Documentazione

Alunno: Lorenzo Cappellotto

Titolo: Asteroids (replica con OpenGL)

Istituto: ISISS Antonio Scarpa

Data: 21-05-18

Versione 1.1

Obiettivo:

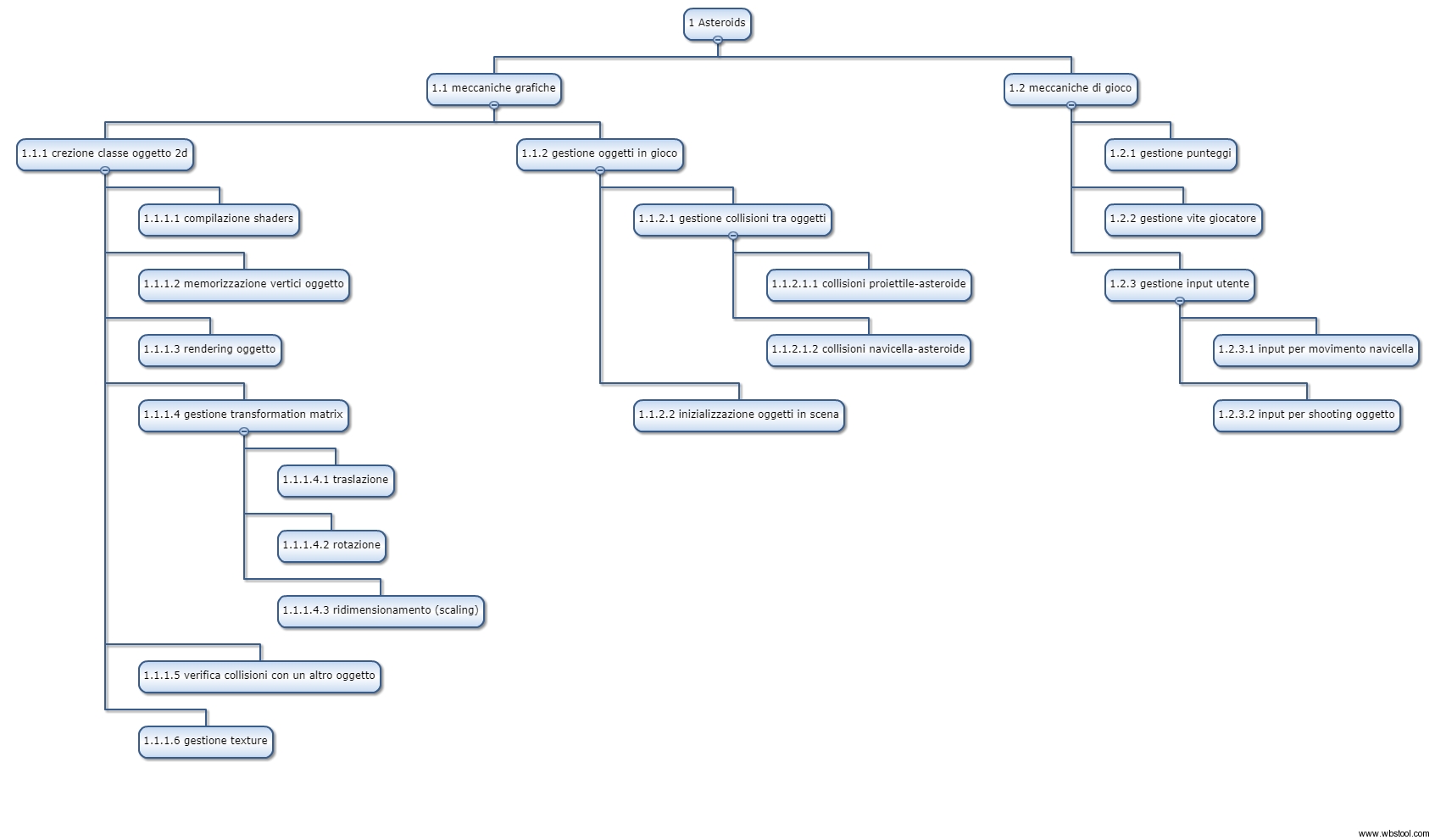
L’obiettivo primario di questo progetto era quello di replicare, nelle meccaniche principali sotto forma di applicazione Windows, il gioco Asteroids usando il linguaggio di programmazione C++ e le API OpenGL. L’obiettivo secondario che mi ero posto, invece, era quello di comprendere il funzionamento delle API grafiche OpenGL in modo da ampliare le mie conoscenze in fatto di computer grafica (anche se vengono rappresentati solamente contenuti grafici 2D, le tecniche applicate rappresentano la base anche per la grafica tridimensionale). Infine vi era un obiettivo trasversale, ovvero quello di riuscire a gestire un progetto informatico dal concepimento fino al suo rilascio.

Lo scopo del gioco è di riuscire a sparare e distruggere tutti gli asteroidi a schermo prima di scontrarsi abbastanza volte da terminare le vite a disposizione. Il gioco si conclude una volta che tutti gli asteroidi vengono distrutti, stampando a terminale il punteggio ottenuto e le vite rimanenti.

WBS al termine del progetto:

il WBS è stato modificato durante lo svolgimento del progetto. È stata aggiunta all’interno della classe oggetto la gestione delle texture in modo da poter differenziare gli oggetti anche sfruttando una immagine anziché solamente in base alle forme costruite dai vertici. A quel punto è stato necessario abbandonare la gestione dei livelli e una parte della dinamica degli asteroidi, i quali, una volta colpiti, avrebbero dovuto suddividersi e generare degli effetti particellari. Queste scelte sono state fondamentali per rimanere in linea con i tempi di consegna previsti.

IMG



Funzionamento:

A livello di programma in esecuzione, il progetto può ricevere input solo da tastiera e restituire le informazioni in output attraverso due finestre distinte.

Gli input che il programma può ricevere riguardano solamente la pressione di alcuni tasti, essi permettono all’utente di muovere la navicella all’interno della finestra (nel caso in cui la navicella, cosi come gli atri oggetti, esca dal campo visivo attraversando uno degli spigoli essa verrà traslata sullo spigolo opposto a quello colpito in modo da dare un effetto di campo chiuso)

I tasti intercettati dal programma si dividono in tre categorie:

* W S o ^ v, per far avanzare la navicella
* A D o < >, per far ruotare la navicella attorno al proprio asse
* Barra spaziatrice, per far sparare alla navicella dei proiettili nella direzione in cui si sta muovendo

Le informazioni restituite in output, invece, si dividono in:

* un terminale, in cui vengono stampate i dati relativi alla partita in corso, come il numero vite, il punteggio ottenuto e altri parametri riguardanti le collisioni
* una seconda finestra al cui interno vi è rappresentato il gioco vero e proprio attraverso computer grafica.

A livello di codice, invece, il progetto può essere suddiviso in 2 parti fondamentali:

* la classe Oggetto. Questa classe si occupa, attraverso le API OpenGL, di tutta la parte di gestione degli oggetti bidimensionali in gioco. Ogni oggetto è caratterizzato da un insieme di vertici, una texture, dei parametri fisici (velocita, angolazione, …) e da dei programmi per GPU per il rendering (generazione immagine).

Nel mio caso ho usato la libreria GLEW per il caricamento della corretta estensione di OpenGL, diversa in base alla piattaforma in cui viene eseguito il programma, e la libreria GLM per eseguire elaborazioni di tipo matematico all’interno della classe, ad esempio la rotazione dei modelli.

Questa classe permette quindi eseguire numerose azioni sui modelli (o oggetti), qui sotto elencherò le principali:

* caricare i vertici. Ogni modello necessita di un insieme di vertici, ossia di un certo numero di punti e di collegamenti, che poi saranno i dati passati alla GPU per ottenere delle immagini grafiche. Per memorizzare i vertici sono necessari tre costrutti di OpenGL: il buffer VBO per memorizzare i vertici, il buffer EBO per memorizzare l’ordine con cui vanno scelti (i collegamenti) e il vettore VAO per collegare i buffer descritti in precedenza in un'unica struttura (uno stesso VBO può essere usato da più VAO senza che questi diano vita allo stesso modello, è fondamentale anche l’ordine con cui vengono scelti.).
* caricare le texture. Le texture sono delle immagini che vengono applicate a un modello per aumentarne il dettaglio. Per ogni oggetto è prevista una di queste immagini, il loro caricamento da file .jpg avviene grazie a una libreria chiamata “stb\_image.h”. Essa fornisce tutti i metodi per ottenere i dati contenuti all’interno di un file immagine e caricarli all’interno di OpenGL. Le immagini ottenute verranno trasformate in Mipmap, ossia delle immagini il cui numero di pixel diminuisce in base alla distanza dalla camera.
* caricare le shader. Le shader sono dei semplici programmi per GPU che permettono all’unità di elaborazione grafica di sapere come processare i valori ad essa inviati. Le shader che ho usato all’interno della pipeline grafica sono due e sono scritte nel linguaggio GLSL.

La prima è chiamata Vertex Shader e serve a manipolare i vertici ad essa passati a seconda della scala dell’oggetto, dell’angolo e della sua posizione nello spazio. La seconda, invece, è chiamata Fragment Shader e permette di calcolare il colore di ogni pixel, in questo caso a partire da una texture in precedenza definita. Esistono altre shader all’interno della pipeline che, non avendo né dichiarato né modificato, non verranno conteggiate lasciando spazio alle loro versioni standard vuote.

* gestire la fisica di ogni oggetto. Dato che ogni oggetto non è statico e deve poter essere mosso all’interno della finestra, sono necessari alcuni parametri fisici per regolarne il movimento rispetto ad altri oggetti, in questo caso vengono memorizzati quattro valori: la posizione nella finestra, la velocità relativa a un singolo frame, l’angolo rispetto all’asse delle ascisse e la scala dell’oggetto, ossia le dimensioni che un oggetto dovrà assumere rispetto agli altri all’interno del mondo di gioco.
* gestire le collisioni con un altro oggetto. Per verificare quando un oggetto collide con un altro ho deciso di sfruttare la tecnica chiamata AABB (Axis - Aligned Bounding Box). L’idea alla base di questa tecnica è quella di costruire delle scatole allineate agli assi cartesiani e perfettamente circoscritte ai modelli. Per verificare se la collisione tra i due oggetti è avvenuta sarà necessario controllare se per entrambi gli assi queste scatole si intersecano.
* renderizzare i vertici. Questa azione riguarda la generazione a video di un risultato grafico, a partire dalla struttura VAO, dalla texture e dalle shader.

Ogni modello viene costruito attraverso dei triangoli, questa scelta permette di costruire più velocemente gli oggetti (a differenza delle semplici linee).

* Il programma vero e proprio è la parte di progetto che, sfruttando le possibilità offerte dalla classe descritta sopra, gestisce la generazione degli oggetti e tutte le operazioni riguardanti la gestione delle stampe a video e le meccaniche relative al gioco, come ad esempio l’aggiornamento degli oggetti a partire dagli input ottenuti e le azioni da eseguire in caso di collisioni tra oggetti.

Completamento:

Gli obiettivi sono stati soddisfatti quasi completamente rispetto all’idea iniziale, la replica riproduce abbastanza fedelmente il gioco originale degli anni 80. Vi sono però delle differenze per quanto riguarda il sistema di movimento della navicella e le collisioni con gli asteroidi, i quali, se colpiti, vengono distrutti direttamente anziché suddividersi in pezzi più piccoli, senza generare alcun effetto particellare.

Questo progetto, seppur simile all’originale, può essere migliorato, oltre che negli aspetti sopra citati, anche attraverso punti descritti in precedenza all’interno del project charter. La principale proposta riguarda l’inserimento di un HUD alternativo alle stampe di variabili di gioco da terminale. Un ulteriore spazio di miglioramento è sicuramente la tecnica di verifica delle collisioni tra oggetti, la migliore in termini di qualità/prestazioni sarebbe la OBB (Oriented Bounding Box).

Conclusioni:

In conclusione l’albero di attività del progetto è stato completato nei tempi e nei termini previsti. Nonostante ci siano degli spunti di miglioramento il progetto è già una valida replica. La parte più interessante e formativa è stata sicuramente la creazione della classe oggetto che, attraverso l’implementazione delle API OpenGL, mi ha permesso di comprendere, seppur in minima parte, questa tecnologia.

Asteroids è un gioco bidimensionale degli anni 80 in cui una navicella comandata dall’utente deve distruggere e schivare tutti gli asteroidi che si presentano a schermo. Nella replica progettata è previsto che gli asteroidi distrutti non si dividano a loro volta in asteroidi più piccoli come nel gioco originale ma che, una volta colpiti, vengano distrutti definitivamente senza presentare a video particelle di nessun tipo.

La classe oggetto che ho creato permette di gestire nella sua interezza qualsiasi oggetto grafico bidimensionale, ossia un insieme di vertici connessi tra di loro per formare figure complesse, a partire dall’inizializzazione, dall’aggiornamento e dalla fisica, fino ad arrivare alla gestione delle collisioni e al rendering a video.

Questa classe rappresenta quindi un livello di astrazione rispetto alla libreria OpenGL, permette di renderizzare oggetti complessi e gestirne la fisica in modo semplice. Attraverso di essa è possibile creare diversi giochi bidimensionali; Asteroids è solo un esempio di cosa si può fare con questa classe.