UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Stickman Fight

Seminarska naloga pri predmetu Računalniška grafika

Matija Ojo

Jan Renar

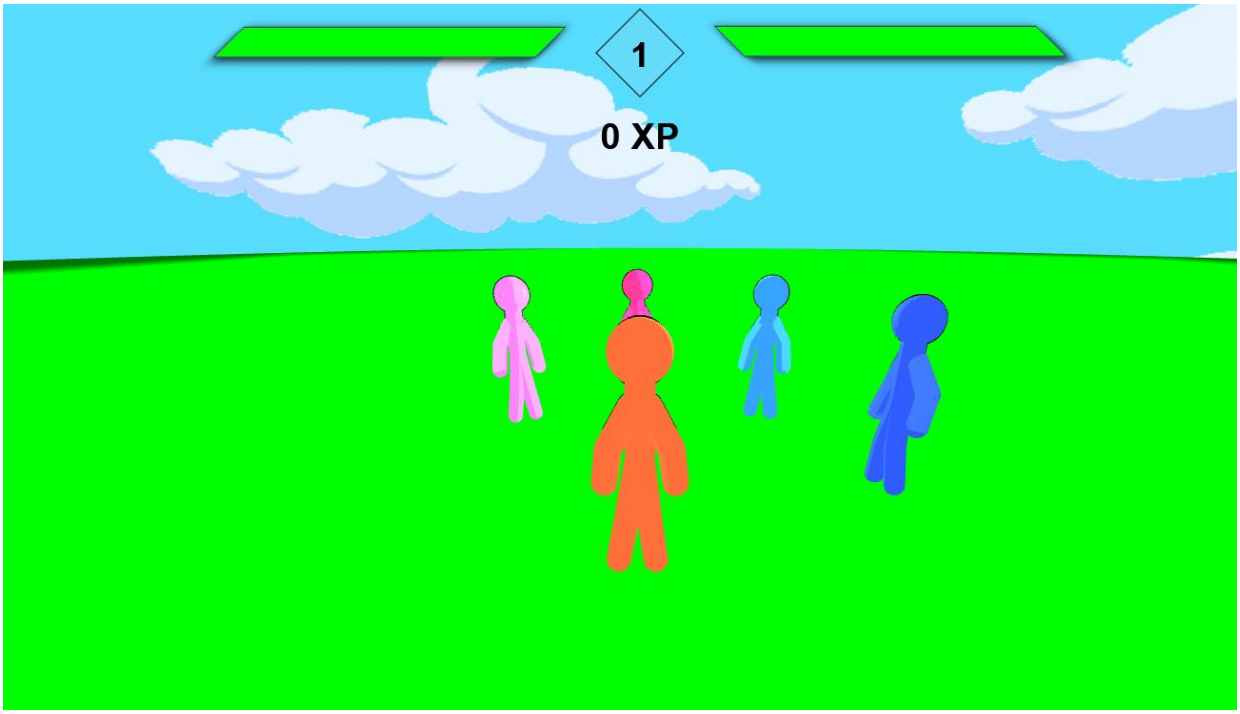
Ljubljana, januar 2022

Kazalo

1. Uvod

V okviru WebGL-a sva naredila igro, v kateri se uporabniki bojujejo s stick figurami.

V igri so implementirane skeletne animacije in skybox – za modela razvita v programu Blender, luči, obroba stick figur ter več igralski način preko omrežja. Strežnik zahteve posluša na vratih 7777, pri čemer je zahtevek potrebno poslati na index.html.



Slika 1: Izgled igre

2. Skeletne animacije

Skeletne animacije temeljijo na »kosteh« in premikanju le teh, kosti namreč določijo kako naj se ustrezne točke 3D modela premikajo, vrtijo ali skalirajo – v nadalje bo za premik, vrtenje in skaliranje uporabljen izraz **transformacija**.

Skeletna animacija je torej le zaporedje transformacij kosti, pri čemer je vsaka označena s keyframe-om. Za koherentno, zvezno animacijo je potrebno med keyframe-i vrednosti ustrezno interpolirati. To je možno storiti ta več načinov, v projektu je uporabljena preprosta linearna interpolacija.

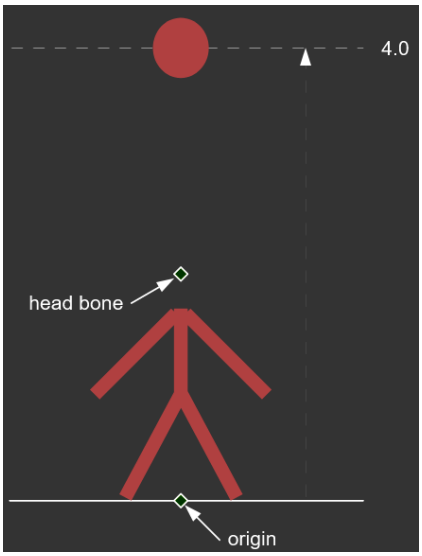
Animacije so shranjene v Javascript objektu, s ključema:

- t: hrani čas določenega keyframe-a
- v: hrani transformacijo določene kosti ob danem času t

2.1 Armatura

Pri skeletnih animacijah je ključnega pomena na kakšen način se shranijo kosti, prebrane iz datoteke formata gltf. Vsaka kost ima poleg imena in transformacije tudi naslednje matrike, ki so praktično obvezne, če želimo upoštevati hierarhijo kosti:

- **Lokalna matrika**: hrani lokalne koordinate kosti in se izračuna med obdelavo animacije (tukaj je shranjena vsaka vrednost kosti med interpoliranjem keyframe-ov, kar je razloženo na prejšnji strani)
- **Svetovna (globalna) matrika**: je ključnega pomena za simuliranje hierarhije, izračuna se namreč tako da se starševa svetovna matrika množi s trenutno lokalno matriko. V primeru ko starš ne obstaja (korenska kost) je ta matrika enaka lokalni matriki. Na ta način vsak premik, rotacija, skalacija kosti vpliva tudi na vse otroke te kosti. Gre za zelo eleganten način doseganja hierarhije, saj lahko premik, rotacijo, skalacijo predstavimo z relativno preprosto matriko, za doseganje hierarhije starš-otrok pa matrike preprosto množimo.
- **Inverzno vezavna matrika** (»Inverse bind pose«): Ker imajo tako točke modela kot kosti modela svoje pozicije v prostoru, nastane problem ko želimo točko modela spremeniti (jo množiti) glede na to kje se nahaja kost. Množijo se namreč pozicija točke s pozicijo kosti in ker gre za popolnoma ločene koordinate v prostoru, bi dobili točko, katere pozicija je enaka produktu teh točk. Rezultat bi bil nekaj takega:



Slika 2: Produkt točk brez inverzno vezavne matrike

Iz tega razloga je potrebno pozicijo kosti nekako “odšteti”, kar je storjeno tako da se svetovna matrika posamezne kosti (globalna pozicija v prostoru) invertira, ter rezultat shrani v novo matriko.

Ko je ta struktura zagotovljena se v vsakem frame-u določi transformacija vsake kosti glede na trenutno vrednost znotraj interpolacije, svetovna matrika se množi z inverzno vezavno matriko, nakar se rezultat shrani v tabelo, ki se nato shrani kot uniforma znotraj vertex senčilnika.

Senčilnik nato upošteva **uteži** kosti in preko **indeksov** (atributa senčilnika) množi 2 zaporedni kosti, kar pomeni, da na določen vertex vplivata kvečjemu 2 kosti, rezultat vseh množenj pa shrani v matriko, ki se množi s pozicijo veretex-ov.

Na tej točki bi rad izpostavil blog, ki skeletne animacije zelo dobro razloži in je k razvoju le teh neizmerno pripomogel.

Stick figura ima naslednje animacije: Idle, Run, Tired, Dies, Hit_Center, Hit_R, Hit_L, Punch_R, Punch_L, Kick_R, Kick_L, Kick_Spin, Kick_Jump, Punch_Uppercut.

Spremenljivka currAnimation znotraj razreda Player določa katera animacija se trenutno izvaja. To omogoča, da preko reference na instancno igralca določamo tudi animacije ob raznih dogodkih kot so kombo napad, figura je pod napadom, figura umre…

3. Kamera in kontrole

Tretjeosebno kamero lahko uporabnik premika z miško, z kolescem pa spreminja oddaljenost kamere od igralca (spremenljivka viewDistance). Kamera je implementirana tako, da se v vsakem klicu funkcije »render« pozicija kamere nastavi na pozicijo igralca, nato pa se translira po Z osi za viewDistance enot. Nazadnje se s funkcijo lookAt iz webgl matrix module-a izračuna matrika pogleda, tako da je kamera vedno usmerjena proti igralcu. Izboljšava bi lahko bila ta, da kamera gleda v isto smer kot igralec, torej dodal bi se vektor določene dolžine, ki kaže v smeri igralca, in bi se upošteval pri lookAt matriki. Na podoben načni namreč delujejo ostale tretjeosebne igre.

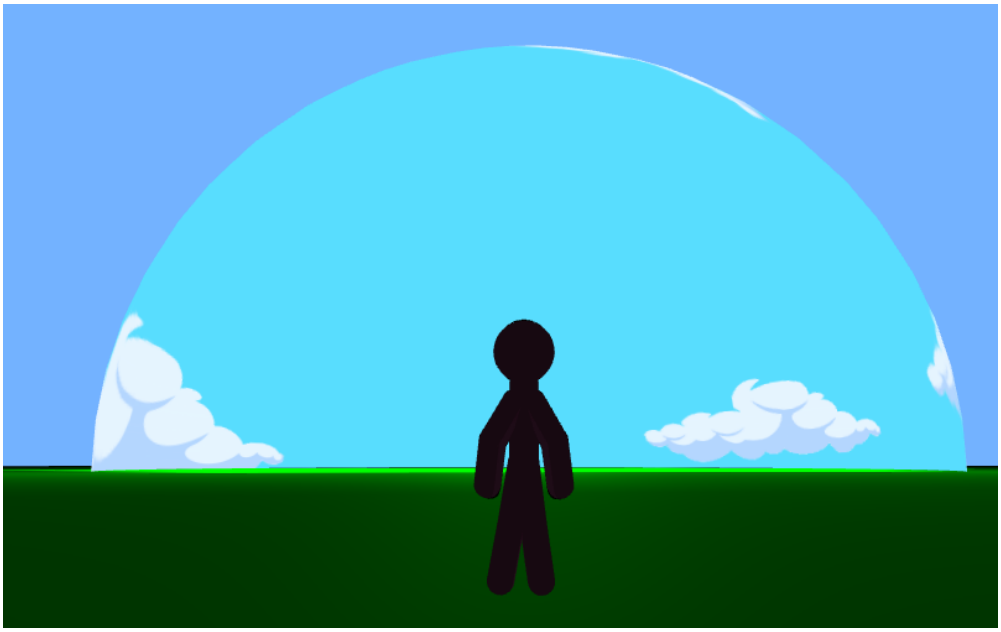
Uporabnik stick figuro premika s tipkami WASD, napade pa izvaja s puščicami (arrow keys). Ob izvedbi več ustreznih napadov, se sproži kombo napad. Kombo napadi preprečujejo to, da bi uporabniki napadali z le eno tipko, kar bi bilo zelo nezanimivo. Dodajo pa tudi dodaten cilj oziroma nivo težavnosti, kar naredi izkušnjo veliko bolj zabavno.

4. Skyball

V programu Blender sem v datoteki stick figure ustvaril kroglo, ji nastavil teksturo, invertiral normale ter skaliral. Na ta način je možno narediti izvoz v le 1 gltf datoteko, kar je zelo priročno. Uporabljena je naslednja tekstura – sliko oblaka sem našel na internetu.

Slika 3: Tekstura za skyball

Spodaj je prikazan dobljeni rezultat s skyball-om, le da nanj gledamo od zunaj. Opomba: ta slika je bila ustvarjena specifično za prikaz skyball-a, v igri stick figure ne morejo zapustiti območja, ki ga omejuje kupola.



Slika 4: Pogled na skyball od zunaj

5. Osvetlitev in obroba figur

Osvetlitev realizirana v okviru Cel shading-a, ki do neke mere igri poda izgled, kot bi bila narisana ročno. Pomagal sem si s tem blogom. Za izračun **zrcalnega** (specular) faktorja je uporabljen Phongov model, medtem ko je izračun **razpršilnega** (diffuse) faktorja implementiran nekoliko drugče.

Razpršilni faktor

Ko je izračunan skalarni produkt med normaliziranim vektorjem luči ter normale površine, je izračunana spremenljivka **delta** s pomočjo funkcije **fwidth**, ki vrne vsoto absolutnih vrednosti odvodov v trenutnem pikslu po x in y smeri, z drugimi besedami, vrne za koliko se trenutni piksel razlikuje od sosednjih. Nazadnje se izračuna spremenljivka diffuseSmooth s funkcijo **smoothstep(a, b, v)**, ki je ravno nasprotna funkciji lerp in vrne vrednost t, za podani parameter v na intervalu [a, b]. Na ta način je doseženo to, da se svetloba širi na nekoliko bolj zvezen način.

Luč

V igri je le ena luč, ki je hardkodirana – enaka za vse igralce in ima vlogo sonca. Pomembna podrobnost je to, da se matrika pogleda množi s transformacijsko matriko luči, šele nato pa se nastavi kot uniforma v senčilniku. Kar pomeni, da se osvetlitev pravilno izračuna tudi ko figuro le vrtimo (spreminjamo matriko pogleda) in ustvari efekt pravega sonca. Demo.

5.1 Obroba figur

Obroba je implementirana s testom šablone (stencil test). Deluje tako, da se najprej izriše stick figura, nato se test šablone nastavi enakost (gl.EQUAL), kar pomeni da se isti piksli ostanejo nespremenjeni, nato pa se »čez« stick figuro izriše še ena stick figura črne barve, ki je nekoliko povečana (extrudirana) v smeri normal. Test šablone pri tem pomaga, da se izrišejo le piksli, ki so »izven« stick figure. Rezultat je prikazan na sliki na naslednji strani in je nekoliko skromen, saj so normale iz nekega razloga popačene, tako je obroba vidna le na določenih mestih. Vsekakor pa pristop s testom šablone deluje.

Slika 5: Izgled stick figure

6. Networking

Socket.io je odprtokodna Javascript knjižnica za spletne aplikacije, ki se izvajajo v realnem času. Omogoča obojestransko komunikacijo med strežnikom in odjemalci z uporabo protokola HTTP in Websocket-ov. Sestavljena je iz dveh delov: knjižnice na strani odjemalcev, ki jo izvaja brskalnik in knjižnice Node.js na strani strežnika. Velika prednost socket.io knjižnice je podpora za asinhrono oziroma ne sekvenčno izvajanje. Pri spletnih aplikacijah, ki delujejo v realnem času je to ključnega pomena, saj gre za velike količine podatkov in dogodkov, ki se vsi izvajajo sočasno. V primeru, da ne bi uporabljali tega načina, bi se vsak nov dogodek, izvedel šele po koncu prejšnjega, kar bi upočasnilo delovanje strežnika in posledično spletne aplikacije.

Stanje igre predstavlja položaje igralcev, njihove pridobljene točke (XP in Combo Multiplier) ter njihove udarce, v nekem določenem trenutku. Ti podatki so shranjeni tako na strežniku kot odjemalcu. Odjemalec po vsakem klicu metode update() pošlje strežniku posodobitev svojega stanja. Iz zahteve strežnik razbere, kateri odjemalec pošilja posodobitev, ga poišče (v tabeli, kjer so shranjeni vsi prisotni igralci) in mu posodobi stanje. Ker gre za igro v realnem času, mora biti komunikacija med strežnikom in odjemalci obojestranska, da lahko tudi odjemalci prejmejo posodobitve stanja kar se da hitro. S tem namenom, ima strežnik ustvarjeno metodo heartbeat(), ki vsakih 10 milisekund, pošlje trenutno stanje igre vsem odjemalcem. To poskrbi,

da imajo vsi odjemalci usklajene podatke o trenutnem stanju igre. Posodobitev vsakih 10 milisekund predstavlja 100 poslanih zahtev vsako sekundo, kar pozitivno vpliva na uporabniško izkušnjo. Če bi bil interval daljši, bi bili zamiki in skoki ostalih igralcev veliko bolj opazni. To se zgodi, če odjemalec nekaj časa ne prejme posodobitve, ostale igralce izrisuje na zadnji poznani lokaciji, ko pa dobi posodobitev, pride do zelo opaznega preskoka, ko igralce spet izriše na novih položajih. Ker smo se odločili za preprostejšo implementacijo strežnika in odjemalcev, metode kot so interpolacija položajev igralcev ali napovedovanje njihovih premikov, ne uporabljamo. Kot možno izboljšavo bi seveda lahko uporabili drugačno implementacijo strežnika z uporabo komunikacije preko UDP protokola (npr. WebRTC).

Podatki katere si izmenjujejo odjemalci in strežnik so običajni Javascript objekti, ki vsebujejo relevantne podatke. Tako odjemalci kot strežnik imajo določene poslušalce dogodkov (event listener-je), ki se sprožijo ko socket.io knjižnica prejme sporočilo prek omrežja s pravilnimi parametri. Te metode so uporabljene vsakič, kadar strežnik pošilja podatke odjemalcu. Primer take metode je socket.on("updatePosition", x,y), ko je ta metoda izvedena, bo igralcu posodobila položaj, na prejete koordinate x in y. Odjemalci vsako posodobitev s strežnika izpisujejo v konzolo, kjer je možno videti prejete podatke.

7. Zaključek

Čeprav sta v igri le 2 modela in 1 tekstura, je bilo dela kar veliko. Delo sva si razdelila tako da je nekdo delal na 3D modelu in animacijah, nekdo pa na omrežnem delu. Z rezultatom, sva zelo zadovoljna, in četudi projekt še zdaleč ni bil enostaven, sva vesela da sva se zanj odločila in se pri tem ogromno naučila.