Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Detekce kolizí vhodná pro pohybující se modely v oblasti VR

Plzeň, 2006 Martin Pokorný

Abstrakt

This diploma thesis describes accelerating methods for collision detection in virtual reality acceleration. Acceleration is based on suitable subdivision of space and a reduction of numer of the candidate objevte to be tested on intersection. This subdivision of space could be use for another purpose than only for collision detection. Described methods are usefull for scenes with objects with non-uniform as well as uniform distribution. Methods were tested and compared together.

Poděkování

Na tomto místě bych chtěl poděkovat hlavně vedoucí mé diplomové práce Doc. Dr. Ing. Ivaně Kolingerové za vedení mé práce, poskytnutí celé řady materiálů, které mi pomohly, a čas, který mi věnovala.

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 17. května 2006

Martin Pokorný

……

Obsah

1 Úvod

Problematika detekce kolizí je velmi komplexní problém. Má využití v mnoha oblastech, jako například CAD/CAM systémy, výrobní simulace, různé modely založené na fyzikálních vlastnostech v molekulárním modelování a virtuální realitě. V každé oblasti jsou požadavky na proces detekce kolizí rozdílné. V CAD/CAM systémech a různých fyzikálních modelech je kladen důraz především na přesnost testu kolize, zatímco např. ve virtuální realitě je zájem především o rychlost testu kolize, zatímco přesnost ustupuje do pozadí. Pro zvýšení rychlosti detekce kolize se používá celá řada urychlovacích metod.

Tato diplomová práce se zabývá detekcemi kolizí ve virtuální realitě. Je zde tedy kladen důraz především na rychlost detekce. Měl jsem k dispozici referenční vypracování detekce kolize z diplomové práce pana Ing. Jaroslava Matouška, na které jsem měl za úkol navázat. Toto vypracování bylo vytvořeno pro scény s rovnoměrným rozložením objektů, kdy ve scéně je většina statická a jen několik objektů dynamických. Úkolem mé práce bylo vyzkoušet v tomto programu jiné urychlovací techniky pro zrychlení detekce pro obecné scény. Právě podle možných druhů scén byly vybrány urychlovací metody, aby detekce byla co nejrychlejší. Metody zde odzkoušené mají za úkol odstranit co největší počet objektů, které nemohou kolidovat. Závěrem práce je pak porovnání implementovaných metod mezi sebou a s referenčním řešením.

2 Teorie

V této kapitole jsou uvedeny urychlovací techniky používané pro zrychlení testu detekce kolize. Na konci této kapitoly je popsán samotný proces detekce kolize a možné problémy, spojené s pohybem objektů v čase. Obsah kapitoly vychází z [1] a je doplněn o další metody. Zdroje k popisovaným metodám jsou následující: kapitola 2.1.1 Pravidelná mřížka [2], 2.1.2 nepravidelná mřížka [2] [3], 2.1.3 K-D Strom [2] [6] [12], 2.1.4 BSP Strom [3] [11], 2.1.5 Voronoiův diagram [5] [8], 2.1.6 Prostorový strom [7], 2.1.7 Segmentový strom [6], 2.1.8 Množina segmentových stromů [7], 2.1.9 Intervalový strom [6], 2.2.1 Osově zarovnaný box [2] [5] [11], 2.2.2 Obalová koule[3] [4] [5], 2.2.3 Orientovaný box [2] [3] [5] [11], 2.2.4 K-DOP [5] [11].

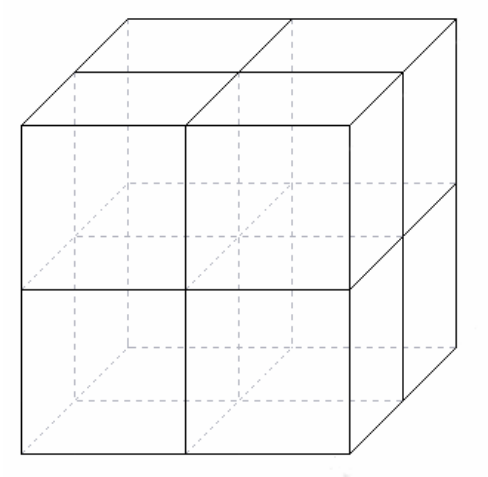
2.1 Rozdělení scény

Pro rychlejší vyhledání objektu v trojrozměrném prostoru je potřebné nějaké systematické rozdělení tohoto prostoru. Konstruují se pomocné struktury zaznamenávající uspořádání těles ve scéně. Tento krok může velice urychlit další testy tím, že vybere do testů pouze objekty, které mají určité společné vlastnosti. Ve většině případů se jedná o polohu v prostoru. Účelem těchto struktur je tedy odstranit co nejvíce objektů, které nemohou s daným tělesem kolidovat.

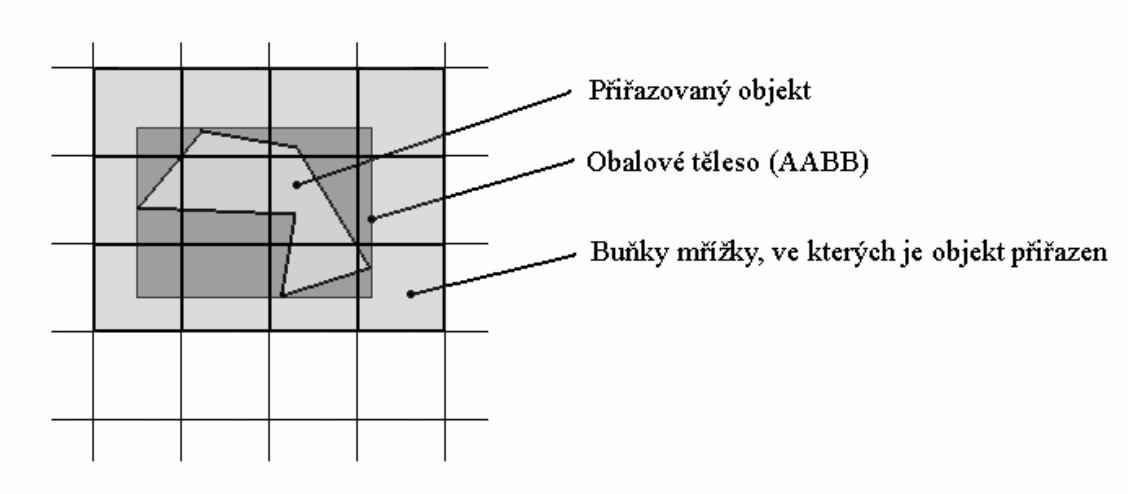
Na rozdělení scény se používají různé datové struktury. Struktury bývají používány podle typu scén a požadavků na aplikaci, neboť každá struktura se více či méně hodí pro určitý typ scény. Dalším aspektem může být paměťová náročnost, kdy určité struktury jsou paměťově více náročné.

2.1.1 Pravidelná mřížka

Pravidelná mřížka je jednou z nejjednodušších struktur pro rozdělení trojrozměrného prostoru. Jedná se o rozdělení prostoru do pravidelných krychlí, jak je na obrázku 2.1.1. Jemnost dělení a tím i počet krychlí je určen při vytvoření struktury. Počet buněk určuje paměťovou náročnost této struktury. Právě paměťová náročnost je hlavní nevýhodou pravidelné mřížky, neboť v mřížce jsou uloženy všechny buňky, ať jsou prázdné nebo ne. Každá buňka si pamatuje objekty, které v ní jsou uloženy, (uložení objektů v mřížce je vidět na obrázku 2.1.2), a každý objekt si pamatuje všechny buňky, ve kterých je uložen.



Obrázek 2.1.1: Pravidelná mřížka trojrozměrného prostoru



Obrázek 2.1.2: Přiřazení předmětu mřížce (pro jednoduchost v E^2)

Výhodou struktury je snadná implementace. Přidávání objektů se provede rychle podle AABB boxu (viz kap 2.2.1) a odebírání se provede jen zrušením ukazatelů buněk mřížky na objekt a objektu na buňky mřížky.

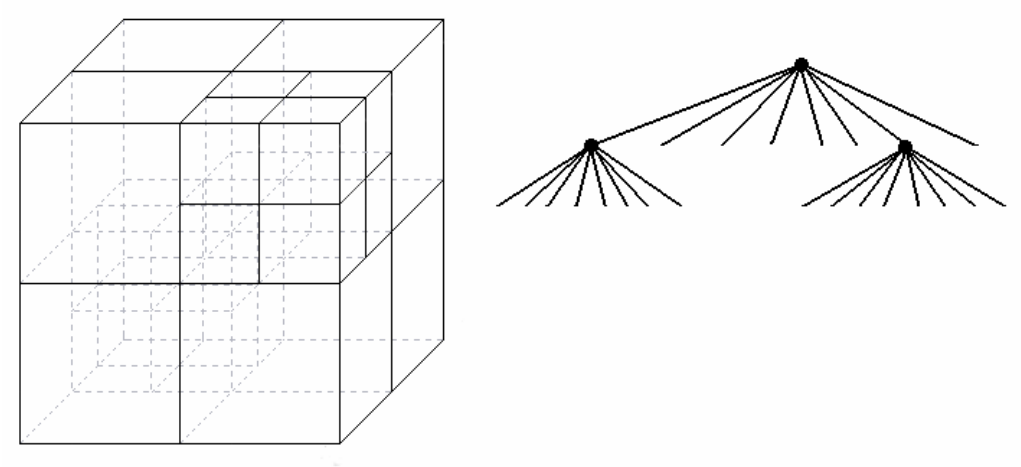
Pravidelná mřížka je určena pro rovnoměrné umístění objektů ve scéně. V případě, kdy by byly všechny objekty v jedné části scény, budou objekty v jedné nebo několika málo buňkách mřížky a mřížka přestane plnit svůj účel, protože do dalších testů postoupí většina objektů.

Velikost a počet buněk je dán aplikací. Závisí však na počtu objektů ve scéně a velikosti objektu. Pokud jsou zvoleny malé buňky, najdeme přesněji, jaké objekty spolu mohou kolidovat a jaké ne, ale velké objekty budou zabírat hodně buněk mřížky. To pak při testování všech buněk, které zabírá, může zpomalit test a struktura bude více paměťově náročná. Na druhou stranu pokud bude mít mřížka velké buňky, bude vybráno více objektů, které se dále testují.

Při testování kolize se pak vybírají pouze ty objekty, které přísluší stejným buňkám jako zkoumaný objekt. Pokud je objekt přiřazen více buňkám, může dojít k vícenásobnému testování. Tuto situaci je nutno v aplikaci ošetřit.

2.1.2 Nepravidelná mřížka

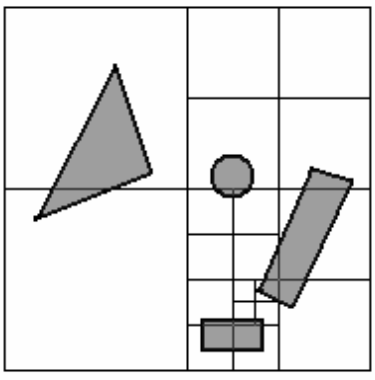
Octtree, neboli oktanový strom, je stromová struktura, která dělí trojrozměrný prostor adaptivně. Buňka (na začátku celý prostor) je vždy rozdělena na osm stejných krychlí viz obr. 2.1.3. Dělení stromu se provádí podle předmětů ve scéně. Jemnost dělení může být dána různými kritérii (např. pokud buňka obsahuje více objektů než je stanoveno). Vždy je ale dána hloubka stromu, kde se má dělení zastavit.



Obrázek 2.1.3: Nepravidelná mřížka (Octtree) [2]

V E^2 odpovídá oktanovému stromu quadtree, na kterém bude dělení stromu názornější. Na obrátku 2.1.4 je vidět jak je prostor dělen pomocí quadtree podle objektů v prostoru.

Strom je tvořen rekurzivně. Prostor je postupně dělen na buňky a tyto buňky jsou řazeny ve stromu. Každá buňka, která nesplňuje daná kritéria (konečným kritériem je hloubka stromu), je rozdělena na osm dalších buněk. V každé buňce je pak uložen seznam objektů, které obsahuje, a objekt si pamatuje, které buňky zabírá.



Obrázek 2.1.4: Quadtree vytvořený podle objektů ve scéně [3]

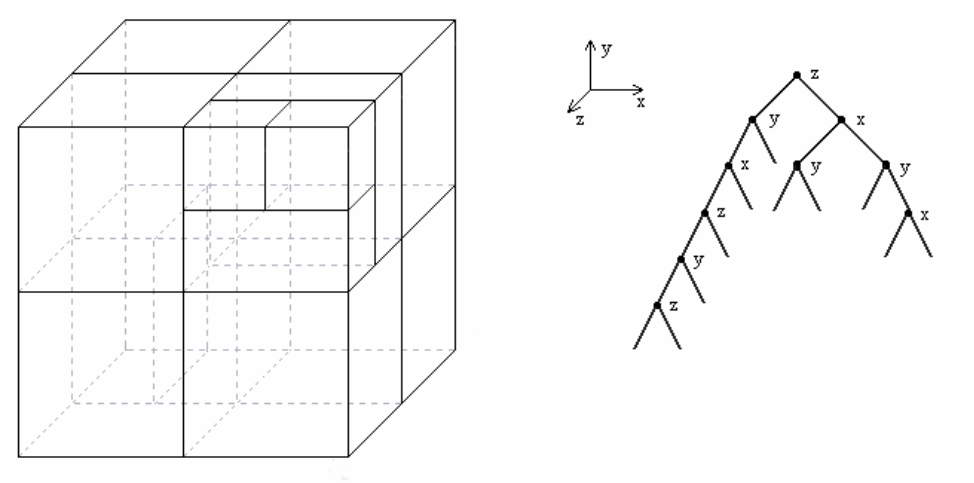
Výhodou oktanového stromu je rychlé nalezení potřebné části prostoru. Pro toto nalezení stačí provést test vůči středu buňky, sestoupit do jednoho z oktantů a test rekurzivně opakovat. Výhodou oproti pravidelné mřížce je adaptivní dělení prostoru jen v části, kde je to potřeba. Tím zpřesníme výběr objektů v těchto částech scény a v místech, kde nic není, ušetříme paměť.

Nevýhodou oktanového stromu je skutečnost, že může obsahovat mnoho prázdných uzlů, jak je vidět např. na obr. 2.1.4. Nevýhodou jsou také násobné ukazatele na objekt, který se může vyskytovat ve více oktanech. Tyto nevýhody jsou zapříčiněny pravidelností dělení. Dělící roviny by bylo možno umisťovat s ohledem na předměty, čímž by se dosáhlo efektivnějšího rozdělení prostoru, ale ztratili bychom výhodu dělitelnosti dvěma.

Stejně jako u pravidelné mřížky se zde musí řešit problém při detekci s vícenásobným testováním.

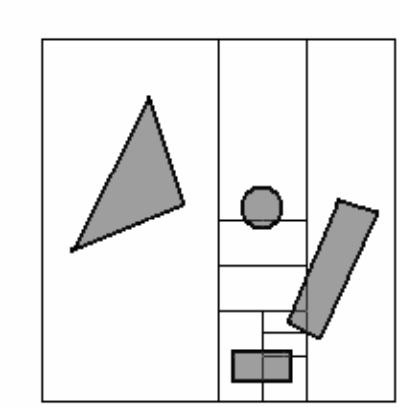
2.1.3 K-D Strom

Tato metoda opět rozděluje prostor dělicími rovinami rovnoběžnými s osami. Rozdíl od předešlé metody je v dělení prostoru. Buňka je vždy rozdělena jen jednou rovinou na dvě části. Výsledná struktura je pak binární strom (strom, kde každý uzel má dva potomky, označovaná jako levé a pravé dítě uzlu). Binární vyhledávací stromy slouží pro vyhledávání v jednorozměrném prostoru. K-D stromy umožňují vyhledávání v N rozměrném prostoru díky tomu, že v jednotlivých uzlech stromu můžeme dělit vždy podle určité osy (viz Obr. 2.1.5) a tyto osy střídat.



Obrázek 2.1.5: Vizualizace K-D Stromu v E^3 [2]

Pro jednodušší představu je uveden jen případ dvourozměrného prostoru, kde se prostor dělí v ose x a v ose y podle vhodnosti. Příklad stromu je na obrázku 2.1.6. Na tomto obrázku je vidět, že pro dělení se vybírá osa podle umístění objektů. Na obrázku jsou objekty stejně umístěny jako na obr. 2.1.4, aby byl vidět rozdíl s nepravidelnou mřížkou.



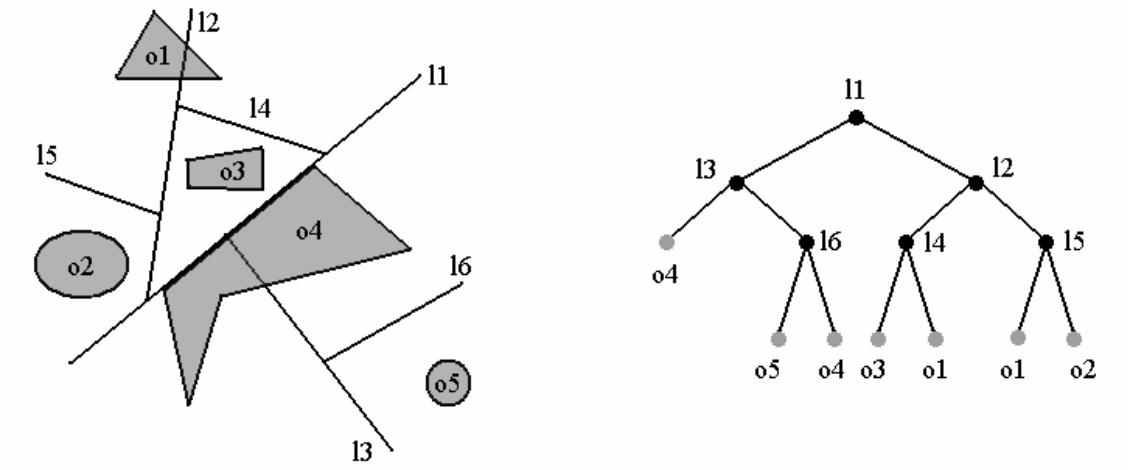
Obrázek 2.1.6: Dělení K-D stromu podle objektů scény

Strom pak v rodičovských uzlech ukládá pouze informace o dělící rovině a v konečných listech je pak uložen seznam objektů, které buňce náleží. Pokud objekt přesahuje přes dělící roviny, musí být obsažen ve více listech. Jemnost dělení může být dána například počtem objektů nebo pevnou hloubkou stromu. Hloubka stromu musí být v každém případě omezena.

Výběr objektů pro detekci kolize pak probíhá velice jednoduše. Projdou se všechny listy, které obsahují zkoumaný objekt, a objekty obsažené v těchto listech mohou s objektem kolidovat.

2.1.4 BSP Strom

BSP strom (BSP - Binary Space Partitioning) je jednou z nejčastěji používaných metod. Metoda je opět založena na binárním stromě. Oproti předchozím metodám se však může daleko lépe přizpůsobit skutečnému rozložení a tvaru objektů ve scéně díky způsobu dělení prostoru. Prostor se v této metodě rozdělí rovinou na 2 konvexní oblasti. Rovina v tomto případě nemusí být rovnoběžná s osami. Poloha dělící roviny se určí v závislosti na poloze objektů. Může se využít i hran objektů, podle kterých je možno dělicí roviny určit, jak je ukázáno na obr. 2.1.7. Po rozdělení prostoru dostaneme dvě skupiny objektů ve dvou buňkách. V závislosti na počtu objektů v buňce nebo dalších parametrech se buňka dělí další rovinou stejným způsobem. Většinou se prostor dělí, dokud není v buňce pouze jeden objekt, nebo dokud není dosáhnuto maximální hloubky stromu. Výsledné rozdělení celého prostoru pak může vypadat, jako je naznačeno na obr. 2.1.7.



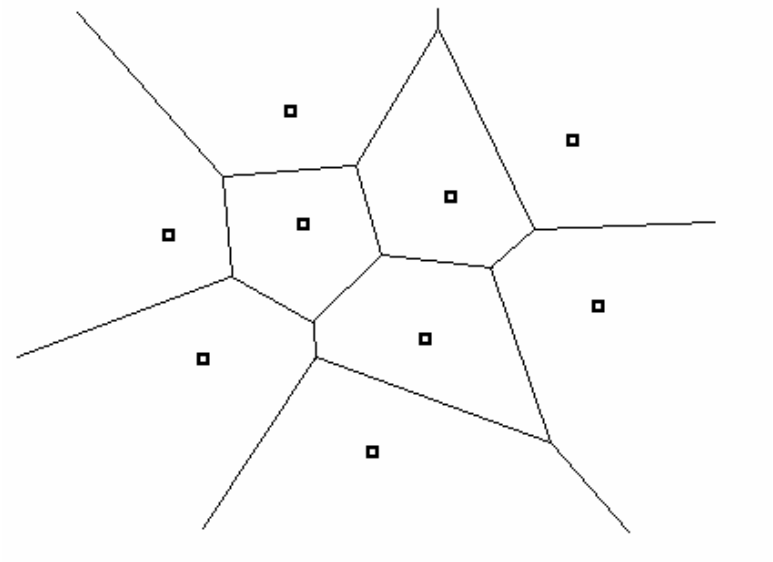
Obrázek 2.1.7: Rozdělení prostrou BSP stromem v závislosti na umístění objektů [11]

Strom je většinou téměř vyvážený, proto procházení stromem je velice rychlé, a tím je rychlé i určení objektů k aktualizaci nebo detekci. Výběr objektů, které mohou s daným objektem kolidovat, je tedy velmi rychlý. Detekce kolize je zrychlena i tím, že do dalších testů jde malý počet objektů, neboť v listech se nachází malý počet objektů, které mohou kolidovat.

2.1.5 Voronoiův diagram

Voronoiův diagram v trojrozměrném prostoru je množina všech buněk vytvořených kolem generujících bodů. Buňku Voronoiova diagramu lze pak definovat jako všechny body prostoru, jejichž eukleidovská vzdálenost od generujícího bodu příslušné buňky je menší než vzdálenost od jakéhokoli jiného generujícího bodu. Body, které mají stejnou vzdálenost od více generujících bodů, tvoří hranice Voronoiova diagramu.
Voronoiův diagram má následující vlastnosti:

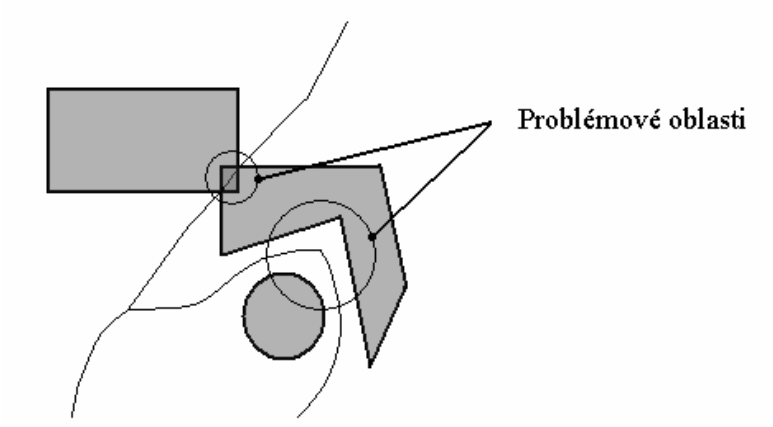
- Každá buňka je konvexní, v E^3 se jedná tedy o konvexní polyhedron.
- V každé buňce se nachází pouze jeden generující bod.
- Oblasti bodů ležící na krajích diagramu nemusí být uzavřené.
- Hrana diagramu je geometrické místo, ke kterému má stejnou vzdálenost d bodů.
- Pokud žádné d+2 body neleží na kružnici, mají uzly stupeň d+1.
- Uzel diagramu je geometrické místo, ke kterému má stejnou vzdálenost nejméně d+1 bodů.
- Je-li p_i nejbližší soused p_j, mají p_i a p_j společnou hranu ve Voronoiově diagramu.



Obrázek 2.1.8: Příklad Voronoivoa diagramu pro množinu generujících bodů

Pro jednodušší představu je na obrázku 2.1.8 zobrazen diagram v E^2. Jak je z obrázku patrné, hranice buňky určí všechny sousedy. Toho se dá při detekci kolize využít. Známe všechny sousedy, a tedy stačí provést test kolize s těmito objekty. Z hlediska výběru objektů, je tato metoda jedna z nejefektivnějších. Nevýhodou je velká časová náročnost vytvoření Voronoiova diagramu v E^3. Při odebrání objektu nebo posunu objektu je nutno buňky diagramu přizpůsobovat, což je také časově náročné.

Zatím jsme mluvili jen o diagramu pro generující body. V případě objektů nastávají další problémy. Pokud jsou objekty nekonvexní, mohou se překrývat jejich konvexní obálky, i když objekty samy se nepřekrývají, viz obr. 2.1.9. V tomto případě už neplatí, že jsou buňky diagramu konvexní. Dalším problémem, kterým bychom se museli zabývat, je průnik dvou objektů a tedy nejasnost hranice mezi těmito objekty, viz opět obr. 2.1.9.

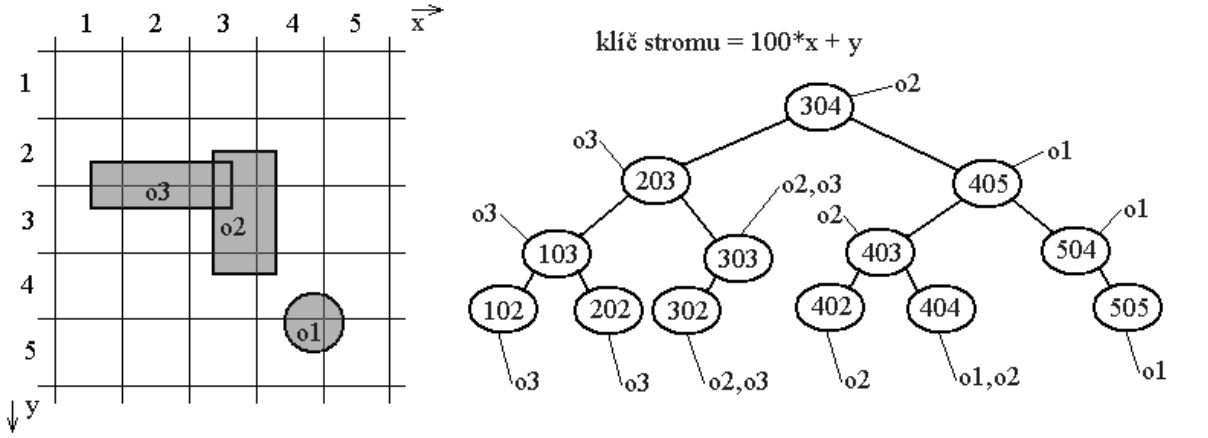


Obrázek 2.1.9: Problémové oblasti Voronoiova diagramu pro tělesa

2.1.6 Prostorový strom (The Regular Spatial Tree)

Tato metoda je modifikací pravidelné mřížky. Prostor je rozdělen stejným způsobem jako u pravidelné mřížky. Na rozdíl od mřížky, kde se buňky ukládají v poli, jsou u této metody buňky uloženy ve stromu (detaily uložení objasníme v následujícím textu). Do tohoto stromu se však ukládají pouze neprázdné buňky. Tím se snižuje paměťová náročnost, jakou má pravidelná mřížka, ovšem za cenu, že se poněkud zvýší přístupový čas do jednotlivých buněk. To je způsobeno tím, že pravidelná mřížka do buňky přistoupí pouze přes indexy oné buňky, zatímco prostorový strom musí buňku vyhledat ve stromě. Pro co nejrychlejší přístup je použit AVL strom - binární vyvážený strom. Vyvážený strom je takový, kde pro každý uzel platí, že rozdíl hloubek jeho větví není větší než 1. Buňky se do stromu vkládají podle klíčů sestavených ze souřadnic v pravidelné mřížce. To znamená, že vezmeme indexy v pravidelné mřížce v souřadnicích x y a z a z nich sestavíme klíč pro každou buňku. Pokud použijeme pro vytvoření klíče vzoreček 2.1, může příklad uložení ve stromě vypadat jako na obrázku 2.1.10. Pro jednodušší představu je opět použita reprezentace v E^2.

$$klíč = k*x + y \qquad (2.1)$$
$$,kde\ k \geq\ maximální\ rozměr\ hrany\ mřížky$$



Obrázek 2.1.10: Příklad prostorového stromu v E^2

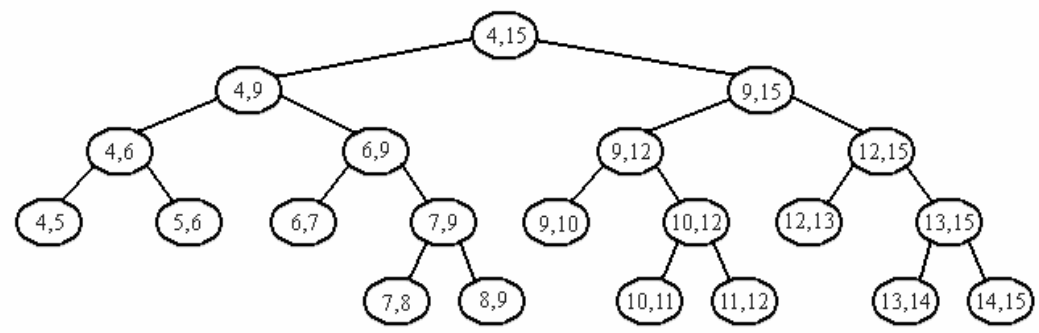
Ke každému uzlu stromu je přiřazena právě jedna buňka v N rozměrném euklidovském prostoru. V této buňce je pak neprázdný seznam objektů, které do této buňky patří, a u objektu je uložen seznam uzlů stromu podle buněk, do kterých patří.

Při detekci je test stejný jako u pravidelné mřížky. Rozdíl nastává při aktualizaci pozice objektu, kdy se strom díky odebírání prázdných buněk a přidávání neprázdných mění.

2.1.7 Segmentový strom (Segment Tree)

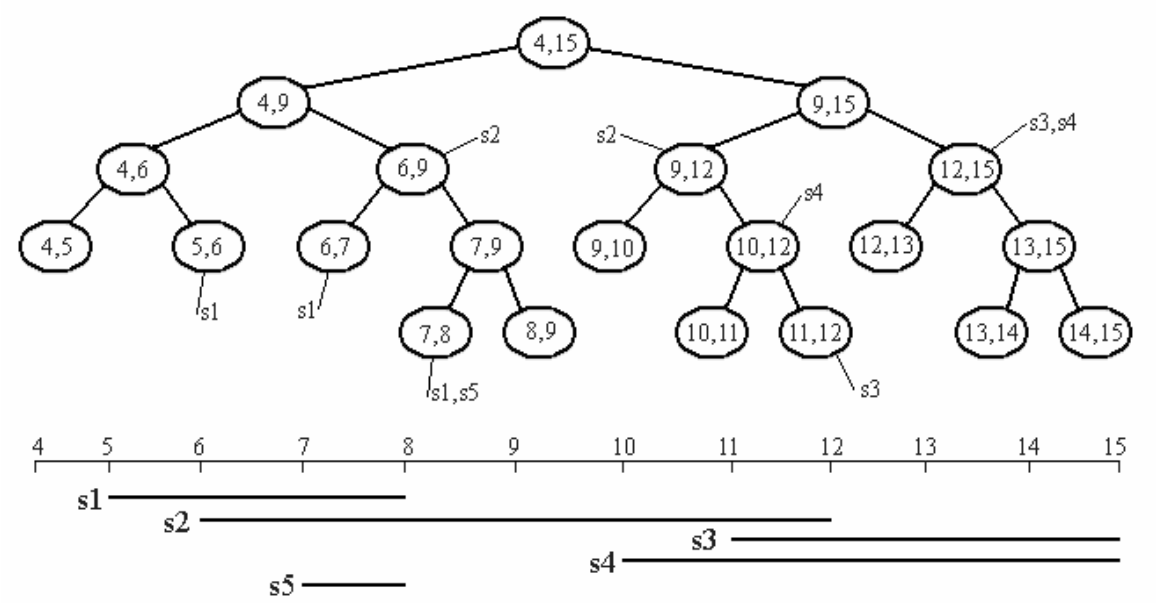
Segmentový strom je datová struktura navržená pro práci s intervaly. Intervaly většinou reprezentují úsečky, kde nás zajímá začátek a konec úsečky. Než se začne strom vytvářet, musíme znát všechny úsečky. Struktura byla navržena jako statická pro neměnnou množinu úseček. Úsečky lze vkládat do stromu i později, ale musíme předem vědět maximum a minimum intervalu celého prostoru, ve kterém se všechny úsečky nacházejí (i později přidávané), aby podle těchto parametrů byl strom vytvořen.

Segmentový strom je založen na binárním stromě. Zobrazení stromu je na obrázku 2.1.11. Každý uzel stromu reprezentuje interval. Tento interval je v uzlu uložen. Uzel může mít vždy dva potomky, kde jejich intervaly vznikají rozpůlením intervalu rodičovského uzlu. Strom se vytvoří rekurzivním způsobem velice rychle. Pokud označíme uzel V(b, e), kde b je začátek intervalu a e je konec intervalu, pak uzel V(b, e) rozdělíme na L(b, (b + e)/2) a R((b + e)/2,e) jak je vidět na obr. 2.1.11. Tímto způsobem pokračujeme u všech takto nově vzniklých uzlů, dokud platí, že e – b > 1. Po skončení tohoto kroku máme vytvořen segmentový strom, do kterého se mohou přiřadit úsečky.



Obrázek 2.1.11: Segmentový strom [6]

Přiřazení úseček je velmi jednoduché. Začne se od kořene stromu a v každém uzlu se zkontroluje, zda interval uzlu (b, e) je obsažen nebo se rovná intervalu úsečky (p_b, p_e). Najdou se tedy takové uzly, kde platí $p_b \leq b$ a zároveň $p_e \geq e$. Pokud se takové uzly najdou, ukončí se vyhledávání v aktuálním podstromu (tj. nepokračujeme synovskými uzly aktuálního uzlu) a pokračuje se dalšími, dokud není přiřazen celý interval úsečky (p_b, p_e).
Ukázka přiřazení úsečky ve stromě je na obrázku 2.1.12.



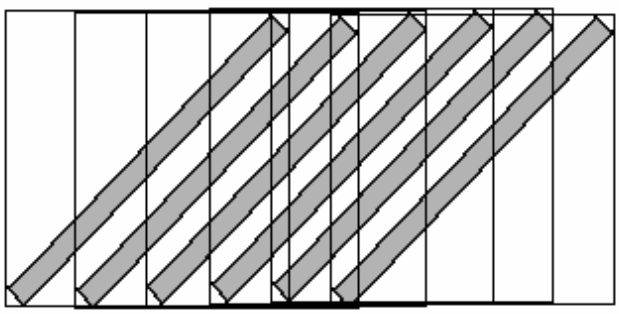
Obrázek 2.1.12: Přiřazení úseček v segmentovém stromě [6]

Odebrání ze stromu je podobné přidávání. Prochází se strom opět podle intervalu odebírané úsečky a v uzlech, kterým náleží, se odebere.

Segmentový strom je univerzální datová struktura. Jde použít v mnoha aplikacích. Například při vyhledání, na jakých úsečkách leží bod, kdy stačí projít větev stromu do listu odpovídajícímu bodu a zaznamenat cestou všechny úsečky, na které narazíme.

2.1.8 Množina segmentových stromů (The Set of Segment Trees)

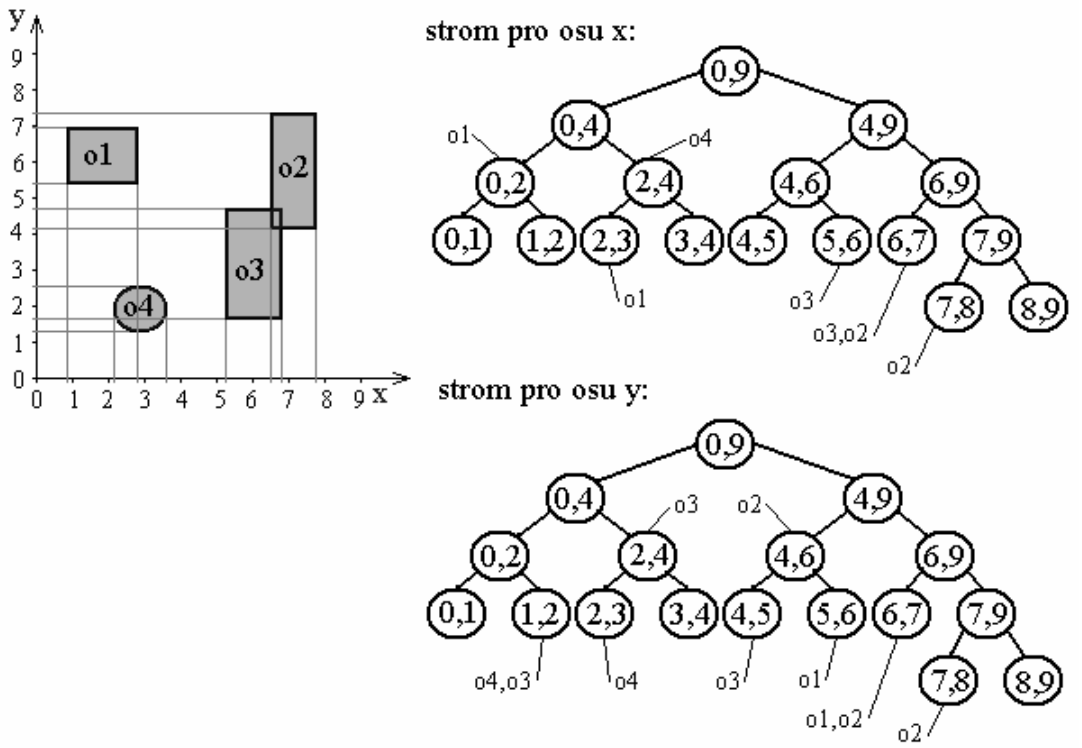
Tato metoda využívá osově zarovnaných obalových boxů (AABB boxy, kapitola 2.2.1). Výsledkem je seznam objektů, jejichž AABB boxy se protínají s AABB boxem zkoumaného objektu. Proto tato metoda může být použita pro scény, kde nemáme informaci o umístění objektů. Nevadí nám případ, kdy se objekty nashromáždí v jedné části scény. Problém ovšem může nastat pro určité tvary a umístění objektů, z důvodu, že AABB boxy neobepínají tělesa příliš těsně. Příklad nevhodné scény je na obr. 2.1.13.



Obrázek 2.1.13: Nevhodná scéna pro množinu segmentových stromů

Metoda si uchovává množinu segmentových stromů. Pro trojrozměrný prostor máme tři segmentové stromy, pro každou osu jeden. Intervaly, podle kterých budou stromy tvořeny, získáme projekcí AABB boxů jednotlivých objektů do příslušné osy.

Ze stromů nyní jednoduchým průchodem zjistíme, zda se v nějakých osách AABB boxy překrývají nebo ne. Příklad přiřazení objektu do množiny stromů je ukázán na obr. 2.1.14.



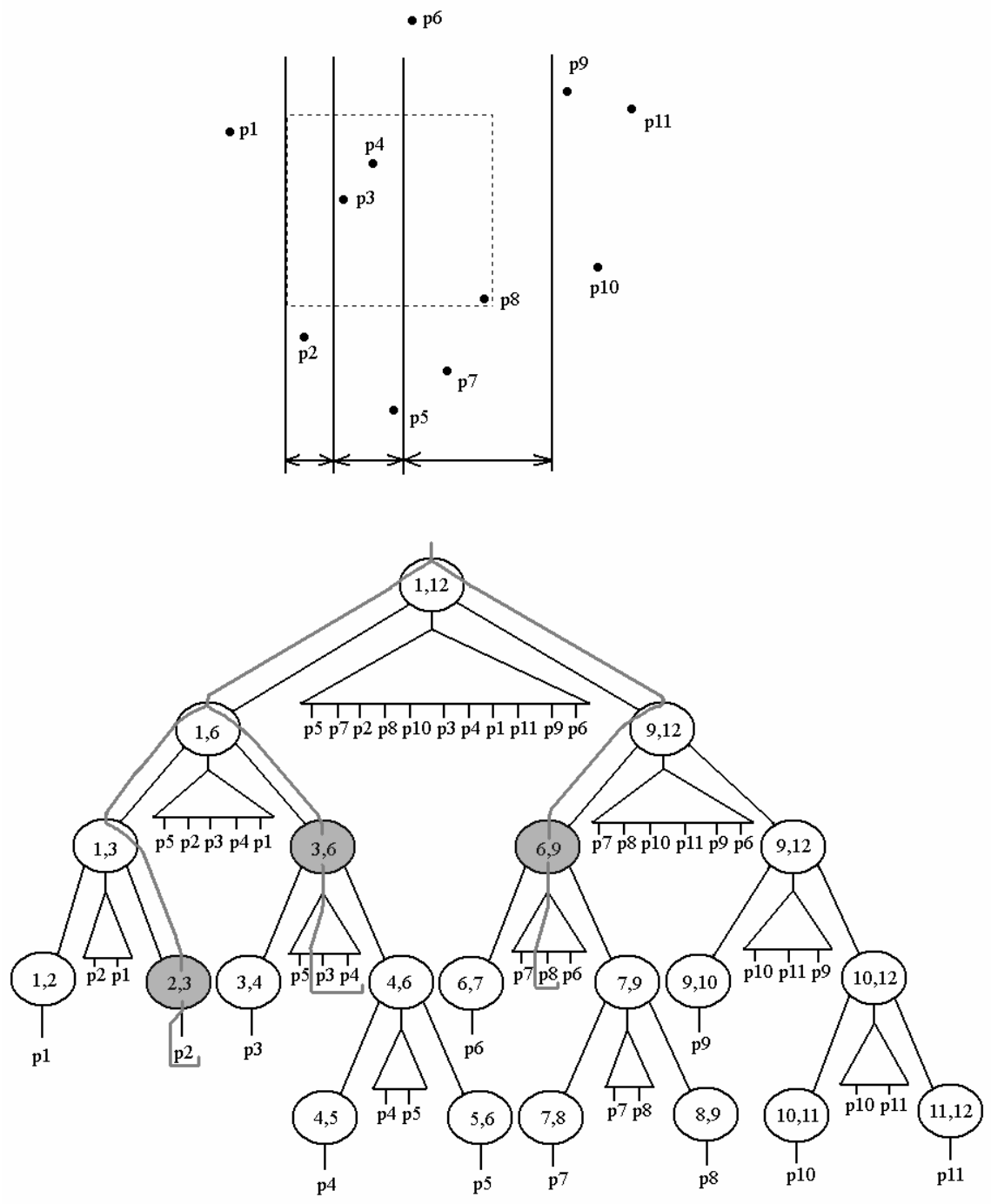
Obrázek 2.1.14: Přiřazení objektu v množině segmentových stromů (pro jednoduchost v E^2)

Při vytváření stromů musíme znát velikost scény, ve které se nacházejí objekty. Pokud pak při pohybu objektu se tento objekt dostane mimo tuto scénu, tak do stromu nemůže být přidán a je tedy vyřazen z detekce kolize. Objekty, které pak mohou kolidovat, získáme tak, že zjistíme, které AABB boxy objektů kolidují ve všech třech stromech s AABB boxem zkoumaného objektu.

2.1.9 Intervalový strom (Range Tree)

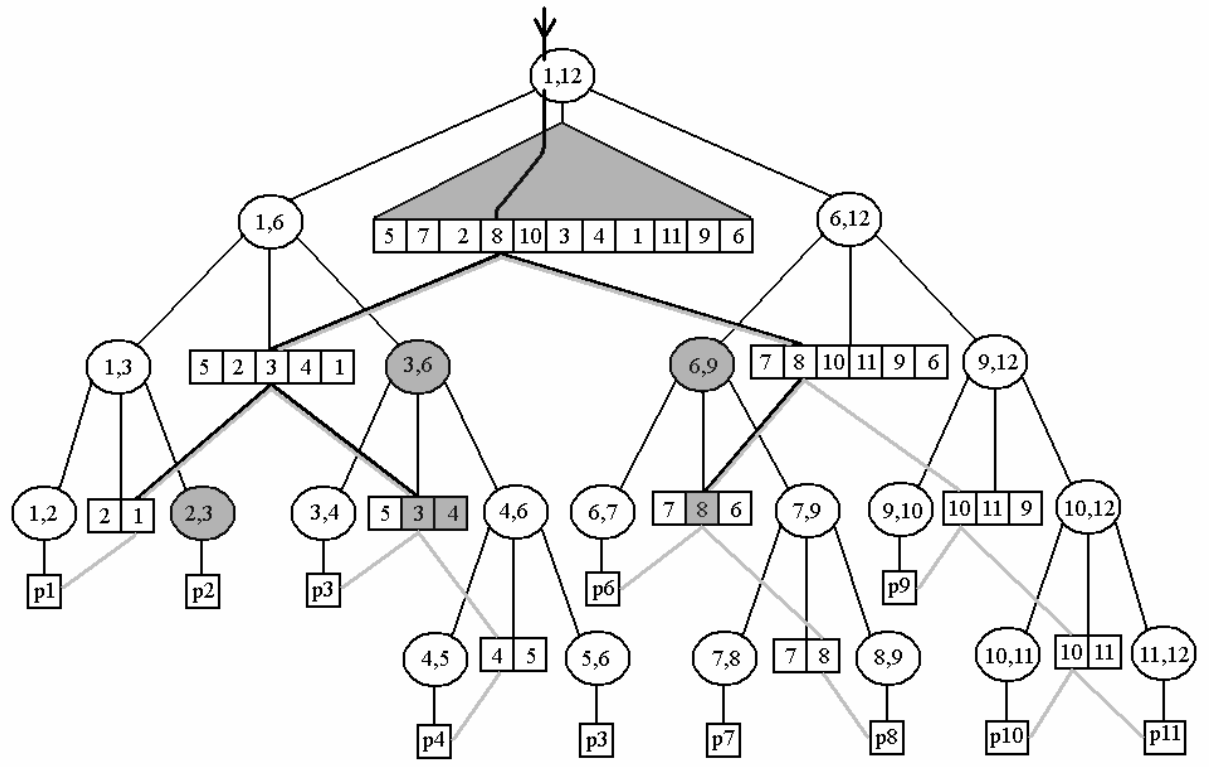
Tento strom je spojením segmentového stromu a binárního vyhledávacího stromu. Používá se pro vyhledávání ve vícerozměrném prostoru. Většinou se vyhledávají body, zda se nalézají v určitém intervalu. Jako základ je použit segmentový strom, který zaznamenává rozložení bodů v jedné ose. Pozice bodů v dalších osách jsou zaznamenány pomocí binárních vyhledávacích stromů, které jsou uloženy v každém uzlu stromu. Pro sestavení binárního stromu se používají pouze ty body, které uzlu náleží podle intervalu segmentového stromu. Příklad vytvořeného stromu je na obrázku 2.1.15.

Při vyhledávání čtvercové oblasti v prostoru nalezneme nejprve interval v ose odpovídající segmentovému stromu. Vyhledávání v segmentovém stromě je v kap. 2.1.7. Na obrázku 2.1.15 je prohledání naznačeno čarou. Jako výsledek dostaneme barevně označené uzly stromu. Druhým krokem je prohledání stromů v těchto uzlech, tím prohledáme druhou osu vyhledávané oblasti a určíme, které body do oblasti patří a které ne.



Obrázek 2.1.15: Vyhledání intervalu v intervalovém stromě [6]

Pro urychlení vyhledávání v binárních vyhledávacích stromech se intervalový strom upravuje způsobem, jaký je naznačený na obr. 2.1.16. Úprava spočívá ve změně všech stromů, kromě hlavního uzlu, na seznamy ukazatelů. Vyhledávání pak probíhá pouze v hlavním uzlu a na seznamy v ostatních uzlech se přechází pomocí ukazatelů, čímž se odstraní vyhledávání objektu v ostatních uzlech.



Obrázek 2.1.16: Intervalový strom s úpravami binárních stromů [6]

2.2 Obalová tělesa

Objekty mají často dosti složitý tvar, proto se používají tzv. obalová tělesa, která více či méně dobře aproximují složitý povrch objektu. Obalových těles je více druhů a pro různé objekty mohou být vhodná jiná obalová tělesa. Pro zpřesnění obalu se proto používají stromy složené z těchto obalových těles, které objekt aproximují daleko lépe. Pokud ovšem víme, jaké objekty se budou ve scéně nacházet, můžeme tomu přizpůsobit i obalová tělesa.

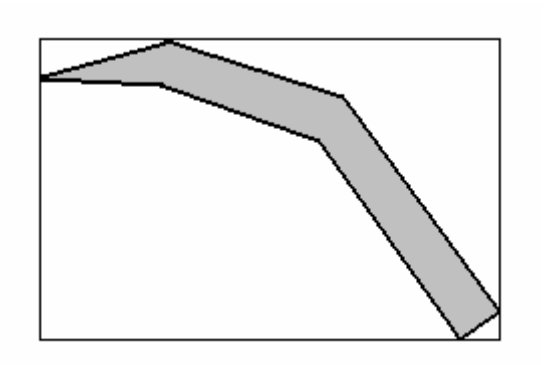
Při detekci kolizí se dají tyto metody dobře využít pro vyloučení objektů, které spolu nemohou kolidovat, pouze testem obalových těles a nemusí se testovat přímo objekty. Čím přesnější budou obalová tělesa, tím více můžeme vyloučit objektů. Ale naopak čím budou obalová tělesa složitější, tím více času spotřebujeme na test, zda obalová tělesa kolidují.

2.2.1 Osově zarovnaný box

Osově zarovnané boxy (AABB - axis-aligned bounding box) jsou vhodné pro osově zarovnaná tělesa, jako například skříň, stěna, budova apod. Tato tělesa obepíná velice dobře, ale pokud o tělesu nic nevíme, není většinou vhodné tyto boxy používat. Pokud se navíc jedná o pohybující se tělesa s rotací, musí se AABB box pokaždé přepočítávat.

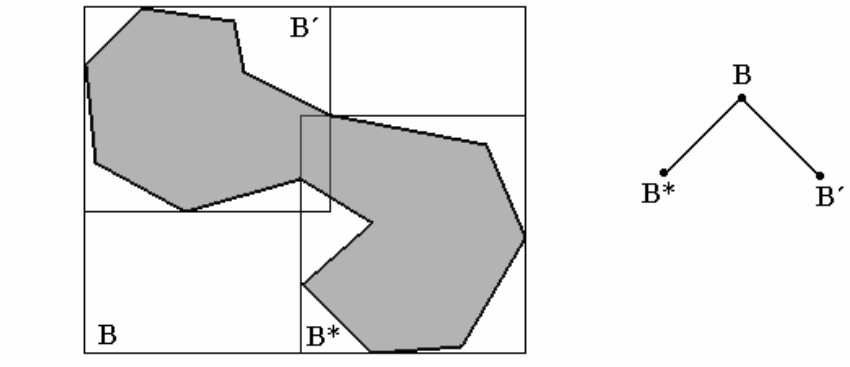
Výhodou AABB boxů je rychlost zjištění, zda spolu dva AABB boxy kolidují. Pro toto zjištění, stačí v nejhorším případě šest porovnání. Jedná se o porovnání intervalů v každé ose. Test je možno ukončit dříve, pokud zjistíme, že v některé ose se intervaly neprotínají. Pokud se protínají AABB boxy, musí se protínat intervaly ve všech osách.

Nevýhodou je že AABB box často obepíná objekt velice volně, jak je vidět na obr. 2.2.1.



Obrázek 2.2.1: Osově zarovnaný box (AABB)

Z AABB boxů lze rovněž vytvářet stromy. AABB stromy obepínají objekt lépe než samotný AABB box, ale stále dosti záleží na druhu objektu. Jak může například takový AABB strom vypadat je na obr. 2.2.2. Tyto stromy však taky neobepínají těleso příliš dobře, proto se o nich nebudu rozepisovat a zaměřím se na stromy z vhodnějších obalových těles.



Obrázek 2.2.2: AABB strom [2]

2.2.2 Obalová koule

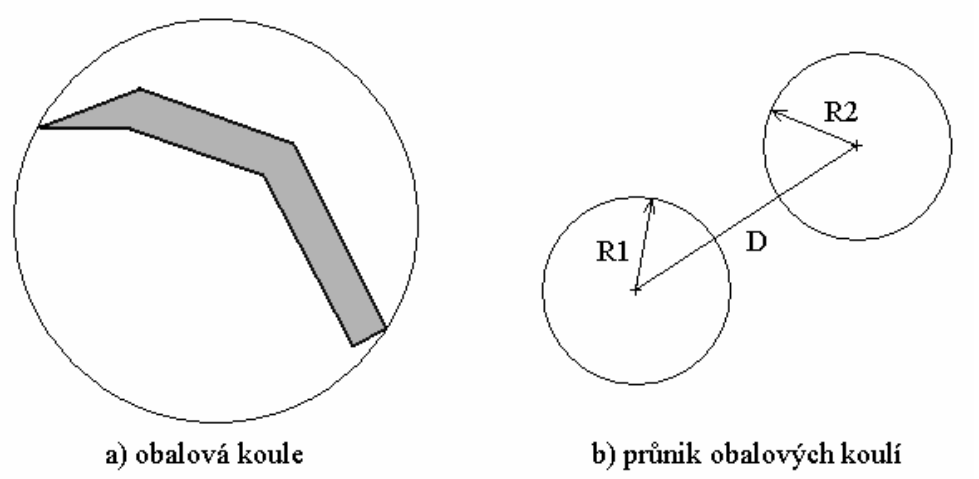
Dalším základním obalovým tělesem je obalová koule (bounding sphere). Jedná se o nejjednodušší obalové těleso. Obalová koule je velmi podobná AABB boxu. Opět obepíná těleso velmi volně, jak je vidět na obr. 2.2.3 a). Využití má ve scénách, kde jsou objekty dále od sebe a test obalových koulí tyto objekty vyloučí z dalších testů.

Výhodou je velice rychlý test na průnik za pomoci vzájemné vzdálenosti objektů a poloměru obalových koulí. Test na průnik (obr. 2.2.3 b)) lze vypočítat podle vzorce 2.2 [5]:

$$\sqrt{D_x^2 + D_y^2 + D_z^2} \leq R_1 + R_2 \qquad (2.2)$$

kde: D *- je vzdálenost středů obalových koulí*

R_1, R_2 *- jsou poloměry obalových koulí*



Obrázek 2.2.3: Obalové koule [5]

Ve scénách, kde jsou objekty blízko sebe, je tato technika nevhodná díky nepřesnosti obálky vůči obepínanému objektu. Další výhodou, na rozdíl od AABB boxu, je invariance vůči rotaci. To znamená, že při rotaci objektu ve scéně není nutné obalovou kouli přepočítávat, jako tomu je u AABB boxů.

Sphere-tree

I z obalových koulí lze sestavit strom. Na rozdíl od AABB stromů, které nejsou moc používané, jsou stromy z obalových koulí používány celkem často. Příčina používání těchto stromů je v invarianci vůči rotaci a díky velice rychlému testu na průnik. Pokud použijeme strom, můžeme tím zvýšit přesnost obepínání za cenu, že budeme muset procházet stromem. Příklad stromu je uveden na obr. 2.2.4.

Při tvorbě stromu se postupně objekt obaluje koulemi. Jako základ je použita jedna obalová koule kolem celého tělesa. Pak se vypočítá polovina objektu podle vzorce 2.3 [1][11]:

$$\mu = \frac{1}{3n}\sum_{1=0}^{n}(p^i + q^i + r^i) \qquad (2.3)$$

kde: μ - *střed aktuální části objektu*
n - *počet trojúhelníků přiřazených k části objektu*
p^i, q^i, r^i - *souřadnice vrcholu i-tého trojúhelníku*

Objekt je pak podle tohoto vypočítaného středu rozdělen na dvě části a pro každou část je vypočítána nová obalová koule (tj. vypočítá se střed části a poloměr z nejvzdálenějšího vrcholu od středu). Pak se v každé části postupuje stejným způsobem, dokud nedostaneme výsledný strom. Při rozdělování části se používá hlavní osa množiny trojúhelníku, která tvoří aktuální část objektu. Na spočítání středu potřebujeme hlavní osu objektu, kterou získáme z kovarianční matice, kterou vypočítáme ze vzorce 2.4 [1][11]:

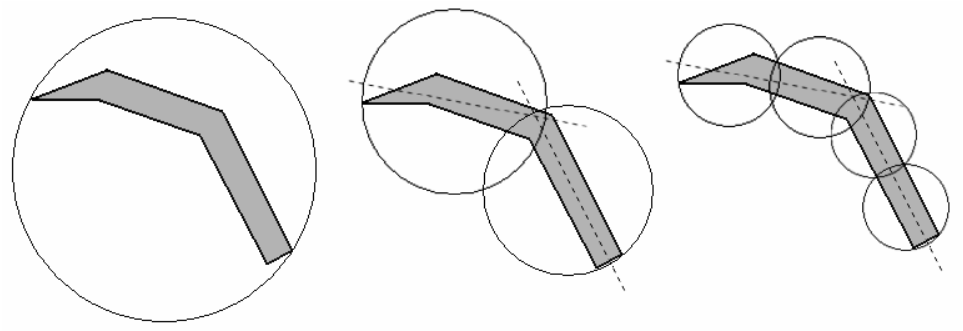
$$C_{jk} = \frac{1}{3n}\sum_{i=0}^{n}(\overline{p_j^i}\,\overline{p_k^i} + \overline{q_j^i}\,\overline{q_k^i} + \overline{r_j^i}\,\overline{r_k^i}) \qquad (2.4)$$

kde: C_{jk} - *kovarianční matice*

$\overline{p^i} = p^i - \mu$ - *vzdálenost vrcholu i-tého trojúhelníku od středu vrcholů*

$\overline{q^i}, \overline{r^i}$ - *stejně jako u p*

Hlavní osu získáme jako první řádek vypočtené matice. Z osy a ze středového bodu (vzorec 2.3) se určí dělící rovina, podle které se pak objekt dělí. Vytvoření stromu je ukázáno na obr. 2.2.4. Dělení na synovské koule lze i jiným způsobem. Například lze využít AABB boxu, který se rozdělí na oktany, a pokud oktan obsahuje trojúhelníky, vezme se to jako nová část, která se uzavře do obalové koule a tato část se může dělit rekurzivně zase pomocí oktanů AABB boxu.

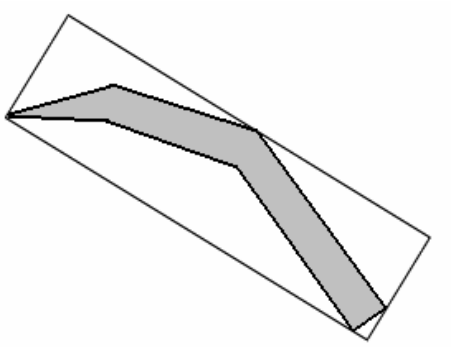


Obrázek 2.2.4: Vytváření stromu obalových koulí

Detekce kolize pak probíhá od hlavního uzlu stromu, kde se nejprve porovnají obalové koule celého objektu a postupně se sestupuje do listů stromu. Střídavě se pak sestupuje ve stromech a testují se navzájem všechny obalové koule. Test může být ukončen předčasně, pokud se nenajdou žádné kolidující koule. Jinak je test ukončen dosažením maximální hloubky stromu, kde se zjistí, zda obalová tělesa kolidují nebo ne.

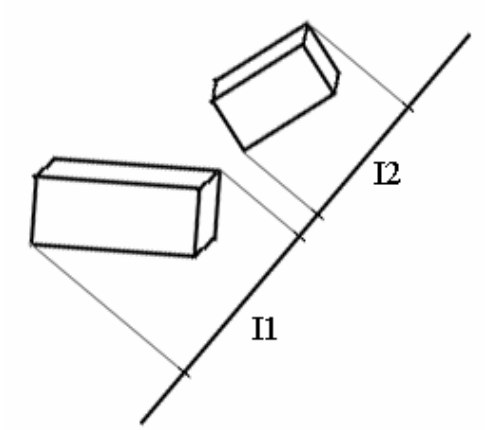
2.2.3 Orientovaný box

Orientovaný box (OBB – Oriented bounding box) je kvádr, který může být v prostoru libovolně natočený. V porovnání s předchozími obalovými tělesy má OBB box lepší aproximaci tělesa. Proto je taky velmi často používán. Příklad OBB boxu je na obr. 2.2.5. Při tvorbě OBB boxů musíme vypočítat osy objektu, podle kterých je OBB box tvořen. Osy objektu se počítají pomocí kovarianční matice, podle vzorce 2.4. Z této matice pak dostaneme osy objektu. Osy jsou v matici uloženy jako řádkové vektory od největšího (hlavní osa objektu) k nejmenšímu.



Obrázek 2.2.5: Orientovaný box (OBB)

V rychlosti testu dvou OBB boxů spočívá také výhoda této metody. K tomuto testu stačí provést v nejhorším případě patnáct jednoduchých testů. V algoritmu se využívá vlastnosti, že OBB boxy jsou konvexní tělesa. Test spočívá v promítnutí OBB boxů na přímku, jak je ukázáno na obrázku 2.2.6. Promítnutí docílíme například tak, že přímku i OBB boxy otočíme takovým způsobem, aby byla přímka totožná s osou x souřadnicového systému, a poté nalezneme největší a nejmenší x-ovou souřadnici OBB boxu. Tento výpočet provedeme pro oba testované OBB boxy a získáme na přímce dva intervaly (viz obr. 2.2.6 I1 a I2). Pokud nedojde k překrytí těchto intervalů, nemohou se oba OBB boxy protínat a test končí. Pokud došlo k překrytí, musí se provést další test. Pro úplný test kolize dvou OBB boxů v třírozměrném prostoru stačí provést test pouze s patnácti přesně určenými přímkami v prostoru. Pokud nalezneme přímku, na které se intervaly neprotínají, test končí. V nejhorším případě jde asi o 200 aritmetických operací. V praxi, díky předčasnému ukončení algoritmu, se provede asi polovina operací.

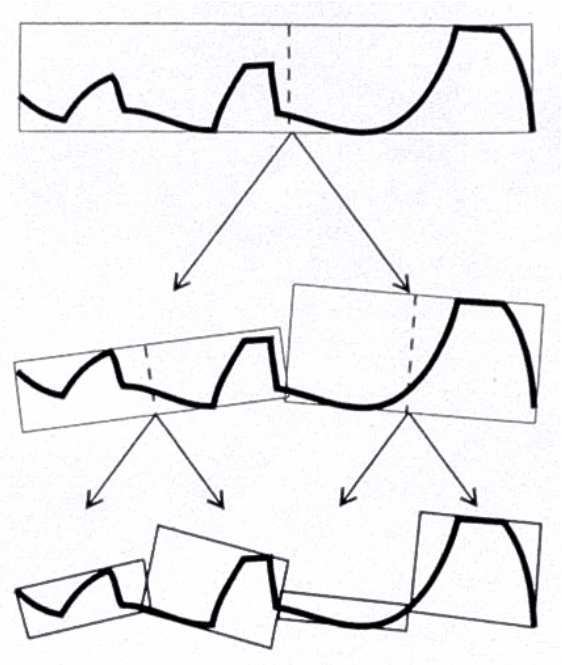


Obrázek 2.2.6: Promítnutí OBB na přímku [3]

OBB tree

Jako u předešlých těles i zde lze sestavovat z OBB boxů strom, a to velice efektivní. OBB strom je při detekci kolize často používanou metodou. Tvorba OBB stromu je podobná tvorbě stromu z obalových koulí, popsaných v kapitole 2.2.2.

Prvním krokem je vytvoření OBB boxu celého objektu. Tento OBB box je pak uložen v hlavním uzlu OBB stromu. Dále se objekt rozdělí na dvě části. K tomuto rozdělení je potřeba vypočítat střed objektu podle vzorce 2.3 a určit hlavní osu z kovarianční matice vypočítané podle vzorce 2.4. Z těchto vypočítaných hodnot rozdělíme objekt, jak je ukázáno na obr. 2.2.7. Pro tyto dvě části spočteme OBB boxy, které pak uložíme do synovských uzlů uzlu hlavního. V synovských uzlech pak provádíme dále dělení objektu stejným způsobem jako u uzlu hlavního. Dělení skončí, pokud je dosáhnuto maximální hloubky stromu nebo dosáhneme jiného daného parametru ukončení dělení. Proces vytvoření OBB stromu je naznačen na obrázku 2.2.7.



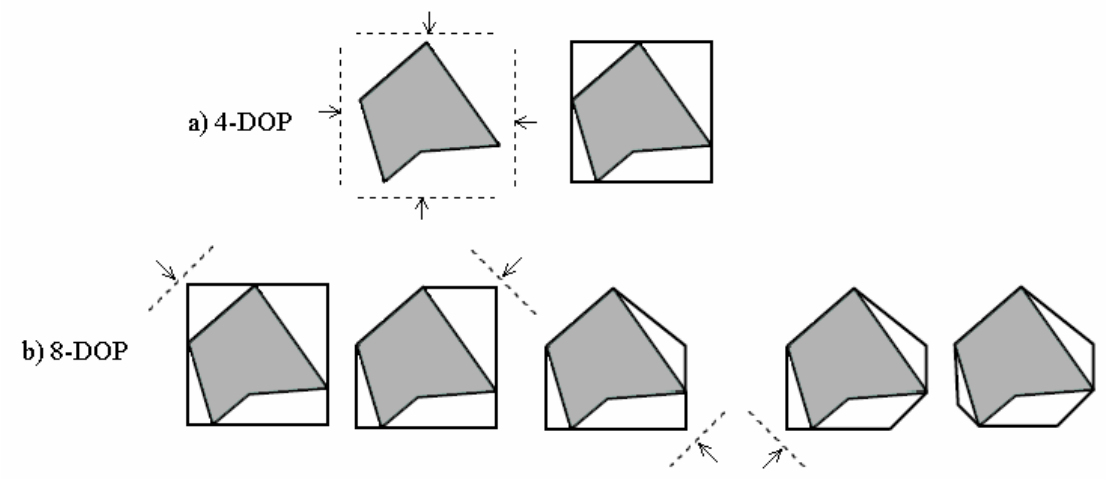
Obrázek 2.2.7: Vytváření OBB stromu [3]

Při kontrole kolize pomocí OBB stromu postupujeme od kořene stromu. Nejprve porovnáváme OBB boxy v kořenech stromů a pokud se protínají, sestoupíme o úroveň níž v prvním stromě. Poté vždy porovnáváme OBB boxy dané úrovně s OBB boxy druhého objektu a pokud narazíme na kolizi některých z nich, sestupujeme ve stromě o úroveň, pokud existují synovské uzly prvního stromu. Pokud ne, projdeme postupně i uzly druhého stromu. Test je ukončen ve chvíli, kdy nenalezneme kolidující OBB boxy nebo dokud neprojdeme celý strom.

2.2.4 K – Dop

Toto obalové těleso je používáno pro aproximaci složitějších objektů. Cílem je vytvořit takové těleso, které bude co nejlepší aproximací objektu, s co nejmenším počtem hran. Tvorbu obalového tělesa pro jednoduchost ukáži na 2D případu, 3D případ je velice podobný. Postupné vytvoření K-DOP objektu je na obrázku 2.2.8. K zde udává číslo (na obrázku K = 8, tedy 8-DOP) určující, kolik může mít obalové těleso nejvýše hran.

Tvorba tělesa začíná pro K = 4, kde se těleso obepne čtyřmi hranami rovnoběžnými s osami x a y, jak je vidět na obrázku 2.2.8 a). Dále se pokračuje v ořezávání, kdy se volí K z řady 4, 8, 16 … Ořezání pro K = 8 je vidět na obr. 2.2.8 b). Během tvorby se ořezávací přímky vždy natáčí o poloviční úhel, než jsou předešlé hrany.



Obrázek 2.2.8: Vytvoření K-DOP

Při detekci kolize u K-DOP těles postupujeme stejným způsobem jako u OBB boxů. Tedy vezmeme přímku v prostoru a promítneme na ni obalová tělesa. Volba přímek pro promítání je zde ovšem jednodušší, neboť se berou podle přímek, kterými se tvořil K-DOP. Počet porovnání je oproti OBB boxům větší, protože K-DOP mají více hran.

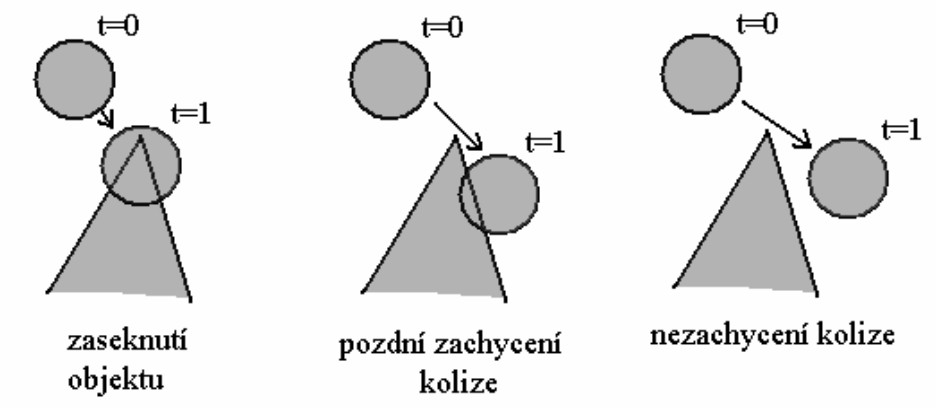
2.3 Pohyb objektu

Objekty ve scéně se z pohledu detekce kolize dělí na statické a dynamické. Statické scény, tedy scény, kde jsou všechny objekty statické, nás z pohledu detekce kolizí nezajímají. Pro nás jsou zajímavé scény dynamické. Takové scény, kde se pohybuje jeden, či více objektů. U těchto objektů je nutno kontrolovat, zda nedochází ke kolizi s ostatními objekty, ať už statickými nebo dynamickými. Podle typu scén je vhodné zvolit některou z uvedených urychlovacích technik rozdělení scény (kap. 2.1) a vhodná obalová tělesa (kap. 2.2). Při malém počtu objektů někdy není nutno rozdělení scény používat. Ovšem pro větší počet předmětů, je nutné pro zachování plynulosti aplikace použít správné urychlovací techniky.

2.3.1 Diskrétní čas

Při animaci a pohybu objektů je čas reprezentován diskrétní veličinou. Tedy plynutí času je v aplikaci nahrazeno časovými body. Mezi těmito body se musí celá scéna aktualizovat a vykreslit. Čím rychlejší je aktualizace scény (součástí aktualizace je i detekce kolize), tím plynulejší je běh času a animace objektů. Snahou by měla být co nejplynulejší animace. V určitých případech může být vysoká rychlost animace úmyslně snižována, protože člověku stačí pro plynulé vnímání okolo 25 snímků za sekundu a můžeme tak ušetřit procesorový čas.

Díky diskrétnímu času se objekty nepohybují plynule, ale „skáčou“ z místa na místo. Z tohoto důvodu nastávají problémy, kdy se objekt může dostat do jiného objektu a zaseknout se nebo ho může úplně přeskočit. Také se může kolize zachytit pozdě, a detekce může označit jiné trojúhelníky, než by měla. V takovém případě pak objekt reaguje neočekávaným způsobem. Několik takových situací je ukázáno na obr. 2.3.1.



Obrázek 2.3.1: Problémy při pohybu v diskrétním čase

Problémy se projeví hlavně v případě velkých skoků objektu. Velké skoky mohou být zapříčiněny buď velkou rychlostí objektu nebo velkými časovými úseky mezi jednotlivými snímky.

2.3.2 Detekce kolize

Detekce kolize se provádí pro všechny dynamické objekty scény. Snahou je co nejrychlejší test na detekci kolize, neboť se provádí při každém překreslení scény. Snahou je také, aby časové skoky byly co nejmenší, aby nedocházelo k problémům popsaným výše, a animace byly plynulé.

Pro efektivní výběr objektů, které mohou kolidovat, se používají výše uvedené techniky. Tyto techniky mají za úkol, co nejrychleji odstranit co nejvíce objektů, které nemohou se zkoumaným objektem kolidovat. Jako první test se používá některé z rozdělení prostoru popsané v kapitole 2.1, které by mělo vyloučit většinu objektů. Zbytek je pak testován pomocí obalových těles popsaných v kapitole 2.2. Pokud kolidují obalová tělesa, musí se přejít na test trojúhelník vůči trojúhelníku. Pokud zjistíme kolizi některých trojúhelníků, tělesa kolidují, a počítá se reakce na kolizi podle trojúhelníků, které kolidují.

3 Praktická část

Tato kapitola je rozdělena na tři podkapitoly. První část (kap. 3.1) popisuje řešení, které jsem zdědil po svém předchůdci J. Matouškovi a které bylo součástí [1]. V druhé části (kap. 3.2) jsou popsány metody, které jsem implementoval a přidal do aplikace. Poslední část (kap. 3.3) obsahuje testy metod popsaných v kap. 3.1 a 3.2, výsledky a jejich zhodnocení.

3.1 Referenční řešení

K dispozici jsem měl naprogramované řešení detekce kolize. Řešení bylo konstruováno pro scény s rovnoměrným rozložením objektů, kde většina objektů je statických a pouze několik dynamických.

3.1.1 Popis základních částí aplikace

Inicializace datových struktur

Načtení objektů:

Všechny objekty scény jsou postupně načteny ze souboru *.ASE (ascii formát 3D Studia Max). Po načtení každého objektu je mu přiřazena poloha pomocí polohového vektoru a souřadnice jeho bodů jsou přepočítány na relativní. Toto se dělá pro pozdější snadnou manipulaci s objektem a využití maticových transformací.

Po přepočtu souřadnic se vypočítají obalová tělesa objektu. Prvním je obalová koule objektu. Dále se vypočítá AABB box a posledním obalovým tělesem je OBB strom. Velikost OBB stromu je dána počtem vrcholů objektu.

Vytvoření pravidelné mřížky:

Velikost jedné buňky a počet buněk se počítá z průměrné velikosti objektu a maximální vzdálenosti objektu od počátku. Velikost mřížky je pak zvětšena nad velikost scény, aby kolize fungovaly, i když se objekt vzdálí do určité vzdálenosti od načtené scény.

Buňky se vytvářejí ve všech osách a s rostoucí vzdáleností mezi objekty velmi rychle narůstá jejich počet. Počet buněk je omezen (okolo 1000 buněk). Pro větší počty je tvorba mřížky časově i paměťově velmi náročná.

Po vytvoření mřížky do ní jsou přiřazeny všechny objekty s využitím AABB boxu vypočtené při tvorbě objektu. Podle těchto boxů jsou do mřížky umisťovány.

Přiřazení kontrolerů:

Kontrolery jsou přiřazeny k objektům a určují dynamické vlastnosti objektu. To znamená, že se podle nich počítá pohyb objektu a reakce na případné kolize. Tyto vlastnosti jsou načteny ze souborů *.dyn. Aplikace je vybavena celou řadou různých možností pohybu, které mohou být závislé přímo na uživateli, nebo na zadaném pohybovém vektoru, gravitaci, či jiném druhu pohybu.

Aktualizace scény

Aktualizací scény rozumíme změnu polohy dynamických objektů a tím pádem změnu datových struktur aplikace. Tento krok se provádí mezi jednotlivými vykreslovanými snímky, tudíž se snažíme, aby čas aktualizace byl co nejmenší. Délka času ovlivňuje i délku skoku objektu, proto by měl být co nejmenší. Při aktualizaci se musí upravit přiřazení v pravidelné mřížce a určité parametry objektu.

Prvním krokem aktualizace je vyjmutí objektu z mřížky. Poté se provede pohyb objektu a spočítají se parametry, které se změnily, jako například AABB box. Posledním krokem je objekt znovu začlenit do mřížky. Po této proceduře nastává kontrola na detekci kolize.

Detekce kolize

Detektor je pro nás nejdůležitější část aplikace, kterou budeme upravovat. Testování probíhá ve čtyřech fázích a výstup každé fáze je seznam objektů, které mohou s testovaným objektem kolidovat. Výstup z poslední části jsou objekty, které kolidují.

Provádějí se následující testy:

1) **Pravidelná mřížka**

   První fází testování je pravidelná rovnoměrná mřížka, kde se vybírají objekty z buněk, které zabírá právě testovaný objekt. Prochází se všechny objekty z těchto buněk, zjistí se, zda už nebyly testovány a pokud ne, tak se přesouvají do dalších testů. U objektu se ukládá stav, zda byl testován, aby se později vyloučilo vícenásobné testování jednoho objektu, ke kterému by mohlo dojít vlivem uložení objektů ve více buňkách mřížky.

   Pokud jsou objekty ve scéně rovnoměrně rozděleny, tak jich je většina tímto krokem vyloučena.

2) **Obalové koule**

   Druhým testem je test průniku obalových koulí. V případě, kdy jsou objekty vůči mřížce malé a jedna buňka mřížky zahrnuje více takovýchto objektů, tento test rychle vyloučí objekty, které jsou od sebe daleko a nemohou tedy kolidovat.

3) **OBB Strom**

   Ve třetí fázi se testují objekty na průnik OBB boxů v OBB stromu. Tyto boxy obepínají objekt relativně těsně a tudíž objekty, které jsou na výstupu tohoto testu, pravděpodobně kolidují se zkoumaným objektem.

   U tohoto testu je dále využito toho, že po testu víme, který OBB box ze stromu kolidoval. Proto můžeme do posledního testu zahrnout jen ty trojúhelníky objektu, které reprezentuje daný OBB box. Tímto se velice zrychlí poslední test především pro objekty s hodně trojúhelníky. V tom případě je objekt rozdělen na více částí v OBB stromu.

4) Vzájemný test trojúhelníků

Posledním testem je test trojúhelníků. Do tohoto testu jsou zařazeny jen určité trojúhelníky objektu. Kritéria pro výběr trojúhelníků v tomto případě jsou orientace normály a příslušnost ke kolidujícímu OBB boxu.

Po skončení posledního testu máme výsledný seznam kolidujících objektů, kterým se přiřadí reakce na kolizi, závislá na kontrolerech přiřazených objektů.

Toto řešení bylo vytvořeno pro scény s rovnoměrným rozložením objektů. Pro tento případ detekce kolize funguje rychle a dobře. V případě, kdy použijeme scénu s velkým počtem objektů, které jsou v jedné části scény, čas testu kolizí se velice zvýší. Pravidelná mřížka je pro takovéto scény nevyhovující, neboť do dalších testů projde velké množství objektů. Z toho důvodu byly vybrány a implementovány metody, které pravidelnou mřížku vylepší, nebo se rozdělení scény nahradí jinou strukturou, která bude objekty k detekci vybírat lépe.

3.2 Popis implementovaných metod

3.2.1 Prostorový strom

Tato metoda byla programována víceméně s cílem seznámit se s aplikací a zkusit do ní zakomponovat vlastní metodu. Metoda vychází z regulární mřížky, která byla již v aplikaci naprogramována. Nevýhodou pravidelné mřížky je její paměťová náročnost. Tento algoritmus je konstruován, aby tuto nevýhodu částečně odstranil.

Velikost a počet buněk, na které je prostor rozdělen, jsou použity stejné jako u regulární mřížky, u které byly tyto hodnoty testovány a optimálně zvoleny. Maximální velikost mřížky v jedné ose je tedy 10 buněk (při korekci pole se přidávají ještě 2 buňky, maximální počet buněk je tedy 12*12*12 = 1728). Přiřazení buněk ve stromu je znázorněno na obr. 3.2.2.

Inicializace:

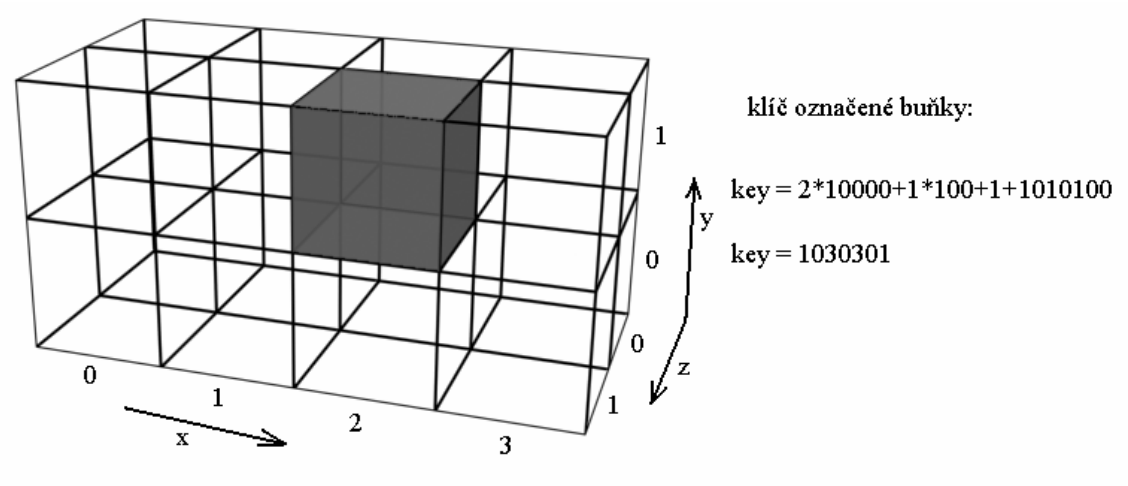
Na začátku se musí vytvořit strom a přiřadit se do něj všechny objekty. Každý načtený objekt zjistí, jaké buňky v pravidelné mřížce zaujímá a pokud ve stromě ještě nejsou přiřazeny, tak jsou vytvořeny příslušné uzly reprezentující tyto buňky. Načítaný objekt je pak uzlům přiřazen. K objektu jsou zároveň přiřazeny ukazatele na uzly stromu, do kterých objekt patří. Po přidání objektu následuje vyvážení AVL stromu, pokud je strom nevyvážený.

Klíč, podle kterého je strom tvořen a vyvažován, je složen ze souřadnic buňky, které má v regulární mřížce. Příklad je na obr. 3.2.1.

Klíč se tvoří následovně:

$$key = x*10000 + y*100 + z + 1010100 \qquad (3.1)$$

kde: *x – je x-ová souřadnice buňky v mřížce*
y – je y-ová souřadnice buňky v mřížce
z – je z-ová souřadnice buňky v mřížce



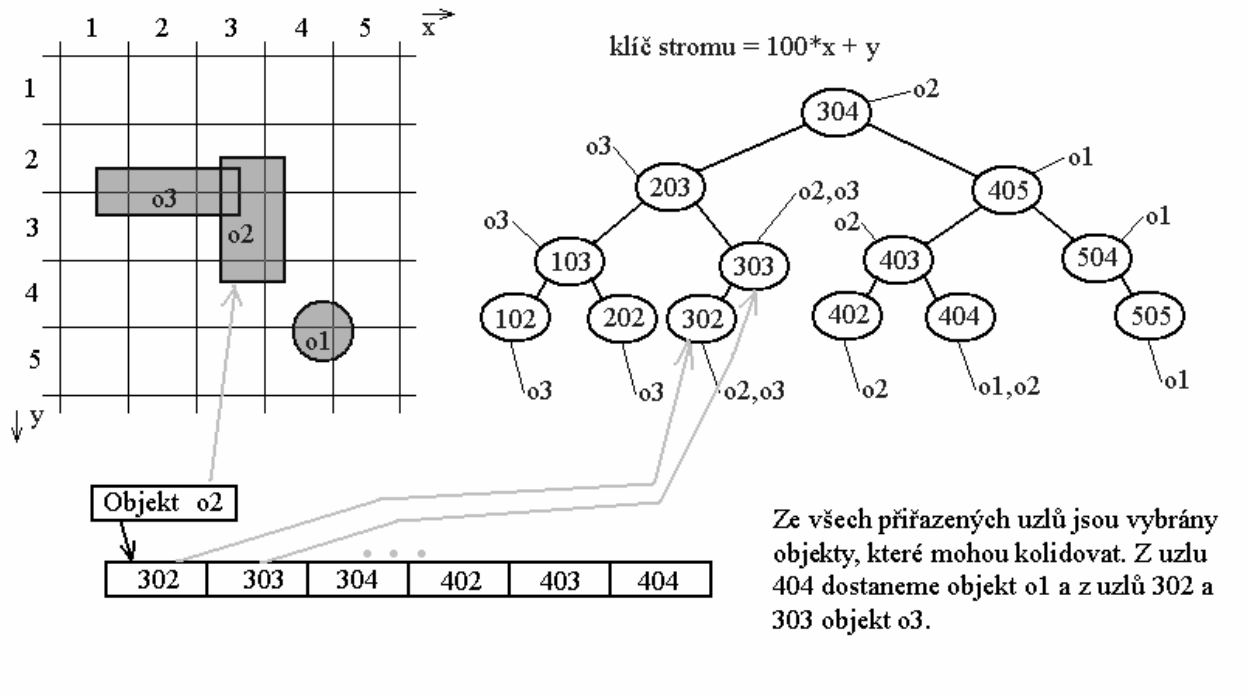
Obrázek 3.2.1: Tvorba klíče pro prostorový strom

Aktualizace scény:

Aktualizace scény probíhá velice podobně aktualizaci u regulární mřížky. Aktualizace se postupně provádí pro všechny dynamické objekty. Každý objekt se nejprve odebere ze stromu. Pak se změní poloha objektu podle kontrolerů (kontroler - viz kap. 3.1.1) jemu přiřazených a znovu se přidá do stromu. Na konci procesu se projdou uzly stromu, kde byl původně objekt umístěn a pokud zůstal některý uzel prázdný, je ze stromu odebrán. Stále musíme kontrolovat, zda je strom vyvážený a v případě nutnosti ho vyvážit.

Detekce kolize:

U objektu, pro který se zjišťuje kolize, se prochází seznam uzlů stromu, které objekt zabírá. V každém uzlu se projde seznam objektů, které patří právě do buňky, kterou uzel reprezentuje (viz obr. 3.2.2). Tyto objekty jsou ty, se kterými může nastat kolize a jsou dále testovány dalšími metodami pro zjištění kolize. Každý otestovaný objekt se označí, aby nedocházelo k vícenásobnému testování.



Obrázek 3.2.2: Výběr objektů z prostorového stromu

3.2.2 Sada segmentových stromů

Pravidelná mřížka i prostorový strom jsou vhodné pro scény s rovnoměrným rozložením objektů. Pro scény se shlukem objektů jsou však nevhodné, neboť pak nemusí skoro žádné objekty vyloučit do dalších fází testování na kolizi. Z tohoto důvodu byla zvolena metoda sady segmentových stromů. Metoda vybírá takové objekty, jejichž AABB boxy se protínají. Pokud se tedy nebude jednat o nevhodnou scénu (viz obr. 2.1.13), tak by výběr objektů do dalších fází měl být velmi dobrý.

Jedná se o první krok detekce kolize, kdy se vybírají objekty, které spolu mohou kolidovat. Podle AABB boxu (AABB boxy objektů jsou vypočítány při načtení objektu) je objekt přiřazen do stromu.

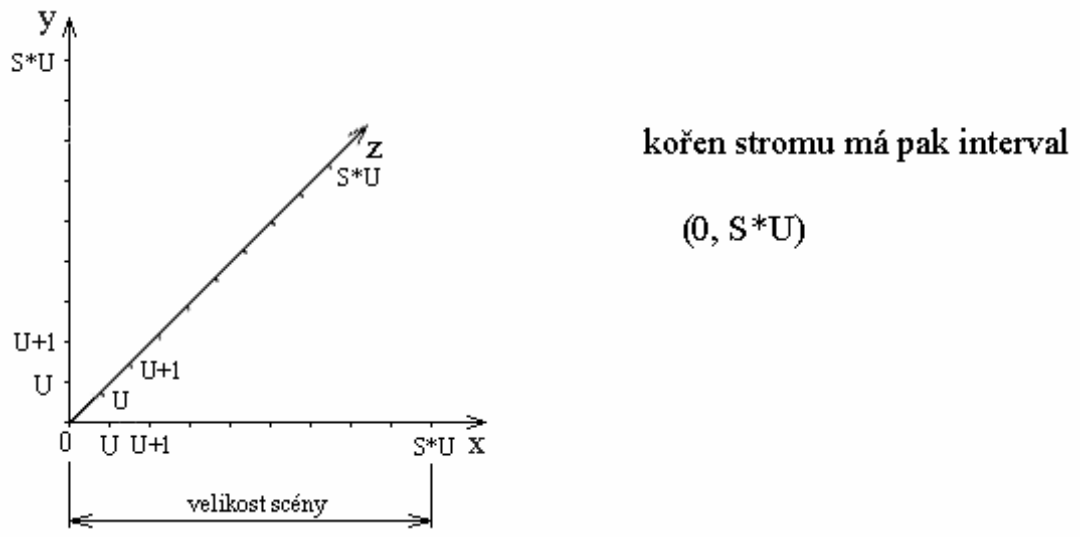
U stromu je nutno zvolit maximální hloubku stromu, aby pro scény s velkými rozměry nezabíral příliš mnoho místa, a naopak upravit malé rozměry scény, aby byl strom využit. Než se strom začne vytvářet, vypočítá se jednotka soustavy, ve které bude strom pracovat. Všechny hodnoty v uzlech pak budou dělitelné touto hodnotou (tzv. jednotka stromu). Běžně používaná desítková soustava má jednotku 1. Pro strom si zde určíme vlastní soustavu s vlastní jednotkou. Například strom v kap. 2.1.7 používá desítkovou soustavu (kde jednotka u = 1), a tedy konečné listy stromu mají interval (x, x+1). V našem případě používáme vlastní soustavu s jednotkou U vypočtenou podle vzorce 3.2 a listy mají interval (x*U, x*U+U).

Pro výpočet jednotky soustavy je určující velikost scény a na kolik částí ji budeme dělit. Vzorec pro výpočet je následující:

$$U = \text{velikost scény} / S \qquad (3.2)$$

Kde: *U ... jednotka stromu*
S ... počet kusů, na které se rozsah scény rozdělí (testována v tab. 2.1)

pro lepší představu jsou proměnné na obrázku 3.2.3

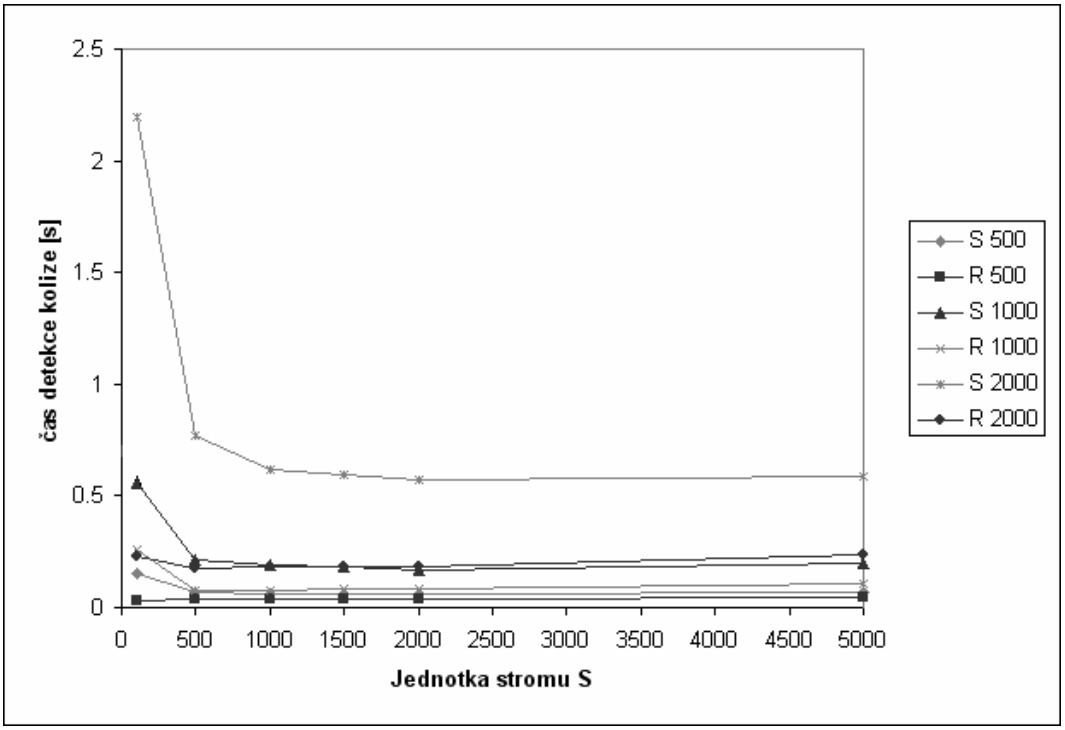


Obrázek 3.2.3: Výpočet jednotky stromu

Pro zjištění, na kolik částí se má velikost scény rozdělit, bylo testováno několik scén s různými hodnotami velikosti segmentového stromu.

Typ scény	Jednotka stromu S					
	100	500	1000	1500	2000	5000
C 500	0.150000	0.065625	0.062500	0.062500	0.060938	0.065625
R 500	0.032813	0.034375	0.037500	0.039063	0.039063	0.046875
C 1000	0.557813	0.214063	0.190625	0.179688	0.168750	0.195313
R 1000	0.256250	0.075000	0.078125	0.085938	0.087500	0.106250
C 2000	2.198438	0.770313	0.618750	0.592188	0.575000	0.589063
R 2000	0.228125	0.171875	0.179688	0.182813	0.182813	0.234375

Tabulka 2.1: Čas detekce [s] v závislosti na parametru S (typ scény: R – rovnoměrné rozložení objektu, C – shlukovité rozložení objektů, Číslo za písmenem znamená počet objektů ve scéně)



Graf 2.1: Čas detekce [s] v závislosti na parametru S

V tabulce 2.1 je vidět, že pro malé rozměry trvá testování scén delší dobu. Pokud rozdělím rozsah na 1000 kusů, jsou už výsledky daleko lepší a do hodnoty 2000 se snižují, ale celkem nepatrně. Pokud jsem použil rozdělení na 5000, tak už se začne projevovat časová náročnost procházení velkého stromu.

Z praktických důvodů a kvůli úspoře paměti jsem si vybral rozdělení na 1000 kusů, kde sice nejsou výsledky nejlepší, ale od vyšších hodnot se moc neliší a zmenší se paměťová náročnost.

Inicializace:

Pro každou osu se vytvoří jeden segmentový strom. Tedy nejprve hlavní kořen, jemuž se nastaví rozmezí celého prostoru, ve kterém se tělesa budou moci pohybovat a kde se bude zkoumat kolize s ostatními objekty.

Velikost zkoumaného prostoru jsem volil z maximální vzdálenosti objektu od počátku. Tato hodnota je pak zvětšena, aby se objekty mohly pohybovat do určité vzdálenosti i mimo prostor, kde se nachází načtená scéna. Vyjádření pomocí vzorce vypadá následovně:

$$D = max\ (Li) * 1.5 \qquad (3.3)$$

Kde: *D ... maximální vzdálenost od počátku*
Li ... vzdálenost objektu i od počátku

Hlavní uzel stromu v každé ose pak bude mít hodnotu (-D, +D) a mohou se do stromu přiřadit všechny objekty. Uzly stromu jsou přidávány postupně, pokud jsou potřeba. Zde se taky volí minimální jednotka stromu, která se počítá ze vzorce:

$$u = 2D / 1000 * 100 \qquad (3.4)$$

Kde: *u ... jednotka stromu*
D ... maximální vzdálenost od počátku

Pro všechny 3 stromy je jednotka stejná, neboť mají stejný rozsah kořene stromu. Objekty se pak přiřadí do příslušných uzlů stromu a k objektu se uloží reference na počáteční a koncový uzel v každém stromě. Tvorba stromu je popsána v kap.2.1.7. Rozdíl je jen v použité soustavě.

Aktualizace scény:

Aktualizace scény probíhá jako u předešlých algoritmů. Tedy objekt je nejprve ze stromu odebrán, je aktualizována jeho poloha a pak je zpátky do stromu přidán. Rozdíl od prostorového stromu je v tom, že zde se prázdné uzly stromu neruší.

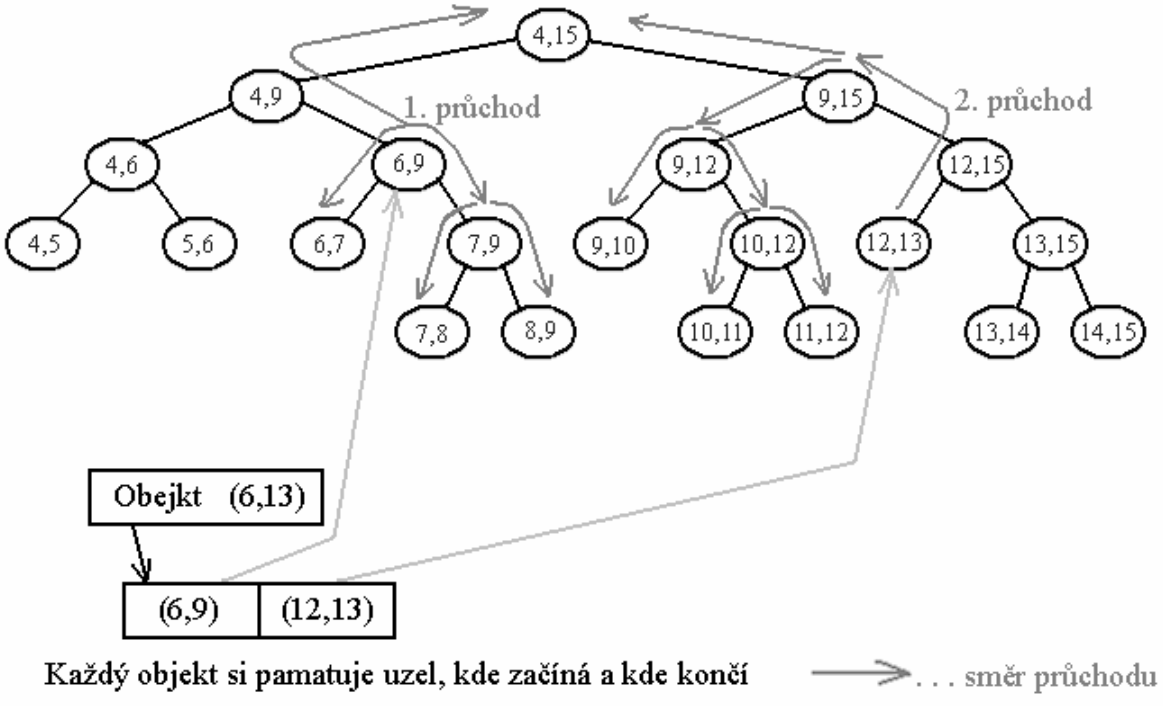
Detekce kolize:

Jedná se o první krok detekce, kdy výsledkem je seznam objektů, které mohou kolidovat s objektem, který zkoumáme. Zkoumaný objekt má u sebe uloženy dva uzly do každého stromu, které určují počáteční a koncový uzel (tedy celkem šest uzlů), kde je objekt uložen (viz obr. 3.2.4). Průchodem stromu získáme z uzlů, kterými procházíme objekty, se kterými zkoumaný objekt může kolidovat. Průchod stromem je ukázán na obrázku 3.2.4 a popsán následujícím pseudoalgoritmem:

```
1. průchod
aktUzel = počáteční uzel ve stromě (získaný od objektu)
projdi všechny potomky uzlu „aktUzel“
aktUzel = rodičovský uzel uzlu „aktUzel“
while  (aktUzel!= hlavnímu uzlu stromu){
        if (maximum intervalu „aktUzel“ < maximum intervalu konečného uzlu){
                projdi celý pravý podstrom uzlu „aktUzel“
        }
        aktUzel = rodičovský uzel
}

2. průchod
aktUzel = konečný uzel ve stromě (získaný od objektu)
projdi všechny potomky uzlu „aktUzel“
aktUzel = rodičovský uzel  uzlu „aktUzel“
while  (aktUzel!= hlavnímu uzlu stromu){
        if (minimum intervalu „aktUzel“ > minimum intervalu počátečního uzlu){
                projdi celý levý podstrom uzlu „aktUzel“
        }
        aktUzel = rodičovský uzel
}
```

Při průchodu nastavujeme příznak, pokud může kolidovat. U každého objektu jsou po všech průchodech nastaveny příznaky, které určují, v kolika osách se objekt protíná se zkoumaným objektem. Prvním krokem je tedy projít jeden segmentový strom reprezentující jednu souřadnici a všem objektům, které mohou kolidovat, se přiřadí příznak. To samé pak provedeme pro zbývající osy a výsledkem je seznam objektů, které jdou dále ke zpracování.

Další zpracování probíhá s rozdílem, že vynechávám test na obalovou kouli, protože objekt už je testován na AABB box.



Obrázek 3.2.4: Procházení stromu při detekci

3.2.3 Intervalový strom

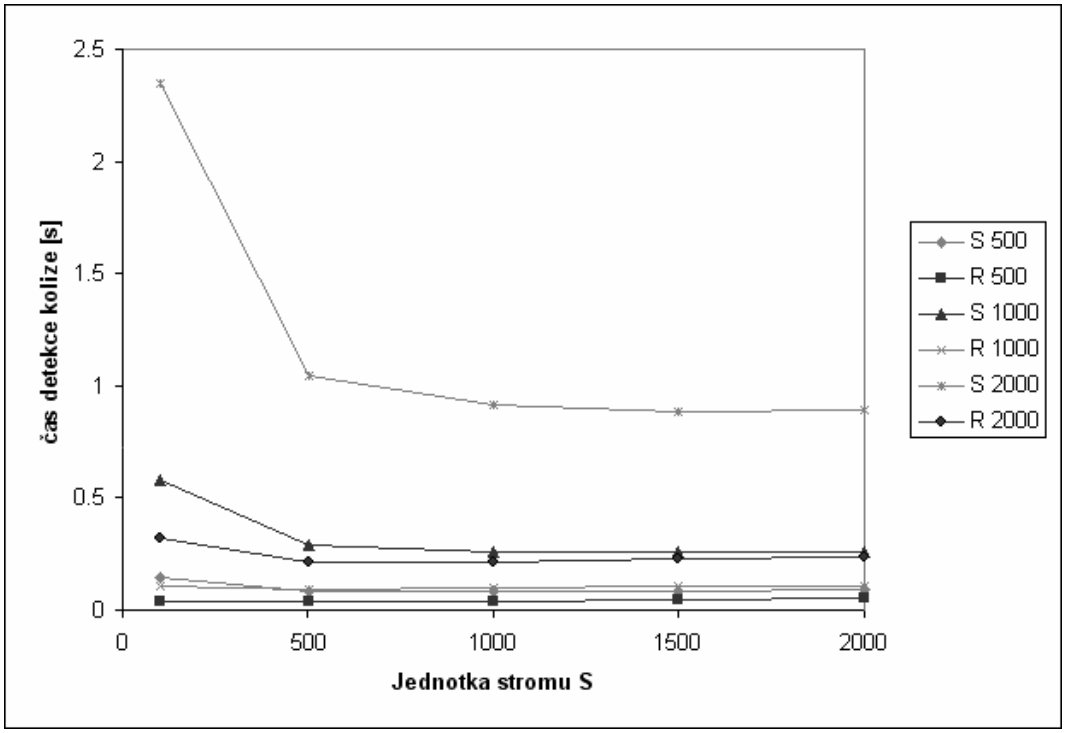
Tento strom je modifikací segmentového stromu. Byl vybrán jako další možné urychlení výběru předmětů. Segmentový strom je zde pouze v jedné ose a zbylé osy jsou reprezentovány v každém uzlu (struktura stromu je vidět na obrázku 3.2.4). Objekty jsou do stromu i do seznamů řazeny podle AABB boxů jako je tomu u segmentového stromu. Přesnost výběru je zde stejná jako u segmentového stromu.

U intervalového stromu, stejně jako u segmentového, musíme zavést určité omezení stromu. Stejným způsobem je tu tedy vypočtena tzv. „jednotka stromu“ podle vzorce 3.2:

Testováním různé jemnosti intervalového stromu jsme došli ke konstantě S. Výsledky testovaní jsou vidět v tabulce 2.2 a na grafu 2.2. Z těchto výsledků jsme zvolili pro konstantu S hodnotu 1000.

Typ scény	Jednotka stromu S (vzorec 2.1)				
	100	500	1000	1500	2000
C 500	0.146875	0.087500	0.084375	0.087500	0.093750
R 500	0.040625	0.037500	0.040625	0.043750	0.050000
C 1000	0.578125	0.290625	0.262500	0.259375	0.259375
R 1000	0.109375	0.093750	0.096875	0.103125	0.106250
C 2000	2.350000	1.040625	0.912500	0.881250	0.893750
R 2000	0.318750	0.215625	0.212500	0.225000	0.237500

Tabulka 2.2: Čas detekce [s] v závislosti na parametru S (Typ scény: R = rovnoměrné rozložení objektu, C = shlukovité rozložení objektů, Číslo za písmenem znamená počet objektů ve scéně)



Graf 2.2: Čas detekce [s] v závislosti na parametru S

Inicializace:

Inicializace spočívá ve vytvoření segmentového stromu pro načtenou scénu. Vytvoření stromu se skládá ze dvou kroků. Prvním krokem je vytvoření segmentového stromu pro osu x a druhým krokem je vytvoření seznamů pro osy y a z ve všech uzlech stromu. Tento proces se provádí pro každý přidávaný objekt.

1. Vytvoření segmentového stromu:
   Prvním krokem vytvoření stromu je vytvoření hlavního uzlu a nastavení parametrů stromu. Velikost scény a jednotka stromu, která je nastavena hlavnímu uzlu, se počítá stejně jako u segmentového stromu. Ze vzorečků 3.3 a 3.4 se vypočítá velikost a jednotka stromu.

Pokud je vytvořen základ stromu, mohou se do něj přidat objekty scény. Postupně se přidávají objekty do uzlů, do kterých náleží, a pokud uzly neexistují, vytvoří se. Tento postup je stejný jako u segmentového stromu, jen se vytváří pouze jeden strom pro osu x.

2. Vytvoření seznamu pro osy y a z:
   Po přidání objektů do segmentového stromu se musí objekt přidat do seznamů pro osy y a z, a to ke každému uzlu vytvořeného stromu, kterému objekty náleží, viz obr. 3.2.5. Při přidávání objektů musí být objekt zařazen na správné místo v seznamu. Do seznamů se objekty přidávají podle AABB boxu, kde se používá minimální hodnota v dané ose (podle této hodnoty jsou objekty seřazeny). Datová struktura je na obr. 3.2.5, kde každý prvek v seznamu má ukazatele na prvek reprezentující stejný objekt v listech aktuálního uzlu, pokud se v těchto listech vyskytuje.
   V případě, kdy se vytváří nový uzel segmentového stromu, se do tohoto musí uzlu zkopírovat i seznamy pro osy y a z. Samozřejmě jen prvky, které tomuto uzlu náleží podle osy x.

Aktualizace scény:

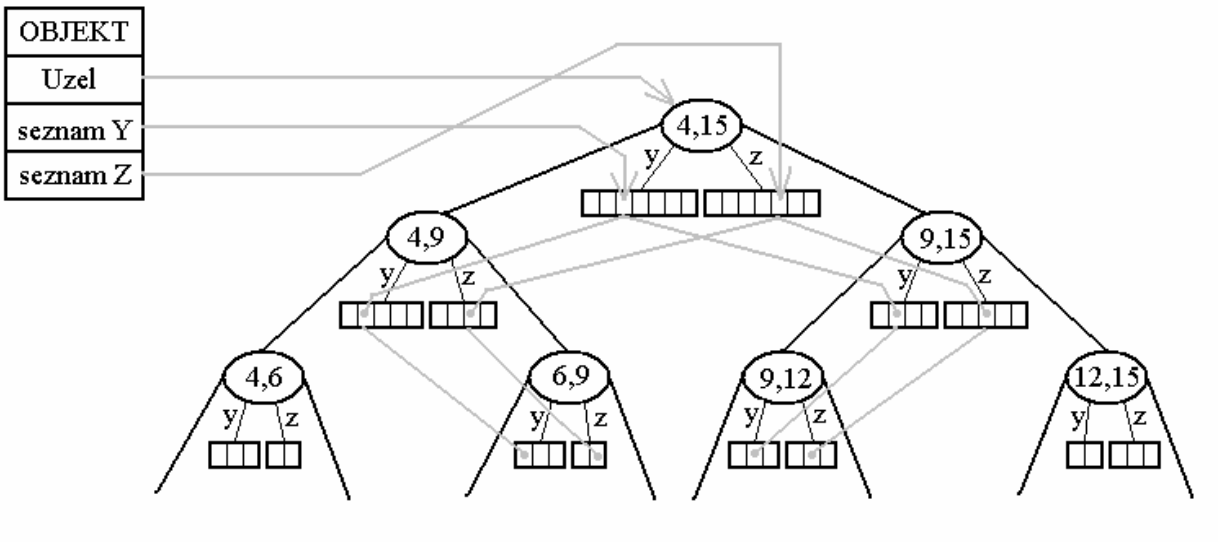
Aktualizace tohoto stromu se liší od aktualizace u předešlých metod. V ose x, kterou reprezentuje segmentový strom, je s objektem zacházeno jako u segmentového stromu. Objekt je tedy ze stromu vyjmut a po aktualizaci jeho polohy a rotace je do stromu opět přidán.

Aktualizace se ovšem liší v seznamech reprezentující osy y a z. Víme jistě, že při odebrání objektu by musel být znovu přidán, což znamená vyjmutí objektu ze seznamů a posunutí všech zbylých prvků a následně poté ho znovu zatřídit na správné místo. Tento krok je proto řešen změnou aktuální pozice v seznamech. Lze předpokládat, že se objekt nebude posouvat příliš rychle, a tedy stačí zkontrolovat pozici objektu v seznamu vůči okolním prvkům a zatřídit na správné místo.

V případech kdy se objekt přesouvá do jiných uzlů stromu, musíme ho do těchto uzlů přidat a případně odebrat z uzlů, kde už se nenachází.

Detekce kolize:

Prvním krokem detekce je průchod stromem, kdy se naleznou prvky, které mohou s objektem kolidovat v ose x, pro kterou je segmentový strom sestaven (Průchod segmentovým stromem je popsán v kap. 3.2.2). Následně se prohledají všechny uzly stromu, které obsahují zkoumaný objekt, a v uzlech se procházejí seznamy pro osy y a z, ze kterých se určí, které objekty mohou kolidovat a které nemohou.



Obrázek 3.2.5: Struktura intervalového stromu

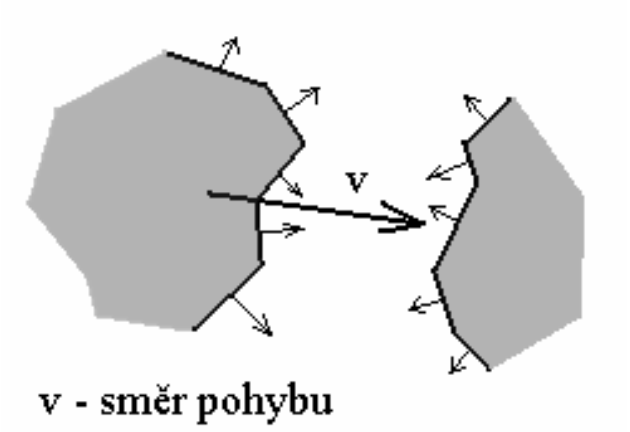
3.2.4 Řešení problémů s diskrétním časem

V rámci diplomové práce jsem experimentálně ověřoval metodu Ing. J. Paruse. Tato metoda slouží pro spočtení průsečíku pohybujícího se trojúhelníku a pohybujícího se bodu. Vstupem je pohybující se trojúhelník a pohybující se bod a výstupem metody jsou všechny průsečíky bodu a trojúhelníku. Bod i jednotlivé vrcholy trojúhelníku se musí pohybovat lineárně, ale každý vrchol trojúhelníku se může pohybovat jinak (trojúhelník se tedy může deformovat a měnit tvar).

Pokud jsme schopni vypočítat průsečík v čase, můžeme se vyvarovat chyb spojených s diskrétním časem (viz kapitola 2.3.1). V aplikaci zkoumáme detekci kolize v časových okamžicích, mezi kterými se tělesa pohybují. Pokud budeme schopni zjistit kolize, ke kterým došlo v časovém intervalu mezi diskrétními časovými okamžiky, můžeme na tyto kolize patřičně reagovat.

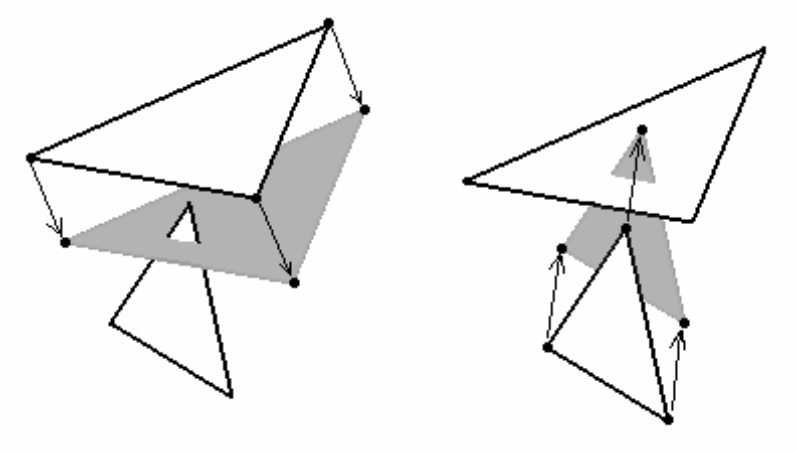
Proces zjištění kolize v čase:

Při výpočtu se musí vzít v potaz všechny trojúhelníky objektu a testovat je s trojúhelníky objektu, s kterým kolizi zjišťujeme. Do testu nevstupují všechny trojúhelníky, ale jen ty, které připadají v úvahu (ty, které mohou kolidovat). Prvním kritériem výběru trojúhelníku je příslušnost k OBB boxu, který koliduje. Druhým kritériem je směr normálových vektorů vůči směru pohybu. Úhel těchto vektorů musí být menší než 90 stupňů, jak je vidět na obrázku 3.2.6.



Obrázek 3.2.6: Výběr trojúhelníku pro test

Pokud máme vybrány trojúhelníky, u kterých připadá v úvahu kolize, použijeme metodu pro zjištění času kolize. Nejprve musíme první trojúhelník rozložit na tři body a testovat tyto body vůči druhému trojúhelníku a pak testovat trojúhelníky ještě naopak, tedy na body je rozložen druhý trojúhelník a první se testuje s těmito body. Proč je nutné dělat oboustranný test je ukázáno na obrázku 3.2.7., kde nevíme, který trojúhelník se pohybuje a který ne.



Obrázek 3.2.7: Vzájemná poloha pohybujících se trojúhelníků

Vlastní test bodu (statického nebo dynamického) vůči trojúhelníku (statickému nebo dynamickému) provádíme v několika krocích:

1) Prvním krokem je zjištění koeficientů kubické rovnice. K tomuto výpočtu jsem použil knihovnu „col3dlib.dll“ [8] a funkci „`ComputeCollisionEqCoefFast2`“, kterou obsahuje. Funkce má parametry:

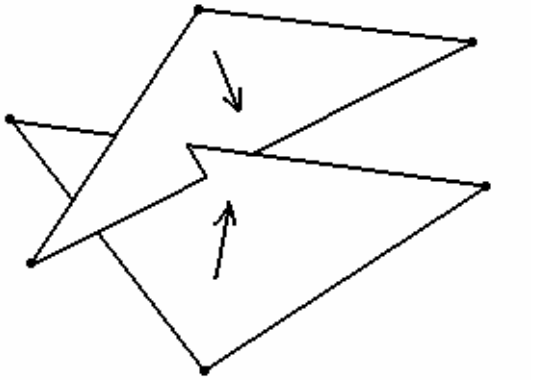
```
ComputeCollisionEqCoefFast2  (TTri  t1,  TTri  t2,  TFloat3  p0,
TFloat3 p1, double a, double b, double c, double d);

       Kde   TFloat3     - je pole 4 reálných čísel double
             TTri        - je pole 3 TFloat3
```

Vstup funkce je pohybující se trojúhelník zadaný počáteční (t1) a konečnou (t2) pozicí a pohybující se bod zadaný počáteční (p0) a konečnou (p1) pozicí. Výstupem funkce jsou koeficienty kubické rovnice a,b,c,d.

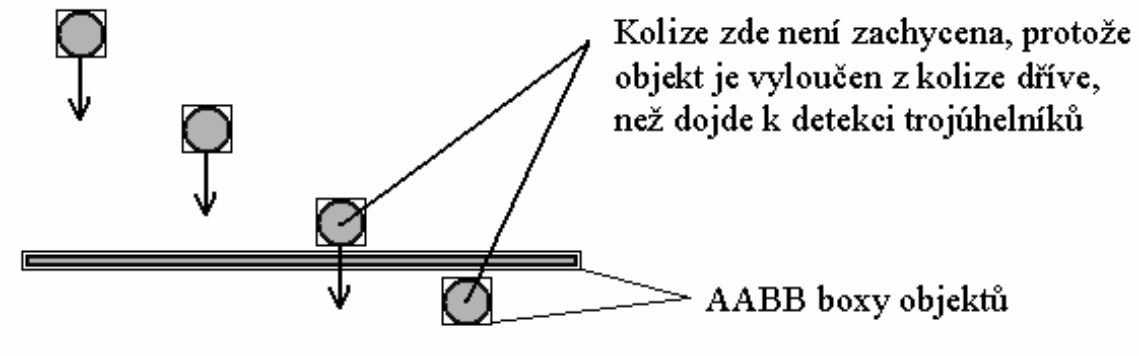
2) Poté, co dostaneme koeficienty kubické rovnice, vyřešíme tuto rovnici. V našem případě, kdy se trojúhelníky netransformují, dostáváme téměř vždy lineární rovnici (kvadratickou rovnici bychom mohli dostat při rotaci trojúhelníku). Po vyřešení rovnice získáme časy, kdy se bod protne s rovinou reprezentující trojúhelník. Není zde však ještě jisté, zda opravdu protíná trojúhelník.
3) Třetím testem je test, zda bod průniku je uvnitř trojúhelníku. Pro otestování bodu vůči trojúhelníku jsem použil metodu popsanou v [10]. Po otestování dostaneme výsledný čas kolize.

U celého objektu se zaznamená nejnižší čas kolize a trojúhelník, který koliduje. Podle kolidujícího trojúhelníku je pak vypočítána reakce. Podle času kolize je objekt posunut do bodu, kdy došlo ke kolizi. Tento proces není zcela správný, neboť zbývající čas by měl být využit už k pohybu po reakci objektu. Ale pro jednoduchost bylo zvoleno toto řešení, aby se minimalizovaly problémy s diskrétním časem.

Problém však není vyřešen úplně. Tato metoda neřeší situaci, pokud dva trojúhelníky kolidují hranou o hranu (viz obr 3.2.8) a pokud některá z použitých urychlovacích technik vyloučí objekt z kolize dříve, než dojde na test trojúhelníků, kolize nemusí být zachycena (viz obr 3.2.9).



Obrázek 3.2.8: Kolize dvou trojúhelníků hranou



Obrázek 3.2.9: Nezachycení kolize

3.3 Experimenty a výsledky

Použitý software a hardware na testování:
- Vývojový software: Microsoft® .NET ™, programovací jazyk C++
- OS: Microsoft® Windows XP SP 2
- Testovací sestava: AMD Athlon(tm) 64 Procesor 3000+, 1 GB RAM

Testovací scény:

Byly zvoleny 2 druhy testovacích scén, na kterých by měla být vidět vhodnost testovaných metod.

1) Scéna A: scéna s rovnoměrně rozmístěnými objekty
2) Scéna B: scéna s jedním shlukem objektů a několika objekty ve zbývajícím prostoru scény.

V obou případech se jedná o scénu o rozměrech -500 až +500 v každé ose. Objekty ve scénách jsou krychle s hranou 2. U shlukovitých scén má shluk tvar krychle o hraně 80. Všechny objekty ve scéně jsou dynamické a mají klidný náhodný pohyb.

Pro oba druhy scén byly vygenerovány scény s různým počtem objektů (250, 500, 750, 1000, 1250, 1500, 1750, 2000, 3000). Pro každý exemplář scény s určitým počtem objektů bylo dále vygenerováno 5 scén, aby se odstranily chyby měření způsobené umístěním objektů.

Testované metody:

Jedná se o metody:

1) Metoda A: rovnoměrná mřížka
2) Metoda B: prostorový strom
3) Metoda C: sada segmentových stromů
4) Metoda D: intervalový strom

Aby bylo možno testované metody vzájemně porovnat, musíme je sjednotit podle určitého kritéria. Zvolil jsem paměťové nároky na použitou datovou strukturu metody. Pokud pro metodu C použijeme velikost stromu zvolenou v kap. 3.2.2, musíme upravit podle velikosti použité paměti ostatní metody. Na jeden strom z této metody připadá 1512 uzlů (v každém uzlu je seznam ukazatelů na objekty, které může uzel obsahovat). Pro celou metodu potřebujeme tři stromy tedy 3 * 1512 = 4536 uzlů. Pokud vezmeme v úvahu, že uzel stromu paměťově odpovídá jedné buňce v rastru u metody A nebo B, musíme vytvořit pole o stejném počtu prvků. Nejbližší vyšší celé číslo, které tomuto odpovídá, je 17 buněk v jedné ose mřížky (17*17*17 = 4913). Pro metodu D stačí zvolit stejně rozvětvený strom jako pro metodu C. Intervalový strom má sice v každém listu 3 seznamy, ale ukládá se zde pouze jeden strom.(tedy 3 stromy * 1512 uzlů s jedním seznamem = 1512 uzlů * 3 seznamy v jednom uzlu).

Jedná se zde jen o pokus sjednotit paměť použitou na vytvoření struktury u metody. Nezapočítávám zde paměť, kterou spotřebují objekty pro uložení informace, kam patří. V tomto případě největší spotřebu má metoda A a B. Ty si totiž pamatují všechny buňky, kterým objekt náleží. U metody C si objekt pamatuje vždy jen 6 ukazatelů na počáteční a koncové uzly ve stromech (viz kap. 3.2.2). U metody D si objekt ukládá informace, ve kterých uzlech leží, což u segmentového stromu o hloubce 11 znamená, že může být maximálně v 18 uzlech.

Testy byly prováděny zvlášť pro inicializaci datové struktury (inicializace proběhne jen jedenkrát, při načtení scény) a zvlášť na detekci kolize.

3.3.1 Inicializace datové struktury

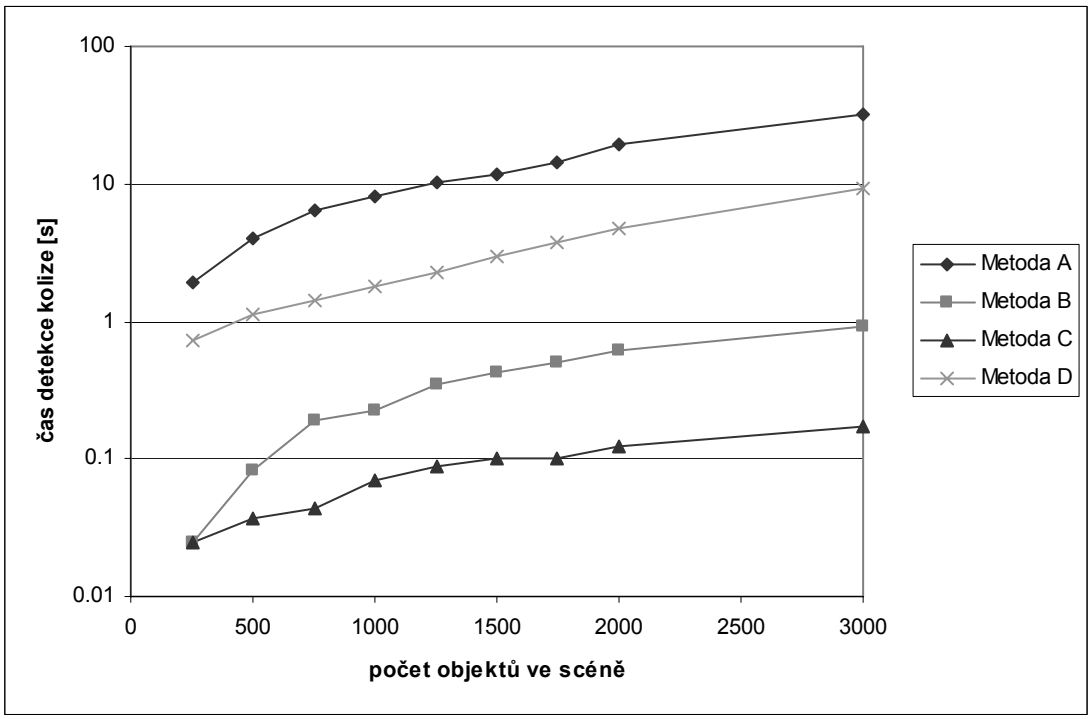
V tabulce 2.3 a na grafu 2.3 je vidět, že při inicializaci struktury vyšla nejhůře regulární mřížka, kde se vytvářejí a inicializují i buňky, které nejsou použity (jsou prázdné). Stromy zde vycházely daleko lépe. U prostorového stromu záleží, kolik buněk objekty zabírají a kolik je třeba vytvořit uzlů stromu. I u velkého počtu rovnoměrně rozložených objektů, kdy je zabrána velká část prostoru, vychází čas inicializace menší než u mřížky.

Při inicializaci mřížky je jedno, zda se jedná o data rovnoměrná nebo shlukovitá, neboť se vždy vytváří celá mřížka. U všech stromů je vidět úspora u shlukovitých dat, kde se nevytváří tak veliká část stromu. U prostorového stromu je tento rozdíl zvlášť patrný, neboť objekty v prostoru zabírají jen několik buněk. Jsou tedy vytvořeny a inicializovány jen tyto buňky, které jsou pak následně uloženy v prostorovém stromu.

Rovnoměrné rozložení objektů:

	0	250	500	750	1000	1250	1500	1750	2000	3000
Metoda A	0	1.9375	3.9625	6.3375	8.1125	10.1500	11.6125	14.2375	19.1875	31.7250
Metoda B	0	0.0250	0.0813	0.1875	0.2250	0.3438	0.4188	0.5063	0.6250	0.9313
Metoda C	0	0.0250	0.0375	0.0438	0.0688	0.0875	0.1000	0.1000	0.1250	0.1750
Metoda D	0	0.7375	1.1156	1.4281	1.8031	2.2781	2.9344	3.7906	4.6719	9.3938

Tabulka 2.3: Čas inicializace datové struktury pro rovnoměrná data [s]

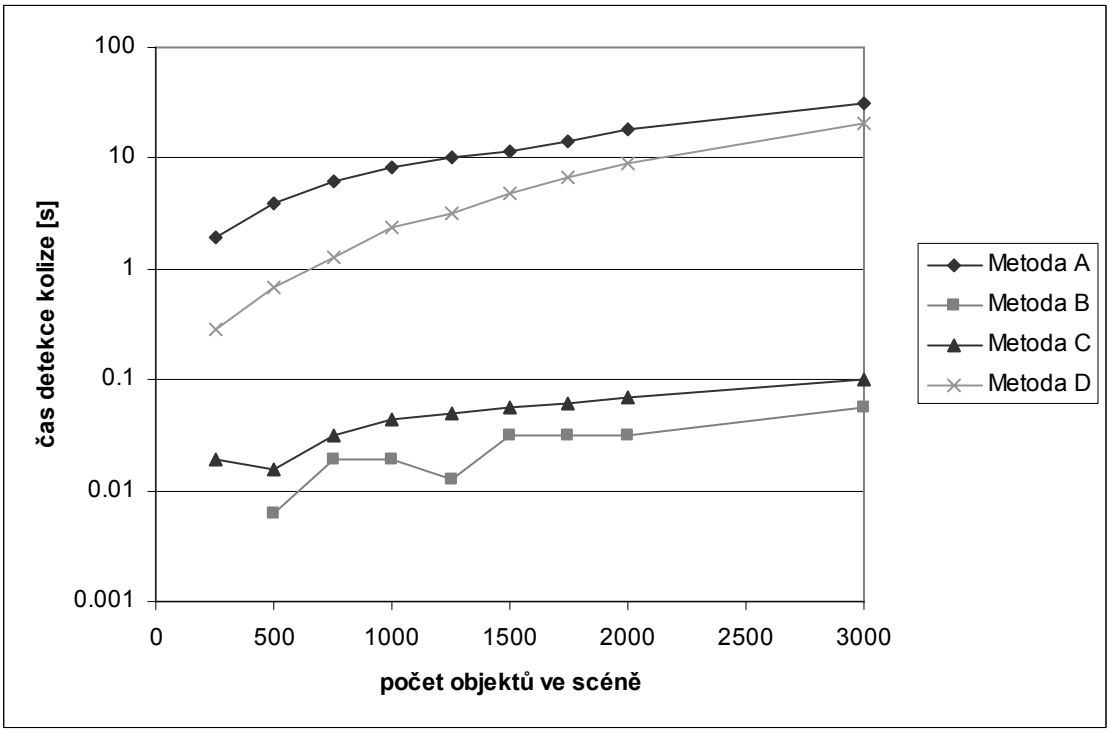


Graf 2.3: Čas inicializace datové struktury pro rovnoměrná data [s]

Shlukovité rozložení objektů:

	0	250	500	750	1000	1250	1500	1750	2000	3000
Metoda A	0	1.9625	3.9438	6.2438	8.2563	10.3063	11.6125	14.1813	18.3063	31.8250
Metoda B	0	0.0000	0.0063	0.0188	0.0188	0.0125	0.0313	0.0313	0.0313	0.0563
Metoda C	0	0.0188	0.0156	0.0313	0.0438	0.0500	0.0563	0.0625	0.0688	0.1000
Metoda D	0	0.2844	0.6844	1.2688	2.3688	3.2188	4.8531	6.7188	9.1594	20.5000

Tabulka 2.4: Čas inicializace datové struktury pro shlukovitá data [s]



Graf 2.4: Čas inicializace datové struktury pro shlukovitá data [s]

3.3.2 Detekce kolize

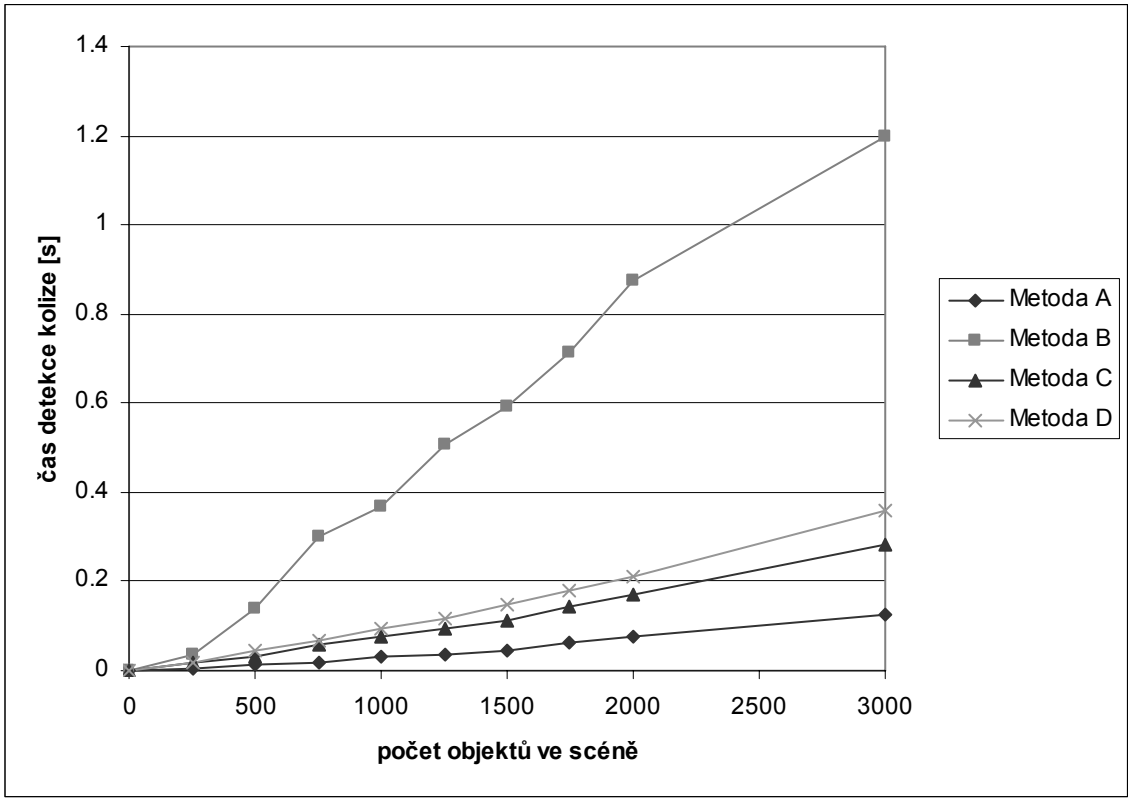
U detekce kolize se projevily rozdíly použitých scén. Regulární mřížka vyšla lépe u rovnoměrně rozložených objektů ve scéně, zatímco u shluků vyšel lépe segmentový a intervalový strom, jak bylo očekáváno. Čas detekce kolize u prostorového stromu odpovídá času u regulární mřížky s určitým navýšením, zapříčiněné procházením stromem, kdy je nutno vytvářet nové uzly, pokud ve stromě nejsou.

U shlukovitých dat záleží, jak se shluk přiřadí do mřížky. Zda je shluk ve více či méně buňkách. Bylo testováno několik scén od každého druhu, aby se tomuto jevu zabránilo.

Rovnoměrné rozložení objektů:

	0	250	500	750	1000	1250	1500	1750	2000	3000
Metoda A	0	0.0063	0.0125	0.0188	0.0313	0.0375	0.0438	0.0625	0.0750	0.1250
Metoda B	0	0.0375	0.1375	0.3000	0.3688	0.5063	0.5938	0.7125	0.8750	1.2000
Metoda C	0	0.0188	0.0313	0.0563	0.0750	0.0938	0.1125	0.1438	0.1688	0.2813
Metoda D	0	0.0188	0.0438	0.0656	0.0938	0.1156	0.1469	0.1781	0.2094	0.3594

Tabulka 2.5: Čas detekce kolize pro rovnoměrná data [s]

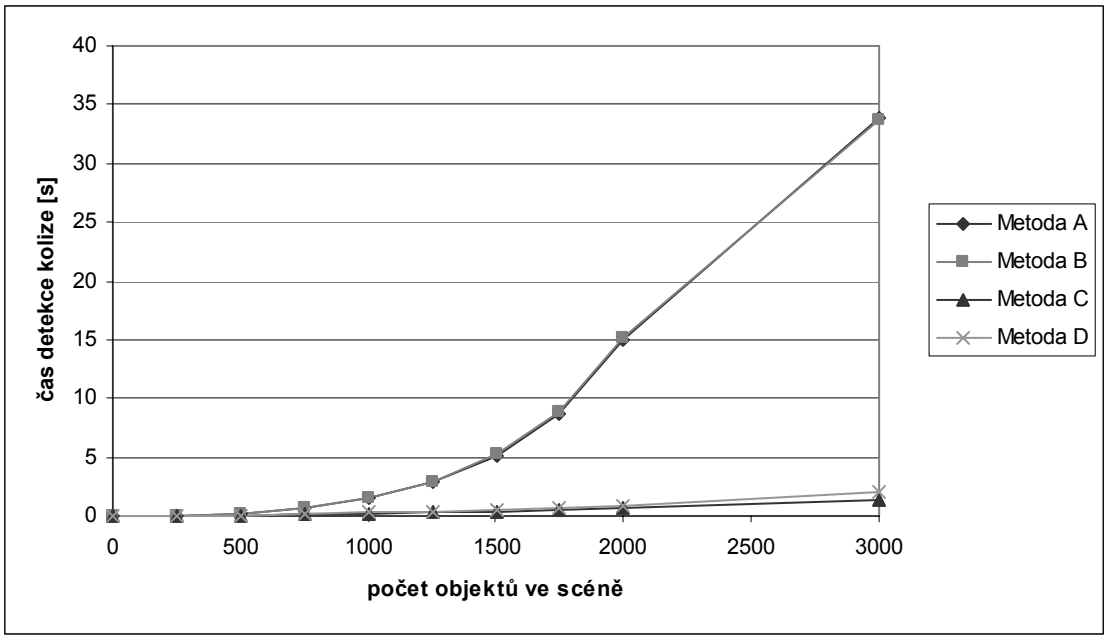


Graf 2.5: Čas detekce kolize pro rovnoměrná data [s]

Shlukovité rozložení objektů

	0	250	500	750	1000	1250	1500	1750	2000	3000
Metoda A	0	0.0500	0.1875	0.6750	1.5875	2.8875	5.1313	8.7250	15.0563	33.8125
Metoda B	0	0.0500	0.1938	0.6875	1.6000	2.9250	5.2000	8.7875	15.0750	33.6813
Metoda C	0	0.0563	0.0563	0.1063	0.1813	0.2688	0.3688	0.4875	0.6125	1.3750
Metoda D	0	0.0281	0.0813	0.1531	0.2656	0.3531	0.5406	0.7063	0.9063	2.0344

Tabulka 2.6: Čas detekce kolize pro shlukovitá data [s]



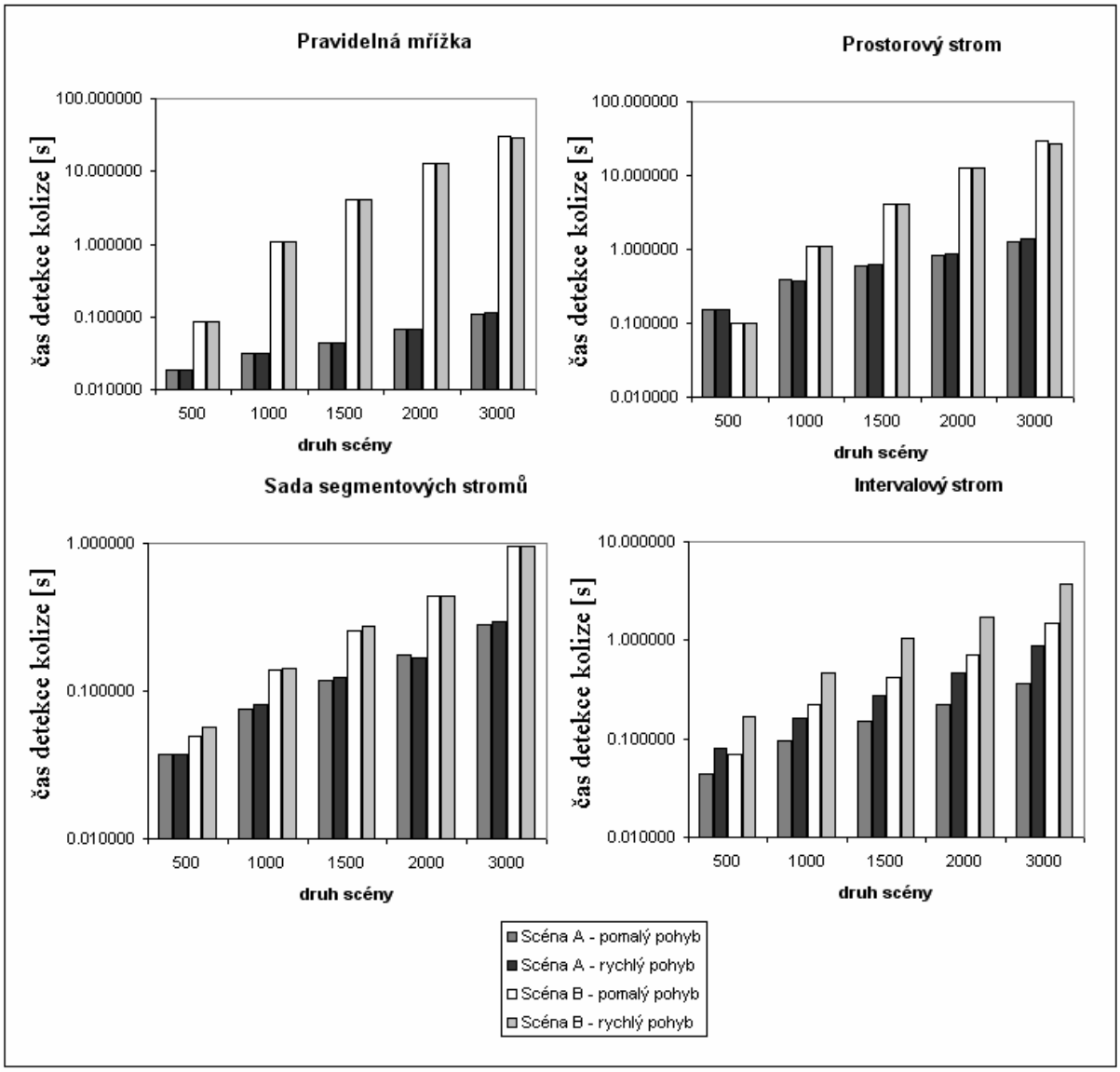
Graf 2.6: Čas detekce kolize pro shlukovitá data [s]

3.3.3 Závislost na druhu pohybu

Metody byly také testovány na vybraných scénách s různým druhem pohybu objektů. Objekty se opět náhodně pohybují všechny. Mění se druh pohybu, kdy jednou se objekty pohybují jen nepatrně a druhý typ pohybu je velice dynamický.

Pravidelná mřížka					
	500	1000	1500	2000	3000
Scéna A - pomalý pohyb	0.018750	0.031250	0.043750	0.068750	0.106250
Scéna A - rychlý pohyb	0.018750	0.031250	0.043750	0.068750	0.112500
Scéna B - pomalý pohyb	0.087500	1.068750	4.175000	12.681250	30.137500
Scéna B - rychlý pohyb	0.087500	1.081250	4.087500	12.706250	29.000000
Prostorový strom					
	500	1000	1500	2000	3000
Scéna A - pomalý pohyb	0.150000	0.387500	0.600000	0.843750	1.281250
Scéna A - rychlý pohyb	0.150000	0.381250	0.637500	0.881250	1.400000
Scéna B - pomalý pohyb	0.100000	1.093750	4.181250	12.756250	29.037500
Scéna B - rychlý pohyb	0.100000	1.093750	4.075000	12.812500	26.962500
Sada segmentových stromů					
	500	1000	1500	2000	3000
Scéna A - pomalý pohyb	0.037500	0.075000	0.118750	0.175000	0.281250
Scéna A - rychlý pohyb	0.037500	0.081250	0.125000	0.168750	0.293750
Scéna B - pomalý pohyb	0.050000	0.137500	0.256250	0.443750	0.943750
Scéna B - rychlý pohyb	0.056250	0.143750	0.275000	0.437500	0.956250
Intervalový strom					
	500	1000	1500	2000	3000
Scéna A - pomalý pohyb	0.043750	0.093750	0.150000	0.218750	0.362500
Scéna A - rychlý pohyb	0.081250	0.162750	0.275000	0.468750	0.875000
Scéna B - pomalý pohyb	0.068750	0.225000	0.425000	0.700000	1.487500
Scéna B - rychlý pohyb	0.168750	0.462500	1.043750	1.731250	3.662500

Tabulka 2.7: Závislost na druhu pohybu objektů [s]



Graf 2.7: Závislost na druhu pohybu objektů

3.3.4 Výsledky experimentů

Při inicializaci datových struktur je vidět rozdíl mezi případem, kdy se musí inicializovat celý prostor, a případem, kdy se inicializuje pouze použitá část prostoru. Inicializace se však provádí pouze při spuštění a není tak důležitá jako detekce kolize testovaná v další fázi.

Detekce kolize vyšla podle předpokladů. V případě rovnoměrných dat vycházela lépe rovnoměrná mřížka a prostorový strom a v případě shluku objektů vycházel daleko lépe segmentový a intervalový strom. Intervalový strom, který měl být vylepšením metody segmentových stromů, vychází bohužel hůře než stromy segmentové.

Pro rovnoměrná data nebo malý počet objektů se podle testů více hodí rovnoměrná mřížka. Rovnoměrná mřížka je snazší na implementaci a pro rovnoměrně rozprostřená data vychází lépe. Pokud by však scény byly typu, kdy je větší část objektů v nějakých shlucích, je lépe použít segmentového stromu. Pokud nemáme představu, jaká data budou na vstupu, zvolil bych segmentový strom. I přesto že pro rovnoměrná data vychází hůře, není zhoršení tak výrazné, jako v případě mřížky a scény se shluky objektů.

Bylo také uvažováno, jaké budou rozdíly ve scénách se skoro statickými objekty na rozdíl od objektů hodně dynamických. Z principů implementace vychází, že tyto vlastnosti objektů čas detekce neovlivní. Detekce je prováděna vyjmutím objektu z mřížky nebo segmentového stromu a zařazením na nové místo. Z tohoto důvodu je jedno, o jak velkou vzdálenost se objekt posune. Druhy pohybu byly otestovány a výsledky vyšly podle očekávání a jsou v tabulce 2.7 a na grafu 2.7. U intervalového stromu je detekce kolize druhem pohybu ovlivněna. Zde záleží, zda se objekty přesunují z uzlu do uzlu, kdy se čas aktualizace zvyšuje.

4 Závěr

Tato diplomová práce měla za úkol najít možná vylepšení získaného referenčního řešení problému detekce kolizí. Referenční řešení bylo zaměřeno pouze na určitý typ scén, z čehož vzešel i výběr implementovaných metod. Implementoval jsem urychlovací metody, které jsou vhodné pro scény s nerovnoměrným rozložením objektů. Všechny implementované metody i s metodou z referenčního zadání jsem otestoval na celé řadě scén. Bylo vyzkoušeno několik metod, ze kterých nejlépe vyšla metoda segmentových stromů. Jako vylepšení této metody byl vyzkoušen ještě intervalový strom. V testech však vycházel stále lépe segmentový strom. Segmentový strom je vhodný jak pro scény se shlukovými daty, tak pro scény s hodně dynamickými objekty.

Další metoda testovaná v této diplomové práci zkoušela odstranit problémy s diskrétním časem. Tyto problémy se částečně podařilo odstranit na úrovni testu trojúhelníků. Dokončení této metody by mohlo být dalším rozšířením.

Seznam použité literatury

[1] Bc. Jaroslav Matoušek: Detekce kolizí objektů reprezentovaných trojúhelníkovými modely (diplomová práce), Západočeská univerzita, 2004.

[2] Gino van den Bergen: Collision Detection in Interactive 3D Computer Animation, University Press Facilities, Eidhoven, 1999.

[3] Žára, J., Beneš B., Felkel P: Moderní počítačová grafika, s. 357-378, Computer Press, 1998.

[4] Bradshaw, G., O'Sullivan C.: Adaptive Medial-Axis Approximation for Sphere-Tree Construction, C. Accepted for publication in ACM Transactions on Graphics.

[5] Bradshaw, G.: Bounding Volume Hierarchies for Level-of-Detail Collision Handling, Thesis - Dept. of Computer Science, Trinity College Dublin, May 2002.

[6] Franco P.Preparata, Michael, I.Shamos: Computational Geometry: An Introduction, Springer Verlag New York berlin heidelberg Tokyo, 1985

[7] M. L. Gavrielova, J. Rokne: Collision Detection Optimization in a Multi-particle System, International Journal of Computer Geometry & Aplications, 2003

[8] Ing. J. Parus: knihovna col3dlib.dll pro počítání průsečíku pohybujícího se bodu a pohybujícího se trojúhelníku

[9] Ing J. Patera: Metody extrakce iso-ploch pro volumetrická data a zobrazování skalárních veličin (diplomová práce), Západočeská univerzita, 2002

Odkazy

[10] http://www.blackpawn.com/texts/pointinpoly/default.html

[11] http://www.fi.muni.cz/~ptx/PA010/Slides/PA010_7.pdf

[12] http://www.fi.muni.cz/~sochor/PA010/Slajdy/SpaceSearch.pdf

Evidenční list

Souhlasím s tím, aby moje diplomová práce byla půjčována k prezenčnímu studiu v Univerzitní knihovně ZČU v Plzni.

Datum: Podpis:

Uživatel stvrzuje svým čitelným podpisem, že tuto diplomovou práci použil ke studijním účelům a prohlašuje, že ji uvede mezi použitými prameny.

Jméno	Fakulta/Katedra	Datum	Podpis