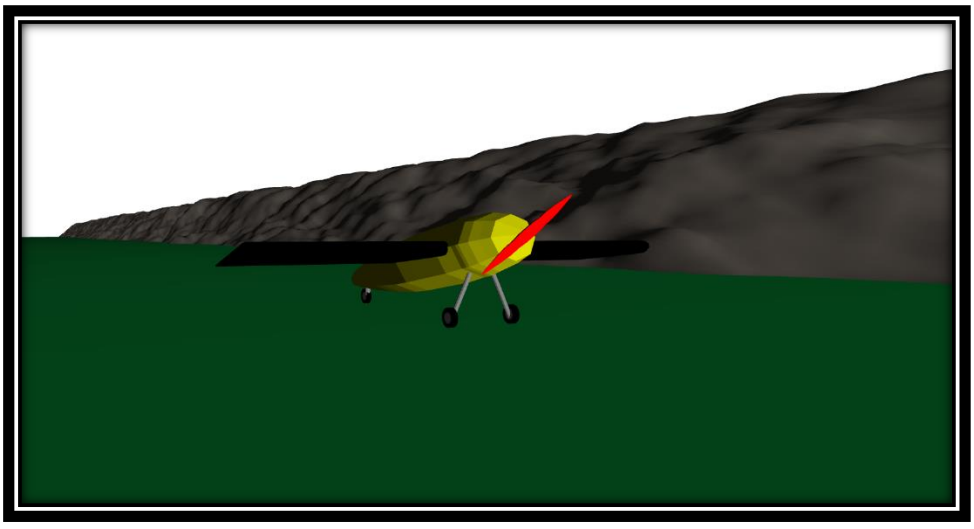
Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko

Hitrostno oviranje letenja

Poročilo Seminarja



Avtor: Timotej Šušteršič, 63210333

Ljubljana, 14.1.2024

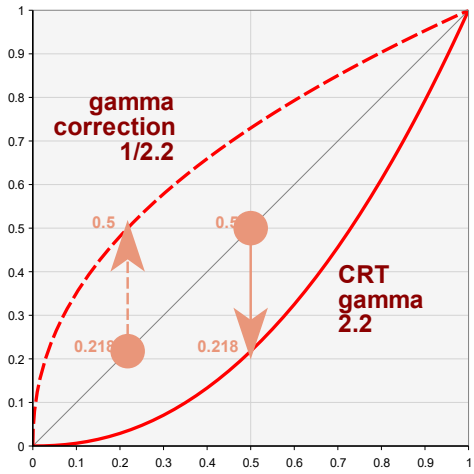
Uvod

Seminar zajema letenje po poligonu ovir. Cilj je končati v čim krajšem možnem času, kjer se ob vsaki ponovitvi težavnost poveča. Seminar temelji na osnovni orodja in GLTF nalagalnika, ki ga nudi predmet. Izdelovanje naloge je zahtevalo veliko korakov. V tem poročilu bom zajel vse ključne funkcionalnosti za delovanje igre in opisal njihovo delovanje.

Senčilnik

V senčilnikih sem implementiral: za senčenje Lambertov model, za odsev Phongov model, ter barvi dodal korekcijo gamma.

- Lambertov model izračuna kot med virom svetlobe in normalo površine, ter ga prišteje barvi. Zraven sem štel barvo luči, ter nek minimalni ambient, da neosvetljeni predmeti niso črni.
- Phongov model izračuna kot med vektorjem odboja žarka luči, ter vektorjem smeri kamere. Rezultat prav tako prišteje barvi površine, kar ustvari vtis odseva. V odsevu sem prav tako upošteval svetilnost luči.
- Korekcija gamma pomeni, da naše oči vidijo svetlost barve nelinearno, računalnik pa to predstavi linearno. Preko korekcije to popravimo, tako da barvo kvadriramo z vrednostjo 2,2.



Slika 1: Gamma Corekcija

Interakcija

Razred omogoča interakcijo predmetom, ko se jim približamo. Bližino zaznavamo enako kot kolizijo, le da so same meje kamere večje in na ta način, pride do kolizije hitreje oz. bližine. Interakcijo sprožimo preko tipke “E”. Začetek in konec, med katerima je minimalna razlika sekunda in pol, sprejemata callback funkcijo za akcije. Dodal sem tudi razširitev razreda za vozila, ki omogoča spreminjanje hierarhije. V tem koraku, spremenimo parametre za kolizijo, prestavimo kamero, ter ji popravimo smer.

Letalo

Letalo predstavlja večji del seminarja, zato je bilo vanj vloženega tudi največ dela. Na začetku sem imel vtis, da naloga ne bo preveč zahtevna, vendar je čas pokazal, da je povezava vseh funkcionalnosti zelo zahtevna. Še posebej, pa je bila zahtevna kalibracija letenja, ki je za realistično in gladko izvajanje, zahtevala veliko iteracij in načrtovanja.

Delovanje

Poskušal sem ga približati dejanskemu vozilu, zato imam motor z obrati, ki daje pospešek. Kontroliramo ga z tipkami “W” in “S”. Rezultat sta hitrost kot sila naprej in sila gravitacije. Obrati motorja narekujejo rotacijo propelerja in morajo od 3000 RPM doseči vsaj 800 RPM da pospešek postane pozitiven. Ko letalo doseže dovolj veliko hitrost lahko vzleti.

Premikanje (Letenje)

Premikanje je v osnovi podobno letenju v igri GTA V.

Letalo se lahko, še posebej v zraku, premika samo naravnost zato vsak frame, vzamem smer letala in ga prištejem forward vektorju, ki je enotski vektor po X osi. Nato je povečan glede na hitrost letala, ter sile gravitacije. Pred tem pa preko kontrolnih tipk, "A, D, Levo, Desno, Gor, Dol", izvajam rotacije preko posamezne osi. Lokalno se izvaja tudi linearna interpolacija rotacij, saj kljub temu, da tipka predstavlja binarno stanje, je v ozadju vrednost rotacije razpon [-1, 1], kjer je 0 mirujoče stanje. Ob pritisku na tipko, se vrednost počasi pomika na svoj maksimum. Ko ni pritisnjena počasi pada nazaj v mirujoče stanje. Ko pa se smer obrne in je vrednost na nasprotni strani sredine, se ta zelo hitro pomika do mirujočega stanja in nato spet počasi, do svoje skrajnosti. Na ta način omogočam zelo gladek prehod rotacij.

Na tleh omogočam samo obrat po Y osi, v zraku pa preko vseh osi.

Pristanek

Pristanek zahteva, da letalo poravnam z tlemi. To je bil tehnično velik zalogaj, saj kljub vseh prednosti kvaternionov, niso vedno najboljši. S tem mislim pretvorbo v Eulerjev sistem.

Prvi izziv je bil translacija na tla. Ni se mi zdela smiselna kolizija s tlemi, zato sem na začetku shranil Y translacijo letala in na ta način hranil nadmorsko višino, ki je tudi povedala, če je letalo v zraku.

Sedaj pa je potrebno nastaviti še rotacijo, kjer moram iz trenutne izluščiti Y os ostali dve pa ponastaviti, kar pa z kvaternioni ni samoumevno. Rešitev je, da vzamem iz rotacije letala forward vektor, enako kot pri premikanju, ter osnovni vektor tal med njima. Nato po enačbi: $cos(angle) = (a * b) / (|a| * |b|)$ ugotovim kot letala, vendar dobim samo vrednost od 0 (naprej) do π (nazaj) kar je 180°, torej ne vem po kateri strani sem obrnjen. Zato sem se izračunal še normalo med vektorjema, kjer negativna pomeni levo in pozitivna desno. Kar ostane je pretvorba v razpon 0° do 360° in rotacija letala.

Trčenje

Dogodek se zgodi, ko letalo pod pre-ostrim kotom pristane (izračunan kot pri pristanku), ter ob koliziji z oviro. Premikanje letala se nadaljuje, le motor ugasne. Dogodek se izpiše na ekranu in igra se ponastavi.

Ovire in njihova kolizija

Ovire so obroči, skozi katere mora letalo preleteti. Ustvarijo se ob začetku igre, tako da kopirajo strukturo že obstoječega kroga, nato pa dobijo svojo translacijo, kjer imajo neko randomizacijo za raznolikost. Glede na težavnost igre, je odvisna tudi oddaljenost med njimi.

Kolizija je med njimi seveda drugačna, ker prvič niso kocke temveč sfere in drugič, moram razločiti kdaj je prišlo do kolizije, ter kdaj je letalo samo letelo skozi. Prvi problem ni težak. Samo uporabim preprosto kolizijo sfer z izračunanjem razdalje med središči: $distance = \sqrt{((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)}$, kjer mora biti prav ta, manjša od vsote njunih polmerov. Drugi problem pa rešim na enak način kot prvi, le da si predstavljam sfero v 2D in letalo v 3D ter po enaki enačbi,

kot prej le v 2D obliki (YZ) preverim, če se razdalja + polmer letala, nahaja znotraj kroga ali na njegovi krožnici.

Tretjeosebna perspektiva (Third Person Controller)

Čeprav sem bil zadovoljen z letenjem letala, sem menil, da ga težko nadzoruješ, ker vidiš samo naravnost in postane zaznavanje ovir zelo težko. Zato sem si zadal izziv, da vozilu dodam tretje osebno perspektivo.

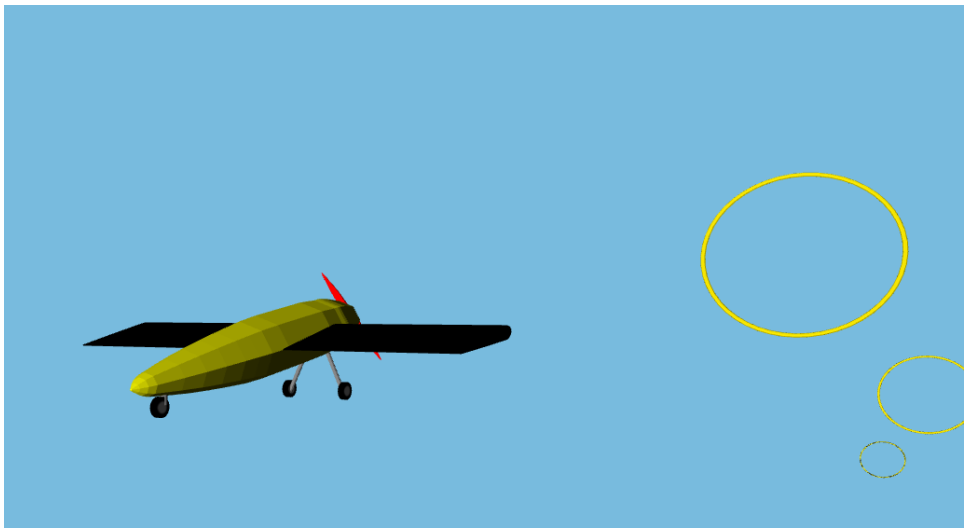
Same kontrole so podobne kot pri prvoosebnem pogledu, z dodatkom razdalje kamere od letala. Za delovanje so potrebne dve stvari:

1. Translacija kamere okrog objekta na določeni razdalji. Ta je bila dokaj preprosta, saj sem na spletu poiskal preproste trigonometrične enačbe za računanje XYZ:

   $$x = distance * \sin(pitch) * \cos(yaw)$$
   $$y = distance * \cos(pitch)$$
   $$z = distance * \sin(pitch) * \sin(yaw)$$

2. Kamera mora vedno biti usmerja proti objektu. Tukaj je bilo dosti več iteracij, in nadaljevanje težav pristajanja letala. Na koncu sem našel čisto in preprosto rešitev. Knjižnica gl-matrix ima metodo targetTo(), kjer ji povemo objekt in tarčo, ta pa vrne mat4 matriko iz katere izluščimo rotacijo in jo pripišemo kameri.

Translaciji sem za lepšo obnašanje dodal linearno interpolacijo, z metodo iz knjižnice lerp(), kjer povemo začetno in končno translacijo, ter hitrost. Efekt je zelo zadovoljiv.

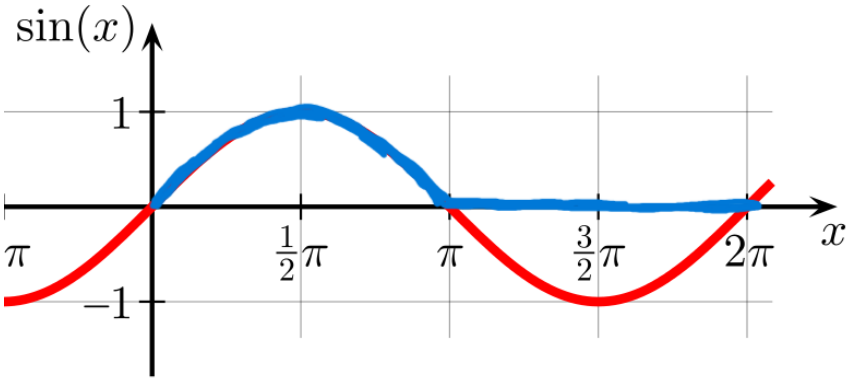


Slika 2: Predstava tretjeosebne perspektive

Sistem Delcev (Particle System)

Za dvig »prahu« pod kolesi letala, sem dodal sistem delcev, ki pod objektom, na naključnih lokacijah začne ustvarjati majhne temne kvadratke oz. delce, ki simulirajo prah. Količino definiraš ob vsakem frame-u. Sam sem izbral kvadratni koren hitrosti letala. Po končani življenjski dobi, se delec odstrani.

Za gibanje delcev sem si izbral sinusno funkcijo. Najprej sem x normaliziral glede na starost in življenjsko dobo [0, 2π]. Nato določil minimalno vrednost 0 in na koncu funkcijo zožil s skalacijo. Na začetku torej majhen lok in nato konstanta 0.



Slika 3: Modra predstavlja gibanje delcev

Slika 4: Praktična predstavitev delcev

Game

Delovanje igre je dokaj preprosto. V Blenderju sem postavil manjšo hiško ob letalu, na kateri je rdeč gumb za pričetek igre. Nato se usedeš v letalo, vzletiš, premagaš vse ovire, tako da čez njih poletiš, letalo obrneš, pristaneš in spet pritisneš enak gumb, kjer si začel. Takrat se čas ustavi, doda 15 sekundna kazen za vsako zgrešeno oviro in poveča težavnost.