ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μηχανές αναπαράστασης τρισδιάστατων γραφικών πραγματικού χρόνου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του

ΦΩΤΗ Σ. ΞΕΝΙΚΟΥΔΑΚΗ

Επιβλέπων: Στέφανος Κόλλιας

Καθηγητής Ε.Μ.Π.

ΕΡΓΑΣΤΗΡΙΟ ΕΠΕΞΕΡΓΑΣΙΑΣ ΕΙΚΟΝΑΣ ΚΑΙ ΒΙΝΤΕΟ

Αθήνα, Ιούλιος 2009

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Επεξεργασίας Εικόνας και Βίντεο

Μηχανές αναπαράστασης τρισδιάστατων γραφικών πραγματικού χρόνου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του

ΦΩΤΗ Σ. ΞΕΝΙΚΟΥΔΑΚΗ

Επιβλέπων: Στέφανος Κόλλιας

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή επιτροπή την 7η Ιουλίου 2009

……..……	……..……..…...	…….....
Στέφανος Κόλλιας	Ανδρέας Σταφυλοπάτης	Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

.....
Φώτιος Σ. Ξενικουδάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών
Ε.Μ.Π.

Περίληψη

Οι μηχανές αναπαράστασης τρισδιάστατων γραφικών αποτελούν εργαλεία τα οποία επιτρέπουν την διαχείριση και την απεικόνιση τρισδιάστατων δεδομένων. Τα δεδομένα αυτά συνθέτουν μια τρισδιάστατη σκηνή η οποία μπορεί να μεταβάλλεται με το χρόνο. Οι μηχανές τρισδιάστατων γραφικών πραγματικού χρόνου χρησιμοποιούν εξειδικευμένους επεξεργαστές γραφικών για την γρήγορη απεικόνιση στιγμιότυπων της σκηνής, επιτρέποντας την διάδραση σε πραγματικό χρόνο του χρήστη με τον τρισδιάστατο χώρο. Οι μηχανές αυτού του είδους χρησιμοποιούν ορισμένες δομές και τεχνικές διαχείρισης της τρισδιάστατης σκηνής για να επιταχύνουν την διαδικασία απεικόνισης και να προσεγγίσουν την ρεαλιστική αναπαράστασή της τρισδιάστατης σκηνής, επιτυγχάνοντας ταυτόχρονα μεγάλη συχνότητα απεικόνισης στιγμιότυπων.

Στόχος της διπλωματικής εργασίας είναι η περιγραφή των τεχνικών που χρησιμοποιούνται στις μηχανές αναπαράστασης τρισδιάστατων γραφικών πραγματικού χρόνου και η ανάπτυξη μιας τέτοιας μηχανής χρησιμοποιώντας τις βασικότερες από αυτές.

Λέξεις κλειδιά

Μηχανές αναπαράστασης τρισδιάστατων γραφικών, μηχανές αναπαράστασης τρισδιάστατων γραφικών πραγματικού χρόνου, OpenGL.

Abstract

Three dimensional graphics engines are tools that manipulate and render three dimensional data that compose a three dimensional scene. Real time three dimensional graphics engines use special graphics processors to render rapidly scene frames, allowing real time user interaction with the three dimensional space. These engines use specific data structures and techniques to represent and manipulate the three dimensional scene and accelerate the rendering process in order to achieve a realistic representation of the three dimensional scene achieving high frame rates at the same time. In this work, those data structures and techniques are described and a prototype engine is created based on those techniques.

Key words

Three dimensional engines, real time three dimensional engines, OpenGL.

Περιεχόμενα

1. Εισαγωγή

Οι μηχανές αναπαράστασης γραφικών πραγματικού χρόνου διαχειρίζονται τα δεδομένα μιας τρισδιάστατης σκηνής με στόχο την γρήγορη αποτύπωσή τους στην οθόνη του υπολογιστή. Τα δεδομένα αυτά είναι γενικά μεταβλητά ως προς τον χρόνο είτε με κάποιο προκαθορισμένο τρόπο, είτε μέσω της παρέμβασης κάποιου εξωτερικού χρήστη. Επιτυγχάνοντας μεγάλη συχνότητα απεικόνισης των στιγμιότυπων της τρισδιάστατης σκηνής, είναι δυνατή η άμεση αλληλεπίδραση κάποιου χρήστη με τον τρισδιάστατο χώρο όπως του παρουσιάζεται στην οθόνη μέσω της μηχανής γραφικών. Με τον τρόπο αυτό είναι εφικτή η δημιουργία διαδραστικών εικονικών τρισδιάστατων χώρων με εφαρμογή σε διάφορους τομής όπως είναι η εικονική πραγματικότητα, η εξομοίωση συστημάτων ή σε εφαρμογές ψυχαγωγίας.

Η απεικόνιση των τρισδιάστατων δεδομένων γίνεται με την χρήση εξειδικευμένων επεξεργαστών γραφικών (Graphics Processing Units) οι οποίο μπορούν να επεξεργαστούν μεγάλο αριθμό πολυγώνων στις τρεις διαστάσεις για την σύνθεση τρισδιάστατων σκηνών. Οι απαιτήσεις για μεγαλύτερη λεπτομέρεια και βαθμού ρεαλισμού του τρισδιάστατου χώρου δημιούργησαν την ανάγκη του λογισμικού μιας μηχανής η οποία να μπορεί να διαχειριστεί τα δεδομένα αυτά σε πολύ μικρό χρονικό διάστημα. Για τον σκοπό αυτό αναπτύχθηκε ένα σύνολο από δομές και τεχνικές στις οποίες βασίζονται οι μηχανές αναπαράστασης τρισδιάστατων γραφικών.

1.1 Αντικείμενο της εργασίας

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η μελέτη της δομής μιας μηχανής αναπαράστασης τρισδιάστατων γραφικών και η ανάπτυξη μιας τέτοιας μηχανής χρησιμοποιώντας τις βασικότερες από αυτές τις διαδεδομένες δομές και τεχνικές. Η μελέτη και η ανάπτυξη περιλαμβάνουν θέματα όπως είναι η αναπαράσταση τρισδιάστατων αντικειμένων ως προς το σχήμα, το χρώμα και την υφή τους, η αναπαράσταση περιβαλλόντων χώρων, η οργάνωση των αντικειμένων σε μια τρισδιάστατη σκηνή καθώς και τεχνικές που χρησιμοποιούνται για την επιτάχυνση της διαδικασίας απεικόνισης.

1.2 Οργάνωση της εργασίας

Στο δεύτερο κεφάλαιο της εργασίας γίνεται αναφορά σε βασικές μαθηματικές έννοιες και εργαλεία τα οποία είναι απαραίτητα για την υλοποίηση μιας μηχανής αναπαράστασης γραφικών καθώς και για την λειτουργία των επεξεργαστών γραφικών και την διαδικασία απεικόνισης. Το τρίτο κεφάλαιο της εργασίας περιγράφει τις βασικές δομές που χρησιμοποιούνται για την αναπαράσταση τρισδιάστατων αντικειμένων και την οργάνωση τους σε μια τρισδιάστατη σκηνή καθώς και ένα σύνολο από τεχνικές οι οποίες εφαρμόζονται για την επιτάχυνση της διαδικασίας απεικόνισης. Το τέταρτο κεφάλαιο περιγράφει την υλοποίηση μιας πρότυπης μηχανής γραφικών η οποία βασίζεται στις κυριότερες από τις τεχνικές αυτές.

2. Συστήματα απεικόνισης

Η αναπαράσταση τρισδιάστατων γραφικών είναι η διαδικασία κατά την οποία μια δεδομένη σκηνή, η οποία έχει περιγραφεί στις τρεις διαστάσεις, μετατρέπεται σε δυσδιάστατη εικόνα έτσι ώστε να μπορεί να αποτυπωθεί στην οθόνη του υπολογιστή ή σε οποιοδήποτε άλλη επιφάνεια. Η σκηνή αυτή μπορεί να αποτελείται από στατικά αντικείμενα, ή και από αντικείμενα που αλλάζουν θέση ή σχήμα στο χρόνο. Το σύνολο των αλγορίθμων και των εφαρμογών που υλοποιούν αυτή την διαδικασία ονομάζεται μηχανή τρισδιάστατων γραφικών.

2.1 Είδη μηχανών τρισδιάστατων γραφικών.

Υπάρχουν δύο βασικά είδη μηχανών τρισδιάστατων γραφικών, αυτές που βασίζονται σε τεχνικές παρακολούθησης ακτίνας (ray-tracing), και αυτές που βασίζονται σε τεχνικές ραστεροποίησης (rasterisation.)

2.1.1 Μηχανές παρακολούθησης ακτίνας (ray-tracing)

Η παρακολούθηση ακτίνας (ray tracing) είναι τεχνική για την δημιουργία εικόνας παρακολουθώντας την διαδρομή του φωτός μέσα από στοιχεία (pixels) που απαρτίζουν την εικόνα. Η διαδικασία αυτή εξομοιώνει την διαδρομή που ακολουθεί το φως όπως περιγράφεται θεωρητικά από την Φυσική. Η διαδρομή του φωτός, στο κενό, είναι μια ευθεία γραμμή μέχρι να διακοπεί από κάποια επιφάνεια. Μόλις η γραμμή του φωτός συναντήσει μια επιφάνεια, τέσσερα πράγματα μπορούν να συμβούν: απορρόφηση, ανάκλαση, διάθλαση και φθορισμός, ανάλογα με τις φυσικές ιδιότητες της επιφάνειας. Στην συνέχεια, η ανακλώμενες ακτίνες μπορεί στην πορεία τους να συναντούν άλλη επιφάνεια επαναλαμβάνοντας την ίδια διαδικασία. Η ακτίνες οι οποίες φτάνουν τελικά στο μάτι του παρατηρητή σχηματίζουν την τελική εικόνα.

Η τεχνική αυτή μπορεί να δημιουργήσει εικόνες με πολύ μεγάλο βαθμό φωτορεαλισμού αλλά με μεγάλο υπολογιστικό κόστος. Για το λόγω αυτό, η τεχνική παρακολούθησης ακτίνας χρησιμοποιείται περισσότερο σε εφαρμογές όπως στατικές ρεαλιστικές αναπαραστάσεις σκηνών ή σε ειδικά εφφέ ταινιών. Λόγω του μεγάλου χρόνου που απαιτείται για την δημιουργία μιας εικόνας δεν χρησιμοποιείται σε εφαρμογές πραγματικού χρόνου.

2.1.2 Μηχανές ραστεροποίησης (rasterisation)

Η ραστεροποίηση είναι η τεχνική με την οποία μια σκηνή, η οποία περιγράφεται διανυσματικά στο χώρο, μετασχηματίζεται στις δύο διαστάσεις για την αναπαράστασή της σε στοιχεία (pixels) της τελικής εικόνας. Η σκηνή περιγράφεται με πολύγωνα στον χώρο, τα οποία με την σειρά τους περιγράφονται από ένα σύνολο τριγώνων. Τα τρίγωνα περιγράφονται από τρία σημεία στον χώρο (vertices). Ο βασικός αλγόριθμος παίρνει μια σειρά από τρίγωνα, τα οποία τελικά περιγράφουν ολόκληρη την σκηνή, και τα μετασχηματίζει σε δυσδιάστατα τρίγωνα σε μια δυσδιάστατη επιφάνεια η οποία μπορεί να αποτυπωθεί είτε στην οθόνη του υπολογιστή, είτε σε κάποιο αρχείο. Η διαδικασία αυτή είναι εξαιρετικά γρήγορή σε σχέση με την τεχνική παρακολούθησης ακτίνας και για αυτό χρησιμοποιείται σε εφαρμογές πραγματικού χρόνου. Οι εφαρμογές πραγματικού χρόνου γενικά απαιτούν άμεση απόκριση στην είσοδο των χρηστών και πρέπει να παράγουν τουλάχιστον εικοσιτέσσερεις εικόνες το δευτερόλεπτο για να επιτύχουν ομαλή κίνηση. Τέτοιες

εφαρμογές, όπου χρησιμοποιούνται μηχανές ραστεροποίησης, είναι οι εφαρμογές εικονικής πραγματικότητας, εξομοίωσης, και παιχνιδιών στον υπολογιστή.

2.2 Μαθηματικό υπόβαθρο

Οι μηχανές αναπαράστασης τρισδιάστατων γραφικών πραγματικού χρόνου, οι οποίες βασίζονται σε τεχνικές ραστεροποίησης, απαιτούν την χρήση ορισμένων μαθηματικών εργαλείων για τον μετασχηματισμό μιας τρισδιάστατης σκηνής στις δύο διαστάσεις. Κάθε σημείο στον τρισδιάστατο χώρο αναπαρίσταται με ένα τρισδιάστατο διάνυσμα και μπορεί να μετασχηματιστεί χρησιμοποιώντας άλγεβρα πινάκων. Οι βασικοί μετασχηματισμοί είναι η μετατόπιση (translation), η περιστροφή (rotation), η μεγέθυνση (scaling) και η προβολή (projection). Οι μετασχηματισμοί αυτοί απαιτούν πίνακες διαστάσεων 4x4 και για το λόγω αυτό το τρισδιάστατο διάνυσμα που αναπαριστά το σημείο στο χώρο πρέπει να επεκταθεί με ένα επιπλέον στοιχείο το οποίο έχει την τιμή της μονάδας. Στη συνέχεια, πολλαπλασιάζοντας από αριστερά με ένα πίνακα μετασχηματισμού προκύπτει το τελικό διάνυσμα.

2.2.1 Πίνακες μετασχηματισμών

Ο μετασχηματισμός της **μετατόπισης** έχει ως αποτέλεσμα την μετατόπιση ενός σημείου στον τρισδιάστατο χώρο κατά μια σταθερή απόκλιση. Ο πίνακας που υλοποιεί τον μετασχηματισμό της μετατόπισης είναι ο ακόλουθος:

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

όπου X, Y, Z είναι οι διαστάσεις του διανύσματος απόκλισης.

Ο μετασχηματισμός της **μεγέθυνσης** έχει ως αποτέλεσμα την μεγέθυνση του διανύσματος θέσης ενός σημείου στον τρισδιάστατο χώρο ως προς την αρχή των αξόνων, και υλοποιείται με τον ακόλουθο πίνακα:

$$\begin{bmatrix} X & 0 & 0 & 0 \\ 0 & Y & 0 & 0 \\ 0 & 0 & Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

όπου X, Y, Z είναι οι τιμές με τις οποίες θα πολλαπλασιαστεί κάθε διάσταση του διανύσματος θέσης αντίστοιχα. Αν οι τρεις τιμές αυτές δεν είναι ίσες τότε θα έχουμε ασσύμετρη μεγέθυνση.

Ο μετασχηματισμός της **περιστροφής** έχει ως αποτέλεσμα την περιστροφή ενός σημείο γύρω από κάποιον από τους τρεις άξονες X, Y, Z. Ο πίνακας μετασχηματισμού που τον υλοποιεί εξαρτάται από τον άξονα γύρω από τον οποίο εκτελείται η περιστροφή. Συγκεκριμένα, για περιστροφή γύρω από τον άξονα X έχουμε:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Για περιστροφή γύρω από τον άξονα Y:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Για περιστροφή γύρω από τον άξονα Z:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

όπου θ σε κάθε περίπτωση είναι η γωνία περιστροφής γύρω από τον εκάστοτε άξονα. Μια περιστροφή γύρω από ένα αυθαίρετο άξονα μπορεί να υλοποιηθεί με διαδοχικούς μετασχηματισμούς της παραπάνω μορφής γύρω από τους άξονες X, Y, Z. Πρέπει να σημειωθεί ότι σε κάποιες περιπτώσεις, με την παραπάνω διαδικασία, μπορεί να προκύψει το πρόβλημα του κλειδώματος αναρτήρων (gimbal lock). Συγκεκριμένα, υπάρχει περίπτωση μετά από την περιστροφή γύρω από κάποιον άξονα, οι άλλοι δύο άξονες να οδηγηθούν στην ίδια κατεύθυνση με αποτέλεσμα να χαθεί ένας βαθμός ελευθερίας. Το πρόβλημα αυτό μπορεί να λυθεί με διάφορους τρόπους, ο πιο συχνός από τους οποίους, για τις μηχανές τρισδιάστατων γραφικών, είναι η χρήση τετράδων, όπως περιγράφεται παρακάτω.

Κάθε μετασχηματισμός στον τρισδιάστατο χώρο μπορεί να υλοποιηθεί με κάποιο συνδυασμό από αυτούς τους τρεις μετασχηματισμούς. Αφού όλα τα σημεία της τρισδιάστατης σκηνής έχουν μετασχηματιστεί με τον επιθυμητό τρόπο θα πρέπει να μετασχηματιστούν στο δυσδιάστατο επίπεδο της εικόνας μέσω του μετασχηματισμού της **προβολής**. Ο απλούστερος μετασχηματισμός προβολής είναι η **ορθογραφική προβολή**. Ο μετασχηματισμός αυτός έχει την ιδιότητα να διατηρεί στην τελική εικόνα παράλληλες όσες γραμμές είναι παράλληλες και στον τρισδιάστατο χώρο, και υλοποιείται αφαιρώντας απλά την z συνιστώσα από τα τελικά τρισδιάστατα σημεία. Για να επιτευχθεί η προοπτική όμως, όπως την αντιλαμβάνεται το ανθρώπινο μάτι, στην τελική εικόνα, χρησιμοποιείται ο **προοπτικός μετασχηματισμός** με τον ακόλουθο πίνακα:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (N+F)/N & -F \\ 0 & 0 & 1/N & 0 \end{bmatrix}$$

Στην περίπτωση αυτή το οπτικό πεδίο είναι μια κομμένη πυραμίδα (frustum), όπου F και N η απόσταση, από το μάτι του παρατηρητή, του μακρύτερου και του κοντινότερου από τα παράλληλα επίπεδά της.

2.2.2 Τετράδες (Quaternions)

Οι τετράδες (quaternions) στα μαθηματικά είναι μια επέκταση στις τέσσερεις διαστάσεις, των μιγαδικών αριθμών την οποία εισήγαγε για πρώτη φορά ο Ιρλανδός μαθηματικός William Rowan Hamilton το 1843 και χρησιμοποιούνται, εκτός των άλλων, σε υπολογισμούς που αφορούν περιστροφές στο χώρο.

Μια τετράδα q μπορεί να εκφραστεί ως q = (w, x, y ,z) = w + xi + yj + zk, όπου w, x, y, z πραγματικοί αριθμοί και i, j, k είναι σύμβολα σε αντιστοιχία με το i των μιγαδικών αριθμών για τα οποία ισχύει:

$$i^2 = j^2 = k^2 = ijk = -1$$

Επιπλέον ισχύουν όλες οι συνήθεις αλγεβρικές ιδιότητες εκτός από την μεταθετική ιδιότητα του πολλαπλασιασμού. Συνήθως εκφράζεται ως ένα βαθμωτό μέγεθος και ένα διανυσματικό μέγεθος τριών διαστάσεων, q = (w, x, y, z) = [w, **v**], όπου **v** = [x, y, z]. Αποδεικνύεται ότι οι μοναδιαίες τετράδες, δηλαδή οι τετράδες για τις οποίες ισχύει:

$$||q||^2 = w^2 + x^2 + y^2 + z^2 = 1,$$

$$\text{για } q = w + xi + yj + zk$$

μπορούν να αναπαραστήσουν περιστροφές και, συγκεκριμένα, περιστροφή γύρω από ένα άξονα **v = [**x, y, z] κατά γωνία Q αντιστοιχεί στην τετράδα:

$$\mathbf{q} = [\cos(Q/2), \sin(Q/2) * \mathbf{v}]$$

Αν περιστραφεί ένα διάνυσμα **v** από μια τετράδα **q** τότε προκύπτει το διάνυσμα **v'** ως εξής:

$$\mathbf{v'} = \mathbf{q} \,.\, \mathbf{v} \,.\, \mathbf{q}^{-1},$$

όπου $\mathbf{q}^{-1} = (w, x, y, z)^{-1} = w - xi - yj - zk$ για μοναδιαίες τετράδες.

Η αναπαράσταση μιας περιστροφής γύρω από κάποιον άξονα με μια τετράδα, είναι γενικά απλούστερη διαδικασία, υπολογιστικά, από την αναπαράσταση της με τους πίνακες περιστροφής. Επιπλέον, επειδή η περιστροφή γύρω από κάποιον αυθαίρετο άξονα δε γίνεται με διαδοχικές περιστροφές γύρω από τους τρεις άξονες X, Y, Z, αλλά με την χρήση μιας τετράδας, αποφεύγεται το πρόβλημα του κλειδώματος αναρτήρων που περιγράφηκε προηγουμένως.

2.2.3 Σφαιρική γραμμική παρεμβολή

Με την χρήση των τετράδων είναι δυνατό ένα είδος παρεμβολής η οποία παρέχει για ένα αρχικό και τελικό διάνυσμα κατεύθυνσης, τα ενδιάμεσα διανύσματα κατεύθυνσης σε μορφή τετράδων. Η παρεμβολή αυτή ονομάζεται σφαιρική γραμμική παρεμβολή **(**Spherical linear interpolation).

Με την χρήση της σφαιρικής γραμμική παρεμβολής σε μοναδιαίες τετράδες, η διαδρομή των τετράδων αντιστοιχεί σε μια αλληλουχία τρισδιάστατων περιστροφών με κανονικό τρόπο. Το αποτέλεσμα είναι η περιστροφή με σταθερή γωνιακή ταχύτητα γύρω από ένα άξονα περιστροφής. Η παρεμβολή αυτή δίνει την συντομότερη διαδρομή από την αρχική στην τελική τετράδα. Η περιστροφή αυτή όμως μπορεί να είναι διπλή, η τετράδα δηλαδή μπορεί να περιστραφεί κατά την μια φορά διανύοντας γωνία μικρότερη από 180^0 η κατά την άλλη διανύοντας γωνία μεγαλύτερη από 180^0. Το συντομότερο μονοπάτι επιλέγεται από την περιστροφή με την μικρότερη γωνία.

2.3 Συστήματα απεικόνισης γραφικών

Με τον όρο συστήματα απεικόνισης γραφικών (rendering pipelines) για τρισδιάστατα γραφικά εννοούμε τις μεθόδους που έχουν αναπτυχθεί και χρησιμοποιούνται από τις μηχανές ραστεροποίησης για την απεικόνιση μιας τρισδιάστατης σκηνής σε μια εικόνα χρησιμοποιώντας εξειδικευμένες κάρτες γραφικών. Τα βασικά συστήματα που χρησιμοποιούνται κυρίως από τις σύγχρονες εφαρμογές είναι η **OpenGL** και το

Direct3D. Και τα δύο αυτά συστήματα μπορούν να συνθέσουν και να αναπαραστήσουν σύνθετες τρισδιάστατες σκηνές από βασικά γεωμετρικά στοιχεία.

2.3.1 OpenGL

Η **OpenGL** (Open Graphics Library) είναι ένα πρότυπο ανεξάρτητο από πλατφόρμα υλοποίησης και γλώσσας προγραμματισμού που καθορίζει μια διεπαφή (API) για την δημιουργία εφαρμογών που χειρίζονται γραφικά στις δύο και στις τρεις διαστάσεις. Αναπτύχθηκε το 1992 από την Silicon Graphics Inc. και χρησιμοποιείται ευρέως για εφαρμογές εικονικής πραγματικότητας, οπτικοποίησης δεδομένων, συστημάτων εξομοίωσης και παιχνιδιών. Από το 2006 η OpenGL ανήκει στην μη κερδοσκοπική ομάδα Khronos Group. Στο βασικό της επίπεδο, η OpenGL αποτελεί μόνο ένα πρότυπο που περιγράφει συναρτήσεις και τον ακριβή τρόπο λειτουργίας τους. Με βάση αυτό το πρότυπο, οι κατασκευαστές καρτών γραφικών υλοποιούν βιβλιοθήκες με τις παραπάνω συναρτήσεις χρησιμοποιώντας το υλικό των καρτών γραφικών για να επιταχύνουν την λειτουργία τους. Τέτοιες υλοποιήσεις υπάρχουν για όλα τα διαδεδομένα λειτουργικά συστήματα. Επιπλέον υπάρχουν υλοποιήσεις βασισμένες αποκλειστικά σε λογισμικό, οι οποίες δεν απαιτούν εξειδικευμένες κάρτες γραφικών χωρίς φυσικά την ανάλογη επιτάχυνση. Μια περιορισμένη έκδοση του προτύπου, η OpenGL ES, χρησιμοποιείται για μικρές συσκευές με περιορισμένες δυνατότητες, όπως κινητά τηλέφωνα, υπολογιστές χειρός και κονσόλες παιχνιδιών.

2.3.2 Direct3D

Το **Direct3D** είναι ένα παρόμοιο πρότυπο, και αποτελεί μέρος του DirectX που αναπτύσσεται από την Microsoft. Υποστηρίζει αποκλειστικά πλατφόρμες που τρέχουν λειτουργικά σύστημα τύπου Windows (από τα Windows 95 και ύστερα). Μια βασική διαφορά τους είναι ότι το Direct3D χρησιμοποιεί μόνο τις δυνατότητες που παρέχει η εκάστοτε κάρτα γραφικών στην οποία τρέχει και δεν παρέχει υλοποιήσεις λογισμικού για τις δυνατότητες που δεν υποστηρίζει αυτή. Χρησιμοποιείται κυρίως σε εφαρμογές παιχνιδιών και δεν είναι τόσο διαδεδομένο σε επαγγελματικές εφαρμογές όπου προτιμάται γενικότερα η OpenGL.

2.4 Βασικά στοιχεία περιγραφής σκηνής

Ένα σύστημα απεικόνισης δύναται να απεικονίσει μια σύνθετη σκηνή η οποία έχει περιγραφεί από βασικά στοιχεία. Τα στοιχεία αυτά μπορεί να είναι είτε η γεωμετρική περιγραφή της σκηνής, είτε εικόνες η οποίες θα αλλάξουν την εμφάνιση των αντικείμενων για να προσδώσουν ρεαλισμό στην τελική εικόνα.

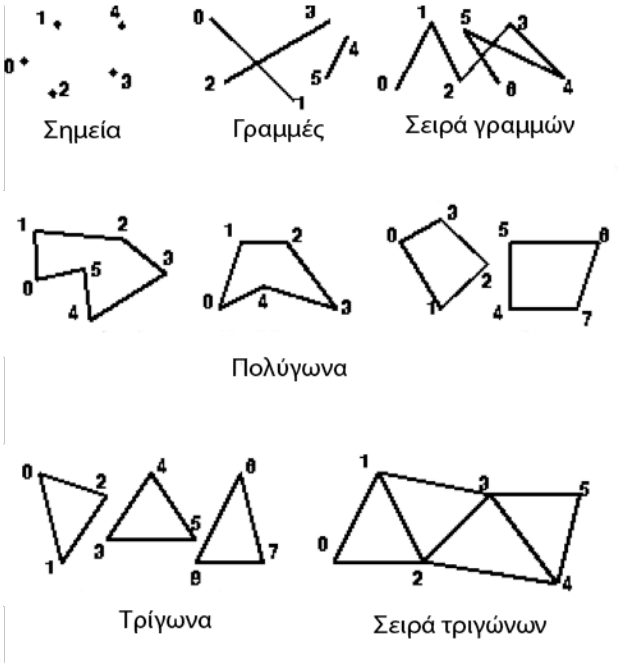
2.4.1 Γεωμετρικά στοιχεία

Τα συστήματα απεικόνισης συνήθως δέχονται τα ακόλουθα βασικά γεωμετρικά στοιχεία για να συνθέσουν μια σκηνή:

- Σημεία
- Ευθύγραμμα τμήματα
- Πολύγωνα

Το κάθε ένα από αυτά τα στοιχεία αποτελείται από μια ή περισσότερες κορυφές στο χώρο (vertices). Το σημείο απαιτεί μια κορυφή, το ευθύγραμμο τμήμα δύο κορυφές και το πολύγωνο τόσες κορυφές όσες είναι η κορυφές του πολυγώνου. Το απλούστερο πολύγωνο, στο οποίο τελικά αναλύονται από το σύστημα απεικόνισης όλα τα πολύγωνα, είναι το τρίγωνο το οποίο είναι και το πιο διαδεδομένο. Τα βασικά στοιχεία αυτά μπορούν να συντεθούν για να περιγράψουν πιο σύνθετα σχήματα ή

επιφάνειες. Για την μείωση την πληροφορίας και την γρηγορότερη απεικόνιση τους, οι επιφάνειες ή τα σχήματα που αποτελούνται από διαδοχικά σημεία μπορούν να περιγραφούν από σειρές σημείων χρησιμοποιώντας μια φορά μόνο κάθε κορυφή που απαιτείται όπως φαίνεται παρακάτω:



Εικόνα 1: Βασικά γεωμετρικά στοιχεία

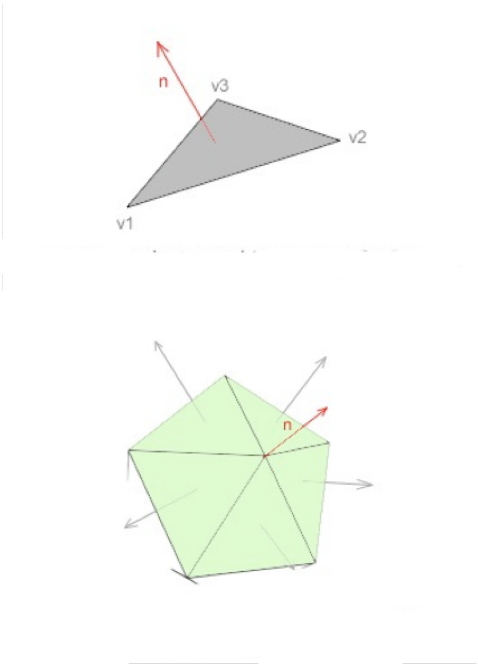
Έτσι για παράδειγμα για την σειρά τριγώνων του σχήματος απαιτούνται 6 σημεία για την αναπαράστασή της αντί για 12 που θα χρειαζόντουσαν για κάθε τρίγωνο ξεχωριστά.

Κάθε επιφάνεια που έχει δοθεί στο σύστημα απεικόνισης αναλύεται σε τρίγωνα τα οποία περιγράφονται από τις κορυφές τους. Για την απεικόνισή τους, σε κάθε κορυφή μπορεί να αντιστοιχηθεί και ένα χρώμα. Το χρώμα στα συστήματα απεικόνισης μπορεί να αναπαρασταθεί με τρεις βασικές συνιστώσες: κόκκινο, πράσινο και μπλε. Ο συνδυασμός αυτών των τριών συνιστωσών μπορεί να παράγει οποιοδήποτε επιθυμητό χρώμα έχει την δυνατότητα να απεικονίσει η οθόνη. Επιπλέον, υπάρχει η δυνατότητα να χρησιμοποιηθεί μια τέταρτη συνιστώσα, η συνιστώσα άλφα (alpha channel). Η συνιστώσα άλφα καθορίζει την διαφάνεια του χρώματος και επιτρέπει στο σύστημα απεικόνισης να σχεδιάσει αντικείμενα που δεν κρύβουν εντελώς τα αντικείμενα που βρίσκονται πίσω τους.

Το σύστημα απεικόνισης κατά την ραστεροποίηση, αφού έχει μετασχηματίσει το τρίγωνο μέσω του προοπτικού μετασχηματισμού στις 2 διαστάσεις, υπολογίζει το χρώμα του εσωτερικού του τριγώνου παρεμβάλωντας σε κάθε σημείο τα χρώματα των κορυφών. Τα συστήματα απεικόνισης έχουν την δυνατότητα να μεταβάλλουν αυτό το χρώμα εξομοιώνοντας τον φωτισμό της σκηνής. Η εξομοίωση αυτή δεν έχει τον ρεαλισμό που έχουν τα συστήματα παρακολούθησης ακτίνας, όπου υπολογίζεται ακριβώς το φως με όλες τις ανακλάσεις του, αλλά δίνει αρκετά ικανοποιητικά αποτελέσματα με γρήγορους υπολογισμούς. Για τους υπολογισμούς αυτούς είναι απαραίτητο να γνωρίζει το σύστημα απεικόνισης την διεύθυνση της επιφάνειας του αντικειμένου σε κάθε κορυφή του σε σχέση με την διεύθυνση του φωτός. Έτσι, όταν είναι επιθυμητή η εξομοίωση του φωτός, πρέπει μαζί με τις συντεταγμένες των

κορυφών κάθε επιφάνειας προς απεικόνιση να δοθεί και το διάνυσμα (normal) της διεύθυνσης της επιφάνειας σε εκείνο το σημείο. Τα συστήματα απεικόνισης δεν υπολογίζουν αυτόματα αυτά τα διανύσματα. Ο υπολογισμός πρέπει να γίνει από πριν και το διάνυσμα διεύθυνσης να δοθεί πριν από κάθε κορυφή στο σύστημα απεικόνισης. Ο υπολογισμός γίνεται ως εξής:

- Για κάθε τρίγωνο του αντικειμένου, υπολογίζεται το κάθετο διάνυσμα **n** από το εξωτερικό γινόμενο $(\mathbf{v_1}-\mathbf{v_2})\mathrm{x}(\mathbf{v_1}-\mathbf{v_3})$, όπου $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$ τα διανύσματα των κορυφών του τριγώνου.
- Για κάθε κορυφή, αθροίζουμε τα κάθετα διανύσματα που υπολογίστηκαν προηγουμένως για κάθε προσκείμενη πλευρά και κανονικοποιούμε το αποτέλεσμα έτσι ώστε το διάνυσμα που προκύπτει να έχει μοναδιαίο μήκος. Το αποτέλεσμα είναι το διάνυσμα διεύθυνσης της επιφάνειας σε εκείνη την κορυφή.



Εικόνα 2: Υπολογισμός κάθετων διανυσμάτων φωτισμού

2.4.2 Στοιχεία εικόνας

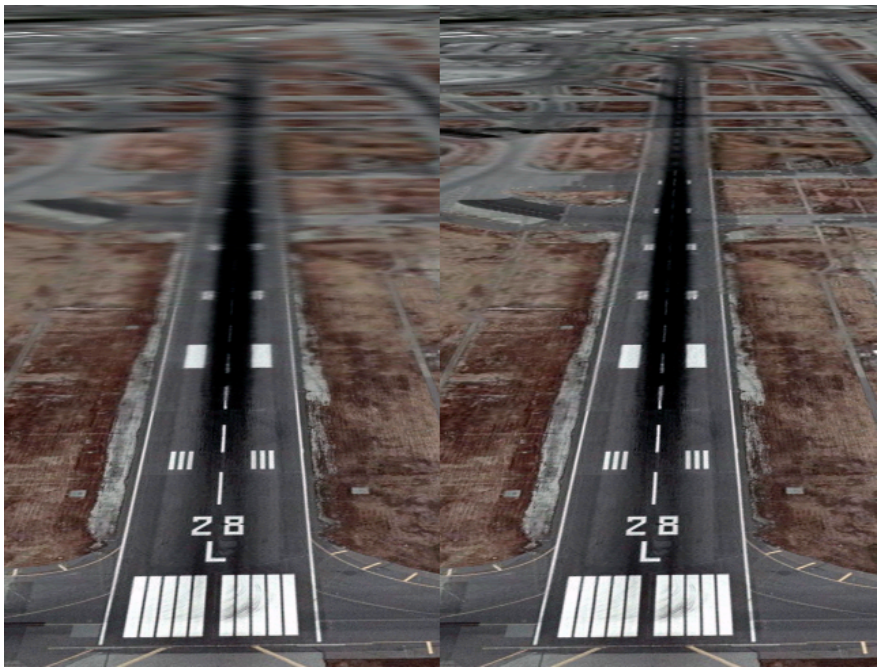
Για να απεικονίσουν ρεαλιστικές σκηνές, τα συστήματα απεικόνισης μπορούν να χρησιμοποιούν στατικές εικόνες (**textures**) οι οποίες σχεδιάζονται πάνω στις επιφάνειες. Με τον τρόπο αυτό, ένα αντικείμενο μπορεί να αποκτήσει συγκεκριμένη υφή χωρίς να χρειαστεί να περιγραφεί με την ακριβή γεωμετρία του. Για παράδειγμα, ένας τοίχος μπορεί να σχεδιαστεί με την χρήση ενός απλού ορθογωνίου παραλληλεπίπεδου και την στατική εικόνα ενός τοίχου σχεδιασμένη επάνω του. Αυτό έχει ως αποτέλεσμα την απεικόνιση αρκετά ρεαλιστικών σκηνών χωρίς το μεγάλο υπολογιστικό κόστος που θα είχε μια πολύπλοκη γεωμετρική περιγραφή. Οι εικόνες αυτές φορτώνονται στην μνήμη του συστήματος και γι' αυτό το λόγο συνήθως γίνεται κάποιος συμβιβασμός ανάμεσα στους περιορισμούς μνήμης του συστήματος και στην επιθυμητή λεπτομέρεια.

Για την αποτύπωση των εικόνων στις επιφάνειες, πρέπει να αντιστοιχηθεί κάθε κορυφή της γεωμετρική περιγραφής των αντικειμένων με ένα σημείο της εικόνας. Μετά τον προοπτικό μετασχηματισμό κάθε επιφάνειας στις δύο διαστάσεις, ο

ραστεροποιητής αντιστοιχεί κάθε σημείο της τελικής επιφάνειας με ένα σημείο της εικόνας υφής παρεμβάλλοντας αντίστοιχα τις συντεταγμένες που του έχουν δοθεί. Όταν η επιφάνεια έχει πολύ μικρότερο μέγεθος από την εικόνα, όπως γίνεται με αντικείμενα που είναι στο βάθος της σκηνής, τότε το σύστημα απεικόνισης πρέπει να επιλέξει το χρώμα που θα αντιστοιχίσει σε κάθε σημείο από μία μεγάλη περιοχή της εικόνας υφής. Αυτό έχει ως αποτέλεσμα μεγάλο υπολογιστικό κόστος για αντικείμενα που ούτως ή άλλως δεν έχουν μεγάλη λεπτομέρεια λόγω του μικρού τους μεγέθους. Για το λόγο αυτό, τα συστήματα απεικόνισης προϋπολογίζουν μικρότερες εκδόσεις της αρχικής εικόνας υφής (**mipmaps**) και χρησιμοποιούν αυτές ανάλογα με το μέγεθος της επιφάνειας. Το σύνολο αυτών των εικόνων ονομάζονται επίπεδα της υφής.

Για την βελτίωση της ποιότητας της εικόνας υφής επάνω στην επιφάνεια μπορούν να χρησιμοποιηθούν επιπλέον φιλτραρίσματα με το ανάλογο υπολογιστικό κόστος:

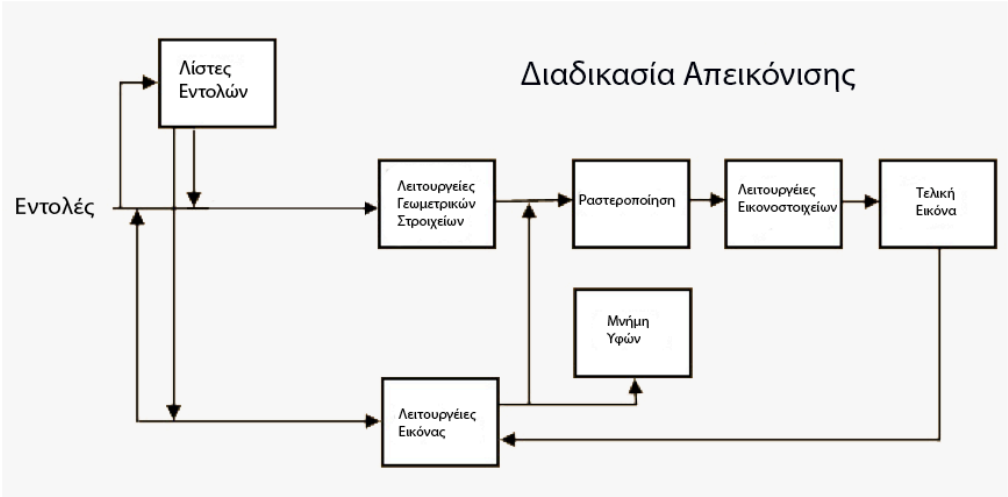
- Διγραμμικό φίλτρο (**Bilinear filtering)**. Εκτός από το χρώμα του σημείου της εικόνας υφής για το κατάλληλο επίπεδο, που αντιστοιχεί στο συγκεκριμένο σημείο της επιφάνειας, χρησιμοποιούνται και τα γειτονικά σημεία αθροισμένα με βάρη ανάλογα με την απόσταση τους.
- Τριγραμμικό φίλτρο (**Trilinear filtering**). Παρόμοια διαδικασία με την προηγούμενη με την προσθήκη ότι χρησιμοποιούνται και σημεία από γειτονικά επίπεδα υφής. Αυτή η διαδικασία βελτιώνει την ποιότητα της εικόνας στο σημείο όπου το σύστημα απεικόνισης αλλάζει το κατάλληλο επίπεδο υφής.
- Ανισοτροπικό φίλτρο (**Anisotropic filtering**). Η διαφορά αυτής της διαδικασίας σε σχέση με τις προηγούμενες είναι ότι, ενώ το διγραμμικό και το τριγραμμικό φίλτρο παίρνουν δείγμα της εικόνας από μια τετράγωνη περιοχή, το ανισοτροπικό φίλτρο χρησιμοποιεί τραπέζιο για την περιοχή ανάλογα με την γωνία της επιφάνειας. Αυτό έχει ως αποτέλεσμα την βελτίωση της ποιότητας της εικόνας σε περιπτώσεις όπου η επιφάνεια δεν είναι κατακόρυφη.



Εικόνα 3: Αριστερά: Χρήση διγραμμικού φίλτρου, Δεξιά: Χρήση ανισοτροπικού φίλτρου

2.5 Διαδικασία απεικόνισης

Τα συστήματα απεικόνισης γραφικών ακολουθούν γενικά μια συγκεκριμένη διαδικασία για την απεικόνιση μιας σκηνής, με κάποιες παραλλαγές ανάλογα με την υλοποίηση. Η διαδικασία αυτή δέχεται ως είσοδο την γεωμετρική πληροφορία που περιγράφει τον χώρο και τα αντικείμενα της σκηνής καθώς και εικόνες η οποίες μπορούν να χρησιμοποιηθούν για να αλλάξουν την εμφάνιση των αντικειμένων και να τους προσδώσουν συγκεκριμένη υφή, και παράγει ως έξοδο την τελική εικόνα η οποία θα εμφανιστεί στην οθόνη. Τα συστήματα απεικόνισης μπορεί να ακολουθήσουν αυτή την διαδικασία άμεσα από την στιγμή που τους δοθεί η είσοδος (immediate mode) ή μπορεί να αποθηκεύσουν όλη την πληροφορία που θα τους δοθεί και ανάλογα με αυτή να αποφασίσουν πότε θα γίνει η απεικόνιση της (retained mode) βελτιστοποιώντας την διαδικασία. Γενικά η διαδικασία αυτή έχει τα εξής στάδια:



Εικόνα 4: Διαδικασία απεικόνισης

2.5.1 Λίστες εντολών
Στις λίστες εντολών αποθηκεύεται ένα σύνολο εντολών το οποίο στην συνέχεια θα οδηγηθεί στην διαδικασία απεικόνισης. Οι εντολές αυτές αφορούν γεωμετρικά δεδομένα και δεδομένα εικόνας. Οι λίστες μπορούν να επαναχρησιμοποιηθούν επιταχύνοντας έτσι την διαδικασία απεικόνισης.

2.5.2 Λειτουργίες γεωμετρικών στοιχείων
Στο στάδιο αυτό, το σύστημα απεικόνισης επεξεργάζεται την γεωμετρική πληροφορία για την απεικόνισή της μέσω του ραστεροποιητή. Η συντεταγμένες των σημείων που απαρτίζουν τα αντικείμενα είναι δοσμένες στο σύστημα αναφοράς του κάθε αντικειμένου. Τα αντικείμενα πρέπει να μετασχηματιστούν με μετασχηματισμούς μεταφοράς και περιστροφής έτσι ώστε να τοποθετηθούν σωστά στο σύστημα αναφοράς της σκηνής (world coordinates). Στη συνέχεια το σύνολο των γεωμετρικών δεδομένων μετασχηματίζεται από το σύστημα αναφοράς της σκηνής στο σύστημα αναφοράς του παρατηρητή (eye coordinates). Πρέπει να σημειωθεί ότι τα συστήματα απεικόνισης όπως η OpenGL θεωρούν ότι ο παρατηρητής είναι τοποθετημένος στο κέντρο των αξόνων και κοιτάει προς το αρνητικό μέρος του άξονα Z. Για το λόγο αυτό, ο μετασχηματισμός των γεωμετρικών δεδομένων από το σύστημα αναφοράς της σκηνής στο σύστημα αναφοράς του παρατηρητή αντιστοιχεί στην μετασχηματισμό των δεδομένων κατά την αντίστροφη διεύθυνση και φορά από την θέση και την κατεύθυνση του παρατηρητή αντίστοιχα. Σε αυτό το σημείο μετασχηματίζονται και τα διανύσματα διεύθυνσης κάθε κορυφής που έχουν υπολογιστεί για τον φωτισμό των αντικειμένων.

Το επόμενο βήμα αυτού του σταδίου είναι ο προοπτικός μετασχηματισμός των γεωμετρικών δεδομένων. Σε αυτό το σημείο όσα σημεία είναι εκτός του οπτικού πεδίου απορρήπτονται από το σύστημα απεικόνισης για την επιτάχυνση της διαδικασίας, αφού δεν θα εμφανιστούν στην τελική εικόνα. Στην συνέχεια όλες οι συντεταγμένες κανονικοποιούνται στο πεδίο τιμών [-1, 1] σε όλους τους άξονες για την αντιστοίχησή τους στην τελική εικόνα. Πρέπει να σημειωθεί ότι σε αυτό το σημείο δεν έχει χαθεί ακόμα η πληροφορία της τρίτης διάστασης η οποία είναι απαραίτητη για την σωστή σειρά σχεδίασης των αντικειμένων.

2.5.3 Λειτουργίες εικόνας
Στο στάδιο αυτό τα δεδομένα εικόνας των υφών μεταφέρονται από το σύστημα στην μνήμη του με κατάλληλο τρόπο για την βέλτιστη χρήση τους. Εδώ γίνεται επίσης και η κατάλληλη επεξεργασία τους για την παραγωγή των επιπέδων υφών που θα χρησιμοποιηθούν αργότερα από τον ραστεροποιητή.

2.5.4 Ραστεροποίηση
Σε αυτό το στάδιο τα γεωμετρικά δεδομένα και τα δεδομένα εικόνας των υφών συντίθενται από τμήματα (fragments). Κάθε τμήμα αντιστοιχεί σε ένα εικονοστοιχείο (pixel) της τελικής εικόνας και περιέχει πληροφορία για το χρώμα του, το βάθος του στη σκηνή καθώς και υπολογισμούς για την εξομάλυνσή του στην τελική εικόνα. Εδώ γίνεται ο υπολογισμός του χρώματος κάθε σημείο με βάση τη θέση του σε κάθε επιφάνεια, το χρώμα των κορυφών, τον φωτισμό και την υφή της επιφάνειας.

2.5.5 Λειτουργίες εικονοστοιχείων
Σε αυτό το στάδιο γίνεται η επεξεργασία όλων των εικονοστοιχείων για την παραγωγή της τελικής εικόνας. Εδώ υπολογίζεται το ακριβές χρώμα κάθε σημείου στην τελική εικόνα συνυπολογίζοντας την διαφάνεια (transparency) και άλλες

διαδικασίες εξομάλυνσης της εικόνας. Επιπλέον, το σύστημα απεικόνισης μπορεί να χρησιμοποιήσει μια σειρά από αποταμιευτές (buffers) που έχουν τις διαστάσεις της τελικής εικόνας και φυλάσσουν πληροφορία για πιο σύνθετους υπολογισμούς ανά σημείο.

2.6 Προγράμματα σκίασης (Programmable Shaders)

Τα σύγχρονα σύστημα απεικόνισης επιτρέπουν την παρεμβολή ειδικών προγραμμάτων στην διαδικασία απεικόνισης τα οποία δίνουν την δυνατότητα στους προγραμματιστές να μεταβάλλουν τόσο τα γεωμετρικά δεδομένα όσο και την τελικά εικονοστοιχία. Τα προγράμματα αυτά εκτελούνται στην κάρτα γραφικών του συστήματος η οποία είναι σχεδιασμένη για τέτοιες Λειτουργίες με αποτέλεσμα να είναι εφικτά, σε πραγματικό χρόνο, εφφέ που δεν θα ήταν δυνατά με την παραδοσιακή διαδικασία. Τέτοια εφφέ είναι για παράδειγμα περισσότερο σύνθετοι φωτισμοί, εξομοίωση φυσικών αντικειμένων όπως είναι οι επιφάνειες νερού και αλλοιώσεις στην κίνηση των αντικειμένων που προσδίδουν μεγαλύτερο ρεαλισμό.

Τα προγράμματα αυτά χωρίζονται σε τρεις κατηγορίες, ανάλογα με το σημείο της διαδικασίας απεικόνισης στο οποίο παρεμβάλλονται:

- Προγράμματα κορυφών (**Vertex Shaders**). Τα προγράμματα αυτά επεξεργάζονται τα γεωμετρικά δεδομένα και μπορούν να μεταβάλουν την θέση τους, το χρώμα τους και την συντεταγμένες τους στην εικόνα υφής. Τα προγράμματα αυτά δεν έχουν την δυνατότητα να αφαιρέσουν ή να δημιουργήσουν νέες κορυφές.
- Προγράμματα γεωμετρίας (**Geometry Shaders**). Τα προγράμματα αυτά έχουν την δυνατότητα να μεταβάλουν την γεωμετρία των αντικειμένων, προσθέτοντας και αφαιρώντας κορυφές δίνοντας στο αντικείμενο λεπτομέρεια η οποία θα ήταν χρονοβόρο να υπολογιστεί μέσω του κανονικού επεξεργαστή.
- Προγράμματα εικόνας (**Pixel Shaders**). Τα προγράμματα αυτά επεξεργάζονται το χρώμα των τελικών εικονοστοχειών για να παράγουν σύνθετους φωτισμούς ή άλλα εφφέ βασισμένα στο χρώμα κάθε σημείου.

3. Μηχανές τρισδιάστατων γραφικών πραγματικού χρόνου

Στο κεφάλαιο αυτό θα περιγραφούν οι μηχανές τρισδιάστατων γραφικών πραγματικού και τα υποσυστήματα από τα οποία αποτελούνται. Οι μηχανές τρισδιάστατων γραφικών πραγματικού χρόνου είναι σύνθετα εργαλεία τα οποία χρησιμοποιούν συστήματα απεικόνισης, τα οποία περιγράφηκαν παραπάνω, για να αναπαραστήσουν σειρά εικόνων πολύπλοκων διαδραστικών τρισδιάστατων σκηνών στο χρόνο με αρκετά μεγάλο βαθμό ρεαλισμού. Επειδή τα συστήματα απεικόνισης βασίζονται στον σχεδιασμό πολύ απλών γεωμετρικών στοιχείων, η λειτουργία των μηχανών τρισδιάστατων γραφικών απαιτεί την μοντελοποίηση όλων των αντικειμένων και του περιβάλλοντος χώρου που συνθέτουν την σκηνή, με τρόπο συμβατό με την λειτουργία των συστημάτων απεικόνισης, καθώς και τον χειρισμό της αλληλεπίδρασής τους στο χρόνο τόσο μεταξύ τους όσο και με κάποιον εξωτερικό χρήστη. Είναι η απαραίτητη η άμεση απόκριση αυτών των μηχανών στις αλλαγές της κατάστασης του εικονικού κόσμου που περιγράφουν για την ικανοποιητική διαδραστική εξομοίωσή του και για το λόγο αυτό η μοντελοποίηση και διαχείριση των δεδομένων πρέπει να γίνεται με τέτοιο τρόπο ώστε η δημιουργία μιας στατικής εικόνας να απαιτεί τον λιγότερο δυνατό χρόνο. Ο χρόνος αυτός είναι συνάρτηση της επεξεργαστικής ισχύς του μηχανήματος το οποίο παράγει την εικόνα, αλλά είναι πολύ περισσότερο συνάρτηση των δομών και τον αλγορίθμων που χρησιμοποιούνται για την ρεαλιστική αναπαράσταση του εικονικού κόσμου.

Οι μηχανές τρισδιάστατων γραφικών είναι πολύπλοκα συστήματα τα οποία αποτελούνται από διάφορα τμήματα. Κάθε ένα από αυτά έχει ξεχωριστό ρόλο στην εξομοίωση και αναπαράσταση μιας τρισδιάστατης σκηνής στο χρόνο.

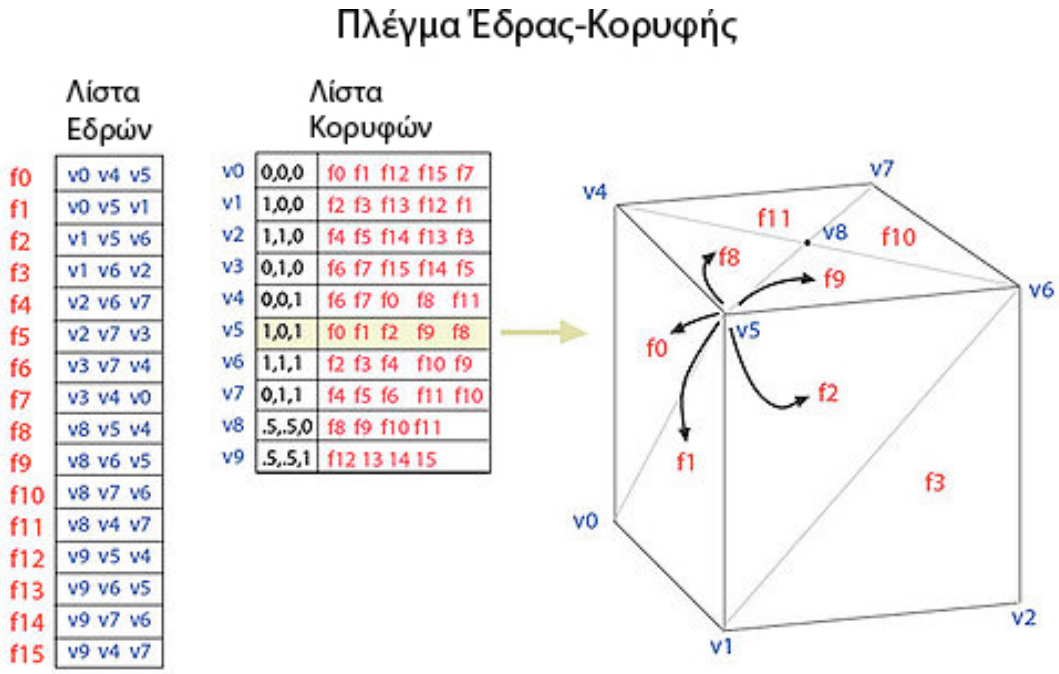
3.1 Μοντελοποίηση αντικειμένων

Ένα από τα βασικότερα τμήματα των μηχανών τρισδιάστατων γραφικών είναι η αναπαράσταση των αντικειμένων που συνθέτουν μια σκηνή. Η αναπαράσταση αυτή θα πρέπει να είναι συμβατή με τον τρόπο λειτουργίας των συστημάτων απεικόνισης και κατά συνέπεια θα πρέπει να μπορεί να αναλυθεί κατά την απεικόνιση σε απλά γεωμετρικά στοιχεία, όπως είναι τα τρίγωνα. Για το λόγο αυτό, η συνηθέστερη μοντελοποίηση αντικειμένων γίνεται με την χρήση πλεγμάτων τριγώνων (**triangle meshes**). Άλλοι τρόποι αναπαράστασης είναι η χρήση επιφανειών **NURBS** για ομαλότερες επιφάνειες και η τα συστήματα σωματιδίων (**particle systems**).

3.1.1 Πλέγμα τριγώνων

Το πλέγμα τριγώνων είναι ένα σύνολο από κορυφές, ακμές και έδρες που καθορίζουν ένα πολυεδρικό αντικείμενο. Οι κορυφές αυτές συνθέτουν, ανά τρεις, τριγωνικές έδρες οι οποίες με την σειρά τους συνθέτουν μια πολυεδρική επιφάνεια η οποία προσεγγίζει την επιφάνεια του αντικειμένου το οποίο επιθυμούμε να αναπαραστήσουμε. Υπάρχουν διάφοροι τρόποι αναπαράστασης ενός πλέγματος τριγώνων με διαφορετικά προτερήματα ανάλογα με την εφαρμογή. Οι παράγοντες που καθορίζουν την κατάλληλη αναπαράσταση είναι η ευκολία εξαγωγής των τριγωνικών πλευρών για την μεταφορά τους στο σύστημα απεικόνισης, και η δυνατότητα ή μη της μεταβολής της γεωμετρίας του αντικειμένου.

Ο πιο διαδεδομένος τρόπος αναπαράστασης ενός πλέγματος τριγώνων για αντικείμενα τα οποία έχουν σταθερή γεωμετρία ονομάζεται πλέγμα έδρας-κορυφής (**face-vertex mesh**). Για την αναπαράσταση αυτή απαιτείται μια λίστα κορυφών του αντικειμένου με τις συντεταγμένες τους στο χώρο και μια λίστα με τις έδρες του αντικειμένου και τις κορυφές από τις οποίες αποτελείται. Η αναπαράσταση αυτή επιτρέπει την εύκολη εξαγωγή των τριγωνικών εδρών για την μεταφορά τους στο σύστημα απεικόνισης. Επιπλέον, είναι δυνατή η μεταβολή του σχήματος του αντικειμένου αλλάζοντας μόνο τις συντεταγμένες των κορυφών αλλά δεν είναι δυνατή η αλλαγή της γεωμετρίας τους καθώς δεν είναι εύκολη η δημιουργία νέων εδρών για τον χωρισμό του αντικειμένου η την συγχώνευσή του με κάποιο άλλο. Σε περιπτώσεις όπου δεν είναι απαραίτητη η μεταβολή της γεωμετρίας του αντικειμένου, είναι ο πιο αποδοτικός τρόπος αναπαράστασης.



Εικόνα 5: Παράδειγμα αναπαράστασης κύβου με Πλέγμα Έδρας-Κορυφής

Η αναπαράσταση αυτή περιλαμβάνει μόνο την γεωμετρική περιγραφή ενός αντικειμένου, δηλαδή οι συντεταγμένες στο χώρο κάθε κορυφής στο σύστημα συντεταγμένων του αντικειμένου. Η χρήση του από μια μηχανής γραφικών μπορεί να απαιτεί επιπλέον πληροφορία ανά κορυφή ανάλογα με την εφαρμογή, όπως για παράδειγμα τα κάθετα διανύσματα σε κάθε κορυφή για τον φωτισμό και την αντιστοιχία κάθε κορυφής με συντεταγμένες σε κάποια εικόνα υφής. Οι πληροφορίες αυτές δεν αυξάνουν την πολυπλοκότητα της δομής αυτής, αποτελούν απλά εμπλουτισμό των δεδομένων.

3.1.2 Συστήματα σωματιδίων (Particle Systems)

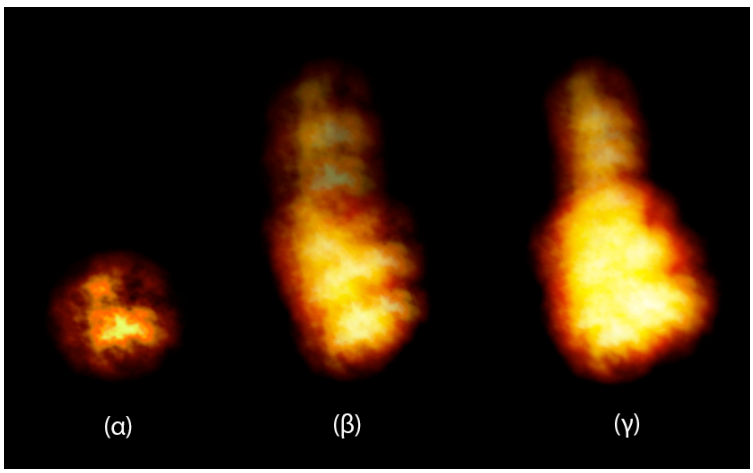
Τα συστήματα σωματιδίων είναι μια τεχνική η οποία χρησιμοποιείται για την αναπαράσταση αντικειμένων ή φαινόμενων τα οποία είτε δεν είναι εύκολο να αναπαρασταθούν με τις μεθόδους μοντελοποίησης που αναφέρθηκαν παραπάνω, είτε απαιτούν μεγάλο υπολογιστικό κόστος για την μοντελοποίησή τους. Παραδείγματα

τέτοιων αντικειμένων είναι φωτιές, εκρήξεις, καπνοί, νερό σε κίνηση, χιόνι, σκόνη, τριχώματα, γρασίδι κτλ. Τέτοια αντικείμενα εξομοιώνονται με ένα σύνολο σωματιδίων (**particles**) τα οποία έχουν μια απλή απεικόνιση ανάλογα με το αντικείμενο που εξομοιώνουν.

Τα συστήματα σωματιδίων έχουν ένα **πομπό** (**emitter**) με την δική του θέση και ταχύτητα. Αυτός ο πομπός λειτουργεί ως πηγή που δημιουργεί σωματίδια και η θέση του καθορίζει το σημείο από το οποίο ξεκινάει η κίνηση τους. Ο ρυθμός δημιουργίας σωματιδίων καθώς και οι αρχικές τους ιδιότητες καθορίζονται από το αντικείμενο που αναπαριστούν. Το κάθε σωματίδιο έχει πεπερασμένο χρόνο ζωής και ορισμένες παραμέτρους, επίσης ανάλογα με το αντικείμενο που αναπαριστούν, οι τιμές των οποίων αλλάζουν στο χρόνο. Οι πιο συνηθισμένες παράμετροι των σωματιδίων είναι η θέση, η ταχύτητά και η κατεύθυνση τους αλλά ανάλογα με την εφαρμογή μπορεί να έχουν και άλλες παραμέτρους όπως είναι για παράδειγμα το χρώμα ή η διαφάνεια. Επιπλέον, οι τιμές των παραμέτρων και η εξέλιξή τους στο χρόνο περιλαμβάνουν ένα τυχαίο παράγοντα για να δώσουν περισσότερο ρεαλιστική αίσθηση.

Ένα τυπικό σύστημα σωματιδίων εξελίσσεται στο χρόνο με βάση κάποιους καθορισμένους κανόνες. Οι κανόνες αυτοί περιγράφουν τις αρχικές τιμές που θα δώσει ο πομπός στα σωματίδια καθώς και την μεταβολή τους στο χρόνο. Ανάλογα με το αντικείμενο, μπορούν να είναι από πολύ απλοί, όπως είναι η κίνηση τους με σταθερή ταχύτητα, μέχρι πολύ σύνθετοι, όπως για παράδειγμα η εξομοίωση φαινομένων όπως είναι η βαρύτητα, η τριβή και άλλες εξωτερικές δυνάμεις. Πολλές φορές οι κανόνες αυτοί λαμβάνουν υπόψη τους και τα υπόλοιπα αντικείμενα τις σκηνής για ανάλογη αλληλεπίδραση.

Η εμφάνιση των σωματιδίων ποικίλει, συνήθως όμως είναι απλή καθώς ο μεγάλος αριθμός των σωματιδίων μπορεί να απαιτεί μεγάλη υπολογιστική ισχύ. Συχνότερη είναι η χρήση στατικών τετράγωνων εικόνων με σταθερή γωνία ως προς τον παρατηρητή, αλλά μπορούν να χρησιμοποιηθούν και απλά γεωμετρικά αντικείμενα.



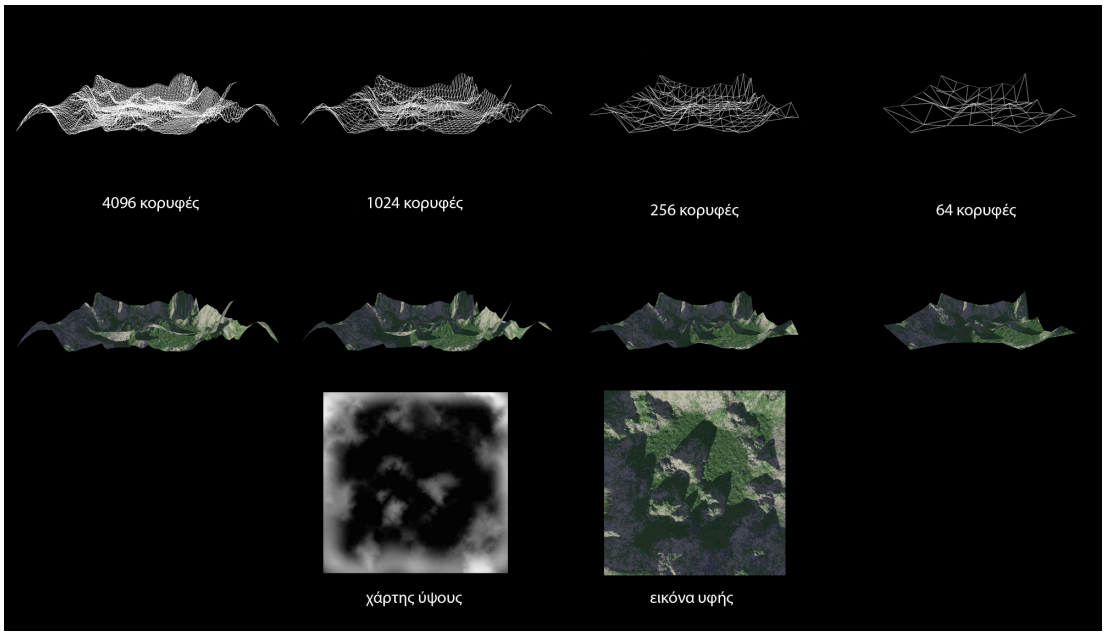
Εικόνα 6: Φωτιά με σύστημα σωματιδίων. (α) εικόνα σωματιδίου, (β) φωτιά με 20 σωματίδια, (γ) φωτιά με 100 σωματίδια.

3.1.3 Χάρτες Ύψους.

Ένας χάρτης ύψους (**heightmap**) είναι ένας δυσδιάστατος πίνακας, κάθε στοιχείο του οποίου έχει μια τιμή η οποία αντιστοιχεί σε μια τιμή ύψους. Ο πίνακας αυτός μπορεί να αντιστοιχηθεί με μια επιφάνεια κάθε σημείο (X, Y) της οποίας έχει ύψος ίσο με

την τιμή του πίνακα σε εκείνο το σημείο. Οι τιμές του μπορούν να θεωρηθούν ότι προκύπτουν από μετατόπιση του αντίστοιχου σημείου από το έδαφος, και μπορούν να οπτικοποιηθούν μέσω μιας μαυρόασπρης εικόνας όπου το μαύρο αντιστοιχεί στα σημεία με ελάχιστο ύψος και το άσπρο στα σημεία με μέγιστο ύψος. Μια μηχανή γραφικών μπορεί να αναπαραστήσει έδαφος βασισμένο σε χάρτη ύψους μετατρέποντας τον σε πλέγμα τριγώνων. Επιπλέον, συνήθως χρησιμοποιείται και μια εικόνα υφής στην οποία είναι σχεδιασμένο το έδαφος.

Ένας τρόπος μετατροπής του χάρτη ύψους σε πλέγμα τριγώνων είναι να θεωρήσουμε ένα ορθογώνιο πλέγμα x*y κορυφών σε διάταξη x στηλών και y γραμμών. Οι γειτονικές κορυφές σχηματίζουν τετράγωνα τα οποία μπορούν να χωριστούν σε δύο τρίγωνα με βάση την διαγώνιό τους. Το μήκος και το πλάτος καθορίζονται αυθαίρετα με βάση την εφαρμογή ενώ το ύψος κάθε κορυφής προκύπτει από τον χάρτη ύψους. Το πλέγμα κορυφών δεν είναι απαραίτητο να έχει τις ίδιες διαστάσεις με τον χάρτη ύψους. Το ύψος κάθε κορυφής μπορεί να προκύψει από παρεμβολή των γειτονικών τιμών του χάρτη. Ανάλογα με τον αριθμό των κορυφών που έχουν επιλεγεί προκύπτει και αντίστοιχος αριθμός τριγώνων. Συγκεκριμένα, από ένα πλέγμα x*y κορυφών προκύπτουν 2*(x-1)*(y-1) τρίγωνα. Μεγάλος αριθμός τριγώνων προσεγγίζει με μεγαλύτερη λεπτομέρεια την επιφάνεια που αναπαριστά ο χάρτης ύψους, αλλά έχει και μεγαλύτερο υπολογιστικό κόστος σχεδιασμού. Παρακάτω φαίνεται η αναπαράσταση ενός χάρτη ύψους για 4096, 1024, 254 και 64 κορυφές.

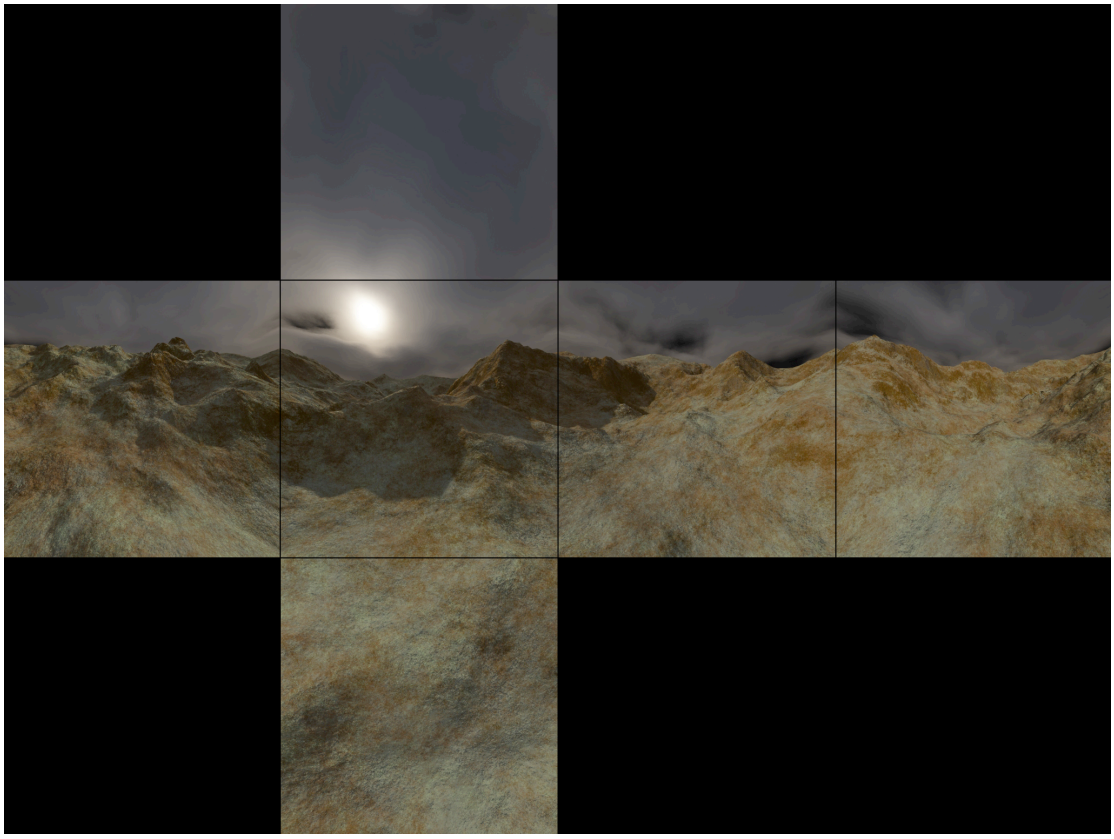


Εικόνα 7: Χάρτης ύψους

3.1.4 Κουτί Ουρανού

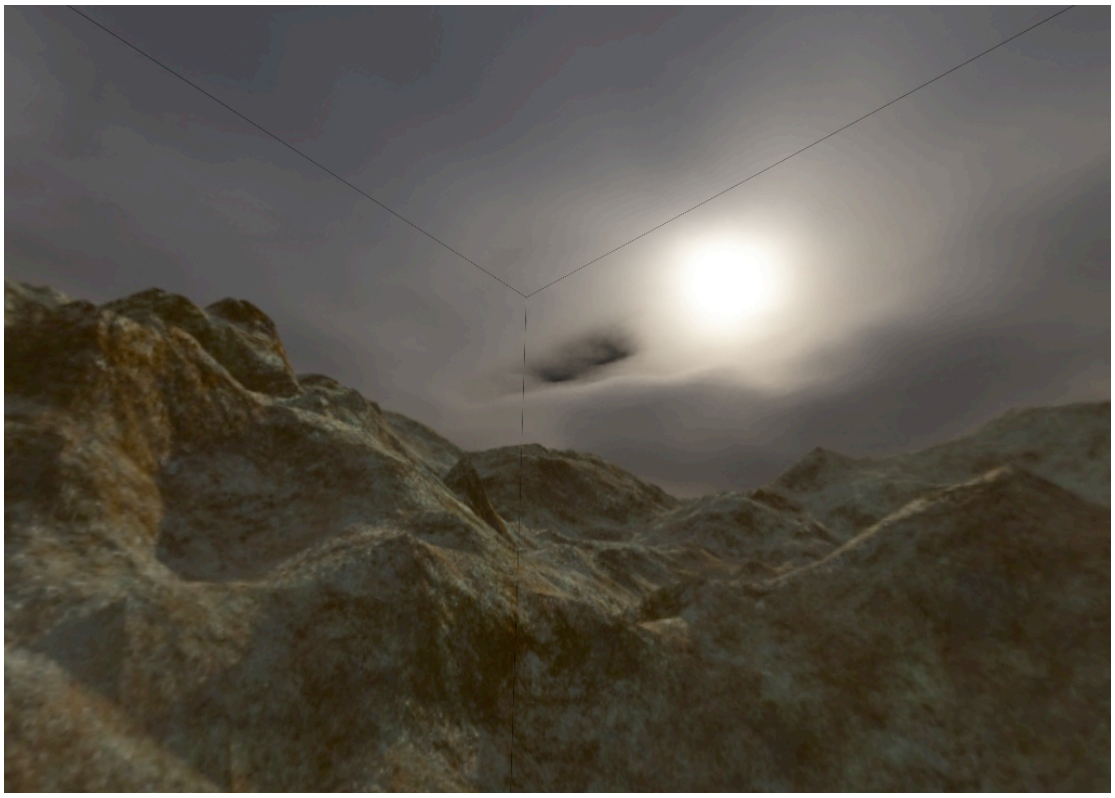
Το κουτί ουρανού (**skybox**) είναι μια τεχνική η οποία δημιουργεί την ψευδαίσθηση του ουρανού και των αντικείμενων του περιβάλλοντος χώρου τα οποία είναι πολύ μακριά από την κάμερα. Η ακριβής αναπαράσταση των αντικειμένων αυτών καθώς και του ουρανού και των ουράνιων αντικειμένων θα απαιτούσε μεγάλο υπολογιστικό κόστος για τον σχεδιασμό μακρινών αντικειμένων τα οποία όμως δεν θα φαινόντουσαν με μεγάλη λεπτομέρεια. Αντ' αυτού χρησιμοποιείται ένα τρισδιάστατος κύβος, το κουτί ουρανού, πάνω στο οποίο έχουν προσχεδιαστεί ο ουρανός και τα μακρινά αντικείμενα, όπως βουνά ή μακρινά κτήρια. Για τον σκοπό

αυτό χρειάζονται έξι εικόνες, μία για κάθε πλευρά του κύβου. Οι εικόνες όμως πρέπει να έχουν παρθεί με τέτοιο τρόπο έτσι ώστε να δίνουν την ψευδαίσθηση συνεχούς περιβάλλοντος χώρου. Αυτό γίνεται παίρνοντας διαδοχικές εικόνες, είτε από φωτογραφία είτε από την απεικόνιση μιας τρισδιάστατης σκηνής. Όλες οι εικόνες πρέπει να παρθούν από το ίδιο σημείο, η κάμερα πρέπει να κοιτάει για κάθε εικόνα κάθετα προς την πλευρά του κύβου στην οποία αντιστοιχεί, και με γωνία θέασης 90^0.. Παρακάτω φαίνονται οι έξι εικόνες του ουρανού και το αποτέλεσμα της σχεδίασης του κουτιού ουρανού. Τα όρια των πλευρών είναι εμφανή για καλύτερη κατανόηση της θέσης του κύβου.



Εικόνα 8: Οι έξι εικόνες υφής ενός κουτιού ουρανού

Παραλλαγή του κουτιού ουρανού είναι ο θόλος ουρανού (**skydome**) ο οποίος αποτελείται από ημισφαίριο ή σφαίρα αντί για κύβο. Για μεγαλύτερο ρεαλισμό μπορούν επιπλέον να χρησιμοποιηθούν απλά γεωμετρικά σχήματα με τις κατάλληλες υφές για την απεικόνιση πχ. σύννεφων ή μακρινών κτηρίων.



Εικόνα 9: Εικόνα περιβάλλοντος χώρου από κουτί ουρανού

3.3 Μοντελοποίηση Κίνησης

Στην γενική περίπτωση, η κίνηση των αντικειμένων σε μια τρισδιάστατη σκηνή αντιστοιχεί σε μεταβολή της θέσης και της κατεύθυνσης τους με μια καθορισμένη ταχύτητα. Σε μακροσκοπικό επίπεδο, αυτό αντιστοιχεί στην κίνηση ενός αντικειμένου στο χώρο, το οποίο πιθανότατα έχει αναπαρασταθεί με ένα πλέγμα τριγώνων, χωρίς να μεταβάλλεται το σχήμα του. Αυτό δεν επαρκεί για να δώσει ρεαλιστικά αποτελέσματα σε περισσότερο πολύπλοκες κινήσεις, όπως για παράδειγμα το περπάτημα ενός ανθρώπου. Διάφορες τεχνικές, οι οποίες θα περιγραφούν παρακάτω, έχουν χρησιμοποιηθεί για να αντιμετωπίσουν τέτοιες περιπτώσεις.

3.2.1 Στιγμιότυπα-Κλειδιά

Η κίνηση με Στιγμιότυπα-Κλειδιά αυτή έχει χρησιμοποιηθεί πρώτα στην κίνηση δυσδιάστατων χαρακτήρων σε κινούμενα σχέδια, αλλά η λογική της έχει επεκταθεί στις τρεις διαστάσεις και χρησιμοποιείται ευρέως. Η τεχνική αυτή μπορεί να αναπαραστήσει προσχεδιασμένες κινήσεις αντικειμένων με την χρήση διαδοχικών στιγμιότυπων της στάσης του αντικειμένου από μία αρχική στάση σε μία τελική. Τα στιγμιότυπα αυτά έχουν την ίδια γεωμετρία αλλά διαφορετικές θέσεις των σημείων. Έτσι, με την αναπαράσταση πλέγματος τριγώνων, οι έδρες που ανήκουν στο αντικείμενο είναι κοινές για κάθε στιγμιότυπο και οι κορυφές του παραμένουν ίδιες στον αριθμό αλλά με διαφορετικές θέσεις, με αποτέλεσμα να απαιτείται μόνο ένας ξεχωριστός πίνακας συντεταγμένων κορυφών για κάθε στιγμιότυπο.

Για μια ομαλή μπορεί να απαιτούνται περισσότερα από 25 στιγμιότυπα ανά δευτερόλεπτο. Η περιγραφή του αντικειμένου δεν περιλαμβάνει όλα τα στιγμιότυπα καθώς κάτι τέτοιο θα απαιτούσε πολύ χρόνο στην σχεδίαση του αντικειμένου αλλά

και στην μνήμη που θα καταλάμβανε. Αντίθετα, χρησιμοποιούνται ορισμένα στιγμιότυπα-κλειδιά ανά συγκεκριμένα χρονικά διαστήματα. Τα ενδιάμεσα στιγμιότυπα υπολογίζονται από τις θέσεις των κορυφών των στιγμιότυπων-κλειδιών πριν και μετά την επιθυμητή στιγμή. Ο πιο συνηθισμένος τρόπος, εξαιτίας της ταχύτητάς του, είναι η γραμμική παρεμβολή των συντεταγμένων των κορυφών. Αν για παράδειγμα έχουμε δύο στιγμιότυπα τις στιγμές t_1, t_2 και μια κορυφή τους έχει συντεταγμένες $\mathbf{v}_1$, $\mathbf{v}_2$ αντίστοιχα σε κάθε στιγμιότυπο, τότε η ίδια κορυφή σε μία ενδιάμεση στιγμή θα έχει συντεταγμένες:

$$\mathbf{v} = \mathbf{v}_1 - (t - t_1)(\mathbf{v}_2 - \mathbf{v}_1)/(t_2 - t_1)$$

Ανάλογα με τον αριθμό των στιγμιότυπων-κλειδιών που έχουν χρησιμοποιηθεί, ειδικά σε περιπτώσεις όπου τμήμα του αντικειμένου περιστρέφεται, η γραμμική παρεμβολή μπορεί να δώσει αίσθηση ανώμαλης κίνησης. Εναλλακτικά, μπορούν να χρησιμοποιηθούν άλλα είδη παρεμβολών, με μεγαλύτερο υπολογιστικό κόστος, όπως είναι οι παρεμβολές με καμπύλες Bezier και οι παραλλαγές τους.

3.2.2 Κίνηση Σκελετού

Η τεχνική αυτή χρησιμοποιείται κυρίως για κίνηση σπονδυλωτών χαρακτήρων, όπως είναι οι άνθρωποι και τα ζώα. Η κίνηση σκελετού απαιτεί, εκτός από την κανονική αναπαράσταση του αντικειμένου, την περιγραφή ενός σκελετού ο οποίος χρησιμοποιείται μόνο για την κίνηση και δεν απεικονίζεται στην τελική εικόνα. Ο σκελετός αυτός είναι μια ιεραρχική δομή από οστά. Κάθε οστό αντιστοιχεί σε ένα τρισδιάστατο μετασχηματισμό (που περιλαμβάνει την θέση και την κατεύθυνση του οστού) και προαιρετικά ένα οστό-πατέρα. Η μετασχηματισμός του κάθε οστού είναι ως προς το οστό-πατέρα και η τελική του θέση και κατεύθυνση στο χώρο προκύπτει από το γινόμενο των μετασχηματισμών του οστού και του τελικού μετασχηματισμού του οστού πατέρα. Έτσι προκύπτει μία ιεραρχική δομή οστών, όπου κάθε οστό επηρεάζεται από το οστό με το οποίο συνδέεται. Κάθε οστό συσχετίζεται με κάποιο τρόπο με ένα τμήμα της αναπαράστασης του αντικειμένου και επηρεάζει την θέση της. Στην κλασσική περίπτωση του πλέγματος τριγώνων, κάθε οστό επηρεάζει την θέση μίας ομάδας κορυφών. Μία κορυφή είναι δυνατό να επηρεάζεται από περισσότερα από ένα οστά και η τελική της θέση προκύπτει από συνδυασμό των θέσεων των οστό συνυπολογίζοντας συνήθως κάποιο βάρος ανά οστό.

Με την τεχνική αυτή, η κίνηση του αντικειμένου προκύπτει από την κίνηση του σκελετού. Ο απλούστερος τρόπος είναι με τη χρήση στιγμιότυπων του σκελετού, ή και τμημάτων του σκελετού για μεγαλύτερη ευελιξία στην περιγραφή κινήσεων, και παρεμβολή τους αντίστοιχα με την τεχνική των στιγμιότυπων-κλειδιών. Με αυτή την τεχνική, επειδή είναι γνωστή η επιθυμητή κατεύθυνση της αρχικού και του τελικού στιγμιότυπου του σκελετού, μπορεί να γίνει παρεμβολή της με την χρήση της παρεμβολής SLERP η οποία έχει περιγραφεί παραπάνω.

3.2.3 Ανάστροφη Κινηματική (Inverse Kinematics)

Η τεχνική αυτή βασίζεται στην κίνηση σκελετού, δηλαδή χρησιμοποιεί ένα σκελετό μαζί την αναπαράσταση του αντικειμένου, αλλά δεν βασίζεται σε προσχεδιασμένες κινήσεις. Χρησιμοποιείται όταν είναι γνωστή η τελική θέση κάποιων οστών του σκελετού και είναι επιθυμητός ο υπολογισμός της θέσης των υπολοίπων. Ένα παράδειγμα είναι η περίπτωση όπου ένας άνθρωπος θέλει να πιάσει ένα αντικείμενο με το χέρι του και το άκρο το σκελετού του χεριού θα μετακινηθεί στην θέση του αντικειμένου που θέλει να πιάσει. Το πρόβλημα της ανάστροφης κινηματικής δεν έχει

αναλυτική λύση στην γενική του περίπτωση, και πρέπει αντιμετωπίζεται συνήθως με τεχνικές μη γραμμικού προγραμματισμού. Επιπλέον, στην γενική περίπτωση, μπορούν να υπάρχουν περισσότερες από μία λύσεις που ικανοποιούν τους περιορισμούς και κάποιες από αυτές δεν οδηγούν σε κινήσεις με την επιθυμητή φυσικότητα.

3.3 Περιβάλλοντες όγκοι και επικάλυψη

Σε πολλές περιπτώσεις, μια μηχανή γραφικών χρειάζεται να γνωρίζει αν κάποιο αντικείμενο επικαλύπτεται από κάποιο άλλο στο χώρο. Στη γενική περίπτωση κάτι τέτοιο θα απαιτούσε ελέγχους σε όλα τα τρίγωνα της αναπαράστασης των αντικειμένων. Σε περιπτώσεις όπου δεν είναι απαραίτητη μεγάλη ακρίβεια, έχει μικρότερο υπολογιστικό κόστος ο αντίστοιχος έλεγχος στους περιβάλλοντες όγκους των αντικειμένων. Περιβάλλων όγκος (**Bounding volume**) ενός αντικειμένου είναι ένας κλειστός όγκος ο οποίος περιέχει πλήρως το αντικείμενο αυτό. Οι περιβάλλοντες όγκοι έχουν συχνά απλούστερες αναπαραστάσεις από τις αναπαραστάσεις που χρησιμοποιούνται για την απεικόνισή των αντικειμένων και για τον λόγο αυτό είναι γρηγορότερος ο έλεγχος της επικάλυψής τους. Ανάλογα με τον βαθμό που προσεγγίζουν το αντικείμενο που περικλείουν υπάρχουν περιπτώσεις όπου οι περιβάλλοντες όγκοι επικαλύπτονται χωρίς να επικαλύπτονται τα ίδια τα αντικείμενα.

3.3.1 Περιβάλλου σφαίρα (Bounding sphere)

Περιβάλλου σφαίρα ενός αντικειμένου είναι μια σφαίρα η οποία περικλείει πλήρως το αντικείμενο αυτό και περιγράφεται από το κέντρο και την ακτίνα της. Ο έλεγχος επικάλυψης δυο περιβάλλουσων σφαιρών είναι πολύ απλός. Δύο σφαίρες επικαλύπτονται αν η απόσταση των κέντρων τους είναι μικρότερη ή ίση από το άθροισμα των ακτίνων τους. Ο έλεγχος αυτός είναι γρήγορος αλλά η Περιβάλλου σφαίρα έχει το μειονέκτημα ότι δεν συνήθως δεν περιβάλλει στενά το αντικείμενο που περικλείει.

3.3.2 Περιβάλλων κύλινδρος (Bounding cylinder)

Περιβάλλων κύλινδρος είναι ένας κύλινδρος ο οποίος περιέχει ένα αντικείμενο. Τις περισσότερες φορές ο άξονας του κυλίνδρου είναι παράλληλος με τον κατάκορφο άξονα της σκηνής. Οι κύλινδροι χρησιμοποιούνται συνήθως για αντικείμενα τα οποία περιστρέφονται μόνο ως προς τον κατακόρυφο άξονα. Δύο κύλινδροι επικαλύπτονται όταν οι προβολές τους στο οριζόντιο επίπεδο (δύο κύκλοι) επικαλύπτονται. Και οι δύο έλεγχοι αυτοί είναι εύκολοι. Εξαιτίας αυτών, οι περιβάλλωντες κύλινδροι χρησιμοποιούνται ως περιβάλλοντες όγκοι για ανθρώπους οι οποίοι στέκονται όρθιοι.

3.3.3 Περιβάλλον κουτί

Ένα κουτί το οποίο περιβάλλει ένα αντικείμενο ονομάζεται περιβάλλον κουτό. Αν οι πλευρές του είναι ανά δύο παράλληλες στους τρεις άξονες αντίστοιχα ονομάζεται **AABB** (**Axis Aligned Bounding Box**). Τα AABB περιβάλλουν αρκετά καλά αντικείμενα με παραπλήσιο σχήμα και χρησιμεύουν για τον έλεγχο επικάλυψης αντικειμένων τα οποία ακουμπούν το ένα πάνω στο άλλο, όπως για παράδειγμα ένα αυτοκίνητο ακουμπά στο έδαφος. Ο έλεγχος επικάλυψης δύο κουτιών τους είναι απλός καθώς απαιτεί ελέγχους των προβολών (ευθύγραμμα τμήματα) τους στους τρεις άξονες αντίστοιχα. Έχουν το μειονέκτημα όμως ότι αν το αντικείμενο που περιβάλλουν περιστραφεί τότε πρέπει να επαναυπολογιστούν.

Παραλλαγή τους είναι τα προσανατολισμένα κουτιά, **OBB** (**Oriented Bounding Box**) τα οποία είναι ορθογώνια παραλληλεπίπεδα αλλά με συγκεκριμένο προσανατολισμό

3.3.4 Διακριτά προσανατολισμένα πολύτοπα (discrete oriented polytopes)

Ένα **discrete oriented polytope** (**DOP**) αποτελεί μια γενική περίπτωση των AABB και είναι ένα κυρτό πολύτοπο (πολύεδρο στις τρεις διαστάσεις) το οποίο περικλείει ένα αντικείμενο. Κατασκευάζεται παίρνοντας έναν αριθμό από κατάλληλα προσανατολισμένα επίπεδα στο άπειρο και μετακινώντας τα μέχρι να συναντήσουν το αντικείμενο. Το DOP είναι το κυρτό πολύτοπο που προκύπτει από την τομή των ημιχώρων που περικλείονται από τα επίπεδα. Το AABB είναι μια συχνή εκδοχή ενός DOP από έξι επίπεδα προσανατολισμένα στους άξονες. Άλλα συχνά DOP που χρησιμοποιούνται στις μηχανές γραφικών είναι τα λοξά περιβάλλοντα κουτιά (**beveled bounding box**) τα οποία αποτελούνται από 10 επίπεδα (αν είναι λοξά μόνο ως προς τις κατακόρυφες ακμές), από 18 (αν είναι λοξά σε όλες τις ακμές) ή 26 επίπεδα (αν είναι λοξά σε όλες τις ακμές και κορυφές). Ένα DOP το οποίο αποτελείται από k επίπεδα ονομάζεται k-DOP. Ο πραγματικός αριθμός των επιπέδων μπορεί να είναι μικρότερος καθώς πολλές πλευρές μπορεί να εκφυλιστούν σε μια ακμή ή κορυφή.

3.3.5 Θεώρημα διαχωριστικών αξόνων (Separating axis theorem)

Για κάποιους περιβάλλοντες όγκους (όπως είναι τα AABB, OBB και τα κυρτά πολύεδρα), ένας αποτελεσματικός έλεγχος επικάλυψής τους είναι αυτός του θεωρήματος διαχωριστικών αξόνων. Σύμφωνα με το θεώρημα διαχωριστικών αξόνων δύο κυρτά σχήματα δεν αποκαλύπτονται αν και μόνο αν υπάρχει άξονας στον οποίο οι προβολές τους δεν επικαλύπτονται. Συνήθως οι άξονες οι οποίοι εξετάζονται είναι οι βασικοί άξονες των όγκων. Στην περίπτωση των AABB και των OBB αυτοί είναι οι άξονες στους οποίους είναι προσανατολισμένοι οι όγκοι.

Για ένα AABB το οποίο έχει μήκος **L** και κέντρο **C** τότε τα όρια της προβολής του m, n σε ένα άξονα **N** είναι:

$$m = b - r, n = b + r, \text{ όπου}$$

$$r = 0.5L_x|N_x| + 0.5L_y|N_y| + 0.5L_z|N_z| \text{ και } b = C * N$$

Ένα OBB είναι παρόμοιο αλλά λίγο πιο σύνθετο. Για ένα με **L** και **C** όπως παραπάνω, και με **I**, **J**, **K**, σαν τους άξονες βάσης του OBB τότε:

$$r = 0.5L_x|N * I| + 0.5L_y|N * J| + 0.5L_z|N * K|$$

και m, n όπως προηγουμένως.

Για τα διαστήματα m,n και o,p ισχύει ότι δεν επικαλύπτονται αν m>p ή o>n. Έτσι, προβάλλοντας τα διαστήματα δύο OBB κατά μήκος των I, J, K αξόνων του κάθε OBB και ελέγχοντας για μη επικάλυψη, είναι δυνατό να εξακριβωθεί η μη επικάλυψη των OBB. Ελέγχοντας επιπλέον για μη επικάλυψη κατά μήκος των εξωτερικών γινομένων των αξόνων αυτών ($I_0 \times I_1$, $I_0 \times J_1$, ...) μπορεί να εξακριβωθεί περισσότερο σίγουρα η μη επικάλυψη.

Η λογική εξακρίβωσης της μη επικάλυψης μέσω της προβολής στους άξονες επεκτείνεται και στα κυρτά πολύεδρα, χρησιμοποιώντας όμως άξονες κάθετους στις έδρες των πολύεδρων αντί για τους βασικούς άξονες, και με τα όρια των προβολών βασισμένα στα ελάχιστα και μέγιστα εσωτερικά γινόμενα κάθε κορυφής με τους άξονες.

3.3.6 Ιεραρχία περιβάλλοντων όγκων (Bounding volume hierarchy)

Μία ιεραρχία περιβάλλοντων όγκων είναι ένα δέντρο από περιβάλλοντες όγκους (όπως σφαίρες, AABB ή/και OBB). Οι κόμβοι στα φύλλα του δέντρου περιέχουν όγκους οι οποίοι περικλείουν ακριβώς κάποιο αντικείμενο (ή πιθανώς ένα μικρότερο τμήμα του αντικειμένου σε μεγαλύτερου βαθμού ιεραρχίες). Ένας κόμβος με κόμβους παιδιά περιέχει έναν όγκο ο οποίος περικλείει όλους τους όγκους των κόμβων-παιδιών του. Η ρίζα του δέντρου περιέχει έναν όγκο ο οποίος περικλείει όλους τους όγκους του δέντρου, δηλαδή ολόκληρη την σκηνή.

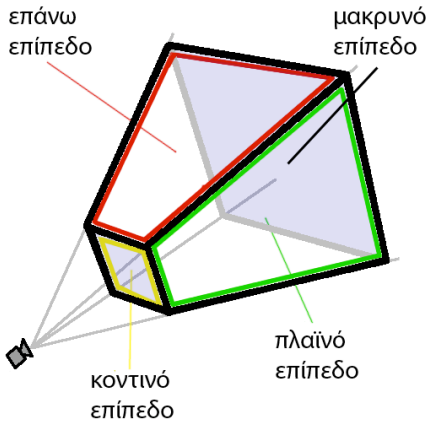
Οι ιεραρχίες όγκων μπορούν να επιταχύνουν ελέγχους επικάλυψης. Αν ο περιβάλλων όγκος ενός αντικειμένου δεν επικαλύπτει τον περιβάλλων όγκο ενός κόμβου στην ιεραρχία, τότε δεν θα επικαλύπτει και τους όγκους των κόμβων παιδιών του και κατά συνέπεια τα αντικείμενα που περικλείουν. Έτσι ξεκινώντας τους ελέγχους από την ρίζα της ιεραρχίας, μπορούν να εξαιρεθούν ολόκληρα υποδέντρα.

3.4 Έλεγχος ορατότητας

Οι μηχανές τρισδιάστατων γραφικών χρησιμοποιούν τα συστήματα απεικόνισης για να απεικονίσουν ένα σύνολο από τρίγωνα με τις υφές τους για την αναπαράσταση μιας τρισδιάστατης σκηνής. Τις περισσότερες φορές μέρος της σκηνής, συχνά μεγάλο, δεν είναι ορατό στην τελική εικόνα καθώς είναι εκτός του οπτικού πεδίου του παρατηρητή. Αν το μη ορατό μέρος της σκηνής σταλεί στο σύστημα απεικόνισης, μετά από τους μετασχηματισμούς που εφαρμόζονται σε κάθε τρίγωνο, θα απορριφθεί από το σύστημα καθώς οι συντεταγμένες του θα υπολογιστούν εκτός της τελικής εικόνας, έχοντας όμως ήδη καταναλώσει υπολογιστική ισχύ. Είναι επομένως σημαντικό για την μηχανή γραφικών να μπορεί να εκτιμήσει εκ των προτέρων, όσο το δυνατόν καλύτερα, τα κομμάτια της σκηνής τα οποία είναι εκτός του οπτικού πεδίου για να περιορίσει την λειτουργία του συστήματος απεικόνισης μόνο σε ότι είναι ορατό.

3.4.1 Αποκλεισμός οπτικού πεδίου

Ο βασικότερος έλεγχος τον οποίο μπορεί να κάνει η μηχανή γραφικών είναι ο αποκλεισμός αντικειμένων από το οπτικό πεδίο (**frustum culling)**. Το οπτικό πεδίο ενός παρατηρητή ή της κάμερας σε μία μηχανή τρισδιάστατων γραφικών έχει το σχήμα μια πυραμίδας με κομμένη κορυφή (**frustum**). Οι βάσεις της πυραμίδας είναι επίπεδα παράλληλα μεταξύ τους και κάθετα ως προς την κατεύθυνση του ματιού του παρατηρητή. Θεωρητικά η κοντινή στον παρατηρητή βάση (**near plane**) είναι τοποθετημένη στο σημείο όπου βρίσκεται ο παρατηρητής, και η μακρινή βάση (**far plane**) είναι τοποθετημένη στο άπειρο. Πρακτικά αυτό δεν είναι δυνατό και καθορίζεται μια συγκεκριμένη απόσταση των δύο πεδίων ανάλογα με το βάθος του οπτικού πεδίου το οποίο επιθυμούμε να είναι ορατό. Οι άλλες πλευρές της πυραμίδας σχηματίζουν ανά δύο γωνία η οποία καθορίζει το εύρος του οπτικού πεδίου (**field of view**). Τα έξι επίπεδα μπορούν να υπολογιστούν εξαρχής από την μηχανή γραφικών για σταθερές διαστάσεις του οπτικού πεδίου.



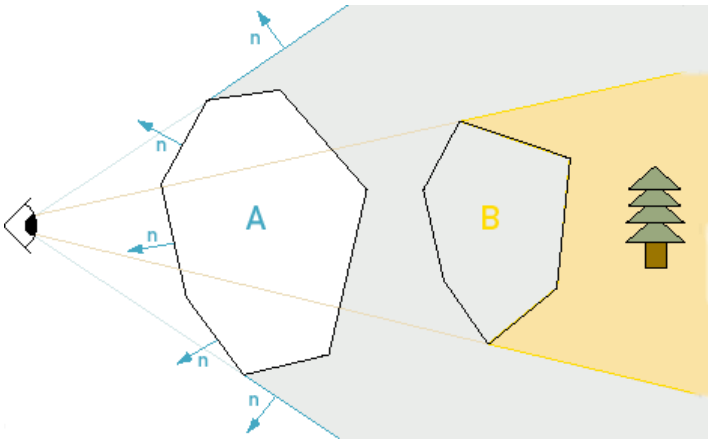
Εικόνα 10: Οπτικό πεδίο

Ο έλεγχος ορατότητας με την χρήση του οπτικού πεδίου μπορεί να γίνει για κάθε αντικείμενο ελέγχοντας την θέση του ως προς τα έξι επίπεδα του οπτικού πεδίου. Αν για κάθε επίπεδο του οπτικού πεδίου όλα τα σημεία του αντικειμένου βρίσκονται στον εξωτερικό ημιχώρο που ορίζει το επίπεδο, τότε το αντικείμενο βρίσκεται εκτός του οπτικού πεδίου και μπορεί να απορριφθεί από την μηχανή γραφικών για την απεικόνισή του. Αντί της αναπαράστασης του αντικειμένου, είναι αποδοτικότερο να χρησιμοποιηθεί ο περιβάλλον όγκος του καθώς ο έλεγχος της θέσης του ως προς κάποιο επίπεδο είναι πολύ γρηγορότερος από τον έλεγχο που απαιτείται από κάποιο σύνθετο αντικείμενο. Επιπλέον, είναι δυνατό να χρησιμοποιηθεί μια ιεραρχία περιβάλλοντων όγκων, όπως περιγράφηκε παραπάνω, έτσι ώστε να αποφευχθεί ο έλεγχος των αντικειμένων που ανήκουν σε υποδέντρο εκτός του οπτικού πεδίου.

3.4.2 Αποκλεισμός κάλυψης.

Συχνά, αντικείμενα τα οποία είναι εντός του οπτικού πεδίου δεν είναι ορατά εξαιτίας άλλων αντικειμένων τα οποία τα καλύπτουν. Για παράδειγμα, αν ο παρατηρητής είναι κοντά σε ένα μεγάλο αντικείμενο, πχ ένα τοίχο, και κοιτάει προς αυτό, τα αντικείμενα πίσω από το αντικείμενο αυτό είναι εντός της πυραμίδας του οπτικού πεδίου αλλά δεν είναι ορατά. Σε αυτή την περίπτωση είναι αποδοτικό να εξαιρεθούν και αυτά από την διαδικασία απεικόνισής τους. Η διαδικασία αυτή όμως είναι αποδοτική μόνο για αντικείμενα τα οποία έχουν μεγάλο όγκο σε σχέση με τα υπόλοιπα αντικείμενα της σκηνής, έτσι ώστε να είναι πιθανό να καλύπτουν σημαντικό αριθμό πολυγώνων. Σε άλλη περίπτωση το κόστος του υπολογισμού της κάλυψης υπερβαίνει το κόστος σχεδιασμού των κρυμμένων πολυγώνων.

Για κάθε αντικείμενο που θεωρούμε ότι είναι πιθανό να καλύπτει μεγάλο αριθμό αντικειμένων χρησιμοποιούμε την αναπαράσταση ενός κυρτού όγκου ο οποίος δεν περιέχει σημεία που δεν ανήκουν στο αντικείμενο. Ο όγκος αυτός θεωρούμε ότι κρύβει τα αντικείμενα πίσω του ως προς τον παρατηρητή. Η περιοχή κάλυψης που δημιουργεί ο όγκος αυτός αποτελείται από την περιοχή του όγκου και το τμήμα του οπτικού πεδίου το οποίο βρίσκεται πίσω από τον όγκο. Η περιοχή αυτή περιβάλλεται από όλες τις έδρες του όγκου οι οποίες είναι στραμμένες προς τον παρατηρητή και από τις έδρες που σχηματίζονται από τις ακμές του όγκου και το σημείο του παρατηρητή. Η κατεύθυνση των εδρών ως προς τον παρατηρητή μπορεί να υπολογιστεί από το εσωτερικό γινόμενο του κάθετου ως προς την έδρα δυάνισμα **n** και της κατεύθυνσης του παρατηρητή.



Εικόνα 11: Κάλυψη αντικειμένων

Ο έλεγχος κάλυψης ενός αντικείμενου από τον όγκο κάλυψης που έχει προκύψει, γίνεται κατά τρόπο αντίστοιχο με την κάλυψη οπτικού πεδίου, ελέγχοντας την θέση των σημείων τους ως προς τα επίπεδα που τον περιβάλλουν και απορρίπτοντας τα αντικείμενα τα οποίο είναι εντός του όγκου. Ο έλεγχος αυτός χρειάζεται μόνο για τα αντικείμενα τα οποία είναι εντός του οπτικού πεδίου, καθώς τα υπόλοιπα έχουν ήδη απορριφθεί από τον έλεγχο κάλυψης οπτικού πεδίου.

3.4.3 Έλεγχος κάλυψης από ιεραρχικές δομές περιγραφής σκηνής.

Σε πολλές περιπτώσεις μπορούν να υπολογιστούν οι περιοχές της σκηνής οι οποίες είναι ορατές από κάποιο αυθαίρετο σημείο χρησιμοποιώντας ιεραρχικές δομές οι οποίες περιγράφουν χωρικά την σκηνή. Οι δομές αυτές χρησιμοποιούνται κυρίως σε σκηνές κλειστού χώρου, όπως για παράδειγμα το εσωτερικό κτηρίων, όπου μόνο μικρό μέρος της σκηνής είναι ορατό κάθε φορά, πχ. ένα δωμάτιο. Οι δομές αυτές περιγράφονται στην επόμενη ενότητα.

3.5 Ιεραρχικές χωρικές δομές

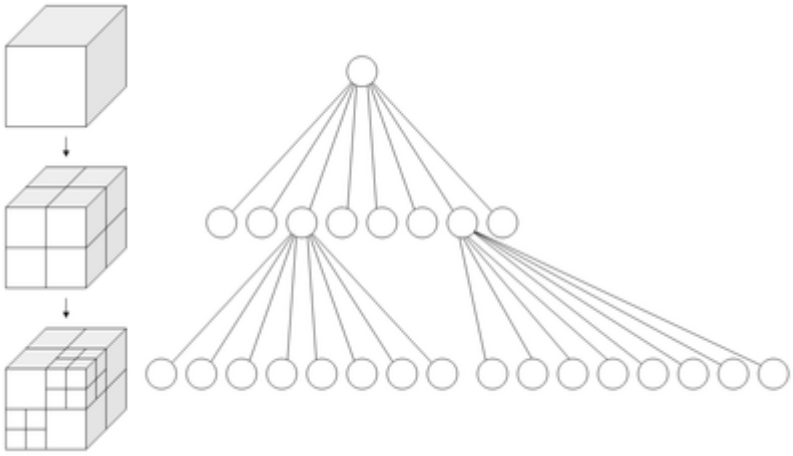
Στην περίπτωση σύνθετων σκηνών είναι πολλές φορές αποδοτικό να χρησιμοποιηθεί κάποιας μορφής ιεραρχική χωρική δομή για την αναπαράστασή τους. Η δομές αυτές, συνήθως αναδρομικές, χωρίζουν αναδρομικά τον χώρο της σκηνής σε μικρότερους υποχώρους, επιταχύνοντας έτσι αρκετές διαδικασίες που απαιτούν ελέγχους των πολυγώνων της σκηνής καθώς ολόκληρα τμήματα της σκηνής μπορούν να ελεγχθούν γρήγορα. Κάτι τέτοιο είναι πολύ χρήσιμο για την μηχανή γραφικών όταν χρειάζεται να αποφασίσει πχ. τα τμήματα της σκηνής τα οποία είναι ορατά αλλά και για άλλους ελέγχους που μπορεί να απαιτούν πρόσβαση στα πολύγωνα της σκηνής.

3.5.1 Οχταδικά δέντρα (Octrees)

Ένα οχταδικό δέντρο είναι μια δενδρική δομή στην οποία κάθε εσωτερικός κόμβος έχει ακριβώς οχτώ παιδιά. Στις μηχανές γραφικών, τα οχταδικά δέντρα χρησιμοποιούνται για να χωρίσουν μια σκηνή σε υποχώρους για την αποδοτικότερη διαχείρισή της.

Σε ένα οχταδικό δέντρο, η ρίζα του δέντρου αντιστοιχεί σε ένα κύβο ο οποίος περικλείει όλο το χώρο της σκηνής, περιέχει δηλαδή όλα τα πολύγωνα της σκηνής. Στη συνέχεια, ο χώρος αυτός διαιρείται σε οχτώ ίσους κύβους, από τρία κάθετα μεταξύ τους επίπεδα. Η διαδικασία αυτή συνεχίζεται αναδρομικά έως ότου κάποιο κριτήριο τερματισμού ικανοποιηθεί για κάποιο κόμβο. Το κριτήριο αυτό είναι

συνήθως ο ελάχιστος αριθμός πολυγώνων που επιθυμούμε να περιέχει ο κάθε κόμβος του οχταδικού δέντρου.



Εικόνα 12: Οχταδικό δέντρο

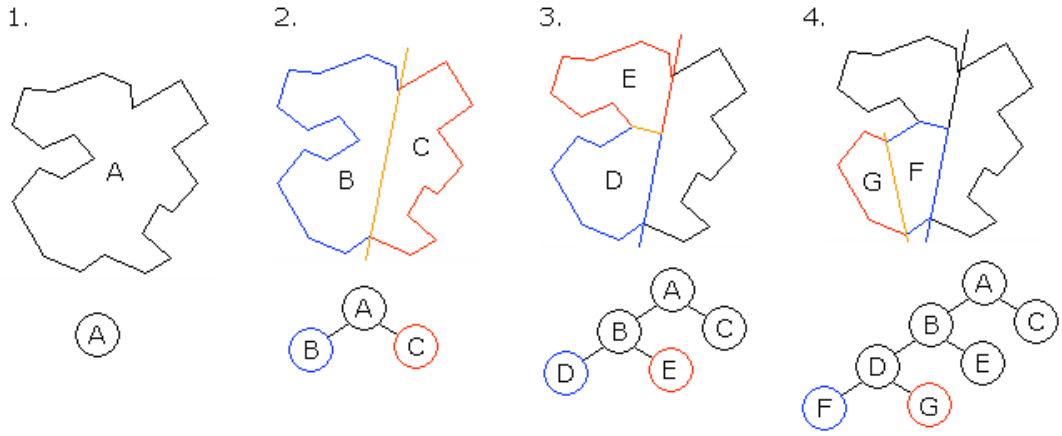
Η διαδικασία απεικόνισης της σκηνής μπορεί να επιταχυνθεί με την χρήση ενός οχταδικού δέντρου. Συγκεκριμένα, ελέγχοντας την θέση του όγκου του οπτικού πεδίου ως προς τους κόμβους του οχταδικού δέντρου, μπορούν να απορριφθούν τα υποδέντρα που είναι εκτός του οπτικού πεδίου. Τα οχταδικά δέντρα χρησιμοποιούνται περισσότερο για την αναπαράσταση εξωτερικών χώρων.

3.5.2 Δέντρα δυαδικού διαχωρισμού χώρου (BSP Trees)

Ο δυαδικός διαχωρισμός χώρου (**BSP**) είναι μια μέθοδος για αναδρομικό διαχωρισμό του χώρου σε κυρτά σύνολα μέσω υπερεπιπέδων. Από τον διαχωρισμό αυτό προκύπτει μία αναπαράσταση της σκηνής μέσω ενός δενδρικής δομής δεδομένων η οποία ονομάζεται δέντρο δυαδικού διαχωρισμού χώρου (**BSP tree**).

Κατά τον δυαδικό διαχωρισμό χώρου, η σκηνή διαχωρίζεται αναδρομικά έως ότου ο διαχωρισμός ικανοποιήσει κάποιες απαιτήσεις ανάλογα με την εφαρμογή. Συνήθως ο διαχωρισμός συνεχίζεται μέχρι οι υποχώροι που θα προκύψουν να περιέχουν μικρό αριθμό πολυγώνων. Ο συνολικός αριθμός πολυγώνων του χώρου θα μεγαλώσει μετά από αυτή την διαδικασία καθώς τα πολύγωνα τα οποία τέμνουν ένα επίπεδο διαχωρισμού θα πρέπει να χωριστούν στα δύο. Επίσης, είναι γενικά επιθυμητό το τελικό δέντρο που θα δημιουργηθεί να είναι σχετικά ισσοροπημένο. Επομένως, το δυσκολότερο μέρος της υλοποίησης της μεθόδου του δυαδικού χωρισμού χώρου είναι η σωστή και αποδοτική δημιουργία του αντίστοιχου BSP δέντρου. Στις τρεις διαστάσεις ο χώρος και τα πολύγωνα χωρίζονται από επίπεδα.

Στην ακόλουθη εικόνα φαίνεται η διαδικασία χωρισμού ενός μη κανονικού πολυγώνου σε μια σειρά από κυρτά πολύγωνα. Κάθε βήμα παράγει πολύγωνα με λιγότερες πλευρές, μέχρι ο αλγόριθμος να καταλήξει στα πολύγωνα G και F τα οποία είναι κυρτά και, σύμφωνα με το κριτήριο που έχει επιλεγεί, δεν απαιτούν περαιτέρω διαχωρισμό. Στην συγκεκριμένη περίπτωση, τα διαχωριστικά επίπεδα που επιλεχθήκαν, δημιουργήθηκαν από τις υπάρχουσες κορυφές του χώρου και δεν διασχίζουν κάποια από τα πολύγωνα του χώρου. Σε αντίθετη περίπτωση, πρέπει τα πολύγωνα που τέμνονται να χωριστούν σε δύο ξεχωριστά πολύγωνα κατά μήκος του επιπέδου, καθώς κάθε υποχώρος που δημιουργείται θα πρέπει να είναι ένα πλήρως ανεξάρτητο αντικείμενο.



Εικόνα 13: Δυαδικός διαχωρισμός χώρου

Σύμφωνα με την παραπάνω εικόνα, το ο χώρος A είναι η ρίζα του δέντρου και περιέχει το σύνολο του αντικειμένου. Το A χωρίζεται σε B και C, το B χωρίζεται σε D και E, και το D χωρίζεται σε G και F όπου και τερματίζει ο αλγόριθμος. Αντίστοιχα θα χωριστεί και ο υποχώρος C.

Καθώς η χρησιμότητα του BSP δέντρου εξαρτάται από το πόσο καλά δημιουργήθηκε, ένας καλός αλγόριθμος είναι απαραίτητος. Οι περισσότεροι αλγόριθμοι δοκιμάζουν αρκετούς διαφορετικούς πιθανούς διαχωρισμούς κάθε υποχώρου μέχρι να καταλήξουν σε ένα σχετικά ισσοροπημένο δέντρο και πολλές φορές είναι δυνατό να απορριφθεί ένα ολόκληρο υποδέντρο που έχει δημιουργηθεί για να χρησιμοποιηθεί μια προηγούμενη εναλλακτική. Κατά συνέπεια η δημιουργία του δέντρου απαιτεί στο σύνολό της αρκετούς υπολογισμούς.

Η δομή του BSP δέντρου επιτρέπει την γρήγορη ταξινόμηση των πολυγώνων του χώρου από το κοντινότερα στα πιο μακρινά. Επιπλέον, κατά την απεικόνιση του χώρου, ολόκληρα υποδέντρα του BSP δέντρου μπορούν να απορριφθούν ελέγχοντας την θέση του όγκου του οπτικού πεδίου ως προς το διαχωριστικό πεδίο που έχει χρησιμοποιηθεί σε κάθε κόμβο. Αν όλες οι κορυφές του όγκου του οπτικού πεδίου είναι από την μια πλευρά του επιπέδου τότε το υποδέντρο που αντιστοιχεί στην άλλη πλευρά του επιπέδου δεν χρειάζεται να απεικονιστεί, επιταχύνοντας έτσι την διαδικασία απεικόνισης.

Η υπολογισμός ενός BSP δέντρου για κάποιο χώρο είναι χρονοβόρος και συνήθως γίνεται εκ των προτέρων. Για το λόγο αυτό, συνήθως χρησιμοποιούνται για να αναπαραστήσουν στατικά τμήματα της σκηνής καθώς, τυχών μεταβολές στην γεωμετρία της σκηνής θα απαιτούσαν επανυπολογισμού του BSP δέντρου. Κινητά αντικείμενα μπορούν να τοποθετηθούν ανάλογα στο BSP δέντρο υπολογίζοντας την θέση τους ως προς τα διαχωριστικά επίπεδα που έχουν χρησιμοποιηθεί σε κάθε κόμβο. Εξαιτίας της μορφής ενός BSP δέντρου και του τρόπου υπολογισμού του, μια τέτοια δομή χρησιμοποιείται συνήθως σε κλειστούς χώρους, όπως είναι το εσωτερικό κτηρίων.

3.5.3 Πύλες (Portals)

Στις μηχανές γραφικών, η χρήση Πυλών (**Portals**) είναι μια μέθοδος για την έλεγχο τον ορατών τμημάτων της σκηνής. Ένα σύστημα πυλών βασίζεται στον διαχωρισμό του χώρου σε περιοχές όπου έχουν κοινή ορατότητα ως προς τις άλλες περιοχές του χώρου. Οι περιοχές αυτές είναι κυρτές πολυγωνικές περιοχές οι οποίες ονομάζονται

τομείς. Γειτονικοί τομείς συνδέονται μέσω διαχωριστικών πολυγώνων τα οποία ονομάζονται πύλες.

Τα συστήματα πυλών χρησιμοποιούνται συνήθως για την απεικόνιση κλειστών χώρων, όπως είναι το εσωτερικό κτηρίων. Στις περιπτώσεις αυτές οι τομείς είναι τα δωμάτια του κτηρίου και οι πύλες με τις οποίες συνδέονται είναι οι πόρτες ή τα παράθυρα τους, τα οποία επιτρέπουν την ορατότητα σε γειτονικούς τομείς. Κατά την διαδικασία της απεικόνισης, σχεδιάζεται ο τομέας στον οποίο βρίσκεται ο παρατηρητής. Αν το οπτικό πεδίο του παρατηρητή περιέχει κάποια πύλη του τομέα, τότε σχεδιάζεται και ο αντίστοιχος γειτονικός τομέας. Ο σχεδιασμός του γειτονικού τομέα απαιτεί μόνο την απεικόνιση του τμήματος το οποίο είναι ορατό μέσα από την πύλη που τον συνδέει.

3.6 Επίπεδα λεπτομέρειας

Το υπολογιστικό κόστος απεικόνισης των αντικειμένων μιας τρισδιάστατης σκηνής είναι γενικά ανάλογο του αριθμού των πολυγώνων από τα οποία αποτελούνται. Για τα αντικείμενα όμως που βρίσκονται μακριά από τον παρατηρητή της σκηνής, η λεπτομέρεια που παρέχει ο μεγάλος αριθμός πολυγώνων δεν είναι ορατή. Ίδιας ή παρόμοιας ποιότητας απεικόνισης μπορεί να επιτευχθεί με σημαντικά μικρότερο αριθμό πολυγώνων. Για το λόγο αυτό, οι μηχανές γραφικών χρησιμοποιούν πολλές φορές τεχνικές οι οποίες μειώνουν τον βαθμό λεπτομέρειας των αντικείμενων όταν το οπτικό αποτέλεσμα δεν παλιώνεται αισθητά. Οι τεχνικές αυτές ονομάζονται επίπεδα λεπτομέρειας (**level of detail**).

Ένας βασικός τρόπος χρήσης της τεχνικής αυτής βασίζεται στον διαχωρισμό του χώρου σε πεπερασμένο αριθμό περιοχών κάθε μια από τις οποίες αντιστοιχίζεται με ένα συγκεκριμένο επίπεδο λεπτομέρειας. Το αποτέλεσμα είναι μια διακριτή ποσότητα επιπέδων λεπτομέρειας και η τεχνική αυτή ονομάζεται διακριτά επίπεδα λεπτομέρειας (discrete level of detail). Συνήθως η τεχνική αυτή χρησιμοποιεί διαφορετικές εκδόσεις των ίδιων αντικειμένων με διαφορετικό βαθμό λεπτομέρειας και χρησιμοποιεί την κατάλληλη έκδοση ανάλογα με την περιοχή στην οποία βρίσκεται το αντικείμενο. Με την τεχνική αυτή δεν είναι εύκολη η ομαλή μετάβαση από το ένα επίπεδο λεπτομέρειας στο άλλο και χρειάζεται προσοχή για να αποφευχθούν απότομες αλλαγές στην απεικόνιση των αντικειμένων.

Ένας εναλλακτικός τρόπος βασίζεται στον επανυπολογισμό του πλέγματος τριγώνων των αντικειμένων ανάλογα με κάποιο μέτρο (συνήθως η απόσταση από τον παρατηρητή). Με αυτή την τεχνική μια διαφορετική έκδοση του πλέγματος παράγεται κάθε φορά με την επιθυμητή ποιότητα και το ανάλογο υπολογιστικό κόστος. Η τεχνική αυτή ονομάζεται συνεχή επίπεδα λεπτομέρειας (continuous level of detail).

3.7 Έλεγχος σύγκρουσης

Σε πολλές εφαρμογές των μηχανών γραφικών είναι χρήσιμο κάθε στιγμή να γνωρίζουμε αν κάποιο αντικείμενο της σκηνής έχει συγκρουστεί με κάποιο άλλο στο χώρο της σκηνής. Το θέμα αυτό δεν σχετίζεται άμεσα με την απεικόνιση των αντικειμένων αλλά είναι απαραίτητο στοιχείο σε πολλές περιπτώσεις όπου είναι επιθυμητή η διαδραστικότητα του χρήστη της εφαρμογής με τον τρισδιάστατο κόσμο. Για παράδειγμα, αν η εφαρμογή περιλαμβάνει την κίνηση ενός ανθρώπινου χαρακτήρα μέσα σε κάποια τρισδιάστατη σκηνή, είναι απαραίτητο να γνωρίζουμε αν

ο χαρακτήρας αυτός έχει συγκρουστεί με κάποιον τοίχο, έτσι ώστε να μην επιτραπεί η διέλευσή του μέσα από αυτόν. Επιπλέον σε μια τρισδιάστατη εξομοίωση, δύο κινούμενα αντικείμενα τα οποία συγκρούονται θα πρέπει να αλλάξουν τα χαρακτηριστικά της κίνησής τους ανάλογα με την σύγκρουση. Το αποτέλεσμα της σύγκρουσης είναι γενικά θέμα φυσικής εξομοίωσης και δεν αποτελεί καθαρά τμήμα των μηχανών γραφικών, και συνήθως υλοποιείται από κάποια εξωτερική μηχανή φυσικής (physics engine). Ο έλεγχος όμως της σύγκρουσης είναι μέρος της μηχανής γραφικών και πολλές από τις δομές που έχουν περιγραφεί παραπάνω χρησιμοποιούνται για τον έλεγχο αυτό.

3.7.1 Έλεγχος σύγκρουσης με περιβάλλοντες όγκους

Γενικά θεωρούμε ότι δύο αντικείμενα συγκρούονται αν οι αναπαραστάσεις τους επικαλύπτονται στο χρόνο. Στην γενική περίπτωση, όπου τα αντικείμενα περιγράφονται από πλέγματα τριγώνων, ο έλεγχος της επικάλυψης απαιτεί έλεγχο της θέσης όλων των πολυγώνων του ενός αντικειμένου σε σχέση με την θέση όλων των πολυγώνων του δεύτερου αντικειμένου. Κάτι τέτοιο γενικά είναι πολύ χρονοβόρο. Για το λόγο αυτό, πριν γίνει ο έλεγχος επικάλυψης των αντικειμένων, γίνεται ο έλεγχος της επικάλυψης των περιβάλλοντων όγκων τους, όπως περιγράφηκε παραπάνω. Αν εξακριβωθεί επικάλυψη των περιβάλλοντων όγκων, μπορεί προαιρετικά, να γίνει έλεγχος επικάλυψης των πολυγώνων των αντικειμένων, όταν είναι επιθυμητή μεγάλη ακρίβεια του σημείου σύγκρουσης των δυο αντικειμένων. Επιπλέον, αν χρησιμοποιηθούν ιεραρχίες περιβάλλοντων όγκων τότε η διαδικασία μπορεί να επιταχυνθεί περισσότερο, ελέγχοντας μόνο τους κόμβους των δύο ιεραρχιών όπου ανιχνεύεται επικάλυψη.

3.7.2 Έλεγχος σύγκρουσης με ιεραρχικές χωρικές δομές

Μια τρισδιάστατη σκηνή γενικά αποτελείται από στατικά αντικείμενα, δηλαδή από αντικείμενα των οποίων η θέση είναι σταθερή στο χρόνο, και από αντικείμενα τα οποία κινούνται μέσα στο χώρο της σκηνής. Έλεγχος επικάλυψης είναι, προφανώς, απαραίτητος μόνο για τα κινούμενα αντικείμενα, σε σχέση με το στατικό τμήμα της σκηνής και με τα υπόλοιπα κινούμενα αντικείμενα. Οι αριθμοί των απαραίτητων ελέγχων μπορούν να μειωθούν με τη χρήση των ιεραρχικών δομών που περιγράφηκαν παραπάνω.

Με την χρήση ιεραρχικών δομών ο χώρος της σκηνής χωρίζεται σε υποχώρους. Σε σχέση με το στατικό τμήμα της σκηνής, ένα κινούμενο αντικείμενο είναι απαραίτητο να ελεγχθεί για σύγκρουση μόνο ως προς τα αντικείμενα των υποχώρων στους οποίους περιέχεται το αντικείμενο ή, ακόμη αποδοτικότερα, ο περιβάλλον όγκος του. Επιπλέον δύο κινούμενα αντικείμενα τα οποία δεν βρίσκονται μέσα στον ίδιο υποχώρο δεν είναι δυνατό να συγκρούονται και ο έλεγχος αυτός μπορεί να αποφευχθεί.

3.7.3 Έλεγχος εκ των υστέρων

Ο απλούστερος τρόπος ελέγχου σύγκρουσης των κινούμενων αντικειμένων είναι να υπολογιστεί η θέση τους για την συγκεκριμένη επιθυμητή στιγμή και να ελεγχθεί, για εκείνη την κατάσταση, η πιθανή επικάλυψή τους. Είναι πιθανό όμως για δύο αντικείμενα Α, Β, όπου το Α διανύει κάθε στιγμή απόσταση μεγαλύτερη από το μέγεθος του Β, την μια στιγμή το Α να βρίσκεται από την μια πλευρά του Β και την επόμενη από την άλλη πλευρά. Σε αυτή την περίπτωση αντί της επικάλυψης του όγκου του Α με τον όγκο του Β, πρέπει να ελεγχθεί η προέκταση του όγκου Α, η οποία περιλαμβάνει όλες τις θέσεις που έχει καταλάβει ο όγκος Α από την μια

χρονική στιγμή στην άλλη. Επιπλέον, αν ανιχνευθεί επικάλυψη, τότε τα αντικείμενα θα έχουν μη αποδεκτή θέση στο χώρο καθώς τμήμα του ενός θα βρίσκεται μέσα στον όγκο του άλλου. Η θέση τους θα πρέπει να διορθωθεί κατάλληλα πριν την απεικόνισή τους.

4. Υλοποίηση

Στα πλαίσια αυτής της εργασίας έχει γίνει η υλοποίηση μιας μηχανής γραφικών η οποία χρησιμοποιεί τις βασικότερες από τις τεχνικές οι οποίες αναφέρθηκαν παραπάνω.

4.1 Περιβάλλον ανάπτυξης

Η ανάπτυξη της μηχανής γραφικών έγινε με τέτοιο τρόπο έτσι ώστε να είναι όσο το δυνατό περισσότερο ανεξάρτητη ως προς την πλατφόρμα εκτέλεσης. Η ανάπτυξη έγινε στο λειτουργικό σύστημα Linux, έτσι ώστε να είναι υπάρχει σχετικά άμεση συμβατότητα με άλλα περιβάλλοντα Unix (πχ. Mac OS X), ενώ χρησιμοποιήθηκαν εξωτερικές βιβλιοθήκες οι οποίες είναι συμβατές με όλα τα γνωστά και ευρέως διαδεδομένα λειτουργικά συστήματα. Η μεταφορά της μηχανής σε περιβάλλον Microsoft Windows μπορεί απαιτεί μικρές αλλαγές σε θέματα διασύνδεσης με το λειτουργικό σύστημα, ο βασικός κορμός της μηχανής όμως είναι γενικά συμβατός και ανεξάρτητος πλατφόρμας.

4.1.1 Γλώσσα και προγραμματισμού

Η πηγαίος κώδικας της μηχανής γραφικών έχει γραφτεί εξ' ολοκλήρου σε γλώσσα προγραμματισμού **C++**. Μια μηχανή γραφικών αποτελεί ένα σύστημα με σύνθετη αρχιτεκτονική λογισμικού, Επιπλέον, εξ' αιτίας των απαιτήσεων για γρήγορους υπολογισμούς ανά εικόνα, η χρήση ενδιάμεσων γλωσσών, όπως Perl ή Python δεν ενδείκνυται. Η χρήση περισσότερο δομημένων γλωσσών προγραμματισμού όπως είναι η C# ή η Java εισάγουν γενικά τον παράγοντα της εικονικής μηχανής (virtual machine) ο οποίος μειώνει την ταχύτητα εκτέλεσης της μηχανής ως προς τις εφαρμογές οι οποίες εκτελούνται κατευθείαν στον επεξεργαστή. Η C++ συνδυάζει την απόδοση της C, ως προς την ταχύτητα, με τις δομές μίας αντικειμενοστραφούς γλώσσας προγραμματισμού, οι οποίες είναι αναγκαίες για μια εφαρμογή της πολυπλοκότητας μιας μηχανής γραφικών.

Η ανάπτυξη έχει γίνει με βάση το πρότυπο ANSI C++ έτσι ώστε να υπάρχει συμβατότητα με όλες τις πλατφόρμες που το υποστηρίζουν. Επιπλέον έχει χρησιμοποιηθεί η βιβλιοθήκη **STL** (Standard Template Library) η οποία αποτελεί μέρος του προτύπου της ANSI C++ και παρέχει υλοποιήσεις ενός συνόλου απαραίτητων δομών όπως είναι ο στοίβες, οι λίστες και οι ουρές, καθώς και σχετικούς αλγόριθμους, όπως πχ. η ταξινόμηση των στοιχείων τους. Η μηχανή γραφικών έχει δοκιμαστεί σε περιβάλλον **Linux** και **Mac OS X** με την χρήση του μεταγλωττιστή **GNU gcc**, ενώ χρησιμοποιήθηκε το σύστημα **autoconf** το οποίο αυτοματοποιεί την διαδικασία της μεταγλώττισης ελέγχοντας όλες τις εξαρτήσεις του πηγαίου κώδικα με εξωτερικές βιβλιοθήκες.

4.1.2 Σύστημα απεικόνισης και διασύνδεση με το λειτουργικό σύστημα.

Η λειτουργία της μηχανής γραφικών έχει βασιστεί στο σύστημα απεικόνισης **OpenGL**. Η επιλογή αυτή έγινε για λόγους συμβατότητας με όλα τις πλατφόρμες καθώς υπάρχει υλοποίησή της για κάθε ευρέως διαδεδομένη πλατφόρμα, σε αντίθεση με το Direct3D το οποίο υποστηρίζεται μόνο από τα λειτουργικά συστήματα τύπου

Microsoft Windows. Η βιβλιοθήκη OpenGL συνοδεύεται από την βιβλιοθήκη **GLU** (OpenGL Utility Library) η οποία υλοποιεί και αυτοματοποιεί υψηλότερου επιπέδου Λειτουργίες οι οποίες είναι συχνά χρήσιμες για την μηχανή γραφικών, όπως για παράδειγμα η κατασκευή διαφορετικών εκδόσεων των εικόνων υφών (texture mipmaps) ή η αυτόματη παραγωγή απλών γεωμετρικών αντικειμένων (κύβοι, σφαίρες κτλ)

Η μηχανή γραφικών απαιτεί ένα τρόπο διασύνδεσης του συστήματος απεικόνισης με το λειτουργικό σύστημα στο οποίο εκτελείται, έτσι ώστε να είναι δυνατή η εμφάνιση της εικόνας σε κάποιο παράθυρο του λειτουργικού. Επιπλέον μια διαδραστική μηχανή γραφικών απαιτεί και διασύνδεση με συσκευές όπως το πληκτρολόγιο και το ποντίκι. Την διασύνδεση του λειτουργικού συστήματος και των συσκευών με το σύστημα απεικόνισης αναλαμβάνει η βιβλιοθήκη **libSDL** (Simple DirectMedia Library). Η libSDL είναι συμβατή με όλες τα ευρέως διαδεδομένα λειτουργικά συστήματα και δίνει πρόσβαση σε συσκευές όπως το πληκτρολόγιο, το ποντίκι καθώς και στην κάρτα γραφικών μέσω της OpenGL.

4.1.3 Εξωτερικές βιβλιοθήκες

Η μηχανή γραφικών που αναπτύχθηκε χρησιμοποιεί μια επέκταση της libSDL, την **SDL_image**, η οποία αναλαμβάνει την ανάγνωση αρχείων εικόνων των περισσότερο γνωστών μορφών (BMP, JPG, GIF, PNG κα) με τρόπο συμβατό με την βιβλιοθήκη libSDL. Η χρήση της είναι απαραίτητη για την χρήση των υφών οι οποίες βασίζονται σε εικόνες αποθηκευμένες στον δίσκο.

Η βιβλιοθήκη **lib3ds** έχει χρησιμοποιηθεί για την ανάγνωση αρχείων τύπου **3DS** τα οποία είναι βασισμένα στην εφαρμογή τρισδιάστατης μοντελοποίησης 3D Studio και περιέχουν γεωμετρικές αναπαραστάσεις αντικειμένων σε μορφή πλέγματος τριγώνων. Η μηχανή γραφικών υποστηρίζει δύο ακόμα τύπους αρχείων αντικειμένων σε μορφή πλέγματος τριγώνων, τα **LWO** και **MD2**. Ο χειρισμός αυτών των αρχείων δεν βασίζεται σε κάποια εξωτερική βιβλιοθήκη, αντ' αυτού η υλοποίηση του αποτελεί μέρος της μηχανής γραφικών και έχει γίνει στα πλαίσια ανάπτυξης αυτής της εργασίας.

Καθώς η μηχανή γραφικών είναι δυνατό να αναπαραστήσει σύνθετες τρισδιάστατες σκηνές, τα δεδομένα που τις συνθέτουν απαιτούν ένα τρόπο οργάνωσής. Έχει αναπτυχθεί ένα σύστημα οργάνωσης των δεδομένων της σκηνής καθώς και των παραμέτρων λειτουργίας της μηχανής γραφικών, βασισμένο στην γλώσσα περιγραφής **XML** (Extensible Markup Language). Η ανάγνωση αρχείων XML έχει γίνει με την χρήση της εξωτερικής βιβλιοθήκης **libxml2**. Η βιβλιοθήκη αυτή είναι επίσης συμβατή με όλα τα περισσότερα λειτουργικά συστήματα.

4.2 Μαθηματικά εργαλεία

Στα πλαίσια ανάπτυξης της μηχανής γραφικών έχουν υλοποιηθεί ένα σύνολο από χρήσιμα μαθηματικά εργαλεία.

4.2.1 Διανύσματα

Ένα διάνυσμα στις τρεις διαστάσεις, και οι σχετικές πράξεις με αυτό, υλοποιείται από την κλάση **Engine3D::Math::Vector3D**. Η κλάση αυτή περιέχει τις πράξεις πρόσθεσης και αφαίρεσης διανυσμάτων, πολλαπλασιασμού με αριθμό, και το εσωτερικό και εξωτερικό γινόμενο διανυσμάτων. Παρέχονται επίσης συναρτήσεις υπολογισμού του μέτρου του διανύσματος καθώς και του τετραγώνου του μέτρου για

τις περιπτώσεις όπου αυτό είναι χρήσιμο. Ένα σημείο στον χώρο μπορεί επίσης να αναπαρασταθεί με την κλάση αυτή καθώς ένα διάνυσμα από την αρχή των αξόνων μέχρι το σημείο, έχει τις ίδιες συντεταγμένες (x, y, z) με το σημείο αυτό.

4.2.2 Πράξεις πινάκων

Ένας πίνακας 3x3 υλοποιείται μέσω της κλάσης **Engine3D::Math:Matrix3D**. Η κλάση αυτή περιέχει τις πράξεις πρόσθεσης και αφαίρεσης πινάκων, πολλαπλασιασμού με αριθμό, πολλαπλασιασμού πινάκων, υπολογισμό του ανάστροφου πίνακα και υπολογισμό της ορίζουσας του πίνακα. Περιέχει επίσης συναρτήσεις οι οποίες επιστρέφουν τον τρισδιάστατο πίνακα μετασχηματισμού ο οποίος αντιστοιχεί σε μετατόπιση κατά ένα δοσμένο διάνυσμα.

4.2.3 Επίπεδο

Η κλάση **Engine3D::Math::Plane** υλοποιεί ένα τρισδιάστατο επίπεδο. Η αναπαράσταση του επιπέδου γίνεται σύμφωνα με την εξίσωση του:

$$Ax + By + Cy + D = 0$$

όπου (x, y, z) οι συντεταγμένες των σημείων που ανήκουν στο επίπεδο. Η κλάση αυτή περιέχει επίσης συνάρτηση η οποία υπολογίζει την απόσταση ενός σημείου από το επίπεδο αυτό.

4.2.4 Τετράδες

Μια τετράδα υλοποιείται μέσω της κλάσης **Engine3D::Math::Quaternion**. Η κλάση αυτή επιτρέπει την δημιουργία μιας τετράδας από τις τέσσερεις παραμέτρους τις αλλά επιτρέπει και την παραγωγή τετράδων οι οποίες αντιστοιχούν σε περιστροφές γωνίας θ γύρω από κάποιον άξονα περιστροφής. Υλοποιούνται οι πράξεις πρόσθεσης και αφαίρεσης τετράδων, εσωτερικού γινομένου τετράδων, του λογαρίθμου τετράδων, υπολογισμού του μέτρου, των αντίστροφων και των συζυγών τετράδων. Υλοποιούνται, επίσης, συναρτήσεις οι οποίες περιστρέφουν ένα δοσμένο διάνυσμα με βάση μια τετράδα, καθώς και συναρτήσεις οι οποίες επιστρέφουν για μια τετράδα, την γωνιά και τον άξονα περιστροφής στην οποία αντιστοιχεί ή τον αντίστοιχο πίνακα μετασχηματισμού περιστροφής. Τέλος, υλοποιείται μέσω αντίστοιχης συνάρτησης η παρεμβολή SLERP.

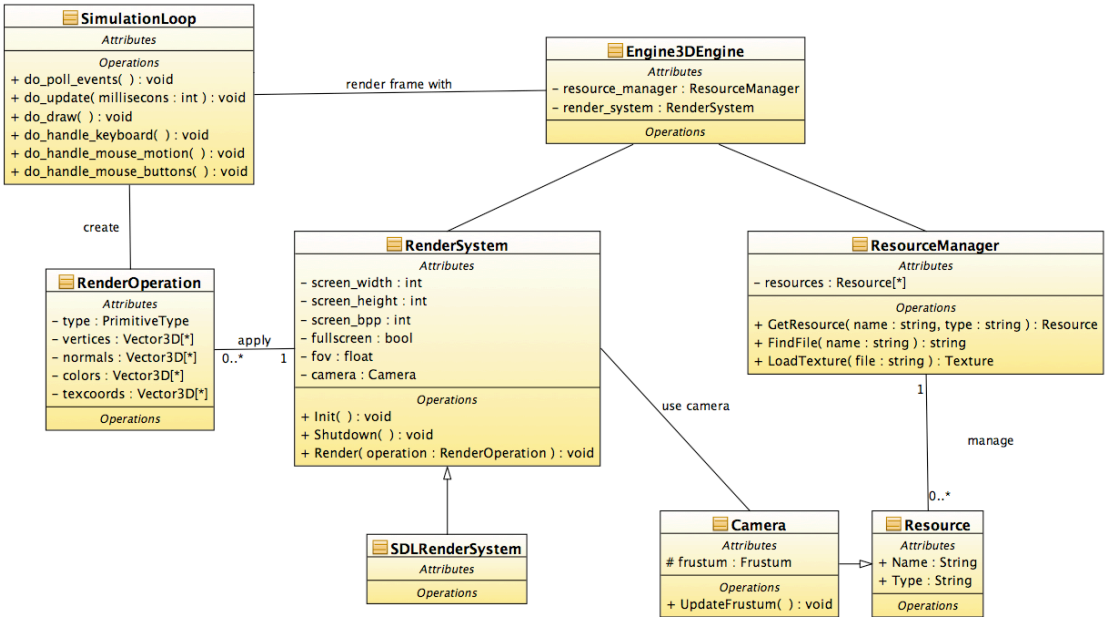
4.2.5 Γενικά μαθηματικά εργαλεία

Ένα σύνολο βασικών μαθηματικών εργαλείων τα οποία είναι χρήσιμα σε πολλές διεργασίες της μηχανής γραφικών περιέχεται στον χώρο ονοματοδοσίας της C++ (namespace) με όνομα **Engine3D::Math**. Ο χώρος αυτός περιέχει συναρτήσεις υπολογισμού τυχαίων αριθμών, μετατροπής γωνίας από μοίρες σε ακτίνια, υπολογισμού οριζουσών πινάκων 2x2, απόστασης δύο σημείων, των καθέτων διανυσμάτων των πλευρών ενός πλέγματος τριγώνων καθώς και ελέγχους συγραμμικότητας τριών σημείων.

4.3 Αρχιτεκτονική

Η αρχιτεκτονική της μηχανής γραφικών βασίζεται στην κλάσης **Engine3D::Engine3D** η οποία χωρίζεται σε δυο υποσυστήματα. Το πρώτο υποσύστημα σύστημα εξυπηρετεί την επικοινωνία της μηχανής με το σύστημα απεικόνισης (OpenGL μέσω SDL) και το δεύτερο υποσύστημα αναλαμβάνει την διαχείριση των διαθέσιμων πόρων της μηχανής γραφικών (αρχεία εικόνων, αρχεία μοντέλων και περιγραφής σκηνών). Επιπλέον, ένα ξεχωριστό σύστημα, το οποίο

ονομάστηκε βρόχος εξομοίωσης, αναλαμβάνει την ανανέωση της κατάστασης της σκηνής σύμφωνα με την παράμετρο του χρόνου και τον συντονισμό με την μηχανή γραφικών για το σχεδιασμό της εικόνας κάθε στιγμή. Η κλάση αυτή, καθώς και οι κλάσεις των δύο υποσυστημάτων, πρέπει να είναι μοναδική κατά την εκτέλεση της εφαρμογής και γι' αυτό το λόγο οι τρεις αυτές κλάσεις ακολουθούν το πρότυπο σχεδιασμού Singleton κατά το οποίο οι κλάσεις αυτές δηλώνονται *static* και η πρόσβαση σε αυτές γίνεται μόνο μέσα από συγκεκριμένες μεθόδους.



Εικόνα 14: Αρχιτεκτονική μηχανής γραφικών

4.3.1 Επικοινωνία με το σύστημα απεικόνισης

Η διεπαφή με το σύστημα απεικόνισης, για λόγους αφαίρεσης, γίνεται μέσω της κλάσης **Engine3D::SDLRenderSystem**. Η κλάση αυτή είναι η μοναδική η οποία περιέχει στην υλοποίησή της κλήσεις προς τις βιβλιοθήκες OpenGl και libSDL. Όλες οι υπόλοιπες κλάσεις της μηχανής γραφικών δεν εξαρτώνται άμεσα από εξωτερικές βιβλιοθήκες, έτσι ώστε οι αλλαγές σε αυτές να επηρεάζουν μόνο την κλάση **SDLRenderSystem**. Οι βασικές παράμετροι οι οποίες καθορίζουν την μορφή της εικόνας που παράγει η OpenGL:

- int **screen_width, screen_height, screen_bpp**
  Αντιστοιχούν στο πλάτος και στο ύφος της εικόνας (**screen_width** και **screen_height** αντίστοιχα) και στον αριθμό των bits που χρησιμοποιούνται για το χρώμα κάθε pixel της εικόνας.

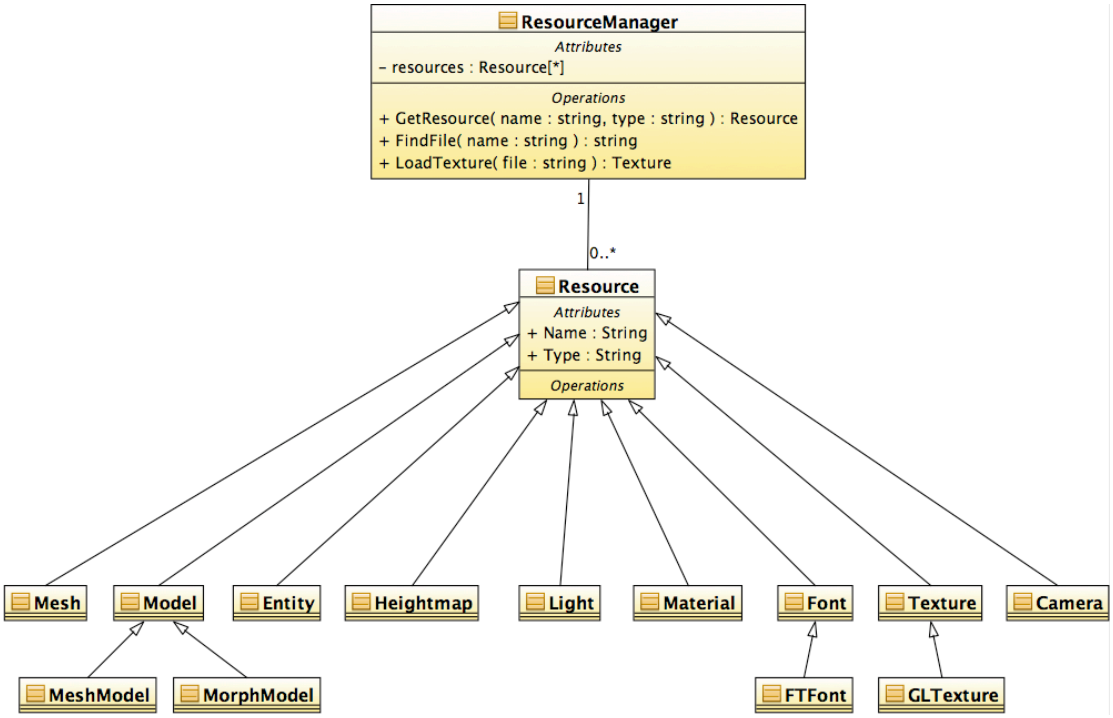
- float **fov**
  Η γωνία του οπτικού πεδίου. Η τιμή αυτή επηρεάζει τον όγκο του οπτικού πεδίου ο οποίος θα πρέπει να επανυπολογιστεί με την αλλαγή της. Η τιμή αυτή όμως γενικά είναι σταθερή κατά την εκτέλεση της λειτουργίας της μηχανής γραφικών.

- bool **fullscreen**
  Επιλογή της εμφάνισης της εικόνας σε παράθυρο του λειτουργικού συστήματος ή στο σύνολο της οθόνης.

Η κλάση **SDLRenderSystem** αναλαμβάνει την διαχείριση των πινάκων μετασχηματισμού που χρησιμοποιεί η OpenGL καθώς και την δημιουργία των κατάλληλων πινάκων μετασχηματισμού που απαιτούνται από συγκεκριμένες περιπτώσεις, όπως είναι ο μετασχηματισμός της ορθογραφικής προβολής και οι απαραίτητοι μετασχηματισμοί για την τοποθέτηση του παρατηρητή μέσα στην σκηνή. Επιπλέον, η κλάση αυτή αναλαμβάνει την διαχείριση των χρωμάτων και των εικόνων υφών που χρησιμοποιεί η OpenGL κάθε φορά.

Τέλος, η κλάση αυτή αναλαμβάνει τον σχεδιασμό των τριγώνων των αντικειμένων μέσω της OpenGL. Η λειτουργία αυτή εκτελείται μέσω της μεθόδου **Render** με την χρήση του αντικειμένου **Engine3D::RenderOperation** το οποίο ομαδοποιεί τις κορυφές, τις πλευρές, τις υφές και τις αντίστοιχες συντεταγμένες υφών ανά κορυφή που θα σχεδιαστούν, και το οποίο χρησιμοποιούν τα υπόλοιπα αντικείμενα της μηχανής γραφικών για να υποδηλώσουν τον τρόπο απεικόνισης των αντικειμένων.

4.3.2 Σύστημα διαχείρισης πόρων



Εικόνα 15: Σύστημα διαχείρισης πόρων

Η μηχανή γραφικών μπορεί να χρησιμοποιεί ένα σύνολο από εικόνες, μοντέλα ή άλλους πόρους τα οποία είναι αποθηκευμένα στον δίσκο και είναι γενικά προσβάσιμα από τις κλάσσεις της μηχανής, την διαχείρισή των οποίων αναλαμβάνει η κλάση **Engine3D::ResourceManager**. Η κλάση αυτή δέχεται ένα σύνολο φακέλους στον δίσκο του συστήματος στους οποίους μπορεί να αναζητήσει τα αρχεία των πόρων,

και αποθηκεύει τους πόρους μέσω υποκλάσεων οι οποίες προέρχονται από την κλάση **Engine3D::Resource**. Κάθε αντικείμενο που αποθηκεύεται στην μνήμη από την κλάση **ResourceManager** αντιστοιχείται με ένα όνομα με το οποίο μπορεί να ανακληθεί. Τα βασικά διαθέσιμα είδη πόρων είναι τα εξής:

- Υλικά (**Materials**)
  Υλοποιούνται από την κλάση **Engine3D::Material**. Περιέχουν περιγραφή των παραμέτρων που καθορίζουν την εμφάνιση των κορυφών από την OpenGL οι οποίες είναι το χρώμα και η εικόνα υφής τους.

- Εικόνες υφών (**Textures**)
  Υλοποιείται από την κλάση **Engine3D::Texture** και αντιστοιχεί σε μία εικόνα υφής. Η κλάση **ResourceManager** χρησιμοποιεί την εξωτερική βιβλιοθήκη SDL_image για την ανάγνωση αρχείων εικόνων και την δημιουργία αντίστοιχων υφών μέσω της μεθόδου **LoadTexture**.

- Πλέγματα τριγώνων (**Meshes**)
  Αντιστοιχούν σε πλέγματα τριγώνων τα οποία αναπαριστούν αντικείμενα και μπορεί να είναι αποθηκευμένα σε αρχεία τύπου 3DS, LWO ή MD2. Η υλοποίησή τους γίνεται από την κλάση **Engine3D::Mesh** η οποία περιγράφεται αναλυτικά παρακάτω.

4.3.3 Βρόχος εξομοίωσης

Το υποσύστημα αυτό αναλαμβάνει την εκτέλεση ενός βρόχου ο οποίος ανανεώνει κάθε στιγμή την κατάσταση της σκηνής μέσω της κλάσης **Engine3D::SimulationLoop** και της μεθόδου της **Run**. Η εκτέλεση της μεθόδου αυτής ισοδυναμεί με συνεχή εκτέλεση του βρόχου εξομοίωσης έως ότου κληθεί η μέθοδος **Stop** ή **Pause** η οποία σταματάει ή παγώνει την εκτέλεση αντίστοιχα. Ο βρόχος εξομοίωσης ελέγχει το σύστημα για τυχών είσοδο από το πληκτρολόγιο ή το ποντίκι, ανανεώνει την κατάσταση της σκηνής ανάλογα με τον χρόνο που έχει περάσει από την τελευταία εκτέλεση και στέλνει στο σύστημα απεικόνισης την σκηνή προς σχεδιασμό.

Οι εφαρμογές οι οποίες χρησιμοποιούν την μηχανή γραφικών η οποία έχει αναπτυχθεί θα πρέπει να υλοποιήσουν μία υποκλάση της **SimulationLoop** η οποία θα παραμετροποιεί την λειτουργία της ανάλογα με τις ανάγκες τους. Συγκεκριμένα, θα πρέπει να υλοποιηθούν οι ακόλουθες μέθοδοι της κλάσης:

- int **do_update**(int *msec*)

  Αναλαμβάνει να ανανεώσει την κατάσταση της σκηνής η οποία αντιστοιχεί στο πέρασμα *msec* χιλιοστών του δευτερολέπτου.

- int **do_draw**(void)

  Αναλαμβάνει τον σχεδιασμό των αντικειμένων της σκηνής μέσω της κλάσης **SDLRenderSystem**. Εδώ μπορούν να εφαρμοστούν οι κατάλληλες τεχνικές για τον αποκλεισμό των αντικειμένων τα οποία δεν θα είναι ορατά στην τελική εικόνα.

- int **do_handle_keyboard**(SDL_KeyboardEvent *);

- int **do_handle_mouse_motion**(SDL_MouseMotionEvent *);
- int **do_handle_mouse_buttons**(SDL_MouseButtonEvent *);

Οι τρεις αυτές μέθοδοι αναλαμβάνουν την απόκριση της εφαρμογές στην είσοδο του χρήση μέσω του πληκτρολογίου, μέσω της κίνησης του ποντικιού και μέσω των πλήκτρων του ποντικιού. Η είσοδος του χρήστη παρέχεται από την βιβλιοθήκη libSDL μέσω των αντικειμένων SDL_KeyboardEvent, SDL_MouseMotionEvent και SDL_MouseButtonEvent.

4.4 Αντικείμενα παραμέτρων και ελέγχου

Η μηχανή γραφικών χρησιμοποιεί ένα σύνολο από αντικείμενα τα οποία καθορίζουν παραμέτρους της σκηνής ή περιγράφουν στοιχειώδεις οντότητες απαραίτητες για την λειτουργία της.

4.4.1 Χρώματα και υλικά

Τα αντικείμενα της σκηνής πέρα από την γεωμετρική περιγραφή τους, χρησιμοποιούν ένα σύνολο από κλάσσεις για να περιγράψουν παραμέτρους όπως είναι το χρώμα ή οι εικόνες υφών. Τα αντικείμενα της κλάσης **Engine3D:Color** αντιστοιχούν σε κάποιο χρώμα με βάση την RGB τιμή τους. Οι τιμές αυτές μπορούν να πολλαπλασιαστούν με κάποιο αριθμό έτσι ώστε να προκύψει παρόμοιο χρώμα με διαφορετική ένταση. Η κλάση αυτή παρέχει επίσης προκαθορισμένα αντικείμενα χρωμάτων για χρώματα με συχνή χρήση (άσπρο, μαύρο κτλ).

Τα αντικείμενα της κλάσης **Engine3D::Texture** αντιστοιχούν σε εικόνες υφών αποθηκευμένες στην μνήμη της κάρτας γραφικών. Η OpenGL αντιστοιχεί μια εικόνα υφής στην μνήμη με ένα αύξοντα αριθμό ο οποίος παράγεται κατά την δημιουργία της εικόνας υφής αποθηκεύεται στο αντικείμενο **Texture** για εύκολη αναφορά του. Η μηχανή γραφικών υποστηρίζει την δημιουργία εικόνων υφών από κάποιο αρχείο εικόνας αποθηκευμένο στο δίσκο, μέσω της κλάσης **ResourceManager** και τις μεθόδου **LoadTexture**. Η μέθοδος αυτή καλεί την αντιστοιχεί μέθοδο της κλάσης **SDLRenderSystem**, για την ανάγνωση των εικόνων με την χρήση της βιβλιοθήκης SDL_image, και αποθηκεύει το αντικείμενο **Texture** που δημιουργήθηκε, στις λίστες πόρων. Η μέθοδος αυτή παράγει αυτόματα και τα επίπεδα των εικόνων υφών (mipmaps) μέσω της βιβλιοθήκης GLU.

Η κλάση **Engine3D::Material** συνδυάζει τις παραπάνω κλάσσεις για να περιγράψει ένα συγκεκριμένο υλικό ενός αντικειμένου της σκηνής, ή με τμήματός του. Συγκεκριμένα, η κλάση **Material** περιέχει ένα αντικείμενο της κλάσης **Texture** για τον καθορισμό της εικόνας υφής του αντικειμένου, και τέσσερα αντικείμενα της κλάσης **Color**: *ambient*, *diffuse*, *specular*, *emission*. Τα χρώματα αυτά καθορίζουν το τελικό χρώμα ενός αντικειμένου μέσω της διαδικασίας φωτισμού της OpenGL.

4.4.3 Μη ορατά αντικείμενα σκηνής

Η μηχανή γραφικών χρησιμοποιεί κάποια αντικείμενα τα οποία δεν είναι ορατά στην σκηνή αλλά καθορίζουν την εμφάνιση της σκηνής επηρεάζοντας παραμέτρους όπως είναι ο φωτισμός και η θέση του παρατηρητή

Η κλάση **Engine3D::Light** αντιστοιχεί σε μια πηγή φωτός τοποθετημένη στην σκηνή. Αποτελείται από ένα αντικείμενο **Vector3D** το οποίο αντιστοιχεί στην θέση της πηγής φωτός στον χώρο, και από τρία αντικείμενα **Color**: *ambient*, *diffuse*,

specular. Το αντικείμενο αυτό χρησιμοποιεί την OpenGL για να φωτίσει την σκηνή από μια επιθυμητή θέση.

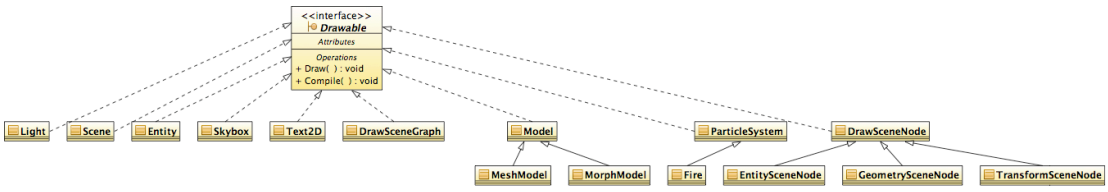
Η κλάση **Engine3D::Camera** αντιστοιχεί στον παρατηρητή της σκηνής και συγκεκριμένα στην θέση του στο χώρο και στον προσανατολισμό του. Η OpenGL θεωρεί ότι η θέση του παρατηρητή βρίσκεται πάντα στην αρχή των αξόνων του συστήματος αναφοράς της σκηνής. Το αντικείμενο **Camera** χρησιμοποιεί τους αντίστροφους μετασχηματισμούς μεταφοράς και περιστροφής, ως προς την θέση και τον προσανατολισμό του αντίστοιχα, για να τοποθετήσει την σκηνή στην επιθυμητή θέση ως προς τον παρατηρητή δημιουργώντας την ψευδαίσθηση της κίνησης του παρατηρητή μέσα στην σκηνή.

Μέρος της κλάσης **Camera** είναι η κλάση **Engine3D::Frustum** η οποία αντιστοιχεί στο οπτικό πεδίο του παρατηρητή. Η θέση του οπτικού πεδίου είναι σχετική της θέσης του αντικειμένου **Camera**. Το αντικείμενο αυτό αποτελείται από έξι επίπεδα τα σχηματίζουν τον όγκο πυραμίδας με κομμένη κορυφή. Οι θέσεις των επιπέδων αυτών υπολογίζονται μέσω του αντικειμένου **Camera** από τον πίνακα μετασχηματισμού που χρησιμοποιεί η OpenGL. Ο πίνακας αυτός μπορεί να καθοριστεί από το αντικείμενο **SDLRenderSystem** καθορίζοντας την παράμετρο της γωνίας του οπτικού πεδίου (FOV) επηρεάζοντας έτσι την θέση των έξι επιπέδων. Η κλάση **Frustum** παρέχει επίσης συναρτήσεις οι οποίες ελέγχουν αν ένα σημείο ή κάποιος περιβάλλον όγκος είναι στο εσωτερικό του οπτικού πεδίου, ελέγχοντας την θέση του ως προς τα έξι επίπεδα. Με τον τρόπο αυτό μπορούν κατά την απεικόνιση να απορριφθούν αντικείμενα εκτός του οπτικού πεδίου, επιταχύνοντας την διαδικασία απεικόνισης.

4.5 Αναπαραστάσεις γεωμετρικών αντικειμένων

Ένα σύνολο από κλάσεις χρησιμοποιούνται για να περιγράψουν τα γεωμετρικά αντικείμενα που ανήκουν στην σκηνή. Τα αντικείμενα αυτά υπακούν σε κάποιες διεπαφές οι οποίες περιγράφουν κοινές Λειτουργίες τους, ανάλογα με το είδος του αντικειμένου.

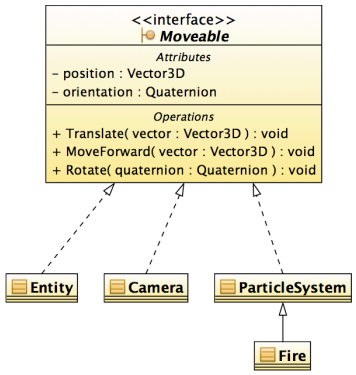
4.5.1 Διεπαφές λειτουργιών



Εικόνα 16: Διεπαφή Drawable

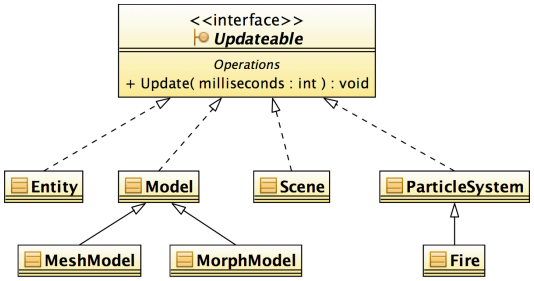
Η διεπαφή **Engine3D::Drawable** χρησιμοποιείται για αντικείμενα τα οποία μπορούν να απεικονιστούν με κάποιο τρόπο από την OpenGL. Οι μέθοδοι που απαιτεί αυτή η διεπαφή είναι οι **Draw** και **Compile**. Η μέθοδος **Compile** χρησιμοποιείται από αντικείμενα για την αρχικοποίηση παραμέτρων τους οι οποίες επηρεάζουν την εμφάνισή τους. Η μέθοδος **Draw** χρησιμοποιείται για κάθε αντικείμενο από τον βρόχο εξομοίωσης **SimulationLoop** κατά την εκτέλεση της μεθόδου **do_draw** για τον σχεδιασμό κάθε αντικειμένου ανά εικόνα. Το αντικείμενο **Light,** το οποίο αναφέρθηκε προηγουμένως υπακούει επίσης στην διεπαφή **Drawable** καθώς επηρεάζει τον φωτισμό της σκηνής και κατά συνέπεια την τελική εικόνα.

Τα αντικείμενα τα οποία υπακούν στην διεπαφή **Drawable** χρησιμοποιούν συχνά μεθόδους οι οποίες ανήκουν στο χώρο ονοματοδοσίας **Engine3D::Draw**. Μέσω των μεθόδων αυτών μπορούν να σχεδιαστούν περιγράμματα αντικειμένων για τον οπτικό έλεγχο της θέσης και του μεγέθους δομών όπως είναι περιβάλλοντα κουτιά και περιβάλλουσες σφαίρες. Επιπλέον ο χώρος αυτός περιέχει μέθοδο η οποία επιτρέπει την σχεδίαση δυσδιάστατων εικόνων οι οποίες έχουν πάντα προσανατολισμό παράλληλο με την προσανατολισμό του παρατηρητή.



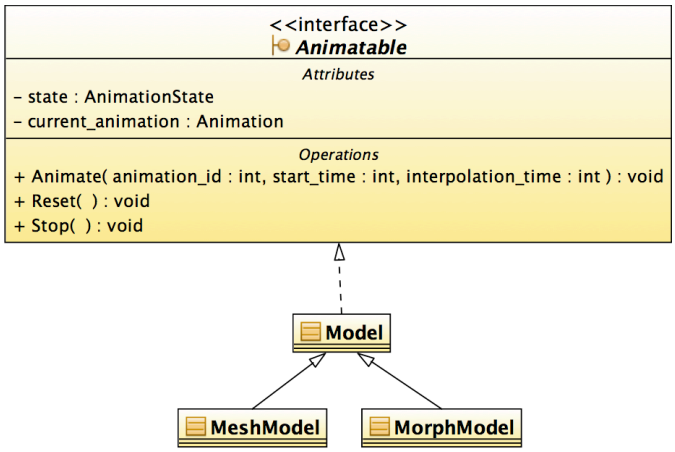
Εικόνα 17: Διεπαφή Moveable

Η διεπαφή **Engine3D::Moveable** χρησιμοποιείται από αντικείμενα τα οποία έχουν ορισμένη θέση και προσανατολισμό στο χώρο της σκηνής. Ένα αντικείμενο **Vector3D** χρησιμοποιείται για την θέση του αντικειμένου και ένα αντικείμενο **Quaternion** για τον προσανατολισμό του. Η διεπαφή αυτή παρέχει τους απαραίτητους μετασχηματισμούς για την τοποθέτηση του αντικειμένου στην σκηνή καθώς και μεθόδους για την μετακίνηση ή την περιστροφή του. Επίσης, παρέχεται μέθοδος η οποία μετακινεί το αντικείμενο ως προς τον άξονα προς τον οποίο είναι προσανατολισμένο το αντικείμενο.



Εικόνα 18: Διεπαφή Updateable

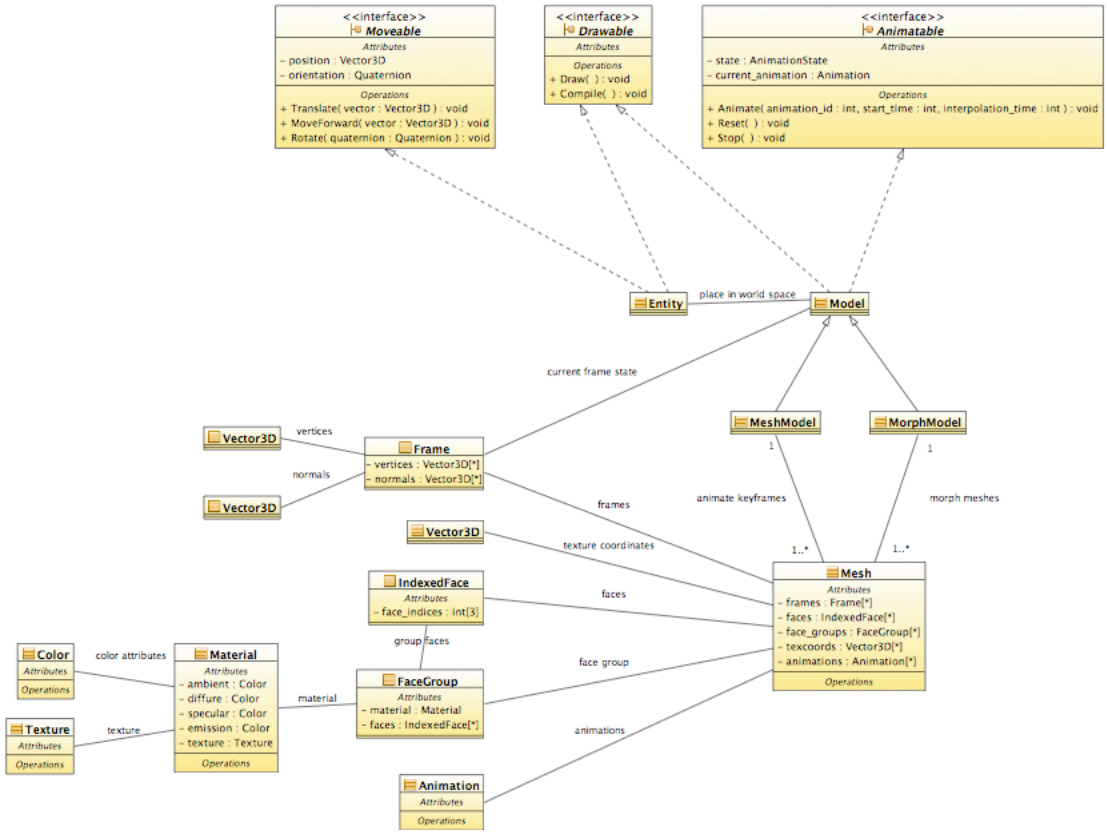
Η διεπαφή **Engine3D::Updateable** χρησιμοποιείται για αντικείμενα των οποίων οι παράμετροι μεταβάλλονται με τον χρόνο. Η βασική μέθοδος της διεπαφής αυτής είναι η **Update** η οποία δέχεται ως παράμετρο τα χιλιοστά του δευτερολέπτου που έχουν περάσει από την τελευταία ενημέρωση του αντικειμένου. Τα αντικείμενα που υπακούν σε αυτή την διεπαφή χρησιμοποιούνται από τον βρόχο εξομοίωσης **SimulationLoop** κατά την εκτέλεση της μεθόδου **do_update** η οποία ανανεώνει την κατάσταση των αντικειμένων.



Εικόνα 19: Διεπαφή Animatable

Η διεπαφή **Engine3D::Animatable** χρησιμοποιείται από αντικείμενα τα οποία υποστηρίζουν κάποιας μορφής animation. Παρέχει μεθόδους με τις οποίες καθορίζεται το επιθυμητό animation και η θέση στο animation αυτό. Επιπλέον παρέχει παραμέτρους που ελέγχουν αν το animation θα εκτελεστεί μια φορά ή θα συνεχίσει να επαναλαμβάνεται κυκλικά μετά το τέλος του.

4.5.2 Πλέγματα τριγώνων



Εικόνα 20: Διάγραμμα κλάσεων πλέγματος τριγώνων

Το περισσότερα τρισδιάστατα αντικείμενα μιας σκηνής περιγράφονται από ένα πλέγμα τριγώνων. Τα πλέγματα τριγώνων υλοποιούνται από αντικείμενα της κλάσης

Engine3D::Mesh. Η κλάση αυτή περιέχει πίνακες των κορυφών του πλέγματος τριγώνων (*vertices*), των τριγώνων του πλέγματος (*faces*), κάθε ένα από τα οποία αποτελείται από τρεις από τις κορυφές αυτές, και των συντεταγμένων κάθε κορυφής για την αντίστοιχη εικόνα υφής (*texcoords*).

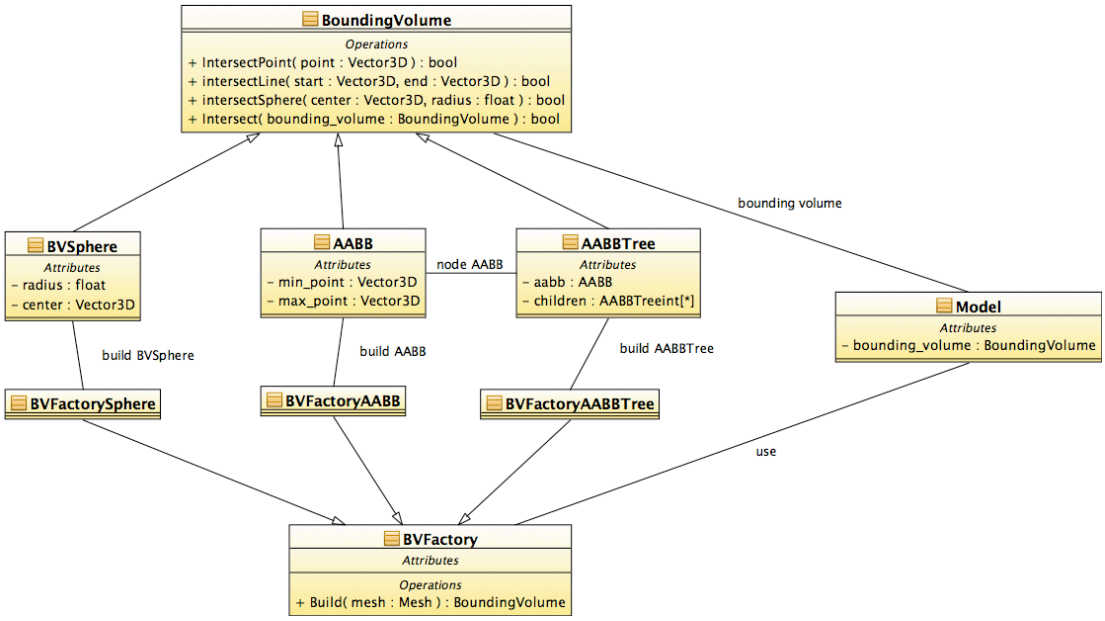
Για την κίνηση με την χρήση στιγμιότυπων τα τρίγωνα του πλέγματος είναι κοινά και απαιτούνται μόνο στιγμιότυπα της θέσης των κορυφών. Για λόγους εξοικονόμησης μνήμης, τα στιγμιότυπα αποτελούν μέρος της κλάσης **Mesh** και υλοποιούνται από την κλάση **Engine3D::Frame**. Έτσι, ο πίνακας *vertices* των κορυφών δεν ανήκει απευθείας στην κλάση **Mesh** αλλά υπάρχει ξεχωριστός πίνακας σε κάθε αντικείμενο **Frame**. Το αντικείμενο αυτό περιέχει επιπλέον ένα πίνακα *normals* τα οποία περιέχουν τα κάθετα διανύσματα φωτισμού ανά κορυφή, τα οποία είναι διαφορετικά για κάθε στιγμιότυπο. Τα στιγμιότυπα κίνησης αποθηκεύονται στον πίνακα *frames* του αντικειμένου **Mesh**. Ένας επιπλέον πίνακας (*animations*) χρησιμοποιείται για να περιγράψει τα πιθανά animations που έχει το πλέγμα τριγώνων και αποτελείται από αντικείμενα τύπου **Engine3D::Animation**. Τα αντικείμενα αυτά περιέχουν δείκτες στον πίνακα *frames* με το αρχικό και το τελικό στιγμιότυπο της κίνησης, η οποία αποτελείται από αυτά και όλα τα ενδιάμεσα διαδοχικά στιγμιότυπα, καθώς και την διάρκεια της κίνησης σε χιλιοστά του δευτερολέπτου.

Τα τρίγωνα του πλέγματος είναι αντικείμενα της κλάσης **Engine3D::IndexedFace** η οποία είναι στην ουσία ένας πίνακας με τρεις δείκτες στον πίνακα κορυφών, για τις τρεις αντίστοιχες κορυφές του τριγώνου. Διαδοχικές ομάδες τριγώνων στον πίνακα *faces* μπορούν να αντιστοιχηθούν με ένα κοινό αντικείμενο **Material**, για την περιγραφή του χρώματος και της εικόνας υφής τους, μέσω των αντικειμένων **Engine3D::FaceGroup** του πίνακα *face_groups* του αντικειμένου **Mesh**.

Εκτός από τις απαραίτητες μεθόδους αρχικοποίησης, διαχείρισης μνήμης και πρόσβασης στους παραπάνω πίνακες του αντικειμένου, το αντικείμενο **Mesh** παρέχει επιπλέον την μέθοδο **Compile** της διεπαφής **Draw**, για την αρχικοποίηση των κάθετων διανυσμάτων φωτισμού μέσω της μεθόδου **ComputeNormals** του χώρου ονοματοδοσίας **Engine3D::Math**, και την μέθοδο **Interpolate** η οποία παρεμβάλει δύο στιγμιότυπα κίνησης για την ομαλή παραγωγή της κίνησης του πλέγματος. Η παρεμβολή αυτή γίνεται τόσο στις κορυφές του πλέγματος όσο και στα κάθετα σε αυτές διανύσματα φωτισμού.

Το αντικείμενο **Mesh** χρησιμοποιείται κυρίως για την αποθήκευση και την διαχείριση του πλέγματος τριγώνων και δεν παρέχει πληροφορία για την θέση του στη σκηνή ή για την τρέχουσα κατάσταση της κίνησής του. Για τον λόγο αυτό δεν χρησιμοποιεί κάποια από τις προκαθορισμένες διεπαφές. Για την ενσωμάτωση του πλέγματος τριγώνου στην σκηνή χρησιμοποιείται το αντικείμενο **Engine3D::Model** το οποίο υπακούει στις διεπαφές **Drawable**, **Updateable**, **Moveable** και **Animatable**. Μέσω του αντικειμένου αυτού, ένα πλέγμα τριγώνων αποκτά θέση και προσανατολισμό στην σκηνή και είναι δυνατό να χρησιμοποιηθεί από το βρόχο εξομοίωσης για την ανανέωση και την απεικόνισή του. Επιπλέον, το αντικείμενο **Model** περιέχει ένα αντικείμενο τύπου **AABB** για τον περιβάλλοντα όγκο του πλέγματος, το οποίο θα περιγραφεί παρακάτω.

4.5.3 Περιβάλλοντες όγκοι



Εικόνα 21: Διάγραμμα κλάσσεων περιβάλλοντων όγκων

Για την επιτάχυνση της διαδικασίας απεικόνισης μέσω απόρριψης αντικειμένων και την επιτάχυνση των ελέγχων σύγκρουσης, έχουν υλοποιηθεί αντικείμενα που χρησιμοποιούνται από την μηχανή γραφικών για την αναπαράσταση περιβάλλοντων όγκων. Οι όγκοι που έχουν χρησιμοποιηθεί είναι η Περιβάλλου σφαίρα και το περιβάλλον κουτί προσανατολισμένο στους άξονες. Οι περιβάλλοντες όγκοι χρησιμοποιούν την διεπαφή **Engine3D::BoundingVolme** η οποία παρέχει μεθόδους για την τοποθέτηση του όγκου στον χώρο και για τον έλεγχο επικάλυψής του με κάποιο σημείο, ευθεία, σφαίρα ή άλλο περιβάλλοντα όγκο ίδιου τύπου. Ο κάθε όγκος μπορεί να κατασκευαστεί με σύμφωνα με κάποιες βασικές παραμέτρους αλλά συνήθως κατασκευάζεται σε αντιστοιχία με κάποιο άλλο γεωμετρικό αντικείμενο που πρέπει να περικλείει. Οι υποκλάσεις του αντικειμένου **Engine3D::BVFactory** υλοποιούν εκδόσεις τις μεθόδου **Build** με την οποία δημιουργούν περιβάλλοντες όγκους για κάποιο δοσμένο αντικείμενο **Mesh**. Οι διαθέσιμοι από την μηχανή όγκοι και οι αντίστοιχες **BVFactory** κλάσσεις είναι η εξής:

- **Engine3D::BVSphere**
  Ο όγκος μιας περιβάλλουσας σφαίρας. Η σφαίρα περιγράφεται από τις παραμέτρους *center* και *radius* οι οποίες καθορίζουν την θέση του κέντρου και την ακτίνα της αντίστοιχα. Κατασκευάζεται από το αντικείμενο **Engine3D::BVFactorySphere.**

- **Engine3D::AABB**
  Ο όγκος ενός περιβάλλοντος κουτιού προσανατολισμένου στου άξονες. Περιγράφεται από δύο σημεία των κορυφών του, της κορυφής *min* με τις μικρότερες συντεταγμένες και της κορυφής *max* με τις μεγαλύτερες συντεταγμένες στους τρεις άξονες. Κατασκευάζεται από το αντικείμενο **Engine3D::BVFactoryAABB.**

- **Engine3D::AABBTree**
  Μέσω αυτού του αντικειμένου κατασκευάζεται μια ιεραρχία περιβάλλοντων όγκων βασισμένη σε αντικείμενα **AABB**. Κατασκευάζεται από το αντικείμενο **Engine3D::BVFactoryAABBTree**

4.5.4 Άλλα τρισδιάστατα αντικείμενα

Για την δημιουργία ικανοποιητικών περιβάλλοντων χώρων και ουρανού, χωρίς μεγάλο κόστος απεικόνισης, η μηχανή γραφικών υποστηρίζει κουτιά ουρανού μέσω του αντικειμένου **Engine3D::Skybox**. Το αντικείμενο αυτό είναι στην ουσία ένας κύβος του οποίου το κέντρο τοποθετείται πάντα στην αρχή του συστήματος αξόνων της OpenGL και ο οποίος είναι πάντα σταθερός στην σκηνή. Αποτελεί υποκλάση της κλάσσεις **Mesh** και χρησιμοποιεί έξι αντικείμενα **Texture** για τις εικόνες των πλευρών. Περιέχει επιπλέον την παράμετρο *distance* η οποία καθορίζει το μέγεθος του κύβου.

Η επιφάνεια του εδάφους παράγεται από ένα αντικείμενο της κλάσης **Engine3D::Heightmap**. Το αντικείμενο αυτό δέχεται ένα δυσδιάστατο πίνακα τιμών τύπου **unsigned char** ο οποίο αντιστοιχεί σε ένα χάρτη ύψους. Ο πίνακας μπορεί να προέλθει είτε από αρχείο με RAW δεδομένα, είτε από κάποιο αρχείο εικόνας. Το αντικείμενο αυτό έχει επιπλέον τις παραμέτρους *min* και *length*, οι οποίες περιέχουν τις ελάχιστες τιμές των συντεταγμένων του αντικειμένου τις επιφάνειας εδάφους στο χώρο και το μήκος του αντικειμένου σε κάθε διάσταση αντίστοιχα. Το πλέγμα τριγώνων που αντιστοιχεί στην επιφάνεια εδάφους κατασκευάζεται με την μέθοδο **CreateMesh** του αντικειμένου. Η μέθοδος αυτή κατασκευάζει ένα αντικείμενο **Mesh** με κορυφές διατεταγμένες στις δύο διαστάσεις σε γραμμές και στήλες. Το πλήθος των κορυφών κάθε γραμμής ή στήλης είναι η αντίστοιχη διάσταση του πίνακα τιμών του χάρτη ύψους, προς ένα βήμα ανά διάσταση το οποίο δίνεται κατά την κλήση της μεθόδου. Μεγάλο βήμα οδηγεί σε πλέγμα τριγώνων με λιγότερες κορυφές και λιγότερη λεπτομέρεια. Η τρίτη διάσταση των κορυφών προκύπτει από παρεμβολή των τιμών του χάρτη ύψους ανάλογα με την θέση της κορυφής στο πλέγμα. Τα τρίγωνα του πλέγματος δημιουργούνται χωρίζοντας στην διαγώνιο τα τετράπλευρα που σχηματίζονται από τέσσερεις γειτονικές κορυφές.

Η μηχανή γραφικών παρέχει επιπλέον υλοποίηση συστήματος σωματιδίων για την εξομοίωση αντικειμένων φωτιάς, καπνού κτλ. μέσω της κλάσης **Engine3D::ParticleSystem**. Η κλάση αυτή υπακούει στις διεπαφές **Moveable, Drawable,** και **Updateable**. Είναι δηλαδή η κλάση που χρησιμοποιείται για την τοποθέτηση στην σκηνή, την απεικόνιση και την ανανέωση του συστήματος σωματιδίων. Υποστηρίζει τον σχεδιασμό ενός αριθμού σωματιδίων που καθορίζεται από την παράμετρο *partictes_number* και αναλαμβάνει τις κατάλληλες κλήσεις προς την OpenGL και τον χώρο ονοματοδοσίας **Engine3D::Draw** για τον σχεδιασμό του συστήματος σωματιδίων. Τα σωματίδια αυτά μοιράζονται κοινή εικόνα υφής μέσω του αντικειμένου **Texture**. Τα σωματίδια του συστήματος υλοποιούνται μέσω της κλάσης **Engine3D::Particle**. Η κλάση αυτή χρησιμοποιείται για την αποθήκευση των παραμέτρων που περιγράφουν ένα ξεχωριστό σωματίδιο. Περιέχει δύο μεταβλητές τύπου **Vector3D** *position* και *velocity* για την θέση και την ταχύτητα του σωματιδίου αντίστοιχα, μια μεταβλητή τύπου **Color** για το χρώμα του, και δύο μεταβλητές τύπου **float** *size* και *energy* για το μέγεθος και την ενέργεια του σωματιδίου. Η ενέργεια είναι μια παράμετρος η οποία μπορεί να χρησιμοποιηθεί αυθαίρετα ανάλογα με το είδος τους συστήματος σωματιδίων (πχ. μπορεί να αντιστοιχεί στον χρόνο ζωής του

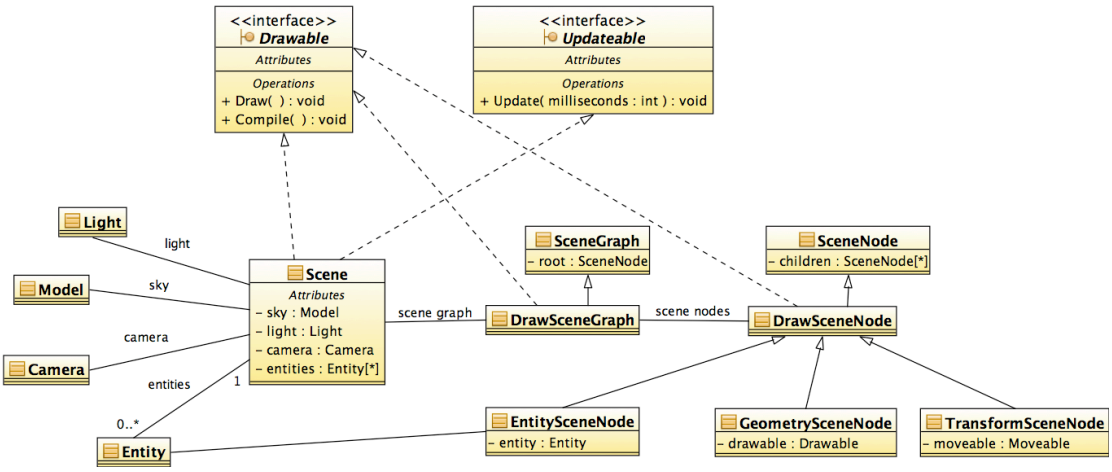
σωματιδίου). Κάθε σωματίδιο έχει επιπλέον μια παράμετρο που καθορίζει αν είναι ενεργό ή ανενεργό.

Η ανανέωση των σωματιδίων δεν υλοποιείται από το αντικείμενο **ParticleSystem**. Η μέθοδος **Update** θα πρέπει να υλοποιηθεί από μια υποκλάση του αντικειμένου αυτού. Στα πλαίσια της μηχανής γραφικών έχει υλοποιηθεί μια τέτοια υποκλάση **Engine3D::Fire** η οποία απεικονίζει φωτιές. Σύμφωνα με αυτή την κλάση τα σωματίδια ξεκινούν από το σημείο 0 του άξονα Y και σε θέση τυχαία ως προς το επίπεδο XZ άλλα σε μικρή απόσταση από το κέντρο των αξόνων. Η ταχύτητά τους τίθεται έτσι ώστε να κινούνται προς τα επάνω ως προς τον άξονα Y και προς το κέντρο. Η ενέργειά τους αντιστοιχεί στον χρόνο ζωής του κάθε σωματιδίου, από μηδέν μέχρι ένα δευτερόλεπτο, ενώ επηρεάζει επίσης και το μέτρο της ταχύτητάς του και το μέγεθος του. Μια επιπλέον διανυσματική παράμετρος *wind* έχει χρησιμοποιηθεί στο αντικείμενο για την εξομοίωση μιας σταθερής εξωτερικής δύναμης του αέρα.

4.6 Ιεραρχική χωρική δομή

Τα τρισδιάστατα αντικείμενα της σκηνής ομαδοποιούνται σε μια δομή γράφου η οποία χρησιμεύει στο να σχεδιάζει κατάλληλα το σύνολο των αντικειμένων και να εφαρμόζει τεχνικές οι οποίες έχουν υλοποιηθεί για την επιτάχυνση της απεικόνισης. Η δομή αυτή μπορεί να τροποποιηθεί ώστε να περιγράψει ιεραρχικά την σκηνή και να επιταχύνει περαιτέρω διαδικασίες όπως είναι ο έλεγχος σύγκρουσης.

4.6.1 Σκηνή



Εικόνα 22: Διάγραμμα κλάσσεων σκηνής

Η περιγραφή της τρισδιάστατης σκηνής ξεκινάει από την κλάση **Engine3D::Scene**. Το αντικείμενο αυτό ακολουθεί τις διεπαφές **Drawable** και **Updateable** για να απεικονίσει ολόκληρη την σκηνή και να ανανεώσει όλα τα αντικείμενά της. Περιέχει ένα αντικείμενο τύπου **Camera** για τον παρατηρητή της σκηνής, ένα αντικείμενο τύπου **Light** για τον φωτισμό και ένα αντικείμενο τύπου **Model** για το μοντέλο του ουρανού, όπως είναι τα αντικείμενα **Skybox**. Επιπλέον, περιέχει μια λίστα από αντικείμενα τύπου **Engine3D::Entity**. Τα αντικείμενα αυτά περιέχουν ένα αντικείμενο τύπου **Model** και το τοποθετούν στην σκηνή μέσω των διεπαφών **Drawable, Updateable** και **Moveable**. Αυτό είναι χρήσιμο στις περιπτώσεις όπου η σκηνή περιέχει πολλά ξεχωριστά αντικείμενα με ίδια απεικόνιση, πχ. δέντρα με το

ίδιο μοντέλο. Αν κάθε δέντρο αντιστοιχεί σε ένα αντικείμενο **Entity** με δείκτη στο ίδιο αντικείμενο **Model** τότε μπορούν να απεικονιστούν πολλά ίδια δέντρα με ένα μόνο αντίγραφο του μοντέλου στην μνήμη.

Η κλάση **Scene** περιέχει επιπλέον ένα αντικείμενο τύπου **Engine3D::DrawSceneGraph** το οποίο αντιστοιχεί στον γράφο της σκηνής. Ο γράφος περιέχει κόμβους οι οποίοι καθοδηγούν την απεικόνιση της σκηνής ανάλογα με την λειτουργία τους. Το βασικό είδος κόμβου είναι του τύπου **Engine3D::EntitySceneNode**. Ο κόμβος αυτός μπορεί να περιέχει αντικείμενα τύπου **Entity** τα οποία αποτελούν μέρος της σκηνής και θα απεικονιστούν κατά την κλήση της μεθόδου **Draw** του αντικειμένου **Scene**. Ο κόμβος τύπου **Engine3D::TransformSceneNode** εφαρμόζει στους κόμβους παιδιά του ένα κοινό μετασχηματισμό μετατόπισης. Περισσότερα είδη κόμβων μπορούν να δημιουργηθούν επεκτείνοντας την κλάση **Engine3D::SceneNode** για πιο εξειδικευμένες ιεραρχικές δομές όπως είναι τα οχταδικά δέντρα και τα δέντρα δυαδικού διαχωρισμού.

4.7 Βοηθητικές κλάσσεις

Ένα σύνολο βοηθητικών κλάσεων έχουν υλοποιηθεί για να επιτελέσουν κάποιες βασικές Λειτουργίες οι οποίες δεν αποτελούν μέρος της μηχανής γραφικών αλλά είναι χρήσιμες για πολλά τμήματά της.

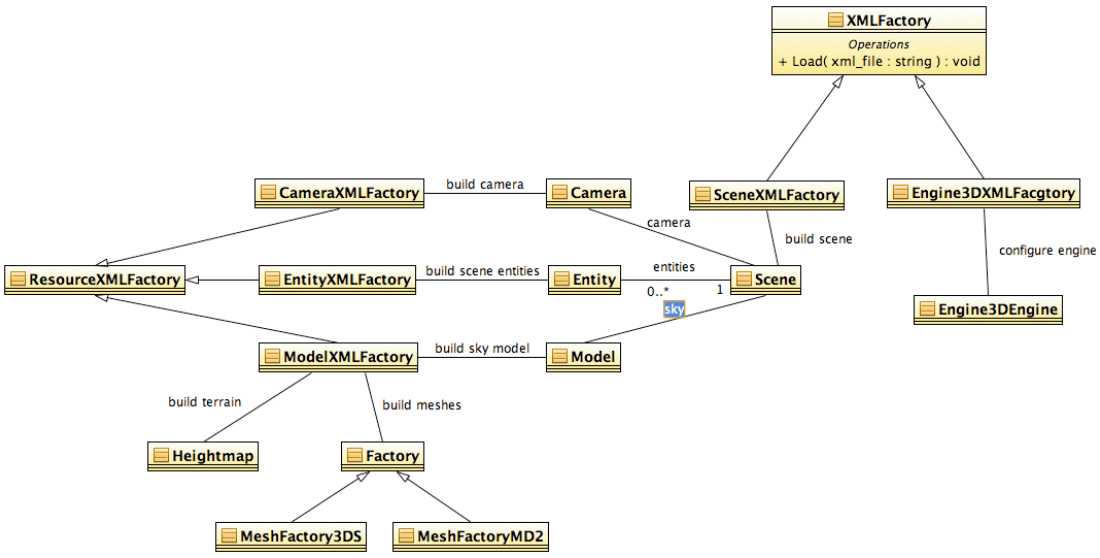
4.7.1 Factories

Τα αντικείμενα τύπου **Engine3D::Factory** και οι επεκτάσεις τους χρησιμεύουν για την αυτόματη κατασκευή σύνθετων αντικειμένων της μηχανής από κάποια απλή περιγραφή ή κάποιο αρχείο. Η κλάση **Engine3D::MeshFactory3DS**, **Engine3D::MeshFactoryMD2** και **Engine3D::MeshFactoryLWO** μπορούν να κατασκευάσουν αντικείμενα τύπου **Mesh** με πλέγματα τριγώνων βασισμένα σε αρχεία τύπου 3ds, md2 και lwo αντίστοιχα. Η κλάση **Engine3D::XMLFactory** μπορεί να αρχικοποιήσει βασικά αντικείμενα της μηχανής μέσω κάποιας απλής περιγραφής τους σε XML. Η επέκταση της **Engine3D::SceneXMLFactory** αρχικοποιεί το αντικείμενο **Scene** της σκηνής με ένα αρχείο XML όπως το παρακάτω:

```
<scene>
  <sky><include file="dragonvale.xml"/></sky>
  <camera name="camera">
    <position x="0" y="50" z="100"/>
  </camera>
  <entity name="akiko">
        <include file="akiko.xml"/>
        <position z="-20"/>
        <animation index="0" state="loop"/>
  </entity>
  <entity name="tree">
        <model><geometry>
                <mesh file="plant2.md2" type="md2" scale="1.2">
        </mesh></geometry></model>
        <position x="10" z="-10"/>
  </entity>
</scene>
```

Η σκηνή αυτή περιέχει ένα κουτί ουρανού, την κάμερα τοποθετημένη σε συγκεκριμένη θέση και δύο αντικείμενα. Με τον τρόπο αυτό, ο χρήστης της μηχανής

μπορεί να αλλάξει την σύνθεση της σκηνής χωρίς να χρειαστεί να αλλάξει τον κώδικα της εφαρμογής του.

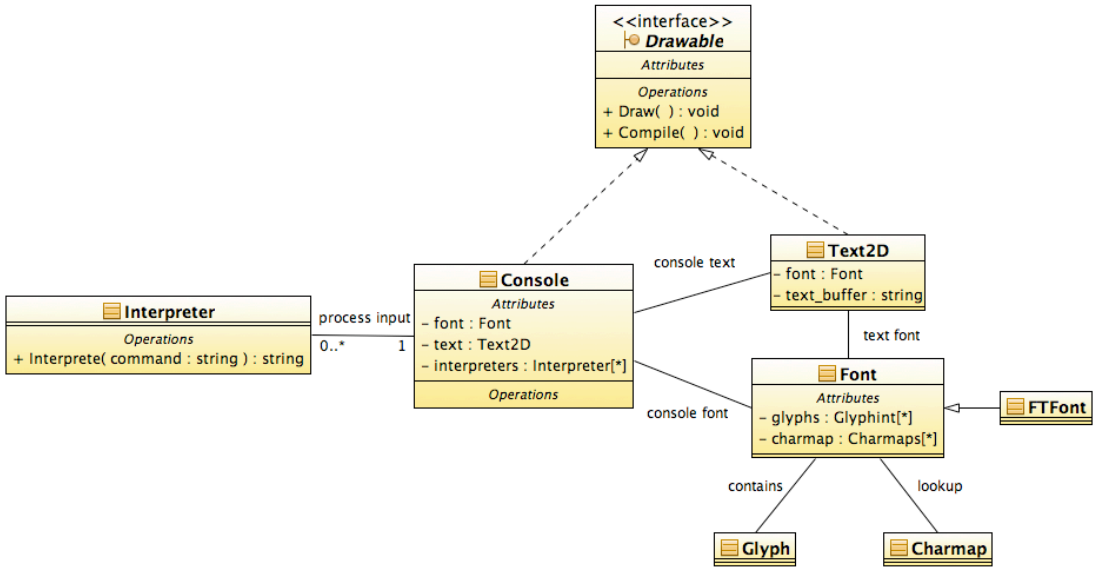


Εικόνα 23: Διάγραμμα κλάσσεων Factory

4.7.2 Γραμματοσειρές

Το αντικείμενο τύπου **Engine3D::Font** μπορεί να φορτώσει κάποια TrueType γραμματοσειρά η οποία με την σειρά της μπορεί να χρησιμοποιηθεί από αντικείμενα τύπου **Engine3D::Text2D** για την απεικόνιση κειμένου πάνω από την εικόνα που σχεδιάζει η μηχανή γραφικών. Το κείμενο αυτό μπορεί να χρησιμοποιηθεί για να εμφανίσει επιθυμητές πληροφορίες σε πραγματικό χρόνο, όπως για παράδειγμα οι εικόνες ανά δευτερόλεπτο που σχεδιάζονται.

4.7.3 Κονσόλα



Εικόνα 24: Διάγραμμα κλάσσεων κονσόλας και γραμματοσειρών

Το αντικείμενο τύπου **Engine3D::Console** υλοποιεί μια κονσόλα η οποία μπορεί να απεικονίσει πληροφορίες και να εκτελέσει εντολές. Οι εντολές αυτές και η λειτουργία

τους περιγράφονται από επεκτάσεις της κλάσσεις **Engine3D::Interpreter** και μπορούν να επηρεάσουν την λειτουργία της μηχανής σε πραγματικό χρόνο.

4.8 Παραδείγματα χρήσης της μηχανής γραφικών

Η μηχανή γραφικών η οποία υλοποιηθηκε στα πλαίσια αυτής της διπλωματικής εργασίας έχει κατασκευαστεί με τέτοιο τρόπο έτσι ώστε να χρησιμοποιείται ως ξεχωριστή ανεξάρτητη βιβλιοθηκή από εφαρμογές οι οποίες θέλουν να δώσουν στον χρήστη την απεικόνιση, πλοήγηση και διάδραση με ένα τρισδιάστατο χώρο. Οι εφαρμογές αυτές είτε χρησιμοποιούν τις κλάσσεις της μηχανής γραφικών, είτε χρησιμοποιούν επεκτάσεις τους οι οποίες προσαρμόζονται στις ανάγκες τις μηχανής. Παρακάτω δίνεται μια σειρά από παραδείγματα χρήσης της μηχανής για την κατασκευή τέτοιων εφαρμογών. Η γλώσσα προγραμματισμού που χρησιμοποιείται για όλα τα παραδείγμα είναι η C++ καθώς αυτή είναι η γλώσσα στην οποία έχει βασιστεί η μηχανή γραφικών.

4.8.1 Παράδειγα 1: Περιστρεφόμενος κύβος

Το παράδειγμα αυτό έχει σκοπό να περιγράψει τον σκελετό μιας βασικής εφαρμογής η οποία αρχικοποίει την μηχανή γραφικών και χρησιμοποιεί τον βρόχο εξομοίωσης. Η μηχανή γραφικών θα χρησιμοποιηθεί σε αυτό το παράδειγμα για την εμφάνιση ενός περιστρεφόμενου κύβου στο κέντρο της οθόνης.

Κάθε εφαρμογή η οποία επιθυμεί να χρησιμοποιήσει την μηχανή γραφικών θα πρέπει να αρχικοποιήσει την κλάσση **Engine3DEngine** και στην συνέχεια να περάσει τον έλεγχο σε μια υλοποίηση κάποιας επέκτασης του βρόχου εξομοίωσης μέσω της κλάσσης **SimulationLoop**. Η αρχικοποίηση, η οποία είναι γενικά κοινή για όλα τα παραδείγματα που αναφέρονται, γίνεται μέσα στην συνάρτηση **main** όπως φαίνεται στον ακόλουθο κώδικα:

main.h:

```
#include <Engine3D/sdl_render_system.h>
#include <Engine3D/engine3d_engine.h>
#include <Engine3D/engine3d_xmlfactory.h>
using namespace Engine3D;

#include "example1.h"

int main(int argc, char **argv)
{
    Engine3DEngine e(new SDLRenderSystem());
    Engine3DXMLFactory ex;
    ex.Load("engine.xml");
    Example1 loop;
    loop.Run();

    return 0;
}
```

Το αντικείμενο **Engine3DEngine** αποτελεί τον πυρήνα της μηχανής γραφικών και είναι κοινό και μοναδικό για κάθε σημείο του κώδικα. Αρχικοποιείται με ένα αντικείμενο τύπου **SDLRenderSystem** το οποίο αποτελεί την διεπαφή της μηχανής

γραφικών με την OpenGL μέσω της βιβλιοθήκης libSDL. Στο σημείο αυτό η μηχανή γραφικών μπορεί να παραμετροποιηθεί με κατάλληλες κλήσεις στις μεθόδους του αντικειμένου της. Είναι ευκολότερο όμως να χρησιμοποιηθεί ένα αντικείμενο τύπου XMLFactory το οποίο έχει κατασκευαστεί για αυτό ακριβώς τον σκοπό και αρχικοποιεί την μηχανή γραφικών μέσω μιας περιγραφής σε κάποιο XML αρχείο. Στο συγκεκριμένο παράδειγμα, το αρχείο αυτό ονομάζεται engine.xml και πρέπει να βρήσκεται στον ίδιο φάκελο με την εφαρμογή. Το ακόλουθο αρχείο αρχικοποιεί την μηχανή γραφικών μέσα σε παράθυρο με μέγεθος 1024x768:

engine.xml:

```
<?xml version="1.0"?>
<engine3d>
  <display xres="1024" yres="768" bpp="32" fullscreen="off"/>
  <viewport x="0" y="0"/>
</engine3d>
```

Για την υλοποίηση του παραδείγματος χρειαζεται μια υλοποίηση κάποιας επέκτασης του βρόχου εξομοίωσης και μια υλοποίηση του αντικειμένου του κύβου. Ο βρόχος στο παράδειγμα αυτό είναι η κλάση **Example1** και ο βρόχος ξεκινάει μέσω της μεθόδου SimulationLoop::Run().

Η καταλληλότερη μέθοδος για την αναπαράσταση ενός κύβου είναι μέσω ενός πλέγματος τριγώνων. Ένας κύβος έχει 8 κορυφές και 6 τετράγωνες πλευρές. Οι πλευρές του κύβου μπορούν να εκφραστούν με τρίγωνα αν χωριστούν με βάση την διαγώνιό τους. Έτσι καταλλήγουμε σε μια αναπαράσταση με 6 κορυφές και 12 πλευρές. Το αντικείμενο της μηχανής γραφικών το οποίο αναπαριστά πλέγματα τριγώνων είναι το **Engine3D::Mesh**. Θα υλοποιήσουμε την κλάση **Cube** ως επέκταση του αντικείμενου αυτού για την αναπαράσταση του κύβου και στην συνέχεια θα χρησιμοποιήσουμε την μηχανή γραφικών για την απεικόνισή του. Ακολουθεί ο κώδικας της κλάσσης **Cube**:

Cube.h:

```
#include "mesh.h"
#define CUBE_DEFAULT_SIZE     1.0f
using namespace std;

class Cube: public Mesh {
    public:

    float distance;

    Cube(float size = CUBE_DEFAULT_DISTANCE);
    ~Cube();
};
```

Cube.cpp:

```
#include "cube.h"

Cube::Cube(float dis) : Mesh(8, 12)
{
    AllocateFaceGroups(1);
```

```
    distance = dis;

    SetVertex( 0, -distance/2, -distance/2, -distance/2);
    SetVertex( 1, +distance/2, -distance/2, -distance/2);
    SetVertex( 2, +distance/2, +distance/2, -distance/2);
    SetVertex( 3, -distance/2, +distance/2, -distance/2);
    SetVertex( 4, -distance/2, -distance/2, +distance/2);
    SetVertex( 5, +distance/2, -distance/2, +distance/2);
    SetVertex( 6, +distance/2, +distance/2, +distance/2);
    SetVertex( 7, -distance/2, +distance/2, +distance/2);
    SetTexcoords( 0, 0, 1);
    SetTexcoords( 1, 1, 1);
    SetTexcoords( 2, 0, 0);
    SetTexcoords( 3, 1, 0);
    SetTexcoords( 4, 1, 1);
    SetTexcoords( 5, 0, 1);
    SetTexcoords( 6, 1, 0);
    SetTexcoords( 7, 0, 0);
    SetFace( 0, 1, 0, 5);
    SetFace( 1, 5, 0, 4);
    SetFace( 2, 5, 6, 2);
    SetFace( 3, 2, 1, 5);
    SetFace( 4, 2, 0, 1);
    SetFace( 5, 2, 3, 0);
    SetFace( 6, 3, 4, 0);
    SetFace( 7, 3, 7, 4);
    SetFace( 8, 3, 6, 7);
    SetFace( 9, 3, 2, 6);
    SetFace( 10, 4, 7, 6);
    SetFace( 11, 4, 6, 5);
    SetFaceGroup(1, 12);
}
```

Όπως φαίνεται από τον κώδικα της κλάσσης, η κλάσσης κληρονομεί την κλάση **Mesh** και την αρχικοποιεί με 8 κορυφές και 12 πλευρές. Στην αρχικοποίηση της κλάσσης δεσμέυεται ένα **FaceGroup** το οποίο περιέχει και τις 12 πλευρές. Αν επιθυμούμε ο κύβος να έχει κάποια εικόνα υφής, τότε θα έχει την ίδια εικόνα για όλες τις πλευρές. Για να θέσουμε διαφορετική εικόνα σε κάθε πλευρά θα πρέπει να δημιουργήσουμε ένα **FaceGroup** για κάθε πλευρά και να του αποδόσουμε τα αντίστοιχα τρίγωνα.

Για την ολοκλήρωση του παραδείγματος πρέπει να υλοποιηθεί η κλάση **Example1**. Σύμφωνα με την περιγραφή της αρχιτεκτονικής της μηχανής γραφικών, ο βρόχος εξομοίωσης περιέχει την μέθοδο *do_draw()* η οποία θα απεικονίζει το αντικείμενο του κύβου και την μέθοδο *do_update()* η οποία θα περιστρέφει τον κύβο με κάποια σταθερή ταχύτητα. Θα χρησιμοποιηθεί επίσης η μέθοδος *do_handle_keyboard()* έτσι ώστε η εφαρμογή να τερματίζεται με το πλήκτρο ESCAPE. Ακολουθεί ο κώδικας της κλάσσης **Example1**:

example1.h:

```
#include <Engine3D/simulation_loop.h>
#include <Engine3D/entity.h>
#include <Engine3D/scene.h>
using namespace Engine3D;

#define ROTATION_SPEED 0.05
```

```
class Example1: public SimulationLoop {
    public:

    Entity *entity;
    Scene *scene;

    Example1();
    ~Example1();

    int do_draw(void);
    int do_update(int);

    int do_handle_keyboard(SDL_KeyboardEvent *);
};
```

example1.cpp:

```
#include <SDL/SDL.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <Engine3D/engine3d_engine.h>
#include <Engine3D/scene_xmlfactory.h>
#include <Engine3D/mesh_model.h>
#include <Engine3D/cube.h>
using namespace std;

#include "example1.h"
using namespace Engine3D;

Example1::Example1()
{
    SceneXMLFactory sx;
    scene = sx.Load("scene.xml");

    Cube *cube = new Cube(10);
    MeshModel *model = new MeshModel(cube);
    entity = new Entity(model);
    scene->AddEntity(entity);

    entity->Rotate(Quaternion(Vector3D(0, 1, 0), 45));
    entity->SetPosition(0, 0, -30);
}

Example1::~Example1()
{
}

int Example1::do_draw()
{
    RenderSystem *rs = Engine3DEngine::GetRenderSystem();

    rs->Clear();
    scene->Draw();
    rs->Flush();

    return 0;
}

int Example1::do_update(int ticks)
```

```
{
    float angle = ROTATION_SPEED * ((float) ticks);
    entity->Rotate(Quaternion(Vector3D(0, 1, 0), angle));
    scene->Update(ticks);

    return 0;
}

int Example1::do_handle_keyboard(SDL_KeyboardEvent *e)
{
    if(e->state == SDL_RELEASED) return 0;
    switch(e->keysym.sym) {
      case SDLK_ESCAPE:
            running = false;
            break;
      default:
            break;
    }

    return 0;
}
```

Στο παράδειγμα αυτό, για λόγους μεγαλύτερης ευκολίας, το αντικείμενο **Cube** θα ενσωματωθεί στο αντικείμενο μιας σκηνής για την απεικόνισή του. Κατά τρόπο αντίστοιχο με την μηχανή γραφικών, η σκηνή θα αρχικοποιηθεί με το αρχείο scene.xml μέσω του αντικειμένου SceneXMLFactory:

scene.xml:

```
<?xml version="1.0"?>
<scene>
  <camera name="camera">
    <position x="0" y="0" z="0"/>
  </camera>
</scene>
```

Σύμφωνα με το αρχείο αυτό η σκηνή αρχικοποιείται με την κάμερα τοποθετημένη στην αρχή των αξόνων.

Για την τοποθέτηση του πλέγματος τριγώνων στον χώρο της σκηνής, το αντικείμενο **Mesh** θα πρέπει να ενσωματωθεί σε ένα αντικείμενο τύπου **Model** και αυτό στην συνέχεια θα πρέπει να ενσωματωθεί σε ένα αντικείμενο τύπου **Entity** το οποίο περιέχει και την πληροφορία της θέσης του αντικειμένου. Ο ακόλουθος κώδικας δημιουργεί τον κύβο μέσω της κλάσσης που υλοποιήθηκε προηγουμένος, και τον προσθέτει στην σκηνή:

```
    Cube *cube = new Cube(10);
    MeshModel *model = new MeshModel(cube);
    entity = new Entity(model);
    scene->AddEntity(entity);
```

Η θέση και ο προσανατολισμός του αντικειμένου στον χώρο της σκηνής γίνεται με τον ακόλουθο τρόπο:

```
    entity->Rotate(Quaternion(Vector3D(0, 1, 0), 45));
    entity->SetPosition(0, 0, -30);
```

Εδω φαίνεται η χρήση του αντικειμένου Quaternion το οποίο μπορεί να χρησιμοποιηθεί για περιστροφές αντικειμένων. Σε αυτό το παράδειγμα, ο κύβος περιστρέφεται γύρω από τον άξονα Z (`Vector3D(0, 1, 0)`) κατά 45 μοίρες.

Καθώς ο κύβος αποτελεί πλέον μέρος της σκηνής, η απεικόνιση του είναι πολύ απλή. Η μέθοδος *do_draw()* περιέχει τα ακόλουθα:

```
    RenderSystem *rs = Engine3DEngine::GetRenderSystem();

    rs->Clear();
    scene->Draw();
    rs->Flush();
```

Η μεταβλητή rs περιέχει την διεπαφή με το σύστημα απεικόνισης. Η συνάρτηση *rs->Clear()* καθαρίζει την οθόνη, η συνάρτηση *scene->Draw()* απεικονίζει την σκηνή με τον κύβο και η συνάρτηση *rs->Flush()* ειδοποιεί το σύστημα απεικόνισης ότι η απεικόνιση έχει ολοκληρωθεί.

Η περιστροφή του κύβου γίνεται μέσω της συνάρτησης *Rotate()*, η οποία περιστρέφει τον κύβο κατά γωνία ανάλογη με τον χρόνο που έχει περάσει από την προηγούμενη ανανέωση:

```
    float angle = ROTATION_SPEED * ((float) ticks);
    entity->Rotate(Quaternion(Vector3D(0, 1, 0), angle));
```

4.8.2 Παράδειγα 2: Μοντέλο πλέγματος τριγώνων με κίνηση

Στο παράδειγμα αυτό η μηχανή γραφικών θα χρησιμοποιηθεί για να απεικονίσει ένα μοντέλο αποθηκευμένο σε αρχείο μορφής MD2. Επιπλέον θα χρησιμοποιηθούν τα αποθηκευμένα στιγμιότυπα του μοντέλου για την κίνηση του. Το παράδειγμα θα βασιστεί στον σκελετό του προηγούμενου παραδείγματος με μικρές αλλαγές.

Η μηχανή γραφικών υποστηρίζει την αυτόματη κατασκευή του αντικείμενου **Mesh** το οποίο περιέχει το πλέγμα τριγώνων του μοντέλου το οποίο θα απεικονιστεί. Εισάγωντας την κατάλληλη περιγραφή του μοντέλου στο αρχείο XML της σκηνής, η μηχανή γραφικών κατασκευάζει αυτόματα τα απαραίτητα αντικείμενα και τα προσθέτει στην σκηνή για απεικόνιση. Επιπλέον, η περιγραφή του μοντέλου μπορεί, για διαχειριστικούς λόγους, να γίνει σε ξεχωριστό αρχείο και να ενσωματωθεί στο αρχείο της σκηνής όσες φορές είναι επιθυμητό. Το αρχείο το οποίο περιγράφει το μοντέλο είναι το ακόλουθο:

model.xml

```
<?xml version="1.0"?>
<model><geometry>
  <mesh file="akiko.md2" type="md2" scale="0.4">
    <!-- 00 stand --> <animation start="0" end="11" length="5000"/>
    <!-- 01 walk  --> <animation start="12" end="15" length="1000"/>
    <!-- 02 run -->   <animation start="16" end="19" length="500"/>
    <!-- 03 jump -->  <animation start="20" end="25" length="500"/>
    <!-- 04 taunt --> <animation start="26" end="33" length="3000"/>
```

```
    <!-- 05 attack --> <animation start="34" end="36" length="1000"/>
    <!-- 06 pain -->   <animation start="37" end="39" length="1000"/>
    <!-- 07 death -->  <animation start="40" end="46" length="4000"/>
    <!-- 08 ready -->  <animation start="47" end="58" length="5000"/>
    <!-- 09 frwrd2 --> <animation start="59" end="61" length="1000"/>
    <!-- 10 bkwrd2 --> <animation start="62" end="64" length="1000"/>
    <!-- 11 jump2 -->  <animation start="65" end="70" length="500"/>
    <!-- 12 block2 --> <animation start="71" end="75" length="1000"/>
    <!-- 13 punch2 --> <animation start="76" end="78" length="1000"/>
    <!-- 14 kick2 -->  <animation start="79" end="81" length="1000"/>
    <!-- 15 spcil2 --> <animation start="82" end="87" length="3000"/>
    <!-- 16 win2 -->   <animation start="88" end="93" length="5000"/>
    <!-- 17 loose2 --> <animation start="94" end="98" length="2000"/>
</mesh></geometry></model>
```

Στο αρχείο αυτό, στο tag *mesh* αναφέρεται το όνομα του αρχείου το οποίο θα φορτωθεί, ο τύπος του αρχείου, και ένας παράγοντας μεγέθυνσης των διαστάσεων του μοντέλου, ο οποίος είναι πορεραιτικός. Στη συνέχεια ακολουθεί μια σειρά από τις κινήσεις οι οποίες είναι αποθηκευμένες. Το αρχείο του μοντέλου έχει αποθηκευμένα μια σειρά από στιγμιότυπα του πλέγματος τριγώνων του, και κάθε κίνηση αποτελείται από ένα αριθμό διαδοχικών στιγμιοτύπων. Η περιγραφή της κίνησης γίνεται με το tag *animation* στο οποίο δίνεται ο αριθμός του αρχικού στιγμιοτύπου, ο αριθμός του τελικού στιγμιοτύπου και η διάρκεια της κίνησης σε χιλιοστά του δευτερολέπτου. Το αρχείο του μοντέλου ενσωματώνεται στο αρχείο της σκηνής με τον ακόλουθο τρόπο:

scene.xml:

```
<?xml version="1.0"?>
<scene>
  <camera name="camera">
    <position x="0" y="0" z="0"/>
  </camera>
  <entity name="model1">
    <include file="model.xml"/>
    <position z="-20"/>
    <animation index="0" state="loop"/>
  </entity>
</scene>
```

Στο αρχείο αυτό με το tag *include* προστίθεται αυτόματα η περιγραφή που έγινε μέσω του αρχείου model.xml. Επιπλέον δίνεται στο μοντέλο η αρχική του θέση και η αρχική κίνηση του.

Το αρχείο του μοντέλου έχει εσωτερική αναφορά στο αρχείο της εικόνας υφής που χρησιμοποιεί. Η μηχανή γραφικών θα αναζητήσει το αρχείο αυτό μέσω του συστήματος διαχείρισης πόρων. Μπορούμε να τοποθετήσουμε όλα τα αρχεία της σκηνής και του μοντέλου σε ένα κοινό φάκελο και να οδηγήσουμε το σύστημα διαχείρισης πόρων να τα αναζητήσει εκεί, αρχικοποιώντας την μηχανή γραφικών με το ακόλουθο αρχείο:

engine.xml:

```
<?xml version="1.0"?>
<engine3d>
```

```
  <display xres="1024" yres="768" bpp="32" fullscreen="off"/>
  <viewport x="0" y="0"/>
  <resource path="./data"/>
</engine3d>
```

Με το tag *resource* το σύστημα διαχείρισης πόρων γνωρίζει τους φακέλους στους οποίους είναι τοποθετημένα τα αρχεία στα οποία αναφέρεται η σκηνή.

Χρησιμοποιώντας αυτά τα αρχεία περιγραφής, και με αντίστοιχο κώδικα με αυτό του προηγούμενου παραδείγματος, Η μηχανή γραφικών θα απεικονίσει το μοντέλο και θα ξεκινήσει την κίνηση του. Για να αλλάξουμε μέσα από τον κώδικα την κατάσταση του μοντέλου, αν θέλουμε να το μετακινήσουμε ή να αλλάξουμε την κίνηση που εκτελεί, θα πρέπει να ζητήσουμε το κατάλληλο αντικείμενο μέσω του συστήματος διαχείρισης πόρων. Κατά την κατασκευή της σκηνής, η μηχανή γραφικών έχει τοποθετήσει αυτόματα το μοντέλο, καθώς και τα υπόλοιπα σχετικά αρχεία, στους διαθέσιμους πόρους, χρησιμοποιώντας το όνομα που έχει δωθεί από την περιγραφή μέσω του attribute *name*:

```
 <entity name="model1">
```

Με την ακόλουθη εντολή μπορούμε να ζητήσουμε από το σύστημα διαχείρισης πόρων το αντικείμενο αυτό:

```
entity = (Entity *) Engine3DEngine::
                        GetResourceManager()->GetResource("model1");
```

Στην συνέχεια μπορούμε να του αλλάξουμε την κίνηση του με την ακόλουθη εντολή:

```
akiko->GetModel()->Animate(animation_id);
```

όπου *animation_id* είναι ο αριθμός της κίνησης, όπως περιγράφεται στο αρχείο xml του μοντέλου.

4.8.3 Παράδειγα 3: Σύστημα σωματιδίων

Στο παράδειγμα αυτό θα χρησιμοποιηθεί η μηχανή γραφικών για την απεικόνιση ενώς συστήματος σωματιδίων το οποίο θα εξομοιώνει μια φωτιά. Για το σκοπό αυτό, χρειάζεται να επεκτείνουμε το αντικείμενο **ParticleSystem** και να παραμετροποιήσουμε κατάλληλα τα σωματίδια που θα συνθέσουν το αντικείμενο. Η κλάσση που θα υλοποιήσει το σύστημα σωματιδίων είναι η ακόλουθη:

fire.h:

```
#include <GL/gl.h>
#include "particle_system.h"

namespace Engine3D {

/**
 * \brief Fire particle system
 */
class Fire: public ParticleSystem {
    Vector3D wind;
```

```
    public:

    Fire(int, GLuint);
    ~Fire();

    void Initialize(Particle &);

    virtual void Update(int t);
    virtual void Compile();
};

} // namespace
```

fire.cpp:

```
#include "fire.h"
using namespace Engine3D;

Fire::Fire(int pn, GLuint text)
{
    texture = text;
    AllocateParticles(pn);
    Compile();

    wind.Set(0, 0, 0);
}

Fire::~Fire()
{
}

void Fire::Update(int ticks)
{
    for(int i = 0; i < particles_number; i++) {
      float energy = particles[i].GetEnergy() - ticks;
      if(energy < 0) Initialize(particles[i]);
      else {
            particles[i].SetEnergy(energy);
            particles[i].SetColor(Color::White * (energy / 1000));
            particles[i].SetPosition(particles[i].GetPosition() +
ticks * particles[i].GetVelocity());
      }
    }
}

void Fire::Compile()
{
    for(int i = 0; i < particles_number; i++) {
      Initialize(particles[i]);
    }
}

void Fire::Initialize(Particle &p)
{
    float e = Math::Rand(1000.0);
    float xp = 0.1 - Math::Rand(0.2);
    float zp = 0.1 - Math::Rand(0.2);
    p.SetEnergy(e);
    p.SetSize(e/6000);
    p.SetPosition(Vector3D(xp, 0, zp));
```

```
    xp /= 1000; zp /= 1000;
    wind.Add(xp / 30, 0, zp / 30);
    p.SetVelocity(wind + Vector3D(-xp, e / 1550000, -zp));
}
```

Το αντικείμενο αυτό αρχικοποιεί κάθε σωματίδιο μέσω της μεθόδου *Compile(),* η οποία ανήκει στην διεπαφή *Drawable.* Η μέθοδος αυτή πρέπει να κληθεί κατά την αρχικοποίηση του συστήματος σωματιδίων. Η αρχικοποίηση ανά σωματίδιο γίνεται με την μέθοδο *Initialize(Particle).*

Για να εξομοιώσουμε το αντικείμενο της φωτιάς, θεωρούμε ότι τα σωματίδια του ξεκινούν από κάποιο σημείο γύρω από την θέση του συστήματος σωματιδίων με συντεταγμένη y=0. Επιπλέον τα σωματίδια πρέπει να κινούνται με κατεύθυνση ανοδική και προς τον άξονα Υ. Θέτουμε την ταχύτητα ως προς τον άξονα Υ ανάλογη με την ενέργεια του κάθε σωματιδίου. Για λόγους μεγαλύτερης λεπτομέρειας, προσθέτουμε επιπλέον μια παράμετρο η οποία αντιστοιχεί σε μια εξωτερική δύναμη που κινεί τα σωματίδια, εξομοιώνοντας την ύπαρξη, για παράδειγμα, αέρα μ σταθερή φορά. Ονομάζουμε την παράμετρο αυτή *wind*. Υπολογίζουμε μια τυχαία αρχική θέση με αυτές τις παραμέτρους και θέτουμε κατάλληλα την ταχύτητα του κάθε σωματιδίου:

```
    float e = Math::Rand(1000.0);
    float xp = 0.1 - Math::Rand(0.2);
    float zp = 0.1 - Math::Rand(0.2);
    p.SetEnergy(e);
    p.SetPosition(Vector3D(xp, 0, zp));
    p.SetVelocity(wind + Vector3D(-xp, e / 1550000, -zp));
```

Η κατάσταση του κάθε σωματιδίου ανανεώνεται μέσω της μεθόδου *Update()*. Σε κάθε κλήση της μειώνεται η ενέργεια του σωματιδίου και η θέση του μετακινείται ανάλογα με την ταχύτητά του. Όταν η ενέργεια του σωματιδίου μηδενιστεί, θεωρούμε ότι το σωματίδιο εξαφανίζεται και το αρχικοποιούμε ξανά για την δημιουργεία νέου σωματιδίου. Στην πράξη, ο αριθμός των σωματιδίων είναι σταθερός.

Η κλάσση **Fire** η οποία δημιουργήθηκε, δέχεται ως παράμετρο μια εικόνα υφής η οποία θα χρησιμοποιηθεί σε κάθε σωματίδιο. Για την αρχικοποίηση του αντικειμένου, πρέπει πρώτα να φορτωθεί από την μηχανή γραφικών η εικόνα υφής. Η αρχικοποίηση γίνεται με τον ακόλουθο τρόπο:

```
    Texture *t = Engine3DEngine::
            GetResourceManager()->LoadTexture("myFire", "fire.png");
    fire = new Fire(50, ((GLTexture *) t)->GetTexture());
    fire->SetPosition(0, 0, -1);
```

Χρησιμοποιούμε το σύστημα διαχείρισης πόρων για την ανάγνωση του αρχείου fire.png και το προσθέτουμε στους διαθέσιμους πόρους με όνομα myFire. Η υπόλοιπη λειτουργεία της εφαρμογής γίνεται όπως έχει περιγραφεί και στα προηγούμενα παραδείγματα. Για την εμφάνιση του συστήματος σωματιδίων αρκει να κληθεί η μέθοδος *Draw* στην συνάρτηση απεικόνισης του βρόχου εξομοίωσης. Η

ανανέωσή του γίνεται με την μέθοδο *Update* στην μέθοδο ανανέωσης του βρόχου. Ο κώδικας του παραδείγματος είναι ο ακόλουθος:

example3.cpp:

```
#include <SDL/SDL.h>
#include <Engine3D/scene_xmlfactory.h>
#include <Engine3D/gl_texture.h>
#include <Engine3D/fire.h>
using namespace std;

#include "example3.h"
using namespace Engine3D;

Example3::Example3()
{
    SceneXMLFactory sx;
    scene = sx.Load("scene.xml");

    Texture *t = Engine3DEngine::
            GetResourceManager()->LoadTexture("myFire", "fire.png");
    fire = new Fire(50, ((GLTexture *) t)->GetTexture());
    fire->SetPosition(0, 0, -1);

    SDL_EnableUNICODE(1);
    SDL_EnableKeyRepeat(SDL_DEFAULT_REPEAT_DELAY,
SDL_DEFAULT_REPEAT_INTERVAL);
}

Example3::~Example3()
{
    if(scene) delete scene;
}

int Example3::do_draw()
{
    RenderSystem *rs = Engine3DEngine::GetRenderSystem();

    rs->Clear();
    scene->Draw();
    fire->Draw();

    rs->Flush();

    return 0;
}

int Example3::do_update(int ticks)
{
    scene->Update(ticks);
    fire->Update(ticks);

    return 0;
}

int Example3::do_handle_keyboard(SDL_KeyboardEvent *e)
{
    if(e->state == SDL_RELEASED) return 0;
    switch(e->keysym.sym) {
      case SDLK_ESCAPE:
            running = false;
```

```
            break;
      default:
            break;
    }

    return 0;
}
```

4.8.4 Παράδειγα 4: Morphing

Η μηχανή γραφικών μπορεί να χρησιμοποιηθεί για την εφαρμογή τεχνικών μορφοποίησης. Κατά την διαδικασία μορφοποίησης ένα μοντέλο μετασχηματίζεται δυναμικά σε ένα άλλο μοντέλο το οποίο περιγράφεται από ίδιο πλέγμα τριγώνων ώς προς τον αριθμό των κορυφών και των αντίστοιχων τριγώνων, με διαφορετική θέση. Για το σκοπό αυτό επεκτείνουμε το αντίκειμενο **Model** της μηχανής γραφικών έτσι ώστε να περιέχει αυθαίρετο αριθμό πλεγμάτων τριγώνων τα οποία θα μετασχηματίζει κατάλληλα σύμφωνα με την διαδικασία μορφοποίησης. Η διαδικασία μορφοποίησης γίνεται με την μέθοδο της παρεμβολής. Το αντικείμενο που θα κατεσκευάσουμε ονομάζεται **MorphModel** και ο κώδικάς του είναι ο ακόλουθος:

morph_model.h:

```
#include "mesh.h"
#include "model.h"

namespace Engine3D {

class MorphModel: public Model {
    int states_number;
    int length, time, target_state;
    Vector3D *vertex_buffer, *normal_buffer, **vertex_states,
**normal_states;
    Mesh *mesh;

    public:

    MorphModel(int /* # meshes */);
    ~MorphModel();

    void SetMesh(Mesh *m);
    void SetMeshState(int i, Mesh *m);
    void Morph(int /* target state */, int /* duration in msec */);
    bool isMorphing() { return length != 0; }

    void Draw();
    void Update(int);

    void Compile();
};

} // namespace
```

morph_model.cpp:

```
#include "engine3d_engine.h"
#include "mesh.h"
#include "morph_model.h"
using namespace Engine3D;

MorphModel::MorphModel(int i)
{
    states_number = i;
    vertex_states = new Vector3D *[i];
    normal_states = new Vector3D *[i];

    for(int j = 0; j < i; j++) {
      vertex_states[j] = normal_states[j] = NULL;
    }

    vertex_buffer = normal_buffer = NULL;

    length = time = 0;
    target_state = 0;
}

MorphModel::~MorphModel()
{
    if(states_number) {
      for(int i = 0; i < states_number; i++) {
            if(vertex_states[i]) delete[] vertex_states[i];
            if(normal_states[i]) delete[] normal_states[i];
      }

      delete[] vertex_states;
      delete[] normal_states;
    }

    if(vertex_buffer) delete[] vertex_buffer;
    if(normal_buffer) delete[] normal_buffer;
}

void MorphModel::SetMesh(Mesh *m)
{
    int vn = m->GetVerticesNumber();
    mesh = m;
    frame = mesh->NewFrame();
    if(vertex_buffer) delete[] vertex_buffer;
    if(normal_buffer) delete[] normal_buffer;

    vertex_buffer = new Vector3D[vn];
    normal_buffer = new Vector3D[vn];

    for(int i = 0; i < states_number; i++) {
      if(vertex_states[i]) delete[] vertex_states[i];
      if(normal_states[i]) delete[] normal_states[i];

      vertex_states[i] = new Vector3D[vn];
      normal_states[i] = new Vector3D[vn];
    }

    for(int i = 0; i < vn; i++) {
      Vector3D pos(mesh->GetVertex(i)), nor(mesh->GetNormal(i));

      vertex_buffer[i].Set(pos);
      frame->vertices[i].Set(pos);
```

```
      normal_buffer[i].Set(nor);
      frame->normals[i].Set(nor);
      for(int j = 0; j < states_number; j++) {
            vertex_states[j][i].Set(pos);
            normal_states[j][i].Set(nor);
      }
    }
}

void MorphModel::SetMeshState(int j, Mesh *m)
{
    int vn = mesh->GetVerticesNumber();

    for(int i = 0; i < vn; i++) {
      vertex_states[j][i].Set(m->GetVertex(i));
      normal_states[j][i].Set(m->GetNormal(i));
    }
}

void MorphModel::Draw()
{
    RenderSystem *rs = Engine3DEngine::GetRenderSystem();
    RenderOperation ro;

    ro.vertices_number = mesh->GetVerticesNumber();
    ro.vertices = frame->vertices;
    ro.normals = frame->normals;
    ro.texcoords = mesh->GetTexcoords();

    int face_groups_number = mesh->GetFaceGroupsNumber();

    for(int i = 0; i < face_groups_number; i++) {
      const FaceGroup &face_group = mesh->GetFaceGroup(i);
      if(!face_group.faces_number) continue;

      ro.material = face_group.material;
      ro.indices_number = face_group.faces_number * 3;
      ro.indices = (unsigned int *) face_group.faces;
      rs->Render(&ro);
    }
}

void MorphModel::Morph(int new_state, int duration)
{
    int vn = mesh->GetVerticesNumber();
    time = 0; length = duration; target_state = new_state;

    for(int i = 0; i < vn; i++) {
      vertex_buffer[i].Set(frame->vertices[i]);
      normal_buffer[i].Set(frame->normals[i]);
    }
}

void MorphModel::Update(int ticks)
{
    aabb.Set(mesh->GetVerticesNumber(), frame->vertices);

    if(!length) return;

    time += ticks;
    if(time >= length) {
```

```
      time = 0; length = 0;

      for(int i = 0; i < mesh->GetVerticesNumber(); i++) {
            frame->vertices[i].Set(vertex_states[target_state][i]);
            frame->normals[i].Set(normal_states[target_state][i]);
      }

    } else {
      float state = ((float) time / (float) length);
      for(int i = 0; i < mesh->GetVerticesNumber(); i++) {
            frame->vertices[i].Set(((vertex_buffer[i] +
                        state*(vertex_states[target_state][i]-
vertex_buffer[i])))));

            frame->normals[i].Set(((normal_buffer[i] +
                        state*(normal_states[target_state][i]-
normal_buffer[i])))));
      }
    }
}

void MorphModel::Compile()
{
    mesh->Compile();
}
```

Το αντικείμενο αυτό χρησιμοποιεί το πρώτο πλέγμα τριγώνων που αποθηκεύεται σε αυτό για την δομή του μοντέλου, και κρατάει από τα υπόλοιπα μοντέλα την πληροφορία της θέσης των κορυφών και των κάθετων διανυσμάτων φωτισμού στους πίνακες vertex_states και normal_states αντίστοιχα. Η μεταβλητή state αποθηκεύει το σημείο παρεμβολής ανάμεσα στα δύο πλέγματα τριγώνω τα οποία πρόκειται να μορφοποιηθούν. Σύμφωνα με τις διεπαφές στις οποίες υπακούει η κλάσση **Model**, το μοντέλο απεικονίζεται και ανανεώνεται με τις μεθόδους *Draw()* και *Update()* αντίστοιχα. Κατά την ανανέωση του μοντέλου υπολογίζονται οι θέσεις των κορυφών και τα αντίστοιχα κάθετα διανύσματα φωτισμού ώς εξής:

```
      float state = ((float) time / (float) length);
      for(int i = 0; i < mesh->GetVerticesNumber(); i++) {
            frame->vertices[i].Set(((vertex_buffer[i] +
                        state*(vertex_states[target_state][i]-
vertex_buffer[i])))));

            frame->normals[i].Set(((normal_buffer[i] +
                        state*(normal_states[target_state][i]-
normal_buffer[i])))));
      }
    }
```

Το σημείο παρεμβολής υπολογίζεται από την δεδομένη χρονική στιγμή (time) και την χρονική διάρκεια της διαδικασίας (length) και στην συνέχεια για κάθε κορυφή υπολογίζεται η θέση της και το αντίστοιχο κάθετο διάνυσμα φωτισμού.

Η απεικόνιση γίνεται με απευθείας κλήση το σύστημα απεικόνισης της μηχανής γραφικών μέσω ενός αντικειμένου **RenderOperation**:

```
    RenderSystem *rs = Engine3DEngine::GetRenderSystem();
```

```
    RenderOperation ro;

    ro.vertices_number = mesh->GetVerticesNumber();
    ro.vertices = frame->vertices;
    ro.normals = frame->normals;
    ro.texcoords = mesh->GetTexcoords();

    int face_groups_number = mesh->GetFaceGroupsNumber();

    for(int i = 0; i < face_groups_number; i++) {
      const FaceGroup &face_group = mesh->GetFaceGroup(i);
      if(!face_group.faces_number) continue;

      ro.material = face_group.material;
      ro.indices_number = face_group.faces_number * 3;
      ro.indices = (unsigned int *) face_group.faces;
      rs->Render(&ro);
    }
```

Στο αντικείμενο αυτό αποδίδονται οι κορυφές και τα κάθετα διανύσματα φωτισμού του μοντέλου που έχει υπολογιστεί από την διαδικασία μορφοποίησης. Στην συνέχεια το μοντέλο απεικονίζεται μέσω αυτού του αντικειμένου και της συνάρτησης *Render()*.

4.8.5 Παράδειγα 5: Πλοήγηση σε σκηνή

Στο παράδειγμα αυτό θα κατασκευαστεί μια σκηνή η οποία θα περιέχει μια επιφάνεια γης μέσω ενός χάρτης ύψους και ένα μοντέλο το οποίο θα μπορεί να κινηθεί αυθαίρετα πάνω σε αυτή. Θα χρησιμοποιηθεί επιπλέον ένα κουτί ουρανό για μεγαλύτερο ρεαλισμό στον περιβάλλοντα χώρο. Όπως και στα προηγούμενα παραδείγματα, η σκηνή θα περιγραφεί από ένα αρχείο xml:

scene.xml:

```
<?xml version="1.0"?>
<scene>
  <sky><include file="skybox.xml"/></sky>
  <camera name="camera">
    <position x="0" y="0" z="0"/>
  </camera>
  <entity name="model1">
    <include file="model.xml"/>
    <position z="-20"/>
    <animation index="0" state="loop"/>
  </entity>
</scene>
```

Στο αρχείο αυτό περιλαμβάνεται το μοντέλο μέσω του αρχείου model.xml το οποίο περιγράφηκε στο Παράδειγμα 2. Το κουτί ουρανού ενσωματώνεται μέσω του αρχείου skybox.xml το οποίο είναι το ακόλουθο:

```
<?xml version="1.0"?>
<model><geometry>
<skybox distance="1000">
```

```
  <image side="top" file="dragonvale_up.jpg"/>
  <image side="bottom" file="dragonvale_dn.jpg"/>
  <image side="left" file="dragonvale_lf.jpg"/>
  <image side="right" file="dragonvale_rt.jpg"/>
  <image side="front" file="dragonvale_ft.jpg"/>
  <image side="back" file="dragonvale_bk.jpg"/>
</skybox>
</geometry></model>
```

Στο αρχείο αυτό περιγράφονται οι εικόνες υφής οι οποίες θα χρησιμοποιηθούν σε κάθε πλευρά του κουτιού. Η επιφάνεια της γης κατασκευάζεται με ένα χάρτη ύψους με τον ακόλουθο τρόπο:

```
    Texture *h = Engine3DEngine::
      GetResourceManager()->LoadTexture("Terrain", "ground.jpg");

    heightmap = new Heightmap("heightmap.bmp");
    heightmap->SetMapMinSizes(Vector3D(-500, -100, -500));
    heightmap->SetMapLengths(Vector3D(1000, 200, 1000));
    Mesh *mesh = heightmap->CreateMesh(4, 4);
    Material *material = new Material();
    material->SetTexture(h);
    mesh->SetMaterial(material);

    MeshModel *model = new MeshModel(mesh);
    model->UseWireframe(false);
    terrain = new Entity(model);
    terrain->SetPosition(0, -2, -10);
```

Αρχικά χρησιμοποιούμε το σύστημα διαχείρισης πόρων για την ανάγνωση της εικόνας υφής που θα χρησιμοποιηθεί στο αντικείμενο. Στην συνέχεια κατασκευάζουμε ένα χάρτη ύψους βασισμένο στην εικόνα heightmap.bmp, και θέτουμε τις παραμέτρους των διαστάσεών του. Κατασκευάζουμε το αντίστοιχο πλέγμα τριγώνων με την μέθοδο *CreateMesh()* η οποία δέχεται ως παραμέτρους το βήμα με το οποίο κατασκευάζονται οι κορυφές του πλέγματος σε κάθε διάσταση. Μεγαλύτερες τιμές βήματος έχουν ως αποτέλεσμα λιγότερες κορυφές στο πλέγμα με συνέπεια την ταχύτερη απεικόνιση του αλλά και μικρότερη λεπτομέρεια. Τέλος, προσθέτουμε το πλέγμα σε ένα αντικείμενο μοντέλου για να του αποδόσουμε συγκεκριμένη θέση μέσα στην σκηνή.

Η κίνηση του μοντέλου γίνεται μέσω του πληκτρολογίου από την αντίστοιχη συνάρτηση του βρόχου εξομοίωσης. Η συνάρτηση αυτή είναι η ακόλουθη:

```
int Example5::do_handle_keyboard(SDL_KeyboardEvent *e)
{
    switch(e->keysym.sym) {
      case SDLK_ESCAPE:
            running = false;
            break;
      case SDLK_w:
            if(e->state == SDL_RELEASED) {
                  speed.SetZ(0);
                  entity->GetModel()->Animate(0);
            } else {
                  speed.SetZ(CAMERA_MOVE_SPEED);
```

```
                entity->GetModel()->Animate(1);
            }
            break;
      case SDLK_s:
            if(e->state == SDL_RELEASED) {
                speed.SetZ(0);
                entity->GetModel()->Animate(0);
            } else {
                speed.SetZ(-CAMERA_MOVE_SPEED);
                entity->GetModel()->Animate(1);
            }
            break;
      case SDLK_a:
            if(e->state == SDL_RELEASED) {
                entity_rotate = 0;
                entity->GetModel()->Animate(0);
            } else {
                entity_rotate = CAMERA_ROTATE_SPEED;
                entity->GetModel()->Animate(1);
            }
            break;
      case SDLK_d:
            if(e->state == SDL_RELEASED) {
                entity_rotate = 0;
                entity->GetModel()->Animate(0);
            } else {
                entity_rotate = -CAMERA_ROTATE_SPEED;
                entity->GetModel()->Animate(1);
            }
            break;
      default:
            speed.SetZ(0);
            break;
    }

    return 0;
}
```

Η κίνηση γίνεται μέσω των πλήκτρων w, s, d και a. Η μεταβλητή speed αποθηκεύει την ταχύτητα του μοντέλου, ενώ η παράμετρος entity_rotate την ταχύτητα περιστροφής του. Έχουμε θεωρήσει ότι η ταχύτητα εκφράζεται ως προς τον προσανατολισμό του μοντέλου. Επιπλέον, με την κίνηση του μοντέλου χρησιμοποιείται διαφορετική κίνηση του μοντέλου η οποία μπορεί να αναπαρηστά το περπάτημά του.

Καθώς το μοντέλο κινείται στο χώρο της σκηνής, θέλουμε η κατακόρυφη θέση του να βρίσκεται επάνω στην επιφάνεια της γής. Επίσης, τοποθετούμε την κάμερα σε σταθερή θέση πίσω από το μοντέλο έτσι ώστε να το ακολουθεί συνεχώς. Η διαδικασία αυτή γίνεται στην συνάρτηση ανανέωσης του βρόχου εξομοίωσης ως εξής:

```
    camera = scene->GetCamera();

    const Vector3D *entity_pos = &(entity->GetPosition());
    entity->MoveForward(ticks*speed);
    entity->SetPosition(entity_pos->GetX(), 10 +
    heightmap->GetHeight(entity_pos->GetX(), entity_pos->GetZ()),
```

```
entity_pos->GetZ());
    entity->Rotate(Quaternion(Vector3D(0, 1, 0),
      ticks * entity_rotate));

    camera->SetOrientation(entity->GetOrientation());
    camera->Rotate(Quaternion(Vector3D(0, 1, 0), 180));
    camera->SetPosition(entity->GetPosition());
    camera->MoveForward(Vector3D(0, 15, 30));
```

Η θέση του μοντέλου μετακινείται ανάλογα με την ταχύτητα που του έχει δοθεί από το πληκτρολόγιο μέσω της μεθόδου *MoveForward()* η οποία το μετακινεί κατά ένα διάνυσμα ως προς τον προσανατολισμό του. Η κατακόρυφη θέση του υπολογίζεται από τον χάρτη ύψους μέσω της μεθόδου *GetHeight()*. Στην συνέχεια, η θέση της κάμερας υπολογίζεται από την θέση του μοντέλου και επιπλέον τοποθετείται πίσω από αυτό κατά μια σταθερή απόσταση, επίσης μέσω της μεθόδου *MoveForward()*.

5. Επίλογος

Στο πλαίσιο αυτής της διπλωματικής εργασίας αναπτύχθηκε ο βασικός κορμός μιας μηχανής αναπαράστασης τρισδιάστατων γραφικών πραγματικού χρόνου. Η υλοποίηση μιας τέτοιας μηχανής αποτελεί ένα πολύπλοκο πρόβλημα τόσο από πλευράς αλγορίθμων διαχείρισης του όγκου των δεδομένων μιας τρισδιάστατης σκηνής, όσο και από πλευράς αρχιτεκτονικής λογισμικού για την αποδοτική διαχείρισή τους. Η ανάπτυξη της τεχνολογίας των επεξεργαστών γραφικών επιτρέπει την ολοένα και γρηγορότερη απεικόνιση μεγάλου αριθμού πολυγώνων, ταυτόχρονα όμως αυξάνουν και οι απαιτήσεις για μεγαλύτερη λεπτομέρεια και μεγαλύτερο βαθμό ρεαλισμού. Η εργασία αυτή μπορεί να επεκταθεί μελλοντικά για να ικανοποιήσει αυτές τις απαιτήσεις.

5.1 Μελλοντικές επεκτάσεις

Ένα σημαντικό κομμάτι μελλοντικής εργασίας μπορεί να αφορά στην χρήση συστημάτων σωματιδίων για την αναπαράσταση αντικειμένων των οποίων το σχήμα είναι ευμετάβλητο και αλληλεπιδρά άμεσα με τα υπόλοιπα αντικείμενα της τρισδιάστατης σκηνής. Παραδείγματα τέτοιων αντικειμένων είναι οι υγροί όγκοι και αντικείμενα τα οποία αναπαριστούν υφάσματα ή τριχώματα. Στον τομέα των φωτισμών επιπλέον έρευνα μπορεί να γίνει για την ρεαλιστική απεικόνιση σκιών, αντανακλάσεων και άλλων φαινομένων του φωτός όπως είναι η διάθλαση.

Καθώς οι σύγχρονοι επεξεργαστές γραφικών επιτρέπουν την χρήση ειδικών προγραμμάτων τα οποία παρεμβαίνουν στην διαδικασία απεικόνισης με αρκετά ευέλικτο τρόπο, μεγάλο κομμάτι της μελλοντικής εργασίας μπορεί να αφορά στην χρήση της για την αναπαράσταση διαφόρων φαινομένων τα οποία προσεγγίζουν τον τρόπο με τον οποίο αντιλαμβάνεται το ανθρώπινο μάτι τον τρισδιάστατο χώρο.

6. Λίστα εικόνων

7. Βιβλιογραφία

[1] James D. Foley, Andries van Dam, et al., Computer Graphics - Principles and Practice in C, Addison-Wesley, 1995

[2] Eric Lengyel, Mathematics for 3D Game Programming and Computer Graphics, Second Edition, Charles River Media, 2003

[3] Dave Shreiner, Mason Woo, OpenGL Programming Guide, Fifth Edition, OpenGL Architecture Review Board, 2005

[4] Randima Fernando, GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics, Addison-Wesley, 2004

[5] Matthew Pharr, GPU Gems 2, Addison-Wesley, 2005

[6] Hubert Nguyen, GPU Gems 3, Addison-Wesley, 2007

[7] Philip Schneider and David Eberly, Geometric Tools for Computer Graphics, Morgan Kaufmann, 2002

[8] Christer Ericson, Real-Time Collision Detection, Morgan Kaufmann, 2004