ΤΕΙ Κρήτης

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΠΟΛΥΜΕΣΩΝ

Πτυχιακή Εργασία

Μελέτη των δυνατοτήτων της πλατφόρμας ηλεκτρονικών παιχνιδιών XNA της Microsoft.

ΕΙΣΗΓΗΤΗΣ: ΑΘΑΝΑΣΙΟΣ ΜΑΛΑΜΟΣ
ΣΠΟΥΔΑΣΤΡΙΑ: ΑΓΓΕΛΙΚΗ ΚΥΤΙΝΟΥ

Ηράκλειο 2010

…σε όλους εκείνους που υπήρξαν δίπλα μου
τα χρόνια της φοιτητικής μου ζωής, πίστεψαν σε μένα
και με υποστήριξαν…

Περίληψη:

Ένας πολύ ενδιαφέρον τρόπος εκμετάλλευσης του υπολογιστή είναι η ανάπτυξη ενός καλού ηλεκτρονικού παιχνιδιού. Από την εποχή του Pong μέχρι και σήμερα, τα βίντεο παιχνίδια έχουν συλλάβει την φαντασία δισεκατομμυρίων ανθρώπων.
Το Xna Game Studio, είναι μια πλατφόρμα που παρέχεται από τη Microsoft και δίνει τη δυνατότητα σε επαγγελματίες ή ερασιτέχνες προγραμματιστές και φοιτητές να δημιουργήσουν και να διαχειριστούν ηλεκτρονικά παιχνίδια χρησιμοποιώντας τη γλώσσα προγραμματισμού C#. Τα παιχνίδια αυτά είναι συμβατά με το λειτουργικό σύστημα των Windows, την κονσόλα βίντεο παιχνιδιών Xbox 360, όπως επίσης και με την πλατφόρμα ψυχαγωγίας Zune της Microsoft.
Σκοπός αυτής της εργασίας ήταν η μελέτη των δυνατοτήτων της πλατφόρμας Xna. Η μελέτη αυτή πραγματοποιήθηκε με τη δημιουργία μιας εφαρμογής εικονικής περιήγησης που ενσωματώνει τους διάφορους τύπους εικονικών κόσμων με τους οποίους συνεργάζεται και το μοντέλο φυσικής που χρησιμοποιήθηκε.
Συγκεκριμένα, δημιουργήθηκε ένας τρισδιάστατος κόσμος με τη βοήθεια λογισμικού σχεδίασης γραφικών τριών διαστάσεων. Χρησιμοποιήθηκαν επιπλέον έτοιμα τρισδιάστατα μοντέλα, ήχοι και υφές από τον παγκόσμιο ιστό. Εισάγοντας τα αντικείμενα αυτά στην πλατφόρμα Xna, μελετήθηκε η θέση, η περιστροφή και η αναπαράστασή τους στην οθόνη. Τέλος, έγινε εισαγωγή της μηχανής φυσικής και η μελέτη της λειτουργίας της πάνω στα μοντέλα που χρησιμοποιήθηκαν.

Abstract:

Perhaps nothing does a better job of harnessing the power of the interest of the user than a good computer game. From the early days of Pong to the latest titles of today, video games have captured the imagination of billions of people.
Xna Game Studio is a platform provided by Microsoft that enables students, professional and non-professional game programmers to create games using C# programming language that will run on Windows, Xbox 360 console and the Microsoft Zune.
The scope of this final year dissertation was to study the advantages offered by Xna platform. This study was presented via a virtual tour application that integrates the various formats of virtual worlds and the physics engine that was used to collaborate with.
Specifically, a three-dimensional (3D) world was created with the help of 3D modeling software. There were also 3D models, sounds and textures downloaded from the World Wide Web. These objects were inserted in the Xna platform and studied for their position, rotation and how they can be drawn on the screen. Finally, the physics engine was applied and studied on how it works on the 3D models that were used.

ευχαριστίες:

θα ήθελα πολύ να πω ένα μεγάλο ευχαριστώ στον κύριο Μαλάμο που με την εμπιστοσύνη, την στήριξη και την υπομονή που έδειξε με βοήθησε να τελειώσω την πτυχιακή μου εργασία

στους φίλους μου που πίστεψαν σε εμένα και με την υπομονή και τα γέλια τους με στήριξαν

στον φίλο και συμφοιτητή μου Γιάννη Ζιάμο που είχε το μεράκι να με βοηθήσει με προβλήματα που προέκυψαν στην εργασία μου και άντεξε τις συναισθηματικές μου εκρήξεις με χιούμορ

και τέλος τους γονείς και την αδερφή μου που με στηρίζουν και με βοηθούν όλα αυτά τα χρόνια που βρίσκομαι μακριά τους…

Αγγελική Κυτίνου
Ηράκλειο 2010

Περιεχόμενα:

1

Εισαγωγή

1.1 Αναδρομή

Ποτέ δεν υπήρξε καλύτερη στιγμή για να μάθετε πώς να το προγραμματίζετε. Οι σύγχρονες γλώσσες προγραμματισμού, σε συνδυασμό με τα ισχυρά και ευρέως διαθέσιμα περιβάλλοντα ανάπτυξης, προσφέρουν ένα εξαιρετικό μέρος για να εργαστεί κάποιος.Ένα ευρύ φάσμα λειτουργικών περιβαλλόντων, συμπεριλαμβανομένων των κινητών συσκευών, των υπολογιστών, της ρομποτικής, των ενσωματωμένων συστημάτων, καθώς και παιχνιδιών, σημαίνει ότι μπορείτε να εφαρμόσετε τις δεξιότητες του προγραμματισμού σας ένα τεράστιο φάσμα διαφορετικών τομέων.
Ο προγραμματισμός σας επιτρέπει να φέρετε τις ιδέες σας στη ζωή, και με την γλώσσα προγραμματισμού C # και το XNA μπορείτε να μάθετε πολλά για το πώς δουλεύουν τα παιχνίδια ακόμα και να δημιουργήσετε τα δικά σας εντελώς νέα παιχνίδια.
Η C# είναι μια μαζικά δημοφιλής γλώσσα προγραμματισμού που χρησιμοποιείται από πολλές χιλιάδες προγραμματιστές λογισμικού σε όλο τον κόσμο.

Πολλοί προγραμματιστές αποφάσισαν να εισέλθουν στον τομέα της πληροφορικής και ειδικότερα του προγραμματισμού, εξαιτίας των ηλεκτρονικών παιχνιδιών. Η ανάπτυξη ενός παιχνιδιού μπορεί να είναι ένας από τους πιο προκλητικούς κλάδους της μηχανικής λογισμικού, μπορεί ίσως να είναι η πιο σημαντική! Ποτέ στο παρελθόν δεν ήταν δυνατό στους περισσότερους προγραμματιστές να δημιουργήσουν παιχνίδια για μια κονσόλα παιχνιδιών. Η Microsoft δείχνει το δρόμο για το πώς μπορεί ένα παιχνίδι να αναπτυχθεί για κονσόλες παιχνιδιών. Η μεγάλη είδηση για το Xbox 360 είναι ότι η Microsoft έχει ξοδέψει τόσο πολύ χρόνο με τα χρόνια στην δημιουργία ενός παραγωγικού και σταθερού περιβάλλοντος ανάπτυξης για τους προγραμματιστές.
Ας κάνουμε μια αναδρομή στις μέρες του Dos. Για να προγραμματίσουν παιχνίδια και να φτιάξουν demos, οι προγραμματιστές συνήθως έπρεπε να γράψουν κώδικα χαμηλού επιπέδου για να μιλήσουν απευθείας με την κάρτα ήχου, την κάρτα γραφικών και με τις συσκευές εισόδου. Αυτό προφανώς ήταν κουραστικό και ο κώδικας αυτός ήταν επιρρεπής σε σφάλματα, διότι οι διαφορετικοί κατασκευαστές χειρίζονταν διαφορετικά το BIOS, τις θύρες εισόδου/εξόδου και την μνήμη, επομένως ο κώδικας μπορούσε να εκτελεστεί σε ένα σύστημα και όχι σε κάποιο άλλο.
Αργότερα, η Microsoft κυκλοφόρησε το Windows 95 λειτουργικό σύστημα. Πολλοί προγραμματιστές ήταν επιφυλακτικοί στην ανάπτυξη παιχνιδιών για Windows-και δικαίως-γιατί δεν υπήρχε τρόπος για να φτάσουν στο επίπεδο του υλικού (hardware) για να κάνουν πράγματα που απαιτούσαν μεγάλη ταχύτητα. Τα Windows 95 είχαν ένα προστατευτικό πρότυπο μνήμης που κρατούσε τους προγραμματιστές μακριά από το να έχουν πρόσβαση σε χαμηλού επιπέδου παρεμβάσεις στο hardware.
Για την επίλυση αυτού του προβλήματος, η Microsoft δημιούργησε μια τεχνολογία που ονομάζεται DirectX. Για την ακρίβεια ονομαζόταν Windows Game SDK στο ξεκίνημά του, αλλά γρήγορα άλλαξε όνομα μετά από έναν δημοσιογράφο που έπαιξε με τα ονόματα του API, το DirectDraw, το DirectSound και DirectPlay, καλώντας το SDK Direct "X". Έτσι γεννήθηκε το DirectX 1.0 λίγους μήνες μετά την κυκλοφορία των Windows 95.

Εξαιτίας του DirectX, οι προγραμματιστές είχαν έναν τρόπο να δημιουργήσουν παιχνίδια με μία πηγή που θα μπορούσε να λειτουργήσει σε όλους τους υπολογιστές, ανεξάρτητα από το υλικό τους. Οι πωλητές hardware ήταν πρόθυμοι να συνεργαστούν με τη Microsoft, για την τυποποίηση μιας διεπαφής για την πρόσβαση του υλικού τους. Δημιούργησαν οδηγούς συσκευής στους οποίους το DirectX θα χαρτογραφούσε το API τους, έτσι είχε ληφθεί μέριμνα για όλες τις εργασίες που προηγουμένως έπρεπε να γίνουν από τους προγραμματιστές παιχνιδιών, έτσι εκείνοι είχαν περισσότερο χρόνο να ξοδέψουν για να δημιουργήσουν τα παιχνίδια τους. Οι πωλητές το ονόμασαν αυτό, Hardware Abstraction Layer (HAL).

Με το πέρασμα των χρόνων είδαμε τρομερή ανάπτυξη στον κλάδο ανάπτυξης ηλεκτρονικών παιχνιδιών. Όταν η Microsoft έφερε στο προσκήνιο το Xna, τον Δεκέμβριο του 2006, αμέσως έγινε ξεκάθαρο ότι αυτή η νέα τεχνολογία έχει να προσφέρει γκάμα δυνατοτήτων σε όλους όσους ασχολούνται με την ανάπτυξη βίντεο παιχνιδιών. Το Xna σχεδιάστηκε από το μηδέν για μεγαλύτερη ευκολία στον προγραμματισμό, ενώ δεν θυσιάζονται οι επιδόσεις ή οι δυνατότητες για την επίτευξη αυτού του στόχου. Τα τελευταία χρόνια έχει αναπτυχθεί μια μεγάλη κοινότητα γύρω από το Xna. Με μια σύντομη αναζήτηση στον παγκόσμιο ιστό, κάθε ενδιαφερόμενος μπορεί να βρει ένα μεγάλο αριθμό από παραδείγματα κώδικα σε πολλά site, να κάνει ερωτήσεις σε διάφορα παράθυρα συζητήσεων (forum) που σχετίζονται με αυτό, ή ακόμα και να συναντήσει άλλους χρήστες που μοιράζονται το ίδιο πάθος σε κάποιο από τα Xna group χρηστών.

Αυτή η εργασία, εξηγεί πως αναπτύσσεται μια τρισδιάστατη εφαρμογή η οποία εκτελείται στο περιβάλλον των Windows, τι εργαλεία χρειάστηκαν και τι προβλήματα παρουσιάστηκαν κατά την πορεία ανάπτυξής της.

Μαθαίνοντας να Προγραμματίζετε

Αν δεν έχετε προγραμματίσει ποτέ πριν, μην ανησυχείτε. Ο προγραμματισμός δεν είναι επιστήμη πυραύλων. Υπάρχουν πολλοί περισσότεροι άνθρωποι στον κόσμο που έχουν μάθει προγραμματισμό από την επιστήμη πυραύλων. Οι κακές ειδήσεις για την εκμάθηση του προγραμματισμού είναι ότι έχετε πολλά διαφορετικά πράγματα για να μάθετε κατά την εκκίνηση, και αυτό μπορεί να προκαλέσει σύγχυση. Αλλά τα κλειδιά για την εκμάθηση προγραμματισμού είναι απλά:

- **Πρακτική**: Να δουλεύετε συνέχεια και να γράφετε προγράμματα και αναγκάστε τον εαυτό σας να σκεφτεί τα πράγματα από την πλευρά επίλυσης των προβλημάτων.

- **Μελέτη**: Κοιτάξτε προγράμματα που έχουν αναπτυχθεί από άλλους προγραμματιστές. Μπορείτε να μάθετε πολλά από τη μελέτη του κώδικα που άλλοι έχουν δημιουργήσει. Ανακαλύπτοντας το πώς κάποιος άλλος έκανε τη δουλειά είναι ένα μεγάλο σημείο εκκίνησης για τη λύση σας. Και να θυμάστε ότι σε πολλές περιπτώσεις, δεν υπάρχει καλύτερη λύση, μόνο λύσεις που είναι καλύτερα σε ένα συγκεκριμένο πλαίσιο (με άλλα λόγια, μερικές φορές χρειάζεται μια προσέγγιση που είναι η γρηγορότερη ή η μικρότερη ή η πιο εύκολη στη χρήση και ούτω καθεξής).

- **Εμμονή:** Η ανάπτυξη προγραμμάτων είναι σκληρή δουλειά. Και εσείς πρέπει να δουλέψετε σκληρά για αυτό. Ο κύριος λόγος που οι περισσότεροι δεν ακολουθούν ως προγραμματιστές είναι ότι τα παρατάνε, όχι γιατί δεν μπορούν να καταλάβουν. Ωστόσο, μη είστε πολύ επίμονοι. Αν δεν έχετε λύσει ένα πρόβλημα προγραμματισμού σε κάποιο μεγάλο χρονικό διάστημα, θα πρέπει να κάνετε ένα διάλειμμα και να ζητήσετε βοήθεια ή, τουλάχιστον, να απομακρυνθείτε από το πρόβλημα και να επανέλθετε σε αυτό αργότερα. Μένοντας επάνω όλη τη νύχτα προσπαθώντας να λύσετε ένα πρόβλημα δεν είναι ένα καλό σχέδιο. Απλά προκαλεί εκνευρισμό το πρωί. Αν ξεκουραστείτε και ξεφύγετε για λίγο, και επιστρέψετε στο πρόβλημα το πρωί, θα εκπλαγείτε πόσο συχνά μπορείτε να το διορθώσετε σε λίγα λεπτά.

1.2 To DirectX

Το Microsoft DirectX είναι μια συλλογή από εφαρμογές διεπαφών προγραμματισμού (API) για τη διεκπεραίωση καθηκόντων που σχετίζονται με τα πολυμέσα, και ιδίως τον προγραμματισμό παιχνιδιών και βίντεο, στις πλατφόρμες της Microsoft. Αρχικά, τα ονόματα αυτών των API, όλα ξεκίνησαν με Direct, όπως Direct3D, DirectDraw, DirectMusic, DirectPlay, DirectSound, και ούτω καθεξής. Το όνομα του DirectX επινοήθηκε ως όρος στενογραφία για όλα αυτά τα APIs (του διαρκούς X για το συγκεκριμένο όνομα API) και σύντομα έγινε το όνομα της συλλογής. Όταν η Microsoft αργότερα ορίζεται για την ανάπτυξη μιας κονσόλας παιχνιδιών, το X είχε χρησιμοποιηθεί ως βάση για την ονομασία του Xbox για να δείχνουν ότι η κονσόλα αυτή βασίζεται σε τεχνολογία DirectX. Το αρχικό X έχει μεταφερθεί στην ονομασία του API που σχεδιάστηκε για το Xbox, όπως το XInput και το Cross-Platform Audio Creation Tool (XACT) (εργαλείο δημιουργίας ήχων), ενώ το πρότυπο DirectX συνεχίστηκε για τα Windows APIs, όπως το Direct2D και το DirectWrite.

Το Direct3D (τα 3D API για γραφικά μέσα στο DirectX) χρησιμοποιείται ευρέως για την ανάπτυξη των βίντεο παιχνιδιών για τα Microsoft Windows, Microsoft Xbox και Microsoft Xbox 360. Το Direct3D χρησιμοποιείται επίσης από άλλες εφαρμογές λογισμικού για την οπτικοποίηση και για εργασίες γραφικών όπως η CAD / CAM μηχανική. Όπως το Direct3D είναι η πιο ευρύτερα δημοσιοποιημένη συνιστώσα του DirectX, είναι κοινό να δούμε τα ονόματα "DirectX" και "Direct3D" να χρησιμοποιούνται εναλλακτικά.

Το λογισμικό DirectX Development Kit (SDK), αποτελείται από βιβλιοθήκες χρόνου εκτέλεσης σε αναδιανομής δυαδική μορφή, μαζί με τα συνοδευτικά έγγραφα και κεφαλίδες για χρήση στην κωδικοποίηση. Αρχικά, οι χρόνοι εκτέλεσης είχαν εγκατασταθεί μόνο από παιχνίδια ή ρητώς από το χρήστη. Τα Windows 95 δεν ξεκίνησαν με το DirectX, αλλά το DirectX περιλαμβάνεται στο Windows 95 OEM Service Release 2. Τα Windows 98 και τα Windows NT 4.0 ήρθαν στην αγορά με το DirectX, όπως έχει κάθε έκδοση των Windows που κυκλοφόρησε μετά. Το SDK είναι διαθέσιμο ως δωρεάν λήψη. Ενώ οι χρόνοι εκτέλεσης είναι ιδιόκτητο, κλειστού κώδικα λογισμικό, πηγαίος κώδικας παρέχεται για την πλειονότητα των δειγμάτων SDK.

Το Direct3D 9Ex, το Direct3D 10 και το Direct3D 11 είναι διαθέσιμα μόνο για τα Windows Vista και τα Windows 7, διότι κάθε μία από αυτές τις νέες εκδόσεις κτίστηκε να εξαρτάται από τα νέα Windows Display Driver Model η οποία είχε εισαχθεί για τα Windows Vista. Η νέα Vista / WDDM αρχιτεκτονική γραφικών περιλαμβάνει μια νέα διαχείριση μνήμης οθόνης που υποστηρίζει εικονικοποίηση (virtualizing) υλικού γραφικών σε πολλαπλές εφαρμογές και υπηρεσίες, όπως το Desktop Window Manager.

History

Στα τέλη του 1994 η Microsoft ήταν στα πρόθυρα της κυκλοφορίας του επόμενου λειτουργικού συστήματος, τα Windows 95. Ο βασικός παράγοντας που θα καθορίσει την αξία κατανάλωσης, θα πραγματοποιηθεί με το νέο της λειτουργικό σύστημα που βασιζόταν σε μεγάλο βαθμό για το τι προγράμματα θα είναι σε θέση να λειτουργούν με αυτό. Τρεις υπάλληλοι της Microsoft – ο Craig Eisler, ο Alex St. John, και ο Eric Engstrom - ανησυχούσαν γιατί οι προγραμματιστές έτειναν να βλέπουν προηγούμενο λειτουργικό σύστημα της Microsoft, το MS-DOS, ως την καλύτερη πλατφόρμα για τον προγραμματισμό παιχνιδιών, που σημαίνει λίγα παιχνίδια θα αναπτύσσονταν για τα Windows 95 και το λειτουργικό σύστημα δεν θα είναι τόσο μεγάλη επιτυχία.

Το DOS επέτρεπε άμεση πρόσβαση σε κάρτες γραφικών, πληκτρολόγια, ποντίκια, συσκευές ήχου, καθώς και όλα τα άλλα μέρη του συστήματος, ενώ τα Windows 95, με το προστατευτικό του πρότυπο μνήμης, περιόρισε την πρόσβαση σε όλα αυτά, που δουλεύουν σε πολύ πιο τυποποιημένο πρότυπο. Η Microsoft χρειαζόταν έναν τρόπο που θα επέτρεπε στους προγραμματιστές να πάρουν αυτό που ήθελαν, και θα το

χρειάζονταν γρήγορα. Η κυκλοφορία του λειτουργικού αυτού συστήματος απείχε μόνο μερικούς μήνες. Ο Eisler (υπεύθυνος ανάπτυξης), ο St. John, και ο Engstrom (διαχειριστής προγράμματος) εργάστηκαν από κοινού για την επίλυση αυτού του προβλήματος, με μια λύση που θα ονομάστηκε τελικά DirectX.

Η πρώτη έκδοση του DirectX κυκλοφόρησε το Σεπτέμβριο του 1995 ως το Windows Games SDK. Ήταν το Win32 η αντικατάσταση του DCI και ταWinG APIs για τα Windows 3.1. Με απλά λόγια, το DirectX επέτρεψε όλες τις εκδόσεις των Microsoft Windows, αρχίζοντας από τα Windows 95, να ενσωματώσουν υψηλής απόδοσης πολυμέσα.

Το DirectX 2.0 έγινε το ίδιο ένα στοιχείο των Windows μαζί με την κυκλοφορία των Windows 95 OSR2 και Windows NT 4.0 στα μέσα του 1996.Όσο τα Windows 95 ήταν ακόμη νέα και λίγα παιχνίδια είχαν κυκλοφορήσει για αυτά, η Microsoft που ασχολιόταν με τη βαρέα προώθηση του DirectX σε προγραμματιστές που ήταν γενικά δύσπιστοι για την ικανότητα της Microsoft να δημιουργήσει μια πλατφόρμα βίντεο παιχνιδιών στα Windows. Ο Alex St. John, που εργαζόταν ως ευαγγελιστής για το DirectX, οργάνωσε μια πολυσύνθετη εκδήλωση στο Computer Game Developers Conference το 1996 όπου ο προγραμματιστής παιχνιδιών, Jay Barnson περιγράφτηκε ως ένα θέμα Roman, που συμπεριλάμβανε πραγματικά λιοντάρια, τηβέννους, και κάτι που έμοιαζε με εσωτερικό καρναβάλι. Ήταν σε αυτό το γεγονός που η Microsoft παρουσίασε το πρώτο Direct3D και DirectPlay, και όπως αποδείχθηκε, το πολλών παικτών (multi-player) MechWarrior 2 που παίζεται μέσω του παγκόσμιου ιστού.

Η ομάδα του DirectX αντιμετώπισε το δύσκολο έργο του ελέγχου κάθε έκδοσης του DirectX από μια σειρά από υλικό εξοπλισμό και λογισμικό. Μια ποικιλία από διαφορετικές κάρτες γραφικών, κάρτες ήχου, μητρικές κάρτες, επεξεργαστές, συσκευές εισόδου, παιχνίδια και άλλες εφαρμογές πολυμέσων εξετάστηκαν με κάθε beta και κάθε τελική έκδοση. Η ομάδα του DirectX, επίσης, εκτέλεσε και προσέφερε δοκιμές που επέτρεπαν στον κλάδο του υλικού να επιβεβαιώσει ότι τα νέα σχέδια υλικού και οι κυκλοφορίες οδηγών θα ήταν συμβατά με το DirectX.

Πριν από το DirectX, η Microsoft είχε συμπεριλάβει την OpenGL για την Windows NT πλατφόρμα τους. Την εποχή εκείνη, η OpenGL απαίτησε "high-end" υλικό και ήταν εστιασμένο στη μηχανική και στις CAD χρήσεις. Το Direct3D που επρόκειτο να είναι ένας ελαφρύς συνεταίρος για την OpenGL, επικεντρώθηκε στην χρήση του παιχνιδιού. Όπως το 3D gaming μεγάλωσε, η OpenGL εξελίχθηκε για να περιλάβει την καλύτερη υποστήριξη για τις τεχνικές προγραμματισμού για αλληλεπιδρόμενες πολυμεσικές εφαρμογές, όπως παιχνίδια, δίνοντας στους προγραμματιστές την επιλογή ανάμεσα στη χρήση της OpenGL ή του Direct3D, όπως τα 3D γραφικά API για τις εφαρμογές τους. Σε εκείνο το σημείο μια "μάχη" ξεκίνησε μεταξύ οπαδών του OpenGL cross-platform και του Windows Direct3D. Παρεμπιπτόντως, η OpenGL υποστηρίχθηκε στη Microsoft από την ομάδα του DirectX. Αν ένας προγραμματιστής επέλεγε να χρησιμοποιήσει OpenGL 3D API γραφικά, τα άλλα APIs του DirectX συνδυάζονταν συχνά με την OpenGL σε παιχνίδια στον υπολογιστή, επειδή το OpenGL δεν περιλαμβάνει το σύνολο των λειτουργιών του DirectX (όπως ο ήχος ή joystick υποστήριξης).

Σε μια κονσόλα συγκεκριμένης έκδοσης, το DirectX χρησιμοποιήθηκε ως βάση για το Xbox της Microsoft και το API της κονσόλας Xbox 360. Το API αναπτύχθηκε από κοινού μεταξύ της Microsoft και της Nvidia, το οποίο ανέπτυξε το συνηθισμένο υλικό γραφικών που χρησιμοποιείται από το αρχικό Xbox. Το Xbox API είναι παρόμοιο με την έκδοση του DirectX 8.1, αλλά δεν έχει τη δυνατότητα ενημέρωσης όπως και οι άλλες τεχνολογίες της κονσόλας. Το Xbox είχε την κωδική ονομασία DirectXbox, αλλά αυτό ήταν για να μειωθεί το Xbox για την εμπορική του ονομασία.

Το 2002 η Microsoft κυκλοφόρησε το DirectX 9 με υποστήριξη για τη χρήση των προγραμμάτων περισσότερης σκίασης από πριν με pixel και vertex έκδοση Shader 2.0. Η Microsoft συνέχισε να ενημερώνει την οικογένεια του DirectX από τότε, εισάγοντας το μοντέλο shader 3.0 στο DirectX 9.0c, που κυκλοφόρησε τον Αύγουστο του 2004.

Από τον Απρίλιο του 2005, το DirectShow διαγράφηκε από το DirectX και αντί αυτού μεταφέρθηκε στο Microsoft Platform SDK. Το DirectX SDK, ωστόσο, εξακολουθεί να απαιτείται για την κατασκευή των δειγμάτων DirectShow.

.Net Framework

Το 2002 η Microsoft κυκλοφόρησε μια έκδοση του DirectX συμβατή με το .NET Framework της Microsoft, επιτρέποντας έτσι στους προγραμματιστές να επωφεληθούν από τη λειτουργικότητα του DirectX από το εσωτερικό των .NET εφαρμογών χρησιμοποιώντας συμβατές γλώσσες, όπως η διαχείριση της C++ ή η χρήση της γλώσσας προγραμματισμού C#. Αυτό το API ήταν γνωστό ως "Managed DirectX" (ή MDX για συντομία), και ζήτησε να λειτουργεί σε 98% απόδοση του υποκείμενου DirectX API. Τον Δεκέμβριο του 2005, τον Φεβρουάριο του 2006, τον Απρίλιο του 2006, και τον Αύγουστο του 2006, η Microsoft κυκλοφόρησε διαδοχικές ενημερώσεις σε αυτήν τη βιβλιοθήκη, με αποκορύφωμα μια έκδοση beta που ονομάστηκε Managed DirectX 2.0. Ενώ το Managed DirectX 2.0 ενοποίησε λειτουργίες που προηγουμένως ήταν διασκορπισμένες σε πολλά σώματα, σε ένα ενιαίο συγκρότημα, η απλούστευση εξαρτήσεων σε αυτό για τους προγραμματιστές ανάπτυξης λογισμικού, η ανάπτυξη σε αυτή την έκδοση στη συνέχεια διεκόπη, και δεν υποστηρίζεται πλέον. Η βιβλιοθήκη του Managed DirectX 2.0 έληξε στις 5 Οκτωβρίου του 2006.

Κατά τη διάρκεια της GDC 2006, η Microsoft παρουσίασε το XNA Framework, μια νέα διαχείριση της έκδοσης του DirectX (παρόμοια, αλλά δεν ταυτίζεται με τη διαχείριση DirectX), που προορίζεται να βοηθήσει στην ανάπτυξη των παιχνιδιών καθιστώντας ευκολότερη την ενσωμάτωση του DirectX, υψηλού επιπέδου Shader Language (HLSL) και άλλα εργαλεία σε ένα πακέτο. Υποστηρίζει επίσης την εκτέλεση διαχειριζόμενου κώδικα στο Xbox 360. Το XNA Game Studio Express RTM ήταν διαθέσιμο στις 11 Δεκεμβρίου 2006, ως δωρεάν λήψη για τα Windows XP. Σε αντίθεση με το DirectX runtime, για τη διαχείριση DirectX, XNA Framework ή τα Xbox 360 APIs (XInput, XACT κ.λπ.) δεν έχουν αποσταλεί ως μέρος των Windows.

Κανένα προϊόν της Microsoft, συμπεριλαμβανομένης της τελευταίες κυκλοφορίες του XNA , δεν παρέχει υποστήριξη του DirectX 10 για το. NET Framework.

1.3 Η OpenGL

Η OpenGL (Open Graphics Library) είναι μια βασική προδιαγραφή που καθορίζει διαγλωσσική, cross-platform API για τη δημιουργία εφαρμογών που παράγουν 2D και 3D γραφικά υπολογιστών. Η διεπαφή (Interface) αποτελείται από περισσότερες από 250 διαφορετικές συναρτήσεις κλήσεων, η οποία μπορεί να χρησιμοποιηθεί για την απεικόνιση περίπλοκων τρισδιάστατων σκηνών από απλά γεωμετρικά σχήματα. Η OpenGL αναπτύχθηκε από την Silicon Graphics Inc (SGI) κατά το 1992 και χρησιμοποιείται ευρέως στο CAD (πρόγραμμα σχεδίασης γραφικών), στην εικονική πραγματικότητα, την επιστημονική οπτικοποίηση (scientific visualization), την οπτικοποίηση της πληροφορίας (information visualization), και την προσομοίωση πτήσης. Χρησιμοποιείται επίσης σε video games, όπου ανταγωνίζεται με το Direct3D στις πλατφόρμες της Microsoft Windows. Η OpenGL διαχειρίζεται από μια τεχνολογία μη κερδοσκοπικού χαρακτήρα, το Khronos Group.

Design

OpenGL εξυπηρετεί δύο βασικούς σκοπούς:

1. Απόκρυψη πολυπλοκότητας της διασύνδεσης με τους διάφορους 3D επιταχυντές, παρουσιάζοντας μια ενιαία, ομοιόμορφη διεπαφή.

2. Απόκρυψη διαφορετικών δυνατοτήτων των πλατφορμών hardware (υλικού) , απαιτώντας τη στήριξη του πλήρους πακέτου χαρακτηριστικών της OpenGL για όλες τις εφαρμογές (χρησιμοποιώντας την άμιλλα λογισμικού, εάν είναι αναγκαίο).

Βασική λειτουργία της OpenGL είναι να αποδεχθεί τα primitives, όπως σημεία, γραμμές και πολύγωνα, και να τα μετατρέψει σε pixels (εικονοστοιχεία). Αυτό γίνεται με τον αγωγό γραφικών (graphics pipeline), γνωστό ως κρατική μηχανή της OpenGL (OpenGL state machine). Οι περισσότερες εντολές OpenGL είτε θέτουν primitives στον αγωγό γραφικών, ή ρυθμίζουν τον τρόπο με τον οποίο ο αγωγός επεξεργάζεται αυτά τα primitives. Πριν από την εισαγωγή του OpenGL 2.0, κάθε φάση του αγωγού μπορούσε να εκτελεστεί με μια προκαθορισμένη λειτουργία και υπήρχε δυνατότητα ρύθμισης μόνο εντός στενών ορίων. Η OpenGL 2.0 προσφέρει διάφορα στάδια που είναι πλήρως προγραμματιζόμενα μέσω GLSL.

Η OpenGL είναι ένα χαμηλού επιπέδου, διαδικαστικό API, που απαιτεί από τον προγραμματιστή να υπαγορεύσει τα ακριβή βήματα που απαιτούνται για να καταστεί μια σκηνή. Αυτό έρχεται σε αντίθεση με περιγραφικά API, όπου ένας προγραμματιστής πρέπει μόνο να περιγράψει μια σκηνή και μπορεί να αφήσει τη βιβλιοθήκη να διαχειριστεί τις λεπτομέρειες της παροχής του. Ο χαμηλού επιπέδου σχεδιασμός της OpenGL απαιτεί από τους προγραμματιστές να έχουν καλή γνώση του αγωγού γραφικών, αλλά τους δίνει και μια κάποια ελευθερία να εφαρμόσουν αλγόριθμους.

Η OpenGL ιστορικά έχει μεγάλη επιρροή στην ανάπτυξη των 3D επιταχυντών, προωθώντας ένα επίπεδο βάσεως της λειτουργικότητας που είναι τώρα κοινό στους καταναλωτές σε επίπεδο hardware:

- Rasterised σημεία, γραμμές και πολύγωνα ως βασικά primitives.
- Ο μετασχηματισμός και ο φωτισμός αγωγού
- Z-buffering
- Texture Χαρτογράφηση (texture mapping)
- Ανάμειξη Alpha (Alpha blending)

Μια σύντομη περιγραφή της διαδικασίας στον αγωγό γραφικών θα μπορούσε να είναι:

- Αποτίμηση, εάν χρειάζεται, των συναρτήσεων πολυωνύμων που καθορίζουν ορισμένες εισροές, όπως οι NURBS επιφάνειες, οι καμπύλες προσέγγισης και η γεωμετρία επιφάνειας.

- Vertex πράξεις (με κορυφές), μετατρέποντας και φωτίζοντάς τες ανάλογα με το υλικό τους. Επίσης αποκόβοντας μη ορατά τμήματα της σκηνής, ώστε να παράγει την προβολή του όγκου.

- Rasterisation ή μετατροπή της προηγούμενης ενημέρωσης σε pixels. Τα πολύγωνα που εκπροσωπούνται από το κατάλληλο χρώμα μέσω αλγορίθμων παρεμβολής.

- Ανά τεμάχιο πράξεις, όπως η ενημέρωση τιμών, ανάλογα με τις εισερχόμενες και στο παρελθόν αποθηκευμένες τιμές βάθους, ή συνδυασμούς χρωμάτων, μεταξύ των άλλων.

- Τέλος, εισέρχονται κομμάτια (fragments) στο frame buffer.

Πολλοί σύγχρονοι 3D επιταχυντές παρέχουν λειτουργίες πολύ πιο πάνω από το σενάριο αυτό, αλλά αυτά τα νέα χαρακτηριστικά είναι γενικά αξεσουάρ αυτού του βασικού αγωγού και όχι ριζικές αναθεωρήσεις του.

History

Στη δεκαετία του 1980, η ανάπτυξη λογισμικού που θα μπορούσε να λειτουργήσει με ένα ευρύ φάσμα των γραφικών ήταν μία πραγματική πρόκληση. Οι προγραμματιστές λογισμικού ανέπτυξαν συνηθισμένες διεπαφές και οδηγούς για κάθε κομμάτι του υλικού. Αυτό ήταν ακριβό και είχε ως αποτέλεσμα την μεγάλη αλληλοεπικάλυψη των προσπαθειών.

Από τις αρχές του 1990, η Silicon Graphics (SGI) ήταν πρωτοπόρος στα 3D γραφικά για τους σταθμούς εργασίας. Το IRIS API GL τους, θεωρήθηκε η εξέλιξη της τεχνολογίας και έγινε το βιομηχανικό πρότυπο, επισκιάζοντας τα ανοιχτά βασιζόμενα πρότυπα PHIGS. Αυτό οφείλεται στο γεγονός ότι "IRIS" GL κρίθηκε πιο εύκολο στη χρήση, αλλά και επειδή υποστήριξε την μέθοδο άμεσης απόδοσης (immediate mode rendering). Αντιθέτως, το PHIGS θεωρήθηκε δύσκολο να χρησιμοποιηθεί και ξεπερασμένο από άποψη της λειτουργικότητας.

Οι ανταγωνιστές της SGI (συμπεριλαμβανομένων των Sun Microsystems, της Hewlett-Packard και της IBM) ήταν επίσης σε θέση να διαθέσουν στην αγορά 3D υλικό, υποστηριζόμενες από επεκτάσεις δημιουργημένες στο πρότυπο PHIGS. Αυτό με τη σειρά του προκάλεσε στο SGI μερίδιο της αγοράς να αποδυναμωθεί όσο περισσότεροι προμηθευτές υλικού 3D γραφικών εισέρχονταν στην αγορά. Σε μια προσπάθεια να επηρεάσει την αγορά, η SGI αποφάσισε να μετατρέψει την IrisGL API σε ένα ανοικτό πρότυπο.

Η SGI έκρινε ότι το ίδιο το IrisGL API, δεν ήταν κατάλληλο για το άνοιγμα που οφείλεται σε χορήγηση άδειας και ζητήματα διπλωμάτων ευρεσιτεχνίας. Επίσης, η IrisGL είχε API λειτουργίες που δεν σχετίζονται με 3D γραφικά. Για παράδειγμα, περιελάμβανε ένα windowing, API για πληκτρολόγιο και ποντίκι, εν μέρει επειδή αναπτύχθηκε πριν από το X Window Σύστημα και τα συστήματα NeWS της Sun αναπτύχθηκαν.

Επιπλέον, η SGI είχε έναν μεγάλο αριθμό πελατών λογισμικού. Με την αλλαγή του OpenGL API, προγραμμάτισαν για να διατηρήσουν τους πελάτες τους κλειδωμένους πάνω στο SGI (και IBM) υλικό για μερικά χρόνια, όσο η στήριξη της αγοράς για OpenGL ωρίμασε. Εν τω μεταξύ, η SGI θα συνεχίσει να προσπαθεί να διατηρήσει τους πελάτες της συνδεόμενη με το υλικό SGI μέσω της ανάπτυξης των προηγμένων και ιδιόκτητων Iris Inventor και Iris Performer APIs προγραμματισμού.

Ως αποτέλεσμα, η SGI κυκλοφόρησε το πρότυπο OpenGL.

Η OpenGL τυποποίησε την πρόσβαση στο υλικό, και ώθησε την ευθύνη ανάπτυξης των προγραμμάτων διεπαφής υλικού, που συχνά ονομάζονται οδηγοί συσκευών (device drivers), στους κατασκευαστές hardware και ανέθεσε λειτουργίες windowing στο υποκείμενο λειτουργικό σύστημα. Με τόσα πολλά διαφορετικά είδη γραφικών υλικού και έχοντάς τα όλα να μιλούν την ίδια γλώσσα προγραμματισμού, είχε σημαντικά αποτελέσματα, δίνοντας στους προγραμματιστές ανάπτυξης λογισμικού μια υψηλότερου επιπέδου πλατφόρμα για την ανάπτυξη τρισδιάστατου λογισμικού.

Το 1992, η SGI έφτασε στη δημιουργία της αναθεωρημένης αρχιτεκτονικής OpenGL (OpenGL ARB), η ομάδα των εταιρειών που θα διατηρήσουν και να επεκτείνουν τις OpenGL προδιαγραφές για τα επόμενα χρόνια. Η OpenGL εξέλιξε από (και είναι πολύ παρόμοια σε στυλ με) προηγούμενες 3D διεπαφές της SGI, το IrisGL. Ένας από τους περιορισμούς του IrisGL ήταν ότι παρείχε μόνο την πρόσβαση σε λειτουργίες που υποστηρίζονται από το υποκείμενο υλικό. Εάν το υλικό γραφικών δεν υποστήριζε μια δυνατότητα, τότε η εφαρμογή δεν θα μπορούσε να το χρησιμοποιήσει. Η OpenGL ξεπέρασε αυτό το πρόβλημα με την παροχή υποστήριξης λογισμικού για τα χαρακτηριστικά που δεν υποστηρίζονται από το υλικό, επιτρέποντας τις εφαρμογές να χρησιμοποιούν προηγμένα γραφικά με σχετικά χαμηλής ισχύος συστήματα.

Το 1994, η SGI έπαιξε με την ιδέα να κυκλοφορήσει κάτι που ονομάστηκε «OpenGL++», η οποία

περιελάμβανε στοιχεία, όπως ένα API σκηνή-διάγραμμα (scene-graph API) (προφανώς με βάση την τεχνολογία των Performer τους). Οι προδιαγραφές διανεμήθηκαν μεταξύ λίγων ενδιαφερόμενων - αλλά ποτέ δεν μετατράπηκε σε προϊόν.

Η Microsoft κυκλοφόρησε το Direct3D το 1995, το οποίο θα καταστεί ο κύριος ανταγωνιστής της OpenGL. Στις 17 Δεκεμβρίου, 1997, η Microsoft και η SGI ξεκίνησαν το έργο Φαρενάιτ (Fahrenheit project), που ήταν μια κοινή προσπάθεια με στόχο την ενοποίηση των OpenGL και Direct3D διασυνδέσεων (και την προσθήκη ενός graph-scene API επίσης). Το 1998, η Hewlett-Packard εντάχθηκε στο έργο. Έδειξε αρχικά κάποια υπόσχεση του να βάλει τάξη στον κόσμο των interactive API (αλληλεπιδρόμενων) 3D γραφικών υπολογιστή, αλλά λόγω των οικονομικών περιορισμών στις υπηρεσίες κοινής ωφέλειας στην SGI, στρατηγικών λόγων στη Microsoft, και της γενική έλλειψης υποστήριξης βιομηχανίας , εγκαταλείφθηκε το 1999.

Οι OpenGL κυκλοφορίες είναι ανάδρομα συμβατές. Σε γενικές γραμμές, οι κάρτες γραφικών που κυκλοφόρησαν μετά τις εκδόσεις της OpenGL υποστηρίζουν αυτά τα χαρακτηριστικά έκδοσης, καθώς και όλα τα προηγούμενα χαρακτηριστικά. Για παράδειγμα, η GeForce 6800, υποστηρίζει όλα τα χαρακτηριστικά μέχρι και την έκδοση της OpenGL 2.0. (Ειδικές κάρτες μπορεί να είναι σύμφωνες προς μια ιδιαίτερη έκδοση OpenGL, αλλά επιλεκτικά δεν υποστηρίζουν ορισμένες λειτουργίες).

1.4 Το Xna Game Studio

Το Xna Game Studio, είναι ένα σύνολο από εργαλεία που δημιουργήθηκε για την απλούστερη ανάπτυξη και σχεδίαση παιχνιδιών δύο διαστάσεων (2D) και τριών διαστάσεων (3D). Όπως αναφέρθηκε παραπάνω, ένα βασικό του πλεονέκτημα είναι ότι τα παιχνίδια που δημιουργούνται μέσω του Xna, είναι συμβατά με διαφορετικές πλατφόρμες (Windows, Xbox 360, Zune). Το Xna περιλαμβάνει επίσης το Xna Framework, το οποίο είναι ένα σύνολο από .NET βιβλιοθήκες, βασιζόμενες στο Microsoft .Net Framework 2.0 που σχεδιάστηκε για την ανάπτυξη παιχνιδιών.
Η πρώτη ολοκληρωμένη έκδοση του Xna Game Studio Express που παρουσιάστηκε από τη Microsoft με δωρεάν λήψη, ήταν στις 30 Αυγούστου του 2006 ενώ ακολούθησε η επόμενη έκδοση την 1η Νοεμβρίου του 2006. Η Microsoft παρείχε την τελική έκδοση την 11η Δεκεμβρίου του 2006.
Στις 24 Απριλίου του 2007, η Microsoft παρουσίασε την ενημερωμένη έκδοση, ονομαζόμενη ως Xna Game Studio Express 1.0 Refresh.
Ακολούθησαν:

- **Xna Game Studio 2.0**, στις 13 Δεκεμβρίου του 2007, το οποίο χαρακτηρίζεται από την ικανότητά του να χρησιμοποιείται από όλες τις εκδόσεις του Visual Studio 2005 (συμπεριλαμβανομένου του Visual C# 2005 Express Edition) και είναι διαθέσιμο με δωρεάν λήψη από το Xna Creators Club website. Επίσης παρέχει τη δυνατότητα χειρισμού του Xbox Live από το λειτουργικό σύστημα των Windows και της κονσόλας Xbox 360.

- **Xna Game Studio 3.0** (για το Visual Studio 2008 ή το Visual C# 2008 Express Edition). Επιτρέπει στην παραγωγή παιχνιδιών να στοχεύουν στην πλατφόρμα Zune της Microsoft. Η τελική έκδοση ελευθερώθηκε στις 30 Οκτωβρίου του 2008. Το Xna Game Studio 3.0, τώρα υποστηρίζει τη C# 3.0, LINQ, όπως και τις περισσότερες εκδόσεις του Visual Studio 2008. Παρέχει και ακόμα περισσότερες νέες δυνατότητες, όπως μια δοκιμαστική μέθοδο που επιτρέπει στους δημιουργούς να προσθέτουν εύκολα τα χαρακτηριστικά στα παιχνίδια τους, υποστήριξη πολλαπλών παικτών και δημιουργίας παιχνιδιών που δουλεύουν σε διαφορετικές πλατφόρμες που είναι συμβατές με το λειτουργικό σύστημα των Windows, το Xbox 360 όπως επίσης και το Zune.

- **Xna Game Studio 3.1**, κυκλοφόρησε την 11η Ιουνίου το 2009. Η εφαρμογή υποστηρίζει αναπαραγωγή βίντεο (video playback), δίνει στους παίκτες τη δυνατότητα επικοινωνίας ακόμα κι αν κάποιος από αυτούς δεν παίζει το ίδιο παιχνίδι (Xbox Live Party), παρέχει υποστήριξη για XACT3 (εργαλείο δημιουργίας ήχου) με νέες δυνατότητες, όπως την ενεργοποίηση φίλτρου σε κάθε κομμάτι μουσικής, παρέχει ένα αναθεωρημένο Audio Api, επίσης δίνει τη δυνατότητα στα παιχνίδια να χρησιμοποιούν Xna Avatars, δηλαδή εικόνες που αντιπροσωπεύουν τους παίκτες.

- **Xna Game Studio 4.0**, ανακοινώθηκε τον Μάρτιο του 2010. Παρέχει υποστήριξη για την ανάπτυξη παιχνιδιών στο Windows Phone 7 και περισσότερες ακόμα δυνατότητες όπως θα δούμε και πιο αναλυτικά παρακάτω.

1.5 Το Xna, το DirectX και η OpenGL

Αν ψάξει κανείς στον παγκόσμιο ιστό για εκπαιδευτικά έγγραφα (tutorials) ανάπτυξης βίντεο παιχνιδιού, θα ανακαλύψει πλήθος επιλογών. Αυτές οι επιλογές κυμαίνονται από το DirectX, το Xna, την OpenGL, μέχρι απλούς κατασκευαστές παιχνιδιών. Αν τίθεται λοιπόν το ερώτημα, ποια είναι η καλύτερη, μέσα από αυτήν την τεράστια γκάμα, για να ξεκινήσει κανείς να ασχολείται με τον προγραμματισμό και την ανάπτυξη παιχνιδιών, υπάρχουν τρεις σημαντικές επιλογές από τις οποίες μπορεί να διαλέξει.
Το Xna είναι σίγουρα μια πολύ καλή αρχή για κάποιον ο οποίος μόλις ξεκίνησε να ασχολείται με το αντικείμενο. Το DirectX, χρησιμεύει σε κάποιον ο οποίος έχει μεγαλύτερους στόχους όπως αυτός του να εργαστεί στη βιομηχανία δημιουργίας ηλεκτρονικών παιχνιδιών, όπως επίσης και η OpenGL, η οποία είναι μια καλή επιλογή, αφού υποστηρίζει και πολλές πλατφόρμες (cross-platform solution).
Πιο αναλυτικά, το Xna είναι ίσως το καλύτερο ξεκίνημα για κάποιον ερασιτέχνη ο οποίος θέλει να μάθει να αναπτύσσει ηλεκτρονικά παιχνίδια. Είναι απλό στη χρήση του και ακόμα πιο εύκολο στην εκμάθηση και χρήση της γλώσσας προγραμματισμού που χρησιμοποιεί (C#). Παρέχει επιπλέον την σημαντική δυνατότητα να είναι συμβατό με την κονσόλα Xbox 360. Αυτό σημαίνει πως οι χρήστες μπορούν εύκολα να προωθήσουν το έργο τους στους φίλους τους, την οικογένειά τους που έχουν την κονσόλα επιτρέποντας ταυτόχρονα στο παιχνίδι να χρησιμοποιείται στα Windows, με πολύ λίγες αλλαγές στον κώδικα. Η πλατφόρμα αυτή, υποστηρίζει και την συνεργασία με το Zune της Microsoft.
Το DirectX έχει το μειονέκτημα μιας απότομης καμπύλης εκμάθησης, λόγω της απαίτησής του να χρησιμοποιεί την γλώσσα προγραμματισμού C++. Το DirectX λοιπόν, απαιτεί από τους προγραμματιστές να έχουν μια σταθερή γνώση της C++, των δεικτών και της διαχείρισης μνήμης. Παρ' όλα αυτά, το DirectX και η C++ είναι οι πιο συχνά χρησιμοποιούμενες τεχνολογίες σε ένα στούντιο ανάπτυξης παιχνιδιών.
Η OpenGL, έχει το μεγάλο πλεονέκτημα να δουλεύει σε διαφορετικές πλατφόρμες. Συνεργάζεται με το λειτουργικό σύστημα των Linux, Windows και Mac. Χρησιμοποιεί την C++ γλώσσα προγραμματισμού, όπως και το DirectX, όμως δεν έχει την ίδια απήχηση όσο αυτό, στην βιομηχανία ανάπτυξης παιχνιδιών. Ωστόσο, και τα δύο αποτελούν πολύ καλή επιλογή όσον αφορά στην ανάπτυξη παιχνιδιών τριών διαστάσεων.

1.6 Λογισμικά που χρησιμοποιήθηκαν στην εργασία

- **Microsoft Visual Studio και .Net Framework:** Το Visual Studio, είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού (IDE), όπου οι προγραμματιστές μπορούν να αναπτύξουν και να υλοποιήσουν τις εφαρμογές τους σε κάποια από τις πολλές γλώσσες προγραμματισμού, συμπεριλαμβανομένης και της Visual C#, για το .Net Framework. Το .Net Framework , είναι το περιβάλλον ανάπτυξης και εκτέλεσης, που επιτρέπει σε διαφορετικές γλώσσες προγραμματισμού και βιβλιοθήκες να συνεργάζονται και να δημιουργούν εφαρμογές για περιβάλλοντα Windows, Web, Windows Mobile, Windows CE και Windows Silverlight.

- **Visual C# (C Sharp):** Πρόκειται για μια αντικειμενοστραφή (object-oriented), απλή όμως ισχυρή γλώσσα προγραμματισμού, η οποία επιτρέπει στους προγραμματιστές να δημιουργήσουν ένα εύρος εφαρμογών. Συνδυαζόμενη με το .Net Framework, η Visual C# χρησιμοποιείται για τη δημιουργία εφαρμογών Windows, υπηρεσίες Web, εργαλεία βάσεων δεδομένων (database tools) και άλλα. Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε το Visual C# 2008 Express Edition.

- **3D Studio Max:** Το 3D Studio Max είναι ένα από τα ισχυρότερα προγράμματα δημιουργίας και επεξεργασίας τρισδιάστατων γραφικών που υπάρχουν σήμερα. Χρησιμοποιείται σε μια μεγάλη ποικιλία εμπορικών και καλλιτεχνικών εφαρμογών, όπως η αρχιτεκτονική, η παραγωγή ταινιών, η διαφήμιση, η οπτική απεικόνιση ιατρικών και επιστημονικών μοντέλων, οι καλές τέχνες καθώς και τα παιχνίδια ηλεκτρονικών υπολογιστών. Μας δίνει τη δυνατότητα να δημιουργήσουμε βασικά αντικείμενα όπως είναι οι κύβοι, κώνοι, σφαίρες κτλ, με καθορισμένο πλάτος, μήκος και ύψος των οποίων τις τιμές μπορούμε να ρυθμίσουμε με μεγάλη ακρίβεια. Για την δημιουργία των μοντέλων που χρησιμοποιήθηκαν στην υλοποίηση της εφαρμογής, χρησιμοποιήθηκε το 3D Studio Max 9.0.

- **Adobe Photoshop:** Το Photoshop είναι το γνωστό σε όλους πρόγραμμα επεξεργασίας γραφικών και στην συγκεκριμένη εργασία στάθηκε χρήσιμο για την ολοκλήρωση της δημιουργίας του heightmap του τρισδιάστατου κόσμου και την επεξεργασία των υπόλοιπων υφών.

1.7 Εξωγενείς παράγοντες

Για την δημιουργία μιας τρισδιάστατης εφαρμογής, χρειάζονται μια συλλογή από τρισδιάστατα μοντέλα, υφες (textures) που ίσως χρειαστούν για να καλύψουν τα μοντέλα, ήχους και άλλα. Ένα μέρος από αυτά δημιουργήθηκε με τη βοήθεια προγράμματος σχεδίασης ενώ τα υπόλοιπα συλλέχτηκαν μέσω του παγκόσμιου ιστού.

1.8 Δομή της εργασίας

Η εργασία αυτή παρουσιάζει τις έννοιες της ανάπτυξης τρισδιάστατων παιχνιδιών Xna, μέσω της πλοήγησης του αναγνώστη σε μια εφαρμογή τριών διαστάσεων:

Κεφάλαιο 1, Εισαγωγή.
Μια σύντομη αναδρομή στην ανάπτυξη παιχνιδιών, τα στάδια ανάπτυξης της πλατφόρμας Xna, τα λογισμικά εργαλεία που χρησιμοποιήθηκαν και το υλικό που συλλέχθηκε για να βοηθήσεις στην ανάπτυξη της εφαρμογής.

Κεφάλαιο 2, Xna Game Studio.
Παρουσιάζει ειδικότερα το Xna Game Studio, την αρχιτεκτονική του συστήματος και τις απαιτήσεις του για την ανάπτυξη ενός τρισδιάστατου παιχνιδιού. Περιγράφει τη δομή μιας εφαρμογής.

Κεφάλαιο 3, Γραφικά.
Μια σύντομη εισαγωγή στο εργαλείο επεξεργασίας τρισδιάστατων γραφικών που χρησιμοποιήθηκε για τη δημιουργία του υλικού και πώς λειτουργεί μια έκταση εδάφους ως heightmap.

Κεφάλαιο 4, Αναπαράσταση γραφικών και ήχων.
Περιγράφει πως γίνεται η προσθήκη του γραφικού υλικού στο Content Pipeline, πώς γίνεται η αναπαράσταση ενός τρισδιάστατου αντικειμένου στη σκηνή, πως λειτουργέι μια κάμερα τρίτου προσώπου και πως αναπαράγουμε ηχητικά εφέ στον τρισδιάστατο κόσμο.

Κεφάλαιο 5, Collision Detection.
Περιγράφει τρόπους εισαγωγής φυσικής αντίδρασης στα αντικείμενα του τρισδιάστατου κόσμου και προτείνει και άλλες μηχανές φυσικής.

Κεφάλαιο 6, Αλληλεπίδραση.
Παρουσιάζει τους τρόπους που ένας χρήστης μπορεί να χρησιμοποιήσει για αν έχει αλληλεπίδραση με την εφαρμογή.
Κεφάλαιο 7, Πρακτικό μέρος.
Σε αυτή την ενότητα, θα εξηγηθούν ενδεικτικά κομμάτια κώδικα, ο ρόλος των μεταβλητών και ο τρόπος που επιδρούν οι διάφορες λειτουργίες στον αλγόριθμο που υλοποιεί την εφαρμογή. Εξηγεί περισσότερο για τη φυσική συμπεριφορά των μοντέλων και την κίνηση στην τρισδιάστατη σκηνή.

Κεφάλαιο 8, Γνωριμία με το Xna Game Studio 4.0.
Μια σύντομη εισαγωγή με την τελευταία έκδοση του Xna της Microsoft.

Κεφάλαιο 9, Βιβλιογραφία
Οι πηγές που χρησιμοποιήθηκαν για να βοηθήσουν στην ανάπτυξη της εφαρμογής.

Αναφορά:
Wikipedia (http://en.wikipedia.org)

2

Xna Game Studio

Σε αυτή την ενότητα θα μιλήσουμε ειδικότερα για το Xna Game Studio και τα μέρη από τα οποία αποτελείται. Όπως είπαμε και στο προηγούμενο κεφάλαιο, το Xna είναι ένα σύνολο από εργαλεία με ένα οργανωμένο περιβάλλον χρόνου εκτέλεσης που παρέχεται από τη Microsoft. Ας το δούμε λοιπόν αναλυτικότερα.

2.1 Παρουσίαση

XNA Framework

Το XNA Framework βασίζεται στην εγγενή εφαρμογή του .NET Compact Framework 2.0 για την ανάπτυξη του Xbox 360 και του .NET Framework 2.0 για τα Windows. Περιλαμβάνει ένα εκτεταμένο σύνολο από βιβλιοθήκες, ειδικές με την ανάπτυξη του παιχνιδιού, για να προωθήσει την επαναχρησιμοποίηση του κώδικα σε όλες τις πλατφόρμες που στοχεύει. Το framework εκτελείται σε μια έκδοση του Χρόνου εκτέλεσης Κοινής Γλώσσας που είναι βελτιστοποιημένη για το παιχνίδι να παρέχει ένα διαχειριζόμενο περιβάλλον εκτέλεσης. Ο χρόνος εκτέλεσης είναι διαθέσιμος για τα Windows XP, Windows Vista και το Xbox 360. Δεδομένου ότι XNA παιχνίδια γράφτηκαν για το χρόνο εκτέλεσης, μπορούν να εκτελεστούν σε οποιαδήποτε πλατφόρμα που υποστηρίζει το XNA Framework με ελάχιστη ή όχι τροποποίηση.
Τα παιχνίδια που εκτελούνται στο framework είναι τεχνικά δυνατόν να είναι γραμμένα σε οποιαδήποτε συμβατή γλώσσα .NET, αλλά μόνο η C#και το XNA Game Studio Express IDE και όλες οι εκδόσεις τουVisual Studio 2005 υποστηρίζονται επίσημα.
Για το λόγο αυτό, το XNA Framework συμπυκνώνει χαμηλού επιπέδου τεχνολογικές λεπτομέρειες που εμπλέκονται στην κωδικοποίηση ενός παιχνιδιού, βεβαιώνοντας ότι το ίδιο το framework φροντίζει τη διαφορά ανάμεσα στις πλατφόρμες, όταν τα παιχνίδια εισάγονται από μια συμβατή πλατφόρμα σε μια άλλη και κατά συνέπεια επιτρέπει στους προγραμματιστές του παιχνιδιού να επικεντρωθούν περισσότερο στο περιεχόμενο και την εμπειρία του παιχνιδιού. Το XNA Framework ενσωματώνεται με έναν αριθμό από εργαλεία, όπως το XACT, για τις ενισχύσεις στη δημιουργία περιεχομένου. Το XNA Framework παρέχει υποστήριξη τόσο για 2D όσο και για 3D δημιουργία παιχνιδιού και επιτρέπει τη χρήση των controllers του Xbox 360 και δονήσεις. Τα παιχνίδια XNA framework που στοχεύουν στην πλατφόρμα Xbox μπορούν, μόνο προς το παρόν να διανεμηθούν στα μέλη της Λέσχης της Microsoft XNA για δημιουργούς (Xna Creator's Club). Οι Desktop εφαρμογές μπορούν να διανέμονται δωρεάν κάτω από την τρέχουσα αδειοδότηση της Microsoft.

XNA Build

Το XNA Build είναι ένα σύνολο εργαλείων διαχείρισης των πόρων (asset pipeline) του παιχνιδιού που βοηθούν με τον καθορισμό, τη διατήρηση, τη διόρθωση, και τη βελτιστοποίηση των πόρων του των ατομικών αναπτυξιακών προσπαθειών στο παιχνίδι.

Ένα asset pipeline (αγωγός πόρων) του παιχνιδιού περιγράφει τη διαδικασία με την οποία το περιεχόμενό του, όπως υφές και 3D μοντέλα, έχουν τροποποιηθεί σε μορφή κατάλληλη για χρήση από τη μηχανή του παιχνιδιού. Το XNA Build συμβάλλει στον εντοπισμό των εξαρτήσεων του αγωγού, και επίσης παρέχει πρόσβαση στο API για να ενεργοποιήσει την περαιτέρω επεξεργασία των εξαρτώμενων δεδομένων.
Τα εξαρτώμενα δεδομένα μπορούν να αναλυθούν για να βοηθήσουν στη μείωση του μεγέθους ενός παιχνιδιού με την εύρεση περιεχομένου που δεν χρησιμοποιείται στην πραγματικότητα. Για παράδειγμα, η XNA Build ανάλυση αποκάλυψε ότι το 40% των υφών που συνοδεύεται μαζί με τον MechCommander 2 ήταν αχρησιμοποίητο και θα μπορούσε να είχε παραλειφθεί.

XNA Game Studio

Το XNA Game Studio είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για την ανάπτυξη των παιχνιδιών. Έξι αναθεωρήσεις έχουν κυκλοφορήσει μέχρι σήμερα.

XNA Game Studio Professional

Το XNA Game Studio Professional ήταν μια προγραμματισμένη έκδοση του XNA IDE με στόχο τους επαγγελματίες προγραμματιστές παιχνιδιών. Με βάση το Visual Studio 2005 Team System, το Xna Studio παρέχει μια δομή συνεργασίας μεταξύ των δημιουργών του περιεχομένου. Οι εργασίες διαχείρισης έργων, όπως η διαχείριση των πόρων, ο εντοπισμός ελαττωμάτων, η αυτοματοποίηση του έργου και οι λίστες του των αντικειμένων δουλειάς, είναι κάπως αυτοματοποιημένες από το XNA Studio.
Το XNA Game Studio Professional δεν είναι πλέον υπό ενεργή ανάπτυξη.

XNA Game Studio Express

Το XNA Game Studio Express απευθύνεται σε φοιτητές, ερασιτέχνες, και ανεξάρτητους σχεδιαστές παιχνιδιών. Είναι διαθέσιμο με δωρεάν λήψη. Το Express θα προσφέρει τις βασικές γνώσεις για την ταχεία ανάπτυξη σε συγκεκριμένα είδη των παιχνιδιών, όπως η πλατφόρμα, στρατηγική πραγματικού χρόνου, καθώς και first-person shooters (πρώτου προσώπου πυροβολητές). Οι προγραμματιστές μπορούν να δημιουργήσουν Windows παιχνίδια δωρεάν με το XNA Framework, αλλά για να εκτελέσουν τα παιχνίδια τους για το Xbox 360 θα πρέπει να πληρώσουν με το ετήσιο τέλος για την πρόσβαση στο Microsoft XNA Creator’s Club/XNA «Creator’s Club".

XNA Game Studio 2.0

Το XNA Game Studio 2.0 κυκλοφόρησε στις 13 Δεκεμβρίου του 2007. Το XNA Game Studio 2.0, έχει την ικανότητα να χρησιμοποιείται με όλες τις εκδόσεις του Visual Studio 2005 (συμπεριλαμβανομένων της δωρεάν Visual C# 2005 Express Edition), με ένα δικτυωμένο API χρησιμοποιώντας το Xbox Live και στα Windows και στο Xbox 360 και να προσφέρει καλύτερο χειρισμό συσκευής. Είναι επίσης διαθέσιμο για δωρεάν λήψη στην ιστοσελίδα του XNA Creator’s Club.

XNA Game Studio 3.0

Το XNA Game Studio3 .0 (για το Visual Studio 2008 ή τη δωρεάν Visual C# 2008 Express Edition), επιτρέπει την παραγωγή παιχνιδιών με στόχο την πλατφόρμα Zune και προσθέτει την υποστήριξη της κοινότητας του Xbox Live .
Μια έκδοση των εργαλείων κυκλοφόρησε το Σεπτεμβρίου 2008. Η τελική έκδοση κυκλοφόρησε στις 30 Οκτωβρίου του 2008. Το XNA Game Studio 3.0 τώρα υποστηρίζει τη C# 3.0, τη LINQ και τις περισσότερες εκδόσεις τουVisual Studio 2008.
Υπάρχουν αρκετά ακόμα νέα χαρακτηριστικά του XNA Game Studio 3.0, όπως μια δοκιμαστική λειτουργία που προστέθηκε στο XNA Game Studio 3.0 που θα επιτρέψει στους δημιουργούς να προσθέτουν εύκολα τα

απαιτούμενα χαρακτηριστικά για τα παιχνίδια τους, υποστήριξη χαρακτηριστικών για συμμετοχή πολλών παικτών του Xbox LIVE , όπως τα in-game invites (πρόσκληση για παιχνίδι), να δημιουργήσει cross-platform παιχνίδια που λειτουργούν στα Windows, στο Xbox 360 και στο Zune.

XNA Game Studio 3.1

XNA Game Studio 3.1 ανακοινώθηκε στο συνέδριο δημιουργών παιχνιδιών (Game Developers Conference) στο San Fransico στις 24 Μαρτίου 2009.
Η εφαρμογή είναι για να περιλαμβάνουν υποστήριξη για αναπαραγωγή βίντεο, ένα αναθεωρημένο API ήχου, Xbox LIVE Party σύστημα και την υποστήριξη για τα παιχνίδια με χρήση Xbox 360 Avatars (είδωλα).

XNA Game Studio 4.0

Το Xna Game Studio 4.0 μόλις κυκλοφόρησε από τη Microsoft. Τώρα υπάρχει μια μεγάλη γκάμα από πλατφόρμες ανάπτυξης ηλεκτρονικών παιχιδιών. Η ανάπτυξή τους πια θα είναι συμβατή με το λειτουργικό σύστημα των Windows, της κονσόλας Xbox 360 αλλά και του Windows Phone 7. Μια αρχιτεκτονική για την ανάπτυξη όλων.

XNA Content Pipeline

ToXNA Framework Content Pipeline είναι ένα σύνολο εργαλείων που θεωρείται "το σημείο σχεδιασμού, κλειδί γύρω από την οργάνωση και την κατανάλωση 3D περιεχομένου ".
Αυτό σημαίνει ότι το XNA Game Studio μπορεί ακόμη να χρησιμοποιηθεί για την ανάπτυξη των εμπορικών παιχνιδιών και άλλων προγραμμάτων για την πλατφόρμα των Windows, αν και στην υποστήριξη του κώδικα δικτύωσης της Microsoft για το Xbox/Windows Live, δεν μπορεί να χρησιμοποιηθεί.
Η αυτο-ανάπτυξη κώδικα του δικτύου μπορεί ακόμη να χρησιμοποιηθεί μέσα στο XNA έργο σας.
Παιχνίδια που δημιουργήθηκαν με το XNA Game studio μπορεί πλέον να διανέμονται μέσω του Xbox Live Community Games.Το λογισμικό μπορεί επίσης να χρησιμοποιηθεί για τη δημιουργία εμπορικών παιχνιδιών το οποία στοχεύουν στα Windows.
Θα μιλήσουμε αναλυτικότερα για το Content Pipeline αργότερα.

XNA Community Games

Τα παιχνίδια Xbox 360 που είναι ανεπτυγμένα στο XNA Game Studio, μπορεί να υποβάλλονται στην Creator's Club κοινότητα, για το οποίο χρειάζεται κάποιο κόστος για τα συμμετοχή ως Premium μέλος. Όλα τα παιχνίδια που υποβάλλονται στην κοινότητα υπόκεινται σε αξιολόγηση από άλλους ομότιμους δημιουργούς. Αν το παιχνίδι περνάει από την αξιολόγηση, τότε εισάγεται στην λίστα του Xbox Live Marketplace.
Οι δημιουργοί μπορούν να ορίσουν μια τιμή των 200, 400 ή 800 βαθμών για το παιχνίδι τους. Ο δημιουργός έχει το 70% από τα συνολικά έσοδα από τις πωλήσεις του παιχνιδιού του ως αφετηρία. Η Microsoft αρχικά είχε προβλεφθεί να λάβει ένα επιπλέον ποσοστό των εσόδων εάν προβλέπεται πρόσθετη κυκλοφορία για ένα παιχνίδι, αλλά αυτή η πολιτική ανακλήθηκε το Μάρτιο του 2009.
Η Microsoft διανέμει επίσης δοκιμαστικούς λογαριασμούς (trial accounts) για εκπαιδευτικές εγκαταστάσεις μέσω του προγράμματος DreamSpark.
Οι λογαριασμοί αυτοί επιτρέπουν στους μαθητές να αναπτύξουν παιχνίδια για την κονσόλα Xbox 360, αλλά ένας premium λογαριασμός εξακολουθεί να απαιτείται για την υποβολή του παιχνιδιού στην αγορά (Marketplace).

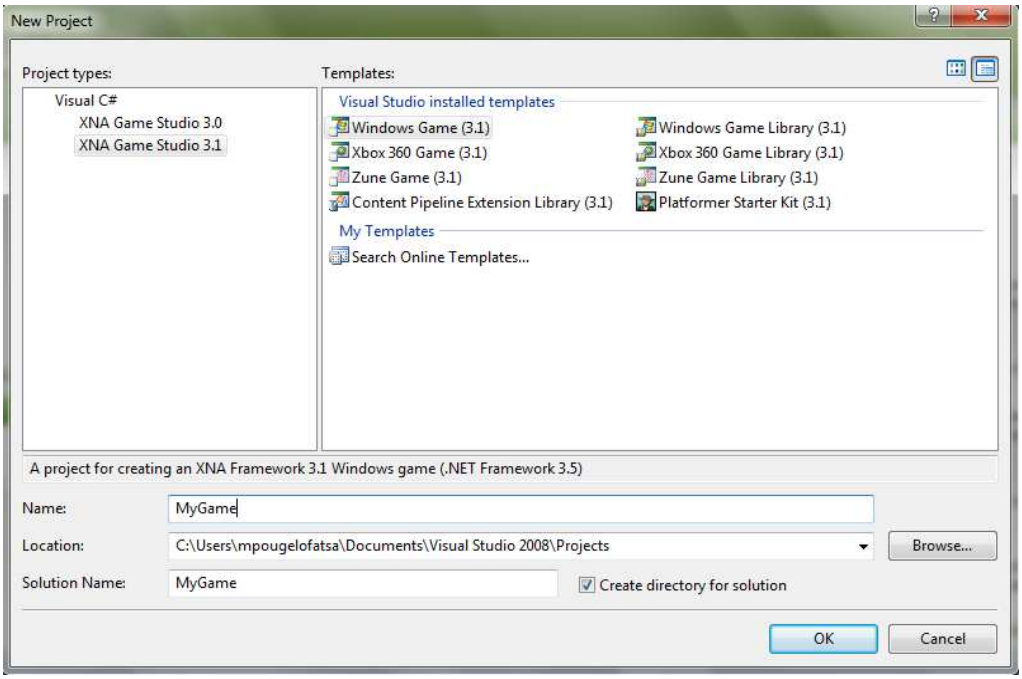
Εναλλακτικές υλοποιήσεις

Ένα έργο που ονομάζεται Mono.XNA σχηματίστηκε για την εισαγωγή του XNA στο open source και cross-platform Mono framework.

2.2 Αρχιτεκτονική Συστήματος

Αφού έχει γίνει η εγκατάσταση του Xna Game Studio, ανοίγοντας το Visual Studio παρατηρούμε πως έχει κάποιες επιλογές για το ξεκίνημα μιας νέας εφαρμογής. Δημιουργείστε ένα νέο αρχείο όπου θα γίνει η ανάπτυξη της εφαρμογής σας, επιλέγοντας File → New Project. Παρατηρείτε ότι εμφανίζεται ένα παράθυρο με επιλογές (Εικ. 1). Σε αυτό το παράθυρο, επιλέγουμε το Windows Game (3.1). Όπως βλέπετε, στην αριστερή πλευρά του παραθύρου υπάρχει η επιλογή της φόρμας που θα χρησιμοποιήσετε, ενώ στην δεξιά πλευρά, ένα πλήθος επιλογών για το είδος της εφαρμογής που θα δημιουργήσετε. Σε αυτήν την περίπτωση επιλέγετε όπως είπαμε και προηγουμένως, το Windows Game (3.1). Δώστε ένα όνομα στο παιχνίδι σας, επιλέξτε το φάκελο που θέλετε να αποθηκευτεί και πατήστε OK.
Οι φόρμες που παρέχει το Xna Game Studio για την ανάπτυξη παιχνιδιών Xna Framework για τα Windows είναι οι ακόλουθες:

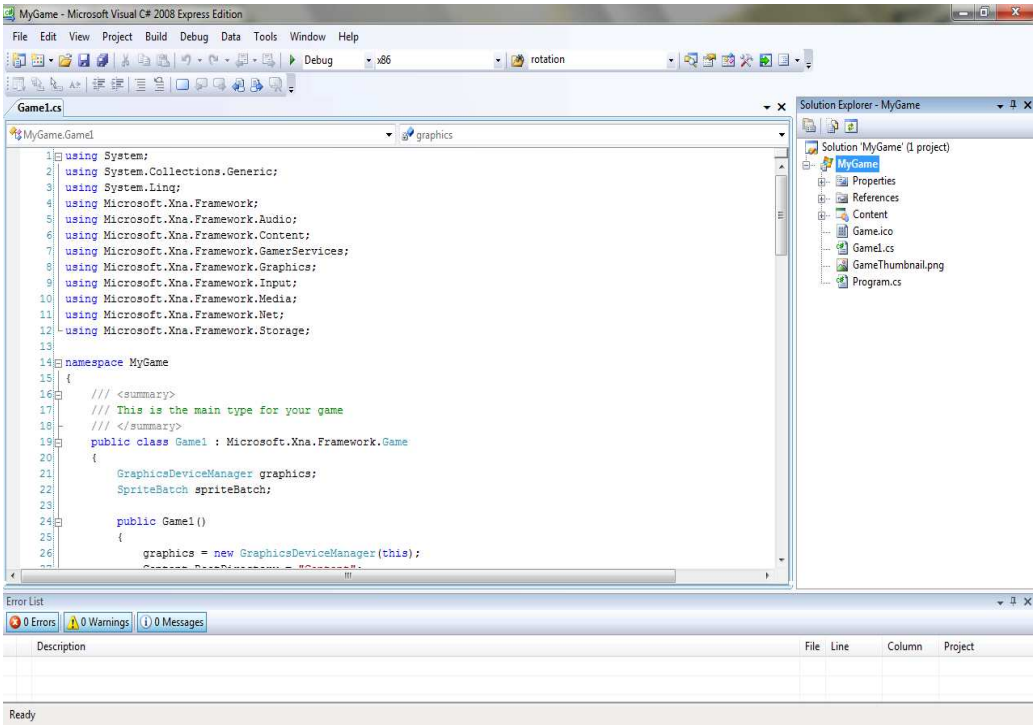
- Windows Game (3.1) – project για την δημιουργία μιας Xna Framework 3.1 εφαρμογής για περιβάλλον Windows.
- Windows Game Library (3.1) – project για την δημιουργία μιας Xna Framework 3.1 βιβλιοθήκης παιχνιδιού για περιβάλλον Windows.
- Content Pipeline Extension Library (3.1) – project για την δημιουργία μιας βιβλιοθήκης Xna Framework 3.1 Content Pipeline Extension.



Εικ. 1 Δημιουργία νέου Project στο Xna

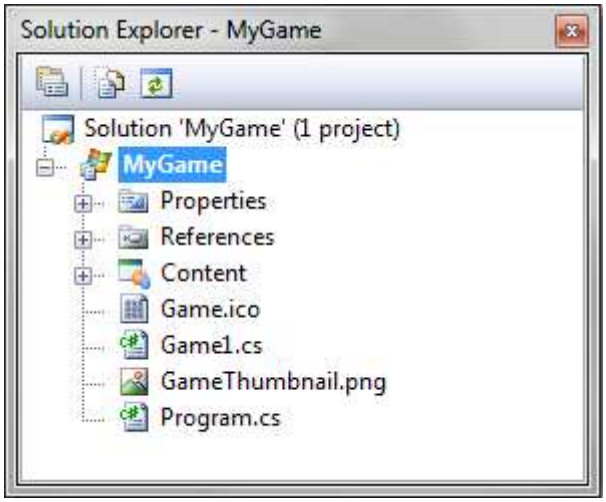
Το περιβάλλον του Xna που θα αναπτύξετε την εφαρμογή σας φαίνεται στην εικόνα 2. Το Xna Game Studio παρέχει μια φόρμα Windows Game (3.1) η οποία δημιουργεί ένα σύνολο αρχείων για το ξεκίνημα της

εφαρμογής. Αυτό το νέο project που μόλις δημιουργήσατε, περιλαμβάνει τον βασικό κώδικα που εκτελεί ένα απλό παράθυρο με χρωματισμένο φόντο.



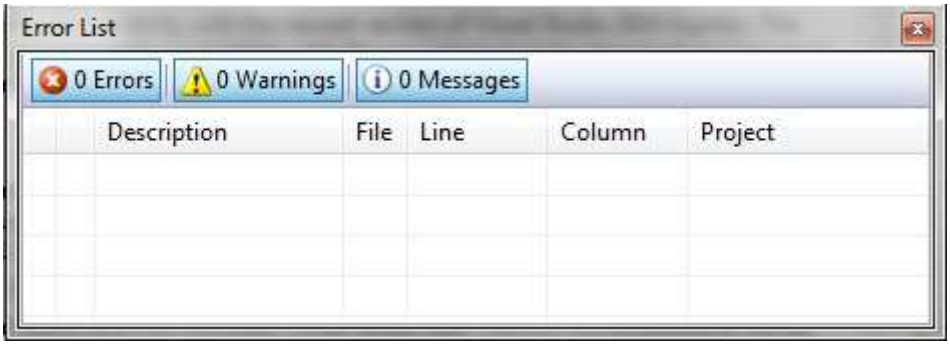
Εικ. 2 Περιβάλλον ανάπτυξης μιας Xna εφαρμογής

Το νέο project περιλαμβάνει σημαντικές ιδιότητες. Δεξιά από το βασικό παράθυρο που θα εισάγετε τον κώδικα για το παιχνίδι σας, υπάρχει ένα παράθυρο που ονομάζεται Solution Explorer και φαίνεται στην εικόνα 3. Αυτές είναι ουσιαστικά οι ιδιότητες όλου του παιχνιδιού. Αυτές οι ιδιότητες ελέγχουν πολλές όψεις του παιχνιδιού σας. Όπως για παράδειγμα, περιλαμβάνουν το γενικό υπόβαθρο μιας εφαρμογής, τις ρυθμίσεις για τα σφάλματα που ενδεχομένως προκύψουν ή επιπλέον πηγές για το project.
Προσέξτε τα αρχεία που δημιουργήθηκαν. Ένα εικονικό αρχείο (Game.ico), ένα αρχείο thumbnail (Game Thumbnail.png), όπως επίσης ένα αρχείο Program.cs και ένα Game1.cs. Αυτά δημιουργούνται εξ' ορισμού από το Xna με τη δημιουργία ενός νέου Project. Το αρχείο ico δημιουργείται για ένα παιχνίδι σε Windows πλατφόρμα ενώ το αρχείο thumbnail για πλατφόρμα Xbox 360.



Εικ. 3 Ο Solution Explorer

Μέσα στον Solution Explorer μπορείτε να εισάγεται νέες κλάσεις, αντικείμενα, μοντέλα, βιβλιοθήκες, αρχεία γραμματοσειρών και άλλα που δημιουργείτε εσείς οι ίδιοι ή που υπάρχουν σε κάποιο φάκελο. Το Content είναι ο χώρος που εισάγεται όλο το δημιουργικό σας υλικό που θα χρησιμοποιηθεί για την ολοκλήρωση του παιχνιδιού σας, όπως μοντέλα, εικόνες, ήχους και άλλα. Θα μιλήσουμε για το Content πιο αναλυτικά στα επόμενα κεφάλαια.
Στο κάτω μέρος του παραθύρου του προγράμματος, υπάρχει ένα παράθυρο που ονομάζεται Error List. Όπως καταλαβαίνετε, αυτό είναι το σημείο που εμφανίζονται όλα τα λάθη και οι προειδοποιήσεις που υπάρχουν στον κώδικα ή στην εισαγωγή κάποιου εξωτερικού αρχείου.



Εικ. 4 Λίστα λαθών

Πατήστε το Play Button  από το μενού εντολών, ή πατήστε F5 για να δείτε το αποτέλεσμα (Εικ. 5).

Εικ. 5 Σκηνή του παιχνιδιού

2.3 Απαιτήσεις Συστήματος

Για τη δημιουργία και την εκτέλεση ηλεκτρονικών παιχνιδιών χρειάζονται τα εξής:

- Ένας ηλεκτρονικός υπολογιστής με λειτουργικό σύστημα Windows και υποστήριξη από την κάρτα γραφικών για τα τρισδιάστατα γραφικά, αν θέλετε να εκτελέσετε τα Xna παιχνίδια σας στον υπολογιστή.
- Τουλάχιστον Windows XP Service Pack 2, ή Windows Vista ή Windows 7.
- Το Microsoft Visual Studio 2008 C# Express Edition ή το Visual Studio 2010 C# Express Edition.
- Το Xna Game Studio 3.1 ή το Xna Game Studio 4.0.
- Αν θέλετε να δοκιμάσετε τα παιχνίδια σας σε κάποια άλλη κονσόλα, όπως είναι το Xbox 360 ή το Zune media player, θα χρειαστείτε και τον ανάλογο εξοπλισμό που θα συνδεθεί με τον υπολογιστή σας.

Σ' αυτή την εργασία χρησιμοποιήθηκε το Xna Game Studio 3.1, το οποίο χρειάζεται για την ανάπτυξη του παιχνιδιού την εγκατάσταση του Visual Studio 2008 C# Express Edition, του Microsoft Xna Game Studio 3.1 και του .NET Framework. Οι προγραμματιστές, όπως προαναφέρθηκε, μπορούν να χρησιμοποιήσουν τις εξής πλατφόρμες: Windows Vista, Windows 7, Windows XP, Xbox 360 και το Zune της Microsoft.

2.4 Εγκατάσταση

Στην εργασία αυτή, όπως είπαμε και προηγουμένως, χρησιμοποιήσαμε το Xna Game Studio 3.1. Για την εγκατάστασή του, χρειάζεται πρώτα να εγκαταστήσουμε το Visual Studio 2008 C# Express Edition. Τα βήματα για την ολοκλήρωση της εγκατάστασης του Xna είναι τα εξής:

1. Εγκατάσταση του Visual Studio C# 2008 Express Edition, ή του Visual Studio 2008 Standard Edition ή υψηλότερα.

2. Λήψη των τελευταίων ενημερώσεων για το Visual Studio από τη Microsoft Update.

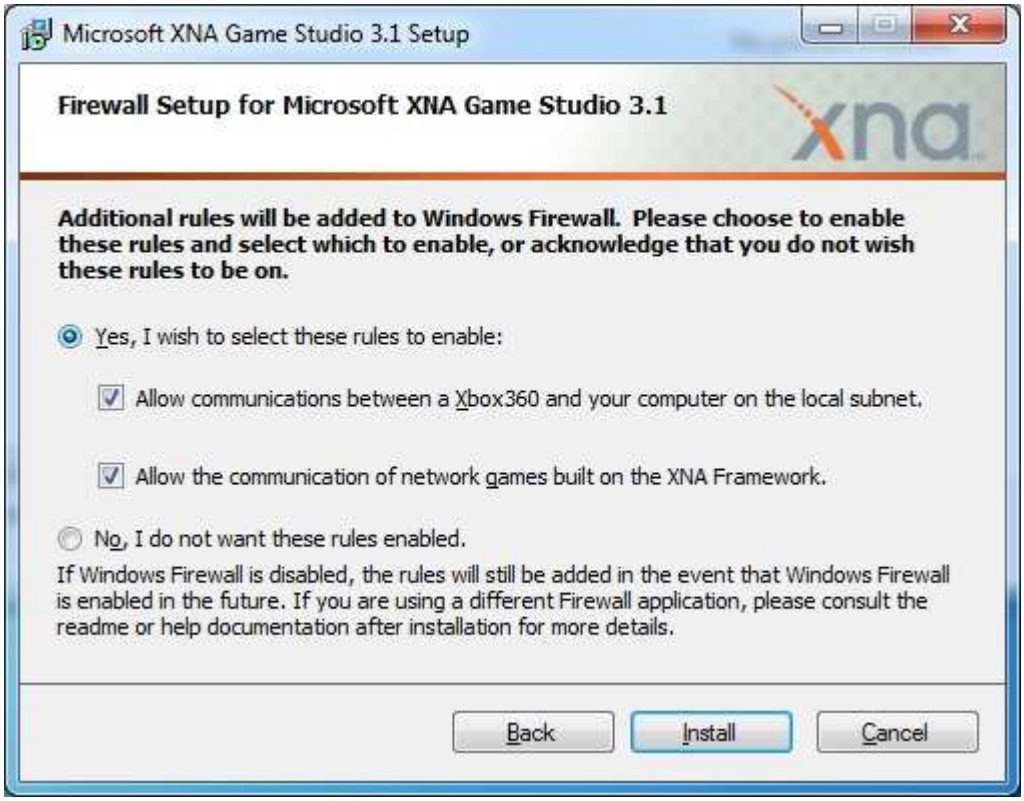
3. Λήψη και εκτέλεση του Microsoft Xna Game Studio 3.1 (θα δούμε αναλυτικά παρακάτω), ακολουθώντας αναλυτικά τις οδηγίες που μας δίνονται κατά τη διάρκεια της εγκατάστασης.

4. Εκίννηση του Visual Studio 2008 από το φάκελο του Microsoft Xna Game Studio από το μενού Έναρξη.

Αφού έχετε κάνει την εγκατάσταση μιας από τις εκδόσεις του Visual Studio που αναφέραμε παραπάνω, εγκαταστήστε το Xna Game Studio 3.1. Η εγκατάσταση για το Xna Game Studio είναι πολύ ξεκάθαρη, αλλά ας τη δούμε βήμα-βήμα.
Στο παράθυρο καλωσορίσματος πατήστε, Next (επόμενο).
Αφού διαβάσετε τους όρους άδειας λογισμικού (EULA- End User License Agreement), πατήστε, Accept (αποδοχή) και επιλέξτε, Next, όπως φαίνεται και στην εικόνα 6.
Η επόμενη οθόνη που εμφανίζεται κατά την εγκατάσταση, ρωτά αν επιθυμείτε να επιτραπεί η επικοινωνία του Game Studio και των Xna παιχνιδιών μέσω του firewall (τοίχος προστασίας) στον υπολογιστή σας. Είναι προτεινόμενο να επιτρέψετε αυτήν την επικοινωνία. Η πρώτη επιλογή, απαιτείται αν θέλετε να έχετε την ικανότητα να αναπτύξετε τα Xna παιχνίδια σας σε μια κονσόλα Xbox 360. η δεύτερη επιλογή, απαιτείται για δικτυακό παιχνίδι στα Xna παιχνίδια. Ενεργοποιήστε και τα δύο Firewall επικοινωνίας και επιλέξτε, Next.

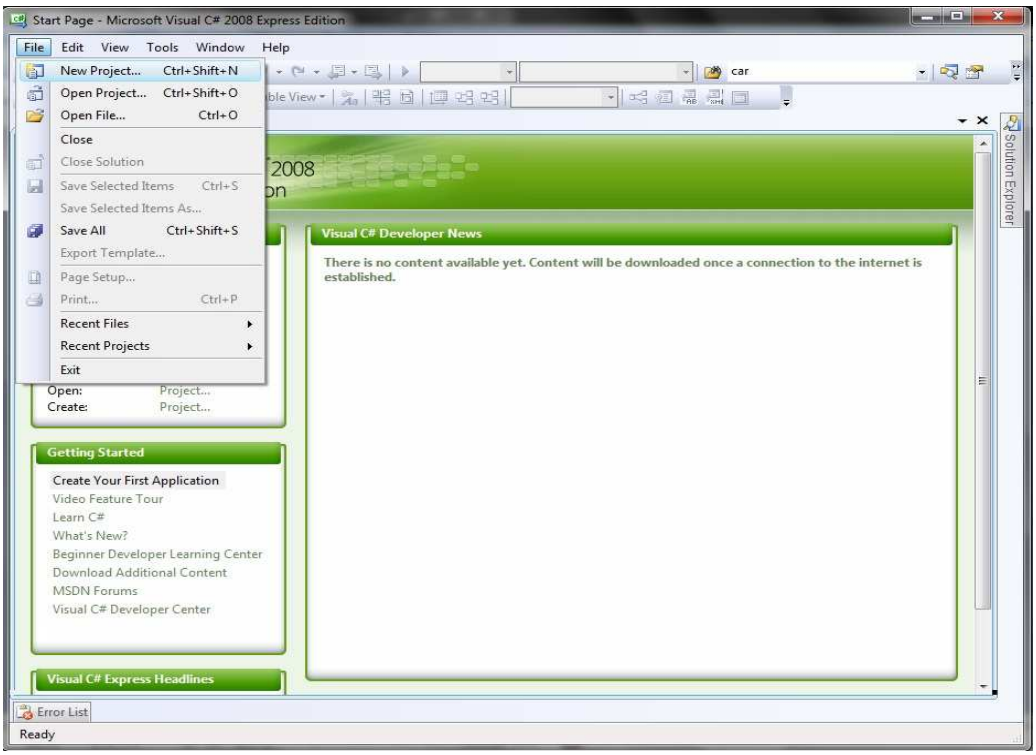
Σημείωση: αν χρησιμοποιείτε κάποιο άλλο firewall λογισμικό στον υπολογιστή σας, ίσως χρειαστεί να προσθέσετε κανόνες, για την επικοινωνία με το Xbox 360 και το Xna Framework από κάποιο άλλο εγχειρίδιο χρήσης (manual).

Εικ. 6 Οθόνη καλωσορίσματος για την Εγκατάσταση του Xna.



Εικ. 7 Ρυθμίσεις Firewall

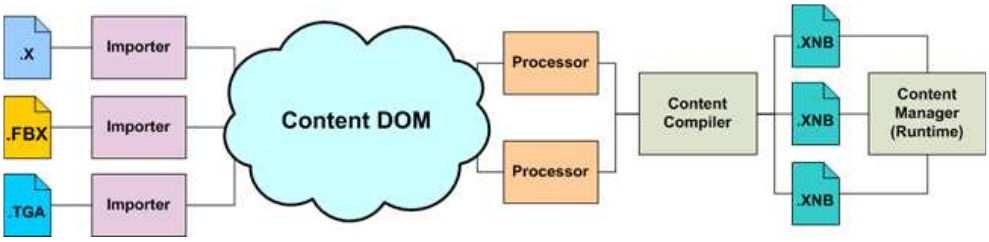
Αφού δεχτείτε τις ιδιότητες του firewall, η εγκατάσταση του Xna Game Studio, θα αντιγράψει τα αρχεία στον υπολογιστή σας και θα ολοκληρώσει την εγκατάσταση.
Μόλις τελειώσει η εγκατάσταση, πατήστε Finish (τέλος) και είστε έτοιμοι να αναπτύξετε τις εφαρμογές σας. Το Xna Game Studio έχει ενοποιηθεί στο Visual Studio, επομένως για να ξεκινήσετε απλά εκκινήστε το Visual Studio 2008, για να δημιουργήσετε τα δικά σας παιχνίδια (Εικ.).



Εικ. 8 Δημιουργία Νέου Project

2.5 Xna Framework Content Pipeline

Σε ένα παιχνίδι που δεν σχετίζεται με το Xna, υπάρχει η ανησυχία του πώς θα εισαχθούν εξωτερικά δεδομένα, όπως ήχοι, γραφικά και τρισδιάστατα μοντέλα.
Το Content Pipeline είναι ένα εργαλείο μέσω του οποίου γίνεται η εισαγωγή του δημιουργικού υλικού κατά το χρόνο εκτέλεσης της εφαρμογής. Έχει τη δυνατότητα εισαγωγής σε: Meshes, εικόνες, μουσική, μοντέλα, ήχο, γραμματοσειρές, εφέ.
Απλουστεύει την επεξεργασία του συνολικού περιεχομένου ώστε οι προγραμματιστές να μπορούν να το χειριστούν πιο εύκολα. Περιέχει μια σειρά από ενέργειες, οι οποίες περιλαμβάνουν εισαγωγείς που διαβάζουν τα δεδομένα και παράγουν μια γνωστή μορφή, έναν επεξεργαστή που παίρνει αυτή τη μορφή, έναν compiler που δημιουργεί το έτοιμο προς χρήση περιεχόμενο και τέλος, τον διαχειριστή του περιεχομένου αυτού. (Σχ.2)



Σχ.1 Το Xna Content Pipeline

Κάτι πολύ ενδιαφέρον σχετικά με το Content Pipeline, είναι ότι βασίζεται στο περιεχόμενο που οι προγραμματιστές συμπεριλαμβάνουν αποτελεσματικά στο έργο τους. Αυτό σημαίνει ότι όταν το έργο δομείται, το περιεχόμενο μετατρέπεται σε μια αναγνωρίσιμη μορφή και μεταφέρεται σε κάποιον γνωστό φάκελο, έτσι το πρόγραμμα γνωρίζει πάντα που μπορεί να βρει το περιεχόμενο αυτό και πώς να το διαβάσει.
Κατά την προσθήκη του περιεχομένου στο Xna πρόγραμμα, χρησιμοποιείται ένας από τους εισαγωγείς περιεχομένου ως μέρος του framework. Αυτοί οι εισαγωγείς κανονικοποιούν τα δεδομένα του περιεχομένου και τα θέτουν σε μια μορφή η οποία μπορεί εύκολα να επεξεργαστεί αργότερα. Οι μορφές αρχείου που υποστηρίζονται είναι οι ακόλουθες:

- 3D μορφές αρχείου: .X (που χρησιμοποιείται από το DirectX), .FBX (δημιουργήθηκε από το Autodesk και υποστηρίζεται από τα περισσότερα εμπορικά και άλλα freeware εργαλεία).
- Material μορφές αρχείου: .FX (αρχεία εφέ, χρησιμοποιούνται για να περιγράψουν λεπτομέρειες που αφορούν την επικάλυψη των 3D μοντέλων ή για την προσθήκη εφέ στην τρισδιάστατη σκηνή).
- 2D μορφές αρχείου: BMP, DDS, JPG, PNG και TGA.
- Αρχεία περιγραφής γραμματοσειρών: SPRITEFONT (xml αρχεία που χρησιμοποιούνται από το Xna).
- Xml αρχεία: XML (μπορούν να χρησιμοποιηθούν για την αποθήκευση των ιδιοτήτων του παιχνιδιού).
- Audio αρχεία: XAP, WAV, WMA και MP3.

2.6 Xna Framework Class Library

Είναι μια βιβλιοθήκη από κλάσεις (classes), διεπαφές (interfaces) και τύπους τιμών (value types) που συμπεριλαμβάνονται στο Xna Game Studio. Αυτή η βιβλιοθήκη παρέχει πρόσβαση στη λειτουργικότητα του Xna Framework και είναι σχεδιασμένη ώστε να αποτελεί το θεμέλιο όπου οι εφαρμογές του Xna Game Studio, τα περιεχόμενα και οι έλεγχοι δομούνται.
Το namespace, είναι το περιβάλλον που δημιουργήθηκε για να προσφέρει μια λογική ομαδοποίηση των αναγνωριστικών ή συμβόλων (όπως τα ονόματα).
Τα namespaces που υπάρχουν εξ' ορισμού κατά την δημιουργία ενός project φαίνονται παρακάτω:

Microsoft.Xna.Framework

Παρέχει τις κλάσεις που χρειάζονται πιο συχνά, όπως τα χρονόμετρα και τον βρόχο του παιχνιδιού (game loop).

Microsoft.Xna.Framework.Audio

Χρησιμοποιεί μεθόδους για να φορτώνει και να χειρίζεται δεδομένα που χρειάζονται στην αναπαραγωγή ήχων.

Microsoft.Xna.Framework.Content

Περιέχει τα στοιχεία του χρόνου εκτέλεσης του Content Pipeline.

Microsoft.Xna.Framework.Design

Παρέχει έναν ενοποιημένο τρόπο μετατροπής τύπων τιμών σε άλλους τύπους.

Microsoft.Xna.Framework.GamerServices

Περιέχει κλάσεις που υλοποιούν τις διάφορες υπηρεσίες που σχετίζονται με τους παίκτες.
Οι υπηρεσίες αυτές επικοινωνούν απευθείας με τον παίκτη και τα δεδομένα του.

Microsoft.Xna.Framework.Graphics

Περιέχει μεθόδους που εκμεταλλεύονται τις δυνατότητες επιτάχυνσης υλικού για την απεικόνιση των τρισδιάστατων αντικειμένων.

Microsoft.Xna.Framework.Graphics.PackedVector

Αντιπροσωπεύει τύπους δεδομένων, με στοιχεία που δεν είναι πολλαπλάσια των 8 bit.

Microsoft.Xna.Framework.Input

Περιέχει κλάσεις για την απαρίθμηση, την αναπαραγωγή και απεικόνιση τραγουδιών, άλμπουμ, λιστών αναπαραγωγής και φωτογραφιών.

Microsoft.Xna.Framework.Media

Περιέχει κλάσεις που υποστηρίζουν το Xbox Live, τους πολλαπλούς παίκτες και την δικτύωση για τα Xna Framework παιχνίδια.

Microsoft.Xna.Framework.Net

Περιέχει κλάσεις που επιτρέπουν την ανάγνωση και γραφή των αρχείων.

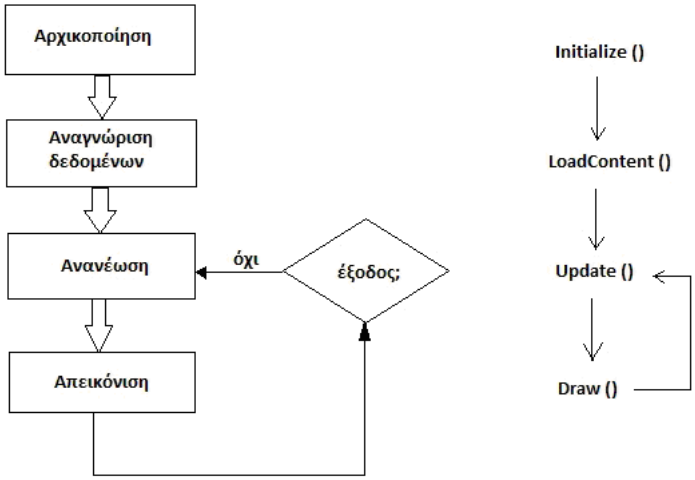
2.7 Τύποι αρχείων των 3D μοντέλων:

Τα τρισδιάστατα μοντέλα που χρησιμοποιούνται για την υλοποίηση του εικονικού κόσμου, δημιουργούνται σε μια primitive μορφή και στη συνέχεια εισάγονται στο Xna μέσω του Content Pipeline, όπως αναφέρθηκε και παραπάνω. Για τα μοντέλα αυτά, τα χρώματα και οι υφές που εφαρμόζονται, μπορούν να δημιουργηθούν εκτός του Xna, μέσω άλλων προγραμμάτων δημιουργίας τρισδιάστατων αντικειμένων, όπως το 3D Studio Max στην περίπτωσή μας, ή άλλων προγραμμάτων σχεδίασης, όπως το Maya, το Blender, το Lightwave και του Modo. Τα μοντέλα που δημιουργούνται από αυτά τα προγράμματα, μπορούν να εξαχθούν σε πολλούς μορφότυπους. Το Xna υποστηρίζει τους .X και .FBX μορφότυπους για τα τρισδιάστατα μοντέλα. Φορτώνοντάς στις Xna εφαρμογές που δημιουργούνται, οι προγραμματιστές μπορούν να εστιάσουν στην επεξεργασία, την θέση, την κίνηση και την περιστροφή τους.

2.8 Δομή μιας εφαρμογής:

Η δομή μια εφαρμογής αποτελείται από κάποια στάδια (Σχ.1):

- Αρχικοποίηση των συσκευών
- Φόρτωση των art assets
- Main game loop (Update, Draw a frame, Repeat)
- Κλείσιμο των συσκευών
- Αποδέσμευση των πόρων



Σχ.2 δομή μιας εφαρμογής Xna

Στα αριστερά απεικονίζεται το game loop στη γενική του μορφή. Η εφαρμογή περνά αρχικά από τις φάσεις της αρχικοποίησης (δέσμευση μνήμης, ορισμός αντικειμένων) και της αναγνώρισης δεδομένων (υφές, μοντέλα, ήχους). Στη συνέχεια εισέρχεται σε ένα loop κατά το οποίο κάνει συνεχώς δύο λειτουργίες, την Ανανέωση (Update) και την Απεικόνιση (Draw ή Render). Στη φάση της Ανανέωσης, η εφαρμογή διαβάζει την είσοδο από τον χρήστη (μετακίνηση joystick ή πλήκτρου) και με βάση αυτή και την τρέχουσα χρονική στιγμή ενημερώνει την κατάσταση των αντικειμένων του κόσμου. Μπορεί δηλαδή να μετακινήσει ένα αντικείμενο, να κάνει ένα χαρακτήρα να πυροβολήσει, ένα βαρέλι να εκραγεί, το γρασίδι να κινηθεί στον άνεμο, κλπ. Επίσης, στη φάση αυτή, ελέγχονται και οι αλληλεπιδράσεις μεταξύ των αντικειμένων του παιχνιδιού, δηλαδή αν κάποιο αντικείμενο συγκρούεται με κάποιο άλλο, όπως επίσης υλοποιείται και η τεχνητή νοημοσύνη (artificial intelligence) ενός χαρακτήρα.
Δημιουργώντας ένα απλό Xna project , η Visual C# τοποθετεί όλον τον κώδικα του project σε μια κλάση (class) με το όνομα Game1. Οι εφαρμογές και τα παιχνίδια που δημιουργούνται είναι συνεπώς ένα αντικείμενο (object) τύπου Game1. Η μορφή του κώδικα είναι ως εξής:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Project
{

    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
```

```
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }


        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }


        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here
        }


        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }


        protected override void Update(GameTime gameTime)
        {
            // Allows the game to exit
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
            ButtonState.Pressed)
                this.Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here

            base.Draw(gameTime);
        }
    }
}
```

2.8.1 Μεταβλητές

Όπως φαίνεται, υπάρχουν κάποιες class-level μεταβλητές οι οποίες παρέχονται αυτόματα από το Xna όπως επίσης ένας constructor και πέντε ακόμα μέθοδοι. Η πρώτη class-level μεταβλητή, είναι τύπου **GraphicsDeviceManager**. Είναι ένα πολύ σημαντικό και χρήσιμο αντικείμενο (object), διότι παρέχει στους προγραμματιστές έναν τρόπο πρόσβασης της μηχανής γραφικών (graphics device) στον υπολογιστή, το Xbox ή το Zune. Το αντικείμενο GraphicsDeviceManager έχει μια ιδιότητα που ονομάζεται GraphictsDevice που παρουσιάζει την πραγματική μηχανή γραφικής αναπαράστασης στην πλατφόρμα που χρησιμοποιείται. Επειδή το graphics device object ενεργεί ως αγωγός μεταξύ της Xna εφαρμογής της κάρτας γραφικών του μηχανήματος (ή ακριβέστερα, της Graphics Processing Unit, ή της GPU στην κάρτα γραφικών), οτιδήποτε δημιουργείται στην οθόνη της Xna εφαρμογής, εκτελείται μέσω αυτού του object.
Η δεύτερη μεταβλητή, είναι ένα instance της **SpriteBatch** κλάσης. Αυτό είναι το βασικό αντικείμενο που χρησιμοποιείται για την αναπαράσταση των Sprites. Ένα Sprite, είναι μια δισδιάστατη ή τρισδιάστατη εικόνα που ενσωματώνεται σε μια μεγαλύτερη σκηνή. Ένα παιχνίδι δύο διαστάσεων δημιουργείται με την αναπαράσταση πολλαπλών sprites στην σκηνή (player sprite, enemy sprite, background sprite, κλπ.).

2.8.2 Μέθοδοι

Η **Initialize** μέθοδος, χρησιμοποιείται για την αρχικοποίηση των μεταβλητών και άλλων αντικειμένων που σχετίζονται με το Game1 object. Το graphics device object, αρχικοποιείται σε αυτό το σημείο και μπορεί να χρησιμοποιηθεί από την Initialize μέθοδο για να βοηθήσει τους δημιουργούς, να αρχικοποιήσουν άλλα αντικείμενα που εξαρτώνται από τις ρυθμίσεις τους, για παράδειγμα σε αυτή τη μέθοδο συνήθως αρχικοποιείται η τιμή του score ενός παιχνιδιού.

Η **LoadContent** μέθοδος, καλείται αμέσως μετά την Initialize μέθοδο και εκτελείται μια φορά κατά την έναρξη του παιχνιδιού. Η μέθοδος αυτή χρησιμοποιείται για να φορτωθεί από το δίσκο όλο το περιεχόμενο της εφαρμογής που θα δημιουργηθεί, δηλαδή εικόνες (υφές), τρισδιάστατα μοντέλα, ήχοι, κλπ. Όπως φαίνεται, εξ' ορισμού η LoadContent δημιουργεί ένα αντικείμενο τύπου SpriteBatch. Αυτό μπορεί να χρησιμοποιηθεί για να φορτωθούν εικόνες στην εφαρμογή.

Η μέθοδος **Update**, καλείται 60 φορές το δευτερόλεπτο (εξ' ορισμού) και κάνει όλες τις απαραίτητες ενημερώσεις στην λογική του παιχνιδιού, στην θέση και την κατάσταση των χαρακτήρων του, επεξεργάζεται την είσοδο από το πληκτρολόγιο ή το controller, κλπ.

Στον πραγματικό κόσμο, για την λήψη μιας φωτογραφίας, μετακινούμε τα έπιπλα, τοποθετούμε τα άτομα στη σκηνή, λέμε στα άτομα που θα υπάρχουν στη φωτογραφία να φτιάξουν τα μαλλιά τους, να χαμογελάσουν και να κάτσουν ακίνητα. Κατ' αναλογία, στην μέθοδο Update εισάγουμε κώδικα που στήνει το σκηνικό και όλα τα αντικείμενα της σκηνής και τα ετοιμάζει για φωτογράφηση. Αυτό είναι κάτι που κάνει η μέθοδος **Draw**. Σε αυτή τη μέθοδο, συμπεριλαμβάνεται όλος ο κώδικας που απεικονίζει τον κόσμο του παιχνιδιού στην οθόνη. Η Draw καλείται επίσης, αμέσως μετά την Update, 60 φορές το δευτερόλεπτο (εξ' ορισμού, όμως αυτό μπορεί να ρυθμιστεί).
Μέχρι στιγμής, η μέθοδος Draw, δεν κάνει τίποτα άλλο, από το να καθαρίζει το παράθυρο και να θέτει το χρώμα του φόντου σε μπλε (Color.CornflowerBlue), το οποίο μπορεί επίσης να ρυθμιστεί από το δημιουργό.

Η αλληλουχία ενημέρωσης της κατάστασης του κόσμου του παιχνιδιού (Update) και της απεικόνισής του (Draw), ονομάζεται game loop (βρόχος παιχνιδιού) και υπάρχει σε όλα τα παιχνίδια, από το πιο απλό μέχρι το πιο σύνθετο. Το game loop ιδανικά εκτελείται 60 φορές το δευτερόλεπτο για ρυθμό απεικόνισης (framerate) 60 Hz. Αν το παιχνίδι έχει πολλά και σύνθετα αντικείμενα με πολύπλοκη φυσική εξομοίωση και τεχνητή νοημοσύνη, ο ρυθμός απεικόνισης μπορεί να είναι αρκετά χαμηλότερος.

Ένα σημείο που χρειάζεται συζήτηση, είναι η παράμετρος **gameTime** που χρησιμοποιείται και από τις δύο μεθόδους Update και Draw. Η παράμετρος αυτή είναι πολύ σημαντική για την λογική ολόκληρου του παιχνιδιού επειδή το παιχνίδι χρειάζεται να ξέρει πόσος χρόνος πέρασε από το τελευταίο βήμα του game loop για να κάνει τους σωστούς υπολογισμούς για παράδειγμα, να υπολογίσει τη σωστή θέση για τα αντικείμενα ανάλογα με την ταχύτητα τους στο παιχνίδι.

Το Xna γενικώς χρησιμοποιεί αντικείμενα για να εκπροσωπήσει και να δώσει πρόσβαση σε λειτουργίες τους, διάφορους πόρους την παιχνιδομηχανής. Το GamePad είναι ένα αντικείμενο που αντιπροσωπεύει το χειριστήριο που τυχόν είναι συνδεδεμένο στην παιχνιδομηχανή (συμπεριλαμβανομένου και του υπολογιστή). Στον κώδικα που δημιουργείται αυτόματα από το Xna, χρησιμοποιείται για να βρεθεί η τρέχουσα κατάσταση του παιχνιδιού και αν ο χρήστης έχει πατήσει κάποιο συγκεκριμένο κουμπί (το Back για την ακρίβεια). Για το πληκτρολόγιο, το αντίστοιχο αντικείμενο είναι το Keyboard. Αντίστοιχα, το GraphicsDevice είναι ένα αντικείμενο που εκπροσωπεί το παράθυρο μέσα στο οποίο απεικονίζεται το παιχνίδι. Η μέθοδος Clear που καλείται στην Draw καθαρίζει το παράθυρο και θέτει ως φόντο ένα συγκεκριμένο χρώμα.

Η κύρια μέθοδος (**main**), έχει την εξής μορφή:

```
namespace Project
{
    static class Program
    {

        // The main entry point for the application.

        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
```

και δημιουργείται αυτόματα σε μια ξεχωριστή κλάση που ονομάζεται Program.cs.

Αναφορά:

Wikipedia (http://en.wikipedia.org),

Rob Miles “Introduction to Programming Through Game Development Using Microsoft Xna Game Studio” (Academic Edition),

Alexandre Santos Lobao, Bruno Evangelista, Jose Antonio Leal de Farias and Riemer Grootjans “Xna 3.0 Game Programming, From Novice to Professional” (Apress),

Aaron Reed “O’ Reilly Learning Xna 3.0”.

3

Γραφικά

3.1 Γραφικά υπολογιστών (computer graphics)

Τα γραφικά υπολογιστών *(computer graphics)* είναι ένας κλάδος της επιστήμης των υπολογιστών που ασχολείται με τη θεωρία και την τεχνολογία σύνθεσης εικόνων σε ηλεκτρονικό υπολογιστή.

Λογισμικό
Για την εισαγωγή, την δημιουργία και την επεξεργασία των γραφικών υπολογιστών χρησιμοποιείται ένα ειδικό λογισμικό που ονομάζεται "Λογισμικό επεξεργασίας εικόνας".
Έτσι, δημιουργούνται γραφικά απεικόνισης διαφόρων σχέσεων - όπως χαρτών και ειδώλων δύο ή τριών διαστάσεων - με την χρήση τελειών, γραμμών, καμπυλών κ.τ.λ. Τα στοιχεία μπορούν να εισαχθούν στον υπολογιστή μέσω διαφόρων συσκευών, (όπως ένας σαρωτής (scanner) ή μια Ψηφιακή φωτογραφική μηχανή, με την μορφή γραμμών, κουκκίδων είτε μέσω του πληκτρολογίου. Όταν παρουσιασθεί το αποτέλεσμα στην οθόνη, ο χρήστης μπορεί να το χειριστεί μετακινώντας το οριζοντίως και καθέτως ή περιστρέφοντάς το, να το επεξεργαστεί ή να το προεκτείνει με την χρήση μιας φωτογραφίδας (light pen) ή μολυβιού με σφαιρίδιο. Ορισμένοι τύποι λογισμικού, όπως τα υπολογιστικά φύλλα, διαθέτουν την δυνατότητα δημιουργίας και επεξεργασίας (εντός ορισμένων πλαισίων) γραφικών παραστάσεων. Η επεξεργασία ενός γραφικού παραμένει, ωστόσο, αποκλειστικό αντικείμενο του λογισμικού επεξεργασίας εικόνας. Κυριότεροι (αλλά όχι μοναδικοί) τύποι τέτοιου λογισμικού είναι το Photoshop της Adobe Corp., το CorelDraw της CorelCorp. και το 3DStudio Max της Autodesk. Υπάρχουν, φυσικά, πολλά ακόμη προγράμματα (ορισμένα, μάλιστα, διανέμονται τελείως δωρεάν), με ποικιλία δυνατοτήτων και χαρακτηριστικών όπως και απαιτήσεων από το Λειτουργικό Σύστημα και το υπολογιστικό σύστημα στο οποίο θα χρησιμοποιηθούν.

Είδη γραφικών υπολογιστών
Τα γραφικά υπολογιστών μπορούν να διακριθούν σε διάφορες κατηγορίες ανάλογα με κάποιο κριτήριο κατηγοριοποίησης. Ανάλογα με το πλήθος των διαστάσεων οι οποίες συμμετέχουν στην απεικόνιση:

- Δισδιάστατα (2d) Γραφικά Υπολογιστών
- Τρισδιάστατα (3d) Γραφικά Υπολογιστών

Ανάλογα με τον χρόνο στον οποίο γίνεται η απόδοσή τους (rendering):

- Στατικά γραφικά Υπολογιστών
- Γραφικά Υπολογιστών Πραγματικού Χρόνου

Δισδιάστατα (2d) Γραφικά Υπολογιστών

Τα δισδιάστατα γραφικά υπολογιστών αποτελούν προσπάθειες απεικόνισης γραφικών δύο διαστάσεων στην οθόνη μιας ψηφιακής συσκευής (π.χ. ενός υπολογιστή). Συνήθως τέτοιου είδους γραφικά χρησιμοποιούνται για την δημιουργία γραφικών διεπαφών χρήστη (GUI-Graphical User Interface), αλλά και για εικονογραφήσεις βιβλίων, περιοδικών κλπ. εντύπων.

Στα γραφικά με υπολογιστή συμπεριλαμβάνεται και η επεξεργασία εικόνας (στατικής ή κινούμενης), η οποία γίνεται με τη βοήθεια ειδικού λογισμικού. Η επεξεργασία εικόνας είναι μια από τις δημοφιλέστερες χρήσεις του υπολογιστή σήμερα, καθώς επιτρέπει τη βελτίωση μιας φωτογραφίας με την εφαρμογή ειδικών φίλτρων, τα οποία μπορούν όχι μόνο να τη βελτιώσουν, αλλά και να την παραλλάξουν (φωτομοντάζ).

Τα δύο διαστάσεων γραφικά μπορούν να καταταγούν στις εξής δύο μεγάλες κατηγορίες:

- **Διανυσματικά γραφικά** (Vector Graphics): Χρησιμοποιούνται για τη δημιουργία εικόνων όπως λογότυποι, σήματα κατατεθέντα κτλ. αλλά και ψευδο-τρισδιάστατων σχημάτων (προοπτική).

- **Γραφικά ψηφίδων** (bitmap graphics): Όλα τα γραφικά που δημιουργούνται από ψηφιοποίηση υπαρκτών αντικειμένων (φωτογραφίες, εικόνες από σαρωτές κτλ.) ανήκουν σε αυτή την κατηγορία.

Η βασική διαφορά των δύο κατηγοριών είναι το χαρακτηριστικό της **ανάλυσης** (resolution). Τα διανυσματικά γραφικά "περιγράφουν" μια εικόνα με τη βοήθεια της αναλυτικής γεωμετρίας και, κατά συνέπεια, με τη βοήθεια εξισώσεων, ενώ τα γραφικά ψηφίδων λειτουργούν όπως ακριβώς ένα ψηφιδωτό: Όσο μικρότερες και περισσότερες ψηφίδες χρησιμοποιούνται, τόσο πιο ευκρινές και ακριβές είναι το τελικό αποτέλεσμα. Η ανάλυση μετράται σε κουκκίδες (ψηφίδες) ανά ίντσα (dots per inch, dpi). Για μια οθόνη, η ανάλυση των 72 ή 96 dpi είναι επαρκέστατη, αν όμως η εικόνα προορίζεται για επαγγελματική εκτύπωση, το ελάχιστο απαιτούμενο είναι οι 300 dpi. Τα διανυσματικά γραφικά είναι ανεξάρτητα ανάλυσης (resolution free), γιατί απλά δε χρησιμοποιούν ψηφίδες για το σχηματισμό της εικόνας.

Ένα ακόμη χαρακτηριστικό των γραφικών είναι το βάθος χρώματος, δηλαδή το πλήθος των δυαδικών ψηφίων (bits) που χρησιμοποιούνται για την περιγραφή του χρώματος κάθε ψηφίδας (ή κάθε περιοχής στα διανυσματικά γραφικά). Το σημερινό στάνταρ είναι για μεν τις οθόνες βάθος χρώματος 24 bits ενώ για τις εκτυπώσεις 32 bits (Η οθόνη και η εκτύπωση χρησιμοποιούν διαφορετικά χρωματικά πρότυπα. Υπάρχουν και γραφικά με μεγαλύτερο βάθος χρώματος, που προορίζονται για ειδικές χρήσεις, καθώς το ανθρώπινο μάτι δε μπορεί να διακρίνει περισσότερα από 16,7 εκατομμύρια χρωματικές διαβαθμίσεις).

Για τις εφαρμογές του Διαδικτύου χρησιμοποιούνται αποκλειστικά τα γραφικά ψηφίδων, γιατί (προς το παρόν!) δεν είναι δυνατή η μετάδοση μέσω Διαδικτύου διανυσματικών γραφικών.

Ο τύπος των γραφικών αναγνωρίζεται συνήθως από την προέκταση του ονόματος του αρχείου, στο οποίο είναι αποθηκευμένα (δηλ. το τμήμα εκείνο του ονόματος που βρίσκεται δεξιά από την τελεία που χωρίζει στα δύο το όνομα ενός αρχείου). Οι πλέον συνήθεις τύποι είναι:

- Διανυσματικά γραφικά: .cdr, .ai

- Γραφικά ψηφίδων: .tif, .bmp, jpg, .gif, .png (Οι τρεις τελευταίοι τύποι είναι οι κατάλληλοι για το Διαδίκτυο).

Τρισδιάστατα (3d) Γραφικά Υπολογιστών

Τα τρισδιάστατα γραφικά υπολογιστών αποτελούν προσπάθειες απεικόνισης γραφικών τριών διαστάσεων στην - απεικόνισης δύο διαστάσεων - οθόνη μιας ψηφιακής συσκευής (π.χ. ενός υπολογιστή). Το γεγονός ότι η απεικόνιση χρησιμοποιεί τρεις διαστάσεις τα καθιστά ιδιαίτερα ρεαλιστικά. Τέτοιου είδους γραφικά χρησιμοποιούνται συνήθως από προγράμματα όπως παιχνίδια υπολογιστών, εικονικούς κόσμους. Τα τρισδιάστατα γραφικά βρίσκουν επίσης εφαρμογή στον κινηματογράφο, για τη δημιουργία σκηνών εικονικών κόσμων (χαρακτηριστικό παράδειγμα οι ταινίες του κύκλου "Ο Άρχοντας των Δαχτυλιδιών") αλλά και ειδικών εφέ, που είναι αδύνατον να γυριστούν ως πραγματικές σκηνές.

Στατικά Γραφικά Υπολογιστών

Τα στατικά γραφικά υπολογιστών αποτελούν αντικείμενα γραφικών τα οποία δεν αποδίδονται την στιγμή που εκτελούνται αλλά έχουν αποδοθεί μία φορά κατά τη δημιουργία τους. Παράδειγμα τέτοιων γραφικών είναι τα μικρά βίντεο, τα οποία εμφανίζονται σε διάφορα παιχνίδια, και τα οποία έχουν "γυριστεί" μια φορά και κάθε φορά που θα τα παρακολουθήσουμε παραμένουν ίδια. Για τη δημιουργία τους χρησιμοποιείται κάποιο πρόγραμμα δημιουργίας γραφικών και κίνησης (animation) όπως το 3D Studio Max, το Maya, το Lightwave, το Blender,το cinema4d κτλ.

Γραφικά Υπολογιστών Πραγματικού Χρόνου

Τα γραφικά υπολογιστών πραγματικού χρόνου αποτελούν αντικείμενα γραφικών τα οποία αποδίδονται την στιγμή που εκτελούνται. Για παράδειγμα τα γραφικά που εμφανίζονται στην οθόνη ενός υπολογιστή, ο οποίος εκτελεί ένα παιχνίδι, ανήκουν συνήθως σε αυτήν την κατηγορία. Για τη δημιουργία τους απαιτείται κάποια μηχανή απόδοσης γραφικών (graphics rendering engine) πραγματικού χρόνου, όπως για παράδειγμα το Ogre3d, το Irrlich, το Crystal Space κτλ.

Εφαρμογές Γραφικών Υπολογιστών

Όπως έχει αναφερθεί και παραπάνω, υπάρχουν αρκετές εφαρμογές των γραφικών υπολογιστών. Μερικές από αυτές είναι οι γραφικές διεπαφές χρήστη, τα παιχνίδια υπολογιστών, οι εικονικοί κόσμοι. Τα γραφικά υπολογιστών βρίσκουν, επίσης, εφαρμογή στην εικονογράφηση εντύπων, στην αρχιτεκτονική σχεδίαση, στη δημιουργία λογοτύπων αλλά και στη δημιουργία ακριβέστατων σχεδίων γενικότερα.

Στην εργασία αυτή, έγινε η χρήση του εργαλείου επεξεργασίας και δημιουργίας τρισδιάστατων γραφικών, το 3D Studio Max, όπως και το εργαλείο επεξεργασίας εικόνας, Adobe Photoshop.

3.2 Εισαγωγή στο 3D Studio Max

Τα πάντα στη φύση μπορούν να αναπαρασταθούν με συνδυασμούς γεωμετρικών σχημάτων. Το 3D Studio Max παρέχει την δυνατότητα δημιουργίας βασικών γεωμετρικών σχημάτων, κύβου, σφαίρας, κυλίνδρου, κώνου, και όχι μόνο, τα οποία ονομάζονται αντικείμενα και μπορεί τόσο η θέση τους όσο και οι διαστάσεις τους να ρυθμιστούν με μεγάλη ακρίβεια.
Αφού κατασκευαστούν τα αντικείμενα γίνεται δυνατή η επεξεργασία τους με διάφορες τεχνικές ώστε να έρθουν στην επιθυμητή τους μορφή. Ο συνδυασμός των επεξεργασμένων πια αντικειμένων μπορεί να δώσει ιδιαίτερα πολύπλοκες και ρεαλιστικές σκηνές.
Το 3D Studio Max επίσης παρέχει τη δυνατότητα τοποθέτησης διάφορων υλικών (materials) πάνω στα αντικείμενα. Τα materials είναι ένα είδος ταπετσαρίας, ή αλλιώς υφές, και η συγκεκριμένη δυνατότητα

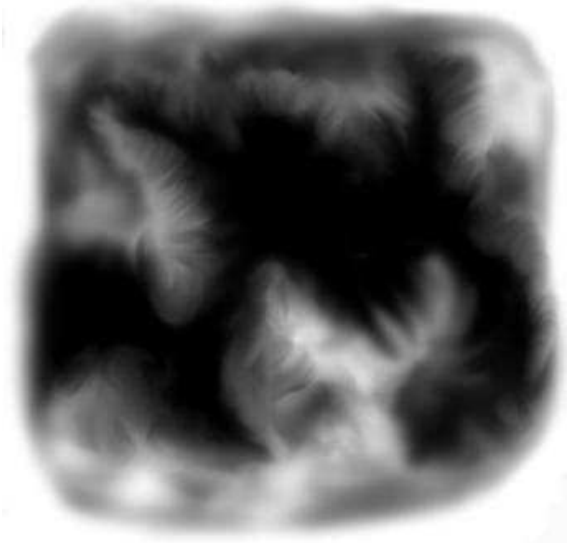
προσθέτει στα αντικείμενα μια δόση ρεαλισμού. Η διαδικασία με την οποία αποδίδεται υφή με κάποιο κανόνα σε κάθε σημείο της επιφάνειας, αποκαλείται απεικόνιση υφής (texture mapping). Πιο απλά θα μπορούσαμε να θεωρήσουμε την απεικόνιση υφής σαν μια επικάλυψη του αντικειμένου με μια εικόνα.
Το 3D Studio Max, παρέχει πολλές ακόμα δυνατότητες που βοηθούν στον ρεαλισμό της σκηνής, όπως ένα ευρύ φάσμα φωτοσκιάσεων που μπορεί να χρησιμοποιηθεί.
Ακολουθεί η τελική διαδικασία του render, η οποία ενοποιεί όλες τις τεχνικές που χρησιμοποιήθηκαν προηγουμένως κατά τη διάρκεια της μοντελοποίησης (modeling), ανάθεσης υφών στα αντικείμενα (texturing) και της χρήσης φωτοσκιάσεων, σε ένα σύνολο από καρέ εικόνων. Στην εργασία αυτή, το 3D Studio Max στάθηκε χρήσιμο για την δημιουργία απλών γεωμετρικών σχημάτων και ενός heightmap που λειτούργησε ως έδαφος στην σκηνή του τρισδιάστατου κόσμου.

3.3 Heightmap

Τα εδάφη (terrains), είναι ένας άριστος τρόπος για να εκπροσωπήσουμε υπαίθρια περιβάλλοντα στην τρισδιάστατη σκηνή μας. Μπορούν να δημιουργούνται τυχαία ή να αποθηκεύονται και να φορτώνονται από μια δισδιάστατη εικόνα γκρι κλίμακας, που ονομάζεται Heightmap.
Σε αυτή την ενότητα θα μιλήσουμε για το τι είναι ένα heightmap, πως λειτουργεί και πως μπορούμε να το δημιουργήσουμε.

3.3.1 Τι είναι τα heightmaps;

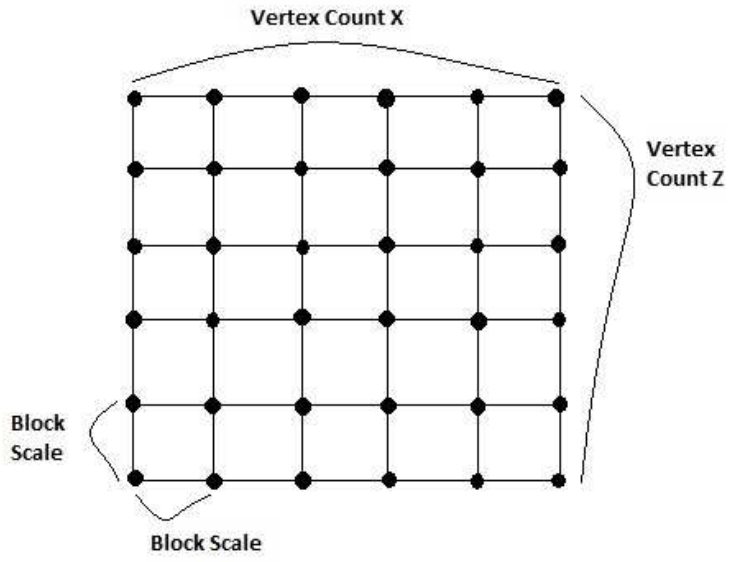
Τα heightmaps είναι αρχεία εικόνας δύο διαστάσεων, που χρησιμοποιούνται για την αποθήκευση του ύψους ενός εδάφους (terrain). Συγκερκιμένα, προσομοιώνει τις λεπτομέρειες που αφορούν το ύψος και το βάθος μιας επιφάνειας, για την απεικόνισή της στον τρισδιάστατο κόσμο. Το heightmap καθορίζει το σχήμα του εδάφους και προσθέτει εξογκώματα (bumps) στην επιφάνεια αυτή. Συνήθως, στα περισσότερα προγράμματα, όπως και στο 3D Studio Max, ονομάζεται «bump map». Το heightmap είναι συνήθως τετράγωνες εικόνες των οποίων το χρώμα (ή μαύρο και άσπρο), αντιστοιχεί σε τιμές όπου το 3D Studio Max μετατρέπει σε υψομετρικά δεδομένα. Συνήθως χρησιμοποιούνται εικόνες της κλιμακας του γκρι με φάσμα που κυμαίνεται από 0-255, όπου 0 είναι το μαύρο χρώμα που αντιπροσωπεύει το χαμηλότερο σημείο στο υψόμετρο του εδάφους και 255 είναι το λευκό χρώμα που αντιπροσωπεύει το υψηλότερο (Εικ.9).



Εικ. 9 heightmap

3.3.2 Πώς λειτουργεί και πως δημιουργείται ένα heightmap;

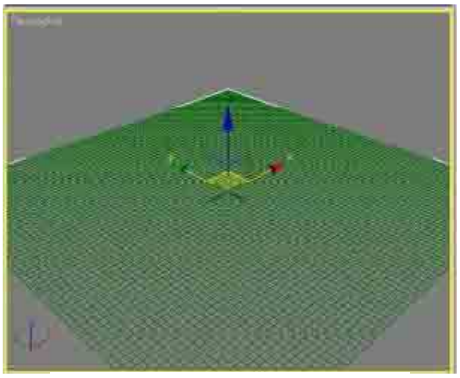
Για να κατασκευάσετε ένα terrain από heightmap, πρέπει πρώτα να φτιάξετε ένα δίχτυ από κορυφές με τις ίδιες διαστάσεις με το heightmap και μετά να χρησιμοποιήσετε την τιμή του ύψους για κάθε σημείο (pixel) πάνω στο heightmap, όπως το ύψος (συντεταγμένη Y) ενός κόμβου πάνω στο δίχτυ.
Εκτός από τη θέση του, κάθε κορυφή στο δίχτυ πρέπει να περιέχει και άλλα χαρακτηριστικά που απαιτούνται για τα εφέ σας, όπως οι συντεταγμένες του texture σας. Στην εικόνα 10 φαίνεται ένα πλέγμα 6x6 από κορυφές πάνω από το x, z plane του κόσμου.



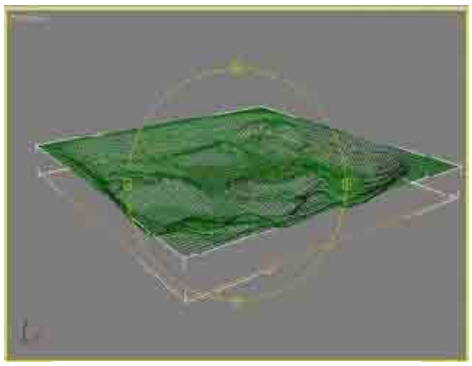
Εικ. 10 6x6 δίχτυ πάνω από το X,Z plane.

Σε ένα δίχτυ από κορυφές, η απόσταση μεταξύ όλων των κάθετων και οριζόντιων γειτονικών κορυφών θα πρέπει να είναι η ίδια. Η απόσταση αυτή αντιπροσωπεύεται από την κλίμακα των μπλοκ, όπως φαίνεται στην εικόνα 7. Μια μικρή απόσταση μεταξύ των κορυφών επιτρέπει την ομαλή μετάβαση ανάμεσα στα ύψη των κορυφών πάνω από το πλέγμα, αλλά θα χρειαστείτε πολλές κορυφές για μια μεγάλη έκταση εδάφους. Μια μεγάλη απόσταση μεταξύ των κορυφών επιτρέπει μεγαλύτερες εκτάσεις, αλλά μπορεί να αποφέρει απότομη μετάβαση ανάμεσα στα ύψη των κορυφών. Για ένα heightmap που περιέχει 256 × 256 pixels, αν η απόσταση μεταξύ κάθε ζεύγους κορυφών (κάθετα και οριζόντια) είναι 1 μέτρο, το συνολικό μέγεθος του παραγόμενου εδάφους θα είναι 255 x 255 μέτρα.
Το heightmap του εδάφους είναι συνήθως αποθηκευμένο σε εικόνες των 8-bit, οι τιμές του ύψους του όμως ποικίλλουν και κυμαίνονται μεταξύ 0 και 255, όπου το 0 (μαύρο χρώμα), όπως είπαμε και παραπάνω, αντιπροσωπεύει το χαμηλότερο δυνατό ύψος για μια κορυφή και το 255 (λευκό χρώμα), αντιπροσωπεύει το υψηλότερο δυνατό ύψος. Μπορείτε να μειώσετε ή να αυξήσετε το διάστημα αυτό χρησιμοποιώντας ένα συντελεστή αλλαγής κλίμακας (scale factor), το οποίο μπορείτε να

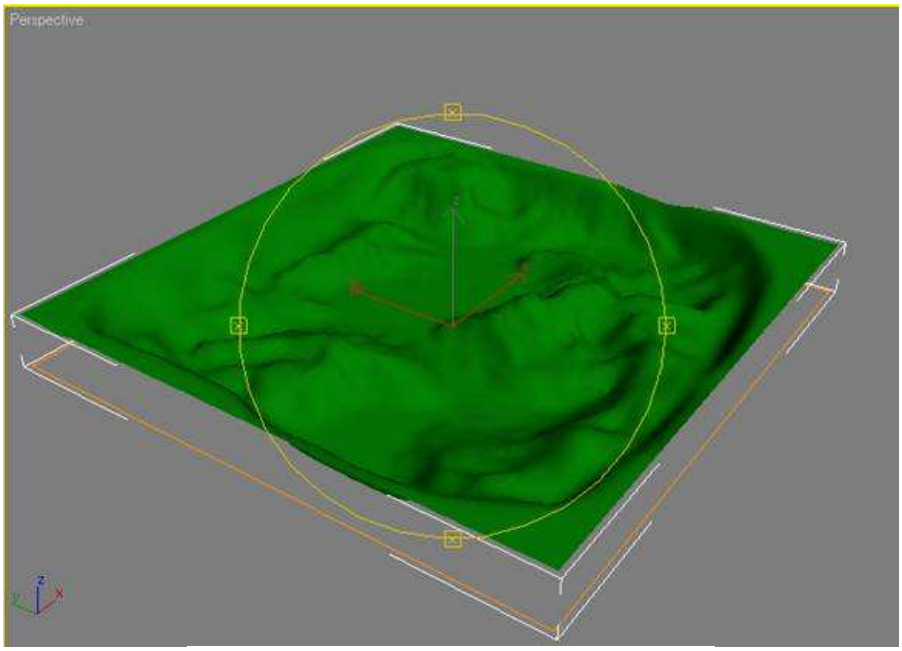
χρησιμοποιήσετε για να πολλαπλασιάσετε τις τιμές του ύψους που είναι αποθηκευμένες στο heightmap, προσαρμόζοντας το φάσμα της. Οι εικόνες 11, 12, 13 και 14 δείχνουν τα στάδια δημιουργίας ενός τρισδιάστατου εδάφους που δημιουργήθηκε με την χρήση του heightmap.



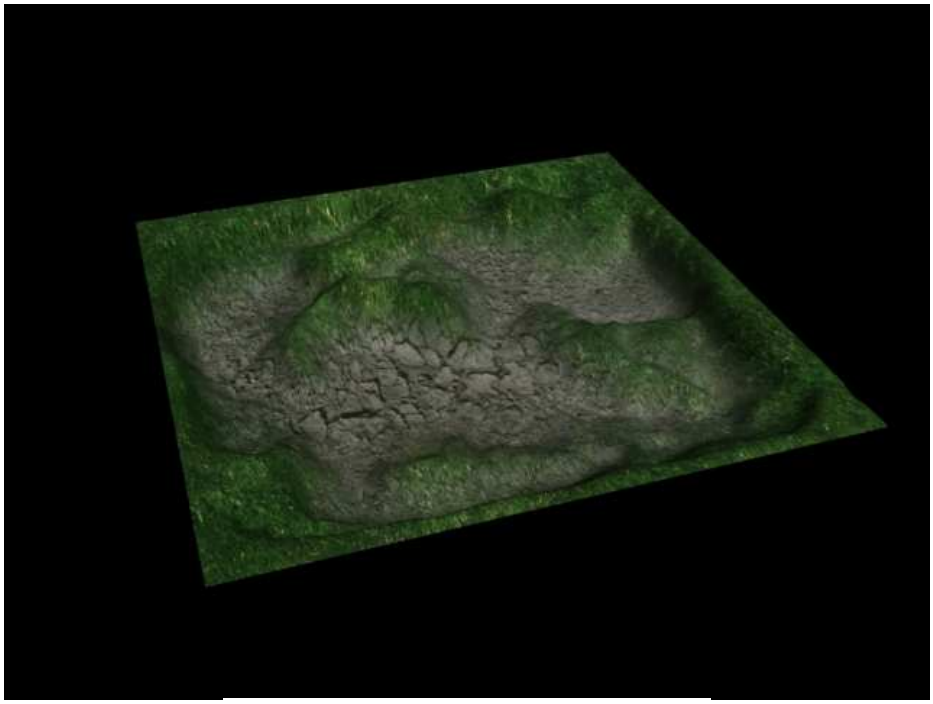
Εικ. 11 Πλέγμα από κορυφές



Εικ. 12 Αλλαγή της κλίμακας (scale)



Εικ. 13 Αρχική μορφή εδάφους χωρίς υφή



Εικ. 14 Τελική μορφή εδάφους με υφή.

Μπορείτε να δημιουργήσετε ή να αποκτήσετε heightmaps με διαφορετικούς τρόπους. Μπορείτε να βρείτε πολλά και διαφορετικά είδη heightmap στον παγκόσμιο ιστό, συμπεριλαμβανομένων των heightmap πραγματικών τοποθεσιών (όπως πόλεις ή ακόμα και τοπία από άλλους πλανήτες). Επειδή τα heightmaps είναι εικόνες τις κλίμακας του γκρι (grayscale images), μπορείτε να χρησιμοποιήσετε οποιοδήποτε εργαλείο επεξεργασίας εικόνας για την κατασκευή ή την επεξεργασία των δικών σας heightmap.

Σημειώστε ότι, όλοι οι τύποι εικόνων που υποστηρίζονται από το Xna Content Pipeline, είναι έγκυροι για την δημιουργία ενός heightmap. Αυτό σημαίνει ότι μπορείτε να χρησιμοποιήσετε σχεδόν οποιαδήποτε φωτογραφία βρίσκετε ως heightmap και να την επεξεργαστείτε χρησιμοποιώντας το πρόγραμμα που επιθυμείτε.

Αναφορά:

Wikipedia (http://en.wikipedia.org),

Youtube video tutorials (www.youtube.com),

Alexandre Santos Lobao, Bruno Evangelista, Jose Antonio Leal de Farias and Riemer Grootjans "Xna 3.0 Game Programming, From Novice to Professional" (Apress),

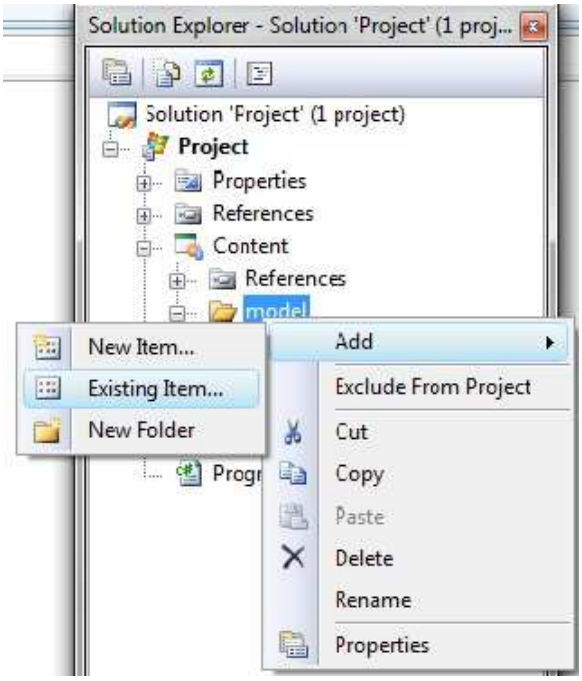
Delta 3D tutorials (www.delta3d.org).

4

Εισαγωγή και αναπαράσταση γραφικών και ήχων

4.1 Προσθήκη υλικού

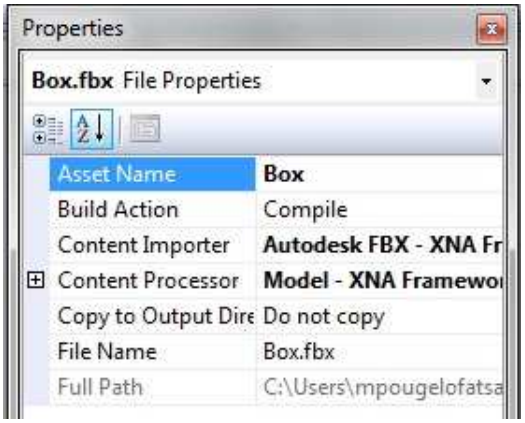
Όπως είπαμε και προηγουμένως, το υλικό που χρησιμοποιείται για την δημιουργία της εφαρμογής, δηλαδή τα μοντέλα, οι ήχοι και λοιπά, φορτώνονται στο Xna από το Content Pipeline.
Αφού έχετε συλλέξει το υλικό που θα χρησιμοποιήσετε, ανοίξτε το Solution Explorer του Visual Studio. Εκεί ακριβώς υπάρχει το σημείο που βρίσκεται το content και θα γίνει η εισαγωγή του περιεχομένου. Για μεγαλύτερη ευκολία, προτείνεται να δημιουργήσετε ξεχωριστούς φακέλους μέσα στο content για να ξεχωρίζετε το είδος του περιεχομένου (ήχοι, εικόνες κ.λ.π), αυτό γίνεται με δεξί κλικ πάνω στο content και Add→new folder. Μέσα στο φάκελο μπορείτε να προσθέσετε το υλικό σας, με δεξι κλικ πάνω του και Add→exising item (Εικ. 15).



Εικ.15 Προσθήκη μοντέλου στο Solution

Αν σε αυτό το σημείο κάνετε build στο solution (Build→Build Solution), το Content Pipeline θα προσπαθήσει να κάνει compile το μοντελο που μόλις προσθέσατε, σ' αυτήν την περίπτωση έναν κύβο. Αν δεν υπάρχουν λάθη, σημαίνει πως το Content Pipeline αναγνώρισε τον τύπο του μοντέλου και ήταν ικανό να το μετατρέψει σε έναν εσωτερικό τύπο του Xna και τώρα αυτό είναι έτοιμο να φορτώσει τον κύβο.
Το Content Pipeline χρησιμοποιεί έναν πόρο για να έχει πρόσβαση στις πηγές του περιεχομένου. Ένας άλλος τρόπος για να εξακριβώσετε αν αναγνωρίστηκε το μοντέλο από το Content Pipeline είναι να δείτε τις

ιδιότητες (properties) των πιο πρόσφατα προστιθέμενων αντικειμένων. Με δεξί κλικ στον Solution Explorer και επιλέγοντας Properties (Εικ.16).



Εικ.16 Ιδιώτητες ενός αρχείου φωτογραφίας

Όπως βλέπετε στην πρώτη παράμετρο (Asset Name), βρίσκεται το όνομα του μοντέλου που προστέθηκε, χωρίς τον τύπο του αρχείου. Εξ' ορισμού, όλα τα ονόματα των υλικών που προστίθεντα στο project δεν περιλαμβάνουν τον τύπο του αρχείου. Αν βλέπετε το Asset Name λοιπόν στις ιδιότητες του μοντέλου, θα ξέρετε ότι το Content Pipeline αναγνώρισε τον κύβο σας. Τα ονόματα των αντικειμένων, τα οποία βρίσκονται στον ίδο φάκελο, πρέπει να είναι μοναδικά.
Όπως παρατηρείτε στην Εικόνα 4, κάτω από την παράμετρο Asset Name, υπάρχουν άλλες δύο παράμετροι. Το Content Importer και το Content Processor. Το γεγονός ότι υπάρχουν αυτές οι δύο ιδιότητες και είναι ρυθμισμένες να είναι Autodesk FBX- XNA Framework και Model-XNA Framework αντίστοιχα, είναι ένα άλλο σημάδι ότι το Content Pipeline έχει αναγνωρίσει τον κύβο σας. Φέρουν την ένδειξη ότι ο κύβος είναι ετοιμος να επεξεργαστεί από το Content Pipeline ως τρισδιάστατο μοντέλο.

4.2 Προσθήκη ήχων και μουσικής

Ακόμα και το καλύτερο παιχνίδι που θα μπορούσατε να υλοποιήσετε, δεν θα είναι αρκετά διασκεδαστικό αν δεν συμπεριλαμβάνει ήχους.
Σε αυτή την ενότητα, θα δούμε πως μπορούμε να βελτιώσουμε ένα παιχνίδι, συμπεριλαμβάνοντας σε αυτό ήχο και εφέ ήχου, με αποτέλεσμα να εξερευνήσουμε τις βασικές έννοιες του ήχου στο Xna.
Το Xna ασχολείται με τον ήχο χρησιμοποιώντας την ίδια δομή που χρησιμοποιεί για τη διαχείριση γραφικών, το Content Pipeline. Για το Xna, ο ήχος είναι απλά ένα άλλο είδος περιεχομένου του παιχνιδιού.

4.2.1 Το εργαλείο δημιουργίας ήχου, XACT

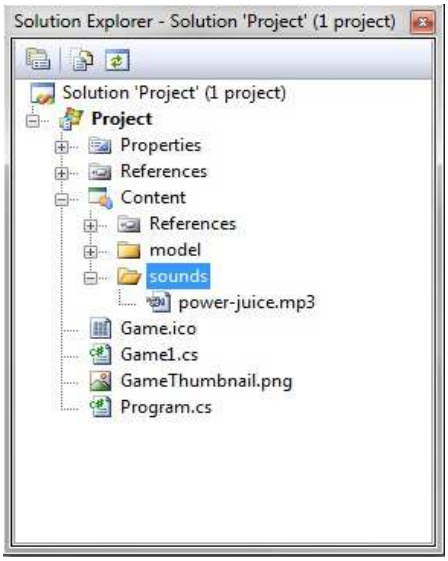
Με το XNA Framework 3.1, υπάρχουν δύο διαφορετικοί τρόποι για την υλοποίηση του ήχου. Σε προηγούμενες εκδόσεις του Xna, οι προγραμματιστές χρησιμοποιούσαν ένα εργαλείο που ονομάζεται Microsoft Cross-platform Audio Creation Tool (XACT) αποκλειστικά για τον ήχο. Χρησιμοποιώντας το XACT, οι προγραμματιστές μπορούσαν να δημιουργήσουν αρχεία ήχου τα οποία επεξεργάζονταν από το Content Pipeline και εφαρμόζονταν χρησιμοποιώντας το API ήχου του XNA Framework. Με το Xna 3.1, το XACT μπορεί ακόμη να χρησιμοποιηθεί για την υλοποίηση ήχου για τον υπολογιστή και το Xbox 360. Εντούτοις, επειδή το Zune δεν υποστηρίζει τη μηχανή XACT, η ομάδα του XNA έχει προσθέσει ένα ξεχωριστό API για τη χρήση του στην ανάπτυξη ήχου στο Zune. Το απλουστευμένο αυτό API υποστηρίζεται επίσης από τα Windows και το Xbox 360.

Γιατί να χρησιμοποιήσετε την πιο περίπλοκη μέθοδο XACT, αν υποστηρίζεται το απλοποιημένο API ήχου από όλες τις πλατφόρμες;

Το XACT σας παρέχεται με ένα μικρό στούντιο ήχου. Μπορείτε να επεξεργαστείτε εύκολα την ένταση, την επανάληψη όπως και άλλες ιδιότητες των ήχων χωρίς να χρειάζεται να επεξεργαστείτε κανέναν κώδικα.
Η μέθοδος XACT των ήχων που χρησιμοποιούνται, προσφέρει ιδιότητες πέρα από τις διαθέσιμες που χρησιμοποιούν το απλοποιημένο API, αλλά ως αντάλλαγμα, είναι μια πιο πολύπλοκη μέθοδος εφαρμογής του ήχου.
Κατά την ανάπτυξη παιχνιδιών για τον υπολογιστή και το Xbox 360, οι προγραμματιστές μπορούν να χρησιμοποιήσουν είτε την μηχανή XACT ή το API που αναπτύχθηκε για τη συμβατότητα με το Zune. Ωστόσο, υπάρχουν ορισμένα μοναδικά πλεονεκτήματα για την εφαρμογή ήχων που χρησιμοποιούν το XACT, όπως είναι η δυνατότητα να δημιουργήσετε κομμάτια μουσικής, να επεξεργαστείτε την ποιότητα του ήχου και να δημιουργήσει επαναλήψεις και άλλα εφέ έξω από τον κώδικά σας.

4.2.2 Χρησιμοποιώντας την απλοποιημένη μορφή για τον ήχο.

Κατά την ανάπτυξη ενός παιχνιδιού για τον υπολογιστή και το Xbox 360, είναι μια καλή ιδέα για να επωφεληθούμε από τα οφέλη που προσφέρει το XACT. Ωστόσο, το XACT δεν υποστηρίζεται από τοZune, έτσι το XNA Framework 3.1 παρέχει ένα απλοποιημένο API για τον ήχο, όπως είπαμε και παραπάνω, που έχει προστεθεί για να επιτρέψει στους προγραμματιστές να αναπαράγουν ήχο για το Zune. Μπορείτε επίσης να χρησιμοποιήσετε το απλοποιημένο αυτό API στο περιβάλλον των Windows και στο Xbox 360, αν διαπιστώσετε ότι δεν χρειάζεστε τα πρόσθετα χαρακτηριστικά που παρέχονται από το XACT.
Για την αναπαραγωγή ενός ήχου χρησιμοποιώντας την απλούστερη μέθοδο, το πρώτο βήμα είναι να προσθέσετε ένα αρχείο ήχου στο project σας. Θυμηθείτε ότι όταν πρόκειται για το XACT, τα αρχεία ήχου δεν προστίθενται στο project στο Visual Studio. Αυτό δεν συμβαίνει ωστόσο, όταν ασχολείστε με την απλοποιημένη μέθοδο. Σε αυτή την περίπτωση, τα αρχεία ήχου αντιμετωπίζονται σαν άλλες πηγές στο content pipeline και πρέπει να προστεθούν στο Visual Studio, με τον ίδιο τρόπο που εισάγονται και τα μοντέλα.
Το API για τον ήχο στο Xna 3.1 υποστηρίζει Wav, Wma και Mp3 τύπους αρχείων. Θα δούμε λοιπόν πώς μπορούμε να εισάγουμε τον ήχο στο παιχνίδι μας. Στο Solution Explorer και στο σημείο που βρίσκεται το Content θα κάνουμε την προσθήκη του ήχου. Προσθέστε έναν νέο φάκελο με όνομα “sounds”, με δεξί κλικ επιλέξτε Add → Existing Item και επιλέξτε το αρχείο που θέλετε από τον φάκελο που είναι αποθηκευμένο (Εικ.17).



Εικ.17 Εισαγωγή του ήχου στο Content Pipeline

Όπως και με τις άλλες πηγές, όταν έχετε προσθέσει το αρχείο στο project, θα πρέπει να είστε σε θέση να προβάλετε τις ιδιότητές του στο Visual Studio (F4) και να δείτε ότι το Content Pipeline το έχει αναγνωρίσει και του έχει δώσει ένα όνομα (asset name) όπως επίσης και τις απαραίτητες ιδιότητες που χρειάζονται.
Μόλις φορτωθεί ο ήχος στο project σας, πρέπει να δημιουργήσετε μια μεταβλητή του τύπου SoundEffect στην οποία θα φορτώσετε το αρχείο ήχου μέσα από το Content Pipeline. Προσθέστε την ακόλουθη μεταβλητή (class level variable) στην κλάση του Game1:

```
SoundEffect Soundeffect;
```

Αφού φτιάξατε μια μεταβλητή ήχου, πρέπει να περάσετε στην μεταωλητη αυτή το SoundEffect μέσα από την μέθοδο LoadContent με τον ακόλουθο τρόπο:

```
Soundeffect = Content.Load<SoundEffect>(@"sounds\power-juice");
```

Για να κάνετε αναπαραγωγή του ήχου, μπορείτε να καλέσετε την μέθοδο Play του αντικειμένου SoundEffect.
Για να αναπαράγετε τον ήχο, όταν ξεκινήσει το παιχνίδι, προσθέστε τον ακόλουθο κώδικα στο τέλος της LoadContent μεθόδου, αμέσως μετά από το σημείο που κάνατε load τον ήχο από το Content Pipeline:

```
Soundeffect.Play();
```

Μπορείτε να εκτελέσετε το πρόγραμμά σας και να δείτε/ακούσετε τα αποτελέσματα.
Το SoundEffect.Play επιστρέφει ένα αντικείμενο του τύπου SoundEffectInstance, το οποίο μπορείτε να χρησιμοποιήσετε για παύση, διακοπή, και αναπαραγωγή του ήχου, καθώς και να ρυθμίσετε την ένταση και άλλες πτυχές του ήχου. Μπορείτε να την χρησιμοποιήσετε ως εξής:

```
SoundEffectInstance soundeffectinstance= Soundeffect.Play();
```

Οι ιδιότητες τις μεθόδου αυτής λοιπόν,θα σας δώσουν περισσότερο έλεγχο πέρα από το πώς ο ήχος αναπαράγεται. Η πιο σύνθετη ιδιότητα της μεθόδου αυτής δέχεται τέσσερα ορίσματα:

- Το πρώτο όρισμα, σας επιτρέπει να ρυθμίσετε την ένταση του ήχου ως μια τιμή μεταξύ 0.0f και 1.0f. Μια τιμή 1.0f αντιστοιχεί στην ένταση που είναι συγχρόνως ρυθμισμένη στο

SoundEffect.MasterVolume, το οποίο είναι από μόνο του μια τιμή μεταξύ 0.0f και 1.0f, όπου το 1.0f αντιστοιχεί στην γενική ένταση του συστήματος.

- Το δεύτερο όρισμα, σας επιτρέπει να ρυθμίσετε την ταχύτητα που αναπαράγεται ο ήχος. Μπορείτε να καθορίσετε μια τιμή μεταξύ -1.0f και 1.0f, όπου η αρνητική τιμή έχει ως αποτέλεσμα την μείωση του ρυθμού, ενώ η θετική την αύξησή του. Το 0 εκφράζει την φυσιολογική ροή του ήχου. Σημειώστε ότι αυτό να χρησιμοποιηθεί για να φτιάξετε διαφορετικές εκδόσεις ενός ήχου, ένα πιστόλι και ένα τουφέκι, για παράδειγμα, θα μπορούσαν να χρησιμοποιούν το ίδιο εφέ, με την αναπαραγωγή του σε διαφορετικό επίπεδο.

- Το τελευταίο όρισμα σας επιτρέπει να καθορίσετε αν ο ήχος θα επαναλαμβάνεται.

4.3 Κάμερες

Μια τρισδιάστατη σκηνή μπορεί να περιλαμβάνει πολλές κάμερες, φώτα και αντικείμενα. Δημιουργώντας μερικές κλασεις για να παρουσιάσετε αυτά τα αντικείμενα μπορείτε να κάνετε το παιχνίδι σας ευκολότερο στην διαχείρηση.
Ανάλογα με το είδος του παιχνιδιού που θα δημιουργήσετε, ίσως θελήσετε διαφορετικό τύπο κάμερας, όπως μια κάμερα με προκαθορισμένη θέση, μια κάμερα πρώτου ή τρίτου ποσώπου, μια πραγματικού χρόνου στρατιγικής κάμερα (RTS) και ούτω καθεξής. Με τόσα διαφορετικά είδη, είναι χρήσιμο να δημιουργήσετε μια βασική κάμερα που θα μπορεί να επεκταθεί για να δημιουργήσετε κι άλλα είδη. Εμείς θα δούμε πως λειτουργεί μια απλή σταθερή κάμερα και μια κάμερα τρίτου προσώπου (TPS).

4.3.1 Απλή 3D κάμερα

Για την απεικόνιση των μοντέλων στην σκηνή του κόσμου, χρειάζεται να δημιουργήσουμε την κάμερα η οποία χρησιμεύει για αυτόν ακριβώς το σκοπό. Ας θυμηθούμε την μέθοδο Draw:

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}
```

Το πρώτο πράγμα που χρειάζεται προσοχή, είναι η παράμετρος που λαμβάνει η μέθοδος Draw. Η παράμετρος αυτή είναι τύπου GameTime και αντιπροσωπεύει τον χρόνο που πέρασε από την εκτέλεση του παιχνιδιού. Γιατί όμως χρειάζεται μια μεταβλητή που κρατάει το χρόνο αυτό. Ο λόγος είναι ότι κάθε υπολογιστής έχει διαφορετικές ταχύτητες. Αυτή η μεταβλητή λοιπόν, βοηθάει να καθορίσουμε πότε τα κινούμενα σχέδια (animations) και άλλα γεγονότα πρέπει να συμβούν με βάση τον χρόνο εκτέλεσης του παιχνιδιού και όχι της ταχύτητας του επεξεργαστή.
Στο τέλος της μεθόδου, καλούμε την βασική μέθοδο Draw ολόκληρου του παιχνιδιύ, η οποία είναι απαραίτητη για να κάνουμε πολλαπλές κλήσεις στις Draw μεθόδους για τα περιεχόμενα του παιχνιδιού και για άλλα αντικείμενα.
Τέλος, η κλήση στο Clear που γίνεται χρησιμοποιώντας την ιδιότητα των γραφικών αντικειμένων, GraphicsDevice. Αυτή η ιδιότητα αντιπροσωπεύει την πραγματική συσκεύη γραφικών του υπολογιστή, του Xbox 360 ή του Zune και επιτρέπει την αναπαράσταση των διάφορων μοντέλων στην οθόνη.

Η μέθοδος Clear, ουσιαστικά σβήνει τα πάντα από τη σκηνή και την γεμίζει με το χρώμα που ορίζεται. Σε αυτήν την περίπτωση είναι το CornFlowerBlue. Μπορούμε να αλλάξουμε το χρώμα ανάλογα με τις προτιμήσεις μας όπως για παράδειγμα σε Color.Purple.
Ενώ εμείς βλέπουμε μια βαρετή σκηνή με το χρώμα που έχουμε επιλέξει, το Xna δουλεύει για να δώσει αυτή τη σκηνή. Καλεί 60 φορές το δευτερόλεπτο τον βρόχο του παιχνιδιού (game loop), σβήνοντας τα πάντα από τη σκηνή και τη χρώματίζει. Επίσης, καλεί και την μέθοδο Update 60 φορές το δευτερόλεπτο για να ελέγξει αν έχει πατηθεί το πλήκτρο Back από το controller του Xbox 360.
Άρα αφού το game loop καλείται 60 φορές το δευτερόλεπτο και καλεί και τις δύο μεθόδους Draw και Update, γιατί θέλουμε να να καθαρίσει τη σκηνή κάθε φορα; Ίσως ακούγεται ανεπαρκες το να καθαρίζει τη σκηνή και να την ξαναγεμίζει με ολόκληρη τη σκηνή και άλλα αντικείμενα για κάθε καινούργιο πλαίσιο (frame), είναι πολύ πιο αποτελεσματικό όμως, από το να προσπαθήσει να παρακολουθήσει οτιδήποτε κινείται στη σκηνή από το ένα frame στο άλλο, να μετακομίσει τα αντικείμενα από τις θέσεις τους και να ζωγραφίσει οτιδήποτε ήταν πίσω από αυτά στο παρελθόν στη θέση από την οποία μετακινήθηκε. Αν αφαιρέσουμε την Clear, το Xna δεν θα σβήνει την σκηνή πριν ζωγραφίσει κάθε frame και θα αποφέρει απροσδόκητα αποτελέσματα.

Τι είναι το frame; Όπως είπαμε, το Xna εξ' ορισμού καθαρίζει την σκηνή και την ξαναζωγραφίζει κάθε φορά που καλείται η Draw. Μια σκηνή που έχει ως αποτέλεσμα μια από αυτές τις κλήσεις αναφέρεται ως frame. Μπορούμε να φανταστούμε ένα παιχνίδι δύο διαστάσεων στο Xna σαν ένα βιβλίο κινουμένων σχεδίων όπου γυρνάμε γρήγορα τις σελίδες του για να μας δώσει έτσι την ψευδαίσθηση της κίνησης. Το Xna λοιπόν, κάνει αυτό ακριβώς. Εξήντα φορές το δευτερόλεπτο η σκηνή καθαρίζεται και ξαναζωγραφίζεται μια καινούργια. Όταν αυτή η νέα σκηνή ξαναφτιαχτεί με κάποιον από τους χαρακτήρες που έχει, σε μια νέα θέση, μας δίνει την ψευδαίσθηση της κίνσης.
Τα πολλαπλά frame δημιουργούν animation σε ένα παιχνίδι και ο αριθμός των frame που ζωγραφίζονται το δευτερόλεπτο αντιπροσωπεύει αυτό που καλείται framerate για το παιχνίδι (60 fps = 60 frames per second).

Αφού λοιπόν προσθέσατε το υλικό σας στο Solution, σε αυτήν τη περίπτωση έναν κύβο, αναγνωρίστηκε από το Content Pipeline και κατανοήσατε την λειτουργία την μεθόδου Draw, δεν μένει τίποτα άλλο από το να φορτώσετε και να εμφανίσετε τον κύβο αυτό στην σκηνή. Πριν λοιπόν επεξεργαστείτε το μοντέλο μέσω κώδικα, θα πρέπει πρώτα να φορτωθεί από το Content Pipeline σε μεταβήτές που θα χρησιμοποιήσετε για να την επεξεργαστείτε.
Το εξ' ορισμού αντικείμενο που χρησιμοποιείται για την αποθήκευση ενός μοντέλου είναι το Model. Προσθέστε λοιπόν στην αρχή του κώδικα της βασικής κλάσης του παιχνιδιού, στις δηλώσεις μεταβλητών το εξής:

```
Model mymodel;
```

Τώρα λοιπόν, θα χρειαστεί να εισάγετε το πραγματικό μοντέλο μέσα σε μια μεταβλητή Model που ονομάζεται mymodel. Για να έχετε πρόσβαση στα δεδομένα από το Content Pipeline, χρησιμοποιείτε την ιδιότητα Content της κλάσης Game. Αυτή η ιδιότητα είναι τύπου ContentManager και παρέχει πρόσβαση σε όλα τα αντικείμενα που εισάγωνται στο Content Pipeline. Η κλαση ContentManager έχει μια Load μέθοδο που θα σας επιτρέψει να εισάγετε το περιεχόμενο που επιθυμείτε.
Όπως έχει αναφερθεί και προηγουμένως, όλα τα γραφικά, οι ήχοι και τα υπόλοιπα αντικείμενα που χρησιμοποιείτε, θα πρέπει να εισάγονται μέσα από την μέθοδο LoadContent. Προσθέστε λοιπόν σε αυτή τη μέθοδο το εξής:

```
mymodel = Content.Load<Model>(@"model\Box");
```

Αυτή η παράμετρος που περάστηκε στην μέθοδο Content.Load είναι το μονοπάτι στο αρχείο του μοντέλου, ξεκινόντας από τον κόμβο Content του Solution Explorer. Το σύμβολο @ προκαλεί την συμβολοσειρά που ακολουθεί να εκτελεστεί κατά γράμμα.
Παρατηρείστε επίσης, ότι στην κλήση αυτής της μεθόδου η παράμετρος που χρησιμοποιείται αντιπροσωπεύει το όνομα του αντικειμένου και όχι το όνομα όλου του αρχείου.
Η μέθοδος Load της κλάσης ContentManager είναι μια γενική μέθοδος που απαιτεί έναν τύπο παραμέτρου που υποδεικνύει το είδος της μεταβλητής που θέλουμε να έχουμε πρόσβαση. Σε αυτήν την περίπτωση έχουμε να κάνουμε με έναν κύβο και περιμένουμε να μας επιστραφεί ένα αντικείμενο τύπου Model.
Το αρχείο του κύβου λοιπόν έχει εισαχθεί μέσα στην μεταβήτη mymodel και είναι έτοιμο να χρησιμοποιηθεί. Η αναπαράσταση των αντικειμένων όπως είπαμε, θα πρέπει να γίνει από την μέθοδο Draw. Πριν από αυτό όμως χρειάζονται κάποιες μεταβλητες που θα χρησιμοποιηθούν για την εμφάνιση του κύβου στη σκηνή. Αρχικά πρέπει να ορίσουμε τη θέση του. Στην αρχή του Game1 λοιπόν, εκεί που δηλώσατε και το μοντέλο, προσθέστε από κάτω:

```
Vector3 modelposition = Vector3.Zero;
```

Εδώ λοιπόν, ορίσατε τρεις συντεταγμένες (για τις τρεις διαστάσεις του χώρου), τις x, y και z. Αυτές τις συντεταγμένες τις θέσατε ίσες με μηδέν (`Vector3.Zero`), στη μέση δηλαδή του τρισδιάστατου κόσμου.
Στη συνέχεια χρειάζεται να ορίσετε κάποιες συντεταγμένες που αφορούν την κάμερα. Προσθέστε από κάτω τα εξής:

```
Vector3 camPosition = new Vector3(0f, 60f, 160f);
Vector3 camLookAt = new Vector3(0f, 50f, 0f);
```

Το camPosition και το camLookAt, ορίζουν που βρίσκεται η κάμερα μέσα στον τρισδιάστατο κόσμο και που κοιτάζει.
Πιο συγκεκριμένα, το camPosition ορίζει τη θέση της κάμερας και δέχεται τρεις τιμές τύπου float. Η πρώτη έχει να κάνει με την μετακίνηση της (shift) δεξιά η αριστερα, την ορίσαμε μηδέν που σημαίνει πως δεν θα έχει μετακίνηση από το κέντρο του κόσμου δεξιά η αριστερά. Η δεύτερη, ορίζει το ύψος που βρίσκεται η κάμερα που σημαίνει στην περίπτωση αυτή, ότι θα είναι 60 μονάδες πάνω από το κέντρο του τρισδιάστατου κόσμου. Η τελευταία τιμή, ορίζει το βάθος, πόσο μπροστά η πίσω δηλαδη, θα βρίσκεται η κάμερα από το κέντρο του κόσμου. Οι θετικές τιμές σημαίνουν κοντά στην οθόνη ενώ οι αρνητικές, μακριά.
Το camLookAt, ορίζει τον προσανατολισμό, τι βλέπει δηλαδή η κάμερα. Οι τιμές που παίρνει έχουν να κάνουν πάλι με τους άξονες x, y και z. Δηλαδή μετακίνηση δεξιά ή αριστερά, ύψος και βάθος. Σ' αυτή την περίπτωση η κάμερα κοιτάζει 50 μονάδες πάνω από το κέντρο του κόσμου.
Τώρα πρέπει να φτιάξουμε δύο κλάσεις, οι οποίες θα χρησιμοποιηθούν για να μας δείξουν την προοπτική της κάμερας, δηλαδή πως η κάμερα βλέπει τη σκηνή. Προσθέστε από κάτω τα εξής:

```
Matrix cameraProjectionMatrix;
Matrix cameraViewMartix;
```

Χρησιμοποιούμε πίνακες για μεγαλύτερη ευκολία στον χειρισμό. Για παράδειγμα, για να κάνουμε περιστροφή σε ένα τρισδιάστατο δημείο, ή για να του αλλάξουμε κλίμακα (scale). Αυτές τις δύο κλάσεις θα τις χρησιμοποιήσουμε στην μέθοδο Initialize για να τους κάνουμε αρχικοποίηση. Προσθέστε λοιπόν στην μέθοδο αυτή τον παρακάτω κώδικα:

```
cameraViewMartix = Matrix.CreateLookAt(
                camPosition,
                camLookAt,
                Vector3.Up);
```

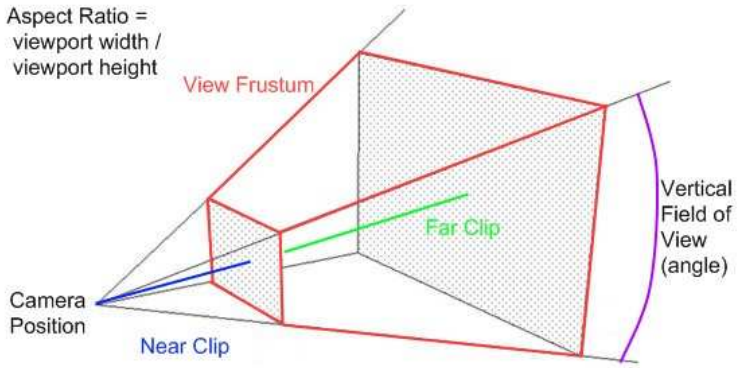
Έτσι, δημιουργήσατε τον πρώτο σας πίνακα. Αυτή είναι μια βοηθητική μέθοδος που δημιουργεί έναν πίνακα (matrix) ο οποίος παρουσιάζει πως μια κάμερα βλέπει τη σκηνή. Τα ορίσματα που παίρνει, είναι η θέση που ορίσατε προηγουμένως όπως επίσης και η προοπτική της κάμερας και ένα διάνυσμα Up που αντιστοιχεί στο πάνω μέρος του τρισδιάστατου κόσμου. Υπάρχει λοιπόν, βοηθητική μέθοδος για αυτό που ονομάζεται `Vector3.Up`.
Από κάτω προσθέστε:

```
cameraProjectionMatrix=Matrix.CreatePerspectiveFieldOfView(
          MathHelper.ToRadians(45f),
          graphics.GraphicsDevice.Viewport.AspectRatio,
          1f,10000f);
```

Και αυτή επίσης, είναι μια βοηθητική μέθοδος που ορίζει τον προσανατολισμό της κάμερας. Ορίζει τις παραμέτρους που χρειάζονται για να πάρει τα τρισδιάστατα αντικείμενα που βλέπει η κάμερα και τα ζωγραφίζει ως δισδιάστατες εικόνες που φαίνονται στη σκηνή του κόσμου. Για να το κάνουμε αυτό, χρειαζόμαστε ένα FieldOfView σε Radians (ακτίνια), δηλαδή πόσο γενικά/πλατειά βλέπει αυτή η κάμερα. Χρησιμοποιούμε την μέθοδο MathHelper.ToRadians και έχουμε οπτική των 45f βαθμών που μετατρέπονται σε ακτίνια. Εναλλακτικά θα μπορούσαμε να γράψουμε:

```
MathHelper.PiOver4; (pi/4=45 degrees)
```

Στη συνέχεια χρησιμοποιούμε το AspectRatio που ορίζει πόσο πλατειά ή ψηλή πρέπει να είναι η τελική δισδιάστατη εικόνα στη σκηνή, είναι ουσιαστικά ο λόγος μεταξύ δύο συντεταγμένων, ύψους και πλάτους και βρίσκεται στο GraphicsDevice στην κλάση Viewport. Τα δύο τελευταία ορίσματα, καθορίζουν το πιο κοντινό (1f) και το πιο μακρυνό (10000f) που έχει οπτική η κάμερα στον τρισδιάστατο κόσμο (Εικ.18).



Εικ.18 3D Κάμερα

Το επόμενο βήμα είναι να φτιάξουμε μια μέθοδο Draw που θα χρησιμοποιεί όλα όσα ορίσαμε προηγουμένως. Φτιάχνουμε μια καινούργια Draw για να μην καλούμε την draw για κάθε ένα από τα μοντέλα. Κάτω λοιπόν από την μέθοδο Draw, προσθέστε τον εξής κώδικα:

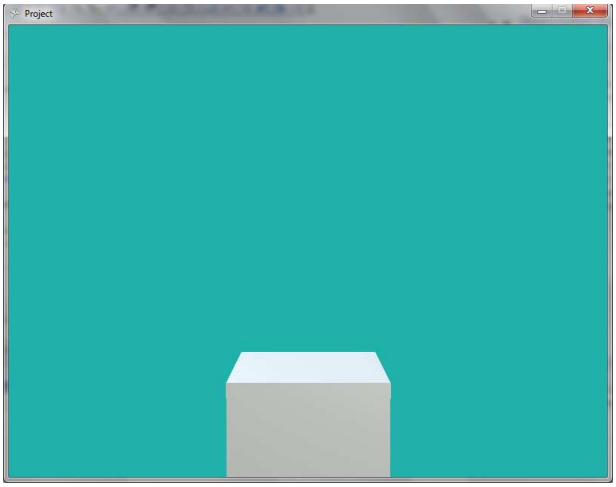
```
void DrawModel(Model model, Vector3 modelPosition)
  {
      foreach(ModelMesh mesh in model.Meshes)
      {
          foreach(BasicEffect effect in mesh.Effects)
          {
              effect.EnableDefaultLighting();
```

```
                effect.PreferPerPixelLighting=true;
                effect.World = Matrix.CreateTranslation(modelPosition);
                effect.Projection = cameraProjectionMatrix;
                effect.View = cameraViewMartix;

            }
            mesh.Draw();
        }
    }
```

Ας τα πάρουμε με τη σειρά. Η μέθοδος αυτή, δέχεται δύο παραμέτροους. Η πρώτη είναι το μοντέλο, ενώ η δεύτερη είναι το σημείο που θα βάλει το μοντέλο στον κόσμο. Ο επίσημος τρόπος αναπαράστασης ενός μοντέλου, είναι αυτός που μόλις γράψατε. Κάθε μοντέλο αποτελείται από meshes. Για κάθε mesh πρέπει να ορίσω μια ενέργεια (effect). Ένα effect, προετοιμάζει την μηχανή γραφικών (graphics device) να κάνει render σε όλα τα τρίγωνα που φτιάχνουν ένα mesh. Καθορίζει που είναι το mesh, που είναι η κάμερα και αν υπάρχουν φώτα στη σκηνή που εμπλουτίζουν το mesh. Όταν ορίζουμε το effect τότε ζωγραφίσζουμε το mesh. Οπότε, ξεκινάμε με έναν βρόχο (`foreach`). Ο πρώτος βρόχος, θα ενεργήσει για κάθε mesh που υπάρχει μέσα στο μοντέλο. Το ModelMesh αντικείμενο, έχει το όνομα mesh. Στη συνέχεια ορίζουμε το effect. Χρησιμοποιούμε μια συντόμευση που μας παρέχει το Xna που ονομάζεται BasicEffect. Επειδή υπάρχει το ενδεχόμενο να είναι περισσότερα από ένα, χρειαζόμαστε άλλον ένα βρόχο (loop). Στη δεύτερη foreach, κάνουμε μια επανάληψη μέσω των basiceffect αντικειμένων στο mesh έτσι ώστε να τα ορίσουμε. Ορίζουμε λοιπόν το προσωρινό effect, μέσω των παραμέτρων `EnableDefaultLighting()` και `PreferPerPixelLighting`. Αυτές οι δύο παράμετροι είναι ο βασικός φωτισμός στο Xna. Στη συνέχεια χρησιμοποιούμε ένα world matrix που ορίζει που βρίσκεται το μοντέλο μέσα στον κόσμο, τι μέγεθος έχει και αν έχει περιστραφεί καθόλου (όπως ξαναείπαμε, ένας πίνακας χρησιμοποιείται για να ορίζει set από τρισδιάστατα σημεία). Ακόμα δεν ασχολούμαστε με την περιστροφή ή το μέγεθος. Όμως χρειάζεται να ορίσουμε την θέση. Το translation είναι κάτι σαν αλλαγή θέσης ή αλλιώς γλίστρημα. Περνάμε σε ένα διάνυσμα που είναι το modelPosition και το μετατρέπουμε σε πίνακα. Χρειαζόμαστε άλλους δύο πίνακες για να ζωγραφίσουμε το μοντέλο, οι οποίοι έχουν να κάνουν με την κάμερα και είναι το ViewMatrix και το ProjectionMatrix που αρχικοποιήσαμε στην μέθοδο Initialize. Τέλος, το `mesh.Draw()`, παίρνει όλα όσα ορίσαμε και τα 3D mesh δεδομένα και τα στέλνει στην μηχανή γραφικών (graphics device) για να τα αναπαραστήσει στη σκηνή. Το τελευταίο κομμάτι είναι να καλέσουμε την μέθοδο DrawModel που μόλις φτιάξαμε. Προσθέστε τον παρακάτω κώδικα στην μέθοδο Draw:

```
DrawModel(mymodel, modelposition);
```

Η draw θα καλέσει την μέθοδο DrawModel ένα-ένα frame και θα την εμφανίσει στη σκηνή. Εκτελέστε το πρόγραμμα που μόλις φτιάξατε και δείτε το αποτέλεσμα (F5 ή το play button). Βλέπετε έναν κύβο στο κέντρο του κόσμου σας (Εικ. 19).



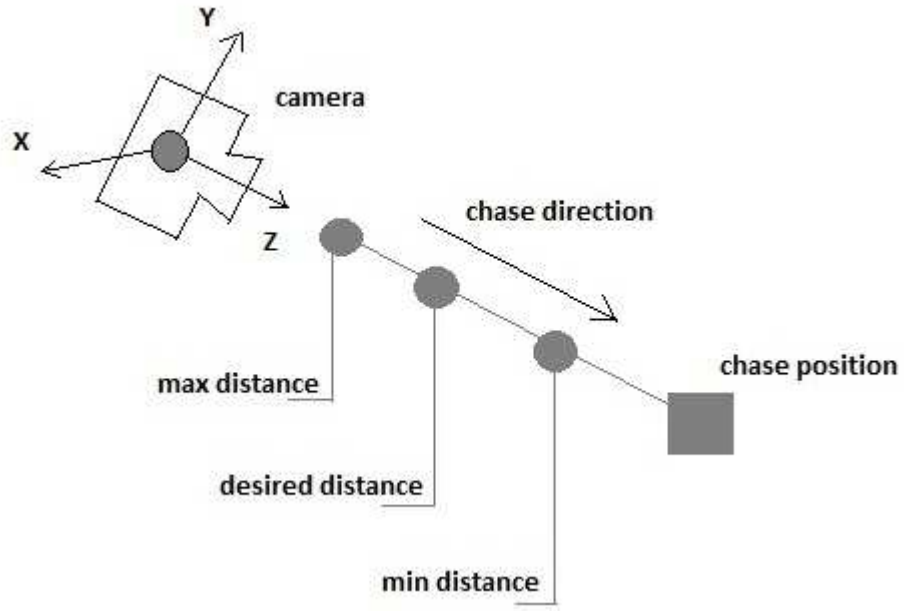
Εικ.19 Κύβος

4.3.2 Κάμερα τρίτου προσώπου

Σε αυτή την ενότητα θα αναφερθούμε στον τρόπο λειτουργίας μιας κάμερας τρίτου προσώπου (Third Person Camera ή Chase Camera). Στόχος της κάμερας τρίτου προσώπου είναι να ακολουθήσει ένα αντικείμενο, ενώ κινείται. Επίσης η απόσταση κατά την οποία η κάμερα ακολουθεί ένα αντικείμενο πρέπει να είναι μεταβλητή. Διαφορετικά, φαίνεται ότι το αντικείμενο δεσμεύεται από την κάμερα, με αποτέλεσμα η κίνηση της κάμερας που ακολουθεί το αντικείμενο να είναι απότομη.
Για να κάνετε μια κάμερα να ακολουθεί ένα αντικείμενο πρέπει να ρυθμίσετε τις εξής παραμέτρους:

- Τον στόχο (chase position), η οποία είναι η θέση του στόχου που η κάμερα θα πρέπει να ακολουθεί.
- Την κατεύθυνση προς τον στόχο (chase direction), η οποία είναι η κατεύθυνση που πρέπει να ακολουθήσει η κάμερα ούτως ώστε να φτάσει το αντικείμενο που στοχεύει.
- Η ταχύτητα με την οποία κυνηγάμε το στόχο (chase speed).
- Η απόσταση που υπάρχει από τον στόχο (chase distance), η οποία είναι η απόσταση ανάμεσα στην κάμερα και τη θέση που κυνηγάει.

Εδώ, θα χαρακτηρίσουμε την απόσταση αυτή με τρεις μεταβλητές: την ελάχιστή, την επιθυμητή και την μέγιστη απόσταση ανάμεσα στην κάμερα και το αντικείμενο που στοχεύει. Η εικόνα 20 παρουσιάζει αυτές τις τρεις παραμέτρους.



Εικ.20 Κάμερα Τρίτου Προσώπου

Κάθε φορά που ενημερώνεται η κάμερα, η θέση της πρέπει να υπολογιστεί εκ νέου. Η ιδανική νέα θέση της κάμερας είναι ίση με τη θέση του στόχου της μείον την κατεύθυνσή της πολλαπλασιαζόμενη επί της απόστασης του στόχου. Η επιθυμητή νέα θέση της κάμερας θα είναι η τελική θέση αν ήταν τοποθετημένη σε σταθερή απόσταση από τον κινούμενο στόχο. Ωστόσο, για να επιτρέψουμε στην κάμερα να κινηθεί ομαλά, η απόσταση ανάμεσα στην κάμερα και την θέση του στόχου της μπορεί να ποικίλλει ανάμεσα σε ένα ελάχιστο και ένα μέγιστο εύρος.

Με αυτό τον τρόπο, η νέα θέση της κάμερας υπολογίζεται με μια γραμμική παρεμβολή μεταξύ της τρέχουσας θέσης της και της επιθυμητής θέσης της κάμερας. Η γραμμική παρεμβολή είναι μια παρεμβολή μεταξύ δύο τιμών που κυμαίνονται γραμμικά με βάση ένα καθορισμένο βάρος, το οποίο είναι συνήθως ένας αριθμός κινητής υποδιαστολής μεταξύ των 0 και 1. Για παράδειγμα, μια γραμμική παρεμβολή μεταξύ των αριθμών 10 και 20 χρησιμοποιώντας τιμή βάρους 0,50 είναι το ίδιο με το να πούμε, "Δώσε μου το 50 τοις εκατό μεταξύ του 10 και του 20" που έχει ως αποτέλεσμα την τιμή 15. Οι γραμμικές παρεμβολές που χρησιμοποιούν τα βάρη 0, 0.25 και 1 έχουν ως αποτέλεσμα τις τιμές 10, 12.5, και 20, αντίστοιχα, καθώς οι τιμές αυτές είναι 0.25 και 100 τοις εκατό μεταξύ των 10 και 20.

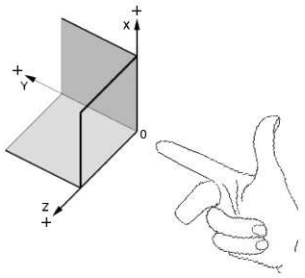
Το βάρος που χρησιμοποιείται για να παρεμβάλλουμε τη θέση της κάμερας υπολογίζεται με βάση τον χρόνο που πέρασε από την τελευταία ενημέρωση και την ταχύτητα της κάμερας. Ωστόσο, επειδή το παρεμβαλλόμενο βάρος πρέπει να κυμαίνεται μεταξύ 0 και 1, θα πρέπει να καταπνίξετε την αξία του, περιορίζοντας το φάσμα μεταξύ 0 και 1. Μπορείτε να χρησιμοποιήσετε την μέθοδο Lerp (Vector3) του Xna για να σας βοηθήσει να παρεμβάλλετε τα διανύσματα. Ως αποτέλεσμα, οι μικρότερες τιμές για την chaseSpeed (ταχύτητα με την οποία κυνηγάει η κάμερα) θα οδηγήσουν σε μια αργή αντίδραση της κάμερας, και σε περισσότερο χρόνο που χρειάζεται για να ξεκινήσει η κάμερα να κινείται μετά από το αντικείμενο. Υψηλότερες τιμές για την ταχύτητα αυτή οδηγούν σε ταχεία αντίδραση της κάμερας και σε μικρότερο χρονικό διάστημα μεταξύ της κίνησης της κάμερας και της κίνησης του αντικειμένου. (Αυτή η αντίδραση της κάμερας συχνά αναφέρεται ως lag).

Ένα άλλο χαρακτηριστικό γνώρισμα που πρέπει να προσθέσετε στην κάμερα είναι η ικανότητα να περιστρέφεται γύρω από το στόχο της. Γι 'αυτό το χαρακτηριστικό, θα θελήσετε έναν τρόπο για να καθορίσετε τη μέγιστη ταχύτητα περιστροφής της κάμερας και της τρέχουσας περιστροφής της κάμερας. Επίσης, επειδή θέλετε η περιστροφή της κάμερας σας να ξεκινάει και να σταματάει ομαλά, θα πρέπει να παρακολουθείτε την τρέχουσα ταχύτητα περιστροφής. Το επιτρεπόμενο φάσμα περιστροφής

κάμερας καθορίζεται μεταξύ της μέγιστης περιστροφής και της ελάχιστης. Εάν η περιστροφή της κάμερας βρίσκεται εκτός των ορίων, πρέπει να το θέσετε έτσι ώστε να συσφίγγονται τα όρια αυτής της περιοχής. Χρειάζεται ένα διάνυσμα που θα αποθηκεύει την τρέχουσα περιστροφή της κάμερας, όπου τα X,Y,Z και οι συνιστώσες του εν λόγω διανύσματος αντιπροσωπεύουν την γωνία της περιστροφής γύρω από τους άξονες Strafe, Up και Heading της κάμερας.
Τέλος, χρειάζεστε ένα διάνυσμα για την αποθήκευση της ταχύτητας με την οποία η γωνία περιστροφής της κάμερας ενημερώνεται.

4.3.3 Σημεία που αξίζει να σταθούμε:

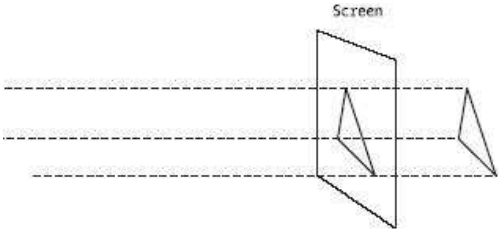
Όταν γίνεται εισαγωγή των μοντέλων στο Xna, η εξ' ορισμού θέση τους είναι το 0,0,0 δηλαδή το κέντρο του κόσμου. Αυτό ισχύει για όλα τα μοντέλα. Το Xna δουλεύει επίσης εξ' ορισμού, με τον κανόνα του δεξιού χεριού για το σύστημα των συντεταγμένων, όπως φαίνεται και στο σχήμα (Σχ. 3).



Σχ.3 Κανόνας δεξιού χεριού

Όσο μεγαλύτερη αρνητική τιμή, τόσο πιο μακριά θα είναι το μοντέλο από την σκηνή, δηλαδή τόσο πιο μακριά θα το βλέπουμε.
Το Xna υσποστηρίζει δύο διαφορετικούς τρόπους προβολής.

- **Perspective Projection**: υπολογίζει την απόσταση από τον άξονα Z και ρυθμίζει τα μοντέλα αναλόγως (Εικ. 4).
- **Orthogonal Projection**: σε αυτήν την περίπτωση η Z συνιστώσα αγνοείται. Τα αντικείμενα έχουν το ίδιο μέγεθος και σε κοντινές και σε μακρυνές αποστάσεις. Χρησιμοποιείται κυρίως για 2D (Εικ. 21).

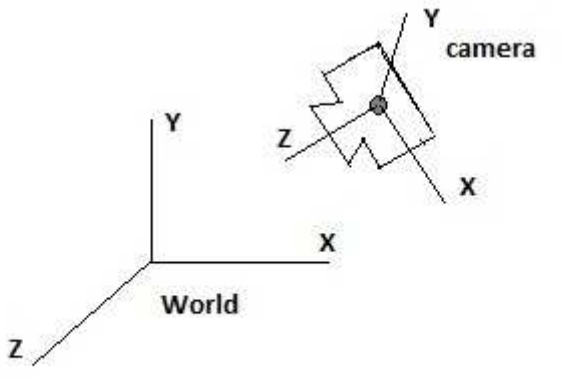


Εικ.21 Orthogonal projection

Σύστημα Συντεταγμένων της κάμερας:

Κάθε φορά που αλλάζετε τις ρυθμίσεις της κάμερας με την μέθοδο LookAt, θα πρέπει να υπολογίζετε τα διανύσματα των τριών αξόνων X (strafe), Y (up) και Z (heading). Στην εικόνα 22 βλέπουμε το σύστημα συνεταγμένων της κάμερας, τοποθετημένο στο σύστημα συντεταγμέων του κόσμου. Σημειώστε ότι, επειδή αυτά τα διανύσματα συνθέτουν το σύστημα συντεταγμένων της κάμερας πρέπει να είναι ενιαία

(και το μήκος τους να είναι ακριβώς 1) και κάθετα μεταξύ τους. Μπορείτε να χρησιμοποιήσετε διανύσματα ένωσης για να αντιπροσωπεύσετε τις κατευθύνσεις γιατί το μέγεθος του διανύσματος δεν παίζει ρόλο σε αυτήν την περίπτωση.



Εικ.22 Σύστημα Συντεταγμένων κάμερας στον 3D κόσμο.

Μπορείτε να υπολογίσετε τα διανύσματα της κάμερας ως εξής:

Heading: το διάνυσμα αυτό είναι η κατεύθυνση από τη θέση της κάμερας προς τη θέση του στόχου της. Περιγράφει την κατεύθυνση που αντιμετωπίζει η κάμερα. Μπορείτε να υπολογίσετε την κατεύθυνση αυτή αφαιρώντας τη θέση της κάμερας από τη θέση που βρίσκεται ο στόχος.

Up: το διάνυσμα αυτό καθορίζει την από πάνω κατεύθυνση της κάμερας και χρησιμοποιείται για τον προσανατολισμό της. Για παράδειγμα, μπορείτε να χρησιμοποιήσετε το διάνυσμα (0,1,0) για να προσανατολίσετε το διάνυσμα αυτό της κάμερας σαν τον Y άξονα του κόσμου.

Strafe: είναι το διάνυσμα που είναι κάθετο προς τα δύο προηγούμενα. Αυτό μπορεί να βρεθεί με την πράξη του εξωτερικού γινομένου το οποίο υπολογίζει ένα διάνυσμα που είναι κάθετο προς τα άλλα δύο την ίδια στιγμή. Η μέθοδος Cross της κλάσης Vector3 του Xna εκτελεί την πράξη του εξωτερικού γινομένου. Σημειώστε ότι, τα διανύσματα που χρησιμοποιούνται στο εξωτερικό γινόμενο πρέπει να έχουν ενιαία διανύσματα (ή θα πρέπει να κανονικοποιήσετε (normalize) το διάνυσμα που προκύπτει μετά την πράξη) και η σειρά με την οποία πέρασε με τη μέθοδο Cross αλλάζει την κατεύθυνση του διανύσματος που προκύπτει.

Αυτά τα τρία διανύσματα από το σύστημα συντεταγμένων της κάμερας χρησιμοποιούνται όποτε χρειάζεστε να μετακινήσετε (transform) την κάμερα βασιζόμενη στους άξονές της. Για παράδειγμα όποτε θέλετε να μετακινήσετε την κάμερα προς την κατεύθυνση που κοιτάει.
Όπως αναφέραμε, τα τρία αυτά διανύσματα θα πρέπει να είναι κάθετα μεταξύ τους, κάτι το οποίο δεν είναι πλήρως διασφαλισμένο. Για παράδειγμα, υποθέστε πως η κάμερα κοιτάζει κυρίως μπροστά, αλλά και ελαφρώς προς τα πάνω. Αν καλούσατε την μέθοδο SetLookAt και χρησιμοποιούσατε το κοινό Up διάνυσμα (0,1,0) σαν τρίτο όρισμα, θα μπαίνατε σε μπελάδες γιατί αυτό το διάνυσμα δεν είναι εντελώς κάθετο στο heading διάνυσμα της κάμερας. (Το strafe διάνυσμα θα είναι εντελώς κάθετο στο heading διάνυσμα όπως και στο up διάνυσμα γιατί προέρχεται από το εξωτερικό τους γινόμενο). Για να είστε σίγουροι ότι το Up διάνυσμα είναι κάθετο στο Heading διάνσυσμα αφού πρώτα έχετε υπολογίσει το strafe, πρέπει να υπολογίσετε εάν νέο Up διάνυσμα χρησιμοποιώντας μια δεύτερη πράξη εξωτερικού γινομένου μεταξύ των strafe και heading διανυσμάτων ως ακολούθως:

```
upVec = Vector3.Cross(strafeVec,headingVec);
```

Αυτό θα σας επιστρέψει τρία διανύσματα για τα οποία μπορείτε να είστε σίγουροι ότι είναι κάθετα το ένα στο άλλο.

Αναφορές:

Alexandre Santos Lobao, Bruno Evangelista, Jose Antonio Leal de Farias and Riemer Grootjans "Xna 3.0 Game Programming, From Novice to Professional" (Apress),

Aaron Reed "O' Reilly Learning Xna 3.0" ,

Riemer Grootjans "Xna 3.0 Game Programming Recipies, A Problem-Solution Approach" (Apress).

Youtube video tutorials (www.youtube.com),

Microsoft Xna official home page (http://msdn.microsoft.com),

Xna Creators Club Online (http://creators.xna.com).

5

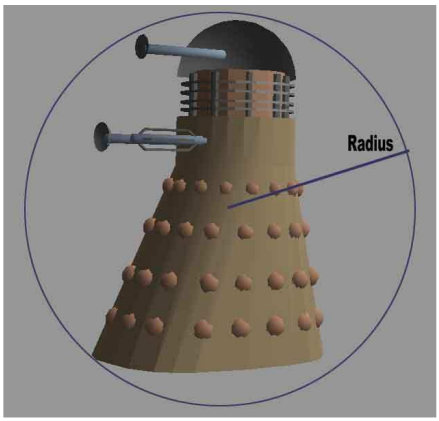
Έλεγχος για σύγκρουση

Προχωρώντας στο επόμενο βήμα, για να γίνει το παιχνίδι μας πιο εντυπωσιακό και λειτουργικό, πρέπει να ελέξουμε αν τα αντικείμενα συγκρούονται μεταξύ τους ή αν βγαίνουν έξω από τα πλαίσια του παραθύρου του παιχνιδιού. Ο έλεγχος για τη συγκρουση των αντικειμένων είναι ένα πολύ κρίσιμο συστατικό. Έχετε παίξει ποτέ κάποιο παιχνίδι βολής (shooter game) όπου χτυπάτε το στόχο σας χωρίς να συμβαίνει τίποτα; Η ένα αγωνιστικό παιχνίδι (racing game) όπου κατευθύνεστε προς τον τοίχο και τελικά περνάτε από μέσα; Αυτό είναι εξοργιστικό για τους παίκτες και είναι αποτέλεσμα φτωχής υλοποιήσης για την ανίχνευση συγκρούσεων.

5.1 Τα Bounding Shapes

Η ανίχνευση συγκρούσεων (collision detection) μπορεί να φτιάξει ή να χαλάσει μια εμπειρία με κάποιο παιχνίδι. Αυτό συμβαίνει διότι όσο πιο ακριβείς και ξεκάθαροι είναι οι αλγόριθμοι ανίχνευσης συγκρούσεων, τόσο πιο αργό γίνεται το παιχνίδι. Αυτό είναι μια ξεκάθαρη εξισορρόπηση μεταξύ της απόδοσης και της ακρίβειας όταν αναφερόμαστε στο collision detection.
Ένας από τους πιο απλούς και γρήγορους τρόπους για να εφαρμόσουμε collision detection είναι μέσω του αλγορίθμου των Bounding Shapes. Ουσιαστικά, τοποθετούμε γύρω από τα τρισδιάστατα αντικείμενα ένα Bounding shape (σφαίρα ή κουτί). Αν το ένα σχήμα πέσει πάνω στο άλλο, τότε συμβαίνει σύγκρουση.

Bounding Sphere: Μια Bounding Sphere είναι μια σφάιρα η οποία περιλαμβάνει όλη τη γεωμετρία του τρισδιάστατου αντικειμένου. Ορίζεται μέσω μιας κεντρικής θέσης και μιας ακτίνας (Εικ. 23). Το Xna μας παρέχει μια κλάση για την χρήση της που ονομάζεται Bounding Sphere. Το Xna μας παρέχει αυτή την κλάση ανά mesh του μοντέλου ώστε να μην χρειάζεται να το υπολογίσουμε εμείς. Θυμηθείτε ότι τα μοντέλα στο Xna είναι φτιαγμένα από ένα και πάνω mesh. Όταν γίνονται συγκρούσεις θέλουμε μια σφαίρα που να περιλαμβάνει ολόκληρο το μοντέλο, που σημαίνει πως την στιγμή που το μοντέλο φορτώνεται, θέλουμε να περνάμε μέσα από όλα τα meshes του ώστε να αναπτύξουμε ένα κύριο τελικό μοντέλο.



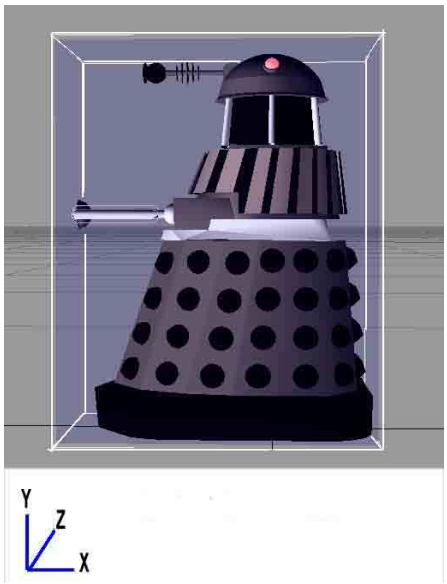
Εικ. 23 Bounding Sphere

Bounding Box: Βασιζόμενοι στο σχήμα της γεωμετρίας του μοντέλου σας, το Bounding Box είναι πιθανό να ταιριάζει καλύτερα από μια σφαίρα. Η κλαση Bounding Box είναι ευθυγραμμισμένη κατά άξονα (Εικ. 24) Κάθε πλευρά του Bounding Box είναι κάθετη προς τον άξονα x, τον άξονα y ή τον άξονα z. Υπάρχουν πολλά πλεονεκτήματα που προσφέρει η χρήση του Bounding Box στον έλεγχο των συγκρούσεων μεταξύ των μοντέλων:

- Η κλαση Bounding Box εφαρμόζει πολύ καλα και ευθύγραμμισμένα τα τετράγωνα σχήματα με τους άξονες x, y ή z. Συγκριτικά με την κλάση Bounding Sphere, η Bounding Box παρέχει πιο τακτοποιημένη εφαρμογή για μη περιστρεφόμενα αντικείμενα.
- Επειδή η κλάση Bounding Box είναι ευθύγραμμη με τους άξονες, μπορείτε να κάνετε ορισμένες μελέτες που καταλήγουν στον έλεγχο τον συγκρούσεων μεταξύ των σταθερών bounding boxes που είναι ταχύτερα από αυτά που είναι περιστρεφόμενα.

Υπάρχουν όμως και κάποια μειονεκτήματα στη χρήση τους:

- Όταν περιστρέψουμε ένα Bounding Box, δεν είναι πια ευθύγραμμο με τους άξονες. Εξαιτίας αυτού, αν περιστρέψουμε ένα μοντέλο στο οποίο έχει γίνει εφαρμογή του Bounding Box θα χρειαστεί να δημιουργήσουμε πάλι το Bounding Box. Αυτό έχει ως αποτέλεσμα την ελάττωση της ταχύτητάς του δεδομένου ότι όλα τα σημεία ενός αντικειμένου επαναλαμβάνονται μέχρι να πάρουν το Bounding Box.
- Αν το μοντέλο στο οποίο έχει εφαρμοστεί το Bounding Box δεν είναι ευθυγραμμισμένο με τους άξονες, το Bounding Box θα έχει κάποιους χώρους κενούς. Το ποσοστό του άδειου χώρου θα είναι το μέγιστο όταν το αντικείμενο περιστροφεί 45 μοίρες από κάποιον άξονα.
- Ο άδειος χώρος σ' ένα Bounding Box μπορεί να έχει ως αποτέλεσμα λάθος τιμές όταν ψάχνει για την σύγκρουση.



Εικ. 24 Bounding Box

5.2 Η μηχανή φυσικής JigLibX

Στην εφαρμογή που δημιουργήθηκε δεν έγινε χρήση των κλάσεων Bounding Box και Bounding Sphere, αλλά εισαγωγή μιας μηχανής φυσικής που μας παρείχε περισσότερες δυνατότητες στη φυσική κίνηση των μοντέλων που χρησιμοποιήθηκαν στη σκηνή του κόσμου. Αυτή η μηχανή ονομάζεται JigLibX.

5.2.1 Μια σύντομη γνωριμία

Το JiglibX είναι μια μηχανή φυσικής η οποία έχει γραφτεί σε C# γλώσσα προγραμματισμού, χρησιμοποιώντας το XNA Framework της Microsoft. Βασίζεται στην μηχανή φυσικής JigLib και βρίσκεται στο στάδιο επέκτασης της. Έχοντας ένα σύστημα για collision και μια rigid body (άκαμπτου σώματος) μηχανή φυσικής, το JigLibX είναι μία από τις ευνοημένες μηχανές φυσικής ανοιχτού κώδικα που έχουν σχεδιαστεί για τη συνεργασία της με το XNA.

Ο στόχος:
Η δημιουργία μιας εξαιρετικής μηχανής φυσικής που είναι φορητή και μπορεί να βελτιστοποιηθεί. Αυτό μπορούμε να το πετύχουμε, έχοντας μια φανταστική κοινωνία όπου ο καθένας συνεισφέρει τις σκέψεις και τις ιδέες για να βοηθήσει όλους μας να πετύχουμε αυτόν τον στόχο.

Εισαγωγή στο JigLibX:
Με το JigLibX, μπορείτε να προσθέσετε μια φυσική συμπεριφορά στο XNA παιχνίδι σας. Αυτή η ενότητα θα περιγράψει εν συντομία πώς μπορείτε να το κάνετε. Ωστόσο, δεν υπάρχει καλή σε βάθος τεκμηρίωση ακόμα, έτσι την ανάγνωση του κώδικα προτιμάται για το πώς λειτουργεί κάτι συγκεκριμένο.

Η λογική:
Η δυναμική rigid body είναι ένα είδος προσομοίωσης που χρησιμοποιείται στα παιχνίδια για να δώσουν την εμφάνιση της «φυσικής» συμπεριφοράς των αντικειμένων. Για κάθε αντικείμενο που θέλετε να προσομοιώσετε, αποτελεί ένα ή περισσότερα σώματα (bodies) που ορίζονται, καθένα από τα οποία έχει ένα ή περισσότερα βασικά στοιχεία (primitives) για σύγκρουση. Για ένα αντικείμενο με περισσότερα από ένα κινητά μέρη, υπάρχει ένα σώμα για κάθε κομμάτι, και τα μέρη είναι δεμένα μεταξύ τους με αρθρώσεις (joints). Σε μια παιδική χαρά για παράδειγμα, θα είχατε ένα σώμα με ένα collision skin για τη βάση, ένα άλλο σώμα με άλλο collision skin για το μέρος που όντως κινείται και θα τα είχατε συνδέσει μεταξύ τους με μια άρθρωση.
Τα στοιχεία για την σύγκρουση (collision primitives), είναι βασικά σχήματα, όπως κουτιά, σφαίρες, κάψουλες (ένας κύλινδρος με στρογγυλές άκρες, όπως είναι ορισμένα είδη χαπιών), ακτίνες / ευθύγραμμα τμήματα, καθώς και ελαφρά πιο προηγμένα σχήματα, όπως planes, heightmaps και triangle meshes. Για να περιγράψετε την προσέγγιση ενός δεδομένου φυσικού σώματος (ενός αυτοκινήτου, μιας κούνιας, μιας πόρτας, κλπ), ίσως να χρειάζεται να χρησιμοποιήσετε περισσότερα από ένα σχήματα και να τα βάλετε όλα μαζί. Αυτό υλοποιείται με τη δημιουργία ενός collision skin και την ανάθεσή του στο σώμα. Σημειώστε ότι όλα τα σχήματα σε ένα collision skin κινούνται μαζί και μαζί με το σώμα. Αν θέλετε διαφορετικά μέρη που κινούνται χωριστά (την πόρτα σε ένα αυτοκίνητο, τους κρίκους μιας αλυσίδας), τότε πρέπει να δημιουργήσετε περισσότερα από ένα σώματα, καθένα από τα οποία έχει διαφορετικό collision skin, και να τα «αγκιστρώσετε» μεταξύ τους χρησιμοποιώντας ένα είδος από αρθρώσεις (joints). Το γεγονός ότι κάθε σώμα έχει ένα σταθερό, άκαμπτο σχήμα είναι η ο λόγος που αυτό το είδος της προσομοίωσης ονομάζεται "άκαμπτη δυναμική σώματος" (rigid body dynamics).

Η ροή του ελέγχου:
Στην προσομοίωση σας, θέλετε να δημιουργήσετε ένα στιγμιότυπο (instance) συστήματος φυσικής. Το σύστημα αυτό είναι το τμήμα του κώδικα που θα προσομοιώσει την κίνηση των σωμάτων. Θέλετε να

δημιουργήσετε ένα σύστημα σύγκρουσης και να το αντιστοιχίσετε με το σύστημα φυσικής, έτσι ώστε ο προσομοιωτής φυσικής να ξέρει πώς να λάβει συγκρούσεις αντικειμένων σε ένα γεγονός (έτσι τα πράγματα δεν περνάνε μέσα από τους τοίχους, δεν πέφτουν μέσα στο έδαφος, κλπ). Δημιουργείτε έπειτα ένα ή περισσότερα σώματα, καθένα από τα οποία έχει κατά κανόνα ένα collision skin με ένα ή περισσότερα primitives σε αυτό. Όταν τελειώσετε, ενημερώνετε το σύστημα φυσικής να προωθήσει το χρόνο και να καταλάβει πως θα μοιάζει ο κόσμος προσομοίωσης μετά από ένα μικρό χρονικό διάστημα που έχει περάσει. Μπορείτε να το κάνετε αυτό, καλώντας την μέθοδο PhysicsSystem.Integrate () και δίνοντάς της τον χρόνο αυτό που πέρασε για να προχωρήσει. Τυπικά, το βήμα του χρόνου σας θα είναι είτε το χρονικό διάστημα από την τελευταία κλήση της μεθόδου GameComponent.Update () (εάν χρησιμοποιείτε ένα μεταβλητό χρονικό βήμα), ή ένα συγκεκριμένο χρονικό διάστημα (όπως το 1/100 του δευτερολέπτου - 0.01f), και θα μπείτε στον κόσμο περισσότερο από ένα χρόνο για να το κάνετε να «πιάσει» με τη ροή του πραγματικού χρόνου.
Στο Xna παιχνίδι σας, στην μέθοδο LoadContent(), δημιουργείτε το σύστημα φυσικής, το σύστημα συγκρούσεων και το συμπληρώνετε με σώματα που έχουν collision skins. Θυμηθείτε όλα τα σώματα που προσθέσατε και τι μοντέλο θέλετε να συνδυάσετε με κάθε σώμα.
Η μέθοδος PhysicsSystem.Integrate() καλείται στην μέθοδο Update του Xna παιχνιδιού σας.
Στην μέθοδο Draw(), επαναλαμβάνετε για κάθε ένα από τα σώματα, κάνετε εξαγωγή του μετασχηματισμού για το σώμα και το εφαρμόζετε στο μοντέλο που φορτώσατε για το σώμα αυτό. Σημειώστε ότι το JigLibX, προϋποθέτει αρχικά ότι το Y σημαίνει πάνω, επομένως η βαρύτητα θα ενεργήσει προς το –Y δηλαδή προς τα κάτω.

Ελεγκτές:
Εάν δημιουργήσετε ένα σωρό από εμπόδια ή βόλους χρησιμοποιώντας κουτιά και σφαίρες, καθώς επίσης και ένα έδαφος με τη χρήση heightmap ή plane, η προσομοίωσή σας θα δείξει τους βόλους και τα κουτιά να πέφτουν στο έδαφος, και να συγκρούονται μεταξύ τους. Μετά από λίγο, η προσομοίωση θα έρθει σε μια στάση, και τίποτα πιο ενδιαφέρον δεν θα συμβεί. Για να έχετε δράση, θα θελήσετε να κάνετε δύο πράγματα:

- Να προσθέσετε νέα σώματα στο σύστημα.
- Να επηρεάσετε τα σώματα που υπάρχουν ήδη.

Για να προσθέσετε ένα νέο σώμα, το κάνετε μέσα από την μέθοδο Update (), πριν καλέσετε την Integrate(). Δημιουργείστε ένα νέο όργανο, δώστε ένα CollisionSkin, και προσθέστε το στο PhysicsSystem. Αυτός είναι ένας πολύ καλός τρόπος για την προσθήκη, για παράδειγμα, μιας μπάλας κανονιού που εκτοξεύεται από πυροβολισμό έξω από ένα κανόνι (υποθέτοντας ότι έχετε ορίσει την ταχύτητα της μπάλας κανονιού σωστά).
Για να επηρεάσετε τα σώματα που υπάρχουν ήδη, πρέπει να καλέσετε τις συναρτήσεις AddBodyForce (), AddBodyTorque (), AddWorldForce () και AddWorldTorque (). Αυτές οι συναρτήσεις θα προσθέσουν δυνάμεις στα σώματα οι οποίες είναι κάτι σαν τα «αόρατα χέρια" που τραβούν τα αντικείμενα. Ωστόσο, οι συσσωρευμένες δυνάμεις «καθαρίζονται» μέσα στη μέθοδο PhysicsSystem.Integrate(), οπότε οτιδήποτε κάνετε στην μέθοδο Update () δεν θα έχει κανένα αποτέλεσμα. Αντί' αυτού, θα πρέπει να προσθέσετε μια υποκατηγορία (subclass) ελεγκτή για στην μέθοδο PhysicsSystem, χρησιμοποιώντας την συνάρτηση PhysicsSystem.AddController().
Κάθε φορά που το σύστημα φυσικής κάνει ενσωμάτωση, καλεί τους ελεγκτές, και οι ελεγκτές μπορούν να προσθέσουν δυνάμεις και ροπές στα διάφορα αντικείμενα. Για παράδειγμα, για να κάνει ένα αντικείμενο να πετάξει προς τα πάνω, μπορείτε να καλέσετε την AddWorldForce(Vector3.Up * * body.Mass 11) για το σώμα. Αρχική μάζα εξ' ορισμού είναι 10 (newtons ανά κιλό, κοντά στη βαρύτητα της Γης), οπότε αυτή θα προσθέσει αρκετή δύναμη για να πράξει στην βαρύτητα του αντικειμένου και σιγά-σιγά να το αναγκάσει να επιταχυνθεί προς τα πάνω. Οι ροπές είναι σαν τις δυνάμεις, με τη διαφορά ότι προκαλούν το αντικείμενο να γυρίσει γύρω από το κέντρο βάρους του, αντί να μετακινείται (translate) στον κόσμο.

Άλλα tips:
Οι μονάδες που χρησιμοποιούνται είναι: newtons, μέτρα, Newton-μέτρα, χιλιόγραμμα και δευτερόλεπτα, γνωστό και ως Διεθνές Σύστημα SI (System International) ή μονάδες του μετρικού συστήματος. Ίσως να είστε σε θέση να κάνετε αλλάξετε κλίμακα (scale) στις εν λόγω μονάδες για διαφορετικά συστήματα μέτρησης, αλλά είναι πιο εύκολο να μετατρέψετε απλά τα πάντα σε μετρικά.
Αν θέλετε ένα σώμα να είναι στον κόσμο, αλλά να μην κινείται, ρυθμίστε την ιδιότητα Immovable να είναι ίση με αλήθεια (true). Αυτό είναι πολύ χρήσιμο για το έδαφος, διότι, διαφορετικά, η βαρύτητα θα το τραβήξει προς τα κάτω μαζί με τα υπόλοιπα αντικείμενα.
Το βήμα του χρόνου που χρησιμοποιείτε πρέπει να είναι μικρό για να πάρει καλά αποτελέσματα της προσομοίωσης. Μια τιμή περίπου 0,01 θα ήταν σωστή. Πολύ μικρό, και η προσομοίωση θα απαιτήσει πολύ μεγάλο υπολογισμό και θα τρέξει αργά. Πολύ μεγάλο, και η προσομοίωση θα γίνει λιγότερο σταθερή. Σημειώστε ότι ο χρόνος μεταξύ των πλαισίων (frames) στα 60 Hz είναι περίπου 0.017, που είναι κάπως υψηλότερη από το ιδανικό, αλλά συνήθως λειτουργεί καλά.
Κοιτάξτε το αντικείμενο του αυτοκινήτου για να δείτε πώς ένα πιο σύνθετο αντικείμενο είναι συναρμολογημένο. Σημειώστε ότι αυτό το αντικείμενο δεν χρησιμοποιεί σφαίρες για τους τροχούς. Αντί να χρησιμοποιεί έναν αριθμό από ακτίνες, που τις ρίχνει προς την κατεύθυνση των τροχών, για την ανίχνευση του εδάφους.

Διευκρινήσεις:
Primitives: είναι βασικά στοιχεία όπως κουτιά, κουτιά με στρογγυλές άκρες, σφαίρες, heightmaps, planes, κουτιά με bounding box και περιστροφή.
Collision skin: είναι εάν γκρουπ από απλά τρισδιάστατα primitives που χτίζουν το σχήμα του σώματος.
Plane: αντιπροσωπεύει μια επίπεδη επιφάνεια στον χώρο.
Heightmap : αντιπροσωπεύει μια ανώμαλη επιφάνεια στο χώρο.
Trianglemesh : αντιπροσωπεύει ένα σύνολο από τρίγωνα (πολύγωνα). Συνήθως είναι γνωστό ως 3D Model.

5.2.2 Περίπτωση Σύγκρουσης (collision event)

Το κομμάτι αυτό έχει να κάνει με τον χειρισμό του παιχνιδιού σε περίπτωση που δύο αντικείμενα συγκρουστούν μεταξύ τους. Το JigLibX λοιπόν, προσομοιώνει φυσική συμπεριφορά, αλλά για ένα παιχνίδι ή κάποια εφαρμογή, θα χρειαστεί να κάνετε κάτι, αν για παράδειγμα ένα διαστημόπλοιο συγκρουστεί με μια τεράστια εξωγήινη κατσαρίδα ή δύο αυτοκίνητα συγκρουστούν μεταξύ τους. Υπάρχουν τουλάχιστον δύο τρόποι για τον χειρισμό των συγκρούσεων.

- ***Χρησιμοποιούμε συναρτήσεις επανάκλησης (callback functions):***

Χρησιμοποιούμε αυτή τη μέθοδο, αν θέλουμε να ρυθμίσουμε τη συμπεριφορά σύγκρουσης απλών ή πολλαπλών αντικειμένων. Πρώτον, δημιουργείται μια συνάρτηση η οποία εφαρμόζεται σε ένα ή περισσότερα αντικείμενα και εκτελείται κάθε φορά που συμβαίνει σύγκρουση.

- **Δημιουργία συναρτήσεων επανάκλησης:**
Αυτή η συνάρτηση πρέπει να επιστρέψει μια τιμή Boolean (true ή false). Αν επιστρέψετε αλήθεια, η μηχανή φυσικής θα επεξεργάζεται επερχόμενη σύγκρουση, αλλιώς θα αγνοείται η σύγκρουση αυτή, θα είναι δηλαδή κάτι σαν φάντασμα, στο οποίο το ένα αντικείμενο θα περνάει μέσα από το άλλο. Χρειάζεται δύο παραμέτρους, οι οποίες θα δοθούν από την επανάκληση. Η πρώτη παράμετρος είναι το αντικείμενό σας (ο ιδιοκτήτης) και η δεύτερη είναι το αντικείμενο με το οποίο συγκρούεται (ο συγκρουόμενος). Μπορείτε να καθορίσετε τη συμπεριφορά για ειδικά αντικείμενα ή κλάσεις αντικειμένων.

- **Εφαρμογή της συνάρτησης επανάκλησης στο αντικείμενο:**
Τώρα μπορείτε να χρησιμοποιήσετε αυτή τη συνάρτηση για κάθε αντικείμενο που χρειάζεστε.

ο ***Χρησιμοποιούμε τα CollisionInfos:***

Εδώ, καθορίζουμε μία λίστα από αντικείμενα για να ελέγξουν για σύγκρουση. Χρησιμοποιώντας την συνάρτηση callback μας δίνει ένα γεγονός, αλλά υπάρχει έλλειψη πληροφοριών όπως το πόσο μέσα στο αντικείμενο είναι το άλλο αντικείμενο. Εδώ μπορούμε να κάνουμε πολλά περισσότερα από ότι στο προηγούμενο παράδειγμα.

- **Προετοιμασία**
Χρειαζόμαστε έναν CollisionFunctor, ο οποίος κάνει έλεγχο για ζεύγη στα οποία συμβαίνει σύγκρουση (αυτό δοκιμάζει για όλα τα δυνατά ζεύγη μόνο μία φορά), για μια λίστα από bodies, που πρέπει να ελέγχονται και μια άλλη λίστα για την αποθήκευση ενός CollisionInfo για κάθε σύγκρουση.
- **Ανίχνευση συγκρούσεων**
Γίνεται ανίχνευση συγκρούσεων σε μια ρουτίνα ενημέρωσης (update routine), για παράδειγμα στην ανανέωση του κόσμου.

5.2.3 Αρχεία που χρειάζεται να εισαχθούν

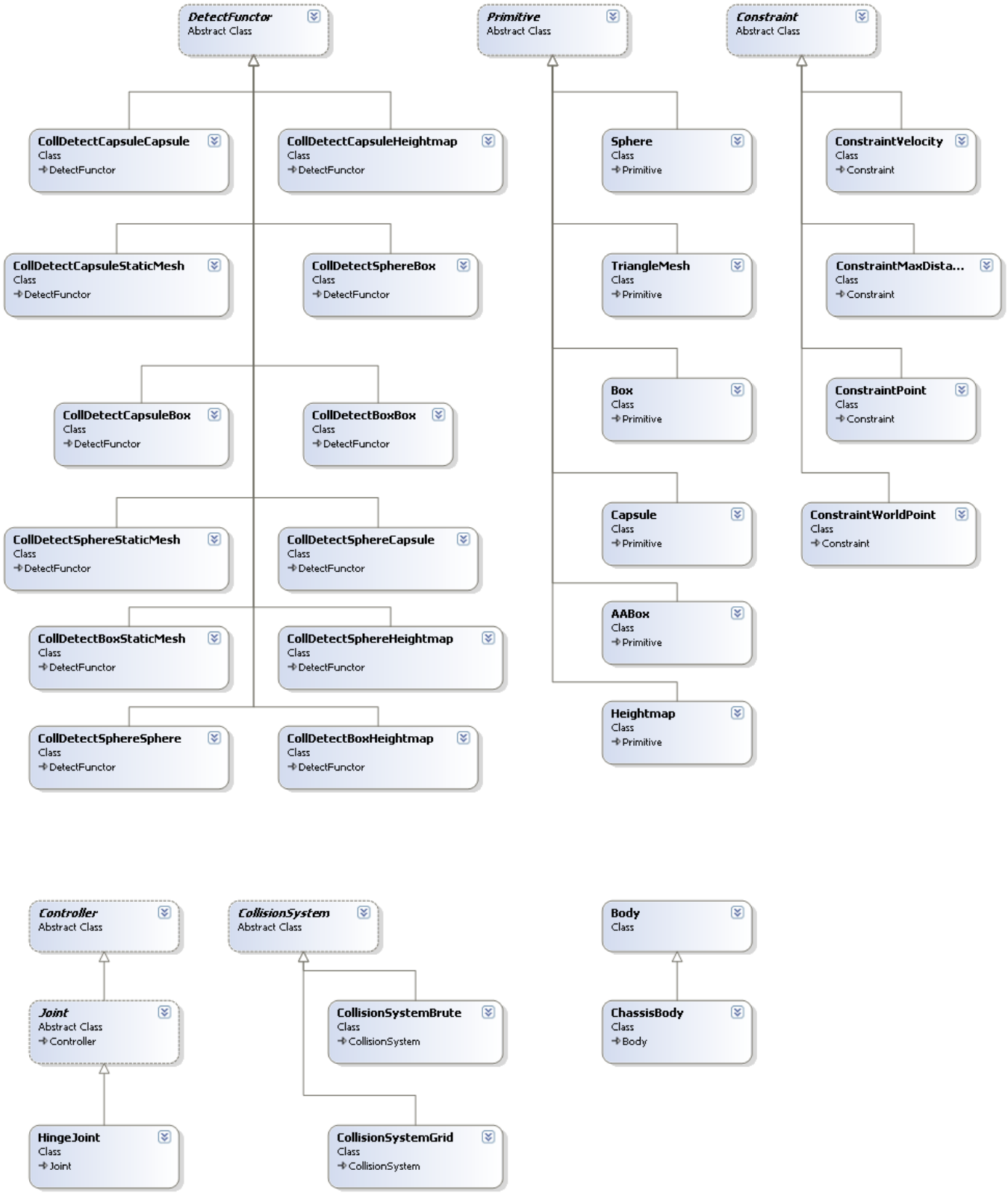
Για να χρησιμοποιήσουμε την μηχανή φυσικής JigLibX και να την εφαρμόσουμε στον κόσμο μας, πρέπει να εισάγουμε τα αρχεία που αντιστοιχούν σε κάθε ένα από τα αντικείμενα που χρησιμοποιούμε στο παιχνίδι, έτσι ώστε να μπορούμε να κάνουμε τις απαραίτητες παρεμβάσεις για το αποτέλεσμα που θέλουμε. Στον παρακάτω πίνακα φαίνονται τα αρχεία που χρειάζεται να εισαχθούν.

Filename	Status
Collision\CollisionDetection\BoxBox.cs	ported
Collision\CollisionDetection\BoxHeightmap.cs	ported
Collision\CollisionDetection\BoxPlane.cs	ported
Collision\CollisionDetection\BoxStaticMesh.cs	ported
Collision\CollisionDetection\CapsuleBox.cs	ported
Collision\CollisionDetection\CapsuleCapsule.cs	ported
Collision\CollisionDetection\CapsuleHeightmap.cs	ported
Collision\CollisionDetection\CapsulePlane.cs	ported
Collision\CollisionDetection\CapsuleStaticMesh.cs	ported
Collision\CollisionDetection\DetectFunctor.cs	ported
Collision\CollisionDetection\SphereBox.cs	ported

Collision\CollisionDetection\SphereCapsule.cs	ported
Collision\CollisionDetection\SphereHeightmap.cs	ported
Collision\CollisionDetection\SpherePlane.cs	ported
Collision\CollisionDetection\SphereSphere.cs	ported
Collision\CollisionDetection\SphereStaticMesh.cs	ported
Collision\CollisionInfo.cs	ported
Collision\CollisionSkin.cs	ported
Collision\CollisionSystem.cs	ported
Collision\CollisionSystemBrute.cs	ported
Collision\CollisionSystemGrid.cs	ported
Collision\Materials.cs	ported
Geometry\Primitives\Box.cs	ported
Geometry\Primitives\Capsule.cs	ported
Geometry\Primitives\Heightmap.cs	ported
Geometry\Primitives\Plane.cs	ported
Geometry\Primitives\Primitive.cs	ported
Geometry\Primitives\Sphere.cs	ported
Geometry\Primitives\TriangleMesh.cs	ported
Geometry\AABox.cs	ported
Geometry\Distance.cs	ported
Geometry\IndexedTriangle.cs	ported
Geometry\Intersection.cs	ported
Geometry\Line.cs	ported
Geometry\Octree.cs	ported
Geometry\IOverlap.cs	ported
Geometry\Rectangle.cs	ported
Geometry\Triangle.cs	ported
Math\Transform.cs	ported
Physics\Body.cs	ported
Physics\Constraint.cs	ported

Physics\ConstraintMaxDistance.cs	ported
Physics\ConstraintPoint.cs	ported
Physics\ConstraintVelocity.cs	ported
Physics\ConstraintWorldPoint.cs	ported
Physics\Controller.cs	ported
Physics\HingeJoint.cs	ported
Physics\Joint.cs	ported
Physics\PhysicsCollision.cs	ported
Physics\PhysicsSystem.cs	ported
Vehicles\Car.cs	ported
Vehicles\Chassis.cs	ported
Vehicles\Wheel.cs	ported

Το διάγραμμα των κλάσεων που χρησιμοποιεί το JigLibX φαίνεται στην εικόνα 25.



Εικ. 25 JigLibX Class Diagram

5.3 Άλλες μηχανές φυσικής

Κάποιες ακόμα μηχανές φυσικής για τα μοντέλα του Xna, προτεινόμενες από το Xna Creators Club είναι οι παρακάτω:

- Farseer – μόνο για δύο διαστάσεων μοντέλα.
- Bullet
    - BulletX: Από το XnaDevRu: Υποστηρίζει τόσο το Xbox360 όσο και τα Windows.
    - XBAP: Από τον Chris Cavanagh, βασισμένο στο BulletX από πάνω.
- Newton
- Oops! 3D Physics Framework
- Bepu physics – για εμπορική χρήση αλλά δωρεάν για μη εμπορική.
- Jello Physics – για παιχνίδια δύο διαστάσεων.
- Physics2D.Net – για παιχνίδια δύο διαστάσεων.

Αν στοχεύετε μόνο στην πλατφόρμα των Windows, τότε μπορείτε να χρησιμοποιήσετε οποιαδήποτε από τις εμπορικές μηχανές φυσικής ή μηχανές ανοιχτού κώδικα (open source). Μερικές έχουν εύκολη δομή, ή ίσως χρειαστεί να την αλλάξετε εσείς.

- PhysX
- ODE (Open Dynamics Engine)
- Newton Game Dynamics
    - Newton physics: Δημιουργήθηκε μέσα σε 8 με 10 ώρες και η ποιότητάς της το αντανακλά.

Αναφορά:

JigLibX official homepage (http://jiglibx.codeplex.com/),

JigLibX wiki page (http://jiglibx.wikidot.com/),

Wikipedia (http://en.wikipedia.org/),

YouTube video tutorials (www.youtube.com),

Resources for Xna Game Developers (www.xnaresources.com),

Xna Creator's Club Online (http://creators.xna.com),

Alexandre Santos Lobao, Bruno Evangelista, Jose Antonio Leal de Farias and Riemer Grootjans "Xna 3.0 Game Programming, From Novice to Professional" (Apress).

6

Αλληλεπίδραση με το χρήστη

Είναι πολύ διασκεδαστικό και όμορφο να βλέπουμε έναν κύβο που δημιουργήσαμε εμείς να εμφανίζεται στην οθόνη. Υπάρχουν πολλά και πιο ενδιαφέροντα πράγματα που μπορούμε να κάνουμε ακόμα με το Xna. Παρ' όλο που ο κύβος εμφανίστηκε στην σκηνή και δείχνει όμορφος, δεν κάνει τίποτα, είναι στάσιμος, που σημαίνει πως πρέπει να τον χειριστούμε εμείς οι ίδιοι. Άλλωστε πόσο διασκεδαστικό είναι ένα παιχνίδι αν δεν υπάρχει αλληλεπίδραση με τον χρήστη; Σε αυτό το κεφάλαιο, θα δούμε πως ένας χρήστης μπορεί να έχει το χειρισμό από το πληκτρολόγιο και από το ποντίκι. Έτσι το παιχνίδι αυτό θα κάνει κάτι επιπλέον από το να δείχνει καλό και όμορφο.

6.1 Είσοδος από το πληκτρολόγιο

Η είσοδος από το πληκτρολόγιο γίνεται μέσα από την κλάση Keyboard, η οποία βρίσκεται μέσα στο Microsoft.XNA.Framework.Input namespace. Η κλάση αυτή έχει μια στατική μέθοδο που ονομάζεται GetState που αποκαθιστά την τρέχουσα κατάσταση του πληκτρολογίου μέσα σε μια KeyboardState δομή.
Η KeyboardState δομή περιέχει τρεις μεθόδους πλήκτρου που μας δίνει την λειτουργικότητα που χρειαζόμαστε. Αυτές είναι:

- `Keys [] GetPressedKeys()`

Η οποία επιστρέφει έναν πίνακα από πλήκτρα που έχουν πατηθεί την στιγμή που καλείται η μέθοδος.

- `bool IsKeyDown (Keys key)`

Η μέθοδος αυτή είναι τύπου boolean που σημαίνει ότι δέχεται σαν τιμές, αλήθεια ή ψέματα (true, false). Αυτό εξαρτάται από το αν το πλήκτρο αντιπροσωπεύεται από την παράμετρο IsDown, αν δηλαδή είναι πατημένο την στιγμή που καλείται η μέθοδος.

- `bool IsKeyUp (Keys key)`

Αυτή η μέθοδος επιστρέφει επίσης αλήθεια η ψέματα. Αυτό εξαρτάται από το αν το πλήκτρο αντιπροσωπεύεται από την παράμετρο IsUp, αν αφέθηκε δηλαδή την στιγμή που καλείται η μέθοδος.
Σαν παράδειγμα της χρήσης του πληκτρολογίου, αν θέλετε να ελέγξετε αν το πλήκτρο 'A' είναι πατημένο, θα δείτε τον παρακάτω κώδικα:

```
if (Keyboard.GetState().IsKeyDown(Keys.A))
{
    //do something
}
```

6.2 Είσοδος από το ποντίκι

Το Xna παρέχει την κλάση Mouse για αλληλεπίδραση με το ποντίκι, η οποία συμπεριφέρεται παρόμοια με την κλαση του πληκτρολογίου. Η κλάση που χρησιμοποιείται για το ποντίκι, περιέχει και αυτή μια μέθοδο που ονομάζεται GetState, η οποία χρησιμοποιείται για να πάρει δεδομένα από το πληκτρολόγιο μέσα σε μια MouseState δομή. Δέχεται δηλαδή την τρέχουσα κατάσταση του ποντικιού. Η κλάση Mouse έχει ακόμα μια μέθοδο άξια σημείωσης. Αυτή είναι: `void SetPosition(int x, int y).` Αυτή η μέθοδος επιτρέπει στους χρήστες να ορίσουν την θέση του ποντικιού. Αυτή η θέση είναι σχετική με την πάνω αριστερή γωνία του παραθύρου του παιχνιδιού.
Η δομή MouseState έχει αρκετές ιδιότητες που μας βοηθάνε να καταλάβουμε τι συμβαίνει με το ποντίκι κάθε στιγμή όταν καλούμε την μέθοδο GetState. Αυτές οι ιδιότητες είναι:

- Η **LeftButton**, η οποία είναι τύπου ButtonState και επιστρέφει την κατάσταση του αριστερού πλήκτρου του ποντικιού.

- Η **MiddleButton**, είναι κι αυτή τύπου ButtonState και επιστρέφει την κατάσταση του μεσαίου πλήκτρου του ποντικιού.

- Η **RightButton**, είναι τύπου ButtonState και επιστρέφει την κατάσταση του δεξιού πλήκτρου του ποντικιού.

- Η **ScrollWheelValue**, είναι τύπου ακεραίου (int). Επιστρέφει την συνολική συσσωρευμένη κίνηση της ρόδας του ποντικιού (scroll wheel) από την στιγμή που ξεκίνησε το παιχνίδι. Για να μάθουμε πόσο κινήθηκε η ρόδα, συγκρίνουμε αυτήν την τιμή με τον ρυθμό της προγούμενης τιμής που είχε η ρόδα.

- **X**, είναι τύπου ακεραίου και επιστρέφει την τιμή την οριζόντιας θέσης του ποντικιού σε σχέση με την πάνω αριστερή γωνία του παραθύρου του παιχνιδιού. Αν το ποντίκι βρίσκεται στην αριστερή πλευρά του παραθύρου, η τιμή αυτή είναι αρνητική. Αν το ποντίκι βρίσκεται στην δεξια πλευρά του παραθύρου, η τιμή αυτή είναι μεγαλύτερη από το πλάτος του παραθύρου.

- **XButton1** και **XButton2**. Αυτές οι δύο ιδιότητες, χρησιμοποιούνται σε περίπτωση που υπάρχουν επιπρόσθετα πλήκτρα και είναι τύπου ButtonState.

- **Y**, είναι τύπου ακεραίου και επιστρέφει την τιμή της κάθετης θέσης του ποντικιού σε σχέση με την πάνω γωνία του παραθύρου του παιχνιδιού. Αν το ποντίκι είναι πάνω από το παράθυρο, η τιμή είναι αρνητική. Αν το ποντίκι βρίσκεται κάτω από το παράθυρο του παιχνιδιού, η τιμή αυτή είναι μεγαλύτερη από το ύψος του παραθύρου.

Ίσως προσέξατε ότι εξ' ορισμού ο κέρσορας του ποντικιού είναι ορισμενος να είναι κρυφός όταν το ποντίκι περνάει πάνω από ένα παράθυρο παιχνιδιού του Xna. Αν θέλετε να φαίνεται ο κέρσορας στο Xna παράθυρο μπορείτε να το κάνετε θέτοντας την ιδιότητα IsMouseVisible ίση με true.
Ανεξάρτητα από το αν ο κέρσορας είναι ορισμένος να φαίνεται ή όχι, η δομή MouseState που επιστρέφεται από την κλήση της GetState μεθόδου, πάντα θα κρατάει την τρέχουσα κατάσταση του ποντικιού.

Αναφορά:

Alexandre Santos Lobao, Bruno Evangelista, Jose Antonio Leal de Farias and Riemer Grootjans
"Xna 3.0 Game Programming, From Novice to Professional" (Apress),

Aaron Reed “O’ Reilly Learning Xna 3.0” .

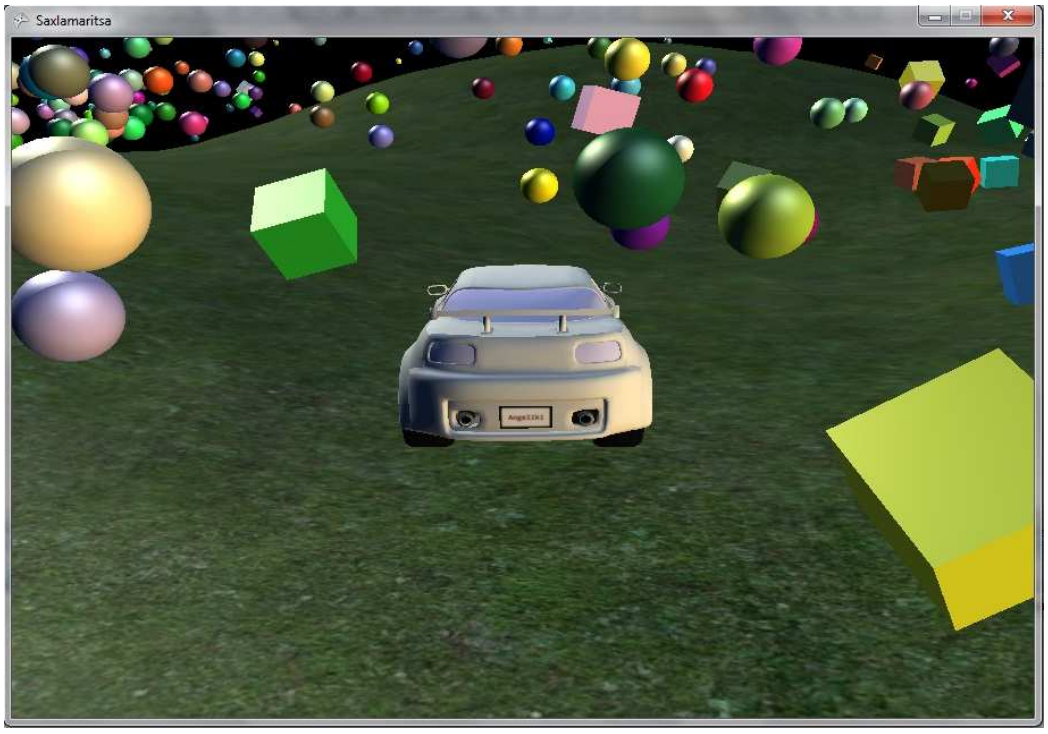
7

Υλοποίηση

Σε αυτή την ενότητα, θα εξηγηθούν ενδεικτικά κομμάτια κώδικα, ο ρόλος των μεταβλητών και ο τρόπος που επιδρούν οι διάφορες λειτουργίες στον αλγόριθμο που υλοποιεί την εφαρμογή. Θα μιλήσουμε περισσότερο για φυσική συμπεριφορά των μοντέλων και κίνηση στην τρισδιάστατη σκηνή.

7.1 Περιγραφή της προγραμματιστικής προσέγγισης

Η εφαρμογή αυτή, βασίστηκε κυρίως στη φυσική συμπεριφορά που έχουν τα τρισδιάστατα αντικείμενα μεταξύ τους, όταν συγκρουστούν, όταν πέσουν από μεγάλο ύψος, όταν κινούνται σε ένα ανώμαλο έδαφος κ.λ.π., καθώς επίσης και στην λειτουργία της κάμερας. Τα κυρίως αντικείμενα από τα οποία αποτελείται αυτή η εφρμογή (το αυτοκίνητο, τα κουτιά, οι σφαίρες, το heightmap), έχουν αρκετές ομοιότητες στη φυσική τους όπως και στη συμπεριφορά τους γενικότερα. Από παρόμοια χαρακτηριστικά, όπως η θέση τους στον τρισδιάστατο κόσμο (X,Y,Z), μέχρι σε κοινές ιδιότητες, όπως η αναπαράστασή τους στη σκηνή ή ο έλεγχος συγκρούσεων. Έτσι κρίθηκε αναγκαία η ομαδοποίησή τους και πιο συγκεκριμένα η δημιουργλθα βασικών κλάσεων στις οποίες θα υπάρχουν όλα αυτά τα κοινά χαρακτηριστικά και οι λειτουργίες.
Τα αντικείμενα που χρησιμοποιήθηκαν στην σκηνή, ήταν ένα αυτοκίνητο, ένας κύβος, μια σφαίρα, ένα heightmap και ένα κομμάτι μουσικής, όπως φαίνεται και στην εικόνα 26.



Εικ. 26 Αντικείμενα της εφαρμογής.

Το αυτοκίνητο ξεκινάει από ένα ορισμένο ύψωμα στην σκηνή του κόσμου και ανάλογα με την μάζα του πέφτει φυσικά στο έδαφος. Ο χρήστης έχει τη δυνατότητα να μετακινεί το αυτοκίνητο στον τρισδιάστατο κόσμο, χωρίς αυτό να περνάει κάτω από το έδαφος, ή μέσα από τα κουτιά και τις σφαίρες με τα οποία ενδεχομένως να συγκρουστεί. Όταν ο χρήστης μετακινήσει το αυτικίνητο εκτός ορίων αυτό πέφτει στο κενό. Τα κουτιά, ορισμένα και αυτά να ξεκινούν από ένα συγκεκριμένο ύψος που δώθηκε μέσω κώδικα, αναπαράγονται μέχρι έναν αριθμό και πέφτουν φυσικά στο έδαφος. Τέλος, το ίδιο συμβαίνει και με τις σφαίρες, οι οποίες αναπαράγονται συνεχώς,κάθε φορά που ανανεώνεται και η σκηνή. Η κάμερα του κόσμου, είναι κάμερα τρίτου προσώπου και ακολουθεί την κίνηση του αυτοκινήτου. Ο ήχος, αναπαράγεται στην αρχή που εκτελείται η εφαρμογή.
Στη συνέχεια θα δούμε μέσα από ενδεικτικά παραδείγματα κώδικα κάθε κλάσης, όπως για παράδειγμα, πώς ρυθμίστηκε η κάμερα, πώς δουλεύει το αντικείμενο του αυτοκινήτου, του κύβου και της σφαίρας, πώς χρησιμοποιήσαμε το heightmap, κ.ά.
Σημειώστε ότι, για την υλοποίηση αυτής της εφαρμογής, χρειάστηκε εκτός από την εισαγωγή των μοντέλων μας στο ContentPipeline, η εισαγωγή κάποιων References (αναφορές) που χρησιμοποιήθηκαν για τα ολοκληρωθεί ο κόσμος. Αυτά ήταν, η βιβλιοθήκη του JigLibX για την εφαρμογή της φυσικής συμπεριφοράς στα μοντέλα, που έγινε από τα References του Solution Explorer και το dll αρχείο για το Generated Geometry, που χρησιμοποιήθηκε για την υλοποίηση του Heightmap, που έγινε από τα References του ContentPipeline. Θα μιλήσουμε πιο αναλυτικά για αυτά πιο κάτω.
Να θυμάστε ότι σε κάθε κλάση που δημιουργούμε και η οποία χρησιμοποιεί την μηχανή φυσικής πρέπει να προσθέτουμε και να αντίστοιχα namespaces.

7.2 Η κλάση Game1

Στο κεφάλαιο 2, μιλήσαμε αναλυτικά για τη βασική δομή μιας εφαρμογής. Δημιουργώντας ένα απλό Xna project , η Visual C# τοποθετεί όλον τον κώδικα του project σε μια κλάση (class) με το όνομα Game1. Οι εφαρμογές και τα παιχνίδια που δημιουργούνται είναι συνεπώς ένα αντικείμενο (object) τύπου Game1.
Με τη δημιουργία αυτής της κλάσης, αυτόματα δημιουργούνται και κάποιες μέθοδοι οι οποίες μας βοηθούν στην ανάπτυξη της εφαρμογής, (Initialize, LoadContent, Update, Draw) και εξηγήθηκαν αναλυτικά στο κεφάλαιο 2 αυτής της εργασίας. Εκτός από αυτές τις μεθόδους, δημιουργήθηκαν και κάποιες ακόμα που μας βοήθησαν στην επίτευξη του στόχου μας, τις οποίες θα δούμε παρακάτω.
Η Game1 κλάση λοιπόν, είναι η βασική κλάση του τρισδιάστατου κόσμου μας. Μέσα σε αυτή, γίνεται η δήλωση των μοντέλων και των ήχων, η αρχικοποίηση μεταβλητών που σχετίζονται με άλλες κλάσεις, η εισαγωγή της φυσικής μηχανής που εφαρμόζεται στα μοντέλα μας, ο χειρισμός του παιχνιδιού από το χρήστη, η κλήση άλλων κλάσεων κ.λπ.
Στην βασική λοιπόν κλάση και έξω από τον Constructor δηλώσαμε όλες τις μεταβλητές που χρησιμοποιήσαμε. Αυτές ήταν:

- Τα μοντέλα του κόσμου
- Οι ήχοι που χρησιμοποιήθηκαν
- Η μηχανή φυσικής
- Η κάμερα
- Ένα αντικείμενο τύπου CarObject που χρησιμοποιήθηκε για το μοντέλου του αυτοκινήτου
- Αντικείμενα τύπου Sphere και τύπου CrashBox

7.2.1 Η εισαγωγή της μηχανής φυσικής

Η μηχανή φυσικής είναι αυτή που ελέγχει ότι σχετίζεται με την φυσική σε κάθε προσομοίωση. Οτιδήποτε χρησιμοποιεί το JigLibX απαιτεί και ένα σύστημα φυσικής. Για να δημιουργήσουμε ένα δεν είναι τόσο δύσκολο. Αρχικά πρέπει να προσθέσουμε τα εξής namespaces στην αρχή του κώδικα σε κάθε κλάση που χρησιμοποιεί το σύστημα φυσικής. Προσθέτουμε λοιπόν τα εξής namespaces:

```
using JigLibX.Math;
using JigLibX.Physics;
using JigLibX.Geometry;
using JigLibX.Collision;
```

Όπως καταλαβαίνετε, τα namespaces αυτά έχουν να κάνουν με τα μαθηματικά και τη φυσική της μηχανής φυσικής που εφαρμόζεται στα μοντέλα, όπως και με τη γεωμετρία τους και τα collision.
Στην βασική κλάση Game1, στην αρχή του κώδικα και έξω από τον constructor προσθέτουμε τον παρακάτω κώδικα:

```
PhysicsSystem world = new PhysicsSystem();
```

Και έτσι έχουμε το δικό μας σύστημα φυσικής. Παρά το γεγονός ότι η οργάνωση δεν έχει ολοκληρωθεί. Αν επρόκειτο να συνεχίσουμε από εδώ και αφήνοντας το σύστημα φυσικής ως έχει, τα αντικείμενά μας απλώς θα περάσουν το ένα μέσα από το άλλο. Συνεπώς, πρέπει να προσθέσουμε ένα σύστημα σύγκρουσης (collision system) στο σύστημα φυσικής μας. Προσθέστε τον ακόλουθο κώδικα κάτω από αυτόν που μόλις προσθέσατε:

```
physicSystem.CollisionSystem = new CollisionSystemSAP();
```

Σε αυτή τη γραμμή κώδικα καθορίζεται ποιο σύστημα σύγκρουσης θα χρησιμοποιηθεί.

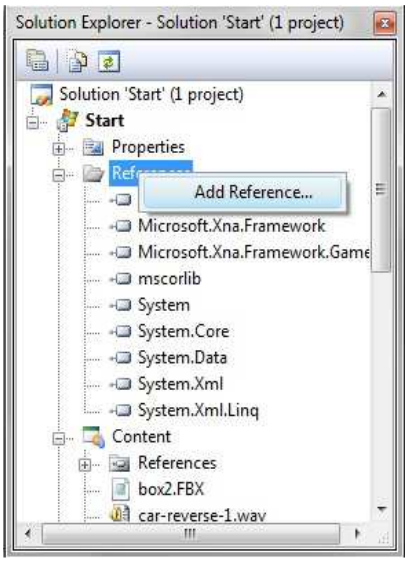
Υπάρχει ένα τελευταίο πράγμα που πρέπει να κάνουμε πριν προχωρήσουμε περισσότερο από τη δημιουργία του συστήματος φυσικής. Θα πρέπει να ενημερώνουμε τον κόσμο κάθε τόσο. Η ενημέρωση αυτή ονομάζεται "ενσωμάτωση" (integrating) στην ορολογία της φυσικής. Όταν ένα σύστημα φυσικής ενημερώνεται, ουσιαστικά εφαρμόζει βαρύτητα σε όλα τα αντικείμενα μας και εντοπίζει περιπτώσεις σύγκρουσης μεταξύ τους. Στην πραγματικότητα είναι πολύ πιο περίπλοκο από αυτό, αλλά οι λεπτομέρειες είναι ασήμαντες σε αυτό το σημείο.
Η ενημέρωση του συστήματος φυσικής είναι εύκολη από την πλευρά μας. Κάνουμε απλώς μια κλήση στη μέθοδο Integrate, του συστήματος φυσικής. Παίρνει μια παράμετρο προσδιορίζοντας πόσος χρόνος έχει περάσει από την τελευταία κλήση της ενσωμάτωσης. Χρησιμοποιώντας τη μέθοδος Integrate είναι τόσο εύκολο όσο και το να προσθέσετε την ακόλουθη γραμμή κώδικα στην μέθοδο Update της βασικής κλάσης του παιχνιδιού, Game1:

```
float timeStep = (float)gameTime.ElapsedGameTime.Ticks /
TimeSpan.TicksPerSecond;
    PhysicsSystem.CurrentPhysicsSystem.Integrate(timeStep);
```

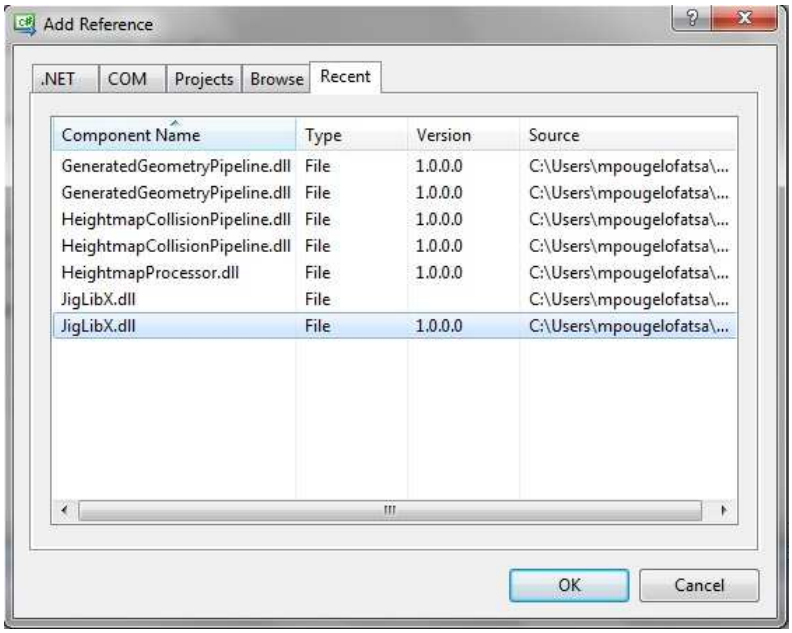
Το PhysicsSystem.CurrentPhysicsSystem είναι ένα στατικό μέλος της κλάσης PhysicsSystem που έχει οριστεί για το τελευταίο σύστημα φυσικής που δημιουργήθηκε. Αυτό είναι το σύστημα φυσικής που ορίσαμε προηγουμένως στην βασική κλάση Game1. Χρησιμοποιούμε αυτό το στατικό μέλος για λόγους ευκολίας και έτσι δεν χρειάζεται να περάσουμε μια μεταβλητή PhysicsSystem σε οποιαδήποτε συνάρτηση το χρειάζεται.
Για την παράμετρο της μεθόδου Integrate χρησιμοποιούμε την παράμετρο gameTime που περνάμε στην μέθοδο Update.

Το σύστημα φυσικής είναι πλέον πλήρης και έτοιμο για την προσομοίωση ενός βασικού κόσμου. Σε αυτό το σημείο όμως ο κόσμος μας είναι μάλλον γυμνός. Δεν υπάρχει τίποτα σε αυτόν ακόμα. Για να κατοικηθεί ο κόσμος με κουτιά και σφαίρες και οτιδήποτε άλλο θέλουμε για να τον εμπλουτίσουμε, θα δημιουργήσουμε μια κλάση για κάθε αντικείμενο που θα δέχεται το σύστημα φυσικής και τις παραμέτρους που χρειάζονται για να ολοκληρωθούν και να λειτουργήσουν σωστά στη σκηνή του κόσμου. Θα μιλήσουμε για αυτές τις κλάσεις αναλυτικότερα παρακάτω.

Όπως είπαμε και παραπάνω πρέπει να κάνουμε και εισαγωγή του JigLibX από τα References του βασικού Project. Πηγαίνοντας λοιπόν στον Solution Explorer, με δεξί κλικ μέσα στο βασικό Project (στην περίπτωσή μας το Start), στο References → Add Reference. Εκεί μας εμφανίζεται μια νέα καρτέλα που περιλαμβάνει τις βιβλιοθήκες που μας επιτρέπει το Xna να εισάγουμε. Στην τελευταία καρτέλα, όπως φαίνεται και στις εικόνες 27 και 28, επιλέγουμε το JigLibX και πατάμε οκ. Διαφορετικά, κάνουμε browse, το βρίσκουμε από κει που είναι αποθηκευμένο και πατάμε οκ.



Εικ. 27 Εισαγωγή αναφοράς για το JigLibX



Εικ. 28 Εισαγωγή αναφοράς JigLibX

7.2.2 Δημιουργία αντικειμένων

Αφού δημιουργήσουμε τις κλάσεις που αντιστοιχούν στα αντικείμενα του κόσμου μας, όπως θα δούμε αναλυτικότερα παρακάτω, στην βασική κλάση του κόσμου μας δημιουργούμε τα αντικείμενα. Το πρώτο βήμα είναι η δήλωσή τους στην κλάση Game1, στην αρχή του κώδικα και έξω από τον constructor.
Προσθέτουμε λοιπόν τον εξής κώδικα:

```
   Model box, sphere, car, wheel,  hm;
   Sphere sph;
CrashBox bx;
CarObject carObject;
```

Με αυτόν τον τρόπο δηλώνουμε τα μοντέλα μας (box, sphere, car, wheel, hm). Αυτά τα μοντέλα αργότερα θα αρχικοποιηθούν στη μέθοδο LoadContent και θα πάρουν τα μοντέλα που έχουμε εισάγει προηγουμένως στο ContetPipeline.
Τα Sphere, CrashBox και CarObject δηλώνουν τρία αντικείμενα που είναι τύπου Sphere, CrashBox και CarObject αντίστοιχα. Αυτές οι δύο είναι κλάσεις (τις οποίες θα δούμε αναλυτικότερα πιο κάτω), οι οποίες περνάνε τις μεταβλητές και τα χαρακτηριστικά τους στα μοντέλα κατά τη δημιουργία τους.
Στην μέθοδο LoadContent φορτώνουμε τα μοντέλα μας στις μεταβλητές που μόλις φτιάξαμε ως εξής:

```
box = Content.Load<Model>("box2");
      sphere = Content.Load<Model>("sphere");
      car = Content.Load<Model>("car");
      wheel = Content.Load<Model>("wheel");
   hm = Content.Load<Model>("haytemap");
```

Κάθε μεταβλητή τώρα περιέχει μέσα της τα μοντέλα που είχαν εισαχθεί νωρίτερα από το ContentPipeline και είχαν αναγνωριστεί επιτυχώς από αυτό.

Για τη δημιουργία των αντικειμένων προσθέτουμε στην ίδια μέθοδο τον παρακάτω κώδικα:

```
        sph = new Sphere(this, sphere, 1f, Matrix.Identity, Vector3.Up * 100f);
                Components.Add(sph);

carObject = new CarObject(this, car, wheel, true, true, 30.0f, 5.0f, 4.7f,
5.0f, 0.20f, 0.4f, 0.05f, 0.45f, 0.3f, 1, 520.0f,
physicSystem.Gravity.Length());
carObject.Car.Chassis.Body.MoveTo(new Vector3(-20, 8, -40), Matrix.Identity);
   carObject.Car.EnableCar();

bx = new CrashBox(this, box, new Vector3(1, 1, 1), Matrix.Identity, new
Vector3(12, 0, -3));
   Components.Add(bx);

HeightmapObject heightmapObj = new HeightmapObject(this, hm, Vector2.Zero);
   this.Components.Add(heightmapObj);
```

Οι μεταβλητές που δημιουργήσαμε προηγουμένως, τώρα παίρνουν τα ορίσματα των αντικειμένων από τις κλάσεις τους από τις οποίες και έχουν οριστεί. Εμείς εδώ τους δίνουμε τις τιμές που θα έχουν όταν εμφανιστούν στη σκηνή του τρισδιάστατου κόσμου μας. Θα δούμε στις παρακάτω ενότητες τι ορίσματα παίρνει το κάθε ένα από αυτά τα αντικείμενα.
Ας δούμε όμως λίγο το αντικείμενο που δέχεται το heightmap. Όπως παρατηρούμε, η δημιουργία των μεταβλητών μπορεί να γίνει και από την LoadContent μέθοδο, όπου και τα περνάμε κατευθείαν στην μεταβλητή αυτή.
Όμως, το ότι εισάγαμε στις μεταβλητές μας τα αντικείμενα δεν σημαίνει πως μπορούμε να τα δούμε και στην σκηνή του κόσμου μας. Αν τρέξουμε το παιχνίδι, θα δούμε ότι ακόμα βλέπουμε την μπλε οθόνη, όπως στην αρχή που ξεκινήσαμε την δημιουργία της εργασίας. Αυτό συμβαίνει γιατί ακόμα δεν έχουμε ορίσει την κάμερα που θα κοιτάζει τον κόσμο μας. Θα μιλήσουμε για αυτό παρακάτω.

7.2.3 Κίνηση αυτοκινήτου

Ο χρήστης σε αυτό το σημείο έχει τη δυνατότητα να κινεί το αυτοκίνητο. Η είσοδος δίνεται από το πληκτρολόγιο. Ο χρήστης μπορεί να κινήσει το αυτοκίνητο με τα βελάκια, μπροστά, πίσω, δεξιά και αριστερά. Για να το πετύχουμε, προσθέτουμε στην μέθοδο Update τον παρακάτω κώδικα:

```
KeyboardState keyState = Keyboard.GetState();
           if (keyState.IsKeyDown(Keys.Escape))
        this.Exit();
```

Σε αυτό το σημείο δέχεται τις καταστάσεις του πληκτρολογίου. Αν το πλήκτρο Escape είναι πατημένο, τότε τερματίζει την εφαρμογή. Στην συνέχεια προσθέτουμε με τον ίδιο τρόπο άλλες τέσσερις καταστάσεις για το πληκτρολόγιο. Αν το πλήκτρο που έχει πατηθεί είναι το πάνω βελάκι, το δεξί βελάκι, το αριστερό βελάκι ή το κάτω βελάκι. Σε κάθε κατάσταση από αυτές, κινεί αναλόγως το αυτοκίνητο δίνοντάς του και μια επιτάχυνση, για παράδειγμα, στην περίπτωση που είναι πατημένο το πάνω βελάκι, τότε θα λέγαμε το εξής:

```
   carObject.Car.Accelerate = 1.0f;
```

7.2.4 Οι μέθοδοι DrawBoxes και DrawSpheres

Οι δύο αυτές μέθοδοι, δημιουργήθηκαν με σκοπό να φτιάχνουν ανάλογα με την επανάληψη που έχουμε ορίσει να έχουν, μπάλες και κουτιά. Έχοντας αυτές τις δύο μεθόδους, δεν είναι απαραίτητο να δημιουργήσουμε τα κουτιά και τις σφαίρες που φτιάξαμε προηγουμένως στην μέθοδο LoadContent.

Η μέθοδος DrawSpheres, δημιουργεί συνεχώς καινούργιες σφαίρες από μια. Και κάθε φορά από αυτή τη μια σφαίρα, βγαίνουν τρεις καινούργιες. Καλείται στην μέθοδο Draw να γεννάει συνεχώς καινούργιες μπάλες. Επομένως κάθε φορά που καλείται η μέθοδος Draw καλείται και η μέθοδος DrawSpheres.

Η μέθοδος DrawBoxes, έχει οριστεί να καλείται στην μέθοδο LoadContent, που σημαίνει ότι θα την καλέσουμε μια φορά και θα εκτελεστεί για όσο κρατάει η επανάληψη που ορίσαμε να έχει. Δημιουργεί σε μια σειρά από δέκα κουτιά, είκοσι καινούργια.
Μια μέθοδο την καλούμε ως εξής:

```
DrawSpheres();
DrawBoxes();
```

Τέλος, στην βασική κλάση καλούμε την κάμερα η οποία χρησιμοποιείται για να βλέπουμε την σκηνή του τρισδιάστατου κόσμου μας και τα αντικείμενα που τον εμπλουτίζουν.

7.3 Η κλάση PhysicsObjects

Η κλάση αυτή χρησιμοποιείται για να δώσει στα αντικείμενα που χρησιμοποιούνται στην εφαρφμογή, το body και το collision skin. Η PhsysicsObject κληρονομεί την κλάση DrawableGameComponent, για το λόγο ότι δεν περιλαμβάνει δική της εξ' ορισμού μέθοδο Draw().
Η κλάση PhysicsObject, κληρονομείται από κάθε αντικείμενο που χρησιμοποιείται στο παιχνίδι μας και χρησιμοποιεί τη μηχανή φυσικής JigLibX. Δίνει στα μοντέλα τις ιδιότητες και τα χαρακτηριστικά που περιλαμβάνει.
Δημιουργεί όπως είπαμε, ένα body και ένα collision skin που εφαρμόζει στα μοντέλα που την κληρονομούν έτσι ώστε να έχουν φυσική συμπεριφορά του αυτού τους και σε σχέση με τα άλλα αντικείμενα που συγκρούονται.
Δημιουργεί ένα μοντέλο το οποίο χρησιμοποιεί στον constructor της και του δίνει κάθε φορά ένα τυχαίο χρώμα.
Επίσης δηλώνει ένα αντικείμενο τύπου CarObject και το οποίο χρησιμοποιεί για το αυτοκίνητο του κόσμου μας. Θα μιλήσουμε για αυτό σε μια από τις επόμενες ενότητες.
Η PhysicsObject περιλαμβάνει την μέθοδο SetMass, η οποία δέχεται έναν δεκαδικό αριθμό, ο οποίο αντιστοιχεί στην μάζα του primitive, την οποία και δίνουμε όταν το δημιουργούμε από την βασική κλάση του παιχνιδιού μας την Game1. εκτός από τη μάζα, δίνει στα primitives και άλλα χαρακτηριστικά, όπως είναι η σταθερότητα και η πυκνότητα των αντικειμένων αυτών.
Τέλος, μέσα από την μέθοδο Draw που έχουμε δημιουργήσει, εφαρμόζει τα εφέ του κόσμου, την προοπτική και τον προσανατολισμό της κάμερας, χρησιμοποιώντας την κάμερα που έχουμε φτιάξει σε κάποια άλλη κλάση και είναι η βασική κάμερα του κόσμου μας και θα την δούμε στην επόμενη ενότητα.

7.4 Η κλάση Camera

Η κλάση Camera είναι όπως καταλαβαίνουμε και η κλάση που δημιουργέι την κάμερα του κόσμου μας. Τι ουσία θα είχε αν δημιουργούσαμε όλον τον τρισδιάστατο κόσμο, εμπλουτισμένο με κτίρια, δρόμους,

αυτοκίνητα, ανθρώπους και οτιδήποτε αγγίζει την φαντασία μας, χωρίς να υπάρχει η κάμερα που μας βοηθάει να τα δούμε όλα αυτά;
Η κλάση της κάμερας, κληρονομεί την κλάση GameComponetn, η οποία είναι η κλάση που μας δίνει λειτουργικότητα στο παιχνίδι και έχει να κάνει με ολόκληρο το περιεχόμενο του παιχνιδιού.
Όπως μιλήσαμε και σε προηγούμενη ενότητα, η κάμερα για να λειτουργήσει σωστά, χρειάζεται κάποια βασικά συστατικά. Αυτά είναι, η θέα της (view), ο προσανατολισμός της (projection), η θέση της (position), ο στόχος της αν είναι κάμερα τρίτου προσώπου (target), έναν αριθμό που ορίζει το πιο κοντινό σημείο που μπορεί να κοιτάξει (nearPlaneDistance), έναν αριθμό που αντιστοιχεί στο πιο μακρυνό σημείο που μπορεί να δει (farPlaneDistance) και το aspect ratio το οποίο είναι ο λόγος δύο διαστάσεων, του ύψους και του πλάτους του παιχνιδιού μας.
Όλα αυτά είναι πίνακες ορισμένοι μέσα στην κλάση της κάμερας, να δέχονται τιμές και να τις επιστρέφουν, ανάλογα με την θέση της κάμερας κάθε φορα.
Η κάμερά μας είναι ορμένη να κοιτάζει κάθε φορά το πίσω μέρος του αυτοκινήτου και να το ακολουθεί κατά την κίνησή του. Η κάμερα έχει μια φυσική κίνηση γιατί έχει οριστεί και η περιστροφή που θα έχει κάθε φορά που στρίβει το αυτοκίνητο, έτσι δεν είναι γατζωμένη επάνω του ώστε να κάνει απότομες κινήσεις.
Η ενημέρωση της θέας της κάμερας ορίζεται σε μια μέθοδο που ονομάζεται UpdateView. Η μέθοδος αυτή περιέχει έναν πίνακα που δέχεται δύο γωνίες που πολλαπλασιάζονται (X και Y) για να δώσουν την περιστροφή της κάμερας και οι οποίες χρησιμοποιούνται από την μέθοδο που ορίζει τον στόχο της κάμερας. Οι γωνίες αυτές χρησιμοποιούν την μέθοδο Math.Asin, η οποία επιστρέφει μια γωνία που το ημίτονό της είναι ο αριθμός που καθορίζεται.
Στην μέθοδο UpdateView δημιουργούμε ένα διάνυσμα Vector3 που αντιστοιχεί στη θέση του στόχου και δέχεται στην πετίπτωσή μας τη θέση που πρέπει να ακολουθεί η κάμερα, δηλαδή τη θέση του αυτοκινήτου μας.
Η μέθοδος αυτή έχει και άλλο ένα διάνυσμα που είναι το UpVector της κάμερας μας. Και δέχεται ένα διάνυσμα up και την περιστροφή της κάμερας,
Τέλος, στην μέθδο UpdateView δίνουμε και τον πίνακα view που είναι η θέα της κάμερας και δέχεται τη θέση της, τη θέση του στόχου και το UpVector που δημιουργήσαμε προηγουμένως.
Η κλάση της κάμερας περιέχει μια μέθοδο που ονομάζεται UpdateProjection όπου περιέχει τον πίνακα του προσανατολισμού της κάμερας και δέχεται το γενικό πεδίο οτικής της κάμερας (fieldofview), το aspect ratio, το κοντινό και το μακρυνό σημείο που μπορεί να δει η κάμερα και που ορίσαμε προηγουμένως.
Μέσα στην βασική κλάση Game1 του παιχνιδιού μας, δίνουμε την θέση που πρέπει να έχει η κάμερα χρησιμοποιώντας τον παρακάτω κώδικα:

```
Vector3 rotation, sss, bla;
          Quaternion ddd;
          Matrix chup;
      carObject.Car.Chassis.Body.Orientation.Decompose(out sss, out   ddd,
      out bla);
          chup = Matrix.CreateFromQuaternion(ddd);
          rotation = chup.Left;
          rotation.Normalize();
      camera.Position    =    carObject.PhysicsBody.Position    +    (new
      Vector3(0, 3, 0) + rotation * 11);
```

Η παράμετρος out που χρησιμοποιούμε σε αυτό το σημείο είναι ένα reference της C#, το οποίο περνάει μεταβλητές με αναφορες. Είναι παρόμοιο με το ref με τη διαφορά ότι όταν χρησιμοποιώ το ref απαιτείται ότι πρέπει η μεταβλητή να έχει αρχικοποιηθέι πριν περάστεί. Για να χρησιμοποιήσουμε την παράμετρο out. Η μέθοδος διευκρύνησης (definition method) και η μέθοδος κήσης (calling method) πρέπει να χρησιμοποιούν ρητά την λέξη κλειδί (keyword) out.
Το Quaternion (τετραδώνιο) που χρησιμοποιούμε. Τα τετραδώνια αποτελούν την γενικευμένη μορφή των μιγαδικών αριθμών που προκύπτουν από την πρόσθεση των βασικών στοιχείων i, j και k σε πραγματικούς αριθμούς, όπου τα i, j και k ικανοποιούν τη σχέση i^2=j^2=k^2=ijk=-1 και ο πολλαπλασιασμός θεωρείται

προσεταιριστικός. Κάθε τετραδώνιο αποτελεί γραμμικό συνδυασμό των βασικών τετραδωνίων 1, i, j και k. Έτσι, μπορεί να εκφραστεί με μοναδικό τρόπο ως a+bi+cj+dk, όπου a, b, c, και d είναι οι πραγματικοί αριθμοί.

7.5 Οι κλάσεις Sphere και CrashBox

Οι κλασεις Sphere και CrashBox είναι οι δύο κλάσεις που αντιστοιχούν στα δύο μοντέλα μας, τον κύβο και την σφαίρα. Οι κλασεις αυτές μοιάζουν πολύ μεταξύ τους. Όπως είπαμε, τα μοντέλα που χρησιμοποιούν την μηχανή φυσικής JigLibX κληρονομούν από την κλάση PhysicsObject η οποία είναι υπεύθυνη για τη δημιουργία του body και του collision skin που εφαρμόζονται στα μοντέλα αυτά.
Η κλάση Sphere, δέχεται ως ορίσματα τον κόσμο στον οποίο θα παραστεί, ένα μοντέλο, που είναι η σφαίρα που έχουμε ήδη εισάγει στο content pipeline και φορτώσει από την μέθοδο LoadContent στο παιχνίδι μας, μια τιμή δεκαδικού τύπου που είναι η ακτίνα τις σφαίρας, έναν πίνακα που αντιστοιχεί στην περιβαλλοντική προσαρμογή της σφαίρας καθώς και ένα διάνυσμα που ορίζει την θέση της στον τρισδιάστατο κόσμο. Τις τιμές αυτές τις δίνουμε εμείς όταν την δημιουργούμε μέσα από την βασική κλάση του παιχνιδιού μας Game1.
Η κλάση CrashBox, δέχεται σαν ορίσματα κι εκείνη τον κόσμο που θα παραστεί, ένα μοντέλο, που είναι ο κύβος μας που επίσης έχουμε εισάγει και φορτώσει στο παιχνίδι, ένα διάνυσμα που αντιστοιχεί στο μέγεθος των πλευρών του κουτιού, έναν πίνακα που αντιστοιχεί στην περιβαλλοντική προσαρμογή του κουτιού και τέλος τη θέση του κουτιού στον τρισδιάστατο κόσμο μας.
Και οι δύο κλάσεις μας, κληρονομούν όπως είπαμε από την κλάση PhysicsObject, που σημαίνει πως δέχονται ένα νέο σώμα και ένα collision skin που εφαρμόζεται το σώμα αυτό. Το νέο αυτό σώμα μαζί με το collision skin που του εφαρμόστηκε, με τη σειρά του εφαρμόζεται πάνω στο μοντέλο που παίρνει τις ιδιότητες της κλάσης και ενεργοποιείται. Του δίνεται ένα scale μια μάζα, και μια μετακίνηση.
Τέλος εφαρμόζεται σε αυτά ένα εφέ χρώματος.

7.6 Η κλάση Heightmap

Η κλάση αυτή, κληρονομεί όπως και όλα τα αντικείμενα, από την κλάση PhysicsObject. Δέχεται σαν ορίσματα, τον κόσμο στο οποίο θα παραστεί, ένα μοντέλο που αντιστοιχεί στο heightmap που έχουμε δημιουργήσει και εισάγει ως εικόνα δύο διαστάσεων από το content pipeline και έχουμε φορτώσει στην μεταβλητή που χρησιμοποιεί μέσα στην μέθοδο LoadContent και ένα διάνυσμα Vecror2 που αντιστοιχεί στην μεταβολή της θέσης του heightmap. Όπως και με τα άλλα αντικείμενα, οι τιμές που αντιστοιχούν σε αυτά τα ορίσματα, δίνονται κατά τη δημιουργία του heightmap στην βασική κλάση του παιχνιδιού μας Game1 και μέσα στην μέθοδο που δημιουργούνται.
Όπως και με τα άλλα αντικείμενα, έτσι και στο heightmap, εφαρμόζεται ένα body και ένα collision skin.
Η διαφορά με το heightmap είναι, πως εδώ δέχεται και κάποιες επιπλέον ιδιότητες που τις παίρνει από την κλάση HeightmapInfo που έχει δημιουργηθεί για τον χειρισμό ενός εδάφους τύπου heightmap και για την οποία θα μιλήσουμε πιο αναλυτικά παρακάτω.

7.7 Η κλάση CarObject

Η κλάση CarObject αντιστοιχεί στο αντικείμενο του αυτοκινήτου. Κληρονομεί από την κλάση PhysicsObjects όπως και όλα τα υπολοιπα αντικείμενα, που του εφαρμόζει ένα body και ένα collision skin για την φυσική συμπεριφορά του και για να μπορεί να συγκρούεται με όλα τα υπόλοιπα αντικείμενα του κόσμου. Σε αυτή την κλάση δηλώνουμε το μοντέλο του αυτοκινήτου που είναι το πάνω μέρος του, καθώς επίσης και μια ρόδα, που θα χρησιμοποιήσουμε για να δημιουργήσουμε και όλες τις υπόλοιπες του

αυτοκινήτου. Αυτά τα δύο τα παίρνουμε ξεχωριστά ως μοντέλα, για να μπορούμε να δώσουμε στο αυτοκίνητό μας μια φυσική κίνηση και οι ρόδες να μην είναι σταθερές πάνω του, αλλα να μπορούν να έχουν την ανάλογη περιστροφή όποτε εμείς στρίψουμε το αυτοκίνητο στην κατεύθυνση που θέλουμε.
Η κλάση αυτή δέχεται ως ορίσματα, τον κόσμο στον οποίο θα παραστεί, το μοντέλο του αυτοκινήτου, το μοντέλο της ρόδας, μια μεταβλητή τύπου boolean για την μπροστά κίνηση του αυτοκινήτου, μια τιμή boolean για την πίσω κατεύθυνση του αυτοκίνητου, έναν δεκαδικό αριθμό που αντιστοιχεί στη μέγιστη τιμή της γωνίας περιστροφής, έναν δεκαδικό αριθμό για τον ρυθμό της περιστροφής, έναν δεκαδικό που αντιστοιχεί στην τριβή της ρόδας σε μια πλαινή κίνηση του αυτοκινήτου, έναν δεκαδικό αριθμό για την τριβή της ρόδας στην μπροστά κίνηση του αυτοκινήτου, τρεις ακόμα δεκαδικούς για την ρόδα, που αντιστοιχούν στην ακτίνα της ρόδας, την κίνηση, την εξισορρόπησή της, ένα ακέραιο που θα είναι ο αριθμός των ακτίνων για κάθε ρόδα, έναν δεκαδικό για την ροπή που ασκείται, έναν δεκαδικό που αντιστοιχεί στην μείωση του πλάτους ταλάντωσης, έναν δεκαδικό που αντιστοιχεί στην μείωση της κίνησης της ρόδας και έναν δεκαδικό που αντιστοιχεί στην βαρύτητα του.
Οι τιμές αυτές δίνονται στο αυτοκίνητο όταν το δημιουργούμε στην βασική κλάση του παιχνιδιού μας την Game1:

```
carObject = new CarObject(this, car, wheel, true, true, 30.0f, 5.0f, 4.7f,
5.0f,     0.20f,     0.4f,     0.05f,     0.45f,     0.3f,     1,     520.0f,
physicSystem.Gravity.Length());
```

Η μάζα του ορίζεται μέσα σε αυτή την κλάση:

```
SetCarMass(1500.0f);
```

Η κλάση CarObject περιέχει μια μέθοδο που ζωγραφίζει τις ρόδες του αυτοκινήτου, η οποία δέχεται σαν ορίσματα ένα μοντέλο τύπου Wheel και μια τιμή Boolean. Μέσα σε αυτή την κλάση ορίζεται ο προσανατολισμός της ρόδας, η περιστροφή της, η μετατόπισή της. Αυτή η μέθοδος χρησιμοποιεί και την κάμερα που έχουμε δημιουργήσει για κάθε mesh της ρόδας. Μέσα στην μέθοδο Draw της κλάσης CarObject ζωγραφίζουμε μία-μία τις ρόδες.
Στην μέθοδο SetCarMass της κλάσης αυτής, η οποία δέχεται έναν δεκαδικό αριθμό που αντιστοιχεί στη μάζα του αυτοκινήτου, εφαρμόζουμε τη μάζα που του δίνοθμε.

7.8 Η κλάση HeightmapInfo

Η HeightMapInfo είναι μια συλλογή στοιχείων σχετικών με το heightmap. Περιλαμβάνει πληροφορίες για το πόσο ψηλά είναι το έδαφος (terrain), και πόσο μεγάλη είναι η απόσταση μεταξύ των κορυφών του. Επίσης, έχει αρκετές λειτουργίες, για να πάρει πληροφορίες σχετικές με το heightmap, συμπεριλαμβανομένου του ύψους του σε διαφορετικά σημεία, και αν ένα σημείο είναι στο heightmap. Είναι το ισοδύναμο του χρόνου εκτέλεσης του HeightMapInfoContent.

Τα πεδία που περιλαμβάνει αυτή η κλάση είναι τα εξής:

- Ένα **terrainscale**, το οποίο είναι η απόσταση μεταξύ κάθε εισόδου στην στην ιδιότητα Height. Για παράδειγμα, αν το το terrainscale είναι 30, το Height [0,0] και το Height [1,0] είναι 30 μονάδες μακριά το ένα από το άλλο.
- Έναν πίνακα από δεκαδικούς που ονομάζεται heights. Είναι ένας πίνακας δύο διαστάσεων από floats που μας λέει το ύψος κάθε θέσης του heightmap.
- Ένα διάνυσμα Vector3 (heightmapPosition), που ορίζει τις γωνίες -x, -z του heightmap στον τρισδιάστατο κόσμο.

- Έναν δεκαδικό που αντιστοιχεί στο πλάτος του heightmap συμπεριλαμβανομένου του terraiscale (heightmapWidth).
- Έναν δεκαδικό αντιστοιχεί στο ύψος του heightmap, συμπεριλαμβανομένου του terrainscale (heightmapHeight).

Ο constructor της κλάσης HeightmapInfo αρχικοποιεί όλες τις τιμές των μελών.
Η κλάση αυτή περιέχει μια συνάρτηση τύπου boolean, η οποία δέχεται μια θέση και μας λέει πότε η όχι, η θέση είναι στο heightmap. Πρώτα βλέπουμε ποια είναι η θέση του heightmap και μετά ελέγχουμε αν αυτή η τιμή βγαίνει έξω από τα όρια του heightmap.
Η κλάση HeightmapInfo διαθέτει επίσης την συνάρτηση GetHeight, η οποία δέχεται μια θέση και επιστρέφει το ύψος του heightmap σ' αυτό το σημείο. Προσοχή! Αυτή η συνάρτηση θα εμφανίσει ένα exception αν η θέση δεν είναι πάνω στο heightmap. Το πρώτο πράγμα που πρέπει να κάνουμε σε αυτή τη συνάρτηση είναι να βρούμε ποια είναι η θέση του heightmap. Αυτό θα μας το κάνει πιο εύκολο αργότερα. Χρησιμοποιούμε διαίρεση ακεραίων για να βρούμε που στον πίνακα με τα ύψη είναι το positionOnHeightmap. Θυμηθείτε ότι η διαίρεση των ακεραίων πάντα στρογγυλοποιεί προς τα κάτω έτσι ώστε το αποτέλεσμα αυτών των διαιρέσεων να είναι οι δείκτες της πάνω αριστερής από τις τέσσερις γωνίες του παραθύρου.
Στην συνέχεια χρησιμοποιούμε modulus για να βρούμε πόσο μακριά είμαστε από την πάνω αριστερή γωνία του παραθύρου. Το mod μας δίνει μια τιμή από 0 εώς το terrainscale τα οποία μετά διαιρούμε προς το terrainscale για να κάνουμε normalize του σε 1.
Αφού υπολογίσουμε τους δείκτες των γωνιών του παραθύρου μας, και βρούμε που είμαστε στο παράθυρο, χρησιμοποιούμε διγραμμική παρεμβολή για να υπολογίσουμε το ύψος μας. Πρώτα υπλογίζουμε τα ύψη από τις άκρες του πάτου και της κορυφής του παραθύρου μας με το να παρεμβάλουμε από την αριστερη και την δεξιά πλευρα.
Στην συνέχεια παρεμβάλουμε μεταξύ των δύο αυτών τιμών για να υπολογίσουμε το ύψος της θέσης μας.
Τέλος, η κλάση HeightmapInfo, περιλαμβάνει άλλη μια κλάση με όνομα HeightMapInfoReader, η οποία και θα φορτώσει την HeightmapInfo όταν ξεκινήσει το παιχνίδι μας. Αυτή η κλάση ια χρειαστεί για να συνδυάσει το HeightMapInfoWriter.

7.9 Ενδεικτικό παράδειγμα φυσικής σε ένα βασικό 3D κόσμο

Αυτή η ενότητα επιδεικνύει πώς αναπτύχθηκε ένα βασικό κομμάτι του τρισδιάστατου κόσμου χρησιμοποιώντας τη μηχανή φυσικής JigLibX. Ως παράδειγμα αυτής ενότητας θα χρησιμοποιήσουμε δύο κουτιά. Το πρώτο κουτί είναι ένας μεγάλος κύβος που είναι σταθερός. Το δεύτερο κουτί είναι ένας μικρός κύβος που είναι ψηλά στον αέρα. Ο μικρός κύβος πέφτει πάνω στο ακίνητο κουτί και αναπηδά μακριά. Αυτό καταδεικνύει τόσο τη βαρύτητα όσο και την ανίχνευση σύγκρουσης.
Σκοπός αυτής της ενότητας είναι η κατανόηση λειτουργίας της φυσικής που εφαρμόστηκε.

7.9.1 Ξεκινόντας το Project

Το πρώτο-πρώτο πράγμα που πρέπει να κάνουμε είναι να δημιουργήσουμε ένα νέο παιχνίδι, ώστε να γράψουμε μέσα όλον μας τον κώδικα. Προσθέστε τα namespaces που χρειάζονται για την εισαγωγή της μηχανής φυσικής. Η δημιουργία του συστήματος φυσικής γίνεται με τον τρόπο ακριβώς που παρουσιάστηκε και παραπάνω. Μόνο που για λόγους απλότητας και καλύτερης οργάνωσης, θα δημιουργήσουμε μια νέα μέθοδο που θα χρησιμοποιήσουμε για να δημιουργήσουμε εκεί το σύστημα φυσικής μας. Προσθέστε τον παρακάτω κώδικα στην βασική κλάση του project σας και έξω από τον constructor:

```
private void InitializePhysics()
```

```
    {
        PhysicsSystem world = new PhysicsSystem();
        world.CollisionSystem = new CollisionSystemSAP();
    }
```

Όπως είπαμε και προηγουμένως, εδώ δημιουργούμε το σύστημα φυσικής. Αφού ακολουθήσετε και τα υπόλοιπα βήματα που περιγράφτηκαν στην ενότητα 7.2.1 για την εισαγωγή της μηχανής φυσικής, θα χρειαστούμε ένα μοντέλο ενός κύβου. Αυτός ο κύβος θα πρέπει να είναι στο κέντρο του κόσμου που δημιουργήθηκε και να είναι μεγέθους 1. Κάντε εισαγωγή του κύβου αυτού στο ContentPipeline με τον τρόπο που έχει περιγραφεί στα προηγούμενα κεφάλαια.
Αφού έγινε η εισαγωγή του μοντέλου μας και η μηχανή φυσικής μας είναι έτοιμη να κάνει προσομοίωση ενός βασικού κόσμου, το επόμενο βήμα είναι να εμπλουτίσουμε τον κόσμο αυτό με τα κουτιά. Για να το πετύχουμε αυτό θα δημιουργήσουμε μια κλάση που αυτοματοποιεί την διαδικασία.

7.9.2 Δημιουργία της κλάσης BoxActor

Η κλάση που θα δημιουργήσουμε ονομάζεται BoxActor. Η κατηγορία αυτή θα χειριστεί τη δημιουργία ενός κουτιού σε προσομοίωση μας και θα την εμφανίσει στην σκηνή. Για να ξεκινήσουμε, δημιουργήστε ένα νέο κενό αρχείο στο project σας και ονομάστε το BoxActor.cs. Βάλτε τα ακόλουθα στο αρχείο:

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using JigLibX.Math;
using JigLibX.Physics;
using JigLibX.Geometry;
using JigLibX.Collision;

public class BoxActor : DrawableGameComponent
{
    private Vector3 position;
    private Vector3 scale;

    private Body _body;
    public Body Body
    {
        get
        {
            return _body;
        }
    }

    private CollisionSkin _skin;
    public CollisionSkin Skin
    {
        get
        {
            return _skin;
        }
    }

    public BoxActor(Game game, Vector3 position, Vector3 scale)
        : base(game)
```

```
    {
        this.position = position;
        this.scale = scale;
    }
}
```

Αυτό ακριβώς είναι το βασικό περίγραμμα της κλάσης BoxActor. Αυτή κληρονομεί από την DrawableGameComponent διότι η BoxActor θα πρέπει να συντάσσεται έτσι ώστε να μπορούμε να δούμε τι συμβαίνει. Το πεδίο της θέσης (position) κρατάει η προκαθορισμένη θέση του κέντρου του κουτιού. Το πεδίο της κλίμακας (scale) κρατάει το μέγεθος κάθε πλευράς του κουτιού.
Το πεδίο _body και η Body ιδιότητα, είναι αναφορές στο σώμα του κουτιού. Ένα body, στο περιεχόμενο του JigLibX είναι κάτι που κινείται και περιστρέφεται. Δεν μπορεί να συγκρουστεί με οτιδήποτε παρ' όλα αυτά χωρίς τη βοήθεια ενός CollisionSkin. Αυτός είναι ο σκοπός της _skin και του Skin των μελών. Είναι αναφορές στο collision skin του σώματος του κουτιού. Ένα collision skin είναι αυτό που περιγράφει τον φυσικό όγκο του σώματος. Είναι αυτό που λέει στο JigLibX πώς να εντοπίζει συγκρούσεις μεταξύ των σωμάτων.

7.9.3 Δημιουργία ενός Body και Collision Skin

Αυτή τη στιγμή ούτε το body ούτε το skin έχουν οριστεί. Αυτό είναι το σημείο που το project μας γίνεται πιο περίπλοκο. Για να ξεκινήσουμε, προσθέστε τον ακόλουθο κώδικα στο τέλος της βασικής μας κλάσης του παιχνιδιού, στον BoxActor constructor.

```
_body = new Body();
_skin = new CollisionSkin(_body);
_body.CollisionSkin = _skin;
```

Ο κώδικας αυτός είναι πολύ απλός. Χρησιμοποιούμε τον constructor της κλάσης Body για τη δημιουργία ενός νέου σώματος. Κάνουμε το ίδιο πράγμα για το collision skin με εξαίρεση ότι περνάμε τη νέο body που δημιουργήθηκε σε αυτό. Η τρίτη γραμμή του κώδικα καθορίζει το collision skin του σώματος στο νέο collision skin που μόλις κάναμε. Τώρα έχουμε τα βασικά του body και του skin έτοιμα, αλλά απέχουν πολύ από το να τελειώσουν.
Το επόμενο πράγμα που πρέπει να κάνουμε είναι να προσθέσουμε ένα primitive στο collision skin. Τα primitives, όπως έχουμε πει και στα προηγούμενα κεφάλαια, είναι βασικά αντικείμενα που συνθέτουν τα πιο περίπλοκα αντικείμενα σε μια μεγάλη προσομοίωση. Παραδείγματα των primitives αποτελούν, οι σφαίρες, τα κουτιά και τα επίπεδα (planes). Δεδομένου ότι το αντικείμενο που θέλουμε για την προσομοίωση δεν είναι τίποτα περισσότερο από ένα κουτί, το μόνο primitive που χρειαζόμαστε είναι ένα κουτί (box).
Για να δημιουργήσουμε ένα box primitive, δημιουργούμε ένα νέο στιγμιότυπο (instance) της κλάσης Box. Μετά τη δημιουργία του box primitive, χρησιμοποιούμε τη μέθοδο AddPrimitive μέσα στην κλάση CollisionSkin. Η μέθοδος αυτή λαμβάνει το νέο μας primitive και το προσθέτει στο collision skin, ώστε το body μας τώρα να μπορεί να συγκρούεται με άλλα αντικείμενα. Τοποθετήστε τον ακόλουθο κώδικα στο κάτω μέρος του constructor.

```
Box box = new Box(Vector3.Zero, Matrix.Identity, scale);
_skin.AddPrimitive(box, new MaterialProperties(0.8f, 0.8f, 0.7f));
```

Η πρώτη γραμμή δημιουργεί μια νέα κλάση Box. Η πρώτη παράμετρος του constructor είναι η θέση του κουτιού σε σχέση με το σώμα (body). Θέλουμε το κουτί στο κέντρο του σώματος, ώστε η θέση να είναι η

αρχική σε αυτήν την περίπτωση. Η επόμενη παράμετρος είναι ο προσανατολισμός (orientation). Αυτό καθορίζει την περιστροφή του κουτιού σε σχέση με το κουτί. Θέλουμε το κουτί να είναι ευθυγραμμισμένο με το σώμα έτσι ώστε να χρησιμοποιούμε τον πίνακα ταυτότητας (identity matrix), το οποίο σημαίνει ότι δεν υπάρχει περιστροφή. Η τρίτη γραμμή είναι το μέγεθος του κουτιού (scale). Χρησιμοποιούμε την κλίμακα (scale) που δίνεται σε αυτή την κλάση, μέσω του constructor. Η δεύτερη γραμμή προσθέτει το κουτί στο collision skin, ώστε το σώμα να μπορεί να συγκρουστεί με άλλα αντικείμενα. Η πρώτη παράμετρος είναι το primitive που θέλουμε να προσθέσουμε το οποίο είναι το νέο μας κουτί. Η δεύτερη παράμετρος καθορίζει ένα προσαρμοσμένο υλικό (custom material). Ένα υλικό (material), στα πλαίσια της Φυσικής είναι αυτό που καθορίζει τις ιδιότητες της επιφάνειας ενός αντικειμένου, όπως η αναπήδηση και η τριβή. Παίξτε με τις παραμέτρους για να δείτε πώς το καθένα επηρεάζει την προσομοίωση. Μπορείτε επίσης να χρησιμοποιήσετε ένα προκαθορισμένο υλικό, αντικαθιστώντας τη δεύτερη παράμετρο με μια τιμή από το MaterialTable.MaterialID (βεβαιωθείτε ότι έχετε θέση ως int).

7.9.4 Ορίζοντας την μάζα του Box

Τώρα που έχουμε τελειώσει με την προσθήκη του κουτιού θα πρέπει να γίνουν προσαρμογές για το σώμα, επειδή η μάζα του έχει αλλάξει. Αυτές οι αλλαγές παρουσιάζονται καλύτερα μέσω ορισμένου κώδικα. Προσθέστε αυτή τη μέθοδο στην κλάση BoxActor.

```
private Vector3 SetMass(float mass)
{
    PrimitiveProperties primitiveProperties = new PrimitiveProperties(
        PrimitiveProperties.MassDistributionEnum.Solid,
        PrimitiveProperties.MassTypeEnum.Mass, mass);

    float junk;
    Vector3 com;
    Matrix it;
    Matrix itCoM;

    Skin.GetMassProperties(primitiveProperties, out junk, out com, out it,
out itCoM);

    Body.BodyInertia = itCoM;
    Body.Mass = junk;

    return com;
}
```

Η μέθοδος αυτή θα καθορίσει τη μάζα του κουτιού μας. Απλά ρωτάει το collision skin για σημαντικές πληροφορίες σχετικά με το σώμα και παρέχει τις πληροφορίες αυτές στο σώμα αυτό. Επιστρέφει τότε το κέντρο της μάζας του σώματος. Καλούμε αυτή τη μέθοδο μετά την προσθήκη του box primitive στο collision skin. Προσθέστε τον ακόλουθο κώδικα στο κάτω μέρος του constructor.

```
Vector3 com = SetMass(1.0f);
```

Αυτό καθορίζει τη μάζα του σώματος ίσο με ένα, αλλά και αποθηκεύει το κέντρο της μάζας για μελλοντική χρήση.

7.9.5 *Ορίζοντας την θέση του Box*

Στη συνέχεια, μετακινούμε το σώμα στη θέση που καθορίζεται στις παραμέτρους του constructor και προσαρμόζουμε το collision skin να ταιριάζει με το μοντέλο μας. Προσθέστε τα ακόλουθα στο κάτω μέρος του constructor.

```
_body.MoveTo(position, Matrix.Identity);
_skin.ApplyLocalTransform(new Transform(-com, Matrix.Identity));
_body.EnableBody();
```

Η πρώτη γραμμή μετακινεί το σώμα. Το μετακινεί στη θέση που καθορίζεται από την πρώτη παράμετρο που είναι η θέση της BoxActor στην περίπτωση αυτή. Η δεύτερη παράμετρος καθορίζει τον προσανατολισμό του κουτιού. Δεν θέλουμε αυτό να αλλάξει έτσι ώστε να το καθορίζουμε στον identity matrix.
Η δεύτερη γραμμή μετακινεί το collision skin στο κέντρο της μάζας του σώματος. Αυτό είναι εκεί για να ευθυγραμμίσει το μοντέλο που θα δούμε στην σκηνή με το σώμα που το JigLibX θα πρέπει να μετακινεί.
Η τελευταία γραμμή ενεργοποιεί το σώμα για την προσομοίωση, διότι είναι απενεργοποιημένο εξ' ορισμού.

7.9.6 *Δημιουργώντας τον κόσμο*

Τώρα που όλο το framework έχει φτιαχτεί, είναι καιρός να δημιουργήσουμε τον κόσμο μας. Πήγαινε πίσω στη μέθοδο InitializePhysics στην βασική κλάση του παιχνιδιού μας και προσθέστε τα εξής στο τέλος της μεθόδου.

```
fallingBox = new BoxActor(this, new Vector3(0, 50, 0), new Vector3(1, 1, 1));
immovableBox = new BoxActor(this, new Vector3(0, -5, 0), new Vector3(5, 5,
5));
immovableBox.Body.Immovable = true;
Components.Add(fallingBox);
Components.Add(immovableBox);
```

Θα πρέπει επίσης να προσθέσετε τον ακόλουθο κώδικα στην κορυφή της κλάσης για να δηλώσετε τις μεταβλητές που χρησιμοποιούνται.

```
BoxActor fallingBox;
BoxActor immovableBox;
```

Ο κωδικός που μόλις προσθέσαμε αξιοποιεί την κλάση BoxActor για να στήσουμε τον κόσμο μας. Η πρώτη γραμμή δημιουργεί το κουτί που πέφτει, το οποίο ανέφερα στην αρχή της ενότητας. Εμείς το δημιουργούμε όπως και κάθε άλλο στοιχείο του παιχνιδιού εκτός από τις πρόσθετες παραμέτρους. Η πρώτη παράμετρος είναι η βασική κλάση του παιχνιδιού. Η δεύτερη είναι η θέση. Η θέση που προσδιορίσαμε είναι πραγματικά ψηλά και ως εκ τούτου πέφτει. Η τελευταία παράμετρος είναι το μέγεθος του κουτιού. Εμείς απλά θέλουμε να είναι ένα μικρό κουτί.
Η δεύτερη γραμμή κάνει ακριβώς το ίδιο πράγμα εκτός από το ότι το ακίνητο κουτί είναι πολύ πιο χαμηλά και μεγαλύτερο.
Η τρίτη γραμμή χρησιμοποιεί την ιδιότητα του Body για την ακινησία του κουτιού (immovable). Θέτουμε την ιδιότητα Immovable του σώματος να είναι αλήθεια, καθιστώντας έτσι ακίνητο το αντικείμενό μας. Με αυτόν τον τρόπο το κουτί που πέφτει θα μπορεί να συγκρουστεί με αυτό το

κουτί. Σε αντίθετη περίπτωση το ακίνητο κιβώτιο θα πέφτει εξίσου γρήγορα με το πρώτο κουτί και δεν θα συγκρουστούν ποτέ.
Στην τέταρτη και πέμπτη σειρά προσθέτουμε τα κουτιά στη συλλογή των στοιχείων που είναι ενσωματωμένη στην βασική κλάση του παιχνιδιού μας. Με αυτόν τον τρόπο ζωγραφίζονται αυτόματα από την game class.

Τώρα η προσομοίωση μας είναι πλήρης. Τα πάντα στο JigLibX έχουν ρυθμιστεί και είναι έτοιμα να ξεκινήσουν. Αν και δεν έχουμε τελειώσει ακόμα. Αν εκτελέσουμε τώρα το παιχνίδι, θα δούμε μια κενή μπλε οθόνη. Δεν θα δούμε τους ωραίους κύβους μας που συγκρούονται. Αυτό συμβαίνει επειδή το JigLibX το μόνο που κάνει το μέρος του έργου σε ένα παιχνίδι. Θα υλοποιήσει την φυσική, αλλά ποτέ δεν θα σχεδιάσει τίποτα στην σκηνή μας. Πρέπει να το κάνουμε εμείς οι ίδιοι.

7.9.7 Εμφανίζοντας τον κόσμο στη σκηνή

Θυμηθείτε το μοντέλο του κύβου μας που κάναμε στην αρχή. Το σημείο αυτό είναι που θα γίνει χρήση του. Εάν ο κύβος δημιουργήθηκε σωστά στο κέντρο του, θα πρέπει να είναι στην αρχική του θέση και πρέπει να έχει μήκος πλευράς 1.
Όπως κάθε μοντέλο, θα πρέπει να το φορτώσουμε πρώτα από το Content Pipeline. Προσθέστε αυτή τη μέθοδο στην κλάση BoxActor.

```
protected override void LoadContent()
{
    model = Game.Content.Load<Model>("boxModel");
}
```

Θα πρέπει να αλλάξετε το "boxModel" σε αυτό που ονομάσατε το μοντέλο σας στο Content Pipeline.
Πρέπει επίσης να προσθέσετε αυτό το πεδίο στην αρχή της κλάσης BoxActor έτσι ώστε να μπορείτε να χρησιμοποιήσετε το μοντέλο ενώ σχεδιάζετε.

```
private Model model;
```

Στη συνέχεια προσθέστε την ακόλουθη μέθοδο στην κλάση BoxActor.

```
private Matrix GetWorldMatrix()
{
    return
        Matrix.CreateScale(scale) *
        _skin.GetPrimitiveLocal(0).Transform.Orientation *
        _body.Orientation *
        Matrix.CreateTranslation(_body.Position);
}
```

Η μέθοδος αυτή δημιουργεί τον world matrix (πίνακας του κόσμου) που θα κάνει το μοντέλο στην σκηνή να ταιριάζει με το σώμα του κουτιού στο JigLibX. Ο πρώτος πίνακας που χρησιμοποιούμε είναι ένας πίνακας scale για να κάνει το κουτί στο μέγεθος που καθορίστηκε στον constructor.
Ο δεύτερος πίνακας περιστρέφει το κουτί στο πώς το collision skin περιστρέφεται.
Ο τρίτος πίνακας περιστρέφει το κουτί, σύμφωνα με τον προσανατολισμό του σώματος.
Ο τελευταίος πίνακας μετακινεί το κουτί στη θέση του σώματος. Εμείς τα συνδυάζουμε για να πάρουμε τον world matrix του κουτιού.

Εκτός από τον world matrix, η αναπαράσταση του κουτιού απαιτεί έναν πίνακα για την οπτική (view) και έναν για τον προσανατολισμό (projection).Δεδομένου ότι κανένα από τα δύο δεν είναι ορισμένο σε κάποια δεδομένη BoxActor θα τα αποθηκεύσουμε στην βασική κλάση του παιχνιδιού. Προσθέστε τον ακόλουθο κώδικα στην κορυφή της κλάσης του παιχνιδιού.

```
private Matrix _view;
public Matrix View
{
    get
    {
        return _view;
    }
}

private Matrix _projection;
public Matrix Projection
{
    get
    {
        return _projection;
    }
}
```

Αυτός είναι ο βασικός κώδικας για την αποθήκευση της οπτικής και του προσανατολισμού (πίνακες) και για να επιτρέπει τις εξωτερικές κλάσεις να έχουν πρόσβαση σε αυτά. Θα πρέπει να προσθέσετε κώδικα για να ρυθμίσετε τους δύο αυτούς πίνακες. Βάλε τα παρακάτω στο κάτω μέρος του constructor της βασικής κλάσης του παιχνιδιού.

```
_projection = Matrix.CreatePerspectiveFieldOfView(
    MathHelper.ToRadians(45.0f),
    (float)graphics.PreferredBackBufferWidth /
(float)graphics.PreferredBackBufferHeight,
    0.1f,
    1000.0f
);
```

Αντιγράψτε τον παρακάτω κώδικα στο τέλος της μεθόδου Update.

```
_view = Matrix.CreateLookAt(
    new Vector3(0, 5, 20),
    fallingBox.Body.Position,
    Vector3.Up
);
```

Αυτός ο view matrix έχει οριστεί να κοιτάζει τη θέση του κουτιού που πέφτει έτσι ώστε να μπορούμε να το βλέπουμε όλη την ώρα. Μπορούμε επίσης να ρυθμίσουμε τη θέση της κάμερας για μια καλή οπτική λίγο πιο πάνω και πίσω από το ακίνητο κουτί.

Τώρα, η βασική κλάση του παιχνιδιού είναι τελείως έτοιμη. Εμείς θα προσθέσουμε μία ακόμη μέθοδο στην κλάση BoxActor και θα έχουμε τελειώσει. Προσθέστε την ακόλουθη μέθοδο στο κάτω μέρος της κλάσης BoxActor.

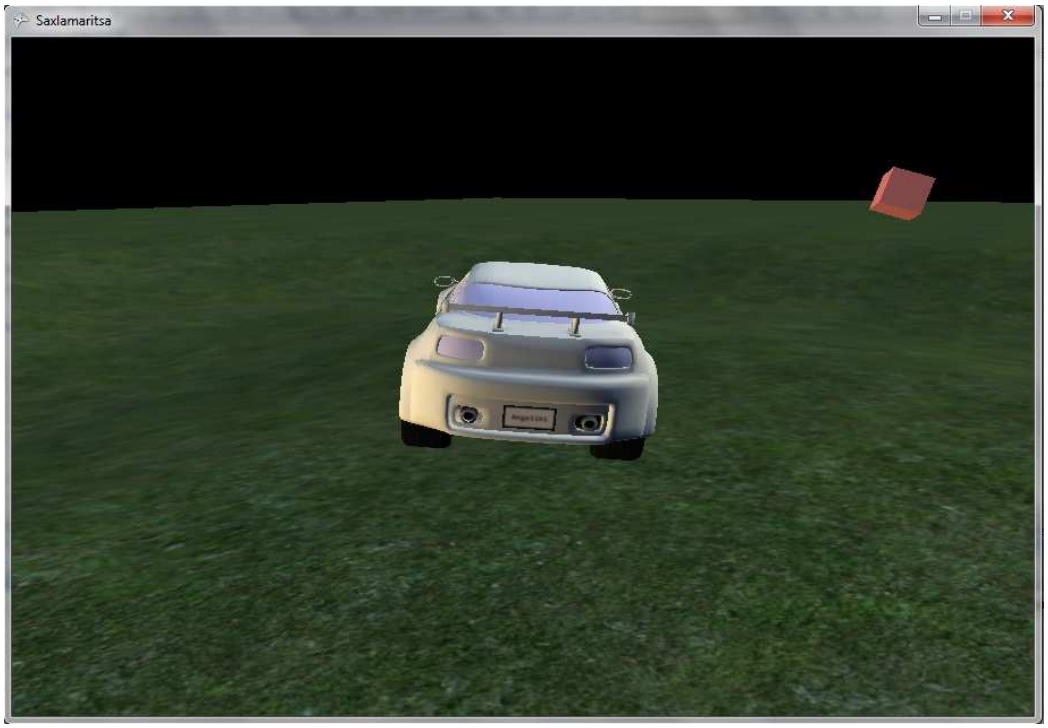
```
public override void Draw(GameTime gameTime)
{
```

```
        BasicWorldGame game = (BasicWorldGame)Game;

        Matrix[] transforms = new Matrix[model.Bones.Count];
        model.CopyAbsoluteBoneTransformsTo(transforms);

        Matrix worldMatrix = GetWorldMatrix();

        foreach (ModelMesh mesh in model.Meshes)
        {
            foreach (BasicEffect effect in mesh.Effects)
            {
                effect.EnableDefaultLighting();
                effect.PreferPerPixelLighting = true;
                effect.World = transforms[mesh.ParentBone.Index] * worldMatrix;
                effect.View = game.View;
                effect.Projection = game.Projection;
            }
            mesh.Draw();
        }
    }
```

Η πρώτη γραμμή καθιστά την κλάση του παιχνιδιού να εκτελείται στο game component στην εφαρμογή μας που είναι η κλάση BasicWorldGame. Αυτό γίνεται έτσι, ώστε να μπορούμε να έχουμε πρόσβαση στους πίνακες view και projection που μόλις προσθέσαμε στην βασική κλάση. Το υπόλοιπο του κώδικα είναι απλά κώδικας για την αναπαράσταση των βασικών μοντέλων μέχρι να φτάσουμε στα μέρη όπου θέτουμε τα τους πίνακες των εφέ. Πολλαπλασιάζουμε τον κανονικό πίνακα του κόσμου (world matrix) με τον world matrix πήραμε από τη μέθοδο GetWorldMatrix. Θέτουμε τότε τους πίνακες view και projection, με αυτούς από την game class.

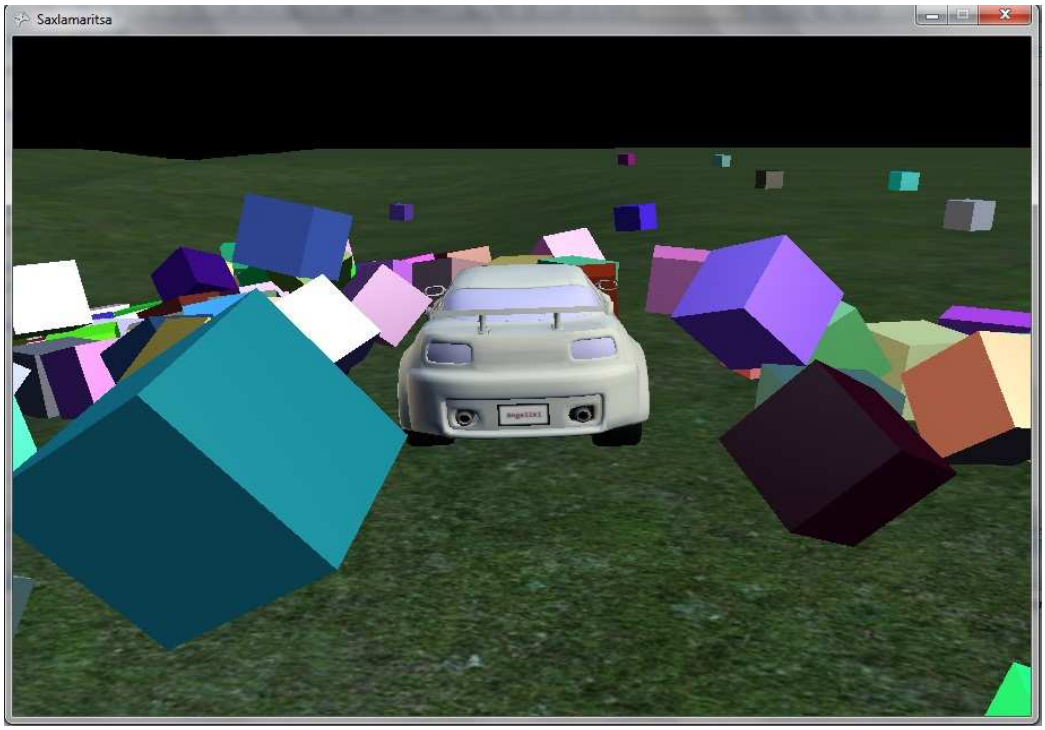
Το project είναι τώρα έτοιμο. Προχωρήστε και εκτελέστε το και θα δείτε το κουτί που πέφτει να συγκρούεται πάνω στο ακίνητο κουτί και να αναπηδά μερικές φορές.
Αυτό αρκεί για την κατανόηση της δημιουργίας ενός βασικού απλού κόσμου με τη βοήθεια της μηχανής φυσικής JigLibX.

7.10 Εικόνες πτυχιακής

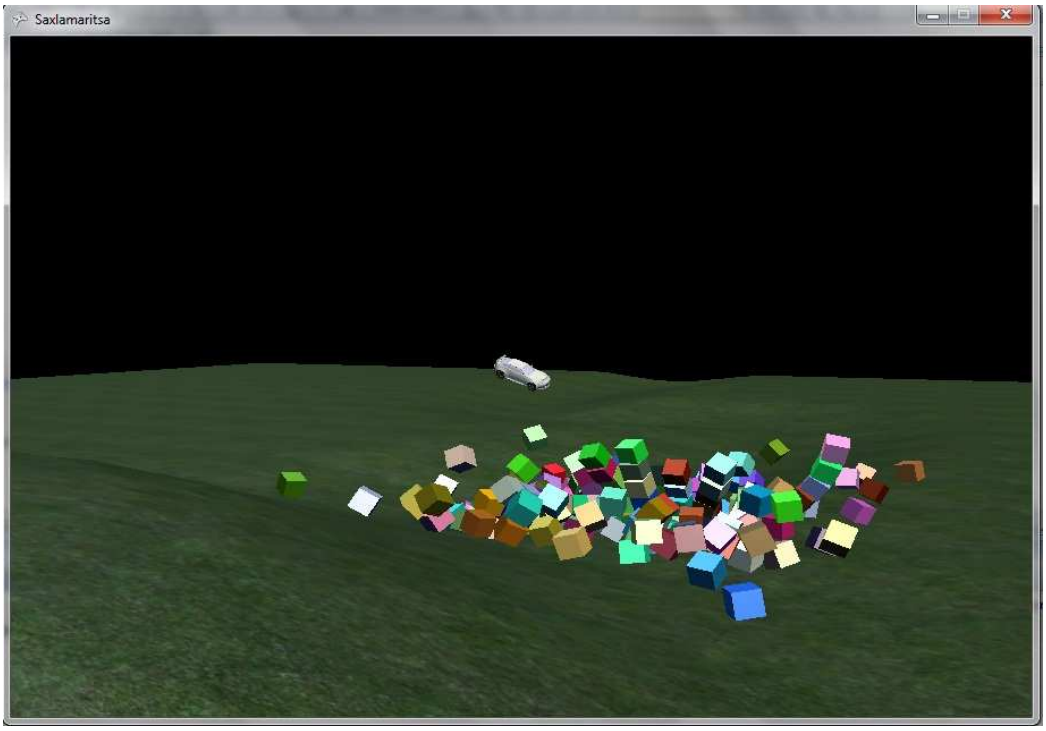


Εικ. 29

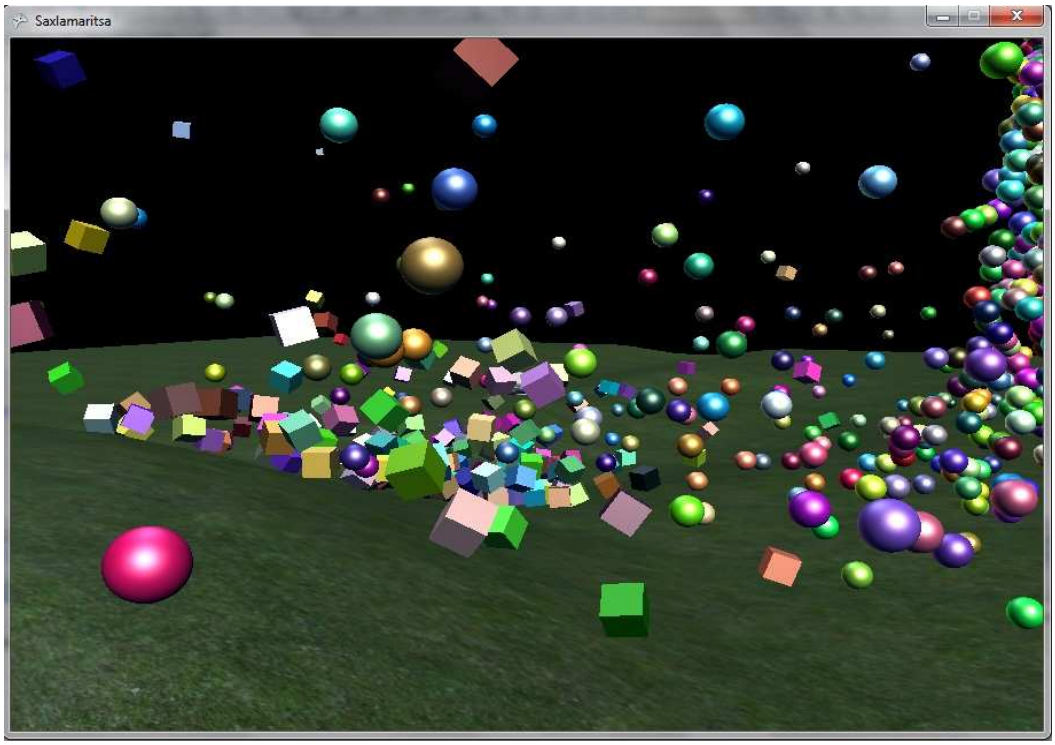
Εικ. 30



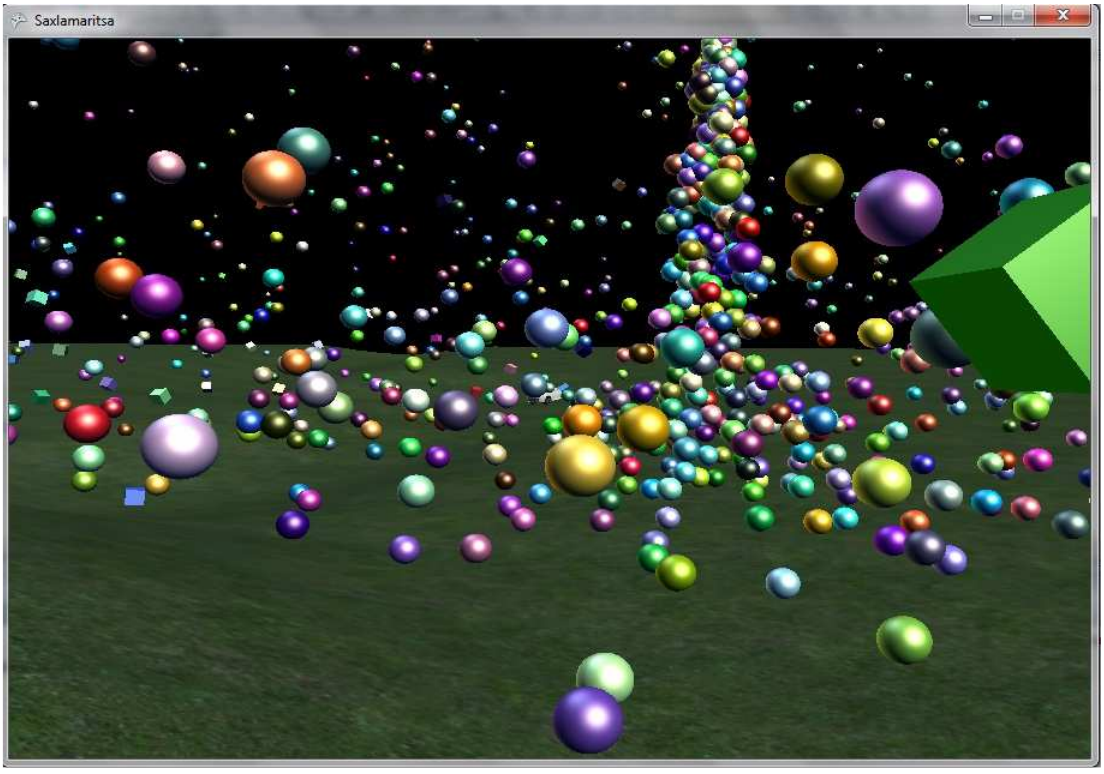
Εικ. 31



Εικ. 32



Εικ. 33



Εικ.34

Εικ. 35

Αναφορά:

Aaron Reed “O’ Reilly Learning Xna 3.0”,

Riemer Grootjans “Xna 3.0 Game Programming Recipies, A Problem-Solution Approach” (Apress),

Alexandre Santos Lobao, Bruno Evangelista, Jose Antonio Leal de Farias and Riemer Grootjans “Xna 3.0 Game Programming, From Novice to Professional” (Apress),

Rob Miles “Introduction to Programming Through Game Development Using Microsoft Xna Game Studio” (Academic Edition),

Xna Creator’s Club Online (http://creators.xna.com),

Youtube video tutorials (www.youtube.com),

JigLibX wiki page (http://jiglibx.wikidot.com).

8

Γνωριμία με το Xna Game Studio 4.0:

Το Xna Game Studio 4.0 μόλις κυκλοφόρησε από τη Microsoft. Τώρα υπάρχει μια μεγάλη γκάμα από πλατφόρμες ανάπτυξης ηλεκτρονικών παιχιδιών. Η ανάπτυξή τους πια θα είναι συμβατή με το λειτουργικό σύστημα των Windows, της κονσόλας Xbox 360 αλλά και του Windows Phone 7. Μια αρχιτεκτονική για την ανάπτυξη όλων.
Microsoft.Xna.Framework.Input.Touch: Όπως είναι γνωστό, το Windows Phone περιλαμβάνει οθόνη αφής (Touch Screen). Για το λόγο αυτό, η ομάδα ανάπτυξης του Xna ήθελε να προσθέσει ένα καινούργιο namespace την αλληλεπίδραση με τους χρήστες. Ο τρόπος εισαγωγής του στον κώδικα της εφαρμογής, γίνεται ως εξής:

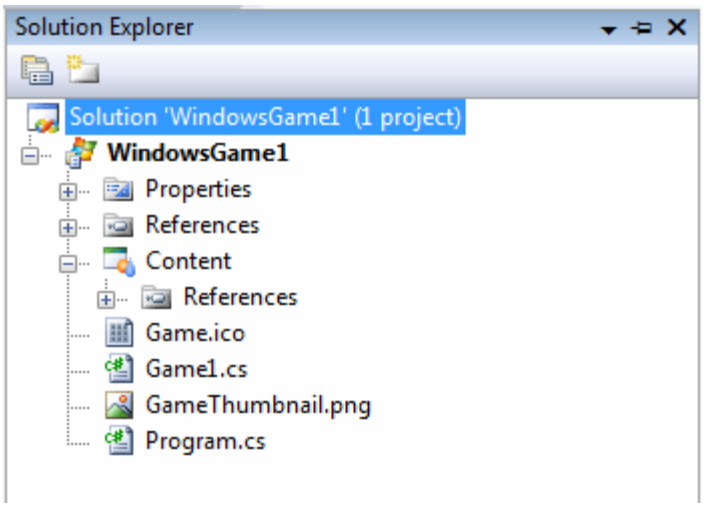
```
using Microsoft.Xna.Framework.Input.Touch;
```

Το namespace αυτό περιέχει κάποια εργαλεία:

- TouchCollection
- TouchLocation
- TouchLocationState
- TouchPanel
- TouchPanelCapabilities

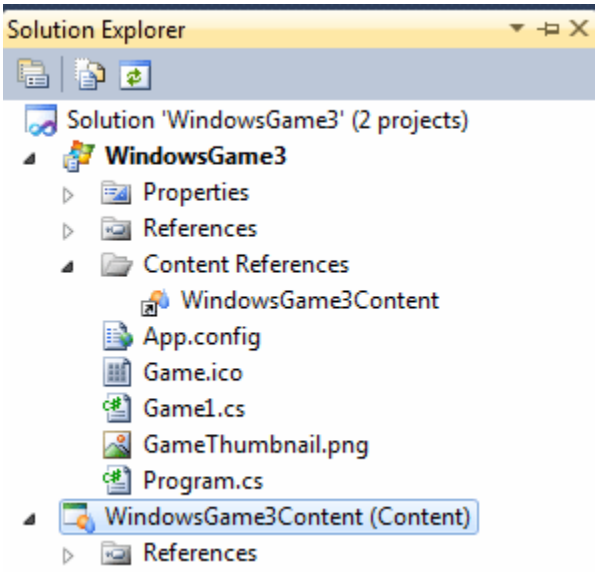
Πιο αναλυτικά:
TouchCollection: Εδώ παρέχεται πρόσβαση στην οθόνη αφής του Windows Phone. Υπάρχει δυνατότητα εισαγωγής νέας Touch Screen τιμής ή όλων των Touch Screen.
TouchLocation: Είναι το σημείο όπου παίρνει όλες τις συντεταγμένες της οθόνης αφής.
TouchLocationState: Αυτές (Invalid, Moved, Pressed, Released), είναι οι πιθανές καταστάσεις της οθόνης αφής. Μπορούν να χρησμοποιηθούν στην TouchLocation ώστε να γίνεται έλεγχος για το αν η οθόνη έχει πατηθεί (pressed), απελευθερωθεί (released), μετακινηθεί (moved), ή ακυρωθεί (invalid).
TouchPanel: Αυτή είναι η οθόνη της συσκευής, από την οποία παρέχονται πληροφορίες για τον τρέχον χειρισμό, την τρέχουσα κατάσταση και τις δυνατότητες που παρέχει η συσκευή.
TouchPanelCapabilities: Εδώ βρίσκονται οι ιδιότητες της συσκευής όπως η πίεση (pressure), η σύνδεση (connection), ο μέγιστος ρυθμός αφής (MaxTouchCount).

Solution Explorer: Στο Xna 3.1, ο Solution Explorer ήταν όπως φαίνεται και στην εικόνα παρακάτω (Εικ. 36):



Εικ.36 Xna Game Studio 3.1 Solution Explorer

Στο Xna 4.0 όμως, βρίσκεται ξεχωριστά από το βασικό project, σαν να έχει προστεθεί ως νέο, το οποίο είναι όμως συνδεδεμένο με το βασικό, όπως φαίνεται και στην Εικ.37:



Εικ.37 Xna Game Studio 4.0 Solution Explorer

Το υλικό (hardware) επιτάχυνε το 3D API στοWindows Phone 7: Αυτό σημαίνει πως τα τρισδιάστατα παιχνίδια θα έχουν την δυνατότητα να παίζονται στο Windows 7 κινητό τηλέφωνο. Είναι ένα πολύ βασικά χαρακτηριστικό το οποίο θα αλλάξει και θα διαμορφώσει την βιομηχανία ανάπτυξης ηλεκτρονικών παιχνιδιών και της αγοράς. Το μόνο που χρειάζεται είναι το Microsoft Visual Studio 2010 και το Xna Game Studio 4.0. είναι ήδη γνωστό πως η συσκευή Zune της Microsoft υποστηρίζει 3D, αλλά είναι αξίσου σπουδαίο το γεγονός ότι υποστηρίζεται και από το Windows Phone.
Xbox Live Υποστήριξη: Είναι ένα σπουδαίο χαρακτηριστικό επίσης. Οι χρήστες μπορούν να προσθέσουν ετικέτες στα παιχνίδια (GameTags) και εικόνες που αναπαριστούν τους ίδιους (avatars), χρησιμοποιώντας τις υπηρεσίες του παιχνιδιού που το κάνουν πιο δυναμικό και διασκεδαστικό.
Όχι Zune στο Xna 4.0: Δυστυχώς, η ανάπτυξη παιχνιδιών τριών διαστάσεων για το Zune θα είναι διαθέσιμη στο Xna 3.1, όχι όμως στο Xna 4.0. Η ομάδα ανάπτυξης του Xna, αποφάσισε να στοχεύσει με το Xna 4.0 στην πλατφόρμα Windows Phone 7 και να μην συμπεριλάβει το Zune σ'αυτό.

Αναφορά:

Wikipedia (http://en.wikipedia.org),

Xna Creator's Club Online (http://creators.xna.com),

Microsoft Xna official homepage (http://msdn.microsoft.com).

9

Βιβλιογραφία και άλλες πηγές

Aaron Reed “O’ Reilly Learning Xna 3.0”

Riemer Grootjans “Xna 3.0 Game Programming Recipies, A Problem-Solution Approach” (Apress)

Alexandre Santos Lobao, Bruno Evangelista, Jose Antonio Leal de Farias and Riemer Grootjans “Xna 3.0 Game Programming, From Novice to Professional” (Apress)

Rob Miles “Introduction to Programming Through Game Development Using Microsoft Xna Game Studio” (Academic Edition)

Microsoft Xna official homepage

Xna Creators Club Online

Xna Community games reviews

JigLibX official homepage

JigLibX wiki page

Wikipedia the free encyclopedia

YouTube: Xna εξήγηση από ειδικούς (video tutorials)

Resources for Xna Game Developers (www.xnaresources.com)

Delta 3D tutorials (www.delta3d.org)