Tallinna Ülikool

Informaatika Instituut

Polügonaalvõrgustikega manipuleerimine Java 3D abil

Bakalaureusetöö

Autor: Risto Seene

Juhendaja: Jaagup Kippar

Autor: ....."....." 2009

Juhendaja: ....."....." 2009

Instituudi direktor: ....."....." 2009

Tallinn 2009

Autorideklaratsioon

Kinnitan, et käesolev lõputöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud.

....

(kuupäev) (allkiri)

Sisukord

Sissejuhatus

Käesolev bakalaureusetöö on edasiarendus eelnevalt valminud seminaritööle „Java 3D redaktor“, mis on kättesaadav järgmistelt aadressidelt: http://www.cs.tlu.ee/instituut/opilaste_tood/seminari_ja_proseminari_tood/2008_sugis/seminaritood/risto_seene/risto_seene_seminaritoo.pdf ja http://www.cs.tlu.ee/instituut/opilaste_tood/seminari_ja_proseminari_tood/2008_sugis/seminaritood/risto_seene/risto_seene_lisa.zip. Töö eesmärgiks on uurida Java 3D eripärasid kolmemõõtmeliste objektide polügonaalvõrgustike haldamisel ja võimalusi dünaamiliste struktuuride loomiseks, mis võimaldaksid nende võrgustike andmetega efektiivselt manipuleerida.

Töö käigus selgitatakse välja polügonaalvõrgustikel põhinevate kolmemõõtmeliste objektide esitamise põhimõtted arvutigraafikas ning uuritakse kuidas käituvad neid objekte kirjeldavad Java 3D klassid. Vastavalt sellele saab luua omapoolseid vajalikke struktuure, mis oleksid võimelised suhtlema vastavate Java 3D klassidega ning manipuleerima neis hallatavate andmetega.

Töö lõpptulemusena lisandub seminaritöö käigus valminud rakendusele lisamoodul – geomeetria redaktor, mis võimaldaks arusaadava kasutajaliidese abil interaktiivselt ja võimalikult optimaalselt manipuleerida Java 3D struktuuride poolt hallatavate objektide geomeetriaga. Valmiva rakenduse eesmärgiks on lihtsustada Java 3D rakenduste tarbeks loodava geomeetria loomist, mis senimaani toimub üldjuhul programmeerides või välistes redaktorites loodu importimises ning seejärel programmeerides optimeerides.

1. Polügonaalvõrgustikud arvutigraafikas

Arvutigraafikas on polügonaalvõrgustik graaf, mille tipud ja servad moodustavad polügone ehk hulknurki, mis omakorda moodustavad tervikliku hulktahkse objekti. Selle graafi tippude asukohad virtuaalses ruumis määratakse kolme koordinaadiga: X, Y ja Z. Tüüpiliselt kasutatavad hulknurgad on kolmnurgad ja nelinurgad, millest soositumad on kolmnurgad tänu nende esitamise ja kuvamise lihtsusele. (Terrazas & Ostuni & Barlow, 2002)

Geomeetrilise kujundi selline esitus ei ole eriti täpne – suurte siledate pindade puhul on säilitatavate andmete hulk suhteliselt väikene, kumerate pindade korral on rahuldava tulemuse saavutamiseks vajalik tunduvalt tihedam võrgustik, mille korral on säilitatavate andmete hulk suurem, kuid sellegipoolest on tegemist ikkagi ainult objekti umbkaudse kirjeldusega. Reaalajaliste rakenduste korral on aga polügonaalvõrgustike tippude hulga hoidmine võimalikult väikesena omaette eesmärgiks.

1.1. Polügonaalvõrgustike haldamine

Reaalajaliste rakenduste korral kasutatakse harva ainult ühte objekti, tavaliselt koosneb stseen paljudest objektidest, mis ei pea olema omavahel kuidagi seotud, seetõttu on erinevate objektide polügonaalvõrgustikud jagatud eraldi tippude massiivideks, millest igaüks kirjeldab omaette objekti. Iga objekti tippude asukohad on määratud selle objekti kohalikus koordinaatsüsteemis, see võimaldab lihtsamalt manipuleerida objekti sisemise geomeetriaga ning vältida informatsiooni kadu objekti transformeerimise korral stseeni koordinaatide suhtes. (Hook, 1995)

Objekti asukoht stseeni suhtes määratakse ära 4x4 transformatsiooni maatriksi abil, mis võimaldab kirjeldada objekti asukohta, rotatsiooni ning suurust virtuaalse maailma koordinaatsüsteemi suhtes. (Hook, 1995; Terrazas & Ostuni & Barlow, 2002)

1.2. Polügonaalvõrgustike kuvamine

Polügonaalvõrgustikel põhinevate objektide kuvamise protsessis omabki säilitatavate andmete hulga hoidmine minimaalsena suurt tähtsust. Nimelt tuleb geomeetria kuvamise käigus töödelda üht või teist moodi praktiliselt kogu stseenis sisalduv info, ning seda tuleb teha interaktiivsuse saavutamiseks vähemalt kümme korda sekundi jooksul (tänapäeval soovitavalt kuni 60 korda sekundis). Erinevate meetoditega on aga võimalik töödeldavate andmete hulka oluliselt vähendada.

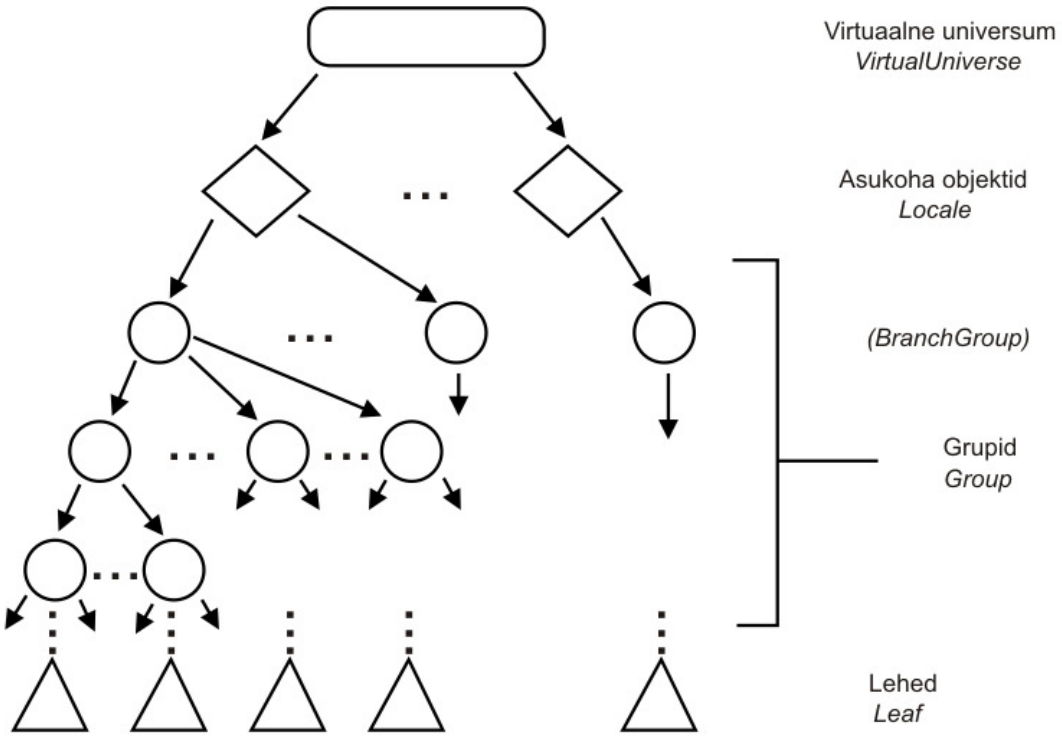
Suurte virtuaalsete maailmade puhul jaotatakse stseen osadeks, seetõttu ongi esimeseks informatsiooni vähendamise meetodiks kaugele jäävates stseeni osades sisalduvate objektide andmete elimineerimine renderdamise protsessist. Järgmiseks leitakse kõik objektid, mis jäävad kaamera vaateväljast välja (näiteks liiga kaugel olevad objektid ning kaamera taha jäävad objektid). Selleks kontrollitakse iga allesjäänud objekti korral, kas tema piirid lõikuvad kaamera vaateväljaga või mitte. (Hook, 1995)

Järgmisena arvutatakse allesjäänud objektide transformatsiooni maatriksite abil nende objektide polügonaalvõrgustike tippude asukohad stseeni koordinaatsüsteemi suhtes. Seejärel on võimalik välja arvutada objekti valgustatust ja muid omadusi, mis kujunevad teiste stseenis sisalduvate objektide koostoimel. (Hook, 1995; Terrazas & Ostuni & Barlow, 2002)

Lõpuks transformeeritakse objektide tipud kaamera koordinaatsüsteemi, arvutatakse perspektiivist tingitud deformatsioonid, järjestatakse tipud kaamera Z-telje (kaugus kaamerast) järgi ning eemaldatakse veel allesjäänud üleliigsed andmed (näiteks teiste hulknurkade taha jäävad hulknurgad). Stseeni kuvand valmib allesjäänud tippude poolt moodustuvate hulknurkade joonistamisega ekraanile. (Hook, 1995; Terrazas & Ostuni & Barlow, 2002)

2. 3D objektide kujutamine Java 3D abil

Java 3D API on kõrgtasemeline rakenduse programmeerimise liides, mille abil on võimalik luua kolmemõõtmelise graafikaga rakendusi. Mitmekesiste virtuaalsete stseenide ehitamist võimaldab paindlik stseeni-graafi (*scene graph*) põhine mudel, mis kujutab endast virtuaalse universumi külge kinnitatud erinevat tüüpi objektidest koosnevat hierarhilist struktuuri, milles iga sõlmpunkt on vajalik stseeni terviklikuks toimimiseks. (Sun Microsystems, Inc., 2007)



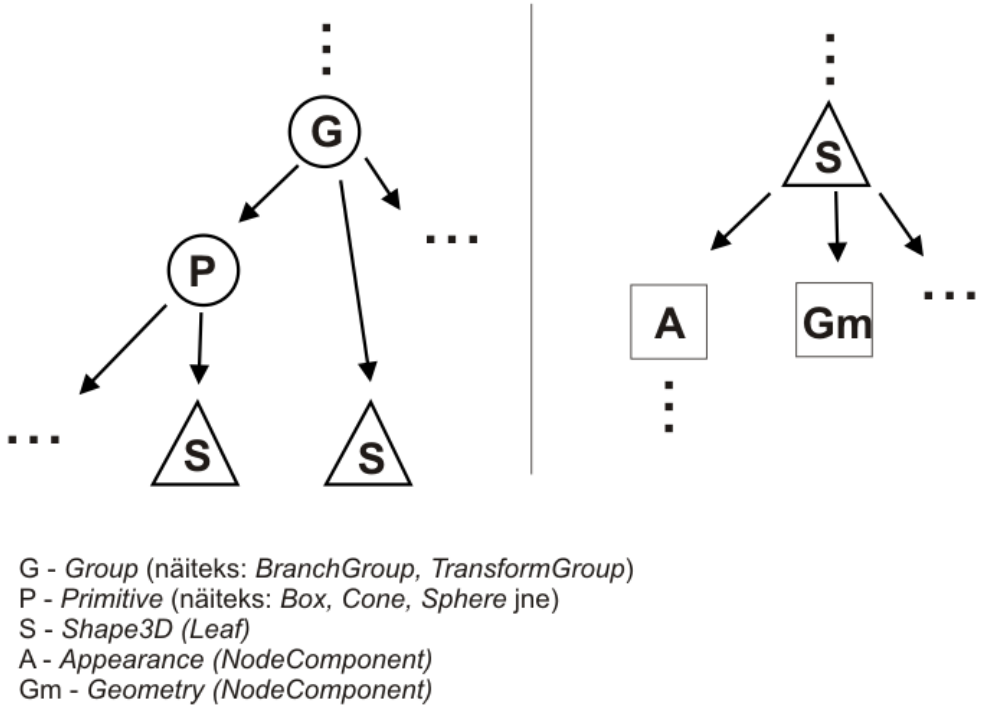
Joonis 2.1. Java 3D hierarhiline stseeni-graafi struktuur

Joonisel 2.1 on näha, et asukoha objektide (*Locale*) külge, mis määravad ära vastava haru ruumilise paiknemise virtuaalses universumis, kinnituvad omakorda erinevad sõlmpunkte grupeerivad *Group* tüüpi objektid. Tähtsaim nendest on *BranchGroup*, mis esindab ühte

stseeni haru, mille külge saab kinnitada ülejäänud stseenis sisalduvaid virtuaalseid objekte.

Reaalselt stseenis nähtavaid objekte esindavad *Leaf* tüüpi objektid. Klassi *Leaf* alamklassideks on näiteks *Light* ja *Shape3D*, millest esimene on baasklassiks erinevatele valgusallikatele ning teine esindab geomeetrist kujundit. (Sun Microsystems, Inc., 2007)

Peale *Shape3D* tüüpi objektide esindavad geomeetrilisi kujundeid ka komplekssed *Primitive* tüüpi objektid (*Box*, *Cone*, *Sphere* ja *Cylinder*), mis on tegelikult klassi *Group* alamklassid ning sisaldavad endas *Shape3D* tüüpi objekte (vt joonis 2.2, vasakul), mis kirjeldavad reaalselt nähtavaid geomeetrilisi kujundeid. *Primitive* klass on vaid ümbris, mis konstrueerib vajalike omadustega *Shape3D* objekti ning manipuleerib sellega.



Joonis 2.2. *Shape3D* tüüpi objektid stseeni-graafis. Vasakul *Shape3D* objektide kinnitumine *Group* tüüpi sõlmpunktide külge, paremal *Shape3D* objektis sisalduvad *NodeComponent* tüüpi elemendid

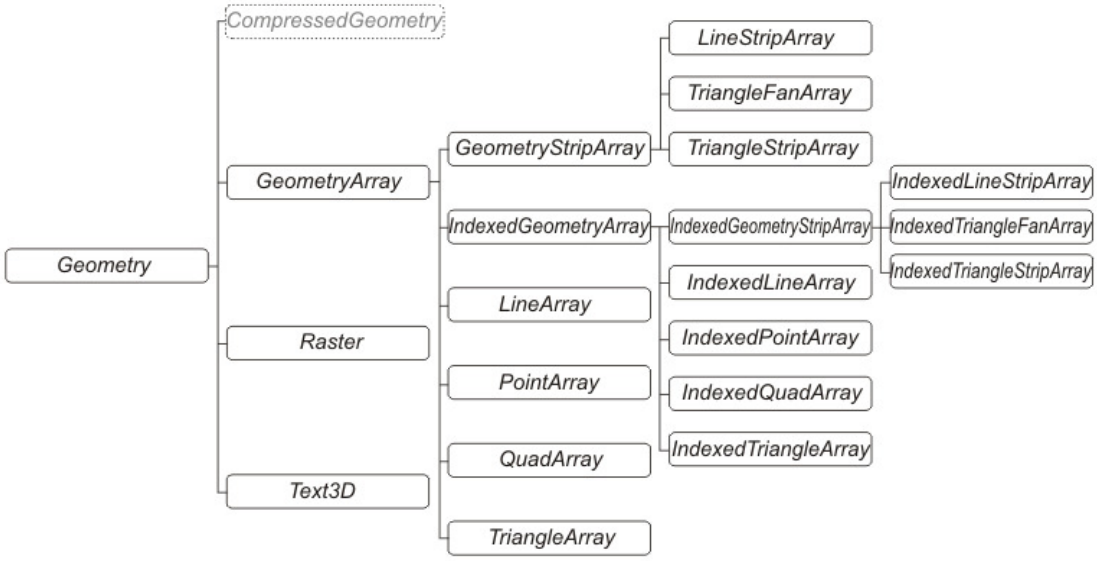
Shape3D objekt sisaldab omakorda ühte *Appearance* tüüpi objekti, mis kirjeldab temale viitava objekti välimust (värv, materjal, tekstuur jne), ning ühte või mitut *Geometry* tüüpi

objekti. *Geometry* objekt ongi see struktuur, milles hoitakse vastava geomeetrilise kujundi kirjeldust.

2.1. Klass *Geometry*

Kõikide geomeetriat kujutavate Java 3D klasside baasklassiks on klass *Geometry*. See on abstraktne klass, mis kirjeldab geomeetria komponendi informatsiooni, mida on vaja talle viitava *Shape3D* tüüpi sõlmpunktil vastava geomeetrilise kujundi kujutamiseks virtuaalses universumis. (Sun Microsystems, Inc., 2007)

Kuna tegemist on abstraktse klassiga, ei kirjeldata selles otseselt geomeetriat, see määrab vaid kõikide oma alamklasside ühised omadused.



Joonis 2.3. Klass *Geometry* ja tema alamklassid

Geometry klassil on terve hulk alamklasse (vt joonis 2.3), nendest otsesed alamklassid on *CompressedGeometry*, *Raster*, *Text3D* ning abstraktne *GeometryArray*.

CompressedGeometry, mis on praeguseks juba vananenud ja selle kasutamine on muutunud ebasoovitatavaks, sisaldab objekti geomeetriat kirjeldavat informatsiooni

pakitud kujul. See on põhiliselt kasulik andmeedastuskiiruse tõstmiseks üle võrgu. (Terrazas & Ostuni & Barlow, 2002)

Klass *Raster* on mõeldud kahemõõtmelise pildi paigutamiseks kolmemõõtmelisse stseeni. See kujutab endast stseenis paiknevat kahemõõtmelist plaati, mille pinnale kantakse vastava pildi pikselid. (Terrazas & Ostuni & Barlow, 2002)

Text3D on klass, mis visualiseerib etteantud teksti stseenis esitatavate kolmemõõtmeliste tähemärkidena. (Terrazas & Ostuni & Barlow, 2002)

2.2. Klass *GeometryArray*

Java 3D kasutab polügonaalvõrgustikel põhinevat reaalajalist kolmemõõtmeliste objektide renderdamise tehnikat. Seetõttu moodustavad kõiki geomeetrilisi kujundeid virtuaalsed hulknurgad ehk polügonaalvõrgustikud, mis koosnevad tippudest, neid ühendavatest servadest ning nende vahele jäävatest pindadest.

Eelnevalt kirjeldatud geomeetrilisi kujundeid kirjeldavaks Java 3D klassiks on *GeometryArray*, mis sisaldab endas kogu polügonaalvõrgustiku moodustavate tippudega seotud andmestikku. Tippudega seotud andmeid hoitakse ühes üldises või mitmes erinevas massiivis. Iga tipu puhul on võimalik kirjeldada selle tipu koordinaate, värvi, normaalvektorit ja tekstuuri koordinaate. (Sun Microsystems, Inc., 2007)

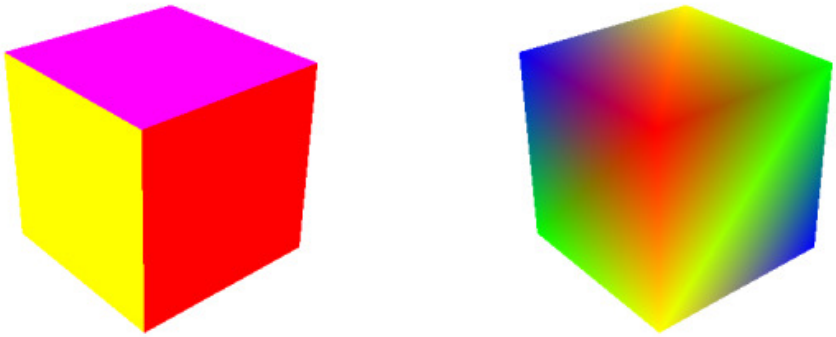
2.2.1. Tippude koordinaadid

Tippude koordinaadid on põhilised polügonaalgeomeetriliste objektide puhul säilitatavad andmed. Iga tipu asukoht objektis selle objekti kohaliku koordinaatsüsteemi suhtes tähistatakse kolme koordinaadiga – vastavalt X, Y ja Z teljel. Tippude koordinaadid on ainsad andmed, mille olemasolu on Java 3D puhul kohustuslik igas *GeometryArray* eksemplaris.

2.2.2. Tippude värvid

Tippude värvid määravad ära iga tipu värvi vastavas objektis. Tippude vahelise pinna värv arvutatakse välja teda ümbritsevate tippude värvide põhjal – kui kõik pinda

moodustavad tipud on ühte värvi, on see ka pinna värviks (vt joonis 2.4, vasakul), erinevat värvi tippude korral interpoleeritakse tippude vahele jääva ala värv (vt joonis 2.4, paremal).



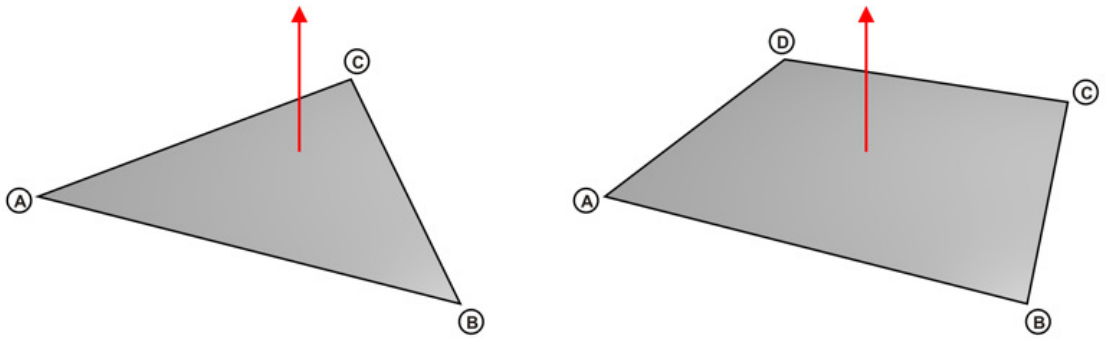
Joonis 2.4. Tippude värvid. Vasakul on värvid määratud pindade järgi, paremal tippude järgi.

Java 3D puhul on kasutusel kahte tüüpi tippude värve: kolme- ja neljakomponendilised. Esimesel juhul määravad värvi R (*red*), G (*green*) ja B (*blue*) komponent ning neljakomponendilisel värvil lisandub A (*alpha*) ehk läbipaistvuse komponent. Kõik komponendid kirjeldatakse *float* tüüpi muutujatega, mille korral 0.0 tähistab komponendi minimaalset väärtust ning 1.0 maksimaalset. (Sun Microsystems, Inc., 2007)

2.2.3. Normaalid

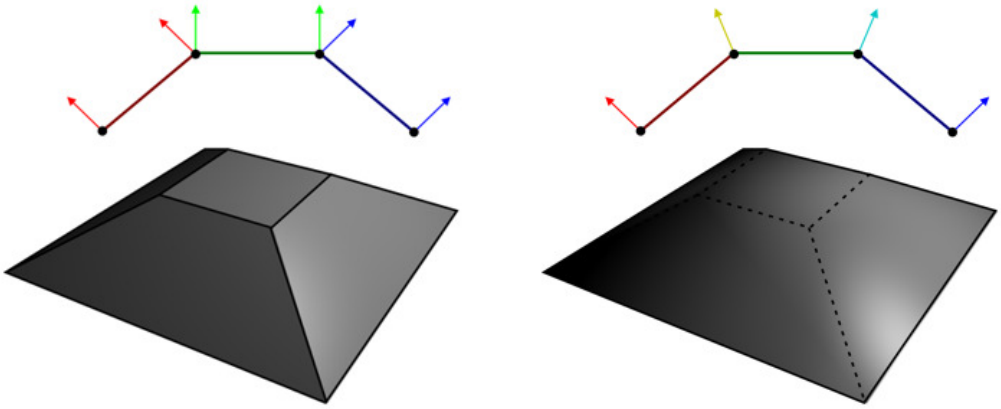
Polügonaalvõrgustikel põhineva geomeetria puhul on kasutusel kahte tüüpi normaale - pinnanormaalid ja tippude normaalid. Mõlemal juhul peab vastav tippude võrgustik moodustama vähemalt kolmetipulisi pindu.

Pinadade normaalid arvutatakse välja objekti renderdamisprotsessi käigus. Pinna normaal kujutab endast pinnaga risti olevat vektorit, mis määrab ära pinna suuna. Nimelt on arvutigraafikas reaalajaliselt kujutatavad pinnad enamasti ühepoolsed – see tähendab seda, et pinnad, mille normaalvektor on suunatud vaatajast (kaamerast) eemale, on nähtamatud, neid ei kaasata optimaalsuse huvides renderdamise protsessi. Pinna suunavektor määratakse ära teda moodustavate tippude järjekorra järgi, see tähendab seda, et esikülje poolt vaadatavat pinda mooduatavad tipud paiknevad vastupäeva (vt joonis 2.5). (Sabbarton, n.d.)



Joonis 2.5. Pinda moodustavate tippude paiknemise järjekord määrab ära selle pinna normaalvektori suuna.

Tippude normaalid määravad ära selle kuidas on vastav objekt varjutatud. Reaalajaliste rakenduste puhul arvutatakse objekti värvus selle mingis piirkonnas mitmete parameetrite alusel: objekti värvust ja heledust vastavas piirkonnas mõjutab tipu või tekstuuri värv ning vastavale pinnale langeva valguse värv, intensiivsus ja pinnale langemise nurk. Objekti pinna ja sellele langeva valguse vaheline nurk määrataksegi ära tippude normaalide suhtes. (Wagner, 2004)



Joonis 2.6. Tippude normaalvektorid määravad ära vastava pinna varjutamise. Vasakul lamedad pinnad, paremal siledad pinnad.

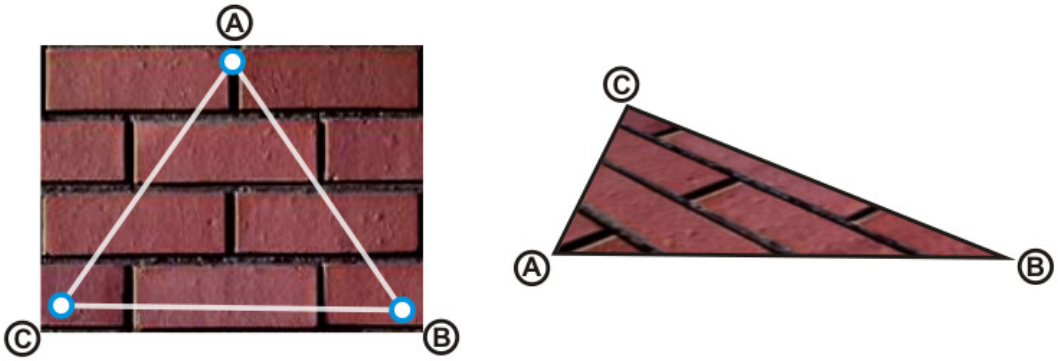
Tippude normaalide põhise varjutamisega on võimalik objekte moodustavaid pindasid renderdada nii lamedatena (*flat*) kui ka siledatena (*smooth*). Lamedana ehk selgepiiriliste teravate servadega näib pind siis, kui kõikide teda moodustavate tippude normaalvektorid

on võrdsed pinna normaalvektoriga (vt joonis 2.6, vasakul). Siledana, näiliselt kumerana, kujutatava pinna korral kasutatakse pinda moodustavate tippude normaalide arvutamisel ka naaberpindade normaale, teisisõnu tipu normaalvektor on kõikide teda vahetult ümbritsevate pindade normaalvektorite keskmine (vt joonis 2.6, paremal). (Wagner, 2004)

2.2.4. Tekstuuri koordinaadid

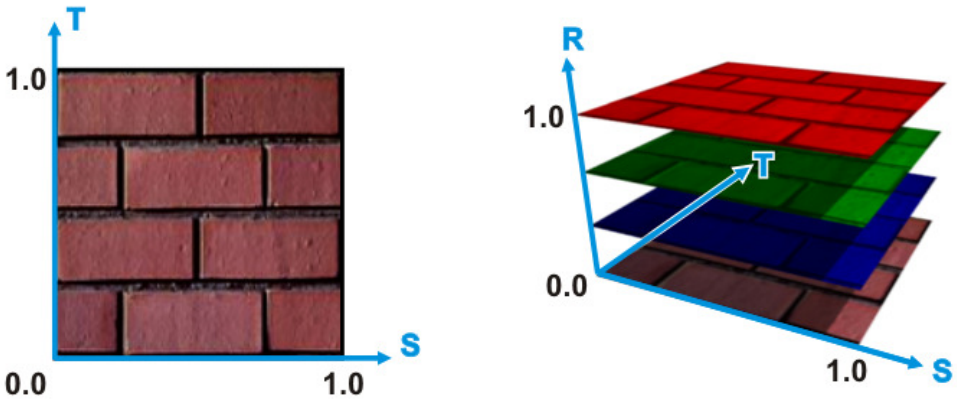
Tekstuuri koordinaadid kirjeldavad seda, kuidas kolmemõõtmelisele objektile määratud tekstuur laotatakse selle objekti pinnale. Tekstuurina kasutatavale pildile määratakse loogilised koordinaadid, mis tähendab seda, et pildi alumine vasak nurk on koordinaatidega (0.0, 0.0) ning ülemine parem nurk (1.0, 1.0). Kõik vahepealsed punktid pildil on määratavad ujukomaarvuliste koordinaatidega. (Bouvier, n.d.; Sun Microsystems, Inc., 2007)

Polügonaalvõrgustikus tähendab see seda, et iga tipu tekstuuri koordinaadid tähistavad mingisugust punkti tekstuuri pildil (vt joonis 2.7, vasakul) ning tippude kuvamisel ekraanile joonistatakse nende tippude poolt moodustuvale hulknurgale vastava tekstuuri pildi osa nii, et vastavate tippude poolt määratud pildi punktid asuvad ekraanil nende tippude poolt määratud kohtades (vt joonis 2.7, paremal).



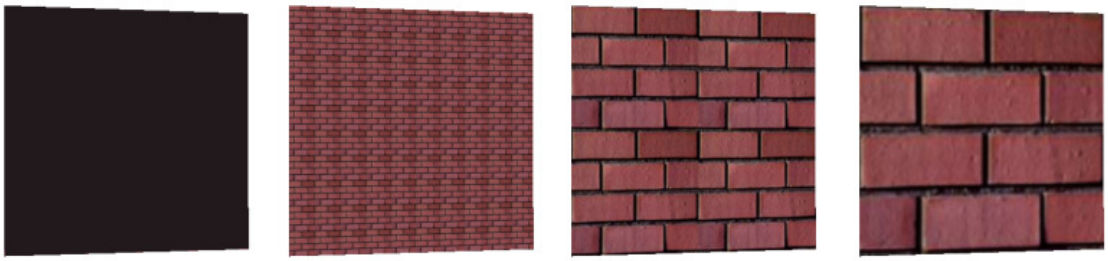
Joonis 2.7. Vasakul kolmnurga tippude tekstuuri koordinaatide määramine, paremal sama kolmnurk stseenis.

Java 3D puhul on võimalik kasutada nii kahe-, kolme-, kui ka neljamõõtmelisi tekstuuri koordinaate. Kõigil neil juhtudel on kõikidel telgedel kasutatavate koordinaatide väärtused nullist üheni ning neid telgi tähistatakse vastavalt S, T, R, Q. Kahemõõtmeliste tekstuuri koordinaatide puhul on asi lihtne – S-telje väärtus tähistab asukohta tekstuuri pildi X-teljel ning T-telje väärtus pildi Y-teljel (vt joonis 2.8, vasakul). Kolmemõõtmeliste tekstuuri koordinaatide kasutamine eeldab ka, et vastavale objektile on määratud kolmemõõtmeline tekstuur. See tähendab seda, et tekstuur koosneb mitmest üksteise peal kohakuti asuvast pildist ning väärtus tekstuuri koordinaatide R-teljel määrabki ära selle, mitmendat pilti sellest tekstuurist kasutatakse vastava tipu jaoks ning kui palju sulanduvad erinevad pildid omavahel objekti pinnal (vt joonis 2.8, paremal). (Sun Microsystems, Inc., 2007)



Joonis 2.8. Kahe- ja kolmemõõtmeline tekstuur.

Neljamõõtmeline tekstuur on keerulisem. Neljanda mõõtme ehk Q-teljel oleva väärtuse abil on võimalik manipuleerida teistel telgedel olevate väärtustega. Joonisel 2.9 oleva neljamõõtmelise tekstuuri näite abil on võimalik järeldada, et tekstuuri S ja T-telje uued koordinaadid arvutatakse vastava telje väärtuse jagamisel Q-telje väärtusega. Siin on ka näha, et ühest väiksema Q-telje väärtuse korral hakkab tekstuur korduma. See tuleneb sellest, et tegelikkuses ei pea tekstuuri koordinaadid jääma 0.0 ja 1.0 vahele, vahemikust välja jäävad väärtused surutakse sellesse vahemikku renderdamise käigus vastava tekstuuri omaduste põhjal.



Joonis 2.9. Neljamõõtmeliste tekstuuri koordinaatide kasutamine. Vasakult: väärtused Q-teljel vastavalt 0.0, 0.1, 0.5 ja 1.0

2.3. Tippude grupeerimine

Tippude grupeerimine on protsess, mille käigus selgitatakse välja kuidas ja millised tipud moodustavad geomeetrilisi primitiive (jooned, kolmnurgad ja nelinurgad), mida oleks võimalik ekraanile kuvada. Kõige lihtsam on opereerida punktgeomeetriaga, mille puhul kuvatakse ekraanile vaid igat tippu esinadavad punktid, joongeomeetria puhul vajatakse iga joone genereerimiseks kahte tippu ning kolmnurkade ja nelinurkade korral vastavalt 3 ja 4 tippu.

Eelkirjeldatud erinevate tippude grupeerimismeetodite jaoks on olemas vastavad Java 3D klassid: *PointArray*, *LineArray*, *TriangleArray* ja *QuadArray*. Tegemist on indekseerimata tippude massiividega, mis tähendab seda, et vastava objekti iga tippu kasutatakse renderdamise käigus ainult üks kord, mis tähendab omakorda seda, et ühelgi hulknurgal ei ole ühiseid tippe isegi siis kui need tipud kattuvad. Sellises olukorras on kõik hulknurgad üksteisest täiesti sõltumatud ning nendega on lihtne manipuleerida, kuid säilitatavate andmete hulk polügonaalvõrgustiku kohta on suhteliselt suur. Näitekse kui risttahukas (8 tippu) on moodustatud indekseerimata nelinurkadest, on säilitatavate tippude arv 24, kuna igas risttahuka nurgas asuvad kolme nelinurkse pinna tipud. (Sun Microsystems, Inc., 2007)

Üheks tippude hulga vähendamise võimaluseks on indekseeritud tippude massiivi kasutamine. See tähendab seda, et koordinaatide massiivis on esitatud kõikide tippude koordinaadid nii, et kattuvate tippude koordinaadid ei korduks. Tippude massiiv on eraldi massiiv, milles on märgitud iga tipu poolt kasutatavate koordinaatide indeks ehk asukoht

koordinaatide massiivis. Sama tehnikat saab kasutada ka kõikide teiste tippude atribuutide korral. Polügonaalvõrgustiku andmete säilitamiseks on see optimaalse mälu kasutuse tõttu hea variant, kuid tippude ja pindadega manipuleerimisel ning tippude arvu muutmisel võib tekkida vajadus massiivide ümberindekseerimiseks, mistõttu vajab see meetod indekseerimata tippudega võrreldes tunduvalt keerulisemaid algoritme. (Sun Microsystems, Inc., 2007)

Teiseks tippude hulga vähendamise võimaluseks on tippude grupperimine ribadeks (*strip*). Selleks on olemas vastavad *GeometryStripArray* alamklassid: *LineStripArray*, *TriangleStripArray* ja *TriangleFanArray*. *LineStripArray* puhul tähendab see seda, et iga joone lõpp-punkt on järgmise joone alguspunktiks, mistõttu on ühe katkematu *n* joonest koosneva jada esitamiseks vaja ainult *n+1* tippu. Sarnane on *TriangleStripArray*, mille korral moodustavad tipud kolmnurkadest koosnevaid ribasid, milles kolmnurgad on ühendatud külgi pidi kokku. *TriangleFanArray* puhul moodustavad kolmnurgad vihmavarju kujulise võrgustiku, milles kõik kolmnurgad paiknevad ringis ümber keskmise ühise tipu. See võimaldab sarnaselt indekseerimisele hoida kokku polügonaalvõrgustiku säilitamiseks kuluvat mälu hulka, kuid tippude ja pindadega manipuleerimine on samuti suhteliselt keeruline. (Sun Microsystems, Inc., 2007)

3. Java 3D redaktor ja geomeetria redaktor

Käesoleva bakalaureusetöö autorile oli peale Java 3D omaduste uurimise äärmiselt tähtsaks ka saadud teadmiste kasutamine praktikas ning programmeerida selline (kasutaja)liides, mis muudaks võimalikuks pääseda ligi Java 3D rohketele objektide geomeetriaga manipuleerimise võimalustele ilma programmeerimist oskamata.

Baasrakendusena on kasutatud töö autori käe all eelnevalt seminaritöö käigus valminud rakendust „Java 3D redaktor“, millele on suhteliselt lihtne lisada uusi funktsionaalsusi. Varem valminud rakenduse osa sisaldab ülejäänud komponente majutavat raamistikku ning stseeni redaktorit, mille abil on võimalik kombineerida erinevatest Java 3D abil loodavatest lihtsatest objektidest ja ka keerulisematest imporditavatest objektidest mitmekesiseid virtuaalseid stseene. Loodud stseene on võimalik eksportida ning kasutada erinevate Java 3D rakenduste poolt. Stseeni redaktor võimaldab stseeni töödelda objektide tasemel: lisada, kustutada ja transformeerida objekte ning manipuleeride mitmesuguste objektide atribuutidega (materjal, tekstuur ja tekstuuri atribuudid; valgusallikate puhul valgustugevus, värvus, jne).

Lisatava osa, geomeetria redaktori, põhiülesandeks on pakkuda võimalust pääseda kolmemõõtmeliste objektide sisse – manipuleerida objektide geomeetriaga. Geomeetria redaktorit planeerides tuli arvesse võtta tüüpilisi polügonaalvõrgustikega tehtavaid manipulatsioone, mis peaksid kindlasti kajastuma valmivas rakenduses, aga ka Java 3D eripärasid, mis nõuavad teatud funktsionaalsuste olemasolu. Esimeseks eesmärgiks oli luua vajalikud põhiklassid, mis suudaksid hallata vastavate Java 3D objektide andmeid ning neid dünaamiliselt muuta. Seejärel sai hakata mõtlema kuidas luua struktuure, mis võimaldaksid lihtsalt ja interaktiivselt ehitada uusi polügonaalvõrgustikke, transformeerida nende tippe ja laotada neile kõikvõimalikke tekstuure, ning viimaks, kuidas teha kasutajale kättesaadavaks ka hulk teisi Java 3D poolt pakutavaid võimalusi (näiteks normaalide ja värvidega manipuleerimine).

4. Geomeetriaga manipuleerimine rakenduses

Võimaldamaks rakenduse dünaamilist käitumist, kus kasutaja saab vastavalt vajadustele Java 3D klasside poolt moodustatud struktuure muuta ja nendega manipuleerida, on tarvis vajalikke abistruktuure, mis suudaksid vastavaid Java 3D klasse hallata. Seetõttu on käesoleva bakalaureusetöö käigus töö autori poolt integreeritud seminaritööna valminud rakendusele suur hulk uusi klasse.

Objektide geomeetriaga manipuleerimise sisulise poole eest vastutavad pakettides *j3deditor.bin.hierarchy* ja *j3deditor.bin.hierarchy.util* asuvad klassid. Lihtsuse ja enamkasutatavuse tõttu on rakenduses kasutatud ainult *GeometryArray* otseste alamklasside poolt võimaldatavaid konstruktsioone, mis on mõeldud indekseerimata tippude massiivide haldamiseks.

4.1. Geomeetria haldamine

Põhiliseks kolmemõõtmeliste objektide polügonaalvõrgustikku haldavaks klassiks on *J3DeGeometryArray*, mis sisaldab endas *GeometryArray* tüüpi objekti ning *Vertex* tüüpi objektide massiivi, mis initsialiseeritakse *J3DeGeometryArray* loomise käigus. *J3DeGeometryArray* on mõeldud ainult *GeometryArray* otsestes mitte-abstraktsetes alamklassides (*PointArray*, *LineArray*, *TriangleArray* ja *QuadArray*) sisalduvate andmete haldamiseks.

Vertex tüüpi objektid ei sisalda otseselt andmeid tippude koordinaatide ja teiste attribuutide kohta, nad peavad meeles oma indeksit, mitmendal kohal nende poolt viidatava tipu andmed asuvad *GeometryArray* massiivides. Samuti sisaldab *Vertex* teisi rakenduse tööks vajalikke andmeid, mis määravad ära tippude hetkelised omadused, näiteks mis olukus vastav tipp parajasti on: normaalolekus, peidetud, aktiivne või liigutatav. Kuigi *Vertex* klass sisaldab suurel hulgal erinevaid meetodeid tippude andmetega manipuleerimiseks, ei pääse ta ise otseselt ligi klassis *GeometryArray* asuvatele andmetele, ta küsib ja muudab reaalseid vastava tipuga seotud andmeid läbi *J3DeGeometryArray* tüüpi objekti.

4.1.1. Ligipääs tippude andmetele

Tippude andmed klassis *GeometryArray* võivad olla hoitud mitmel erineval kujul. Esimese võimaluse puhul, mis on väiksemate objektide korral sagedamini kasutatav, kasutatakse *GeometryArray* objekti siseseid massiive. Sisemistes massiivides paiknevate andmetega manipuleerimine on suhteliselt lihtne, iga tipu kõik atribuudid on eraldi küsitavad ja muudetavad vastavate meetodite abil. Näiteks tipu koordinaatide üks küsimise võimalus on näidatud koodinäites 4.1.

```
//ühe tipu x, y ja z koordinaatide hoidmise massiiv
float[] vertex = new float[3];

//vastava indeksiga tipu koordinaatide küsimine massiivi
geometryArray.getCoordinate(index, vertex);
```

Koodinäide 4.1. Tipu koordinaatide küsimine *GeometryArray* sisesest massiivist.

Kui klassi *GeometryArray* eksemplaril on määratud *by reference* omadus, siis ei ole objekti sisemised massiivid kasutusel, kasutatakse rakenduse poolt initsialiseeritud massiive, milles õigete andmete leidmine jääb programmeerija ülesandeks. Näiteks mingi tipu koordinaatide küsimiseks tuleb küsida *GeometryArray* käest viide tippude koordinaatide massiivile ning seejärel leida massiivist vastava tipu kolm koordinaati (vt koodinäide 4.2).

```
//ühe tipu x, y ja z koordinaatide hoidmise massiiv
float[] vertex = new float[3];

//kõikide tippude koordinaatide massiivi küsimine
float[] coordinates = geometryArray.getCoordRefFloat();

//vastava indeksiga tipu koordinaatide leidmine
vertex[0] = coordinates[3 * index];
vertex[1] = coordinates[3 * index + 1];
vertex[2] = coordinates[3 * index + 2];
```

Koodinäide 4.2. Tipu koordinaatide leidmine tippude koordinaatide massiivist.

By reference omaduse korral võib veel olla määratud ka *interleaved* omadus, mille korral asuvad kõikide tippude kõik andmed ühes ühises *float* tüüpi massiivis. Sellisel juhul on vastava objekti renderdamine küll kiirem, kuna andmete küsimiseks tehtavate erinevate meetodite välja kutsumisi tehakse tunduvalt vähem kui *GeometryArray* sisemiste massiivide korral, kuid tippude andmete kätte saamine on tunduvalt keerulisem.

```
//ühe tipu x, y ja z koordinaatide hoidmise massiiv
float[] vertex = new float[3];

//kõikide tippude kõikide atribuutide küsimine
float[] coordinates = geometryArray.getInterleavedVertices();

//ekvivalentsete atribuutide vahe atribuutide massiivis
int between = (int)(coordinates.length/geometryArray.getVertexCount());

//vastava indeksiga tipu koordinaatide leidmine
vertex[0] = coordinates[(index + 1) * between - 3];
vertex[1] = coordinates[(index + 1) * between - 2];
vertex[2] = coordinates[(index + 1) * between - 1];
```

Koodinäide 4.3. Tipu koordinaatide leidmine tippude atribuutide massiivist.

Nagu koodinäitest 4.3 selgub, tuleb esiteks küsida kõiki tippude andmeid sisaldav massiiv ning seejärel leida vastava tipu indeksi põhjal otsitavate andmete asukohad massiivis.

Seetõttu toimubki rakenduses tippude andmetega manipuleerimine klassi *J3DeGeometryArray* vahendusel. Klassi eksemplari loomise käigus tehakse kindlaks, kas hallatavad andmed asuvad tavalises, *by reference* või *interleaved* tüüpi *GeometryArray* objektis ning vastavalt sellele määratakse ka *J3DeGeometryArray* objekti tüüp. Klass ise sisaldab ühtseid tippude koordinaatide, värvide, tekstuuri koordinaatide ja normaalide küsimise ning määramise meetodeid, milles tehtav tegevus sõltub vastava *J3DeGeometryArray* eksemplari tüübist.

4.2. Polügonaalvõrgustikega manipuleerimine

4.2.1. Polügonaalvõrgustiku omaduste muutmine

Klassi *GeometryArray* puhul on tegemist suhteliselt staatilise kontruktsiooniga. Tippude arv, erinevate tippude atribuutide ja *GeometryArray* omaduste olemasolu on määratav vaid objekti loomise käigus. On küll olemas meetod *setValidVertexCount()*, millega saab ära määrata *GeometryArray* viimase nähtava tipu indeksi, kuid üldine tippude arv jääb muutumatuks terve *GeometryArray* objekti eluea jooksul. Seetõttu on käesoleva bakalaureusetöö käigus loodud rakenduse osas tippude lisamise võimalus lahendatud järgmiselt: geomeetrias sisalduvate tippude arv määratakse geomeetria loomisel ning nähtavate tippude arv võrdsustatakse nulliga. Tippe lisades suurendatakse nähtavate tippude arvu ning määratakse nähtavale ilmuvatele tippudele kasutaja poolt määratud koordinaadid. Seda protsessi saab jätkata seni kuni kõik olemasolevad tipud on kasutusel, seejärel tuleb polügonaalvõrgustiku suurust muuta.

Seetõttu sisaldab klass *J3DeGeometryArray* meetodeid, millega saab juba olemasoleva polügonaalvõrgustiku suurust või teisi omadusi muuta. Mõlemal juhul luuakse uute omadustega *GeometryArray* objekt ning seejärel kopeeritakse olemasolevate tippude andmed uude polügonaalvõrgustikku. Muuta saab tippude hulka, *GeometryArray by reference* ja *interleaved* omaduste olemasolu ning seda, milliseid tippude atribuute on võimalik kasutada.

Viimane omadus on eriti kasulik näiteks olukorras, kus imporditud objekti polügonaalvõrgustikul puuduvad näiteks tippude normaalide või tekstuuri koordinaatide määramise võimalused, aga kasutajal on neid parasjagu vaja. Lihtsalt on võimalik juba olemasoleva objekti andmed kopeerida teise teiste omadustega objekti ilma, et peaks uute omadustega objekti käsitsi otsast peale looma.

4.2.2. Klass *GeometryManipulator*

Polügonaalvõrgustike omadustega tuleb manipuleerida suhteliselt ettevaatlikult. Seetõttu, et vältida võimalike vigade tekkimist programmi töös, mis võivad kaasneda oskamatust andmete muutmisest väljaspoolt paketti, sisaldab klass *J3DeGeometryArray* üsna vähe

avalikke meetodeid. Et polügonaalvõrgustike andmetega oleks siiski võimalik midagi teha, asub samas paketis (*j3deditor.bin.hierarchy*) klass nimega *GeometryManipulator*, mis sisaldab suurel hulgal erinevaid meetodeid klassi *J3DeGeometryArray* poolt hallatavate andmetega ohutult manipuleerimiseks.

Klassi *GeometryManipulator* eksemplari loomise käigus määratakse talle *J3DeGeometryArray* objekt, mille andmetega manipuleerimiseks ta on mõeldud. Ühele *J3DeGeometryArray* objektile võivad korraga viidata mitu klassi *GeometryManipulator* eksemplari. Erinevalt klassist *J3DeGeometryArray*, mis hoolitseb ainult polügonaalvõrgustiku otseste andmete haldamise eest, tegeleb *GeometryManipulator* enamjaolt geomeetriaga manipuleerimisega seotud andmete haldamisega: näiteks millised tipud või pinnad on vastavas polügonaalvõrgustikus parajasti aktiivsed, ning mida nendega parasjagu tehakse.

4.2.3. Koordinaatidega manipuleerimine

Nii tippude kui ka tekstuuri koordinaatide muutmine toimub teiste tippude atribuutidega võrreldes tunduvalt keerulisemalt. Kui näiteks värvi saab aktiivsetele tippudele määrata mingi värvivaliku dialoogi kaudu, siis koordinaatidega paindlikuks manipuleerimiseks peab kasutaja saama valitud tippe ekraanil vabalt liigutada ning tulemus peab olema ka koheselt tajutav. Seetõttu sisaldab klass *GeometryManipulator* meetodeid tippude tüüpiliste transformatsioonide võimaldamiseks: liigutamine, pööramine ümber valiku keskpunkti ning valitud polügonaalvõrgustiku osa suuruse muutmine.

Tippude liigutamine on kõige lihtsam: kõiki aktiivseid tippe liigutatakse kasutajaliidese komponentide poolt määratud ulatuses mööda vajalikke koordinaattelgi. Selleks läbitakse tsükliga aktiivsete tippude massiiv, mis *GeometryManipulator* objektis on *Vertex* tüüpi objektide massiv ning kutsutakse iga tipu korral välja *translate()* või *translateTex()* meetod (vastavalt sellele, kas tegemist on tippude või tekstuuri koordinaatidega), millele antakse sisse *float* tüüpi massiiv, mis esinadab asukoha muutust mööda vastavaid koordinaattelgi.

Nii tippude pööramiseks kui ka valitud võrgustiku osa suuruse muutmiseks määratakse vastavasse režiimi sisenemisel aktiivsete tippude keskpunkt. See on võimalik nii tippude kui ka tekstuuri koordinaatide puhul. Kuigi Java 3D võimaldab kuni neljamõõtmelisi tekstuuri koordinaate, toimub tekstuuri koordinaatide liigutamine, pööramine ja valiku suuruse muutmine ainult tekstuuri S ja T telgede suhtes. Kolmanda ja neljanda mõõtme koordinaate on olemasolu korral võimalik muuta vastava dialoogi vahendusel.

Polügonaalvõrgustiku aktiivse osa suuruse muutmise korral läbitakse tsükliga kõik aktiivsed tipud ning arvutatakse nende tippude uued koordinaadid valiku keskpunkti suhtes vastavalt suuruse muutusele (vt koodinäide 4.4).

```
//koordinaatide küsimine varem leitud valiku
//keskpunkti esindava "Vertex" tüüpi objekti käest
float[] centerCoordinates = center.getCoordinates();

//aktiivsete tippude massiivi läbimine
for(int i = 0; i < selectedVertices.length; i++){
      Vertex v = selectedVertices[i];
      float[] coordinates = v.getCoordinates();
      for(int j = 0; j < coordinates.length; j++){
            //uue koordinaadi välja arvutamine
            //scaleChange – valitud võrgustiku suuruse muutus
            coordinates[j] +=
            ((coordinates[j] – centerCoordinates[j]) * scaleChange);
      }

      //uute koordinaatide määramine tipule
      v.setCoordinates(coordinates);
}
```

Koodinäide 4.4. Tippude koordinaatide arvutamine polügonaalvõrgustiku aktiivse osa suuruse muutmise korral.

Polügonaalvõrgustiku osa pööramine ümber selle võrgustiku osa keskpunkti läbitakse tsükliga kõik aktiivsed tipud ning leitakse iga tipu uued koordinaadid meetodiga *rotateVertex()* (vt koodinäide 4.5).

```
//vertex – tipu koordinaadid tasandil
//axis – pööramise telje koordinaadid tasandil
//rotation – pööramise nurk radiaanides
Point2f rotateVertex(Point2f vertex, Point2f axis, float rotation){

      //pööramise telje koordinaatide lahutamine tipu koordinaatidest
      vertex.sub(axis);

      //tipu pööramine "angle" radiaani ümber koordinaatide alguspunkti
      vertex.set((float)(vertex.getX() * Math.cos(angle) +
            vertex.getY() * -Math.sin(angle)),
            (float)(vertex.getX() * Math.sin(angle) +
            vertex.getY() * Math.cos(angle)));

      //pööramise telje koordinaatide liitmine tipu koordinaatidele
      vertex.add(axis);

      return vertex;
}
```

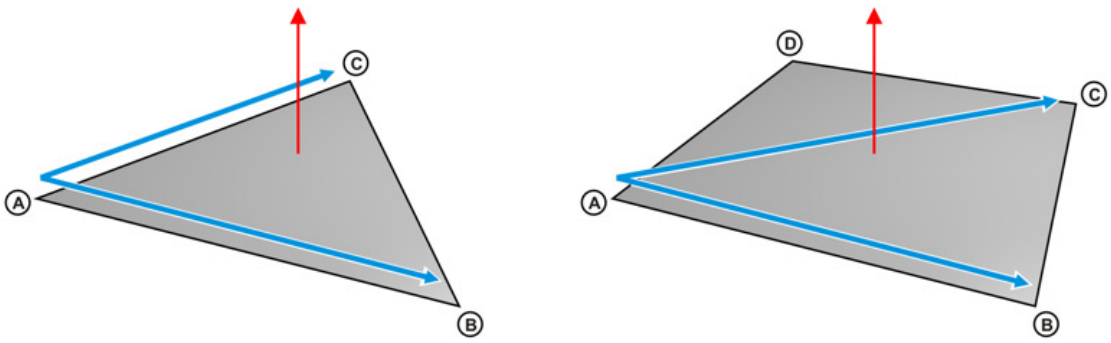
Koodinäide 4.5. Meetod *rotateVertex()* tipu pööramiseks ümber määratud telje.

4.2.4. Normaalvektoritega manipuleerimine

Pindade normaalidega manipuleerimiseks on klassis *GeometryManipulator* olemas vajalikud meetodid. Pinnanormaalidega manipuleerimine käib tippude kaudu, kuna pindade suunad määratakse ära automaatselt renderdamise protsessi käigus vastavalt tippude järjekorrale. Pindade ümberpööramise protsess toimub järgmiselt: tsükliga läbitakse aktiivsete pindade massiiv (kahemõõtmeline *Vertex* tüüpi massiiv ehk 1, 2, 3 või 4 kohaliste *Vertex* tüüpi massiivide massiiv) ning iga pinna moodustavate tippude järjekord pööratakse ümber.

Tippude normaalvektoritega manipuleerimiseks on spetsiaalne klass *NormalCalculator*, mis sisaldab meetodeid nii pindade lamendamiseks kui ka silumiseks.

Pindane lamendamine on suhteliselt lihtne ning kiire protsess. Tsükliga läbitakse kõik aktiivsed pinnad, iga pinna kolme tipu koordinaatide järgi arvutatakse välja selle pinna normaalvektor (vt joonis 4.1, koodinäide 4.6) ning seejärel määratakse saadud vektor kõikide seda pinda moodustavate tippude normaalvektoriteks.



Joonis 4.1. Pinna normaalvektori arvutamiseks vajalike pinnaga paralleelsete vektorite leidmine.

```
//face - kolme kohaline Vertex tüüpi massiiv
//pinnaga paralleelsete vektorite leidmine
Vector3f v1 = new Vector3f(face[1].getX() - face[0].getX(),
      face[1].getY() - face[0].getY(),
      face[1].getZ() - face[0].getZ());
Vector3f v2 = new Vector3f(face[2].getX() - face[0].getX(),
      face[2].getY() - face[0].getY(),
      face[2].getZ() - face[0].getZ());

//pinna normaalvektori leidmine
float nx = (v1.getY() * v2.getZ()) - (v1.getZ() * v2.getY());
float ny = - (v2.getZ() * v1.getX()) - (v2.getX() * v1.getZ());
float nz = (v1.getX() * v2.getY()) - (v1.getY() * v2.getX());
Vector3f normal = new Vector3f(nx, ny, nz);

//vektori normaliseerimine (vektori pikkus peab olema 1)
normal.normalize();
```

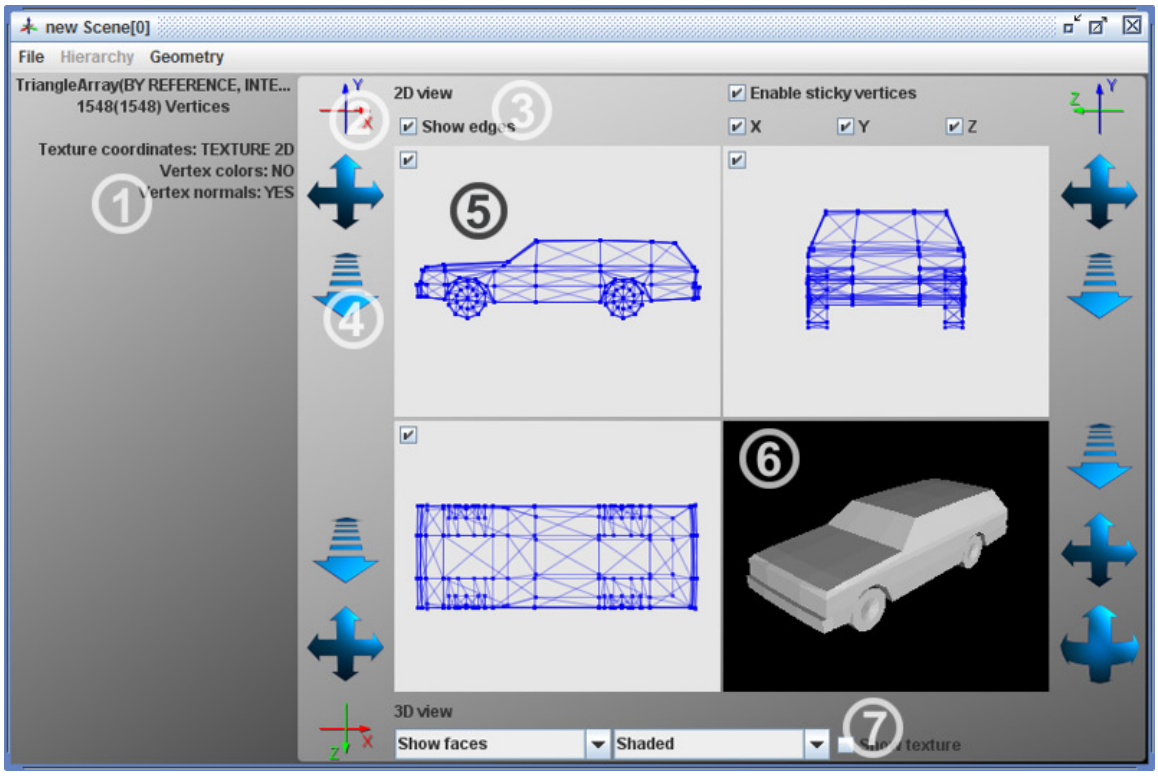
Koodinäide 4.6. Pinna normaalvektori arvutamine selle pinna kolme tipu järgi.

Pindade silumine on aga tunduvalt keerulisem ja aeganõudvam protsess, väga suure tippude hulga korral võib käesolevas rakenduses kasutatud meetodi puhul aega kuluda minuteid. Pindade silumise protsess algab aktiivsete pindade lamendamisega, mis tagab selle, et kõikide tippude normaalid oleksid normaliseeritud ning näitaksid pinnanormaalidega samasse suunda. Järgnevalt läbitakse tsükliga kõik aktiivsed tipud, iga tipu korral leitakse kõik teised tipud, mille asukoha koordinaadid on võrdsed eelnevaga ning arvutatakse kõikide nende tippude normaalide keskmine, seejärel määratakse saadud vektor kõikide nende tippude normaalvektoriteks. Tänu sellele, et tegemist on indekseerimata tippude massiiviga, on kirjeldatud protsess suhteliselt aeglane ka vaatamata sellele, et arvutatud normaaliga tipud jäetakse edasisest kontrollist välja.

5. Kasutajaliides

5.1. Kasutajaliidese visuaalne külg

Võimaldamaks ligipääsu käesoleva bakalaureusetöö käigus valminud geomeetriaga manipuleerivate komponentide poolt pakutavatele funktsionaalsustele, sai täiendatud ka rakenduse kasutajaliidest. 3D objektidega manipuleerimiseks on rakendusele lisatud spetsiaalne töölaud, geomeetria redaktor (vt joonis 5.1), mis on stseeni redaktori menüü kaudu avatav juhul, kui vastavas stseenis on aktiivne täpselt üks geomeetriat sisaldav objekt.



Joonis 5.1. Geomeetria redaktor. 1 – geomeetria informatsiooni paneel, 2 – vastava 2D vaate suunda määrav teljestik, 3 – 2D vaadete paneel, 4 – vastava 2D vaate kontrollnupud, 5 – 2D vaade, 6 – 3D vaade, 7 – 3D vaate paneel.

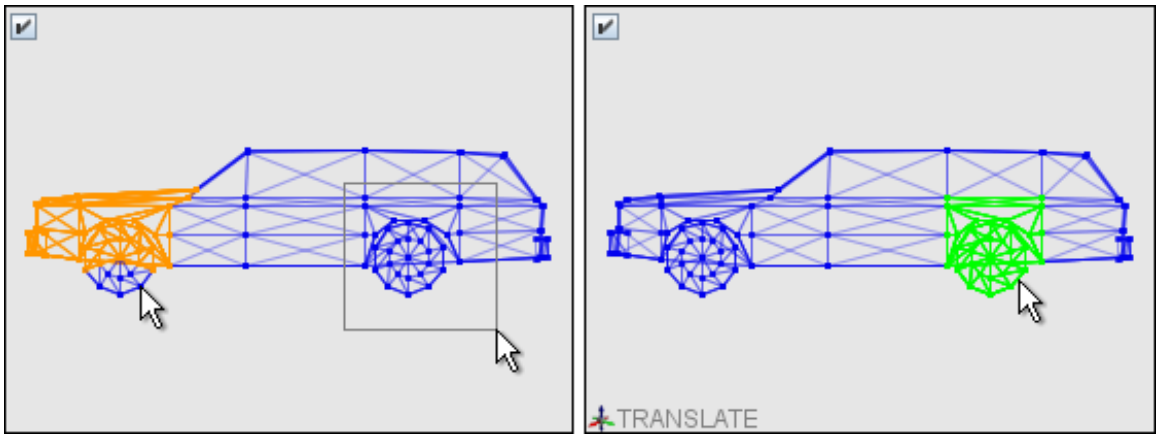
Geomeetria informatsiooni paneelil (vt joonis 5.1, 1) kuvatakse vastava *GeometryArray* kohta käiv info ning sellel paremat hiire nuppu klõpsates avaneb menüü erinevate tippude ja pindadega manipuleerimise võimalustega.

Geomeetria redaktori tööpaneel sisaldab kolme 2D vaadet ja ühte 3D vaadet, mida on võimalik vajaduse korral välja vahetada tekstuuri koordinaate näitava 2D vaatega. Iga vaate kõrval asuvad kontrollnupud selle vaate liigutamiseks ja suumimiseks ning 3D vaate puhul ka kaamera pööramiseks. 2D vaadete kõrval asuvad ka vaadete suundi näitavad teljestikud, millele klõpsates on võimalik vaate suunda muuta.

5.1.1. 2D vaade

2D vaade on geomeetria redaktori tähtsaim komponent. Sellel visualiseeritakse objekti geomeetria polügonaalvõrgustikku moodustavad tipud ja servad ning selle abil on ka võimalik tippe juurde lisada, vajalikke tippe aktiivseks muuta ja neid liigutada.

Tippe on võimalik juurde lisada mõnel 2D vaatel hiire parema nupuga klõpsates, juhul kui vastava polügonaalvõrgustiku kõik tipud ei ole veel kasutusel. Nähtavate tippude arvu võrdsustumise korral tegelike tippude arvuga, antakse kasutajale sellest märku vastava teatega, et rohkem ei ole võimalik sellesse polügonaalvõrgustikku tippe lisada. Vajadusel saab võrgustiku suurust muuta ning tippude lisamist jätkata.



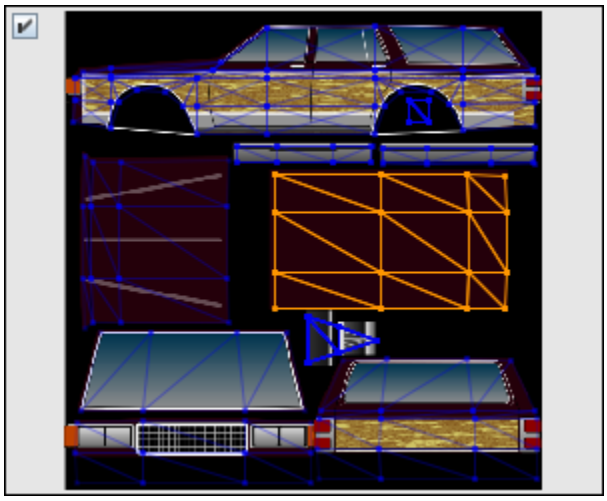
Joonis 5.2. 2D vaadete režiimid. Vasakul tippude valimine, paremal tippude liigutamine vastavas transformatsiooni režiimis.

Tippude valimine ehk aktiivseks muutmine on võimalik nii ükshaaval kui ka mitmekaupa. Ükshaaval valimise korral on võimalik määrata, kas aktiivseks muudetakse ainult üks kursorile lähimatest tippudest või ka ülejäänud samade koordinaatidega ehk esimesega kattuvad tipud. Hulgakesi valimiseks tuleb lohistada kursor üle vastava

piirkonna ning tekkinud nelinurga piiridesse jäävad tipud muudetakse aktiivseks (vt joonis 5.2, vasakul).

Peale tavarežiimi on 2D vaatel ka kolm transformatsiooni režiimi, millesse saab siseneda vastava klahvivajutusega juhul kui vähemalt üks tippudest on aktiivne. Erinevad transformatsiooni režiimid on mõeldud aktiivsete tippude liigutamiseks, pööramiseks ümber valiku keskpunkti ja valiku suuruse muutmiseks. Nendesse režiimidesse saab siseneda ja neist väljuda vastavalt klahvidega T (*translate*), R (*rotate*), S (*scale*). Ühes transformatsiooni režiimis olemisest annavad märku aktiivsete tippude roheline värvus ja vastava režiimi nimetus vaate vasakus allservas (vt joonis 5.2, paremal).

Sarnaselt käitub ka tekstuuri koordinaatidega manipuleerimiseks mõeldud 2D vaade. Sellel on olemas kõik eelkirjeldatud omadused, erinevusteks on vaid taustapildi olemasolu (vt joonis 5.3) ning tippude lisamise võimaluse puudumine.

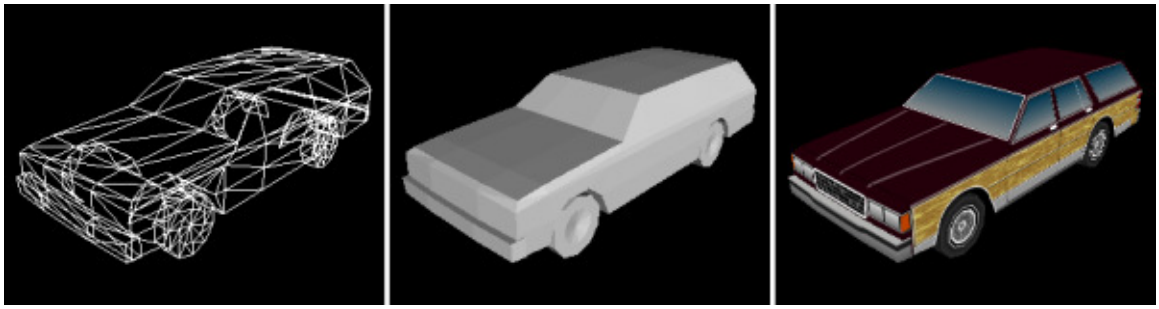


Joonis 5.3. Tekstuuri koordinaadid 2D vaates.

Tekstuuri koordinaatide vaates on parajasti nähtavad ainult need tipud, mis on aktiivseks muudetud mõnes teises 2D vaates. Nähtavaid tippe saab teistest vaadetest sõltumatult vastavalt vajadusele aktiivseks muuta ning neid vastavalt taustal näidatavale tekstuuri pildile vajalikku kohta nihutada.

5.1.2. 3D vaade

3D vaade on vajalik töödeldava objekti kuju ja sellega tehtavate muudatuste tajumiseks kolmemõõtmelises ruumis. Vastavalt polügonaalvõrgustiku manipuleeritavatele omadustele, on 3D vaatel võimalik valida sobiv objekti esitamise režiim.



Joonis 5.4. 3D vaate režiimid. Vasakult: nähtavad ainult servad, nähtavad pinnad ja objekt varjutatud, objekt varjutamata ja tekstuur nähtav.

Esimeseks 3D vaate muudetavaks omaduseks on valitav polügonaalvõrgustiku renderdamise režiim: kas näha on ainult võrgustiku tipud, servad (vt joonis 5.4, vasakul) või pinnad (vt joonis 5.4, keskel). Teiseks, kas objekt on varjutatud (vt joonis 5.4, keskel) või mitte. Ning kolmandaks, kas objekti tekstuur on nähtav (vt joonis 5.4, paremal) või mitte. Eelkirjeldatud omaduste erinevaid režiime on võimalik ka omavahel kombineerida.

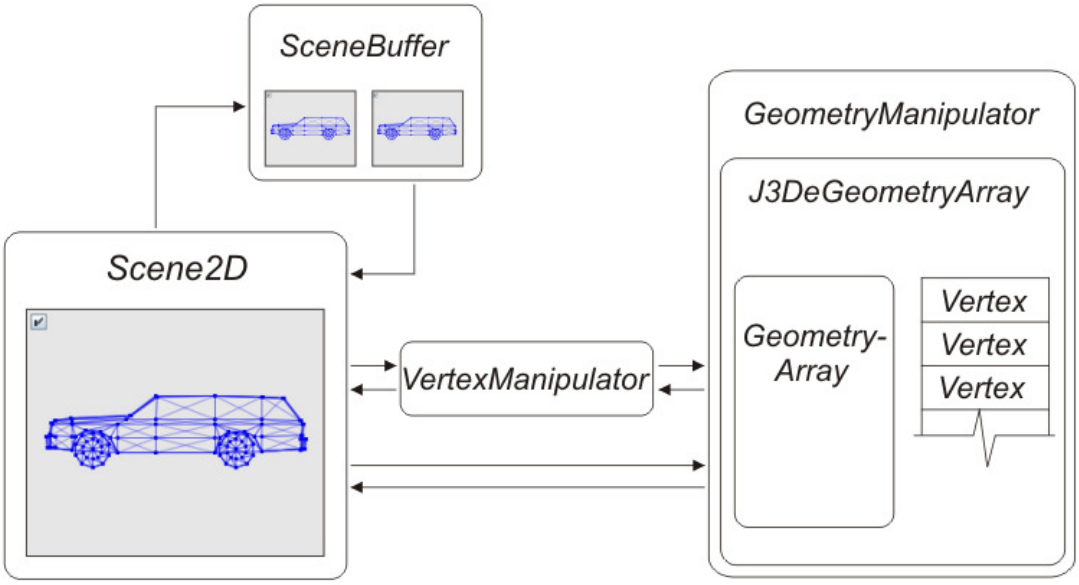
5.2. Kasutajaliidese ülesehitus

Geomeetria redaktori visuaalse poolega seotud klassid asuvad sarnaselt stseeni redaktorile pakettides *j3deditor.bin.editor*, *j3deditor.bin.editor.tools* ning *j3deditor.bin.editor.util*. Geomeetria redaktori baaskomponendiks on teisi komponente sisaldav klass *GeometryEditor*, milles kirjeldatav komponent vahetatakse stseeni redaktori vastu geomeetria redigeerimise režiimi sisenemisel.

5.2.1. Komponentide vaheline suhtlus

Geomeetria redaktori tähtsaimat komponenti 2D vaadet esindab paketis *j3deditor.bin.editor* asuv klass *Scene2D* ning teised temaga seotud klassid, mille

vahendusel käib suhtlus kolmemõõtmeliste objektide geomeetria polügonaalvõrgustikke haldava klassiga *J3DeGeometryArray* (vt joonis 5.5).



Joonis 5.5. Kasutajaliidese komponentide suhtlus polügonaalvõrgustikke haldavate klassidega.

Scene2D on 2D vaadet kirjeldav klass, mis hoolitseb nii objekti polügonaalvõrgustiku kuvamise eest ekraanil kui ka kasutaja poolsete muudatuste delegeerimise eest geomeetriat haldavatele klassidele. *SceneBuffer* uuendab klassi *Scene2D* poolt näidatava kuvandi puhvreid eraldi lõimes ning hoolitseb selle eest, et vastav 2D vaade kuvaks ekraanil viimati uuendatud puhvis olevaid andmeid.

VertexManipulator on staatiline klass, mis sisaldab erinevaid meetodeid tippude koordinaatide konverteerimiseks kolmemõõtmelise ruumi ja erinevat tüüpi 2D vaadete koordinaadistike vahel. Iga *Scene2D* eksemplar teab vaid seda, kas ta on üks tippude koordinaate kuvavatest vaadetest või hoopiski tekstuuri koordinaate kuvav vaade – see kuidas ja mitu tippu ta kuvab ning see, kuidas tippude liigutamise käsud suunatakse edasi vastavat polügonaalvõrgustikku kontrollivasse *GeometryManipulator* tüüpi objekti, määratakse ära *VertexManipulator* klassis paiknevate meetodite poolt arvestades vastava vaate omadusi.

5.2.2. Reageeriv kasutajaliides

Kolmemõõtmeliste objektide puhul võivad hallatavate andmete hulgad kasvada suhteliselt kiiresti väga suureks. Selliste andmetega erinevate arvutustehete tegemine võib aega võtta rohkem kui mõnikümmend millisekundit. Sellistel juhtudel peab vastavaid arvutusi tegema kasutajaliidesest eraldi lõimedes, et vältida kasutajaliidese hangumist märgatavaks ajaks.

Java kasutajaliidese klasside ja teiste graafiliste komponentide eest hoolitsevaks peamiseks lõimeks on sündmuste dispetšer (*event dispatching thread*), mis tegeleb nii sündmuste järjekorda (*event queue*) lisatud sündmuste poolt määratud tegevuste, kui ka erinevate komponentide ülejoonistamise nõuete täitmisega. Nimelt iga kord kui kasutaja teeb mingi toimingu, näiteks vajutab nupule, lisatakse viide vastava nupu kuularile sündmuste dišpetseri järjekorda ning selles kuularis sisalduv kood täidetakse siis kui järjekord jõuab vastava kuularini. Samuti toimitakse ka komponentide ülejoonistamisega, kui komponendi visuaalne pool muutub või kutsutakse rakenduses vastava komponendi *repaint()* meetod välja, lisatakse vastav komponent ootejärjekorda ning esimese võimaluse korral kutsub sündmuste dišpetser välja selle komponendi *paint()* meetodi. Seetõttu tulebki olla ettevaatlik mahukate algoritmide paigutamisega sündmuste kuularite ja komponentide *paint()* meetodite sisse.

Käesoleva rakenduse arendamise käigus tekkisid mahukamate polügonaalgeomeetriliste objektide korral kasutajaliidese hangumise probleemid seoses 2D vaatega. Iga muudatuse korral tippude asukohas või seisundis, tühjendatakse 2D vaade ning joonistatakse kõik vaate piiridesse jäävad tipud sellele uuesti. Iga tipu joonistamine hõlmab aga endas ka selle tipu asukoha välja arvutamist vastava 2D vaate suhtes. Kui töödeldav objekt sisaldab rohkelt üle tuhande tipu, muutub ka kasutajaliides suhteliselt aeglaseks ja võib hanguda peale igat liigutust häirivalt pikaks ajaks.

Seetõttu ei ole otstarbekas arvutada tippude asukohti ja joonistada neid ekraanile 2D vaate *paint()* meetodis. Selle asemel on hoopiski loodud abiklass *SceneBuffer*, mis sisaldab kahte *BufferedImage* tüüpi objekti. Kui sündmuste dišpetser lõim kutsub välja *Scene2D paint()* meetodi, pöördub *Scene2D* enda *SceneBuffer* objekti poole ning

joonistab ekraanile vastavas stseeni puhvris oleva eelnevalt valmis joonistatud *BufferedImage* tüüpi objektis sisalduva pildi. Stseeni puhvris oleva pildi ülejoonistamine toimub ainult juhul kui vastav *Scene2D* on seda nõudnud – kui stseenis on midagi muutunud või kui *Scene2D* objekti suurus ekraanil on muutunud. Kahte *BufferedImage* tüüpi objekti sisaldab *SceneBuffer* seetõttu, et kui ühele nendest parasjagu joonistatakse, siis teisel olev pilt on pidevalt kättesaadav sellega seotud *Scene2D* objektile, ning kui esimene *BufferedImage* on valmis joonistatud, siis vahetatakse nad omavahel ära, ehk see millele joonistati, tehakse kättesaadavaks *Scene2D* objektile ning teist saab vajadusel hakata üle joonistama.

SceneBuffer'is tehtava töö taga seisab spetsiaalselt rakenduse tarbeks tehtud klass *BackgroundThread*, mis on lihtsa esitusega madala prioriteetsusega lõim, kuid mis on sellegipoolest suhteliselt kiire paljude järjestikuste ülesannete täitmiseks. *BackgroundThread* sisaldab *ConcurrentLinkedQueue* tüüpi järjekorda, millesse saab lisada *BackgroundTask* liidest omavaid objekte. Lisamiseks on erinevad meetodid ilma kotrollita lisamiseks ja kontrolliga lisamiseks, viimase puhul lisatakse objekt järjekorda vaid juhul kui samasugust objekti seal veel või enam ei ole. Liides *BackgroundTask* nõuab, et teda omavatel klassidel oleks olemas meetod *process()*, milles olev kood täidetaksegi *BackgroundThread* lõimes vastava objekti järjekorrast eemaldamise järel.

Eelkirjeldatud lahenduse korral ei kiirene 2D vaadete ülejoonistamise kiirus, kuid ülejoonistamise ajal jääb ülejäänud kasutajaliides kasutajale reageerivaks ning stseenis sisalduvate tippudega ja stseeni enda vaatepunktiga on võimalik manipuleerida katkestamatult.

Kokkuvõte

Käesoleva bakalaureusetöö põhitulemuseks on seminaritöö käigus valminud rakendusele lisandunud geomeetria redaktor, mis hõlmab endas vastavaid Java 3D struktuure haldavaid klasse, redaktori tööks vajalikke funktsionaalsusi pakkuvaid klasse ning kasutajaliidese komponente, mille abil on need funktsionaalsused kasutajale kättesaadavad. Samuti täienes ka eelnevalt valminud rakenduse osa – komponente ja algoritme sai optimeeritud vastavalt autori täienenud teadmistele ning lisandunud osade

Valminud geomeetria redaktori eesmärgiks on sarnaselt ülejäänud rakenduse osale võimaldada ise luua kolmemõõtmelisi stseene ja selles sisalduvaid objekte või muuta imporditud objektide omadusi vastavalt kasutaja vajadustele ning eksportida saadud tulemus kujule, mis lubaks selle kasutamist teistes Java 3D rakendustes ilma programmeerimist nõudvate täienduste tegemiseta. Erinevus eelnevalt valminud rakenduse osast seisneb selle, et geomeetria redaktor võimaldab pääseda ligi individuaalsete objektide sisule, ning manipuleerida ilma programeerimiseta seal paiknevate andmetega.

Valminud rakendust võib käsitleda kui abivahendit Java 3D rakenduste loomisel või Java 3D põhimõtete tundmaõppimisel, uurides rakenduse ehitust ning ka lihtsalt testides, mis võimaldab visualiseeritud näidete abil enesele selgeks teha, mida ja kuidas mingi Java 3D vahend tegelikult teeb. Samuti saab rakendust kasutada baasrakenduse või prototüübina mahukama ja täielikuma redaktori loomisel.

Summary

Manipulation of Polygonal Meshes with Java 3D

by Risto Seene

The aim of this bachelor's thesis is to study how are polygonal meshes stored in Java 3D classes and how is one able to manipulate that data.

During the writing process of this work the author has studied how is vertex-based geometry represented and rendered in computer graphics in general and how is it done in Java 3D. Gathered knowledge has been used to develop a tool for creating and manipulating geometry in Java 3D. Completed tool – geometry editor – has been integrated into the application called „Java 3D editor", which author had made before.

The output of this work is a basic set of tools for creating three-dimensional scenes and objects which can be used in various Java 3D applications. Creating geometry for 3D scenes is usually done using third-party packages, rather than programming directly in Java, however imported objects usually require some tweaking by programmers. Created tool can be used to either create your own or edit imported 3D content which can be exported and then used directly in other Java 3D applications without further processing.

Kasutatud kirjandus

Bouvier, D., J. (n.d.). Getting Started with the Java 3D API: Textures. Viimati loetud 25. aprill 2009, aadressil http://java.sun.com/javase/technologies/desktop/java3d/collateral/j3d_tutorial_ch7.pdf.

Hook, B. (1995). *Building a 3D Game Engine in C++*. New York: John Wiley & Sons, Inc.

Sabbarton, R. (n.d.). Calculating Face Normals. Viimati loetud 25. aprill 2009, aadressil http://www.fullonsoftware.co.uk/snippets/content/Math_-_Calculating_Face_Normals.pdf.

Sun Microsystems, Inc. (2007). Java 3D API Specification 1.5.1. Viimati loetud 25. aprill 2009, aadressil http://download.java.net/media/java3d/javadoc/1.5.1/index.html.

Terrazas, A., Ostuni, J., Barlow, M. (2002). *Java Media APIs: Cross-Platform Imaging, Media, and Visualization.* Indianapolis: Sams Publishing.

Wagner, M. (2004). Generating Vertex Normals. Viimati loetud 25. aprill 2009, aadressil http://www.emeyex.com/site/tuts/VertexNormals.pdf.

Lisa 1. Rakenduse moodustavad paketid ja klassid

j3deditor.bin – rakenduse baaspakett

- InnerFrame
- MainDesktop
- MainFrame

j3deditor.bin.components – multifunktsionaalsed graafilised komponendid

- ButtonPanelLayout
- CenterLayout
- ComplexButton
- ComponentUtils
- InputDialog
- MultiPanel
- PopupDialog
- SimpleButton
- SimpleButtonEvent
- SimpleButtonListener
- TransparentPanel

j3deditor.bin.editor – stseeni ja geomeetria redaktori põhikomponendid

- Axes2D
- CameraPanel
- EditorFrame
- EditorPanel
- GeometryEditor
- InfoPanel
- ObjectPanel
- Scene2D
- Scene3D
- SceneBuffer
- SceneEditor
- TransformPanel

j3deditor.bin.editor.tools – redaktorite abikomponendid

- AppearancePanel
- ChangeGeometryDialog
- GeometryPanel
- IndexDialog
- LightPanel
- MaterialPanel
- NewGeometryDialog
- NodePanel
- ObjPanel
- TextureAttributesPanel
- TextureDialog
- TexturePanel

UpdateblePanel
ViewPanel2D
ViewPanel3D

j3deditor.bin.editor.util – redaktorite abiklassid

GELayout
GeometryEditorListener
MainLayout
SELayout
VertexManipulator

j3deditor.bin.hierarchy – stseeni-graafi komponente haldavad klassid

GeometryManipulator
HierarchyManager
HierarchyNode
J3DeAppearance
J3DeCamera
J3DeGeometry
J3DeGeometryArray
J3DeImageComponent
J3DeLight
J3DeMaterial
J3DeObject
J3DePrimitive
J3DeScene
J3DeShape3D
J3DeTexture
J3DeTextureAttributes
Vertex
VirtualVertex

j3deditor.bin.hierarchy.util – stseeni-graafi komponentide abiklassid

Copyable
Cuttable
Deletable
HierarchyEvent
HierarchyEventListener
HierarchyWorker
HNodeType
J3DePoint2D
J3DeSerializable
LightType
NormalCalculator
ObjImporter
Pasteable
PrimitiveType
ProjectLoader
ProjectSaver

SGFExporter
SGSExporter
Shape3DType
VirtualCamera
VirtualScene
VirtualScene2D
VirtualScene3D

j3deditor.bin.hierarchy.view – hierarhia halduri komponendid

HierarchyFrame
HierarchyViewer
HTreeModel
TreeRenderer

j3deditor.bin.util – multifunktsionaalsed abikomponendid

BackgroundTask
BackgroundThread
ByteConverter
CancellableTask
Deployer
ImageLoader
Messaging
ProgressChangeListener
ProgressIndicator
Translator
UserData
Utilities
XMLParser

j3deditor.bin.util.preferences – sätete dialoogi komponendid

LangTab
PrefDialog
PrefPanel
ViewTab

j3deditor.bin.util.startup – käivitamisel näidatavat infot kuvavad komponendid

StartPanel
StartUp