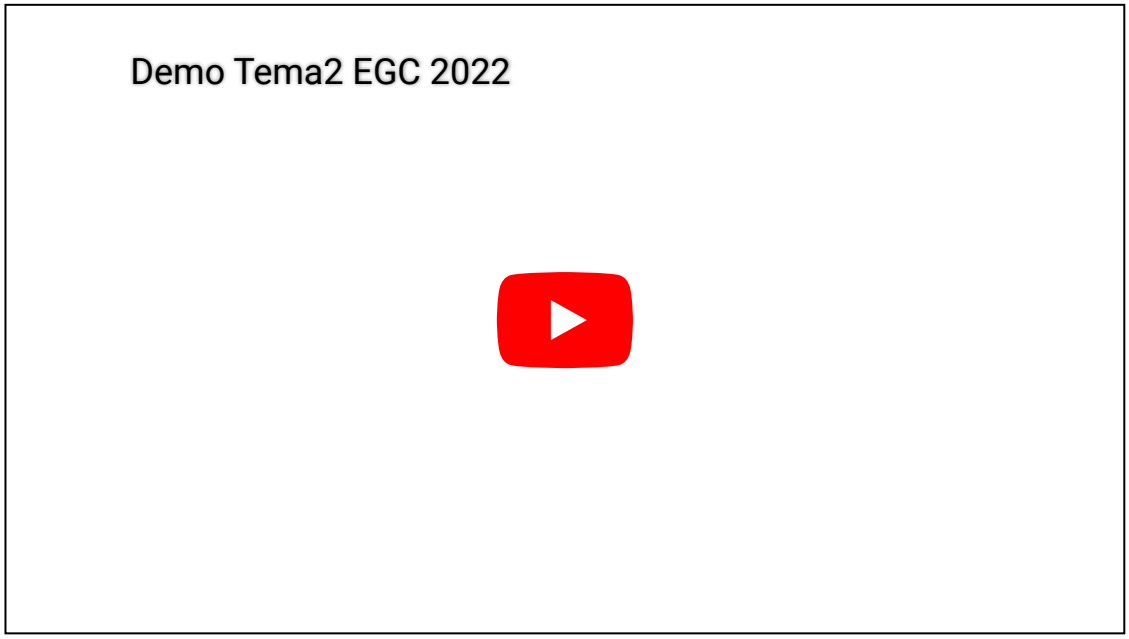
Tema 2 - Car Race

- **Responsabili:** Vlad Drăghici, Cristi Lambru, Bogdan Vasile
- **Lansare:** 21 noiembrie 2022
- **Termen de predare:** 16 decembrie 2022, ora 23:59
- **Regulament:** Regulament General
- **Notă: Orice informație ce nu a fost acoperită în acest document este la latitudinea voastră!**

În cadrul temei 2, veți avea de implementat un joc de curse de mașini parțial. Va trebui să creați pista cursei, controlul mașinii pe pistă, un set de obstacole dinamice ce vor avea un traseu predefinit pe pistă și vor substitui mașinile adversare cu care se realizează întrecerea și un efect de curbură a pistei din poziția mașinii.

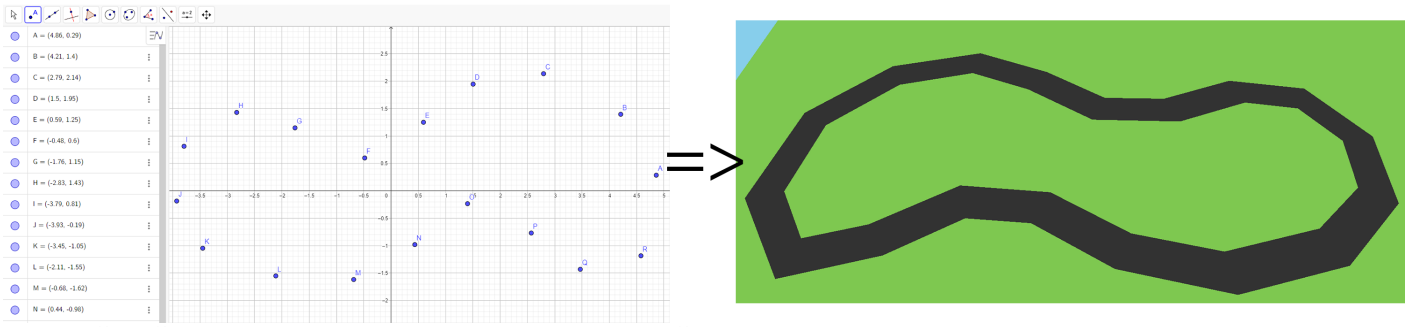
Puteți viziona un demo construit pe baza frameworkului de laborator care acoperă cerințele detaliate mai jos.



Curbura terenului este doar un **efect vizual**. Toate calculele, împreună cu generarea geometriei tuturor obiectelor din scenă se realizează într-un spațiu în care nu se ține cont de curbură.

Pista cursei

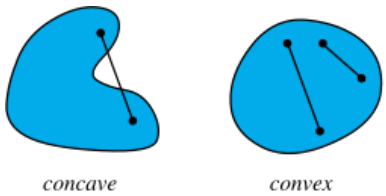
Pentru a genera o pistă pe care să meargă mașina, este nevoie inițial de o mulțime de puncte care să o definească. Mulțimea este formată din puncte în planul XOZ (2D) și poate fi obținută folosind orice tool (chiar și foaia si pixul). Pentru exemplul următor s-a folosit Geogebra (https://www.geogebra.org/classic [https://www.geogebra.org/classic]).

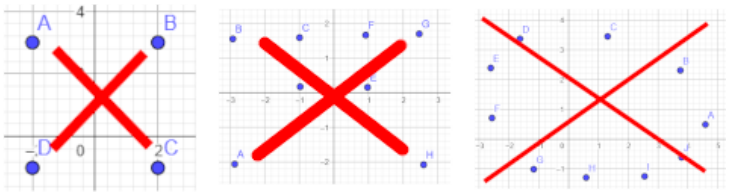


Menționăm că pista de curse din demo este una complexă. Pentru simplitate, în toate explicațiile ce vor urma se va folosi o pistă generată de un număr mic de puncte.

Pista trebuie să respecte 2 condiții:

1. Să nu existe niciun unghi drept între oricare 2 segmente consecutive
2. Să fie un poligon concav





De exemplu, cele 3 mulțimi de puncte din partea dreaptă de mai sus nu sunt permise.

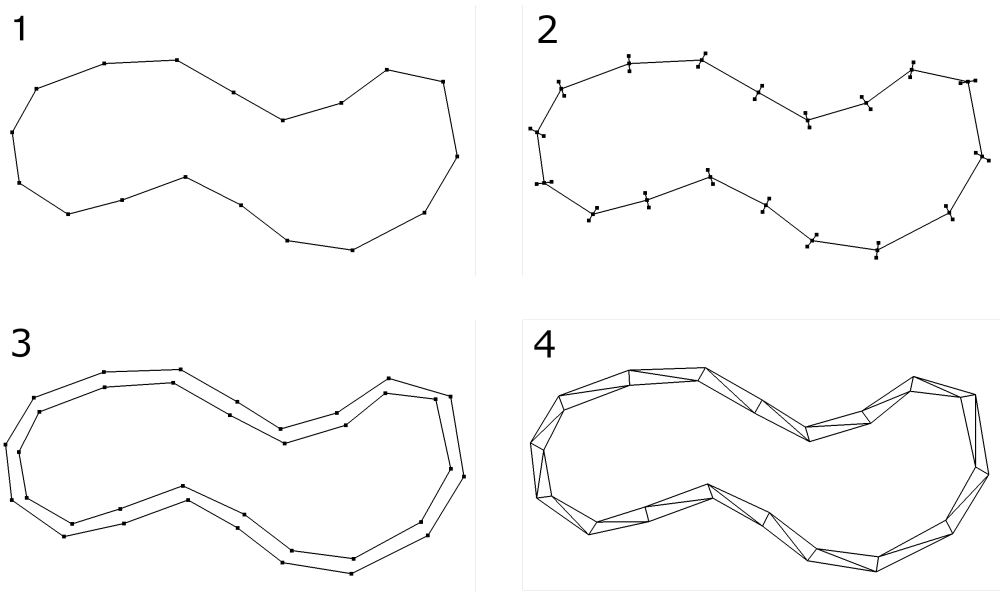
Generare geometrie pistă

Geometria pistei de curse trebuie generată prin asamblare din cod. Nu se poate crea geometria într-un software extern și importa în aplicație.

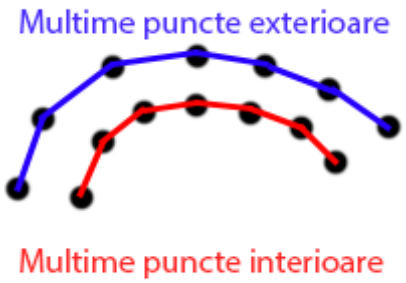
Explicația care urmează este o recomandare de generare a geometriei pistei, dar aceasta poate fi asamblată în orice fel, atât timp cât sunt respectate cele 2 restricții de mai sus.

Generarea pistei de curse se poate realiza conform pașilor de mai jos. O reprezentare vizuală a rezultatelor este prezentată în continuare.

1. Se creează punctele unui poligon ce definește pista de curse. Noi am folosit Geogebra.
2. Se creează două seturi de puncte suplimentare, în interiorul și exteriorul poligonului ce definește pista. Punctele se pot crea de-alungul perpendicularei fiecărui segment. Acest proces este detaliat mai jos.
3. Se creează muchiile dintre punctele poligoanului interior și exterior.
4. Se generează triunghiurile dintre punctele celor două poligoane. Triunghiurile se creează într-un singur obiect de tip mesh.



Pentru a genera geometria pistei (o suprafață 2D), este nevoie de 2 mulțimi de puncte: una interioară și una exterioară. Fiecare segment exterior este (aproximativ) paralel cu omologul lui din interior.

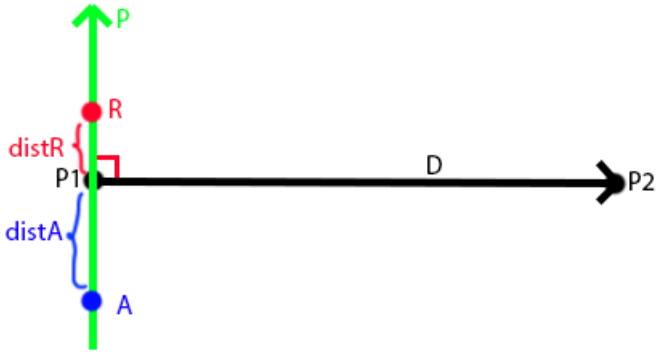


Aceste 2 mulțimi se pot obține folosind tool-urile deja discutate, însă este destul de costisitor la nivel de timp. O soluție mai elegantă este prezentată în continuare, unde de la o mulțime de puncte putem genera oricâte mulțimi de puncte paralele cu mulțimea inițială (paralelism la nivel de segmente).

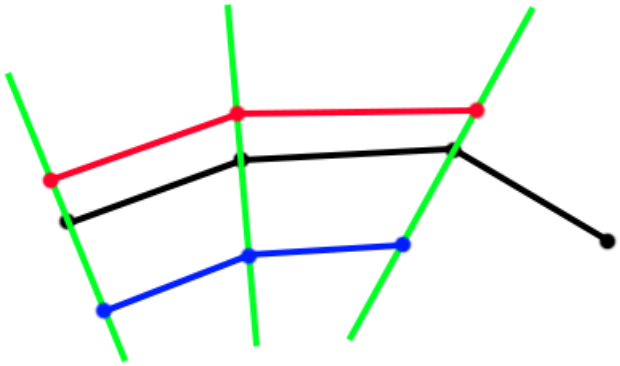
Să presupunem că avem 2 puncte consecutive P1(x1, 0, z1), P2(x2, 0, z2) din mulțimea de puncte care definește traseul:

1. Se determină vectorul direcție de la P1 către P2 notat D (negru in poza).
2. Se determină vectorul perpendicular pe D notat P (verde in poza): P = cross(D, UP), UP fiind vectorul perpendicular pe planul XOZ.
3. Folosind vectorul P și poziția P1 putem să obținem punctele exterioare / interioare (Roșu și Albastru): R = P1 + distRoșu * P, A = P1 - distAlbastru * P.

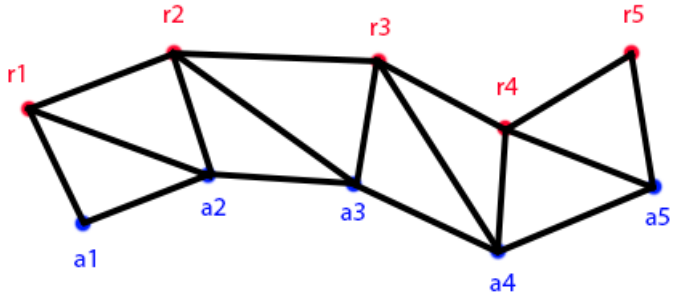
$$P1(x1, 0, z1) P2(x2, 0, z2)$$
$$\vec{D} = P2 - P1$$
$$\vec{P} = cross(\vec{D}, \overrightarrow{UP}), \overrightarrow{UP} = (0, 1, 0)$$
$$R = P1 + distRoșu * \vec{P}$$
$$A = P1 - distAlbastru * \vec{P}$$



Astfel, dacă aplicăm același lucru pentru toate segmentele, obținem noile mulțimi de puncte exterioare / interioare:



Având cele 2 mulțimi (exterioară și interioară), putem să construim suprafața pistei. Presupunem că avem punctele exterioare r1, r2, r3 și cele interioare a1, a2, a3. Trebuie să se construiască cele 4 triunghiuri (r1, a1, a2), (r1, a2, r2), (r2, a2, a3). (r2, a3, r3).



Astfel, dacă aplicăm același lucru pentru toate punctele din ambele mulțimi obținem pista.

O posibilă generare mai bună utilizează media perpendicularelor pe 2 segmente consecutive pentru a crea punctele de pe cele două poligoane, interior și exterior.

Deplasare obstacole dinamice

Pe pistă, pe lângă mașina jucătorului, trebuie să se deplaseze și alte mașini (adversari). Aceștia se deplasează pe un traseu predefinit paralel cu mulțimea generatoare de pistă. Acest traseu se poate obține cu tehnica prezentată mai sus.

Generare copăcei lângă pistă

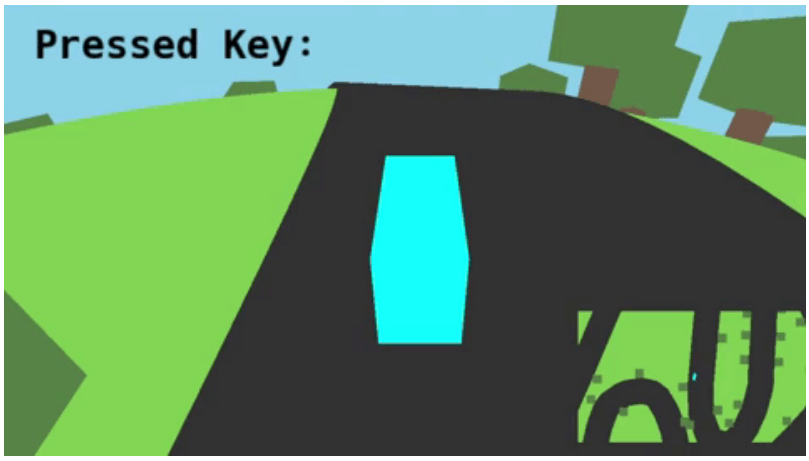
Pe lângă pistă se află copăcei cu scop decorativ. Constant pe ecran trebuie să se vadă cel puțin un copac (necesită densitate mare de copaci). Copacii nu trebuie să blocheze pista.

Control mașină

Dintre toate mașinile create pe pistă, pe una dintre ele va trebui să o alegeți ca fiind mașina jucătorului. Aceasta poate fi controlată folosind tastele **W, A, S, D**, unde W-S reprezintă mișcarea față-spate, iar A-D va schimba orientarea mașinii (detaliat mai jos).

Control cameră

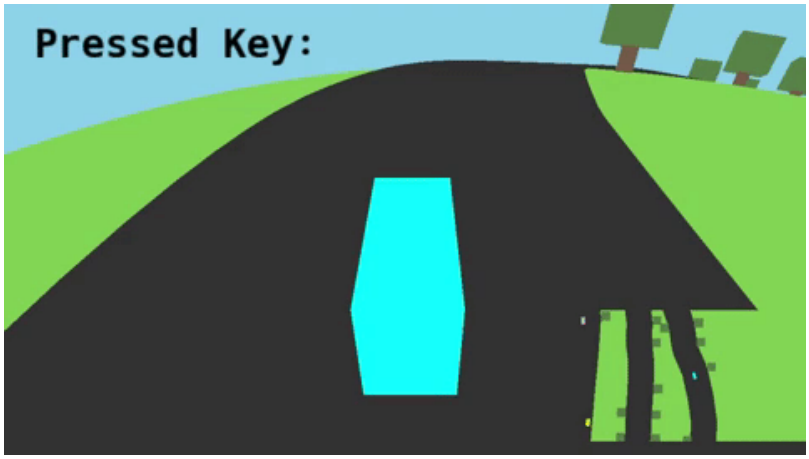
Perspectiva jucătorului va fi una **third-person**, deci vom avea camera poziționată în spatele mașinii, uitându-se spre aceasta. Pentru un control cât mai fluid, vom dori ca mașina să rămână centrată pe direcția camerei, deci la apăsarea tastelor A-D se va face atât o rotație a mașinii, cât și o rotație a camerei:



Pentru a păstra **poziția camerei la aceeași distanță în spatele mașinii**, la mișcarea față-spate (W-S), două alternative ar fi:

- Actualizarea poziției camerei odată cu cea a mașinii, la fiecare input valid.
- Actualizarea constantă (deci în Update(), fără verificări) a poziției camerei ca fiind: poziția mașinii la care se adaugă un offset (acel offset fiind distanța pe care camera o păstrează față de mașină).

Deplsarea față-spate (W-S) va implica mișcarea mașinii în funcție de rotația camerei, astfel încât comportamentul dorit să fie asemănător cu exemplul de mai jos:



Hint:

1) Calculele de deplasare și rotație pot deveni destul de complicate destul de repede, de aceea vă recomandăm să lucrați cu vectori, pornind de la exemplele și teoria din Laboratorul 4 [https://ocw.cs.pub.ro/courses/egc/laboratoare/04].

2) Pentru deplasarea față-spate (W-S), vă recomandăm folosirea vectorilor camerei (în cazul acesta, vectorul **forward**) pentru a afla direcția de deplasare a mașinii.

Exemplu: Din moment ce camera își schimbă vectorul de forward la rotația pe OY (lab 5 [https://ocw.cs.pub.ro/courses/egc/laboratoare/05]), o soluție ar fi folosirea acesteia pentru direcția de deplasare.

Utilizând ambele tehnici (rotația mașinii și deplasarea folosind vectorul forward al camerei), rezultatul final arată așa:

Verificare mașină dacă se află pe pistă

Pentru a verifica dacă mașina se află pe pistă, avem la dispoziție mai multe posibilități. Pentru simplitate, vom considera suficient dacă se verifică doar centrul mașinii să se afle în interiorul pistei. Orice soluție ce oferă o verificare corectă a acestei informații este bună.

Noi recomandăm oricare dintre următoarele 2 opțiuni:

- Verificarea dacă poziția mașinii (doar (x, z)) se află în cel puțin unul din triunghiurile ce definesc geometria pistei. Câteva resurse bune pentru acest proces se pot găsi aici https://www.baeldung.com/cs/check-if-point-is-in-2d-triangle [https://www.baeldung.com/cs/check-if-point-is-in-2d-triangle], https://blackpawn.com/texts/pointinpoly/ [https://blackpawn.com/texts/pointinpoly/].
- Verificarea dacă poziția mașinii (doar (x, z)) se află la o distanță mai mică de jumătate din lățimea pistei față de cel puțin un segment ce definește pista. O resursă buna pentru acest proces este aici http://paulbourke.net/geometry/pointlineplane/ [http://paulbourke.net/geometry/pointlineplane/].

Atenție!

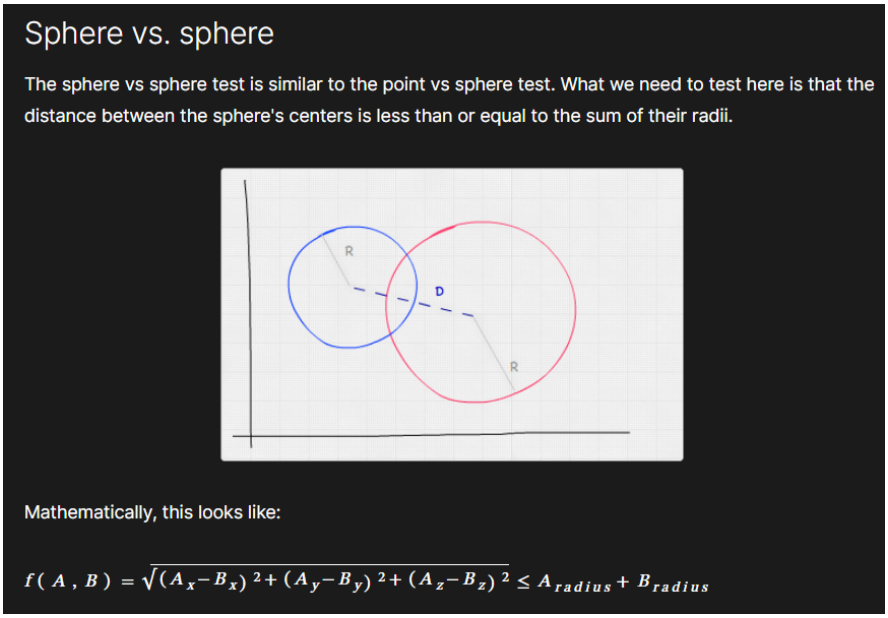
Pentru a avea un comportament asemănător celui din demo-ul video, trebuie ca mașina să nu se blocheze dacă se lovește de exteriorul pistei. Pentru a preveni acest blocaj, o strategie ar fi următoarea:

- În locul unei verificări constante în **Update()**, se poate face acest check la fiecare încercare de input;

Exemplu: Dacă se ține apăsat W, trebuie verificat mai întâi dacă cumva poziția nouă obținută în urma deplasării nu face coliziune. Dacă face, atunci inputul trebuie ignorat, altfel executat în mod normal.

Verificare coliziuni obstacole dinamice

Pentru a avea niște interacțiune cu mașiniile inamice, va trebui să le puteți “detecta”. Această detectare presupune o verificare a poziției mașinii jucătorului în raport cu fiecare mașină inamică. Pentru a realiza acest lucru, cel mai simplu mod pe care vi-l recomandăm este o verificare de coliziune sferă-sferă [https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection]:



În cazul nostru particular, sferele ar avea centrul în centrul mașinii, iar raza acesteia presupune distanță de verificare de coliziune.

În momentul în care este dectată intersecția dintre mașina controlată de jucător și un obstacol dinamic, mașina jucătorului trebuie să se oprească. Orice tastă apasată pentru controlul deplasării sau rotației nu mai trebuie să modifice poziția sau orientarea mașinii. Acest efect de oprire dispare în momentul în care nu mai există intersecție.

Minimap

În unul din colțurile ecranului (nu contează ce colț, noi am ales dreapta-jos pentru a fi mai ușor de vizualizat dar puteți alege orice poziție), va trebui să randați și un **minimap**.

În esență, un minimap nu reprezintă altceva decât un alt viewport [https://www.geeksforgeeks.org/window-to-viewport-transformation-in-computer-graphics-with-implementation/] unde trebuie să randați aceeași scenă iarăși.

Modul în care se poate crea un nou viewport in OpenGL (aveți un exemplu și in scheletul laboratorului 4) este folosind funcția:

```
glViewport(GLint x, GLint y, GLint width, GLint height)
```

Pentru a putea vedea de sus scena (cum este prezentat și în exemplul de mai jos), ar trebui să se întâmple următoarele lucruri:

- Mutarea poziției camerei deasupra mașinii (și prin urmare și poziția centrului camerei);
- Din moment ce se dorește o vedere de ansamblu asupra unei bucăți din traseu, se va trece camera într-o **proiecție ortografică** folosind funcția:

```
glm::ortho(float left, float right, float bottom, float top, float zNear, float zFar)
```

Atenție! Observați că rotația camerei nu se mai păstrează, dar deplasarea ei în funcție de jucător **da**!

Curbură teren

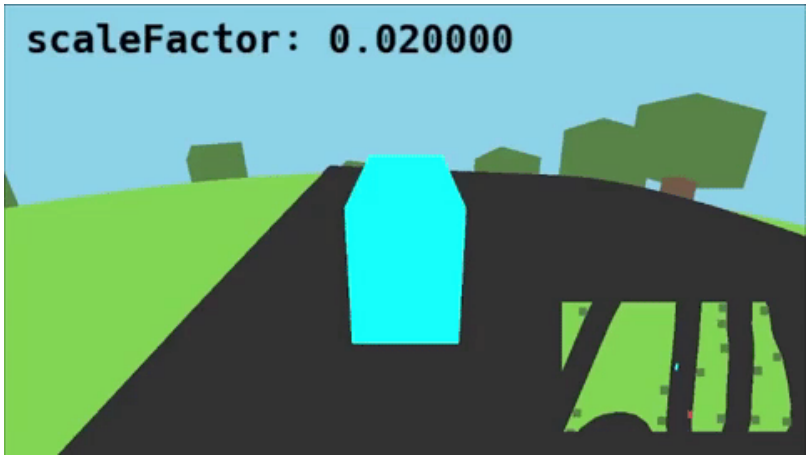
Crearea curburii din shader

Realizeaza curburii terenului se realizează la pasul de desenare a obiectelor din scenă. Acest efect se creează prin modificarea componentei y a coordonatelor pentru toți vertecșii obiectelor din scenă. Procesul este realizat în vertex shader. Componenta y a tuturor vertecșilor se modifică după cum urmează.

$$Pos_{v_y} = Pos_{v_y} - \|Pos_{car} - Pos_v\|^2 \cdot scaleFactor$$

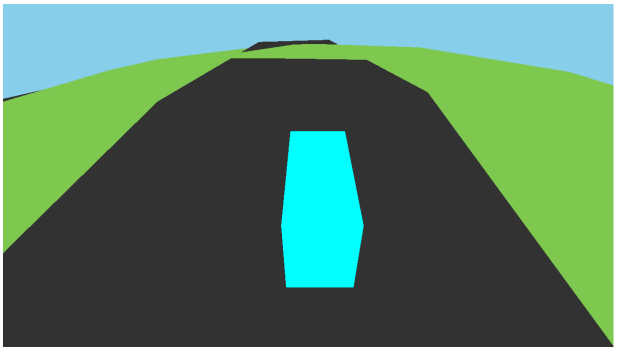
- Pos_v - poziția în spațiul lume a vertexului procesat de vertex shader
- Pos_{car} - poziția în spațiul lume a mașinii controlate de jucător
- Pos_{v_y} - componenta y a poziției vertexului
- $scaleFactor$ - un factor de scalare ce controlează curbura terenului

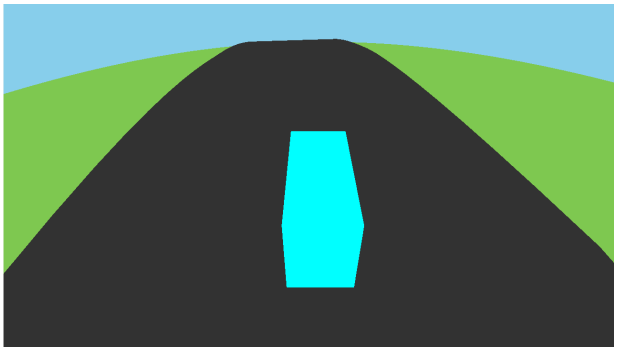
Factorul de scalare este proporțional cu dimensiunea obiectelor din scenă. Pentru demo-ul de mai sus, unde lățimea mașinii este de 1 unitate, a fost folosit un factor de scalare de 0.02.



Densitate de triunghiuri teren

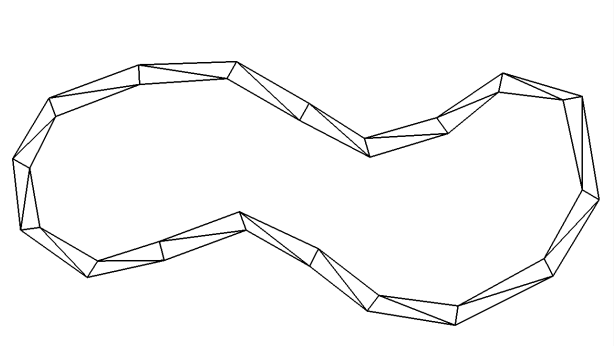
Deoarece realizarea curburii terenului este un proces ce modifică pozițiile vertecșilor, rezultatul vizual este influențat de densitatea de triunghiuri ale terenului. Cu cât terenul are o densitate mai mare de triunghiuri, cu atât efectul de curbura este mai natural. O densitate mică produce un rezultat vizual mai 'pătrățos'. O comparație între rezultatele celor două poate fi vizualizată mai jos. În stânga este prezentat rezultatul pentru o densitate mică și în dreapta pentru o densitate mare.

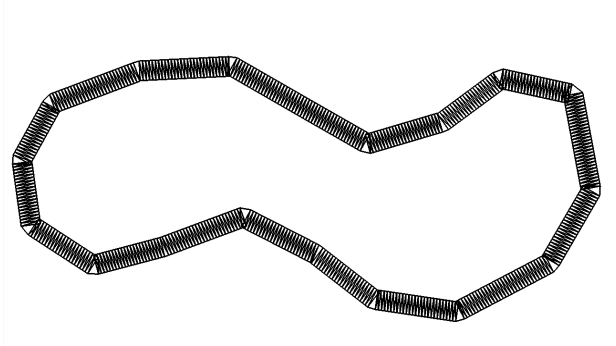




În plus, geometria terenului de iarbă poate trece prin geometria pistei pentru o densitate mică de triunghiuri, așa cum se poate vedea mai sus în stânga.

Pentru a elimina aceste artefacte vizuale, este necesară generarea terenului de iarbă și a pistei cu o densitate mare de triunghiuri. De exemplu, o comparație între geometria celor două rezultate de mai sus este după cum urmează. Geometria pistei a fost generată conform descrierii de mai sus.





Funcționalități obligatorii (150 puncte)

- Teren 40p
  - Generare geometrie teren iarbă și pistă 30p
  - Generare copăcei lângă pistă 10p
    - Generare locații copăcei 7.5p
    - Desenare copăcei 2.5p
- Obstacole dinamice 20p
  - Generare trasee diferite 7.5p
  - Desenare obstacole 2.5p
  - Deplasare obstacole 10p
- Control mașină 30p
  - Control deplasare și rotație 15p
  - Poziționare corectă cameră în spatele mașinii 10p
  - Oprire mașină la coliziunea cu obstacolele dinamice 2.5p
  - Oprire mașină la părăsirea pistei 2.5p
- Verificare corectă mașină dacă este pe pistă 15p
- Verificare corectă mașină dacă intersectează obstacolele dinamice 5p
- Minimap 20p
  - Calculare corectă proiecție ortografică 15p
  - Desenare într-o altă poartă de vizualizare 5p
- Curbură teren 20p
  - Efect de curbură din shader 10p
  - Generare geometrie teren cu o densitate mare de triunghiuri 10p

Atenție! Nerespectarea celor 2 restricții în generarea pistei de curse, 1. să nu existe niciun unghi drept între oricare 2 segmente consecutive, 2. să aibă formă concavă, atrage după sine nepunctarea elementelor ce utilizează informația pistei.

Exemple de funcționalități bonus

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

- Finalizarea jocului de curse de mașini. Realizarea verificării dacă a fost parcursă toată pista și cuantificarea numărului de tururi parcurse. Păstrarea timpului necesar parcurgerii fiecărui tur și finalizarea intrecerii după parcurgerea unui anumit număr de tururi. Afișarea unui UI ce conține numărul de tururi rămase de parcurs. Pentru a verifica dacă s-a parcurs un tur, se pot folosi puncte de verificare de-alungul traseului.
- Decorarea pistei cu diferite elemente vizuale, de exemplu crearea unei linii discontinue în centrul pistei.
- Desenarea mai multor tipuri de copăcei și a altor decorațiuni de o parte și de alta a pistei.
- Decorarea mașinii controlate de jucator și a obstacolelor dinamice, de exemplu realizarea formei de mașină ce conține și roți ce se rotesc în timpul deplasării. Roțile pot să aibă și o rotație față de axa OY când mașina își schimbă direcția de deplasare.
- Orice imbunătățește vizual scena.
- Adăugarea de mecanici mai complexe de controlare a mașinii. De exemplu, un efect de accelerație sau inerție.
- Adăugarea de pickups care odată lovite, oferă jucătorului diferite bonusuri (de exemplu, o creștere în viteză sau un control mai bun al vehiculului).

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 **Tema2**, cu fișierele **Tema2.cpp** și **Tema2.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 **Tema2**, 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/Tema2/Tema2.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/2022/02.txt · Last modified: 2022/12/08 19:21 by andrei.lambru