Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή εργασία

«Δημιουργία 3D game σε περιβάλλον Unity Game Engine και χρήση του Kinect για την αλληλεπίδραση με τους χαρακτήρες»

Κορναράκης Ιωάννης (ΑΜ: 2704)

Επιβλέπων Καθηγητής: κ. Μαλάμος Αθανάσιος

Επιτροπή Αξιολόγησης:

Ηράκλειο, Σεπτέμβριος 2014

Ευχαριστώ τους Σταμούλια Ανδρέα και Καπετανάκη Κώστα για τη πολύτιμη καθοδήγηση και υποστήριξη τους καθώς επίσης και τους Κορναράκη Μιχάλη και Τυμπακιανάκη Γεώργιο για τη βοήθεια τους στο στάδιο της έρευνας.

Abstract

The goal of this thesis is to study the most recent methods of creating realistic 3D models, the range of options given by Unity Game Engine and integrating Microsoft Kinect to that engine, utilizing MS-SDK, in order to develop a realistic 3D game, taking as input intuitive player movement.

During this thesis, there was a research conducted on new technologies and a plethora of designing software, mostly Autodesk 3ds Max, Adobe Photoshop, Pixologic ZBrush and Unity Game Engine, developed by Unity Technologies.

At the time of writing this thesis, there was no Unity implementation of Kinect using MS-SDK so we aimed to work on that.

The workflow used to create the final result involves mainly 4 different programs: Autodesk 3ds Max, ZBrush, Unity Game Engine and Adobe Photoshop.

To start with, we create a low-poly model on 3ds Max. This model is then imported into Zbrush where we have the option of slightly upping the poly-count to make a smoother but still light on resources low-poly model to use. At this point, our work can be split into two parallel paths, one, creating additional sub-objects for the model, like clothes, adding high fidelity geometry details and colorization and exporting them as a texture and normal map respectively, in ZBrush, and in the meanwhile going back to 3ds Max to UV Unwrap the model, rig it and create animations.

The final model, comprising of a rigged, animated model with texture and normal maps is then imported into Unity. There, we focus on 3 different processes: creating the environment, setting up the animations for our character and scripting the character's control, the camera behavior etc.

At any point in ZBrush or Unity when we're dealing with maps, we might have to go into Photoshop to fix any image errors.

At this stage, the game is complete but controlled by keyboard input, so we use Kinect to map the player's joints and check their relative position to determine if they're in a desired range. If so, we call functions that communicate with the character controller and make our model move accordingly.

Lastly, we go into details describing the programs used and their capability and we take a look into ways this application could be further developed and what more functions could be implemented to make it a complete game.

Σύνοψη

Στόχος της πτυχιακής εργασίας είναι η μελέτη των πιο πρόσφατων μεθόδων δημιουργίας ρεαλιστικών 3D μοντέλων, το εύρος των δυνατοτήτων που προσφέρει το Unity Game Engine και η σύνδεση του Microsoft Kinect με το engine αυτό, μέσω του MS-SDK του Kinect, με σκοπό τη δημιουργία ενός ρεαλιστικού 3D game με κίνηση του χαρακτήρα βάση ενστικτωδών κινήσεων του παίκτη.

Στα πλαίσια της πτυχιακής αυτής, έγινε έρευνα πάνω στις νέες τεχνολογίες και σε μια πληθώρα σχεδιαστικών λογισμικών, από τα οποία όμως δόθηκε ιδιαίτερη βάση στο 3ds Max της Autodesk, το Photoshop της Adobe, το ZBrush της Pixologic και το Unity Game Engine της Unity Technologies.

Κατά τη περίοδο συγγραφής της παρούσας πτυχιακής δεν υπήρχε κάποια προϋπάρχουσα εφαρμογή σύνδεσης του Kinect με το Unity μέσω του MS-SDK οπότε στοχεύσαμε στο να δημιουργήσουμε ακριβώς αυτό.

Η ροή εργασίας που ακολουθήσαμε για να δημιουργήσουμε το τελικό αποτέλεσμα κάνει χρήση κυρίως τεσσάρων προγραμμάτων: Autodesk 3ds Max, ZBrush, Unity Game Engine και Adobe Photoshop.

Για αρχή, δημιουργούμε ένα μοντέλο μικρού αριθμού πολυγώνων (low-poly) στο 3ds Max. Το μοντέλο αυτό εισάγεται έπειτα στο ZBrush όπου έχουμε την επιλογή να αυξήσουμε ελαφρώς τον αριθμό πολυγώνων του ώστε να έχουμε ένα πιο ομαλά αλλά ακόμα ελαφρύ low-πολύ. Στο σημείο αυτό μπορούμε να δουλέψουμε παράλληλα σε δύο διαφορετικά επίπεδα, δημιουργώντας υπο-αντικείμενα για το μοντέλο μας, όπως ρούχα, προσθέτοντας λεπτομέρειες υψηλής ευκρίνειας και χρωματισμό και εξάγοντας αυτά ως texture και normal maps αντίστοιχα, στο ZBrush, και ταυτόχρονα να επιστρέψουμε στο 3ds Max και να κάνουμε το UV Unwrap του μοντέλου μας, το σκελετό του και τις κινήσεις.

Το τελικό μοντέλο, που απαρτίζεται από το μοντέλο του χαρακτήρα μας, με σκελετικό σύστημα, κινήσεις, ρούχα και texture και normal maps, εισάγεται σε αυτό τη σημείο μέσα στη Unity. Εκεί, επικεντρωνόμαστε σε τρείς διαδικασίες, τη δημιουργία του περιβάλλοντος, τον ορισμό των κινήσεων πάνω στο χαρακτήρα μας και το γράψιμο του κώδικα που θα το μετακινεί, που θα ελέγχει τη κάμερα κλπ.

Σε οποιοδήποτε σημείο είτε στο ZBrush είτε στη Unity όπου ασχολούμαστε με maps, μπορεί να χρειαστεί να μεταβούμε στο Photoshop για να διορθώσουμε τυχόν ατέλειες.

Στο στάδιο αυτό, το παιχνίδι είναι ολοκληρωμένο, αλλά χειρίζεται με είσοδο από το πληκτρολόγιο, οπότε χρησιμοποιούμε το Kinect για να εντοπίσουμε τα joints του παίκτη και να ελέγξουμε τη σχετική τους θέση, ώστε να προσδιορίσουμε το αν βρίσκονται μέσα σε ένα επιθυμητό εύρος. Αν συμβαίνει αυτό, καλούμε συναρτήσεις που επικοινωνούν με το χειριστή του χαρακτήρα και κάνουμε το μοντέλο μας να κινείται αναλόγως.

Τέλος, παρουσιάζουμε τα προγράμματα που χρησιμοποιήθηκαν καθώς και τις δυνατότητες που προσφέρουν και γίνεται αναφορά σε τρόπους με τους οποίους μπορεί η εφαρμογή να εξελιχθεί και τι επιπλέον λειτουργίες μπορούν να αναπτυχθούν πάνω στο τελικό αποτέλεσμα.

Περιεχόμενα

Πίνακας Εικόνων

Κεφάλαιο 1
Εισαγωγή

1.1. Περιγραφή Προβλήματος

Το Kinect είναι μία συσκευή ανίχνευσης κίνησης της Microsoft ,βασιζόμενο σε μια κάμερα βάθους, και προσφέρει στους χρήστες τη δυνατότητα αλληλεπίδρασης με την κονσόλα/υπολογιστή δίχως την ανάγκη για κάποιο άλλο χειριστήριο, μέσω μιας διεπαφής χρήστη που αξιοποιεί φυσικές χειρονομίες και προφορικές εντολές του χρήστη.

Το Microsoft Kinect έχει χρησιμοποιηθεί σε μια πληθώρα παιχνιδιών από διάφορες μεγάλες εταιρίες, όπως τη Microsoft, τη Ubisoft, τη THQ και τη Lucas Arts. Στα παιχνίδια αυτά όμως είτε έχουμε ένα στατικό χαρακτήρα στο χώρο και το Kinect αναγνωρίζει απευθείας κίνηση του σώματος για προσομοίωση χορού, γυμναστικής κ.α., είτε έχουμε μια αυτοματοποιημένη κίνηση του χαρακτήρα και αναγνώριση gestures (χειρονομιών) για την εκτέλεση συγκεκριμένων λειτουργιών. Όλα τα παιχνίδια αυτά επίσης έχουν δημιουργηθεί σε μηχανές διαφορετικές του Unity, που δεν είναι διαθέσιμες στο ευρύ κοινό

Πιο συγκεκριμένα, ας δούμε μερικά παραδείγματα χρήσης του Kinect σε παιχνίδια που βρίσκονται αυτή τη στιγμή στην αγορά. Στο Gunstringer της Microsoft, ο παίκτης ελέγχει έναν χαρακτήρα που ακολουθεί μια προδιαγεγραμμένη πορεία και στοχεύει τα όπλα του και πυροβολεί σύμφωνα με τη θέση των χεριών του παίκτη.

Figure 1.1-1: Gunstringer

Στο Kinect Sports της Microsoft δίνεται στον παίκτη η δυνατότητα να χειριστεί το χαρακτήρα του σε διάφορα σπορ όπως ποδόσφαιρο, τένις κ.α. Ο εκάστοτε χαρακτήρας είναι στατικός στο χώρο, και αναγνωρίζεται κίνηση των χεριών ή των ποδιών για να εκτελεστεί κάποια συγκεκριμένη λειτουργία, π.χ. στο ποδόσφαιρο, η κατεύθυνση στην οποία μετακινούμε το πόδι μας καθορίζει προς ποια από τις τρείς προεπιλεγμένες κατευθύνσεις θα γίνει πάσα.

Figure 1.1-2: Kinect Sports

Η LucasArts σε συνεργασία με την Terminal Reality και τη Microsoft Game Studios δημιούργησε το Kinect Star Wars. Το παιχνίδι αυτό χρησιμοποιεί ένα ενδιαφέροντα τρόπο αλληλεπίδρασης με το χαρακτήρα, καθώς κάνει track τα χέρια του παίκτη και προκαλεί το χαρακτήρα να κουνήσει τα δικά του χέρια με παρόμοιο τρόπο (δεν είναι ακριβής αναπαράσταση της κίνησης). Χρησιμοποιεί επίσης σε ορισμένα σημεία αναγνώριση συγκεκριμένων προκαθορισμένων κινήσεων, κυρίως των χεριών. Ο χαρακτήρας είτε παραμένει στην ίδια θέση κατά τη διάρκεια της σκηνής, είτε κινείται πάνω σε μια έτοιμη τροχιά.


Figure 1.1-3: Kinect Star Wars

Η σειρά παιχνιδιών Just Dance είναι δημιούργημα της εταιρίας Ubisoft και κάθε τίτλος παρέχει μια ποικιλία τραγουδιών και μουσικών στις οποίες μπορεί να χορέψει ο παίκτης. Για το σκοπό

αυτό το παιχνίδι, μέσω του Kinect, αναγνωρίζει τη στάση του σώματος και ανά συγκεκριμένα χρονικά διαστήματα τη συγκρίνει με μια προκαθορισμένη στάση του χορού.

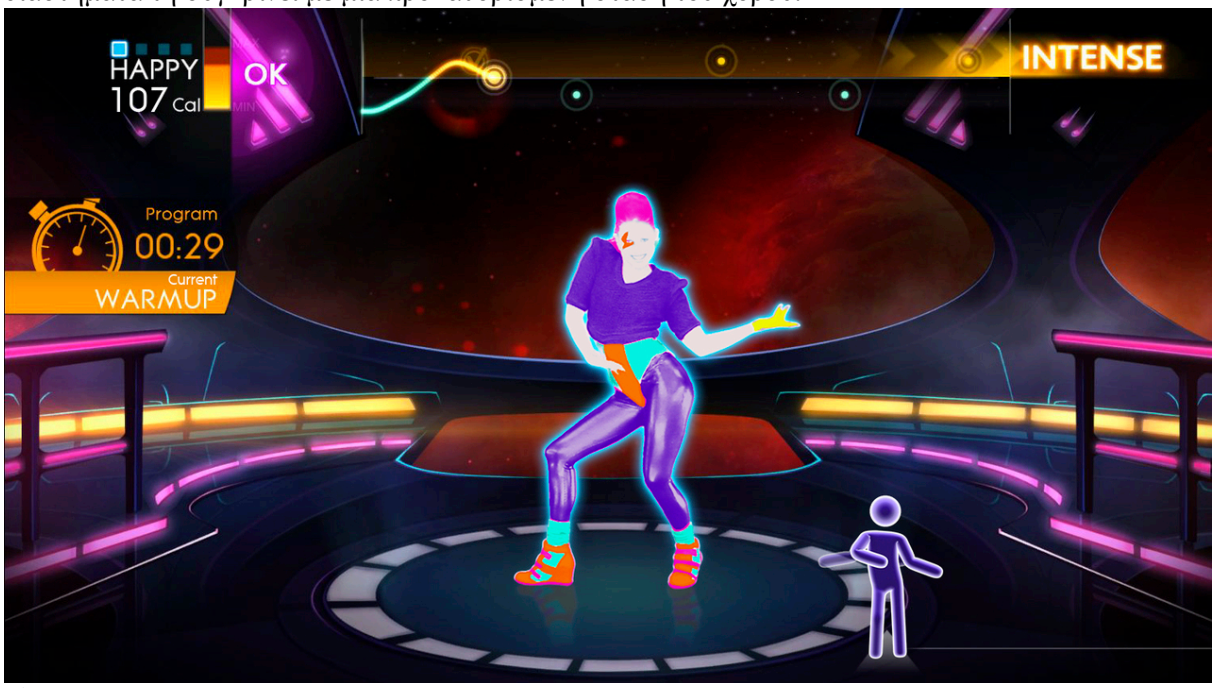

Figure 1.1-4: Just Dance 4

Η Microsoft έχει επίσης δημιουργήσει το παιχνίδι Kinectimals. Στο Kinectimals ο παίκτης φροντίζει ένα εικονικό κατοικίδιο κάνοντας χρήση αποκλειστικά των χεριών του και φωνητικής αναγνώρισης.

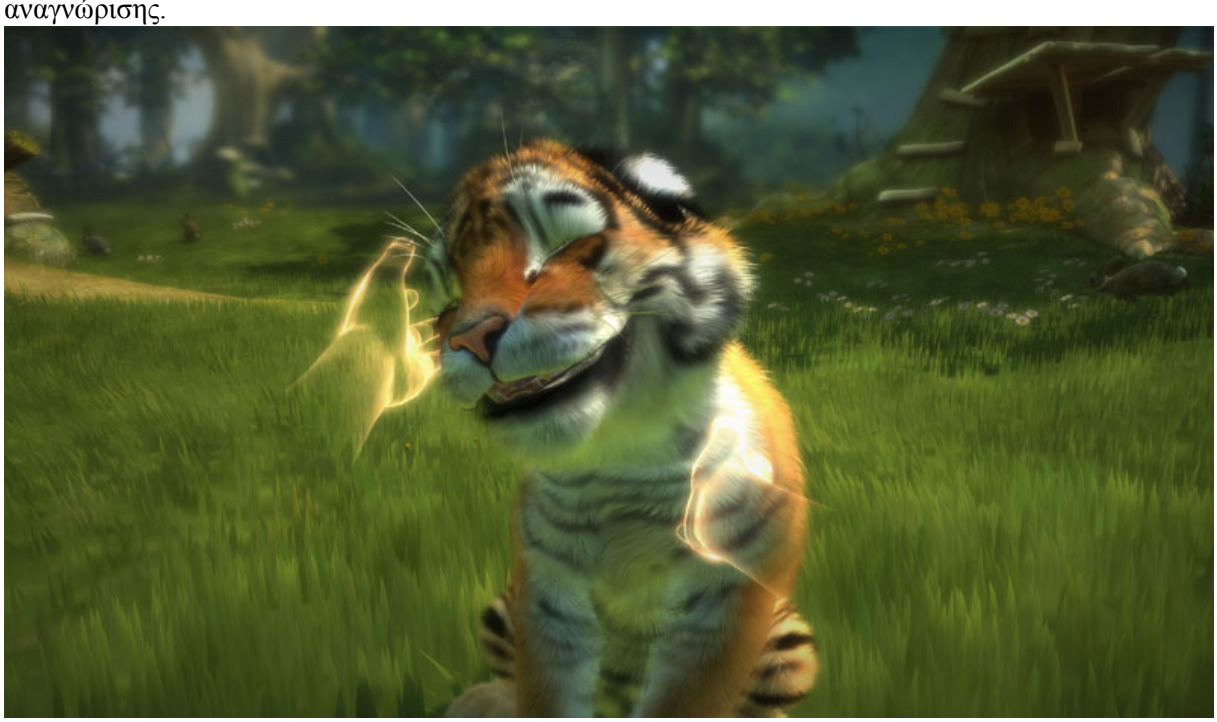

Figure 1.1-5: Kinectimals

Στα παραπάνω παραδείγματα βλέπουμε εν δράση τις δυνατότητες του Kinect, δηλαδή την αναγνώριση συγκεκριμένων κινήσεων και χειρονομιών (gestures), την αναγνώριση της θέσης των επιμέρους μελών του σώματος, την απευθείας απεικόνιση των κινήσεων του παίκτη πάνω στο μοντέλο του χαρακτήρα και τις φωνητικές εντολές.

Το πρόβλημα που εντοπίστηκε με τις προηγούμενες υλοποιήσεις αλλά και όλες τις υλοποιήσεις που έχουν γίνει από μεγάλες εταιρίες με το Kinect είναι ότι στη προσπάθεια τους να αναδείξουν το Kinect κατέληξαν να εστιάζουν ολόκληρο το παιχνίδι πάνω σε αυτό, αντί να το χρησιμοποιούν ως επιπλέον εργαλείο πάνω στις προϋπάρχουσες, δοκιμασμένες τεχνολογίες, περιορίζοντας έτσι τον παίκτη αποκλειστικά και μόνο στις λειτουργίες που παρέχει το Kinect για τη χρήση του παιχνιδιού. Επίσης, έχοντας υπ' όψη μας το πόσο δημοφιλές είναι το Unity, θεωρήθηκε σημαντική απώλεια η έλλειψη συνδεσιμότητας μεταξύ των δύο.

Το Unity είναι μια από τις ευκολότερες και πλέον προσβάσιμες μηχανές δημιουργίας παιχνιδιών και πρόκειται για την καλύτερη και ισχυρότερη λύση για τη δημιουργία multi-platform εφαρμογών και παιχνίδια για κινητές συσκευές. Η μηχανή της Unity είναι μια εύχρηστη δίοδος των ενδιαφερόμενων στο χώρο του game development και διαθέτει αρκετά εργαλεία για τη δημιουργία ενός αξιοπρεπέστατου παιχνιδιού. Είναι προφανές λοιπόν το πόσο επιθυμητή είναι η διασύνδεση μεταξύ Unity και Kinect ώστε κάποιος που ενδιαφέρεται ως επί το πλείστον για τον αισθητήρα βάθους του Kinect ή τις φωνητικές εντολές να μπορέσει γρήγορα και εύκολα να προγραμματίσει ένα παιχνίδι στην δωρεάν έκδοση του Unity, χωρίς να καταπιάνεται με πολυπλοκότερες μηχανές.

1.2. Κίνητρα για την εκπόνηση της εργασίας

Με την συγκεκριμένη υλοποίηση αποσκοπούμε στη δημιουργία ενός πλαισίου παιχνιδιού σε Unity Game Engine στο οποίο μπορούμε να έχουμε κίνηση μέσα στο χώρο με είσοδο από το Kinect.

Σε αντίθεση με τα παιχνίδια που υπάρχουν ήδη αυτό που επιθυμούμε είναι μια εφαρμογή που δε χρησιμοποιεί το Kinect ως την κατά κόρον πηγή ψυχαγωγίας, δηλαδή που να μην βασίζεται τόσο αποκλειστικά στο τρόπο λειτουργίας του Kinect και να βασίζει το παιχνίδι πάνω στο τι μπορεί αυτό να κάνει, αλλά να το χρησιμοποιεί ώστε να μετατρέπει το σώμα του παίκτη σε χειριστήριο, ανοίγοντας έναν τεράστιο ορίζοντα επιλογών όσον αφορά το ίδιο το παιχνίδι. Πιστεύω ότι με αυτή τη λογική μπορεί να υπάρξει ακόμα μεγαλύτερη χρήση του Kinect στην βιομηχανία των video games αφού δεν θυσιάζει αρετές που έχουμε συνηθίσει να έχουμε σε παιχνίδια (ελευθερία κίνησης, πολυπλοκότητα ιστορίας και gameplay, ανοικτοί κόσμοι) αλλά μπορεί επίσης να συνδυαστεί και με άλλες τεχνολογίες για ην παραγωγή εφαρμογών εικονικής πραγματικότητας όπου ο παίκτης αισθάνεται ότι βρίσκεται πραγματικά μέσα στο χώρο του παιχνιδιού. Τέτοιες τεχνολογίες που θα μπορούσαν να συνδυαστούν με το Kinect είναι το Oculus Rift, το Razer Hydra, το Virtuix Omni και το Control VR. Είμαι της άποψης ότι ένας συνδυασμός τουλάχιστον τεσσάρων από των πέντε προαναφερθεισών τεχνολογιών είναι ικανός να παράγει μια άκρως ρεαλιστική εικονική πραγματικότητα για τον παίκτη.

Η εφαρμογή μας δεν είναι περιορισμένη σε κάποιο είδος (genre) παιχνιδιού, αντί αυτού προσφέρεται για τη δημιουργία ενός μεγάλου εύρους ειδών, προγραμματίζοντας τη λογική για το εκάστοτε είδος. Ήδη έχουν δημιουργηθεί τα απαραίτητα script για τη μετατροπή του σε fighting game, όπως είναι τα Mortal Kombat, Tekken, Street Fighter κ.α.

Τη στιγμή που τελειώνει αυτή η εργασία τον Σεπτέμβρη του 2014, υπάρχουν δύο πακέτα σύνδεσης του Kinect με τη Unity, το ZigFu και το Kinect with MS-SDK. Το ZigFu δεν είναι βασισμένο στο επίσημο SDK της Microsoft, αλλά στο OpenNI και κατ' επέκταση στο custom ZDK που δημιουργήθηκε. Προσφέρει όλες τις δυνατότητες του Kinect στο Unity αλλά πάσχει από θέματα συμβατότητας και σταθερότητας.

Το Kinect with MS-SDK από την άλλη κυκλοφόρησε μόλις στις 9 Ιουλίου του 2014, λίγο πριν τελειώσει η υλοποίηση της εργασίας και προσφέρει ότι είναι δυνατόν να προσφέρει το Kinect, αν και με κάποια σημαντικά στοιχεία αποκλειστικά στην μη δωρεάν έκδοση και έχει φτιαχτεί με την Kinect-κεντρική ιδεολογία που αναφέραμε πιο πάνω και όχι με την παιχνιδο-κεντρική στην οποία στοχεύουμε εμείς. Ένα παράδειγμα αδυναμίας αυτής της λογικής είναι η έλλειψη δυνατότητας

συνεχόμενης εισόδου μέσω του Kinect, κάτι που είναι απαραίτητο στη κίνηση ενός χαρακτήρα στο χώρο.

1.3. Στόχοι εργασίας

Οι στόχοι που θέσαμε στην αρχή της εργασίας αποδείχτηκαν είτε ανέφικτοι είτε ελλιπείς οπότε χρειάστηκε να αναθεωρηθούν αρκετές φορές κατά την εκπόνηση της. Οι τελικοί στόχοι έχουν ως εξής:

- Δημιουργία ενός τουλάχιστον ρεαλιστικού ανθρώπινου μοντέλου
  - Αυτό συμπεριλαμβάνει το base mesh, τα maps που χρησιμοποιούμε για να του δώσουμε την εμφάνιση του και οποιαδήποτε επιπλέον αντικείμενα χρειάζονται, όπως μαλλιά, μάτια, ρούχα κλπ.
- Δημιουργία σκελετικού συστήματος για το μοντέλο
  - Απαραίτητο τόσο για τη δημιουργία έτοιμων κινήσεων, όσο και τη πιθανή κίνηση του χαρακτήρα απευθείας από την είσοδο του Kinect
- Δημιουργία βασικών κινήσεων για το μοντέλο
- Δημιουργία περιβάλλοντος
  - Περιλαμβάνει το έδαφος, τη βλάστηση, τον ουρανό, το νερό και οποιοδήποτε άλλο εφέ θα βρίσκεται στον χώρο του παιχνιδιού
- Προγραμματισμός του ελέγχου του χαρακτήρα
- Χρήση των δεδομένων που δίνει το Kinect ως είσοδο ελέγχου για το χαρακτήρα

1.4. Δομή Εργασίας

Η παρούσα πτυχιακή εργασία αποτελείται από την εισαγωγή και άλλα 3 κεφάλαια. Στο αμέσως επόμενο κεφάλαιο παρουσιάζεται η μεθοδολογία εκπόνησης της εργασίας καθώς και το τι ερευνήθηκε μέσα στα πλαίσια αυτής και ακολουθεί η γενική διαδικασία δημιουργίας του παιχνιδιού μας με τα ενδιάμεσα στάδιο. Έπειτα, βλέπουμε αναλυτικά και βήμα προς βήμα όλες τις ενέργειες που διενεργήσαμε ώστε να επιτύχουμε το σκοπό μας, και στο 4^{ο} κεφάλαιο έχουμε μια επισκόπηση του τελικού αποτελέσματος, βλέπουμε εν συντομία τα προβλήματα που αντιμετωπίστηκαν προτού ολοκληρωθεί η υλοποίηση και παρουσιάστηκαν μερικές ιδέες για το τι θα μπορούσε να υλοποιηθεί στο μέλλον πάνω σε αυτή το πλαίσιο παιχνιδιού. Ιδιαίτερο ενδιαφέρον παρουσιάζει επίσης και το παράρτημα, στο οποίο έχουμε μια περιγραφή των κυρίων προγραμμάτων που χρησιμοποιήθηκαν και μια ανάλυση των δυνατοτήτων που προσφέρουν.

1.5. Επεξήγηση συχνά χρησιμοποιούμενων όρων

Framework

Όσον αφορά το προγραμματισμό, ένα πλαίσιο λογισμικού (Software Framework) είναι μια αφηρημένη έννοια σύμφωνα με την οποία ένα λογισμικό που παρέχει γενικές λειτουργίες μπορεί να αλλάξει επιλεκτικά με έναν πρόσθετο κώδικα που γράφει ο χρήστης του πλαισίου, παρέχοντας έτσι πιο συγκεκριμένες εφαρμογές λογισμικού.

Game Engine

Ένα Game Engine (μηχανή παιχνιδιού) είναι ένα πλαίσιο λογισμικού που έχει σχεδιαστεί για τη δημιουργία και την ανάπτυξη video games. Χρησιμοποιείται από προγραμματιστές παιχνιδιών για τη δημιουργία παιχνιδιών για κονσόλες, κινητά τηλέφωνα και υπολογιστές. Η βασική λειτουργικότητα που τυπικά προέρχεται από μια μηχανή παιχνιδιού περιλαμβάνει μια μηχανή renderer, για 2D ή 3D γραφικά, μια μηχανή φυσικής ή ανίχνευσης συγκρούσεων, ήχους, scripting, κινήσεις animation, τεχνητή νοημοσύνη, δικτύωση, streaming, διαχείριση μνήμης, threading, υποστήριξη τοπικοποίησης (localization) και ένα γράφημα σκηνής.

Rendering

Το Rendering είναι η διαδικασία της δημιουργίας μιας εικόνας από ένα 2D ή 3D μοντέλο (ή περισσότερων, σε ό,τι μπορεί συλλογικά να χαρακτηριστεί ως σκηνή) μέσω προγραμμάτων του υπολογιστή. Ένα αρχείο σκηνής μπορεί να περιλαμβάνει αντικείμενα σε μια αυστηρά καθορισμένη δομή της γλώσσας ή των δεδομένων, γεωμετρία, οπτική γωνία, υφή, φωτισμό καθώς και πληροφορίες σκίασης ως περιγραφή της εικονικής σκηνής.

3D μοντέλο

Στα 3D γραφικά, 3D Modeling ονομάζεται η διαδικασία ανάπτυξης μιας μαθηματικής αναπαράστασης οποιασδήποτε τρισδιάστατης επιφάνειας ενός αντικειμένου με τη χρήση ειδικού λογισμικού. Το προϊόν αυτής της διαδικασίας είναι ένα 3d μοντέλο. Τα 3d μοντέλα αναπαριστούν ένα τρισδιάστατο αντικείμενο χρησιμοποιώντας ένα σύνολο σημείων σε ένα 3d χώρο ενωμένα μεταξύ τους με διάφορα γεωμετρικές οντολογίες, όπως τρίγωνα, γραμμές, καμπύλες επιφάνειες κλπ. Καθότι ουσιαστικά μια συλλογή δεδομένων στο 3d χώρο, τα 3d μοντέλα μπορούν να κατασκευαστούν δια χειρός, μέσω αλγόριθμου ή με σάρωση τρισδιάστατων αντικειμένων.

Πολύγωνα

Τα πολύγωνα χρησιμοποιούνται στα 3d γραφικά για να συνθέσουν τρισδιάστατες εικόνες. Συνήθως αλλά όχι πάντα τριγωνικά, τα πολύγωνα συναντώνται όταν μοντελοποιείται η επιφάνεια ενός αντικειμένου, επιλέγονται τα σημεία και το αντικείμενο γίνεται rendered σε wireframe μοντέλο. Ο συνολικός αριθμός των πολυγώνων ενός μοντέλου καθορίζει τον αριθμό των πολυγώνων που γίνεται rendered σε κάθε frame. Όσο πιο μεγάλος ο συνολικός αριθμός των πολυγώνων τόσο πιο λεπτομερές το μοντέλο.

Wireframe

Σκελετός ενός τρισδιάστατου μοντέλου στο οποίο αναπαρίστανται μόνο τα σημεία και οι ακμές που τα ενώνουν.

Mesh (πλέγμα)

Ένα πλέγμα πολυγώνων είναι μια συλλογή από κορυφές, ακμές και επιφάνειες που καθορίζουν το σχήμα ενός πολυεδρικού αντικειμένου σε 3D γραφικά υπολογιστών και μοντελοποίησης στερεών. Οι επιφάνειες συνήθως αποτελούνται από τρίγωνα (triangle mesh), τετράπλευρα (quads) ή άλλα απλά πολύγωνα αφού αυτές απλοποιούν την απόδοση της μορφής του πλέγματος, αλλά μπορούν επίσης να αποτελούνται από γενικότερα κοίλα πολύγωνα, ή πολύγωνα με τρύπες.

Low-Poly

Low-poly είναι ένα πολυγωνικό πλέγμα ενός 3D γραφικού μοντέλου υπολογιστή που έχει ένα σχετικά μικρό αριθμό πολυγώνων. Low-poly γραφικά χρησιμεύουν σε εφαρμογές πραγματικού χρόνου (π.χ. παιχνίδια) σε αντίθεση με high-poly γραφικά που χρησιμεύουν σε ταινίες κινουμένων σχεδίων ή ειδικά εφέ. Ο χαμηλός αριθμός των πολυγώνων σε ένα πλέγμα γραφικού είναι ένας σημαντικός παράγοντας για τη βελτιστοποίηση των επιδόσεων ενός συστήματος, αλλά μπορεί να δώσει μια ανεπιθύμητη χαμηλή ποιότητα εμφάνισης στα προκύπτοντα γραφικά.

Texture Maping

Το Texture Maping είναι μια μέθοδος για την προσθήκη λεπτομέρειας, υφής επιφάνειας, ή χρώματος σε ένα γραφικό ή 3D μοντέλο υπολογιστή. Η εφαρμογή του σε 3D γραφικά ξεκίνησε από τον Edwin Catmull το 1974.

Animation

Το Animation στους ηλεκτρονικούς υπολογιστές είναι ουσιαστικά ένας ψηφιακός διάδοχος των τεχνικών stop motion που χρησιμοποιείται στο παραδοσιακό animation, με 3D μοντέλα και καρέ-καρέ κίνηση από 2D εικόνες. Τα κινούμενα αυτά ψηφιακά σχέδια είναι πιο ελέγξιμα από ότι αντίστοιχα χειρόγραφα κινούμενα σχέδια και επίσης επιτρέπουν τη δημιουργία εικόνων που δεν θα ήταν εφικτό

χρησιμοποιώντας κάποια άλλη τεχνολογία ή τεχνική. Το computer animation επίσης επιτρέπει σε έναν γραφίστα να παράγει περιεχόμενο χωρίς τη χρήση ηθοποιών και σκηνικών.

Particles

Ο όρος «Σύστημα σωματιδίων» (Particle System) αναφέρεται σε μια τεχνική φυσικής παιχνιδιών και γραφικών, που χρησιμοποιεί ένα μεγάλο αριθμό από πολύ μικρά sprites και άλλα αντικείμενα γραφικών για να προσομοιάσει ορισμένα «ασαφή» φαινόμενα, τα οποία είναι κατά τα άλλα πολύ δύσκολο να αναπαραχθούν με τις συμβατικές τεχνικές rendering. Τέτοια φαινόμενα μπορεί να είναι εκρήξεις, τρεχούμενο νερό, φωτιά, σκόνη και άλλα.

Wrapper

Μια συνάρτηση wrapper είναι μια υπορουτίνα μιας βιβλιοθήκης λογισμικού ή ενός προγράμματος που η κύρια λειτουργία της είναι να καλεί μια δεύτερη υπορουτίνα ή ένα σύστημα με λίγο έως καθόλου επιπλέον υπολογισμό.

Κεφάλαιο 2
Μεθοδολογία

Προτού καταλήξουμε στη ροή εργασίας που χρησιμοποιήθηκε εν τέλει στην υλοποίηση της παρούσας εργασίας, έγινε μελέτη πάνω σε 3d τεχνολογίες, τεχνικές δημιουργίας μοντέλων, texturing, animation, γραφικά, και κωδικοποίηση σε C#, Javascript και Python.

2.1. Μέθοδος Ανάλυσης & Ανάπτυξης Πτυχιακής Εργασίας

Για να επιτύχουμε τη λύση του προβλήματος που επιθυμούμε είναι αναγκαία η δημιουργία ενός 3d παιχνιδιού από την αρχή ώστε να έχουμε τον απόλυτο έλεγχο και τα δικαιώματα να τροποποιήσουμε κάθε λειτουργία και λεπτομέρεια του όπως κρίνουμε εμείς απαραίτητο. Έτοιμα αντικείμενα και πακέτα χρησιμοποιήθηκαν όπου θεωρήσαμε ότι δε θα προσέφερε κάποια λειτουργική διαφορά στην υλοποίηση και που η προσωπική δημιουργία αυτών θα ήταν εξαιρετικά χρονοβόρα, τουλάχιστον για τα πλαίσια μιας πτυχιακής εργασίας. Πιο συγκεκριμένα, πήραμε έτοιμα τα μοντέλα των κτιρίων και των βράχων που φαίνονται στο τελικό αποτέλεσμα, το σύστημα νερού και τα ογκομετρικά σύννεφα. Επίσης, παρά το ότι είχε δημιουργηθεί ένα πακέτο αναγνώρισης του Kinect με Unity, βασισμένο πάνω στον κώδικα του ZigFu, στο τελικό αποτέλεσμα χρησιμοποιούμε το πακέτο Kinect with MS-SDK του Unity καθώς η λογική του δε διέφερε από τη λογική του δικού μας αλλά ως κώδικας ήταν πολύ περισσότερο βελτιστοποιημένος και με μικρότερες καθυστερήσεις. Από το πακέτο χρησιμοποιήθηκε δηλαδή μόνο το κομμάτι που δίνει τη πρόσβαση στο σκελετό του Kinect.

Η γενικότερη μεθοδολογία που ακολουθήθηκε ήταν η δοκιμή ενός συνόλου προγραμμάτων και τεχνικών, η επιλογή αυτών που κριθήκαν καταλληλότερες και η δημιουργία όλων των επιμέρους στοιχείων βάσει αυτών των επιλογών.

Τα προγράμματα που εξετάστηκαν ήταν το 3ds Max, το Blender, το Poser Pro 2014 και το MakeHuman. Η επιλογή του ZBrush για τον λεπτομερή σχεδιασμό των μοντέλων ήταν μονόδρομος καθώς είναι το μοναδικό στο είδος του. Εξετάστηκαν τρείς τεχνικές δημιουργίας μοντέλων, η Box Modeling, η Edge-Loop Modeling και η Poly-by-poly Modeling. Οι δύο τελευταίες χρησιμοποιήθηκαν για τη δημιουργία του σώματος και του κεφαλιού αντίστοιχα.

2.2. Συγκριτική μελέτη λογισμικών που απαιτούνται για τη δημιουργία παιχνιδιών

Για την δημιουργία των 3d μοντέλων ερευνήθηκαν διάφορα σχεδιαστικά προγράμματα μέχρι να καταλήξουμε στη χρήση του 3ds Max. Μελετήθηκαν επίσης συγκεκριμένες τεχνικές δημιουργίας μοντέλων στο 3ds Max. Όσον αφορά το κώδικα, είχαμε την επιλογή μεταξύ των τριών παρεχόμενων γλωσσών του Unity, C#, Javascript και Boo, βασισμένη σε Python και επιλέχθηκε η C# χωρίς να υπερτερεί σε κάτι έναντι των άλλων δύο, απλά ως ευκαιρία εκμάθησης και αυτής, καθώς γνώριζα ήδη Javascript και ασχολήθηκα με Python αρκετά κατά τη δοκιμή του προγράμματος Blender.

2.2.1. 3d Προγράμματα

2.2.1.1. Blender

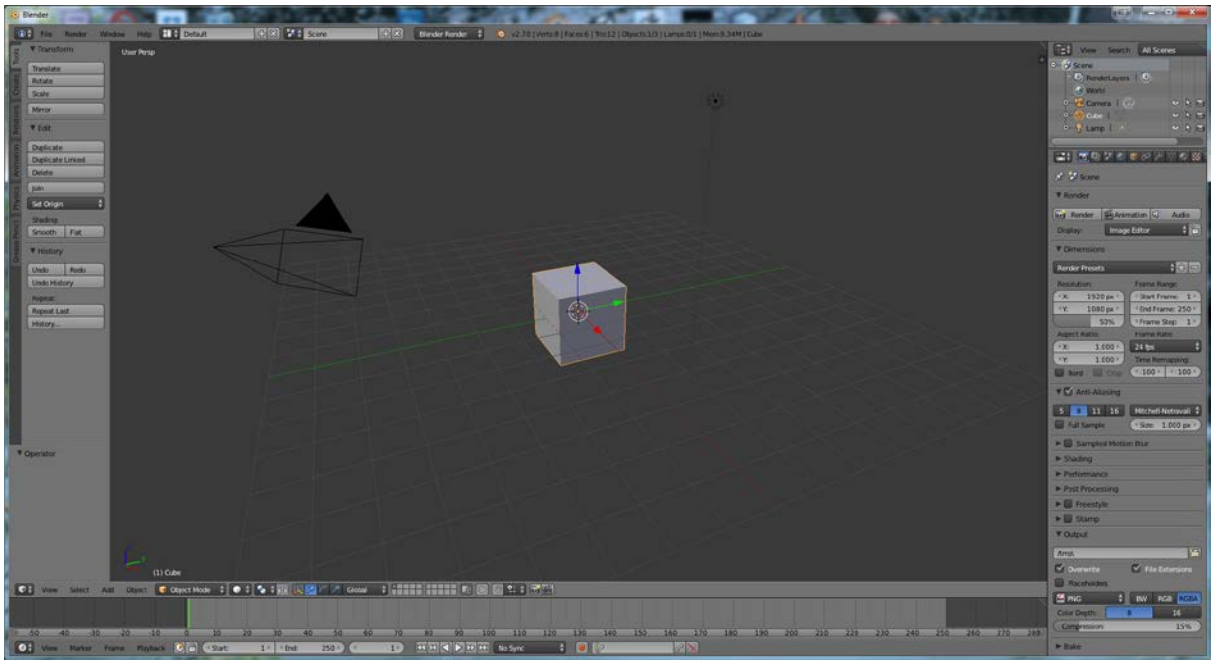

Figure 2.2-1: Blender

Το Blender της εταιρίας Blender Foundation είναι ένα πρόγραμμα που προσφέρει σχεδιασμό 3d, rigging και animation, rendering καθώς και μια απλοϊκή μηχανή δημιουργίας παιχνιδιών. Με διεπαφή χρήστη και χειρισμό που διαφέρει από το 3ds Max ήταν δύσκολη η μετάβαση από το ένα στο άλλο, αλλά κρίθηκε αναγκαία για λόγους συμβατότητας και επεκτασιμότητας που προσφέρει η ενναλακτική λύση.

2.2.1.2. 3ds Max

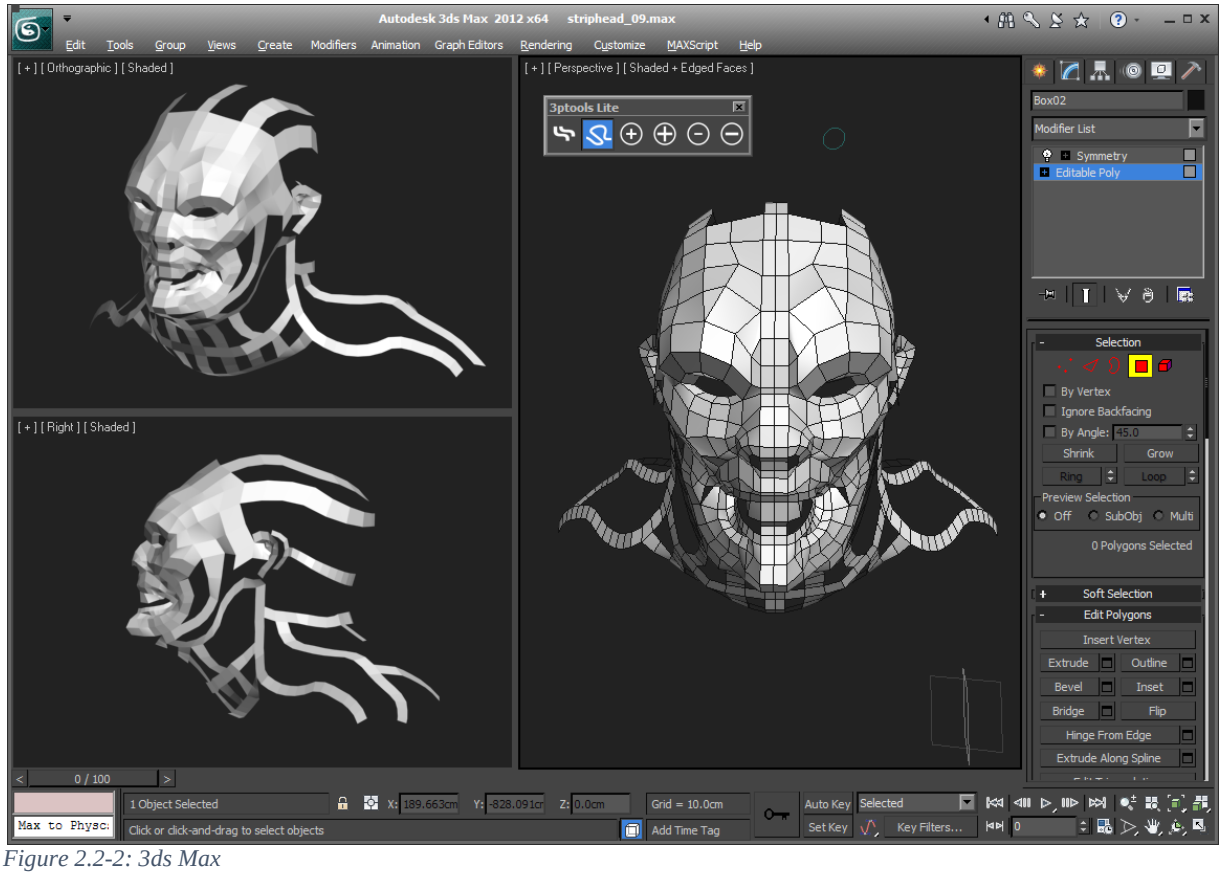

Figure 2.2-2: 3ds Max

Το Autodesk 3ds Max προσφέρει λειτουργίες 3d σχεδίασης, texturing, skinning και animation σε ένα εύχρηστο περιβάλλον. Προσφέρει τεράστιο αριθμό επιλογών και εργαλείων ο οποίος δεκαπλασιάζεται αν συμπεριλάβουμε όλα τα εξωτερικά plugins που μπορούμε να κατεβάσουμε ή να αγοράσουμε. Λόγω της δημοτικότητας της εταιρίας και των χρόνων ύπαρξης της, το 3ds Max είναι το πλέον συμβατό με άλλα πρόγραμμα σχεδίασης 3d.

2.2.1.3. Poser Pro 2014

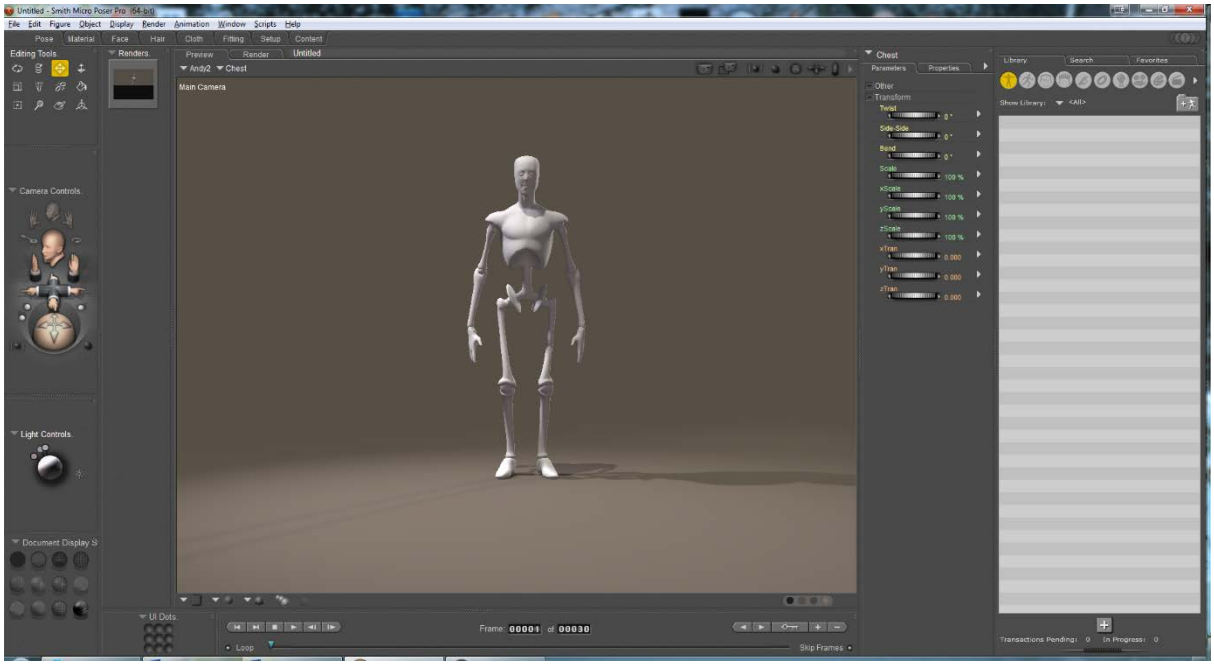

Figure 2.2-3: Poser Pro 2014

Το Poser είναι ένα εργαλείο αυτοματοποίησης της διαδικασίας του rigging και του animation ενός μοντέλου. Σε μικρό χρονικό διάστημα παρατηρήθηκαν οι αδυναμίες εξατομίκευσης του αποτελέσματος οπότε και σταμάτησε η χρήση του.

2.2.1.4. MakeHuman

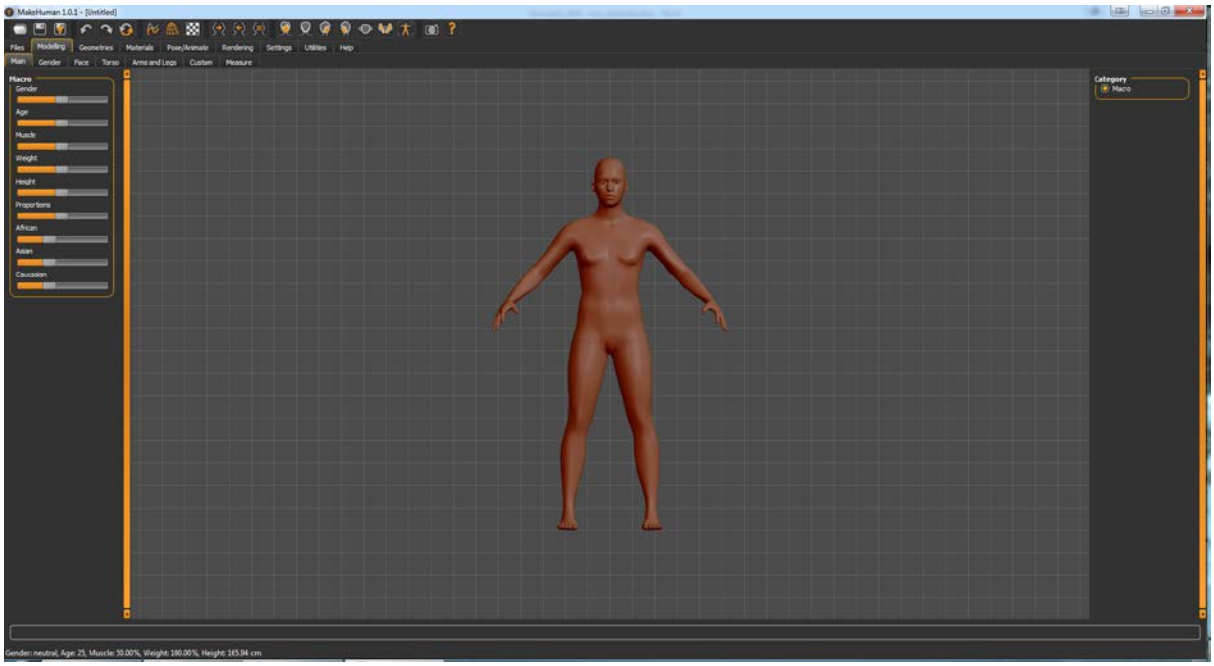

Figure 2.2-4: MakeHuman

Το MakeHuman πρόκειται για ένα open source πρόγραμμα δημιουργίας ανθρώπινων μοντέλων. Η διαδικασία όμως που ακολουθεί είναι τόσο αυτοματοποιημένη και ελλιπής που προτιμήθηκε να δημιουργηθούν από την αρχή τα μοντέλα, παρά τον επιπλέον φόρτο εργασίας που κάτι τέτοιο θα συμπεριλάμβανε.

2.2.2. Λόγοι επιλογής λογισμικών

Αρχικά επιλέξαμε για τη βασική σχεδίαση των 3d μοντέλων το 3ds Max, κυρίως λόγω της σχετικής ευκολίας εκμάθησης των βασικών στοιχείων. Στη συνέχεια, για την δημιουργία ενός μεγαλύτερου βαθμού λεπτομέρειας θεωρήθηκε μονόδρομος η επιλογή του ZBrush καθώς είναι το ισχυρότερο και πιο διαδεδομένο λογισμικό που κάνει χρήση της τεχνικής 3d Sculpting. Για το στάδιο του rigging και του animation θέλαμε στην αρχή να χρησιμοποιήσουμε το Blender, αλλά παρατηρήθηκε ότι το σκελετικό σύστημα που προσφέρει δεν είναι συμβατό με το διαχειριστή κινήσεων Mecanim του Unity, κάτι που θέλαμε να χρησιμοποιήσουμε. Αν και το Blender θεωρητικά προσφέρει περισσότερες λειτουργίες μέσω του game engine mode του, στη πράξη αυτό μας είναι περιττό μιας και θα χρησιμοποιήσουμε τη Unity Game Engine για τη δημιουργία του παιχνιδιού.

	Autodesk 3ds Max	**Blender by Blender Foundation**
3d Design	✓	✓
Texturing	✓	✓
Rigging & Animation συμβατό με Mecanim	✓	✗
Game Engine	✗	✓

Figure 2.2-5: Συγκριτικός πίνακας 3ds Max και Blender

2.2.3. Τεχνικές Σχεδίασης

2.2.3.1. Box Modeling

Το Box Modeling είναι μια τεχνική δημιουργίας 3d αντικειμένων, έχοντας ως βάσης ένα πρωτόγονο (primitive) πολύγωνο, συνήθως κουτί, εξ ου και η ονομασία της τεχνικής. Σύμφωνα με αυτήν, κάνουμε extrude τις πλευρές του κύβου, δηλαδή δημιουργούμε καινούριες ακμές και πλευρές από αυτό και δίνουμε σιγά σιγά το σχήμα που επιθυμούμε. Το μεγάλο θετικό της τεχνικής αυτής είναι ότι με το να χρησιμοποιούμε έναν κύβο για να παράγουμε όλο το μοντέλο, θα έχουμε υπό κανονικές συνθήκες μόνο τετράπλευρες επιφάνειες που όχι μόνο είναι πολύ ελαφριές στον υπολογισμό τους από τα προγράμματα που τις υποστηρίζουν, αλλά μπορούν επίσης πολύ εύκολα να χωριστούν είτε σε δύο είτε σε τέσσερα τρίγωνα, για τα προγράμματα που υποστηρίζουν μόνο τέτοιου είδους επιφάνειες.

2.2.3.2. Edge-Loop Modeling

Ένας βρόγχος ακμών (edge-loop), όσον αφορά τα 3D γραφικά, μπορεί να οριστεί γενικά ως ένα σύνολο συνδεδεμένων ακμών σε μια επιφάνεια. Συνήθως η τελευταία ακμή του συνόλου συναντά και πάλι την πρώτη σχηματίζοντας έτσι ένα βρόγχο. Το σύνολο αυτών των ακμών μπορεί να είναι για παράδειγμα τα εξωτερικά άκρα μιας επίπεδης επιφάνειας ή το τμήμα που περιβάλλει μια «τρύπα» σε μια επιφάνεια.

Πιο συγκεκριμένα, ένας βρόγχος ακμών ορίζεται ως ένα σύνολο ακμών, όπου ο βρόγχος ακολουθεί τη μεσαία ακμή σε κάθε «διασταύρωση τεσσάρων κατευθύνσεων». Ο βρόγχος σταματάει όταν συναντήσει μια διασταύρωση διαφορετικού βαθμού (διασταύρωση τριών ή πέντε κατευθύνσεων για παράδειγμα). Έστω μια ακμή που στο ένα άκρο της συναντά τρείς άλλες, δημιουργώντας έτσι μια διασταύρωση τεσσάρων κατευθύνσεων. Ακολουθώντας συνεχώς το μεσαίο «μονοπάτι» είτε θα βρεθούμε ξανά στο ίδιο σημείο, οπότε θα έχουμε κλείσει βρόγχο, είτε ο βρόγχος θα διακοπεί σε μια διασταύρωση διαφορετικού βαθμού.

Τα edge-loops είναι ιδιαίτερα χρήσιμα σε οργανικά μοντέλα που χρειάζεται να κινούνται. Στην οργανική μοντελοποίηση, τα edge-loops παίζουν κύριο ρόλο στη σωστή παραμόρφωση του πλέγματος πολυγώνων. Ένα ορθά φτιαγμένο 3D μοντέλο πρέπει να λάβει προσεκτικά υπόψη του την τοποθέτηση και τον τερματισμό των βρόγχων. Σε γενικές γραμμές, τα edge-loops ακολουθούν τη δομή και το περίγραμμα των μυών τους οποίους μιμούνται. Για παράδειγμα, στη μοντελοποίηση ενός

ανθρώπινου προσώπου, πρέπει να ακολουθούν τους σφιγκτήρες μύες γύρω από τα μάτια και το στόμα. Αυτό βασίζεται στην ιδέα ότι με το να μιμούνται τα edge-loops τον τρόπο με τον οποίο είναι δομημένοι οι μύες, υποστηρίζουν και την φυσική παραμόρφωση που υφίστανται οι μύες όταν συστέλλονται και διαστέλλονται. Ένα edge-loop μιμείται στενά το πώς λειτουργεί ένας αληθινός μυς, και αν είναι σωστά δομημένο παρέχει τη δυνατότητα να αντιγράψει φυσικά την πραγματική παραμόρφωση και μορφή σε οποιαδήποτε θέση.

2.2.3.3. Poly-by-poly Modeling

Η τεχνική poly-by-poly ορίζει ότι ξεκινάμε με ένα απλό πολύγωνο, ένα plane, και επεκτείνουμε (extrude) τις ακμές του για να δημιουργήσουμε μια επιφάνεια που ταιριάζει στις ανάγκες μας. Για να το πετύχουμε αυτό είναι σχεδόν πάντα απαραίτητη η χρήση ενός σημείου αναφοράς, στη περίπτωση μας, την εικόνα ενός προσώπου. Η τεχνική αυτή αν και πολύ πιο χρονοβόρα από οποιαδήποτε άλλη, μας δίνει μεγάλο έλεγχο πάνω στις λεπτομέρειες του μοντέλου, κάτι που είναι ιδανικό για το σχεδιασμό ενός προσώπου.

2.2.3.4. Ψηφιακή γλυπτική (Digital Sculpting)

Πρόκειται για μια ακόμα αρκετά νέα μέθοδο μοντελοποίησης και έχει γίνει πολύ δημοφιλής μέσα στα λίγα χρόνια που χρησιμοποιείται. Υπάρχουν επί του παρόντος τρείς τύποι ψηφιακής γλυπτικής: Displacement, που είναι ο πιο ευρέως χρησιμοποιούμενος τύπος αυτή τη στιγμή, ο ογκομετρικός και ο δυναμικός διαχωρισμός επιφανειών. Το Displacement χρησιμοποιεί ένα πυκνό πρότυπο (συχνά δημιουργείται από τις επιφάνειες υποδιαίρεσης ενός πλέγματος ελέγχου) και αποθηκεύει νέες θέσεις για τις κορυφές, μέσω της χρήσης μιας χαρτογράφησης 32bit, που αποθηκεύει τις προσαρμοσμένες θέσεις

2.2.3.5. UVW Unwrapping

Το UVW Mapping είναι μια μαθηματική τεχνική για χαρτογράφηση συντεταγμένων. Ορίζεται από το τετράγωνο των πραγματικών στον κύβο των πραγματικών, είναι δηλαδή η απεικόνιση 3D συντεταγμένων σε δισδιάστατη επιφάνεια. Το UVW, όπως και το βασικό Καρτεσιανό Σύστημα Αξόνων, έχει τρείς άξονες, με την τρίτη διάσταση W να αντιστοιχεί στο βάθος, το οποίο επιτρέπει σε χάρτες υφής, texture maps, να τυλίξουν πολύπλοκα αντικείμενα Κάθε σημείο στο UVW map αναφέρεται σε ένα σημείο της επιφάνειας του 3D μοντέλου. Η δουλειά του σχεδιαστή ή του προγραμματιστή είναι να παράγει τη μαθηματική συνάρτηση που δημιουργεί αυτό το χάρτη. Σε γενικές γραμμές, όσο πιο ομαλά είναι «ξετυλιγμένα» τα πολύγωνα, τόσο πιο εύκολο είναι για τον σχεδιαστή να δημιουργήσει τις υφές. Μόλις ολοκληρωθεί η υφή το μόνο που μένει είναι να ξανατυλίξουμε το χάρτη πάνω στο μοντέλο, εμφανίζοντας την υφή με ένα πολύ πιο ευέλικτο και προηγμένο τρόπο, αποτρέποντας τις συνήθεις γραφικές παραμορφώσεις που συνοδεύουν τις πιο απλοϊκές μεθόδους χαρτογράφησης, όπως η επίπεδη προβολή (planar projection). Για το λόγο αυτό το UVW Unwrapping χρησιμοποιείται όταν θέλουμε να δώσουμε υφή σε πολύπλοκα αντικείμενα, όπως είναι ο χαρακτήρας μας.

2.2.3.6. Rigging

Η πρωταρχική τεχνική animation είναι η vertex animation, μια διαδικασία δηλαδή κατά την οποία ορίζουμε σε κάθε σημείο του μοντέλου που θα βρίσκεται σε μια δεδομένη χρονική στιγμή. Σε χαρακτήρες που προορίζονται για παιχνίδια, προτιμάται όμως μια διαφορετική τεχνική, το Skeletal Animation.

Το Skeletal Animation (animation με υποβοηθητικό σκελετικό σύστημα) είναι μια τεχνική στην οποία ένας χαρακτήρας αναπαρίσταται από δύο μέρη: Ένα 3D μοντέλο που χρησιμοποιείται για την εμφάνιση του χαρακτήρα (ονομάζεται και skin) και μια ιεραρχική δομή αλληλοσυνδεμένων αντικειμένων που ονομάζονται οστά ή bones. Η δομή αυτή είναι γνωστή ως σκελετός ή αλλιώς rig. Η τεχνική του skeletal animation χρησιμοποιείται ως επί το πλείστον για την κίνηση οργανικών μοντέλων, αλλά χρησιμεύει στην αποσαφήνιση της διαδικασίας του animation οπότε μπορεί να χρησιμοποιηθεί για να ελέγξει την φυσική παραμόρφωση οποιουδήποτε μοντέλου.

Σύμφωνα με αυτήν την τεχνική, δημιουργούμε οστά για το μοντέλο μας. Τα κάθε οστό έχει ένα τρισδιάστατο μετασχηματισμό, που περιλαμβάνει την τοποθεσία, περιστροφή και μέγεθος αυτού, και ένα προαιρετικό οστό «γονέα». Τα οστά έτσι δημιουργούν μια ιεραρχική δομή. Ο πλήρης μετασχηματισμός κάθε οστού είναι παράγωγο του μετασχηματισμού του οστού-γονέα και του δικού

του τοπικού μετασχηματισμού. Αυτό επιτρέπει την αλληλεπίδραση μεταξύ των συγγενικών οστών, ώστε να μπορούμε, για παράδειγμα, μετακινώντας τον μηρό να μετακινείται μαζί του και η κνήμη.

Όπως προαναφέρθηκε, κάθε οστό αντιστοιχίζεται σε ένα μέρος της οπτικής αναπαράστασης του χαρακτήρα, δηλαδή το μοντέλο. Η διαδικασία με την οποία γίνεται αυτή η αντιστοίχιση ονομάζεται Skinning. Η συσχέτιση μεταξύ σημείων και οστών είναι σχέση πολλά-προς-πολλά. Αυτό σημαίνει ότι κάθε οστό επηρεάζει ταυτόχρονα πολλά σημεία, αλλά αντίστοιχα και κάθε σημείο μπορεί να επηρεάζεται από παραπάνω από ένα οστά. Χρησιμοποιείται έτσι, για κάθε οστό, ένας πίνακας που εμπεριέχει όλα τα συσχετισμένα με αυτό σημεία και το βαθμό στον οποίο το οστό επηρεάζει το κάθε ένα. Ο βαθμός αυτός λέγεται βάρος (vertex weight ή blend weight).

2.2.3.6.1. Σε σύγκριση με το vertex animation:

Πλεονεκτήματα:

- Τα οστά αντιπροσωπεύουν μεγάλο αριθμό σημείων, οπότε διευκολύνεται η κίνηση.
- Χρειαζόμαστε έλεγχο πάνω σε λιγότερα χαρακτηριστικά του μοντέλου.
- Μπορούμε να εστιάσουμε σε κίνηση μεγαλύτερου επιπέδου.
- Κάθε οστό μπορεί να μετακινηθεί ανεξάρτητα.

Μειονεκτήματα:

- Δεν προορίζεται για ρεαλιστική κίνηση μυών και δέρματος

Το μειονέκτημα που αναφέρθηκε μπορεί να αντιμετωπιστεί χρησιμοποιώντας αντικείμενα αποκλειστικά για να ελέγχουν τους μύες, ή αναπτύσσοντας ένα ακριβές μυοσκελετικό σύστημα με εικονικές εξομοιώσεις ανατομίας.

2.2.3.7. Inverse Kinematics

Inverse Kinematics (στη συνέχεια θα αναφέρεται ως IK) είναι μια μέθοδος υπολογισμού θέσης αρθρώσεων που προήλθε από την ανάγκη έργων της ρομποτικής να περιστρέφουν αρθρώσεις ενός βραχίονα έτσι ώστε να τοποθετούν το άκρο του στο επιθυμητό σημείο στο χώρο. Αντίστοιχα, τα IK Solvers που βρίσκονται μέσα στο 3ds Max μας επιτρέπουν να μετακινούμε το τέλος μόνο των άκρων και να έχουμε μια ρεαλιστική κίνηση των υπολοίπων μερών.

2.2.3.8. Skybox

Το skybox είναι μια μέθοδος προσθήκης φόντου που κάνει ψηφιακούς τρισδιάστατους χώρους (level) να φαίνονται μεγαλύτεροι από ότι είναι. Αυτό επιτυγχάνεται με ένα πλαίσιο-κύβο γύρω από το level στο οποίο προβάλλεται η εικόνα του πιο απομακρυσμένου περιβάλλοντος ώστε να δημιουργείται η ψευδαίσθηση μεγαλύτερου περιβάλλοντα χώρου, χωρίς να χρειάζεται αυτός να γίνεται render και να προσθέτει στο φόρτο εργασίας του επεξεργαστή και της κάρτας γραφικών. Σε παλαιότερες χρήσεις του skybox αυτό έμενε σταθερό ως προς το χαρακτήρα για να διατηρείται η ψευδαίσθηση της απόστασης. Νεότερες μηχανές γραφικών όμως, όπως η Source, μετακινούν το skybox, αλλά πολύ πιο αργά από το υπόλοιπο level, δίνοντας έτσι μεγαλύτερο ρεαλισμό στην απόδοση των αποστάσεων και προοπτικών. Αυτό καθιστά δυνατή τη προσθήκη 3d αντικειμένων όπως βουνά και κτίρια τα οποία μπορούν να ενωθούν πχ πρώτοι όροφοι σε ένα ουρανοξύστη είναι κανονικά rendered στο level ενώ οι υπόλοιποι είναι rendered στο skybox αλλά σε πολύ μικρότερη κλίμακα (3d skybox).

2.2.3.9. Shaders

Όταν αναφερόμαστε σε shaders στην πληροφορική και στα γραφικά, εννοούμε ένα πρόγραμμα που χρησιμοποιείται για να κάνει σκίαση, να παράγει δηλαδή τα κατάλληλα επίπεδα του χρώματος μέσα σε μια εικόνα ή ειδικά εφέ. Ένας ορισμός με απλά λόγια θα μπορούσε να είναι: «Ένα πρόγραμμα που διδάσκει έναν υπολογιστή πώς να σχεδιάσει κάτι με ένα συγκεκριμένο και μοναδικό τρόπο».

Ο Shader κάνει τους υπολογισμούς του για το rendering πάνω στη hardware γραφικών, με υψηλό βαθμό ευελιξίας. Η πλειοψηφία των shaders κωδικοποιούνται για μία μονάδα επεξεργασίας γραφικών (Graphic Processing Unit – GPU), αν και αυτό δε πρόκειται για αυστηρή απαίτηση. Οι γλώσσες σκίασης συνήθως χρησιμοποιούνται για τον προγραμματισμό του προγραμματιζόμενου GPU

Rendering Pipeline, ο οποίος έχει ως επί το πλείστον αντικαταστήσει τον fixed-function pipeline που επέτρεπε μόνο κοινούς μετασχηματισμούς γεωμετρίας και σκίαση pixel, ενώ οι shaders υποστηρίζουν προσαρμοσμένες ιδιότητες και εφέ. Η θέση, η απόχρωση (hue), ο κορεσμός χρώματος (saturation), η φωτεινότητα και η αντίθεση κάθε pixel, σημείου ή texture που χρησιμοποιείται για την απόδοση μίας τελικής εικόνας, μπορούν να μεταβληθούν όσο εκτελούνται άλλες διεργασίες, χρησιμοποιώντας αλγόριθμους που ορίζονται από το shader, και μπορούν να τροποποιηθούν με εξωτερικές μεταβλητές ή υφές που εισάγει το πρόγραμμα μέσω του shader.

Οι Shaders χρησιμοποιούνται ευρέως στον κινηματογράφο κατά την επεξεργασία των σκηνών μετά τα γυρίσματα, σε computer-generated εικόνες (CGI) και σε video παιχνίδια, για την παραγωγή μίας φαινομενικά άπειρης ποικιλίας αποτελεσμάτων. Πέρα από απλά μοντέλα φωτισμού, πολυπλοκότερες χρήσεις των shaders περιλαμβάνουν τροποποίηση των χρωματικών παραμέτρων, τη δημιουργία blur, το εφέ light bloom, τον ογκομετρικό φωτισμό, το normal mapping για στοιχεία βάθους, το φωτογραφικό εφέ bokeh, τη τεχνική cel shading (toon shading), το posterization, το bump mapping, τη παραμόρφωση, το chroma keying (γνωστό ως «bluescreen/greenscreen" εφέ), ανίχνευση ακμών και κίνησης, psychedelic εφέ και ένα ευρύ φάσμα άλλων.

Η ονομασία προήλθε από τη δημοσίευση «RenderMan Interface Specification, Version 3.0» της Pixar το Μάιο του 1988, έναν οδηγό στη παραγωγή φωτορεαλιστικών γραφικών.

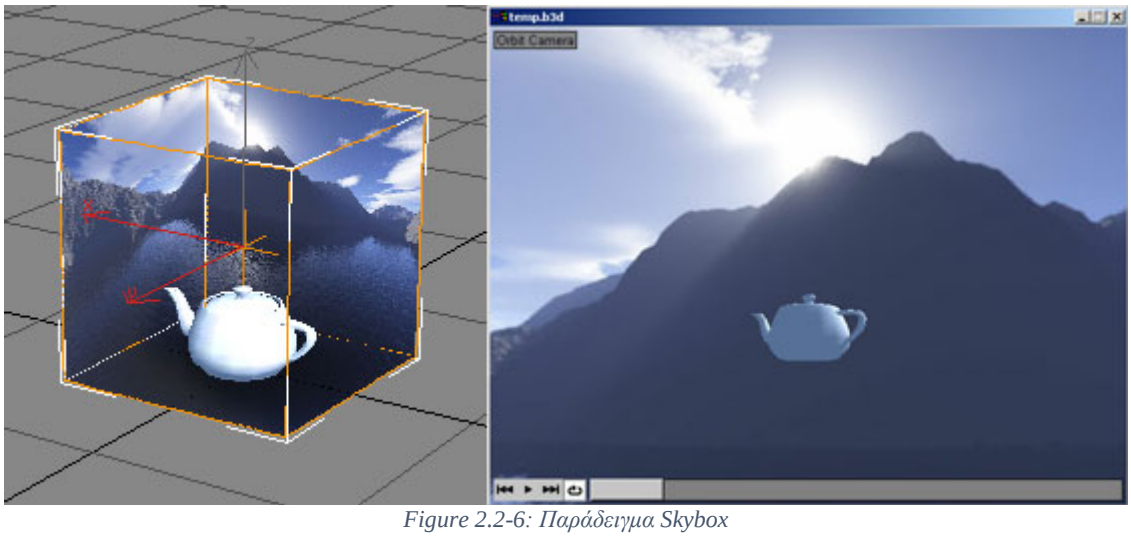

Figure 2.2-6: Παράδειγμα Skybox

2.2.3.10. Mecanim

Το Mecanim είναι ένα ισχυρό και ευέλικτο σύστημα animation, αποκλειστικό στη Unity, που βοηθά στη διαχείριση των κινήσεων ανθρωποειδών και μη χαρακτήρων με φυσική, ρευστή μετάβαση από τη μια κίνηση στην άλλη.

Όντας ενσωματωμένο μέσα στη Unity καθιστά τη χρήση τρίτων προγραμμάτων για τη βελτιστοποίηση του animation περιττή. Παρέχει όλα τα απαραίτητα εργαλεία για τη δημιουργία μηχανών καταστάσεων, ελεγκτές και blend trees κατευθείαν μέσα στη Unity.

Μπορεί να χρησιμοποιηθεί για το animation οποιουδήποτε αντικειμένου στη σκηνή, από πολύπλοκους χαρακτήρες μέχρι sprites και φώτα. Επίσης, χάρη στα AnimationEvents είναι δυνατό να τρέξει κάποιο script μέσα από την αναπαραγωγή ενός animation.

Η σταθερότητα που παρέχει η Unity σε συνδυασμό με τις μεθόδους βελτιστοποίησης του Mecanim εξασφαλίζουν μια ομαλή και ελαφριά σε υπολογιστική ισχύ απόδοση κατά την εκτέλεση.

2.3. Ροή εργασίας

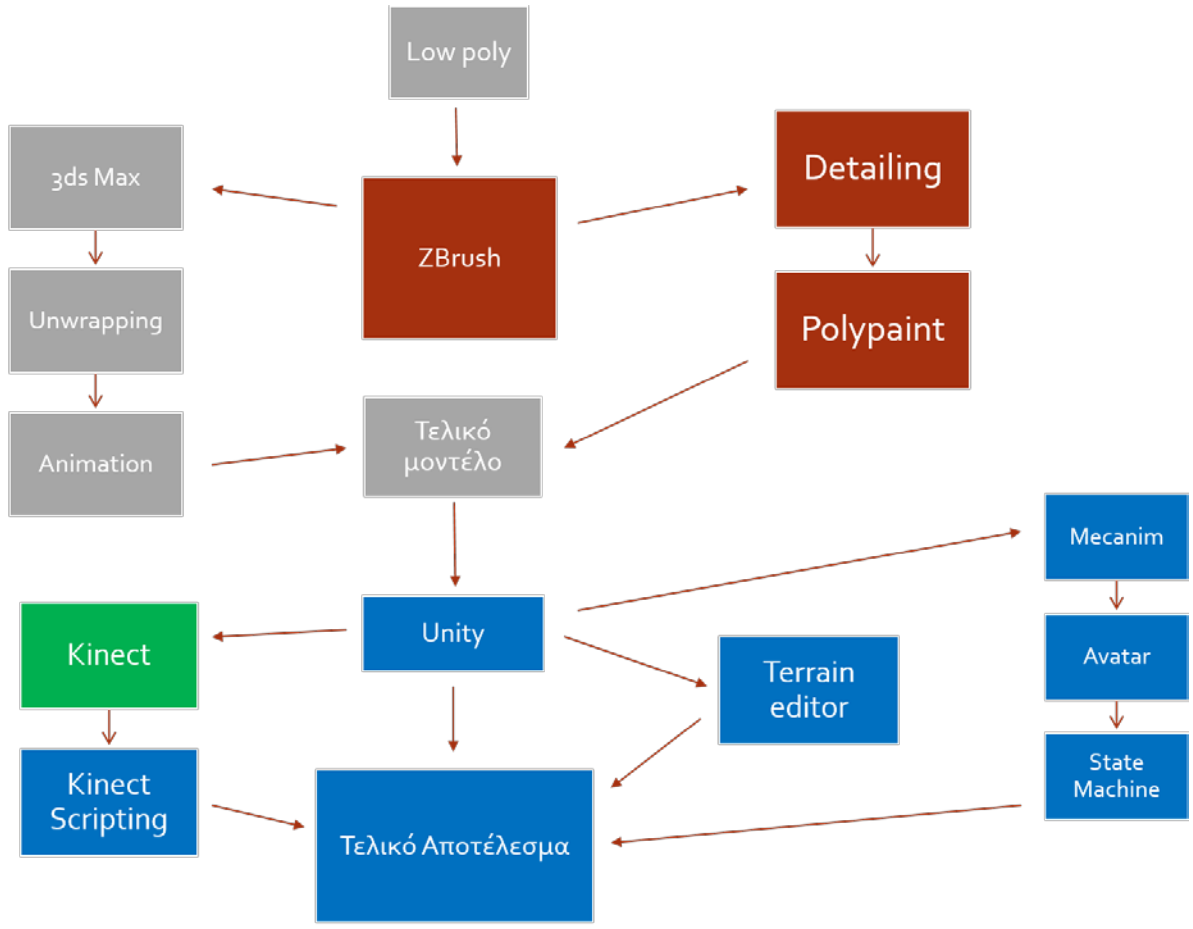

Figure 2.3-1: Workflow

1.1. Περιγραφή ροής

Ξεκινάμε δημιουργώντας ένα βασικό, ανθρώπινο, 3d μοντέλο στο 3ds Max, αξιοποιώντας και συνδυάζοντας δύο διαφορετικές τεχνικές, την Edge-Loop Modeling για το σώμα και την Poly-by-poly Modeling για το κεφάλι. Αφού ενώσουμε τα δύο ξεχωριστά αυτά κομμάτια, μεταφέρουμε το μοντέλο μας μέσα στο ZBrush, όπου του κάνουμε ένα subdivision για να πάρουμε ένα μοντέλο, που και αν είναι ακόμα αρκετά low-poly για να μην γίνεται βάρος στη game engine, είναι πολύ πιο ομαλό από το προηγούμενο. Στο σημείο αυτό διακλαδωνόμαστε προς δύο κατευθύνσεις, επιστροφή στο 3ds Max και συνέχιση εργασίας στο ZBrush.

Επαναφέρουμε το καινούριο low-poly μοντέλο μας μέσα στο 3ds Max όπου κάνουμε unwrap τις επιφάνειες του, του προσαρμόζουμε στο σκελετικό σύστημα Biped και δημιουργούμε τις επιθυμητές κινήσεις.

Παράλληλα και αφού εισάγουμε πλέον το μοντέλο που έχει γίνει unwrapped, κάνουμε κι άλλα subdivision στη γεωμετρία του μοντέλου μας και σε κάθε στάδιο σμιλεύουμε όλο και περισσότερες λεπτομέρειες. Έπειτα, περνώντας ξανά από κάθε στάδιο ξεχωριστά, χρωματίζουμε τις επιφάνειες μας με τη μέθοδο Polypaint του ZBrush. Τέλος, όλη η λεπτομέρεια και το χρώμα που έχουμε βάλει θα γίνουν export ως normal και texture map αντίστοιχα.

Έχοντας τελειώσει με τα παραπάνω, έχουμε στη διάθεση μας ένα ολοκληρωμένο μοντέλο το οποίο μπορούμε να εισάγουμε στο Unity. Εκεί, έχουμε πάλι κάποιες σχεδόν ανεξάρτητες μεταξύ τους διεργασίες να ακολουθήσουμε προτού τα συνδυάσουμε όλα μαζί στο τελικό αποτέλεσμα. Οι διεργασίες αυτές είναι ο προγραμματισμός του Kinect, η σχεδίαση του περιβάλλοντος μέσω του

Terrain Editor της Unity και το στήσιμο του Mecanim για το χαρακτήρα μας, το σύστημα δηλαδή που προσαρμόζει τις κινήσεις μας για το παιχνίδι και επιλέγει ανά πάσα στιγμή ποια κίνηση θα εκτελείται.

Κεφάλαιο 3
Υλοποίηση

3.1 Χρήση του Autodesk 3ds Max στη πτυχιακή

Ο αρχικός σκοπός ήταν το 3ds Max να χρησιμοποιηθεί μόνο για τη μοντελοποίηση του χαρακτήρα, τη δημιουργία δηλαδή του βασικού, low-poly μοντέλου και το uvw unwrap, και να αντικατασταθεί από το Blender για τη δημιουργία του σκελετού και των animation. Μεταφέροντας όμως το έργο μέσα στη Unity, παρατηρήθηκε ότι το Armature του Blender δεν υποστηρίζεται από το Mecanim της Unity, το κύριο, πλέον πρόσφατο και ισχυρό εργαλείο της Unity για animation χαρακτήρων, οπότε και προτιμήθηκε εν τέλει το σκελετικό σύστημα Biped του 3ds Max και κατ' επέκταση έγιναν τα animation πάνω σε αυτό.

Το βασικό μοντέλο δημιουργήθηκε με βάση δύο μεθοδολογίες, edge-loop modeling για το σώμα και poly-by-poly για το πρόσωπο:

3.1.1.1 Εφαρμογή:

Ξεκινώντας, δημιουργούμε ένα primitive αντικείμενο, συνήθως κύλινδρο. Το πλήθος των πλευρών του κυλίνδρου έχει μεγάλη σημασία, καθώς κατά κύριο λόγο θα κρατήσουμε τον ίδιο αριθμό σημείων πάνω στην ακμή για το μεγαλύτερο μέρος του μοντέλου.

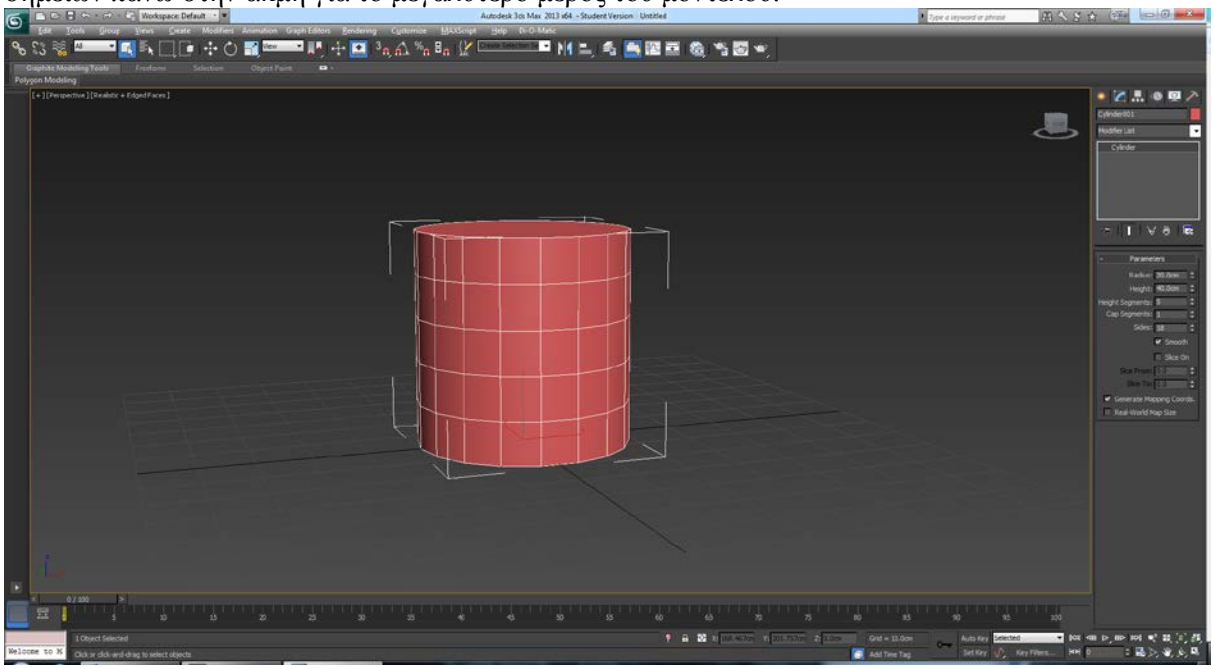

Figure 3.1-1: Δημιουργούμε ένα κύλινδρο

Στη συνέχεια, διαγράφουμε τις δυο παράλληλες πλευρές του αντικειμένου και το «ανοίγουμε» στις άκρες.

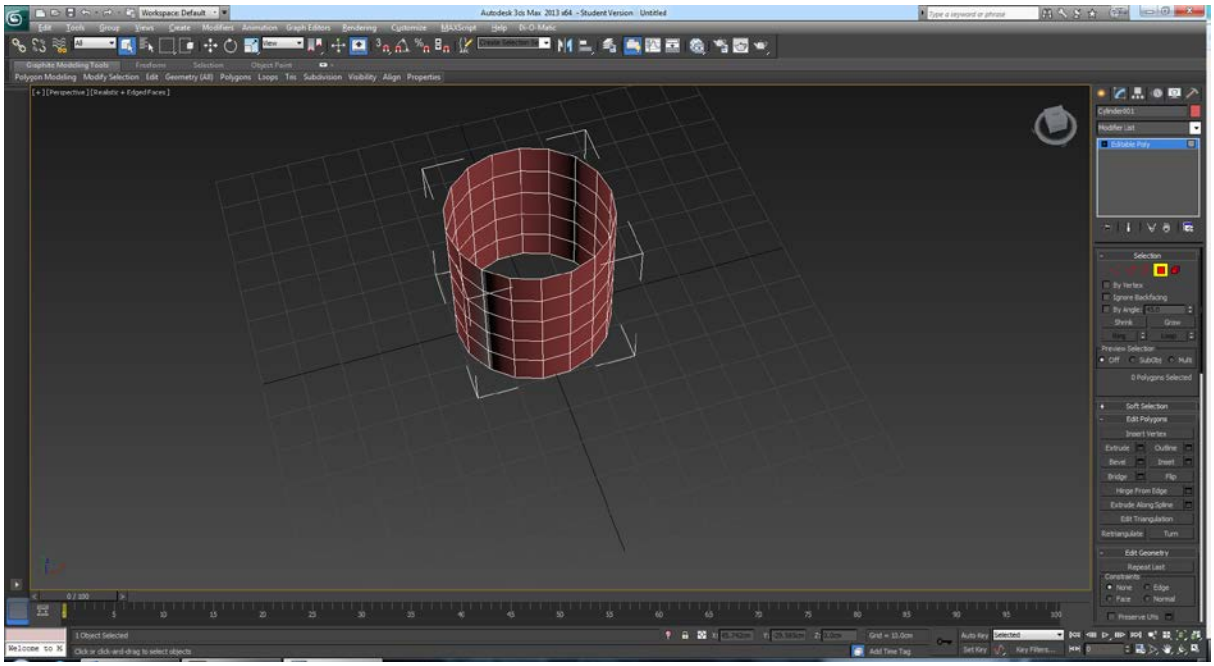

Figure 3.1-2: Διαγράφουμε τα παράλληλα faces

Έπειτα, επιλέγοντας το edge-loop, τον κύκλο που δημιουργούν δηλαδή οι ακμές του σχήματος, με shift+translate αντιγράφουμε και επεκτείνουμε το σχήμα. Κάνοντας scale την επιλογή μπορούμε να δώσουμε τη μορφή που επιθυμούμε. Έτσι δημιουργούμε αρχικά τον βασικό κορμό.

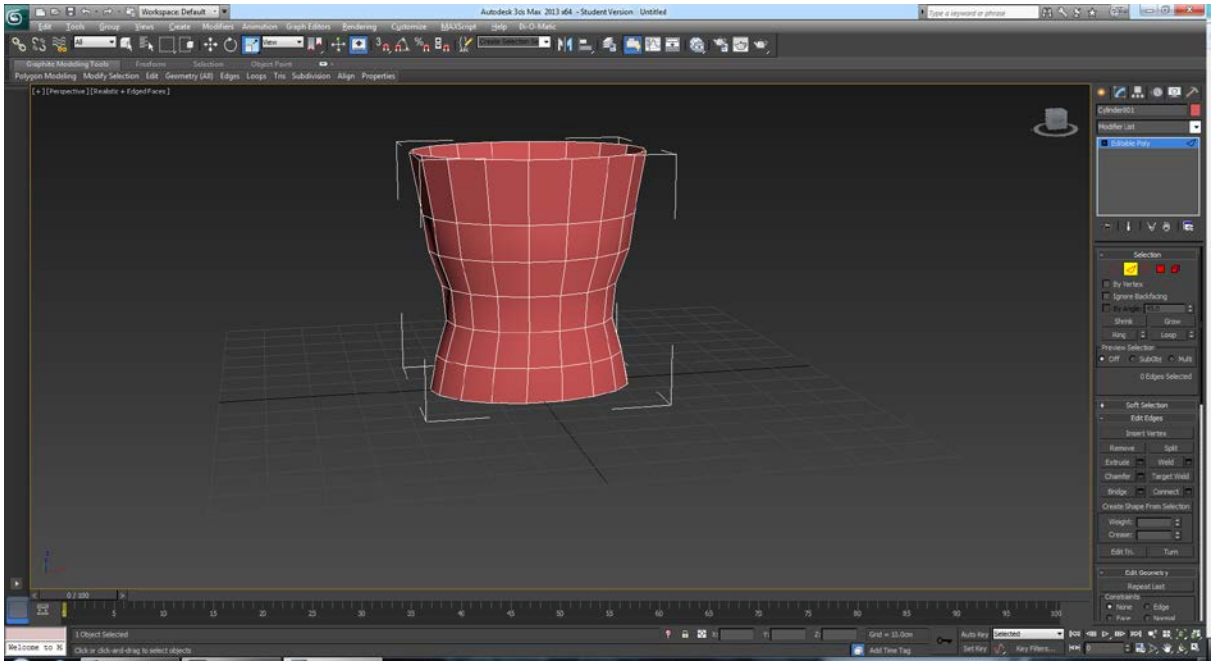

Figure 3.1-3: Κάνουμε Scale τα Edge-Loops

Όταν φτάσουμε σε σημείο που πρέπει να ξεκινήσει διαφορετικό loop, στις αρθρώσεις για παράδειγμα των ώμων ή της βάσης της λεκάνης, επεκτείνουμε μεμονωμένες ακμές έτσι ώστε να δημιουργήσουμε καινούριους βρόγχους, και από εκεί συνεχίζουμε με τον ίδιο τρόπο για να δημιουργήσουμε τα άκρα.

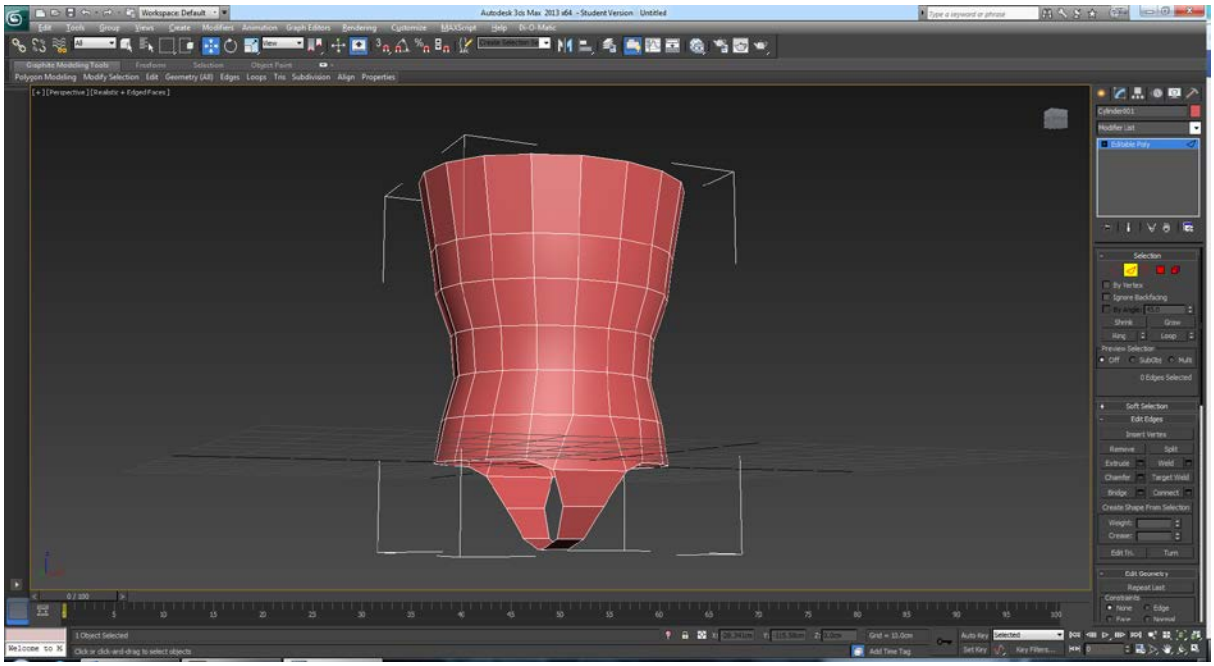

Figure 3.1-4: Δημιουργούμε καινούρια Edge-Loops

Εναλλακτικά, μπορούμε με την ίδια αρχική διαδικασία να φτιάξουμε τα άκρα ξεχωριστά και ύστερα να τα ενώσουμε με τον κορμό, απλά πρέπει να δώσουμε ιδιαίτερη προσοχή στον αριθμό σημείων των ακμών, έτσι ώστε μετά την ένωση να έχουμε ένα ευπαρουσίαστο αποτέλεσμα.

Μπορούμε επίσης, αφού είμαστε ικανοποιημένοι με το βασικό σχήμα του κορμού, να διαγράψουμε τη μισή άκρη και να την αναδημιουργήσουμε κάνοντας χρήση του symmetry modifier.

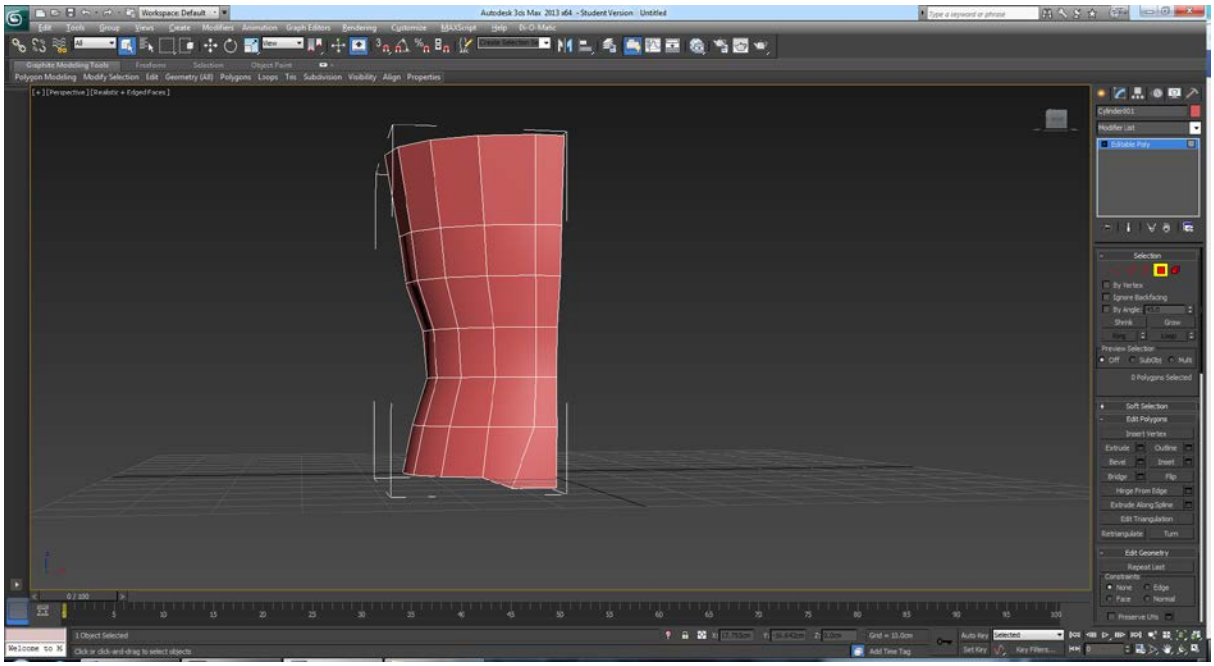

Figure 3.1-5: Διαγράφουμε το μισό mesh

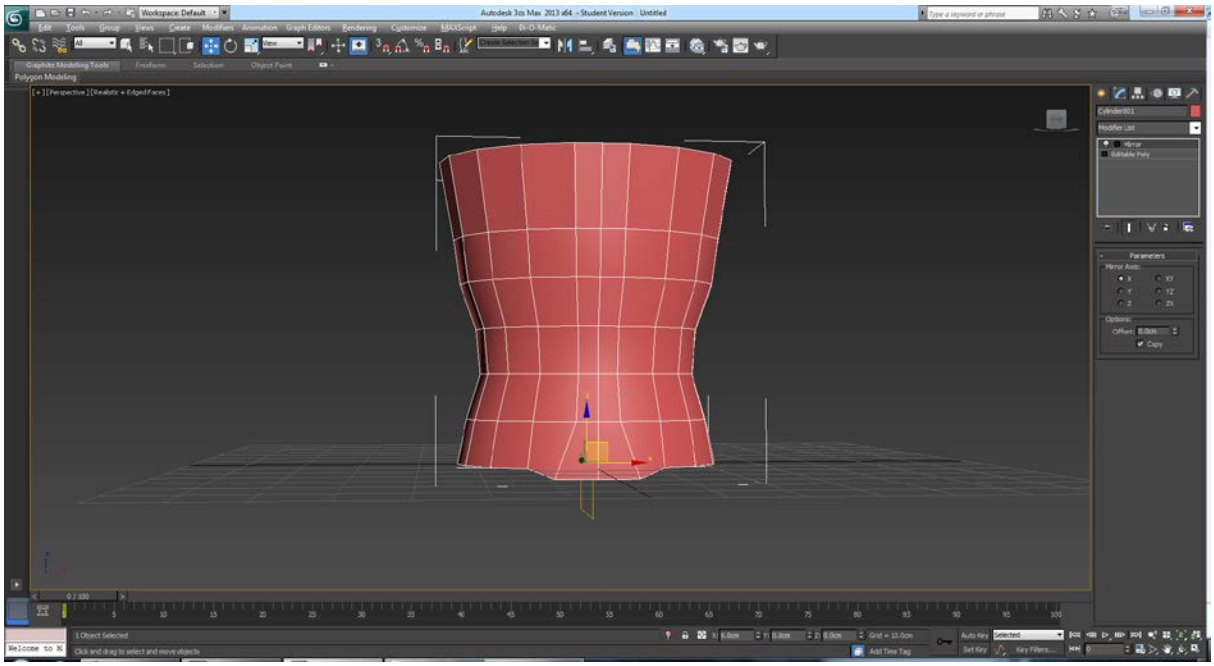

Figure 3.1-6: Εφαρμόζουμε το Mirror Modifier

Με την τεχνική αυτή πετυχαίνουμε ένα απολύτως συμμετρικό μοντέλο, και υποδιπλασιάζουμε το φόρτο εργασίας αφού φτάνει να σχεδιάσουμε μόνο τη μια πλευρά και να αφήσουμε το modifier να δημιουργήσει την άλλη. Πρέπει να σημειωθεί όμως, ότι από τη στιγμή που θα διαγράψουμε το μισό μοντέλο, τα edge-loops που κόβονται χάνουν την ιδιότητα επέκτασης που είχαν, και για αυτό το λόγο προτείνεται να έχουμε πρώτα φτάσει σε ένα ικανοποιητικό επίπεδο.

Στην παρακάτω εικόνα φαίνεται το low-poly σώμα. Οι παλάμες και τα δάκτυλα, όπως και τα πόδια, δημιουργήθηκαν ξεχωριστά και έγιναν attach πάνω στο υπόλοιπο μοντέλο.

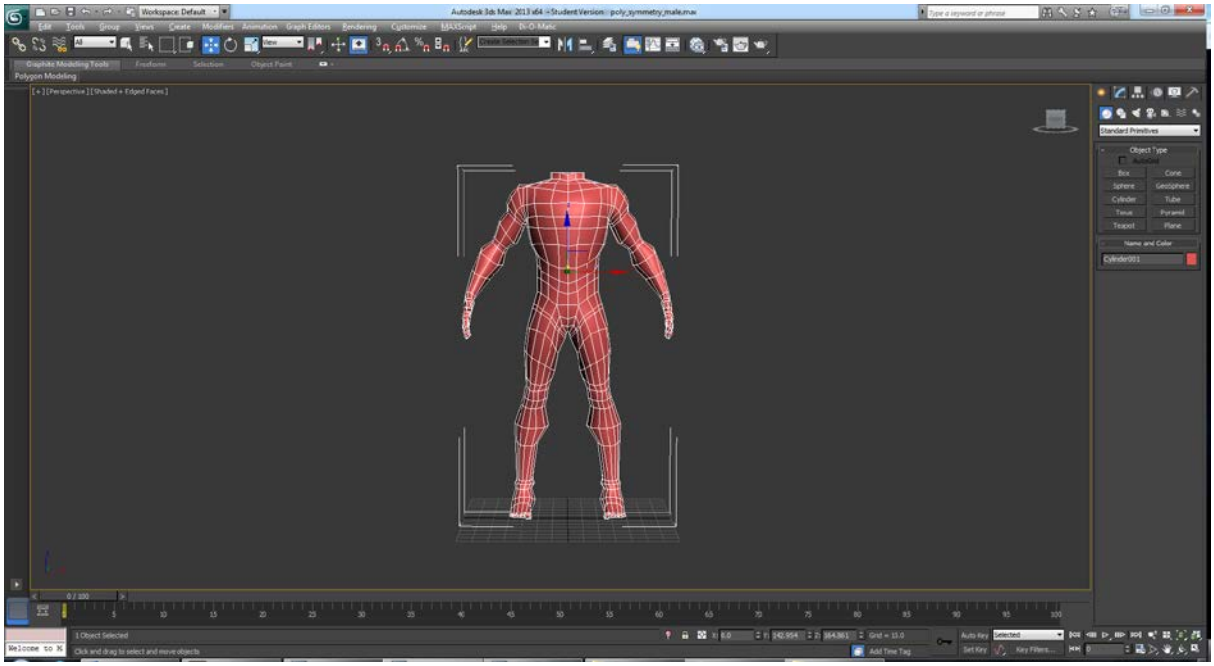

Figure 3.1-7: Ολόκληρο το σώμα

Στη συνέχεια θα δούμε τη διαδικασία μοντελοποίησης του κεφαλιού με τη τεχνική poly-by-poly modelling.

3.1.3 Poly-by-Poly Modelling:

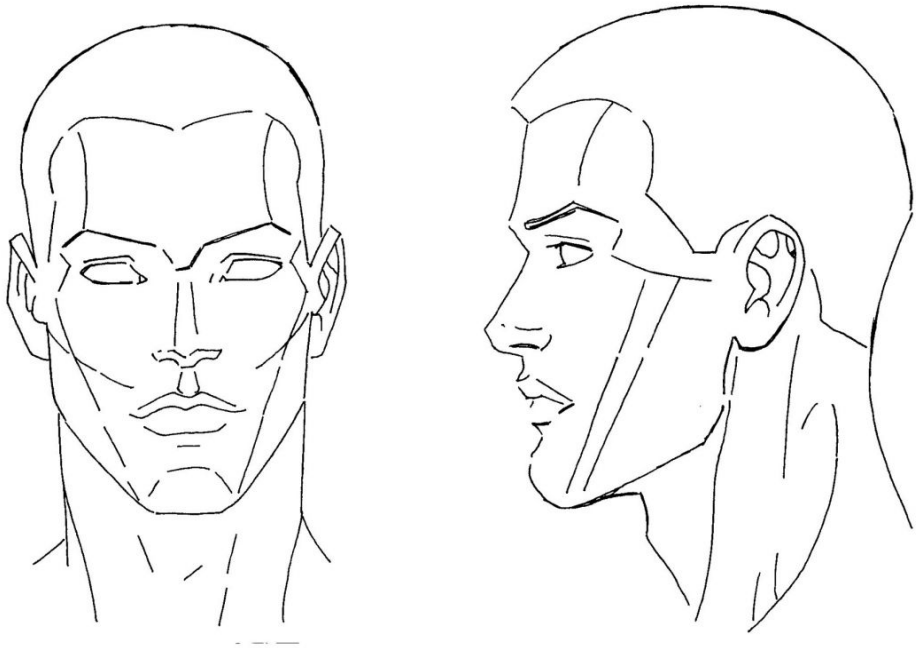

Figure 3.1-8: Εικόνα αναφοράς

3.1.3.1. Εφαρμογή:

Όπως και για το σώμα, θα δημιουργήσουμε μόνο τη μια πλευρά και θα εφαρμόσουμε το symmetry modifier για την άλλη.

Ξεκινάμε δημιουργώντας ένα plane στην περιοχή γύρω από το μάτι και το τοποθετούμε ώστε να ακολουθεί τη γραμμή του προσώπου.

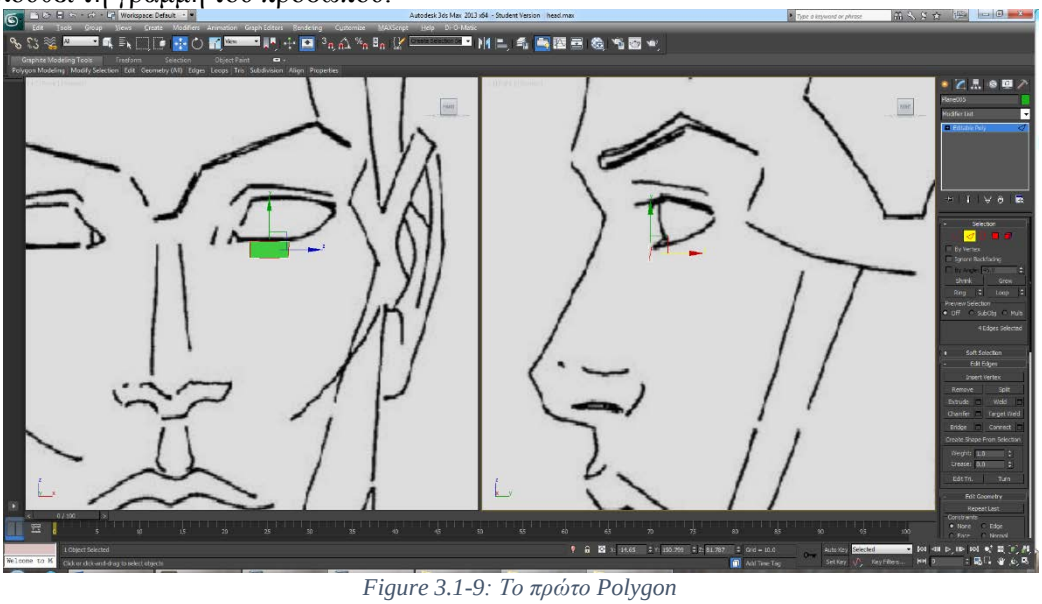

Figure 3.1-9: Το πρώτο Polygon

Στη συνέχεια, αφού το μετατρέψουμε σε editable poly για να είναι τροποποιήσιμο, με shift drag επεκτείνουμε μια πλευρά και την τοποθετούμε και στα δύο views καταλλήλως.

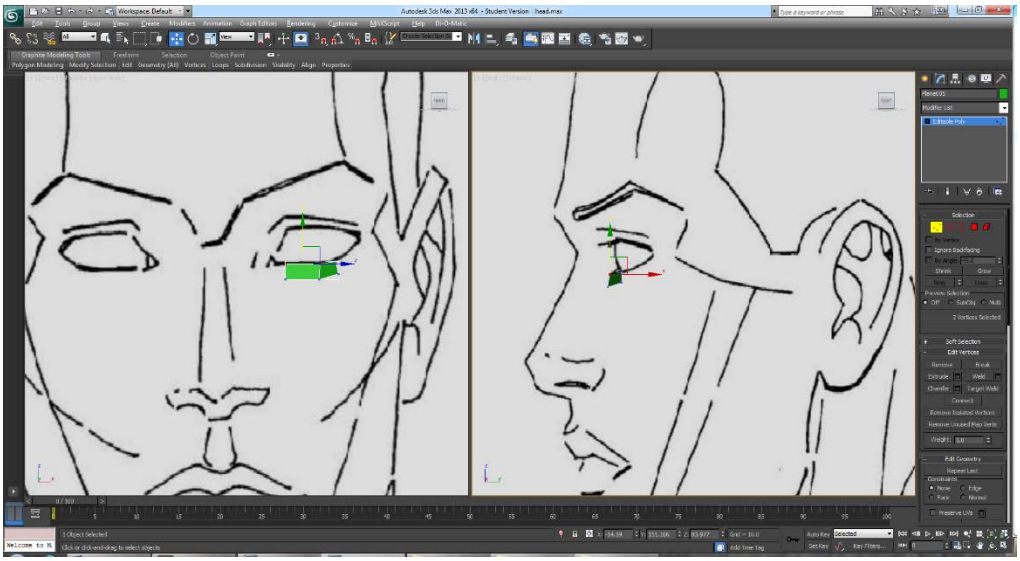

Figure 3.1-10: Επεκτείνουμε τις ακμές

Συνεχίζουμε με την ίδια λογική και απλά ενώνουμε τα σημεία όπου χρειάζεται.

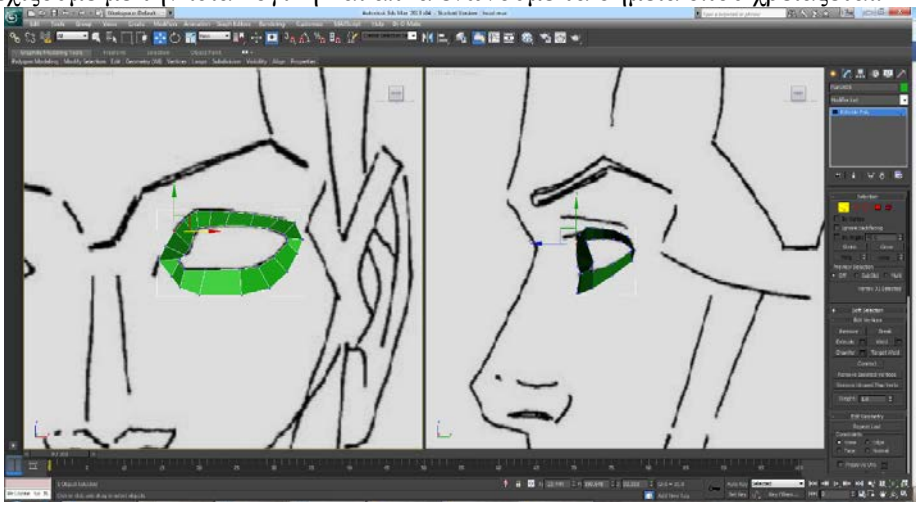

Figure 3.1-11: Δημιουργούμε Edge-Loop

Έχουμε δημιουργήσει εδώ ένα edge-loop γύρω από την τρύπα που αφήνει το μάτι στο μοντέλο. Συνεχίζοντας επεκτείνουμε το edge-loop αυτό ώστε να επιτύχουμε ένα οργανικό μοντέλο που θα μας επιτρέψει να φτιάξουμε φυσικά animations.

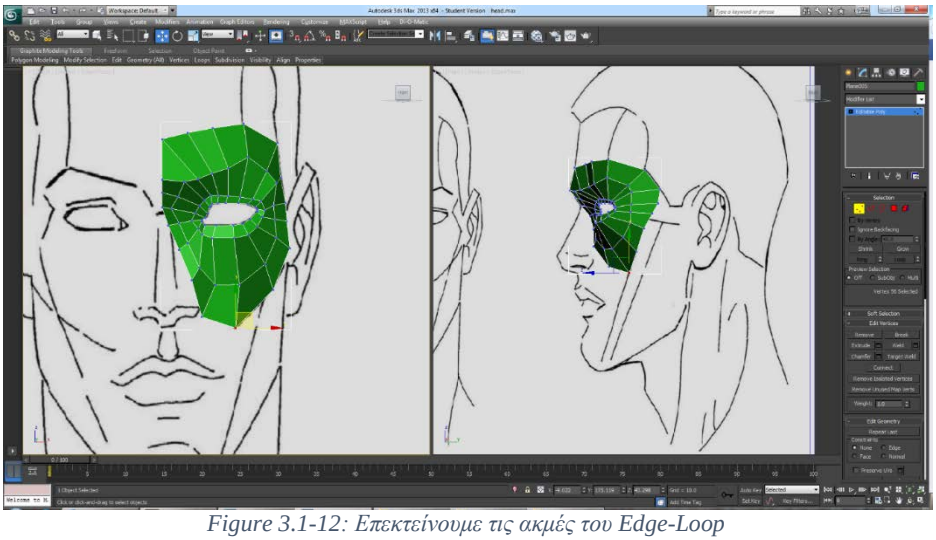

Figure 3.1-12: Επεκτείνουμε τις ακμές του Edge-Loop

Σε αυτό το σημείο, επειδή έφτασα μέχρι τη μέση του προσώπου, επέλεξα να ενεργοποιήσω το symmetry modifier.

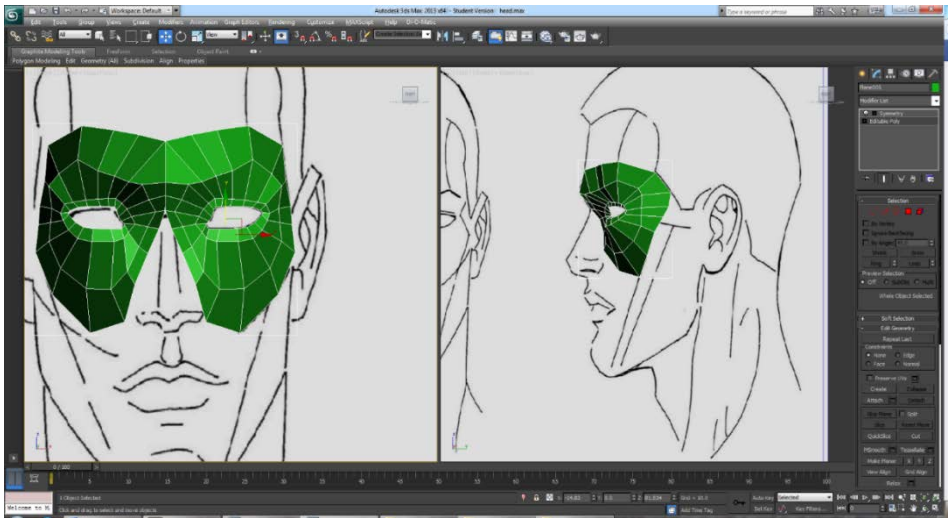

Figure 3.1-13: Mirror Modifier

Και συνεχίζουμε όπως και πριν μέχρι να φτάσουμε σε ένα ικανοποιητικό αποτέλεσμα. Στο σημείο αυτό μας ενδιαφέρει να δημιουργήσουμε το γενικό σχήμα του προσώπου και να έχουμε καλή τοπολογία, δηλαδή αρκετά πολύγωνα στα σημεία που χρειαζόμαστε λεπτομέρεια, και να μην υπάρχουν κομμένα edge-loops.

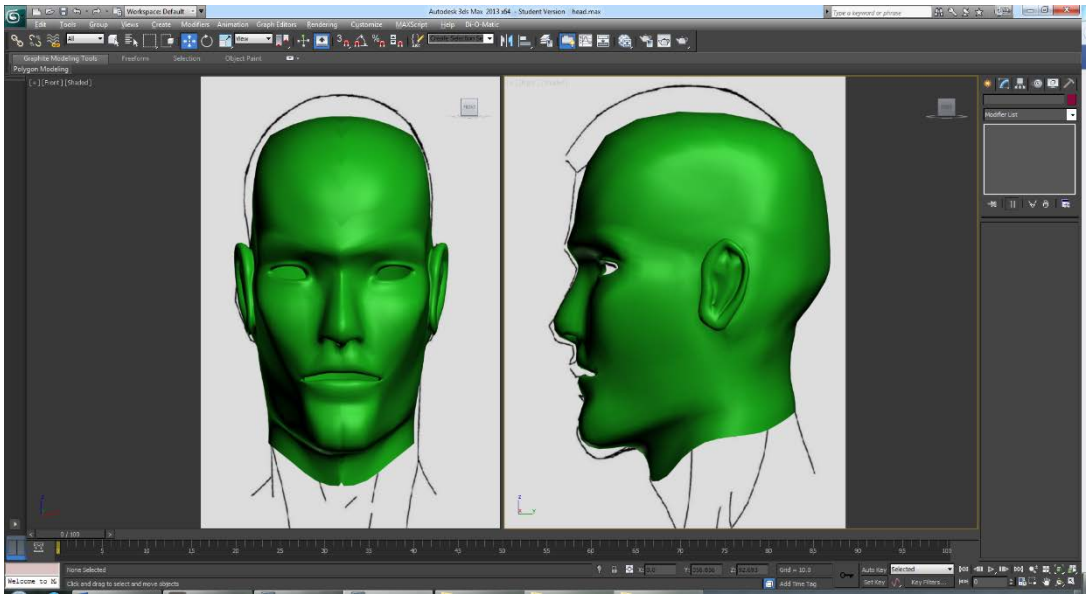

Figure 3.1-14: Ολοκληρωμένο κεφάλι με Smoothing Groups

3.1.4 Ολοκληρωμένο μοντέλο:

Αφού έχουμε τελειώσει με τα επιμέρους στοιχεία του μοντέλου τα συνενώνουμε σε ένα mesh.

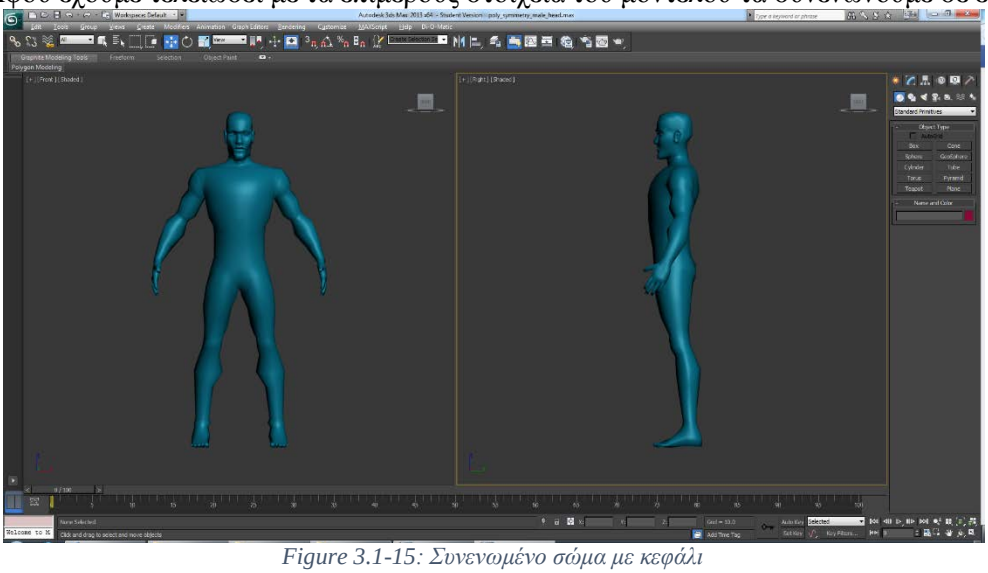

Figure 3.1-15: Συνενωμένο σώμα με κεφάλι

Στη συνέχεια, περνάμε το μοντέλο στο ZBrush για να το φέρουμε σε ακόμα καλύτερη μορφή, και να του δώσουμε λίγο περισσότερα πολύγωνα, κρατώντας το ακόμα σε αρκετά χαμηλά επίπεδα ώστε να μην επηρεαστεί η επίδοση. Αυτό όμως θα αναπτυχθεί στο επόμενο κεφάλαιο, οπότε θα μελετήσουμε τη πορεία που ακολουθήσαμε αφού τροποποιήσαμε το μοντέλο και το επιστρέψαμε στο 3ds Max.

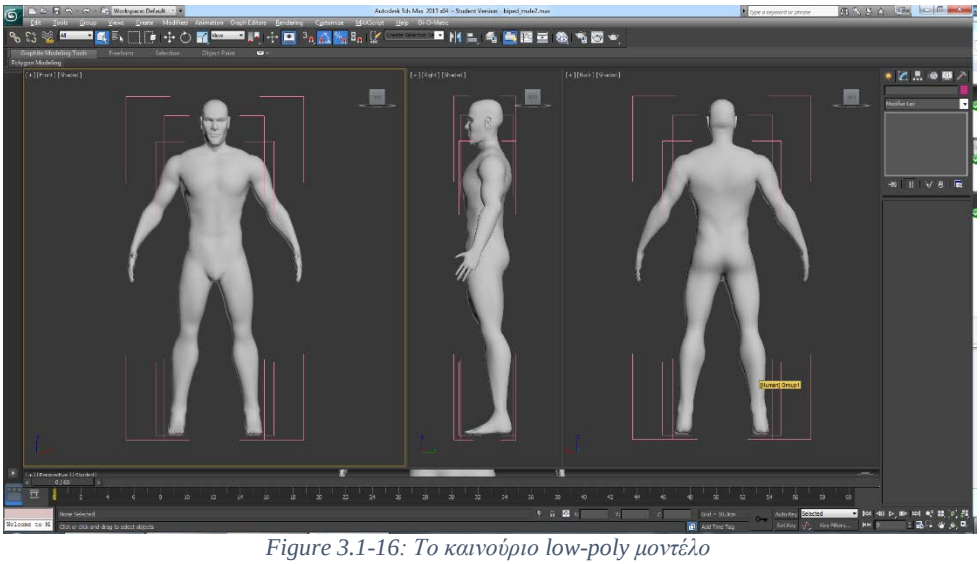

Figure 3.1-16: Το καινούριο low-poly μοντέλο

Η έκδοση αυτή του μοντέλου διαθέτει την τελική τοπολογία, σωστά edge-loops, και αριθμό πολυγώνων ικανό να αναδείξει την επιθυμητή λεπτομέρεια κάνοντας χρήση των texture και normal maps που δημιουργούμε μέσω ZBrush. Είμαστε δηλαδή σε κατάλληλο στάδιο για να ξεκινήσουμε το UVW Unwrapping που θα μας επιτρέψει να ντύσουμε το χαρακτήρα μας με χρώμα, και να δώσουμε στο τελικό αποτέλεσμα τη «ψευδαίσθηση» της λεπτομέρειας.

Ένα τελευταίο πράγμα που πρέπει να γίνει, είναι να δημιουργήσουμε μάτια, και το εσωτερικό του στόματος. Το στόμα βρέθηκε ως έτοιμο μοντέλο από το animium.com, και απλά τοποθετήθηκε στη σωστή θέση. Τα μάτια είναι απλά μια mirrored σφαίρα.

Στο στάδιο αυτό έχουμε δημιουργήσει και το ρουχισμό του χαρακτήρα καθώς επίσης και μάτια, αλλά όλες οι διαδικασίες που ακολουθούμε για το σώμα εφαρμόζονται ακριβώς με τον ίδιο τρόπο και στα υπόλοιπα αντικείμενα οπότε είναι περιττό να τα αναλύσω ξεχωριστά.

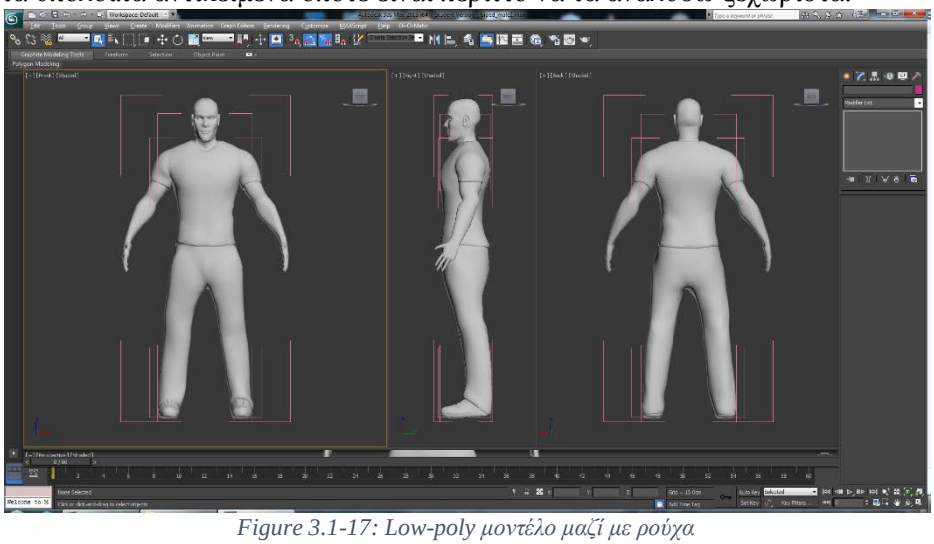

Figure 3.1-17: Low-poly μοντέλο μαζί με ρούχα

3.1.5 Δημιουργία μαλλιών με planes και object-painting:

Ένας συνήθης τρόπος δημιουργίας μαλλιών για video games είναι με τη χρήση πολλών, ημιδιάφανων planes, τοποθετημένων πάνω στο κεφάλι. Για να δημιουργήσουμε το σύνολο των Planes χρειαζόμαστε μόνο εργαλεία που βρίσκονται ήδη μέσα στο 3ds Max και ένα σημείο αναφοράς, στη συγκεκριμένη περίπτωση, έχω πάρει ένα αντίγραφο του κεφαλιού του χαρακτήρα.

Ξεκινάμε με ένα plane χωρισμένο σε 4 τμήματα το οποίο μετατρέπουμε σε Editable Poly και του δίνουμε ένα καταλληλότερο σχήμα.

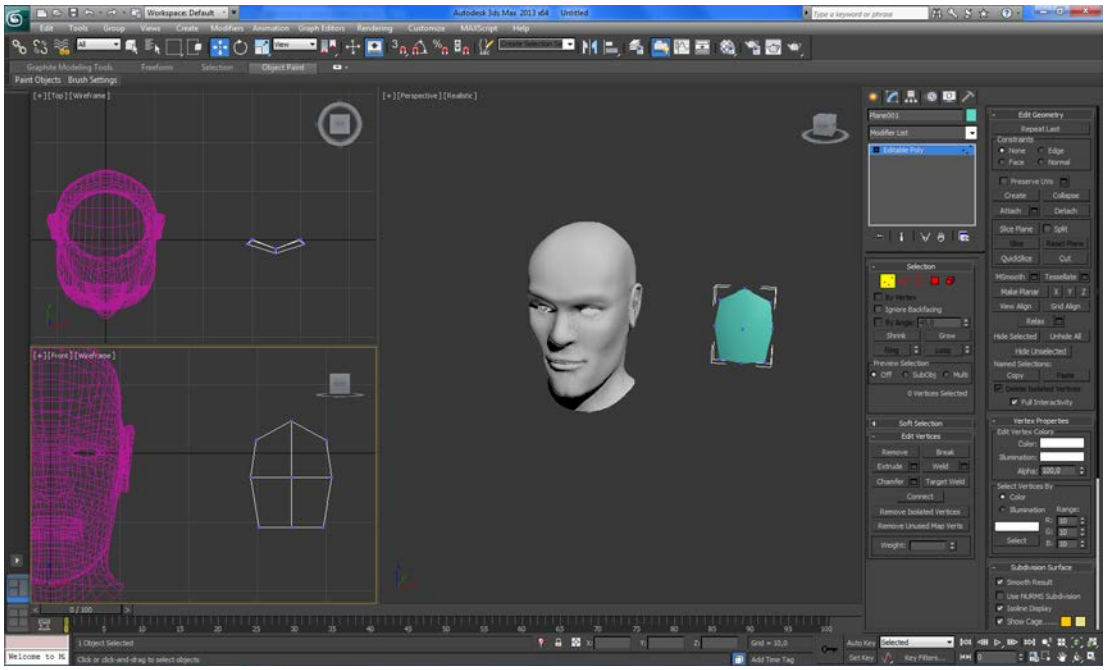

Figure 3.1-18: Δημιουργούμε ένα plane

Το συγκεκριμένο plane αργότερα θα γίνει unwrap ώστε να ελεγχθεί η θέση του texture πάνω του, και θα του εφαρμοστεί μια εικόνα δέσμης μαλλιών, με διαφάνεια στα κενά σημεία.

Για την ώρα, θέλουμε αντίγραφα αυτού του plane που να ακολουθούν το σχήμα του κεφαλιού. Για να το πετύχουμε αυτό, χρησιμοποιούμε το εργαλείο Object Paint του 3ds Max, που μας επιτρέπει να τοποθετούμε αντίγραφα των επιλεγμένων μοντέλων πάνω σε μια επιφάνεια με πινέλο, σαν να ζωγραφίζουμε. Από τις ρυθμίσεις του Object Paint επιλέγουμε Edit Object List και Pick και επιλέγουμε το plane που έχουμε δημιουργήσει. Στη συνέχεια επιλέγουμε Paint και αρχίζουμε να «βάφουμε» πάνω στο κεφάλι.

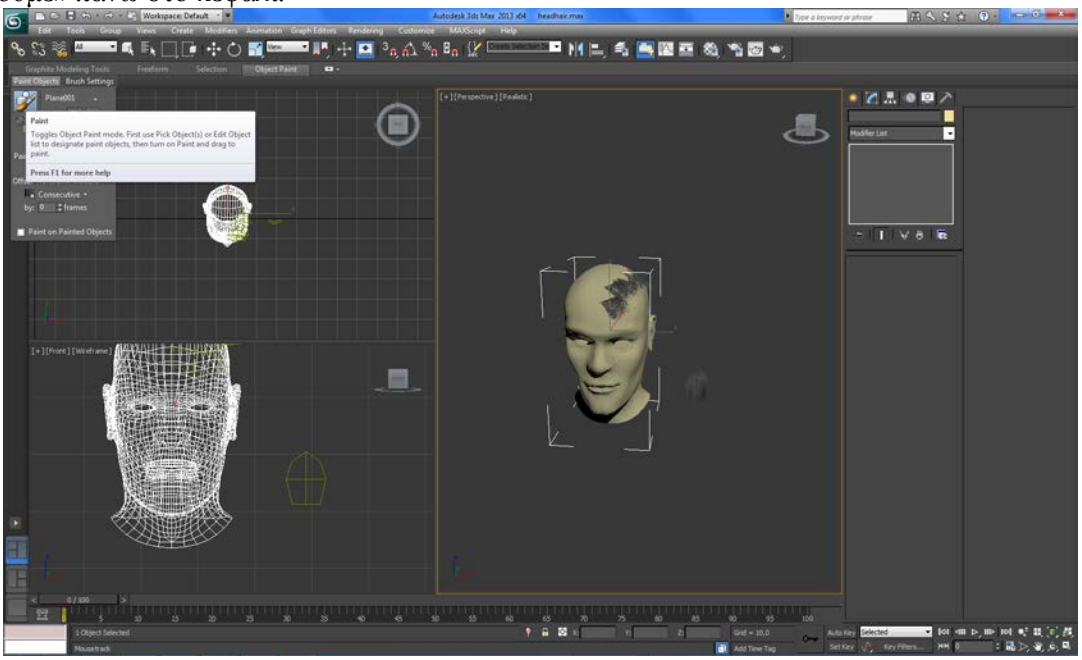

Figure 3.1-19: "Βάφουμε" τα planes πάνω στο κεφάλι

Τέλος, από τις ρυθμίσεις του Object Paint μπορούμε να ορίσουμε την περιστροφή των αντιγράφων, το μέγεθος τους τη θέση τους, είτε με συγκεκριμένο τρόπο είτε με παραμέτρους τυχαιότητας (random).

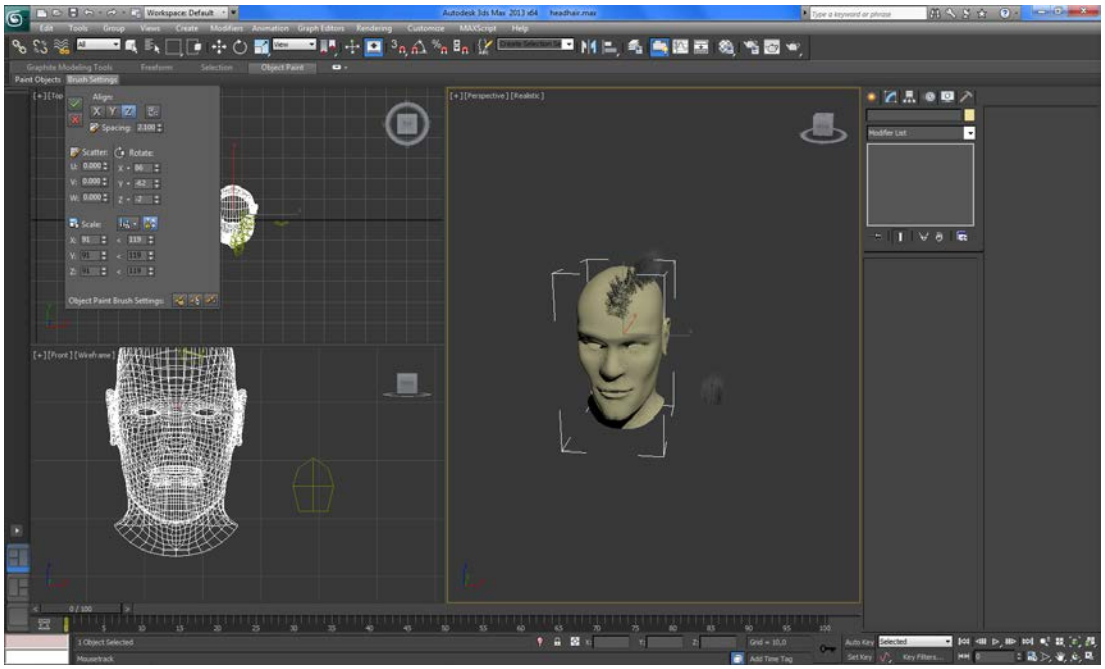

Figure 3.1-20: Ευθυγραμμίζουμε τα planes

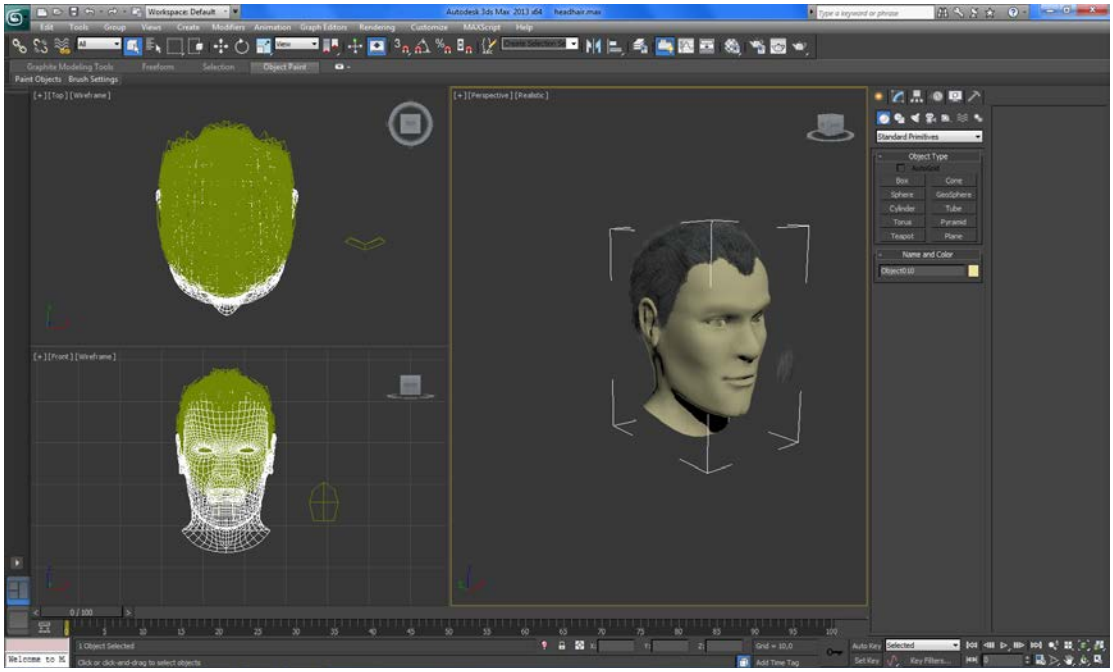

Figure 3.1-21: Το τελικό αποτέλεσμα των μαλλιών μαζί με alpha maps.

Να σημειώσουμε ότι λόγω διαφορετικής λειτουργίας των alpha maps στην Unity, τα μαλλιά μέσα στο παιχνίδι θα φαίνονται διαφορετικά.

3.1.6 UVW Mapping & Unwrapping:

3.1.6.1 Εφαρμογή:

Αρχικά, εφαρμόζουμε στο μοντέλο μας το Unwrap UVW Modifier. Αυτό μας επιτρέπει να επιλέξουμε «ραφές», δηλαδή ακμές στις οποίες θα χωριστεί η επιφάνεια του mesh, ώστε να ισοπεδωθεί και να χαρτογραφηθεί σε δύο διαστάσεις. Οι κομμένες ακμές εμφανίζονται πράσινες στο 3ds Max, ενώ οι εξεταζόμενες ακμές, αυτές δηλαδή που έχουμε ορίσει εμείς, αλλά πριν κοπούν, φαίνονται μπλε. Αν το 3ds Max έχει προσπαθήσει να δημιουργήσει τις δικές του ακμές, οι οποίες δεν είναι ιδιαίτερα σωστές, δεν έχουμε παρά να κάνουμε collapse to modifier, να πατήσουμε More.. στη

καρτέλα Utilities στα δεξιά και έπειτα UVW Remove και UVW στις επιπλέον επιλογές που θα εμφανιστούν.

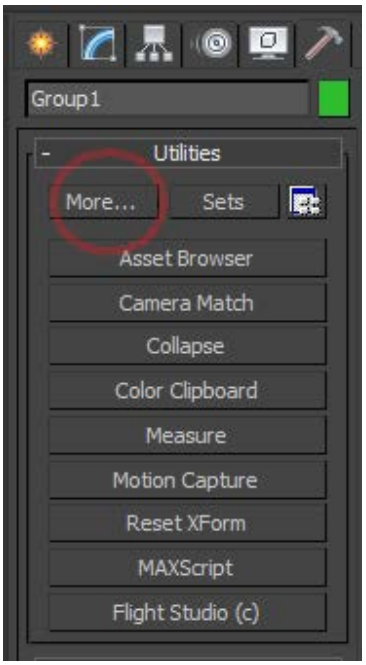

Figure 3.1-22: Utilities

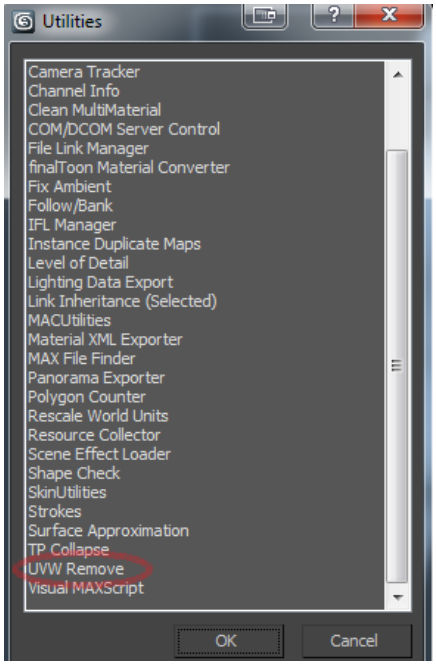

Figure 3.1-23: More

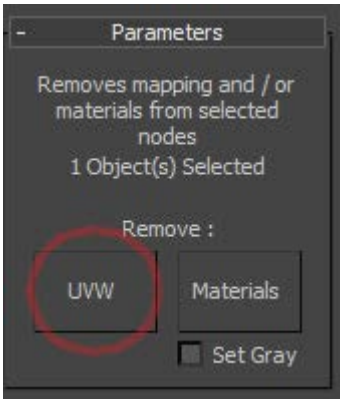

Figure 3.1-24: UVW Remove Parameters

Στη συνέχεια, αν κάναμε collapse τα modifiers, προσθέτουμε πάλι το Unwrap UVW και επιλέγουμε τις δικές μας ραφές με τη μέθοδο point-to-point seam.

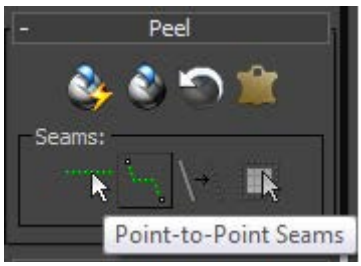

Figure 3.1-25: Το κουμπί Point-to-point Seams

Η μέθοδος αυτή μας επιτρέπει να επιλέγουμε σημεία πάνω στο μοντέλο, και το εργαλείο θα βρίσκει τη συντομότερη διαδρομή μεταξύ των σημείων αυτών, δημιουργώντας έτσι μια αλληλουχία ακμών.

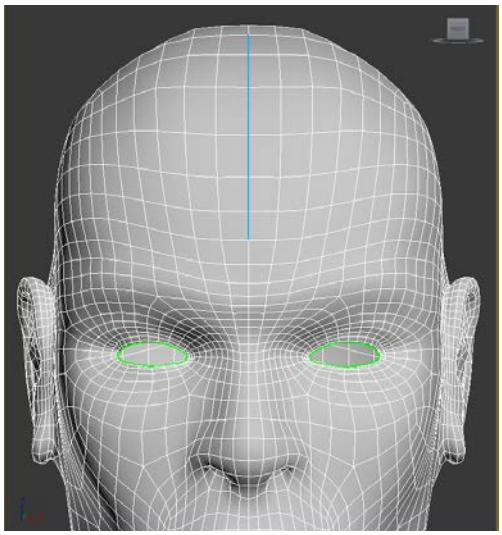

Figure 3.1-26: Επιλέγοντας seams πάνω στο μοντέλο

Οι ακμές αυτές εμφανίζονται μπλε, μέχρι να οριστικοποιήσουμε την απόφαση μας και τις δηλώσουμε ως seams, μπαίνοντας σε Peel Mode, οπότε και θα γίνουν πράσινες. Ακολουθεί το μοντέλο με τις ραφές (seams) στις οποίες κατέληξα:

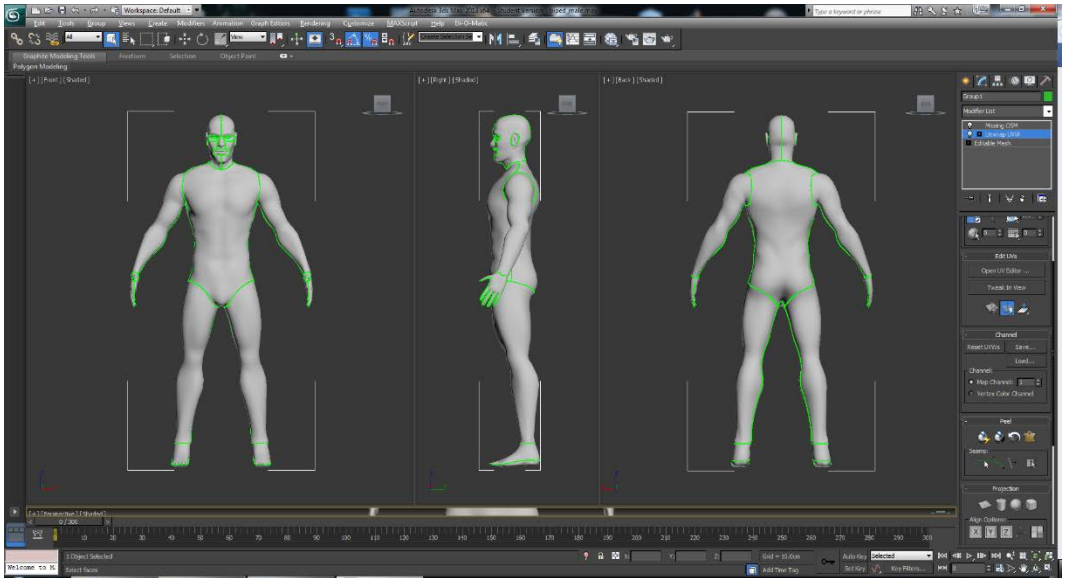

Figure 3.1-27: Το μοντέλο μαζί με τα οριστικοποιημένα seams

Από αυτό το σημείο το μόνο που μας μένει είναι να ορίσουμε πώς κάθε «νησί», κάθε ξεχωριστό κομμάτι δηλαδή, θα ξεδιπλωθεί σε δύο διαστάσεις. Αυτό θα γίνει με το εργαλείο Pelt Map.

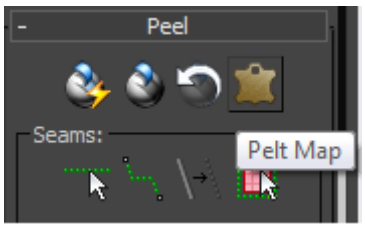

Figure 3.1-28: Το κουμπί του Pelt Map

Το εργαλείο αυτό δημιουργεί ένα «stretcher», ένα σύνολο σημείων σε κύκλο γύρω από το mesh μας, που θα χρησιμοποιήσει για να «τεντώσει» τα σημεία του πλέγματος, βεβαιώνοντας ότι κανένα σημείο δεν θα υπερκαλύπτει κάποιο άλλο.

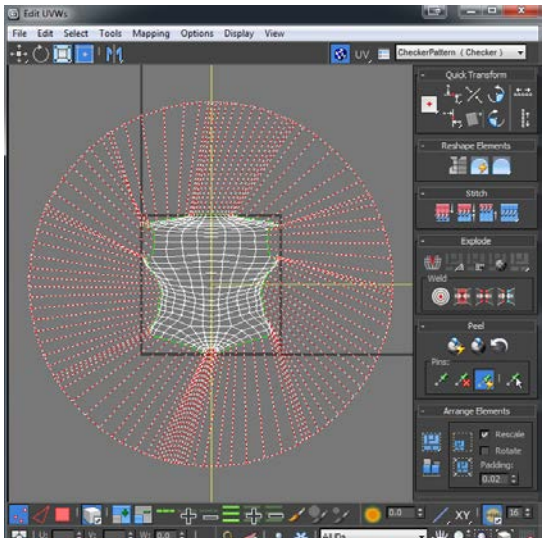

Figure 3.1-29: Η επιφάνεια μαζί με τον stretcher

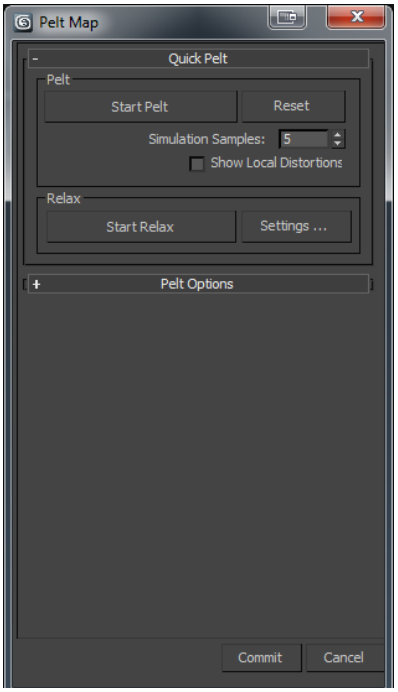

Figure 3.1-30: Το παράθυρο του Pelt Map

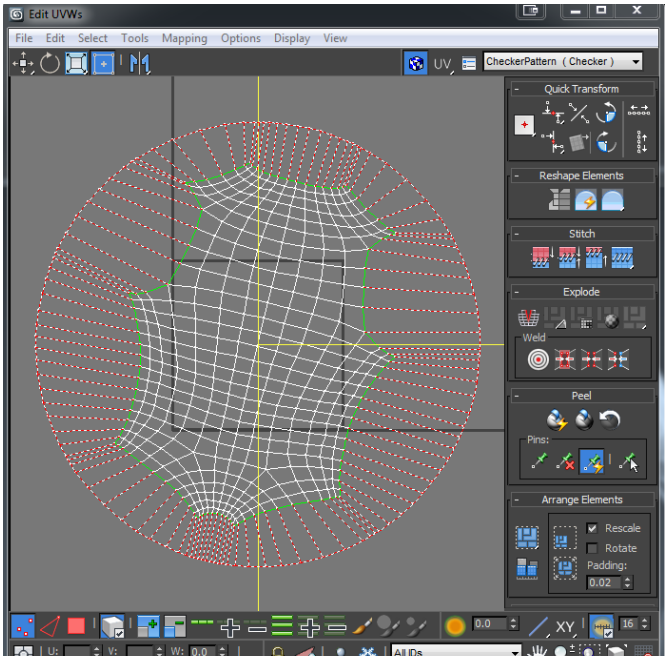

Figure 3.1-31: Το αποτέλεσμα του Pelt

Αφού περάσουμε από αυτή τη διαδικασία για κάθε κομμάτι, τα τοποθετούμε πάνω σε ένα τετράγωνο (packing) για να ορίσουμε ποια θα είναι η διευθέτηση (layout) των σημείων στο δισδιάστατο χώρο, και έχουμε αυτό το αποτέλεσμα:

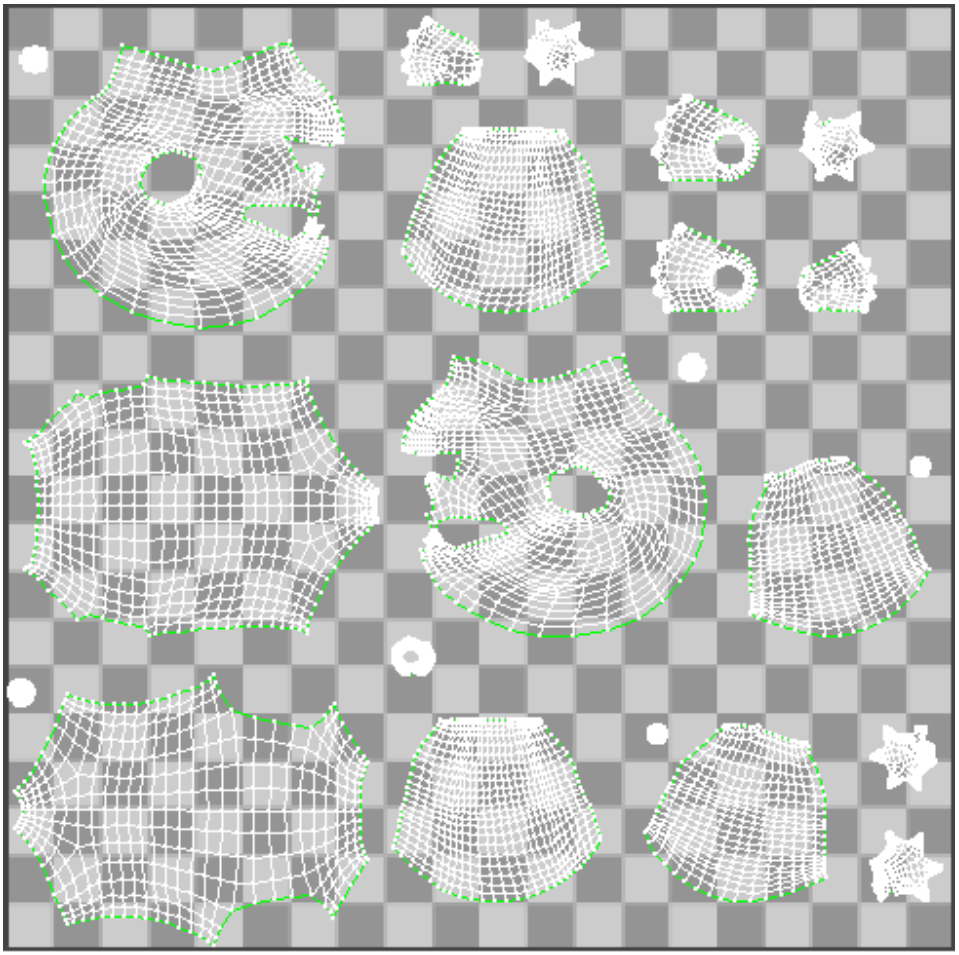

Figure 3.1-32: Το UVW Map

Η ίδια διαδικασία χρησιμοποιήθηκε για τα ρούχα, τα μαλλιά και τα μάτια, με τα layout τους χωρισμένα σε τρία group, ένα για τη μπλούζα και το παντελόνι, ένα για τα μαλλιά και ένα για τα παπούτσια και τα μάτια. Ο διαχωρισμός αυτός έγινε για να μπορούμε να έχουμε διαφορετικό υλικό στα τρία τμήματα αυτά, δηλαδή ένα που δε θα γυαλίζει για το ύφασμα, ένα που θα γυαλίζει για το πλαστικό και τα μάτια και ένα που θα υποστηρίζει διαφάνεια για τα μαλλιά. Τα δόντια, ούλα και γλώσσα, όντας ξεχωριστά αντικείμενα, δεν χρειάστηκαν unwrap καθώς έχουν ανά τμήμα ομοιόμορφο χρώμα. Ακολουθούν τα UVW Maps των υπολοίπων:

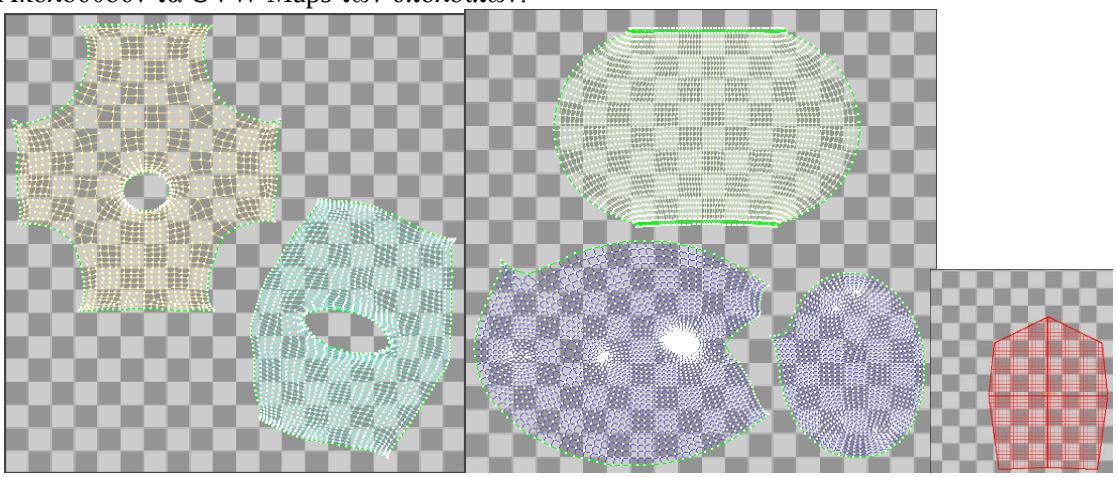

Figure 3.1-33: Μπλούζα και παντελόνι (αριστερά), μάτι και παπούτσι (μέση), μαλλιά (δεξιά)

Το παρόν μοντέλο, μαζί με το Unwrap UVW modifier θα εξαχθεί ως αρχείο με κατάληξη .obj και θα εισαχθεί στο ZBrush ώστε να δημιουργηθούν οι υφές.

3.1.7 Δημιουργία σκελετού (Rigging):

3.1.7.1 Εφαρμογή:

Το 3ds Max μας παρέχει ένα έτοιμο σκελετικό σύστημα με όνομα Biped. Το σύστημα αυτό είναι παραμετροποιήσιμο (μπορούμε να επιλέξουμε αριθμό σπονδύλων, αριθμό διαχειριστών για τα δάκτυλα, κ.α.), κάνει χρήση συστημάτων IK για την κίνηση των άκρων, διαθέτει ελεγκτές για την εύκολη δημιουργία walk-cycle (περπατήματος) και είναι συμβατό με το διαχειριστή animation του Unity, πράγμα που «έσπρωξε» την επιλογή λογισμικού για τη δημιουργία του rig και των animation σε 3ds Max αντί του Blender.

Πέρα από τις βασικές λειτουργίες του 3ds Max, χρησιμοποιήθηκε το BonesPro, ένα plugin του Max, δημιουργημένο από την εταιρία **3D-IO** Games & Video Production GmbH. Το plugin αυτό βοήθησε σημαντικά στη βελτιστοποίηση του skinning και στην εξομάλυνση της γενικής διαδικασίας, αν και για λόγους συμβατότητας, χρειάστηκε το skinning του BonesPro να μεταφερθεί στο υποδεέστερο Skin modifier του 3ds Max. Αξίζει να σημειωθεί ότι το BonesPro παρέχει δυνατότητα χρήσης αντικειμένων-οστών για την εξομοίωση μυϊκών ομάδων, όπως αναφέρθηκε παραπάνω.

Ξεκινώντας, από τη καρτέλα Create->Systems επιλέγουμε Biped, το τοποθετούμε στη σκηνή, και το εφαρμόζουμε στο μοντέλο μας.

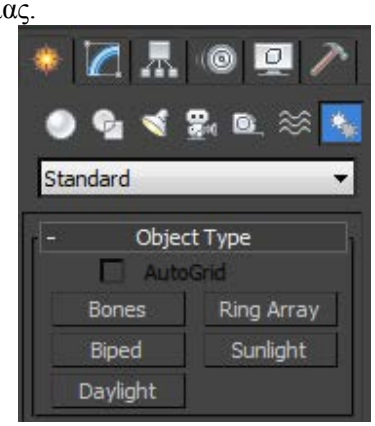

Figure 3.1-34: Η καρτέλα Systems

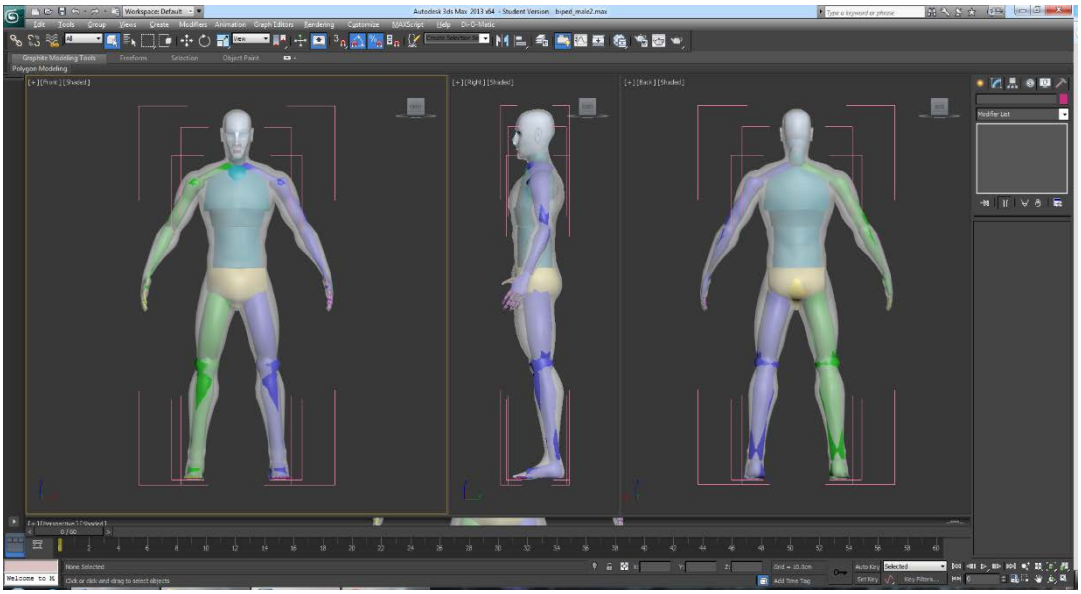

Figure 3.1-35: Μοντέλο μαζί με Skeletal Rig

Στο επόμενο βήμα, αφού έχουμε εγκαταστήσει το BonesPro, το προσθέτουμε ως modifier στο μοντέλο μας, και του ορίζουμε ποια οστά θέλουμε να επηρεάζουν τα σημεία του μοντέλου, στην περίπτωση μας όλα. Το modifier αυτό προορίζεται για τη δημιουργία φυσικού, ρεαλιστικού skin που

θυμίζει τις πραγματικές κινητικές παραμορφώσεις ενός σώματος. Για να το πετύχει αυτό, δίνει μια ελαστική περιοχή επιρροής σε κάθε οστό, εργαλεία επεξεργασίας των περιοχών αυτών, αλλά και επιλογή προσαρμογής της επιρροής με weight painting, το «βάψιμο» βαρών πάνω στην επιφάνεια με βάση ένα συγκεκριμένο χρωματικό κώδικα (μπλε-0%, κόκκινο-100%).

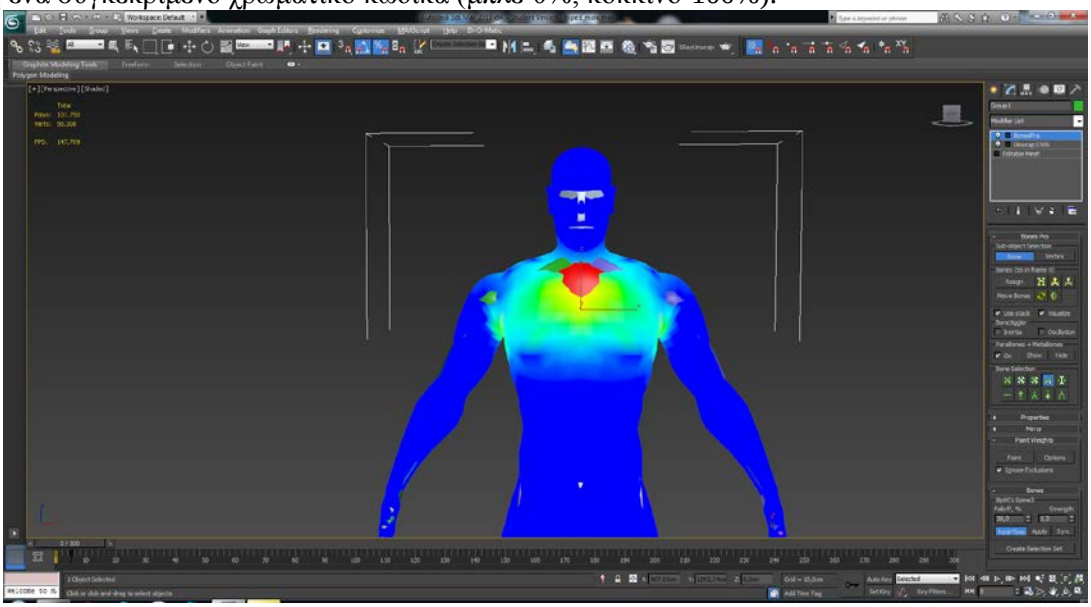

Figure 3.1-36: Ο χρωματικός κώδικας

Όσο εργαζόμαστε πάνω στα βάρη των σημείων, μπορούμε να κινούμε οστά και αρθρώσεις και να βλέπουμε σε πραγματικό χρόνο την επίδραση των οστών πάνω στο μοντέλο.

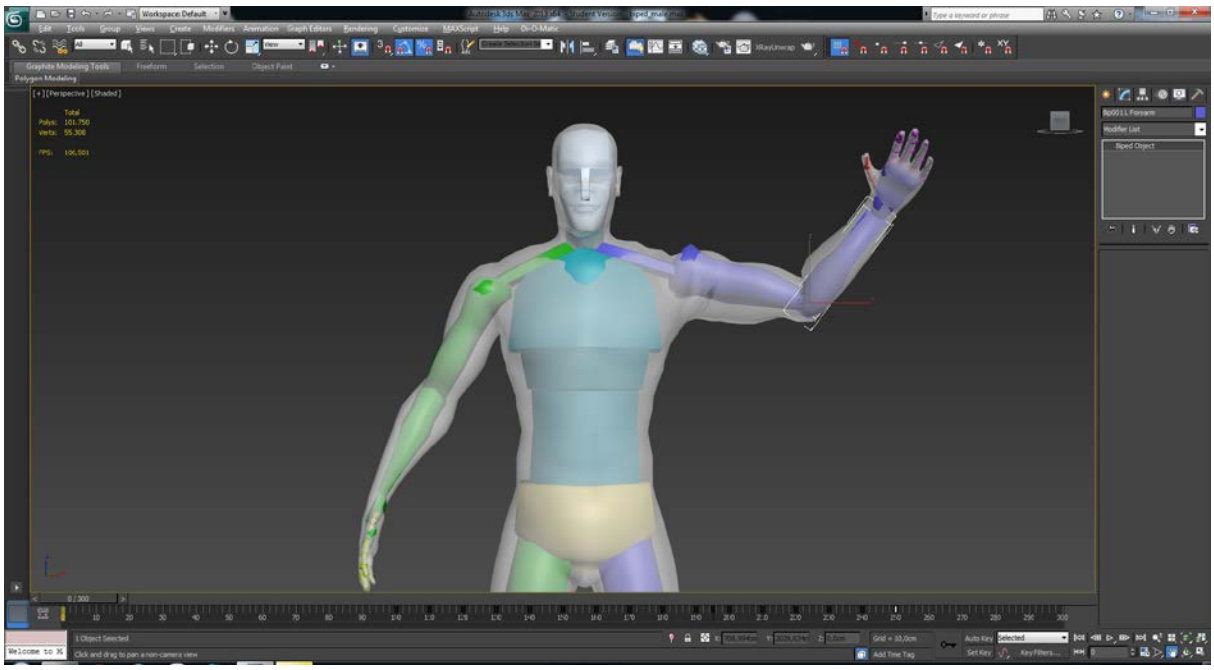

Figure 3.1-37: Κίνηση αριστερού χεριού

Παρατηρούμε για παράδειγμα ότι υπάρχει μια αφύσικη παραμόρφωση στα δάχτυλα. Αυτό συμβαίνει γιατί τα οστά που αντιστοιχούν στο μεσαίο δάκτυλο έχουν βάρος σε σημεία του δείκτη.

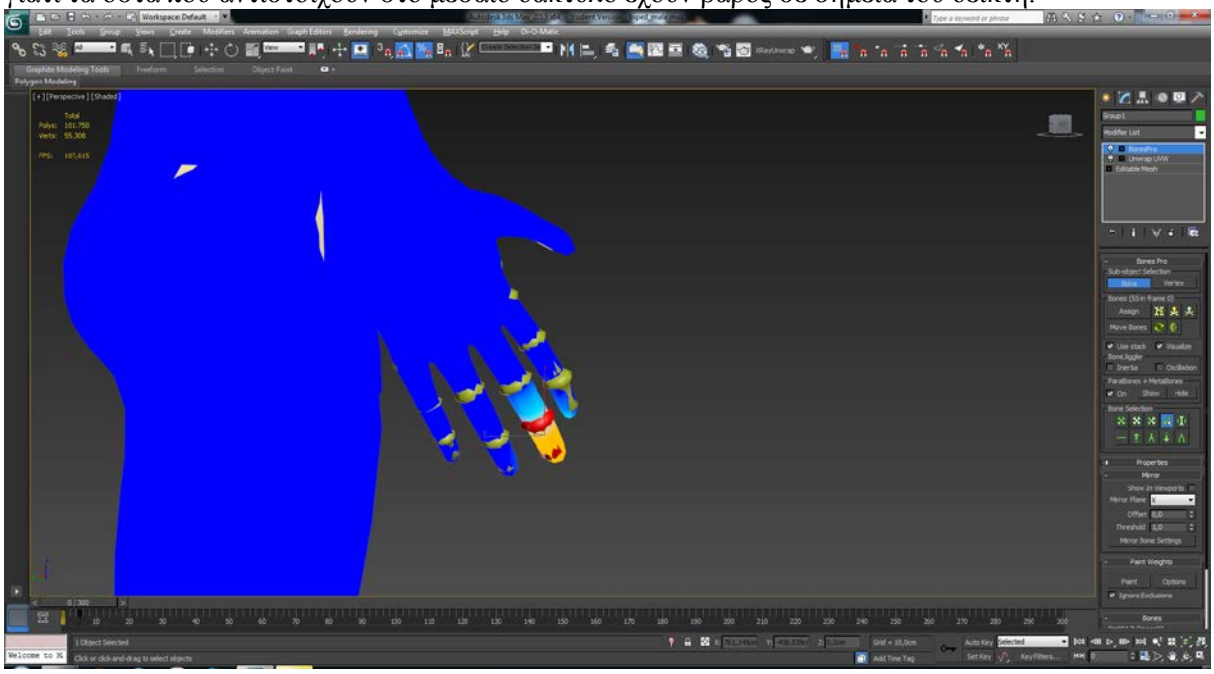

Figure 3.1-38: Λανθασμένο weight

Για να το διορθώσουμε, επιλέγουμε με sub-object selection > bone τα οστά στα οποία θέλουμε να επεξεργαστούμε την επιρροή, και με το paint weight tool «βάφουμε» τα σημεία του άλλου δακτύλου. Σημείωση: με απλό click αυξάνουμε το βάρος, ενώ κρατώντας πατημένο το alt και κάνοντας click το μειώνουμε.

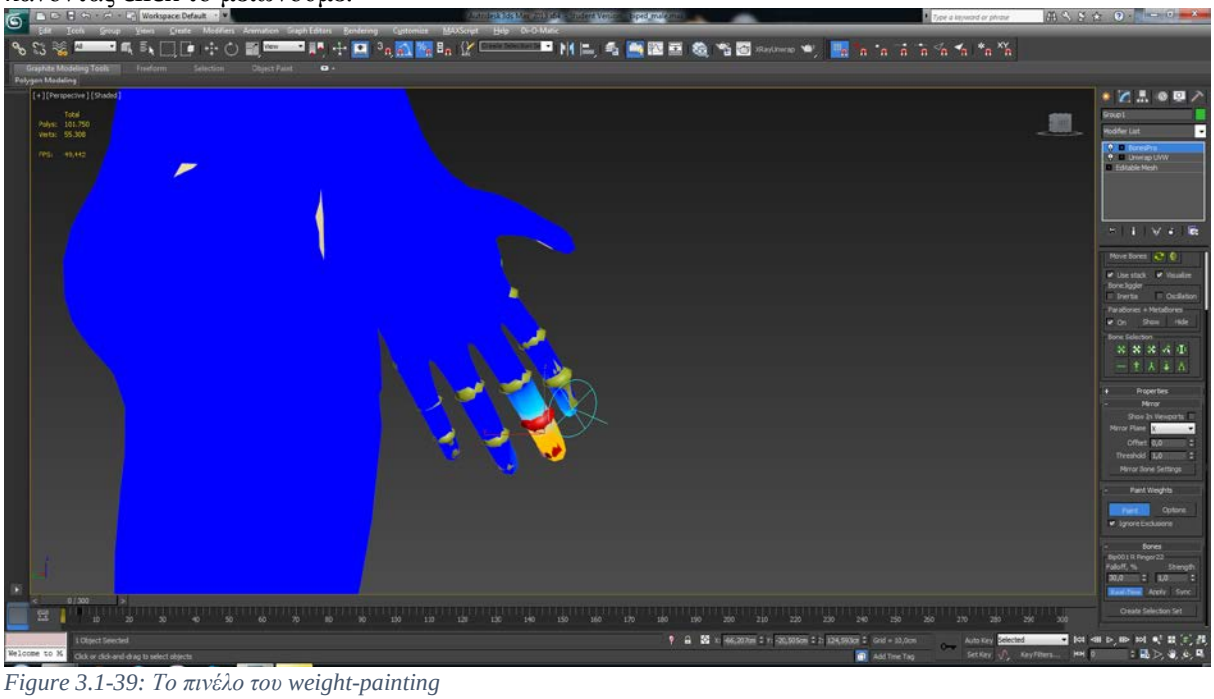

Figure 3.1-39: Το πινέλο του weight-painting

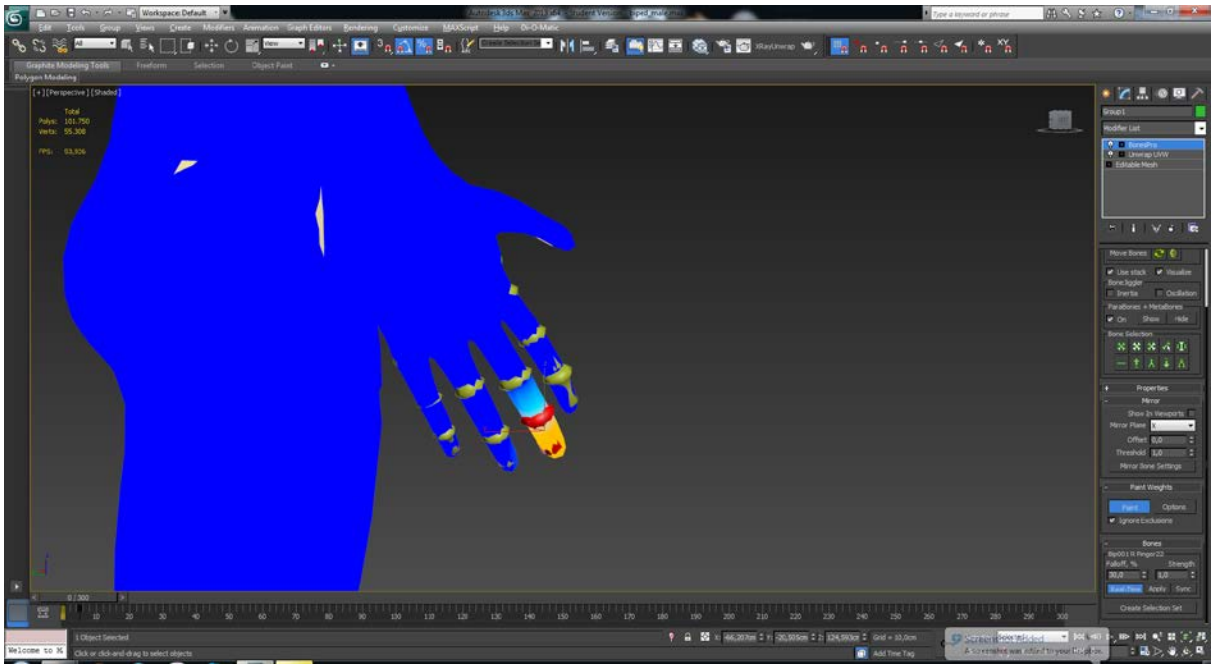

Figure 3.1-40: Διόρθωση λάθους

Το BonesPro μας δίνει τη δυνατότητα να εφαρμόσουμε την τεχνική mirror για συμμετρικά οστά, όπως είναι τα άκρα. Με οποιοδήποτε οστό επιλεγμένο και την επιλογή του Mirror->Show in Viewports ανοιχτή, μας εμφανίζεται ο άξονας που χρησιμοποιείται για την συμμετρία, καθώς και το οστό που αντιστοιχεί στην επιλογή μας.

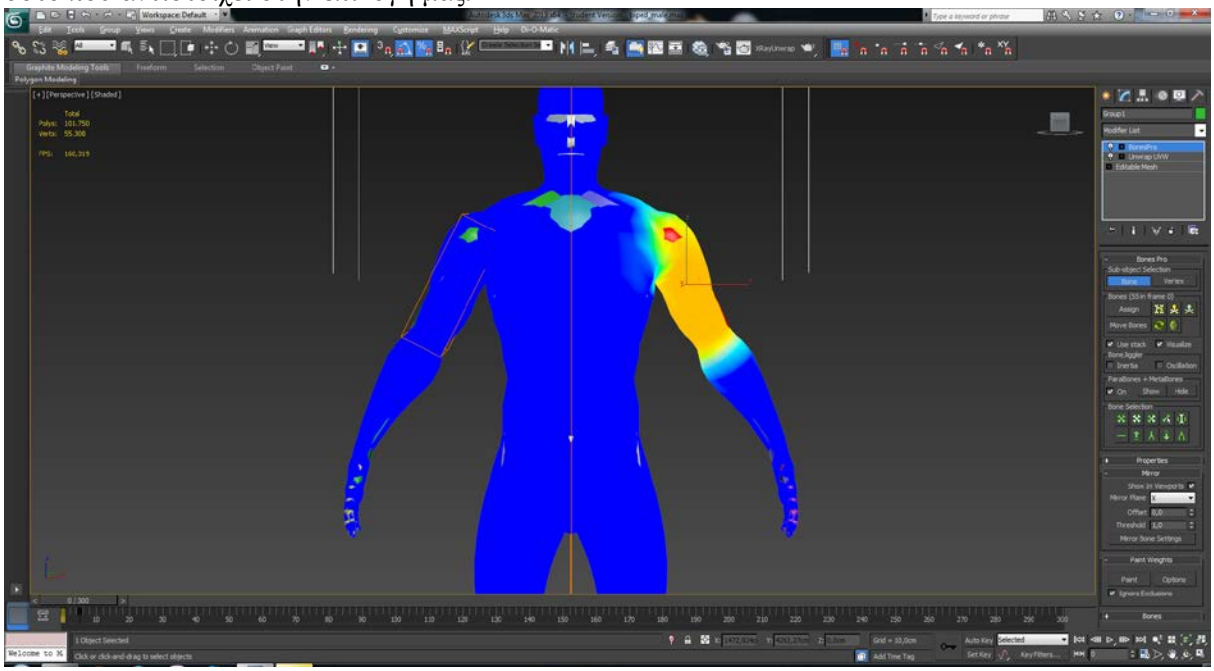

Figure 3.1-41: BonesPro Mirror

Σκοπός μας είναι να δημιουργήσουμε ένα φυσικό αποτέλεσμα, αποφεύγοντας αφύσικες παραμορφώσεις σε οποιαδήποτε στάση.

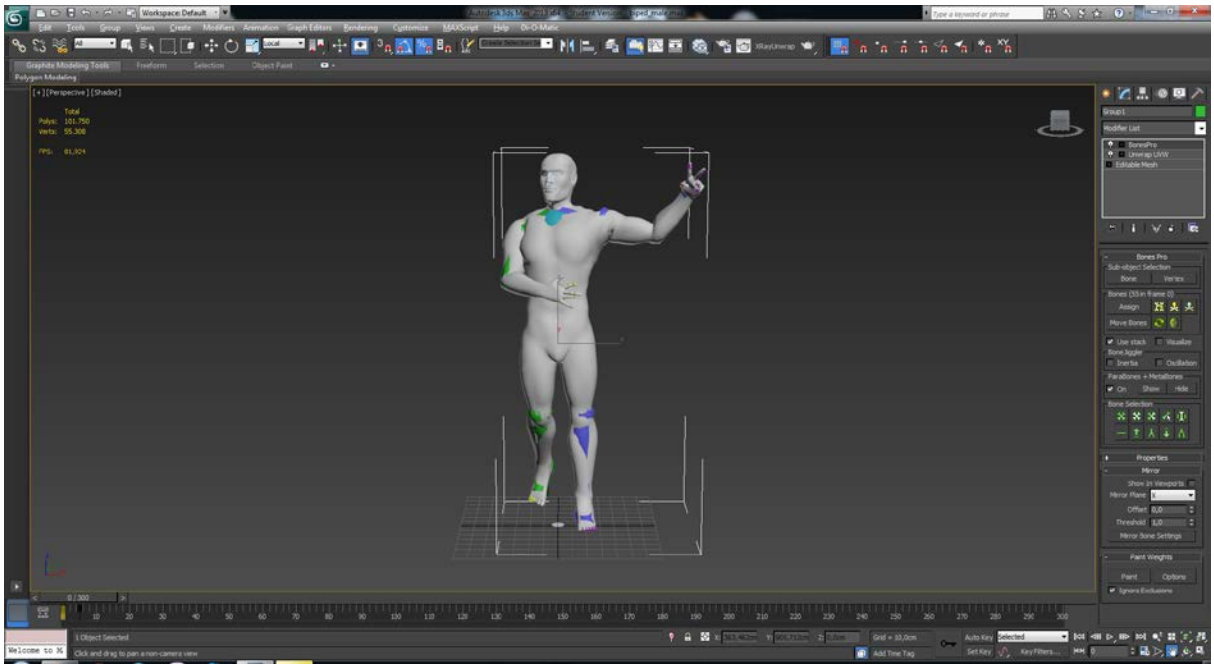

Figure 3.1-42: Έλεγχος κινήσεων

Αφού τελειώσουμε με το skinning και είμαστε ικανοποιημένοι με το αποτέλεσμα, μετατρέπουμε το BonesPro modifier σε Skin modifier με την επιλογή που προσφέρει το ίδιο το plugin. Ακολουθούμε την ίδια διαδικασία και για τα υπόλοιπα αντικείμενα που επηρεάζονται από παραπάνω από ένα οστό (ότι ακολουθεί την κίνηση ενός μόνο οστού και δε παραμορφώνεται, μπορεί απλά να συνδεθεί εξ' ολοκλήρου με το οστό αυτό και να μη γίνει skin).

3.1.8 Animation

Όπως προαναφέρθηκε, το animation γίνεται με τη μέθοδο του Keyframing, με το να θέτουμε δηλαδή την επιθυμητή θέση των οστών σε συγκεκριμένες χρονικές στιγμές, και να αφήνουμε το 3ds Max να επιλύει και να υπολογίζει τη θέση στα ενδιάμεσα στάδια.

Αρχικά, ορίζουμε μια πρώτη στάση-πόζα για το χαρακτήρα στο frame 0, και δημιουργούμε ένα keyframe. Για να δημιουργήσουμε το keyframe ο πιο απλός τρόπος είναι να επιλέξουμε το Auto Key, και επιλέγοντας ταυτόχρονα όλα τα οστά (Ctrl+A) να τους κάνουμε μια μικρή κίνηση και να τα επαναφέρουμε εκεί που ήταν. Αυτό θα δημιουργήσει ένα keyframe για κάθε οστό στο σημείο του timeline στο οποίο βρισκόμαστε, και θα εμφανίσει ένα μαύρο κουτάκι πάνω στο frame για να μας το υποδείξει.

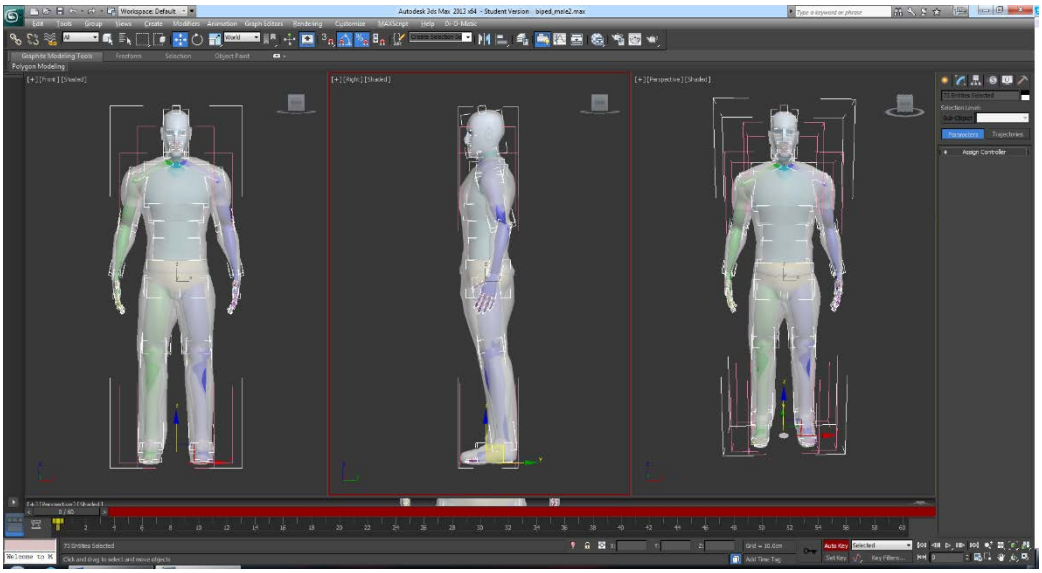

Figure 3.1-43: Δημιουργώντας το πρώτο keyframe

Αν θέλουμε το animation να επαναλαμβάνεται, τότε έχοντας ακόμα επιλεγμένα όλα τα οστά, επιλέγουμε με box selection τα keyframes στο πρώτο frame του timeline και με shift+drag τα αντιγράφουμε στο τελευταίο. Έτσι βεβαιώνουμε ότι το animation θα τελειώνει στην ίδια στάση με την οποία αρχίζει, κάτι το οποίο βοηθάει στο να έχουμε μια ομαλή επανάληψη της κίνησης.

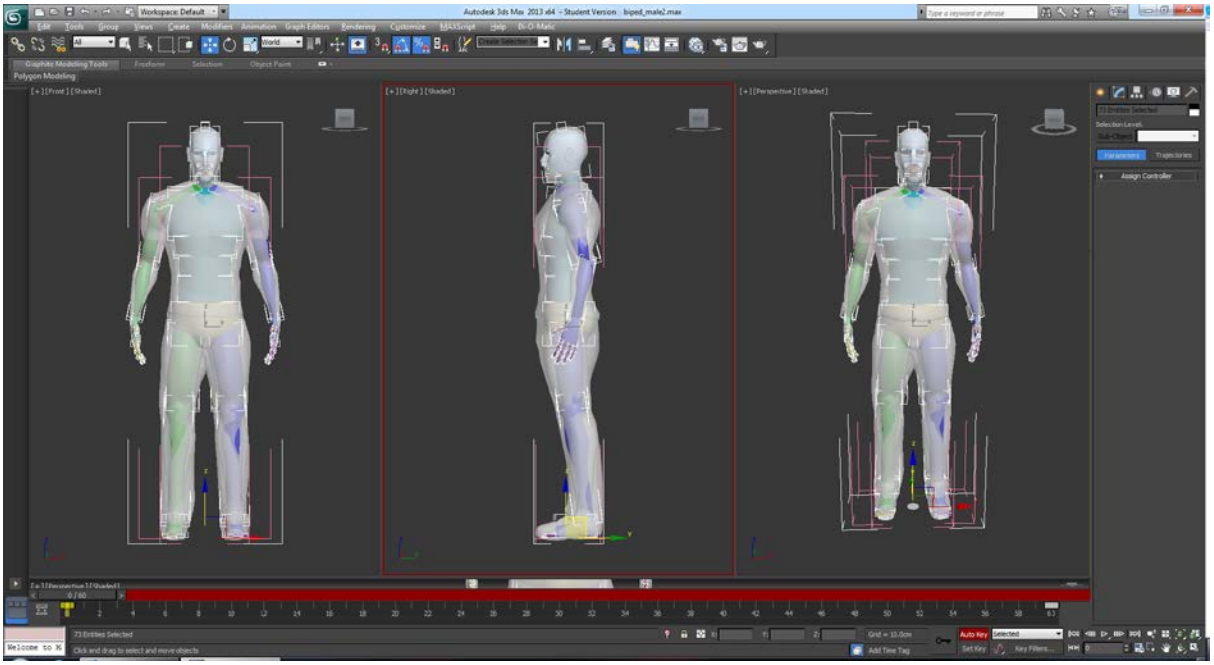

Figure 3.1-44: Αντιγράφουμε το πρώτο keyframe στη τελευταία θέση

Στη συνέχεια, μετακινούμαστε σε κάποιο άλλο frame του timeline και μετακινούμε ένα κόκαλο. Εφόσον το Auto Key παραμένει ανοιχτό, τότε η κίνηση μας θα δημιουργήσει άλλο ένα keyframe στη θέση αυτή.

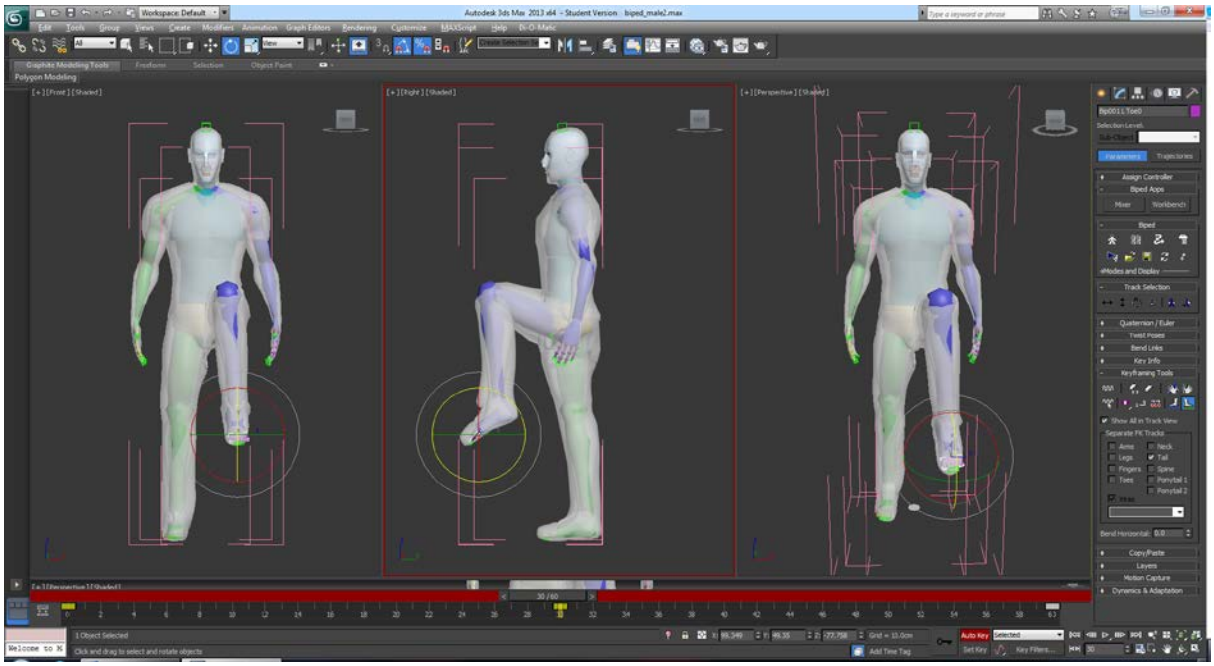

Figure 3.1-45: Δίνουμε καινούριες θέσεις στα keyframes

Αν σύρουμε τον δρομέα του timeline σε κάποιο frame μεταξύ αυτού που δημιουργήσαμε μόλις και του πρώτου, θα παρατηρήσουμε ότι το κόκαλο που μετακινήσαμε βρίσκεται σε μια ενδιάμεση θέση μεταξύ των δύο αυτών θέσεων.

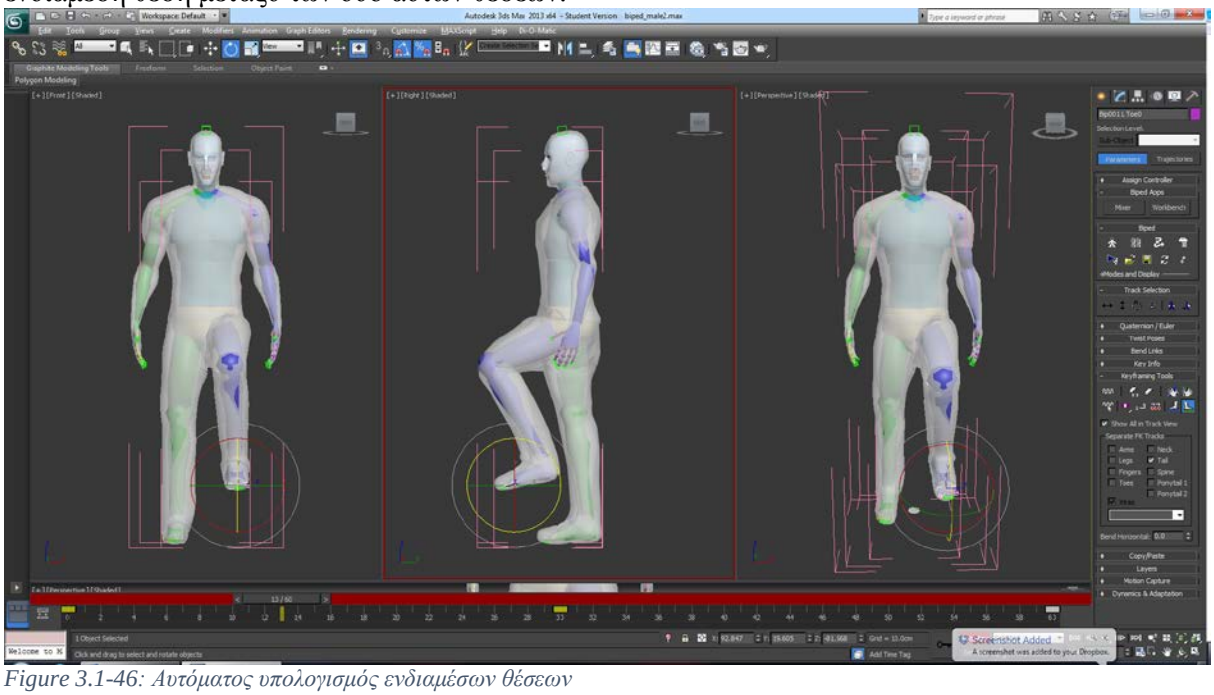

Figure 3.1-46: Αυτόματος υπολογισμός ενδιαμέσων θέσεων

Με αυτή τη λογική συνεχίζουμε και δημιουργούμε τις κινήσεις για τον χαρακτήρα μας. Το Biped του 3ds Max μας δίνει κάποιες πολύ χρήσιμες δυνατότητες, όπως για παράδειγμα το να «κλειδώσουμε» προσωρινά τη θέση των ποδιών, έτσι ώστε ο χαρακτήρας μας να πατάει συνεχώς στο έδαφος, και τα joint constraints, το ότι δηλαδή αποτρέπονται οι αφύσικες κινήσεις, π.χ. να λυγίζει το γόνατο προς λάθος κατεύθυνση.

Αξίζει να αναφερθεί ότι χάρη στο Mecanim του Unity, θα μπορούσαμε να δημιουργήσουμε το animation μόνο με το σκελετό, χωρίς το skin, αλλά επιλέχθηκε αυτή η διαδικασία ώστε να ελέγχουμε σε κάθε βήμα τη φυσικότητα της κίνησης. Αφού τελειώσουμε το animation, αποθηκεύουμε ξεχωριστά

το αρχείο. Κάθε animation θα χρειαστεί το δικό του αρχείο, το οποίο και θα είναι διαφορετικό από το βασικό αρχείο που δεν περιέχει animations.

3.1.9 Συνδεσιμότητα:

Τα αρχεία που δουλεύουμε στο 3ds Max είναι της μορφής .max. Για να τα εισάγουμε στο ZBrush τα κάνουμε export σε .obj, ενώ για το Unity σε .fbx. Το κάθε animation θα πρέπει να είναι και αυτό του τύπου .fbx αλλά ακολουθούμε ένα συγκεκριμένο συμβιβασμό ονομασίας ιδιαίτερο στη Unity, όπου αν το αρχικό αρχείο είναι biped_male.fbx το αρχείο που περιέχει το animation με όνομα Walk θα πρέπει να ονομάζεται biped_male@Walk.fbx. Με αυτόν τον τρόπο, το Unity θα αναγνωρίσει απευθείας τη σχέση μεταξύ των δύο αυτών αρχείων.

3.2 Χρήση του ZBrush στη πτυχιακή:

3.2.1 Γενικά

Το ZBrush χρησιμοποιήθηκε στα πλαίσια της πτυχιακής εργασίας σε δύο στάδια. Κατά το πρώτο στάδιο, το βασικό μοντέλο (base mesh) του 3ds Max σμιλεύτηκε σε 6 επίπεδα γεωμετρίας, με το κάθε επίπεδο να έχει τον τετραπλάσιο αριθμό πολυγώνων από το προηγούμενο (2005 πολύγωνα στο 1^{ο} επίπεδο, 2,046 εκατομμύρια πολύγωνα στο 6^{ο}) και δημιουργήθηκαν ρούχα με τη τεχνική του Sub-Tool Extract.

Στο δεύτερο στάδιο, αφού το μοντέλο επέστρεψε στο 3ds Max (από αυτό το σημείο και έπειτα χρησιμοποιήθηκε το 2^{ο} επίπεδο λεπτομέρειας του μοντέλου στα 8039 πολύγωνα) για να γίνει unwrapped, του δίνουμε χρώμα με Polypaint, σχεδιάζοντας δηλαδή κατευθείαν πάνω στο μοντέλο. Τέλος, μετατρέπουμε τις λεπτομέρειες και το χρώμα του υψηλότερου σταδίου σε normal και texture map αντίστοιχα για χρήση τους μέσα στη Unity.

3.2.2 Γλυπτική

Το ZBrush προσφέρει μια πληθώρα διαφορετικών πινέλων για τη σμίλευση ενός μοντέλου.

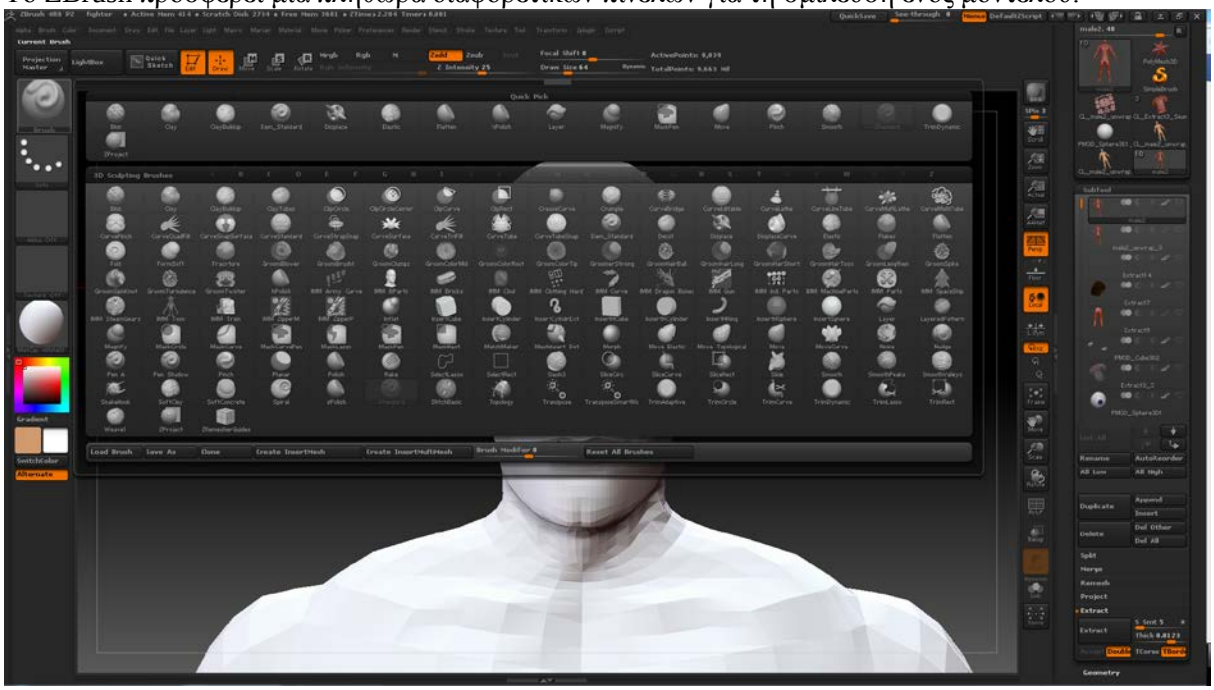

Figure 3.2-1: Τα πινέλα του ZBrush

Από αυτά χρησιμοποιήθηκαν κατά κόρον τα Standard, Dam_Standard, Clay Buildup, Move και το Smooth, το οποίο είναι εύκολα προσβάσιμο κρατώντας πατημένο το Shift.

Ξεκινώντας, κάνουμε import το μοντέλο μας σε αρχείο .obj ως tool του ZBrush και το τοποθετούμε στη σκηνή. Αυτή τη στιγμή είμαστε σε λειτουργία 2.5D οπότε αν προσπαθήσουμε να κάνουμε κάποια αλλαγή πάνω στο μοντέλο, απλά θα το κάνουμε μέρος του φόντου. Για να ενεργοποιήσουμε τη λειτουργία 3D πατάμε το κουμπί Edit στη μπάρα πάνω από τον καμβά.

Πατάμε το X στο πληκτρολόγια για να ενεργοποιήσουμε τη συμμετρία. Με το πινέλο Standard αρχίζουμε να δίνουμε σχήμα στο μοντέλο μας. Με click δημιουργούμε «βουνά» και με alt+click δημιουργούμε «κοιλάδες». Όπως προαναφέρθηκε, με shift+click χρησιμοποιούμε το πινέλο Smooth οπότε και εξομαλύνουμε την επιφάνεια. Το μοντέλο μας στην παρούσα φάση δε διαθέτει αρκετά πολύγωνα για να αποτυπώσουμε κανενός είδους λεπτομέρεια οπότε εστιάζουμε στο να ορίσουμε καλύτερα το περίγραμμα του χαρακτήρα.

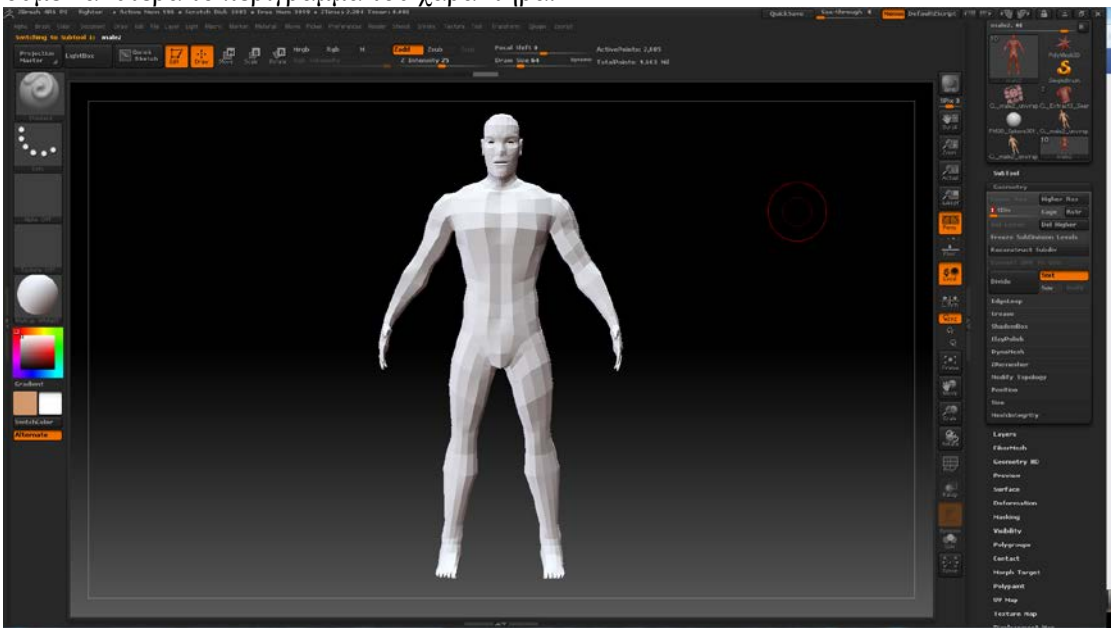

Figure 3.2-2: Επίπεδο 1

Στη συνέχεια, από τις επιλογές της γεωμετρίας στα δεξιά (Geometry) πατάμε Divide το οποίο θα διαιρέσει κάθε ενεργό σημείο του μοντέλου δια 4, κρατώντας έτσι το σχήμα του mesh αλλά δίνοντας μας περισσότερα πολύγωνα να σμιλέψουμε.

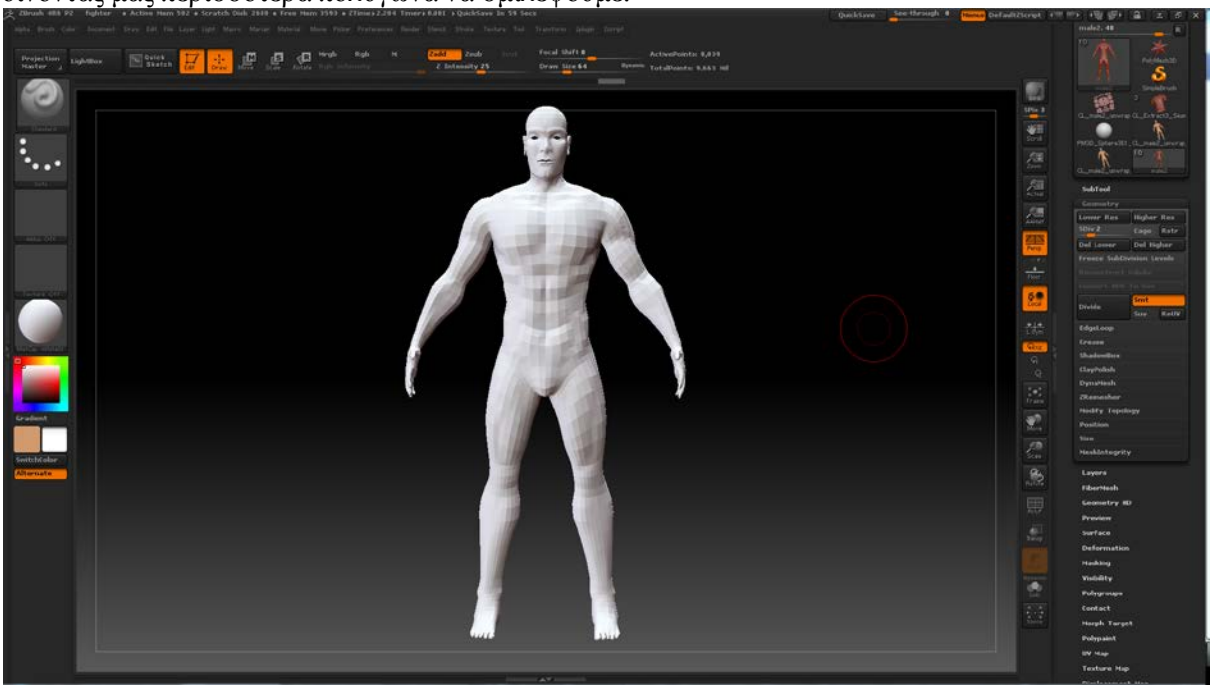

Figure 3.2-3: Επίπεδο 2

Το Move πινέλο μας βοηθάει να πιάσουμε ένα κομμάτι του μοντέλου και να το μετακινήσουμε προς μια κατεύθυνση. Χάρη σε αυτό, τροποποιούμε το πρόσωπο και το κάνουμε να φαίνεται πιο φυσικό και ανθρώπινο.

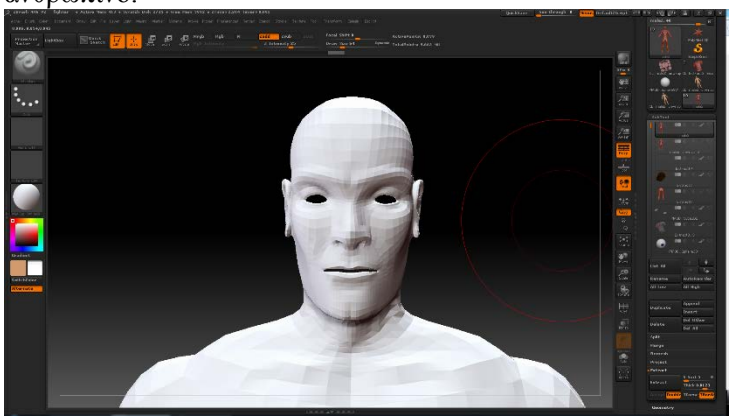

Figure 3.2-4: Αρχικό πρόσωπο

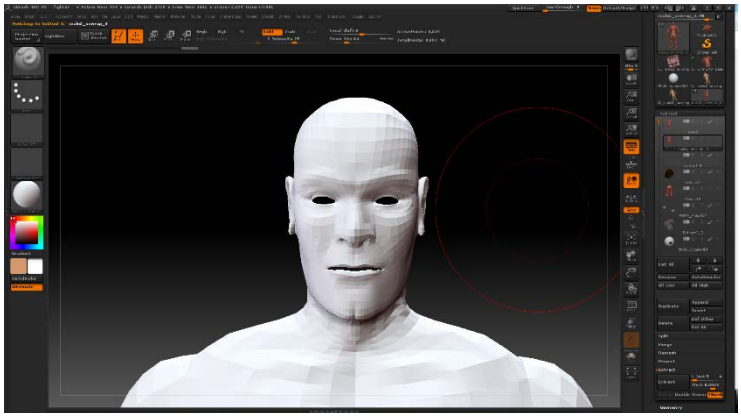

Figure 3.2-5: Διορθωμένο πρόσωπο

Το Clay Buildup πινέλο, δίνει την αίσθηση ότι προσθέτει ένα επιπλέον στρώμα πάνω στο μοντέλο το οποίο μετά μπορούμε να κάνουμε Smooth, κάτι ιδιαίτερα χρήσιμο στη βασική δημιουργία περιγράμματος μυών.

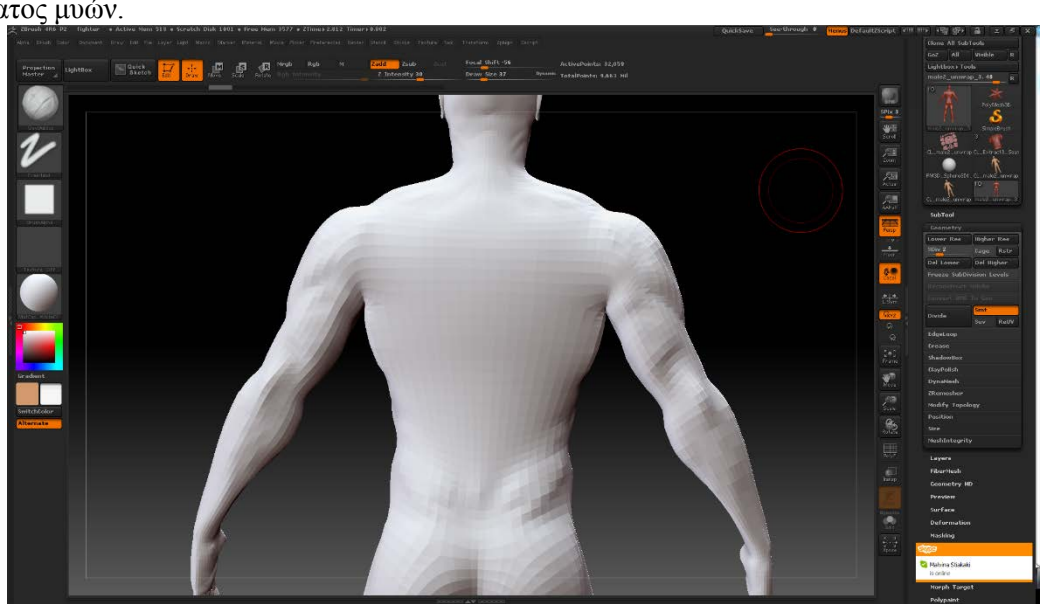

Figure 3.2-6: Επίπεδη επιφάνεια

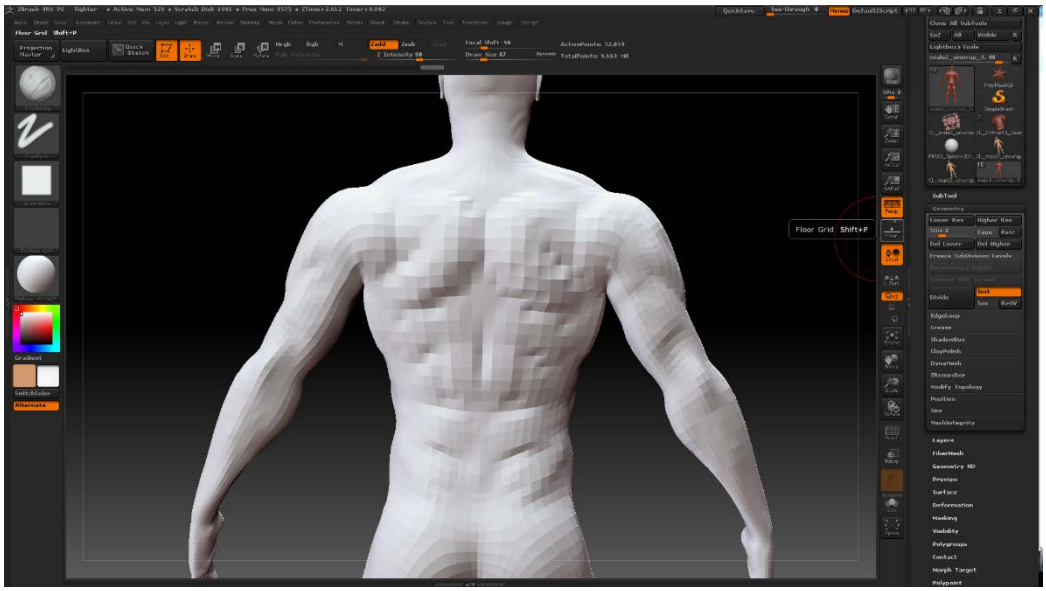

Figure 3.2-7: Clay Buildup

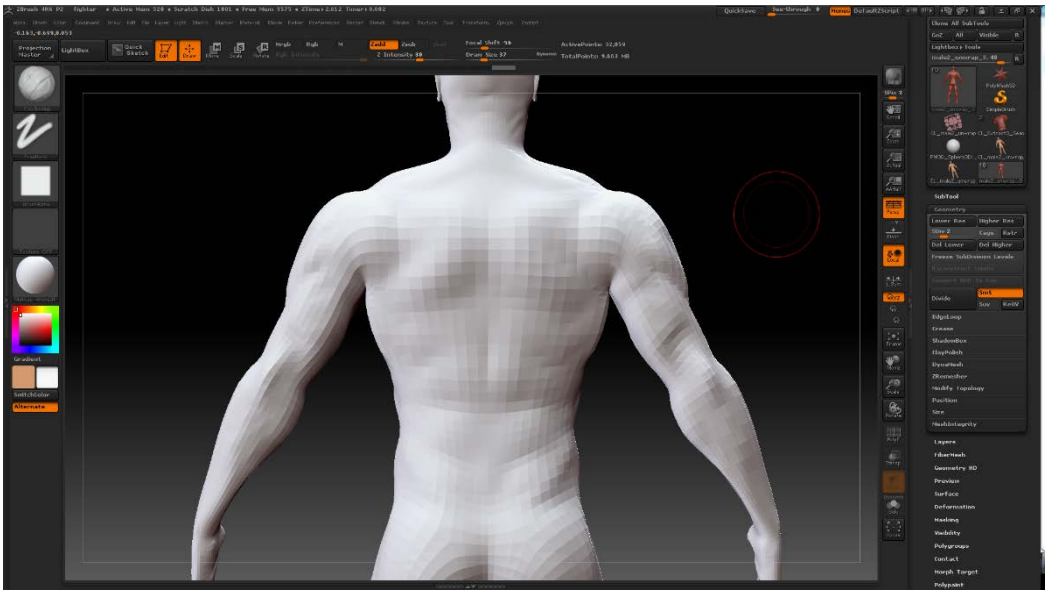

Figure 3.2-8: Smooth

Συνεχίζουμε την ίδια διαδικασία, δηλαδή με το Clay Buildup να περιγράφουμε τους μυς, με το Move να τροποποιούμε τη θέση των λεπτομερειών, με το Smooth να εξομαλύνουμε τις επιφάνειες και με το Standard να δίνουμε περισσότερο όγκο, όπου αυτό είναι επιθυμητό. Όταν φτάνουμε σε σημείο να μην έχουμε τη δυνατότητα να προσθέσουμε περισσότερες λεπτομέρειες λόγω του αριθμού των πολυγώνων, κάνουμε ξανά Divide και συνεχίζουμε. Όσο ανεβαίνουμε επίπεδα λεπτομέρειας αρχίζουμε να χρησιμοποιούμε και το πινέλο Dam_Standard που είναι ουσιαστικά ένα μικρό, αντίστροφο του Standard, ιδανικό για να διαγράψουμε και να κάνουμε πιο έντονες τις διακυμάνσεις της επιφάνειας.

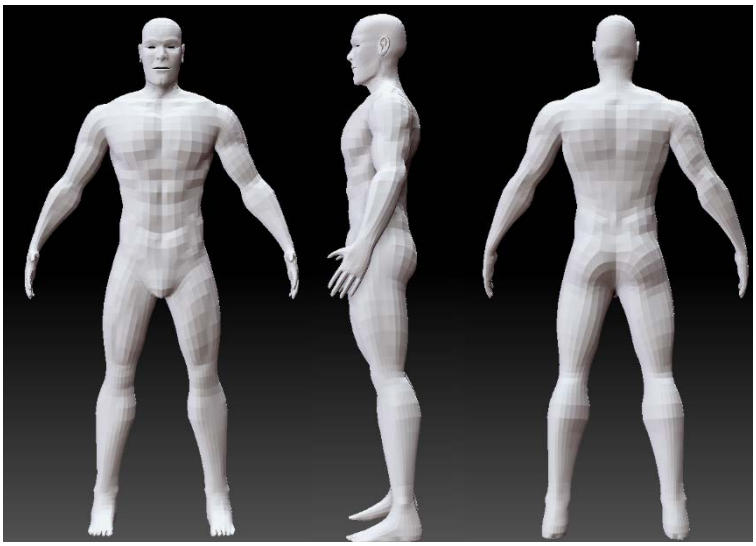

Figure 3.2-9: Επίπεδο 2 (καινούριο low-poly)

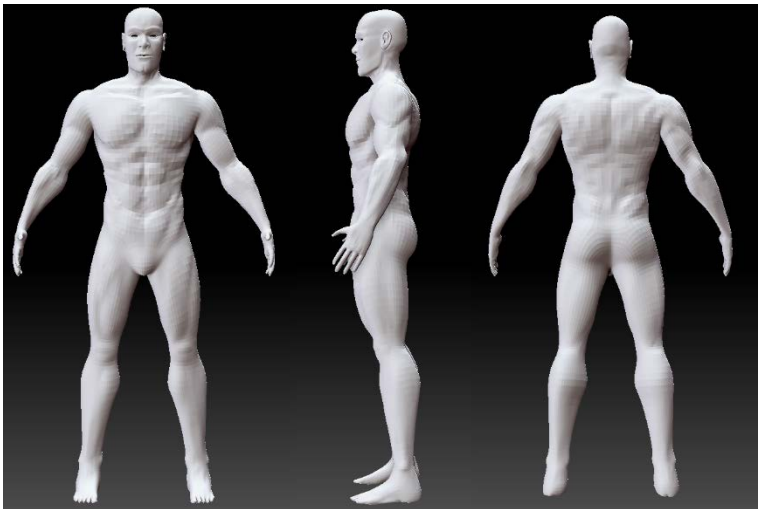

Figure 3.2-10: Επίπεδο 3

Figure 3.2-11: Επίπεδο 4

Figure 3.2-12: Επίπεδο 5

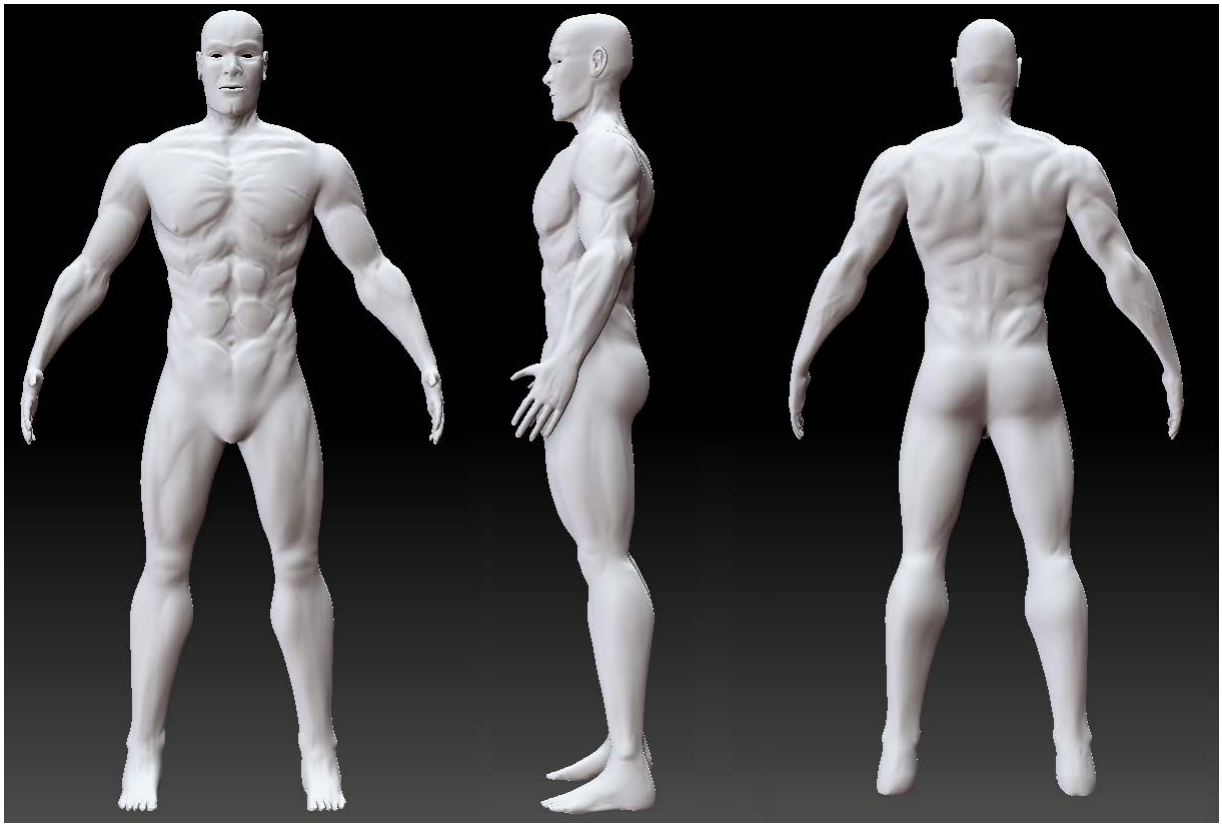

Figure 3.2-13: Επίπεδο 6

Έχοντας επιτύχει το τελευταίο επίπεδο του Divide, όχι μόνο έχουμε ένα πολύ πιο ομαλό αποτέλεσμα, αλλά έχουμε την ευκαιρία να κάνουμε μικρές αλλαγές και να προσθέσουμε λεπτομέρειες που δε φαίνονται εύκολα σε μια εικόνα ολόκληρου του μοντέλου.

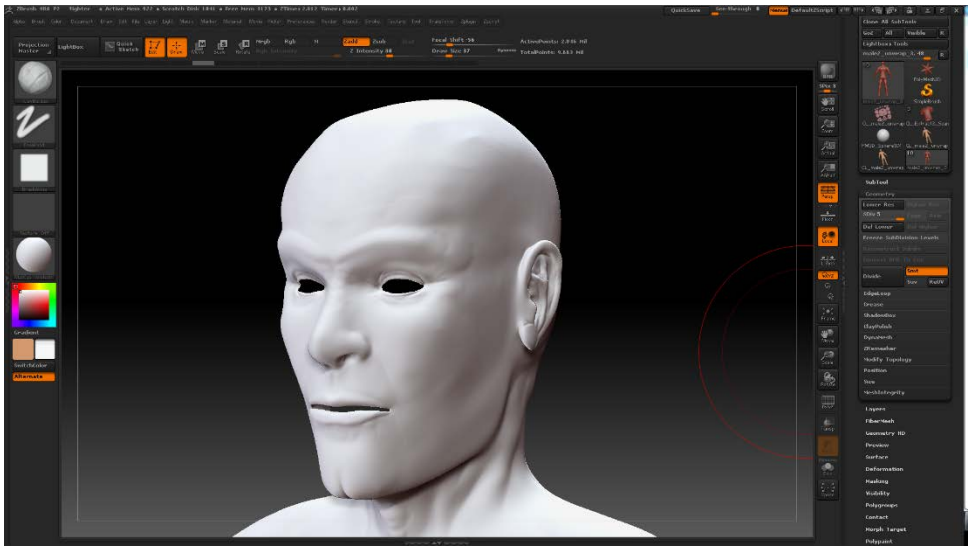

Figure 3.2-14: Λεπτομέρειες προσώπου

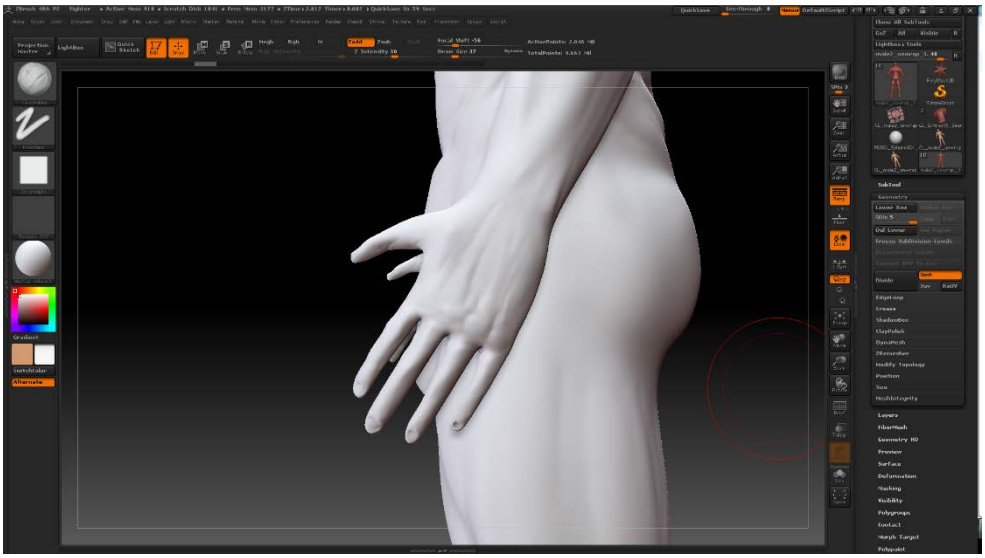

Figure 3.2-15: Λεπτομέρειες χεριών

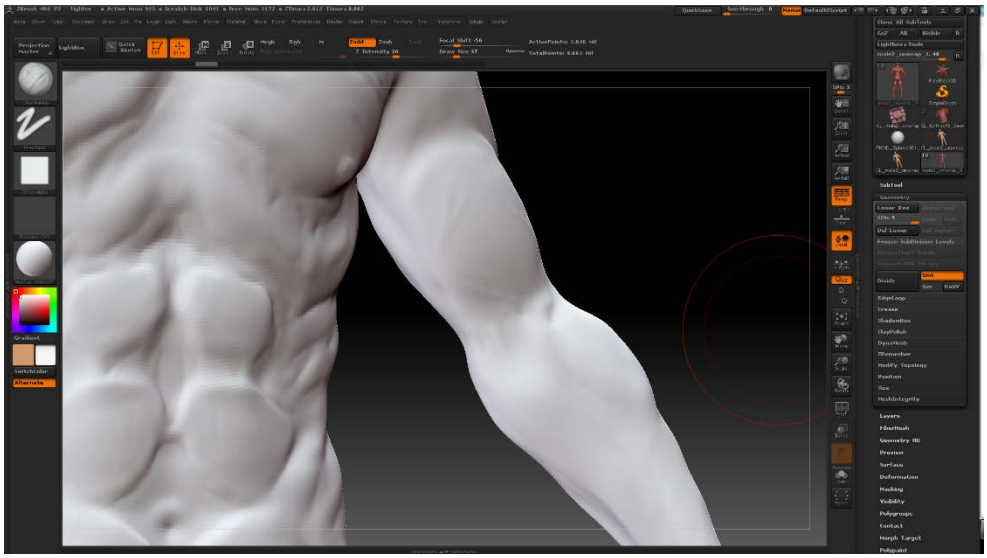

Figure 3.2-16: Λεπτομέρειες κορμού

3.2.3 Sub-Tool Extract – Δημιουργία υπο-αντικειμένων

Αφού έχουμε τελειώσει με τη γλυπτική πάνω στο μοντέλο του χαρακτήρα μας, ήρθε η ώρα να δημιουργήσουμε τα ρούχα του. Αυτό θα γίνει με το εργαλείο Sub-Tool Extract που βρίσκεται στη δεξιά μπάρα. Αρχικά, σε χαμηλό επίπεδο πολυγώνων, καλύπτουμε με Mask την επιφάνεια που θέλουμε να καλύψουμε με το ρούχο μας. Για παράδειγμα, για να φτιάξουμε μια μπλούζα, θα κάνουμε Mask το κορμό και μέρος των χεριών. Το πινέλο μας μετατρέπεται σε πινέλο Mask κρατώντας πατημένο το Ctrl.

Figure 3.2-17: Mask περιοχής

Από τα δεξιά, επεκτείνουμε το μενού του Extract, ορίζουμε το πάχος του καινούριου αντικειμένου και τον σχετικό αριθμό πολυγώνων που θέλουμε να έχει στο πρώτο στάδιο και πατάμε Extract.

Figure 3.2-18: Επιλογές Extract

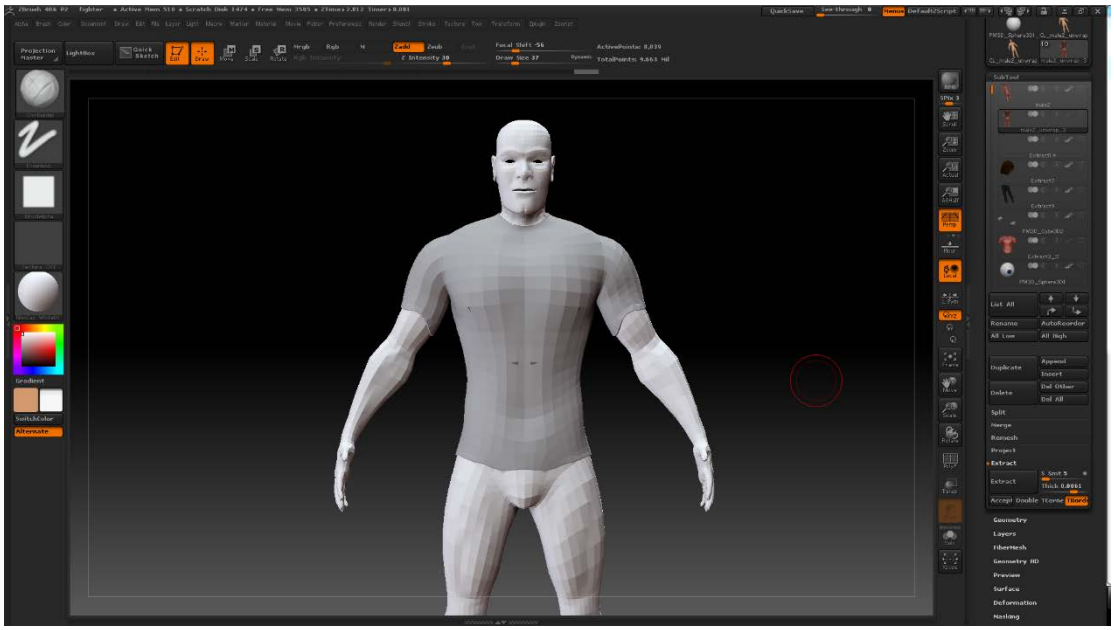

Figure 3.2-19: Extracted SubTool

Αφού λειάνουμε το περίγραμμα και το σχήμα του νέου αντικειμένου, δουλεύουμε την μπλούζα με τον ίδιο τρόπο που δουλέψαμε το σώμα. Σημειώνεται ότι επειδή το εσωτερικό της μπλούζας και γενικά των ρούχων δεν πρόκειται σε καμία περίπτωση να είναι εμφανές, μπορούμε να κάνουμε το αντικείμενο Split σε 3 κομμάτια (εξωτερικό, περίγραμμα, εσωτερικό), να διαγράψουμε ότι δε φαίνεται και να συνενώσουμε ξανά τα άλλα δύο.

Figure 3.2-20: Στάδια λεπτομέρειας μπλούζας

Με ακριβώς την ίδια διαδικασία δημιουργήθηκε το παντελόνι, ενώ τα παπούτσια ξεκίνησαν ως ένα mirrored box που σμιλεύτηκε στο σχήμα παπουτσιού σε μόλις δύο στάδια.

Figure 3.2-21: Στάδια λεπτομέρειας παντελονιού

Figure 3.2-22: Στάδια λεπτομέρειας παπουτσιών

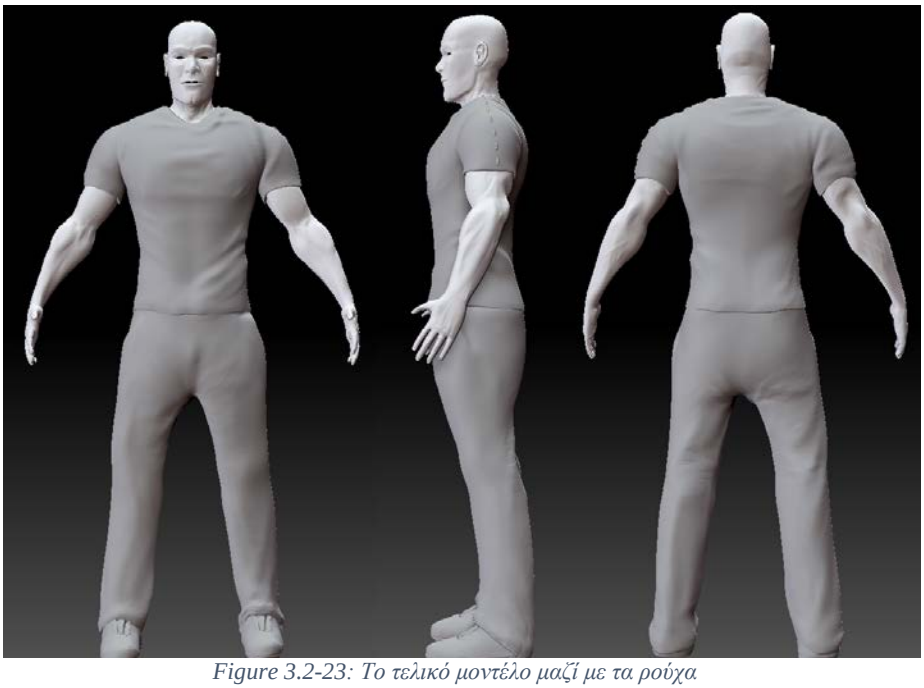

Figure 3.2-23: Το τελικό μοντέλο μαζί με τα ρούχα

3.2.4 Polypaint - Χρωματισμός

Σειρά έχει να δώσουμε χρώμα στο χαρακτήρα μας. Κρύβουμε ξανά τα υπόλοιπα αντικείμενα πατώντας πάνω στο σύμβολο με το μάτι δίπλα από το μοντέλο του χαρακτήρα στο υπομενού Sub Tools, επιλέγουμε από την παλέτα χρωμάτων ένα βασικό χρώμα και υλικό, και στο μενού Color επιλέγουμε Fill Object.

Figure 3.2-24: Επιλογή βασικού χρώματος

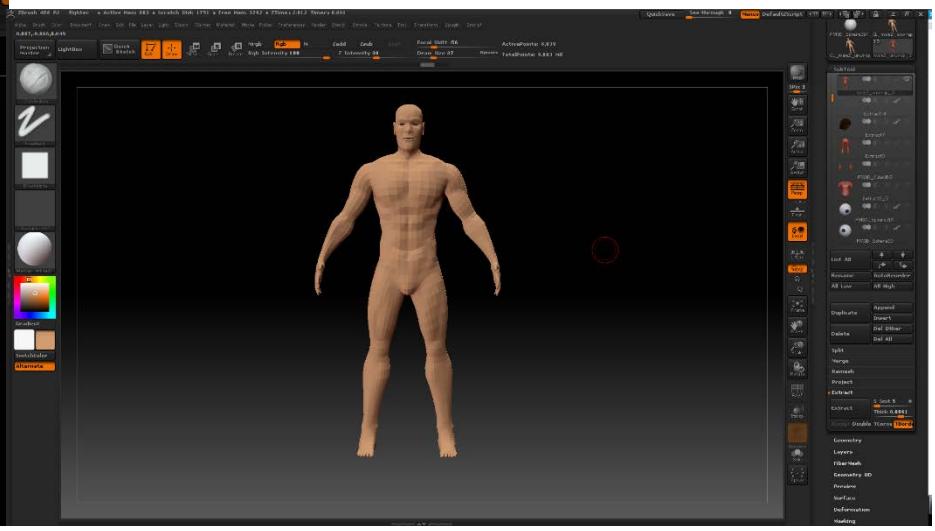

Figure 3.2-25: Fill Object

Δίνουμε έτσι σε ολόκληρο το μοντέλο ένα ομοιόμορφο χρώμα. Η λογική παραμένει ίδια με του σμιλέματος: ξεκινάμε από το χαμηλότερο επίπεδο πολυγώνων και προσθέτουμε σε κάθε ένα όση περισσότερη λεπτομέρεια μπορούμε.

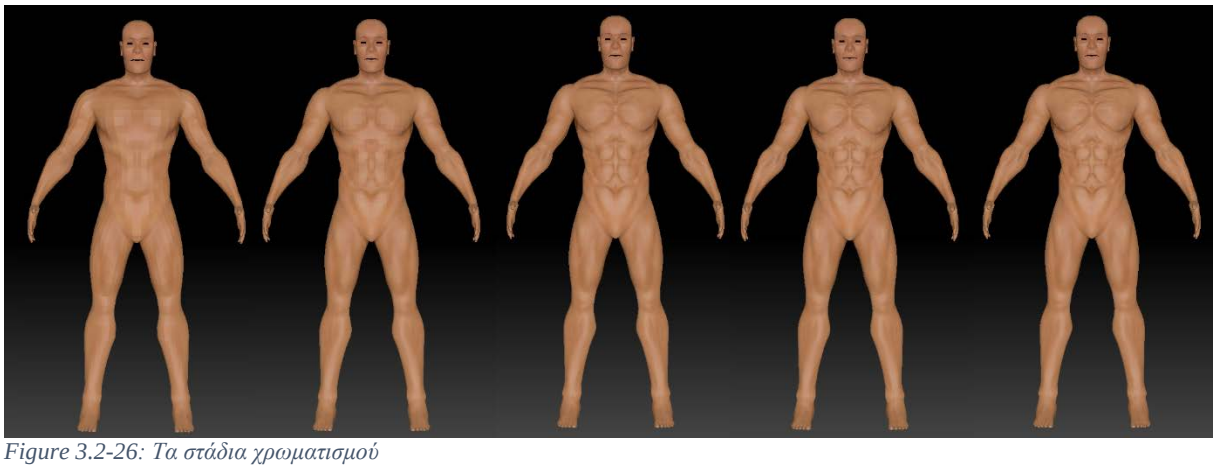

Figure 3.2-26: Τα στάδια χρωματισμού

Χρησιμοποιώντας τα διάφορα Strokes και Alphas των πινέλων, μπορέσαμε να δημιουργήσουμε έναν αληθοφανή χρωματισμό στο ψηλό επίπεδο λεπτομέρειας. Πιο συγκεκριμένα, με Color Spray Stroke και Alpha 23 δίνουμε την αίσθηση δέρματος, βάφοντας πολύ ελαφριά με μικρές παραλλαγές του διαφορετικού χρώματος, θυμίζοντας τις διακυμάνσεις στο χρώμα που προκαλούν οι πόροι του δέρματος, με Drag Rect Stroke, Alpha 22 και μια σκούρα απόχρωση του μπλε δημιουργήσαμε υποτυπώδεις φλέβες σε σημεία του σώματος, όπου είχε λογική να φαίνονται, και με Freehand Stroke, Alpha 58 και μαύρο χρώμα προσθέσαμε τριχοφυΐα.

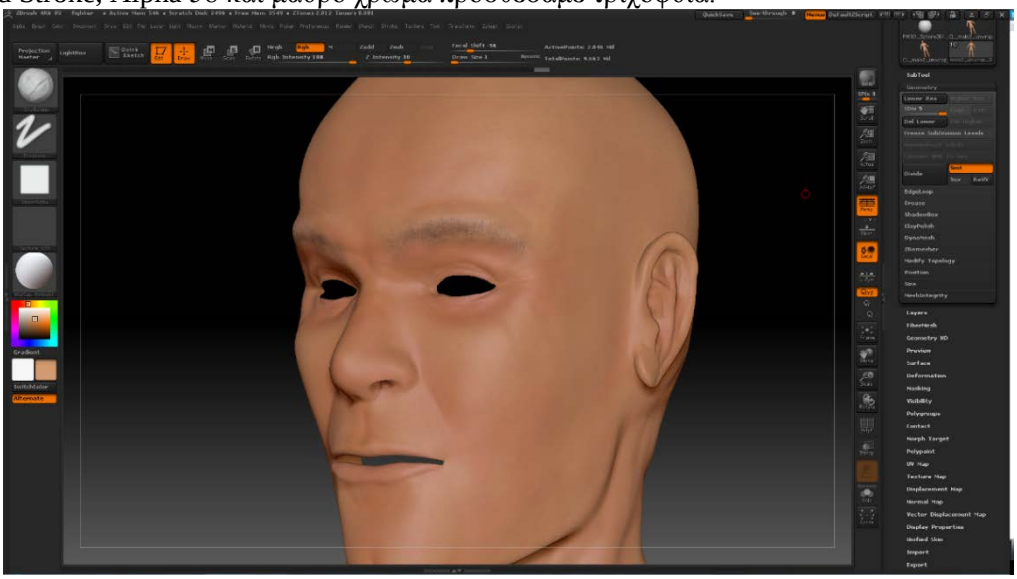

Figure 3.2-27: Λεπτομέρειες προσώπου

Figure 3.2-28: Λεπτομέρειες κορμού

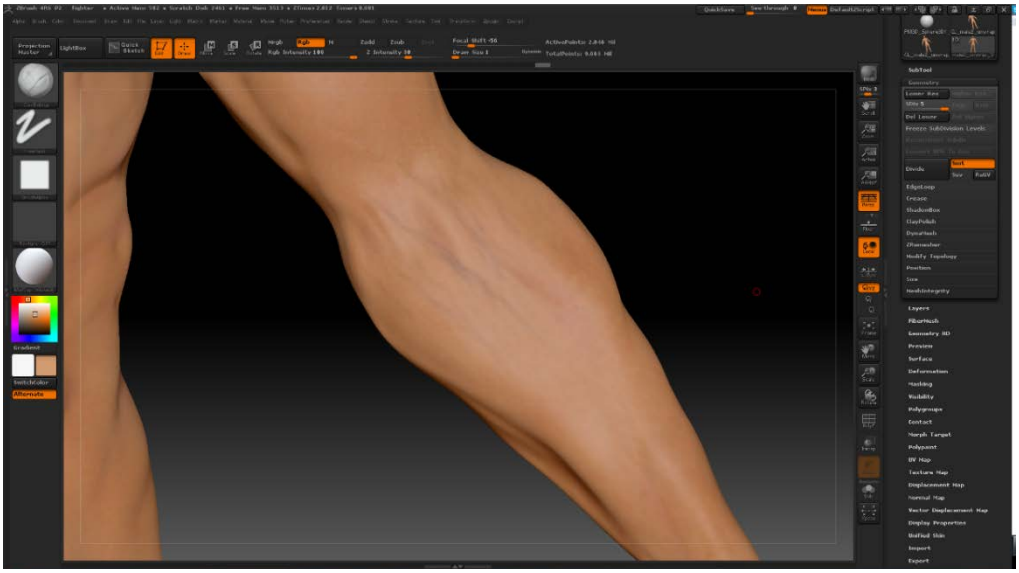

Figure 3.2-29: Λεπτομέρειες χεριών

Η ίδια διαδικασία μας βοήθησε και στο χρωματισμό των ρούχων, με εξαίρεση το λογότυπο της μπλούζας. Για να το πετύχουμε αυτό χρησιμοποιήσαμε το εργαλείο Spotlight. Το Spotlight μας επιτρέπει να προβάλλουμε μια εικόνα πάνω στον καμβά μας, και να την αποτυπώσουμε με Polypaint κατευθείαν πάνω στο αντικείμενο. Έτσι, φορτώσαμε την λογότυπο ως texture στο ZBrush, ανοίξαμε το Spotlight και αφού το τοποθετήσαμε πάνω από το επιθυμητό σημείο, σε paint mode βάψαμε από πάνω του, αντιγράφοντας την εικόνα ανάγλυφα πάνω στο mesh.

Figure 3.2-30: Spotlight

Figure 3.2-31: Το logo πάνω στη μπλούζα

Figure 3.2-32: Τελικό αποτέλεσμα

3.2.5 Multi Map Exporter

Έχοντας τελειώσει με την γλυπτική αλλά και το χρωματισμό του χαρακτήρα, μας μένει μόνο να εξάγουμε τους χάρτες υφών που θέλουμε να χρησιμοποιήσουμε. Το ZBrush διαθέτει ένα εργαλείο κατάλληλο για αυτή τη δουλειά, το Multi Map Exporter. Το Multi Map Exporter ξεκίνησε ως επιπλέον plugin του ZBrush που μπορούσαν οι χρήστες να κατεβάσουν από το site της Pixologic,

αλλά στην πορεία, αφού αναγνωρίστηκε η χρησιμότητα του, έγινε μέρος του βασικού πακέτου. Το εργαλείο αυτό μας προσφέρει ένα γρήγορο τρόπο να εξάγουμε διάφορους χάρτες για εξωτερική χρήση. Από αυτούς εμείς χρειαζόμαστε texture και normal maps. Το texture map είναι ουσιαστικά η απεικόνιση του Polypaint πάνω στο UVW map του μοντέλου, ενώ το normal map είναι μια αναπαράσταση των ανωμαλιών της επιφάνειας, ώστε ο renderer, μέσω του φωτισμού, να μας δώσει επιπλέον λεπτομέρεια. Μέσα στην υλοποίηση θα χρησιμοποιηθεί το low-poly μοντέλο του χαρακτήρα, λόγω μικρού αριθμού πολυγώνων, οπότε το normal map είναι απαραίτητο για να μπορέσουμε να εμφανίσουμε όλη τη λεπτομέρεια που έχουμε δημιουργήσει.

Από το μενού ZPlugin, ανοίγουμε το υπομενού Multi Map Exporter, επιλέγουμε ποια maps επιθυμούμε και πατάμε πάνω στο κουμπί Export Options. Αυτό μας εμφανίζει τις επιλογές των maps. Το texture map δεν μας δίνει κάποια επιλογή, αλλά στο normal map επιλέγουμε το υψηλότερο subdivision, tangent, adaptive, SmoothUV και SNormals. Οι επιλογές αυτές θα μας δώσουνε ένα καθαρό normal map σε τοπικές συντεταγμένες, χωρίς παραμορφώσεις, με περισσότερη έμφαση σε σημεία με μεγάλη λεπτομέρεια, όπως το πρόσωπο. Αφού σιγουρευτούμε για τις επιλογές μας, πατάμε Create All Maps και σε σύντομο χρονικό διάστημα δημιουργούνται.

Figure 3.2-33: Texture Map

Figure 3.2-34: Normal Map

3.3 Το Unity στη πτυχιακή

3.3.1 Εισαγωγή

Η μηχανή Unity ήταν το κύριο εργαλείο που χρησιμοποιήθηκε για την υλοποίηση της πτυχιακής και πάνω σε αυτήν συνδυάστηκαν όσα κάναμε νωρίτερα με πολλά άλλα για τη δημιουργία του τελικού αποτελέσματος, ένα παιχνίδι δηλαδή που μας δίνει τη δυνατότητα κίνησης στο χώρο με δυνατότητες επέκτασης σε πολλά διαφορετικά είδη. Θα αναλύσουμε πώς δημιουργήσαμε το περιβάλλον, πώς εισαγάγαμε το χαρακτήρα μας μέσα στη σκηνή του Unity, πώς στήσαμε το Mecanim και τα State Machines που έδωσαν κίνηση στο χαρακτήρα, πώς προγραμματίσαμε ορισμένες λειτουργίες και τέλος πώς εφαρμόσαμε τα ειδικά εφέ. Στο τέλος του κεφαλαίου το αποτέλεσμα που θα έχουμε θα είναι μια εφαρμογή με ρεαλιστικά γραφικά, έναν χαρακτήρα που κινείται ελεύθερα στο χώρο με input από το πληκτρολόγιο, ρεαλιστικό νερό, έδαφος με δέντρα και βλάστηση, ήχους περιβάλλοντος, συστήματα particles και light effects. Στο επόμενο κεφάλαιο θα γίνουν όλες οι απαραίτητες αλλαγές ώστε ο χρήστης να μπορεί να χειρίζεται το χαρακτήρα μέσω του Kinect.

3.3.2 Το περιβάλλον

3.3.2.1 Το έδαφος (Terrain)

Heightmap και Textures:

Ένα έδαφος στη Unity είναι κατά βάση ένα επίπεδο με συνήθως μεγάλη ανάλυση και επιπλέον εργαλεία. Ξεκινώντας, δημιουργούμε ένα κενό έδαφος πατώντας στο GameObject > Create Other > Terrain και από το παράθυρο του Inspector θέτουμε το ύψος του στα 600, έτσι ώστε το βασικό ύψος να είναι υπερυψωμένο και να μπορούμε να έχουμε χαμηλότερες τιμές από αυτό. Το αντικείμενο terrain μπορεί να δεχτεί heightmap για να δώσει σχήμα στο έδαφος. Δημιουργήσαμε το παρακάτω heightmap στο Photoshop και το αποθηκεύσαμε σε μορφή .raw.

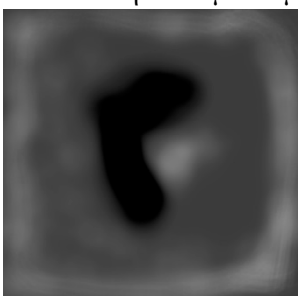

Figure 3.3-1: Το RAW Heightmap

Με το συγκεκριμένο Heightmap, το ευθύγραμμο επίπεδο μας πήρε αυτή τη μορφή:

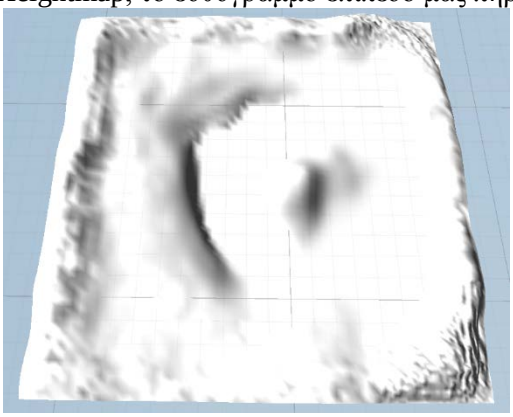

Figure 3.3-2: Terrain με Heightmap

Προσθέτουμε στα textures του Terrain 6 εικόνες: μια που θα χρησιμοποιηθεί ως βάση σε όλο το έδαφος, και πέντε για το συνδυασμό τους και το χρωματισμό των διαφόρων περιοχών. Στη συνέχεια, αρχίζουμε και βάφουμε το έδαφος όπως επιθυμούμε.

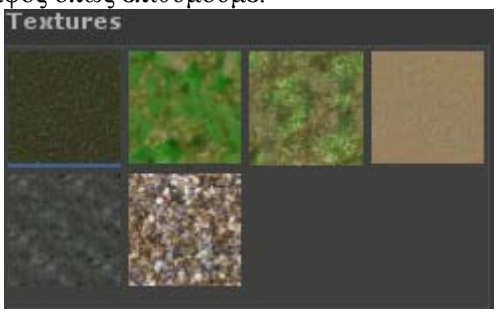

Figure 3.3-3: Τα έξι textures του Terrain

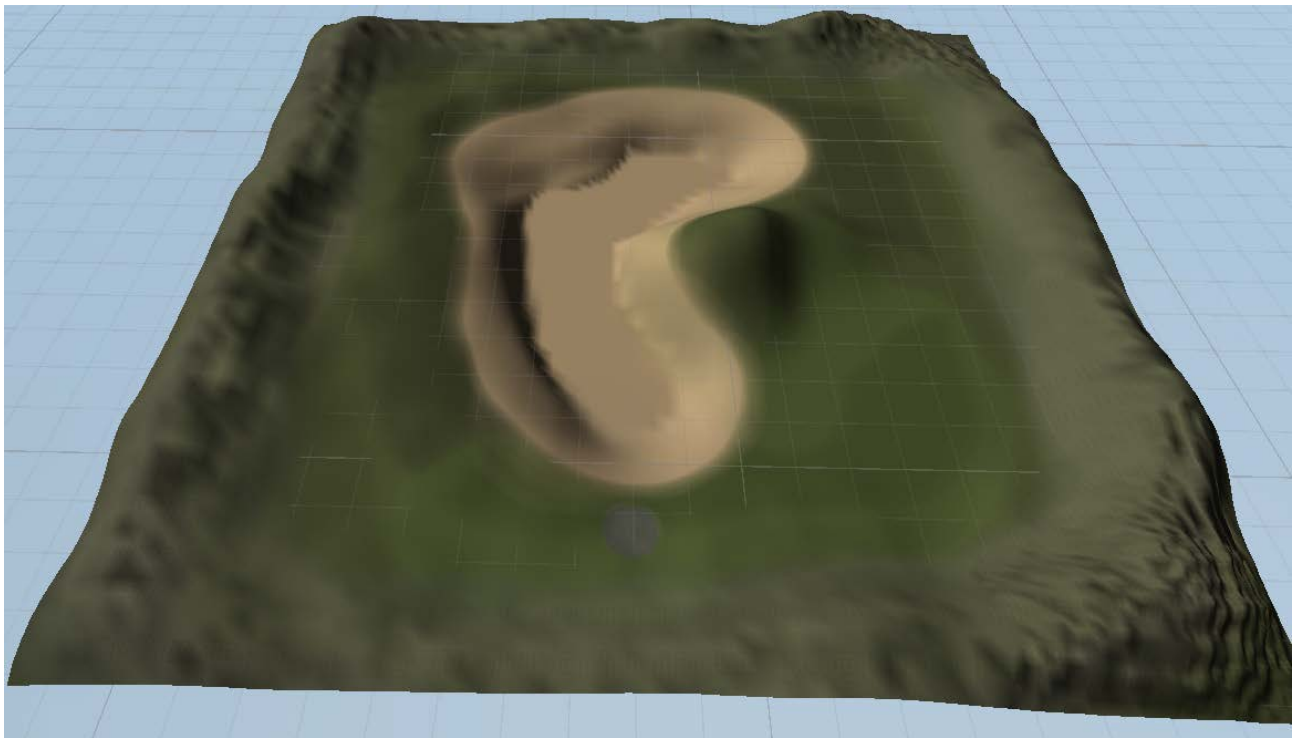

Figure 3.3-4: Terrain με Heightmap και Textures

Βλάστηση (Trees, Grass και Details):

Για την ώρα το έδαφος μας δεν είναι ιδιαίτερα ρεαλιστικό, γιατί βασίζεται αποκλειστικά στο σχήμα του και σε γραφικά δύο διαστάσεων. Για το λόγο αυτό, θα προσθέσουμε λεπτομέρειες, όπως δέντρα, γρασίδι και φυτά.

Με αντίστοιχο τρόπο όπως προσθέσαμε textures για το έδαφος, προσθέτουμε μοντέλα δέντρων, 2D textures γρασιδιού και μικρά meshes άλλης βλάστησης, στη συγκεκριμένη περίπτωση, λουλούδια, και για άλλη μια φορά, «βάφουμε» και τοποθετούμε τα αντικείμενα αυτά πάνω στο έδαφος μας. Το Unity παρέχει από μόνο του κάποια meshes δέντρων και λεπτομερειών όπως και textures γρασιδιού, αλλά μπορούμε πολύ εύκολα να εισάγουμε και να χρησιμοποιήσουμε ή ακόμα και να δημιουργήσουμε δικά μας.

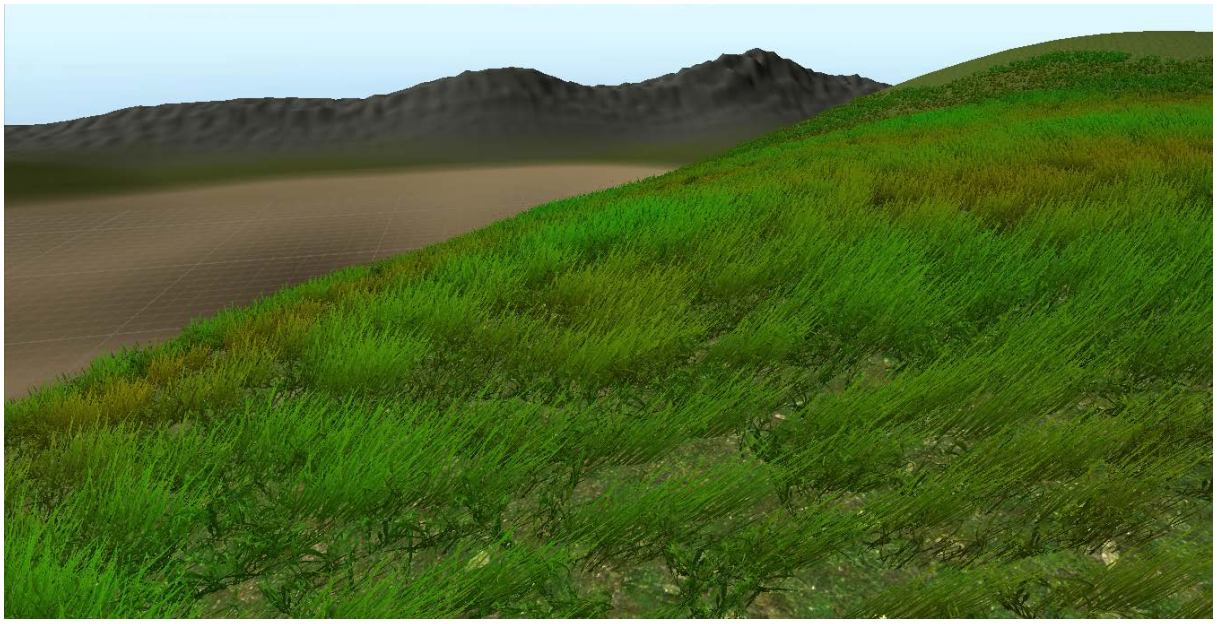

Figure 3.3-5: Terrain Grass

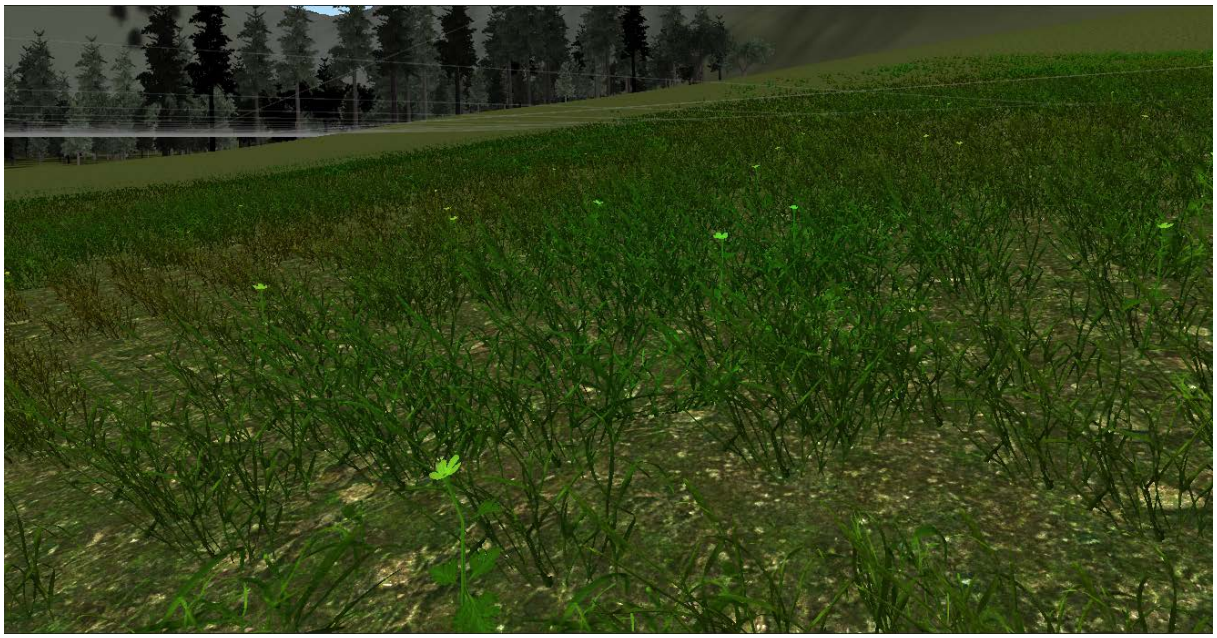

Figure 3.3-6: Terrain Grass και Details

Figure 3.3-7: Terrain Trees

Μέσα από τις ρυθμίσεις του Terrain μπορούμε να ορίσουμε τη διασπορά των αντικειμένων που τοποθετούμε, τις διακυμάνσεις στο μέγεθος τους, την ανάλυση του, κ.α. Συγκεκριμένα για το γρασίδι μπορούμε να ορίσουμε τυχαίες περιοχές όπου αυτό έχει αποχρωματιστεί, καθώς και το αν θα κουνάει σύμφωνα με τον άνεμο του Terrain, για να δώσουμε μια παραπάνω νότα ρεαλισμού.

3.3.2.2 Φωτισμός

Για λόγους ρεαλισμού, επιθυμούμε να έχουμε μια μοναδική πηγή φωτός που να αναπαριστά τον ήλιο στη σκηνή μας. Δημιουργούμε λοιπόν ένα κατευθυνόμενο (directional) φως σε μια μεγάλη απόσταση από το terrain, με intensity 0.55, soft shadows υψηλής ποιότητας, επιλέγουμε να εμφανίζεται το φωτοστέφανο του και του ορίζουμε λάμψη (flare) Sun, από τα standard assets της Unity. Φροντίζουμε επίσης να ευθυγραμμίσουμε τη θέση του φωτός με την απεικόνιση του ήλιου στο skybox μας που θα δούμε αμέσως μετά, ώστε να μην φαίνεται αφύσικο.

Figure 3.3-8: Halo και Sun Flares

3.3.2.3 Ο ουρανός (Skybox και Volumetric Clouds)

Skybox

Το Skybox στη Unity:

Η Unity κάνει χρήση ενός συμβατικού, κυβικού skybox, και μπορεί να οριστεί είτε για μεμονωμένες κάμερες, είτε για ολόκληρο το project. Στη περίπτωση μας, αφού θέλουμε και οι δύο κάμερες να δείχνουν τον ίδιο ουρανό, θα ορίσουμε το Skybox από τις επιλογές Render Settings. Εκεί στο Skybox Material επιλέγουμε το Sunny 2 Skybox.

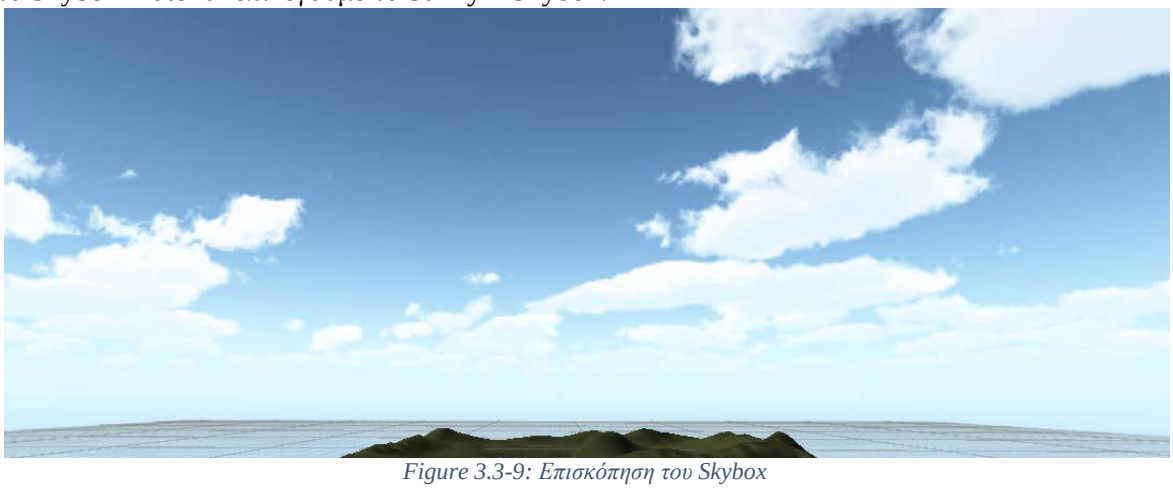

Figure 3.3-9: Επισκόπηση του Skybox

Σύννεφα με όγκο (Volumetric Clouds)

Για περισσότερο ρεαλισμό, θέλαμε και σύννεφα που να μην είναι στάσιμα στο χώρο, κάτι το οποίο αν γίνει μέσω του skybox δεν έχει αισθητικά όμορφο αποτέλεσμα. Χρησιμοποιήθηκε έτσι το asset Cloud System της Edelweiss Interactive για τη δημιουργία σύννεφων με όγκο, που θα είναι κανονικά αντικείμενα στο χώρο. Το συγκεκριμένο asset συνδυάζει γεννήτριες particles μαζί με δικούς του shaders για τη παραγωγή αυτού του αληθοφανούς αποτελέσματος.

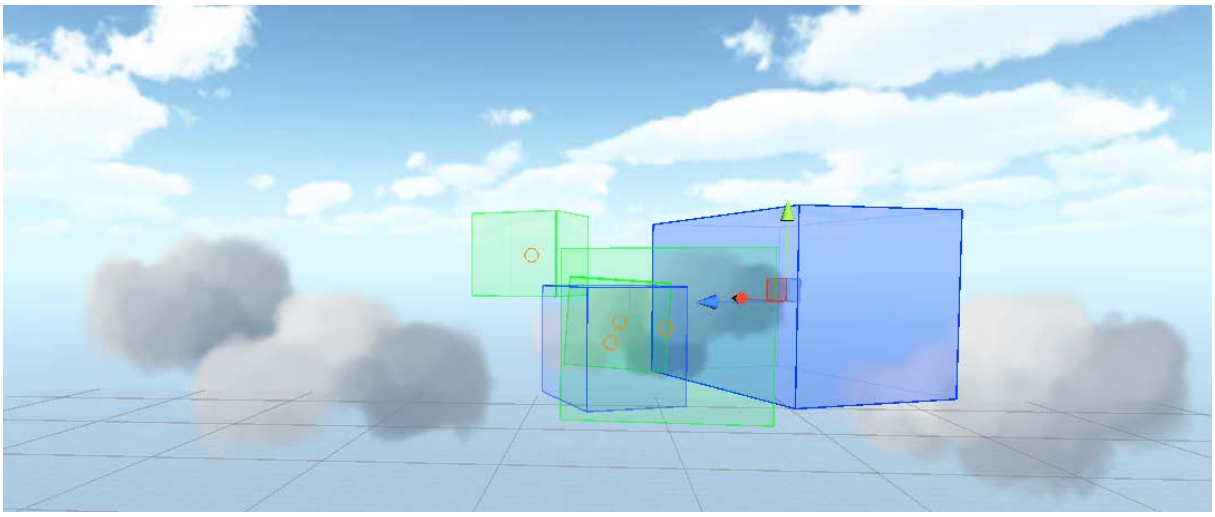

Figure 3.3-10: Οι γεννήτριες particles του Cloud System

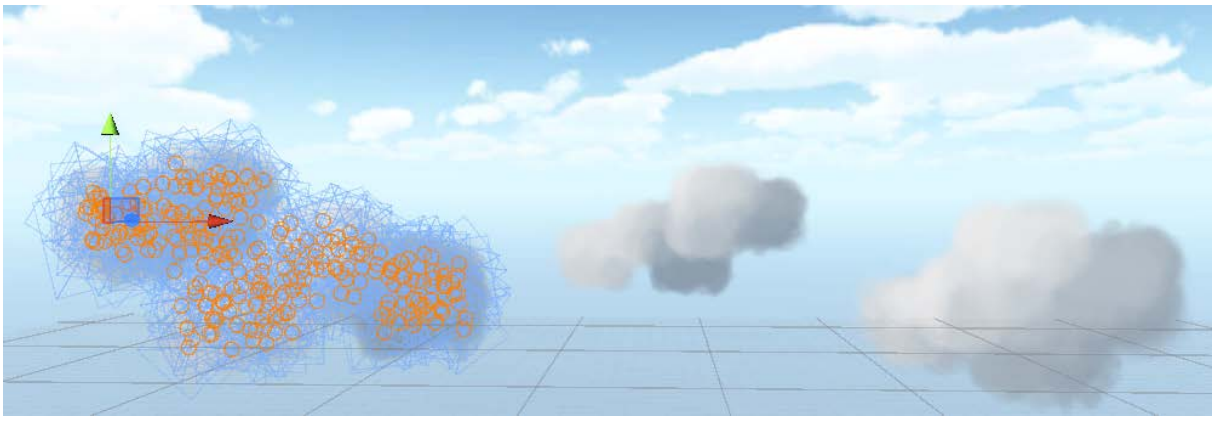

Figure 3.3-11: Τα particles του κάθε σύννεφου του Cloud System

3.3.2.4 Το νερό (SUIMONO Water System)

Το βασικό αντικείμενο νερού της Unity δεν καλύπτει τις ανάγκες ρεαλισμού μας για την πτυχιακή, οπότε στραφήκαμε στο SUIMONO. Πρόκειται για ένα asset του Unity δημιουργημένο από την εταιρία «tanuki digital». Προσφέρει ένα σύνολο εργαλείων και προκατασκευασμένων αντικειμένων για την αναπαράσταση φυσικού, ρεαλιστικού νερού. Προσφέρει επίσης επιλογές νερού με προκαθορισμένες ρυθμίσεις, αλλά όπως και κάναμε, μπορούν να τροποποιηθούν για να ταιριάξουν στη σκηνή μας.

Το SUIMONO χρειάζεται δύο αντικείμενα στη σκηνή για να δουλέψει σωστά, το SUIMONO Module που περιέχει τις ρυθμίσεις και τα script που δίνουν την εμφάνιση στο νερό, και το SUIMONO Surface που είναι η ίδια η επιφάνεια που αναπαριστά τη μάζα του νερού. Αφού τοποθετήσουμε και τα δύο στη σκηνή ρυθμίζουμε από το Module την εμφάνιση του νερού ως εξής:

Figure 3.3-12: Παράδειγμα ρυθμίσεων του SUIMONO

Η υλοποίηση του νερού μας βασίστηκε πάνω σε ένα τροποποιημένο, προκαθορισμένο μοντέλο νερού το «light blue wave» του SUIMONO που με τις κατάλληλες ρυθμίσεις μας έδωσε αυτό το αποτέλεσμα:

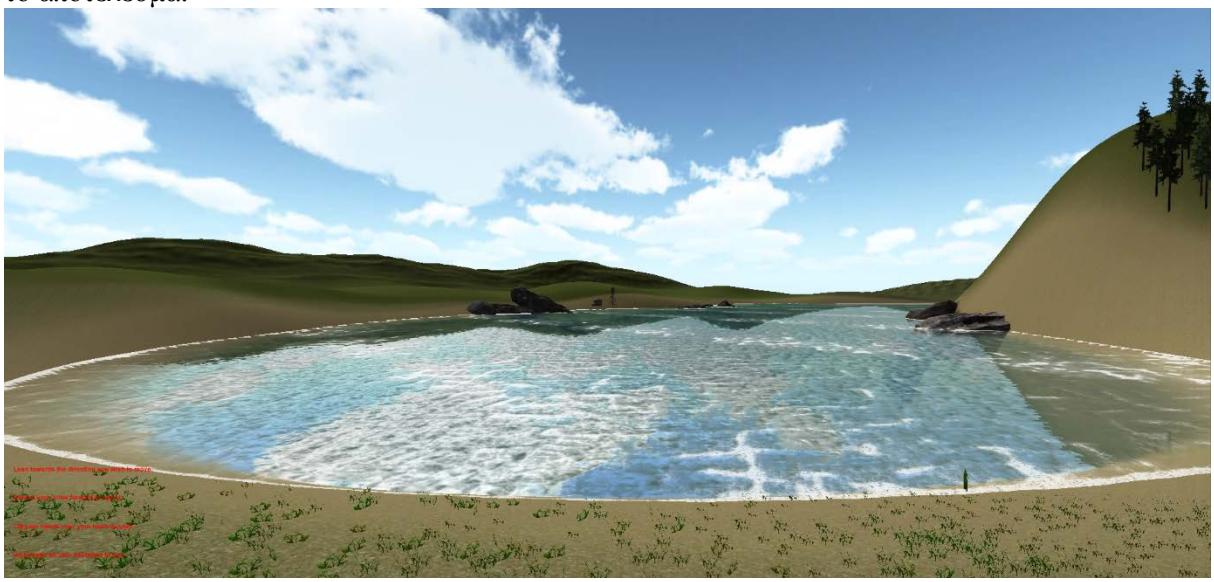

Figure 3.3-13: Τροποποιημένο SUIMONO Interactive Water Systems

3.3.2.5 Επιπλέον στοιχεία (Πέτρες, κτίρια)

Ακόμα το τοπίο μας φαίνεται άδειο, οπότε επιλέγουμε να χρησιμοποιήσουμε έτοιμα μοντέλα κτιρίων και βράχων για να του δώσουμε λίγη σύσταση. Το asset με τις πέτρες λέγεται Rocks Pack – Freebies, του Nobiax, και τα δύο είδη κτιρίων είναι της σειράς Shanty Town της Unity Technologies. Και τα δύο αυτά assets διατίθενται δωρεάν στο Asset Store.

Figure 3.3-14: Τελικό περιβάλλον

3.3.2.6 Βελτιστοποίηση (Optimization)

Αφού έχουμε τελειώσει με το περιβάλλον μας χρειάζεται να προβούμε σε ορισμένες ενέργειες που θα βελτιστοποιήσουν την απόδοση της εφαρμογής χωρίς να μειώσουν τη ποιότητα παραγωγής. Οι δύο αυτές ενέργειες είναι το «lightmapping» και το «batch static draw calls».

LightMapping:

Το Lightmap είναι μια διαδικασία κατά την οποία προϋπολογίζονται οι σκιές των ακίνητων αντικειμένων μιας σκηνής και προβάλλονται ως texture πάνω στα αντικείμενα που δέχονται τη σκίαση. Η Unity μας δίνει διάφορες επιλογές για το τι θα κάνουμε lightmap όπως αν θα δημιουργήσουμε lightmap μόνο του εδάφους, ορισμένων μόνο αντικειμένων κ.α. Για τη μέγιστη δυνατή απόδοση και λόγω του ότι έχουμε μόνο ένα φως στη σκηνή, επιλέγουμε να δημιουργήσουμε το lightmap πάνω σε αυτό. Από το μενού Window εμφανίζουμε το παράθυρο Lightmapping, επιλέγουμε το φως μας και οποιεσδήποτε άλλες ρυθμίσεις επιθυμούμε και πατάμε Bake Scene. Στη συγκεκριμένη περίπτωση χρησιμοποιήσαμε Directional Lightmaps με High Quality.

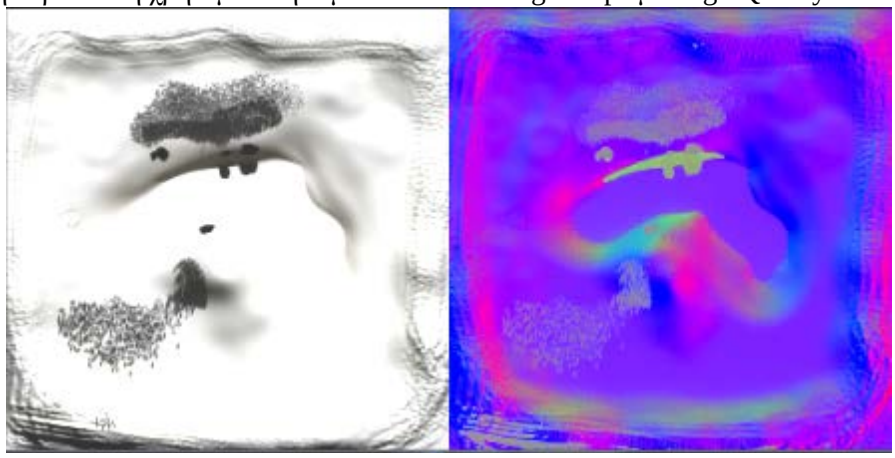

Figure 3.3-15: Το Lightmap του terrain

Batch Draw Calls:

Το πλήθος των Draw Calls μας δείχνει πόσες φορές το δευτερόλεπτο καλείται ο renderer να εμφανίσει κάτι. Μπορούμε να ορίσουμε στοιχεία να εμφανίζονται στη ίδια «παρτίδα», οπότε να εμφανίζονται σε κοινό draw call, και η Unity αυτομάτως κάνει batch όλα τα αντικείμενα μιας σκηνής

τα οποία έχει ορίσει ο χρήστης ως στατικά. Επιλέγουμε λοιπόν το έδαφος, το νερό, τα κτίρια και τις πέτρες και τσεκάρουμε την επιλογή static στον Inspector.

3.3.3 Εισαγωγή του χαρακτήρα

Προτού εισάγουμε τον χαρακτήρα μας μέσα στο Unity, επαληθεύουμε ότι έχουμε ό,τι χρειαζόμαστε. Το βασικό αρχείο του 3ds Max που δημιουργήσαμε πρέπει να έχει το skeletal rig, τα meshes του χαρακτήρα και των υπόλοιπων στοιχείων, και το skin modifier σε κάθε ένα. Πρέπει επίσης να έχουμε και τα texture και normal maps. Τα maps του χαρακτήρα και των ρούχων εξήχθησαν από το ZBrush με το Multi Map Exporter και επεξεργάστηκαν στο Adobe Photoshop για διόρθωση πιθανών ατελειών. Στο Photoshop επίσης προστέθηκε το texture του ματιού στο map που περιλαμβάνει και τα παπούτσια καθώς επίσης δημιουργήθηκε και το texture των μαλλιών.

Figure 3.3-16: Το texture των μαλλιών

Το μοντέλο του χαρακτήρα μας εξήχθη από το 3ds Max σε μορφή .fbx, αρχικά χωρίς animation, με το όνομα biped_male.fbx, και έγινε import ως asset μέσα στο Unity. Από τα Import Options ορίσαμε scale factor στο 0.01 και σύραμε το αντικείμενο πάνω στη σκηνή. Αυτή τη στιγμή είναι λευκό, οπότε εισάγουμε ως asset και τα textures. Επιλέγουμε ξεχωριστά τα αντικείμενα που απαρτίζουν το χαρακτήρα και στα οποία έχουμε ορίσει τα maps μας (σώμα, ρούχα, παπούτσια-μάτια, μαλλιά) και στο κάθε ένα θέτουμε στο inspector το texture που βγάλαμε από το ZBrush και το Photoshop. Στα ρούχα ορίσαμε Bumped Diffuse shader, αφού θέλουμε να κάνουμε χρήση normal map, και στο group μάτια-παπούτσια ορίσαμε Specular Bumped Diffuse Shader, αφού θέλουμε επιπλέον το υλικό να γυαλίζει στο φως. Για τα μαλλιά χρησιμοποιήθηκε Transparent/Cutout/Specular shader. Ο τύπος Transparent εμφανίζει διαφάνεια και το Cutout ορίζει ότι θα υπάρχουν διαφανή αλλά και αδιαφανή τμήματα. Το Specular τμήμα του shader μας δίνει όπως και προηγουμένως μια ελαφριά κατοπτρική λάμψη, όπως θα ήταν φυσικό σε μαλλιά κάτω από τον ήλιο.

Για το δέρμα αρχικά χρησιμοποιήθηκε απλό Bumped Specular shader καθώς θέλαμε απλά να δώσουμε τη λεπτομέρεια από τα normal maps και να κάνουμε το δέρμα να γυαλίζει ελαφρώς, αλλά αντιμετωπίσαμε ένα αναπάντεχο πρόβλημα με το animation κατά το οποίο σε ορισμένες κινήσεις φαινόταν το δέρμα μέσα από τα ρούχα. Βρέθηκε τότε μια απλή, προσωρινή λύση κατά την οποία όταν θέλουμε να εμφανίσουμε το μοντέλο μας μαζί με ρούχα, αλλάζουμε το texture map του σώματος σε ένα αντίγραφο του αρχικού, αλλά με διαφάνεια στα σημεία που προεξείχαν, και για αυτό το λόγο χρησιμοποιήσαμε τελικά το Transparent/Cutout/Bumped Specular. Το αποτέλεσμα φαίνεται φυσικό και δεν προδίδει τον τρόπο με τον οποίο επετεύχθη. Μια πιο μόνιμη, αλλά πολυπλοκότερη και πιο βαριά σε απαιτήσεις λύση θα ήταν αντί για skinned ρούχα, να έχουμε cloth simulation πάνω στις κινήσεις του σώματος. Τέλος, αναθέτουμε στο αντικείμενο του χαρακτήρα και ένα Audio Source component με το οποίο αργότερα θα προσθέσουμε ήχους βημάτων.

Figure 3.3-17: Ο imported χαρακτήρας με textures και Audio Source

3.3.4 Mecanim και State Machines – Διαχειριστής κινήσεων

3.3.4.1 Δημιουργία του Avatar

Αρχικά, κύριο μας μέλημα είναι να στοχεύσουμε το σκελετό που δημιουργήσαμε στο 3ds Max πάνω στο Mecanim Avatar. Από τα import options του μοντέλου, επιλέγουμε την καρτέλα Rig και ορίζουμε το Animation Type ως Humanoid. Στο Avatar Definition επιλέγουμε Create From This Model, καθώς αυτό θα είναι το βασικό Avatar πάνω στο οποίο θα λειτουργήσουν τα animations. Εφαρμόζουμε τις επιλογές μας με το κουμπί Apply και παρατηρούμε ότι έχει ενεργοποιηθεί ένα καινούριο κουμπί, το Configure… με ένα τικ δίπλα του. Αυτό δηλώνει ότι το Unity πέτυχε μια αυτόματη χαρτογράφηση των οστών μας πάνω στο Avatar. Για παν ενδεχόμενο, ελέγχουμε το αποτέλεσμα και διορθώνουμε ότι αναγνωρίστηκε λανθασμένα. Στο παράδειγμα μας, Οι δείκτες και οι αντίχειρες είχαν αντιστοιχηθεί ανάποδα, οπότε σύραμε τα σωστά οστά αυτή τη φορά στις κατάλληλες θέσεις του Avatar.

Figure 3.3-18: Λεπτομέρεια του Avatar

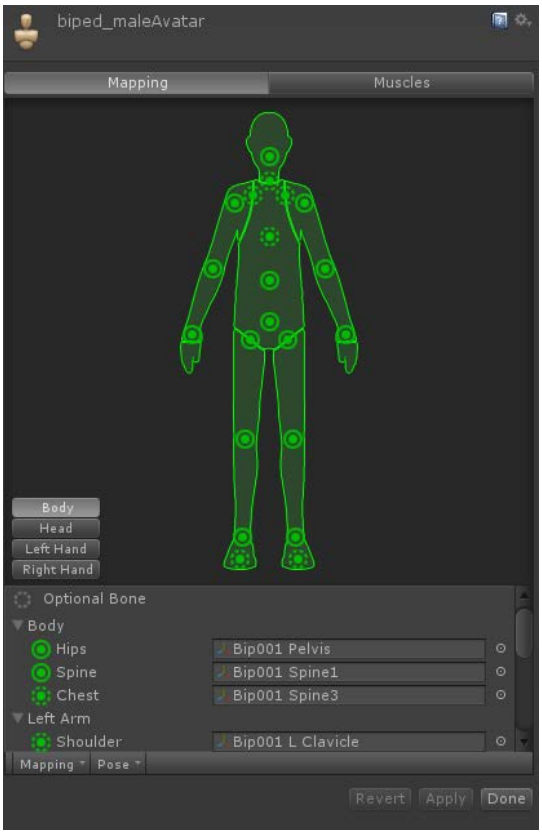

Figure 3.3-19: Το Avatar του χαρακτήρα

Σειρά έχει να ορίσουμε και τα animations να χρησιμοποιούν το ίδιο avatar. Η Unity έχει δύο τρόπους να εισάγει εξωτερικά animations, είτε με όλα τα animations σε ένα αρχείο, είτε με κάθε animation σε ξεχωριστό. Το 3ds Max δεν υποστηρίζει εγγενή διαχωρισμό των κινήσεων πάνω σε κοινό χρονοδιάγραμμα, οπότε επιλέξαμε τον δεύτερο τρόπο για μεγαλύτερη ελευθερία μεταγενέστερης τροποποίησης των κινήσεων. Όπως αναφέραμε στο κεφάλαιο 1, η Unity ακολουθεί έναν ιδιαίτερο τρόπο ονομασίας όσον αφορά τα animations, οπότε έχοντας αποθηκεύσει τα animations μας σε ξεχωριστά αρχεία, τα εξάγουμε σε .fbx και τα εισάγουμε στη Unity με ονόματα όπως biped_male@Idle, biped_male@Walk και ούτω καθεξής. Τα καινούρια αυτά αντικείμενα δε χρειάζεται να τα τοποθετήσουμε πάνω στη σκηνή, απλά από τα Import Options του καθ' ενός, στην καρτέλα Rig, επιλέγουμε Humanoid Animation και Copy From Other Avatar στο Avatar Definition. Επιλέγουμε το Avatar που δημιουργήσαμε νωρίτερα, και το animation είναι έτοιμο να χρησιμοποιηθεί στη μηχανή καταστάσεων.

3.3.4.2 Δημιουργία των Μηχανών Καταστάσεων

Για να αξιοποιήσουμε τις μηχανές καταστάσεων χρειαζόμαστε ένα αντικείμενο του τύπου Animator, οπότε το δημιουργούμε από το μενού Assets > Create > Animator Controller και το θέτουμε στο μοντέλο μας ως Component. Το αντικείμενο αυτό μας δίνει πρόσβαση στα State Machines όταν ανοίγουμε το παράθυρο του Animator από το μενού Window > Animator. Τα State Machines θυμίζουν στη δομή τους UML, οπότε η λογική είναι να δημιουργήσουμε καταστάσεις-animations στα οποία η μετάβαση θα γίνεται όταν πληρούνται ορισμένες προϋποθέσεις. Ξεκινώντας, κάνουμε δεξί click στην επιφάνεια του Animator και επιλέγουμε Create Empty State την οποία ονομάζουμε Idle. Η κατάσταση αυτή θα εμπεριέχει το Idle animation, την κίνηση δηλαδή που θα κάνει ο χαρακτήρας όταν δεν δέχεται κάποια επίδραση από το χρήστη. Στο πεδίο Motion επιλέγουμε το animation που θέλουμε, και αν το επιθυμούμε μπορούμε να επηρεάσουμε την ταχύτητα αναπαραγωγής, καθώς επίσης και αν θέλουμε να κατοπτρίσουμε την κίνηση στον y άξονα. Η τελευταία αυτή επιλογή θα μας φανεί χρήσιμη όποτε θέλουμε να έχουμε συμμετρικές κινήσεις, όπως τα left και right shuffle τη περίπτωση μας. Με τον ίδιο τρόπο δημιουργούμε μια κατάσταση για κάθε

animation, και με δεξί click στη κατάσταση Any State δημιουργούμε μια μετάβαση (transition) σε κάθε μία.

Για να μπορούμε να ορίσουμε ανά πάσα στιγμή μέσω κώδικα τις καταστάσεις, δημιουργούμε στον Animator μια καινούρια παράμετρο (κάτω αριστερά) με όνομα CharacterState και δίνουμε στα transitions ως έλεγχο διαφορετικές τιμές της παραμέτρου αυτής. Για παράδειγμα, όταν το CharacterState είναι 0, θα πηγαίνουμε στη κατάσταση Idle, όταν είναι 1 στη κατάσταση WalkForward κ.λ.π.

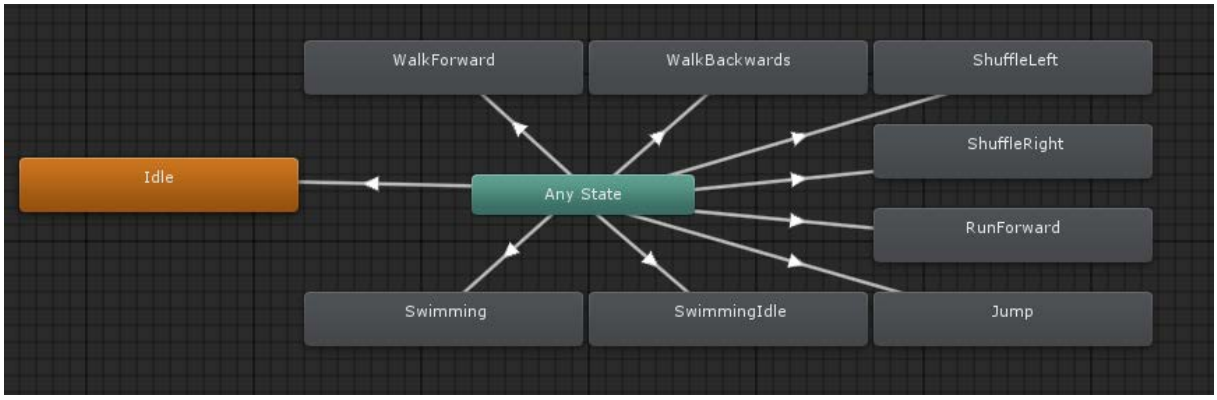

Figure 3.3-20: Τα State Machines του χαρακτήρα

3.3.5 Scripting

Προτού δούμε πώς υλοποιήθηκαν συγκεκριμένες λύσεις με κώδικα, λίγα λόγια για τη C# (C Sharp) μέσα στη Unity:

- Στην αρχή, δηλώνουμε ποιες τεχνολογίες της Unity χρησιμοποιούμε μέσα σε κάθε script με τη σύνταξη using, π.χ. using Unity.Engine;
- Για την ευκολότερη κατανόηση της λειτουργίας του κώδικα, μπορούμε να ορίσουμε απαραίτητα στοιχεία για τη χρήση του script, π.χ. [RequireComponent (typeof (Rigidbody))]
- Όταν δημιουργούμε εξωτερικές μεταβλητές, μπορούμε και μέσα από τον Unity Editor να ορίσουμε τις τιμές τους. Σε περίπτωση που θέλουμε να αποκρύψουμε κάποια μεταβλητή για λόγους ενθυλάκωσης του πηγαίου κώδικα, ορίζουμε τη μεταβλητή με [HideInInspector].
- Υπάρχουν διάφοροι τρόποι να αρχικοποιήσουμε ένα script. Οι δύο πιο συχνές συναρτήσεις που επιτυγχάνουν αυτό το σκοπό είναι η Awake() και η Start().
    - Η Awake() τρέχει τον κώδικα που βρίσκεται μέσα της με το που φορτώνεται το αντικείμενο που περιέχει το script στη σκηνή, ασχέτως αν είναι ενεργοποιημένο ή όχι.
    - Η Start() εκτελείται όταν το ενεργοποιημένο script φορτώνεται στη σκηνή. Οι συναρτήσεις Start() τρέχουν πάντα μετά τις Awake().
    - Αυτό είναι χρήσιμο όταν θέλουμε η αρχικοποίηση ενός script Β να εξαρτάται από την αρχικοποίηση ενός script Α, οπότε το Α θα χρησιμοποιεί Awake() και το Β Start().
    - Άλλες παρόμοιες συναρτήσεις είναι η OnGUI(), OnEnable(), OnCollision() κ.α.
- Αντίστοιχα με τις διάφορες συναρτήσεις αρχικοποίησης έχουμε και διάφορες συναρτήσεις επανάληψης Update στις οποίες γίνεται ο επαναλαμβανόμενος έλεγχος για τις λειτουργίες του παιχνιδιού, όπως η κίνηση του χαρακτήρα, ο έλεγχος για collisions, οι αλλαγές στα animation κ.α. Τα τρία διαφορετικά είδη update είναι τα εξής:
    - Update(): Τρέχει μια φορά σε κάθε frame που περνάει κατά την αναπαραγωγή της σκηνής.
    - FixedUpdate(): Τρέχει μια φορά ανά προκαθορισμένα (fixed) διαστήματα. Το είδος αυτό του update είναι απαραίτητο όταν χρησιμοποιούμε φυσική και δυνάμεις σε κάποιο RigidBody.
    - LateUpdate(): Έχει τη χαμηλότερη προτεραιότητα από τις update και είναι χρήσιμη όταν θέλουμε κάποιο κομμάτι κώδικα να αντιδρά σε άλλα updates.
- Μπορούμε να δημιουργούμε δικές μας συναρτήσεις και να τις καλούμε μέσα στη instantiate ή στην update συνάρτηση.

- Μπορούμε να επηρεάσουμε script διαφορετικών αντικειμένων είτε άμεσα, καλώντας συναρτήσεις τους (για scripts κοινής γλώσσας), είτε έμμεσα στέλνοντας αναγνωρίσιμα μηνύματα με κάποια εντολή για το ξένο script (υποστηρίζει σύνδεση μεταξύ διαφορετικών γλωσσών π.χ. C# σε Javascript, Javascript σε Boo κ.λ.π.

3.3.5.1 Η κίνηση του χαρακτήρα

Για να μπορέσουμε να ελέγχουμε το χαρακτήρα από το πληκτρολόγιο και αργότερα με το Kinect, χρειαζόμαστε ένα script που θα μεταφράζει τις εισόδους που δίνουμε εμείς σε κίνηση και ταυτόχρονα θα θέτει το κατάλληλο CharacterState ώστε να επιλέγει το σωστό animation ανά πάσα στιγμή ο animator και το state machine. Δυστυχώς, ο έτοιμος κώδικας του Unity για έλεγχο χαρακτήρα δεν είχε φτιαχτεί με σκοπό να χρησιμοποιηθεί μαζί με το Mecanim, οπότε στα πλαίσια της πτυχιακής δημιουργήθηκε από την αρχή ένας καινούριος Controller. Το script αυτό καθορίζει τι δυνάμεις θα ασκούνται στο RigidBody του χαρακτήρα όποτε θέλουμε να κάνει κάποια κίνηση και μεταβάλλει την τιμή του CharacterState ανάλογα με τη κίνηση αυτή.

```
void SetCharacterFreeMovementAnimation (float verticalAxis, float horizontalAxis, ref float
targetSpeed)
{
	if (jumping) {
		_characterState = CharacterState.Jumping;
	}else if (verticalAxis != 0 || horizontalAxis != 0) {
		if (swimming) {
			_characterState = CharacterState.Swimming;
		} else if (verticalAxis > 0f) {

			if(isRunning){
				targetSpeed =2*runSpeed;
				_characterState=CharacterState.Running;
			}
			else{
				targetSpeed=walkSpeed;
				_characterState = CharacterState.Walking;
			}
		} else if (verticalAxis < 0f) {
			targetSpeed=walkSpeed;
			_characterState = CharacterState.WalkingBack;
		} else if (horizontalAxis < 0f) {
			_characterState = CharacterState.ShuffleLeft;
		} else if (horizontalAxis > 0f) {
			_characterState = CharacterState.ShuffleRight;
		}
	} else if (swimming) {
		_characterState = CharacterState.SwimmingIdle;
	}else{
		targetSpeed=0;
		_characterState = CharacterState.Idle;
	}

	//this line sets the CharacterState parameter on the character model's animator controller
every frame
	animController.SetInteger("CharacterState", (int)_characterState);
	}
```

Figure 3.3-21: Η συνάρτηση που δίνει τιμή στο CharacterState

```
void FixedUpdate ()
	{
		SetCharacterFreeMovementAnimation (vAxis, hAxis, ref speed);
		if (grounded || swimming) {
			Vector3 targetVelocity = new Vector3 (hAxis, 0, vAxis);
			targetVelocity = transform.TransformDirection (targetVelocity);
			targetVelocity *= speed;
			velocity = rigidbody.velocity;
			Vector3 velocityChange = (targetVelocity - velocity);
velocityChange.x = Mathf.Clamp (velocityChange.x, -maxVelocityChange, maxVelocityChange);
velocityChange.z = Mathf.Clamp (velocityChange.z, -maxVelocityChange, maxVelocityChange);
			velocityChange.y = 0;
			rigidbody.AddForce (velocityChange, ForceMode.VelocityChange);

			}
			else {
				rigidbody.useGravity=false;
			}

		if (!swimming) {
			rigidbody.AddForce (new Vector3 (0, -gravity * rigidbody.mass, 0));
				}
		grounded = false;

		if (hAxis != 0) {
			transform.Rotate(0,rotationSpeed*hAxis,0,Space.Self);
		}
	}
```

Figure 3.3-22: Η κίνηση του χαρακτήρα

3.3.5.2 Camera Smooth Follow με Linecast

Για τις ανάγκες της εφαρμογής χρειάστηκε να δημιουργήσουμε μια καινούρια συμπεριφορά της κάμερας που δεν προϋπήρχε μέσα στη Unity. Θέλουμε η κάμερα να ακολουθεί ομαλά τον παίκτη, αλλά να μην μπαίνει μέσα σε αντικείμενα που βρίσκονται μεταξύ της κάμερας και του μοντέλου του χαρακτήρα. Για να το πετύχουμε αυτό χρησιμοποιήσαμε την τεχνική του Linecast. Το Linecast είναι μια νοητή ευθεία με αφετηρία και τέλος, η οποία εντοπίζει collisions με αντικείμενα μεταξύ των δύο αυτών σημείων. Στη περίπτωση μας, εξετάζουμε ένα Linecast από το μετασχηματισμό του χαρακτήρα ως την κάμερα και αν εντοπιστεί κάποιο εμπόδιο μεταξύ κάμερας και μοντέλου, μετακινούμε την κάμερα στο σημείο από όπου προήλθε το collision. Δημιουργούμε λοιπόν μια κάμερα-παιδί του χαρακτήρα και της θέτουμε το παρακάτω script.

```
using UnityEngine;
using System.Collections;

public class CameraCollision : MonoBehaviour {

	public float minDistance = 1.0f;
	public float maxDistance = 4.0f;
	public float smooth = 10.0f;

	Vector3 dollyDir;
	float distance;

	void Awake()
	{
		dollyDir = transform.localPosition.normalized;
		distance = transform.localPosition.magnitude;
	}

	void Update()
	{
	Vector3 desiredCameraPos = transform.parent.TransformPoint( dollyDir * maxDistance );

		RaycastHit hit;
		if( Physics.Linecast( transform.parent.position, desiredCameraPos, out hit ) )
		{
			distance = Mathf.Clamp( hit.distance, minDistance, maxDistance );
		}
		else
		{
			distance=maxDistance;
		}
		transform.localPosition=Vector3.Lerp(transform.localPosition, dollyDir * distance,
	Time.deltaTime * smooth);
	}
}
```

Figure 3.3-23: CameraCollision.cs

Πρέπει να σημειώσουμε ότι το παρών script εντόπιζε collision με τα μαλλιά του παίκτη, λόγω του ότι πρόκειται για διαφορετικό αντικείμενο, και μας έδινε λανθασμένη κίνηση. Αυτό που κάναμε για να το διορθώσουμε είναι να φτιάξουμε ένα κενό object, παιδί του χαρακτήρα, τοποθετημένο λίγο πίσω του, και να κάνουμε το object αυτό parent της κάμερας. Έχουμε έτσι το ίδιο αποτέλεσμα με τις ίδιες λειτουργίες.

3.3.5.3 Swim Trigger

Για τον έλεγχο του πότε ο χαρακτήρας μας βρίσκεται σε θέση να αρχίσει να κολυμπάει, χρειάστηκε ένα box με trigger collider, έναν collider δηλαδή που δεν είναι εμπόδιο, αλλά αναγνωρίζει πότε κάποιο άλλο αντικείμενο εισέρχεται σε αυτό. Τοποθετήσαμε το box αυτό λίγο πιο κάτω από την επιφάνεια του νερού, και του επικολλήσαμε ένα script που όποτε βρει collision με το μοντέλο του χαρακτήρα, στέλνει μήνυμα στον CustomController ώστε να αρχίσει (ή να σταματήσει) να κολυμπάει.

```
using UnityEngine;
using System.Collections;

public class CheckTrigger : MonoBehaviour {
	public CustomController player;

	void OnTriggerExit(Collider other){
		if (other.gameObject.tag == "Player") {
			player.IsSwimming(false);
		}
	}
	void OnTriggerStay(Collider other){
				if (other.gameObject.tag == "Player") {
					player.IsSwimming (true);
				}
		}
}
```

Figure 3.3-24: CheckTrigger.cs

3.3.6 Ειδικά Εφέ

3.3.6.1 Post Processing Image Effects

Η Unity μας δίνει τη δυνατότητα να προσθέσουμε ειδικά εφέ στην κάμερα, τα οποία υπολογίζονται μετά το render της σκηνής. Τα εφέ που χρησιμοποιήσαμε είναι το Vignetting, το Depth of Field Scatter και το Bloom and Lens Flares. Όλα τα Image Effects είναι απλά στη χρήση τους: δεν έχουμε παρά να τα ορίσουμε ως component της κάμερας και να τα ρυθμίσουμε ώστε να ταιριάζουν στις ανάγκες μας.

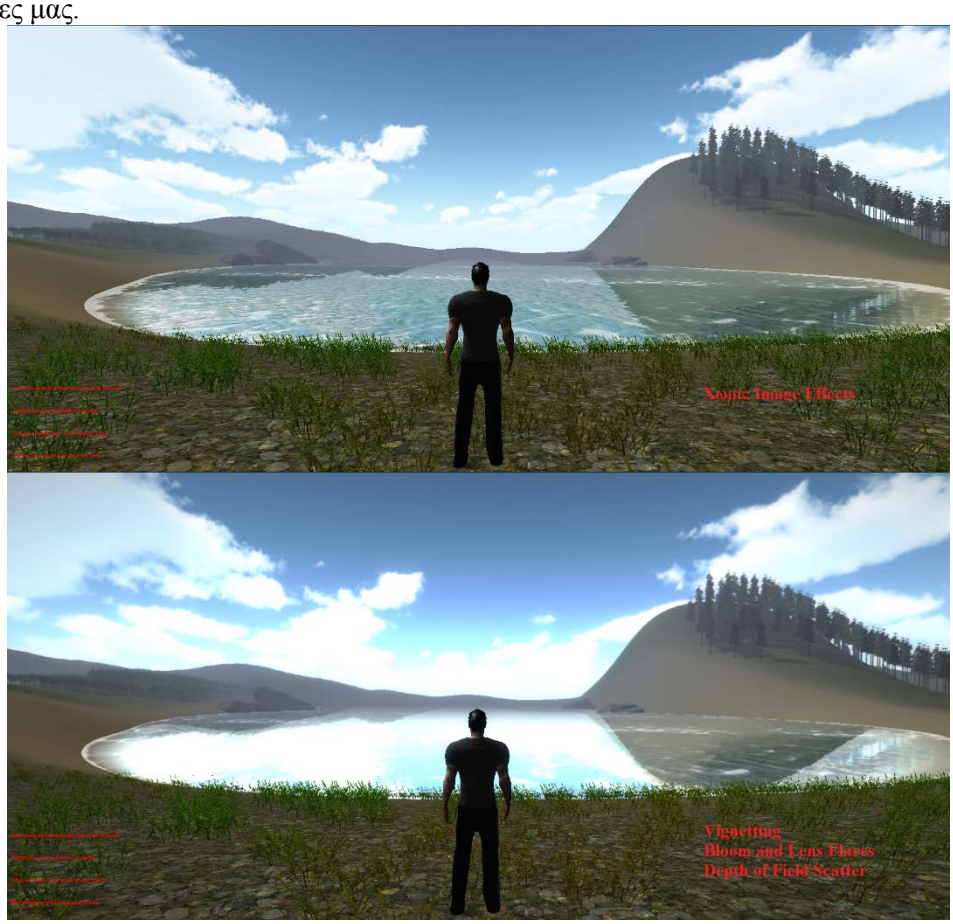

Figure 3.3-25: Σύγκριση χωρίς και με ειδικά εφέ

Μια μικρή δυσκολία που παρουσιάστηκε είναι ότι το σύστημα SUIMONO χρησιμοποιεί τα δικά του εφέ για να αναπαραστήσει το υποθαλάσσιο περιβάλλον, όταν η κάμερα βρίσκεται κάτω από την επιφάνεια του νερού, αυτό όμως αναδεικνύει ένα προγραμματιστικό λάθος που υπάρχει στην τελευταία έκδοση της μηχανής Unity, κατά το οποίο όταν ορίζουμε δύο φορές το ίδιο εφέ, η εικόνα μας περιστρέφεται κατά 360^{o} μοίρες και παραμορφώνεται.

Για να διορθώσουμε το πρόβλημα αυτό, τροποποιήσαμε τον κώδικα που ελέγχει τη κίνηση της κάμερας, ώστε να απενεργοποιεί τα εφέ όταν βρίσκεται μέσα στο νερό.

```
[HideInInspector] public  Vignetting vign;
[HideInInspector] public  DepthOfFieldScatter dofs;
[HideInInspector] public  BloomAndLensFlares balf;
.
.
.
vign = gameObject.GetComponent<Vignetting> ();
dofs = gameObject.GetComponent<DepthOfFieldScatter> ();
balf = gameObject.GetComponent<BloomAndLensFlares> ();

if(transform.position.y<= -14.5){
                        vign.enabled=false;
                        dofs.enabled=false;
                        balf.enabled=false;
        }else{
                        vign.enabled=true;
                        dofs.enabled=true;
                        balf.enabled=true;
                }
```

Figure 3.3-26: Η αναγκαία τροποποίηση στον κώδικα του CameraCollision.cs

3.3.6.2 Particle System

Particles στη πτυχιακή:

Όπως αναφέρθηκε νωρίτερα, έχουμε ήδη χρησιμοποιήσει particles σε προηγούμενα στάδια, όπως είναι τα σύννεφα του Cloud System και ο αφρός του νερού του SUIMONO, αλλά τώρα θα εξετάσουμε ένα μικρό σύστημα που αποτελείται αποκλειστικά από particles και σχηματίζει ένα επίπεδο σκόνης που αιωρείται στον αέρα.

Αρχικά χρειαζόμαστε μια γεννήτρια σωματιδίων, και επιλέγουμε να είναι του τύπου Ellipsoid Particle Emitter. Στη συνέχεια ρυθμίζουμε την εκπομπή των σωματιδίων. Δεν υπάρχει κάποιος συγκεκριμένος τρόπος για να το κάνουμε αυτό, καθώς εξαρτάται από το αποτέλεσμα που θέλουμε να πετύχουμε. Για να δημιουργήσουμε τη σκόνη μας, χρησιμοποιήσαμε τις εξής ρυθμίσεις:

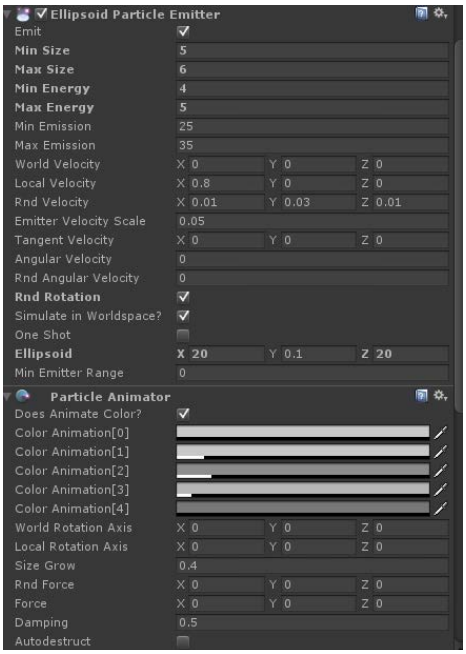

Figure 3.3-27: Οι ρυθμίσεις του Ellipsoid Particle Emitter

Αφού έχουμε ορίσει ακριβώς πως θα δημιουργούνται και θα κινούνται τα particles ορίζουμε και το πώς θα φαίνονται χρησιμοποιώντας την παρακάτω εικόνα ως texture.

Figure 3.3-28: Το texture του κάθε particle

Figure 3.3-29: Τα particles εν δράσει

3.3.6.3 Ήχοι

Όσο ρεαλιστικά και αν φτιάξουμε τα γραφικά, η εφαρμογή μας θα παραμένει μακριά από το στόχο μας αν δεν υπάρχουν ήχοι για να μας κάνουν να νιώθουμε σαν να είμαστε μέσα στη σκηνή. Οι ήχοι που θέλουμε να δημιουργήσουμε είναι και περιβαλλοντικοί, ήχοι δηλαδή που ακούμε στην φύση και ήχοι που προκαλούνται από τον παίκτη, όπως το περπάτημα του. Οι ήχοι που θέλουμε να μετακινούνται μπαίνουν ως component σε κάποιο προϋπάρχον αντικείμενο ενώ οι υπόλοιποι μπορούν να προστεθούν σε καινούρια, κατά τα άλλα κενά αντικείμενα. Οι περιβαλλοντικοί ήχοι που χρησιμοποιούμε βρέθηκαν μέσα από το youtube.com και είναι ήχοι δάσους (παίζει κοντά στη περιοχή με τα δέντρα), ήχοι κυμάτων (παίζει κοντά στην ακτή) και ήχος ανέμου (παίζει σε ολόκληρη τη σκηνή). Ορίσαμε επίσης και ένα sound source ως child του χαρακτήρα για τον ήχο των βημάτων, που μέσω του CustomController χρησιμοποιεί δύο διαφορετικά αρχεία ήχου ανάλογα αν ο παίκτης περπατάει ή τρέχει.

```
FixedUpdate(){
.
.
.
PlayAudio();
}
void PlayAudio(){
                if (_characterState == CharacterState.Walking) {
                        if (audio.clip != walk) {
                                audio.Stop ();
                                audio.clip = walk;
                        }
                        if (!audio.isPlaying) {
                                audio.Play ();
                        }

                } else if (_characterState == CharacterState.Running) {
                        if (audio.clip != run) {
                                audio.Stop ();
                                audio.clip = run;
                        }
                        if (!audio.isPlaying) {
                                audio.Play ();
                        }
                } else {
                        audio.Stop ();
                }
        }
```

Figure 3.3-30: Η απαραίτητη τροποποίηση του CustomController.cs για τον έλεγχο του ήχου

3.4 Kinect

3.4.1 Unity Kinect Wrapper

Το σύστημα που δίνει στη Unity τη δυνατότητα να χρησιμοποιήσει το Kinect, δημιουργήθηκε από τον Rumen Fikov (RF Solutions στο Asset Store), ερευνητή στο Research Institute for User Centered Technologies στο Voralberg University of Applied Sciences στην Αυστρία.

Το σύστημα αυτό βρίσκεται από τις 9 Ιουλίου 2014 ως asset στο Asset Store, λέγεται Kinect with MS-SDK, και η απλή του έκδοση διατίθεται δωρεάν. Απαιτεί την έκδοση 4.3.4 του Unity ή νεότερη, και το Kinect SDK εγκατεστημένα.

Το Kinect with MS-SDK παρέχει βασικά αναγνώριση του skeletal rig του Kinect και αναγνώριση ορισμένων χειρονομιών.

3.4.2 Τροποποίηση πτυχιακής για υποστήριξη Kinect

Όταν ξεκίνησε αυτή η πτυχιακή δεν υπήρχε κάποια σύνδεση του Unity με το SDK των Windows. Αντί αυτού, υπήρχε μια υλοποίηση με το OpenNI, ένα ανεπίσημο SDK για το Kinect, στο οποίο πάνω βασιστήκαμε για να μεταφέρουμε αυτές τις λειτουργίες στο MS-SDK. Η αρχική ιδέα για την υλοποίηση της εφαρμογής μας ήταν το σώμα του χαρακτήρα μας μέσα στο παιχνίδι να ελέγχεται απευθείας μέσω του Kinect, δηλαδή, ό,τι κίνηση κάνει ο παίκτης απέναντι από τη κάμερα αντιγράφεται από το μοντέλο του χαρακτήρα.

Στη πράξη, κάτι τέτοιο θα ήταν ανέφικτο για τον ίδιο τον παίκτη αλλά και το τελικό αποτέλεσμα δεν ήταν αισθητικά ωραίο. Επιλέχθηκε έτσι να σχεδιαστεί ένα είδος αναγνώρισης βασικών κινήσεων που να «πυροδοτούν» προκαθορισμένες λειτουργίες του χαρακτήρα. Η αναγνώριση χειρονομιών που προσφέρει το asset Kinect with MS-SDK δεν κάλυπτε τις ανάγκες μας, καθώς δε μπορούσε να δώσει συνεχόμενη είσοδο όπως χρειάζεται σε ένα παιχνίδι. Έτσι προτιμήθηκε να συνδυάσουμε κατά κάποιο τρόπο τις δύο αυτές ιδεολογίες: Ελέγχουμε τη σχετική θέση ορισμένων τμημάτων του σκελετού και αναλόγως αυτής εκτελούμε συγκεκριμένες συναρτήσεις που μετακινούν το χαρακτήρα μας. Για παράδειγμα, αν εντοπιστεί κίνηση του κορμού προς τα εμπρός, ο χαρακτήρας μας αρχίζει να περπατάει, αν ο κορμός γείρει προς τα δεξιά, ο χαρακτήρας στρίβει δεξιά, και ούτω καθεξής.

Με τη κυκλοφορία του Kinect with MS-SDK στο Asset Store, η υλοποίηση μας μεταφέρθηκε πάνω σε αυτό ώστε να προσφέρουμε ένα πιο ολοκληρωμένο πακέτο για περαιτέρω επέκταση των δυνατοτήτων της εφαρμογής σε επόμενες εκδόσεις.

3.4.2.1 Βασική λειτουργία του Wrapper

Το πακέτο του RF Solutions εμπεριέχει script για απευθείας κίνηση ενός μοντέλου σύμφωνα με τη κίνηση του σκελετού του Kinect, έναν Gesture Listener και μια λίστα των gestures και περιγραφή των επιθυμητών θέσεων, ώστε να αναγνωρίζονται από τον αισθητήρα, τον Kinect Wrapper που εκτελεί τη συνομιλία μεταξύ Unity και Microsoft Kinect for Windows SDK, και τον KinectManager, ένα script που διαχειρίζεται όλες τις λειτουργίες του πακέτου. Από αυτά μας ενδιαφέρει ο τρόπος λειτουργίας του KinectWrapper (ίδια λογική με τον δικό μας wrapper που είχε υλοποιηθεί στις αρχές της πτυχιακής) και το KinectManager, το οποίο ενσωματώθηκε στην κύρια κάμερα ως component.

3.4.2.1.1 KinectWrapper.cs

Το script του KinectWrapper περιέχει μια πληθώρα επιλογών και στοιχείων, ως επί το πλείστον για λειτουργίες που είτε δε παρέχονται στη δωρεάν έκδοση του πακέτου, είτε απλά δε χρειαζόμαστε. Μεταξύ αυτών συμπεριλαμβάνονται ρυθμίσεις για αναγνώριση μορφασμών προσώπου, αναγνώριση φωνής, error codes κ.α.

Το σημαντικότερο κομμάτι κώδικα, και το μοναδικό από αυτό το αρχείο που μας ενδιαφέρει είναι κοινό και απαραίτητο σε κάθε wrapper που συνδέει Unity με Kinect.

```
public enum SkeletonJoint
        {
                HEAD = NuiSkeletonPositionIndex.Head,
                NECK = NuiSkeletonPositionIndex.ShoulderCenter,
                SPINE = NuiSkeletonPositionIndex.Spine,  // TORSO_CENTER
                HIPS = NuiSkeletonPositionIndex.HipCenter,  // WAIST

                LEFT_COLLAR = -1,
                LEFT_SHOULDER = NuiSkeletonPositionIndex.ShoulderLeft,
                LEFT_ELBOW = NuiSkeletonPositionIndex.ElbowLeft,
                LEFT_WRIST = NuiSkeletonPositionIndex.WristLeft,
                LEFT_HAND = NuiSkeletonPositionIndex.HandLeft,
                LEFT_FINGERTIP = -1,

                RIGHT_COLLAR = -1,
                RIGHT_SHOULDER = NuiSkeletonPositionIndex.ShoulderRight,
                RIGHT_ELBOW = NuiSkeletonPositionIndex.ElbowRight,
                RIGHT_WRIST = NuiSkeletonPositionIndex.WristRight,
                RIGHT_HAND = NuiSkeletonPositionIndex.HandRight,
                RIGHT_FINGERTIP = -1,

                LEFT_HIP = NuiSkeletonPositionIndex.HipLeft,
                 LEFT_KNEE = NuiSkeletonPositionIndex.KneeLeft,
                LEFT_ANKLE = NuiSkeletonPositionIndex.AnkleLeft,
                LEFT_FOOT = NuiSkeletonPositionIndex.FootLeft,

                RIGHT_HIP = NuiSkeletonPositionIndex.HipRight,
                RIGHT_KNEE = NuiSkeletonPositionIndex.KneeRight,
                RIGHT_ANKLE = NuiSkeletonPositionIndex.AnkleRight,
                RIGHT_FOOT = NuiSkeletonPositionIndex.FootRight,
                END
        };
```

Figure 3.4-1: Αντιστοίχιση των αναγνωρισμένων σημείων σώματος του Kinect σε μεταβλητές

Αυτός ο enumerator αντιστοιχεί το δείκτη των κλειδώσεων του σκελετού του Kinect σε μεταβλητές της επιλογής μας. Με τον τρόπο αυτό, αν θέλουμε να εξετάσουμε οποιοδήποτε οστό θα το κάνουμε για παράδειγμα ως εξής: *player1JointsPos[(int)KinectWrapper.SkeletonJoint.HEAD];*

3.4.2.1.2 KinectManager.cs

Ο KinectManager όπως αναφέραμε, διαχειρίζεται τις λειτουργίες του Kinect στην εφαρμογή μας. Διαθέτει επίσης την επιλογή εμφάνισης των δεδομένων που καταγράφει η συσκευή του Kinect πάνω στην κάμερα μας, το οποίο και επιλέγουμε να ενεργοποιήσουμε. Αυτό το πετυχαίνει μετατρέποντας τα δεδομένα του χρώματος και του βάθους ξεχωριστά σε Texture2d τα οποία εφαρμόζει πάνω σε ένα πλαίσιο της γραφικής διεπαφής χρήστη (GUI).

```
usersMapSize = KinectWrapper.GetDepthWidth() * KinectWrapper.GetDepthHeight();
usersLblTex = new Texture2D(KinectWrapper.GetDepthWidth(),
KinectWrapper.GetDepthHeight());
usersMapColors = new Color32[usersMapSize];
usersPrevState = new ushort[usersMapSize];
usersMapRect = new Rect(cameraRect.width - cameraRect.width * MapsPercentWidth,
cameraRect.height, cameraRect.width * MapsPercentWidth, -cameraRect.height *
MapsPercentHeight);

usersDepthMap = new short[usersMapSize];
usersHistogramMap = new float[8192];
```

Figure 3.4-2: ComputeUserMap()

```
void OnGUI()
  {
                if(KinectInitialized)
                {
            if(ComputeUserMap && (/**(allUsers.Count == 0) ||*/ DisplayUserMap))
            {
              GUI.DrawTexture(usersMapRect, usersLblTex);
            }

                if(ComputeColorMap && (/**(allUsers.Count == 0) ||*/ DisplayColorMap))
                        {
                                GUI.DrawTexture(usersClrRect, usersClrTex);
                        }
                }
  }
```

Figure 3.4-3: OnGUI()

Ο KinectManager επίσης ορίζει τον παίκτη που εντοπίζεται πιο κοντά στο αισθητήρα ως πρώτο παίκτη και τον δεύτερο πιο κοντινό ως δεύτερο παίκτη. Στη περίπτωση μας έχουμε μόνο έναν παίκτη οπότε η λειτουργία μας είναι περιττή, παρ' όλα αυτά, κρατάμε τη δυνατότητα αυτή για ενδεχόμενες μελλοντικές επεκτάσεις στην εφαρμογή.

3.4.2.2 Αλλαγές στον κώδικα του KinectManager.cs και του CustomController.cs

Όπως προαναφέρθηκε η λογική που ακολουθούμε είναι να εντοπίζουμε συγκεκριμένες σχετικές θέσεις σημείων του σώματος, και όταν αυτές έχουν τις τιμές που θέλουμε, να καλούμε συναρτήσεις του script που ελέγχει το χαρακτήρα μας. Το πρώτο πράγμα που πρέπει να κάνουμε είναι να κρατήσουμε το σύστημα αξόνων για κάθε σημείο που μας ενδιαφέρει σε μια καινούρια μεταβλητή για ευκολότερη πρόσβαση. Εφόσον ο παίκτης μετακινείται και άρα συνεχώς μεταβάλλονται οι θέσεις των μελών του, τόσο η καταχώρηση των τιμών των μεταβλητών όσο και ο έλεγχος της θέσης πρέπει να βρίσκονται μέσα στην Update() του KinectManager.cs.

```
Vector3 headPos = player1JointsPos[(int)KinectWrapper.SkeletonJoint.HEAD];
Vector3 hipsPos = player1JointsPos[(int)KinectWrapper.SkeletonJoint.HIPS];
Vector3 handL = player1JointsPos[(int)KinectWrapper.SkeletonJoint.LEFT_HAND];
Vector3 handR = player1JointsPos[(int)KinectWrapper.SkeletonJoint.RIGHT_HAND];
Vector3 footL = player1JointsPos[(int)KinectWrapper.SkeletonJoint.LEFT_FOOT];
Vector3 footR = player1JointsPos[(int)KinectWrapper.SkeletonJoint.RIGHT_FOOT];

        if(handL.y>=headPos.y+jumpThreshold && handL.y>=headPos.y+jumpThreshold){
                                player.Jump();
                        }
                        if(headPos.z<=hipsPos.z-walkThreshold){
                if(handL.z<=hipsPos.z-runThreshold && handR.z<=hipsPos.z -runThreshold){
                                        player.Run();
                                }
                                else{
                                        player.StopRunning();
                                        player.MoveForward();
                                }
                        }else if(headPos.z>=hipsPos.z+(walkThreshold/2)){
                                player.MoveBackwards();
                        }else{
                                player.StopMoving();
                        }
                        if(headPos.x<=hipsPos.x-turnThreshold){
                                player.TurnLeft();
                        }else if(headPos.x>=hipsPos.x+turnThreshold){
                                player.TurnRight();
                        }else {
                                player.StopTurning();
                        }
                }
```

Figure 3.4-4: Οι έλεγχοι των joints

Στη παρούσα φάση που βρισκόμαστε, ο χαρακτήρας έχει προγραμματιστεί ώστε να δέχεται τις κινήσεις του κατευθείαν από το πληκτρολόγιο οπότε χρειάζεται να ανασχεδιαστεί η λογική μας. Μια πρώτη σκέψη που μελετήθηκε ήταν η προσομοίωση πατήματος πλήκτρων του πληκτρολογίου όταν εντοπίζονται οι επιθυμητές κινήσεις, αλλά κάτι τέτοιο δε φαίνεται να υποστηρίζεται από το API του συστήματος. Ακολουθήσαμε έτσι μια διαφορετική τακτική, κατά την οποία περάσαμε τις λειτουργίες που εκτελούνταν, όταν αναγνωριζόταν κάποιο συγκεκριμένο πάτημα πλήκτρου, σε καινούριες συναρτήσεις. Έχοντας όμως υπόψη τον τρόπο λειτουργίας ενός φυσικού τρόπου κίνησης, έγινε ορατή η ανάγκη, σε αντίθεση με τη συμβατική λογική κίνησης ενός χαρακτήρα, να δημιουργηθούν συναρτήσεις και για να σταματάει η εκάστοτε κίνηση, όταν το σώμα βρίσκεται σε μια σχετική στάση ηρεμίας. Ο κώδικας μας λοιπόν διαμορφώθηκε ως εξής.

```
public void Jump(){
	if(!swimming){
		if (canJump) {
				canJump=false;
				jumping=true;
				grounded=false;
	rigidbody.velocity = new Vector3 (velocity.x, CalculateJumpVerticalSpeed (), velocity.z);
				_characterState = CharacterState.Jumping;

			}
		}else{
			rigidbody.AddForce (new Vector3 (0, gravity * rigidbody.mass, 0));
		}
	}
	public void Run(){
		isRunning = true;
	}
	public void StopRunning(){
		isRunning = false;
	}
	public void MoveForward(){
		vAxis = 1;

	}
	public void MoveBackwards(){
		vAxis = -1;
		StopRunning ();
	}
	public void StopMoving(){
		vAxis = 0;
		StopRunning ();
	}

	public void TurnLeft(){
		hAxis = -1;
	}
	public void TurnRight(){
		hAxis = 1;
	}
	public void StopTurning(){
		hAxis = 0;
	}
}
```

Figure 3.4-5: Οι καινούριες συναρτήσεις

Κατά τα άλλα, η κίνηση δουλεύει με τον ίδιο τρόπο όπως και πριν με μικρές μόνο αλλαγές:

- Η κίνηση στο χώρο γίνεται ακόμα με βάση τις τιμές ενός οριζόντιου και ενός κάθετου άξονα, απλά τώρα οι τιμές αυτές δίνονται από εξωτερική συνάρτηση
- Ο «διακόπτης» για το αν ο χαρακτήρας μας τρέχει ή περπατάει είναι μια μεταβλητή Boolean που παίρνει τιμή από τις συναρτήσεις Run() και StopRunning() αντίστοιχα, αντί για το αν πατιέται ένα κουμπί ή όχι.
- Η εντολή άλματος υπέστη τη μικρότερη τροποποίηση, πέρα από μια βελτιστοποίηση στη λογική της και δουλεύει ακόμα σύμφωνα με μια Boolean

μεταβλητή που ελέγχει αν ο χαρακτήρας πατάει στο έδαφος και με το «διακόπτη» που δίνει η συνάρτηση Jump()

Figure 3.4-6: Το παιχνίδι μαζί με το UserMap του παίκτη

Κεφάλαιο 4
Αποτέλεσμα

6.1. Δυνατότητες της εφαρμογής

Τρέχοντας ένας παίκτης το παιχνίδι που δημιουργήσαμε, θα βρεθεί να κοιτά από προοπτική τρίτου προσώπου το χαρακτήρα, ο οποίος βρίσκεται σε μια μικρή κατασκήνωση, σε ένα ξέφωτο μεταξύ δάσους και παραλίας. Γύρω του ακούγονται ήχοι από πουλιά που κάθονται στα κοντινά δέντρα, καθώς και ο άνεμος που φυσάει στο ανοιχτό τοπίο. Κοντά στο χαρακτήρα βρίσκεται και μία φωτιά αλλά και δίπλα από τις σκηνές υπάρχουν αποθέματα ξύλων για αυτήν. Ο παίκτης γέρνει προς τα εμπρός, και ο χαρακτήρας αρχίζει να περπατά! Δίνοντας μια κλίση στο σώμα του, ο παίκτης μπορεί να πει στον χαρακτήρα προς τα πού θέλει να στραφεί, ενώ φέρνοντας τα χέρια του μπροστά, προκαλεί το χαρακτήρα να αρχίσει να τρέχει. Όσο περιφέρεται στο χώρο, ο παίκτης παρατηρεί το περιβάλλον γύρω του: δέντρα διαφόρων μεγεθών προσφέρουν ένα σκιερό τοπίο, γρασίδι στο έδαφος, αλλού διάσπαρτο και αλλού πυκνό, αλλού πράσινο και υγιές και αλλού κίτρινο και ξεραμένο, σύννεφα μετακινούνται στον ουρανό από πάνω του, κύμα σκάει στη παραλία και αχνοφαίνονται κάποια παραιτημένα κτίρια στο βάθος. Ο παίκτης έχει απόλυτη ελευθερία σε ένα μεγάλο περιβάλλον, ακόμα και να κάνει άλματα ή να κολυμπήσει, και όλα αυτά, χωρίς να χρειαστεί να πλησιάσει ποντίκι ή πληκτρολόγιο, παρά μόνο για να ξεκινήσει και να κλείσει την εφαρμογή.

6.2. Δυσκολίες που αντιμετωπίστηκαν

- Σχεδιασμός μοντέλου
    - Το αρχικό μοντέλο είχε γίνει με την τεχνική Box Modeling, η οποία όμως αποδείχτηκε λιγότερο πρακτική στη δημιουργία χαρακτήρων που προορίζονται για animation, οπότε και προτιμήθηκε η τεχνική Edge-Loop Modeling
- Σκελετικό Σύστημα
    - Για ένα μεγάλο διάστημα, η διαδικασία του skeleton rigging και του animation γινόταν στο πρόγραμμα Blender της Blender Foundation. Αποδείχτηκε όμως ότι το Armature σκελετικό σύστημα του Blender δεν υποστηρίζεται από τον διαχειριστή κινήσεων Mecanim του Unity, οπότε χρησιμοποιήσαμε 3ds Max kai το σκελετικό σύστημα Biped.
- Bug με Post Processing Image Effects
    - Υπάρχει ένα λάθος στην Unity που παραμένει μέχρι και την τελευταία έκδοση κατά το οποίο όταν υπάρχουν δύο ίδια εφέ εικόνας ταυτόχρονα πάνω στην ίδια κάμερα, η προβολή αλλοιώνεται και περιστρέφεται κατά 180^{o} στον άξονα z. Αυτό μας δημιούργησε πρόβλημα κατά το κολύμπι του χαρακτήρα, καθώς το πακέτο του νερού εφαρμόζει κάποια δικά του εφέ όταν κολυμπάμε, μερικά από αυτά κοινά με τα προϋπάρχοντα στην κάμερα μας. Για να το αντιμετωπίσουμε, γράψαμε κώδικα που απενεργοποιεί τα συγκεκριμένα εφέ όταν βρισκόμαστε κάτω από το νερό.
- Λογική Χειρονομιών Kinect
    - Όταν αποφασίστηκε να μπορούμε να κινούμε το χαρακτήρα μας εξ ολοκλήρου μέσω του Kinect και όχι μόνο να κουνάμε τα άκρα του, έγινε κατανοητή τόσο η αδυναμία των χειρονομιών (gestures) να μας δίνουν την απαραίτητη συνεχόμενη είσοδο), όσο και της δυσκολίας για τον παίκτη να εκτελεί ακριβώς τις κινήσεις που θέλει να κάνει ο χαρακτήρας. Επιλέχθηκε έτσι μια τροποποίηση στην προηγούμενη λογική, που μας επέτρεψε να χρησιμοποιήσουμε τις σχετικές θέσεις των μελών του σώματος.

6.3. Μελλοντική Εργασία και Επεκτάσεις

Η συγκεκριμένη εφαρμογή παρέκκλινε από τον αρχικό της στόχο ενός παιχνιδιού συγκεκριμένου είδους, και μετατράπηκε σε ένα γενικότερο πλαίσιο στο οποίο πάνω μπορούν να στηθούν διάφορα είδη και λογικές παιχνιδιών. Ήδη έχουν δημιουργηθεί script τόσο για την μετατροπή της εφαρμογής σε fighting game όσο και για την ενσωμάτωση επιπλέον τεχνολογιών.

Οι δυνητικά υλοποιήσιμες εφαρμογές βασισμένες πάνω σε αυτό το framework παρουσιάζουν ιδιαίτερο ενδιαφέρον καθώς θα μπορούσαν να τροποποιηθούν για να καλύψουν ένα εύρος video game genres, όπως fighting, third person action adventure, RPG ή ακόμα και racing. Το Kinect της Microsoft είναι μια πολύ δημιουργική τεχνολογία που συνεχώς εξελίσσεται και η Unity Game Engine εδραιώνεται όλο και περισσότερο ως μια από τις πιο εύχρηστες μηχανές παιχνιδιών, και σίγουρα είναι η καλύτερη όσον αφορά mobile games, έτσι το μόνο που περιορίζει τον εκάστοτε προγραμματιστή είναι η φαντασία του. Άμεσα υλοποιήσιμες λειτουργίες θα μπορούσε να είναι ένα μενού δημιουργίας χαρακτήρα, περισσότερα περιβάλλοντα, εναλλαγή μέρας και νύχτας, εναλλαγή μεταξύ έμμεσου και άμεσου ελέγχου του χαρακτήρα μέσω Kinect, πλοήγηση στα μενού με φωνητικές εντολές κ.α.

Μία συγκεκριμένη κατεύθυνση στην οποία θα στρεφόμασταν χωρίς την πίεση του χρόνου θα ήταν η υποστήριξη multiplayer, είτε με την ίδια συσκευή είτε με απομακρυσμένη σύνδεση δύο διαφορετικών. Ακόμα, σε αρχικά concept στάδια βρισκόταν και μια υλοποίηση που συνδύαζε είσοδο και έλεγχο χαρακτήρων με το Kinect, και σύνδεση με smartphone για ελευθερία επιπλέον επιλογών.

5. Παράρτημα

I. 3ds Max

Ιστορία και εκδόσεις:

Το πρώτο λογισμικό με όνομα «3D Studio» δημιουργήθηκε για λειτουργικό DOS από τον Gary Yost και το Yost Group, και εκδόθηκε από την εταιρία Autodesk. Μετά την έκδοση 4 του 3D Studio DOS, το πρόγραμμα γράφτηκε εκ νέου για την πλατφόρμα Windows NT, και υιοθέτησε το όνομα «3D Studio MAX». Η έκδοση αυτή δημιουργήθηκε επίσης από το Yost Group, αλλά εκδόθηκε από την Kinetix, που ήταν ο τομέας Πολυμέσων και Ψυχαγωγίας της Autodesk.

Η Autodesk αγόρασε τα δικαιώματα του προϊόντος κατά τη δεύτερη έκδοση του καινούριου 3D Studio Max και εσωτερίκευσε την ανάπτυξη εξ ολοκλήρου κατά τη διάρκεια των δύο επόμενων εκδόσεων. Αργότερα, το όνομα του προγράμματος άλλαξε σε «3ds max» για λόγους συμβατότητας με τη πολιτική ονομασία της Discreet, μια εταιρία λογισμικού, βασισμένη στο Montreal, που είχε αγοράσει η Autodesk.

Όταν επανεκδόθηκε (έκδοση 7), το πρόγραμμα άρχισε να χρησιμοποιεί ως λογότυπο το σύμβολο της Autodesk και το όνομα άλλαξε ξανά και έγινε «3ds Max», ενώ η επίσημη ονομασία του προϊόντος έγινε η σύγχρονη «Autodesk 3ds Max». Η νεότερη έκδοση τη στιγμή εγγραφής της πτυχιακής είναι η Autodesk 3ds Max 2015 με ημερομηνία κυκλοφορίας 20 Μαρτίου 2014.

Οι λειτουργίες του 3ds Max:

MaxScript:

Είναι μια ενσωματωμένη scripting γλώσσα που μπορεί να χρησιμοποιηθεί για την αυτοματοποίηση επαναλαμβανόμενων διαδικασιών, τη σύνδεση ήδη υπαρχουσών λειτουργιών για νέες δυνατότητες, την ανάπτυξη καινούριων εργαλείων και διεπαφών χρήσης και πολλά άλλα. Πρόσθετα εργαλεία (plugins) μπορούν να γραφούν εξ ολοκλήρου σε *MaxScript.*

Character Studio:

Το Character Studio ξεκίνησε ως εξωτερικό plugin, όμως από την έκδοση 4 του Max και μετά βρίσκεται ενσωματωμένο στο 3ds Max, επιτρέποντας στους χρήστες να δημιουργήσουν animation για τους εικονικούς τους χαρακτήρες. Το Studio κάνει χρήση ενός σκελετικού συστήματος γνωστό ως «Biped», το οποίο διαθέτει ρυθμίσεις που μπορούν να τροποποιηθούν σύμφωνα με τις ανάγκες των μοντέλων και των animation.

Scene Explorer:

Πρόκειται για ένα εργαλείο που παρέχει στο χρήστη μια ιεραρχική δομή των στοιχείων της 3D σκηνής και βοηθά στη διαχείριση μεγάλων, πολύπλοκων σκηνών.

Import/Export:

Το Autodesk 3ds Max υποστηρίζει ένα μεγάλο εύρος καταλήξεων αρχείου για εξαγωγή και εισαγωγή, πράγμα που καθιστά τη μεταφορά ενός έργου από ένα πρόγραμμα σε άλλο εύκολη. Η αρχική εγκατάσταση του Max υποστηρίζει εξαγωγή σε *.fbx, .3ds, .ai, .ase, .dae, .dwf, .dwg, .dxf, .flt, .htr, .igs, .obj, .pxproj, .sat, .xtl, .w3d, .wire, .wrl,* ενώ μέσω plugins υποστηρίζονται πολλά ακόμη όπως *.x3d και .xhtml.*

Λειτουργίες για texturing:

Το 3ds Max προσφέρει μια πληθώρα εργασιών για τη δημιουργία texture και planar maps, όπως tiling, mirroring, decals, angle, rotate, blur, UV stretching και relaxation, remove distortion, preserve UV και export του UV template. Μέσω της ροής εργασίας των textures δίνονται στο χρήστη πολλές δυνατότητες όσον αφορά τη δημιουργία maps, αλλά και το UVW Unwrap, τη δημιουργία «ραφών» (seams) πάνω στα μοντέλα, ώστε να δίνουμε υφή σε πολύπλοκες τοπολογίες.

Keyframing:

Η διαδικασία δημιουργίας animation (χωρίς εξομοίωση) του 3ds Max κάνει χρήση των keyframes, σταθερά σημεία δηλαδή πάνω σε ένα χρονοδιάγραμμα από frames, με όλα τα ενδιάμεσα σημεία να υπολογίζονται από το πρόγραμμα.

Παραμετροποίησιμη κίνηση με καμπύλες:

Το 3ds Max μας παρέχει τη δυνατότητα να δημιουργήσουμε animation χρησιμοποιώντας συναρτησιακές καμπύλες για τον έλεγχο της θέσης, περιστροφής, μεγέθους, κατεύθυνσης, ταχύτητας κ.α. Είναι δυνατός ο συνδυασμός τέτοιων καμπυλών διαφόρων αντικειμένων ώστε να επιτευχθεί animation αλληλεξαρτώμενων ή χρονικά μεταβαλλόμενων κινήσεων, φυσικά όμως, το τελικό αποτέλεσμα μπορεί να μετατραπεί σε keyframe animation για μεγαλύτερη ελευθερία κινήσεων.

Skinning:

Skinning λέγεται η διαδικασία κατά την οποία ένα 3d μοντέλο αντιστοιχίζεται με ένα σκελετικό σύστημα ώστε να μπορούν να δημιουργηθούν animation πάνω σε αυτό. Για να το πετύχει αυτό το 3ds Max δίνει στους χρήστες δύο εργαλεία, το Physique modifier και το πιο πρόσφατο Skin modifier, το οποίο και χρησιμοποιήθηκε μέσα στα πλαίσια αυτής της πτυχιακής.

Σκελετικό Σύστημα και Inverse Kinematics:

Οι εικονικοί χαρακτήρες μπορούν να εξοπλιστούν με, προσαρμοσμένους από το χρήστη, σκελετούς φτιαγμένους από «οστά» του 3ds Max, Inverse Kinematics Solvers και εργαλεία χρήσης Motion Capture δεδομένων.

Για κάθε οστό ορίζεται μια περιοχή δράσης πάνω στο μοντέλο, και για κάθε σημείο της περιοχής αυτής ένα «βάρος», ένας δεκαδικός αριθμός δηλαδή που ορίζει πόσο επηρεάζει το συγκεκριμένο σημείο η κίνηση του οστού.

Ακόμα, μπορούμε να κάνουμε retarget ένα σκελετικό σύστημα σε ένα άλλο. Αυτό μας δίνει τη δυνατότητα να επαναχρησιμοποιήσουμε animation, αρχικά δημιουργημένο για διαφορετικό χαρακτήρα, ή ακόμα να χρησιμοποιήσουμε έτοιμα animation που προέρχονται από motion capture (καταγραφή ρεαλιστικής κίνησης με κάμερα και/ή sensors.

Ενσωματωμένη μηχανή εξομοίωσης φυσικής υφάσματος:

Το 3ds Max διαθέτει τη δικιά του ενσωματωμένη μηχανή εξομοίωσης φυσικής υφάσματος που επιτρέπει στο χρήστη να μετατρέψει σχεδόν οποιοδήποτε 3D αντικείμενο επιθυμεί σε ρουχισμό, ή να δημιουργήσει ενδύματα από το μηδέν. Η επίλυση συγκρούσεων είναι ταχύτατη και ακριβής ακόμα και σε πολύπλοκες εξομοιώσεις.

II. ZBrush

Σύντομη εισαγωγή:

Το ZBrush δημιουργήθηκε από την εταιρία Pixologic Inc. που ιδρύθηκε από τον Ofer "Pixolator" Alon και τον Jack Rimokh και παρουσιάστηκε για πρώτη φορά το 1999 στο ετήσιο συνέδριο των computer graphics (CG), SIGGRAPH. Η πρώτη demo έκδοση του προγράμματος, το ZBrush 1.55, παρουσιάστηκε το 2002 και ακολούθησαν οι εκδόσεις 3.1 το 2007, 3.5 το 2009 , 4 το 2010 και η τωρινή ZBrush 4R6 το 2013.

Το ZBrush είναι ένα σχεδιαστικό πρόγραμμα για τη δημιουργία 3D αντικειμένων που συνδυάζει τεχνολογίες 3D/2.5D, δημιουργία υφών και χρωματισμό μοντέλων. Το ZBrush εισήγαγε έναν καινούριο ορισμό, το «pixol», που αποθηκεύει φωτισμό, χρώμα, υλικό και πληροφορίες βάθους για όλα τα αντικείμενα πάνω στη σκηνή. Η κύρια διαφορά του ZBrush σε σύγκριση με άλλα, πιο παραδοσιακά προγράμματα μοντελοποίησης είναι ότι η λειτουργία του μοιάζει περισσότερο με γλυπτική.

Το ZBrush χρησιμοποιείται για τη δημιουργία μοντέλων υψηλής ανάλυσης, της τάξης των δεκάδων εκατομμυρίων πολυγώνων, για χρήση σε ταινίες, παιχνίδια και animated films, από εταιρίες όπως η ILM, η Electronic Arts κ.α. Το ZBrush κάνει χρήση δυναμικών επιπέδων ανάλυσης ώστε να επιτρέπει στους «γλύπτες» να κάνουν από πολύ γενικές μέχρι πολύ λεπτομερείς αλλαγές στα μοντέλα τους. Ένα από τα δυνατά σημεία του ZBrush είναι ότι έδωσε τη δυνατότητα δημιουργίας ενός επιπέδου λεπτομερειών που πρωτύτερα περιορίζονταν σε bump maps. Τα προκύπτοντα στοιχεία του μοντέλου μπορούν να εξαχθούν ως normal maps και να χρησιμοποιηθούν σε ένα μοντέλο με αντιστοιχία στην τοπολογία, αλλά μικρότερο κατά πολύ αριθμό πολυγώνων, εμφανίζοντας έτσι έναν υψηλό βαθμό λεπτομέρειας, χωρίς να αυξάνουμε τις απαιτήσεις του συστήματος σε υπολογιστική ισχύ.

Ακόμα, με την τεχνολογία GoZ ("Go ZBrush") που βρίσκεται μέσα στο πρόγραμμα από την έκδοση 3.2 OSX, το ZBrush προσφέρει άμεση συνδεσιμότητα και ενσωμάτωση με τα Autodesk Maya, Autodesk 3ds Max, Cinema4D, LightWave 3D, Poser Pro, Daz Studio, EIAS και Modo.

Χαρακτηριστικά εργαλεία:

Pixol

Όπως και τα pixel, κάθε pixol εμπεριέχει πληροφορίες για τη θέση του στον άξονα X και Y και τιμές χρώματος. Εμπεριέχει επιπλέον τιμή βάθους (θέση στον άξονα Z), περιστροφή και υλικό. Τα αρχεία του ZBrush κρατάνε pixol στοιχεία, αλλά όταν οι χάρτες εξάγονται, για παράδειγμα σε μορφή JPEG ή PNG), τότε οι πληροφορίες αυτές απλοποιούνται και τα στοιχεία pixol χάνονται. Η τεχνολογία του pixol είναι στη θεωρία παρόμοια με αυτή του voxel, ένα άλλο είδος 3D pixel.

3D Brushes

Η βασική έκδοση του ZBrush προσφέρει 30 3D εργαλεία γλυπτικής με αρκετά ακόμα διαθέσιμα από την εταιρία. Τα πινέλα αυτά διαθέτουν πολλές παραμετροποιήσιμες επιλογές, μεταξύ άλλων τη σκληρότητα, το είδος της πινελιάς, την υφή και την ορατότητα, που αλλάζουν το σχήμα του πινέλου.

Polypaint,

Το Polypainting επιτρέπει στους σχεδιαστές να δώσουν χρώμα και υφή στο μοντέλο, χωρίς να έχουν πρώτα κάνει unwrap τις επιφάνειες του, δίνοντας τους τη δυνατότητα να βάφουν κατευθείαν πάνω στα πολύγωνα.

Illustration

Το ZBrush δίνει τη δυνατότητα σμίλευσης σε 2.5D και διαθέτει διάφορα εργαλεία για να το πετύχει αυτό, αξιοποιώντας το ιδιόκτητα pixol για να αποθηκεύει χρώμα, βάθος, υλικό, θέση και φωτισμό.

Transpose

Το ZBrush επίσης προσφέρει μια λειτουργία παρόμοια του skeletal animation άλλων 3D προγραμμάτων. Το Transpose επιτρέπει στο σχεδιαστή να απομονώσει ένα τμήμα του μοντέλου και να του δώσει τη στάση που επιθυμεί χωρίς να χρησιμοποιεί σκελετικό σύστημα. Το εργαλείο αυτό δεν προορίζεται για animation, αλλά για διαφοροποίηση της σωματικής στάσης των μοντέλων.

ZSpheres

Οι σχεδιαστές μπορούν να δημιουργήσουν ένα βασικό μοντέλο με ομοιόμορφη κατανομή πολυγώνων και loop edges, ώστε να το μετατρέψουν στη συνέχεια σε σμιλεύσιμο αντικείμενο, ξεκινώντας με μια απλή σφαίρα και επεκτείνοντας την με επιπλέον "ZSpheres" μέχρις ότου καταλήξουν στο βασικό σχήμα του επιθυμητού μοντέλου.

GoZ

Κάνοντας την πρώτη του εμφάνιση στην έκδοση ZBrush 3.2 για λειτουργικό OSX, το GoZ αυτοματοποιεί το στήσιμο των UVW Maps των 3D μοντέλων στις εφαρμογές που υποστηρίζουν GoZ. Με την αποστολή του αντικειμένου στο ZBrush, το GoZ θα αντιστοιχίσει αυτομάτως το υπάρχον μοντέλο υψηλής ποιότητας στο καινούριο μοντέλο, και θα φροντίσει ώστε να διορθωθούν τυχόν προβλήματα, όπως η εσφαλμένη διάταξη σημείων και πολυγώνων. Το προϊόν αυτής της διαδικασίας είναι αμέσως έτοιμο για περεταίρω σμίλευση, εξαγωγή maps ή μεταφορά του σε άλλη GoZ εφαρμογή.

Best Preview Render

Το ZBrush διαθέτει ένα ενσωματωμένο σύστημα rendering, το Best Preview Render, που δίνει τη δυνατότητα χρήσης χαρτών περιβάλλοντος 360 μοιρών για το φωτισμό σκηνών με HDRI εικόνες. Ο Best Preview Render (BPR) αξιοποιεί ένα καινούριο σύστημα ελέγχου του φωτισμού που ονομάζεται LightCaps. Χάρη στο σύστημα αυτό, μπορούμε όχι μόνο να ορίσουμε το πώς τα φώτα τοποθετούνται στη σκηνή γύρω από το μοντέλο μας, αλλά και να παράγουμε περιβάλλοντα βασισμένα σε αυτό για μελλοντικό HDRI Render. Επιτρέπει επίσης μεταβολή των υλικών των μοντέλων σε πραγματικό χρόνο και υποστηρίζει ιδιότητες υλικών όπως subsurface scattering (υποδερμική διάθλαση) και περιβαλλοντικές ή scan-line ανακλάσεις.

DynaMesh

Πρόκειται για ένα εργαλείο που παράγει σε μικρό χρονικό διάστημα ένα νέο μοντέλο, αντίγραφο του αρχικού, αλλά με ομοιόμορφη κατανομή πολυγώνων, ώστε να βελτιώσει την τοπολογία και να αφανίσει τυχόν παραμόρφωση και τέντωμα των επιφανειών.

FiberMesh

Το FiberMesh δίνει στους χρήστες τη δυνατότητα να «φυτρώσουν» ίνες-πολύγωνα από τα μοντέλα τους για τη δημιουργία τριχώματος ή φυτικών αντικειμένων. Δίνει επίσης και ένα τρόπο επεξεργασίας και τροποποίησης πολλών πολυγώνων ταυτόχρονα με τις Groom Brushes (χτένες).

ZRemesher

Το ZRemesher (προηγουμένως QRemesher) είναι ένα αυτοματοποιημένο σύστημα αναδημιουργίας της τοπολογίας ενός μοντέλου, βασιζόμενο πάντα στο αρχικό μοντέλο. Η καινούρια τοπολογία είναι σε γενικές γραμμές καθαρότερη και ομοιόμορφη. Η διαδικασία του ZRemesher μπορεί ωστόσο να καθοδηγηθεί από το χρήστη ώστε η νέα τοπολογία να ακολουθεί συγκεκριμένες καμπύλες του μοντέλου και να διατηρεί περιοχές με υψηλά επίπεδα λεπτομέρειας.

Το γραφικό περιβάλλον του ZBrush:

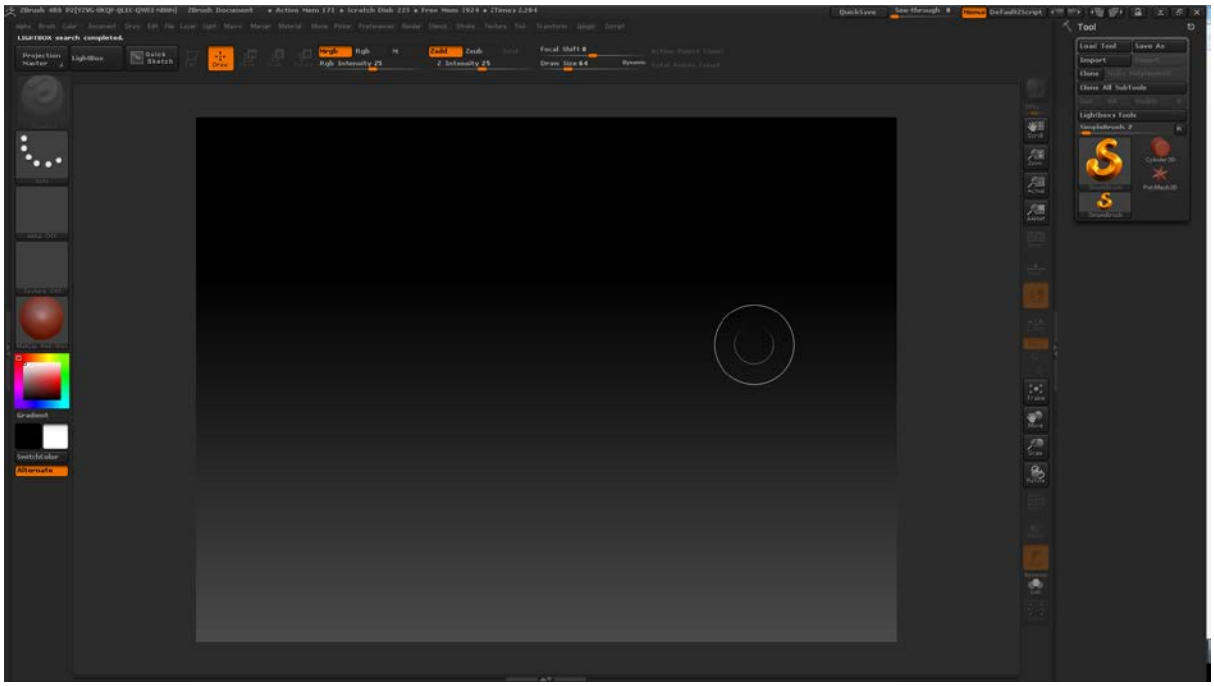

Figure II-1: Το Interface του ZBrush

Στο κέντρο της οθόνης μας έχουμε τον καμβά. Ότι σχεδιάζουμε, είτε σε 3D είτε σε 2.5D εμφανίζεται εδώ. Στη δεξιά του πλευρά βρίσκεται μια κάθετη μπάρα με τις επιλογές του, όπως εργαλεία για τη πλοήγηση του (Frame, Move, Scale, Rotate), για την εμφάνιση συμμετρίας, για την απομόνωση τμημάτων του μοντέλου και άλλα. Στην πάνω πλευρά της οθόνης βρίσκεται το γενικό μενού. Όλες οι λειτουργίες και τα εργαλεία του ZBrush βρίσκονται μαζεμένα εδώ. Στη δεξιά πλευρά της οθόνης, έχουμε μια content sensitive μπάρα, δηλαδή μια λίστα επιλογών ανάλογη με το τι έχουμε επιλεγμένο και το τι κάνουμε εκείνη τη στιγμή. Ακριβώς πάνω και αριστερά από τον καμβά μπορούμε να δούμε τις επιλογές των πινέλων μας, όπως το σχήμα και το χρώμα (αριστερά) και το είδος της λειτουργίας του πινέλου (χρώμα, βάθος, υλικό ή συνδυασμό αυτών), την ένταση και το μέγεθος (πάνω). Πρόσβαση σε αυτές τις επιλογές μας δίνει και το δεξί click πάνω στον καμβά, κάτι ιδιαίτερα χρήσιμο για να μην αποσπάται η προσοχή μας κατά τη σχεδίαση.

Figure II-2: Μενού δεξί click

Φυσικά, η διάταξη του γραφικού περιβάλλοντος του ZBrush είναι απολύτως προσαρμόσιμη στις ανάγκες του κάθε χρήστη.

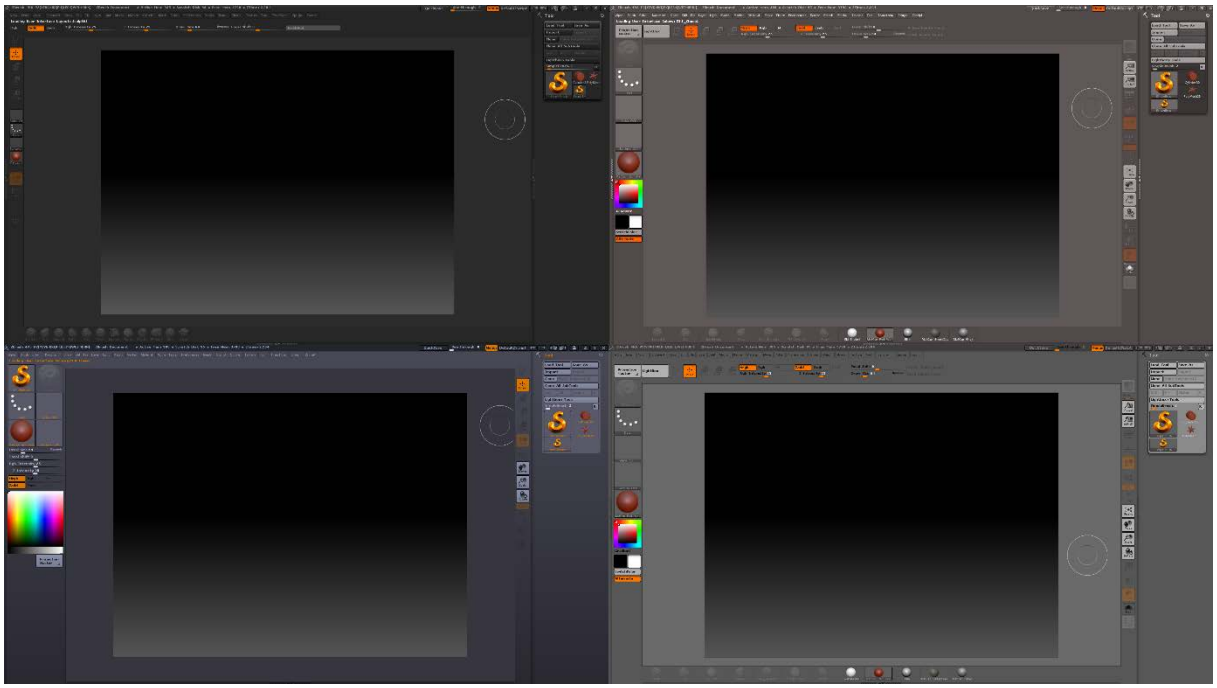

Figure II-3: Παραδείγματα διαφορετικών layout

III. Unity

Γενικά για το Unity

Το Unity είναι ένα σύστημα δημιουργίας παιχνιδιών, με τη δυνατότητα να τρέχουν πανομοιότυπα σε διαφορετικές πλατφόρμες, που αναπτύχθηκε από την Unity Technologies. Συμπεριλαμβάνει game engine και ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment - IDE). Χρησιμοποιείται για την ανάπτυξη video games για web sites, desktop, κονσόλες και κινητές συσκευές. Αρχικά ανακοινώθηκε, στο Worldwide Developers Conference της Apple το 2005, αποκλειστικά για Mac OS. Έκτοτε έχει επεκταθεί και στοχεύει περισσότερες από δεκαπέντε πλατφόρμες και είναι το βασικό λογισμικό πακέτο ανάπτυξης (Software Development Kit – SDK) για το Nintendo Wii U με ένα δωρεάν αντίτυπο παρεχόμενο από την Nintendo μαζί με κάθε developer license του Wii U.

Το Unity είναι αξιοσημείωτο για την ικανότητα του να δημοσιεύει παιχνίδια σε πολλαπλές πλατφόρμες. Μέσα σε ένα έργο, οι προγραμματιστές έχουν την επιλογή μεταφοράς του σε κινητές συσκευές, περιηγητές διαδικτύου, desktops και κονσόλες. Οι υποστηριζόμενες πλατφόρμες περιλαμβάνουν BlackBerry 10, Windows Phone 8, Windows, OS X, Linux (κυρίως Ubuntu) Android, iOS, Unity Web Player (συμπεριλαμβανομένου του Facebook), Adobe Flash, PlayStation 3, PlayStation 4, PlayStation Vita, Xbox 360, Xbox One, Wii U και Wii. Το Unity περιλαμβάνει επίσης ένα server για assets και τη μηχανή φυσικής της NVidia, PhysX.

Το Unity Pro είναι διαθέσιμο έναντι μηνιαίας ή εφάπαξ χρέωσης και το Unity Free διατίθεται δωρεάν για μη-εμπορική χρήση.

Χαρακτηριστικά

Ροή εργασίας (Workflow)

Το Unity προσφέρει ένα ισχυρό πακέτο εργαλείων για την βελτιστοποίηση και διατήρηση ενός γοργού ρυθμού εργασίας. Δίνει τη δυνατότητα εισαγωγής οποιουδήποτε asset (προκατασκευασμένο τμήμα υλοποίησης) χάρη σε ένα ολοκληρωμένο σύστημα αγοράς assets. Διευκολύνει επίσης της δημιουργία σύνθετων κόσμων, με κλιμακούμενη δομικά στοιχεία για κάθε σκηνή, αλλά και το προγραμματισμό του παιχνιδιού με τρείς γλώσσες προγραμματισμού, C#, UnityScript (παραλλαγή της Javascript) και Boo (βασισμένη σε Python) με υλοποίηση πάνω στο .NET πλαίσιο ανοιχτού κώδικα MONO.

Ποιότητα Παραγωγής

Rendering

- Υψηλή εικονική πιστότητα με το DirectX 11
- High Dynamic Range (HDR) Rendering
- 100 ενσωματωμένοι shaders
- Προϋπολογισμός ορατότητας

Φωτισμός

- Επαναληπτικό Lightmapping
- Διπλό Lightmapping για μεγάλες σκηνές
- Καθολικός φωτισμός και εκπέμποντα υλικά
- «Σκληρές» σκιές σε πραγματικό χρόνο
- Οριστικοποίηση (“baking”) φωτισμού
- Αυτοματοποιημένο UV Unwrapping

Ειδικά Εφέ

- Μεταφορά real-time render σε texture
- Ρεαλιστικό νερό
- Σύστημα particles Shuriken
- Ρύθμιση μέσω συναρτησιακών καμπυλών

Ήχος

- Ενσωματωμένος έλεγχος ηχητικών κομματιών
- Υψηλός ακουστικός ρεαλισμός
- Χαμηλοπερατά/Υψιπερατά φίλτρα διεύλευσης
- Εφέ: Distortion, Chorus, Echo, Reverb κ.α.

Υλικά (Materials)

- Εισαγωγή και δημιουργία materials
- Υποστήριξη διαδικαστικών υλικών
- Απλοποιημένη διαδικασία οριστικοποίησης εμφάνισης

Έδαφος

- Αποδοτική σχεδίαση εδαφών
- Πλούσιες λεπτομέρειες
- Εργαλεία δημιουργίας δέντρων με χαμηλό κόστος ισχύς
- Επιλεκτικό rendering βασισμένο στην απόσταση

Φυσική

- NVIDIA® PhysX® Physics
- Δύο είδη εξομοιωτών φυσικής υφάσματος
- Rigidbodies για εφαρμογή δυνάμεων
- Εύκαμπτα αντικείμενα
- Ragdolls
- Σύνδεσμοι, ελατήρια και κλειδώσεις
- Εξομοίωση τριβής τροχών αυτοκινήτου

Τεχνητή Νοημοσύνη

- Ενσωματωμένος αλγόριθμος εύρεσης μονοπατιού

Mecanim

Σύντομη εισαγωγή

Το Mecanim είναι ένα ισχυρό και ευέλικτο σύστημα animation, αποκλειστικό στη Unity, που βοηθά στη διαχείριση των κινήσεων ανθρωποειδών και μη χαρακτήρων με φυσική, ρευστή μετάβαση από τη μια κίνηση στην άλλη.

Όντας ενσωματωμένο μέσα στη Unity καθιστά τη χρήση τρίτων προγραμμάτων για τη βελτιστοποίηση του animation περιττή. Παρέχει όλα τα απαραίτητα εργαλεία για τη δημιουργία μηχανών καταστάσεων, ελεγκτές και blend trees κατευθείαν μέσα στη Unity.

Μπορεί να χρησιμοποιηθεί για το animation οποιουδήποτε αντικειμένου στη σκηνή, από πολύπλοκους χαρακτήρες μέχρι sprites και φώτα. Επίσης, χάρη στα AnimationEvents είναι δυνατό να τρέξει κάποιο script μέσα από την αναπαραγωγή ενός animation.

Η σταθερότητα που παρέχει η Unity σε συνδυασμό με τις μεθόδους βελτιστοποίησης του Mecanim εξασφαλίζουν μια ομαλή και ελαφριά σε υπολογιστική ισχύ απόδοση κατά την εκτέλεση.

Μεταφορά Animation

Το Mecanim κάνει χρήση ενός Avatar ως μεσάζοντα για τη διαχείριση κινήσεων από διαφορετικά rigs. Αποσκοπεί έτσι στην διευκόλυνση της αντιστοίχισης οστών μεταξύ διαφορετικών skeletal rigs και καθιστά δυνατή τη χρήση έτοιμων animation από το asset store ή από άλλα προγράμματα.

Το Avatar του Mecanim μπορεί να στηθεί εύκολα με αυτοματοποιημένη χαρτογράφηση οστών και αλγόριθμους υπολογισμού της στάσης του σώματος. Διαθέτει επίσης επιλογές για τον έλεγχο της μυϊκής παραμόρφωσης και αξιοποιεί ένα μοντέλο μάζας για τον εντοπισμό του κέντρου μάζας του χαρακτήρα και τον μέσο προσανατολισμού του σώματος για τον υπολογισμό της τροχιάς των φυσικών κινήσεων. Όλα τα παραπάνω είναι εύκολα μεταφέρσιμα σε διαφορετικούς χαρακτήρες, απλά χαρτογραφώντας εκ νέου το καινούριο σκελετικό σύστημα στο Avatar του Mecanim.

Δέντρα συγχώνευσης κινήσεων (Blend Trees) και Μηχανές Πεπερασμένων Καταστάσεων (State Machines)

Χάρη στο Mecanim μπορούμε εύκολα να δημιουργήσουμε και να τροποποιήσουμε πολύπλοκα Blend Trees και State Machines ώστε να έχουμε τον απόλυτο έλεγχο στο πώς κινούνται οι χαρακτήρες μας. Ο Editor της Unity μας παρέχει εργαλεία για τον διαχωρισμό, τη δημιουργία βρόγχων και την παραμετροποίηση των κινήσεων που έχουμε δημιουργήσει σε εξωτερικά προγράμματα. Τα Animation αυτά μπορούν έπειτα να χρησιμοποιηθούν ως «φύλλα» σε ένα Δέντρο Συγχώνευσης Κινήσεων, ή ως καταστάσεις σε μια Ιεραρχική Μηχανή Καταστάσεων.

Τα Blend Trees μας δίνουν τη δυνατότητα να δημιουργήσουμε ένα μεγάλος εύρος κινήσεων χρησιμοποιώντας απλά μερικά κομμάτια animation. Στον Editor του Blend Tree ορίζουμε τις παραμέτρους συγχώνευσης και προβάλλουμε μια προεπισκόπηση της τελικής κίνησης σε 3D ενώ με 2D διάγραμμα μπορούμε να συγχωνεύσουμε animations ανάλογα με δύο παραμέτρους σε ένα κοινό κόμβο.

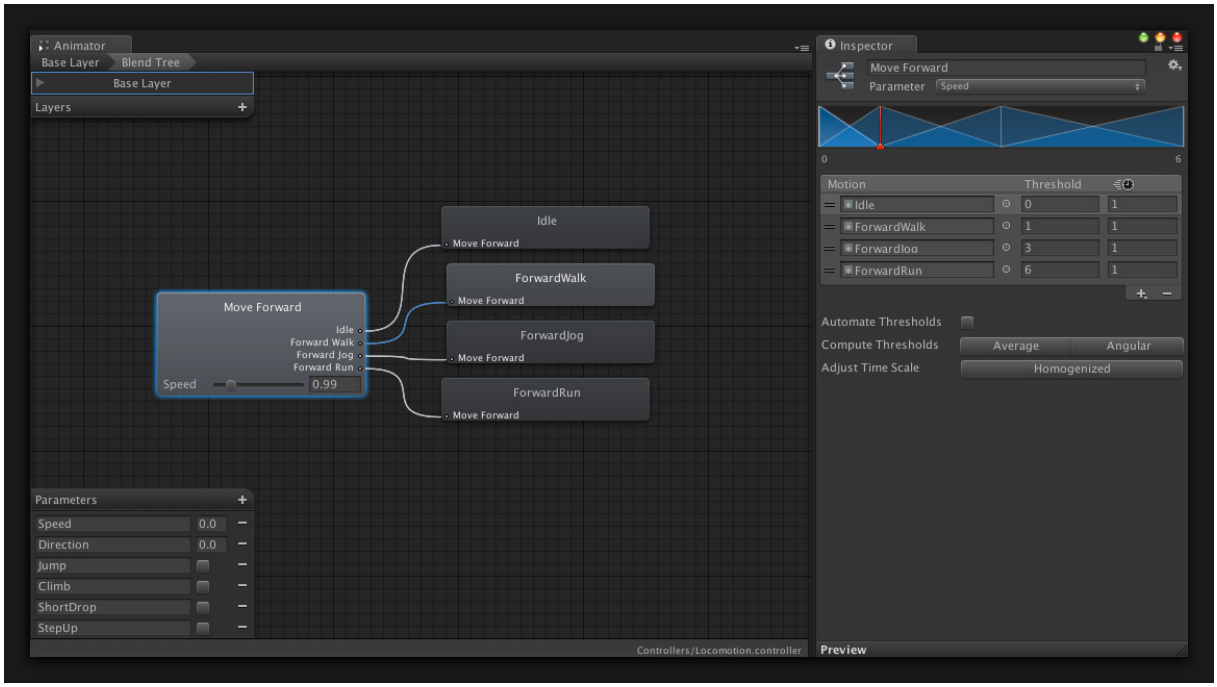

Figure III-1: Παράδειγμα Blend Trees

Το εργαλείο Animator μας δίνει πρόσβαση στις πολυεπίπεδες, ιεραρχικές Μηχανές Πεπερασμένων Καταστάσεων του Mecanim. Μπορούμε να διαχειριστούμε ένα αυθαίρετο αριθμό επιπέδων, με κάθε επίπεδο να χρησιμοποιεί τις δικές του Μηχανές ή να χρησιμοποιεί κοινές μηχανές με ένα επίπεδο «Master». Τα επίπεδα μπορούν να παρακαμφθούν ή να συνδυαστούν και με τη χρήση μιας Μάσκας Σώματος (Body Mask) να χειρίζονται διαφορετικά τμήματα του σώματος. Επιπλέον, με τις Μηχανές Πεπερασμένων Καταστάσεων μπορούμε να σπάσουμε έναν πολύπλοκο διαχειριστή-ελεγκτή σε μικρότερα επαναχρησιμοποιούμενα κομμάτια.

Η προεπισκόπηση μετάβασης σε συνδυασμό με τις Μηχανές Καταστάσεων μας επιτρέπει να αλλάξουμε το συγχρονισμό πολλαπλών κινήσεων και της μετάβασης από τη μια κίνηση στην άλλη, δημιουργώντας έτσι ένα ομαλό και φυσικό αποτέλεσμα. Τέλος, μπορούμε σε πραγματικό χρόνο, καθώς τρέχει το παιχνίδι να ελέγξουμε τη ροή εργασίας των Μηχανών με το την αναπαράσταση Live Link.

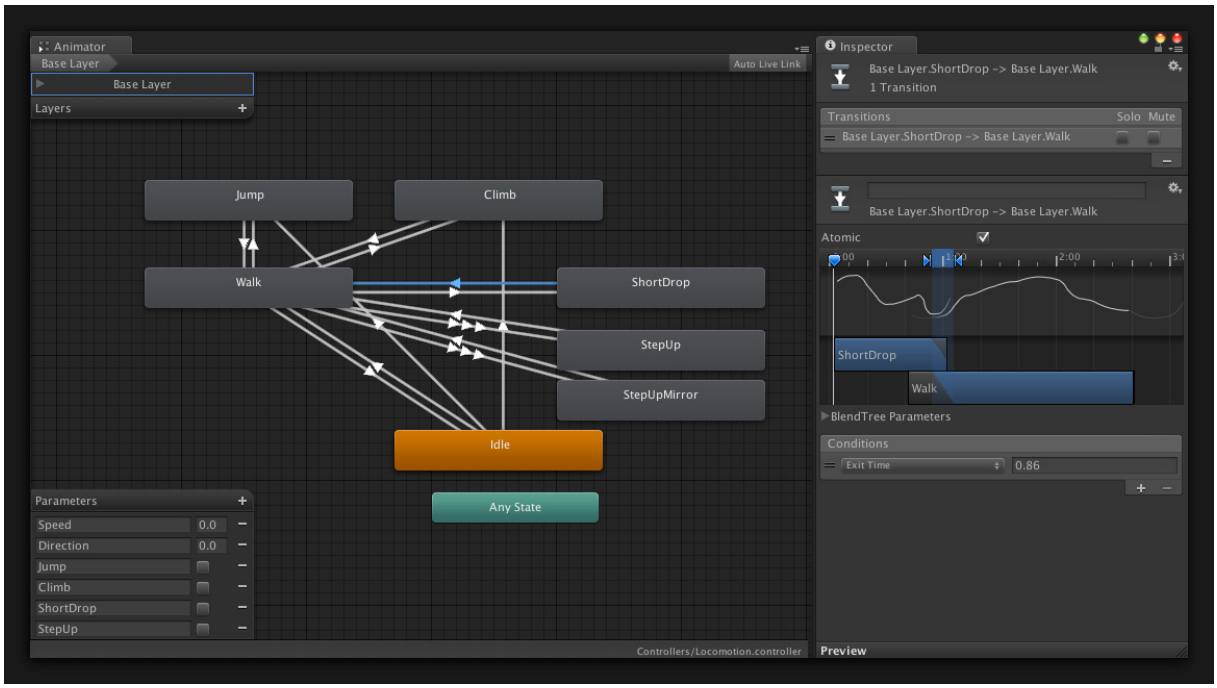

Figure III-2: Παράδειγμα State Machines

IK Rigs

Σε τελικό στάδιο μπορούμε να χρησιμοποιήσουμε τα ενσωματωμένα, αυτοματοποιημένα IK Rigs του Mecanim για να τοποθετήσουμε τα πόδια στο έδαφος, τα χέρια στα άκρα μιας επιφάνειας ή να κάνουμε το χαρακτήρα μας να στρέφει το βλέμμα του σε κάποιο αντικείμενο, με τις κατάλληλες προσαρμογές στις κινήσεις του σώματος, του κεφαλιού και του προσώπου. Παρέχονται επίσης και άλλες λειτουργίες όπως σταθεροποίησης ποδιών, ευθυγράμμισης τροχιάς κέντρου μάζας και δευτερεύουσες κινήσεις, με στόχο την υλοποίηση πραγματικά ρεαλιστικών κινήσεων.

Επίδοση

Η Unity παρέχει εργαλεία που βοηθούν στην εγγύηση αξιόπιστης απόδοσης, ομαλό framerate και βέλτιστη εμπειρία για τους παίκτες.

Εντοπισμός και εξάλειψη συμφορήσεων

Το εργαλείο Profiler παρέχει εύχρηστα στοιχεία στον προγραμματιστή και διευκολύνει την αξιολόγηση των ενεργών διεργασιών και της χρήσης της μνήμης του συστήματος ανά πάσα χρονική στιγμή με μεγάλη ακρίβεια. Εμφανίζει τον φόρτο που επιφέρει στην απόδοση κάθε στοιχείο της σκηνής, από τα πολύπλοκα συστήματα όπως το ShaderLab μέχρι μεμονωμένα μοντέλα και textures. Το Profiler είναι προγραμματίσιμο για ακόμα μεγαλύτερο έλεγχο των assets μας και μπορεί να χρησιμοποιηθεί πάνω σε τοπικό δίκτυο για την επίδοση σε εξωτερικές συσκευές.

Occlusion Culling

Η Unity διαθέτει μια αποκλειστική, προϋπολογισμένη μέθοδο απόρριψης μη-ορατών αντικειμένων όσον αφορά τις κάμερες της σκηνής. Η «Precomputed Occlusion Culling Solution» όπως ονομάζεται, δημιουργήθηκε μαζί με το λογισμικό Umbra και εξασφαλίζει ότι μόνο ότι φαίνεται από την κάμερα θα γίνει render. Το Occlusion Culling της Unity δουλεύει τόσο σε desktop, όσο και σε εφαρμογές web, κινητών συσκευών και κονσόλων μειώνοντας αισθητά τις καθυστερήσεις κατά τη λειτουργία.

Υποστήριξη επιπέδων λεπτομέρειας

Όσο μεγαλώνουν οι σκηνές με τις οποίες εργαζόμαστε, τόσο γίνεται πιο σημαντικό το θέμα της βελτιστοποίησης της επίδοσης. Ένας από τους τρόπους που έχουμε για να το πετύχουμε αυτό είναι να διαθέτουμε μοντέλα με διαφορετικά επίπεδα ανάλυσης-λεπτομέρειας (level-of-detail, LOD) ανάλογα με την απόσταση τους από τη κάμερα. Η Unity μας παρέχει αυτή την ευκολία με τα LODGroups.

Βελτιστοποίηση γραφικών κατά την έκδοση-«χτίσιμο»

Για την βελτίωση της απόδοσης, η Unity αυτομάτως συνδυάζει τα μικρά τμήματα γεωμετρίας σε δεσμίδες μεγαλύτερων κατά την εξαγωγή του έργου μας σε εκτελέσιμο αρχείο.

Μείωση μεγέθους εκτελέσιμου αρχείου

Μπορούμε εύκολα να παραλείψουμε οποιοδήποτε μέρος της μηχανής Unity δε χρησιμοποιήθηκε και δεν είναι αναγκαίο στο τελικό αποτέλεσμα, μειώνοντας έτσι το μέγεθος για κινητές συσκευές, για εφαρμογές διαδικτύου κ.α.

Επισκόπηση του περιβάλλοντος της Unity

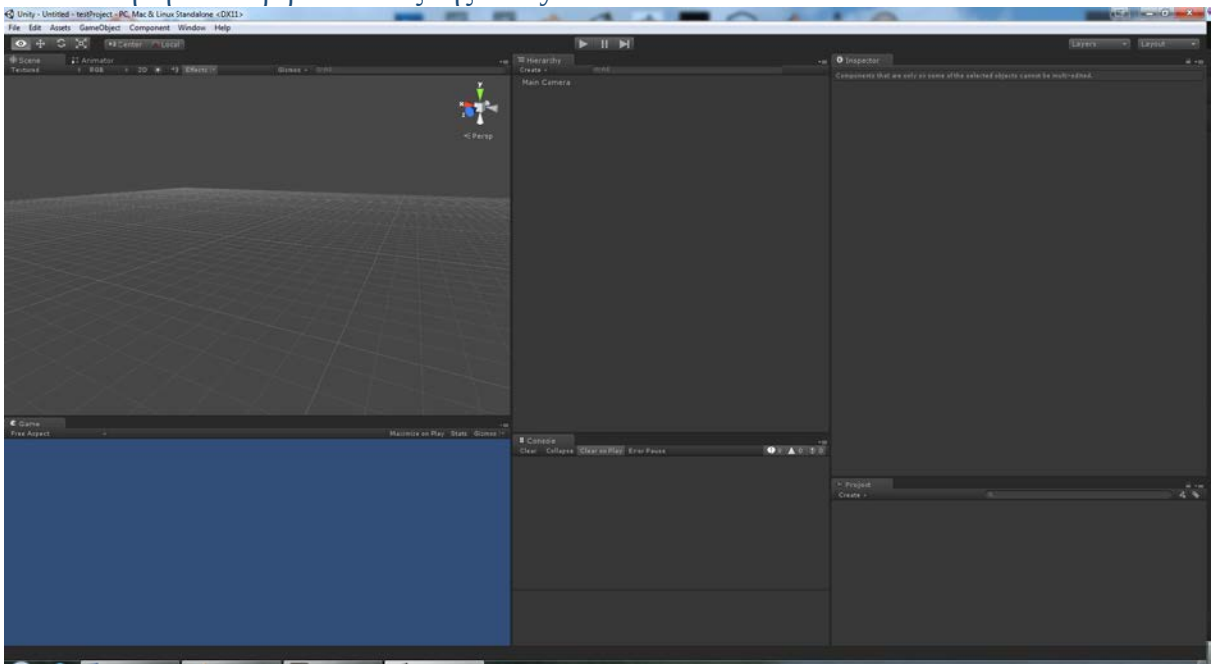

Figure III-3: Το γραφικό περιβάλλον της Unity Game Engine

Το layout της Unity που έχουμε επιλέξει μας δίνει 6 τμήματα με χρήσιμα στοιχεία το κάθε ένα, αν και προφανώς είναι εύκολα προσαρμόσιμο στις ανάγκες του κάθε χρήστη. Στα αριστερά έχουμε τη σκηνή μας, όπου τοποθετούμε όλα τα αντικείμενα που χρειαζόμαστε, και μπορούμε εύκολα να μετακινηθούμε και να κάνουμε αλλαγές. Σε διαφορετική καρτέλα πίσω από τη σκηνή βρίσκεται ο Animator που μας επιτρέπει να ελέγξουμε τις κινήσεις των αντικειμένων μας. Κάτω από αυτά βρίσκεται το Game View, μια προεπισκόπηση δηλαδή του τι φαίνεται όταν τρέχει η εφαρμογή. Στη μέση και πάνω βρίσκεται το παράθυρο ιεραρχίας, όπου καταγράφεται ότι βρίσκεται πάνω στη σκηνή με ιεραρχικό τρόπο (γονείς, παιδιά, groups κ.α.) ενώ από κάτω έχουμε την κονσόλα στην οποία μπορούμε να δούμε χρήσιμα μηνύματα που εμφανίζονται κατά την εκτέλεση, όπως errors ή εξόδους του προγράμματος και του Debug Log. Στα δεξιά βρίσκεται ο Inspector και εξερευνητής του Project. Ο Inspector μας εμφανίζει τις ρυθμίσεις του αντικειμένου που έχουμε επιλέξει και το Project Explorer όλα τα assets που έχουμε εισάγει στο Project, είτε τα έχουμε χρησιμοποιήσει στη σκηνή είτε όχι.

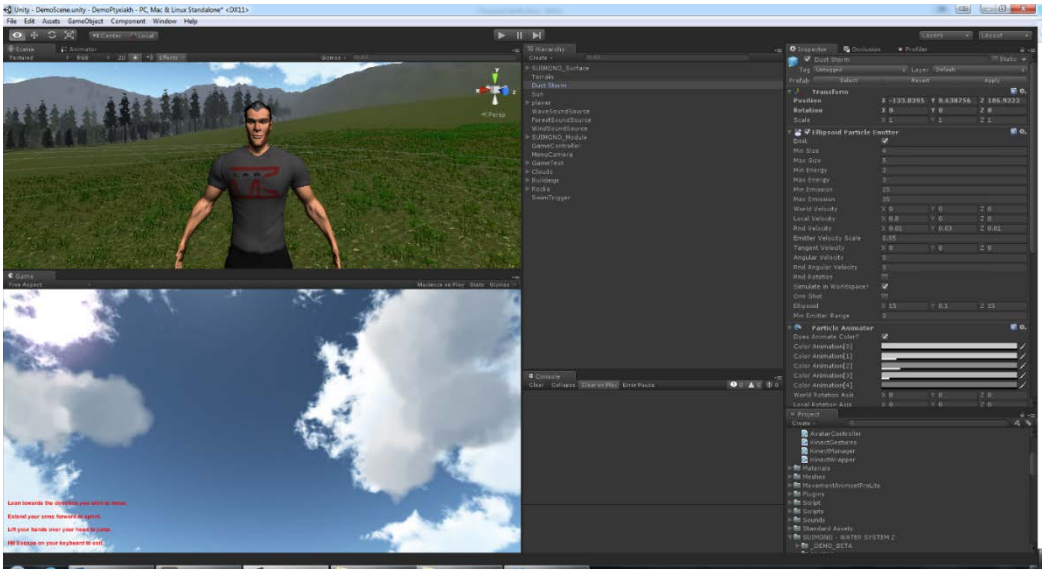

Figure III-4: Παράδειγμα των παραθύρων εν χρήση κατά την υλοποίηση της πτυχιακής

IV. Kinect

Figure IV-1: Η συσκευή Kinect

Γενικά

Το Kinect (γνωστό ως Project Natal κατά το στάδιο του development) είναι μία σειρά συσκευών ανίχνευσης κίνησης της Microsoft αρχικά για τη κονσόλα Xbox 360 και το λειτουργικό Windows, και αργότερα για το Xbox One. Βασισμένο σε μια κάμερα βάθους, αλλά σε στυλ webcam, προσφέρει στους χρήστες τη δυνατότητα αλληλεπίδρασης με την κονσόλα/υπολογιστή δίχως την ανάγκη για κάποιο άλλο χειριστήριο, μέσω μιας διεπαφής χρήστη που αξιοποιεί φυσικές χειρονομίες και προφορικές εντολές. Η πρώτη γενιά Kinect κυκλοφόρησε το Νοέμβριο του 2010, σε μια προσπάθεια

να διευρύνει το κοινό του Xbox πέρα από το τυπικό target-group του, τους gamers (παίκτες video games). Για Windows κυκλοφόρησε την 1η Φεβρουαρίου 2012.

Το Kinect βρίσκει ανταγωνισμό σε άλλες συσκευές ανίχνευσης κίνησης για φορητές κονσόλες, όπως το Wii Remote Plus για το Wii και το Wii U, το PlayStation Move/PlayStation Eye για το PlayStation 3 και το PlayStation Camera για το PlayStation 4.

Η Microsoft κυκλοφόρησε το κιτ ανάπτυξης λογισμικού (SDK) του Kinect στις 16 Ιουνίου 2011 για τα Windows 7. Αυτό το SDK είχε ως στόχο να επιτρέψει στους προγραμματιστές να γράφουν εφαρμογές για το Kinect σε C++/ CLI, C# και Visual Basic .NET.

Τεχνολογία

Το Kinect είναι βασισμένο σε τεχνολογία λογισμικού παραγόμενη από την Rare, θυγατρική εταιρία της Microsoft Game Studios που ανήκει στη Microsoft, και σε τεχνολογία κάμερας βάθους από την Ισραηλινή εταιρία PrimeSense, που δημιούργησε ένα σύστημα ικανό να αναγνωρίσει συγκεκριμένες χειρονομίες, καθιστώντας εφικτή τη χρήση ηλεκτρονικών συσκευών χωρίς κάποιο χειριστήριο, χρησιμοποιώντας έναν προβολέα και κάμερα υπέρυθρης ακτινοβολίας και ένα ειδικό μικροτσίπ για την ανίχνευση κινήσεων αντικειμένων και ατόμων σε 3 διαστάσεις. Αυτός ο 3d Scanner με την ονομασία Light Coding, κάνει χρήση ενός τύπου 3d ανασχηματισμού από 2d εικόνες.

Ο αισθητήρας του Kinect αποτελείται από μια οριζόντια μπάρα συνδεδεμένη σε μια μικρή βάση με ένα μηχανικό άξονα περιστροφής και είναι σχεδιασμένος για να τοποθετείται κατά μήκος πάνω ή κάτω από την οθόνη. Η συσκευή περιέχει μια κάμερα χρώματος RGB, έναν ανιχνευτή βάθους και ένα σύστημα πολλαπλών μικροφώνων που τρέχει ιδιόκτητο λογισμικό, που προσφέρουν καταγραφή 3d κίνησης ολόκληρου του σώματος, αναγνώριση προσώπων και φωνής. Τα μικρόφωνα του αισθητήρα του Kinect καθιστούν ικανό τον προσδιορισμό της ηχητικής πηγής και την καταστολή περιβαλλοντικού θορύβου επιτρέποντας λειτουργίες όπως chat χωρίς headset.

Ο αισθητήρας βάθους απαρτίζεται από έναν προβολέα μιας υπέρυθρης ακτίνας σε συνδυασμό με ένα μονόχρωμο συμπληρωματικό ημιαγωγό μετάλλου-οξειδίου (Complementary Metal-Oxide Semiconductor – CMOS), που καταγράφει video σε 3d υπό οποιεσδήποτε συνθήκες φωτισμού. Το εύρος του αισθητήρα είναι προσαρμόσιμο και το λογισμικό του Kinect είναι ικανό να βαθμονομεί αυτομάτως τον αισθητήρα ανάλογα με το φυσικό περιβάλλον του παίκτη.

Σύμφωνα με πληροφορίες παρεχόμενες σε μεταπωλητές, το Kinect υποστηρίζει αναγνώριση ως και έξι ατόμων, συμπεριλαμβανομένων και δύο ενεργών παικτών στους οποίους μπορεί να γίνεται ανάλυση κίνησης με καταγραφή ως είκοσι αρθρώσεων για τον κάθε ένα. Ωστόσο, η PrimeSense δηλώνει ότι ο αριθμός των ατόμων που μπορεί να «δει» η συσκευή περιορίζεται μόνο από το πόσοι μπορούν να χωρέσουν στο οπτικό πεδίο της κάμερας.

Κυκλοφορία

Η Microsoft είχε προϋπολογισμό διαφήμισης πεντακοσίων εκατομμυρίων δολαρίων (US $) για την έναρξη της κυκλοφορίας του Kinect, μεγαλύτερο και από το σύνολο των επενδύσεων για τη κονσόλα Xbox 360. Η διαφημιστική καμπάνια "You Are the Controller" («Εσύ είσαι το Χειριστήριο») με σκοπό να απευθυνθεί σε καινούρια ακροατήρια, συμπεριέλαβε διαφημίσεις σε προϊόντα όπως δημητριακά και αναψυκτικά, διαφημίσεις στη τηλεόραση και σε περιοδικά, ακόμα και πάρτυ και συναυλίες με διάσημους καλεσμένους.

Kinect για Windows

Στις 21 Φεβρουαρίου 2011, η Microsoft ανακοίνωσε την κυκλοφορία μιας μη-εμπορικής έκδοσης του Kinect SDK για Windows την άνοιξη του 2011, που κυκλοφόρησε τελικά στις 16 Ιουνίου για Windows 7, σε 12 χώρες. Το SDK περιλαμβάνει οδηγούς (drivers) συμβατούς με Windows 7 για τη συσκευή Kinect. Προσφέρει δυνατότητες τους Kinect στους προγραμματιστές ώστε να δημιουργήσουν εφαρμογές με C++, C# ή Visual Basic, χρησιμοποιώντας το Microsoft Visual Studio 2010, και περιλαμβάνει τις παρακάτω λειτουργίες:

- Raw Sensor streams (Πρωτόγονα δεδομένα του αισθητήρα): Πρόσβαση σε δεδομένα χαμηλού επιπέδου από τον αισθητήρα βάθους, την κάμερα και τα μικρόφωνα.

- Skeletal tracking (Ανίχνευση σκελετικού συστήματος): Η δυνατότητα ανίχνευσης και παρακολούθησης του skeletal rig ενός ή δύο ατόμων που βρίσκονται στο οπτικό πεδίο του Kinect, για εφαρμογές ανίχνευσης χειρονομιών.
- Ανώτερες ηχητικές ικανότητες: Οι δυνατότητες διαχείρισης ήχου περιλαμβάνουν εξελιγμένη καταστολή θορύβου και ακύρωση ηχούς, εντοπισμός πηγής ήχου και ενσωμάτωση με το API αναγνώρισης ομιλίας των Windows.
- Παραδείγματα κώδικα και Documentation

Το Μάιο του 2012, ο Craig Eisier, γενικός διευθυντής του Kinect για Windows ανέφερε ότι σχεδόν 350 εταιρίες συνεργάζονταν με τη Microsoft για τη δημιουργία εφαρμογών Kinect για Windows.

Έκδοση 1.5

Το Μάρτιο του 2012, η Microsoft ανακοίνωσε ότι η νέα έκδοση του Kinect SDK για Windows θα ήταν διαθέσιμη το Μάιο του 2012. Η έκδοση 1.5 διατέθηκε στο κοινό στις 21 Μαΐου 2012 και πρόσθεσε καινούριες δυνατότητες, υποστήριξη σε πολλές νέες γλώσσες και κυκλοφόρησε για πρώτη φορά σε 19 επιπλέον χώρες.

- Το Kinect 1.5 SDK συμπεριλαμβάνει το Kinect Studio, μια νέα εφαρμογή που επιτρέπει στους χρήστες να καταγράφουν και να αναπαράγουν κινήσεις για την αλληλεπίδραση με άλλες εφαρμογές.
- Υποστήριξη για λειτουργία καθίσματος (seated mode) που επιτρέπει σε εφαρμογές να αναγνωρίζουν το κεφάλι, το λαιμό και τα χέρια του χρήστη είτε αυτός βρίσκεται όρθιος μπροστά στη κάμερα είτε κάθεται.
- Τέσσερις επιπλέον γλώσσες για τη φωνητική αναγνώριση – Γαλλικά, Ισπανικά, Ιταλικά και Ιαπωνικά
- Διαθέσιμο σε Χόνγκ Κόνγκ, Νότια Κορέα και Ταϊβάν το Μάιο, Αυστρία, Βέλγιο, Βραζιλία, Δανία, Φινλανδία, Ινδία, Ολλανδία, Νορβηγία, Πορτογαλία, Ρωσία, Σαουδική Αραβία, Σιγκαπούρη, Νότια Αφρική, Σουηδία, Ελβετία και Ηνωμένα Αραβικά Εμιράτα τον Ιούνιο

Εκδόσεις 1.6, 1.7, 1.8

Το SDK του Kinect για Windows για τον αισθητήρα πρώτης γενιάς του Kinect, ενημερώθηκε μερικές ακόμα φορές, με την έκδοση 1.6 να κυκλοφορεί στις 8 Οκτωβρίου 2012, την έκδοση 1.7 στις 18 Μαρτίου 2013 και τέλος την έκδοση 1.8 στις 17 Σεπτεμβρίου 2013.

Έκδοση 2

Η δεύτερη γενιά του Kinect για Windows, βασισμένο στο Kinect του Xbox One, συμπεριλαμβανομένου ενός νέου αισθητήρα, ήταν προγραμματισμένο να κυκλοφορήσει το καλοκαίρι του 2014. Μια αρχική έκδοση της νέας συσκευής άρχισε να δίνεται σε προγραμματιστές από τις 22 Νοεμβρίου 2013, ενώ στις 15 Ιουλίου 2014 κυκλοφόρησε μια αρχική έκδοση του καινούριου SDK.

Βραβεία

- 2011 MacRobert Award for Engineering Innovation
- 2011 T3's "Gadget of the Year" award, "Gaming Gadget of the Year"
- 2nd Rank "The 10 Most Innovative Tech Products for 2011" στην απονομή Popular Mechanics Breakthrough Awards στη Νέα Υόρκη.

Βιβλιογραφία

http://www.autodesk.com/

http://www.autodesk.com/products/3ds-max/features/all/gallery-view

http://area.autodesk.com/maxturns20/history

http://docs.pixologic.com/user-guide/

Yetter, Matthew 2004 "Basic Concepts: The Concept of the Pixol" ZBrush 2 Practical Guide. Σελ: 11–12.

http://docs.pixologic.com/features/main-features/

http://unity3d.com

http://unity3d.com/legal/eula

http://docs.unity3d.com/Manual/Shaders.html

http://docs.unity3d.com/Manual/ScriptingSection.html

http://docs.unity3d.com/Manual/Editor.html

https://developer.valvesoftware.com/wiki/Skybox

http://www.opengl.org/wiki/Geometry_Shader

http://unity3d.com

http://unity3d.com/legal/eula

http://docs.unity3d.com/Manual

http://blogs.msdn.com/b/kinectforwindows/

http://msdn.microsoft.com/en-us/library/microsoft.kinect.depthimageformat.aspx

http://www.microsoft.com/en-us/kinectforwindows/

Προγράμματα που χρησιμοποιήθηκαν

Autodesk 3ds Max 2013
Adobe Photoshop CS5
Pixologic ZBrush 4R6
Unity Game Engine 4.5.0f6