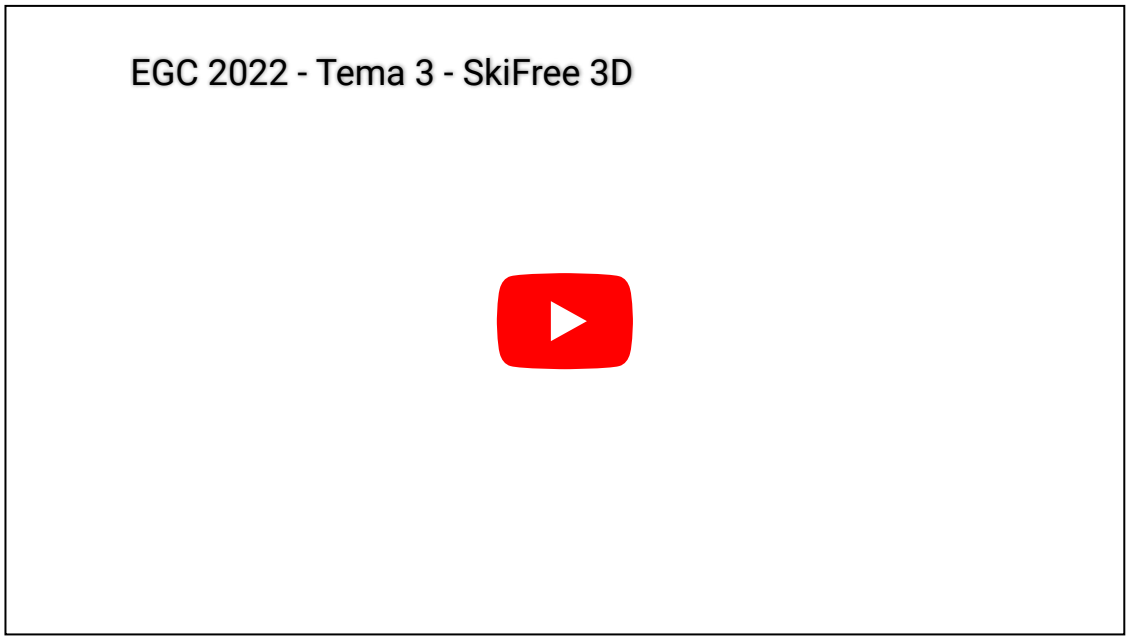
Tema 3 - SkiFree 3D

- **Responsabili:** Andrei Lăpușteanu, Anca Cristea, Robert Caragicu
- **Lansare:** 18 decembrie 2022
- **Termen de predare:** 15 ianuarie 2023, ora 23:59
- **Regulament:** Regulament General
- **Notă: Orice informație ce nu a fost acoperită în acest document este la latitudinea voastră!**

În cadrul temei 3 veți avea de implementat un joc 3D în care controlați un schior pe o pantă ce trebuie să adune cadouri și să evite diverse obstacole. Jocul este de tip endless, obiectivul jucătorului fiind obținerea a cât mai multe puncte.

Pentru inspirație, puteți testa jocul original aici: https://classicreload.com/win3x-skifree.html [https://classicreload.com/win3x-skifree.html]

O posibilă implementare a temei este prezentată în următorul video demo.



Construcția scenei

Scena 3D va fi formată din:

- Terenul jocului (panta de ski)
- Jucător
- Obstacole și elemente colectabile
- Surse de lumină atașate obiectelor 3D

Este permisă importarea de mesh-uri create în aplicații de modelare 3D.

Terenul jocului

Personajul se deplasează în lume (schiază pe pantă) și astfel își modifică coordonatele globale în lume. Obstacolele și cadourile rămân la poziții fixe în scenă.

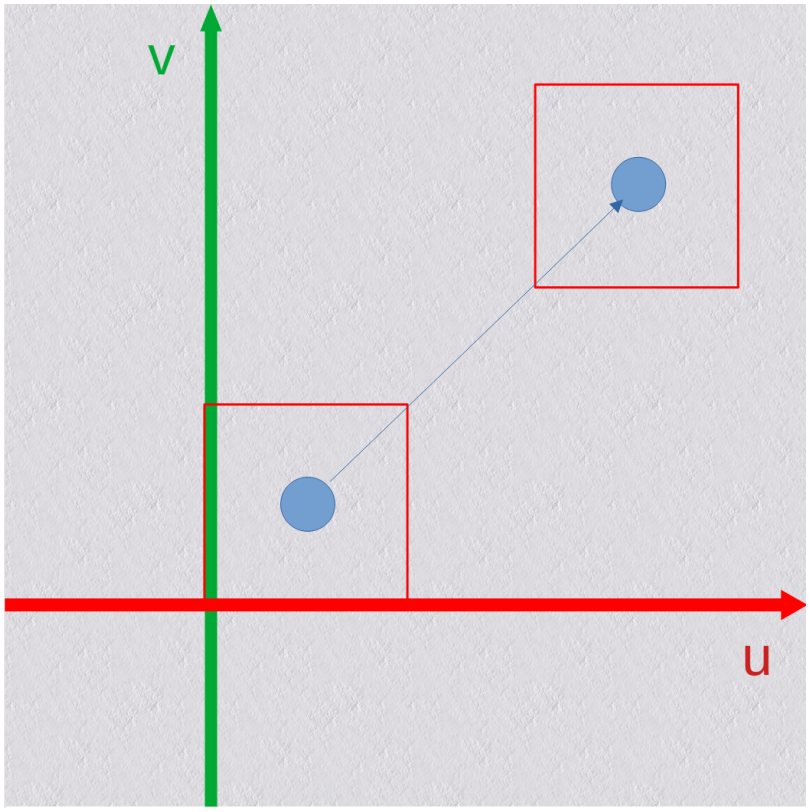
Terenul va fi realizat folosind un pătrat (quad) mare pe care va fi aplicată o textură ce se repetă.

Pentru a nu fi nevoiți să generăm pătrate noi în timp ce se deplasează jucătorul pe pantă, pătratul va fi desenat în poziția jucătorului. Astfel, terenul stă lipit de personaj.

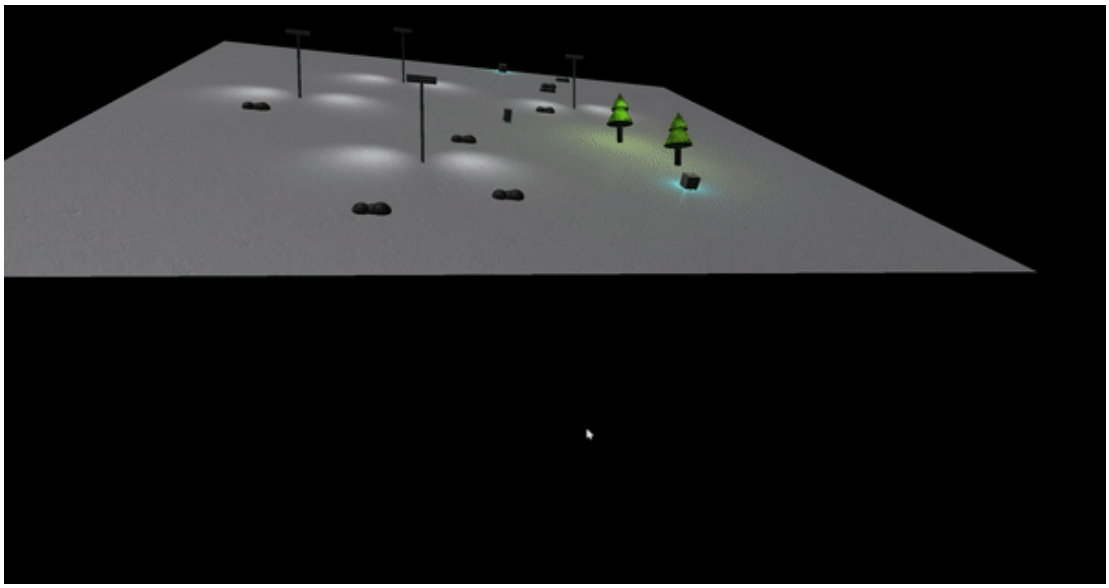
Iluzia de mișcare a terenului sub personaj va fi dată de modificarea coordonatelor textură a terenului. Această modificare constă într-un deplasament aplicat tuturor UV-urilor mesh-ului terenului. Pentru a nu modifica mesh-ul terenului se trimite la shader-ul ce desenează terenul o uniformă de tipul vec2 ce reprezintă deplasamentul

coordonatelor textură. Este important ca viteza cu care se modifică UV-urile să fie la fel cu viteza personajului, pentru ca un pixel din imagine să rămână la poziție fixă în lume.

Imaginea de mai jos reprezintă imaginea terenului repetată. Sistemul de coordonate este cel de textură. Inițial planul este dreptunghiul roșu cu colțul stânga jos în origine iar locul unde ar fi desenat personajul este reprezentat cu un cerc. După ce personajul s-a mișcat într-o altă poziție, observăm noile coordonate de textură calculate ale planului mutat în poziția personajului.



În următorul GIF este prezentată scena jocului dintr-un punct fix. Se observă generarea și ștergerea obstacolelor, faptul că terenul rămâne cu centrul la poziția jucătorului, precum și modalitatea prin care sunt modificate coordonatele de textură ale pantei pentru a reda iluzia de mișcare.



Precum se observă și în animația de mai sus, terenul trebuie să aibă un unghi de înclinare, obiectele se vor genera pe această pantă înclinată. În demo am setat un unghi de înclinare de 30°.

Jucătorul

Jucătorul este format din cel puțin 3 paralelipipede (corp și cele două schiuri) – configurația se poate crea folosind primitivele disponibile în framework. Mesh-urile care alcătuiesc jucătorul trebuie create astfel încât obiectul de tip jucător să poată fi translatat și rotit ca o singură unitate (mesh-urile să fie “ancorate” unele de celelalte). Mesh-urile ce alcătuiesc jucătorul nu trebuie texturate (în mod obligatoriu).

Obstacole

În timpul jocului, player-ul poate întâlni pe pantă 3 tipuri de obstacole, anume:

- **Copăcei**, formați din cel puțin 2 mesh-uri. Exemplul din demo se utilizează de primitivele de tip cub si con pentru a defini obiectul. Este necesară aplicarea unei texturi pe fiecare primitivă din obiect

Primitiva de tip con nu se regăsește în framework. Dacă doriți s-o utilizați, aceasta se poate descărca de aici cone.zip

- **Pietre**, formate din cel puțin 2 mesh-uri. Exemplul din demo se utilizează de 3 primitive translatate și scalate de tip sferă. Este necesară aplicarea unei texturi pe fiecare primitivă din obiect
- **Stâlpi de iluminat**, alcătuiți din 2 mesh-uri. Exemplul din demo se utilizează de 2 primitive translatate și scalate de tip cub. Este necesară aplicarea unei texturi pe fiecare primitivă din obiect

Collectibles (cadouri)

În demo, obiectul de tip **collectible** a fost imaginat sub forma unui **cadou** pe care jucătorul îl poate colecta. Obiectul trebuie alcătuit dintr-un singur mesh de tip cub. Este obligatorie texturarea acestui obiect.

Următoarea imagine prezintă tipurile de obiecte implementate în demo. De la stânga la dreapta se poate observa câte un exemplu de: jucător, copac, stâlp de iluminat, piatră și cadou.

Nu sunt aplicate calcule de iluminare în imaginea de mai sus.

Controlul jucătorului (input)

Schiorul se va deplasa încontinuu pe pantă cu o viteză constantă, fără a fi necesară apăsarea vreunei taste, singura formă impusă de control va fi cea a direcției de deplasare a acestuia. Direcția de deplasare va fi controlată folosind poziția cursorului în viewport – este suficientă folosirea coordonatei X a poziției cursorului.

Nu este necesar ca jucătorul să urmărească îndeaproape direcția dictată de către cursor. Este suficient ca acesta să se deplaseze stânga/dreapta în niște limite impuse. Spre exemplu, în demo, direcția de deplasare a jucătorului poate fi controlată într-un interval de ±45°.

Următorul GIF ilustrează controlul direcției de deplasare a jucătorului folosind poziția cursorului pe viewport.

Nu sunt aplicate calcule de iluminare în GIF-ul de mai sus.

Poziționarea camerei

Camera va fi de tip perspectivă și va urmări jucătorul. Nu există o configurație impusă asupra poziționării acesteia atât timp cât camera implementată este de tip third person. În demo a fost aleasă o configurație a camerei similară jocului clasic – vă recomandăm această variantă – o perspectivă de tip third person din spatele jucătorului ar fi cauzat probleme deoarece obstacolele generate la run-time s-ar fi instanțiat în mod vizibil în fața acestuia (efect de “pop-in”).

Generarea de obstacole și elemente colectabile

Obiectele 3D ce reprezintă obstacole și elemente colectabile trebuie generate dinamic în timpul rulării jocului. Condiții impuse:

- Acestea trebuie instanțiate și șterse în afara viewport-ului jocului
- Algoritmul pentru generarea obstacolelor trebuie să instanțieze obiectele de tip obstacol și collectible menționate anterior (copăcei, pietre, stâlpi de iluminat, cadouri). Nu este impusă o metodă de alegere a tipului de obiect ce urmează a fi instanțiat, în demo obiectul este ales aleatoriu din lista de posibile obiecte instanțiabile
- Algoritmul pentru generarea obstacolelor trebuie să dispună obiectele 3D pe pantă astfel încât jucătorul să fie nevoit să le ocolească (poziție de instanțiere randomizată în anumite limite)
- Algoritmul pentru generarea obstacolelor trebuie să ofere rezultate satisfăcătoare independent de direcția de deplasare a jucătorului

Coliziuni

Veți avea de implementat coliziuni între jucător și obstacole și între jucător și cadouri.

Mai multe informații despre ce reprezintă coliziunile si cum se pot implementa:

- https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection [https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection]
- https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection [https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection]
- https://www.youtube.com/watch?v=aTbw71EpamY [https://www.youtube.com/watch?v=aTbw71EpamY]
- https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection [https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection]

Mențiune: În cadrul acestei teme sunt acceptate coliziuni simple de tip sferă-sferă, sferă-AABB și AABB-AABB (AABB = Axis-Aligned Bounding Box (en.), Paralelipiped dreptunghic aliniat cu axele (ro.))

În cadrul implementării din demo, fiecare obiect a fost încadrat într-un obiect de coliziune AABB (în 3D). În timpul jocului, în fiecare frame se testează coliziunea dintre jucător și toate celelalte obstacole și colectabile instanțiate.

Iluminare

Iluminarea scenei se va implementa folosind 3 tipuri de surse de lumină, anume:

- O lumină **direcțională** va ilumina toate obiectele din scenă cu aceeași intensitate (ex. Soare). Nu este neapărat nevoie de reprezentarea sursei de lumină print-un obiect, însă se va verifica existența acesteia. Specific luminii de tip direcțional este faptul că vectorul luminii incidente (L) nu depinde de poziția luminii (precum în cazul luminilor de tip point și spot). Așadar, pentru fiecare fragment, iluminarea va fi calculată folosind același vector L (corespunzător direcției luminii). De exemplu, se poate alege ca lumina direcțională să aibă direcția sursei de lumină orientată direct în joc (perpendicular pe XoZ), așadar, direcția s-ar seta astfel:

```
glm::vec3(0.f, -1.f, 0.f);
```

- Stâlpii de iluminat vor emite două lumini de tip **spotlight** care vor porni din “becurile” obiectului, respectiv din marginile paralelipiedului superior. Lumina trebuie să se observe pe suprafața zăpezii. De asemenea, orice obiect care trece pe sub stâlpul de iluminat va fi iluminat de spot.
- Elementele colectabile și copacii vor avea o sursă de lumină **punctiformă**, de diverse culori, poziționată în interiorul sau apropierea obiectului. Componentele unei surse de lumină pot avea aceeași culoare aleasă aleatoriu sau dintr-o gamă mai largă de culori, însă NU toate obiectele vor emite aceeași culoare. În plus, fiecare sursă de lumină punctiformă va avea o arie de acoperire pentru lumina emisă, fapt controlat print-un factor de atenuare constrâns de distanța dintre obiectul luminat și sursa de lumină.

Intensitatea luminilor de tip point și spot **trebuie** atenuată pe baza distanței (dintre sursă și punctul în care se calculează iluminarea). Lumina de tip direcțional **nu** trebuie atenuată.

Precum este prezentat în demo, implementarea voastră trebuie să suporte **randarea mai multor surse de lumină în același frame!**

Calcul și afișare scor

Scopul jocului este obținerea unui punctaj cât mai mare. Acesta trebuie calculat și afișat la finalul jocului în consolă. Scorul trebuie să corespundă numărului de cadouri colectate.

Gameplay loop

În continuare sunt prezentate câteva detalii referitoare la desfășurarea jocului care nu au fost acoperite în secțiunile anterioare:

- După pornire, jocul este în desfășurare atât timp cât nu a fost detectată o coliziune cu un obstacol
- În cazul în care s-a detectat o coliziune între jucător și obstacol, obiectul de tip jucător este oprit din deplasare iar scorul este afișat în consolă. Este necesară implementarea unui funcționalități de resetare a jocului (de exemplu, la apăsarea unui taste)
- În cazul în care s-a detectat o coliziune între jucător și obiect colectabil, scorul este incrementat iar obiectul colectabil este șters de pe ecran

Funcționalități obligatorii (150 puncte)

- Texturare + asamblare obiecte 3D (55p)
    - Desenare teren înclinat (15p)
        - Desenare pătrat (quad) teren (5p)
        - Modificare UV-uri pentru a reda efectul deplasării (10p)
    - Cadouri (8p)

        - Copaci (8p)
        - Stâlpi de iluminat (8p)
        - Jucător (8p)
        - Pietre (8p)
    - Iluminare (40p)
        - Iluminare direcțională (10p)
        - Iluminare de tip spot atașată stâlpilor de iluminat (15p)
        - Iluminare punctiformă atașată copacilor si cadourilor (15p)
    - Elemente de gameplay (55p)
        - Deplasare și orientare jucător (10p)
        - Coliziune jucător-obstacole (10p)
        - Coliziune jucător-cadouri (7.5p)
        - Apariție și ștergere obstacole în afara viewport-ului (10p)
        - Apariție și ștergere cadouri în afara viewport-ului (7.5p)
        - Calcul și afișare scor (5p)
        - Resetare joc (5p)

Exemple de funcționalități bonus

- Personajul să încetinească după coliziunea cu un obstacol, obstacolul să dispară și să îi scadă din viață jucătorului + implementarea unui sistem de vieți precum și afișarea numărului de vieți rămase pe ecran (sub forma unui UI)
- Mai multe tipuri de obstacole de același fel (mai mulți copaci sau mai multe pietre)
- Cadouri cu scor diferit și care oferă jucătorului diferite abilități (să sară, să schieze mai repede, etc.)
- Pomii să aibă mai multe surse de lumină (ca un brad împodobit), culorile luminilor să varieze în timp
- Coroana copacilor să sufere o deformare în timp prin intermediul vertex shader-ului (efect bătaie vânt)

Orice funcționalitate suplimentară implementată (care nu este inclusă în cerințele obligatorii) poate fi considerată ca punctaj bonus dacă este suficient de complexă. Funcționalitățile bonus se iau în considerare doar dacă funcționalitățile obligatorii au fost realizate.

Întrebări și răspunsuri

Pentru întrebări vom folosi forumurile de pe Moodle. Orice nu este menționat în temă este la latitudinea fiecărui student!

Notare

Baremul este orientativ. Fiecare asistent are o anumită libertate în evaluarea temelor (de exemplu, să dea punctaj parțial pentru implementarea incompletă a unei funcționalități sau să scadă pentru hard coding). Același lucru este valabil atât pentru funcționalitățile obligatorii, cât și pentru bonusuri.

Tema trebuie încărcată pe Moodle. Pentru a fi punctată, tema trebuie prezentată la laborator. Vor exista laboratoare speciale de prezentare a temelor (care vor fi anunțate).

Indicații suplimentare

Tema va fi implementată în OpenGL și C++. Este indicat să folosiți framework-ul și Visual Studio.

Pentru implementarea temei, în folderul **src/lab_m1** puteți crea un nou folder, de exemplu **Tema3**, cu fișierele **Tema3.cpp** și **Tema3.h** (pentru implementare POO, este indicat să aveți și alte fișiere). Pentru a vedea fișierele nou create în Visual Studio în Solution Explorer, apăsați click dreapta pe filtrul **lab_m1** și selectați **Add→New Filter**. După ce creați un nou filtru, de exemplu **Tema3**, dați click dreapta și selectați **Add→Existing Item**. Astfel adăugați toate fișierele din folderul nou creat. În fișierul **lab_list.h** trebuie adăugată și calea către header-ul temei. De exemplu: **#include “lab_m1/Tema3/Tema3.h”**

Arhivarea Proiectului

- În mod normal arhiva trebuie să conțină toate resursele necesare compilării și rulării
- Înainte de a face arhiva asigurați-vă că ați curățat proiectul Visual Studio:
  - Click dreapta pe proiect în **Solution Explorer → Clean Solution**
  - Ștergeți folderul **/build/.vs** (dacă nu îl vedeți, **este posibil să fie ascuns**)
- În cazul în care arhiva tot depășește limita de 50MB (nu ar trebui), puteți să ștergeți și folderul **/deps** sau **/assets** întrucât se pot adăuga la testare. Nu este recomandat să faceți acest lucru întrucât îngreunează mult testarea în cazul în care versiunea curentă a bibliotecilor/resurselor diferă de versiunea utilizată la momentul scrierii temei.

egc/teme/2022/03.txt · Last modified: 2022/12/20 12:14 by andrei.lapusteanu