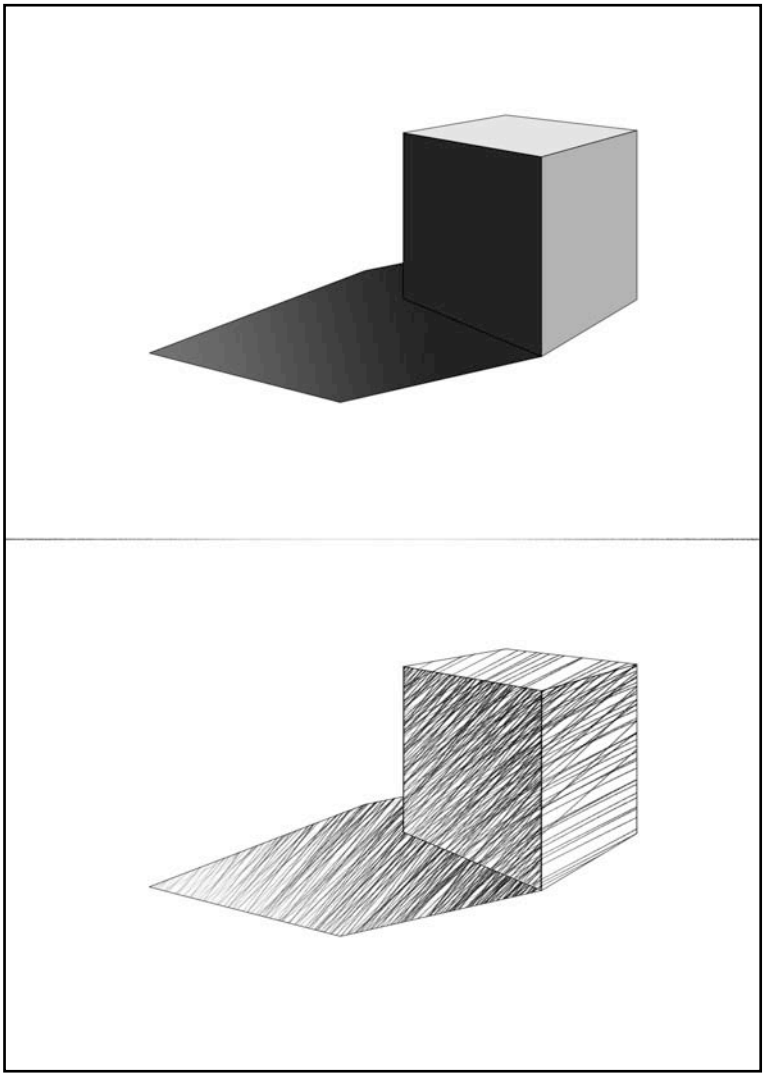
Nem-fotórealisztikus renderelés
építészeti alkalmazásai

Kovács Dániel Dávid
Konzulens: Dr. Szirmay-Kalos László

2011. október 28.



.

Nem-fotórealisztikus renderelés
építészeti alkalmazásai

Kovács Dániel Dávid
Konzulens: Dr. Szirmay-Kalos László

2011. október 28.

Tartalomjegyzék

Bevezető

A dolgozat célja annak vizsgálata, miként lehet az építészeti szabadkézi rajzot a számítógépes grafika eszközeivel közelíteni. Az építészeti grafika műfaji sajátosságai miatt elsősorban a nagyméretű sík felületek megjelenítésére adunk megoldási javaslatot. Síkoknál nem lehet a görbületi irányból meghatározni a sraffozás irányát. Kézi technikával készült képeken szembeötlő lehet, hogy a vonalseregek egy rendszert alkotnak. Ezt alapelvként kihasználjuk a képek generálásához.

A dolgozatnak célja egy jellegzetes fajta sraffozási technika megismerése, számítógépes alkalmazása. Nem célja azonban a tollvonások megjelenítésének megoldása, erre már jól működő megoldások léteznek. Ilyen az Adobe Photoshop ecsetkezelő motorja, adódik, hogy kihasználjuk ezt a képességét. A képalkotás kimenete így könnyűszerrel lehet megfelelően rétegezett psd-file, ami beilleszthető az építészeti gyakorlatban megszokott utómunka-folyamatba.

A javasolt eljárás két lépcsőben kívánja megoldani a problémát. A végső képet egy Javascriptben implementált programmal előállítjuk elő. Maszkok segítségével jól körbehatárolt területekre rajzolhatunk. A szürkeárnyalatos grádienseket több rétegben felhordott, vezériránnyal és (a normálisok irányában megadott) ritkulással-sűrűsödéssel definiált vonalsereggel jelenítjük meg. Ezeket egy külön programmal állítjuk elő a háromdimenziós jelenetből: különválasztjuk az egyes elemekhez tartozó megvilágított, önárnyékos és vetett árnyékos felületeket, ezek leírásait.

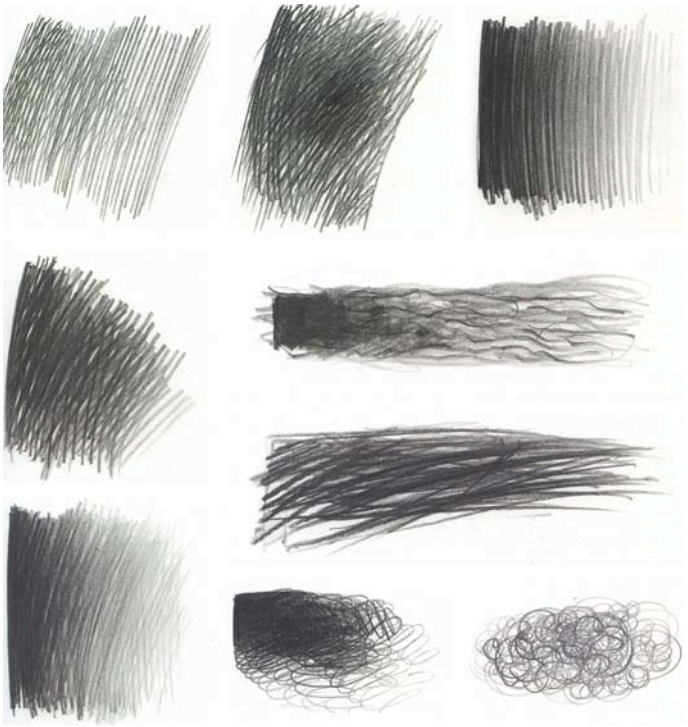
A dolgozat több, jól elhatárolható részre tagolódik. Először áttekintjük a kézi rajztechnikákat, elemezzük, és mintákon keresztül bemutatjuk az elérni kívánt kinézetet. Összefoglaljuk a létező nem-fotórealisztikus renderelési technikákat. Tárgyaljuk a megvilágított, az önárnyékos és a vetett árnyékos részek képi sajátosságait. Vizsgáljuk, hogy világos, középtónusú és sötét régiókra osztva miként lehet ezeket sraffozással megjeleníteni, és ez miként kapcsolható egy egyszerű árnyalási modellhez. Ismertetjük az implementációt, valamint megfogalmazzuk a továbbfejlesztés lehetséges irányait.

Az építészeti rajz sajátosságai

A művészeti grafika

“Természetes törekvésünk, hogy a tér dimenzióit, hatásait minden ábrázolási módban kifejezzük. A plasztikus ábrázolásban az árnyék a tér harmadik dimenzióját jelenti, és foltként a formára utal. Az árnyék a legelvontabb síkképnek is mélységet ad, és a valósághoz közelebb hozza a tervet.” [1]

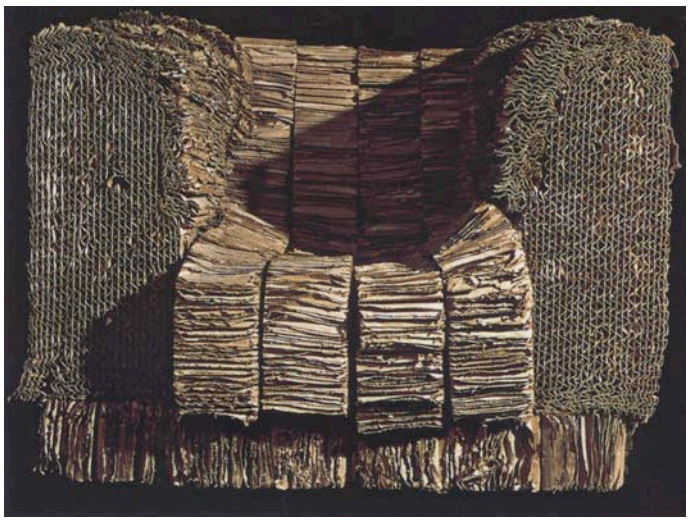
Az építészeti rajz célja bemutatni egy tömeg, vagy tömegrendszer térbeliségét, az alkotóelemek térbeli viszonyait. Ezen felül cél lehet az anyagi minőségek érzékeltetése is. Ennek egy eszköze lehet a vonalas rajz, ahol a sraffozás sűrűsége, a vonal minősége és iránya mind utalhat a fenti jellemzőkre.



[1] pp65

“A tónust a vonalak ritmikus sorozataival és lapos összemetszésükkel érjük el. Minden következetes vonalritmus tónushatást eredményez. Ha a vonalak túl meredeken, vagy kuszán metszik egymást, elveszthetik felületi folthatásukat. A tónus, miként az írás, mindig őriz valami személyes jelleget.”[1]

Pedagógiai szempontból mindenképpen megállja a helyét az az állítás, miszerint a kusza vonalak elvesztik folthatásukat. Személyes tapasztalat alapján, miként a rajz nem mérnöki diszciplína, hanem (alkalmazott-) művészeti eszköz, a szabályok áthághatóak. Ugyanakkor a pedagógiailag valid szabályok áthágásával már nem garantált a kívánatos eredmény, hacsak nem párosul (egzakt módon nehezen megfogható) ízléssel, és jómanuális képességgel. Általánosságban megállapítható, hogy a kellemes látványt nyújtó képekben közös, hogy a vonalak egy egységes rendszer részei.



[2] pp57

Egy sraffozott, fekete-fehér rajzhoz akár egy valós modellről készült fénykép is adhat megfelelő inspirációt, kiindulási pontot. A fenti Ghery-fotel képét tónusaiban, vonalstruktúrájában könnyedén, egyszerű képlettel lefordíthatnánk vonalas rajzzá. Az egyes foltok vonalrendszerei látványvilágukban egy közös képi rendszer elemei.

[8] pp51

Nem építészeti rajzoknál is megfigyelhető ez a fajta képi vonalrendszer. A görbült felületeket térbeliségét érzékelteti a felületi textúra és a fény-árnyék viszonyok.

[8] pp521

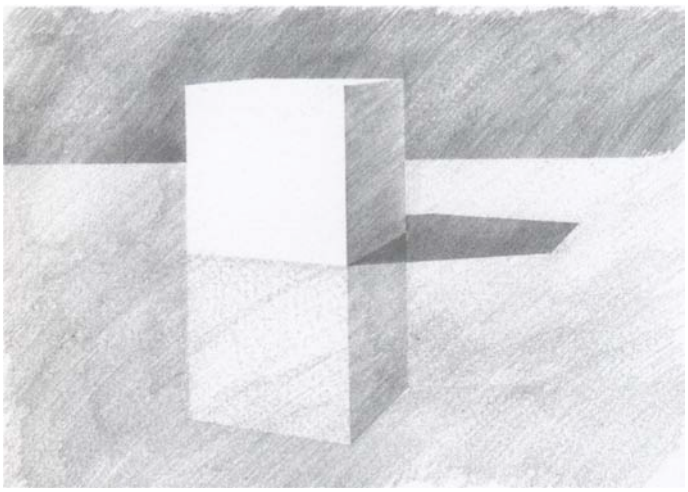
A görbült felületek megjelenítésére - a görbületi irányok jelölésén túl - megoldás lehet az, ha elemi síkokra bontjuk. Példa erre a fenti kép drapériái, plasztikusságukat az élek mentén a széles becsillanó foltok adják, valamint az egymáshoz vett viszonyaik.

[8] pp186

Miként a rinocéroszt ábrázoló Dürer- rajzon, a fenti képen is megfigyelhető, hogy az anyagszerű sraffozás (textúra) és tónus nem feltétlenül szétválaszthatóak. Egy irányrendszert alkotnak, a tónusozás gradiensét a sűrűsödésük és ritkulásuk adja.

Az építészeti grafika

Az építészeti ábrázolás vizsgálatával szűkítjük a vizsgált rajzi lehetőségek körét. Mivel a célunk egy vagy több nagyléptékű tömeg térbeli viszonyainak ábrázolása, vizsgáljuk meg, hogy miként tehetjük ezt meg síklapok ábrázolásával. Miként az megfigyelhető a fenti Dürer-rajzokon, tetszőleges görbe felületek megjelenítését (a vázlatos képi hatás keretein belül mindenképpen) visszavezethetjük elemi síkdarabok ábrázolására. Legjobb példa erre a *[8] pp521* rajz.

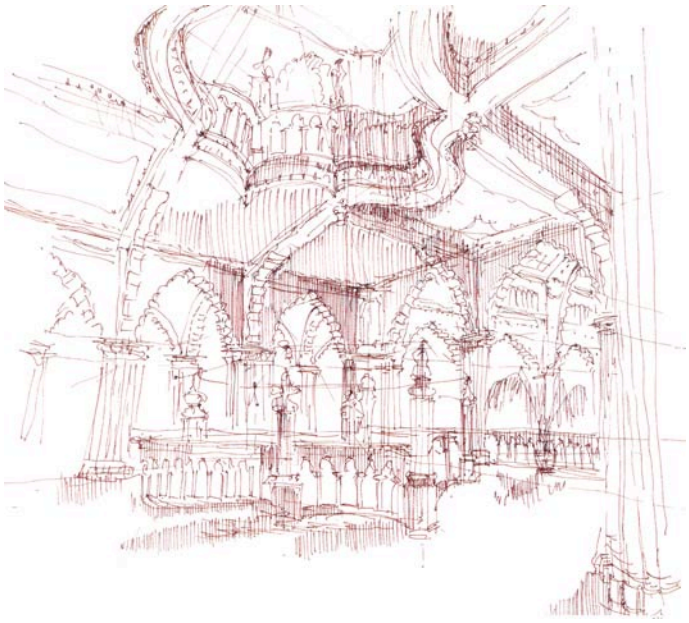


[1] pp24

Miként minden programozási nyelvben tett első lépés a ‘Hello, World!’ alkalmazás, az építészeti rajz elsajátítását szolgáló ‘kockológia’ klasszikusának számít a fenti rajz. Egy egyszerű kockán több alapvető rajzi összefüggés jól megfigyelhető. Egy vonalsereg - ha az elemei egy rajzi rendszeren belül maradnak - egy jól definiált foltot alkot. Ez a folt tónusértékkel, a tónus változásán keresztül gradienssel bír. Ezen felül a vonalsereg irány a foltnak is irány-szerű tulajdonságot kölcsönöz. Több ilyen foltot egymás mellé állítva megfigyelhető válik a test geometriája a fény-árnyék viszonyokon, a takaráson és a jelzett, vagy valóban ábrázolt irányokon keresztül.

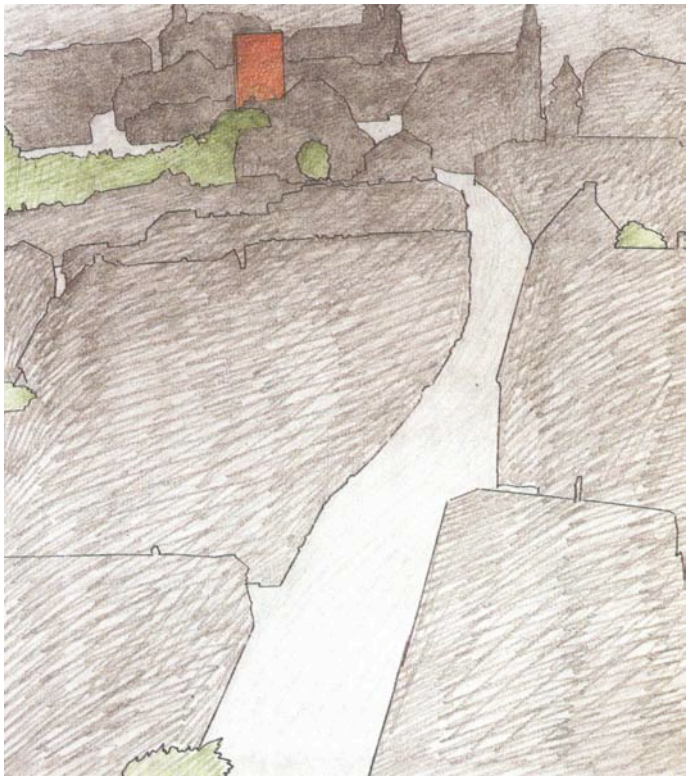
[1] pp113

Miként egy jó fényképnél, az egyes elemek összhatása nem zajos, hanem egy összefüggő egységet alkot. Ehhez nem szükséges az összes képi elemet egyforma részletességgel, precizitással kidolgozni. Hangsúlyos kompozíciós elem lehet a kidolgozottság, ami által a kép egy része fókuszba kerül, a többi pedig vázlatosan jelzi a kontextust, amiben a képi struktúrát értelmeznünk kell.



[1] pp152

A különböző precizitással kidolgozott részleteket, és ezek kontrasztját illusztrálja a fenti kép is. A képi világnak egyedi ízt ad az egyes vonalak lendülete. A foltokon belül az egyes vonalaknak külön-külön is kifejezőerejük van.



[3] pp 203

A képi fókuszt meghatározhatja az egyes foltok kontrasztja, és a foltokon belül a sraffozás feszessége is. A periférián lazábban kapcsolódnak egymáshoz, a középpontban összefüggő sötét foltot alkotnak.

A nagyléptékű sík felületeken felfoghatjuk a tónust, mint képtérben végrehajtott kitöltést. A horizontális-vertikális vonalrendszer megfelelő eszköz lehet a tömeg erőjátékának a reprezentálására is. Ekként az egyes elemek mintázata tükrözi a vízszintes és a függőleges teherhordás irányát.

[5] pp13

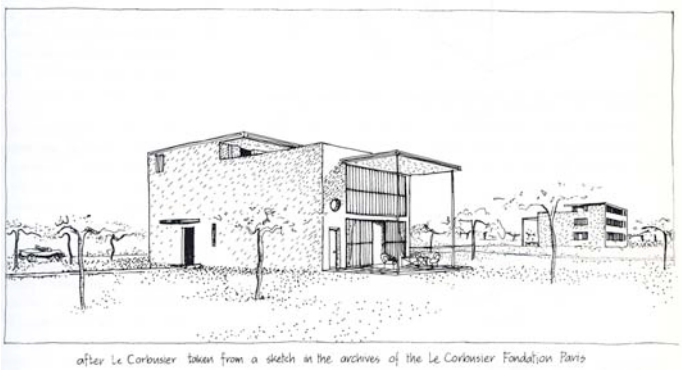
Tónusértékek
A képeket továbbvizsgálva megállapíthatjuk, hogy a teljes szürke-skálához képest igen szerény számú diszkrét intezitás-értékekkel leírható egy vonalas kép. Elkülöníthetünk világos, középtónusú és sötét területeket.

Világosként értelmezzük a tömegek megvilágított részeit. Miként egy túlexponált fényképfelvételen, a részletek itt is elvesznek, a geometriát a környezet csupán jelzi. További jelzés lehet a geometria kontúrja a kép értelmezéséhez .

[6] pp283

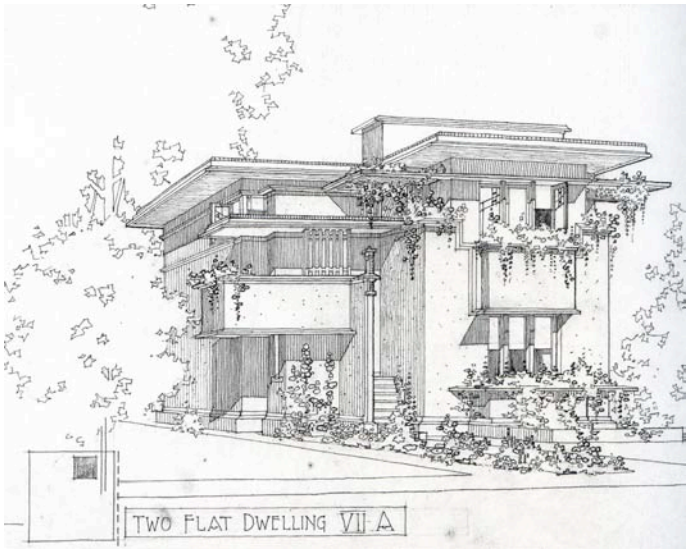
Középtónust kap minden önárnyékos és vetett árnyékos része a képnek. Ez az a folt, és egyben vonalrendszer), ami összefogja a képet, végighúzódik a hangsúlyos részeken. Ettől a folttól lesz értelmezhető a megvilágított geometria is, és válik üresből - a

tipográfiából kölcsönzött szóval élve - negatív térré. A középtónuson belül lehetnek eltérő intenzitású területek, ezek több rétegben a középtónusú alaprétegre épülnek.



[6] pp91

A sötéteket két részre bonthatjuk. Az önárnyékos felületeket kezelhetjük síkonként külön, sötét tónusuk viszonylag egyenletes, a reflexiók és a szórt fény befolyásolhatják. Vetett árnyék megvilágított felületeken keletkezik, egybefüggő foltonként külön rétegként kezeljük, amit felhordunk az árnyék nélküli képre.



[7] pp242

A tónusértéket befolyásolhatja a felületi anyag saját színe, de az itt elemzett rajzi műfajban ettől eltekintünk. Megkülönböztethetjük az egyes felületi elemek anyagi minőségét a tónusozáson túl egyéb kontraszttal, például a sraffozás vonali minőségének változtatásával.

Nem-fotórealisztikus renderelési technikák

Stuart Green [10] nyomán az NPR technikák alapvetően más szemszögből közelítik meg a képalkotás folyamatát, mint a fotórealisztikus eljárások. Míg az utóbbi egy modell alapjn szimulál, és célja minél pontosabb eredményt adni. Jellemzően az egész kép ugyanazon a befejezettségi szinten áll, az egyes részeinek a részletezettsége nem tér el.

Ezzel szemben a nem-fotórealisztikus eljárások stilizálnak, pontosságuk csupán közelítő. Alapja a látvány, és valamely művészi technika. A kép egyes részeinél lehetőség van változó kidolgozottságot elérni.

Az eljárások lehetnek kép- és modelltér-beliek, igényelhetik a képalkotáshoz a felhasználó beavatkozását, vagy dolgozhatnak önállóan.

Simon Schofield [11] példákat mutat olyan képekre, amelyek több különféle eljárással készült részekből állnak. Ezekről általánosan elmondható, hogy míg az egyes részek külön-külön megállják a helyüket, az összhatás igen esetlen. A képekből hiányzik a kompozíció, ami azokat koherens egésszé alakítaná.

A létező NPR-eljárások sokfélék. A rajzfilmes megjelenítés (cartoon shading) és a műszaki illusztráció (technical illustraton) megtalálta a maga alkalmazási területeit. Ezeken túl léteznek expresszívebb festői technikák is.

Festői NPR-technikák

Aaron Hertzmann [12] szerint a festő technikák többféle úton közelíthetik meg a képalkotást. Egy megoldás lehet a képet több rétegben, egyre nagyobb részletgazdagsággal, csökkenő ecsetmérettel foltokban felhordani (Layering). A képalkotás lehetséges ecsetvonások szimulálásával is, ahol az ecset a gradiensre merőleges útvonalat követ, és meghatározott számú vezérlőpont használata után új ecsetvonásba kezd. Ez a technika több festői stílushoz képes alkalmazkodni a sokféle paramétere segítségével.

Vonalas ábrázolások

Aaron Hertzmann [13] tárgyalja a kontúrok meghatározását. Ezt elvégezhetjük képtérben éldetektálással. Ennél jobb megoldás mélység-térképen (depth-map) éleket keresni. Az elgondolás alapja, hogy nagy változások szomszédos képpontok között csak több objektum esetén jellemzőek.

Kontúrokat meghatározhatunk továbbá a poligonhálóból is, amit az olyan élek adnak, amelyek látható és nem látható poligonokat választanak el a modellen.

Sraffozás modelltérben

Michael Batchelder és Kacper Wysocki [14] megkísérelik a Quake világát valós időben, modelltérben sraffozva megjeleníteni. Az eredmény figyelemre méltó, de az általunk elérni kívánt látványvilág szempontjából zsákutca. A modell-textúraként alkalmazott sraffozott felületek a térbeli viszonyoknak megfelelően skálázódak. A közeli felületek nagy, vastag vonalakkal kitöltöttek, a távoliak alig látható aprókkal. Ezzel szemben az általunk elérni kívánt képeken nem kívánatos ha sraffozó vonal vastagságán perspektivikus hatások érvényesülnének.

Sraffozás
Michael Salisbury et al. [15] olyan eljárást javasolnak, amiben az egyes felületekre a felhasználó adhatja meg a felületre helyezett textúra irányát. Ezt a textúra-irányt egyeztetve a felület görbületével olyan rajzolatot generálnak, amin a felhordott textúra intenzitása megegyezik az eredeti képből számított helyi átlagokkal. A javasolt módszer erősen támaszkodik a felület görbületi irányaira és az alkalmazott textúra orientációra. A módszer kiváló kisléptékű tárgyak és organikus formák illusztrálására.

Ezzel szemben az itt javasolt módszer textúra nélküli megjelenítést javasol görbület nélküli, nagyléptékű sík felületekre. Ez a megoldás az építészeti tömegvázlatok jellemző képi világának a visszaadását tűzte ki maga elé.

Javasolt renderelési eljárás

Kétszintű rendszer
A javasolt eljárásban a képet két lépésben állítjuk elő. Az építészeti grafika műfajáról tett megfontolások alapján a modellt sík felületekből álló geometriaként kezeljük. Első lépésben a cél több szürkeárnyalatos képet előállítani a modellről, amiknek a színértékét az egyes síklapok intenzitása adja. Második lépésben a már képtérbe leképezett modellből származó poligonokból megfelelő gradiensű sraffozott foltokat állítunk elő. A poligon két célt szolgál. Megadja a formát, amit ki kell maszkolni a kész vonalseregből, ezen túl pedig a szürkeárnyalat értéke meghatározza az egyes vonalak távolságát a vonalseregen belül. A vonalak sűrűsége adja ki a folt intenzitását.

A modelltér-beli szintézishez sugárkövető programot használunk az alábbiakban ismertetett kimenetekkel. Erre a lépésre bármilyen más eszköz is megfelelő, ami az itt leírt kimeneti képeket produkálja.

A képtér-beli szintézishez az első lépésben előállított képek feldolgozására van szükség. Az itt ismertetett eljárás erre az Adobe Photoshop programot használja, amit egy Javascriptben implementált program vezérel. Ennek az előnye, hogy az eljárás kimenete egy olyan rétegekből felépített kép, ami könnyen beilleszthető egy tipikus építészeti munkafolyamatba. További előny, hogy a kép elkészítésénél használhatjuk a program fejlet ecset-motorját, és a kép feldolgozásához magas-szintű eszközöket nyújt. A script interpretált jellege miatt a későbbiekben könnyedén testreszabható.

Szintézis modelltérben
A képszintézis első lépését egy sugárkövető programmal a legegyszerűbb végrehajtani. A képpontok intenzitásának a meghatározásához Lambert diffúz modelljét [9] használhatjuk;

$$\mathrm{L} \;=\; \mathrm{L}_{in} \cdot \cos\theta$$

ahol
L a felületről visszaverődő intenzitás
L_{in} a felületre eső intenzitás
θ a beesési pontban a felületi normális és a beeső fény által bezárt szög.

Az eredeti képletben az így kapott érték megszorzódik egy k számmal, ami a felület anyagi tulajdonságágának a függvénye. Mi ezt konstans 1-nek vesszük.

A vetett árnyék meghatározásához a talált felületi pontból sugarat indítunk a fényforrás felé. Ha ez elmetsz bármilyen felületet, akkor az előzőben megtalált pontra árnyék vetül. Az árnyékos pont intenzitása legyen egy kis érték, amit a diffúz megvilágítás ad.

Nevezzük a pont és a fényforrás távolságát X-nek, a pont és az árnyékot vető pont távolságát x-nek, c tetszőlegesen meghatározható arányossági tényező. Ha az árnyékos pont intenzitását súlyozzuk

$$c\frac{x}{X}$$

értékével, akkor elérhetjük, hogy az árnyékot vető pontokhoz közelebbi árnyékpontok sötétebbek, a távoliabbak világosabbak legyenek.

első közelítésben eltekintünk a tükröződések kezelésétől, az itt ismertetett módon ez könnyedén integrálható a renderelési eljárásba. Ehhez a tükörképként látszódó poligonokat is önálló síklapokként kell kezelni és külön rétegeket alkotni belőlük.

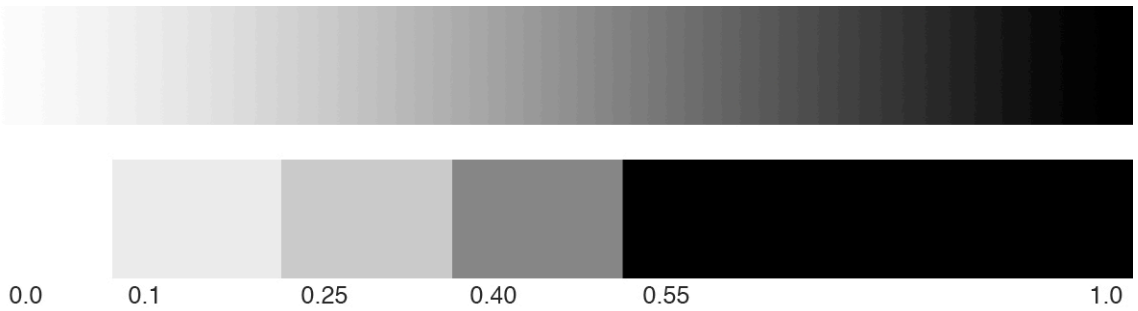
Az így előállított képen, mivel síklapokkal dolgozunk, egy-egy poligon vagy teljes egészében megvilágított, vagy önárnyékos. Ha nem párhuzamos fényforrással dolgozunk, akkor keletkezhetnek a képen gradiensek, ez a rajzi megjelenés szempontjából kívánatosnak mondható.

Az így előállított képet alkalmassá kell tegyük arra, hogy képtérben feldolgozzuk. A későbbiekben intenzitás helyett kitöltöttséggel fogunk számolni, ahol 1 a fekete, 0 a fehér, a kettő közötti értékek pedig a szürke átmenet.

Két kép-puffert használunk, az egyikben azt tároljuk, hogy adott képernyő-koordinátán melyik azonosítójú síklap látszik, milyen értékkel. A másikban a vetett árnyékokat. Itt elegendő csupán a képpont értékét tárolni, ha szükség lenne arra, hogy ez melyik síklapra vetül, az kiolvasható az előbbi tömbből. Arra, hogy melyik pont veti az árnyékot, a képpont-beli érték kiszámítása után már nincs szükség.

Ebből a két pufferből külön-külön képet állítunk elő a középtónusnak, az egyes síklapoknak, és a vetett árnyéknak.

A felbontáshoz a folytonos(nak tekinthető) szürkeárnyalatos skálát diszkrét tartományokra osztjuk fel. Ez a felosztás a látvány megfigyelésén alapul, olyan sávokat keresünk a folytonos skálán, amiket szabad szemmel nagyjából azonos értékűnek látunk. E szerint a felosztás:



Az átlagos kitöltöttség alatt az egyes képpontokra vett kitöltöttség-értékek pixel-szám szerinti átlagát értjük.

Előállítjuk a középtónus alaprétegét: az olyan lapokat, amelyeknek az átlagos kitöltöttsége 0.1 alatt van, elhagyjuk. A megmaradó képekből (a vetett árnyékok képeit is beleértve) mindenhol, ahol van nemfehér pixel, 0.1 értéket írunk a középtónus rétegbe, a többi helyen 0-t.

A megmaradt síklapok képeit kiigazítjuk, miután ezek színének egy részét figyelembe vettük a középtónus rétegénél. Ehhez eldobjuk az összes olyan lapot, aminek az átlagos kitöltöttsége kisebb, mint 0.25, ami a mi diszkrét felosztásunk szerint a harmadik tartomány alsó határa. Az ez után megmaradt lapok minden képpontjának az értékéből levonunk 0.1-et.

Az így megmaradt kitöltöttség-értékeket a képtér-beli szintézis során egyenletesen osztjuk el a sraffozási rétegek között.

A képtérbeli szintézishez íly módon előállítottuk a következő különálló képeket:
- Egy kép a középtónus-foltoknak. Az összefüggő foltokat a képtér-beli feldolgozás során egyszerűbb szétválasztani.
- Egy kép a vetett árnyékoknak. Az összefüggő foltok szétválasztása megegyezik a középtónus-foltokéval.
- Minden síklapnak egy-egy külön kép.

Szintézis képtérben
A képtér-beli szintézishez egy Javascriptben implementált programot használunk. Ennek a programnak a feladata az előző lépésben előállított képek feldolgozása egyesével, majd az így kapott eredmény kompozitálása.

A teljes képre egy sraffozási irány értelmezett, ez a vonalsereg normálisával adott. Az első réteg után felhordott további rétegek irányát ehhez a normálishoz képest vett kis szögváltoztatással határozzuk meg. Az egyes rétegek szögváltása változtatható paraméter, az implementációban használt értékek javaslatok a kívánatosnak vélt vizuális hatás eléréséhez. A közös sraffozási irány adja meg azt az egységes képi rendszert, amit a manuálisan előállított grafikáknál vizsgáltunk.

A fent ismertetett, diszkrét tartományokra osztott szürke-skála határozza meg, hogy egy folt hány réteg sraffozásból épül fel. A fehér tartományt nulladiknak értelmezve annyi réteget kell előállítanunk, ahányadik tartományba esik az összefüggő folt átlagos kitöltöttsége. Így

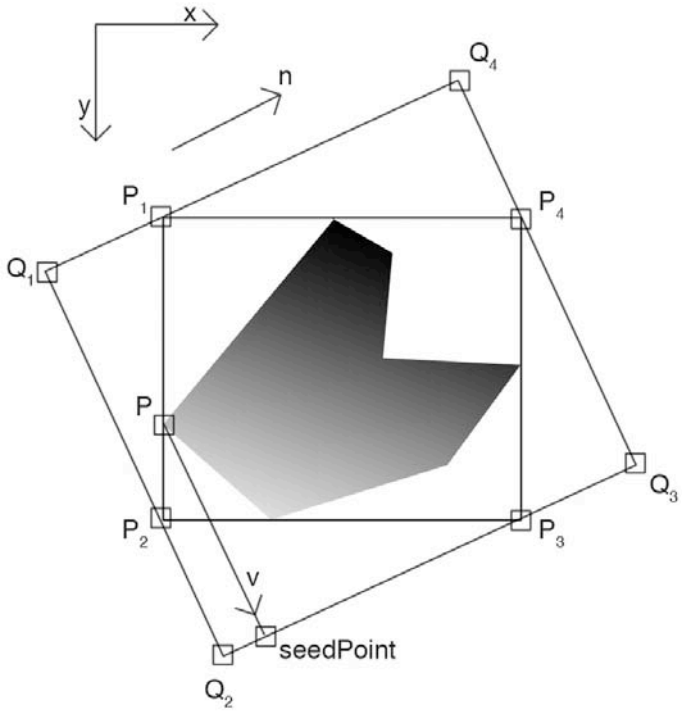
- 0.0 .. 0.1: 0 réteg
- 0.1 .. 0.25: 1 réteg
- 0.25 .. 0.40: 2 réteg
- 0.40 .. 0.55: 3 réteg
- 0.55 .. 1.0: 4 réteg

Az egyes sraffozási rétegek előállítása azonos, különbség csupán a vonalsereg normálisában van. Az itt ismertetett javaslat szerint a normálistól vett eltérés szöge

- az 1. rétegben: 0°
- a 2. rétegben: 10°
- a 3. rétegben: 15°
- a 4. rétegben: 20°

Az első réteg minden esetben a középtónushoz tartozik.

Egy sraffozási réteg előállítása



A vonalsereg rajzolásához iterálva egy irányba léptetjük a $\mathrm{seedPoint}$-ot, így megkapjuk a vonalsereg kezdőpontjait. A léptetések iránya és a sraffozási vonalak meghatározzák $\mathrm{Q}_1Q_2Q_3Q_4$ ferde befoglaló téglalapot, amit a számítás egyszerűsítése kedvéért a $\mathrm{P}_1P_2P_3P_4$ befogaló téglalap köré szerkesztünk. A számítás menete megegyező lenne, ha a teljes képtér köré szerkesztenénk a ferde befoglaló téglalapot, ugyanakkor a vonalrajzolás sebessége ebben az implementációban nem elhanyagolható.

Belátható, hogy az n normálvektor irányára pontosan négy esetben kapjuk ugyanazt a ferde befoglaló téglalapot. Ez a négy eset az n vektor négy, egymáshoz képest 90°-kal elforgatott képe, amelyek külön-külön a négy síknegyed egyikébe esnek. Ezeket jelöljük $\mathrm{n}^I, n^{II}, n^{III}, n^{IV}$ -gyel. Különbséget kell tennünk az n és a $\mathrm{-n}$ normálvektor között, ha kihasználjuk az ecsetmotor lehetőségeit, és figyelembe vesszük az ecset dinamikáját. Ezzel szimulálhatjuk az, amikor rajzolás közben lassan csökkentünk a ceruza nyomásán, és a rajzolt vonal lassan elfogy. A választott képi mintánk a ceruza-rajz, így ez a képi megjelenés szempontjából ez kívánatos.

Látható, hogy a négy esetben ugyanazon ferde befoglaló téglalap négy különböző oldalára kell P pontot vetítenünk, hogy megkapjuk a $\mathrm{seedPoint}$ kezdőpontot. Ezt megtehetjük esetszétválasztással aszerint, hogy az n normálvektor melyik síknegyedbe esik. Ebben az esetben a négy esetre négy különböző módon tudjuk n vektorból származtatni $\mathrm{Q}_1, Q_2, Q_3, Q_4$ csúcsokat. Másik megoldás az lehet, ha egy zárt képletet tudunk konstruálni, ami mind a négy esetre megadja a keresett vetítővektort. Mi ezt az utóbbi utat fogjuk követni. Az ehhez szükséges lépések:

- A befoglaló téglalap meghatározása
- A ferde befoglaló téglalap meghatározása
- P pont és a sraffozás irányának megfelelő gradiensek meghatározása
- v vetítővektor meghatározása, ahol $\mathrm{v} = \mathrm{seedPoint}$ - P
- egy sraffozás vonal megrajzolása

• a vonalsereg megrajzolása

A befoglaló téglalap meghatározása
A befogaló téglalap meghatározása visszavezethető egyszerű minimum- és maximumkeresési feladatra. Ehhez először a képet vektoros körvonallá alakítjuk. Kihasználva a Photoshop adta magas szintű képfeldolgozási eszközeit, és figyelembe véve ezek programozhatóságát, ennek a lépései a következők:

• kiválasztjuk a bal felső 10x10 pixeles sarkot. A bemeneti képekre követelmény, hogy ez a régió mindig fehér legyen.
• kiválasztjuk a képen az összes hasonló színű pixelt
• invertáljuk a kiválasztást

Ezzel a három lépéssel a program *varázspálca* eszközének a funkcionalitását reprodukáltuk. A varázspálca közvetlenül Javascriptből nem elérhető.

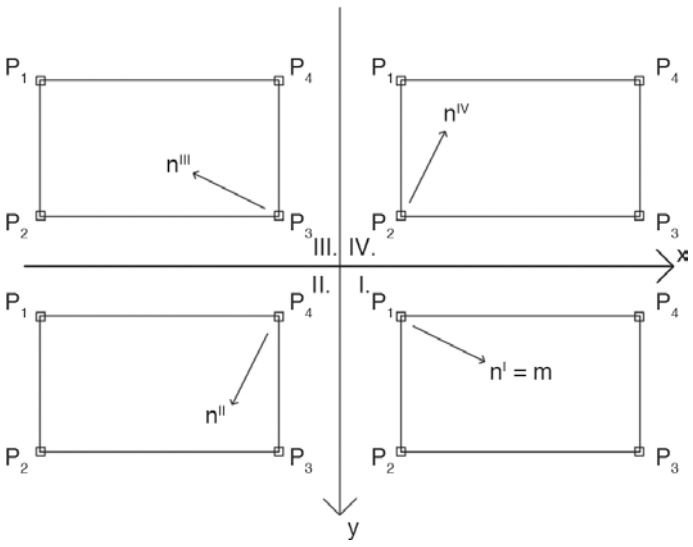
• a kiválasztást vektoros körvonallá (*pathItem*) konvertáljuk

Így a körvonal pontjain iterálva meghatározhatjuk a legkisebb és a legnagyobb x, illetve y koordinátát. Ezek legyenek $\mathrm{x}_{min}, x_{max}, y_{min}, y_{max}$. Ekkor a befoglaló téglalap koordinátái (a fenti ábra jelöléseit követve)

$$\begin{aligned}\mathrm{P}_1 &= (x_{min}, y_{min})\\ \mathrm{P}_2 &= (x_{min}, y_{max})\\ \mathrm{P}_3 &= (x_{max}, y_{max})\\ \mathrm{P}_4 &= (x_{max}, y_{min})\end{aligned}$$

Ahhoz, hogy elegendő legyen csupán a csúcspontokat végignézni, kihasználtuk, hogy poligonokkal dolgozunk, nincsenek íves élek, amielyek az így meghatározott befoglaló téglalapon kívülre esnének.

A ferde befoglaló téglalap meghatározása
A ferde befoglaló téglalap csúcspontjaihoz először meghatározzuk m vektort, ami n^k első síknegyedbe $\mathrm{k}\cdot\frac{\pi}{2}$ forgatással transzformált képe, ahol k az n vektor által elfoglalt síknegyed sorszáma.



vezessük be

$$\mathrm{X} \;=\; \begin{cases} -1 & a<0 \\ 0 & a=0 \\ 1 & a>0 \end{cases} \;;\; a \;=\; \left(\overline{P_2P_1} \;\times\; n\right).z$$

és

$$\mathrm{Y} \;=\; \begin{cases} -1 & a<0 \\ 0 & a=0 \\ 1 & a>0 \end{cases} \;;\; a \;=\; \left(P_1P_4 \;\times\; n\right).z$$

ahol X az előjelével meghatározza, hogy n vektor az x-tengely pozitív, vagy negatív tartományában van-e, Y hasonlóan az y-tengelyre. 0 érték esetén a tengelyen van, ezt később speciális esetként kezeljük.

Ekkor
- az első síknegyedre $\mathrm{X} = +1,\ \mathrm{Y} = +1$
- a második síknegyedre $\mathrm{X} =$ -1, $\mathrm{Y} = +1$
- a harmadik síknegyedre $\mathrm{X} =$ -1, $\mathrm{Y} =$ -1
- a negyedik síknegyedre $\mathrm{X} = +1,\ \mathrm{Y} =$ -1

adódik. Megfigyelhetjük továbbá, hogy az egyes síknegyedekből az elsőbe forgatáshoz síknegyedenként a következő transzformációs mátrixok szükségesek:

az első síknegyedre

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

a második síknegyedre

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

a harmadik síknegyedre

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

a negyedik síknegyedre

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Ezt a négy esetet egyszerre kielégíti a következőképpen konstruált matrix:

$$\begin{bmatrix} \frac{X+Y}{2} & \frac{Y-X}{2} \\ \frac{X-Y}{2} & \frac{X+Y}{2} \end{bmatrix}$$

A tengelyirányú esetekre ez viszont még nem a kívánt eredményt produkálja.

• y-irányú n-re:

$$\mathrm{X} = 0,\ \mathrm{Y} = +1$$

így a transzformációs mátrix

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

• x-irányú n-re:

$$\mathrm{X} = +1,\ \mathrm{Y} = 0$$

így a transzformációs mátrix

$$\begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

• -y-irányú n-re:

$$\mathrm{X} = 0,\ \mathrm{Y} = \text{-}1$$

így a transzformációs mátrix

$$\begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

• -x-irányú n-re:

$$\mathrm{X} = \text{-}1,\ \mathrm{Y} = 0$$

így a transzformációs mátrix

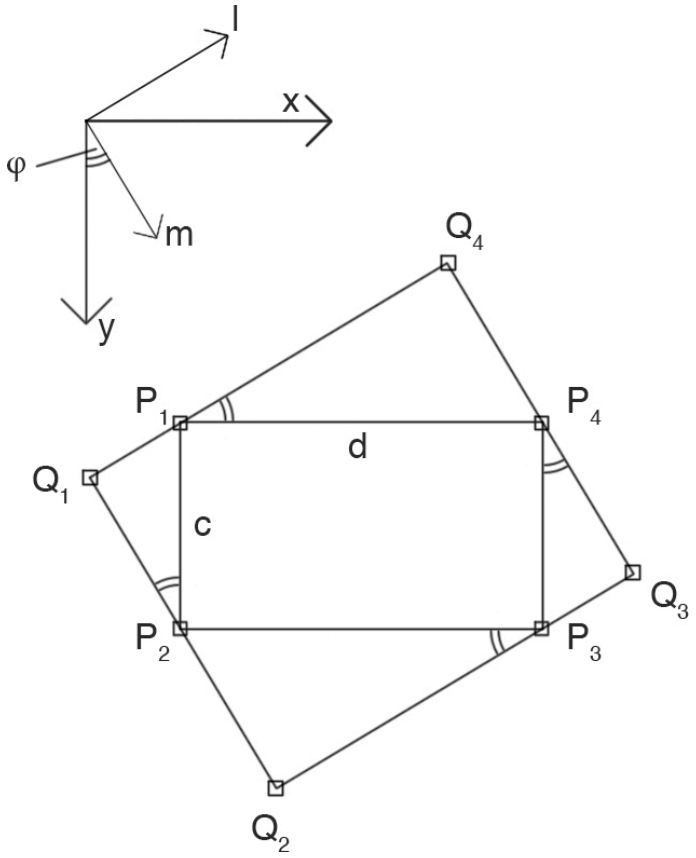
$$\begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

Az y-irányú n-vektort vegyük első síknegyedbelinek, a többit ennek megfelelően értelemszerűen. Ekkor oly módon javíthatjuk meg a transzformációs mátrixunkat, ha a nem kívánt elemeit a mátrixnak Y vagy X értékével kinullázzuk. Mivel a transzformációból kapott m vektort normalizáljuk, elhanyagoljuk a transzformációs vektor skalárszorzóit. Ekkor:

$$\begin{aligned} \mathrm{A} &= \begin{bmatrix} Y^2 \cdot (X+Y) & X^2 \cdot (Y-X) \\ X^2 \cdot (X-Y) & Y^2 \cdot (X+Y) \end{bmatrix} \\ \mathrm{m} &= (A \cdot n).normalize() \end{aligned}$$

ahol m minden esetben az n vektor első síknegyedbe forgatott képe. Vegyük fel továbbá l vektort, ami hasonlóképpen származtatott képe a sraffozási vonalak irányvektorának. Legegyszerűbben m vektor forgatásából kaphatjuk meg:

$$\mathrm{l} = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \cdot m$$



Ha

$$\mathrm{c} = |P_1P_2|$$
$$\mathrm{d} = |P_1P_4|$$

akkor

$$\mathrm{Q}_1 = P_1 - l \cdot c \cdot \sin\varphi$$
$$\mathrm{Q}_2 = P_2 + m \cdot d \cdot \sin\varphi$$
$$\mathrm{Q}_3 = P_3 + l \cdot c \cdot \sin\varphi$$
$$\mathrm{Q}_4 = P_4 - m \cdot d \cdot \sin\varphi$$

tengelyirányú normálvektorra helyesen a befoglaló téglalapot adja.

P pont és a gradiensek meghatározása

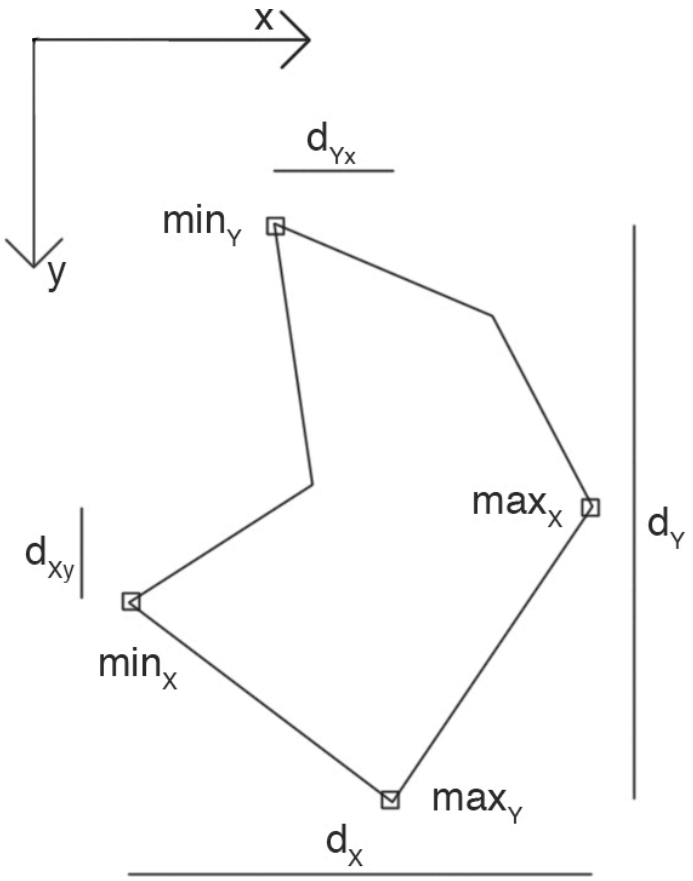
A P pont a körvonal poligonnak az a pontja lesz, amin átmegy át az első sraff-vonal. Ez a poligonnak az $\mathrm{n}$vektor kezdőpontja felőli ‘legtávolabbi pont, aminek a meghatározása egy egészértékű programozási feladat lehetne. Ugyanakkor használva a Photoshop eszközeit, ez a probléma visszavezethető egy forgatásra és minimum-, maximumkeresésre.

- Elforgatjuk a poligont az y-tengely és az n normálvektor által bezárt szöggel. A szöget minden esetben az óramutató járásával ellentétes irányú forgatáshoz keressük.
- Megkeressük a minimális és maximális x, illetve y koordinátájú pontokat, valamint a talált pontban a folt színét. Mivel csak a körvönalat forgattuk el, az implementációban megjegyezzük a kollekcióban a talált pontok sorszámait, és az eredeti, el nem forgatott

korvonalból kiolvassuk a koordinátáikat, majd az így kapott pontokban megvizsgáljuk a folt színét.

ekkor legyenek
$\min_X$ a minimális x-koordináta, $\min_{Xy}$ a hozzá tartozó pont y-koordinátája,
$\min_Y$ a minimális y-koordináta, $\min_{Yx}$ a hozzá tartozó pont x-koordinátája,
$\max_X$ a maximális x-koordináta, $\max_{Xy}$ a hozzá tartozó pont y-koordinátája,
$\max_Y$ a maximális y-koordináta, $\max_{Yx}$ a hozzá tartozó pont x-koordinátája.
$\mathrm{d}_X = max_X - min_X$, feltételezzük, hogy soha sem 0
$\mathrm{d}_{Xy} = max_{Xy} - min_{Xy}$
$\mathrm{d}_Y = max_Y - min_Y$, feltételezzük, hogy soha sem 0
$\mathrm{d}_{Yx} = max_{Yx} - min_{Yx}$
$[\min_X]$ a minimális x-koordinátájú pontban a kitöltöttség
$[\min_Y]$ a minimális y-koordinátájú pontban a kitöltöttség
$[\max_X]$ a maximális x-koordinátájú pontban a kitöltöttség
$[\max_Y]$ a maximális y-koordinátájú pontban a kitöltöttség
$\mathrm{di}_X = [min_X] - [max_X]$
$\mathrm{di}_Y = [min_Y] - [max_Y]$

r_X az elforgatott körvonal szerinti x-tengely mentén vett gradiens értéke
r_Y az elforgatott körvonal szerinti y-tengely mentén vett gradiens értéke



$$\text{I: } \mathrm{d}_Y \cdot r_Y + d_{Yx} \cdot r_X = -di_Y$$
$$\text{II: } \mathrm{d}_X \cdot r_X + d_{Xy} \cdot r_Y = -di_X$$

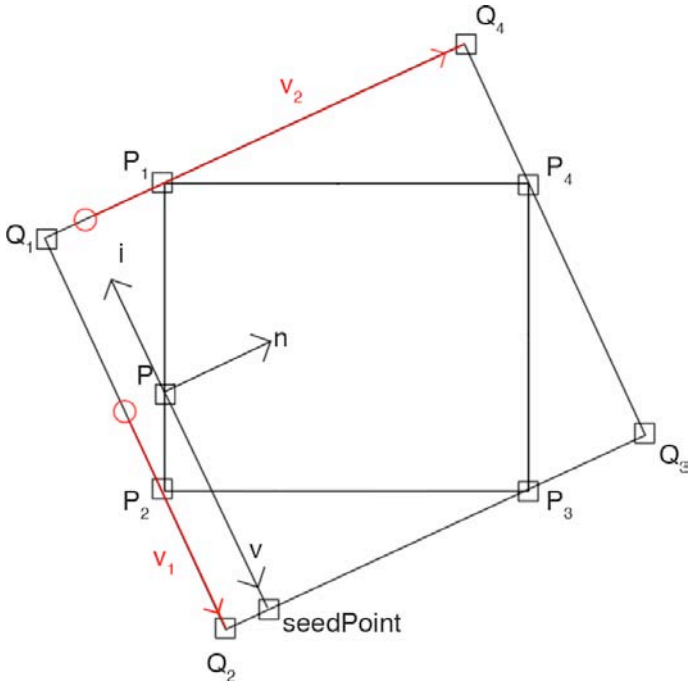
$$\text{I: } \mathrm{r}_Y = \frac{-di_Y - d_{Yx} \cdot r_X}{d_Y}$$

$$\text{II: } \mathrm{r}_X = \left(d_{Xy}\frac{di_Y}{d_Y} - di_X\right) / \left(d_X - d_{Xy}\frac{d_{Yx}}{d_Y}\right)$$

A P pont a $\min_Y$-hoz tartozó pont.

A v vetítővektor meghatározása

Hasonlóan a ferde befoglaló téglalaphoz, a cél itt is egy zárt képlet meghatározása, ami együtt kezeli a négy síknegyedbe eső n normálvektorok esetét. Ez megtehető, ha megvizsgáljuk n és m vektorok viszonyát a skalárszorzatukon keresztül.



$$\mathrm{i} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot n,$$

$$\mathrm{v}_1 = \left(\overline{PQ_2} \cdot m\right) \cdot m$$

$$\mathrm{v}_2 = \left(\overline{PQ_4} \cdot l\right) \cdot l$$

ahol i a sraffozás irányvektora, n a normálisa. Mindkét vektor normált. m és l vektorokat már korábban definiáltuk. Ennek megfelelően határozzuk meg a és b értékét oly módon, hogy a megfelelő merőleges komponenseket nullázzák ki a v vetítővektor összeállításakor. Legyen

$$\mathrm{a} = \mathrm{i} \cdot m = \begin{cases} +1 & : i = i^I \\ 0 & : i = i^{II}, i^{IV} \\ -1 & : i = i^{IV} \end{cases}$$

$$\mathrm{b} = \mathrm{i} \cdot l = \begin{cases} +1 & : i = i^{IV} \\ 0 & : i = i^{I}, i^{III} \\ -1 & : i = i^{II} \end{cases}$$

Ekkor megállapíthatjuk, hogy

$$\frac{a^2 - a}{-2} \Leftrightarrow a < 0$$

ezzel kiválaszthatjuk a vektorral ellentétes irányú komponenst. Így v vetítővektorra a következő képlet adódik:

$$\mathrm{v} = \mathrm{a}^2 \cdot v_1 - \tfrac{a^2-a}{2} \cdot \overline{Q_1Q_2} + b^2 \cdot v_2 - \tfrac{b^2-b}{2} \cdot \overline{Q_1Q_4}$$

Ez y-irányú m vektorra is megfelelő vetítővektort produkál.

Egy sraffozási vonal megrajzolása
Egy vonal megrajzolásához a $\mathrm{seedPoint}$-ból a megfelelő hosszúsággal i irányvektor irányába vonalat húzunk. Ezt a 'megfelelő hosszúságot' nevezzük j vektornak, amit a következőképpen határozunk meg:

$$\mathrm{j} = \mathrm{a} \cdot \overline{Q_2Q_1} + b \cdot \overline{Q_4Q_1}$$

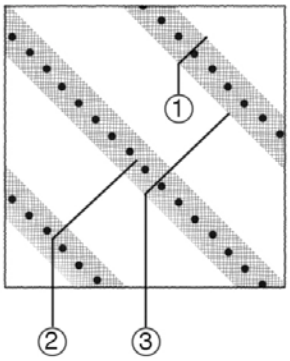
ahol a és b az előzőekben meghatározott együtthatók. A vonalrajzolás pszeudokódja:

```
drawLine(seedPoint, seedPoint + j);
```

A vonalsereg megrajzolása
A sraffozás kitöltöttsége és a használt ecset mérete meghatározza a sraffozási vonalak távolságát a következők szerint:

$$patternWidth = \frac{brush\mathrm{Size}}{fillRatio}$$

Ahol
$\mathrm{patternWidth}$ a sraffozási vonalak tengelytávolsága (1 és 2),
$\mathrm{brushSize}$ az ecset mérete (3),
$\mathrm{fillRatio}$ a kitöltöttség, értéke 0 és 1 közötti.



Elméletileg a vonal rajzolásakor az ecset meghatározásához figyelembe vehetnénk r_X gradienst, azonban egyszerű eszközökkel nincs lehetőség Javascriptből az ecset

tulajdonságainak állítására. Ha lenne, a gradiens alapján meg lehetne határozni a vonal kezdő és vég-vastagságát.

Az erre merőleges irányú r_Y gradienst a vonalak közti távolság meghatározásához használjuk fel. Mivel a kitöltöttség értékét több réteg között osztjuk meg, a felhasználása előtt r_Y értékét el kell osztanunk a folthoz használt sraffozási rétegek számával.

Az első $\mathrm{seedPoint}$-ban, ha r_Y sötétedő gradienst ad, a kitöltöttség kezdeti értékét határozzuk meg 0.1-ben, ami a fehér utáni első szürkesáv alsó kitöltöttségi értéke. Ha világosodik a folt, vegyük a sáv felső határát, 0.25-öt.

Legyen egy tetszőleges sraffozási vonalnál vett kitöltöttség-érték i_0. Ekkor a következő vonalnál érvényes i_1 kitöltöttséget meghatározhatjuk:

$$\mathrm{i}_1 = i_0 + r_Y \cdot patternWidth$$

A vonal kezdőpontjának a léptetését patternWidth hosszon n normálvektor irányába kell megtenni az egyes vonalrajzolások között. A vonalrajzolást addig kell végezni, amíg a kiinduló $\mathrm{seedPoint}$ -tól a kezdőpontok léptetésével meg nem tettünk legalább d_Y hosszúságú utat.

így a vonalsereg rajzolására a következő pszeudokód adódik:

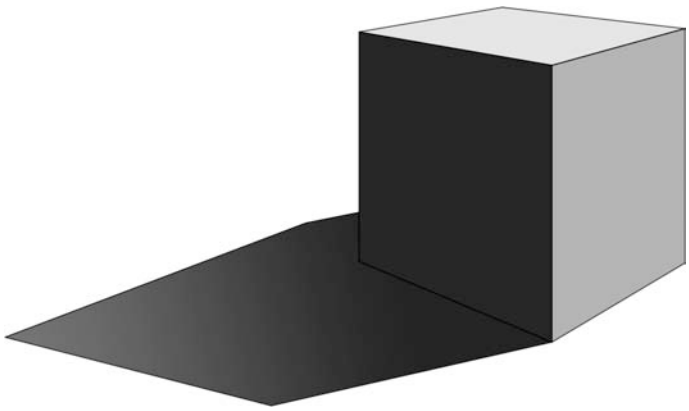
```
patternWidth = brushSize / fillRatio;
sumSteps = 0;

while(sumSteps < dY){
      drawLine(seedPoint, seedPoint + j);
      seedPoint += n * patternWidth; //n normált
      sumSteps += patternWidth
      fillRatio += rY * patternWidth;
      patternWidth = brushSize / fillRatio;
}
```

Ahhoz, hogy megtörjük a vonalak merev rendszerét, egyenes vonalak helyett véletlen hosszúságú végponti érintőkkel meghatározott Beziér-görbéket használunk. Megfelelően paraméterezve a kívánt hatást nem zavaró mértékű véletlenszerűség adódik a sraffozásban. Ezt az implementációban a wonk_factor nevű paraméter határozza meg.

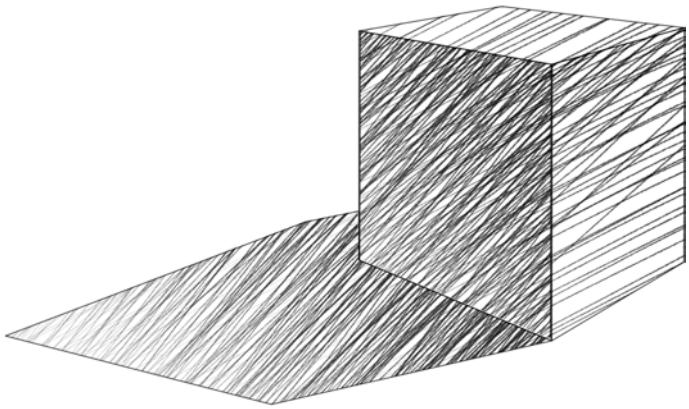
Végeredmény
Mivel az újszerű gondolat veleje a renderelés második, a képtérben végrehajtott lépésében van, egy kézzel készített képsort adtunk meg bemenetnek. Ez nem befolyásolja a gondolat helyességének igazolását, a sugárkövetés ismert, klasszikusnak mondható eljárás. A bemeneti képek elkészítésésnl szem előtt tartottuk az itt felállított követelményeket. Az implementációban a középtónust jelölő képet az egyszerűség kedvéért a fájl nevében elhelyezett kezdő ‘0’ karakterrel különböztettük meg a többitől.

A bementi kép teljes egészében:



A szürkeárnyalatos kimeneti kép, amit a következő rétegekre lehet bontani az ismertetett eljárás alapján. Az utolsó kocka a kontúrokat tartalmazza, ezt a második lépcső nem dolgozza fel, a kívánt rajzi hatás függvényében kompozitálhatjuk a kész sraffozott képpel.

Ekkor a második lépcső kimenetéből összeállított, sraffozott rajz:



Tanulságok

Továbbfejlesztés lehetőségei

- Kívánatos lehet az egyes objektumokra külön-külön meghatározni a sraffozás főirányát
- Az ecsetmotor hozzáférhetetlenségét megkerülve az r_X gradiens implementálása
- A Beziér-görbék érintőihez használt véletlen szám miatt ez az eljárás nehezen adaptálható mozgókép készítéséhez.
- Mivel a Photoshop lehetőséget ad rá, érdemes lenne grafikus felhasználói felületet adni a a renderelés paramétereinek állításához
- A kontúrokra is alkalmazni a vonalak dinamikájáról mondottakat

Kihívások

A fejlesztés számos tanulságos kihívás elé állított. A nehézségek egy részének az a forrása, hogy ennél kevésbé összetett feladatok elvégzésére tervezték a Photoshop Javascripten keresztül vezérelhető felületét. A kézzel elvégezhető feladatok egy részére nincs lehetőség. Mások, mint például a vektoros poligon forgatására az egyetlen lehetőség használat közben ‘lehallgatni’ a rendszer üzeneteit, és az itt átadott paramétereket saját változókkal heyettesíteni. Ez a megközelítés pusztán említés szintjén dokumentált.

Érett fejlesztői eszközöknél szokatlanok az ilyen léptékű pontatlanságok, mint ami a Photoshop Javascript dokumentációjában van. Erre két példa:

- Hibás a pontokat reprezentáló osztály leírása, x és y mezők helyett a koordinátákat kételemű tömbökben tárolja.
- Pontok helyzetét tetszőleges mértékegységben ki lehet olvasni, de megadni csak 72 dpi felbontású képernyő mellett feltételezett pontokban lehet.

Hivatkozások

[1]
Dobó Márton, Molnár Csaba, Peity Attila, Répás Ferenc
Valóság Gondolat Rajz
Műszaki Könyvkiadó, 1999.
ISBN 963 16 3024 2

[2]
Frank Gehry, architect
Guggenheim Museum Publications, 2001.
ISBN 0 8109 6929 7

[3]
Janáky István
a hely
Műszaki Könyvkiadó, 1999.
ISBN 963 16 3034 X

[4]
Edward Robbins
Why architects draw
MIT press, 1997.
ISBN 0 262 68098 X

[5]
Dorine van den Beukel
Architecture Drawings
The Peppin Press, 1997.
ISBN 90 5496 041 8

[6]
Geoffrey H. Baker
Le Corbusier - an analysis of form
Spon press, 2001.
ISBN 0 419 16120 1

[7]
Robert McCarter
Frank Lloyd Wright
Phaidon Press, 1997.
ISBN 0 7148 3854 3

[8]
Klaus Albrecht Schröder, Maria Luise Sternath
Albrecht Dürer
Hatje Cantz Verlag, 2003
ISBN 3 7757 1330 1

[9]
Geometrical Considerations
and Nomenclature for Reflectance

F.E. Nicodemus, J.C. Richmond, and J.J. Hsia
Institute for Basic Standards National Bureau of Standards
Washington, D.C. 20234

[10]
Stuart Green
Introduction to Non-Photorealistic Rendering
LightWork Design Ltd.
Siggraph 99 course 17

[11]
Simon Schofield
Non-Photorealistic rendering - The Artist's Perspective
Slade School of Fine Art, University College London
Siggraph 99 course 17

[12]
Aaron Hertzmann
Painterly Image Processing
Media Research Laboratory, New York University
Siggraph 99 course 17

[13]
Aaron Hertzmann
Introduction to 3D Non-Photorealistic Rendering: Silhuettes and Outlines
Media Research Laboratory, New York University
Siggraph 99 course 17

[14]
Michael Batchelder and Kacper Wysocki
TechnicalQuake
McGill University, CS department

[15]
Michael P. Salisbury, Michael T. Wong, John F. Hughes, David H. Salesin
Orientable Textures for Image-Based Pen-and-Ink Illustration
University of Washington

[16]
Szirmay-Kalos László, Antal György, Csonka Ferenc
Háromdimenziós grafika, animáció és játékfejlesztés
ComputerBooks, 2011
ISBN 963 618 303 1

[17]
Adobe Photoshop CS4 Javascript Scripting Reference
© 2008 Adobe Systems Incorporated.

[18]
Adobe Photoshop CS4 Scripting Guide
© 2008 Adobe Systems Incorporated.

Melléklet

a program forráskódja

```
/*
 * Constants
 */

var X_AXIS = new vector(1,0);
var Y_AXIS = new vector(0,1);

//normal vector of hatching (first layer)
var NORMAL = new vector(1,2);

//rotation of layers 2..4 of hatching
var D_FI = new Array(5,2.5,7.5);

//name of target-document
var TARGET = "target_image";

//number of hatch layers
/*
 * 0 layers: [0 .. 0.1[
 * 1 layer:  [0.1 .. 0.25[
 * 2 layers: [0.25 .. 0.4[
 * 3 layers: [0.4 .. 0.55[
 * 4 layers: [0.55 .. 1.0]
 *
 * 0: white
 * 1: black
 */
var INTENSITY_REGIONS = new Array(0.0,0.1,0.25,0.4,0.55,1.0);

//rotations in degrees
var LAYER_ROTATIONS = new Array(0,10,15,20,25);
//convert to radians
for(var i = 0; i<LAYER_ROTATIONS.length; i++) LAYER_ROTATIONS[i] = LAYER_ROTATIONS[i] * Math.PI /
180;

var BRUSH_SIZE = 3.0000;

var WONK_FACTOR = 10;
//-------------------------------------------------------

/*
 * vector class
 * [x,y]
 */
function vector(x,y){
	this.x = x;
	this.y = y;
}

vector.prototype.getX = function(){
	return this.x;
}

vector.prototype.getY = function(){
	return this.y;
}

vector.prototype.toString = function(){
	return (new Array("(", this.x,",", this.y ,")")).join("");
}

vector.prototype.length = function(){
	return Math.sqrt(this.x * this.x + this.y * this.y);
}

vector.prototype.dot = function(v){
	return (this.x * v.x + this.y * v.y);
}

//returns normalized Z component of cross product this x v
vector.prototype.crossZ = function(v){
	var res = (this.x * v.getY() - this.y * v.getX());
	if (res<0) return -1;
	if (res>0) return 1;
	return 0;
```

```
}

//if vector is nullvector, nullvector is returned
vector.prototype.normalize = function(){
	if (this.length() == 0) {
		return new vector(0,0);
	}
	else {
		l= this.length();
		return new vector(this.x / l, this.y / l);
	}
}

vector.prototype.add = function(v){
	return new vector(this.x + v.x, this.y + v.y);
}

vector.prototype.multiplyByScalar = function(n){
	return new vector(n*this.x , n*this.y);
}

vector.prototype.sub = function(v){
	return new vector(this.x - v.x, this.y - v.y);
}

vector.prototype.angleTo = function(v){
	var dotProduct = this.normalize().dot(v.normalize());
	var rad = Math.acos(dotProduct);
	var deg = rad / Math.PI * 180;

	if (this.crossZ(v) < 0) deg = 360 - deg;

	return deg;
}

vector.prototype.rot90 = function(){
	return new vector(this.y,(-1) * this.x);
}

vector.prototype.toString = function(){
	return "(" + this.x + "," + this.y + ")";
}

vector.prototype.rotateRad = function(r){
	return new vector(this.x * Math.cos(r) + this.y * Math.sin(r), -this.x * Math.sin(r) +
this.y * Math.cos(r));
}
//--------------------------------------------------------------------------

/*
* Matrix class
* multiply with vector:
*                     [ x ]
*                     [ y ]
* [ a11 a12 ]
* [ a21 a22 ]
*
* result is vertical vector
*/

function matrix(a11,a12,a21,a22){
	this.a11 = a11;
	this.a12 = a12;
	this.a21 = a21;
	this.a22 = a22;
}

matrix.prototype.toString = function(){
	return (new Array("[" , this.a11 , " " , this.a12 , "]\n[" , this.a21 , " " , this.a22 ,
"]")).join("");
}

matrix.prototype.multiplyVector = function(v){
	return new vector((this.a11 * v.x + this.a12 * v.y),(this.a21 * v.x + this.a22 * v.y));
}

//-----------------------------------------------------------

/*
 * Preferences class
```

```
 */

function prefs(){
	this.rulerUnits = app.preferences.rulerUnits;
	this.displayDialogs = app.displayDialogs;
}

prefs.prototype.save = function(){
	this.rulerUnits = app.preferences.rulerUnits;
	this.displayDialogs = app.displayDialogs;
}

prefs.prototype.restore = function(){
	app.preferences.rulerUnits = this.rulerUnits;
	app.displayDialogs = this.displayDialogs;
}

prefs.prototype.setScriptingPrefs = function(){
	app.preferences.rulerUnits = Units.PIXELS;
	app.displayDialogs = DialogModes.NO;
}

//-------------------------------------Color intensity functions

/*
 * returns average color intensity of selection
 * if nothing is selected, intensity of the whole document is returned
 * (histogram returns: 255: white;0: black)
 * function returns [0..1]
 * 0: white
 * 1: black
 */
function averageIntensity(){
	var activeDoc = app.activeDocument;
	var h = activeDoc.histogram;
	var avg=0;
	var pixelCount=0;

	//clip whites
	for(var i=0;i<250;i++){
		pixelCount += h[i];
		avg += i * h[i];
	}

	if (pixelCount == 0) return 0;
	return 1 - ((avg / pixelCount) / 255);
}

/*
 * samples a 2 x 2 pixel selection
 * returns average intensity
 * smaller sample would fail on borders
 *
 * returns [0..1]
 * 0: white
 * 1: black
 */
function intensityAround(x,y){
	//select one pixel at x,y
	var sampleRadius=1;
	app.activeDocument.selection.select([[x-sampleRadius,y-sampleRadius],[x-sampleRadius,y
+sampleRadius],[x+sampleRadius,y+sampleRadius],[x+sampleRadius,y-
sampleRadius]],SelectionType.REPLACE, 0, true);
	var intensity = averageIntensity();
	app.activeDocument.selection.deselect();
	return intensity;
}

/*
 * returns intensity [0..1]
 * 0: white
 * 1: black
 */
function intensityAt(x,y){
	var i = app.activeDocument.colorSamplers.length;
	var p = new Array(x,y);
	app.activeDocument.colorSamplers.add(p);
	var sampler = app.activeDocument.colorSamplers[i];
	//will use average of red-green-blue channels - gray returns slightly different value
```

```
        var intensity = (sampler.color.rgb.red + sampler.color.rgb.green + sampler.color.rgb.blue)/
3;
        //var intensity = sampler.color.gray.gray;

        sampler.remove();

        return 1 - (intensity / 255);
}

//--------------------------------------------------------------------------------

/*
 * drawLine(start, end)
 * start,end: vector with x and y properties
 * draws on active documents active layer
 *
 * creating new points do not play well with ruler units
 * instead if pixels it assumes points at 72 dpi
 * correct with assumedDPI / realDPI
 * not tested with other units than pixels!
 */

function drawLine(start,end){
        var assumedDPI = 72;
        var realDPI = app.activeDocument.resolution;
        var a = assumedDPI / realDPI;

        //create new path
        var lineArray = new Array();
        lineArray.push(new PathPointInfo);
        lineArray.push(new PathPointInfo);
/*
        lineArray[0].kind = PointKind.CORNERPOINT;
        lineArray[0].anchor = Array(start.x * a,start.y * a);
        lineArray[0].leftDirection = lineArray[0].anchor;
        lineArray[0].rightDirection = lineArray[0].anchor;

        lineArray[1].kind = PointKind.CORNERPOINT;
        lineArray[1].anchor = Array(end.x * a,end.y * a);
        lineArray[1].leftDirection = lineArray[1].anchor;
        lineArray[1].rightDirection = lineArray[1].anchor;
*/

        lineArray[0].kind = PointKind.SMOOTHPOINT;
        lineArray[0].anchor = Array(start.x * a,start.y * a);
        lineArray[0].leftDirection = Array((start.x + end.x)*a/2,(start.y + end.y)*a/2);
        lineArray[0].rightDirection = Array((start.x + end.x)*a/2,(start.y + end.y)*a/2);

        lineArray[1].kind = PointKind.SMOOTHPOINT;
        lineArray[1].anchor = Array(end.x * a,end.y * a);
        lineArray[1].leftDirection = Array((start.x + end.x)*a/2,(start.y + end.y)*a/2);
        lineArray[1].rightDirection = Array((start.x + end.x)*a/2,(start.y + end.y)*a/2)

        var lineSubPathArray = new Array();
        lineSubPathArray.push(new SubPathInfo());
        lineSubPathArray[0].operation = ShapeOperation.SHAPEADD;
        lineSubPathArray[0].closed = false;
        lineSubPathArray[0].entireSubPath = lineArray;

        var myPathItem = app.activeDocument.pathItems.add("line", lineSubPathArray);

        myPathItem.strokePath(ToolType.BRUSH);

        myPathItem.remove();
}

//make hatch lines a bit wonky
function drawWonkyLine(start,end){
        var assumedDPI = 72;
        var realDPI = app.activeDocument.resolution;
        var a = assumedDPI / realDPI;

        var rnd1 = Math.random() /WONK_FACTOR;
        var rnd2 = Math.random() /WONK_FACTOR;

        //create new path
        var lineArray = new Array();
        lineArray.push(new PathPointInfo);
        lineArray.push(new PathPointInfo);
```

```
        lineArray[0].kind = PointKind.SMOOTHPOINT;
        lineArray[0].anchor = Array(start.x * a,start.y * a);
        lineArray[0].leftDirection = Array((start.x + (end.x - start.x)*rnd1)*a,start.y*a);
        lineArray[0].rightDirection = Array((start.x + (end.x - start.x)*rnd1)*a,start.y*a);

        lineArray[1].kind = PointKind.SMOOTHPOINT;
        lineArray[1].anchor = Array(end.x * a,end.y * a);
        lineArray[1].leftDirection = Array((end.x - (end.x - start.x)*rnd2)*a,end.y*a);
        lineArray[1].rightDirection = Array((end.x - (end.x - start.x)*rnd2)*a,end.y*a);

        var lineSubPathArray = new Array();
        lineSubPathArray.push(new SubPathInfo());
        lineSubPathArray[0].operation = ShapeOperation.SHAPEADD;
        lineSubPathArray[0].closed = false;
        lineSubPathArray[0].entireSubPath = lineArray;

        var myPathItem = app.activeDocument.pathItems.add("line", lineSubPathArray);

        myPathItem.strokePath(ToolType.BRUSH);

        myPathItem.remove();
}
//-------------------------------------------------------------------------------

/*
 * rotatePath(pathName,angle)
 * pathName: name of path to be rotated
 * angle: angle of rotation
 *
 * copied from actionlistener output - strangly enough path rotatation cannot be scripted,
selection rotation can
 */

function rotatePath(pathName,angle){
        var idslct = charIDToTypeID( "slct" );
    var desc2 = new ActionDescriptor();
    var idnull = charIDToTypeID( "null" );
        var ref1 = new ActionReference();
        var idPath = charIDToTypeID( "Path" );
        ref1.putName( idPath, pathName );
    desc2.putReference( idnull, ref1 );
        executeAction( idslct, desc2, DialogModes.NO );

        // =======================================================

        var idTrnf = charIDToTypeID( "Trnf" );
    var desc3 = new ActionDescriptor();
    var idnull = charIDToTypeID( "null" );
        var ref2 = new ActionReference();
        var idPath = charIDToTypeID( "Path" );
        var idOrdn = charIDToTypeID( "Ordn" );
        var idTrgt = charIDToTypeID( "Trgt" );
        ref2.putEnumerated( idPath, idOrdn, idTrgt );
    desc3.putReference( idnull, ref2 );
    var idFTcs = charIDToTypeID( "FTcs" );
    var idQCSt = charIDToTypeID( "QCSt" );
    var idQcsa = charIDToTypeID( "Qcsa" );
    desc3.putEnumerated( idFTcs, idQCSt, idQcsa );
    var idOfst = charIDToTypeID( "Ofst" );
        var desc4 = new ActionDescriptor();
        var idHrzn = charIDToTypeID( "Hrzn" );
        var idPxl = charIDToTypeID( "#Pxl" );
        desc4.putUnitDouble( idHrzn, idPxl, 0.000000 );
        var idVrtc = charIDToTypeID( "Vrtc" );
        var idPxl = charIDToTypeID( "#Pxl" );
        desc4.putUnitDouble( idVrtc, idPxl, 0.000000 );
    var idOfst = charIDToTypeID( "Ofst" );
    desc3.putObject( idOfst, idOfst, desc4 );
    var idAngl = charIDToTypeID( "Angl" );
    var idAng = charIDToTypeID( "#Ang" );
    desc3.putUnitDouble( idAngl, idAng, angle );
        executeAction( idTrnf, desc3, DialogModes.NO );
}

//-------------------------------------------------------------------------------

//yay for the crappy documentation: in theory: pathPoint.anchor.x, pathPoint.anchor.y; in reality:
pathPoint.anchor[0] for x, pathPoint.anchor[1] for y;

/*
```

```
 * function getX(Point)
 * retrieves x from photoshop's point structure
 */
function getX(p){
	return p[0];
}


/*
 * function getY(Point)
 * retrieves y from photoshop's point structure
 */
function getY(p){
	return p[1];
}

//--------------------------------------------------------------------------------
/*
 * Code begins here
 *
 */

main();

function main(){
	var P = new prefs();
	P.setScriptingPrefs();

	//setBrushMasterDiameter(BRUSH_SIZE);
//create target document

//for every open document
	var l = app.documents.length;
	for(var i = 0; i<l; i++){
		var activeDoc = app.documents[i];
		app.activeDocument = activeDoc;
		processDocument(activeDoc);
	}
	P.restore();
}

function isMidToneLayer(){
	if (app.activeDocument.name[0] == '0') return true;
	return false;
}

function processDocument(doc){
	//create targetLayer
	var targetLayer = doc.artLayers.add();
	targetLayer.name = doc.name;
	targetLayer.kind = LayerKind.NORMAL;
	targetLayer.blendMode = BlendMode.MULTIPLY;

	doc.activeLayer = doc.layers[doc.layers.length - 1];

	doc.activeLayer.isBackgroundLayer = false;

	//remove all paths - we don't need them
	if (doc.pathItems.length > 0) doc.pathItems.removeAll();

	//magic wand top left corner
	doc.selection.select([[0,0],[0,10],[10,10],[10,0]], SelectionType.REPLACE, 0, true);
	doc.selection.similar(0,true);
	//erase white pixels
	doc.selection.clear();
	//invert selection, create path from contour
	doc.selection.invert();

	//get average intensity for selection
	var avgIntensity = averageIntensity();

	//convert selection to path
	doc.selection.makeWorkPath(5);

	var l = doc.pathItems.length;
	for(var i = 0; i < l; i++){
		if (doc.pathItems[i].kind == PathKind.WORKPATH) {

			doc.pathItems[i].name = "myShapePath";
			//doc.pathItems[i].kind = PathKind.NORMALPATH;
```

```
                shapePath = doc.pathItems[i];
            }
        }

        //count number of hatch layers
        var passes = 0;
        var l = INTENSITY_REGIONS.length;
        for (i = 0; i < l; i++)
            if (avgIntensity > INTENSITY_REGIONS[i])
                passes = i;

        //for each pass hatchLayer on targetlayer
        if (isMidToneLayer()) {
                passes = 1;
        }

        for(var i = 0; i< passes; i++){
            var k=i+1;
            if (isMidToneLayer()) {
                k = 0;
            }
            var n = NORMAL.rotateRad(LAYER_ROTATIONS[k]);
            hatchLayer(doc,shapePath,targetLayer,n,passes);
        }

        //select shapePath, invert, clear; instead of mask!
        doc.activeLayer = targetLayer;
        shapePath.makeSelection();
        doc.selection.invert();
        doc.selection.clear();
        doc.selection.deselect();
}

/*
 * doc: active document
 * n: normal
 * shapePath: path from magic wand selection
 * targetLayer: draw on this layer
 * passes: number of layers in total; divide calculated gradient with passes
 */
function hatchLayer(doc,shapePath,targetLayer,n,passes){
        //save active layer
        var activeLayer = doc.activeLayer;

        //get angle between Y-axis and Normal
        var rotateAngle = n.angleTo(Y_AXIS);

        var duplicatePath = shapePath.duplicate(shapePath.name + "_dpl");
        //rotate duplicate path
        rotatePath(duplicatePath.name,rotateAngle);

        //find indices of minima and maxima
        var bbx = boundingBox(duplicatePath);

        //delete duplicate
        duplicatePath.remove();

        var last = shapePath.subPathItems.length-1;

        //get (x,y) from original path for points with found indices
        var myPoint = shapePath.subPathItems[last].pathPoints[bbx.minXindex].anchor;
        var minXpoint = new vector(getX(myPoint),getY(myPoint));

        var myPoint = shapePath.subPathItems[last].pathPoints[bbx.minYindex].anchor;
        var minYpoint = new vector(getX(myPoint),getY(myPoint));

        var myPoint = shapePath.subPathItems[last].pathPoints[bbx.maxXindex].anchor;
        var maxXpoint = new vector(getX(myPoint),getY(myPoint));

        var myPoint = shapePath.subPathItems[last].pathPoints[bbx.maxYindex].anchor;
        var maxYpoint = new vector(getX(myPoint),getY(myPoint));

        //get intensity values at found points
        //WILL GET ERROR ON TRANSPARENT PIXEL!!! use 3x3 avg
        var minXintensity = intensityAround(minXpoint.x, minXpoint.y);
        var minYintensity = intensityAround(minYpoint.x, minYpoint.y);
        var maxXintensity = intensityAround(maxXpoint.x, maxXpoint.y);
        var maxYintensity = intensityAround(maxYpoint.x, maxYpoint.y);

        //calculate x and y dir gradients.
```

```
var dX = bbx.maxX - bbx.minX;
var dY = bbx.maxY - bbx.minY;
var dXy = bbx.maxXy - bbx.minXy;
var dYx = bbx.maxYx - bbx.minYx;
var diX = (minXintensity - maxXintensity) / passes;
var diY = (minYintensity - maxYintensity) / passes;

/*
 * I.  dy*ry + dyx*rx = -diy
 * II. dx*rx + dxy*ry = -dix
 */

var rx = (-diX + dXy*diY / dY) / (dX - (dYx * dXy) / dY); //x dir gradient
var ry = (-diY - dYx*rx) / dY; //y dir gradient

//Bounding box corners P1: topleft, P2:bottomleft, P3:bottomright, P4:topright
var bBox = boundingBox(shapePath);
var P1 = new vector(bBox.minX,bBox.minY);
var P2 = new vector(bBox.minX,bBox.maxY);
var P3 = new vector(bBox.maxX,bBox.maxY);
var P4 = new vector(bBox.maxX,bBox.minY);

//P is projected on angled bounding box -> seedPoint;
var P = minYpoint;
//Q is last seedPoint;
var Q = maxYpoint;

doc.activeLayer = targetLayer;
drawLine(P1, P2);
drawLine(P2, P3);
drawLine(P3, P4);
drawLine(P4, P1);

//angled bounding box
var i = n.rot90().normalize(); //dir vector to n

var x = (P1.sub(P2)).crossZ(n); //x of matrix A
var y = (P4.sub(P1)).crossZ(n); //y of matrix A

/*
 *     [y*y*(x+y)/2 x*x*(y-x)/2]
 * A = [x*x*(x-y)/2 y*y*(x+y)/2]
 * m = A * n
 */
var A = new matrix(y*y*(x+y)/2, x*x*(y-x)/2, x*x*(x-y)/2, y*y*(x+y)/2); //A * n = m
var m = (A.multiplyVector(n)).normalize(); //n transformed into 1st quarter
var l = m.rot90().normalize(); //dir vector to m

var c = (P1.sub(P2)).length(); //p1p2 length
var d = (P1.sub(P4)).length(); //p1p4 length

var phi = m.angleTo(Y_AXIS); //angle of m to Y_axis
var sinPhi = Math.sin(phi * Math.PI / 180);

var Q1 = P1.sub(l.multiplyByScalar(c * sinPhi)); //angled bbx corner ccw left to P1
var Q2 = P2.add(m.multiplyByScalar(d * sinPhi)); //angled bbx corner ccw left to P2
var Q3 = P3.add(l.multiplyByScalar(c * sinPhi)); //angled bbx corner ccw left to P3
var Q4 = P4.sub(m.multiplyByScalar(d * sinPhi)); //angled bbx corner ccw left to P4

//angled bounding box test
doc.activeLayer = targetLayer;
drawLine(Q1, Q2);
drawLine(Q2, Q3);
drawLine(Q3, Q4);
drawLine(Q4, Q1);
//drawLine(P, P.add(n.multiplyByScalar(100)));

var a = Math.round(i.dot(m)); //multiplyer for m dir component of v
var b = Math.round(i.dot(l)); //multiplyer for l dir component of v
var aa = Math.round((a*a - a)/(-2)); //zero out non-neg m dir components of v
var bb = Math.round((b*b - b)/(-2)); //zero out non-neg l dir components of v

var v1 = m.multiplyByScalar(a*a * (m.dot(Q2.sub(P)))); // (+)
var v2 = Q2.sub(Q1).multiplyByScalar(aa); // (-)
var v3 = l.multiplyByScalar(b*b * l.dot(Q4.sub(P))); // (+)
var v4 = Q4.sub(Q1).multiplyByScalar(bb); // (-)
//projection vector of P -> seedPoint
//v = v1 - v2 + v3 - v4
var v = v1.sub(v2).add(v3).sub(v4);
```

```
        //seedPoint = P + v
        var seedPoint = P.add(v);
        var lastSeedPoint = seedPoint.add(m.multiplyByScalar(dY * b)).add(l.multiplyByScalar(dY *
a));

        //multiplier for hatchline spacing
        //hatching length
        //draw hatchlines on targetLayer
        var fillRatio = 0;
        if(ry < 0) { //lightens
                fillRatio = (INTENSITY_REGIONS[1] + INTENSITY_REGIONS[2]) / 2;
        }
        else { //darkens
                fillRatio = (INTENSITY_REGIONS[1] + INTENSITY_REGIONS[1]) / 2;
        }

        //fillRatio = fillRatio / passes;

        var j = Q1.sub(Q2).multiplyByScalar(a).add(Q1.sub(Q4).multiplyByScalar(b));
        var patternWidth = BRUSH_SIZE / fillRatio;
        var sumSteps = 0;
        var N = n.normalize();

        doc.activeLayer = targetLayer;

        while(sumSteps < dY){
                drawWonkyLine(seedPoint, seedPoint.add(j));
                seedPoint = seedPoint.add(N.multiplyByScalar(patternWidth));
                sumSteps = sumSteps + patternWidth;

                fillRatio = fillRatio + ry * patternWidth;
                patternWidth = BRUSH_SIZE / fillRatio;

                if(patternWidth < 0) break; //shouldn't happen, but...
        }

        //reset initial active layer
        doc.activeLayer = activeLayer;
}

/*
 * boundingBox
 * finds bounding box of shapePath
 * returns minX, minY, maxX, maxY, minXindex, minYindex, maxXindex, maxYindex,
 * minXy,minYx,maxXy,maxYx
 */
function boundingBox(shapePath){
        var result = new Object();

        var last = shapePath.subPathItems.length-1;

        var myPoint = shapePath.subPathItems[last].pathPoints[0].anchor
        var minX = getX(myPoint);
        var minY = getY(myPoint);
        var maxX = getX(myPoint);
        var maxY = getY(myPoint);

        var minXindex = 0; //index of point with minimal X coordinate in path
        var maxXindex = 0; //index of point with maximal X coordinate in path
        var minYindex = 0; //index of point with minimal Y coordinate in path
        var maxYindex = 0; //index of point with maximal Y coordinate in path

        var l = shapePath.subPathItems[last].pathPoints.length;
        for(var i = 0; i<l; i++){
                var myPoint = shapePath.subPathItems[last].pathPoints[i].anchor;
                if (getX(myPoint) > maxX){
                        maxX = getX(myPoint);
                        maxXindex = i;
                }
                if (getX(myPoint) < minX){
                        minX = getX(myPoint);
                        minXindex = i;
                }
                if (getY(myPoint) > maxY){
                        maxY = getY(myPoint);
                        maxYindex = i;
                }
                if (getY(myPoint) < minY){
                        minY = getY(myPoint);
                        minYindex = i;
```

```
            }
        }

        result.minX = minX;
        result.minY = minY;
        result.maxX = maxX;
        result.maxY = maxY;
        result.minXindex = minXindex;
        result.minYindex = minYindex;
        result.maxXindex = maxXindex;
        result.maxYindex = maxYindex;

        var myPoint = shapePath.subPathItems[last].pathPoints[minXindex].anchor;
        result.minXy = getY(myPoint); //y coordinate of point minX

        var myPoint = shapePath.subPathItems[last].pathPoints[minYindex].anchor;
        result.minYx = getX(myPoint); //x coordinate of point minY

        var myPoint = shapePath.subPathItems[last].pathPoints[maxXindex].anchor;
        result.maxXy = getY(myPoint); //y coordinate of point maxX

        var myPoint = shapePath.subPathItems[last].pathPoints[maxYindex].anchor;
        result.maxYx = getY(myPoint); //x coordinate of point maxY

        return result;
}
```