VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

# Procedūrinė humanoidų animacija

Atliko: II kurso, 9 grupės studentai:

Kęstutis Gustaitis (parašas)

Andrius Tamulionis (parašas)

Osvaldas Grigas (parašas)

Darbo vadovas:

dr. Rimvydas Krasauskas (parašas)

Recenzentas:

Tomas Sakalauskas (parašas)

Vilnius
2007

**Turinys**

# Įvadas

Žmogus jau išmoko deklaratyviai užrašyti muziką. Mes turime penklinę, taktus, natas, diezus, bemolius ir kitus specialiuosius žymenis, kurių pagalba galime keisti garsų toną, skambėjimo trukmę, garsumą ir t.t. Pagal susitarimą žinome, ką muzikinėje gamoje reiškia natos "do", "re", "mi", taip pat žinome, kad aštuntinė skamba dvigubai trumpiau nei ketvirtinė. Todėl užrašydami muziką mes operuojame ne garsų dažniais, decibelais ar milisekundėmis, bet abstraktesniais vienetais - deklaratyviais žymenimis, po kuriais slepiasi visos techninės detalės. Tokią notaciją sėkmingai pavyko perkelti į kompiuterinę erdvę - buvo sukurtas MIDI formatas.

Egzistuoja panašus poreikis ir formaliai žmogaus judesių notacijai. Choreografas nori užrašyti šokį, kineziterapeutas - gydomąją judesių programą, aerobikos instruktorius - rytinę mankštą ir pan. Yra pasiūlyta ne viena tokia notacija. Populiariausiųjų autoriai yra Rudolfas Benešas (*Benesh movement notation*) ir Rudolfas Labanas (*Labanotation*). Šių žmonių sumąstyti metodai ir simbolika dažniausiai naudojami choreografijoje. [DN] Tačiau jie nėra kompiuterizuoti. Pavyzdžiui, programinė įranga, gebanti redaguoti Labano notaciją, tarnauja tik kaip teksto redaktorius. T.y. nors ir yra sukurti specialūs žymenys įvairių žmogaus kūno dalių padėčiai erdvėje užrašyti, choreografas visvien turi žinoti, ką kiekvienas žymuo reiškia, kad galėtų įsivaizduoti, kaip atrodys galutinis rezultatas - šokis. [LAB] Kitaip sakant, trūksta šokio vizualizacijos. Tai prilygsta situacijai, kai nebuvo kompiuterių, visa muzika buvo rašoma popieriuje, o savo kūrinį kompozitorius galėjo išgirsti tik tuomet, kai jį atlikdavo orkestras.

Pagrindinis mūsų darbo tikslas - rasti priemones (ir, esant poreikiui, sukurti naujas), kurios leistų paprasčiausiu būdu, kuris būtų suprantamas žmogui ir kurį galėtų apdoroti kompiuteris, užrašyti žmogaus kūno judesių programą. Šiam tikslui reikės sukurti humanoido animacijų kompiuterinį formatą, kuris leistų abstrahuoti žmogaus judesius, juos apjungti į sekas ir kaupti judesių bibliotekas. Tokio tipo animacijas vadinsime "procedūrinėmis". Panagrinėkime, pavyzdžiui, šokį arba rytinę mankštą. Tai yra judesių programos – sudėtingos judesių sekos. Mūsų formatas turi leisti kurti panašias programas, pasitelkiant judesius, sukauptus bibliotekoje. Iš techninės pusės formatas turi būti nepriklausomas nei nuo humanoidų išvaizdos, t.y. nuo jų tinklelio (mesh) geometrijos, nei nuo vidinės virtualaus skeleto struktūros. Judesių programos turi būti nesunkiai, intuityviai užrašomos ir redaguojamos paprastomis priemonėmis (pvz., teksto redaktoriumi). Žinoma, sukurta judesių programa būtų bevertė, jei nebūtų galimybės ją atvaizduoti, naudojant animuotą humanoido figūrą.

## Darbo tikslai

- Išnagrinėti esamus formatus bei technologijas, naudojamas humanoidų modeliams kurti ir juos animuoti. Pasirinkti tas, kurios labiausiai tinka mūsų užduočiai.
- Panaudoti esamus arba sukurti keletą naujų humanoidų modelių ir animacijų (pavyzdžiui: ėjimo, bėgimo, šuolio). Realizuoti glodų, vizualiai patrauklų animacijų apjungimą.
- Sukurti nuo humanoido modelio nepriklausantį formatą, kuris leistų procedūriškai jungti animacijas bei kaupti jas judesių bibliotekoje.
- Sukurti procedūrinių animacijų peržiūros programą.

# 1. Egzistuojančių priemonių apžvalga

Norėdami pasiekti užsibrėžtus tikslus, nutarėme kiek įmanoma išnaudoti jau sukurtus sprendimus bei priemones. Jų ieškojome tarp egzistuojančios programinės įrangos, jos taikymų, pripažintų *web* standartų, trimačių modelių ir animacijų formatų: tiek atvirų, tiek specialios paskirties. Šiame skyriuje trumpai aprašysime priemones, kurias savo nuožiūra atrinkome kaip tinkamiausias mūsų uždaviniui.

## 1.1. XML

Šiame darbe bus plačiai naudojamas XML (Extensible Markup Language) duomenų formatas. [XML] Jis labiausiai tinka struktūrizuotų, ypač hierarchinio pobūdžio duomenų saugojimui ir perkėlimui tarp pačių įvairiausių platformų bei operacinių sistemų. XML'e saugomais duomenimis lengva manipuliuoti: persiųsti, nuskaityti, redaguoti, konvertuoti ir t.t. Savo ruožtu XML remiasi Unicode standartu ir atitinkamai paveldi visus jo teikiamus daugiakalbystės privalumus.

Tiesa, šis formatas, kad ir lankstus, turi vieną trūkumą: trimačių modelių geometrijos užrašymas ir jų animacijų notacija paprastai reikalauja operuoti dideliais duomenų kiekiais - tokių laikmenų užimama vieta atmintyje gali labai išaugti, ypač tuomet, kai jos saugojimui naudojama tokia deklaratyvi tekstinė notacija kaip XML. Paprasčiausias, nors ir ne pats efektyviausias sprendimas būtų XML failų suspaudimas, tarkim, *gzip* formatu. Žvelgiant į ateitį, tinkamas kandidatas XML duomenų efektyviam saugojimui galėtų būti "Efficient XML" formatas [AGILE]. Jis yra dvejetainis, pilnai suderinamas su tekstiniu XML ir gali užimti nuo kelių iki kelių šimtų kartų mažiau vietos darbinėje atmintyje bei kietajame diske, kartu užtikrinant greitesnį duomenų apdorojimą. Efficient XML šiuo metu nėra standartas, tačiau W3C (World Wide Web Consorcium) jį išrinko iš visų "binarinio XML" kandidatų ir ruošiasi standartizuoti. [EXI]

## 1.2. XSLT

Realizuojant procedūrines animacijas, reikalingos kelios transformacijos iš vieno failų formato į kitą. Kadangi įvairių duomenų kodavimui nutarėme kuo plačiau naudoti XML sintaksę, natūralus sprendimas duomenų transformavimui XSLT (Extensible Stylesheet Language Transformations) [XSL]. Tai yra W3C standartas, leidžiantis deklaratyviai aprašyti vienos XML struktūros duomenų transformaciją į kitą XML struktūrą, į HTML arba į

paprastą tekstinį dokumentą. XSLT failas savo ruožtu pats naudoja XML sintaksę. Įvairioms programavimo kalboms yra sukurtos arba standartinės, arba trečiųjų šalių atviro kodo bibliotekos, mokančios atlikti XML dokumentų transformacijas su XSLT.

### 1.3. OGRE

Tai viena pagrindinių priemonių mūsų darbe. Ją naudosime galutiniam rezultatui - humanoido animacijai - atvaizduoti. OGRE (Object-Oriented Graphics Rendering Engine) yra lankstus trimatės grafikos variklis, parašytas C++ programavimo kalba taip, kad palengvintų darbą ir padarytų intuityvų aplikacijų kūrimą panaudojant aparatūriškai akseleruotą trimatę grafiką [OGRE]. Klasių biblioteka abstrahuoja visas sistemos naudojamų bibliotekų (arba Direct3D, arba OpenGL) detales, tokiu būdu suteikdama intuityvesnę programavimo sąsają (API), paremtą objektine paradigma. Nors OGRE dažnai naudojamas kuriant trimačius žaidimus, tai nėra žaidimų variklis. Jis pritaikomas visur, kur reikalinga sparti trimačių objektų vizualizacija.

OGRE turi daug pranašumų, lyginant su kita programine įranga. Pirmiausia, tai plačiausiai naudojamas atviro kodo grafikos variklis. Nemokamai platinamas pagal LGPL licenciją, todėl puikiai tinka mūsų darbui. Dauguma kitų, kad ir komercinių, techniškai įspūdingų variklių, vis gi nėra lengvai suprantami, jiems dažnai trūksta išsamios dokumentacijos, kad būtų galima efektyviai juos panaudoti. Daugelis jų turi ilgą galimybių sąrašą, kuris neturi vientiso vaizdo ir atrodo kaip padrikas funkcijų rinkinys. Tuo tarpu OGRE, dėl savo populiarumo ir atviro kodo, yra nuolat aktyviai vystomas, kartu stengiantis išlaikyti vientisą architektūrą. Kiekviena diegiama nauja galimybė yra kruopščiai apsvarstoma, kad kiek įmanoma grakščiau įsilietų į bendą funkcijų visumą, ir pilnai dokumentuojama.

Variklis naudoja lanksčią klasių hierarchiją ir vartotojui leidžia kurti komponentus, kurių pagalba galima pritaikyti OGRE savo poreikiams. Tarkim, norint pavaizduoti uždaros erdvės (*indoor*) scenas, galima naudotis esamu BSP/PVS komponentu. Norint pakeisti uždarą aplinką atvira (pvz., kambarį kraštovaizdžiu) panaudojamas kitas komponentas, o likusi programos dalis lieka nepakeista.

Taigi trumpai galima būtų pasakyti, kad mūsų apsisprendimą nulėmė variklio kūrime naudojamas principas "kokybė vietoj kiekybės", jo platinimo licencija, lankstumas ir aiški dokumentacija bei dėl šių priežasčių egzistuojanti nemaža OGRE vartotojų bendruomenė, į kurią galima kreiptis patarimų.

### 1.4. OGRE failų formatai ir XMLConverter

Modelio atvaizdavimui ir animacijai naudojamas atviro kodo trimatės grafikos variklis OGRE bei šiam varikliui suprantami formatai: 3D modelio geometrijos formatas OGRE MESH ir skeleto animacijos formatas OGRE SKELETON. Minėtieji formatai gali būti užrašomi patogesne XML forma - taip gauname visus XML privalumus: įskaitant galimybę XSLT priemonėmis konvertuoti iš OGRE formatų į kitus, ir atvirkščiai. Tačiau atvaizduodamas animaciją OGRE naudoja dvejetainę formą. Konvertavimo tarp XML ir dvejetainės formos problemą išsprendžia specializuotas OGRE įrankis XMLConverter. Tai išorinė programa, kurios pagalba MESH ir SKELETON failus, užrašytus XML forma, galima konvertuoti į dvejetainę formą. Galima ir atvirkštinė konversija. Kadangi XMLConverter nemokamai platinamas kartu su OGRE grafikos varikliu, dažnai naudojome jį savo darbe.

SKELETON formatui keliame tokį reikalavimą: visi humanoido sąnariai ir kaulai privalo turėti tokius pavadinimus, kokie specifikuoti H-Anim standarte (žr. toliau). Kaulų-sąnarių hierarchija taip pat turi atitikti H-Anim standartą. Toks reikalavimas būtinas, norint visiškai atskirti modelio geometriją nuo animacijos, t.y. norint turėti galimybę kaupti animacijų bibliotekas, nepriklausomas nuo konkrečių modelių.

### 1.5. Alternatyvos OGRE failų formatams

Sprendimą naudoti OGRE formatus humanoidų modelių geometrijai bei animacijai natūraliai lėmė tai, kad atvaizdavimui pasirinkome OGRE grafikos variklį. Tokio sprendimo akivaizdus trūkumas - šie formatai yra suprantami tik OGRE variklio. Norėtųsi turėti bendresnės paskirties formatą, kurį palaikytų platesnė trimatės grafikos dizainerių bei programuotojų bendruomenė. Teko susipažinti su keliomis alternatyvomis. Patraukliausiai atrodo du variantai: COLLADA ir X3D. Abu jie yra ISO standartai ir abu naudoja XML sintaksę.

#### *1.5.1. COLLADA*

COLLADA (COLLAborative Design Activity) [COL] yra *Khronos* grupės darbo vaisius. Į šią grupę įeina daug stambių vaizdo plokščių, kitos kompiuterinės ir buitinės įrangos gamintojų, trimačio modeliavimo, komercinių žaidimų ir kitos programinės įrangos kūrėjų. [KHR] COLLADA buvo kurtas siekiant turėti bendrą formatą interaktyvioms 3D aplikacijoms. Šiam formatui jau sukurti eksportavimo įrankiai populiariausioms modeliavimo

programoms: 3D Studio Max, Maya, Softimage XSI, Blender ir kt. Kita vertus, formato ambicingas tikslas - vienoje vietoje išsaugoti visą informaciją ir meta-informaciją apie trimačius modelius, animaciją, apšvietimą, kameras, šešėliavimo algoritmus ir t.t., kad būtų galima tą pačią 3D sceną redaguoti skirtingose programose neprarandant duomenų - gerokai viršija mūsų uždavinio poreikius. Dėl to COLLADA failai paprastai būna gan stambūs ir gremėzdiški. Mūsų nagrinėjamos probleminės srities atžvilgiu jie kupini pašalinės informacijos.

### *1.5.2. X3D*

X3D atrodo labiau tinkamas mūsų užduočiai [X3D]. Tai kažkada buvusio populiariaus, bet dabar naudojamo tik specifiniuose taikymuose VRML formato atžala. Jį sukūrė Web3D konsorciumas [WEB3D]. X3D, nors ir sukurtas VRML pagrindu, lenkia savo pirmtaką daugelyje sričių. Svarbiausia - jis gali būti užrašomas naudojant XML sintaksę. X3D taip pat leidžia klasikinę VRML sintaksę (kurios mes nenaudosime) bei dvejetainę formą - pastaroji galėtų pasitarnauti stambių failų kompresijai. Šis formatas taip pat atneša daug naujovių, kurių neturėjo VRML, įskaitant ir humanoidų animaciją. Apskritai, jo funkcijų pilnai pakanka mūsų uždaviniui. Priešingai nei COLLADA, X3D yra optimizuotas duomenų perdavimui internetu, o minėtoji dvejetainė forma leidžia jį dar labiau suspausti.

## 1.6. H-Anim

Mūsų sukurtoje programoje animacijas bus galima taikyti keliems modeliams - humanoidams. Iš esmės animacija yra taikoma ne modelio geometrijai, bet skeletui. Dėl to visų modelių kaulų skeletų struktūra turi būti vienoda.

Kompiuterinėje grafikoje neapsimoka kurti humanoido skeleto struktūros tokios, kokia yra realaus žmogaus. Tai pernelyg kruopštus darbas, be to, išoriškai nėra matomas kiekvieno kaulo judėjimas. Todėl yra priimtas ISO/IEC 19774 standartas, dar vadinamas H-Anim (Humanoid Animation) [HANIM]. Standartas aprašo humanoido kaulų ir sąnarių hierarchiją. Jis nėra pririštas prie konkrečios vaizdavimo technologijos, ir sukurtas būtent tam, kad modeliuotojai ir animacijų kūrėjai galėtų dirbti nepriklausomai vieni nuo kitų, naudodami įvairias modeliavimo ir animacijos kūrimo technologijas. H-Anim standartą naudojame savo modeliuose dar ir dėl to, kad jį sukūrė tas pats Web3D konsorciumas, todėl H-Anim yra integruotas į X3D.

### 1.7. Modeliavimo programos

Humanoidų modelių kūrimui galima pasitelkti įvarias modeliavimo programas: Milkshape3D, CharacterFX, Blender, 3D Studio Max, Maya ir kt. Daugeliui jų egzistuoja OGRE eksporteriai. Jie sukuria dvejetainius MESH ir SKELETON failus, kur atitinkamai saugomas pats modelis (MESH) ir jo animacija (SKELETON). Šiuos failus grafikos variklis naudoja užkraudamas modelį ir jo animaciją. Vystydami grafikos variklį, jo kūrėjai nuolat tobulina ir eksporterius. Mes išbandėme humanoido modelį, sukurtą 3D Studio Max modeliavimo programa, eksportuoti, įkelti į OGRE ir animuoti. Kadangi visa tai įgyvendinti pavyko, modelių kūrimo procese rekomenduojame naudoti trimačio modeliavimo programas.

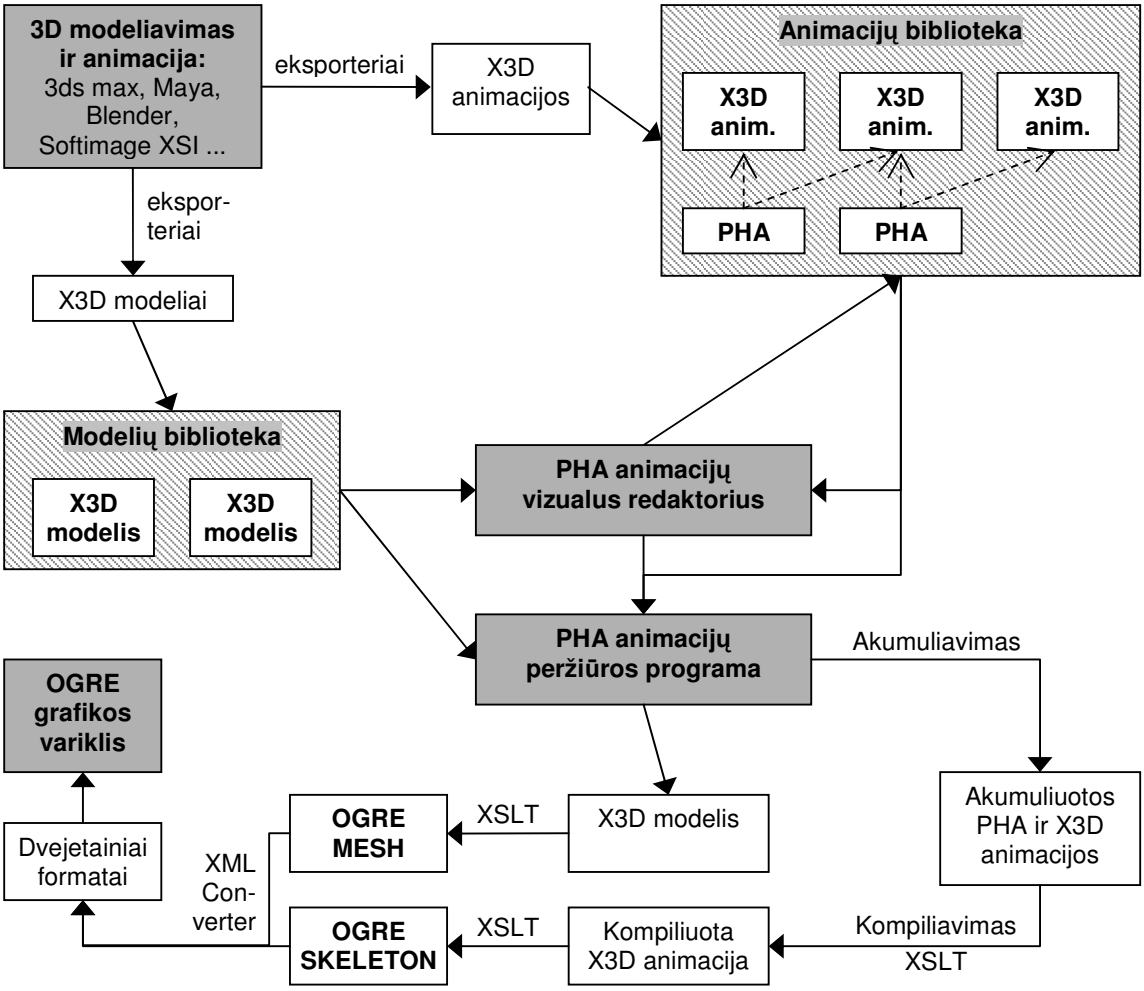
# 2. Procedūrinių animacijų schema

Kiekviena iš prieš tai aprašytų priemonių (išskyrus COLLADA) atlieka savo rolę mūsų sumanytoje procedūrinių animacijų kūrimo, kaupimo ir atvaizdavimo schemoje. Tačiau vien jų nepakanka. Reikalinga formali notacija, kuri leistų kombinuoti animacijas į judesių sekas ir jas procedūriškai abstrahuoti. Tai yra vienintelis formatas, kuriam neradome egzistuojančio atitikmens, todėl teko jį sukurti patiems. Tokias procedūrines humanoidų animacijas sutrumpintai vadinsime PHA. Jas detaliau aprašėme 4-ame skyriuje "Procedūrinių animacijų formatas".

## 2.1. Pradinė schema

Pirmiausia pateiksime tokią schemą, kokią ruošėmės įgyvendinti darbo pradžioje. Ją sudarėme susipažinę su esamomis animacijų technologijomis ir norėdami laikytis galiojančių standartų, kad mūsų procedūrinės animacijos būtų kuo lankstesnės ir kuo plačiau pritaikomos. Ši schema apima visą darbo procesą, pradedant modelių bei animacijų kūrimu ir baigiant procedūriškai komponuotų animacijų atvaizdavimu. Pažvelkime į ją iš arčiau.

- Tiek trimačiai humanoidų modeliai, tiek ir jų žemo lygio (ne procedūrinės) animacijos kuriamos su egzistuojančia specializuota programine įranga: 3ds max, Maya, Blender ar kt.. Tokiu būdu grafikos dizaineriai gali išnaudoti savo turimus įgūdžius. Vis dėlto reikalajama, kad modeliavimo programa gebėtų eksportuoti animacijas ir modelius X3D formatu, nes būtent jį pirmiausia buvome pasirinkę kaip mūsų animacijų redaktoriaus bei peržiūros programų įvesties formatą.
- Eksportuoti modeliai kaupiami modelių bibliotekoje, o eksportuotos animacijos - animacijų bibliotekoje. Šiame kontekste X3D animacijos yra vadinamos "žemo lygio", nes jos tiesiogiai judina humanoido skeleto kaulus. Tuo tarpu toje pačioje bibliotekoje kaupiamos ir "aukšto lygio" animacijos, kurios sudarytos iš kreipinių į žemo lygio animacijas bei kitas aukšto lygio animacijas. Tokias vadinsime kombinuotomis, procedūrinėmis arba sutrumpintai PHA animacijomis.
- Vizualus redaktorius, kurį planavome sukurti, leistų redaguoti turimas PHA animacijas bei kurti naujas ir išsaugoti jas bibliotekoje. Redaktorius dirbtų aukštesniame (procedūriniame) lygyje, t.y. juo nebūtų galima judinti individualių skeleto kaulų ar keisti turimų X3D animacijų. Rezultatus būtų galima stebėti ekrane po kiekvieno pakeitimo.




**Schema 1. Pradinė procedūrinių animacijų schema**

- Procedūrinių animacijų peržiūros programa 3D humanoidų modelių vizualizacijai naudoja OGRE grafikos variklį. Kadangi OGRE reikalauja specifinių, tik jai suprantamų įvesties duomenų formatų, reikia atlikti keletą transformacijų:
  - Pirmiausia reikia sukompiliuoti procedūrines animacijas. Kompiliacija suvokiama kaip dviejų žingsnių procesas. Pirmiausia identifikuojami visi PHA ir X3D failai, kurie reikalingi pasirinktai animacijai vizualizuoti (PHA failai gali "kviesti" bet kurią bibliotekoje esančią animaciją). Visos reikalingos procedūros bei žemo lygio animacijos sukeliamos į vieną ilgą XML failą. Tai vadiname akumuliavimu. Antrame žingsnyje visos procedūrinės animacijos "išskleidžiamos" iki žemiausio lygmens: akumuliuotas XML failas transformuojamas į dar ilgesnį žemo lygio X3D animacijų seką. Šioje

schemoje transformacijai iš vienos XML struktūros į kitą numatėme naudoti XSLT šablonus.

- Sukompiliuota ("išskleista") X3D animacija dar kartą transformuojama - šį kartą jau su kitu XSLT - į OGRE SKELETON XML formatą. Atitinkamai pasirinktas X3D modelis transformuojamas (taip pat su XSLT) į OGRE MESH XML formatą.
- Abu OGRE XML formatai konvertuojami į savo dvejetainius atitikmenis, naudojant specializuotą OGRE įrankį XMLConverter.
- Gauti dvejetainiai failai paduodami OGRE varikliui, kuris atvaizduoja kombinuotą humanoido animaciją.

## 2.2. Pradinės schemos realizacijos kliūtys

Pirmoji schema išėjo gana sudėtinga, nes reikalavo daug transformacijų tarp skirtingų duomenų formatų. Pabandę ją realizuoti, susidūrėme su rimtomis kliūtimis. Daugelis jų susiję su tuo, kad modelių ir animacijų formatu pasirinkome X3D.

### *2.2.1. X3D eksportavimas iš modeliavimo įrankių*

Trūksta X3D eksporterių iš populiarių 3D modeliavimo įrankių. Kai kurioms modeliavimo programoms tokių eksporterių apskritai nepavyko rasti. Tie, kuriuos radome ir išbandėme, arba yra mokami (o bandomoji versija nėra pilnai funkcionali), arba neveikia, arba veikia netaisyklingai, t.y. eksportuoja tokį X3D turinį, kuris ne iki galo suderinamas su standartu. Blogiausia tai, kad nei vienas iš jų nepalaiko H-Anim standarto. Išimtis - Flux Studio [FLUX], tačiau tai yra specializuota, gana siauro profilio modeliavimo programa, pritaikyta būtent X3D ir VRML formatams, todėl ji nėra plačiai naudojama trimatės grafikos dizainerių. Taip pat bandėme eksportuoti į X3D pirmtaką - VRML formatą, nes šis standartas yra senesnis ir plačiau palaikomas egzistuojančiuose modeliavimo įrankiuose, tokiuose kaip 3D Studio Max, Maya ir Blender. Tačiau ir čia iškilo panašios problemos - ne visi eksporteriai dirbo kokybiškai. Be to, eksportuotą VRML failą dar reikėjo transformuoti į X3D, ko specializuoti konvertavimo įrankiai dėl nesuprantamų priežasčių nesugebėjo padaryti.

### *2.2.2. Veikiančių H-Anim modelių ir animacijų trūkumas*

Testavimui mums reikėjo X3D humanoidų. Negalėdami jų eksportuoti iš turimų modeliavimo programų, ieškojome internete. Radome labai nedaug. Iš tų, kuriuos aptikome, ne visi laikosi H-Anim standarto. Tų, kurie laikosi, negalėjome naudoti, nes tą draudė jų naudojimo ir platinimo sąlygos. Susidarėme įspūdį, kad susidomėjimas X3D ir H-Anim (išskyrus akademinį) yra kol kas labai menkas.

### *2.2.3. X3D transformavimas į OGRE*

Nėra įrankių konvertuoti X3D į OGRE formatą. Tiesa, pradinėje schemoje buvo numatyta, kad šią transformaciją atliksime mes patys. Tačiau vien faktas, kad plati OGRE atviro kodo programuotojų bendruomenė iki šiol nepasiūlė tokio įrankio (nors yra sukurti konverteriai ir eksporteriai iš daugybės kitų populiarių formatų), privertė susimąstyti. Tokio konverterio projektas buvo kažkieno pradėtas, tačiau dėl neaiškių aplinkybių numarintas. Atidžiau panagrinėję X3D ir OGRE formatus, įsitikinome, kad jie iš tiesų yra labai skirtingi, ir transformacija iš vieno į kitą būtų visai netriviali.

- X3D labiau pritaikytas interakcijai tarp vartotojo ir virtualaus pasaulio. Juo galima programuoti ryšius tarp trimačių bei abstrakčių (pvz, laiko) objektų pozicijos, elgsenos ir kitų atributų. X3D aprašomas "pasaulis" yra savarankiškas, jo komponentai geba reaguoti į vartotojo veiksmus ar aplinkos pasikeitimus. Animacijų notacija bei modeliavimo struktūros čia yra labai deklaratyvios.
- OGRE saugo modelius ir animacijas kaip statinius resursus. Palyginti su X3D, tai labai žemo abstrakcijos lygio formatas. Jis operuoja tik tokiais vienetais kaip tinklelio geometrija (*mesh*), kaulai ir paprasčiausi animacijos primityvai (*keyframes*).

Padarėme išvadą, kad ne X3D, o kaip tik OGRE yra tinkamesnis kaip žemo lygio formatas aukštesnio lygio procedūrinėms animacijoms.

### *2.2.4. Akumuliavimo ir kompiliavimo žingsniai*

Idėja kombinuoti visas procedūriškai kviečiamas animacijas į vieną ilgą X3D failą, o tada jį vienu ypu transformuoti į OGRE, ir viską atlikti su XSLT, turi didelį trūkumą. Šio trūkumo nepastebėjome, kol nepradėjome galvoti, kaip glodžiai apjungti gretimas animacijas. Toliau šiame darbe aptariamos įvairios interpoliavimo galimybės, tačiau jose naudojama kvaternionų

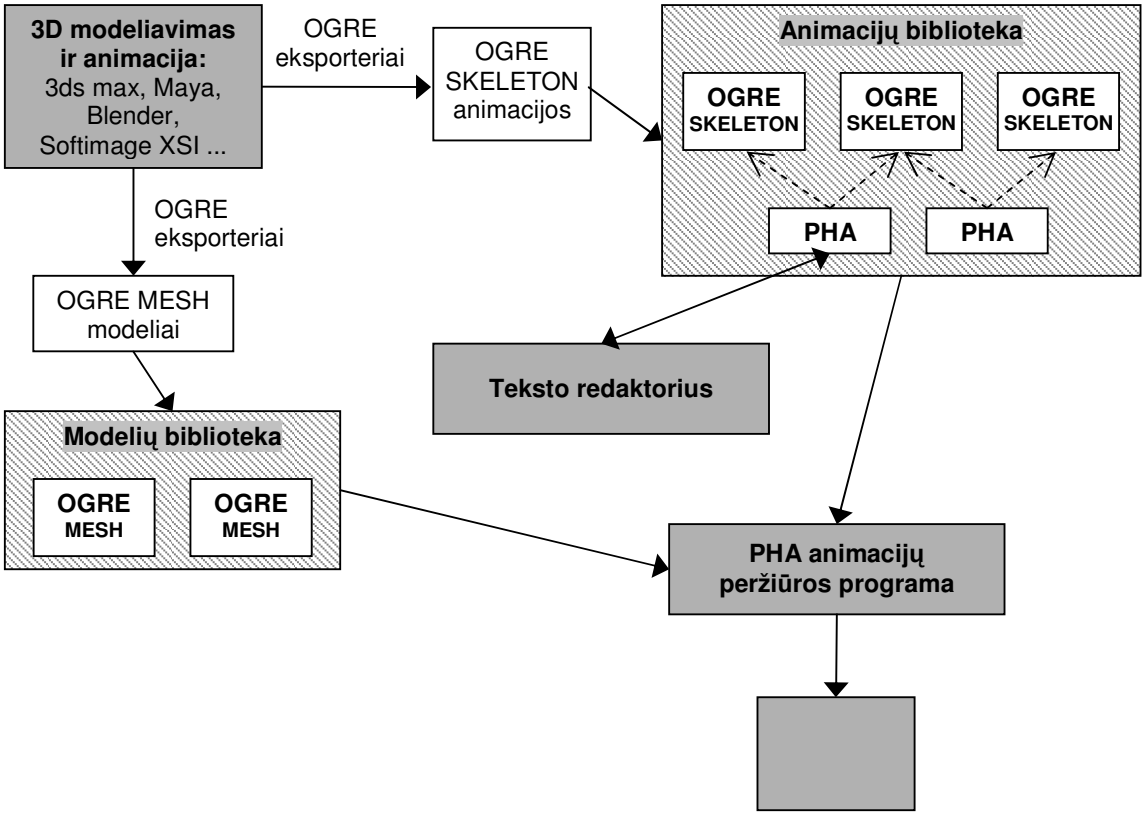
aritmetika reikalauja animacijų apjungimą atlikti programiškai. Deklaratyvios XSLT transformacijos tam visiškai netinkamos.

### 2.3. Koreguota schema

Galiausiai apsisprendėme atsisakyti X3D kaip žemo lygio animacijų formato ir pakeisti jį OGRE SKELETON. Kartu modelių bibliotekoje vietoj X3D atsirando OGRE MESH. Po tokio pakeitimo mūsų schema gerokai supaprastėjo. Iš jos buvo pašalintos kai kurios transformacijų grandys, nes dabar animacijos saugomos bibliotekoje tokiu formatu, kad be papildomų žingsnių gali būti nuskaitytos ir atvaizduotos mūsų peržiūros programoje. Nebeliko ir eksportavimo problemos, nes visiems populiariems trimačio modeliavimo įrankiams yra sukurti gerai veikiantys OGRE eksporteriai. Būtent tokią schemą mums ir pavyko įgyvendinti.

Akivaizdus koreguotos schemos trūkumas - ji pririšta prie OGRE modelių bei animacijų formato, kuris yra suprantamas tik vieno grafikos variklio. Sprendimą, kaip jau minėta, lėmė tai, kad OGRE formatas, nors ir turėdamas mažiau galimybių, šiuo metu yra plačiau naudojamas ir geriau integruotas į populiarius modeliavimo įrankius nei X3D. Vis dėlto suprantame poreikį naudoti pripažintus standartus, todėl X3D laikome rekomendacija ateičiai.

Koreguotoje schemoje taip pat atsisakėme procedūrinių animacijų kūrimo bei redagavimo programos, nes žmogus, kuriam buvo patikėta užduotis sukurti tokią programą, paliko mūsų darbo grupę. Vietoj to mes dėjome pastangas sukurti kuo paprastesnį ir suprantamesnį PHA formatą, kad procedūrines animacijas būtų galima nesunkiai užrašyti bet kokiu teksto redaktoriumi.




**Schema 2. Koreguota procedūrinių animacijų schema**

# 3. Procedūrinių animacijų schemos realizacija

## 3.1. Modelių atskyrimas nuo animacijų

Mūsų schemoje modeliai ir animacijos kaupiami atskirose bibliotekose. OGRE įrankiai iš modeliavimo programos eksportuoja du failus: OGRE MESH su plėtiniu *.mesh*, kuriame yra modelio tinklelis, ir OGRE SKELETON su plėtiniu *.skeleton*, kuriame saugoma modelio kaulų struktūrą bei kaulų animacijos. Taigi atskyrimas įmanomas per skeletą. Modelio geometrija (tinklelis) pririšta prie nematomo skeleto, kurio kaulams ir sąnariams suteikiami vardai. Animacijose aprašoma, kaip juda tie sąnariai ir kaulai, o peržiūros programa pagal skeleto judesius atitinkamai judina modelio "odą" (t.y. tinklelį). Kad modelio atskyrimas nuo animacijos būtų universalus, t.y. kad būtų galima keliems modeliams taikyti tą pačią animaciją, reikalinga standartinė, visiems modeliams vienoda skeleto struktūra ir bendri kaulų bei sąnarių pavadinimai. Būtent tokią kaulų hierarchiją apibrėžia H-Anim standartas, kuriuo mes ir naudojamės.

## 3.2. Reikalavimai modeliams

Kaip jau buvo minėta anksčiau, tam, kad būtų galima užkrauti animaciją skirtingiems modeliams, visi naudojami modeliai turės atitikti tokius reikalavimus:

- Humanoidas modeliuojamas stovėjimo pozicijoje, žvelgiantis teigiama Z kryptimi, į viršų rodo Y ašis, o X ašis atitinka humanoido kairę;
- Koordinačių pradžios taško (0, 0, 0) padėtis humanoido atžvilgiu privalo būti jo žemės (atramos) lygyje, tarp modelio pėdų;
- Pėdos viena nuo kitos nutolusios klubų plotį atitinkančiu atstumu ir pilnai liečia žemę (nepasistiebęs). Pėdos apačia nuliniame aukštyje (Y=0). Rankos tiesios ir nuleistos žemyn, delnai nukreipti į vidinę pusę. Plaštaka turi būti plokščia, t.y. delnas ir pirštai lygiagretūs Y ašiai, išskyrus nykščius, kurie nukreipti 45 laipsnių kampu teigiama Z kryptimi. Koordinačių sistema (X, Y ir Z kryptys) kiekvienam nykščio sąnariui išlieka tokia pati, kaip ir visam modeliui (globali).
- Modelio burna užčiaupta, akys plačiai atmerktos;
- Humanoidas turėtų atitikti realius žmogaus išmatavimus. Dydžiai nurodomi metrais. Įprastas modelio ūgis – 1,75 metro.

Tokioje padėtyje visų sąnarių kampai turėtų būti nuliniai. Kitaip tariant, visuose *rotation* (sukimo) sąnarių laukuose privalo būti reikšmės pagal nutylėjimą (0 0 1 0), taip pat ir *translation* (postūmio) laukuose reikšmės (0 0 0), o *scale* (mastelio keitimo) faktoriaus turėtų būti paliktos reikšmės pagal nutylėjimą (1 1 1). Vienintelis reikšmių laukas, kuris turės reikšmes ne pagal nutylėjimą, yra *center*. Jis naudojmas nusakyti taško, apie kurį bus atliekami sąnario ir jo jungiamų kūno dalių (jei tokios egzistuoja) posūkiai, koordinates. Nusakant tokias modelio sąnarių reikšmes pagal nutylėjimą postūmiams, posūkiams ir mastelio keitimui, reikia grąžinti kūną į neutralią poziciją.

Kiekvieno sąnario laukas *center* turi atitikti tokią padėtį, kad sąnarys judėtų taip kaip realiame žmogaus kūne.

Humanoidą sudaro serija hierarchiškai susijusių sąnarių, kurių kiekvienas jungia kūno dalis. Pavyzdžiui:

```
...
DEF hanim_l_shoulder Joint {  name „l_shoulder“
   Center 0.167 1.36 -0,0518
      Children [
      DEF hanim_l_elbow Joint { name „l_elbow“
         center 0.196 1.07 -0.0518
         ...
      }
      DEF hanim_l_upperarm Segment {  name „l_upperarm“
      ...
```

Be išvardintų reikalavimų modeliai pirmiausiai turėtų atitikti jų skeletui keliamus sąnarių hierarchijos (pateikta priede) ir identifikavimo reikalavimus (remsimės aukščiau pateiktu pavyzdžiu):

- Laukas *name* naudojamas sąnariams identifikuoti.
- Kiekvieno sąnario DEF name atitinka *name* lauką sąnariui, bet su skiriamuoju prefiksu jo pradžioje. Prefiksas gali būti bet koks, bet vienodas visiems to humanoido sąnariams. Jis naudingas identifikuojant tuo atveju, kuomet keli humanoido modeliai aprašyti vienoje laikmenoje. Vienintelio modelio laikmenoje atveju, prefiksas turėtų būti “hanim_”. Pavyzdžiui, kairio peties atveju “DEF hanim_l_shoulder”.

Papildomi nestandartiniai sąnariai ir kūno dalys taip pat gali būti apibrėžti. Tėra trys reikalavimai, kuriuos jie turi atitikti:

- Visų pirma visi standartiniai sąnariai, kurie yra naudojami, privalo naudoti nurodytus vardus.
- Nė vienas naujas nestandartinis sąnarys negali įsiterpti į standartinių sąnarių "grandinę". Pavyzdžiui, ranka negali turėti papildomos alkūnės. Tačiau nauji priedai (pvz., plaukai, nagai, t.t.) gali būti prijungti prie humanoido kuriant naujus sąnarius, kurie taptų vaikiniais sąnariais standartiniams. Šie nestandartiniai sąnariai gali būti ir standartinių, ir nestandartinių sanarių vaikais.
- Nestandartiniai sąnariai privalo būti prijungti tokiu būdu, kad neatsirastų susikirtimų su standartinių sąnarių judinamomis kūno dalimis, net jei ir nėra jokios naujai atsiradusios kūno dalies animacijos.

Standartinių sąnarių animacijos neturėtų priklausyti nuo nestandartinių sąnarių animacijų (ar jų vaikų), kurių tėvais šios yra.

Papildomai sukurtiems sąnariams (neįeinantiems į standartą) turi būti duodamas "x_" prefiksas (pavyzdžiui *hanim_x_pigtails*), norint atskirti nuo standartinių, kurie galėtų turėti tokius pačius pavadinimus.

Nors H-Anim standartas numato galimybę apriboti sąnarių (Joint) posūkius įvedus laukų llimit (lower limit), limitOrientation ir ulimit (upper limit) reikšmes, tačiau dėl jų atitikmens nebuvimo OGRE SKELETON formate teoriškai iškyla pavojus dėl interpoliacijos tarp dviejų padėčių rezultato. Vis dėlto praktikoje toks atvejis būtų labai retas, nes interpoliuojamas kampas turėtų būti lygus arba didesnis už pusę apsisukimo (180 laipsnių). Toks apribojimas reikalingas tam, kad humanoido sąnarys neatsidurtų nerealioje tikro žmogaus atitinkamo sąnario padėtyje.

Į tokią nepageidautiną padėtį galėtų patekti peties sąnarys atliekant iteraciją tarp modelio su visiškai pakelta ranka į viršų ir su tos pačios rankos, atvestos atgal (dažniausiai bėgimo metu), padėtimi. Interpoliacija, atliekama trumpesniu keliu, duotų nepageidaujamą ir realybėje žmogui nebūdingą judesį, kai ranka leidžiama žemyn ne priekyje, o užnugaryje.

Tokios situacijos nebūtų būdingos kitiems, tarkim, kelio sąnariams. Sunku įsivaizduoti situaciją, t.y. dvi padėtis, tarp kurių kelis (arba alkūnė) lenktųsi į priešingą nei jam būdinga pusę.

Šio nepageidaujamo rezultato išvengsime, jeigu animacijų pradžioje ir pabaigoje humanoido kūno padėtys bus artimos viena kitai. Juk daugeliu atvejų skirtumas tarp pirmosios

iš jungiamų animacijų pabaigos padėties ir antrosios animacijos pradinės padėties bus nedidelis, t. y. žymiai mažesnis nei 180 laipsnių. O jeigu toks atvejis vis dėlto pasitaikytų, tarp šių dviejų animacijų galima įterpti jungiamąją animaciją, parašytą kuria nors modeliavimo programa, ir atlikti du interpoliavimus.

### 3.3. Animacijų jungimas

Abstrakčios procedūrinės animacijos vykdymo metu pavieniai skeleto judesiai yra sujungiami į sekas. Ar visais atvejais toks sujungimas bus įmanomas? Ar visada jis duos pageidaujamą rezultatą? Imkime paprasčiausią pavyzdį – ėjimą. Tai animacija, kurioje pakaitomis atliekamas "žingsnis kaire" ir "žingsnis dešine". PHA faile tokiu atveju būtų pakartotinai kviečiamos abi žemo lygio (OGRE SKELETON) animacijos, t.y. žingsniai. Teoriškai pirmoji animacija turėtų sekti iškart po antrosios ir prasidėti toje pačioje vietoje, kur antroji baigėsi. T.y. "žingsnis dešine" turėtų padėti dešinę koją tiksliai toje padėtyje, nuo kurios ji atsispirtų animacijoje "žingsnis kaire". Priešingu atveju gautume tarsi "sukarpytą" vaizdą. Kadangi abi animacijas, tikėtina, kurtų vienas autorius, jis į tai atsižvelgtų. Tačiau kai kalbame apie numatomas animacijų bibliotekas, kurias kartu kurtų galbūt visai nepažįstami žmonės, tokios prielaidos daryti nebegalime. Vienas sprendimas būtų nustatyti privalomą bet kokios animacijos pradžios ir pabaigos pozą. Žinoma, tai ne išeitis, nes tai labai apribotų animatorių galimybes.

Taigi darykime prielaidą, kad bus dažnai bandoma kurti animacijas iš fragmentų, kurie nedera vienas su kitu idealiai. Mes turime užtikrinti, kad tokiais atvejais perėjimai nuo vieno fragmento prie kito būtų kuo sklandesni, kuo mažiau pastebimi. Čia galima pasitelkti interpoliaciją. Galima sugeneruoti papildomus animacijos taškus (*keyframes*) tose vietose, kur būtų reikalingi tokie "perėjimai". Būtų galima leisti pačiam PHA kūrėjui pasirinkti, ar toks perėjimas reikalingas, ar ne, bei reguliuoti perėjimo trukmę. OGRE grafikos variklis moka manipuliuoti skeletu ir palaiko tokius perėjimus.

Vienas didžiausių uždavinių visame mūsų darbe yra gražus akiai skirtingų animacijų fragmentų sujungimas. Kuriant modelius ir juos animuojant ne visada lengva turėti identišką pradžios ir pabaigos padėtį. Kai kuriais atvejais vienodos pradžios ir pabaigos pozicijos tiesiog neįmanomos, todėl paleidžiant kelias animacijas nuosekliai, bus ryškiai matomi trūkčiojimai, gadinantys bendrą vaizdą. Norint to išvengti galimi du keliai: arba redaguoti jau turimas animacijas ir jas sutaikyti, arba generuoti kokią nors trečią – tarpinę – animaciją.

Pirmasis būdas labai neefektyvus, nes po kelių redagavimų originalios animacijos gali visiškai nebeatitikti pradinės paskirties, tad savo darbe taikysime antrąjį.

### *3.3.1. Interpoliavimas kvaternionais*

Humanoidų animacijos specifika yra tokia, kad didžioji dalis judesių aprašoma per tam tikrus kaulų posūkius apie sąnarius, taigi jungiamosios animacijos sukūrimas susiveda į interpoliavimą tarp tų posūkių, interpoliavimą tarp paskutinės kaulo orientacijos pirmojoje animacijoje ir pradinės orientacijos antrojoje. Susipažinome su keliais posūkių saugojimo ir interpoliavimo metodais, o praktiniams eksperimentams pasirinkome būdus, naudojančius kvaternionus. Svarbiausia tokio pasirinkimo priežastis ta, jog kvaternioninė interpoliacija gali duoti glodų rezultatą, ko negali padaryti kiti metodai. Be to, atliekamas greitas ir ne itin sudėtingas pavertimas į matricų pavidalą, su kuriomis galima atlikti kitas transformacijas. Taip pat ir mūsų naudojamas grafikos variklis OGRE turi patogų darbo su kvaternionais modulį, o visi posūkių veiksmai atliekami per kvaternionus. Literatūroje minimi tokių metodų trūkumai, tačiau mūsų darbe jie didelės įtakos neturės. Naudojant šį formatą negalima atlikti kelis kartus apsisukančio posūkio, be to, kyla įvairių interpoliavimo nesklandumų, jeigu interpoliuojamo posūkio kampas didesnis nei 180 laipsnių. Absoliuti mūsų programoje jungiamų posūkių dauguma turės žymiai mažesnį nei 180 laipsnių kampą, galbūt kelių ar keliolikos laipsnių. Kelis kartus apsisukančioms animacijoms tikrai nebus naudojamos interpoliacijos - teks naudoti specialiai tokiam atvejui sukurtas tarpines animacijas.

Praktinius bandymus atlikome įgyvendindami du kvaternioninius interpoliavimo būdus. [JBL] Pimasis, taip vadinamas LERP (Linear Quaternion Interpolation), yra tiesinė kvaternioninė interpoliacija, gana paprasta, bet nėra tokia jau bloga. Gaunamas gana gražus akiai posūkis. Nauja orientacija surandama iš pradžios bei pabaigos orientacijos kvaternionų ir interpoliacijos laiko pagal formulę:

$$\mathrm{LERP}(q1, q2, t) = q1 \cdot (1 - t) + q2 \cdot t$$

Literatūroje prie šio būdo trūkumų minimi nepageidaujami rezultatai, kai kampas tarp orientacijų yra arti 180 laipsnių. [MBA] Taip pat svyruoja interpoliavimo greitis: iš kraštų turėtų būti atliekama lėčiau, o per vidurį greičiau. Praktiniuose animacijų jungimo bandymuose didelių posūkio atlikimo greičio svyravimų nepastebėjome netgi gana dideliems mūsų animacijų specifikai kampams (per 90 laipsnių), tuo tarpu mažų kampų interpoliavimo svyravimai visiškai nepastebimi.

Šis būdas gali labai pagelbėti ateityje. Tokios tarpinės interpoliacinės animacijos kuriamos realiu vaizdavimo metu. Eksperimentuodami nuosekliai jungėme dvi animacijas – taigi vienu metu vykdoma tik viena tarpinė animacija. Realioje peržiūros programoje gali būti daug lygiagrečiai sujungtų animacijų grupių, kuriose kiekvienoje po kelis nuosekliai sujungtus judesius. Be to, juk interpoliuojama tarp kiekvieno humanoido kaulo orientacijų atskirai, tad tų tarpinių animacijų skaičius turėtų būti ne toks jau ir mažas. LERP būdas yra labai geras skaičiavimų greičio atžvilgiu, taigi tokiais atvejais labai tiktų.

Antrasis būdas, kurį nagrinėjome, vadinamas SLERP (Spherical Linear Quaternion Interpolation) – sferinė tiesinė kvaternionų interpoliacija. Šiuo metodu tiksliau skaičiuojamos interpoliavimo padėtys, bet jis, be abejo, yra brangesnis laiko atžvilgiu. Naudojama tokia formulė:

$$\cos(\alpha) = q1 \cdot q2$$

$$\mathrm{SLERP}(q1, q2, t) = \frac{q1 \cdot \sin((1-t)\cdot\alpha) + q2 \cdot \sin(t \cdot \alpha)}{\sin(\alpha)}$$

Šis būdas interpoliuojamas tolygiai ir neturi tokių greičio svyravimų, kaip anksčiau aprašytasis. Praktikoje matomas rezultatas nedaug skiriasi nuo to, kurį duoda LERP naudojimas. Skirtumas didesnis didesniems kampams, taip pat jis ryškiau matomas galūnių animacijose – kojų ir rankų pirštuose.

### *3.3.2. Interpoliacijos trukmė*

Tarpinėmis interpoliacinėmis animacijomis sujungti judesiai tampa nuoseklūs bei inertiški, tačiau tuomet iškyla kita problema – kokia turėtų būti tos tarpinės animacijos trukmė? Yra tokių atvejų, kai kelis animacijos fragmentus kuria vienas autorius ir tuomet vieno fragmento pradinė bei kito galutinė padėtys yra labai artimos arba net sutampa. Tuomet interpoliuojama tarp dviejų identiškų orientacijų. Gaunamas trumpo judesio sustabdymo įspūdis, kurio tikriausiai visai nereikia. Šiuo atveju interpoliacijos galbūt apskritai nereikia arba jos trukmė turi būti lygi 0. Tuo tarpu jeigu padėtys nutolusios viena nuo kitos gana daug, trumpos trukmės interpoliacija įvyks kaip blykstelėjimas ir taip pat neduos laukiamų rezultatų. Taigi tarpinės animacijos trukmė turi priklausyti nuo nueinamo kelio – interpoliuojamo posūkio kampo dydžio. Ta priklausomybė nėra taip paprastai nusakoma. Humanoido animacija sudaryta iš daugelio kaulų posūkių, o kiekvieno kaulo judėjimui aprašyti skiriamas atskiras takelis. Kiekvienas kaulas turi savo orientaciją, todėl vienų kaulų orientacijos tarp pirmojo

animacijos segmento ir antrojo gali skirtis labiau nei kitų. Pavyzdžiui, dviejose kojų animacijose skirsis kojų kaulų orientacijos, o rankos išliks nejudančios – jų orientacijos bus identiškos bet kokiame segmente, todėl ir posūkiai atliekami skirtingu kampu. Eksperimentiškai geriausius rezultatus davė skaičiuojamas visų kaulų posūkių kampų vidurkis. Kuo šis vidurkis didesnis, tuo didesnė turi būti tarpinės animacijos trukmė.

Ši trukmė taip pat priklauso ir nuo jungiamų animacijų trukmės, tiksliau sakant, nuo jų greičio, nes svarbu ir tai, kokio laipsnio posūkiai atliekami. Greitis gaunamas nueitą kelią (šiuo atveju intepoliuojamų posūkių kampų vidurkį) dalinant iš trukmės. Greitį tarpinei animacijai apskaičiuoti nesunku, nes tokioje animacijoje interpoliuojama tik tarp dviejų padėčių, todėl lengva paskaičiuoti kampų vidurkį. Jungiamose animacijose padėčių gali būti daug, todėl nueitu keliu negalime laikyti tiesiog skirtumą tarp pradinės ir galinės padėties. Išeitis būtų suskaičiuoti greitį kurioje nors mažoje animacijos atkarpoje. Taigi galime apskaičiuoti greitį mažoje atkarpoje pirmojo jungiamojo segmento pabaigoje ir antrojo pradžioje. Tarpinės animacijos greitis būtų gaunamas, pavyzdžiui, iš šių greičių vidurkio arba iš vidutinio greičio, o pagal greitį galima apskaičiuoti trukmę. Eksperimentiniai rezultatai rodo, kad gražesnės interpoliacijos gaunamos trukmei skaičiuoti naudojant jungiamų segmentų vidutinį greitį. Taip skaičiuojant trukmę gana geri rezultatai gaunami ir sujungiant dvi labai skirtingos trukmės animacijas (tarkim viena 10 kartų ilgesnė už kitą). Savo animacijos redagavimo programoje esame numatę galimybę vartotojui pačiam nurodyti tarpinių animacijų trukmę, tačiau pasiūlysime ir porą apskaičiuojamų būdų.

Dar viena tarpinių animacijų nauda būtų tokiu atveju, jei galėtume kurti greitėjančios ar lėtėjančios animacijos įspūdį (pavyzdžiui, realistiškas perėjimas tarp ėjimo animacijos ir bėgimo). Iki šiol visas animacijas tiek tarpines, tiek netarpines vykdėme laikui kintant tiesiškai, t. y. kiekviename kadre laikas padidėja vienodą skaičių laiko vienetų. Greitėjimo ar lėtėjimo įspūdį sukeltų netiesinis interpoliacijos laiko keitimas. Šia kryptimi eksperimetus planuojame atlikti artimiausiu metu.

### *3.3.3. Interpoliacija ir kaulų struktūra*

Dar viena aktuali interpoliacijos problema, kuri mūsų darbe ypač ryški – tai tarpinės animacijos tarp skirtingiems modeliams skirtų animacijų. Kaip jau minėjau, animacijoje kiekvienam kaului skiriamas atskiras takelis, o praktiškai interpoliuojama tarp šių takelių porų. Jeigu modelis padarytas nesilaikant tam tikrų standartų, tuomet labai sunku jį sieti su kitu modelu, o taip pat jo animacijas leisti per kitą modelį. Praktiškai neįmanoma korektiškai

suporuoti kaulų takelių ir interpoliacija tikrai neduos laukiamų rezultatų. Šią problemą lengvai išsprendžia modelių ir animacijų pririšimas prie H-Anim formato, kuris aprašo standartinius kaulų pavadinimus ir struktūrą. Turint vienodus pavadinimus lengva suporuoti kaulų takelius, tuomet aišku, tarp ko vykdyti interpoliaciją. Tokiu atveju nėra problemos, jeigu apjungiamos animacijos judina nevienodą skaičių kaulų – interpoliaciją užtenka vykdyti tarp tų kaulų, kurie turi porą kitame segmente, o kitur nevykdyti. Detaliau reikalavimus modeliams aprašėme prieš tai buvusiame skyriuje.

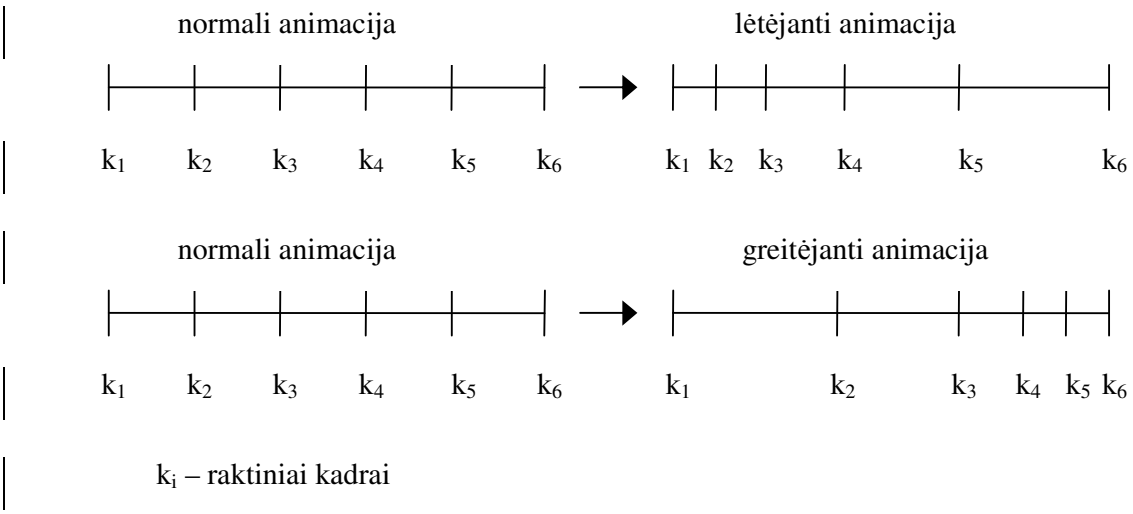
## 3.4. Tarpinės animacijos kūrimas

Kaip jau minėta anksčiau, tarpinė animacija sukuriama žinant jos trukmę. Vartotojas pageidaujamą trukmę gali nurodyti PHA faile atributu *transition*. Jei *transition* nenurodomas, jo reikšmė paliekama apskaičiuoti programai. Norint paleisti animacijas be tarpinės animacijos nurodoma transition reikšmė 0. Automatiniu būdu tarpinės trukmė apskaičiuojama remiantis vidutiniais kaulų judėjimo greičiais trijuose pirmosios animacijos pabaigos segmentuose bei trijuose antrosios pradžios segmentuose. Tarpinėje animacijoje sukuriami du raktiniai kadrai, nutolę vienas nuo kito per apskaičiuotą ar nurodytą laiko trukmę. Paprasčiau tariant į pirmąjį raktinį kadrą nukopijuojamas paskutinis pirmosios animacijos kadras, o į antrąją – pirmasis antrosios kadras, tačiau įrašomi tik tie takeliai (jie atitinkamai nurodo kaulų pozicijas ir orientacijas), iš kurių susidaro pora. Takeliai rišami prie konkretaus kaulo, pagal unikalų kaulo pavadinimą, tad jeigu abejose animacijose kaulas judinamas, jis turės takelius tokiu pačiu pavadinimu. Likę kadrai gaunami interpoliuojant animacijos vykdymo metu. Šioje vietoje pasidaro svarbu, kad raktiniuose kadruose kaulų orientacijos skirtųsi mažiau nei 180 laipsnių.

### *3.4.1. Galimi tarpinės animacijos patobulinimai*

Tarpinė animacija kuriama remiantis skirtumų tarp kaulų orientacijų vidurkiu. Taip pat vidutinis greitis jungiamuose segmentuose skaičiuojamas remiantis kaulų nueinamų lankų vidurkiu. Tokiu būdu išeina, jog greičiausiai visų kaulų greitis kis netolygiai pereinant nuo jungiamos animacijos prie tarpinės ir atvirkščiai. Galbūt geresnis variantas vaizdavimo atžvilgiu būtų nustatyti tarpinę trukmę kiekvienai takelių porai atskirai, tuomet visos tarpinės animacijos trukmė būtų ilgiausia iš nustatytų trukmių, o likusios derinamos su ilgiausia. Tokiu atveju nepakaktų ir kuriamų dviejų raktinių kadrų: kiekvienam takeliui reikėtų dviejų

animacijios pradžioje bei pabaigoje, bei dviejų papildomų, jų pozicijas laike nustatant pagal minėtas tarpines takelių trukmes. Greičio kitimo tolygumui pagerinti reikėtų tas pozicijas laike išdėstyti netiesiškai (**schema 3**), pavyzdžiui, norint, kad animacija tolygiai lėtėtų, arčiausiai pradžios esančius raktinius kadrus patraukti dar arčiau pradžios, norint, kad animacija greitėtų, raktinius kadrus stumti link pabaigos.




**Schema 3. Raktinių kadrų išdėstymas laike**

### 3.5. Taktinės animacijos

OGRE variklyje animacijų trukmė nurodoma sekundėmis, kai animacija jau padaryta ir eksportuota iš kokios nors modeliavimo programos, šios trukmės nebegalima keisti. Tačiau natūralus noras leisti vartotojui nurodyti, kokį laiko tarpą trunka viena ar kita animacija. Iš to kilo taktinių animacijų idėja, kai animacijos vykdomos tam tikrą skaičių taktų viena po kitos. Tokiu atveju intuityviai du segmentai turėtų būti vaizduojami antras iš karto po pirmo, nepaliekant laiko vykdyti tarpinei animacijai. Intuityviai segmentų taktų suma turėtų būti lygi pilnos animacijos trukmei taktais, o atskirai vaizduojant ir tarpinius segmentus bendra trukmė be abejo padidėtų, ypač kai jungiamų segmentų pradinės ir galinės padėtys nutolusios daugiau. Taigi belieka tarpiniam segmentui neskirti individualaus vykdymo laiko bei įkomponuoti jį į jungiamų segmentų taktus.

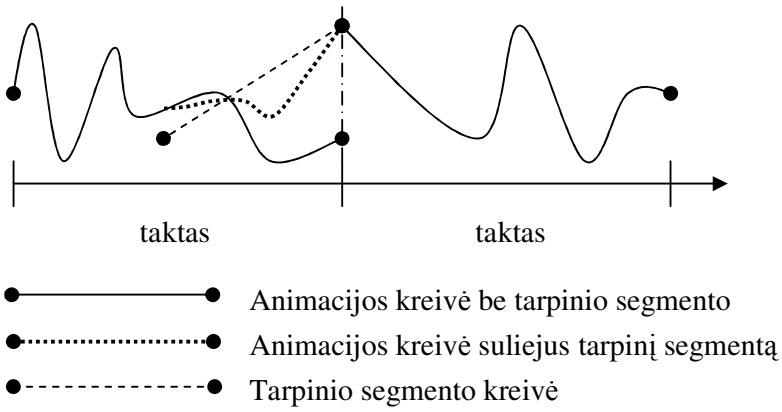
### *3.5.1. Dviejų animacijų sujungimas paskirstant svorius*

Vienas įkomponavimo į jungiamų segmentų taktus būdų yra vykdyti tarpinį segmentą kartu su jungiamais segmentais. Tokiu atveju vienu kažkuriuo metu būtų vaizduojamos dvi animacijos: vienas iš jungiamų segmentų ir tarpinis segmentas, šie segmentai suliejami. Suliejimo metu abu segmentai daugiau ar mažiau svarbūs – ši svarba reguliuojama per segmentų suliejimo svorius. Be to, juk tarpinio segmento pradžia yra ne kas kita kaip pirmojo segmeto pabaiga, o tarpinio pabaiga lygi antrojo pradžiai. Kitaip sakant, suliedami pirmo ir tarpinio segmentų pabaigas su atitinkamais svoriais 0 ir 1 gausime antojo segmento pradžią. Taigi iš esmės sprendžiami du klausimai: kaip paskirstyti suliejimo svorius bei kada pradėti ir baigti tokį paskirstymą.

### *3.5.2. Skirtinga tarpinės animacijos pradžia*

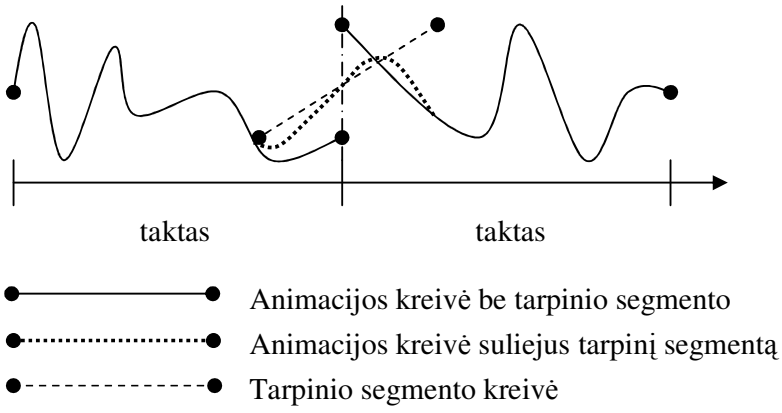
Suliejimo svorių paskirstymas priklauso nuo apskaičiuotos arba nurodytos tarpinės animacijos trukmės. Visi segmentai – tiek jungiami, tiek tarpiniai – turi būti vykdomi nuo pradžios iki galo, tad svorių paskirstymą reikia atlikti visą laiką, kol bus vykdomas tarpinis segmentas. Pradedant tarpinį segmentą, jo svoris lygus 0, o jungiamo lygus 1. Įgyvendinome du skirtingus atvejus:

- Tarpinis segmentas vykdomas pirmojo jungiamojo segmento vaizdavimo metu ir baigiasi kartu su jungiamuoju segmentu (**schema 4**). Svorių paskirstymo pradžia laike nustatoma iš pirmojo segmento trukmės atimant tarpinio segmento trukmę. Pradžioje tarpinio segmento svoris 0, o pabaigoje 1. Atitinkamai pirmo jungiamojo segmento svoris kinta nuo 1 iki 0 pabaigoje. Tokiu atveju jungiamojo segmento pabaiga nebus atliekama taip, kaip sukurta animacijoje, ji bus pakeičiama taip, kad tolygiai įsilietų į antrąjį segmentą. Be abejonės bazinės, jungimosios animacijos pakeitimas nėra pageidaujamas reiškinys, tačiau šis būdas išsaugo nepakeistą segmento pradžią. Tokio būdo silpnoji pusė – gali būti ryškiai matomas kaulų sukimosi greičio pokytis prasidedant antrajam segmentui. Be to, atsiranda apribojimai tarpinio segmento trukmei – ji negali būti didesnė nei pirmo jungiamojo trukmė.




**Schema 4. Tarpinio segmento suliejimas pabaigoje**

- Antruoju atveju tarpinis segmentas vykdomas kartu su pirmo jungiamojo pradžia bei antro jungiamojo pabaiga (**schema 5**). Svorių paskirstymo pradžia laike nustatoma iš pirmojo segmento trukmės atimant pusę tarpinio segmento trukmės, o pabaiga prie antro jungiamojo segmeto pradžios pridedant pusę tarpinio trukmės. Pradžioje tarpinio segmento svoris lygus 0, viduryje 1, o pabaigoje vėl 0. Tada pirmojo segmento svoris suliejimo pradžioje lygus 1, jam baigiantis 0, o antrojo segmento – 0 prasidedant ir 1 baigiantis tarpiniam segmentui. Šiuo atveju pakeičiama ne tik jungiamojo segmento pabaiga, bet ir pradžia. Praktiškai gali būti taip, jog didžioji dalis animacijos bus vykdoma ne taip kaip nurodyta originalioj animacijoj. Jeigu akcentai sudėlioti pradžioje ar pabaigoje, jie bus susilpninti ar iš vis išnaikinti. Be to, apribojimas tarpinio segmento trukmei – ji negali būti ilgesnė, už trumpesniąją iš jungiamų segmentų trukmę. Šio varianto pranašumas tas, kad gaunamas tolygesnis animacijų sujungimas, dauguma atvejų ištaiso greičių netolygumus.



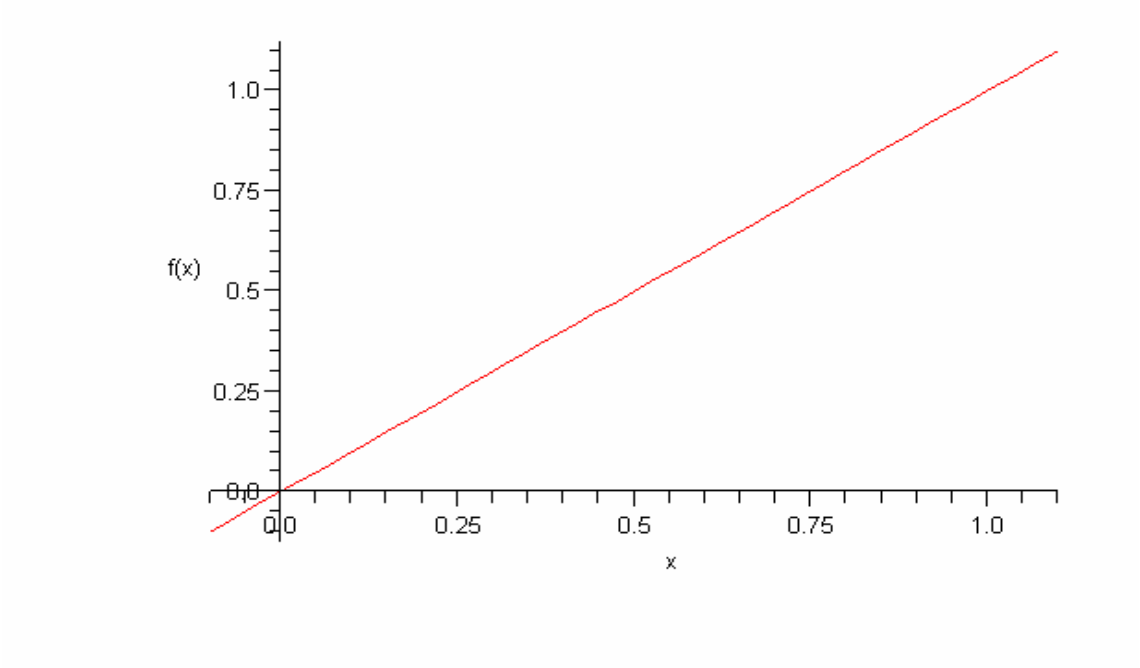

**Schema 5. Tarpinio segmento suliejimas pabaigoje ir pradžioje**

### *3.5.3. Suliejimo svorių paskirstymo funkcijos*

Greičių netolygumams ištaisyti taip pat bandėme naudoti kelias svorių paskirstymo funkcijas. Tokios funkcijos apskaičiuoja tarpinio segmento svorį tam tikru jo vaizdavimo laiko momentu. Pirmuose variantuose reikšmės kinta nuo 0 iki 1, likusiuose nuo 0 iki 1 ir vėl 0. Argumentas x tolygiai kinta nuo 0 iki 1 ir nurodo, kokia tarpinio segmento dalis jau įvykdyta.

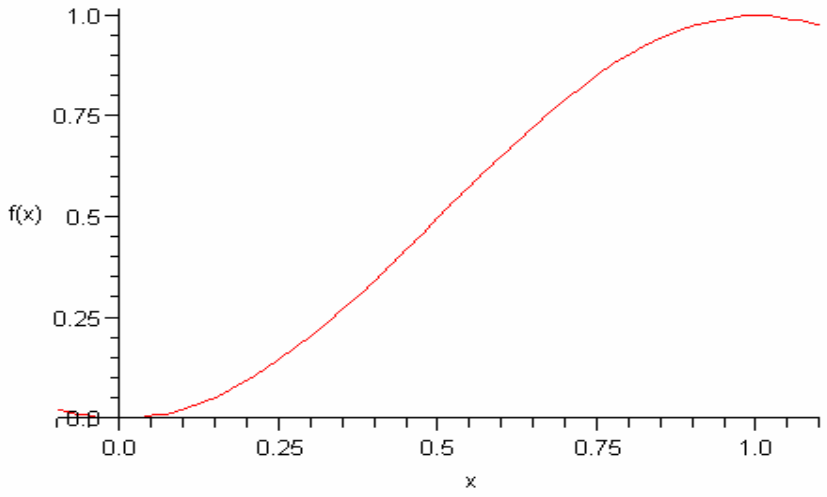
- Paprasčiausias tiesinis paskirstymas. Užima mažai skaičiavimo laiko.

$$f(x) = x, x = 0...1;$$




- Paskirstymas pasinaudojant cos() funkcija. Duoda didesnį greičių tolygumą, bet brangesnis skaičiavimų atžvilgiu.

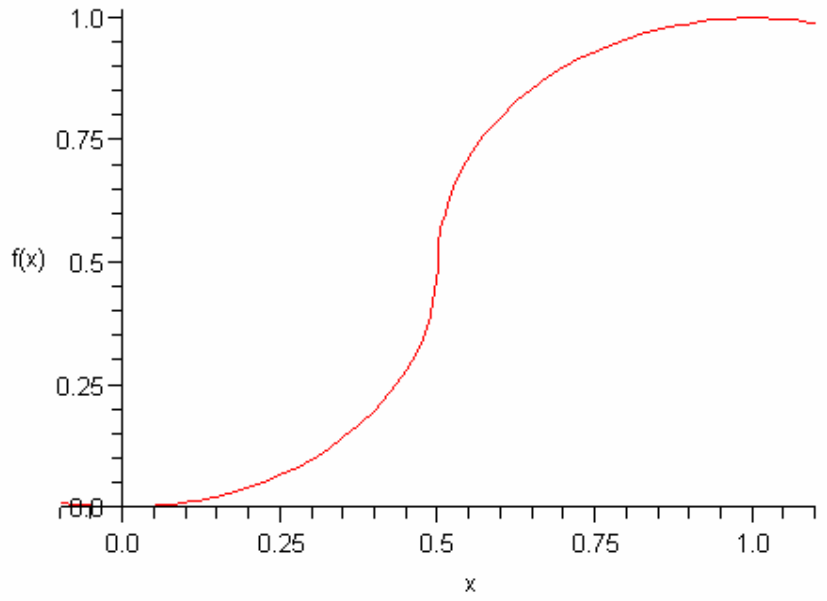
$$f(x) = \frac{1}{2} - \frac{1}{2} \cdot \cos(\pi \cdot x), x = 0...1;$$




- Apskritiminis paskirstymas. Duoda gerą tolygumą kraštuose ir viduryje.

$$f(x)=\frac{1}{2}+\frac{|x-0.5|}{x-0.5}\cdot\sqrt{\frac{1}{4}-\left(x-\frac{1}{2}-\frac{1}{2}\cdot\frac{|x-0.5|}{x-0.5}\right)^2},x=0\ldots\frac{1}{2},\frac{1}{2}\ldots1;$$

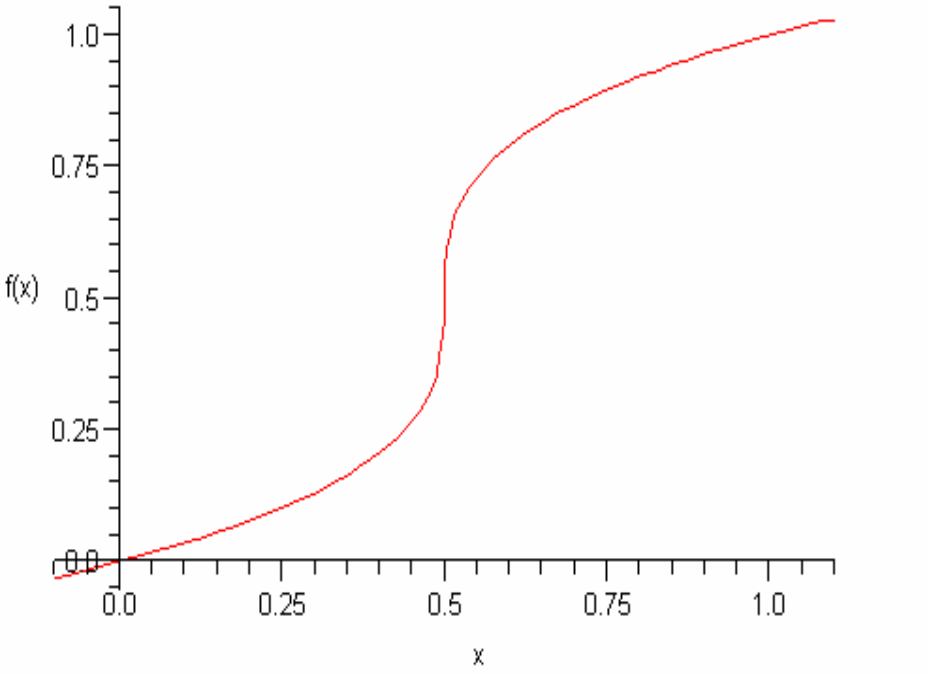
$$f\left(\frac{1}{2}\right)=\frac{1}{2};$$



- Paskirstymas su kubine parabole. Duoda gerą paskirstymą kraštuose, bet prastesnį viduryje.

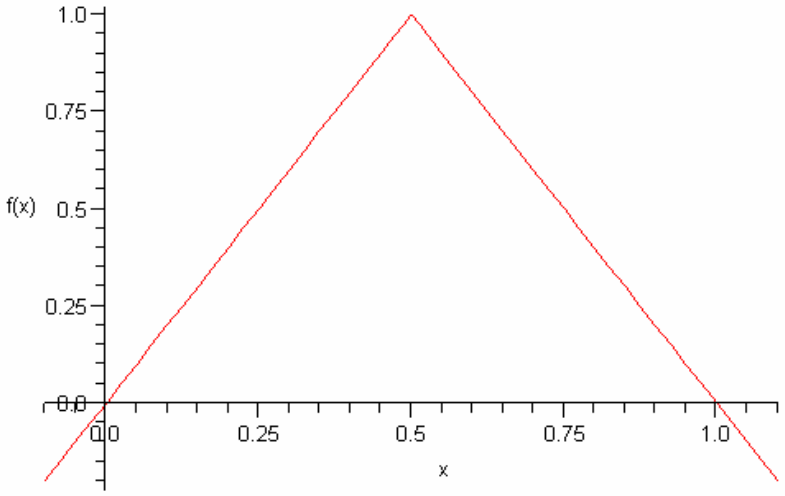
$$f(x) = \frac{1}{2} + \frac{|x - 0.5|}{x - 0.5} \cdot \sqrt[3]{\frac{x}{4} - \frac{1}{8}}, x = 0 \ldots \frac{1}{2}, \frac{1}{2} \ldots 1;$$

$$f\left(\frac{1}{2}\right) = \frac{1}{2};$$



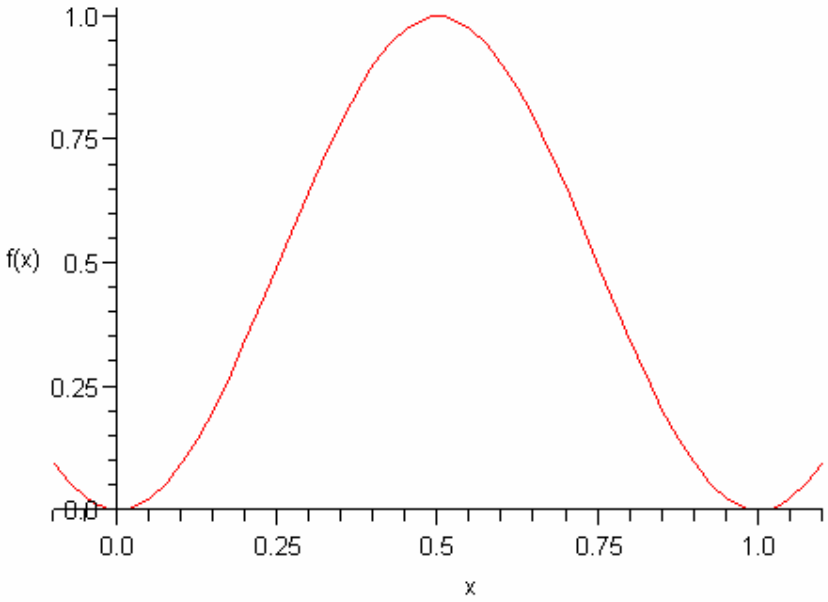
- Tiesinis paskirstymas, kai svoris kinta 0 -> 1 -> 0.

$$f(x) = 1 - |2 \cdot x - 1|, x = 0...1;$$



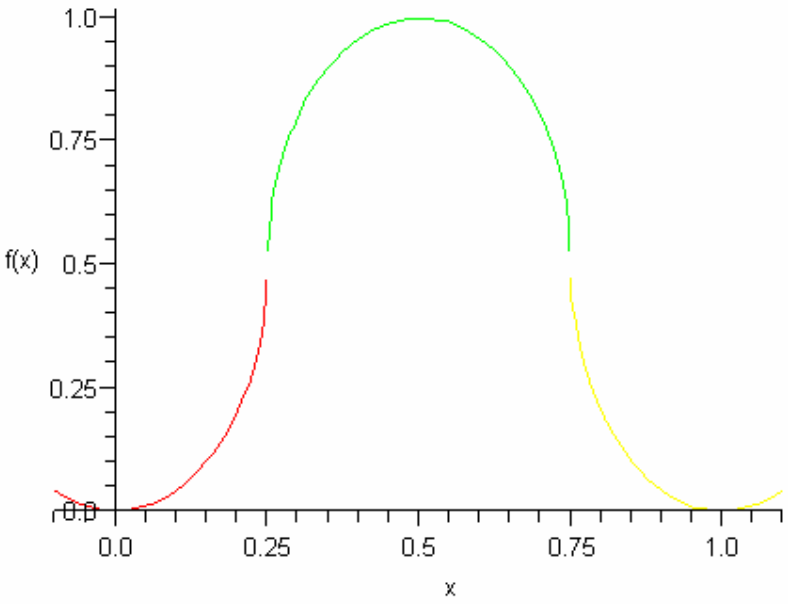

- Paskirstymas su cos() funkcija, kai svoris kinta 0 -> 1 -> 0.

$$f(x) = \frac{1}{2} - \frac{1}{2} \cdot \cos(2 \cdot \pi \cdot x), x = 0...1;$$



- Apskritiminis paskirstymas, kai svoris kinta 0 -> 1 -> 0.

$$f(x)=\begin{cases}\frac{1}{2}-\sqrt{\frac{1}{4}-(2\cdot x)^2}, x=0...\frac{1}{4};\\ \frac{1}{2}+\sqrt{\frac{1}{4}-(2\cdot x-1)^2}, x=\frac{1}{4}...\frac{3}{4};\\ \frac{1}{2}-\sqrt{\frac{1}{4}-(2\cdot x-2)^2}, x=\frac{3}{4}...1;\end{cases}$$




### *3.5.4. Animacijų suliejimo problemos*

Vykdant dvi animacijas vienu metu pagal atitinkamus svorius derinamos dvi kiekvieno kaulo orientacijos, paimtos iš animacijos takelių raktinių kadrų. Informacija esanti bazinėse animacijose ir pagamintuose tarpiniuose segmentuose gali atrodyti tvarkinga ir tokie segmentai atskirai vykdomi problemų nekelia, visgi juos vykdant kai kurie kaulai pradeda judėti netolygiai. Vienas tokių netolygumų atsiradimo atvejis būna tada, kai bazinėje animacijoje kurio nors kaulo orientaciją žymintis kvaternionas atitinka didesnio nei 180 laipsnių kampo posūkį apie vektorių. Sukurta bazinė animacija išsaugoma XML failuose, kuriuose *rotate* mazgai (reiškia kaulų orientaciją) nurodomi „ašies - kampo“ formatu: posūkio

kampas radianais ir sukimosi ašis – trimatis vektorius. Programoje užkraunant animacijų failus šis formatas keičiamas į kvaternioninį. Taigi visos orientacijos užrašomos kvaternionais, tačiau dėl tam tikrų priežasčių gaunami visų priešingų ženklų kvaternionai negu turėtų būti gaunami, nors jie ir reiškia vieną ir tą pačią kryptį bei vykdant animaciją atskirai tai reikšmės neturi. Kitaip sakant kvaternionas $q_1 = (a, b, c, d)$ žymi tą pačią kryptį kaip ir kvaternionas $q_2 = (-a, -b, -c, -d)$. Tačiau vienas iš jų atitinka kampą, didesnį nei 180 laipsnių. Tarkim $q_1$ atitinka posūkį laipsniu $\alpha$ apie vektorių $v_1 = (l, m , n)$, tuomet $q_2$ atitiks posūkį $180 – \alpha$ apie vektorių $v_2 = (-l, -m, -n)$. Orientacija po šių posūkių būtų vienoda, tik atlikta skirtingais keliais. Tarkim $\alpha < 180$ laipsnių, tuomet turėtų būti gaunami kvaternionai, tokie kaip $q_1$, bet retkarčiais gaunami panašūs į $q_2$ kvaternionai, kurie savaime nėra klaida, nors duoda nepageidaujamas pasekmes. Paprasčiausias būdas išvengti tokių pasekmių yra perrinkti kiekvieno kaulo visų raktinių kadrų orientacijas ir jas „pataisyti“ – kur reikia pakeisti visus kvaterniono ženklus.

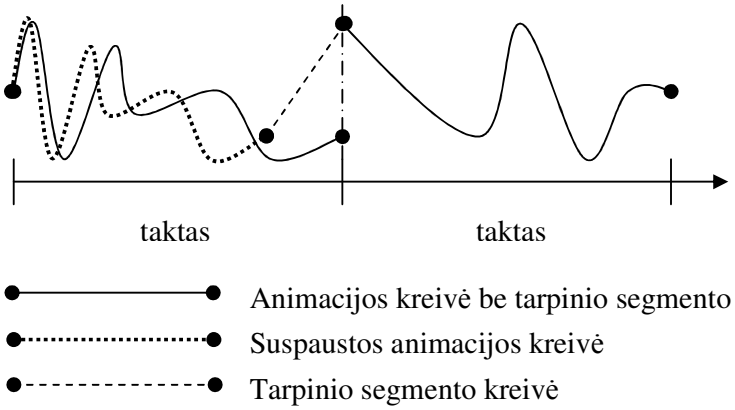
Antras atvejis, kai atskirai animacijos vykdomos gerai, o atliekant kartu atsiranda nepageidaujami efektai, yra sunkiau ištaisomas. Šiuo atveju orientacijos žyminčios didesnius nei 180 laipsnių kampus atsiranda ne raktiniuose kadruose, o interpoliacijose tarp jų, pavyzdžiui, tarpinio segmento vykdymo metu. Tai yra tas atvejis, kai antro jungiamojo segmento pradžia pakankamai smarkiai nutolusi nuo pirmo pabaigos. Atliekant tarpinį segmentą atskirai, problemų nekyla, nes interpoliacija atliekama trumpiausiu keliu, bet bandant sulieti su jungiamaisiais segmentais gaunami nepageidaujami rezultatai. Šios problemos iki galo išspręsti nepavyko, tačiau galimas sprendimas būtų generuoti tarpinį segmentą daugiau nei iš dviejų raktinių kadrų bei tikrinti tuose raktiniuose kadruose esančius kvaternionus.

### *3.5.5. Animacijų jungimo metodas be segmentų suliejimo*

Taigi animacijų jungimas taikant bazinių segmentų ir tarpinio suliejimą turi tam tikrų trūkumų, tačiau ir skirti papildomo laiko tarpinio segmento individualiam atlikimui neišeina turint galvoje taktines animacijas. Galimas dar vienas sujungimo būdas, kurio mes neįgyvendinome. Tokiu atveju tektų vykdyti tarpinį segmetą individualiai, bet tuo laiku, kada turėtų būti atliekami jungiamieji segmentai, arba kuris nors vienas jungiamasis segmentas (**schema 6**). Taigi jungiamuosius segmentus reikėtų įvykdyti šiek tiek greičiau tuo būdu paliekant vietos tarpinio segmento vykdymui. Tuomet tarpinis segmentas atliekamas nuosekliai po pirmojo, po jo nuosekliai atliekamas antrasis. Būdai paskirstyti taktus gali būti

keli, pavyzdžiui, tarpinį segmentą tarsi sukabinti su pirmuoju ir vykdymui skirti pirmojo segmeto takto dalį, proporcingą tarpinės animacijos trukmei. Kitas pavyzdys, tarpinei animacijais skirti pirmojo bei antrojo segmeto taktų dalis, proporcingas jungiamųjų animacijų trukmei. Toks metodas turėtų nemažai privalumų. Visų pirma, animacijos būtų atliekamos taip, kaip yra sukurtos, nemodifikuojant jų pradžios ar pabaigos. Antra, pavyktų pašalinti dauguma problemų susijusių su animacijų suliejimu taikant svorius. Be to, neliktų apribojimų tarpinės animacijos trukmei.

Deja, tokiu atveju neliktų keleto taktinių animacijų privalumo. Jungiamųjų segmentų trukmės nebeatitikų taktų trukmių, bent jau segmeto pradžia arba pabaiga nesutaptų su takto pradžia ar pabaiga. Jeigu tarpinės animacijos trukmė palyginus gana maža, toks metodas tiktų gana neblogai, tačiau tuomet jungiamųjų animacijų pradžios ir pabaigos pozicijos turi skirtis minimaliai. Bendrai paėmus sugeneruota tarpinės animacijos trukmė gali būti pakankamai ilga, pavyzdžiui kelis kartus ilgesnė už jungiamosios animacijos trukmę. Tuomet tarpinis segmentas nustelbs kurį nors vieną jungiamąjį, o šio tikroji pradžia arba pabaiga ryškiai skirsis nuo takto pradžios arba pabaigos, animacijos akcentai labai iškryps iš tos vietos, kurioje jie turėtų būti rodomi.




**Schema 6. Tarpinio segmento įterpimas**

### 3.6. Reikalavimai ir rekomendacijos animacijų kūrimui

Kad kokie nors iš minėtų animacijų jungimo metodų veiktų vaizdavimo programoje pačios animacijos turi būti kuriamos prisilaikant tam tikrų standartų. Humanoidų animacija paremta tam tikrų tarpusavyje hierarchiškai susijusių mazgų judėjimu. Šie mazgai erdvėje atitinka virtualius humanoido sąnarius, apie kuriuos sukami kaulai. Vaizdavimo programoje ir OGRE variklyje šiuos mazgus atitinka *Bone* objektai, t .y. kaulai, per juos vyksta ne tik humanoido, o

ir bet kokio kito objekto animacija. Šie kaulai žymi tiktai vietą, apie kurią reikia sukti (praktiškai panašu į sąnarį) ir turi savo orientaciją – kryptį. Animuojamo objekto tinklelis (*mesh*) pririšamas prie kaulų (arba sąnarių) objektų, o keičiant pastarųjų pozicijas ir kryptis, juda visas objektas. Kuriant animaciją kiekvieno kaulo postūmių ir poslinkių duomenys surašomi į raktinius kadrus. Norint sklandžiai vykdyti nuoseklias animacijas vaizdavimo programoje visų pirma šiuose raktiniuose kadruose turi būti teisinga informacija.

Raktiniuose kadruose informacija dažniausiai gali būti įrašyta trijuose mazguose: *translate*, *rotate* ir *scale*. Šiuolaikiniai įrankiai, tokie kaip 3DStudioMax ar panašios programos, leidžia redaguoti visus mazgus, tačiau nevisada tai į naudą kuriant humanoidų animaciją. Humanoidų animacijos specifika tokia, kad didžioji dalis judesių aprašoma kaulų posūkiais apie sąnarius, taigi svarbiausias yra *rotate* mazgas. Tuo tarpu *translate* mazge esanti informacija daugeliu atveju yra netgi nepageidaujama. Tik šakninis kaulas – visos kaulų hierarchijos tėvas – gali turėti *translate* mazgą, kuriame duomenys nenuliniai. Tai svarbu dėl tų pačių animacijų panaudojimo ne vienam, o keliems modeliams. Tik posūkių duomenis naudojančios animacijos be didesnių problemų gali būti panaudotos įvairių proporcijų ir dydžių modeliams animuoti. Atliekant kaulines animacijas per postūmius, skitingų proporcijų modeliams postūmiai išliks vienodi, todėl vieni modeliai išsitampys, kiti susispaus. Mūsų vaizdavimo programoje vienas pagrindinių dalykų yra animacijų perpanaudojimas, tad labai svarbu, jog jos būtų korektiškai atliekamos įvairiems modeliams, o tai įmanoma tik tada, kai atliekami tik kaulų posūkiai.

Kaip jau minėta, kuriant animaciją posūkių duomenys surašomi į raktinius kadrus. Konkrečiau į raktinius kadrus įrašomas pokytis laipsniais tarp pačios pirmosios kaulo krypties ir esamos krypties. Taigi gaminant animaciją labai svarbus pats pirmasis raktinis kadras, arba nulinis kadras. Norint modeliui panaudoti kitą animaciją arba kito modelio animaciją kaulų krypčių pokyčiai turi būti skaičiuojami nuo vienos ir tos pačios pozicijos. Taigi modeliuojant humanoido judesius visų modelių visų animacijų nuliniame raktiniame kadre turi būti fiksuota standartinė pozicija, apie kurią plačiau rašėme skyrelyje apie reikalavimus modeliams. Tačiau tai nereiškia, kad visos animacijos privalo turėti standartinę pradžią. Tas svarbu tik modeliuojant judesius kokioje nors animacijų kūrimo programoje. Standartinė padėtis užfiksuota nuliniame raktiniame kadre, kituose kadruose užfiksuoti pokyčiai nuo standartinės padėties. Eksportuojant animaciją į OGRE formatą, kurį naudojame saugojimui ir vaizdavimui, galima neįtraukti nulinio kadro, tuomet reali animacijos pradžia bus pirmasis raktinis kadras iš modeliuotos animacijos, kuris praktiškai gali būti bet koks. Tokiu būdu

kurtos animacijos neturės pradžios apribojimų, bet išsaugos tinkamus duomenis visuose raktiniuose kadruose.

# 4. Procedūrinių animacijų formatas

Mes sukūrėme procedūrinių humanoidų animacijų (PHA) formatą, kuris naudoja XML sintaksę. Remdamasis kai kuriais procedūrinio programavimo principais PHA suteikia galimybę kurti labai aukšto lygio animacijas. (Žr. pavyzdį priede B.)

## 4.1. PHA failas

PHA failas yra XML failas su plėtiniu *.pha*. Kiekvienas PHA failas laikomas atskiru moduliu, kadangi jame apibrėžtos procedūros yra pasiekiamos tik jam pačiam ir tik tiems moduliams, kurie į jį kreipiasi. Laikantis procedūrinio programavimo principų, viename modulyje turėtų būti laikomos tik tarpusavyje susijusios animacijos, pavyzdžiui, tam tikra kūno judesių grupė arba kelios to paties judesio variacijos.

Kaip ir bet kuris kitas XML failas, modulis prasideda tokia antrašte:

```
<?xml version="1.0"?>
```

Šakniniame elemente *pha* atributas *tickLength* nurodo vieno takto ilgį sekundėmis:

```
<pha tickLength="0.8">
```

Kiekvienas modulis privalo deklaruoti savo takto ilgį. Takto ilgis galioja tik tame modulyje, kuriame yra aprašytas, ir tik tiesiogiai kviečiamose to modulio procedūrose. T.y. jei viena modulio procedūra kviečia kitą, pastarosios iškvietimo metu jos takto ilgis gali keistis (žr. 4.4. Procedūros).

Šakninio elemento vaikai gali būti tik trijų tipų XML elementai: *load*, *include* ir *procedure*.

## 4.2. Žemo lygio animacijų užkrovimas

Elementas *load* naudojamas žemo lygio animacijų užkrovimui. Čia žemo lygio animacijos suprantamos kaip tos, kurios tiesiogiai judina humanoido kaulus (t.y. ne procedūros). Reikia pažymėti, kad pats PHA formatas nėra pririštas nei prie OGRE SKELETON, nei prie X3D. Savo darbe testavimo tikslais žemo lygio animacijoms saugoti naudojome OGRE SKELETON. Tačiau jei ateityje kas nors sugalvotų jį pakeisti X3D ar kitu formatu, PHA notacija nuo to nesikeistų.

```
<load src="katalogas.pakatalogis.Failas"/>
```

Atribute *src* nurodomas modulio pilnas vardas, panašiai kaip Java programose nurodomas pilnas klasės vardas - taškais atskiriant vardus visus paketų (katalogų), kuriuose yra pageidaujamas failas. Pateiktame pavyzdyje failo kelias kompiuteryje būtų toks:

```
%ROOT%/katalogas/pakatalogis/Failas.skeleton
```

Čia *%ROOT%* - šakninis procedūrinių animacijų katalogas, kurio padėtis diske nurodoma peržiūros programos konfigūracijoje. Jame laikomos ir žemo lygio animacijos. Reikia pastebėti, kad atributo *src* reikšmėje nenurodomas failo plėtinys: mūsų peržiūros programa automatiškai ieško *.skeleton* failų.

### 4.3. Kitų PHA modulių naudojimas

Moduliarumas realizuojamas elementu *include*:

```
<include src="dances.swing.Balboa"/>
```

Šis elementas deklaruoja, kad PHA modulis kviečia procedūras, esančias kitame PHA modulyje. Atribute *src* nurodomas naudojamo modulio pilnas vardas, lygiai kaip užkraunant žemo lygio animacijas. Kiekvieno modulio visos procedūros yra "viešos", t.y. gali būti kviečiamos iš kitų jas naudojančių modulių.

### 4.4. Procedūros

Modulis gali turėti vieną arba daugiau procedūrų. Kiekviena procedūra turi vardą (atribute *name*), kuris privalo būti unikalus tame modulyje, ir kviečia vieną arba daugiau žemo lygio animacijų arba kitų procedūrų. Aprašas atrodo taip:

```
<procedure name="walk">
    <call name="left_step" duration="1"/>
    <call name="right_step" duration="1"/>
</procedure>
```

Bet kuris modulis gali turėti vieną specialią procedūrą vardu "main". Ši yra kviečiama automatiškai, vos tik modulis užkraunamas peržiūros programoje.

### 4.5. Animacijų kvietimas

Elementu *call* aprašomas žemo lygio animacijos arba procedūros iškvietimas. Nors šie du animacijų tipai yra labai skirtingi, kvietimo metu jos yra visiškai lygiavertės - taip turi atrodyti

PHA programuotojui, kuris tiesiog užrašo kviečiamos animacijos pavadinimą, o vykdymo detalės paslepiamos po peržiūros programos realizacija.

```
<call name="walk" duration="2" transition="0.2" repeat="3"/>
```

Šiame pavyzdyje kviečiama animacija "walk". Kvietimo metu nurodoma, kad animacijos trukmė (*duration*) turi būti 2 taktai, tarpinės animacijos trukmė (*transition*) - 0.2 takto, ir kad animacija turi būti vykdoma 3 kartus iš eilės. Atributai *name* ir *duration* yra privalomi, kiti du - ne. Laikomasi principo: kviečiančioji procedūra visada kontroliuoja kviečiamosios trukmę. Reali kviečiamosios animacijos trukmė gali būti ilgesnė arba trumpesnė, tačiau ji privalo "prisitaikyti" prie iškeltų reikalavimų, t.y. peržiūros programa automatiškai perskaičiuoja visas kviečiamojoje procedūroje naudojamas taktų reikšmes, stengdamasi įsprausti ją į nurodytus *duration* rėmus. Jei tarpinės animacijos trukmė nenurodoma, ji apskaičiuojama automatiškai, naudojant 3-iame skyriuje aprašytus metodus. Jei nenurodomas *repeat*, animacija vykdoma lygiai vieną kartą.

Kviečiamąją animaciją būtina vienareikšmiškai identifikuoti atribute "name". Jei naudojamas nepilnai kvalifikuotas vardas (pavyzdžiui, "walk"), tokio pavadinimo animacija turi būti vienintelė ne tik kviečiančiajame, bet ir visuose naudojamuose moduliuose bei visuose užkrautuose žemo lygio *.skeleton* animacijų failuose. Jeigu toks kvietimas sukeltų dviprasmybę, reikia nurodyti pilnai kvalifikuotą animacijos vardą, pradedant katalogų keliu, įrašant vardą modulio, kuriame yra pageidaujama animacija, ir baigiant pačios animacijos vardu. Visi kelio komponentai atskiriami tašku. Pavyzdžiui:

```
<call name="my_library.simple_animations.Steps.walk" duration="2"/>
```

# 5. Procedūrinių animacijų peržiūros programa

Programa, pasitelkusi OGRE grafikos variklį, sukuria trimatį humanoido modelio atvaizdą ir jį animuoja pagal nurodytą procedūrinės animacijos failą (PHA). Pirmiausia nuskaitomas ir apdorojamas PHA failas. Apdorojama PHA seka, sudaromas reikalingų *.skeleton* failų sąrašas bei nuosekliai vykdomų bazinių animacijų sąrašas. Tuomet užkraunamas modelis ir jo medžiaga. Pagal anksčiau sudarytą sąrašą užkraunami visi reikalingi .skeleton failai bei iš jų paimamos animacijos prijungiamos vaizduojamam modeliui. Toliau inicijuojamas tarpinių animacijų vykdymo būdas bei pačios tarpinės animacijos. Galiausiai ištaisomi neteisingai įrašyti kvaternionai.

## 5.1. Programos galimybės

- Galimybė pasirinkti modelį iš modelių bibliotekos.
- Galimybė pavaizduoti PHA animaciją pasirinktam modeliui.
- Galimybė modelį (arba modelius) apžiūrėti iš įvairių pusių. Numatome įgyvendinti interaktyvų kameros padėčių valdymą.
- Galimybė reguliuoti animacijos greitį – pagreitinti, sulėtinti arba sustabdyti.

## 5.2. Programos konfigūracija

Pradinius duomenis programa nuskaito iš *pha.config.xml* bei *resources.cfg* konfigūracinių failų. Resources.cfg nurodomas katalogas (arba katalogai) kuriame laikomi modeliai. Šiame kataloge laikomi patys modelių failai OGRE formatu (*.mesh*) bei visi su jais susiję failai. Pirmiausia, šalia modelio privalo būti bazinis *.skeleton* failas. Jame animacijų gali ir nebūti, svarbu kad būtų kaulų struktūra. Programoje animuojamas modelis užkraunamas kartu su savo kaulų struktūra iš *.skeleton* failo, vėliau prie jos bandoma priderinti animacijas iš kitų failų. Taip pat modelių kataloge laikomi medžiagos failai (*.material*) bei reikalingi tekstūrų failai.

Konkretus modelis, kurį norima vaizduoti programoje nurodomas *pha.config.xml* faile. Šis failas turi būti xml sintaksės, o jame privalo būti tokie elementai: *rootDir*, *model*, *material*, *transition* ir *file*. Užkraunamas modelis nurodomas kaip *model* elemento atributas, rašomas tik *.mesh* failo pavadnimas be kelio. Modelio medžiaga nurodoma elemento *material* atributu, rašomas medžiagos pavadinimas iš kurio nors modelių katalogo *.material* failo.

Elemento *rootDir* atributu nurodomas šakninis visų animacijų katalogas. Visos PHA ir bazinės *.skeleton* animacijos turi būti šiame kataloge arba kataloguose, esančiuose šiame kataloge.

Elemento *transition* atributu norodomas tarpinių animacijų vykdymo būdas. Galimos dvi reikšmės. Reikšmė „1“ generuoja tokias tarpines animacijas, kurios vykdomos pirmo jungiamojo segmento pabaigoje, t. y. nemodifikuojama jungiamųjų segmentų pradžia. Reikšmė „2” generuoja tokias tarpines animacijas, kurios vykdomos pirmo jungiamojo segmento pabaigoje bei antro pradžioje, t. y. modifikuojama jungiamųjų segmentų pradžia bei pabaiga.

Elemento *file* atributu nurodomas vaizduojamas PHA failas. Jis turi būti rašomas reliatyviai nuo šakninio katalogo. Vaizduojams failas taip pat gali būti nurodomas iš komandinės eilutės. Tokiu atveju jis gali būti tik šakniniame kataloge. Jei komandinė eilutė tuščia, vaizduojamas failas nuskaitomas iš konfigūracinio failo.

### 5.3. Vartotojo sąsaja

Animacijos vaizduojamos nuosekliai įterpiant tarpinius segmentu. Galima greitinti arba lėtinti atlikimą (klavišai **B** ir **V**). Sustabdyti ir vėl paleisti animaciją galima klavišu **P**. **Space** klavišu animacijos paleidžiamos nuo pradžių. Klavišo **I** paspaudimai duoda įvairias posūkių bei postūmių interpoliacijų kombinacijas. Pagal nutylėjimą postūmiai bei posūkiai interpoliuojami tiesiškai.

Kameros pozicija keičiama rodyklių klavišais – galima stumdyti į kairę arba dešinę, tolyn arba artyn. Klavišais **PageUp** ir **PageDown** kamera stumdoma aukštyn arba žemyn. Galima įvairiomis kryptimis pasukti modelį, klavišai **J** ir **L** suks apie Y ašį, o klavišų **K** ir **O** paspaudimais galima sukti apie X ašį.

### 5.4. XML apdorojimo modulis

Turint procedūrinių humanoidų animacijų failus xml formatu iškilo poreikis priemonės, kurios pagalba būtų nuskaitomi vaizdavimo programai reikalingi duomenys ir pateikiami patogia forma.

Kadangi egzistuoja daug priemonių darbui su xml formatu, būtų buvę kvaila jomis nepasinaudoti. Tačiau iš gausybės reikėjo pasirinkti kažką konkretaus, kas galėtų mums labiausiai tikti. Kadangi vaizdavimo programą rašėm “Microsoft Visual C++ 2005 Express

Edition", tai natūralu, kad svarstėm galimybę naudoti ".Net" priemones darbui su xml formato failais, taip pat Apache'io "Xerces C++ Parser" dėl jo populiarumo. Galiausiai pasirinkom "TinyXML", kurio funcionalumo mums užteko (nenaudojom nei DTD, nei XSL) ir kuris suprantamesnis bei parastesnis lyginant su "Xerces C++ Parser".

### *5.4.1. Modulio naudojimas*

Sukurtame modulyje XML failo apdorojimui naudojama "phaParser" klasė. Jos "parse" metodas, kuriam parametru nurodomas konkretus failas atlieka visą darbą. Galutiniame rezultate reikalingi duomenys patalpinami atitinkamuose "phaParser" klasės atributuose. Animacijų takto ilgis sekundėmis saugomas "tick" atribute, failų sąrašas ("vector"), kuriame saugojamos jungiamos animacijos suformuojamas "files" atribute, o pagrindinis sąrašas - seka (atributas "set") sudaromas iš objektų (apjungiamų animacijų). Kiekvienas sąrašo animacijos objektas turi visą reikalingą informaciją: animacijos pavadinimą, trukmę taktais ir tarpinės (jungiamosios) animacijos trukmę, kuri sektų po aprašytosios animacijos, jei tokia būtų bei kartojimų skaičius, jei toks buvo nurodytas. Kad nesusidarytų klaidingas įspūdis, paminėsiu, kad "phaParser" klasėje "parse" metodas nėra vienintelis, tačiau kiti metodai yra pagalbiniai minėtajam.

### *5.4.2. PHA failų iškvietimas*

Kadangi formate numatyta, jog viename PHA faile gali būti iškviečiamas kitas PHA failas, o tame kitame dar kitas ir taip toliau, tai natūralu, kad failų nuskaityme prireikė rekursijos. Kartu su rekursija iškilo amžino ciklo, šiuo atveju rekursijos, grėsmė. Jos valdymui prireikė tikrinimo mechamizmo. Kadangi iš iškviečiamo failo reikalingos visos procedūros išskyrus pagrindinę (pavadinimu "main"), o jų dubliavimas nepageidautinas, tai prieš nuskaitant kiekvieną PHA failą patikrindami ar jis dar nebuvo nuskaitytas išsprendžiame ne tik galimos amžinos rekursijos grėsmę, bet ir išvengiame pakartotinio procedūrų nuskaitymo. Kiekvieno nuskaitomo failo pavadinimą įrašome į sąrašą.

Dubliavimo išvengimui tą patį principą naudojame ir sudarydami failų pavadinimų, kuriuose aprašytos jungiamos animacijos sąrašą.

### *5.4.3. Procedūrų nuskaitymas*

Nuskaitytos procedūros surašomos į sąrašą (išskyrus “main” procedūrą – atskiras sąrašas). Procedūra - kitų procedūrų ir apjungiamų animacijų seka. Tad galutiniam rezultate gaunam sąrašą iš sąrašų. Iš pagrindinės procedūros sudarome atskirą sąrašą. Norėdami gauti apjungiamų animacijų seką nuskaitinėjam iš pagrindinės (“main”) procedūros animacijas ir tikrinam ar jų nėra procedūrų sąraše, jei nėra, tuomet traktuojame kaip animaciją ir įrašom į sąrašą, jei yra susirandame ją ir ją sudarančių animacijų sąrašą ir paeiliui tikrinam ar animacijos nėra procedūros, ir taip toliau. Galime pastebėt, kad ir šiuo atveju naudojama rekursija. Čia taip pat reikia naudoti rekursijos valdymo priemones, kurios neleistų susidaryt amžinai rekursijai procedūrų iškvietimuose. Skirtingai nė su failais, ta pati procedūra gali būti iškviečiama daug kartų skitingose vietose nesudarydama jokių problemų, todėl šiuo atveju toks pat rekursijos tikrinimas kaip kad su failais nebūtų teisingas. Tad tikrinimo mechanizmas šiek tiek kitoks: nuskaitoma procedūra įrašoma į nuskaitytų procedūrų sąrašą, tačiau ją nuskaičius iš šio sąrašo pašalinama. Tokiu būdu gauname, kad sąraše egsiztuoja tik procedūros iškviečiusios ją (tėvinės) ir pati procedūra. Tad prieš nuskaitant kiekvieną procedūrą patikriname ar jos nėra šiame sąraše ir tokiu būdu išvengiame amžinos procedurų rekursijos.

### *5.4.4. Taktų reikšmių perskaičiavimai*

Kiekviena animacija turi parametrą, nurodantį jos trukmę taktais. Animacijų procedūra taip pat yra animacija ir ji nėra išimtis. Todėl gali natūraliai kilti klausimas: Kas jeigu nurodytas procedūros trukmės laikas skiriasi nuo ją sudarančių animacijų laiko trukmių sumos? Modulis netgi netikrina ar laikai yra lygūs. Ji perskaičiuoja procedūros animacijų laikus taip, kad jų suma būtų lygi proceduroje nurodytai trukmei. Perskaičiavimas atliekamas gana paprastai: suskaičiuojama procedūros animacijų laiko suma ir iš jos padalimamas procedūroje nurodytas trukmės laikas taip gaunant koeficientą, iš kurio padauginus visų tos procedūros animacijų laiko trukmes gaunamos naujos, kurios jom priskiriamos neatsižvelgiant ar tos animacijos tėra apjungiamos animacijos ar procedūros.

Animacijos turi dar vieną laiko parametrą, kuris nurodo galimos tarpinės animacijos trukmė. Būtų viskas lyg ir aišku, tik vėl klausimas iškyla procedūrų atveju. Kadangi procedūra galutiniam rezultate susiveda į paprastų apjungiamų animacijų seką, tai ši reikšmė (jeigu nurodyta) priskiriama paskutiniajai esančiai toje sekoje. Tačiau ir tai neviskas.

Procedūra turi parametrą, kuris nurodo kiek kartų iš eilės jinai bus iškviečiama (kartojama). Šiuo atveju atsiranda du galimi variantai. Galima procedūroje nurodytą tarpinės animacijos trukmės reikšmę priskirti kiekvieno ciklo paskutinei animacijai arba paskutinio ciklo paskutinei animacijai. Antrasis variantas pasirodė logiškesnis, todėl buvo pasirinktas pastarasis.

# 6. Ateities darbai

Šiame skyriuje pateikiami sumanymai, kuriuos matome kaip natūralų mūsų darbo tęsinį.

## 6.1. Procedūrinių animacijų formato patobulinimai

Šiuo metu PHA formate yra įgyvendinti vos keli procedūrinio programavimo principai: būtent, moduliarumas ir procedūrinė abstrakcija. Ateityje procedūros galėtų turėti parametrus, taip suteikdamos galimybę dar lanksčiau kontroliuoti judesių sekas. Tarkim, vienu parametru būtų galima nurodyti *repeat* atributo reikšmę, tokiu būdu valdant atitinkamo judesio kartojimų skaičių. Tokiu atveju reikėtų įvesti ir kintamojo sąvoką. O turint kintamuosius, būtų galima realizuoti daugiau valdymo struktūrų: sąlyginį vykdymą (kviesti animacijų bloką ar ne) bei ciklus (kiek kartų kartoti animacijų bloką).

Jeigu kai kuriose sudėtingose judesių programose prireiktų dar didesnio lankstumo, vien procedūrinio programavimo principų gali nebeužtekti. Tokiose situacijose galime neišsisukti be objektinės paradigmos. Tarkim, modulyje (klasėje) "Swing_Dances" turime ilgą procedūrinę šokio animaciją "Shim_Sham", kurios vienas iš fragmentų, realizuotas procedūroje "Half_Break", gali būti atliekamas keliomis skirtingomis variacijomis. Tarkim, kiekvienas šokių mokytojas turi savo unikalią šio judesio variaciją, tačiau jis nenori tik dėl jos vienos perrašyti viso ilgo šokio. Čia galėtų gelbėti polimorfizmas. Procedūrą "Half_Break" deklaravus kaip virtualų metodą, būtų galima sukurti savo klasę pavadinimu "My_Swing_Dances", kuri paveldėtų klasę "Swing_Dances" ir perrašytų metodą "Half_Break" taip, kaip pageidautų klasės kūrėjas - šokių mokytojas. Tokiu būdu, atlikę tik minimalius pakeitimus, gautume viso šokio variaciją.

## 6.2. Peržiūros programos vystymas

Programa vartotojui suteiks galimybę pasirinkti humanoido modelį ir, pasitelkus OGRE grafikos variklį, sukurs jo trimatį atvaizdą. Vartotojui bus pasiūlytas animacijų sąrašas, kurį sudarys animacijų bibliotekoje esančios animacijos. Iš sąrašo pasirinkus animaciją, programa ją pritaikys pasirinktam humanoidui ir parodys rezultatą. Kadangi animacija nepriklausys nuo modelio tinklelio geometrijos (*mesh*), ant visų modelių bus galima išbandyti bet kokią animaciją.

### *6.2.1. Papildomi reikalavimai peržiūros programai*

- Galimybė tuo pačiu metu animuoti kelis modelius.
- Galimybė vaizduoti žemo lygio animaciją pasirinktam modeliui.
- Galimybė vykdyti animaciją atbulai.
- Galimybė vaizduoti animaciją be tarpinių interpoliacinių dalių.
- Galimybė peržiūrėti animacijos fragmentą

### *6.2.2. XML modulio pakeitimai*

Šiame darbe naudoti xml formato failai nebuvo dideli (tik kelių kilobaitų dydžio). Su jais problemų nebuvo, tačiau mes nežinome kaip sėkmingai būtų apdorojami tikrai dideli failai, kurių dydis skaičiuojamas megabaitais. Kita vertus, prireikus dirbti su dideliais failais, būtų naudojami ne paprasti tekstiniai, o binariniai failai (kaip kad mūsų naudojam grafikos varikliuke “OGRE”).

Kadangi pats formatas dar turėtų tobulėti (pavyzdžiui sąlyginis vykdymas), jei tik bus naudojamas, tai turėtų atitinkamai tobulėti ir XML apdorojimo modulis.

## 6.3. Procedūrinių animacijų vizualus redaktorius

Programa leis vartotojui apjungti animacijas į animacijų seką ir išsaugoti jas kaip vientisą animaciją, neprarandant struktūrinės informacijos. Tokiu būdu atsiras galimybė iš pateiktų pradinių elementarių animacijų gauti vis sudėtingesnes. Kad geriau įsivaizduotume, pateiksime paprastą pavyzdį. Tarkime, pradžioje turime tokį animacijų sąrašą: "žingsnis dešine koja", "žingsnis kaire koja". Iš šių animacijų sudarę seką, kur paeiliui pirmą animaciją keičia antra, gausime naują animaciją, kurią galima būtų pavadinti "ėjimu". Panašiai gautume ir "bėgimo" animaciją, tik jos pradinės animacijos turėtų būti: "šuoliukas dešine koja" ir "šuoliukas kaire koja". Toks animacijų apjungimas būtų paprasčiausias, t.y. kai baigiasi viena animacija, iš karto prasideda kita. Tačiau to nepakaks realistiškai ėjimo/bėgimo simuliacijai. Humanoidui einant ar bėgant, juda ne tik kojos, bet ir rankos, ir kitos kūno dalys, ir jos visos juda tuo pat metu. Tokiai animacijai sukurti reikės lygiagrečiai vykdyti kelias animacijas. Ėjime žingsnis kaire koja turi būti atliekamas kartu su dešinės rankos mostu, ir atvirkščiai. Tiek mosto, tiek žingsnio animacija privalo trukti tiek pat laiko.

Iš pavyzdžių matome, kad naujų animacijų kūrimui apjungiant jau esamas, neužteks prie vienos prilipdyti kitą, taip pat bus poreikis animacijas sulieti. Be to, gali tekti ištempti ar suspausti animacijų trukmę bei keisti kitus parametrus.

Laikui bėgant animacijų sąrašas nepaliaujamai augs. Visiškai identiškų animacijų gal ir neatsiras, bet labai panašių bibliotekoje gali susikaupti nemažai. Tad iškils poreikis kažkokiu būdu jas grupuoti ir klasifikuoti. Tai būtų galima palikti ir pačiam vartotojui, tačiau kartu norėtųsi šį klasifikacijos procesą bent minimaliai specifikuoti, t.y. įvesti konkrečias taisykles, kurios vėliau, tikėtina, praverstų kuriant išbaigtas animacijų bibliotekas.

### *6.3.1. Reikalavimai vizualiam redaktoriui*

- Galimybė pasirinkti žemo lygio animaciją iš animacijų bibliotekos. Leidžiama pasirinkti vieną, kelias arba nė vienos žemo lygio animacijos.
- Galimybė pasirinkti PHA animaciją iš animacijų bibliotekos. Leidžiama pasirinkti vieną, kelias arba nė vienos PHA animacijos. Neleidžiama į PHA animacijos vidų įdėti nuorodą į ją pačią, t. y. redaguojamame faile negali būti nuorodos į jį patį. Taip pat neleidžiama įterpti tų PHA animacijų, kuriose yra nuorodų į redaguojamą animaciją.
- Galimybė jungti animacijas nuosekliai.
- Galimybė jungti animacijas lygiagrečiai. Numatome lygiagrečiai vykdomas animacijas jungti į kelias grupes (pvz.: kairės pėdos, dešinės pėdos, kairės kojos, ir pan.).
- Galimybė redaguoti kiekvienos animacijos trukmę (pagreitinti arba sulėtinti), sulyginti lygiagrečių animacijų grupių trukmes.
- Galimybė redaguoti tarpinių interpoliacinių animacijų trukmę arba palikti nustatyti kompiuteriui.
- Galimybė nustatyti, kiek kartų animacija ar jų grupė bus kartojama (ciklas). Kartoti galima vieną arba daugiau kartų.
- Redaguojama animacijų seka išsaugoma PHA formatu.
- Detali ir smulki programos dokumentacija.

## Išvados

Trimačių humanoidų modelius bei animacijas patogiausia kurti specializuotomis programomis: "3D Studio Max", "Maya", "Blender" ir kt. Tačiau šios programos pritaikytos galutiniam rezultatui, t.y. vientisai animacijai, gauti. Mūsų sukurtas procedūrinių animacijų formatas ir tokių animacijų peržiūros programinė įranga praverčia tuomet, kai yra prasminga turėti ne tik galutinę judesių seką, bet ir jos komponentes - individualius judesius ar jų posekius, kai yra poreikis tuos judesius iš naujo kombinuoti pasirinkta tvarka, negriaunant kiekvieno jų vidinės struktūros.

Kai į humanoido animaciją žvelgiame ne kaip į užbaigtą ir nebekeičiamą judesį, bet kaip į kelių smulkesnių judesių kombinaciją, gauname kur kas daugiau lankstumo. Tokia abstrakcija leidžia animatoriams pakartotinai panaudoti anksčiau sukurtus judesius. Be to, animacijų procedūros keitimas ar papildymas yra nepalyginamai paprastesnis nei vientisos animacijos keitimas. Minėtos procedūrinių animacijų savybės ypač praverstų kuriant sudėtingas judesių programas, kurios gali būti dažnai keičiamos ar pildomos, taip pat tokias, kurios dažnai kartoja tuos pačius judesius, arba kai kelios judesių programos turi naudoti tas pačias judesių sekas ar posekius. Pavyzdžiui, tai puikiai tiktų šokių modeliavimui, nes šokis kaip tik ir yra viena ilga judesių programa, susidedanti iš tam tikra tvarka surikiuotų, pakartotinai naudojamų komponenčių.

Turėdami omeny potencialų taikymą šokiams modeliuoti, įvedėme *takto* sąvoką. Visų procedūriškai kviečiamų animacijų trukmės nurodomos ne sekundėmis, bet taktais. Tokiu būdu gaunamas itin lankstus animacijų trukmės valdymas. Taktas šokyje yra tiek pat reikšmingas kiek ir taktas muzikoje.

Mums pavyko realizuoti vizualiai glodų animacijų apjungimą, kuris patenkinamai veikia net ir tuo atveju, kai nieko nežinome nei apie apjungiamų judesių specifiką, nei apie jų pradines bei galutines padėtis. Tą įgyvendinome kiekvienai gretimų judesių porai generuodami tarpinę animaciją, naudodami kvaterioninės interpoliacijos metodą, ir suliedami ją su pirmosios animacijos pabaiga ir/arba antrosios pradžia, tolygiai didindami suliejimo svorį. Vis dėlto yra situacijų, kuriose mūsų išbandyti apjungimo metodai neduoda pageidaujamo rezultato, todėl reikalingas tolesnis tyrimas.

Šiuo darbu taip pat parodėme, kad animacijų atskyrimas nuo modelių yra įmanomas ir praktiškai įgyvendinamas, jeigu naudojame visiems modeliams bendrą humanoido skeleto struktūrą - mūsų atveju tai H-Anim. Tai leidžia nepriklausomiems modeliuotojams ir

animatoriams keistis sukurtomis animacijomis ir/arba modeliais be papildomų suderinamumo rūpesčių.

Tikimės, kad mūsų darbas atskleis procedūrinių animacijų galimybes ir privalumus, o pati idėja išpopuliarės ir bus vystoma kartu su sukurtu formatu. Galbūt net atsiras standartinės humanoido judesių bibliotekos.

## Literatūros sąrašas


[AGILE] AgileDelta, Inc. Efficient XML
http://www.agiledelta.com/efficientxml.html

[COL] COLLADA 1.4 Specification
http://www.khronos.org/cgi-bin/fetch/fetch.cgi?collada_spec_1_4

[DN] Dance notation
http://en.wikipedia.org/wiki/Dance_notation

[EXI] Analysis of the Efficient XML Interchange Working Group's Measurements
http://www.w3.org/XML/EXI/report.html

[FLUX] MediaMachines, Inc. Flux Studio
http://www.mediamachines.com/make.php

[HANIM] Humanoid Animation Working Group:
http://www.h-anim.org

[JBL] Jonathan Blow. Hacking Quaternions. 2004
http://number-none.com/product/Hacking%20Quaternions/index.html

[KHR] Khronos Group
http://www.khronos.org/

[LAB] Labanotation
http://web.archive.org/web/20031008203947/http://www.dance.ohio-state.edu/labanlab/

[MBA] Mantas Barčius. Kursinis darbas „Posūkio interpoliacija erdvėje“, Vilniaus Univeritetas, 2006

[OGRE] OGRE – Object-Oriented Graphics Rendering Engine
http://www.ogre3d.org

[WEB3D] Web3D Consortium
http://www.web3d.org

[X3D] Extensible 3D (X3D) Specifications
http://www.web3d.org/x3d/specifications/
http://www.web3d.org/x3d/content/examples/Basic/HumanoidAnimation/index.html

[XML] Extensible Markup Language (XML) 1.0 (Third Edition):
http://www.w3.org/TR/2004/REC-xml-20040204/

[XSL] The Extensible Stylesheet Language Family (XSL):
http://www.w3.org/Style/XSL/

**Priedai**

Priedas A. Pilna H-Anim humanoido sąnarių ir kaulų (segmentų) hierarchija:

```
HumanoidRoot : sacrum
 sacroiliac : pelvis
 |  l_hip : l_thigh
 |    l_knee : l_calf
 |      l_ankle : l_hindfoot
 |        l_subtalar : l_midproximal
 |          l_midtarsal : l_middistal
 |            l_metatarsal : l_forefoot
 |  r_hip : r_thigh
 |    r_knee : r_calf
 |      r_ankle : r_hindfoot
 |        r_subtalar :  r_midproximal
 |          r_midtarsal : r_middistal
 |            r_metatarsal : r_forefoot
 vl5  : l5
  vl4 : l4
    vl3  : l3
      vl2 : l2
        vl1  : l1
          vt12 : t12
            vt11 : t11
              vt10  : t10
                vt9 : t9
                  vt8 : t8
                    vt7 : t7
                      vt6  : t6
                        vt5 : t5
                          vt4 : t4
                            vt3 : t3
                              vt2 : t2
                                vt1  : t1
                                  vc7 : c7
                                  | vc6 : c6
                                  |   vc5 : c5
                                  |     vc4  : c4
                                  |       vc3 : c3
                                  |         vc2  : c2
                                  |           vc1 : c1
                                  |             skullbase : skull
                                  |               l_eyelid_joint : l_eyelid
                                  |               r_eyelid_joint : r_eyelid
                                  |               l_eyeball_joint : l_eyeball
                                  |               r_eyeball_joint : r_eyeball
                                  |               l_eyebrow_joint : l_eyebrow
                                  |               r_eyebrow_joint : r_eyebrow
                                  |               temporomandibular : jaw
```

```
l_sternoclavicular : l_clavicle
| l_acromioclavicular : l_scapula
|   l_shoulder : l_upperarm
|     l_elbow : l_forearm
|       l_wrist : l_hand
|         l_thumb1 : l_thumb_metacarpal
|           l_thumb2 : l_thumb_proximal
|             l_thumb3 : l_thumb_distal
|         l_index0 : l_index_metacarpal
|           l_index1 : l_index_proximal
|             l_index2 : l_index_middle
|               l_index3 : l_index_distal
|         l_middle0 : l_middle_metacarpal
|           l_middle1 : l_middle_proximal
|             l_middle2 : l_middle_middle
|               l_middle3 : l_middle_distal
|         l_ring0 : l_ring_metacarpal
|           l_ring1 : l_ring_proximal
|             l_ring2 l_ring_middle
|               l_ring3 : l_ring_distal
|         l_pinky0 : l_pinky_metacarpal
|           l_pinky1 : l_pinky_proximal
|             l_pinky2 : l_pinky_middle
|               l_pinky3 : l_pinky_distal
r_sternoclavicular : r_clavicle
  r_acromioclavicular : r_scapula
    r_shoulder : r_upperarm
      r_elbow : r_forearm
        r_wrist : r_hand
          r_thumb1 : r_thumb_metacarpal
            r_thumb2 : r_thumb_proximal
              r_thumb3 : r_thumb_distal
          r_index0 : r_index_metacarpal
            r_index1 : r_index_proximal
              r_index2 : r_index_middle
                r_index3 : r_index_distal
          r_middle0 : r_middle_metacarpal
            r_middle1 : r_middle_proximal
              r_middle2 : r_middle_middle
                r_middle3 : r_middle_distal
          r_ring0 : r_ring_metacarpal
            r_ring1 : r_ring_proximal
              r_ring2 : r_ring_middle
                r_ring3 : r_ring_distal
          r_pinky0 : r_pinky_metacarpal
            r_pinky1 : r_pinky_proximal
              r_pinky2 : r_pinky_middle
                r_pinky3 : r_pinky_distal
```

Priedas B. Procedūrinių humanoidų animacijų (PHA) failo pavyzdys.

```
<?xml version="1.0"?>
<pha tickLength="0.8">

  <load src="feet_movements1.Steps"/>
  <load src="feet_movements2.Cool_Steps"/>

  <include src="basic_movements.Turns"/>
  <include src="basic_movements.More_Turns"/>

  <procedure name="normal_walk">
    <call name="feet_movements1.Steps.left_step" duration="1"/>
    <call name="feet_movements1.Steps.right_step" duration="1"/>
  </procedure>

  <procedure name="limp_walk">
    <call name="feet_movements1.Steps.left_step" duration="1.3"/>
    <call name="feet_movements1.Steps.right_step" duration="0.7"/>
  </procedure>

  <procedure name="cool_walk">
    <call name="feet_movements2.Cool_Steps.left_step" duration="1.5"/>
    <call name="feet_movements2.Cool_Steps.right_step" duration="1.5"/>
  </procedure>

  <procedure name="main">
    <call name="normal_walk" duration="1.8"/>
    <call name="normal_walk" duration="2.3"/>
    <call name="normal_walk" duration="2" repeat="5"/>
    <call name="turn_left" duration="4" transition="0.5"/>
    <call name="limp_walk" duration="2" transition="0.5" repeat="5"/>
    <call name="turn_right" duration="4"/>
  </procedure>

</pha>
```