Modulární zobrazovací systém ReSt

Radim Halíř, Josef Vašica
MFF UK Praha,
Malostranské nám. 25,
118 00 Praha1,
e-mail: halir@cspguk11 resp. vasica@cspguk11

Abstrakt

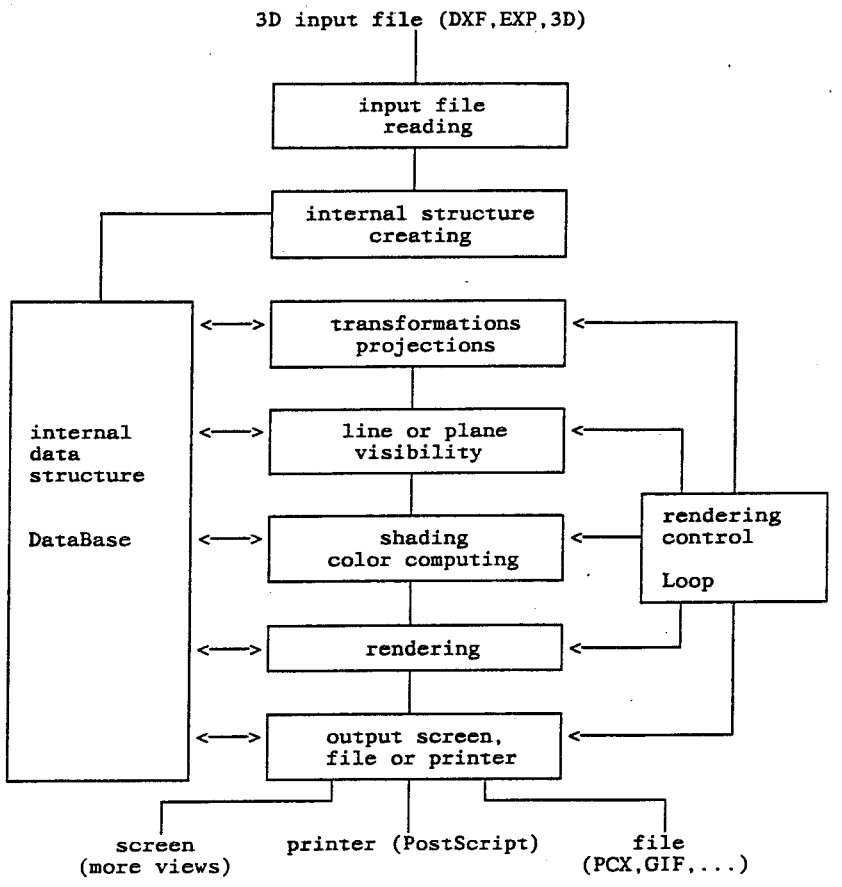
Cílem příspěvku je podat stručnou informaci o zobrazovacím systému ReSt, jeho návrhu i problémech při jeho implementaci.

Jedná se o uživatelsky modifikovatelný a rozšiřitelný systém pro zobrazování 3D scén. Oproti jiným zobrazovacím programům umožňuje ReSt uživateli kromě zobrazení scény pomocí standartních algoritmů (čárová i plošná viditelnost, různé druhy stínování) také modifikovat jednotlivé části zobrazovacího procesu od čtení scén přes projekci scény a způsob jejího zobrazení až po výběr výstupního zařízení či formátu. Tím systém plní hlavní cíl stanovený při jeho návrhu: umožnit experimentování se zobrazovacími algoritmy, aniž by uživatel musel kromě testovaného algoritmu psát cokoliv dalšího (tj. moduly pro čtení dat ze vstupních souborů, jejich organizaci ve vnitřní paměti v průběhu výpočtu či pro výstup výsledného obrázku). Danou scénu lze zobrazit i v několika různých pohledech, přičemž v každém pohledu je možné užít jinou projekci i zobrazovací algoritmus. Součástí programu není editor, požadovanou scénu je nutné vytvořit mimo tento systém (např. v AutoCADu či v PrIMu). Program je napsán v jazyce C++ s využitím prostředků objektového programování.

Úvod

Hlavním cílem vzniku tohoto systému bylo utvořit vyhovující prostředí pro experimentování s algoritmy počítačové grafiky. Tento systém tedy umožňuje modifikovat již utvořené moduly nebo napsat moduly nové a po začlenění do prostředí tohoto systému je používat na místě původních. Potom je možné při zkoušení nového algoritmu (např. plošné viditelnosti) naprogramovat pouze tento a zapojit jej na odpovídající místo. Není však nutné zabývat se vstupy a výstupy dat, určováním barev, výstupy v různých formátech atd.

Systém ReSt se nezabývá problémy vytváření scén a neobsahuje tedy ani editor těchto scén. Vstupní data je tedy nutné vytvořit mimo tento systém (AutoCAD, PriMo, ...) a potom je načíst pomocí známých datových formátů do prostředí systému. Tato data jsou při čtení převáděna do vnitřní reprezentace. Zvolení vhodného přístupu k tomuto problému je jednou z nejobtížnějších otázek při tvorbě podobného systému. Je nutné uvědomit si, že s vnitřní reprezentací budou pracovat všechny moduly a že budou vytvářeny i moduly nové. Musí být tedy dostatečně obecná, ale zároveň musí vycházet ze všeobecně známého teoretického modelu reprezentace.



Obr. 1: *Schéma systému ReSt*

Systém ReSt je koncipován tak, že neumožňuje pracovat s několika scénami najednou. Umožňuje však zpracovávat danou scénu několika různými způsoby. Tyto rozdílné přístupy k dané scéně se potom nazývají pohledy. V každém pohledu je potom možno definovat různé světelné zdroje, vlastní světelné charakteristiky těles, scénu různými způsoby transformovat atd. To vše nad stejnou základní bází dat.

Při zpracování daného pohledu scény nastává problém vzájemné komunikace mezi jednotlivými moduly a velké množství pracovních

dat. Moduly si mezi sebou potřebují předávat data v napřed zvoleném formátu : např. modul plošné viditelnosti předává modulu stínování informace o viditelných konvexních polygonech v rámci dané plošky. Je tedy nutno zajistit, aby nedocházelo k hromadění těchto dat, ale aby byla postupně zpracovávána a předávána skrz všechny následující moduly od čtení z databáze až po výstup na obrazovku nebo do zvoleného souboru.

Snadná modifikovatelnost všech již implementovaných modulů je podpořena výraznou modularitou celého systému (viz obr. 1). Takto navržený systém umožňuje změny v zobrazování scény pouhou výměnou daného modulu nebo zařazením nového modulu. Tyto moduly ovšem musí splňovat požadavky na jejich rozhraní dané jádrem systému.

Organizace dat zobrazované scény

Aby bylo možné zajistit snadnou modifikovatelnost a uživatelskou rozšiřitelnost celého systému, je nutné organizovat všechna data zobrazované scény v jediném objektu a ten pak mezi jednotlivými moduly systému sdílet. Tento objekt je v programu nazván DataBase.

Ještě dříve, než se dostaneme k popisu tohoto objektu, je nutné se několika slovy zmínit o vlastní reprezentaci objektů v jednotlivých scénách. Pro uložení zobrazovaných 3D scén byl zvolen tzv. povrchový model, neboť zrovna tato reprezentace objemu je nejvýhodnější pro řešení viditelnosti objektů, jejich stínování či vůbec vykreslování scény. Hlavním objektem při užití povrchového modelu je plocha; jednotlivá tělesa ve scéně jsou popsány pomocí konvexních rovinných plošek, které buď přesně kopírují nebo alespoň co nejlépe aproximují (např. u koule) povrch daného objektu. Konvexnost není u těchto plošek zcela nutná, ale vzhledem k tomu, že konvexní plošky radikálně zjednodušují algoritmy pro výpočet viditelnosti i pro vykreslování scény, užívají se téměř standartně. Také předpoklad rovinnosti plošek dále zjednodušuje tyto algoritmy. Navíc, pokud se použije dostatečně malých plošek v kombinaci se složitějšími metodami stínování (aproximace barvy nebo normály), namusí být toto úmyslné zkreslení povrchu objektů ve scéně vůbec patrné. Pro vlastní popis scény je pak užita modifikovaná struktura okřídlených hran.

Veškerá data zobrazované scény jsou v projektu uložena ve vnitřní paměti. Pomineme-li problém nedostatku paměti (který lze vyřešit virtualizací paměti), je s organizací rozsáhlých dat spojeno mnoho dalších problemů, například:

- jak daná data uložit do paměti co nejefektivněji vzhledem k velikosti potřebných pomocných informací (jedná se o různé pomocné ukazatele atd.)
- jak data v paměti organizovat, aby nad nimi šly co nejefektivněji provádět tzv. dynamické operace, to jest vytváření, modifikace a mazání daných dat; samozřejmým požadavkem je také co nejrychlejší přístup ke konkrétním datům

- jak minimalizovat fragmentaci paměti při uložení větší skupiny dat
- jak zajistit pokud možno co největší nezávislost dat na jejich konkrétním uložení v paměti
- je-li možné do nějaké struktury umístit data obecně různých typů (a také různých velikostí)
- ...

Tyto problémy byly velice uspokojivě vyřešeny užitím speciální datové struktury nazvané Handler. Ve své podstatě se jedná o množinu dynamicky přidělovaných a uvolňovaných paměťových stránek pevné velikosti obsahujících buď přímo požadovaná data nebo ukazatele na tato data umístěná v paměti. Výhodou přímého uložení dat je zabránění fragmentace a také úspora paměti, pokud budou vlastní data uloženy mimo a v jednotlivých stránkách budou pouze ukazatele na ně, je zase možné pracovat s daty různých typů a velikostí. Stav jednotlivých lokálních stránek (prázdná, ne zcela zaplněná nebo zcela zaplněná) je evidován pomocí bitových polí, čímž je např. umožněno rychlé vkládání nových objektů.

Každý uložený objekt je identifikován tzv. handlerem; jedná se o údaj složený ze dvou částí: indexu do globální tabulky jednotlivých stránek a pozice objektu v rámci dané lokální stránky. Na objekty se tedy neodkazujeme pomocí ukazatelů, ale právě pomocí handlerů. Tím je zaručena úplná nezávislost uložených dat na jejich umístění v paměti. Vzhledem k tomu, že handlery jsou poměrně malá celá čísla, je oproti použití ukazatelů dosaženo i značné paměťové úspory (hlavně při silně provázaných datových strukturách, jakými jsou například obousměrné seznamy či okřídlené hrany).

S daty lze v handlerech provádět veškeré základní operace od jejich vkládání, modifikace a mazání až po vyhledávání objektů s požadovanými vlastnostmi.

Pokud uživatel požaduje extrémní rychlost provádění operací s daty, může užít datové struktury Hash. Jedná se o implementaci hašovacího schématu organizace dat s lineárními oblastmi přetečení pro každou hašovanou hodnotu (tj. všechny prvky se stejnou hašovací hodnotou jsou uspořádány v lineárním seznamu). Tento přístup umožňuje do dané hašovací tabulky umístit libovolný počet prvků, se vzrůstajícím počtem prvků ovšem roste počet kolizí a tím i průměrný čas pro vyhledání požadovaného prvku. Díky lokálním oblastem přetečení lze také prvky nejenom rychle vkládat a přistupovat k nim, ale dokonce i poměrně jednoduše odstraňovat. Funkci pro výpočet hašovací hodnoty jednotlivých prvků definuje uživatel stejně jako funkci pro porovnání dvou prvků, čímž je umožněna snadná přizpůsobivost struktury Hash pro konkrétní aplikace.

K popisu scény jsou použity následující struktury: Point a Vertex pro body, Edge pro hrany, Plane pro plochy a Solid pro tělesa. Dále jsou zde ještě uchovány parametry jednotlivých pohledů View a další pomocné údaje. Veškerá data jsou v databázi uložena pomocí výše popsaných handlerů.

Všechny body scény jsou v databázi uloženy pomocí struktury Point. Pro uchování jejich souřadnic lze užít jak vektory (tj. tři složky X, Y, Z) nebo homogenní souřadnice (tj. čtyři složky X, Y, Z, V), formát volí uživatel sám dle konkrétní aplikace. Na této volbě formátu je celý zobrazovací proces zcela nezávislý, neboť obě struktury Vector a Homogen se na sebe umí automaticky převádět. Z uložených souřadnic bodů se dle parametrů jednotlivých pohledů počítají (při projekcích) souřadnice vrcholů vzhledem k daným pohledům. Pro zjednodušení některých operací obsahuje každý bod také odkaz na jednu z hran, která tento bod obsahuje. Aby bylo možné naráz vykreslit několik různých pohledů na danou scénu, bylo nutné umístit kopie jednotlivých bodů do každého z pohledů, v každém se souřadnicemi odpovídajícími tomuto pohledu. Vzhledem k tomu, že do těchto kopií je výhodné umístit i další informace potřebné pro vykreslování scény v daném pohledu, vznikla nová struktura Vertex.

Pro reprezentaci hran scény byla použita tzv. struktura okřídlených hran Edge. Každá hrana obsahuje: odkazy na oba koncové body hrany, odkazy na plochy, mezi kterými se tato hrana nachází a také odkazy na předchozí a následující hrany v obou plochách. Díky použité technice handlerů se lze z hrany odkazovat na původní souřadnice koncových bodů či na tyto souřadnice zprojektované vůči libovolnému pohledu (stačí, pokud bude mít objekt Point v databázi a objekty Vertex v pohledech vždy stejné handlery). Každá z hran je buď skutečná (normální hrany) nebo virtuální (neskutečná hrana vzniklá při aproximaci oblých povrchů rovinnými ploškami například u koule nebo u válce). V okřídlených hranách je ukryta většina informací o scéně (kromě optických vlastností ploch a těles), dále popisované struktury mají v databázi spíš pomocný charakter.

Prvním z těchto "pomocných" objektů je struktura Plane pro popis všech plošek ve scéně. Jelikož byla zvolena povrchová reprezentace objektů ve scéně, mělo by se vlastně jednat o nejdůležitější třídu celé databáze, ale vzhledem k tomu, že většina informací je již uložena v okřídlených hranách, obsahuje tato třída jen několik položek. Jsou to: barva plošky (plus její případné další optické charakteristiky), odkaz na libovolnou z hran této plošky (zbývající hrany lze zjistit pomocí informací uložených v okřídlených hranách), odkaz na těleso, do kterého ploška patří a položka pro normálu této plošky směřující ven z tělesa (pokud ploška ovšem netvoří plát nebo neuzavřené těleso; tato normála je důležitá pro řešení viditelnosti).

Jednotlivá tělesa scény jsou v databázi uložena pomocí struktur Solid. Za těleso se v tomto projektu považuje množina všech navzájem spojených ploch, tedy jako tělesa se chápou i pláty či jednotlivé osamělé plochy. Podle toho, z jakých plošek je těleso tvořeno, je každé těleso buď uzavřené (normální regulární tělesa) nebo tzv. oboustranné (tj. obsahující plošku, která může být viditelná z obou stran; jedná se např. o pláty či otevřená tělesa). Objekty Solid reprezentující tělesa obsahují pouze odkaz na libovolnou plochu tohoto tělesa a případně ještě charakteristiky společné pro všechny plochy, z nichž je dané těleso tvořeno (například průsvitnost či index lomu).

Při řešení viditelnosti a zobrazování scény lze uvažovat i vliv bodových světelných zdrojů, například při vykreslování vržených stínů nebo při výpočtu pomoci ray-tracingu.

Po tom, co jsme si popsali všechny struktury související s uchováním dat vlastní scény, dostáváme se k popisu struktury charakterizující jednotlivé pohledy na zobrazovanou scénu View. Jednotlivé pohledy umožňují uživateli dívat se na stejnou scénu z různých míst, zobrazit různé detaily, volit různé transformační a projekční funkce atd. Aby bylo možné zobrazit více pohledů na stejnou scénu současně, je nutné některé data z databáze scény v jednotlivých pohledech duplikovat. Jedná se o ty položky, které jsou závislé na parametrech jednotlivých pohledů; v našem případě, kdy uvažujeme pohledy na stejnou scénu (tj. ve všech pohledech bude zobrazena tatáž scéna, nijak upravovaná ani doplňovaná, jediným rozdílem mezi pohledy je různý způsob projekce souřadnic), stačí duplikovat pouze souřadnice bodů a případných světelných zdrojů, které se ve scéně vyskytují. Tento přístup částečně omezuje univerzalitu jednotlivých pohledů (v pohledech například nelze přidávat a ubírat nové objekty, měnit jejich topologii, posouvat je či jinak měnit), jeho výhodou je značná úspora paměti, neboť zbývající data scény mohou být všemi pohledy sdílena.

Vzhledem k tomu, že pohledy musí být nezávislé na dalším zpracování zprojektované scény (např. při řešení viditelnosti), obsahuje každý pohled souřadnice všech bodů a světelných zdrojů zpracovávané scény, tedy i těch, které případně nebudou v daném pohledu vůbec zobrazeny. S duplikací některých položek databáze v pohledech je spojen jeden problém: vždy, nezávisle na aktuálním pohledu, je třeba zajistit přístup ke korektním položkám (například od hrany k souřadnicím jejich koncových vrcholů). Vzhledem k tomu, že k organizaci všech dat v databázi jsou užity struktury Handler, stačí aby byla odpovídající data umístěna ve všech pohledech (a také ve vlastní databázi) na stejném místě (tj. aby měla ve všech strukturách stejný handler). Pak je výběr správných dat zajištěn pouhým výběrem požadovaného pohledu, bez jakýchkoliv změn v databázi.

Jakým způsobem má být scéna do jednotlivých pohledů zprojektována, specifikuje uživatel volbou parametrů pozorovatele a případnou dodatečně definovanou transformační funkcí. Pozice pozorovatele je charakterizována následujícími parametry: místem, kam se pozorovatel dívá, směrem, odkud se dívá, jeho vzdáleností a zvětšením scény. Dle těchto parametrů pak probíhá automatická projekce souřadnic do daného pohledu. Pokud uživatel vyžaduje dodatečnou transformaci souřadnic (např. perspektivní projekci) nebo chce provádět celou projekci sám, může tak učinit pomocí transformační funkce, kterou sám definuje. Může být použita jakákoliv transformace, jen je nutné si uvědomit, že se transformují pouze body a světelné zdroje, tj. že hrany zůstanou úsečkami a plochy se budou i nadále považovat za rovinné.

Vzhledem k tomu, že vlastní projekce souřadnic dle parametrů pozorovatele je lineární transformace, provádí se součinem

homogenních souřadnic s odpovídající projekční maticí uloženou v pohledu. Metoda, která projekce provádí, se stará o případný přepočet projekční matice (pokud byly změněny parametry pozorovatele) a do pohledů projektuje vždy jen ty body a světelné zdroje, u kterých je to potřeba (například při přidání nové plošky do databáze jen souřadnice vrcholů této plošky). Ty položky pohledu, které bylo nutné znovu zprojektovat, jsou označeny a je tedy možné, aby i dalšími moduly provádějící zpracování a vykreslení scény v daném pohledu znovu zpracovaly pouze tyto změněné prvky.

Nad strukturou DataBase je definováno mnoho metod, počínaje tvorbou nebo změnou parametrů jednotlivých objektů přes vytváření jednotlivých pohledů a projekce až po metodu pro čtení vstupních 3D souborů různých formátů (DXF, EXP, 3D) spojenou s vytvářením těles a správnou orientací normál všech plošek. Po načtení scény, vytvoření těles a orientaci normál je požadovaná scéna uložena v paměti a může začít její zpracování a vykreslení pomocí dalších modulů projektu ReSt (projekce, viditelnost, stínování, zobrazení, uložení vygenerovaného obrázku do souboru nebo jeho vytisknutí či vykreslení ...).

Řízení zobrazování tokem dat

Při zpracovávání již kompletního pohledu, tj. při jeho vlastním zobrazování, se objevuje problém velkého množství pomocných dat, která si musí jednotlivé moduly předávat. Pro názornost lze užít příkladu zobrazování stínovaných ploch. Předpokládejme, že modul viditelnosti je schopen určovat viditelnost konvexních oblastí těchto ploch a tyto oblasti předávat jako svůj výsledek. Modul stínování tyto oblasti přebírá a určuje barvu každého jejich bodu. Tyto body i s jejich barvami (např. v systému RGB) předává dále modulu zobrazování. Protože zobrazovaná scéna může mít tisíce plošek a v nich řádově stejné množství viditelných oblastí, nelze všechny tyto oblasti vypočítat najednou před určováním barvy. Je tedy nutné zajistit jejich postupné určování tak, že po zjištění několika takových částí se zavolá modul stínování a ten je zpracuje. Stejný problém je samozřejmě analogicky mezi stínováním a zobrazováním.

Celý program je potom řízen tokem dat, která se řadí do front před jednotlivými moduly. Po naplnění vstupní fronty je daný modul vyvolán, čímž ji začne zpracovávat a výsledky ukládá do výstupní fronty. Po úplném vyčerpání vstupní fronty se opět volá modul, který ji plní a tak stále dokola až do zpracování všech potřebných dat.

Při všech těchto operacích s pomocnými daty je nutno zajistit dostatečnou rychlost jejich ukládání i vyvolávání a zároveň jejich bezpečnost.

Popišme nyní jednotlivé složky tohoto řízení modulů od nejzákladnějších po globální.

Na nejnižší úrovni je potřeba zajistit uchovávání předávaných dat. Tato data lze ukládat mnoha způsoby - např. pro každý "balík" dynamicky alokovat paměť a takto vytvořené záznamy řadit do spojových seznamů. Tento přístup s sebou nese nevýhodu neregulované alokace paměti a při nesprávném hospodaření ji lze velmi rychle vyčerpat.

Předávaná data lze ale chápat jako posloupnost Bytů určitého formátu. Před tuto posloupnost lze ještě umístit hlavičku obsahující informaci o charakteru a velikosti těchto dat. Touto úvahou dostáváme konečnou podobu předávaných dat - nazývejme ji v následujícím zpráva. Struktura realizující tuto zprávu je velice jednoduchá :

```
struct { Type_t MsgType;
         Size_t MsgSize;
         char * MsgContent;
       } Message;
```

Poslední položkou je ukazatel na první znak této zprávy, která je potom kopírována. Zprávy tohoto formátu se ukládají do bufferů. Bufferem chápeme souvislou část paměti, do které se cyklicky ukládají jednotlivé Byty. Se strukturou bufferu jsou spojeny následující metody :

Err_t Put(*Message* *) ukládající danou zprávu do bufferu,
Err_t Read(*Message* *) čtoucí první zprávu bufferu do dané struktury,
Err_t Delete() mazající první zprávu z bufferu.

Možné návratové hodnoty těchto metod jsou následující :

INPUTEMPTY ... vstupní buffer je prázdný (pouze Read),
OUTPUTFULL ... výstupní buffer je plný (pouze Put),
CONTINUE ... vše je O.K.

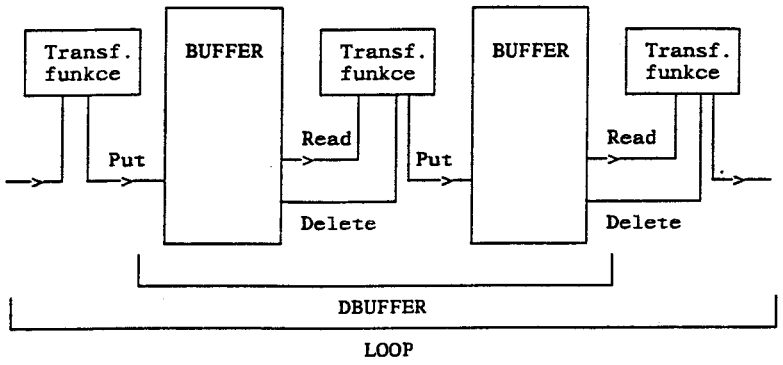
Metody pro čtení a mazání zpráv v bufferu jsou odděleny z důvodu bezpečnosti předávaných dat. Celkový význam tohoto rozdělení bude zřejmý z celkového schematu běhu jádra systému.

Jednotlivé buffery musí být ovšem vzájemně propojeny tak, aby data mohla "protékat" kolonou vytvořenou těmito buffery. Pro tento účel je vytvořena hierarchicky druhá struktura obsahující dva buffery a uživatelsky definovatelnou funkci transformující data vstupního bufferu do výstupního. Tato struktura má následující tvar :

```
class { Buffer * InputBuffer;
        Buffer * OutputBuffer;
        Err_t (* Transform)( Buffer *, Buffer *);
      } DBuffer;
```

Globální struktura (Loop) se potom skládá z posloupnosti takto vytvořených dvojic bufferů s tím, že výstupní buffer jedné dvojice je zároveň vstupním bufferem dvojice následující (viz obr. 2). Tento globální objekt také zajišťuje správné pořadí volání jednotlivých transformujících funkcí daných dvojic. Je

nutné zajistit, aby se po naplnění bufferu zavolala funkce, která tento vyprazdňuje. Pokud tento buffer bez problémů vyprázdní (tj. její výstupní buffer se nenaplní), vyvolá se opět předcházející funkce. Pokud je však výstupní buffer zaplněn a transformující funkce do něj chce zapisovat, je nutno jej nejprve uvolnit zavoláním následující transformační funkce a až potom pokračovat v zaplňování.



Obr. 2: *Struktury pro řízení tokem dat*

Zde se konečně dostáváme ke smyslu oddělení funkcí Delete a Read. Protože transformační funkce nemá povinnost uchovávat si přečtenou zprávu, může nastat situace, kdy po přečtení a zpracování již nemá možnost výsledek uložit do výstupního bufferu. V případě sloučení operací Delete a Read a vnitřního neuchovávání zpráv (uchovávání může být i paměťově náročné) dojde ke ztrátě poslední zprávy. Takto se ale v nejhorším případě tato zpráva přečte a zpracuje znovu.

V následující kapitole je popsána jedna z možných variant využití tohoto pojetí řízení programu tokem dat.

Metody stínování a jejich návaznost na vnitřní struktury projektu ReSt

Předpokládejme požadavek na maximální možnou kvalitu výstupního obrázku při využití standardních algoritmů plošné viditelnosti v rámci tohoto projektu. Utvoříme tedy následující kolonu modulů :

načtení scény | projekce do pohledů | plošná viditelnost | stínování | zobrazení na výstupní zařízení.

Modul stínování je tedy klasickým příkladem modulu pracujícího jak se vstupním, tak s výstupním bufferem. Musíme tedy mít definován tvar zprávy od viditelnosti i pro zobrazování.

O metodách viditelnosti lze obecně říci, že jsou schopny produkovat viditelné části ploch ve tvaru konvexního mnohoúhelníku v rámci jednotlivé plochy. Z tohoto již vyplývá tvar zprávy pro stínování. Obsahuje handler dané plochy, počet bodů daného mnohoúhelníku a seznam těchto bodů. Fakticky je tato zpráva rozdělena na následující části:

```
struct { HANDLER ParentPlane;
         Size_t PointNum;
       } ShadowMsgHead;
```

a po sobě následující

```
struct { Real_t X, Y, Z;
       } ShadowMsgPoint;.
```

Protože všechny implementované metody zpracovávají viditelné oblasti daných ploch po řádcích, tvoří tyto řádky současně i výstup těchto metod. Barva se určuje pro každý bod této řádky, což ovšem platí pouze teoreticky a v praxi se používá krokování přes tuto řádku o konstantní úsek, jehož hodnota je přístupná pro všechny moduly - umístěno v pohledu.

Výstupní zprávy lze rozdělit do dvou tříd. První - - obsahující pouze počet bodů dané řádky, koncové body a barvy všech vypočítaných bodů v formátu RGB - je vhodná zejména pro krátké řádky. Má následující tvar:

```
struct { Size_t PointNum;
       } ColorMsgHead; ,
```

dvakrát za sebou

```
struct { Real_t X, Y;
       } ColorMsgPoint;
```

a posloupnost

```
struct { Real_t R, G, B;
       } ColorMsgTable;
```

Druhá možnost výstupu stínování - vhodná zejména pro dlouhé řádky - obsahuje navíc tabulku barev použitých na dané řádce a na místě bodů řádky indexy do této tabulky. Má tedy tuto strukturu:

```
struct { Size_t ColorNum;
         Size_t PointNum;
       } ColorTableMsgHead; ,
```

posloupnost

```
struct { Real_t R, G, B;
       } ColorMsgTable; ,
```

dvakrát za sebou

```
struct { Real_t X, Y;
       } ColorMsgPoint;
```

a posloupnost

```
struct { Index_t ColorIndex;
       } ColorTableMsgIndex; .
```

Vlastní transformující funkce je napsána následujícím způsobem: a) Test úspěšného zapsání naposled vypočítané řádky - v případě úspěchu se pokračuje na dalším, v opačném případě vydá hodnotu **OUTPUTFULL** a při dalším vyvolání se tento řádek počítá znovu.

b) V případě dokončení celé oblasti načtení nové a následující test úspěšnosti této operace. V případě neúspěchu vydá hodnotu **INPUTEMPTY** a ukončí práci.

c) V případě úspěšného provedení všech operací vrací hodnotu **CONTINUE**.

Implementace

Systém ReSt byl vytvořen skupinou 6 studentů pod vedením dr. Pelikána jako softwarový projekt MFF UK Praha. Pro implementaci celého projektu byl použit jazyk C++, tedy objektové rozšíření populárního jazyka C. Byla vytvořena základní "kostra" volně rozšiřitelného systému, tj. navrhnut způsob řízení výpočtu tokem dat a implementována sdílená datová struktura DataBase. Z jednotlivých modulů byly naprogramovány následující: čtení souborů s popisem třírozměrné scény (formáty DXF, EXP a 3D), projekce scény do jednotlivých pohledů, čárová viditelnost (Appelův algoritmus), plošná viditelnost (Z-buffer), stínování (konstantní, Gourardovo i Phongovo), výstup vygenerovaného obrázku na obrazovku, tiskárnu (PostScript) a do souborů (GIF, PCX, oktalové stromy...).

Celý projekt v současné době obsahuje asi 10.000 řádků zdrojového textu. Systém ReSt je volně šiřitelný a je možné jej získat na výše uvedených adresách autorů.

Závěr

Cílem projektu nebylo vytvoření konkrétní aplikace, ale vůbec návrh jednoduše rozšiřitelného a modifikovatelného systému pro zobrazování třírozměrných scén. Tento úkol se použitím sdílené databáze scény a řízením výpočetního procesu tokem dat podařilo dostatečně uspokojivě splnit. Koncepce celého projektu nabízí mnoho možností dalšího rozšiřování a "vylepšování", ať již o klasické nebo i nové algoritmy a metody používané v třírozměrné počítačové grafice. Jako příklad uveďme např. zavedení kvalitnější reprezentace povrchů pomocí Beziérových a spline ploch, použití objemové reprezentace pomocí CSG modelu, antialiasing pro vylepšení kvality výsledných obrázků, výstup na plotter pomocí jazyku HPGL či zobrazení scény pomocí ray-tracingu nebo radiačními metodami.

Geometrické modelování tvarově složitých objektů

František JEŽEK
OI-ARC ZČU, Americká 42
306 14 PLZEŇ
e-mail: JEZEK@JONAS.ZCU.CS

1 Slovník pojmů, terminologie

Souřadnice

- kartézské
- homogenní – bodu (x, y) se přiřazují homogenní souřadnice $(\beta x, \beta y, \beta)$, vektoru [x,y] souřadnice [x,y,0].
- barycentrické – jedná se o souřadnice "na trojúhelníku". Označme $\mathbf{b}_i, i = 1, 2, 3$ vrcholy trojúhelníka, pak bod $\mathbf{p}$ má barycentrické souřadnice (a_1, a_2, a_3), pro něž

$$\mathbf{p} = a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + a_3\mathbf{b}_3$$

Afinní invariantnost – objekt má stejnou vlastnost před i po provedení afinní transformace (tj. násobení souřadnic regulární maticí).

Fergusonova kubika – uplatnění Hermitovy interpolace na křivky. Je určena počátečním a koncovým bodem a **tečnými vektory** v těchto bodech.

Spline – matematický model chování pružného laťkového křivítka. Kubický spline – po částech polynomická funkce (3. stupně) se spojitými derivacemi až do druhého řádu.

Parametrizace křivky – "časový režim" pohybu po křivce.

- přirozená – parametrem je délka oblouku,
- uniformní – každému oblouku "je přiděleno stejné časové kvantum",
- neuniformní
  - chordálová

$$t_{i+1} - t_i = c|\mathbf{P}_{i+1} - \mathbf{P}_i|$$

    Epstein (87) dokázal, že v opěrných bodech nemůže vzniknout singularita.
  - Leeova (dostředivá) – minimalizuje dostředivé síly

$$t_{i+1} - t_i = \sqrt{c|\mathbf{P}_{i+1} - \mathbf{P}_i|}$$