Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Képtérbeli animációs módszer 3D objektumok vonalkázott megjelenítésére

Tudományos Diákköri Konferencia

Dolgozat

Készítette
Lengyel Zoltán András

Konzulens
dr. Umenhoffer Tamás

2014. október 22.

Tartalomjegyzék

Kivonat

Ezen dolgozat egy képtérbeli vonalkázó algoritmus bemutatásáról szól. A vonalkázás egy ábrázolási stílus, mely során a kép vékony hosszú vonásokból épül fel, melyek jellemzően ceruzával vagy tussal lettek húzva. A vonások képtérbeli definiálásának legfőbb előnye az, hogy elkerülhetjük a geometriai feldolgozást, amennyiben képtérbeli adatokból dolgozunk, valamint hogy könnyű lesz feltételeket adni a képtérbeli vonássűrűségre. Ezen megközelítés esetén azonban nehézséget okoz annak biztosítása, hogy animáció során a vonás primitívek a felületekkel együtt mozogjanak, valamint hogy a vonások megfelelően érzékeltetni tudják a mögöttük rejlő 3D felület formáját.

Az algoritmus a luminancia gradienst és a görbületet felhasználva határozza meg az egyes vonások irányát és görbítését. Az időbeli koherencia biztosításának érdekében, képtérbeli elmozdulás alapján mozgatjuk külön-külön a vonás primitíveket. Az elmozgatás következtében a képtérbeli primitív eloszlás el fog torzulni. Eldobásos mintavételezés és alacsony diszkrepanciájú sorozatok segítségével a képtérben összesűrűsödött vonás primitíveket kiszűrjük, a kiritkult területeket pedig új primitívekkel töltjük fel. Az algoritmus teljes mértékben GPU-n került implementálásra, *geometry shader* és *vertex transform feedback* felhasználásával. Az eredmény valós idejű megjelenítésre alkalmas.

Bevezető

A számítógépes grafika technikáinak széles köre a nem-fotorealisztikus *(NPR)* vagy illusztratív kép alkotásra törekszik, ilyen feladat például a kézi rajz imitálása. A fotorealisztikus technikákhoz hasonlóan, az NPR technikák is háromdimenziós alakzatokkal dolgoznak, azonban a tradicionális média megjelenítését próbálják utánozni. A 3D modellezés és a modern animációs munkafolyamatok jóval hatékonyabb képalkotást tesznek lehetővé a manuális megközelítéseknél, azonban nagyon nehéz velük a klasszikus technikák jellegzetes megjelenítését reprodukálni. A háromdimenziós technológiával készült képeknek korlátozott szabadságuk van a geometriai határok, finom fény átmenetek és felületi anyagok stílusos érzékeltetése kapcsán. Bár számos megközelítés létezik melyek adott technikák utánzására képesek, gyakorlati felhasználhatóságuk korlátos.

Az animációs filmek a renderelés minőségét előnyben részesítik annak sebességgével szemben, de ez nem jelenti azt hogy a renderelési idő nem fontos. A hosszabb renderelési idő nagyban befolyásolhatja a produkciós költséget, ezért az offline renderelésre szánt NPR technikáknak is törekedniük kell a sebesség növelésére. Egy másik fontos tényezőre, az előfeldolgozási követelményekre is ügyelni kell. Produkciós környezetben a modellezés, a renderelés és a végső kompozitálás különböző folyamatok részeit képezik melyeknek mind megvannak a sajátosságaik, ezért az eltérő fázisok között az adat megosztása nehézkes lehet. Nagymértékű előfeldolgozás vagy képkockánkénti geometriai feldolgozás valós idejű alkalmazás esetén kerülendő.

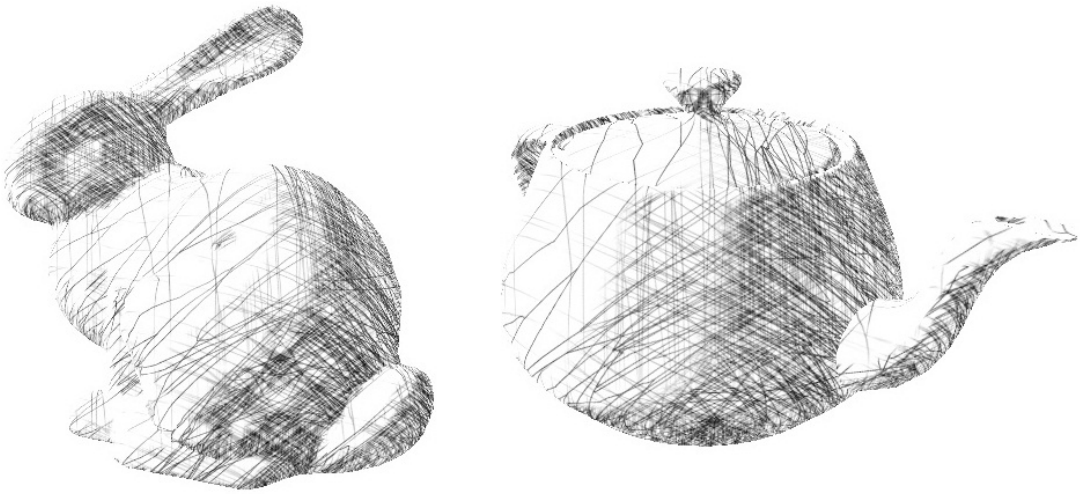
NPR technikák alkalmazása esetén mind valós idejű mind produkciós környezetben, a legnagyobb kihívás az időbeli koherencia garantálása. Nehéz biztosítani azt, hogy a stilizált vonások, illusztratív részletek vagy megvilágítási jellegzetességek követni tudják a geometriát animáció közben. Amennyiben ezek a tulajdonságok képtérben rögzítettek, egy zavaró *zuhanykabin ajtó (shower door)* nevezetű jelenséget okoznak: úgy tűnik mintha az NPR megjelenítés egy torzító üveg ajtó hatására jönne létre, amin keresztül szemléljük a virtuális színteret. Az elvárás az, hogy az előállított stílusjegyek a geometriával együtt mozogjanak hirtelen változások vagy vibrálás nélkül.

Ezen dolgozatban ceruzával és tollal készített rajzok illusztrálásával foglalkozunk. Ilyen rajzok esetén a kép vékony vonalak sokaságából épül fel. A vonások árnyékolt helyeken sűrűbben, a megvilágított helyeken pedig ritkábban helyezkednek el. Irányukkal és görbületükkel a mögöttük található geometriát érzékeltetik. A tradicionális kétdimenziós technika utánzáshoz biztosítani kell, hogy a vonások sűrűsége, hossza és szélessége képtérben konzisztens maradjon.

1. fejezet

Vonalkázási technikák

A legegyszerűbb megoldás a művészi stílusok utánzására az objektumok művészi textúrával való ellátása. Ez a vonalkázásra is érvényes: vonalkák sokaságát tartalmazó textúrát rendelve az objektumhoz, azok rajzolt szerű hatást kelthetnek (1.1. ábra). Ebben az esetben a megfelelő kép elkészítése azonban közel sem egyszerű. Mindenek előtt a textúrának ismételhetőnek kell lennie, különben a vonalak törést szenvednek a textúra szélei mentén. A kép részletességének megválasztása sem egyszerű, mivel a felülethez közeledve több vonalnak kellene megjelennie, nem pedig nagyobb vonalaknak. A konzisztens képtérbeli vonás sűrűség részletességi szint (mipmap) technikákkal és a textúrák összemosásával garantálható [1, 2, 3, 4, 5, 6]. Az egyik hátrány azonban az, hogy a maximális részletességet a mipmap szintek és a textúra felbontása korlátozza. Egy másik probléma, hogy nehéz garantálni hogy a textúrán található vonások, az objektum fő geometriai jellemzőit érzékeltetni tudják. Az ehhez szükséges UV paraméterezést manuálisan nagyon időigényes lenne elkészíteni, az elérhető automatizált módszerek pedig az objektumok geometriai feldolgozását igényelik, és sokszor nem adnak megfelelő eredményt.



1.1. ábra. *Textúra alapú hatching*

A technikák egy másik nagy csoportja új geometriát hoz létre a vonalkák számára. Ezt a csoportot két alcsoportra oszthatjuk aszerint, hogy a vonás primitíveket objektumtér-

ben vagy képtérben definiáltuk-e. Az objektumtérbeli megközelítések a vonás primitíveket közvetlenül a 3D objektumok felületén definiálják [7, 8, 9]. Ekkor a vonások, mint polygon hálók lesznek megjelenítve. A főgörbületi irányok kiszámolására is szükség van, mivel a vonások ezeket követve jobban érzékeltetni tudják a felület formáját. Ezen megközelítések előnye, hogy a vonásokat könnyű az objektumok felületéhez rögzíteni, így az időbeli koherencia könnyen biztosítható. Ezzel szemben a vonások láthatóságának számítása, valamint a vonások egyenletes képtérbeli sűrűségének garantálása nehézségeket jelent. Ezen technikák esetén a vonás primitívek ugyan azon a transzformációs csővezetéken mennek keresztül, mint a színtér többi objektuma.

A vonalkákat képtérben is definiálhatjuk [10, 11]. Ezen technikák a képtérben egyenletesen elhelyezett vonás primitívekkel dolgoznak. A vonalak irányának és görbítésének meghatározása közel sem egyértelmű ebben az esetben. A vonalaknak a mögöttük található felület formáját kell érzékeltetniük, amihez egy megfelelő képtérbeli vektor mezőt kell felépíteni. Különböző megközelítések különböző tulajdonságok alapján állítják elő ezt a vektor mezőt. Használhatják valamilyen bemeneti kép gradiensét, vagy akár képtérbeli görbületi értékeket is. Az utóbbi előállítása történhet képtérbeli adatok alapján is, amennyiben pár szükséges információ, mint például kameratérbeli mélység és normálok rendelkezésünkre állnak. A képtérbeli megközelítések esetén a legnagyobb nehézség, az időbeli koherenciának garantálása.

2. fejezet

Motiváció

A célunk egy képtérbeli vonalkázó algoritmus implementálása, amely megtartja a képtérbeli megközelítés előnyeit, miközben az időbeli koherenciára is ügyel. A vonás primitívek egyenletesen lesznek elosztva a képtérben, egy objektumhoz közeledve több vonal fog megjelenni. A vonalak irányukkal és hajlításukkal az alattuk található felület formáját érzékeltetik. A 2.1. ábrán egy példa látható egy művész által készített vonalkázott rajzra. Animáció során a vonás primitívek a felülettel együtt mozognak, hirtelen állapot változások és vibrálás nélkül. Fontos, hogy a mozgás hatására a vonalak képtérbeli eloszlása ne torzuljon el.

Kerülnünk kell a komplex geometriai feldolgozást, sőt az algoritmust függetleníteni érdemes a geometriától, mivel ekkor a teljesítmény is függetlenné válik a színtér komplexitásától. Amennyiben csak általános képtérbeli információkra támaszkodunk, az algoritmus interaktív módon felhasználható lesz, mint egy utófeldolgozó lépés. Így könnyen integrálható lesz a meglévő kompozitáló keretrendszerekbe. A megfelelő teljesítmény érdekében minden számítást a GPU-val kell elvégezni.

2.1. ábra. *Példa egy vonalkázott rajzra.*

3. fejezet

Az algoritmus bemutatása

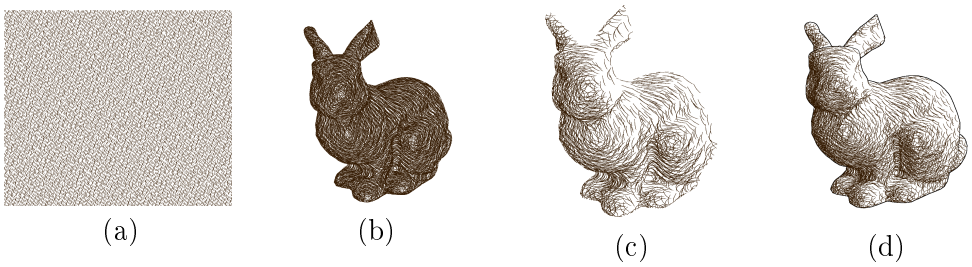
A megvalósítás a BSC-s szakdolgozatom eredményeit veszi alapul [12]. Ezen dolgozatban a vonalkázás fő alapegységei részletesen be lettek mutatva (lásd 3.1. ábra). A dolgozat hangsúlya a vonalak paraméterezéséhez használt képtérbeli tulajdonságok elemzésén volt. Három tulajdonság használhatóságát vizsgáltam meg: megvilágítás, mélység és főgörbület. A vonalak egyenként mint háromszög hálók kerülnek renderelésre. A kezdeti egyenletes eloszlás könnyen garantálható, amennyiben a primitíveket véletlenszerűen egyenletes eloszlás alapján helyezzük el. A megvilágítottság úgy érzékeltethető, ha a fényes részekről a luminancia erősségével arányos mértékben vonalakat szűrünk ki.

A meglévő munkához képest a legnagyobb előrelépés, az időbeli koherencia garantálása volt animáció közben. A vonalak egy sebességkép alapján el lesznek mozgatva, ami eltorzítja a képtérbeli vonal eloszlást. Ezen torzulás javításához, az összesűrűsödött primitíveket kiszűrjük, a kiüresedett területekre pedig új primitíveket generálunk. Ezen lépések után kiszámítjuk a megjelenítendő vonalak forgatását és hajlítását, majd csak ezután történik a tényleges renderelés. Az aktuálisan módosított részecske halmaz lesz az alapja a következő kép renderelésének. Az algoritmus bemenete képkockánkként változó képbufferek, olyan információk melyeknek a számítógépes animációban gyakran használnak, ezért a népszerűbb animációs eszközök alapból támogatják az exportálásukat. Az adatok a következők: megvilágítottság, mélység, normál, egyedi azonosító (ID) és sebességkép. A 3.2. ábrán láthatóak az algoritmus fő komponensei.

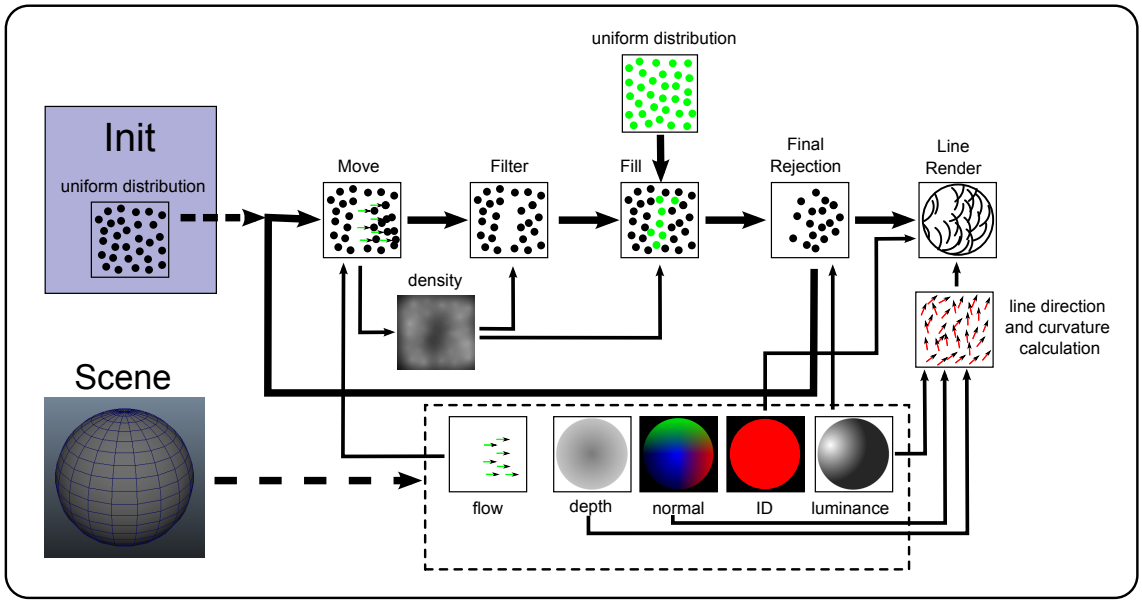
Egy másik jelentős fejlesztés, a vonalkázás alapjául szolgáló megközelítések automatikus ötvözése. Bizonyos vonalak a főgörbületi irányt, más vonalak pedig a luminancia változást használják irányuk és görbítésük meghatározásához. A következő alfejezetekben az algoritmus fő komponensei kerülnek részletes bemutatásra.

3.1. Vonal részecskék generálása

Az első lépés a lehetséges képtérbeli részecske pozíciók halmazának előállítása. Ezen képtérbeli pontokból fogjuk kihúzni a vonás primitíveket a későbbiekben. A récskéknek egyenletes eloszlásúnak kell lenniük, és a teljes képteret ki kell tölteniük. A kézi rajz véletlenszerű hatásának utánzásához fontos, hogy a részecskék elhelyezkedése ne kövessen valamilyen



3.1. ábra. *A vonalkázó algoritmus fő elemei balról jobbra: a vonalak pozíciói képtérben (a), megfelelően elforgatott és hajlított vonalak (b), a fényesség szerinti elhagyjás után megtartott vonalak (c), él felismerés útján nyert kontúrok (d).*



3.2. ábra. ***Az időbeli koherenciát is biztosító algoritmus fő elemei. Az inicializálás során egyenletes képtérbeli sűrűségű részecskék lesznek generálva. Minden képkocka elkészítése során bementi képek készülnek a normálokról, mélységről, megvilágításról, objektum ID-ről, és a képtérbeli sebességről. A részecskék a sebesség kép alapján el lesznek mozgatva a képtérben. Ennek hatására a sűrűség egyenletlen lesz, amit a részecskék szűrésével majd új véletlen részecskék hozzáadásával javítunk. A vonalak paraméterezése az előző állapotuk függvényében történik. A megvilágításnak megfelelően számos részecske nem kerül kirajzolásra. A végső lépésben a részecskékből háromszöghálót készítünk, melyek vonal textúrát kapnak.***

felismerhető mintát.

A véletlen részecskék generálása az algoritmus inicializáló szakaszában történik. Ekkor egy nagy méretű buffert töltünk meg véletlen pozíciókkal, Halton sorozat alkalmazásával. Ezen buffer mérete jóval nagyobb mint az egy képkockán elvárt vonal mennyiség, mivel a későbbiekben az animáció következtében újabb és újabb véletlen pozíciókra lesz szükségünk. Amennyiben a buffer végére érünk, ismét annak elejéről kezdjük a pozíciók felhasználását. Minden egyes részecskéhez, a két pozíció koordináta mellett tárolunk egy harmadik egységnyi véletlen értéket is, a prioritást. A prioritásnak a primitív szűréseknél

lesz szerepe, de egyelőre csak annyit érdemes tudni róla, hogy a pozíciókhoz hasonlóan a képtérben egyenletes eloszlással rendelkező érték. A folyékony animáció eléréséhez egyéb vonalra jellemző dolgok is el lesznek tárolva, mint például vonal hossz, irány és görbület. Ezen vonal állapot változók azonban egy másik bufferbe kerülnek, és csak a vonás primitív létrehozásakor jönnek létre, nem inicializáláskor.

3.2. Részecskék mozgatása

Amint a kamera vagy az objektumok mozogni kezdenek, biztosítanunk kell hogy a vonás primitívek a felülettel együtt mozognak. Ez elengedhetetlen a *shower door* hatás elkerülése végett. Mivel képtérben dolgozunk, kézenfekvő megoldás a képtérbeli sebesség kép, ismertebb nevén az *optical flow* felhasználása. Az algoritmusunk 3D-s színterekre van tervezve, ezen környezetben lehetőség van a sebesség kép egyszerű és pontos előállítására. Az előző képkocka megfelelő transzformációs mátrixainak megőrzésével megkaphatjuk a vertexek előző képkockabeli pozícióját is, ami után az elmozdulás egy egyszerű vektor különbséggel számítható.

A sebesség kép alapján, a vonás primitíveket el tudjuk tolni a képtérben minden egyes képkocka renderelése előtt. A transzformáció után egyes részecskék nem mozdulnak el, mások a képtér egy új pontjára kerülnek, bizonyos részecskék pedig ki lesznek tolva a látható képtérből. Már egészen kis elmozdulás esetén is, a primitívek eloszlása egyenlőtlenné válik.

3.3. Sűrű területek szűrése

A részecskék elmozgatása után helyre kell állítanunk az egyenletes képtérbeli eloszlást. Ennek érdekében, először is a túl sűrűn elhelyezkedő részecskéket kiritkítjuk. Ez azt jelenti, hogy csak azokat a részecskéket tartjuk meg, amik pozíciójuk körüli helyi sűrűséget nem növelik meg túlságosan. A sűrűség frissítése minden egyes részecske kirajzolása után nem lehetséges valós idejű környezetben, mivel ekkor a hardver nem tudná párhuzamosítani a részecskék renderelést. Ezen oknál fogva, a részecskék eldobásának döntését csak a helyi sűrűség értékekre alapozzuk, az aktuális képkocka előállítása során már kirajzolt részecskék a döntést nem befolyásolják. A megvalósításához az eldobásos mintavételezés elméletére támaszkodunk.

A klasszikus eldobásos mintavételezés problémában, egy elvárt eloszlásnak megfelelően szeretnénk mintavételezni, de nem tudjuk az inverziós módszert alkalmazni. Az eldobásos mintavételezés esetén kiválasztunk egy jól ismert és megfelelően skálázott eloszlást, amit könnyű mintavételezni, és ami a kívánt eloszlást felülről korlátozza. A legegyszerűbb az egyenletes eloszlást használni. Ha minden egyenletesen elosztott mintához egy véletlen prioritás értéket rendelünk, majd eldobjuk a mintát ha annak prioritása a skálázott elvárt sűrűség fölött van, akkor a maradék minták az elvárt eloszlásnak megfelelőek lesznek.

A mi esetünkben az egyenletes eloszlás az elvárt és a részecskéink eloszlása egyenlőtlen. Ezért a minta sűrűsége lokálisan meghaladhatja az elvárt egyenletes eloszlást. A részecskék szűréséhez tudnunk kell azok szomszédságában a sűrűséget. Ennek kiszámításához apró

köröket renderelünk a részecskéink pozícióiba, amiknek a középponttól véve lineáris lecsengésük van. Ezen körök additív összemosása után egy közelítő sűrűség függvényt kapunk. A körök sugara és kitöltésüknek erőssége jelentősen befolyásolja a sűrűség kép simaságát, ezen paraméterek empirikus úton lettek meghatározva. A paramétereket befolyásolja az aktuálisan használt felbontás és az elvárt vonal mennyiség. Az alábbi kifejezéseket a paraméterek finomra hangolása után kaptuk, ezek jól működtek eltérő felbontások és vonal számok esetén is:

$$\text{snippet\_size} = \sqrt{\frac{\text{number\_of\_pixels}}{\text{desired\_line\_number}}} * \frac{6.5}{\text{window\_resolution}},$$
$$\text{snippet\_power} = 0.04 * \text{linearFalloff}().$$

A sűrűség kép segítségével minden egyes részecskét megvizsgálunk, és eldobjuk őket amennyiben a prioritás értékük a helyi sűrűséggel skálázva az elvárt egyenletes sűrűség fölött van. Mivel a prioritások eloszlása egyenletes, a vonalak is egyenletesen lesznek kiszűrve, így a sűrű területeken létrejövő új sűrűség egyezni fog az egyenletes sűrűséggel. A szűrés után a részecskék prioritását vissza kell skálázni az egység tartományba, mivel a magas prioritású részecskék ki lettek szűrve. Ez biztosítani fogja, hogy a prioritás értékek eloszlása ismét egyenletes legyen.

3.4. Ritkás területek feltöltése

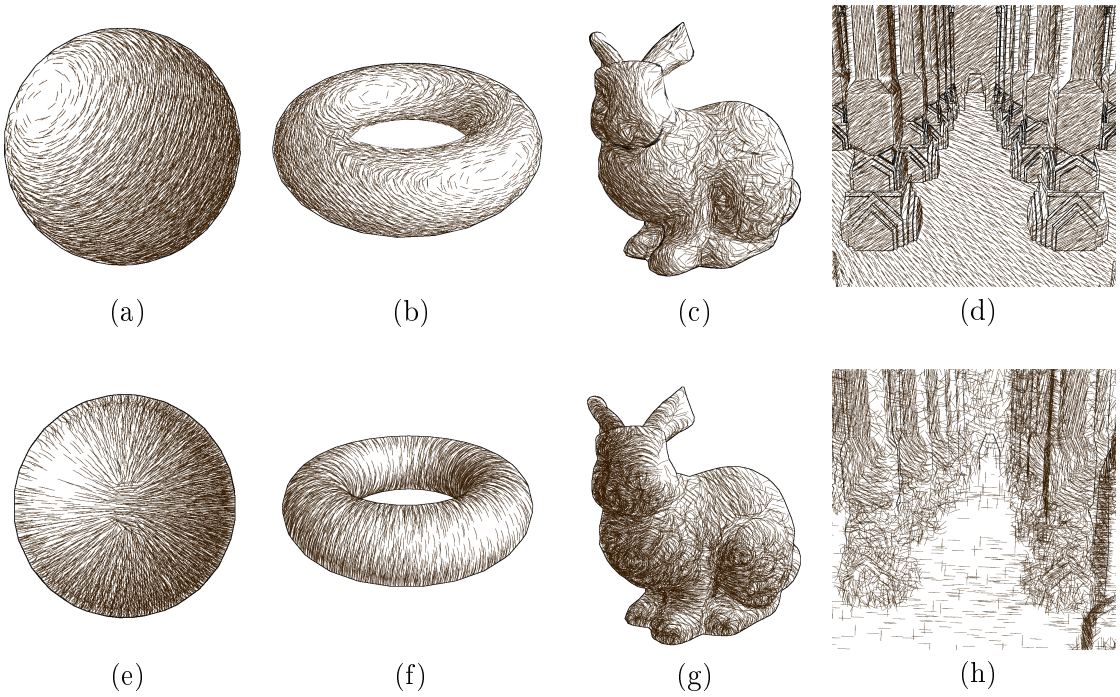
A szűrés folyamata után a kiüresedett területeket fel kell töltenünk. Ezt úgy oldjuk meg, hogy véletlen minták egy új halmazát próbáljuk meg elhelyezni a képtérben. Ismét az előző sűrűségképet fogjuk használni, új részecskéket generálunk, és az elhelyezni kívánt új véletlen részecskékből azokat tartjuk meg melyeknek a prioritása a sűrűség érték felett van. Így a ritkább területeken több részecskét tartunk meg, míg ahol a sűrűség már megfelelő minden részecskét eldobunk. A részecskék prioritását ismét az egység tartományba kell skálázni. Az új részecskék az eredeti szűrt részecske buffer végére fognak kerülni. Ezen a ponton ismét egyenletes képtérbeli eloszlással rendelkezünk, miközben a primitívek ha tehetik az objektumok felületével együtt mozognak. Ez azt az illúziót kelti mintha a vonalak objektumtérben lennének definiálva, így ez egy meghatározó lépés az időbeli koherencia garantálásához.

3.5. Megvilágítás

A vonalkázás sűrűsége a fény-árnyék jelenségek érzékeltetésére is alkalmas. A megvilágított terülteken ritkábban találhatóak vonalak a sötét területeknél. Ismét az eldobásos mintavételezés klasszikus problémájával találkozunk. Minden részecskét, aminek a prioritása az invertált megvilágítottsági érték alatt van eldobunk. Ebben az esetben a részecskéket ténylegesen nem dobjuk el, csak azok kirajzolását kihagyjuk. Tehát a megvilágítottság miatt nem kirajzolt részecskék továbbra is a megjelenítéshez használt bufferben maradnak.

3.6. Vonalak iránya és hajlítása

A vonalak renderelése előtt, meg kell határoznunk azok irányát és görbítésüknek mértékét. Ezen tulajdonságokat úgy kell megválasztanunk, hogy a vonalak minél jobban érzékeltetni tudják a mögöttük található geometria formáját, valamint hogy utánozzuk azt, hogy a művész milyen módon tenné ezt. A BSC-s szakdolgozatomban [12] arra az eredményre jutottunk, hogy nincsen olyan tulajdonság aminek segítségével minden geometriai típust jól lehet érzékeltetni. A 3.3 ábra szemlélteti különböző megközelítések különböző objektumokon milyen eredményt adnak. Általánosságban elmondható, hogy a meghatározó görbülettel rendelkező objektumok esetén a legjobb érzékeltetést az adja, ha a vonalak a főgörbületek mentén haladnak. Ezen elhelyezési mód a művészek rajzaiban is természetes.

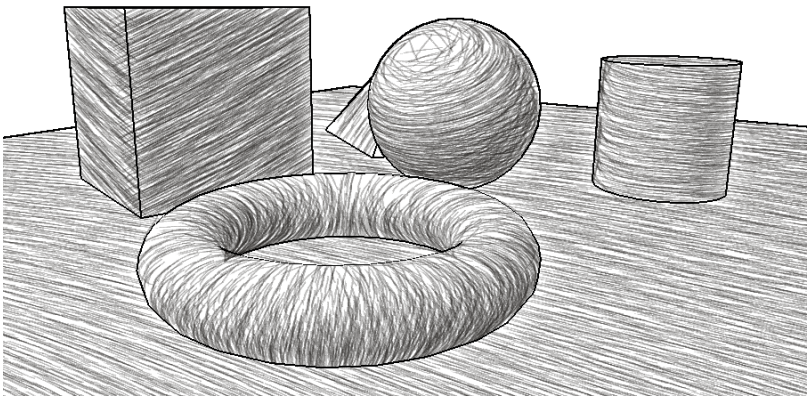


3.3. ábra. *Vonalkázás fényesség (fent) és főgörbületek (lennt) felhasználásával. Különböző geometriai formáknak különböző megközelítésre van szükségük a megfelelő érzékeltetéshez.*

A főgörbület számítható a görbületi tenzorból, ami a mélység mező Hesse mátrixa. Mivel a felületi normálok meghatározzák a mélység gradienst, a Hesse-féle mátrix a felületi normálok irány menti deriváltjai alapján az alábbi módon írható fel:

$$H = \begin{pmatrix} \frac{\partial \vec{n}}{\partial \vec{u}} \cdot \vec{u} & \frac{\partial \vec{n}}{\partial \vec{v}} \cdot \vec{u} \\ \frac{\partial \vec{n}}{\partial \vec{u}} \cdot \vec{v} & \frac{\partial \vec{n}}{\partial \vec{v}} \cdot \vec{v} \end{pmatrix}$$

ahol az $\vec{u}$ és $\vec{v}$ ortonormált tangens vektorai a felületnek. A mi esetünkben ezek az $\vec{x}$ és $\vec{y}$ képtérbeli vektorok felületre vett vetületei. Azaz a Hesse, a képtérbeli iránymenti deriváltjai a képernyőre vetített normál vektoroknak. A gradienst Sobel szűrő segítségével számítjuk. A mátrix sajátértékei és sajátvektorjai meghatározzák a maximális és minimális normál görbületet és a hozzájuk tartozó irányt, így a főgörbület nagyságát és irányát is.



3.4. ábra. *A vonalak forgatásához és görbítéséhez vonalanként fogunk döntést hozni, a fényességet és a főgörbületet használó megközelítés között.*

A tesztelések során arra jutottunk, hogy a luminancia gradiens is jól használható tulajdonság, és alkalmazható olyan esetekben amikor a főgörbületi irány nem. Ilyen esetek például a sima felületek, melyeknek egyáltalán nincsenek görbületeik, vagy a gömb alakú formák ahol nincsen főgörbületi irány. A luminancia gradiens azonban nem kínál megfelelő érzékeltetést a főgörbülettel rendelkező formák esetén, mint például a tórusznál, ahogyan azt a 3.3 ábra is szemlélteti.

Ahhoz hogy a két megközelítés előnyeit kombináljuk, mindkét tulajdonságot kiszámoljuk és a luminancia alapút használjuk azokban az esetekben, amikor a főgörbületi irány bizonytalannak tűnik. Ezt úgy tudjuk eldönteni, hogy a maximális és minimális görbület a felületen közel azonos, ezért meghatározó főgörbület nem jelölhető ki. A 3.4. ábra ezen automatikus megközelítés választó eredményét mutatja be.

3.7. Állapot animáció

Most már tudjuk, hogy egy adott részecskéből növesztett vonalat hogyan kellene forgatni és hajlítani, tehát minden információ rendelkezésünkre áll ahhoz, hogy a végső vonalkázott képet elkészítsük. Azonban ha minden képkockában az éppen kiszámolt paraméterekkel rajzoljuk ki a vonalakat, hirtelen állapotváltozásokat fogunk tapasztalni, amik az időbeli koherenciára rossz hatással vannak. Ezen hirtelen állapotváltozásoknak a legtermészetesebb esete a vonalak kiszűrése és hozzáadása során jelentkezik. Ekkor ugyanis a kiszűrt vonalak hirtelen eltűnnek, az új vonalak pedig azonnal megjelennek a képernyőn. Egy másik egyszerű állapotváltozást tapasztalhatunk a statikus vonalak esetén is, amikor a fényforrás elmozdul.

Az időbeli koherencia érzetén sokat segíthet, ha nem engedjük hogy a vonal állapotok hirtelen megváltozhassanak. Folyékonyabb hatást kelt, ha a vonalak a középpontjukból fokozatosan kinőnek illetve fokozatosan összemennek életciklusuk során. Hasonlóan az irány és hajlítás paraméterek is csak idővel fokozatosan változhatnak meg. Ennek megvalósításához azonban szükségünk van a vonalak előző képkockabeli állapotára. Mivel egy pont csak 4 float változó tárolására képes, egy új buffert hozunk létre az aktív vonalak állapotának

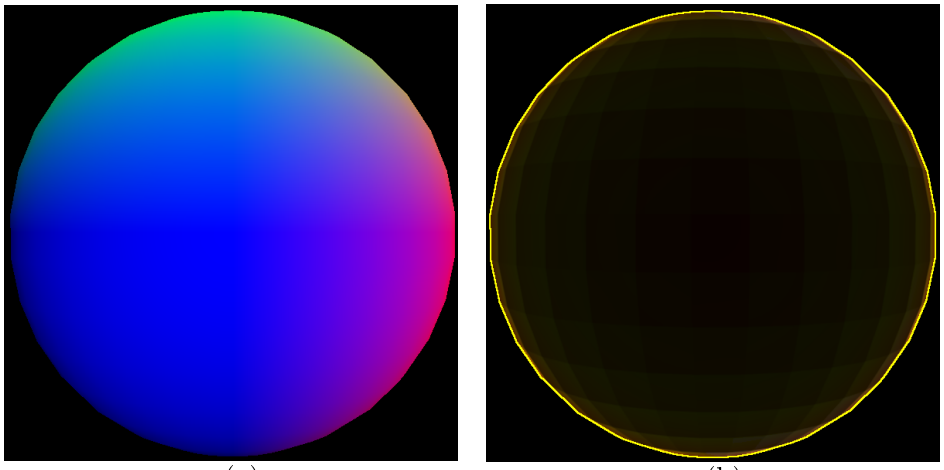
tárolásához. Itt az alábbi értékeket tároljuk el: irány, hajlítás, objektum ID, hossz. Az objektum ID tárolása azért lesz hasznos, mert felismerhetünk vele olyan eseteket amikor egy részecske egy másik objektumra tolódik át a mozgatás során.

Az algoritmus minden képkocka előállítása során a kiszámolt vonal paramétereket ezen állapot bufferbe lementi, magához az új állapot kiszámolásához pedig felhasználja az előbbiekben lementett paramétereket. Az eltelt időt is figyelembe véve így könnyen megoldható lesz, hogy a vonalak egy maximális sebességgel változtathassanak csak irányt, valamint egy maximális sebességgel nőjenek meg vagy menjenek össze, hozzáadás vagy szűrés esetén. Meggondolandó azonban, hogy a megjelenítés és eltüntetés késleltetése milyen hatással lesz a torzult eloszlás javításának menetére. A megjelenítés késleltetése nem okoz gondot, mivel az újonnan hozzáadott vonalak bár nem lesznek láthatóak azonnal, a sűrűség képen mégis teljes értékű részecskének tekinthetjük őket a hozzáadásuk pillanatától kezdve. A kiszűrés késleltetése azonban számos problémát vet fel. Ekkor ugyanis az eloszlás javítása érdekében kiszűrt vonalak a következő képkockában is ott lesznek még, tehát gátoljuk az eloszlás helyreállítását. Mivel a mozgás hatására tovább sűrűsödő részecskék nem tűnnek el azonnal, a szűrő újabb és újabb részecskéket próbál eldobni a környékükön, aminek hatására a kelleténél nagyobb területek üresednek ki. A probléma kezelésének egyszerű módja, ha az egyszer már kiszűrt részecskéket nem vesszük bele a sűrűség képbe. A már kiszűrt részecskék megkülönböztetéséhez egy új változót kell felvenni, aminek még van hely az eredeti részecske bufferekben (3.1. ábra). Ezek után a szűrő kiszűrt állapotba billenti az eldobandó részecskéket, amik bár fokozatosan tűnnek csak el, az eloszlás javításába nem zavarnak bele.

positionX	positionY	priority	filtered
direction	curve	objectID	length

3.1. táblázat. *Egy vonal összes adatának tároláshoz 2 db 4D-s vektorra van szükség.*

Érdemes figyelmet fordítani az állapot animáció sebességének megválasztására. Bár a célunk a hirtelen változások elkerülése, a lassú elhúzódó animáció is zavaró lehet. Konstans sebességet alkalmazva számos kellemetlen hatást tapasztalhatunk. A képtérbe beérkező új felületeken csak fokozatosan nőnek meg a vonalak, ezért a folyamatos mozgás hatására az új területek üresek lesznek. Kellően intenzív mozgás esetén egyáltalán nem lesz idejük a vonalaknak megnőni, mivel az új vonalak nagyon hamar ki is lesznek szűrve. Intenzív mozgatás közben, ezért nem fognak látszódni a vonalak. A megjelenés konstans sebességének növelése nem jó ötlet, mivel az általános esetekben úgy túl gyors lenne az animáció. A problémán sokat segíthetünk az animáció sebességének megfelelő skálázásával. Kézenfekvő megoldás a sebesség kép használata, tehát az animáció sebességének növelése a részecske sebességével arányosan. Így intenzív mozgás esetén a nagy sebesség vektor miatt, a vonalak szinte azonnal megjelenhetnek és a kép nem fog kiüresedni. Ez a skálázás azokon az eseteken nem segít, amikor az intenzív mozgás hirtelen megáll vagy nagyon lelassul. Ekkor ugyanis az utoljára megjelenő felületre felvett részecskéknek már nincs nagy sebességük, így előfordulhat hogy nagy üres területeket látunk. Célszerű lenne aszerint is skálázni az



(a) (b)

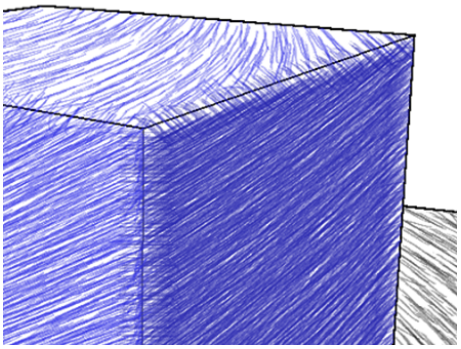
3.5. ábra. *A bal oldalon a normál kép látható, a jobb oldalon annak a deriváltja. Bár az interpolált normálok simák, a derivált képen megjelenik a tesszelláció.*

animáció sebességét, hogy a részecske környékén mennyi vonal látható. Ezzel pont azt fogalmaznánk meg, hogy nem szeretnénk nagy kiüresedett területeket látni. A sűrűség kép eddigi formájában ezen információ szolgáltatására nem alkalmas, hiszen a frissen felvett részecskék is teljes értékűek a sűrűség szempontjából, ezért a sűrűség akkor is megfelelő ha a vonalak még nem látszódnak. Bevezetünk ezért egy új képtérbeli adatot a 'látható sűrűséget'. A látható sűrűség számítása, az eredeti sűrűségétől annyiban különbözik csak, hogy a kép előállításához elhelyezett körök súlya skálázva van a részecske aktuális hosszával. Így bár az eredeti sűrűség minden képkockában megjavul, a látható sűrűség alacsony értékeket fog adni az olyan területekre, ahol sok új vonal keletkezett. Ezen értéket használva az animáció sebességének skálázására, felgyorsíthatjuk a vonalak megjelenítését nagy kiüresedett területeken. A látható sűrűséget az eredeti sűrűség előállítása során a kép egy másik csatornájában kiszámolhatjuk, így nincsen jelentős teljesítménybeli költsége.

Állapot vibrálások tapasztalhatóak maga a részecske mozgatás következményeként is. Ez azért fordulhat elő, mert a részecske elmozgatása során a részecske csak korlátos pontossággal tudja követni a hozzá tartozó felületi pontot az objektumon. Ez különösen akkor jelent gondot, ha a részecske polygon határ közelében van. Az egész jelenség oka az, hogy az objektumok normáljai jellemzően elsőrendű közelítéssel vannak kiszámolva. Ekkor bár az interpolált normál kép még sima lesz, annak gradiensén ismét megjelenik az objektum tesszellációja (lásd 3.5. ábra). Mivel mi a vonalak állapotának számítását a fényesség és a normálok gradiensére alapoztuk, a polygon határok mentén éles állapot ugrások jelenhetnek meg, az objektum szögletességének megfelelően. Ezért előfordulhat, hogy a polygon határon lévő vonal két vonal állapot között fog vibrálni.

A problémán valamennyire segít, hogy produkciós környezetben a modellek kellően tesszelláltak. Valós idejű környezetben az éles átmenetek redukálásához, próbálkozhatunk simító szűrők használatával. A szűrőt a normál képre alkalmazzuk a deriválás előtt. Mivel a fontos részleteket nem akarjuk elveszíteni, valamint a különböző objektumokat egymással valamint a háttérrel nem szabad összemosni, élmegtartó szűrőre van szükség. Az élmegtartó

szűrő sajnos nagyon teljesítmény igényes, főleg mivel a látható eredményhez többször kell futtatni minden képkocka során. Az olyan esetekre amikor az objektum valóban szögletes *(pl.: kocka)*, sem az élmegtartó szűrő sem a magas tesszelláltság nem tud megoldást nyújtani. Az állapot átmenet ekkor ugyanis természetes módon éles, ha mindenáron megpróbáljuk elkenni a normálokat az érzékeltetés is sérülni fog (lásd 3.6. ábra).



3.6. ábra. *Kellően engedékeny élmegtartási paraméterek mellett bár az élek menti állapot vibrálás csökkenthető, a vonalak elkezdenek ráfordulni a kocka éleire.*

Az élmegtartó simító szűrők helyett egy 'anti-flicker' névre keresztelt technika került bevezetésre. Az anti-flicker a vonalak irányának és hajlításának számítása során opcionálisan elvégezhető extra lépés. Bekapcsolása esetén a vonal paraméterei nem csak a részecske pozíciójában található képtérbeli pontot használva lesznek kiszámítva, hanem számos, a részecskéhez közeli pont alapján is. A közeli vonal állapotok közül azt fogjuk használni, ami a legközelebb esik az előző képkockabeli állapothoz, tehát azt ahol a legkisebb az eltérés a vonal iránya és hajlítása között. Így a polygon határra került részecskék, rezgés helyett tartják az első állapotot amit felvettek. A gyakorlatban 12 közeli pontra lesz kiszámolva a vonal irány és hajlítás egy helyett, ami bár költséges, az élmegtartó szűrő használatánál még mindig gyorsabb.

3.8. Végső render

A vonalak irányának és görbítési mértéküknek meghatározása után, a vonalak görbített és textúrázott háromszög hálóként lesznek kirajzolva, úgy hogy a hardveres keverés (blending) be van kapcsolva. A vonalak ortografikus vetítés nélkül direkt kirajzolásra kerülhetnek, mivel a képtérbeli mintáknak nincs szükségük 3D-s transzformációkra, valamint láthatósági tesztekre sem. Egy objektum ID kép segítségével a vonalakat pixelenként vághatjuk, hogy azok ne haladjanak át objektum határokon. Ez hasznos lehet amennyiben hosszú vonalakat használunk, de rövidebb vonalak esetén a kisebb átlógások az objektumok határainál természetes hatást keltenek.

4. fejezet

Implementáció

A vonalkázó algoritmust egy példaalkalmazásban implementáltuk, OpenGL és GLSL shaderek felhasználásával. Az algoritmus bemenetét képező bufferek, mint például a mélység vagy normál kép, minden egyes képkockánál ki lesznek számolva. Az implementáció könnyen módosítható úgy, hogy egy offline renderelő kép folyamát dolgozza fel. A bufferek feldolgozása, mint például a gradiensek vagy a főgörbület számítása, egy teljes képernyőt kitöltő négyszög fragment árnyalójában történik.

Az algoritmus a legtöbb lépésfolyamata során részecskék dinamikus tömbjét dolgozza fel. Ezen feldolgozás geometria árnyalóval (geometry shader) történik. A shaderek bemenete pont primitívek, melyek a részecske adatait tárolják. A geometry shader egy ilyen részecskét lát, aminek adatait módosíthatja ha szükséges, vagy teljesen el is dobhatja azt. A geometry shader kimenete ugyancsak egy pont primitív, amit tovább lehetne küldeni a raszterizálónak, de a mi esetünkben ez a primitív egy másik bufferbe lesz vezetve tényleges rajzolás nélkül. A kimenet visszacsatolható egy adott GPU-n található buffer adott pozíciójába is *(transform feedback)*, így két buffer tartalmának összevonására is van lehetőség. Ezen összevonásra a részecske feltöltő fázisban van szükség. A bufferekbe aktuálisan kiírt részecskék számát lekérdezhetjük az OpenGL API segítségével, így mindig tudhatjuk mennyi az aktuális vonal szám.

A sűrűség kép előállításához, a geometry shader kis négyszögeket növeszt minden részecskéből, melyeknek a középpontos lecsengését a fragment shaderben számoljuk ki. A végső renderelésnél a geometry shader egy (a vonalat jelképező) hajlított háromszög hálót generál, ami ebben az esetben tovább lesz küldve és raszterizálva lesz. Az implementációnk minden számítást és szükséges adatot a GPU-n tart, a bemeneti adatok feldolgozásától kezdve a végső megjelenítésig.

5. fejezet

Eredmények és tervek

Mivel az algoritmusunk teljes mértékben képtérben dolgozik, a teljesítmény nem függ a feldolgozott geometria komplexitásától, csak a képek felbontásától. Azt azért érdemes megemlíteni, hogy a bemeneti bufferek előállítása újabb render meneteket igényelhet, így a kombinált teljesítmény végső soron mégis függeni fog a geometria komplexitásától, de nem ez a meghatározó. Ezen hatás jelentősen csökkenthető multiple render target felhasználásával, amivel minden szükséges adatot egy render meneten belül bufferekbe írhatunk. Méréseink szerint az algoritmus teljesítményének egyik szűk keresztmetszete, a sűrűség kép előállítása. Ez azért van így, mert az összemosások miatt komoly mennyiségű pixel felülírás történik itt. Ezen gyorsíthatunk ha kisebb köröket használunk a sűrűség kép előállításához, ekkor azonban a sűrűség zajosabb lesz. A vonalszám nincs hatással a sűrűség kép előállításának költségére, mivel a felhasznált körök sugara a felbontásnak és a vonalszámnak függvénye. Több vonal esetén a sugár csökkenni fog, így a feldolgozott fregmensek száma nem változik.

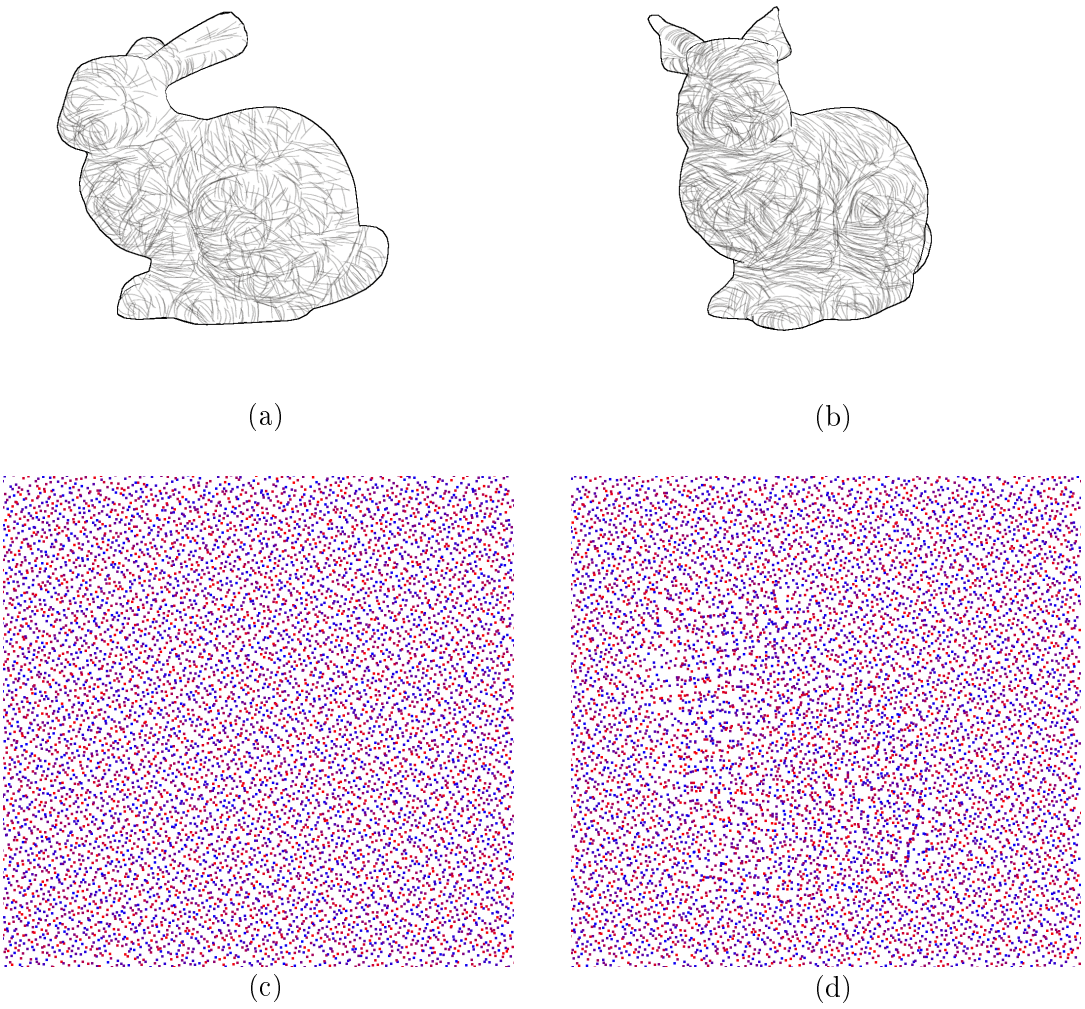
Az anti-flicker bekapcsolásának jelentős teljesítménybeli ára van, főleg nagy vonalszám esetén. Amennyiben az egyes vonal állapot számítások száma, már összevethető a képernyő pixeleinek számával, érdemes a vonal állapotokat minden képtérbeli pontban előre kiszámolni. Ez nemcsak nagy sebesség javulást adhatna, hanem a mérési eredményekben tapasztalt erős vonalszámtól való függés nagy részét, képernyő mérettől való függésre cserélhetné. Az egyik jövőbeli terv, ezen vonal állapot kép implementálása, így az anti-flicker felgyorsítása nagy vonalszám esetén.

Vonalak száma	FPS anti-flicker nélkül	FPS anti-flickerrel
10000	107.6	84.5
30000	83.4	52.0
60000	63.3	33.4
120000	42.5	19.6

5.1. táblázat. *Teljesítmény tesztek Geforce 9600 GT kártyával, 1024x700-as felbontás mellett. Egy korszerűbb kártyán (Geforce GTX 690) az algoritmus anti-flicker és 120000 vonal mellett is 100 FPS felett teljesít.*

Bár az eljárás elsősorban vonalkázásra lett kifejlesztve, a konkrét ceruzarajzon kívül más dolgokra is felhasználható. A vonások átparaméterezésével és a textúrák lecserélésével könnyen más nem-fotorealisztikus technikákat is előidézhetünk, például festést. Egy másik jövőbeli terv a kísérletezés új technikák előállításával, az algoritmus minimális módosításával.

A végső cél pedig az algoritmus ipari felhasználhatóságának a felmérése, amihez animátorok véleményét kellene kérni. Az algoritmust használó demo alkalmazás bár kínál egy egyszerű felhasználói felületet a teszteléshez, a technikai részleteket nem ismerőknek nehézkes lenne a használata. Továbbá bár elvi akadálya nincs annak hogy az algoritmus egy animátor program kimenetét dolgozza fel, egy ehhez szükséges keretrendszer még nincs megvalósítva. Ezen két akadály elhárításához az algoritmust burkoló új alkalmazásra lenne szükség, ami képes egy renderelő rendszer kimenetének feldolgozására, valamint a kezelőfelülete kellően egyszerű ahhoz, hogy az eljárást részleteiben nem ismerők is tudják azt használni.



5.1. ábra. *Egy animáció képkockái. A bal oldalon, a kezdeti kép és a hozzá tartozó részecske eloszlás látható. A bal oldalon, egy későbbi kép látható a kamera elforgatása után, és a hozzá tartozó részecske eloszlás. Az egyenletes eloszlás nem veszik el az animáció hatására. Némi csomósodás megfigyelhető az objektum széleinél, de ez a végső képen nem látható.*

6. fejezet

Konklúzió

Bemutattunk egy valós idejű vonalkázó algoritmust, ami véletlenszerűen vonalakat helyez el a képtérben. A vonalak a felületek képtérbeli elmozdulása szerint lesznek elmozgatva. Az egyenlőtlen képtérbeli sűrűség, amit a részecskék elmozgatása okoz, eldobásos mintavételezésre alapuló módszerekkel lett javítva. A vonalak, mint textúrázott háromszög hálók kerülnek renderelésre.

Az eredmény egy olyan vonalkázó algoritmus, ami megoldást kínál a képtérbeli technikák időbeli koherencia problémáira, miközben megtartja ezen technikák előnyeit:

- nincs szükség láthatósági tesztekre
- az egyenletes képtérbeli részecske sűrűség garantált
- a teljesítmény független a renderelt geometria komplexitásától

A vonalak stílusa részletesen testre szabható, azok sűrűségének, hosszának, szélességének, maximális hajlításának és textúrájának módosításával. Az implementáció a GPU-ra támaszkodik, és valós időben fut nagy felbontású környezetekben is.

Irodalomjegyzék

[1] Lee, Hyunjun and Kwon, Sungtae and Lee, Seungyong. International Symposium on Non-Photorealistic Animation and Rendering (NPAR)

[2] Yongjin Kim and Jingyi Yu and Xuan Yu and Seungyong Lee. Line-art Illustration of Dynamic and Specular Surfaces

[3] Lake, Adam and Marshall, Carl and Harris, Mark and Blackstein, Marc. Stylized rendering techniques for scalable real-time 3D animation

[4] Emil Praun and Hugues Hoppe and Matthew Webb and Adam Finkelstein. Real-Time Hatching

[5] Paiva, Afonso and Vital Brazil, Emilio and Petronetto, Fabiano and Sousa, Mario Costa. Fluid-based hatching for tone mapping in line illustrations

[6] Zander, Johannes and Isenberg, Tobias and Schlechtweg, Stefan and Strothotte, Thomas. High Quality Hatching

[7] Umenhoffer, Tamás and Szécsi, László and Szirmay-Kalos, László. Hatching for Motion Picture Production

[8] Elber, Gershon. Interactive Line Art Rendering of Freeform Surfaces

[9] Matthew Kaplan and Bruce Gooch and Elaine Cohen. Non-Photorealistic Animation and Rendering 2000 (NPAR '00)

[10] Aaron Hertzmann and Denis Zorin. Illustrating Smooth Surfaces

[11] Michael P. Salisbury and Sean E. Anderson and Ronen Barzel and David H. Salesin. Interactive Pen-And-Ink Illustration

[12] Zoltán Lengyel, and Tamás Umenhoffer. Real time screen space hatching on the GPU