

Rockland

Tito Dal Canton

1 maggio 2003

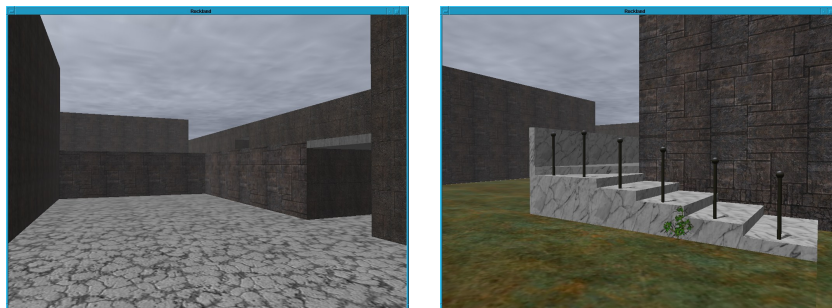
Esperimento di motore 3D per creare ambienti realistici in cui muoversi
<http://www.dalcanton.it/tito/progetti/rockland>

01/05/2003

FUNZIONA. Ho praticamente finito la parte *tosta* del motore lightmap, e sembra proprio funzionare benissimo, sebbene gran parte del codice non sia per nulla elegante. Per le ombre portate ho riutilizzato il vecchio codice che creava i poligoni trasparenti, poi sovrapposti alle superfici. Anzichè venire “messi via”, i poligoni ombra vengono ora proiettati in coordinate lightmap e... disegnati nel buffer dei lumel (ho riutilizzato un vecchio algoritmo per disegnare poligoni che usavo ai tempi della grafica in DOS ;-). Dato che questi lunghi calcoli vengono svolti come fase di preprocess—in `BuildLightmap()`—e le ombre sono poi applicate come seconda texture, il tutto è veramente molto veloce e comodo. L’unica cosa importante da fare è rendere *adattiva* la risoluzione delle lightmap, perchè ovviamente non posso avere la risoluzione che uso per uno scalino anche per una parete di 20m². Inoltre, se su un poligono non ci sono ombre è perfettamente inutile applicarci una lightmap 32x32 quando la cosa funziona ugualmente bene con una banale texture 1x1. La qualità delle ombre dipende molto dalla risoluzione, ma credo che comunque mi accontenterò di lightmap molto piccole (a meno di non trovarmi a lavorare su una Matrox con 256 Mb di RAM).

29/04/2003

Sto sperimentando con le lightmap, ottenute tramite multitexturing. Sembra una cosa molto pulita, elegante e soprattutto veloce. La qualità dell’illuminazione non è eccellente, ma di sicuro non è peggiore di quella nativa OpenGL. Implementare le ombre portate non dovrebbe essere troppo difficile.



Per calcolare il valore di un singolo *lumel* (pixel della lightmap) utilizzo semplicemente la $l = -\vec{N} \cdot \vec{L}$ dove \vec{N} è la normale al punto illuminato ed \vec{L} è il versore direzione della luce del sole. Siccome \vec{N} ed \vec{L} hanno norma 1 per costruzione, il prodotto scalare è esattamente il coseno dell’angolo luce/normale, e la luminosità è quindi proporzionale a questo coseno—come deve essere. Il cambiamento di segno è dovuto al fatto che quando la luce

è rivolta verso il poligono, luce e normale sono vettori con angolo superiore a 90° , quindi il prodotto scalare è negativo. Se invece esso è positivo, assegnamo luminosità zero al punto—significa che la luce proviene da “dietro” il poligono, e non può quindi illuminarlo. Infine viene aggiunto il valore della luminosità ambientale al lumel, per dare un po’ di luce anche dove non batte direttamente il sole. Soddisfacente.

In definitiva credo che utilizzerò proprio questo sistema per l’illuminazione, in quanto ho anche notato parecchi bug riguardanti lo *stencil buffer* testando Rockland in Linux sui Mac (ATI Rage e Radeon).

Ho finalmente compiuto la transizione di questo documento da formato ASCII al più consono e figo L^AT_EX.

25/03/2003

Non ci siamo assolutamente con l’illuminazione. Se uso quella nativa OpenGL, devo tessellare in modo inquietante il mondo (appesantendolo), e non riesco a implementare le ombre portate. Se tento di risolverla sovrapponendo poligoni trasparenti o con lo stencil, sorgono n problemi di depth sorting. Credo che la soluzione migliore siano le *lightmap*, utilizzando le funzioni di multitexturing—tecnica molto usata anche nei giochi, guarda caso. Per i punti luce dinamici (proiettili ecc.) potrei invece usare una semplice “macchia” con un `TRIANGLE_FAN` reso opportunamente trasparente e grande a seconda della distanza tra la luce e la superficie. Comunque, dovrebbe essere abbastanza difficile comporre le lightmap tenendo conto dei vari punti luce e della luce del sole. . .

13/03/2003

Ho capito come risolvere la questione illuminazione-orrenda. Bisogna tessellare. Bisogna suddividere i poligoni grandi in tanti “tasselli” piccoli (più sono meglio è, idealmente dovrebbero essere infiniti :-). Infatti, così facendo ci sono più normali che caratterizzano l’andamento della superficie (una per ogni vertice di tasselli) quindi l’illuminazione è *molto* più precisa, e tende alla perfezione al crescere delle suddivisioni. In particolare, se si vuole rendere correttamente una luce in movimento (proiettile p.es.) non c’è altro modo che avere superfici tessellate. La tecnica del *vertex lighting* non consente di fare altrimenti.

10/03/2003

Prove di *environment sphere mapping* su paletti e superfici metalliche riflettenti. Buoni risultati.

09/02/2003

Il corso di geometria vettoriale a Pd ha un sacco di applicazioni pratiche in Rockland...dovrò implementare tutte queste nuove conoscenze appena mi sono liberato del tutto.

20/01/2003



Aggiunto, in onore degli uomini MeTA~LAbS, il *doge*, figura semi-inquietante e apparentemente inutile che si aggira per le map sections muovendosi secondo i ritmi del caso. Inutile chiedersi come sia capitato là, o per quale ragione.

Ampliato `original-complex.rml`.

18/12/2002

Implementata una bozza di rilevamento collisioni tra oggetti, e un miglior collision detection tra camera e pareti. Ora è impossibile “affacciarsi” oltre le pareti, perchè la camera ne viene sempre tenuta lontana di una distanza almeno pari al suo raggio.

14/11/2002

Implementato un tubo di metallo da cui esce un vapore fin troppo realistico. E si può addirittura controllare il vapore con un rubinetto! Davvero impensabile.

28/09/2002

Aggiunta una gradevole illuminazione basata sulle normali ai vertici. Rockland sta diventando abbastanza accogliente per ospitare un' IA ;-) Comunque—anche se molto veloce, con la GeForce 2—l'illuminazione è di qualità piuttosto scandente, e ci sono parecchie imprecisioni specialmente in zone ad alta densità di vertici. Errori di implementazione da parte mia? Non credo: le normali dei vertici sono calcolate con risultati molto soddisfacenti, in teoria dovrebbe essere tutto a posto. Boh. Prima o poi mi comprerò quel terribile mattone su OpenGL...

25/06/2002

Introdotti gli oggetti e sistemato quasi perfettamente il rilevamento delle collisioni degli oggetti stessi con il mondo. La cosa bella è che non è necessario calcolare complesse mappe di collisione, come avviene per gli altri motori 3D, ma rockland calcola tutto in tempo reale a partire dalla posizione delle varie map section. Questo semplifica enormemente le cose, ma probabilmente non è il sistema più efficiente (e intelligente :-).

14/01/2002

Il porting su SDL è completo, funziona tutto abbastanza bene. Ora lo sviluppo di Rockland continua sotto Linux (gcc è più flessibile di CodeWarrior e soprattutto posso lavorare nei ritagli di tempo). Migliorato il sistema di rilevamento delle collisioni con il mondo (ora si scivola sulle pareti! :-). Modificato il formato delle mappe `.rml` in modo che non sia più necessario specificare un numero per ogni poligono, ma basti farlo solo per i poligoni "importanti" (p.es. pavimento e soffitto di una map section), tramite il tag `<id n>` in `<poly>`. Questo ID può essere usato nel tag per le map section `<ms foobar>`.

06/01/2002

Dopo un prolungato calo di interesse per il Mac torno ad occuparmi di Rockland... per portarlo su Linux. Ma non solo linux: infatti Rockland sarà portato su SDL, Simple Directmedia Layer, una API OpenSource multi-piattaforma che permette di gestire grafica, OpenGL, audio e input in modo perfettamente identico su quasi tutti i sistemi operativi (compreso Windows hah). Il vecchio Rockland per Mac scomparirà quindi, essendo tra l'altro incompatibile con il nuovo MacOS X.

03/03/2001

Finalmente ho capito come funziona il sistema di coordinate di OpenGL! La X aumenta verso destra, la Y aumenta verso l'alto e la Z aumenta verso il giocatore (cioè “viene fuori” dallo schermo), al contrario di quello che si pensa normalmente. Ho sistemato dunque tutto il codice, cambiando il segno di tutte le Z dei poligoni in fase di rendering. Questo perchè, secondo me, il fatto che la Z sia rivolta verso la camera è un po' strano e può risultare scomodo. Nella realtà di Rockland, quindi, a Z maggiori corrispondono distanze maggiori dall'osservatore.

18/01/2001

Ecco che inizio a scrivere questo file! La mia storia passata sta scomparendo dalle mie memorie, quindi devo iniziare a tener traccia dei cambiamenti più importanti. Mi dispiace solo di aver perso quasi completamente le date precise della storia di Rockland. Ma non dimenticherò mai il primo Rockland: un grande cortile sterile di roccia, con un serbatoio metallico al centro. Era il 1998, OpenGL era un sogno lontano e QuickDraw 3D RAVE—allora l'unica API per accedere all'hardware 3D sui Mac—turbava le mie notti con assurdi bug. L'idea del rilevamento delle collisioni era ancora lontana, e vagavo per il mondo in cerca dell'algoritmo per determinare se un punto è dentro ad un poligono.

Poi tutto cambiò: arrivarono le idee, il cortile divenne qualche corridoio, poi una serie di stanze e scalette. Allora mi si presentò il terribile limite: uno stranissimo bug di RAVE. La rimozione dei poligoni nascosti non funzionava a dovere, mostrando in primo piano alcune stanze lontane. Stanco di tentativi senza successo, abbandonai Rockland a malincuore. Poi ecco: Apple decise di implementare OpenGL nativamente nel MacOS. La rinascita. L'illuminismo della grafica 3D. Rockland venne riscritto da zero, ed acquistò nuova vita. Movimenti di una fluidità inaudita, frame rate spaventosi, e, soprattutto, niente più bug di hidden surface removal. Il porting durò diverse settimane di vario impegno, ma alla fine ero soddisfatto. Un problema con il sistema di coordinate mi obbligò ad abbandonare il vecchio mondo di Rockland e testare il nuovo motore su mappe molto più piccole.

Adesso che tutto è risolto ricreerò quella serie di corridoi e stanze in cui tanto ho vagato ai tempi di RAVE. L'ultima versione RAVE è stata compilata nel giugno 2000, e da quella data non ho più toccato RAVE (né ho intenzione di farlo mai più). Qualche versione dell'antico Rockland è salvata per sempre nei miei CD di backup—la mia infanzia informatica.

Oggi ho apportato le seguenti modifiche:

- Implementato un nuovo formato di file mappa: è una specie di XML, funziona a tag ed è piuttosto comodo e versatile (2 palle scriversi la descrizione di un mondo attraverso l'XML però...)
- L'angolo di elevazione va ora da -60 gradi (quando ti guardi i piedi) a +60 gradi (quando scruti il cosmo). Prima andava per conto suo :-)
- Ho rimosso il noioso limite dei 60 FPS, necessario con la prima versione di OpenGL per un difetto di InputSprocket. Adesso è molto piu' fluido.
- Tolle alcune parti di vecchio codice ora inutile, migliorate alcune cosette di scarso interesse.