

Idee sulla gestione della memoria di Cassata

Fulvio Satta

22 luglio 2007

Quanto segue va preso esplicitamente come idee ed appunti scritti per non dimenticarsi tutto e non come qualcosa che verrà implementata in Cassata.

Molte potrebbero essere le castronerie e le cose da vedere e reinterpretare scritte in questo documento, che va preso solo in via puramente indicativa e con la concezione che possa cambiare tutto, tuttavia se qualcuno volesse esprimere qualche giudizio in merito è il benvenuto.

1 Il problema

Un rendering può occupare davvero molta RAM. Tuttavia la maggior parte di questa è dovuta a calcoli che non è necessario rieseguire continuamente durante il rendering.

Evitare di memorizzare dati in RAM se non quelli strettamente necessari porterebbe a grossi problemi prestazionali. Di contro memorizzare qualunque dato indiscriminatamente porta ad un'occupazione enorme della RAM in alcuni rendering, ed a volte persino all'impossibilità di portare a termine un rendering per via dell'insufficienza di memoria indirizzabile (problema molto sentito nelle ancora comuni architetture a 32 bit).

Una possibile soluzione potrebbe sembrare quella di eseguire della cache sul disco, enormemente più grande della RAM, ma questo non funziona perché il disco è troppo lento, e spesso è conveniente rieseguire i calcoli piuttosto che rileggerli. Questo però non vuol dire che questa sia la regola generale, a volte può convenire usare il disco.

Il problema è perciò riuscire a bilanciare prestazioni ed occupazione in RAM, capendo quando e quali dati ricalcolare, quando è conveniente usare una cache sul disco e come svolgere tutte queste operazioni. Il tutto dev'essere ampiamente controllabile dall'utente, che deve avere la possibilità di decidere quanta RAM e quanto disco utilizzare. A complicare le cose c'è anche la cache, che se ben sfruttata permette di accelerare ulteriormente il rendering.

2 Cosa memorizzare

Il tema sarebbe piuttosto complesso, se non fosse per una proprietà davvero interessante del rendering: la coerenza della scena.

La maggior parte dei raggi (secondo alcuni test fatti sono la quasi totalità) sono tra di loro coerenti nello stesso intorno. Questo vuol dire che se un raggio, ad esempio, colpisce un oggetto ci sono ottime probabilità che un raggio vicino colpisca lo stesso oggetto.

Per questo è possibile usare algoritmi standard sulla gestione della cache, come LRU, FIFO od LFU.

Tuttavia avere una gestione uniforme della cache non è conveniente. Il renderer usa diversi tipi di dati, ed ogni tipo di dato può avere un buon modo di gestire la propria cache.

In particolare si può gestire il tutto in blocchi (non necessariamente definiti come tali). Ogni blocco ha una tag che indica quando può essere eliminato, se ad esempio è conveniente eseguire la cache sul disco o qualcos'altro. Si può anche dare una priorità a questi blocchi tramite la tag. Tutto questo può essere svolto anche all'interno di macroblocchi che condividono la stessa tag, e quindi la procedura di tagging può essere implicita, per quanto presente.

3 Come memorizzare

La memorizzazione andrebbe fatta in memory pools, in modo da aumentare la consistenza della cache.

Questo in apparenza sembra in disaccordo con la possibilità di eliminare delle cose, ma non è così. In effetti semplicemente si tiene un memory pool che contiene l'intera dimensione del macroblocco memorizzato, e quindi si sostituiscono all'interno i microblocchi.

È necessario che le fasi di cancellazione dei microblocchi siano abbastanza ampie, così da poter inserire microblocchi coerenti in aree di memoria contigue (il che aumenta l'uso intensivo della cache), ma che contemporaneamente non siano troppo grossi tanto da eliminare troppi dati che potrebbero servire successivamente. Forse una politica adattativa a seconda dell'uso dei dati che se ne fa statisticamente è la cosa ideale.

Inoltre è necessario disporre di una struttura dati adeguata a salvare gli intervalli non utilizzati di microblocchi. Potrebbe essere utile un albero bilanciato con possibilità di avere elementi con chiavi identiche, dove la chiave è il numero di microblocchi liberi, così da poter avere uno spazio di dimensioni adeguate per il dato da memorizzare (che può occupare più microblocchi) in un tempo logaritmico.

Le scritture dovrebbero il più possibile essere raggruppate, in modo da poter sfruttare meglio possibile il bus, ma tuttavia non so tecnologicamente come si sia combinati su questo fronte, perciò potrebbe non essere necessario, è tutto da controllare.