Ciao Marco,

provo a documentare le questioni che devo affrontare. Magari spiegando riordino le idee a mia volta. Adesso ogni oggetto sullo schermo è composto a un cluster di sprite ed è codificato da questo struct

typedef struct {

enum ObjStat Status;

enum ObjType Type;

i16 x,y;

u8 NumSprt;

u8 \*dx,\*dy;

u16\* pat;

u16\* col;

i16 rx,ry,rz;

i16 vx,vy,vz;

} Object;

I campi x,y sono le coordinate 2D sullo schermo. I campi rx,ry,rz sono le coordinate nello spazio 3D. I campi vx,vy e vz sono velocità nello spazio 3D. Il resto dei campi serve per gestire il rendering degli sprite che compongono l’oggetto e lo stato dell’oggetto.

Per proiettare da 3D a 2D gli oggetti, le coordinate rx ed ry sono trasformate in x e y secondo la regola

y = d\*ry/(d+rz).

Analogamente per x ed rx. Oggi il codice è questo:

// d = distanza z dallo schermo della camera (distanza focale)

// xc,yc posizione della camera sul piano dello schermo

#define d 128

#define xc 128

#define yc 128

…

// angolo di tilt lungo l’asse di vista (va dalla camera verso il piano dello schermo)

// COS[] e SIN[] sono i16 scalati per 256

cs = MyCos[tilt]/2;

sn = MySin[tilt]/2;

// il /2 è una ottimizzazione per tenere conto del fatto che 256/2 = 128 = d e che non ho messo la distanza focale nella formula della proiezione

…

// pMyObj punta all’oggetto corrente

// z è la distanza tra camera e oggetto corrente

i16 z = d+pMyObj->rz;

// esegue la traslazione attorno al centro della rotazione, la rotazione di tilt, la proiezione (scalando per z -NB manca d al numeratore ma è assorbito dal fattore di scala di cos e sin e dal /2) e trasla nuovamente

pMyObj->x = ((pMyObj->rx-xc)\*cs-(pMyObj->ry-yc)\*sn)/z + xc;

pMyObj->y = ((pMyObj->rx-xc)\*sn+(pMyObj->ry-yc)\*cs)/z + yc;

// sceglie il frame da visualizzare tra le scale 16 disponibili per 30 possibili angoli

// poiché la distanza massima è 320, divido per 20 per riportare il risultato tra 0 e 15

u8 frame = (pMyObj->rz<0) ? (0):(pMyObj->rz/20);

// le dimensioni dei frame precalcolate tengono conto del coefficiente di proiezione d/z

// imposta per l’oggetto i, il frame corrispondente

SetObjFrame(i,frame+(33+tilt\*16));

Gli stessi semplici calcoli con i medesimi parametri sono sviluppati in matlab per renderizzare le onde nei 30x3 = 90 frame che poi comprimo con la quantizzazione vettoriale.

**Cosa potrei adattare da E3D:**

1. Il framework generale per orientare la camera in una direzione qualsiasi
2. La struttura degli oggetti 3D da manipolare
3. Eventuali funzioni per la gestione di collisioni e angoli di vista

**Cosa servirebbe in aggiunta:**

1. Un framework per rappresentare in dati le traiettorie dei nemici
2. Una regola per associare l’angolo delle ali rispetto all’orizzonte in funzione del raggio di curvatura della traiettoria
3. Una regola per animare il mirino del giocatore nello spazio 3D collegandola al movimento dell’aereo del giocatore
4. Una regola più realistica per rappresentare il movimento dell’aereo del giocatore

Trovi il sorgente al link. Il file C che fa queste trasformazioni è

template.c

Fammi sapere se hai osservazioni o domande

AR