώνουμε να διατηρήσουμε τη γεωμετρία/επιφάνεια του αντικειμένου. Το διαταραγμένο κανονικό διάνυσμα θα χρησιμοποιηθεί στη συνέχεια για τη σκίαση του αντικειμένου. Για να εφαρμοστεί η τεχνική έπρεπε να βρεθεί ένας τρόπος διαταραχής του κανονικού διανύσματος της επιφάνειας, που να είναι ανεξάρτητος από τον προσανατολισμό και την θέση της επιφάνειας. Αυτός ο περιορισμός

προκύπτει από την παρατήρηση ότι εάν το αντικείμενο, στο οποίο είχε εφαρμοστεί η τεχνική, χρησιμοποιηθεί σε animation, δεν θα θέλαμε οι διαταραχές της επιφάνειας να αλλάζουν σχήμα ή εμφάνιση. Αυτός είναι και ο λόγος για τον οποίο η συνάρτηση διαταραχής πρέπει να βασίζεται στα τοπικά παράγωγα της επιφάνειας. Ακολούθως παρατίθεται το μαθηματικό μοντέλο αυτής της τεχνικής.

Έστω $\vec{S}(u,v)$ η διανυσματική έκφραση της επιφάνειας ενός αντικειμένου σε παραμετρική μορφή και $\vec{N}$ το κανονικό διάνυσμα σε μια θέση (u,v). Αν $B(u,v)$ είναι η τιμή του χάρτη ύψους, τότε η νέα θέση $\vec{S}'(u,v)$ της διαταραγμένης επιφάνειας θα είναι:

$$\vec{S}'(u,v) = \vec{S}(u,v) + \vec{B}(u,v) \cdot \vec{N} \quad \textbf{(Σχέση 7)}$$

Κατά την παράσταση της επιφάνειας (scan conversion) $\vec{S}(u,v)$, υπολογίζεται η νέα θέση $\vec{S}'(u,v)$ καθώς και το νέο κανονικό διάνυσμα:

$$\vec{N}' = \frac{\partial \vec{S}'(u,v)}{u} \times \frac{\partial \vec{S}'(u,v)}{v} \quad \textbf{(Σχέση 8)}$$

Από τη ***Σχέση 7***, αν πάρουμε τις μερικές παραγώγους της μετατοπισμένης κατά $B(u,v) \cdot \vec{N}$ επιφάνειας θα έχουμε:

$$\frac{\partial \vec{S}'(u,v)}{u} = \frac{\partial \vec{S}(u,v)}{u} + B_u \vec{N} + B(u,v) \frac{\partial \vec{N}}{\partial u} \quad \textbf{(Σχέση 9)}$$

και

$$\frac{\partial \vec{S}'(u,v)}{v} = \frac{\partial \vec{S}(u,v)}{v} + B_v \vec{N} + B(u,v) \frac{\partial \vec{N}}{\partial v} \quad \textbf{(Σχέση 10)}$$

όπου B_u, B_v είναι οι παράγωγοι κατά u και v του χάρτη ύψους.

Με δεδομένο ότι ο μετασχηματισμός του ανάγλυφου δεν επιφέρει μεγάλες διαταραχές του διανύσματος $\vec{N}$, δηλαδή η τοπική ανύψωση που θέλουμε να επιτύχουμε είναι μικρή σε σχέση με την επιφάνεια στην οποία συντελείται, μπορούμε να θεωρήσουμε τους όρους $B(u,v)\frac{\partial \vec{N}}{\partial u}$ και $B(u,v)\frac{\partial \vec{N}}{\partial v}$ αμελητέους. Έτσι αντικαθιστώντας τις ***Σχέσεις 9*** και ***10*** στη ***Σχέση 8***, προκύπτει ότι το νέο κανονικό διάνυσμα $\vec{N}'$ θα είναι:

$$\vec{N}' = \left(\frac{\partial \vec{S}(u,v)}{u} + B_u \vec{N}\right) \times \left(\frac{\partial \vec{S}(u,v)}{v} + B_v \vec{N}\right) = \frac{\partial \vec{S}(u,v)}{u} \times \frac{\partial \vec{S}(u,v)}{v} + \frac{\partial \vec{S}(u,v)}{u} \times B_v \vec{N}$$

$$+ B_u \vec{N} \times \frac{\partial \vec{S}(u,v)}{v} + B_u B_v \left(\vec{N} \times \vec{N}\right)$$

Όμως,

$$\frac{\partial \vec{S}(u,v)}{u} \times \frac{\partial \vec{S}(u,v)}{v} = \vec{N},\ \vec{N} \times \vec{N} = 0 \text{ και } \frac{\partial \vec{S}(u,v)}{u} \times B_v \vec{N} = -B_v \vec{N} \times \frac{\partial \vec{S}(u,v)}{u}$$

Οπότε:

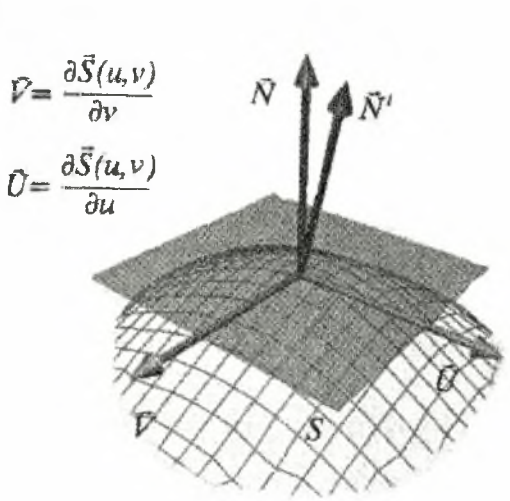
$$\vec{N}' = \vec{N} + B_u \vec{N} \times \frac{\partial \vec{S}(u,v)}{v} - B_v \vec{N} \times \frac{\partial \vec{S}(u,v)}{u}$$

Θεωρώντας μοναδιαία τα διανύσματα $\vec{U} = \frac{\partial \vec{S}(u,v)}{u}$, $\vec{V} = \frac{\partial \vec{S}(u,v)}{v}$, παρατηρούμε ότι το $\vec{N}$ είναι κάθετο στα $\vec{U}$ και $\vec{V}$ (Εικόνα 13). Έτσι έχουμε:

$$\vec{N}' = \vec{N} - B_u \vec{U} - B_v \vec{V} \quad \textbf{(Σχέση 10)}$$

Με τη ***Σχέση 10*** μπορούμε να υπολογίσουμε απευθείας το κανονικό διάνυσμα. Δηλαδή, έχουμε μια συνάρτηση διαταραχής που βασίζεται στα τοπικά παράγωγα της επιφάνειας.

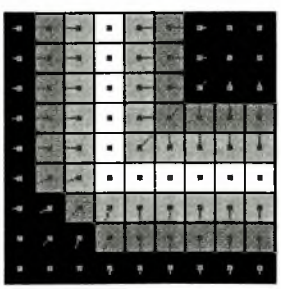
Η μέθοδος αυτή έχει πολύ μεγάλο υπολογιστικό πλεονέκτημα σε σχέση με την μέθοδο του Displacement Mapping (παρουσιάζεται παρακάτω), όμως επειδή δεν μετατοπίζει τα σημεία της επιφάνειας αλλά στρέφει μόνο το κανονικό διάνυσμα στα όρια των αντικειμένων, η επιφάνεια φαίνεται επίπεδη και το οπτικό αποτέλεσμα δεν είναι ικανοποιητικό.



Εικόνα 13: Εξάρτηση του κανονικού διανύσματος $\vec{N}$ από τις μερικές παραγώγους της επιφάνειας ως προς τις παραμέτρους u και v.

Το μόνο που απομένει είναι να αναφέρουμε τον τρόπο μετατροπής της πληροφορίας του heightMap σε διάνυσμα, έτσι ώστε να μεταβληθεί το κανονικό διάνυσμα.

Τα φωτεινά εικονοστοιχεία του πίνακα ύψους, αντιπροσωπεύουν μεγαλύτερο ύψος από τα σκοτεινά. Με βάση αυτήν την πληροφορία μπορούμε να σχεδιάσουμε για κάθε εικονοστοιχείο έναν πίνακα. Αυτοί οι πίνακες αντιπροσωπεύουν την κλίση της επιφάνειας σε εκείνο το σημείο, όπως φαίνεται και στην παρακάτω εικόνα.



Εικόνα 14: Κλίση εικονοστοιχείων

Ένας τρόπος για να υπολογιστούν τα παραπάνω διανύσματα είναι να υπολογιστεί για κάθε pixel η κλίση στους άξονες X και Y.

$$x\_gradient = pixel(x-1, y) - pixel(x+1, y)$$
$$y\_gradient = pixel(x, y-1) - pixel(x, y+1)$$

Με αυτές τις κλίσεις μπορούμε να ρυθμίσουμε το κανονικό διάνυσμα της επιφάνειας σε κάθε σημείο:

$$\vec{N}' = \vec{N} - x\_gradient \cdot \vec{U} - y\_gradient \cdot \vec{V}$$

OGSL Shader

Vertex Shader

anthousis@gmail.com

Fragment (Pixel) Shader

anthousis@gmail.com

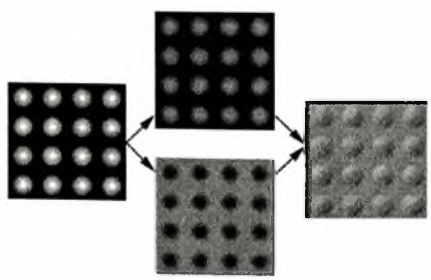
Εικόνα 15: Bump Mapping Shader

3.4 Emboss Bump Mapping

Η τεχνική του Bump Mapping χρησιμοποιεί per-pixel φωτισμό και κατά συνέπεια απαιτεί υπολογισμούς για κάθε εικονοστοιχείο, εξαιτίας της τροποποίησης του κανονικού διανύσματος. Ωστόσο αυτοί οι υπολογισμοί ήταν απαγορευτικοί για τις παλιές κάρτες γραφικών.

Η τεχνική του Emboss Bump Mapping, ξεκινάει από έναν πίνακα υψών όπως και το Bump Mapping, αλλά οι ομοιότητες των δύο τεχνικών σταματούν εκεί. Στην τεχνική αυτή, αντί για να προσπαθούμε να υπολογίσουμε ένα διαταραγμένο κανονικό διάνυσμα για per-pixel φωτισμό, υπολογίζουμε απευθείας την ένταση φωτισμού. Εξαιτίας αυτού βέβαια και σε αντίθεση με το Bump Mapping στο Emboss Bump Mapping, δεν μπορούμε να χρησιμοποιήσουμε όποιο μοντέλο φωτισμού θέλουμε.

Το πρώτο βήμα της τεχνικής αυτής είναι η επεξεργασία του χάρτη υψών και η παραγωγή δύο ξεχωριστών χαρτών. Ο πρώτος από τους δύο χάρτες αποτελείται από τη μισή φωτεινότητα του αρχικού χάρτη υψών. Ο δεύτερος χάρτης αποτελείται από τη μισή φωτεινότητα του αντίστροφου του χάρτη υψών. Η διαδικασία της φωτοσκίασης, γίνεται σε τρία στάδια (τα στάδια αυτά θα μπορούσαν να συμπτυχθούν με χρήση multitexturing). Το πρώτο στάδιο χρησιμοποιεί τον πρώτο από τους δύο χάρτες που παράχθηκαν, το δεύτερο στάδιο χρησιμοποιεί το δεύτερο χάρτη, όπου όμως οι συντεταγμένες υφής έχουν μετατοπιστεί κατά ένα μικρό διάστημα προς την πηγή του φωτός. Τέλος, στο τρίτο βήμα γίνεται προσαρμογή των αποτελεσμάτων με per-vertex φωτισμό και αν θελήσουμε και με κάποιο texture υφής. Στην παρακάτω εικόνα φαίνονται τα πρώτα δύο βήματα της διαδικασίας.



Εικόνα 16: Emboss Bump Mapping

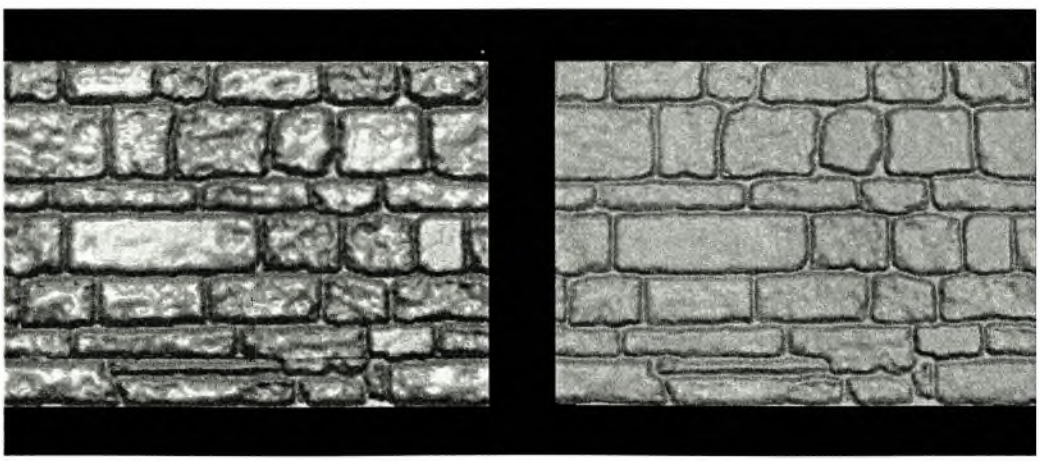
Επειδή δεν γίνεται κάποιος μετασχηματισμός του κανονικού διανύσματος, το Emboss Bump Mapping, μπορεί να χρησιμοποιηθεί μόνο με διάχυτο φωτισμό. Συνεπώς, εξαιτίας του per-vertex φωτισμού, το οπτικό αποτέλεσμα στερείται λεπτομέρειας και παρουσιάζεται θολό.

Το κύριο πλεονέκτημα της τεχνικής αυτής πηγάζει από τους περιορισμούς της. Για την εφαρμογή της δεν χρειάζεται υποστήριξη από την πλευρά του υλικού και μπορεί εύκολα να υλοποιηθεί και σε υλικό που δεν υποστηρίζει Multitexturing.

3.5 Normal Mapping

Η τεχνική του Normal Mapping που προτάθηκε από τον Peercy το 1997 στη δημοσίευση *«Efficient Bump Mapping Hardware»* (γνωστή και ως Dot3 bump mapping), αποτελεί στην ουσία μια βελτίωση της κλασικής Bump Mapping τεχνικής, καθώς και την πιο γνωστή τεχνική για bump mapping πραγματικού χρόνου. Η βασική διαφορά αυτή της τεχνικής, είναι ότι ενώ στο Bump Mapping έχουμε διαταραχή του υπάρχοντος κανονικού διανύσματος της επιφάνειας, στο Normal Mapping, έχουμε ολική αντικατάσταση του διανύσματος με προϋπολογισμένα κανονικά διανύσματα. Αυτό επιτυγχάνεται με αντικατάσταση των 8-bit gray-scale χαρτών ύψους, που χρησίμευαν για την διαταραχή των κανονικών διανυσμάτων, με 24-bit χάρτες που περιέχουν αναπαραστάσεις των κανονικών διανυσμάτων (Normal Maps) για κάθε σημείο της επιφάνειας.

Οι χάρτες ύψους μπορούν να αναπαραστήσουν περιορισμένες επιφάνειες. Συχνά, επιφάνειες με υψηλή ανύψωση δεν αναπαρίστανται με ακρίβεια, γεγονός που περιορίζει το επίπεδο της φαινομενικής ανύψωσης. Οι περιορισμοί που επιβάλλονται εξαιτίας των 256 διαφορετικών επιπέδων, μπορούν να οδηγήσουν σε κοκκώδη εμφάνιση επιφανειών, ενώ οι ομαλές επιφάνειες και οι απότομες γωνίες δεν αναπαρίστανται πιστά.



Εικόνα 17: Διαφορές στην λεπτομέρεια απεικόνισης μεταξύ Normal Mapping (αριστερά) και Bump Mapping (δεξιά)

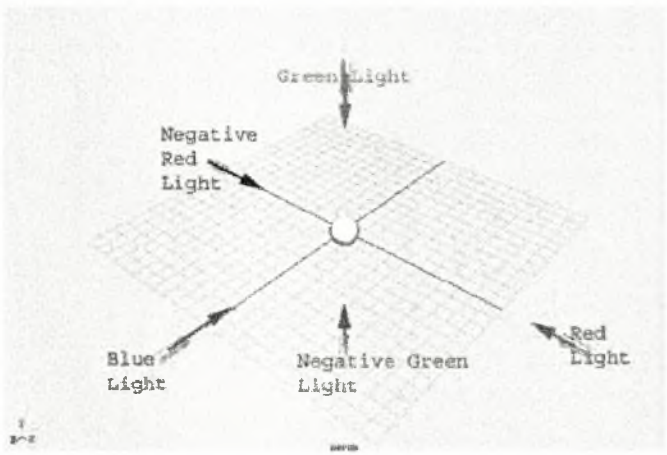
Σε αντίθεση, οι χάρτες κανονικών διανυσμάτων, περιέχουν αναπαραστάσεις των κανονικών διανυσμάτων. Όπως τα κανονικά διανύσματα χρησιμοποιούνται για τον υπολογισμό του φωτισμού και των ανακλάσεων πάνω στην γεωμετρία του αντικειμένου, τα Normal Maps μπορούν να χρησιμοποιηθούν για τον προσδιορισμό της γωνίας με την οποία πέφτει το φως στις διάφορες επιφάνειες ή της γωνίας που χρησιμοποιείται για τους υπολογισμούς της ανάκλασης. Για την αναπαράσταση των κανονικών διανυσμάτων, γίνεται χρήση εικόνων 24-bit, όπου τα διαφορετικά χρώματα αντιπροσωπεύουν διαφορετικές κατευθύνσεις.

Συγκεκριμένα, η αναλογία του RGB μοντέλου με τα κανονικά διανύσματα είναι η εξής: το κόκκινο χρώμα αναπαριστά τον άξονα x, το πράσινο αναπαριστά τον άξονα y και το μπλε αναπαριστά τον άξονα z.

Εικόνα 18: Normal Map

Υπάρχουν διάφοροι τρόποι για την δημιουργία Normal Maps. Στην παρούσα εργασία, χρησιμοποιήθηκε το Plugin της NVIDIA για το Adobe Photoshop, που παρήγαγε normal maps από τα texture υφής, δίνοντας στον χρήστη αρκετές παραμέτρους για πειραματισμό. Ένας άλλος τρόπος, είναι να παραχθεί ένας πίνακας κανονικών διανυσμάτων από ένα 3D μοντέλο με υψηλή λεπτομέρεια μέσω κάποιου rendering λογισμικού, όπως τα 3D Max, Maya και Lightwave. Η διαδικασία, περιλαμβάνει την τοποθέτηση μιας κόκκινης πηγής φωτός στα δεξιά του αντικειμένου, μιας πράσινης από πάνω του και την τοποθέτηση αρνητικών πηγών κόκκινου και πράσινου φωτός στις αντίθετες κατευθύνσεις αντίστοιχα. Επιπλέον, πρέπει να τοποθετηθεί μια πηγή μπλε φωτός μπροστά (στον άξονα του z) από το αντικείμενο. Αρνητικό

μπλε φως δεν χρειάζεται, καθώς η εικόνα δημιουργείται από τον άξονα των z, οπότε δεν είναι ορατή η πίσω πλευρά του αντικειμένου.



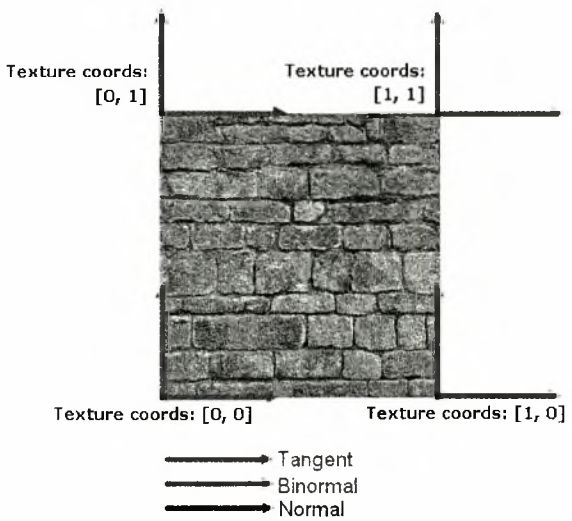
Εικόνα 19: Μορφοποίηση φωτισμού για την δημιουργία Normal Map

Εκτός του προαναφερθέντος τρόπου, υπάρχουν και αυτοματοποιημένα εργαλεία, τα οποία δέχονται ως είσοδο τα αντικείμενα υψηλής και χαμηλής ανάλυσης και παράγουν στην έξοδο το Normal Map. Ένα τέτοιο εργαλείο είναι το Normal Mapper της ATI. Το εργαλείο αυτό, προβάλλει μια ακτίνα φωτός από κάθε εικονοστοιχείο στο μοντέλο χαμηλής ανάλυσης, βρίσκει το σημείο τομής αυτής της ακτίνας με το μοντέλο υψηλής ανάλυσης και αποθηκεύει σε ένα texture το κανονικό διάνυσμα του υψηλής ανάλυσης μοντέλου για το σημείο τομής. Ένα παρόμοιο εργαλείο είναι και το Melody της NVIDIA.

Σε αυτό το σημείο πρέπει να γίνει μια παρατήρηση σχετικά με τα δεδομένα που μας δίνουν τα Normal Maps. Στη γενική του μορφή ένα κανονικό διάνυσμα ορίζεται από 3 διαστάσεις $[x, y, z]$ οι οποίες έχουν εύρος $[-1,1]$. Σε αντίθεση, τα Normal Maps, όπως έχουμε αναφέρει, είναι σε RGB format, κάτι που συνεπάγεται εύρος $[0,1]$. Έγκειται στον προγραμματιστή να μετατρέψει τις τιμές στο ζητούμενο εύρος. Η διαδικασία αυτή αναφέρεται ως *«αποσυμπίεση»* των χρωματικών τιμών του πίνακα κανονικών διανυσμάτων. Η εξίσωση μετατροπής από το εύρος $[0,1]$ στο $[-1,1]$, είναι η εξής:

$$x = 2 * (colorValue - 0.5)$$

Τέλος, είναι ιδιαίτερα σημαντικό τα δεδομένα του χάρτη κανονικών διανυσμάτων να είναι εκφρασμένα σύμφωνα με τα τοπικά παράγωγα της επιφάνειας. Αυτό το σύστημα συντεταγμένων, στα γραφικά υπολογιστών ονομάζεται Tangent Space και τα τρία διανύσματα που το ορίζουν ονομάζονται Tangent, Binormal και Normal. Στην ουσία τα Tangent και Binormal αποτελούν τις u και v παραμέτρους της διανυσματικής έκφρασης της επιφάνειας ενός αντικειμένου $\vec{S}(u,v)$ (Βλέπε Bump Mapping). Πρέπει συνεπώς να δημιουργηθεί ένας TBN πίνακας, μέσω του οποίου θα μετατραπούν όλες οι συνιστώσες του προβλήματός μας στο σύστημα συντεταγμένων της επιφάνειας.



Εικόνα 20: Σύστημα Συντεταγμένων Επιφάνειας (Tangent Coordinate System)

Φυσικά θα μπορούσε να γίνει και η αντίστροφη διαδικασία, η οποία όμως είναι ασύμφορη υπολογιστικά. Π.χ. για το διάχυτο φωτισμό, που υπολογίζεται ως το εσωτερικό γινόμενο του κανονικού διανύσματος και του διανύσματος του φωτός, θα πρέπει και οι δύο πίνακες να είναι στο ίδιο σύστημα συντεταγμένων. Μετατρέποντας το διάνυσμα του φωτός για κάθε κορυφή, τα διανύσματα φωτός μεταξύ δύο κορυφών υπολογίζονται με γραμμική παρεμβολή. Στην αντίστροφη διαδικασία, θα έπρεπε να μετατρέψουμε όλα τα κανονικά διανύσματα στο σύστημα συντεταγμένων που βρίσκεται το διάνυσμα του φωτός.

OGSL Shader

Vertex Shader

anthousis@gmail.com

Fragment (Pixel) Shader

anthousis@gmail.com

Εικόνα 21: Normal Mapping Shader

3.6 Displacement Mapping

Το Displacement Mapping αποτελεί μια ισχυρή τεχνική προσθήκης λεπτομερειών σε τρισδιάστατα μοντέλα και σκηνικά κατά τη διάρκεια του Rendering. Την στιγμή που το Bump Mapping, δίνει την ψευδαίσθηση υψηλότερης λεπτομέρειας στα αντικείμενα που εφαρμόζεται, το Displacement Mapping, αυξάνει στην πραγματικότητα την λεπτομέρεια της επιφάνειας, δίνοντας κατά αυτόν τον τρόπο σωστά γεωμετρικά μοντέλα. Επιπλέον δεν παρουσιάζει τα σφάλματα που παρουσιάζει το Bump Mapping σε μεγάλες οπτικές γωνίες. Ένα από τα μεγαλύτερα πλεονεκτήματα της τεχνικής Displacement Mapping είναι η δυνατότητα της χρήσης της τόσο για την προσθήκη λεπτομέρειας σε ένα αντικείμενο, όσο και για την δημιουργία του ίδιου του μοντέλου. Για την εφαρμογή της τεχνικής, αναγκαία θεωρείται η ύπαρξη ενός χάρτη ύψους σύμφωνα με τον οποίον θα γίνουν οι τροποποιήσεις στην αρχική γεωμετρία του αντικειμένου.

3.6.1 Vertex Displacement Mapping

Το Vertex Displacement Mapping αποτελεί μια από τις εφαρμογές του Displacement Mapping που βασίζεται στην μετακίνηση των κορυφών της αρχικής γεωμετρίας σύμφωνα με το χάρτη υψών. Στη συνέχεια, κάθε pixel μετατοπίζεται μέσω γραμμικής παρεμβολής. Για να έχουμε ικανοποιητικό οπτικό αποτέλεσμα, αν η εικόνα που επιλέξουμε ως υφή είναι ανάλυσης 256x256 εικονοστοιχείων, θα πρέπει η περιοχή της εφαρμογής της να είναι υποδιαιρεμένη σε τουλάχιστον 65.536 μικρότερα πολύγωνα κάτι που συνεπάγεται υψηλές υπολογιστικές απαιτήσεις. Αυτός είναι και ο λόγος που αυτή η μέθοδος δεν χρησιμοποιείται πολύ στις μέρες μας. Ακολούθως παρατίθεται το βασικό μαθηματικό μοντέλο της τεχνικής.

Όπως αναφέρθηκε προηγουμένως, αν $\vec{S}(u,v)$ η διανυσματική έκφραση της επιφάνειας ενός αντικειμένου σε παραμετρική μορφή, $\vec{N}$ το κανονικό διάνυσμα σε μια θέση (u,v) και $B(u,v)$ η τιμή του χάρτη ύψους, τότε η νέα θέση $\vec{S}'(u,v)$ της διαταραγμένης επιφάνειας θα είναι:

$$\vec{S}'(u,v) = \vec{S}(u,v) + \vec{B}(u,v) \cdot \vec{N}$$

Κατά την παράσταση της επιφάνειας (scan conversion) $\vec{S}(u,v)$, υπολογίζεται η νέα θέση $\vec{S}'(u,v)$ καθώς και το νέο κανονικό διάνυσμα:

$$\vec{N}' = \frac{\partial \vec{S}'(u,v)}{u} \times \frac{\partial \vec{S}'(u,v)}{v}$$

Ένα από τα μεγαλύτερα μειονεκτήματα της μεθόδου αυτής είναι ότι τα κανονικά διανύσματα της επιφάνειας, όπως και εκείνα των κορυφών, είναι λάθος, δεδομένου ότι ισχύουν για το αρχικό γεωμετρικό σχήμα. Επειδή προγραμματιστικά είμαστε μέσα στον vertex shader, δεν μπορούμε να έχουμε πρόσβαση στις συνιστώσες των κάθετων στην διαταραγμένη επιφάνεια διανυσμάτων που διαμορφώνουν την τρέχουσα επιφάνεια που επεξεργαζόμαστε και κατά συνέπεια δεν μπορούμε να υπολογίσουμε εκ νέου το κανονικό διάνυσμα.

Δύο είναι οι λύσεις στο παραπάνω πρόβλημα. Η πρώτη είναι η μη χρήση normals, η οποία μπορεί να επιτευχθεί μόνο με στατικό φωτισμό. Η δεύτερη λύση είναι να παρέχουμε τα κανονικά διανύσματα στον pixel shader μέσω ενός χάρτη κανονικών διανυσμάτων. Μια απλή μέθοδος για την επίτευξη αυτής, είναι η δημιουργία ενός χάρτη κανονικών διανυσμάτων (για Normal Mapping βλέπε ενότητα 2.5). Αυτή η τεχνική λειτουργεί αρκετά καλά εκτός κάποιων μικρών ατελειών, όπως η παρουσία φωτισμού σε κάποια σημεία που δεν θα έπρεπε (Normal Vertex Displacement Mapping).

Τέλος αξίζει να σημειωθεί, ότι για αυτήν την μέθοδο, απαιτείται ύπαρξη Shader Model 3 που μεταφράζεται σε μια NVIDIA GeForce 6xxx (ή νεότερη) ή αντίστοιχα μιας εκ των ATI X1800, X1900 (ή νεότερη). Ο ακριβής λόγος για την ύπαρξη αυτών, είναι ότι χρειαζόμαστε πρόσβαση σε τουλάχιστον ένα texture από τον Vertex Shader για την εφαρμογή της μεθόδου. Κάτι τέτοιο δεν ήταν δυνατόν μέχρι την εμφάνιση της τεχνολογίας Shader Model 3 που δίνει πρόσβαση σε τουλάχιστον 4 textures σε επίπεδο Vertex Shader (Vertex Texture Fetching).

OGSL Shader

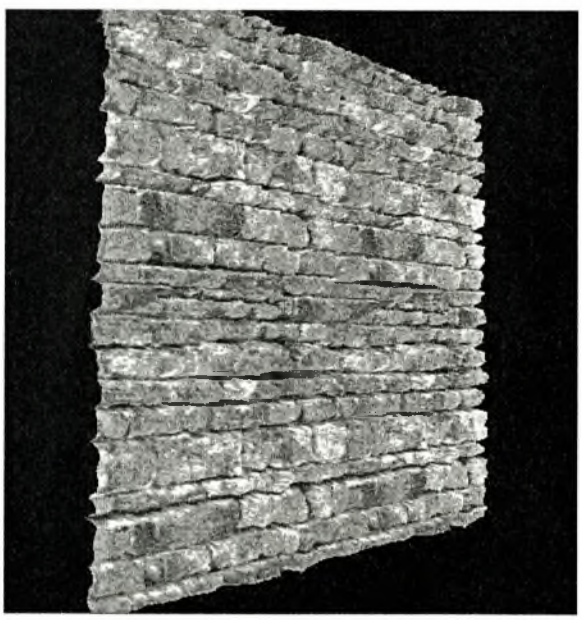
Vertex Displacement

Vertex Shader

anthousis@gmail.com

Fragment Shader

anthousis@gmail.com



Εικόνα 22: Vertex Displacement Mapping

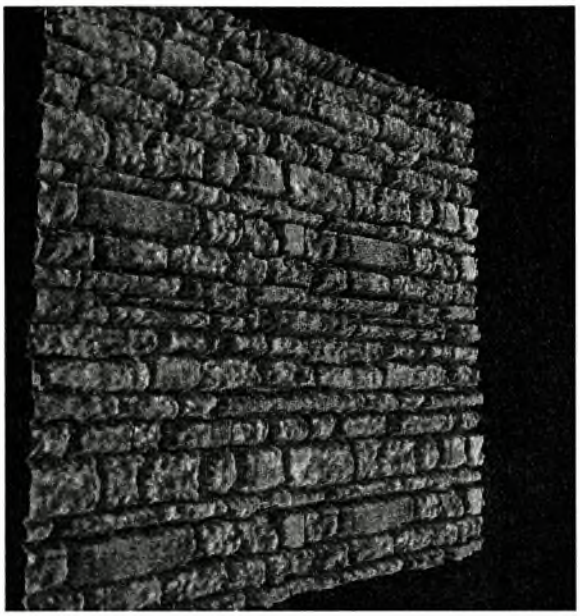
Normal Vertex Displacement

Vertex Shader

anthousis@gmail.com

Fragment Shader

anthousis@gmail.com



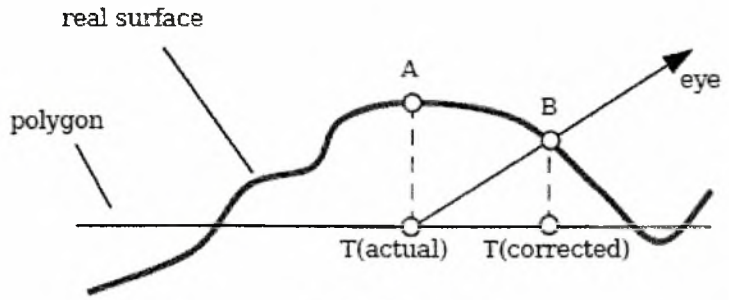
Εικόνα 23: Normal Vertex Displacement Mapping

3.7 Parallax Mapping

Η τεχνική του Parallax Mapping (γνωστή και ως Photonic Mapping, Offset mapping και Virtual Displacement Mapping) που προτάθηκε στη δημοσίευση *«Detailed Shape Representation with Parallax Mapping»* το 2001, αποτελεί μια βελτίωση των σχετικά απλών τεχνικών Bump Mapping (με τον όρο Bump Mapping εδώ συμπεριλαμβάνονται και οι τεχνικές που αποτελούν παραλλαγές της μεθόδου του Blinn).

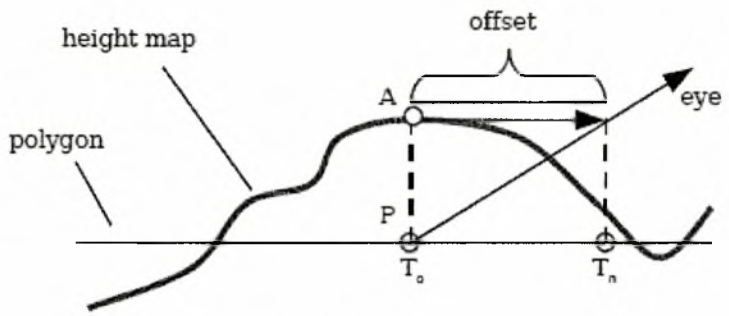
Το μειονέκτημα αυτών των τεχνικών είναι ότι αδυνατούν να αναπαραστήσουν το «motion parallax» φαινόμενο. Αυτός είναι και ο λόγος που οι επιφάνειες των τεχνικών αυτών φαίνονται επίπεδες υπό συγκεκριμένες οπτικές γωνίες. Σε αντίθεση, η τεχνική του Parallax Mapping, βασίζεται στην ιδέα της μετατόπισης των συντεταγμένων της υφής έτσι ώστε αυτές να συμφωνούν με ένα χάρτη υψών και το οπτικό αποτέλεσμα να μιμείται την μετακίνηση των πραγματικών γεωμετρικών σημείων (Displacement Mapping). Στην ουσία, αυτό που κάνει η τεχνική αυτή, είναι να προσπαθεί μέσω παραμόρφωσης της υφής να αναπαραστήσει την οπτική γωνία του ματιού μας με ένα αντικείμενο. Ακολουθεί η ανάλυση της τεχνικής.

Εφαρμόζοντας μια εικόνα υφής που αναπαριστά μια μη λεία επιφάνεια σε ένα επίπεδο πολύγωνο, το αποτέλεσμα που παίρνουμε είναι μια επιφάνεια χωρίς βάθος. Η Εικόνα 24 μας βοηθάει να καταλάβουμε γιατί συμβαίνει αυτό. Κοιτάζοντας το πολύγωνο υπό γωνία το διάνυσμα του ματιού καταλήγει στο σημείο A. Σε αντίθεση, αν κοιτούσαμε την πραγματική επιφάνεια που αντιπροσωπεύει η υφή μας, θα βλέπαμε το σημείο B. Αν μπορούσαμε να διορθώσουμε τις συντεταγμένες της υφής που αντιστοιχούν στο σημείο που βρισκόμαστε θα βλέπαμε το σωστό σημείο. Η διαδικασία του parallax mapping, για κάθε εικονοστοιχείο που απεικονίζεται, διορθώνει τις συντεταγμένες υφής που του αντιστοιχούν, ώστε να προσομοιώσει την κίνηση του ματιού σε σχέση με την μη ομαλή επιφάνεια.



Εικόνα 24

Τρία είναι τα στοιχεία που χρειαζόμαστε για τον υπολογισμό της μετατοπισμένης συντεταγμένης υφής για κάθε εικονοστοιχείο: 1. την αρχική συντεταγμένη, 2. μια τιμή ύψους της επιφάνειας που προσπαθούμε να προσομοιώσουμε και 3. ένα διάνυσμα από το εικονοστοιχείο που απεικονίζουμε προς το μάτι σε tangent space (σύστημα συντεταγμένων επιφάνειας). Στην Εικόνα 25 φαίνεται ο τρόπος που μπορεί να προσαρμοστεί το διάνυσμα του ματιού μέσω του πίνακα υψών, ώστε να πάρουμε μια μετατόπιση για τις συντεταγμένες της υφής.



Εικόνα 25

Ο πίνακας υψών, όπως έχει αναφερθεί και σε προηγούμενη ενότητα, περιέχει μια τιμή ύψους για κάθε εικονοστοιχείο εύρους 0 έως 1. Οι αρχικές συντεταγμένες, όπως και οι τιμές ύψους, είναι αποθηκευμένες σε tangent space και κατά συνέπεια το διάνυσμα από το εικονοστοιχείο προς το μάτι μας πρέπει να μετατραπεί στο tangent space. Οπότε, όπως και στο Normal Mapping, πρέπει να δημιουργηθεί ένας πίνακας μετατροπής διανυσμάτων σε σύστημα συντεταγμένων επιφάνειας.

Αφού διαβαστεί το ύψος για το σημείο T_o (Εικόνα 25) από το Height Map, κλιμακώνεται κατά ένα παράγοντα s και πολώνεται κατά μια τιμή b, έτσι ώστε να μετατραπεί από το εύρος $[0,1]$ σε ένα εύρος που προσομοιώνει τις φυσικές διαστάσεις της επιφάνειας που αναπαρίσταται. Για παράδειγμα, μια υφή τούβλινου τοίχου πού καλύπτει μια εικονική περιοχή 2x2 μέτρα και πάχους 0,02 μέτρων, θα έπρεπε να έχει παράγοντα κλιμάκωσης $s = {}^{0.02}\!/_{2} = 0.01$. Πολώνοντας με $b = 0$, θα είχαμε μετατροπή του εύρους σε $[0.0, 0.01]$, ενώ με πόλωση $b = -0.01$ θα είχαμε εύρος $[0.01, 0.0]$. Η μέση τιμή αυτών των δύο αποτελεί την πιο σωστή για τις περισσότερες υφές: $b = -0.5 \cdot s$. Μια τιμή του b που θα εξαιρούσε το 0.0 από το εύρος, δεν θα ήταν πρακτική, καθώς θα σήμαινε ότι η επιφάνεια που αναπαρίσταται δεν έχει σημείο επαφής με το πολύγωνο στο οποίο εφαρμόζεται. Το νέο ύψος δίνεται από τη σχέση $h_{sb} = h \cdot s + b$. Έπειτα υπολογίζεται η μετατόπιση (offset) καταγράφοντας ένα παράλληλο στο πολύγωνο διάνυσμα, από το σημείο A (σημείο ακριβώς πάνω από το P, Εικόνα 25) προς το διάνυσμα του ματιού. Το νέο αυτό διάνυσμα είναι η μετατόπιση και προστίθεται στο T_o (αρχικές συντεταγμένες υφής) για να παραχθούν οι νέες συντεταγμένες.

$$T_n = T_0 + (h_{sb} \cdot V(x,y) / V(z)) \text{, } \textbf{\textit{(Σχέση 12)}}$$

όπου V το διάνυσμα από το εικονοστοιχείο προς το μάτι.

OGSL Shader

Vertex Shader

anthousis@gmail.com

Fragment Shader

anthousis@gmail.com

Εικόνα 26: Parallax Mapping Shader

3.8 Relief Mapping

Η τεχνική του Relief Mapping, παρουσιάστηκε αρχικά το 2000 από τον Manuel Oliveira στην δημοσίευση *«Relief Texture Mapping»*, αλλά πήρε την τελική της μορφή έπειτα από ένα χρόνο (2001) στη δημοσίευση των Fabio Policarpo και Manuel Oliveira *"Real-Time Relief Mapping on Arbitrary Polygonal Surfaces"*.

Η τεχνική αυτή μοιάζει αρκετά με την τεχνική του Parallax Mapping δεδομένου ότι και οι δύο διορθώνουν τις συντεταγμένες υφής σύμφωνα με την οπτική γωνία. Η τεχνική του Parallax Mapping, δίνει μια προσεγγιστική λύση στο πρόβλημα και παρουσιάζει πολύ καλά οπτικά αποτελέσματα με πολύ μικρό κόστος, στην πραγματικότητα όμως μπορεί να χρησιμοποιηθεί μόνο για θορυβώδεις μη κανονικές προσκρούσεις, καθώς οι επιφάνειες δεν είναι ακριβείς αναπαραστάσεις της πραγματικότητας και παραμορφώνονται όταν αλλάζει η οπτική γωνία. Επιπλέον, δεν υποστηρίζει τον υπολογισμό σκιών. Σε αντίθεση η τεχνική του Relief Mapping, παρουσιάζει ορθά οπτικά αποτελέσματα (όχι προσεγγιστικά), ενώ δίνει την δυνατότητα υπολογισμού σκιών για πρώτη φορά, χωρίς να είναι αναγκαία η χρήση κάποιου shadow map.

Οι δυνατότητες αυτές του Relief Mapping, οφείλονται στην προσθήκη ενός Ray Tracer μέσα στον Pixel Shader, μέσω του οποίου μπορούν να γίνουν οι υπολογισμοί για το ακριβές εικονοστοιχείο απεικόνισης καθώς και για την προσθήκη σκιών. Και σε αυτήν την τεχνική γίνεται χρήση ενός χάρτη υψών και ενός χάρτη κανονικών διανυσμάτων.

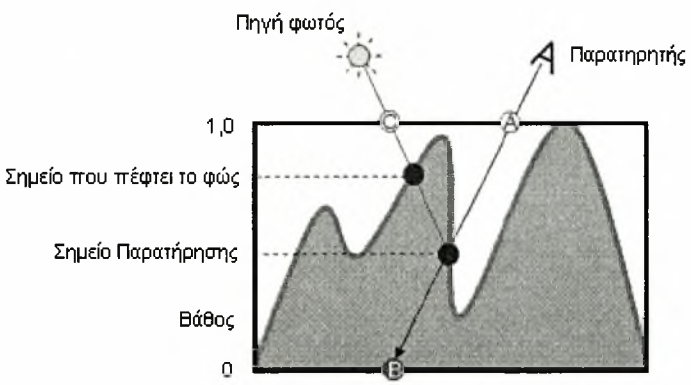
Η διαδικασία του Relief Mapping, μπορεί να θεωρηθεί ότι αποτελείται από τα ακόλουθα βήματα για κάθε εικονοστοιχείο προς απεικόνιση:

- Υπολογισμός του διανύσματος παρατήρησης viewVec (διάνυσμα από το σημείο παρατήρησης προς το σημείο της επιφάνειας που κοιτάμε)
- Μετασχηματισμός του viewVec στο σύστημα συντεταγμένων της επιφάνειας
- Με δεδομένα το νέο διάνυσμα παρατήρησης viewVec', τις συντεταγμένες υφής που αντιστοιχούν στο εικονοστοιχείο που βρισκόμαστε καθώς και

τον πίνακα υψών, υπολογίζουμε μέσω του ray tracer τις συντεταγμένες υφής για το σημείο τομής του διανύσματος viewVec' με το heightMap.

- Φωτοσκίαση του εικονοστοιχείου με χρήση του κανονικού διανύσματος από τον αντίστοιχο χάρτη.

Φυσικά θα πρέπει να υπολογιστούν εκ νέου τα ecPosition (οι συντεταγμένες στον 3D χώρο του εικονοστοιχείου που κοιτάμε) και lightVec (το διάνυσμα από την πηγή του φωτός προς το εικονοστοιχείο προς απεικόνιση). Πλέον, εφόσον διαθέτουμε το διάνυσμα του φωτός, είναι πολύ εύκολο να πραγματοποιήσουμε real – time shading του αντικειμένου μας, καθώς αυτό που απομένει είναι να εκτελέσουμε ακόμη μια φορά τον ray-tracing αλγόριθμο, μόνο που πλέον τα δεδομένα θα είναι ο πίνακας υψών, οι νέες συντεταγμένες υφής και το διάνυσμα του φωτισμού. Αν βρεθεί τομή με το σημείο προς απεικόνιση, αυτό πρέπει να φωτιστεί. Αλλιώς το σημείο αυτό κρύβεται εξαιτίας κάποιου σημείου με μεγαλύτερο ύψος και πρέπει να το σκιάσουμε.

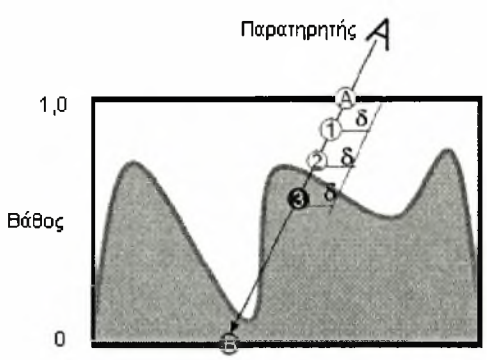


Εικόνα 27: Real Time Shadow Calculation

Ο αλγόριθμος του ray tracing που χρησιμοποιείται στην μέθοδο του Relief Mapping, χωρίζεται σε δύο μέρη. Αρχικά πραγματοποιείται μια σειριακή αναζήτηση (Linear Search) καθορισμένου αριθμού βημάτων, μέχρι να βρεθεί το πρώτο σημείο που τέμνει το διάνυσμα παρατήρησης τον πίνακα υψών (Εικόνα 27) και έπειτα πραγματοποιείται μια δυαδική αναζήτηση με λιγότερα βήματα αναζήτησης, γύρω από το σημείο τομής, έτσι ώστε να εντοπιστεί αυτό με μεγαλύτερη ακρίβεια. Τα βήματα των δύο αναζητήσεων, καθορίζονται από

την επιθυμητή ακρίβεια που θέλουμε να πετύχουμε, πρέπει όμως να δοθεί προσοχή και στον τομέα της απόδοσης. Συγκεκριμένα, τις παραμέτρους αυτές θα μπορούσαμε να τις δίνουμε ως δεδομένα εισόδου στον shader μας, έχοντας τοιουτοτρόπως την δυνατότητα να αυξάνουμε την λεπτομέρεια ανάλογα με τις εκάστοτε απαιτήσεις μας. Κάτι τέτοιο όμως, θα ανάγκαζε τον compiler του shader, να μην «ξεδιπλώσει» τα loops των επαναλήψεων και θα απαιτούσε Shader Model 3 από την κάρτα γραφικών.

Σε σύγκριση με την προσέγγιση του parallax mapping, στην συγκεκριμένη τεχνική η τιμή του ύψους διατηρείται στο εύρος $[0,1]$. Επίσης, αντί του υπολογισμού ενός προσεγγιστικού offset (μέσω του παράλληλου στο πολύγωνο διάνυσματος) που θα φέρει τις αρχικές συντεταγμένες προς το μάτι μας εισάγοντας σφάλμα, η τεχνική του ray tracing, μέσω σταθερού βήματος (εξαρτάται από τις επαναλήψεις που έχουμε επιλέξει) και με γραμμική αναζήτηση, πλησιάζει σιγά-σιγά τις συντεταγμένες μέσω του διανύσματος διεύθυνσης που παρακολουθούμε (είτε το διάνυσμα ματιού, είτε το διάνυσμα του φωτός) και ελέγχει αν κάθε τιμή είναι ορθή. Δηλαδή ελέγχει αν η προκύπτουσα τιμή τέμνει τον πίνακα ύψους. Έπειτα ακολουθεί την ίδια διαδικασία, μεταξύ του σημείου όπου βρέθηκε η τομή και του προηγουμένου του, με χρήση δυαδικής αναζήτησης, ώστε να υπολογιστεί με ακρίβεια το σημείο της τομής του διανύσματος διεύθυνσης με τον πίνακα ύψους.



Εικόνα 28: Γραμμική αναζήτηση για το πρώτο σημείο τομής

Στην μέθοδο αυτή όπως και στο Parallax Mapping, όλοι οι υπολογισμοί μας γίνονται στο tangent space, οπότε και απαιτούνται οι ανάλογες μετατροπές (Βλέπε Normal Mapping).

Όπως θα φανεί και από τις εικόνες παρακάτω, η μέθοδος αυτή μας δίνει μεγάλο ρεαλισμό στην απεικόνιση, ενώ είναι η πρώτη μέθοδος που χωρίς να αλλάζει την γεωμετρία του αντικειμένου, μας δίνει την δυνατότητα να προσθέσουμε ρεαλιστικές σκιές σε πραγματικό χρόνο στο μοντέλο μας. Όπως είναι φυσικό, οι υπολογιστικές της απαιτήσεις είναι πολύ μεγαλύτερες από εκείνες των υπόλοιπων τεχνικών, χωρίς όμως να ξεπερνούν αυτές του Displacement Mapping.

OGSL Shader

Vertex Shader

anthousis@gmail.com

Fragment Shader

anthousis@gmail.com

Εικόνα 29: Relief Mapping Shader

4. Σύνοψη

Η χρήση τεχνικών υφής αποτελεί ένα από τα κατεξοχήν χαρακτηριστικά των 3D γραφικών εφαρμογών και κυρίως της βιομηχανίας λογισμικού παιχνιδιών. Η έρευνα και η ανάπτυξη τεχνικών που προσπαθούν να εκμεταλλευτούν τις δυνατότητες που δίνουν οι νέες κάρτες γραφικών, είναι διαρκής και αυξανόμενη. Ξεκινώντας από το Bump Mapping και το Emboss Bump Mapping, η βιομηχανία των παιχνιδιών πέρασε στο Normal Mapping που αποτελεί την πιο διαδεδομένη τεχνική υφής στις μέρες μας. Ο κύριος λόγος για αυτήν την διαδεδομένη χρήση, αποτελούν τα αρκετά ρεαλιστικά αποτελέσματα, σε συνδυασμό με την ελάχιστη υπολογιστική επιβάρυνση. Ωστόσο δεν είναι μικρός ο αριθμός των εταιριών που φρόντισαν να εκμεταλλευτούν τις νέες διαστάσεις ρεαλισμού που προκύπτουν από περισσότερο εξελιγμένες και σύγχρονες τεχνικές, όπως το Parallax Mapping. Το Parallax Mapping έχει αποδειχθεί στην πράξη ότι μπορεί να βελτιώσει σε μεγάλο βαθμό τον ρεαλισμό της απεικόνισης, επιβαρύνοντας σε αποδεκτά πλαίσια το υπολογιστικό σύστημα. Δεν πρέπει να ξεχνάμε ότι το υλικό του τομέα των γραφικών αναπτύσσεται με ραγδαίους ρυθμούς και έγκειται στους προγραμματιστές να εκμεταλλευτούν στο έπακρο τις νέες δυνατότητες που τους προσφέρονται. Σε αντίθεση με το Parallax Mapping, η μέθοδος του Relief Mapping δεν μπορεί να εφαρμοστεί ακόμη, καθώς το frame rate των εφαρμογών μειώνεται αισθητά, εξαιτίας των πρόσθετων υπολογισμών με συνέπεια το αποτέλεσμα να μην είναι ικανοποιητικό όταν πρόκειται για real time εφαρμογές. Η λύση στο πρόβλημα αυτό αναμένεται να δοθεί στο εγγύς μέλλον από μια παραλλαγή του Relief Mapping που ονομάζεται Steep Parallax Mapping και η οποία υπόσχεται ίδιο επίπεδο ρεαλισμού, αλλά και ικανοποιητικό frame rate ώστε να είναι δυνατή η εφαρμογή αυτής στα 3D παιχνίδια.

Σε αντίθεση με τις παραπάνω τεχνικές που δεν μεταβάλλουν την γεωμετρία των αντικειμένων, το Vertex Displacement Mapping δεν επικράτησε στις εφαρμογές πραγματικού χρόνου εξαιτίας των αυξημένων απαιτήσεών του. Ωστόσο χρησιμοποιείται για την προσθήκη λεπτομέρειας σε αντικείμενα καθώς και για την δημιουργία μοντέλων.

Όλες οι παραπάνω τεχνικές, σε διαφορετικό βαθμό η καθεμία και χρησιμοποιώντας διαφορετικές προσεγγίσεις, επιτυγχάνουν την απεικόνιση υφής σε αντικείμενα. Κάποιες από αυτές τις τεχνικές μεταβάλλουν την γεωμετρία των αντικειμένων, κάποιες χρησιμοποιούν μεγάλη υπολογιστική ισχύ, ενώ κάποιες επιτυγχάνουν τον στόχο τους με αρκετό ρεαλισμό χωρίς να έχουν πολύ μεγάλες απαιτήσεις. Με την διαρκή ανάπτυξη του υλικού των γραφικών και την συνεχή εμφάνιση νέων τεχνικών, αυτό που απομένει είναι να δούμε ποιες από αυτές τις νέες τεχνικές θα μπορέσουν να καθιερωθούν εκμεταλλευόμενες στο έπακρο τις δυνατότητες που προσφέρονται από το υλικό των γραφικών, διατηρώντας παράλληλα το frame rate των εφαρμογών σε ικανοποιητικά επίπεδα.

Βιβλιογραφία

1. Addison Wesley, "*OpenGL Shading Language 2nd Edition (Orange Book)"*
2. Addison Wesley, "OpenGL Programming Guide*, 2nd Edition (Red Book)"*
3. Bill Licea-Kane, ATI Research, Inc. "*The OpenGL Shading Language"*, GameDevelopers Conference
4. Simon Green, NVIDIA, "*Stupid OpenGL Shader Tricks"*, GameDevelopers Conference
5. John Kessenich, Dave Baldwin, Randi Rost, *"The OpenGL Shading Language"*
6. James Blinn, "*Simulation of Wrinkled Surfaces*"
7. Michael I. Gold, NVIDIA, *"Emboss Bump Mapping"*
8. Mark Peercy, John Airey, Brian Cabral, Silicon Graphics Computer Systems, *"Efficient Bump Mapping Hardware"*
9. Tomomichi Kaneko, Toshiyuiki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda, Susumu Tachi, *"Detailed Shape Representation with Parallax Mapping"*
10. Terry Welsh, Infiscape Corporation, *"Parallax Mapping with Offset Limiting: A PerPixel Approximation of Uneven Surfaces"*
11. Michael Doggett, ATI Research, *"Displacement Mapping"*
12. Manuel M. Oliveira, Gary Bishop, David McAllister, *"Relief Texture Mapping"*

13. Fabio Policarpo, Manuel M. Oliveira, Joao L. D. Comba, *"Real-Time Relief Mapping on Arbitrary Polygonal Surfaces"*

14. Manuel M. Oliveira, Fabio Policarpo, *"An Efficient Representation for Surface Details"*

15. Θ. Θεοχάρης, Α. Μπεμ, *"Γραφικά: Αρχές και Αλγόριθμοι"*

16. NVIDIA *"GPU Gems 2"*

Πηγές από το Διαδίκτυο

http1. www.ati.com/developer/tools.html
http2. developer.nvidia.com/
http3. www.3dkingdoms.com/tutorial.htm#creating
http4. freespace.virgin.net/hugo.elias/graphics/x_polybm.htm
http5. en.wikipedia.org/wiki/Bump_mapping
http6. www.blacksmith-studios.dk/projects/downloads/bumpmapping_using_cg.php
http7. www.ogre3d.org/wiki/index.php/BumpMapping
http8. www.delphi3d.net/articles/viewarticle.php?article=phong.htm
http9. en.wikipedia.org/wiki/Displacement_mapping
http10. members.ozemail.com.au/~owenp/relief_textures.htm
http11. g3d-cpp.sourceforge.net/html/guideshaders.html
http12. www.gamedev.net/reference/articles/article1903.asp
http13. www.gametutorials.com
http14. www.flipcode.com/articles/article_3dbumpmap-pf.shtml
http15. www.inf.ufrgs.br/~oliveira/RTM.html
http16. nehe.gamedev.net
http17. 66.70.170.53/Ryan/nrmphoto/nrmphoto.html
http18. en.wikipedia.org/wiki/Normal_mapping
http19. www.clockworkcoders.com/oglsl/
http20. www.ozone3d.net/tutorials/
http21. en.wikipedia.org/wiki/Parallax_mapping
http22. www.pinwire.com/modules.php?name=News&file=print&sid=82
http23. fabio.policarpo.nom.br/relief/index.htm
http24. www.cs.unc.edu/~ibr/projects/RT/RT.html
http25. graphics.cs.brown.edu/games/SteepParallax/index.html