Τεχνολογία Πολυμέσων
Εργασία 1 – Skeletal Animation

- ## Οδηγίες Χρήσης:

  Για να buildαρετε το project το εκανα αρκετά απλό.

  1. Πηγαίνετε στο root folder του project «/animation-demo”
  2. Ανοίγετε το command line prompt σε αυτό το φάκελο
  3. Πατατε build (εκτελεί το build.bat) για παραγωγή .exe
  4. Πατατε run (run.bat) για να το τρέξετε!

- ## Περιγραφή Τεχνολογιών:

  Έχω προσπαθήσει να κάνω το API όσο πιο απλό γίνεται.

  Λοιπόν, καταρχήν όλο το demo βρίσκεται στο /src/game.c, απο εκεί υπάρχουν τρείς συναρτήσεις, η init κανει το model loading, συγκεκριμένα η read_collada_maya διαβάζει το μοντέλο (δέρμα + σκελετός) και η read_collada_animation διαβάζει το animation που θα πεχτεί (βρίσκονται στο src/collada_parser.h).

  Μετά έχουμε την update η οποία κάνει το interpolation μεταξύ των Animation Keyframes, ο κώδικας για αυτό βρίσκεται στο src/animation.h (duh) και τέλος η render η οποία απλά δείχνει το μοντέλο μας της οποίας η συνάρτηση render_animated_model(..) βρίσκεται επίσης στο src/animation.h.

  Γενικά η τεχνολογία που χρησιμοποιούμε λέγεται skeletal animation και αυτό που βασικά κάνει είναι να κινεί meshes αποτελεσματικά, στην ουσία είναι μια μέθοδος συμπίεσης αφου αναγει το mesh σε μεγαλύτερα κομματια (Joints) με αποτέλεσμα να κάνει το animation φθηνότερο (υπολογιστικα) απο το να έχουμε να κινούμε 100.000 ξεχωριστά vertices 60 φορες καθε δευτερόλεπτο! Πρακτικά υπολογίζει το πού πρέπει να βρίσκεται κάθε vertex του mesh μας υπολογίζοντας την σχετική του θέση σε σχέση με ένα η περισσότερα joints. Η εξίσωση για το “skinning” όπως λέγεται αυτή η πράξη είναι:

$$outv = \sum_{i=0}^{n} \{((v * BSM) * IBMi * JMi) * JW\}$$

where:

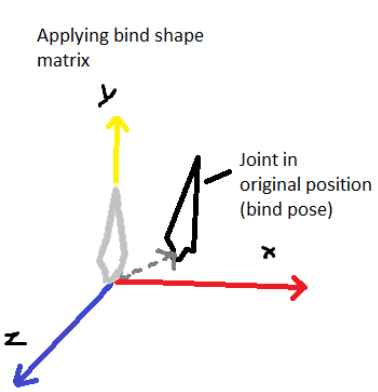
- n: The number of joints that influence vertex v
- BSM: Bind-shape matrix
- IBMi: Inverse bind-pose matrix of joint i
- JMi: Transformation matrix of joint i
- JW: Weight of the influence of joint i on vertex v

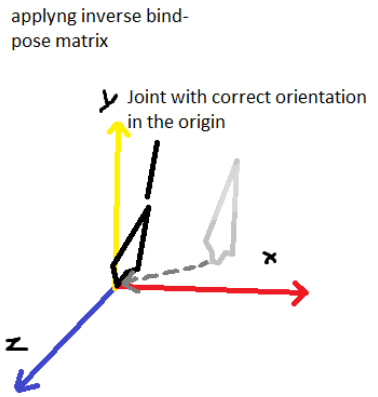
Note: v, BSM, IBMi, and JW are constants with regards to some skeletal animation. Depending on your application, it may be beneficial to premultiply BSM with IBMi or v with BSM.

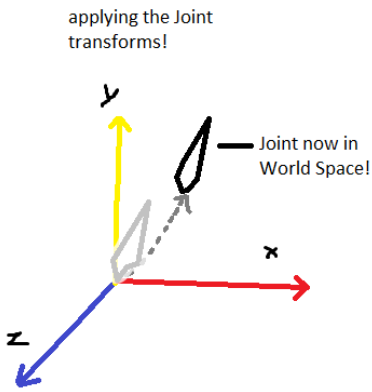
(source Collada Spec)

Ουσιαστικά για να πάμε απο ενα κανονικό σημείο στο σημείο που θα έπρεπε να βρίσκεται το vertex μας, πρώτα το πολλαπλασιάζουμε με το Bind Shape Matrix, το οποιο είναι το orientation του mesh στο χώρο, μετα με το Inverse Bind Matrix του joint για το οποίο υπολογίζουμε το οποίο πρακτικά μας πάει απο την Bind θεση μας (T-pose) στην αρχή των αξόνων για κάθε joint, μετα πολλαπλασιάζουμε me το Joint Matrix το οποίο ειναι η WORLD θέση που πρέπει να είναι το κάθε Joint σε καποια στιγμή, τέλος πολλαπλασιάζουμε με το weight του κάθε Joint (το πόσο το καθε Joint επιρεάζει καθε vertex) και κάνουμε αυτήν την πράξη για κάθε joint το οποίο επιρεάζει το κάθε Joint φερνοντας το τελικά στην σωστή του θέση!

Έχω κάνει απο κάτω ενα μικρό σχεδιο για να σας το δείξω:







- Το λογισμικό που χρησιμοποιήθηκε:

1. Χρησιμοποίησα το Win32 API για να φτιάξω ενα Windows παράθυρο και να παρω Function Pointers προς συναρτήσεις OpenGL. (Για κάποιο λογο αυτό δεν ειναι αυτόματο στα windows)
2. Χρησιμοποίησα επίσης το OpenGL API (3.3+) για να δείξω πράγματα στην οθόνη.

3. Τελος χρησιμοποίησα το stb_image.h, μια Single Header Library του Sean Barrett για να φορτώνω εικόνες. Δεν νομιζω να χρειάζεται επεξήση για αυτην την επιλογη!

- # Πηγές Πληροφόρησης:

1. Game Engine Architecture 2nded. Ch. 11(Animation Systems), για το Animation System.
2. http://wazim.com/collada-tutorial1, για τον Collada Parser.
3. Τα streams του Johnathan Blow περι Skeletal Animation.

- # Σύντομη περιγραφή προβλημάτων:

Τα κύρια σημεία οπου δυσκολεύτηκα ήταν κυρίως τα Μαθηματικά!

Το κατα πολύ μεγαλύτερο πρόβλημα σε όλη την υλοποίηση ήταν το Interpolation των γωνιών των Joints του Animation System. Το θέμα είναι οτι ο μόνος τρόπος να κάνεις (σωστό) interpolation σε γωνίες είναι με χρήση Quaternions, και επειδή ήθελα το Engine να είναι αυτοδύναμο έπρεπε να κάτσω και να βάλω Quaternions στην βιβλιοθήκη μαθηματικών μου.

Ενα άλλο πρόβλημα που είχα ήταν το Order of Transformations. Γενικά λόγω του οτι το Direct3D και το OpenGL εχουν διαφορετικά συστήματα συντεταγμένων και πολύς κόσμος απο τις πηγές μου χρησιμοποιύσε Direct3D το να κάνω στο μυαλό μου τις μετατροπές δεν ήταν καθόλου ευκολο.