KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Tomas Balzaris

# Kadrinės ir skeletinės animacijos metodų palyginimas ir jų taikymas praktikoje

Magistro darbas

Darbo vadovas
dr. Tomas Blažauskas

Kaunas, 2006

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Tomas Balzaris

# Kadrinės ir skeletinės animacijos metodų palyginimas ir jų taikymas praktikoje

Magistro darbas

Kalbos konsultantė
Lietuvių k. katedros dėst.
I. Mickienė
2006-05-28

Vadovas
dr. Tomas Blažauskas
2006-05-28

Recenzentas
prof. E. Bareiša
2006-05-28

Atliko
IFM-0/1 gr. stud.
Tomas Balzaris
2006-05-28

Kaunas, 2006

## SUMMARY

### Frame and Skeletal Animation Techniques' Comparison and Implementation

Modern technology and computer science is spreading and developing very fast. The use and facilities evolve too. One of the fields where computer science is widely used – computer games. Computer animation makes games more interactive and attractive to the user. The purpose of this project was to review and analyze the animation techniques used in modern games development. Review file formats witch are used in computer games to store game objects and animation data. Try to point out the strengths and weaknesses of animation techniques. Main attention was committed to skeletal and key frame animation. Implement skeletal and key frame animation techniques.

# TURINYS

## Paveikslėlių sąrašas



## Lentelių sąrašas

# 1. Įvadas

Vis sparčiau ir sparčiau progresuojant technologijoms ir kompiuterių mokslui, kartu plečiasi ir vystosi jų panaudojimo galimybės bei taikymai. Vienas iš tokių taikymų yra kompiuteriniai žaidimai. Ši rinka kasmet plečiasi ir didėja, apimdama vis platesnes sritis: nuo žaidimų internete ir mobiliuosiuose telefonuose iki naujausias technologijas ir aparatūrinę įrangą naudojančių žaidimų. Žaidimų rinka apima labai plačią informacinių technologijų sritį nuo specializuotų žaidimų kompiuterių gamybos (ang. console), grafinių plokščių, garso plokščių, iki specializuotos programinės įrangos ir programavimo kalbų.

Pastaruoju metu sparčiai besivystant žaidimų industrijai, tobulėja ir žaidimuose naudojama animacija. Tai pasiekiama daugiausia dėl to, kad atsiranda vis galingesni procesoriai, kurie leidžia atlikti nemažai skaičiavimų, be to vaizdo apdorojimas jau senokai patikėtas vaizdo plokštėms. Tikroviškesnė animacija žaidėjams leidžia labiau įsijausti į patį žaidimą, tuo žaidimo aplinką dar labiau priartindama prie tikrovės, padarydama patį žaidimą dar patrauklesniu bei įdomesniu.

Taigi, kas yra animacija? Animacija remiasi žmogaus akies bei smegenų fiziologija. Žmogaus smegenys išlaiko objekto vaizdą, kurio realybėje tuo tarpu jau nebėra. Šis fenomenas, vadinamas – vaizdo išlaikymu atmintyje, susijęs su tinklainės struktūra bei neuronais. Tolygi animacija pasiekiama be pertraukos rodant vaizdus didesniu nei išlaikymo dažniu. Šis veiksmas, kurio metu seną vaizdą pakeičia naujas, prieš jam dingstant iš atminties, mums sukuria judėjimo iliuziją. Vaizdo išlaikymas tęsiasi kelis šimtąsias sekundės dalis. Eksperimentais nustatyta, kad norint pasiekti tolygią animaciją reikia rodyti vaizdą 22 – 30 kadrų (vaizdų) per sekundę dažniu.

Sakykime, kad animacija bus rodoma 20 kadrų per sekundę dažniu, vadinasi kiekvienas kadras turės būti atnaujintas bei atvaizduotas per 1/20 sekundės. Kai kuriems atvaizdavimo šablonams reikia, kad senesni kadrai būtų panaikinami, prieš atvaizduojant naująjį, nes kitu atveju, vaizduoklio (televizoriaus ir pan.) ekrane, paliekamas vaizduotų objektų pėdsakas. Dėl šios priežasties kadrų atnaujinimo operacijų seka: perpiešti – ištrinti. Kadro perpiešimui duotas laikas pusė laiko skirto kadro atnaujinimui ir atvaizdavimui, šiuo atveju tai būtų 1/40 sekundės.

Kompiuteriniuose žaidimuose aukštas kadrų dažnis reikalauja nemažai procesoriaus resursų. Taigi programuotojams tenka balansuoti tarp kadrų dažnio ir sistemos greičio, bei resursų. Štai kodėl dauguma žaidimų suteikia galimybę pasirinkti skirtingą grafikos lygį. Naudojant mažesnę skiriamąją gebą ir paprastesnę grafiką, padidėja kadrų dažnis ir sukuriama tolygesnė animacija.

Objektų, naudojamų animacijoje, saugojimui naudojami įvairiausi rinkmenų formatai. Dažnai kiekvienas žaidimas turi savo rinkmenos tipą, kurioje saugoma informacija apie žaidimo objektą ir jo animaciją.

Šiuo darbu siekiama detaliau susipažinti su animacijos taikymu kompiuteriniuose žaidimuose; apžvelgti naudojamas animacijos sistemas, jų silpnąsias bei stipriąsias savybes. Apžvelgti, animacijos saugojimui, naudojamų rinkmenų tipus, bei pabandyti išskirti jų pagrindinius bruožus. Realizuojant pasirinktus animacijos metodus, buvo pasiūlyti realizacijos algoritmai, plačiau aptariami 3 skyriuje. Darbas atliktas naudojant RUP (Rational Unified Process) modelį, *„MS Visual Studio 2003“* programavimo terpę, *„DirectX 9“* biblioteką. Animacijos kūrimui buvo panaudotas *„3D Studio MAX 7“* programinis paketas.

# 2. Animacijos metodų analizė

## 2.1. Duomenų saugojimo failai

Žaidimo metu naudojamiems objektams bei veikėjams perkelti iš programos, kurioje jie buvo sukurti, į žaidimą naudojami keli failų formatai.

**1 lentelė. Failų formatai**

| Rinkmenos formatas | Aprašymas |
|---|---|
| .x | X rinkmenos struktūra paremta tam tikrais šablonais, be to vartotojas ir pats gali susikurti šablonus, kurie gali būti nenumatyti pradinėje bibliotekoje. Šablonas aprašo, kaip saugomas objektas. |
| .MD2 | MD2 formatas sukurtas „Quake 2“ žaidimui |
| .MD3 | MD3 formatas, naujesnė MD2 formato versija, naudojama „Quake 3“ ir daugelyje kitų žaidimų. Rinkmenoje animacija saugoma kadrais, kurie turi atitinkamus pavadinimus. Raktinių kadrų naudojimas labai palengvina animacijos valdymą. |
| .3ds | Grafinio paketo „3D Studio Max“ rinkmenos tipas. |

Kaip matome iš 1 lentelės, pastaruoju metu sukurta nemažai rinkmenų formatų, kurie gali išsaugoti paviršiaus trikampių tinklus (skin meshes), bet vienas iš paprastesnių būdų X failo naudojimas, tuo labiau, kad jį palaiko ir „DirectX“ biblioteka. Šiose rinkmenose gali būti saugomi netik paviršiaus trikampių tinklai (skin meshes), bet ir statiniai trikampių tinklai. X rinkmenoje duomenys saugomi kaip šablonų rinkinys. Šablonai turi aprašymus, kuriuose kaip bus talpinami duomenys šablone. Kiekvienam specifiniam duomenų tipui naudojamas atskiras šablonas. Šablonai gali būti sudaryti iš šablonų, tai leidžia mums sukurti hierarchines scenas. Plačiau apibūdinsiu keletą šablonų, kuriuos naudosiu savo darbe.

**Frame**

Šablonas naudojamas kadro duomenų saugojimui. Kadrai turi savo transformacijų matricas, juose taip pat gali būti aprašyti kiti šablonai.

**FrameTransformMatrix**

Kadro transformacijų matrica, talpinama `Frame` šablone.

**Mesh**

Statinis trikampių tinklas su medžiagomis (materials) saugomas šiame šablone. Paviršiaus trikampių tinklų atveju, visas veikėjas bus saugomas kaip vientisas trikampių tinklas. Šis tinklas bus suskaidytas į poaibius, kurių kiekvienas bus veikiamas konkrečių kaulų.

**XSkinMeshHeader**

Šiame šablone saugoma svarbiausia informacija susijusi su eksportuotu paviršiaus trikampių tinklu.

**SkinWeights**

Reali informacija susijusi su paviršiaus trikampių tinklu saugoma šiame šablone. Šis šablonas apibrėžia, kaip konkretus kaulas gali paveikti trikampių tinklą. Jis sukuriamas atskirai kiekvienam kaului, kuris veikia trikampių tinklą. Pavyzdžiui, jei turime 16 kaulų veikiančių trikampių tinklą, tai `Mesh` šablonas turės 16 `SkinWeights` šablonų.

Vienintelis skirtumas tarp paviršiaus trikampių tinklų ir statinių trikampių tinklų – tai `XSkinMeshHeader` ir `SkinWeights` šablonai.

**AnimationKey**

Šablonas saugantis animacijos raktus. Kiekvienu atveju šablone išsaugomas vis skirtingas animacijos raktas (pasukimo, pozicijos, mastelio keitimo, matricos) ir masyvas reikšmių.

**Animation**

Šiame šablone saugomi konkretaus kadro animacijos raktai. Jame turi būti bent vienas `AnimationKey` šablonas, taip pat turi būti nuoroda į kadrą.

**AnimationSet**

`AnimationSet` savyje talpina `Animation` šablonus.

## 2.2. Atviros (emplicit) ir užslėptos(explicit) animacijos metodai

Animacijos algoritmai skirstomi į dvi pagrindines kategorijas: atvirą (explicit) ir užslėptą (implicit) metodą. Atviras metodas saugo animuotų viršūnių seką kiekvienam kadrui. Kai visa tai išsaugoma ir užrašoma, animacijos duomenų nuskaitymui gali būti naudojamos įvairios technikos, užtikrinančios vientisą animaciją. Atviroji (explicit) animacija lengvai suprantama, ją lengva realizuoti programiškai, be to naudojama paprasta matematika, nereikalaujanti didelių skaičiavimų. Antra vertus, animuojamų objektų viršūnių saugojimas naudoja labai daug atminties. Norėdami atkurti tikroviškus judesius, turime saugoti daug veikėjo pozų. Tipinė MD3 rinkmena (atviros animacijos formatas naudojamas *„Quake III“* žaidime) vienam veikėjui užima apie 10 MB.

Kita animacijos algoritmų kategorija – užslėpta (implicit) animacija. Šis metodas saugo aukštesnio lygio judesio apibūdinimą. Pavyzdžiui, skeletinės animacijos (skeletal animation) sistemos saugo informaciją apie kiekvieną sąnarį (sujungimą) mūsų virtualiame modelyje. Po to realiu laiku, pagal šiuos duomenis apskaičiuojama objekto animacija. Šiame metode naudojami gana sudėtingi trigonometriniai bei matriciniai skaičiavimai, kurie yra reiklūs kompiuterio pagrindiniam procesoriui (CPU). Bet atminties naudojimas sumažėja iki minimumo, kadangi reikia išsaugoti tik nedideles duomenų struktūras, norint perteikti judesius.

Atvira animacija buvo labai populiari ankstyvuosiuose trimatės grafikos (3D) žaidimuose, kadangi procesoriai nebuvo tokie galingi. Daug dažniau, atviros animacijos metodai buvo naudojami žaidimuose, kur vienu metu reikėdavo pavaizduoti grupę veikėjų. Norint naudoti uždaros animacijos metodus, reikėtų atlikti labai daug skaičiavimų. Vis dėlto uždaros animacijos metodai tampa vis populiaresni, nes žaidimuose naudojamos animacijos tampa vis sudėtingesnes, tikroviškesnės bei detalesnės.

Vienas iš uždaros (implicit) animacijos metodo privalumų – prisitaikymas prie scenarijaus. Pavyzdžiui, veikėjas iš uždarų patalpų išeina į kalvotą vietovę. Jei naudosime atviros animacijos metodus, tai mes turime būti numatę, kaip veikėjas turi eiti kalvotoje vietovėje. Naudodami uždaros animacijos metodą mes lengvai galime modifikuoti veikėjo animaciją realiu laiku, kad ji prisitaikytų prie netaisyklingo paviršiaus.

## 2.3. Atviros animacijos metodika

### 2.3.1. Kadrų (frame) animacija

Pati paprasčiausia veikėjo animacijos technika kilusi iš tradicinės animacijos metodikos. Senosios mokyklos animacijoje, veikėjo kadrai buvo piešiami ant celofaninių lapų, kiekviena veikėjo poza buvo nupiešiama ant atskiro lapo. Kadrų animacijos principas labai panašus. Kiekviena veikėjo poza išsaugoma atskirai ir po to rodoma tam tikru tempu (dažniausiai 25 kadrai per sekundę).

Ką turime išsaugoti kiekviename kadre? Pirmiausia turime išsaugoti viršūnių koordinates. Saugome tik vieną tekstūrų koordinatės kopiją, nes ji nekis per visą animaciją. Taip pat nėra būtina kiekvienam kadrui saugoti viršūnių spalvų, normalių naudojamų apšvietimui skaičiuoti. Tai labai mažina užimamos atminties kiekį.

Netgi su šiais patobulinimais, kadrų animacijai reikia labai daug atminties. Šis grubios jėgos (brute-force) požiūris gali būti rekomenduojamas naudoti tik labai specifinėse situacijose, kai tereikia tik kelių kadrų, kurie nepasižymėtų dideliu trikampių skaičiumi.

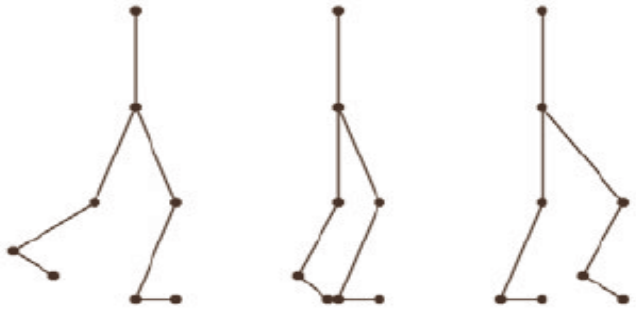
Kadrų animacija turi dar vieną minusą. Kadangi žaidimuose kadrų skaičius visada kinta, tai priklauso nuo scenos sudėtingumo, tai reikia užtikrinti, kad atskirų objektų animacijos nesiskirtų. Pavyzdžiui, veikėjas juda daug lėčiau nei kitos aplinkoje naudojamos animacijos. Kadrų animacijos problema ta, kad mes turime viršūnių tinklus (meshes) apskaičiuotus diskrečiu laiko momentu. Mes galime pasirinkti reikšmę, kuri yra mažiausiai nutolusi nuo reikšmės duotu laiko momentu, bet tokiu atveju visada matysime animacijos trūkčiojimą.

Norėdami išlaikyti tolygią animaciją, turime tiksliai apskaičiuoti tarpinius kadrus, kad galėtume pavaizduoti gerą priartėjimą prie pradinės animacijos su norimu tikslumu. Tai mums leidžia atlikti interpoliacija, kuri naudoja matematines funkcijas, leidžiančias apskaičiuoti norimas reikšmes.

### 2.3.2. Raktinių (keyframe) kadrų animacija

Kita panaši animacijos rūšis – raktinių kadrų animacija. Norint geriau suprasti šią animaciją, reikia panagrinėti, kaip animacija kuriama tokiomis programomis, kaip *„3D Studio Max“*, *„Maya“* arba *„XSI“*. Animatoriai objekto animaciją dažniausia kuria nurodydami raktinius kadrus, kai kiekvienas toks kadras apibūdina naują objekto poziciją erdvėje. Todėl

nereikia saugoti visų kadrų, užtenka išsaugoti tik raktinius kadrus. Animacijos varikliukas rūpinsis tarpinių kadrų suskaičiavimu.



**1. pav. Animacijos seka**


**2. pav. Interpoliuota pozicija**

MD2 rinkmenos formatas pirmą kartą buvo panaudotas *„Quake II“* žaidime. Šis formatas naudojo raktinių kadrų animaciją (keyframe). Šiame formate saugoma kiekviena objekto pozicija. Iš tikrųjų saugomos visos animuojamo objekto viršūnės. Taigi jei modelį sudaro 500 viršūnių, tai jos transformuotos ir būdavo saugomos kiekviename raktiniame kadre.

Yra keletas raktinių kadrų animacijos schemų. Pirma, dažniausiame variante, trūkstamus kadrus skaičiuojame naudodamiesi tiesine interpoliacija. Tiesinės interpoliacijos skaičiavimui gali būti naudojama ši formulė:

```
Interpolator=(timevalue-lastkeyframe)/ (nextkeyframe-lastkeyframe);
Interpolated_value=lastvalue*(1-Interpolator) + nextvalue*Interpolator;
```

`timevalue` – laiko reikšmė;

`lastkeyframe` – paskutinis raktinis kadras;

`nextkeyframe` – kitas raktinis kadras;

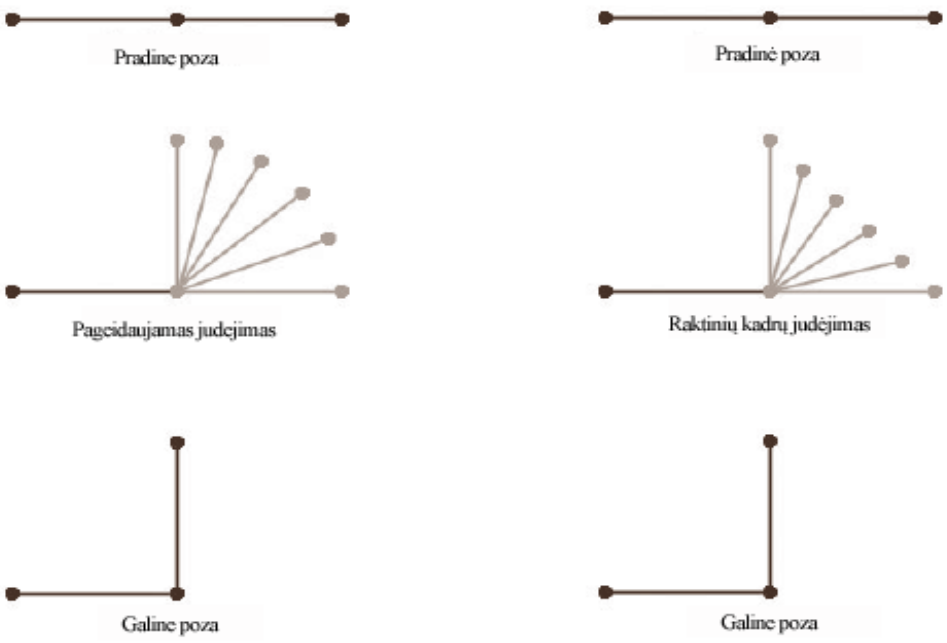
*lastvalue* – paskutinė reikšmė;

*nextvalue* – kita reikšmė.

Funkcijos algoritmas realizuojantis tiesinę interpoliaciją:

```
funkcion Interpolate()
{
  float alpha=(currenttime-time1)/(time2-time1);
  float alphainv=1-alpha;
  point res;
  res.x=p1.x*alphainv + p2.x*alpha;
  res.y=p1.y*alphainv + p2.y*alpha;
  res.z=p1.z*alphainv + p2.z*alpha;
}
```

Tiesinė interpoliacija užtikrina tolygų judėjimą ir mažas procesoriaus sąnaudas. Nepaisant to tiesinė interpoliacija gali nepadėti pasiekti norimų rezultatų. Dauguma animacijos programų naudoja galingus interpoliatorius (Bezier, Hermite ir t.t.) ir, kai pasiekiamas galutinis rezultatas, kai kurie judesiai atrodo nenatūralūs.




**3. pav. Raktinių kadrų animacija**

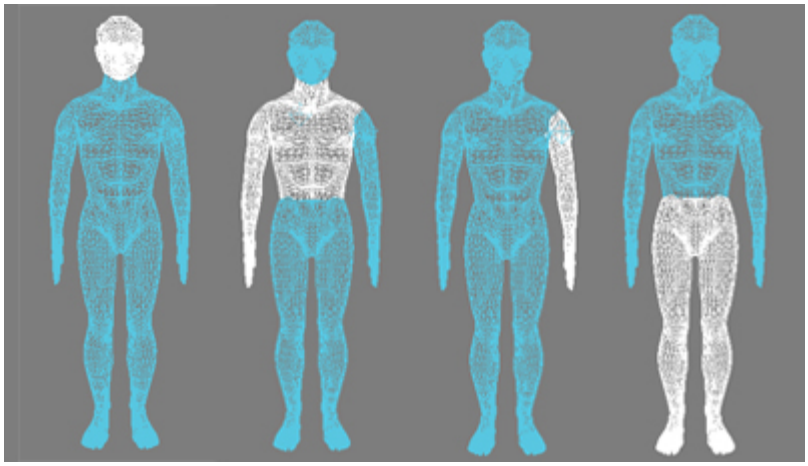
Vienas iš būdų gerinti kokybę – įsitikinti ar raktiniai kadrai nėra per toli vienas nuo kito, taip mažinami blokiniai judesiai. Matematinė lema teigia, kad bet kokia begalinė kreivė gali būti sutraukta į tiesią liniją, jei intervalas pakankamai mažas. Bandymai ir klaidos šioje situacijoje yra bene geriausias pasirinkimas, kadangi greiti judesiai reikalauja didesnio dažnio.

Dar geresnis būdas, užtikrinantis animacijos kokybę, tai automatiškai išgauti raktinius kadrus iš didesnės rezoliucijos animacijos sekos, kadangi pasirinkti raktiniai kadrai užtikrina, kad išlaikomas fiksuotas kokybės lygis, be to, minimizuojamas blokinis judėjimas. Programavimo esmė gana paprasta: animacija išsaugojama aukštesniame vaizdavimo dažnyje ir tada, pasinaudojus analizatoriumi, surandami kadrai, kuriuos būtų galima pašalinti iš animacijos beveik be jokių pašalinių rezultatų galutinei animacijai. Tai galima atlikti iteracijos būdu, kiekvieną kartą pašalinant mažiausiai reikšmingą kadrą, tol kol pasiekiamas užsibrėžtas lygis.

### 2.3.3. Žyminė (tagged) interpoliacija

Abu anksčiau aptartus animacijos metodus lengva programuoti, bet vis dėlto programuotojui kyla kai kurios problemos. Kaip jau minėta anksčiau, nemažas atminties naudojimas gali tapti rimta problema, jei mes norime animuoti objektus, turinčius didelį skaičių trikampių, arba norime išsaugoti daug raktinių kadrų.

Pavyzdžiui, kuriame veikėją veiksmo žaidimui. Veikėjas turi atlikti įvairius veiksmus (eiti, ramiai stovėti, bėgti, šokinėti ir t.t.), kurių atlikimas vaizduojamas animacija.
Viskas lyg ir būtų gerai, bet sakykime, kad veikėjas vienu metu atlieka kelis veiksmus. *„Quake II“* žaidimo variklyje, ši problema gali būti pastebėta daugelio vartotojų rėžime. Norint sumažinti animacijos resursus, tokie sudėtingesni veiksmai paprasčiausiai nebuvo numatyti. *„Quake III“* kūrėjų komanda rado sprendimą šiai problemai, kurią jie pavadino žymine (tagged) animacija ir realizavo ją MD3 animacijos sistemoje. Pagrindinė šio sprendimo idėja, veikėjo kūną padalinti į atskiras dalis: galvą, liemenį, kojas, rankas (1 paveikslas).



**4. pav. Veikėjo suskaidymas į atskiras dalis**

Kiekviena iš šių dalių turėjo savo animaciją. Taigi sudėtingesnės animacijos buvo pasiekiamos sujungiant skirtingas veikėjo dalių animacijas.

Kai suskirstėme veikėjo kūną į atskiras dalis, dabar reikia jas visas sujungti į visumą, kad visa tai atrodytų kaip vientisas veikėjas. Norint tai padaryti, reikia, kad veikėjo modeliuotojas rankiniu būdu nurodytų kiekvienos kūno dalies atsparos tašką. Šie taškai vadinami žymėmis (tags). Dabar mes turime tris kūno dalis ir keturias žymes (tags). Žymės nurodo kiekvienos kūno dalies padėtį erdvėje. Pirmoji žymė naudojama pažymėti pagrindo lygį, kad veikėjas tvirtai stovėtų ant paviršiaus. Ji dažniausiai priskiriama kojoms. Antroji žymė rodo kojų bei liemens susijungimo vietą. Trečioji žymė susieja liemenį ir galvą. Ketvirtoji žymė dažniausiai yra prie dešinės rankos, kad veikėjas galėtų laikyti ginklą, kuris gali būti keičiamas.

Kyla klausimas, kaip išlaikyti tolydų perėjimą tarp kojų ir liemens, tarp liemens ir galvos? Mes turėtume sulieti abi kūno dalis kartu, norėdami užtikrinti idealų sujungimą. *„Quake III“* animacijos sistemoje kūno dalis nėra suliejamos, jos tiesiog uždedamos viena ant kitos panašiai kaip plytos. Kad žaidėjai to nepastebėtų, kūno dalys truputi įsiskverbia viena į kitą, taigi visas veikėjas visada išlieka vientisas. Platūs veikėjų diržai ir tekstūros padeda perteikti vientisumo jausmą.

Žymių sistemos gali būti praplėstos ir jų pagalba galima pasiekti gana neblogų rezultatų. Pavyzdžiui, jums gali prireikti susieti veikėją su kokia nors transporto priemone. Viskas, ką reikia padaryti, tai sužymėti transporto priemonę, kad animacijos varikliukas žinotų kur patalpinti kiekvieną dalį erdvėje.

Žymių sistemos naudoja pastebimai mažiau atminties, negu įprasta raktinių kadrų sistema, ypač tai atvejais, kai norime atlikti skirtingus veiksmus su skirtingomis kūno dalimis, vienu metu.

Akivaizdu, kad atminties naudojimas augs tiesiškai, priklausomai nuo viršūnių skaičiaus. Procesoriaus (CPU) naudojimas daugmaž išliks stabilus ir panašus į tą, kuris buvo naudojant įprastą raktinių kadrų sistemą, kadangi mes interpoliuojame raktinius kadrus ir patalpiname geometrinius blokus erdvėje. Jei grafinė plokštė paliko aparatūrines transformacijas, tai kiekvienos dalies patalpinimas erdvėje beveik nereikalauja procesoriaus laiko.

Nepaisant visų minėtų privalumų, tokia animacijos sistema turi ir savų trūkumų. Pirmiausia bus ribojamas veikėjo detalumas. Taigi jei mums reikės pavaizduoti daug skirtingų veikėjų, naudojančių tas pačias animacijas, mes turėsime turėti labai daug šių animacijų kopijų ir vėliau ar anksčiau pritrūksime atminties.

Antra, žymių animacijos varikliukas nesuteiks mums prisitaikymo prie aplinkos pakitimo. Pavyzdžiui, ėjimas nelygiu paviršiumi, toliau esančių daiktų siekimas ir t.t., su aptartomis animacijos sistemomis yra beveik neįmanomas.

Taigi, kaip matome, žymių animacijos sistema, naudojama protingai gali būti daug naudingesnė už raktinių kadrų animaciją, dažnai sumažinanti atminties naudojimą dvigubai arba netgi daugiau. Veikėjo suskaidymas į daugiau dalių nei reikia, gali tik pabloginti dalykus. Realizuojant žymių animacijos sistemą, mums tereikia išsaugoti animacijos rodykles (pointers) kiekvienai kūno daliai, kad galėtume sekti, kuri animacija ir kuris kadras yra vaizduojamas. Tai atlikti padeda geometrinis objektas, kuris žymi atraminį tašką ir orientaciją. Dažniausiai pasirenkamas trikampis, kurio viena viršūnė naudojama atraminio taško padėčiai ir tada du vektoriai nuo šio taško į kitus du nurodo koordinačių ašis. Atliekant sandaugos veiksmą su šiais vektoriais gaunamas vektorius parodantis trečiąją koordinačių sistemos ašį.

## 2.4. Uždaros animacijos metodika

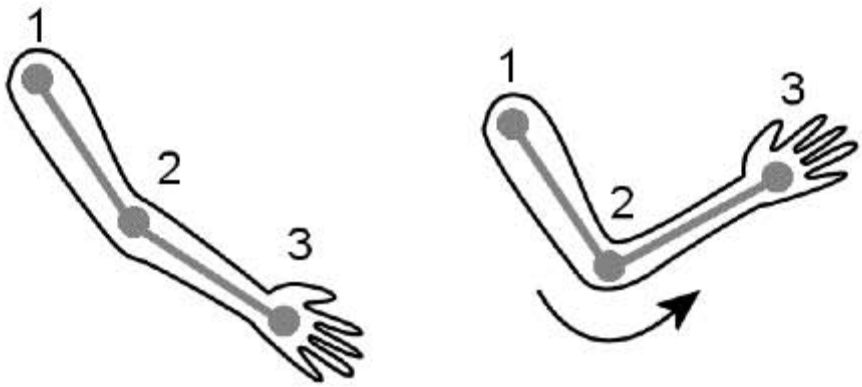
Uždara animacija turi nemažai privalumų, lyginant su atvira animacija. Pirmiausia tai labai sumažinamas atminties sunaudojimas, nes yra išsaugoma tik informacija apie sąnario konfigūraciją, o tai yra daug efektyviau nei saugoti visą viršūnių tinklą, kiekviename raktiniame kadre. Antra, naudojant tokias sistemas galima pritaikyti įvairių efektų, tokių kaip fiziškai teisinga animacija, prisitaikymas prie nelygios vietovės bei realios sąveikos tarp veikėjų. Ir galiausiai uždaros animacijos sistemos gali dalintis animacijos duomenimis tarp skirtingų objektų. Pavyzdžiui, jie jūsų žaidime yra septyni skirtingi veikėjai, tai informacija apie sąnarių padėtį ėjimo ciklo metu užtenka išsaugoti vieną kartą, kadangi viršūnių padėtis bus apskaičiuojama realiu laiku.

Pagrindiniai uždaros animacijos sistemų trūkumai: realizavimo sudėtingumas ir didelės skaičiavimo sąnaudos. Programuoti skeletinės animacijos sistemą yra daug sunkiau, nei išsaugoti atviros animacijos sistemos duomenis. Jei jūs planuosite panaudoti IK arba fiziškai teisingas animacijos sistemas, lygtys nėra labai paprastos, taigi animacijos posistemės taip pat reikalaus daugiau procesoriaus laiko.

### 2.4.1. Skeletinė animacija

Skeletinės animacijos principas remiasi tuo, kad kaulai yra sujungti sąnariais, kuriuos judina raumenys ir sausgyslės. Skeletinėje animacijoje sąnariai, jungiantys kaulus, sudaro tam tikrą hierarchiją. Veikėjo modelis yra sudaromas iš keleto elementų:

- Sąnarių (joints) – dažniausiai pateikiami pasukimo arba perkėlimo transformacijų matrica.
- Kaulai (bones) – segmentai jungiantys sąnarius. Kaulas dažniausiai identifikuojamas sąnariu, kaulo pradžioje.
- Skeletas (skeleton) – medžio viršūnės reprezentuoja sąnarius, o briaunos – kaulus.
- Oda (skin) – trikampių tinklas. Kiekvieno trikampio viršūnės susietos tik su vienu kaulu.




**5. pav. Skeletinės animacijos sistema**

Pateiktame 2 paveikslėlyje pagrindinis sąnarys pažymėtas 1 numeriu. Žemesniame hierarchijos lygmenyje yra alkūnės sąnarys, pažymėtas 2 numeriu ir pačiame žemiausiame lygmenyje – rankos sąnarys pažymėtas 3 numeriu. Pavertimas arba pasukimas visada veikia visus tolesnius sąnarius esančius žemesniuose hierarchijos lygmenyse. Taigi jei mes pasukame rankos sąnarį, likę sąnariai nėra paveikiami, nes ranka yra žemiausiame hierarchijos lygmenyje. Pasukus alkūnės sąnarį, ši transformacija bus pritaikyta visiems tolesniems sąnariams.

Pagrindinė skeletinės animacijos taisyklė: sąnariai gali būti tik pasukami, nes bet koks jų pozicijos pakeitimas gali sukelti objekto deformacijas. Įvairūs pasukimai leidžia sumodeliuoti visas fiziškai įmanomas veikėjo pozas. Kaip jau minėta anksčiau, sąnariai ir juos jungiantys kaulai sudaro hierarchinę struktūrą. Kiekvienas sąnarys turi vieną tėvą ir gali turėti pasirenkamą skaičių vaikų.

Skeletinės animacijos atveju, mums nereikia saugoti objekto viršūnių kiekvienam raktiniam kadrui. Kiekvienai objekto pozai nusakyti mums tereikia žinoti kiekvieno sąnario, įeinančio į modelį, pasukimo kampą.

Sakykime turime žaidimo veikėjo trikampių tinklą (mesh), kai kiekviena viršūnė veikiama vieno ar kelių kaulų. Kiekvienas kaulas turi tris laisvės laipsnius. Tai galima optimizuoti sąnariams su mažiau laisvės laipsnių.

Algoritmas žingsniai turėtų būti tokie: pirmiausia apdorojame bei apskaičiuojame esamą kiekvienos viršūnės transformaciją. Skaičiavimus pradedame nuo pagrindinio (root) kaulo, judėdami toliau ir sujungdami transformacijas. Šis procesas yra rekursinis, kartojamas tol, kol bus apdoroti visi kaulai.

Pradedame nuo pagrindinio kaulo, toliau atliekame uždarą ciklą. Išorinis ciklas eina per kalus, kurie yra tiesioginiai palikuonys to kaulo, kurį mes parinkome kaip pagrindinį. Sakykime, jei pradėtume atlikinėti šiuos veiksmus nuo dubens, tai mums reikėtų pereiti tris kaulus: nugarkaulį, kairįjį bei dešinįjį šlaunikaulį. Tada atliekame veiksmus su viršūnėmis, apskaičiuojame ar kaulas, kurį mes nagrinėjame sąveikauja su šia viršūne. Tai atliekama pasinaudojant svorio funkcija. Svorio funkcijos reikšmės dažniausiai nurodomos tuo metu, kai modeliuojamas veikėjas. Tačiau jei neturite tokios funkcijos, tai ją galite pasirašyti patys, jums tereikia tik apskaičiuoti svorius remiantis viršūnės atstumu nuo kaulo.

Grįžtant prie pradinio algoritmo turime du ciklus. Išorinis apeina visus kaulus, o vidinis eina per viršūnes apskaičiuodamas kaulo – viršūnės svorius. Jei svoris yra mažesnis už slenkstinę reikšmę, tai mes tą viršūnę paliekame nepaveiktą nagrinėjamo kaulo, jei priešingai, tai viršūnei pritaikoma transformacija.

Norėdami tai atlikti pirmiausia turime perkelti ją į kaulo sukimosi (pivot) tašką. Prieš taikydami matematinius skaičiavimus turime įsitikinti, kad kaulas yra tinkamai centruotas. Transformacijos matricą galime įkelti į steką. Šiame etape mes pritaikome kaulo transformaciją viršūnei. Reikia atsiminti tai, kad kiti kaulai taip pat gali veikti šią viršūnę, taigi mes turime kaupti kiekvieno analizuoto kaulo rezultatus.

Tai rekursinis algoritmas, jei einamasis kaulas turi palikuonių, tai mes turime kviesti tą pačią funkciją. Kai pasiekiamos galinės viršūnės, visas jų tinklas bus paveiktas skirtingų kaulų.

Kitas būdas – tai pertvarkyti kodą, taip, kad būtų išsaugomos matricos vietoj viršūnių koordinačių. Taip galime išsaugoti atmintyje (cashe) rezultatus, kurie nesikeičia, pereinant nuo vieno kadro prie kito. Ši technika turi net du privalumus. Pirmiausia galime išsaugoti atmintyje dalines kaulų struktūras ir jų matricas, kūno dalims, kurios nesikeičia. Antra, mes galime pasinaudoti geresniu animacijos palaikymu vėlesnėse *„DirectX“* versijose. Pavyzdžiui, *„DirectX 9*“ versijoje galime pasinaudoti *SetTransform* sąsaja. Viršūnių svoriai yra perduodami lanksčiu viršūnių formatu (Flexible – Vertex – Format, FVF).

### 2.4.2. Galūnių pašalinimas

Galūnių nukirtimas (galimybė pašalinti galūnes animuotam veikėjui realiu laiku) gali būti labai lengvai pritaikyta bet kuriai skeletinės animacijos sistemai. Šis uždavinys gali būti išskaidyta šį dvi dalis: geometrijos pašalinimas (tuščių vietų užpildymas) bei animavimas.

Norėdami pašalinti veikėjo ranką ar kojį, tai lengvai galime atlikti nuspręsdami, kurie kaulai bus pašalinti ir ištrindami visas viršūnes, kurias pašalintas kaulas veikė. Pavyzdžiui, norime pašalinti visą ranką. Taigi, einame per visus kaulus ir apskaičiuojame viršūnių svorius. Tas viršūnes, kurių svoriai yra didesni už nurodytą slenkstinę reikšmę pažymime. Šis procesas taikomas ir kitiems kaulams esantiems hierarchijoje. Pseudo algoritmas turėtų atrodyti taip:

```
Void Amputate (bone root, mesh data)
      Su kiekvienu vaiku
            Su kiekviena viršūne, priklausančia
            pradiniam trikampių tinklui
                  Suskaičiuoti svorį
                  Jei svoris > slenkstinę reikšmę
                        Pažymėti viršūnę kaip ištrintą
            Jei kaulas turi vaikų
                        Amputate (bone, mesh, finalmesh)
```

Kai bus atliktas šis žingsnis, turėsime pradinį trikampių tinklą, su pažymėtomis viršūnėmis. Tereikia atnaujinti trikampių tinklą ištrinant pažymėtas viršūnes. Atsiradusią skyle reikia padengti trikampiais. Tai atliekame pasirinkdami panaikinto kaulo atramos tašką kaip pradinį ir atliekame žvaigždinį padalijimą į trikampius.

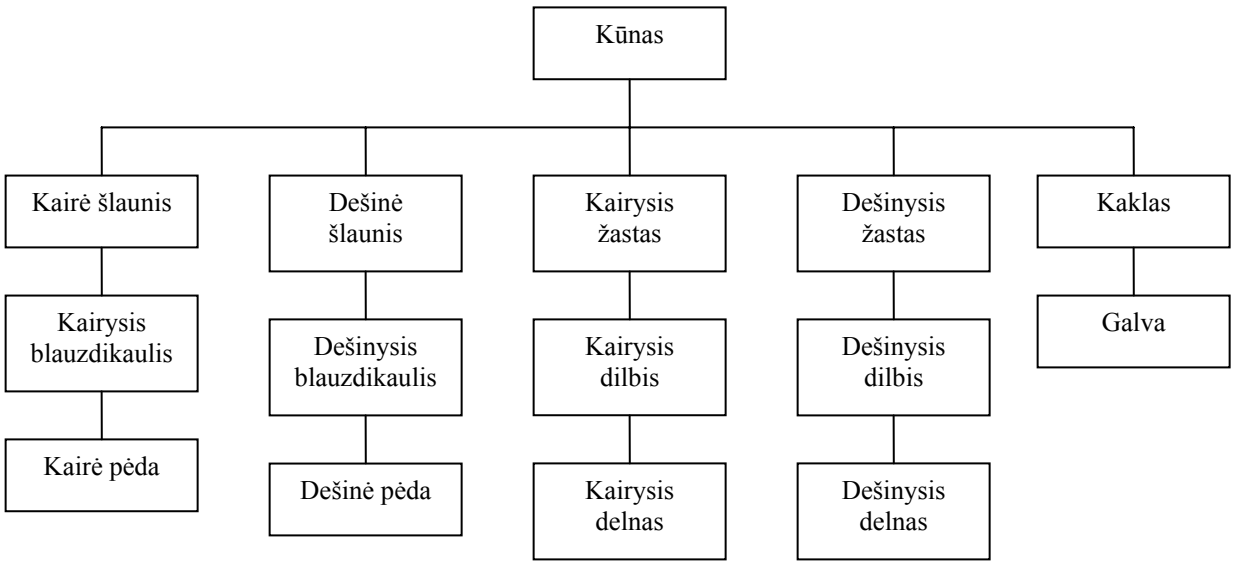
Animacijoje nebus didelių pakitimų. Pagrindinis pokytis tame, kad reikės pašalinti keletą viršūnių ir trikampių tinklo.

# 3. Skeletinės ir kadrinės animacijos architektūra ir realizacija

## 3.1. Skeletinė animacija

Skeletinė animacija dažniausiai naudojama organinių modelių animacijai. Ji remiasi idėja, kad objektams formą suteikia daug kaulų, kurie gali būti paprastai animuojami pakeičiant tik kaulo pozicija ir orientaciją.

Paviršiaus viršūnių tinklas (skin mesh) – hierarchinės scenos tipas. Šios scenos naudojamos sujungti objektus tarpusavyje. Pavyzdžiui, pirštas sujungtas delnu, kuris savo ruožtu jungiasi su dilbiu ir taip toliau. Kiekvienos objekto koordinatės pateikiamos santykinai (relative) tėvinei erdvei, vadinasi dilbiui atliekant kokį nors judesį delnas ir pirštas taip pat atliks judesį. Šiame paveikslėlyje matome, kaip galime sudaryti žmogaus kūno modelį naudodamiesi hierarchinių scenų metodu.




**6. pav. Žmogaus kūno modelis**

Turėti veikėją sudaryta iš atskirų fiksuotų objektų nėra patogu, nes tai sukelia įtrūkimus ties sąnariais. Norint išvengti šios problemos buvo pradėti naudoti paviršiaus trikampių tinklus (skin meshes). Paviršiaus trikampių tinklai turi ta pačią kaulų struktūrą, pagrindinis skirtumas tame, kad viena paviršiaus dalis gali būti transformuojama kelių kaulų. Taigi viršūnių poziciją taip pat gali veikti vienas ar keli kaulai. Ši technika – vadinama viršūnių suliejimu (vertex

blending). Viršūnių suliejimo metodas nurodo, kad kiekviena viršūnė turėtų suliejimo (blending) svorį, kad programa kuri apdoroja viršūnę galėtų nustatyti kaip kaulai veikia nagrinėjamą viršūnę. Pavyzdžiui, jei viršūnę paveiksime keliais kaulais, tai paveiktos viršūnės pasaulio koordinates apskaičiuosime taip:

```
Vw = Vm * M1 * w + Vm * M2 * (1 - w).
```

Čia `Vw` – viršūnės pasaulio koordinatės,

`Vm` – pozicija lokalioje erdvėje,

`M1, M2` – transformacijų matricos,

`w` – perėjimo svoris.

Nuskaitant X failą, reikės užkrauti kadrus ir sukonstruoti hierarchinę sceną. Taip pat reikės susieti paviršiaus trikampių tinklą su kaulais. X failo sukūrimas modeliuotojo darbas, dažniausiai atliekamas kokia nors komercine ar atviro kodo modeliavimo programa („3D Studio Max“, „Maya“, „Blender“ ir pan.).

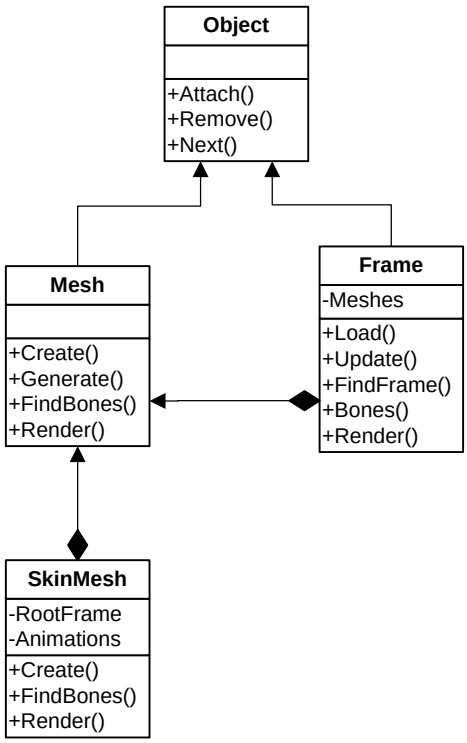
Norėdami atlikti veiksmus su X failu, turėsime naudoti biblioteką skirtą X failų apdorojimui. `IDirectXFile` – pagrindinė bibliotekos sąsaja (interface). `IDirectXFileEnumObject` objekto metodai naudojami išrinkti duomenis iš konkretaus X failo. Metodas `GetNextDataObject` eina per visus aukščiausio lygio šablonus, norėdami išgauti informacija iš vidinių šablonų naudojame `GetNextObject` metodą. Prieš nuskaitydami šablono duomenis turime sužinoti jo tipą, tam naudojamas `GetID` metodas, o duomenų nuskaitymui `GetData` metodas.

Trikampių tinklo užkrovimui naudosime `D3DXLoadSkinMeshFromXof` funkciją, kuri gražins rodyklę į `ID3DXSkinMesh` objektą. Šiame objekte saugoma paviršiaus trikampių tinklo informacija. Ši informacija saugoma grupėmis, taigi kiekviena grupė gali būti transformuojama skirtingų kaulų rinkinių. Funkcija gražins masyvą medžiagų (materials), kurias naudoja paviršiaus trikampių tinklas. Taip pat bus gražinti du buferiai: pirmajame bus saugomi visų kaulų pavadinimai, o antrajame saugomos jų transformacijos. Šios transformacijos – tai kaulų poslinkiai (bone offset). Vadinasi norėdami, kad kaulas paveiktų trikampių tinklą, turime jį deformuoti skirtumu tarp kaulo dabartinės transformacijos ir kaulo pradinės transformacijos. Kitaip sakant turime transformuoti viršūnes į kaulų lokalią erdvę ir tada jas transformuoti atgal naudojant naują kaulo transformaciją. Pavyzdžiui, sakykime, kad turime kaulą kurio pozicija (0,50,0) o viršūnės pozicija (0,51,0). Kaulas veikia tik šią viršūnę. Jei mes kaulą perkelsime į

poziciją (0,51,0), tai viršūnės pozicija turėtų būti (0,52,0), bet jei mes tik sudaugintume, gautume, kad viršūnės pozicija (0,102,0). Taigi norėdami šito išvengti naudosime kaulo poslinkio (offset) matricą transformuoti viršūnę iš pradinės pozicijos į poziciją reliatyvią kaulo pozicijai. Gauta pozicija bus (0,1,0), kuri bus kaulų matricos pagalba transformuota į poziciją (0,52,0).

`ID3DXSkinMesh` objektas saugo tik duomenis, norėdami atvaizduoti objektą pirmiausia jį turėsime konvertuoti į *`ID3DXMesh`* objektą. Tai atliks funkcija `ConvertToBlendedMesh`. Kaip jau minėjau anksčiau trikampių tinklas padalintas į grupes. Kiekviena grupė turi būti atvaizduota su jai priklausančia medžiaga (material) ir konkrečiu kaulu rinkiniu.

Kaip matome 7 paveikslėlyje, `Mesh` ir `Frame` klasės išvestos (derived) iš klasės `Object`. `Object` tikslas suteikti objektams galimybę būti susietiems medžiu. `Frame` – scenos hierarchijos pagrindinis konstrukcinis elementas, `Mesh` saugo informaciją apie trikampių tinklą. Pradinis scenos kadras saugomas `SkinMesh`. Visos operacijos susijusios su paviršiaus trikampių tinklu bus inicijuotos `SkinMesh` klasėje.




**7. pav. „SkinAnim“ klasių diagrama**

Šis algoritmas aprašo kaip scena sukuriama iš X failo.

```
SkinMesh::Create()
Begin
      Inicializuoti X failo sąsaja
```

```
      Atidaryti X failą
      Su kiekvienu aukštesnio lygio šablonu atlikti šiuos veiksmus
      Begin
            Išrinkti X failo duomenų objektą
            Perduoti duomenų objektą RootFrame.Load
      End
      Susieti kaulus su paviršiaus viršūnių tinklu
End

Frame::Load()
Begin
      Patikrinti duomenų objekto tipą
      Jei tipas yra Mesh
      Begin
            Sukurti naują Mesh objektą
            Prikabinti naują objektą prie kadro
            Perduoti duomenų objektą Mesh::Create
      End
      Kitu atveju jei tipas yra FrameTransformationMatrix
            Užkrauti transformacijų matricą
      Kitu atveju jei tipas yra Frame
      Begin
            Sukurti naują Frame objektą
            Prikabinti naują objektą prie kadro
            Suteikti vaiko kadrui (child frame) šablono pavadinimą
            Su kiekvienu vaiko šablonu (child template)
            Begin
                  Išrinkti X failo duomenų objektą
                  Perduoti jį NewFrame.Load
             End
       End
End

Mesh::Create()
Begin
      Objektą pavadinti šablono pavadinimu
```

```
      Užkrauti paviršiaus trikampių tinklą
      Sugeneruoti pereinamą (blended) trikampių tinklą iš paviršiaus
      trikampių tinklo objekto
      Užkrauti medžiagas (materials)
End
```

Kai susikūrėme paviršiaus trikampių tinklą, dabar visą tai galime atvaizduoti (render). Atvaizdavimo veiksmas susideda iš dviejų etapų. Pirmojo etapo metu apskaičiuojama visų kaulų pasaulio matrica (world matrix) ir išsaugoma `Mesh` objekte. Antrojo etapo metu bus atvaizduotas pats paviršiaus trikampių tinklas. Algoritmas aprašantis šią operaciją.

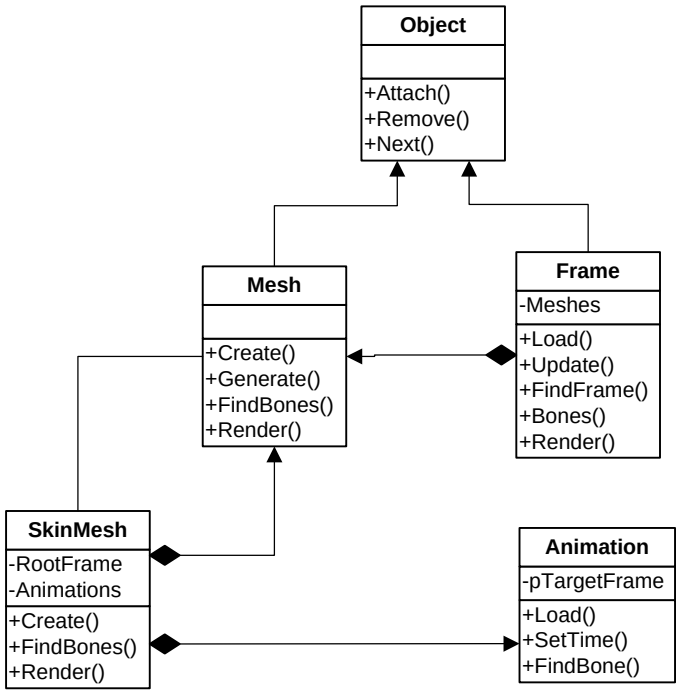
```
SkinMesh::Render()
Begin
      Įgalinti viršūnių suliejimą (blending)
      Su kiekvienu paviršiaus trikampių tinklo poaibiu
      Begin
            Nustatyti kaulų transformacijos matricas
            Nustatyti medžiagas (materials)
            Atvaizduoti
      End
      Išjungti viršūnių suliejimą
End
```

## 3.2. Skeletinė animacija X rinkmenose

Paviršiaus trikampių tinklo animacijos išsaugojimas X rinkmenoje nėra kuo nors išsiskiriantis. X tipo rinkmena saugo raktinius kadrus, programa pati sugeneruoja tarpinius kadrus, naudodama tiesinę interpoliaciją. Rinkmenoje saugomi taip vadinami animacijos raktai: pasukimo, mastelio keitimo, pozicijos ir matricos. Pasukimas aprašomas kvaternionais ir gali būti interpoliuojama naudojant sferinę tiesinę interpoliaciją. Funkcija `D3DXQuaternionSlerp` realizuoja šią interpoliaciją.

## 3.3. Skeletinės animacijos realizavimas

Norėdami realizuoti paviršiaus trikampių tinklų animaciją turėsime pridėti naują klasę. Ši klasė saugos informaciją susijusią su animacija, bei rodyklę į kadrą. Klasėje taip pat bus `SetTime` funkcija, kuri atnaujins kadro transformacijos matricą. Taigi atnaujinta klasių diagrama atrodys taip:




**8. pav. „SkinAnim“ klasių diagrama**

```
SkinMesh::Create()
Begin
      Inicializuoti X failo sąsaja
      Atidaryti X failą
      Su kiekvienu aukštesnio lygio šablonu atlikti šiuos veiksmus
      Begin
            Išrinkti X failo duomenų objektą
            Perduoti duomenų objektą RootFrame.Load
      End
      Susieti kaulus su paviršiaus viršūnių tinklu
      Susieti kaulus su animacijomis
```

```
End

Frame::Load()
Begin
      Patikrinti duomenų objekto tipą
      Jei tipas yra Mesh
      Begin
            Sukurti naują Mesh objektą
            Prikabinti naują objektą prie kadro
            Perduoti duomenų objektą Mesh::Create
      End
      Kitu atveju jei tipas yra FrameTransformationMatrix
            Užkrauti transformacijų matricą
      Kitu atveju jei tipas yra Frame
      Kitu atveju jei tipas Animation
            Liepti SkinMesh užkrauti naują animaciją
      Begin
            Sukurti naują Frame objektą
            Prikabinti naują objektą prie kadro
            Suteikti vaiko kadrui (child frame) šablono pavadinimą
            Su kiekvienu vaiko šablonu (child template)
            Begin
                  Išrinkti X failo duomenų objektą
                  Perduoti jį NewFrame.Load
             End
       End
End

SkinMesh::LoadAnimation()
Begin
      Sukurti naują Animation objektą
      Prikabinti objektą prie medžio
      Su kiekvienu vaiko šablonu (child template)
            Naujam objektui iškviesti Animation::Load
End
```

```
Animation::Load()
Begin
      Patikrinti duomenų objekto tipą
      Jei tipas – rodyklė
      Begin
            Pasiimti nurodytą šabloną
            Nuskaityti jo vardą
            Išsaugoti vardą
      End
      Kitu atveju jei tipas – duomenys
      Begin
            Patikrinti animacijos rakto tipą
            Nuskaityti atitinkamą raktą
      End
End
```

`SetTime` funkcijos algoritmas.

```
Animation::SetTime()
Begin
      Jei yra matricos raktas (matrix key)
      Beign
            Gauti artimiausią matricą duotu laiko momentu
            Perduoti ją einamam kadrui
      End
      Kitu atveju
      Begin
            Inicializuoti TransMat matricą
            Jei yra mastelio keitimo raktas
            Begin
                  Apskaičiuoti tikslias mastelio reikšmes
                  Paruošti mastelio matricą
                  Papildyti TransMat matricą
            End
            Jei yra pasukimo raktas
            Begin
```

```
                Apskaičiuoti tikslų pasukimo kvaternioną
                Paruošti pasukimo matricą
                Papildyti TransMat matricą
          End
          Jei yra pozicijos raktas
          Begin
                Apskaičiuoti tikslią pozicijos reikšmę
                Paruošti pozicijos matricą
                Papildyti TransMat matricą
          End
          Nustatyti TransMat einamam kadrui
     End
End
```

## 3.4. 3ds rinkmena

3ds rinkmena susideda iš duomenų blokų. Kiekvienas duomenų blokas prasideda vienodai:

- Nurodomas bloko ID,
- Nurodomas bloko ilgis.

Bloko ID (*chunk_id*) – unikalus kodas, kuris identifikuoja bloke esančių duomenų tipą, bei gali nurodyti ar yra šalutinių blokų. Blokas savyje gali turėti kitų blokų, todėl rinkmena palaiko hierarchinę struktūrą. Antroje lentelėje pateikiu sąrašą blokų, kurie buvo naudojami šiame darbe.

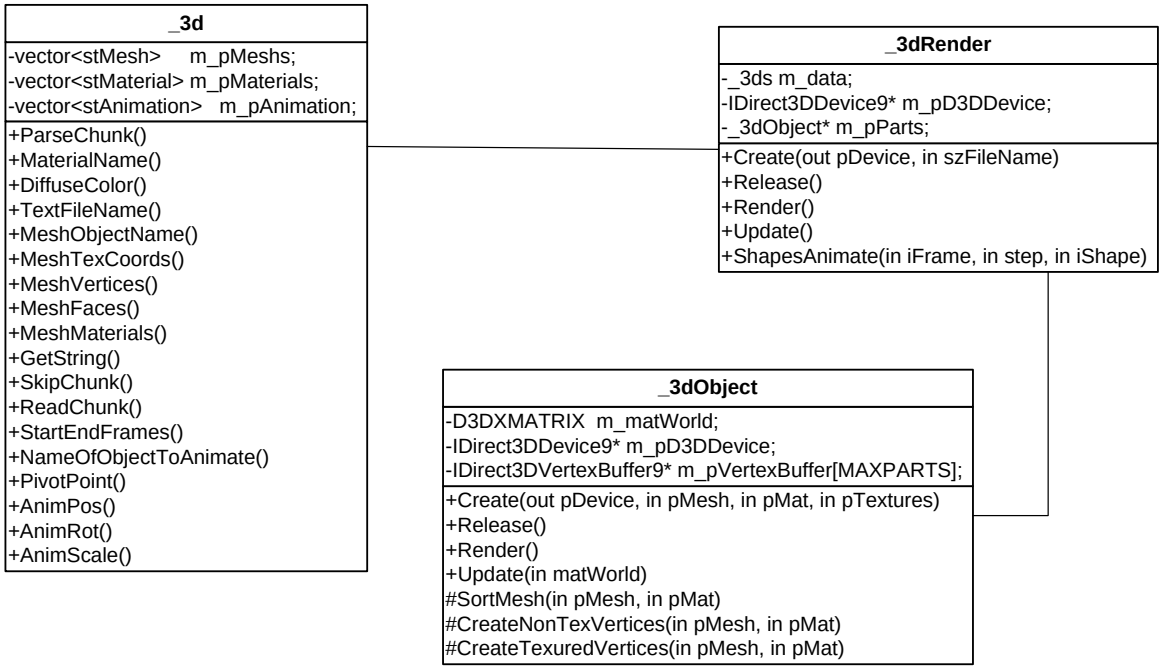
**2 lentelė. 3ds rinkmenos blokų aprašymai**

| Pavadinimas | Bloko ID | Aprašymas |
|---|---|---|
| **PRIMARY** | 4D4D | 3ds rinkmenos pradžia |
| **EDIT3DS** | 3D3D | Rinkmenoje esančių objektų, aprašymo pradžia |
| **KEYF3DS** | B000 | Informacija susijusi su animacija |
| **VERSION** | 0002 | Rinkmenos versija |
| **MESH_VERSION** | 3D3E | Trikampių tinklo versija |
| **KFVERSION** | 0005 | Animacijos versija |
| **COLOR_F** | 0010 | Spalvos formatas: float r, g, b; |
| **COLOR_24** | 0011 | Spalvos formatas: char r, g, b; |
| **LIN_COLOR_24** | 0012 | Spalvos formatas: char r, g, b; |
| **LIN_COLOR_F** | 0013 | Spalvos formatas: float r, g, b; |
| **MASTER_SCALE** | 0100 | Mastelis |
| **IMAGE_FILE** | 1100 | Nuoroda į failą |
| **AMBIENT_LIGHT** | 2100 | Taškinis šviesos šaltinis |
| **NAMED_OBJECT** | 4000 | Objekto pavadinimas |
| **OBJ_MESH** | 4100 | Objekto trikampių tinklas |
| **MESH_VERTICES** | 4110 | Trikampių tinklo viršūnės |
| **VERTEX_FLAGS** | 4111 | Viršūnių savybės |
| **MESH_FACES** | 4120 | |
| **MESH_MATER** | 4130 | Trikampių tinklo medžiaga |
| **MESH_TEX_VERT** | 4140 | Trikampių tinklo, tekstūrų viršūnės |

| Pavadinimas | Bloko ID | Aprašymas |
|---|---|---|
| **HEIRARCHY** | 4F00 | Hierarchija |
| **MATERIAL** | AFFF | Medžiaga |
| **MAT_NAME** | A000 | Medžiagos pavadinimas |
| **MAT_AMBIENT** | A010 | Medžiagos spalva |
| **MAT_DIFFUSE** | A020 | Išsklaidoma spalva |
| **MAT_SPECULAR** | A030 | Atspindima spalva |
| **MAT_SHININESS** | A040 | Ryškumas (blizgumas) |
| **MAT_EMISSIVE** | A080 | Išspinduliuojama spalva |
| **MAT_TEXMAP** | A200 | Medžiagos tekstūros koordinatės |
| **MAT_TEXFLNM** | A300 | Tekstūros failas |
| **OBJ_LIGHT** | 4600 | Šviesos objektas |
| **OBJ_CAMERA** | 4700 | Kameros objektas |
| **ANIM_HEADER** | B00A | Animacijos antraštė |
| **ANIM_OBJ** | B002 | Animacijos objektas |
| **ANIM_NAME** | B010 | Animacijos pavadinimas |
| **ANIM_POS** | B020 | Pozicija |
| **ANIM_ROT** | B021 | Pasukimas |
| **ANIM_SCALE** | B022 | Mastelis |

## 3.5. Raktinių kadrų metodo architektūra ir realizacija

Raktinių kadrų animacijai realizuoti buvo sukurta programa „KeyFrameAnim“, kurią sudaro trys klasės. Klasė `_3ds` atsakinga už 3ds rinkmenos nuskaitymą, `_3dObject` klasė visus rinkmenoje esančius modelius sudeda į atskirus objektus, o klasė `_3dRender` leidžia juos atvaizduoti kompiuterio ekrane.




**9. pav. „KeyFrameAnim“ klasių diagrama**

Kadrinė animacija realizuojama ShapesAnimate funkcija, kurios algoritmas pateikiamas žemiau.

```
Void ShapesAnimate ()
Begin
      Priskiriamas animuojamo objekto pavadinimas
      Einama per rinkmenoje esančių objektų sąrašą
            Jei pavadinimas sąraše sutampa su nurodytu pavadinimu
                  Tai pažymėti objektą kaip animuojamą
            Kitu atveju pažymėti kaip neanimuojamą
      Pozicijos animavimas
      Posūkio animavimas
      //Pastumti objektą į koordinačių pradžią ir tik tada atlikti
```

```
        //posūkio transformaciją, po to objektą grąžinti į
        //buvusią poziciją.
        Mastelio animavimas
        Visų gautų animacijų apjungimas
        Gautos transformacijos matricos pritaikymas, animuojamam
objektui
End
```

# 4. Eksperimentinis tyrimas

## 4.1. Sukurtos sistemos kokybės tyrimas

Vertinant produktą, buvo apklausti potencialūs vartotojai, taip įvertinant realizuotų metodų pilnumą. Vartotojas tyrimo metu užpildė lentelę, kurioje nurodė sistemos funkcionalumą. Išpildyti reikalavimai pažymėti pliusu.

**3 lentelė. Sistemos funkcionalumo tyrimas**

| Nr. | Funkcionalumas | Išpildymas |
|---|---|---|
| 1. | 3ds formato palaikymas | +/- |
| 2. | X formato palaikymas | + |
| 3. | Realizuota skeletinė animacija | + |
| 4. | Realizuota kadrinė animacija | + |
| 5. | Galimybė pačiam tobulinti produktą | + |

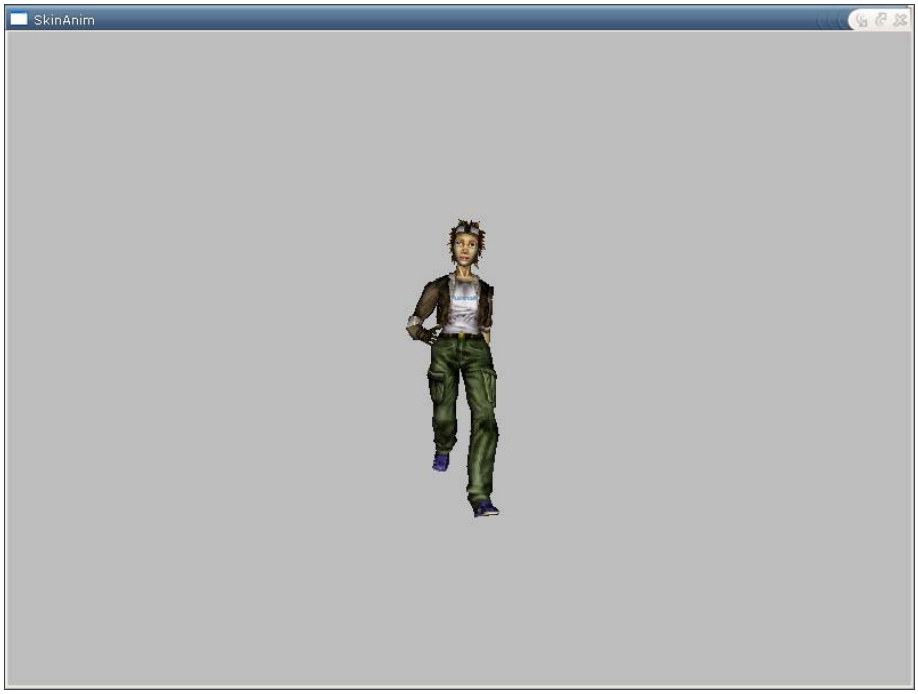
Tiriant produkto kokybę, buvo atsižvelgta į realizuotų metodų naudojimo paprastumą, bei galimybes juos tobulinti pačiam vartotojui. Tyrimo metu paaiškėjo, kad nepilnai buvo realizuotas 3ds failo nuskaitymas, trūko kai kurių objekto savybių. Šios savybės naudojamos išgauti papildomą realistiškumo efektą, gali būti gana lengvai realizuotos ir pačių vartotojų, šiek tiek geriau susipažinus su 3ds rinkmenos formatu.

# 5. Vartotojo vadovas

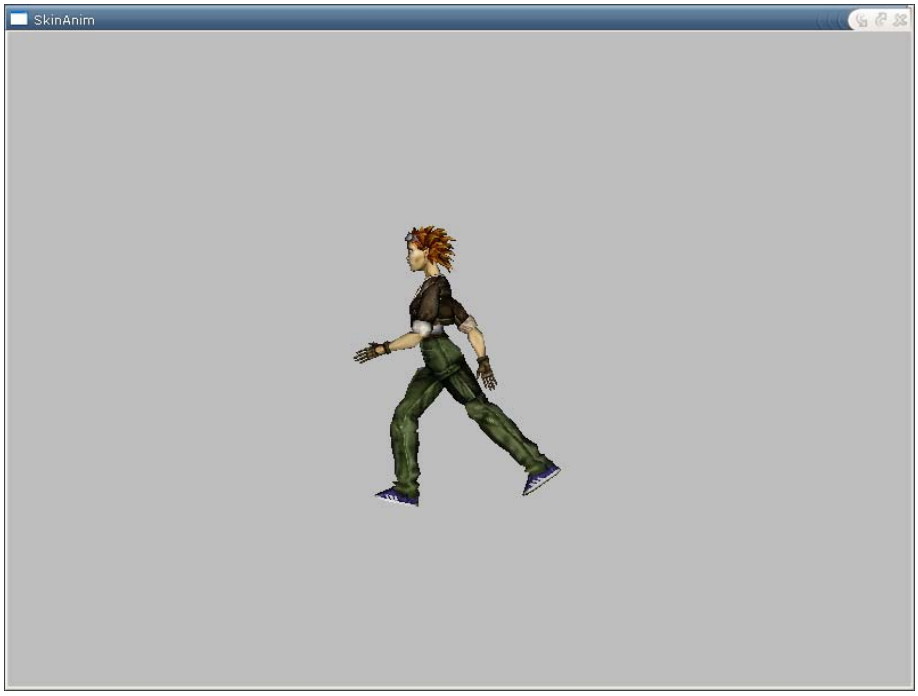
Realizuojant dažniausiai naudojamus animacijos metodus, buvo sukurtos dvi programos: „SkinAnim“ ir „KeyFrameAnim“. Norint paleisti šias programas reikia „Windows“ operacinės sistemos, „DirectX 9“, bei vaizdo plokštės palaikančios „DirectX 9“. Kūrime buvo naudojama „MS Visual Studio 2003“, „DirectX 9“. Rezultate buvo gautos realiu laiku veikiančios tolygios animacijos. Šiuos metodus galima panaudoti ir kitose programose, kadangi buvo sukurta biblioteka šiems metodams saugoti. Programuojant tiesiog tereikia įtraukti „Animation“ biblioteką. Norint naudotis skeletinė animacija reikia kreiptis į `Animation.Skin` metodą, o naudojantis kadrine animacija – į `Animation.KeyFrame` metodą.

Šios bibliotekos numatomas vartotojų ratas – programuotojai besidomintys kompiuterinių žaidimų kūrimų.

Žemiau pateiktose iliustracijose vaizduojamos „SkinAnim“ ir „KeyFrameAnim“ programos, kurios realizuoja aptartus animacijos metodus.

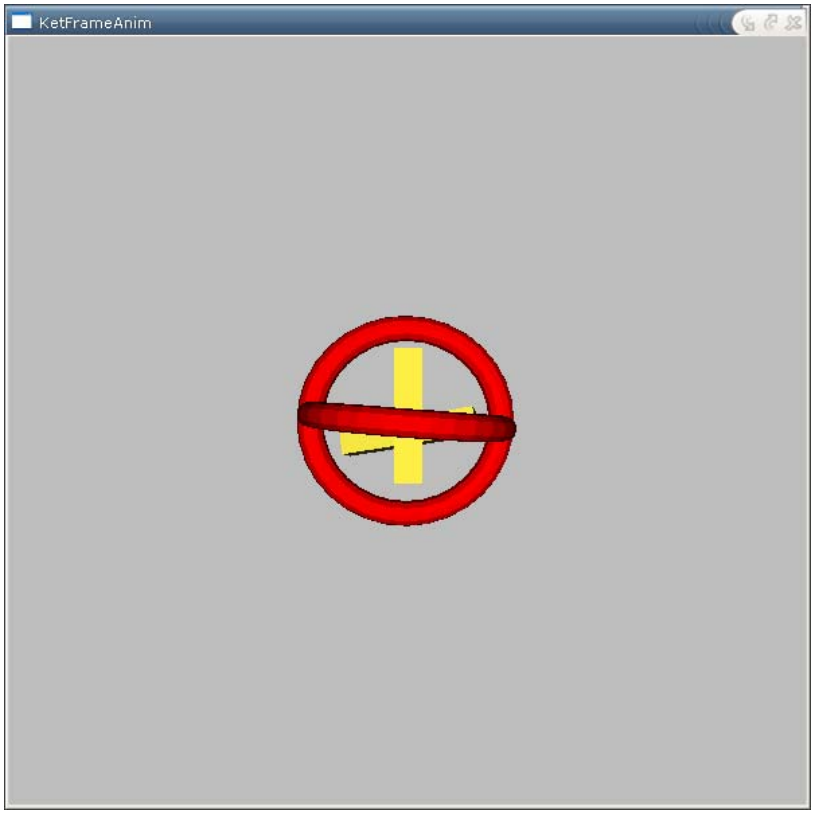



**10. pav. Skeletinės animacijos realizacija - „SkinAnim“**




**11. pav. Skeletinės animacijos realizacija - „SkinAnim“**


**12. pav. Skeletinės animacijos realizacija – „SkinAnim“**




**13. pav. Raktinių kadrų metodo realizacija – „KeyFrameAnim“**

# 6. Išvados

1. Išnagrinėti ir pateikti kompiuteriniuose žaidimuose naudojami animacijos metodai. Šie metodai buvo suskaidyti į dvi pagrindines grupes: atviros ir užslėptos animacijos metodus. Išanalizavus abiejų grupių metodus, nustatyta, kad iš visų animacijos metodų dažniausiai naudojami raktinių kadrų bei skeletinė animacijos metodai.
2. Nustatyta, kad skeletinei animacijai reikia mažiau atminties nei raktinių kadrų animacijai, nes nereikia saugoti kelių vieno modelio trikampių tinklo kopijų. Be to, skeletinę animaciją patartina naudoti norint animuoti organinius modelius.
3. Pastebėta, kad kuriant modelį reikia atkreipti dėmesį į objekto centrinio taško orientaciją pasaulio koordinačių atžvilgiu. Orientacijos neatitikimas gali sukelti ne mažų problemų realizacijos etape.
4. Realizuoti realiu laiku veikiantys kadrinės ir skeletinės animacijos algoritmai. Rezultatas, dvi parodomosios programos „SkinAnim“ ir „KeyFrameAnim“, vaizduojančios objektų animaciją realizuotą skirtingais metodais. Sukurta biblioteka, leidžianti panaudoti šiuos metodus kitose sistemose.
5. Aptarti rinkmenų tipai, naudojami animuojamų objektų saugojimui, bei perkėlimui iš vienos sistemos į kitą. Nustatyta, kad informacija rinkmenose saugoma tam tikrais blokais: X rinkmenų tipe – tai šablonai, 3ds rinkmenų atveju – informacijos blokai (data chunks).

# 7. Terminų ir santrumpų žodynėlis

**RUP** (*Rational Unified Process*) – programinės įrangos projektavimo modelis.

**MD3** – „Quake 3“ žaidimo veikėjų failo formatas.

**MD2** – „Quake 2“ žaidimo veikėjų failo formatas.

**X** – „DirectX“ failo formatas, saugoti žaidimo modeliams.

**3D** – trimatis.

**2D –** dvimatis

**CPU –** kompiuterio procesorius.

**Medžiagos** (*materials*) – nusako kaip atrodys atvaizduotas objekto paviršius.

**Skeletinė animacija** (skeletal animation) – animacijos sistema, kurios pagrindą sudaro skeleto sistema susieta su paviršiaus trikampių tinklu.

**Galūnių pašalinimas** (limb slicing) – objekto modifikavimo technika, pašalinti tam tikras objekto dalis.

# 8. Literatūra


1. Stefan Zerbst, Oliver Düvel. 3D Game Engine Programming, 2004
2. David H. Eberly. 3D Game Engine Design, 2001
3. Frank D. Luna. Introduction to 3D Game Programming with DirectX 9.0, 2003
4. Julio Sanches, Maria P. Canton. DirectX 3D Graphics Programming Bible, 2000
5. Bruno Miguel Teixeira do Sousa. Game Programming All in One, 2002
8. Cen Cebenoyan. Efficient Animation, GDC2001 prezentacija, 2001
9. Wendy Stahler. Beginning Math and Physics for Game Programmers, 2004
10. Kompiuterinių žaidimų kūrimo svetainė.
    Prieiga per internetą www.devmaster.net