Fondamenti di Computer Graphics M (8 cfu)

Lab 02 - 2D Animazione & Gameplay

Michele Righi

Settembre 2023

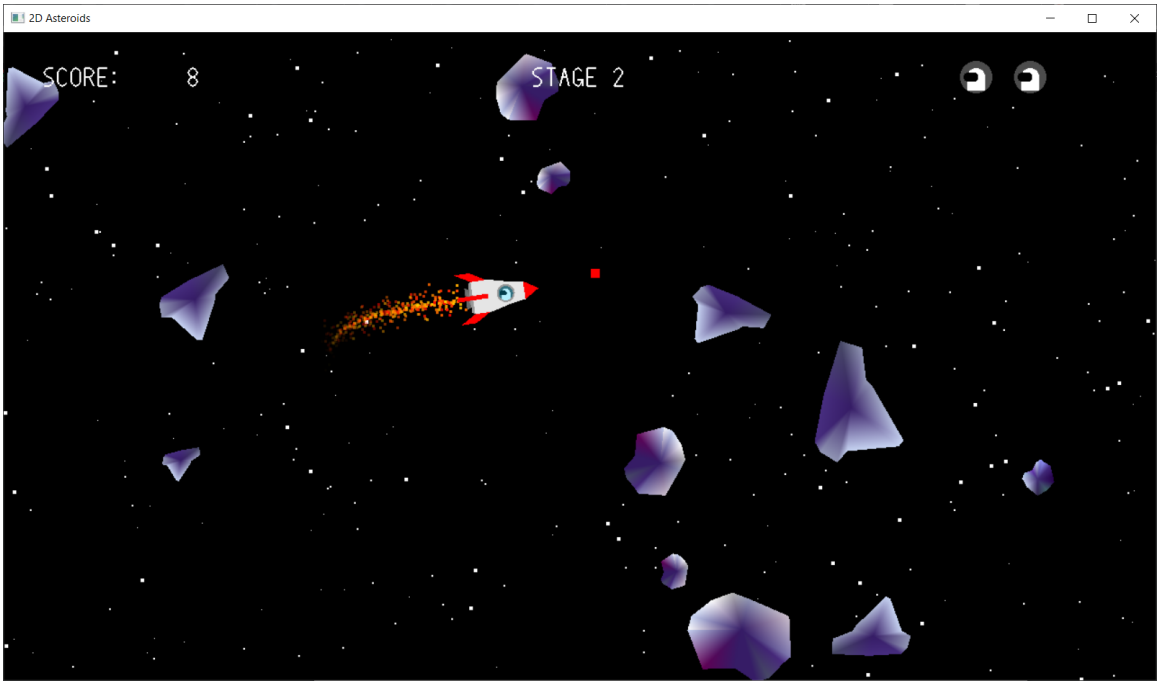


Figure 1: Demo 2D Asteroids

Traccia

Si richiede di realizzare una demo di animazione digitale 2D interattiva simile al materiale fornito. Lo studente può utilizzare il codice fornito come materiale di ispirazione e consultare il sito glslsandbox che mostra diversi esempi 2D. Tuttavia non saranno accettati elaborati che si limiteranno ad apportare banali modifiche al materiale di base.

1 Introduzione

Per questa esercitazione ho voluto realizzare una versione più moderna del famoso gioco arcade *Asteroids* di fine anni '70.

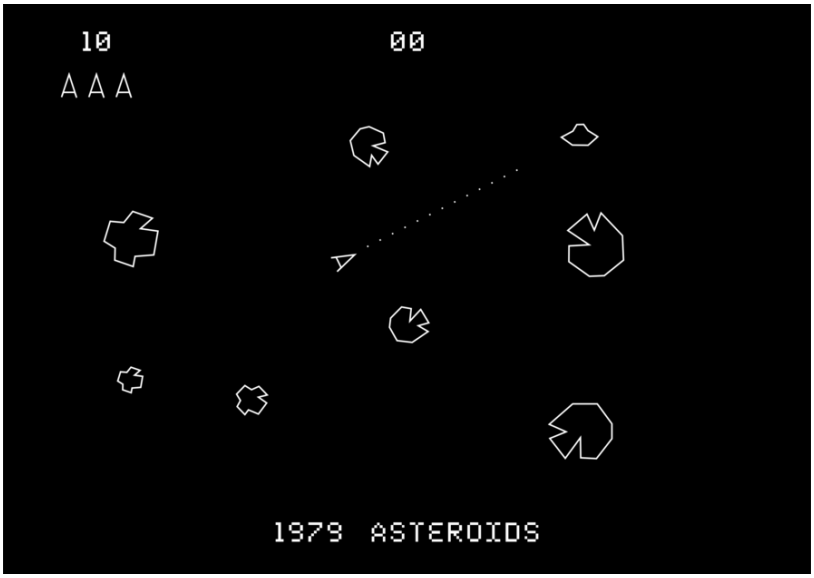


Figure 2: Asteroids from Atari, Inc. 1979

2 Struttura del Progetto

Per implementare la soluzione, ho impostato il progetto seguendo una struttura modulare, separando i componenti in file diversi:

- `02_main.cpp` contiene il main, le funzioni di inizializzazione ed il game loop, all'interno del quale vengono chiamate le funzioni di update e draw di ciascun modulo;
- `commons.h` è un file che si occupa di includere in modo ordinato tutte le librerie, di sistema e personali, necessarie al progetto (ad esempio OpenGL);
- `defs.h` contiene costanti ed enumerativi globali;
- `structs.h` espone le strutture dati comuni;
- `utils.h` espone macro e funzioni di utilità;
- i file rimanenti (`asteroids.h`, `bullets.h`, `colliders.h`, `explosions.h`, `figure.h`, `firetrail.h`, `input.h`, `spaceship.h`, `stars.h`, `text.h`, `ui.h`) contengono codice e logica dei rispettivi componenti/moduli.

2.1 Game Loop

Ho deciso di impostare il game loop basandomi sulla tecnica del **delta timing**, che permette di disaccoppiare la logica del gioco dagli FPS. In questo modo, se per qualunque motivo gli FPS dovessero variare, il comportamento degli elementi di scena e lo scorrere del tempo nel gioco non varierebbero.

```
static void update(int value)
{
    // Wait until 16ms has elapsed since last frame
    while (glutGet(GLUT_ELAPSED_TIME) - game.timeSinceStart <= 16);

    // Delta time = time elapsed from last frame (in seconds)
    game.deltaTime = (glutGet(GLUT_ELAPSED_TIME) - game.timeSinceStart) / 1000.0f;

    // Clamp delta time so that it doesn't exceed ~60 fps
    game.deltaTime = MIN(0.05f, game.deltaTime);

    // Update elapsed time (for next frame)
    game.timeSinceStart = glutGet(GLUT_ELAPSED_TIME);

    // process input & update functions

    glutTimerFunc(UPDATE_DELAY, update, 0);
}
```

Utilizzando questa tecnica, è possibile eseguire le operazioni riguardanti la logica di gioco come funzioni di delta time, che rappresenta la frazione di secondo corrispondente al frame corrente.

2.2 Stati di Gioco

Ho suddiviso il gioco in stati per consentire la navigazione tra le varie schermate e ottimizzare l'esecuzione, aggiornando solo i moduli visibili/necessari: `GAME_MENU`, `GAME_MENU_CONTROLS`, `GAME_RUNNING`, `GAME_PAUSED`, `GAME_STAGE_COMPLETED`, `GAME_NEXT_STAGE_STARTING`, `GAME_OVER`.

2.3 Input e Registrazione Comandi

Per poter registrare ed utilizzare l'input di gioco dell'utente, ho creato una piccola libreria (file `input.h`) che espone una struttura dati `Input` e delle funzioni per registrare gli eventi. Queste ultime vengono registrate tramite le funzioni di GLUT `glutKeyboardFun()` e `glutKeyboardUpFun()`. Quando l'utente preme o rilascia un tasto della tastiera, la chiave relativa nella struttura Input viene aggiornata rispettivamente con un 1 o uno 0. In questo modo è possibile ottenere un input continuo e in tempo reale.

Successivamente, i moduli che devono accedere all'input, ad esempio la navicella per muoversi o sparare, possono semplicemente controllare se il tasto di interesse è premuto nella struttura Input.

```
void inputSpaceship()
{
    if (input.keyboard.keys['w'])
    {
        // move forward
    }
}
```

3 Gameplay

Il gameplay consiste nel controllare una navicella spaziale, muovendola e sparando proiettili. Nella scena sono presenti degli asteroidi che si muovono e hanno 3 dimensioni (*grande*, *medio* e *piccolo*): quando un asteroide viene colpito da un proiettile, si divide in due di dimensioni minori, tranne nel caso in cui siano già di tipo piccolo.

L'obiettivo del gioco è distruggere tutti gli asteroidi nella scena senza perdere tutte e tre le vite. Avanzando di livello, il numero di asteroidi aumenta e, di conseguenza, anche la difficoltà. Il gioco termina quando l'utente perde tutte le vite.

3.1 Menu

All'avvio dell'applicazione, al giocatore viene presentata la schermata di menu mostrata in Fig. 3. Da questa schermata è possibile visualizzare la lista dei comandi col tasto 'H', oppure iniziare una partita premendo la barra spaziatrice.

Figure 3: Schermata del menu

3.2 Comandi

La schermata dei comandi mostra all'utente le associazioni dei tasti e la loro funzione:

- `H` (nel menu), mostra/nasconde i comandi;
- `W`, muove la navicella in avanti;
- `A/D`, ruota la navicella in senso antiorario/orario;
- `Space`, spara un proiettile;
- `C`, mostra/nasconde i collider degli oggetti in scena;
- `L`, mostra/nasconde il rendering delle linee (invece dei triangoli);
- `B`, rende grande/piccola la navicella (per apprezzarne meglio i dettagli);
- `O`, apre/chiude l'oblò della navicella;
- `P`, mette in pausa il gioco;
- `Escape`, torna al menu;

3.3 Gioco

Una volta premuta la barra spaziatrice, inizia il gioco vero e proprio: l'utente assume il controllo della navicella e può muoversi liberamente, sparando agli asteroidi e accumulando punti. All'inizio di ciascuna stage, o quando la navicella viene distrutta, il giocatore ottiene uno scudo temporaneo, che gli garantisce 3 secondi di invulnerabilità: se durante questo periodo colpisce un asteroide, la navicella non viene distrutta.

Ogni asteroide distrutto assegna al giocatore un punteggio che varia a seconda della grandezza dell'asteroide (i più piccoli danno più punti).

3.4 Stage Completato/Game Over

Alla fine di ogni stage, o quando l'utente perde tutte e 3 le vite, il punteggio accumulato durante quel livello viene sommato al totale.

(a) Schermata di stage completato

(b) Schermata di game over

4 Rendering

Per renderizzare gli elementi viene sfruttata una semplice shader *pass-through*, caricata tramite la libreria `ShaderMaker.h` fornita dalla professoressa.

4.1 Oggetti di Scena

4.1.1 Sfondo

Lo sfondo è costruito partendo da una base nera, che rappresenta il vuoto dello spazio, su cui viene aggiunto un insieme di punti bianchi, che rappresentano le stelle. Le stelle sono raggruppate in 3 insiemi, in base alla dimensione/lontananza, per un totale di 300 stelle.

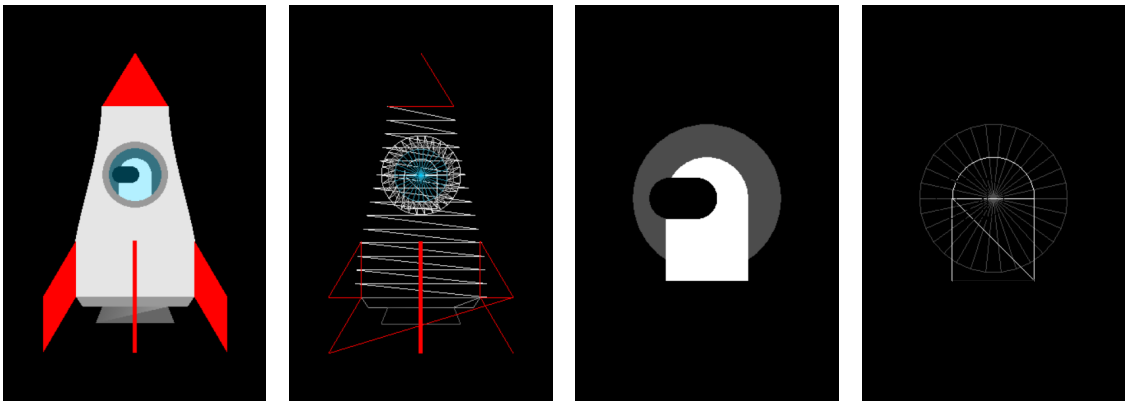
4.1.2 Navicella

La navicella è l'oggetto più complesso, costituito da un insieme numeroso di parti (Fig. 5):

- punta, costituita da un triangolo;

- pinne laterali, costituite ciascuna da un rombo (due triangoli);
- pinna centrale, una linea;
- scafo, disegnato tramite due funzioni periodiche (coseno) per disegnare le curve dei lati, riempite dai triangoli;
- propulsore, costituito da due trapezi isosceli, di colori differenti per rendere l'effetto della lucentezza del metallo;
- cabina dell'astronauta, costituita da un cerchio pieno;
- astronauta:
  - visiera, costituita da due semicerchi ed un rettangolo (due triangoli);
  - tuta, costituita da un semicerchio nella parte superiore, ed un quadrato per la parte inferiore (due triangoli);

  NB: l'astronauta rimane sempre rivolto nella stessa direzione (per farlo, la matrice di modellazione delle sue figure viene ruotata nel senso opposto a quella della navicella);
- bordo metallico dell'oblò, costituito da una corona (vedere la funzione `buildHollowCircle()` in `figure.cpp`);
- vetro dell'oblò, costituito da un cerchio pieno trasparente (utilizzo modalità `GL_BLEND` e canale alpha di RGBA);

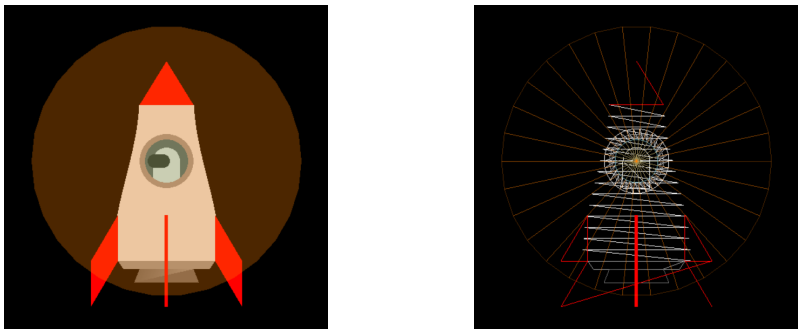


(a) Triangoli navicella (b) Linee navicella (c) Triangoli astronauta (d) Linee astronauta

Figure 5: Rendering navicella

4.1.3 Scudo

Lo scudo della navicella è costituito da un cerchio trasparente, che copre tutta l'area della navicella

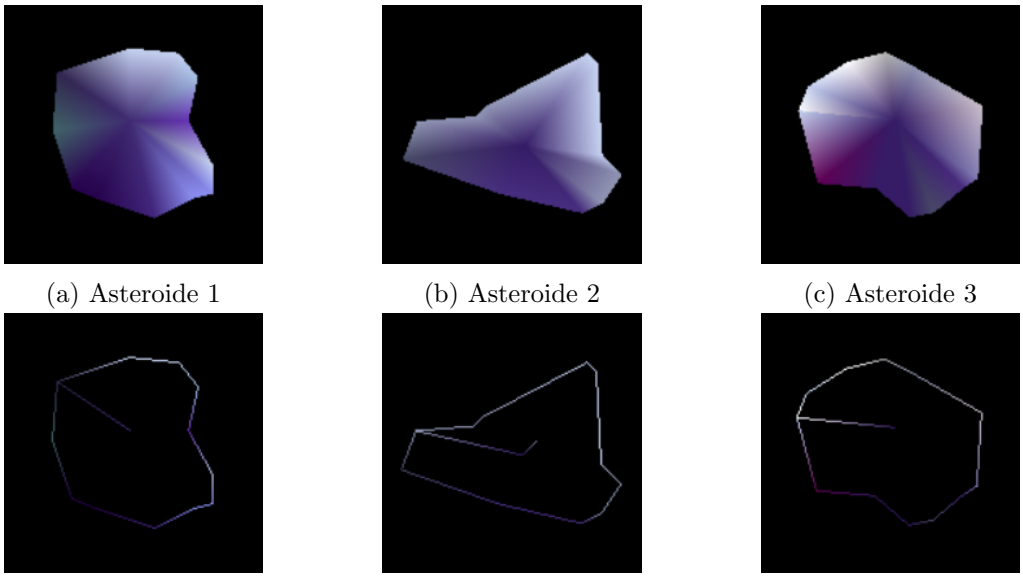


(a) Triangoli scudo (b) Linee scudo

Figure 6: Rendering scudo

4.1.4 Asteroidi

Ho disegnato gli asteroidi inserendo le coordinate dei vertici manualmente, prendendo spunto, per la forma ed i colori, da un'immagine trovata su Google. Ne ho fatti di 3 tipologie differenti, come visibile in Fig. 7.



(a) Asteroide 1 (b) Asteroide 2 (c) Asteroide 3

(d) Linee asteroide 1 (e) Linee asteroide 2 (f) Linee asteroide 3

Figure 7: Rendering asteroidi

4.1.5 Vite

Le vite sono disegnate utilizzando lo sfondo della cabina della navicella e la grafica dell'astronauta (vedi Fig. 5c).

4.1.6 Proiettili

I proiettili sono costituiti da un singolo punto rosso, di dimensione `10.0f`.

4.1.7 Testo

Per implementare il testo ho creato una libreria `'text.h'` che permette di creare scritte dato un buffer di caratteri. Per realizzarne le graffiche ho utilizzato punti e linee, e ho scritto i vertici manualmente (vedi Fig. 8). I caratteri non implementati vengono sostituiti da un *hyphen* `'-'`.

Figure 8: Testo

5 Fisica

Tutti gli elementi del gameplay vengono wrappati all'interno della scena: quando un oggetto esce da un lato della viewport, ricompare sul lato opposto.

5.1 Stelle

Per rendere l'esperienza di gioco più immersiva e naturale, ho sfruttato l'**effetto parallasse**: al muoversi della navicella, le stelle più vicine (grandi) si spostano più velocemente di quelle lontane.

5.2 Navicella

Il movimento della navicella è vincolato dal propulsore: la navicella si può muovere solo in avanti o ruotare. Per rendere il movimento più realistico, ho inserito una decelerazione della velocità, quando l'utente non preme i tasti di movimento/rotazione.

```
1 void updateSpaceship(float deltaTime)
2 {
```

```
    if (spaceship.angularSpeed != 0.0f)
    {
        spaceship.heading += spaceship.angularSpeed * deltaTime;
        if (spaceship.heading > 2 * PI)
            spaceship.heading -= 2 * PI;
        if (spaceship.heading < 0.0f)
            spaceship.heading += 2 * PI;
    }
    if (spaceship.forwardSpeed != 0.0f)
    {
        spaceship.pos.x += cos(spaceship.heading) * spaceship.forwardSpeed * deltaTime;
        spaceship.pos.y += sin(spaceship.heading) * spaceship.forwardSpeed * deltaTime;
        // [...]
    }
}
```

5.3 Asteroidi

Gli asteroidi si muovono seguendo un moto rettilineo uniforme, con direzione e verso casuali.

5.4 Proiettili

Come gli asteroidi si muovono seguendo un moto rettilineo uniforme, ma assumono la direzione e il verso della navicella quando vengono sparati.

5.5 Sistema di Collisioni

Per il sistema di collisioni ho utilizzato dei collider circolari, rappresentati dalla struttura `CircleCollider`, che contiene in particolare un centro e un raggio. Gli oggetti che possono collidere (navicella, asteroidi, proiettili) contengono questa struttura, che viene utilizzata controllando se la distanza tra i due centri è minore o uguale alla somma dei raggi: in questo caso si avrà una collisione.

Ho anche inserito un controllo per abilitare/disabilitare il rendering dei collider, come visibile in Fig. 9.

```
bool isCollidingCircle(CircleCollider collider1, CircleCollider collider2)
{
    return distance(collider1.pos, collider2.pos) < collider1.radius + collider2.radius;
}
```

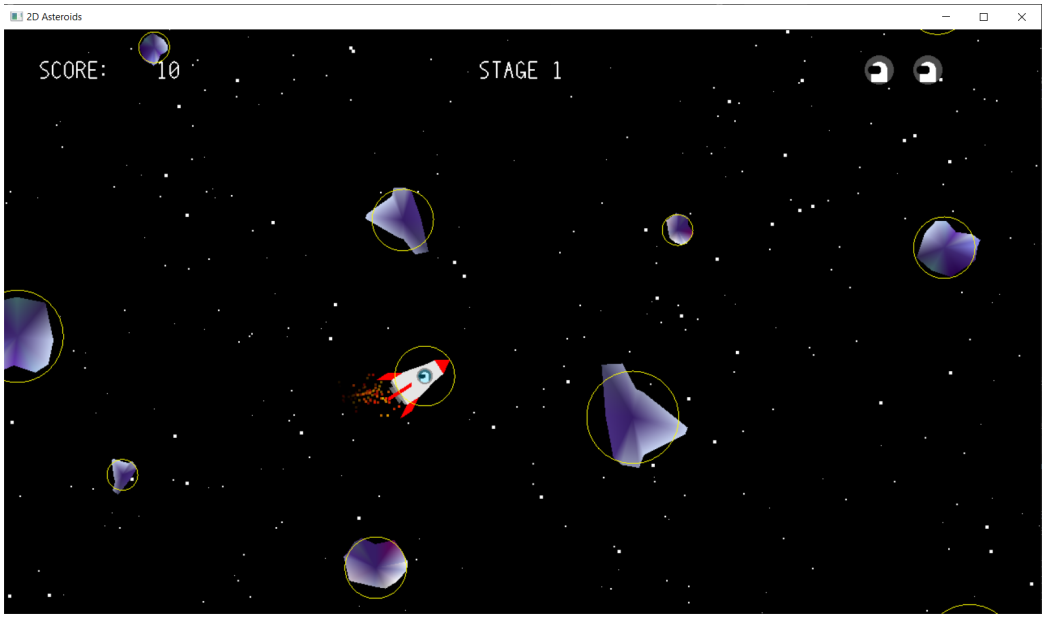
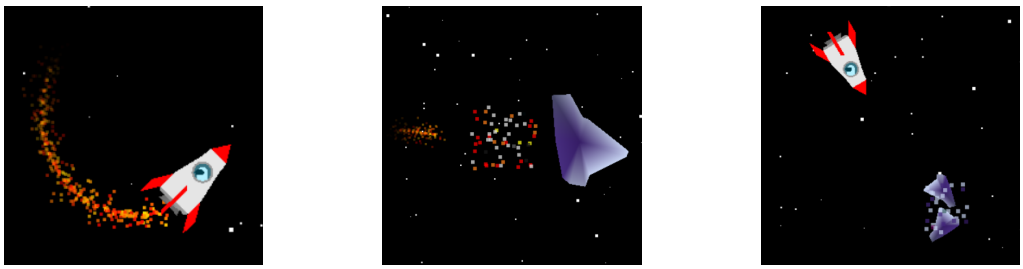


Figure 9: Collider visibili (tasto `'c'`)

6 Animazioni

6.1 Particellari



(a) Scia navicella (b) Esplosione navicella (c) Esplosione asteroide

Figure 10: Particellari

6.1.1 Firetrail

Quando la navicella si muove in avanti, genera una scia di fuoco (Fig. 10a), realizzata tramite un sistema particellare: le particelle vengono generate in punti randomici all'interno di un'area circolare, con centro spostato rispetto a quello della navicella (di modo che si trovi sotto il propulsore). Inoltre, vengono generate solo se la navicella ha velocità superiore ad una certa soglia, e il numero dipende dalla velocità e dalle sue dimensioni.

```
// Spawn firetrail only if the speed is greater than a threshold
if (spaceship.forwardSpeed > 10.0f)
{
    float xSpawnCenter, ySpawnCenter;
    xSpawnCenter = spaceship.pos.x +
        (spaceship.radius + spaceship.scale * 1.25f) * cos(spaceship.heading + PI);
    ySpawnCenter = spaceship.pos.y +
        (spaceship.radius + spaceship.scale * 1.25f) * sin(spaceship.heading + PI);

    spawnFiretrailParticles(
        { xSpawnCenter, ySpawnCenter, 0.0f },
        spaceship.forwardSpeed,
        spaceship.heading - PI,
        20.0f * (spaceship.forwardSpeed / SPACESHIP_MAX_FORWARD_SPEED) *
            (spaceship.radius / spaceship.originalRadius),
        5.0f
    );
}
```

6.1.2 Esplosioni

Navicella Quando la navicella priva di scudo viene colpita da un asteroide, esplode generando delle particelle: le particelle hanno colori che rappresentano i detriti della navicella, fuoco e fumo (vedi Fig. 10b).

Asteroidi Quando un asteroide viene colpito da un proiettile, esplode generando delle particelle: le particelle hanno colori simili a quelli dell'asteroide (vedi Fig. 10c).

6.2 Testo e UI

Per migliorare ulteriormente l'esperienza dell'utente, ho deciso di aggiungere qualche animazione al testo, in particolare:

- testo lampeggiante (presente nel menu e durante la schermata di Game Over, per indicare all'utente quale tasto premere per proseguire);
- testo che incrementa/decrementa velocemente (presente durante il gioco, quando lo scudo è attivo, e nelle schermate di Stage Completato e Game Over, per aggiornare il punteggio).