Animace

Diplomová práce

Vypracoval: Irena Lálová
17.10.yyyy

Obsah

1.Úvod....1

2.Animace....2

2.1.Obecná animace a její počítačová podpora....2
2.2.Animační jazyky....3
2.2.1.Lineární seznam poznámek....3
2.2.2.Základní účelové jazyky....3
2.2.3.Grafické jazyky....3
2.3.Metody řízení animace....4
2.3.1.Plně explicitní řízení....4
2.3.2.Procedurální řízení....4
2.3.3.Systémy založené na omezeních....4
2.3.4.Sledování aktivní akce....5
2.3.5.Herci....5
2.3.6.Kinematika a dynamika....5
2.3.7.Fyzikálně založená animace....5
2.4.Základní pravidla pro vytváření animace....5
2.4.1.Zmáčknutí a roztažení....6
2.4.2.Zpomalování a zrychlování....6
2.4.3.Etapy....6
2.5.Problémy spojené s animací....6
2.5.1.Dočasný aliasing....6
2.5.2.Mezisnímková souvislost....6
2.5.3.Velká dočasná rozlišovací schopnost....7

3.Reprezentace scény....8

3.1.BSP-stromy....8
3.1.1.Reprezentace mnohostěnu....8
3.1.2.Generické BSP-stromy....9
3.1.3.Reprezentace regulární množiny....10
3.1.4.Vytváření BSP-stromu....10
3.1.5.Iterativní algoritmus vytváření BSP-stromu....11
3.1.6.Heuristiky pro výběr dělící nadroviny....11
3.1.7.Klasifikace bodu....12
3.2.Množinové operace na mnohostěnu....12
3.2.1.Regularizované množinové operace....13

3.2.2.Použití BSP-stromů ... 13
3.2.3.Převod CSG-reprezentace na BSP-strom ... 14
3.2.4.Hranice ... 15
3.2.5.Redukce BSP-stromu ... 16
3.3.Popis vstupního souboru ... 16
3.4.Implementace ... 17
3.4.1.BSP-strom ... 17
3.4.2.Vytváření BSP-stromu ... 17
3.4.3.Klasifikace bodu ... 18
3.4.4.Množinové operace ... 19
4.Zobrazení ... 20
4.1.Promítací metody ... 20
4.1.1.Rovnoběžné promítání ... 21
4.1.2.Středové promítání ... 21
4.1.3.Pohledové transformace ... 22
4.2.Prostorové ořezávání ... 22
4.3.Určování viditelnosti ... 23
4.3.1.Dělení algoritmů podle časové složitosti ... 23
4.3.2.Dělení algoritmů podle reprezentace těles ... 23
4.3.3.Prioritní seznam stěn ... 24
4.4.Stínování ... 25
4.4.1.Konstantní stínování ... 25
4.4.2.Stínování obecných ploch ... 25
4.5.Vržené stíny ... 25
4.5.1.Stínová tělesa ... 26
4.5.2.SVBSP algoritmus ... 26
5.Vytváření středních snímků ... 29
5.1.Interpolace pozice ... 29
5.2.Interpolace orientace ... 30
5.2.1.Kvaterniony ... 30
5.3.Úvod ... 34

1. Úvod

Animovat, vzato doslovně, znamená přivést k životu. Lidé obvykle považují animaci za synonymum pro pohyb. Tento pojem však zahrnuje všechny změny mající vizuální účinek. Máme tím na mysli v čase proměnnou pozici - pohybová dynamika, dále tvar, barvu, průhlednost, strukturu a texturu tělesa - aktualizační dynamika a změny osvětlení, pozice kamery, orientace a ohniska, jakožto i změny zobrazovacích metod.

Animace se široce využívá v zábavném průmyslu a masmédiích (počítačové hry, filmy, atd.), ale uplatňuje se rovněž ve výuce a v průmyslových aplikacích, jako jsou například řídící systémy nebo letové simulátory. Vědecké aplikace počítačové grafiky se souhrnně nazývají "vědecká vizualizace". Vizualizace není pouhá aplikace počítačové grafiky ve vědě a inženýrství. Vyžaduje totiž spolupráci i jiných oborů, jako je například zpracování signálů, výpočtová geometrie a teorie databází. Animace je obvykle generována simulací vědeckých jevů. Výsledkem simulace mohou být velké datové množiny reprezentující dvojdimenzionální nebo trojdimenzionální data, která jsou následně převáděna na obrazy tvořící animaci. V jiném případě může simulace generovat polohu a umístění zkoumaného tělesa, které pak musí být nějakým způsobem zobrazeno, aby byla vytvořena požadovaná animační sekvence.

Počítačová animace je velmi mladý rozsáhlý obor, který se nadále vyvíjí. Se stále zdokonalující se technikou rostou možnosti animace a její tvorba je jednodušší a rychlejší než byla dříve. Ve své diplomové práci jsem se zaměřila na část animace "vytváření středních snímků (interpolace mezi snímky)" s cílem, aby výsledný pohyb byl hladký a spojitý. Hlavní důraz je kladen na interpolaci otočení. Součástí návrhu je samozřejmě i výběr vhodné datové struktury, která by jednak reprezentovala zkoumanou scénu, hlavně by ale umožňovala její rychlé zobrazování (nejlépe v reálném čase) a to společně s určováním viditelnosti. Výsledný program by tedy měl provádět to, co se někdy nazývá "interpolací pozice kamery", tzn. pro zadané pozice kamery (a směru, kterým se pozorovatel dívá), dopočítat pozice mezi nimi a výslednou sekvenci pohledů na scénu, tak jak je vidět z těchto pozic, zobrazit. Pro popis scény a animace je použit jednoduchý jazyk, který by se dal zařadit do třídy "Lineární seznam poznámek" ..... odkaz. Řízení animace je plně explicitní. Implementace byla provedena v programovacím jazyce Borland C++.

2. Animace

Tradiční, nepočítačová, animace je samostatný obor a nebudeme zde probírat všechny její postupy. Zaměříme se především na základní pojmy počítačově založené animace. Začneme u obecné animace, kde si ukážeme, jak lze využít počítače k jejímu vytváření. Dále se zaměříme na animaci, která je vytvářena z velké části na počítači. Velmi důležité je také vhodně popsat a řídit vytváření animační sekvence, proto se zmíníme o základních animačních jazycích a metodách pro řízení animace. Na závěr uvedeme důležitá pravidla pro animaci a hlavní problémy, které jsou s ní spjaty.

Některé z animačních technik, které zde budou uvedeny, byly již částečně nebo zcela implementovány pomocí hardwaru. Jedná se o architektury poskytující základní animaci v reálném čase, které jsou základem například pro stavbu letových simulátorů a jiných řídících systémů pracujících v reálném čase.

2.1. Obecná animace a její počítačová podpora

Tvorbu obecné animace [16], [19] lze popsat následující posloupností kroků:

- Napsání "příběhu" pro animaci.
- Vytvoření nástinu animace[1] ve formě sekvence skic ukazující základní strukturu a ideje animace.
- Nahrání zvukové stopy (pokud ji animace obsahuje).
- Propojení odpovídajících obrázků a zvuku.
- Nakreslení přesných klíčových snímků[2] - snímků, ve kterých jsou animované entity v extrémních nebo charakteristických pozicích. Z těchto snímků se vychází při určování dalších pozic.
- Vykreslení středních snímků[3].
- Vytvoření zkušebního filmu[4].
- Transformace snímků zkušebního filmu na buňky[5] výsledného filmu (ručně nebo fotokopírováním). Při tvorbě buněk lze použít jeden plán jako pozadí - neměnné a jiný jako popředí, které se v čase mění.
- Vybarvení jednotlivých buněk a seřazení do správného pořadí.
- Nasnímání výsledného filmu.

[1] storyboard
[2] key frames
[3] inbetweening
[4] pencil test
[5] cells

Celý proces vytváření animace je ve své podstatě sekvenční. Obvykle, zvláště pokud je zpracováván na počítači, je však iterativní - konečný vzhled animace například vyžaduje rozšíření některých sekvencí, přihrání dalších částí zvukové stopy, apod.

Mnoho etap obecné animace přímo vybízí k použití počítačové podpory, zejména proces vytváření středních snímků a vybarvování. Dříve však, než může být použit počítač, je nutné kresby digitalizovat - sejmout pomocí scanneru, digitalizačního tabletu nebo vytvořit pomocí kreslícího programu. Takto získaná kresba někdy potřebuje ještě následně upravit - odfiltrovat šum, zaostřit hrany, apod.

2.2. Animační jazyky

Existuje řada různých jazyků pro popis animace a další nové stále vznikají. Mnoho animačních jazyků splývá s jazyky modelovacími, takže popisují jak tělesa v animaci, tak i vlastní animaci těchto těles. Animační jazyky lze rozčlenit do následujících tří skupin:

2.2.1. Lineární seznam poznámek

Každá událost v animaci je popsána číslem počátečního a koncového snímku a akcí, která má být provedena. Jednotlivé akce navíc obvykle ještě mají parametry, takže například příkaz

```
42, 53, B ROTATE "PALM", 1, 30
```

znamená "mezi snímky 42 a 53 otoč těleso nazvané PALM okolo osy 1 o 30 stupňů". Písmeno B určuje interpolační metodu (například tabulku hodnot), která se má použít k rozložení rotace mezi jednotlivé snímky. Neboť příkazy definují individuální akce s přesně určenými čísly snímků, je jejich pořadí nezávislé. To už ale nemusí platit v případě provádění dvou operací ve stejný časový okamžik.

Jako příklad lze uvést jazyk Scefo (SCEne FOrmat) [36].

2.2.2. Základní účelové jazyky

Jinou možností, jak popsat animaci, je vložit animační schopnosti do nějakého základního programovacího jazyka (např. LISP) [30], [38], [22]. Hodnoty proměnných jazyka pak mohou být použity jako parametry pro různé procedury, které vytváří skutečnou animaci. Možnosti těchto jazyků jsou velké, avšak většinou vyžadují značné programátorské zkušenosti na straně uživatele.

Příkladem jazyka této skupiny je například ASAS [30].

2.2.3. Grafické jazyky

Jistou nevýhodou výše uvedených textových jazyků je, že pouhým pohledem do scriptu nezískáme žádnou představu o tom, jak bude výsledná animace vypadat. Grafické animační jazyky popisují animaci vizuálněji. Tyto jazyky se používají pro vyjadřování, editování a pro celkové pochopení současných změn v animační sekvenci. Základní myšlenkou těchto jazyků je nahrazení textového paradigma vizuálním - místo explicitního popisu akcí, animátor vytváří obraz akcí.

Například diagramatický animační jazyk DIAL [7] zobrazuje sekvence událostí v animaci jako sérii paralelních řádek značek. Vertikální čára označuje začátek akce a pomlčky čas, po který se má akce provádět.

2.3. Metody řízení animace

Řízení animace je něco zcela nezávislého na jazyce, který ji popisuje. Většinu řídících mechanismů lze adaptovat na rozličné typy jazyků. Mechanismy řízení animace sahají od plně explicitního řízení, kde animátor explicitně popisuje pozici a atributy pro každý objekt ve scéně, až po vysoce automatizované řízení.

2.3.1. Plně explicitní řízení

Explicitní řízení je nejjednodušší kategorie řízení animace. Animátor zde popisuje vše, co se v animaci objevuje. Buď specifikuje jednoduché změny jako je například změna měřítka, posunutí, rotace nebo udává informace k jednotlivým klíčovým snímkům spolu s interpolačními metodami, které se mezi snímky mají použít. Tyto interpolace mohou být zadány explicitně nebo pomocí interaktivních systémů - myši, joysticku, hmatových rukavic aj.

Příkladem této kategorie je například systém BBOP [35].

2.3.2. Procedurální řízení

Jedná se o procedurální modely, ve kterých různé elementy modelu komunikují tak, aby určily své vlastnosti. Například lze takto modelovat vztah tráva - vítr [29].

2.3.3. Systémy založené na omezeních

Některé objekty ve fyzickém světě se pohybují po přímkách, ale velké množství těles se pohybuje v závislosti na jiných tělesech se kterými jsou v kontaktu a tento pohyb již lineární být nemusí. Tyto pohyby lze modelovat pomocí omezení. Tímto omezením může například být, že míček padající na plochu musí stále zůstat na její jedné straně - nemůže projít skrz.

Do této třídy patří například Sketchpad systém od Sutherlanda [37] nebo metasystém ThingLab od Borninga [3].

2.3.4. Sledování aktivní akce[6]

Trajektorie těles v animaci lze rovněž generovat sledováním aktivní akce. Existuje mnoho metod sledování. Tradiční animace používá "rotoscoping". V této metodě se nejdříve vytvoří film, ve kterém animační hrdiny představují lidé. Animátor dále film překreslí, rozšíří pozadí a nahradí lidské herce jejich animovanými ekvivalenty. Touto technikou lze získat velice realistické pohyby. Jindy se na klíčové body lidského těla připojí nějaký typ indikátoru a

[6]Tracking Live Action

sledují se pozice těchto indikátorů. Tato technika byla například použita pro vytvoření grafické loutky [11].

2.3.5. Herci[7]

Používání "herců" je procedurální řízení vysoké úrovně. Herci v animaci jsou malé prográmky, které se vyvolávají jedenkrát pro každý snímek, aby určily vlastnosti určitého objektu v animaci. V objektově orientovaném programování herec odpovídá objektu.

Herci byli odvozeni z podobného pojetí ve Smalltalku [14] a jiných jazycích.

2.3.6. Kinematika a dynamika

Kinematika se vztahuje k pozici a rychlosti bodů. Naproti tomu dynamika bere v úvahu i fyzikální zákony (Newtonovy pohybové zákony, Euler-Lagrangeovy rovnice apod.). V některých případech potřebujeme úlohy kinematiky a dynamiky obrátit (jakou sílu musíme použít, aby se daná krychle posunula do bodu A?). Pro jednoduché systémy může mít tato třída úloh jednoznačné řešení, obvykle ale musíme ještě přidat další omezení.

Problémy tohoto typu se velmi často vyskytují při animaci lidských postav [34], [12],[42].

2.3.7. Fyzikálně založená animace

Jedná se zejména o fyzikálně založené modely oblečení, plasticity a pohybu pevných těles. Tyto modely obvykle vedou k řešení soustav parciálních diferenciálních rovnic. Musíme také vzít v úvahu, že síly, které se podílejí na pohybu, nemusí být vždy předem známy - objekty mohou používat vlastní zdroje, například svaly.

2.4. Základní pravidla pro vytváření animace

V průběhu doby, jak se animace vyvíjela, vznikala také základní pravidla pro vytváření animace, která se používají dodnes [19], [18]. Uvedeme si zde pouze pár nejdůležitějších zásad. Je nutné ale zdůraznit, že tato pravidla nejsou absolutní, mnoho moderních animátorů se nedrží těchto tradičních zásad a přesto dosahují vynikajících výsledků [20], [21].

2.4.1. Zmáčknutí a roztažení[8]

Jedná se o jednoduché, ale nejdůležitější z tradičních pravidel, které se používá k určování fyzikálních vlastností tělesa pomocí deformace tvaru. Například koule z gumy se při dopadu na zem deformuje, oproti tomu koule ze železa se deformuje málo, ale může se při dotyku se zemí zakymácet.

[7] Actors

[8] squash and stretch

2.4.2. Zpomalování a zrychlování[9]

Toto druhé pravidlo slouží k získání hladké interpolace, neboť trhavé pohyby ruší celkový dojem animační sekvence. Velmi důležité je dodržování tohoto pravidla zejména při interpolaci polohy kamery.

2.4.3. Etapy

Ať se jedná o zábavní průmysl nebo o vědeckou vizualizaci, je důležité vhodně rozdělit jednotlivé akce do etap. Tím máme na mysli výběr pohledu, který nejlépe zahrnuje nejdůležitější informace o událostech v animaci a rovněž izolaci událostí tak, aby pouze jedna věc v daném okamžiku poutala pozornost pozorovatele (u vědeckých simulací to nemusí být vždy možné).

2.5. Problémy spojené s animací

Stejně jako při přesunu ze dvou dimenzí do tří, tak i při přechodu ze tří do čtyř dimenzí (přidáním jedné dimenze - času) se objevují nové problémy.

2.5.1. Dočasný aliasing

Tento problém lze řešit podobně jako ve dvou dimenzích tak, že zvýšíme dočasné rozlišení.

2.5.2. Mezisnímková souvislost

V ideální animaci se obrázky mezi jednotlivými snímky mění velice málo. Pokud by tomu tak nebylo, výsledná animační sekvence by byla trhaná. Využitím této mezisnímkové souvislosti se dá ušetřit mnoho času.

2.5.3. Velká dočasná rozlišovací schopnost

Filmy se obvykle vytváří s rozlišovací schopností 24 fps (snímků za sekundu). Při tomto rozlišení se některé pohyby, zejména pokud se velké objekty pohybují velkou rychlostí v blízkosti pozorovatele, mohou jevit jako trhané. Všeobecně je také znám problém kol u vagónu, která občas vypadají jako by se točila pozadu. Tyto problémy se řeší zvýšením počtu snímků za sekundu. Například technologie popsaná v [23] umožňuje vytvářet filmy s rozlišovací schopností 60 fps na 70-ti mm film.

[9] slow-in and slow-out

3. Reprezentace scény

Při animaci, zejména pokud generujeme animační sekvenci v reálném čase, je velmi důležitá volba vhodné datové struktury pro reprezentaci scény. V této práci jsou použity BSP-stromy, zvláště pro své velké výhody při počítání viditelnosti mezi statickou skupinou polygonů tak, jak jsou vidět z libovolného bodu pohledu. Reprezentace mnohostěnu a počítání viditelnosti ovšem nejsou jediná možná využití této struktury. Pomocí BSP-stromů se dá provádět i ořezávání, klasifikace bodu a množinové operace na mnohostěnu. V neposlední řadě se dá pomocí těchto stromů implementovat i jednoduchý algoritmus na určování vržených stínů. Na závěr této kapitoly bude uvedena definice vstupního souboru pro zadání určité scény a popis implementace výše uvedených algoritmů.

3.1. BSP-stromy

BSP-stromy (Binary Space-Partitioning trees) patří do třídy reprezentací, které dělí vstupní prostor[10]. Základní myšlenkou reprezentací této skupiny je dekompozice tělesa na kolekci přilehlých neprotínajících se těles, která jsou jednodušší než původní těleso a obecně nemusí být stejného typu. Dalším reprezentantem této třídy jsou například velmi populární osmistromy.

Osmistromy dělí prostor pomocí nadrovin, které jsou navzájem kolmé a půlí na každé úrovni stromu všechny dimenze. BSP-stromy jsou binární stromy, které také rekursivně dělí prostor na dvojice podprostorů, ale pomocí nadroviny libovolné orientace a pozice. Každý vnitřní uzel BSP-stromu je spojen s nadrovinou, která dělí prostor na dva podprostory. Pokud budeme uvažovat normály směřující ven z tělesa, pak levý potomek bude odpovídat podprostoru "za" a pravý podprostoru "před" rovinou. Tyto podprostory dále dělíme až je výsledný prostor homogenní. Ten pak odpovídá listům stromu reprezentujících IN (uvnitř) a OUT (vně) buňky. Musíme také vzít v úvahu přesnost, s jakou se operace počítají. Proto budeme uvažovat, že nadroviny v uzlech mají určitou "tloušťku". Nutno poznamenat, že tato reprezentace je nezávislá na dimenzi prostoru.

Příklady v této kapitole jsou pro názornost pouze dvoudimenzionální. Výsledky použití popisovaných algoritmů na třírozměrná tělesa jsou uvedeny v příloze ....

3.1.1. Reprezentace mnohostěnu

Studium mnohostěnů má dlouhou historii a věk počítačů mu dal znovu význam v nových disciplínách jako je například počítačová geometrie, geometrické modelování nebo robotika a počítačové vidění. Tato atraktivita pramení v relativní jednoduchosti lineárních výpočtů oproti výpočtům nelineárním. V případech, kdy záleží na rychlosti výpočtu (a není zapotřebí

[10] spatial-partitioning representations

velká přesnost), může být použitím aproximací nelineárních těles docíleno velmi efektivních alternativ nelineárních výpočtů.

Mezi nejpřednější metody reprezentace mnohostěnů v poslední době patří hraniční reprezentace[11] (B-rep.). Ta vychází z představy, že nejvýznamnější částí tělesa je jeho povrch, který tvoří hranici mezi hmotou tělesa a okolním prostorem. *D*-dimenzionálni mnohostěn je reprezentován množinou (*d-1*)-dimenzionálních mnohostěnů, které se nazývají stěny. Stěny jsou dále reprezentovány (*d-2*)-mnohostěny až po (*d=0*)-mnohostěn - vrchol. Alternativní reprezentací konvexních mnohostěnů je volumetrický přístup, kde je mnohostěn definován jako průnik množiny poloprostorů.

Reprezentace pomocí BSP-stromu je volumetrická reprezentace obecného lineárního mnohostěnu. Obecný mnohostěn může obsahovat díry a libovolné množství připojených komponent (oblasti napojení mohou být i oblasti bez vnitřku, jako například bod). Vnitřek obecného mnohostěnu nemusí být úplně ohraničený - může být nekonečný. Nelistové uzly BSP-stromu jsou pak označeny nadrovinami a listové uzly odpovídají buňkám konvexního dělení *d*-prostoru.

V následujícím textu se rovněž budeme odkazovat na reprezentaci pomocí CSG-stromu (Constructive Solid Geometry Representation). V této reprezentaci vnitřní uzly stromu reprezentují (regularizované) množinové operace a listy jsou instance primitivních těles (jako například krychle, kužel aj.).

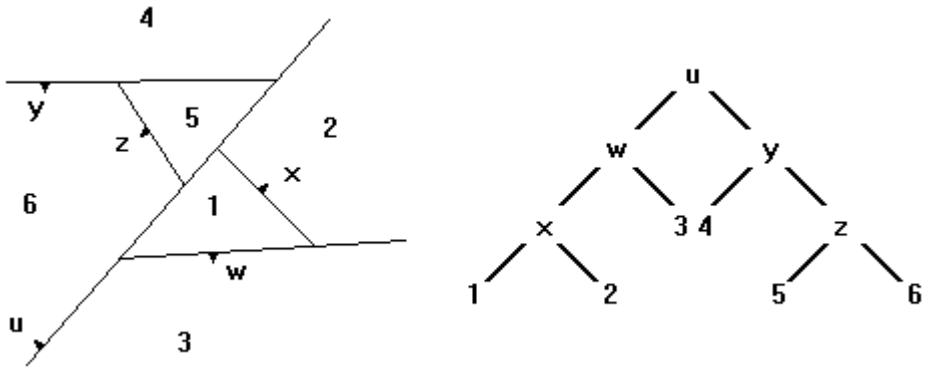
3.1.2. Generické BSP-stromy

Při rekurzivním dělení *d*-dimenzionálního prostoru dělíme určitý podprostor libovolnou nadrovinou, která protíná jeho vnitřek. Jako výsledek dostaneme dva nové podprostory, které mohou být dále děleny. Rozdíl mezi těmito dvěma prodprostory vytvořenými nadrovinou je dán orientací normálového vektoru nadroviny (obvykle je jedno, kterou orientaci normálového vektoru zvolíme). Obrázek č. .... ukazuje vztah mezi dělením prostoru a odpovídajícím BSP-stromem.

Generický BSP-strom je možné rozšířit přidáním dalších množin do struktury stromu:

- Listy mohou obsahovat kolekci množin (objektů) obsažených kompletně uvnitř odpovídající buňky.
- Interní uzly mohou obsahovat množiny ležící na odpovídající nadrovině.

[11] Boundary Representation



Označení:

H	.....	nadrovina;
H^+	.....	pravý poloprostor, přední , ve směru normály (tj. po dosazení do rovnice nadroviny získáme hodnotu >0) ;
H^-	.....	levý poloprostor, zadní (tj. <0);
v	.....	uzel;
$R(v)$	.....	oblast prostoru reprezentovaná uzlem v (odpovídá průsečíku otevřených poloprostorů na cestě z kořene do v). Tato oblast je vždy konvexní, neprázdná a nemusí být kompletně ohraničená - je topologicky otevřená.
H_V	.....	dělící nadrovina odpovídající uzlu v;
$SHp(v)$	...	část nadroviny[12] H_V (její průsečík s $R(v)$). Tyto části nadrovin příslušející k uzlům na cestě z kořene do v obsahují hranici $R(v)$.

3.1.3. Reprezentace regulární množiny

Nejprve si uvedeme definici regulární množiny. Množina je regulární, pokud je uzávěrem svého vnitřku ($S=cl(int\ S)$). Reprezentace regulární množiny pomocí BSP-stromu je velice jednoduchá. Každé listové buňce přiřadíme atribut IN (uvnitř množiny) nebo OUT (mimo množinu). Protože každá buňka je otevřená a z definice $R(v)$ víme, že je neprázdná, lze sestrojit sjednocení všech IN buněk, jehož uzávěrem je reprezentovaná regulární množina. Hranice množiny je tvořena body mezi IN buňkami a OUT buňkami a všechny takovéto body leží v částech nadrovin stromu.

3.1.4. Vytváření BSP-stromu

Při vytváření BSP-stromu budeme vycházet z množiny stěn tvořících jeden nebo více disjunktních mnohostěnů. Stěnou označujeme *(d-1)*-dimenzionální hranici *d*-mnohostěnu. H_f odkazuje na nadrovinu obsahující stěnu *f*. Dále budeme také uvažovat, že normály stěn směřují ven z tělesa. Toho lze docílit tím, že budeme vyžadovat orientaci stěn (pořadí jejích bodů) proti směru hodinových ručiček.

[12] sub-hyperplane

V každém kroku rekursivní proces vybere nadrovinu H, která rozdělí aktuální množinu stěn F do tří množin - $F(H^+)$, $F(H^-)$ a $F(H)$, odpovídajících podprostorům H^+, H^- a H. Dělení stěny $f \in F$, je definováno jako výsledek vytváření následujících množin, z nichž jedna nebo dvě mohou být prázdné:

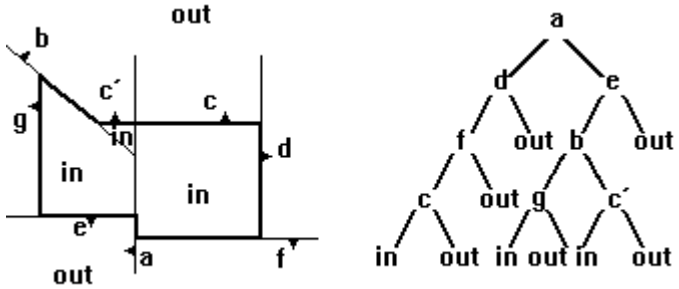
$$f^+ = cl(H^+ \cap int\, f)$$

$$f^- = cl(H^- \cap int\, f)$$

$$f^0 = cl(int(H \cap int\, f))$$

Množina $F(H)$ zůstane v novém uzlu a proces pokračuje rekursivně na druhých dvou množinách, dokud seznam aktuálních stěn není prázdný. Jednou možnou nevýhodou tohoto algoritmu je, že každý mnohostěn je dekomponován na množinu konvexních oblastí (IN buněk). Počet těchto dekompozic závisí na volbě dělící nadroviny při vytváření stromu. Vhodným výběrem této nadroviny se tato nevýhoda dá částečně odstranit. Různé heuristiky pro výběr dělící nadroviny, kterými se tato nevýhoda dá částečně odstranit, budou uvedeny později.

Abychom docílili toho, že všechny body hranice mnohostěnu musí ležet na částech nadrovin výsledného BSP-stromu, budeme dělící nadroviny vybírat z aktuální množiny stěn. Druhým požadavkem je správná klasifikace buněk. Listu přiřazujeme hodnotu pouze v případě, když dělením množiny stěn dostaneme stěny pouze na jedné straně od dělící nadroviny. V tomto případě víme, že se jedná o homogenní oblast, tj. oblast ležící buď celá uvnitř nebo celá vně daného mnohostěnu. Tudíž oblast tvořící buňku lze klasifikovat jako IN nebo OUT. Z toho, že dělící nadroviny vybíráme z množiny aktuálních stěn a že uvažujeme normály těchto stěn směřující ven z tělesa můžeme usoudit, že levý list musí být IN buňka (tj. leží v zadním poloprostoru stěny) a pravý list OUT buňka (tj. leží v předním poloprostoru). Na obr. ... je uveden BSP-strom vytvořený uvedeným algoritmem z konkávního polygonu.



Způsobem výběru dělících nadrovin se snažíme optimalizovat především velikost a vyváženost výsledného stromu. Hledání optimálního výběru nadroviny by bylo výpočetně velmi náročné, proto je vhodné použít heuristiky. Pro množinu stěn definujeme množinu kandidátů. Tu budou tvořit stěny, které jsou uvažovány za potenciální nadroviny. Testovací množina bude obsahovat stěny, vůči kterým bude každý kandidát testován. Tento test může mít tři různé výsledky: před, za, protíná. Heuristika potom bude funkcí počtu výskytů jednotlivých výsledků tohoto testu. Vybrána je ta stěna z množiny kandidátů, která maximalizuje danou heuristiku.

Možné heuristiky:

$$Heur_1(front, back, split) = (-|\ back - front\ |) - w_{split} * split$$

$Heur_2(front, back, split) = (front * back) - w_{split} * split$

$Heur_3(front, back, split) = front - w_{split} * split$

Váha w_{split} umožňuje "řídit" heuristiku. Důvod, proč protnutým kandidátům přiřadit zápornou váhu je ten, že dělení stěn zvyšuje velikost stromu a tím i celkový výpočetní čas. První dvě heuristiky se snaží vyrovnat počet stěn na každé straně kandidáta. Třetí je motivována CSG-stromem s konvexními primitivními tělesy a snaží se maximalizovat počet stěn vně nějakého primitiva, což ulehčuje eventuální zjednodušování stromu. Výsledky použití jednotlivých heuristik jsou uvedeny v příloze .....

3.1.5. Klasifikace bodu

Problém klasifikace bodu se dá specifikovat následovně. Pro danou množinu *S* a bod *p* určete, zda leží v *int S*, *ext S* nebo *bd S* (ve vnitřku *S*, vnějšku *S* nebo na hranici *S*), za předpokladu, že *S* je regulární.

Řešení problému je rekursivní proces, který začíná od kořene stromu a pokračuje směrem k listům. K řízení rekurze slouží pozice bodu vzhledem k nadrovině. Pro vyřešení problému potřebujeme zjistit, kdy je okolí bodu homogenní - tudíž IN nebo OUT, nebo nehomogenní - tudíž na hranici. Pokud bod leží v listové buňce, pak o jeho okolí víme, že je homogenní. Jestliže *p* leží v nějaké nadrovině, proces musí pokračovat hledáním v obou podstromech, abychom určili všechny buňky, ve kterých uzávěr *p* leží. Pokud hodnoty těchto buněk nejsou stejné, pak *p* leží na hranici, jinak víme zda leží IN nebo OUT podle získané hodnoty. Nesmíme opomenout přidat nadrovinám určitou "tloušťku", tj. definovat určitý interval okolo nuly, který je mapován na nadrovinu. Strojová přesnost určuje dolní hranici tohoto intervalu.

Tento problém se dá řešit pro tři dimenze (*d=3*) v čase *O(n)* pro hraniční reprezentaci s *n* stěnami. Pro libovolný BSP-strom sestrojený z *n* stěn tento čas může být nejvíce *O(n)* (nejvyšší známá horní hranice velikosti takovéhoto stromu je $O(n^d)$ [24]. Neboť velikost vyváženého BSP-stromu je *O(n)* (pro danou množinu stěn jeho sestrojení nemusí být vždy možné), potom tento problém lze řešit v *O(log n)*.

3.2. Množinové operace na mnohostěnu

Jedním z nejmocnějších prostředků pro modelování tvaru těles jsou množinové operace sjednocení, průnik a rozdíl těles. Každá z těchto operací je do jisté míry obdobou určité technologické operace, pomocí které jsou vytvářeny skutečné výrobky.

Provádění množinových operací u těles určených hraniční reprezentací je poměrně složité. Množinové operace mění totiž podstatným způsobem tvar těles a dochází k rušení a vytváření nových vrcholů, hran a ploch. Oproti tomu CSG-strom je sám o sobě strukturou, která popisuje vznik tělesa pomocí množinových operací. Provést takovou operaci se dvěma tělesy tedy znamená připojit jim odpovídající CSG-stromy do nově vzniklého kořenového uzlu. My si ukážeme provádění množinových operací pomocí BSP-stromů.

3.2.1. Regularizované množinové operace

Z hlediska modelování těles je důležité, aby výsledkem množinových operací prováděných s tělesy bylo opět těleso, případně prázdná množina. Je zřejmé, že při různých polohách operandů nemusí být tato podmínka splněna. Výsledkem může být pouze společná plocha, hrana nebo dokonce jen jeden vrchol. Při modelování těles se proto používají tzv. regularizované množinové operace, které vždy vytvoří pevné těleso, nebo prázdnou množinu. Regularizované množinové operace se značí následovně: $\cap^*$, $\cup^*$, $-^*$, $\sim^*$ (průnik, sjednocení, rozdíl a doplněk).

Každé těleso lze definovat jako množinu bodů. Tato množina se dá rozdělit na body vnitřní a hraniční. Hraničními body rozumíme ty body, jejichž vzdálenost od tělesa a jeho doplňku je nulová. Hraniční body ve své podstatě nemusí být částí tělesa (například otevřená množina neobsahuje žádný svůj hraniční bod). Vnitřní body jsou pak doplňkem tělesa vzhledem k bodům hraničním.

Regularizace dané množiny je definována jako uzávěr vnitřních bodů množiny. Z toho tedy vyplývá, že neobsahuje hraniční body, které nejsou přilehlé k nějakému vnitřnímu bodu.

$$A\ op^*B = cl(int(A\ op\ B))$$

3.2.2. Použití BSP-stromů

Protože se zabýváme regulárními množinami (regulární množina je rovna své regularizaci), budeme uvažovat pouze regularizované množinové operace.

Nejprve budeme uvažovat unární operátor doplňku. Pro daný BSP-strom, který reprezentuje množinu S, lze BSP-strom reprezentující doplněk této množiny jednoduše vytvořit aplikováním doplňku na hodnoty listových buněk. To znamená, že všechny IN buňky se změní na OUT a naopak. Musí se tedy změnit orientace každého hraničního polygonu ve vnitřních uzlech stromu. Z toho vyplývá, že provést doplněk hraniční reprezentace lze pouhým otočením orientace všech stěn.

Pro vyhodnocování binárních operátorů použijeme zjednodušování výrazů v geometrickém smyslu. Uvažujme například výraz $S_1 \cap^* S_2$. Pokud zjistíme, že pro nějakou oblast R platí $R \subseteq ext\ S_2$, pak v R může být výraz zjednodušen na $S_1 \cap^* \varnothing$, kde $\varnothing$ označuje prázdnou množinu. Naproti tomu, pokud $R \subseteq int\ S_2$, výraz se redukuje na $S_1 \cap^* U_E$, kde U_E je universální množina zúžena na R. V obou případech můžeme provést zjednodušení aniž bychom něco věděli o struktuře S_1 v R, která může být libovolně komplexní. Analogické případy existují pro ostatní operace a jsou uvedeny v tabulce 1. V kontextu CSG-stromů lze tento postup označit jako "prořezávání"[13].

[13] pruning

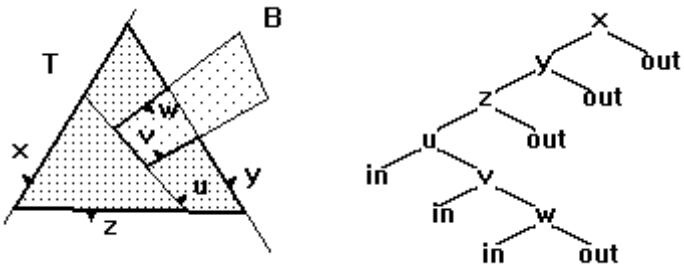
Operace	Levý operand	Pravý operand	Výsledek
$\cup^*$	S	in	in
	S	out	S
	in	S	in
	out	S	S
$\cap^*$	S	in	S
	S	out	out
	in	S	S
	out	S	out
$-^*$	S	in	out
	S	out	S
	in	S	$\sim^*$S
	out	S	out

Tabulka 1: Zjednodušování výrazů

Abychom mohli využít tuto techniku, musíme prostor rozdělit na takové oblasti, ve kterých bude alespoň jeden operand homogenní. To znamená, pro daný výraz S_1 *op* S_2 , definovaný na nějakém prostoru, hledáme takové dělení, že pro každou oblast R_i dělení platí: $R_i \subseteq int\ S_j$ nebo $R_i \subseteq ext\ S_j$ j=1,2.. Ke splnění těchto podmínek lze využít BSP-stromy. Pro vyhodnocování výrazu $S_1 -^* S_2$ použijeme jeho ekvivalent $S_1 \cap^* \sim^* S_2$.

3.2.3. Převod CSG-reprezentace na BSP-strom

Jak již bylo uvedeno, CSG reprezentace množiny *S* je binární strom, ve kterém vnitřní uzly



reprezentují regularizované množinové operace a listy jsou instance primitivních těles.

Uvažujme například klasifikaci množiny *s* vzhledem k *S*. Při reprezentaci CSG-stromem lze množinu klasifikovat tak, že ji budeme klasifikovat vzhledem ke každému primitivnímu tělesu a jednotlivé výsledky kombinovat v závislosti na množinovém výrazu, který CSG-

strom reprezentuje. Alternativou je, převést CSG reprezentaci na BSP-strom a množinu *s* klasifikovat vzhledem k této struktuře.

Následující algoritmus je velmi podobný klasickému algoritmu na vytváření BSP-stromu. Jediným rozdílem je, že nemáme jako vstup jednoduchý seznam stěn, ale CSG-strom, který obsahuje stěny v listech. V každém kroku opět vybíráme dělící nadrovinu *H* podle heuristiky. Dále jsou generovány dvě kopie CSG-stromu a modifikovány tak, aby reprezentovaly množinu poloprostorech vzniklých dělením podle *H*. Jinými slovy, stěny základních těles nahradíme podmnožinou stěn ležících v daném poloprostoru. Při nalezení homogenních oblastí můžeme CSG-strom zjednodušit podle tabulky 1. Pokud se v určité oblasti CSG-strom zjednoduší na pouhou hodnotu IN/OUT, problém je v dané oblasti vyřešen a lze ho reprezentovat listovou buňkou BSP-stromu. Celý problém je tedy řešen vytvářením a hledáním homogenních oblastí, které reprezentují jednotlivé listy výsledného BSP-stromu. Na obrázku ... je zobrazeno výsledné dělení a BSP-strom po provedení výrazu $T -^{*} B$.

Chyba! Záložka není definována.**Hranice**

Výše uvedený algoritmus generuje BSP-strom, který je vhodný pro klasifikaci bodu nebo pro algoritmus sledování paprsku. Stěny v interních uzlech už ale neodpovídají hranici reprezentované množiny *S*. Protože hraniční reprezentace je vhodná pro vykreslování polygonu a BSP-strom poskytuje prioritní pořadí stěn, budeme chtít tyto hraniční stěny *S* generovat.

Lze ukázat, že hranice výsledku libovolné množinové operace je podmnožinou hranic operandů. To znamená, že hranice množiny *S* je celá obsažena v částech nadrovin výsledného BSP-stromu. Pouze části původních stěn, které leží v těchto nadrovinách mohou tvořit hranici množiny *S*. Abychom našli příslušné části původních stěn uchovávaných v určitém vrcholu *v*, vložíme tyto stěny do obou podstromů *v*. Tímto postupem získáme množinu nových stěn, podmnožina kterých tvoří hranici *S*.

Při klasifikaci stěn hledáme body ležící mezi OUT buňkou a IN buňkou, neboť to jsou právě body tvořící hranici množiny *S*. Stěny uchovávané ve vrcholu *v* si rozdělíme na dva seznamy - na stěny se stejnou orientací jako má dělící nadrovina a s opačnou orientací. Stěny ze seznamu se stejnou orientací vkládáme nejprve do levého (IN) podstromu vrcholu *v*, a ty části, které jsou klasifikovány jako IN se dále vloží do pravého (OUT) podstromu. Stěny, které jsou výsledně klasifikovány jako OUT, leží mezi IN buňkou a OUT buňkou a tudíž tvoří hranici množiny *S*. Při vkládání stěn ze seznamu s opačnou orientací je pořadí vkládání opačné - nejprve do pravého podstromu a pak do levého. Tímto dostaneme části hranice množiny *S*, které mají opačnou orientaci než daná nadrovina.

Pokud budeme uvažovat inkrementální algoritmus, můžeme využít skutečnost, že je vždy vyhodnocována pouze jediná množinová operace. Lze tak odstranit vkládání stěn do obou podstromů. Uvažujme například sjednocení. Víme, že okolí zadního (IN) poloprostoru stěny obou operandů tvoří vnitřek výsledku. Tudíž a priori víme, že stěny se stejnou orientací jako má nadrovina vložené do levého (IN) podstromu leží celé v IN buňkách a podobně stěny s opačnou orientací vložené do pravého podstromu. Proto každá stěna musí být klasifikována pouze vzhledem k jednomu podstromu: stěny se stejnou orientací vzhledem k pravému a s opačnou orientací vzhledem k levému podstromu. Výsledné stěny, které leží v OUT buňkách leží na hranici množiny. Pro průnik lze provést obdobnou analýzu, ze které vyplývá, že stěny

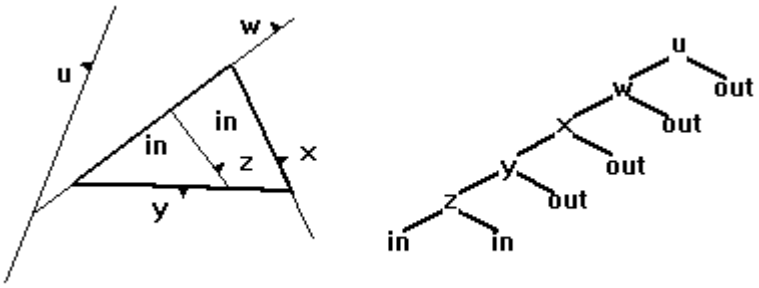
se stejnou orientací stačí vkládat do levého podstromu a stěny s opačnou orientací než je orientace nadroviny do pravého a uchováváme výsledné stěny, které leží v IN buňkách.

Tato technika nám sice zaručuje, že sjednocení ponechaných stěn tvoří hranici *S*, ale nezaručuje nám, že tato množina stěn v každém uzlu je disjunktní. Pokud stěny mají stejné atributy (např. barva), tato redundance se projeví ve vzrůstu časových a prostorových nároků. Tuto redundanci lze eliminovat spojením stěn, tj. vytvořením nezávislého sjednocení stěn se stejnou orientací a stěn s opačnou orientací pro každý uzel.

3.2.4. Redukce BSP-stromu

Jakmile máme zkonstruovaný BSP-strom, který reprezentuje výsledek vyhodnocení množinových operací, je možné tento strom dále zredukovat zrušením uzlů, které nemají vliv na reprezentovanou množinu. První takový případ nastává, pokud oba podstromy vrcholu mají stejnou hodnotu. Protože je *R(v)* homogenní, můžeme podstromy s kořenem *v* nahradit buňkou se stejnou hodnotou. Uveďme, že v nadrovině takovéhoto uzlu nemůže ležet žádná stěna hranice. Tento případ může vzniknout při zjednodušování výrazů, pokud nehomogenní oblast je zjednodušena na homogenní. Na obr. ... můžeme podle tohoto pravidla zrušit uzel *z*. Toto zjednodušení lze provádět jako součást konstrukce stromu.

Dále lze odstranit uzel, jehož jeden potomek je buňka a jehož nadrovina neobsahuje žádnou část hranice. Neboť tato nadrovina nijak nepřispívá k rozdělení prostoru, strom s kořenem v tomto uzlu může být nahrazen svým netriviálním podstromem. Tuto redukci lze provádět během konstrukce hraničních stěn. V příkladě uvedeném na obr. ... lze tedy nahradit uzel *u*uzlem *w*.



3.3. Popis vstupního souboru

Scéna je zadávána pomocí jednoduchého vstupního souboru. Soubor je rozdělen na sekce, přičemž každá sekce začíná určitým návěštím. Návěští je umístěno na samostatném řádku a uzavřeno do složených závorek. Po návěští následují řádky tvořící tělo sekce, které pokračuje až do dalšího návěští. Libovolný text na začátku řádky v těle sekce je až do znaku ´:´ považován za komentář. Tento soubor je možné vytvořit libovolným textovým exitorem, nebo lze naimplementovat interaktivní program, který bude tento soubor generovat. Následuje popis jednotlivých sekcí.

{POINTS}

∇ v této sekci se popisují jednotlivé body. Každý bod musí být na zvláštním řádku a jeho souřadnice jsou odděleny mezerami. Každému bodu je při načítání přiřazen jednoznačný index - HANDLER, který odpovídá jeho pořadí v souboru (první bod má index 0). Těchto indexů se používá při specifikování stěn. Tato sekce se může vyskytovat vícekrát a na libovolném místě souboru.

{PLANES}

∇ tato sekce obsahuje definice jednotlivých stěn těles. Každá stěna, stejně jako již zmíněné body, musí být na zvláštním řádku a jednotlivé body jsou odděleny mezerami. Na body stěny se odkazuje pomocí jejich indexů. Jediným omezením je, že tyto body již musí být definovány. Jak již bylo uvedeno, body stěny musí být uvedeny v pořadí určeném směrem proti směru hodinových ručiček. První číslo na řádku definuje barvu stěny. Tato sekce se také může vyskytovat vicekrát v souboru, avšak jejímu druhému (a dalšímu) výskytu musí předcházet sekce OP.

{OP}

∇ v této sekci se definuje množinová operace (regularizovaná), která se provést. Jedním operandem je BSP-strom z již zadaných stěn, druhým pak množina stěn definovaná v následující sekci PLANES. Možné jsou operace "UNION", "INTERSECTION" a "DIFFERENCE". Zvolená operace musí být uvedena na zvláštním řádku.

{SHADOW}

∇ tato sekce není povinná a specifikuje se v ní, který ze zadaných bodů označuje pozici světelného zdroje. Index tohoto bodu musí být opět uveden na samostatném řádku.

Soubor popisující konkrétní scénu může například vypadat následovně:

```
{POINTS}
-20 -40 -40
 40 -40 -40
 40 -40  10
-20 -40  10
.....
{PLANES}
4  3 2 1 0
.....
{POINTS}
16: -500 500 -500
.....
{OP}
DIFFERENCE
{PLANES}
2  11 10  9  8
.....
{SHADOW}
16
```

3.4. Implementace

3.4.1. BSP-strom

Struktura BSP-stromu je ve své podstatě rekursivní. Celá implementace návrhu je objektově orientovaná. Proto každému uzlu odpovídá objekt *BSPNode*.

```
union Child              ... struktura reprezentující potomky
{
  BSPNode* chnode;       ... ukazatel na podstrom
  BOOL     chleaf;       ... hodnota listové buňky (IN/OUT)
};

class BSPNode            ... uzel BSP-stromu
{
  private:
    int dim;             ... dimenze reprezentovaného prostoru
    Vector abc;          ... normála nadroviny (koeficienty
                             a,b,c rovnice roviny v obecném
                             tvaru)
    double d;            ... koeficient d v rovnici roviny
    PlaneList* planeson;     seznam stěn uzlu
    BOOL node[2];        ... typ levého a pravého potomka
                             (uzel/list)
    Child left;          ... levý potomek
    Child right;         ... pravý potomek
    HandlerPoints* points;..  ukazatel na seznam bodů
}
```

3.4.2. Vytváření BSP-stromu

Seznam metod třídy *BSPNode*, které se používají k vytváření BSP-stromu:

```
BSPNode(int dimense, HandlerPoints* Points,PlaneList* planes,
   int& Uroven,int& Deleni,int heuristika=0);
void ChoosePlane(PlaneList* planes,int heuristika);
int DividePlanes(PlaneList* planes,PlaneList* LeftList,
   PlaneList* RightList);
FLAGS Test(Plane* TestPlane);
FLAGS TestPoint(HANDLER Point);
BOOL Inter(Plane* testplane,PlaneList* RightList,
   PlaneList*LeftList);
HANDLER Cross(HANDLER lastpoint,HANDLER firstpoint);
```

Vytváření struktury BSP-stromu se provádí rekursivním voláním konstruktoru třídy *BSPNode*. Proces je ukončen, jakmile je množina stěn prázdná.

V metodě se nejprve vybere ze seznamu stěn nadrovina. Vhodná nadrovina se volí v metodě *ChoosePlane*. Která z heuristik se má použít se zadává pomocí parametru *heuristika*. Pokud hodnota tohoto parametru je mimo rozsah naimplementovaných heuristik, zvolí se vždy první stěna ze seznamu. Výsledky použití různých heuristik je uveden v příloze .... Množina kandidátů je tvořena prvními pěti stěnami ze seznamu, testovací množina je celý seznam. Váha w_{split} byla zvolena osm. Po zvolení nadroviny se příslušná stěna zařadí do seznamu *planeson* a spočítá se normála stěny (koeficienty rovnice nadroviny *abc* a *d*). Pro větší přesnost výpočtu je vektor *abc* normalizován.

Dále se seznam stěn rozdělí podle zvolené nadroviny na stěny "před" a na stěny "za" nadrovinou. Rozdělení se provádí v metodě *DividePlanes*. Pokud je nutno nějakou stěnu rozdělit, zařídí to metoda *Inter*. Test stěny a bodu vzhledem k nadrovině zajišťují metody *Test* (vrací hodnoty FRONT, BACK, INTER) a *TestPoint* (vrací hodnoty FRONT, BACK, ON). Hodnota vrácená metodou *DividePlanes* určuje počet stěn, které bylo nutno v průběhu rozdělit.

Pro takto získané seznamy se rekursivně vyvolá metoda *BSPNode*, čímž obdržíme příslušné podstromy. Referencí volané parametry *Uroven* a *Deleni* kumulují hloubku stromu a počet stěn, které bylo nutno v průběhu vyváření stromu rozdělit.

3.4.3. Klasifikace bodu

Seznam metod třídy *BSPNode*, které provádějí klasifikaci bodu:

```
BOOL ClassifyPoint(Vector& cpoint);
FLAGS TestPoint(const Vector& Point);
```

Klasifikace bodu vzhledem k množině reprezentované BSP-stromem se provádí rekursivním voláním metody *ClassifyPoint* tohoto stromu. V každém kroku se rozhoduje podle výsledku metody *TestPoint*, která určí, zda se daný bod nalézá před (FRONT), za (BACK), nebo na (ON) příslušné nadrovině. Pokud je bod klasifikován jako FRONT, proces pokračuje na pravém podstromu, při klasifikaci BACK na podstromu levém. Jakmile dospějeme do listu stromu, výsledná klasifikace bodu je rovna hodnotě této listové buňky (IN/OUT). Pokud se bod nalézá v příslušné nadrovině, je nutno pokračovat na obou podstromech a porovnat získané výsledky. Bod je výsledně klasifikován jako OUT, pouze pokud výsledky klasifikace na obou podstromech jsou rovněž OUT. To znamená, že body ležící na hranici množiny budou klasifikovány jako IN.

3.4.4. Množinové operace

Seznam metod třídy *BSPNode*, které slouží k provádění množinových operací:

```
BOOL Operation(PlaneList* planes,enum operation operace,
   int& Uroven,int& Deleni,int heuristika=0);
BOOL DivideOri(PlaneList* planes,PlaneList* Same,
   PlaneList* Oposit);
BOOL INPlanes(PlaneList* planes);
```

```
BOOL NahradNode(BOOL node);
int DividePlanes(PlaneList* planes,PlaneList* LeftList,
   PlaneList* RightList);
BSPNode(int dimense, HandlerPoints* Points,PlaneList* planes,
   int& Uroven,int& Deleni,int heuristika=0);
int PlaneList::DeleteSame();
```

V tomto návrhu je naimplementován inkrementální algoritmus pro výpočet množinových operací. To znamená, že vždy uvažujeme pouze jednu operaci. Jedná se o vyhodnocování výrazu "BSP-strom op^* hraniční reprezentace", kde hraniční reprezentaci představuje množina stěn.

Provádění množinových operací je opět rekurzivní proces, ve kterém se opakovaným voláním metody *Operation* prochází již vytvořený BSP-strom. V každém kroku se seznam stěn rozdělí na dva podseznamy, podle vztahu jednotlivých stěn k příslušné nadrovině. Po provedení zjednodušení uvedeného v tabulce 1, rekurze pokračuje na příslušných dvou podstromech. Pokud při procházení BSP-stromu dospějeme až do listové buňky a seznam stěn je i po zjednodušení neprázdný, nahradíme tento list podstromem vzniklým ze stěn tohoto seznamu (vytvoření nového podstromu pomocí konstruktoru *BSPNode*).

Po zpracování obou podstromů uzlu, by se měla upravit část hranice, kterou stěny v uzlu reprezentují. Čas potřebný pro vykreslení extra polygonů je však menší, než čas, za který se provede aktualizace hranice. Proto jsou hraniční polygony upravovány pouze pokud se jedná o operaci průniku. Viditelná plocha objektů, které se protínají, je stejná, jako viditelná plocha jejich sjednocení. Části stěn tvořící hranici se počítají v metodě *INPlanes*. Pokud zjistíme, že příslušné stěny neobsahují žádnou část výsledné hranice, lze tento uzel nahradit jeho netriviálním podstromem pomocí metody *NahradNode*. Po upravení seznamu stěn uzlu se může stát, že seznam stěn tvořících výslednou hranici bude obsahovat duplicitní stěny. Odstranění duplicitních stěn provádí metoda seznamu stěn *DeleteSame*. V implementaci se využívá analýzy pro inkrementální algoritmus uvedené v **Chyba! Záložka není definována.** a každá stěna je tedy klasifikována pouze vzhledem k jednomu podstromu. Rozdělení stěn na stěny se stejnou orientací jako je dělící nadrovina a na stěny s opačnou orientací se provádí v metodě *DivideOri*.

Referencí volané parametry *Uroven* a *Deleni* opět kumulují hloubku stromu a počet stěn, které bylo nutno v průběhu vyvářeni stromu rozdělit. V tomto případě ale hloubka stromu představuje nejvyšší hloubku, dosaženou při rozšiřování původního BSP-stromu.

4. Zobrazení

Ideální způsob vytvoření obrazu trojrozměrného objektu na dvojrozměrné ploše obrazovky, či výkresu není jednoznačně stanoven. Nejprve se zaměříme na promítací metody. V technických odvětvích dávají konstruktéři přednost vybraným pohledům na objekty (půdorys, nárys, bokorys), pro obecný pohled volí různé typy rovnoběžného promítání, které sice neodpovídá skutečnosti, ale je vhodné pro případ odměřování délek či úhlů. Architekti naopak používají téměř výlučně středové promítání, ve kterém se vzdálenější objekty jeví menšími tak, jak tomu je ve skutečnosti. Aby nedocházelo k nežádoucím efektům, je v některých případech nezbytné použít ořezávání vzhledem k zornému objemu.

Zobrazení objektů by mělo vytvářet co nejvěrnější dojem pohledu do prostoru, zprostředkovat informace o hloubce objektů a jejich rozmístění ve scéně. Dalším krokem při zobrazování prostorových objektů je tedy řešení viditelnosti - určení, které části objektů jsou viditelné a které zakryté. Výsledný dojem se dá ještě zlepšit vystínováním jednotlivých objektů ve scéně a vykreslením vržených stínů.

4.1. Promítací metody

Pojmem promítání, někdy též projekce, v počítačové grafice označujeme transformaci, která převede body z trojrozměrného prostoru do prostoru dvourozměrného [27]. Prakticky to znamená, že jedna ze souřadnic (nejčastěji z) nabyde nulové hodnoty, takže ji lze vynechat. Dále se omezíme pouze na planární projekce, tedy na promítání na rovinnou plochu.

Transformace souřadnic se výhodně provádějí pomocí transformačních matic. Ve trojrozměrném prostoru jsou to čtvercové matice řádu 4. Vyjádříme-li bod o souřadnicích (x,y,z) pomocí homogenních souřadnic jako $P=[x_h,y_h,z_h,1]$, můžeme jeho transformaci maticově vyjádřit jako:

$$P' = P \begin{vmatrix} A_{xx} & A_{yx} & A_{zx} & P_x \\ A_{xy} & A_{yy} & A_{zy} & P_y \\ A_{xz} & A_{yz} & A_{zz} & P_z \\ T_x & T_y & T_z & 1 \end{vmatrix},$$

kde koeficienty A_{jj} jsou používány pro vyjádření otáčení, změnu měřítka a zkosení, koeficienty T_j vyjadřují posunutí. Při promítání se používají prvky P_j.

V následujícím textu budeme označovat jako promítací paprsek přímku vedenou promítaným bodem, jejíž směr závisí na zvolené promítací metodě. Průmětna je rovina v prostoru, na kterou promítací paprsky dopadají. Průsečíky promítacích paprsků s průmětnou vytvářejí průmět - obraz v rovině. Dále budeme většinou předpokládat, že průmětna je shodná s rovinou xy.

4.1.1. Rovnoběžné promítání

Při rovnoběžném promítaní jsou všechny promítací paprsky rovnoběžné. Výsledkem je obraz, který zachovává relativní rozměry prostorových objektů, ale nemá přirozený vzhled. Podle úhlu, který svírají promítací paprsky s průmětnou, lze rovnoběžné promítání rozdělit na pravoúhlé (pro úhel 90°) a kosoúhlé (pro ostatní úhly). Nejjednodušší pravoúhlá promítání jsou průměty do průměten kolmých na jednu ze souřadnicových os. Matice .... je maticí pravoúhlého promítání, při kterém zanedbáváme souřadnice *z*.

//////matice str.281

Pokud je při pravoúhlém promítání umístěna průmětna obecně, zobrazí se na ni více než jedna stěna krychle, jejíž stěny jsou rovnoběžné se souřadnicovými osami. Takový pohled se nazývá axonometrický pravoúhlý průmět. Nejčastěji používané axonometrické promítání je promítání izometrické. Při něm je průmětna umístěna tak, že protíná všechny tři osy souřadnic ve stejné vzdálenosti od počátku. Všechny osy jsou v izometrickém promítání stejně kráceny, takže relativní velikosti objektů jsou zachovány. V technické praxi se používají další typy axonometrie jako je dimetrie nebo trimetrie.

Kosoúhlý průmět získáme promítnutím bodů do průmětny ve směru, který není kolmý k průmětně. Toto promítnutí lze provést jako zkosení ve směru roviny *xy* a kolmé promítnutí do této roviny. Z kosoúhlých promítání se nejčastěji používá kavalérní a kabinetní promítání.

4.1.2. Středové promítání

Při středovém promítaní, často nazývaném perspektivní, všechny paprsky procházejí jedním bodem - středem promítání. Ve výsledném obrazu se úsečky vzdálené více od středu promítání zobrazí jako kratší než stejně dlouhé úsečky bližší.

Střed promítání (x_C, y_C, z_C) bývá často umístěn na ose *z* v konečné vzdálenosti od průmětny *xy*. V takovém případě má souřadnice *(0,0,d)*, kde číslo *d* označuje vzdálenost od průmětny. Použijeme-li trojrozměrné homogenní souřadnice, můžeme zapsat středovou transformaci v maticovém tvaru ....

////// matice str. 658

Homogenizační faktor výsledného vektoru není tentokrát roven jedné, ale po vynáso.....

Je-li trojrozměrný objekt středově promítnut do roviny, pak jakákoliv množina rovnoběžných hran objektu je promítnuta jako množina sbíhajících se úseček. Jako rovnoběžné se promítají pouze rovnoběžné úsečky, které jsou současně rovnoběžné i s průmětnou. Bod, ve kterém se sbíhají průměty ostatních rovnoběžných úseček, se nazývá úběžník. Pokud patří úběžník množině úseček rovnoběžných s některou souřadnicovou osou ve světové soustavě souřadnic, pak se jedná o hlavní úběžník. Podle počtu těchto hlavních úběžníků lze středové promítání rozdělit na jednobodové (jednoúběžníkové), dvoubodové a trojbodové.

Rovnoběžná projekce je vlastně speciálním případem středového promítání, při kterém se pozorovatel nalézá v nekonečnu.

4.1.3. Pohledové transformace

Transformace, které provádějí převod z obecné polohy do standardní promítací polohy, se nazývají pohledové transformace. Jak již bylo uvedeno, je výhodné, pokud je průmětnou rovina *xy* a paprsky jsou vedeny buď rovnoběžně s osou *z* (rovnoběžné promítání), nebo procházejí středem promítání umístěným na záporné (kladné) poloose *z*.

Nejprve se musíme zmínit o zadávání parametrů promítání. Důležitým údajem je stanoviště pozorovatele, které odpovídá středu promítání. Dále je třeba určit směr promítání, který se někdy také nazývá směr pohledu. Zadává se buď trojicí souřadnic, nebo pomocí dvojice úhlů - azimutu a zenitu. Azimut určuje odchylku od osy *x* měřenou v rovině rovnoběžné s rovinou *xy*, zenit určuje úhel od roviny *xy*. Zbývá ještě zvolit orientaci průmětny pomocí dalšího vektoru. Ten se nazývá vertikální vektor a je kolmý na směr pohledu. Zadáme ho pomocí úhlu měřeného od roviny *xy*. Tím máme zadánu uživatelskou pohledovou soustavu souřadnic. Hledáme tedy transformaci, která převede tuto soustavu souřadnic do světové soustavy souřadnic.

Pohledovou transformaci provedeme jako posloupnost posunutí a otočení. Po jejím provedení zbývá všechny transformované prostorové objekty promítnout do roviny *xy* požadovaným typem promítání.

4.2. Prostorové ořezávání

V grafických systémech je žádoucí vybrat úsek trojrozměrného prostoru, ve kterém leží promítané objekty. Při rovnoběžném promítání vybereme objekty nejsnáze tak, že definujeme zorný hranol (pohledový hranol), který obklopí potřebnou prostorovou oblast. Zorný hranol má stěny rovnoběžné se souřadnicovými osami *x* a *y*. Hranol je kolmý na průmětnu a jeho přední a zadní stěna jsou určeny dvěma rovinami rovnoběžnými s průmětnou.

Výběr prostorové oblasti určené k zobrazení se u středového promítání provádí pomocí zorného jehlanu. Jeho vrchol je v místě stanoviště pozorovatele a stěny vytínají rovnoběžník odpovídající okénku v průmětně. Po zadání přední a zadní stěny se zorný jehlan mění na komolý. Nutnost použití zorného jehlanu vyplývá i z charakteru středového promítání, při němž dochází k deformaci tvaru průmětů u objektů, které jsou příliš vzdáleny od osy *z*. Volba vrcholového úhlu zorného jehlanu by měla odpovídat zornému úhlu lidského oka. Ten však není jednoznačně stanoven. Ve skutečnosti je lidský zrak schopen periferním viděním pojmout obraz ve vodorovném rozsahu 150° a svislém 120°. Pro středové promítání se tedy volí úhel zorného jehlanu většinou v rozmezí 40°-60°.

Pro odstranění částí scény mimo zorný objem se používá metod ořezávání. Jednou možnou metodou je využití BSP-stromů. Pokud budeme zorný objem (hranol, jehlan, ...) reprezentovat pomocí BSP-stromu, pak stačí vždy před zobrazením stěny ji klasifikovat vzhledem k tomuto stromu a kreslit pouze ty části, které byly klasifikovány jako IN.

4.3. Určování viditelnosti

Problém řešení viditelnosti je jedním z nejobtížnějších problémů v počítačové grafice. V zásadě jde o určení bodů, částí hran a povrchů, které jsou neviditelné (zakryté jinými

plochami) z dané pozice pozorovatele. Složitost problému eliminace neviditelných hran a povrchů vyústila v nepřeberné množství algoritmů, které používají různé principy a jejich kombinace. U algoritmů používaných v aplikacích reálného času je požadavkem poskytnou řešení vždy v rámci 1/50 nebo 1/30 sekundy. Budeme-li však požadovat realističtější výstup s více podrobnostmi včetně stínování, průhlednosti a průsvitnosti objektů nebo s odrazy, pak jsou tyto algoritmy pomalejší a často vyžadují mnoho minut až hodin k řešení daného problému.

4.3.1. Dělení algoritmů podle časové složitosti

Z pohledu časové složitosti se algoritmy dají rozdělit na objektově orientované a obrazově orientované. Složitost objektově orientovaných algoritmů je úměrná n^2, kde n je počet objektů ve scéně. Základem těchto algoritmů je porovnávání vzájemných poloh těles, resp. jejich částí, jako jsou plochy a hrany.

U obrazově orientovaných algoritmů je složitost úměrná součinu $n.p$, kde n je počet objektů a p je počet pixelů rastru. Tyto algoritmy určují pro každý pixel obrazovky, která část promítnutého objektu je v tomto pixelu nejblíže pozorovateli, a tedy viditelná. Poznamenejme ale, že toto dělení není jednoznačné.

4.3.2. Dělení algoritmů podle reprezentace těles

Způsob zápisu tvaru tělesa má rozhodující vliv na metodu odstraňování zakrytých částí, a tedy na kvalitu zobrazení tělesa. Nejprve byly vyvíjeny algoritmy, které zpracovávaly údaje o tělesech popsaných hraniční reprezentací. Tyto algoritmy se dají rozdělit na čárové[14] a plošné[15]. S nástupem barevných obrazovek došlo k rychlému rozvoji rastrových algoritmů, které řeší viditelnost těles popsaných rozličnými způsoby.

Výstupem čárových algoritmů je obvykle seznam viditelných částí čar (hran), které lze libovolně zvětšovat. Tyto algoritmy jsou vhodné pro zobrazování na vektorových výstupních zařízeních ve vysoké přesnosti. Často jsou vykreslovány i zakryté části, například jinou barvou nebo jiným typem čáry. Reprezentanty této skupiny jsou Robertsův algoritmus [31] a Appelův [1].

Oproti tomu výstupem plošných algoritmů je většinou pole pixelů. Jedním takovým algoritmem je například *z-buffer* [4] (v překladu se většinou používá termín paměť hloubky), který lze implementovat jak softwarem, tak pomocí hardwaru. Paměť hloubky vlastně reprezentuje dvourozměrné pole, jehož rozměry jsou totožné s rozměry obrazovky. Každá položka obsahuje souřadnici z nejbližšího bodu vzhledem k pozorovateli, jehož průmět leží v odpovídajícím pixelu obrazové paměti.

Samostatné podmnožiny tvoří algoritmy tvořící řádkový rozklad[16] (například Watkinsův alg. [41]), algoritmy rekurzivně dělící prostor (například Warnockův alg. [40]) a v neposlední

[14]Hidden line elimination

[15]Hidden surface elimination

[16]Scan-line algorithms

řadě algoritmy, jejichž výstupem je prioritní seznam stěn[17]. Poslední uvedenou skupinou se budeme zabývat podrobněji.

Mnohem složitější a i časově náročnější jsou algoritmy pro určování viditelnosti zakřivených povrchů. Aby bylo možné použít výše uvedené algoritmy, musí být nejprve tyto povrchy aproximovány mnoha malými stěnami. Významným reprezentantem této skupiny je metoda sledování paprsku [13].

4.3.3. Prioritní seznam stěn

Princip této poměrně rozšířené skupiny metod spočívá v přímém vykreslování ploch (stěn) na obrazovku, a to v pořadí od nejvzdálenějších k nejbližším vzhledem k pozorovateli (kritériem je porovnání *z*-ových souřadnic). Bližší plochy překryjí zadní, čímž je viditelnost vyřešena přirozeným způsobem. V případě, že se plochy protínají nebo se zakrývají navzájem, je nutné přistoupit k jejich dělení, což vede k jisté komplikaci jinak jednoduchého algoritmu.

Nejznámějším představitelem této skupiny je tzv. malířův algoritmus popsaný v [26]. My se však zaměříme na algoritmus využívající reprezentace objektů pomocí BSP-stromů [9], [10]. Tento algoritmus je vhodný zejména pokud počítáme viditelnost statické scény tak, jak je viděna z libovolného bodu pohledu. Výhodou je lineární zobrazování pokaždé, když je změněna poloha pozorovatele. Tomu ale předchází relativně velký inicializační čas a prostorově náročný předzpracovávací krok.

Základní myšlenka algoritmu je následující. Pokud se nám podaří nalézt rovina, která odděluje jednu skupinu stěn od druhé, pak skupina, která je ve stejném poloprostoru této roviny jako pozorovatel, může zastínit, ale nemůže být zakryta skupinou stěn v opačném poloprostoru. Pokud si uvědomíme, že na tomto principu je založena konstrukce BSP-stromu, jak byla uvedena v 3.1.4., předzpracovávací krok je totožný s vytvořením BSP-stromu reprezentujícího danou scénu a požadovaný prioritní seznam stěn získáme pouhým průchodem tohoto stromu. Zda při průchodu stromu budeme pokračovat nejprve do levého nebo pravého podstromu určíme podle vztahu (úhlu, který svírají) mezi normálou nadroviny příslušející k danému uzlu a vektoru, který určuje směr pohledu.

Výhodou tohoto, oproti malířovu algoritmu je, že se nemusí znovu dělit všechny stěny, pokud se změní pohled na scénu. Počet těchto dělení závisí na volbě vhodné heuristiky při vytváření BSP-stromu a obecně jich může být více, než při použití malířova algoritmu. Dalším zjednodušením je, že se vyhneme problémům s možným cyklickým překrytím polygonů.

4.4. Stínování

Pod pojmem stínování rozumíme vykreslování barevných objektů různými odstíny barev. Pomocí stínování lze odlišit případné křivosti a zaoblení ploch, a docílit tak přirozeného vzhledu prostorových objektů. Samostatnou kapitolou je kresba vržených stínů.

[17] List-priority algorithms

Většina následujících stínovacích metod je určena pro rastrové displeje. Neřeší viditelnost ploch, ale pouze barevné odstíny bodů, vypočítaných při rozkladu plochy na pixely. Z toho vyplývají také algoritmy pro určování viditelnosti, do kterých bývají zařazeny.

4.4.1. Konstantní stínování

Tato technika vychází ze zjednodušených předpokladů, kdy zdroje světla i pozorovatel jsou v nekonečnu. Konstantní stínování je postačující k zobrazení těles ohraničených lineárními ploškami, přičemž prakticky nepřináší zvýšení výpočetní složitosti, neboť jas, resp. barva je konstantní v rámci daného n-úhelníka. Barevný odstín, který je přiřazen všem pixelům plochy je vypočítán podle normály plochy.

U obecnějších těles je konstantní odstín plošek negativním jevem, protože místo zkvalitnění obrázku zdůrazňuje, že oblý povrch je ve skutečnosti jen aproximován skupinou plošek.

4.4.2. Stínování obecných ploch

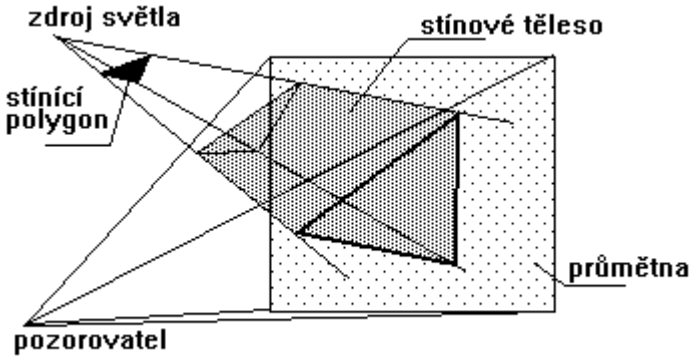
Pro stínování obecných ploch lze použít Gouraudovo stínování [15], které je založeno na interpolaci jasu vycházející ze znalosti normály plošech ve vrcholech n-úhelníka sítě, která zakřivenou plochu aproximuje. Ještě lepší výsledný vjem poskytuje stínování Phongovo [28], které interpoluje normálový vektor podél zobrazovaného řádku, čímž je zajištěna lepší aproximace křivosti povrchu podél řádky.

4.5. Vržené stíny

Nalezení vrženého stínu, tedy takové části plochy, která je před dopadem světla zastíněna jiným tělesem, je poměrně náročná úloha. Při výpočtech musíme brát v úvahu nejen polohu zobrazované plochy a zdrojů světla, nýbrž i vzájemné polohy všech těles a ploch ve scéně.

4.5.1. Stínová tělesa[18]

První metody pro určování vržených stínů se snažily využít již existujících algoritmů pro



[18] Shadow volume

určování viditelnosti ploch. Důležitým rozšířením je vytvoření stínových těles [6]. Pro každou plošku je vytvořeno umělé (stínové) těleso určené hranicí plošky, paprsky z určitého zdroje světla a hranicemi zobrazovacího prostoru. Plochy tohoto tělesa jsou přidány ke vstupním datům popisujícím scénu. Stínové těleso je ohraničeno na jedné straně původní ploškou a na opačné straně zvětšenou kopií tohoto polygonu. Kopie je umístěna v takové vzdálenosti od zdroje, kde energie vyzařovaná zdrojem je zanedbatelná - zóna působnosti zdroje. Při zobrazování scény jsou pak stínová tělesa uvažována jako zcela průhledná. Pokud při určování barvy skutečného tělesa zjistíme, že testovaný bod na tělese leží uvnitř stínového tělesa, pak je tento bod ve stínu. Počet stínových těles, ve kterých bod leží, určuje intenzitu (tmavost) stínu.

V reálném světě jsou ale používána světla se směrovým charakterem. Při určování osvětlení povrchu je v takovém případě nutno uvažovat, zda daný bod povrchu leží uvnitř světelného kuželu směrového zdroje světla. Ještě složitější situace vznikne při použití plošného zdroje světla. Takový zdroj světla vytváří polostíny, jejichž odstín je proměnlivý od úplného stínu až po stín žádný. Tato světelná situace může být řešena nahrazením plošného zdroje několika zdroji bodovými.

Moderní algoritmy, jako je například metoda vrhání paprsku nebo radiační metody, řeší uvedené problémy odlišně. Za tyto výsledky však platíme výrazně delším časem zpracování scény.

4.5.2. SVBSP algoritmus

Jedná se o algoritmus generování vržených stínů pro polygonální scénu, která je osvětlená jedním nebo více bodovými zdroji světla. Algoritmus lze snadno implementovat na jakémkoliv grafickém systému, který umožňuje rychlý řádkový rozklad polygonu a dosahuje zobrazování nenáročného prostředí v téměř reálném čase.

Algoritmus, který zde bude popsán kombinuje části dvou druhů současných algoritmů generování stínů: dvouprůchodového tělesově-prostorového přiblížení a přiblížení pomocí stínových těles. V dvouprůchodovém tělesově-prostorovém přiblížení scéna musí být celá v pohledovém objemu světelného zdroje světla a musí být transformována perspektivní transformací vzhledem k tomuto zdroji. Zde popisovaná technika rozděluje polygony scény na osvětlené části a na části, které jsou ve stínu a přitom nevyžaduje žádné transformace polygonů během generování stínů. Alternativa tohoto algoritmu může být použita i k určování plošné viditelnosti (světelný zdroj umístíme do bodu pozorovatele a získáme prioritní seznam nepřekrývajících se osvětlených - viditelných - částí polygonů scény).

Stínový objem generovaný tímto algoritmem nemusí být uzavřený (lze ho oříznout pohledovým objemem) a neobsahuje aktuální polygony scény. Výhody tohoto algoritmu vycházejí ze síly BSP-stromů a jejich zobecnění pro modelování mnohostěnu.

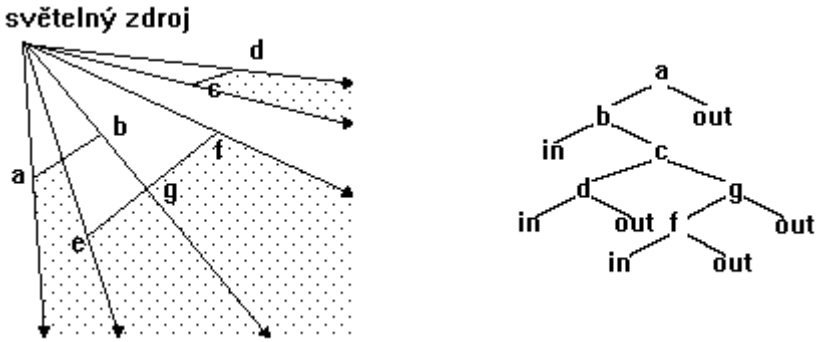
BSP-strom stínového objemu (SVBSP) je modifikací klasického BSP-stromu. Každý vnitřní uzel SVBSP-stromu je spojen se "stínovou plochou", která je definovaná bodem zdroje světla a hranou polygonu přivráceného ke zdroji světla (pokud je použit vektorový zdroj světla, pak je stínová rovina definovaná směrovým vektorem zdroje světla a hranou tohoto polygonu). Tímto získáme "stínový polygon", který ohraničuje stínový objem. Směr normály stínové plochy se použije k určení poloprostoru, ve kterém je objekt umístěn. V listech pak jsou buňky IN a OUT, které určují, kdy je nebo není oblast vnitřkem stínového objemu.

Základní kroky, které se musí provést pro každý polygon přivrácený ke světelnému zdroji:

- Určení stínů - polygon je filtrován SVBSP-stromem, abychom určili, které části jsou zastíněné a které osvětlené.
- Zvětšení SVBSP-stromu - vytvoří se stínový objem pro každou osvětlenou část polygonu a přidá se do SVBSP-stromu.

Jako nejpřímější metoda, jak určit, zda je polygon zastiňován, se nabízí porovnání zkoumaného polygonu se stínovými objemi všech ostatních polygonů. Polygony, které jsou dále od zdroje světla než právě testovaný polygon na něj nemohou vrhat stín. Z toho vyplývá, že pokud budeme polygony zpracovávat v pořadí odpředu dozadu, vzhledem ke světelnému zdroji, pak každý polygon bude porovnáván pouze se stínovými objemy polygonů, které už byly zpracovány, tj. těmi, které jsou ke zdroji blíže, než testovaný polygon. Toto pořadí může být získáno ze základního BSP-stromu, který je vytvořen z polygonů scény, tak, že ho budeme procházet z pohledu světelného zdroje. Tento základní BSP-strom se musí přepočítat pouze v případě, že se polygony scény změní, nikoliv při pohybu světelného zdroje.

Efektivnější, než porovnávat polygon s každým jednotlivým stínovým objemem všech polygonů ležících před ním, je uchovávat jeden spojený stínový objem. Ten je postupně rozšiřován sjednocováním se stínovým objemem každého nového polygonu v pořadí jejich zpracovávání. Výsledný spojený stínový objem je množina nekonečných pyramid vycházejících z bodového zdroje světla.



Určování, které části polygonu jsou ve stínu, se provádí filtrováním polygonu SVBSP-stromem. Pokud polygon leží v obou poloprostorech roviny příslušející uzlu, rozdělí se. Části, které leží v listových buňkách, označených jako IN, jsou ve stínu, naproti tomu části ležící v OUT buňkách jsou osvětlené. Neboť SVBSP-strom je vytvářen inkrementálně, každý polygon je porovnáván pouze s tou částí stromu, která byla již vytvořena. Poznamenejme, že polygony, které jsou odvrácené od zdroje světla leží celé ve stínu a tudíž není nutné je vkládat do stromu.

SVBSP-strom musí být dále zvětšen, aby obsahoval každou osvětlenou část stínového objemu. Proto vytvoříme množinu stínových stěn pro hrany osvětlených částí (pouze ty mohou vrhat stíny) a přidáme je do již existujícího stromu. Na obr. ... je uveden dvojdimenzionální příklad stínového objemu a jemu příslušející SVBSP-strom.

Uvedený algorimus lze snadno modifikovat pro více světelných zdrojů tak, že se pro každý zdroj vytvoří samostaný SVBSP-strom. Celá scéna je nejprve zpracována vzhledem k jenomu světelnému zdroji, pak k dalšímu. V paměti tedy musí být vytvořen vždy pouze jeden SVBSP-strom.

Je zřejmé, jak důležité je udržovat SVBSP-strom dobře vyvážený. To nám velmi usnadní průchod stromem, neboť každý polygon musí být filtrován až do listů. Samozřejmě musíme sledovat i velikost stromu. Jednou možností je, uvažovat pouze obrysové hrany tělesa. Zmenšit velikost stromu lze i tak, že vytváříme stínové stěny pouze pro polygony, které uživatel označil, že můžou vrhat stíny. V případě jednoho zdroje světla může být někdy výhodnější uchovávat části polygonů, které jsou ve stínu, nežli části osvětlené. Polygon je pak zobrazován tak, že části ve stínu se kreslí přes původní nerozdělený polygon.

V případě více zdrojů světla je vhodné stromy zpracovávat v pořadí, které bude mít za následek nejmenší fragmentaci polygonů. Jednou možnou heuristikou je zpracovávat stromy v pořadí od stromu odpovídajícího zdroji, ke kterému je přivráceno nejméně stěn až ke stromu, kterému odpovídá nejvíce takovýchto stěn. Jinou heuristiku je nutné použít, pokud předpokládáme, že umístění některých zdrojů světla se může měnit.

BSP-stromy nepředstavují tedy sjednocenou kostru pouze pro počítání viditelnosti, klasifikaci bodů a množinové operace na mnohostěnu, ale také pro interaktivní generování stínů.

4.6. Implementace

Není dokončeno!

5. Vytváření středních snímků

Pro část animace, kdy se vytvářejí střední smínky, je velmi výhodné použít počítačové podpory. S tímto ale vyvstávají i nové problémy. Pokud interpolaci provádí člověk, snadno pochopí smysl a vlastnosti sekvence (jedná se o padající míček nebo o kutálející se?). Počítačově založený systém ale obvykle obdrží pouze počáteční a koncovou pozici objektů.

Návrh a implementace vhodné interpolační metody je vlastně podstatou této práce. Omezíme se na interpolaci pozice a orientace pevného trojrozměrného tělesa. Nejjednodušší metodou je interpolace lineární, nebo pomocí spline funkcí. Tyto interpolace lze vhodně použít k interpolaci pozice. Interpolace orientace je poměrně náročnější problém a bude jí věnována většina této kapitoly.

5.1. Interpolace pozice

Lineární interpolace[19] pro dané hodnoty v_s a v_e, které označují určitý atribut (pozici, barvu, velikost) v počátečním a koncovém snímku, určuje hodnoty v_t v mezi-snímcích podle vzorce .....

$$v_t = (1 - t)v_s + tv_e, \quad t \in \langle 0,1 \rangle$$

Lineární interpolace je v určitých případech vhodná, ale má určitá omezení. Pokud například použijeme lineární interpolaci k určování pozic míče hozeného do vzduchu, výsledná dráha bude poněkud nerealistická. Přestože lineární interpolace vytváří spojitý pohyb, negeneruje spojité derivace. Proto se mohou při jejím použití při interpolaci pozice objevit prudké změny v rychlosti.

Tyto nedostatky lineární interpolace se řeší použitím spline funkcí. Spline funkce lze použít k plynulé interpolaci libovolného parametru jako funkce času. Abychom získali plynulé počáteční a koncové změny a relativně konstantní poměry změn mezi, můžeme například použít funkci *f(t)* uvedenou na obr. .... Hodnotu pak lze interpolovat podle vzorce ... Neboť derivace pro *t=0* a *t=1* je rovna nule, počátek a konec změn atributu *v* jsou plynulé. Tyto křivky tedy umožňují, aby se individuální body (nebo objekty) pohybovaly spojitě v prostoru a čase.

$$v_t = (1 - f(t))v_s + f(t)v_e, \quad t \in \langle 0,1 \rangle$$

V animaci můžeme specifikovat cestu spline funkce, interpolující pohyb, pro každý bod v jednotlivých snímcích. Tyto funkce ale poskytují nejhladší pohyb, pokud mají málo řídících bodů. Proto je lepší specifikovat pozice jen pár bodů a pouze v několika časových okamžicích a rozšířit interpolaci na mezilehlé body a časové úseky. Pokud budeme například interpolovat obraz složený z čar, stačí interpolovat příslušné vrcholy.

[19] *lerping*- Linear intERPolation

Při interpolaci trojrozměrných pevných těles interpolujeme pozici jejich středu podle nějaké spline trajektorie. Navíc lze určit rychlost, kterou se trajektorie prochází, a to buď vyznačením stejně velkých časových úseků na interpolované cestě, nebo vyjádřením rychlosti podél této cesty jako funkce času.

5.2. Interpolace orientace

Interpolace orientace je náročnější problém. Ani vlastní specifikování orientace není jednoduché. Pokud budeme definovat orientaci pomocí tří Eulerových úhlu, tj. hodnot otočení okolo základních os, pak je velmi důležité pořadí specifikace. Uvažujme například knihu, jejíž hřbet je vlevo. Pokud tuto knihu otočíme o 90° okolo osy *x* a potom o -90° okolo osy *y*, bude hřbet knihy směrem k nám. Pokud ale provedeme tyto rotace v opačném pořadí, bude hřbet směřovat dolů. Interpolací jenotlivých Eulerových úhlů dostaneme nepřirozenou interpolaci rotací.

Množinu všech rotací lze vyjádřit pomocí algebraické struktury kvaternionů [17]. Kvaterniony lze také uvažovat jako body na čtyřrozměrné kouli. Interpolace mezi dvěma kvaternionama (mezi dvěmi orientacemi) tedy odpovídá sférické lineární interpolaci[20] mezi odpovídajícími body na kouli [32]. V příloze .... jsou uvedeny potřebné vzorce pro převod mezi jednotlivými popisy rotací.

5.2.1. Kvaterniony

Jednotkové kvaterniony lze vyjádřit ve tvaru ...., kde *a,b,c,d* jsou reálná čísla splňující ..... Kvaternion .... odpovídá rotaci o úhel $\phi=2\cos^{-1}(a)$ okolo osy *(b c d)*v třírozměrném prostoru. Kvaterniony, stejně jako rotace, tvoří nekomutativní grupu s násobením. Neboť splňují ..... , lze je považovat za body na kouli v čtyřrozměrném prostoru.

$$q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$$

$$a^2+ b^2+ c^2+d^2= 1,\ a, b, c, d\in R$$

$$i^2= j^2= k^2= -1$$

Abychom mohli definovat aritmetické operace, rozdělíme čtyři složky kvaternionů na reálnou část, skalár *s= a,*a na imaginární část, vektor *v=(a,b,c)*. Při sčítání pouze sečteme skalární a vektorové části obou kvaternionů. Velmi důležité je násobení definované vzorcem .... neboť násobením příslušných kvaternionů lze snadno provést kombinování rotací. Výsledkem těchto operací je nová orientace v "kvaternionových souřadnicích". V rozboru uvedném v [32] autoři dospěli k závěru, že kvaternionové souřadnice jsou nejvhodnější pro interpolaci orientace.

$$[s_1,v_1][s_2,v_2] = [(s_1s_2 -v_1\cdot v_2),(s_1v_2+ s_2v_1 + v_1\times v_2)]$$

Počet souřadnic potřebných pro vyjádření kvaternionu je velmi malý (například vzhledem k vyjádření pomocí rotačních matic se jedná o tři čísla místo devíti). Kompaktnost a jednoduchost kvaternionů je jejich velkou výhodou, ale uveďme také jejich tři nejzávažnější

[20] *slerp*

nedostatky. Zaprvé, každé orientaci objektu odpovídají dva kvaterniony, neboť rotace okolo osy *v* o úhel ϕ je to samé, co rotace okolo osy *-v* o úhel $-\phi$. Příslušným kvaternionům odpovídají protilehlé body na čtyřrozměrné kouli. Proto při interpolaci z jednoho kvaternionu do druhého máme dvě možné cesty. Obvykle je vybírán ten oblouk, který je kratší.

Orientace a rotace není úplně to samé. Rotace o 360° se velmi liší od rotace o 0°, ale obě tyto rotace lze reprezentovat jedním a tým samým kvaternionem. Proto při definování vícenásobné rotace je nutné specifikovat hodně mezilehlých řídících bodů.

Posledním vážným nedostatkem je, že kvaterniony poskytují isotropické metody rotace, tzn. že interpolace je nezávislá na všem kromě vztahu mezi počáteční a koncovou rotací. Toto je ideální například pro interpolaci padajících těles. Pokud interpoluleme pozici kamery, je pro nás přirozené, že daný pohled zachovává svislost. Toto omezení kvaterniony nesplňují a musí být eventuelně dále řešeno.

5.2.2. Sférická interpolace

Nejprve budeme uvažovat poze dvě zadané orientace reprezentované pomocí kvaternionů. Tyto kvaterniony zobrazíme na kouli ve čtyřrozměrném prostoru. Body na jejich spojnici pak udávají interpolované orientace do koncového bodu. Přímá spojnice však nezachovává vše z geometrie rotačního prostoru. Pokud by interpolované body byly rovnoměrně rozloženy podél této přímky, pak by výsledná animace uprostřed sekvence rotovala rychleji. Jesliže však budeme uvažovat kruhový oblouk[21] mezi těmito body, pak můžeme rotovat aniž bychom uprostřed zrychlovali.

Vzorec pro sférickou lineární interpolaci z q_1 do q_2 lze vyjádřit dvěma způsoby, kde parametr $u \in \langle 0,1 \rangle$. První .... vychází ze struktury grupy, narozdíl od druhého ...., který vychází ze 4D geometrie. První tvar se poměrně snadno analyzuje, pro aplikace je však praktičtější tvar druhý.

$$Slerp(q_1, q_2; u) = q_1 (q_1^{-1} q_2)^u$$

$$Slerp(q_1, q_2; u) = \frac{\sin(1-u)\theta}{\sin\theta} q_1 + \frac{\sin u\theta}{\sin\theta} q_2 \text{ , kde } q_1 \cdot q_2 = \cos\theta$$

V animaci však obvykle potřebujeme plynule spojit více než dvě klíčové pozice. Při vytváření výchozí posloupnosti kvaternionů, mezi nimiž budeme interpolovat, nesmíme zapomenout na fakt, že každé rotaci odpovídají dva protilehlé body. Pokud tedy převádíme orientaci, např. z maticové podoby, na kvaterniony, sestrojujeme posloupnost tak, že každý následující kvaternion zvolíme na té straně koule, která je předchozímu bodu blíže. V případě interpolace mezi více klíčovými pozicemi už nevystačíme se sférickou interpolací, neboť je zapotřebí spojitosti vyššího řádu.

5.2.3. Interpolační křivky na kouli

Nyní potřebujeme sestrojit hladké křivky na kouli. V úvahu připadají spline křivky a Bézierovy křivky. Spline křivky (interpolační křivky sestrojené z kubických polynomiálních

[21] křivka, ve které koule protíná rovinu procházející příslušnými dvěma body a počátkem

částí) minimalizují celkové "kroucení" křivky a mezi jednotlivými částmi splňují spojitost druhého řádu. Pomocí Lagrangeových násobků lze omezit výslednou křivku tak, aby ležela na kouli [5]. Vzniklý vzorec spline křivky je však velmi složitý a náročný na řešení. Další nevýhodou je, že pokud změníme jeden bod, změní se celá křivka.

Pro naši úlohu se tedy jeví jako vhodnější Bézierovy křivky [2]. Tyto křivky ale prochází pouze svým prvním a posledním definičním bodem a my potřebujeme, aby interpolační křivka procházela všemi zadanými body. Trik spočívá v tom, že budeme spojovat dohromady krátké Bézierovy křivky - segmenty. Ve výsledku nám postačí spojitost prvního řádu.

Kromě toho, že křivka prochází koncovými body, je rovněž tečná ke svým koncovým úsekům. Jednotlivé segmenty sloučíme přes spoje a srovnáme jejich délky. Výsledné napojení je hladké. Pokud zadané orientace v klíčových snímcích umístíme do těchto spojů, pak každá krátká křivka nás spojitě posune do dalšího klíčového snímku. Přestože dva segmenty, které hraniční se spojem, na sebe musí hladce navazovat, jeden z nich lze zvolit. Tyto volby určují osy a rychlosti rotace, se kterými procházíme klíčovými body. Výběr může provádět ručně animátor, je však vhodnější jej automatizovat.

K určení jednotlivých segmentů jsou zapotřebí čtyři body - kvaterniony. Koncové dva vždy odpovídají kvaternionům klíčových snímků. Zbylé dva body je nutno definovat tak, aby byla dodržena výše uvedená pravidla. Uvažujme posloupnost klíčových kvaternionů q_{n-1}, q_n, q_{n+1} vyjádřených jako čtyřrozměrné vektory. Řídící bod segmentu a_n, který následuje za q_n, lze například definovat pomocí vzorce .... Odpovídající bod segmentu b_n před q_n pak může být vyjádřen jako ... Touto volbou je zajištěno hladké napojení ve spoji, nehledě na to, jak je zvoleno a_n.

Nyní máme definovanou každou krátkou křivku pomocí čtyř kvaternionů, q_n, a_n, b_{n+1}, q_{n+1}. Nechť parametr u nabývá hodnot od *0* do *1*tak, jak křivka vychází z bodu q_n a pokračuje do q_{n+1}. Bod křivky odpovídající parametru u lze určit podle schématu ..., kde $(p:q)_u$ označuje sférickou lineární interpolaci ... Slerp(p,q:u). Pro větší názornost je uveden obr. .....

$q_n = p_0^{(0)}$

$(p_0^{(0)}:p_1^{(0)})_u = p_0^{(1)}$

$a_n = p_1^{(0)}$ $(p_0^{(1)}:p_1^{(1)})_u = p_0^{(2)}$

$(p_1^{(0)}:p_2^{(0)})_u = p_1^{(1)}$ $(p_0^{(2)}:p_1^{(2)})_u = p_0^{(3)} = q_{n+u}$

$b_{n+1} = p_2^{(0)}$ $(p_1^{(1)}:p_2^{(1)})_u = p_1^{(2)}$

$(p_2^{(0)}:p_3^{(0)})_u = p_2^{(1)}$

$q_{n+1} = p_3^{(0)}$

Z uvedeného vyplývá, že tečny koncových úseků segmentů jsou určeny hodnotami a_n a b_{n+1}. Stejně jako u původních Bézierových křivek, je velikost této tečny trojnásobkem délky příslušného segmentu. To znamená, že rotujeme třikrát rychleji, než sférická interpolace podél oblouku. Tuto rychlost lze upravit tak, že koncové úseky zkrátíme na jednu třetinu původní velikosti. Výsledek bude takový, že a_n bude blíže ke q_n a b_{n+1} ke q_{n+1}.

6. Literatura

[1] Appel, A., "The Notion of Quantitative Invisibilitz and the Machine Rendering of Solids", Proceedings of the ACM National Conference, Thompson Books, Washington, DC, 1967

[2] Bézier, P.E., Numerical Control - Mathematics and Applications, John Wiley and Sons, London, 1972

[3] Borning, A., "Thinglab - A Constraint-Oriented Siulation Laboratory", Technical Report SS1-79-3, Xerox Palo Alto Research Center, Palo Alto, CA, July 1979

[4] Catmull, E., A Subdivision Algorithm for Computer Display of Curved Surfaces, Ph. D. Thesis, Report UTEC-CSc-74-133, Computer Science Defartment, University of Utah, Salt Lake City, UT, December 1974

[5] Courant, R., and D. Hilbert, Methods of Mathematical Physics, Volume I, Interscience Publishers, Inc., New York, 1953

[6] Crow, F.C., "Shadow Algorithms for Computer Graphics", SIGGRAPH 77

[7] Feiner, S., S., D. Salesin, and T. Banchoff, "DIAL: A Diagrammatic Animation Language", CG & A, 2(7), September 1982

[8] Foley, James D., Andries van Dam, Steven K. Fernier, and John F. Hughes, Computer Graphics, Principles and Practice, Addison-Wesley, 1990

[9] Fuchs, H., Z.M. Kedem, and B.F. Naylor, "On Visible Surface Generation by A Priori Tree Structures", SIGGRAPH 80

[10] Fuchs, H., G.D. Abram, and E.D. Grant, "Near Real-Time Shaded Display of Rigid Objects", SIGGRAPH 83

[11] Ginsberg, C. M., and D. Maxwell, "Graphical Marionette", in Proceedings of the SIGGRAPH/SIGART Interdisciplinary Workshop on Motion: Representation and Perception, Toronto, April 4-6, 1983

[12] Girard, M., and A. A. Maciejewski, "Computational Modeling for the Computer Animation of Legged Figures", SIGGRAPH 85

[13] Glassner, A.S., An Introduction to Ray Tracing, Acadmic Press, London, 1989

[14] Goldberg, A., and Kay, A., SMALLTALK-72 Instruction Manual, Learning Research Group, Xerox Palo Alto Research Center, Palo Alto, CA, March 1976

[15] Gouraud, H., Computer Display of Curved Surfaces, Ph. D. Thesis, Univ. of Utah, 1971

[16] Halas, J., and R. Manvell, The Technique of Film Animation, Hastings House, New York, 1968

[17] Hamilton, W.R., Lectures on Quaternions: Containing a Systematic Statement of a New Mathematical Method; of Which the Principles Were Communicated in 1843 to the Royal Irish Academy; and Which Has Since Formed the Subject of Successive Courses of Lectures, Delivered in 1848 and Subsequent Years, in the Halls of Trinity College, Dublin: With Numerous Illustrative Examples, Hodges and Smith, Dublin, 1853

[18] Lasseter, J., "Principles of Traditional Animation Applied to 3D Computer Animation", SIGGRAPH 87

[19] Laybourne, K., The Animation Book, Crown, New York, 1979

[20] Leaf, C., The Owl Who Married a Goose, film, National Film Board of Canada, 1974

[21] Leaf, C., The Metamorphosis of Mr. Samsa, film, National Film Board of Canada, 1977

[22] Magnenat-Thalmann, N., and Thalmann, D., Computer Animation: Theory and Practice, Springer-Verlag, Tokyo, 1985

[23] Marketing Department, Brochure, Showscan Film Corp., Culver City, CA, 1989

[24] Naylor, Bruce F., "A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes", Ph.D. Thesis, University of Texas at Dallas, May 1981

[25] Naylor, Bruce, John Amantides and William Thibault, "Merging BSP Trees Yields Polyhedral Set Operations", Computer Graphics, Volume 24, August 1990

[26] Newell, M.E., R.G. Newell, and T.L. Sancha, "A Solution to the Hidden Surface Problem", Proceedings of the ACM National Conference 1972

[27] Noll, M., "A Computer Technique for Displaying N-dimensional Hyperobjects", CACM, 10(8), August 1967

[28] Phong, B.T., "Illumination for Computer Generated Images", Ph. D. Thesis, Univ. of Utah, 1973

[29] Reeves, W. T., and R. Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Prticle Systems", SIGGRAPH 85

[30] Reynolds, C. W., "Computer Animation with Scripts and Actors", SIGGRAPH 82

[31] Roberts, L.G., Machine Perception of Therr Dimensional Solids, Lincoln Laboratory, TR 315, MIT, Cambridge, MA, May 1963

[32] Shoemake, K., "Animating Rotation with Quaternion Curves", SIGGRAPH 85

[33] Skala, Václav, Algoritmy počítačové grafiky I-III, Západočeská univerzita Plzeň, 1992

[34] Special Issue on Modeling the Human Body for Animation, CG & A, 2(11) N. Badler, ed., November 1982

[35] Stern, G., "Bbop - A System for 3D KeyFrame Figure Animation", in Introduction to Computer Animation, Course Notes 7 for SIGGRAPH 83, New York, July 1983

[36] Strauss, P., BAGS: The Brown Animation Generation System, Ph.D. Thesis, Technical Report CS-88-22, Computer Science Department, Brown University, Providence, RI, May 1988

[37] Sutherland, I. E., "Sketchpad: A Man - Machine Graphical Communication System", in SJCC, Spartan Books, Baltimore, MD, 1963

[38] Symbolics, Inc., S-Dynamics, Symbolics, Inc., Cambridge, MA, 1985

[39] Thibault, William C. and Bruce F. Naylor, Set Operations on Polyhedra Using Binary Space Partitioning Trees, Computer Graphics, vol. 21, 1987

[40] Warnock, J., A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures, Technical Report TR 4-15, NTIS AD-753 671, Computer Science Department, University of Utah, Salt Lake City, UT, June 1969

[41] Watkins, G.S., A Real Time Visible Surface Algorithm, Ph.D. Thesis, Technical Report *UTEC-CSc-70-101, NTIS AD-762 004, Computer Science Department, Univesity of Utah, Salt Lake City, UT, June 1970

[42] Wilhelms, J., "Using Dynamic Analysis for Realistic Animation of Articulated Bodies", CG & A, 7(6), June 1987

[43] Žára, Jiří a kolektiv, Počítačová grafika - principy a algoritmy, Grada, 1992