Claudio Bacchetta

Daniele Ingrosso

7 dicembre 2020

Tutorial Motore Grafico

Premessa

In questo tutorial non spiegherò tutte le parti del codice perché altrimenti risulterebbe un opera megalitica, mi limiterò a spiegare quindi solo le parti fondamentali del codice.

Specifico inoltre che questo NON è il codice originale del Doom engine, questo è un motore grafico simil-doom, le caratteristiche di questo è che risulta più semplice e incompleto rispetto all'originale perchè fa affidamento su una dll esterna chiamata PixelToaster che permette di creare il proprio software di rendering; inoltre PixelToaster permette di avere input da mouse e tastiera.

Per quanto riguarda invece gli strumenti servirà un ambiente di sviluppo c++ (questo programma è stato sviluppato con visual studio 2019), la libreria PixelToaster e la libreria cstdint.

Il codice è descritto in ordine di scrittura e non di esecuzione.

Codice

```
#include <cstdint>
#include "PixelToaster.h"
#include <iostream>

using namespace PixelToaster;
using namespace std;

//risoluzione
#define SCREEN_W 640
#define SCREEN_H 480

//campo visivo
#define FOV 0.66
```

- All'inizio includo le due librerie che mi serviranno ovvero "cstdint" che mi fornisce integer di larghezza fissa; "PixelToaster" la libreria già accennata prima che mi servirà per il rendering grafico e per gli input da mouse e tastiera.
- Proseguo definendo la risoluzione dello schermo all'avvio ovvero un 640x480 una finestra di dimensioni ridotte; per poi andare a definire il fov ovvero il campo visivo a 66°, il fov non deve avere ne valori troppo alti ne troppo bassi per evitare una visuale distorta.

```
//movimento,passi,rotazione
#define MOVSPEED 0.5
#define ROTSPEED 0.04

//framebuffer
struct Frame {
        uint32_t width;
        uint32_t height;
        TrueColorPixel *data;
};
//mappa
struct World {
        uint32_t width;
        uint32_t height;
        uint8_t *data;
};
//telecamera
struct State {
        double posx;    //posizione
        double posy;
        double dirx;    //direzione
        double diry;
        double camx;    //piano proiezione
        double camy;
};
```

- Definisco la velocità di movimento e la velocità di rotazione (non modificare con valori troppo alti perchè si rischia un crash del programma)
- Creo la struttura Frame e world che devono essere delle stesse dimensioni della finestra aperta. La struttura state o stato rappresenta le caratteristiche della telecamera che sono sei (due per categoria): posizione,direzione e piano di

proiezione. Questi rappresentano: la posizione del giocatore che essendo bidimensionale avrà solo coordinate (x,y); la direzione in cui sto guardando che di nuovo avrà coordinate (x,y); ed infine il piano di proiezione o piano di camera che rappresenta lo schermo del computer che avrà come gli altri coordinate (x,y).

```
void onMouseMove(DisplayInterface& display, Mouse mouse)
{
	double oldDirX = 300;
	printf("onMouseMove:  x=%f\n",
		mouse.x
		);
	oldDirX = mouse.x;
	if (oldDirX < 200) {

		double oldDirX = state->dirx;
		state->dirx = state->dirx * cos(rotSpeed) - state->diry * sin(rotSpeed);
		state->diry = oldDirX * sin(rotSpeed) + state->diry * cos(rotSpeed);
		double oldcamx = state->camx;
		state->camx = state->camx * cos(rotSpeed) - state->camy * sin(rotSpeed);
		state->camy = oldcamx * sin(rotSpeed) + state->camy * cos(rotSpeed);
	}
	if (oldDirX >400 ) {

		double oldDirX = state->dirx;
		state->dirx = state->dirx * cos(-rotSpeed) - state->diry * sin(-rotSpeed);
		state->diry = oldDirX * sin(-rotSpeed) + state->diry * cos(-rotSpeed);
		double oldcamx = state->camx;
		state->camx = state->camx * cos(-rotSpeed) - state->camy * sin(-rotSpeed);
		state->camy = oldcamx * sin(-rotSpeed) + state->camy * cos(-rotSpeed);
	}

}
```

- In questo passaggio usando assets della dll PixelToaster creo una funzione in grado di prendere in input segnali dal mouse, poi dichiaro la posizione di x (essendo che il mouse mi servirà per spostare la visuale orizzontalmente l'asse y non verrà utilizzato) a 300 che rappresenta indicativamente la metà dello schermo. Dopo aver fatto ciò creo due if con la variabile x prima minore di 200 poi maggiore di 400 in questo modo quando si sposterà il cursore del mouse verso sinistra cioè verso 200 la visuale ruoterà verso sinistra mentre quando si sposterà verso destra cioè i 400 la visuale ruoterà verso destra.

```
void onKeyPressed(DisplayInterface& display, Key key) {
	if (key == Key::W) {
		int currX = (int)state->posx;
		int currY = (int)state->posy;
		int nextX = (int)(state->posx + state->dirx * moveSpeed * 2);
		int nextY = (int)(state->posy + state->diry * moveSpeed * 2);

		if (world->data[nextX + currY * world->width] == 0) {
			state->posx += state->dirx * moveSpeed;
		}
		if (world->data[currX + nextY * world->width] == 0) {
			state->posy += state->diry * moveSpeed;
		}
	}
	if (key == Key::S) {
		int currX = (int)state->posx;
		int currY = (int)state->posy;
		int nextX = (int)(state->posx - state->dirx * moveSpeed * 2);
		int nextY = (int)(state->posy - state->diry * moveSpeed * 2);

		if (world->data[nextX + currY * world->width] == 0) {
			state->posx -= state->dirx * moveSpeed;
		}
		if (world->data[currX + nextY * world->width] == 0) {
			state->posy -= state->diry * moveSpeed;
		}
	}
```

- Per lo spostamento in avanti utilizzo sempre asset di PixelToster ma questa volta per prendere in input segnali da tastiera, in sintesi creo due if dove nel primo se premo o tengo premuto la “w” il giocatore si muove in avanti mentre se premo o tengo premuto la “s” il giocatore si muove indietro.

```
void DrawColumn(RayHit what, World world, State state, Frame frame, uint32_t column) {
	//tipo blocco ril.
	uint8_t type = world.data[what.mapX + what.mapY * world.width];
	//colore tipo blocco
	uint8_t r, g, b;
	switch (type)
	{
	case 1:
	{
		r = 0;
		g = 255;
		b = 0;
		break;
	}
	case 2:
	{
		r = 155;
		g = 155;
		b = 155;
		break;
	}
	case 3:
	{
		r = 0;
		g = 0;
		b = 255;
		break;
	}
	case 4:
	{
		r = 255;
		g = 0;
		b = 0;
		break;
	}
	}//fine switch
```

- A questo punto creo una funzione che all’interno conterrà la composizione dei muri; con uno switch creo quattro casi, ogni caso rappresenta un muro e ogni

muro possiede colori diversi, nella costruzione della mappa quando voglio posizionare un muro colorato andrò a scrivere il numero corrispondente al colore scelto.

```
//cielo
for (i = 0; i < (frame.width * frame.height/2); i++) {
        frame.data[i].r = 135;
        frame.data[i].g = 206;
        frame.data[i].b = 250;
}
//terreno
for (i = (frame.width * frame.height / 2); i < (frame.width * frame.height); i++) {
        frame.data[i].r = 102;
        frame.data[i].g = 51;
        frame.data[i].b = 0;
}
```

- Il processo invece per creare soffitto e pavimento è semplice, si divide orizzontalmente in due lo schermo e alle due parti si dà un colore differente.

```
//map loading
World world;
//world.txt file con la mappa
if (!LoadWorld("world.txt", &world)) {
        cout <<("Error loading world file!")<<endl;
        exit(0);
}
```

- Per quanto riguarda invece il main esso conterrà l'inizializzazione delle variabili di stato, di schermo, del listener ecc; ma anche il loop principale e il caricamento della mappa che andrà a controllare nella cartella in cui si trova il programma se c'è un file .txt chiamato world che rappresenta la mappa di tutto il mondo di gioco.

```
128
128
22222222222222122222222222221222222222222212222222222221       22222222222222222222222222222222222222222222
2             2             2             2             2       2       2       2       2                   2
2             2             2             2             2       2       2       2       2                   2
2             2             2             2             2       2       2       2       2                   2
2             2             2             2             2       2222 2222222 2222222222 222222            2
2             2             2             2             2       2                                           2
2             2             2             2             2       2                               2           2
2             2             2             2             222222222       222 222222 22222 22   2           2
2             2             2             2             2       2       2       2       2   2   2           2
2             2             2             2             2       2       2       2       2   2   2           2
2             2             2             2             2       2       2       2       2   2   2           2
2             2             2             2             2       2       2       2       2   2   2           2
2             2             2             2             2       2       2       2       2   2   2           2
2             22222223 32222222222223 32222222222223 3222222224 4222       22222222222222222221122222222222222222
2                                                                           2
2             3                                                             2
2             22222 222222 22222222222222222222222222222222               2
2             2       2    2                                 2             2
2             2       2    2                                 22222211222222
2             2       2    2
22222222222222212222222222222222
```

- Questa qui sopra è la composizione della mappa di gioco che il programma leggerà, i primi due numeri rappresentano la grandezza massima della mappa in

questo caso corrisponde a un quadrato di 128x128; gli altri numeri che formano le pareti invece indicano la composizione dei muri in questo caso ho creato le pareti della stanza usando il numero 2 che corrisponde al colore grigio nel codice.
Inoltre il piano cartesiano x y ha lo zero nell'angolo in alto a sinistra, nel codice il punto di creazione o spawn è alle coordinate (3,3) che corrisponde alla parte in alto a sinistra della stanza più grande