Tento text obsahuje možné otázky k tématům z předmětu Počítačová grafika I. (zdroj matfyz fórum) a různé poznámky k jejich odpovědím. Ty poznámky jsou psané na rychlo a navíc vůbec nemusí být správně, tedy ať je čtenář bere spíše jako možnou inspiraci. ;) Rok vzniku je 2011 (zimní zkouškové období).

Markéta Popelová.

2D GRAFIKA

Barevné systémy

1. HSV - napsat, co víte, plus nastínit převod RGB to HSV

Rastrové obrázky a grafické formáty

2. Navrhněte grafické formáty/kompresní algoritmy pro následující činnosti – uložení screenshotu ve windows, poslání fotky emailem (+náhled té fotky), publikace fotky na webu, digit. fotku k archivaci.

Zobrazování monochromatických obrázků

Gamma korekce

3. Napište vzoreček, který charakterizuje svítivost (intenzitu) pixelu na CRT monitoru v závislosti na přivedeném vstupním signálu (napětí). Co z toho plyne pro zobrazování v počítačové grafice? (Tady pozor, ten vzorecek ve slajdech je spatne a kdo nechodil na prednasky, tak to nevi!). Spravne je I = K*(V + epsilon)^gama.
   Vztah mezi intenzitou a jasem není lineární (nebo ho lidské oko nevnímá lineárně)? Intenzita zase nezávisí lineráně na napětí přiváděném na stínítko monitoru. Ten vztah je I é K(V+\epsilon)^gamma, kde K,\epsilon a \gamma jsou vhodné konstanty, V je napětí a I je jas. Tedy pro rovnoměrně odstupňované jasové odstíny je třeba použít logaritmickou stupnici intenzit. A počet zobrazovaných obstínů závisí na dynamickém rozsahu výstupního zařízení.

Půltónování

4. Navrhnete pultonovaci matici pro laserovou tiskarnu s 99 odstiny.
   *Pozor. Na to nestačí matice 10x10 (asi že to není hezký násobek), je třeba matice 14x14, neboť pak to vyjde následovně: prázdná matice odpovídá bílé. Dále vždy pro každý další odstín se přidá po dvou tečkách → tím získáme 98+1 odstínů, na což máme 98*2 políček matice. Tu matici tam rozestavit nějak po spirále. Aby to byl tečkový nepravidelný inkremesntální rastrový filtr.*
5. Půltónovací matice pro 51 odstínů na ČB laserové tiskárně
   Klasicky teckovy raster obdlznikovy, ktory mame v slajdoch. Pozor! Inkrementalne pravidelne rastre su vhodne len pre ihlickove tlaciarne, na ostatnych sa tecky rozplyvaju. Pre ne treba navrhnut teckovy raster.
   http://forum.matfyz.info/viewtopic.php?f=98&t=2694
6. Navrhěte rozptylovací matici 8x8 pro obrazovku (stačí myšlenka, není třeba ji celou vypisovat)
   - *Na maticové rozptylování se dá použít libovolná matice půltónovacího rastru (třeba pravidelná inkrementální). Následně několik pixelů dostane jednu společnou matici. Každý pixel se pak rozhodne, zda na jeho místě bude tečka či ne podle následujícího: namapuje se do matice [x mod n, y mod n] a vezme, zda daná hodnota v matici je menší než hodnota jeho barvy (jasu). → Zkresluje drobné detaily (čáry).*
   - *Dále je možné náhodné rozptylování, které prostě vygeneruje náhodné číslo a porovná ho s hodnotou barvy (jasu) v pixelu a podle toho vykreslí tečku či ne. Rychlé, fajn nahodilost, ale u ČB často příliš zašumělé.*
   - *Třetí možnost je použít nějakou metodu distribuce chyby, třeba Floyd-Steinberga. Obecně vezme hodnotu, zaokrouhlí ji na nejbližší zobrazovanou hodnotu (u ČB 0/1) a podle toho vykreslí + nevykreslené rozdělí mezi okolní nenakreslené pixely (FS 7/16,1/16,5/16,3/16). Pro akumulaci chyb na*

následující řádce používá pomocný buffer. Bezva výstup, ale pomalejší, nelze se vracet, nutný buffer, nutno kreslit po řádkách.

Metody distribuce chyby

7. Popis ake by si pouzil datove struktury pre Floyd Steinbergov algoritmus distribuce chyby na obrazok s 255 farbami. Vystupny obrazok ma byt ciernobiely. Jak by to slo zpresnit cela ta distribuce?
   Chcel by som spresnit zadanie 2. otazky: datovu strukturu sme mali navrhnut pre Floyd Steinberg-a, pripadne pre F. Sierra. (staci celociselny buffer pre jeden riadok, tj int buf[width + cca 1])

   Dalej sme mali navrhnut sposob "presnej distribucie chyby." Pod pojmom presna distribucia chyby sa skryva problem ako v pripade celociselneho buffera reprezentovat napr 3 * 5/16 co najpresnejsie. Pri deleni totizto dochadza k zmensovaniu chyby (napr 3 * 5/16: namiesto skutocnej chyby 15/16 sa ulozi 0 po celociselnom deleni). Mozne riesenia:
   - vyuzit buffer float-ov
   - pri intovom bufferi distribuovat chybu do n-1 pixelov klasickym sposobom tj celkova_chyba * vahovy_koeficient a chybu posledneho pixelu urcit ako celkova_chyba - sucet ulozenych chyb v n-1 pixeloch
   - pri intovom bufferi vobec nedelit a uchovavat 16-nasobne vacsie hodnoty chyb (v pripade FS), tj napr namiesto 3*5/16 = 15/16, uchovavat priamo 3*5 a vobec nedelit 16 (delime az ked nakumulovanu chybu pouzivame tj pri spracovani pixelu
   - vtedy mozeme zvysok po deleni presunut na okolite nespracovane pixely, tj okolite chybove policka buffera)
8. Navrhnete upravu Floyd-Steinbergova algoritmu tak, aby na stejnem vstupu nedaval pokazde stejny vystup.

Reprodukce (zobrazování) a redukce barev

Metody redukce barev:

- *Univerzální barevná paleta + následné rozptylování, ideálně distribuce chyby. Rychlé, ale ne až tak fajn výsledky.*
- *Adaptovaná barevná paleta optimalizovaná pro daný obrázek. Opět lze využít některou z metod rozptylování.*
  - *Metoda shlukové analýzy. Vytvoří si barevný histogram všech použitých barev včetně četností. V něm pak spojuje barvy do skupin, až jich je jen daný počet. Spojuje „blízké barvy" (metriky minimální/maximální vzdálenost složek, rozptyl, atd.). Vylepšení octree, který začíná s N barvami jako N skupinami a s každou novou barvou nějaké dvě shlukne, tedy končí opět s N skupinami. Což je rychlejší, ale dává horší výsledky (není to symetrické).*
  - *Heckbertův algoritmus (median cut). Opět zaznamená barvy do 3D histogramu. Obalí je jednou skupinou (kvádr), kterou postupně dělí až na daný počet skupin. Kterou skupinu a jak dělit se různí. Typicky se dělí nejdelší hrana kvádru v těžišti/ polovině/ průměru/ mediánu vstupních pixelů. Ideální je podle mne medián.*

9. U **median-cut** algoritmu pro redukci palety napište metody výběru místa rozdělení a určete jejich kvalitu.
10. Navrhnout dva algoritmy pro redukci barev v obrazku - melo se predpokladat, ze vstupni obrazek ma radove stovky tisic barev, vysledek se mel ulozit v GIFu (tzn. pocet barev se mel zredukovat na 256). Jeden algoritmus mel byt co nejrychlejsi, pricemz na kvalite vystupu moc nezalezi, druhy mel mit naopak co nejvetsi kvalitu vystupu a to i za cenu, ze bude pomaly.

Kreslení čar a křivek v rovině

11. Popsat princip Midpoint (Bresenhamovych) algoritmu na kresbu car (kruznic, elips) - obecne. Jako priklad uvest treba odvozeni algoritmu pro kruznici.

Kreslení úsečky – Bresenhamův algoritmus:

Na vstupu [x1,y1] a [x2,y2] koncové body úsečky. Zvolím [x,y]:=[x1,y1]. Nechť |dy| < |dx|. Dále budu postupně zvětšovat x v každém kroku. V každém kroku se rozhodnu, zda zvětšit i y, nebo ještě ne. To podle toho, zda má úsečka v bodě s x-ovou souřadnicí x y-ovou souřadnici blíže y nebo y+1.

Na začátku položme E:=dy/dx,x:=x1,y:=y1. E budeme postupně zvětšovat o směrnici dy/dx a zjišťovat, zda již překročilo ½ či ne. Pokud ne, pokuze E+=dy/dx. Jinak E+=dy/dx-1/2 a y+=1. Tedy:

$$E' = E + dy/dx <= 1/2$$
$$2dxE' = 2dxE + 2dy <= dx$$
$$dx(2E'-1) = dx(2E-1) + 2dy <= 0$$
$$D' = D + 2dy <= 0$$

Tedy jsme položili D:=dx(2E-1). Z toho dovodíme, že na začátku D:=2dy-dx. Pak pokuď je D + 2y <= 0, tak D+=2dy a y nechme, jinak D+=2dy-dx a y+=1.

Kreslení kružnice – Bresenhamův algoritmus:

Nakreslí se jen 1/8 oblouku (zbytek je symetrický). Označme F(M) rovnici kružnice: F(M):=(M_x)^2 + (M_y)^2 – R^2. Dosadíme-li do ní nějaký bod M=[x,y], zjistíme zda je bod vně kružnici (F(M)>=0) či uvnitř (F(M)<0).

Začneme tedy s body x:=0, y:=R. F(M) = x^2 + y^2 – R^2 = 0. Následně nás zajímá, zda je bod M'=[x+1,y-1/2]:

- *uvnitř, tedy F(M')<0, pak x+=1,y nech a F(M'')=(x+2)^2 + (y-1/2)^2 -R^2 = F(M') + 2x + 3.*
- *vně, tedy F(M')>=0, pak x+=1,y-=1 a F(M'')=(x+2)^2 + (y-3/2)^2 – R^2 = F(M') + 2x – 2y + 5.*

Tímto máme hotový algoritmus. Místo F(M) pro měnící se M budeme psát D. Na začátku D:=(R-1/2)^2 – R^2 = 1/4 – R. (Můžeme však počítat 1-R, neboť jen porovnáváme s 0.) Dále pokud D<0, tak D+=2x+3, x+=1. Jinak D+=2x-2y+5,y-=1,y+=1.

12. Kubicka krivka v rovine – diferencny algoritmus (nechcel konkretne v nejakom prog. jazyku, skor principy). Jeho vyhody, nevyhody, implementacne problemy.
    Netreba sa ucit matice, len popisat princip, ako ich ziskat (napoveda: c=P(2h)-P(h), atd...). Chcel, aby tam bol spomenuty aj problem s presnostou a pricnip adaptivneho algoritmu, cize dynamicke menenie kroku tak, aby sa jednak zbytocne nevykreslovali niektore pixely (funkcia ''ide pomaly''), ale tiez aby funkcia zostala spojita a nevznikali diery (funkcie "ide rychlo").

Diferenční algoritmus: Vypočítává hodnoty polynomu P v bodě 0,h,2h,... pomocí čtveřice a,b,c,d (to se vykresluje na výstup), kterou vždy zleva přenásobí maticí – na inicializaci, krok (d+=c; c+=b; b+=a; ale s původními hodnotami). Dále zvětšení kroku na dvojnásobek či zmenšení na polovinu. Algoritmus se nazývá adaptivní, neboť je sám schopen rozhodnout, jak si upravit krok, aby nenakreslil nějaké pixely moc daleko od sebe – či jimi neplýtval. Hodnoty a,b,c,d se mohou reprezentovat skoro-celočíselně, tedy v 32-bitových registrech s pevnou desetinnou čárkou. Pak ale kvůli akumulaci chyb nelze nakreslit více než 100px za sebou, což jde ještě nějak prodloužit tím, že se zvětší přesnost členu d, či se ta přesnost řídí dynamicky (reaktivně ;)).

Vyplňování n-úhelníka

13. Navrhněte ve vašem oblíbeném programovacím jazyce datovou strukturu pro uložení hrany pro řádkový algoritmus **vyplňování n-úhelníku** a řekněte jaké 2 druhy vyplňování se užívají. Napište stručný komentář k jednotlivým parametrům.

Ořezávání v rovině

14. Odvodit vzorecek pro vypocet pruseciku pri parametrickem orezavani obdelnikovym oknem, dale pak urcit kolik a jakych aritmetickych operaci se provede pri orezavani usecky jednou polorovinou (operace se mely rozdelit na +/-, *, /, porovnani apod.).
15. Orezavani polygon vs obdelnik popsat algoritmus & kolik se provede operaci (+,-,*,/,sqrt,sin,cos,...,porovnani) u jednotlivych vrcholu v nejhorsim pripade.
16. Odvoďte postup pro parametrické ořezání úsečky, kolik potřebuje operací pro ořezání úsečka vs. polorovina.

Antialiasing

17. Anti-aliasing, popis co to je, jeho vyznam, na co sa pouziva + popis obecne metody antialiasovania (nie konkretne algoritmy). Zakladni principy implementace (ne konkretni algoritmy).

Kódování rastrových obrázků

18. Upravte quad-tree pro reprezentaci obdelnikoveho bitmapoveho obrazku libovolne velikosti. (Slo opravdu jen o odpoved na: co delat kdyz na startu konstrukce quad-tree pro bitmapu dostanete obdelnik.) Navrhněte způsoby, jak kódovat kvadrantovým stromem rastrové obdélníky libovolných rozměrů.
19. Operacia XOR na kvadrantovom strome.

Procházím oba stromy a jakmile narazím v jednom na list, tak už jasně vidím, co bude ve výsledku (je-li list prázdný, tak tam bude to z druhého podstromu, co není prázdné; je-li plný, tak naopak prázdné kusy druhého podstromu).

20. Quad tree – navhrnout obrázek 64x64, kde bude mít quad tree nejmenší účinnost.
21. Popiste algoritmus pro operaci rozdilu mezi stejne velkymi kvadrantovymi stromy (quadtree). Strom obsahuje pouze binarni hodnoty "1" - uvnitr mnoziny, "0" - vne.
22. Máte 24 záznamů pro **X-transition list** a máme určit jaký největší obrazec lze zobrazit a jaký bude mít obsah.

3D GRAFIKA

23. Doslova: "Jakymi daty byste reprezentovali kameru pro rovnobeznou kolmou projekci." Popiste zpusob ziskani zobrazovaci matice teto kamery. Definovat parametry kamery pri kolmem rovnobeznem promitani a popsat vytvoreni promitaci matice.

Pro implementaci kamery potřebujeme znát rovinu průmětny a směr pohledu (tedy normálový vektor průmětny). Následně bod v prostoru zobrazíme na průmětnu (a navíc si můžeme pamatovat jeho hloubku = vzdálenost od roviny průmětny). Transformační matici vytvoříme tak, že použijeme transformace na to, aby osa promítání (normálový vektor průmětny) splynula s osou z (např.) a pak bod v tomto novém prostoru [x,y,z] → promítneme snadno [x,y] zahozením třetí složky (resp. ta znázorňuje hloubku). Tomuto tedy odpovídá jednotková matice, která má z-sloupci nuly. Obě matice spolu vynásobíme. (Typicky ale máme matici 4x4, neboť to znázorňujeme v homogenním souřadném systému, což nám dovoluje jaksi pracovat s nekonečnem – ale výhodné především pro perspektivní projekci).

On má na slidech: směr pohledu N (= normálový vektor průmětny), svislý vektor u (čímž ale asi myslí bod, který zobrazujeme – snad?), převedení do základního pohledu Cs(S,uxN,u,N).

24. Popis udaje potrebne pre reprezentaciu kamery, ktora snima perspektivne, objasni sposob transformacie priestoru a aku velku maticu budes potrebovat. Jake parametry potrebuju na urceni perspektivni kamery. Jak z danych parametru sestavim projekcni matici. Kolik prvku ta matice bude mit.

*Potřebujeme střed projekce S, normálový vektor průmětny N, vzdálenost od průmětny d (ale to je prostě vzdálenost S od N). Pak svislý vektor u převedeme do základní polohy (tzn. střed projekce do počátku, normálu průmětny, tedy směr poheldu, do osy z): Cs(S,uxN,u,N). A následná perspektivní projekce vypadá [x*d/z,y*d/z,z]. Matice bude 4x4.*

Viz http://herakles.zcu.cz/education/zpg/cviceni.php?no=5.

Reprezentace 3D scény a hierarchický model

25. Mame databazi objektu, ktere konstrukter muze umistit do 3D sceny, jake matematicke operace se provedou, kdyz je do sceny umistena nova instance objektu? Dale pak se melo rozhodnout, zda je mozne do sceny umistit vice nez jednu instanci od nejakeho

objektu a pokud ano, tak rict, cim se ty instance od sebe budou lisit.

Musí se vypočítat transformační matice pro umístění objektu do scény. Je to možné, liší se pak umístěním. (?).

26. Popis vhodne datove struktury pre CSG (elementarne modely + operacie). Ber do uvahy, ze ich chceme zobrazovat ray-castingom.

V listech jsou elementární tělesa. V uzlech jsou množinové operace na tělesech, které jsou znázorněny maticově. Elementární tělesa musí umět odpovědět, zda je bod uvnitř tělesa. A také musí dát průsečíky s paprskem (přímkou) a říci, kdy vchází dovnitř a kdy vychází ven.

27. CSG - popsat algoritmus vržení paprsku, kolik vykonáme porovnání s elem. tělesy, pokud má scéna devet operací sjednocení.

Na každý paprsek provedeme protnutí s elementárním tělesem → a výsledné body transformujeme v uzlech.

28. Scena v CSG (primitiva, mnozinove operace, …) - idealni datova struktura (vcetne naznaceni v oblibenem prog jazyce) s durazem na to ze se to bude pouzivat na raytracing a editaci (ta editace ve vzorovem reseni byla implementovana napr uz rovnou poznacenyma inverznima maticema)

2-manifold v povrchové reprezentaci scény: pro každý bod existuje okolí, které je topologicky ekvivalentní s rovinou.

OpenGL

Souřadné systémy: Souřadnice modelu → modelová transformace → světové souřadnice → pohledová transformac → +souřadnice kamery → projekční transformace → +ořezávací souřadnice → perspektivní dělení (?) → normalizovaný prostor → okénkový tranformace (ořezávání) → souřadnice v okně.

Phongovo stínování: jednoduché na výpočet, +- věrohodné fyzikálně, kvůli lesklé složce je nejlepší ho počítat v každém pixelu s interpolací normály. Světlo se skládá ze 3 složek: zbytkové světlo (místo odrazů), difuzní světlo (ideálně matné těleso) a lesklý odraz (neostrý odraz, odlesk).

Trojúhelníková síť

29. Navrhnout datovou strukturu pro databázi reprezentující scénu v B-rep (ploškovém) modelu kde stěny jsou trojúhelníky + co by tam ještě mohlo být za pomocné data.

Je potřeba si pamatovat všechny použité vrcholy. A potom trojice vrcholů, které znázorňují plošky a odkazují do seznamu vrcholů (který je nějak očíslovaný). Corner table to řeší takto: tabulka vrcholů (souřadnice, normála, barva, texturové souřadnice) a tabulka rohů (z toho se nějak dá poznat celý trojúhelník a jeho sousedé či co). Jo tak – tabulka rohů uchovává pro každý vrchol tyto údaje: on sám, protější roh – a jestli snad i sousední rohy...?

3D viditelnost

30. Popište nejnáročnější test v **malířově algoritmu**. Jak se pozná, že došlo k zacyklení plošek a jak se tento problém řeší.
31. Popisat algoritmus Z-buffer, vyhody, nevyhody, ake sceny sa nim mozu generovat, zlozitost, použití.
    Zlozitost zavisi na pocte pixelov vsetkych plosiek. Je to pixelovy algoritmus. Z-buffer je snad jediny algoritmus, ktory dokaze vykreslit akukolkvek scenu, ktora je rozlozitelna na pixely. (ploskovy model, kreslenie 2-rozmernych funkcii,...). Pri ploskovom modeli sa mozu plosky zacyklit aj presekavat.
32. Popište Warnockův řádkový algoritmus (podle mě tu mělo být Watkinsonův a tak jsem raděj napsal obě varianty) pro vykreslování 3D scény. Řekněte, kdy bude mít největší složitost (při konstantním počtu objektů ve scéně).
33. Popsat vizualizacni cast Watkinsova algoritmu (v podstate jen co se dela pri pruchodu radkem a co se zobrazuje). Navrhnout scenu N trojuhelniku tak, aby mel Watkinson co nejvetsi casovou slozitost. Watkinsův alg. řádkového vykreslování – popsat jak řeší viditelnost, najít nejhorší možný případ scény s N stěnami (trojúhelníčky), kde mu to bude asymptoticky trvat nejdéle. (stěny se nesmí

prosekávat).

OSTATNÍ

Nepodařilo se mi zařadit jinam.

34. Navrhněte datovou strukturu pro bitovou masku ve window manageru. Měla by umět rychle ořezávat (průnik bitová maska a okno), dělat rozdíl 2 bitových masek a určovat neprázdnost bitové masky (zda je nutno překreslovat obrazovku).
35. Dávná ústní zkouška: reprezentace barev, řádkový algoritmus, malířův algoritmus, ořezávání v n-úhelníku, palety, reprezentace 3D scény...