Interaktivno platformno agnostično sledenje žarkov v realnem času s spletnimi tehnologijami

Žiga Lesar, Ciril Bohak, Matija Marolt
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana
{ziga.lesar, ciril.bohak, matija.marolt}@fri.uni-lj.si

Izvleček

Metoda sledenja potem je trenutni de facto standard za fotorealistično upodabljanje 3D prostorov zaradi svoje konceptualne in algoritmične enostavnosti. V zadnjih letih je bila uspešno uporabljena za upodabljanje prosojnih medijev in volumetričnih podatkov, a se njena širša uporaba ni prijela. Večina implementacij namreč za hitrejše doseganje rezultatov cilja na specifične platforme oz. strojno opremo, zato so posledično manj razširljive in zahtevnejše za namestitev. Kljub temu nam nedavne izboljšave na področju spletnih tehnologij omogočajo dostop do grafične strojne opreme iz spletnega brskalnika na platformno agnostičen način. V članku je predstavljena sodobna implementacija metode sledenja potem za volumetrične podatke, razvita v programskem jeziku JavaScript in s programskim vmesnikom WebGL 2.0. Rešitev podpira uporabo poljubnih 2D prenosnih funkcij in heterogenih volumetričnih podatkov, hkrati pa je interaktivna, platformno agnostična, enostavno razširljiva ter se izvaja v realnem času tako na namiznih kot mobilnih napravah.

Ključne besede: sledenje potem, WebGL, volumetrično upodabljanje, spletne tehnologije, progresivno upodabljanje, stohastično upodabljanje.

Abstract

Interactive platform-agnostic real-time path tracing using web technologies

Path tracing is the current de facto standard for photorealistic rendering, largely due to its conceptual and algorithmic simplicity. Lately it has been successfully applied to rendering of participating media and volumetric data, but it has not received much attention. In fact, most implementations target specific platforms or hardware to achieve better performance, and are therefore less extensible and more difficult to deploy. However, modern web browsers allow access to graphics hardware in a platform-independent manner. In this work, we present a modern implementation of volume path tracing developed in JavaScript and WebGL 2.0. Our solution supports arbitrary volume data and 2D transfer functions while being interactive, platform-agnostic, easily extensible and can run on both desktop and mobile devices in real time.

Keywords: Path tracing, WebGL, volumetric rendering, web technologies, progressive rendering, stochastic rendering.

1 UVOD

Metode neposrednega volumetričnega upodabljanja (angl. direct volume rendering, DVR) in sledenja žarkom so se od svojega spočetja v osemdesetih letih (Kajiya, 1986; Levoy, 1990) precej razvile. Prvotne sheme upodabljanja, kot sta projekcija največje intenzitete (angl. maximum intensity projection, MIP) in emisijsko-absorpcijski model (angl. emission-absorption model, EAM), so bile deležne številnih izboljšav, razširitev in prilagoditev za izvajanje na grafični strojni opremi. Večina teh izboljšav cilja na boljšo vizualno podobo s simuliranjem določenih učinkov globalne osvetlitve, npr. sence, sipanje žarkov in ambientno zastiranje. Predlagani so bili tudi modeli za simuliranje sistemov leč kamere, ki nam omogočajo nastavljanje ekspozicije in globinske ostrine (Barsky et al., 2003; Kolb et al., 1995). Tovrstni učinki bistveno prispevajo k dojemanju velikosti, globine in oblik predmetov v 3D prostoru, zato je veliko raziskovalne dejavnosti usmerjene v izboljšavo njihove kakovosti in računske učinkovitosti. Predlagane so bile številne aproksimacijske rešitve, na primer lokalne metode in računanje v koordinatnem sistemu zaslona, toda visokokakovostno upodabljanje je neizogibno računsko zahtevno. Večina sodobnih implementacij izkorišča grafično strojno opremo za vzporedno izvajanje operacij, toda specifike strojne opreme in popolnoma

drugačen programski model silijo razvoj metod k upoštevanju številnih omejitev, kar posledično privede do rešitev, ki delujejo le v določenem okolju, npr. na določeni platformi, s konstantno prenosno funkcijo ali s statično osvetljavo.

Poleg tega ne moremo uporabiti običajnih metod za upodabljanje neprosojnih materialov, saj tega inherentno ne podpirajo, ali pa jih moramo močno prilagoditi (npr. prilagoditev dvosmernega sledenja poti, kot ga predstavljata Lafortune in Willems, 1996). Prilagoditve so pogosto računsko prezahtevne. Splošni pristop za upodabljanje tako neprosojnih kot prosojnih materialov uporablja stohastične metode, ki najprej izračunajo sliko nižje kakovosti, nato pa to sliko skozi čas izboljšujejo. Družina takih metod se imenuje sledenje žarkom Monte Carlo (angl. Monte Carlo ray tracing, MCRT).

Slika 1: **Prikaz progresivnega izboljševanja izrisa**

Metoda MCRT je že dolgo de facto standard za fotorealistično upodabljanje, saj jo enostavno formuliramo, implementiramo in paraleliziramo. Če jo združimo s fizikalno osnovanim prenosom svetlobe, lahko progresivno simuliramo tudi obnašanje svetlobe v prosojnih medijih, kar privede do interaktivne rešitve in realističnih upodobitev. Obstajajo številne implementacije, ki so prilagojene za izvajanje na grafični strojni opremi in z omenjenimi omejitvami predstavljene v Davidovič et al. (2014), Hachisuka et al. (2015), Hadwiger et al. (2006) in Kroes et al. (2012).

Zahvaljujoč sodobnemu napredku na področjih grafične strojne opreme in spletnih tehnologij lahko dandanes razvijamo kompleksne grafične aplikacije, ki se izvajajo v spletnem brskalniku. Sodobni prispevki s področja volumetričnega upodabljanja temu trendu ne sledijo in se v veliki meri osredotočajo le na metodologijo, ne pa na samo implementacijo, kar privede do različnih oblik platformne odvisnosti. Primeri obsegajo uporabo Microsoftovega programskega vmesnika DirectX za dostop do grafične strojne opreme (ki deluje le na Microsoftovih platformah) ali uporabo tehnologije Nvidia CUDA za paralelizacijo (ki deluje le na grafičnih karticah Nvidia). Kljub temu da so te tehnologije močno podprte in v široki uporabi, je v mnogih primerih nesmiselno omejiti rešitve na določene platforme ali strojno opremo predvsem z vidikov dostopnosti in vzdrževanja. Spletne tehnologije nam po drugi strani omogočajo primerljive računske zmožnosti na platformno agnostičen način, npr. s programskim vmesnikom WebGL za zahtevnejše grafične aplikacije.

Standard WebGL so od njegovega nastanka leta 2011 sprejeli mnogi programerji aplikacij in spletnih brskalnikov kot platformno agnostičen način za dostop do zmogljivosti sodobne grafične strojne opreme. Temelji na standardu OpenGL ES 2.0, ki je prirejen za delovanje na vgrajenih sistemih in mobilnih napravah. Zmogljivost standardov ES običajno sledi nekaj let za običajnimi standardi za namizne naprave, tako da je nabor funkcionalnosti glede na najsodobnejšo tehnologijo omejen. Poleg tega je programski vmesnik načrtovan z mislijo na slabše zmogljivosti vgrajenih sistemov, kar pogosto pomeni edinstvene izzive pri razvoju kompleksnih grafičnih aplikacij.

V zadnjih letih je bila v pripravi nova različica standarda, WebGL 2.0. Trenutno je podpora novega standarda privzeto vključena v večini pomembnejših spletnih brskalnikov, tako na namiznih kot na mobilnih napravah (Google Chrome 56 za namizne naprave, Google Chrome 58 za Android, Android WebView 58, Opera 43 in Mozilla Firefox 51). Nova različica standarda v primerjavi s staro prinaša nemalo dobrodošlih izboljšav, predvsem z boljšo računsko učinkovitostjo in dopolnitvami programskega vmesnika, kar olajša razvoj in interakcijo s strojno opremo. Izstopata dva omembe vredna dodatka: 3D teksture, ki so še posebno dobrodošle pri upodabljanju volumetričnih podatkov, in številni novi formati tekstur, kar nam omogoča upodabljanje v visokem dinamičnem razponu (angl. high dynamic range, HDR) brez uporabe razširitev standarda. Čeprav so formati s plavajočo vejico podprti že v osnovnem

profilu standarda, je pisanje v take teksture še vedno dostopno le prek standardne razširitve.

WebGL je že bil uporabljen za razvoj naprednih platformno agnostičnih ogrodij za upodabljanje (Hachisuka, 2015; Congote et al., 2011), toda te rešitve se osredotočajo le na upodabljanje neprosojnih materialov ali pa implementirajo le primitivne metode za volumetrično upodabljanje. Kolikor nam je znano, v času nastanka tega dela še ni obstajalo ogrodje, ki bi temeljilo na novejšem standardu WebGL 2.0.

V tem delu združujemo metodo MCRT s standardom WebGL 2.0 v platformno agnostični aplikaciji za fotorealistično upodabljanje volumetričnih podatkov. Aplikacija je zasnovana z mislijo na interaktivnost v realnem času ter omogoča neomejen nadzor uporabnika nad prenosno funkcijo, kamero in osvetlitvijo. Zgrajena je modularno, kar omogoča enostavno razširitev in prilagoditev, deluje pa na vseh namiznih in mobilnih napravah, ki podpirajo standard WebGL 2.0 z razširitvijo za pisanje v teksture v formatu s plavajočo vejico.

2 PREGLED PODROČJA

Pregledni članek (Jönsson et al., 2014) opisuje široko paleto osvetlitvenih modelov, ki se uporabljajo v sodobnih aplikacijah za upodabljanje volumetričnih podatkov. Novejša publikacija (Fong et al., 2017) je uporabljena kot referenca za najpomembnejše in najobetavnejše metode v industriji. Nedavne objave s področja neposrednega upodabljanja volumetričnih podatkov z uporabo grafične strojne opreme (Balsa Rodríguez et al., 2014; Beyer et al., 2015) omogočajo natančen vpogled v stanje tehnologije ter razkrivajo smer razvoja področja. Omenjene publikacije so bile primarni vir za delo, predstavljeno v tem besedilu.

Najzgodnejše metode za upodabljanje volumetričnih podatkov, na katerih temeljijo tudi novejši prispevki, so še vedno v široki uporabi predvsem zaradi enostavnosti implementacij ter samih algoritmov. Natančne opise in primerjave nekaterih najbolj pogostih pristopov najdemo v Max (1995). Te metode imajo številne slabosti, tako metodološke (npr. pristranskost) kot tudi zaznavne (ne ponujajo dovolj opornikov za pomoč pri zaznavanju oblik in globine). Raziskave so se dolga leta osredotočale na opornike, ki so posledica osvetlitvenih učinkov, npr. sence, in ambientnega zastiranja, saj bistveno izboljšajo uporabnikovo zaznavanje (Lindemann in Ropinski, 2011). Publikacija Lesar et al. (2015) naslavlja isti problem v kontekstu upodabljanja volumetričnih angiografskih slik.

Najenostavnejši pristop za vpeljavo globalne osvetlitve v volumetrično upodabljanje zahteva izračun osvetlitvenega volumna, ki ga nato vzorčimo med sledenjem žarkom, kot je opisano v Behreuns in Ratering (1998). Ta tehnika je časovno in pomnilniško potratna, toda služila je kot podlaga za razvoj številnih naprednejših tehnik. Zelo popularna tehnika, izpeljana iz osvetlitvenega volumna, imenovana globoke sence in predstavljena v Lokovic in Veach (2000), je bila prilagojena za različne scenarije, vključno za uporabo v sledenju žarkom na grafični strojni opremi, kot opisujejo Hadwiger et al. (2006). Čeprav so globoke sence hiter in priročen način za upodabljanje senc v volumetričnih podatkih, je njihova glavna slabost v tem, da so primerne le za simulacijo neposredne osvetlitve z enim točkastim izvorom svetlobe.

Pogosto dobimo boljše rezultate, če namesto množice izvorov svetlobe senčenje obravnavamo kot posledico zastiranja bližnjih predmetov, kar grobo ustreza globalni osvetlitvi v lokalni okolici. Družina takih metod se imenuje ambientno zastiranje (angl. ambient occlusion, AO), vsem pa je skupna aproksimacija globalne osvetlitve z obravnavanjem lokalne okolice, kot je opisano v izvirnem članku Zhukov et al. (1998). Obstajajo številne izpeljanke te metode, npr. aproksimacija v koordinatnem sistemu zaslona (angl. screen-space ambient occlusion), predstavljena v Ritschel et al. (2009), ter prilagoditve za izvajanje na grafični strojni opremi.

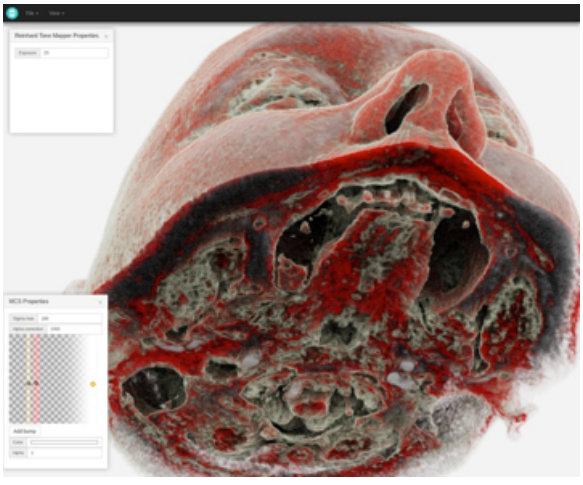
Opisane metode izboljšajo prostorsko predstavo z vizualnimi oporniki, ki temeljijo na globalni osvetlitvi, čeprav so ti lahko le približni. V tem delu poudarjamo pomen natančne globalne osvetlitve, kot je navedeno v Banks in Beason (2009), ki pravi, da je uporaba naprednih metod za upodabljanje volumetričnih podatkov tako rekoč nična. Kroes et al. (2012) rešuje to težavo tako, da predlaga splošno metodo, ki jo lahko enostavno integriramo v aplikacije. S tem skušajo avtorji stimulirati izboljšave na tem področju. Predstavljena rešitev dovoljuje uporabo poljubnega števila luči ter poljubne prenosne funkcije, hkrati pa poenoti obravnavo transparentnih medijev in trdnih materialov. Z enotno obravnavo se poenostavi tudi sam algoritem, kar prispeva k hitrosti izvajanja aplikacije in posledično k interaktivnosti. Simulaciji Monte Carlo v njihovem delu in aplikaciji, opisani v

tem prispevku, uporabljata model enojnega sipanja, prepustnost medija oz. dolžina proste poti pa se vrednoti po Woodcockovi metodi. Hitrost upodabljanja je neposredno povezana s hitrostjo vrednotenja prepustnosti medija oz. dolžine proste poti, zato je Woodcockova metoda deležna tudi nadaljnjih izboljšav (Szirmay-Kalos et al., 2011; Novák et al., 2014).

Aplikacija, predstavljena v tem delu je implementirana z jezikom JavaScript in programskim vmesnikom WebGL 2.0. Nekaj poskusov implementacije metod Monte Carlo za upodabljanje volumetričnih podatkov z WebGL že obstaja. Congote et al. (2011) so pokazali, da so spletne tehnologije že dovolj zrele za realnočasovno in interaktivno upodabljanje volumetričnih podatkov. Predstavitvena aplikacija je tudi platformno agnostična. Upodabljanje na mobilnih napravah je zaradi omejitev strojne opreme precej težje, toda nekaj raziskav se je tega že lotilo. Nekatere izkoriščajo lokalno računsko moč (Mobeen in Feng, 2012; Schiewe et al., 2015), ostale pa težje izračune odložijo na strežnik (Lee in Nam, 2014). Vse implementacije temeljijo na starejšem standardu WebGL 1.0 in razširitvi za pisanje v več okvirjev, implementirajo pa vizualizacijske metode iz dela (Max, 1995), ki ne ponujajo nobenih vizualnih opornikov za lažjo prostorsko predstavo. Obstaja implementacija metode Monte Carlo z jezikom GLSL (Hachisuka, 2015). Delo opisuje rešitev, ki naj bi bila pripravljena za izvajanje v spletnih brskalnikih, toda tega avtorji niso poskusili. Čeprav je to delo dobra referenca za stohastične metode v jeziku GLSL, je upodabljalnik primeren le za izris trdnih modelov, ne pa tudi prosojnih. Obstaja tudi delujoča implementacija upodabljalnika trdnih modelov s spletnimi tehnologijami, dostopna na `madebyevan.com/webgl-path-tracing`.

3 METODE

V tem delu združujemo sodobne stohastične metode za upodabljanje, ki jih implementiramo v spletnih tehnologijah v obliki platformno agnostične aplikacije za interaktivno raziskovanje volumetričnih podatkov. Uporaba stohastičnega pristopa omogoča progresivno upodabljanje fotorealističnih slik v realnem času, kot je prikazano na sliki 1. Aplikacija je razvita s spletnima tehnologijama JavaScript in HTML 5, do grafične strojne opreme pa dostopa prek elementa `<canvas>` in programskega vmesnika WebGL 2.0. Zunanji knjižnici Bootstrap in jQuery sta uporabljeni za razvoj grafičnega uporabniškega vmesnika, ki deluje tako na mobilnih kot namiznih napravah. Za popoln nadzor nad učinkovitostjo izrabe grafične strojne opreme je programski vmesnik WebGL uporabljen brez zunanjih knjižnic. Slika 2 prikazuje zaslonski posnetek aplikacije na namizni napravi.



Slika 2: **Prikaz delovanja razvite aplikacije v namizni različici brskalnika Google Chrome**

Del aplikacije, namenjen upodabljanju, upravlja kontekst upodabljanja, kar vključuje kontekst WebGL skupaj z vsemi podatki, ki so neodvisni od kakršnih koli specifik upodabljanja, med drugim volumetrične podatke in prenosno funkcijo. Do teh podatkov dostopajo različne stopnje cevovoda upodabljanja, opisanega v naslednjem razdelku. Logika upodabljanja je razdeljena na dve zaporedni stopnji: volumetrično upodabljanje in naknadno procesiranje. Prva stopnja progresivno računa sevalnost (angl. radiance) v formatu HDR, druga pa jo pretvori v barvno informacijo v formatu LDR, primernem za izris na zaslonu. S tovrstnim modularnim pristopom je omogočeno naknadno procesiranje v več stopnjah, volumetrično upodabljanje pa lahko vrača fizikalno pravilne rezultate. Medpomnilniki prve stopnje cevovoda morajo zato hraniti podatke v formatu HDR. Poudarjamo, da osnovni standard WebGL 1.0 ne podpira tekstur v formatu s plavajočo vejico, a je ta funkcionalnost dostopna kot standardna razširitev `OES_texture_float`, ki jo lahko vključimo v času izvajanja aplikacije. WebGL 2.0 podpira formate s plavajočo vejico v osnovnem profilu, toda upodabljanje v okvir tovrstnega formata je še vedno dostopno le prek standardne razširitve `EXT_color_buffer_float`, na katero se predstavljena aplikacija

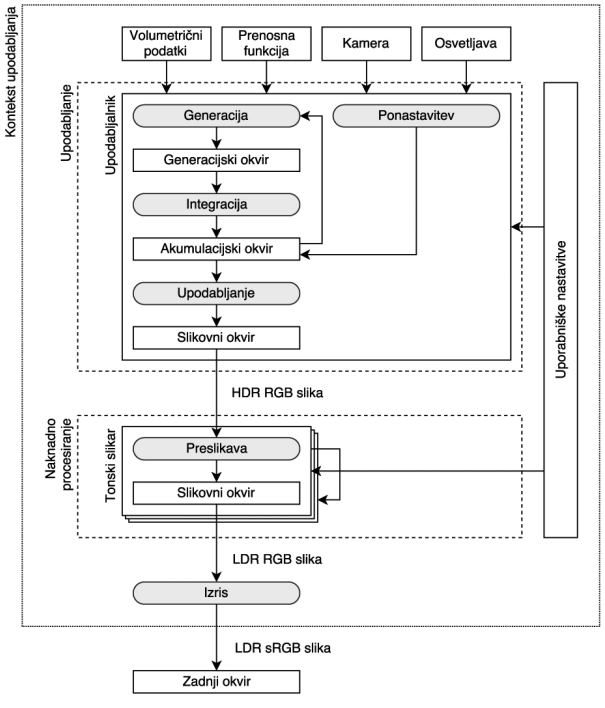
močno zanaša. Tako volumetrični podatki kot prenosna funkcija so shranjeni v formatu s plavajočo vejico v grafičnem pomnilniku.

3.1 Cevovod upodabljanja

Cevovod upodabljanja je shematično prikazan na sliki 3. Prva stopnja cevovoda je volumetrično upodabljanje, ki ga predstavlja abstraktni volumetrični upodabljalnik. Koraki volumetričnega upodabljanja so zasnovani z ozirom na interaktivnost, zato so še posebno primerni za uporabo z metodo MCRT. Proces upodabljanja progresivno izboljšuje izrisano sliko, zato ta stopnja cevovoda vključuje vzvratne povezave. Druga stopnja cevovoda je naknadno procesiranje, ki je sestavljeno iz poljubnega števila zaporednih procesorskih enot. Zadnji korak naknadnega procesiranja je predstavljen z abstraktnim tonskim slikarjem, ki zmanjša dinamični razpon slike in vrne rezultat, primeren za izris na zaslonu.

Stopnjo upodabljanja sestavljajo štirje koraki:

1. generacija,
2. integracija,
3. upodabljanje,
4. (ponastavitev).

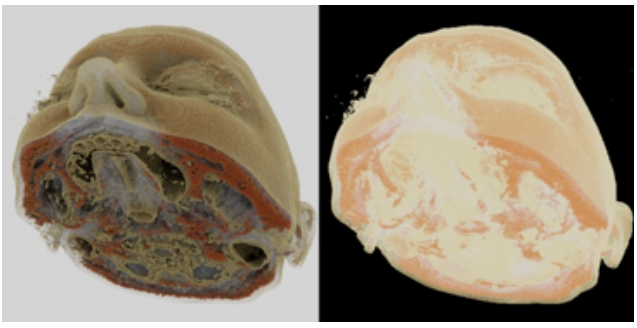


Slika 3: **Cevovod za upodabljanje. Volumetrične podatke in prenosne funkcije skupaj z ostalimi parametri scene posredujemo v stopnjo za volumetrično upodabljanje, v kateri se konstruira slika v formatu HDR RGB, ki jo posredujemo v stopnjo naknadnega procesiranja za pretvorbo slike v format LDR RGB.**

V vsaki iteraciji se zaporedoma izvedejo prvi trije koraki. V koraku generacije se izračuna nov približek slikovnih podatkov, ki se nato v koraku integracije skupaj s podatki, pridobljenimi v prejšnjih iteracijah, akumulira v novem, boljšem približku. Akumulirani podatki se v zadnjem koraku, koraku upodabljanja, pretvorijo v barvno informacijo slike. V koraku generacije se najprej generirajo poti svetlobnih žarkov, nato pa propagirajo skozi medij. Zaradi številnih dostopov do volumetričnih podatkov in prenosne funkcije je ta korak običajno najbolj časovno zahteven, zato mora biti skrbno implementiran, da omogoča izvajanje v realnem času. Ponastavitev upodabljalnika in akumuliranih podatkov se izvede ob spreminjanju podatkov ali parametrov, ki se uporabljajo v procesu generacije, denimo kamere, osvetljave, volumetričnih podatkov ali prenosne funkcije.

Vsak konkretni tip upodabljalnika mora implementirati vmesnik abstraktnega upodabljalnika z opisanimi koraki ter zagotoviti parametre za ustvarjanje generacijskega, akumulacijskega in slikovnega okvirja, ki se uporabljajo znotraj konteksta WebGL. Poudarimo, da je naloga programerja, ki implementira vmesnik upodabljalnika, da določi pomen in format podatkov znotraj generacijskega in akumulacijskega okvirja, podatki v slikovnem okvirju pa morajo biti barve v formatu HDR RGB, ki jih nato pošljemo naprej v stopnjo naknadnega procesiranja.

Opisani abstraktni koncept lahko uporabimo brez prilagoditev za upodabljanje največje intenzitete, emisijsko-absorpcijskega modela, nivojskih ploskev ali sledenja potem. Za projekcijo največje intenzitete denimo shranjujemo trenutne največje vrednosti v akumulacijskem okvirju, v koraku generacije pa vzorčimo volumetrične podatke le enkrat na iteracijo za vsako slikovno točko. V vsaki iteraciji integracijskega koraka nato obdržimo največjo vrednost med generacijskim in akumulacijskim okvirom. Upodabljalnik nivojskih ploskev v akumulacijskem okvirju hrani trenutno razdaljo do najbližje nivojske ploskve in njeno normalo v tej točki, v koraku upodabljanja pa te informacije uporabi za lokalno osvetlitev na način odloženega upodabljanja. Implementacija upodabljalnika MCRT v akumulacijskem okvirju hrani ocene sevalnosti za vsako slikovno točko, korak upodabljanja pa izkorišča za odstranjevanje šuma ali drugo filtriranje. Vsi omenjeni upodabljalniki so vključeni v programskem ogrodju, opisanem v tem delu.



Slika 4: **Primerjava upodobitve istih podatkov z metodo sledenja potem (levo) in emisijsko-absorpcijsko metodo (desno)**

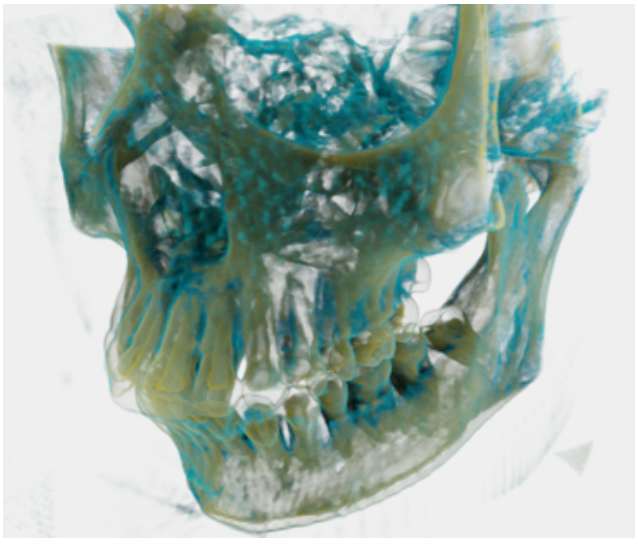
Slika v formatu HDR RGB, ki je produkt stopnje volumetričnega upodabljanja, vstopi v stopnjo naknadnega procesiranja. Ta je sestavljena iz zaporedja korakov, vključno s tonskim slikanjem in gama korekcijo, ki proizvedejo vmesne slike. Zadnji korak te stopnje mora proizvesti sliko v formatu LDR RGB, ki se na koncu izriše na uporabnikov zaslon prek brskalnikovega elementa HTML `<canvas>`. V našem ogrodju stopnjo naknadnega procesiranja izkoriščamo za izvajanje enostavnega Reinhardovega tonskega slikarja z nastavljivo ekspozicijo.

3.2 Implementacija upodabljalnika Monte Carlo

V aplikaciji, predstavljeni v tem delu, smo implementirali upodabljalnik največje intenzitete in upodabljalnik nivojskih ploskev, da lahko primerjamo njuno časovno učinkovitost in izris z upodabljalnikom MCRT. Obe implementaciji upodabljalnikov uporabljata stohastičen pristop, kot je opisano v prejšnjem razdelku. Upodabljalnik MCRT sledi istemu pristopu in je zgrajen po vzoru implementacije (Kroes et al., 2012), tako da uporablja Woodcockovo sledenje (Szirmay-Kalos et al., 2011; Woodcock et al., 1965) za izračun točke sipanja svetlobe in s tem poenostavi senčenje. V primerjavi s popularno Riemannovo vsoto je ta pristop nepristranski, poenostavi pa se tudi uporaba večkratnega prioritetnega vzorčenja (angl. multiple importance sampling, MIS) za usmerjanje smeri sipanja v izvore svetlobe.

Parametre kamere uporabimo za izračun parametrizacije žarka za vsako slikovno točko, ki nato skupaj s transformacijo volumetričnega objekta služi za izračun parametrizacijskega obsega. Žarke, ki ne sekajo volumetričnega objekta, uporabimo le za vzorčenje slike okolja. Preostali žarki se propagirajo skozi medij z metodo Woodcockoveka sledenja in s tem določijo točko sipanja ter izvor svetlobe. Za oceno prepustnosti uporabljamo cenilko, kot opisujejo Novák et al. (2014). V tem delu smo implementirali koherentno propagacijo vseh žarkov skozi volumen, kar izboljša uporabo predpomnilnika grafične kartice, saj v vsakem koraku dostopamo do podobnih lokacij v volumnu. Če točke sipanja z Woodcockovo metodo ne najdemo, žarek potuje neovirano skozi medij, zato vzorčimo sliko okolja. V nasprotnem primeru vzorčimo fazno funkcijo, da poleg točke določimo še smer sipanja, ali pa žarek usmerimo v stohastično izbran izvor svetlobe. Od tu dalje ponovno uporabimo metodo Woodcockovega sledenja za oceno prepustnosti medija na poti od točke sipanja do izvora svetlobe. Opisani postopek konvergira k rešitvi enačbe upodabljanja z enkratnim sipanjem. Poleg tega so vsi parametri upodabljanja, vključno s položajem kamere, osvetlitvijo in prenosno funkcijo, popolnoma interaktivni in ob spremembah zahtevajo le ponastavitev cevovoda. Slika 5 prikazuje različne prenosne funkcije, interaktivno prilagojene na istih podatkih, slika 6 pa prikazuje izris s prosojnimi materiali.

Slika 5: **Uporaba različnih prenosnih funkcij pri izrisu različnih podatkov**



Slika 6: **Prikaz upodabljanja s prosojnimi materiali**

Vsa interakcija medija s svetlobo se računa v formatu s plavajočo vejico v visoki ločljivosti, kar v senčilnikih omogočimo z direktivo `precision highp float`. Rezultati se shranjujejo v okvirjih v formatu s plavajočo vejico, kar omogočimo prek standardne razširitve `EXT_color_buffer_float`, kar nam omogoča neposredno hranjenje sevalnosti brez opaznih numeričnih napak pri izgubi natančnosti. Iz istega razloga sta v takem formatu shranjeni tudi prenosna funkcija in slika okolja.

Slika volumetričnega upodabljalnika, ki je sestavljena iz ocen sevalnosti za vse slikovne točke, nato vstopa v stopnjo naknadnega procesiranja, v kateri se prilagodi za boljšo zaznavo ter izris na zaslonu.

3.3 Naknadno procesiranje

Programsko ogrodje dovoljuje poljubno število korakov naknadnega procesiranja, ki sliko volumetričnega upodabljalnika preoblikujejo na poljuben način in s tem izboljšajo zaznavanje. Vhod te stopnje cevovoda je izhod stopnje volumetričnega upodabljanja, torej slika v formatu HDR RGB. Koraki te stopnje hranijo vmesne rezultate v poljubnem formatu, toda zadnji korak mora vedno ustvariti sliko v formatu LDR RGB, primerno za izris na zaslonu. V tej stopnji lahko izvajamo tudi različno filtriranje, denimo odstranjevanje šuma, ki pomaga pri konvergenci končne slike. V trenutni implementaciji cevovoda uporabljamo tonsko slikanje po metodi Reinhard et al. (2002), da zmanjšamo dinamični razpon slike. Metoda, opisana v Kroes et al. (2012), uporablja še dodaten korak za gama korekcijo.

4 VREDNOTENJE IN REZULTATI

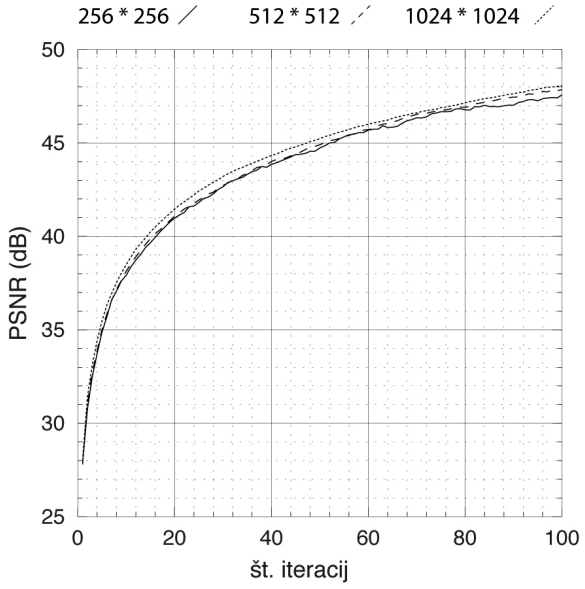
Aplikacijo, opisano v tem delu (dostopno na `https://github.com/terier/vpt`), smo ovrednotili z ozirom na podporo brskalnikov, hitrost konvergence ter hitrost izvajanja. V tem delu se ne ukvarjamo s pretakanjem podatkov prek omrežja ali iz sekundarne shrambe, zato smo ob merjenju hitrosti izrisa vse podatke v celoti hranili v grafičnem pomnilniku. Aplikacijo smo ovrednotili na dveh različnih napravah: na prenosnem računalniku z integrirano grafično kartico Intel HD graphics 530 (z računsko močjo 403,2 GFLOPS) in na pametnem telefonu s čipom Adreno 430 (z računsko močjo 388,8 GFLOPS).

4.1 Podpora spletnih brskalnikov

Podporo smo analizirali s pomočjo statističnih podatkov s spletne strani `https://www.webglstats.com`, ki zbira podatke o implementacijah standardov WebGL na uporabniških napravah z različnih spletnih strani. Standard WebGL 1.0 je v času pisanja tega dela podprt na 98 odstotkih naprav, medtem ko je WebGL 2.0 podprt na 68 odstotkih naprav. Poleg podpore standarda smo preverili še podporo dveh standardnih razširitev, potrebnih za delovanje aplikacije: `EXT_color_buffer_float` za hranjenje slik v formatu s plavajočo vejico, ki je podprta na 93 odstotkih naprav, ter `WEBGL_lose_context` za zaznavanje izgube ali ukinitve konteksta WebGL s strani spletnega brskalnika, ki je podprta na 93 odstotkih naprav. Za merjenje hitrosti izvajanja enega prehoda cevovoda standard WebGL 2.0 ponuja razširitev `EXT_disjoint_timer_query_webgl2`, ki je na voljo na 8 odstotkih naprav, toda v našem primeru je nismo mogli uporabiti, saj je izbrane naprave ne podpirajo. Kot alternativo smo uporabili kombinacijo funkcij `performance.now()` in `gl.finish()`.

4.2 Hitrost konvergence

Hitrost konvergence naše implementacije smo ovrednotili z izračunom največjega razmerja signal – šum (angl. peak signal-to-noise ratio, PSNR) za okvirje različnih velikosti v odvisnosti od števila iteracij cevovoda. Uporabili smo tri različne velikosti okvirja: 256 × 256, 512 × 512 in 1024 × 1024, vse na volumetričnih podatkih dimenzij 128 × 128 × 128. Pri vseh volumnih smo uporabili izotropno fazno funkcijo. Za potrebe izračuna vrednosti PSNR smo kot referenčno sliko uporabili rezultat po 1000 iteracijah cevovoda. Rezultati so prikazani na sliki 6. Zaradi velikega števila slikovnih pik so vrednosti PSNR primerljive med različnimi velikostmi okvirja. Velikost okvirja neposredno vpliva na čas izvajanja posamezne iteracije, toda ne na hitrost konvergence. Vrednosti PSNR zaradi lokalnosti izračuna ne moremo uporabiti za vrednotenje prostorske natančnosti slike. S slike je razvidno, da metoda konvergira hitro in ustvari prepoznavno sliko v 20 do 30 iteracijah, kot posledica metode Monte Carlo pa so donosi vedno manjši, s teoretično hitrostjo konvergence $O(n^{-\frac{1}{2}})$.



Slika 7: **PSNR vrednosti, odvisne od števila iteracij za različne velikosti slikovnega medpomnilnika: 256 × 256 (polna črta), 512 × 512 (črtkana črta) in 1024 × 1024 (pikčasta črta) za velikosti volumna 128 × 128 × 128**

4.3 Hitrost upodabljanja

Iterativna izvedba cevovoda temelji na funkciji spletnega brskalnika `requestAnimationFrame`, ki je sinhronizirana na frekvenco osveževanja zaslona, torej v običajnih okoliščinah 60 iteracij na sekundo. Omejitvi smo se izognili s prilagajanjem senčilnikov, da smo lahko znotraj enega prehoda cevovoda izračunali več iteracij. Po drugi strani se ne moremo izogniti blokiranju glavne niti brskalnika, če dolgotrajne iteracije presežejo osveževalni cikel, saj trenutno ne obstaja nobena standardna metoda za ustvarjanje konteksta WebGL v lastni niti. Ustvarjanje konteksta v spletnih delavcih (angl. web workers) prek vmesnika `OffscreenCanvas` ni standardizirano, poleg tega pa je večinoma nepodprto na mobilnih napravah. Število iteracij za generiranje enega okvirja je v naši aplikaciji nastavljivo, tako da lahko vzdržujemo interaktivnost tudi na mobilnih napravah. Hitrost upodabljanja smo ovrednotili z merjenjem števila iteracij cevovoda na sekundo z različnimi dimenzijami okvirja in volumetričnih podatkov. Meritve, opravljene v brskalnikih Google Chrome za Linux 56 (prenosni računalnik) in Google Chrome za Android 58 (pametni telefon), so prikazane v tabeli 1. Iz meritev je razvidno, da je naša implementacija dovolj hitra za interaktivno uporabo tudi na mobilnih napravah. Za dovolj majhne velikosti okvirja in volumna smo naleteli na omejitve funkcije `requestAnimationFrame`, zato so najvišje vrednosti omejene na 60 iteracij na sekundo.

Tabela 1: **Povprečno število iteracij na sekundo cevovoda v odvisnosti od velikosti volumna pri fiksni velikosti slikovnega medpomnilnika 512 × 512 (levo) in v odvisnosti od velikosti slikovnega medpomnilnika pri fiksni velikosti volumna 128 × 128 × 128 (desno)**

Naprava	64^3	128^3	256^3	256^2	512^2	1024^2
Pametni telefon	36	15	11	29	15	10
Prenosnik	60	36	22	60	36	18

5 SKLEPI

V prispevku smo predstavili realnočasovno, interaktivno in platformno agnostično aplikacijo za neposredno upodabljanje volumetričnih podatkov, ki omogoča fotorealistično upodabljanje na namiznih in mobilnih napravah. Temelji na reševanju fizikalnega modela za transport svetlobe v heterogenem mediju s stohastičnim pristopom na grafični strojni opremi. Aplikacija teče v spletnih brskalnikih, ki podpirajo standard WebGL 2.0 z razširitvami za formate s plavajočo vejico. Kolikor nam je znano, taka aplikacija še ne obstaja. V aplikaciji je implementirano razširljivo ogrodje, zgrajeno s tehnologijo WebGL 2.0 in z jezikom JavaScript, kar omogoča modularen razvoj metod upodabljanja. Naš predlog rešitve je zmožen progresivnega in interaktivnega upodabljanja brez predprocesiranja vhodnih podatkov in ponovne obdelave po spremembah osvetlitve in prenosne funkcije.

Prihodnje izboljšave vključujejo optimizacijo ogrodja ter z njim zgrajenih upodabljalnikov. Fong et al. (2017) ponuja obširen pregled trenutnega stanja tehnologije, ki se uporablja v produkcijskih okoljih. Služila bo kot neprecenljiv vir za naknadne izboljšave, s katerimi bomo opisane ideje prenesli v spletno okolje. Prvotne izboljšave bodo vključevale večkratno prioritetno vzorčenje, opisano v Kroes et al. (2012), in pohitritev sledenja žarkom s sledenjem rezidualov, opisano v Novák et al. (2014). Takoj zatem so na vrsti boljši modeli kamer z optimizacijo postopka generiranja žarkov. Poleg izboljšav metod upodabljanja je nujne prenove potreben tudi uporabniški vmesnik. Prav tako naša aplikacija trenutno ne podpira upodabljanja večjih volumetričnih podatkov, ki jih ne moremo v celoti zapisati v pomnilnik. Težavo rešujejo številne metode za pretakanje podatkov z diska ali prek omrežja.

Verjamemo, da smo z našo aplikacijo naredili pravi korak v smeri platformno agnostičnega razvoja na

področju upodabljanja volumetričnih podatkov. S tem želimo celotno področje in predvsem najnovejše metode približati uporabnikom brez težavnih namestitvenih postopkov ali omejitev platform. Upamo, da bo industrija sprejela ta prispevek kot uporabno alternativo obstoječim aplikacijam.

6 LITERATURA

[1] Balsa Rodríguez, M., Gobbetti, E., Iglesias Guitián, J. A., Makhinya, M., Marton, F., Pajarola, R., in Suter, S. K. (September 2014). State-of-the-Art in Compressed GPU-Based Direct Volume Rendering. *Computer Graphics Forum*, 33(6), 77–100. https://doi.org/10.1111/cgf.12280.

[2] Banks, D., in Beason, K. (November 2009). Decoupling Illumination from Isosurface Generation Using 4D Light Transport. *IEEE Transactions on Visualization and Computer Graphics* 15(6), 1595–1602. https://doi.org/10.1109/TVCG.2009.137.

[3] Barsky, B. A., Horn, D. R., Klein, S. A., Pang, J. A., in Yu, M. (2003). Camera Models and Optical Systems Used in Computer Graphics: Part II, Image-Based Techniques. *Computational Science and Its Applications — ICCSA 2003*. 256–265. https://doi.org/10.1007/3-540-44842-X_27.

[4] Behreuns, U. in Ratering, R. (1998). Adding shadows to a texture-based volume renderer. *Proceedings of the 1998 IEEE symposium on Volume visualization – VVS '98*, ACM Press, 39–46. https://doi.org/10.1145/288126.288149.

[5] Beyer, J., Hadwiger, M. in Pfister, H. (Maj 2015). State-of-the--Art in GPU-Based Large-Scale Volume Visualization. *Computer Graphics Forum* 34(8), 13–37. https://doi.org/10.1111/cgf.12605.

[6] Congote, J., Segura, A., Kabongo, L., Moreno, A., Posada, J. in Ruiz, O. (2011). Interactive visualization of volumetric data with WebGL in real-time. *Proceedings of the 16th International Conference on 3D Web Technology – Web3D '11*, ACM Press, 137–146. https://doi.org/10.1145/2010425.2010449.

[7] Davidovič, T., Křivánek, J., Hašan, M. in Slusallek, P. (Junij 2014). Progressive Light Transport Simulation on the GPU. *ACM Transactions on Graphics*, 33(3), 1–19. https://doi.org/10.1145/2602144.

[8] Fong, J., Wrenninge, M., Kulla, C. in Habel, R.: Production volume rendering. *ACM SIGGRAPH 2017 Courses on – SIGGRAPH '17*, ACM Press, 1–79. https://doi.org/10.1145/3084873.3084907.

[9] Hachisuka, T. (Maj 2015). Implementing a Photorealistic Rendering System using GLSL. *arXiv,* 1–4. https://arxiv.org/abs/1505.06022.

[10] Hadwiger, M., Kratz, A., Sigg, C. in Bühler, K. (2006). GPU--accelerated deep shadow maps for direct volume rendering. *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware – GH '06*, ACM Press, 49–52. https://doi.org/10.1145/1283900.1283908.

[11] Jönsson, D., Sundén, E., Ynnerman, A. in Ropinski, T. (Februar 2014). A Survey of Volumetric Illumination Techniques for Interactive Volume Rendering. *Computer Graphics Forum,* 33(1), 27–51. https://doi.org/10.1111/cgf.12252.

[12] Kajiya, J. T. (Avgust 1986). The rendering equation. *SIGGRAPH Comput. Graph*. 20(4), 143–150. https://doi.org/10.1145/15886.15902.

[13] Kolb, C., Mitchell, D. in Hanrahan, P. (1995). A Realistic Camera Model for Computer Graphics. *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 317–324. https://doi.org/10.1145218380.218463.

[14] Kroes, T., Post, F. H. in Botha, C. P. (Julij 2012). Exposure Render: An Interactive Photo-Realistic Volume Rendering Framework. *PLoS ONE*. 7(7). https://doi.org/10. 1371/journal.pone.0038586.

[15] Lafortune, E. P. in Willems, Y. D. (1996). Rendering Participating Media with Bidirectional Path Tracing. *Rendering Techniques '96 Eurographics*. https://doi.org/10.1007/978-3-7091-7484-5_10.

[16] Lee, W., Nam, D. (2014). Volume Rendering Architecture of Mobile Medical Image using Cloud Computing. *The Journal of The Institute of Internet, Broadcasting and Communication,* 14(4), 101–106.

[17] Lesar, Ž., Bohak, C. in Marolt, M. (Marec 2015) Evaluation of angiogram visualization methods for fast and reliable aneurysm diagnosis. *Progress in Biomedical Optics and Imaging – Proceedings of SPIE,* 9416, p. 9416. https://doi/10.1117/12.2082179.

[18] Levoy, M. (Julij 1990). Efficient ray tracing of volume data. *ACM Trans. Graph*. 9(3). 245–261. https://doi.org/10.1145/78964.78965.

[19] Lindemann, F. in Ropinski, T. (December 2011). About the Influence of Illumination Models on Image Comprehension in Direct Volume Render- ing. *IEEE Transactions on Visualization and Computer Graphics*, 17(12), 1922–1931. https://doi.org/10.1109/TVCG.2011.161.

[20] Lokovic, T. in Veach, E. (2000). Deep shadow maps. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press, 385–392. https://doi.org/10.1145/344779.344958.

[21] Max, N. (Junij 1995). Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2), 99–108. https://doi.org/10.1109/2945.468400.

[22] Mobeen, M. M. in Feng, L. (2012) Ubiquitous Medical Volume Rendering on Mobile Devices. *International Conference on Information Society (i-Society 2012). IEEE*, 93–98.

[23] Novák, J., Selle, A., Jarosz, W. (November 2014). Residual ratio tracking for estimating attenuation in participating media. *ACM Transactions on Graphics*, 33(6), 1–11. https://doi.org/10.1145/2661229.2661292.

[24] Reinhard, E., Stark, M., Shirley, P. in Ferwerda, J. (Julij 2002). Photographic tone reproduction for digital images. *ACM Transactions on Graphics*, 21(3), 267–276. https://doi.org/10.1145/566654.566575.

[25] Ritschel, T., Grosch, T. in Seidel, H. P. (2009, februar). Approximating dynamic global illumination in image space. In Proceedings of the 2009 symposium on Interactive 3D graphics and games (str. 75–82). ACM.

[26] Schiewe, A. Anstoots, M. in Krüger, J. (2015). State of the Art in Mobile Volume Rendering on iOS Devices. *Eurographics Conference on Visualization (EuroVis), 139–143*. https://doi.org/10.2312/eurovisshort.20151139.

[27] Szirmay-Kalos, L., Tóth, B. in Magdics, M. (Marec 2011). Free Path Sampling in High Resolution Inhomogeneous Participating Media. *Computer Graphics Forum*, 30(1), 85–97. https://doi.org/10.1111/j.1467-8659.2010.01831.x.

[28] Woodcock, E., Murphy, T., Hemmings, P. in Longworth, S. (1965). Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. *Proceedings of Conference on Applications of Computing Methods to Reactor Problems*, 557–579.

[29] Zhukov, S., Iones, A. in Kronin, G. (1998). An ambient light illumination model. *Rendering Techniques '98 Eurographics*, 45–55. https://doi.org/10.1007/978-3-7091-6453-2_5.

Žiga Lesar je asistent in doktorski študent na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Ukvarja se predvsem z računalniško grafiko in visoko zmogljivim računalništvom, raziskuje pa interaktivno upodabljanje medicinskih podatkov s spletnimi tehnologijami. Za svoje delo je leta 2014 prejel univerzitetno Prešernovo nagrado.

▪

Ciril Bohak je asistent na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Njegovi raziskovalni interesi so računalniška grafika, interakcija med človekom in računalnikom, tehnologija iger, poigritev, e-učenje in pridobivanje informacij iz glasbe. Trenutno poučuje pri predmetih Računalniška grafika, Tehnologija iger in Računalniško podprto oblikovanje. Je eden izmed ustanovnih članov slovenske skupnosti HCI.

▪

Matija Marolt je izredni profesor na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Je predstojnik Laboratorija za računalniško grafiko in multimedije. Njegove raziskave so na področjih pridobivanja informacij iz glasbe s poudarkom na semantičnih opisih in razumevanju zvočnih signalov, pridobivanju in organizaciji glasbenih arhivov in interakcije med človekom in računalnikom.