VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

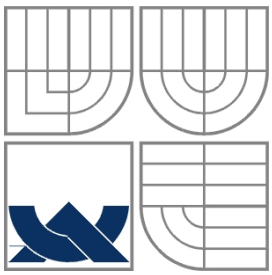
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZOVANIE A INTERAKCIA OBJEKTOV UMIESTNENÝCH DO 3D KRAJINY

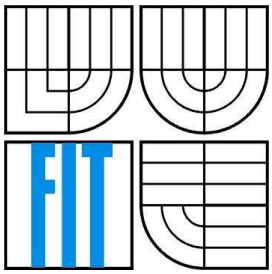
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Tomáš Koza
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZOVÁNÍ A INTERAKCE OBJEKTŮ UMÍSTĚNÝCH DO 3D KRAJINY

RENDERING AND INTERACTION OF OBJECTS IN 3D LANDSCAPE

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

Tomáš Koza

VEDOUCÍ PRÁCE
SUPERVISOR

Herout Adam, Ing., Ph.D.

BRNO 2007

Abstrakt

Cílem této práce bylo vytvořit systém powerupů (3D objekty) pro program TankGame. Práce obsahuje teoretický základ pro zobrazování objeků v 3D scéně, popisuje techniky používané při zobrazování těchto objektů. Dále poskytuje přehled procesu renderování 3D scény a popis implementace konkrétního zobrazovacího stroje v programu TankGame. Na závěr popisuje implementaci konkrétního systému zobrazovaných objektů (powerupů) pro program TankGame.

Klíčová slova

3D objekt, 3D scéna, renderování, stínování, osvětlovací model, radiozita, ray tracing, geometrie, rasterizace, 3D pipeline, VRML, ASE, OpenGL, kamera, TankGame, powerup .

Abstract

Goal of this bachelor’s thesis was to create a system of powerups for program TankGame. This document contents theoretical base for rendering of objects in 3D scene, it describes algorithms used in rendering of these objects. It also contains overview of rendering process and description of concrete graphical engine used in program TankGame. In the end there is description of concrete implementation of renderable objects (powerups) for program TankGame.

Keywords

3D object, 3D scene, rendering, shading, lighting model, radiosity, ray tracing, geometry, rasterisation, 3D pipeline, VRML, ASE, OpenGL, view, TankGame, powerup.
Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Citace

Tomáš Koza: Zobrazovanie a interakcia objektov umiestnených do 3D krajiny, bakalářská práce, Brno, FIT VUT v Brně, 2007

Zobrazovanie a interakcia objektov umiestnených do 3D krajiny

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Adama Herouta, Ing., Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Rád bych poděkoval Adamu Heroutovi, Ing., Ph.D. za pomoc při vytváření této práce.

Obsah

Úvod

Zadanie bakalárskej práce:

1. Zoznámte sa s problematikou zobrazovania 3D krajiny v reálnom čase.
2. Zoznámte sa s problematikou zobrazovania objektov v rozmerných scénach.
3. Popíšte techniky používané pri zobrazovaní objektov rozmiestnených v rozmernej scéne.
4. Navrhnite štruktúru tried vhodnú pre umiestňovanie objektov do krajiny pre systém pracujúci v reálnom čase.
5. Implementujte vhodnú časť navrhnutého systému tried, tak aby bola zlučiteľná s programom TankGame
6. Vyhodnoťte vlastnosti navrhnutého systému, demonštrujte použiteľnosť na príkladoch
7. Zhodnoťte dosiahnuté výsledky a navrhnite možnosti pokračovania projektu; Vytvorte plagát pre prezentovanie projektu

Nasledujúci text bakalárskej práce bude najprv pojednávať o problematike zobrazovania 3D objektov, popíše techniky zobrazovania 3D objektov, techniky zobrazovania rozmerných scén a problém umiestňovania 3D objektov do rozľahlej krajiny. V prvej kapitole sa bude taktiež pojednávať o problematike interakcie takýchto objektov. Nezanedbateľnou časťou prvej kapitoly je popis zobrazovania objektov a komplexnejších scén v reálnom čase, ako typický príklad takejto aplikácie bude demonštrovaná problematika renderovania objektov v 3D počítačových hrách.

Kapitola číslo dva sa venuje konkrétnej implementácii systému zobrazovania 3D objektov v rozmernej 3D scéne v programe TankGame. V kapitole nájdete pohľad na program TankGame vo všeobecnosti a na jeho techniky zobrazovania 3D objektov.

Tretia kapitola sa zameriava na konkrétnu implementáciu systému umiestňovania a interakcie 3D objektov pre program TankGame s názvom TPowerUp. Kapitola obsahuje podrobnejší pohľad na systém PowerUpov v počítačových hrách, popis hierarchie tried a popis implementácie jednotlivých metód.

1 Zobrazovanie a interakcia 3D objektov umiestnených do 3D krajiny

Kapitola sa zaoberá problematikou 3D objektov vo všeobecnosti. Rozoberá techniky ukladania objektov v pamäti, ich následné zobrazovanie a zobrazovanie objektov v 3D scéne a s tým spojené problémy. Kapitola poukazuje na problémy vznikajúce pri zobrazovaní rozsiahlych 3D scén a venuje sa rozoberaniu možných riešení daných problémov. V ďalších podkapitolách sa čitateľ oboznámi s problematikou interakcie takýchto objektov v 3D krajine, načrtnutý bude taktiež problém s umiestňovaním objektov do 3D krajiny so všetkými charakteristickými črtami reálnej krajiny. Záver je venovaný predovšetkým podrobnému popisu zobrazovania 3D objektov v systémoch pracujúcich v reálnom čase – ako vhodný príklad je v dokumente uvedený zobrazovací stroj používaný v 3D počítačových hrách.

1.1 Zobrazovanie 3D objektov

1.1.1 Reprezentácia 3D objektov

Objekty nemožno zobrazovať v troch dimenziách ak nie sú vhodne reprezentované, čo v našom prípade znamená vhodne uložené v pamäti. Kritériá na reprezentáciu 3D objektov zahŕňajú predovšetkým úplnosť a presnosť. Postupnosť bytov, ktorú chceme používať na zobrazovanie objektu, teda musí popisovať modelovaný objekt úplne (musia byť obsiahnuté všetky jeho časti) a presne. Z formálneho hľadiska je veľmi dôležité aby bola reprezentácia objektu vzhľadom na operácie konzistentná, teda aby nedochádzalo k neželaným deformáciám objektov spôsobenými nevhodnou reprezentáciou. Z formálneho hľadiska je taktiež veľmi dôležitá regulárnosť reprezentácie, pretože pri zobrazovaní objektov môže dôjsť k problémom a nežiaducim efektom, ak sa pokúsime do scény umiestniť a následne zobraziť nereálny objekt.

Keďže sa tento text vo veľkej miere venuje zobrazovaniu trojrozmerných objektov v reálnom čase, je pre nás jedným z najdôležitejších kritérií efektívnosť, čo chápeme v tomto prípade najmä ako možnosť čo najrýchlejšie zobraziť daný objekt. 3D scény, najmä pokiaľ ide o simuláciu krajiny z reálneho sveta, obsahujú množstvá objektov. Z tohto dôvodu stojí v pomyseľnom rebríčku kritérií hodnotenia reprezentácie 3D objektov za kompaktnosť, teda náročnosť na pamäť, hneď za efektívnosťou.

Možností ako reprezentovať 3D objekt je veľmi veľa a ich počet výrazne presahuje rámec tejto publikácie. V nasledujúcich riadkoch sa teda budú rozoberať modely, ktoré boli (alebo sú) využívané v praxi.

1.1.1.1 Konštruktívna geometria

Jedným zo základných spôsobov reprezentácie 3D objektov je použitie konštruktívnej geometrie. Pri použití CSG (constructive solid geometry) je objekt reprezentovaný množinou primitívnych objektov (kužeľ, guľa, kocka, valec atď) a postupnosťou booleovských operátorov (zjednotenie, rozdiel, prienik), ktoré určia tvar telesa. Teleso reprezentované takýmto spôsobom je v pamäti uložené pomocou stromu. Výhodou takejto reprezentácie je možnosť parametrického modelovania – parametrizácie operácií v strome. CSG sa úspešne používajú v CAD systémoch pre modelovanie. CSG modely sa nevyužívajú až tak často, no napriek tomu ich možno nájsť v niektorých populárnych systémoch pre generovanie scény v reálnom čase, ako napríklad Unreal Engine alebo Source Engine. Aj v týchto systémoch je CSG reprezentácia používaná najmä na jednoduchšie objekty, pretože pre náročné objekty (ľudské telo) sa využíva skôr reprezentácia pomocou polygónov.

1.1.1.2 Dekompozičný model

Pri použití dekompozičného modelu sa objekt rozloží na elementárne objemové jednotky – kocky. Ide o diskrétny popis modelu, pri ktorom sa ako základná jednotka popisu používa Voxel, čo je objemová jednotka reprezentujúca hodnotu bunky v regulárnej sieti trojrozmerného priestoru. Dekompozičné modely je možné použiť pri zobrazovaní 3D objektov a rozsiahlych 3D scén, čo bolo využité napríklad v programe Delta Force od Novalogic. Ide teda o zaujímavý a použiteľný model 3D objektov, avšak v praxi málo využívaný z dôvodu nekompatibility so štandardnými grafickými kartami, ktoré sú primárne určene na zobrazovanie 3D objektov a scén.

1.1.1.3 Hraničný spline model

Popis objektov reprezentovaných pomocou hraničného spline modelu sa skladá s bodov, kriviek a spline plôch. Takéto modely nesú úplnú informáciu pre popis objektu. Presnosť reprezentácie objektu možno regulovať presnosťou aproximácie spline plôch. Takto reprezentované objekty bývajú veľmi reálne a pamäťovo nenáročné. Ich použitie scénach zobrazovaných v reálnom čase však nie je veľmi časté, pretože ich zobrazovanie je výpočetne veľmi náročné, keďže takýto objekt treba pred zobrazením transformovať na štandardný polygónový model. Istou nevýhodou takýchto modelov môže byť aj nutnosť neustále strážiť a kontrolovať regulárnosť (uzavretosť) modelu. Táto modelová reprezentácia sa využíva pri veľmi presnom geometrickom modelovaní v systémoch CAD/CAM.

1.1.1.4 Polygonálny model

Reprezentácia 3D objektu pomocou polygonálneho modelu využíva ako základ vertex, čo je bod v trojrozmernom priestore. Spojením dvoch vertexov dostávame úsečku, pridaním ďalšieho bodu získame trojuholník, čo je najzákladnejší polygón. V zásade sa využívajú hlavne trojhranné polygóny, takže najčastejším útvarom polygonálneho modelu je trojuholník. Záleží však na systéme, ktorý objekt vytvára, ukladá a transformuje. Polygonálny model je najpoužívanejší, pretože je priamo podporovaný hardwarom a tým pádom najrýchlejšie zobraziteľný. Ak grafický systém využíva pre uchovávanie modelu akýkoľvek spôsob, v konečnom dôsledku sa daný model vždy transformuje na polygonálny model. Tento spôsob uchovávania 3D objektov využíva aj náš referenčný program TankGame (viz. TODO) a pracuje s ním aj systém PowerUpov implementovaný do programu TankGame (viz. TODO), preto sa ďalej pod pojmom model bude myslieť polygonálny model.

1.1.2 Renderovanie 3D objektov

Fáza prechodu z modelu do obrázku zobraziteľného na monitore počítača sa nazýva renderovanie (rendering), teda spracovanie 3D objektov do takej podoby aby boli zobraziteľné na monitore. Ide o proces transformácie sady čísiel, ktoré udávajú tvar objektu v trojrozmernom priestore na sadu čísiel určujúcich, ktorý bod na monitore bude svietiť akou farbou. Pri tomto procese treba riešiť niekoľko problémov súvisiacich so zobrazením trojrozmerného sveta do dvojrozmerného. V prvom rade treba z modelu objektu vybrať len tie jeho časti, ktoré je z daného pohľadu na scénu vidieť. Nemá zmysel renderovať objekty, ktoré sa potom na obraze nevyskytnú, tým by sa proces zobrazovania spomalil, čo je pri systémoch pracujúcich v reálnom čase zásadné. Tým vyvstávajú problémy ako detekcia odvrátených strán a orezávanie objektov.

1.1.2.1 Odvrátené strany objektov

Sú to časti objektu, ktoré sú vzhľadom na pozorovateľa scény (kameru) odvrátené, teda ich nemožno pozorovať. Problém s odvrátenými stranami objektov sa rieši pomocou normálových vektorov. Normálový vektor vektoru A je vektor kolmý na vektor A. V prípade priestorových objektov reprezentovaných pomocou polygónov sa počítajú normálové vektory polygónov a pomocou nich sa neskôr zistí smer natočenia daného polygónu a tým pádom aj vzťah vzhľadom k pozorovateľovi objektu.

1.1.2.2 Orezávanie (clipping)

Orezávanie 3D objektov môže prebiehať pred samotným renderovaním, čo je náročnejšie na programovanie, ale v konečnom dôsledku efektívnejšie. Pri orezávaní pred samotným renderovaním treba spomedzi všetkých polygónov vybrať tie, ktoré sa nachádzajú v pohľade pozorovateľa scény. Pohľad má väčšinou tvar zrezaného ihlana alebo tvar kvádra, je to objem scény, ktorý sa bude vykresľovať. V prípade, že sa jedná o zrezaný ihlan hovoríme o perspektívnej projekcii, v prípade

kvádra o paralelnej projekcii. TODO (algoritmy na orezávanie v 3D). Druhý prístup k orezávaniu trojrozmerných objektov je vykonať orezávanie až po spracovaní objektov a následnej rasterizácii. Potom čo sa objekty transformujú, sa na výsledný obraz použijú algoritmy pre orezávanie 2D plôch, ako napríklad algoritmy Sutherland – Hodgman alebo Weiler – Atherton.

1.2 Zobrazovanie 3D scény

Pod pojmom 3D scéna rozumieme množinu trojrozmerných objektov, ktoré sa môžu navzájom ovplyvňovať. Objekty môžu ovplyvňovať svoju viditeľnosť, môžu na seba vrhať tiene a v prípade lesklého materiálu aj odlesky. Na riešenie týchto problémov existuje niekoľko techník, ktoré budú stručne načrtnuté na nasledujúcich riadkoch.

1.2.1 Kamera

Prvým a najzásadnejším problémom pri zobrazovaní scény je nastavenie kamery, teda vhodného pohľadu na scénu. Bez pohľadu sa totižto nedá počítať transformácia z trojrozmernej reprezentácie na dvojrozmernú, pretože tento výpočet sa vždy vzťahuje na nejakého pozorovateľa. Na začiatku je však potrebné definovať základné pojmy.

- *Premietací lúč (projection beam)* – je polpriamka vychádzajúca z premietaného (priestorového) bodu, ktorej smer závisí na zvolenej premietacej medóde.
- *Premietacia plocha (viewing plane)* – je plocha v priestore, na ktorú dopadajú premietacie lúče a v mieste dopadu vytvárajú priemet (obraz v premietacej ploche).

V ďalšom texte sa obmedzíme len na premietanie do rovinnej premietacej plochy. Rovinné premietanie delíme na dva základné skupiny – rovnobežné (paralelné) a stredové (perspektívne). Rovnobežné premietanie je charakteristické *smerom premietania*, stredové premietanie je charakterizované *stredom premietania*.

Základom úlohy premietania je formulácia geometrickej situácie, teda určenia miesta, kde stojí pozorovateľ, vymedzením pozície a orientácie premietacej plochy a stanovenie smeru a cieľu pozorovania. Kamera totižto zhotovuje snímky na premietaciu plochu, ktorá je kolmá na hlavnú optickú os kamery. Ak máme daný bod umiestnenia kamery [$kamera_x$, $kamera_y$, $kamera_z$, 1] a zvolený bod pozorovania [$cieľ_x$, $cieľ_y$, $cieľ_z$, 1], smer pozorovania $\vec{L}$ = (L_x, L_y, L_z, 0) jednoducho vypočítame ako:

$$\begin{bmatrix} L_x \\ L_y \\ L_z \\ 0 \end{bmatrix} = \begin{bmatrix} ciel'_x \\ ciel'_y \\ ciel'_z \\ 1 \end{bmatrix} - \begin{bmatrix} kamera_x \\ kamera_y \\ kamera_z \\ 1 \end{bmatrix}$$

Po normalizácii $\vec{L}$ dostaneme jednotkový vektor $\vec{l}$. Orientáciu kamery (natočenie okolo optickej osy) určuje vektor $\vec{u}$, o ktorom predpokladáme, že je kolmý na $\vec{l}$ (rovnobežný s premietacou plochou). Ďalším vektorom určujúcim nastavenie kamery je vektor $\vec{p}$, ktorý ukazuje doprava v smere osy x súradnicového systému premietacej plochy. Poslednou vlastnosťou kamery je vektor $\vec{h}$ určujúci otočenie okolo osy y súradnicového systému premietacej plochy. Orientáciu pohľadu teda možno nastaviť pomocou troch vzájomne kolmých jednotkových vektorov $\vec{l}, \vec{h}, \vec{p}$.

1.2.2 Viditeľnosť

Cieľom algoritmov pre riešenie viditeľnosti je nájdenie tých objektov a ich častí, ktoré sú viditeľné z určitého miesta (pozícia kamery). Tieto algoritmy sú vždy zviazané s konkrétnou reprezentáciou priestorových objektov. Dobre sa spracovávajú objekty popísané hraničnou reprezentáciou, hlavne teda polygonálne modely. Drvivá väčšina algoritmov pracuje práve s polygonálnou reprezentáciou objektov. V prípade, že je objekt reprezentovaný nejakým alternatívnym spôsobom, pre použitie algoritmov riešiacich viditeľnosť sa prevádza do polygonálneho modelu. Možná strata dát, je vyvážená rýchlosťou zobrazovania, ktorá je pre náš problém grafických systémov pracujúcich v reálnom čase najdôležitejšie.

V počítačových hrách a simuláciách sa na riešenie viditeľnosti scény používajú dva základné princípy. Prvým z nich je Z-buffering, teda spracovanie pomocou hĺbkového bufferu. Jedná sa o rastrový algoritmus riešenia viditeľnosti, ktorý je rýchly, ale náročný na pamäť. Hĺbkový buffer obsahuje Z súradnice najbližších bodov plôch, ich farba je v pamäti obrazu. Pri tomto algoritme sa každá plocha spracováva iba raz, Z-buffering je ľahko paralelizovateľný a jednoducho hardwarovo realizovateľný. Variácia na Z-buffering, ktorej výsledky sú rovnomernejšie sa nazýva W-buffering.

Ďalším rastrovým algoritmom pre riešenie viditeľnosti je ray-casting, teda vrhanie lúčov. Jeho podstata spočíva vo vrhaní lúčov z premietacej plochy do scény a detekcie kolízie vrhaných lúčov s objektmi v trojrozmernej scéne. Po detekcie kolízie s objektom sa zapíše priesečník a vo výsledku sa zisťuje najbližší priesečník k pozorovateľovi a ten sa potom vykreslí na obrazovku. Tento algoritmus je v porovnaní so Z-bufferingom (alebo W-bufferingom) pomalší, no dáva lepšie

výsledky. Jeho realizácia v hardwary je náročnejšia, no medzi jeho ďalšie výhody patrí jednoduchá realizácia vizuálnych efektov.

1.2.3 Osvetlenie a tieňovanie

Aby scéna vyzerala realistickejšie musí systém pre zobrazovanie trojrozmerných objektov simulovať osvetlenie a tieňovanie objektov. Riešenie osvetlenie objektov je modelovanie toho ako objekty odrážajú okolité svetlo. To samozrejme záleží najmä od optických vlastností daného povrchu. Medzi základné osvetľovacie modely patria Lambertov osvetlovací model, Phongov osvetlovací model a BRDF. Lambertov osvetľovací model počíta len s difúznym odrazom a ide o empirický osvetľovací model. Pri ideálnej difúzii sa svetlo odráža konštantne do všetkých smerov. Pri Lambertovom osvetľovacom modeli závisí intenzita difúzie na uhle dopadu svetelného lúča. Ďalší v rade, teda Phongov osvetlovací model, je ďalší z rady empirických osvetľovacích modelov. Ide vlastne o obohatenie Lambertovho modelu o reflexnú zložku, teda o odrazenie svetla. Ideálna reflexia svetla je v smere odrazu symetricky podľa normály. Phongov osvetľovací model teda pracuje z ďalšími premennými ako reflexná vlastnosť povrchu, ambientné svetlo, či ostrosť odlesku. Posledný menovaný so zástupu osvetľovacích modelov je BRDF, ktorý pracuje na základe fyzikálnych zákonov. Využíva sa pri fotorealistickom renderovaní a pri špeciálnych efektoch, je využitie v grafických systémoch pracujúcich v reálnom čase (počítačové hry a simulácie) je v globálnom nasadení nemysliteľný, keďže fyzikálne výpočty sú pomerne náročné.

Tieňovanie objektov môže razantne zvýšiť realističnosť zobrazovania scény objektov. Po aplikácii tieňovania na objekt s nízkym počtom polygónov dostaneme lepší výsledný efekt ako pri zvýšení počtu polygónov. Pričom spracovanie väčšieho počtu polygónov je náročnejšie ako aplikovanie jednoduchého tieňovacieho algoritmu. Medzi dva základné druhy tieňovania objektov v scéne patria Gourandovo tieňovania a Phongove tieňovanie. Gourandova metóda je relatívne rýchla, ktorá počíta tieňovanie objektu z vrcholov plôch. Je však potrebné poznať normálové vektory vo vrcholoch, tie sa však dajú jednoducho získať priemerom normálových vektorov okolitých plôch. Farby vo vnútri plôch sa potom lineárne interpolujú z hodnôt vo vrcholoch. Metóda berie do úvahy zakrivenie objektu a je dostatočne realistická pre bežné aplikácie. Phongove tieňovanie je komplexnejšia a omnoho náročnejšie na výpočet, avšak poskytuje oveľa reálnejšie výsledky. Farba sa pri tejto metóde počíta z osvetľovacieho modelu v každom bode plochy. Tento spôsob tieňovania objektov je založený na interpolácii normálových vektorov, nie na interpolácii farby, tak ako tomu bolo v predchádzajúcom prípade. V systémoch pracujúcich v reálnom čase sa Phongovo tieňovanie využíva len zriedkakedy.

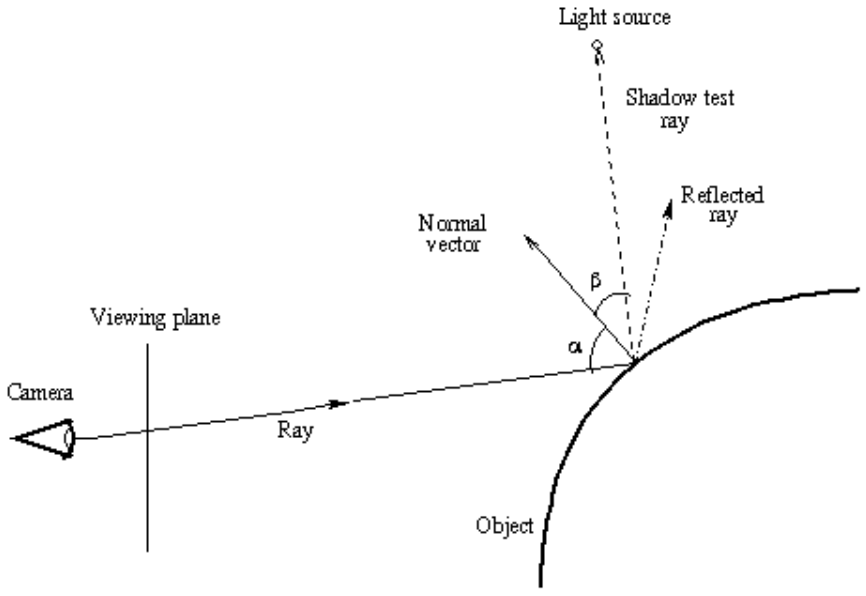
1.2.4 Radiozita a Ray tracing

Radiosita a Ray tracing patria medzi najpokročilejšie spôsoby vizualizácie trojrozmerných scén. Po použití techník radiosity a ray tracingu možno dosiahnuť až takmer fotorealistických výsledkov. V minulosti boli tieto techniky používane skôr na predrenderované vizualizácie, no v súčasnosti sa vyžívajú aj v systémoch pracujúcich v reálnom čase, ako napríklad v počítačovej hre Quake 4 od ID Soft.

1.2.4.1 Ray tracing

Ide o globálnu realistickú obrazovú metódu, ktorá sleduje priechod svetelného lúču scénou. Lúče sa šíria so svetelných zdrojov do scény, pričom sa berú do úvahy parametre svetla ako smer, farba a intenzita. Niektoré lúče zasiahnu objekty v scéne a podľa ich optických vlastností sa lomia, rozptyľujú a odrážajú. Kompletné smerovanie lúčov od zdroja svetla je však veľmi náročné, preto sa v praxi používa opačný prístup a síce back ray tracing, teda spätné sledovanie lúčov. Projekčné lúče sa v tomto prípade vysielajú cez pixely obrazu scény. Hľadá sa čo je v danom pixely vidieť a akú svetelnú energiu lúč prináša. Kvalita ray tracingu (a back ray tracingu) závisí na počtu odrazov lúča, ktoré sledujeme.

Príklad fungovania algoritmu ray tracingu:



1.2.4.2 Radiozita

Radiozita je globálna osvetlovacia metóda. Ide o aplikáciu metódy konečných prvkov na vyriešenie rovnice renderovania (opisuje prúdenie svetla scénou) pre scény s čisto difúznymi povrchmi. V praxi sa pri použití radiozity scéna rozdelí na niekoľko malých plôch, pre ktoré sa potom počíta osvetlenie. Pre každý priechod algoritmu sa vypočítava celkové množstvo svetla prichádzajúce na danú plochu zo všetkých ostatných plôch v scéne. Určité množstvo svetla sa pohltí, zvyšok sa považuje za

odrazený späť do scény a využije sa pri ďalšom priechode algoritmu. Náročnosť radiozity je kvadratická, čo z nej robí prakticky nepoužiteľnú metódu pre systémy pracujúce v reálnom čase. Základný algoritmus radiozity sa však dá prispôsobiť, tak aby je náročnosť klesla. Dá sa totižto prepísať ako problém renderovania textúrami mapovanej scény. Použitím BSP stromu môže dokonca masívne znížiť náročnosť pri lokalizovaní plôch, ktoré sú od ostatných kompletne izolované.

1.3 Grafické stroje pracujúce v reálnom čase

Grafické systémy zobrazujúce scény zložené z trojrozmerných objektov sú veľmi citlivé na výpočtový čas. Program, ktorý má na starosti spracovanie modelových reprezentácií do dvojrozmerného pola bodov na monitore, sa bude v nasledujúcom texte označovať ako stroj, alebo engine (zaužívaný anglický názov). Typickým príkladom takéhoto stroja herný engine v 3D počítačových hrách, ktoré na tomto poli dosiahli najvyššiu úroveň. Na nasledujúcich riadkoch bude popísaný tok dát v rúre (pipeline) 3D enginu. 3D pipeline sa dá rozdeliť na niekoľko častí. Nasleduje rozdelenie 3D pipeline a popis činností jednotlivých častí.

3D pipeline

1. Aplikácia / Scéna
    - Načítanie databázy scény / objektov
    - Umiestnenie objektov, smerovanie a umiestnenie kamery
    - Animovaný pohyb modelov objektov
    - Popis obsahu 3D scény
    - Kontrola viditeľnosti objektov
    - Výber úrovne detailov (LOD)
2. Geometria
    - Transformácie (posunutie, otočenie, zmena mierky)
    - Transformácia z modelového súradnicového systému na súradnicový systém scény
    - Transformácia zo súradnicového systému scény do súradnicového systému pohľadu (premietacej plochy)
    - Projekcia
    - Odstránenie plôch mimo zorný uhoľ (môže sa vykonať aj neskôr)
    - Odstránenie odvrátených plôch
    - Osvetlenie
    - Orezávanie
3. Renderovanie / Rasterizácia

- Tieňovanie
- Textúrovanie
- Hmla
- Test alfa priehľadnosti
- Vrhanie tieňov
- Test hĺbkového bufferu
- Antialiasing
- Zobrazenie

1.3.1 Aplikácia / Scéna

3D aplikácia sama o sebe môže byť považovaná za súčasť toku dát grafickým pipelinom, aj keď sama o sebe nie je súčasťou grafického pod systému. Avšak celý proces generovania obrazu začína. 3D objekty reprezentované polygonálnym modelom sú väčšinou vytvárané externými programami (3D Studio Max, Aya, Lightwave) a následne uložené do súboru. Prvotnou úlohou grafickej aplikácie je načítanie všetkých potrebných objektov, ktoré tvoria vybranú scénu. Aplikácia musí objekty umiestniť do scény a správne ich nastaviť. Napríklad implicitné nastavenie modelu humanoida je tzv. póza DaVinci, teda s upaženými rukami a roztiahnutými nohami. Aplikácia teda musí umiestniť končatiny postavy do správnej pozície. S tým je spojená aj animácia jednotlivých objektov a načítanie nasledujúceho stavu objektu v nasledujúcom frame.

Ďalšou dôležitou činnosťou hornej vrstvy celého systému je umiestnenie a správne nastavenie kamery. U kamery treba nastaviť jej pozíciu v scéne, bod na ktorý je zameraná (smerovanie kamery) a jej natočenie vzhľadom na jednotlivé osy. Všetko samozrejme záleží na type pohľadu na scénu. Či už z vlastných očí pozorovateľa, alebo z vtáčej perspektívy.

Po umiestnení kamery nastáva čas na odstránenie tých objektov zo zoznamu, ktoré nie sú v zornom uhle kamery. Na počiatočnú detekciu objektov mimo zorného uhľa sa používa veľmi jednoduchý ale efektívny systém s hraničným zaobalením objektu iným, jednoduchším telesom, napríklad kockou alebo guľou. Po nahradení komplexného modelu jednoduchým modelom kocky nemusí program kontrolovať niekoľko sto polygónov, ale iba 18 trojuholníkov kocky, čím sa rapídne ušetrí čas, ktorý sa môže využiť na iné výpočty.

Vo fáze prípravy scény na ďalšie spracovanie program vyberá podľa vzdialenosti od kamery vhodný model objektu. Každý objekt má uložených niekoľko variantov, ktoré sa navzájom líšia počtom polygónov. Jeden objekt býva zvyčajne uložený až v piatich rôznych úrovniach detailov. Je len logické, že vo veľkej vzdialenosti sa aplikácia nebude snažiť zobrazovať objekty s veľkým počtom polygónov, ktoré by užívateľ vôbec nespozoroval. Týmto sa rapídne zvýši výkon aplikácie a opäť sa ušetri niečo z procesorového času.

Jeden z trikov, ktorý sa pri ukladaní a načítavaní polygonálnych modelov využíva, je uchovanie zoznamu trojuholníkov v takzvaných Strip a Fan. Inak povedané v pásoch a vejároch. Miesto toho aby aplikácia ukladala a spracovávala jednotlivé trojuholníky s troma vrcholmi, spracuje sadu trojuholníkov uložených v páse. Takto sa eliminuje problém spracovávania rovnakých vrcholov, ktoré majú viaceré trojuholníky spoločné. Základom sú tri vrcholy prvého trojuholníka, pridaním ďalšieho trojuholníka do zoznamu sa pridá len jeden nový vrchol na spracovanie, čím sa zvýši výkon grafického stroja. Používanie vejárov a pásov je optimalizované v každej štandardnej grafickej karte a je to najlepší spôsob ako popísať objekt zložený z veľkého množstva polygónov.

1.3.2 Geometria

V tejto časti procesu spracovávania grafiky dochádza najprv ku geometrickým transformáciám. Geometrické transformácie sú jedným z najčastejších operácií nad vrcholmi trojuholníkov. Prvým krokom k akcelerácii grafiky, preto býva implementácia hardwarovej podpory týchto transformácií, čo je dnes už bežným štandardom. Objekty môžeme pohybovať a meniť použitím štyroch základných operácií – posunutie, zmena mierky, otočenie a skosenie. Tieto operácie sa na vrcholy jednotlivých trojuholníkov aplikujú použitím matíc so štandardnými pravidlami maticovej matematiky.

Posunutie: Ide o posunutie objektu pozdĺž niektorej z osí súradnicového systému. Normálne by sa posunutie realizovalo pričítaním alebo odčítaním (pričítaním záporného čísla), ale z dôvodu efektivity sa v 3D grafických strojoch na posunutie používa násobenie maticou.

Rotácia: Ako názov napovedá, objekt môžeme otočiť okolo ľubovoľnej osy súradnicového systému. V najzákladnejšom prípade, kedy je objekt posunutý do počiatku súradnicového systému sa operácia otočenia realizuje násobením sínusu alebo kosínusu každej súradnice uhlom otočenia (theta). V prípade, že sa objekt nenachádza v počiatku súradnicovej sústavy, treba ho tam najprv presunúť, potom aplikovať maticu otočenia a následne posunúť späť na jeho pozíciu. Ak potrebujeme otočiť objekt okolo viacerých osí, je dôležité poradie jednotlivých otočení.

Zmena mierky: používa sa na zmenu veľkosti modelov pri vytvorení ilúzie perspektívnej projekcie. Matematická operácia používaná pri zmene mierky je násobenie. Ak chcem napríklad zväčšiť objekt dvojnásobne, vynásobíme každú súradnicu číslom dva. Zmena mierky môže byť jednotná, teda, že všetky súradnice vynásobia rovnakým číslom, alebo sa každá jedna vynásobí iným. V prípade, že násobíme záporným číslom vytvárame zrkadlový obraz.

Zkosenie: ide o zmenu tvaru objektu vzhľadom na jednu z osí. Matematická operácia je pridanie funkcie jednej súradnice do druhej. Napríklad pridaním hodnoty x-ovej súradnice k y-ovej a ponechaní x-ovej v pôvodnej hodnote.

Transformácie sa dajú skladať do jednej matice. To znamená, že sa najprv vytvorí jedna matica zo všetkých požadovaných operácií, ktoré sa majú vykonať a až potom sa aplikuje na všetky vrcholy. Je to efektívnejší prístup ako aplikovať každú jednu operáciu postupne. Matica používaná pre

operácie nad trojrozmernými objektmi nemá veľkosť 3x3 ako by každého mohlo napadnúť, ale miesto toho používa homogenizované súradnice a výsledná matica má teda rozmery 4x4. Homogénne súradnice dovoľujú použiť určité transformácie, ktoré by inak vyžadovali sčítanie, ako napríklad posunutie, realizovať násobením.

Príklad matice zloženej z otočenia okolo osy X a operácie posunutia:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix}$$

Po vstupe do geometrickej časti spracovávania grafiky, treba previesť objekty z modelového súradnicového systému do súradnicového systému scény a potom do súradnicového systému pohľadu. Pri konverzii z jedného súradnicového systému na druhý sa využívajú práve maticové transformácie. Niektoré konverzie vyžadujú len jedinú transformáciu, iné zase kombináciu viacerých. Napríklad transformácia zo súradnicového systému scény do súradnicového systému pohľadu vyžaduje posunutie a rotáciu.

Časť geometrická v sebe zahŕňa aj procesy súvisiace s ušetrením práce, ktorá musí byť vykonaná v poslednej fáze. Prvý krok v šetrení práce je výber tých polygónov, ktoré nie sú v zornom uhle kamery. Tento proces sa nazýva triviálne odmietanie polygónov, pretože na rozdiel od orezávania ide o pomerne jednoduchú operáciu. Prebieha test či x, y a z súradnice všetkých vrcholov trojuholníka sú v zornom uhle. Ak sa aspoň jeden z bodov nachádza v ihlane pohľadu kamery, ide o trojuholník, ktorý je minimálne čiastočne viditeľný. Ak sa ostatné body nachádzajú mimo kameru, je nutné takýto trojuholník orezať, čím vznikne ne-trojuholníkový polygón, ktorý bude musieť byť neskôr rozdelený na trojuholníky.

Ďalšou operáciou je odmietnutie trojuholníkov, ktoré sú vzhľadom na pozorovateľa odvrátené. V skutočnosti býva viac ako polovica trojuholníkov ľubovoľného modelu odvrátená. Táto operácia sa v angličtine pomenováva ako back-face bulling. Jediným prípadom, kedy by sme chceli ponechať aj odvrátené trojuholníky, je ak pred nimi stoja priesvitné trojuholníky. Túto operáciu má však na starosti aplikačná časť, keďže algoritmus back-face culling priehľadnosť neberie do úvahy. Pri výbere odvrátených strán sa pracuje s normálovými vektormi polygónov. Princíp tohto algoritmu spočíva vo vysielaní priameho lúča zo stredu kamery. Následne sa zmeria uhol medzi týmto priamym vektorom a normálovým vektorom trojuholníka. Ak je uhol väčší ako 90 stupňov, jedná sa o odvrátenú stranu, ak je menší jedná sa o viditeľný trojuholník.

V tejto časti dochádza aj k výpočtu osvetlenia, využívajú sa niektoré z algoritmov popísané v kapitole 1.2.3. V tejto časti máme vybraté polygóny, ktoré sú v zornom uhle kamery a sú vzhľadom

k pozorovateľovi privrátené, sú na správnom mieste, majú správnu časť animácie a majú správne vypočítanú mieru osvetlenia. Na rad prichádza posledná časť geometrickej časti a tou je orezávanie polygónov, ktoré sú z časti v zornom uhle kamery a z časti mimo neho. Na orezávania sa používa jeden z algoritmov pomenovaných po svojich tvorcoch, ako napríklad: Liang-Barsky, Cyrus-Beck-Liang-Barsky, Nichol-Lee-Nichol a Sutherland-Hodgeman.

1.3.3 Renderovanie / Rasterizácia

Potom čo boli vykonané predchádzajúce operácie, prichádza na rad fáza vlastnej rasterizácie, teda vytvorenia dvojrozmerného poľa bodov. Ako prvé sa aplikuje tieňovanie. Využíva sa väčšinou jeden z algoritmov popísaných v časti 1.2.3.

Ďalšou úlohou tejto časti celého grafického systému je nanášanie textúr. Textúra predstavuje spôsob, ako priradiť povrchu objektu rôzne premenlivé vlastnosti. Umožňujú väčšiu realističnosť modelov a ich jednoduchšiu geometriu. Textúry môžeme vytvoriť pomocou obrázku alebo pomocou procedúry. S obrázkovými sa jednoduchšie pracuje a väčšinou majú lepší výsledný efekt, zatiaľ čo procedurálne sú menej náročné na pamäť. Pri použití textúry sú dôležité dva kroky, načítanie textúry a vlastné nanesenie na povrch objektu. Pri nanášaní textúry je dôležité nájsť vhodnú projekčnú funkciu. Ak je povrch popísaný matematickou funkciou, je možné textúru naniesť inverznou funkciou k tejto funkcii, ak samozrejme taká funkcia existuje. Mapovanie textúry ale väčšinou prebieha tapetovaním, teda nalepovaním na povrch objektu. Textúr rozoznávame niekoľko druhov: odlesková textúra, svetelná textúra, tieňová textúra, bump mapa alebo enviroment textúra.

Dôležitým problémom pri používaní textúr je ich deformácia pri perspektívnej projekcii. Je to spôsobené pohľadovou transformáciou vrcholov polygónov, z ktorých sa lineárnou aproximáciou vypočítavajú súradnice textúr až v priestore zobrazenia. Riešením je mapovanie textúry v priestore objektu a transformace každého rasterizovaného bodu objektu zvlášť. Ako už bolo v tomto texte viac krát spomenuté, pri systémoch pracujúcich v reálnom čase je dôležitá rýchlosť. Urýchlenie zobrazovania textúrovaných objektov umožňuje MIP-mapping. Použitie MIP-mappingu taktiež odstraňuje aliasing textúr. Princíp MIP-mappingu spočíva v uložení niekoľkých obrazov z rôznym rozlíšením. Pričom RGB obrazy sú ukladané po zložkách. Pre zvýšenie kvality textúr sa v dnešnej dobe štandardne používa taktiež anizotropické filtrovanie. Anizotropné filtrovanie odstraňuje rozmazávanie textúr zobrazovaných vo veľkej diaľke a pod veľkým uhlom. Princíp anizortropického filtrovania spočíva v projekcii do tvaru lichobežníka miesto štvorca.

Po nanesení textúr sa presúvame k ďalšej pixelovej operácii a to k nanášaní hmly. Jednou z vecí, ktoré sa snaží väčšina systémov pracujúcich v reálnom čase dosiahnuť, je navodenie správnej atmosféry. Väčšinou sa predsa využívajú v zábavnom priemysle. Na to je ako stvorená práve hmla. Dva základné typy hmly sú lineárna a exponenciálna hmla. Hmlu možno nanášať po pixeloch alebo po vrcholoch. Samozrejme, že nanášanie po pixeloch poskytuje lepšie výsledky čo sa týka kvality, no

stojí o niečo viac výpočtového času. Ďalšie dva pokročilejšie spôsoby nanášania hmly sú hmla založená na rozsahu (range-based fog), ktorá sa stala možnosťou s príchodom vertex-shaderov v DirectX8. Ďalšia možnosť sa nazýva hmlová tabuľka, kde sú hodnoty hmly uložené vo vyhľadávacej tabuľke, odkiaľ sa nanášajú na každý pixel. Jednou z najdôležitejších úloh hmly je, že dovoľuje objektom v diaľke postupne zmiznúť, miesto toho aby len zmizli zo snímku na snímok.

V procese rasterizácie sa ďalej testujú alfa hodnoty. Pomocou nich je možné vytvárať efekty vody, skla, priehľadných a priesvitných objektov. Alfa hodnota býva väčšinou uložená vo vrcholoch využívajúc formát RGBA. Pri štandardných 32 bitoch vyjde na každú zložku 8 bitov, pričom maximálna hodnota alfa je 1 a minimálna 0, pričom 1 znamená kompletne nepriehľadné a 0 absolútne priehľadné. Na rozdiel od nepriehľadných objektov, musia tie priehľadné byť hĺbkovo testované odzadu dopredu, aby bolo možné zistiť farbu presvitajúceho svetla. Základný vzorec pre výpočet je:

co = a x cs + (1 - a) x cd (RTR, p. 86)

co = farba výsledného pixelu
a = alfa hodnota (medzi 0 a 1)
cs = farba priesvitného pixelu (nazývaného zdroj)
cd = farba uzavretého pixelu (nazývaného cieľ)

Ďalším bežným použitím alfa úrovne je niečo čo nazývame alfa mapovanie, ktoré kombinuje alfa hodnotu s textúrou. Používa sa najmä na vytváranie textúr a odtlačkov zvláštnych tvarov, keďže väčšina textúr je buď obdĺžniková alebo štvorcová.

Tiene sú ďalším z prvkov, ktoré pomáhajú zvýšiť pocit reálnosti v 3D scéne. Najprv musí existovať zdroj, buď pracujeme s pixelmi alebo s vrcholmi. Ďalej je potrebné poznať zdroj svetla a potom objekt, ktorý tomuto svetlu stojí v ceste, nazývaný *clona*. Nakoniec je v scéne povrch, na ktorom sa tieň zobrazí, nazývaný prijímač. Tiene samy o sebe majú dve časti, prvá sa nazýva *umbra*, čo je vnútorná časť tieňa a *penumbra*, čo je vonkajšia a hraničná časť tieňa. Penumbra vytvára rozdiel medzi tvrdými a mäkkými tieňmi, pretože v prípade tvrdých tieňov okraj tieňa končí náhle a neexistuje žiadny prirodzený prechod. Pri mäkkých tieňoch farba okraju prechádza z farby tieňa (väčšinou čierna) do farby priľahlých pixelov.

K riešeniu viditeľnosti dochádza pomocou hĺbkového bufferu, alebo inak z-bufferu, ktorého princípy sú popísané v kapitole 1.2.2.

Pred samotným zobrazením na monitore dochádza ešte k posledným úpravám obrazu v podobe odstraňovania aliasu. Alias je jav, ktorý vzniká nízkofrekvenčným vzorkovaním signálu s relatívne vysokou frekvenciou. V prostredí 3D počítačovej grafiky sa alias chápe ako zubaté hrany pri rastrovom zobrazení vektorovej grafiky. Dva prístupy k riešeniu odstraňovania aliasu sú Ordered Grid Supersampling (OGSS) a Rotated Grid Supersampling (RGSS). OGSS zoberie scénu rozvzorkuje ju na vyššie rozlíšenie, zoberie nejaké množstvo pod-vzorkov a vypočíta hodnotu pixelu spriemerovaním hodnôt týchto pod-vzorkov. OGSS môžete byť aplikované v niekoľkých stupňoch:

- 2X: 3D scéna je rozvorkovaná 2 krát len v smere jednej osy. Takže pre rozlíšenie 800 x 600 sa bude pracovať s 800 x 1200 a len 2 pod-vzorky sa použijú na výpočet výslednej farby.
- 4X: 3D scéna je rozvorkovaná 2 krát na oboch osách. Takže pre rozlíšenie 800 x 600 sa bude pracovať s 1600 x 1200 a použijú sa 4 pod-vzorky na výpočet výslednej farby.
- 9X: 3D scéna je rozvorkovaná 3 krát na oboch osách. Takže pre rozlíšenie 800 x 600 sa bude pracovať s 2400 x 1800 a použije sa 9 pod-vzorkov na výpočet výslednej farby.
- 16X: 3D scéna je rozvorkovaná 4 krát na oboch osách. Takže pre rozlíšenie 800 x 600 sa bude pracovať s 3200 x 2400 a použije sa 16 pod-vzorkov na výpočet výslednej farby

RGSS, ktorý bol kedysi využívaný firmou 3dfx je trochu podbný OGSS, ale miesto rozvzorkovania obrazu na vyššie rozlíšenie sa vytvoria dve alebo štyri kópie obrazu, potom sa zoberie geometria scény a posunie sa, alebo skôr roztrasie sa do dvoch, či štyroch smerov (v závislosti na počte kópií). Pixely z týchto roztrasených kópií sa potom spriemerujú aby sa tak vytvorila výsledná hodnota pixelu.

Na konci celého procesu dôjde to čo bola len hŕba vrcholov sa pretransformovalo na trojrozmernú scénu, ktorú môžeme vidieť pred sebou, ktorá ak grafický stroj pracuje s rýchlosťou 60 snímkou za sekundu, vydrží na obrazovke približne 17 milisekúnd.

2 TankGame

Súčasťou zadania bakalárskej práce je študovanie konkrétnej implementácie grafického systému pracujúceho v reálnom čase. Týmto konkrétnym systémom je program TankGame, ktorý vznikol na Ústave Počítačovej Grafiky a Multimédií. Jedná sa o jednoduchý grafický engine, schopný zobrazovať 3D ak 2D objekty a schopný generovať nekonečnú krajinu. V tomto grafickom engine sú ďalej implementované základy tankovej počítačovej hry, ktorá má ukázať jeho možnosti.

2.1 Objekty

Program TankGame pracuje s polygonálnou reprezentáciou modelov objektov. Na ich uloženie využíva dva typy formátov súborov a to VRML a ASE súbory. Ďalšou možnosťou je použiť základné primitíva, ako kocka, ktoré sú definované v bázovej triede pre modely s názvom TModel. Načítanie VRML súborov má na starosti trieda TModelVRML, ktorá uchováva načítaný súbor v triede TVRMLFile. Program dokáže VRML súbor rozparsovať a renderovať, takže práca s takýmito modelmi je jednoduchá. Jediným problémom použitia VRML modelov v programe TankGame je limitácia VRML parseru len na určitú časť syntaxe a sémantiky štandardných VRML súborov. Napríklad nie je možné používať príkazy pre základné primitíva a taktiež boli objavené prípady, kedy celý program padal, v prípade, že VRML súbor neobsahoval definovanú textúru. Modely načítané zo súboru vo formáte ASE sú uložené v triede TModelASE, ktorá je tak isto ako trieda TModelVRML podedená od triedy popisujúcej modely všeobecne, s názvom TModel. Podobne ako trieda TModelVRML pracuje aj trieda TModelASE s triedou reprezentujúcou súbor v danom formáte s názvom TASEFile. Na nasledujúcich riadkoch budú stručne popísané obidva používané formáty.

2.1.1 Formát VRML

VRML (Virtual Reality Modeling Language) je štandardným súborovým formátom pre popis trojdimenzionálnej interaktívnej vektorovej grafiky. VRML súbor je vlastne textový súbor, v ktorom je možné definovať vrcholy a hrany trojrozmerných objektov spolu s povrchovou farbou, textúrou, lesklosťou, priesvitnosťou, atď. Výhodou VRML formátu je jeho nezávislosť na platforme, takže je 18 možné objekt popísaný vo VRML súbore renderovať na rôznych systémoch ako MS Windows, Macintosh a Unix. Tento formát bol pôvodne zameraný na webové aplikácie, ale vďaka svojím výhodám sa často používa aj iných aplikáciách. Inou výhodou tohto formátu je jeho rozšíriteľnosť. Rozšírenia do VRML totižto používajú objekty, ktoré majú schopnosť popisovať samé seba. Objekty, ktoré nie sú súčasťou štandardu VRML môžu poskytnúť popis, ktorý interpret môže spracovať podľa potreby. Pomocou VRML súborov je možné pridávať vyššiu detailnosť len do určitej časti objektu.

V súbore tohto formátu môžu byť okrem samostatných objektov uložené aj celé scény, obsahujúce niekoľko rôznych objektov.

Príklad jednoduchého VRML súboru:

```
#VRML V2.0 utf8
# Simple cone
Shape {
        appearance Appearance {
                material Material {
                }
                texture ImageTexture {
                }
        }
        geometry Cone {
        }
}
```

2.1.2 Formát ASE

ASE je skratka od Ascii Scene Export, z čoho vyplýva, že ide o súbor obsahujúci ascii znaky, ktorými je daný objekt popísaný. Súbory vo formáte ASE sú natívne pre program 3D Studio Max, čo je najpoužívanejší nástroj na modelovanie trojrozmerných objektov. Exportovať súbory v tomto formáte však dokážu aj iné programy. Medzi výhody formátu ASE patrí jeho veľká rozšírenosť. Valná väčšina dnešných grafických enginov využíva práve tento formát, medzi ne patria aj veľmi známe grafické stroje ako Unreal Engine, či Doom3 Engine. ASE súbor sa v zásade vytvára exportovaním z nejakého grafického štúdia (spravidla 3D Studio Max), jeho syntax je pomerne komplexná, preto na rozdiel od VRML súborov sa len zriedkakedy upravujú ručne, teda prostredníctvom textového 19 editoru. Dokáže však popísať scénu oveľa konkrétnejšie, dajú sa v ňom ukladať aj transformácie animácie daného objektu.

2.2 Renderovanie scény

Program TankGame využíva na renderovanie objektov a výpočty súvisiace s grafikou, knižnicu OpenGL, ktorá zaobaľuje množstvo z nutných činností (popísané v časti 1.3), ktoré sa pri zobrazovaní 3D scény musia vykonávať.

2.2.1 OpenGL

OpenGL (Open Graphics Library) je štandardná špecifikácia definujúca rozhranie pre grafické aplikácie (2D aj 3D), ktoré je nezávislé na platforme. Interface pozostáva z viacej než 250 funkcií, ktoré môžu byť použite na vykresľovanie komplexných trojrozmerných scén zo základných primitív. OpenGL bol vyvinutý firmou Silicon Graphics Inc. v roku 1992. Využíva sa v priemysle počítačových hier, ako aj v CAD systémoch a virtuálnej realite. OpenGL má dve hlavné úlohy, skryť komplexnosť komunikácie s grafickým hardwarom a skryť rôzne možnosti tohto hardwaru, tak aby mohol programátor pracovať s jednotným rozhraním, bez nutnosti zaujímať sa o konkrétne špecifikácie hardwaru.

Rozhranie OpenGL prijíma údaje o bodoch, úsečkách a polygónoch a transformuje ich na pixely, inak povedané vykonáva rasterizáciu. Väčšina príkazov OpenGL buď predáva grafické primitíva do grafického pipelinu, alebo nastavuje spôsob akým budú tieto primitíva spracované. Jedná sa o nízko úrovňové procedurálne rozhranie, ktoré núti programátora exaktne definovať kroky potrebné k vykreslenie scény. To znamená, že programátor musí mať dobrý prehľad o fungovaní grafického pipelinu (časť 1.3), ale na druhej strane poskytuje oveľa viac voľnosti ako iné vysoko úrovňové rozhrania.

OpenGL vykonáva väčšinu z činností procesu vykresľovania scény, ktoré boli popísane ako 3D grafický pipeline v kapitole 1.3. Programátor sa preto nemusí zaoberať kódovaním algoritmov, ktoré majú na starosti osvetlovanie, tieňovanie alebo výber polygónov mimo kamery. Stačí len pomocou konfiguračných príkazov vybrať spôsob tieňovania objektu (OpenGL poskytuje všetky základne typy tieňovania) a o zvyšok sa postará OpenGL, čo značne uľahčuje programovanie aplikácií, ktoré zobrazujú objekty v trojrozmerných scénach. V poslednej dobe sa však objavuje trend, ktorý žiada nahradiť fixnú funkcionalitu za programovateľnosť v oblastiach, ktoré sa stali veľmi komplexnými (spracovanie vrcholov a fragmentov). Preto OpenGL poskytuje jazyk vystavaný na ANSI C nazvaný The OpenGL Shading Language, ktorý ponúka programátorovi grafiky možnosť presne definovať čo sa má s dátami v konkrétnej časti pipelinu stať. Nezávislé preložiteľné objekty napísané v tomto jazyku sa nazývajú shadery.

S OpenGL úzko súvisí aj GLUT, ktorý je programom TankGame využívaný. Je to skratka od OpenGL Utility Toolkit a jedná sa o knižnicu utilít pre OpenGL. Poskytuje rozhranie pre definovanie okna, kontrolovanie okna a monitorovanie vstupu a výstupu z klávesnice a myši. Obsahuje tiež funkcie pre vykreslenie niektorých základných grafických telies, ako napríklad: kocka, guľa alebo pri testovaní veľmi populárny Utahský (Newellov) čajník.

2.2.2 Funkcia Render

O renderovanie v programe TankGame sa starajú tri funkcie a to Render, Render3D a Render2D. Funkcia Render nerobí nič iné, len volá funkcie Render2D a Render3D, v prípade potreby ešte

vytvorí okno pre sekundárnu kameru, ktorá nasleduje zadaný cieľ. Funkcia Render3D robí presne to čo jej názov napovedá, teda vykresľuje trojrozmerné objekty. Parametrami funkcie Render3D sú kamera a úroveň detailov. Funkcia najprv nastavuje parametre renderovacieho systému OpenGL (typ tieňovania, osvetlenie atď.), potom vykreslí nekonečnú scénu. Po vykreslení scény prehľadáva zoznamy všetkých objektov a postupne ich vykresľuje do scény. Ide len o volanie metódy Render každého objektu, ktorý ma byť vykreslený. Renderovanie samotného objektu má teda na starosti jeho trieda. Funkcia Render2D má na starosti vykreslenie dvojrozmerných objektov, ktorými sú napríklad hlavné menu, vyskakovacie menu a výpis údajov v ľavej časti obrazovky.

Vykresľovanie jednotlivých objektov majú na starosti metódy príslušných tried, ako už bolo spomenuté. Väčšina z nich je približne rovnaká, preto bude na ilustráciu popísaná jedna z nich. Ide o funkciu Render triedy TTank, ktorá vykresľuje objekt tanku. Na začiatku funkcia ukladá do pamäti globálnu transformačnú maticu, tak aby nedošlo k jej zmene, v prípade, že by bola potrebná v nejakej inej časti kódu. Potom dochádza k nastaveniu súradnicového systému, tak aby sa objekt vykreslil na príslušné miesto. Po posunutí dochádza k vykresleniu jednotlivých častí tanku, pri ktorom je nutné nastaviť svetlo a materiál objektu. V prípade potreby sa ešte vykresľuje štít alebo značky.

Vďaka OpenGL sa pri zobrazovaní objektov netreba starať o nanášanie textúry, o odstraňovanie odvrátených polygónov, alebo o orezávanie. Stačí len vhodne definovať sadu polygónov vytvárajúcich daný objekt.

2.2.3 Kamera

Pri vykresľovaní scény je veľmi dôležitá kamera, pretože koniec koncov ona určuje, ktorá časť scény sa vlastne vykreslí a ktorá nie. V programe TankGame sú pre simuláciu kamery dve triedy TCamera a TFrustrum. Druhá menovaná je abstraktom ihlana pohľadu kamery, využíva sa priamo v triede TCamera, ktorá už zaobstaráva samotný pohľad na scénu. Od triedy TCamera sú podedené dva konkrétny typy kamery a to užívateľom kontrolovaná kamera a kamera, ktorá nasleduje zadaný cieľ.

3 Powerup

Na demonštráciu zobrazovania objektov a ich interakcie s ostatnými objektmi v scéne som sa rozhodol implementovať systém powerupov pre program TankGame. V nasledujúcich riadkoch bude popísaný princíp powerupov a bude vysvetlené ich postavenie v počítačových hrách. Ďalej sa bude táto časť bakalárskej práce venovať konkrétnej implementácie triedy TPowerUp, ako aj systému pre náhodné generovanie a rozmiestňovanie powerupov do scény. Na záver kapitoly budú vymenované všetky úpravy zdrojových kódov programu TankGame, ktoré boli pre funkcionalitu powerupov nevyhnutné.

3.1 Definícia powerupu

Termín powerup sa používa v terminológii herného priemyslu. Ide o akési vylepšenie, podľa názvu zvýšenie sily. Všeobecne sa však môže jednať o akékoľvek zvýšenie vlastnosti herného objektu. Väčšinou sa jedná o pričítanie určitého čísla k vlastnostiam nejakého objektu, ktorý s daným powerupom príde do kontaktu. Pričítať sa môže pochopiteľne aj záporné číslo, takže často krát powerupy naopak znižujú schopnosti nejakého herného objektu. powerupy bývajú veľmi často realizované pomocou objektov náhodne rozmiestnených v scéne. Hráč ma potom na výber či daný powerup zoberie alebo nie – či s ním príde do kontaktu a aktivuje jeho trigger.

Pri návrhu powerupov pre program TankGame bol kladený dôraz na zlepšenie hrateľnosti rozvíjajúcej sa hry. Z tohto dôvodu sa powerupy do scény TankGameu rozmiestňujú anonymne. V pravidelných intervaloch sa na náhodnom mieste v scéne umiestni náhodne vybraný powerup. Každý powerup má pritom rovnaký model, tak aby hráč nemal znalosti o tom aký trigger po kontakte s powerupom vyvolá. Po scéne sa teda po určitej dobe nahromadí niekoľko powerupov, z ktorých približne polovica má na hráčom ovládaný tank pozitívny účinok a polovica negatívny. Hráči sa teda v snahe otočiť nepriaznivý stav bitky môžu uchýliť k akejsi lotérii, ktorá im v prípade šťastia môže zaručiť víťazstvo. Do hry to pridáva istú dávku náhody a neurčitosti, čo sú aspekty, ktoré ak sú použité v správnej miere pomáhajú zlepšiť hrateľnosť počítačových hier.

3.2 Hierarchia tried powerupov

3.2.1 Bázová trieda

Hierarchia tried pre powerupy bola navrhnutá tak aby sa veľmi jednoducho dali doprogramovať ďalšie a ďalšie powerupy, podľa potreby a typu hry. Pre tento účel bola vytvorená bázová trieda TPowerUp, ktorá poskytuje základné rozhranie pre powerup. V triede TPowerUp je vlastnosť pre 22

uloženie pozície v scéne, ktorá je reprezentovaná pomocou objektu triedy TVector, ktorý sa v programe TankGame štandardne využíva na ukladanie pozície v scéne. Trieda ďalej obsahuje ukazateľ na bázovú triedu pre model, ktorý slúži na uloženie modelu powerupu. Keďže po kontakte sa powerupom sa väčšinou vyvolá nejaký efekt, pomocou ktorého si hráči môžu všimnúť, že došlo k spusteniu triggeru daného powerupu, je v triede TPowerUp deklarovaný ukazateľ na triedu TEffect. Inštancie tejto triedy sa v programe TankGame používajú na uchovávanie rôznych efektov. Poslednou vlastnosťou tried TPowerUp je premenná Active booleovského typu, ktorá podľa svojej hodnoty určuje či je powerup aktívny, alebo nie.

Rozhranie pre tvorbu nových powerupov predstavujú virtuálne funkcie Render a TimeTick. Funkcia Render slúži na vykreslenie objektu powerupu do scény. Túto funkciu si tvorca nového powerupu musí doprogramovať sám aj napriek tomu, že všetky powerupy v hre TankGame mali mať pôvodne rovnaký model. Trieda bola takto riešená z dôvodu, aby sa poprípade mohli generovať powerupy s veľmi malou pravdepodobnosťou, no za to s veľmi veľkým efektom, ktoré by pre rozlíšenie mali iný model. Funkciu TimeTick majú v sebe definovanú takmer všetky triedy programu PowerUp. Ide o funkciu, ktorá sa vola v pravidelných intervaloch pre každý objekt v scéne. Pomocou nej si potom každý objekt môže definovať čo sa s ním s plynutím herného času bude diať. Napríklad v prípade v prípade objektu tanku sa vo funkcii TimeTick aktualizuje otočenie delovej veže a aktualizuje sa tiež strela vystrelená tankom.

V triede TPowerUp sa ďalej nachádzajú funkcie, ktoré vracajú premenné triedy. Funkcia pre nastavenie pozície powerupu, pričom táto funkcia neberie do úvahy terén scény. To je problém s umiestňovaním objektov do trojrozmernej krajiny. Väčšinou chceme objekt umiestniť presne na povrch. Na to je v programe TankGame implementovaná metóda, ktorá vráti výšku terénu v danom bode plochy. Túto metódu využíva aj funkcia DeployRandomly, ktorej parametrami sú údaje definujúce stred a rozmery obdĺžnika v ktorého ploche sa powerup náhodne umiestni do scény.

3.2.2 Implementácia konkrétnych powerupov

Na demonštráciu schopností bázovej triedy TPowerUp bolo pre program TankGame implementovaných päť konkrétnych powerupy. Sú to triedy TMine, TMedKit, TSpeedUp, TSlow a TDamageIncrease, ktoré sú všetky podedené z bázovej triedy TPowerUp. Všetky vymenované powerupy majú, tak ako bolo už naznačené, spoločný model, ktorým je kocka. Táto kocka je potiahnutá textúrou, tak aby pripomínala balíček, respektíve bedňu s neznámym obsahom. Po kontakte s ľubovoľným z týchto powerupov sa v scéne, na mieste, kde predtým stál powerup, objaví efekt, ktorý je k dispozícii v programe TankGame. Efekt pripomína výbuch. TMine reprezentuje mínu, takže po kontakte s týmto objektom sa tanku, ktorý sa dostal do jeho blízkosti odpočítajú hitpointy – čo sú body udávajúce stav poškodenia. Opačný efekt má powerup TMedkit, ktorý naopak hitpointy pričíta, čím tank opraví. TSpeedUp a TSlow sú dva powerupy s navzájom inverznými 23

triggermi (tak isto ako TMine a TMedkit) a menia maximálnu rýchlosť tanku. TDamageIncerase, ako názov napovedá, zvyšuje palebnú silu tanku.

K renderovaniu objektu každého powerupu dochádza vo funkcii Render, ktorá u každej jednej konkrétnej implementácii vymenovaných powerupov ukladá transformačnú maticu, posunie pozíciu vykresľovania na pozíciu powerupu uloženú v premennej Position typu TVector. Následne sa v tejto funkcii volá metóda modelu typu VRML triedy TModelVRML Render, ktorá sa postará o vlastné vykreslenie textúrovaného objektu do scény.

K vlastnej interakcii powerupu s tankom dochádza vo funkcii TimeTick, ktorá testuje či došlo ku kolízii s niektorým s aktívnych tankov v scéne. Pre tento účel je funkcii predávaný zoznam všetkých tankov, ktorý je následne prechádzaný a v každom priechode sa testuje či niektorý tank zo zoznamu vošiel do štvorca vymedzujúceho oblasť kontaktu. Ak došlo ku kontaktu, nastaví sa premenná Active na false a vyvolá sa príslušný trigger, ktorým je napríklad v prípade powerupu triedy TMine metóda triedy tanku UpdateHitPoints.

Všetky triedy boli implementované s komentárom vo formáte generátoru programovej dokumentácie doxygen, tak aby sa dokumentácia dala doplniť o informácie o nových triedach a metódach zo zdrojového súboru PowerUp.cpp a hlavičkového súboru PowerUp.h.

3.2.3 Generátor powerupov

Aby boli powerupy použiteľné v počítačovej hre, musí sa nejaký systém starať o ich náhodný výber a náhodné rozmiestňovanie po scéne a to všetko v určitých časových intervaloch. Pre tento účel bola vytvorená trieda TPowerUpGenerator. Generátor powerupov pracuje v určitom okruhu, ktorý je definovaný obdĺžnikom. Pri volaní implicitného konštruktoru sa okruh nastaví na štvorec, ktorý sa nachádza v bode [0 , 0], čo je stred nekonečnej krajiny. K dispozícii je aj druhý konštruktor, ktorý obsahuje parametre pre definovanie obdĺžnika ľubovoľných rozmerov. Posledný s parametrov je časový rozdiel medzi umiestňovaním powerupov do scény, ktorý je v prípade implicitného konštruktora nastavený na čas desať sekúnd.

K samotnému generovaniu powerupov dochádza k funkcii TimeTick, ktorá musí byť volaná v hlavnej funkcii TimeTick celého programu TankGame. Funkcia potom testuje časový rozdiel od posledného vygenerovaného powerupu a ak tento časový rozdiel prekročí hodnotu premennej time_difference uloženej v inštancii triedy TPowerUpGenerator, dôjde k vygenerovaniu náhodného čísla, podľa ktorého sa vyberie jeden z piatich powerupov, ktoré boli na demonštráciu do programu implementované.

Na generovanie náhodného čísla, ktoré sa potom použije na výber powerupu používa trieda TPowerUpGenerator objekt triedy TRandomNumber, ktorá je súčasťou programu TankGame. Objekt tejto triedy sa musí najprv inicializovať počiatočnou hodnotou od ktorej sa generovanie pseudonáhodných čísiel odvíja. Aby bolo generovanie čísiel čo najviac náhodné, nastavuje sa 24

počiatočná hodnota na súčin náhodného čísla vygenerovaného funkciou rand() zo štandardnej knižnice jazyka C a aktuálnym reálnym časom.

3.2.4 Úpravy v zdrojových kódoch programu TankGame

Nasleduje vymenovanie úprav, ktoré boli pre funkčnosť systému powerupov nutné. Powerupy boli implementované pre verziu TankGamu pre viacerých hráčov. V súbore Tank.h bola v triede TTankSimple presunutá štruktúra config s konfiguráciou tanku do verejnej časti triedy, tak aby sa k nej dalo pristupovať aj z ostatných objektov – aby mohli powerupy meniť vlastnosti tanku. V rovnakom súbore došlo k rovnakej úprave s premennou magazine v triede TTank, tak aby sa mohla meniť útočná sila tanku. V súbore Unit.cpp bola upravená metóda triedy TUnit s názvom UpdateHitpoints tak aby prijímala aj záporné čísla. Do súboru Global.h bol do triedy Global pridaný zoznam všetkých powerupov. V súbore TankGame.h boli pridané identifikátory pre nové tlačidlá v menu. Funkcia TimeTick v súbore TankGame.cpp bola doplnená o správu všetkých powerupov, taktiež bola doplnená funkcia Render, doplnené boli tiež položky v menu. Do súboru Makefile bola pridaná zmienka o dvoch nových súboroch, v ktorých sú uložené triedy týkajúce sa powerupov a to PowerUp.cpp a PowerUp.h.

4 Záver

Práca na tejto bakalárke bola sériou prekonávania ťažkých mét. Prvou ťažkou métou bolo naštudovanie zdrojových súborov programu TankGame, čo mne osobne prinieslo novú skúsenosť s prácou na projekte, ktorý ma rozmery niekoľko krát väčšie ako bežné tímové projekty predmetov programu informačné technológie. Druhou métou bolo naštudovanie OpenGL a implementácie systému powerupov a následnej integrácie do takto zložitého systému s ohľadom na hrateľnosť a zábavnosť výsledného herného produktu. Treťou métou bolo samotné písanie bakalárskej práce a študovanie teoretického základu fungovania grafických systémov využívaných v počítačových hrách (alebo iných systémoch pracujúcich v reálnom čase). Pre mňa osobne to prinieslo množstvo nových informácií o fungovaní grafického pipelinu a o zobrazovaní zložitých 3D scén.

Jedným z hlavných cieľov bakalárskej práce bolo naprogramovať systém powerupov do hry TankGame, ktorý som splnil. Táto stránka práce bola viac menej ukončená, bez možnosti ďalšieho vážneho rozširovania. Samozrejme sa dajú doplniť detaily ako nové efekty, nové komplexnejšie powerupy, ktoré by ovplyvňovali celú scénu a podobne. Druhý cieľ, teda naštudovať a pochopiť fungovanie zobrazovania 3D objektov a scén je možné rozvíjať aj ďalej v budúcnosti prakticky donekonečna. Jedná sa totižto o najrýchlejšie sa rozvíjajúce odvetvie programovania. Prakticky každý polrok prichádzajú velikáni grafiky s novými a novými prvkami a technikami. Táto téma ma preto veľký potenciál na spracovanie napríklad v diplomovej práci.

Námetom na ďalšie projekty by teda bolo napríklad vytvorenie grafického enginu, alebo implementovanie niektorých pokročilých grafických techník do programu TankGame, ako napríklad antialiasing, anizotropické filtrovanie, či použitie bump máp, čo sú techniky bez ktorých sa dnešná počítačová hra neobíde.

Literatúra

[1] Žára J., Beneš B., Sochor J., Felkel P. Moderní počítačová grafika, 2004

[2] WikiPedia.org, OpenGL. Dokument dostupný na URL http://en.wikipedia.org/wiki/Opengl (máj 2007)

[3] WikiPedia.org, Ray tracing. Dokument dostupný na URL http://en.wikipedia.org/wiki/Raytracing (máj 2007)

[4] WikiPedia.org, Anisotropic filtering. Dokument dostupný na URL http://en.wikipedia.org/wiki/Anisotropic_filtering (máj 2007)

[5] WikiPedia.org, VRML. Dokument dostupný na URL http://en.wikipedia.org/wiki/VRML (máj 2007)

[6] UPGM FIT VUT, Stránky projektu TankGame, podzim 2005, Dokument dostupný na URL http://merlin.fit.vutbr.cz/TankGame/ (máj 2007)

[7] Extremetech.com, 3D Popeline. Dokument dostupný na URL http://www.extremetech.com/article2/0,1697,1154772,00.asp (máj 2007)

[8] Extremetech.com, Game engine anatomy. Dokument dostupný na URL http://www.extremetech.com/article2/0,1697,1150107,00.asp (máj 2007)

Zoznam príloh

Príloha 1. Zdrojové texty dostupné v zložke src s názvom PowerUp.cpp a PowerUp.h

Príloha 2. CD so zdrojovými textami programu TankGame vrátane zdrojových súborov PowerUp.cpp a PowerUp.h

Príloha 3. Manuál dostupný na CD prílohe v súbore Manual.pdf