VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Magistrantūra 2 kursas
Kompiuterinio modeliavimo kryptis

Baigiamasis magistro darbas

Spindulių trasavimas ir padalijimo paviršiai

Atliko: Tatjana Kalinka
Tomas Stulpinas

Darbo vadovas: a. Saulius Narkevičius

Recenzentas: dr. Rimvydas Krasauskas

Vilnius
2006

Turinys

Anotacija

Spindulių trasavimas ir padalijimo paviršiai yra svarbūs įrankiai realistiškai atrodantiems vaizdams generuoti.

Padalijimas – tai algoritmas, leidžiantis gauti glotnius paviršius pakartotinai dalijant gardeles.

Spindulių trasavimas yra technologija, kuri remiasi apšvietimo skaičiavimu. Jos dėka galima gauti atspindžius, permatomumą, spindulių lūžimą kertant skaidrius objektus, taipogi realistiškus šešėlius.

Mūsų darbo tikslas buvo suderinti šiuos du metodus, kuriant programinę priemonę, kuri leistų gauti sudėtingų objektų idealiai glotnius aukštos kokybės realistiškus vaizdus. Siekdami to, mes pritaikėme ir tokias kompiuterinės grafikos technologijas, kaip dengimas tekstūromis ir tūrių algebra.

Summary

Ray tracing and subdivision

Ray tracing and subdivision surfaces are important tools for generating realistic looking images.

Subdivision is an algorithmic technique to generate smooth surfaces as a sequence of successively refined polyhedral meshes.

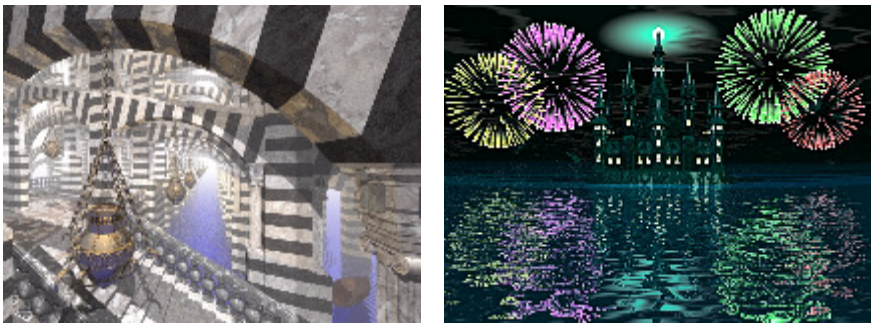
Ray tracing is a technique that performs global calculations of lighting and shading, reflection and transmission of light, casting of shadows, and other effects. The basic idea behind ray tracing is to follow the paths of light rays around a 3-D scene.

Our goal is generation of a high-quality realistic images by combining these two techniques. We also implemented other computer graphics methods designed to increase image realism (Texture Mapping) and to simplify modeling process (Boolean operations with solids).

Įvadas

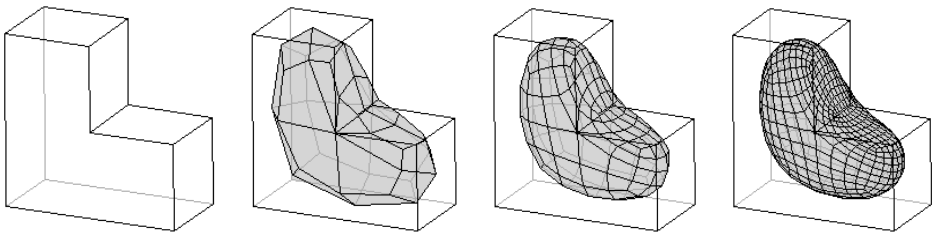
Šiame darbe yra nagrinėjamos kompiuterinės trimatės grafikos metodai ir technologijos, skirtos realistiškai atrodančių modelių konstravimui ir vaizdavimui: spindulių trasavimas, padalijimo paviršiai, tūrių algebros operacijos su kūnais, dengimas tekstūromis.

Spindulių trasavimas yra technologija, skirta realistiniam kompiuterinės grafikos vaizdų kūrimui, ji remiasi apšvietimo skaičiavimu. Jos dėka galima gauti atspindžius (veidrodinius), spindulių lūžimą kertant skaidrius objektus, taipogi realistiškus šešėlius. Pagrindinė spindulių trasavimo idėja yra šviesos spindulių kelių sekimas, pradedant nuo stebėtojo akies.



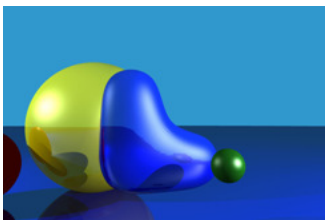
Pav.1: Spindulių trasavimo pagalba gauti vaizdai.

Paviršių padalijimas (surface subdivision) — tai labai patogus ir santykinai paprastas metodas laisvos formos paviršiams gauti. Jo esmė tokia, kad pakartotinai dalijant tinklelius gauname glotnius paviršius. Padalijimo algoritmai yra patrauklūs tuo, kad nereikalauja taisyklingo (reguliaraus) tinklelio (skirtingai nuo Bezier paviršių ir B-splainų).



Pav.2: Padalijimo paviršiaus konstravimas

Dauguma padalijimo paviršių vizualizavimo programų apsiriboja paviršiaus tinklelio vaizdavimu ir baziniu apšvietimo vizualizavimu. Dažnai kyla noras suteikti tokiems paviršiams daugiau fizinių savybių (atspindį, permatomumą). Tuo tikslu galima pritaikyti padalijimo paviršių vaizdavimui spindulių trasavimo technologiją.



Pav. 3: Padalijimo paviršius, pavaizduotas spindulių trasavimo pagalba

Trimatės kompiuterinės grafikos (CAD) sistemos operuoja ne paviršiais, o užpildytais kūnais (solids). Tūrių algebros operacijos — sąjunga, sankirta, skirtumas — yra plačiai taikomos kompiuterinėje grafikoje bei modeliavime.

Pav. 4: Tūrių algebros operacijų su 2 kūnais rezultatai

Deja, tokių operacijų rezultatas negali būti pristatytas kaip padalijimo paviršius — galima tik surasti tam tikrą rezultato aproksimaciją. Mes problemą sprendžiame spindulių trasavimo priemonėmis.

Kita populiari kompiuterinės grafikos operacija — trimačių objektų dengimas tekstūromis (2D vaizdais). Jos pagalba galima imituoti ivairiausias modeliuojamų paviršių charakteristikas.

Pav. 5: Padengtas tekstūra objektas

Mūsų darbo tikslai:

- Susipažinti su padalijimo paviršių konstravimo algoritmais.
- Susipažinti su spindulių trasavimo technologija.
- Realizuoti bazinį spindulių trasavimą padalijimo paviršiams.
- Realizuoti tūrių algebros algoritmus padalijimų paviršiams.
- Realizuoti padalijimų paviršių dengimą tekstūromis.
- Realizuoti judesio šleifo (motion blur) bei kameros fokusavimo gylio efektus.
- Realizuoti adaptyvų padalijimą.
- Parašyti spindulių trasavimo įskiepį studentų sukurtam padalinimo paviršių peržiūros įrankiui.

1. Spindulių trasavimo technologija

1.1. Bendra technologijos apžvalga

Šiuo metu kompiuterinėje trimatėje grafikoje egzistuoja du pagrindiniai virtualios scenos perspektyvaus paišymo (renderingo) metodai.

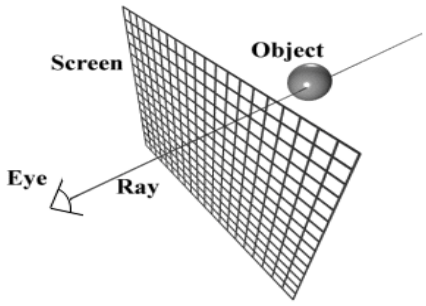
Pirmas metodas yra paremtas nuosekliu kiekvieno scenos primityvo vaizdavimu. Šio metodo esminis trūkumas yra tas, kad kai objektai užstoja vienas kitą, atsiranda ekrano sričių, kurias tenka perpiešti kelis kartus. Nepaisant to, aprašytas metodas (Paint method) dabar yra praktiškai standartas realaus laiko kompiuterinėje grafikoje. Yra sukurta eilė programinių bei aparatinių priemonių skirtų šį metodą pagreitinti.

Egzistuoja kitas metodas trimatiems vaizdams gauti – spindulių trasavimo metodas (Ray tracing). Lyginant su klasikiniais Paint metodais, spindulių trasavimas nusileidžia greičiu. Be to, spindulių trasavimui nėra sukurtų greitintuvų. Pagrindinis spindulių trasavimo metodo privalumas yra didesnis vizualių efektų, pasiekiamų dinamiškai generuojant vaizdą, pasirinkimas [Kas02] . Žemiau yra pateikiamas detalesnis šio metodo aprašymas.

1.2. Algoritmo aprašymas

Spindulių trasavimo metodo esmė yra kiekvieno ekrano taško nuoseklus spalvinimas. Kiekvienam taškui skaičiavimai yra atliekami tik vieną kartą, nepriklausomai nuo to, kiek objektų projekcijų yra tame taške.

Spindulių trasavimo metodas remiasi šviesos spindulio kelio sekimu iš „akies“ (kameros) pro ekraną į sceną. Tokiu būdu mes tarsi grįžtame atgal šviesos spindulio praeitu keliu: nuo galo (akies) iki pradžios (šviesos šaltinio).

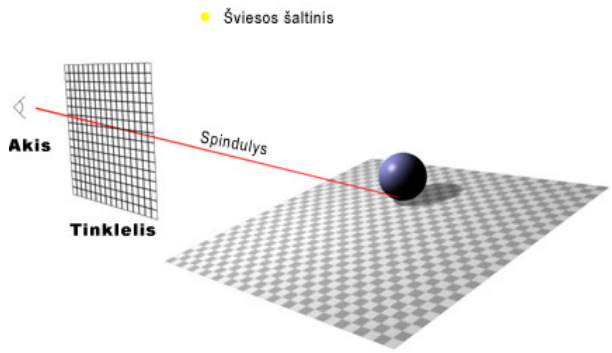


Pav. 6

Spindulių trasavimo bazinis algoritmas savo prigimtimi yra rekursyvus – tai reiškia, kad rezultatas pasiekiamas procesui kelis kartus kartojant patį save [Buc99] .

Algoritmą galima aprašyti pažingsniui:

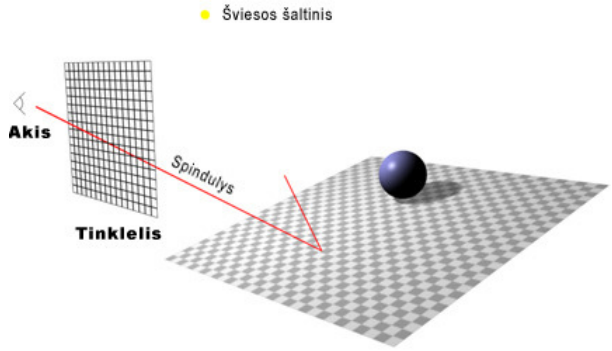
1) Iš akies pro tam tikrą ekrano tašką paleidžiamas spindulys.



Pav. 7. Iš akies paleistas spindulys kerta objektą (sferą)

2) Surandami visi scenos objektai, kuriuos kerta spindulys.

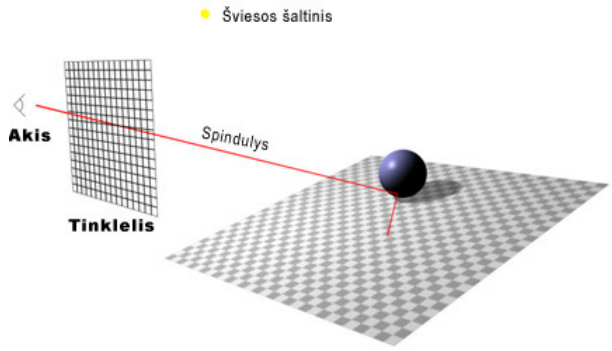
3) Nustatoma, kuri sankirta yra artimiausia. Iš sankirtos taško paleidžiamas spindulys į šviesos šaltinį (shadow ray). Jeigu prieš pasiekiant šviesos šaltinį spindulys kirs kitą scenos objektą, tai pirmos sankirtos taškas yra to objekto šešėlyje.



Pav. 8. Radus spindulio ir objekto susikirtimo tašką, iš sankirtos paleidžiamas spindulys į šviesos šaltinį

Prasideda rekursinis procesas: iš sankirtos taško paleidžiami „antriniai“ (secondary) spinduliai:

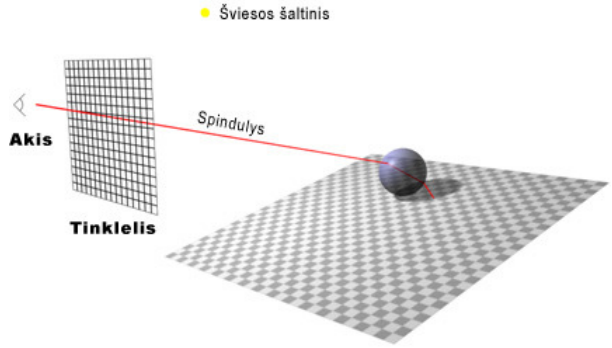
4) Atspindžio spindulys (reflected ray). Jeigu spindulys kirs scenos objektą, apskaičiuojamas lokalus apšviestumo modelis ir jo poveikis pirmos sankirtos taško apšviestumui (contribution from reflected ray).



Pav. 9. Po susikirtimo su sfera paleidžiamas antrinis spindulys

5) Jeigu spindulys kirto permatomą objektą, paleidžiamas lūžio spindulys (transmitted ray). Jeigu lūžio spindulys kerta scenos objektą, kaip ir atspindžio spindulio atveju apskaičiuojamas lokalus apšviestumo modelis.

Atspindžio spinduliai gali generuoti kitus atspindžio spindulius, o tie, savo ruožtu, kitus ir t.t.



Pav. 10. Spinduliui kirtus permatomą objektą, paleidžiamas lūžio spindulys

1.3. Fizikiniai metodo pagrindai

Šviesa, kaip žinoma, turi ir dalelių, judančių tiesine trajektorija, srauto, ir elektromagnetinės bangos savybių. Šviesos intensyvumą nusako bangos amplitudė, o šviesos spalvą — bangos dažnis arba ilgis λ.

Į šviesos spindulį galima žiūrėti kaip į skirtingo dažnio bangų, besiskleidžiančių viena kryptimi, sumą. Bangos, kurios ilgis λ, įtaką nusako funkcija I(λ), vadinama tam tikro šviesos spindulio spektraline charakteristika. Tai, ką žmogaus akis interpretuoja kaip vieną ir tą pačią spalvą, gali būti sukelta daugelio šviesos šaltinių su skirtingomis spektralinėmis charakteristikomis I(λ). Todėl mums užtenka trijų bazinių šviesos šaltinių su λ reikšmėmis, atitinkančiomis raudoną, žalią ir mėlyną spalvas, kad kombinuojant jas gautume beveik visus atspalvius, kuriuos skiria žmogaus akis.

Spindulių trasavimo metodui įgyvendinti taip pat svarbu išsiaiškinti šviesos sklidimo erdvėje savybes. Galima išskirti du pagrindinius atvejus: šviesos sklidimas vienalytėje aplinkoje ir tai,

kaip šviesa elgiasi riboje tarp dviejų aplinkų.

Vienalyte aplinka šviesa sklinda tiesiai, pastoviu greičiu. Šviesos bangų sklidimo vakuume ir toje aplinkoje greičių santykis vadinamas lūžio rodikliu. Dažniausiai šis rodiklis priklauso nuo bangos ilgio λ.

Šviesai sklindant aplinka galimas eksponentinis slopinimas, kurio koeficientas $e^{-\beta l}$, kur l — spindulio praeitas atstumas, o β — slopinimo koeficientas [ŠB95] .

1.4. Pagrindinis spindulių trasavimo modelis

Mūsų modelis turės kelis apribojimus:

- Visi šviesos šaltiniai yra taškiniai
- Nekreipsime dėmesio į lūžusio spindulio krypties priklausomybę nuo bangos ilgio
- Objekto apšviestumą įtakos difuzinis atspindys ir visiškas atspindys.

Tam, kad nustatytume taško P apšviestumą, pirmiausia reikia nustatyti, kiek šviesos jis gauna betarpiškai nuo šviesos šaltinių.

Tam, kad nustatytume taško antrinį apšviestumą, paleisime iš taško P vieną spindulį ta kryptimi, iš kurios atkeliautų atsispindėjęs spindulys, ir vieną spindulį ta kryptimi, iš kurios atkeliautų lūžęs spindulys. Taigi tam, kad nustatytume taško apšviestumą, mums užtenka atsekti sąlyginai nedidelį spindulių skaičių.

Tačiau tokiu būdu atsispindėjusių spindulių, kurie neatitinka visiškam atspindžiui, įtaka yra ignoruojama.

Visiems tokiems dydžiams, kurių poveikį ignoruojame, kompensuoti įvestas foninis apšvietimas: šviesa, kuri skleidžiasi tolygiai iš visų pusių ir kurios neužtemdo šešėliai.

Tada taško P apšviestumas yra nusakomas šia formule [ŠB95] :

$$I(\lambda) = K_a I_a(\lambda) C(\lambda) + K_d C(\lambda) \sum_i I_i(\lambda)(n, l_i) + K_S \sum_i I_i(\lambda) \frac{F_r(\lambda,\theta) D(\alpha) G}{(n, l_i)(n, v)} + K_S I_r(\lambda) F_r(\lambda, \theta_r) e^{-\beta d_r} +$$

$$+ K_t I_t(\lambda)(1 - F_r(\lambda, \theta_t)) e^{-\beta d_t}$$

$I_a(\lambda)$ — foninės šviesos intensyvumas;

$I_a(\lambda)$ — i-to šviesos šaltinio intensyvumas;

$I_a(\lambda)$ — atsispindėjusio spindulio intensyvumas;

$I_a(\lambda)$ — lūžusio spindulio intensyvumas;

$C(\lambda)$ — P taško spalva;

K_a — foninės šviesos koeficientas;

K_d — difuzinės šviesos koeficientas;

K_S — visiško atspindžio koeficientas;

K_t — lūžusio spindulio poveikis;

n — normalė taške P;

l_i — vienetinis vektorius, nukreiptas iš taško P į i-tą šviesos šaltinį;

θ_r — atsispindėjimo kampas;

θ_t — lūžimo kampas;

d_r — atsispindėjusio spindulio praeitas kelias;

d_t — lūžusio spindulio praeitas kelias;

β_r — atsispindėjusio spindulio užgesimo koeficientas;

β_t — lūžusio spindulio užgesimo koeficientas;

$F_r(\lambda,\theta)$ — Frenelio koeficientas, nusakantis atsispindėjusios energijos dalį.

Šis modelis yra pakankamai tikslus fizikine prasme, tačiau praktiškai jį realizuoti yra sunku. Pavyzdžiui, Frenelio koeficientai, nors ir daro įtaką vaizdo realistiškumui, praktikoje naudojami labai retai. Priežastis ta, kad jie reikalauja sudėtingų apskaičiavimų, o antra vertus, mes neturime galimybės tiksliai nustatyti dydžių, įeinančių į formulę, priklausomybę nuo bangos ilgio l.

Todel savo darbe mes panaudojome paprastesnį Vitedo apšviestumo modelį:

$$I(\lambda) = K_a I_a(\lambda) C(\lambda) + K_d C(\lambda) \sum_i I_i(\lambda)(n, l_i) + K_S \sum_i F_{I_i}(\lambda)(n, h_i)^k + K_S I_r(\lambda) e^{-\beta d_r} +$$

$$+ K_t I_t(\lambda) e^{-\beta d_t}$$

1.5. Optimizavimo būdai

Taikant spindulių trasavimo metodą, didžiausią dalį laiko (nuo 75 iki 95 procentų) užima susikirtimo tikrinimas. Nesunku atspėti, kad optimizuojant algoritmą pagrindinis dėmesys yra skiriamas susikirtimo tikrinimų skaičiui sumažinti. Gerus rezultatus duoda hierarchinių struktūrų konstravimas, kai scenos objektai apjungiami į didesnes struktūras, o tos savo ruožtu apjungiamos tarpusavyje ir taip tol, kol nebus sukurtas vienas bendras apvalkalas [Bir02]. Tam yra naudojami tokie metodai, kaip OctTree, BSD ir t.t.

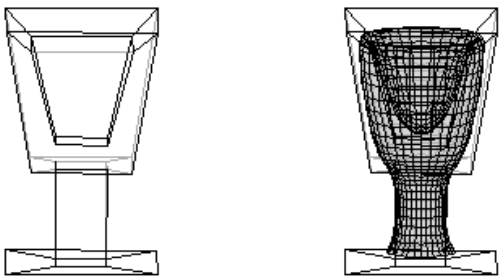
Strukturizuoti galima ne tik objektus, bet ir scenos erdvę. Tam yra taikomi ne tik ortogonalūs metodai (OctTree), bet ir radialiniai (tangentinių segmentų, kubinių segmentų ir pan.). Centru yra laikomas šviesos šaltinis arba kamera [Kas02] .

2. Paviršių padalijimo metodas

2.1. Metodo apžvalga

Kompiuterinėje trimatėje grafikoje didelė reikšmė yra suteikiama paviršių konstravimui. Pastaruoju metu tokie metodai kaip Bezje kreivės (Bezier Curves), splainai (Spline Methods), besiremiantys matematiškomis išraiškomis ir generuojantys tik apribotą kiekį paviršių užleidžia vietą kitiems, lankstesniems, metodams – padalijimo paviršiams.

Padalijimo paviršiai yra apibrėžiami daugiakampių tinkleliu (yra algoritmai, pritaikyti trikampių, keturkampių, šešiakampių tinkleliams).



Pav.11. Smulkinant daugiakampius tinklelis tankėja.

Pritaikius tam tikrą algoritmą smulkiname daugiakampius - tinklelis tankėja (Pav.11).

Pakartotinai taikant algoritmą paviršius darysis vis glotnesnis; padalinimų skaičiui artėjant prie begalybės, ribinis paviršius bus idealaus glotnumo.

Padalijimo paviršiai yra tyrinėjimų objektas jau daugiau kaip 20 metų. Pirmus du algoritmus pasiūlė Katmulas su Klarku (Catmull-Clark) ir Doo su Sabinu (Doo-Sabin) 1978 metais [CC78]. Vėliau buvo atrasti tokie populiarūs padalinimo algoritmai kaip Loop, Butterfly, Kobbelt ir kiti.

Šiandien padalinimo paviršiai yra plačiai naudojami kompiuterinėje grafikoje ir animacijoje, pavyzdžiui, jų pagalba buvo sukonstruoti animacinių filmų “Geri's Game”, “Bug's Life”, “Toy Story 2” personažai. Per pastaruosius metus padalinimo paviršiai buvo integruoti į tokius kompiuterinio trimačius modeliavimo paketus kaip *Pixar's Renderman*, *Alias|Wavefront's Maya 3.0*, *Newtek's Lightwave 3D*, *Nichimen's Mirai*, *Intel Architechture Lab's 3D Software Technologies, 3D Studio MAX* ir kitus.

Padalijimo paviršių populiarumo priežastis yra šios grupės algoritmų privalumai prieš kitus paviršių konstravimo metodus:

- Padalijimo paviršių algoritmus yra lengva įgyvendinti.
- Palijimo paviršių algoritmai yra rekursyvūs, todėl konstruojant juos galima reguliuoti jų detalumo lygį.
- Jeigu dirbant su NURBS paviršiais atskirus paviršiaus gabalus reikėjo specialiais metodais sujungti tarpusavyje, kad jie atrodytų vientisai, padalijimo paviršių atveju to

nereikia.

- Vartotojas gali laisvai redaguoti padalinimo paviršiaus tinklelį, nesirūpindamas apie tai, kaip išsaugoti paviršiaus glotnumą.

Galima išskirti tokius bazinius padalijimo paviršių konstravimo principus:

1. Tankesniam tinkleliui gauti pradinis tinklelis papildomas naujais taškais.
2. Naujų taškų koordinatės yra funkcijos nuo senų taškų koordinačių; senų taškų pozicijos naujame tinklelyje gali pasikeisti.

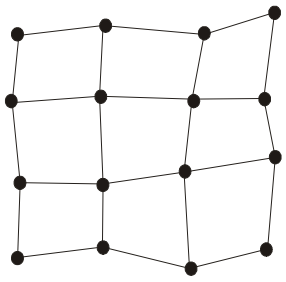
Siekiant geriau suprasti šį paviršių sudarymo metodą panagrinėsime detaliau vieną populiariausių padalijimo būdų – Catmull-Clark algoritmą.

2.2. Catmull-Clark padalijimo algoritmas

Catmull-Clark paviršiai yra bikubinių B-splainų apibendrinimas netaisyklingam tinkleliui. Edas Katmulas ir Džimas Klarkas pastebėjo, kad taisykles, pagal kurias atliekamas kubinių B-splaininių paviršių dalijimas, galima pritaikyti ne tik stačiakampiams, bet ir kitokiems tinkleliams [CC78] .

Prieš pradedant aiškinti Catmull-Clark algoritmą, reikėtų įvesti kelias sąvokas.

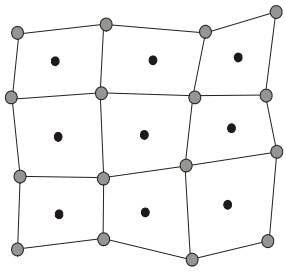
Tarkime, mes turime tinklelį (Pav.12) ir norime jo pagrindu sukonstruoti padalijimo paviršių.



Pav. 12 Pradinis tinklelis

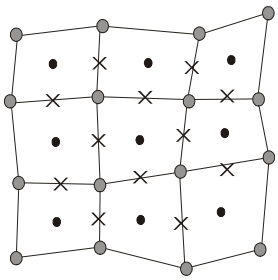
Naujus taškus, kuriuos reikia sukurti, galima pasiskirstyti į tris grupes:

1. Sienų (face) taškai atitinka pradinio tinklelio daugiakampius (Pav.13)



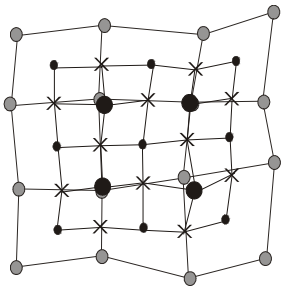
Pav. 13

2. Briaunų (edge) taškai — atitinka pradinio tinklelio briaunas (Pav.14)



Pav. 14

3. Viršūnių (vertixes) taškai — atitinka pradinio tinklelio viršūnes (Pav.15)



Pav. 15

Taigi, Catmull-Clark algoritmą galima suformuluoti taip [Joy96]:

1. Pirmiausia apskaičiuojamos sienų taškų koordinatės: ieškomas sienos kampų koordinačių vidurkis.

2. Apskaičiuojamos briaunų taškų koordinatės: tai bus atitinkamos briaunos vidurio taško ir sienų taškų, atitinkančių gretimas sienas, koordinačių vidurkis.

3. Apskaičiuojami viršūnių taškai, kurie bus lygus Q, 2R ir $\frac{(n-3)}{n}S$ vidurkiui [Joy96].

Čia Q — visų sienų taškų, gretimų pradiniam viršūnių taškui, vidurkis;

R — visų pradinio tinklelio briaunų, išeinančių iš atitinkamos viršūnės, centrų vidurkis;

S — pradinis viršūnių taškas.

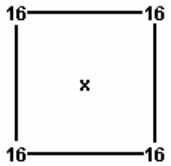
Apskaičiavus taškų koordinates, formuojamas naujas tinklelis:

4. Kiekvienas sienos taškas jungiamas su tos sienos taškais.

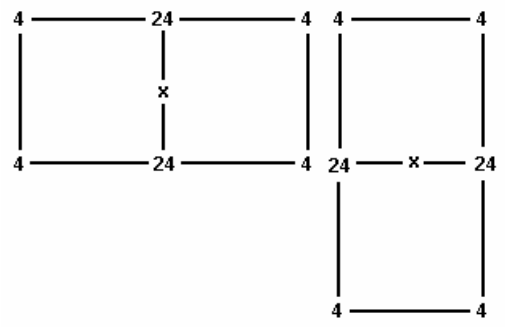
5. Kiekvienas viršūnės taškas jungiamas su briaunų taškais, kurių briaunos išeina iš atitinkamos pradinio tinklelio viršūnės.

Taip vadinami *šablonai* leidžia lengvai suprasti, kaip padalijimo algoritmai veikia. Šablono centre yra naujas taškas. Skaičiai rodo, kaip atitinkami seno tinklelio taškai veikia naują [Sab03].

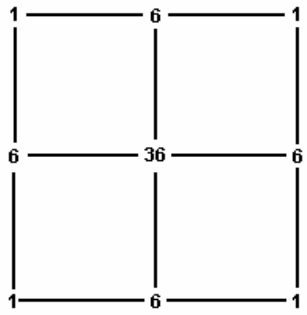
Pav.16-17 pavaizduoti šablonai Catmull-Clark reguliariam tinkleliui (reguliarus tinklelis — toks, kurio visos sienos yra keturkampiai). Šiuo atveju, norint apskaičiuoti naujo taško koordinatę, reikia sudėti seno tinklelio koordinates, padaugintas iš nurodytų skaičių ir padalintas iš 64.



Pav. 16: Šablonas rodo, kiek naujas sienų taškas ima iš pradinio tinklelio taškų



Pav. 17: Šablonas rodo, kiek nauji briaunų taškai ima iš pradinio tinklelio taškų

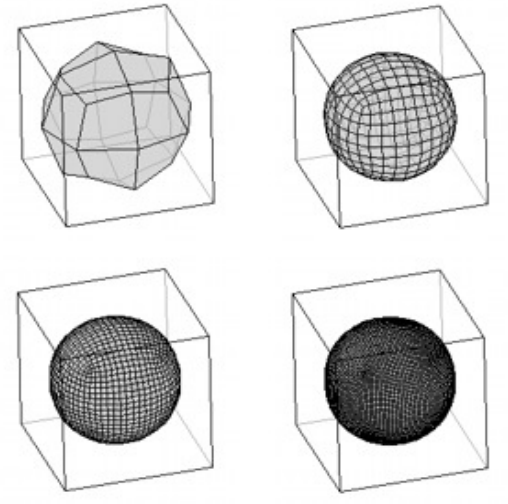


Pav. 18: Šablonas rodo, kiek naujas viršūnių taškas ima iš pradinio tinklelio taškų.

2.3. Catmull-Clark algoritmo taikymo pavyzdys

Geriau įsivaizduoti, kaip veikia Catmull-Clark algoritmas ir apskritai kaip yra konstruojami padalijimo paviršiai, padeda šis pavyzdys.

Catmull-Clark algoritmas taikomas kubui. Paveikslas žemiau iliustruoja, kaip atrodys paviršiai, kuriuos gausime atitinkamai po dviejų, trijų, keturių ir penkių padalinimų. Atkreipkite dėmesį, jog jau po penkių padalijimų paviršius atrodo beveik tolydus.



Pav. 19

3. Tūrių algebra

3.1. Tūrių algebros operacijos

Tūrių algebros operacijos – tai natūralus būdas sudėtingos formos kūnams iš paprastų sudėtinių dalių (primityvų) konstruoti. Šitas metodas yra plačiai taikomas kompiuterinėje grafikoje, kadangi dauguma objektų (ypač dirbtinės prigimties) yra sudaroma iš standartinių geometrinių figūrų – kubų, cilindrų, sferų.

Tūrių algebra remiasi Bulio algebros operacijomis. Yra trys pagrindiniai operatoriai – sąjunga, sankirta, skirtumas. Pateikus operatoriui du objektus iš jų formuojamas trečias.[Loh00]

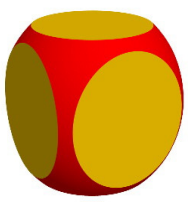
Dviejų objektų **sąjunga** duoda objektą, apimantį erdvę, kurią užima abu duotieji objektai.

Pav. 20. Iš kairės į dešinę: du permatomi objektai; jų sąjunga.

Skirtumo operacijos rezultatas priklauso nuo objektų tvarkos. Rezultatas yra pirmasis objektas iš jo atėmus erdvę, kurioje pastarasis persidengia su antruoju objektu. Kitais žodžiais tariant, ši operacija leidžia „išgręžti“ skyles objekte arba sukurti savitą reljefą.

Pav. 21, 22: Skirtumo operacijos rezultatai.

Sankirtos operacijos rezultatas yra objektas, užimantis tą erdvę, kurioje duotieji du objektai persidengia. Rezultatas yra erdvė, priklausanti abiems objektams.



Pav. 23: Sankirtos operacija pritaikyta kubui ir sferai

Pav. 24: Sankirtos operacija pritaikyta dviems sferoms

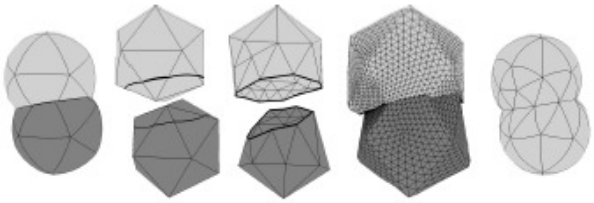
3.2. Tūrių algebra padalijimo paviršiams

Tūrių algebros operacijas yra pakankamai lengva realizuoti, kai „sudedamosios dalys“ yra paprastos geometrinės figūros, tačiau procesas žymiai sudėtingėja, kai norime taikyti šias operacijas objektams, kuriuos apibrėžti yra naudojami komplikuoti skaičiavimai, pavyzdžiui, padalijimo paviršiams.

Vienas būdas tai padaryti remiasi objektų sankirtų radimu. Problema yra ta, kad dažnai negalima rasti tokios matematinės išraiškos, kuri tiksliai aprašytų sankirtos kreivę, be to, skaičiavimai dar labiau sudėtingėja, kai norime atlikti iš karto kelias tūrių algebros operacijas.

Siekiant sumažinti skaičiavimų laiką ir gaunamų figūrų sudėtingumą, metodas yra patobulintas [BKZ98] . Atliekami keturi žingsniai (Pav. 25):

1. Apskaičiuojama (apytikriai) objektų sankirtos kreivė – jos atvaizdžiai ant abiejų kūnų.
2. Surandomas sankirtos kreivės atvaizdis tinkleliuose, panaudotuose objektams suformuoti.
3. Tinkleliai yra pertvarkomi – iš jų „iškerpamos“ nematomos sritys.
4. Tinkleliai optimizuojami ir, pasinaudojant specialius metodus, apjungiami.



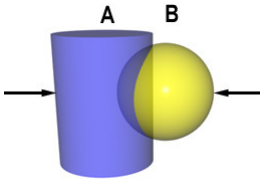
Pav. 25. Dviejų padalijimo paviršių sankirtos radimas

3.3. Tūrių algebra besiremianti spindulių trasavimu

Kaip matome, tūrių algebros operacijų įgyvendinimas padalijimo paviršių lygyje yra sudėtingas procesas, be to, rezultatus gauname apytikslius. To galime išvengti, atlikdami tūrių algebros operacijas spindulių trasavimo lygyje [Kas02]. Tokiu atveju garantuotai gausime vientisas, be „skylių“ figūras.

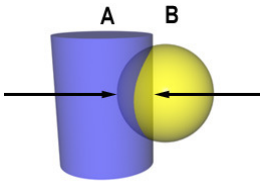
Spindulių trasavime tūrių algebra yra įgyvendinama “įėjimo/išėjimo” testais, t. y. tikrinant

šviesos spindulio ir objektų susikirtimus. Paprasčiausias atvejis yra sąjungos operacija (Pav. 26). Kai spindulys kerta kurį nors objektą, A arba B susikirtimo taškas laikomas A ir B sąjungos paviršiaus tašku ir vaizduojamas atsižvelgiant į suteiktas paviršiui sąvybes bei apšviestumą. Jei spindulis kerta abu objektus, imamas pirmas (artimesnis) susikirtimo taškas.



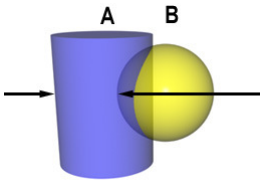
Pav. 26: Sąjunga – $A \cup B$

Kiek sudėtingiau realizuojama sankirtos operacija (Pav. 27), kada matoma yra tik ta objektų dalis, kuri priklauso ir objektui A, ir objektui B. Kai spindulys pirmą kartą kerta objektą A („įeina“ į jį), susikirtimo taškas nevaizduojamas, bet vyksta tikrinimas ar tas pats spindulys, prieš „išeidamas“ iš objekto A (t.y. prieš kirsdamas objekto paviršių antrą kartą) susikerta su objektu B. Jei susikirtimo su objektu B nėra, spindulys ignoruojamas. Tuo atveju, kai susikirtimo su objektu B taškas yra rastas, toks taškas yra laikomas priklausančiu objektų sankirtai ir atitinkamai vaizduojamas.



Pav. 27: Sankirta – $A \cap B$

Analogiškai yra atliekama skirtumo operacija Pavyzdžiui, reikia surasti A ir B skirtumą (Pav. 28). Kai spindulys, nekirsdamas objekto B, kerta objektą A, toks taškas automatiškai laikomas priklausančiu naujam paviršiui. Kiek sudėtingiau yra, jei šviesos spindulys iš pradžių kerta objektą B: toks taškas yra ignoruojamas, tačiau jei po to yra randamas susikirtimas su objektu A, tai „išėjimo“ iš objekto B taškas laikomas priklausančiu skirtumo paviršiui ir vaizduojamas. Reikia atkreipti dėmesį į tai, kad, skirtingai nuo aukščiau minėtų atvejų, kur normalės aptiktuose taškuose sutapdavo su atitinkamų pradinių objektų normalėmis, čia reikia nukreipti B objekto normalę į priešingą pusę (kadangi normalė turi būti nukreipta „į išorę“).



Pav. 28: Skirtumas – $A - B$

Pastebėtina, kad realizuojant tūrių algebros operacijas spindulių trasavimo lygyje dažnai reikalaujama ieškoti bei lyginti dviejų susikirtimo taškų koordinates. Operuojant parametrinėje koordinačių sistemoje lengvai yra nustatoma, ar pirmas susikirtimo taškas priklauso kitam objektui. Tuo galima pasinaudoti, optimizuojant metodą [Kas02] .

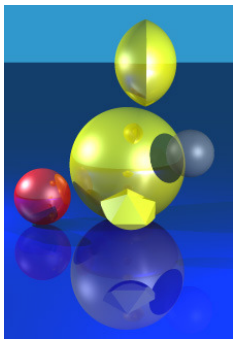
3.3.1. Tūrių algebros praktinės realizacijos ypatumai ir kodo pavyzdžiai

```
public class Bool extends GObject {
```

```
  public    GObject obj1;
  public    GObject obj2;
  public    GObject activeObj; //normalei surasti
  DoubleVal t1, t2;
double    ClosestDist;
String    operation;
```

Tūrių algebros objektams saugoti buvo sukurta nauja klasė Bool, kurioje laikomi du objektai (jie gali būti bet kokie – paprasti arba Bool). Objektas activeObj žymi objektą, kuris tuo metu yra aktyvus, t.y. vaizduojamas. t1, t2 yra saugomi atstumai atitinkamai iki obj1 ir obj2. Kintamasis operation saugoja operacijos tipą (sankirta, sąjunga, skirtumas).

Pav. 29. Problemos pavyzdys.



Pav. 30. Problema išspręsta.

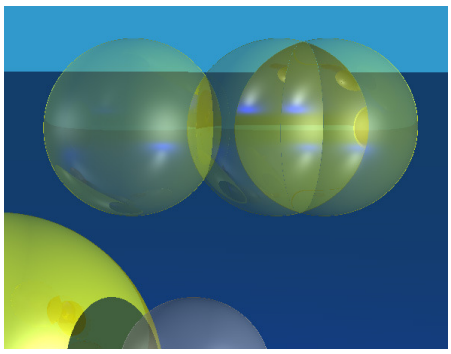
Su šia problema (Pav. 29) buvo susidurta, bandant realizuoti sankirtą nenaudojant activeObj kintamojo. Deja, praktika parodė, kad tokiu būdu nepavyksta nustatyti, kurio objekto paviršių vaizduoti.

Paprasčiausiai realizuojama operacija – sankirta.

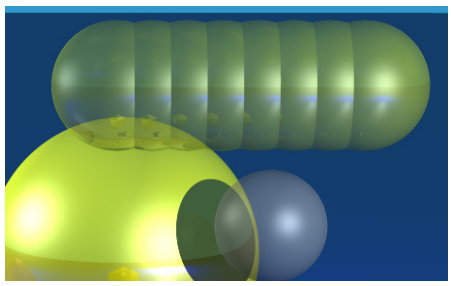
```
public boolean IntersectIntersection ( Ray ray, DoubleVal t ) {
  boolean intersectOK1;
  boolean intersectOK2;

  ClosestDist = Constants.INFINITY;
  t1.value = t2.value = t.value;
  intersectOK1 = intersectOK2 = false;

  intersectOK1 = obj1.Intersect ( ray, t1 );
  intersectOK2 = obj2.Intersect ( ray, t2 );
  if ( intersectOK1 &&
       intersectOK2 &&
       t1.value < Constants.INFINITY &&
       t2.value < Constants.INFINITY ) {
    if ( t1.value > t2.value ) {
      t.value   = t1.value;
      activeObj = obj1;
    }
    else {
      t.value = t2.value;
      activeObj = obj2;
    }
    return ( t1.value > Constants.GeomThreshold ) &&
           ( t2.value > Constants.GeomThreshold );
  }
       else
         return false;
}
```

Šis metodas veikia tokiu principu: ieškome susikirtimo su obj1 ir obj2. Jei spindulys susikerta su abiem objektais (t.y. intersectOK1 ir intersectOK2 yra „true“), žiūrime, kuris objektas yra toliau (t.y. lyginame t1 ir t2 reikšmes). Rade tolimesnį objektą, jį priskiriame activeObj, t.y. skaičiuojant tekstūrą arba spalvą, apšvietimą duomenys bus imami būtent iš šio objekto.

Jei spindulys kerta tik vieną iš objektų arba nekerta nei vieno iš jų, grąžiname reikšmę „false“ (t.y. šių dviejų objektų sankirta nevaizduojama).



Pav. 31. Netikėtos komplikacijos.



Pav. 32. Galutinis sąjungos rezultatas

Atliekant daugiau nei dviejų objektų sąjungą (pavyzdžiui, vienas iš sąjungos objektų taip pat yra sąjunga) atsirado netikėtų komplikacijų: skaičiuojant permatomumą, vienas iš objektų buvo traktuojamas kaip atskiras objektas ir tapo pilnai matomas (Pav. 31). Problemą buvo išspręsta taip: prieš rekursyviai skaičiuojant permatomumą, metodas SetEntering visiems susijusiems objektams nurodo, kad spindulys yra sąjungos viduje.

Sąjungos realizavimas yra šiek tiek sudėtingesnis nei sankirtos.

```
public boolean IntersectUnion ( Ray ray, DoubleVal t ) {
  boolean intersectOK1;
  boolean intersectOK2;

  if ( !Entering ) {
    ClosestDist = Constants.INFINITY;
    t1.value = t2.value = t.value;
    intersectOK1 = intersectOK2 = false;

    intersectOK1 = obj1.Intersect ( ray, t1 );
    intersectOK2 = obj2.Intersect ( ray, t2 );
    if ( intersectOK1 && t1.value > Constants.GeomThreshold ) {
      t.value   = t1.value;
      if ( !Constants.shadowtest )
        activeObj = obj1;
    }
    if ( intersectOK2 &&
         t2.value < t1.value &&
         t2.value > Constants.GeomThreshold ) {
      t.value = t2.value;
      if ( !Constants.shadowtest )
        activeObj = obj2;
    }
    return ( intersectOK1 || intersectOK2 );
  }
  else {
    ClosestDist = Constants.INFINITY;
    activeObj.Intersect ( ray, t );
    return ( t.value > Constants.GeomThreshold );
```

```
    }
}
```

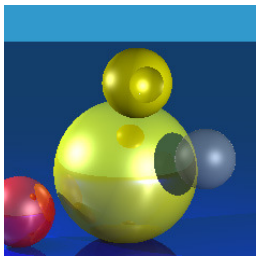
Sąjungos operacijai realizuoti GObject klasėje buvo pridėtas naujas boolean tipo kintamasis – Entering. Jis yra skirtas pažymėti, kad spindulys įėjęs į objektą (žymėjimas daromas rekursyviai – tai prasminga tuo atveju, kai sąjungoje dalyvauja daugiau nei 2 objektai) ir kad būtų atsižvelgiama tik į tuos spindulius, kurie išeina iš objekto.

Sąjungos vaizdavimo principas toks pat, kaip ir sankirtos, tik šįkart aktyviu pažymime objektą, kuris yra arčiau spindulio šaltinio (sankirtoje mes ėmėme objektą, kuris yra toliau nuo spindulio šaltinio). Be to, užtenka, kad spindulys kirstų bent vieną iš dviejų objektų (sankirtoje buvo privaloma, kad spindulys kirstų abu objektus).

Iš PointLight.java:

```
Constants.shadowtest = true;
Occlude = Constants.Scene.Intersect ( ray, t );
Constants.shadowtest = false;
```

Naudojamas papildomas parametras Constants.shadowtest. Jo reikšmė visada būna „true“ išskyrus tuo atveju, kai tikrinamas šešėlis. Kiekvieno ekrano taško apšvietimo skaičiavimo metu yra tikrinama ar kokie nors objektai tiesiogiai neužstoja šviesos šaltinio (pvz, PointLight). Šio susikirtimo tikrinimo metu yra ieškoma sankirtos ir su mūsų sąjungos objektu, o Constants.shadowtest parametras mums garantuoja, kad šešėlio tikrinimo metu nebus pakeistas aktyvus objektas.

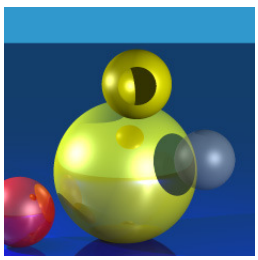


Pav. 33.

Realizuojant skirtumo operacija, iškyla nenumatytos problemos skaičiuojant šešėlius (Pav. 33). Reikėjo atsižvelgti į du atvejus:

1). Objektas gali užstoti pats save, taip sudarydamas šešėlį.

2). Tikrinant šešėlį, activeObj turi nekisti. Pirmąją problemą išsprendžiau apsirašydamas papildomą algoritmą rasti tolimiausiam sankirtos taškui ir spindulį testRay. Antrąją problemą išspendžiau aprašydamas globalų kintamąjį Constants.shadowtest, kuris garantuoja, jog sankirtos radimo metu nebus keičiama activeObj reikšmė.



Pav. 34. Šešėliai vaizduojami tvarkingai, antru atveju mažoji sfera yra įdėta giliau

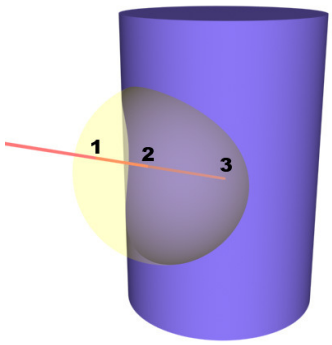
Paskutinės – skirtumo – operacijos logika tokia: - iš pirmojo objekto atimamas antras, t.y.

pirmasis objektas laikomas pagrindiniu. Skirtumo realizacija pasirodė esanti sudėtingiausia. Šią operaciją reikėjo aprašyti dviems atvejams – skirtumo vaizdavimui ir šešėlių skaičiavimui.

```
public boolean IntersectDifference ( Ray ray, DoubleVal t ) {
  boolean     intersectOK1;
  boolean     intersectOK2;
  boolean     intersectObj2SecondTime;

  t1.value = t2.value = t.value;
  intersectOK1 = intersectOK2 = intersectObj2SecondTime = false;

  intersectOK1 = obj1.Intersect ( ray, t1 );

  if ( Constants.shadowtest ) {//seselio testavimas
    intersectOK2 = obj2.Intersect ( ray, t2 );
    if ( !intersectOK2 ) { //kerta pagrindini objekta tik
      t.value = t1.value;
      return ( t.value > Constants.GeomThreshold );
    }
    if ( t1.value < t2.value ) {
      t.value = t1.value;
      return false;
    }
    else
      t.value = t2.value;
    return true;
  }
  else {//ne seselio testavimas
    if ( obj2.Intersect ( ray, t2 ) ) {
      if ( t1.value < t2.value && intersectOK1 ) {
        t.value = t1.value;
        activeObj = obj1;
        return ( t.value > Constants.GeomThreshold );
      }
      if ( t2.value < Constants.INFINITY ) {
        double t_Old = t2.value;
        Ray testRay = new Ray ( ray.Point(t2.value), ray.Dir );
        obj2.InverseNormal = false;
        intersectObj2SecondTime = obj2.Intersect ( testRay, t2 );
        obj2.InverseNormal = true;
        t2.value += t_Old;
        if ( intersectObj2SecondTime && t2.value > t1.value ) {
          t.value = t2.value;
          activeObj = obj2;
        }
        else {
          t.value = t1.value;
          activeObj = obj1;
        }
      }
    }
    else {
      activeObj = obj1;
      t.value = t1.value;
    }
  }
  return ( t.value > Constants.GeomThreshold );
}
```

Šešėlio vaizdavimas skaičiuojamas paprastai: skaičiuojame ar spindulys kerta abu objektus. Jei jis kerta tik pagrindinį objektą, grąžiname susikirtimo atstumą ir patvirtinimą, kad spindulys kirto kliūtį. Jei spindulys kirto abu objektus, lyginame atstumus (t.y. kurį objektą kirto pirmą) ir grąžiname atitinkamas reikšmes.

Kai skaičiuojame skirtumą, žiūrime, kuriuos objektus spindulys kirto (ir kurį iš jų pirmą, jei kirto abu). Jei kertamas tik pagrindinis objektas, tiesiog priskiriame jį aktyviam objektui ir grąžiname atstumą. Kitu atveju, kai kirsti abu objektai, iš antrojo (ne pagrindinio!) objekto sankirtos taško (Pav. 35, taškas 1) paleidžiame papildomą spindulį ta pačia kryptimi. Jei papildomas spindulys kerta nepagrindinį objektą (36 pav., taškas 3) ir atstumas iki šaltinio (kameros) yra didesnis už pirmojo (Pav. 35, taškas 2), reiškia, kad nepagrindinis objektas yra aktyvus, (activeObj priskiriame obj2), o matomas taškas – 3 (Pav. 35).



Pav. 35. 1 – spindulys kerta papildomą objektą pirmą kartą.
2 – spindulys kerta pagrindinį objektą.
3 – papildomas spindulys kerta papildomą objektą antrą kartą.

4. Dengimas tekstūromis

Dengimas tekstūromis (Texture Mapping) yra viena žinomiausių ir plačiausiai naudojamų kompiuterinės grafikos technologijų, skirtų realistiškų vaizdų gavimui. Priežastys yra tai, tad tekstūrų naudojimas leidžia išvengti ilgų skaičiavimų, sutaupyti resursus vaizduojant paviršiaus sudėtingą reljefą (žemės, medžiagos ir pan.). Dengimas tekstūromis yra realizuojamas tiek programiniu, tiek mašininiu lygiu. Mes nagrinėsime tekstūrų dengimą programiniu būdu.

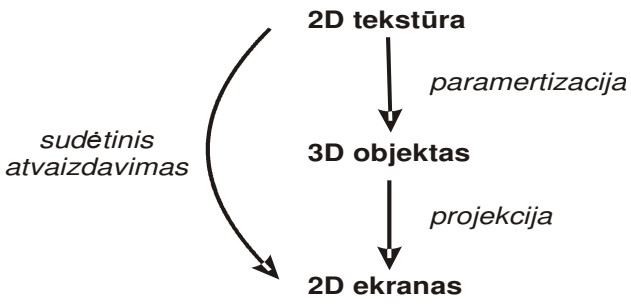
3.1. Pagrindiniai dengimo tekstūromis metodai

Tekstūra – tai yra dvimatis piešinys, tam tikru būdu atvaizduojamas ant trimačio objekto paviršiaus. Dažniausiai tekstūrų šaltiniais būna vaizdiniai failai (Image Mapping), tačiau yra naudojamos ir programos generuojamos tekstūros. Kiekvienas būdas turi savo privalumų ir trūkumų.

Failuose gali būti saugomi įvairiausio pobūdžio tekstūros. Tokios tekstūros yra populiariausios, kadangi vienintelis jų trūkumas – fiksuotas detalumas. Dažnai tam, kad suteikti objektui realistišką vaizdą, naudojamos dvi teksturos – didesnė ir detalesnė – kai objektas rodomas arti žiūrovo ir mažesnė, ne tokia detali – kai į objektą žiūrima iš tolo. Tekstūrai generuoti gali būti naudojamas tiek visas dvimatis vaizdas, tiek jo dalys. Dažnai dvimatis vaizdas būna tam tikru būdu transformuojamas, taip, kad natūraliai atrodytų būdamas atvaizduotas į objektą. Šis procesas (Image Warping) yra plačiai naudojamas uždedant tekstūras ant apvalios formos paviršių.

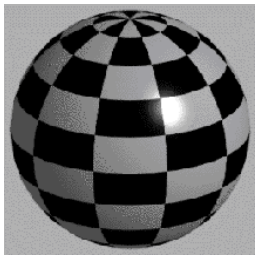
Pav. 36. Netransformuoto (kairėje) ir transformuoto (dešinėje) vaizdų uždėjimo ant sferos rezultatai.

Tekstūros uždėjimo remiantis dvimačių vaizdų procese galima išskirti du pagrindinius etapus (Pav.37).

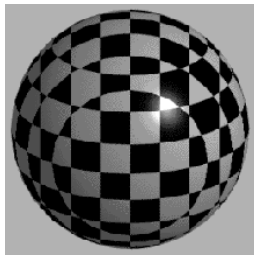


Pav.37.

Tais atvejais, kai detalumas yra itin svarbus, galima generuoti tekstūrą programiškai (Procedural Texture Mapping). Tai pat procedūrinis tekstūros generavimas dažnai leidžia pasiekti efektus, kurios kitaip gauti būtų žymiai sudėtingiau.



Pav. 38. Tekstūra pagal 2D vaizdą.



Pav. 39. Programiškai sugeneruota tekstūra.

Vienas objektas gali būti padengtas keliomis tekstūromis. Tekstūra gali nusakyti ne tik objekto paviršiaus spalvas, bet ir kitokias jos charakteristikas: permatomumą, atsispindėjusio intensyvumą, švietimosi lygį, iškreipimą ir t.t. Tai supaprastina modeliavimo procesą ir leidžia pasiekti įspūdingų efektų.

3.2. Dengimo tekstūromis technologijos

Šiuo metų yra sukurta daug įvairiausių dengimo tekstūromis metodų. Kokį jų naudoti priklauso nuo vaizdo generavimo technologijos, nuo objektų, ant kurių dėsime tekstūrą, ypatybių, nuo to, kam vaizdas bus skirtas. Pavyzdžiui, norint gauti realaus laiko animaciją, parenkamas greičiausiai veikiantis algoritmas, o generuojant fotorealistinį, aukštos kokybės vaizdą, svarbiausias yra tekstūrų natūralumas.

Renderinant vaizdą, dažniausiai tekstūra yra priskiriama visai eilei nuskaitytų objekto taškų. Tai yra greitas ir plačiai paplitęs vaizdų generavimo metodas, yra sukurta daug įvairių algoritmų tai daryti, kurie skiriasi tikslumu, greičiu bei įgyvendinimo sudėtingumu. Šių algoritmų pavyzdžiai: afininis (Affine Mapping), nuskaitytos eilutės padalinimo (Scanline Subdivision), parabolinis (Parabolic Mapping).

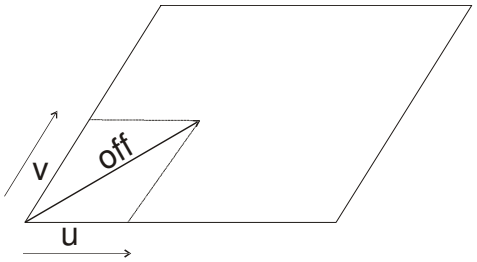
Spindulių trasavimo metodas kiekvieną vaizdo tašką skaičiuoja atskirai, todėl tokie algoritmai jam netinka. Mūsų atveju dengimo tekstūra procesas susideda iš dviejų etapų:

1) Tekstūros taško, atitinkančio objekto taškui, suradimas (tam tikros funkcijos pagalba paviršius susiejamas su tekstura).
2) Objekto taško spalvos nustatymas remiantis tekstūra.

Egzistuoja dauguma įvairiausių paviršiaus ir tekstūros susiejimo funkcijų, tačiau tarp jų nėra universalių: visos jos yra pritaikytos tam tikriems atvejams ir visiškai netinka kitiems. Pasirinkome populiariausius trimačių objektų į plokščią tekstūrą atvaizdavimo metodus. „Plokščias“ atvaizdavimas tinka objektams, susidedantiems iš plokščių sienų (kubas, tetraedras), polinis – sferiniams, apvaliems.

1) „Plokščias“ (Planar Projection)

Tai dvimačio objekto (plokštumos, plokščios trimačio kūno sienos) atvaizdavimo į dvimatę tekstūrą metodas. Laikoma, kad abu šie objektai tam tikru būdu (nebūtinai lygiagrečiai) yra patalpinti trimatėje erdvėje, ir skaičiuojama projekcija vieno į kitą.

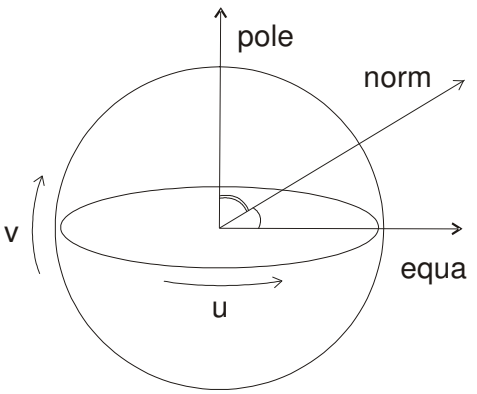


Pav. 40. Plokščios tekstūros parametrizacija

```
if (mappingType == "PLANAR"){
   off = point.subtract(point, center.subtract
                              (center, new Vector(sideLength/2)));
   u = (off.x*sideLength+off.y*sideLength)/(sideLength*sideLength*2);
   v = (off.x*sideLength+off.y*sideLength)/(sideLength*sideLength*2);
   sd.texture.applyColorMap(u, v, sd);
}
```

2) „Polinis“ (Polar Projection)

Polinis metodas naudojamas sferiniams kūnams dengti. Tai nėra projekcija tikra to žodžio prasme – tiesiog kampai tarp sferos taškų normalių ir jos „polių“ bei normalės į „ekvatorių“ laikomi stačiakampio (kuris atvaizduojamas į tekstūrą) pločiu ir ilgiu.



Pav. 41. Sferinės tekstūros parametrizacija

```
if (mappingType == "SPHERE"){
```

```
   norm = point.divide(point.subtract
               (point, center), Math.sqrt(2*sideLength*sideLength));
   lat = Math.acos(-1*(norm.scalarProduct(norm, pole)));
   v = lat*0.318;

   temp = equa.scalarProduct(equa, norm)/Math.sin(lat);
   if (temp > 1){ temp = 1; }
   else if (temp < -1){ temp = -1; }

   ll = 0.15916*Math.acos(temp);

   if (pole.scalarProduct(pole.vectorProduct(pole, equa),norm) > 0){
      u = ll;
   } else { u = 1 - ll; }
}
```

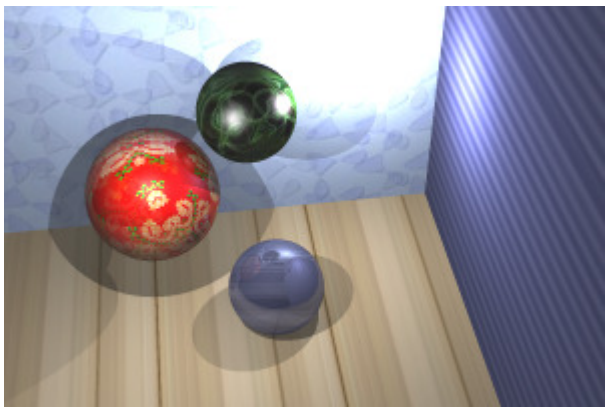
3) „Cilindrinis“ (Cylindrical Projection)

Naudojamas sferiniams kūnams dengti, realizacijos atžvilgiu yra pirmųjų dviejų dengimo būdų derinys. Sferos „platumos“ koordinates atvaizduojamos į stačiakampį kaip lygiagrečios vienodu atstumu viena nuo kitos išsidėsčiusios linijos. Ilgumos koordinatės atvaizduojamos į linijas, išsidėsčiusias nelygiai ir joms statmenas.

```
if (mappingType == "CILLINDER"){
   norm = point.divide(point.subtract
         (point, center), Math.sqrt(2*sideLength*sideLength));
   off = point.subtract(point, center.subtract
                              (center, new Vector(sideLength/2)));
   v = (off.x*sideLength+off.y*sideLength)/(sideLength*sideLength*2);
   lat = Math.acos(-1*(norm.scalarProduct(norm, pole)));
   temp = equa.scalarProduct(equa, norm)/Math.sin(lat);
   if (temp > 1){ temp = 1; }
   else if (temp < -1){ temp = -1; }
   ll = 0.15916*Math.acos(temp);
   if (pole.scalarProduct(pole.vectorProduct(pole, equa), norm) > 0){
      u = ll;
   } else { u = 1 - ll; }
}
```

Taško spalva yra nustatymas pagal keturius artimiausius parametrizuotam taškui (u,v) tekstelius (tekstūros pikselius).

```
      int   ix = (int) x;
      int   iy = (int) y;
      int   jx = ix + 1;
      int   jy = iy + 1;

      x -= ix;
      y -= iy;
      if (jx >= width) jx = 0;
      if (jy >= height) jy = 0;

      RGB c00 = new RGB (red[ix][iy], green[ix][iy], blue[ix][iy]);
      RGB c01 = new RGB (red[ix][jy], green[ix][jy], blue[ix][jy]);
      RGB c10 = new RGB (red[jx][iy], green[jx][iy], blue[jx][iy]);
      RGB c11 = new RGB (red[jx][jy], green[jx][jy], blue[jx][jy]);

      t.color.x = ((1-tx)*(1-ty)*c00.red+(1-tx)*ty*c01.red+
                   tx*(1-ty)*c10.red+tx*ty*c11.red)/255;
      t.color.y = ((1-tx)*(1-ty)*c00.green+(1-tx)*ty*c01.green
                  +tx*(1-ty)*c10.green+tx*ty*c11.green)/255;
      t.color.z = ((1-tx)*(1-ty)*c00.blue+(1-tx)*ty*c01.blue
                  +tx*(1-ty)*c10.blue+tx*ty*c11.blue)/255;
```



Pav. 42. Plokščios ir sferinės tekstūros pavaizduotos spindulių trasavimo pagalba

3) Pagal „išklotinę“ (Mapping)

Šito metodo esmė – kiekvienai objekto sienai priskiriamas tekstūros gabalėlis, kuriuo ji bus dengiama. Tokio dengimo būdo privalumas – jis yra intuityviai suvokiamas, t. y., rezultatas yra pakankamai lengvai nuspėjamas, kas leidžia be didelių pastangų paruošti tinkamas tekstūras. Trukumas yra tas, kad šis metodas reikalauja papildomos informacijos: praktiškai tai reiškia, jog, jeigu informaciją apie objektą yra saugoma duomenų strūktūroje, prireiks antros duomenų struktūros, kurioje bus laikomas kiekvienos objekto sienos atvaizdis plokščioje tekstūroje. Mūsų atveju, kadangi dirbama yra su dinamiškai dalinamais padalinimo paviršiais, tai reiškia, kad visus padalinimus reikės atlikti dukart, o tai užima daug laiko ir atminties.

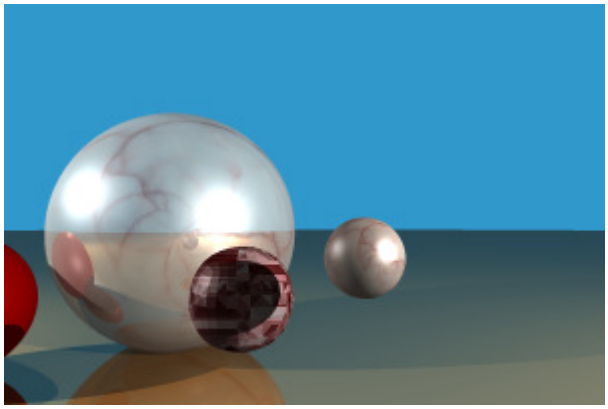
Musų programa apdoroja paviršiaus išklotinę saugojančius .tex failus ir, remiantis jų duomenimis, atlieka paviršiaus dengimą. Kaip paviršius bus atvaizduotas į tekstūrą, neturi reikšmės: svarbu yra, kad paviršius ir jo išklotinė turėtų vienodą sienų skaičių.

// teksturos failo nuskaitymas; duomenų struktūros sukurimas:

```
 <...>
 DatFileReader reader = new DatFileReader(fileName + ".dat" );
 surfaceData = new MeshSurfaceData(reader);

 if ( Constants.textureFromFile ) {
    TexFileReader texReader = new TexFileReader(texFileName+ ".tex" );
    subdivisionTexture.meshMap = new MeshSurfaceData(texReader);
    subdivisionTexture.setMappingType("CUSTOM");
 }
<...>
```

// teksturos parametrizuotų koordinačių suradimas:

```
<...>
else if ( mappingType == "CUSTOM" ) {
Vector point1, point2, point3;
nt i = (int)sideLength;
point1 = new Vector(meshMap.points[meshMap.faces[i][0]].x,
                    meshMap.points[meshMap.faces[i][0]].y, 0);
point2 = new Vector(meshMap.points[meshMap.faces[i][1]].x,
                    meshMap.points[meshMap.faces[i][1]].y, 0);
point3 = new Vector(meshMap.points[meshMap.faces[i][2]].x,
                    meshMap.points[meshMap.faces[i][2]].y, 0);
u = point1.x * point.x + point2.x * point.y + point3.x * point.z;
v = point1.y * point.x + point2.y * point.y + point3.y * point.z;

<...>
```

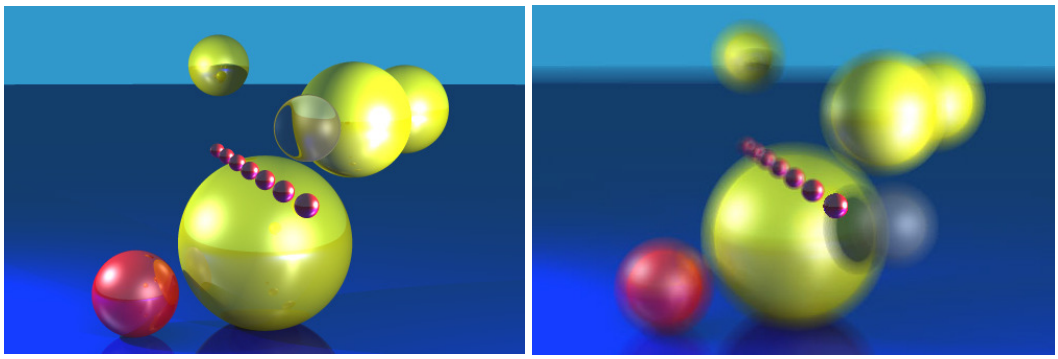


Pav. 43: Kubas, padalintas pagal Catmull-Clark algoritmą
ir padengtas tekstūra, panaudojant „išklotinės“ failą.

5. Specialūs efektai

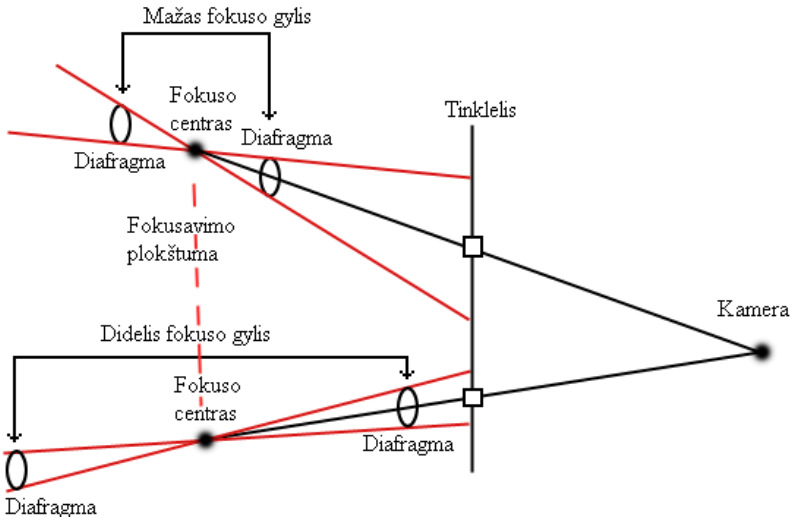
Spindulių trasavimo technologija leidžia be didelių pastangų realizuoti keletą vizualių efektų, susijsių su kameros filmavimo ypatumais. Mūsų tikslas buvo pritaikyti mūsų programai judesio šleifo (Motion blur) bei kameros fokusavimo gylio (Depth of field) efektus.

5.1 Kameros fokusavimo gylio (depth of field) efektas



Pav. 44, 45 – kairėje matome paveiksliuką be kameros fokuso, dešinėje – su kameros fokusavimo gylio efektu. Fokuso centras – pirmoji raudona sfera.

Norėdami išgauti šį efektą, pirmiausia turime apsibrėžti židinio plokštumą (pastaroji yra lygiagreti ekranui – pav. 7, 8, 9, 10). Židinio plokštuma (ją galima apibrėžti atstumu nuo kameros) bus fokuso centre, t.y. visi objektai, esantys plokštumoje, bus idealiai ryškūs. Kiekvienam taškui leidžiame spindulį, ir tas taškas, kuriame jis kerta židinio plokštumą, yra fokuso taškas. Tada leidžiame papildomus spindulius suformuotus taip, kad jie visi kirstų fokuso tašką. Kiekvienam spinduliui apskaičiuojame taško spalvą, o rezultatą gauname suradę taškų spalvų vidurkį.



Pav. 46 – Kameros fokuso veikimo principai

Pastebime, kad spinduliai susikerta fokuso centre. Prieš ir po susikirtimo jie apima erdvę, vadinamą fokuso erdve. Kuo toliau spindulys yra nuo fokuso centro, tuo didesnis erdvės skersmuo. Skersmens dydis nusako scenos dydį, kuris dalyvauja taško spalvos skaičiavime. Kuo didesnis skersmuo, tuo mažiau ryškūs yra kontūrai, nes daugiau objektų sudaro taško spalvą. Ten, kur erdvės skersmuo yra mažesnis už ekrano tašką, ir yra fokuso centras. Tie objektai, kurie patenka į erdvės skersmenį šiame taške, yra idealiai ryškūs. Kuo mažesnė yra diafragma, tuo didesnis yra fokuso gylis.

Mūsų realizacijoje fokuso gylio efektas yra nustatomas trimis parametrais – plokštumos atstumu nuo kameros, diafragmos skersmeniu, ir spindulių kiekiu. Žinant plokštumos atstumą ir kameros duomenis (poziciją erdvėje ir krypties vektorių), apskaičiuojame fokuso centrą:

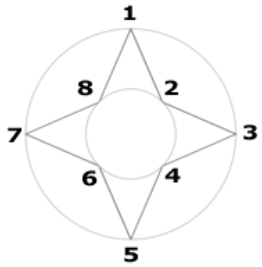
```
public static void SetDOF ( double distance, double radius ) {
   Vector tmp = new Vector ( EyeDir );
   DefaultLength = tmp.Length ( EyeDir );
   DOFradius = radius;
   double ratio = distance / DefaultLength;
   tmp = tmp.Multiply ( tmp, ratio );
   DOFEye = tmp.Add ( tmp, Eye);
 }
```

Kiekvienam taškui leidžiame nustatytą spindulių kiekį, truputį pastumdami kamerą įvairiomis kryptimis. Kameros pastūmimo stiprumas priklauso nuo vartotojo nurodyto diafragmos skersmens. Po kameros pastūmimo reikia apskaičiuoti krypties vektorių taip, kad jis kirstų fokuso centrą:

```
public static void TransformDOFCamera ( Vector move ) {
   Eye = Eye.Add ( Eye, move );
   EyeDir = DOFEye.Subtract ( DOFEye, Eye );
   double currentLength = EyeDir.Length ( EyeDir );
   double ratio = DefaultLength / currentLength;
   EyeDir = EyeDir.Multiply ( EyeDir, ratio );
   double debuging = EyeDir.Length ( EyeDir );
 }
```

Po kiekvieno pastūmimo ir spindulio apskaičiavimo gauta taško spalva yra išsaugoma, o kamera grąžinama į pradinę padėtį. Atlikus nurodytą kiekį pastūmimų, skaičiuojame gautų taško

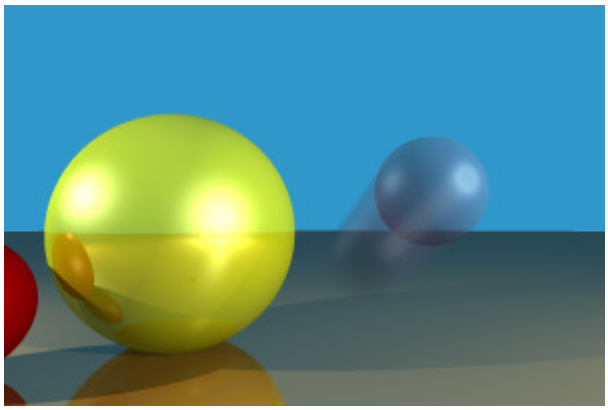
spalvų vidurkį. Buvo išbandyta keletas kameros pastūmimo principų (pvz: pastumti kamerą atsitiktine kryptimi tam tikru atstumu, kamerą stumti taisyklingu apskritimu, kurio centras – pradinė kameros pozicija ir t.t.), tačiau geriausius vizualinius rezultatus davė kamerų išdėstymas tokiu principu: kameros tolygiai dėstomos apskritimu (pav. 47), tačiau kas antra kamera yra dedama perpus mažesniu atstumu nei nustatyta diafragma.



Pav. 47: 8 kamerų išsidėstymas. Didysis apskritimas apskaičiuojamas pagal vartotojo nurodytą parametrą.

5.2 Judesio šleifo (motion blur) efektas

Kai filmuojamas objektas greitai juda, jo vaizdas juostoje ar nuotraukoje tampa neryškus (Pav. 44). Judesio šleifo efektas yra plačiai naudojamas trimatėje animacijoje, siekiant tikroviškumo.


Pav.48

Spindulių trasavimo technologija leidžia be didelių pastangų simuliuoti tokį efektą. Egzistuoja daug būdų tam įgyvendinti, kurių pagrindas yra judančio objekto trasavimas skirtingose padėtyse ir visų gautų rezultatų suvedimas į vieną.

Siekiant implementuoti judesio šleifo efektą, ivedėme laiko sąvoką. Laiką žymi realus skaičius, priklausantis intervalui [0; 1] – tiek laiko lieka atvira mūsų virtualios kameros diafragma. Objektams suteikiame judėjimo kryptį ir greitį bei judėjimo pradžios ir pabaigos laiką (objektas nebūtinai turi judėti visą filmavimo laiką – jis gali staiga sujudėti arba sustoti).

Papildomi objekto parametrai:

```
public class GObject {
<...>
   public double startTime = 0.0;
   public double finishTime = 1.0;
   public double speed = 0;
   public Vector motionDirection = new Vector();
```

Šviesos spinduliui buvo suteiktas papildomas parametras – paleidimo laikas:

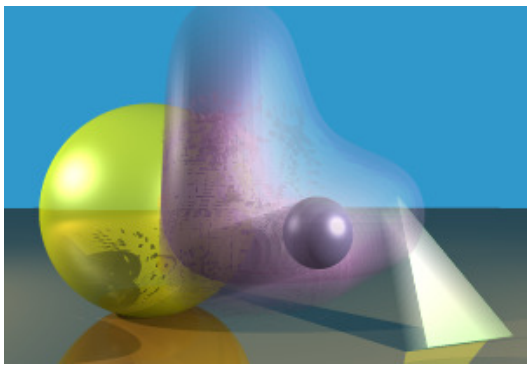
```
public class Ray {
   <...>
   double time;
```

Render klasė buvo papildyta metodu `motionBlurRenderScene`, turinčių papildomą parametrą `momentAmount`. Šis parametras rodo, kiek kartų, keičiant spindulių laiką, bus renderinama scena. Galutinis rezultatas yra visų tų scenų vidurkis.

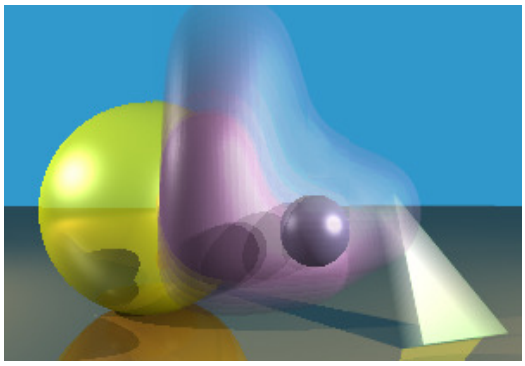
```
static public void motionBlurRenderScene(
              double halfWidth, double halfHeight, int nx, int ny,
              int momentAmount, String picFileName){
   <...>
   for ( i = 0, y = halfHeight; i < ny; i++, y -= hy ) {
     System.out.print ( "." );
     for ( j = 0, x = - halfWidth; j < nx; j++, x += hx ) {
       Constants.amount++;
       color = new Vector( 0.0 );


       // Šitas ciklas realizuoja judesio šleifo efektą:
       // nurodytą skaičių kartų paleidžia į tašką spindulį,
       // kiekvieną kartą skirtingu laiku
       // galutinė taško spalva – rezultatų vidurkis

       for ( int k = 0; k < momentAmount; k++ ) {

         ray.setTime( ( double ) ( k ) / momentAmount );
         Tracer.camera ( x, y, ray );
         Constants.ifTriangle = 0;
         Constants.closestDist = Constants.INFINITY;
         instColor = Tracer.trace ( Constants.AIR, 1.0, ray );

         instColor = instColor.clip ( instColor );
         Vector temp = new Vector(color.multiply
                              (instColor,1.0/ momentAmount));
         color = color.add(color, temp );
       }
        <...>
```

Analogiškai buvo implementuotas judesio šleifo efektas paskirstytam (distributed) trasavimui.

Geometriniams objektams buvo apibrėžta postūmio transformacija. Padalinimo paviršiaus atveju buvo susidurta su problemomis, iškilusiomis dėl Octobox strukturos ypatumų (Pav. 49) Klaidos buvo aptiktos ir pašalintos (Pav. 50).



Pav. 49

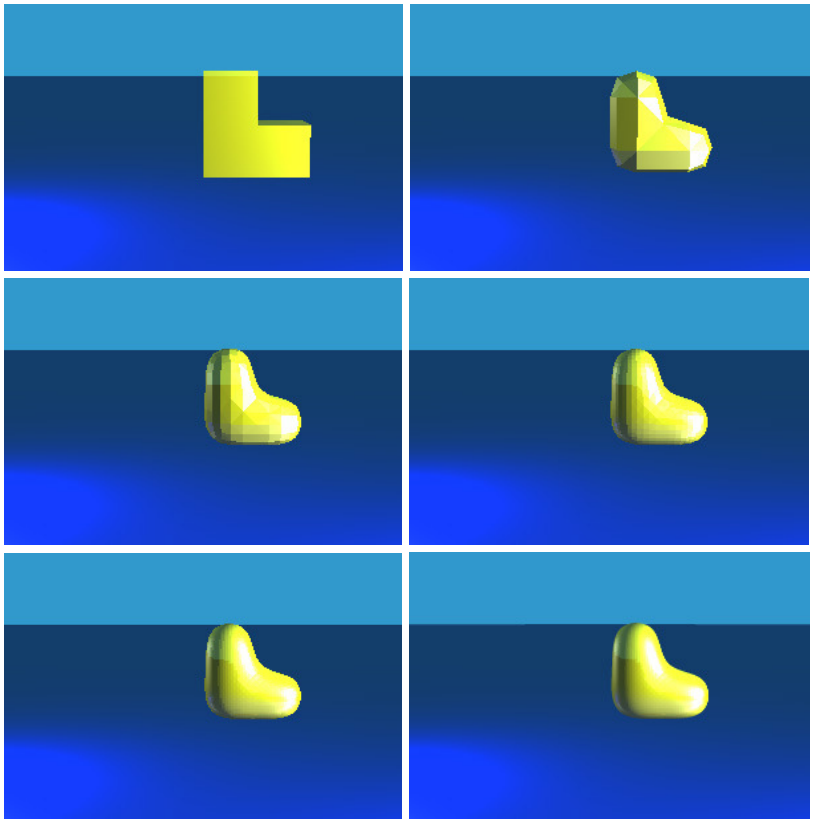


Pav. 50

6. Adaptyvus padalijimas

6.1. Padalijimo žingsnių nustatymas pagal kraštinės projekciją

Šis algoritmas remiasi bandomuoju trasavimu, kai iš scenos yra atmetami visi vientisi objektai, paliekami tik tinkleliai, atsisakoma tekstūrų ir atspindžių. Kiekvieno objekto tinklelis yra sudalomas į trikampius (tai yra aktualu tuo atveju, kai pradinis tinklelis būna sudarytas iš keturkampių – pvz, pritaikytas Catmull-Clark algoritmui). Tada vyksta bandomasis trasavimas, kurio metu kiekvienam trikampiui pažymimos jo projekcijų į ekraną koordinatės. Jos panaudojamos tam, kad galėtumėme rasti ilgiausią trikampio kraštinę, jos ilgį išreiškiant ekrano taškais (pikseliais).



Pav. 51. Ilgiausios kraštinės: 41 taškas, 20 taškų, 10 taškų, 5 taškai, 2 taškai, 1 taškas (idealus glodumas)

Turėdami objekto trikampių ilgiausią kraštinę ir žinodami Catmull-Clark algoritmo savybę, kad kiekviename padalinimo žingsnyje briaunos sutrumpėja per pusę, galime paprastai suskaičiuoti, kiek padalinimo žingsnių reiks atlikti. Atlikus 4 padalijimo žingsnius, ilgiausia kraštinė bus tik 2 taškai (Pav. 51).

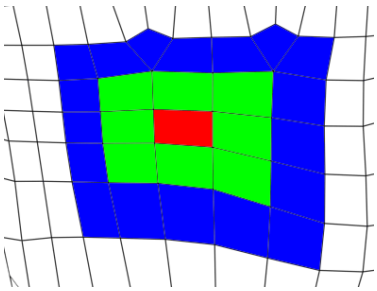
Šis bandomasis trasavimas trunka itin trumpai, nes bandymas vyksta su grubiais, nesudalytais objektais (kuo grubesnis objektas, tuo mažiau trikampių jį sudaro ir tuo mažiau skaičiavimų reikia atlikti).

Šio metodo privalumas yra tai, kad sudalinimo žingsnių kiekis nustatomas kiekvienam objektui individualiai – tai reiškia, kad norint gauti idealų glodumą toli esantį arba mažą objektą užtenka sudalyti keletą kartų, o arti esantį objektą – žymiai daugiau.

Deja, šis metodas turi ir trūkumų – esant dideliam objektų kiekiui, dažnai neužtenka atminties saugoti visus sudalintus tinklelius. Kita problema – jei matoma tik nedidelė objekto dalis (pavyzdžiui, turime žmogaus modelį, o ekrane matome tik plaštaką), padalijimas vis tiek atliekamas visam objektui. Šias problemas išsprendžia kitas mūsų algoritmas: dinamiškas tinklelio dalijimas trasavimo metu.

6.2. Tinklelio dalijimas trasavimo metu

Šis algoritmas veikia tokiu principu: vartotojas nustato (arba yra parenkamas automatiškai) adaptyvaus padalijimo žingsnį. Tada vyksta trasavimas, o sankirta su objektu vyksta taip: pirma randamas trikampis, kurį kerta spindulys. Pagal unikalų trikampio numerį yra nustatoma, kuriai sienai jis priklausė prieš objekto suskaidymą į trikampius. Priklausomai nuo pasirinkto padalijimo žingsnio, surenkame aplink mūsų sieną esančias kitas sienas:

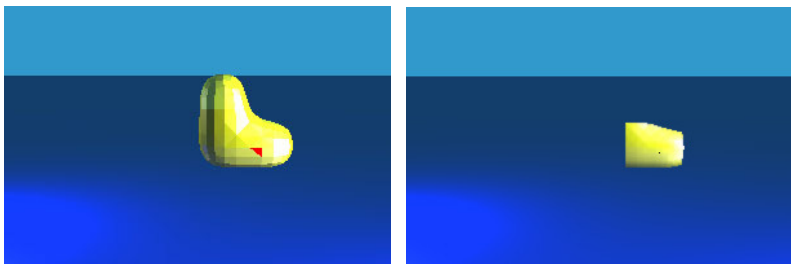


Pav. 52 – Į raudonai pažymėtą keturkampį pataikė spindulys. Jei planuojame atlikti vieną padalijimo žingsnį, imsime tik raudoną sieną ir jį supantį žalių sienų žiedą. Jei planuojame atlikti du žingsnius – reikia papildomai paimti ir mėlynų sienų žiedą. Jei norėtumėm atlikti tris padalijimo žingsnius – reikėtų paimti dar vieną žiedą, ir t.t.

Techniškai daugiakampių surinkimas vyksta remiantis HalfEdge struktūra (ja pagrįstas mūsų Catmull-Clark algoritmas) – pirmiausia surenkame raudono keturkampio taškus, tada renkame kiekvienam taškui priklausančias sienas, o pasikartojančias atmetame. Jei reikia surinkti daugiau nei vieną žiedą, operaciją kartojame rekursyviai.

Turėdami naujai sudarytą tinklelį, jį sudaliname Catmull-Clark algoritmu, ir ieškome sankirtos su spinduliu.

Šią operaciją kartojame kiekvienai objekto sienai, į kurią pataikė spindulys.



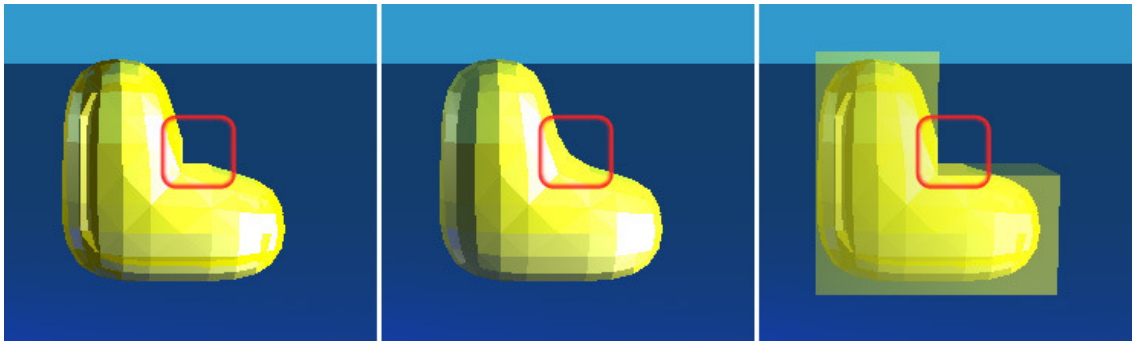
Pav. 53. Spindulys kirto raudonai pažymėtą trikampį (kairėje). Padalijimo žingsnis – 2 (t.y. reikia surinkti du žiedus). Dešinėje matome lopą po dviejų padalijimo žingsnių. Juodu tašku pažymėta vieta, kurią kerta spindulys.

Šio metodo trūkumas – reikia daug operacijų konstruojant lopą, jį dalijant ir ieškant sankirtos su spinduliu.

Pastebėjome, kad labai dažnai sekantis spindulys kerta tą patį trikampį, tada yra konstruojamas lopas, kuris yra toks pat, kaip ir prieš tai buvusio spindulio. Kilo mintis – po lopo sukūrimo ir sankirtos paieškos, pastarąjį galima išsaugoti ir skaičiuojant kitą sankirtą juo pasinaudoti – atkristų brangiai kainuojanti lopo sukūrimo operacija.

Ši idėja realizuota tokiu būdu – išsaugomas lopas kartu su trikampio, kurį kirto spindulys, numeriu. Jei kitas spindulys kerta tą patį trikampį (nustatyti galime pagal jo unikalų numerį) – naudojamas išsaugotas lopas. Sekantis žingsnis buvo patobulintas lopo saugojimas – buvo saugomas ne vienas, o du paskutiniai lopai. Tai leido sutaupyti dar daugiau laiko trasavimo metu. Praktika parodė, kad lopų išsaugojimas yra prasmingas – idealiu atveju paveiksliukas nupiešiamas apie keturis kartus greičiau.

Deja, dalijant adaptyviai, atsiranda krašto problema (pav. 54) – pradinis tinklelis skiriasi nuo galutinio. Tais atvejais, kai pradinis tinklelis dalijant tiesiog „susitraukia“, viskas yra gerai. Tačiau, kaip matome pavyzdyje, išlinkimas išeina iš pradinio tinklelio ribų, todėl jis nėra trasuojamas.



Pav. 54. Adaptyvaus padalijimo krašto problema: Pirmoje dalyje matome dviejų žingsnių adaptyvų padalijimą, pritaikytą pačiam grubiausiam tinkleliui. Antroje dalyje matome, kaip turėtų būti. Trečioje dalyje pavaizduoda priežastis: grubaus tinklelio trasavimo metu į tam tikrus taškus (esančius raudonai pažymėtoje paveiksliuko dalyje) spindulys nekliūva, todėl ta vieta nėra adaptyviai sudalijama.

Praktiniais bandymais nustatėme, kad objekto kontūrams nustatyti užtenka vos vieno-dviejų padalijimo žingsnių, todėl šią problemą spręsti geriausia būtų taip: itin grubus objektas tiesiog sudalomas keletą kartų, o tada trasuojamas adaptyviai. Teoriškai problema išlieka, tačiau kontūro

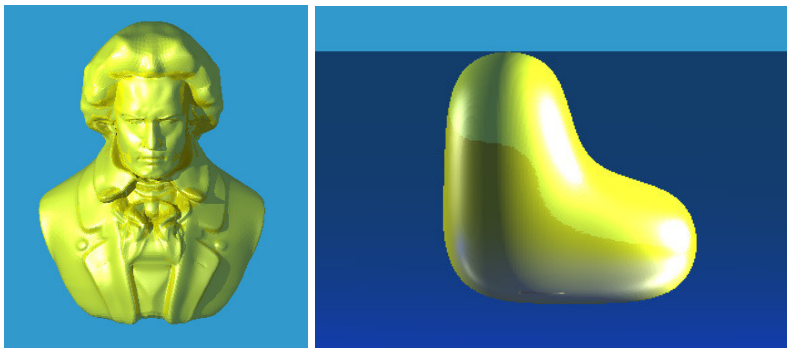
paklaida tampa tokia maža, kad vizualiai nėra pastebima.

Mūsų siūlomas sprendimas – sujungti nagrinėtų metodų geriausias savybes, taip optimizuojant kokybę, trasavimo laiką ir atminties panaudojimą. Šis sprendimas pašalintų esamų metodų trūkumus ir išryškintų jų privalumus.

6.3 Abiejų metodų sujungimas

Kadangi adaptyvus padalijimas leidžia sutaupyti atmintį laiko sąskaita, galime padaryti paprastą išvadą – jis nereikalingas tol, kol idealiam glodumui gauti atminties užtenka. Šiuos metodus galime sujungti tokiu principu – praktiniais bandymais esame apytiksliai nustatę, kiek trikampiukų telpa atmintyje (vertiname visų scenos objektų trikampių kiekį), taigi, jei idealiam glodumui reikia daugiau sudalijimo žingsnių, naudojame adaptyvų padalijimą. Tai leistų maksimaliai išnaudoti turimą atmintį, įjungiant adaptyvų padalijimą, jei jos pritrūktų.

6.4 Metodų palyginimas



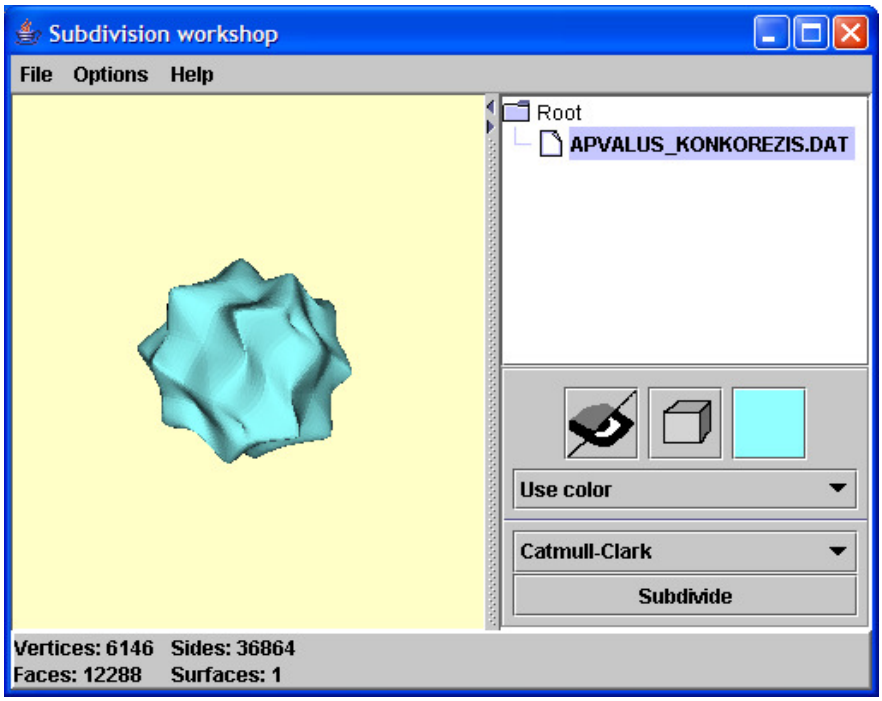
Pav. 55, 56. Testuojami objektai

	Pagal kraštinės ilgį	Naudojant adaptyvų padalijimą (su buferių sistema)	Naudojant metodų sujungimą
Pav. 55 (150x100)	1min 06 sec	7 min 15 sec	1min 06 sec
Pav. 55 (300x200)	7 min 45 sec	29 min 12 sec	7 min 45 sec
Pav. 56 (300x200)	1 min 58 sec	04 min 27 sec	1 min 58 sec
Pav. 56 (450x300)	Pritrūko atminties	12 min 14 sec	12 min 55 sec

Lentelė 1. Našumo palyginimas

7. Įskiepis padalinimo paviršių laboratorijai Workframe

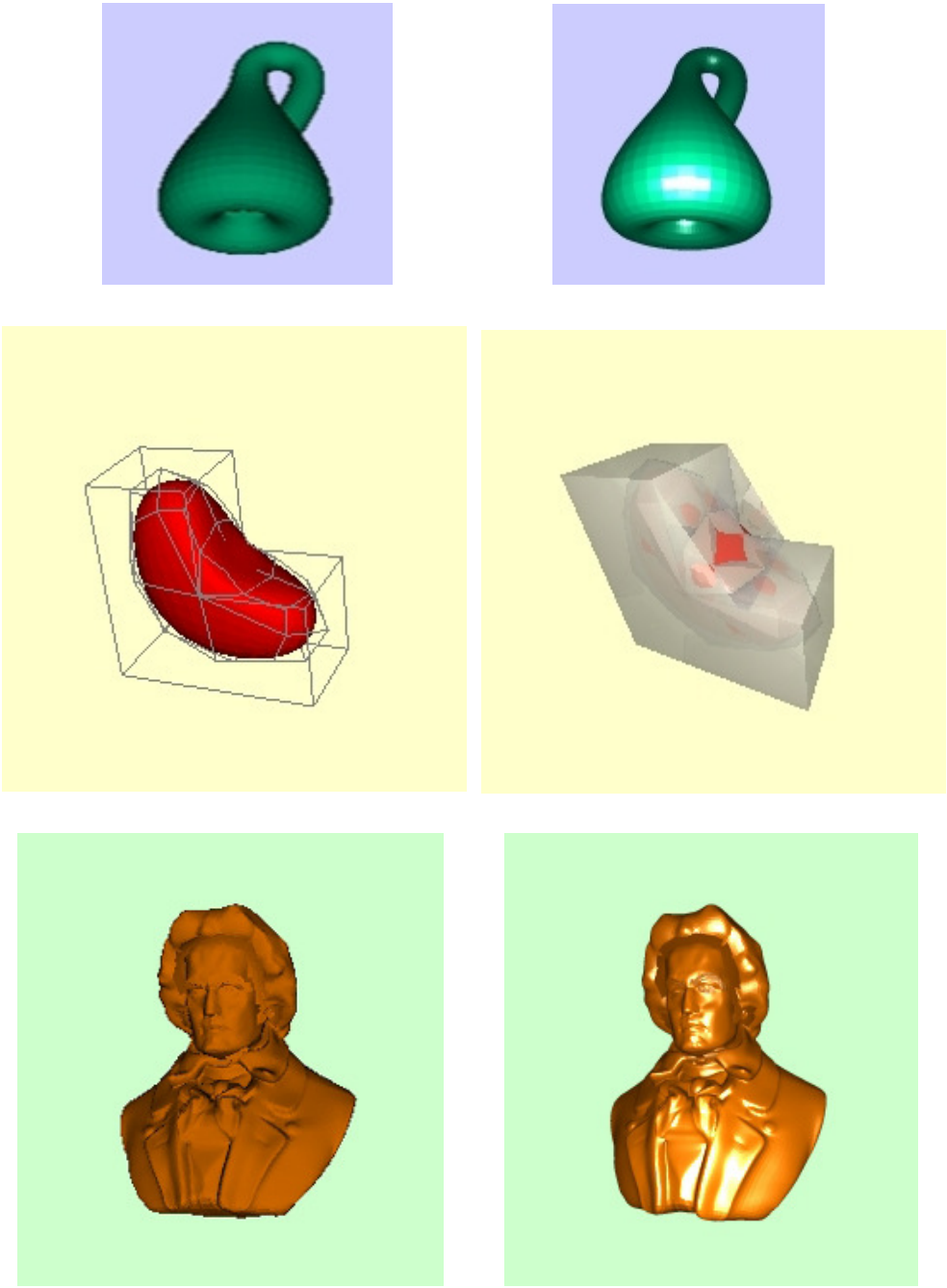
Workframe – tai studentų sukurta padalinimo paviršių peržiūros programa. Ji gali vaizduoti trimačius uždarus ir neuždarus paviršius, leidžia taikyti jiems įvairius padalinimo algoritmus (Pav.57). Paviršius galima vaizduoti keliais būdais: briaunainio, gardelės arba suglodintu pavidalu. Programa duoda galimybę lengvai peržiūrėti bei palyginti skirtingų padalinimo algoritmų taikymo rezultatus. Lankstaus įskiepių mechanizmo deka nesunku yra įtraukti papildomus padalinimo, o taip pat įvedimo ir išvedimo, algoritmus.



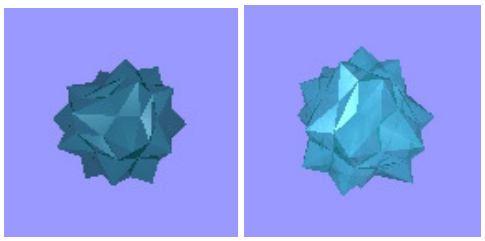
Pav. 57.

Mes sukūrėme spindulių trasavimo išvedimo įskiepį (Pav. 60). Pagal programos pateikiamus duomenis įskiepis generuoja trimačio pasaulio sceną, atlieka trasavimą ir rezultatą įrašo į nurodytą .jpg formato failą (Pav. 58). Šiuo atveju spindulių trasavimas leidžia:

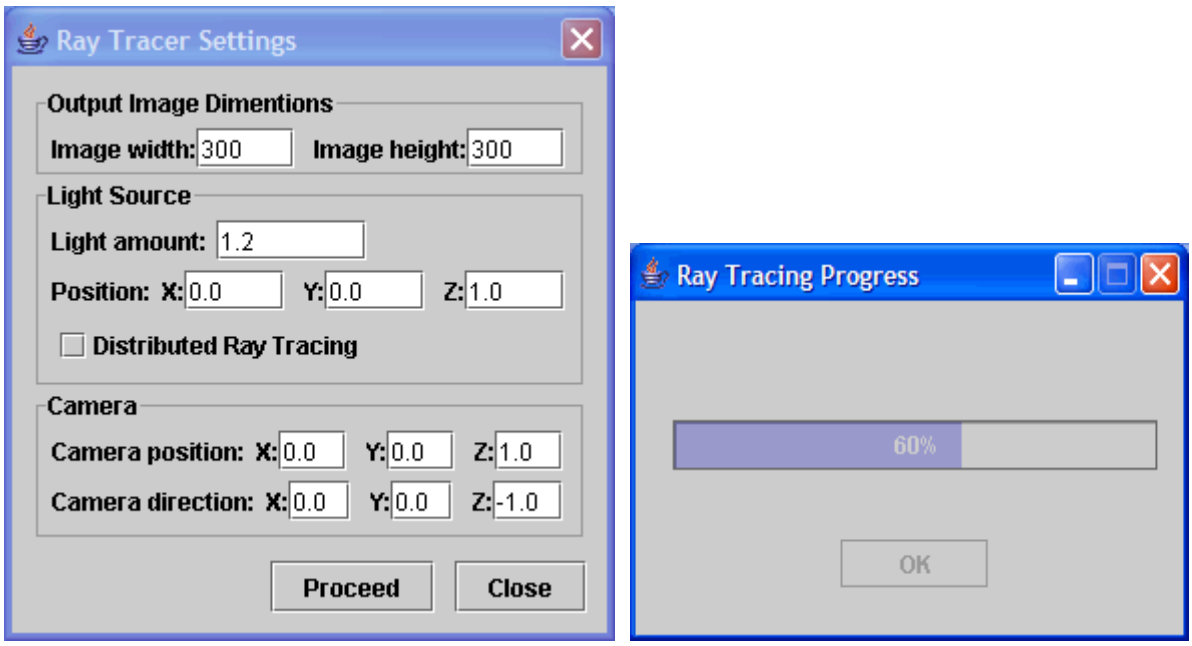
- išsaugoti peržiūrimų paviršių kokybišką fotorealistišką vaizdą.
- įsitikinti vizualiu paviršiaus glotnumu (spindulių trasavimas išryškina sienų kampus)
- suteikti paviršiams papildomas charakteristikas (permatomumą, veidrodinį atspindį) (Pav.59)



Pav. 58: Paviršiai, vieną kartą padalinti Catmull-Clark algoritmu:
kairėje – paviršių peržiūros programoje,
dešinyje – vaizdas, gautas spindulių trasavimo įskiepiu



Pav.59: Nepermatomas (kairėje) ir permatomas (dešinėje) vaizdai



Pav. 60. Spindulių trasavimo įskiepis

8. Išvados ir rekomendacijos

Šiame skyriuje mes trumpai apibendrinsime ką esame padarę, taip pat pateiksime kelias idėjas kurios galėtų būti panaudotos plečiant musų programos galimybes. Pirmiausia išvardinsime kas buvo atlikta:

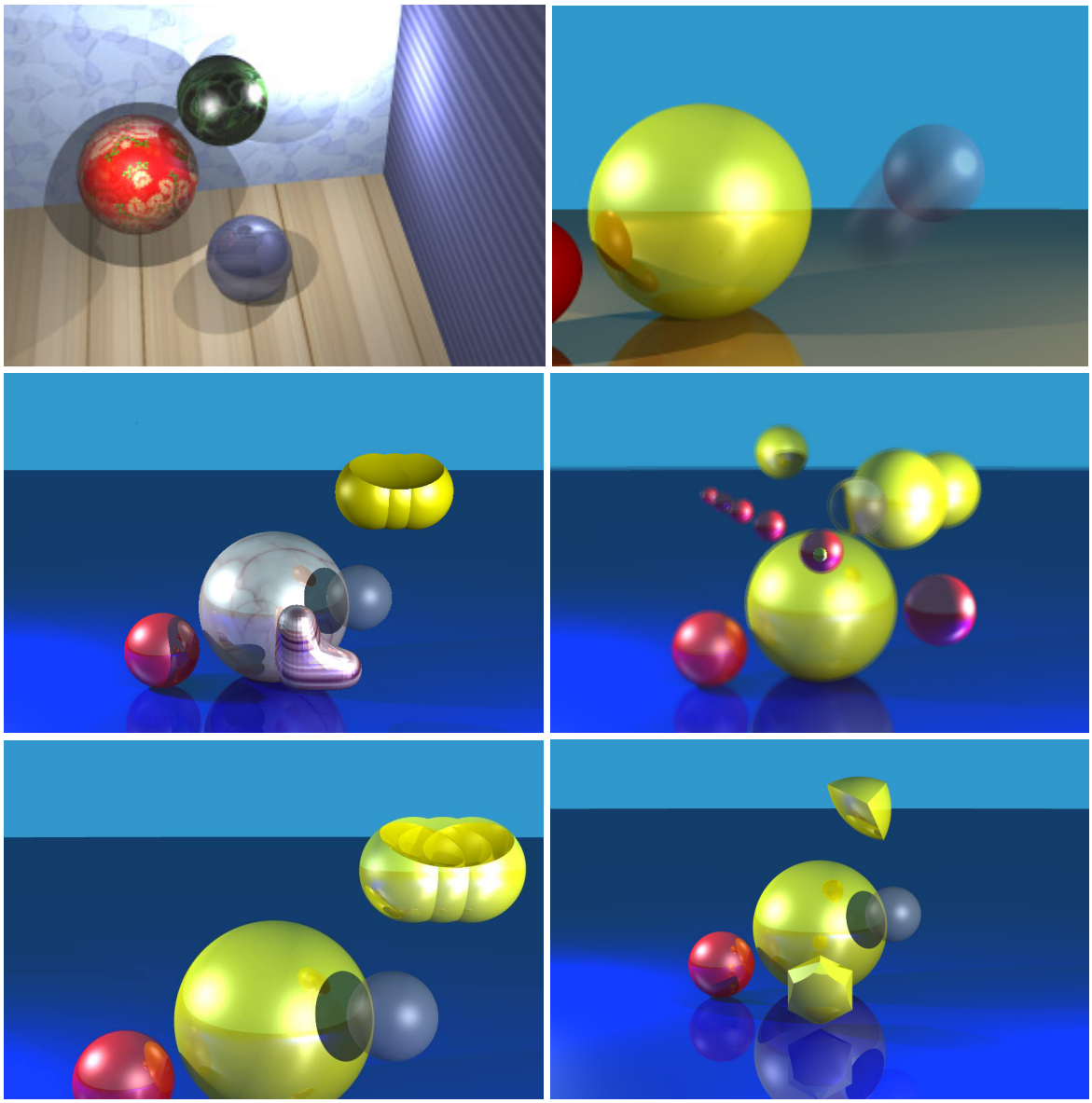
- Sukurta programa, atliekanti spindulių trasavimo algoritmą. Programai gali būti pateikiami ne tik padalijimo paviršiai, bet ir kiti geometriniai objektai – sferos, plokštumos, trikampiai ir pan. Programa surašo rezultatą į .tga formato vaizdo failą. Dauguma šį darbą iliustruojančių paveiksliukų buvo gauta pasinaudojant mūsų programa.
- Realizuoti tūrių algebros algoritmai padalijimų paviršiams. Mūsų programa leidžia atlikti paviršių sąjungą, sankirtą bei skirtumą vaizdavimo lygyje.
- Įgyvendintas dengimas tekstūromis. Realizuoti „automatiniai“ dengimo būdai – „plokščias“, „cilindrinis“, „polinis“, taip pat reikalaujantis papildomų duomenų dengimas pagal išklotinę.
- Realizuoti judesio šleifo ir kameros fokusavimo gylio vizualūs efektai.
- Realizuotas adaptyvus padalijimas. Paviršius gabalais dalijamas Catmull-Clark algoritmu tokiu būdu, kad atrodytų idealiai glotnus.
- Parašytas spindulių trasavimo įskiepis studentų sukurtam padalinimo paviršių peržiūros įrankiui.

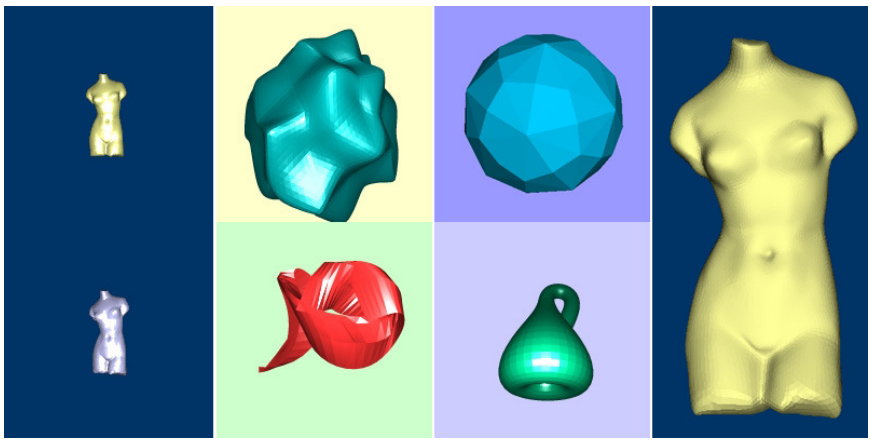
Rekomenduojamos tolimesnio darbo kryptys:

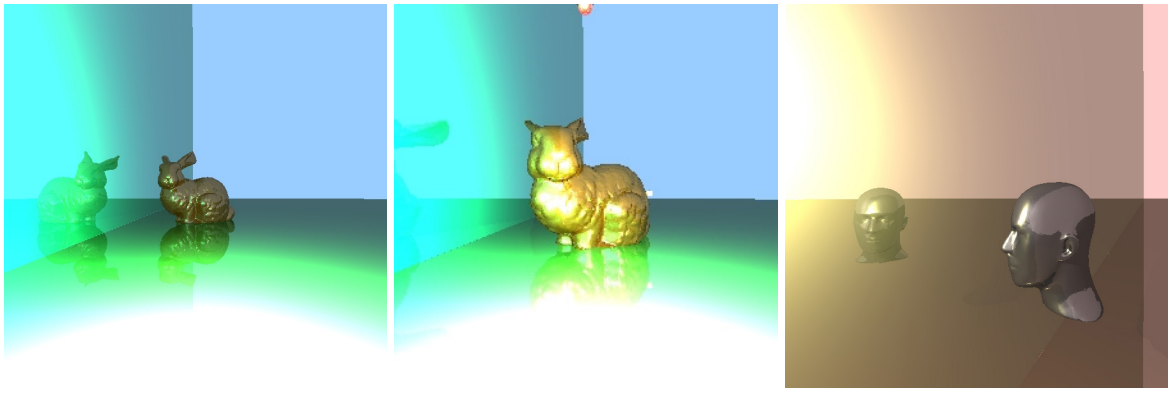
- Patobulinti dengimą tekstūromis – čia galima numatyti dvi kryptis:
  1. Optimizuoti tekstūros padalijimą atliekant dengimą pagal išklotinę. Dabar faktiškai padalijimas atliekamas dukart – paviršiui ir jo tekstūrai.
  2. Sudaryti galimybę lengviau ir efektyviau reguliuoti „automatinį“ dengimą tekstūromis – tai leistų gauti be didelių pastangų gauti pakankamai nusakomus rezultatus.
- Optimizuoti judesio šleifo efektą. Dabar siekiant gauti šį efektą turime kelis kartus atlikti viso vaizdo trasavimą. To galima būtų išvengti, traktuojant judantį kūną kartu su jo praeitu keliu kaip vieną dalinai permatomą objektą. Tokiu atveju trasavimas vyktų tik vieną kartą, ir visas procesas užimtų mažiau laiko.
- Realizuoti adaptyvų padalijimą, paremtą kitais – ne tik Catmull-Clark – padalijimo algoritmais.
- Realizuoti padalijimo paviršių trasavimą skaičiuojant ne kiekvieno trikampio normalę (kaip yra dabar), o ir gretimų trikampių normalių sumos vidurkį.

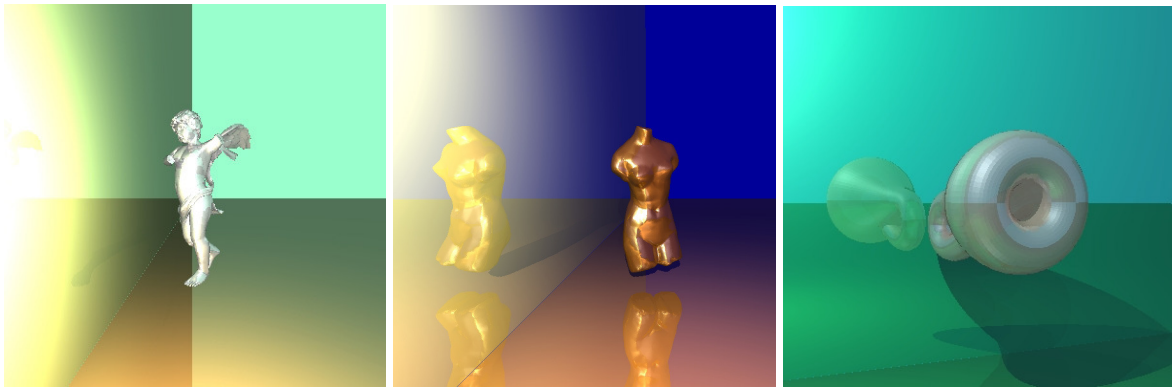
- Įtraukti į įskiepį galimybę identifikuoti kiekvieną matomą paviršių pagal jam sugeneruoti pritaikytų padalijimo algoritmų kombinaciją.
- Suteikti galimybę įskiepyje dengti paviršius tekstūromis.

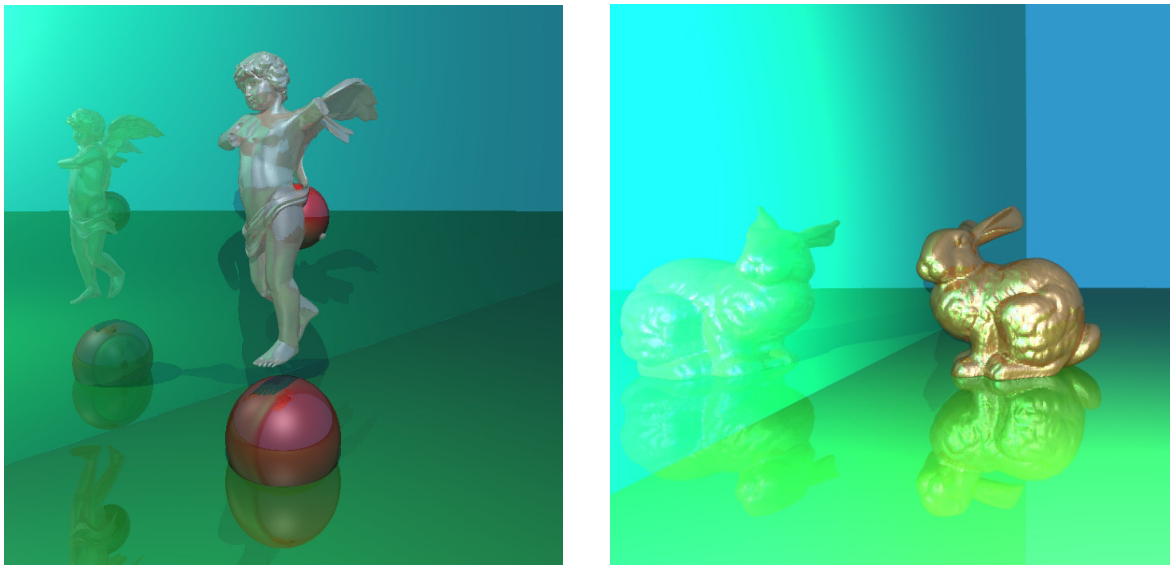
9. Galerija

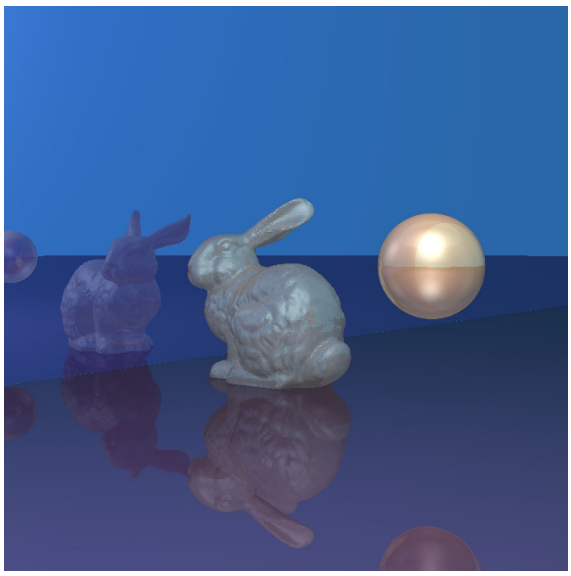












Literatūros sąrašas

[Bir02] N.S.Biriukova *Lekcii po kompjuternoj grafike*. 2002
http://www.skgtu.ru/etasks/GraphicsTask/lection6/5.htm

[BKZ98] Henning Biermann, Daniel Kristjansson, Denis Zorin *Approximate Boolean Operations on Free-form Solids* 1998

[BLV99] Laurent Balmelli, Thomas Liebling, Martin Vetterli *Computational analysis of mesh simplification using global error*. 1999

[Buc99] Jamis Buck *The Recursive Ray Tracing Algorithm* 1999
http://www.geocities.com/jamisbuck/raytracing.html

[Bus03] Samuel R.Buss *3-D Computer Graphics*. Cambridge University Press, 2003

[CC78] E. Catmull and J. Clark. *Recursively generated B-spline surfaces on arbitrary topological meshes.* Computer Aided Design, 10:350–355, 1978.

[Dam04] Andries van Dam *Raytracing* 2004

[Ham99] T. Hammersley *The Basics of Raytracing* 1999.02.11
http://tfpsly.planet-d.net/Docs/TomHammersley/raytrace.htm

[Joy96] Ken Joy *On-Line Geometric Modeling Notes*. Computer Science Department, University of California, Davis 1996.11.07
http://graphics.cs.ucdavis.edu/CAGDNotes/CAGD-Notes.html

[Kas02] Dmitrij Kascheev *Ray Tracing* 2002
http://people.ulstu.ru/~tll/d_RayTrc.htm

[Loh00] Friedrich A. Lohmüller *CSG - Constructive Solid Geometry* 2000

[NB03] Jeffry S. Nimeroff, Norman I. Badler *Texture Resampling While Ray-Tracing* 2003

[Owe99] G. Scott Owen *Ray Tracing* 1999
http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm

[Pow04] Ken Power 2004
http://glasnost.itcarlow.ie/~powerk/Graphics/Notes/node12.html

[Sab03] Malcolm Sabin *Subdivision Tutorial Workbook*. Numerical Geometry Ltd 2003

[SW99] Peter Shirley, Changyaw Wang *Distribution Ray Tracing: Theory and Practice* 1999

[ŠB95] E.V.Šykin, A.V.Boreskov *Kompjuternaja grafika*. Moskva, Dialog-Mifi, 1995