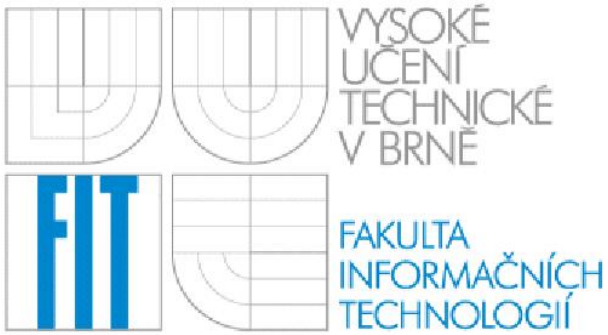
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ




# Diplomová práce

**2006** **Petr Bulíček**

## Poděkování

**Chci poděkovat svému vedoucímu Ing. Janu Pečivovi za přátelský přístup a pomoc při řešení obtížných problémů.**

## Prohlášení

**Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Pečivy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.**

.....

**Petr Bulíček**

## Abstrakt

Projekt je zaměřen na optimalizaci zobrazování velmi rozsáhlých scén. Obsahuje návrh a vytvoření systému generujícího vesmírná tělesa, která podle potřeby mění úroveň detailů a k vytváření scén pomocí těchto těles. Popisuje použité metody a navrhuje možné rozšíření systému.

Generovaná tělesa mají pseudonahodný tvar vytvořený na základě požadované velikosti a členitosti. Na scénách, které jsou těmito tělesy vytvářeny, je demostrováno využití algoritmu pro snížování úrovně detailů.

Pro demonstraci algoritmu na snižování detailů je rovněž implementován pohyb po scéně pomocí myši. A pro lepší dojem z prezentační scény jsou v ní implementovány kolize.

Tento systém byl vytvořen, aby byl použit jako součást akademického projeku Space-Game. Proto je navržen tak, aby jeho uživatelský vstup byl co nejjednodušší a rychlost generování těles byla co možná největší.

Program je vytvořen v jazyce C++ s využitím grafického rozhraní OpenGL a knihovny Open Inventor.

## Klíčová slova

Vesmírné těleso, úroveň detailů, OpenGL, Open Inventor, snímků za sekundu, FPS

## Abstract

This project is direct to optimize display very wide scenes. It include design and building system, which is generating space solids which can change level of detail and scenes compound from this solids. Project describes used methods and propoese possible extends.

Generated solids have pseudorandom shape based on desired size and segmentation. On created scenes is demonstrated application algorithm for lowering details.

For demonstration level of detail alghoritm was implementet movement throught scene. And for better sensation was implemented collision too.

System was created as a part of academic project SpaceGame. Therefore is propose, in order to user input was easy and speed of generating solids was as fast as posible.

Project has been created in C++ with graphic interace OpenGL and Open Inventor library.

## Key words

Space solid, level of detail, OpenGL, Open Inventor, frame per second, FPS

# Obsah

**1 Úvod ..... 7**

**2 Struktura programu ..... 8**

**3 OpenGL v programu ..... 9**

**3.1 Co je to OpenGL ..... 9**

**3.2 Co je Open Inventor? ..... 11**

**3.3 Využití technik OpenGL ..... 12**

**4 Vesmírné těleso ..... 13**

**4.1 Co je vesmírné těleso? ..... 13**

**4.2 Stavba vesmírného tělesa ..... 13**

**5 Algoritmus na dělení trojúhelníků ..... 14**

**5.1 Způsob rozdělení ..... 14**

**5.2 Implementace rozdělení ..... 14**

**6 Zvýšení detailu ..... 15**

**6.1 Docílení vyššího detailu ..... 15**

**6.2 Kam posunout nový bod? ..... 15**

**6.3 Implementace posunu bodu ..... 16**

**7 Úrovně detailů ..... 18**

**7.1 Co je to úroveň detailu ..... 18**

**7.2 Proč používat úrovně detailů ..... 18**

**7.3 Metody přepínání úrovní detailů ..... 20**

**8 Použití vesmírného tělesa ..... 21**

**9 Scény ..... 23**

**9.1 Scény pro demonstraci výsledků ..... 23**

**9.2 Prezentační scéna ..... 23**

**9.3 Testovací scéna ..... 25**

**10 Generování do souboru ..... 26**

**11 Pohyb ve scéně ..... 27**
**11.1 Důležitost pohybu ..... 27**
**11.2 Způsob ovladání pohybu ..... 27**
**11.3 Přírustek dráhy ..... 27**

**12 Detekce kolizí ..... 28**
**12.1 Mezí čím se kolize detekují ..... 28**
**12.2 Snížení počtu detekcí ..... 28**
**12.3 Způsob detekce a reakce na kolize ..... 28**
**12.4 Kde jsou detekce implementovány ..... 29**

**13 Komponenty zobrazující měřené hodnoty ..... 30**
**13.1 Důvod použití komponent zobrazujících měřené hodnoty ..... 30**
**13.2 Popis komponenty zobrazující měřené hodnoty ..... 30**
**13.3 Použití komponent zobrazujících měřené hodnoty ..... 31**
**13.4 Popisky grafů ..... 31**

**14 Vyhodnocení výsledků ..... 33**
**14.1 Změny detailů jednoho tělesa ..... 33**
**14.2 Změny detailů ve scénách ..... 35**
**14.2.1 Prezentační scény ..... 35**
**14.2.2 Testovací scény ..... 36**
**14.3 Vyhodnocení naměřených hodnot ..... 38**
**14.3.1 Měření nad jedním tělesem ..... 38**
**14.3.2 Měření nad scénami s konstantní velikostí testovací krychle ... 39**
**14.3.3 Měření nad scénami s konstantní vzájemnou vzdáleností těles 41**
**14.3.4 Shrnutí výsledků ..... 46**

**15 Možná rozšíření ..... 47**

**16 Závěr ..... 48**

**17 Literatura ..... 49**

**18 Přílohy ..... 50**

**19 Seznam obrázků ..... 52**

# 1 Úvod

Problematika snižování úrovně detailů je v realtime grafice nutností, pokud chceme docílit atraktivního vzhledu. Vývoj a výroba výkonnějšího hardwaru jde sice dopředu obrovskou rychlostí, ale ruku v ruce s ním jde dopředu i vývoj vizuálně lepších a hardwarově náročnejších efektů používaných v grafických aplikacích.

Chceme-li u aplikace, která se renderuje v reálném čase, dosáhnout plynulého pohybu, je zapotřebí minimalizovat počet zpracovávaných primitiv tak, aby výsledek byl z vizuálního hlediska přijatelný. Toto je hlavním cílem této práce. V tomto případě jsou tyto primitivy trojúhelníky (viz literatura [4]) a hlavním principem pro změnu úrovně detailů je algormitmus, který tyto trojúhelníky dělí. V následujících kapitolách se pokusím objasnit, jak tento program pracuje.

V kapitole 2 uvádím soubory z nichž se projekt skládá a implementační jazyk.

V kapitole č. 3 najdeme informace o rozhraní OpenGL a Open Inventor a o jejich vzájemném postavení.

V kapitole č. 4 je řečeno, co je to vesmírné těleso a čím je tvořeno.

5. kapitola vysvětluje princip algoritmu na delělení trojúhelníků a nastiňuje způsob jeho implementace.

V kapitole č. 6 najdeme informace důvodu zvyšování detailu, jak k tomu dochází a jak je zvýšení detailu v tomto projektu implementováno.

7. kapitola nám říká, co je to úroveň detailu, proč je používáme a jakým způsobem je přepínáme.

8. kapitola ukazuje použití jednoho vesmírného tělesa.

V 9. kapitole je popsáno jaké sceny jsou v této práci pomocí vesmírných těles vytvářeny.

V 10. kapitole je vysvetlen důvod a okolnosti generování scén do souborů a jejich zpetného načítání.

11. kapitola odůvodnuje implementaci pohybu po scéně a popisuje způsob ovládání pohybu.

Kapitola č. 12 pojednává o kolizích o jejich detekci a minimalizací počtů detekcí.

Kapitola č. 13 vysvetluje význam komponent, které zobrazují hodoty měřené v projektu.

14. kapitola obsahuje vyhodnocení dosažených výsledků. Popisuje jak vizuální výsledky optimalizovaných scén, tak výkonové.

Ve 15. kapitole jsou uvedena možná rozšíření o která by mohla být v budoucnu práce obohacena.

V rámci ročníkového projektu byla zpracovaná část týkající se tvorby vesmírného tělesa, vytvoření úrovní detailů a jejich přepínání. Jedná se o kapitoly 4, 5, 6 a 7.

V semestrálním projektu byl doladěn způsob přepínání úrovní detailů a tvorba scén. Jsou to kapitoly 7 a 9.

# 2 Struktura programu

Program je vytvořen v jazyce C++ s využitím grafického rozhraní OpenGL. Nad rozhraním OpenGL je použita ještě knihovna Open Inventor, sloužící pro tvorbu realtimové grafiky.

Samotný systém se skládá z osmnácti souborů. Šest souborů obsahuje zdrojové kódy. Mnou vytvořené jsou create.cpp a create.h. Obsahují třídu pro generovování vesmírných těles a třídu pro vytváření scény z těchto těles. Soubory SoPerfGraph.cpp, SoPerfGraph.h, SoScreenPositioner.cpp a SoScreenPositioner.h obsahují třídy pro zobrazení informace o počtu snímků za vteřinu, celkovém počtu trojůhelníku ve scéně, počtu viditelných trojúhelníků a rychlosti pohybu kamery. Ty jsou vytvořeny vedocím mé práce, Ing. Janem Pečivou. Zbylé soubory jsou textury, které jsou nanášeny na tělesa pro lepší vizuální dojem. Jsou to earth.jpg, earthmab.jpg, fullmapb.jpg, jupeqb.jpg, mars.jpg, marsb.jpg moon.jpg, moonb.jpg neco.jpg, sun.jpg, venus.jpg a venusmeb.jpg. Tyto textury byly staženy ze stránky *http://www.cortland.edu/flteach/civ/DavidWeb/resources.htm*.

Jelikož tento systém byl vytvořen, aby byl součástí většího projektu, je iterakce s uživatelem řešena pouze formou paramerů se kterými se projekt spouští z příkazové řádky.

# 3 OpenGL v programu

## 3.1 Co je OpenGL

OpenGL je softwarové rozhraní ke grafické kartě. Příkazy OpenGL se používají ke specifikaci objektů a operací potřebných k vytvoření trojrozměrné aplikace(viz literatura [4] a [7]).

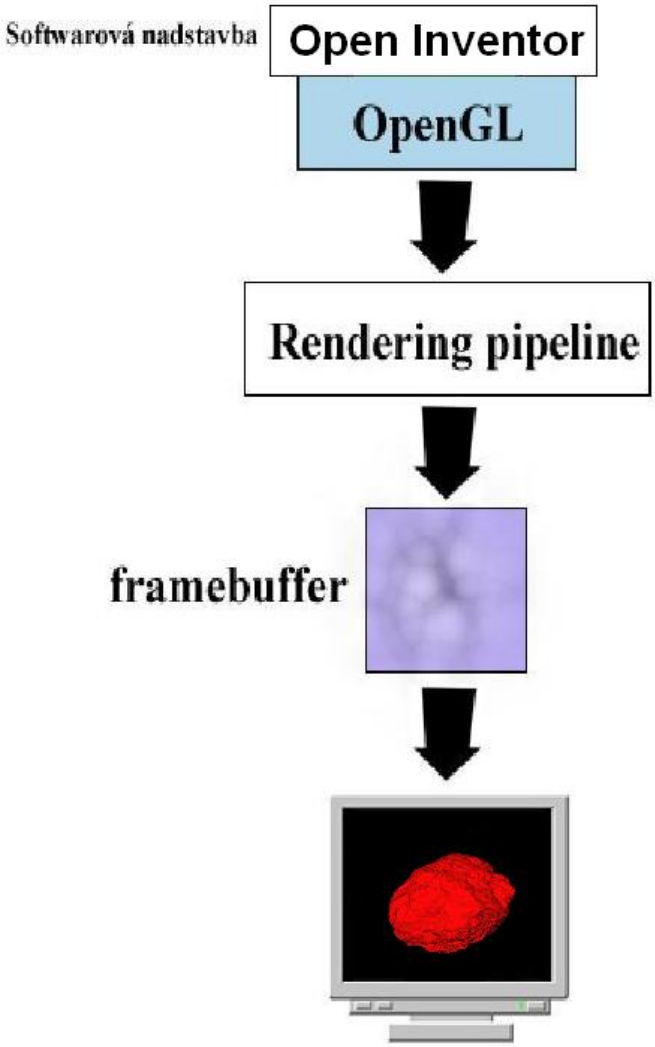
OpenGL je hardwarově nezávislé rozhraní. Dnes je dostupné ve většině operačních systémů (Windows, Unix, Linux, Sun, Irix). Tato hardwarová a platformová nezávislost je umožněna díky nepoužívání příkazů pro práci s okny nebo pro zpracování uživatelského vstupu (viz literatura [4]a [7]).

Pomocí OpenGL lze vykreslit model pouze použitím základních příkazů vykreslujícím body, čáry a polygony (viz literatura [4]).

Při zobrazení pomocí této metody jsou na vstupu objekty 3D scény, které jsou převedeny na 2D obrázek pomocí vykreslovacího řetězce (tzv. rendering pipeline viz Obrázek 1). Tento obrázek je uložen v tzv. paměti snímku (framebufferu viz Obrázek 1), odkud se již zobrazuje na monitor. Zobrazení snímku viz Obrázek 1 (viz literatura [4] a [7]).

OpenGL je tzv. stavový stroj. Pokud nastavíme v programu nějakou vlastnost, pak platí do té doby, dokud není nastavena vlastnost jiná (viz literatura [4] a [7]).

Výhodou OpenGL je, že vývojář na její použití nepotřebuje žádnou licenci.



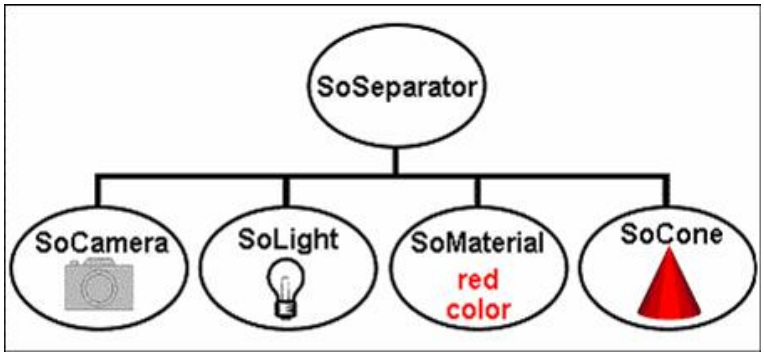

Obrázek 1: Vykreslení pomocí OpenGL a knihovny Open Inventor (převzato viz [19])

## 3.2 Co je Open Inventor

Open Inventor je velmi populární knihovna pro tvorbu realtimové 3D grafiky, tedy i her. Poskytuje programátorovi rozsáhlou množinu C++ tříd, které skrývají před programátorem vlastní OpenGL API a posunují ho na mnohem vyšší úroveň. Tak může programátor mnohem rychleji vyvinout to, co potřebuje (viz literatura [5]).

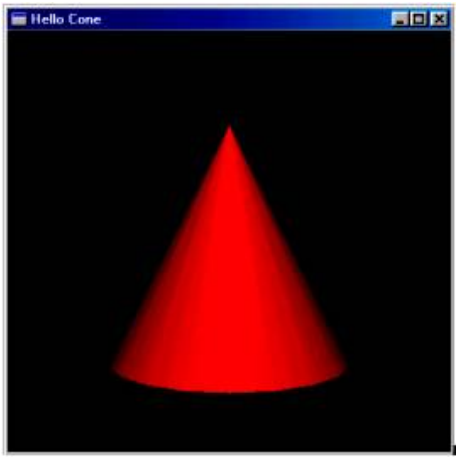
Navíc, aplikace napsané v Open Inventoru jsou obyčejně rychlejší než ty přímo psané v OpenGL a jsou rovněž platformově nezávislé, stejně jako OpenGL (viz literatura [5]).

V tomto příkladu vytvoříme minimální aplikaci zobrazující červený kužel osvětlený jedním světlem. Graf scény, zobrazující červený kužel osvětlený jedním světlem, vidíme na obrázku 2.




Obrázek 2: Graf jedoduché scény (převzato, viz kapitola [19])

Kořen grafu tvoří objekt typu SoSeparator. Když se podíváme pod separátor, zjistíme, že má čtyři syny: kamera, světlo, materiál a kužel. Kamera (SoCamera) je nod, který určuje umístění pozorovatele a některé další atributy pohledu do scény. Světlo (SoLight) osvěcuje scénu bílým světlem. Materiál (SoMaterial) udává optické vlastnosti kužele, jednoduše řečeno - udává jeho barvu. Posledním nodem je pak vlastní kužel, což je nod specifikující geometrii tělesa. Výsledkem takového grafu je pak scéna znázorněná na obrázku 3.




Obrázek 3: Jednoduchá scéna (převzato, viz kapitola [19])

## 3.3 Využití technik OpenGL

Procesu vytváření obrazu scény ve framebufferu říkáme rendering. Data scény jsou pak často výsledkem simulace nějakého modelu viz obrázek 1. Aby na scéně bylo vůbec něco vidět, je scéna osvícena. Světlo přidá tělesu lesk a osvítí celou scénu. Je použito materiálové nastavení, to definuje vlastnosti povrchu tělesa. Primitivu, kterou používáme jako hlavní stavební jednotkou tělesa je trojúhelník (viz literatura [4]).

Jak jsem se již dříve zmínil, k těmto možnostem OpenGL, přistupuji skrze knihovnu Open Inventor.

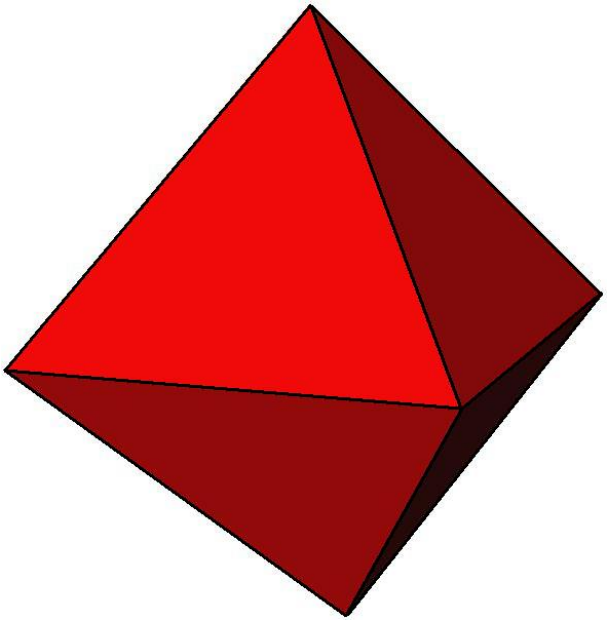
# 4 Vesmírné těleso

## 4.1 Co to je vesmírné těleso

V tomo projektu se pod pojmem vesmírné těleso myslí jakýkoliv organický objekt, který se může ve vesmíru vyskytovat. Takže je to vlastně planeta, mesíc, asteroid nebo nějaký úlomek kamene.

## 4.2 Stavba vesmírného tělesa

Základ tělesa tvoří osm trojúhelníků (viz obrázek 4), které jsou sestaveny do dvou jehlanů bez základen. Spojením techto dvou jehanů vznikne ojekt pripomínající diamant. Na tyto trojúhelníky je pak aplikován algoritmus, který pro každý další vyšší detail rozdělí každý zatím nerozdělený trojúhelník na ctyři nové a je jimi nahrazen. Toto bude podrobně vysvetleno dále viz. obrázek 5.



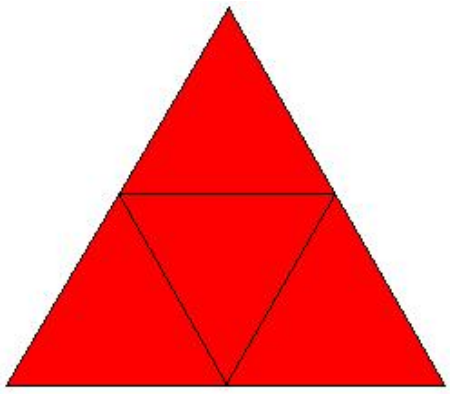
Obrázek 4: Základ vesmírného tělesa

# 5 Algoritmus na dělení trojúhelníků

## 5.1 Způsob rozdělení

Základem tělesa je 8 trojúhelníků, jak už jsme si řekli. Základním stavbním prvkem je tedy trojúhelník. Jedinou cestou pro deailnější zobrazení, je zmenšení základního stavebního prvku. Toho lze docílit jedině tak, že jeden trojúhelník nahradíme několika menšími. Já jsem zvolil, podle mého názoru, ideální a jednoduché řešení. Spojením středů stran, nám vzniknou z jednoho velkeho 4 menší trojúhelníky, jak je vidět na obrázku 5.

K tomuto způsobu zvyšování detailu jsem se nechal inspirovat z algoritmu použitého na sránce http://www.gamedev.net/reference/articles/article2074.asp (viz literatura [6]).



Obrázek 5: Rozdělení trojúhelníku

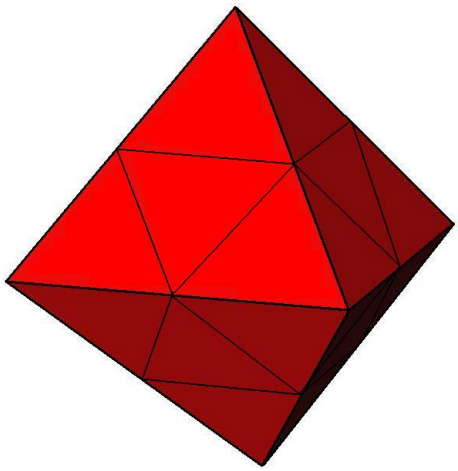
## 5.2 Implementace rozdělení

Souřadnice vrcholů jednotlivých trojúhelníků jsou uloženy v dynamickém poli. V jiném dynamickém poli jsou uloženy trojice čísel. Tyto troice čísel jsou indexy pozic konkrétních bodů. Takže každá troice těchto indexů reprezentuje jeden trojůhelník. Algoritmus si z pole vyjímá tyto troice, spočítá středy stran podle konktétních bodů na které indexy odkazují a z těchto šesti bodů (3 body jsou vrcholy původního trojúhelníku a 3 body jsou spočtené středy stran tohoto trojúhelníku) sestaví 4 nové trojúhelníky. Do pole vrcholů se přidají 3 nově vzniklé body a do nového dynamického pole se ukládají nově vzniklé trojice indexů, které představují odkazy na vrcholy nově vznikých trojúhelníků. Toto je provedeno pro všechny trojice z původního pole indexů a původní pole je tím nově vzniklým posléze nahrazeno.

# 6 Zvýšení detailu

## 6.1 Docílení vyššího detailu

Rozdělením vetšího trojúhelníku na několik menších jsme sice zvýšili počet stavebních jednotek ze kterých se těleso skládá a můžeme docílit toho, aby bylo zobrazeno detailněji. Ale zatím jsme toho nedocílili. V tomto okamžiku máme sice více trojúhelníku, ale těleso vypadá stále naprosto stejně jak před aplikací dělícího algoritmu. Každé 4 nové trojůhelníky totiž tvoří jednu plochu jak je vidět na orázku 6. Je tedy potřeba nově vzniklé vrcholy posunout tak, aby neleželi na hraně původního trojúhelniku. V tomto projektu se posunutí provede po přímce spojující tento bod se středem tělesa smerem ke středu nebo od něj.



Obrázek 6: Základní těleso po dělení trojúhelníku

## 6.2 Kam posunout nový bod?

Způsob, jakým budeme body posouvat, nám určí výsledný vzhled tělesa (viz literatura [6]). Vzdálenost všech vrcholů základního tělesa od středu je stejná. Pokud budeme všechny body posouvat na stejnou vzdálenost od středu tělesa a pokud je tato vzdálenost rovna vzdálenosti vrcholů základního tělesa, pomom výsledným tělesem bude koule. Toto je nejjednodušší případ. Chceme-li vytvořit těleso, které vypadá jako úlomek kamene, nebo jako planeta (koule se zvlneným povrchem), je situace složitejší. Nové body musíme posouvat o pseudonáhodné vzdálenosti.

To znamená, že bude posunut o vzdálenost, která bude náhodně vybrána z určitého intervalu. Pro jednu úroven detailů bude interval stejný.

Čím vyšší uroveň detailů, tím bude interval menší, z toho vyplývá, že pro vetší detail bude dochazet k menším změnám ve tvaru tělesa. A povrch tělesa se bude vyhlazovat. Tím docílíme vizuálně atraktivnějšího vyhlazeného vzhledu tělesa.

## 6.3 Implementace posunu bodu

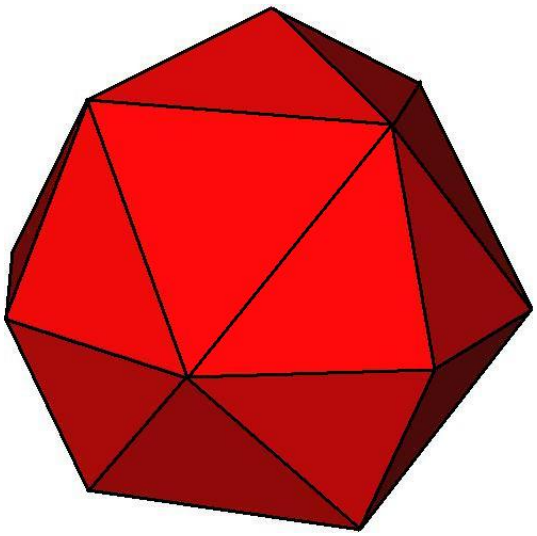
Již při dělení se každý nový bod odsnune od středu na vzdálenost, která je rovna průměrné vzdálenosti od středu bodů, jejichž spojnici jsme rozdelili pro vznik tohoto nového bodu.

Budeme procházet pole, ve kterém jsou uloženy všechny vrcholy. Procházíme jej od místa, kde začínají dosud neposunuté body do konce.

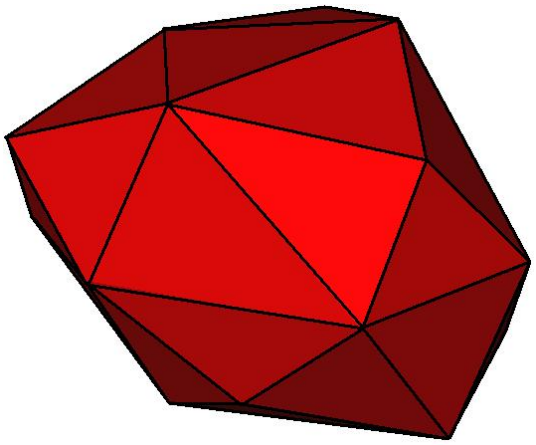
Rozptyl nám udává horní hranici intervalu, ze kterého se bude náhodně vybírat hodnota, o kterou se bod bude posouvat. Spodní hranice se vypocítá z horní hranice tak, aby horní i spodní hranice byla od současné polohy bodu stejně vzdálené, ale každá v opačném směru.

Pokud má být výsledným tělesem koule, tj. pokud je parametr rozptylu roven nule, nebude se bod posouvat vubec. Jelikož body, z jeichž průměrné vzdálenosti se počítá vzdálenost tohoto bodu je stejná, bude nový bod stejně daleko od středu jak tyto 2 body a proto výsledné těleso nemuže být nic jiného než koule. Jak těleso vypádá po prvním dělení, je-li rozptyl roven nule vidíme na obrázku 7.

Pokud je rozptyl větší než nula, vybere se pro kazdý nový bod náhodně hodnota vzdálenosti z intervalu, který se z hodnoty rozptylu vypočítá. A na tuto vzdálenost od středu tělesa je nový bod posunut. Pro každou následujuící úroveň detailu je hodnota rozptylu dělena dvěma z důvodu uvedeného v kapitole 6.2. Jak těleso vypádá po prvním dělení, je-li rozptyl roven 750 vidíme na obrázku 7.



Obrázek 7: Základní těleso po dělení trojúhelníku a odsunutí bodů, je-li rozptyl roven 0



Obrázek 8: Základní těleso po dělení trojúhelníku a odsunutí bodů, je-li rozptyl roven 750

# 7 Úrovně detailů

## 7.1 Co to je úroveň detailů

Výše jsem popsal, jak ze základního tělesa vytváříme další tělesa tak, že dělíme trojúhelníky a posouváme nově vzniklé body. Takže nám vnikně několik těles.

Tato tělesa mají základní tvarové rysy shodné. Každé takto vniklé těleso je jednou z úrovní detailu jednoho jediného tělesa. Takže všechny takto vzniklá tělesa reprezentují jedno těleso. Zobrazeno je vždy pouze jedno z nich, tedy jen jedna z úrovní. Které z nich to bude bude popsáno dále.

## 7.2 Proč používat úrovně detailů

Otázkou zůstává, proč vlastně vytvářet několik těles, když bude zobrazeno pouze jedno z nich. Proč si nevybrat pouze jedno z nich a ostatní nevypustit. Odpověď je jednoduchá.

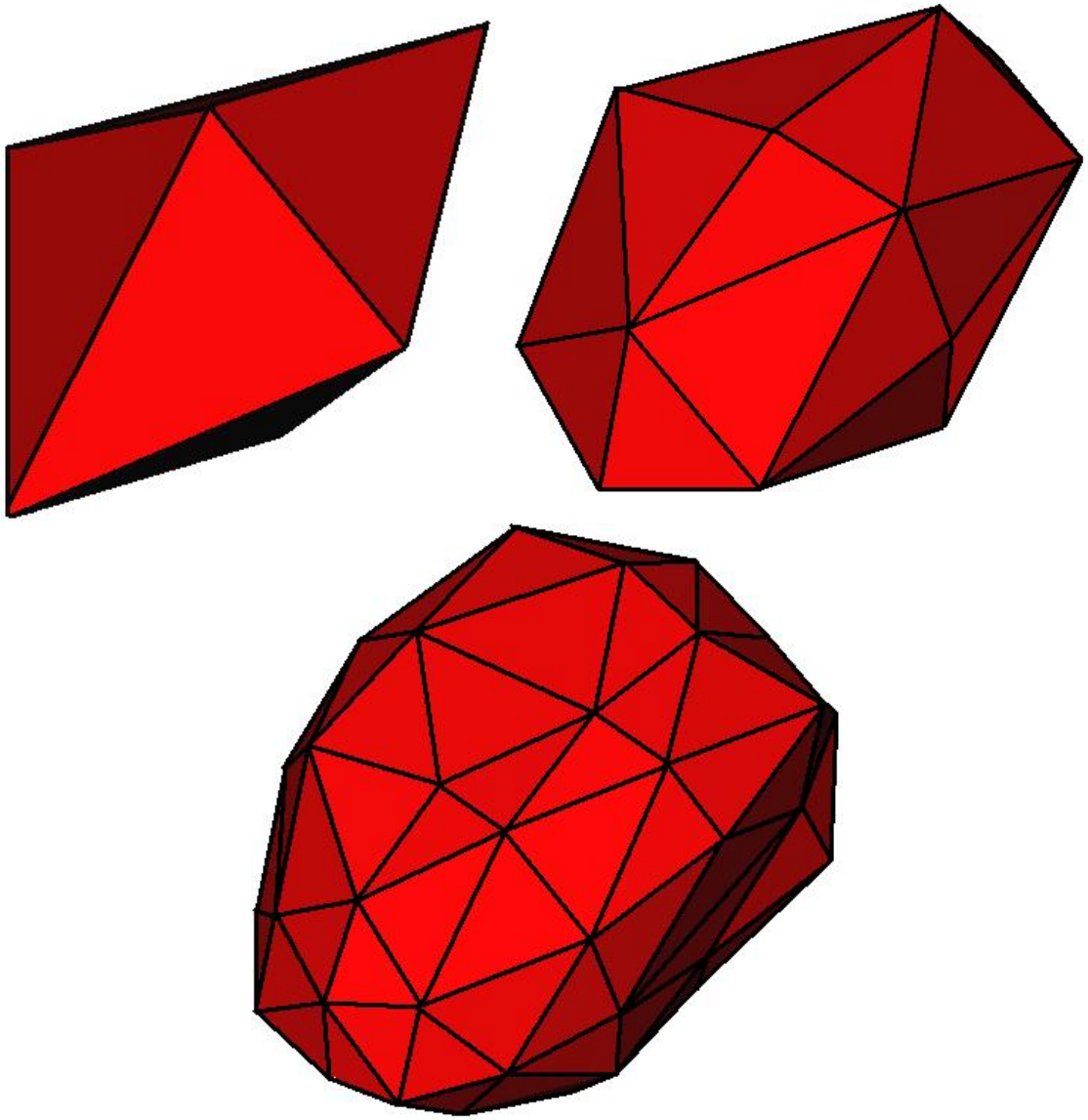
Chceme-li dosáhnout vizualně atraktivního zobrazení tělesa a zároven plynulého chodu aplikace, nelze to dělat jinak než používat více těles a vhodně tato tělesa měnit.

Kdybychom totiž meli pouze těleso s nejvyšším detailem, tj. největším počtem trojúhelníků, tak by aplikace při větším počtu takovýchto těles na scéně rozhodně nebyla plynulá. Pokud bychom meli pouze těleso s nízkým detailem, tak by aplikace byla plynulá i při velkém počtu těchto těles na scéně, ale o vizuální atraktivnosti by nemohla být řeč. Proto je nutný tento kompromis.

Pokud bude těleso hodně vzdálené od kamery, bude zobrazeno těleso s nejmensím detailem (nejmenším počtem trojúhelníku), tomu odpovídá základní těleso. Když se bude ke kameře přibližovat, bude postupně nahrazováno tělesy s detailem vyšším. Tím si zajistíme, že těleso bude vypadat stále atraktivně a zároven se bude zobrazovat pouze tolik trojúhelníků, kolik bude nezbytně nutné k tomu, aby atraktivně vypadalo.

Tímto ovšem vyvstal další problém. Kolik úrovní detailů použít. Idealní je používát co možná nejvetší počet úrovní, aby přepínání mezi úrovněmi probíhalo tak, že ho uživatel ani nezaregistruje. Jenže vygenerování kazdé další úrovně trvá přibližně třikrát tak dlouho jako vygenerování všech předchozích. Takže je opet nutno volit vhodný kompromis mezi dobou trvání generování a vizuálním dojmem. V této práci je tím kompromisem pět úrovní detailů.

Na obrázku 9 jsou ukázány 3 úrovně detailů, vidíme na něm, jak velký přírustek trojůhelníků v každém dalším detailu je.



Obrázek 9: Ukázka tří úrovní detailů

## 7.3 Metody přepínání úrovní detailů

Knihovna Open Inventor ma implementovány dvě třídy pro přepínání mezi tělesy (viz literatura [8]).

První z nich je *SoLOD*. Tato třída přepíná tělesa, na základě jejich vzdálenosti od kamery. Intervaly vzdáleností ve kterých budou jednotlivá tělesa zobrazena se musi ručně nastavit, tak aby to bylo přijatelné.

Toto není pro náš případ příliš vhodné, protože tento způsob je závislý na velikosti okna, ve kterém je scéna zobrazena. Zmeníme-li velikost okna, budou se sice modely porád přepínat ve vzdálenostech, které jsme si nastavili, ale vizuální dojem stejný nebude.

Druhou trídou je *SoLevelOfDetail*. Ta přepína tělesa podle počtu pixelů, které na obrazovce zaujímá pomyslný kvádr, který obaluje těleso. Rovněž zde je potřeba ručně zadat intervaly počtu pixelů, ve kterých budou jednotlivá tělesa zobrazena. Ale je zřejmé, že tento způsob je na velikosti okna ve kterém bude scena zobrazena naprosto nezávislý.

Přestože je tento způsob výpočetně náročnejší než ten předešlý, tak jsem ho použil, protože se pro tento projekt výborně hodí.

# 8 Použití vesmírného tělesa

Když chceme vesmírné těleso použít, musíme si vytvořit instanci třídy *create* a přidat do kořene scény potomka, kterého vrací metoda *createObject*.

Tato metoda má dva parametry. Prvím určíme, jaký poloměr bude výsledné těleso mít. Druhým parametrem nastavíme hodnotu proměnné rozptyl. Pokud bude tento parametr roven nule, bude výsledné těleso koule. Čím větší hodnoty bude nabývat, tím vetší bude členitost výsledného tělesa.

Zdrojový kód, který zajistí zobrazení tělesa červené barvy, které má tvar koule o poloměru 2 by mohl vypadat asi takto:

```
// inicializace okna
#ifdef _WIN32
    HWND window = SoWin::init(argv[0]);
#else
    QWidget window = SoQt::init(argv[0]);
#endif
if (window == NULL) exit(1);

// kořen scény
SoSeparator *root = new SoSeparator;
root->ref();

// materiál
SoMaterial *material = new SoMaterial;
material->diffuseColor.setValue(1.0, 0.0, 0.0);
root->addChild(material);

// vytvoření tělesa
create teleso;
root->addChild(teleso.createObject(2,0));
```

```
// okno prohlížeče
#ifdef _WIN32
    SoWinExaminerViewer viewer = new SoWinExaminerViewer(window);
#else
    SoQtExaminerViewer viewer = new SoQtExaminerViewer(window);
#endif
viewer->setSceneGraph(root);
viewer->setTitle("Deleni trojuhelniku");
viewer->show();

// renderovaci smycka
#ifdef _WIN32
    SoWin::show(window);
    SoWin::mainLoop();
#else
    SoQt::show(window);
    SoQt::mainLoop();
#endif

// uvolnění paměti
delete viewer;
root->unref();
```

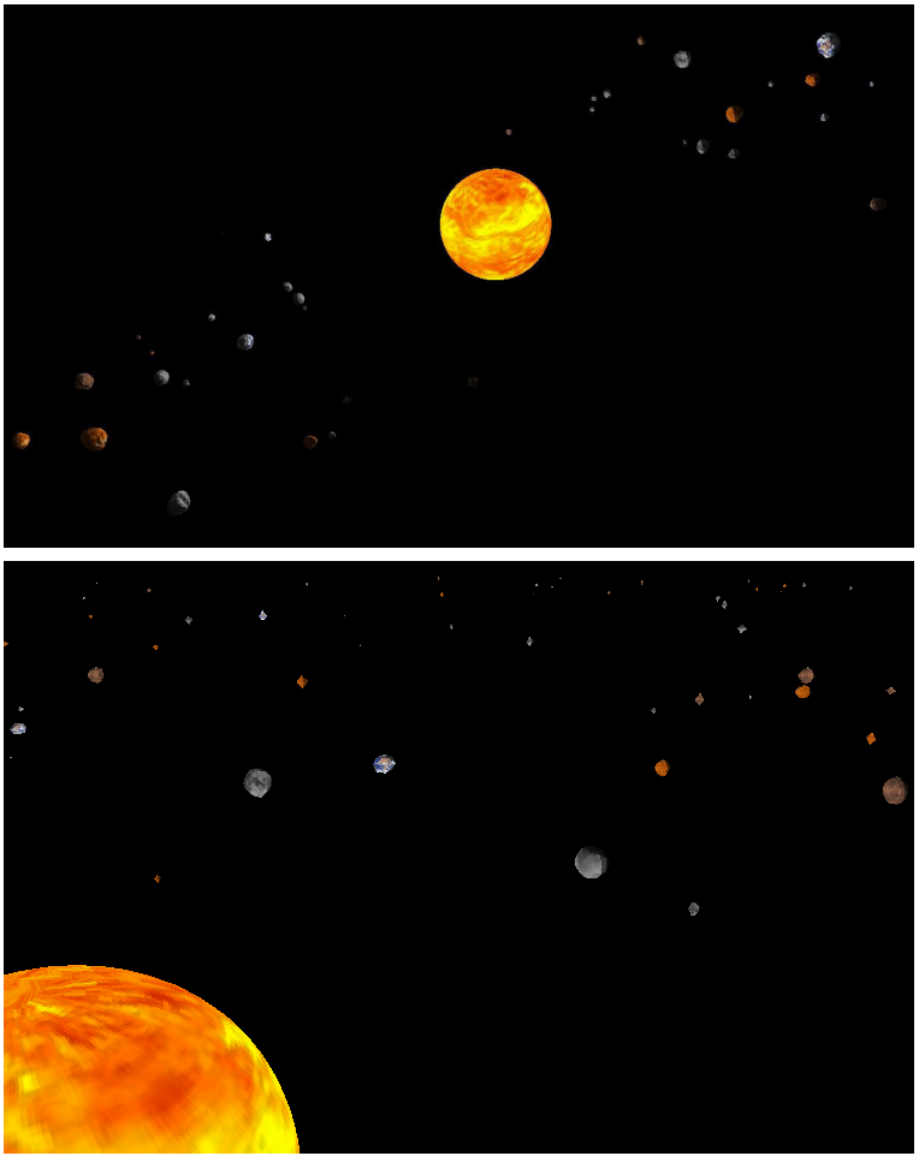
# 9 Scény

## 9.1 Scény pro demonstraci výsledků

V předchozí kapitole bylo ukázáno jakým způsobem se vytvoří jedno těleso. Na scéně, ve které je jedno těleso se však nedá demonstrovat přínos algoritmu, který snižuje detaily. Trojúhelníků je totiž tak nízký počet, že i když detaily nesnižujeme tak se bez problemů stačí vykreslovat. Pro demonstraci přínosu snižování detailů je proto nutné vygenerovat vetší množství těles.K tomuto účelu byly vytvořeny dvě scény. Jedna pro prezentaci vizuálních výsledků této prace viz kapitola 9.1 a druhá pro otestování jejího přínosu z hlediska výkonu viz kapitola 9.2.

## 9.2 Prezentační scéna

První z nich zajišťuje metoda *presentation* třídy *generate*, která má 2 parametry. První parametr je boolovskeho typu a určuje, zda se má nebo nemá používat algoritmus pro snižování detailů. Druhým parametrem je celé číslo, které určí počet vygenerovaných těles. Výsledná tělesa budou náhodné velikosti a budou mít náhodnou členitost povrchu aby scéna nepúsobila jednotvárně. Nagenerují se na oběžné dráhy kolem hvězdy, která tvoří střed prezentační scény. Uprostřed scény je dále umístěn světelný zdroj, který osvětluje okolní tělesa, což vbuzuje dojem, že je světlo vyzařováno z hvězdy. Obrázky č. 10 a 11 ukazují jak tato scéna může vypadat.

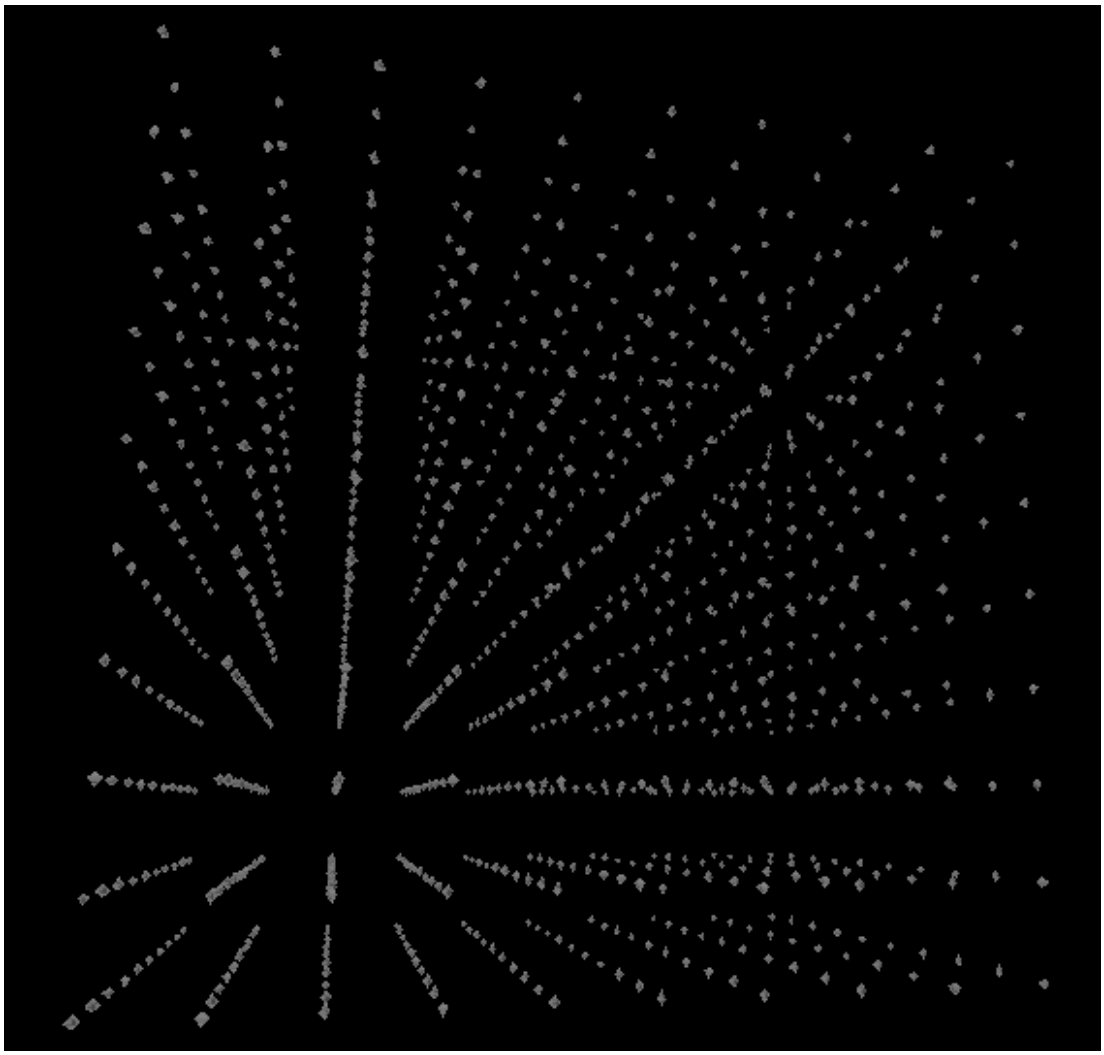


Obrázek 10: Ukázky prezentačních scén

## 9.3 Testovací scéna

Na prezentační scéně se nedá objektivně porovnávat výkonový přínos snižování detailů, jelikož se tělesa generují na náhodné pozice. Proto práce obsahuje možnost generovat ještě jednu scénu, která se bude používat pro vyhodnocení výsledků práce. V této scéně je možno generovat libovolný počet těles do libovolně veliké krychle. Počet těles ve scéně je definován na základě zadaného počtu těles na hraně krychle. Celkový počet těles ve scéně pak bude samozřejmně třetí mocnina tohoto počtu. Velikost krychle je definována zadanou vzdáleností středů dvou sousedních těles. Jak taková scéna může vypadat znázorňuje orázek 13.

Tato scená nemá z hlediska akademického projektu SpaceGame význam a proto je implementována pouze v *main* metodě této práce.



Obrázek 11: Ukázka testovací scény

# 10 Generování do souboru

U rosáhlých scén se může generování protáhnou na nepříjemně velkou dobu i pokud nechceme použit algoritmu na snižování detailů. Pokud algoritmu použito nebude, stejně se musí projít pomocí algoritmu na dělení trojůhelníků pres všechny úrovně detailů. Přestože se nakonec bude používat pouze ta nejvyšší z nich. To vyplývá z principu tvorby vesmírného tělesa. Kdyz se zamyslíme kolik trojúhelníku se musí rozdělit a kolik se z tohoto důvodu musí přepočítat souřadnic není divu že se tento výpočet může pro velký počet těles protáhnout. Doba generování se při mnou použitém způsobu již nijak zvatelně snížit nedá. Proto byla implementována možnost uložení vygenerované scény do souboru s možností jejího zpětného načtení.

Načítání scény ze souboru se bohužel nedá s možnostmi, které přináší její generování srovnat. Když scénu generujeme, můžeme si zvolit její parametry. Tedy kolik bude obsahovat těles, jestli bude použit algoritmus na snižování detailů či nikoliv. Tělesa budou dále vygenerovaná na náhodných pozicích a budou mít náhodnou velikost a tvar. Pokud se scéna bude načítat ze souboru, načte se tak jak do něj byla uložena. Takže tělea budou vždy stejné velikosti, budou mít stejný tvar budou stále na stejných pozicích a bude jich stejný počet. Zda bude nebo nebude používán algoritmus na snižování detailů rozhodne skutečnost, zda byl nebo nebyl tento algoritmus použit pri uložení scény do souboru.

Chceme-li mít tedy scénu rychle načtenou je nutné zvolit tento kompromis. Velikost generovaných souborů může dosáhnout i desítek mega bytů. Což není nic podivuhodného vzhledem k obrovským počtům trojůhelníků, které mohou rozsáhlé scény obsahovat.

# 11 Pohyb ve scéně

## 11.1 Důležitost pohybu

Další nezbytnou složkou pro prezentaci algoritmu na snižování detailů je pohyb kamery po vytvořené scéně. Na statické nehybné sceně se dá měřit počet snímků které se stačí vyrendrovat za vteřinu i počet momentálně zobrazených trojúhelníku, ale není možné na takové scéně ukázat, že tělesa mění detail. Tedy vypadají zblízka stejně jako kdyby se detaily neměnily.

## 11.2 Způsob ovladání pohybu

Na pohyb nejsou zadáním kladeny žádné speciální požadavky, a tak jsem ho vytvoři tak aby bylo co nejjednodušší a nejpohodlnější. Veškerý pohyb je ovládán myší. Pohybem myši se se nastavuje směr pohledu kamery. Levým tlačítkem se zvyšuje rychlost pohybu vpřed, tedy tam kam směřuje pohled. Pravým tlačítkem se rychlost pohybu dopředu snižuje až na hranici nulové rychlosti, pak se začne zvyšovat rychlost vzad.

## 11.3 Přírustek dráhy

Přírustek dráhy je zpracován tak, aby závisel nejen na okamžité rychlosti, ale i na době mezi zobrazením dvou snímků. Tím je dosaženo konstantího přírustku dráhy na různě rychlých počítačích samozřejmně při stejných rychlostech. To znamená, že kamera se bude při určité rychlosti pohybovat stále stejně rychle nezávisle na době mezi snímky.

Pokud by závislost pohybu na době mezi snímky implementována nebyla, tak by se kamera při vysokém počtu snímku za vteřinu pohybovala rychleji, než při malém počtu snímků. Přírustek dráhy se k okamžité hodnotě totiž přičítá v době mezi zobrazením dvou snímků.

# 12 Detekce kolizí

## 12.1 Mezí čím se kolize detekují

Detekce kolizí, tedy rozpoznání stavu kdy do sebe dvě tělesna proniknou těsně před tím, než k tomu dojde není nutná pro prezentaci výsledků této práce, ale kladně přispívá k výslednému dojmu. Abychom měli jistotu, že ke kolizím (vnikání těles do sebe) nedojde je nutné detekovat kolizi každého pohybujícího se předmetu se všemi ostatními. Dále je nutné detekovat kolizi každého nově vytvářeného objektu se všemi objekty ve scéně se vyskytujícími. S tím souvisí další problém. Pokud nově vytvářený objekt koliduje s jiným, je nutno nový objekt umístit jinam, což přináší další další detekci kolize se všemi objekty a riziko, že ke kolizi dojde znovu. Pokud by přídávání těles do scény probíhalo tímto způsobem, mohla by tvorba scény trvat velice dlouho.

## 12.2 Snížení počtu detekcí

Z důvodů uvedených v predchozm odstavci jsem se rohodl generovat tělesa v prezentační scéně na oběžné dráhy tak, aby ke kolizim dojít nemohlo. Tím je zjištěna maximální rychlost sestavení scény z vygenerovaných těles. Má to ještě jednu velikou výhodu a tou je snížení výpočetní náročnosti detekce kolizí. Kdyby bylo ve scéně například sto těles pohybujících se po oběžných drahách kolem středu scény tak by se mezi každým snímkem musela provést detekce kolízí všech těles se všemi. Pro těchto sto těles by se muselo provést 4950 detekcí (součet aritmetické posloupnosti 99-1). Zvoleným řesením tyto detekce odpadají, takže jedinými detekcemi jsou detekce kolízí kamery s ostaními tělesy.

## 12.3 Způsob detekce a reakce na kolize

Kolize jsou detekovány nejjednodušším možným způsobem. Jediným pohybujícím se objektem v této práci je kamera. Takže veškeré detekce se dějí mezi kamerou a ostatními objekty. Kolize je zjištěna pokud dojde k vzájemnému průniku pomyslných koulí, které obalují tělesa u nichž kolizi zjišťujeme. Což znamené že pokud se k sobě dvě tělesa přiblíží na vzdálenost menší, než je součet polomerů koulí, které tyto tělesa obalují je detekována kolize. Pokud k tomu dojde je pohybující se těleso zastaveno. Tato detekce se provádí mezi zobrazením dvou snímků v okamžiku, kdy se k poloze pohybujícího se objektu přičte přírustek dráhy. V nové poleoze se otestuje vzdálenost kamery od všech ostatních těles. Pokud je s některým z nich detekována kolize, sníží se rychlost pohybujícího se tělesa na nulu a tím je zastaveno.

## 12.4 Kde jsou detekce implementovány

Možnost detekovat kolize je implementována pouze v prezentační scéně. V testovací scéně (telesa vygenerována do krychle) je možnost detekovat kolize bezpředmětná. tato scéna je vytvořena pro změření výkonosti a přínosu algoritmu na snižování detailu a detekce kolizí by naměřené výsledky zkreslovala.

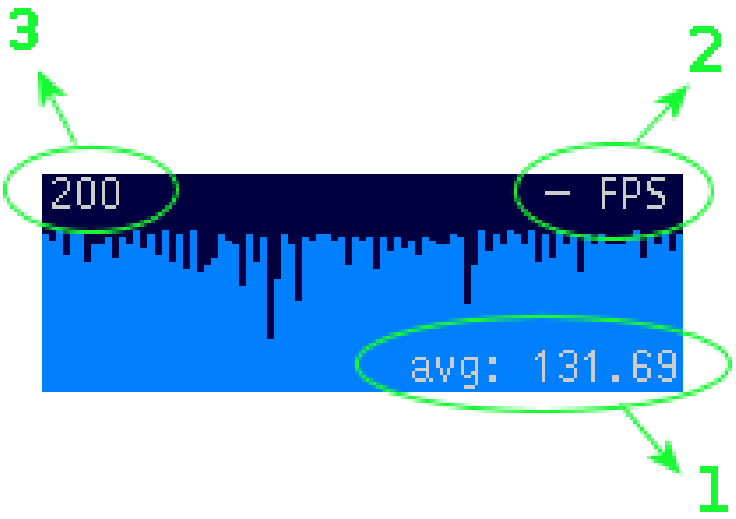
# 13 Komponenty zobrazující měřené hodnoty

## 13.1 Důvod použití komponent zobrazujících měřené hodnoty

Abychom mohli zjistit přínos použití algoritmu na snižování detailů je potřeba porovnat jaké parametry vykazuje referenční scéna s použitím algoritmu a bez jeho použití. Jak bude tato referenční scená vypadat a jak je definována bylo popsano v kapitole 9.3. Pro provnání je potřeba zmeřit jaké parametry tato referenční scéna za různých okoloností vykazuje. Okolnostmi mám na mysli počet těles ve scéně, použití nebo nepoužití algoritmu pro snižování detailů a vzájemné vzdálenosti těles. Za parametry považuji počet viditelných trojúhelníků ve scéně a počet snímků, který se stihne rendrovat za vteřinu (*FPS*), popř. doba mezi zobrazením dvou snímků.

## 13.2 Popis komponenty zobrazující měřené hodnoty

Nejpohodlnější způsob jak tyto údaje zjišťovat je odečítat jejich okamžité hodnoty přímo z obrazovky. A práve k tomuto účelu byla vedoucím mojí práce, ing. Pečivou vytvořena třída *SoPerfGraph*, kterou v této práci využívám. Tato třída zobrazuje aritmetický průměr z hodnot, které se jí předali k zobrazení za poslední vteřinu behu. Dále je zobrazen interval, ve kterém je graficky znázorněna historie třídě předávaných hodnot, na obrázku 12 je tato hodnota označena číslicí 1. Graf si sám mění měřítko podle hodnot, které aktuálně zobrazuje. Tuto hodnotu označuje na obrázku 12 číslice 3.




Obrázek 12: Detail komponenty zobrazující měřenou hodnotu

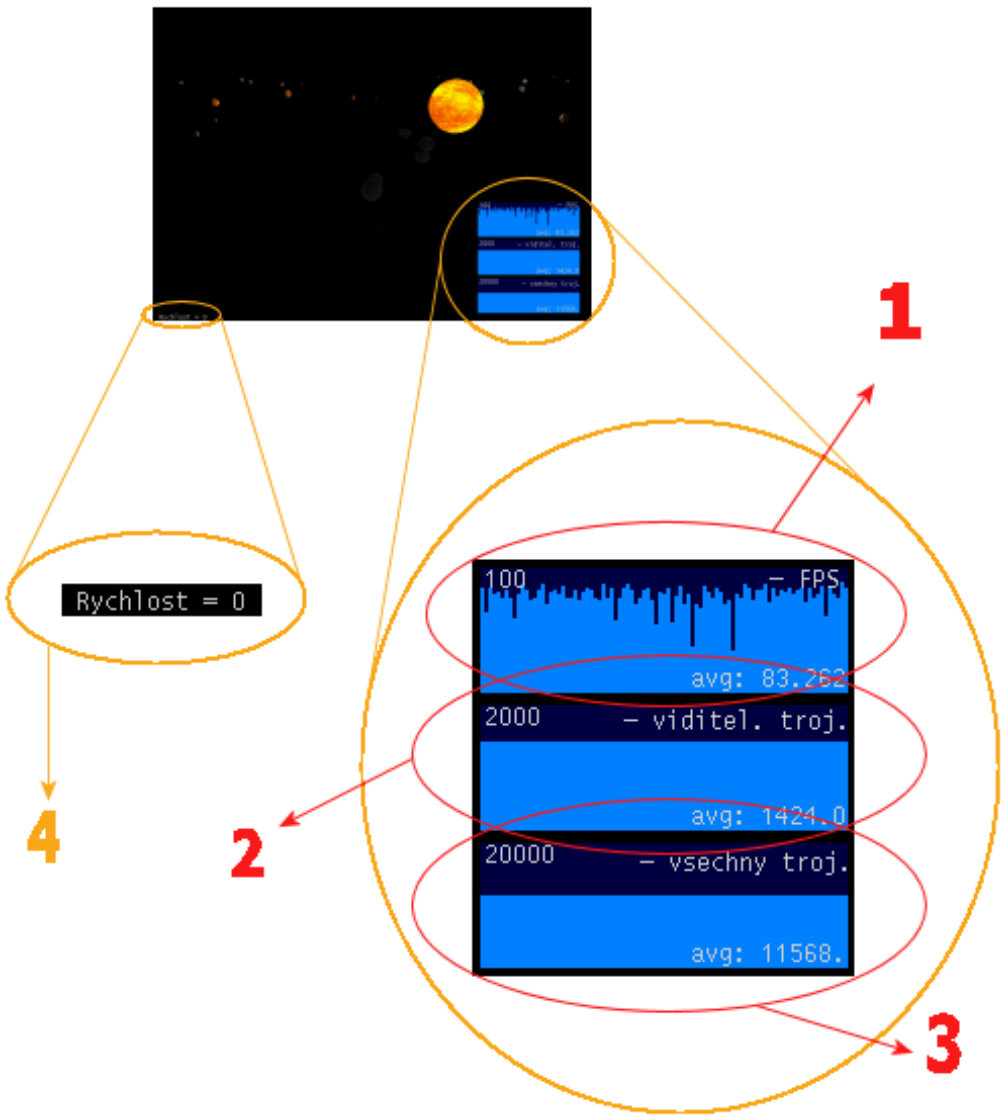
## 13.3 Použití komponent zobrazujících měřené hodnoty

V této práci jsou použity tři tyto komponenty, tedy tři instance třídy *SoPerfGraph*. První z nich je použita pro zobrazování počtu snímků vyrendrovaných za vteřinu (*FPS*), tomu odpovídá graf na orázku 13 označený číslicí 1. Druhá instance se používá pro zobrazení počtu trojúhelníků, které jsou právě ve scéně zobrazeny, odpovídá jí graf na obrázku 13 označen číslicí 2. Poslední je použita pro zobrazení všech trojúhelníků vyskytujících se ve scéně, tedy i těch které jsou mimo pohled kamery (vedle ní nebo za ní). Této odpovídá graf na obrázku 13 označený číslicí 3.

## 13.4 Popisky grafů

Pro vytváření textových popisů ve scéně slouží další třída vytvořená ing. Pečivou *SoScreenPositioner*. V práci jsou použity čtyři instance této třídy. Tři jsou použity pro popis grafů zobrazujících měřené hodnoty. Popiska je v každém grafu umístěna v pravém horním rohu, na obrázku 12 je označena číslicí 2.

Dále jsem jednu z instancí třídy *SoScreenPositioner* použil pro zobrazení informace o aktuální rychlosti kamery. Tato popiska je umístěna v levém dolním rohu scény. Na obrázku 13 je její umístění označeno číslicí 4.



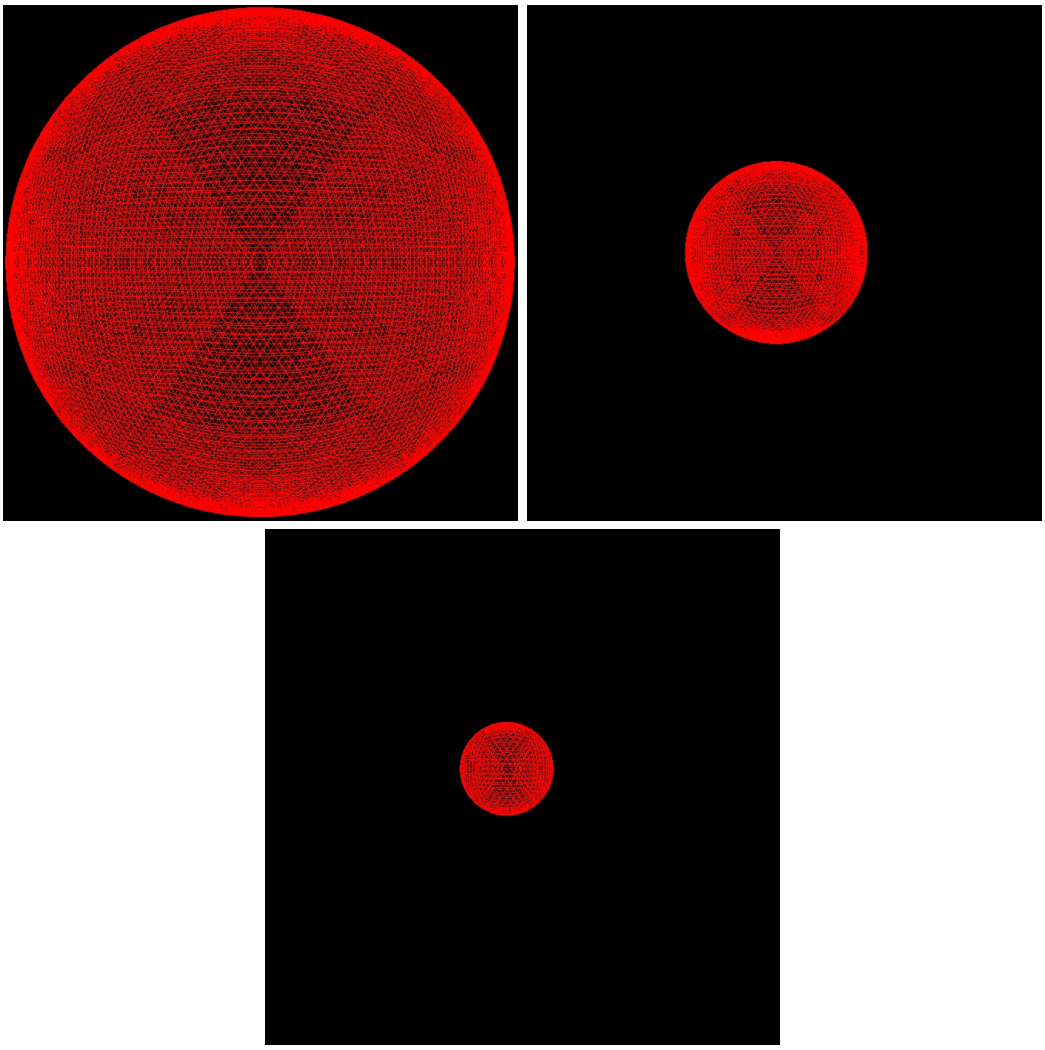

Obrázek 13: Umístění komponent

# 14 Vyhodnocení výsledků

## 14.1 Změny detailů jednoho tělesa

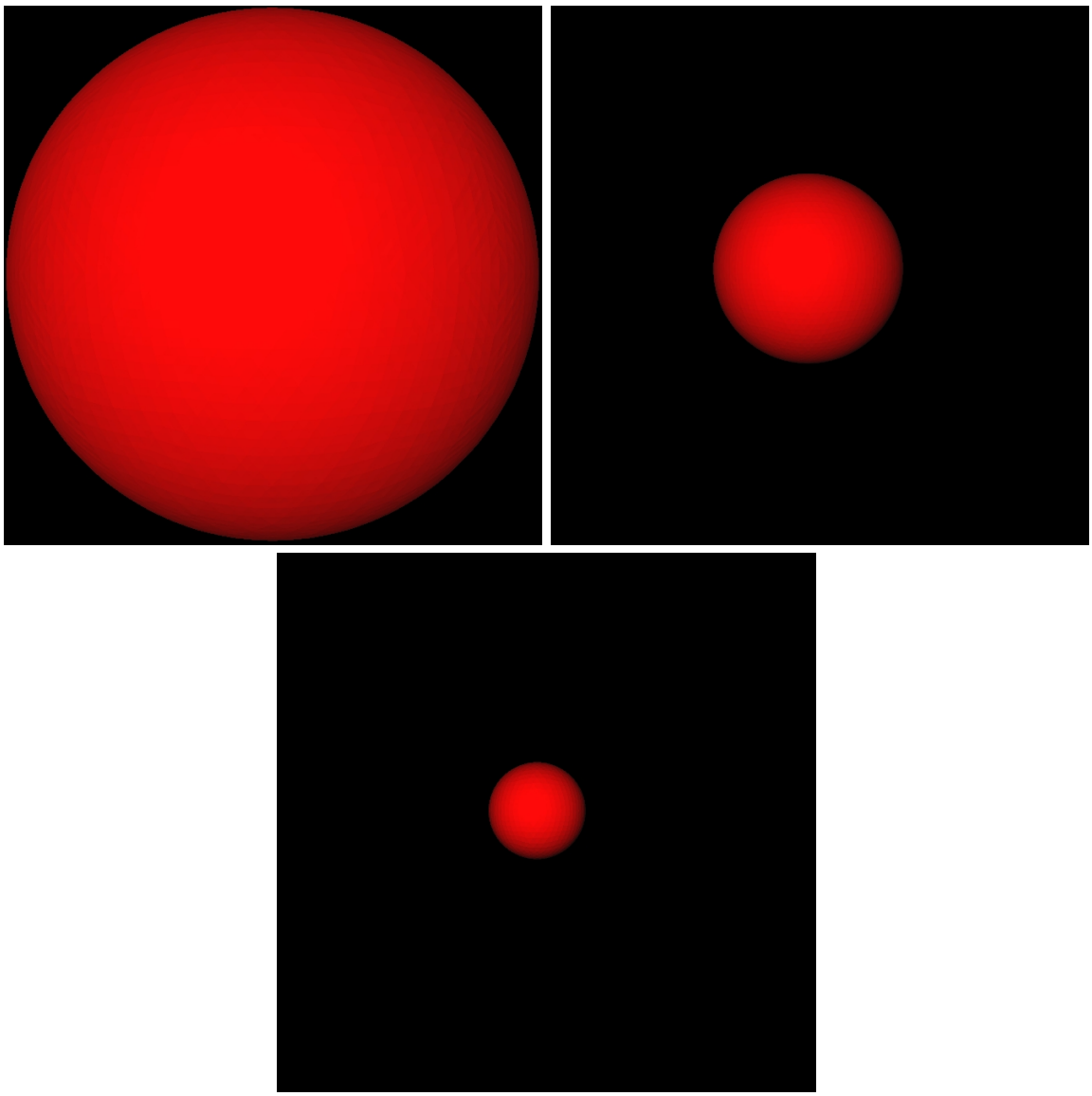
Jako první výsledek této práce ukážu na obrázcích funkčnost algoritmu, který mění detaily. Na obrázcích 14 a 15 je zobrazeno těleso s rozptylem rovným nule, tedy koule, vzdalující se od kamery.

Na obrázku 14 je zobrazena jako drátěný model a je vidět že hustota drátěné sítě je ve všech třech vzdálenostech přibližně stejná. Z toho vyplývá, že se počet trojúhelníků nepochybně s rostoucí vzdáleností snižuje.



Obrázek 14: Drátěné modely tělesa, které se vzdaluje od kamery

Na obrázku 15 je tatáž koule zobrazena už jako běžný model. A je na ní vidět, že prestože se při vzdalování od kamery snižuje počet trojúhelníků, těleso po vizuální stráce vypadá stále stejně.



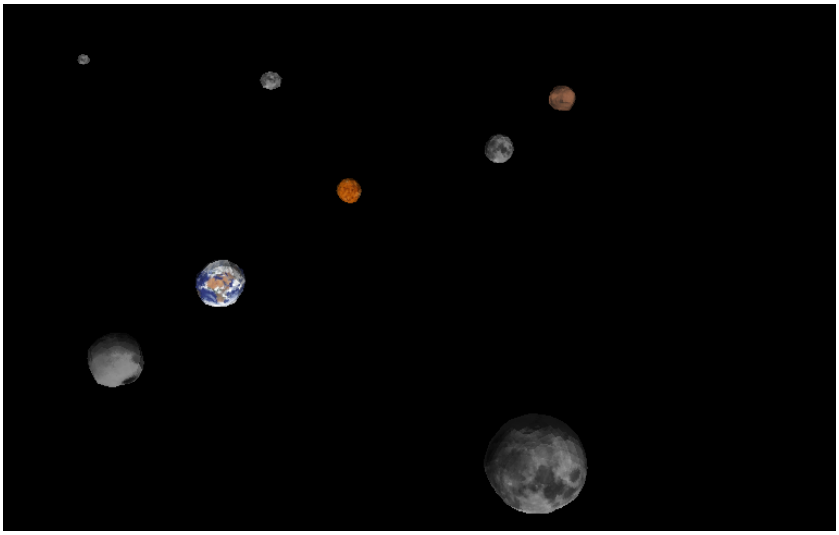
Obrázek 15: Modely tělesa, které se vzdaluje od kamery

## 14.2 Změny detailů ve scénách

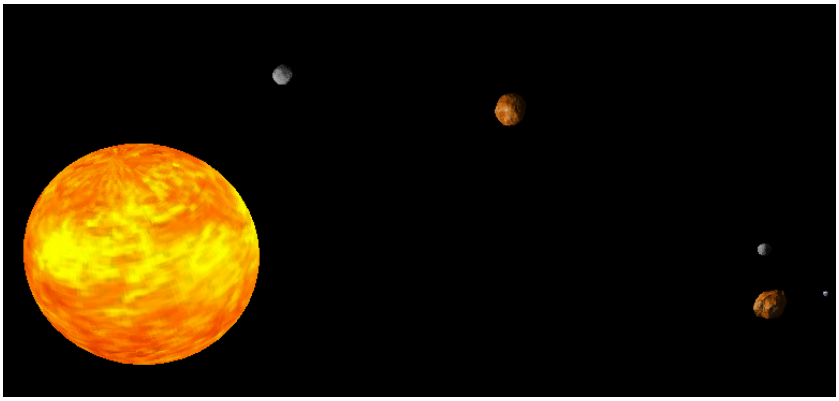
V této kapitole bude na obrázcích 16 - 19 provedeno porovnání scén z vizuální stránky při použití algoritmu na snižování detailů a bez jeho použití.

### 14.2.1 Prezentační scény

Na obrázku 16 je pohled do prezentační scény v níž je algoritmus na změnu detailů použit. Na obrázku 17 je rovněž pohled do prezentační scény, ve které tento algoritmus použit není a podle mého názoru není patrný vizuální rozdíl mezi tělesy z obou scén přestože počet trojúhelníku, ze kterých se jednotlivá tělesa skládají se rapidně liší.



Obrázek 16: Ukázka prezentační scény s použitím algoritmu na dělení troúhelníků

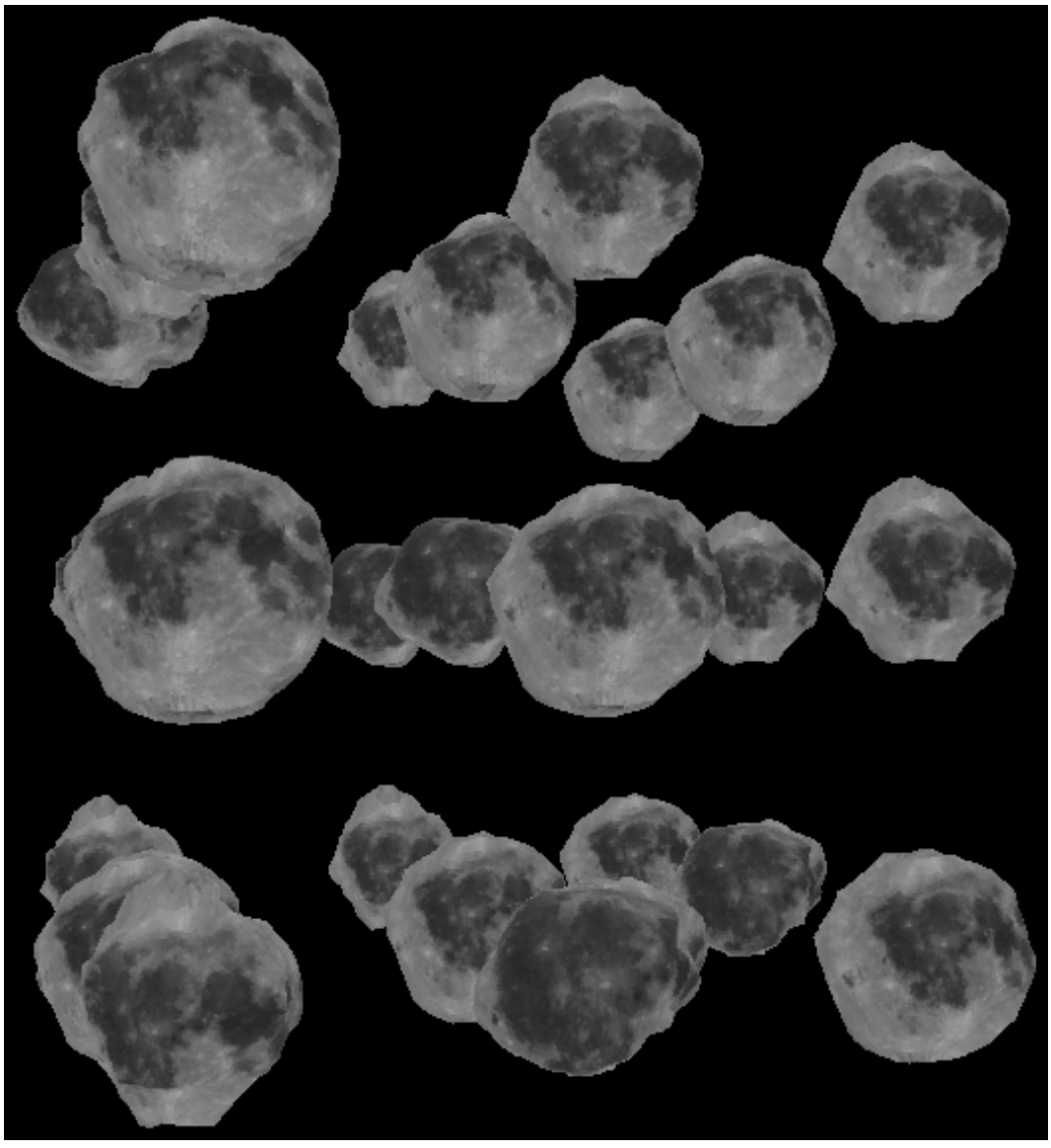


Obrázek 17: Ukázka prezentační scény bez použití algoritmu na dělení troúhelníků

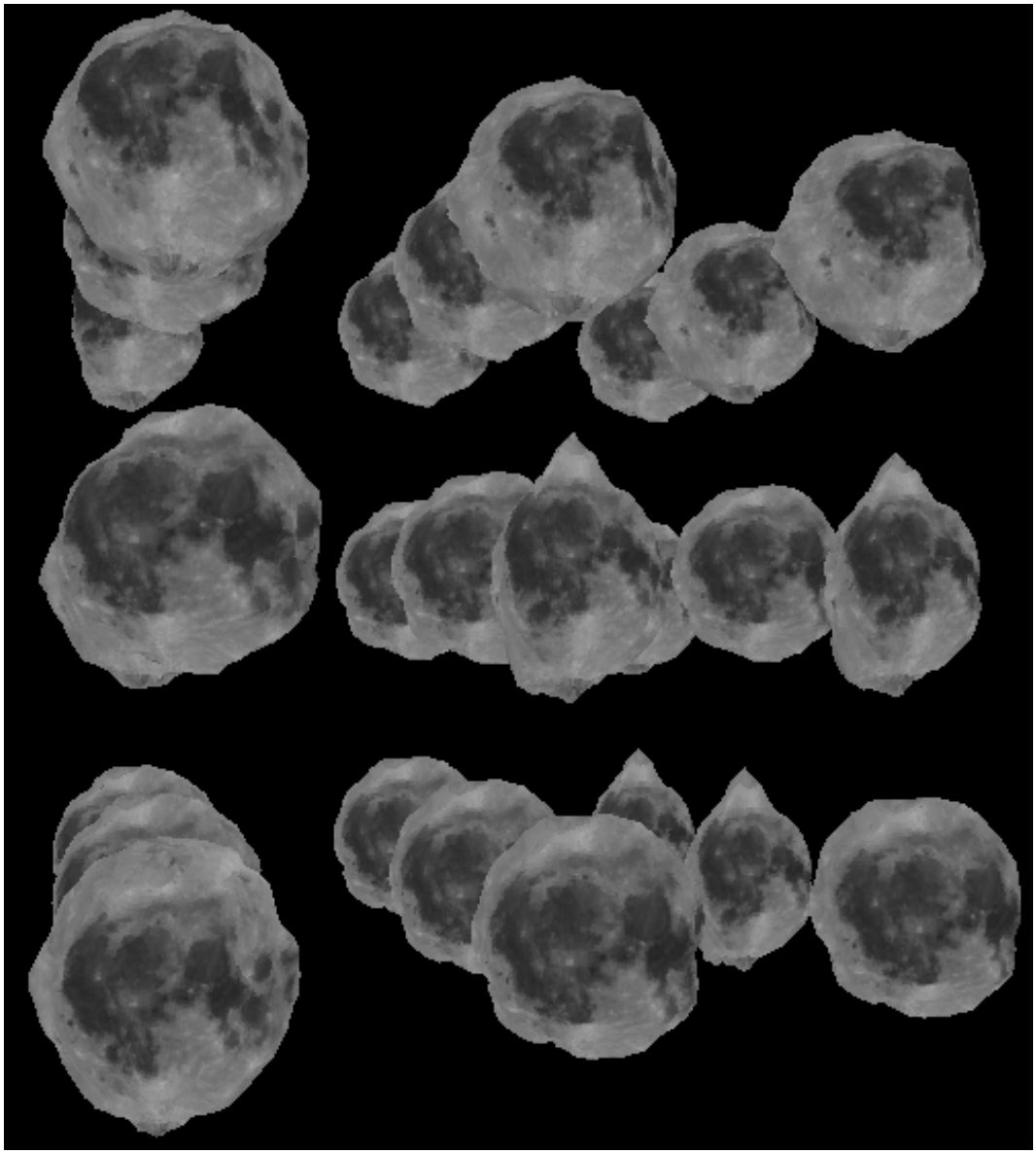
### 14.2.2 Testovací scény

Obrázky 18 a 19 poskytnou ještě objektivnější porovnání vzhledu těles pri použití algoritmu na snižování detailů a bez jeho použití než obrázky 16 a 17, protože pomocí testovací scény je možno porovnávat vzheld na totožných scénách. Obe tyto scény obsahují 27 těles (3 tělesa na hraně) a vzdálenost těles na hraně je 200 bodů.

Na obrázku 18 je ve které je použit algoritmus na snižování detailů. Počet trojúhelníku v této scéně je 24 576. Na obrázku 19 tento algoritmus použit není a scéna je složena z 55 296. Přestoze obsahuje téměř dvojnásobek trojúhelníků než scéna na obrázku 18, je kvalita zobrazení na stejné úrovni.



Obrázek 18: Ukázka testovací scény s použitím algoritmu na dělení troúhelníků



Obrázek 19: Ukázka testovací scény bez použití algoritmu na dělení troúhelníků

## 14.3 Vyhodnocení naměřených hodnot

Veškeré hodnoty uvedené v této kapitole byly naměřeny, kvůli objektivitě měření, na testovací scéně která byla k tomuto účelu vytvořena. Veškerá měření byla provedena na jednom počítači s konfigurací: Procesor - Athlon 1700XP, velikost operační paměti - 512MB, grafická karta - Titanium 4200, AGP 8x, 128MB.

### 14.3.1 Měření nad jedním tělesem

Jako lehký úvod do této kapitoly zaměřené na vyhodnocování naměřených hodnot zařazuji porovnání hodnot nameřených ve scéně s jedním tělesem. Hodnoty v tabulce 1 odpovídají měření při použití algoritmu na snižování detailů. Hodnoty naměřené ve stejné scéně bez použití algoritmu na snižování detailů jsou obsaženy v tabulce 2. Jedná se pouze o jeden řádek, protože bez použití algoritmu vrací scéna stejné výsledky bez ohledu na vzdálenost kamery od tělesa.

| Úroveň detailu | Počet trojúhelníků | FPS | Doba mezi snímky [s] |
|---|---|---|---|
| 1 | 8 | 186 | 0,005376344 |
| 2 | 32 | 180 | 0,005555556 |
| 3 | 128 | 170 | 0,005882353 |
| 4 | 512 | 160 | 0,006250000 |
| 5 | 2048 | 145 | 0,006896552 |

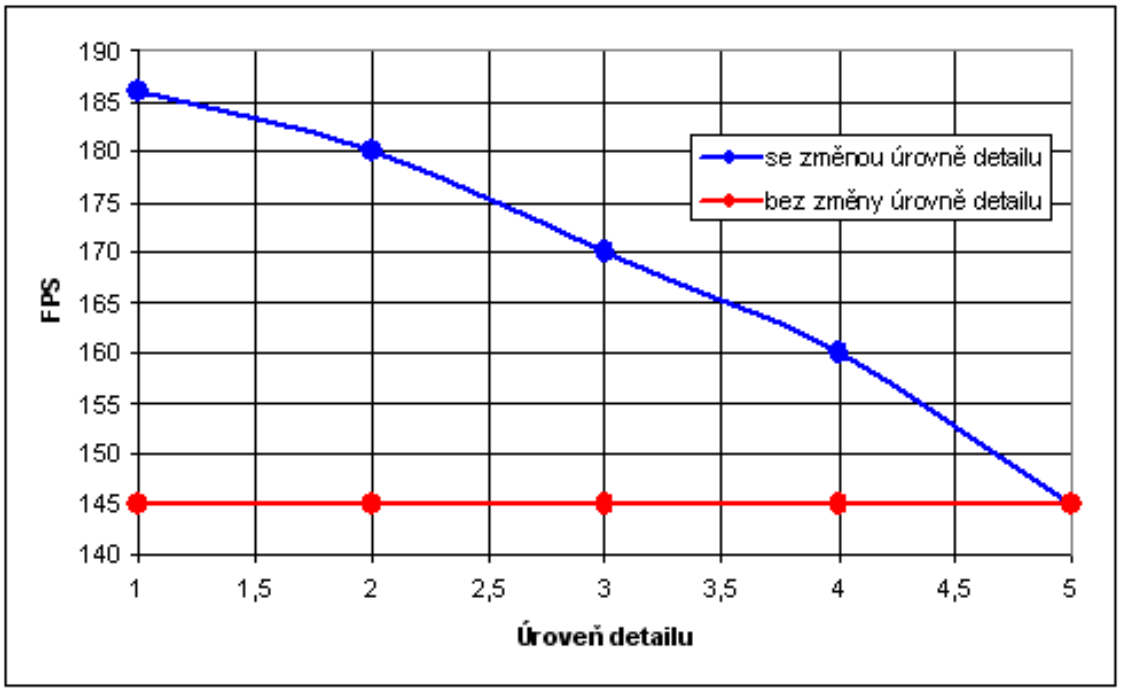
Tabulka 1: Jedno těleso s použitím algoritmu na snižování detailů

| Počet trojúhelníků | FPS | Doba mezi snímky [s] |
|---|---|---|
| 2048 | 145 | 0,006896552 |

Tabulka 2: Jedno těleso bez použití algoritmu na snižování detailů

Hodnoty z tabulky 1 a tabulky 2 jsou zakresleny v grafu na obrázku 20. U hodnot v tabulce 2 sice ke změnám úrovní detailu nedochází, ale jelikož se dají úrovně detailu považovat za vzdálenost od kamery, tak je možné tyto 2 křivky porovnávat v jedné tabulce.

I kdyz se jedná o scénu, ve které je pouze jedno těleso, tak se od sebe tyto dve scény výkonově značně liší. S pribývajícím počtem těles se budou odlišnosti dále prohlubovat.




Obrázek 20: Závislost FPS na úrovni detailu pro jedno těleso

### 14.3.2 Měření nad scénami s konstantní velikostí testovací krychle

Scénami s konstantí velikostí testovací krychle jsou myšleny testovací scény, ve kterych se mění počet těles a tato tělesa jsou rozložena v kostce o konstatním objemu. Z toho vyplývá, že se mení vzájemné vzdálenosti těles od sebe. Pro malý počet těles ve scéně budou vzájemné vzdálenosti těles velké, pro zvyšující se počet těles se bude vzájemná vzdálenost snižovat. Vzdálenost kamery od středu scény byla pro všechna měření konstantní a byla nastavena tak, aby byla v pohledu kamery vidět krychle celá.

V tabulce 3 jsou hodnoty naměřené ve scéně s použitím algoritmu na snižování detailů, v tabulce 4 jsou hodnoty naměřené bez jeho použití. V obou tabulkách jsou hodnoty měřené na krychli s hranou 10 000 bodů.

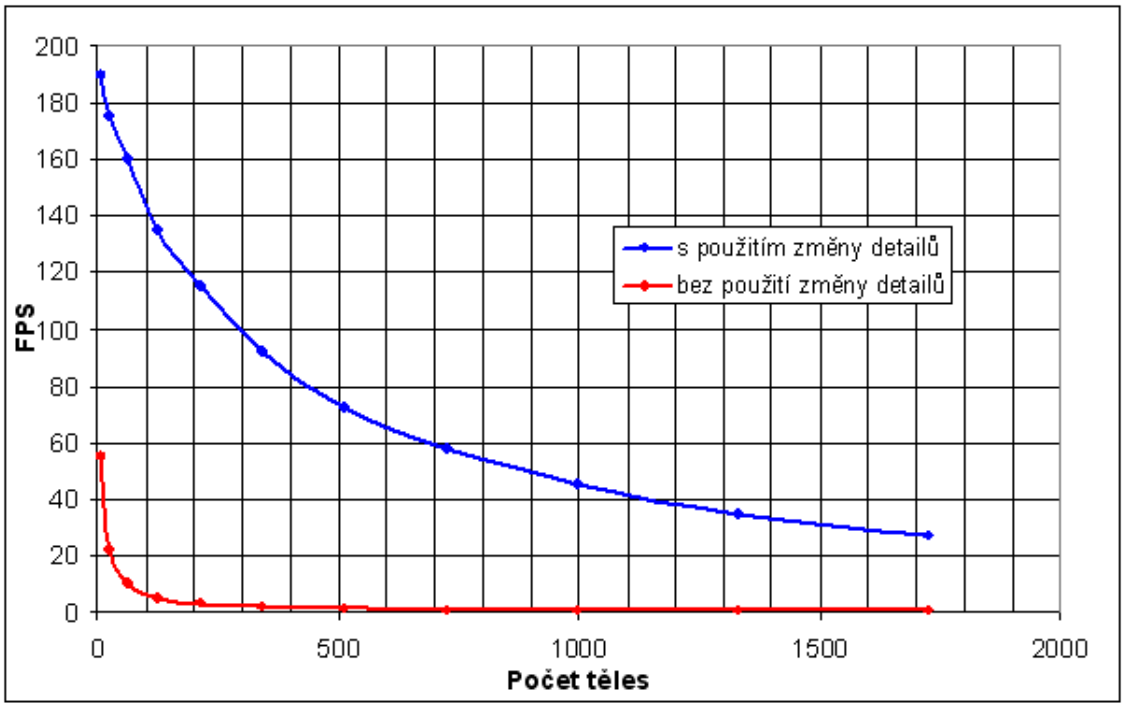
| Počet těles na hraně | Počet těles | Počet trojúhelníků | FPS | Doba mezi snímky [s] | Vzdálenost na hraně [body] |
|---|---|---|---|---|---|
| 2 | 8 | 64 | 190 | 0,005263158 | 10000 |
| 3 | 27 | 216 | 175 | 0,005714286 | 5000 |
| 4 | 64 | 512 | 160 | 0,006250000 | 3333 |
| 5 | 125 | 1 000 | 135 | 0,007407407 | 2500 |
| 6 | 216 | 1728 | 115 | 0,008695652 | 2000 |
| 7 | 343 | 2844 | 92 | 0,010869565 | 1667 |
| 8 | 512 | 4096 | 72 | 0,013888889 | 1429 |
| 9 | 729 | 5832 | 58 | 0,017241379 | 1250 |
| 10 | 1000 | 8000 | 45 | 0,022222222 | 1111 |
| 11 | 1331 | 10648 | 35 | 0,028571429 | 1000 |
| 12 | 1728 | 13824 | 27 | 0,037037037 | 833 |

Tabulka 3: Scéna s konstatní velikost testovací krychle s použitím algoritmu na změnu detailů

| Počet těles na hraně | Počet těles | Počet trojúhelníků | FPS | Doba mezi snímky [s] | Vzdálenost na hraně [body] |
|---|---|---|---|---|---|
| 2 | 8 | 16284 | 55 | 0,018181818 | 10000 |
| 3 | 27 | 55296 | 22 | 0,045454545 | 5000 |
| 4 | 64 | 131072 | 10 | 0,100000000 | 3333 |
| 5 | 125 | 256000 | 5 | 0,200000000 | 2500 |
| 6 | 216 | 442368 | 3 | 0,333333333 | 2000 |
| 7 | 343 | 702464 | 2 | 0,500000000 | 1667 |
| 8 | 512 | 1048576 | 1,4 | 0,714285714 | 1429 |
| 9 | 729 | 1492992 | 0,9 | 1,111111111 | 1250 |
| 10 | 1000 | 2048000 | 0,7 | 1,428571429 | 1111 |
| 11 | 1331 | 2725888 | 0,5 | 2,000000000 | 1000 |
| 12 | 1728 | 3538944 | 0,4 | 2,500000000 | 833 |

Tabulka 4: Scéna s konstatní velikost testovací krychle bez použití algoritmu na změnu detailů

Závislosti počtu snímků za vteřinu na počtu těles ve scéně z hodnot uvedených v tabulce 3 a tabulce 4 jsou zakresleny v grafu na obrázku 21. Už z těchto dvou křivek se dá udělat obrázek o tom, jaký přínos má algoritmus pro snižování detailů. Obě křivky jsou exponenciální, ale ta která odpovídá datům naměřeným ve sceně bez použití tohoto algoritmu klesá mnohem strměji a už pro malý počet těles ve scéně se dostává do nepoužitelných hodnot FPS, což je 25 a méně. Zatímco když algoritmus použit byl, drží se křivka i pro vysoký počet těles v použitelné oblasti hodnot FPS.




Obrázek 21: Závislost FPS na počtu těles ve sceně s konstantní velikostí testovací krychle

### 14.3.3 Měření nad scénami s konstantní vzájemnou vzdáleností těles

V následujících tabulkách jsou zaznamenána meření která jsou provedena na scénach ve kterých se měnil počet těles a jejich vzájemná vzdálenost zůstávala stejná. Provedl jsem čtyři měření pro čtyři hodnoty vzájemných vzdáleností s použitím algoritmu na změnu detailů. Výsledky těchto meření jsou zaznamenány v tabulkách 5-8.

V tabulce 9 je měření bez použití tohoto algoritmu. Stačilo provést pouze jedno měření, protože na naměřené hodnoty nemá vzájemná vzdálenost těles žádný vliv. Měření byla proveda tak, že jsem kameru nastavil co nejblíže testovací kostce tak, aby se kostka jestě vešla do pohledu celá.

| Vzájemná vzdálenost těles = 200 | | | | |
|---|---|---|---|---|
| Počet těles na hraně | Počet těles | Počet trojúhelníků | FPS | Doba mezi snímky [s] |
| 2 | 8 | 16384 | 56 | 0,017857143 |
| 4 | 64 | 17792 | 51 | 0,019607843 |
| 6 | 216 | 28032 | 32 | 0,031250000 |
| 8 | 512 | 53440 | 19 | 0,052631579 |
| 10 | 1000 | 65312 | 14 | 0,071428571 |
| 12 | 1728 | 71256 | 13 | 0,076923077 |
| 14 | 2744 | 88936 | 11 | 0,090909091 |
| 16 | 4096 | 103330 | 9 | 0,111111111 |
| 18 | 5832 | 118870 | 6 | 0,166666667 |
| 20 | 8000 | 134730 | 5 | 0,200000000 |

Tabulka 5: Scéna s konstatní vzdáleností těles rovnou 200 s použitím dělícího algoritmu

| Vzájemná vzdálenost těles = 400 | | | | |
|---|---|---|---|---|
| Počet těles na hraně | Počet těles | Počet trojúhelníků | FPS | Doba mezi snímky [s] |
| 2 | 8 | 5248 | 103 | 0,009708738 |
| 4 | 64 | 7616 | 81 | 0,012345679 |
| 6 | 216 | 13344 | 55 | 0,018181818 |
| 8 | 512 | 15160 | 41 | 0,024390244 |
| 10 | 1000 | 21872 | 31 | 0,032258065 |
| 12 | 1728 | 24912 | 23 | 0,043478261 |
| 14 | 2744 | 26104 | 17 | 0,058823529 |
| 16 | 4096 | 33729 | 12 | 0,083333333 |
| 18 | 5832 | 46752 | 9 | 0,111111111 |
| 20 | 8000 | 64000 | 6 | 0,166666667 |

Tabulka 6: Scéna s konstatní vzdáleností těles rovnou 400 s použitím dělícího algoritmu

| Vzájemná vzdálenost těles = 600 | | | | |
|---|---|---|---|---|
| Počet těles na hraně | Počet těles | Počet trojúhelníků | FPS | Doba mezi snímky [s] |
| 2 | 8 | 2560 | 128 | 0,007812500 |
| 4 | 64 | 4160 | 104 | 0,009615385 |
| 6 | 216 | 6024 | 83 | 0,012048193 |
| 8 | 512 | 7456 | 59 | 0,016949153 |
| 10 | 1000 | 9032 | 40 | 0,025000000 |
| 12 | 1728 | 13872 | 27 | 0,037037037 |
| 14 | 2744 | 21976 | 18 | 0,055555556 |
| 16 | 4096 | 32768 | 12 | 0,083333333 |
| 18 | 5832 | 46656 | 9 | 0,111111111 |
| 20 | 8000 | 64000 | 6 | 0,166666667 |

Tabulka 7: Scéna s konstatní vzdáleností těles rovnou 600 s použitím dělícího algoritmu

| Vzájemná vzdálenost těles = 800 | | | | |
|---|---|---|---|---|
| Počet těles na hraně | Počet těles | Počet trojúhelníků | FPS | Doba mezi snímky [s] |
| 2 | 8 | 1408 | 143 | 0,006993007 |
| 4 | 64 | 3296 | 114 | 0,00877193 |
| 6 | 216 | 3936 | 93 | 0,010752688 |
| 8 | 512 | 4504 | 64 | 0,015625000 |
| 10 | 1000 | 8000 | 42 | 0,023809524 |
| 12 | 1728 | 13824 | 27 | 0,037037037 |
| 14 | 2744 | 21952 | 18 | 0,055555556 |
| 16 | 4096 | 32768 | 12 | 0,083333333 |
| 18 | 5832 | 46656 | 9 | 0,111111111 |
| 20 | 8000 | 64000 | 6 | 0,166666667 |

Tabulka 8: Scéna s konstatní vzdáleností těles rovnou 800 s použitím dělícího algoritmu

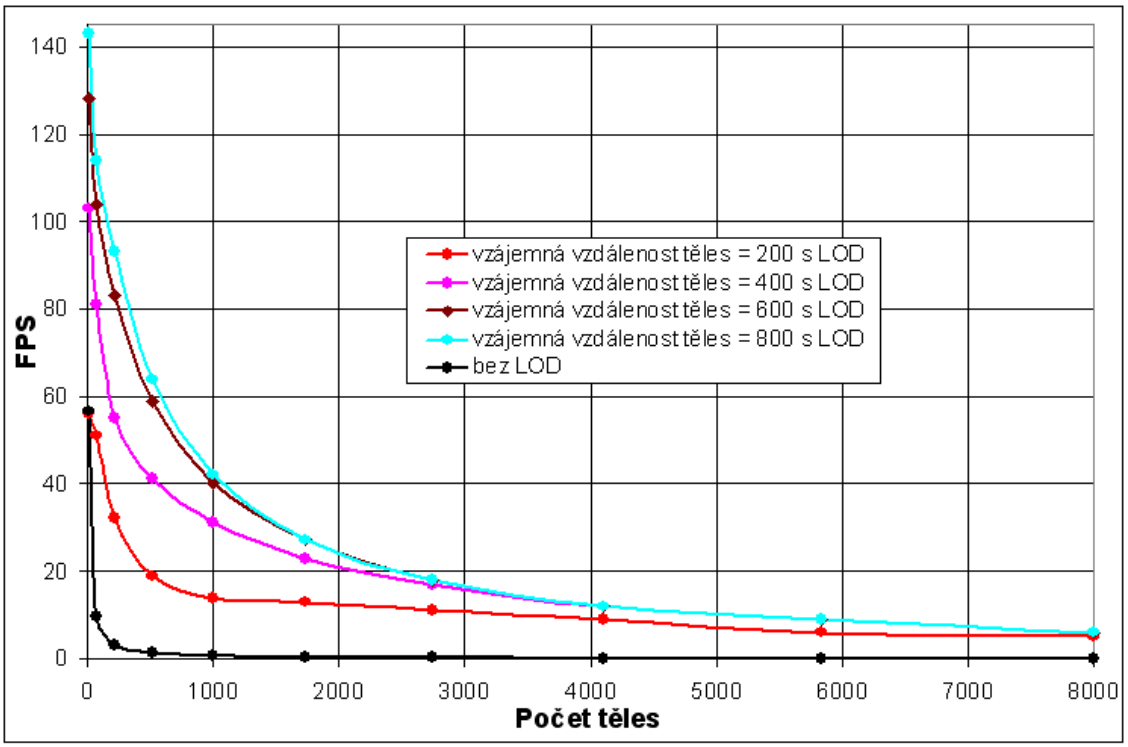
| Počet těles na hraně | Počet těles | Počet trojúhelníků | FPS | Doba mezi snímky [s] |
|---|---|---|---|---|
| 2 | 8 | 16384 | 56,6 | 0,017667845 |
| 4 | 64 | 131072 | 9,59 | 0,104275287 |
| 6 | 216 | 442368 | 3,000 | 0,333333333 |
| 8 | 512 | 1048576 | 1,257 | 0,795544948 |
| 10 | 1000 | 2048000 | 0,646 | 1,547987616 |
| 12 | 1728 | 3538944 | 0,376 | 2,659574468 |
| 14 | 2744 | 5619712 | 0,239 | 4,184100418 |
| 16 | 4096 | 8388608 | 0,158 | 6,329113924 |
| 18 | 5832 | 11943936 | 0,111 | 9,009009009 |
| 20 | 8000 | 16384000 | 0,080 | 12,500000000 |

Tabulka 9: Scéna s konstatní vzájemnou vzdáleností těles bez použití dělícího algoritmu

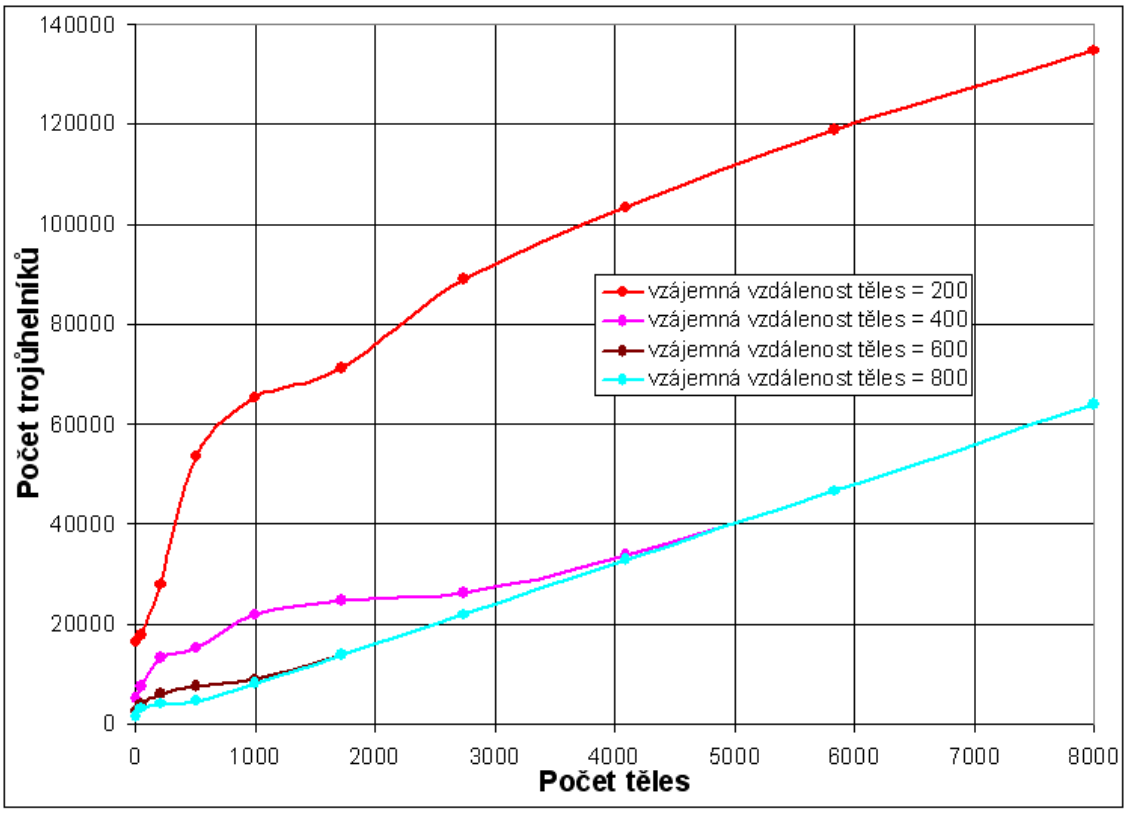
Vyhodnocení hodnot z těchto tabulek je zaznamenáno v grafech na obrázcích 22 a 23.

V grafu na obrázku 22 jsou závislosti počtu snímků za vteřinu na počtu těles. Pro porovnání jsou do tohoto grafu zaneseny i hodnoty naměřené ve scéně bez použití algoritmu na snižování detailů. Ty jsou opět výrazně horší oproti ostaním u kterách tento algoritmus použit byl. Je vidět, že zvyšováním vzájemné vdálenosti těles dosáhneme zvýšení počtu snímků za vteřinu, což je celkem logické. Čím vyšší je hodnota vzájemné vzdálenosti, tím menšího nárustu FPS po dalším zvýšení vzájemné vzdálenosti dosáhneme. Dalším zajímavým poznatkem, který toto měření přineslo, je že výsledný počet snímků za vteřinu není závislý pouze na počtu trojúhelníků obsažených ve scéně, ale i na počtu těles. Z tabulek 5-8 je vidět, že ve scénách se stejným počtem trojúhelníků a různým počtem těles vykazuje scéna s vyším počtem těles výrazně nižší počet snímků za vteřinu.

Graf na obrázku 23 zobrazuje závislosti počtu trojůhelníků na počtu zobrazených těles při různých úrovních vzájemné vzdálenosti těles. Na tom je vidět jaký vliv má počet těles a hustota vliv na počet trojůhelníků ve scéně. I zde je vidět, že čím větší je vzájemná vzdálenost těles tím menší vliv má další zvyšování této hodnoty.




Obrázek 22: Závislost FPS na počtu těles při konstantí vzájemné vzdálenosti těles




Obrázek 23: Závislost počtu trojúhleníků na počtu těles při konstantí vzájemné vzdálenosti těles

### 14.3.4 Shrnutí výsledků

V předchozích třech podkapitolách je na obrázcích a naměřených výsledcích demostrován přínos aplikace algoritmu na snižování detailu, který je značný a jedná se především o výkonový přínos. S jeho použitím je možné vytvářet velmi rozsáhlé scény v nichž počet snímků, které zvládne aplikace renderovat za sekundu, dostatečně velký na to, aby pohyb po scéně byl ještě plynulý. Bez jeho použití je maximalní rozsah scény, ve které je pohyb plynulý, zanedbatelný. To je hlavním přínosem používání tohoto algoritmu. Jeho využitím se podařilo optimalizovat velmi rozsáhlé scény bez patrné újmy na kvalitě zobrazení.

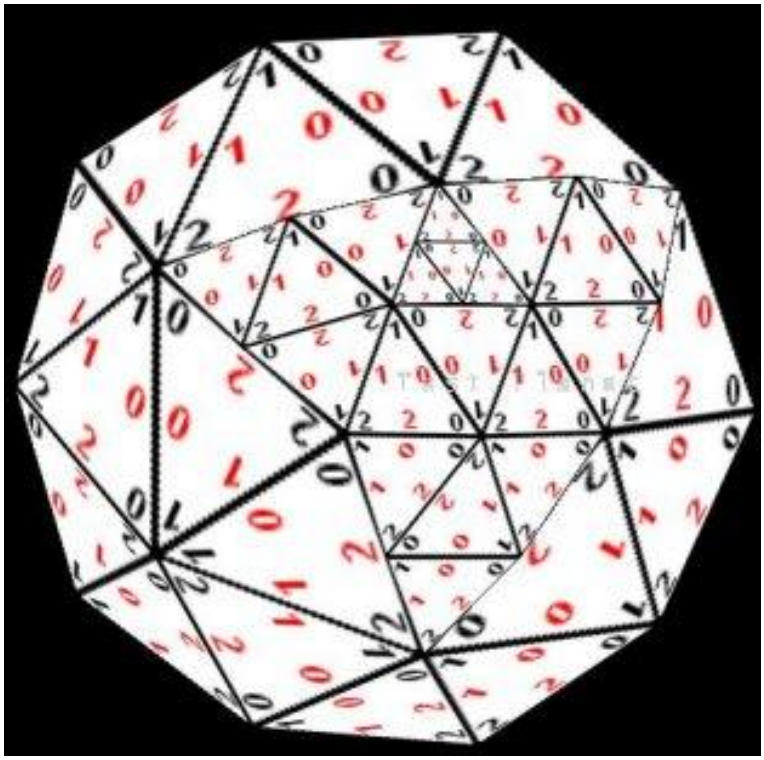
# 15 Možná rozšíření

I když generovaná tělesa jsou tvarově vcelku uspokojivá, musí se udělat ještě spousta práce proto, aby vypadala alespoň částečně reálně.

Aby se tvar těles blížíl realitě, bylo by zapotřebí upravit algoritmy generující náhodný tvar tak, aby generoval pouze jeden druh těles. Například pouze planety, nebo pouze asteroidy. Pak by bylo možno generovaný tvar optimalizovat tak, že by se co nejvíce blížil realitě. Tímto směrem by pravděpodobně směřoval další vývoj práce.

Dalším krokem k reálně vyhlížejícím vesmírným tělesům jsou pěkně vyhlížející textury. Nýní jsou sice tělesa texturami opatřena, ale nanesení textur není uspokojivé. Při bližším pohledu je na tělese zřetelný nehezký šev. Tento šev je způsoben chybou v knihovně Open Inventor, která by měla být v dohledné době opravena. Pokud se tak nestane je mozno nanesení textur rešit v práci.

Pro zvýšení výkonu aplikace by mohl být implementován lepší algoritmus na dělení trojúhelníků. V úvahu připadá takový, který by nedělil všechny trojúhlníky tělesa najednou, ale jen trojúhelníky, které dosáhli určité velikosti nebo vzdálenosti od kamery viz obrazek 24.



Obrázek 24: Lepší algoritmus na dělení trojúhelníků (převzato, viz kapitola [19])

# 16 Závěr

Na základě požadavků byl vypracován systém schopný optimalizovat velmi rosáhlé scény, které byly speciálně pro tyto účely vytvořeny. Optimalizované scény jsou jak z výkonového, tak z vizuálního hlediska uspokojivé.

Myslím si, že tento projekt je přínosem zejmená pro vývojáře her, protože snižování detailů je nutností pro zachování plynulosti pohybu v realtimových aplikacích. Byl bych velice potěšen, kdyby se tato práce stala součástí akademického projektu SpaceGame, nebo kdyby byla použita alespoň některá z jejích částí.

Za vlastní přínost považuji generovaná tělesa náhodných tvarů a scény které jsou z techto těles vytvořeny. Tvary jichž se mi podařilo náhodným generováním dosáhnout osobně považuji za více než uspokojivé. Dále za vlastní přínos považuji experimentálním způsobem zjištěný minimální počet úrovní detailů a okamžiky v nichž se tyto úrovně přepínají, takže přepínání mezi úrovněmi je z vizuálního hlediska přijatelné.

Projekt jsem si zvolil, protože počítačová grafika mě zajímá a ta realtimová obzvlášť. Zadání projektu bylo z mého pohledu úspěšně splněno.

# 17 Literatura

[1] Hlavenka, J. a kol.: Výkladový slovník výpočetní techniky a komunikací.
Computer Press 1997
ISBN 80-7226-023-5

[2] Virius, M.: Od C k C++.
Nakladatelství Kopp, 2000
ISBN 80-7232-110-2

[3] Wernecke, J.: The Inventor Mentor.
Addison-Wesley Professional, 1994
ISBN 02-0162-495-8

[4] Přednášky a cvičení z předmětu Počítačová grafika.
http://www.fit.vutbr.cz/study/courses/index.php?id=366

[5] Open Inventor - tutoriály
http://www.root.cz/clanky/open-inventor

[6] Algoritmus na dělení trojúhelníků
http://www.gamedev.net/reference/articles/article2074.asp

[7] Chvojka I.: Modelování explozí metodou částicových systémů
Ročníkový projekt, 2004

[8] Dokumentace knihovny Open Inventor
http://doc.coin3d.org/Coin/index.html

# 18 Přílohy

CD-ROM se zdrojovými kódy, dokumentací v elektronické podobě a zkompilované příklady použití práce.

## Uživatelská příručka

Interakce s uživatelem je řešena pomocí parametrů se kterými se projekt spustí z příkazové řádky. Nápovědu je možné zobrazit pokud jej spustíme s parametrem *-h* nebo - *-help*. Pak se nám vypíše následující seznam použitelných parametrů. Popis u parametrů je myslím dostatečně názorný pro pochopení jejich významu.

| Napoveda | |
|---|---|
| –help | help |
| -h | help |
| -l | vypnuti zmen detailu (implicitne zapnuta, nereaguje s parametrem in) |
| -k | zapnuti kolizi (implicitne vypnuty) |
| -sc:count | count = pocet teles generovanych kolem hvezdy (implicitne 10) |
| -out | vygenerovana scena bude ulozena do souboru (nelze s parametrem -in) |
| -in | scena bude nactena ze souboru (nelze s parametrem -out) |
| -file:name | name = jmeno souboru pro parametry -in a -out (implicitne scene.dat) |
| | |
| -t | testovaci scena (implicitne se generuje prezentacni scena) |
| -hr:hrana | hrana = vzdalenost teles na hrane testovaci krychle(implicitne 180) |
| -sp:pocet | pocet = pocet teles na hrane testovaci krychle (implicitne 5) |

Pokud jej spustíme bez parametrů spustí se s implicitním nastavením. Implicitně se generuje prezentační scéna ve které je 10 těles vygenerovaných kolem hvězdy. Je v ní zapnuto používání algoritmu pro snižování detailu a kolize jsou vypnuty.

# Příklady použití projektu

*create.exe -h*
- vypíše v konzoli nápovědu

*create.exe -k*
- vygeneruje se prezentační scéna s 10 tělesy kolem slunce, ve které budou zapnuty kolize i použití algoritmu pro snižování detailu.

*create.exe -l -sc:50 -out -file:data.dat*
-do souboru data.dat se nageneruje prezentační scéna s 50 tělesy kolem slunce ve které bude vypnuto použití algoritmu pro snižování detailu, pokud soubor existuje, bude přepsán. Po uložení do souboru se scéna zobrazí. V zobrazené scéně budou vypnuty kolize.

*create.exe -in -file:data.dat -k -l*
-ze souboru data.dat se načte scéna ve které budou zapnuty kolize, pokud soubor neexistuje vygeneruje se scéna s implicitními hodnotami. Parametr *-l* nebude brán v potaz, takže bude použit algoritmus pro snižování detailů.

*create.exe -t -k*
-bude vygenerována testovací scéna s 5 tělesy na hraně a vzdálenost těles na hraně krychle bude 180 bodů. Parametr *-k* nebude brán v potaz. V testovací scéně kolize nelze zapnout. Bude použito algoritmu pro snižování detailů.

*create.exe -t -sp:15 -hr:500 -l*
-bude vygenerována testovací scéna s 15 tělesy na hraně a vzdálenost těles na hraně krychle bude 500 bodů. Nebude použito algoritmu pro snižování detailů.

# 19 Seznam obrázků

Vykreslení pomocí OpenGL a knihovny Open Inventor
[Chvojka I.: Modelování explozí metodou částicových systémů] ..... 10
Graf jedoduché scény ..... 11
Jednoduchá scéna ..... 12
Základ vesmírného tělesa ..... 13
Rozdělení trojúhelníku ..... 14
Základní těleso po dělení trojúhelníku ..... 15
Základní těleso po dělení trojúhelníku a odsunutí bodu, je-li rozptyl roven 0 ..... 17
Základní těleso po dělení trojúhelníku a odsunutí bodu, je-li rozptyl roven 750 ..... 17
Ukázka tří úrovní detailu ..... 19
Ukázky prezentačních scén ..... 24
Ukázka testovací scény ..... 25
Detail komponenty zobrazující měřenou hodnotu ..... 30
Umístění komponent ..... 32
Drátěné modely tělesa, které se vzdaluje od kamery ..... 33
Modely tělesa, které se vzdaluje od kamery ..... 34
Ukázka prezentacní scény s použitím algoritmu na dělení troúhelníku ..... 35
Ukázka prezentacní scény bez použití algoritmu na dělení troúhelníku ..... 35
Ukázka testovací scény s použitím algoritmu na dělení troúhelníku ..... 36
Ukázka testovací scény bez použití algoritmu na dělení troúhelníku ..... 37
Závislost FPS na úrovni detailu pro jedno těleso ..... 39
Závislost FPS na počtu těles ve scéně s konstantní velikostí testovací krychle ..... 41
Závislost FPS na počtu těles při konstantí vzájemné vzdálenosti těles ..... 45
Závislost počtu trojúhleníku na počtu těles pči konstantí vzájemné vzdálenosti těles ..... 46
Lepší algoritmus na dělení trojúhelníku
[http://www.gamedev.net/reference/articles/article2074.asp] ..... 47