Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aljaž Pungerčar

# Postopek 3D modeliranja in animacije za uporabo v igri

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Narvika Bovcon

Ljubljana, 2019

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi predstavite izdelavo 3D karakterja za računalniško igro. Zmodelirajte in skiparite osebo, oblačila in opremo, dodajte ustrezne materiale in teksture. Model optimizirajte za igro. Animirajte nekaj ključnih dejavnosti osebe, kot je npr. hoja. Junaka predstavite v izbranem 3D okolju. Opišite vsa uporabljena orodja in celoten proces izdelave.

*Zahvaljujem se mentorici izr. prof. dr. Narviki Bovcon in moji družini.*

# Kazalo

# Uporabljena terminologija in seznam uporabljenih kratic

Pri izdelavi diplomske naloge smo se pogosto srečali z angleškimi izrazi, za katere v slovenskem jeziku še ni najbolj ustreznih prevodov. Pri prevajanju le-teh smo se potrudili čimbolj približati pravemu pomenu, za lažje razumevanje pa je ob večini teh izrazov v oklepajih naveden angleški prevod.

| **kratica** | **angleško** | **slovensko** |
|---|---|---|
| **FPS** | frames per second | okvirji na sekundo |
| **IK** | inverse kinematics | inverzna kinematika |
| **PBR** | physically based rendering | senčenje na osnovi fizike |
| **UE4** | Unreal Engine 4 | |

# Povzetek

**Naslov:** Postopek 3D modeliranja in animacije za uporabo v igri

**Avtor:** Aljaž Pungerčar

V diplomski nalogi je opisan eden izmed najbolj razširjenih pristopov k 3D oblikovanju in animaciji karakterjev, ki jih lahko nato uporabimo v modernih računalniških igrah. V nalogi je na konkretnem praktičnem primeru prikazan potek dela, ki obsega vse od samega zbiranja referenčnih slik, skic, idej, do prve grobe oblike modela, dodajanja detajlov in kasneje njegove animacije ter končno teksturiranja. V igralnem okolju demonstriramo uporabo narejenih modelov. Tekom te naloge je predstavljenih tudi več problemov, s katerimi smo se soočili, kot je na primer optimizacija ali zmanjšanje števila poligonov, ki sestavljajo mrežo našega modela.

Celoten postopek zahteva uporabo množice programske opreme. Za oblikovanje in animacijo je uporabljen odprtokodni Blender 2.79, za igralni pogon Unreal Engine 4 ter za teksturiranje Substance Painter. Vsak izmed teh programov je v nalogi na kratko predstavljen.

Cilj diplomske naloge je predstaviti celotni postopek izdelave igralnega karakterja in razjasniti nejasnosti, ki se lahko ob tem pojavijo.

**Ključne besede:** 3D modeliranje, animacija, računalniške igre.

# Abstract

**Title:** 3D modeling and animation of a character in a computer game

**Author:** Aljaž Pungerčar

This diploma thesis describes one of the most widely spread approaches to 3D modelling and animation of characters for use in modern video games. The workflow is presented in detail with practical examples. It consists of collecting reference images, ideas, modelling our first rough shape of the model, adding details and later creating animations and finally texturing it. In the game environment, an example usage of made characters is demonstrated. Over the course of this thesis multiple problems are encountered, e.g. reduction of number of polygons from our base mesh.

This whole process requires usage of mainly three different computer programs. Blender 2.79 is used for modelling and animation, Unreal Engine 4 for final demonstration in real game environment and Substance Painter for texturing part of the workflow. Each software is also briefly presented in a separate section of the thesis.

The goal of this diploma thesis is to present a complete workflow when creating playable characters and clarify any ambiguities one may encounter.

**Keywords:** 3D modelling, animation, computer games.

# Poglavje 1

# Uvod in motivacija

V svetu računalniške grafike je napredek neustavljiv. Razvijalci iger se z vsakim letom uspejo s svojimi stvaritvami še bolj približati našemu resničnemu dojemanju sveta, pri tem pa jih nekoliko ovira razvoj strojne opreme, za katerega bi lahko rekli, da zaostaja za razvojem programske opreme. Zato morajo biti razvijalci iger, s tem mislimo tako na programerje kot na 3D oblikovalce in ostale, previdni pri snovanju svojih izdelkov. Programerji morajo pametno in učinkovito zastaviti in izpeljati svoje cilje, da stvari gladko tečejo na skorajda vsakem modernem računalniku, prav tako pa morajo biti 3D oblikovalci pozorni, da svojim kreacijam ne dodajajo preveč nepotrebnih detajlov, ki bi lahko upočasnili izrisovanje scen in s tem celoten sistem.

V tej diplomski nalogi je predstavljen eden izmed bolj razširjenih in učinkovitih pristopov k 3D oblikovanju karakterjev, ki so v skoraj vsaki igri glavni akterji in središče pozornosti. Po v celoti končanem prvem karakterju bomo še opisali, kako lahko iz že narejenega modela oblikujemo novega, ne da bi začeli od začetka.

# Poglavje 2

# Predstavitev uporabljene tehnologije

## 2.1 Blender

Blender je odprtokodno orodje, ki je uporabno za vrsto različnih funkcij [1]. Glavne so: modeliranje, kiparjenje, teksturiranje, izdelava okostja za animirane modele (angl. rigging), animacija, simulacije fizike, izrisovanje končnih fotorealističnih scen, urejanje video posnetkov, izdelava vizualnih učinkov (ang. VFX). Blender nudi obenem tudi svoj igralski pogon imenovan Blender Game, ki pa v času pisanja te diplomske naloge zaenkrat še zaostaja za konkurenčnimi pogoni, kot so Unity in Unreal Engine 4 (UE4).

Slabost Blender orodja je, da obstaja na trgu različna programska oprema, specializirana posebej za določeno področje izdelovanja 3D vsebin. Za primer lahko vzamemo programsko opremo ZBrush podjetja Pixologic, ki je fokusirano predvsem na samo kiparjenje 3D modelov in tako na tem področju v prednosti pred Blenderjem. Podobno sta programa Substance Painter in Substance Designer podjetja Allegorithmic uporabljena izključno za teksturiranje in izdelavo lastnih materialov in na tem področju občutno zmogljivejša. Vseeno pa se Blender ponaša s ključno prednostjo za študente in neprofesionalne uporabnike in to je njegova odprtokodnost, kar pomeni, da ga lahko

uporablja kdor koli za kakršne koli namene, brez nakupa profesionalnih licenc, kot je to pogoj pri ostalih profesionalnih orodjih, nekaj izmed njih naštetih v prejšnjem odstavku. To je tudi razlog, da je v tej diplomski nalogi uporabljen za večino opravljenega dela, od modeliranja do animacije.

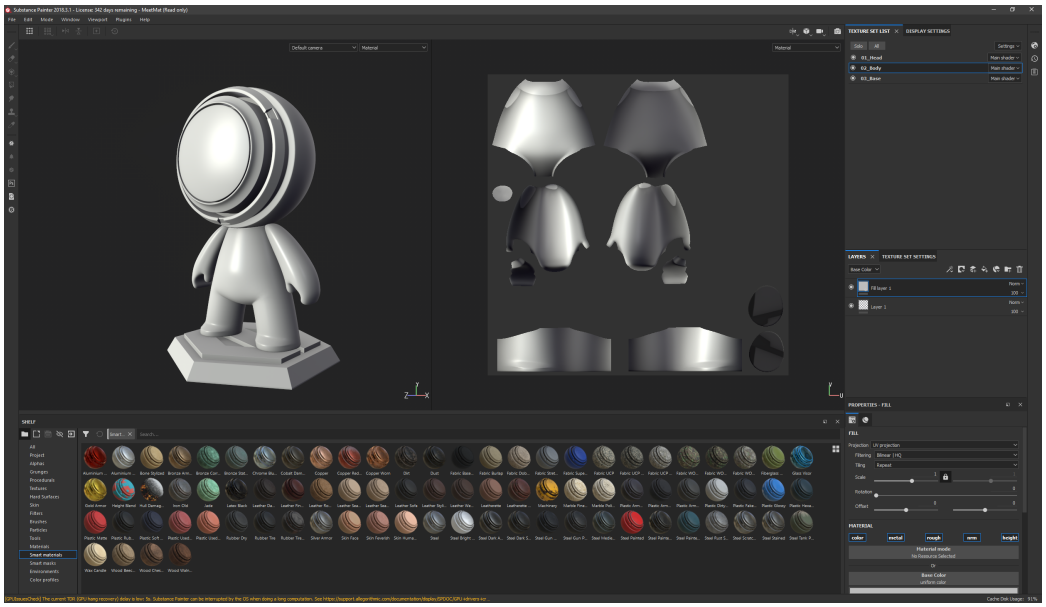
Slika 2.1: Blender, uporabniški vmesnik

## 2.2 Substance Painter

Substance Painter je orodje podjetja Allegorithmic in v času pisanja te naloge eno izmed vodilnih orodij na področju teksturiranja 3D modelov, saj je bilo uporabljeno za množico modernih uspešnic s področja video iger.

Velika prednost je njegovo nedestruktivno okolje, saj deluje na podoben način kot Adobe Photoshop, ki temelji na uporabi plasti (angl. layers). Prav tako je nedestruktiven, ko želi uporabnik zamenjati resolucijo svojih tekstur, zelo pogosto je namreč, da uporabnik v samem programu uporablja nižjo resolucijo, na primer 1024x1024, ko pa je izdelek končan in pripravljen za izvoz, pa poveča resolucijo na 4096x4096 in s tem dodatno izboljša kvaliteto

svojega izdelka. Dodatna prednost Substance Painterja je tudi njegova kompatibilnost z igralnimi pogoni, med njimi tudi z UE4, saj je zelo enostavno iz Painterja izvoziti vse potrebne teksture za PBR (angl. Physically Based Rendering, senčenje na osnovi fizike). Program smo uporabili za teksturiranje našega modela nato pa vse narejene teksture prenesli v UE4.



Slika 2.2: Substance Painter, uporabniški vmesnik

## 2.3 Unreal Engine 4

Unreal Engine 4 je igralni pogon podjetja Epic Games [6]. V času pisanja te diplomske naloge je eden izmed bolj popularnih igralnih pogonov. Od konkurenčnih pogonov se razlikuje predvsem zaradi ponujenega novega načina programiranja, imenovanega Načrti (angl. Blueprints), ki nadomesti klasično kodiranje. Ponaša se tudi z izpopolnjenim animacijskim sistemom, lepšo privzeto osvetljavo in vizualnostjo ter njegovo vsestransko zmogljivostjo. Odprtokodnost pogona je dodatna prednost.

UE4 smo uporabili za prikaz delovanja našega izdelanega karakterja in izdelavo pokrajine.

Slika 2.3: UE4, uporabniški vmesnik

# Poglavje 3

# Predstavitev mrež in modifikatorjev

Modeliranje mrež se navadno začne z neko osnovno geometrijsko obliko, to je lahko navadna ploskev, kocka, cilinder... S pomočjo funkcionalnosti 3D oblikovalskih orodij nato oblikujemo večje, bolj kompleksne oblike.

Vsaka mreža je sestavljena iz treh osnovnih struktur: oglišč, robov in ploskev.

### Oglišča

Oglišča so najbolj osnovni sestavni del mreže. Vsako oglišče je svoja točka v 3D prostoru, ima torej svojo pozicijo.

### Robovi

Rob vedno povezuje dve oglišči med seboj z ravno črto. Z njimi povezujemo oglišča v ploskve.

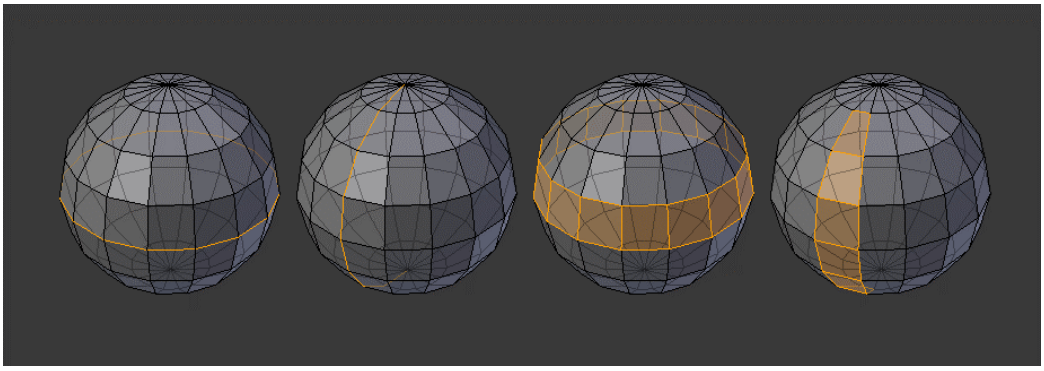
### Ploskve

Ploskve so uporabljene za dejansko površino objektov. Ko se objekt izrisuje, so ploskve tisto, kar nam je vidno. Definirane so kot površina med tremi (trikotnik), štirimi (štirikotnik) ali več oglišči (n-kotnik) in omejene z robovi.

Trikotniki so vedno planarni in tako preprosti za izračune. Po drugi strani pa se štirikotniki odlično deformirajo in so bolj priporočljivi pri animaciji in deljenju ploskev.

**Robne zanke**

Zanke robov in ploskev so nizi robov ali ploskev, ki skupaj tvorijo nepreki-njeno zanko, kot je prikazano na sliki [3.1]. Na prvih dveh sferah (gledano z leve) sta dve robni zanki, ki povezujeta oglišča tako, da ima vsako oglišče v zanki natanko dva soseda, ki nista v zanki in sta na nasprotnih straneh, ra-zen v primeru začetnega in končnega oglišča, ko se zanka ciklično ne zaključi.



Slika 3.1: Prikaz robnih zank

Robne zanke so pomemben koncept predvsem pri organskem modeliranju in animaciji, saj omogočajo učinkovito deformiranje in deljenje ploskev.

**Modifikatorji**

V Blenderju poznamo modifikatorje, to so avtomatske operacije, ki delujejo nad celotnim objektom na nedestruktiven način.

Pri oblikovanju naših modelov smo uporabili naslednje modifikatorje:

- Mirror, ki prezrcali celoten objekt preko izbrane osi,
- Multiresolution, ki razdeli vse ploskve (1 štirikotnik razdeli na 4 manjše),
- Solidify, ki ploskvam doda globino,
- Armature, ki omogoča deformiranje objektov z okostji.

# Poglavje 4

# Izdelava karakterja

## 4.1 Ideja in vpeljava v Blender

Preden začnemo oblikovati model, moramo imeti dobro zastavljeno in jasno idejo o tem, kaj sploh želimo narediti. Pomagamo si lahko z referenčnimi slikami, po katerih se ravnamo, svetovni splet nam je pri tem v veliko pomoč, lahko pa jih narišemo sami. Še posebej nam pridejo prav referenčne slike pri modeliranju človeškega telesa, saj je človeška anatomija dokaj kompleksna.

Slika 4.1: Anatomija človeškega telesa (Vir: `https://www.artstation.com/artwork/008nw`, [Dostopano: 24. 01. 2019])

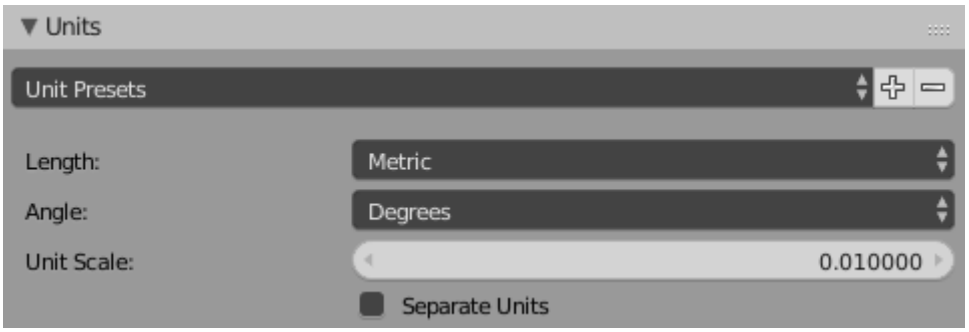
Večina orodij za 3D oblikovanje, med njimi tudi Blender, nam omogoča, da referenčne slike vpeljemo v delovno okolje in jih uporabimo kot osnovo, na kateri izdelamo obliko [4.2].

Slika 4.2: Prikaz referenčne slike in začetnega modela

## 4.2 Oblikovanje grobe oblike

Najprej moramo določiti, v kakšnih enotah bo predstavljen naš model. Privzeto je Blender okolje nastavljeno tako, da je ena enota enakovredna enemu metru. V obzir moramo vzeti igralno okolje UE4, kjer je ena enota enakovredna enemu centimetru. Če pustimo stvari, tako kot so, bomo imeli pri uvozu karakterja v UE4 težave, saj bo za faktor 100 manjši od želenega. Torej, navadna kocka velikosti 2x2 v Blenderju, bo v UE4 velika 2x2cm. To težavo bi lahko dokaj enostavno odpravili, če ne bi bil naš karakter animiran, in sicer s skaliranjem objektov za faktor 100. Ker pa je animiran, se pojavijo težave prav pri skaliranju, saj ima vsak ključni okvir (angl. keyframe) zapisane

položaje in rotacije kosti, ki vplivajo na končno pozo karakterja. Tako bi morali pri skaliranju animiranih modelov za faktor 100 vplivati tudi na vsak keyframe, kar pa v času pisanja te naloge v Blenderju še ni implementirano. Vseeno obstaja rešitev, ki se je moramo poslužiti preden začnemo s procesom animacije. V scenskih nastavitvah nastavimo metrično skalo na 0.01 [4.3]. To pomeni, da je sedaj vsaka Blender enota enakovredna 0.01m, torej enemu centimetru, kot je to v izbranem igralnem pogonu, UE4. V kolikor smo naš model oblikovali brez teh nastavitev in ga še nismo animirali, potem ga moramo le skalirati za faktor 100. V nasprotnem primeru pa nam ne preostane drugega, kot da se ponovno lotimo animacije, od začetka.




Slika 4.3: Nastavitev skale v Blenderju

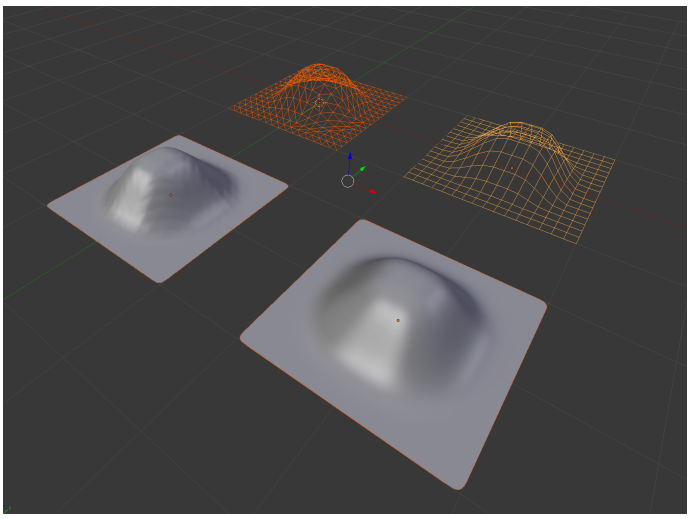
Našo mrežo, ki bo predstavljala človeško telo, lahko sestavimo na mnogo različnih načinov. V tej diplomski nalogi smo izbrali način, v katerem najprej oblikujemo grobo obliko, potem pa izdelamo detajle. Pri tem je treba biti pozoren, da je naša mreža, kjer je le mogoče, sestavljena iz štirikotnikov. Večina igralnih pogonov, tudi UE4, vse mreže, ki jih uvozimo, triangulira, kar pomeni, da razdeli vse poligone, ki imajo več kot 3 stranice, v trikotnike. Vseeno pa je delo s trikotniškimi ali večkotniškimi mrežami v 3D oblikovalskih orodjih precej omejeno.

Nekaj prednosti dela z mrežami, sestavljenimi iz štirikotnikov:

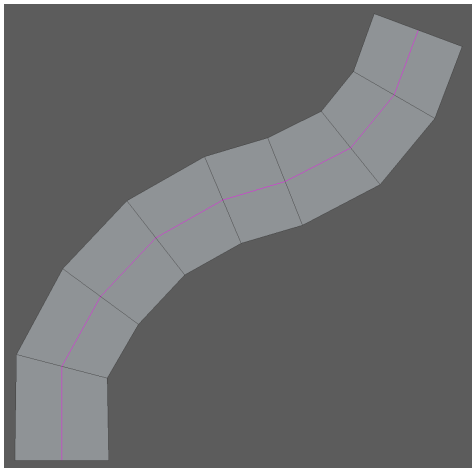
- z njimi lahko dosežemo pravilno urejeno topologijo (topologija je ure-

ditev poligonov, ki sestavljajo mrežo, več o topologiji v poglavju Topologija [4.4.1]) našega modela,

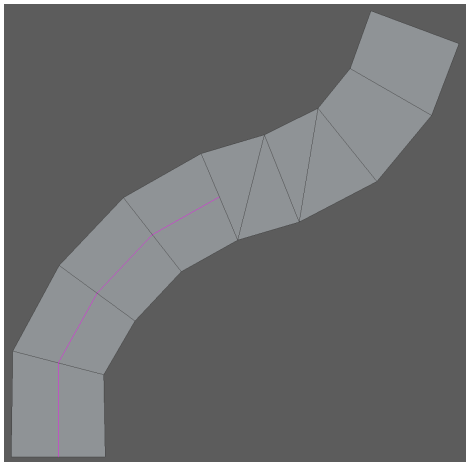
- lažje dodajanje in odstranjevanje robnih zank [4.5], s katerimi lahko bolje definiramo določeno področje,
- boljša deformacija v primeru animacij,
- ob uporabi delitev ploskev (angl. subdivide), štirikotniki ne povzročajo artefaktov, kot je to pogosto pri trikotnikih in večkotnikih [4.4].



Slika 4.4: Prikaz deljenih ploskev, desno spodaj štirikotniki, levo zgoraj trikotniki.



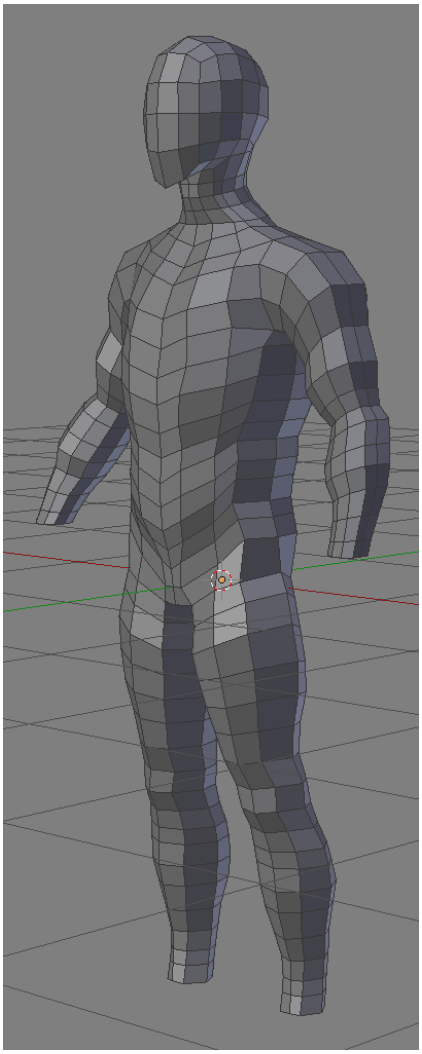
Slika 4.5: Primer dobro zastavljene geometrije in poteka zank, z vijolično barvo je označena zanka, ki jo lahko dodamo geometriji.



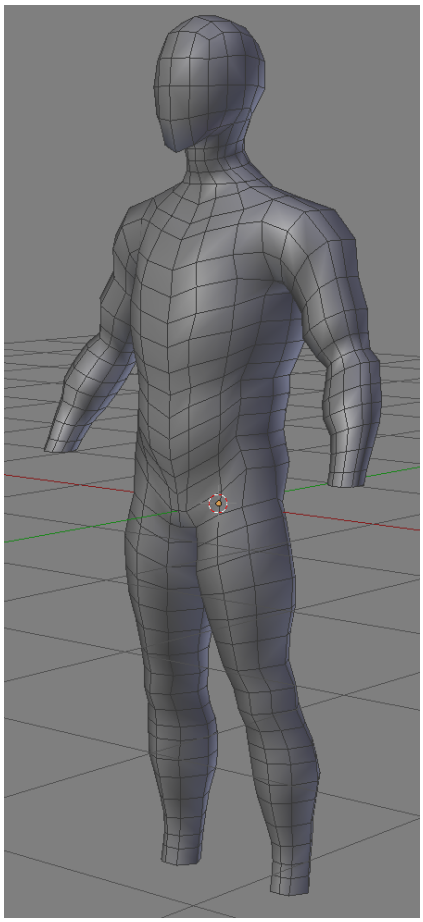
Slika 4.6: Primer slabo zastavljene geometrije, zanka se ne more zaključiti.

Našo prvo obliko človeškega telesa smo dobili tako, da smo začeli z navadno kocko, katero smo z izvlečenjem (angl. extrusion, način ustvarjanja dodatne geometrije z izvlekom obrisa 2D predmetov), translacijo, rotacijo in skaliranjem oglišč in poligonov, oblikovali približno po referenčni sliki. Tako smo dobili grobo silhueto z nizkim številom poligonov (1884 trikotnikov). Prav zato je tudi površina modela zelo ostra, ne gladka, saj je vsak poligon

velik in usmerjen v svojo smer in posledično drugače odbija svetlobo kot sosednji [4.7]. To lahko rešimo tako, da uporabimo možnost gladkega senčenja (angl. smooth shading), ki v resnici ne spremeni geometrije, pač pa samo interpolira normale in s tem daje iluzijo gladkih površin [4.8].



Slika 4.7: Model brez gladkega senčenja



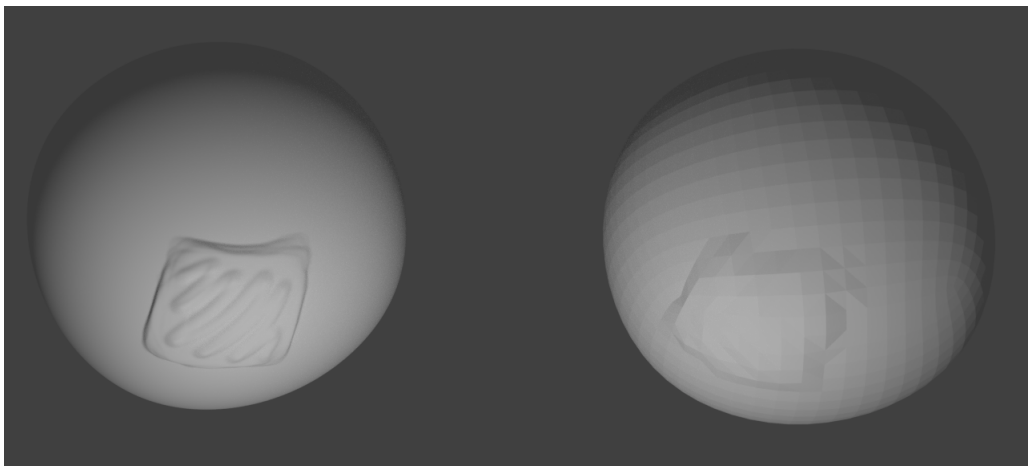
Slika 4.8: Model z gladkim senčenjem

## 4.3 Kiparjenje

Ko imamo osnovno obliko zastavljeno, nas čaka, odvisno od tega, na kakšno stopnjo realizma ciljamo, kiparjenje (angl. sculpting). V kolikor bi izdelovali karakter za low-poly igro (to je igra, v kateri so 3D modeli predstavljeni z nizkim številom poligonov), bi ta korak lahko preskočili.

Kiparjenje je zelo podobno oblikovanju gline iz resničnega življenja. Namesto da manipuliramo s posameznimi oglišči in poligoni, uporabljamo tako imenovane „čopiče“ (angl. brush), to so orodja, ki vplivajo na večjo površino mreže hkrati. V Blenderju imamo na voljo veliko število različnih orodij za kiparjenje, ki imajo različne učinke na mrežo. Najbolj osnovna med njimi so orodje za ustvarjanje gladkih površin, orodje za dodajanje in odstranjevanje gline in orodje za ustvarjanje pregibov.

Vendar pa ne moremo učinkovito oblikovati našega trenutnega modela, saj ima premalo geometrije oziroma poligonov, s katerimi bi lahko delali. Zato uporabimo modifikator Subdivision Subsurface, ki razdeli ploskve.



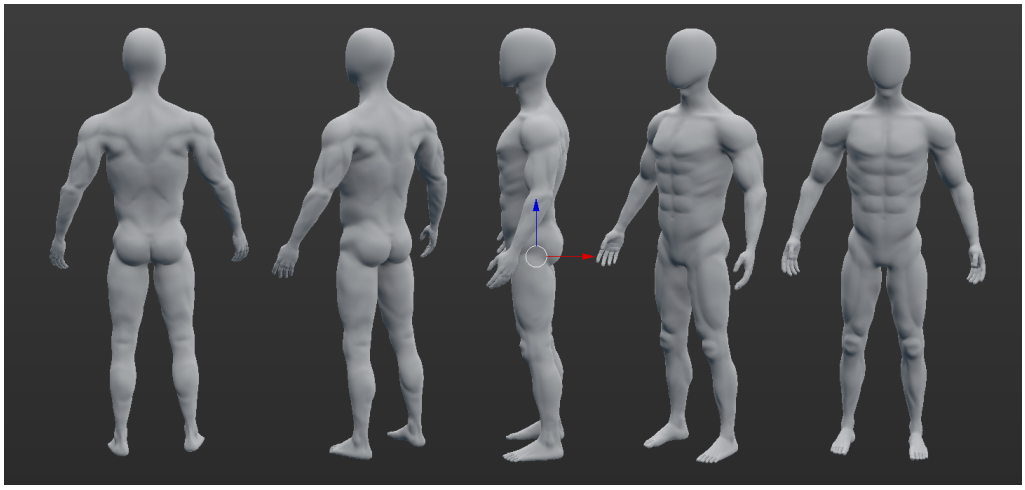
Slika 4.9: Prikaz modifikatorja Multiresolution, ki ima enak učinek kot Subdivision Surface.

Na zgornji sliki [4.9] sta dva enaka objekta, le da so na levem objektu

ploskve razdeljene 9-krat, medtem ko so na desnem le 4-krat. Tako je levi objekt sestavljen iz približno 3.000.000 trikotnikov in ima vidno bolj prefinjene detajle, desni pa iz 3.000 trikotnikov in so detajli zelo grobi, skorajda neopazni.

Pri kiparjenju je pomembno, da ne začnemo takoj dodajati najmanjših detajlov. Boljši videz in končno dodelanost dobimo s postopnim obdelovanjem. Najprej obdelamo celotno silhueto, da se čimbolj prilega končni željeni obliki modela, nato pa preidemo na večje detajle, v našem primeru so to večji sklepi (komolci, kolena). Ko smo zadovoljni z izgledom, se spustimo v najdrobnejše detajle, kako daleč gremo z njimi, pa je odvisno od naše želje po realizmu in od časovnega okvirja, v katerem moramo model izdelati.

Slika 4.10: Model človeškega telesa po nekaj fazah kiparjenja. Dlani in stopala še manjkajo, prav tako noge še niso podrobno obdelane.



Slika 4.11: Model človeškega telesa po končanem kiparjenju. Celoten model ima v tej fazi 729.770 trikotnikov.

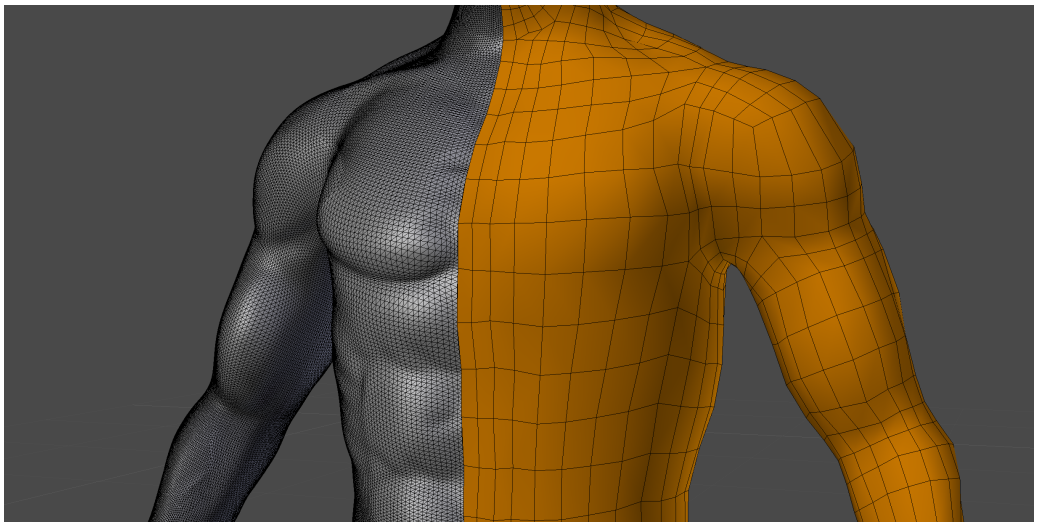
## 4.4 Retopologija

Preden so se razvili postopki retopologije, so morali 3D oblikovalci začeti in končati svoje modele na način, ki je opisan v poglavju Oblikovanje grobe oblike, le da ni bilo dovolj, da so na tak način oblikovali samo grobo obliko, pač pa so morali z dodajanjem, izvlečenjem, rotacijo in translacijo oglišč in poligonov doseči čimvečji nivo podrobnosti. Razlog za to je v tem, da so modeli, izdelani na način, opisan v poglavju Kiparjenje, prezahtevni celo za najsodobnejše grafične kartice, saj morajo za vsak okvir (angl. frame), ki se v moderni računalniški igri posodobi približno 60-krat na sekundo, ponovno izrisati, v našem primeru, 729.770 trikotnikov. In to velja samo za naš karakter, v igri pa jih je na zaslonu lahko izrisanih mnogo več, da ne omenjamo vseh ostalih objektov v okolici.

Zakaj je torej kiparjenje uporabno? Zato, ker lahko na tak način dosežemo večji nivo realizma, ustvarimo podrobnejše detajle, pri tem pa ni potrebno razmišljati o trikotnikih, štirikotnikih, številu vseh poligonov in ostalih stvareh, povezanih s topologijo. Tak način je 3D oblikovalcem bolj domač.

Težava se pojavi, ko želimo oblikovani model animirati in uporabiti v igralnem okolju, saj je, kot že prej omenjeno, prezahtevna za grafične kartice. Tu nastopi postopek retopologije.

Retopologija je izraz za postopek, v katerem oblikujemo popolnoma novo mrežo poligonov, ki se čim bolj natančno prilega skiparjenemu modelu in ima ustrezno topologijo.

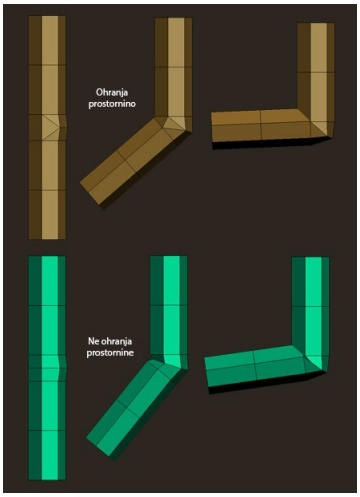


Slika 4.12: Na levi strani je prikazan skiparjen model z visokim številom poligonov, na desni pa retopologiran model.

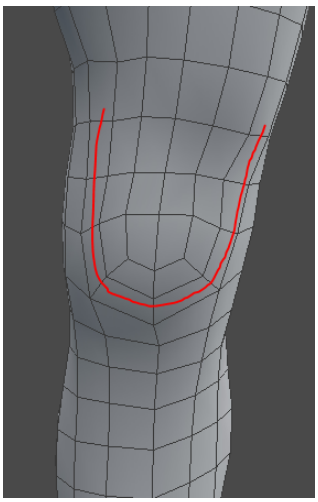
### 4.4.1 Topologija

Topologija je način urejenosti poligonov, ki sestavljajo mrežo. Priporočljivo je, da vsebuje večinoma štirikotnike. Dobra topologija je osredotočena na zagotavljanje ustrezne silhuete, dobro definira robne zanke in potek le-teh, minimizira raztegovanje poligonov in omogoča lažje označevanje šivov, ki so razloženi v poglavju UV-mapiranje [4.5]. Grobo pravilo, po katerem se lahko ravnamo, je, da ne ustvarjamo dodatne geometrije tam, kjer ni potrebna. Na ravnih površinah, brez posebnih detajlov, je lahko geometrija dokaj redka. Področja, kjer imamo veliko drobnih podrobnosti, pa zahtevajo bolj gosto geometrijo, oziroma večje število poligonov. Prav tako je bolj gosta geometrija zaželjena tam, kjer pričakujemo deformacije. Končno število poligonov bo vplivalo na hitrost izrisovanja. V postopku naše retopologije smo dosegli število 10.190 [4.13].

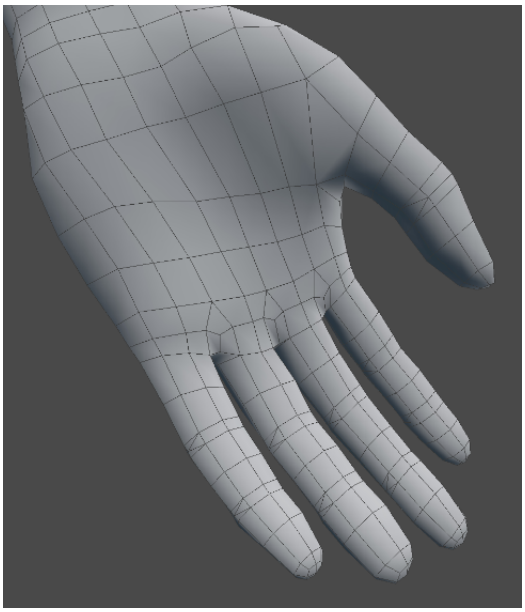
Slika 4.13: Končni model po retopologiji



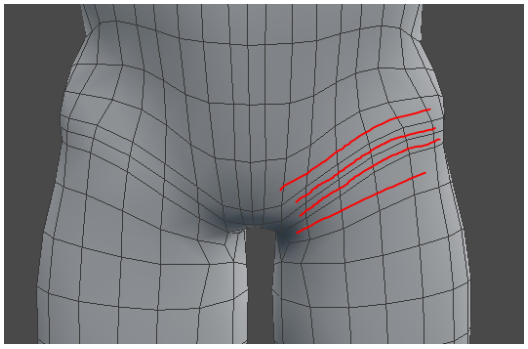

Slika 4.14: Dobra topologija ob deformacijah ohranja svojo prostornino, medtem ko je slaba ne ohranja (Vir: `http://wiki.polycount.com/wiki/Limb_Topology`, [Dostopano: 25. 01. 2019])



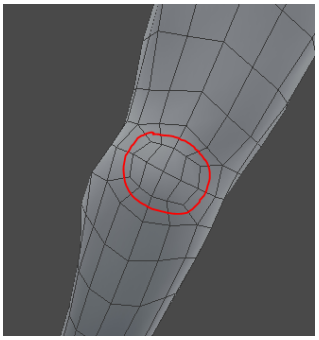
Slika 4.15: Urejena topologija okoli kolena



Slika 4.16: Urejena topologija roke



Slika 4.17: Urejena topologija okoli bokov



Slika 4.18: Urejena topologija komolca

Ko primerjamo število poligonov z ostalimi modeli, moramo vedeti, da so te številke relativne in je njihova sprejemljivost odvisna od tega, za kakšno igro gre. Če v naši igri nastopa le nekaj karakterjev, potem si najverjetneje lahko privoščimo visoko število poligonov zanj, ki pa se lahko giblje od 40.000 do 100.000 [2]. To je le groba ocena. Ko pa gre za na primer strateško igro, pri kateri se na zaslonu izrisuje po 500 ali več vojščakov, pa moramo biti s številom poligonov za vsakega izmed njih skromni, tja do 5.000 [3].

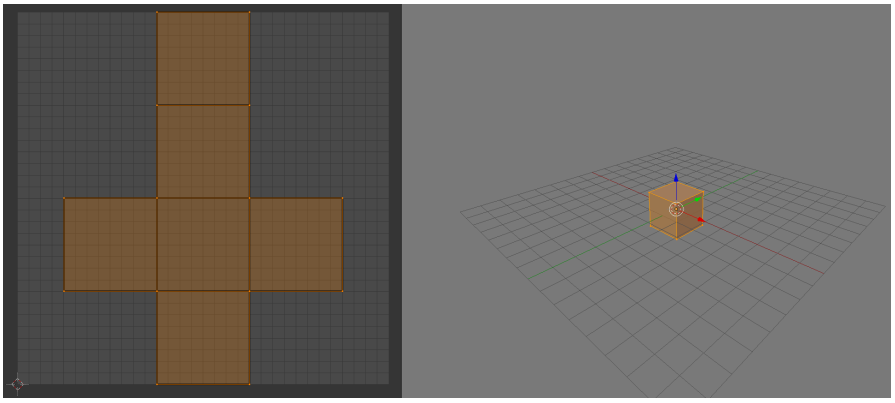
## 4.5 UV-mapiranje

Ko govorimo o teksturah, mislimo na 2D slike. Da na naših modelih dosežemo višjo stopnjo realizma, uporabljamo različne tipe tekstur, ki jim pravimo mape. Difuzna mapa, ki shranjuje barvne informacije o modelu, je le ena izmed njih. To je lahko navadna fotografija česarkoli, lahko pa jo zgeneriramo sami z ročnim barvanjem naših modelov.

Da pa lahko uspešno povezujemo barvne piksle iz 2D teksture s posameznimi oglišči modela, ki je predstavljeno v 3D, moramo našo 3D mrežo (X, Y, Z) razgrniti na 2D (X, Y, oziroma kot bomo kasneje videli, U, V) podlago ali sliko. S tem ima vsak pixel na sliki točno določeno oglišče v mreži.

Najlažje si UV mapiranje predstavljamo na primeru kartonaste kocke. Kocka je 3D objekt, enako kot modeli v Blenderju.

Če s škarjami razrežemo kocko, jo lahko položimo na ravno podlago ali mizo [4.19]. Recimo, da od leve proti desni poteka U os, zgoraj-spodaj pa V os. Tako je kocka sedaj v 2D sistemu (UV). Za teksturne koordinate uporabljamo UV osi, namesto XY, ki sta vedno uporabljeni poleg Z osi v klasičnem 3D sistemu. Ko kocko ponovno sestavimo iz 2D v 3D, se vsaka UV točka prenese v XYZ točko na kocki.



Slika 4.19: Primer razgrnitve 3D kocke na 2D podlago

V Blenderju imamo popolnoma proste roke pri razrezovanju naših objek-

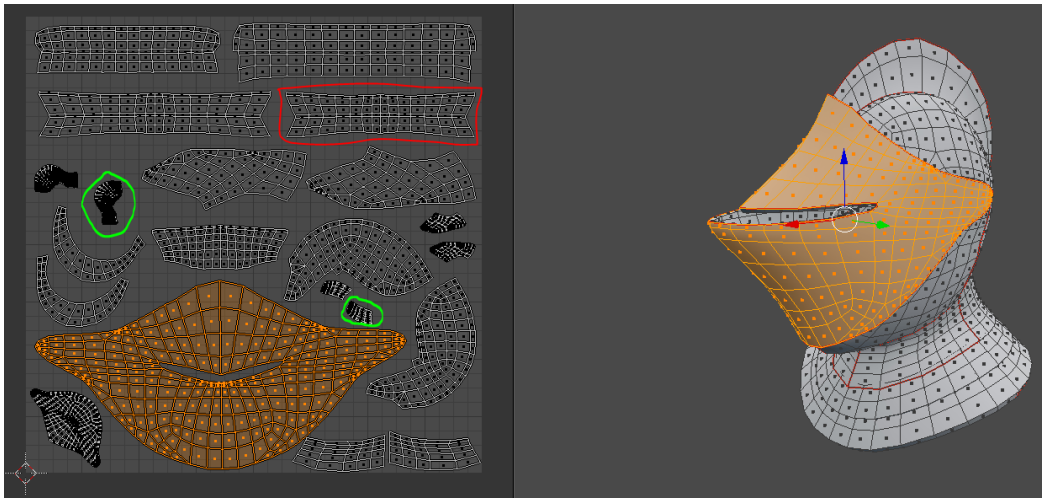
tov. Za najpreprostejše objekte imamo na voljo nekaj različnih algoritmov, ki avtomatsko razgrnejo površino izbranega objekta (mrežo), to so na primer kockasto, sferično ali cilindrično lepljenje. Za bolj mehanske objekte ali arhitekturo lahko uporabimo Blenderjev algoritem imenovan Smart UV Project. Vendar pa bomo deležni najboljših rezultatov, če se razrezovanja lotimo sami, ročno.

Ročno se razrezovanja v Blenderju lotimo z označevanjem šivov (angl. seam). Pri šivanju se šiv nahaja tam, kjer sta dva konca blaga zašita skupaj. Dobra analogija za označevanje šivov je lupljenje pomaranče. Z nožem zarežemo v lupino in jo olupimo. Ti rezi so enaki šivom v naši 3D mreži in vsak košček lupine, ki ga odrežemo, ponazarja svoj UV otok (angl. UV island).

Pri označevanju šivov moramo biti pozorni na raztegovanje naše mreže v UV prostoru. Več kot je šivov ali UV otokov, manj raztegovanja je prisotnega, obenem pa to ni najbolj idealno za kasnejše teksturiranje, saj je šive s teksturami težko zakriti. Zato moramo poiskati pravo ravnotežje med številom šivov in količino raztezanja. Dobro načelo je, da označujemo šive predvsem na mestih, kjer bodo manj vidni. To je lahko na spodnji strani rok, notranji strani nog ipd.

Ko razrezujemo zrcalne objekte, na primer obe roki, in jih imamo namen teksturirati enako, je pametno, da prihranimo UV prostor tako, da obe roki odvijemo na isti UV prostor. To pomeni, da se bodo UV otoki obeh rok prekrivali in bodo oglišča obeh rok prenesena na isti prostor v teksturi.

Pomembno je, da imamo ob razrezovanju mreže v mislih teksturiranje. Kje bo naš model imel največ detajlov? Kateri deli bodo zakriti in kateri na najbolj izpostavljenem mestu? Vse to nam pride prav, če vemo, da večji kot je UV otok, več detajlov lahko narišemo na ta del teksture. To pomeni, da je ključnega pomena, da tiste UV otoke, ki so pomembnejši od ostalih, povečamo in jim dodelimo več UV prostora in s tem možnost podrobnejših detajlov, medtem ko tiste, ki bodo komaj vidni, zmanjšamo [4.20].



Slika 4.20: Z rdečo barvo je obkrožen eden od UV otokov. Z zeleno barvo je obkrožen UV otok, ki v modelu predstavlja notranjost čelade in je zato pomanjšan. Z oranžno barvo je označen UV otok, ki v modelu čelade predstavlja najpomembnejšo površino in je temu primerno največji.

## 4.6 Izdelava okostja

Okostje ali armatura (angl. armature) je v Blenderju predstavljena kot množica posameznih kosti (angl. bones), ki so med seboj povezane, odvisne. Vsako izmed njih lahko premikamo, obračamo in s tem vplivamo na vse ostale odvisne kosti, pri tem pa po možnosti še deformiramo mrežo, na katero se armatura sklicuje.

Brez ustrezno izdelane armature, ki pravilno deformira dele mreže, ne moremo začeti s postopki animiranja. V našem primeru je potrebno izdelati okostje za človeka. Pri tem se je dobro držati ustreznega poimenovanja kosti (izbrali smo angleško poimenovanje, lahko pa bi se odločili za slovensko):

- glavna kost je imenovana „hips“,
- kosti hrbtenice so poimenovane „spine1“ in „spine2“,
- vrat in glava sta „neck“ in „head“,
- ključnici sta poimenovani „clavicle.L“ in „clavicle.R“,
- kosti rok in nog so „upperArm.L“, „lowerArm.L“, „upperLeg.L“, „lowerLeg.L“ in podobno za desno stran,
- dlani sta „hand.L“ in „hand.R“, prsti pa so poimenovani po sistemu „index1.L“, „index2.L“, „index3.L“ in tako naprej za vse ostale prste, z zaključkom „.R“ na desni strani,
- kosti v stopalih sta poimenovani „foot.L“ in „toes.L“ ter s črko R na desni strani.

Zgoraj naštete so kosti, ki našo mrežo dejansko deformirajo. Da pa ni potrebno vsake kosti manipulirati ročno, imamo še dodatne kontrolne kosti, ki samo spreminjajo položaje deformacijskih kosti in ne vplivajo direktno na mrežo.

Zaporedje poteka dela si sledi nekako takole:

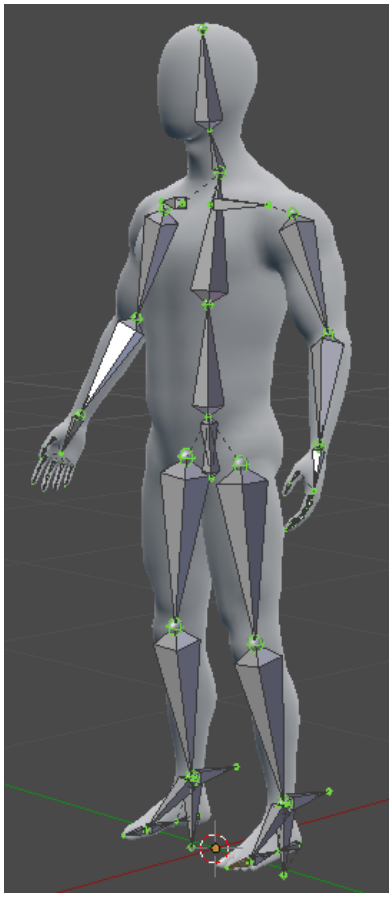
- začnemo z eno samo kostjo imenovano „hips“, ki se nahaja v predelu bokov našega karakterja,
- iz te glavne kosti z izvlečenjem (angl. extrusion) izdelamo nove. Pri tem pazimo na odvisnost med kostmi. Tako je na primer naša glavna kost neodvisna od vseh ostalih, medtem ko je kost „spine1“ odvisna od kosti „hips“, „spine2“ pa od „spine1“,
- tako izdelamo še ostale kosti za roke, noge in glavo. Blender se ponaša s funkcijo, s katero lahko izdelamo le eno stran okostja, na primer levo, potem pa celotno okostje preslikamo še na drugo. Da to uspe, se moramo držati konsistentnega poimenovanja kosti, navedenega zgoraj,
- ko imamo deformacijske kosti ustrezno postavljene, se lotimo kontrolnih. Mednje spadajo tudi kosti, ki sestavljajo sistem inverzne kinematike.

### 4.6.1 Inverzna kinematika

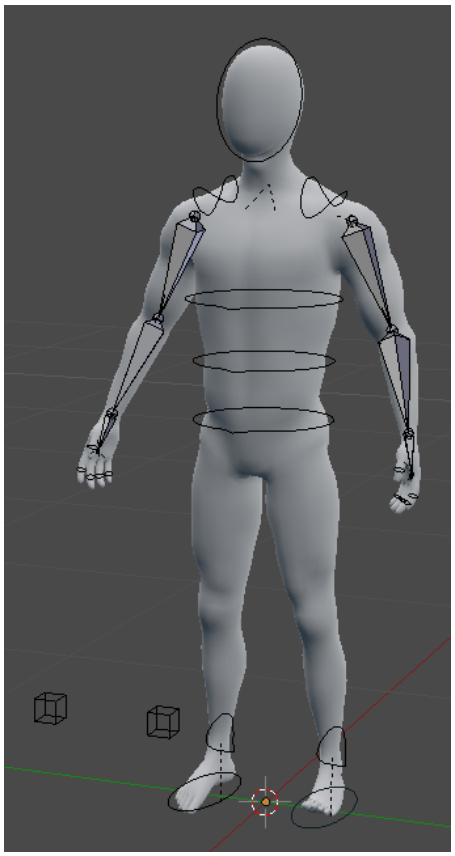
IK je kratica za inverzno kinematiko, ki nam olajša animacijski proces in pomaga pri kompleksnejših animacijah.

Inverzna kinematika nam omogoča, da zadnjo kost v verigi kosti premaknemo na željeno mesto, ostale pa ji pravilno sledijo. Predstavljajmo si, da želimo levo stopalo premakniti na poljubno mesto. Z nastavljenim IK sistemom lahko preprosto izberemo kost „foot.L“ in jo premaknemo, ostale kosti noge („lowerLeg.L“ in „upperLeg.L“) pa ji sledijo. Brez IK sistema bi morali najprej premakniti zgornji del noge nato spodnji del in na koncu še samo stopalo.

Na sliki levo [4.21] je vidna celotna armatura. Na desni sliki [4.22] pa je armatura, kjer so deformacijske kosti skrite, saj z njimi direktno nimamo opravka, razen v primeru rok, kjer bi lahko uporabili inverzno kinematiko, a se zaradi preprostosti zanjo nismo odločili.



Slika 4.21: Celotna armatura človeškega telesa

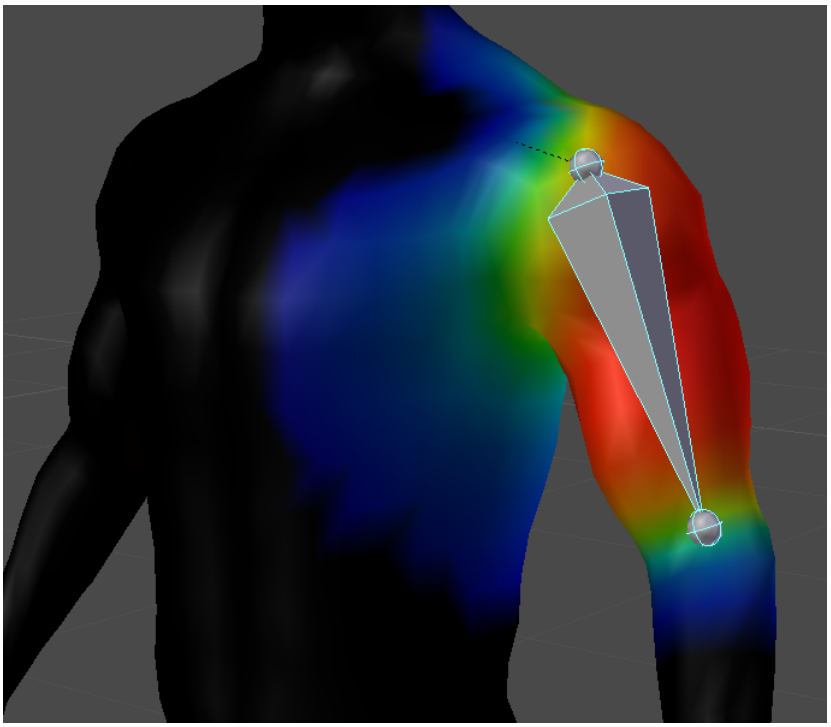


Slika 4.22: Vidne samo kontrolne kosti

### 4.6.2 Določanje vpliva okostja

Na tem mestu imamo urejeno armaturo z delujočo inverzno kinematiko, vendar pa se ob premikanju kosti model še ne deformira. Programu moramo namreč najprej sporočiti, kako vsaka izmed kosti sploh vpliva na model. Temu procesu rečemo barvanje vpliva (angl. weight painting).

Prvi korak, ki ga naredimo, je, da v Blenderju izberemo naš model in obenem še armaturo ter uporabimo funkcijo avtomatskega barvanja vpliva [4.23].



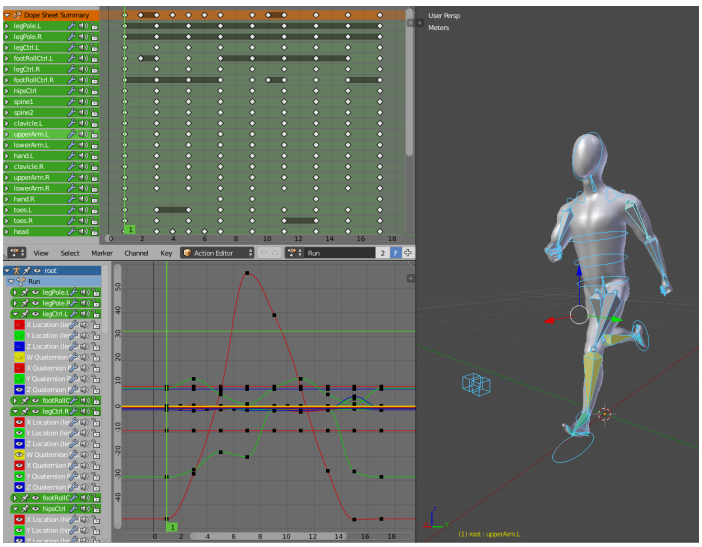
Slika 4.23: Prikaz vpliva kosti „upperArm.L“ na model. Vpliv je največji, kjer je model obarvan rdeče, in najmanjši, kjer je obarvan temno modro.

V tej fazi premikamo vsako izmed kosti v armaturi in preverimo, ali je avtomatsko barvanje ustrezno določilo vplive posameznih kosti. V kolikor so potrebni popravki, lahko to storimo z ročnim barvanjem, kjer uporabljamo, podobno kot pri kiparjenju, posebne čopiče.

## 4.7 Animacija

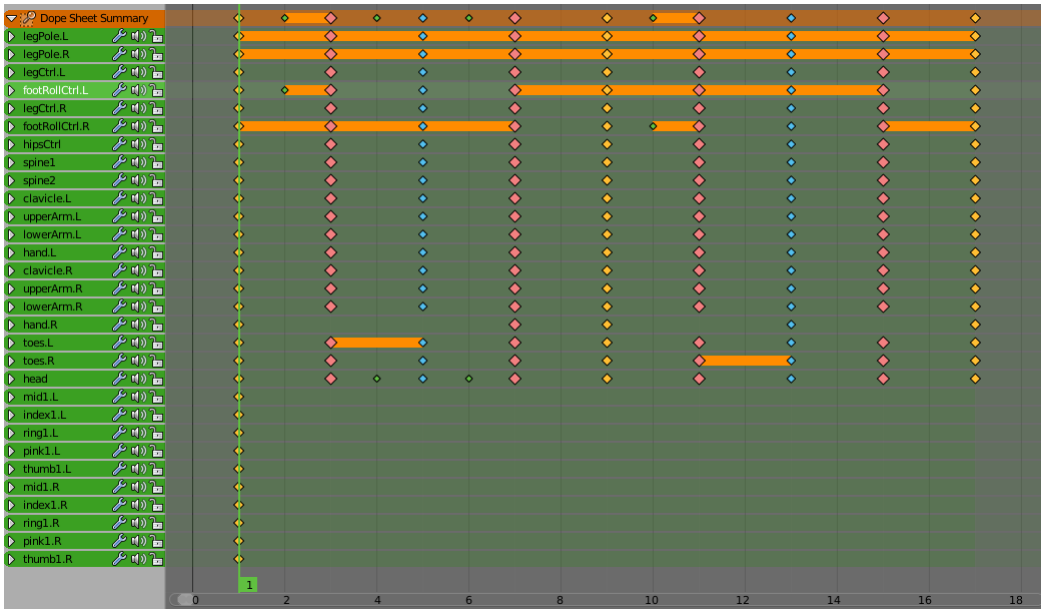
Animacija je premikanje modela v odvisnosti od časa. V našem primeru smo animirali nedejavno stanje, hojo, tek, skok, počep in dve obliki napada.

V Blenderju je animacija zaporedje različnih okvirjev, ki si sledijo na časovnici [4.24]. Na tem mestu je potrebno razložiti pojem okvirja v animaciji (angl. frame). Okvir je točka na časovnici animacije, v kateri je shranjena pozicija in rotacija vsake kosti v naši armaturi. Namen okvirjev, je da omogočajo interpolirane animacije. To pomeni, da lahko modelu - za primer vzemimo kocko - v prvem okvirju določimo horizontalno pozicijo 0, v desetem pa horizontalno pozicijo 20. S pomočjo interpoliranja računalnik sam določi pozicijo na vmesnih okvirjih, na primer v petem določi pozicijo 10, odvisno od tega, kakšen način interpoliranja izberemo. Pri našem animiranju karakterja je potrebno interpolirati pozicijo in rotacijo po vseh treh oseh (X, Y in Z).



Slika 4.24: Levo zgoraj je prikazana časovnica animacije teka v Blenderju, kjer je za vsako kost vstavljen okvir. Spodaj levo pa so izrisane krivulje, ki predstavljajo interpolacije med temi okvirji.

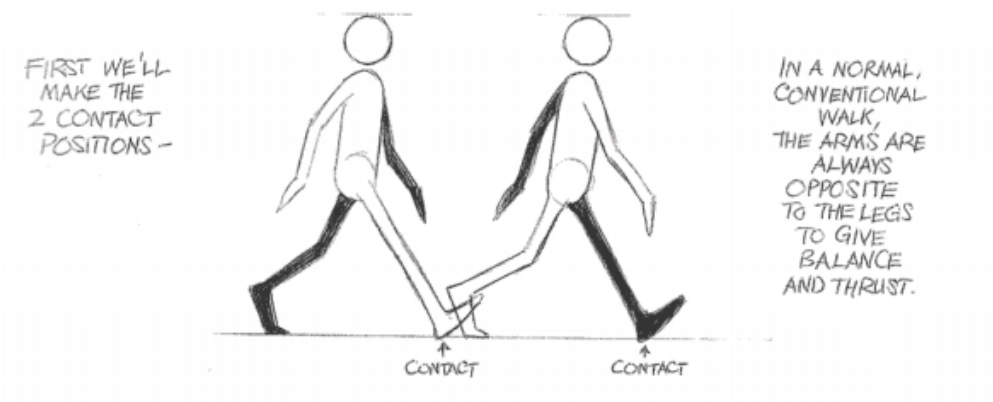
Pri animiranju v resnici samo postavljamo model v različne poze in te poze shranjujemo na časovnico, kjer jim rečemo okvirji. Za postopek dela lahko izberemo več načinov, v našem primeru smo izbrali najbolj organiziran potek dela, imenovan angleško „pose to pose“ (od poze do poze) [7]. Pri tem je prvi korak, da vstavimo v animacijo ključne okvirje (angl. keyframe). To so najpomembnejše poze, ki zastavijo nadaljnji potek animiranja in predstavljajo ključne trenutke v animaciji. Za ključnimi okvirji je potrebno vstaviti okvirje, ki jim pravimo ekstremi. Ekstremi so poze, v katerih animiran model doseže najbolj ekstremne pozicije. Tretji tip okvirjev, ki jih vstavimo, so prehodni okvirji (angl. „passing pose“ ali „breakdown“). Na tem mestu je animacija v zadnji fazi, kar pomeni, da jo moramo le še pozorno dodelati in popraviti napake, da dobimo gladko, tekoče gibanje. Pri tem si pomagamo z vstavljanjem vmesnih okvirjev tam, kjer opazimo napake.



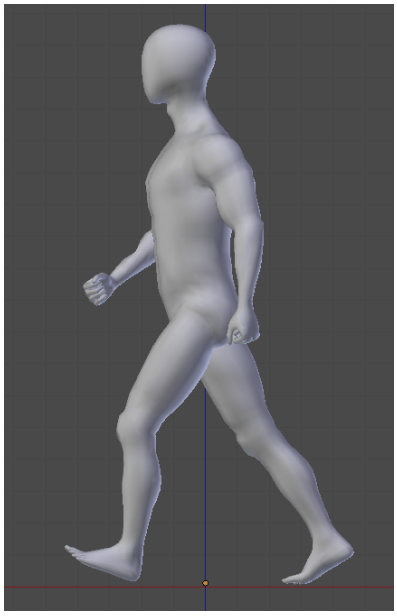

Slika 4.25: Oranžna – ključni okvirji ali kontaktne poze. Rdeča – ekstremi. Modra – prehodne poze. Zelena – vmesni popravki.

### 4.7.1 Hoja

Pri hoji najprej zastavimo ključne okvirje v pozi, ko je naš karakter tik pred izvajanjem koraka. Z drugimi besedami ji rečemo kontaktna poza. Z obema nogama se še dotika tal [4.26]. Tu potrebujemo v resnici dve pozi. V eni je spredaj leva noga, v drugi desna.

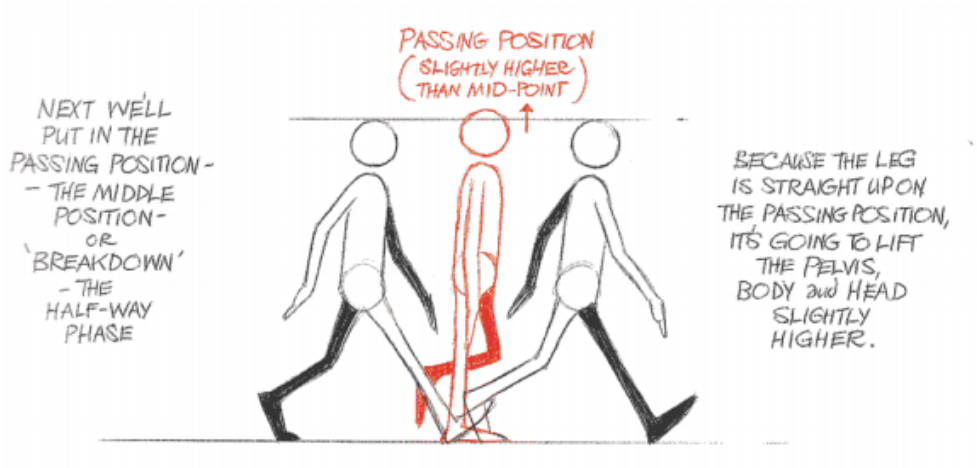



Slika 4.26: Skica kontaktnih poz (Vir: The Animator's Survival Kit [7])



Slika 4.27: Kontaktna poza

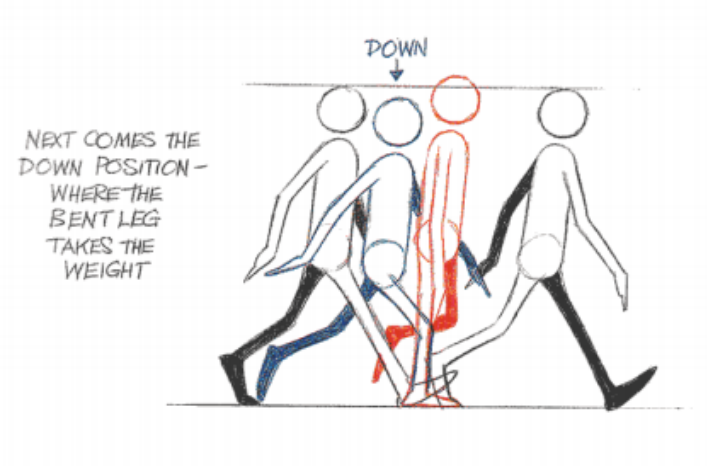
Druga poza je prehodna poza, v kateri se nogi zamenjata, tista, ki je bila zadnja, je sedaj v zraku in se pomika naprej, sprednja pa se pomika nazaj in je stojna noga [4.28]. Celoten karakter je v tej pozi zaradi skoraj iztegnjene stojne noge dvignjen.




Slika 4.28: Skica prehodne poze (Vir: The Animator's Survival Kit [7])

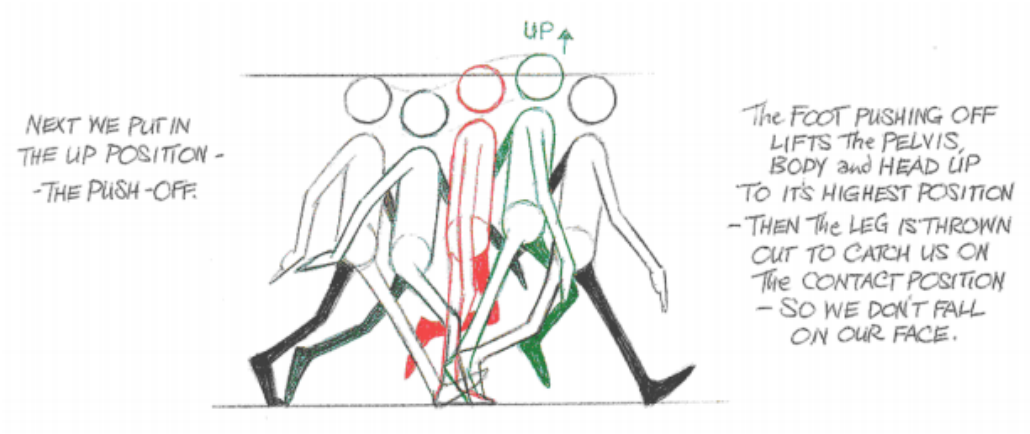
Slika 4.29: Prehodna poza

Preostaneta nam še dve pozi, ekstrema. Prvi ekstrem je, ko je karakter v najnižji poziciji med hojo [4.30], drugi pa, ko je v najvišji [4.32].




Slika 4.30: Skica prvega ekstrema (Vir: The Animator's Survival Kit [7])

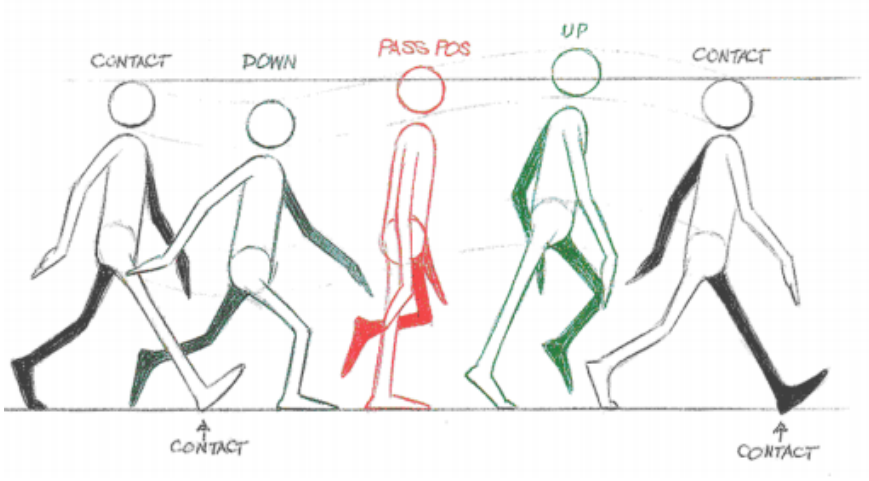
Slika 4.31: Prvi ekstrem




Slika 4.32: Skica drugega ekstrema (Vir: The Animator's Survival Kit [7])

Slika 4.33: Drugi ekstrem

Na tem mestu imamo vse glavne poze: prva kontaktna, prvi ekstrem, prehodna, drugi ekstrem, druga kontaktna. Skupaj sestavljajo en korak. Za celoten cikel hoje potrebujemo še drugi korak, ki vsebuje enake poze, le obratne. Na sliki [4.34] je prikazan desni korak (desna vodilna noga), potrebujemo še levega (leva vodilna noga).



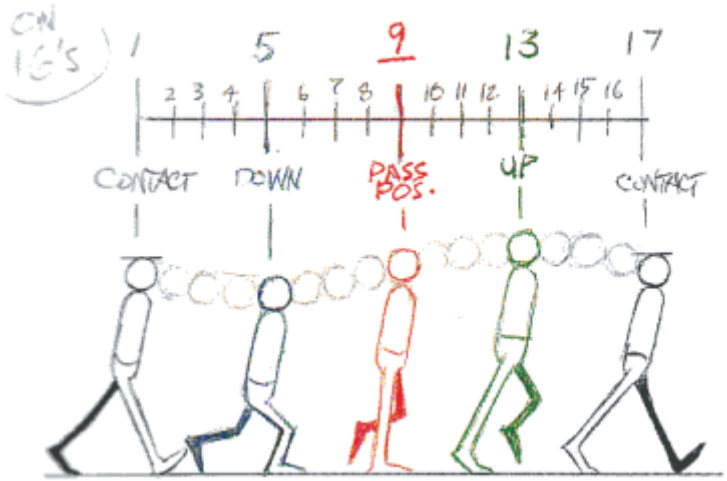

Slika 4.34: Skica vseh glavnih poz (Vir: The Animator's Survival Kit [7])

Preden pa se lotimo drugega koraka, moramo ustrezno določiti hitrost izvajanja našega cikla hoje. Hitrost animacije oziroma hitrost hoje našega karakterja je odvisna od števila prikazanih okvirjev na sekundo, ki jih prikazujemo na zaslonu. Angleško se izrazu reče „frames per second“. V večini računalniških iger si želimo doseči vrednost 60 FPS (angl. kratica za „frames per second“), vendar pa dejanska vrednost vedno niha. V kolikor je scena, ki jo izrisujemo, zelo zahtevna, grafična kartica ne zmore vedno ohranjati konstantnega izrisovanja. To bi pomenilo, da bi naš karakter ob različnih vrednostih FPS hodil različno hitro. K sreči večina igralnih pogonov, tudi UE4, učinkovito prilagaja hitrost izvajanja animacij glede na FPS, zato nas v procesu animiranja kasnejše nihanje te vrednosti ne skrbi. Vseeno pa si moramo izbrati določeno hitrost osveževanja, pri kateri bomo animirali naš karakter. V industriji se animatorji pogosto odločijo za vrednost 24 FPS, ki je nekako standardna v času pisanja te diplomske naloge. Ko imamo določeno

hitrost osveževanja (24 FPS), lahko določimo tudi hitrost animacije, v našem primeru hoje.

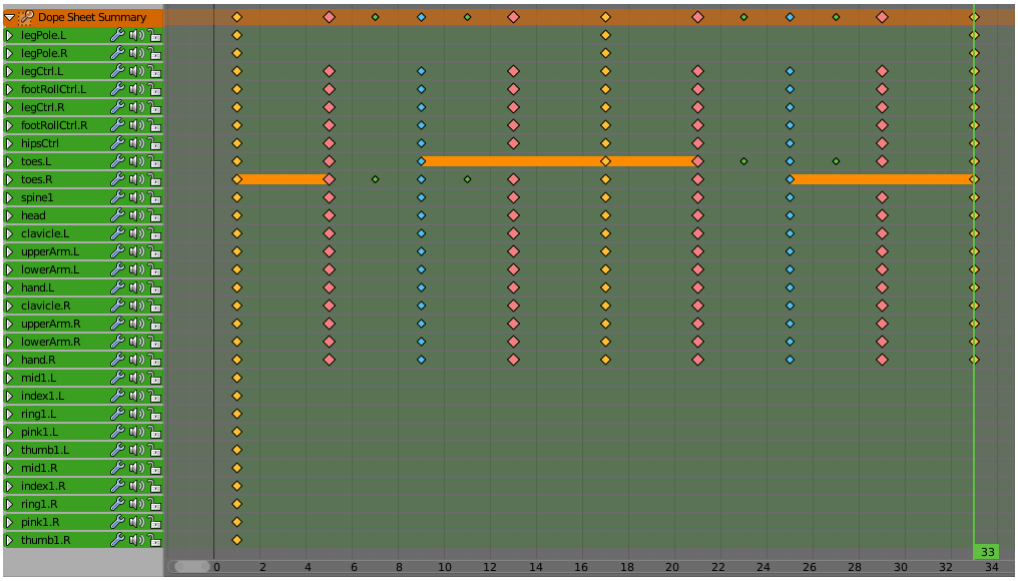
Hitrost naše animacije je torej odvisna od števila okvirjev. Na primer, animacija dolga 24 okvirjev, se bo izvajala natanko 1 sekundo, animacija dolga 48 okvirjev pa 2 sekundi.

Za hojo smo izbrali trajanje enega koraka 16 okvirjev (2/3 sekunde), pri tem je 17. okvir enak prvemu [4.35]. Dva koraka – 1 cikel – pa trajata 32 okvirjev (4/3 sekunde).




Slika 4.35: En korak traja 16 okvirjev, 17. je enak prvemu zato ga ne štejemo. Enakomerna porazdelitev okvirjev (Vir: The Animator's Survival Kit [7])

Na tej točki je hitrost našega cikla hoje definirana, kar pomeni, da lahko naše zgrajene poze vstavimo na ustrezna mesta na časovnici [4.36].

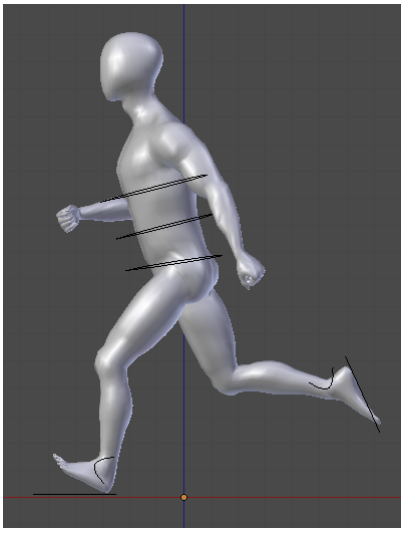



Slika 4.36: Celoten cikel hoje traja 32 okvirjev. Druga polovica okvirjev je enaka prvi, le da je obratna. V prvi polovici je vodilna leva noga, v drugi desna.

### 4.7.2 Tek

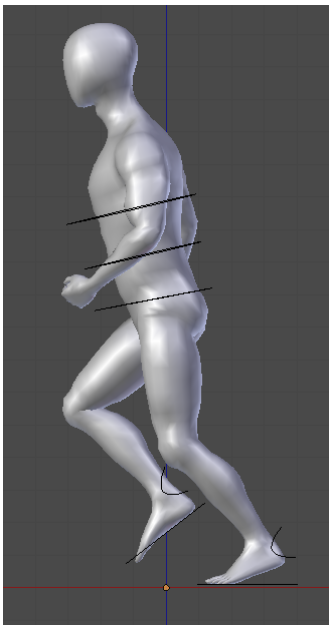
Animacija teka je zelo podobna animaciji hoje, le da se celoten cikel izvaja hitreje, naš karakter pa je v bolj agresivnih pozah in v nekaterih okvirjih celo v zraku. Celoten cikel se bo izvedel v 16 okvirjih, torej dvakrat hitreje kot cikel hoje.

Najprej zastavimo kontaktno pozo in jo vstavimo na 1. in 9. mesto, pri čemer je na 9. mestu obratna.



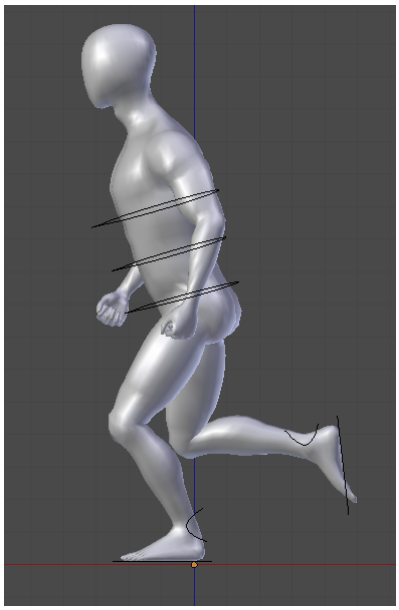
Slika 4.37: Kontaktna poza

Druga poza je prehodna kjer se nogi zamenjata in se karakter že pripravlja na odriv [4.38]. Vstavimo jo na 5. mesto na časovnici.

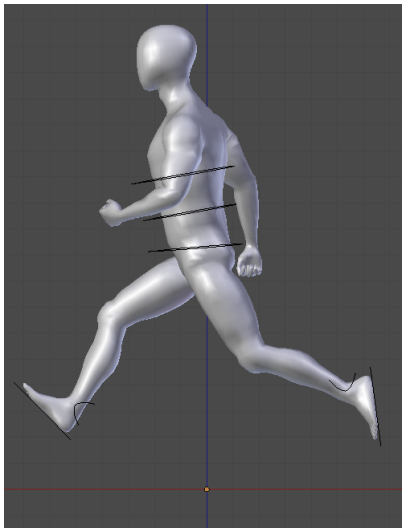


Slika 4.38: Prehodna poza v ciklu teka

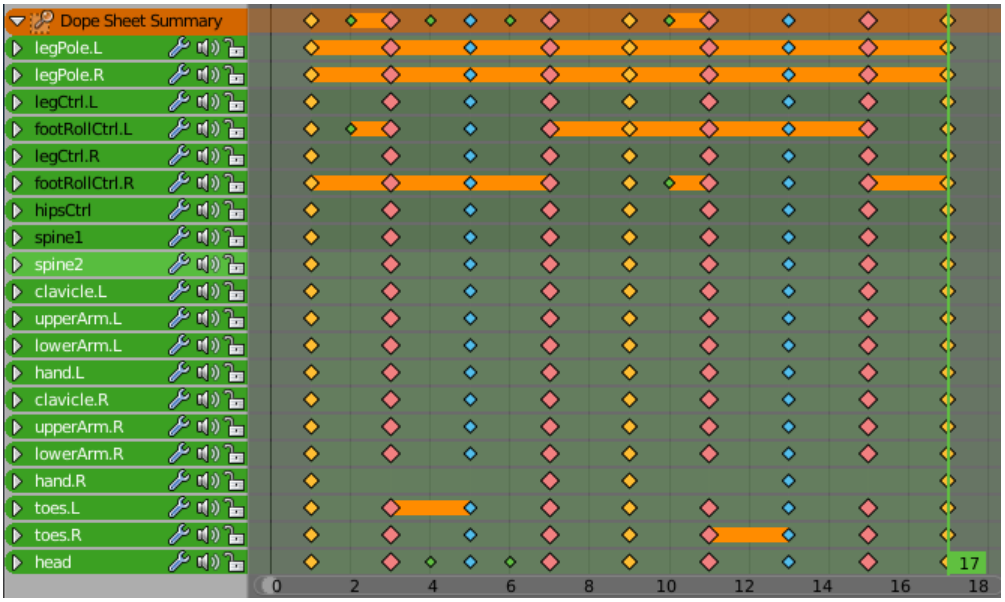
Naslednji pozi sta ekstrema. V prvem je karakter v najnižji točki [4.39]. V drugem ekstremu pa je že v zraku [4.40].



Slika 4.39: Prvi ekstrem



Slika 4.40: Drugi ekstrem




Slika 4.41: Vsi okvirji v ciklu teka. Druga polovica je, tako kot pri hoji, obratna prvi polovici.

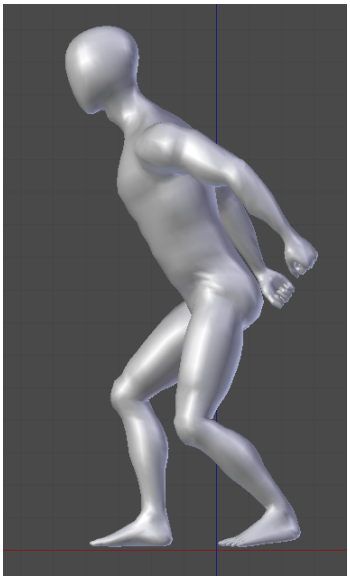
### 4.7.3 Skok

Animacija skakanja se ne ponavlja, tako kot se ponavljata animaciji hoje in teka. Ponavlja se samo del animacije in sicer takrat, ko je naš karakter v zraku in se dviguje oziroma pada. Zato moramo pravzaprav animacijo skoka razdeliti na tri dele. Prvi del je odskok ali odriv, drugi del je, ko je karakter v zraku in lebdi (ta del se ponavlja), tretji in zadnji del animacije pa je doskok oziroma pristanek.

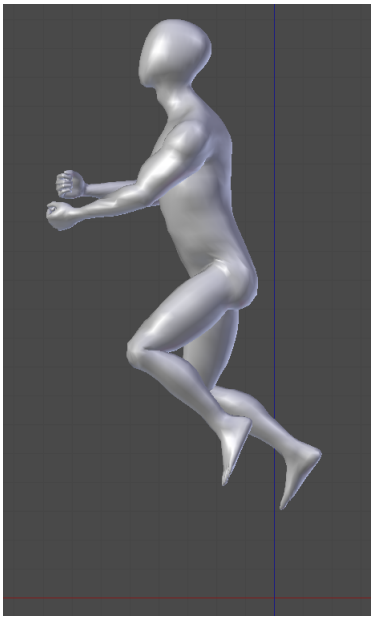
Tak način smo izbrali zaradi kasnejše implementacije skoka v igralnem pogonu UE4. Razlog je ta, da karakter ob skoku ni vedno enako časa v zraku. V kolikor skoči na recimo pol metra visoko polico, bo v zraku veliko manj časa, kot če skoči iz deset metrov visoke pečine v morje. Zato je nujno, da v igralnem okolju ponavljamo animacijo lebdenja.

### Odriv

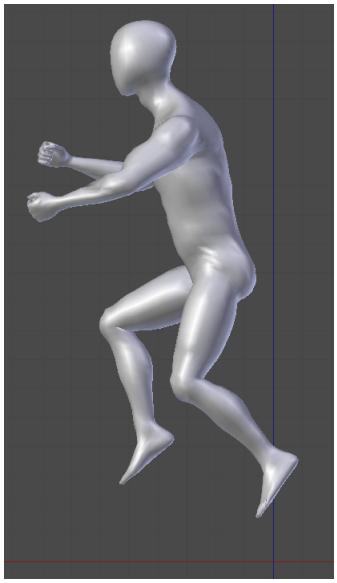
Animacija odriva traja 13 okvirjev.



Slika 4.42: 1. okvir animacije odriva



Slika 4.43: 7. in srednji okvir animacije, karakter v najvišjem položaju.



Slika 4.44: 13. in zadnji okvir animacije odriva. Karakter je malce nižje kot v 7. okvirju in je pripravljen na vstop v ponavljajočo animacijo lebdenja.

**Lebdenje**

V tej animaciji se ne zgodi nič posebnega. Vsega skupaj traja sicer 20 okvirjev, vendar samo s tremi ključnimi okvirji, od tega sta prvi in zadnji okvir enaka zadnjemu okvirju iz animacije odriva (s tem dosežemo gladek prehod iz enega dela animacije v drugega), tretji pa je samo manjša variacija, ki nakazuje na neko gibanje med lebdenjem. V kolikor tega gibanja ne bi bilo, bi bil karakter med lebdenjem zelo tog in neprepričljiv.

Slika 4.45: Animacija lebdenja, srednji okvir

**Doskok**

Doskok traja 7 okvirjev. Prvi okvir je podoben zadnjemu okvirju iz animacije lebdenja.

Slika 4.46: Karakter med pristajanjem

Slika 4.47: Karakter po pristanku

### 4.7.4 Nedejavno stanje

Nedejavno stanje (angl. „idle state“) je animacija, ki se izvaja vedno, ko se ne izvaja nobena druga animacija. S pomočjo te animacije deluje naš karakter bolj živo in prepričljivo, saj prikazuje manjše gibe in dihanje, ko stoji pri miru.

Skupno traja animacija 120 okvirjev, od tega so samo trije ključni, vsi ostali pa avtomatsko interpolirani. Prvi in zadnji sta ponovno enaka, saj se animacija ves čas ponavlja.

Slika 4.48: Prvi okvir nedejavnega stanja

Slika 4.49: Drugi okvir nedejavnega stanja

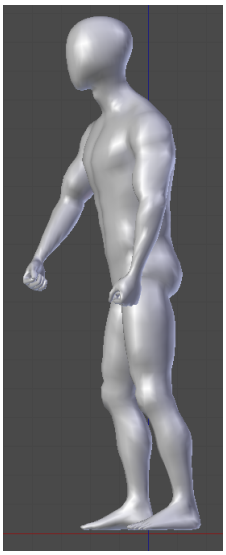
### 4.7.5 Napad 1

Prvi napad je animacija, ki se lahko izvaja med ostalimi animacijami, kot so nedejavno stanje, hoja ali tek. Traja 23 okvirjev.

7. okvir vsebuje pozo, v kateri se karakter pripravi na zamah. Roka je skrčena in dvignjena v zrak [4.50].

Slika 4.50: Sedmi okvir animacije prvega napada. Karakter pripravljen na zamah z roko.

11. okvir vsebuje pozo, v kateri je karakter že zamahnil z roko [4.51].



Slika 4.51: 11. okvir animacije prvega napada

Celoten napad se pravzaprav zgodi med 7. in 11. okvirjem in sicer z interpoliranjem vmesnih okvirjev. Sam napad je primerno kratek prav zaradi

majhnega števila okvirjev (7, 8, 9, 10, 11 – 5 okvirjev, približno 0.21 sekunde). Ostali okvirji služijo vračanju iz animacije napada v nedejavno stanje.

### 4.7.6 Napad 2

Napad 2 se od prvega razlikuje v tem, da se med tem napadom karakter ne sme premikati. Traja 52 okvirjev, struktura pa je podobna Napadu 1 [4.7.5].

Prvi in zadnji okvir sta enaka zadnjemu okvirju iz nedejavnega stanja. 11. okvir je poza, v kateri se karakter agresivno pripravi na zamah, tako da se prevesi na desno nogo in nagne nazaj [4.52].

Slika 4.52: Karakter nagnjen nazaj, prav tako se nazaj in navzgor pomika desna roka.

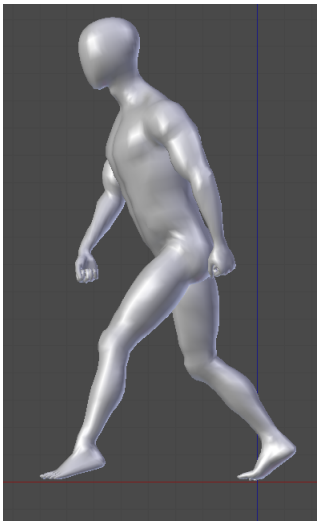
20. okvir je poza, v kateri se karakter s telesom že začne nagibati naprej v napad, roka pa je zaradi zakasnitve šele dobro prišla v svoj ekstrem [4.53].

Slika 4.53: 20. okvir animacije Napad 2

Naslednji ključni okvir je 25. V tej pozi je karakter zamahnil z roko in je s telesom močno nagnjen naprej [4.54].

Slika 4.54: 25. okvir animacije Napad 2

V 35. okvirju se karakter s trupom in rokami že vrača v pozo nedejavnega stanja, sprednja noga pa je še vedno na tleh, saj se nanjo opira [4.55].



Slika 4.55: 35. okvir animacije Napad 2, vračanje v nedejavno pozo.

Zadnji okvir je poza nedejavnega stanja.

## 4.8 Modeliranje in dodajanje opreme karakterju

Brez oblek, orožja in ostale opreme je naš model le generično človeško telo. V tem poglavju si bomo pogledali, kako modelirati različno opremo in jo dodati obstoječemu okostju.

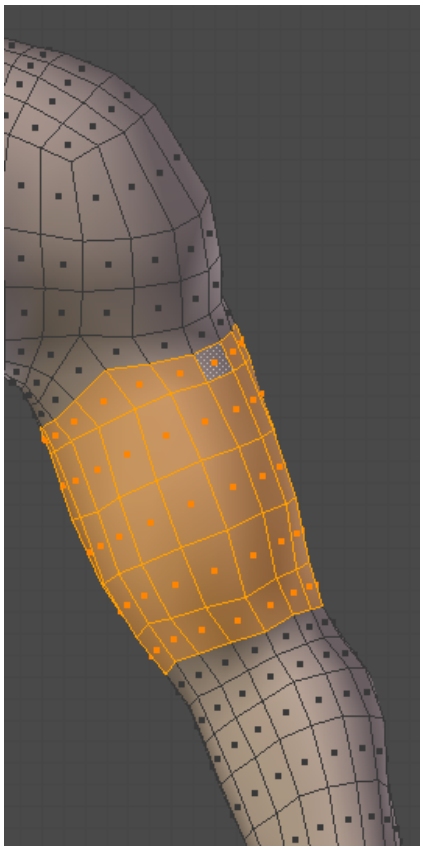
Z različno opremo lahko iz enega modela (v našem primeru človeškega telesa) naredimo več različnih karakterjev, vsakega s svojo opremo. Na tak način ni potrebe po ponovnem kiparjenju človeškega telesa za vsak karakter posebej.

Za to diplomsko nalogo smo izdelali tri različne karakterje, oziroma bolje rečeno, tri sete opreme za telo, ki je podlaga za vse tri karakterje.

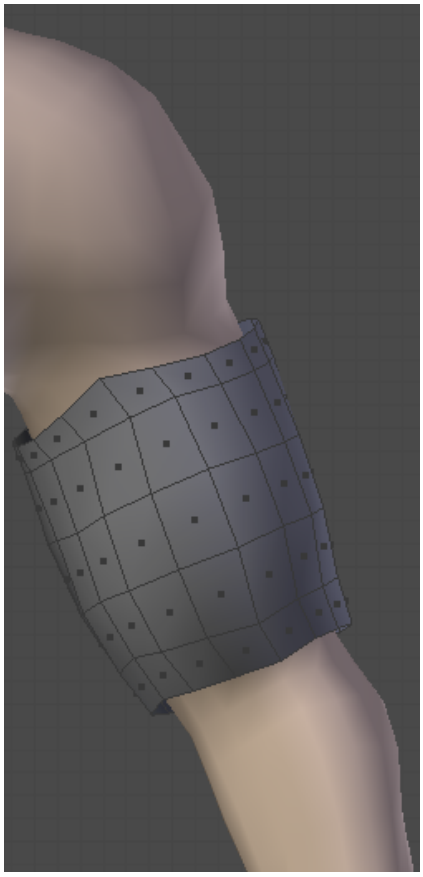
### 4.8.1 Špartanski bojevnik

Za ta primer smo posebej izdelali čelado, prsni oklep, krilo, par ščitnikov za roke, par ščitnikov za noge, sandale, ogrinjalo, meč in ščit.

Za opremo, ki se prilega na telo (oklep, krilo, ščitniki, sandali) si pomagamo z obstoječim retopologiranim modelom, tako da za vsak kos opreme izberemo ploskve na modelu [4.56], jih podvojimo, izvlečemo in ločimo od modela [4.57].



Slika 4.56: Izbrane ploskve na retopologiranem modelu

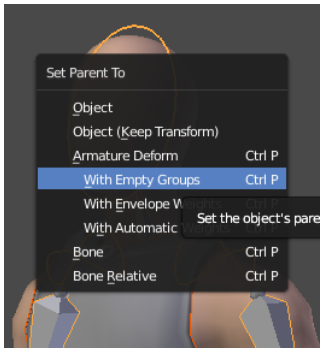


Slika 4.57: Ploskve podvojene, izvlečene in ločene od modela

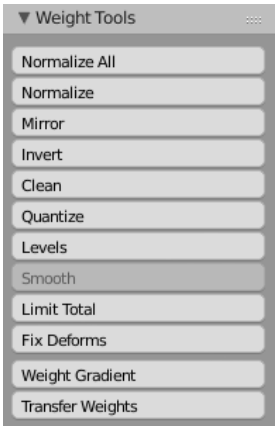
Na tem mestu se lahko izdelana oprema prekriva z našim modelom, kot je vidno na sliki [4.58]. Ta problem lahko rešimo z uporabo oblikovnih ključev (angl. „shape key“), ki so razloženi v poglavju Oblikovni ključi [4.8.3].

Slika 4.58: Oprema bojevnika

Zaenkrat se narejena oprema še ne deformira s telesom. Najprej moramo opremo dodati pod hierarhijo armature [4.59] nato pa vplive, ki smo jih nastavili v poglavju Določanje vpliva okostja [4.6.2], prenesti na opremo [4.60].




Slika 4.59: Dodajanje opreme v hierarhijo armature




Slika 4.60: Prenos vplivov na novo opremo z uporabo funkcije „Transfer weights“.

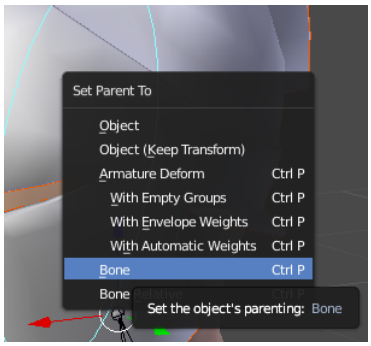
Sedaj se oprema ustrezno deformira. V primeru, da so potrebni manjši popravki, lahko to naredimo z ročnim barvanjem vpliva.

Čelado oblikujemo na klasičen način, podobno kot grobo obliko v poglavju Oblikovanje grobe oblike [4.2]. Pomagamo si z referenčnimi slikami [4.61].

Slika 4.61: Primer referenčne slike za modeliranje čelade

Slika 4.62: Model čelade

Čelada se kot taka ne deformira, mora pa se premikati skupaj z glavo. To dosežemo tako, da jo pripnemo neposredno na kost v armaturi, ki predstavlja glavo, imenovano „head“ [4.63]. V resnici ji s tem določimo starša v hierarhiji.

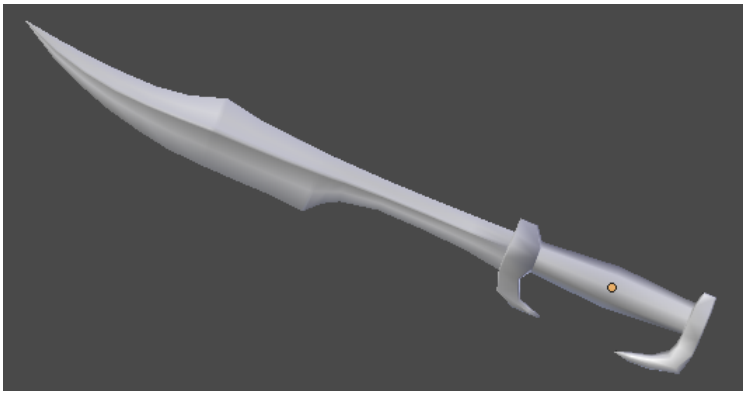



Slika 4.63: Pripenjanje čelade neposredno na izbrano kost v armaturi

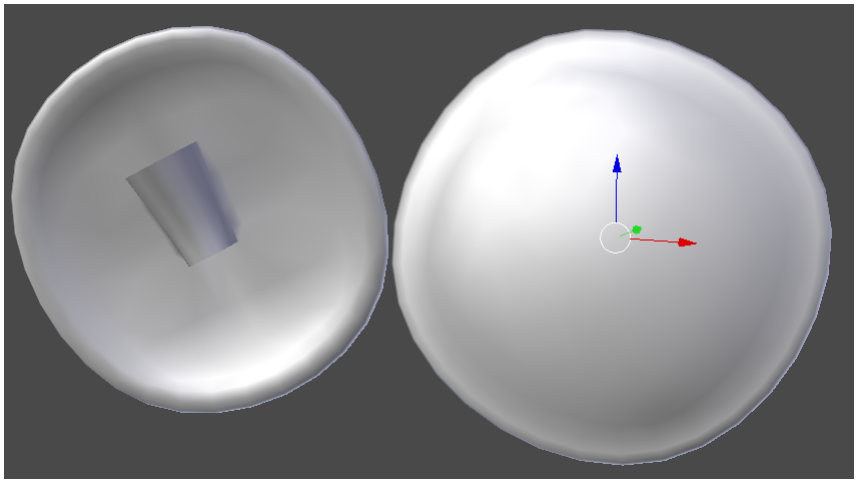
Plašča se lotimo na drugačen način. UE4 namreč ponuja simuliranje blaga. To pomeni, da se v igralnem okolju model, ki je ustrezno označen kot blago, kot tako tudi obnaša in je nanj moč vplivati s fizikalnimi silami. Zato plašč v Blenderju oblikujemo zelo preprosto [4.64], brez podrobnosti, kot so na primer gubanje blaga.

Slika 4.64: Model plašča

Tako kot čelado, tudi plašč pripnemo neposredno na kost v armaturi, ki predstavlja vrat, imenovano „neck“. Preostaneta nam še meč [4.65] in ščit [4.66]. Ponovno si pomagamo z referenčnimi slikami.



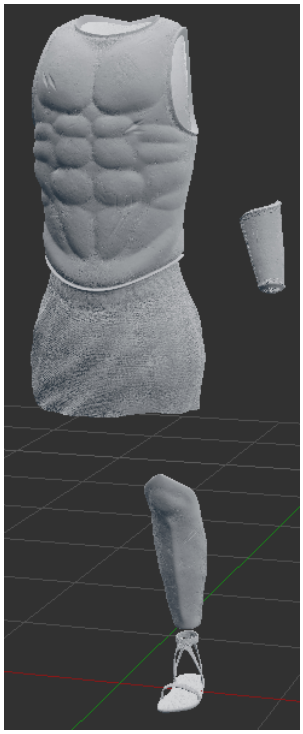
Slika 4.65: Meč



Slika 4.66: Ščit

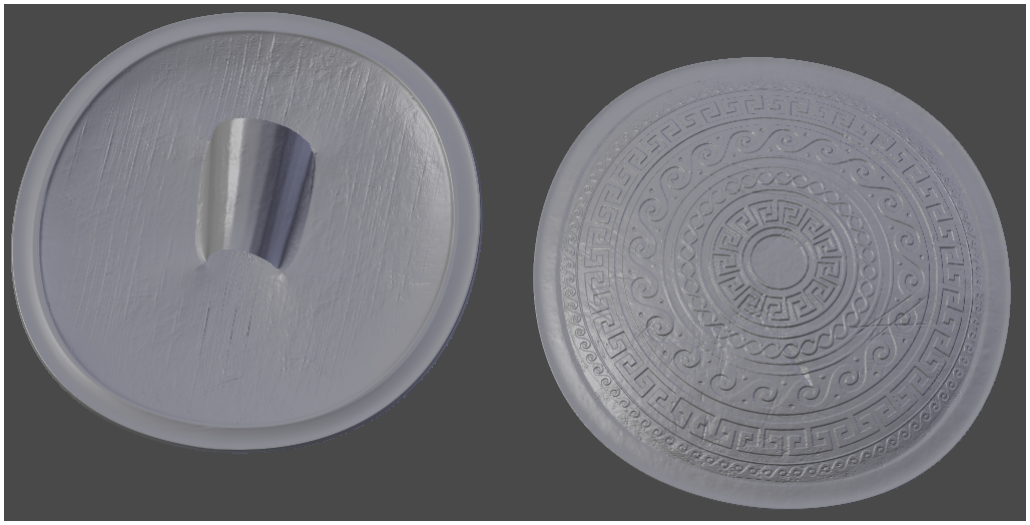
Vse potrebne modele imamo sicer pripravljene za uporabo, vendar pa so brez kakršnih koli podrobnosti. Le-te dodamo s kiparjenjem, na način kot smo dodajali podrobnosti modelu človeškega telesa v poglavju Kiparjenje [4.3]. V poglavju Teksturiranje [4.9] detajle iz skiparjenih modelov preslikamo na prejšnje modele, ki so primerni za igralno okolje.

Slika 4.67: Skiparjena čelada



Slika 4.68: Skiparjena ostala oprema. Zaradi uporabe zrcalnega modifikatorja je prikazana samo leva stran.

Slika 4.69: Skiparjen meč



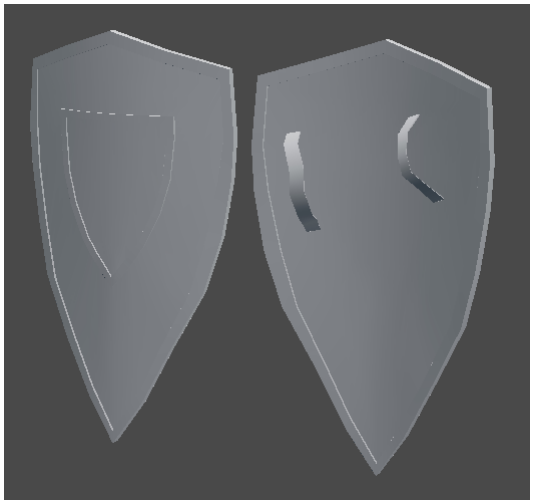
Slika 4.70: Skiparjen ščit

### 4.8.2 Srednjeveška viteza

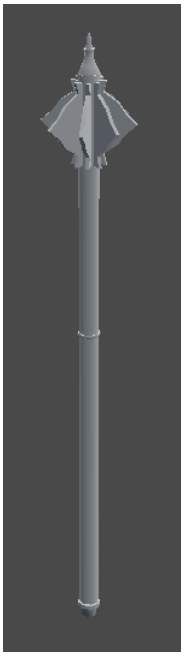
Na način, opisan v poglavju Špartanski bojevnik [4.8.1], smo naredili še dve izvedbi srednjeveških vitezov, prvi je vitez templar [4.71], drugi pa francoski vitez [4.72].

Slika 4.71: Vitez templar. Oprema: čelada, verižni oklep, ogrinjalo, rokavice, obutev

Slika 4.72: Francoski vitez. Oprema: čelada, oklep, ščitniki, verižni oklep, rokavice, krilo



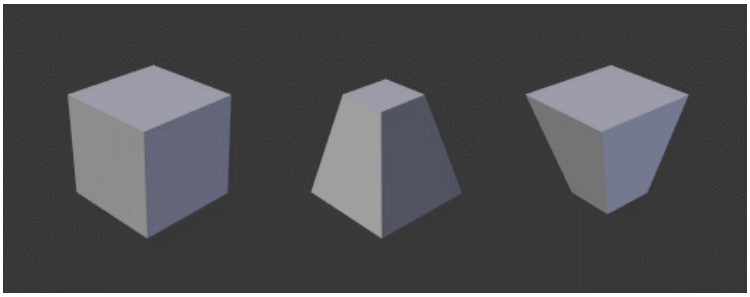
Slika 4.73: Ščit za viteza



Slika 4.74: Orožje za viteza

### 4.8.3 Oblikovni ključi

Oblikovni ključi (angl. „shape keys“) se uporabljajo za animiranje deformacij, ki spreminjajo obliko objektom.



Slika 4.75: Primer objekta, na katerem so uporabljeni različni oblikovni ključi (Vir: Blender dokumentacija [1])

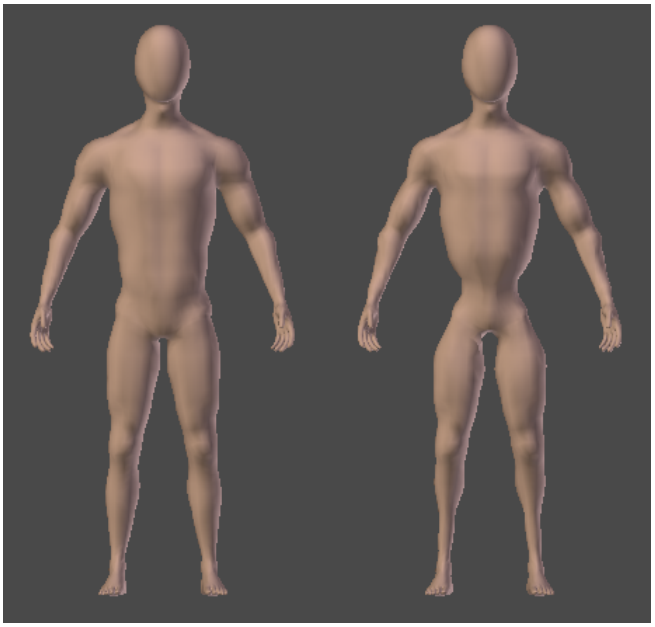
Oblikovne ključe pa lahko uporabimo tudi drugače. Pogosto je, da se objekti med animacijo med seboj prekrivajo. Primer je naš karakter, kjer se telo prekriva z oblačili [4.76].

Slika 4.76: Prekrivanje oblačil s telesom

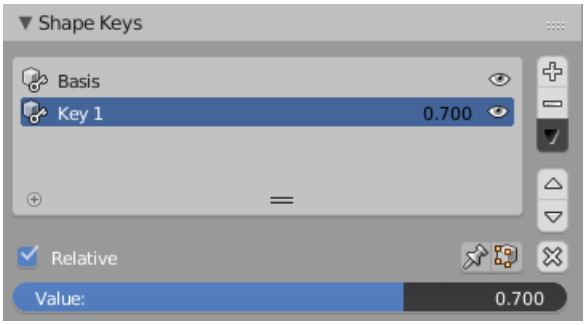
Tega problema se lahko lotimo na različne načine. Prvi je, da ročno spremenimo vplive kosti na oblačila ali telo z ročnim barvanjem vplivov, glej poglavje Določanje vpliva armature.

Drugi način je, da izbrišemo tiste dele mreže, ki se prekrivajo in jih tako ali tako ne bi v igralnem okolju nikoli videli. Pri tem je nekaj prednosti in nekaj slabosti. Prednost je v tem, da s tem optimiziramo izrisovanje, saj lahko krepko zmanjšamo število poligonov in obenem rešimo problem prekrivanja. Slabost pa je, da je ta način destruktiven. Mreže, ki smo jo izbrisali, namreč ne moremo več dobiti nazaj.

Tretji način pa je z uporabo oblikovnih ključev. Z njimi lahko manipuliramo z mrežo v realnem času. Poglejmo si primer uporabe [4.77]. Levi model je nespremenjen, vrednost oblikovnega ključa „Key 1“ pa je enaka 0. Ko pa povečamo vrednost tega ključa na 0.7 [4.78], se spremeni tudi oblika modela, kar je vidno na desnem modelu [4.77]. V našem primeru smo ključ zasnovali na način, da skrči predele telesa, kjer se nahaja oprema (oblačila). S tem rešimo problem prekrivanja.



Slika 4.77: Primer uporabe oblikovnega ključa




Slika 4.78: Uporabniški vmesnik za urejanje oblikovnih ključev

## 4.9 Teksturiranje

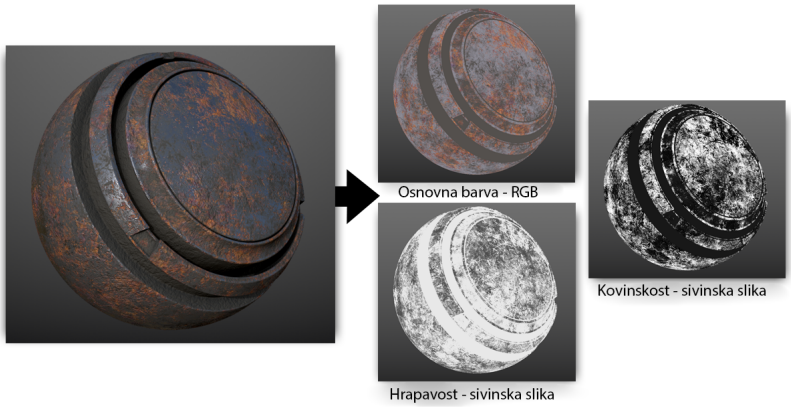
### 4.9.1 Metoda PBR

PBR (angl. kratica za Physically Based Rendering) je metoda senčenja in izrisovanja, ki zagotavlja bolj natančno predstavitev interakcije svetlobe s površinami [5]. Ker temelji na natančnih fizikalnih formulah, se z uporabo PBR metode izognemo ugibanju lastnosti površin, poleg tega pa so teksturirani objekti pravilno osvetljeni pri vseh svetlobnih pogojih.

Pri uporabi metode PBR se lahko odločimo za enega izmed dveh načinov poteka dela. Prvi se imenuje „Metal / Roughness“, drugi pa „Specular / Glossines“. Kljub temu, da je pri vsakem nekaj prednosti in slabosti, sta si enakovredna. V tej diplomski nalogi smo se odločili za prvi način, ker je bolj intuitiven in bolj pogosto uporabljen v industriji.

„Metal / Roughness“ način je definiran z množico kanalov, ki jih posredujemo senčilniku (angl. shader) v obliki različnih map oziroma tekstur. Mape, specifične za ta način dela, se imenujejo osnovna barva (angl. base color), kovinskost (angl. metallic) in hrapavost (angl. roughness).

Senčilnik poleg teh map uporablja še okoljsko senčenje (angl. ambient occlusion) in normale.



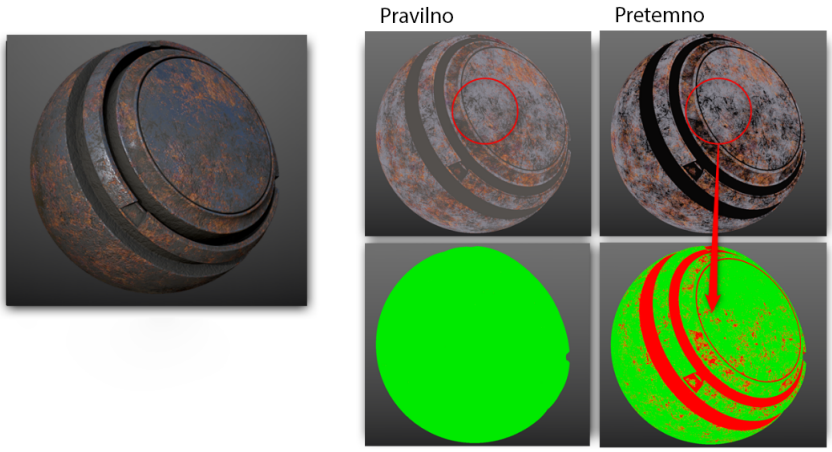

Slika 4.79: Prikaz „Metallic / Roughness“ načina (Vir: Senčenje na osnovi fizike [5])

### Osnovna barva

Osnovna barva (angl. base color) je RGB tekstura (ali mapa), ki vsebuje dva tipa podatkov. Prvi je odbojna barva za dielektrike (površine, ki niso kovinske), drugi podatek pa je odbojna vrednost za kovine [4.80].


Slika 4.80: Osnovna barva vsebuje odbojne barve in odbojne vrednosti (Vir: Senčenje na osnovi fizike [5])

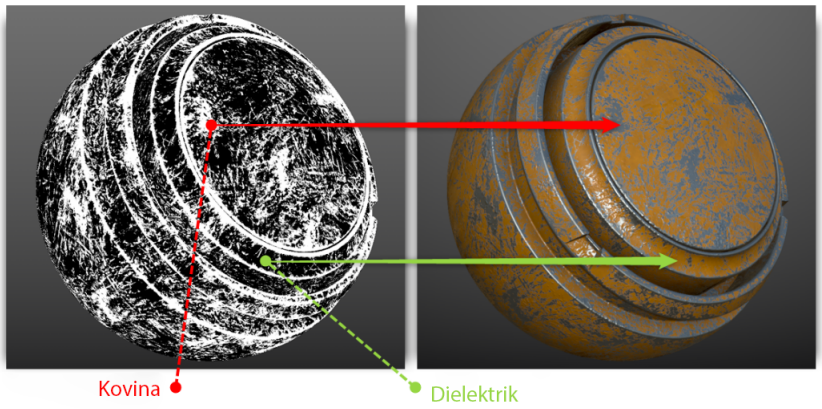
V naravi so objekti svetlejši, kot si jih predstavljamo. Za primer vzemimo oglje, ki je zelo temne barve, ni pa stoodstotno črno. Tako tudi sveže zapadli sneg ni stoodstotno bel. Zato se moramo pri izbiri barv držati omejitev. Za temne dielektrike naj ne bi izbrali barvne vrednosti nižje od vrednosti 30 sRGB, za svetle pa ne višje od 240 sRGB [4.81]. Za kovine je odbojna vrednost visoka, od 180 do 255 sRGB.




Slika 4.81: Razpon osnovne barve za dielektrike (Vir: Senčenje na osnovi fizike [5])

**Kovinskost**

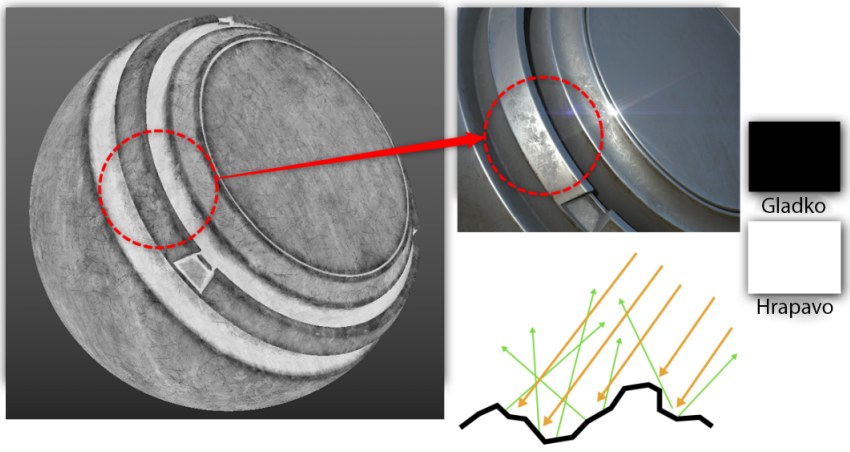
Mapa kovin se uporablja za definiranje področij, ki predstavljajo kovinske površine. Je sivinska tekstura in sporoča senčilniku, kako naj interpretira vrednosti iz mape osnovnih barv. Mapa kovin je največkrat binarna, z vrednostjo 0 (črna barva – 0 sRGB), tam kjer je površina dielektrična, in vrednostjo 1 (bela barva – 255 sRGB), tam kjer je kovinska. Uporabimo lahko tudi vmesne vrednosti, predvsem tam, kjer želimo ponazoriti oksidacijo ali pa blatne površine.




Slika 4.82: Pobarvana kovina se obravnava kot dielektrik (Vir: Senčenje na osnovi fizike [5])

### Hrapavost

Mapa hrapavosti (roughness map) se uporablja za ponazarjanje hrapavosti/gladkosti površin. Pri hrapavih površinah – kot je recimo papir – se odbita svetloba močno razprši, pri gladkih površinah – kot je recimo pokrov avtomobila – pa se odbita svetloba ne razprši tako močno, ostane bolj usmerjena. V obeh primerih pa je jakost odbite svetlobe enaka. Temne vrednosti v teksturi predstavljajo gladke površine, svetle vrednosti pa grobe [4.83].




Slika 4.83: Mapa hrapavosti opiše površinske nepravilnosti, ki povzročijo različne odbojnosti svetlobe (Vir: Senčenje na osnovi fizike [5])

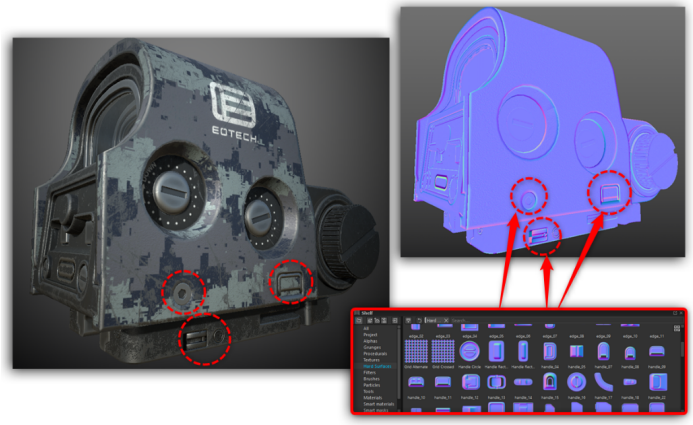
### Okoljsko senčenje

Mapa okoljskega senčenje (angl. ambient occlusion) določa, koliko okoljske svetlobe lahko dostopa do katerekoli točke na površini. S to sivinsko teksturo lahko poudarimo razpoke ali udrtine, do katerih svetloba v naravi težko dostopa [4.84].


Slika 4.84: Okoljsko senčenje (Vir: Senčenje na osnovi fizike [5])

### Normale

Mapa normal se uporablja za ponazarjanje površinskih podrobnosti [4.85]. Je RGB tekstura, kjer vsak izmed RGB kanalov ustreza X, Y in Z koordinatam ploskovnih normal. Ploskovna normala je usmerjenost ploskve. Mape normal lahko uporabimo, da prikažemo podrobnosti iz enega modela na drugem. V našem primeru smo projicirali podrobnosti iz skiparjenih modelov z visokim številom poligonov in veliko podrobnosti na retopologirane modele, ki imajo manjše število poligonov in so tako rekoč brez podrobnosti.



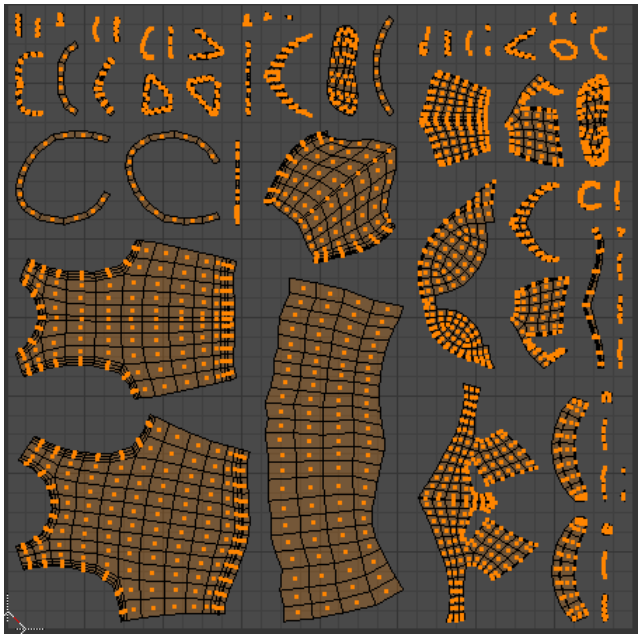
Slika 4.85: Normale lahko uporabimo za prikaz površinskih podrobnosti (Vir: Senčenje na osnovi fizike [5])

### 4.9.2 Preslikovanje podrobnosti

Mapo normal lahko uporabimo za prikazovanje podrobnosti na modelu. S tem v resnici ne spremenimo geometrije modela, pač pa z manipuliranjem usmerjenosti ploskovnih normal le uprizorimo podrobnosti. Na ta način lahko podrobnosti iz skiparjenih modelov (ki so prezahtevni za igralno okolje) učinkovito prikažemo na retopologiranih modelih (ki so primerni za igralno okolje). Tak pristop se v zelo veliki meri uporablja v industriji video iger.

Preden začnemo s preslikovanjem podrobnosti, moramo biti prepričani, da imamo dva modela (skiparjen in retopologiran model), ki sta na istem mestu in se prekrivata [4.86]. Urejene morajo biti tudi UV-koordinate retopologiranega modela [4.87]. Tako bo naš program vedel, na katero mesto v teksturnem prostoru preslikati posamezne podrobnosti.

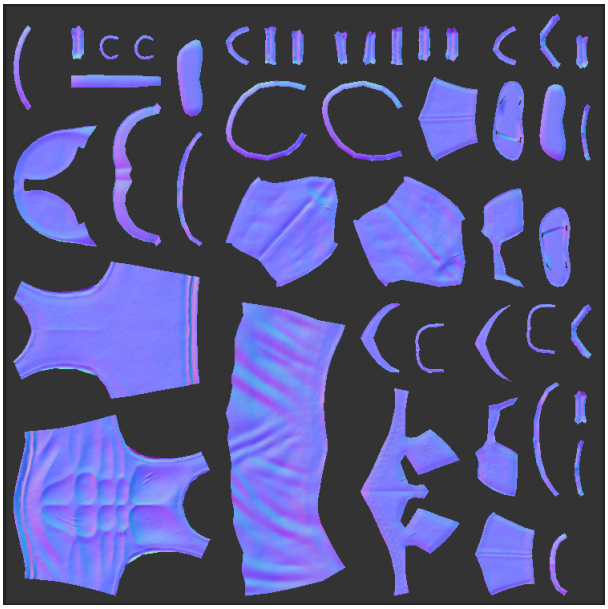
Slika 4.86: Skrajno levo je skiparjen model, katerega podrobnosti želimo preslikati na retopologiran model (skrajno desno). Na sredini sta oba modela na istem mestu in se primerno prekrivata.



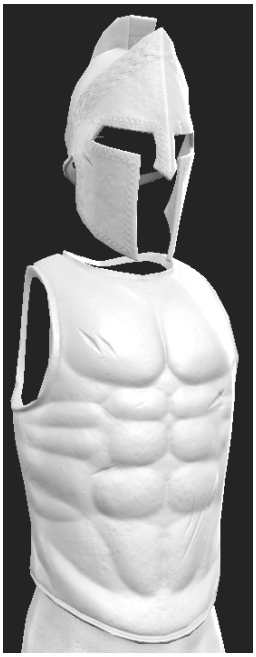
Slika 4.87: UV-koordinate

Ko imamo oba modela in UV-koordinate pripravljene, lahko začnemo s postopkom preslikovanja podrobnosti. To lahko storimo v poljubnem programu, v našem primeru smo se odločili kar za Substance Painter, kjer bomo vse modele tudi teksturirali.

Sam postopek preslikovanja podrobnosti poteka tako, da izbrano orodje za preslikavo začne iz neke razdalje od prvega modela (retopologiranega, nizko število poligonov) in iz njega pošlje žarek proti drugemu modelu (skiparjenemu, veliko število poligonov). Ko ta žarek zadane drugi model, orodje pozna UV-koordinate točke, iz katere je bil poslan, in tja zapiše podrobnosti zadete površine oziroma normale. Na ta način se iz vsake točke na retopologiranem modelu odda žarek, zapiše podrobnosti in na koncu ustvari mapa normal [4.88].



Slika 4.88: Mapa normal po preslikovanju podrobnosti



Slika 4.89: Retopologiran model. Senčilniku smo podali mapo normal, ki vsebuje podrobnosti, preslikane iz skiparjenega modela.

### 4.9.3 Teksturiranje špartanske oprave

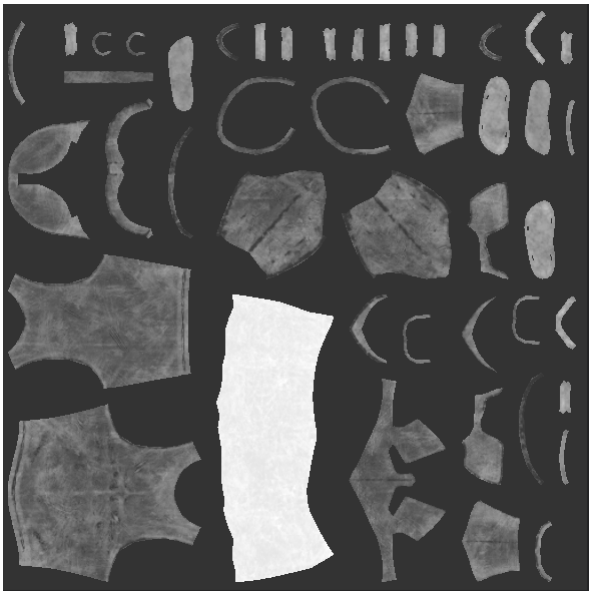
Ko imamo ustvarjeno mapo normal, je potrebno opraviti še ostalo teksturiranje. Ustvariti moramo še vsaj mapo osnovnih barv, hrapavosti, kovinskosti in okoljskega senčenja.

V Substance Painterju imamo na voljo množico že ustvarjenih materialov, od različnih vrst blaga do različnih vrst kovin. Z uporabo in spreminjanjem določenih parametrov lahko že ustvarjene materiale prilagodimo našim željam. V Substance Painterju je podprta tudi uporaba mask, s katerimi označimo, na katere dele modela želimo vplivati s posameznimi materiali.

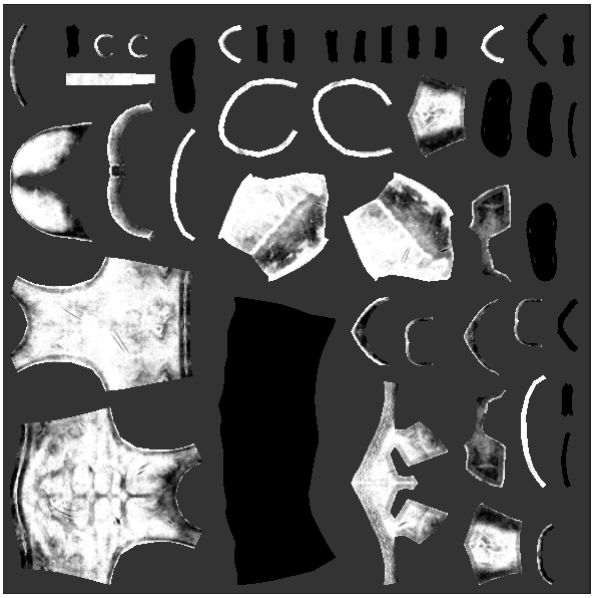

Slika 4.90: Uporabniški vmesnik programa Substance Painter in končano teksturiranje.

Po končanem teksturiranju [4.90] imamo vse potrebne mape, ki so skupaj z našim modelom pripravljene na uporabo v igralnem okolju UE4.

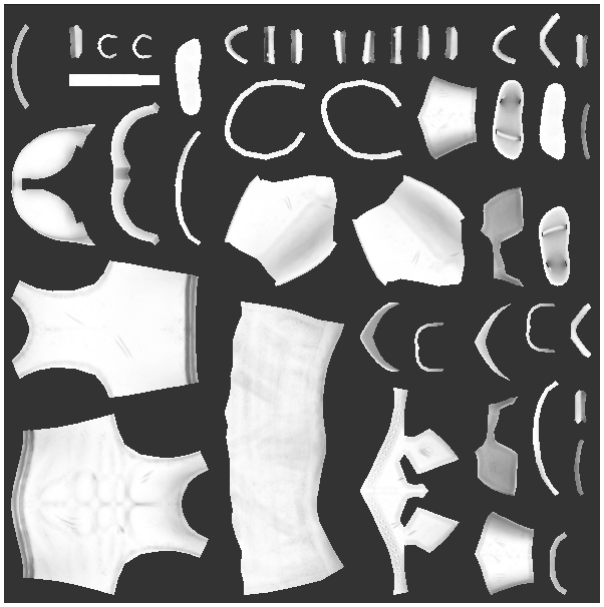
Slika 4.91: Tekstura osnovnih barv



Slika 4.92: Tekstura hrapavosti



Slika 4.93: Tekstura kovinskosti

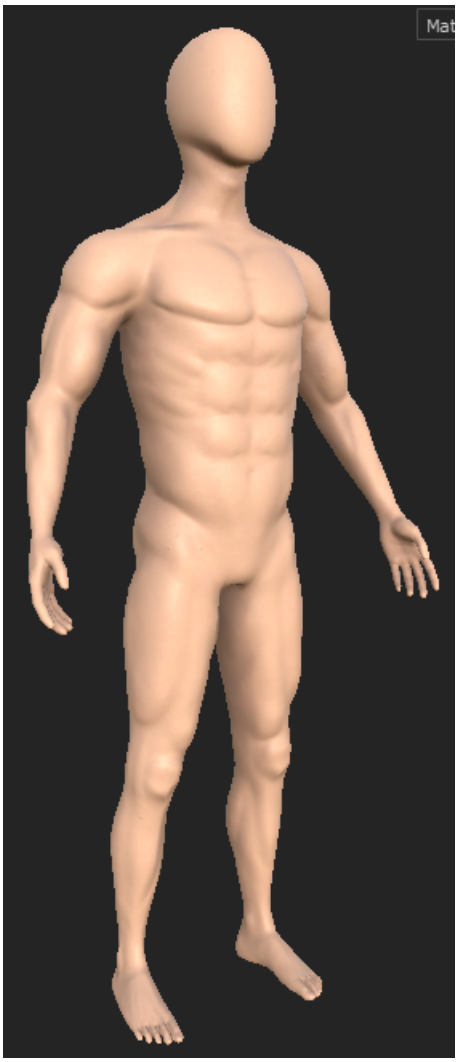


Slika 4.94: Tekstura okoljskega senčenja

### 4.9.4 Teksture ostalih modelov

Na način, opisan v prejšnjih poglavjih, smo preslikali podrobnosti in opravili teksturiranje še za ostale modele.

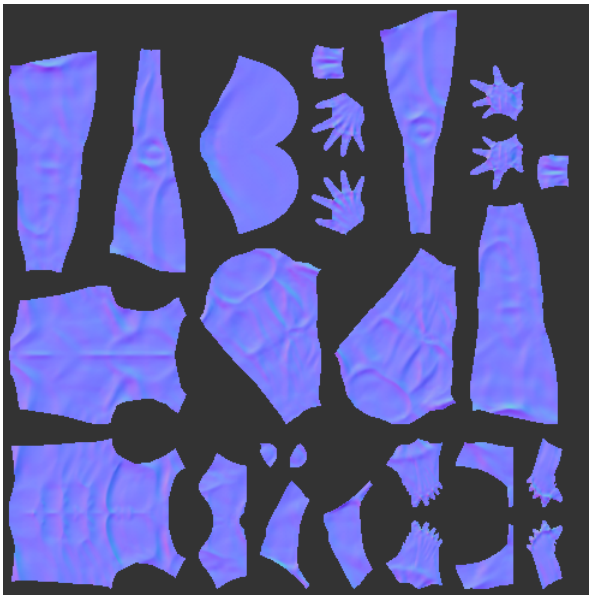
#### 4.9.4.1 Človeško telo



Slika 4.95: Teksturiran model človeškega telesa

Slika 4.96: Tekstura osnovnih barv

Slika 4.97: Tekstura hrapavosti

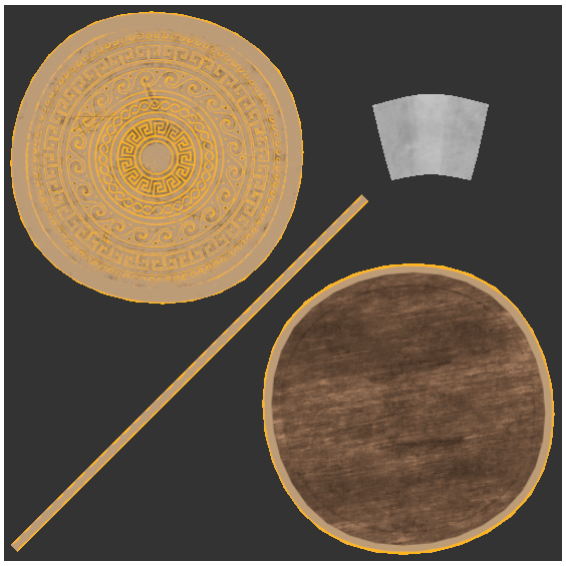


Slika 4.98: Tekstura normal

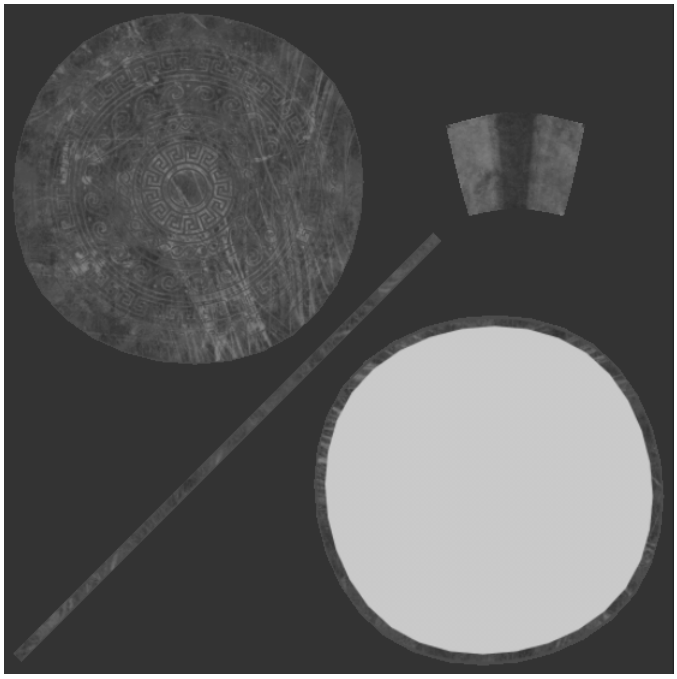
Slika 4.99: Tekstura okoljskega senčenja

#### 4.9.4.2 Špartanski ščit

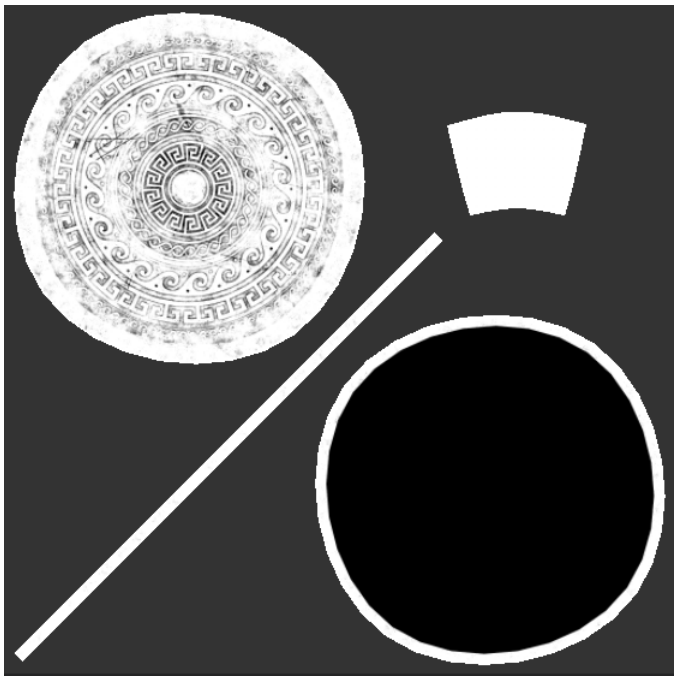
Slika 4.100: Teksturiran špartanski ščit



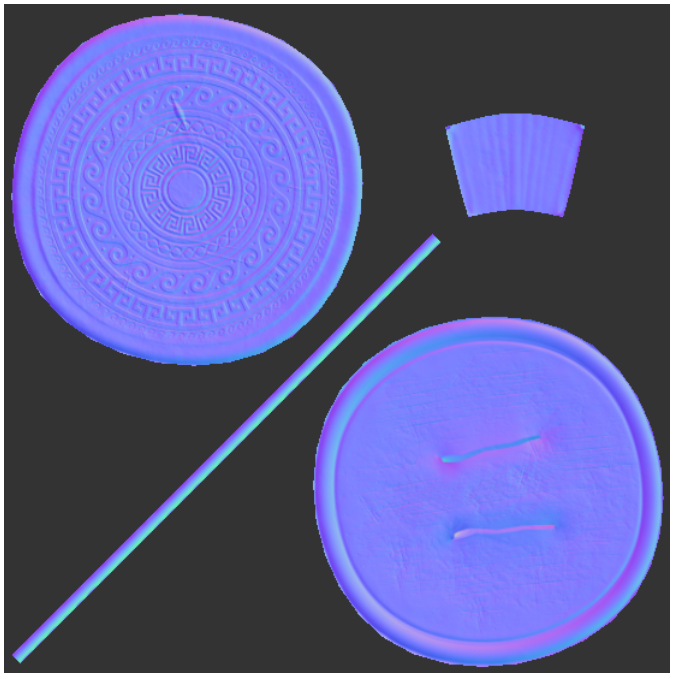
Slika 4.101: Tekstura osnovnih barv



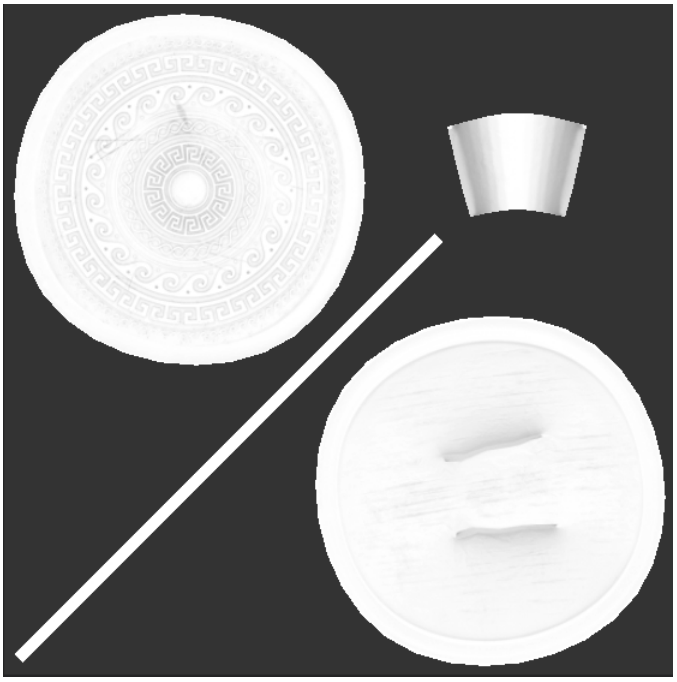
Slika 4.102: Tekstura hrapavosti



Slika 4.103: Tekstura kovinskosti



Slika 4.104: Tekstura normal



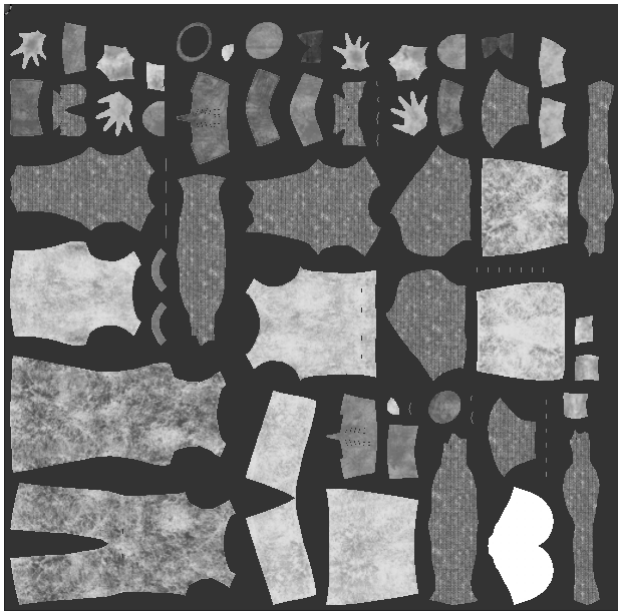
Slika 4.105: Tekstura okoljskega senčenja

#### 4.9.4.3 Vitez templar

Slika 4.106: Teksturiran model viteza templarja

Slika 4.107: Tekstura osnovnih barv

Slika 4.108: Tekstura kovinskosti

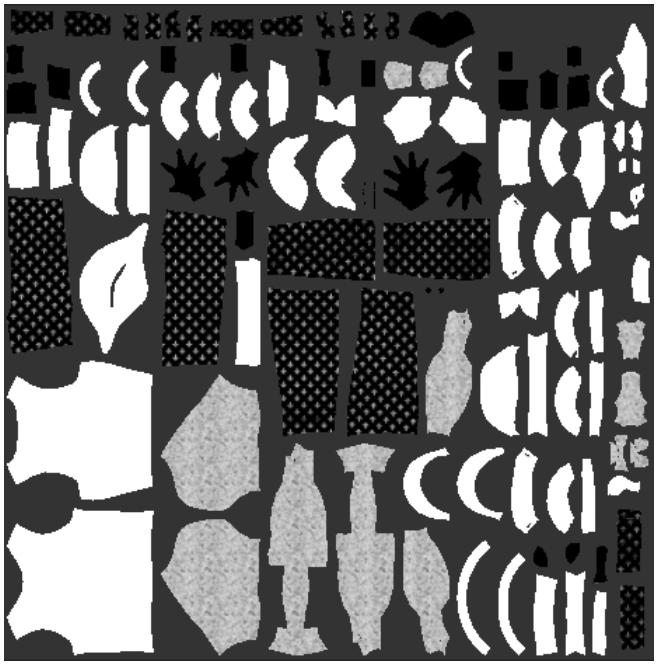


Slika 4.109: Tekstura hrapavosti

#### 4.9.4.4 Francoski vitez

Slika 4.110: Teksturiran model francoskega viteza

Slika 4.111: Tekstura osnovnih barv



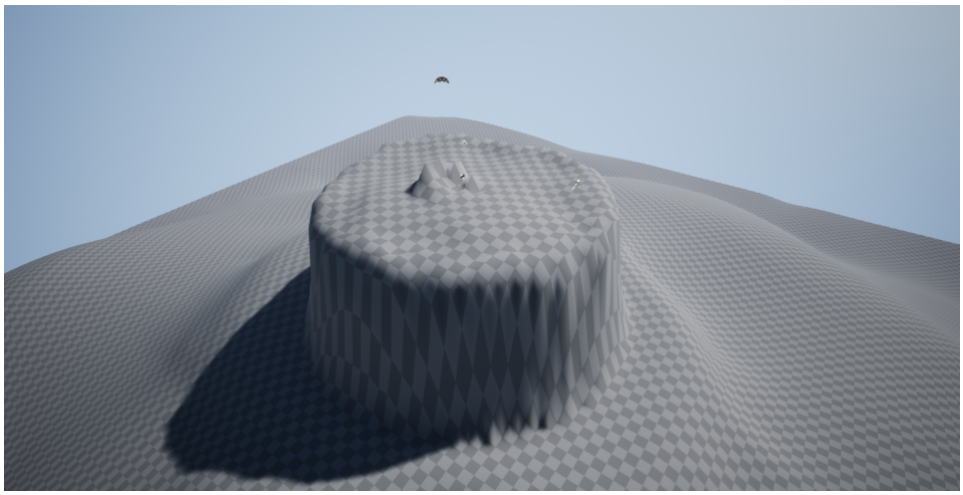
Slika 4.112: Tekstura kovinskosti

Slika 4.113: Tekstura hrapavosti

## 4.10 Igralno okolje

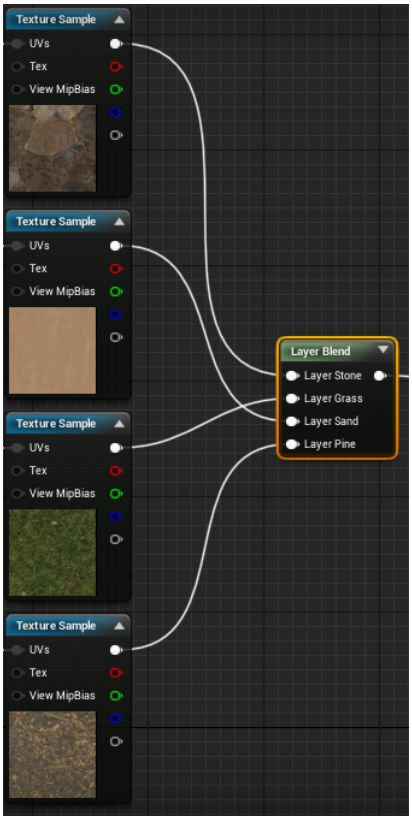
### 4.10.1 Ustvarjanje pokrajine

V igralnem pogonu Unreal Engine 4 imamo za ustvarjanje pokrajin na voljo sistem, imenovan angl. „Landscape“. Z njim lahko ustvarimo poljubna področja, kot so gore, doline, ravne površine, kotline, jame. Postopek ustvarjanja pokrajin je zelo podoben kiparjenju [4.3]. V našem primeru smo izdelali preprost otok [4.114], na katerem so demonstrirani karakterji, ki smo jih izdelali v sklopu te diplomske naloge.

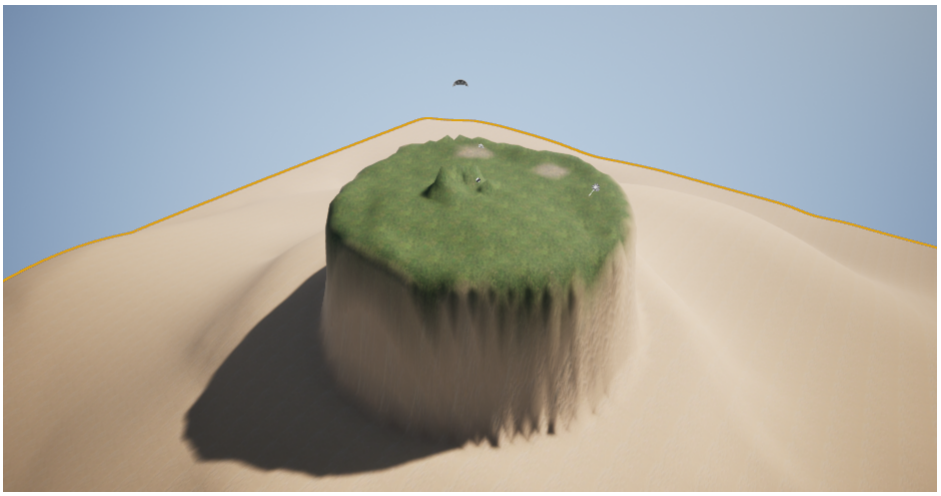


Slika 4.114: Model pokrajine v igralnem okolju

Naslednji korak je izdelava materiala za pokrajino, ki je v naravi sestavljena iz več različnih površin (trava, pesek, kamenje, blato...). Pri tem si v igralnem pogonu pomagamo s kombiniranjem plasti (angl. „layer blend“) [4.115]. S tem senčilniku določimo, za katere predele pokrajine uporabi katere teksture. Same predele pokrajine pa ročno označimo sami [4.116].




Slika 4.115: Kombiniranje plasti (pravzaprav tekstur)



Slika 4.116: Prikaz pokrajinskega materiala

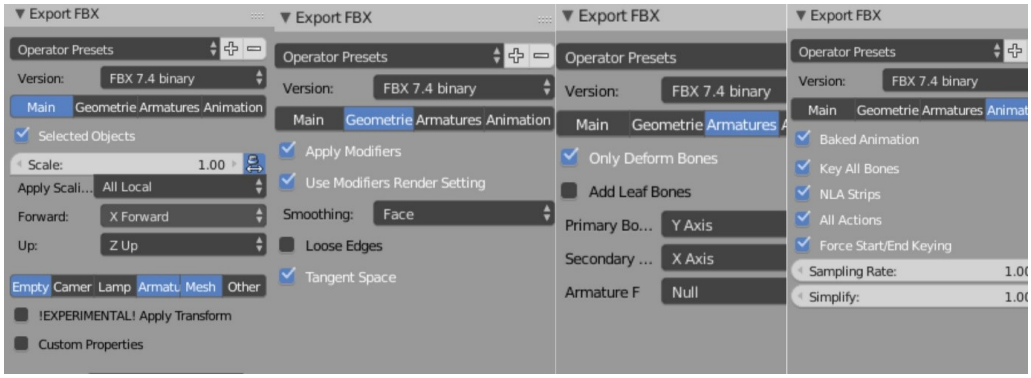
**Ocean in ostala okolica**

Za ocean lahko uporabimo odprtokodni dodatek, razvit s strani skupnosti UE4 [4]. Za zaključni izdelek smo otok obdali s skalovjem, dodali travo na travnate površine in par dreves ter na sredini otoka izdelali manjšo skalnato gručo s stopniščem [4.117].

Slika 4.117: Prikaz končane pokrajine

### 4.10.2 Izvoz modelov

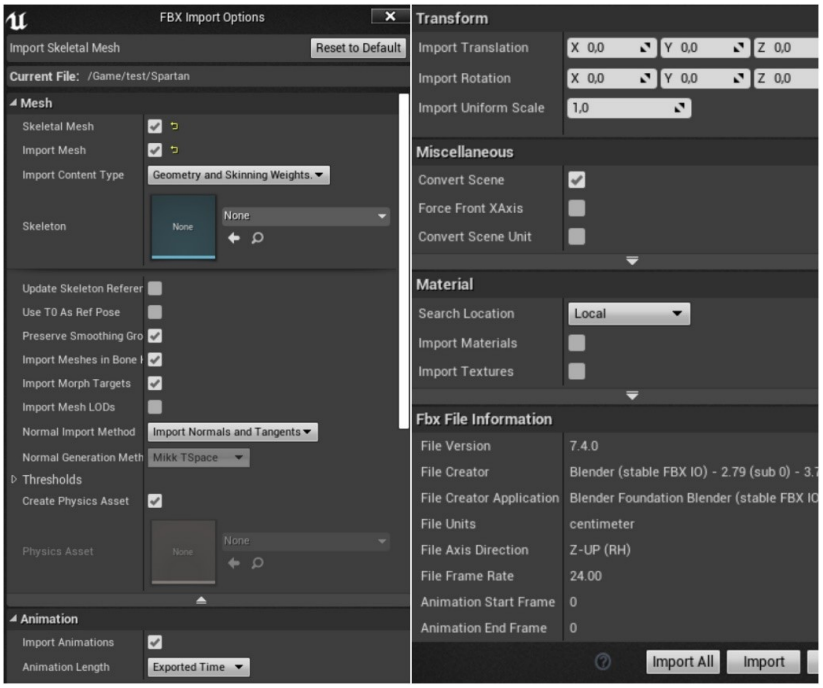
Preden lahko v igralno okolje uvozimo narejene modele iz Blenderja, jih moramo iz Blenderja izvoziti v datoteke primernega formata. Pogosto uporabljen format, ki ga UE4 podpira, je format FBX, ki lahko vsebuje vse od samega modela, armature, animacij, tekstur in materialov. V našem primeru smo izpustili teksture in materiale, saj smo le-te izdelali posebej v programu Substance Painter [4.118].




Slika 4.118: Izvozne nastavitve v Blenderju

### 4.10.3 Uvoz modelov

Ko uvažamo karakter v formatu FBX v igralni pogon UE4, moramo preveriti nastavitve uvoza [4.119].




Slika 4.119: Uvozne nastavitve v UE4

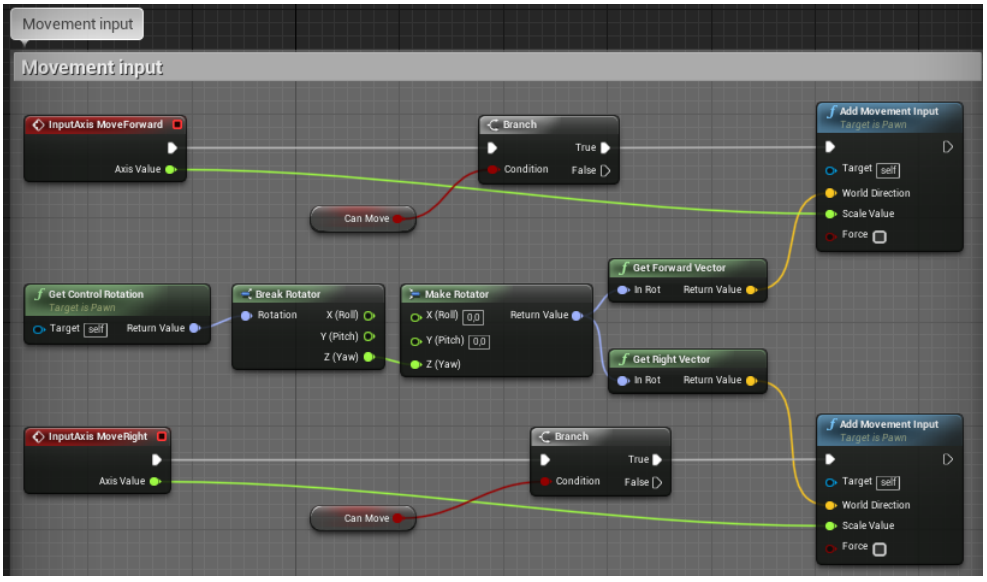
### 4.10.4 Programiranje v UE4

V UE4 lahko programiramo na dva različna načina. Prvi je klasični, s programiranjem v jeziku C++. Drugi način pa je vizualno programiranje z „Načrti“ (angl. „Blueprints Visual Scripting“), ki temelji na povezovanju vozlišč, dogodkov, funkcij in spremenljivk med seboj.

V naši diplomski nalogi smo z uporabo vizualnega programiranja implementirali premikanje naših karakterjev in menjavanje med njimi, tako da lahko izbiramo, katerega izmed njih kontroliramo.

#### 4.10.4.1 Načrt karakterja

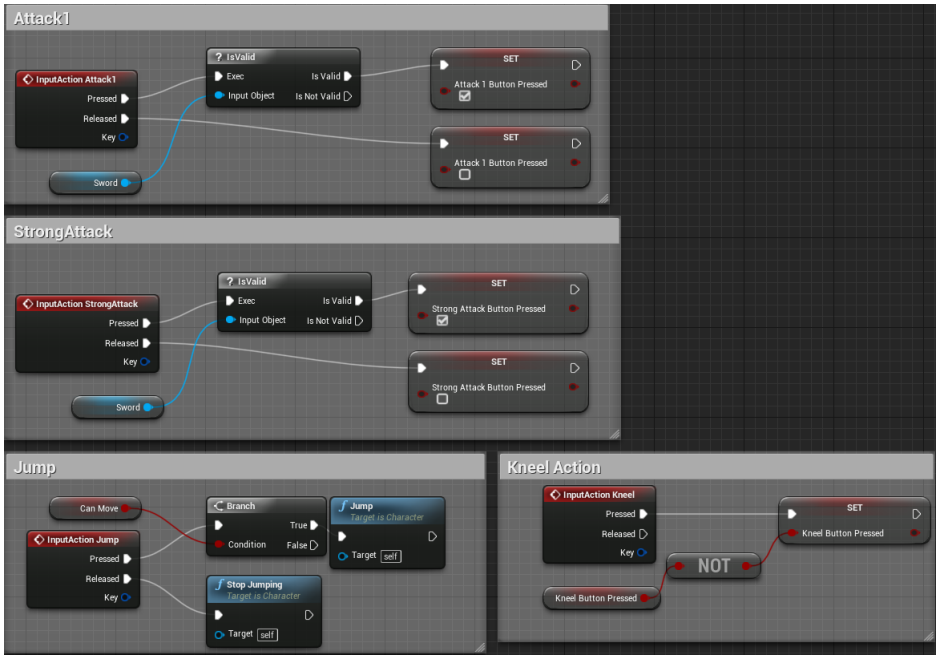
Vse v zvezi s premikanjem in dodajanjem/odstranjevanjem orožja je implementirano v enem Načrtu, ki spada v razred „Character Class“.




Slika 4.120: Implementirano premikanje

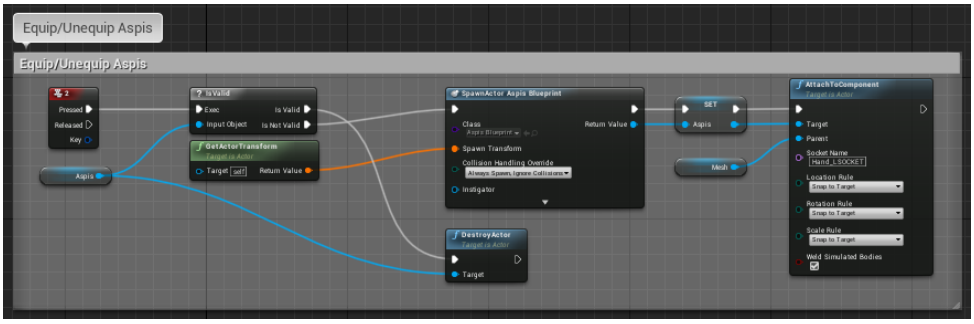
Na sliki [4.120] vidimo, da sta za premikanje odgovorna dva dogodka. Ta dogodka se prožita ob pritisku na tipke za premikanje, ki jih definiramo v nastavitvah UE4. V primeru, da je karakterju premikanje dovoljeno, pridobimo

vektor, ki pove, v katero smer je karakter obrnjen, nato pa kličemo funkcijo, ki doda premikanje karakterju v to smer.




Slika 4.121: Implementirane glavne akcije: Napad 1, Napad 2, Skok in Počep

Na sliki [4.121] ima vsaka akcija svoj dogodek, ki se proži ob pritisku na tipko, določeno v nastavitvah. V kolikor ima karakter - ob pritisku na tipki za napad - na sebi orožje, spremenimo boolean vrednosti spremenljivk „Attack1 Button Pressed“ in „Attack2 Button Pressed“ na True. Ti spremenljivki prideta prav, ko želimo sporočiti Načrtu, odgovornemu za animacije, naj začne z izvajanjem ustrezne animacije. Na podoben način implementiramo še akcijo počepa. Ker ima naš Načrt za premikanje že vgrajeno logiko za skakanje, lahko za skakanje le preverimo, ali je tipka za skok pritisnjena, nato pa pokličemo funkcijo „Jump“, ali v nasprotnem primeru funkcijo „Stop Jumping“.




Slika 4.122: Implementacija dodajanja in odstranjevanja orožja

Dodajanje ali odstranjevanje orožja se izvede ob pritisku na tipko 1 (meč) ali 2 (ščit). V kolikor ima karakter ob pritisku na ustrezno tipko na sebi že orožje, to orožje odstranimo oziroma ga uničimo s klicem funkcije „DestroyActor“. Če pa na sebi še nima orožja, to orožje najprej postavimo v sceno s funkcijo „SpawnActor“ in ga nato pripnemo na določeno kost v okostju našega karakterja s funkcijo „AttachToComponent“. S tem se orožje giblje s karakterjem.

#### 4.10.4.2 Načrt animacije

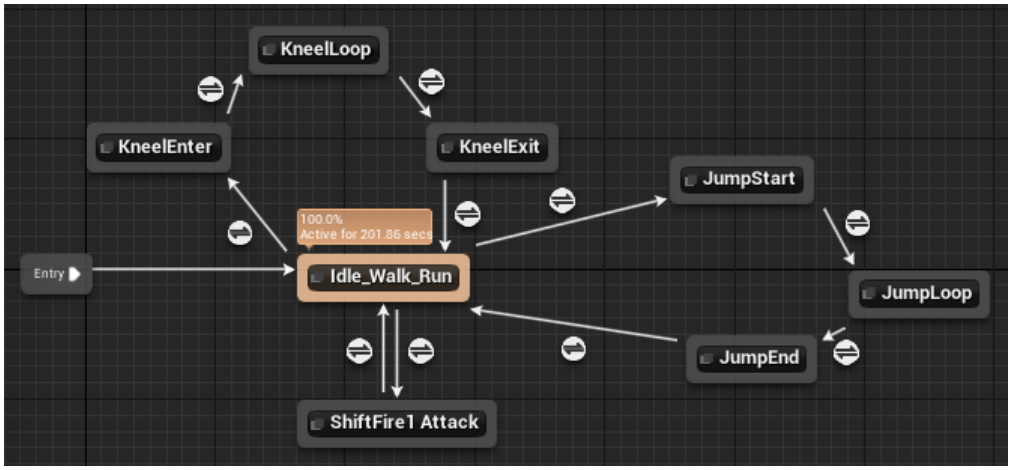
Da se vse naše animacije izvajajo ob ustreznem dogodku ali času, moramo zasnovati Načrt animacije. Načrt animacije ima dva dela. Prvi se imenuje Graf dogodkov (angl. „Event graph“), drugi pa Graf animacij (angl. „Anim graph“). V Grafu dogodkov z uporabo dogodkov in branjem spremenljivk, ki smo jih definirali v Načrtu karakterja vplivamo na potek animacij. Sam potek animacij pa zasnujemo v Grafu animacij.

Preden implementiramo Graf animacij, je priporočljivo, da imamo vse potrebne animacije pripravljene na uporabo. Za premikanje (nedejavno stanje, hoja in tek) uporabimo sredstvo imenovano „Blend Space 1D“. S tem lahko učinkovito mešamo med animacijami glede na izbrano spremenljivko, v našem primeru je to hitrost [4.123]. Poimenujmo ta „Blend Space“ z imenom „Idle_Walk_Run“.

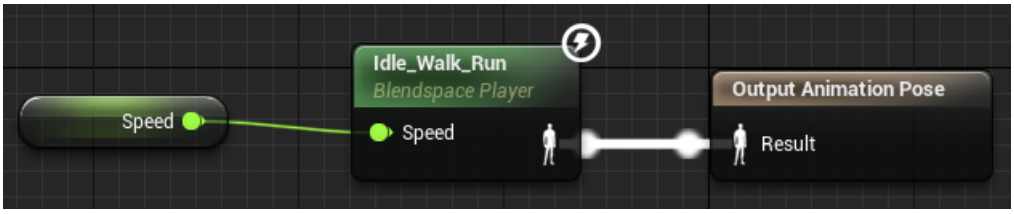

Slika 4.123: „Blend Space“. Prva točka na časovnici predstavlja animacijo nedejavnega stanja, druga hojo in tretja tek. Glede na hitrost karakterja, bo „Blend Space“ ustrezno menjal (mešal) med temi tremi animacijami.

Animacija Napad 1 [4.7.5] se lahko izvaja med ostalimi animacijami, kot sta na primer hoja in tek. To dosežemo v pogonu UE4 z uporabo animacijskih montaž (angl. Animation Montage). S tem bomo določili, da se animacija Napad 1 izvaja na zgornji polovici okostja (od hrbtenice gor), medtem ko se na spodnji polovici okostja lahko izvaja recimo hoja. Tako se bodo ob napadanju med hojo animacije ustrezno izvajale.

Na tem mestu je vse pripravljeno za Graf animacij [4.124]. Ob samem začetku izvajanja je smiselno, da začnemo v privzetem stanju, ki ga bo predstavljal „Blend Space“ imenovan „Idle_Walk_Run“, ki mu podamo spremenljivko hitrosti, ki bo vplivala na to, katera animacija se bo izvajala (nedejavno stanje, hoja ali tek) [4.125].




Slika 4.124: Graf animacij



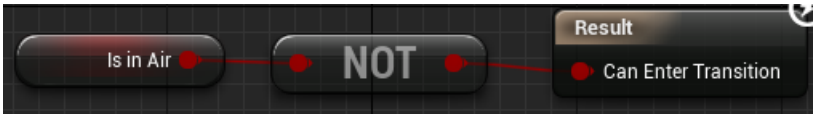

Slika 4.125: „Idle_Walk_Run“ vozlišču podamo spremenljivko „Speed“.

Ob pritisku na tipko za izvajanje animacije Napad 2 [4.7.6], spremenimo stanje iz „Idle_Walk_Run“ v „ShiftFire1 Attack“. V tem stanju začnemo z izvajanjem animacije Napad 2, ob koncu te animacije pa se vrnemo v privzeto stanje.

Ob pritisku na tipko za izvajanje animacije skoka se prevesimo v stanje „JumpStart“. Tu začnemo z izvajanjem animacije Odriv. Na tem mestu je potrebno definirati pogoj, ob katerem se prevesimo v naslednje stanje „JumpLoop“. V tem stanju se izvaja animacija Lebdenja. Pogoj zastavimo tako, da se prehod v stanje „JumpLoop“ začne, že malo preden se konča animacija Odriv [4.126].


Slika 4.126: Pogoj, ob katerem se prevesimo v stanje „JumpLoop“ iz stanja „JumpStart“.

Naslednje stanje v ciklu skoka je doskok ali „JumpEnd“. Z izvajanjem animacije Doskok začnemo, ko naš karakter ni več v zraku [4.127].



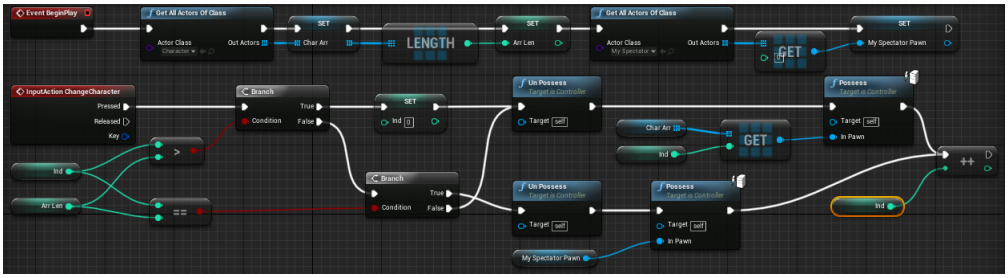

Slika 4.127: Pogoj, ob katerem se prevesimo v stanje „JumpEnd“ iz stanja „JumpLoop“.

Ob koncu skoka se je potrebno vrniti v privzeto stanje „Idle_Walk_Run“. Ta pogoj zastavimo podobno kot pri začetku animacije Lebdenje. Da je prehod med animacijami gladek, začnemo s prehodom v naslednje stanje že med izvajanjem trenutne animacije [4.128].


Slika 4.128: Pogoj, ob katerem se prevesimo v stanje „Idle_Walk_Run“ iz stanja „JumpEnd“.

#### 4.10.4.3 Načrt kontrolerja

Načrt kontrolerja je odgovoren za menjavanje med karakterji v sceni.




Slika 4.129: Implementacija kontrolerja, ki menjava med karakterji v sceni.

Ob začetku izvajanja kontroler najprej poišče vse karakterje v sceni in njihove reference shrani v niz (angl. array). Poleg vseh karakterjev shrani še referenco na kamero, ki nam služi kot gledalec (angl. spectator) in s katero se lahko poljubno sprehajamo po sceni.

Ko pritisnemo na tipko, ki je zadolžena za menjavo karakterjev (v našem primeru smo izbrali tipko TAB), se sproži dogodek „ChangeCharacter“. Ob tem dogodku najprej preberemo spremenljivko „Ind“, glede na katero izbiramo med karakterji v nizu. Za samo menjavo poskrbi funkcija Possess.

## 4.11 Končni izdelek

Slika 4.130: Scena 1

Slika 4.131: Scena 2

Slika 4.132: Karakter 1

Slika 4.133: Karakter 2

Slika 4.134: Karakter 3

Slika 4.135: Scena 3

Slika 4.136: Scena 4

Slika 4.137: Scena 5

Slika 4.138: Scena 6

# Poglavje 5

# Zaključek

V diplomski nalogi smo uspešno rešili zastavljeno nalogo, to je predstaviti celoten postopek izdelave igralnega karakterja, vključno s teksturiranjem in animacijo. Z uporabo več različnih programskih orodij smo določene naloge lahko opravili bolj učinkovito kot sicer.

Izdelava 3D igralnega karakterja obsega več različnih področij dela. Vsako izmed njih smo skušali čimbolj jedrnato opisati in predstaviti na praktičnem primeru. Bolj realistične rezultate bi lahko dosegli, če bi se osredotočili na eno samo področje in imeli pri tem nekaj več izkušenj.

Menim, da lahko ta diplomska naloga služi vsakemu, ki se podaja v 3D oblikovanje in animacijo. Pri tem je lahko v pomoč kot vir informacij ali pa le kot vodilo, katerega se pri delu poskušamo držati.

# Literatura


[1] Blender dokumentacija. Dosegljivo: https://docs.blender.org/manual/en/latest/. [Dostopano: 24. 01. 2019].

[2] Model iz igre Tom Clancy's Rainbow Six Siege. Dosegljivo: https://www.artstation.com/artwork/0eKg8. [Dostopano: 24. 01. 2019].

[3] Modeli iz igre Total War Battles: Kingdom. Dosegljivo: https://www.artstation.com/artwork/9Nv4Q/. [Dostopano: 24. 01. 2019].

[4] Ocean, dodatek za Unreal Engine 4. Dosegljivo: https://forums.unrealengine.com/community/work-in-progress/1519072-community-project-free-ocean-water-shader. [Dostopano: 24. 01. 2019].

[5] Senčenje na osnovi fizike. Dosegljivo: https://www.allegorithmic.com/pbr-guide. [Dostopano: 24. 01. 2019].

[6] Unreal Engine 4 dokumentacija. Dosegljivo: https://docs.unrealengine.com/en-us/. [Dostopano: 24. 01. 2019].

[7] Richard E. Williams. *The Animator's Survival Kit.* Faber & Faber, 2009.