PHYSICAL BASED ANIMATIONS AND MATHEMATICAL MODELING

State examination: S3 Detekcia kolízie

Autor: Marián Kravec

Úvod

Úlohou detekcie kolízie je určiť kedy a kde dôjde ku kontaktu čiže kolízii medzi jednotlivými objektami.

Táto detekcia sa rozdeľuje do troch fáz:

- Broad Phase - úlohou tejto fázy je s čo najmenšou výpočtovou zložitosťou určiť dvojice objektov ktoré by mohli byť potenciálne v kolízii, inými slovami, úlohou tejto fázy je čo najjednoduchšie odseparovať dvojice objektov pri ktorých s určitosťou kolízia nenastáva aby výpočtovo náročnejšia ale presnejšie metódy v ďalších fázach už s týmito dvojica nemuseli pracovať
- Mid Phase - v tejto fáze sa snažíme množinu dvojíc objektov v potenciálnej kolízii ďalej zúžiť využitím o niečo presnejší ale výpočtovo zložitejších metód, výsledkom tejto fázy je stále množina iba potenciálnych kolízii avšak by mala byť menšia ako vstupná množina
- Narrow Phase - posledná fáza určí ktoré dvojice objektov z množiny potenciálnych kolízii su naozaj v kolízii. Okrem toho úlohou tejto fázy je aj určiť či ide o dotyk alebo prienik, v prípade prieniku aká je prieniková hĺbka a kde presne kolízia nastala

Broad Phase

Mriežka

Jedným z bežných spôsobov riešenia tejto fázy je obalenie objektov do AABB obálok čiže do kvádrov ktorých hrany sú rovnobežné s osami priestoru. Priestor následne pomocou rovnomernej mriežky rozdelíme na bunky čiže kocky (respektíve štvorce v 2D prípade). Následne určíme v ktorých bunkách mriežky sa jednotlivé objekty nachádzajú. Ak zistíme, že v jednej bunke sa nachádzajú 2 a viac objektov tvrdíme, že tieto objekty sú potenciálne v kolízii, naopak ak 2 objektu nemajú žiadnu spoločnú bunku tvrdíme, že tieto objekty určite v kolízii nie sú. Problémom tohto prístupu je, že je potrebné správne určiť veľkosť jednej bunky. V prípade veľmi veľkých buniek metóda vyhodnotí príliš veľa dvojíc objektov ako potenciálne v kolízii čo výrazne spomaľuje výpočet ďalšej fázy. Naopak ak sú bunky veľmi malé je ich počet veľmi veľký čo spomaľuje výpočet tejto fázy.

Tento problém sa často rieši pomocou hierarchických mriežok kde existuje niekoľko úrovní mriežok so stále väčšou a väčšou jemnosťou (zväčša ja jemnosť každou úrovňou zdvojnásobuje čiže sa bunka rozdelí na podbunky so stranou polovičnej dĺžky). Pri tomto postupe sa najskôr určí rozlíšenie objektu, čiže po ktorú úroveň hierarchie má význam daný objekt počítať. Vďaka tomu veľké objekty majú malé rozlíšenie a nekomplikujú výpočet tým, že by zaberali veľké množstvo malých buniek a malé objekty majú veľké rozlíšenie vďaka čomu pri kontrole ich prieniku sa vzácnejšie stáva, že ich nesprávne označíme za potenciálne v kolízii kvôli tomu, že patria do spoločnej veľkej bunky.

Rozlíšenie objektu X sa počíta následovne: $Res(X) = i$ kde pre i platí $a \leq \frac{Size(X)}{s_i} \leq b$ kde $Size(X)$ je veľkosť najväčšieho rozmeru AABB obálky a s_i je veľkosť bunky na i-tej úrovni, koeficienty a a b sa často nastavujú na $a = 0.5$ a $b = 1$.

Následne sa podobne ako v predchádzajúcej metóde sa v ďalšom kroku určí do ktorých buniek daný objekt patrí avšak v tomto prípade sa toto určovanie príslušnosti robí od najnižšej úrovni (najväčších buniek) až

po úroveň rozlíšenia danej obálky.

Keď kontrolujeme či dvojica objektov je v kolízii robíme to na najväčšej spoločnej úrovni, čiže ak máme objekty X a Y tak úroveň na ktorej kontrolujeme, či majú spoločnú bunku je $Min(Res(X), Res(Y))$.

Ďalším problémom ktorý v týchto metódach vzniká je riedkosť mriežok, čiže situácia keď veľké množstvo buniek je prázdnych no aj napriek tomu si zapamätávame informáciu o ich existencii, tento problém sa bežne rieši metódou priestorového hashovania kde sa informácia o prítomnosti objektu v bunke zapíše do hashovacej tabuľky na pozíciu hashu pozície bunky.

SAP

Inou metódou je metóda Sweep-And-Prune (SAP) ktorá vy kontrolu či sú objekty v kolízii využíva vetu o deliacej osi. Táto veta hovorí, že dva objekty nie sú v kolízii vtedy a práve vtedy ak existuje os na ktorej obrazy týchto objektov majú prázdny prienik, táto veta je ekvivalentná s vetou o deliacej rovine ktorá hovorí, že dva objekty nie sú v kolízii vtedy a práve vtedy ak existuje rovina deliaca priestor tak že jeden z objektov sa nachádza celý nad rovinou a druhý celý pod. Dá sa ukázať, že deliaca os je normálov deliacej rovinu z čoho sa dá odvodiť ekvivalencia týchto viet. Výhodou vety o deliacej osi je indiferencia na posun osi (dôležitý je iba smer) zatiaľ čo pri vete o deliacej rovine je posun roviny môže spôsobiť, že rovina už nebude deliaca.
Ukazuje sa, že počet osí ktoré treba skontrolovať či nie sú deliacimi nie je neobmedzený, v prípade použitia AABB obálok stačí skontrolovať osi priestoru (zväčša jednotková vektory, 2 pre 2D a 3 pre 3D), v prípade OBB obálok (kvádrov otočených tak aby ich objem čo najmenší) stačí skontrolovať vlastné vektory týchto objektov a 3D prípade aj ich vzájomné vektorové súčiny (nanajvýš 4 pre 2D a 15 pre 3D).
Ak použijeme AABB obálky, čiže množina kontrolovaných smerov su jednotkové vektory, vieme potenciálnu kolíziu určiť tak, že si zobrazíme všetky objekty na tieto osi a následne prechádzame po jednotlivých osiach pričom pri prechádzaní po osi si pamätáme ktoré objekty už začali ale ešte neskončili, ak objekt začal pričom existujú objekty ktoré začali skôr ale ešte neskončili, tieto objekty považujeme za potenciálne kolidujúce, a vložime ich ako dvojicu do potenciálnych kolízii danej osi. Keď tieto množiny vypočítame pre všetky osi, výsledná množina je prienikom týchto množín (čiže dvojice pre ktorá ani jedna os nebola deliaca).

Priebežný update

Počítanie týchto metód odznova vždy pre každý frame (posun objektov) by bolo výpočtovo náročné preto je potrebné byť schopný hodnoty vypočítané v predchádzajúcom čase a iba ich updatovať pomocou informácii o posune objektov.
V prípade rovnomernej mriežky je pri posune objektu updatnuť iba bunky ktoré obal objektu mohol opustiť respektíve do nich vojsť.
V prípade hierarchickej mriežky nie je optimalizácia updatu nutná keďže vďaka vlastnosti, že objekt je počítaný iba do nevyhnutnej úrovne mriežky objekt nikdy nebude patriť do veľkého množstva buniek a preto jeho update je spočíva v odstránení pôvodnej informácii o jeho príslušnosti a jeho pridanie nanovo (samozrejme iba ak sa objekt pohol).
V prípade SAP metódy využívajúcej AABB obálky nám stačí pri posune obálky skontrolovať či posunom začiatku obálky zmenil vzťah z koncom inej obálky (napríklad pred posunom obálka X začínala za koncom obálky Y ale to sa po posune zmenilo a už obálka X začína pred koncom obálky Y z čoho vyplýva, že už sú v potenciálnej kolízii) a podobne skontrolovať posun koncov obálok. Následne je ešte nutné nanovo vypočítať prienik množín potenciálnych kolízii jednotlivých osí.

Manažovanie párov potenciálne kolídujúcich objektov

Hlavnou snahou tejto fázy je určiť potenciálne kolidujúce páry objektov. Informácia o tom o ktoré páry ide musí byť nejakým spôsobom odovzdaná ďalšej fáze.

Medzi triviálne prístupy patrí:

- matica kolízii - matica ktorá na pozícii (i, j) obsahuje informáciu či i-ty a j-ty objekt sú v potenciálnej kolízii.Tento prístup umožňuje jednoduché zisťovanie s ktorými objekty je konkrétny objekt v potenciálnej kolízii a update informácie o konkrétnej dvojici. Problémom tohto prístupu je, že vyžaduje kvadraticky veľa pamäte čo v prípade nízke počtu kolízii veľkého počtu objektov spôsobuje že matica je veľmi riedka a zbytočne zaberá príliš veľa miesta.
- zoznam dvojíc - tento prístup jednoducho vráti zoznam dvojíc objektov ktoré sú potenciálne v kolízii. Vďaka tomu, že obsahuje iba nevyhnutnú informáciu a priestorovo veľmi efektívny. Jeho problémom je pomalá kontrola či sa dvojica v zozname nachádza a update zoznamu.

Obe triviálne prístupy majú svoje problémy preto sa tento problém často rieši komplikovanejšou štruktúrou a to priestorovou hashovacou tabuľkou vďaka čomu máme štruktúru je priestorovo pomerne šetrná a zároveň ponúka rýchle vyhľadávanie a update.

Mid Phase

Väčšinou sa presnejšie určovanie či došlo ku kolízii dvoch objektov z potenciálne kolidujúcich sa väčšinou robí rozdeliť obálku objektu na viac konvexných častí vďaka čomu obálka oveľa lepšie opisuje objekt ale nie je extrémne výpočtovo komplexná.

Bounding Volume Hierarchy

V tejto metóde sa vytvára viacero obálok ktoré vytvárajú stromovú (hierarchickú) štruktúru kde v koreni je tá najjednoduchšia obálka ktorá jedným konvexným útvarom opisuje celý objekt a každá hlbšia vrstva stromu opisuje objekt pomocou väčšieho počtu presnejších konvexných útvarov pričom v tejto štruktúre platí, že zjednotenie častí objektu ktoré opisovanými podobálkami detí určitého vrcholu tvorí časť objektu ktorú opisuje práve ich rodič.

Strom môže byť generovaný zhora nadol kde najskôr vytvoríme obálku celého objektu a následne objekt delíme na menšie časti pre ktoré počítame jednotlivé obálky. Druhý spôsob generovania je zdola nahor kde si najskôr objekt rozdelíme ne jednotlivé časti, pre tieto časti vytvoríme obálky a následne postupne časti spájame a vytvoríme obálky týchto nadobjektov kým nespojíme všetky do jedného pôvodného objektu.

Na určenie či sú dva objekty v potenciálnej kolízii sa využíva metóda Tandem Traversal. Táto metóda dostane na začiatku obálky z koreňov stromov testovaných objektov a postupne rekurzívne ide hlbšie v stromoch kým neurčí, že objekty nie sú v kolízii alebo nenájde kolidujúce listy.
Metóda funguje tak, že na vstupe dostane dvojicu obálok T_x a T_y zo stromov prislúchajúcich k objektom X a Y, následne vykoná následovný algoritmus:

TandemTraversal(T_x, T_y):

- T_x a T_y nie sú v kolízii
  - vráť $\emptyset$
- T_x a T_y sú v kolízii
  - T_x aj T_y sú listy
    - vráť (T_x, T_y)
  - iba T_x je list
    - vráť $\bigcup_{T_y^i = potomok T_y} \text{TandemTraversal}(T_x, T_y^i)$
  - iba T_y je list

    * vráť $\bigcup_{T_x^i = potomok T_x} \text{TandemTraversal}(T_x^i, T_y)$
- ani jeden vrchol nie je list
    * geometria T_y je väčšia
        · vráť $\bigcup_{T_y^i = potomok T_y} \text{TandemTraversal}(T_x, T_y^i)$
    * geometria T_x je väčšia
        · vráť $\bigcup_{T_x^i = potomok T_x} \text{TandemTraversal}(T_x^i, T_y)$

kDOP

kDOP čiže k-Discrete Orientation Polytopes je typ obálky ktorý vzniká prienikom k negatívnych polpriestorov, pričom každý rovina deliaca priestor je definovaná normálovým vektorom (orientáciou) a vzdialenosťou od stredu objektu. Tieto normály (orientácie) sú pre všetky objekty rovnaké. Obálka AABB sa dá považovať za špecifický prípad kDOP konkrétne 4DOP v 2D priestore a 6DOP v 3D priestore kde normálové vektory sú jednotkové vektory a mínus jednotkové vektory.

Pri kontrole kolízie dvoch obálok nám stačí použiť štandardnú SAT metódu (metódu využívajúcu vetu o deliacej priamke) využitím našich k orientácii.

OBB

Ďalším typom obálky je OBB čiže Oriented Bounded Box, táto obálka sa podobá na AABB obálku tým, že v základnej verzii ide v oboch prípadoch o kváder avšak narozdiel od AABB obálky hrany OBB obálky ne musia byť rovnobežné s osami priestoru ale sú účené vlastnými vektormi objektu (takže stále platí, že sú na seba kolmé alebo rovnobežné). Čiže táto obálka je definovaná vlastnými vektormi objektu (normalizovanými), veľkosťami v jednotlivých smeroch vlastných vektorov a stredom objektu.

Ako bolo spomenuté v časti Broad Phase - SAP, na takéto obálka sa dá dobre použiť SAT metóda v ktorej sa ako kontrolované osi použijú práve vlastné vektory (ak ich vektorové súčiny).

OBB obálky sa dajú modifikovať aj na hierarchickú verziu podobnú binárnemu BVH generovaním stromu zhora nadol. Najskôr sa vypočíta OBB celého objektu a následne sa objekt rozdelí rovnou prechádzajúcom stredom obálky kolmou na najdlhšiu os obálky a pre jednotlivé polovice sa vypočítaju samostatné obálky a postup sa opakuje.

Typy kolízii

Guľa x guľa - ku kolízii medzi dvomi guľami dochádza vtedy keď vzdialenosť ich stredov je menšia ako súčet ich polomerov. Normála kolízie je vektor rovnobežný so spojnicou stredov a body kolízie sú body na guliach ležiace na tejto spojnici. Penetračná hĺbka je v tomto prípade súčet ich polomerov mínus vzdialenosť ich stredov.

Guľa x kapsula - v prípade kontroly kolízie medzi kapsulou a guľou si ako prvé zistíme kde sa guľa nachádza relatívne ku kapsuly tak že si priestor rozdelíme na tri časti rovinami kolmými na spojnicu centier kapsule (stredov gulý na okrajoch kapsule) prechádzajúcimi týmito centrami, následne vypočítame v ktorej z oblastí sa stred našej gule nachádza, ak sa nachádza v pozitívnom polpriestore oboch rovín tak riešim kolíziu gule s guľou na jednom okraji kapsule ak sa nachádza v negatívnom polpriestore oboch rovín tak riešime kolíziu gule s guľou na druhom konci kapsule a ak je v negatívnom polpriestore jednej roviny a pozitívnom druhej (obraz stredu gule na spojnicu centier kapsule sa nachádza medzi centrami) tak riešime kolíziu gule s valcom ktorého výška je vzdialenosť centier kapsule, polomer je polomer gulí na okrajoch a

stred je stred spojnice centier.

Kapsula x kapsula - v tomto prípade sa postupuje do istej miery ako pri guli a kapsule tým, že najprv určujeme v ktorých oblastiach jedného objektu sa druhý nachádzať s tým rozdielom, že určujeme polohu oboch centier kapsúl vzájomne (poloha obe centrá jednej kapsule relatívne k druhej a vice versa). Následne na základe určenej vzájomnej polohy sa následne počíta buď kolízia gule s guľou, gule s valcom alebo valca s valcom.

Aproximate Convex Decomposition

Jedným z hlavných problémov pri riešení kolízii sú nekonvexné objekty, jedným zo spôsobov riešenie je približná konvexná dekompozícia ktorá sa snaží rozdeliť nekonvexný objekt sa množiny konvexných, respektíve takmer konvexných objektov (za takmer konvexných objekt považujeme objekt ktorého rozdiel objemu a objemu jeho konvexného obalu je pod určenou hranicou). Narozdiel od hierarchických obálok ktoré sa snažia v nižších vrstvách aproximovať nekonvexný objekt obálkami častí objektu ktoré je nutné prepočítavať sa konvexná dekompozícia väčšinou robí iba raz na začiatku simulácie.

Jednou z možných stratégii je použiť či už hierarchické OBB alebo BVH aby sme objekt rozdelili na veľké množstvo malých objektov pri ktorých už predpokladáme asi aproximatívnu konvexnosť, tieto objekty následne zoradíme podľa veľkosti a snažíme sa spájať dohromady kým ich spojenia spĺňajú aproximatickú konvexnosť.

Narrow Phase

Ako bolo spomenuté na začiatku táto fáza má viac úloh. Konkrétne ide o tri problémy a to:

- určiť ktoré z potenciálne kolidujúcich dvojíc objektov sú naozaj v kolízii
- určiť podrobnosti kolízie ako typ (dotyk alebo prienik), bod dotyku, hĺbku prieniku
- určiť či ide o novú kolíziu alebo pokračujúcu kolíziu ktorá existovala už v predchádzajúcom časovom kroku

Minkowského priestor

Jedným zo spôsobov ktorý sa v tejto fáze používa je Minkowského priestor, čiže priestor v ktorom si definuje tri špeciálne binárne operácie nad konvexnými telesami a to (A a B sú konvexné telesá a t je vektor):

- Minkowského súčet: $A \oplus B = \{a + b | \forall a \in A \wedge \forall b \in B\}$
- Minkowského rozdielu: $A \ominus B = \{a - b | \forall a \in A \wedge \forall b \in B\}$
- Minkowského posun: $A \oplus t = \{a + t | \forall a \in A\}$

Z týchto operácii je pre nás najdôležitejšia operácia rozdielu keďže táto operácia má jednu špeciálnu vlastnosť a to, že poloha počiatku priestoru relatívne k objektu ktorý vznikne rozdielom dvoch konvexných útvarov hovorí o kolízii daných objektov a to následovne:

- počiatok je mimo vzniknutého telesa: objekty nie sú v kolízii, pričom vzdialenosť od počiatku do najbližšieho bodu na povrchu telesa sa nazýva separačná vzdialenosť
- počiatok je na povrchu vzniknutého telesa: objekty sa dotýkajú
- počiatok sa nachádza vnútri vzniknutého telesa: objekty sú v kolízii a vzdialenosť počiatku od najbližšieho bodu na povrchu telesa je penetračnou hĺbkou kolízie

Na výpočet spomínaných vzdialeností (separačná alebo penetračná) sa počíta takzvaný dotykový vektor. Ide o najkratší vektor posunu ktorý posunie druhý objekt (myslený druhý objekt použitý pri výpočte Minkowského rozdielu objektu) tak, že sa objekty dotýkajú. Veľkosť tohto vektoru je vzdialenosťou ktorú chceme počítať (separačná alebo penetračná) a smer je smer v ktorom je táto vzdialenosť nadobudnutá.

Pre výpočet tohto vektora sa často používa veta o dotykovom vektore ktoré hovorí, že vektor posunie objekty do dotyku vtedy a práve vtedy keď leží na hranici ich Minkowského rozdielu. Takže sa problém mení z hľadania vektora posúvajúci jeden objekt do dotyku s druhým na problém hľadania najkratšieho vektoru ležiaceho na hranici ich Minkowského rozdielu.

Keďže je naivné hľadanie tohto vektoru prechádzaním povrchu telesa vzniknutého Minkowským rozdielom výpočtovo náročné a časovo zdĺhavé tak sa za týmto účelom využívajú rôzne aproximačné algoritme.

Aproximácia blízkosti konvexných telies

Jedným z bežných algoritmov je GJK algoritmus (Gilbert-Johnson-Keerthi Algorithm) ktorý iteratívne generuje simplexy buď pridaním bodu (vytvorím viacrozmerného simplexu avšak obmedzený počtom rozmerov priestoru) alebo vybraním podsimplexu obsahujúceho priebežne najbližší bod.

Pri určovaní ktorý podsimplex obsahuje priebežný bod sa dá robiť pomocou riešenie sústavy lineárnych rovníc čo je však výpočtovo náročné, lepším spôsobom je pomocou simplexu rozdeliť priestor na Voronoiove oblasti a následne určiť v ktorej oblasti sa nachádza počiatok priestoru, najbližší podsimplex je simplex prislúchajúcu tejto oblasti.

Iným príkladom algoritmu na určovanie blízkosti dvoch telies je V-Clip algoritmus (Voronoi Clipping Algoritm) tento algoritmus rozdelí povrch skúmaných objektov na črty (vrcholy, hrany, steny) a pre každú určí jej vonkajšiu Voronoiovu oblasť, čiže oblasť v ktorej pre každý bod platí, že sa nachádza mimo objektu a kolmá vzdialenosť bodu od danej črty je menšia ako vzdialenosť od hociktorej inej črty objektu.

Následne pri určovaní blízkosti objektov sa využíva veta o Voronoiových oblastiach ktorá hovorí, že ak máme dva nekolidujúce konvexné mnohosteny A a B, kde črta $X \in A$ a črta $Y \in B$ a body $x \in X$, $y \in Y$ sú najbližšie body týchto čŕt, tak body x a y sú najbližšie body celých objektov A a B práve vtedy keď bod x patrí vonkajšej Voronoiovej oblasti črty Y a bod y patrí vonkajšej Voronoiovej oblasti črty X.

V-Clip algoritmus túto vetu využíva tak, že začína s nejakou dvojicou čŕt objektov a následne iteratívne sa pozerá na okolité črty a vyberie z okolitých čŕt tak aby v každom kroku buď klesla vzdialenosť medzi skúmanými črtami alebo sa zmenšil súčet ich rozmerov. Keďže uvažujeme, že naše objekty majú iba konečný počet čŕt tento algoritmus skonverguje k riešeniu v konečnom čase. Pričom ak si z okolitých budeme vyberať tie s minimálnou vzdialenosťou (nie hociktoré z tých spĺňajúce podmienky pre ďalší krok) tento algoritmus väčšinou skonverguje po pomerne malom počte krokov.