Lezione 6 20/03/2024

Integrazione dei modelli 3D

Dietro ai vari engine (unity, unreal...) tutti usano lo **scene graph**. Serve una struttura gerarchica per poter organizzare gli elementi del virtual world sia nello spazio, ma anche dal punto di vista logico. Questo serve anche per gestire le collisioni. Molto spesso quando creo il modello 3D si assegnano anche i valori per la collision detection.

Nel virtual world bisogneranno anche essere scelte le luci, i suoni, la virtual camera, e vanno messe nello scene graph, di modo che questi elementi verranno gestiti a runtime.

Lo **scene graph** è una struttura gerarchica dove vengono posizionati elementi di tipi differenti (geometrici, luci, suoni, ...).

La **scena** deve essere renderizzata (grafica, audio, aptica) dal punto di vista dell'utente, ovvero la virtual camera che rappresenta l'utente. Di base questa è una **scena statica**.

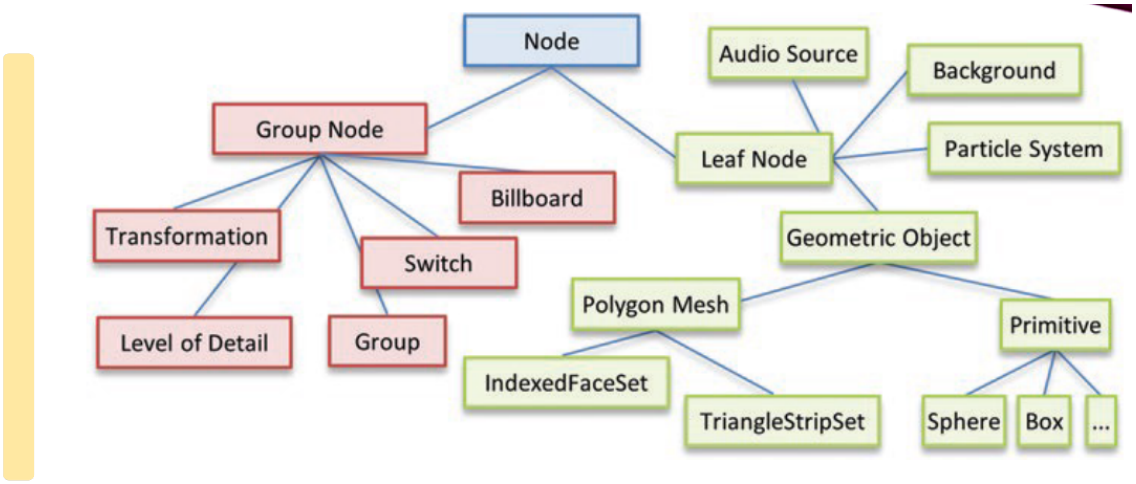
Se invece la scena presenta elementi che si possono muovere indipendentemente (a runtime), allora questa è una **scena animata**.

Se ci sono elementi che possono interagire con l'utente allora abbiamo una **scena interattiva**, dove gli oggetti reagiscono agli eventi come l'input dell'utente o all'interazione con altri oggetti, cambiando il proprio stato.

Vengono usati i **DAG (direct acyclic graph)**, ovvero una struttura dove ci sono dei nodi collegati da archi, ad un nodo possono arrivare più frecce entranti, è aciclico perché si cerca un grafo dove si può trovare un path che non passa per elementi precedenti del percorso.

Alternativamente vengono usati i **tree**, dove però un nodo non può avere più archi entranti. Però è una struttura più semplice da navigare.

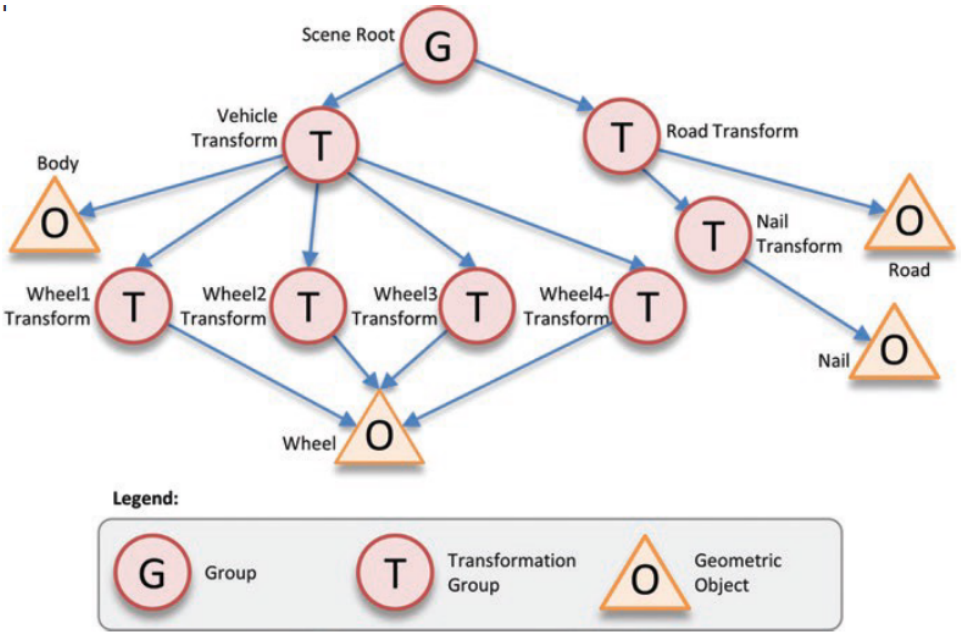
Con il DAG si può risparmiare a livello di presentazione, per esempio raggruppando degli elementi in un nodo singolo (es 4 gambe di tavolo che si muove).



I nodi foglia (verde), sono i nodi da cui non escono altre frecce, vengono di solito mostrati visualmente o tramite audio.

Ci sono anche i nodi che usano lo scene graph per strutturare la scena, e quindi questi nodi sono dei gruppi (rosso), oppure un **nodo trasformazione** che è quello che dice come scalare l'oggetto (coordinate locali per i nodi figlio) rispetto alle coordinate globali. Ci sono i nodi switch per le animazioni, ... li vedremo. Quindi questi nodi possono aggiungere il comportamento.

Viene più semplice usare dei **gruppi di trasformazione**, per esempio se sposto un oggetto composto da molti nodi, posso selezionare il gruppo per muovere l'intero oggetto, così che il grafo venga percorso a cascata e tutti i nodi siano utilizzati.



Il nodo transform può dirmi per esempio dove sta la ruota (anteriore, posteriore, destra, sinistra, con informazioni come rotazione etc). Questi nodi sono delle matrici che contengono informazioni. Quando viene fatto il rendering finale della scena, si parte dalla root, e per ogni nodo parent sapremo dove è posizionato, e scendendo ogni nodo di transform ha una matrice che va moltiplicata per quella di prima, quando si arriva alla foglia si trova la posizione locale dell'oggetto, e quindi l'engine saprà dove effettuare il rendering di ciascun oggetto.

Oggetti 3D

I modelli 3D definiscono la geometria in un modo che possa essere processata precisamente ed efficientemente dal pc, e si dividono in due categorie:

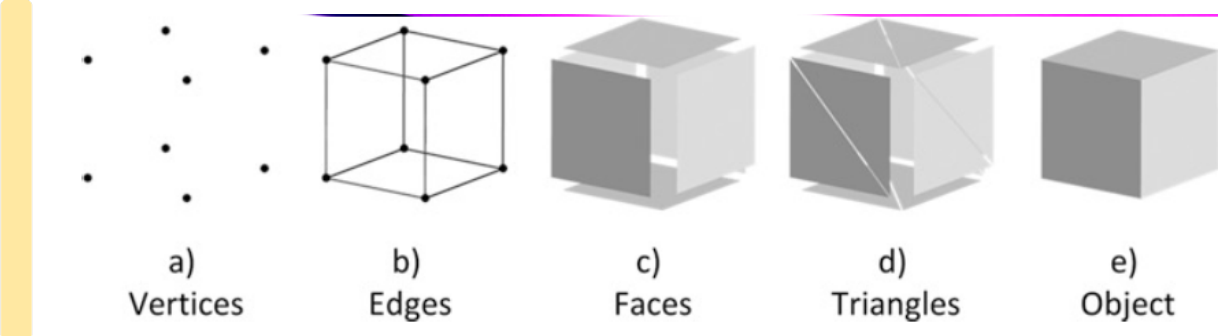
- **surface model**: non sono pensati per includere un volume 3D, quindi ci interessano i poligoni della surface ma non ci interessa se all'interno c'è un oggetto.
- **solid model**: modelli 3d che modellano un elemento che racchiude un volume. Si può quindi ipotizzare di fare qualcosa di fisico con questo volume, per esempio le collisioni.

Surface models

Sono dati dall'unione di tanti poligoni. Di solito si usano triangoli. I vantaggi sono che ci sono tante librerie efficienti per questo tipo di modelli, che possono essere gestiti bene. Il contro è che non si riesce a fare una superficie curva, dovresti mettere tanti triangoli piccoli ma non sarà mai una superficie rotonda.

Sono usati solamente **poligoni planari** (i cui vertici sono piatti) (di solito triangoli o quadrilateri).

L'insieme delle facce (composte da poligoni) compongono la superficie dell'oggetto.

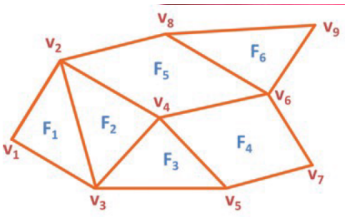


Polygon mesh

La **mesh di poligoni** è un insieme di poligoni che descrivono una faccia (surface). Si possono modellare triangoli o poligoni qualunque.

Ci sono due liste: una di vertici e una delle facce (che dice quali vertici compongono la faccia).

Questo tipo di struttura permette di avere un numero di vertici a scelta, però non è molto efficiente.

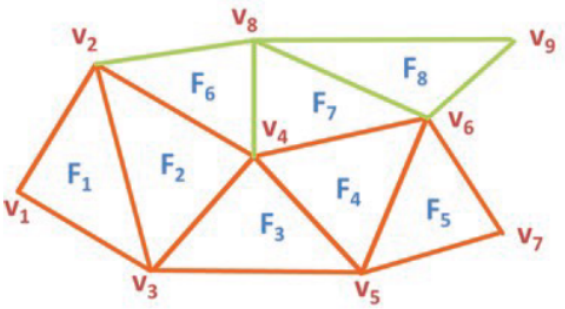


Vertices	
v_1	$x_1 y_1 z_1$
v_2	$x_2 y_2 z_2$
v_3	$x_3 y_3 z_3$
v_4	$x_4 y_4 z_4$
...	...
v_9	$x_9 y_9 z_9$

Faces	
F_1	$v_1 v_3 v_2$
F_2	$v_2 v_3 v_4$
F_3	$v_3 v_5 v_4$
F_4	$v_5 v_7 v_6 v_4$
F_5	$v_2 v_4 v_6 v_8$
F_6	$v_6 v_9 v_8$

Triangle strips

Questa struttura si limita solo ai triangoli, ed è più efficiente.



Triangle Strip$_1$	$v_1 v_3 v_2$	$\rightarrow F_1$
	v_4	$\rightarrow F_2$
	v_5	$\rightarrow F_3$
	v_6	$\rightarrow F_4$
	v_7	$\rightarrow F_5$
Triangle Strip$_2$	$v_2 v_4 v_8$	$\rightarrow F_6$
	v_6	$\rightarrow F_7$
	v_9	$\rightarrow F_8$

Definisco la prima faccia con i suoi 3 vertici, a quel punto al posto di ripetere tutti e 3 i vertici ne riporto solo uno, usando 2 dei precedenti (di modo che gli edges non si incrocino). Quindi solo N + 2 vertici devono essere definiti per N triangoli, contro i 3 * N della struttura precedente. Questo la rende molto più efficiente.

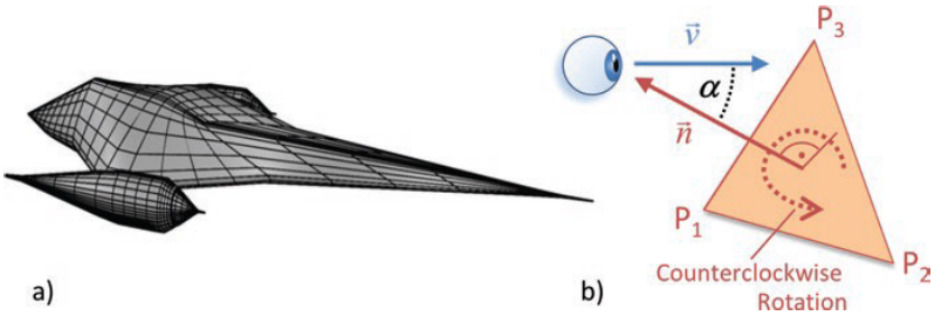
Solid models e B-Rep

Un solid object serve per potergli dare una massa, dargli un centro (calcolato dall'engine), e quando questo oggetto deve rispondere alle collisioni con altri oggetti o con l'utente.

Per la collision detection può essere vantaggioso approssimare gli oggetti con **bounding volumes (bounding representation, b-rep)** che ne definiscono il

bordo in modo più semplice, per semplificare il lavoro dell'engine nelle collisioni.

Bisogna poter fare una distinzione tra l'interno e l'esterno di una boundary face, i poligoni del retro possono non essere mostrati.



Fib a: an example of a b-rep solid
Fig b: The polygon normal ($\vec{n}$) and the viewing direction ($\vec{v}$) form an angle α.
If α is between 90° and 270°, the viewer if looking at the front of the polygon

Nel rendering di un'oggetto ci interessa quale faccia dell'oggetto sta vedendo. Quindi si va a calcolare per ciascuna delle facce dell'oggetto il vettore normale, ovvero una freccia che esce perpendicolare al poligono planare, che ci dice l'orientamento della faccia. Quindi se l'utente sta guardando quel poligono, si può calcolare l'angolo alpha tra il vettore del poligono e quello dell'utente, allora se è abbastanza piccolo (fino a 70-90°) allora è nel campo visivo dell'utente. Man mano quelle con angoli più alti saranno renderizzati con dettagli più bassi, oppure non saranno renderizzati.

Primitive instancing

Alternativamente tramite le **primitive** (cilindri, capsule, sfere) si può andare a giocare con le dimensioni di questi elementi, per fare un bounding volume accettabile rispetto all'oggetto reale. Chiaramente si ha poco margine di personalizzazione su questi oggetti primitivi.

Se il bounding volume è piazzato troppo lontano dall'oggetto reale però, questo può creare problemi di realismo perché le collisioni non sono corrette.

Appearance

Come posso dare un'aspetto realistico agli oggetti? Questo può essere fatto sia dentro che fuori il game engine (con modelli che hanno già una texture etc).

Si entra nell'ambito dei **materials**. Ci sono diversi tipi di material models e material systems, dipendono dall'engine. Qui ci preoccupiamo del colore, la trasparenza, la riflessione della luce.

Il **modello di phong** (più vecchio, più efficiente) utilizza la **luce ambientale** e come si riflette sull'oggetto, utilizza la **luce diffusa** (data la superficie, e come arriva la luce, il colore dell'oggetto), e la **specular reflection** cioè come brilla. Non si riesce a modellare una scena super realistica, ma è più leggero sul pc.

I game engine moderni usano la **physically based rendering (PBR)** che crea scene fotorealistiche, ma richiede molta potenza computazionale. L'obiettivo è proprio quello del massimo realismo possibile.

Utilizza questi parametri:

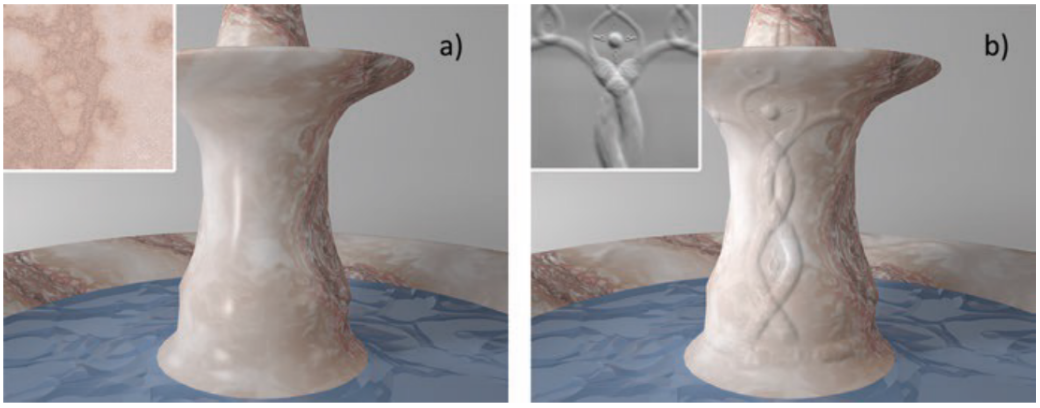
- Albedo: il colore base dell'oggetto
- Aspetto metallico, range da 0 a 1 che dice quanto è metallico, e quindi come la luce sarà riflessa
- Ruvidità, un valore tra 0 e 1

Vengono date anche informazioni sulla trasparenza e sulla lucidità.

Di solito viene aggiunta una texture, ovvero un'immagine che viene assegnata ad un'oggetto. Spesso le texture sono un trucco per andare ad ingannare l'occhio dell'utente, cioè se ho modelli 3D troppo complessi posso semplificare il modello ed applicare una texture che fa percepire all'utente una geometria che non c'è.

Lo step successivo sono le **bump mapping, i normal mapping e i displacement mapping**.

La displacement è l'unica che va effettivamente a cambiare la geometria dell'oggetto, mentre le altre 2 giocano con la luce e come viene riflessa, ma non cambiano la geometria.



Queste mappe vanno a giocare sul vettore normale e quindi sull'effetto di profondità. L'**ambient occlusion** ci dice per ogni punto di questa mappa, come viene ricevuta la luce esterna e come va mostrata.

Nel **bump mapping** la mappa è in scala di grigi, che ci dice i punti chiari e scuri. Questa viene sommata alla texture.

Nel **normal mapping** sono delle varianti del bump mapping che danno direttamente il vettore normale salvato nella normal map, non bisogna sommare nulla.

Le **shader** sono dei programmi eseguiti dall'hardware grafico (GPU) che rendono possibili dei design ancora più variegati (e migliori) per le superfici degli oggetti.

Animazioni

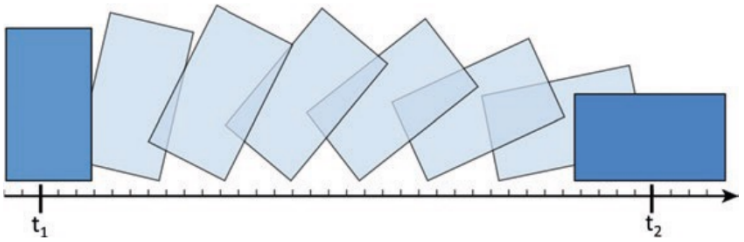
Tendenzialmente gli oggetti non saranno tutti statici, alcuni dovranno manifestare un comportamento.

Un oggetto è modellato da un insieme di proprietà, alcune caratteristiche built in del sistema, altre specificate da noi, come la posizione, la rotazione, etc, queste sono proprietà che possono cambiare nel tempo.

Per descrivere un oggetto gli do delle proprietà, che hanno un valore.

Ci sono anche animazioni basate sulla fisica, sono più realistiche e più complesse, vanno assegnati dei volumi per le collisioni. Queste si basano sulle leggi della fisica, quindi il game engine si preoccupa di fare queste animazioni.

I **keyframe** sono gli step a livello di tempo per la sequenza dell'animazione. Quindi io posso dire come cambia la proprietà da T1 a T2, e poi sarà il game engine a calcolare il come cambiare le proprietà in ogni istante di tempo.

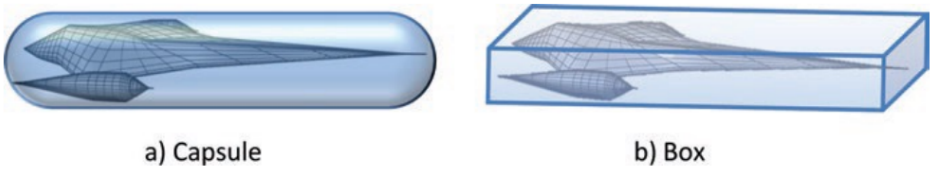


L'animazione quindi, dato che è fatta dal game engine, potrebbe anche essere diversa rispetto a quella che mi aspettavo, potrebbe non essere realistica.

Gli oggetti 3D devono avere un **rigid body**, che possa reagire alla fisica. I sistemi moderni di game engine (gestiscono il mondo, l'interazione, ...) contengono un altro engine che si preoccupa della fisica. Normalmente non mettiamo mano a quell'engine, non possiamo cambiare le regole che usa, ma possiamo settare il nostro ambiente di modo che il game engine possa gestirlo, quindi mettendo i colliders, i rigid body, assegnando la massa degli oggetti, la velocità iniziale, l'elasticità.

Configurando il mondo iniziale e le caratteristiche degli oggetti, il game engine sarà in grado di occuparsi della simulazione e l'animazione di tipo fisico.

Le collisioni sono fatte creando dei bounding volume, ovvero aggiungendo un corpo rigido agli oggetti geometrici, creando il **collision proxy**.



Il collision proxy non è renderizzato e può essere semplice, come sfere, cubi o capsule, ma può anche essere più complesso, tramite una **convex hull** (una mesh di poligoni solidi).

Gli oggetti più complessi sono collegati da punti di giuntura, come per esempio il braccio di una persona collegata al corpo. Quindi il motore fisico gestirà i motion constraint di conseguenza, di modo che non dobbiamo farli a mano.

Alla fine di ogni step di simulazione, l'engine deve capire dove sono gli elementi per fare il rendering. Il motore fisico darà quindi in output tutte le matrici di trasformazione degli elementi, queste vengono integrate nel geometric world e l'engine capisce dove disegnare gli elementi.