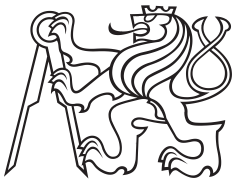
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Editor geometrie v systému VRUT
(Virtual Reality Universal Toolkit)

Daniel Aschermann

Školitel: doc. Ing. Jiří Bittner, Ph.D.
Květen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Aschermann** Jméno: **Daniel** Osobní číslo: **474486**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Editor geometrie v systému VRUT

Název bakalářské práce anglicky:

Geometry editor in VRUT

Pokyny pro vypracování:

Zmapujte způsoby editace geometrie v existujících modelovacích nástrojích (Maya, Blender) a herních enginech (Unity, UE4). Popište základní společné principy a rozdíly ve zmapovaných nástrojích a vyberte vhodnou sadu funkcí pro realizaci editoru geometrie v systému Virtual Reality Universal Toolkit (VRUT).
Vytvořte modul pro systém VRUT, který umožní interaktivní editaci geometrických transformací objektů. Po výběru objektu bude možné upravovat pozici, rotaci a měřítko vybraného uzlu. Zmapujte možnosti pro implementaci editačních operací nad vrcholy polygonálních sítí jako jsou úprava pozice, normály nebo texturovacích souřadnic. Implementujte alespoň jednu z těchto operací. Implementujte také režim, ve kterém bude možné vytvářet nové instance vybraného objektu kliknutím na zvolené pozice ve scéně.
Vytvořený modul otestujte na nejméně třech scénách různé složitosti. Realizujte základní studii použitelnosti s nejméně třemi uživateli.

Seznam doporučené literatury:

[1] Václav Kyba. Modulární 3D prohlížeč. Diplomová práce ČVUT FEL, 2008.
[2] Steve Marschner, Peter Shirley. Fundamentals of Computer Graphics. A K Peters/CRC Press 2015.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jiří Bittner, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

doc. Ing. Jiří Bittner, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání	Podpis studenta

Poděkování

Děkuji panu docentu Ing. Jiřímu Bittnerovi, Ph.D. za trpělivost, vstřícnost a vedení této práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci s názvem Editor geometrie v systému VRUT vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 10. května 2020

.....
podpis

Abstrakt

Tato práce se zabývá implementací modulu na úpravu geometrie do softwaru VRUT a obecným využitím a funkcionality softwaru VRUT. Mimo jiné ale zkoumá i jiné softwary podobných typů na úpravu geometrie a zabývá se rozdíly v jejich využití a daném rozhraní.

Klíčová slova: VRUT, editor geometrie, polygonální síť, transformace, manipulátor

Školitel: doc. Ing. Jiří Bittner, Ph.D.

Abstract

This thesis deals with module implementation and geometry editation in VRUT (Virtual Reality Universal Toolkit) software. Among other things it researches other similar softwares that operate with geometry and investigates their differences both in utilization and interface.

Keywords: VRUT, geometry editor, polygon mesh, transformation, manipulator

Title translation: Geometry editor for VRUT

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Cílem práce je implementace editoru geometrie v systému VRUT. Jedná se o modul umožňující snadnou práci s geometrií objektů. Mnoho jiných softwarů má podobné funkcionality a jsou tedy dobrou inspirací pro správné fungování a použitelnost.

Každá scéna se skládá z objektů. Tyto objekty jsou hmotnou částí scény, kterou zpravidla vnímá uživatel jako první. Někdy je potřeba upravovat jejich pozici, rotaci, nebo velikost. Pro snazší transformace objektů jsou proto v softwarech implementovány editory geometrie, které poskytují snadnou interaktivní práci s vybranými objekty. Navíc ale také v některých softwarech nabízí úpravu polygonových sítí daných objektů, tedy jejich geometrie. Tyto možnosti zatím žádný z modulů systému VRUT nenabízí, a proto by mohla jeho přítomnost v systému usnadnit práci jak uživatelům, tak i ostatním vývojářům.

Práce je strukturovaná následovně: první kapitola je úvodem do světa manipulátorů, softwarů, jejichž nedílnou součástí je editace geometrie pomocí manipulátorů, a jejich rozdíly, jak ve vzhledu, tak ve využití. Následně se práce zabývá samotným softwarem, k čemu slouží, jak se ovládá a vyvíjí. Tato sekce je důležitá především pro čtenáře neseznámené se systémem VRUT. Další kapitola pak popisuje implementaci a přístup k daným problémům, jejich řešení a případné varianty implementace. Poslední kapitola zmiňuje výsledky implementace a testování modulu.

Kapitola 2

Editor geometrie v ostatních modelovacích nástrojích

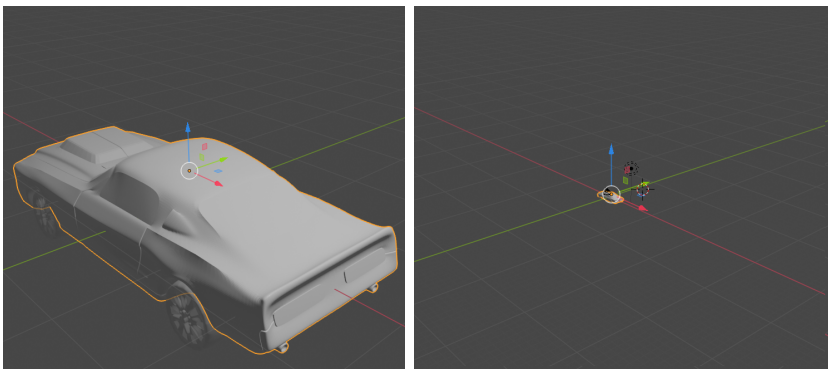
Editor geometrie a jeho využití se liší ve většině modelovacích nástrojích. Liší se hlavně na základě využití daných modelovacích nástrojů. Pro tuto práci a svoji inspiraci jsem si vybral nástroje Blender, Unity a Unreal Engine. Blender je méně podobný systému VRUT než ostatní dva softwary, ale zato prosperuje intuitivními editačními nástroji nad vrcholy polygonálních sítí. Unity a Unreal Engine jsou systému VRUT více podobné především proto, že jsou zaměřeny na práci s celou scénou. Blender se zaměřuje spíše na úpravu a modelování jednoho objektu.

2.1 Blender

Blender je open-source software pro modelování a vykreslování 3D počítačové grafiky a animací s využitím různých technik (např. sledování paprsku, radiosita, scanline rendering, GI). Vlastní interface je vykreslován pomocí knihovny OpenGL. Je to jeden z nejznámějších modelovacích nástrojů a i přes to, že se svým využitím systému VRUT moc neblíží, tak se stal velmi přínosným nástrojem pro vývoj modulu na úpravu geometrie a to nejenom jako inspirace pro jednoduchou ovladatelnost manipulátoru, ale i pro vymodelování jeho geometrie.

2.1.1 Blender - translační manipulátor

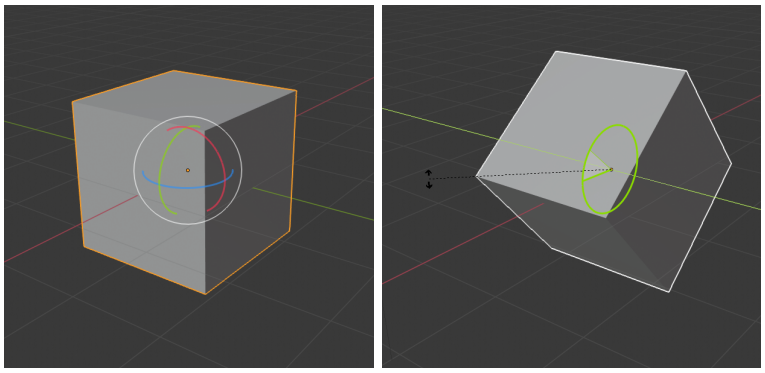
Blender má velmi intuitivní manipulátor. Kromě možnosti úpravy pozice objektů i v rovině si v Blenderu můžeme všimnout automatického zvětšování manipulátoru dle vzdálenosti od kamery tak, aby si zachovával stále stejnou velikost na obrazovce. Jedna z věcí, která vyčnívá především z hlediska uživatelské zkušenosti je to, že ve chvíli, kdy manipulátor uchopíme, manipulátor zmizí a místo něj se vykreslí ve scéně pouze přímka resp. přímky roviny, po kterých můžeme objekt posouvat. Manipulátor tak při přímé interakci nepřekáží ve scéně. Je pochopitelné, že v modelovacím softwaru, jako je právě Blender, minimalizovali vývojáři výskyt manipulátoru.



Obrázek 2.1: Blender - translační manipulátor; velikost manipulátoru závislá na vzdálenosti od kamery

2.1.2 Blender - rotační manipulátor

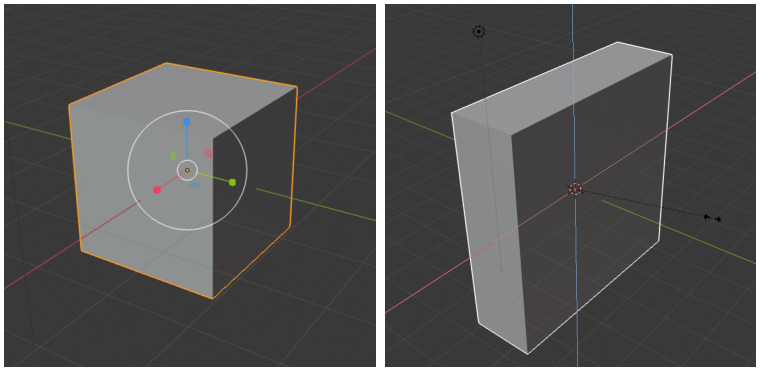
Rotační manipulátor je v Blenderu trochu odlišný tím, jak se vykresluje. Rotační transformace se zde upravují pouze globálně a manipulátor se neotáčí s transformovaným objektem. Podobně jako u translace manipulátor po nakliknutí zmizí a místo něj se pak vykresluje pouze kružnice, na které můžeme objektem otáčet. Na kružnici se pak vyplňuje výseč, která se barví různými stupni šedi podle toho, kolikrát jsme objekt zrotovali okolo dané osy.



Obrázek 2.2: Blender - rotační manipulátor; výseč ukazující danou rotaci

2.1.3 Blender - manipulátor měřítka

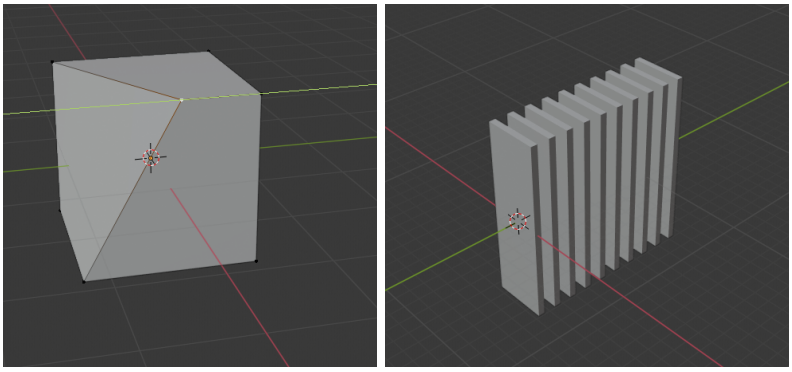
U scale manipulátoru je z hlediska vzhledu a chování řada způsobů a technik podobných tomu translačnímu. Důležitou věcí, kterou by měl ale takový manipulátor disponovat, je právě možnost zvětšování objektu samotného ve všech směrech prostoru najednou, a ne pouze v jednotlivých směrech os, nebo jejich rovin. Právě taková funkce by měla mít v systému jako je VRUT velké využití.



Obrázek 2.3: Blender - manipulátor měřítka; při interakci se manipulátor nevykresluje

2.1.4 Blender - editace vrcholů polygonální sítě

Blender jakožto nástroj pro modelování přináší mnoho možností v oblasti vytváření a samotné editace vrcholů polygonální sítě. Od velmi využívané funkce *extrude* až po nástroj *loop tool*, který přináší plno možností pro vytváření nových objektů jedné instance. Blender jako jediný z vybraných modelovacích nástrojů disponuje možností úpravy vrcholů polygonální sítě, normál etc., a proto se stal svojí praktičností a použitelností skvělou inspirací pro implementaci modulu na úpravu geometrie v systému VRUT.



Obrázek 2.4: Blender - úprava pozice vrcholu polygonální sítě objektu; loop tool

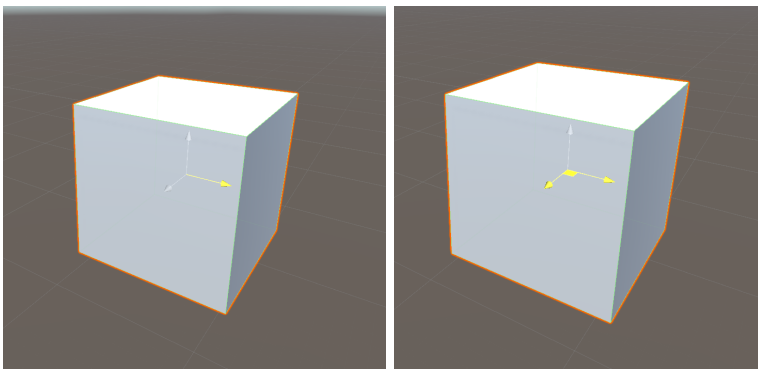
2.2 Unity

Unity je multiplatformní herní engine vyvinutý společností Unity Technologies. Byl použit pro vývoj her pro PC, konzole, mobily a web.
Unity poskytuje možnosti vývoje pro 2D i 3D hry libovolného žánru a zaměření. Kromě grafického prostředí pro tvorbu, podporuje také tvorbu skriptů především v jazyce C# a UnityScript.

Tento software je systému VRUT o něco blížší; manipulátor zde není to, okolo čeho by se celý software točil, je zde mnoho jiných funkcionalit od vytváření menu pro hru přes hratelnost po nastavení koeficientu tření vybraného objektu.

2.2.1 Unity - translační manipulátor

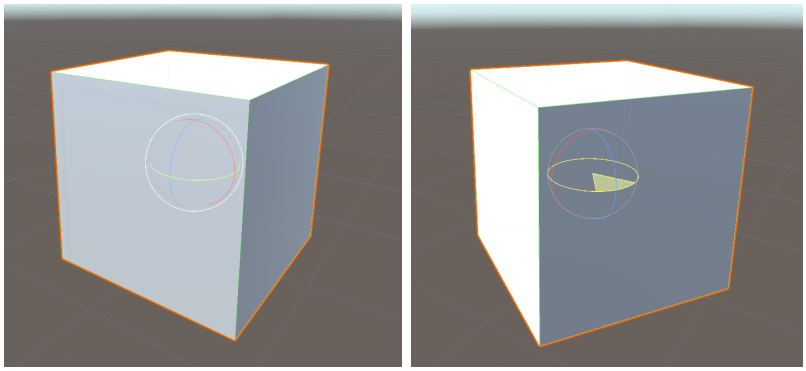
Vzhledem k tomu, že Unity není modelovací nástroj (jako Blender), tak se zde při interakci manipulátor nijak neschovává (nevadí, když překáží chvíli ve scéně). Pouze se změní materiál daného vektoru a zbytek geometrie manipulátoru zešedne. Takový manipulátor je trochu jednodušší na implementaci a vzhledem ke skutečnosti, že je softwaru VRUT využitím bližší než Blender, tak se modul pro úpravu geometrie v systému VRUT blíží technikami a způsoby zacházení spíše Unity. Translační manipulátor zde umožňuje, stejně jako v Blenderu, posun v rámci roviny určené dvěma vektory.



Obrázek 2.5: Unity - translační manipulátor; změna barvy geometrie při interakci

2.2.2 Unity - rotační manipulátor

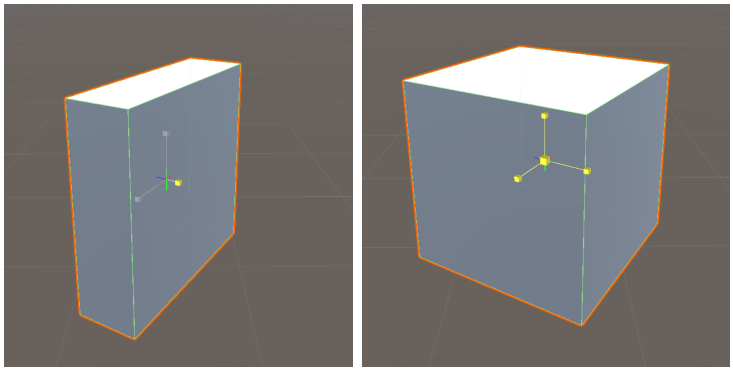
Rotační manipulátor se obdobně, jako translační manipulátor v Unity nijak neskrývá při interakci s jednotlivými oblouky, stejně jako v Blenderu, se vyplňuje výseč kružnice dle daného otočení. Navíc umožňuje rotovat daným objektem v rámci pohledu kamery, při čemž je potřeba trochu výpočetně náročnější metoda projekce na tzv. trackball.



Obrázek 2.6: Unity - rotační manipulátor; výseč značící velikost rotace

2.2.3 Unity - manipulátor měřítka

Scale manipulator je podobný translačnímu; změna materiálu geometrie, se kterou interagujeme na žlutou a ostatní geometrie na šedou, je obdobná. Zde ale rozhodně stojí za zmínku "krychlička"v jeho středu, která umožňuje změnu měřítka celého objektu, tedy ve všech směrech. Taková funkce má pravděpodobně větší využití než ostatní změny měřítka pouze v jeden směr. Další, pro uživatele samozřejmá věc je, že kdykoliv je manipulátor upuštěn, srovná se do svých původních rozměrů a zachovává tak velikost všechna geometrie manipulátoru.



Obrázek 2.7: Unity - manipulátor měřítka; transformace manipulátoru s objektem

2.3 Unreal Engine

Unreal Engine je herní engine, který byl vytvořen firmou Epic Games. Původně byl tento engine určen pouze pro střílečky z pohledu první osoby, ale našel využití i v některých MMORPG, RPG a adventurách.

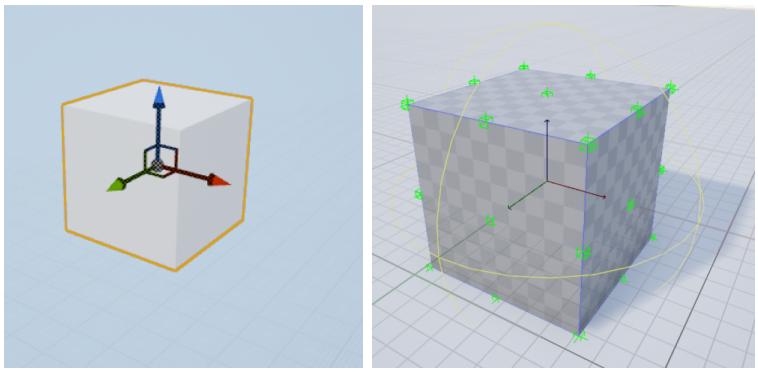
Jádro Unreal Enginu, napsané v programovacím jazyku C++, podporuje mnoho různých platforem, jako jsou Microsoft Windows, Linux, Mac OS a Mac OS X na PC.

Využitím se Unreal velmi podobá Unity, a proto si také tyto dva enginy konkurují. Ovšem každý z těchto softwarů vyniká v jiných rysech a podle toho, jaké požadujeme od vyvíjené hry vizuální efekty, cílovou platformu nebo jaká je například velikost vývojářského týmu, by se mohlo vyplatit pracovat právě v jednom z těchto enginů.

2.3.1 Unreal - translační manipulátor

Jak je uvedeno výše, Unreal Engine je velmi podobný Unity. V případě translačního manipulátoru se technikou a mechanikami moc neliší. Rozdíl je především v geometrii manipulátoru a to především v té, která umožňuje translaci objektu v rovině. Část manipulátoru, která je uvnitř tělesa, je šrafovaná a manipulátor se tak zdá být skoro jednobarevný. Pokud je ale část manipulátoru vně, tak je geometrie barevná klasicky; pak se také materiál geometrie manipulátoru, se kterou právě interagujeme, změní na žlutou.

Navíc také Unreal Engine nabízí ve speciálním okně zobrazení normál, vrcholů atd.

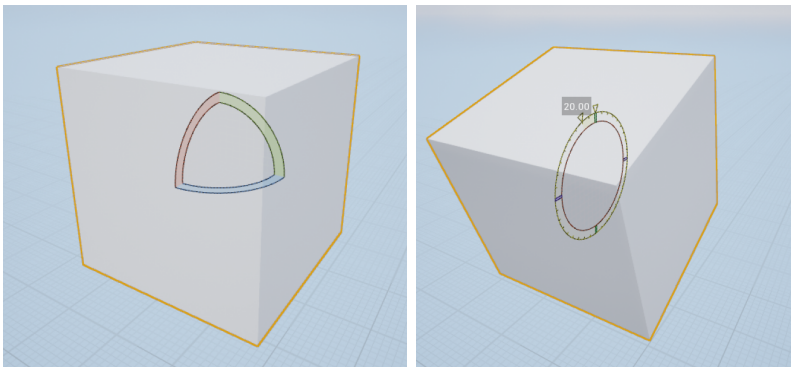


Obrázek 2.8: Unreal - translační manipulátor; zobrazení vrcholů objektu

2.3.2 Unreal - rotační manipulátor

Rotační manipulátor se na druhou stranu velmi liší od ostatních testovaných softwarů. Jak jeho geometrie ve stavu, kdy je nehybný, tak ve stavu, kdy s ním uživatel interaguje. V nečinném stavu se vykresluje v podstatě pouze 1/8 koule, dle jejíhož středu můžeme rotovat vybraným tělesem. Při vybrání geometrie manipulátoru, podobně jako v Blenderu, zmizí celý manipulátor

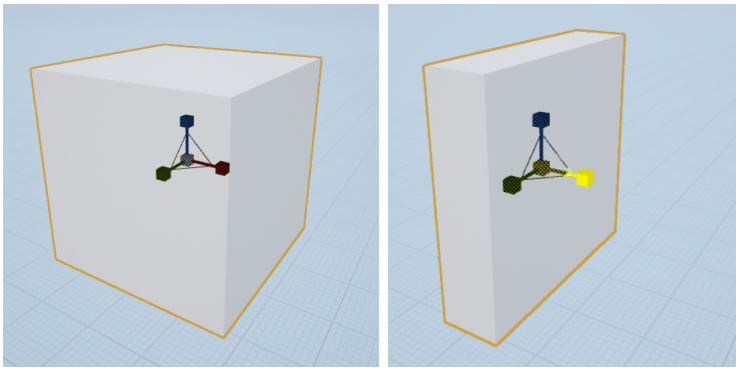
a zobrazí se nám pouze oblouk, podle kterého můžeme s objektem manipulovat. U oblouku se pak zobrazují pro lepší přesnost i stupně úhlu dané rotace.



Obrázek 2.9: Unreal - rotační manipulátor; při interakci se vykresluje pouze kružnice značící velikost rotace

2.3.3 Unreal - manipulátor měřítka

Manipulátor měřítka se od toho v Unity také tolik neliší. Jediná věc kterou se manipulátor měřítka odlišuje od toho v Unity je, že geometrie manipulátoru se nijak netransformuje, mění se pouze materiál vybrané geometrie.



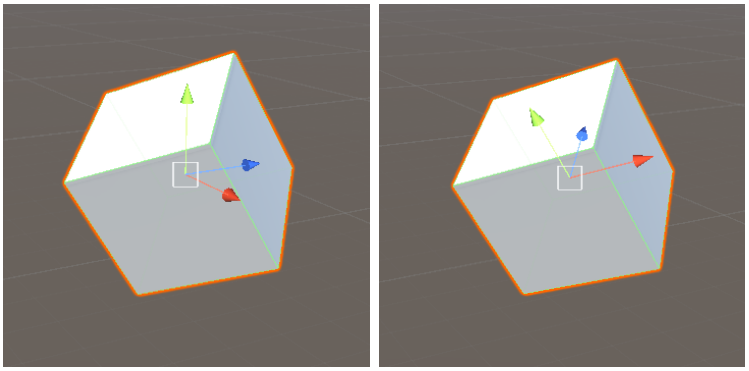
Obrázek 2.10: Unreal - manipulátor měřítka; uvnitř a vně objektu je jinak zbarvený; manipulátor se netransformuje s objektem

2.4 Lokální/globální souřadnice

Všechny uvedené softwary navíc umožňují přepínat mezi úpravou v globálních a lokálních souřadnicích. Přináší to více způsobů a možností do světa editorů geometrie. Například v Unity je možné mezi těmito možnostmi přepínat v horním panelu aplikace (viz. obrázek 2.11).

Obrázek 2.11: Unity: možnost přepínat mezi lokálními/globálními souřadnicemi (tlačítko úplně vpravo)

Transformace v rámci lokálních souřadnic se od té v globálních liší tím, že bere v potaz rotaci transformovaného tělesa a pak například při změna měřítka nemůže nikdy dojít ke zkosení, protože všechny transformace zachovávají objekt v pravoúhlých tvarech.



Obrázek 2.12: Unity: globální souřadnice (vlevo), lokální souřadnice (vpravo)

2.5 Vlastní zkušenosti

Se všemi uvedenými softwary jsem přišel již do styku, ale při dokumentaci každého z nich jsem narážel na mnoho věcí, které stály za implementaci i v editoru geometrie systému VRUT.

Nejvíce jsem ovšem pracoval s Blenderem a to díky předmětu VGO (vytváření grafického obsahu) a s Unity díky předmětu HRY (Počítačové hry).

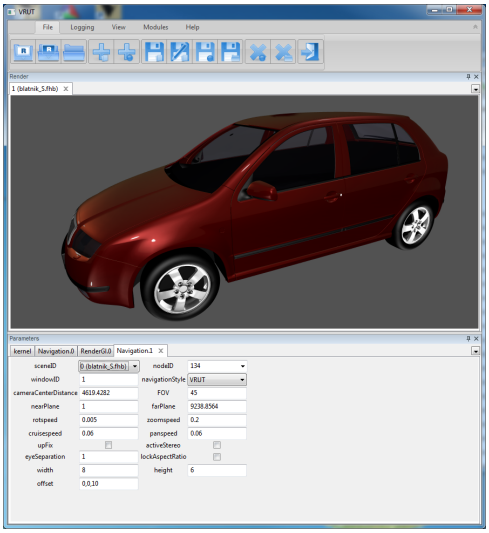
V Unreal Enginu jsem dělal pouze krátce při spolupráci s katedrou počítačové grafiky. Samotný Blender se nedá s Unity a ani s Unrealem moc srovnávat. Za zmínku snad stojí jen odlišnost geometrie manipulátoru a jeho chování. Dalo by se skoro říct, že Blender spíše funguje dobře s uvedenými enginy.

Blender slouží pouze k vymodelování objektů, které můžeme pak importovat do scén enginů, kde pak můžeme řešit například herní logiku atd., a dále jejich polygonální síť, normály atp. už neměníme, nebo na to enginy ani nejsou uzpůsobené. Blender má totiž z hlediska modelování mnohem intuitivnější prostředí a nabízí mnohem více nástrojů na takové úpravy, obdobně jako Maya, 3DSMax a další.

Kapitola 3

Aplikace VRUT

VRUT je aplikace pro vizualizaci a editaci 3D dat vytvářená ve spolupráci ŠKODA AUTO a.s., ČVUT, MU, MeU, ZCU a VŠB.



Obrázek 3.1: VRUT - ukázka pracovního prostředí aplikace

Projekt s názvem VRUT vznikl v rámci spolupráce katedry počítačové grafiky a interakce ČVUT FEL s firmou Škoda Auto. Jeho podstatou je zobrazení grafických dat a podpora modulů. Moduly umožňují rozšíření funkcí hlavní aplikace při relativní nezávislosti na ní.

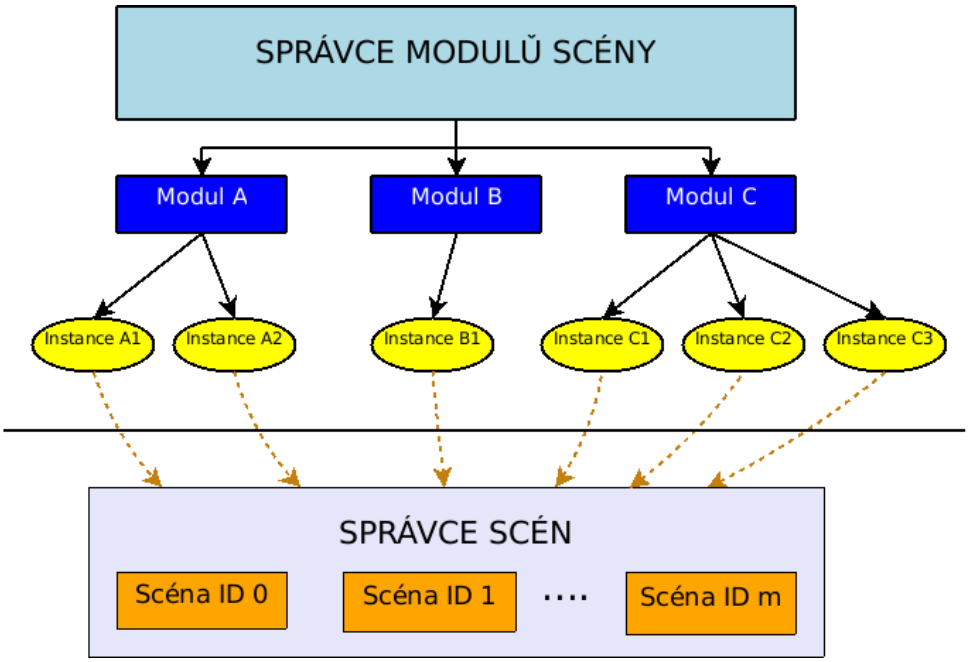
Název VRUT je zkratkou anglického *Virtual Reality Universal Toolkit*.

3.1 Sestavení a kompilace VRUT

VRUT je možné zkompilovat mnoha způsoby a v mnoha konfiguracích. Na kompilaci ve Windows je používán nástroj CMAKE. Pro základní kompilaci jsou potřeba knihovny wxWidgets a GLEW, pro další různé moduly je ještě potřeba knihovny ODE, VulkanSDK, GLUT, CUDA, ALUT a OPENGL.

3.2 Jak VRUT funguje?

Hlavní a nezbytnou součástí VRUTu je *Jádro*, které představuje hlavní část aplikace. Jádro poskytuje prostředky modulům systému a ty s jádrem komunikují. Ke každé ze spuštěných scén se vytváří nová instance modulu.



Obrázek 3.2: Instancování modulů do scén v systému VRUT

Každý modul pak představuje jinou funkční část aplikace. Modul GeometryEditor, jehož implementací se zabývá tato práce, neimplementuje například pohyb kamery ve scéně – to implementuje modul Navigation. Proto je skoro nezbytné mít k modulu GeometryEditor vždy ve scéně i instanci modulu Navigation atp.

3.3 Uživatelské rozhraní

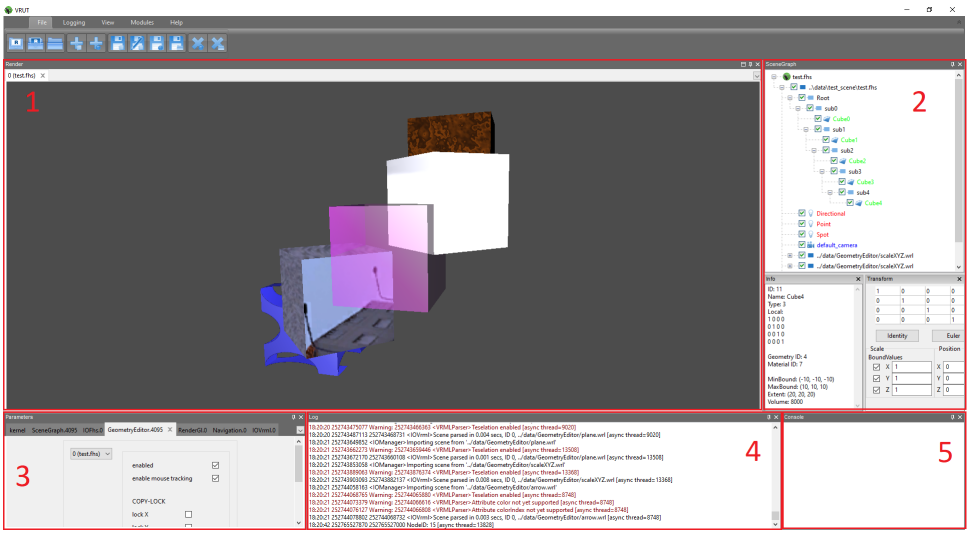
UI je v systému VRUT velmi snadno upravitelné, některé moduly mají pro různé účely i svá vlastní okna, ze kterých si můžeme postavit vlastní UI. Hlavním oknem je zde *Render*, kde se zobrazuje aktuální scéna. Dalšími nezbytnými pomocníky jsou okna *Parameters*, *LOG* a *Console*.

Okno *Parameters* nabízí měnit různé proměnné modulů, například jejich aktivní scénu. Okno *LOG* je výstup aplikace, jehož úroveň výstupu lze nastavit pomocí příkazu *setloglevel* v posledním nepostradatelném okně *Console*.

Ovšem místo vypisování řádek do *Console* při každém spuštění programu je lepší použít soubor *autoexec.cfg*. Pro testování modulu GeometryEditor jsem použil následující konfiguraci a následující UI (viz. obrázek 3.3), které

je možné si libovolně nakonfigurovat a následně uložit jedním z tlačítek v horním panelu aplikace.

```
setloglevel messages     <- nastaveni urovne vystupu

usemodule Navigation      <- inicializace modulu
runmodule sceneGraph
usemodule rendergl

importscene "..\data\test_scene\ssdoTest\colorTestScene.wrl"
<- import sceny
#importscene "..\data\test_scene\ssdoTest\colorBounce.wrl"
#importscene "..\data\test_scene\ssdoTest\city.wrl"
#importscene "..\data\test_scene\ssdoTest\testScene05.wrl"
#importscene "..\data\test_scene\ssdoTest\temple.wrl"
#importscene "..\data\Tracks\dalnice_new\dalnice.vrut"

setparam Navigation.farPlane 10000000

runmodule geometryEditor     <- spusteni modulu GeometryEditor
renderscene 0                <- vykreslení sceny s indexem 0
setparam SceneGraph.sceneID 0
<- nastaveni instance modulu pro scenu s indexem 0
setparam GeometryEditor.sceneID 0
#renderscene 1

#colorTestScene - info nastaveni
NearPlane: 100, Radius: 100
```



Obrázek 3.3: VRUT - popis uživatelského rozhraní;
1 - renderovaná scéna; 2 - graf scény; 3 - karty aktuálně spuštěných modulů;
4 - LOG, výstup aplikace; 5 - konzole

3.4 Moduly systému VRUT

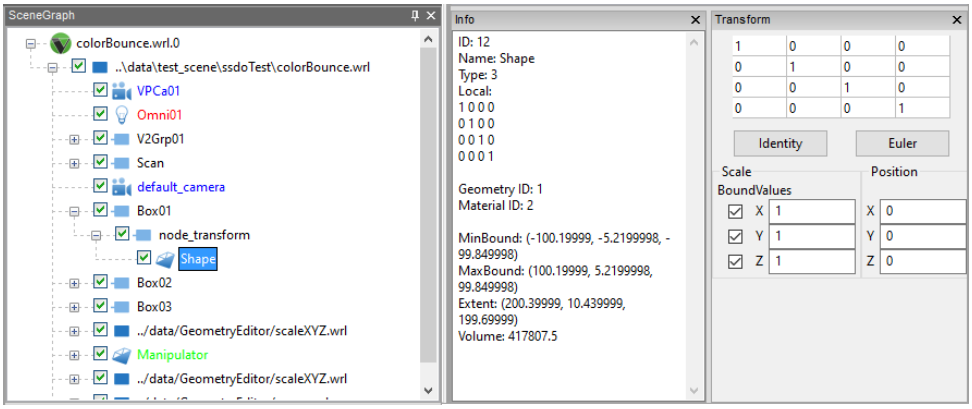
Systém VRUT je stále vyvíjející se aplikací a proto je nezávislost modulů opravdu velkou výhodou pro budoucí rozvoj. Množství modulů, které nabízí, je obrovské. Mezi nimi jsou některé, co si zaslouží být v této práci vyzdvihnuty, protože bez nich by byla implementace modulu na úpravu geometrie naprosto zbytečná.

3.4.1 Modul Navigation

Modul navigation, jak je zmíněno výše, umožňuje pohyb ve scéně. Na veškerý pohyb je v podstatě potřeba jen myš. Pomocí levého tlačítka myši se otáčíme okolo vybraného pivota, kterého lze kdykoliv změnit dvojitým kliknutím kolečkem myši. Levým tlačítkem se pak můžeme posouvat po rovině dané pohledovým vektorem kamery. Přibližování a oddalování od scény funguje jako snad úplně ve všech softwarech na bázi kolečka myši. Karta modulu nabízí neskutečné množství nastavitelných parametrů, jako například FOV (Field of View), rotation/zoom speed, far/near-plane atd.

3.4.2 Modul SceneGraph

Graf scény a správná práce s ním představuje část implementace modulu na úpravu geometrie. Modul SceneGraph vybrané scény zobrazuje graf scény, uzly scény, jejich ID, transformační matice atd. a nabízí škálu operací nad vybranými uzly. Modul sice není potřebnou pomůckou při práci s geometrií a její úpravou, ale byl nezbytnou pomůckou při implementaci modulu na úpravu geometrie v systému VRUT.



Obrázek 3.4: Modul SceneGraph zobrazující ID, matici atd. vybraného uzlu

Kapitola 4

Popis implementace

Tato kapitola popisuje implementaci celého modulu na úpravu geometrie v systému VRUT. Implementace se skládá z několika podúloh a funkcí. Zatím se modul skládá pouze ze 4 souborů, *GeometryEditor.cpp/.h* a *Manipulator.cpp/.h*. V systému VRUT je možnost nadefinovat i rozložení karty modulu, a to se implementuje v souboru *GeometryEditorLayout.xrc*.

4.1 Základní komponenty

Tato sekce popisuje všechny kroky pro vytvoření jakéhokoliv modulu v systému VRUT. Jsou to základní postupy a potřebné funkcionality pro většinu modulů.

4.1.1 Vytvoření modulu

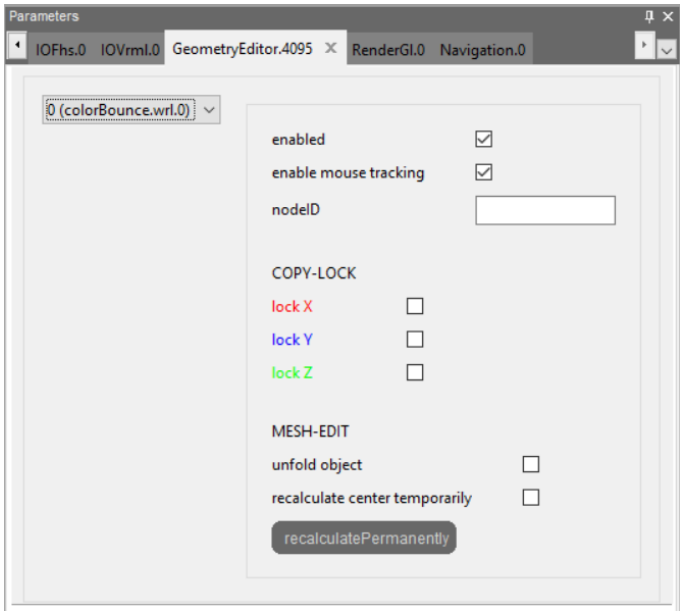
Na vytvoření modulu jsem se nechal inspirovat modulem LightEditor, jedním z jednodušších modulů ve VRUT. Není komplexní a skládá se jen z jednoho souboru. Modul bylo potřeba přidat do souboru *CMakeLists.txt* ve složce modules a znovu pomocí nástroje CMAKE sestavit VRUT. Pak už se práce odehrává pouze ve *Visual Studiu*.

4.1.2 Zachycení událostí

Jádro informuje všechny moduly o nových událostech a je již na jednotlivých modulech, jaké události odchytí. Nejprve je potřeba implementovat funkci *processEvent(Event *evt)*, která tyto události zachycuje. V modulu GeometryEditor to jsou jen události myši a pár kláves. Modul zachycuje i změny stavu checkboxů na jeho kartě v aplikaci a aktualizuje pak tyto hodnoty, jako například *mouseTracking*, který povoluje/zakazuje sledování událostí myši, *enabled*, který povoluje/zakazuje veškerou funkcionalitu modulu atd.

4.1.3 Karta modulu

Karta modulu je nakonfigurována v souboru *GeometryLayout.xrc (XML resources)* a nabízí mnoho nastavitelných parametrů, o jejichž využití se dočtete dále v této práci.

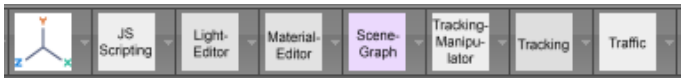


Obrázek 4.1: Karta modulu GeometryEditor

4.1.4 Ikona modulu

Každý z modulů má přiřazenou ikonu, která se zobrazuje v horním panelu aplikace. Obrázek ikony konvertujeme do souboru **.h* za pomoci příkazu *bin2c* v systému VRUT. Tento soubor pak musíme zahrnout v *mainwindow.cpp*.

```
#include "../data/resources/modules/GeometryEditor.png.h"
ADDICONTOMAP(GeometryEditor);
```



Obrázek 4.2: Ikony modulů systému VRUT (editor geometrie úplně vlevo)

4.2 Inicializace

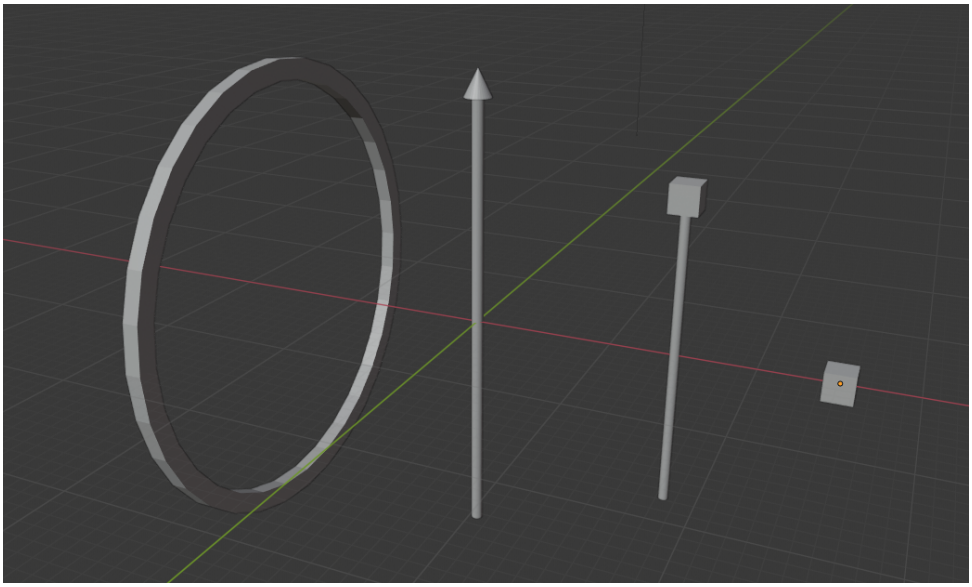
Veškerá inicializace probíhá ve chvíli, kdy se zachytí událost o změně scény. Veškeré objekty a materiály se pak naimportují do nově vybrané scény a ze scény původní se odstraní.

4.2.1 Vložení jednoduchého objektu do scény

K vložení jednoduchého objektu do scény může posloužit funkce *drawGraphNode()*, která na dané pozici vykreslí geometrii kvádru o zadaných rozměrech. K testování správné funkčnosti výběru objektu sloužilo právě vkládání jednoduchého objektu na pozici průsečíku paprsku s objektem. Později pak byla funkce *drawGraphNode()* nahrazena jednodušší funkcí *loadObject(filename, pos, root)*. Ta ovšem potřebuje již hotový objekt ve formátu **.wrl*. Z Blenderu jsem exportoval objekty do formátu *.stl* a pak pomocí online converteru převedl do *.wrl*.

4.2.2 Vložení složitého objektu do scény

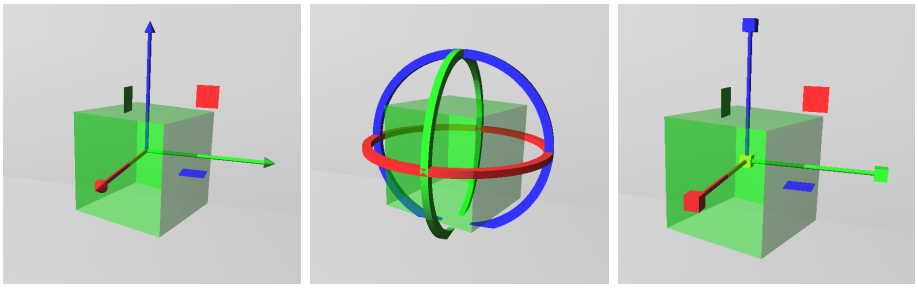
Vložení složitějšího objektu do scény bylo z hlediska kódu výrazně jednodušší. Import je implementován ve funkci *loadObject(filename, pos, root)* za použití příkazů *GET_EVT_IO_SCENE_IMPORT*, který zajistí import objektu do scény, a příkazu *GET_EVT_IO_SCENE_IMPORT_DONE*, který zajistí čekání na dokončení importu. Geometrie manipulátoru je vymodelována v Blenderu a exportována do *.wrl*.



Obrázek 4.3: Geometrie manipulátoru (zleva): rotační manipulátor, translační manipulátor/vektor normál, manipulátor měřítka, vrchol polygonální sítě objektu/manipulátor měřítka

4.2.3 Typy implementovaných manipulátorů

Obdobně jako v ostatních softwarech jsou i v systému VRUT implementovány tři manipulátory. Translační, na změnu pozice objektu, rotační, na změnu rotace/otočení daného objektu a manipulátor měřítka pro možnost zmenšení/zvětšení objektu ve scéně.



Obrázek 4.4: Manipulátory v systému VRUT (zleva); translační manipulátor, rotační manipulátor, manipulátor měřítka

4.2.4 Inicializace manipulátoru

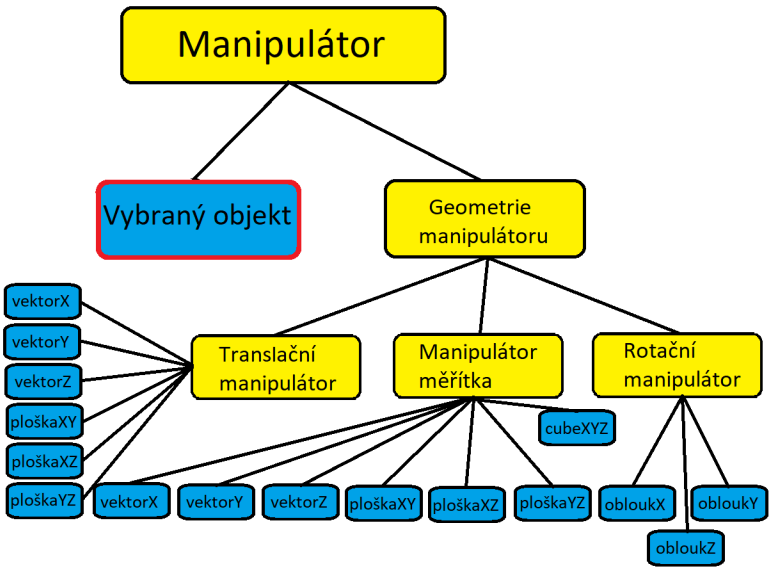
Způsobů chování manipulátoru v grafu scény bylo v návrhu hned několik:

1. Objekt a manipulátor zvlášť
2. Objekt dítětem jednoho z manipulátoru na úrovni geometrie manipulátoru
3. Objekt dítětem jednoho z manipulátoru nad úrovní geometrie manipulátoru
4. Objekt dítětem manipulátoru nad úrovní geometrie všech manipulátorů

Z těchto možností je nakonec využita poslední hned z několika důvodů. Tento přístup nabízí velmi snadné přepínání mezi manipulátory bez toho, aniž by ve scéně docházelo k problikávání vybraného objektu. Nastavuje se totiž jen vizibilita geometrie různých manipulátorů a objekt tak zůstává neměnný.

Uzel veškeré geometrie manipulátoru je transformován v případě, že pozice těžiště geometrie vybraného tělesa neodpovídá jeho světovým souřadnicím. Tento parametr má uživatel možnost měnit na kartě modulu *(recalculate center temporarily)*.

Navíc když chceme aby při oddálení kamery zachovával manipulátor konstantní velikost na obrazovce, stačí transformovat pouze uzel, ve kterém se nachází veškerá geometrie manipulátoru. V případě metody č. 2 bylo potřeba matici právě vybraného objektu násobit inverzí dané matice na změnu měřítka.



Obrázek 4.5: Struktura manipulátoru

4.3 Selekce objektu

V modulu GeometryEditor jsou implementovány dva způsoby výběru. První, intuitivnější, je vyvolán kliknutím kolečkem myši a druhý, méně intuitivní, pomocí *Textboxu* na kartě modulu. To nabízí navíc možnost transformovat ve scéně i uzly bez geometrie.

4.3.1 Výběr objektu

Výběr objektu je implementován ve funkci *select(evt, addToSelection)* a je vyvolán událostí myši, kliknutím kolečkem myši, kde se zjistí ID objektu pomocí RayCastu. Pro ten je ovšem potřeba zjistit projekční matici, transformační matici kamery, souřadnice myši na obrazovce atd.

```
int x = evt.GetX();
int y = evt.GetY();
Matrix pm = scene->GetProjectionMatrix(camera->GetID(),
aspectRatio, false);
Matrix camWorldMat(scene->GetWorldTransMatrix(camera->GetID()));
```

Objektu je pak, podle jeho identifikátoru přiřazena ve vektoru `ignoreList` hodnota *true*:

```
ignoreList[selectedObjectNode->GetID()] = true;
```

Když je `ignoreList` přeposlán jako jeden z parametrů funkci *CastRay(ray, ignoreNodes(optional), info)*, pak jeho elementy bude paprsek ignorovat. Jednoduše to pak nabízí možnost interagovat i s vektory manipulátoru, které mohou být uvnitř objektu.

Vybraný uzel je pak přeposlán třídě *Manipulator*, funkci, která zajistí přesun manipulátoru do těžiště vybraného objektu.

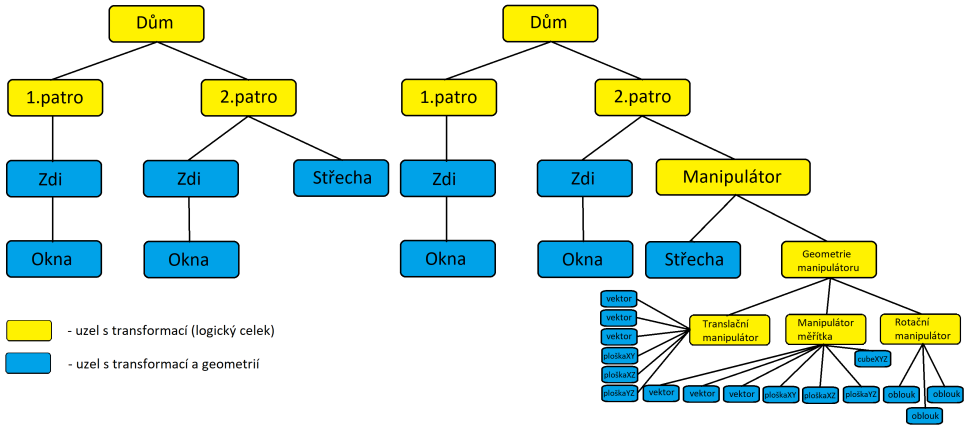
```
Manipulator::moveManipToSelectedObjectPos(node){}
```

4.3.2 Práce s grafem scény

Graf scény má zde velké využití. Pro snazší aplikaci transformace jsou vytvořeny pro každý z manipulátorů v grafu scény uzly, jejichž potomky jsou už uzly s geometrií daného manipulátoru. Rodičovský uzel manipulátoru se pak po výběru konkrétního objektu ve scéně stane potomkem rodiče vybraného objektu a vybraný objekt se stane potomkem rodičovského uzlu manipulátoru. Na přemísťování uzlů v grafu scény se používá příkaz *GET_EVT_SCENE_NODE_MOVE(sceneID, srcID, targetID)*.

Veškeré transformace jsou aplikovány na rodičovský uzel manipulátoru. Při deselekci vybraného objektu, nebo při výběru objektu jiného, se transformuje původně vybraný objekt lokální maticí manipulátoru, která v sobě nese veškeré doposud aplikované transformace nad daným objektem za pomoci manipulátoru.

Tato funkcionalita je implementována ve funkci:



Obrázek 4.6: Chování grafu scény po vybrání objektu; původní scéna (vlevo); graf scény po vybrání objektu k transformaci (vpravo)

```
Manipulator::moveManipToSelectedObjectPos(node){}
```

V případě, že se chceme manipulátoru zbavit, stačí dvakrát kliknout kamkoliv ve scéně nebo zmáčknout klávesu *Enter*. Tím se zavolá výše uvedená funkce s hodnotou `node = NULL`, která uloží pozici daného objektu a nastaví hodnotu vizibility rodičovského uzlu manipulátoru na *false*.

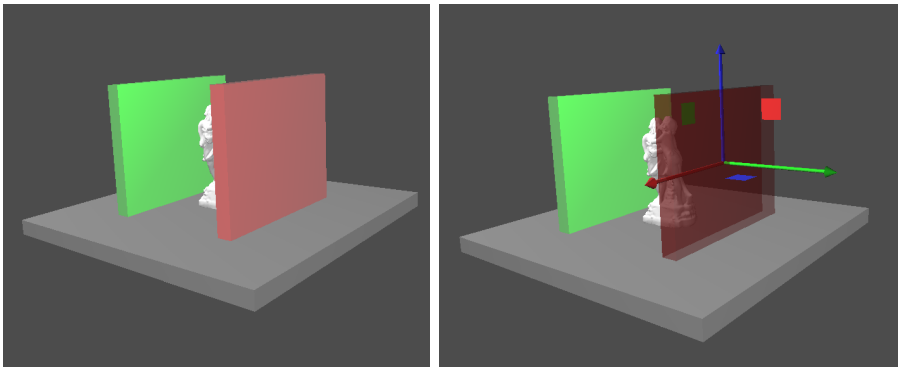
```
scene->SetActive(this->ID, false);
```

Další implementovaná funkcionalita je vyvolána stiskem klávesy *Escape*, která nastaví transformační matici vybraného objektu na matici původní, tedy před transformací. Jednoduše se uloží transformační matice manipulátoru po výběru objektu do proměnné `beforeObjectMAT` a v případě zmáčknutí klávesy *Esc* se vykoná následující kód:

```
scene->SetTransformation(this->ID, beforeObjectMAT);
moveManipToSelectedObjectPos(NULL);
```

4.3.3 Změna materiálu vybraného objektu

Vybranému objektu se změní materiál na nový, který má *flags* nastavený na *BLENDED*, *reflectivity (odrazivost)* na *0.6* a složky *diffuse* a *specular* na původní hodnoty materiálu vybraného objektu. Ten je tedy po vybrání průhledný, což zajistí, aby byly vidět ostatní objekty scény za vybraným objektem, včetně geometrie manipulátoru uvnitř vybraného objektu.



Obrázek 4.7: Změna materiálu vybraného objektu

4.3.4 Globální transformace

Modul GeometryEditor umožňuje pouze globální transformace objektů. K tomu, aby byl manipulátor vždy správně orientovaný, slouží funkce:

```
Manipulator:inverseManipMatrix(matrix, manipulator){},
```

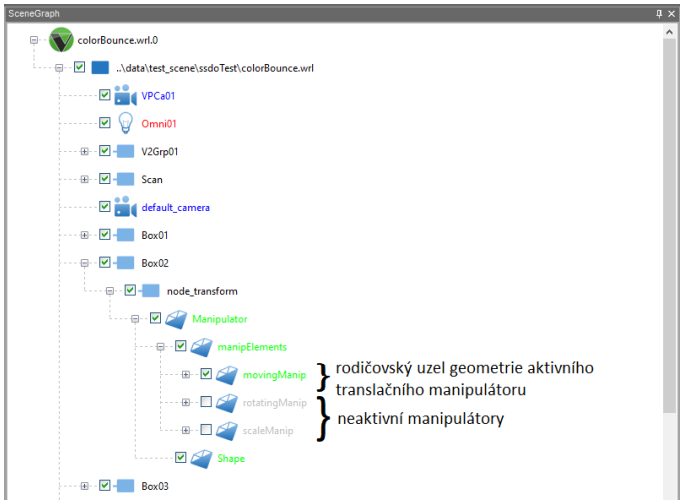
která násobí jednotlivé složky geometrie manipulátoru inverzí modelovací matice vybraného objektu.

4.4 Využití SceneGraph

SceneGraph je modul systému VRUT, který zobrazuje vždy aktuální stav vybrané scény. Pro správnou implementaci editoru geometrie je potřeba sledovat chování grafu scény v každém momentu interakce jak s objekty, tak i s manipulátorem samotným.

4.4.1 Volba manipulátoru (ovládání)

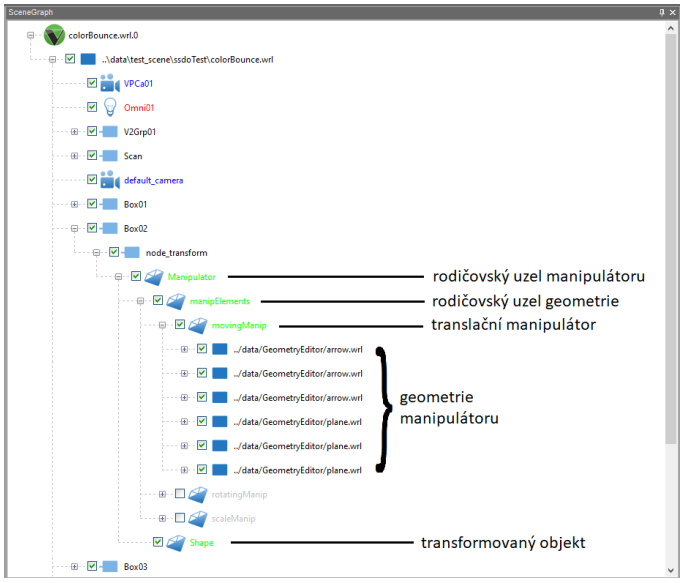
Pro jednodušší práci s manipulátorem a úpravu objektů je ovládání v systému VRUT dost podobné softwaru Blender, který pomocí kláves G (grab), R (rotate) a S (scale) přepíná mezi jednotlivými manipulátory. Díky zjednodušené implementaci stačí pouze přepínat vizibilitu jednotlivých rodičovských uzlů geometrie manipulátoru.



Obrázek 4.8: Chování grafu scény při změně manipulátoru

4.4.2 Zachování měřítka manipulátoru na obrazovce

Jedna z dalších věcí, která se vyskytuje ve všech testovaných softwarech, je zachování velikosti manipulátoru na obrazovce. Zde je implementace poměrně jednoduchá, při oddálení od objektu nebo při přiblížení k němu se volá funkce *checkSizeManip()*, která změní matici rodičovského uzlu geometrie manipulátoru dle vzdálenosti kamery od vybraného objektu. Bylo proto ale třeba pozměnit chování v grafu scény.



Obrázek 4.9: Aby manipulátor zachovával velikost na obrazovce, mění se pouze transformace rodčovského uzlu geometrie

4.5 Interkace s manipulátorem

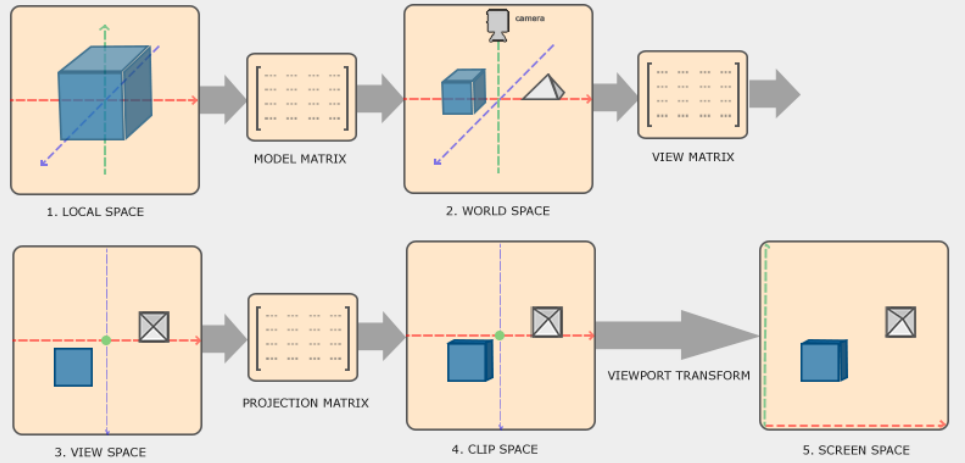
Pro intuitivní a správné chování manipulátoru při interakci s ním je potřeba nějakým způsobem převést vektor určený pohybem myši na obrazovce do světových souřadnic scény, nebo naopak zjistit směrový vektor manipulátoru a ten promítnout do tzv. *screen space (souřadnice obrazovky)* [x,y]. Jde tedy o promítnutí z 3D prostoru na 2D rovinu. V obou případech se jedná o projekci vektoru prvního do prostoru určeného vektorem druhým. Pro implementaci v systému VRUT je zvolena druhá metoda, projekce do souřadnic obrazovky.

4.5.1 Projekce do souřadnic obrazovky

V původním návrhu byla transformace závislá pouze na horizontální souřadnici myši, což udávalo potřebnou hodnotu pro dočasné testování modulu. Pro přirozenější chování manipulátoru je třeba směrový vektor geometrie manipulátoru na obrazovce, se kterým uživatel právě interaguje. Pro to je potřeba projekce z lokálních souřadnic (LOCAL SPACE) do souřadnic obrazovky (SCREEN SPACE).

Z lokálních souřadnic do souřadnic obrazovky

Vykreslovací pipeline (řetězec) z **LOCAL SPACE** do **SCREEN SPACE** je znázorněna na obrázku 4.10.



Obrázek 4.10: Zobrazovací řetězec (Graphics pipeline) (image link)

Matici pro transformaci z **LOCAL SPACE** do **CLIP SPACE** získáme následovně:

$$M_{world2screen} = M_{projection} \times M_{view} \times M_{model}$$

Pak je potřeba touto maticí vynásobit vektor určující pozici objektu ve

scéně, který je snadné získat z modelovací matice.

V matici $\begin{pmatrix} a & b & c & x \\ d & e & f & y \\ g & h & i & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$ pak vektor $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ odpovídá dané pozici ve scéně.

Vektor v **CLIP SPACE** získáme takto:

$$v_{clip} = M_{world2screen} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

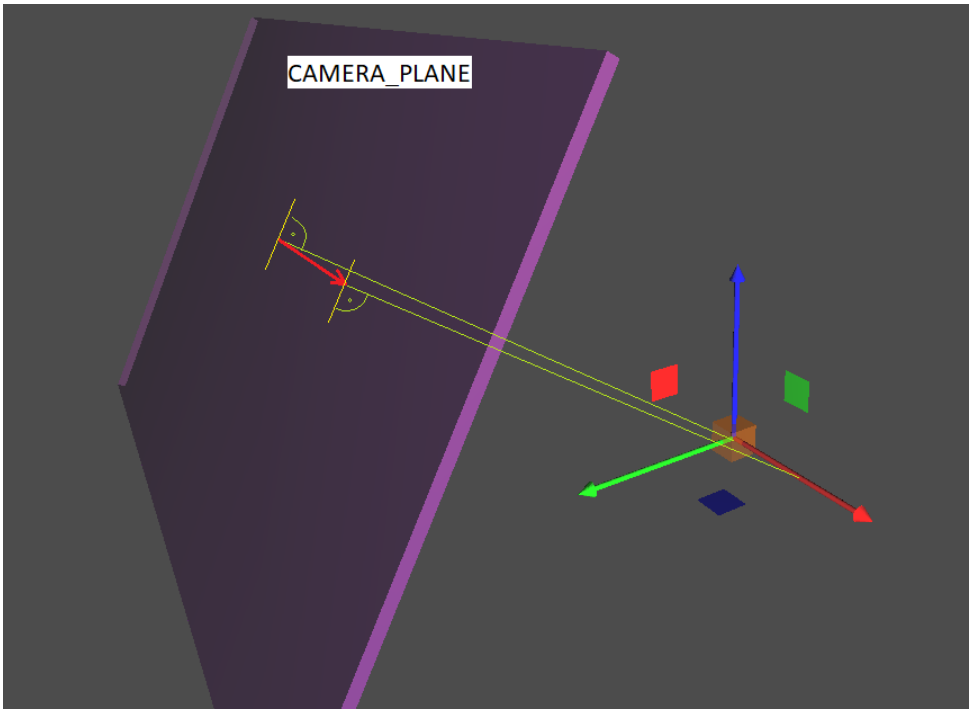
Ten je ještě potřeba převést do **SCREEN SPACE**:

$$v_{screen}.x = (((v_{clip}.x + 1.0f) \times 0.5f) \times width) + px;$$

$$v_{screen}.y = (((-v_{clip}.y + 1.0f) \times 0.5f) \times height) + py;$$

Využití projekce

Pro získání směru vektoru vybrané geometrie manipulátoru je třeba projekce vždy dvou bodů, pozice vybrané geometrie manipulátoru a pozice manipulátoru samotného ve scéně.



Obrázek 4.11: Projekce geometrie vektoru manipulátoru do souřadnic obrazovky

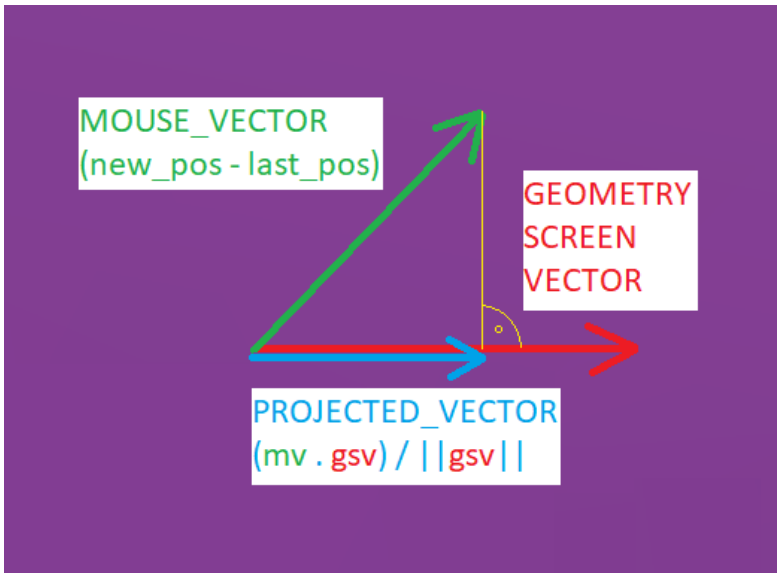
Z těchto dvou bodů promítnutých do **SCREEN SPACE** se získá potřebný směrový vektor geometrie manipulátoru. Pak je potřeba vektor vytažený

myší, který se aktualizuje v závislosti na poslední a aktuální pozici myši na obrazovce, tedy:

$$v_{mouse} = pos_{new} - pos_{last}$$

V tuto chvíli jsou k dispozici dva vektory, vektor vytažený myší a vektor směru geometrie manipulátoru. Projekce vektoru myši na vektor geometrie v rovině udává hodnotu, která určuje velikost posunu.

$$size_{projection} = \frac{v_{mouse}.v_{geometry\_screen\_proj}}{||v_{geometry\_screen\_proj}||}$$



Obrázek 4.12: Hodnota transformace objektu odpovídá projekci MOUSE VECTOR na GEOMETRY SCREEN VECTOR

4.5.2 Transformace v rámci roviny

Pro usnadnění implementace transformace v rámci roviny, jak změny měřítka, tak translace, je implementován `vector<bool>[3]`, ve kterém se dle vybraných plošek manipulátoru nastavují hodnoty. Například v případě, že je vybraná ploška v rovině XY, nastaví se první a druhá hodnota vektoru na *true*.

4.6 Transformace

Transformace manipulátoru je závislá na hodnotě projekce vektoru myši na promítnutý vektor geometrie do souřadnic obrazovky. Pak se jen mění transformační matice v závislosti na vektorech geometrie, se kterými uživatel interaguje. Směr transformace je tedy daný směrem geometrie manipulátoru, se kterou uživatel interaguje, a hodnota získaná projekcí určuje velikost daného vektoru. V případě, že se jedná o translační manipulátor, určuje směr

translace vektor, za který uživatel tahá. Velikost translace určuje projekce vektoru myši na vektor geometrie.

4.6.1 Transformace objektu

Aplikace transformačních matic probíhá za pomoci příkazů *scene->Transform()* a *scene->SetTransformation()*. Ovšem s transformační maticí objektu se pracuje pouze při nakliknutí objektu a při selekci jiného, neboli deselekci daného objektu. Když je tedy objekt transformován pomocí manipulátorů ve scéně, jeho transformační matice zůstává stejná. Objekt je totiž potomkem mateřského uzlu manipulátoru v grafu scény a transformace aplikujeme pouze na mateřský uzel.

Transformační matice objektu se tedy mění při výběru jiného objektu následovně:

```
if (selectedObject){
    setMaterial(selectedObject, savedMaterial);
    Transform(selectedObject, localManipulatorMAT);
    MOVE_SCENE_NODE(selectedObject, manipulatorParent);
    MOVE_SCENE_NODE(manipulator, sceneRoot);
    Transform(manipulator, Matrix());
}
```

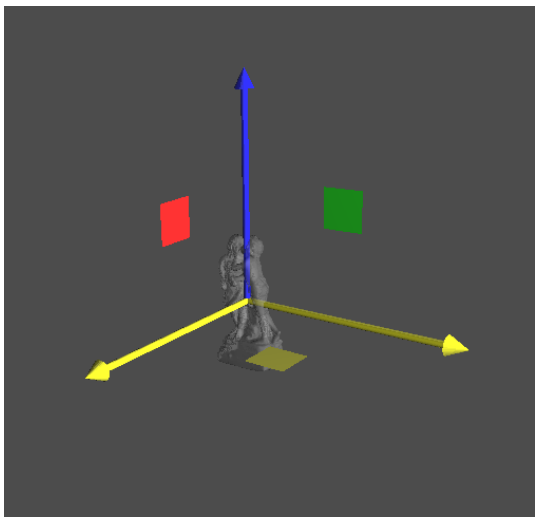
Při výběru objektu je ovšem potřeba zařídit, aby byl manipulátor ve středu vybrané geometrie. To je zařízené tak, že manipulátor převezme od objektu translační složku lokální transformační matice a translační složky lokální transformační matice vybraného objektu se nastaví nuly.

4.6.2 Aplikace translace

Translace spočívá v nastavení transformační matice dle vektoru `vector<bool> vectorToMove[3]` (viz. sekce 4.5.2). V závislosti na jednotlivých hodnotách násobí translační matice mezi sebou. Hodnoty translačních matic jsou předpočítané dle projekce do souřadnic obrazovky (viz. sekce 4.5.1). V případě, že tedy **ID objektu** odpovídá plošce umožňující translaci v rámci roviny, jedná se o složení dvou translací určených projekcí vektorů ležících v rovině dané plošky.

Při aplikaci translace je potřeba nejprve vynásobit modelovací transformaci její inverzí, poté se aplikuje daná translace a následně se vynásobí původní modelovací maticí. Tento přístup je potřeba, protože každý z rodičovských uzlů transformovaného objektu může mít jinou transformační matici.

```
case TRANS:
  if (vectorToMove[0])
    m *= Matrix::Translation(Vector3(shiftX, 0, 0));
  if (vectorToMove[1])
    m *= Matrix::Translation(Vector3(0, shiftY, 0));
  if (vectorToMove[2])
    m *= Matrix::Translation(Vector3(0, 0, shiftZ));
  // this->ID je id materskeho uzlu manipulatoru
  scene->Transform(this->ID, globalManipMatrix * m
                   * globalManipMatrix.Inverse());
  break;
```



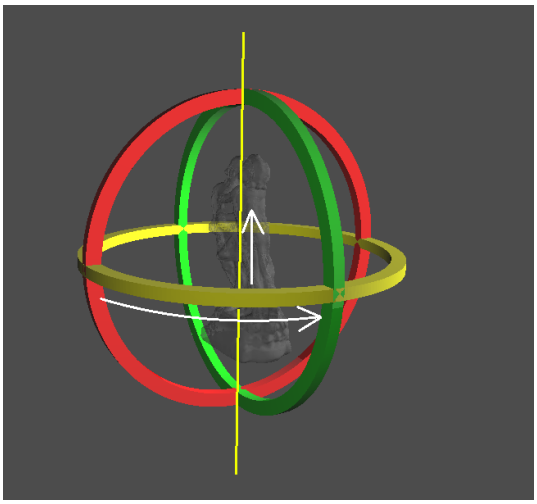
Obrázek 4.13: Chování translačního manipulátoru; změna materiálu geometrie vektorů roviny

4.6.3 Aplikace rotace

Aplikace rotace probíhá obdobně v závislosti na souřadnicích myši, je ovšem trochu jednoduše implementována a změna rotace je závislá na pohybu myši po směrovém vektoru daným osou oblouku. Rychlost rotace, neboli konstanta zpomalující rotaci manipulátoru, je ještě násobena proměnnou `distanceReduction`, která se rovná hodnotě $1/distance_{camera,manip}$. Proměnné `shiftX, shiftY, shiftZ` jsou totiž hodnotou $distance_{camera,manip}$ pronásobené a to by způsobovalo rychlejší rotaci při větší vzdálenosti kamery od vybraného objektu.

Při aplikaci rotace je potřeba podobně jako u translace:
1) nejprve modelovací matici manipulátoru vynásobit inverzí lokální matice manipulátoru
2) poté matici vynásobit inverzí modelovací matice rodičovského uzlu manipulátoru (původního rodiče transformovaného objektu)
3) následně se aplikuje daná rotace
4) násobení matice modelovací maticí rodičovského uzlu manipulátoru
5) a na posledním místě lokální matice manipulátoru.

```
case ROTATE:
  rotationSpeed = 20.0f * distanceReduction;
  if (dragingItem == ROTATE_MANIP.xID)
    m = Matrix::RotationX(shiftX / rotationSpeed);
  else if (dragingItem == ROTATE_MANIP.yID)
  ...
  scene->Transform(this->ID, localManMAT->Inverse() *
  manipParentGlobMAT.Inverse()  * m *
  manipParentGlobMAT * *localManMAT);
  break;
```



Obrázek 4.14: Chování rotačního manipulátoru; pohyb myší po ose oblouku způsobuje rotaci

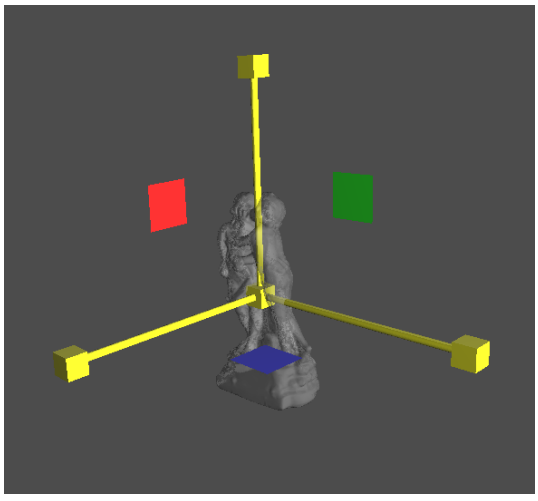
4.6.4 Aplikace změny měřítka

Změna měřítka je implementovaná poměrně jednoduše. V závislosti na pohybu myši se jen mění konstanta určující zvětšování/zmenšování ve směru daném geometrií manipulátoru.

V případě změny měřítka ve všech směrech souřadnicového systému, tedy v případě, že chce uživatel celý objekt zmenšovat/zvětšovat, je daná transformace závislá pouze na horizontální souřadnici myši.

Aplikace scale funguje obdobně jako u rotace (viz. sekce 4.6.3).

```
case SCALE:
  if (vectorToMove[0])
    m *= Matrix::ScaleMat(Vector3(
    shiftX > 0 ? 1.02f : 0.98f, 1.0f, 1.0f));
  if (vectorToMove[1])
    m *= Matrix::ScaleMat(Vector3(
    1.0f, shiftY > 0 ? 1.02f : 0.98f, 1.0f));
  if (vectorToMove[2])
    m *= Matrix::ScaleMat(Vector3(
    1.0f, 1.0f, shiftZ > 0 ? 1.02f : 0.98f));
  else if (dragingItem == SCALE_MANIP.wID) {
    scaleCoef = (newMousePos.x > lastMousePos.x) ?
    1.05f : 0.95f;
    m=Matrix::ScaleMat(Vector3(scaleCoef, scaleCoef,
    scaleCoef));
  }
  scene->Transform(this->ID, localManMAT->Inverse() *
      manipParentGlobMAT.Inverse()  * m *
      manipParentGlobMAT * *localManMAT);
  break;
```



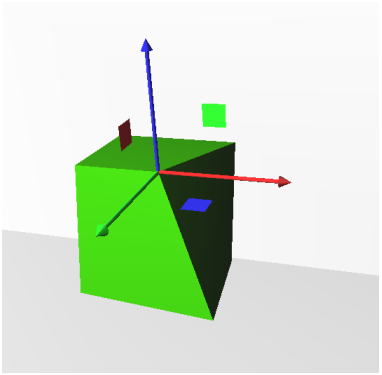
Obrázek 4.15: Interakce s manipulátorem měřítka; uchopení geometrie krychličky ve středu manipulátoru umožní změnu měřítka ve všech směrech → zbarví vektory v jejichž směr dochází k transformaci

4.7 Polygonální síť

Každá geometrie objektu je určitá síť, která se skládá z různých primitiv, trojúhelníků, mnohoúhelníků atp. Nese mnoho informací, včetné pozice vrcholů, směru normál a indexu, kde jednotlivá primitiva v seznamů vrcholů začínají.

4.7.1 Úprava vrcholů polygonální sítě

Úprava pozice vrcholů funguje obdobně jako transformace objektu. Pro výběr vrcholu objektu je ovšem potřeba přidržet klávesu **TAB**. Tímto se vytvoří ve scéně malá krychlička, která se přesune do pozice vybraného vrcholu. Manipulátor bude totiž měnit pouze pozici této krychličky, avšak po každém upuštění levého tlačítka myši se přepočítá pozice původně vybraného vrcholu tak, aby odpovídala pozici krychličky ve scéně.



Obrázek 4.16: Úprava polygonální sítě pomocí manipulátoru

Detekce vybraného vrcholu

Jaký z vrcholů geometrie objektu je vybrán záleží na tom, jaký je neblíže průsečíku paprsku, který je dán kliknutím kolečkem myši na objekt, s upravovaným objektem. Tato metoda tedy prochází všechny vrcholy geometrie vybraného objektu.

Implementace

Geometrii vybraného objektu lze získat jednoduše pomocí příkazu:

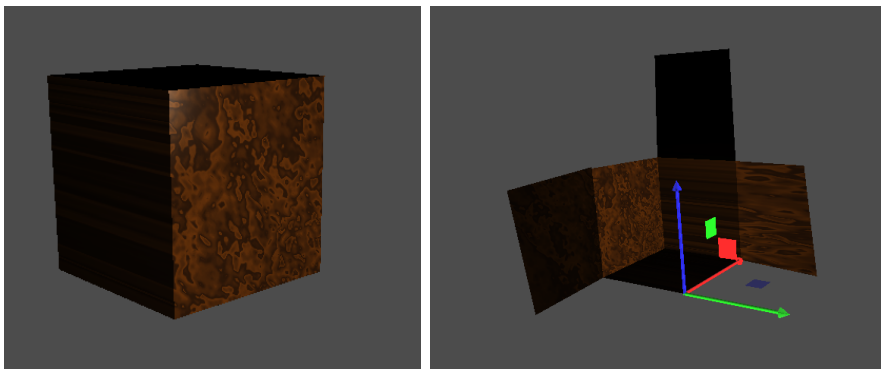
```
smart_pointer<GeometryTriangles> g = scene->GetGeometry(objectID);
```

Ta už v sobě nese potřebné elementy. Po úpravě vektoru vrcholů je potřeba geometrii daného objektu ve scéně aktualizovat:

```
// naklonovani puvodni geometrie objektu
GeometryTriangles* geo2 = (GeometryTriangles *)g->Clone();
// prirazeni noveho vektoru vrcholu naklonovane geometrii
geo2->vertices.assign(newVertices.begin(), newVertices.end());
scene->AddGeometry(geo2);
```

4.7.2 Úprava vrcholů primitiv objektu

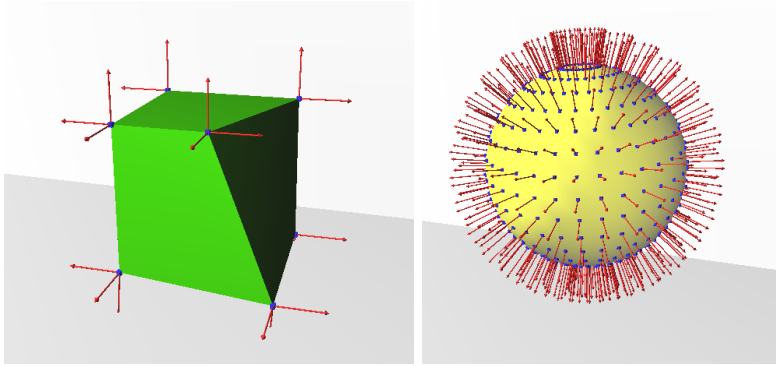
Tato funkcionalita se moc neliší od obyčejné úpravy vrcholů polygonální sítě objektu. Avšak tato funkcionalita nabízí úpravu pozic vrcholů jednotlivých primitiv objektu. Polygonální síť geometrie je tak možné úplně rozložit. Pro takové chování je třeba zaškrtnout na kartě modulu checkbox *unfold object*.



Obrázek 4.17: Úprava polygonální sítě primitiv objektu

4.7.3 Vykreslení vrcholů a normál polygonální sítě

Na pozicích vrcholů upravované geometrie objektu se při přidržení klávesy **TAB** zobrazí malé geometrie krychliček a vektorů, které slouží ke zvýraznění pozic vrcholů a směru normál. Identifikátory vykreslované geometrie se ukládají do vektoru, který při upuštění klávesy **TAB** všechnu geometrii smaže.



Obrázek 4.18: Úprava polygonální sítě primitiv objektu

Výpočet pozic vrcholů

Na pozicích vrcholů objektu se vykreslují nové objekty, jejichž transformační složka matice odpovídá:

$$v_{world} = M_{objWorld} \times v_{local}$$

Ve vektoru vrcholů geometrie jsou totiž lokální pozice vrcholů.

Výpočet směru normál

Normálová matice (3x3) se počítá jako transpozice inverze lineární části modelovací matice. Směrový vektor normály je pak zjištěn vynásobením normálové matice s daným lokálním směrem normály.

$$n_{world} = (M^{-1}_{objWorld3x3})^T \times n_{local}$$

Pro správnou rotaci geometrie normály je využita funkce

```
Matrix LookAt(eye, target, upVector){}
```

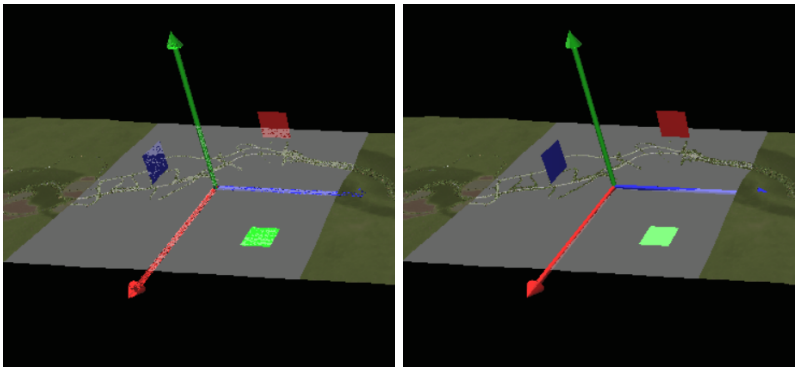
Do té je ale potřeba posílat jiný *upVector* v případě, že směr vykreslované normály odpovídá danému *upVector*. To je ošetřeno jednoduchou podmínkou. Modelovací matice vykreslované normály je pak vypočítána následovně:

$$M_{distanceScale} \times LookAt(v_{world}, v_{world} + n_{world}, up)$$

Matice $M_{distanceScale}$ slouží k zachování stejné velikosti všech vrcholů a normál na obrazovce, její elementy jsou tedy vypočítány dle vzdálenosti daného vrcholu od kamery.

4.8 Neobvykle definovaná scéna

V systému VRUT je importováno mnoho scén v různých formátech. Dle způsobu importu a vymodelování scény se pak ve scénách projevují různá chování objektů a scén samotných. Některé jsou například vymodelovány v milimetrech, jsou ve skutečné velikosti a geometrie manipulátoru se pak zobrazuje přerušovaně, dochází k tzv. *z-fightingu*. Stačí ovšem na kartě modulu *Navigation* nastavit hodnotu *NearPlane* na 1000+.

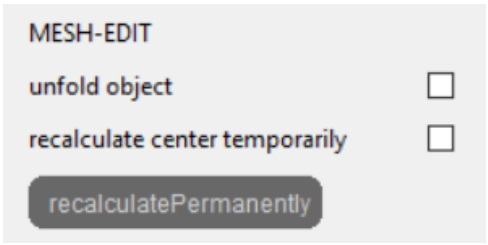


Obrázek 4.19: Zobrazení geometrie manipulátoru v nascalované scéně NearPlane: 1 (vlevo), NearPlane: 1000 (vpravo)

Některé scény jsou ovšem naimportované tak, že objekty nemají nastavené modelovací matice. Mají jen definované vrcholy geometrie rovnou ve světových souřadnicích. Manipulátor implementovaný v modulu GeometryEditor ovšem funguje tak, že se řídí právě modelovacími souřadnicemi objektu a vykresluje

se na pozici dané modelovací maticí. Je proto potřeba implementovat způsob, který bude zobrazovat manipulátor na pozici těžiště vybrané geometrie.

V modulu GeometryEditor jsou implementovány dvě různé funkcionality. První pouze transformuje uzel geometrie manipulátoru do těžiště tělesa, rotace a změna měřítka tak stále počítají se středem v počátku scény. Druhá nabízí trvalé přepočítání modelovací matice a vrcholů geometrie objektu.



Obrázek 4.20: Karta modulu - úprava geometrie (zhora): rozložení objektu, transformace manipulátoru vždy do těžiště vybrané geometrie, trvalé přepočítání transformace objektu

4.8.1 Implementace

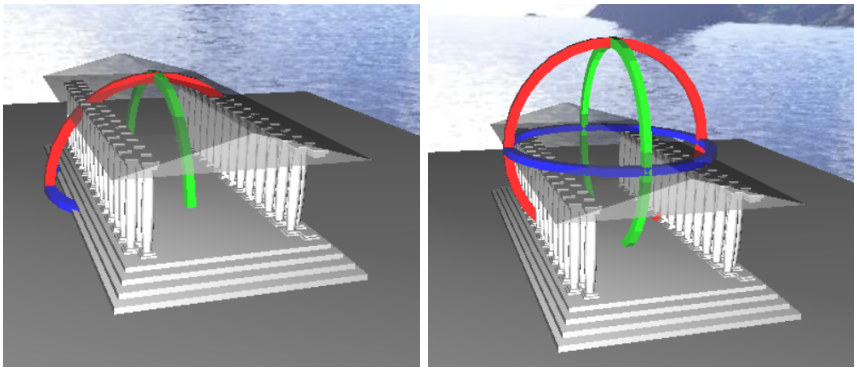
Modelovací matici můžeme jendoduše získat vypočítáním těžiště geometrie daného tělesa.

$$v_{translation} = \frac{\sum_{i=0}^{n} v_i}{n}$$

Translační matici $M_{translation}$ z vektoru $v_{translation}$ už je snadné vytvořit tak, že nahradíme poslední sloupec jednotkové matice elementy vektoru. Pak je ovšem potřeba přepočítat pozice vrcholů geometrie daného objektu.

$$v_{new\_vertex\_pos} = M^{-1}_{translation} \times v_{old\_vertex\_pos}$$

Následně nahradíme modelovací matici objektu maticí $M_{translation}$ a těleso ve scéně by mělo být manipulátorem jednoduše transformovatelné.

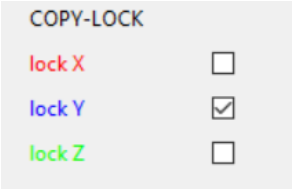


Obrázek 4.21: Manipulátor ve scéně temple.wrl; manipulátor v počátku před úpravou (vlevo), manipulátor ve středu geometrie tělesa po úpravě (vpravo)

4.9 Kopírování objektu

V modulu GeometryEditor je funkce kopírování vybraného objektu vyvolána klasicky kombinací **Ctrl+C**, tím se uloží do proměnné *objectToCopy* **ID** právě vybraného objektu.

Kombinace **Ctrl+V** pak vyvolá vložení, ovšem pro vložení objektu do scény je potřeba navíc kliknout do scény, kde se na průsečík paprsku myši s objektem vykreslí nová geometrie objektu. Navíc karta modulu umožňuje zamknutí jednotlivých souřadnic kopírovaného objektu.

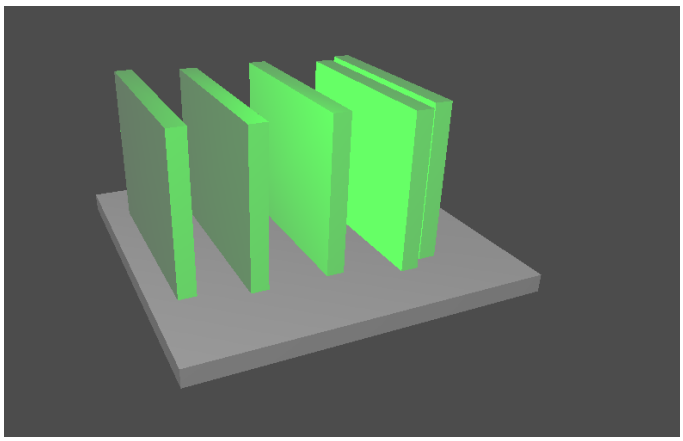


Obrázek 4.22: Možnost uzamknutí jednotlivých souřadnic kopírovaného objektu

4.9.1 Implementace

Transformační matice se pro vykopírovávaný objekt sestaví následovně:

```
Matrix m = scene->GetWorldTransMatrix(objectToCopy);
m._m41 = COPY_LOCK.x ? m._m41 : mouseIntersectionPoint.x;
m._m42 = COPY_LOCK.y ? m._m42 : mouseIntersectionPoint.y;
m._m43 = COPY_LOCK.z ? m._m43 : mouseIntersectionPoint.z;
//nastaveni transformacni matice vykopirovaneho objektu
scene->SetTransformation(coppiedObjectID, m);
```



Obrázek 4.23: Vykopírování objektu, COPY_LOCK.y = true

4.10 Krok zpět - UNDO

Undo je nezbytnou pomůckou všech editorů geometrie. V systému VRUT je implementovaná struktura rozlišující tři typy akcí.

```
enum ACTION_TYPE {TRANSFORMATION, MESH_EDIT, OBJ_COPY};

struct undoElement {
  NODE_ID id;
  Matrix m;
  ACTION_TYPE type;
};
```

Do vektoru `vector<undoElement>` `undoVector` se pak ukládají jednotlivé operace. Vektor funguje jako zásobník LIFO (*Last In First Out)*, je tedy ideálním prvkem pro zpětné krokování. Element, který do něj uložíme jako poslední, tedy akce, kterou jsme vykonali jako poslední, je jako první na řadě v případě zpětného krokování. Maximální možný zpětných počet kroků je sto.

4.10.1 Aplikace transformace

Uložení jakékoliv transformace aplikované pomocí manipulátoru funguje tak, že se do proměnné *id* uloží identifikátor vybraného objektu, do proměnné *m* globální transformační matice vybraného objektu a do proměnné *type* pochopitelně **TRANSFORMATION**. Uložení je vyvoláno upuštěním levého tlačítka myši a to pouze v případě, že se element přidávaný do zásobníku nerovná elementu na vrchu zásobníku.

Aplikace zpětného kroku (dále jen UNDO) pak probíhá tak, že se danému objektu nastaví lokální transformační matice následovně:

$$M_{obj\_local} = M_{obj\_undo\_global} \times M^{-1}_{obj\_parent\_global}$$

4.10.2 Úprava polygonální sítě geometrie

Uložení editace polygonální sítě objektu probíhá podobně, jako u transformace. Do proměnné *id* se ukládá identifikátor upravovaného objektu a do proměnné *type* pochopitelně **MESH_EDIT**, ovšem do proměnné *m* se ukládá matice:

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & id \end{pmatrix}, \text{ kde } \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$ je původní pozice vrcholu mající index ve vektoru vrcholů na pozici dané hodnotou *id*.

Aplikace UNDO úpravy polygonální sítě objektu pak probíhá obdobně, jako dopředný krok (viz. sekce 4.7.1).

4.10.3 Vykopírování objektu

Uložení elementu akce kopírovaného objektu je velmi jednoduché. Matice je totiž naprosto nepotřebná složka elementu a proto se pouze klasicky do

proměnné *id* uloží identifikátor nově vykopírovaného uzlu a do proměnné *type* hodnota **OBJ_COPY**.

UNDO pak jednoduše, v případě, že je uzel stále ve scéně, daný uzel ze scény odstraní.

```
if (scene->GetNode<SceneNode>(element.id))
  scene->Remove(element.id);
```

Kapitola 5
Výsledky práce

Tato kapitola popisuje testování modulu na pěti scénách různé složitosti. Realizace základní studie použitelnosti nástroje je testována se třemi uživatel. Scény jsou jednotlivě popsány a rozebrány. Uživatelům byl po testování podán jednoduchý dotazník. Otázky a odpovědi uživatelů na otázky jsou popsány níže.

Modul GeometryEditor je implementovaný v pěti souborech, kde soubor **.xrc* pouze popisuje rozmístění komponent karty modulu (viz. tabulka 5.1).

Název souboru	Počet řádků kódu
GeometryEditor.cpp	1070
GeometryEditor.h	176
Manipulator.cpp	683
Manipulator.h	188
GeometryEditorLayout.xrc	337

Tabulka 5.1: Počet řádků kódu v jednotlivých souborech

5.1 Testované scény

V systému VRUT jsem testoval modul GeometryEditor na více scénách různé složitosti. Každá ze scén přinášela nové chyby, které bylo třeba opravit ale i inspirace, jak lépe implementovat různé funkcionality modulu.

Rozměry testovaných scén se dočtete v tabulce 5.2.

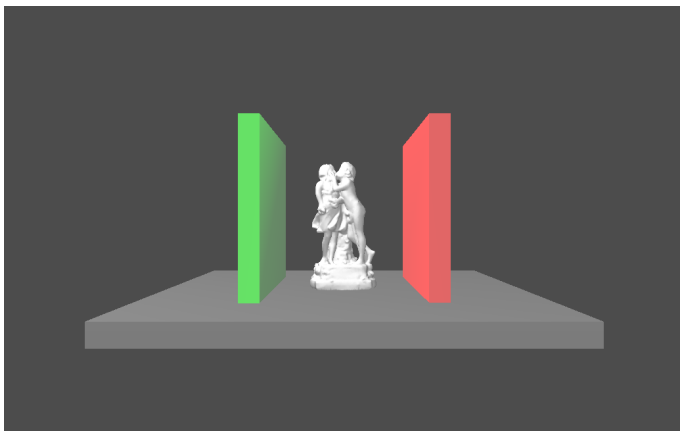
Scéna	Počet			
	uzlů grafu scény	geometrie	stěn	vrcholů
colorTestScene.wrl	34	5	982	2988
test.fhs	72	24	50 357	151 176
colorBounce.wrl	19	4	50 783	152 412
temple.wrl	637	316	104 713	314 160
dalnice.vrut	321	275	94 403	2 990 949

Tabulka 5.2: Rozměry testovaných scén

5.1.1 Scéna colorBounce

Základní funkcionality modulu, transformace, vykopírování objektu a úprava polygonální sítě, byly testovány ve scéně `colorBounce.wrl`. Je velmi jednoduchá a pro testování správnosti základních transformací opravdu ideální.

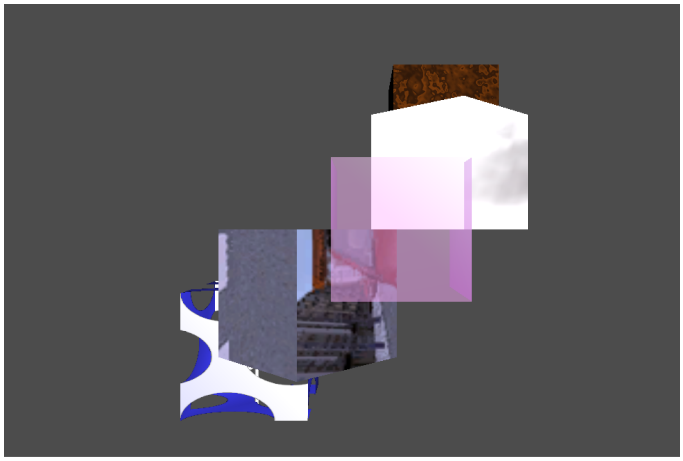
Tato scéna byla také využita na základní studii použitelnosti se třemi uživateli (viz. sekce 5.3).



Obrázek 5.1: Scéna colorBounce

5.1.2 Scéna sceneTest

Scéna `sceneTest.wrl` posloužila k testování správné změny materiálu vybraného objektu. Na jednoduchých objektech jsou v této scéně různé textury.

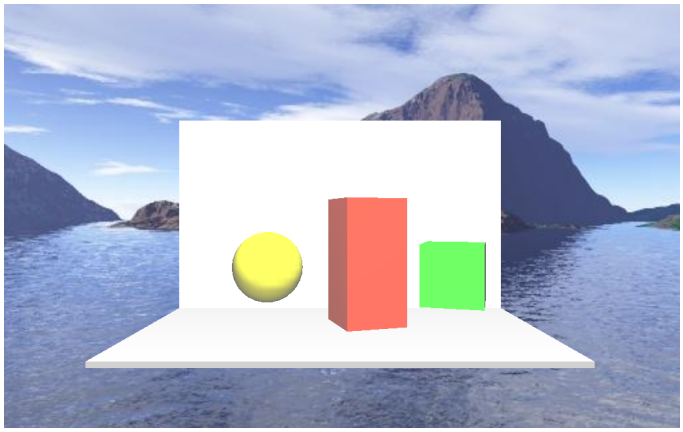


Obrázek 5.2: Scéna sceneTest

5.1.3 Scéna colorTestScene

Tato scéna je také jedna z jednodušších, avšak je zde ideální objekt na testování správnosti směru normál, a tím je sféra. Pak jsou zde zvláštně

definované kvádry, jejichž modelovací matice neodpovídá jejich těžišti a manipulátor se zde tedy zobrazoval trochu jinde. Pro přepočítání stačilo správně implementovat funkci tlačítka **recalculatePermanently**.

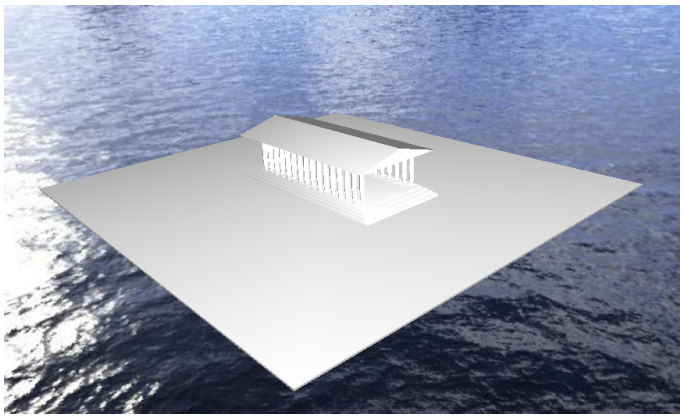


Obrázek 5.3: Scéna colorTestScene

5.1.4 Scéna Temple

`Temple` je opravdu zvláštně vymodelovaná scéna, která si zaslouží představit. Všechny objekty scény jsou v grafu scény na stejné úrovni. Sloupy chrámu se skládají ze dvou patek (kvádry) dole a dvou patek nahoře a mezi nimi je jeden válec. Všechny tyto části sloupů jsou v grafu scény na stejné úrovni se všemi ostatními objekty a pojmenovány Box0-255 a Cylinder0-62.

Pro správné chování manipulátoru při výběru objektu je zde opět potřeba funkce přepočítání těžiště geometrie (viz. sekce 4.8.1).

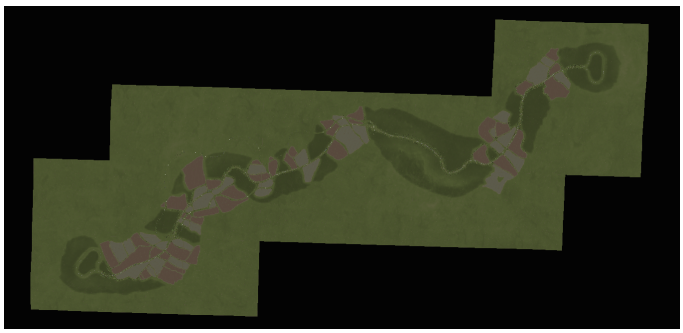


Obrázek 5.4: Scéna Temple

5.1.5 Scéna dálnice

V systému VRUT je také vymodelovaná obrovská scéna dálnice. Jsou v ní mosty, krajina, stromy atd. Takhle velká scéna už má pochopitelně větší nároky na hardware a obdobně jako ve scéně `Temple` je ve scéně `dalnice.vrut`

pro správné chování manipulátoru využít tlačítko **recalculatePermanently**, nebo checkbox **recalculate center**.



Obrázek 5.5: Scéna dálnice

5.2 Ovládání modulu (uživatelská příručka)

Modul GeometryEditor slouží k úpravě pozice, rotace a měřítka ve scénách systému VRUT. Dovolil jsem si zmínit i příkazy umožňující pohyb ve scéně, které neimplementuje modul GeometryEditor ale modul Navigation.

Vstup	Funkce
MYŠ	
Přidržení levého tlačítka myši + táhnutí	rotace ve scéně okolo vybraného pivota / interakce s manipulátory
Táhnutí pravým tlačítkem	posun v rámci roviny kamery
Kliknutí kolečkem	výběr objektu ve scéně
Dvojklik levým tlačítkem	zrušit výběr objektu ve scéně / potvrdit transformaci vybraného objektu
Rolování kolečkem myši	přibližování/oddalování scény
KLÁVESNICE	
Enter (numpad)	zrušit výběr objektu ve scéně / potvrdit transformaci vybraného objektu
Esc	zrušit výběr objektu a transformovat objekt do stavu před výběrem
Delete	odstranit vybraný objekt ze scény
G	přepnout manipulátor do translačního režimu (posun)
R	přepnout manipulátor do rotačního režimu (rotace)
S	přepnout manipulátor do režimu změny měřítka
TAB+Kolečko myši	výběr vrcholu geometrie
Přidržení TAB	zobrazí vrcholy a normály vybraného objektu
Přidržení Ctrl	vypne funkci rotaci ve scéně při tahu myší
Ctrl+C	zkopírování do schránky vybraný objekt
Ctrl+V	vložení zkopírováného objektu na kolečkem nakliknutou pozici
Ctrl+Z	krok zpět, UNDO

Tabulka 5.3: Ovládání editoru geometrie

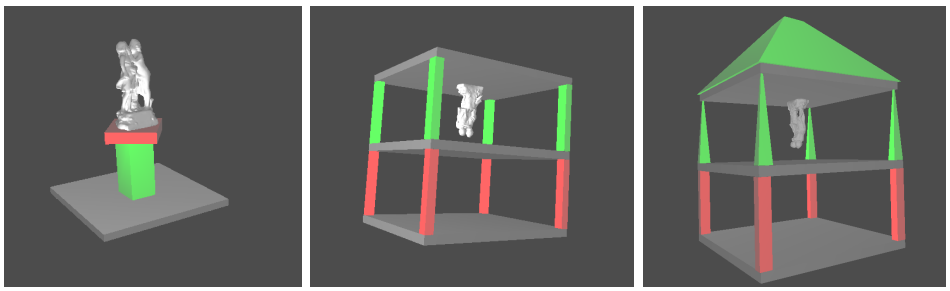
5.3 Uživatelské zkušenosti - dotazník

Testování modulu s nejméně třemi uživateli bylo za podmínek způsobených virem COVID-19 trochu obtížnější, ale nakonec nebyl problém tři osoby sehnat. Bylo by nejlepší kdyby byli testovaní uživatelé již s grafickými softwary jako je systém VRUT (Blender, Unity atd.) seznámeni, ale takový komfort se ovšem nepovedlo zařídit.

První uživatel, David, studuje Design na ČVUT pod fakultou architektury a s některými grafickými softwary byl již dříve seznámen. Druhý uživatel, Štěpán, kolega z FELu, který také studuje Počítačové hry a grafiku byl ideálním uživatelem na testování modulu, grafických softwarů již v životě vyzkoušel opravdu mnoho a používá je na denní bázi. Třetí uživatel, Karolína, je studentem gymnázia a její zkušenosti s grafickými softwary jsou v podstatě nulové, i tak ale mohla posoudit základní rysy modulu, jako je intuitivnost a spolehlivost.

5.3.1 Zadání a průběh testování

Každému uživateli byly před testováním ukázány základní funkcionality modulu, bylo mu poskytnuto ovládání systému a modulu samotného. Uživatel dostal daný počáteční stav scény `colorBounce.wrl` a měl zadané tři stavy scény, do kterých měl pomocí implementovaného nástroje postupně scénu upravit.



Obrázek 5.6: Finální stavy scény

Po dokončení testování vyplnil každý uživatel dotazník, který byl zaměřený na jejich postup, kroky, ale i osobní názory a doporučení. Odkaz na dotazník zde.

Časová značka	12.05.2020 9:59	13.05.2020 15:40	14.05.2020 9:29
1) Pracoval/a jste už dříve s nějakými editory geometrie?	Ano	Ano	Ano
2) Případně, že ano, s jakými?	Rhinoceros 6, Inventor	Blender, Unity, Maya, 3DSMax	SketchUp
3) Znáte některé z následujících softwarů?	Maya, Blender, Unity, UnrealEngine	Maya, Blender, Unity, UnrealEngine	žádný
4) Jak moc intuitivní vám přišel manipulátor v systému VRUT? (1-5)	2	2	1
5) Líbila se vám geometrie manipulátorů? (to, jak vypadají?)	Ano	Ne	Ano
6) Případně, že ne, co byste na vzhledu změnili/zlepšili a proč?		přidal bych průhlednost gizma, při změně měřítka bych netransformoval manipulátor (scale)	
7) Vyzkoušeli jste si při testování posun po rovině?	Ne	Ano	Ano
8) Vyzkoušeli jste si při testování změnu měřítka v rovině nebo ve všech směrech souřadnicového systému?	Ne	Ano	Ano
9) Použil/a jste vykopírování objektu? (Ctrl+C, Ctrl+V, kluknutí na místo vložení)	Ano	Ano	Ano
10) Použili jste krok zpět neboli UNDO? (Ctrl+Z)	Ano	Ano	Ano
11) Byla první úloha testu obtížná? (1-5)	1	1	1
12) Pokud ano, proč?			
13) Jaké všechny manipulátory jste použil/a v první úloze?	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka
14) Byla druhá úloha testu obtížná? (1-5)	3	2	2
15) Pokud ano, proč?	rozjel se mi objekt po nascalování a rotaci, nedržel v pravouhlych tvarech, jak jsem chtěl. Objekt se po výběru položil o 180 stupnu.		Neintuitivní rotace
16) Jaké všechny manipulátory jste použil/a v druhé úloze?	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka
17) Byla třetí úloha testu obtížná? (1-5)	4	4	4
18) Pokud ano, proč?	Chtělo by to přichycování k hranám objektů	úprava polygonů bolí	Neumím používat nový nástroj
19) Jaké všechny manipulátory jste použil/a ve třetí úloze?	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka, Úprava polygonální sítě	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka, Úprava polygonální sítě	Translační manipulátor, Rotační manipulátor, Manipulátor měřítka, Úprava polygonální sítě
20) Nefungovalo vám něco, nebo přestal program v průběhu vaší práce pracovat?	Ano	Ne	Ne
21) Pokud ano, jste schopný/á popsat situaci po které nastalo spadnutí programu?	zmáčknutí klávesy DELETE při úpravě polygonální sítě		

Obrázek 5.7: Uživatelský dotazník - Odpovědi
Otázka č. 4: 1 - snadné se s ním naučit, 5 - nemožné se s ním naučit
Otázky č. 11, 14, 17: 1 - jednoduchá, 5 - velmi náročná

5.3.2 Reakce na dotazník

U všech testování jsem byl přítomen, proto jsem měl možnost odchytit většinu případných chyb nebo zvláštních chování systému.

Za opravdu podstatnou připomínku jsem považoval chybu popsanou v dotazníku uživatelem č.1. Nešlo o implementačně náročnou věc a chyba byla ihned opravena.

Chybu, kterou popisuje uživatel č.1 v otázce č.15 *Proč považuje druhou úlohu testu za obtížnou?* není vyloženě chybou. Modul nabízí pouze globální transformace, a tak došlo při testování k pouhému zkosení, kterého se dá dosáhnout kombinací rotace a změny měřítka. Avšak chyba druhá, kde se uživateli položil objekt po výběru o 180 stupňů je pro mě zatím záhadou, nepodařilo se mi znovu vyvolat takové chování programu.

Připomínka uživatele č.2 v otázce č.6 k manipulátoru na změnu měřítka by se dala jednoduše realizovat. Pro intuitivnější interakci mi ovšem přijde lepší manipulátor transformovat s vybraným objektem.

Třetí úloha testu byla velmi obtížná, uživatel si musel zvyknout na způsob výběru vrcholů objektu. Všichni tři uživatelé hodnotili obtížnost této úlohy stupněm 4. Uživateli č.1 by zjednodušilo práci přichycování transformovaného objektu k hranám ostatních objektů - velmi obtížné na implementaci, ale bez pochyb by to usnadnilo práci nejen s touto úlohou. Uživateli č.3, nejméně zkušenému v oboru počítačové grafiky, dělalo problémy ovládání nástroje na úpravu vrcholů polygonální sítě, to by v případě více takových připomínek nebyl implementačně náročný problém.

Kapitola 6

Závěr

Cílem projektu bylo zmapovat způsoby editace geometrie v existujících modelovacích nástrojích, vybrat vhodnou sadu funkcí pro realizaci editoru geometrie v systému VRUT a implementovat do něj příslušný modul.

Podařilo se vytvořit funkční editor geometrie umožňující interaktivní editaci geometrických transformací objektů (translace, rotace, změna měřítka). Mimo to byly úspěšně implementovány i další funkcionality: editace vrcholů polygonální sítě a zobrazení vrcholů a normál geometrie. Modul byl úspěšně testován na pěti scénách různé složitosti a byla provedena i základní studie jeho použitelnosti se třemi uživateli. Výsledky testování a zpětná vazba uživatelů jsou v neposlední řadě inspirací pro možná budoucí rozšíření tohoto modulu.

Systém VRUT je aplikace velkých rozměrů a možností. To, ve spojení s různorodostí scén a implementací ostatních modulů, nabízí mnoho možností rozšíření implementovaného modulu, ale také proto bude modul do budoucna vyžadovat větší pozornost.

Editory geometrie v ostatních modelovacích nástrojích nabízí stále více možností než modul implementovaný v systému VRUT. Editor geometrie bude mít v systému VRUT jistě velké využití a proto je velkou výhodou správná implementace a kladení velkého důrazu na intuitivnost nástroje. Modul je nepochybně stále možné zlepšovat.

Jako jedna z dalších možností vývoje modulu se nabízí například možnost úpravy normál polygonální sítě nebo možnost výběru více objektů najednou. Ve třídě GeometryEditor je navíc implementovaná většina funkcionalit modulu, které by bylo určitě lepší strukturovat a oddělit do separovaných souborů.

Výsledný modul funguje velmi podobně jako v testovaných softwarech a je plně funkční. Modul umožňuje snadnou práci s objekty a má velmi jednoduché ovládání. Intuitivnost a korektnost byly prioritou, ale je stále mnoho směrů ve kterých je možné modul zlepšovat a vyvíjet.

Literatura

[1] David J. Eck. *Introduction to Computer Graphics*, Hobart and William Smith Colleges, 2016.

[2] Václav Kyba. *Modulární 3D prohlížeč*. Diplomová práce. ČVUT FEL, 2008.

[3] Steve Marschner, Peter Shirley. *Fundamentals of Computer Graphics*, A K Peters/CRC Press, 2015.

[4] *Dokumentace aplikace VRUT*. ŠKODA Auto a.s., ČVUT FEL, 2017

[5] Philip J. Schneider, David H. Eberly. *Geometric Tools For Computer Graphics*, Morgan Kaufmann, 2003.

[6] Allen Sherrod. *Game Graphics Programming*, Stacy L. Hiquet, 2008.

[7] Hughes John F..*Computer Graphics*, Addison-Wesley, 2014.

[8] John Vince. *Rotation Transforms for Computer Graphics*, Springer Science & Business Media, 2011.

[9] Veysi İşler, Haşmet Gürçay, Hasan Kemal Süher and Güven Çatak. *Contemporary Topics in Computer Graphics and Games*, Peter Lang GmbH, Internationaler Verlag der Wissenschaften, 2019.

[10] Jones Huw. *Computer Graphics Through Key Mathematics*, Springer, 2001.

[11] David Salomon. *Transformations and Projections in Computer Graphics*, Springer-Verlag London, 2007.

[12] John Vince. *Quaternions for Computer Graphics*, Springer, 2011.