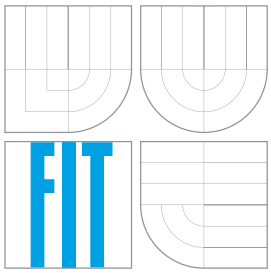
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SUBDIVISION SURFACES

SUBDIVISION SURFACES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DÁVID OTOČKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. STANISLAV SUMEC, Ph.D.

BRNO 2007

Zadání

1. Prostudujte a popište techniku Subdivision Surfaces.
2. Vyberte a navrhněte algoritmy pro efektivní realizaci techniky Subdivision Surfaces.
3. Implementujte knihovnu nebo modul pro vytváření těles technikou Subdivision Surfaces.
4. Demonstrujte funkčnost vytvořeného kódu na různých příkladech.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu, prezentujte projekt plakátkem.

Licenčná zmluva

Licenčná zmluva je uložená v archíve Fakulty Informačních Technologií Vysokého Učení Technického v Brně.

Abstrakt

Cieľom tejto práce bolo vytvoriť knižnicu pre realizáciu techniky Subdivision Surfaces a demonštrovať jej funkčnosť. Knižnica ako aj demonštračný program je napísaná v jazyku C++, na vytvorenie okna programu som použil WinAPI a pre prácu s grafikou OpenGL. Implementované sú tri základné algoritmy a to síce algoritmus Butterfly, Loop a Catmull–Clark. Na demonštráciu slúži pár základných objektov implementovaných priamo do knižnice.

Klíčová slova

Subdivision, Surfaces, programovací jazyk C++, OpenGL, WinAPI, Loop, Butterfly, Catmull-Clark

Abstract

The object of my bachelor work was to create a library, that will realise the technique of Subdivision Surfaces and demonstrate the functionality of the library on different examples. The library as well as the program is written by using C++ language, for window creation is used WinAPI and for work withthe graphics OpenGL. There are three implemented algorithms and those are Butterfly, Loop and Catmull–Clark. For demonstration are used few basic objects, that are implemented directly in the library.

Keywords

Subdivision, Surfaces, programming language C++, OpenGL, WinAPI, Loop, Butterfly, Catmull-Clark

Citace

Dávid Otočka: Subdivision Surfaces, bakalářská práce, Brno, FIT VUT v Brně, 2007

Subdivision Surfaces

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pana Stanislava Sumca Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Dávid Otočka
13. května 2007

Obsah

Kapitola 1

Úvod

V modernej dobe sa čoraz častejšie stretávame s úchvatným spracovaním tvarov niektorých modelov v počítačovej grafike. Sme prekvapení tým, ako dokonale vyzerajú postavičky a modely z animovaných filmov tvorených na počítači, či v počítačových hrách. Zlepšenie tohoto úkazu však nie je len vďaka lepšiemu hardvéru, ale hlavne vďaka inovatívnym technikám schopnými pracovať s rôznymi polygonálnymi modelmi, ktoré menia aj tie najhranatejšie modely na krásne hladké a oku lahodiace. Technika subdivision surfaces, ktorá bola predmetom mojej práce je jednou z takýchto techník. Vo svojej práci som sa pokúsil implementovať niektoré z algoritmov pre realizáciu tejto techniky. Pre implementáciu som sa rozhodol použiť jazyk C++. Mojou úlohou bolo vytvoriť knižnicu a zároveň aj demonštračný program, ktorý ju využíva. Program je implementovaný pre platformu Windows s využitím WinAPI. Pre prácu s grafikou je použité OpenGL.

V nasledujúcom texte sa budem zaoberať tým, na čo som narazil pri tvorbe knižnice ako aj programu. V druhej kapitole popisujem kedy sa technika Subdivision Surfaces dostala do povedomia ľudí a základné teoretické poznatky, ktoré treba vedieť pri jej realizácii. V nasledujúcej kapitole je to potom už teoretický rozbor algoritmov, ktoré som sa rozhodol implementovať. Nachádza sa tu ich teoretický popis a postupy využívané pri ich implementácii. V ďalšej kapitole je to už samotná tvorba knižnice, popis tried a štruktúr použitých v mojej implementácii, ich previazanosť, ako aj moja verzia implementácie jednotlivých algoritmov. V poslednej časti sa nachádza implementácia demonštračného programu a porovnanie implementovaných algoritmov na konkrétnom príklade. Na záver zhodnotím celú prácu navrhnem možné vylepšenia.

Kapitola 2

Subdivision Surfaces

V tejto kapitole sa budem venovať teoretickej časti ohľadom techniky Subdivision Surfaces. Kedy sa objavila, čo sa používa, aké sú pravidlá a schémy používané pri Subdivision Surfaces.

2.1 Čo je Subdivision Surfaces a prečo túto techniku použiť

Subdivision Surfaces je anglický termín a do češtiny, alebo slovenčiny by sa prekladal veľmi ťažko a preto sa používa tento anglický ekvivalent. Technika Subdivision Surfaces sa dostala do povedomia ľudí najmä od roku 1998, keď štúdio pre tvorbu animovaných filmov Pixar odhalilo jeho najnovší prírastok do rodiny krátkych animovaných filmov s názvom Geri's Game [8]. Bol to krátky film o starčekovi, ktorý v parku rád hraje šach sám so sebou. Nielenže bol tento film ohromujúco animovane spracovaný, ale v tej dobe bol na veľmi vysokej technologickej úrovni. Tento krátky film umožnil predstaviť novú zbraň v produkcii filmov štúdia Pixar a to síce techniku Subdivision Surfaces.

Subdivision Surfaces je technika popisujúca povrch objektu, využívajúc pri tom polygonálny model. Rovnako ako u polygonálneho modelu môže byť povrch ľubovolného tvaru, či veľkosti. Ale na rozdiel od polygonálneho modelu je povrch generovaný technikou Subdivision Surfaces perfektne hladký. Táto technika umožnuje vziať akýkoľvek originálny polygonálny model a za pomoci algoritmov, či už aproximačných, alebo interpolačných pridávať nové body a existujúce polygóny deliť na viac polygónov, ktoré budú menšie. Potom už vlastne ani nezáleží na tom ako ďaleko je kamera od objektu pretože povrch zaberaný kamerou sa môže znovu a znovu deliť až kým nebude vyzerať úplne hladký.

Každá schéma pre Subdivision Surfaces začína pôvodným polygonálnym povrchom, ktorý sa nazýva kontrolná mriežka. Potom sa kontrolná mriežka delí na ďalšie polygóny a všetky vrcholy sa posúvajú podľa nejakého súboru pravidiel. Pravidlá pre delenie sú rôzne od schémy ku schéme a práve od toho aké pravidlá sa použijú závisí tvar a vlastnosti povrchu, ktorý delením vznikne. Pravidlá väčšiny schém zachovávajú, alebo posúvajú pôvodné vrcholy a vkladajú nové. Sú aj schémy, ktoré pri každom kroku rušia staré vrcholy a vypočitavajú len nové, tie sú však v drvivej menšine.

Jednou z vecí, ktoré majú kontrolná mriežka a výsledný povrch spoločnú je, že sú vlastne topologicky identické. Topológia je spôsob popísania povrchu štruktúry, ktorý nie

je zmenený elastickou deformáciou, ako napríklad naťahovanie, či krútenie. Topológia je práve jedným z dôvodov prečo použiť Subdivision Surfaces. Môžete vytvoriť výsledný povrch z akejkoľvej ľubovolnej mriežky, ideálne uzatvorenej, pretože niektoré algoritmy majú s otvorenými mriežkami problémy a výsledný povrch môže mať ľubovolnú topológiu v závislosti od kontrolnej mriežky, ktorá doň vstupovala. Preto je technika Subdivision Surfaces ideálna pre modelovanie postáv a ďalších rôznych vecí, či už vo filmovom, alebo hernom priemysle. Podrobnejšie na [7].

Skôr ako sa bližšie začnem zaoberať algoritmami, tak najskôr vysvetlím základné veci týkajúce sa povrchov a mriežok v Subdivision Surfaces.

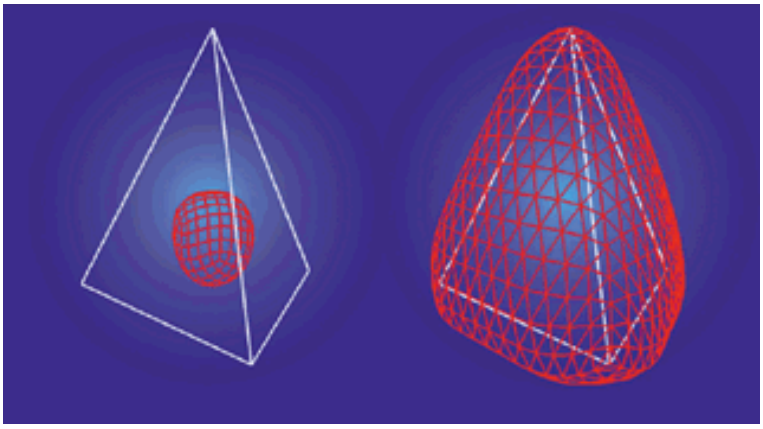
2.2 Kontinuita, interpolácia, aproximácia

Prvou základnou charakteristikou každej schémy je *kontinuita*. Schémy majú kontinuitu C^n, kde n znamená koľko derivácií je spojitých. Ak má povrch kontinuitu C^0, potom žiadna z derivácií nie je spojitá, čo znamená, že samotný povrch nemá žiadne otvorené diery. Ak je kontinuita povrchu C^1 znamená to uzatvorený povrch, ktorého dotyčnice sú spojité – nie sú tu žiadne ostré spoje. Väčšina povrchov, s ktorými sa pracuje má kontinuitu C^1, občas C^2 na niektorých miestach, ale vo všeobecnosti môžeme tvrdiť C^1. Kontinuita je práve ďalším významným dôvodom prečo použiť Subdivison Surfaces.

Pokým stupeň kontinuity je u väčšiny schém rovnaký, sú tu niektoré charatkteristiky, ktorými sa schémy od seba výrazne líšia. Jednou z nich je aj to, či je daná schéma *aproximačná*, alebo *interpolačná* a to závisí na type algoritmu, ktorý pre danú schému použijeme. Ak je daná schéma aproximačná, znamená to, že vrcholy kontrolnej mriežky neležia na výslednom povrchu. S každým krokom delenia na viac polygónov sa body kontrolnej mriežky posúvajú bližsie k výslednémy povrchu. Povrch generovaný aproximačnou schémou vyzerá veľmi dobre, môže mať veľmi málo zvlnení a záhybov. Aj keď bude mať kontrolná mriežka množstvo ostrých hrán, aproximačná schéma sa ich bude snažiť vyhladiť, čím viac sa budú ostrejšie body posúvať k výslednému povrchu. Na druhú stranu toto môže byť nevýhoda aproximačnej schémy, pretože sa s ňou niekedy ťažko pracuje. Je pri nej dosť ťažke mať víziu nového povrchu pri tvorbe kontrolnej mriežky a vytvoriť viacero zvlnení a záhybov, pretože schéma sa ich bude snažiť vyhladiť.

Ak je daná schéma interpolačná, znamená to, že body kontrolnej mriežky ležia na výslednom povrchu. Teda pri každom rekurzívnom kroku delenia sa s existujúcimi vrcholmi kontrolnej mriežky nehýbe. Výhodou interpolačnej schémy je, že na začiatku je viac zrejmé, ako bude výsledný objekt – povrch vyzerať, pretože vrcholy kontrolnej mriežky sú všade na výslednom povrchu. Avšak niekedy môže byť ťažké prinútiť interpolačný algoritmus, aby výsledný povrch vyzeral ako chceme, pretože na povrchu sa môžu vygenerovať vydutiny. Väčšinou to však nie je až taký veľký problém.

Obrázok 2.1 ukazuje príklad aproximačnej a interpolačnej schémy. Biela čiara je kontrolná mriežka a červená je čiarový model výsledného povrchu po niekoľkých krokoch delenia. Je pekne vidieť, že zakiaľ aproximačná schéma sa sťahuje ďalej od mriežky, tak interpolačná sa drží pri vrcholoch pôvodného povrchu. Viac informácii na [7].



Obrázok 2.1: Dve schémy delenia hranolu. Schéma na ľavo je aproximačná, schéma v pravo je interpolačná.

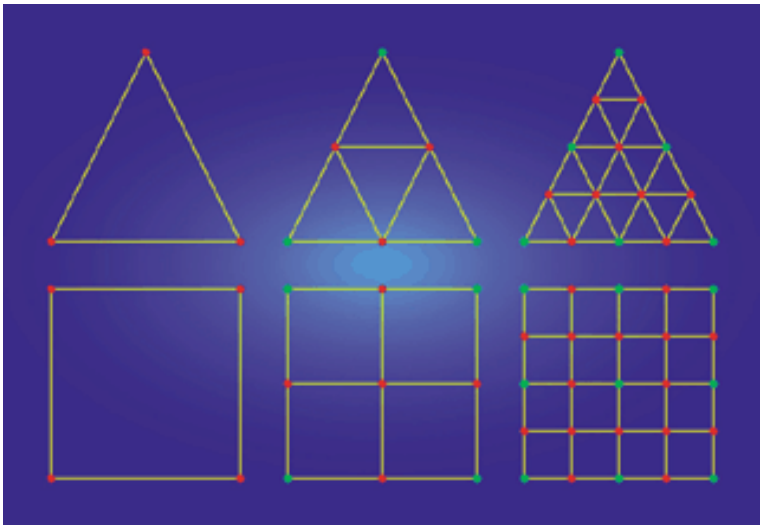
2.3 Uniformita, polygóny, extraordinálne body

Ďalšia charakteristika schémy prináša so sebou 4 ďalšie pojmy. Schéma môže byť *uniformná*, alebo *neuniformná*, *stacionárna*, alebo *nestacionárna*. Tieto pojmy znamenajú akým spôsobom sú pravidlá schémy – algoritmu aplikované na výsledný povrch. Ak je schéma uniformná znamená to použitie rovnakých pravidiel delenia pre všetky oblasti kontrolnej mriežky. Ak je však schéma neuniformná, môže byť na jednu hranu použité jedno pravidlo a na druhú iné pravidlo. Pri stacionárnej schéme je použitý pri každom kroku delenia ten istý súbor pravidiel, pokým v nestacionárnej schéme je pri každom kroku použitý iný súbor pravidiel delenia. Schémy, ktoré som implementoval sú všetky stacionárne a uniformné.

Inou chgarakteristikou schémy je tvar polygónov, s ktorými daná schéma pracuje. Preto možno vo všeobecnosti schémy rozdeliť na trojuholníkové, alebo štvoruholníkové. Prvá menovaná pracuje s kontrolnou mriežkou, ktorá je zložená z polygónov v tvare trojuholníka, druhá pracuje s polygónmi v tvare štvoruholníka. Väčsina štvoruholníkových schém pracuje s ľubovolnými n–uholníkmi a pri trojuholníkových schémach sa n–uholníky prevádzajú na trojuholníky. Niekedy však prevod môže radikálne zmeniť výsledný povrch.

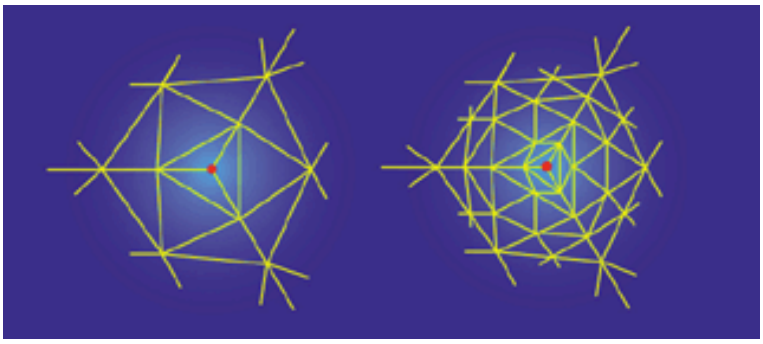
Obrázok 2.2 ukazuje trojuholníkovú schému v porovnaní so štvoruholníkovou. Zatiaľ, čo trojuholníková vkladá body na existujúce hrany, štvoruholníková musí vytvoriť nový bod. Preto sa dá povedať, že trojuholníková je o niečo ľahšia.

Ďalšou vlastnosťou schém je valencia vrcholov. Valencia vrcholu znamená vlastne počet hrán, ktoré z daného vrcholu vychádzajú. Väčšina vrcholov v schéme má rovnakú valenciu. Vrcholy s touto valenciou sú *regulárne* vrcholy danej schémy. Vrcholy, ktoré majú inú valenciu ako všetký ostatné vrcholy danej schémy sa nazývajú *extraordinálne*. Väčšina schém má problémy s analýzou povrchu v blízkosti extraordinálnych vrcholov a snažia sa ich neprodukovať. Takže počet extraordinálnych vrcholov je daný základnou kontrolnou mriežkou a nemení sa resp. meniť by sa nemal. Obrázok 2.3 ukazuje dva kroky s extraordinálnym bodom v strede. Je možné vidieť, že po kroku delenia extraordinálny bod zostáva, ale



Obrázok 2.2: Príklad troj- a štvor-uholníkového delenia polygónu.

nevzniká žiaden nový.

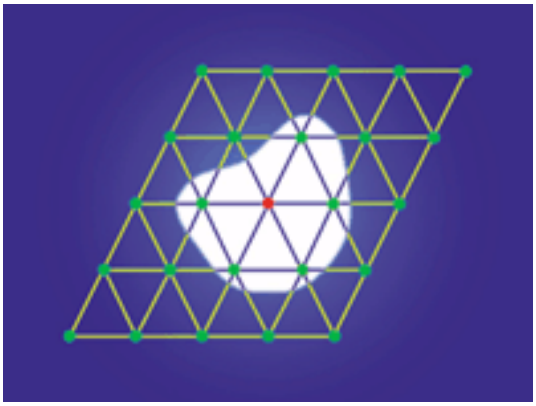


Obrázok 2.3: Mriežka pred(vľavo) a po(vpravo) jednom kroku delenia. Červený bod je extraordinálny vrchol.

2.4 Vyhodnotenie povrchu

Vyhodnotenie je povrchu je proces, pri ktorom sa zoberie kontrolná mriežka, pridávajú sa nové vrcholy, polygóny sa delia na viaceré, ktoré sú menšie, aby sa našla lepšia polygonálna aproximácia výsledného povrchu. Je mnoho ciest ako možno vyhodnotiť povrch, no všetky schémy sa dajú vyhodnotiť rekurzívne. A vačšina, vrátane tých, ktoré sú implementované, môže byť vyhodnotená pomocou vrcholových bodov danej kontrolnej mriežky. Pri interpolačnýc schémach to znamená, že môžete explicitne vypočítať normály daného povrchu pomocou vrcholových bodov, použitím tzv. tangentových masiek. Pri aproximačných schémach môžete explicitne vypočítať konečnú–limitnú pozíciu vrcholov, použitím tzv. vyhodnocovacích masiek. Maskou sa nemyslí maska binárna, ako by sme mohli poznať z programova-

nia, ale skôr ako výstrihy šablóny, ktoré sú umiestnené do kontrolnej mriežky a ich tvar nám potom ukazuje, ktoré z vrcholov a akou váhou sú použité pri výpočte výsledku. Obrázok 2.4 ukazuje príklad aplikácie takejto masky na povrch pri vrchole [7].



Obrázok 2.4: Hypotetická maska. Biela oblasť masky určuje, ktoré vrcholy sa použijú pri výpočte zahrnujúcom červený vrchol.

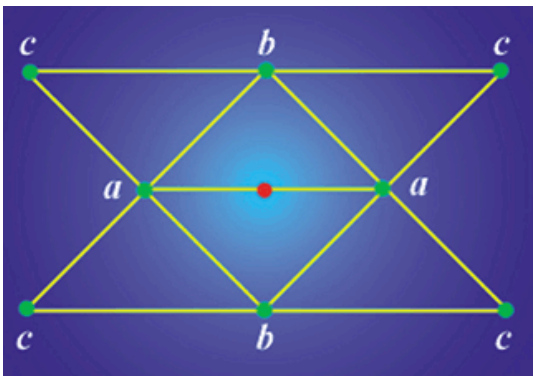
Kapitola 3

Algoritmy pre techniku Subdivision Surfaces

V tejto kapitole sa budem podrobne zaoberať mnou vybranými algoritmami pre realizáciu techniky Subdivision Surfaces.

3.1 Algoritmus Butterfly

Schéma, ktorú popíšem ako prvú sa nazýva Butterfly. V práci je implementovaný jednoduchý Butterfly, no existuje ešte modifikovaný Butterly, ktorý trochu priblížim no nebudem sa ním príliš zaoberať.



Obrázok 3.1: 8–bodová šablóna použitá v jednoduchej Butterfly schéme.

Schéma Butterfly ako taká, má zaujímavú históriu. V roku 1990 páni Dyn, Levin, a Gregory publikovali článok o tejto schéme a bol to vlastne popis vôbec prvej Butterfly schémy. Názov Butterfly znamená v preklade motýľ a je odvodený od tvaru šablóny, ktorá sa používa pri tvorbe nových vrcholov. Šablónu je možné vidieť na obrázku 3.1. Schéma je interpolačná a pracuje s trojuholníkovými polygónmi. Červený bod na obrázku je nový bod, ktorý vznikne po aplikácii šablóny na polygonálny model a výpočtom z pozícií vrcholov nachádzajúcich sa v šablóne. Táto schéma len pridáva vrcholy pozdĺž hrán, pravidlá pre pridávanie vrcholov sú pomerne jednoduché. Pre každú hranu sa vlastne zoberú body, ktoré patria do šablóny a pomocou ich váhy, sa vypočíta nový vrchol. Váha jednotlivých bodov, ktorá sa použije pri šablóne z obrázku 3.1 bude nasledovná:

$$a : \frac{1}{2}, b : \frac{1}{8} + 2w, c : -\frac{1}{16} - w \tag{3.1}$$

V rovnici 3.1 parameter w znamená, ako na tesno bude výsledný povrch posunutý ku kontrolnej mriežke. Ak je w rovné $-\frac{1}{16}$, potom schéma len lineárne interpoluje koncové vrcholy a povrch nie je hladký.

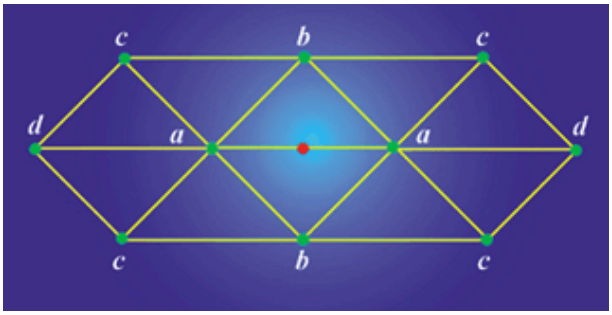
Problém pri schéme Butterfly vzniká keď sa v okolí vybranej hrany nedá aplikovať šablóna. Hlavne ak je valencia koncových vrcholov danej hrany menšia ako 5. Nie je dostatok informácií na použitie šablóny a práve vtedy je paremeter w rovný $-\frac{1}{16}$. Povrch generovaný týmto algoritmom teda nie je hladký v okolí extraordinálnych bodov. Tento problém rieši rozšírený algoritmus Butterfly 3.1.1.

3.1.1 Rozšírený butterfly

Ako som už napísal modifikovaný algoritmus Butterfly rieši aj otázku extraordinálnych bodov. Na výpočet nového bodu využíva trochu zložitejšiu šablónu, ktorú je možné vidieť na obrázku 3.2. Kde váha jednotlivých vrcholov je:

$$a : \frac{1}{2} - w, b : \frac{1}{8} + 2w, c : -\frac{1}{16} - w, d : w \tag{3.2}$$

Navyše ešte využíva špeciálnu šablónu na extraordinálne body. Ale keďže som rozšírenú verziu neimplementoval, tak ju nebudem podrobnejšie rozpisovať. Chcel som len poukázať na to, že existuje tak ako aj u množstva ostatných algoritmuv existujú iné modifikované verzie.



Obrázok 3.2: 10–bodová šablóna použitá v rozšírenej Butterfly schéme.

Podrobnejší popis o algoritme Butterfly, či jeho modifikovanej verzii na [7].

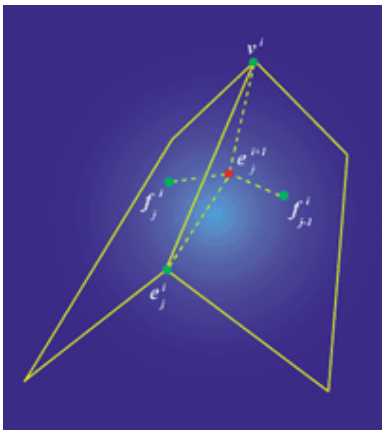
3.2 Algoritmus Catmull–Clark

Ďalšia schéma, ktorou sa budem zaoberať má názov po jej autoroch a to Edwinovi Catmullovi a Jimovi Clarkovi [10]. Je to schéma, ktorú v modifikovanej podobe Pixar použil v Geri's Game. Schéma Catmull–Clark pracuje s akýmkoľvek n–uholníkom, čo je jedna z jej výhod a produkuje štvoruholníky, čiže musí riešiť otázku pridávania bodov do mriežky. A kedže ide o schému aproximačnú musí sa vysporiadať aj s posúvaním starých vrcholov 2.2. Pravidlá pre túto schému sú preto o niečo komplexnejšie a výpočet nového vrcholu je robený po niekoľkých krokoch. Pravidlá sú nasledovné:

- Pre každý polygón v starej kontrolnej mriežke vypočítať nový bod, ktorý sa nachádza v strede daného polygónu – vypočíta sa ako priemer všetkých bodov, ktoré tvoria daný polygón. Pre lepšie vysvetlenie ho budem volať *face point*.
- Pre každú hranu v starej kontrolnej mriežke vypočítať nový bod, ktorý ako priemer dvoch koncových bodov hrany a dvoch stredových bodov polygónov, ktoré zvierajú danú hranu a boli vypočítané v predchádzajúcom kroku. Ilustrácia je na obrázku 3.3. Budem ho volať *edge point*.
- Na koniec posunúť staré vrcholy využívajúc pritom susediace body a to podľa vzorca:

$$\frac{F + 2R + (n-3)P}{n} \tag{3.3}$$

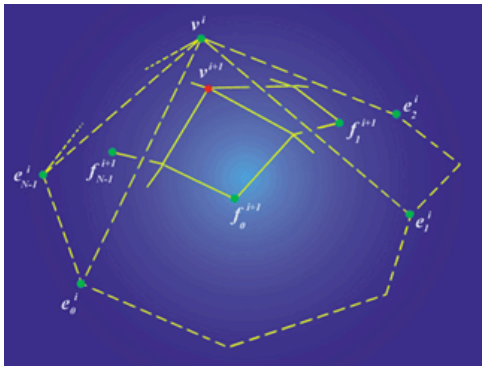
  Kde P je starý bod mriežky, F je priemer všetkých n face pointov pre polygóny dotykajúce sa P, R je priemer všetkých n stredových bodov hrán, dotykajúcich sa bodu P. Ilustrácia je na obrázku 3.4.



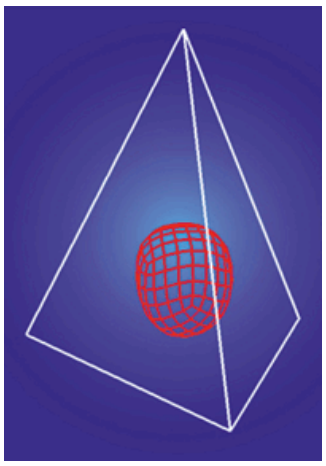
Obrázok 3.3: Ilustrácia výpočtu bodu pre hranu. Červený bod bude vypočítaný bod.

Na koniec sa vytvoria nové hrany správnym postupom pospájania bodov, ktorý je nasledujúci:

- Každý nový face point sa spojí s novým edge pointom tých hrán, ktoré definujú starý polygón.



Obrázok 3.4: Ilustrácia výpočtu posunu originálneho bodu(červený). Zelenou sú body použité pri výpočte.



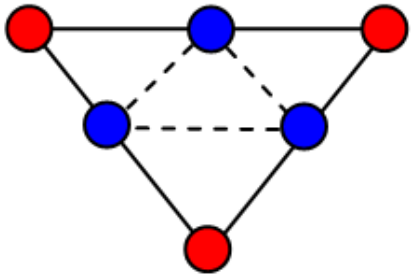
Obrázok 3.5: Kontrolná mriežka štvorstenu pred(biela) a polygonálny povrch po(červená) niekoľkých krokoch delenia algoritmom Catmull–Clark.

- Každý nový vrchol, ktorý vznikol posunutím starého vrcholu, spojím s edge pointmi hrán, ktoré vychádzali zo starého vrcholu.

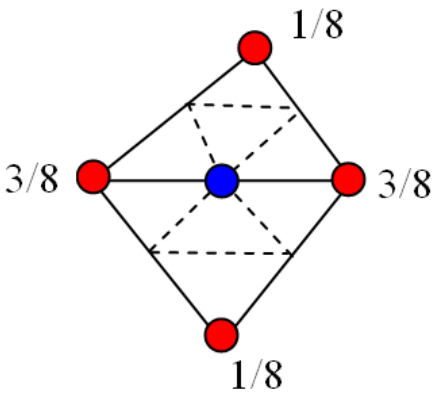
Pospájanie jednotlivých hrán ešte podrobne rozoberiem v mojej implementácii schémy Catmull–Clark. Po jednom kroku delenia budú všetky polygóny kontrolnej mriežky štvoruholníky. Preto schéma vnáša nové extraordinálne body len počas prvého kroku delenia. To znamená, že po prvom kroku je pevne daný počet extraordinálnych vrcholov a už sa ďalej nemení. Obrázok 3.5 ukazuje štvorsten po niekoľkých krokoch delenia algoritmom Catmull–Clark. Informácie som čerpal z [7], [4] a [10].

3.3 Loop–ov algoritmus

Poslenou schémou, ktorá je v ráci implementovaná je schéma Charlesa Loopa, ktorú publikoval ako svoju diplomovú prácu na univerzite v Utahu [5]. Loop–ova schéma je aproximačná, ale na rozdiel od schémy Catmull–Clark pracuje len s trojuholníkovou kontrolnou mriežkou. Pravidlá pre realizáciu tejto schémy sú preto o niečo jednoduchšie.



Obrázok 3.6: Vytvorenie 4–och 3–uholníkov z 1.



Obrázok 3.7: Body a ich váha použité pri výpočte bodu hrany.

Pravidlá pre vytvorenie nových a posunutie starých bodov v Loop–ovej schéme sú nasledovné:

- Pre každú hranu v kontrolnej mriežke pridaj nový bod a pre každý trojuholník v mriežke vytvor 4 nové pospájaním bodov hrán, ktoré boli práve vytvorené. Spôsob výpočtu bodu hrany je jednoduchý a je ilustrovaný na obrázku 3.7. Každá hraná má 2 koncové vrcholy a pomocou týchto 2 koncových vrcholov môžeme určiť protilahlé vrcholy danej hrany. Potom spravíme lineárnu kombináciu týchto vrcholov ako sú na obrázku 3.7 a dostaneme nový vrchol hrany. Celý proces tvorby 4-och trojuholníkov z 1-ného je potom na obrázku 3.6.

- Každý vrchol, ktorý sa nachádza v starej kontrolnej mriežke je aj v novej a pri výpočte jeho pozície sa použijú všetky body, s ktorýma susedí, tj. všetky body, s ktorými je spojený hranou. Počet vrcholov, s ktorými starý vrchol susedí je n a udáva hodnotu

konštanty β. Pre výpočet konštanty β existuje viacero vzorcov, ale najjednocuchší je asi:

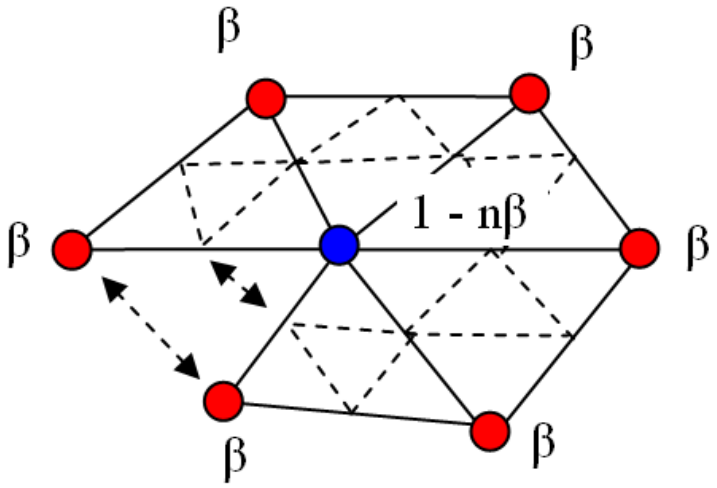
$$\beta = \begin{cases} \frac{3}{8n} & \text{pre } n > 3 \\ \frac{3}{16} & \text{pre } n = 3 \end{cases}$$

Staré vzorce, použité aj v implementácii sú:

$$\beta = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} - \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right) \tag{3.4}$$

$$V^{i+1} = (1 - n\beta)\, V^i + \beta \sum_{j=0}^{n} V_j^i \tag{3.5}$$

Ilustrácia je na obrázku 3.8.



Obrázok 3.8: Body použité pri výpočte pozície vrcholu v novej mriežke a konštanta β.

Loop–ov algoritmus je jeden z najrozšírenejších a práve preto som sa ho rozhodol implementovať. Informácie, ako aj inšpiráciu k programu som čerpal z [3], [2] a [5].

Kapitola 4

Tvorba knižnice

V tejto kapitole sa budem venovať tvorbe knižnice a tomu, ako som do knižnice metódu Subdivision Surfaces implementoval. Popíšem štruktúry a použité funkcie, s krátkym popisom ich činnosti.

4.1 Samotná knižnica

Inšpiráciu ako vytvoriť knižnicu som hľadal v [1]. Celý projekt Subdivision Surfaces som realizoval v prostredí Microsoft Visual Studio©. Knižnica je vlastne kolekcia podprogramov–funkcií, ktoré sú zapúzdrené v jednom module a môže ich využívať viacero programov. Pôvodný nápad bol hneď vytvárať knižnicu a potom k nej spraviť demonštračný program, avšak nakoniec som sa rozhodol napísať všetko naraz ako jeden celý program. Každý algoritmus, ako aj mriežka majú svoje vlastné hlavičkové a zdrojové súbory. Potom čo som celý projekt vyskúšal a odladil, oddelil som vlastný demonštračný program od súborov budúcej knižnice a nastavením parametrov prekladača v programovacom prostredí Microsoft Visual Studio som vytvoril súbor *Subdivision.lib*, ktorý mi zapúzdruje všetky funkcie a štruktúry použité v algoritmoch pre realizáciu techniky Subdivision Surfaces. Demoštračný program už v konečnej podobe využíval len funkcie z tejto knižnice ako to požadovalo zadanie práce. Všetok kód knižnice je napísaný v programovacom jazyku C++. Knižnica by mala byť prenosná aj na iné platformy, pretože mimo vlastne definovaných využíva len štandartné funkcie C++.

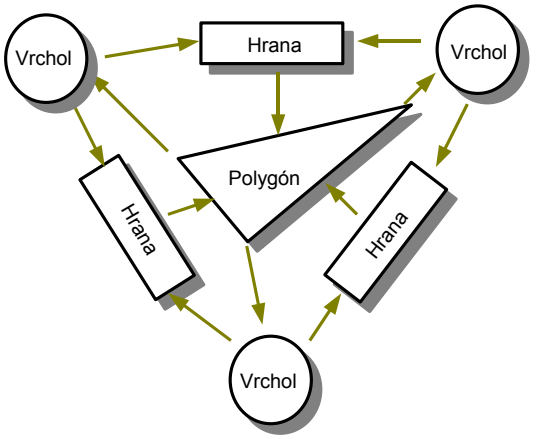
Väčšina vecí v knižnici je riešená staticky a preto je knižnica dosť náročná na hardvér. Po niekoľkých iteráciách sa doba výpočtu algoritmu viditeľne spomaľuje a preto nie je príliš vhodná na delenia, ktoré majú viac ako 4–5 krokov. V ďalších sekciách sa budem bližšie venovať implementácii jednotlivých častí knižnice.

4.2 Reprezentácia kontrolnej mriežky

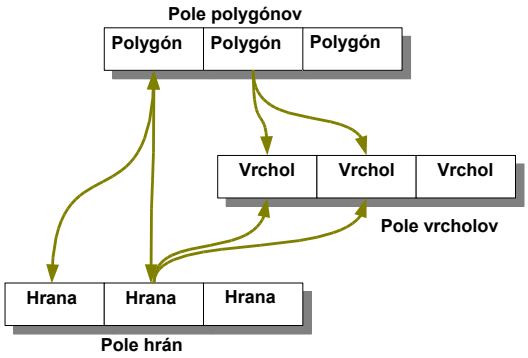
Hlavným pilierom celej knižnice je reprezentácia kontrolnej mriežky. Bez dostatočne dobrej reprezentácie dát by sa algoritmy, ktoré využívajú šablóny, alebo výpočet bodov pomocou okolitých bodov len ťažko realizovali. Inšpiráciu pre tvorbu štruktúry, ktorá by obsahovala všetky údaje, ktoré potrebujem som hľadal [6]. Svoju predstavu štruktúry som implementoval do triedy *SubdivisionMesh*, od ktorej sú potom dedené triedy pre jednotlivé algoritmy. Trieda *SubdivisionMesh* obsahuje funkcie spojené s pridávaním polygónov, vrcholov, hrán,

počtom vrcholov, polygónov, výpočtom normál a iné.

Hlavnými množinami reprezentujúcimi údaje sú 3 polia, ktoré obsahujú dôležité informácie o kontrolnej mriežke. Jedným je pole, ktoré obsahuje štruktúry polygónov. Ďalším je pole obsahujúce štruktúry hrán a posledným tretím poľom je pole obsahujúce všetky vrcholy danej mriežky. Ďalším poľom je pole normál jednotlivých polygónov, no to nie je pri výpočtoch algoritmov až také dôležité a využíva sa hlavne pri vykreslovaní.



Obrázok 4.1: Všeobecná štruktúra topologickej informácie v datovej štruktúre kontrolnej mriežky. Všimnite si informácie, ktoré by mali mať jednotlivé časti.



Obrázok 4.2: Previazanosť polí obsahujúcich informácie o dôležitých častiach mriežky.

Kedže som hovoril o existencii poľa štruktúr polygónov a poľa štruktúr hrán je jasné, že v rámci triedy *SubdivisionMesh* sú definované ešte 2 štruktúry a to síce štruktúra *Polygon*, ktorá obsahuje pole indexov vrcholov, z ktorých je daný polygón zložený a pole hrán, ktoré tvoria daný polygón. Ďalšou štruktúrou je štruktúra *Edge*, ktorá obsahuje 4 indexy. 2 indexy bodov tvoriacich danú hranu a 2 indexy polygónov, ktoré danú hranu zvierajú. Môže sa to

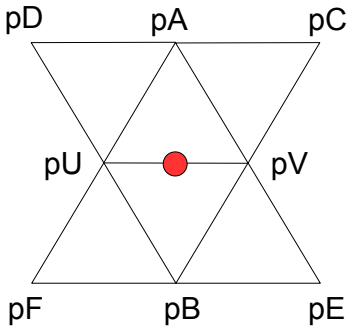
na prvý pohľad zdať trochu mätúce, ale to ktoré informácie sú potrebné krásne ilustruje obrázok 4.1. Podľa toho, čo som popísal o svojej triede, tak mám všetky tieto informácie uložené. To ako sú polia v mojej triede previazané ilustruje obrázok 4.2. Preto už nebol až taký problém implementovať algoritmy.

4.2.1 Demonštračné objekty

Na to, aby sa knižnica mohla vyskúšať sú v nej pridané niektoré základné objekty. A to síce kocka zložená zo štvoruholníkových polygónov, kocka zložená z trojuholníkových polygónov a hranol. Možnosťou je aj načítanie zjednoduchého .obj súboru. Pozor však, knižnica pracuje len s *uzavretými* kontrolnými mriežkami. Knižnica je navyše implementovaná tak, že algoritmy, ktoré pracujú len s trojuholníkovými kontrolnými mriežkami sú schopné pracovať aj s ľubovolnými n–uholníkmi a to vďaka funkcii *Triangulate()*. Táto funkcia je totiž volaná pred každým algoritmom pracujúcim s trojuholníkovou mriežkou a jej prácou jej prevedenie n–uholníkovej mriežky na trojuholníkovú. Vďaka tejto funkcii sú algoritmy Loop a Butterfly v mojej knižnici schopné pracovať s akýmkoľvek n–uholníkom. Ak chcete načítať .obj súbor treba ho zadať ako parameter príkazového riadku pri špúšťaní programu. Formát vstupujúceho .obj súboru by mal byť nasledovný, s iným formátom program nepracuje a môže vypisovať chybové hlášky:

```
v  1.0  0.0  0.0
v  1.0  0.0  1.0
v  0.0  1.0  0.0
f  1  2  3
```

4.3 Butterfly



Obrázok 4.3: Body a šablóna použitá pri implementácii Butterfly. Červenou je nový bod.

Algoritmus Butterfly je implementovaný v súboroch *butterfly.h* a *butterfly.cpp*. Trieda *ButterflySubdivisionMesh* je dedená od triedy *SubdivisionMesh*. Nachádzajú sa v nej len

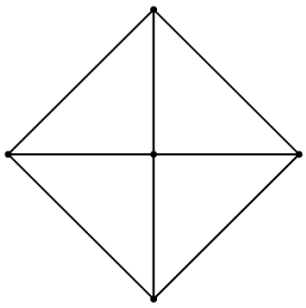
2 funkcie a to síce funkcia *NewVertices()*, ktorej úlohou je výpočet nových vrcholov za pomoci šablóny butterfly. Funkcia funguje zhruba tak, že prechádza všetky hrany kontrolnej mriežky a pre každú hranu si uloží koncové body pU a pV. Potom prejde celé pole polygónov a pomocou bodov pU, pV zistí, ktoré 2 polygóny zvierajú danú hranu a teda určí body pA a pB. Po zistení týchto dvoch bodov potom postupuje podobne ako pri zisťovaní bodov pA,pB a určí aj zvyšné body "krídeľ" šablóny butterfly a to pC, pD, pE a pF. Nakoniec vypočíta podľa vzorca 4.1 nový bod hrany a uloží ho do poľa nových bodov, ktoré je jej návratovou hodnotou a má vlastne rovnakú indexáciu ako pole hrán. Rozloženie bodov a šablóna použitá pri mojej implementácii je na obrázku 4.3.

Vzorec pre výpočet nového bodu hrany E:

$$E = \frac{1}{16}\left(8\left(pU + pV\right) + 2\left(pA + pB\right) - \left(pC + pD + pE + pF\right)\right) \tag{4.1}$$

Druhou funkciou je funkcia *Subdivide()*, ktorej úlohou je už len prechod všetkých polygónov starej mriežky a ich rozdelenie na menšie polygóny za pomoci poľa nových bodov získaných z funkcie *NewVertices()*. A to tak, že si pre starý bod v polygóne zistí indexy 2 hrán, ktoré majú spoločný bod, ktorým je jeden zo starých vrcholov mriežky a vďaka rovnosti indexov v poli hrán a v poli nových bodov vypočítaných pre každú hranu pridá ďalšie 2 vrcholy z poľa nových vrcholov. Pre každú hranu jeden, ktoré budú tvoriť nový polygón po spojení so starým vrcholom. Nový polygón pridám do mriežky. Toto sa opakuje 4–krát, čím z jedného trojuholníku vytvorím 4. Inšpiroval som sa tu [2].

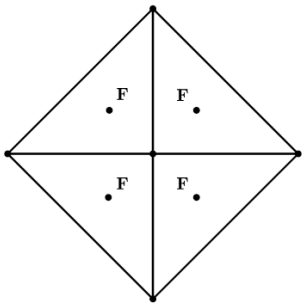
4.4 Catmull–Clark



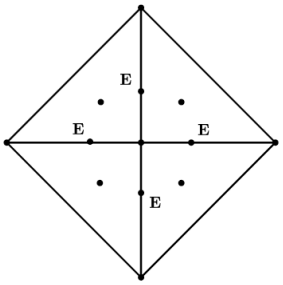
Obrázok 4.4: Príklad časti trojuholníkovej mriežky.

Pre implementáciu algoritmu Catmull–Clark, slúži trieda *CatmullSubdivisionMesh*, ktorá je dedená z triedy *SubdivisionMesh* a obsahuje všetky dôležité funkcie pre tento algoritmus a funkciu *Subdivide()*, ktorá vykonáva 1 krok delenia kontrolnej mriežky. Na začiatku mám bod, spájajúci n polygónov. Pri vysvetlení budem uvažovať trojuholníkovú mriežku, ktorú mozžno vidieť na obrázku 4.4.

Najskôr vo funkcii *FacePoints()* prechádzam celé pole polygónov a pre každý počítam stredový bod, tak ako je to znázornené na obrázku 4.5 a výsledné pole tvorí návratovú hodnotu tejto funkcie. V ďalšej funkcii, do ktorej vstupujem s poľom bodov pre polygóny,



Obrázok 4.5: Znázornenie nových bodov, vypočítaných pre každý polygón.

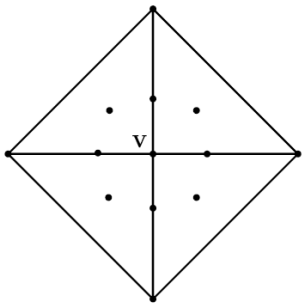


Obrázok 4.6: Znázornenie nových bodov, vypočítaných pre každú hranu.

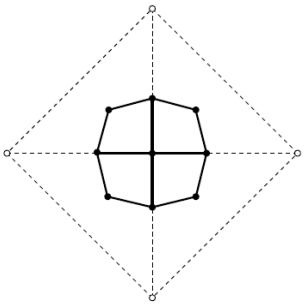
ktoré vrátila funkcia *FacePoints()*, si vypočítam nové body hrán. Body hrán sa vypočítajú ako priemer 2 koncových bodov tvoriacich danú hranu a 2 stredových bodov polygónov zvierajúcich danú hranu. Výsledok je na obrázku 4.6.

Nakoniec ešte vo funkcii *VertexPoints()* vypočítam nové polohy starých bodov a to za pomoci vzorca 3.3. Výsledok pre daný príklad je na obrázku 4.7. Vo funkcii *Subdivide()* potom pospájam všetky vypočítané body a to tak, že prechádzam celé pole vrcholov a pre každý vrchol pridávam do výslednej kontrolnej mriežky polygón tvorený novým vrcholom, 2 bodmi hrán vychádzajúcich z daného vrcholu a bodom polygónu, ktorému patria hrany použitých bodov hrán. Koľko polygónov pridám záleží na na počte polygónov obsahujúcich nový vrchol, s ktorým pracujem. Ilustrácia je na obrázku 4.8.

Inšpiráciou ako na túto schému bola [4], kde je schéma Catmull–CLark podrobne rozobraná.



Obrázok 4.7: Nová poloha vrcholu. V dvojrozmernom vnímaní sa javí na tom istom mieste ako predtým, no v 3D nemá tú istú polohu.



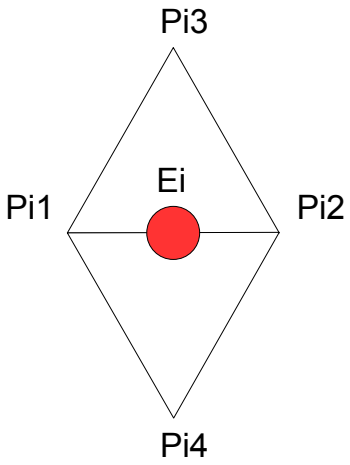
Obrázok 4.8: Výsledok po spojení bodov a pridaní polygónov.

4.5 Loop

Algoritmus Loop je realizovaný prostredníctvom triedy *LoopSubdivisionMesh* dedenej z *SubdivisionMesh*. Trieda *LoopSubdivisionMesh* obsahuje 3 základné funkcie pre realizáciu schémy Loop. Hlavnou je opäť funkcia *Subdivide*, ktorá má na starosť 1 krok delenia kontrolnej mriežky. Najskôr si pre každý jeden vrchol starej kontrolnej mriežky vypočítam jeho novú polohu a to pomocou vzorca 3.5 spomenutého v 3–tej kapitole. Pri výpočte novej polohy potrebujem poznať počet všetkých susediacich vrcholov s daným vrcholom a toto mi zabezpečuje funkcia *VertexNeighbourCount(int VertexIndex)*, ktorá je súčasťou triedy, *SubdivisionMesh*. Potom čo poznám počet všetkých vrcholov susediacich s daným vypočítam konštantu β a to podľa vzorca 3.4 uvedeného v 3–tej kapitole.

Ku konečnému výpočtu potrebujem ešte sumu koordinátov susediacich vrcholov a toto mi zabezpečuje ďalšia funkcia, ktorá je súčasťou triedy *LoopSubdivisionMesh* a to síce funkcia *VertexNeighbourCoordSum(int vertex)*. Po výpočte nových polôh pre každý vrchol kontrolnej mriežky vypočítam nové body hrán. Prejdem celé pole hrán a pre každú vypočítam jeden bod. Pre výpočet bodu hrany slúži funkcia *ComputeEdgeVertex(int edge)* a pracuje

ako je ilustrované na obrázku 4.9 využívajúc vzorec 4.2.



Obrázok 4.9: Šablóna použitá pri výpočte bodu hrany.

$$E^i = \frac{3}{8}\left(P_1^i + P_2^i\right) + \frac{1}{8}\left(P_3^i + P_4^i\right) \tag{4.2}$$

Keď mám vypočítané nové polohy starých vrcholov a nové body pre každú hranu, tak ich už len pospájam a to tým spôsobom, že z jedného trojuholníku v starej kontrolnej mriežke dostanem štyri v novej. Nakoniec pridám polygóny do kontrolnej mriežky. Pri implementácii algoritmu Loop som si pomohol tu [2] a [3].

Kapitola 5

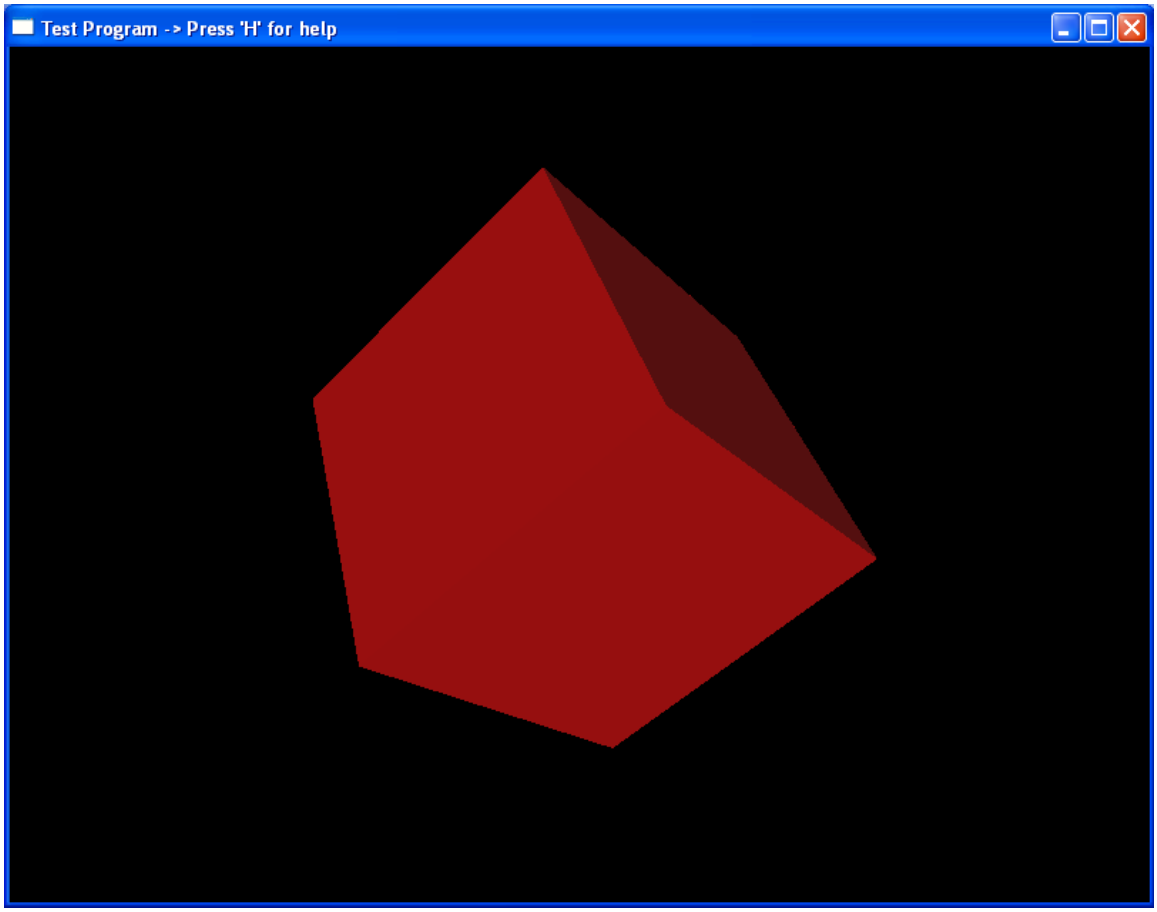
Demonštračný program

Môj demonštračný program využívajúci knižnicu *Subdivision.lib* je vytvorený pre platformu Windows. Pre správne spustenie je dôležité, aby systém podporoval štandart OpenGL. Pre dobré zobrazenie a manipuláciu s programom je potrebný silnejší výpočetný výkon a dobrá grafická karta. Program by nemal mať problémy so spustením ani na horších počítačoch, ktoré obsahujú systém Windows XP, avšak potom treba počítať s tým, že aplikácia sa bude trhať a výpočet nového povrchu objektu technikou Subdivision Surfaces bude podstatne dlhší. Úvodnú obrazovku tvorí prvý demonštračný objekt a možno ju vidieť na obrázku 5.1.

Prepínanie medzi objektami pre demonštráciu, ktoré sú implementované priamo v knižnici som použil klasický príkaz *switch*. Pred volaním schém pracujúcich len s trojuholníkovými kontrolnými mriežkami volám funkciu *Triangulate()*, ktorá je tak isto súčasťou knižnice *Subdivision.lib*. Jej popis je v sekcii 4.2.1. V programe je tiež implementovaná základná rotácia objektu, aby bol vidieť z viacerých strán, nemožno s ním však manipulovať ani zoomovať.

Po stlačení klávesy H vyskočí okno s nápovedou. Nápoveda k ovládaniu programu vyzerá takto:

- A – zoom out.
- Z – zoom in.
- L – Loop subdivision. Vykoná jeden krok delenia schémou Loop.
- B – Butterfly subdivision. Vykoná jeden krok delenia schémou Butterfly.
- C – Catmull–Clark subdivision. Vykoná jeden krok delenia schémou Catmull–Clark.
- N – naplní kontrolnú mriežku ďalším demonštračným objektom v poradí.
- W – prepína medzi čiarovým a polygonálnym modelom.
- E – vypíše čas trvania algoritmu v sekundách v presnosti na tisíciny.
- T – zmení polygóny mriežky na 3-uholníky ak je to potrebné.
- I – výpis počtu polygónov a vrcholov tvoriacich objekt.
- R – resetuje mriežku.



Obrázok 5.1: Úvodné okno po spustení programu.

Objekty medzi, ktorými možno prepínať sú kocka zložená zo štvoruholníkových polygónov, kocka zložená z trojuholníkových polygónov a hranol, prípadne objekt načítaný zo súboru vo formáte, ako je popísaný v kapitole 4.2.1.

5.1 OpenGL a WinAPI

Demonštračný program je napísaný v jazyku C++ a pre zobrazenie využíva prácu s knižnicou OpenGL. Objekty sú zobrazené ako polygonálne modely, ale aj ako čiarové modely. Pri polygonálnom zobrazení sa potom ešte rozhoduje či sú to polygóny štvor– alebo troj–uholníkové. Pred naprogramovaním zobrazenia som si pozorne preštudoval tutoriál na [9], z ktorého som použil základné veci pre realizáciu zobrazenia ako aj vytvorenia okna.

Samotný program je odladený za pomoci knižníc *glut* a preto je pre jeho chod nutná knižnica *glut.dll*. Pre vytvorenie okna je použité WinAPI, čo je programovacie rozhranie používané pre komunikáciu s operačným systémom Windows. To ako správne vytvoriť okno v rozhraní WinAPI a ako s ním správne manipulovať som sa dozvedel tu [9].

5.2 Porovnanie jednotlivých algoritmov

V tejto kapitole zhrniem výsledky testovania rýchlosti algoritmov na počítači Pentium 2,66Ghz, 512MB RAM a 128MB grafickej karte. Pre zistenie časovej náročnosti algoritmov bola v demonštračnom programe využitá knižnica *ctime*. Funkcia *clock()* zmerala čas pred algoritmom a po algoritme a potom som tieto dva časy od seba odčítal a tak dostal výsledný čas algoritmu. Výsledky testovania zhrňuje nasledujúca tabuľka:

Kroky delenia	**Objekt**	**Počet vrcholov a polygónov**	**Algoritmus**	**Čas[s]**
0.	kocka	12 poly, 8 vrch	žiadny	0
1.	kocka	36 poly, 38 vrch	Catmull–Clark	0.016
2.	kocka	144 poly, 146 vrch	Catmull–Clark	0.125
3.	kocka	576 poly, 578 vrch	Catmull–Clark	1.14
4.	kocka	2304 poly, 2306 vrch	Catmull–Clark	16.484
1.	kocka	48 poly, 26 vrch	Loop	0.031
2.	kocka	192 poly, 98 vrch	Loop	0.094
3.	kocka	768 poly, 386 vrch	Loop	0.859
4.	kocka	3072 poly, 1538 vrch	Loop	11.078
1.	kocka	48 poly, 26 vrch	Butterfly	0.016
2.	kocka	192 poly, 98 vrch	Butterfly	0.11
3.	kocka	768 poly, 386 vrch	Butterfly	0.937
4.	kocka	3072 poly, 1538 vrch	Butterfly	10.781

Tabulka 5.1: Výsledky testovania náročnosti algoritmov na kocke.

Je možné vidieť, ako algoritmus Catmull–Clark vytvára menej polygónov a vrcholov a to je zapríčinené tým, že výstupné polygóny z algoritmu Catmulll–Clark sú štvoruholníky.

Kapitola 6

Záver

Počítačová grafika je veľmi zaujímavý odbor s rýchlym vývojom, najmä vďaka svetu hier a filmov a stále sa zlepšujúcemu výkonu hardvéru. Väčšinu informácii k mojej práci som čerpal na internete, kde sú v súčastnosti asi najaktuálnejšie informácie. Dnešné polygonálne modely, či už v hrách, filmoch alebo niekde inde sú zobrazené s dokonalým a hladkým povrchom nech sa kamera priblíži akokoľvek blízko. Práve toto je väčšinou docielené technikou Subdivision Surfaces a nikto nevie kam ju v nasledujúcich rokoch ešte posunieme.

Jednou z úloh mojej bakalárskej práce bolo vytvoriť knižnicu pre realizáciu techniky Subdivision Surfaces. V práci sú implementované 3 základné algoritmy techniky Subdivision Surfaces a ich funkčnosť je demonštrovaná na demonštračnom programe. Demonštračný program zároveň umožňuje načítanie zo súboru, takže implementáciu knižnice je možné otestovať aj na reálnych dátach. Myslím si, že úlohu som splnil. Určite by sa práca dala rozšíriť najmú v oblasti optimalizácie, poprípade pridania ďalších algoritmov. A to hlavne v rýchlosti algoritmov pri väčšom počte polygónov. Ďalším rozšírením by mohla byť práca nielen s uzatvorenými, ale aj otvorenými kontrolnými mriežkami.

Technika Subdivision Surfaces ma zaujala od chvíle, ako som ju začal detailnejšie rozoberať a rozšírila moje obzory v oblasti počítačovej grafiky ako aj programovania, či už v jazyku C++, alebo prácou s OpenGL. Rád by som sa v budúcnosti venoval tejto problematike ďalej.

Zoznam použitých zdrojov

[1] Eckel, B.: *Thinking in C++, Volume 1, 2nd Edition*. Prentice Hall Inc., 2000.

[2] Endres, S.: Subdivision Project. [online], [cit. 2007-05-06].
URL `http://vorlon.case.edu/~sxe19/index.html`

[3] Fisher, M.: Subdivision. [online], [cit. 2007-04-28].
URL `http://www.its.caltech.edu/~matthewf/Chatter/Subdivision.html`

[4] Joy, K. I.: Catmull-Clark Surfaces. [online], [cit. 2007-04-26].
URL
`http://graphics.idav.ucdavis.edu/education/CAGDNotes/Catmull-Clark.pdf`

[5] Loop, C. T.: *Smooth Subdivision Surfaces Based on Triangles*. Diplomová práce, Department of Mathematics, The University of Utah, 1987, [cit. 2007-04-28].
URL `http://research.microsoft.com/%7Ecloop/thesis.pdf`

[6] Robert F. Tobler, S. M.: A Mesh Data Structure for Rendering and Subdivision. Technická zpráva, VRVis Research Center, Donau-City Wien, 2006.

[7] Sharp, B.: Subdivision Surface Theory. [online], [cit. 2007-04-26].
URL `http://www.gamasutra.com/features/20000411/sharp_pfv.htm`

[8] Studios, P. A.: Geri's game. [online], [cit. 2007-04-26].
URL `http://www.pixar.com/shorts/gg/index.html`

[9] Turek, M.: CZ NeHe OpenGL vše o programování 3D grafiky pod knihovnou OpenGL. [online], [cit. 2007-05-07].
URL `http://nehe.ceskehry.cz/`

[10] Wikipedia: Catmull-Clark subdivision surface. [online], [cit. 2007-04-28].
URL `http://en.wikipedia.org/wiki/Catmull-Clark_subdivision_surface`

Zoznam príloh

A Polygonálne a mriežkové modely kocky na začiatku, a v každom z troch krokov delenia metódou Catmull–Clark, Loop, Butterfly.

B CD, ktoré obsahuje súbory so zdrojovými kódmi knižnice a programu, programovú dokumentáciu, elektronickú verziu práce a spustiteľný demonštračný program.

Príloha A