Lösningsskiss till tentamen TNM061 2014-06-03

Lösningarna är inte alltid fullständiga. I vissa fall ges endast översiktliga anvisningar om vad som krävs för full poäng.

1 a) Man beräknar ekvationen separat för flera färgkanaler, exempelvis genom att använda RGB-vektorer för alla ljusintensiteter och materialfärger (I_x och k_x) och tolka en multiplikation av två färger som komponentvis multiplikation av RGB-vektorerna.

1 b) Skalärprodukten blir negativ. Ljusbidraget ska då sättas till noll. Detta motsvarar att ljuskälllan ligger bakom ytan, och då ska den inte belysa ytan direkt. Det är alltså en god modell av vad som händer i verkligheten.

1 c) För att räkna på flera ljuskällor beräknar man varje ljuskällas bidrag separat, och summerar resultatet. Det allmänna ljuset (om det används) brukar anses gemensamt för hela scenen, men de diffusa och spekulära termerna beräknas separat för varje ljuskälla:

$$I = I_a k_a + \sum_i I_{d,i} k_d (\hat{N} \cdot \hat{L}_i) + I_{s,i} k_s (\hat{R}_i \cdot \hat{V})^n$$

1 d) Ljusintensiteten beror av avståndet till ljuskällan, och eventuellt även av riktningen. Färgen på ytan beror oftast av var på ytan man tittar (texturmappning). Riktningen till en ljuskälla som inte ligger oändligt långt bort varierar över ytan av ett objekt.

2 a) Den vänstra bilden visar en reflexion av tekannans delar i sig själv. Det är den som är renderad med raytracing. I den högra bilden visar reflexionen bara andra objekt. Det är den som är renderad med environment mapping.

2 b) Reflexionsmappning är en enkel tabellslagning i en textur, även om texturkoordinaterna beräknas på annat sätt än för en direkt mappning på objektets yta. Den lämpar sig väl för snabbb rendering med en GPU. Reflexionsmappen kan beräknas långt i förväg och lagras på fil, eller beräknas dynamiskt i ett förberäkningspass.

2 c) Raytracing är en olämplig metod för realtidsrendering, eftersom dagens GPU-arkitekturer renderar varje triangel för sig. Man kan alltså inte skicka strålar mellan objekt. Hela scenen finns inte tillgänglig i grafikkortets minne samtidigt, åtminstone inte på en sådan form att den går att söka i på ett enkelt sätt.

3 a) För full poäng skall minst följande förklaras: strålen utgår från ögat/kameran. Vid träff på en diffus yta beräknas en lokal reflexionsmodell. Vid träff på speglande yta beräknas en reflekterad stråle som behandlas rekursivt på samma sätt som ursprungsstrålen. Vid träff på en genomskinlig yta beräknas en bruten stråle som också behandlas rekursivt. Tydlig figur krävs för full poäng.

3 b) När den lokala ljusmodellen beräknas för diffus belysning skickar man en stråle mot ljuskällan, en så kallad “shadow feeler ray”. Om något annat objekt ligger ivägen tas inte den ljuskällans bidrag med i beräkningen.

4 a) Förhållandevis få trianglar gör modellen utrymmessnål i minnet och snabb att rendera.

4 b) Det mesta av den visuella skillnaden mellan olika spelare görs med byte av texturbilder. Då kan samma modell användas för många spelare, vilket sparar minne och gör renderingen snabbare.

4 c) När det är många objekt i scenen förenklar man renderingen genom att minska antalet trianglar ytterligare, minska upplösningen på texturbilderna och använda en enklare shader som bara beräknar diffus reflexion.

5) Metoden kallas “bump mapping” eller “normal mapping”. En slät yta tilldelas normaler som varierar över ytan beroende på en texturbild. Geometriskt sett är ytan slät, men för ljusreflexionen ser det ut som om den hade en varierande normal, dvs att den är skrovlig. Förklarande figur krävs för full poäng. Bump mapping fungerar något annorlunda än normal mapping. Båda är OK som förklaring.

6) I närbild ser man att ansiktsdragen har skarpa konturer, men att de är något kantiga. Det är därför tydligt att de består av trianglar. Mönstret på kroppen blir däremot litet pixligt och suddigt när man kommer riktigt nära. Det är därför tydligt att det är en bildbaserad textur. Anledningen till denna lösning är dels att ansiktet ofta är i fokus och behöver vara skarpt även i närbild, men framför allt beror det på att ansiktet ska animeras för att munnen och anletsdragen ska röra sig när figuren talar och agerar. En fri animering görs enkelt med polygoner, men är förhållandevis svårt, nästan omöjligt rentav, att göra med bildbaserade texturer.

7 a) Visa att $p(0)=p_0$, och att $p(1)=p3$. Eftersom $p_{3A}=p_{0B}$ följer att kurvan är kontinuerlig i skarven.

7 b) Derivera polynomet med avseende på t . Visa att $p'(0)=3(p_1-p_0)$ och att $p'(1)=3(p_3-p_2)$. Att p_{2A} , $p_{3A}=p_{0B}$ och p_{1B} ligger på samma avstånd längs en rät linje innebär att $p_{3A}-p_{2A}=p_{1B}-p_{0B}$. Derivatan är alltså kontinuerlig i skarven.

7 c) Derivera en gång till. De givna bivillkoren räcker inte för att garantera kontinuitet för andraderivatan i skarven. Det enkla svaret är “nej”. Det mer formellt korrekta svaret är “nej, inte nödvändigtvis, eftersom de givna förutsättningarna inte är tillräckliga”.

8 a) För att följa strukturen i koden skall kroppen vara barn till rotnoden. Armar, ben och huvud skall läggas som barn till var sin gren därifrån, med en translation och en rotation i var sin translationsgrupp för att medge separat kontroll av de animerade rotationerna. Translationerna skall ske före rotationerna i hierarkin. Figur krävs för full poäng. Det skall framgå vilka delar av koden som motsvarar olika delar av grafen.

8 b) Multiplikationen skall göras i ordningen $M \cdot T$ så att senare transformationer görs i det aktuella koordinatsystemet snarare än i globala koordinater. (Allt eftersom man närmar sig objektet i grafen skall transformationsmatrisen stå närmare koordinatvektorn. Matrisen direkt ovanför objektet skall operera på objektets koordinater.)

8 c) Det skall läggas till kod mellan `M.init()` och `TeddyBody.render(M)` som ändrar transformationsmatrisen M. Det är lämpligt att göra en translation och minst en rotation, eventuellt flera rotationer runt olika axlar, med translationen före rotationen. Att göra rotationen först är direkt olämpligt.