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

MĚŘENÍ GEOMETRICKÝCH PARAMETRŮ 3D MODELŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Pavel Svoboda
AUTHOR

BRNO 2007

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

MĚŘENÍ GEOMETRICKÝCH PARAMETRŮ 3D MODELŮ

GEOMETRIC MEASUREMENT OF 3D MODELS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Pavel Svoboda
AUTHOR

VEDOUCÍ PRÁCE Ing. Ondřej Šiler
SUPERVISOR

BRNO 2007

Zadání bakalářské práce

Měření geometrických parametrů 3D modelů

Geometric measurement of 3D models

Vedoucí:
Šiler Ondřej, Ing., UPGM FIT VUT
Oponent:
Švub Miroslav, Ing., UPGM FIT VUT
Přihlášen:
Svoboda Pavel
Zadání:

1. Seznamte se s problematikou měření geometrických parametrů 3D modelů
2. Nastudujte možnosti práce s 3D modely pomocí knihovny OpenSceneGraph
3. Navrhněte různé způsoby měření geometrických parametrů na modelu
4. Implementujte některé metody pomocí knihovny OpenSceneGraph
5. Zhodnoťte dosažené výsledky a stanovte další vývoj projektu

Kategorie:
Počítačová grafika
Implementační jazyk:
C++
Operační systém:
MS Windows, Linux, Unix
Literatura:

1. OpenSceneGraph online dokumentace http://www.openscenegraph.org
1. Žara J., Beneš B., Felkel P.: Moderní počítačová grafika. 1. vyd. Praha, Computer press 1998, 448 s., ISBN 80-7226-049-9

Licenční smlouva

Licenčí smlouva je uložena v archivu Fakulty informačních technologií Vysokého učeni technického v Brně.

Abstrakt

Práce pojednává o problematice měření základních geometrických vlastností na modelu pomocí knihovny OpenSceneGraph. Implementováno bylo jednoduché získávání hodnot vzdáleností a velikostí úhlů jak na tělese, tak v promítnuté rovině, definované uživatelem. Nechybí potřebná teorie pro měření v eukleidovském prostoru, ani vysvětlení použitých metod a konstrukcí pro implementaci pomocí Open Scene Graph knihoven. Práce zahrnuje jednoduché přiblížení použití knihovny OpenSceneGraph pro potřeby zadání. Součástí je také dokumentace a návod k ovládání.

Klíčová slova

Vzdálenost, úhel, geometrické vlastnosti, OpenGl, OpenSceneGraph, visitor, graf scény, strom scény

Abstract

Work is discussing questions about measuring essential geometric property on model using OpenSceneGraph library. Program contains simple distance and angle measurement on model and distance and angle measurement on projection plane, which is defined by user. There is not missing usefull theory for measuring in Euclidean space, neither explanation of used methods and constructs for implementation by Open Scene Graph. For better understandig of work with OpenSceneGraph here is simply described use of OpenSceneGraph library. Manual and technical documentation is part of this work too.

Keywords

Distance, angle, geometric properities, OpenGl, OpenSceneGraph, visitor, scene graph

Citace

Pavel Svoboda: Měření geometrických parametrů 3D modelů, bakalářská práce, Brno, FIT VUT v Brně, 2007

Měření geometrických parametrů 3D modelů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Ondřeje Šilera.
Další informace mi poskytli Ing. Přemysl Kršek, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Děkuji svému vedoucímu práce Ing. Ondřejovi Šilerovi za jeho ochotu při zodpovídání otázek a pomoc při řešení problémů.

© Pavel Svoboda, 2007.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah ... 1
1 Úvod ... 2
2 Teoretický rozbor ... 3
2.1 Analytická geometrie ... 3
2.1.1 Trojrozměrný eukleidovský prostor ... 3
2.1.2 Vektory ... 4
2.1.3 Rovina ... 5
2.1.4 Přímka ... 6
2.1.5 Metrické vlastnosti lineárních útvarů v E3 ... 6
2.2 Open Scene Graph ... 7
2.2.1 Graf scény ... 8
3 Návrh řešení ... 9
4 Implementace ... 10
4.1 Úvod ... 10
4.2 Užití Open Scene Graph knihoven ... 10
4.2.1 Práce s grafem scény v OSG ... 10
4.2.2 Smart pointers a správa paměti v OSG ... 11
4.2.3 Stavové množiny v OSG ... 11
4.2.4 Visitory a průchod grafem scény ... 13
4.2.5 Callback funkční třída ... 14
4.2.6 HUD v OSG ... 15
4.2.7 Komunikace v OSG ... 16
4.2.8 Text v OSG ... 18
4.3 Knihovna DistanceAngle ... 18
4.3.1 UML popis knihovny ... 18
4.3.2 Třída DistanceAngle ... 19
4.3.3 Třída PointText ... 22
4.3.4 Třída projectionRec ... 22
4.3.5 Třída Hud ... 24
4.4 Výsledná podoba grafu scény ... 24
5 Závěr ... 25
Literatura ... 26
Seznam příloh ... 27

1 Úvod

Pod pojmem měření parametrů geometrických 3D modelů si lze představit mnoho různých vlastností a atributů. V této práci se diskutují ty nejzákladnější a v praxi i snad nejčastěji používané. Vyspělé nástroje pro práci s trojrozměrnými modely dozajista obsahují metody pro získaní požadovaných informací. Na poli malých opensource projektů a programů však tyto vlastnosti programu jsou postrádané. Z tohoto důvodu vznikl projekt, který si klade za cíl vytvořit knihovnu, která ve spolupráci s OpenSceneGraph knihovnami dodá uživateli jednoduchý nástroj pro měření základních geometrických parametrů, konkrétně vzdálenosti mezi dvěma zadanými body, velikosti úhlu mezi dvěma polopřímkami definovanými třemi body, dále pak vzdálenosti mezi dvěma body promítnutými z tělesa na uživatelem definovanou projekční rovinu a velikost úhlu dvou polopřímek promítnutých na rovinu. Praktické využití těchto informací závisí na oboru. Přínosným se projekt jeví ve světle strojního inženýrství, stavebnictví, ale i architektury, či počítačově zpracovávaných medicínských dat.

Práce je dělena do 5 základních částí, které na sebe logicky navazují. V teoretickém rozboru je popsán postup, jak lze řešit zjišťování potřebných informací pomocí matematického aparátu. Zopakovány jsou definice a teze, které jsou stěžejní pro využití v projektu. Vysvětlen je postup jejich použití.

Návrh obsahuje postup správného využití informací z teoretického rozboru. Popisuje jejich propojení a včlenění do kontextu problému. Zároveň pojednává o vlastnostech, které by teoreticky měly být implementovány pro co nejvyšší komplexnost knihovny.

V části zabývající se implementací je důkladně vysvětlen postup řešení problémů. Dopodrobna je rozpracován popis implementovaných stěžejních funkcí knihovny. Pomocí UML diagramu je zřetelně znázorněna hierarchie tříd, použitých pro vytvoření knihovny. Je stručně vysvětlen postup a princip práce s OpenSceneGraph knihovnou.

V závěru nechybí zhodnocení práce a dosažených výsledků, kritické poukázání na neimplementované funkce knihovny, shrnutí a možnost dalšího vývoje.

V příloze je dodána technická dokumentace i uživatelský manuál, Nechybí ukázky ovládání a využití knihovny.

2 Teoretický rozbor

Vhodným nástrojem pro měření geometrických parametrů 3D modelů je matematický aparát zabývající se oborem analytické geometrie. Zodpoví základní otázky typu jak měřit. V dalších částech bude popsán systém, pomocí kterého se popisuje metrický prostor. Budou uvedeny a vysvětleny základní pojmy a nakonec budou tyto informace propojeny a zužitkovány, jinými slovy bude znázorněno jak tento matematický nástroj použít. Druhá část se zaměří na samotný OpenSceneGraph, knihovní nástroje pro objektově orientovanou práci nad OpenGl.

2.1 Analytická geometrie

Je mimo možnosti této práce popsat důkladně a s nadhledem analytickou geometrii, na které v podstatě staví výpočty měření a operací. Přes to, zde budou uvedeny základní kameny, na kterých projekt stojí z pohledu matematiky.

2.1.1 Trojrozměrný eukleidovský prostor

Je nutné zpočátku popsat systém, který bude používán pro měření. Pro účely projektu je nejvhodnější trojrozměrný eukleidovský prostor.

Zvolíme tedy v prostoru kartézskou soustavu souřadnic. Zvolíme pevný bod O, který budeme nazývat počátkem a tři navzájem kolmé přímky x, y, z procházející počátkem O, které nazveme osami. Přirozeným způsobem přiřadíme každému bodu reálné číslo tak, že číslo 0 je přiřazeno počátku soustavy souřadnic O. Kartézskou soustavu souřadnic budeme rozumět čtveřici <O, +x, +y, +z>, kde O je počátek a polopřímky +x, +y, +z jsou kladné části souřadnicových os x, y, z. Každému bodu A v prostoru můžeme nyní jednoznačně přiřadit uspořádanou trojici reálných čísel (a_1, a_2, a_3) a naopak každé uspořádané trojici reálných čísel (a_1, a_2, a_3) je přiřazen jednoznačně bod A prostoru. Řekneme, že čísla a_1, a_2, a_3 jsou souřadnice bodu A v kartézské soustavě souřadnic <O, +x, +y, +z>.

Pro eukleidovský prostor s dimenzí n značíme E_n. Zavedeme-li v n-rozměrném prostoru kartézskou soustavu souřadnic, platí pak, že vzdálenost d mezi dvěma body A, B o souřadnicích (y_1,y_2,..,y_n) a (y_1,y_2,..,y_n) je rovna :

$$d=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \qquad 1.$$

Definice trojrozměrného eukleidovského prostoru

Uspořádanou trojici A = (a_1, a_2, a_3) budeme nazývat bodem trojrozměrného eukleidovského prostoru $E_3 = R^3$, je – li definována vzdálenost

$$\rho(A,B)=\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+(a_3-b_3)^2} \qquad 2.$$

dvou libovolných bodů A a B = (b_1, b_2, b_3)

2.1.2 Vektory

Vektor bude definován tak, jak je obvyklé pro potřeby geometrie a fyziky. Tedy vektor představuje veličinu, která kromě velikosti obsahuje i směr, tím se liší od skaláru – obyčejného čísla, které má pouze velikost. Samotný vektor je objekt, jehož vlastnosti na použitém souřadnicovém systému nezávisí.

2.1.2.1 Definice vektoru

Vektor a je množina všech souhlasně orientovaných rovnoběžných úseček stejné délky.

1. Orientovanou úsečkou $\overrightarrow{AB}$ rozumíme uspořádanou dvojici bodů A, B, kde první bod A nazýváme počátečním a druhý bod B koncový.
2. Je-li $\overrightarrow{AB} \in a$, nazýváme úsečku $\overrightarrow{AB}$ umístěním vektoru a, které budeme nazývat vektorem a značit $\overrightarrow{AB} = B - A$.
3. Je-li $\overrightarrow{OX} \in x$, kde O je počátek souřadné soustavy, potom souřadnice bodu X jsou souřadnicemi vektoru x. Vektor $\overrightarrow{OX}$ nazýváme polohový vektor bodu X.

Věta 1. Pro dvě orientované úsečky $\overrightarrow{AB}$, $\overrightarrow{CD}$ platí $\overrightarrow{AB} \in x$ a zároveň $\overrightarrow{CD} \in x$ právě tehdy, když platí vztahy $b_i - a_i = d_i - c_i$ pro i = 1,2,3.

– Věta 1. říká, že pro výpočet souřadnic vektoru nezáleží na výběru jeho umístění.

Nechť $\overrightarrow{AB}$ je umístění vektoru u. Velikostí vektoru x, kterou označíme |u| nazýváme vzdálenost bodů A, B. Jestliže |u| = 1, potom říkáme, že x je **jednotkový vektor**.

Věta 2. Pro velikost vektoru u = (u_1, u_2, u_3), platí

$$|u| = \sqrt{u_1^2 + u_2^2 + u_3^2} \qquad 3.$$

Věta 3. Vektory x, y které jsou **lineárně závislé**, tj. $x = k \cdot y, \ k \in \mathbb{R}$, se nazývají **kolineární**. Jsou rovnoběžné a jejich vektorový součin je nulový.

Věta 4. **Lineární kombinací** dvou vektorů x, y je vektor z, pro který platí, že

$$z = s \cdot x + r \cdot y \quad s, r \in \mathbb{R} \qquad 4.$$

2.1.2.2 Operace s vektory

Skalární součin vektorů

Nechť φ je úhel dvou nenulových vektorů u, v. Skalárním součinem vektorů x, y rozumíme číslo, které budeme označovat x, y a které definujeme rovností

$$u \cdot v = |u||v| \ \cos(\varphi) \qquad 5.$$

Jestliže je alespoň jeden z vektorů nulový, pak definujeme $x \cdot y = 0$

Věta 1. Skalární součin dvou vektorů je roven nule právě tehdy, když oba vektory jsou buď nenulové na sebe kolmé, nebo alespoň jeden z vektorů je nulový.

Věta 2. Pro libovolné dva vektory u = (u_1, u_2, u_3), v = (v_1, v_2, v_3) platí:

$$\boldsymbol{u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3} \qquad 6.$$

Vektorový součin

Nechť jsou dány vektory u = (u_1, u_2, u_3) a v = (v_1, v_2, v_3). Vektor

$$\left(\begin{vmatrix} u_2 & u_3 \\ v_2 & v_3 \end{vmatrix} . \begin{vmatrix} u_3 & u_1 \\ v_3 & v_1 \end{vmatrix} . \begin{vmatrix} u_1 & u_2 \\ v_1 & v_2 \end{vmatrix} \right)$$

který značíme $u \times v$, se nazývá vektorový součin.

Věta 1. Vektorový součin $u \times v$ je vektor kolmý na vektory $u, v \in V_3$.

Věta 2. Pro každé dva vektory $u, v \in V_3$ platí

$$\boldsymbol{|u \times v| = |u||v| \sin(\varphi)} \qquad 7.$$

kde φ je úhel vektorů u, v.

2.1.3 Rovina

Předpokládejme, že v prostoru E_3 jsou dány body A, B, C neležící na jedné přímce. Těmito body prochází jediná rovina, kterou označíme ABC. Určíme vektory u = B - A , v = C – A, které jsou zřejmě lineárně závislé. Potom můžeme říci, že bod X leží v rovině ABC právě tehdy, když vektor X - A je lineární kombinací vektorů u, v. Potom platí

$$\boldsymbol{X - A = r \cdot u + s \cdot v \quad r, s \in \mathbb{R}}$$

z čehož plyne

$$\boldsymbol{X = A + r \cdot u + s \cdot v \quad r, s \in \mathbb{R}} \qquad 8.$$

Rovnici (1) nazýváme **vektorovou rovnicí** roviny ABC. Rovina ABC prochází bodem A říkáme, že má zaměření u, v.

2.1.3.1 Parametrická rovnice roviny

Jestliže A = (x_0, y_0, z_0), u = (u_1, u_2, u_3) a v = (v_1, v_2, v_3), potom dosazením do vektorové rovnice dostaneme rovnice

$$\begin{aligned} x &= x_0 + r\,u_1 + s\,v_1 \\ y &= y_0 + r\,u_2 + s\,v_2 \\ z &= z_0 + r\,u_3 + s\,v_3 \\ & r, s \in \mathbb{R} \end{aligned} \qquad 9.$$

které se nazývají parametrické rovnice roviny ABC.

2.1.3.2 Obecná rovnice roviny

Předpokládejme, že nenulový vektor n = (a, b, c) je kolmý k rovině α procházející bodem A. Označme X = (x, y, z) libovolný bod roviny α . Vektory n a $\vec{AX}$ jsou na sebe kolmé a tedy platí

$$n \cdot \vec{AX} = 0$$

Dostaneme :

$$\begin{aligned} n \cdot \vec{AX} &= n \cdot (X - A) \\ n \cdot X - n \cdot A &= (a, b, c) \cdot (x, y, z) - n \cdot A \\ ax + by + cz - n \cdot A &= 0 \end{aligned}$$

položíme – li $n \cdot A = -d$, dostaneme rovnici roviny ve tvaru

$$\boldsymbol{ax + by + cz + d = 0} \qquad 10.$$

Tuto rovnici nazveme obecnou rovnicí roviny α . Bod X = (x, y, z) leží v rovině α právě tehdy, když souřadnice x, y, z vyhovují rovnici roviny α . Vektor n se nazývá **normálový vektor** roviny α .

2.1.4 Přímka

Přímka může být dána v prostoru E_3 dvěma různými body A, B, kterou označíme AB. Mějme dále vektor u = B-A. Libovolný bod X leží na přímce AB právě tehdy, když vektor u a X-A jsou lineárně závislé, tzn. když existuje $t \in \mathbb{R}$ tak, že

$$X - A = t \cdot u$$

rovnici upravíme na tvar

$$\boldsymbol{X = A + t \cdot u}$$

tuto rovnici nazýváme vektorovou rovnicí přímky.

2.1.4.1 Parametrická rovnice přímky

Jestliže je dána přímka p bodem A = (x_0, y_0, z_0) a směrovým vektorem přímky u = (u_1, u_2, u_3) potom můžeme zapsat parametrickou rovnici přímky takto

$$\begin{aligned} x &= x_0 + t\,u_1 \\ y &= y_0 + t\,u_2 \\ z &= z_0 + t\,u_3 \\ t &\in \mathbb{R} \end{aligned} \qquad 11.$$

2.1.5 Metrické vlastnosti lineárních útvarů v E_3

Pro potřeby projektu se v zásadě potřebují tyto informace : odchylka dvou přímek a vzdálenost dvou bodů.

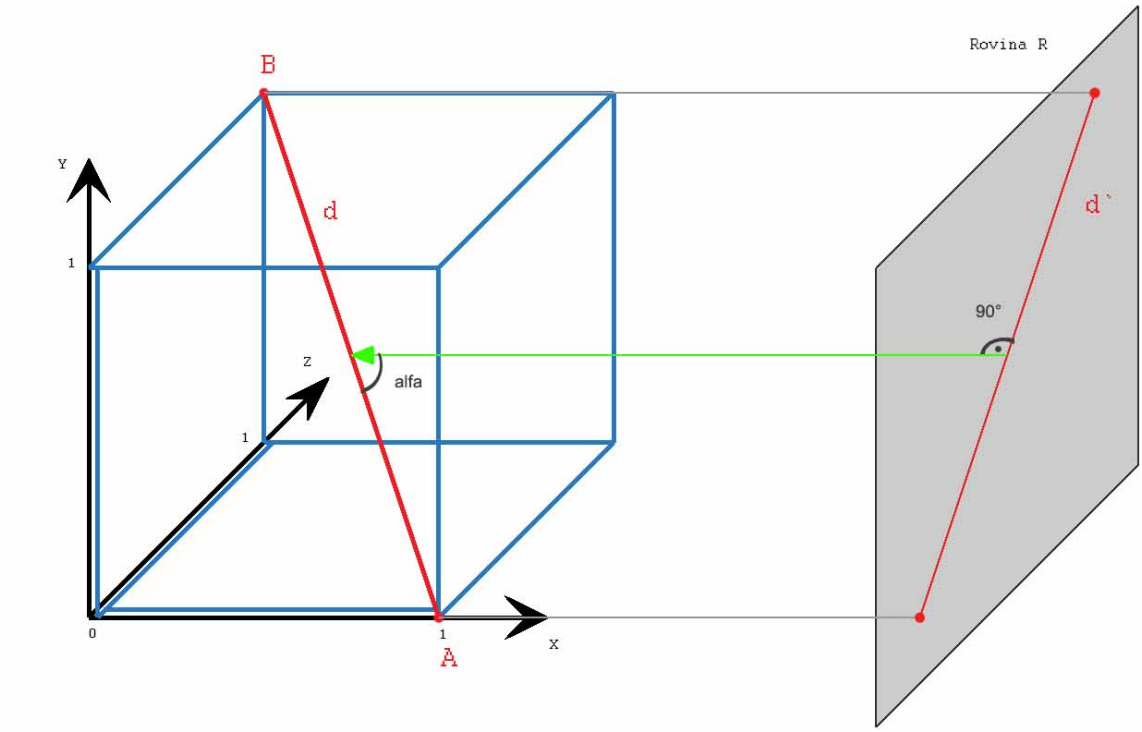
Odchylkou přímek p, q se směrovými vektory u, v rozumíme úhel φ pro který platí

$$\boldsymbol{\cos(\varphi) = \left| \frac{u\,v}{|u| \cdot |v|} \right|} \qquad 12.$$

Vzdálenost dvou bodů A = (x, y, z), B = (x_1, y_1, z_1) v E_3 je rovna

$$\rho(A, B) = \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2} \qquad 13.$$

Měření vzdálenosti a úhlu na promítnuté rovině lze řešit pomocí jednoduchých goniometrických funkcí, konkrétně sinus(α). Řešení je nastíněno na obrázku 1.



Obrázek 1: Zjištění velikosti úsečky, promítnuté na projekční rovinu.

Zjistíme úhel (na obrázku úhel alfa) mezi normálou roviny, kterou zadal uživatel a samotnou úsečkou definovanou dvěma body A,B. Velikost d` je pak rovna :

$$|d1| = \sin(alfa) \cdot |d| \qquad 14.$$

2.2 Open Scene Graph

Open Scene Graph, známý také pod zkratkou OSG, je OpenSource projekt nezávislý na okenním toolkitu, který je vyvíjen pro užití v grafických aplikacích jako hry, zobrazování dat a vizualizačním modelování.

OSG pracuje na bázi grafu scény, což poskytuje objektově orientovaný nástroj nad OpenGl, který umožňuje vývojářům objektový přístup namísto od nízkoúrovňového způsobu programování.

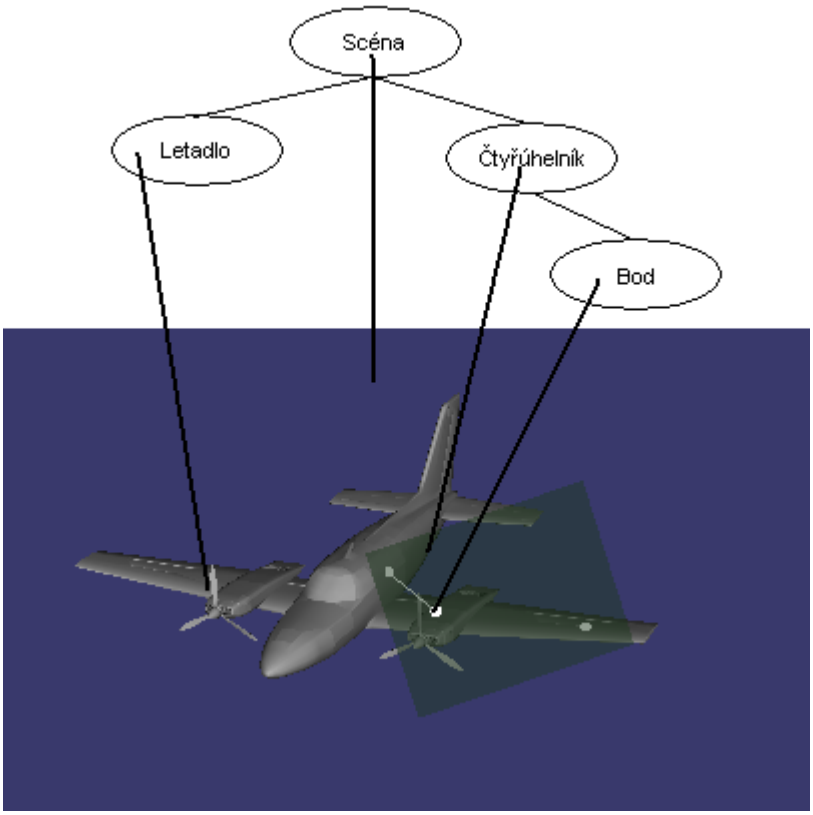
OSG běží na všech Windows systémech, OSX, GNU/Linux, IRIX, Solaris, HP-Ux, AIX a FreeBSD operačních systémech..

2.2.1 Graf scény

Graf scény je acyklický orientovaný graf. Popisuje scénu z pohledu objektů. Dovoluje odvozovat objekty od jiných, již v grafu zakomponovaných objektů. Každý uzel grafu představuje vlastnost a popis, například geometrie tělesa, jeho transformace atd. Hrany grafu reprezentují asociace mezi uzly na základě odvození a vlastností. Kořenový uzel bývá nazýván svět, nebo scéna. Na něj jsou připojovány objekty této scény. Open Scene Graph k popisu scény používá několik typů uzlů, jejichž základním předkem je uzel typu osg::Node:

1. osg::Geode - Geometric Node, i přes to, že jde o list, slouží k napojení dalších prvků, které reprezentují viditelné objekty – samotnou geometrii.
2. osg::Group – slouží jako uzel, který se chová jako kontejner, pro další uzly. Obsahuje metody pro správu objektů jako přidávání a odstraňování na něj napojených uzlů.
3. osg::Transform – uzel, který reprezentuje transformaci nad objektem popsaným v uzlu Geode, popřípadě nad všemi prvky uzlu Group. Dědí od objektu typu osg::Group, může tedy sloužit jako kontejner pro více objektů ve scéně.

Graf scény a jeho reprezentaci vidíte na obrázku 2.



Obrázek 2: Graf scény

3 Návrh řešení

Měření geometrických parametrů 3D modelů by mělo být lehce implementovatelné do programu, ve kterém najde své využití. Momentálně nejpraktičtějším řešením se zdá zakomponovat metody pro měření do speciální třídy, která bude k dosažení skrze samostatnou knihovnu, jež by uživatel – vývojář lehce připojil k programu. Projekt je vyvíjen nad Open Scene Graph knihovnami. Ty zaobalují OpenGl a přidávají možnosti objektového programování. Cílem tedy je vytvořit knihovnu, která by obsahovala třídu, starající se o samotné měření a odchytávání vstupů a následného zobrazení získaných informací. Pro uživatele to má stěžejní výhodu a to, že stačí ve svém programu tuto třídu vytvořit, ta svým konstruktorem zaregistruje metody pro odchytávání uživatelských vstupů z klávesnice a myši.

Uživatelsky jednoduché by bylo, kdyby třída skýtala intuitivní rozhraní pro používání. Jako jednoduchým ovládacím prvkem se zdá být kombinace myši a klávesnice, kdy uživatel stisknutím definované klávesy určí typ operace nebo měření a myší poskytne informace pro výpočet. Praktičtější by bylo, kdyby třída obsahovala strukturu pro uložení typu stisknuté klávesy a to v případě, kdy je jednoduší jedním stiskem klávesy zapnout provádění operace a druhým stiskem zrušit, než neustálým přidržením. Způsob ovládání je důležitý, měl by tedy být jednoduchý a intuitivní.

Otázkou je, jak poskytnout informace uživateli, které požaduje. Zpočátku byla zamýšlena varianta, že se budou tisknout do konzole na standardní výstup. Efektivnější a zároveň i snad mírně přehlednější je ale možnost, že výsledky měření budou zobrazeny přímo ve scéně na místě, které by bylo neustále v konstantní vzdálenosti od kamery a automaticky by se natáčelo ke kameře. Anglicky se toto nazývá HUD – Head Up Display. Písmo by mělo být pokud možno dobře čitelné a jednoduché.

Při měření nad tělesem je potřeba uživatele informovat o označených bodech, to znamená přidat do scény objekt označující pozici. Opět je třeba dbát na to, že by neměl tento prvek silněji narušovat scénu. Zároveň by se měl chovat jako bod konstantní velikosti, to znamená, že při jakémkoliv nastavení kamery by měl na obrazovce zabírat stejné místo.

Třída by měla umožňovat získávání informací typu vzdálenost dvou bodů označených na tělese. Měření úhlu mezi dvěma polopřímkami definovanými na tělese. V praxi použitelné měření je i promítnutí zadaných bodů na tělese na uživatelem definovanou projekční rovinu a výpočet vzdálenosti těchto dvou projekčních bodů, stejně tak i co se úhlů týče. Sofistikovanější metody by mohly nabízet vyspělé funkce pro měření vzdálenosti mezi dvěma body po povrchu tělesa. Možnost měření velikosti objemu určité části tělesa a stejně tak jeho povrchu.

4 Implementace

4.1 Úvod

Implementovány byly základní metody pro získání požadovaných informací, jako je vzdálenost dvou bodů na tělese, úhel mezi dvěma polopřímkami a ekvivalenty promítnuté na uživatelem definovanou rovinu. Základním nástrojem pro tvorbu knihovny byl použit programovací jazyk c++ s knihovnami Open Scene Graph. Funkčnost a použitelnost byla odzkoušena na stroji s operačním systémem Windows XP sp2 a v dnešní době průměrně výkonnou grafickou kartou Ati mobility radeon x700. Vzhledem k tomu, že se jedná o práci s openGl, je namístě zdůraznit, že výkon silně souvisí s možnostmi grafické karty. Open Scene Graph knihovny jsou použity ve verzi 1.2. Jako vývojové prostředí bylo použito Microsoft Visual Studio 2005, získané přes MSDN AA licenci.

Zpráva ohledně implementace vysvětlí práci s Open Scene Graph komponenty, budování grafu scény a typické konstrukce použití. Přiložen je UML diagram popisující stavbu třídy. V další části je důkladně popsána struktura výsledné třídy, její komponenty a vztahy mezi nimi. K závěru je zdokumentováno ovládání a přiloženo několik ukázek použití.

4.2 Užití Open Scene Graph knihoven

V části „Teoretický rozbor“ je popsán a vysvětlen pojem graf scény. Zde budou shrnuty základní poznatky pro práci s Open Scene Graph knihovnami ve vztahu k tomuto projektu. Bude popsáno vytvoření grafu a jeho správy, tím se rozumí přidávání a odstraňování uzlů, takzvané smart pointers, díky kterým lze v osg jednoduše nechat správa paměti na samotném OSG, způsob dědění vlastností uzlů potomků od uzlů rodičů, stavové množiny v OSG a možnosti jak procházet graf scény a vyhodnocovat tak různá data.

4.2.1 Práce s grafem scény v OSG

Uzel je reprezentován třídou, která obsahuje metody vždy v závislosti na typu uzlu. Graf scény je během běhu programu modifikovatelný, to znamená, že lze přidávat, ale také odstraňovat uzly za běhu.

Uzel typu **osg::Group** : funguje jako kontejner pro další uzly. Chceme-li přidat nový uzel, voláme metodu osg::Group::addChild(osg::Node* node), můžeme také využít metodu, která nám poskytne možnost rozhodnout, na které místo v kontejneru osg::Group uzel uložit, osg::Group::insertChild(unsigned int index, osg::Node * Node). Chceme-li uzel odstranit, využijeme osg::Group::removeChild(), kde jako parametr buď uvedeme ukazatel přímo na uzel, nebo index pod kterým je v kontejneru uložen.

Uzel typu **osg::Geode** : jeho podstatou je opět funkce kontejneru, ale tentokrát speciálních uzlů (dědí od virtuální třídy osg::Drawable), které popisují geometrii objektu. Do uzlu přidáme geometrii pomocí metody osg::Geode::addDrawable (osg::Drawable * node). Stejně jako v uzlu osg::Group i zde můžeme určit pozici přidávaného uzlu, slouží k tomu metoda osg::Geode::setDrawable (unsigned int index, osg::Drawable * drawable). K rušení uzlu ze scény využijeme metodu osg::Geode::removeDrawable (osg::Drawable * drawable), popřípadě osg::Geode::removeDrawables (unsigned int index, unsigned int number = 1), kde parametr number udává kolik dalších uzlů bude odstraněno po zadaném indexu včetně.

Uzel typu **osg::Transform** : chceme li manipulovat s objekty ve scéně, nepůjde to jinak než pomocí transformace. V praxi, uzel osg::Transform obsahuje jako své potomky uzly typu osg::Geode, nebo osg::Group. Pomocí transformačních matic jsou aplikovány transformace typu translace, rotace a změna měřítka na objekty v osg::Geode uzlech a následně jsou poté vyrenderovány ve scéně. Uzel typu osg::Transform dědí od osg::Group, tedy metody pro přidávání a odstraňování potomků jsou totožné.

V závěru se hodí říci, že typů uzlů je samozřejmě více[1], zde ale nebudou popsány, jelikož v projektu nebyly použity.

4.2.2 Smart pointers a správa paměti v OSG

Open Scene Graph obsahuje šablonu osg::ref_ptr<>, díky které programátor může přenechat správu uvolňování uzlů z paměti na samotném OSG. Vytvoříme-li uzel scény klasickou cestou jako osg::Geode* krychle = new osg::Geode(), je vnitřní počítadlo objektu nastaveno na 0. Připojíme-li uzel do grafu, bude mu počítadlo zvýšeno o 1 při každém připojení. Problém je, že bude-li se referenční počítadlo uzlu rovnat 0, nedojde k automatickém odstranění z paměti a je na programátorovi, aby tak učinil. Pomocí chytrých ukazatelů se tomuto lze šikovně vyhnout, protože vytvoříme-li uzel krychle takto : osg::ref_ptr<osg::Geode> krychle = new osg::Geode(), dostane se tímto uzel pod kontrolu samotného OSG a po odstranění ze scény, tj v momentě, kdy se bude jeho referenční počítadlo rovnat 0 bude automaticky zrušen v paměti. K samotnému uzlu se následně dostaneme pomocí metody osg::ref_ptr<>::get(), která vrací běžný ukazatel na objekt. Obecně je doporučováno používat tuto konstrukci pro vytváření uzlů a jejich správu.

4.2.3 Stavové množiny v OSG

Stavové množiny poskytují možnost ovlivnit OpenGl automat při průchodu grafem. V projektu byly využity pro definování, na kterých uzlech se nemá vypočítávat osvětlovací model a nastavení průhlednosti objektu. Jejich častým využitím je ale i popis textur a nebo materiálu. Atributy nastavujeme následujícím způsobem :

1 Viz příloha dokumentace k Open Scene Graph

1. Vytvoříme si ukazatel na množinu stavů – StateSet již vytvořeného uzlu (Zde projRecPat – osg::PositiontAttitudeTransform)

   osg::StateSet* StateSet = projRecPat->getOrCreateStateSet ();

2. Nyní můžeme nastavovat atributy, v tomto případě vlastnost průhlednost.

   StateSet->setMode (GL_BLEND, osg::StateAttribute::ON);

Nemá- li uzel definovánu stavovou množinu, je dle nastavení děděna množina předka. Při průchodu tuto vlastnost – dědění můžeme ovlivnit pomocí nastavení atributu osg::StateAttribute::Override, který zapříčiní, že i přes to, že v synovským uzlu změníme některý z atributů, který zdědil z otce, tak se tato změna neprojeví, neboť Override atribut otce přepíše vše na své hodnoty.

Atribut osg::StateAttribute::Protected zapříčiní opak atributu osg::StateAttribute::Overide, tedy uzel bude moci měnit i atributy, které v předkovi byly nastaveny jako Override.

Obrázek 3: Princip dědění stavových množin a jejich atributů (OpenSceneGraph Tutorials [2])

Obrázek 3 naznačuje princip dědění stavových atributů. Kořen grafu má nastavenou průhlednost textury. Tato jednoduchá stavová množina budu děděna všemi potomky, dokud nebude

jejich stavová množina změněna. Pravý podstrom kořene grafu naznačuje tuto variantu. Stavová množina v pravém podstromu kořenového uzlu nebyla změněna, tudíž dědí od kořene stromu. V uzlu(5) došlo jen ke změně textury. Levý podstrom kořenového uzlu je už ovlivněn jinak. Stavová množina uzlu(1) byla změněna, ostatní parametry zůstaly stejné a tak jsou děděny od kořene grafu, zatímco změněný parametr je dále děděn od uzlu(1), kde došlo ke změně. U uzlu(2) byla přidána vlastnost mlha – fog ON a atribut dědičnosti stavové množin byl nastaven na Override. V levém potomku uzlu(2) – uzel(3) byl atribut mlhy vypnut – fog Off. Dokud ale nemá tento uzel nastaven atribut dědičnosti na protected a předek - uzel(2) je nastaven na Override, zůstane atribut mlhy nastaven jako zapnutý – fog ON, jak bylo definováno v předkovi – uzel(2). Pravý potomek uzlu(2) – uzel(4) nastaví atribut mlhy na Protected a tím může přepsat nastavení předka – uzlu(2).

4.2.4 Visitory a průchod grafem scény

Průchod grafem lze v OSG řešit pomocí tzv. vistoru. Visitory jsou navrženy tak, aby dovolily uživatelům definovat specifické funkce spojené s určitým typem uzlu a typem průchodu grafem. Typy visitorů jsou CullVisitor – sbírá informace, UpdateVisitor – pro zaslání požadavku, EventVisitor – pro zaslání zprávy, IntersectVisitor – počítá průsečíky a další[2]. Jde nastavit jakým způsobem má visitor procházet grafem a to TRAVERSE_NONE – bez průchodu, TRAVERSE_PARENTS – průchod předky, TRAVERSE_ALL_CHILDREN – průchod všemi potomky, TRAVERSE_ACTIVE_CHILDREN – průchod aktivními potomky. V Open Scene Graph jsou uzly odvozeny od základní třídy osg::Node, která má metodu void osg::Node::traverse(NodeVisitor & nv). Ta pošle NodeVisitor všem svým potomkům pomocí volání jejich metody void osg::Node::accept(). Popřípadě tuto metodu obsahuje samotný NodeVisitor, jí pak jako parametr stačí dát referenci na právě zpracovávaný uzel, o rozeslání všem jeho potomkům se postará posléze sama. Pro pohyb nahoru grafem se místo traverse() metody volá ascend(). Ve visitoru pak stačí definovat metodu osg::NodeVisitor::apply(), která se provede při průchodu. Parametrem této metody můžeme rozlišit různé typy uzlů, chceme-li například, aby se metoda provedla na všech typech uzlů, dodáme ji parametr apply(osg::Node & uzel), ze kterého dědí všechny uzly. Chceme-li, aby se vykonala jiná akce na uzlu typu osg::Geode a uzlu typu osg::Group, přetížíme metodu apply takovýmto způsobem :

```
apply(osg::Group & group)
{
        tělo metody apply pro uzly typu osg::Group
        traverse(group)
}

apply(osg::Geode & geode)
```

2 Viz příloha dokumentace k Open Scene Graph

```
{ tělo metody apply pro uzly typu osg::Geode }
```

Visitor posléze inicializujeme od uzlu, od kterého potřebujeme. Například tak, že jej předáme jako parametr metodě accept.

```
uzel->accept (visitor);
```

4.2.5 Callback funkční třída

Callback může být chápán jako uživatelsky definovaná funkce, která se automaticky provede v závislosti na typu průchodu grafem. Callback může být spojen se samostatným uzlem, nebo s daným typem uzlů. Během průchodu grafem určitého typu se callback vyvolá vždy, je-li spojen s uzlem, přes který se tento průchod aplikuje.

Callback je v pravém slova smyslu funkční třída, existuje několik typů, které záleží na typu průchodu grafem.

- **CullCallback** – definujeme v případě, že budeme chtít vykonat určitou akci při požadavku na ořezání scény.
- **UpdateCallback** – použijeme při změně geometrie.
- **EventCallback** – zpracování právy od Vieweru.

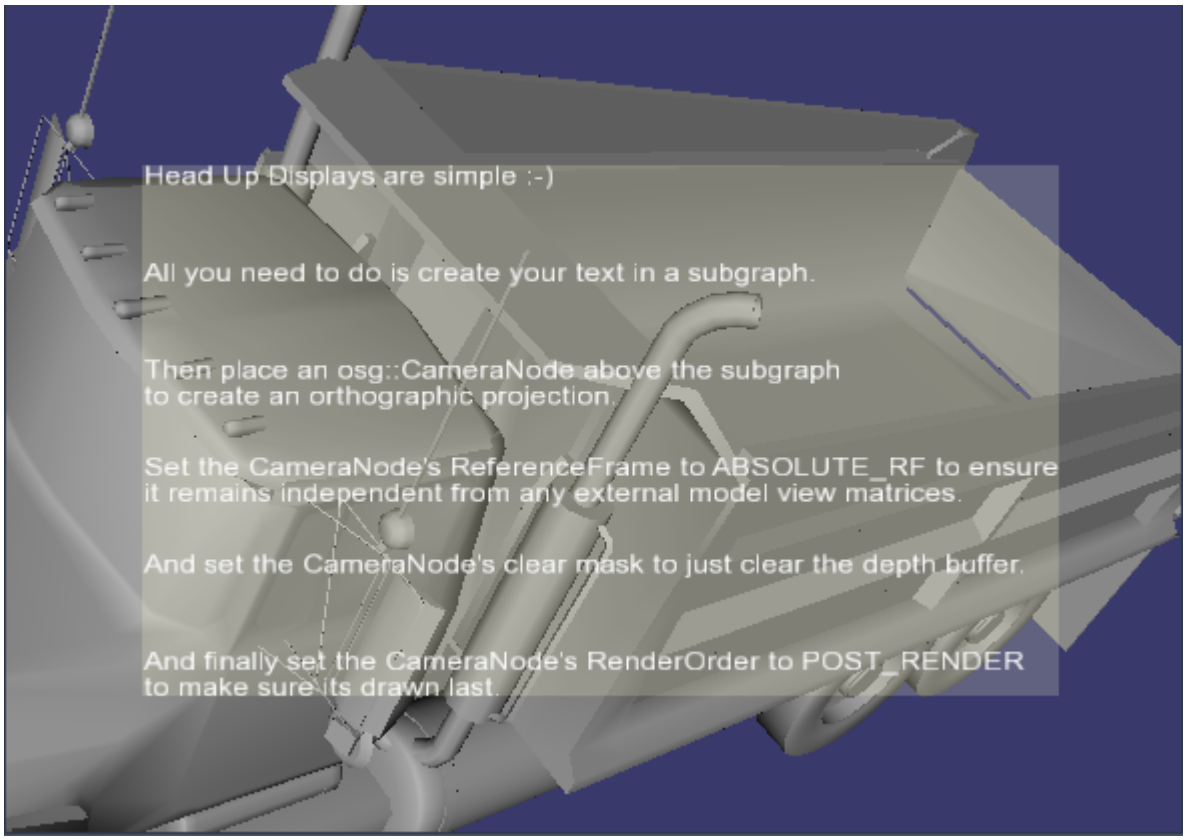
Jak je napsáno, callback v OSG je funkční třídou, vytvoříme jej poděděním od třídy osg::NodeCallback. Potom stačí přepsat virtuální operátor (), který má dva parametry, ukazatel na uzel a ukazatel na NodeVisitor.

```
Class Callback : public osg::NodeCallback
{
        public:
        virtual void operator()(osg::Node* node, osg::NodeVisitor* nv)
        {
                Definované tělo...
                traverse(node, nv);
        }
};
```

Oba parametry si dosadí samo OSG, jediné co musí programátor udělat je zaregistrovat callback k příslušnému uzlu pomocí metody uzlu setUpdateCallback(osg::NodeCallback * callback) - v případě typu průchodu-traversal Update.

4.2.6 HUD v OSG

Jde v podstatě o řešení, jak zajistit, aby některé informace byly jakoby přilepené ke kameře a tudíž se chovaly staticky vůči procházení scény ve vieweru. HUD je znázorněn na obrázku 4, kdy objektem v pozadí lze hýbat, zatímco text na bílém, průhledném čtyřúhelníku zůstává na místě.



Obrázek 4: Head Up Display – HUD (Ukázka z dema osghud – součást OSG)

Způsobů, jak vytvořit HUD je více. Možností je připojit do grafu scény uzel typu osg::CameraNode :

1. Vytvoříme uzel typu CameraNode, který se chová jako kamera a zároveň lze připojit do grafu scény.

   ```
   osg::CameraNode* camera = new osg::CameraNode;
   ```

2. Nastavíme ortografickou projekci.

   ```
   camera->setProjectionMatrix(osg::Matrix::ortho2D(0,1280,0,1024));
   ```

3. Zajistíme, aby uzel nebyl ovlivněn žádnou transformací po předkovi.

   ```
   camera->setReferenceFrame(osg::Transform::ABSOLUTE_RF);
   ```

4. Zinicializujeme pohledovou matici na jednotkovou.

   ```
   camera->setViewMatrix(osg::Matrix::identity());
   ```

5. Nastavíme pořadí vykreslování, vše co bude připojeno na uzel camera se vykreslí po tom, co se vykreslí celá scéna.

```
camera->setRenderOrder(osg::CameraNode::POST_RENDER);
```

Tímto jsme vytvořili Head Up Display. V projektu byl zvolen druhý postup, ten je následující :

1. Vytvoříme si projekční matici : čtyřúhelník, na kterém je promítnut HUD

   ```
   HUDProjectionMatrix = new osg::Projection();
   ```

2. Nastavíme typ projekce a rozměry viewportu náležející k HUD

   ```
   HUDProjectionMatrix->setMatrix(osg::Matrix::ortho2D(0,1024,0,768));
   ```

3. Vytvoříme pohledovou matici, kterou posléze připojíme jako potomka k projekční matici.

   ```
   osg::ref_ptr<osg::MatrixTransform> HUDModelViewMatrix = new
   osg::MatrixTransform();
   HUDProjectionMatrix->addChild(HUDModelViewMatrix.get());
   ```

4. Zinicializujeme pohledovou matici na jednotkovou.

   ```
   HUDModelViewMatrix->setMatrix(osg::Matrix::identity());
   ```

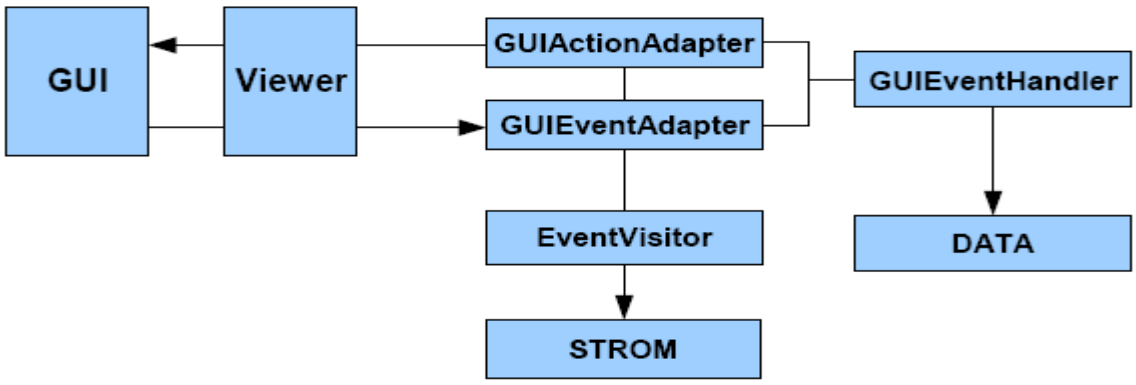
5. Zajistíme, aby nebyl uzel pohledové matice ovlivněn žádnou transformací.

   ```
   HUDModelViewMatrix->setReferenceFrame(osg::Transform::ABSOLUTE_RF);
   ```

Dále už jen stačí na pohledovou matici připojit uzly, které se mají chovat jako Head Up Display.

4.2.7 Komunikace v OSG

Komunikace stojí na zasílání zpráv. Snad nejdůležitější jsou zprávy od GUI – OSG aplikaci. To se děje přes prostředníka – okenní toolkit, v OSG implementovaným Viewerem z OpenProduceru. Dále pak mohou být zprávy zasílány uzlům grafu scény, nebo uživatelským datům mimo graf scény. Hierarchii a uspořádání dokonale vystihuje obrázek 5.



Obrázek 5: Schéma znázorňuje systém zasílání a přijímání zpráv v OSG (OpenSceneGraph Tutoriál [1])

Stiskne-li uživatel aplikace tlačítko myši někde ve scéně, předá GUI tuto zprávu Vieweru, ten ji zabalí do formátu OSG, tím je objekt třídy osg::GuiEventAdapter. V něm jsou uloženy veškeré související informace, jak s nimi naložíme je už na nás. Buď můžeme poslat EventVisitor grafem scény a reagovat budou callback funkce, nebo zprávu zpracujeme mimo graf scény pomocí funkční třídy odvozené od osg::GUIEventHandler. V projektu je použito zpracování pomocí GUIEventHandleru. Naopak i OSG aplikace může posílat zprávy samotnému GUI a to přes GUIActionAdapter. V projektu nebylo zasílání zpráv ke GUI použito, proto zde tato možnost popsána nebude[3].

osgGA::GUIEventHandler

V projektu je vytvořen objekt typu osgGA::GUIEventHandler, ve kterém se definuje jak na které zprávy reagovat. Instance Vieweru obsahuje svůj vnitřní seznam objektů typu osgGA::GUIEventHandler, který je při obdržení zprávy procházen a vyhodnocován. Je-li v objektu zpráva zpracována a už se nemá posílat dál, vrátí metoda virtual bool osgGA::GUIEventHandler::handle (const GUIEventAdapter &, GUIActionAdapter &); bool hodnotu true, pokud se má dále zpracovat dále, vrací false. Zprávy lze potom odlišit a reagovat na ně tímto způsobem :

```
//GUIEventAdapter je objekt zaobalující zprávu od GUI pro OSG aplikaci.
bool handle(const osgGA::GUIEventAdapter& ea, osgGA::GUIActionAdapter& aa)
{
//Metodou getEventType() zjistíme o jaký typ zprávy jde
        switch(ea.getEventType())
        {
        //Jedná se o stisk klávesy
        case(osgGA::GUIEventAdapter::KEYDOWN):
                //Metoda getKey() vrací o jakou konkrétně klávesu šlo.
                if(ea.getKey() == 'a')
                {
                        //Zde můžeme reagovat na zprávu.
                        return true;
                }
```

Samotný objekt typu osgGA::GUIEventHandler definovaný uživatelem je nakonec potřeba zaregistrovat do seznamu Vieweru :

```
viewer.getEventHandlerList().push_front(osgGA::GUIEventHandler & h)
```

3 Více viz příloha dokumentace k Open Scene Graph

Tímto jej umístěn na začátek seznamu a tedy námi definované akce na různé zprávy se provedou jako první.

4.2.8 Text v OSG

Text je jako jakýkoliv jiný objekt ve scéně uzlem v grafu scény. OSG nabízí dobře propracovaný systém pro zobrazování textu. Text je odvozen od třídy typu osg::Drawable. Plně podporuje práci s TrueType písmy. Jako všechny třídy odvozené od osg::Drawable i osgText::Text se musí připojit na uzel typu osg::Geode v grafu, aby mohl být vyrenderován. Samotná třída osgText::Text poskytuje metody pro intuitivní nastavování parametrů textu, jako písmo, velikost, barvu, umístění, natočení a mnoho dalších[4]. Text samotný nastavíme metodou osgText::Text::setText(std::string) popřípadě osgText::Text::setText(osgText::string), v druhém případě jde o speciální typ stringu z OSG, který podporuje multibyte kódování. V Open Scene Graph jde text metodou osgText::Text::setText (std::string) bez problému měnit za běhu programu.

4.3 Knihovna DistanceAngle

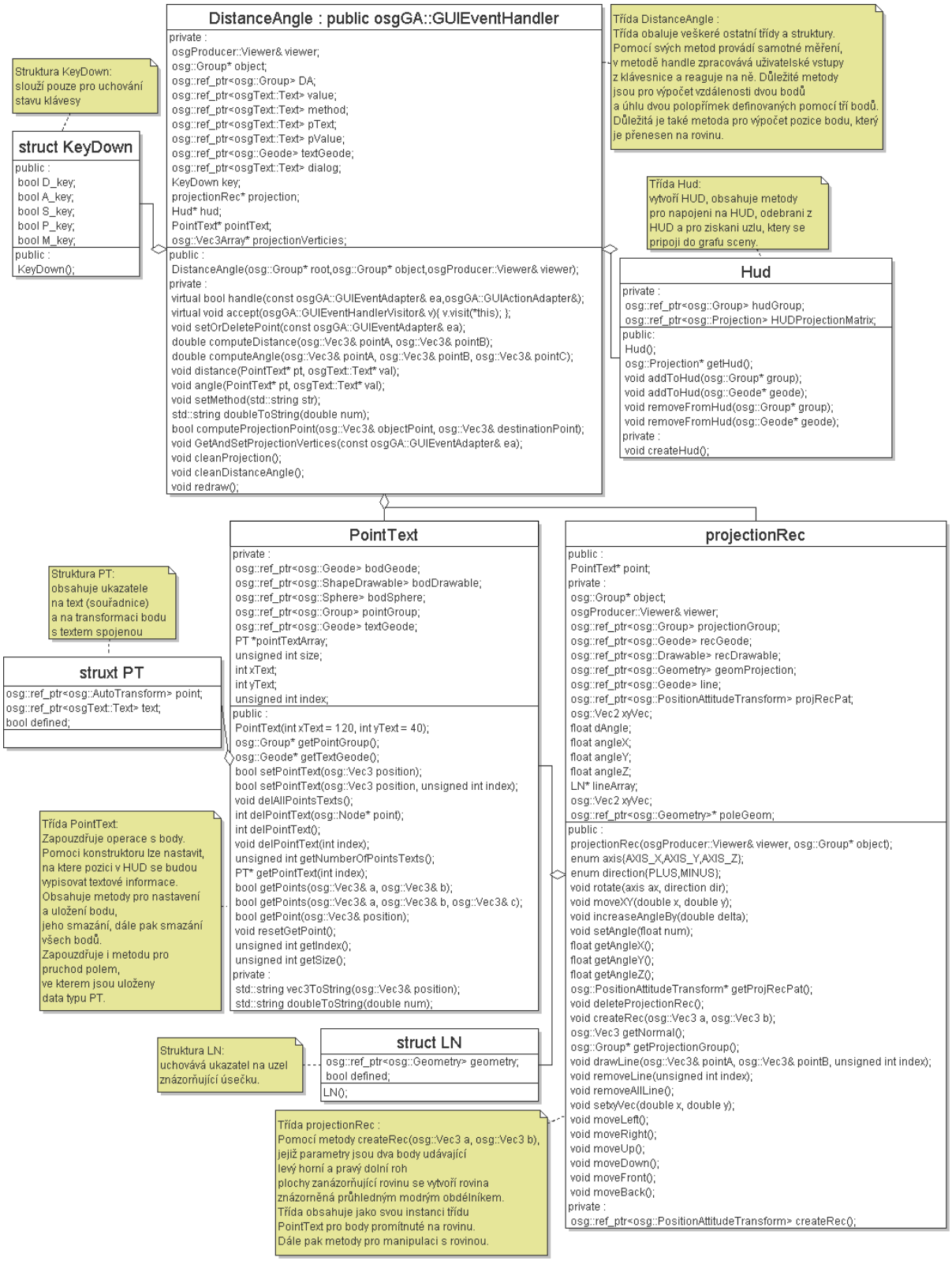
V dalších podkapitolách bude popsána samotná knihovna DistanceAngle. Vysvětlena bude její struktura, podrobně si popíšeme části, ze kterých se skládá a jakým způsobem spolu jednotlivé komponenty komunikují. Knihovna je ve svém základě velice jednoduchá k použití, uživateli, který ji chce zakomponovat do svého projektu stačí vytvořit objekt typu DisatnaceAngle.

4.3.1 UML popis knihovny

Na obrázku 6 je vyobrazen UML popis knihovny. Je z něj zřejmé, že samotná třída DistanceAngle se skládá z dalších 4 objektů a to typu KeyDown, PointText, ProjectionRec a Hud. Tyto čtyři třídy jsou ve své podstatě závislé na své agregační třídě DistanceAngle, pro kterou ostatně vznikly.

- struct KeyDown uchovává informace o stavu ovládacích kláves.
- struct PT je pomocná datová struktura, která nese informaci o provázaní zadaného bodu na objektu s textem souřadnice vypisovaným na HUD
- Struct LN nese ukazatel na uzel, který reprezentuje úsečku v grafu scény
- class PointText zaobaluje veškerou správu zadaných bodů a jejich souřadnic
- class ProjectionRec má na starost vše, co se týče projekční roviny
- class Hud vytváří Head Up Display, na který se v grafu připojují textové informace
- DistanceAngle je třída která zaobaluje a propojuje komponenty tak, aby vzájemně komunikovaly a byly použitelné, zároveň sbírá data ze svých podobjektů a provádí výpočetní operace s nimi.

4 Viz příloha dokumentace k Open Scene Graph



Obrázek 6: UML návrh knihovny DistanceAngle

4.3.2 Třída DistanceAngle

Třída DistanceAngle zaobaluje implementované měření a práci s prvky, které s měřením na tělese či na projekční rovině souvisí. V projektu dědí od třídy osgGA::GUIEventHandler, je tomu tak z

důvodu virtuální metody virtual bool osgGA::GUIEventHandler::handle (const GUIEventAdapter &, GUIActionAdapter &); ve které je definován způsob, jak naložit se zprávami od GUI. Třída definuje, jak napojit na graf scény uzly pro zobrazení informací o měření. Hlavním uzlem je osg::Group DA, který se chová jako kontejner pro další uzly, které reprezentují textové, a grafické informace. Konstruktor třídy potřebuje jako své parametry ukazatel na kořenový uzel scény – na ten je připojen uzel DA, ukazatel na uzel reprezentující měřené těleso typu osg::Group a referenci na instanci Vieweru z OpenProducer knihovny (Ta je ve verzi OSG 1.2 přímo v OSG). Uzel reprezentující těleso udává odkud budou vysílány visitory pro hledání průsečíku kliknutí myši a tělesa. Pomocí reference na viewer je možné registrovat osgGA::GUIEventHandler do vnitřního seznamu vieweru, který je procházen po přijmutí zprávy, samotná registrace se provádí v konstruktoru třídy. Pomocnou strukturou je KeyDown, která obsahuje boolovské hodnoty definující, je li určitá klávesa zmáčknuta či ne. Využívá se pro definování operace pomocí klávesy a myši zároveň. Objekt typu projectionRec se stará o projekční rovinu, to znamená promítání bodů z tělesa na ni a manipulaci s ní. Objekt se jménem hud a typu Hud zaobaluje Head Up Display a práci s uzlem, který se o HUD transformaci stará. Objekt pointText typu PointText se stará o zadané body a jejich souřadnice v textové formě, chová se jako kontejner s metodami pro jeho procházení, zadávání a rušení bodů. Vnitřní uzel třídy DistanceAngle osg::Geode TextGeode obsahuje veškerý text, který se týká měření (druh a výsledek) a dialogu (v případě zadávání projekční roviny), v konstruktoru se připojí na HUD

Hlavními metodami jsou angle(PointText* pt, osgText::Text* val) a distance(PointText* pt, osgText::Text* val). Ty díky tomu, že se jim předává ukazatel na zdroj bodů (PointText* pt) se můžou použít jak pro práci s body na tělese tak pro práci s body promítnutými na projekční rovině. Stěžejní metodou ve smyslu ovládání a rozesílání zpráv je metoda bool handle (const osgGA::GUIEventAdapter & ea, osgGA::GUIActionAdapter &).

- **void angle(PointText* pt, osgText::Text* val)** Metoda potřebuje dva parametry, ukazatel na třídu PointText, ze které si zjistí pozice definovaných bodů a ukazatel na objekt typu osgText::Text reprezentující textovou informaci velikosti úhlu, která se vypíše ve scéně. Po získání souřadnic z PointText* pt se tyto informace zpracují v metodě computeAngle(osg::vec3 & a, osg::vec3 & b, osg::vec3 & c), ve které se podle vzorce[5] vypočítá odchylka dvou polopřímek definovaných právě body ba a bc.

- **void distance(PointText* pt, osgText::Text* val)** Funguje na stejném principu jako metoda angle(), opět použije první parametr jako zdroj souřadnic bodů, se kterými se pracuje a do druhého parametru uloží text, který se vypíše – hodnota měření. Uvnitř je volána funkce

5 Vzorec 12 – výpočet odchylky dvou vektorů. Kapitola 2.1.5

computeDistance(osg::vec3 & a, osg::vec3 & b), která podle vzorce[6] vypočítá vzdálenost dvou bodů v eukleidovském prostoru.

- **bool handle(const osgGA::GUIEventAdapter& ea, osgGA::GUIActionAdapter&)** Obsahuje přepínač, kterým se ovládá a definuje měření. V parametru osgGA::GUIEventAdapter & ea je zabalena zpráva pro OSG od vieweru, která obsahuje informace o typu zprávy a atributy jako klávesa, nebo pozice myši. Metoda vrací bool hodnotu, podle které se určí, jestli se má zpráva poslat dále, nebo ne.

- **void GetAndSetProjectionVertices(const osgGA::GUIEventAdapter& ea)** Metoda sbírá zadané body od uživatele, které následně poskytne objektu projection typu projectionRec. Uživatel je vyzván, aby zadal dva body projekční roviny a to horní levý a dolní pravý. Při stisku myši GUI zašle zprávu aplikaci, která ji odchytí a zpracuje. Pomocí metody vieweru bool bool osgProducer::Viewer::computeNearFarPoints (float x, float y, unsigned int cameraNum, osg::Vec3 & NearPoint, osg::Vec3 & FarPoint); vypočítá souřadnice dvou bodů. NearPoint naplní světovými souřadnicemi bodu, který vznikne průsečíkem pomyslného paprsku (ten je definovaný pomocí směrového vektoru nastavení pohledu kamery a souřadnicemi kliknutí myši) a obalového tělesa 3D modelu (tzv. BoundingBox nebo BoundigCube), jedná se o ten průsečík, který je blíže ke kameře. FarPoint se zjistí pomocí stejného principu, jen se jedná o bod, který je od kamery více vzdálen. Zde ovšem použijeme jen NearPoint. Jakmile uživatel zadá oba dva, předají se dál pro vytvoření projekční roviny. Je namístě zmínit, že v pravém slova smyslu nejde o rovinu, ale jen o část roviny, která je reprezentována definovaným čtyřúhelníkem.

- **bool computeProjectionPoint(osg::Vec3 & objectPoint, osg::Vec3 & destPoint)** Pro výpočet pozice promítnutého bodu z tělesa na projekční rovinu je potřeba znát tyto informace : souřadnice bodu a normálový vektor uživatelem definované roviny. Souřadnice dodáme metodě jako parametr, normálu si metoda sama vyžádá od objektu projection starajícího se o projekční rovinu. Po té co známe souřadnice bodu a normálový vektor roviny můžeme definovat polopřímku, jejímž směrovým vektorem bude obrácený normálový vektor roviny a bodem bod na tělese. Tuto polopřímku předáme jako parametr Visitoru – osgUtil::IntersectVisitor[7] ve formě osg::LineSegment[8], který projde zadanou částí grafu scény a existuje-li průsečík polopřímky s rovinou, najde jej a vypočítá. Výsledné souřadnice udávají

6 Vzorec 13 – výpočet vzdálenosti dvou bodů v Eukleidovském prostoru. Kapitola 2.1.

7 Viz Visitory a průchod grafem scény. Kapitola 4.2.4

8 Viz příloha dokumentace k Open Scene Graph.

promítnutý bod z tělesa na rovinu. V případě nenalezení průsečíku vrací metoda bool hodnotu false, v případě úspěchu true.

4.3.3 Třída PointText

Třída se chová jako speciální kontejner pro body a jejich souřadnice v textové formě. V konstruktoru je vytvořeno dynamicky alokované pole struktur PT (Ty obsahují dvě položky, ukazatel na Textový uzel a ukazatel na uzel reprezentující bod) o velikosti 3. Díky tomu lze jednoduše svázat informaci o bodu s informací zobrazených souřadnic. Pomocí konstruktoru třídy lze určit pozice textu zobrazovaného na HUD části scény. Tím se stává tato třída obecnější, neboť ji jde bez jakýchkoliv úprav využít i ve třídě projectionRec[9]. Do grafu scény jsou z ní připojeny uzly pointGroup (kontejner pro zobrazované body) a textGeode (kontejner pro textové informace o souřadnicích). Bod je ve scéně připojen jako uzel speciálního typu transformace, která se automaticky mění podle polohy kamery. Díky tomu se bod jeví konstantní velikosti při jakémkoliv přiblížení či oddálení kamery k/od 3D modelu. Konkrétně se jedná o třídu osg::AutoTransform, která dědí z osg::Transform, a její metodu void osg::AutoTransform::setAutoScaleToScreen(bool), kterou pomocí bool hodnoty v parametru nastavíme tuto vlastnost (true – znamená přepočítávat transformaci, false opak).

Třída obsahuje metody pro pohyb a správu pole, ve kterém jsou uloženy struktury, které obsahují ukazatele na uzly reprezentující body a uzly reprezentující text. Z tohoto hlediska je významná metoda bool PointText::getPoint(osg::Vec3 & position).

- **bool getPoint(osg::Vec3 & position)** Po jejím volání jsou v parametru position vráceny souřadnice bodu, zároveň dojde k posunutí v poli na následující prvek. Tím lze voláním této metody, dokud nevrátí bool hodnotu false, procházet celým polem. Při překročení posledního prvku dojde k nastavení ukazatele opět na počáteční prvek.
- **bool SetPointText(osg::Vec3 position, unsigned int index)** Metoda nastaví pozici bodu na požadovaném indexu v poli a samotný bod připojí do scény a definuje jako platný.
- **int delPointText(osg::Node* point)** Metoda zruší uzel v grafu, předaný pomocí ukazatele v parametru. Definuje uzel jako neplatný a vrátí jeho jeho pozici v poli.

4.3.4 Třída projectionRec

Třída zaobaluje práci s projekční rovinou. V jejím konstruktoru se inicializují vnitřní objekty a dojde k nastavení uzlu, který projekční rovinu představuje, pomocí atributu stavové množiny tak, aby nebyl propočítáván osvětlující model. Samotná rovina vzniká až ve chvíli, kdy se uživatel rozhodne ji definovat pomocí dvou bodů – levého horního a dolního pravého. V pravém slova smyslu nejde o rovinu ale jen o její část, kterou reprezentuje průhledný čtyřúhelník. Jedním z objektů třídy je objekt typu PointText, to znamená, že třída obsahuje v sobě uložené body s jejich souřadnicemi, které se po

9 Viz třída projectionRec. Kapitola 4.3.4

výpočtu ve třídě DistanAngle promítnou na rovinu. Toho se využije pří výpočtu vzdálenosti a úhlu mezi promítnutými body/polopřímkami.

Nejdůležitější metodou je **void projectionRec::createRec (osg::Vec3 pointA, osg::Vec3 pointB)**, která ze dvou definovaných bodů vytvoří čtyřúhelník, jenž znázorňuje a zároveň definuje rovinu. Čtyřúhelník je vytvořen následujícím způsobem: z vieweru se metodou osg::Matrixd& osg::CameraNode::getViewMatrix() zjistí pohledová matice kamery. Z této matice se pomocí metody void osg::Matrixd::getLookAt (osg::Vec3 & eye, osg::Vec3 & center, osg::Vec3 & up) zjistí souřadnice kamery – eye, bodu, do kterého pohled kamery směřuje – center a vektor, který udává, směr, kde je „strop“ scény. Z těchto tří informací metoda dopočítá vektor, který udává směr doleva – směr kolmý jak na vektor definovaný body center a eye (nazvěme jej vektor forward), tak na vektor udávající směr nahoru „ ke stropu scény“ (vektor up). Vektorovým součinem tedy získáme vektor udávající směr doleva (vektor left). V tuto chvíli tedy známe dva body a směrový vektor left. Je potřeba dopočítat velikost vzdálenosti levého horního bodu (známe) od pravého horního bodu (neznáme). Tu vypočítáme pomocí goniometrických funkcí, zjistíme odchylku[10] vektoru up a vektoru definovaného jako levý horní bod – pravý dolní bod (vektor AB). Kosinem tohoto úhlu vynásobíme velikost[11] vektoru AB a dostaneme velikost, kterou je potřeba vynásobit s vektorem udávající směr doprava (left * -1) a výsledek přičíst k hornímu levému bodu, získáváme pravý horní bod. Poslední, levý dolní bod, získáme stejným způsobem, jen velikost násobíme vektorem left. Stačí nadefinovat normály – ty jsou rovny vektoru forward, a projekční čtyřúhelník je hotov.

Obsaženy jsou zde metody pro manipulaci s rovinou a to pohyb po absolutních osách scény – ve směru x, y, z. Pohyb je realizován o pevně zvolenou vzdálenost 0.5, která je relativní k jednotkám použitým ve scéně, proto jsou vhodné pro použití pomocí klávesnice.

- void projectionRec::moveLeft(); Pohyb roviny po ose x + 0.5
- void projectionRec::moveRight(); Pohyb roviny po ose x - 0.5
- void projectionRec::moveUp(); Pohyb roviny po ose y + 0.5
- void projectionRec::moveDown(); Pohyb roviny po ose y - 0.5
- void projectionRec::moveFront(); Pohyb roviny po ose z + 0.5
- void projectionRec::moveBack(); Pohyb roviny po ose z - 0.5

Vytvořeny jsou také metody pro rotaci kolem těchto tří os, bohužel nejsou implementované a odladěné, proto, nejsou v projektu využity.

Důležitou metodou je **osg::Vec3 projectionRec::getNormal()**. Ta vynásobí transformační matici s normálou definovanou v uzlu typu osg::Geometry (v té je definována geometrie čtyřúhelníku, reprezentující rovinu) a výsledek vrátí. Pomocí této normály a bodu na tělese se později

10 Vzorec 12. Kapitola 2.1.5

11 Vzorec 3. Kapitola 2.1.2.1

definují přímky, na kterých se hledá průsečík s rovinou – promítnutí bodu na roviny. Využívá se zde vztahu vektor normály = transformační matice x původní vektor normály.

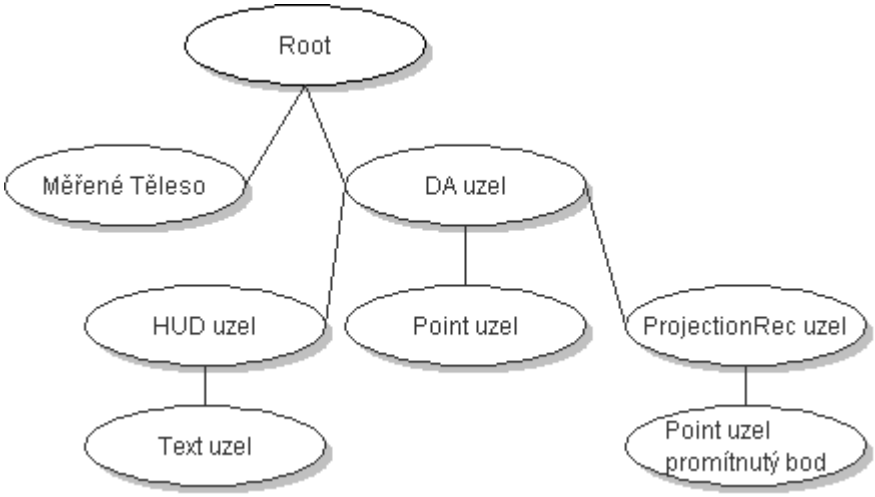
Je-li bod z tělesa promítnut na projekční čtyřúhelník, je jeho promítnutí znázorněno úsečkou mezi bodem na tělese a promítnutým bodem, o vykreslení a přidání do grafu scény se stará metoda void projectionRec::drawLine(osg::Vec3 a, osg::Vec3 b, unsigned int index). První dva její parametry definují počáteční a koncový bod, index potom udává pozici v poli LineArray.

4.3.5 Třída Hud

O zobrazení Head Up Display se stará třída Hud. Obsahuje metody pro připojení na HUD a odebrání z HUD. Jedná se o void addToHud (osg::Geode* geode), popřípadě void addToHud (osg::Group* group), jejich protipólem jsou metody void removeFromHud (osg::Geode* geode) a void removeFromHud (osg::Group* group) V jejím konstruktoru se vytvoří Head Up Display pomocí způsobu popsaného v kapitole 4.2.6.

4.4 Výsledná podoba grafu scény

Výsledná podoba grafu scény je popsána obrázkem 7:



Obrázek 7: Výsledná podoba grafu scény

Třída DistanceAngle obsahuje uzel DA, který se chová jako kontejner (je typu osg::Group). Uzel zajišťující transformaci HUD je připojen na DA uzel. Na samotný HUD uzel je připojen uzel použitý pro veškerý text, který se vypisuje na HUD. Point uzel reprezentuje definované body na tělese a je připojen na DA uzel. Posledním synem DA uzlu je ProjectionRec uzel, ten je v grafu scény reprezentován jako projekční čtyřúhelník. Body promítnuté na projekční čtyřúhelník jsou reprezentované uzlem Point uzel – promítnutý bod.

5 Závěr

Projekt, jenž si kladl za cíl implementovat měření geometrických parametrů 3D těles pomocí Open Scene Graph své splnil. Je pravdou, že zůstalo jen u jednoduchých metod měření, jak už bylo zmíněno v práci a to měření vzdálenosti na tělese, měření úhlů a jejich ekvivalenty promítnuté na projekční rovinu, zároveň ale položil slušné základy pro její eventuální další vývoj. Implementace by si určitě zasloužila sofistikovanější ovládání projekční roviny, na které už bohužel nevyšlo. Velikou výhodou a paradoxně současně i nevýhodou je jednoduchost implementace knihovny do jiných projektů. Při potřebě začlenit knihovnu pro měření vzdáleností a úhlů na tělese do programu, stačí programátorovi ji vytvořit a dodat potřebné parametry konstruktoru a o více se již nemusí starat. Na druhou stranu chybí jakákoliv možnost parametrizace knihovny jiným způsobem, než přímým zásahem do kódu. Možnosti parametrizace pomocí metod knihovny, by sice byli určitě efektivní, ale zapříčinilo by to nepříjemné nabobtnání zdrojového kódu.

Přínosem této práce je tedy určitě položení základu knihovny. Možnost relativně přesně měřit geometrické parametry těles ve smyslu vzdálenosti a úhlu. Povedenou funkcí je měření na projekční rovině, kde se podařilo jednoduše zvládnout problém, jak transformovat velikost a úhel z tělesa na uživatelem definovaný projekční čtyřúhelník.

Z hlediska dalšího vývoje by si knihovna zasloužila pokročilejší funkce jako například měření vzdálenosti mezi dvěma body po povrchu tělesa, měření vyznačené části nebo celého povrchu tělesa a stejně tak objemu. Z tohoto pohledu už ale nejde o zcela triviální operace. Každopádně je knihovna lehce rozšiřitelná a tudíž by stačilo jen naprogramovat způsob měření. Získávání informací pro jeho měření jako zadávání bodů, jejich výpis atd. je v knihovně vytvořen obecně a znovupoužitelně.

Literatura

[1] Ústav počítačové grafiky a multimédií, Fakulta informačních technologií. *OpenSceneGraph Tutoriál*, 2006.

[2] Sulivan J. *OpenSceneGraph Tutorials*. Dokument dostupný na URL http://www.nps.navy.mil/cs/sullivan/osgtutorials (květen 2007).

[3] Martz Paul. *OpenSceneGraph Quick Start Guide.* Louisville, CO 80027 USA, Skew Matrix Software LLC 2007, Dokument dostupný na URL http://www.lulu.com/content/767629 (květen 2007).

[4] OpenSceneGraph online dokumentace http://www.openscenegraph.org (květen 2007).

[5] Andes Computer engineering, *Open Producer – Tutorial.* Dokument dostupný na URL http://www.andesengineering.com/Producer/Tutorial/

[6] Merefield Rob, Brian Malloy. *3D Game Development Tutorials Using SDL and OSG.* Dokument dostupný na URL http://www.cs.clemson.edu/~malloy/courses/3dgames-2007/tutor/ (květen 2007)

[7] Tišnovský Pavel. *Grafická knihovna OpenGl.* Dokument dostupný na URL http://www.root.cz/serialy/graficka-knihovna-opengl/ (květen 2007)

[8] Fakulta metalurgie a materiálového inženýrství, Vysoká škola báňská, *Analytická geometrie.* Dokument dostupný na URL http://mdg.vsb.cz/FMMI/Html/Mat2/Pdf/analgeom.pdf (květen 2007).

Seznam příloh

Příloha 1. Manuál k používání knihovny DistanceAngle

Příloha 2. Zdrojová dokumentace knihovny DistanceAngle

Příloha 3. Zdrojové kódy knihovny DistanceAngle

Příloha 4. Ukázková aplikace použití knihovny DistanceAngle

Příloha 5. Dokumentace k Open Scene Graph 1.2