Použitie octree štruktúry na urýchlenie metódy ray tracing.

Jozef Martinka
Katedra aplikovanej matematiky
Matematicko fyzikálna fakulta Univerzity Komenského,
842 15 Bratislava, Slovenská republika
e-mail: jozef.martinka@st.fmph.uniba.sk

Abstrakt.

Niektoré spôsoby urýchlenia metódy *ray tracing* využívajú priestorové rozdelenie scény a jej reprezentáciu pomocou *octree* štruktúry. Hlavné problémy spojené s použitím *octree* sú: vhodná reprezentácia, efektívny spôsob vytvorenia (rozdelenia scény na kocky *octree*) a rýchly "prechod" sledovaného lúča cez štruktúru. V článku sú stručne popísané niektoré prístupy k riešeniu týchto problémov.

1. Úvod.

Ray tracing (ďalej len RT) je v súčasnosti jedna z najznámejších metód pre zobrazovanie trojrozmerných scén pomocou počítača (navrhnutá Whittedom v roku 1980). Jej použitie dáva výsledky približujúce sa fotorealistickému zobrazeniu modelovanej scény, no na druhej strane, je to časovo veľmi náročná metóda i pri použití najrýchlejších počítačov, aké sú v súčasnosti k dispozícii. To znemožňuje použiť túto metódu napríklad pre animáciu v reálnom čase. Preto sa vynakladá značné úsilie na nájdenie spôsobov, ako čo najviac urýchliť prácu RT.

2. Stručný prehľad niektorých spôsobov urýchlenia RT.

Ukázalo sa, že časovo najnáročnejším problémom RT je nájdenie priesečníkov sledovaných lúčov s telesami v scéne. Tento problém obsahuje dva podproblémy:

a) Ako určiť, ktoré telesá v scéne lúč pretína s veľkou pravdpodobnosťou? (Možno však formulovať aj takto: Určiť, ktoré telesá lúč určite nepretína.)

b) Ako reprezentovať telesá tak, aby bol výpočet priesečníka s lúčom dostatočne jednoduchý?

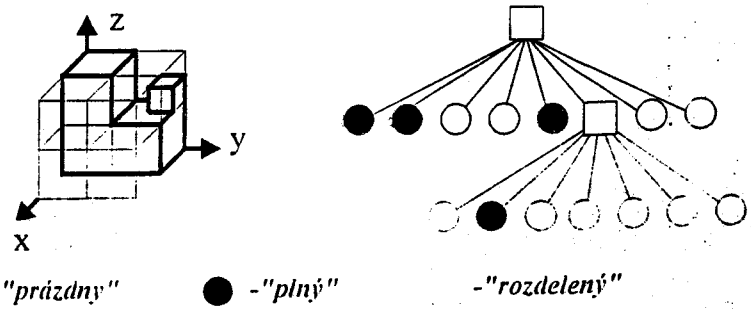
V ďalšom texte je pozornosť venovaná problému a). Prvé metódy na rýchlejšie určenie toho, či lúč potenciálne pretína teleso, používajú ohraničujúce objemy telies [SoŽa89]. Za ohraničujúce objemy sa najčastejšie volí guľa alebo kváder, lebo prienik lúča s týmito telesami sa dá relatívne rýchlo vypočítať. Iné metódy využívajú koherenciu susedných lúčov, t.j. predpokladá sa, že susedné lúče s veľkou pravdepodobnosťou pretínajú tie isté telesá. Ďalšie metódy sa snažia nejakým spôsobom rozdeliť scénu na časti. Potom sledujú, cez ktoré časti sledovaný lúč prechádza a na základe toho počítajú prienik lúča len s tými telesami, ktoré sa v týchto častiach nachádzajú. Objavujú sa aj metódy, ktoré používajú niektoré netradičné matematické postupy, napr. princíp duality [Koli93]. Tieto metódy sú väčšinou závislé na spôsobe matematického popisu scény.

3. Techniky delenia priestoru scény.

Techniky delenia priestoru umožňujú rozdeliť priestor scény na nejaké časti, pričom sa uchováva informácia o tom, ktoré telesá ležia v jednotlivých častiach. Existujú dva hlavné prístupy k rozdeľovaniu scény na časti: rovnomerné a adaptívne delenie priestoru. Pri rovnomernom delení sa priestor rozdelí na jednotky objemu rovnakej veľkosti a tvaru - *voxle* (najčastejšie bývajú tvaru kocky). Príkladom rovnomerného rozdelenia priestoru je kváder, ktorý ohraničuje scénu, rozdelený na $i \times j \times k$ kociek rovnakej veľkosti. Pri adaptívnom delení sa priestor scény delí na objemy rôznej veľkosti a tvaru podľa určitých kritérií delenia. Kritériom delenia môže byť napríklad počet telies v danom objeme. Ak je počet telies väčší ako stanovená hranica, tak sa daný objem rozdelí na menšie časti. Kritériá pre určenie hĺbky, do akej je ešte výhodné scénu deliť sa neustále vyvíjajú [MSHG92, Sung91]. Zakladajú sa nielen na počte telies v danej kocke, ale aj na pravdepodobnosti, s akou lúč pretína kocku v určitej hĺbke delenia, prípadne i na objeme, ktorý telesá v kocke zaberajú, a podobne. Od voľby kritérií závisí hlavne pamäťová náročnosť dátových štruktúr, v ktorých sa jednotlivé časti priestoru uchovávajú.

4. Adaptívne delenie priestoru s použitím OCTREE štruktúry.

Predpokladajme, že je scéna ohraničená najmenšou kockou, do ktorej sa ešte celá zmestí. Teraz sa na základe zvolených kritérií (napr. počet telies v kocke) buď v delení kocky nepokračuje, alebo sa rozdelí na 8 rovnakých kociek. Potom sa aplikuje kritérium a prípadne nasledujúce delenie na každú z 8 nových menších kociek. Tento postup sa opakuje dovtedy, kým existuje nejaká kocka, ktorá vyhovuje kritériu pre ďalšie delenie. V kritériu by mala byť zahrnutá aj maximálna hĺbka delenia, pretože v praxi sú pamäťové možnosti obmedzené. Štruktúra, ktorá vznikne, môže byť reprezentovaná stromom, ktorého vrcholy sú jednotlivé kocky (obr.1). Ak kocka neobsahuje žiadne teleso, tak sa v strome reprezentuje listom s hodnotou *"prázdny"* (*empty*). Ak kocka obsahuje nejaké telesá a ďalej sa nedelí na podkocky, tak sa tiež reprezentuje listom stromu s označením *"plný"* (*full*). Vtedy navyše obsahuje odkaz na zoznam telies, ktoré obsahuje. Nakoniec, ak je kocka rozdelená, tak sa v strome reprezentuje vrcholom, ktorý je rodičom ďalších 8 vrcholov (kociek) a označí sa ako *"rozdelený"* (*heterogenous*). Preto sa táto štruktúra nazýva *octree*.



obr.1 *octree*

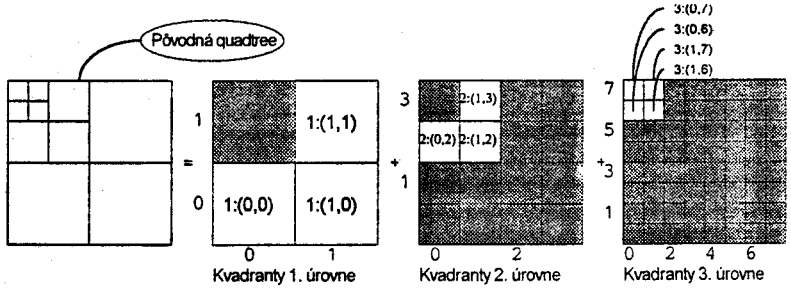
5. LVSI kódovanie OCTREE štruktúry.

Kelvin Sung v [Sung91] navrhol reprezentáciu *octree* štruktúry pomocou ***LVSI*** (Leveled Voxel Space Index), kde je každá kocka identifikovaná súradnicami L:(i,j,k). L je hĺbka v ktorej sa kocka nachádza (najväščia kocka ohraničujúca scénu má hĺbku 0), (i,j,k) sú súradnice kocky v hĺbke L z intervalu $\langle 0; 2^L - 1 \rangle$. Jednoduchým spôsobom možno zistiť súradnice rodiča i potomka každej kocky. Rodič kocky L:(i,j,k) ma súradnice L-1:(i div 2, j div 2, k div 2). Potomkovia kocky L:(i,j,k) majú súradnice L+1:(2i+a, 2j+b, 2k+c) pre všetkých osem možnosti trojic $(a,b,c) \in \{0,1\}^3$.

Na reprezentáciu *octree* v pamäti K. Sung navrhuje použiť hašovaciu tabuľku. Každý riadok tabuľky má osem priehradok. Riadok tabuľky, do ktorého patrí kocka so súradnicami L:(i,j,k), sa určí na základe jej súradníc nasledujúcou funkciou :

riadok=((i>>1) | ((j>>1)<<L) | ((k>>1)<<2L)) mod *TableSize*,

kde >> je bitový posun vpravo, << vľavo, | je logický súčet a *TableSize* je veľkosť tabuľky. Súčtom najnižších bitov v súradniciach i, j, k dostane adresu priehradky v danom riadku tabuľky (0-7).



obr. 2 LVSI zúžená na 2D quadtree.

Táto reprezentácia je výhodná z hľadiska pamäťovej náročnosti, lebo neobsahuje údaje o prázdnych a rozdelených kockách *octree* štruktúry. Uchováva len informácie o listoch, ktoré zodpovedajú plným kockám. Z hľadiska vyhľadania daného listu je však vo väčšine prípadov pomalšia ako stromová reprezentácia.

6. Vytvorenie OCTREE štruktúry.

6.1. Problém určenia existencie prieniku telesa a kociek OCTREE štruktúry.

Jedným z problémov vytvorenia *octree* štruktúry je určenie, či dané teleso má, alebo nemá s danou kockou *octree* prienik. Riešenie závisí od reprezentácie telies. Pre telesá reprezentované pomocou hraničných polygónov tento problém riešia Z. Tang a S. Lu v [TaLu88].

Pre RT stačí uchovávať informácie len o tých častiach priestoru, cez ktoré prechádzajú povrchy telies, pretože sledované lúče do telies vchádzajú a z telies vychádzajú cez ich povrch.

Telesá v scéne však možu byť reprezentované aj pomocou základných objemových telies (guľa, valec, kužeľ, atď.) s použitím známej CSG reprezentácie (popísanej napr. v [KuWy??]). Táto reprezentácia je výhodná pre RT, lebo je jednoduché preniesť množinové operácie na intervaly na pretinajúcom lúči. Nie je však vhodná na vytvorenie *octree* štruktúry, lebo hľadať prienik kociek napríklad s kužeľom alebo toroidom nie je celkom jednoduché. Tento problém možno riešiť rôznymi spôsobmi. Relatívne jednoducho sa dá určiť existencia prieniku kocky s guľou alebo s kvádrom, ktorého steny sú rovnobežné s rovinami xy, yz, xz. Preto sa často používajú na vytvorenie *octree* štruktúry, napríklad tak, že sa každé teleso scény obalí kvádrom (resp. guľou), alebo sa povrch telies pokryje množinou "dostatočne" malých kvádrov (prípadne kociek, alebo gúľ). Potom sa pri hľadaní prieniku kociek *octree* s telesom hľadá prienik s kvádrom (množinou kvádrov, kociek, gúľ), ktorý dané teleso (resp. jeho povrch) aproximuje.

Pre svoju implementáciu som sa pokúsil navrhnúť iný spôsob vytvorenia *octree*, popísaný v nasledujúcej kapitole.

6.2. Dočasná triangulácia povrchov telies.

Pred začatím vytvárania *octree* štuktúry sa povrchy všetkých telies aproximujú trojuholníkmi. Záznam o každom trojuholníku nesie so sebou informáciu o tom, ktorému telesu patrí. Štruktúra sa potom vytvorí nad množinou trojuholníkov, čo je relatívne jednoduchá úloha. Na výstupe tohto procesu je zoznam kociek označených pomocou LVSI súradníc, pričom každá kocka *octree* informuje o tom, ktoré telesá obsahuje. Neobsahuje však žiadnu informáciu o trojuholníkoch, ktoré boli použité len ako prostriedok pre vytvorenie *octree* (každý trojuholník odovzdal kocke, s ktorou mal prienik, informáciu o telese, ktorého povrch aproximoval). V nasledujúcom odseku je popísaný algoritmus použitý pre určenie existencie prieniku trojuholníka a kocky.

Treba vari poznamenať, že povchy telies je nutné aproximovať aj z vonkajšej, aj z vnútornej strany (obr.3). Inak by nepresnosť aproximácie mohla spôsobiť, že v niektorých kockách *octree* budú informácie o prechádzajúcom povrchu chýbať. Neplatí to však pre telesá, ktoré sú reprezentované pomocou hraničných polygónov. Vtedy stačí triangulovať ich hraničné polygóny. Veľmi často sa používa reprezentácia povrchu vyrobená priamo z trojuholníkov a teda jeho trianguláciu už netreba vyrábať.

obr.3 2D rez trojuholníkovou obálkou povrchu telesa

Tento postup umožňuje vytvoriť *octree* štruktúru s väčšou presnosťou aproximácie (do väčšej hĺbky *octree*) povrchu telies, ako metódami, ktoré sú popísané na konci predošlej kapitoly. Jeho časová a pamäťová náročnosť je však väčšia.

6.3. Prienik trojuholníka s kockou.

Čiastočným problémom vytvorenia *octree* štruktúry postupom popísaným vyššie, je vyriešenie otázky, či má daný trojuholník s danou kockou prienik. Nezaujíma nás ako tento

prienik vyzerá, ale len to, či neprázdny prienik existuje alebo nie. Na riešenie tohto problému možno použiť nasledujúci postup:

V prvej fáze (kroky 1,2,3) možno použiť zovšeobecnený Cohen - Sutherlandov algoritmus na orezanie úsečky v 3D (popísaný napr. v [FVFH90]). Jeho činnosť však treba redukovať len na zistenie existencie prieniku úsečky (hrany trojuholníka), lebo výsledné orezanie úsečky nás nezaujíma. Postupne treba overiť:

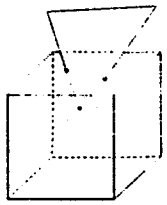
1) Či nejaký vrchol trojuholníka leží v kocke (obr. 4a). Ak áno, tak neprázdny prienik existuje, inak

2) Či všetky vrcholy súčasne ležia nad (pod, pred, za, vľavo, vpravo) kockou (obr. 4b). Ak áno, tak neprázdny prienik neexistuje. inak

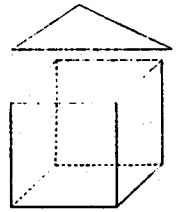
3) Nájdeme priesečníky hrán trojuholníka so stenami kocky (obr. 4c). Ak aspoň jeden leží v niektorej stene kocky, tak neprázdny prienik existuje. inak

4) Ak predošlé tri testy neurčia jednoznačne existenciu neprázdneho prieniku, treba ešte testovať prípad, keď všetky tri vrcholy ležia mimo kocky a žiadna hrana nepretína kocku. V tomto prípade ešte existuje možnosť, že vnútro trojuholníka má s kockou neprázdny prienik (obr. 4d). To sa dá zistiť tak, že sa nájde prienik trojuholníka s tou telesovou uhlopriečkou kocky, ktorá je "najkolmejšia" na rovinu trojuholníka. Ak tento prienik neexistuje, tak určite takýto trojuholník nemá s kockou prienik.

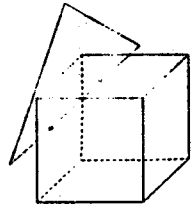
Krok 4) dopĺňa test, ak kroky 1) až 3), ktoré sú založené na testovaní vrcholov a hrán trojuholníka, neurčia (ne)existenciu prieniku.



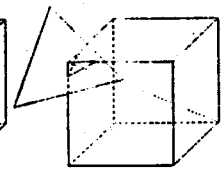
obr. 4a



obr. 4b



obr. 4c



obr. 4d

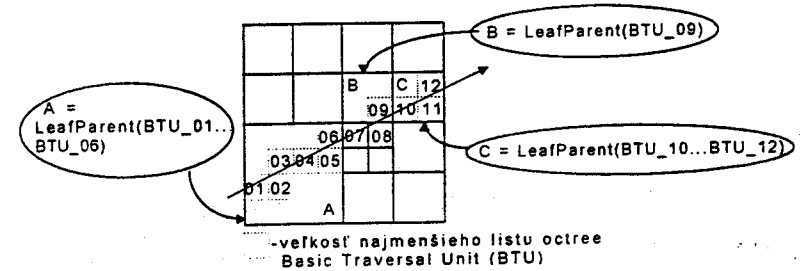
7. Prechod lúča cez scénu reprezentovanú pomocou OCTREE štruktúry.

Doteraz sa neurčovalo, ktorými kockami a v akom poradí lúč prechádza cez *octree* štruktúru.

Jednu z možností, ako nájsť príslušný list *octree* uvádzajú Fujimoto a Iwata v [Fulw85]. Je to vertikálne prehľadávanie stromu, ktorý *octree* štruktúru reprezentuje. V ich práci tiež uvádzajú *3DDDA* (Three Dimensional Digital Differential Analyser) algoritmus na určenie postupnosti kociek, ktorými lúč prechádza v priestore rovnomerne rozdelenom na rovnaké kocky. Ten používajú na horizontálne prehľadávanie stromu, keď je potrebné určiť ako lúč prechádza cez osmicu kociek *octree*, na ktoré bola rozdelená rodičovská kocka. Týchto osem kociek tvorí rovnomerne rozdelený priestor 2 x 2 x 2 rovnakých kociek a teda v ňom môže byť použitý *3DDDA*. Stromová reprezentácia *octree* umožňuje síce rýchle vyhľadanie listu v strome, ale je nevýhodná z hľadiska pamäťovej náročnosti, lebo sú v nej uložené informácie aj o prázdnych aj o rozdelených kockách.

Iný spôsob riešenia tohto problému podáva Kelvin Sung vo svojom článku [Sung91]. *octree* štruktúru reprezentuje pomocou *LVSI* (pozri kap. 5) a ukladá ju v hašovacej tabuľke. Na prechod lúča kockami *octree* využíva modifikovaný *3DDDA* (popísaný napr. v [Fulw85]). Priestor sa fiktívne rovnomerne rozdelí na kocky (nazýva ich *BTU* - Basic

Traversal Unit) takej veľkosti, aká majú najhlbšie listy (najmenšie kocky) *octree*. No v skutočnosti sa uchováva len *octree* štruktúra reprezentovaná pomocou *LVSI* a uložená v hašovacej tabuľke.



obr. 5 Modifikovaný 3DDDA zúžený na 2D quadtree.

Kocky *BTU* sú použité ako základné jednotky pre *3DDDA* algoritmus. Modifikovaný *3DDDA* pracuje tak, aby našiel potrebný list *octree* na základe toho, ktorou *BTU* jednotkou práve sledovaný lúč prechádza (obr.5). Sung navrhol funkciu *GetLeafParent()*, ktorá pre *BTU* zadanú pomocou *LVSI* súradníc nájde v hašovacej tabuľke list *octree* reprezentujúci kocku, v ktorej táto BTU leží.

```
GetLeafParent (Level, i, j, k)
  LeafParent = HashTableLookup (Level, i, j, k)
  while (LeafParent == NULL)
   Level = Level - 1
   i = i >> 1 /* delenie 2 */
   j = j >> 1
   k = k >> 1
   LeafParent = HashTableLookup (Level, i, j, k)
  return (LeafParent)
```

obr. 6 Funkcia *GetLeafParent*.

8. Záverečné poznámky.

Metódy popísané v tomto článku sú časťou teoretického podkladu mojej práce v rámci Študentskej vedeckej konferencie MFF UK Bratislava. Jej cieľom je štúdium a implementácia metódy *ray tracing*. Článok však neobsahuje dosiahnuté výsledky, pretože práca na implementácii ešte nebola ukončená.

9. Zoznam použitej literatúry:

[Fulw85] FUJIMOTO, Akira, - IWATA, Kansei: Accelerated Ray Tracing, Computer Graphics: Visual Technology and Art (Proceedings of Computer Graphics Tokyo '85), pp. 41-65.

[FuKu85] FUJIMURA, Kikuo, - KUNII, T. L.: A Hierarchical Space Indexing Method, Computer Graphics: Visual Technology and Art (Proceeding of Computer Graphics Tokyo '85), pp. 21-33.

[FVFH90] FOLEY, J.D., - VAN DAM, A., - FEINER, S., - HUGHES, J.: Computer Graphics:Principles and Practice, Addison-Wesley Publishing Company, Inc., 1990.

[Koli93] KOLINGEROVÁ, I.: Využití duálního prostoru pro metodu sledování paprsku, Zimní škola PG, Plzeň, 1993.

[KuWy??] KUNII, T. L., - WYVILL, G.: CSG and Ray Tracing Using Functional Primitives, pp. 137-152, Computer Generated Images: The State of the Art.

[Mart92] MARTINKA, J.: Ray Tracing, ŠVK 1992, MFF UK Bratislava.

[Mart93] MARTINKA, J.: Ray Tracing (implementácia urýchľovacieho algoritmu), ŠVK 1993, MFF UK Bratislava.

[MSHG92] McNEIL, M.D.J., - SHAH, B.C., - HÉBERT, M.-P., - LISTER, P.F., - GRIMSDALE, R.L.: Performance of Space Subdivision Techniques in Ray Tracing, Computer Graphics forum, Volume 11 (1992), number 4, pp. 213-220.

[Ruži91] RUŽICKÝ, E.: Úvod do počítačovej grafiky, skriptum, MFF UK Bratislava, 1991.

[SoŽa89] SOCHOR, J., - ŽÁRA, J.: Světlo a stín v počítačové grafice, Proceedings of Moderní programování I. diel, 1989

[Sung91] SUNG, Kelvin: A DDA Octree Traversal Algorithm for RayTracing, Eurographics '91, pp. 73-85.

[TaLu88] TANG, Zesheng, LU, Shengkai: A New Algorithm for Converting Boundary Representation to Octree, Eurographics '88, pp. 105-116.

[Zeit92] ZEITLBERGER, René: The Octree Data Structure, Seminar aus Informatik, Technische Universität Wien, 1992.

PARALLELISATION OF THE RAY–TRACING ALGORITHM

Jiří Žára *(zara@cs.felk.cvut.cz)*
Aleš Holeček *(zholecek@sun.felk.cvut.cz)*
Jan Přikryl *(zprikryl@sun.felk.cvut.cz)*

CTU, Fac. of Electrical Eng.
Dept. of Computer Science
Karlovo nám. 13,
121 35 Praha 2.

Abstract

The two typical methods for distribution of ray–tracing rendering algorithm are presented in this article. The implementation of a distributed ray-tracer on a network of UNIX workstations is described in details. The first results are discussed from the point of view of memory load, time of computation and cost of communication among processes.

Keywords

computer graphics, rendering, parallel algorithm, ray-tracing, PVM.

1 Ray–tracing algorithm

The methods of computer generated, realistic looking pictures of three dimensional scenes are characterized by their extremely high claims on computing equipment and the fact that they are incredibly time consuming. Typical algorithms (ray tracing, radiosity method) are so complex and complicated, that their direct transformation into computer hardware is not effective yet. The performance capacity of a single CPU and the amount of memory available on today's personal computers and workstations are still low for large, reality describing scenes when using the algorithms mentioned above.

Although some rendering methods were already implemented into hardware of graphics workstations (z–buffer, Gouraud shading), the ray–tracing technique is still too complex method and its hardware support is a task for computing equipments in the future.

We can compare typical features of now-a-days rendering methods done by hardware with ray-tracing in the following table:

z–buffer & Gouraud shading	ray–tracing
integer arithmetics in raster	floating point arithmetics in 3D space
sequential processing of polygons	computations in whole 3D scene
constant number of rendered faces	recursive generation of rays

The disadvantages of the ray–tracing algorithm are clear, but the quality of images rendered by ray–tracing is so high, that this algorithm is used in many applications. Its improvement and increasing of its efficiency are one of current topics in computer graphics.

The ray–tracing algorithm is based on tracing of rays, that are shot from viewpoint through the screen window to the scene. When an intersection point between a ray and the nearest solid or face is found, a new generation of rays is created — one reflected ray, one refracted ray (if solid is transparent or semitransparent) and several "shadow" rays (one for each light source). Recursive creation of rays is finished either after several generations or in case of achievement of