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

Magistrinis darbas

Tekstų įterpimas į 3D objektų vizualizacijas prezentacijų programinėje įrangoje

Atliko:
Viktorija Plukė parašas

Vadovas:
dr. Rimvydas Krasauskas

Vilnius
2021

Turinys

Sutartinis terminų žodynas **3**

Santrauka **5**

Summary **6**

Įvadas **7**

1. TEX kokybės šriftas **9**
1.1. Kas yra TEX 9
1.2. Kaip veikia TEX 11
1.3. TEX šrifto atvaizdavimas 14
1.3.1. Vektorinės grafikos atvaizdavimo algoritmas 15

2. Trimačių objektų atvaizdavimas **18**
2.1. Kas yra WebGL 18
2.2. Kaip veikia WebGL 19
2.3. Šrifto atvaizdavimas WebGL 20
2.3.1. Tekstas kaip HTML elementas 21
2.3.2. Tekstas kaip 3D scenos objektas 23

3. TEX kokybės šrifto atvaizdavimas 3D scenoje **26**
3.1. Kas yra KaTex 26
3.2. Kaip veikia KaTex 27
3.2.1. Dvimatė TEX kokybės žymė (kaip HTML elementas) 3D scenoje 28
3.3. Kas yra Asymptote 30
3.4. Kaip veikia Asymptote 30
3.4.1. Trimatė TEX kokybės žymė (kaip scenos objektas) 3D scenoje 33

4. Alternatyvi koncepcija TEX ir WebGL suderinamumui **39**
4.1. Esamų technologijų pernaudojimas 40
4.2. Trūkstama komponentė 42
4.3. Tikėtinas rezultatas 44

Išvados ir rekomendacijos **45**

Ateities tyrimų planas **47**

Literatūros šaltiniai **48**

Priedai **51**

A. Trimačiai objektai su žymėmis atvaizduojami su LATEX kuriamame turinyje (.pdf formatu) **52**

B. 3D objekto perpiešimui su *Asymptote* reikalingų duomenų pavyzdys **54**

Sutartinis terminų žodynas

- TEX - programavimo kalba, skirta kompiuterinei tipografikai. T. p. tipografinis standartas, t. y. - labai kokybiška (*Tex* standarto) tipografika. T. p. bendrinis apibūdinimas programinei įrangai, kurią sukūrė ir iki šiol palaiko D.Knuth'as. T. p. vidinio mechanizmo pavadinimas (angl. *Tex engine*) kitų autorių sukurtose programinėse įrangose, parašytose pernaudojant D.Knuth'o programinį kodą.
- LATEX - Leslie Lamport aprašytas *Tex* mechanizmo komandas paleidžiantis aukštesnio lygio (vartotojui labiau suprantamas) makrokomandų rinkinys, kuris atsakingas už autorinio turinio rašymo organizavimą. T. y. labiausiai reikalingos makrokomandos rašant knygas, straipsnius ir kitas, dažniausiai mokslinio turinio, publikacijas ar dokumentus.
- pdfTex, XeTex, LuaTex - bendrai vadinami *Tex* mechanizmais, t. y. kitų autorių sukurta programinė įranga, parašyta pernaudojant D.Knuth'o programinį kodą.
- MetaFont - D.Knuth'o sukurta *Tex* kalbai tinkanti šrifto technologija, aprašanti rastrinius šriftus. T. y. schema, pagal kurią generuojami spaudos ženklų rastrai (angl. *bitmap*).
- DVI - 1982 metais D.Knuth'o sukurtas *Tex* išvesties (angl. *output*) failo formatas (angl. *DeVice Independent format*) naudojamas kaip įvestis (angl. *input*) vidinei tvarkyklei (angl. *DVI driver*). Pastaroji DVI failus paverčia grafiniais duomenimis.
- PDF (.pdf) - angl. *Portable Document Format*, tekstinio failo (dokumento) formatas.
- SVG (.svg) - angl. *Scalable Vector Graphics*, vektorinės grafikos failo formatas.
- OpenType (.otf) - keičiamo dydžio (angl. *scalable*) kompiuterinio šrifto formatas, išvystytas *TrueType* formato pagrindu.
- TrueType (.ttf) - spaudos ženklų kontūro standartas atsiradęs 1980-tais, skirtas *PostScript* technologijai. T. p. šrifto formatas įprastai naudotas operacinėse sistemose, kurį programuotojas galėjo kontroliuoti pikselių lygmenyje.
- PostScript (.ps) - puslapio aprašomoji programavimo kalba (angl. *Page Descriptive Language*). T. p.tekstinio failo formatas.
- MetaPost - grafinė programavimo kalba, kurią supranta *Tex* mechanizmas, skirta tekstiniame turinyje braižyti vektorinės grafikos diagramas/grafikus, naudojantis algebriniais ar geometriniais aprašymais.
- GPU - kompiuterio grafinis procesorius (angl. *graphics processing unit*).
- OpenGL - angl. *Open Graphics Library*, aplikacijų programavimo sąsaja (angl. *Application Programming Interface*), skirta 2D ir 3D vektorinei grafikai atvaizduoti (angl. *render*), naudojanti programuojamų šešėliuoklių principą.
- WebGL - aplikacijų programavimo sąsaja skirta interaktyvios 2D ir 3D grafikos atvaizdavimui interneto naršyklėje be papildomų įskiepių. WebGL šešėliuoklės suprogramuotos GLSL ES kalba (angl. *Graphic Library Shading Language for Embedded Systems*). T. p. formatas - 3D scena interneto naršyklėje.

- HTML (.html) - angl. *Hypertext Markup Language*, hiperteksto žymėjimo kalba skirta turiniui atvaizduoti interneto naršyklėje. T. p. failo formatas.
- SDF - orientuoto atstumo laukai (angl. *Signed Distance Fields*), duomenys aprašantys rastrinį spaudos ženklą.
- MSDF - angl. *multi-channel SDF*, spaudos ženklo aprašymui naudojami orientuoto atstumo laukų duomenys (SDF) ir papildoma trijų spalvų kanalų reikšmė.
- KaTex - JavaScript programavimo kalba parašyta biblioteka, galinti realiu laiku generuoti *Tex* matematines išraiškas .html formatu.
- Asymptote - vektorinė programavimo kalba skirta generuoti trimatę vektorinę grafiką PostScript, PDF, SVG ir WebGL formatais.
- Billboard - Asymptote žymės tipas „skydas“, kuomet tekstas visuomet atvaizduojamas iš kairės į dešinę ir neapverčiamas aukštyn kojomis nepriklausomai nuo to, kaip bus pasukamas trimatis objektas.
- Embedded - įterptinis Asymptote žymės tipas, kuomet tekstas gali būti atvaizduotas apverstas (aplink savo vertikalią ar horizontalią ašį), jeigu pasukamas trimatis objektas.
- Furjė transformacijos formulė $f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\, e^{2\pi i \xi x} d\xi$ - praktiniams pavyzdžiams pasirinkta dėl nestandartinio retesnių spaudos ženklų išdėstymo. Čia simboliai, skirtingai nei įprastame tekste vienas paskui kitą, rikiuojami viršuje ir apačioje (pavyzdžiui integralo ribos $\int_{-\infty}^{\infty}$), virš kito simbolio (pavyzdžiui stogelis $\hat{f}$ arba kėlimas laipsniu $e^{2\pi i \xi x}$) ir t. t. Tokio teksto nėra įmanoma surinkti klaviatūra, tad buvo užtikrinama, kad žymei formuoti naudojamas $\TeX$ šriftas.
- ThreeJS - JavaScript programavimo kalba parašyta biblioteka ir aplikacijų programavimo sąsaja, skirta atvaizduoti animuotą 3D kompiuterinę grafiką interneto naršyklėje, naudojant WebGL šešėliuokles.

Santrauka

Šiame darbe analizuojami būdai įterpti kokybiškas tekstines žymes (angl. *labels*) į trimačių objektų vizualizacijas prezentacijų programinėje įrangoje. Kaip kokybės standartas pasirinktas TEX šriftas. Vienas iš jo privalumų yra galimybė atvaizduoti ne tik paprastą tekstą, bet ir sudėtingas matematines formules. Buvo ieškoma sprendimo kaip atvaizduoti TEX kokybės tekstą naudojant grafinio procesoriaus programavimą, t. y. šešėliuoklių kalbą, nes būtent šis principas taikomas 3D kompiuterinei grafikai atvaizduoti. Iškilo problema - norint į trimatę sceną įterpti sudėtingą matematinę formulę, tekstą reikėjo versti rastru ir prarasti TEX standartui būdingą aukštą kokybę, arba naudotis internetinėmis technologijomis, t. y. *HTML* elementais, ir nepaisyti grafinio procesoriaus programavimo principo. Šių būdų buvo atsisakyta ir pasiūlytos dvi, kitokio pobūdžio, koncepcijos, kaip vektorinės grafikos ir 3D kompiuterinės grafikos atvaizdavimo priemonių nesuderinamumo problemą galima būtų išspręsti. Taip pat praktiškai išbandyta ir aprašyta šiuo metu prieinama technologija, leidžianti įterpti TEX tekstines žymes į 3D scenas ir atvaizduoti jas prezentacijų programinėje įrangoje.

Summary

Labeling Visualisations of 3D Objects within Presentation Software

When working with 3D graphics and creating 3D objects, it is known that labeling the object is not an easy task to complete. There is no unified way to insert labels into a 3D scene and this can therefore lead to further problems.

To present a 3D object via slides, article, publication or in any other static presentation form, there is the need to create a presentative visualisation of the object. In simple terms a 3D object is transformed into a flat picture. Any labels that are required to be displayed within that image, must be added before the process of that transformation.

Adding simple text while creating 3D graphics is resolvable in a defined number of ways. However, in cases where the label text must be very specific, such as mathematical, geometric or other scientific expressions or formulas, the problem is further magnified. In mathematics and related discipline publications, such expressions are handled with LaTeX typesetting programme. When wanting to use the created text somewhere outside LaTeX, another TeX supporting software is essential. It was determined, that labeling the 3D object with all mathematics needed in the stage of creating the 3D scene is possible using *KaTeX* (the library rendering text based on TeX in browsers) or *Asymptote* (the descriptive vector graphics language that provides a framework for technical drawing with labels and equations which are typeset with LaTeX).

After studying these solutions, the practical experiment was accomplished, and a labeled 3D object with the following visualisation was created in different ways that emphasized the advantages and disadvantages of every tool. These pros and cons had not been determined in set terms previously. At this stage the experiment revealed the main limitation of the current situation. Apparently, when using *KaTeX* the text of the label is not rendered using shading language in the way the rest of the scene is. Meanwhile, when using *Asymptote*, the 3D scene must be also created with *Asymptote*, so in the case where the 3D graphics were created with more sophisticated tool, like ThreeJS library, there is no way to add vector graphics quality *Asymptote* labels into the scene.

Therefore, another problem was identified and two possible solutions to it were proposed. They are only theoretical concepts yet and must be properly analysed and practically conducted in the future. These improved and completely different approaches will allow users to insert high quality labels, based on TeX text, into elaborate 3D graphics of the creator's choice. The visualisation of the results within the presentation software will therefore be less complicated and more qualitative compared to now.

Įvadas

Norint į su *LaTex* kuriamą prezentacinį turinį įtraukti autoriaus sukurtą trimatį objektą, kitaip sakant, vizualizuoti 3D sceną dvimatėje plokštumoje - publikacijos puslapyje, prezentacijos skaidrėje ir pan., - tikimasi, kad 3D scenoje esančiam tekstui bus taikomas toks pats kokybės standartas, kaip ir kitam prezentacijos tekstui. T. y. jeigu trimatis objektas, kurį norima vizualizuoti, turi žymes (angl. *labels*), jų tekstas turėtų būti tokios kokybės, kaip bet kuris kitas su *LaTex* kuriamas tekstas. Tačiau svarbu ne vien tik žymės teksto kokybė, bet ir įterpimo momentas. Vizualizuojant sukurtą 3D objektą be reikiamų žymių, t. y. pateikiant jį kaip iliustratyvų pavyzdį reprezentaciniame turinyje (pavyzdžiui su *LaTex* rašomame moksliniame straipsnyje), žymių įterpimo klausimą tenka spręsti ne vieną kartą, juk dažnu atveju objektą reikia atvaizduoti keliuose puslapiuose/-skaidrėse, keliais skirtingais rakursais arba keičiant objekto mastelį. Taip pat, galimai, redaguojant galutinį rezultatą, pavyzdžiui pastebėjus klaidą. Tuomet kiekvienu atveju reikia ne tik įterpinėti paties objekto vizualizaciją (paveikslėlį), bet ir papildomai tame paveikslėlyje „perdėlioti“ visas objekto žymes. Taigi, neįterpus reikiamų žymių laiku, šis trūkumas jaučiamas vizualizavimo etape ir apsunkina turinio pateikimo auditorijai procesą.

Siekiant išvengti papildomo vizualizacijų redagavimo, trimačio objekto žymės, jeigu jų reikia, turi būti sukuriamos 3D scenos kūrimo/saugojimo/atvaizdavimo (angl. *render*) etape, tuomet įterpiamos į prezentacinį turinį, kartu su visu objektu, kaip jau egzistuojanti trimatės scenos dalis. Tačiau, įterpti tekstą kuriant 3D scenas nėra paprasta, nes kompiuterinės grafikos kūrimo įrankių tikslas bei veikimo principas skiriasi nuo tekstinių redaktorių bei teksto atvaizdavimo įrankių veikimo principų ir tikslų. Šis uždavinys tampa dar sudėtingesniu, jeigu norima išsaugoti tiek teksto, tiek 3D scenos kokybę, prie kurios yra įpratę šių atskirų įrankių naudotojai.

Teksto aukščiausia kokybe yra laikomas 1978 metais Donald'o Knuth'o sukurtas *Tex* standartas - tai vektorinės grafikos bet kokios kalbos abecelės ar tiksliųjų mokslų srities (matematikos, fizikos ir kt.) išraiškų spaudos ženklai, prieinami kiekvienam vartotojui nepriklausomai nuo jo turimų technologinių priemonių (kompiuterio, operacinės sistemos, spausdintuvo ar ekrano rezoliucijos). Kompiuterinės grafikos standartu yra laikomas OpenGL - šios aplikacijų programavimo sąsajos specialia šešėliuoklių kalba galima suprogramuoti įspūdingus vaizdo efektus, bet kokio įmantrumo 3D scenas ir kokį tik norisi trimatį objektą. Šio **darbo pagrindinis tikslas** yra apjungti šiuos du standartus ir **įterpti kokybišką tekstinę žymę šešėliuoklių kalba atvaizduojamoje 3D scenoje**. Tokiu būdu gautą rezultatą (trimatį objektą su įterpta *Tex* teksto žyme) **vizualizuoti** su *LaTex* kuriamame **prezentaciniame turinyje ir įvertinti** įterpimo procesą bei žymės kokybę.

Tačiau šešėliuoklių kalba nėra skirta tekstui atvaizduoti, ji skirta perduoti duomenis grafiniam procesoriui, kuris savo ruožtu išrastruos gautus duomenis atitinkančius trikampius. Tuo tarpu *Tex* standarto šriftas kokybišku laikomas būtent todėl, kad nėra rastruojamas iš trikampių. *Tex* spaudos ženklai yra vektorinės grafikos atvaizdavimo rezultatas. Taigi, susiduriama su **problema - nėra aišku, kokius duomenis perduoti šešėliuoklėms, kad grafinis procesorius išrastruotų vektorinės grafikos kokybei prilygstantį spaudos ženklo kontūrą**. Taip pat svarbu, kad tai galėtų būti bet kokie spaudos ženklai, netgi tokia nestandartinė ženklų seka, kaip sudėtinga matematinė formulė. Tokiose išraiškose spaudos ženklai nebūtinai išsidėsto vienas paskui kitą iš eilės, bet gali būti ir kito ženklo viršuje (pavyzdžiui kėlimas laipsniu) arba apačioje (pavyzdžiui žymint indeksą) arba kelios sekos išsidėsčiusios viena virš kitos (pavyzdžiui trupmenos skaitiklyje ir vardiklyje). *LaTex* tokias išraiškas atvaizduoti moka, tačiau kokiomis priemonėmis jas įkelti į 3D sceną ir atvaizduoti šešėliuoklėmis?

Ieškant atsakymo į šį klausimą, 2020-jų metų mokslinio tiriamojo darbo projekte buvo nu-

spręsta praktinius bandymus pradėti nuo internetinės (angl. *web*) trimačių objektų ir *Tex* standarto bendrystės, kadangi paaiškėjo, kad tiek vienas, tiek kitas, gali būti atvaizduojami interneto naršyklėje. Poskyryje 2.3.1 aprašyta pasiekto rezultato teorinė dalis, o poskyryje 3.2.1 parodyti praktiniai rezultatai. Nors atrodytų, kad tikslas pasiektas - šalia trimačio objekto naršyklėje atsirado tekstinė žymė su sudėtinga matematine formule - rezultatas nebuvo pakankamas, kadangi žymės tekstas atvaizduojamas ne per šešėliuoklių kalbą, o naudojant internetines technologijas, t. y. HTML kalbą (angl. *Hypertext Markup Language*).

Tęsiant esamos situacijos analizę ir sprendimo paiešką, šiame darbe buvo suformuluoti tokie metodologiniai uždaviniai:

1. Išnagrinėti kaip veikia aukščiausiu šrifto standartu laikoma *Tex* technologija ir kokiu principu jos spaudos ženklą galima būtų išrastruoti grafiniu procesoriumi.

   Pirmas skyrius skirtas *Tex* mechanizmo evoliucionavimo, makrokomandų veikimo ir rezultato atvaizdavimo aprašymui. Taip pat apžvelgtas vektorinės grafikos rastravimo grafiniu procesoriumi algoritmas, suponuojantis, kad norimo rastruoti teksto duomenys turėtų būti pateikti *TrueType* formatu (kvadratinių B-splainų ir tiesių segmentų derinys).

2. Išnagrinėti kaip veikia šešėliavimas ir kokiu būdu įmanoma šešėliuoklėmis atvaizduoti tekstą trimatėje scenoje.

   Antrame skyriuje aprašomas WebGL veikimo principas ir visi įmanomi būdai (įskaitant ir per internetines technologijas) atvaizduoti tekstą 3D erdvėje.

3. Išanalizuoti esamą situaciją, gal būt jau yra sukurtos priemonės atvaizduoti *Tex* kokybės šriftą 3D scenose ir vizualizuoti juos prezentaciniame turinyje.

   Trečiame skyriuje aprašytos ir praktiškai išbandytos dvi rastos priemonės. *KaTex* biblioteka, kurios pagalba norimą matematinę formulę galima „prikabinti“ prie trimačio objekto per internetines technologijas (neperduodant šešėliuoklėms žymės duomenų); ir vektorinės grafikos programavimo kalba *Asymptote*, kuri skirta *Tex* kokybės žymėms atvaizduoti 3D scenoje (perduoda šešėliuoklėms Bezjė paviršių duomenis).

4. Įvertinti ar dabar egzistuojančiomis priemonėmis pasiekiamas šiame darbe iškeltas tikslas.

   Sprendimas įterpti žymes per internetines technologijas paminėtas kaip galimas, tačiau nepakankamas rezultatas, tad šiame darbe labiau buvo vertinamos *Asymptote* teikiamos galimybės. Išanalizuotas unikalus *Asymptote* algoritmas ir identifikuotas atvejis, kuomet šios priemonės gali neužtekti. Ketvirtame skyriuje pasiūlytos dvi alternatyvios koncepcijos, kaip galima būtų pernaudojant *Asymptote* veikimo principą, arba pasinaudojant 1 ir 2 skyriuose aprašytomis teoretinėmis galimybėmis, atvaizduoti *Tex* teksto žymes su ThreeJS sukurtoje trimatėje scenoje.

1. $\TeX$ kokybės šriftas

Nemažai akademinės bendruomenės atstovų savo autoriniam turiniui kurti naudoja *LaTex*. Tačiau nebūtinai visi gali apibrėžti kas tiksliai tai yra. Neretai *LaTex* suvokiamas kaip rinkoje esančių tekstinių redaktorių alternatyva - nemokama, teikianti kokybiškesnį rezultatą, nors ir sudėtingesnė pagal savo vartojimo principą. Tai nėra visiška tiesa. *LaTex* nėra tekstinis redaktorius, tai *tipografikai* skirtas labiausiai reikalingų priemonių rinkinys [1.1]. „Tipografika“ yra ketinamo spausdinti turinio apipavidalinimas, kuomet surinktas tekstas išdėstomas pagal tam tikrą principą, parenkamas šriftas, jo spalvinis tonas, atskirų dalių proporcijos ir pozicija puslapyje. Tipografika gali būti suvokiama kaip menas [7], bet gali būti laikoma ir mokslu, kadangi reikalauja gebėjimo suprasti ir vykdyti nustatytas taisykles [4]. Kaip teigia knygos [20] autorius, rinkti tekstą spausdinimui atrodo lengva ir paprasta kol neprireikia surinkti ko nors panašaus į:

$$\lim_{\varepsilon \rightarrow 0_{+}} \frac{\int_{a_i}^{a_i+\varepsilon} \sqrt{1+(x-\mu)^2} dx}{\Phi(\varepsilon)} \tag{1.1}$$

Tokiu atveju meniniai gebėjimai turi likti nuošaly, o pirmenybė suteikiama griežtam taisyklių laikymuisi. Poreikis turėti visiems prieinamą tipografinį įranki matematiniams (ir kitų, susijusių, disciplinų) moksliniams tekstams rinkti, kadaise ir paskatino Donald'ą Knuth'ą sukurti *Tex* [tariama T-e-k].

Tex iš esmės yra programavimo kalba, skirta kompiuterinei tipografikai. Ją sukurti užtruko mažiau nei metus, o išleisti knygą apie naudojimąsi - dešimtmetį [22]. Su šia knyga prasidėjo naujas etapas matematinių/mokslinių tekstų publikavime.

LaTex [tariama Lah-tek arba Lay-tek] yra makrokomandų (angl. *macros*) rinkinys [20], skirtas darbui su *Tex* kalba palengvinti. Šios makrokomandos gali būti renkamos ir vykdomos bet kokiame tekstiniame redaktoriuje, palaikančiame paprasto teksto (angl. *plain text*) formatą.

Autoriniame turinyje norint surinkti aukščiau nurodytą matematinę formulę, tekstiniame redaktoriuje turime parašyti tokią *LaTex* makrokomandą:

```
\begin{equation}
  \lim _{\varepsilon \rightarrow 0_{+}} \frac{\int_{a_{i}}^{a_{i}+
  \varepsilon} \sqrt{1+(x-\mu)^{2}} d x}{\Phi(\varepsilon)}
\end{equation}
```

Pirmoje ir paskutinėje eilutėje yra *LaTex* makrokomandos, o viskas tarp jų, *Tex* kalba parašytas nurodymas kokius simbolius kokiose pozicijose ir kaip atvaizduoti [1.2]. Praėjus 40-čiai metų šios komandos ir nurodymai veikia puikiai ir jų rezultatai atvaizduojami visuose labiausiai naudojamuose 2D turinio formatuose (dokumentuose, internetiniuose puslapiuose, vektorinės grafikos failuose, rastriniuose paveikslėliuose ir pan.), tačiau perkelti šių komandų funkcionalumą (arba jų vykdymo rezultatus) į 3D erdvę, nėra paprasta užduotis.

1.1. Kas yra $\TeX$

Tex kalbą D.Knuth'as sukūrė prieš gerus 40 metų, kuomet buvo išleista jo knyga „Programavimo menas“ (angl. *“The Art of Computer Programming”*) ir jos spausdinimo kokybė autoriui pasirodė per prasta [17]. Nusprendęs aprašyti savo tipografinį standartą, D.Knuth'as net nenutuokė, kad pradėjo naują mokslinių publikacijų erą, kuri tęsiasi iki šiol. Jo sukurti algoritmai išsprendė

daugybę sudėtingų tipografikos uždavinių, tokių kaip automatinis teksto perkėlimas į kitą eilutę, brūkšnių ir brūkšnelių dėliojimas ir, žinoma, matematinių išraiškų (formulių) atvaizdavimas. Siekiant kokybiško šių tikslų įgyvendinimo, D.Knuth'ui teko sugalvoti ir suprogramuoti atskirą šriftą, tinkantį *Tex* kalbai. Taip atsirado atskira, iki tol nematyta, šrifto technologija *MetaFont* [1.3].

Šiomis priemonėmis pasiekus išskirtinę tipografikos kokybę *Tex* išpopuliarėjo ir nepraranda naudotojų iki šiol. Principas, kuriuo veikia *Tex* kalba yra *Tex macros* - unikalus makrokomandų rinkinys, kuris leidžia kiekvienam vartotojui pačiam kontroliuoti savo turinio tipografinį procesą [1.2]. Svarbu ir tai, kad *Tex* yra atviro kodo(angl. *open-source*) ir vienodai gerai veikia bet kokio kompiuterio bet kokioje operacinėje sistemoje. Nepriklausomai nuo įrenginio, *Tex* atvaizduoja identišką, vienodai kokybišką, rezultatą (kuomet įvestis (angl. *input*) yra tokia pati). Identišką, tai reiškia, kad teksto eilutės bus perkeltos tose pačiose vietose, puslapių skaičius bus tas pats, šrifto dydis spalva ir stilius bus tokie patys nepaisant kokia kompiuterine programa buvo apdorotas tekstas, kokia operacinė sistema yra naudojamame kompiuteryje ir kokiu spausdintuvu jis buvo atspausdintas (ar kokiame ekrane peržiūrėtas). Dabar kiekvienas matematikas, fizikas ar kitos srities mokslininkas gali pats preciziškai tiksliai kontroliuoti savo rašto darbų turinio išvaizdą bei nesijaudinti, kad spaustuvėms pateikus savo „rankraštį“ (.tex formato failą) spausdintas jo variantas nuvils taip, kaip *“The Art of Computer Programming”* leidimas nuvylė D.Knuth'ą 1977-aisiais.

Devintajame dešimtmetyje D.Knuth'as nusprendė baigti darbus ir daugiau nebetobulinti *Tex*. Jam buvo svarbu, kad ši programinė įranga būtų stabili ilgalaikėje perspektyvoje, tad diegti jokių naujų funkcijų neketino. 1989-aisiais vis tik jis atliko paskutinius, labai reikalingus pataisymus (pavyzdžiui vienam simboliui skirta saugojimo vieta buvo padidinta nuo 7 iki 8 bitų ir kt. [23]) ir jau 1990-aisiais savo straipsnyje [24] pareiškė, kad *Tex* ir su juo susijusios programinės įrangos vystymas baigtas, tačiau kiti, jeigu tik nori, gali naudotis atvirai prieinamu jos kodu ir tobulinti, plėsti ar kitaip gerinti.

Įdomu tai, kad D.Knuth'as iki šiol prižiūri originalų *Tex* kodą ir periodiškai pataiso kokią nors atrastą klaidą (angl. *bug*). Kita *Tex* peržiūra ir pataisymai numatyti 2021-aisiais metais, tačiau tai nereiškia, kad po kiekvieno pataisymo atsiranda nauja *Tex* „versija“ [17].*Tex* (be jokių versijų) vadinama ta vienintelė programinė įranga, kurią sukūrė ir iki šiol palaiko pats D.Knuth'as.

Prekinis ženklas ir logotipas TEX priklauso Amerikos matematikų draugijai (angl. *American Mathematical Society*)). Nors D.Knuth'as neapribojo ir nesuvaržė galimybių kitiems naudoti originalų *Tex* kodą siekiant jį tobulinti ar papildyti, tačiau jis neleido kitų patobulintą/papildytą kodą vadinti *Tex* arba *Tex* versija (su „versijavimo“ numeriu) [24]. Todėl *Tex* pagrindu (naudojant D.Knuth'o originalų kodą) parašyta programinė įranga yra „adaptacija“ arba „išvestinė“ ir jos pavadinimui tapo įprasta naudoti „Tex“ kaip priesagą, pavyzdžiui *pdfTex*, *XeTex* arba *LuaTex* ir pan. Šios programos yra parašytos nadojant originalų D.Knuth'o kodą, tačiau papildomai pridėjus funkcionalumo ir įvairių savybių, kurių neturi pirmapradis D.Knuth'o *Tex*. Šias programas bendrai įprasta laikyti/vadinti *Tex* mechanizmais (angl. *Tex engines*), nes iš esmės būtent jos organizuoja visą tipografikos procesą.

Tex mechanizmo adaptacijų poreikis atsirado labai greitai po D.Knuth'o paskelbtos vystymo darbų pabaigos. Jau 1990-aisiais kai kurios originalaus *Tex* dalys pradėjo „senti“ ir atsilikti nuo tuometinių pažangesnių technologijų. Vienas iš pavyzdžių yra failo, į kurį *Tex* generuodavo rezultatą, formatas. Tai buvo DVI, tačiau dauguma vartotoju jį konvertuodavo į *PostScript* formatą [1.3], o pastarsis dar kiek vėliau buvo visiškai užgožtas naujai atsiradusio PDF. Taip pat, po truputį, įsibėgėjo interneto era, tad vartotojai panoro, kad *Tex* standartas persikeltų ir į internetą (kokybiškas šriftas būtų atvaizduojamas internetiniuose puslapiuose). Įgyvendinant šiuos poreikius, pagrindiniai *Tex* tipografiniai algoritmai (teksto perkėlimas į eilutę, lygiavimas, matematiniai simboliai ir

t.t.) nepakito, tiesiog juos, taip pat kokybiškai, teko išmokti pritaikyti naujose aplinkose (kituose failų formatuose, internetiniuose puslapiuose ir pan.)

1.2. Kaip veikia TEX

Tex mechanizmais (programos sukurtos D.Knuth'o kodo pagrindu, tokios kaip *pdfTex*, *XeTex* ar *LuaTex*) laikomos programos atsakingos už tipografinio proceso organizavimą ir vykdymą. Atrodytų, kad LATEX yra vienas iš tokių mechanizmų. Tačiau taip nėra. *LaTex* yra *Tex* mechanizmo komandas paleidžiantis aukštesnio lygio (vartotojui labiau suprantamas) makrokomandų rinkinys. Šias makrokomandas aprašė Leslie Lamport devintame dešimtmetyje ir pavadino savo vardu. Patys *Tex* mechanizmai (įskaitant ir originalų D.Knuth'o) neturi grafinės vartotojo sąsajos, o veikia naudojant komandinę eilutę. Pagal dabartinių programavimo kalbų analogiją,*Tex* mechanizmą galima būtų apibūdinti kaip kompiliatorių, kuris bet kokią, parašyta jam suprantama kalba, įvestį (angl. *input*) perkompiliuoja į planine kalba (angl. *target language*) parašytą išvestį (angl. *output*). Kalbant dar paprasčiau, *Tex* mechanizmą galima laikyti tiesiog dokumentų kompiliatoriumi. Norint organizuoti ir vykdyti šį kompiliavimo procesą, reikia įvestyje turėti ne tik savo tekstą (publikacijos turinį), bet ir tikslias, *Tex* kalbos sintaksę atitinkančias, instrukcijas (komandas), kurios nurodytų, ką *Tex* mechanizmui daryti su įvestais simboliais.

Komandų, kurias supranta *Tex* yra šimtai. Jos sukurtos D.Knuth'o ir yra esminiai *Tex* programavimo kalbos konstravimo vienetai, skirti valdyti vykdomojo mechanizmo elgesį. Šias komandas galima vadinti primityvais. Vėliau atsiradę kiti *Tex* pagrindu veikiantys mechanizmai (žymiausi *pdfTex*, *XeTex* ir *LuaTex*) naudoja tas pačias D.Knuth'o *Tex* primityvias komandas, bet turi dar kiekvienas savo papildomų primityvų, kurios leidžia išplėsti originalaus *Tex* funkcionalumo galimybes. Dėl skirtingų papildomų komandų šie trys mechanizmai nėra visiškai tarpusavyje suderinti. T. y failas aprašytas su vienu mechanizmu (pavyzdžiui su *pdfTex*) negali būti korektiškai įvykdytas su kitu (pavyzdžiui su *LuaTex*), nes tiesiog liks neatpažintos tam tikros, tik tam mechanizmui būdingos, primityvios komandos. Tačiau šie primityvai yra žemiausiame lygmenyje (angl. *low-level* esančios komandos skirtos *Tex* mechanizmui, o yra dar vienas „aukštesnis" tipografikos proceso valdymo lygmuo - makrokomandos.

Tex makrokomandos (angl. *Tex macros*) leidžia vartotojui sukurti labiau kompleksines instrukcijas ir „vienu sakiniu" paleisti veikti kelias (ar net keliasdešimt) primityvių *Tex* komandų. Kadangi, *Tex* yra programuojamas mechanizmas, tad suprogramuoti reikiamas naudingas makrokomandas galėjo kiekvienas pagal poreikį. Dabar jau yra tokių makrokomandų, kurių dėka pats *Tex* turi galimybę atpažinti kokiam būtent mechanizmui (*pdfTex*, *XeTex* ar *LuaTex*) skirtos įvedamos instrukcijos ir pritaikyti savo elgesį. Tai reiškia, kad *Tex*, gali „suimituoti" reikiamo primityvo veikimą savo (tam konkrečiam mechanizmui būdingomis) turimomis primityviomis komandomis. Paprastai tai pavyksta, nebent atsiranda kokia nors labai reta ir specifinė komanda, būdinga tik vienam iš jų ir nėra makrokomandos kaip kiti galėtų ją „suimituoti". Tuomet įvyksta klaida [17].

Norint valdyti *Tex* mechanizmą makrokomandomis ir jomis „vykdyti" primityvias komandas žemesniame lygmenyje, reikia jas išmanyti. Bent jau tas, kurios atsakingos už autorinio turinio rašymo organizavimą. Tam Leslie Lamport ir aprašė labiausiai reikalingų (rašant knygas, straipsnius ir kitas publikacijas ar dokumentus) makrokomandų rinkinį. Iš esmės *LaTex* yra didelis makrokomandų paketas, kurio prireikia norintiems rašyti savo (dažniausiai mokslinio turinio) tekstą. Šiame pakete esančios makrokomandos padės autoriui pačiam valdyti visus reikiamus tipografikos niuansus. Tokie nurodymai kaip teksto išdėstymas puslapyje, šrifto dydis, atstumas tarp eilučių, paraščių ar raidžių, naudojant *LaTex* makrokomandas nesunkiai perduodami vykdymui. Be to *LaTex*

yra suderinamas su kitais makrokomandų paketais. Prie jo galima prijungti lentelių braižymo, cheminių elementų diagramų piešimo ar matematinių išraiškų spausdinimo makrokomandų rinkinius. O dabar tapo įmanoma su *LaTex* suderinti ir trimačių objektų atvaizdavimą savo prezentaciniame turinyje [3.3].

Interneto forumuose neretai tenka perskaityti ir tokius sudurtinius pavadinimus kaip *pdfLaTex*. Šiuo atveju, tai reikštų, kad naudojamas *LaTex* makrokomandų paketas organizuojantis *pdfTex* mechanizmo procesą. Atitinkamai galėtų būti ir *LuaLaTex* arba *XeLaTex*. Patikslinti koks būtent *Tex* mechanizmas naudojamas (šiuo metu labai maža tikimybė susidurti su originalaus D.Knuth'o *Tex* naudojimu) gali būti svarbu, nes ne visi trys palaiko kai kurias, dabar dažnai reikalingas naudoti, funkcijas.

pdfTex atsirado 2001-ais metais ir plačiai paplito dėl atnaujinto formato, kuriame atvaizduojamas rezultatas. Šis mechanizmas atvaizdavo (angl. *render*) turinį tiesiai .pdf formato faile, tad vartotojams nebereikėjo konvertuoti originalų DVI į *PostScript*, o tuomet pastarąjį į PDF.

XeTex 2004 metais dar labiau išplėtė *Tex*naudotojų gretas, mat leido dirbti su tekstu užrašytu UTF-8 (angl. *Universal Transformation Format – 8-bit*) koduotėje. Atsirado galimybė tvarkyti įvairiakalbius tekstus įskaitant ir tokius sudėtingus šriftus kaip arabiški rašmenys. Tuo pačiu jis palaikė ir *OpenType* formato šriftą, o vėlesnės šio mechanizmo versijos, be jokių sunkumų, atvaizdavo *OpenType* pagrindu sukurtą matematinę tipografiką.

LuaTex naudojamas nuo 2016 metų, šiuo metu laikomas galingiausiu ir universaliausiu *Tex* mechanizmu [17]. Jis ne tik palaiko *pdfTex* ir *XeTex* funkcionalumą (nors ir vykdo procesus kitokiu principu), bet ir turi savo skriptinę programavimo kalbą *Lua*, kuri įgalina dar daugiau ir dar sudėtingesnio funkcionalumo vykdymą. Pavyzdžiui, į *LuaTex*, pagal nutylėjimą, jau yra integruota *MetaPost* [3.3] grafinė kalba, skirta braižyti/apipavidalinti sudėtingus matematinius grafikus ir kitus techninius brėžinius.

Tačiau retas šiandieninis naudotojas bendrauja su *Tex* mechanizmu tiesiogiai, per komandinę eilutę. Turinio autoriui paprastai svarbiau yra susitelkti į teksto apipavidalinimą, nei į priemones, vykdančias apipavidalinimo procesą. Tad dabar, kalbant apie publikacijų rengimą ir turinio tipografiką, turimas omenyje darbas su patogiuoju *LaTex* - šablonai pritaikyti *LaTex*, papildomos makrokomandos derinamos su *LaTex*, nauji makrokomandų paketai integruojami į *LaTex* ir t. t.

Panagrinėkime kaip veikia makrokomandos. Naudojantiems *LaTex* įprasta matyti .tex išplėtimo failą struktūruotą tokia tvarka, kur publikacijos turinys surašytas tarp eilučių \begin{document} ir \end{document}. Tačiau, jeigu pamatytume .tex failą, kuris parašytas originalia D.Knuth'o *Tex* kalba arba tiesiog tas komandas, kurias kompiliuoja *Tex* mechanizmas, tokios struktūros ten nerastume. Iš tikrųjų, beveik jokios aiškios struktūros ten nėra, .tex failo turinys būtų tiesiog viena ilga spaudos ženklų seka (įskaitant ir perkėlimo į kita eilutę (angl. *line break*) ženklą). Spaudos ženklų išdėstymo būdų šioje sekoje gali būti begalė, taigi, *Tex* kompiliatoriaus užduotis iteruoti per kiekvieną jų ir priskirti, pagal reikšmę, atitinkamai kategorijai.

Šis priskyrimas yra viena pagrindinių *Tex* koncepsijų. Kiekvienam spaudos ženklui priskiriama reikšmė (skaičius), vadinama kategorijos kodu. Iš viso yra 16 kategorijų (skaičiuojama nuo 0 iki 15). Patikrinęs kiekvieno sekoje esančio spaudos ženklo reikšmę, pagal specialią kategorijų „lentelę“,*Tex* mechanizmas priskiria atitinkamos kategorijos numerį. Tolimesniam kompiliavimo procesui tuomet tampa svarbu nebe koks būtent simbolis yra sekoje, o kokiai kategorijai jis pri-

klauso (koks numeris priskirtas). Pagal šį kodą *Tex* kompiliatoriui „tampa aišku" ką su tuo spaudos ženklu reikia daryti.

Dirbant su *LaTex*, su šiais kategorijų kodais niekada netenka susidurti. Nebent įvyksta kokia nors klaida ir pranešime apie klaidą parodoma tam tikrų skaičių seka, kurios dalis būtų kategorijos kodas. Tačiau *Tex* mechanizmui šie skaičiai yra esminis komponentas, organizuojantis veikimą. Pavyzdžiui, „radęs" sekoje štai tokį „\" simbolį, mechanizmas priskirs jam kategorijos kodą „0", ir, pabaigęs kodų priskirymo procesą, pagal tą nulį, „žinos", kad tai yra *Tex* komandos pradžia, o ne simbolis, kurį reikia atvaizduoti išeities faile.

Įsivaizduokime, kad dirbama su failu *mano.tex*, kurio viduje yra toks spaudos ženklų fragmentas *Labas \jobname* (komanda \jobname skirta atspausdinti failo su .tex išplėtimu pavadinimą, arba, jeigu naudojamos \products ar \components komandos, tuomet tų produktų ar komponentų pavadinimus [1]). *Tex* kompiliatorius „mato" visus šio fragmento spaudos ženklus kaip sveikuosius skaičius (kodus), kurio viena dalis sudaryta iš dešimtainės spaudos ženklo *ASCII* kodo išraiškos:

L	a	b	a	s		\	j	o	b	n	a	m	e	
76	97	98	97	115	32	92	106	111	98	110	97	109	101	32

1 lentelė. ASCII kodai

O kita dalis iš priskirtos (nuo 0 iki 15) kategorijos:

L	a	b	a	s		\	j	o	b	n	a	m	e	
11	11	11	11	11	10	0	11	11	11	11	11	11	11	10

2 lentelė. Kategorijos kodai

Šių skaičių radimas yra pirmoji įvesties vykdymo proceso dalis, vadinama „nuskaitymu". Toliau kompiliatoriui reikia suprasti kaip elgtis su turimomis (šiame pavyzdyje trimis (11, 10 ir 0) kategorijomis. Atpažinęs pavyzdžiui 11-tą kategoriją („raidės"), *Tex* mechanizmas, naudodamas aukščiau parodytas reikšmes, pagal specialią formulę paskaičiuos sudėtinę reikšmę, vadinamą „spaudos ženklo žetonu" (angl. *character token*). Žetono reikšmė toliau bus perduota į vidinį, raidžių atvaizdavimo (nes 11-tos kategorijos spaudos ženklai skirti atvaizdavimui išeities faile) algoritmą. Galiausiai, šios kategorijos spaudos ženklai bus tiesiog atspausdinti pasirinkto formato faile [18].

Formulė, pagal kurią apskaičiuojama žetono reikšmė priklauso nuo konkretaus *Tex* mechanizmo. Kuomet buvo pereita prie 8-ių bitų dydžio simbolių, žetonas buvo apskaičiuojamas pagal formulę $T = 256 * C + A$, čia C yra kategorijos numeris, o A yra simbolio *ASCII* kodas. *Tex* mechanizmams, kurie „supranta" *Unicode* koduotės šriftus (*XeTex* ir *LuaTex*), tokios formulės nepakanka, nes atsiradus įvairiakalbėms abėcėlėms, spaudos ženklų kodai viršijo 256 ribą. Pavyzdžiui *XeTex* žetonus skaičiuoja naudodamas $T = 2^{21} * C + A$ (kur A yra simbolio *Unicode* kodas), o*LuaTex* turi dar sudėtingesnį konceptą [18].

Atpažinęs 10-tą kategoriją („tarpai"), *Tex* mechanizmas žino, kad tai tarpas tarp spaudos ženklų (*ASCII* kodas= 32). Toks tarpas traktuojamas kaip savotiška „guma", kuria „surišti" žodžiai tarpusavyje ir ji gali išsitempti ar susitrakti, priklausomai nuo to, kiek žodžių gali tilpti į nustatytą sulygiuotą eilutę. Spaudos ženklas „tarpas" *Tex* kompiliatoriui nėra fiksuoto dydžio tarpžodinis atstumas. Jis dinamiškai didėja arba mažėja, siekiant tolygiai surikiuoti visas pastraipos eilutes. Reikia atkreipti dėmesį, kad ne visi „tarpai" gauna kategorijos kodą „10". Kai kurie tarpai, tam

tikromis aplinkybėmis, tiesiog praleidžiami. Pavyzdžiui, jeigu renkant tekstą tarp žodžių bus padėti trys tarpai, kompiliatorius žodžių sujungimui naudos tik pirmąjį, o likusieji du bus tiesiog ignoruojami. Taip pat visi tarpai palikti po specialios komandos pavadinimo bus absorbuoti ir niekam nenaudojami.

Atpažinęs nulinę kategoriją, t. y. „sutikęs“ į kairę pasvirusį brūkšnį (angl. *escape character*), *Tex* mechanizmas persijungia į visiškai kitokį skaitymo režimą. Jam tampa aišku, kad prasideda komanda, kurią reikės vykdyti, taigi, labai svarbu kokios kategorijos yra po pasvirusio brūkšnelio sekantis spaudos ženklas.

Po nulinės kategorijos gali sekti dviejų tipų spaudos ženklai: 11-tos kategorijos (kaip mūsų nagrinėjamame pavyzdyje) arba ne 11-tos kategorijos. Jeigu po nulinės kategorijos simbolio seka 11-tos kategorijos spaudos ženklai (iki tarpo, kuris priklauso 10-tai kategorijai), šie simboliai laikomi „kontrolės žodžiu“, kitaip sakant atpažįstami kaip komandos pavadinimas. Pavyzdžiui čia nagrinėjamas „jobname“ yra komandos pavadinimas, arba „kontrolės žodis“. Kuomet po nulinės kategorijos seka ne 11-tos kategorijos spaudos ženklas (tarpas, kitas pasviręs brūkšnys ar pan.), jis laikomi ne „kontrolės žodžiu“, o „kontrolės simboliu“ ir tuomet proceso vykdymas priklauso nuo to simbolio. Bet kokiu atveju, *Tex* mechanizmas „supranta“, kad bet kokios kategorijos spaudos ženklai, sekantys iš karto po nulio, nėra skirti atvaizdavimui faile (jiems nereikia skaičiuoti spaudos ženklo žetonų), jie skirti komandoms aprašyti, tad jiems reikia skaičiuoti „komandos žetono“ reikšmę (angl. *command token*).

Komandos žetono reikšmė sudaroma kitaip nei spaudos ženklų žetono. Čia, pagal komandos pavadinimo spaudos ženklų kodus skaičiuojama maišos funkcija. Pavyzdžiui nagrinėjamos komandos „jobname“ maišos funkcijos kodas (angl. *hash code*), kurią suskaičiuoja *Tex* mechanizmas yra 5504 [18]. Ši reikšmė perduodama toliau. *Tex* kompiliatoriaus viduje turi savotišką komandų „žodyną“, kuriame nurodoma, koks algoritmas turi būti paleistas esant vienokiai ar kitokiai komandai. Šiame „žodyne“ komandos saugomos ne žodžiais (pavyzdžiui „jobname“) o jų maišos funkcijos kodais (pavyzdžiui 5504). Be to, ten aprašyta ne tik kokį algoritmą vykdyti atpažinus reikšmę, bet ir kada ji turi būti vykdoma. Kai kurios komandos gali būti vykdomos ne iš karto, o priklausomai nuo prieš ar po esančių komandų. Tai ypač dažnai nutinka, kuomet „atrandama“ makrokomanda, kuri yra kelių primityvių komandų rinkinys. Aptikus būtent tokią, ji skaidoma į kelias smulkesnes, o tuomet tų smulkesnių aprašymo vėl ieškoma „žodyne“ ir, atradus, vykdoma taip, kaip nurodyta [18].

1.3. TEX šrifto atvaizdavimas

11-tai kategorijai priskiriami tie spaudos ženklai, kuriuos reikia atvaizduoti išeities faile (kokio formato jis bebūtų). Panagrinėkime, kaip *Tex* vyksta atvaizdavimo procesas. Kaip buvo minėta, D.Knuth'as kartu su *Tex* programavimo kalba ir veikimo principu sukūrė dar ir atskirą šrifto technologiją, kurią pavadino *MetaFont*. Dar ir šiandien ši technologija yra galingas, funkcionalus įrankis, tačiau reikalaujantis specifinių žinių ir juo naudojimosi įgūdžių [16].

MetaFont yra ir programavimo kalba, aprašanti rastrinius šriftus, ir, tuo pačiu, interpretatorius, vykdantis *MetaFont* kodą. Šios technologijos užduotis - generuoti spaudos ženklų rastrus (angl. *bitmap*). Iš esmės, *MetaFont* yra schema, aprašanti kaip „piešti“ tam tikras spaudos ženklų grupes. Tai savo laiku buvo visiškai naujas konceptas, nes dauguma šriftų buvo tiesiog esamų spaudos ženklų „piešinių“ rinkinys. *MetaFont* vadovaujasi preciziškai tiksliomis „piešimo“ taisyklėmis, kurios valdomos specialiais nustatymais. Priklausomai nuo to, kokie paduodami parametrai, *MetaFont* pritaiko savo veikimą ir išrastruoja tikslų raidės (ar kito simbolio) atvaizdą. Toks principas

užtikrina, kad esant tiems patiems nustatymams, atvaizduotos raidės bus identiškos bet kokiame atvaizdavimo/spausdinimo įrenginyje [21].

Spaudos ženklų atvaizdavimui, tiksliau, kiekvienam jų fragmentui (kojelės, uodegėlės, kabliukai, apskritimai, etc) suformuoti,*MetaFont* naudoja geometrines lygtis. Šios kalbos išskirtinumas, kaip ir *Tex*, tas, kad ji programuojama makrokomandomis. Makrokomandomis kontroliuojamos kiekvieno spaudos ženklo savybės, perduodant skaitines reikšmes kaip argumentus geometrinėms lygtims. Priklausomai nuo šių reikšmių dėliojami rastro taškai. Spaudos ženklų kreivių konstravimo principai aprašyti, specialiame *MetaFont* formų „žodyne“, tad programuotojui pačiam nereikia nuolat skaičiuoti kontrolinių kreivių taškų. Dauguma šrifto charakteristikų (dažniausiai raidžių aukštis, plotas, pasvirimo laipsnis ar pan.) ateina iš *Tex* makrokomandų, o tuomet jau iš kito specialaus failo (angl. *source file*) imami visi kiti skaitiniai duomenys (geometriniai skaičiavimai ir t.t.). Šios lygtys tiko beveik visiems tuo metu vartotojams reikalingiems šriftams (daugumoje anglų kalbos abėcėlei ir skyrybos ženklams), tad pagal jas kokybiškai buvo atvaizduojamas kiekvienas (11 kategorijos) spaudos ženklas.

Turėti ne paruoštus spaudos ženklų atvaizdus, o algoritmą, galintį „piešti“ pagal pageidavimus buvo didžiulis pokytis kompiuterinėje grafikoje. Sąlyginai, pagrindiniame įvesties šaltinyje reikėjo aprašyti ne tiek jau ir daug parametrų, o tuomet laisvai atvaizduoti visas abecelės raides. Tačiau vėliau *MetaFont* algoritmai spaudos ženklų rastravimui tapo per daug riboti.

MetaFont išeities failai yra .mf formato. Pagal juose esančius duomenis *MetaFont* gali didinti, mažinti, pakreipti, pasukti, apversti, paslinkti, perkelti ir kitaip transformuoti spaudos ženklus. Tačiau tai vyksta prieš rastravimo procesą. Atlikus, pagal nustatytus kriterijus, „piešimą“, rezultatas išsaugomas .pk (*packed*) formato faile (originaliai D.Knuth'as naudojo .gf (*generic font*) formatą, nes .pk formatas dar neegzistavo). Šiuose .pk failuose buvo statiniai raidžių *bitmapai*, kuriuos atvaizduoti galėjo specialios spausdintuvų tvarkyklės (angl. *drivers*). Tuo tarpu atvaizduoti šį formatą ekrane nebuvo įmanoma,nes nekintamo dydžio jau „nupieštas“ raidžių rastras neprisitaiko prie ekrano parametrų (dydžio ir rezoliucijos). Norint išsaugoti šrifto kokybę (peržiūrėti failą skirtingo dydžio ekranuose), failai buvo konvertuojami į .tfm (*Tex Font Metric*) formato failą, kuriame, simboliai yra kintamo dydžio. Taip galima buvo užtikrinti, kad atvaizduojami spaudos ženklai bus vienodi, nepriklausomai nuo naudojamos įrangos. Tačiau tuometinio rastro kokybė, tuometu vis dar nebuvo tokia, prie kokios esame pripratę šiandien, tad neilgai trukus, nuo ženklo „piešimo“ buvo pereita prie kontūrų „braižymo“ [34].

Spaudos ženklo kontūras yra kreivė, apibrėžianti raidės plotą. Kreivę prie sparčiai kintančių kompiuterinės grafikos poreikių (rezoliucija, mastelio pokyčiai, ir pan.) pritaikyti žymiai lengviau. Kartu su šiuo konceptu atsirado puslapio aprašomoji kalba (angl. *page descriptive language*) *PostScript*.Pastaroji .tfm faile esančius spaudos ženklų rastrų kontūrus „pertvarko“ į Bezjė kreivių rinkinį ir jį atvaizduoja .ps (*PostScript*) arba .pdf formatu. Tikslūs *Tex* mechanizmo skaičiavimai generavo puikius duomenis kreivėms „braižyti“, tad šis naujas principas (vektorinė grafika) užtikrino nepriekaištingos kokybės spaudos ženklų atvaizdavimą [35].

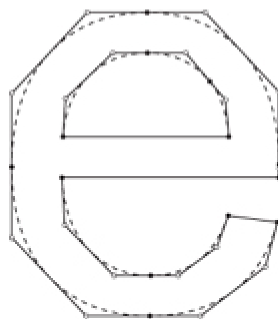
1.3.1. Vektorinės grafikos atvaizdavimo algoritmas

Integravus *PostScript*, *Tex* šriftas virto vektorine grafika. Vektorinis spaudos ženklas yra sudarytas iš uždarų trajektorijų ir kreivių sluoksnių. Šios trajektorijos yra kvadratinės ir kubinės Bezjė kreivės, kurias išraižo piešimo mechanizmas. Žvelgiant į vektorinę struktūrą iš bet kokio atstumo (artinant ar tolinant), jos kontūrų atvaizdavimo kokybė nesikeičia, neatsiranda „išplaukusios“ linijos ir nepradeda matytis pikselių mozaika. Tai yra esminis vektorių privalumas lyginant juos

su rastruojamu atvaizdu, kuris tėra tik spalvų reikšmių masyvas. Trajektorijos ir kreivės (bendrai vadinami vektoriais) gali būtų naudojamos ne tik teksto, bet iš esmės bet kokiai struktūrai ar formai atvaizduoti. Šios formos atvaizdavimo kokybė nepriklausys nuo turimo ekrano, ar kito vaizdo resurso, rezoliucijos.

Norint *Tex* spaudos ženklus matyti 3D scenoje (kaip trimačio objekto žymes), jie turės būti teseliuojami (padalinti į trikampius), nes *GPU* gavęs šešėliuoklių [2.2] duomenis gali „piešti" tik trikampius. Tuo tarpu vektorinės grafikos atvaizdavimo algoritmams taikomi kriterijai yra ne tik nuo rezoliucijos nepriklausomas rezultatas, bet ir kuo mažiau geometrinės reprezentacijos (kuo mažiau trikampių). Šiuolaikiniai grafikos apdorojimo įrenginiai, palyginus, neblogai atvaizduoja glotnius objektus, sudarytus iš trikampių, bet, anot straipsnio [26] autorių, tą patį glotnų objektą galima atvaizduoti dar geriau, naudojant vektorius.

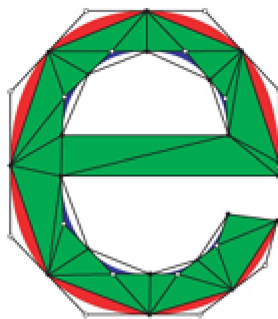
Straipsnis [26] aprašo algoritmą, kurio esminis konceptas - parametrinės kreivės $[x(t), y(t)]$ pavertimas neišreikštine plokštumos kreivės forma $f(x, y) = 0$. Išgaubta Bezjė kreivės dalis (jos kontroliniai taškai) atvaizduojama kaip poligonas, o šešeliuoklė, pagal kreivės neišreikštinę formą, nustato ar pikselis yra formos viduje ar išorėje. Šis procesas efektyvesnis nei įprasta *GPU* interpoliacija, nes tereikia pasirinkti tinkamą neišreikštinę formą. Pavyzdyje [26] nagrinėjamas *TrueType* formato spaudos ženklo atvaizdavimas, kurio kontūrus apibrėžia kvadratinių B-splainų ir tiesių segmentų derinys:



1 pav. Spaudos ženklo B-splainai ir linijos segmentai

Plotai dešinėje kreivės pusėje yra „viduje", kairėje pusėje esantys plotas yra „išorėje". Tuščiaviduriai taškai yra B-splainų kontroliniai taškai, parodantys užapvalinimus. Juodi taškai žymi vietą, kur apvalumas gali „nutrūkti" (pavyzdžiui atsiras smailas kampas).

Pradžiai, įvedami papildomi taškai tarp esamų kontrolinių taškų ir taip B-splainai paverčiami Bezjė splainais, kurie „lanksčiau" prisitaiko prie poligono kontūro. Taip kiekvienas buvęs B-splaino kontrolinis taškas atitiks kvadratinę Bezjė kreivę:



2 pav. Spaudos ženklo Bezjė splainai ir padalinimas į trikampius

Plotas esantis nepertrūkusių kreivių viduje padalinamas į trikampius ir tuomet suformuojama po atskirą trikampį kiekvienai kvadratinei Bezjė kreivei. Po tokio padalinimo gauname vidaus trikampius (žalios spalvos) ir kraštų trikampius, kurių viduje yra kreivės (raudonos ir mėlynos spalvos).

Vidaus trikampiai užpildomi ir atvaizduojami įprastai. O kraštų trikampiai padalinami į „išgaubtus“ ir „įgaubtus“, priklausomai nuo to, kokia kreivė yra to trikampio viduje. Straipsnio [26] autoriai teigia, kad toliau naudoja specialią šešeliuoklės programą, kuri nustato ar pikselis yra kreivės apriboto ploto viduje ar išorėje. Prieš paleidžiant šią programą, trikampių su kreivėmis viršūnėms priskiriamos UV koordinatės. Kuomet šie trikampiai bus atvaizduojami projekcijos perspektyvoje, *GPU* atitinkamai perspektyvai interpoliuos UV reikšmes ir perduos rezultatus pikselių šešeliuoklei [2.2]. Pastaroji, vietoj įprastų spalvos reikšmių, procedūrinei tekstūrai kurti naudos interpoliuotas reikšmes ir skaičiuos $U^2 - V$ rezultatą. Rezultato ženklas nurodys pikselio priskyrimą: išgaubtoms kreivėms teigiamas rezultatas reikš, kad pikselis yra kreivės išorėje, o neigiama reikšmė „nuspalvins“ pikselį kreivės viduje. Įgaubtoms kreivėms - atvirkščiai. Tokiu būdu atvaizduoto spaudos ženklo dydis yra proporcingas kontūro kreivės aprašymui, tad jo kraštai apsaugoti nuo įprasto rastro „nelygumų“ ar spaudos ženklo dydžio pokyčių. Kokybė neprarandama, nes „nuspalvinami“ tik tie pikseliai, kurie yra kreivės apriboto ploto viduje. Didėjant kreivei, didėja ir jos apribotas plotas.

Algoritmo autoriai [26] teigia, kad toks vektorinės grafikos atvaizdavimas procedūrinėje tekstūroje veikia todėl, kad, kaip matome paveikslėlyje, procedūrinės tekstūros koordinatės $[0, 0]$, $[0.5, 0]$ ir $[1, 1]$ yra ir kreivės $u(t) = t, v(t) = t^2$ Bezjė kontroliniai taškai. Tai ne kas kita o algebrinės (neišreikštinės) kreivės $U^2 - V = 0$ parametrizavimas. Sakykim, kad P yra sudėtinė transformacija iš UV erdvės į spaudos ženklo kreivių erdvę, tuomet į regos taško ir perspektyvos erdvę ir galiausiai į ekrano erdvę. Galiausiai tai bus projekcinis pikselių dėliojimas iš pradinės 2D į tokią 2D, kur visi ekrane atvaizduojami kvadratinės kreivės segmentai turės tokį P. Taip išeina, kad *GPU* interpoliuodamas procedūrinės tekstūros koordinates, iš tikrųjų skaičiuoja $P - 1$ reikšmę kiekvienam pikseliui. Dėl to UV erdvėje tampa labai paprasta sužinoti ar pikselis yra kreivės apriboto ploto viduje ar išorėje. Čia, vos dviejų aritmetinių veiksmų, implicitinė lygtis yra labai paprasta.

Galima būtų ir ekrano erdvėje išvesti algebrinę lygti kiekvienam kreivės segmentui, tačiau tuomet reikės daugybės operacijų skirtų kiekvieno sekančio kreivės koeficiento skaičiavimams ir daugybės operacijų šių koeficientų reikšmių įvertinimui pikselių šešėliuoklėje. Be to, šie koeficientai keičiasi priklausomai nuo projekcijos perspektyvos, o tai reikalautų atitinkamo visų lygčių perskaičiavimo. Tokio proceso koncepcija smarkiai nusileidžia straipsnyje [26] siūlomai UV procedūrinės tekstūros koordinačių interpoliacijai, kurią atlieka *GPU*.

Tai, kad procedūrinės tekstūros koordinačių priskyrimas, vienodas visoms (polinominėms)

kvadratinėms kreivėms, o šešėliuoklės lygtis yra kompaktiška ir nesudėtinga yra laimingas atsitiktinumas nutinkantis tik kvadratinių kreivių atveju. Norint atvaizduoti sudėtingesnius vektorinės grafikos meno kūrinius, kvadratinių kreivių gali nepakakti. Tokiu atveju prireikia kubinių splainų. Pasirodo šešėliuoklės lygtis ten irgi yra nesudėtinga, tačiau procedūrinės tekstūros koordinačių reikšmių priskyrimas labai netrivialus. Kaip algoritmas elgiasi su kubiniais splainais, gana išsamiai aprašyta [26] straipsnyje. Laikantis prielaidos, kad spaudos ženklai nėra įmantrūs meno kūriniai, norint juos atvaizduoti 2D plokštumoje pakanka kvadratinių Bezjė kreivių.

2. Trimačių objektų atvaizdavimas

Trimačių objektų atvaizdavimo istoriją galima skaičiuoti nuo pat tų laikų, kai dar nebuvo išrastas kompiuteris. Euklido geometrija, Renė Dekarto objektų formų ir pozicijų erdvėje aprašymas, J.J.Sylvesterio matricos - be visų šių „išradimų“ kompiuterinė grafika negalėtų egzistuoti. O kuomet pasaulis susipažino su Pjero Bezjė darbais ir prie kompiuterio prijungė pelę, trimačių objektų kūrimas tapo atskira mokslo (ir meno) šaka. Šešėliavimas bei tekstūros atvaizdavimo (angl. *texture mapping*) koncepcija atsirado dar iki 90-ųjų, o jau 1992-aisiais metais OpenGL tapo grafikos *API* (angl. *Application Programming Interface*) standartu. XXI-ame amžiuje, tikriausiai nėra žmogaus, kuris nežinotų, kad dabartinė animacija, kompiuteriniai žaidimai ar erdvinės nuotraukos išmaniajame telefone yra 3D grafikos rezultatas, tačiau tikrai ne kiekvienas žino, kad visi šie, labai tikroviški, vaizdai tėra tik didelis skaitinių reikšmių rinkinys.

2.1. Kas yra WebGL

Dažnai manoma, kad WebGL yra tiesiog įrankis kurti trimačius objektus. Tai nėra visiškai teisinga. WebGl labiau yra rastravimo mechanizmas (angl. *rasterization engine*) [32]. 1992-ais metais atsiradusi OpenGL aplikacijų programavimo sąsaja (*API*), tapo atviru, t. y. ne komerciniu standartu, turėjusiu daug įtakos trimatės grafikos, programinės įrangos ir net kino filmų kūrimo srityse. WebGL kilo iš 2007 metais išleistos OpenGL ES 2.0 versijos. Ši versija skirta įterptinėms sistemoms (angl. *embedded system*). 2012 metais atnaujinta OpenGL ES 3.0 versija sąlygojo daugybės įrenginių, tokių kaip išmanieji telefonai, planšetiniai kompiuteriai, žaidimų konsolės ir kt. galimybes palaikyti 3D grafiką [27].

OpenGL ES 2.0 versijoje atsirado naujas principas - programuojama šešėliuoklė (angl. *programmable shader function*). Šešėliuoklė, tai programa parašyta šešėliavimo kalba - į C programavimo kalbą panaši kalba, kuria galima suprogramuoti įspūdingus vaizdo efektus. Ši programavimo kalba pavadinta GLSL (*OpenGL Shading Language*). OpenGL šešėliuoklių programavimo kalba skirta įterptinėms sistemoms yra GLSL ES (*GLSL for Embedded Systems*). WebGL šešėliuoklės suprogramuotos būtent GLSL ES kalba [27].

Norint kurti trimačius objektus naudojant vien tik WebGL, reikia gerai išmanyti erdvinę matematiką - matricas, normalizuotas koordinates, erdvines figūras (pavyzdžiui: nupjautinį kūgį), vektorinę sandaugą, interpoliacijas, šviesos atvaizdo skaičiavimus, kompleksinius skaičius ir daugybę kitų dalykų. WebGL mechanizmas savaime šių formų apskaičiuoti nemoka. Kuomet programuotojas surašo visus paskaičiavimus, WebGL juos supranta kaip instrukciją pagal kurią reikia „nupiešti“ taškus, linijas ar trikampius. Be detalių nurodymų kaip ir ką „piešti“ WebGL nieko nuveikti negalės. Šis „piešimas“ yra rastravimas, perduodamas į *GPU* [31].

Rastras yra dvimatė koordinačių X ir Y plokštuma. O rastravimas yra taškų šioje plokštumoje

išdėstymas laikantis matomumo principo (angl. *visibility problem*). Matomumas nusako kurios trimačio objekto dalys yra matomos žiūrovui (kamerai), o kurios tampa „nematomomis“, nes jas užstoja kitos objekto dalys arba iš vis jos išeina už žiūrovo (kameros) matymo lauko ribų [31].

Tas pačias instrukcijas kaip rastruoti taškus, linijas ar trikampius (iš pastarųjų sudaroma trimačio objekto visuma) turi suprasti ir kompiuterio grafinis procesorius (*GPU*) [31].

2.2. Kaip veikia WebGL

Iš esmės, programuotojo instrukcijas kaip rastruoti trimačio objekto dalis, sudaro funkcijų poros. Poroje viena funkcija vadinama *viršūnių šešėliuoklė* (angl. *vertex shader*), o kita *fragmentų šešėliuoklė* (angl. *fragment shader*). Abi šios funkcijos parašytos tipizuota GLSL ES programavimo kalba.

Viršūnių šešėliuoklė skirta apskaičiuoti viršūnių (angl. *vertex*) savybes (poziciją erdvėje ir dydį). Viršūne laikomas taškas dvimatėje arba trimatėje erdvėje, kuriame susikerta dvimačio ar trimačio objekto briaunos. Pavyzdžiui: trys trikampio viršūnės, aštuoni kubo kampai, ir pan. Pagal funkcijai pateiktas viršūnės koordinates ir pikselių skaičių, nustatoma tos viršūnės pozicija ir dydis (jeigu pastarasis nenurodytas kitaip, jis bus 1.0, t. y. vienas pikselis). Šešėliuoklėje gavęs visus šiuos duomenis, WebGL mechanizmas supranta kur erdvėje išdėstyti pikseliai, kuriuos reikia „nuspalvinti“ ir gali pradėt „piešti“ taškus, linijas arba trikampius) [27].

Fragmentų šešėliuoklė kviečiama pradėtame rastravimo procese. Ji nustato kiekvieno „piešiamo“ primityvo fragmento spalvą. „Fragmentas“ yra WebGL terminas, nusakantis vieną vienetą iš kurių sudarytas paveikslėlis. Panašiai kaip pikselis [27]. Kiekvieno fragmento spalvos nustatymas yra šviesinimo (angl. *lightening*) procesas. Kadangi spalvos reikšmė skaičiuojama pagal RGBA formatą, bet kokios pirmos trys (RGB) reikšmės didesnės už nulį reikš kitokią spalvą nei juoda (juodos spalvos RGB yra 0, 0, 0), reiškia šių skaičių perdavimas funkcijai yra juodo taško spalvinimas šviesesne spalva, t. y. šviesinimas. Priskyrus norimą kodą (juodą arba šviesesnį), fragmentas ekrane bus atvaizduotas pasirinkta spalva.

Šiuo procesu ir paremtas visas WebGL veikimas. Nuolat kviečiamos šių funkcijų poros, ir per jas perduodamos instrukcijos vykdomos *GPU* lygmenyje. Šešėliuoklės inicializuojamos per funkciją *initShaders()*, kuriai į parametrus paduodama viršūnių ir fragmentų šešėliuoklių pora *String* formatu. Per šias funkcijas perduodamų duomenų primityvumas ar sudėtingumas priklauso išskirtinai nuo programuotojo, o ne nuo WebGL [32].

Kuriant trimačius objektus svarbu ne tik „kaip piešti“ bet ir „ant ko“ piešti. Toks piešimo paviršius yra vadinamas „drobė“ (angl. *canvas*). Tai nėra kažkoks specifinis WebGL terminas, drobė atsirado su HTML5 ir turi savo *<canvas>* HTML žymę (angl. *tag*). Ši žymė apibrėžia piešimo plotą internetiniame puslapyje, kuriame, naudojant JavaScript, galima dinamiškai piešti kompiuterinę grafiką. Panašiai yra ir kuriant trimačius objektus. Pradedant dirbti bet kokia WebGL programa, visuomet pradžioje reikia nustatyti drobės dydį [27].

Dydis turi atitikti atvaizduojamą sritį. Kaip ir bet kuris paveikslėlis, drobė (ją galima laikyti tuščiu paveikslėliu, kuriame dar niekas nepavaizduota) turi du dydžius. Vienas nurodo kiek į drobę telpa pikselių. Antras nurodo kiek jų yra atvaizduojami. Toliau kalbant apie drobę, omenyje bus turimas tas jos dydis, kuris atvaizduojamas, o ne bendras pikselių kiekis.

Aprašius drobės dydį per HTML žymę, ja galima naudotis rastruojant su WebGL. Tačiau toks drobės aprašymas veiks tik *<canvas>* žymę palaikančiose naršyklėse. Jeigu naršyklė šios žymės

nepalaiko, tuomet žymė bus ignoruojama ir ekrane nebus nieko pavaizduota [27]. Svarbu atkreipti dėmesį, kad WebGL nei už drobę, nei už jos atvaizdavimą nėra atsakingas. Viskas, ką gali WebGL, tai, naudojant dvi šešėliuokles sudėlioti pikselius ant tinkamo dydžio, HTML dokumente aprašytos, drobės. Kitaip sakant, *rastruoti* - sukurti rastrą. Už visus kitus sprendimus atsakingas programuotojas. Norint išrastruoti įmantresnį objektą, pavyzdžiui spaudos ženklą, reikia padavinėti funkcijoms sudėtingesnius duomenis, tad būtent čia ir reikalingos erdvinės matematikos žinios. Sudėtingesni duomenys sąlygos sudėtingesnius šešėliuoklių darbo rezultatus, tad bus atvaizduojamos sudėtingesnės formos. Taip pat nuo programuotojo priklauso ir kaip duomenys bus perduodami į *GPU*. Straipsnyje [32] išskiriami keturi būdai:

Buferiai ir Atributiniai kintamieji Buferiai yra dvejetainių duomenų masyvai skirti perduoti duomenis į *GPU*. Sukūrus buferį kaip atskirą objektą jame galima saugoti viršūnių pozicijas, statmenų matmenis, tekstūros koordinates (UV), verteksų spalvas ir bet kokius kitus rastravimui reikalingus duomenis. Atributiniai kintamieji nurodo kaip konkrečiai „ištraukti“ duomenis, saugomus buferiuose. Pavyzdžiui buferyje galima saugoti viršūnių koordinačių reikšmes po tris viename masyvo elemente. Kadangi GLSL ES yra tipizuota kalba, privaloma pasirinkti duomenų tipą. Pavyzdyje [32] pasirinkti 32 bitų *float* tipo (liet. *slankiojo kablelio*) skaičiai. Tuomet atributuose nurodoma iš kokio konkrečiai buferio reikia „imti“ reikšmes, bei, kad kiekviename elemente reikia tikėtis trijų, 32 bitų, slankiojo kablelio reikšmių. Atributiniuose kintamuosiuose taip pat nurodoma kuriame buferio elemente šios reikšmės prasideda ir per kiek baitų reikia „pasislinkti“ norint iš vieno elemento patekti į kitą. Gavęs šias konkrečias instrukcijas, *GPU* gali vykdyti procesą.

Uniforminiai kintamieji (angl. *uniforms*) Tai iš bet kurios programos vietos pasiekiami, kintamieji, kuriems priskiriama reikšmė prieš paleidžiant šešėliuokles. Tokiuose kintamuosiuose paprastai saugomi abiems šešėliuoklėms bendri duomenys. Inicijavus tokius kintamuosius, ir perduodant jose saugomus duomenis, nebereikia kiekvieną kartą teikti instrukcijų kaip jų reikšmes „pasiimti“. *GPU* žinos kaip nuskaityti universalius duomenis ir galės vykdyti procesą. Be to, labai patogu šiuose kintamuosiuose keisti jų reikšmes pagal poreikį [27].

Varijuojami kintamieji (angl. *varyings*) Tai taip pat iš bet kurios programos vietos pasiekiami kintamieji, tačiau jie skirti perduoti duomenis iš viršūnių į fragmentų šešėliuoklę. Nors dažniausiai jie deklaruojami tuo pačiu vardu ir yra to paties tipo, bet jų reikšmės nėra perduodamos tokios kaip yra. Pieš patenkant į fragmentų šešėliuoklę duomenys interpoliuojami priklausomai nuo piešiamos formos, tuomet *GPU* taiko interpoliuotą reikšmę kiekvienam fragmentui [27].

Tekstūros Tai bet kada galintys būti panaudoti (angl. *randomly access*) duomenų masyvai, kuriuose dažniausiai saugomi paveikslėlių (angl. *image*) duomenys, pavyzdžiui spalvos. Fragmentų šešėliuoklės veikimo metu šie paveikslėlio duomenys nuskaitomi ir perduodami rastruojamos formos fragmentams. Tokiu būdu *GPU* nupiešia formą, kurios tekstūra (spalvos) yra tokios kaip pasirinktame paveikslėlyje [27]. Tačiau šiuose masyvuose galima saugoti ir kitokius duomenis, nebūtinai spalvas [32].

2.3. Šrifto atvaizdavimas WebGL

Intuityviai atrodo, kad teksto rašymas ir atvaizdavimasbė yra vienas pagrindinių funkcionalumų, būdingų programinėms įrangoms, tad įvesti ir paskui perskaityti norimą tekstą turėtų būti

paprastas ir lengvai įgyvendinamas procesas. Tačiau, WebGL nėra pats savaime daug gebantis mechanizmas, tad nenuostabu, kad jame nėra jokios įprastos (angl. *built-in*) teksto atvaizdavimo funkcijos. Kaip tik priešingai, pasirodo atvaizduoti tekstą, naudojant vien tik WebGL, yra nepaprastai sudėtinga. Norint „rašyti“ su WebGL šešėliuoklėmis, reikia pranešti ne tik apie nesuskaičiuojamą galybę pasaulio kalbų raidžių, bet ir kokį išlaikyti atstumą tarp tų raidžių, koks jų dydį, kad tekstą reiktų atvaizduoti nuo viršaus į apačią ir iš kairės į dešinę. O gal reikia iš dešinės į kairę? O jeigu ten ne vien abėcėlių raidės, o matematinės formulės, natos, hieroglifai? Visų šių (ir daugybės kitų) teksto atvaizdavimo niuansų aprašymas virsta be galo komplikuota užduotimi. Tokios žemo lygmens *API* kaip WebGL, tam tiesiog nepritaikytos. Tačiau tai toli gražu nereiškia, kad atvaizduoti tekstą per WebGL nėra jokių galimybių. Iš tikrųjų yra net keli būdai tekstui atvaizduoti, tačiau vien tik WebGL mechanizmo čia nepakaks.

2.3.1. Tekstas kaip HTML elementas

Atsiradus JavaScript bei HTML5 technologijoms, internetiniai puslapiai iš statinių virto dinaminiais. Juose, naudojant drobę, tapo įmanoma atvaizduoti judančius dvimačius paveikslėlius - pradedant šokančiais animaciniais herojais ir baigiant interaktyviais žemėlapiais. Su WebGL buvo nueita dar toliau, dabar galima HTML dokumentuose, naudojant JavaScript galimybes, atvaizduoti taip pat ir 3D grafiką - trimatės erdvės vaizdo žaidimus, trimates duomenų vizualizacijas, 3D objektus, kuriais galima manipuliuoti (vartyti ir apžiūrėti iš visų pusių) ir pan. Taigi, naudojant standartines internetines technologijas, galima demonstruoti ekrane (ir internete) su WebGL išpieštas 3D formas. Norint šias formas ar jų demonstravimo pobūdį aprašyti/keisti, pakanka sukurti/redaguoti .html ir/ar .js formato dokumentus. O tam tinka bet koks teksto redagavimo įrankis [27].

Laikantis prielaidos, kad tekstinė žymė savaime nėra trimatis objektas, o tik 2D tekstas internetiniame puslapyje, tuomet jam nebūtinas WebGL rastravimo procesas. Tiek HTML5, tiek JavaScript turi galimybių tekstui įterpti į internetinį puslapį, reiškia įmanoma tą įterpimą suderinti ir su 3D formos atvaizdavimu. Straipsnio [33] autorius tai įvardina kaip lengviausią kelią ir siūlo tekstą įterpti naudojant naršyklės ir JavaScript, o ne rastruoti jį su WebGL.

Kaip buvo minėta anksčiau [2.1], prieš pradedant piešti trimačius objektus, HTML dokumente turi būti aprašytas drobės dydis. Dabar tampa aišku, kad drobėje bus ne tik WebGL išrastruota forma bet ir tekstas šalia/virš/po ja. Reiškia tiek drobės žymę *<canvas>*, tiek žymę (angl. *tag*), kurioje bus tekstas, verta dėti į vieną HTML elementą - *<div>* konteinerį.

Kadangi viskas veikia interneto naršyklėje, dabar jau galima turimam konteineriui taikyti norimą stilizavimą (formą, spalvą, dydį, elgesį) naudojant CSS. Tokiu būdu nebelieka poreikio modifikuoti sudėtingo WebGL šešėliavimo proceso, o galima, palyginti paprastai, pasinaudoti .html formatui prieinamu funkcionalumu ir perdengti (angl. *overlay*) drobėje tekstą su trimačiu objektu.

Nors naudoti CSS yra nesudėtinga, tačiau galutinis rezultatas nebūtinai iškarto bus toks, kokio tikimasi. Paprastai įdėtas tekstinis laukas bus statinis, o norint, kad jis atliktų informatyvios žymės vaidmenį, reikia jį paversti dinaminiu. Tam reikės dviejų *<div>* konteinerių - vieno drobei, kito tekstui. Tuomet jie CSS aprašomi atskirai vienas nuo kito, tad tekstui galima suteikti pageidaujamą judėjimo kryptį, greitį ir kitus efektus. Tuo tarpu drobė su savo konfigūracijomis lieka „apačioje“, t.y. tekstas perdengia ją ir joje atvaizduojamus trimačius objektus.

Dabar beliko turimą tekstą HTML elemente „pririšti“ prie kurio nors trimačio objekto taško. Tai bus kiek sudėtingiau, nes reikės dirbti jau su WebGL kodu ir tinkamai aprašyti duomenis. Taip pat svarbu nepamiršti ir mokėti nurodyti perspektyvinę projekciją (angl. *perspective projection*).

Kaip buvo aptarta [1.3.1], kiekvienas pikselis turi savo P (transformaciją iš vienokios erdvės į

kitą), tai svarbu norint aprašyti perspektyvinę projekciją. *GPU* reikalinga taškų koordinačių erdvėje matrica, kuri galės būti paversta teksto erdve (angl. *clip space*). Tačiau priklausomai nuo regimo lauko, teksto matomumas vis kitoks (jis turi būti atvaizduojamas tai iš šono, tai iš viršaus ir pan), tad jo išdėstymo erdvėje koordinates reikia nuolat perskaičiuoti. Straipsnyje [30] nurodoma, kad įprastai, perduodant koordinačių matricą WebGL šešėliuoklių porai, jos reikšmės bus sudaugintos su viršūnių lokaliomis koordinatėmis (angl. *local space*) ir rezultatas taps nauja (pasikeitusios projekcijos) teksto erdve (angl. *clip space*). Tuomet reikšmės bus paverstos pikseliais, naudojamais tekstui atvaizduoti. Šios koordinačių reikšmės vadinamos UV koordinatėmis, o nuolatinis jų perskaičiavimas perspektyvinės projekcijos atitikimui - UV taškų interpoliavimu. Erdvę, kurioje interpoliuojamos UV koordinatės ir formulę, pagal kurią vyksta perskaičiavimas, kaip paaiškinta [1.3.1], priklauso nuo algoritmo - vieni yra greitesni ir efektyvesni, nei kiti.

Neatsižvelgti į perspektyvinę projekciją negalime, net kai žymė yra dvimatis objektas tvarkomas per HTML. 3D scena nuo dvimatės plokštumos skiriasi tuo, kad turi kelių dimensijų projekciją ir tai net plokščiame ekrane turi „matytis“. Šis efektas (perspektyvos „matymas“) yra realizuojamas tokiu principu, kad labiau nutolę objektai (ar jų dalys) atrodo mažesni už arčiau esančius objektus (ar jų dalis). Tad turint žymėje teksto eilutę (pavyzdžiui 2D matematinę formulę), pasukus ja šonu, „arčiau ekrano“ esantys simboliai turės būti atvaizduoti didesni, nei paskutiniai, esantys „ekrano gilumoje“.

Norint pasiekti tokį efektą su HTML žyme, vienas iš paprastesnių būdų UV X ir Y reikšmes padalinti iš Z reikšmės. Straipsnyje [30] pateikiamas toks pavyzdys: jeigu yra tiesė nuo taško (10, 15) iki taško (20, 15) ir ji yra 10 vienetų ilgio. Netaikant projekcijos perspektyvos, ji būtų brėžiama tiesiog 10 pikselių ilgio. Nuo pradinio iki galutinio taško. Taip nupiešus liniją, ji neatrodytų kaip esanti trimatėje erdvėje. Atrodytų, kad linija nubrėžta tiesiog plokštumoje. Bet jeigu pridedama dalyba iš Z reikšmės, tuomet keičiantis linijos pozicijai erdvėje (kintant Z reikšmei) ji piešiama skirtingo ilgio. Tuomet tampa akivaizdu, t. y. „matosi“, kad linija ne dvimatėje plokštumoje, o trimatėje erdvėje. 10 pikselių ilgis išlieka tik tuo atveju, jeigu Z reikšmė yra vienetas: $10/1 = 10$; $20/1 = 20$; $abs(10 - 20) = 10$. Tačiau, jeigu Z reikšmė didėja (linija tolsta), pavyzdžiui dvigubai ($Z = 2$), tuomet jau linija tampa per pusę trumpesnė: $10/2 = 5$; $20/2 = 10$; $abs(5 - 10) = 5$. O jeigu Z nutolsta dar labiau, pavyzdžiui per du trečdalius ($Z = 3$), linijos ilgis irgi lieka tik trečdalis nuo pradinės reikšmės: $10/3 = 3.333$; $20/3 = 6.666$; $abs(3.333 - 6.666) = 3.333$.

3 pav. Didėjant Z (linijai tolstant), atitinkamai mažėja X (kelio plotis) ir Y (eglutės aukštis)

Šis projekcijos principas tinka dvimatei HTML žymei, tuomet ji prisitaiko prie matymo perspektyvoje „mažėjančių“ matmenų. Pats WebGL rūpinasi trimačių objektų pozicionavimu tokiu pat būdu. Perduotos viršūnių šešėliuoklei reikšmės X, Y, Z ir W, kurios skirtos viršūnių pozicijai nustatyti, automatiškai padalinamos iš W. Žinant šią savybę, reikia tiesiog nepamiršti padavinėti

tokias matricas, kuriose W turi tokią pat reikšmę kaip Z [30]. Visą šią aritmetiką galima atlikti ir turimame JavaScript dokumente. Rezultatas bus tas pats - tekstas susietas su trimačio objekto tašku, judantis kartu ir bet kokioje perspektyvoje išlieka tinkamoje vietoje. Norint tokią 3D sceną su žyme panaudoti prezentaciniame kontekste (skaidrėse, publikacijoje ir pan.), nereikės atlikti paveikslėlių karpymo ir klijavimo ant viršaus procedūrų [3.2.1].

2.3.2. Tekstas kaip 3D scenos objektas

Aukščiau buvo aptartas paprasčiausias teksto įterpimo variantas per HTML elementą, tačiau yra būdų ir tekstą atvaizduoti naudojant patį WebGL mechanizmą, t. y. padavus šešėliuoklėms reikiamus duomenis, *GPU* išrastruos tekstą kaip bet kurį kitą 3D scenos objektą. Tokio teksto kaip scenos objekto pozicionavimas ir derinimas su kitais scenos objektais kontroliuojamas lygiai tiek pat, kiek kiti su WebGL kuriami objektai, tad iš esmės tekstas būtų lygiavertis 3D scenos „dalyvis“, o jo pozicionavimas kaip kito objekto žymės priklausytų nuo programuotojo sugebėjimų valdyti šešėliuokles.

Geometrija Su WebGL tikrai įmanoma atvaizduoti spaudos ženklus tuo pačiu principu kaip ir bet kokius kitus objektus, t. y. turimą spaudos ženklo kontūrą padalinti į trikampius ir trikampių koordinates perduoti WebGL mechanizmui. Tereikia mokėti apskaičiuoti kiek, kur ir kokių trikampių reikės, pateikti skaičiavimus šešeliuoklėms. Tuomet *GPU*, jeigu skaičiavimai teisingi, tekstą išrastruos. Dar daugiau, iš trikampių sudarius reikiamą spaudos ženklą, tampa įmanoma ne tik jį atvaizduoti kaip dvimatį, bet ir „iškelti“ (angl. *extrude*) formą ir paversti raide trimate. Tačiau apskaičiuoti kiek ir kokių trikampių reikės spaudos ženklams nėra paprasta užduotis. Pasak straipsnio [36] autoriaus, norint aptarti šiam veiksmui reikalingus algoritmus ir procesus prireiktų atskiro mokslinio darbo. Čia aptarsime tik pagrindinius spaudos ženklo padalinimo į trikampius veikimo principus. Pradžiai reikia turėti teksto eilutę (angl. *string*) išsaugotą tokiu formatu, kuriame būtų nurodytos spaudos ženklų Bezjė kreivių kontrolinių taškų koordinates. Turint Bezjė kreivių duomenis, reikėtų atskirti jomis apribotus plotus. Čia galimai tiktų [1.3.1] aprašytas algoritmas, bet gal įmanoma rasti ir kitokį. Turint visus plotus, juos patogu būtų surikiuoti mažėjančia tvarka, taip pat suindeksuoti kurie plotai yra su skylėmis, kurie yra kitų plotų viduje ir pan. Tuomet tuos plotus (atsižvelgiant į turimas skyles ar tarpusavio dalis) padalinti į trikampius ir gautas koordinates perduoti šešėliuoklėms.

Žinoma, beveik visom šio proceso dalim yra sukurta įvairių įrankių, galinčių palengvinti darbą. Straipsnyje [36] pateikiama nemažai nuorodų, tačiau sukonfigūruoti jas visas vienam procesui gali būti sudėtinga. Tai ir nelabai prasminga, kadangi jau yra vienas įrankis, kuris atliktų visą procesą nuo teksto eilutės „gavimo“ iki šešėliuoklių reikšmių perdavimo WebGL. Tai ThreeJS [4] biblioteka. Surinkus norimą tekstą, ThreeJS padalins jo spaudos ženklus į trikampius, pateiks reikiamus duomenis šešėliuoklėms ir atvaizduos su WebGL. Jeigu teksto eilutės negalima surinkti klaviatūra, galima būtų paduoti JSON (*JavaScript Object Notation*) formato failą, kuriame būtų reikiamo teksto Bezjė kreivių duomenys, o toliau procesą užbaigs ThreeJS.

Atrodytų paprasta, tačiau reikia turėti omeny, kad vienai raidei „išpiešti“ reikiamas trikampių kiekis yra didžiulis. Straipsnio [36] autoriaus paskaičiavimais eilutei *Hello World* atvaizduoti, WebGL išrastravo 7,396 trikampius ir 22,188 viršūnes. Tai nėra racionalu nei programavimo darbų apimtimi, nei procesoriaus veikimo prasme (palyginus su optimaliu Bezjė kreivių atvaizdavimu). Norint naudoti šrifto geometriją žymėms kurti racionalumo rodiklis nebūtų itin svarbus, juk žymė, palyginus, nėra didelės apimties tekstas, tačiau kiltų klausimas dėl kokybės. Geometrijos atvaiz-

davimas labai priklauso nuo naršyklės, monitoriaus ir kitų techninių aplinkybių (pavyzdžiui veiksmų, atliktų po tokios žymės sukūrimo (angl. *post-processing*)). Labai didelė tikimybė, kad žymės kokybė, kai kuriuose įrenginiuose, bus per prasta, o ir iš esmės rastruoto teksto atvaizdas neprilygs įprastam *Tex* kokybės vektoriniam standartui. Kalbant konkrečiai apie matematinės formulės žymę, t. y apie tekstą, kurio paprastai nesurinksi kompiuterio klaviatūra, o, labiausiai tikėtina, turėsi *LaTex* suformuotu .pdf formatu, lieka neaišku kaip iš .pdf failo „ištraukti“ spaudos ženklų Bezjė kreivių duomenis (ir sudėti juos į .json failą) ir ar tikrai čia jos yra tokios, kurių užtektų kokybiškam atvaizdavimui, gal būt *Tex* šrifto kreivių reikėtų ieškoti ne galutiniame .pdf formate, o ankstesniame veikimo etape?

Tekstūra Ko gero paprasčiausias būdas „nupiešti“ tekstą su WebGL yra sukurti tekstūrą su norimu tekstu. Tereikia turėti įrankį, galintį reikiamą tekstą išsaugoti kaip rastrinį paveikslėlį (angl. *bitmap*). Kitaip sakant, sukurti teksto atvaizdą. Vienos sukurtos tekstūros užtenka bet kokiam teksto kiekiui, tad skirtingai nuo aukščiau paminėtos geometrijos, kur kiekvienai raidei naudojama daugybė resursų, į vieną tekstūrą galima įrašyti ištisus puslapius teksto ir tai nepadidins atvaizdavimo kaštų. WebGL „nupieš“ tik vieną keturkampį (angl. *quad*), dvi jo puses (angl. *faces*) ir šešias viršūnes. Kiek tame keturkampyje bus spaudos ženklų svarbu tik todėl, kad kuo jų bus daugiau, tuo jie bus smulkesni, kad visi tilptų į nustatytus tekstūros „rėmus“.

Straipsnyje [36] minima, kad tekstūrą galima kurti ir be turimo teksto atvaizdo (nepaduodant rastrinio failo). Tam tereikia sukurti papildomą drobę ir ją kaip tekstūrą „uždėti“ ant aukščiau paminėti keturkampio (angl. *quad*). Kuriant tekstūrą tokiu būdu patogu tai, kad jokio failo su tekstu nereikia saugoti atmintyje, jis atvaizduojamas vykdymo metu „nuskaitant“ iš kodo. Tik reikia turėti omeny, kad tokiu atveju turi būti nurodoma ir kiekvieno žodžio pozicija drobėje. Turint failą apie teksto dalių išdėstymą galvoti nereikia, jos bus atvaizduotos tokios, kokios yra atvaizduotos atmintyje saugomame faile.

Techniškai, šis teksto atvaizdavimo su WebGL būdas tiktų žymei su matematine formule sukurti. Nors kurti tekstūros tiesiai kode neįmanoma, nes formulės nėra galimybės surinkti kompiuterio klaviatūra, tačiau su *LaTex* jau atvaizduotos formulės rastrinį paveikslėlį galima „pasigaminti“ palyginus nesudėtingai. Svarbu, kad failas su atvaizdu nebūtų labai didelis, kitaip kelti jį į WebGL ir versti tekstūra gali užimti daug laiko. Įrankių, galinčių *Tex* šriftą paversti rastriniu paveikslėliu jau yra sukurta, vienas jų [28] suteikia galimybę *Tex* formulę išsaugoti .jpg, .png ar .svg formatu. Dedikavus atsakomybę padaryti *Tex* šrifto atvaizdą specialiai tam pritaikytam įrankiui, likusią dalį nesunkiai įvykdys ThreeJS. Padavus .png failą su matematine formule, ThreeJS pavers paveikslėlį norimų savybių tekstūra ir atvaizduos ekrane kaip objektą 3D scenoje [17]. Manipuliuoti šia formule galima lygiai tiek pat, kiek ir bet kokia kita ThreeJS bibliotekoje sukurta tekstūra, tad kurti žymes per tekstūrą galėtų būti vienas iš problemos sprendimų. Straipsnyje [36] pateikiamos nuorodos ir į kitokius įrankius, skirtus tekstą versti tekstūra, tačiau taip pat pateikiami ir šio atvaizdavimo būdo trūkumai. Jau išrastruotas toks tekstas jį padidinus, apvertus ar kitaip transformavus, gali tapti neryškus arba kaip tik „kampuotas“ (matysis rastro pikseliai). Tokia kokybė gali netikti prie *Tex* standarto pripratusiems vartotojams. Ypač turint omeny, kad veliau, dedant objektą su tokia žyme į su *LaTex* kuriamą prezentacinį turinį, reikės dar kartą daryti jo paveikslėlį, taigi atvaizdo kokybė dar labiau sumažės (lyginant su likusio turinio kokybe).

Rastrinis šriftas Kaip teigia straipsnio [36] autorius, aukščiau išdėstyti būdai nėra tinkami norint atvaizduoti didelį teksto kiekį (pavyzdžiui pastraipą ar net puslapį tekstinio turinio). Skaičiuoti milijonus viršūnių spaudos ženklų atvaizdavimui yra per daug neefektyvu, o didelį teksto kiekį

pavertus viena tekstūra, spaudos ženklai gali būti per smulkūs, o galimybės kokybiškai tokį tekstą išdidinti yra ribotos. Didelies apimties tekstams atvaizduoti su WebGL yra kitas būdas, tačiau, kadangi tai nelabai aktualu žymių kūrimo kontekste (žymių turinys nebūna didelės apimties), konceptas bus tik trumpai paminėtas.

Rastrinis šriftas reiškia, kad kiekvienas spaudos ženklas atvaizduojamas atskirai vienoje tekstūroje, vadinamoje tesktūriniu atlasu. Sukuriant teksto eilutę vykdymo metu (angl. *at runtime*), kiekvienam spaudos ženklui ir plotui aplink jį parenkamas atitinkamas „piešinys" iš tekstūrinio atlaso ir atvaizduojamas atskirame keturkampyje (angl. *quad*). Kurti po vieną keturkampį kiekvienam atvaizduojamam spaudos ženklui, pagal straipsnyje [36] pateiktus skaičiavimus, maždaug 3.1 karto efektyviau, nei kurti kiekvienam ženklui trikampius (geometriją). Tačiau kiekvieno simbolio atvaizdui galioja tie patys dėsniai, kaip ir bet kokiam rastriniam paveikslėliui - smarkiai priartinus kontūrą, matysis pikselių kampai ir vaizdas bus neryškus. O jeigu yra žinoma, kad atvaizduotas tekstas bus smarkiai didinamas, tenka kurti dar vieną tekstūrinį atlasą su tokio dydžio spaudos ženklų pavyzdžiais, kokio reikia. Tuomet reikia suprogramuoti veiksmą, kad tekstą didinant, atvaizdavimui būtų naudojamas būtent jis.

Nors yra sukurta nemažai įrankių, kurie generuoja rastrinius spaudos ženklų rinkinius, kuriuos būtų galima panaudoti tekstūriniam atlasui, tačiau norint turėti galimybę tokiu būdu kurti matematinių formulių žymes, prieš tai, ko gero, tektų sukurti atskirą rinkinį su formulėse naudojamų simbolių „piešiniais". Tačiau, kai kuriuos simbolius būtų labai sudėtinga pritaikyti. Formulėse, skirtingai nei įprastame tekste, spaudos ženklai nebūtinai rikiuojami eilute vienas paskui kitą. Pavyzdžiui trupmenos dalybos ženklas („ilgas brūkšnys") kur tiek vardiklyje tiek skaitiklyje būtų kiti spaudos ženklai, arba laipsnis, kuriuo keliamas skaičius, arba indeksas ir panašių ženklų pozicijas aprašyti tekstūriniame atlase nebūtų paprasta užduotis. Tad nors šis būdas ir yra palyginus greitas ir paprastas atvaizduoti įprastam, kad ir labai didelės apimties, tekstui, jis netinka matematinės formulės žymei dėl riboto *pre-rastruotų* simbolių tekstūriniame atlase skaičiaus. O be to, tokios žymės kokybė, nebūtų smarkiai geresnė nei tik su tekstūra sukurtos žymės, nes spaudos ženklas vis tik būtų rastrinis paveikslėlis, o ne vektorinė grafika.

SDF Vienas iš variantų pagerinti žymės teksto kokybę ir priartinti jį prie vektorinės grafikos atvaizdavimo kokybės yra vietoj rastrinio spaudos ženklo aprašymo tekstūriniame atlase naudoti SDF (*Signed Distance Fields*. SDF (liet. *orientuoto atstumo laukai*) šriftas irgi yra atvaizduojamas pagal tekstūros atlasą, tik šį kartą atlase saugomas ne rastruotas spaudo ženklo „piešinys", o jo forma, pagal kurią paskui generuojamas ir atvaizduojamas spaudos ženklas. Naudojant zodį „forma", turima omeny, kad SDF yra tam tikras poligonas, kurio kiekviename pikselyje saugoma reikšmė nurodo atstumą iki artimiausio paviršiaus, o šio atstumo ženklas nurodo ar pikselis yra formos viduje (neigiamas) ar išorėje (teigiamas). Pasak straipsnio [36] autoriaus, tokiu būdu atsiranda galimybė atvaizduoti aukštos rezoliucijos formą (spaudos ženklą) naudojant mažos rezoliucijos SDF. Pavyzdžiui galima sukurti 16 pt dydžio SDF ir pagal jį atvaizduojamus spaudos ženklus didinti iki 100 pt dydžio neprarandant kokybės.

Kokybę užtikrina tai, kad SDF vietoj rastrui įprastų reikšmių (tarpinė spalva tarp dviejų pikselių) skaičiuojama bilinijinė interpoliacija, t. y. atstumo tarp dviejų taškų (pikselio ir artimiausio krašto) santykį. Šio santykio, reikšmė didėjant plotui reikšmingai nesikeičia, tad informacija apie spaudos ženklo formą yra labai panaši. Iš esmės, kuo didesnis SDF tekstūros atlase, tuo tikslesnė yra atstumo santykio reikšmė ir tuo didesnė yra atvaizduojamo spaudos ženklo kokybė. Tačiau, tai galioja tik tuo atveju, jeigu atstumo tarp taškų pokytis yra tiesinis. Jeigu atstumas iki artimiausio krašto nėra tiesinis, pavyzdžiui turint smailius kampus, interpoliacijos reikšmės tampa ne tokios

tikslios, tad didinant SDF smailūs kampai atvaizduojami bukesni arba užapvalinti. Šiems netikslumams išvengti naudojamas MSDF (*multi-channel SDF*). Be įprasto SDF čia dar veikia trys spalvų kanalai, kurių dėka spaudos ženklo kontūro kampai atvaizduojami kokybiškai.

MSDF Vieno spaudos ženklo aprašymui naudoti dar ir tris spalvų kanalus vienareikšmiškai reiškia didesnį duomenį kiekį tiek vienam simboliui, tiek visam MSDF formų tekstūriniam atlasui. Tačiau būtent tas daugiau duomenų turintis „sunkesnis“ atvaizdas ir garantuoja aukštesnę kokybę nei galima pasiekti su paprastu SDF. MSDF taip pat saugo taško santykinį atstumą iki artimiausio krašto, tačiau ten, kur atstumo pokytis nėra tiesinis (randa smailą kampą), pakeičiamas spalvos kanalas. Pikselis „išpiešiamas“ tik toje vietoje, kur sutampa dvi ar daugiau spalvų. Yra keletas papildomų SDF ir MSDF naudojimo technikų, skirtų išrastruoti vektorinės grafikos kokybę atitinkančius spaudos ženklus. Jas ir visą SDF veikimo procesą, išsamiai aprašo straipsnio [19] autorius.

Nors MSDF principas yra kitoks nei anksčiau [1.3.1] aprašyto algoritmo, tačiau jeigu su juo galima pasiekti vektorinės grafikos atvaizdavimo kokybę naudojant ThreeJS sugeneruotas šešėliuokles, galima būtų pamėginti pritaikyti šį būdą *Tex* kokybės žymėms atvaizduoti. Tereikia išspręsti problemą kaip sukurti matematinės formulės, kurią norime matyti žymės turinyje, spaudos ženklų tekstūrinį SDF atlasą. Straipsnyje [36] nurodoma, kad norint sukurti savo SDF, nestandartiniams simboliams (nerandamiems įprastų abėcėlių tekstūrų atlasuose), reikia savo simbolius pateikti *TrueType* formatu. Reiškia su *LaTex* sukurtą matematinę formulę reikia atvaizduoti ne įprastame .pdf, o .ttf formato faile [22].

3. TEX kokybės šrifto atvaizdavimas 3D scenoje

Išanalizavus kaip veikia *Tex* mechanizmas ir kaip veikia WebGL mechanizmas, beliko rasti būdų sujungti jų veikimo principus. D.Knuth'o palikta galimybė [1.1] jo sukurto kodo pagrindu kurti naujus, labiau patobulintus ir pritaikytus šiam laikmečiui įrankius, sąlygojo įvairių programų (ir programavimo kalbų) atsiradimą. Pastarųjų dėka, *Tex* kokybės šriftas atvaizduojamas įvairių formatų išeigos failuose (angl. *output file*). Būtent per formatų suderinamumą - kuomet *Tex* išeigos failo formatas suprantamas WebGL formato rėmuose - pavyko norimą matematinę išraišką įkomponuoti į 3D sceną, o galiausiai bendrą rezultatą (atvaizduotą trimatį objektą su žymėmis) įterpti į prezentacinį, su *LaTex* kuriamą turinį. Šiam tikslui įgyvendinti prireikė papildomų įrankių.

3.1. Kas yra KaTex

Vienas iš paprasčiausių būdų [2.3.1] sukurti žymę 3D scenoje yra pasinaudoti HTML funkcionalumu. Tačiau norint, kad žymės turinys būtų ne paprastas tekstas, o pagal *LaTex* makrokomandą atvaizduojama matematinė išraiška, pavyzdžiui Furjė transformacijos formulė [3.2.1], reikia papildomo įrankio, galinčio *Tex* šriftą paversti HTML elementu.

KaTex yra JavaScript programavimo kalba parašyta biblioteka, galinti realiu laiku generuoti *Tex* matematines išraiškas .html formatu. Atsiradus šiai bibliotekai, papildyti turinį matematinėmis formulėmis tapo įmanoma ne tik tekstinėse publikacijose, bet ir internetiniuose puslapiuose. *KaTex* ne tik palaiko D.Knuth'o „auksinį tipografinį standartą“ [2], bet ir atvaizduoja formules sinchroniškai, tad nereikia perkrauti puslapio. Taip pat, biblioteka veikia visose šiuolaikinėse naršyklėse

įskaitant Chrome, Safari, Firefox, Opera, ir IE [10]. Visi šie išvardinti dalykai yra didžiulis privalumas naudotojui. Taigi, *KaTex* yra kokybiškas, greitas, veikiantis visose naršyklėse, bet svarbiausia, gali atvaizduoti tas pačias išraiškas kaip ir*LaTex*, tik ne .pdf, o .html formato failuose. Tokiu pat formatu galima peržiūrėti ir su WebGL atvaizduotą trimatį objektą.

KaTex nėra vienintelė JavaScript biblioteka, leidžianti įkomponuoti *LaTex* formules HTML dokumentuose. Vienas iš dažniausiai minimų (ir lyginamų su *KaTex*) variantų yra biblioteka *MathJax*. Pastaroji irgi geba atvaizduoti *Tex* kalbos išraiškas, yra nemokama ir atviro kodo. *MathJax* oficialiame puslapyje [3] teigiama, kad ši biblioteka taip pat, kaip ir *KaTex* buvo sukurta išaugus akademinių publikacijų pateikimo ir pristatymo internete poreikiui.

Šių dviejų bibliotekų palyginimui yra net sukurtas atskiras įrankis (paskutinį kartą atnaujintas 2020 metų balandžio mėnesį) [11]. Lyginant *KaTex* bei dvi *MathJax* versijas, skaičiuojamas apdorojimo bei puslapio parsiuntimo į naršyklę procesų vykdymo laikas. Pateikiant pakankamai sudėtingas matematines išraiškas (pasikartojančios trupmenos, eilutės elementų sumavimas bei sandauga, formulė tekste, graikiškos abėcėlės raidės, Lorenzo transformacijų, Maxwell'o lygčių, vektorinės sandaugos) matuojama atvaizdavimo proceso trukmė. Atlikus testą, daugelį išraiškų *KaTex* atvaizdavo šiek tiek greičiau nei *MathJax*.

Šis veikimo trukmės skirtumas detaliai paaiškintas 2014 metų straipsnyje [25]. *MathJax*, pasirodo, labai preciziškai matuoja kiek teksto yra aplink kiekvieną atvaizduojamą formulę. Šie matavimai įgalina kuo kokybiškiau atvaizduoti kiekvieną formulės pikselį. Be to, *MathJax* veikia asinchroniškai. Reiškia formulės atvaizduojamos tik tada, kuomet visas puslapis iki galo atsiųstas į naršyklę. Pasak autoriaus, būtent dėl šių dviejų priežasčių *MathJax* veikia kiek lėčiau nei sinchroniškasis *KaTex*.

3.2. Kaip veikia KaTex

KaTex pagrindinė atvaizdavimo funkcija yra *katex.render()*. Jos dėka bet kokia *Tex* kalbos išraiška bus atvaizduota tame HTML elemente (angl. *tag*), koks bus nurodytas funkcijos argumentuose. Dažniausiai argumentuose nurodomas HTML elementas yra *div*. Šį elementą (su jame esančia formule), galima nesunkiai pozicionuoti, keisti įvairias jo savybes ar kitaip apipavidalinti naudojant CSS priemones. Tokiu būdu tipografinis procesas persikelia iš tekstinių į internetines publikacijas.

Be to, „patalpinus“ formulę į *div* elementą, tampa įmanoma ją komponuoti su kitu objektu - pavyzdžiui su grafika ar paveikslėliu. „Prikabinus“*div* su formule prie kitame HTML elemente esančio, atvaizdavimui paruošto, objekto, ji tampa to objekto žyme (angl. *label*). Būtent šitoks formulės panaudojimas ir buvo išbandytas praktikoje, turint su WebGL sukurtą trimatį objektą, egzistuojantį savo 3D scenoje.

Nors praktiniam eksperimentui veikimo trukmė nebuvo esminiu rodikliu, tiek *MathJax*, tiek *KaTex* yra tinkami *LaTex* žymei sukurti, tačiau *KaTex* biblioteką pavyko lengviau sukonfigūruoti reikiamai aplinkai, tad šio darbo praktinei daliai buvo naudojama būtent ji. Atlikus eksperimentą, paaiškėjo, kad *KaTex* puikiai tinka į *div* elementą įdėtą matematinę išraišką „perduoti“ į WebGL ir „prikabinti“ prie trimačio objekto kaip žymę, tad mėginti atkartoti to paties su *MathJax* prasmės nebuvo.

3.2.1. Dvimatė TeX kokybės žymė (kaip HTML elementas) 3D scenoje

Kadangi straipsnio [33] autorius kartu su instrukcijomis pateikė ir pavyzdinį išeities kodą, bei leidimą juo naudotis, buvo nuspręsta ne perrašyti viską nuo nulio, o pernaudoti kodą su tam tikromis modifikacijomis, atitinkančiomis šio darbo tikslą. Pakeitimui, kaip trimatis objektas pasirinktas paprastas stačiakampis gretasienis. Jis sukurtas nepamirštant projekcijos perspektyvos, kuomet šešėliuoklių porai paduodama matrica, kurioje W reikšmė lygi Z. Taip pat, pagal pavyzdį, formulei sukurtas atskiras HTML: elementas konteineris, pavadintas *floating-div*. Jis atskirai nuo drobės stilizuojamas CSS ir yra susietas su stačiakampio tašku [100, 0, 0, 1], kuris atitinka priekinio viršutinio dešinio kampo poziciją. Importuota *KaTex* biblioteka ir jos funkcijai *render()* perduota tokia *Tex* išraiška:

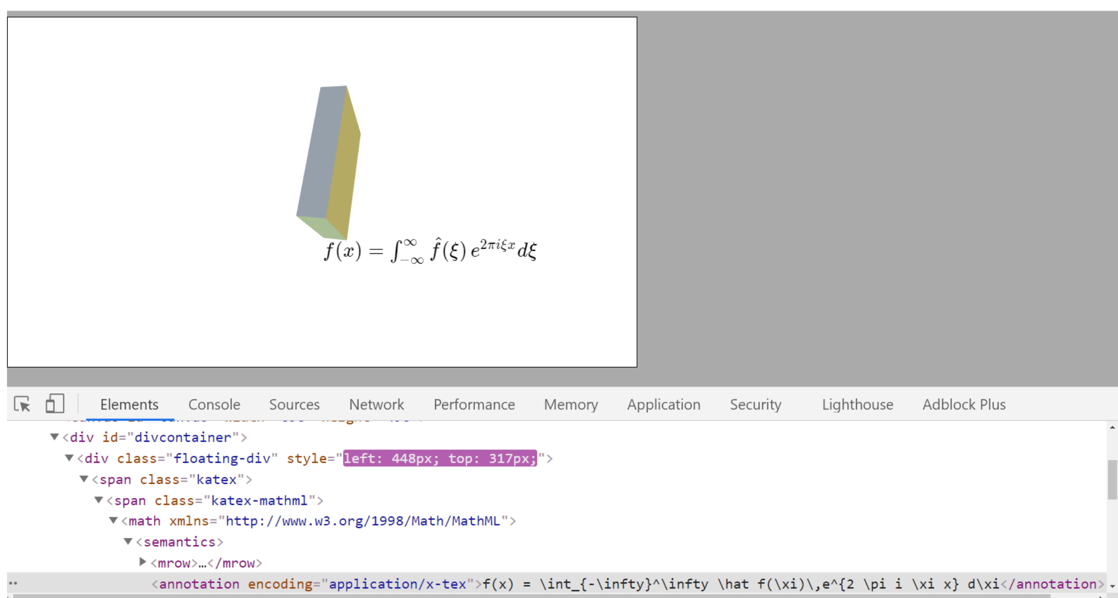
```
f(x) = \int_{-\infty}^\infty \hat f(\xi)\,e^{2 \pi i \xi x} d\xi
```

Ši išraiška patalpinta į HTML *floating-div* elementą ir naršyklėje ji turi būti atvaizduota kaip virš drobės judanti, susieta su gretasienio viršūne, Furjė transformacijos formulė:

$$f(x) = \int_{-\infty}^{\infty} \hat f(\xi)\, e^{2\pi i \xi x} d\xi \tag{3.1}$$

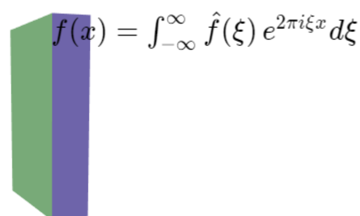
Kiti kodo aspektai nebuvo modifikuoti, jie puikiai tiko. Pagal kode pateiktus komentarus matome, kur buvo panaudoti buferiai ir atributai, taip pat inicializuotos abi šešėliuoklės - viršūnių bei fragmentų, bei joms paduodami duomenys atskirose masyvuose. Taip pat šis autoriaus kodas yra puikus įrodymas, kad WebGL yra tik rastravimo mechanizmas, o visus sprendimus dėl objekto dydžio, spalvų, elgesio, drobės, įterpto teksto, duomenų tipo ir perdavimo į *GPU* būdo, priima programuotojas. To pasekoje, paprastam judančiam stačiakampiui gretasieniui su perdengtu *Tex* standarto tekstu atvaizduoti prireikė trijų skirtingų formatų dokumentų (.js, .html ir .css) ir, bendroj sumoj, daugiau kaip trijų tūkstančių kodo eilučių.

Atidarius naršyklės lange viso šio kodo rezultatą, matome tokį vaizdą:



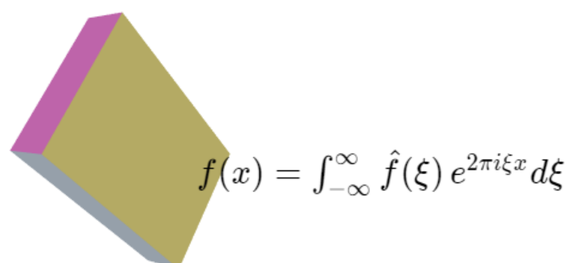
4 pav. *Tex* išraiška susieta su 3D objektu

Čia įterpta žymė perdengia (angl. *overlay*) drobę, bet yra pritaikyta jos dydžiu bei riboms. Formulė pozicionuojama atitinkamai vienai stačiakampio gretasienio viršūnei, juda kartu su ja ir taip sudaro įspūdį, kad yra 3D scenos dalis. Tai pilnai išsprendžia problemą, kuomet reprezentacijai naudojami keli trimačio objekto rakursai ir kiekvieną kartą pasukus trimatį objektą nebereikia iš naujo „klijuoti“ žymės į atitinkamą vietą:



5 pav. Formulė prie 3D objekto kitokiu rakursu

Kuriant tokio stačiakampio gretasienio vizualizacijas (paveikslėlius, nadojamus skaidrėse, publikacijose ir kt.), užtenka tiesiog ekrano nuotraukos (angl. *screen-shot*). Jų galima pasidaryti kiek tik reikia, o pastebėjus klaidą, nesunkiai modifikuoti. Prezentuojant vizualizacijas matome, kad nei formulė, nei pats stačiakampis gretasienis nepasikeitė. Pakito tik taško su kuriuo susieta *Tex* išraiška, bei atitinkamai visų kitų viršūnių, pozicija objekto erdvėje. Tad galime pasirinkti vis kitokią žiūros perspektyvą ir demonstruoti objektą kartu su žyme iš įvairiausių, prezentacijos turiniui reikiamų pusių:



6 pav. Kitokia objekto ir formulės pozicijos perspektyva

Visose perspektyvose 3D scena atrodo tvarkinga, išbaigta ir vientisa. Nepaisant reprezentatyvaus vaizdo, formulė nėra integruota į 3D sceną kaip objektas ir nėra, taip kaip stačiakampis gretasienis, rastruojama su WebGL. Žymė, yra tik vienas iš HTML elementų, perdengiantis kitus elementus ir susietas su vieno taško (gretasienio viršūnės) pozicionavimu.

Gavus tokius rezultatus, galima konstatuoti, kad šis praktinis eksperimentas pavyko. Pačią žymę nebuvo sudėtinga įdėti, o *Tex* šrifto atvaizdavimu (angl. *render*), pasirūpino *KaTex* biblioteka. Didžiausią dalį kodo užėmė paties trimačio objekto aprašymas bei žymės su objekto tašku „surišimas“. Jeigu trimatis objektas jau yra sukurtas ir tereikia įterpti žymę, kad paruošti jo atvaizdą prezentaciniam turiniui, šis žymės uždėjimo būdas būtų pats greičiausias ir paprasčiausias. Žymės „surišimas“ su objekto tašku gali būti ir ne toks sudėtingas, jeigu objektui kurti buvo naudojamas

ne tik grynas WebGL, o kokia nors papildoma WebGL biblioteka, palengvinanti darbą ir atliekanti skaičiavimus už programuotoją (pavyzdžiui ThreeJS [4]).

3.3. Kas yra Asymptote

D.Knuth'o darbai (*Tex* ir *MetaFont*) [1.1] įkvėpė John'ą D.Hobby žengti dar vieną žingsnį į priekį ir sukurti *MetaPost*. Tai grafinė programavimo kalba, nuo 2016 metų integruota į *LuaTex* [1.2] mechanizmą, kurios dėka savo publikacijos turinyje autorius gali braižyti vektorinės grafikos diagramas naudojantis algebriniais ar geometriniais aprašymais. Maža to, *MetaPost* suteikė galimybę į nubrėžtas diagramas įterpti žymes, kurių turinys gali būti bet koks tekstas atvaizduojamas su *Tex*.

Kanados miesto Alberta universiteto studentai, 2002 metais nusprendę visiškai perrašyti grafinį *MetaPost* mechanizmą, žengė dar toliau. Jau 2008 metais, suimportavus į *LaTex* papildomą makrokomandų paketą, galima buvo sudaryti ne tik plokščias algebrines ir geometrines diagramas, bet ir trimates. Tai, kas prasidėjo kaip tiriamasis studentų darbas, peraugo į galingą aprašomąją vektorinės grafikos programavimo kalbą, skirtą kurti *Tex* kokybės standartą palaikantiems techniniams brėžiniams ir iš dvimačių plokščių objektų „pagaminti" trimačius.

Žymėms, kurios yra svarbi bet kokios matematinės diagramos, grafiko ar brėžinio dalis, šioje kalboje skirtas išskirtinis dėmesys. Čia *label* yra vienas iš keturių kalbos primityvių „tipų" (primityvų) ir tuo pačiu viena pagrindinių funkcijų, kuriai padavus atitinkamus argumentus, sukuriama žymė (angl. *label*). Šios žymės visiškai valdomos vartotojo, jas gana nesudėtinga įterpti ir, žinoma, jų turinys gali būti bet koks *Tex* standarto tekstas, įskaitant (bet neapsiribojant) matematines išraiškas ir sudėtingas formules. Šios kokybiškos ir paprastos naudoti žymės, kartu su trimatėmis diagramomis, „persikėlė" ir į 3D erdvę. Taigi, su *Asymptote* (taip buvo pavadinta programavimo kalba) galima sugeneruoti aukštos kokybės trimatę vektorinę grafiką ne tik *Tex* vartotojams įprastais *PostScript* ar PDF formatais, bet taip pat ir WebGL[13].

3.4. Kaip veikia Asymptote

Pagrindinė mintis kuriant *Asymptote* konceptą buvo palikti žymių teksto tipografikos rūpesčius *LaTex* makrokomandoms. Tai jau įgyvendino J.D.Hobby'io *MetaPost* naudojami algoritmai. Tačiau *Asymptote*, neapsiribojanti dvimate erdve bei sekanti su naujausiomis technologijomis, kurių *MetaPost* kūrimo laikais dar nebuvo, šiuos algoritmus gerokai patobulino. Vienas iš reikšmingiausių patobulinimų - įterpti papildomą *PostScript* sluoksnį su panaudojus *\includegraphics* makrokomandą. Tuomet rezultatas iškarto atvaizduojamas .ps (o vėliau .pdf) formatu, tuo tarpu *MetaPost* atvaizdavo diagramas tik DVI formato failuose, kuriuos reikėdavo konvertuoti į PDF vėlesniame etape. Turinti papildomą *PostScript* sluoksnį, *Asymptote* komunikuoja su atvaizdavimo mechanizmu (angl. *renderer*) dvejomis „gijomis" tame pačiame kanale. Iš karto perduodami dviejų tipų duomenys: žymės dydžio ir turinio informacija, skirta *Tex*, bei žymės atvaizdavimo informacija, skirta *PostScript*. Tokiu būdu suformuotos žymės transformacijos (pavyzdžiui „iškirpimas", pozicionavimas diagramoje ir pan.) atliekami jau per *PostScript* funkcionalumą (nebe per *Tex* makrokomandas) [14].

Kitas svarbus su *Asymptote* atsiradęs pokytis yra 64 bitų skaitinės reikšmės. Toks dydis apsaugo ne tik nuo nepakankamo atminties kiekio sveikiesiems skaičiams, bet ir suteikia pakankamą tikslumą (užtektinai vietos skaičiams po kablelio) trupmeninėms reikšmėms. Padidinus skaitinių reikšmių dydį iki 64 bitų, teko tobulinti ir J.D.Hobby'io *MetaPost* Bezjė kreives „piešiančius" algoritmus, kad jie galėtų susidoroti su smarkiai padidėjusiais duomenų kiekiais. *Asymptote* kitaip nei

MetaPost dalina objekto paviršius į jungius plotus bei kitaip skaičiuoja liestinių ir sankirtos taškų koordinates (kontrolinius taškus). Taip pat algoritmus reikėjo pritaikyti ir 3D objektams sukurti. *Asymptote* kūrėjai teigia [14] pritaikę Simpsono taisyklę ir Bezjė kreives pavertė į kubinius polinomus. Pagal jų išvadas, tai veikia ne tik efektyviau nei įprastas Bezjė padalinimas, bet ir leidžia suvaldyti netaisyklingą funkcijų (pavyzdžiui tokios kaip $xsin(1/x)$) elgesį, jam tik prasidedant.

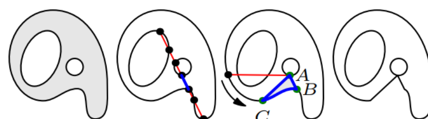
Asymptote trimačiams objektams ir žymėms (pastarosios irgi yra trimačiai objektai 3D scenoje) kurti pritaikė atskirą modulį *three*, kuris „įjungia" *Asymptote* vektorinės grafikos atvaizdavimo mechanizmą (angl. *vector graphics rendering engine*), suprantantį OpenGL kalbą. Norint iš dvimačio objekto (pavyzdžiui *Tex* spaudos ženklo) padaryti trimatę žymę, tam skirtos *Asymptote* kalbos funkcijos rašomos .asy formato faile, kuris yra kompiliuojamas. Kompiliavimo metu duomenys paverčiami daugianariais, kurie perduodami sugeneruotame JavaScript ir HTML tarpiniame programiniame išeities kode. Šio kodo reikšmes „pasiima" statinis JavaScript failas, *Asymptote* kūrėjų sukurta AsyGL biblioteka, ir suformuoja atitinkamas šešeliuokles. Tuomet šešėliuoklės perduodamos vektorinės grafikos atvaizdavimo mechanizmui, o pastarasis atvaizduoja 3D sceną WebGL ar .prc (angl. *Product Representation Compact*) formatu.

Anot kūrėjų [15], unikalus yra *Asymptote* kompiliavimo metu vykdomas algoritmas, „paverčiantis" plokščias dvimates formas (tai gali būtį įvairių funkcijų grafikai, geometrinės figūros ar *Tax* spaudos ženklai) trimatėmis. Algoritmo užduotis turint dvimačio objekto duomenis (Bezjė kreives), sugeneruoti atitinkamo trimačio objekto vektorinį aprašymą (kubinius Bezjė paviršius). Būtent vektorinė objekto išraiška yra svarbi kuomet reikia užtikrinti publikuojamo turinio kokybę, nes, kaip nutiko D.Knuth'ui 1977-aisiais [1.1], niekada nėra žinoma kokie yra įrenginių, kurie atvaizduos/spausdins turinį, pajėgumai ir savybės. Taigi, *Asymptote* algoritmui svarbiausia yra taip aprašyti objekto paviršių (angl. *surface*), kad jis būtų atvaizduojamas vienodai kokybiškai bet kokio tipo įrenginiuose.

Daugianariai (angl. *polynomial*). Paviršiaus aprašymui vietoj netolygių racionalių B-splainų (angl. *NURBS, Non-uniform rational B-spline*), parametrizuojami specialūs daugianariai (polinomai). Kaip ir anksčiau aprašytame [1.3.1] algoritme vektoriniam šriftui atvaizduoti, netolygių racionaliųjų B-splainų parametrizavimo atsisakoma. Jie netinka ne tik kokybiškam dvimačio spaudos ženklo kreivių atvaizdavimui, bet ir trimačiams paviršiams aprašyti, kadangi jie nesikeičia projekcijos perspektyvose. O norint spaudos ženklą ne tik kokybiškai atvaizduoti, bet ir „pakelti" ir padaryti trimačiu, reikalingi kintantys projekcijos perspektyvoje, afininiai (pritaikius afininę transformaciją Bezjė kreivei, gauta kreivė irgi yra Bezjė kreivė) Bezjė splainai [12]. J.D.Hobby *MetaPost* aprašė du (.ps ir .pdf formatu) Bezjė kreives „piešiančius" algoritmus: krypties (angl. *Hobby's 2D Direction Algorithm*) ir kontrolinių taškų (angl.*Hobby's 2D Control Point Algorithm*) [14]. *Asymptote* kūrėjai šiuos algoritmus generalizavo (angl. *three-dimensional generalization of Hobby's algorithms*) [12] ir suskaidė atvaizduojamų 3D objektų kreives į kubinio polinomo segmentus (keturi kontroliniai taškai) (angl. *cubic polynomial segments*), o paviršius į tenzorinės sandaugos skiautes su 16 kontrolinių taškų, straipsnyje jie vadinami *Bezjė paviršiais* (angl. *Bezier surface*) [29].

Bezulate algoritmas Norint gauti šiuos neišsikreipiančius Bezjė paviršius (angl. *non-degenerate Bezier patches*) [29], reikia, plokštumoje esantį (pavyzdžiui vieno spaudos ženklo) Bezjė kreivės apsuptą plotą padalinti į mažesnius, paprastai jungius plotus. Vykstant padalinimui visuomet patikrinama ar dalinamas plotas yra kreivės viduje ar išorėje, o tuomet vyksta jo suskirstymas į segmentus. Segmentą formuojantys taškai turi būti sujungti tarpusavyje ir išdėstyti toje pačioje

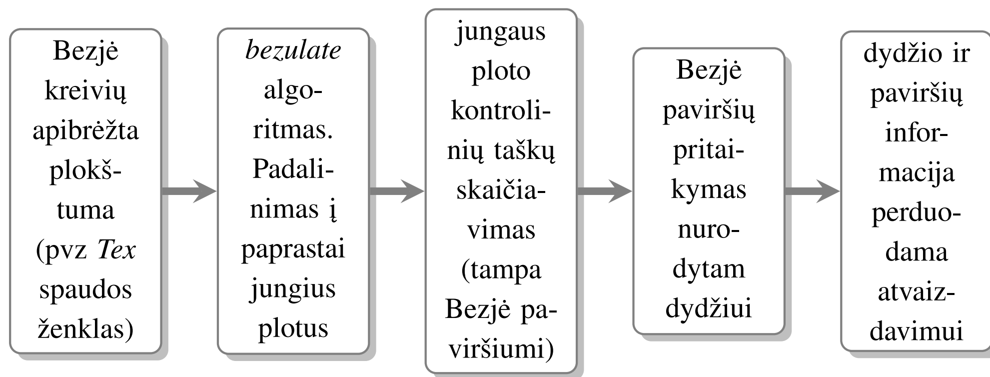
kreivėje. Jeigu vidurio taškas (su kuriuo mėginami jungti kiti taškai) nepriklauso kreivei arba, jeigu jungianti taškus linija kerta kreivę daugiau negu du kartus, toks taškas nėra jungiamas į segmentą. Atradus visus tinkamus taškus, jie apjungiami ir pastarasis plotelis atskiriamas nuo likusių kreivės taškų. Likusius taškus algoritmas rekursyviai vėl mėgina sujungti į kitus segmentus. Jeigu iškarto tokių paprastai jungių plotų nepavyksta aptikti (pavyzdžiui spaudos ženklas turi „skylių“), vyksta atskirų spaudos ženklo dalių skaidymas. Algoritmas tuomet ieško atkarpos, kurios taškai sujungia vidinę kreivę su išorine. Jeigu tokia yra, ji tampa formuojamo ploto kraštine, pavyzdžiui AB. Tuomet algoritmui reikia rasti tokį tašką C, kad kraštinė AC nesiliestų su išorine kreive daugiau kaip vieną kartą, o su vidine kreive liestųsi tik taške A. Radus tokį tašką formuojamas plotas ABC, kurio viduje jau nebėra jokios vidinės kreivės (jis visas užpildytas). Kai tik toks plotas atsiranda, jis „išimamas“, o likusios išorinės kreivės apjungiamos. Apjungtame plote algoritmas vėl ieško ABC ploto pagal nustatytus reikalavimus. Šiems surastiems paprastai jungiems ploteliams atliekami tenzoriniai skaičiavimai ir rezultatai aprašomi 16 kontrolinių taškų (koordinatės erdvėje) ir keturiais, spalvą nusakančiais, taškais. Atlikus skaičiavimus šios skiautės yra laikomos Bezjė paviršiais ir jos yra mažiausias *Asymptote* trimačio objekto paviršiaus ploto vienetas. Jo koordinatės perduodamos šešeliuoklėms atvaizdavimui [29].



7 pav. Rastas paprastai jungus plotas, kuriam bus atliktas koordinačių (16 taškų) skaičiavimas. Iliustracija perpiešta iš [15]

Suveikus „bezuliacijai“, vykdomas ir optimalaus 3D objekto dydžio nustatymo algoritmas [29], Bezjė paviršiai turi tilpti į nustatytą „dėžutės“, kurioje bus atvaizduojami, dydį. Dydžio informacija ateina į mechanizmą tuo pat metu, tad tokio dydžio erdvėje ir atidedami Bezjė paviršių koordinačių taškai. Mažinant ar didinant plotą, kartu mažėja ar didėja visas objektas. Čia pat, jeigu reikia, paskaičiuojamas ir centrinis taškas, apie kurį nuolat „suksis“ žymė, jeigu norima, kad žymė būtų *Billboard* principo [3.4.1]. Taip, gaunamas vektorinis rezultatas, kuris žymiai kompaktiškesnis už daugybę, 3D objektui nupiešti reikalingų trikampių viršūnių. Šie, palyginti nedidelies apimties geometriniai duomenys (16 kontrolinių taškų kiekvienam kubiniam Bezjė paviršiui ir 4 paviršiaus spalvai aprašyti naudojami taškai) perduodami šešėliuoklėms, ir atvaizduojamas labai tikslus (ar beveik labai tikslus) trimatis paviršius [12].

Toks išskirtinis procesas, paverčiantis dvimatį *Tex* šriftą vektorinės grafikos kokybės trimačiu objektu yra viena iš unikalių *Asymptote* funkcijų.



8 pav. Asymptote veikimo schema

Šiai dienai nėra kito būdo sukurti vektorinės grafikos kokybės žymės 3D scenoje. Atlikus praktinį eksperimentą, su *Asymptote* pavyko sukurti įvairiaus tipo, visas, šiuo metu galimas, trimačio objekto *Asymptote* žymes.

3.4.1. Trimatė TEX kokybės žymė (kaip scenos objektas) 3D scenoje

Nagrinėjant šios praktikos rezultatus, labiausiai pastebima atvaizduotos žymės kokybė. Kaip ir bet kokį vektorinės grafikos rezultatą, net ir labai išdidinus (priartinus), spaudos ženklo kontūras lieka tolygus ir (lyginant su HTML elemento žyme) beveik nekampuotas. Taip pat pasijuto nesudėtingas objekto su žyme įterpimas į su *LaTex* kuriamą prezentacinį turinį - nebereikėjo daryti turimos 3D scenos ekrano nuotraukos.

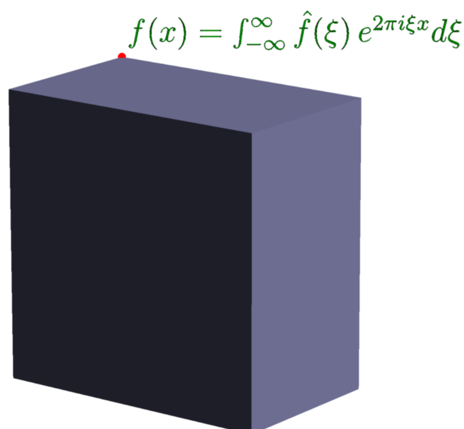
Paprasčiausia žymė *Tex* išraiška susieta su vienu trimačio objekto tašku. Tai vienas paprasčiausių ir aiškiausių variantų kaip įsivaizduojame savo objekto žymes. Jeigu 3D scenoje yra taškas bei poreikis prie jo užrašyti reikiamą tekstą, ši paprasčiausia žymė lengviausiai įgyvendinamas sprendimas. Neskaitant gretasienio piešimo kodo, bei parankinio kodo (tokio kaip *import*) eilučių, paprasčiausios žymės „uždėjimas“ užima vos dvi kodo eilutes. Vienoje sukuriama *string* tipo eilutė, kurios reikšmė tarp kabučių atitinka *LaTex* komandose naudojamą matematinės išraiškos užrašymą:

```
string txt=
"$f(x) = \int_{-\infty}^\infty \hat f(\xi)\,e^{2 \pi i \xi x} d\xi$";
```

Kitoje eilutėje kviečiama *Asymptote* kalba parašyta *label* funkcija, kuriai kaip argumentai paduodami sukurta *string* eilutė, taškas, su kuriuo reikia susieti žymę (aprašytas kuriant trimatį objektą), bei nurodymai kaip atvaizduoti žymę. Be spalvos pasirinkimo, čia dar yra teksto išdėstymo galimybė. Jos yra dvi: taip vadinamas *Billboard*, kuomet žymės tekstas nuolat atvaizduojamas iš kairės į dešinę ir neapsiverčia aukštyn kojomis nepriklausomai nuo to, kaip pasukamas objektas, bei *Embedded* stilius, kuomet žymė apsiverčia (aukštyn kojomis ar veidrodiniu principu) pagal tai, kaip paverčiama trimatė forma. Siekiant gauti ne prastesnį rezultatą, nei buvo gautas naudojant HTML [3.2.1], buvo pasirinktas *Billboard* variantas, kuomet žymės tekstas nuolat atvaizduojamas korektiškai:

```
label(txt, top, deepgreen, align=NE, Billboard);
```

Šių dviejų eilučių pakanka *Asymptote* kalba perduoti instrukcijas mechanizmui, kuris atvaizduoja visiškai kokybišką *Tex* standartą atitinkančią sudėtingą matematinę išraišką ir supozicionuoja ją norimoje 3D scenos vietoje:

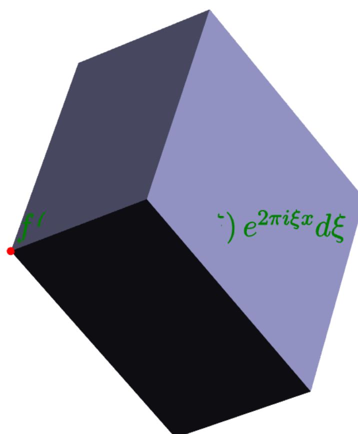


9 pav. Paprasčiausia 3D objekto *Tex* šrifto žymė

Rašant instrukcijas tiesiogiai WebGL, tokiam rezultatui pasiekti prireikė daugiau nei trijų tūkstančių kodo eilučių [3.2.1]. O norint įterpti rezultatą į turimą, su *LaTex* kuriamą, tipografinį turinį, teko daryti ekrano nuotraukas. Su *Asymptote* pakako įprastų makrokomandų *\begin{asy}* ir *\end{asy}* [A]. Prieš tai reikėjo importuoti *asy* paketą ir sukurti vykdomąjį *.asy* formato failą su tam tikrais nustatymais (jo dėka kompiliuojamas *Asymptote* kodas ir rezultatas atvaizduojamas pasirinktu formatu).

Taigi, naudojant *Asymptote* paprasčiausios žymės sukūrimas, susiejimas su trimačiu objektu bei gauto rezultato įterpimas į prezentacinį turinį tapo nesudėtinga procedūra. Tačiau prieinamas paprastas 3D scenų žymėjimas (angl. *labeling*) atskleidė keletą naujų problemų, su kuriomis nesusiduriama dėliojant žymes dvimačiuose brėžiniuose ar diagramose.

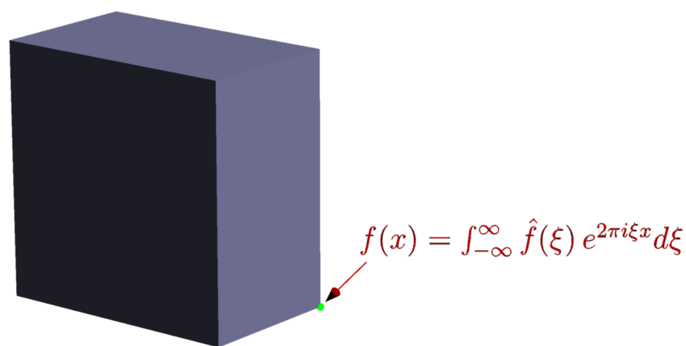
Tiek žymė, tiek trimačio objekto paviršius, turi savo plokštumas 3D scenoje. Kai kuriose perspektyvose šios plokštumos susikerta tarpusavyje. Kitaip sakant, kai kuriomis kryptimis pasukus gretasienį, pažymėtas taškas lieka „kitoje pusėje“, kartais jo nebesimato. Tuomet nesimato (arba matosi tik dalis) ir žymės. Persidengimo atvejų (kuri plokštuma būtų „priekyje“) kol kas nėra galimybės kontroliuoti. Tad kai kuriose perspektyvose žymė visiškai praranda savo informatyvumą:



10 pav. Valdyti plokštumų persidengimo nėra galimybės

Sprendžiant tokią problemą, reikalinga papildoma žymės savybė, nurodanti jos plokštumos poziciją kitų plokštumų atžvilgiu. T.y. įvykus persidengimui (angl. *overlap*), žymės pozicija galėtų būti *always on top* (liet. *visuomet virš kitų*). Šią savybę galima būtų įjungti arba išjungti (angl. *toggle*), priklausomai nuo konkretaus atvejo. Tokiu būdu autoriui atsirastų galimybė kontroliuoti savo žymės informatyvumą.

Žymė rodyklė Tai kitokio pobūdžio žymė, iš dalies sprendžianti čia iliustruojamą plokštumų persidengimo problemą. Nuo paprasčiausios žymės ji skiriasi tuo, kad yra siejama ne su trimačio objekto tašku, o su rodykle, kuri rodo (išlaikydama tam tikrus atstumus) į reikiamą objekto tašką:

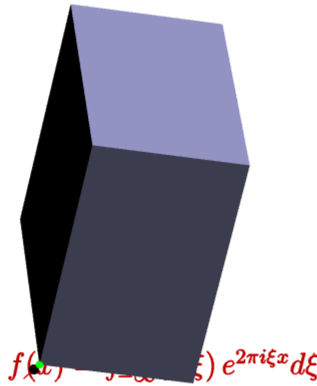


11 pav. *Tex* šrifto žymė rodyklė

Rodyklė irgi yra trimatė. Tokios pat rodyklės naudojamos ir koordinačių ašims atvaizduoti. Kadangi brėžiant koordinačių ašis žymės prie jų yra dažnai reikalingos, logiška buvo aprašant funkciją *arrow* suteikti jai *string* tipo parametrą, kurio argumentas bus panaudotas žymei šalia rodyklės atvaizduoti. Kiti galimi argumentai yra rodyklės kryptis, ilgis, spalva ir taškas, su kuriuo ji yra siejama. Iš esmės, instrukcija tokiai žymei sukurti (jau turint žymės teksto *string* eilutę) *Asymptote* kalba užrašoma viena eilute:

```
arrow(txt, right, Y+Z, 1cm, heavyred);
```

Nors dėka rodyklės padidinus atstumą tarp žymės ir trimačio objekto plokštumos ne taip lengva atrasti perspektyvą, kurioje šios plokštumo susikirstų, tačiau aukščiau aprašyta problema, vis tik, iki galo nėra išspręsta. Galima sakyti, ji išlieka dėl to, kad rodyklė irgi yra trimatis objektas ir jai netaikomas anksčiau paminėtas *Billboard* principas, kuomet ji atvaizduojama nuolat išlaikant savo formą. Gretasienį dabar galima pasukti taip, kad rodyklė nebeatrodys kaip rodyklė (o pavyzdžiui bus tik apskritimas) ir tuomet persidengs ne tik stačiakampio paviršiaus ir formulės plokštumos, bet taip pat ir rodyklės paviršiaus, kad ir labai nedidelio, plokštuma. Tokioje perspektyvoje intuityviai tikimasi, kad rodyklė su visa žyme pasislinks į šoną ir toliau priekiniu galu rodys į žymimą tašką, o kitame gale „laikys“ aiškiai matomą, neapverstą, nepersidengusią su gretasieniu formulę. Tačiau taip neįvyksta:



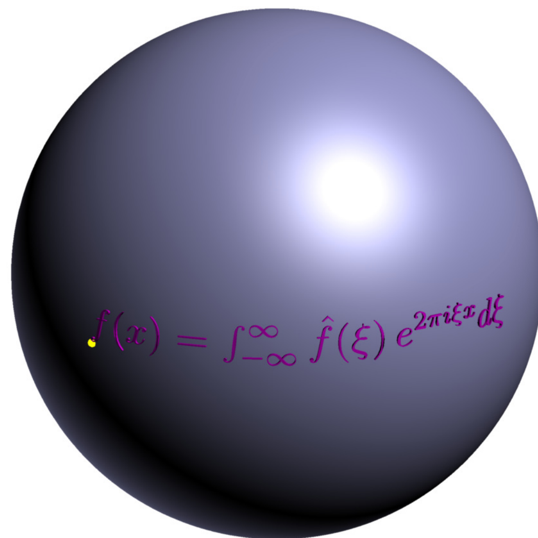
12 pav. Trimatei rodyklei netaikomas *Billboard* principas

Akivaizdu, kad tokioje perspektyvoje žymė rodyklė tampa labiau klaidinanti nei informatyvi. Sprendžiant šią problemą, reikėtų papildyti rodyklės atvaizdavimo algoritmą centro tašku [3.4], kurį turi *Billboard* tipo žymė. Esminis skirtumas tas, kad toks taškas išlaikytų rodyklės priekinių trijų (*snapelio*) ir galinio taško (*uodegos*) pozicionavimo vientisumą. Kitaip sakant atstumas tarp jų jokiose perspektyvose „nemažėtų“, o būtų išlaikomas toks pat. Tuomet rodyklė būtų (jeigu reikia) dar viena *Billboard* tipo žymė, skirta atitolinti paprastos žymės tekstą nuo žymimo trimačio objekto taško. Tokiu būdu bet kokioje pozicijoje jos atrodytų kaip rodyklės, o jų išdėstymas ekrane padėtų suvaldyti plokštumų paviršių persidengimus. Įgyvendinus aukščiau aprašytą *always on top* principą, jį galima būtų pritaikyti ir rodyklei, tais atvejais, kuomet rodyklė persidengs su trimačio objekto paviršiaus plokštuma. Dabar esančios rodyklės, toliau liktų 3D grafikų koordinačių ašims braižyti, tačiau žymėms esančioms šalia šių ašių išlieka ta pati persidengimo problema. Kol kas žymės, esančios prie trimačių rodyklių automatiškai „nepasislenka“ į tokias pozicijas, kad aiškiai matytųsi, nepersidengtų ir liktų informatyvios bet kokioje perspektyvoje.

Tačiau, gal būt, ne visada yra poreikis, nuolat matyti žymę? Jeigu trimatis objektas pasukamas tokiu kampu, kad pažymėtas taškas nebėra matymo perspektyvoje, kodėl jo žymė vis dar turi būti matoma? Jeigu norima vadovautis tokia logika, tuomet tiktų kitokio pobūdžio žymė.

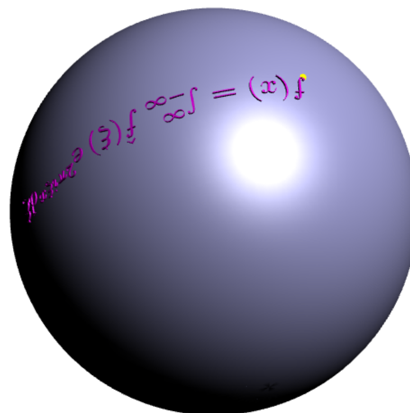
Žymė ant paviršiaus Ši žymė susieta ne tik su tašku, bet taip pat ir su trimačio objekto paviršiumi (angl. *surface*). Jai nebegalioja aukščiau aprašytas *Billboard* principas. Tokia žymė pasisuka (apsiverčia, „pradingsta“, „nutolsta“) ir yra (arba nėra) matomumo zonoje lygiai tiek pat, kiek pasisuka ir yra matomumo zonoje trimačio objekto paviršiaus dalis, ant kurios ji supozicionuota.

Pozicionuoti žymę ant paviršiau galima dviem būdais. Pirmas, intuityvus variantas, kuomet žymės tekstas tiesiog „išpieštas“ ant paviršiaus. Kitaip sakant, trimatė formulė, tiesiog „guli“ ant objekto reljefo. Tokią žymę patogu turėti, nes čia visiškai nelieka plokštumų persidengimo problemos. Žymės plokštuma atitinka objekto paviršiaus plokštumos padėtį 3D scenoje, tad išvengiama tiek šių dviejų susikirtimo, tiek žymės su kitų žymių (jeigu žymė yra ne viena) plokštumomis persidengimo. Taip pat tokia žymė gana korektiškai dera prie trimačio objekto dinamiškumo. Ji „atsiranda“ ir „išnyksta“ iš matymo zonos lygiai taip, kaip intuityviai tikimasi:



13 pav. *Tex* šrifto žymė ant paviršiaus

Tačiau galima būtų ginčytis dėl žymės ant paviršiaus informatyvumo. Kuomet žymės tekstas neišlaiko *Billboard* principo ir gali būti apverstas ar apsuktas 180° kampu, didelė tikimybė, kad informacija, kurią ta žymė turėtų aiškiai pateikti, gali likti nesuprasta. Taip pat tokia žymė prideda suvokimą apie „taisyklingą“ trimačio objekto padėtį. Pavyzdžiui sfera, paprastai, neturi jokių suvokiamų padėčių, kaip ją bepasuksi 3D scenoje, sunku pasakyti, kur yra jos „viršus“ ar „apačia“. Tačiau ant sferos paviršiaus atsiradus žymei, kai kurios sferos pozicijos taps suvokiamos, kaip „netaisyklingos“. Pavyzdžiui atsiras perspektyva, kuriose sfera 3D scenoje bus suvokiama kaip „aukštyn kojomis“ (nes žymės tekstas bus apverstas), nepaisant to, kad pati sfera nei „galvos“, nei „kojų“ neturi, jas „turi“ tik žymės tekstas:

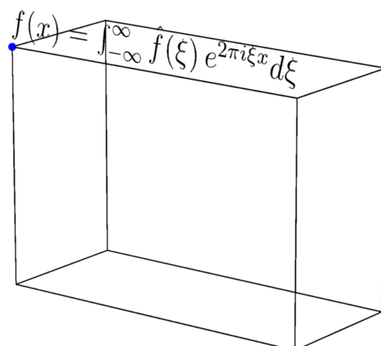


14 pav. Dėl žymės ant paviršiaus 3D objekto padėtis suvokiama kaip „aukštyn kojomis“

Taip pat reikia įvertinti, ar žymė yra siejama su vienu objekto tašku. Jeigu taip, būtina nepamiršti jo „nuspalvinti“, „nupiešti“ ar kaip kitaip pažymėti. Kitu atveju nebus aišku ką tiksliai šia žyme norima pažymėti. Jeigu reikiamas taškas nebūtų paryškintas, galėtų atrodyti, kad žymė skirta tiesiog visai sferai.

Kitas žymės ant paviršiaus variantas yra žymės teksto „pastatymas“ ant paviršiaus (o ne „išpiešimas“ jame). Žymės plokštuma ne uždedama ant paviršiaus plokštumos, o pastatoma kurios nors paviršiaus tiesės (arba kreivės) atžvilgiu 90° kampu. Tokios žymės privalumai ir trūkumai

iš esmės yra tokie patys kaip ir aukščiau aprašyto varianto, tik papildomai prisideda rūpestis dėl aplink esančių ir galimai persidengiančių kitų objektų (dažniausiai kitų, ne ant paviršiaus esančių, žymių). Tokio pobūdžio žymė ant paviršiaus labiau tiktų objektams, kurie turi tik kontūrus (nėra užpildyti), tačiau yra poreikis žymę išdėstyti atitinkamai jų briaunoms:



15 pav. Žymė ant paviršiaus. *Tex* išraiška susieta su 3D objekto briauna

Trimatė žymė dar vienas variantas, galimas įgyvendinti *Asymptote* kalba. Tai yra pačios formulės iškėlimas (angl. *extrude*) kaip atskiro 3D objekto. *Asymptote* kūrėjai skelbia, kad tai ko gero vienintelė kalba pakėlusi Tex standartą į trimatį lygmenį [15]. Dabar yra galimybė savo straipsnyje ar skaidrėse šalia trimačio objekto (arba be jo) bet kokį *Tex* spaudos ženklą padaryti trimačiu ir įterpti jį į tekstą įvairiais rakursais. Tokia žymė atkreips dėmesį, nes smarkiai išsiskirs ir bus ganėtinai užakcentuota:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\, e^{2\pi i \xi x} d\xi$$

16 pav. Trimatė žymė. *Tex* išraiška kaip 3D objektas

Trimatė žymė derėtų būti naudojama su trimatėmis rodyklėmis. Pavyzdžiui, jeigu yra trimatė koordinačių ašis ir tas ašis žyminčios „rodyklės“ yra trimatės, žymes prie tų ašių būtų logiška taip pat padaryti trimatėmis. Tokių žymių informatyvumas galėtų būti kvestionuojamas, nes vos pasukus trimatį grafiką kitokiu rakursu, žymės tekstas taptų ne itin įskaitomas ir/ar suprantamas. Šis variantas tiktų nesudėtingoms žymėms, pavyzdžiui X, Y, Z raidėms šalia koordinačių ašių. Kokia grafiko perspektyva bebūtų, intuityviai vistiek galima būtų suprasti kas yra pažymėta. Tačiau, esant komplikuotai matematinei išraiškai trimatės žymės poreikis gali būti abejotinas. Bet kokiu atveju, panašu, kad kol kas tai yra vienintelė galimybė *Tex* šriftą atvaizduoti trimatėje perspektyvoje, ir esant poreikiui tai padaryti, tenka pripažinti *Asymptote* kalbos unikalumą šiame aspekte.

4. Alternatyvi koncepcija TEX ir WebGL suderinamumui

Kaip parodė praktika [3.2.1] ir [3.4.1], žymių įterpimo būdas, tipas ir kokybė labai priklauso nuo *API* su kuriuo sukurta (ar norima sukurti) 3D scena. Kuo žemesnio lygio (angl. *low-level*) *API* naudojama objekto atvaizdavimui (pavyzdžiui naudojant vien tik WebGL), tuo sudėtingiau prie 3D objekto „prikabinti“ reikiamos kokybės ir norimo funkcionalumo žymes. Su WebGL net ir tokia, atrodytų, paprastai nesudėtinga procedūra kaip teksto atvaizdavimas, nėra lengva užduotis programuotojui [2.3]. Iš esmės nėra galimybių apsieiti be papildomų įrankių (bibliotekų ar aukštesnio lygio *API*). Tai taikytina ne tik teksto atvaizdavimui. Paprasčiausios formos (pavyzdžiui stačiakampio gretasienio) atvaizdavimui naudojant papildomą *API*, išeities kodo eilučių kiekis sumažėja nuo kelių šimtų iki vienos ar dviejų. Todėl nieko nuostabaus, kad daugelis programuotojų šiais laikais 3D scenoms kurti naudoja ne gryną WebGL.

Siekiant sumažinti laiko ir išteklių sąnaudas, bei taip pasiekti sudėtingesnių, įmantresnių, daugiau funkcionalumo turinčių rezultatų, trimačių objektų kūrimui su WebGL, dažniausiai naudojama ThreeJS biblioteka. Yra ir kitų alternatyvų, palengvinančių programavimą su WebGL - *PlayCanvas*, *Pixi* ar *Greensock* bibliotekos. Tačiau, kadangi ThreeJS yra labiau koncentruota į vieną tikslą (3D scenos) ir turi mažiau papildomų („pašalinių“) funkcijų [9], šiame darbe bus laikomasi principo, kad dauguma naudoja būtent šią biblioteką. Taip pat beveik visos teksto įterpimo technikos [2.3] siūlo naudotis šios bibliotekos teikiamomis galimybėmis, o kitas nurodo kaip alternatyvias.

ThreeJS yra JavaScript 3D biblioteka, panagrinėkime ką tai ištiesų reiškia. Istoriškai naršyklė visada buvo dvimatėje plokštumoje, juk ekranas yra plokščias. Pirmoji (ir labiausiai naudojama) *API*, kuri padėjo plokščiame ekrane išvysti „nupieštus“ objektus (figūras, spalvas ir pan.) buvo CSS. Be CSS internetiniai puslapiai būtų tiesiog paprasto teksto rinkiniai. Tuomet HTML atsirado *canvas* elementas, kuriame, kaip ir tapyboje naudojant drobę, tapo įmanoma „piešti“ ir atvaizduoti „piešinius“ ekrane. Vėliau atsirado SVG (*Scalable Vector Graphics*) formatas, suteikiantis galimybę aprašyti tokius primityvus kaip linijos (vektoriai) ir iš tų linijų formuoti („piešti“) objektus, pavyzdžiui logotipus, piktogramas ir pan. [8]. Naudojant šias *API* galima pakankamai nesunkiai „nupiešti“ stačiakampį ir atvaizduoti jį ekrane (interneto naršyklėje), tačiau nei viena šių technologijų nebuvo sukurta tam, kad ekrane galima būtų atvaizduoti stačiakampį gretasienį, t. y. trimatį objektą. Tuo tarpu ThreeJS, gali visus šiuos (canvas, svg ir css) ekrane atvaizduoti 3D scenoje. Tiksliau sakant, perskaičiuoja taškų koordinates (kokie taškai ir kur ekrane jie turi būti atvaizduoti) ir perduoda skaičius WebGL mechanizmui. WebGL „gavęs“ JavaScript kalba atliktus paskaičiavimus, „išverčia“ juos į šešėliuoklių kalbą. Šešėliuoklės yra pakankamai bendrinės ir iš esmės nieko nežino apie 2D ar 3D grafiką [8], jos vienodai tiksliai gali perduoti paskaičiavimus tiek skirtus atvaizduoti stačiakampį gretasienį, tiek skirtus sumažinti ekrane rodomo video ryškumui. Tokiems skaičiavimams abstrahuoti ir skirtas ThreeJS.

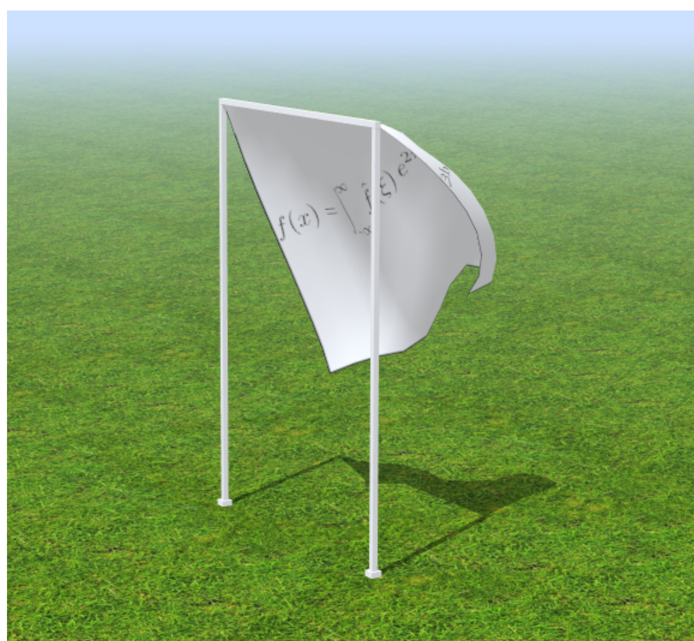
Taigi, ThreeJS pagrindinis tikslas apskaičiuoti duomenis, reikalingus teseliacijai (poligono padalinimas į trikampius) vykdyti. Bet kokia kreivė, pagal užduotus parametrus, kurie kontroliuoja trikampių kiekį tam tikrame plote, bus paversta trikampių viršūnių rinkiniu ir šių viršūnių koordinatės perduotos šešėliuoklėms. Padalinimo į trikampius pats programuotojas valdyti nebegali. Jis gali tik rinktis iš įvairių esamų, jau sukurtų, funkcijų, kuri labiausiai atitinka tikėtiną rezultatą ir jomis manipuliuoti. Bet kada, žinoma, galima sukurti papildomo funkcionalumo įskiepį (angl. *plug-in*), jeigu dabartinis ThreeJS funkcionalumas nėra pakankamas.

Iš esmės, pripratus naudotis ir gerai įvaldžius ThreeJS, trimatės scenos kuriamos naudojant

būtent šią biblioteką, tad intuityviai norėtųsi į tokias scenas įterpti *Tex* kokybės žymes būtent su šia *API* (ne per HTML [3.2.1] ir ne su visiškai kitu įrankiu *Asymptote*). Tačiau, jeigu norima žymių, kurios būtų 3D scenos objektais, ir kad jų kokybė būtų kuo artimesnė vektorinei grafikai (*Tex* standartas yra kokybės standartas). Taigi, iškyla klausimas, ką daryti, jeigu su ThreeJS jau yra sukurta sudėtinga įmantri 3D scena, ir norima į ją įterpti tokios kokybės ir funkcionalumo žymes, kurias siūlo *Asymptote*?

4.1. Esamų technologijų pernaudojimas

Norint sukurti žymę su ta pačia Furjė transformacijos formule naudojant ThreeJS, iš visų anksčiau aprašytų technikų [2.3] pavyko viena - naudojant tekstūrą. Dėl to, kad tekstūrą galima nesunkiai supozicionuoti norimoje scenos vietoje ant bet kokio paviršiaus ir, kad į ją galima „sutalpinti“ kokį tik norisi tekstą, dirbti buvo labai patogu ir greita (palyginus su darbu be ThreeJS). Panaudojus savybę *alphaMap*, galima tekstūros foną padaryti „permatomą“, tad matysis tik tekstūroje esantis paveikslėlis, pavyzdžiui su *LaTex* sukurtos matematinės formulės atvaizdas:



17 pav. *Tex* matematinė išraiška ThreeJS tekstūroje

Atrodytų tikslas pasiektas - tekstūros pagalba, galime įterpti *Tex* standarto formulės atvaizdą į žymiai įmantresnę nei gretasienis, ThreeJS 3D sceną. Tačiau aptarkime šios formulės tekstūroje kokybę. *Tex* šriftas šiuo metu yra laikomas vienu aukščiausių tipografijos standartų dėl savo spaudos ženklų grafikos išskirtinumo, išliekančio nepriklausomai nuo įrenginio [1.1]. *Asymptote* kalbos esmė buvo neprarasti šios kokybės, tad visi čia veikiantys algoritmai ir atvaizdavimo mechanizmas šį tikslą įgyvendina [3.3]. Tuo tarpu perdavus ThreeJS tekstūrai kurti, kad ir labai kokybiškos formulės atvaizdą (rastrinį paveikslėlį), paskui dar padarius sukurtos scenos ekrano nuotrauką ir įterpus ją (vėl gi kaip rastrinį paveikslėlį) į su *LaTex* sukurtą turinį, žymės kokybė ne tik kad neprilygs *Asymptote* žymėms, bet ir iš esmės smarkiai nusileis likusio turinio teksto kokybei, prie kurios pripratę*LaTex* naudotojai.

Žymės kokybė prastesnė nei vektorinės grafikos, tačiau 3D scena daug įmantresnė. Furjė transformacijos formulė prikabinta ne prie stačiakampio gretasienio, kad galima būtų vizualiai pademonstruoti skirtumą tarp 3D scenos kūrimo su ThreeJS ir su *Asymptote* bei atsakyti į klausimą,

kodėl negalima nuo šiol visus 3D objektus su žymėmis kurti tik su *Asymptote*. Šiai dienai jokia kita *API* negali sukurti tokios kokybės žymių, kurias galima sukruti su *Asymptote*. Tačiau pastaroji nėra suderinta nei su ThreeJS, nei su kokia nors kita, įspūdingą grafiką kuriančia biblioteka, tad norint sukurti tokią [17] 3D sceną su kokybiškomis vektorinės grafikos žymėmis, ją reikia „perpiešti", su pačia *Asymptote*. Tai būtų išties, labai sudėtinga, nes *Asymptote* neturi tiek daug technikų ir įrankių įmantrioms, ne algebrinių ar geometrinių skaičiavimų formoms kurti.

Gal būt, jeigu būtų prieinami šioje (17) scenoje esančių objektų Bezjė kreivėmis apribotų plotų duomenys, *Asymptote* kompiliatorius juos gavęs galėtų viską „perpiešti"? Atsakymo į šį klausimą nėra, nes ne tik nėra aišku iš kur gauti šiuos reikalingus duomenis, bet ir kaip reikėtų juos perduoti *Asymptote* mechanizmui, jeigu atsirastų jų šaltinis. Kol kas *Asymptote* neturi funkcionalumo, kuomet paduodamas vektorinės grafikos (ar bet koks kitoks) failas, o iš jame esančio 2D objekto „padaromas" 3D. Arba padavus kitur sukurto trimačio objekto duomenis, kad *Asymptote* jį atkartotų savo aplinkoje.

Įdomu tai, kad bandymą su *Asymptote* atkartoti jau sukurtą trimatį objektą pavyko surasti. Štai žymusis ThreeJS arbatinukas:

18 pav. Oficialiai prieinamas https://threejs.org/examples/webgl_geometry_teapot.html

O čia - įgyvendinta idėja jau sukurtą su ThreeJS objektą atvaizduoti su *Asymptote*. Pateiktame pavyzdyje nėra nurodyta iš kur paimti reikalingi duomenys, o reikšmes tiesiog suvestos rankiniu būdu (angl. *hard-coded*):

19 pav. Oficialiai prieinamas https://asymptote.sourceforge.io/gallery/3Dwebgl/teapot.html

Jeigu pažiūrėtume į vektorinės grafikos arbatinuko išeities kodą [B], pamatytume kiek daug koordinačių reikia paduoti *Asymptote* algoritmui, kad atvaizduotų ne *Tex* šriftą, ne funkcijos grafiką, ar formą, aprašomą geometrine ar matematine lygtimi, o tiesiog norimą, su ThreeJS jau sukurtą, trimatį objektą. Tampa aišku, kad perpiešti 17 paveiksle pavaizduotą objektą su *Asymptote* ne išeitis. Reikia ieškoti būdų perduoti kokybiškas *Asymptote* žymes ThreeJS bibliotekai, o ne atvirkščiai.

4.2. Trūkstama komponentė

Pagrindinė priežastis kodėl *Asymptote* ir ThreeJS šiai dienai „neranda bendros kalbos" yra skirtingas jų „piešimo" procesų principas. *Asymptote* yra vektorinės grafikos *API* ir jos atvaizduojami rezultatai būtent todėl ir yra tokios kokybės, kurią norisi turėti savo 3D scenoje. Iš kitos pusės, šios kokybės užtikrinimas (mažiausias *Asymptote* objekto paviršiaus vienetas yra kubinis Bezjė paviršius, aprašomas 16 kontrolinių taškų [3.4]) yra ir ta kliūtis, kurią sudėtinga įveikti norint kurti įmantrias ir neįprastas 3D scenas. Kaip parodė pavyzdys su arbatinuku, suskaičioti ir pateikti reikiamas objekto Bezjė paviršių reikšmes nėra aiškus, ar nors kiek automatizuotas, procesas. Tuo tarpu, ThreeJS turi daugybę priemonių įmantrioms 3D scenoms kurti ir norint tai daryti, net nebūtina labai išsamiai išmanyti skaičiuojamąją geometriją, todėl trimatės grafikos kūrėjai taip pripratę naudotis šia *API*. Tačiau, ThreeJS neturi vektorinės grafikos atvaizdavimo mechanizmo, tad kokio įmantrumo bebūtų scena, jos rezultatas visada yra trikampiai (ne Bezjė ar kokios kitokios kreivės). Tokio geometrinio atvaizdavimo kokybės pakanka trimačiams objektams, tačiau rastriniai teksto spaudos ženklai vis tik neprilygs savo kokybe vektoriniams. Taigi, norint turėti vektorinės grafikos kokybės žymes su ThreeJS sukurtoje 3D scenoje, reikia rasti būdą, priemonę ar techniką pritaikyti [1.3.1]) aprašytą (arba ne blogesnį) kreivių atvaizdavimo algoritmą. Kitaip sakant, reikia sukurti ThreeJS vektorinės grafikos atvaizdavimo mechanizmą.

Žinant, kad vektorinės grafikos atvaizdavimo mechanizmą turi *Asymptote*, pradžiai reikėtų pamėginti tuo pasinaudoti. Žinome [3.4], kad mažiausias *Asymptote* objekto paviršiaus vienetas yra kubinis Bezjė paviršius. Tai pat žinome, kad paviršių aprašantys 16 kontrolinių taškų perduodami šešėliuoklių generavimui, o pastarosios per specialią *AsyGL* biblioteką, kuri veikia kaip tarpinis „koridorius" (angl. *a port*), keliauja į *Asymptote* kūrėjų C++ kalba suprogramuotą, vektorinės grafikos atvaizdavimo mechanizmą (angl. *C++ vector graphics renderer*). Tuomet pikseliai išdėliojami nurodyto dydžio erdvėje ir rezultatas atvaizduojamas pasirinktinai .html, .pdf, .eps, .svg ar .png formato faile. Reiškia, iš esmės, turime kokybišką *Tex* šrifto žymę kaip atskirą 3D objektą, tereikia ją, neprarandant kokybės, perduoti į ThreeJS. Jau žinome, kad rastrinio paveikslėlio formatu (.png) perduota *Tex* žymė nėra tinkamos kokybės, o perdavus vektorinės grafikos failą .svg, ThreeJS nemokės atvaizduoti kreivių ir vistiek jį pavers rastriniu atvaizdu.

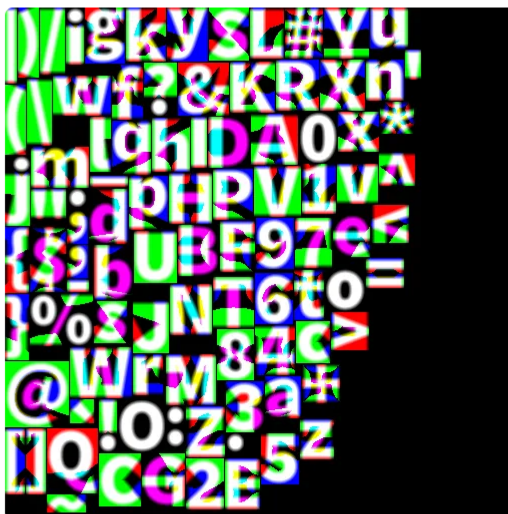
Tačiau dar žinome, kad ThreeJS moka tvarkyti ir atvaizduoti įvairius trikampių, į kuriuos padalintas trimatis objektas, „tinklelius" (angl. *mesh*). Vadovaujantis dokumentacija [6], kitais įrankiais sukurtus 3D modelius galima „įkrauti" (angl. *load*) į ThreeJS, pasinaudojant bibliotekoje jau esamais įvairiais „įkrovėjais" (angl. *loader*). Kaip teigia straipsnio [5] autorius, vienas populiariausių tinklelio įkrovimo formatų yra OBJ. Tai paprastas tekstinis failas, kurio viduje yra 3D geometrijos duomenys, t. y. viršūnių koordinatės, UV koordinatės, tekstūros koordinatės ir kitos skaitinės reikšmės, neturinčios jokių konkrečių vienetų (faile gali būti informacija apie objekto mastelį užrašoma žmogui suprantamu komentaru). Neretai .obj failas turi savo MTL (*Material Template Library*) formato failą, kuriame aprašomas OBJ faile saugomo objekto paviršiaus savybės (šešėliai, raštai, nelygumai ir pan.). Turint *Asymptote* žymės .obj (ir, jeigu reikia, .mtl) failą, jį galima būtų perduoti į ThreeJS per esančius „įkrovėjus". Tokiu būdu gavęs skaitines reikšmes, ThreeJS jau žinos kaip su jomis sugeneruoti šešėliuokles ir perduoti jas WebGL atvaizdavimui.

Sprendžiant šią užduotį, visų pirma reikėtų susisiekti su *Asymptote* kūrėjais ir aptarti galimybę atvaizduoti su *Asymptote* sukurtą rezultatą OBJ formatu. Kitas variantas - suprogramuoti įrankį, kuris konvertuotų WebGL formato duomenis į OBJ ir tokiu būdu perduoti kokybišką žymę ThreeJS bibliotekai. Tai būtų gana logiška, juk kiekviena *API* galėtų atlikti tą darbą, kuriam buvo sukurta - *Asymptote* „gamintų" aukščiausios kokybės *Tex* šrifto tekstines žymes, o ThreeJS jas

„įkomponuotų“ į savo įmantrias 3D scenas.

Bet gal būt galima pasiekti vektorinės grafikos žymės kokybę ir be *Asymptote* „įsikišimo“. Jau buvo aptarta [2.3], kad tekstas atvaizduotas per MSDF savo kokybe yra ne prastesnis už vektorinę grafiką. O straipsnio [19] autorius dar pateikia naudojimosi technikas, kurių dėka šriftas kokybiškai atvaizduojamas net ir su prastu trimatės grafikos atvaizdavimo procesoriumi (angl. *lowest-end 3D graphics hardware*). Šiai dienai sukurti *Tex* šrifto žymės (konkrečiau - matematinės formulės žymės) per MSDF nepavyksta, nes matematinėms išraiškoms naudojami nestandartiniai spaudos ženklai bei jų išdėstymas nebūtinai yra vienoje eilutėje vienas po kito išsirikiavę simboliai. Taigi, nėra sukurta tokio tekstūrinio atlaso, kuriame būtų aprašytos matematinių spaudos ženklų formos. Yra atskiri rinkiniai, pavyzdžiui graikiškos abecelės raidės, arba standartiniai aritmetiniai operatoriai, tačiau surinkti juos į vieną komplektą būtų labai problematiška.

Straipsnyje [36] yra aprašytas būdas kurti savo šrifto MSDF. Žinoma, matematinių formulių simbolius tikriausiai neįmanoma sutalpinti į vieną „abėcėlę“ aprėpiančią visus galimus prireikti spaudos ženklus. Bet, ko gero, ir nereikia daryti visų įmanomų spaudos ženklų tekstūrinio atlaso (kaip jau buvo minėta [2.3] MSDF aprašymai yra „sunkesni“ ir jiems reikia daugiau vietos atmintyje palyginus su SDF), kuriant matematinę žymę, būtų racionalu sudaryti tik būtent toje formulėje esančių spaudos ženklų MSDF tekstūrinį atlasą. Taigi, reikėtų suprogramuoti įrankį, kuriam padavus įvesties failą su reikiama matematine išraiška (gal būt .svg formatu arba *LaTex* turiniui įprastu .pdf formatu), būtų sugeneruojamas MSDF failas su atitinkamais spaudos ženklais ir atstumais aplink juos:

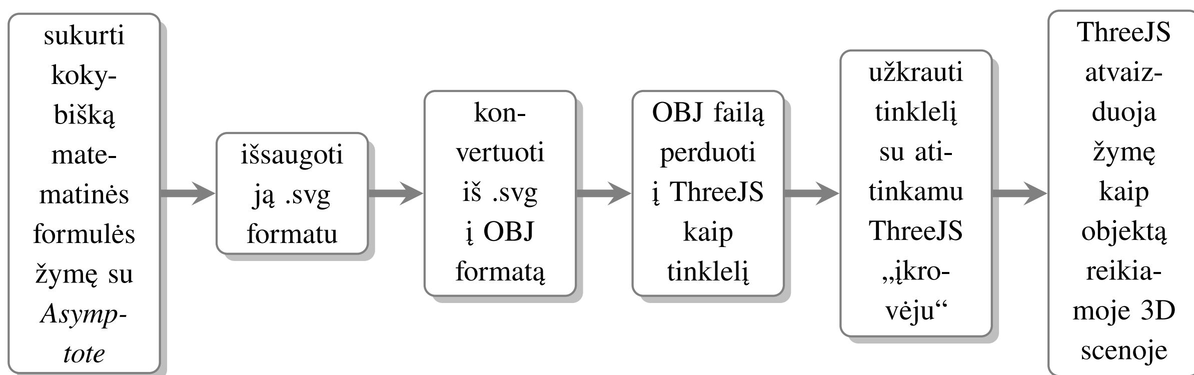


20 pav. MSDF failo, reikalingo tekstūriniam atlasui sudaryti, pavyzdys, pateikiamas [36]

Nors straipsnyje [36] pateikiama nemažai nuorodų į įrankius galinčius generuoti MSDF failus, tačiau kol kas nei vienas rastas pavyzdys nėra matematinės formulės MSDF. O ir pagal aprašymą, šiems įrankiams reikia paduoti norimo šrifto *TrueType* formato failą. Reiškia, naujai kuriamas įrankis, gavęs .svg ar .pdf formato failą su formule, turės paversti jį į .ttf ir, tuomet, pagal panašų principą, kaip ir kiti MSDF generatoriai pateikti rezultatą. Tuomet jau bus įmanoma tekstūros atlasą perduoti ThreeJS ir mėginti atvaizduoti kokybišką žymę.

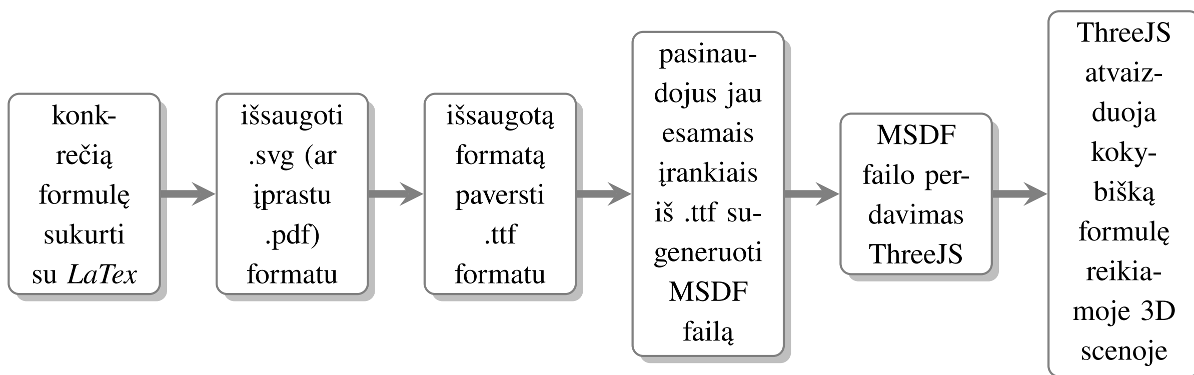
4.3. Tikėtinas rezultatas

Aukščiau aprašyti modeliai, reikalauja papildomo išsamaus tyrimo. Kol kas nėra vienareikšmiško atsakymo ar kuris nors jų užpildys dabartinę spragą ir „išmokys“ ThreeJS biblioteką atvaizduoti vektorinės grafikos kokybės tekstą (ne bet kokį tekstą, o sudėtingas matematines išraiškas). Tereikia apsispręsti kokia kryptimi judėti toliau. Ar naudoti galingosios *Asymptote* aukštos kokybė žymes ir sukurti tiesiog „tiltą“, kuriuo jau sukurta žymė emigruotų į ThreeJS:



21 pav. Asymptote žymės perdavimo į ThreeJS modelis

Ar vis tik naudotis tik ThreeJS teikiamomis galimybėmis, nepriklausant nuo kitų *API* ir naudotis tuo, kas dabar jau yra prieinama, tik šiek tiek pakoreguoti reikiamų spaudos ženklų MSDF rinkinio generatorių, kad atpažintų *LaTex* matematines formules:



22 pav. Matematinės formulės atvaizdavimo per MSDF modelis

Tiek vieno, tiek kito būdo išnagrinėjimas ir praktinis įgyvendinimas labai prisidėtų prie atsakymo, sprendžiant šiame darbe iškilusį klausimą „ką daryti, jeigu su ThreeJS jau yra sukurta sudėtinga įmantri 3D scena, ir norima į ją įterpti tokios kokybės ir funkcionalumo žymes, kurias siūlo *Asymptote*?“.

Išvados ir rekomendacijos

- Išanalizavus *Tex* veikimą, paaiškėjo, kad esminis šios technologijos uždavinys išlaikyti pasiektą aukščiausią šrifto kokybę nepriklausomai nuo aplinkos, kurioje kuriamas tekstas. Šis principas turi galioti ir žymės, kurią norima įterpti į 3D sceną, tekstui. Išanalizavus WebGL veikimą, paaiškėjo, kad trimačius objektus (ištisas 3D scenas) „išpiešia" grafinis procesorius (*GPU*), reiškia žymės 3D scenoje tekstas taip pat bus „išpieštas" su *GPU*. Norint, kad tai įvyktų, kiekvienas spaudos ženklas turi būti padalintas į trikampius. Kuo preciziškiau spaudos ženklo plotas bus padalintas ir į kuo smulkesnius trikampius, tuo kokybiškesnis bus ženklo formos ir kontūro kreivių atvaizdavimas, t. y. tuo spaudos ženklas bus arčiau vektorinės grafikos atvaizdavimo standarto, kurio tikimasi iš *Tex* teksto. Šiame darbe buvo apžvelgti visi įmanomi teksto atvaizdavimo su WebGL būdai, aptarta jų rezultato kokybė ir galimybė kiekvieną būdą panaudoti *Tex* matematinės formulės atvaizdavimui 3D scenoje.

- Įmanoma įterpti *Tex* teksto žymę į 3D sceną nenaudojant *GPU* ir šrifto geometrijos (spaudos ženklo dalinimo į trikampius). Kai kurie autoriai rekomenduoja (ir tai pavyko praktikoje) įterpti žymę kaip HTML elementą. HTML elemente atvaizduoti *Tex* šriftą yra nesudėtinga naudojant *KaTex* biblioteką. Tai išsprendė problemą kuomet buvo sudėtinga tą patį trimatį objektą su žyme prezentacinėje įrangoje pavaizduoti daugiau nei vieną kartą. Tapo įmanoma objektą su žyme vizualizuoti keliuose puslapiuose/skaidrėse, keliais skirtingais rakursais arba keičiant objekto mastelį, taip pat redaguojant galutinį rezultatą (pavyzdžiui, pastebejus klaida). *Tex* teksto žymė, esanti HTML elemente, gali dinamiškai prisitaikyti, keistis ir/ar judėti kartu su objektu, tačiau ji nėra atvaizduojama naudojant WebGL šešėliuokles.

- Įterpti *Tex* teksto žymę į 3D sceną naudojant WebGL šešėliuokles nėra sudėtinga, jeigu tam naudojama atskira technologija - *Asymptote* vektorinės grafikos programavimo kalba. *Asymptote* pagrindinis uždavinys yra į su *LaTex* kuriamą prezentacinį turinį įterpti matematines diagramas, grafikus ir brėžinius (su žymėmis), kurių kokybė būtų lygiai tokio paties *Tex* standarto, kaip ir likęs tekstas. Taip pat *Asymptote* autoriai sukūrė unikalų algoritmą, kuris visus šiuos dvimatėje plokštumoje esančius matematinius brėžinius, jų žymes ar apskirtia bet kokį *Tex* šriftą „perkelia" į trimatę erdvę ir, naudojant šešėliavimo kalbą, turinyje (arba naršyklėje) atvaizduoja kaip kokybišką, vektorinės grafikos 3D objektą.

- Išbandyta praktikoje, *Asymptote* kalba, iš tiesų, suteikė galimybę daug paprasčiau, greičiau ir kokybiškiau kurti žymes 3D scenose ir įterpti jas į prezentacinį turinį, nei tai buvo galima padaryti naudojant HTML ir *KaTex* biblioteką. Tačiau, kadangi pats žymės įterpimas tapo nesudėtingu, atsirado kiti trimačių scenų sąlygoti aspektai, apie kuriuos nebuvo žinoma žymes naudojant 2D erdvėje. Palyginimui, dvimatėje plokštumoje objekto ir jo žymės atvaizdas yra statiniai, t. y. jų padėtis nesikeičia, taigi nesikeičia nei perspektyva, nei žymių išdėstymas, nei matomumas bei informatyvumas. 3D scenoje objektai yra dinamiški, jie gali būti regimi skirtingose perspektyvose ir tai turi tiesioginės įtakos prie jų esančių žymių išdėstymui. Atsiranda plokštumų persidengimo, „netaisyklingos" objekto padėties erdvėje suvokimo bei žymės teksto įskaitomumo problemos. Siekiant jų išvengti, kiekvieną 3D sceną reikėtų vertinti atskirai bei, atsižvelgiant į joje esančio objekto niuansus, nustatyti kokio pobūdžio *Asymptote* žymių (paprastų, su rodykle, išdėstytų ant paviršiaus ar ant briaunos) poreikis yra konkrečiame prezentaciniame turinyje.

- *Asymptote* trimačių objektų žymės išlaiko *Tex* standarto kokybę ir yra nesudėtingai įterpiamos į su *LaTex* kuriamą prezentacinį turinį, tačiau 3D scena, kurioje tos žymės gali egzistuoti irgi turi būti sukurta naudojant *Asymptote*. Jeigu 3D scena yra sukurta kitomis priemonėmis, pavyzdžiui trimatei grafikai kurti plačiai naudojama ThreeJS biblioteka, įterpti į ją su *Asymptote* sukurtas žymes nėra galimybių. Kadangi ThreeJS bibliotekos pagrindinis uždavinys yra kurti sudėtingas ir įmantrias 3D scenas, kurių neįmanoma sukurti su *Asymptote*, *Tex* šrifto kokybiškas įterpimas į šias scenas lieka neišspręsta problema. Įmanoma įkelti *Tex* teksto atvaizdą (rastrinį paveikslėlį) ir, išsaugojant jį kaip tekstūrą, sukomponuoti kaip trimačio objekto žymę, tačiau tokios žymės kokybė, įterpus ją į prezentacinį turinį, neprilygsta vektorinės grafikos standartui, prie kurio lipratę *LaTex* naudotojai.

- Išanalizavus ThreeJS bibliotekos galimybes, buvo pasilūlytos dvi koncepcijos, kaip, galimai, būtų įmanoma su šia technologija sukurtoje 3D scenoje kokybiškai atvaizduoti *Tex* standarto matematinę formulę:

  Su *Asymptote* sukurtą kokybišką *Tex* teksto 3D žymę, įkelti į ThreeJS kaip tinklelį (angl. *mesh*). Tam reikėtų žymę, kurią *Asymptote* atvaizduoja .pdf, .html, .svg ar .eps formatu konvertuoti ir išsaugoti kaip OBJ. Formatas OBJ yra pažįstamas ThreeJS bibliotekai ir čia yra sukurtos priemonės ir aprašytos technikos tokio tinklelio atvaizdavimui.

  Sukurti norimos *Tex* teksto matematinės formulės MSDF tekstūros atlasą, kurį įkėlus į ThreeJS, kiekvieno spaudos ženklo duomenys bus perduoti šešėliuoklėms ir atvaizduoti 3D scenoje. MSDF yra vienas iš būdų padalinti spaudos ženklą į trikampius ir kokybiškai „išpiešti“ kontūro kreives su *GPU*. Norint sukurti MSDF failą, *Tex* matematinę formulę reikia išsaugoti .ttf formatu.

  Šiuos du konceptus rekomenduojama išsamiau paanalizuoti ir pamėginti praktiškai įgyvendinti ateities moksliniuose darbuose.

Ateities tyrimų planas

Šiame darbe aptarti ir praktiškai išbandyti galimi būdai įterpti *Tex* standarto žymes į trimačių objektų vizualizacijas tam naudojant tris skirtingus įrankius (*KaTex, Asymptote* ir ThreeJS tekstūrą). Vieni yra paprastesni, kiti sudėtingesni, vieni yra labai kokybiški, o kiti, palyginus, ne tokie kokybiški. Tačiau nei vienas iš šių būdų netinka norint įterpti su *Asymptote* sukurtų žymių kokybei prilygstančias *Tex* šrifto žymes į su ThreeJS biblioteka sukurtas įmantrias 3D scenas.

Norint sujungti šias dvi technologijas (*Asymptote* žymę su ThreeJS 3D scena), reikia išanalizuoti galimybę išsaugoti *Asymptote* algoritmo generuojamus šešėliuoklių duomenis, OBJ formatu. Žinant, kad *Asymptote* šešėliuoklėms perduoda 16 kontrolinių taškų kiekvienam Bezjė paviršiui ir 4 spalvą aprašančius taškus, nėra aišku ar konvertavus šiuos duomenis į OBJ, jie liks suprantami ThreeJS generuojamoms šešėliuoklėms. O jeigu bus perskaičiuojami kitokia išraiška, kaip tai įtakos žymės atvaizdavimą ir vektorinės grafikos *Tex* kokybę.

Jeigu paaiškėtų, kad perskaičiavus duomenis, kokybiškos *Asymptote* žymės atkurti nepavyksta, arba apskritai, būtų nuspręsta nenaudoti *Asymptote*, o viską daryti su ThreeJS, reikia išanalizuoti galimybę kurti *Tex* matematinių formulių MSDF tekstūrinį atlasą. Panašu, kad su dabar egzistuojančiomis priemonėmis tai būtų įmanoma padaryti, turint *Tex* formulės .ttf formatą. Reikia išsiaiškinti, kaip tokį formatą gauti ir ar pagal jį sukurtų MSDF duomenų užteks kokybiškai žymei su formule atvaizduoti ThreeJS 3D scenoje.

Literatūros šaltiniai

[1] About jobname tag.
https://tex.stackexchange.com/tags/jobname/info Žiūrėta: 2020-12-23.

[2] About Katex (official webpage).
https://katex.org/ Žiūrėta: 2020-12-23.

[3] About Mathjax (official webpage).
https://www.mathjax.org/#about Žiūrėta: 2020-12-23.

[4] Bauhaus typography.
http://www.designhistory.org/Avant_Garde_pages/BauhausType.html Žiūrėta: 2020-12-23.

[5] Graphics and rendering in Three.js.
https://www.oreilly.com/library/view/programming-3d-applications/9781449363918/ch04.html Žiūrėta: 2020-12-23.

[6] Loading 3D models (official ThreeJS documentation webpage).
https://threejs.org/docs/#manual/en/introduction/Loading-3D-models Žiūrėta: 2020-12-23.

[7] Tipografika kaip menas.
http://grafika.vda.lt/?page_id=815 Žiūrėta: 2020-12-23.

[8] Dusan Bosnjak. What is Three.js?, 2018.
https://medium.com/@pailhead011/what-is-three-js-7a03d84d9489 Žiūrėta: 2020-12-23.

[9] Chris Botman. An introduction to Three.js, 2016.
https://humaan.com/blog/web-3d-graphics-using-three-js Žiūrėta: 2020-12-23.

[10] Murray Bourne. Katex - a new way to display math on the web, 2014.
https://www.intmath.com/blog/mathematics/katex-a-new-way-to-display-math-on-the-web-9445.

[11] Murray Bourne. Katex and Mathjax comparison demo, 2020.
https://www.intmath.com/cg5/katex-mathjax-comparison.php Žiūrėta: 2020-12-23.

[12] John C. Bowman. The 3D Asymptote generalization of MetaPost Bézier interpolation. PAMM · Proc. Appl. Math. Mech. 7, 2007.
https://onlinelibrary.wiley.com/doi/pdf/10.1002/pamm.200700359 Žiūrėta: 2020-12-23.

[13] John C. Bowman. Asymptote: Interactive Tex-aware 3D vector graphics. TUGboat, Volume 31, No. 2, 2010.
https://www.tug.org/TUGboat/tb31-2/tb98complete.pdf Žiūrėta: 2020-12-23.

[14] John C. Bowman and Andy Hammerlindl. Asymptote: A vector graphics language. TUGboat, Volume 29, No. 2, 2008.
https://tug.org/TUGboat/tb29-2/tb92complete.pdf Žiūrėta: 2020-12-23.

[15] John C. Bowman and Orest Shardt. Asymptote: Lifting Tex to three dimensions. TUGboat, Volume 30, No. 1, 2009.
https://www.tug.org/TUGboat/tb30-1/tb94complete.pdf Žiūrėta: 2020-12-23.

[16] Dave Crossland. Why didn't METAFONT catch on? TUGboat, Volume 29 (2008), No. 3, 2008.
https://www.tug.org/TUGboat/tb29-3/tb93complete.pdf Žiūrėta: 2020-12-23.

[17] Graham Douglas. What's in a name: A guide to the many flavours of Tex, 2017.
https://www.overleaf.com/learn/latex/Articles/What's_in_a_Name:_A_Guide_to_the_Many_Flavours_of_TeX Žiūrėta: 2020-12-23.

[18] Graham Douglas. A six-part series: How do Tex macros actually work?, 2019.
https://www.overleaf.com/learn/latex/A_six-part_series:_How_do_TeX_macros_actually_work%3F Žiūrėta: 2020-12-23.

[19] Chris Green. Improved alpha-tested magnification for vector textures and special effects, 2007.
https://steamcdn-a.akamaihd.net/apps/valve/2007/SIGGRAPH2007_AlphaTestedMagnification.pdf Žiūrėta: 2020-12-23.

[20] Harvey J. Greenberg. A simplified introduction to LaTex. University of Colorado at Denver Department of Mathematical Sciences, 2004.

[21] Donald E. Knuth. The concept of a METAFONT. Visible Language, XVI 1, 3-27, 1982.
https://s3-us-west-2.amazonaws.com/visiblelanguage/pdf/16.1/the-concept-of-a-meta-font.pdf Žiūrėta: 2020-12-23.

[22] Donald E. Knuth. The TeXbook. Addison-Wesley Educational Publishers Inc, 1984.

[23] Donald E. Knuth. The new versions of Tex and METAFONT. TUGboat, Volume 10, No. 3, 1989.
https://www.tug.org/TUGboat/tb10-3/tb25complete.pdf Žiūrėta: 2020-12-23.

[24] Donald E. Knuth. The future of Tex and METAFONT. TUGboat, Volume 11, No. 4, 1990.
https://www.tug.org/TUGboat/tb11-4/tb30complete.pdf Žiūrėta: 2020-12-23.

[25] Christian Lawson-Perfect. Katex is a (partial) alternative to (some of) Mathjax, 2014.
https://aperiodical.com/2014/09/katex-the-fastest-math-typesetting-library-for-the-web/ Žiūrėta: 2020-12-23.

[26] Charles Loop and Jim Blinn. Rendering vector art on the GPU, 2005.
https://developer.nvidia.com/gpugems/gpugems3/part-iv-image-e ects/chapter-25-rendering-vector-art-gpu Žiūrėta: 2020-12-23.

[27] Kouichi Matsuda and Rodger Lea. WebGL programming guide: Interactive 3D graphics programming with WebGL. Addison-Wesley Professional, 2013.
http://uniguld.dk/wp-content/guld/DTU/webgrafik/0321902920_WebGL.pdf Žiūrėta: 2020-12-23.

[28] Joseph Rautenbach. Converting Latex equations to images online with Docker and Node.js.
https://latex2image.joeraut.com/ Žiūrėta: 2020-12-23.

[29] Orest Shardt and John C. Bowman. Surface parametrization of nonsimply connected planar Bézier regions. Computer-Aided Design, Volume 44, Issue 5, 2012.
http://dx.doi.org/10.1016/j.cad.2011.05.010 Žiūrėta: 2020-12-23.

[30] Gregg Tavares. 3D perspective, 2014.
https://webglfundamentals.org/webgl/lessons/webgl-3d-perspective.html Žiūrėta: 2020-12-23.

[31] Gregg Tavares. Rasterization vs 3D libraries, 2014.
https://webglfundamentals.org/webgl/lessons/webgl-2d-vs-3d-library.html Žiūrėta: 2020-12-23.

[32] Gregg Tavares. WebGL fundamentals, 2014.
https://webglfundamentals.org/webgl/lessons/webgl-fundamentals.html Žiūrėta: 2020-12-23.

[33] Gregg Tavares. WebGL text - HTML, 2014.
https://webglfundamentals.org/webgl/lessons/webgl-text-html.html Žiūrėta: 2020-12-23.

[34] Geoffrey Tobin. METAFONT for beginners. Comprehensive Tex Archive Network, 1994.
https://www.ntg.nl/doc/tobin/mf4begin.pdf Žiūrėta: 2020-12-23.

[35] Kees van der Laan. What is Tex and METAFONT all about? NLUUG meeting Fall '93, 1994.
https://www.ntg.nl/doc/vanderlaan/overview.pdf Žiūrėta: 2020-12-23.

[36] Daniel Velasquez. Techniques for rendering text with WebGL, 2019.
https://css-tricks.com/techniques-for-rendering-text-with-webgl/ Žiūrėta: 2020-12-23.

Priedai

Dokumentą sudaro du priedai:

priede [A] pateikiamas į šį darbą įterptas *Asymptote* išeities kodas, atvaizduojantis trimačius objektus;

priede [B] pateikiamas su *Asymptote* atvaizduoto trimačio objekto išeities kodas.

A. Trimačiai objektai su žymėmis atvaizduojami su LaTeX kuriamame turinyje (.pdf formatu)

1 išeities kodas. Gretasienis su paprasta žyme

```
1
2   \begin{asy}
3   import three;
4   import labelpath3;
5   size(6cm,0);
6
7   string txt="$f(x) = \int_{-\infty}^\infty \hat f(\xi)\,e^{2 \pi i \
       xi x} d\xi$";
8
9   real height = 0.5;
10  triple top = (0, 0, height);
11  real width = 0.5;
12  triple right = (0, width, 0);
13  real depth = 0.3;
14  triple front = (depth, 0, 0);
15
16  // solid box
17  draw(scale(depth, width, height)*unitcube, surfacepen=paleblue);
18
19  dot(top, red);
20
21  // a billboard label on a red dot
22  label(txt, top, deepgreen, align=NE);
23
24  \end{asy}
```

2 išeities kodas. Gretasienis su žyme rodykle

```
1
2  \begin{asy}
3  import three;
4  import labelpath3;
5  size(9cm,0);
6
7  string txt="$f(x) = \int_{-\infty}^\infty \hat f(\xi)\,e^{2 \pi i \xi
      x} d\xi$";
8
9  real height = 0.5;
10 triple top = (0, 0, height);
11 real width = 0.5;
12 triple right = (0, width, 0);
13 real depth = 0.3;
14 triple front = (depth, 0, 0);
15
16 // solid box
17 draw(scale(depth, width, height)*unitcube, surfacepen=paleblue);
18
19 dot(right, green);
20
```

```
21  // an arrow with a label
22  arrow(txt, right, Y+Z, 1cm, heavyred);
23
24  \end{asy}
```

3 išeities kodas. Sfera su žyme ant paviršiaus

```
1
2   \begin{asy}
3   import three;
4   import labelpath3;
5   size(7cm,0);
6
7   string txt="$f(x) = \int_{-\infty}^\infty \hat f(\xi)\,e^{2 \pi i \xi
      x} d\xi$";
8
9   real height = 1;
10  triple top = (0, 0, height);
11  real width = 1;
12  triple right = (0, width, 0);
13  real depth = 1;
14  triple front = (depth, 0, 0);
15
16  // solid sphere
17  draw(scale(depth, width, height)*unitsphere, surfacepen=paleblue);
18
19  dot(front, yellow);
20
21  draw(surface(yscale(0.1)*txt, scale(depth, width, height)*unitsphere,
      0, 0, 0.008), magenta);
22
23  \end{asy}
```

4 išeities kodas. TEX tekstas 3D

```
1
2   \begin{asy}
3   import three;
4   size(10cm,0);
5
6   string txt="$f(x) = \int_{-\infty}^\infty \hat f(\xi)\,e^{2 \pi i \xi
      x} d\xi$";
7
8   // 3D formula
9   currentprojection=perspective(100,100,200,up=Y);
10  draw(scale3(4)*extrude(txt,2Z),gray);
11
12  \end{asy}
```

B. 3D objekto perpiešimui su *Asymptote* reikalingų duomenų pavyzdys

5 išeities kodas. Su *Asymptote* atvaizduoto arbatinuko išeities kodas

```
1  import three;
2  import settings;
3  size(5cm);
4
5  currentprojection=perspective(250,-250,250);
6  currentlight=Viewport;
7
8  triple[][][] Q=
9  {
10 {
11    {(39.68504,0,68.0315),(37.91339,0,71.75197),(40.74803,0,71.75197)
          ,(42.51969,0,68.0315)},
12    {(39.68504,-22.22362,68.0315),(37.91339,-21.2315,71.75197)
          ,(40.74803,-22.8189,71.75197),(42.51969,-23.81102,68.0315)},
13    {(22.22362,-39.68504,68.0315),(21.2315,-37.91339,71.75197)
          ,(22.8189,-40.74803,71.75197),(23.81102,-42.51969,68.0315)},
14    {(0,-39.68504,68.0315),(0,-37.91339,71.75197),(0,-40.74803,71.75197)
          ,(0,-42.51969,68.0315)}
15 },
16 {
17    {(0,-39.68504,68.0315),(0,-37.91339,71.75197),(0,-40.74803,71.75197)
          ,(0,-42.51969,68.0315)},
18    {(-22.22362,-39.68504,68.0315),(-21.2315,-37.91339,71.75197)
          ,(-22.8189,-40.74803,71.75197),(-23.81102,-42.51969,68.0315)},
19    {(-39.68504,-22.22362,68.0315),(-37.91339,-21.2315,71.75197)
          ,(-40.74803,-22.8189,71.75197),(-42.51969,-23.81102,68.0315)},
20    {(-39.68504,0,68.0315),(-37.91339,0,71.75197),(-40.74803,0,71.75197)
          ,(-42.51969,0,68.0315)}
21 },
22 {
23    {(-39.68504,0,68.0315),(-37.91339,0,71.75197),(-40.74803,0,71.75197)
          ,(-42.51969,0,68.0315)},
24    {(-39.68504,22.22362,68.0315),(-37.91339,21.2315,71.75197)
          ,(-40.74803,22.8189,71.75197),(-42.51969,23.81102,68.0315)},
25    {(-22.22362,39.68504,68.0315),(-21.2315,37.91339,71.75197)
          ,(-22.8189,40.74803,71.75197),(-23.81102,42.51969,68.0315)},
26    {(0,39.68504,68.0315),(0,37.91339,71.75197),(0,40.74803,71.75197)
          ,(0,42.51969,68.0315)}
27 },
28 {
29    {(0,39.68504,68.0315),(0,37.91339,71.75197),(0,40.74803,71.75197)
          ,(0,42.51969,68.0315)},
30    {(22.22362,39.68504,68.0315),(21.2315,37.91339,71.75197)
          ,(22.8189,40.74803,71.75197),(23.81102,42.51969,68.0315)},
31    {(39.68504,22.22362,68.0315),(37.91339,21.2315,71.75197)
          ,(40.74803,22.8189,71.75197),(42.51969,23.81102,68.0315)},
32    {(39.68504,0,68.0315),(37.91339,0,71.75197),(40.74803,0,71.75197)
          ,(42.51969,0,68.0315)}
33 },
```

```
34   {
35     {(42.51969,0,68.0315),(49.60629,0,53.1496),(56.69291,0,38.26771)
         ,(56.69291,0,25.51181)},
36     {(42.51969,−23.81102,68.0315),(49.60629,−27.77952,53.1496)
         ,(56.69291,−31.74803,38.26771),(56.69291,−31.74803,25.51181)},
37     {(23.81102,−42.51969,68.0315),(27.77952,−49.60629,53.1496)
         ,(31.74803,−56.69291,38.26771),(31.74803,−56.69291,25.51181)},
38     {(0,−42.51969,68.0315),(0,−49.60629,53.1496),(0,−56.69291,38.26771)
         ,(0,−56.69291,25.51181)}
39   },
40   {
41     {(0,−42.51969,68.0315),(0,−49.60629,53.1496),(0,−56.69291,38.26771)
         ,(0,−56.69291,25.51181)},
42     {(−23.81102,−42.51969,68.0315),(−27.77952,−49.60629,53.1496)
         ,(−31.74803,−56.69291,38.26771),(−31.74803,−56.69291,25.51181)},
43     {(−42.51969,−23.81102,68.0315),(−49.60629,−27.77952,53.1496)
         ,(−56.69291,−31.74803,38.26771),(−56.69291,−31.74803,25.51181)},
44     {(−42.51969,0,68.0315),(−49.60629,0,53.1496),(−56.69291,0,38.26771)
         ,(−56.69291,0,25.51181)}
45   },
46   {
47     {(−42.51969,0,68.0315),(−49.60629,0,53.1496),(−56.69291,0,38.26771)
         ,(−56.69291,0,25.51181)},
48     {(−42.51969,23.81102,68.0315),(−49.60629,27.77952,53.1496)
         ,(−56.69291,31.74803,38.26771),(−56.69291,31.74803,25.51181)},
49     {(−23.81102,42.51969,68.0315),(−27.77952,49.60629,53.1496)
         ,(−31.74803,56.69291,38.26771),(−31.74803,56.69291,25.51181)},
50     {(0,42.51969,68.0315),(0,49.60629,53.1496),(0,56.69291,38.26771)
         ,(0,56.69291,25.51181)}
51   },
52   {
53     {(0,42.51969,68.0315),(0,49.60629,53.1496),(0,56.69291,38.26771)
         ,(0,56.69291,25.51181)},
54     {(23.81102,42.51969,68.0315),(27.77952,49.60629,53.1496)
         ,(31.74803,56.69291,38.26771),(31.74803,56.69291,25.51181)},
55     {(42.51969,23.81102,68.0315),(49.60629,27.77952,53.1496)
         ,(56.69291,31.74803,38.26771),(56.69291,31.74803,25.51181)},
56     {(42.51969,0,68.0315),(49.60629,0,53.1496),(56.69291,0,38.26771)
         ,(56.69291,0,25.51181)}
57   },
58   {
59     {(56.69291,0,25.51181),(56.69291,0,12.7559),(42.51969,0,6.377957)
         ,(42.51969,0,4.251961)},
60     {(56.69291,−31.74803,25.51181),(56.69291,−31.74803,12.7559)
         ,(42.51969,−23.81102,6.377957),(42.51969,−23.81102,4.251961)},
61     {(31.74803,−56.69291,25.51181),(31.74803,−56.69291,12.7559)
         ,(23.81102,−42.51969,6.377957),(23.81102,−42.51969,4.251961)},
62     {(0,−56.69291,25.51181),(0,−56.69291,12.7559),(0,−42.51969,6.377957)
         ,(0,−42.51969,4.251961)}
63   },
64   {
65     {(0,−56.69291,25.51181),(0,−56.69291,12.7559),(0,−42.51969,6.377957)
         ,(0,−42.51969,4.251961)},
66     {(−31.74803,−56.69291,25.51181),(−31.74803,−56.69291,12.7559)
         ,(−23.81102,−42.51969,6.377957),(−23.81102,−42.51969,4.251961)},
```

```
67      {(−56.69291,−31.74803,25.51181),(−56.69291,−31.74803,12.7559)
            ,(−42.51969,−23.81102,6.377957),(−42.51969,−23.81102,4.251961)},
68      {(−56.69291,0,25.51181),(−56.69291,0,12.7559),(−42.51969,0,6.377957)
            ,(−42.51969,0,4.251961)}
69    },
70    {
71      {(−56.69291,0,25.51181),(−56.69291,0,12.7559),(−42.51969,0,6.377957)
            ,(−42.51969,0,4.251961)},
72      {(−56.69291,31.74803,25.51181),(−56.69291,31.74803,12.7559)
            ,(−42.51969,23.81102,6.377957),(−42.51969,23.81102,4.251961)},
73      {(−31.74803,56.69291,25.51181),(−31.74803,56.69291,12.7559)
            ,(−23.81102,42.51969,6.377957),(−23.81102,42.51969,4.251961)},
74      {(0,56.69291,25.51181),(0,56.69291,12.7559),(0,42.51969,6.377957)
            ,(0,42.51969,4.251961)}
75    },
76    {
77      {(0,56.69291,25.51181),(0,56.69291,12.7559),(0,42.51969,6.377957)
            ,(0,42.51969,4.251961)},
78      {(31.74803,56.69291,25.51181),(31.74803,56.69291,12.7559)
            ,(23.81102,42.51969,6.377957),(23.81102,42.51969,4.251961)},
79      {(56.69291,31.74803,25.51181),(56.69291,31.74803,12.7559)
            ,(42.51969,23.81102,6.377957),(42.51969,23.81102,4.251961)},
80      {(56.69291,0,25.51181),(56.69291,0,12.7559),(42.51969,0,6.377957)
            ,(42.51969,0,4.251961)}
81    },
82    {
83      {(−45.35433,0,57.40157),(−65.19685,0,57.40157)
            ,(−76.53543,0,57.40157),(−76.53543,0,51.02362)},
84      {(−45.35433,−8.503932,57.40157),(−65.19685,−8.503932,57.40157)
            ,(−76.53543,−8.503932,57.40157),(−76.53543,−8.503932,51.02362)},
85      {(−42.51969,−8.503932,63.77952),(−70.86614,−8.503932,63.77952)
            ,(−85.03937,−8.503932,63.77952),(−85.03937,−8.503932,51.02362)},
86      {(−42.51969,0,63.77952),(−70.86614,0,63.77952)
            ,(−85.03937,0,63.77952),(−85.03937,0,51.02362)}
87    },
88    {
89      {(−42.51969,0,63.77952),(−70.86614,0,63.77952)
            ,(−85.03937,0,63.77952),(−85.03937,0,51.02362)},
90      {(−42.51969,8.503932,63.77952),(−70.86614,8.503932,63.77952)
            ,(−85.03937,8.503932,63.77952),(−85.03937,8.503932,51.02362)},
91      {(−45.35433,8.503932,57.40157),(−65.19685,8.503932,57.40157)
            ,(−76.53543,8.503932,57.40157),(−76.53543,8.503932,51.02362)},
92      {(−45.35433,0,57.40157),(−65.19685,0,57.40157)
            ,(−76.53543,0,57.40157),(−76.53543,0,51.02362)}
93    },
94    {
95      {(−76.53543,0,51.02362),(−76.53543,0,44.64566)
            ,(−70.86614,0,31.88976),(−56.69291,0,25.51181)},
96      {(−76.53543,−8.503932,51.02362),(−76.53543,−8.503932,44.64566)
            ,(−70.86614,−8.503932,31.88976),(−56.69291,−8.503932,25.51181)},
97      {(−85.03937,−8.503932,51.02362),(−85.03937,−8.503932,38.26771)
            ,(−75.11811,−8.503932,26.5748),(−53.85826,−8.503932,17.00787)},
98      {(−85.03937,0,51.02362),(−85.03937,0,38.26771),(−75.11811,0,26.5748)
            ,(−53.85826,0,17.00787)}
99    },
```

```
100 {
101   {(−85.03937,0,51.02362),(−85.03937,0,38.26771),(−75.11811,0,26.5748)
        ,(−53.85826,0,17.00787)},
102   {(−85.03937,8.503932,51.02362),(−85.03937,8.503932,38.26771)
        ,(−75.11811,8.503932,26.5748),(−53.85826,8.503932,17.00787)},
103   {(−76.53543,8.503932,51.02362),(−76.53543,8.503932,44.64566)
        ,(−70.86614,8.503932,31.88976),(−56.69291,8.503932,25.51181)},
104   {(−76.53543,0,51.02362),(−76.53543,0,44.64566)
        ,(−70.86614,0,31.88976),(−56.69291,0,25.51181)}
105 },
106 {
107   {(48.18897,0,40.3937),(73.70078,0,40.3937),(65.19685,0,59.52755)
        ,(76.53543,0,68.0315)},
108   {(48.18897,−18.70866,40.3937),(73.70078,−18.70866,40.3937)
        ,(65.19685,−7.086619,59.52755),(76.53543,−7.086619,68.0315)},
109   {(48.18897,−18.70866,17.00787),(87.87401,−18.70866,23.38582)
        ,(68.0315,−7.086619,57.40157),(93.5433,−7.086619,68.0315)},
110   {(48.18897,0,17.00787),(87.87401,0,23.38582),(68.0315,0,57.40157)
        ,(93.5433,0,68.0315)}
111 },
112 {
113   {(48.18897,0,17.00787),(87.87401,0,23.38582),(68.0315,0,57.40157)
        ,(93.5433,0,68.0315)},
114   {(48.18897,18.70866,17.00787),(87.87401,18.70866,23.38582)
        ,(68.0315,7.086619,57.40157),(93.5433,7.086619,68.0315)},
115   {(48.18897,18.70866,40.3937),(73.70078,18.70866,40.3937)
        ,(65.19685,7.086619,59.52755),(76.53543,7.086619,68.0315)},
116   {(48.18897,0,40.3937),(73.70078,0,40.3937),(65.19685,0,59.52755)
        ,(76.53543,0,68.0315)}
117 },
118 {
119   {(76.53543,0,68.0315),(79.37007,0,70.15748),(82.20472,0,70.15748)
        ,(79.37007,0,68.0315)},
120   {(76.53543,−7.086619,68.0315),(79.37007,−7.086619,70.15748)
        ,(82.20472,−4.251961,70.15748),(79.37007,−4.251961,68.0315)},
121   {(93.5433,−7.086619,68.0315),(99.92125,−7.086619,70.68897)
        ,(97.79527,−4.251961,71.22047),(90.70866,−4.251961,68.0315)},
122   {(93.5433,0,68.0315),(99.92125,0,70.68897),(97.79527,0,71.22047)
        ,(90.70866,0,68.0315)}
123 },
124 {
125   {(93.5433,0,68.0315),(99.92125,0,70.68897),(97.79527,0,71.22047)
        ,(90.70866,0,68.0315)},
126   {(93.5433,7.086619,68.0315),(99.92125,7.086619,70.68897)
        ,(97.79527,4.251961,71.22047),(90.70866,4.251961,68.0315)},
127   {(76.53543,7.086619,68.0315),(79.37007,7.086619,70.15748)
        ,(82.20472,4.251961,70.15748),(79.37007,4.251961,68.0315)},
128   {(76.53543,0,68.0315),(79.37007,0,70.15748),(82.20472,0,70.15748)
        ,(79.37007,0,68.0315)}
129 },
130 {
131   {(0,0,89.29133),(22.67716,0,89.29133),(0,0,80.7874)
        ,(5.669294,0,76.53543)},
132   {(0,0,89.29133),(22.67716,−12.7559,89.29133),(0,0,80.7874)
        ,(5.669294,−3.174809,76.53543)},
```

```
133     {(0,0,89.29133),(12.7559,−22.67716,89.29133),(0,0,80.7874)
            ,(3.174809,−5.669294,76.53543)},
134     {(0,0,89.29133),(0,−22.67716,89.29133),(0,0,80.7874)
            ,(0,−5.669294,76.53543)}
135   },
136   {
137     {(0,0,89.29133),(0,−22.67716,89.29133),(0,0,80.7874)
            ,(0,−5.669294,76.53543)},
138     {(0,0,89.29133),(−12.7559,−22.67716,89.29133),(0,0,80.7874)
            ,(−3.174809,−5.669294,76.53543)},
139     {(0,0,89.29133),(−22.67716,−12.7559,89.29133),(0,0,80.7874)
            ,(−5.669294,−3.174809,76.53543)},
140     {(0,0,89.29133),(−22.67716,0,89.29133),(0,0,80.7874)
            ,(−5.669294,0,76.53543)}
141   },
142   {
143     {(0,0,89.29133),(−22.67716,0,89.29133),(0,0,80.7874)
            ,(−5.669294,0,76.53543)},
144     {(0,0,89.29133),(−22.67716,12.7559,89.29133),(0,0,80.7874)
            ,(−5.669294,3.174809,76.53543)},
145     {(0,0,89.29133),(−12.7559,22.67716,89.29133),(0,0,80.7874)
            ,(−3.174809,5.669294,76.53543)},
146     {(0,0,89.29133),(0,22.67716,89.29133),(0,0,80.7874)
            ,(0,5.669294,76.53543)}
147   },
148   {
149     {(0,0,89.29133),(0,22.67716,89.29133),(0,0,80.7874)
            ,(0,5.669294,76.53543)},
150     {(0,0,89.29133),(12.7559,22.67716,89.29133),(0,0,80.7874)
            ,(3.174809,5.669294,76.53543)},
151     {(0,0,89.29133),(22.67716,12.7559,89.29133),(0,0,80.7874)
            ,(5.669294,3.174809,76.53543)},
152     {(0,0,89.29133),(22.67716,0,89.29133),(0,0,80.7874)
            ,(5.669294,0,76.53543)}
153   },
154   {
155     {(5.669294,0,76.53543),(11.33858,0,72.28346),(36.85039,0,72.28346)
            ,(36.85039,0,68.0315)},
156     {(5.669294,−3.174809,76.53543),(11.33858,−6.349609,72.28346)
            ,(36.85039,−20.63622,72.28346),(36.85039,−20.63622,68.0315)},
157     {(3.174809,−5.669294,76.53543),(6.349609,−11.33858,72.28346)
            ,(20.63622,−36.85039,72.28346),(20.63622,−36.85039,68.0315)},
158     {(0,−5.669294,76.53543),(0,−11.33858,72.28346)
            ,(0,−36.85039,72.28346),(0,−36.85039,68.0315)}
159   },
160   {
161     {(0,−5.669294,76.53543),(0,−11.33858,72.28346)
            ,(0,−36.85039,72.28346),(0,−36.85039,68.0315)},
162     {(−3.174809,−5.669294,76.53543),(−6.349609,−11.33858,72.28346)
            ,(−20.63622,−36.85039,72.28346),(−20.63622,−36.85039,68.0315)},
163     {(−5.669294,−3.174809,76.53543),(−11.33858,−6.349609,72.28346)
            ,(−36.85039,−20.63622,72.28346),(−36.85039,−20.63622,68.0315)},
164     {(−5.669294,0,76.53543),(−11.33858,0,72.28346)
            ,(−36.85039,0,72.28346),(−36.85039,0,68.0315)},
165   },
```

```
166  {
167     {(−5.669294,0,76.53543),(−11.33858,0,72.28346)
           ,(−36.85039,0,72.28346),(−36.85039,0,68.0315)},
168     {(−5.669294,3.174809,76.53543),(−11.33858,6.349609,72.28346)
           ,(−36.85039,20.63622,72.28346),(−36.85039,20.63622,68.0315)},
169     {(−3.174809,5.669294,76.53543),(−6.349609,11.33858,72.28346)
           ,(−20.63622,36.85039,72.28346),(−20.63622,36.85039,68.0315)},
170     {(0,5.669294,76.53543),(0,11.33858,72.28346),(0,36.85039,72.28346)
           ,(0,36.85039,68.0315)}
171  },
172  {
173     {(0,5.669294,76.53543),(0,11.33858,72.28346),(0,36.85039,72.28346)
           ,(0,36.85039,68.0315)},
174     {(3.174809,5.669294,76.53543),(6.349609,11.33858,72.28346)
           ,(20.63622,36.85039,72.28346),(20.63622,36.85039,68.0315)},
175     {(5.669294,3.174809,76.53543),(11.33858,6.349609,72.28346)
           ,(36.85039,20.63622,72.28346),(36.85039,20.63622,68.0315)},
176     {(5.669294,0,76.53543),(11.33858,0,72.28346),(36.85039,0,72.28346)
           ,(36.85039,0,68.0315)},
177  },
178  {
179     {(0,0,0),(40.3937,0,0),(42.51969,0,2.12598),(42.51969,0,4.251961)},
180     {(0,0,0),(40.3937,22.62047,0),(42.51969,23.81102,2.12598)
           ,(42.51969,23.81102,4.251961)},
181     {(0,0,0),(22.62047,40.3937,0),(23.81102,42.51969,2.12598)
           ,(23.81102,42.51969,4.251961)},
182     {(0,0,0),(0,40.3937,0),(0,42.51969,2.12598),(0,42.51969,4.251961)}
183  },
184  {
185     {(0,0,0),(0,40.3937,0),(0,42.51969,2.12598),(0,42.51969,4.251961)},
186     {(0,0,0),(−22.62047,40.3937,0),(−23.81102,42.51969,2.12598)
           ,(−23.81102,42.51969,4.251961)},
187     {(0,0,0),(−40.3937,22.62047,0),(−42.51969,23.81102,2.12598)
           ,(−42.51969,23.81102,4.251961)},
188     {(0,0,0),(−40.3937,0,0),(−42.51969,0,2.12598),(−42.51969,0,4.251961)
           }
189  },
190  {
191     {(0,0,0),(−40.3937,0,0),(−42.51969,0,2.12598),(−42.51969,0,4.251961)
           },
192     {(0,0,0),(−40.3937,−22.62047,0),(−42.51969,−23.81102,2.12598)
           ,(−42.51969,−23.81102,4.251961)},
193     {(0,0,0),(−22.62047,−40.3937,0),(−23.81102,−42.51969,2.12598)
           ,(−23.81102,−42.51969,4.251961)},
194     {(0,0,0),(0,−40.3937,0),(0,−42.51969,2.12598),(0,−42.51969,4.251961)
           }
195  },
196  {
197     {(0,0,0),(0,−40.3937,0),(0,−42.51969,2.12598),(0,−42.51969,4.251961)
           },
198     {(0,0,0),(22.62047,−40.3937,0),(23.81102,−42.51969,2.12598)
           ,(23.81102,−42.51969,4.251961)},
199     {(0,0,0),(40.3937,−22.62047,0),(42.51969,−23.81102,2.12598)
           ,(42.51969,−23.81102,4.251961)},
200     {(0,0,0),(40.3937,0,0),(42.51969,0,2.12598),(42.51969,0,4.251961)}
```

```
201     }
202     };
203
204     draw(surface(Q),material(blue, shininess=0.85, metallic=0),render(
            compression=Low));
```