VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
KOMPIUTERINIO IR DUOMENŲ MODELIAVIMO KATEDRA

Baigiamasis magistro darbas

3D formatų konvertavimas: iš VRML į WebGL

Atliko:
Egidijus Venckus parašas

Vadovas:
Doc., Dr. Rimvydas Krasauskas

Vilnius
2019

Turinys

Santrauka

Šiame darbe nagrinėjamos interneto naršyklėse naudojamų trimatės kompiuterinės grafikos formatų konvertavimo iš VRML į WebGL galimybės. Pagrindinis tikslas - atlikti VRML formato ir WebGL technologijos analizę, išsirinkti platformą, kuria naudojantis bus programuojama, ir konvertuoti 3D objektų geometrijas, medžiagas, animacijas ir vartotojo navigaciją. Pristatomame darbe buvo atlikta VRML ir WebGL analizė, išsiaiškinti veikimo principai bei technologijų skirtumai. Taip pat apžvelgtos ir kitos technologijos, kuriomis naudojantis galima atvaizduoti trimatę kompiuterinę grafiką interneto naršyklėse. Išnagrinėjus įvairias 3D grafikos kūrimo programas pasirinkta JavaScript kalbą naudojanti Three.js biblioteka, kurios pagalba buvo kuriamas konverteris. Magistro darbe pristatyti algoritmai, kuriais naudojantis galima išnagrinėti į tarpinį X3D formatą konvertuotą VRML failą ir atvaizduoti indeksuotų sienų aibėje aprašytas geometrijas bei tekstūras. Siekiant praktiškai patikrinti ar sukurti algoritmai veikia gerai, iš VRML į WebGL buvo konvertuoti įvairūs 3D objektai, o tarp jų ir sudėtingus indeksuotų sienų aibės mazgus turintys VRML failai - Vilniaus Šv. Nikolajaus cerkvės ir Vilniaus Universiteto kiemelių modeliai. Taip pat įvertinta sukuriamų WebGL objektų įtaka naršyklės greitaveikai ir išsaugomo failo dydžiui. Išnagrinėti VRML ir Three.js bibliotekos animacijų kūrimo modeliai bei pristatyta VRML animacijų konvertavimo į WebGL veiksmų schema. Sukurtas Three.js konverteris buvo palygintas su kitu plačiai naudojamu X3DOM konverteriu. Palygintos algoritmų objektų geometrijos, tekstūrų, spalvų, animacijų ir navigacijos konvertavimo galimybės. Three.js konverteris pasižymėjo geresne greitaveika, platesnėmis įvairių animacijų ir didelių VRML failų konvertavimo galimybėmis.

Summary

3D Format Conversion: VRML to WebGL

VRML (Virtual Reality Modeling Language) is a hierarchical three-dimensional (3D) computer graphic format used in the web browsers that defines the behavior and geometry of 3D scenes. However, while in the past VRML was very popular, now it's not widely used. Currently, the popularity of 3D graphics in the web is very closely related to WebGL technology. WebGL is an application programming interface (API) that uses JavaScript programming language to determine the interactivity of 3D graphics. In this thesis we examined the possibilities of converting three-dimensional computer graphic formats - VRML to WebGL. At first, we analyze the main working principles of VRML format and compared it to WebGL technology in order to choose the platform for software that would be able to convert 3D objects with their geometry, materials, animations and navigation system. We also reviewed other similar technologies that can be used to display 3D graphics on web browsers. After examined various 3D graphics development programs we selected the Three.js library that uses JavaScript language. This library was used to create the converter. In this thesis we shown algorithms that can be used to analyze to mediate X3D file format converted VRML file and display geometries, textures and normals described in indexed face set. In order to verify in practice whether the algorithms that were created are working well, various VRML objects have been converted from VRML to WebGL, including a VRML files containing a complex indexed face set nodes - Vilnius St. Nicholas Church and Vilnius university courtyards. We also evaluated WebGL objects impact to web browser performance and the sizes of the WebGL files saved. After analyzing VRML and Three.js library animation models we presented VRML animation conversion to WebGL action scheme. We also changed standard Three.js library camera control functionality to more accurately convert VRML format navigation system. The proposed Three.js converter has been compared to another widely used X3DOM converter that also have been used to display VRML and X3D files in a web browser. We compared both algorithms geometry's, textures, animation and navigation conversion options. Performance in frames per seconds was measured on various web browsers. The ability to save created WebGL scene to other 3D graphics formats was analyzed as well.

Įvadas

VRML (Virtualios realybės modeliavimo kalba) - hierarchinis trimatės (3D) kompiuterinės grafikos formatas, apibrėžiantis 3D scenos elgesį ir geometriją. Šis formatas pirmą kartą pristatytas 1994 metais ir buvo pirmasis oficialus standartizuotas 3D grafikos formatas naudoti internete. Ypač didelį susidomėjimą šiuo formatu rodė inžinieriai, dizaineriai, mokytojai.

Tačiau nors praeityje VRML ir buvo sulaukęs populiarumo, ši technologija šiuo metu yra nevystoma ir plačiai nenaudojama. Priežasčių kodėl taip nutiko galėjo būti keletas: VRML formatas niekuomet nebuvo įdiegtas interneto naršyklėse. Todėl norint pamatyti VRML 3D vizualizacijas reikėjo įsirašyti specialius įskiepius, o tai galėjo būti nepatogu daugeliui vartotojų. Taip pat šios technologijos užmiršimą galėjo lemti tai, jog ji atsirado tiesiog per anksti. Tuo metu kompiuteriai dar nebuvo pakankamai galingi, buvo labai svarbus failų dydis, nes interneto greitis buvo nedidelis. Todėl modeliuoti ir atvaizduoti sudėtingus ir didelius VRML failus interneto naršyklėse buvo sunku. Vis dėlto VRML formatu yra sukurta daug ir įvairių 3D scenų, kurias atnaujinus į šiuolaikinius formatus galėtume vėl atvaizduoti interneto naršyklėse. VRML konvertavimas taip pat gali būti naudingas, nes leistų nesudėtinga sintakse aprašyti ir atvaizduoti trimatę kompiuterinę grafiką interneto naršyklėse nenaudojant jokių papildomų įskiepių.

Sumažėjus VRML populiarumui, ilgą laiką susidomėjimo 3D grafika žiniatinklyje nebuvo. Vėliau daugelį metų interaktyvus turinys interneto naršyklėse buvo kuriamas naudojant tuo metu dominavusią „Adobe Flash“ technologiją. Tačiau ir ši technologija šiuo metu išgyvena savo išnykimo amžių. Vis mažiau puslapių naudoja šią technologiją ir yra manoma, kad ilgainiui „Adobe Flash“ visiškai pasitrauks iš interneto.

Šiuo metu 3D grafikos populiarumas internete yra itin glaudžiai susijęs su WebGL technologija. WebGL – tai aplikacijų programavimo sąsaja (API), kuri naudodama JavaScript programavimo kalbą, sąlygoja trimatės kompiuterinės grafikos interaktyvumą. WebGL ypač išpopuliarėjo dėl to, kad suteikia galimybę tiesiogiai programuoti grafinį procesorių ir grafinę plokštę turinčiuose stacionariuose ar nešiojamuosiuose kompiuteriuose, mobiliuose įrenginiuose. Be to, visos populiariausios interneto naršyklės palaiko WebGL nereikalaudamos jokių įskiepių ar papildomų programų. Todėl ši technologija kiekvienais metais vis labiau populiarėja.

Šiame darbe panaudota medžiaga iš mokslinio tiriamojo darbo, kuris buvo atliktas ta pačia tema. Panaudota medžiaga - skyriai: "Trimatė kompiuterinė grafika internete", "Naudoti įrankiai ir metodai", "VRML geometrijos ir tekstūros konvertavimas į WebGL".

Šio darbo tikslas yra išnagrinėti VRML formatą ir naujesnę bei daugiau galimybių turinčią WebGL technologiją ir konvertuoti VRML 3D objektų geometriją, medžiagas, animacijas ir vartotojo navigaciją. Taip pat įvertinti sukurto algoritmo greitaveiką, palyginti su X3DOM konverteriu. Taigi, pagrindiniai šio darbo uždaviniai yra:

1. Atlikti šiuo metu galimo VRML konvertavimo į WebGL analizę ir galimybes.
2. Išnagrinėti ir suprasti VRML formatą ir WebGL technologiją.
3. Pasirinkti biblioteką, su kuria bus programuojamas konverteris.
4. Išnagrinėti VRML formato geometrijų, medžiagų, tekstūrų, animacijų ir navigacijos mazgus ir konvertuoti juos į WebGL.
5. Pabandyti konvertuoti į WebGL įvairius VRML failus, tarp kurių ir Vilniuje esančią Šv. Nikolajaus cerkvę, Vilniaus Universiteto kiemelius.
6. Įvertinti WebGL greitaveiką interneto naršyklėje.
7. Palyginti sukurtą Three.js algoritmą su kitu plačiai naudojamu X3DOM konverteriu.

1. Trimatė kompiuterinė grafika internete

Galimybė atvaizduoti trimatę kompiuterinę grafiką interneto naršyklėse atsirado jau seniai. Pirmuosius bandymus galime sieti su 1994 metais pasirodžiusiu VRML formatu. Tačiau tik pastaraisiais metais 3D grafika tampa vis didesne ir svarbesne interneto naršyklių dalimi. Išaugusį susidomėjimą galėjo lemti reiklesni interneto vartotojai bei sudėtingesnės aplikacijos. Vis daugiau programų, kurios anksčiau veikdavo tik kaip darbalaukio programos, yra perkeliamos į interneto naršykles. Tokia kryptis reikalauja galimybės sukurti ir atvaizduoti sudėtingas interaktyvias vizualizacijas internete. Perkelti į interneto naršykles bandoma net ir aukštos kokybės kompiuterinius žaidimus, todėl yra manoma, kad susidomėjimas ir poreikis atkurti trimatę kompiuterinę grafiką interneto naršyklėse ateityje tik didės.

Šiame skyriuje apžvelgsime šiuo metu naudojamus 3D formatus ir technologijas, kurios įgalina interaktyvios trimatės grafikos atvaizdavimą interneto naršyklėse.

1.1. VRML

Virtualios realybės modeliavimo kalba (VRML) yra failo formatas, skirtas kurti interaktyvius 3D objektus ir pasaulius, kurie gali būti atvaizduojami interneto naršyklėse (panašiai kaip HTML formatas yra skirtas atvaizduoti tekstui). Pirmą kartą šis formatas pristatytas 1994 metais, o 1997 metais pasirodė VRML 2.0 versija. Tai buvo pirmasis standartizuotas (ISO/IEC 14772) 3D grafikos standartas, naudojamas interneto naršyklėse [16]. Šis formatas leido vartotojams matyti interaktyvią 3D grafiką - apžiūrėti objektus iš visų pusių, vaikščioti po 3D pasaulį ar net žaisti žaidimus. Norint peržiūrėti VRML failą, reikia naudoti specialias programas, kurios gali interpretuoti ir atvaizduoti VRML failo objektus (BS Contact Stereo, freeWRL web 3D/virtual reality viewer). Taip pat galima naudoti ir interneto naršyklę, tačiau tokiu atveju reikia įdiegti specialų įskiepį.

Visą VRML pasaulį sudaro objektai, kurie vadinami mazgais (nodes), ir savybės, kurios vadinamos laukais (fields). Laukai gali reikšti bet ką, pradedant nuo tam tikro daikto matmenų iki mazgo, kuris yra pirmojo viduje. Kiekvienas laukas turi tipą, nurodantį kokius duomenis jis gali turėti savyje. Yra keletas būdų sukurti VRML failą. Galima naudoti tam pritaikytus 3D modeliavimo paketus ir jų konverterius. Kitas būdas - paprasčiausiai parašyti visą VRML formatą tekstiniu redaktoriumi ir išsaugoti kaip .wrl (nuo anglų kalbos žodžio "world") arba .wrz (zip kompresuotiems failams) failą. Jeigu VRML faile nėra nurodoma kokio nors lauko reikšmė, jam suteikiama numatyta (default) reikšmė. Dėl šios priežasties sukurti paprastą kubą 3D erdvėje tereikia vos 10-15 teksto eilučių. Didžioji dauguma reikšmių įgyja numatytąsias vertes, ir jų atskirai nurodyti nereikia. VRML failo struktūra yra hierarchinė ir ją sudaro:

- Antraštė, kuri parodo naršyklei, kad failas yra VRML ir nurodo jo versiją. Antraštė yra privaloma.
- Komentarai, kurie išskiriami ženklu # ir kompiliatoriaus yra ignoruojami.
- Mazgai.
- Laukai - mazgų atributai, kuriuos galima keisti.
- Atributų vertės.

1 pav. vaizduojamas paprasčiausias VRML sintaksės pavyzdys - kubas. Visi VRML failai prasideda standartine pirmąja eilute - *#VRML V2.0 utf8*. Tai yra pranešimas naršyklei, kad dirbame su VRML failu, kuris naudoja versiją 2.0. *utf8* reiškia, kad naudojame UTF-8 teksto kodavimą. Kaip matyti iš pavyzdžio, VRML failo struktūra yra hierarchinė - kiekvienas mazgas turi pradžią

ir pabaigą, o viduje gali būti laukai arba dar vieni mazgai. Visa ši mazgų visuma vadinama *scenos grafu* (scene graph) [16].

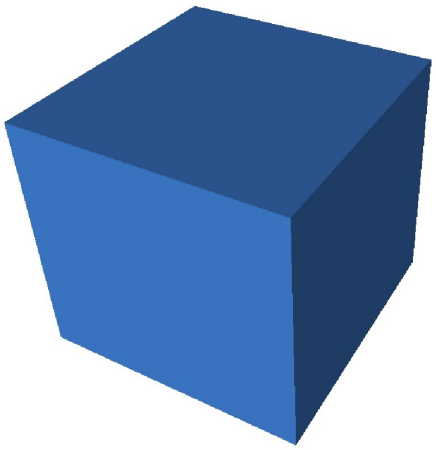
VRML pasižymi ne tik išsišakojusio medžio struktūra, bet ir paveldėjimu. Naudojant *DEF* raktažodį galima suteikti mazgams vardus. O vėliau naudojant komandą *USE* juos iš naujo panaudoti. Toks funkcionalumas leidžia sutaupyti laiko modeliuojant, o VRML failas užima mažiau vietos. VRML naudoja standartinę Dekarto (*x y z*) koordinačių sistemą, o visi atstumai matuojami metrais. Kampai matuojami radianais [23].

```
#VRML V2.0 utf8

NavigationInfo { type "EXAMINE"  "ANY" }

# čia yra komentaras

Shape {
   geometry Box {
                  size 1 1 1
   }
   appearance Appearance {
      material Material {
         diffuseColor .3  .6 1
      }
   }
}
```



1 pav. VRML failo ir sugeneruoto 3D objekto pavyzdys

1 pav. pavyzdyje *Shape*, *Appearance*, *Material*, *Box* yra mazgai, o *material*, *geometry*, *size* ir *diffuseColor* yra laukai. Visa kita - laukų reikšmės. Dažniausiai VRML failai yra didesni ir gali talpinti tūkstančius eilučių. Internete galima rasti daug nemokamų VRML failų saugyklų įvairiomis temomis.

VRML turi daug ypatybių, tačiau pagrindinės ir svarbiausios mums galėtų būti šios:

- Žiūrėjimo taškai (Viewpoints) - nustatytos kameros pozicijos.
- Transformacijos (Transform) - postūmis (translation), posūkis (rotation) ir tempimas (scale).
- Geometriniai mazgai - dėžė (box), cilindras (cylinder), kūgis (cone), sfera (sphere), taškų aibės (point set), indeksuotų tiesių aibės (indexed line set), indeksuotų sienų aibės (indexed face set), aukščių gardelės (elevation grid), tekstai.
- Medžiagos (Materials) - difuzinės spalvos (difuseColor), emisijinės spalvos (emissiveColor), veidrodinis atspindys (specular), šviesos intensyvumas (ambientIntensity), žvilgėjimas (shininess), permatomumas (transparency).
- Apšvietimas (Lighting) - kryptinė (directional), taškinė (point) ir vietinė (spot) šviesa.
- Tekstūros - VRML palaiko PNG, GIF, JPEG, ir MPEG video formatus.
- Animacijos - pozicijos, deformacijų, posūkių ir kt.
- Įvykių jutikliai (Events) - laiko, lietimo, judesio, matomumo, tempimo ir artumo.
- Skriptai (Scripting) - galimybė naudoti JavaScript, Java ir VRMLScript kalbas.

VRML formatas turi 54 skirtingus mazgus, kuriais naudojantis galima modeliuoti 3D pasaulį. O laukai (savybės), kurie gali būti panaudoti pasirinktam mazgui, priklauso nuo kiekvieno mazgo individualiai. Vieni mazgai turi tik 1 lauką, o kitiems galima panaudoti netgi 14 [6]. Todėl modeliuojant 3D pasaulį su VRML formatu pasirinkimas yra labai didelis. Verta paminėti,

kad standartiniais VRML mazgais nebūtina apsiriboti. Galima kurti ir naujus mazgus naudojant prototipus (Prototypes).

1.2. X3D

2001 metais Web3D.org pristatė X3D formatą, kuris yra VRML variantas pritaikytas XML kalbai. Kaip ir VRML, X3D yra ISO standartas (ISO/IEC-19775-1). Tačiau ir šis formatas niekuomet nebuvo įtrauktas į interneto naršykles, nors ir buvo patogesnis naudoti. Šiuo metu X3D yra laikomas paskutine VRML formato versija su XML sintakse. Kadangi teoriškai VRML ir X3D formatai yra tokie patys, skiriasi tik mazgų ir laukų aprašymo stilius, šie formatai labai lengvai konvertuojami iš vieno į kitą. Nemokamą šių formatų konverterį galima rasti internete [11]. Šiame darbe visi VRML failai prieš pradedant su jais dirbti yra konvertuojami į X3D formatą. Tai yra daroma dėl kelių priežasčių:

- Konvertavimo algoritmui analizuojant VRML failą yra sunku interpretuoti ir suprasti kur baigiasi ir prasideda kiti susiję laukai ar mazgai.
- XML struktūros faile lengviau suprasti kur yra kiekvieno mazgo ar lauko pradžia ir pabaiga, kokios yra laukų vertės.
- XML failą paprasčiau nagrinėti, nes galime pasinaudoti jau egzistuojančiomis DOM funkcijomis, pavyzdžiui DOMParser.

X3D formato sintaksė pavaizduota 2 pav. Sukuriama ta pati mėlyna dėžė, kuri yra pavaizduota ir 1 pav. Kaip matyti iš šio pavyzdžio, X3D formatas yra ta pati VRML kalba, tik parašyta patogesne XML sintakse. Mazgų, laukų ir jų reikšmių vertės yra vienodos. O visas tekstas patalpintas *<X3D/>* elemente. X3D failai saugomi su galūne .x3d (**e**Xtensible **3D** Graphics). Nors šiame darbe ir dirbsime su X3D formatu, visus likusius pavyzdžius vaizduosime VRML formatu. Tai bus daroma dėl lengvesnės ir žmogui labiau suprantamesnės VRML sintaksės [22].

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D>
 <Scene DEF='scene'>
  <Shape>
   <Appearance>
    <Material diffuseColor='0.3 0.6 1'/>
   </Appearance>
   <Box size='1 1 1'/>
  </Shape>
 </Scene>
</X3D>
```

2 pav. X3D formato pavyzdys - mėlyna dėžė

1.3. WebGL

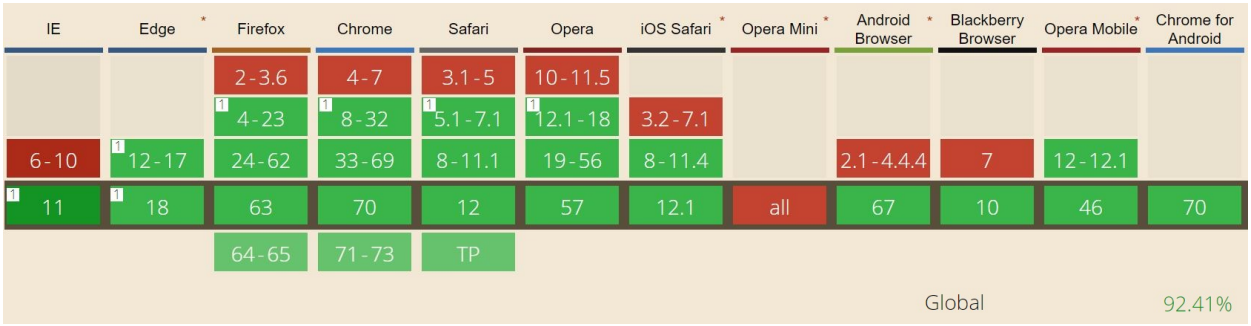
WebGL (Web-based Graphics Library) – aplikacijų programavimo sąsaja (API), veikianti JavaScript pagrindu ir suteikianti galimybę atvaizduoti trimatę ir dvimatę grafiką interneto naršyklėse be jokių papildomų įskiepių. WebGL technologija paremta OpenGL ES 2.0. ir 3D grafikos atvaizdavimui naudoja HTML5 <canvas> elementą. 2011 m. kovo 3 d. buvo pristatyta pirmoji

WebGL versija 1.0. WebGL yra prižiūrima „Khronos Group“, kuri taip pat kontroliuoja ir kitus atviro kodo OpenGL (Open Graphics Library) projektus.

Šiuo metu WebGL yra viena iš populiariausių trimatės grafikos atvaizdavimo priemonių tinklapiuose. Tai lėmė kelios priežastys:

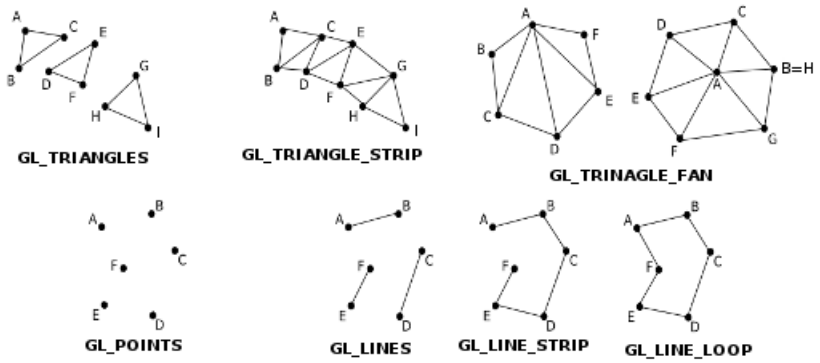
- WebGL suteikia galimybę tiesiogiai programuoti grafinį procesorių tiek grafinę plokštę turinčiuose stacionariuose ar nešiojamuosiuose kompiuteriuose, tiek mobiliuose įrenginiuose naudojant JavaScript ir GLSL programavimo kalbas.
- WebGL pasižymi didele 3D grafikos atkūrimo sparta ir galimybe sukurti įspūdingas 3D grafikas interneto naršyklėse.
- WebGL yra dalis HTML5 šeimos technologijos. Todėl atvaizduoti ir naudoti WebGL galima visose naršyklėse tiek stacionariuose kompiuteriuose, tiek mobiliuose įrenginiuose palaikančiuose HTML5 ir OpenGL ES 2.0.
- WebGL yra nuolatos tobulinamas ir vystomas.

WebGL populiarumas kiekvienais metais vis didėja, o šią technologiją palaikančių naršyklių skaičius vis auga. Pagrindinė WebGL populiarumo priežastis yra papildomų įskiepių ar programų atsisakymas. Tai yra didžiulis privalumas lyginant šią technologiją su VRML formatu. 3 pav. vaizduojamas dabartinis WebGL paplitimas žiniatinkliuose. Kaip matyti iš paveiksliuko, WebGL palaiko beveik visos populiariausios interneto naršyklės ir jų versijos. Šiuo darbo rašymo metu šis procentas siekia 92,41 % [12]. Palyginimui 2016 metais tokių naršyklių buvo 85 %, o 2014 metais palaikymas siekė tik 65 %. Žaliai pažymėtos naršyklių versijos palaiko WebGL technologiją. Raudonai pažymėtos versijos nepalaiko.



3 pav. WebGL palaikančios naršyklės [12].

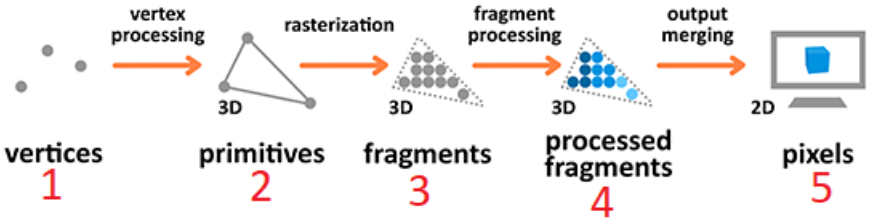
WebGL galime apibūdinti kaip rasterizavimo programą, kuri gali piešti taškus, linijas arba trikampius priklausomai nuo programuotojo parašyto kodo. Visas 3D objektų kūrimas yra paremtas dvejomis GLSL (OpenGL Shading Language) programavimo kalba parašytomis funkcijomis, kurios vadinasi *viršūnių šešėliuoklė* (Vertex Shader) ir *fragmentų šešėliuoklė* (Fragment Shader arba Pixel Shader). Viršūnių šešėliuoklė yra kviečiama kiekvienai nurodytai viršūnei (vertex), ir jos paskirtis yra viršūnių transformacijos, tekstūrų koordinačių, normalių apskaičiavimas ir kiti su būsima objekto geometrija susiję skaičiavimai. Geometrijos gali būti kuriamos pasirenkant iš 7 galimų primityvų, kurie pavaizduoti 4 pav. Fragmentų šešėliuoklė yra kviečiama kiekvienam ekrano pikseliui. Šios funkcijos paskirtis yra nuspalvinti kiekvieną primityvo pikselį pagal spalvos, apšvietimo, medžiagos ir kitus parametrus [20].



4 pav. WebGL naudojami primityvai - trikampiai, linijos ir taškai [7].

Supaprastinta WebGL veikimo schema pavaizduota 5 pav. Žingsniai, kurios reikia atlikti norint atvaizduoti ekrane paprasčiausią trikampį, yra:

1. Sukuriamas HTML <canvas> elementas ir naudojantis juo inicijuojamas WebGL atvaizdavimo kontekstas.
2. Trikampio geometrijos viršūnės nurodomos naudojant WebGL primityvus ir išsaugomos kintamajame (1 žingsnis).
3. Jeigu trikampio geometrija turės būti mažinama, didinama, sukama ar kitaip veikiama - sukuriamos transformavimo matricos.
4. Sukuriamos viršūnių ir fragmentų šešėliuoklės. Su šia funkcijų pora sukuriama programa, kurios paskirtis kreiptis į grafinį procesorių. Sukurta programa sukompiliuojama.
5. Viršūnes saugantis kintamasis susiejamas su viršūnių ir fragmentų programa.
6. Naudojantis viršūnių šešėliuokle, geometrija atvaizduojama per WebGL primityvus (2 žingsnis).
7. Sukurti primityvai rasterizuojami. Atrenkami pikseliai, kurie patenka į primityvus (3 žingsnis).
8. Kiekvienam į primityvą patenkančiam pikseliui iškviečiama fragmentų šešėliuoklė, kuri priklausomai nuo parametrų parenka pikselio spalvą (4 žingsnis).
9. Iškviečiamos specialios piešimo funkcijos, kartu nurodant kurį primityvą naudosime. Trikampis atvaizduojamas ekrane (5 žingsnis).



5 pav. Supaprastinta WebGL veikimo schema [2].

Kaip matyti iš pateikto pavyzdžio, WebGL naudojama žemesnio lygio sintaksės programavimo kalba, todėl atvaizduoti trimatę kompiuterinę grafiką yra sudėtinga, reikalauja žymiai daugiau rankinio darbo bei kodo eilučių nei anksčiau minėti VRML ir X3D formatai. Pavyzdžiui atvaizduoti kubą, kuris rodomas 1 pav. naudojant VRML ar X3D formatą tereikia teksto redaktoriumi parašyti 10 - 15 eilučių. Tuo tarpu gauti panašų rezultatą naudojant vien tik WebGL API reikėtų parašyti 200-250 kodo eilučių! Todėl dažniausiai trimatės kompiuterinės grafikos niekas nekuria vien tik su WebGL technologija. Tam yra pasitelkiamos specialios bibliotekos, kurios atlieka sudėtingus

matematinius skaičiavimus už mus. Naudojant bibliotekas atvaizduoti tą patį kubą gali prireikti tik 30-40 kodo eilučių [20]. Papildomas bibliotekas naudoti yra naudinga, nes jos automatiškai apskaičiuoja sudėtingas matricas, projekcijas ir viršūnių koordinates, taip išvengiant didelės klaidų tikimybės. Viena iš populiariausių ir labiausiai vystomų WebGL bibliotekų šiuo metu yra Three.js. Todėl programuojant VRML į WebGL konverterį naudosime šią biblioteką, kuri bus aptarta 2 skyriuje.

1.4. CSS

CCS (Cascading Style Sheets - pakopiniai stilių šablonai) - kalba, skirta nusakyti kita struktūrine kalba aprašyto dokumento vaizdavimą ir dažniausiai naudojama internetiniuose puslapiuose kuriant HTML dokumentus. *CSS 3D transformacijos* yra vienas iš CCS kalbos papildymų, kuris buvo sukurtas Apple kompanijos ir pradžioje naudotas *Safari* interneto naršyklėje. Vėliau išpopuliarėjo ir šiuo metu yra palaikomas visose populiariausiose interneto naršyklėse. *CSS 3D transformacijos* leidžia sukurtiems elementams panaudoti standartines 3D naudojamas transformacijas, tokias kaip postūmis, sukimas ar tempimas. Transformacijos pasižymi dideliu greičiu, nesudėtingu naudojimu ir leidžia sukurti paprasčiausias 3D animacijas ir efektus. Pavyzdžiui erdvėje besisukantis ir į vartotojo paspaudimus reaguojantis kubas. Vis dėlto CSS 3D grafikos galimybės yra labai ribotos, todėl ši kalba yra naudojama tik paprasčiausioms trimatėms kompiuterinėms grafikoms kurti [18].

1.5. Stage3D

Stage3D - *Adobe Flash* aplikacijų programavimo sąsaja, naudojanti ActionScript programavimo kalbą ir leidžianti interneto naršyklėse atvaizduoti interaktyvią 3D grafiką. Savo paskirtimi ir veikimu Stage3D yra panaši į WebGL. Tai žemo lygio programavimo kalba, kuri trimatės kompiuterinės grafikos sukūrimui naudoja objektų viršūnių koordinates, šešėliuoklių programas bei transformacijų matricas. Šešėliuoklės, skirtingai nuo WebGL naudojamos GLSL kalbos, yra kuriamos specialia AGAL (Adobe Graphics Assembly Language) kalba [18]. Tiesioginis priėjimas prie įrenginio grafinio procesoriaus leidžia Stage3D pasiekti didelę 3D grafikos atkūrimo spartą bei aukštos kokybės grafiką. Kadangi tai yra žemo lygio programavimo kalba, šiam API taip pat yra sukurta papildomų bibliotekų, kurios palengvina programuotojų darbą. Norint panaudoti Stage3D interneto naršyklėje, į ją turi būti įdiegiamas specialus įskiepis. Dėl šios priežasties ši technologija nėra tokia populiari kaip WebGL.

1.6. JavaFX

JavaFX yra bibliotekų rinkinys, skirtas kurti interaktyvią vartotojo grafinę sąsają. JavaFX sukūrė kompanija *Sun Microsystems*, kaip atsaką panašaus veikimo *Adobe Flash* ir *Microsoft Silverlight* technologijoms. Aplikacijos, sukurtos naudojant JavaFX gali būti vykdomos stacionariuose kompiuteriuose, mobiliuose telefonuose, planšetėse, televizoriuose ar interneto naršyklėse. Norint naudoti šią biblioteką interneto naršyklėje, į ją turi būti įdiegiamas įskiepis. Kaip ir dauguma technologijų, kurios reikalauja specialaus įskiepio, JavaFX nėra plačiai naudojamas sprendimas norint atvaizduoti trimatę kompiuterinę grafiką interneto naršyklėse [25] [13].

2. Naudoti įrankiai ir metodai

Šiame skyriuje apžvelgiama Three.js biblioteka, kuri bus naudojama norint konvertuoti VRML formato failą į WebGL. Taip pat išanalizuotos ir kitos 3D grafikos kūrimo programos, kurios iš dalies gali konvertuoti VRML ar X3D formato failus į WebGL.

2.1. Three.js

Neturėjo praeiti daug laiko, kol kažkas kurdamas WebGL 3D objektus, ir kiekvieną kartą rašydamas sudėtingas matricas, vektorius, primityvų pozicijas, imtų ir sukurtų biblioteką, kuri automatiškai atliktų ir palengvintų visus šiuos sudėtingus bei nuolat pasikartojančius skaičiavimus. Šiuo metu egzistuoja ne viena JavaScript biblioteka, palengvinanti WebGL programavimą. Vienos iš populiariausių ir dažniausiai naudojamų yra GLGE (http://www.glge.org/), SceneJS (http://www.scenejs.org/), ar babylon.js (www.babylonjs.com). Kiekviena iš minėtų bibliotekų veikia kiek skirtingai, tačiau visų jų bendras tikslas ir paskirtis yra ta pati - palengvinti WebGL technologijos realizavimą, leidžiant aprašyti objektus aukštesnio lygio sintakse [20].

Programuojant VRML į WebGL konverterį šiame darbe buvo pasirinkta naudoti Three.js biblioteka, sukurta Ricardo Cabello - programuotojo iš Ispanijos, Barselonos dar žinomo kaip Mr.doob pseudonimu. Pirma šios bibliotekos versija pasirodė 2010 metais [17]. Visas Three.js programinis kodas yra patalpintas GitHub svetainės talpykloje. Šiuo metu daugiau nei 900 žmonių prisideda prie šios bibliotekos nuolatinio tobulinimo bei įvairių klaidų taisymo. Three.js bibliotekai suteikta MIT licencija - bet kas gali ją parsisiųsti, keisti ir naudoti savo tikslams visiškai nemokamai [5].

Priežastys, kodėl konverterio kūrimui buvo pasirinkta naudoti Three.js biblioteka:

- Three.js yra viena iš populiariausių WebGL bibliotekų, kurią naudoja daugybė WebGL programuotojų. Ši biblioteka gerai dokumentuota ir yra nuolat tobulinama.
- Three.js sukurta naudojantis geriausiomis programavimo praktikomis, todėl pasižymi dideliu greičiu ir stabilumu.
- Three.js turi daugybę funkcijų, palengvinančių animacijų ir sudėtingų geometrijų kūrimą. Tai bus svarbu konvertuojant sudėtingus VRML failus.
- Three.js leidžia išeksportuoti sukurtus WebGL objektus į *.glTF*, *.json*, *.obj*, *.glb* ar *.stl* failus. Vėliau šiuos failus galima vėl atkurti į WebGL objektus.
- Three.js yra nemokama, o pagrindinis bibliotekos failas užima nedaug vietos, todėl yra patogus naudoti interneto naršyklėse.
- Norint naudotis sukurtu konverteriu nereikės diegtis jokios programos ar naudotis papildomais įskiepiais.
- VRML/X3D failas bus konvertuojamas į WebGL formatą iškart, be jokių tarpinių formatų.

Šiuo metu Three.js bibliotekoje egzistuoja konverteris iš VRML formato į WebGL. Tačiau bandant importuoti įvairius VRML failus ir analizuojant programinį kodą paaiškėjo, kad šis konverteris yra labai ribotas ir mūsų netenkina, nes:

- Konverteryje kiekviena VRML eilutė nagrinėjama atskirai. Todėl bet koks neatitikimas failo eilutėje ar laukų vertėse priverčia algoritmą sustoti. Nors VRML peržiūros programos tokį failą vis tiek lengvai nuskaito.
- Nepalaikomos tekstūros, normalės, animacijos, vartotojo navigacija, įvykių jutikliai.
- Nepalaikomos svarbios VRML geometrijos - taškų aibės, indeksuotų tiesių aibės, aukščių gardelės, tekstai.
- Neatsižvelgta į numatytąsias VRML failo vertes. Todėl geometrijos ir kiti mazgai, kurie naudoja numatytąsias vertes atvaizduojami netiksliai arba neatvaizduojami visai.

- Įvairios kitos problemos (spalvos, 'url' laukas ir t.t).

Dabartinis Three.js konverteris gali importuoti tik paprasčiausius ir nesudėtingus VRML failus. Be to šiame darbe bus importuojamas ne VRML failas, o X3D failas, kuris šiame darbe yra tiesiog VRML failas parašytas patogesniu XML formatu. Three.js biblioteka darbo rašymo metu neturėjo jokio X3D failų konverterio, todėl jis bus kuriamas kaip visiškai naujas sprendimas, panaudojant tam tikras Three.js esančio VRML konverterio savybes.

2.2. Blender

Blender yra viena iš populiariausių nemokamų 3D grafikos kūrimo programų. Taip pat tai viena iš nedaugelio sistemų, kuri palaiko VRML failų importavimą. Tai galėjo lemti faktas, kad Blender buvo sukurta 1995 metais, ir nuo to laiko nuolat tobulinama [19]. Kaip tik tuo metu VRML buvo labai populiarus, todėl natūralu, kad Blender buvo implementuota VRML failų importavimo galimybė. Vis dėlto VRML failo konvertavimas į WebGL naudojant Blender programą mūsų netenkino, nes:

- Importuojami VRML failai yra be tekstūrų, o importuojant sudėtingesnius failus programa dažnai išmeta klaidas.
- Išeksportavus VRML objektus į kitus formatus, šie dažnai būna su trūkumais (dingsta geometrijos, atsiranda neatitikimai)
- Nepalaikomos VRML animacijos, įvykių jutikliai.
- Blender turi ribotas galimybes išeksportuoti suimportuotus objektus į WebGL.
- VRML failas konvertuojamas į WebGL formatą ne iškart, reikalauja papildomo darbo bei programų diegimo.

2.3. X3DOM

2009 metais buvo pristatyta X3DOM biblioteka, kuri leido paprasčiau atvaizduoti X3D failus interneto naršyklėse. Naudojantis šia biblioteka, X3D formato tekstas gali būti parašomas tiesiai į HTML dokumentą, o trimatė grafika interneto naršyklėje sukuriama naudojantis WebGL [15]. Jeigu WebGL nepalaikomas, tikrinama ar naršyklėje yra įdiegti InstantReality arba Flash11 įskiepiai, ir tuomet 3D vaizdas atkuriamas naudojantis šiomis programomis. X3D failo tekstas nurodomas specialiame HTML failo <x3D/> elemente.

Naudojantis šia biblioteka, VRML konvertavimas į WebGL gali atrodyti labai paprastas. Konvertuojame VRML failą į X3D formatą naudojantis jau sukurtais konverteriais. Gautą X3D failo tekstą įkopijuojame į HTML failo <x3D/> elementą ir panaudodami X3DOM biblioteką atvaizduojame 3D objektą jau naudodami WebGL. Tačiau šis konvertavimo kelias ir X3DOM biblioteka nebuvo pasirinkta dėl keleto priežasčių:

- Bandant atvaizduoti sudėtingą X3D failą, biblioteka išmesdavo klaidos pranešimus. Tai gali reikšti, kad ši biblioteka nepalaiko svarbių X3D formato galimybių.
- Konvertavus VRML failą su populiariausiomis animacijomis, ne visi mazgai buvo atvaizduojami.
- X3DOM trūksta galimybių išeksportuoti sukurtą WebGL objektą į kitus formatus, todėl naudojantis šia biblioteka nebūtų galimybės paprastai išsaugoti sukurto WebGL objekto.
- Biblioteka nėra tokia populiari ir taip gerai dokumentuota, kaip Three.js. Todėl rasti reikalingos informacijos bibliotekos tobulinimui gali būti sunku.

2.4. 3ds Max

3ds Max yra labai daug galimybių turinti 3D grafikos modeliavo programa, gebanti importuoti ir senus VRML failus. Nors ši programa yra mokama, tam tikrą šios programos versiją galima išbandyti nemokamai. Bandant importuoti VRML failus buvo susiduriama su panašiomis problemomis, kaip ir su Blender programa. Vis dėlto verta paminėti, kad 3ds Max sugebėdavo atvaizduoti VRML animacijas. Tačiau programa nuolatos nustodavo veikti bandant suimportuoti didelius ir sudėtingus VRML failus ar objektus su iškėlimo geometrijomis. Kitas didelis šios programos minusas yra tai, kad ji standartiškai nepalaiko eksportavimo į WebGL. Norint turėti tokią galimybę, reikia pirkti papildomus įskiepius. Taip pat ši programa užima daug vietos ir reikalauja galingesnio kompiuterio.

2.5. Kitos programos

2014 metais pasirodė straipsnis, kuriame pristatomos metodikos ir aplikacija, skirta VRML į WebGL konvertavimui [26]. Autorių pateiktuose pavyzdžiuose sukurta aplikacija konvertavo tik paprasčiausias geometrijas, o didžioji straipsnio dalis buvo skirta algoritmo greitaveikos įvertinimui skirtingose konvertavimo algoritmo stadijose. Straipsnyje nebuvo rašoma apie tekstūrų, navigacijos ar animacijų konvertavimą. Todėl sunku įsivaizduoti šios sukurtos aplikacijos galimybes. Tolesnių šių autorių darbų VRML tema nepavyko rasti, todėl galima teigti, kad straipsnyje pristatyta aplikacija galėjo konvertuoti tik paprasčiausius VRML 3D failus.

Galima rasti ir daugiau 3D grafikos modeliavo programų, kurios gali importuoti VRML failus. Pvz. Rhino, SketcUp ar Solidworks. Šios programos palaiko VRML importavimo galimybę, tačiau nebuvo išbandytos, nes yra mokamos, galinčios kainuoti net iki kelių tūkstančių dolerių [14]. Taip pat šios programos gali stokoti eksportavimo į WebGL galimybių.

Verta paminėti ir Clara.io programą, kuri turi VRML importavimo galimybes, o ja naudotis tereikia interneto naršyklės. Tačiau Clara.io VRML importavimo galimybės yra labai ribotos. Tai gali lemti faktas, kad ši programa naudoja Three.js biblioteką ir jos VRML konverterį, kuris kaip buvo minėta anksčiau, turi daug trūkumų.

3. VRML geometrijos ir tekstūros konvertavimas į WebGL

Literatūros analizė parodė, kad šiuo metu jau egzistuoja tam tikri įrankiai, kurie gali konvertuoti VRML arba X3D formatą į WebGL. Tačiau išbandžius šiuos įrankius su testiniais VRML failais, buvo susidurta su problemomis. Dideli ir sudėtingi VRML failai nebuvo konvertuojami visai, o paprastesniuose VRML failuose nekonvertuojamos tekstūros, animacijos. Todėl galime teigti, jog šiuo metu dar nėra sukurto universalaus ir paprasto būdo atvaizduoti VRML formato failus interneto naršyklėse naudojantis WebGL technologija.

Tokio konverterio poreikis gali atsirasti netolimoje ateityje. Interneto vartotojai tampa vis reiklesni, o internetiniai puslapiai vis sudėtingesni. Augantys poreikiai gali reikalauti galimybės atvaizduoti internete auštos kokybės trimatę kompiuterinę grafiką. Tai galima pasiekti su WebGL, kuri šiuo metu palaikoma beveik visose interneto naršyklėse. Todėl vis daugiau programuotojų pradeda naudoti WebGL technologiją, nebijodami, kad interneto vartotojai negalės matyti 3D grafikos. Tačiau WebGL naudoja labai žemo lygio programavimo kalbą, kuria norint naudotis reikia turėti gilių žinių apie esminius su 3D grafika susijusius elementus - matricas, šėšėliuoklių programas, vektorių matematiką ir grafinio procesoriaus veikimo principus. Šią situaciją gerina nemokamos bibliotekos, tokios kaip Three.js, tačiau ir jomis naudotis nėra taip paprasta. Norint internetiniame puslapyje atvaizduoti paprastą 3D kubą, naudojant tik WebGL reikėtų parašyti 200-250 kodo eilučių. Tas pas rezultatas naudojantis Three.js biblioteka gali būti pasiekiamas su 30-40 eilučių. Tačiau situaciją gali dar labiau palengvinti VRML formatas, kuriuo sukurti tą patį kubą galima vos su 10 teksto eilučių. Be to VRML formato sintaksė žmogui yra labai paprasta ir suprantama. Todėl siekiant interneto puslapyje atvaizduoti interaktyvią trimatę grafiką gali reikėti parašyti paprastą VRML arba X3D formato tekstą ir panaudoti konverterį. Tai dar labiau palengvintų ir sumažintų reikalingą atlikti programavimo darbą. VRML konverterio poreikis taip pat gali atsirasti dėl siekio atgaivinti senus VRML ar X3D failus ir juos iš naujo panaudoti internetiniuose puslapiuose.

WebGL turi daugiau funkcijų ir pasižymi didesnėmis galimybėmis nei konvertuojami VRML ir X3D formatai. VRML neturėjo galimybės atvaizduoti atspindžius arba šešėlius. Taip pat trūko galimybių sukurti sudėtingas animacijas ir vartotojo interaktyvumą. Tai tik kelios iš daugelio savybių, kurios dabar yra įmanomos su WebGL technologija.

Toliau šiame skyriuje apžvelgiami sukurti algoritmai, kuriais naudodamiesi išanalizuosime vieną iš pagrindinių VRML mazgų - indeksuotų sienų aibę. VRML failą konvertuosime į WebGL objektą, kuris gali būti peržiūrimas interneto naršyklėje ir išeksportuojamas į kitus WebGL palaikomus formatus. Taip pat šiame skyriuje aprašysime atliktus bandymus ir gautus rezultatus. Darbe naudojamų VRML failų šaltiniai pateikti E priede.

3.1. X3D failo analizė

VRML failai, kuriuos norime konvertuoti į WebGL formatą, pirmiausia konvertuojami į tarpinį X3D formatą. Nemokamą VRML/X3D konverterį galima rasti internete [11]. Tarpinis konvertavimas atliekamas, nes X3D formatas turi XML struktūrą, kurią lengviau analizuoti ir nagrinėti.

Failo konvertavimo į WebGL veiksmų diagrama pateikta C priede. Elementai nagrinėjami naudojantis standartinėmis naršyklėse esančiomis DOM (**D**ocument **O**bject **M**odel) funkcijomis. Trumpai paaiškinsime failo skaitymo algoritmą:

- 1. Pirmiausia sukuriama Three.js scena kurioje bus talpinami visi sukurti 3D objektai.
- 2. Iškviečiama konvertavimo algoritmo funkcija, kuriai perduodama X3D failo nuoroda.

- 3. Sukuriamas hierarchinės struktūros JSON objektas. XML failas nagrinėjamas panaudojant rekursinę funkciją, kurios pagalba išnagrinėjami visi elementai. JSON objektas kuriamas dėl patogesnio XML atributų išsaugojimo. Pavyzdžiui spalvos iš XML į JSON perrašomos kaip RGB spalvų lentelės reikšmės, o viršūnių padėtys - kaip dvimačiai masyvai.
- 4. Rekursinei funkcijai baigus darbą, pradedama vykdyti dar viena rekursinė funkcija. Jos pagalba analizuojamas JSON objektas.
- 5. Pagal elemento tipą, kuris sutampa su VRML mazgo pavadinimu, priskiriamos numatytosios VRML vertės. Šios vertės objektui priskiriamos tik tuomet, jeigu nebuvo nurodytos konvertuojamame faile.
- 6-13. Pagal elemento tipą sukuriami įvairūs 3D objektai - medžiagos, geometrijos, kamera, animacijos. Objektas yra pridedamas prie tėvinio Three.js elemento, taip sukuriant hierarchinę 3D scenos struktūrą.
- 14. Jeigu nagrinėjamame faile nebuvo nurodyta, sukuriame numatytąją kamerą, šviesas, navigacijos informaciją. Per naujo peržiūrime maršrutus, kurių nepavyko panaudoti animacijos konvertavimo algoritme.
- 15. Konverterio sukurtą objektą pridedame prie algoritmo pradžioje sukurtos scenos. Algoritmas baigia darbą ir vartotojas interneto naršyklėje gali peržiūrėti konvertuotą failą.

Three.js turi VRML failų konverterį, tačiau kaip buvo minėta anksčiau, jis yra labai ribotas ir mūsų netenkino. X3D failų konvertavimo galimybės Three.js biblioteka neturi, todėl jis buvo kuriamas naujas, pritaikius VRML konverterio savybes. Gerai veikiančios VRML konvertavimo algoritmo savybės buvo perkeltos į kuriamą X3D konverterį, ir pritaikytos XML struktūros failui. Pagrindinės perkeltos ir gerai veikiančios konvertavimo galimybės:

- Paprasčiausios geometrijos konvertavimas (dėžė, sfera, cilindras ir t.t)
- Objektus supančio fono ir šviesos sukūrimas.
- Postūmio, posūkių, tempimo transformacijų ir spalvų konvertavimas.
- Pagrindinių VRML laukų ir mazgų paveldėjimo galimybės.
- Nesudėtingų indeksuotų sienų aibių geometrijos konvertavimas.

Sukurtos naujos konvertavimo galimybės:

- XML struktūros X3D failo nuskaitymas ir perrašymas į JSON failą.
- Indeksuotų tiesių aibių geometrijos konvertavimas.
- Papildytas indeksuotų sienų aibių geometrijos konvertavimas (konvertuojamos ir normalės).
- Tekstūrų konvertavimas.
- Aukščio gardelės, ekstruzijos, taškų, teksto geometrijos konvertavimas.
- Numatytųjų reikšmių objektams priskyrimas.
- Praplėstos VRML laukų ir mazgų paveldėjimo galimybės.
- Animacijų, sukurtų naudojant laiko, plokštumos, paspaudimo sensorius ir interpoliatorių konvertavimas.
- Peržiūros taškų konvertavimas, animuojant kameros perėjimą iš vieno taško į kitą.
- Papildyta sukurtos 3D scenos peržiūros navigacija.

3.2. Indeksuotų sienų aibė

Indeksuotų sienų aibė (IndexedFaceSet = IFS) - tai aibė sienų, kurių viršūnės aprašomos tiksliai nurodant jų koordinates. Tai leidžia sukurti praktiškai bet kokius objektus. Dėl šios priežasties šis mazgas yra vienas iš svarbiausių ir dažniausiai naudojamų VRML kalboje. Todėl norint konvertuoti didžiąją dalį sudėtingų VRML failų, implementuoti indeksuotų sienų aibės konvertavimo algoritmą yra labai svarbu [3].

6 pav. yra pavaizduoti visi galimi IFS laukai ir numatytosios laukų reikšmės. Trumpai apžvelgsime kiekvieną iš jų:

- **coord** - talpina mazgą *Coordinate*, kurio lauke *point* saugomos visos 3D objekto geometrijos koordinatės.
- **coordIndex** - tai sienų sąrašas, kurios nurodomos viršūnių indeksais. Skirtingų sienų indeksai atskiriami nurodant pabaigą -1.
- **color** - talpina mazgą *Color*. Nurodoma kiekvienos sienos spalva.
- **colorPerVertex** - ši vertė parodo, kaip spalvos bus vaizduojamos ant skirtingų sienų.
- **normal**, **normalIndex**, **normalPerVertex** - normalės, kurios nurodomos norint pakeisti numatytuosius medžiagų atspalvius ir reagavimą į šviesą.
- **texCoord** - tekstūros koordinačių sąrašas.
- **ccw** - leidžia pakeisti sienų orientaciją į priešingą.
- **convex** - nurodoma ar geometrijos yra persidengiančios ar ne.
- **solid** - lemia sienos rodymą tik iš vienos pusės arba iš abiejų.
- **creaseAngle** - sulenkimo kampas tarp dviejų gretimų sienų normalių glodžiam spalvų tonavimui.

```
IndexedFaceSet {
        coord NULL
        coordIndex [ ]
        color NULL
        colorIndex [ ]
        colorPerVertex TRUE
        normal NULL
        normalIndex [ ]
        normalPerVertex TRUE
        texCoord NULL
        texCoordIndex [ ]
        ccw TRUE
        convex TRUE
        solid TRUE
        creaseAngle 0.0
}
```

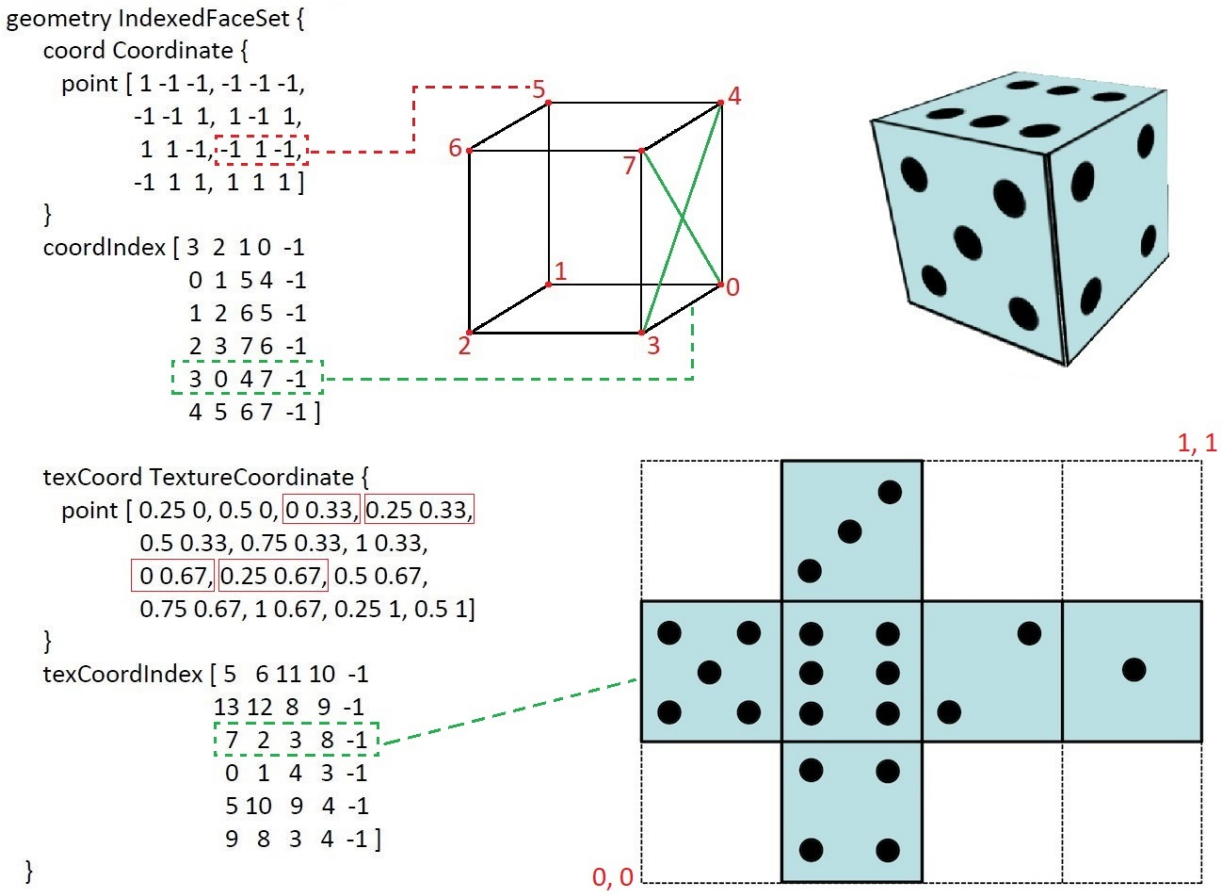
6 pav. Indeksuotų sienų aibės mazgas ir numatytosios laukų vertės.

Siekiant geriau suprasti, kaip VRML kalboje interpretuojama indeksuotų sienų aibė, išnagrinėsime pavyzdį, kuriame vaizduojamas paprasčiausias kauliukas, kurio kiekvienoje kraštinėje yra atvaizduojama skirtinga tekstūros dalis. Kauliuką aprašanti IFS ir gaunamas 3D objektas pavaizduoti 7 pav.

Laukas *coord* talpina mazgą *Coordinate*, kurio lauke *point* yra visų 3D taškų koordinačių sąrašas. Todėl pradžioje apibrėžiame aštuonias viršūnes, kurios numeruojamos nuo 0 iki 7. 7 pav. pavyzdyje viršūnės su koordinatėmis (-1 1 -1) numeris yra 5. Laukas *coordIndex* - tai sienų sąrašas. Sienos aprašomos viršūnių indeksais. Skirtingų sienų indeksai atskiriami nurodant pabaigą -1, kaip matome pavyzdyje žemiau. Po to nurodome sienas, išvardindami kiekvienos iš jų viršūnes (tam naudojame *IndexCoord* mazgą). Pavyzdžiui, 7 pav. kauliuko siena pažymėta žaliu kryžiumi turi viršūnes, kurių numeriai (indeksai) yra: 3, 0, 4, 7.

Norint uždėti paveiksliuką ant 3D geometrijos, naudojama dvimatė plokštuma, kuri yra užpildyta be galo pasikartojančiais tekstūros failo stačiakampiais. Koordinačių pradžia (0,0) sutampa su tekstūros kairiuoju apatiniu kampu, o taškas (1,1) - su viršutiniu dešiniuoju. Tai nepriklauso ar

tekstūros išmatavimai yra 400x600, 1024x768, ar 1000x40. Norint uždėti tekstūrą tam tikru būdu, yra naudojamos 2D tekstūros plokštumos transformacijos ir *texCoord* laukas, kuriame yra *TextureCoordinate* mazgas. Tai - sąrašas tokių tekstūros koordinačių, kurios yra naudojamos *texCoordIndex* lauke, indeksuotų sienų aibės mazge. Jeigu *texCoordIndex* duomenų nėra IFS mazge, tekstūros koordinatės yra atvaizduojamos į IFS viršūnes pagal jų natūralią tvarką. Dėl to bet kurią tekstūros vaizdo dalį galima atvaizduoti į bet kurią pasirinktą objekto geometrijos paviršiaus dalį.



7 pav. VRML failo ir sugeneruoto 3D objekto pavyzdys

3.3. Indeksuotų sienų aibės konvertavimas

Kaip jau buvo minėta 1.3 skyriuje, visos WebGL geometrijos kuriamos naudojantis primityvais - taškais, linijomis arba trikampiais. Three.js bibliotekoje kuriant paprasčiausius objektus galima naudotis papildomomis geometrijomis, tokiomis kaip dėžė, sfera, cilindras ir t.t. Tačiau tokios geometrijos, kuri atitiktų indeksuotų sienų aibę - nėra. Todėl norint atvaizduoti sudėtingas geometrijas, jas reikia kurti naudojantis trikampiais (Three.js biblioteka palaiko ir kvadratus). Reikalingas algoritmas, kuris IFS aprašytas geometrijas ir tekstūras transformuotų į trikampius. Tik tuomet galėsime juos atvaizduoti su WebGL technologija.

3.3.1. Geometrijos konvertavimas

Yra daug įvairių algoritmų, kuriais galime bet kokį daugiakampį atvaizduoti naudodami trikampius. Šiame darbe mes panaudosime taip vadinamą *Vėduoklės trianguliacijos* metodą ir pritaikysime jį IFS konvertavimui. Šiuo būdu visi trikampiai kuriami iš tam tikro pasirinkto pradinio

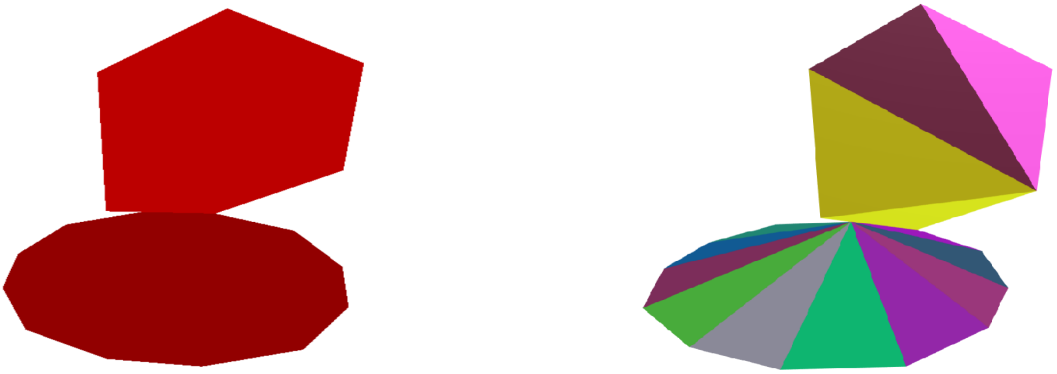
taško. Naudojant šį algoritmą daugiakampiui su *n* kampų, turėsime nupiešti *n-3* įstrižaines, o naują daugiakampį sudarys *n-2* trikampiai [24].

1 algoritmas. Indeksuotų sienų aibės geometrijos ir tekstūros konvertavimas į WebGL

for $sienos$ **do**
  $siena = sienos[i]$
  $textura = texturos[i]$
  $stok = 0$
  while *(siena.ilgis >= 3* **and** *stok < (siena.ilgis - 2))* **do**
    $[i1, i2, i3] = gautiKoordinates(stok, ccw)$
    $virsunes+ = rastiVirsunes(i1, i2, i3)$
    $UV = rastiVirsunesUV(textura, stok, ccw)$
    $visiUV+ = UV$
    $nustatytiSpalvas(i, stok)$
    $stok++$
  end while
end for
$geometrija.virsunes = virsunes$
$geometrija.UV = visiUV$
$atnaujintiGeometrija()$

1 algoritme *sienos* yra dvimatis masyvas, gautas iš IFS *coordIndex* lauko. Vidiniai masyvai yra skirtingų sienų indeksai, kurie VRML atskiriami nurodant pabaigą -1. Algoritmas nagrinėja kiekvieną sieną atskirai, ir suranda naujas trikampių koordinates, kurie sudarytų šią sieną. Kiekvieną kartą kai nagrinėjame vis kitą *sieną*, prie *i* nario yra pridedamas 1. Kintamieji *i1*, *i2* ir *i3* rodo kurias koordinates iš VRML *point* lauko reikia naudoti. Tą atlieka *rastiVirsunes()* metodas, kuris grąžina naujas, sieną sudarysiančio trikampio koordinates. Visi vienos sienos trikampiai turės 1 bendrą viršūnę, nes *i1 = siena[0]*. Taigi tai pačiai sienai *i1* elementas visuomet vienodas. Tai yra būdinga *Vėduoklės trianguliacijos* algoritmo savybė. Svarbu paminėti IFS mazge nurodytą *ccw* narį, kuris gali įgyti *Taip* arba *Ne* reikšmes. Jeigu IFS *ccw* įgyja reikšmę *Ne*, tuomet sienų orientacija turi būti pakeičiama į priešingą. Ši ypatybė yra pasiekiama kintamųjų *i2* ir *i3* vertes sukeičiant vietomis, todėl *gautiKoordinates()* metodas grąžins pakeistas į priešingą naujojo trikampio koordinates. Algoritme taip pat išnagrinėjami ir *normalių* mazgai, kurie keičia objekto medžiagų atspalvius ir reagavimą į šviesą. Jų nagrinėjimas yra labai panašus kaip ir tekstūrų, todėl algoritme nebus aprašytas.

Patikrinti ar 1 algoritmas veikia gerai, naudosime VRML failą, kuriame vaizduojama 3D geometrinė figūra, kurią sudaro 2 sienos. Taisyklingas dvylikakampis ir netaisyklingas šešiakampis. Šios dvi sienos pakreiptos viena kitos atžvilgiu 90° laipsnių kampu ir liečiasi viena kraštine. VRML failo kodas yra patalpintas A priede. VRML kalboje tai yra vienas objektas, sudarytas iš 2 sienų. Konvertavus šią geometriją į WebGL, pagal *Vėduoklės trianguliacijos* metodą gauname, jog mūsų nauja geometrija bus sudaryta iš (12-2)+(6-2) = 14 trikampių. Gautas WebGL objektas pavaizduotas 8 pav. Siekiant parodyti kokie trikampiai sudaro šį objektą, generuojant kiekvieną atskirą trikampį jis buvo nuspalvinamas skirtinga atsitiktine spalva. Kaip matyti iš sukurto WebGL objekto, gauta geometrija sutampa su geometrija, gauta naudojant *BS Contact Stereo* programą, kuri yra skirta peržiūrėti VRML failams. Taip pat galime pamatyti, kad sukurto WebGL objekto tos pačios sienos trikampiai prasideda iš vieno taško.



(a) VRML IFS geometrija, susidedanti iš 2 sienų. Peržiūrima *BS Contact Stereo* programa.

(b) WebGL geometrija, sudaryta iš 14 mažesnių trikampių. Peržiūrima *Chrome* interneto naršyklėje.

8 pav. VRML į WebGL konvertuotos geometrijos pavyzdys

3.3.2. Tekstūros konvertavimas

Indeksuotų sienų aibės mazge nurodomos ne tik sienų pozicijos erdvėje, bet ir kiekvienos sienos spalvos ir tekstūros. Tekstūroms dažniausiai naudojamas *.jpg* ar *.png* paveiksliukas, kuris būna nurodytas *ImageTexture* mazge, *url* lauke. Paprastai IFS sienos yra nuspalvinamos tokia spalva, kokia nurodyta *Material* mazge, *difuseColor* lauke. Tačiau jeigu IFS yra nurodytas *color* mazgas, sienos yra nuspalvinamos šiame lauke nurodytomis vertėmis.

Tekstūrų konvertavimas pavaizduotas 1 algoritme. Kaip ir geometrijos konvertavime, nagrinėjame kiekvieną sieną atskirai, prie *i* nario vis pridėdami 1. Išskaidę vieną sieną į trikampius, surandame kokios tekstūros plokštumos koordinatės atitiks kiekvieno trikampio koordinates. Šių koordinačių radimą atlieka *rastiVirsunesUV()* metodas. Masyvas *UV* saugo vieno trikampio tekstūros plokštumos koordinates. O visi jie saugomi dvimačiame *visiUV* masyve, kuris vėliau leidžia susieti geometrijos koordinates su tekstūros koordinatėmis.

Sienų spalvas nudažo *nustatytiSpalvas* metodas. *spalvos* gaunamos iš IFS *color* lauko ir išsaugomos masyve. Kiekvienas šio masyvo elementas saugo RGB spalvų lentelės reikšmę. Trikampio spalvą nulemia *i* elemento reikšmė, todėl kiekvienas trikampis, kuris priklauso tai pačiai sienai, bus nudažytas ta pačia spalva. Konkrečios sienos spalvą lemia *colorIndex* lauko reikšmės. Jeigu šis laukas VRML faile nenurodytas, naudojama numatytoji sienos reikšmė. Pabaigoje *atnaujintiGeometrija()* metodas iškviečia specialias Three.js bibliotekos funkcijas, kurios nurodo WebGL geometrijos tekstūras pagal nustatytas UV koordinates, atnaujina spalvas ir perskaičiuoja normalių vektorius.

Norint patikrinti kaip veikia algoritmas, buvo konvertuotas tekstūras naudojantis VRML failas. Jame pavaizduotas bokštelis, kurio geometrija sukuriama naudojant IFS, o tekstūros uždedamos naudojantis .png paveiksliukus. Gautas rezultatas vaizduojamas 9 pav. Kairėje pusėje matyti kaip atrodo VRML failas, jį peržiūrint *BS Contact Stereo* programa. Dešinėje pusėje vaizduojamas gautas WebGL objektas, peržiūrimas interneto naršykle. Taip pat parodoma iš kokių trikampių jis buvo sudarytas, nuspalvinant juos vis kita spalva. Nors bokštelio IFS geometrija atrodo paprasta, tačiau konvertavus ją į WebGL, ji buvo sudaryta iš 500 mažesnių trikampių. Abu bokšteliai atrodo panašūs, tačiau WebGL bokštelis atrodo labiau kampuotas, galima pastebėti šią geometriją sudarančius trikampius ir kvadratus. Taip pat WebGL spalvos atrodo tamsesnės, o medžiaga blizgesnė. Tokį skirtumą galėjo lemti skirtingas VRML ir WebGL objektų medžiagos šviesos interpretavimas.

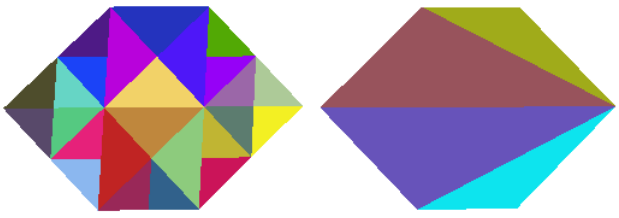
(a) VRML (b) WebGL

9 pav. VRML bokštelis konvertuotas į WebGL

Naudojantis sukurtais algoritmais į WebGL buvo konvertuotas didelis ir daug sudėtingų IFS laukų turintis VRML failas. Šiame faile buvo pavaizduota Vilniaus Šv. Nikolajaus cerkvė (B priedas). Kaip matyti iš gauto rezultato, WebGL objekto medžiagos pasižymi kitokiais atspindžiais ir reagavimu į apšvietimą. Taip pat WebGL objektas vaizduojamas geresne rezoliucija ir ryškesnėmis spalvomis. Kitokias objektų medžiagų spalvas galėjo lemti skirtingos numatytosios VRML ir Three.js bibliotekos medžiagų parametrų vertės. Rezoliucijų skirtumus ir gražesnes tekstūras WebGL objekte galėjo lemti skirtingos VRML ir WebGL objektų peržiūrėjimo programos. Siekiant patikrinti ar algoritmas gali konvertuoti didelę VRML sceną, buvo konvertuotas failas kuriame vaizduojami Vilniaus universiteto kiemeliai (VU-kiemai.wrl). Šiame VRML faile buvo panaudota daugybė įvairaus dydžio ir sudėtingumo indeksuotų sienų aibių mazgų. Pabandžius konvertuoti šį failą, gautas rezultatas sutapo su vaizdu, matomu naudojant specialią VRML failų peržiūrėjimo programą. Todėl galime teigti, kad algoritmas gerai konvertuoja įvairias indeksuotų sienų aibes, kuriose naudojami koordinačių, tekstūrų ir normalių mazgai.

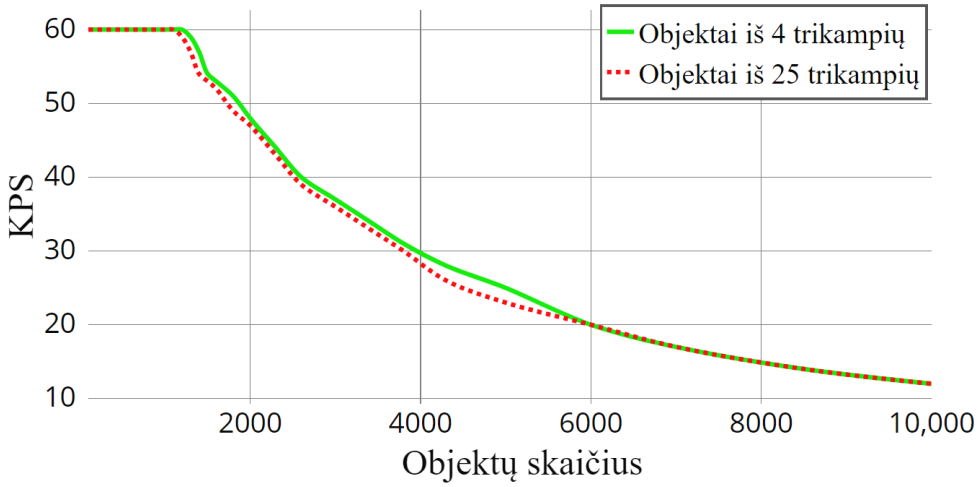
3.4. WebGL greitaveikos įvertinimas

Konvertavus N-cerkve.wrl failą, sukurtą 3D sceną sudarė iš 10118 trikampių sudaryti 345 objektai. Gautas WebGL objektas naršyklėje buvo rodomas 60 kadrų per sekundę (KPS) dažniu. Tai aukščiausias rezultatas, kokį galima paskiekti su Three.js biblioteka ir WebGL technologija, leidžiantis be jokių trikdžių peržiūrėti sugeneruotą 3D objektą naršyklėje. Tačiau importuojami VRML failai gali būti didesni arba sudaryti iš daugiau objektų. Todėl reikia įvertinti, kokią įtaką didėjantis objektų arba juos sudarančių trikampių skaičius turi naršyklės atkuriamų KPS skaičiui. Ši analizė padėtų iš anksto numatyti reikalingus atlikti pakeitimus, siekiant optimizuoti VRML failo konvertavimo algoritmą. Siekdami patikrinti WebGL greitaveiką, naudosime šešiakampius objektus, pavaizduotus 10 pav. Bandymus atliksime *Chrome* interneto naršyklėje.



10 pav. Šešiakampiai WebGL objektai, sudaryti iš 4 ir 25 trikampių.

Šie šešiakampiai yra vienodo dydžio ir išmatavimų. Generuodami skirtingą šių objektų skaičių, tikrinome kaip tai paveikia naršyklės atkuriamų kadrų per sekundę skaičių. Gauti rezultatai pateikti 11 pav. Iš grafiko matyti, kad didžiausią įtaką naršyklės KPS turi WebGL objektų skaičius. Šiam skaičiui peržengus 1200, naršyklės KPS pradėjo mažėti, o objektų skaičiui pasiekus 10000, tesiekė 5 kadrus per sekundę. Tuo tarpu mažesnis objektą sudarančių trikampių skaičius turėjo labai mažą įtaką, ir leisdavo naršyklėse pasiekti tik 1-2 kadrais didesnį KPS skaičių. Taigi, didžiausią įtaką naršyklės atkuriamų kadrų per sekundę dažniui turi reikiamų sugeneruoti WebGL objektų skaičius. Tuo tarpu mažesnis objektą sudarančių trikampių skaičius turi tik nežymią įtaką WebGL greitaveikai. Todėl siekiant išgauti kuo didesnį naršyklėje atkuriamą KPS, reikia stengtis sukurti kuo mažiau objektų. VRML failo konvertavime tai galima pasiekti apjungiant greta esančias geometrijas į vieną objektą.



11 pav. Chrome naršyklės atkuriamų kadrų per sekundę dažnio priklausomybė nuo WebGL objektų ir juos sudarančių trikampių skaičiaus

WebGL veikia interneto naršyklėse, todėl svarbu yra ne tik atkuriamų kadrų per sekundę skaičius, bet ir sukurtų objektų dydis. Siųsti didelius failus gali būti brangu arba dėl lėto interneto greičio tai gali trukti labai ilgai. Three.js biblioteka turi įrankius, kurie gali išeksportuoti sugeneruotą 3D sceną į *JSON*, *gltf*, *obj* ir kitus failų formatus. Todėl galime įvertinti, kaip naršyklėje sugeneruotų objektų ir trikampių skaičius lemia išeksportuoto WebGL failo dydį. Tam sugeneruosime skirtingą skaičių šešiakampių, susidedančių iš 4 arba 25 trikampių (10 pav.), ir išsaugosime kaip JSON failą. Gauti rezultatai pavaizduoti 1 lentelėje. $Dydis_4$ ir $Dydis_{25}$ nurodo išeksportuotų failų dydžius kilobaitais, atitinkamai objektus kuriant iš 4 arba 25 trikampių.

1 lentelė. JSON failo dydžio kitimas keičiantis objektų ir juos sudarančių trikampių skaičiui.

$Objektai$	$Dydis_4(KB)$	$Dydis_{25}(KB)$	$Santykis_{25/4}(KB)$
100	64	143	2,23
300	191	427	2,24
600	381	854	2,24
800	508	1138	2,24

Kaip matyti iš gautų duomenų, skirtingas objektą sudarančių trikampių skaičius turi didelę įtaką WebGL failo dydžiui. Tas pats skaičius objektų, sudarytų iš 4 trikampių, vidutiniškai užimdavo

2,24 karto mažiau vietos nei objektų, sudarytų iš 25 trikampių. Todėl konvertuojant VRML failą į WebGL, ir siekiant kad sukurti objektai užimtų kuo mažiau vietos, reikia juos kurti naudojant kuo mažiau trikampių. Konvertavimo algoritme tai galima pasiekti naudojant skirtingus ir sudėtingesnius nei *Vėduoklės trianguliacijos* metodus arba apjungiant indeksuotų sienų aibes.

3.4.1. BufferGeometry ir Geometry klasės

Sukurti objekto geometriją naudojantis Three.js biblioteka yra du būdai. Naudoti $Geometry$ arba $BufferGeometry$ klasių objektus. Naudoti $Geometry$ yra paprastesnis ir lengvesnis kelias, rekomenduojamas pradedantiesiems kurti trimatę grafiką. Taip pat šį būdą yra patogiau naudoti, jeigu kuriamas projektas nėra labai didelis, yra kuriama daug sudėtingų animacijų arba greitaveika nėra labai svarbi. Kuriant geometrijas naudojant $Geometry$ klasę, visi atributai (viršūnės, spalvos, koordinatės ir t.t) yra priskiriami ir saugomi kaip Three.js objektai. Pavyzdžiui, objekto koordinatės ar viršūnių padėtys nurodomos naudojant Three.js trimačius vektorius, o spalva nurodoma kaip Three.js spalvos objektas. Tokia struktūra yra lengvai suprantama programuojant algoritmus bei leidžia lengviau pasiekti ir keisti įvairius geometrijos parametrus. Tai ypač palengvina animacijų kūrimą, kuomet geometrijos viršūnės, koordinatės ar normalės turi būti nuolat keičiamos.

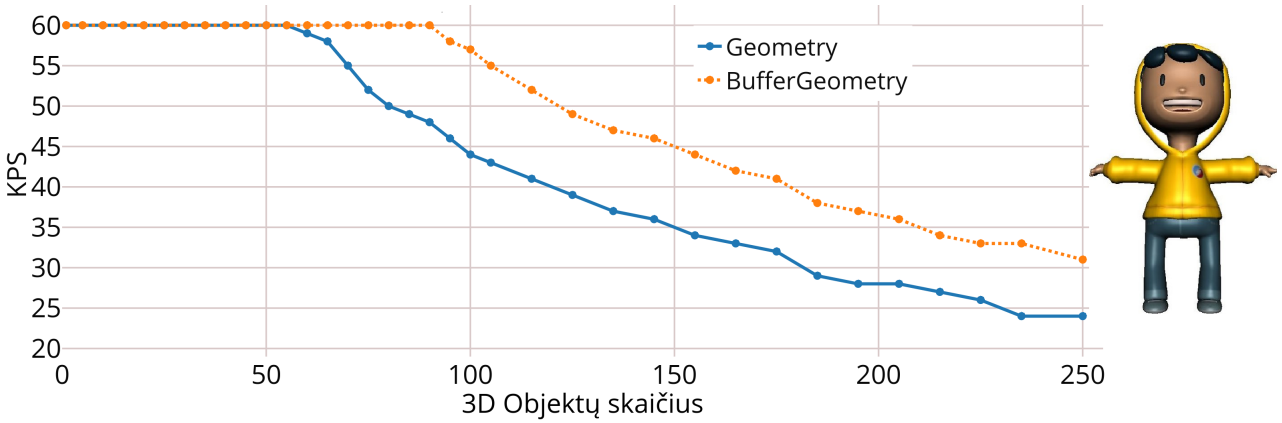
Naudojant $BufferGeometry$ klasę, visos 3D objekto geometrijos atributų vertės saugomos primityvių kintamųjų masyvuose [21]. Toks saugojimas leidžia sutaupyti laiko kuriant šešėliuoklių ir viršūnių programas, nes nebereikia atlikti konversijų dėl JavaScript skaičių ir Three.js objektų naudojimo. Išskaičiuoti reikšmes šiems masyvams yra sudėtingiau nei naudojant Three.js objektus ir reikalauja gilesnių 3D programavimo žinių. Sudėtingesnis yra ir animacijų kūrimas, nes reikia ne tik žinoti naujas atributų reikšmes, bet ir nuolat perskaičiuoti kurioje vietoje masyvuose šias reikšmes reikia pakeisti. Todėl šią klasę yra rekomenduojama naudoti kuriant statinius objektus, kurie nebus animuojami. Pagrindinis $BufferGeometry$ klasės naudojimo privalumas yra tai, jog sukurti WebGL objektai pasižymi geresne greitaveika. Taip yra dėlto, nes iš anksto paruošti duomenys perduodami įrenginio vaizdo plokštei, taip sumažinant procesoriaus skaičiavimo iteracijų skaičių. Geresnė greitaveika ypač pastebima kuomet naršyklėje atvaizduojamų objektų skaičius yra didelis. Todėl $BufferGeometry$ yra rekomenduojama naudoti kuriant didelius projektus arba kuomet greitaveika yra labai svarbi.

Sukurtame VRML konvertavimo algoritme geometrijos gali būti kuriamos naudojant $Geometry$ arba $BufferGeometry$ klases. Tačiau galutinėje konvertavimo algoritmo versijoje yra nustatyta, jog visi objektai kuriami naudojant $Geometry$ objektą. Šiame darbe konvertuojant VRML failus nėra sukuriama labai daug objektų, todėl $BufferGeometry$ naudojimo įtaka nepasimatytų konvertuojamų failų greitaveikoje. Taip pat $BufferGeometry$ nenaudojama dėl paprastesnio VRML animacijų konvertavimo. Tobulinant konvertavimo algoritmą, arba norint konvertuoti didelius animuotus VRML pasaulius, reikėtų papildyti algoritmą, kad jis galėtų konvertuoti objektų animacijas, kurių geometrijos buvo sukurtos naudojantis $BufferGeometry$ klases.

Vis dėlto, kad įsitikintume $BufferGeometry$ nauda konvertuojant VRML failus, išmatuosime kokia yra WebGL greitaveika kuriant objektus naudojant $Geometry$ arba $BufferGeometry$ klases. Tam konvertuosime VRML failą, kuriame pavaizduotas berniukas (Berniukas.wrl). Šis failas pasižymi didele indeksuotų sienų aibe ir aukštos kokybės tekstūromis, todėl bandant atvaizduojant didelį šių objektų kiekį naršyklėje, bus matomas greitaveikos suprastėjimas. Pradžioje visus matavimus atliksime visų geometrijų kūrimui naudodami $BufferGeometry$ klasę. Vėliau viską pakartosime naudodami $Geometry$ klasę. Tokiu būdu galėsime patikrinti kokia yra $BufferGeometry$ naudojimo įtaka geresnei naršyklės greitaveikai. Tai vėliau leistų lengviau nu-

spręsti ar verta tobulinti konvertavimo algoritmą leidžiant ir animuotiems VRML failams panaudoti $BufferGeometry$ klasę.

Gauti greitaveikos matavimo rezultatai pateikti 12 pav. Y ašyje pažymėtas *Chrome* naršyklės atkuriamų kadrų per sekundę skaičius. Esant mažam objektų skaičiui, gaunamas maksimalus įmanomas WebGL KPS - 60. X ašyje pažymėtas objektų (berniukų) skaičius, kuris intervalais buvo keičiamas nuo 1 iki 250. Visi objektai taip pat buvo animuojami, sukant juos aplink z ašį. Vienas apsisukimas truko 3 sekundes. Nauji objektai scenoje buvo dėliojami jų pozicijos koordinates pastumiant tam tikru atstumu nuo originalios. Tokiu būdu didinant berniukų skaičių, vis didesnę ekrano užimamą dalį sudarė besisukantys WebGL objektai. Toks objektų atvaizdavimo būdas leidžia atkurti sąlygas, kuomet yra konvertuojama daug ir įvairių animuotų VRML objektų, kurie ekrane išsidėsto įvairia tvarka. Kaip matyti iš gautų rezultatų, objektai, sukurti naudojantis $BufferGeometry$ klase, pasižymi geresne greitaveika. Naudojant $Geometry$ klasę, naršyklės greitaveika pradėjo mažėti objektų skaičiui pasiekus 60. Tuo tarpų su $BufferGeometry$ greitaveika suprastėjo berniukų skaičiui pasiekus 90. Taigi naudojant šią klasę buvo galima konvertuoti 50 % daugiau objektų nei su $Geometry$ ir vis dar išsaugoti maksimalų galimų kadrų per sekundę skaičių. Toliau didinant konvertuojamų objektų skaičių, $BufferGeometry$ vidutiniškai užtikrindavo 10 kadrų per sekundę geresnę greitaveiką nei $Geometry$ tam pačiam objektų skaičiui. Todėl konvertuojant labai didelius VRML failus ir norint užtikrinti geriausią įmanomą greitaveiką, geometrijas kurti geriausia naudojant $BufferGeometry$. $BufferGeometry$ panaudojimas VRML failuose su animacijomis galėtų būti atliekamas tolesniuose darbuose.



12 pav. VRML failo konvertavimas naudojantis $Geometry$ (mėlyna) ir $BufferGeometry$ (oranžinė) klases. Dešinėje pusėje pavaizduotas konvertuojamas VRML objektas - berniukas.

4. VRML animacijų konvertavimas į WebGL

Konvertuoti VRML animacijas yra svarbu, nes tai viena iš dažniausiai naudojamų savybių. Animacijos suteikia 3D pasauliui dinamiškumo, gyvumo ir leidžia vartotojui interaktyviai dalyvauti aplinkoje. VRML animacijos yra sukuriamos naudojant įvykių jutiklius. Tuo tarpu Three.js naudojama animacijų sistema skiriasi nuo VRML. Ji yra sukurta žaidimų kūrimo variklių *Unity* ir *Unreal Engine 4* pavyzdžiu ir animacijoms naudoja Track/Clip/Action/Mixer/Animator modelį [9]. Šiuo metu esantis Three.js VRML konverteris nepalaiko ir nekonvertuoja animacijų. Todėl bus sukurtas naujas VRML animacijų interpretavimo ir konvertavimo į WebGL algoritmas. Šiame skyriuje išnagrinėsime VRML ir Three.js animacijų modelius ir pasiūlysime būdus, kurie konvertuotų VRML animacijas į Three.js. Taip pat aprašysime bandymus ir gautus rezultatus.

4.1. VRML animacijos

Kurti VRML animacijoms, taip pridedant dinamiškumo ir interaktyvumo, naudojamas įvykių jutiklių (Event) modelis. VRML pasaulyje sukuriami sensoriai, kurie seka tam tikrus vartotojo veiksmus (žymeklio priartėjimas prie objekto, kompiuterio pelės ar klaviatūros mygtuko paspaudimas ir t.t). Suaktyvinus šiuos sensorius, pradedama vykdyti VRML animacija. Įvykių jutiklius galima įsivaizduoti kaip programos dalis, kurios prisiriša prie tam tikrų VRML mazgų ir gali perduoti informaciją kitiems mazgams. Tokiu būdu VRML mazgai gali sąveikauti tarpusavyje. Paveiktas vienas VRML mazgas perduoda informaciją kitiems susietiems mazgams, kad kažkas pasikeitė, ir jie turi atitinkamai sureaguoti. Tokiu būdu sukuriamas domino efektas. Paveikus vieną VRML mazgą, šis pradeda veikti kitus susietus mazgus ir prasideda VRML animacija [23].

Įvykių jutiklius išsiųsti ir priimti gali tik tam tikri VRML mazgų laukai. Kurie mazgai gali dalyvauti animacijose yra nurodyta VRML formato dokumentacijoje. Kaip pavyzdį panagrinėkime VRML animacijoms labai dažnai naudojamą transformacijų mazgą. Jis yra pavaizduotas 13 pav. su visais galimais laukais ir numatytosiomis jų vertėmis.

```
Transform {
  eventIn       MFNode      add_children
  eventIn       MFNode      remove_children
  exposedField  SFVec3f     center           0 0 0
  exposedField  MFNode      children         []
  exposedField  SFRotation  rotation         0 0 1  0
  exposedField  SFVec3f     scale            1 1 1
  exposedField  SFRotation  scaleOrientation 0 0 1  0
  exposedField  SFVec3f     translation      0 0 0
  field         SFVec3f     bboxCenter       0 0 0
  field         SFVec3f     bboxSize         -1 -1 -1
}
```

13 pav. VRML Transformacijos mazgas. Kairysis stulpelis parodo laukų sąsajos tipus. [6]

Pirmajame (apvesta stačiakampiu) 13 pav. stulpelyje pavaizduotos reikšmės nurodo VRML lauko sąsajos tipą (interface type). Kiekvienas VRML laukas gali turėti vieną iš keturių galimų sasajos tipo reikšmių. Šios reikšmės yra:

- **field** - laukas neveikiamas animacijų.
- **eventIn** - laukas gali priimti įvykių jutiklį.
- **eventOut** - laukas gali sugeneruoti įvykio jutiklį.
- **exposedField** - laukas gali sugeneruoti ir priimti įvykių jutiklį.

Visos VRML animacijos susijusios su mazgų laukų reikšmių pakeitimu. Vadinasi visi VRML laukai, kurie turi sąsajos tipą - **field**, negali būti panaudoti kuriant animacijas. Kaip matyti iš šio

pavyzdžio, tik 2 transformacijos mazgo laukai yra nepanaudojami animacijoms. Tai *bboxCenter bboxSize* laukai. Visoje VRML kalboje laukų, kurie negali būti paveikti animacijomis yra apie 30% [6]. Visi likę laukai gali priimti ir/arba išsiųsti įvykių jutiklius, todėl yra naudojami animacijose.

Kaip buvo minėta ankščiau, VRML animacijos veikia domino principu, kuomet vienas paveiktas VRML mazgas perduoda kitiems per įvykių jutiklius surištiems mazgams informaciją ir taip prasideda animacija. Tačiau tuomet yra reikalinga paveikti pati pirmą mazgą. Tai VRML animacijoje visuomet padaro kažkuris išorinis veiksnys. Tai, pavyzdžiui, gali būti vartotojo pelės mygtuko paspaudimas, žymeklio užvedimas ant objekto ar tam tikro laiko atėjimas. Šis pirminis įvykis pradeda veikti susietus VRML laukus, kurie tuomet sukelia kitus įvykių jutiklius. Taip prasideda grandininė reakcija ir vartotojas mato VRML animaciją.

Vienas iš svarbiausių ir dažniausiai naudojamų įvykių generavimo mazgų yra *Laiko jutiklio* mazgas (TimeSensor). Šis mazgas animacijose yra populiarus dėl to, nes jame galima nurodyti laiko tarpą, kuriam praėjus automatiškai sugeneruojamas *įvykis*, paveikiantis ir kitus VRML mazgus ir laukus. Taip pat šiame mazge nurodoma ir animacijos trukmė. Tokiu būdu galima sukurti laiko trukme kontroliuojamą animaciją, kuri pradėtų veikti ir be vartotojo įsikišimo. Animacija gali prasidėti iš karto atidarius VRML failą ar tik po tam tikro laiko tarpo.

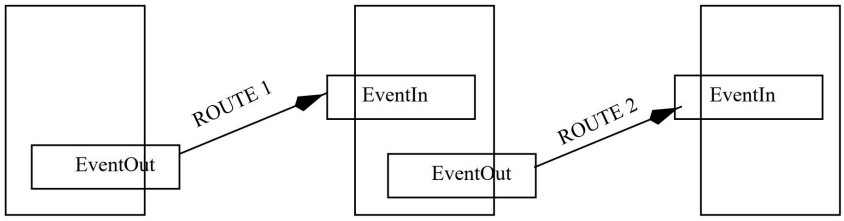
Du VRML laukai, kurių vienas gali išsiųsti įvykio jutiklį, o kitas priimti, susiejami naudojantis maršrutu (Route). Maršrutai sujungia du VRML laukus ir dažniausiai yra aprašomi VRML failo pabaigoje. Veikimo principus paaiškinsime pateikdami maršruto sintaksės pavyzdį, kuris pavaizduotas 14 pav.

```
ROUTE Node1.eventOut_changed TO Node2.set_eventIn
```

14 pav. VRML marštruto sintaksės pavyzdys.

Node1 ir *Node2* yra mazgų vardai, kurie VRML faile yra pažymėti raktažodžiu DEF. *eventOut* yra įvykis, kuris priklauso VRML laukui, galinčiam sugeneruoti įvykių jutiklį. *eventIn* yra įvykis, kuris priklauso VRML laukui, galinčiam priimti įvykių jutiklį. Svarbu paminėti, kad jungiant du laukus ROUTE komanda, šie du laukai turi būti to pačio VRML tipo. Suprasti, kuris įvykis priklauso kuriam laukui yra paprasta - įvykio pavadinimas sutampa su VRML lauko pavadinimu. Raktažodis *changed* yra pridedamas prie įvykio pavadinimo, kurio lauko sąsajos tipas yra *eventOut*. Raktažodis *set* yra pridedamas prie įvykio, kurio sąsajos tipas yra *eventIn*. Tai leidžia lengviau skaityti ir suprasti maršrutų sintaksę. Vis dėlto, šie raktažodžiai yra neprivalomi, jeigu lauko jutiklio sąsajos tipas yra *exposedField*.

Vienas maršrutas leidžia sujungti ir sukurti įvykį tik tarp dviejų VRML laukų. Tačiau dažniausiai kuriant animacijas to neužtenka, todėl yra kuriamos maršrutų grandinės. Įprastas daugelio VRML laukų sujungimas animacijoms naudojant Maršrutų/Įvykių modelį pavaizduotas 15 pav.



15 pav. VRML Maršrutų/Įvykių modelis. [23]

Kaip matyti iš 14 pav. pateikto pavyzdžio, maršrutų sintaksė, leidžianti sukurti įvykius ir sujungti du VRML laukus yra paprasta. Tačiau verta pastebėti, kad šiuo būdu nėra nurodomos naujos

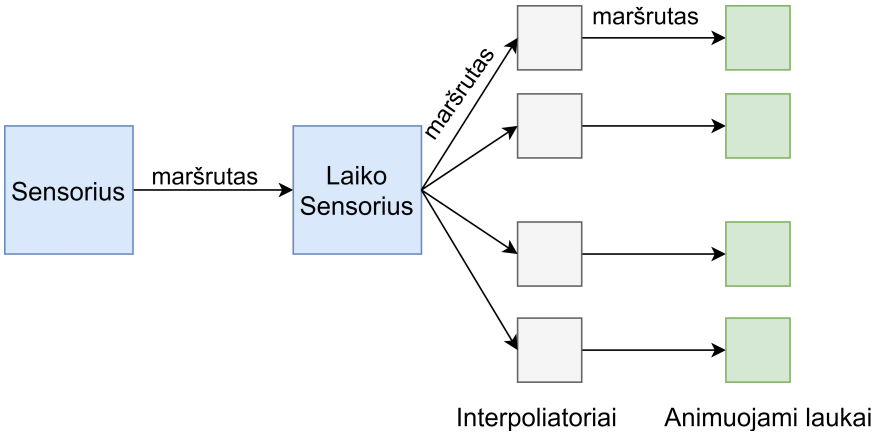
VRML mazgų vertės. Norint animuoti objektus, reikia nurodyti naujas laukų vertes. Tam VRML kalboje naudojami specialūs mazgai - interpoliatoriai. VRML turi 6 interpoliatorių rūšis, kurie naudojami animuoti skirtingas objekto savybes. Pavyzdžiui, pozicijos interpoliatorius naudojamas siekiant animuoti ir pakeisti objekto koordinates, tuo tarpu spalvų interpoliatorius naudojamas norint pakeisti objekto spalvą. Interpoliatoriai yra glaudžiai susiję su maršrutais, sensoriais ir animacijomis. Interpoliatorių sintaksė yra labai paprasta - juos visada sudaro du laukai. Raktas (Key) ir rakto vertės (KeyValue). Interpoliatoriaus sintaksės pavyzdys parodytas 16 pav.

```
DEF position PositionInterpolator {
    key [ 0  1 ]
    keyValue [ 4 .5 0, -4 .5 0 ]
}
```

16 pav. Pozicijos interpoliatoriaus.

Rakto vertės visuomet yra rėžiuose nuo 0 iki 1. Naujos animuojamo VRML lauko vertės nurodomos rakto vertės lauke. Rakto ir rakto verčių ilgiai turi būti vienodi. 16 pav. pateiktame pavyzdyje yra nurodomos naujos VRML objekto koordinatės. Kuomet rakto vertė yra 0, objekto koordinatės yra [4, 0.5, 0]. Rakto vertei pasiekus 1, objekto koordinatės priskiriamos [-4, 0.5, 0]. Visas tarpines rakto vertes, kuomet pvz. rakto vertė yra 0.5, VRML naršyklė apskaičiuoja automatiškai ir jeigu animacija yra tiesinė, jų nurodyti nebūtina.

Išsiaiškinus pagrindinius VRML animacijų elementus, galime nubraižyti įprastą VRML animacijos schemą. Ji yra pavaizduota 17 pav.



17 pav. VRML animacijos diagrama.

Pradžioje aktyvinamas pirminis, išorinis sensorius, kuris paveikia 3D pasaulio sceną. Tai galėtų būti pelės ar klaviatūros mygtuko paspaudimas. Naudojant maršrutus, šis sensoriaus surišamas su *Laiko Sensoriaus* mazgu. Šiame mazge nurodoma animacijos trukmė sekundėmis, jos pradžia ir pabaiga. Nurodomi ir kiti svarbūs parametrai, pvz. ar animacija turi pasikartoti, ar veikti tik vieną kartą. Laiko sensoriaus mazgas surišamas su interpoliatoriais, kuriuose nurodomos naujos VRML mazgų laukų vertės. Pavyzdžiui, jeigu animacijos trukmė yra 5 sekundės, tai naudodami pozicijos interpoliatrių iš 16 pav., animacijos pradžioje esant 0 sekundei, naudosime naujas reikšmės kuomet raktas yra 0. Tai mus duos naujas objekto koordinates [4, 0.5, 0]. Animacijos pabaigoje, penktą sekundę panaudosime raktą su reikšme - 1, tuomet naujos koordinatės bus [-4, 0.5, 0]. Kokios objekto koordinatės turėtų būti 2.5 sekundę, turėtų parodyti interpoliatoriaus raktas su 0.5 reikšme. Tačiau kadangi tokio rakto nėra, naujas koordinates VRML naršyklė paskaičiuoja automatiškai.

Šiuo atveju jos bus [0, 0.5, 0]. Toks interpoliatorių veikimas sukuria gražią ir tolygią animaciją. Taip pat sutaupomas laikas ir failo dydis, nes nereikia nurodinėti visų naujų animuojamo objekto laukų verčių. Užtenka nurodyti tik pradžios ir pabaigos vertes. Tačiau dažnai nurodomos ir tarpinės vertės, kuomet raktas yra 0.5 ar 0.3. Tai naudojama atvejais, kuomet objektas yra sukamas arba animacija yra netolydi. Verta paminėti, jeigu animacija yra pasikartojanti, interpoliatoriaus rakto su verte 0 ir su verte 1 rakto verčių reikšmės turėtų sutapti. Kitu atveju sukurtoje animacijoje atsiras trūkinėjimas.

Naudojant laiko sensorius, interpoliatorius ir maršrutus galima sukurti didžiąją dalį VRML animacijų. Šių trijų mazgų užtenka sukurti sudėtingas ir įvairias animacijas. Tokios VRML animacijos pavyzdys pateiktas 18 pav.

```
# Rotating Box
Group {
  children [
      DEF Column Transform {
        children [
            Shape {
              geometry Box {}
              appearance Appearance { }
            }
        ]
      }
      DEF Clock TimeSensor {cycleInterval 2.0  loop TRUE} (2)
      DEF ColumnPath OrientationInterpolator {
        key [ 0.0 0.50 1.0 ] keyValue [ 0.0 0.0 1.0 0.0 0.0 0.0 1.0 3.14 0.0 0.0 1.0 6.28 ]
      }
  ]
}
ROUTE Clock.fraction_changed TO ColumnPath.set_fraction (1)
ROUTE ColumnPath.value_changed TO Column.set_rotation   (3)
```

18 pav. VRML animacija - besisukantis kubas.

18 pav. pavaizduotas paprastas VRML animacijos pavyzdys - besisukantis kubas. Ši animacija veikia naudodama laiko sensorius, interpoliatorius ir maršrutus ir yra sukuriama šia seka:

- Maršrutas, pažymėtas numeriu (1) suriša du VRML mazgus - Laiko sensorių (žalia spalva) ir orientacijos interpoliatorių (mėlyna spalva). Tokiu būdu sukuriamas šiuos du mazgus siejantis įvykių jutiklis. Keičiantis laiko sensoriaus vertei, iš interpoliatoriaus paimamos vis kitos rakto vertės.
- Laiko sensoriaus mazge (2) nurodoma animacijos trukmė (cycleInterval = 2 sekundės), animacijos pasikartojimas (loop = true). Visi kiti šio mazgo laukai turi numatytąsias reikšmės, todėl VRML faile nėra nurodomi. Animacijos pradžią ir pabaigą nurodo StartTime ir StopTime laukai. Jeigu StartTime >= StopTime, tuomet laiko sensoriaus mazgas iškarto sugeneruoja įvykį. Kadangi šios numatytosios VRML vertės yra 0, įvykis sugeneruojamas iš karto ir VRML animacija prasideda vos tik atsidarius VRML failą.
- Maršrutas, pažymėtas numeriu (3), suriša pozicijos interpoliatorių su transformacijos mazgo sukimosi (Rotation) lauku. Vis kitos interpoliatoriaus rakto vertės yra priskiriamos transformacijos mazgo sukimosi laukui. Vartotojas pradeda matyti besisukantį kubą. Kubas apsisuka per 2 sekundes ir animacija vėl prasideda iš naujo.

Iš 18 pav. pavyzdžio matyti, kad animacijos pradžioje, laikui esant 0 sekundžių, transformacijos mazgo sukimosi lauko vertės nustatomos [0, 0, 1, 0]. Praėjus pusei animacijos laiko (1 sekundei), panaudojama rakto reikšmė 0.5 ir sukimosi vertės nustatomos [0, 0, 1, 3,14]. Tokiu būdu kubas per vieną sekundę pasukamas 180 laipsniu kampu aplink z ašį. Animacijai baigiantis, vertės nustato-

mos [0, 0, 1, 6,28] ir vienas animacijos ciklas baigiasi. Kadangi laiko sensoriaus mazge nurodytas pasikartojimas, animacijai pasibaigus ji pradedama rodyti iš naujo. Tarpinės vertės, kuomet laiko sensoriaus laikas yra 0.5 ar 1.5 sekundės VRML naršyklės yra parenkamos automatiškai, išvedant interpoliatoriaus verčių vidurkį. Tokiu būdu vartotojas mato sklandžią ir tolygią VRML animaciją. Kadangi naudojama pasikartojanti animacija, galima matyti, kad orientacijos interpoliatoriaus pirmos ir paskutinės rakto vertės sutampa.

4.2. Three.js animacijos

2015 metais Three.js animacijų sistema buvo visiškai pakeista, ir šiuo metu naudojama architektūra yra panaši į naudojamą Unity ar Unreal Engine 4 trimatės grafikos kūrimo programose. Tai palengvina animacijų kūrimo darbą, nes animacijų sistema nėra tokia specifinė kaip VRML atveju. Žemiau pateikiame svarbiausius Three.js animacijų sistemos objektus [9].

- **Keyframe Tracks** - raktinė informacija, kurią sudaro animuojamo objekto parametro pavadinimas, laiko ir naujų verčių masyvai. Taigi viename raktiniame takelyje išsaugoma tai, kam VRML formate išsaugoti reikalingi panaudoti 3 skirtingi mazgai - laiko sensorius, interpoliatorius ir maršrutas. Three.js turi 6 skirtingus raktinių takelių objektus, kurie naudojami animuoti skirtingiems parametrams.
- **Animation Clips** - visos atskiros Three.js animacijos yra saugomos animacijų klipuose. Animacijos klipas yra atsakingas už vieną 3D objekto dalies animaciją. Jeigu animuojame einantį žmogų, vienas animacijos klipas gali būti skirtas žmogaus ėjimui, kitas šuoliui, o trečias - pasilenkimui. Viename animaciniame klipe gali būti nurodomi keli raktiniai takeliai, kurie yra atsakingi už skirtingų objekto parametrų animaciją. Pvz. vienas takelis skirtas animuoti spalvos pasikeitimui, o kitas objekto koordinačių kitimui ir t.t.
- **Animation Mixer** - skirtas kontroliuoti keletą animacijų vienu metu, su galimybe sujungti ar išskirti animacijas.
- **Animation group** - skirtingus Three.js objektus galima sugrupuoti panaudojant animacijos grupes. Tokiu būdu animuojant grupę, yra animuojami ir visi toje grupėje esantys objektai. Tai padeda sutaupyti laiko ir palengvina animacijų sinchronizavimą.
- **Animation Actions** - aukščiausiai Three.js animacijos hierarchijoje esantis objektas, kuris valdo animacijų mikserius ir kontroliuoja skirtingų animacijos klipų veikimą. Animacijų veiksmai yra atsakingi už animacijų paleidimą, sustabdymą, kartojimą, sinchronizavimą ir kitas svarbias savybes. Animacijų veiksmai sujungia įvairias animacijas į vieną ir leidžia jas vykdyti arba sustabdyti sinchronizuotai vienu metu. Tokiu būdu vienu metu gali būti valdomos visos scenos animacijos.

Nors Three.js animacijų sistema yra pakankamai gera sukurti tiek paprastas tiek ir sudėtingas animacijas, dažnai yra naudojamos ir išorinės bibliotekos, dar labiau palengvinančios animacijų kūrimą. Viena iš populiariausių tokių bibliotekų yra Tween.js. Ši biblioteka dažnai naudojama paprastoms animacijoms, kuomet tereikia žinoti pradines ir galutines objekto parametrų vertes. Su Tween.js biblioteka mes animuosime VRML peržiūros taškus, kurie yra naudojami sklandžiam kameros perėjimui nuo vieno žiūrėjimo taško prie kito. Tween.js yra naudinga ir dėl to, nes nėra būtina nurodyti pradines ir galutines parametrų vertes. Galima nurodyti ir formulę, todėl naudojant dar mažiau informacijos ir programinio kodo galima sukurti įvairesnę animaciją.

19 pav. pateiktas programinis kodas, sukuriantis tą pačią kaip ir 18 pav. pavaizduotą animaciją - besisukantį kubą.

```
scene = new THREE.Scene();
var Object3D = new THREE.Object3D();
var geometry = new THREE.BoxBufferGeometry( 2, 2, 2 );
var material = new THREE.MeshBasicMaterial();
var Box      = new THREE.Mesh( geometry, material );
Object3D.add(Box);
scene.add( Object3D );
q1 = new THREE.Quaternion().setFromAxisAngle(new THREE.Vector3(0, 0, 1).normalize(), 0);
q2 = new THREE.Quaternion().setFromAxisAngle(new THREE.Vector3(0, 0, 1).normalize(), 3.14);
q3 = new THREE.Quaternion().setFromAxisAngle(new THREE.Vector3(0, 0, 1).normalize(), 6.28);
KF = new THREE.QuaternionKeyframeTrack(
'.quaternion', [0, 1, 2], [q1.x, q1.y, q1.z, q1.w, q2.x, q2.y, q2.z, q2.w, q3.x, q3.y, q3.z, q3.w]);
clip = new THREE.AnimationClip('default', 2, [KF]);
mixer = new THREE.AnimationMixer(Object3D);
clipAction = mixer.clipAction(clip);
clipAction.play();
```

19 pav. WebGL animacija sukurta su Three.js biblioteka - besisukantis kubas.

Pradžioje sukuriama Three.js scena, kurioje talpinami visi 3D objektai. Taip pat sukuriama dėžė ir 3D objektas (Object3D), kuris atitinka VRML transformacijos mazgą. Vykdant animaciją sukamas ne kubas, bet 3D objektas kuriame yra patalpinta dėžė. Tokia struktūra palengvina animacijų kūrimą. Kadangi objekto orientacija žinoma 3 momentais - animacijos 0, 1 ir 2 sekundę, reikia sukurti 3 kvaternijonų vektorius, kurie saugotų sukimosi informaciją. Tai yra atliekama sukuriant 3 *THREE.Quaternion()* objektus. Sukuriamas Three.js *QuaternionKeyFrameTrack* objektas. Jame yra nurodoma, kuris objekto atributas bus animuojamas (*.quaternion*, tai VRML atitinka *rotation* lauką). Taip pat jame nurodomi ir animacijos sekundžių intervalai ir naujos atributo parametrų vertės - kvaternijonų vektoriai. Sukuriamas animacijos klipas, kuriame nurodomas raktinius takelius talpinantis masyvas ir animacijos trukmė - 2 sekundės. Paskutinėje stadijoje sukuriamas animacijos mikseris, kuriame nurodomas animuojamas objektas. Taip pat sukuriamas animacijos veiksmas, pradedantis vykdyti animaciją. Galiausiai vartotojas mato interneto naršyklėje kubą, kuris per 2 sekundes apsisuka aplink z ašį.

4.3. VRML animacijų konvertavimas

Šiuo metu VRML Three.js esantis konverteris nekonvertuoja animacijų. Todėl reikia sukurti algoritmą, kuris interpretuotų VRML mazgus susijusius su animacijomis ir pagal juos sukurtų Three.js animacijos objektus.

Sukurto VRML animacijų konvertavimo į WebGL algoritmo schema pateikta D priede. VRML konvertavimas prasideda nagrinėjant HTML elementus ir konvertuojant juos į JSON elementus. Kiekvienam į JSON konvertuotam VRML mazgui yra priskiriama grupė. Visi JSON elementai, kuriems buvo priskirta animacijos grupė, nagrinėjami atskirame konvertavimo algoritme. Šiame magistriniame darbe VRML mazgai buvo priskirti animacijų grupei ir konvertuoti į WebGL:

- timesensor - laiko sensorius.
- touchsensor - paspaudimo sensorius.
- planesensor - plokštumos sensorius.
- route - maršrutas.
- audioclip - garso klipas.
- orientationInterpolator - orientacijos interpoliatorius.
- coordinateInterpolator - koordinačių interpoliatorius.
- positionInterpolator - pozicijos interpoliatorius.

- colorInterpolator - spalvų interpoliatorius.
- scalarInterpolator - tempimo interpoliatorius.

Iš šių mazgų matyti, kad konvertuojamos VRML animacijos, kurios pradeda veikti dėl laiko, pelės užvedimo ar paspaudimo ant 3D objekto. Konvertuoti visi interpoliatoriai, išskyrus vieną - normalių interpoliatorių, kuris naudojamas keisti objekto medžiagos reagavimą į šviesą. Visi animacijose naudojami mazgai turi DEF raktažodį, todėl šiuos konvertuotus objektus išsaugome specialiame animacijų masyve, dėl greitesnio animacijų objektų pasiekimo. Trumpai aprašysime ir paaiškinsime animacijos konvertavimo veiksmų schemą (D priedas):

- 1. Animacijų konvertavimo algoritme nagrinėjami tik tie VRML elementai, kurie buvo priskirti animacijų grupei.
- 3. Interpoliatoriai ir laiko sensoriai išsaugomi globaliame animacijų kintamųjų masyve. Šio masyvo vertės pasiekiamos pagal DEF raktažodį. Pagal interpoliatoriaus tipą rakto vertės išsaugomos naujuose kintamuosiuose. Pavyzdžiui jeigu konvertuojamas orientacijos interpoliatorius - bus animuojamas objekto sukimasis, todėl galime iš anksto perskaičiuoti rakto vertes į Three.js kvaternionų vektorius.
- 3-4. Jeigu elemento tipas yra garso klipas, sukuriamas ir kintamajame išsaugomas Three.js garso grotuvas. Jeigu elemento tipas yra tempimo arpa paspaudimos sensorius, prie šio objekto papildomai išsaugomas tėvinio elemento Id numeris. Tai yra reikalinga, kad žymekliui judant ekrane žinotume, kada pradėti vykdyti su šiais mazgais susijusias animacijas.
- 6-8. Kiekvienas maršrutas turi nuorodas į 2 objektus. Vienas objektas siunčia įvykių jutiklį, o kitas jį priima. Pagal DEF raktažodį yra ieškoma ar šie VRML mazgai jau yra sukurti kaip Three.js objektai. Radus abu objektus, algoritmas tęsia darbą. Neradus bent vieno iš dviejų objektų, šis maršrutas yra įrašomas į specialų masyvą, pakartotiniam paleidimui. Tokie pakartotiniai maršrutai yra peržiūrimi pačioje pabaigoje, kuomet konvertavimo algoritmas būna konvertavęs visus galimus VRML objektus. Toks funkcionalumas reikalingas, nes maršrutuose gali būti susiejami mazgai, kurie dar nebuvo kaip objektai sukurti VRML konverterio.
- 9. Tikriname koks yra maršrute nurodyto įvykio jutiklio siuntėjo (eventOut) elemento tipas.
- 10. Elemento tipas - laiko sensorius. Konvertuojamose animacijose laiko sensorius visuomet susietas su interpoliatoriumi. Todėl pagal laiko sensoriuje nurodytą animacijos trukmę ir interpoliatoriaus rakto verčių reikšmes sukuriame laiko trukmės masyvą, kuris vėliau bus naudojamas raktinių takelių kūrime. Taip pat prie interpoliatoriaus išsaugome ir kitus svarbius laiko sensoriaus parametrus - pasikartojamumą, laiko pradžią, laiko pabaigą ir t.t.
- 11. Elemento tipas - paspaudimo arba plokštumos sensorius. Šiems elementams sukuriame jutiklius, kurie fiksuoja kompiuterio pelės judėjimą ir mygtukų paspaudimus. Prie šių elementų taip pat yra išsaugota jų tėvinių elementų id. Todėl judinant kompiuterio pelę, yra nuolatos sekama virš kokių Three.js objektų juda žymeklis. Jeigu žymeklis nuvestas prie objekto, su kuriuo susieta animacija, animacija pradeda veikti. Animacijos paleidimas arba sustabdymas vyksta panaudojant animacijos mikserį, kuris buvo susietas su sensoriumi.
- 12. Jeigu elementas yra interpoliatorius, sukuriamas *KeyFrameTrack* objektas. Koks parametras bus animuojamas yra užkoduota maršruto pavadinime. Three.js turi 6 skirtingus raktinius tekelius (KeyFrameTrack). Algoritme takelis parenkamas atsižvelgus į tai, koks parametras bus animuojamas. Pavyzdžiui, jei animuojamas sukimasis, yra kuriamas *VectorKeyframeTrack* takelis. Jeigu animuojamas objekto mastelio kitimas, yra naudojamas *NumberKeyframeTrack* takelis.

- 13. Šioje algoritmo stadijoje yra žinomas animuojamas objektas ir raktiniai takeliai. Sukuriamas animacijos mikseris, kuriame nurodoma koks objektas yra animuojamas. Jeigu laiko sensoriuje nurodyta pradžios trukmė (startTime) yra didesnė negu animacijos sustojimo trukmė (stopTime), animacija pradedama vykdyti iš karto. Priešingu atveju animacija sukuriama, tačiau nepaleidžiama. Tuomet yra laukiama, kol ją paleis vartotojo pelės paspaudimas arba žymeklio nutempimas virš tam tikrų WebGL objektų. Visi animacijų mikseriai išsaugomi animacijų kintamajų masyve, todėl yra greitai pasiekiami ir juose esanti animacija gali būti lengvai paleidžiama arba sustabdoma.

Kuriant ir testuojant animacijos konvertavimo algoritmą buvo naudojamas VRML failas, kuriame pavaizduoti pagal muziką šokantys 2 žmogeliukai (Dance.wrl). Animacija aktyvuojama paspaudus su žymekliu ant vieno iš žmogeliukų. Šis failas buvo pasirinktas, nes jame yra dažniausiai VRML animacijoms naudojami mazgai - laiko sensorius, paspaudimo sensorius, garso klipas ir įvairūs interpoliatoriai. VRML failo geometriją, tekstūras ir animacijas pavyko konvertuoti į WebGL, gautas rezultatas buvo peržiūrimas ir testuojamas naudojant Chrome interneto naršyklę (20 pav.).

20 pav. Konvertuota VRML animacija.

5. Navigacijos konvertavimas

Navigacija yra svarbi 3D pasaulio savybė. Nuo jos priklauso kaip vartotojas galės peržiūrėti sukurtus objektus. Šiame skyriuje aptarsime du su navigacija susijusius VRML mazgus - navigacijos informaciją (NavigationInfo) ir peržiūros taškus (ViewPoints).

5.1. Navigacijos informacija

VRML informacija apie navigaciją yra saugoma *NavigationInfo* mazge. Šio mazgo lauke *Type* yra nurodomas navigacijos tipas, kuris ir lemia kaip vartotojas naviguos sukurtame 3D pasaulyje. Skirtingos VRML naršyklės gali pasiūlyti ir daugiau navigacijos tipų, bet tik šie žemiau išvardinti keturi yra oficialūs:

- WALK - numatytasis VRML navigacijos tipas. Vartotojas gali apžiūrėti objektus ir judėti 3D pasaulyje naudodamasis klaviatūros rodyklių mygtukus.
- FLY - tipas, skirtas imituoti skraidymo efektą. Tokiu būdu žymeklį judinant aukštyn, objektai yra pritraukiami. Žymekliui judant žemyn, vaizdas yra tolinamas nuo vartotojo.
- NONE - navigacija yra išjungiama. Norint peržiūrėti objektą iš kitos perspektyvos, reikia naudoti peržiūros taškus.
- EXAMINE - naudojant šį navigacijos tipą, galima kameros atžvilgiu pasukti objektą. Tai yra patogus ir dažnai naudojamas navigacijos tipas, leidžiantis apžiūrėti sukurtą sceną iš bet kurio 3D pasaulio taško.

Three.js bibliotekoje navigacija sukuriama animuojant kameros judėjimą scenoje. Kameros animacija susiejama su žymeklio judėjimu ekrane ar klaviatūros mygtukų paspaudimais. Three.js turi 11 skirtingų kameros kontroliavimo ir animavimo klasių, todėl konvertuodami failą, pagal VRML navigacijos tipą parenkame reikalingą Three.js klasę.

Šiame darbe tiksliausiai konvertuojamas *EXAMINE* navigacijos tipas, nes jis yra patogiausias naudoti mūsų konvertuojamuose VRML failuose. Konvertuodami šį tipą, naudojome Three.js *TrackballControls* klasę, nes jos sukuriama navigacija yra panašiausia į naudojamą VRML. Testuojant kameros animaciją buvo pastebėta, kad VRML *EXAMINE* navigacijos tipas yra patogesnis negu sukuriamas Three.js bibliotekos. Dėl to konvertavimo algoritme *TrackballControls* buvo patobulintas. *TrackballControls* tašką, kurio atžvilgiu yra sukama kamera visuomet parenka kaip koordinačių pradžios tašką [0, 0, 0]. Tuo tarpu VRML navigacijoje kameros sukimo taškas parenkamas kaip visos 3D pasaulio scenos centras, kuris ne visuomet sutampa su koordinačių pradžios tašku. Toks funkcionalumas buvo sukurtas ir konvertavimo algoritme. Konvertavus visus objektus yra suskaičiuojamos sukurto 3D pasaulio centro koordinatės, ir šis taškas nustatomas kaip pradinis kameros sukimosi taškas. Taip pat sukurtas funkcionalumas, kuris leidžia klaviatūros mygtuko paspaudimu parinkti naują kameros žiūrėjimo tašką. Tai palengvina navigaciją, kuomet yra peržiūrimas ne vienas, o daugelis konvertuotų objektų.

5.2. Peržiūros taškai

Naudojantis peržiūros taškais galima nurodyti poziciją erdvėje, iš kurios vartotojas peržiūri 3D sceną. Tai naudingas VRML mazgas, nes jo pagalba galima iš anksto parinkti taškus erdvėje, iš kurių turėtų būti apžiūrimas objektas. Viename VRML faile galima nurodyti daug peržiūros taškų. Kamera iš vieno peržiūros taško į kitą perkeliama naudojantis naršyklės mygtukais. Paspaudus

mygtuką vartotojas mato kameros animaciją, perkeliant ją iš vienų koordinačių į kitas. VRML peržiūros taško mazgas ir numatytosios laukų vertės pavaizduotos 21 pav.

```
Viewpoint {
        fieldOfView 0.785398
        position 0 0 10
        orientation 0 0 1 0
        description ""
        jump TRUE
}
```

21 pav. Peržiūros taškų mazgas ir numatytosios vertės

fieldOfView laukas parodo peržiūros kampą radianais. Tai yra atitikmuo kameros objektyvo židinio nuotoliui. *Position* - tai pozicija erdvėje, iš kurios bus peržiūrima sukurta 3D pasaulio scena. *Orientation* parodo kuria kryptimi yra nukreipta kamera. *Description* leidžia suteikti peržiūros taškui pavadinimą. *Jump* parodo kaip animuojamas kameros perėjimas iš vieno peržiūros taško prie kito. Jeigu *Jump* yra *Taip*, vartotojas mato sklandžią kelių sekundžių trukmės animaciją. Priešingu atveju kamera perkeliama prie kito peržiūros taško akimirksniu.

Konvertuoti VRML peržiūros taškus naudojama Tween.js biblioteka. Ši biblioteka pasirinkta dėlto, nes yra skirta kurti sklandžioms perėjimo animacijoms. Todėl puikiai tiko konvertuoti VRML peržiūros taškus. Visi VRML faile esantys peržiūros taškų elementai buvo išsaugomi ir sunumeruojami peržiūros taškų masyve. Kameros pozicijos koordinatės nustatomos panaudojant pirmo surasto peržiūros taško mazgo informaciją. Naršyklės lange sukuriami du mygtukai, kurie leidžia perjungti kamerą iš vieno peržiūros taško į kitą. Vartotojui paspaudus mygtuką, masyve surandamas sekantis elementas, kuriame saugomos naujos kameros koordinatės ir peržiūros kampas. Radus elementą, sukuriama ir iš karto įvykdoma vieną kartą pasikartojanti Tween.js animacija. Sukurti šią animaciją reikia žinoti pradinę ir galutinę taško padėtį. Pradinė padėtis randama žinant dabartinę kameros poziciją erdvėje. Galutinės vertės surandamos iš surasto masyvo elemento. Užpildžius parametrus, nurodomas animuojamas kameros objektas ir animacija paleidžiama veikti akimirksniu. Vartotojas mato sklandžią kameros animaciją nuo vieno peržiūros taško koordinačių prie kito.

6. Three.js ir X3DOM konverterių palyginimas

Kaip buvo minėta 2 skyriuje, šiuo metu konvertuoti ir iškart interneto naršyklėje atvaizduoti VRML ir X3D failus galima naudojantis X3DOM arba Three.js bibliotekomis. Kadangi X3DOM konvertuoja daugiau mazgų nei Three.js, palyginsime šiame magistro baigiamajame darbe sukurtą konverterį su X3DOM konverteriu.

6.1. Geometrijos ir tekstūrų palyginimas

Siekiant palyginti du algoritmus, konvertuosime VRML failą, kuriame atvaizduojamas berniukas (Berniukas.wrl). Šis failas buvo pasirinktas dėl to, nes yra sudarytas tik iš indeksuotų sienų aibės mazgų, kuriuose yra tekstūrų ir normalių vektorių laukai. Tai pats svarbiausias ir dažniausiai naudojamas mazgas, kurį konvertuoja abu lyginami algoritmai. Berniuko 3D objektas taip pat pasirinktas dėlto, nes yra labai detalus, tekstūrai naudojamas aukštos rezoliucijos paveiksliukas. Todėl konvertavus šį failą į WebGL, jis bus sudarytas iš daugybės mažesnių objektų ir trikampių. Sukurtos 3D scenos greitaveikai interneto naršyklėje didelę įtaką turi konvertavimo algoritmų sukuriamos šešėliuoklių programos, kurios yra siunčiamos į vaizdo plokštę. Skirtingas šešėliuoklių programų sukūrimas turėtų būti vienas iš pagrindinių lyginamų algoritmų skirtumų. X3DOM ir Three.js konverterių greitaveiką lyginsime matuodami objekto konvertavimo laiką ir naršyklės atkuriamų kadrų per sekundę skaičių algoritmams baigus darbą. Greitaveika yra svarbu, nes dažniausiai 3D scenoje yra atvaizduojamas ne vienas 3D objektas, o daugybė. Vartotojai gali neigiamai vertinti aplikaciją, jeigu atkuriamų kadrų per sekundę skaičius bus mažesnis nei 30, ir vartotojas matys sukurtos 3D scenos trūkinėjimą.

Konvertuojamas failas pavaizduotas 22 pav. Kairėje matyti kaip atrodo berniukas jį peržiūrint specialiai VRML failams peržiūrėti skirta BS Contact programa. Centre - pavaizduotas rezultatas, gautas naudojant sukurtą Three.js konverterį. Dešinėje - konvertavus su X3DOM konverteriu.

22 pav. BS contact (kairėje), Three.js (centre) ir X3DOM konverterių sukurti 3D objektai

Kaip matyti iš 22 pav., visų konvertuotų 3D objektų geometrijos atvaizduojamos vienodai, tačiau skiriasi tekstūros. X3DOM objektas pasižymi didesniu blizgesiu ir šviesos atspindžiais. Tokius skirtumus lėmė skirtingai algoritmuose implementuotos medžiagas sukuriančios šešėliuoklių programos. VRML kalboje yra naudojama tik viena medžiaga, kurios išvaizda nusakoma naudojant spalvų, permatomumo, blizgesio, atspindžio, šviesos intensyvumo laukus. Kaip matyti,

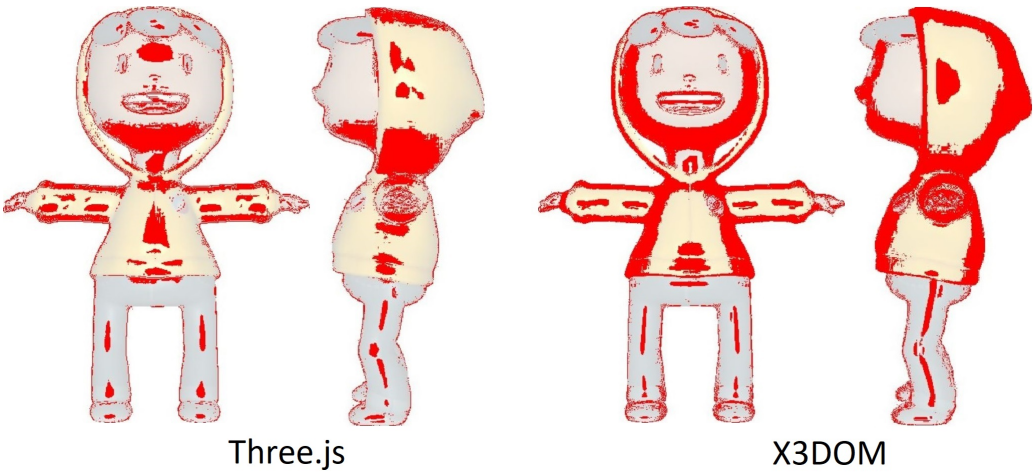
galimų varijuoti parametrų skaičius nėra didelės, todėl sukurti įvairiomis savybėmis pasižyminčias medžiagas yra sunku. Tuo tarpu Three.js biblioteka leidžia naudoti daug skirtingų medžiagų, šio darbo rašymo metu jų buvo 16 [10]. Visgi šiame darbe visiems objektams panaudota ta pati *MeshPhongMaterial* medžiaga. Taip buvo pasielgta dėl to, nes savo savybėmis ji buvo panašiausia į naudojamą VRML. Tačiau ši medžiaga pasižymi tikroviškesniu nei VRML reagavimu į šviesą, didesniu šešėlių generavimu. Šiuos skirtumus ir matome 22 pav.

Patikrinti sukurtų objektų panašumą galime naudodami vaizdų panašumo įvertinimo algoritmus. Šiame darbe panaudosime struktūrinio panašumo (Structural SIMilarity - SSIM) algoritmą [8]. SSIM algoritmas naudojamas norint sužinoti kiek vienas paveiksliukas yra panašus į kitą. Pagrindinis paveiksliukas gaunamas iš specialiai VRML failams peržiūrėti skirtos BS Contact programos. Lyginamieji paveiksliukai gaunami iš interneto naršyklės konvertavus failą su X3DOM ir sukurtu Three.js konverteriu. Algoritmo grąžinama reikšmė - SSIM indeksas, kuris gali įgyti reikšmes nuo 0 (visiškai skirtingi paveiksliukai) iki 1 (paveiksliukai identiški). Paveiksliukų lyginimo rezultatai pateikti 2 lentelėje. Matavimai buvo atliekami su Berniukas.wrl failu, sukurtą 3D objektą peržiūrint iš 3 pozicijų - priekio, nugaros ir šono.

2 lentelė. VRML failo (berniuko) SSIM indeksų palyginimas.

$Pozija$	$Three.js$	$X3DOM$
Iš priekio	0.9429	0.9246
Iš šono	0.9117	0.9078
Iš nugaros	0.9530	0.9468

Kaip matyti iš 2 lentelės rezultatų, gauti SSIM indeksai yra didesni nei 0.9. Tai reiškia, kad konverterių sukurtas ir naršyklėse atvaizduojamas berniukas yra panašus į sukuriamą specialiai VRML failams peržiūrėti skirta programa. Kad konvertuojami objektai yra panašūs galime matyti ir 22 pav. Taip pat 2 lentelės rezultatuose matyti, kad sukurto Three.js konverterio SSIM indeksų reikšmės yra didesnės. Tai reiškia, kad struktūriškai sukurtas Three.js paveiksliukas yra artimesnis BS Contact sukuriamam vaizdui nei X3DOM konverteris. Tačiau šis skirtumas yra labai nedidelis. Gautas 0.9 rezultatas yra geras, ir siekti gauti SSIM indeksą artimesnį vienetui nėra būtinybės. Taip yra dėl to, nes VRML failas lyginamas tik su viena VRML failams peržiūrėti skirta programa (BS Contact). Tuo tarpu peržiūrėti VRML failams yra sukurta ne viena programa, ir tas pats VRML failas skirtingose programose nėra identiškas. Pavyzdžiui, jeigu lygintume tą patį objektą skirtingose VRML naršyklėse, taip pat negautumėme SSIM indekso, lygaus 1. Todėl sukurti Three.js konverterį, kuris konvertuotų VRML failus su SSIM indeksu lygiu 1 (lyginant su BS Contact programa) nėra būtinybės. SSIM algoritmas visus lyginamus paveiksliukus konvertuoja į pilką paletę. Pavyzdžiui, jeigu palygintume spalvotą BS Contact sukurto berniuko paveiksliuką su juodai baltu Three.js sukurtu vaizdu, gautume SSIM indeksą lygų 0.9427. Tai beveik tas pats rezultatas, gautas palyginus spalvotus paveiksliukus (0.9429). Taigi SSIM algoritmas neįvertina visų RGB paletės spalvų. Todėl siekiant įvertinti Three.js ir X3DOM konverterio spalvas, lyginsime kiekvieno paveiksliuko pikselio spalvų skirtumus. Jeigu pikselio spalva nesutampa su BS Contact naršyklės sugeneruoto paveiksliuko pikseliu daugiau nei 10 %, toks pikselis bus nudažomas raudona spalva. Tai leis palyginti konvertavimo algoritmų atkuriamų spalvų panašumą su BS Contact programa. Gauti rezultatai pateikti 23 pav.



23 pav. Three.js (kairėje) ir X3DOM konverterių sukuriamų objektų spalvų palyginimas su BS Contact naršykle. Raudona spalva žymi didžiausius skirtumus.

Iš 22 pav. ir 23 pav. matyti, kad sukurto Three.js konverterio didžiausias spalvų neatitikimas yra objekto vietose, kurios atspindi šviesą. Šviesos atspindėjimas sukuriamas VRML faile nurodant normalių vektorių padėtis. Taigi sukurtas konverteris nevisiškai tiksliai atkuria 3D objekto šviesos atspindžius. Tuo tarpu X3DOM konverteris šviesos atspindžius konvertuoja tiksliau. Vietose, kuriuose objekte sukuriamas šviesos atspindys, nėra raudonų žymių. Tačiau X3DOM konverteris netiksliai konvertuoja medžiagos šešėlius. VRML faile, peržiūrimame BS Contact naršykle (21 pav.), matyti, kad objekto šonuose sukuriamas šešėlis. Jis sugeneruojamas dėl medžiagos reagavimo į šviesos šaltinį, kuris yra priešais 3D objektą. Kaip matyti iš X3DOM sukuriamo objekto, jo kraštuose nėra šešėlių (22 pav.), todėl 23 pav. matyti raudoni pažymėjimai objekto kraštuose. Tokių didelių raudonų žymių nėra Three.js sukurtame objekte, todėl galime sakyti, kad Three.js konverterio sukurtas konverteris tiksliau atkuria medžiagos šešėlius negu X3DOM konverteris. Palyginus abu 23 pav. pavaizduotus objektus, matome kad didesnė dalis raudonai nudažytų pikselių yra X3DOM konverterio paveiksliuke. Todėl sukurtas Three.js konverteris spalvas atkūrė panašesnes į BS Contact programos nei X3DOM konverteris.

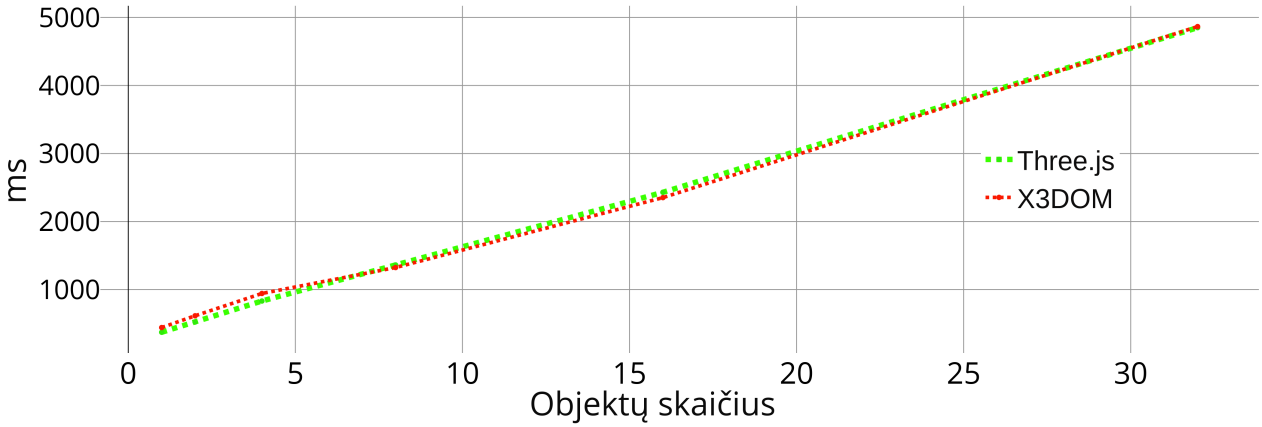
Konvertavimo algoritmą galima patobulinti naudojant ir kitas medžiagas, pvz. *MeshPhysicalMaterial* ar *MeshStandardMaterial*, tačiau šių medžiagų naudojimas reikalauja didesnio kompiuterio darbo ir resursų, o tai sumažina naršyklėje atkuriamų kadrų per sekundę skaičių. Tuo tarpu *MeshPhongMaterial* pasižymi labai gera greitaveika, nes ją generuojantys algoritmai yra labai efektyvūs. Labai gerą *MeshPhongMaterial* medžiagos kūrimo greitaveiką galėjo lemti faktas, kad šis rasterizavimo algoritmas buvo pristatytas jau 1973 metais, ir nuo to laiko nuolat tobulinamas. Todėl ši Three.js bibliotekos naudojama medžiaga yra gerai optimizuota ir nereikalauja didelių resursų. Kaip atrodytų konvertuojamas berniuko VRML failas naudojant *MeshBasicMaterial* medžiagą, kuri nereaguoja į šviesą arba *MeshPhysicalMaterial*, kuri yra labai populiari žaidimų kūrimo programose, pvz. Unity, Unreal Engine 4 ar 3D Studio Max galime pamatyti 24 pav. Skirtingų Three.js medžiagų automatinį panaudojimą VRML konvertavime galima būtų panaudoti tikroviškesniam VRML objektų sukūrimui ir naujų savybių pridėjimui. Tačiau ši galimybė šiame darbe nebus nagrinėjama ir visiems konvertuojamiems failams yra naudojama *MeshPhongMaterial* medžiaga.

(a) MeshBasicMaterial

(b) MeshStandartMaterial

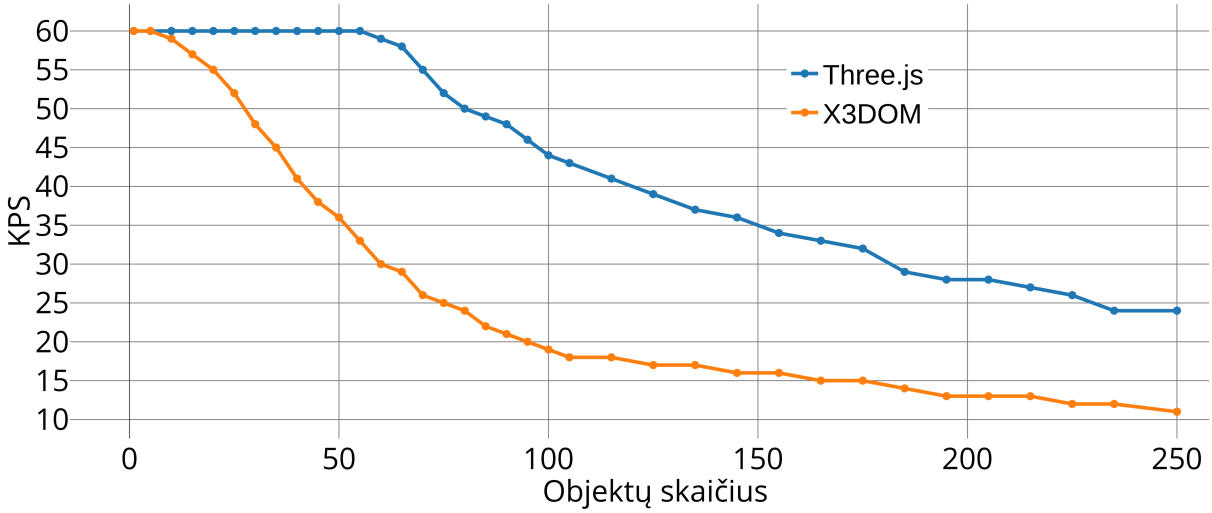
24 pav. Skirtingų medžiagų panaudojimas Three.js bibliotekoje

Siekiant palygint X3DOM ir Three.js konverterius, buvo matuojamas laikas, per kurį abu algoritmai gali konvertuoti nagrinėjamus berniukus ir atvaizduoti juos interneto naršyklėje. Gauti rezultatai pateikti 25 pav. Berniukų skaičius buvo didinamas nuo 1 iki 32. Vienas berniuko objektas užėmė 1.5 MB failą. Todėl testuojant algoritmų greitį su 32 berniukais, algoritmai turėdavo nuskaityti ir išanalizuoti 48 MB tekstinį failą. Kiekvienas matavimas buvo atliekamas 10 kartų, o 25 pav. grafike pavaizduotas gautų duomenų vidurkis. Kaip matyti iš grafiko, abu konvertavimo algoritmai konvertuodavo VRML failus per beveik tokį patį laiko tarpą. Esant nedideliam failui, (mažam objektų skaičiui) X3DOM konverteris buvo nežymiai greitesnis. Tačiau didėjant konvertuojamų objektų skaičiui, abu konvertavimo algoritmai konvertuoja VRML failus per beveik tokį patį laiko tarpą. Todėl galime sakyti, kad šiuo požiūriu nė vienas algoritmas neišsiskyrė. Taip pat matyti, kad failų konvertavimo trukmė yra tiesinė. Tai reiškia, kad algoritmų veikimo trukmė priklauso tik nuo konvertuojamo failo dydžio, bet ne nuo kuriamų WebGL objektų skaičiaus. Failų konvertavimai ir matavimai buvo atliekami naudojantis *Chrome* interneto naršykle.



25 pav. Algoritmo veikimo trukmės (milisekundėmis) priklausomybė nuo konvertuojamų objektų skaičiaus. Žalia spalva pažymėta sukurto konverterio laikinė priklausomybė. Raudona spalva pažymėtas X3DOM algoritmo laikas.

VRML failo konvertavimo laikas svarbus, tačiau dar svarbesnė savybė yra sukurto WebGL objekto greitaveika - naršyklės atkuriamų kadrų per sekundę skaičius. Tai priklauso nuo konvertavimo algoritmų kuriamų šešėliuoklių programų. Kuo sklandžiau ir optimaliau parašytos šios programos, tuo mažiau skaičiavimų turi atlikti vaizdo plokštės procesoriai ir naršyklėje pasiekiamas didesnis kadrų per sekundę skaičius. Greitaveikos matavimo rezultatai naudojant *Chrome* interneto naršyklę pavaizduoti 26 pav. Konvertuojami objektai yra anksčiau minėti berniukai.



26 pav. Chrome naršyklės atkuriamų kadrų per sekundę skaičiaus priklausomybė nuo konvertuojamų objektų skaičiaus. Mėlyna spalva pažymėta sukurto konverterio greitaveika. Oranžine spalva pažymėti X3DOM algoritmo rezultatai.

Objektų skaičius greitaveikos matavimuose buvo keičiamas nuo 1 iki 250. Visi konvertuoti berniukai taip buvo animuojami, sukant juos aplink Z ašį 2 Hz dažniu. Taip buvo pasielgta norint sukurti tikroviškesnes 3D pasaulio sąlygas. Kaip matyti iš 26 pav. pateiktų rezultatų, sukurtas Three.js konverteris pasižymi geresne greitaveika nei X3DOM algoritmas. Naudojant sukurtą algoritmą greitaveika pradėjo mažėti tik pasiekus 60 konvertuojamų berniukų skaičių. Tuo tarpu X3DOM konverteris galėdavo maksimaliu 60 KPS dažniu atvaizduoti tik 10 objektų. Taip pat verta paminėti, kad matavimai buvo atliekami nejudinant kompiuterio pelės, ir nepakeičiant kameros žiūrėjimo taško. Pradedant judinti žymeklį į įvairias kryptis, X3DOM sukurtų WebGL objektų greitaveika sumažėdavo vidutiniškai dar 10 KPS. Tuo tarpu naudojant Three.js konverterį, kadrų per sekundę skaičius sumažėdavo tik 1-2 vienetais. Tai reiškia, kad Three.js bibliotekoje yra geriau veikianti ir optimizuota vartotojo navigacija, kuri užtikrina nesikeičiantį kadrų per sekundę skaičių net aktyviai judinant žymeklį ir taip keičiant vartotojo kameros poziciją erdvėje. Tai naudinga savybė, jeigu norėtume konvertuoti VRML formatu sukurtą žaidimą ir judėti 3D pasaulyje.

Egzistuoja daug programų, kuriose galima peržiūrėti įvairius 3D objektus. Pavyzdžiui, naudojant nemokamą Jmol programą, galima peržiūrėti įvairias molekules. Ši programa taip pat suteikia galimybę atvaizduojamus objektus išeksportuoti į VRML ar X3DOM formato failus [4]. Todėl naudodami sukurtą Three.js konverterį galime peržiūrėti molekules interneto naršyklėje. Palyginsime sukurto Three.js ir X3DOM konverterių greitaveiką atvaizduojant įvairias biomolekules. Informacija apie molekules gauta iš baltymų duomenų bazės [1]. Gauti matavimų rezultatai pateikti 3 lentelėje. Pirmajame stulpelyje nurodytas PDB (Protein Data Bank) numeris, leidžiantis identifikuoti molekulę. Antrame stulpelyje nurodomas molekulės struktūros svoris. Kuo šis svoris didesnis, tuo daugiau atomų ir jungčių sudaro molekulę. Pavyzdžiui, molekulę su PDB numeriu 6A0C sudaro 587 atomai, tuo tarpu molekulę su PDB numeriu 6FS1 sudaro 2517 atomai. Trečiame ir ketvirtame stulpelyje - naršyklės atkuriamų kadrų per sekundę skaičius, matuojamas peržiūrint molekulę Chrome interneto naršykle ir judinant pelės kursorių, tuo pačiu ir molekulę ekrane.

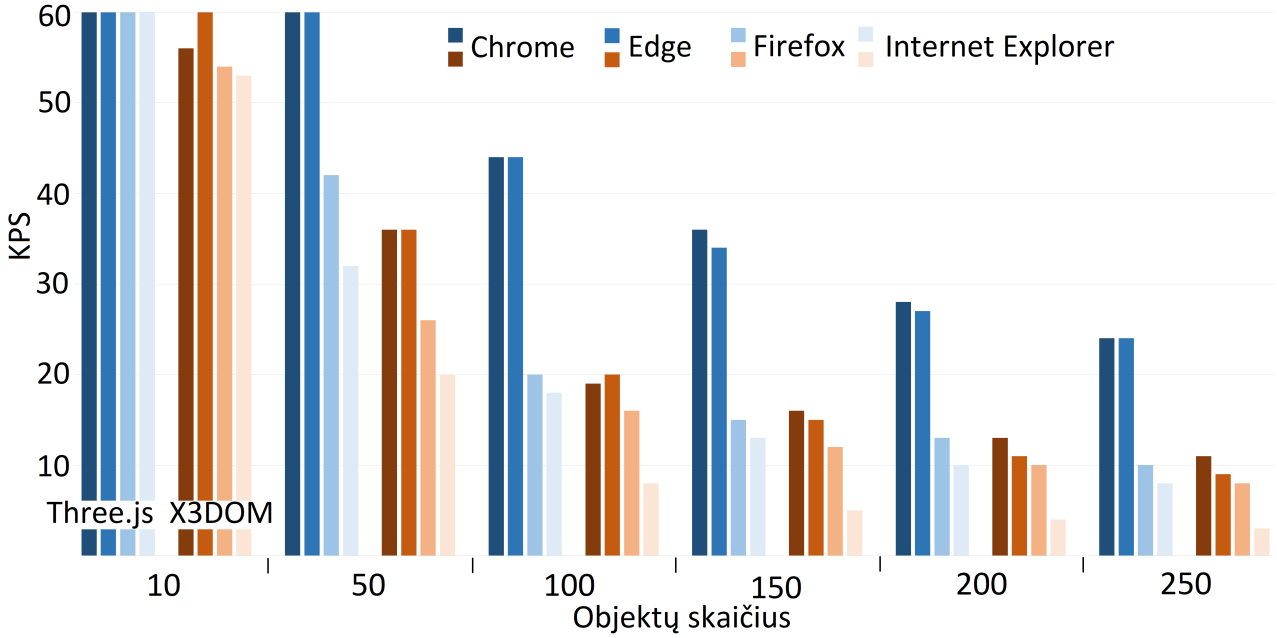
3 lentelė. Molekulių atvaizdavimo Chrome interneto naršyklėje greitaveika.

PDB	*Svoris*	*Three.js*	*X3DOM*
6A0C	8386.04	43	20
3BSE	11510.14	34	12
6H06	25312.37	22	6
6FS1	35484.50	14	4
6A83	45624.92	11	3

Kaip matyti iš 3 lentelės rezultatų, didėjant molekulės struktūros svoriui, abiejų konverterių KPS pradeda mažėti. Tai yra priimtinas rezultatas, nes didėjant svoriui molekulę sudaro vis daugiau objektų. Iš gautų rezultatų taip pat matyti, kad peržiūrint tą pačią molekulę su sukurtu konverteriu, pasiekiama geresnė greitaveika. Todėl su Three.js konverteriu galima atvaizduoti didesnio struktūrinio svorio molekules nei su X3DOM konverteriu.

Šiuo metu visos populiariausios interneto naršyklės palaiko WebGL technologiją. Tačiau skirtingose naršyklėse WebGL palaikymo lygis ir greitaveika skiriasi. 27 pav. pateikti greitaveikos matavimai naudojant įvairias interneto naršykles.

Melsvi stulpeliai žymi sukurto Three.js konverterio matavimus. Rusvi stulpeliai rodo X3DOM algoritmo greitaveiką. X ašyje yra pažymėtas konvertuojamų objektų (berniukų) skaičius, kuris buvo 10, 50, 100, 150, 200 ir 250. Y ašyje parodytas naršyklės atkuriamų kadrų per sekundę skaičius. Kaip matyti iš gautų rezultatų, didžiausias KPS buvo gaunamas naudojantis *Chrome* ir *Edge* interneto naršykles. O mažiausia greitaveika pasiekiama su *Internet Explorer* naršykle. Taip pat verta pastebėti, kad sukurto Three.js konverterio greitaveika buvo didesnė nei X3DOM visose lyginamose naršyklėse. Taigi geresnę greitaveiką naudojant sukurtą Three.js konverterį galima pasiekti nepriklausomai kokią interneto naršyklę naudoja vartotojas.



27 pav. Chrome, Edge, Internet Explorer, Firefox naršyklių greitaveika naudojant sukurtą Three.js ir X3DOM konverterius.

6.2. Animacijų palyginimas

Kaip jau minėjome anksčiau, Three.js bibliotekos numatytasis konverteris nekonvertuoja jokių animacijų, o X3DOM konverteris konvertuoja tik tam tikras animacijas. Šiame darbe sukurtas konverteris konvertuoja pagrindines ir dažniausiai naudojamas animacijas - sukurtas su interpoliatoriais. Taip pat konvertuojamos animacijos, kurias sukuria laiko, paspaudimo ar plokštumos sensoriai. Siekiant patikrinti X3DOM ir sukurto konverterio galimybes, konvertavome VRML failus, kuriuose naudojamos specifinės animacijos, ir žiūrėjome kaip į WebGL jas konvertuoja lyginami algoritmai. Gauti rezultatai pateikiami 4 lentelėje.

4 lentelė. Sukurto Three.js ir X3DOM algoritmų animacijos konvertavimo palyginimas.

Animacija	*Three.js*	*X3DOM*
Koordinačių interpoliatorius	Taip	Taip
Normalių interpoliatorius	Ne	Taip
Orientacijos interpoliatorius	Taip	Taip
Pozicijos interpoliatorius	Taip	Taip
Mastelio interpoliatorius	Taip	Taip
Spalvų interpoliatorius	Taip	Taip
Sukimosi animacijos	Taip	Taip
Skaidrumo animacija	Taip	Ne
Kursoriaus sekimo animacija	Taip	Ne
Paspaudimo sensoriaus animacijos	Taip	Ne
Plokštumos sensoriaus animacijos	Taip	Ne
Laiko sensoriaus animacijos	Taip	Taip
Garso animacija	Taip	Ne

6.3. Navigacijos konvertavimas

Kaip jau buvo minėta 5 skyriuje, VRML galimi 4 navigacijos tipai, iš kurių patogiausias ir dažniausiai naudojamas yra *Examine*. Žemiau pateiktoje lentelėje palyginamos X3DOM ir Three.js algoritmų navigacijos konvertavimo galimybės.

5 lentelė. Sukurto Three.js ir X3DOM algoritmų navigacijos palyginimas

Navigacija	*Three.js*	*X3DOM*
NONE	Taip	Taip
FLY	Taip	Taip
WALK	NE, sukurtas tik prototipas	NE
EXAMINE	Taip	Taip
Peržiūros taškai	Taip	Ne

Kaip matyti iš 5 lentelės duomenų, X3DOM negali konvertuoti *WALK* navigacijos tipo ir peržiūros taškų. Peržiūros taškai yra svarbi ir dažnai naudojama VRML savybė, todėl juos konvertuoti svarbu. Sukurtame konverteryje peržiūros taškai yra perjungiami dviem mygtukais. Šio funkcionalumo veikimas yra pristatytas 5.2 skyriuje.

6.4. Išsaugojimas kitu formatu

WebGL nėra 3D failo formatas, tai yra technologija, kuria naudojantis galima interneto naršyklėje atvaizduoti interaktyvę trimatę grafiką. Todėl kuomet konvertuotas VRML failas yra peržiūrimas interneto naršyklėje, sukurti 3D objektai egzistuoja tik iki kol vartotojas neuždaro savo interneto naršyklės. Vis dėlto yra konverterių, kurie visą sukurtą WebGL sceną gali išsaugoti kitais formatais, kuriuos vėliau galima importuoti ir iš naujo peržiūrėti interneto naršyklėse. Taip pat galima išsaugoti ir tokiais 3D formatais, kuriuos galima peržiūrėti ir trimatės grafikos kūrimo programomis, pvz. Blender ar 3D Max.

Vis dėlto, nors ir neoficialus, tačiau labai dažnai WebGL sukurtos scenos išsaugojimui yra naudojamas glTF (GL Transmission Format) formatas. Šis formatas pirmą kartą pasirodė 2011 metais. Jį sukūrė *Kronos group*, kuri taip pat yra organizacija, kurianti ir tobulinanti WebGL technologiją. Kaip JPG formatas yra naudojamas išsaugoti paveiksliukus, GIF - animuotus paveiksliukus, taip glTF formatas sukurtas peržiūrėti trimačius objektus. glTF buvo kuriamas manant kad tai bus oficialus WebGL išsaugojimo ir atkūrimo formatas [21]. Kitas populiarus būdas išsaugoti WebGL sceną yra naudojantis JSON failo formatą.

Three.js biblioteka siūlo keletą formatų, kuriais galima išsisaugoti sukurtus WebGL objektus. Tuo tarpu peržiūrint VRML failą X3DOM konverteriu, išeksportuoti naršyklėje rodomą vaizdą į kitus formatus nėra galima. Tai didelis šio konverterio minusas, nes išsaugojus VRML failus į glTF formatą juos galime peržiūrėti ir kitomis trimatės grafikos peržiūrėjimo programomis. Taip pat JSON ir glTF failų formatai yra populiarūs, tuo tarpu VRML yra vis rečiau naudojamas. Todėl esant galimybei konvertuotą VRML failą išsaugoti kitu formatu, galime jį ilgiau išlaikyti ir neprarasti.

Žemiau pateiktoje lentelėje palyginami konvertuotų VRML failų dydžiai su kitu formatu išsaugota WebGL scena.

6 lentelė. Konvertuotos VRML scenos išsaugojimas kitu formatu.

$VRMLfailas$	$VRML(KB)$	$JSON(KB)$	$glTF(KB)$	$glb(KB)$
Puodelis.wrl	31	108	285	212
Dangtis.wrl	103	887	599	448
Dance.wrl	172	450	268	184
N-cerkve.wrl	1498	30250	6237	4597
Astuonkojis.wrl	1519	6222	6802	5101
Berniukas.wrl	3144	6222	6802	5101
VU-kiemai.wrl	3926	34660	16279	11886
Skeletas.wrl	10184	46340	52157	39014
SkeletasNormales.wrl	19080	55051	52424	39214

3D sceną eksportuosime į JSON, glTF ir glb failus, nes šie formatai yra populiarūs, taip pat juose išsaugomos ne tik objektų geometrijos ir medžiagos, bet ir tekstūros. Naudojant Three.js galima išsaugoti sukurtus objektus ir kitais formatais, bet juose nėra saugoma visa scenos informacija. Pavyzdžiui, išeksportuojant į .stl ar .obj formatą nėra išsaugoma informacija apie tekstūras, apšvietimą ar kamerą.

Kaip matyti iš 6 lentelės duomenų, konvertuotą VRML failą išsaugant JSON, glTF ar glb formatu, jo dydis visuomet didesnis, ir gali padidėti keletą kartų. Daug kartų mažesnį VRML failo

dydį gali lemti DEF ir USE raktažodžių naudojimas, kurie leidžia iš naujo panaudoti jau sukurtus VRML objektus. Tai leidžia sutaupyti daug laiko ir smarkiai sumažina VRML failo dydį. Visgi išsaugojimas JSON, glTF ar glb formatu turi ir privalumų: yra žymiai daugiau programų, kurios gali nuskaityti šiuos failus. Taip pat VRML tekstūrų informacija yra saugoma atskiruose .jpg ar .png failuose. Tuo tarpu išsaugojus sceną JSON ar glTF formatu, visa informacija apie tekstūras yra saugoma viename faile, todėl nebereikia jaudintis dėl atskirų tekstūrų failų. Iš pateiktų duomenų taip pat matyti, kad .glb failas visuomet užima mažiau vietos nei .glTF. Taip yra todėl, nes .glb yra tos pačios .glTF struktūros, tik išsaugotas binariniu formatu. Tai leidžia sutaupyti vietos.

Išvados ir rekomendacijos

Šiame darbe atlikta VRML formato ir WebGL technologijos analizė bei pasirinkta Three.js biblioteka, su kuria sukurtas konvertavimo algoritmas. Taip pat išnagrinėtos ir kitos programos, kurių pagalba galima atvaizduoti trimatę kompiuterinę grafiką interneto naršyklėje. Sukurtas algoritmas išnagrinėja XML struktūros failą ir naudodamasis Three.js biblioteka ir WebGL technologija sukuria ir atvaizduoja 3D objektus. Testuojant algoritmo veikimą buvo konvertuojami įvairūs VRML failai, o tarp jų 3D scenos, vaizduojančios Vilniuje esančią Šv. Nikolajaus cerkvę, Vilniaus universiteto kiemelius. Programa taip pat buvo palyginta su kitu plačiai naudojamu X3DOM konverteriu. Įvertintas abiejų algoritmų veikimo laikas, sukurtų objektų greitaveika interneto naršyklėje. Taip pat geometrijų, tekstūrų, spalvų, animacijos bei navigacijos konvertavimo galimybės. Atlikus šį darbą, galime pateikti tokias išvadas ir rekomendacijas:

1. Šiuo metu nėra paprastų ir patogių įrankių, kurie leistų pilnai konvertuoti VRML ar X3D formato failus ir atvaizduoti juos interneto naršyklėje naudojantis WebGL technologija.
2. Sukurtas algoritmas, išnagrinėjantis VRML failą ir konvertuojantis svarbiausius mazgus, susijusius su geometrijomis, tekstūromis, animacijomis ir vartotojo navigacija.
3. Konvertuoti VRML failą į WebGL yra paprasčiau naudojant tarpinį X3D formatą. XML struktūros failas yra patogus analizei, nes galime panaudoti standartines DOM funkcijas.
4. Vertinant WebGL greitaveiką, buvo nustatyta, jog naršyklės atkuriamų kadrų per sekundę skaičius priklauso nuo sugeneruotų objektų skaičiaus. Tuo tarpu objektus sudarančių trikampių skaičiaus KPS turi labai nedidelę įtaką.
5. Objektą sudarančių trikampių skaičius turi įtaką išeksportuojamo WebGL failo dydžiui. Objektą sudarančių trikampių skaičių padidinus 6,25 karto, išeksportuojamo JSON failo dydis vidutiniškai padidėdavo 2,24 karto.
6. Objektų geometrijas naudinga kurti naudojant *BuferGeometry* klasę. Jos dėka su testuojamu VRML failu buvo galima naršyklėje atvaizduoti iki 50 % daugiau 3D objektų neprarandant maksimalios 60 KPS greitaveikos. Tačiau šios klasės naudojimas apsunkina geometrijų ir animacijų kūrimą bei reikalauja gilių žinių susijusių su 3D programavimu.
7. Sukurtas algoritmas VRML failus nuskaitydavo ir atvaizduodavo tokiu pačiu greičiu kaip ir X3DOM konverteris.
8. Sukurtas algoritmas objektus atvaizduodavo didesniu kadrų per sekundę skaičiumi nei X3DOM konverteris. Konvertuojant VRML failą su aukštos rezoliucijos tekstūromis, X3DOM konverteris užtikrindavo maksimalią greitaveiką iki 10 objektų. Tuo tarp sukurtas algoritmas užtikrindavo maksimalią galimą greitaveiką iki 60 objektų. Konvertuojant kitus VRML failus buvo stebima tokia pati tendencija.
9. Testuojant Three.js konverterį skirtingose naršyklėse, geriausia greitaveika pasižymėjo Chrome ir Edge programos. Taip pat visose lyginamose naršyklėse sukurto konverterio greitaveika buvo didesnė nei X3DOM konverterio.
10. X3DOM konverteris gali atvaizduoti daugiau VRML failų, kuriuose naudojami ne tokie populiarūs mazgai. Tačiau neatvaizduoja didelėmis ir sudėtingomis indeksuotų sienų aibėmis pasižyminčių VRML failų. Taip pat X3DOM konverteris tiksliau atkuria VRML atspindžius, tuo tarpu sukurtas Three.js konverteris tiksliau atkuria medžiagos šešėlius.
11. VRML formato konverteris gali būti patogus įrankis, siekiant labai paprasta sintakse aprašyti 3D objektus ir atvaizduoti juos interneto naršyklėje naudojantis WebGL technologija be jokių papildomų įskiepių.

Gairės

1. Automatizuoti VRML failo konvertavimą į tarpinį X3D formatą. Taip konverteris galėtų vienu metu konvertuoti dviejų formatų failus.
2. Sukurtas konverteris gali atvaizduoti failus, kurių negali X3DOM, tačiau yra tokių failų, kurių negali atvaizduoti Three.js konverteris, bet gali X3DOM algoritmas. Todėl tolesniuose darbuose reiktų išplėsti sukurto algoritmo nuskaitomus ir konvertuojamus mazgus, kad jis galėtų konvertuoti ne tokius svarbius ir populiarius VRML laukus.
3. Optimizuoti šiuo metu sukurtą algoritmą, pašalinant bet kokias nereikalingas operacijas.
4. Konvertuojant animacijas, objektų geometrijos sukūrimui panaudoti Three.js *BuferGeometry* klasę. Ši klasė pasižymi didesne greitaveika, nes iš anksto tinkamai paruošti duomenys perduodami grafinio vaizdo procesoriui.
5. Kuriant objektus su Three.js biblioteka, medžiagų kūrimui panaudoti ne tik *MeshPhongMaterial*, bet ir kitas klases.
6. Esant dideliam konvertuojamam VRML failui, apjungti greta esančias indeksuotų sienų aibes, taip sumažinant objektų skaičių ir padidinant WebGL greitaveiką bei sumažinant išeksportuojamos scenos dydį.
7. Didelę įtaką naršyklės atkuriamų kadrų per sekundę dažniui turi reikiamų sugeneruoti WebGL objektų skaičius. Todėl siekiant išgauti kuo didesnį naršyklėje atkuriamą KPS, reikia stengtis sukurti kuo mažiau objektų. VRML failo konvertavime tai galima pasiekti patobulinant algoritmą, kad jis apjungtų greta esančias geometrijas į vieną objektą. Tačiau tokiu būdu gali būti prarandama galimybė konvertuoti objekto animacijas.
8. Konvertuojant VRML failą į WebGL, ir siekiant, kad sukurti objektai užimtų kuo mažiau vietos, juos reikia kurti naudojant kuo mažiau trikampių. Konvertavimo algoritme trikampių skaičių galima sumažinti išbandant sudėtingesnius nei *Vėduoklės trianguliacija* metodus arba patobulinant algoritmą, kad jis galėtų apjungti gretimose sienose esančius trikampius ir taip pašalintų perteklinę trianguliaciją.

Literatūros šaltiniai

[1] Baltymų duomenų bazė (pdb - protein data bank).
https://www.rcsb.org.

[2] Explaining basic 3d theory.
https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Basic_theory.

[3] Indeksuotų sienų aibės aprašymas.
http://www.lighthouse3d.com/vrml/tutorial/index.shtml?indfs.

[4] jmol programa skirta peržiūrėti įvairias chemines struktūras.
http://jmol.sourceforge.net.

[5] Mit licenzija.
https://opensource.org/licenses/MIT.

[6] Numatytosios vrml formato vertės.
http://www.agocg.ac.uk/train/vrml2rep/appendix/appxanod.htm.

[7] Opengl įvadas.
http://math.hws.edu/eck/cs424/notes2013/06_Into_3D_OpenGL.html.

[8] The ssim index for image quality assessment.
http://www.cns.nyu.edu/~lcv/ssim/.

[9] Three.js animacijų modelis.
https://threejs.org/docs/#manual/en/introduction/Animation-system.

[10] Three.js naudojamos medžiagos.
https://threejs.org/docs/#api/en/materials/Material.

[11] Vrml konvertavimas į x3d formatą.
http://doc.instantreality.org/tools/x3d_encoding_converter.

[12] Webgl technologiją palaikančios naršyklės.
https://caniuse.com/#feat=webgl.

[13] Html5 3d visualisations, 2011.
https://courses.isds.tugraz.at/ivis/surveys/ss2011/g2-survey-web-ivis-3d.pdf.

[14] 30 best 3d design/3d modeling software tools, 2018.
https://all3dp.com/1/best-free-3d-modeling-software-3d-cad-3d-design-software/.

[15] Johannes Beh, Peter Eschler, Yvonne Jung, and Michael Zollner. X3dom – a dom-based html5/ x3d integration model.
http://www.web3d.org/wiki/images/3/30/X3dom-web3d2009-paper.pdf.

[16] Gavin Bell, Rikk Carey, and Chris Marrin. The virtual reality modeling language specification. version 2.0, 1996.
http://gun.teipir.gr/VRML-amgem/spec/vrmlspec.pdf.

[17] Jos Dirksen. Three.js essentials. Packt Publishing Ltd, 2014.
http://www.dominictran.com/pdf/ThreeJS.Essentials.PACKT.pdf.

[18] Alun Evans, Marco Romeo, and Arash Bahrehmand. 3d graphics on the web: A survey. Interactive Technologies Group, Universitat Pompeu Fabra, Barcelona, Spain, 2014.

[19] Jonathan Lampel. The beginners guide to blender, 2015.
https://www.blenderhd.com/wp-content/uploads/2015/08/BeginnersGuideToBlender.pdf.

[20] Tony Parisi. Webgl: Up and running. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2012. .

[21] Tony Parisi. Programming 3d applications with html5 and webgl.
O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2014.

[22] Roberto Ranon. Introduction to x3d. University of Udine, Italy.
http://hcilab.uniud.it/let-web3d/x3d-intro.pdf.

[23] Daniel K. Schneider and Sylvere Martin-Michiellot. Vrml primer and tutorial, 1998.
https://tecfa.unige.ch/guides/vrml/vrmlman/vrmlman.pdf.

[24] Tzvetalin Simeonov Vassilev. Optimal area triangulation. University of Saskatchewan, 2005.
http://www.collectionscanada.gc.ca/obj/s4/f2/dsk3/SSU/TC-SSU-08232005111957.pdf.

[25] Klaral Waerner. 3d graphics technologies for web applications. Linkoping University, 2012.
http://liu.diva-portal.org/smash/get/diva2:536657/FULLTEXT03.pdf.

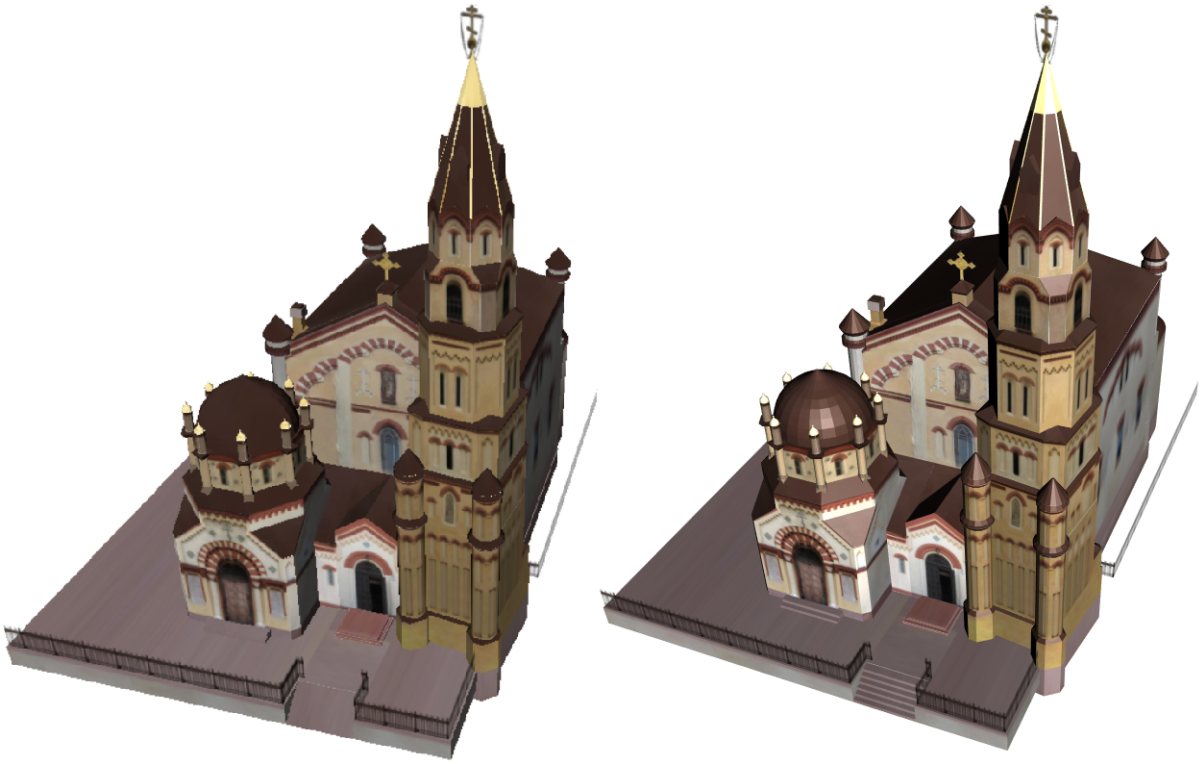
[26] Fatma Ziwar and Rimon Elias. Vrml to webgl web-based converter application.
The German University in Cairo, Egypt, 2014.

Priedai

A. VRML indeksuotų sienų aibės pavyzdys

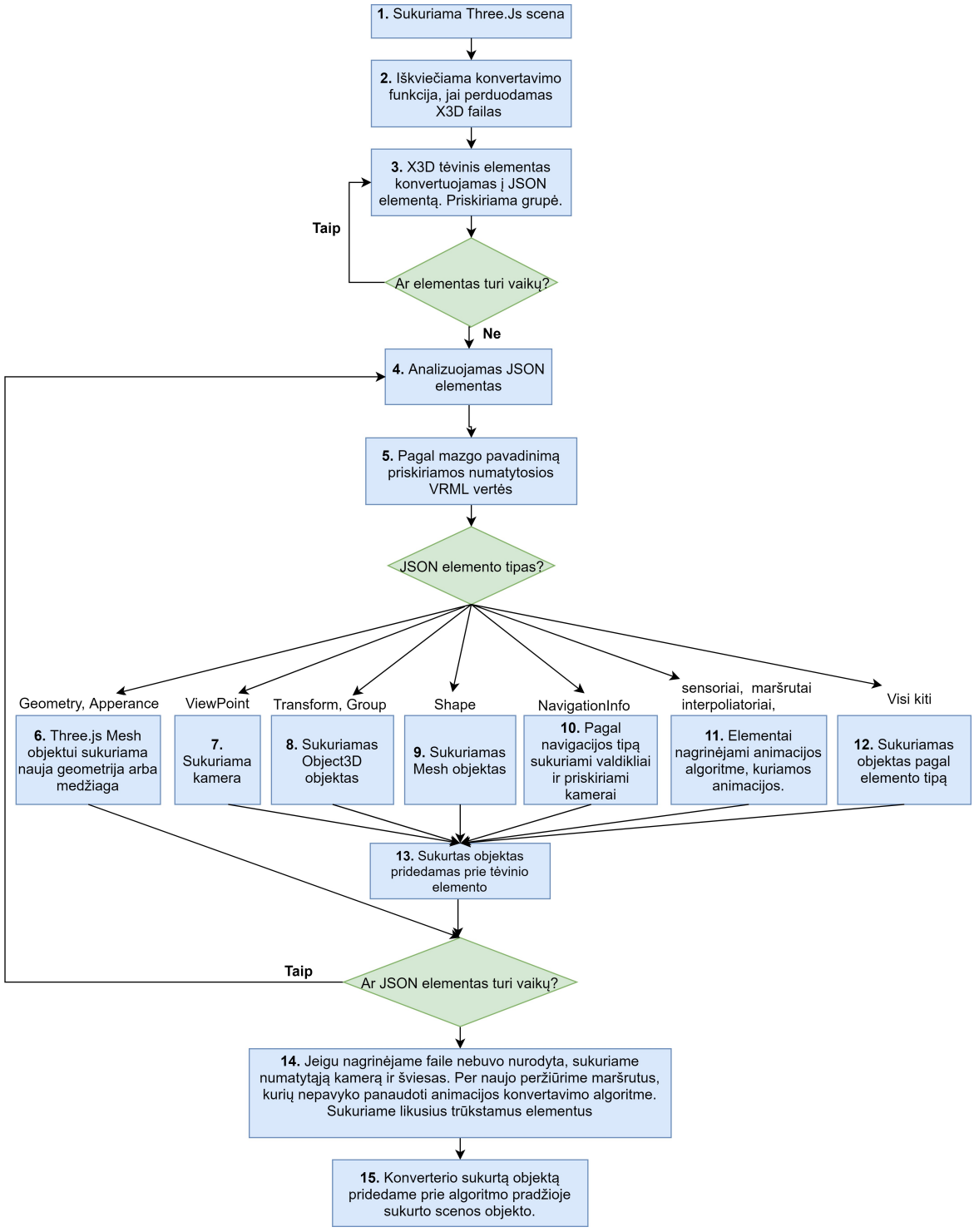
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance' profile='Full' version='3.0'
xsd:noNamespaceSchemaLocation='http://www.web3d.org/specifications/x3d-3.0.xsd'>
  <Scene DEF='scene'>
      <Shape>
        <NavigationInfo type='"EXAMINE"'/>
        <Viewpoint description='Viewpoint1' orientation='-0.69 -0.72 0 0.8' position='-3.3 3.5 5.8' fieldOfView='0.84'/>
        <Background groundAngle='1.57' groundColor='1 1 1 1 1 1' skyAngle='1.2 1.57' skyColor='0 0 1 0 0 0.6 1 0 0'/>
        <Appearance>
          <Material shininess='0.4'/>
          <ImageTexture url='""'/>
        </Appearance>
        <IndexedFaceSet solid='false' coordIndex='0 1 2 3 4 5 -1 6 7 8 9 10 11 12 13 14 15 16 17'>
          <Coordinate point='0 5 2 0 5 6 0 2 8 0 -1 6 0 -1 0 0 2 0 0 0 0 0 0 2 0 1 4 0 3
                             5 0 5 5 0 7 4 0 8 2 0 8 0 0 7 -2 0 5 -3 0 3 -3 0 1 -2 0'/>
        </IndexedFaceSet>
        </Shape>
  </Scene>
</X3D>
```

B. Vilniaus Šv. Nikolajaus cerkvė konvertuota į WebGL

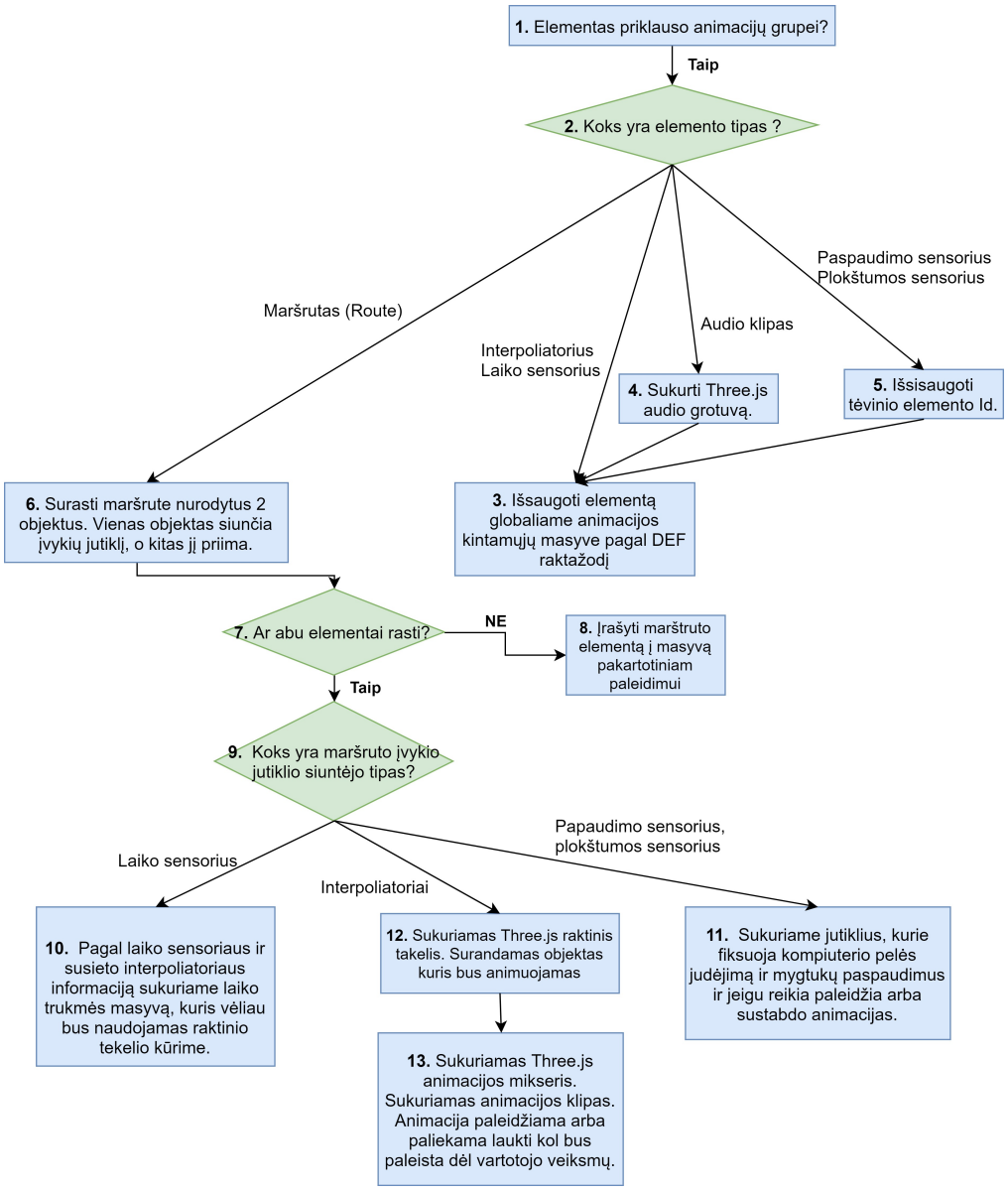


28 pav. Kairėje - VRML failas, peržiūrimas BS Contact Stereo programa. Dešinėje - WebGL objektas, peržiūrimas interneto (Chrome) naršykle.

C. VRML konvertavimo į WebGL veiksmų diagrama



D. VRML animacijų konvertavimo į WebGL veiksmų schema



E. Darbe naudojami VRML failai

VU-kiemai.wrl - Vilniaus universiteto kiemai. R. Krasauskas, T. Monkevičius, "VU kiemai - interaktyvi 3D vizualizacija", dalinai paremta "Vilnius - Europos kultūros sostinė 2009" projekto.

N-cerkve.wrl - Vilniaus Šv. Nikolajaus cerkvė. A. Bistrickis, 3D architektūrinė vizualizacija: Šv. Nikolajaus cerkvė, Kursinis darbas, MIF VU, 2007.

Dance.wrl - šokio animacija. M. Isenburg, J. Snoeyink. Coding with ASCII: compact, yet text-based 3D content, in: Proceedings of the First International Symposium on 3D Data Processing Visualization and Transmission, 2002

Piano.wrl - koncertinė pianino klasė. Kompiuterinės grafikos projektas "PIANINAS". Artūras Baniukevič. Vilniaus universitetas. 2007.

Berniukas.wrl - Shay D Pixel Version 2. Mario Nagumura. http://x3dgraphics.com/examples/

X3dForAdvancedModeling/ShayDPixel/ShayDPixelVersion2.wrl

Astuonkojis.wrl - Rocktopus. http://x3dGraphics.com/examples/X3dForAdvancedModeling/AdditiveManufacturing/Rocktopus.wrl

Puodelis.wrl - Cad Teapot. Alan Hudson, Don Brutzman. http://www.web3d.org/x3d/content/examples/Basic/CAD/CadTeapot.wrl

Skeletas.wrl - Bones All skeleton. Damon Hernandez, Joe Williams. http://www.web3d.org/x3d/content/examples/Basic/Medical/BonesAllSkeleton.wrl

SkeletasNormales.wrl - Skeleton Complete Normals. Damon Hernandez, Joe D. Williams, Don Brutzman. http://www.web3d.org/x3d/content/examples/Basic/Medical/SkeletonCompleteNormals.wrl

Boeing747.wrl - Boeing747. Nabil Ouerghi. https://savage.nps.edu/Savage/AircraftFixedWing/Boeing747Tunisia/Boeing747.wrl

Dangtis.wrl - Uav Beehive. http://x3dgraphics.com/examples/X3dForAdvancedModeling/AdditiveManufacturing/UavBeehive.wrl

Molekulės PDA numeris – 600C, 3BSE, 3H06, 6FS1, 6A83 gaunamos iš Jmol programos įrašius PDA numerį.