ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ
KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

Bakalářská práce

Uživatelské rozhraní pro modifikaci parametrů vykreslovacích algoritmů

Jiří Netušil
Softwarové technologie a management

Květen 2015

Vedoucí práce: Ing. Ladislav Čmolík, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jiří Netušil**

Studijní program: Softwarové technologie a management
Obor: Web a multimedia

Název tématu: **Uživatelské rozhraní pro modifikaci parametrů vykreslovacích algoritmů**

Pokyny pro vypracování:

Analyzujte typy parametrů, které se běžně předávají GLSL programům (např. barva, pozice a směr pohledu kamery) a různé způsoby jakými lze takové parametry zadávat pomocí grafického uživatelského rozhraní. Na základě analýzy navrhněte a implementujte knihovnu, která umožní vytvořit uživatelské rozhraní pro změnu hodnot těchto parametrů. Dále navrhněte a implementujte časovou osu, která umožní definovat a přehrát změnu hodnot parametrů v čase. Pro vlastní vykreslování 3D geometrie použijte existující knihovnu, kterou dodá vedoucí práce. Při návrhu i implementaci se zaměřte na snadnou rozšiřitelnost výsledné knihovny o nové prvky uživatelského rozhraní a jednoduché aplikační rozhraní, skrze které bude možné knihovnu používat. Funkčnost knihovny demonstrujte na pěti až deseti příkladech s různou složitostí.

Seznam odborné literatury:

R. S. Wright, B. Lipchak and N. Haemel. OpenGL Superbible: comprehensive tutorial and reference. Addison-Wesley, 2007.

Vedoucí: Ing. Ladislav Čmolík, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

V Praze dne 4. 11. 2014

Prohlášení

Tímto prohlašuji, že práci s názvem „Uživatelské rozhraní pro modifikaci parametrů vykreslovacích algoritmů“ jsem vypracoval samostatně, a veškerou použitou literaturu jsem uvedl mezi zdroji.

V Praze dne: . . . . . .

Jiří Netušil

Poděkování

Tímto bych chtěl poděkovat především svému vedoucímu práce panu Ing. Ladislavu Čmolíkovi, za jeho ochotu a pomoc při potížích na pravidelných konzultačních hodinách, a také své rodině, která mne po celou dobu studia ve všem plně podporovala.

Abstrakt

V současné době neexistuje jednoduchý způsob vytvoření uživatelského rozhraní a ovládacích prvků pro parametry v knihovně Tigr využívající Java Open Graphics Library k vykreslení 3D scény. Cílem mé práce je analyzovat a vytvořit prostředí, ve kterém se budou nacházet implementace nejčastějších ovládacích prvků pro parametry, s možností jednoduchého rozšíření.

Klíčová slova: Uživatelské rozhraní, Java, Java Open Graphics Library, Tigr, tiger

Abstract

At this time, there is no simple way to create Graphical User Interface and controllers for parameters used in Java library Tigr, which uses Java Open Graphics Library to render a 3D scene. Goal of my thesis is to analyse and implement an environment containing implementations of the most common controllers for parameters, enabling to easily implement new extensions.

Keywords: Graphical User Interface, Java, Java Open Graphics Library, Tigr, tiger

Obsah

Seznam obrázků

Úvod

Cílem mé práce je rozšíření knihovny Tigr o třídy a metody, které umožní uživateli (programátorovi) vytvořit ovládací prvky pro vlastní implementaci scény, k ovládání předem zvolených prvků v reálném čase, jinak řečeno uživatel nebude nucen při potřebě změny, například pozice kamery, efektu v shaderu či barvy pozadí pokaždé běžící program vypnout a ručně parametry v kódu přepsat. Výsledné ovládací prvky by mělo být možné přiřadit na dokovatelné panely, které by bylo možné přesunout či skrýt, také za chodu programu. Výsledek mé práce by měl být snadno použitelný a rozšiřitelný. Při psaní textu předpokládám, že čtenář rozumí programovacímu jazyku Java, a také má základní zkušenosti s OpenGL. K pochopení některých částí je potřeba i znalosti lineární algebry.

Pro tuto práci jsem se rozhodl, protože téma 3D grafiky mi připadá zajímavé a zároveň se pracuje v programovacím jazyce Java, ve kterém se cítím v současnosti nejlépe. Předměty, které se na škole zabývaly OpenGL, s ním pracovaly v jazyce C, C++, nebo C#, se kterými nemuseli studenti, stejně jako já, být zcela seznámeni. Vypracováváním této práce se snad obohatím o nějaké vědomosti o OpenGL, a také si myslím, že výsledek mé práce by mohl pomoci názorně ukázat chování funkcí a shaderů v OpenGL i prostřednictvím Javy.

Práce je rozdělená do tří velkých kapitol s vlastními podkapitolami.

První kapitola, Analýza a návrh, se zabývá seznámením s knihovnou Tigr a jejím stavem ve verzi, kterou jsem dostal na začátku vytváření své práce, dále jak se s knihovnou pracuje a co ji tvoří. Dále analyzuji parametry a navrhuji, jak vyřešit možnost manipulace s nimi za chodu programu. V kapitole ještě navrhuji ovládací prvky pro analyzované parametry, celkový design uživatelského rozhraní, a nakonec hledám knihovnu pro práci s dokovatelnými panely.

Implementace popisuje už realizované parametry a ovládací prvky. Kapitola se tak dělí na Parametry, kde postupně procházím implementační vlastnosti každého parametru, a Ovládací prvky, kde popisuji chování každého z nich, a zároveň je u každého obrázek výsledné podoby prvku.

Ve Výsledcích je představeno deset příkladů, které jsou zamýšleny jako názorná ukázka pro vytvoření a práci s vlastní implementací uživatelského rozhraní.

Všechny informace ohledně OpenGL beru z knihy *OpenGL SuperBible Fourth Edition Comprehensive Tutorial and Reference 4th Edition.*

Pokud čtenáři není jasné něco z Lineární algebry, která je používaná v OpenGL ve velkém rozsahu, mohu z osobních zkušeností doporučit knihu *Úvod do algebry, zejména lineární.*

Analýza a návrh

Tato kapitola ze zabývá analýzou jak současného stavu knihovny, tak možnostmi rozšíření a návrhy na možné implementace. První část se zabývá knihovnou Tigr, ve které pracuji, a je následována popisem postupu a práce s ní. Dále popisuji balíčky, které knihovna využívá, analýzu parametrů a třídu GlslParameter. V Návrhu uživatelského rozhraní popisuji svou představu o základním rozložení okna. Výběr knihovny pro práci s dokovatelnými panely se zabývá výzkumem a pokusy o implementaci UI. Poté se zabývám návrhem ovládacích prvků pro základní parametry, pokročilými ovládacími prvky pro kameru a časovou osu, a na závěr rozebírám závislosti mezi parametry.

Knihovna Tigr

Tato knihovna je neustále vyvíjena a rozšiřována vedoucím mé práce, Ing. Ladislavem Čmolíkem. Využívá OpenGL a a knihovny OpenGL Extremely Easy-to-use Manipulators (dále jen gleem), které obohacuje o třídy pro snazší vytvoření scény.

Knihovna tedy k funkčnosti potřebuje (importuje) sadu knihoven - JOGL 2, GLUEGEN a vecmath.

JOGL je knihovna umožňující používání OpenGL v programovacím jazyce Java, která se od roku 2010 stala nezávislým open source projektem pod BSD licencí. JOGL umožňuje využívání většiny funkcí OpenGL, dostupných programům v jazyce C skrz Java Native Interface (JNI). Nabízí tak nejen standardní funkce OpenGL, ale také OpenGL Utility (GLU), ne však OpenGL Utility Toolkit (GLUT), které nahrazuje vlastními funkcemi pro práci s okny (AWT, Swing a další sady využívající Native Window API). Má práce je vytvářena na JOGL 2, jinak řečeno pro nejlepší výsledky je doporučeno používat tuto. Nižší verzi bych nedoporučoval, ale je možné, že s novější dosáhnete stejných výsledků, i když nevylučuji potřebu menších úprav.

GLUEGEN generuje Java a JNI kód potřebný k volání funkcí jazyku C, na kterém se dá říci, že je OpenGL založený.

Vecmath obsahuje různé třídy pro objektovou reprezentaci především vektorů a matic a operace s nimi.

V knihovně také existuje prototyp parametru pro úpravu vlastností v shaderu v reálném čase a jeden jeho ovládací prvek. Jedná se o FloatParameter, zastupující v shaderu datový typ float, a takzvaný FloatSlider, jednoduchý posuvník mezi dvěma předem zadanými hodnotami, k jeho manipulaci.

Práce s knihovnou Tigr

Práce s knihovnou je poměrně jednoduchá, většina potřebných operací je již prováděna za vás, nebo alespoň značně ulehčena. Běžný postup vytvoření vlastního příkladu je takový, že do nové scény načtete trojrozměrný objekt, zadáte způsob vykreslení (pro průchody vykreslením načtete chtěné shadery, kterým poté předáte požadované parametry), a nakonec vytvoříte okno, kterému scénu předáte.

Scénu představuje instance třídy Scene, načítání objektu umožňuje třída ObjLoader, kterou načtete do Scene libovolný 3D objekt typu „obj“, ať už stažený, nebo vámi vytvořený například v Blenderu. Při načítání musíte uvést absolutní cestu k souboru.

Detaily o průchodech vykreslováním si popíšeme později, nejdůležitější pro nás je momentálně fakt, že v tomto kroku se načítají shadery a předávají do nich proměnné (parametry). Tento krok je důležitý, protože spousta efektů na scéně se provádí až při vykreslování, a pokud uvedeme v shaderu nějaký z efektů závislý na vstupu, nesmíme zapomenout k němu parametr přiřadit, jako například kdybychom řekli, že chceme možnost změny barvy světla, barva světla bude požadována jako vstup, a pokud ji nepřiřadíme, při spuštění programu nám bude ohlášena chyba a nic se na plátně nevykreslí.

Poslední krok, vytvoření okna, je pravděpodobně nejjednodušší, ale v této práci jeden z nejdůležitějších. Třída Window představuje výsledné okno, ve kterém se nachází plátno a případné ovládací prvky, a zpracovává vykreslování na okno. Ve třídě Window se totiž nachází animátor, který určuje, jak často se plátno překresluje, pokud vůbec. Díky v něm se nacházejícím ovládacím prvkům se také jedná o prostředníka mezi koncovým uživatelem a běžícím programem.

Balíčky knihovny Tigr

Knihovna obsahuje poměrně velké množství balíčků (package). Během své práce přicházím do styku prakticky se všemi, ale ne do všech zasahuji. Nejdůležitější z nich si nyní podrobně představíme.

gleem a gleem.linalg

Gleem byl již zmíněný dříve, jedná se sice o knihovnu vytvořenou Kenneth B. Russellem, avšak do projektu není naimportována jako například JOGL, ale jako balíčky s dostupnými a upravitelnými zdrojovými kódy.

Balíček gleem obsahuje především kód pro práci s kamerou ve scéně, je zde například v příkladech nejčastěji používaná třída ExaminerViewer, kterou také pro své účely používám, rozšiřuji a upravuji.

Gleem.linalg obsahuje základní, i mírně pokročilé, prvky pro reprezentaci objektů - vektory, matice, přímky, plochy a další. Tyto třídy nerozšiřují GlslProgramParameter.

tiger.core

V tiger.core se nachází třídy Window a GlslProgramParameter. Jejich důležitost pro moji práci jsme si již vysvětlili.

tiger.ui

Obsahuje implementace jednotlivých prvků uživatelského rozhraní, jako například FloatSlider, a také jejich listenery. Zmíněný je především proto, že do tohoto balíčku patří i ovládací prvky, popřípadě další s nimi související třídy, které budu vytvářet.

tiger.example

V tomto balíčku najdeme několik už hotových spustitelných ukázek, jak s knihovnou pracovat, vytvořených mým vedoucím práce. Zároveň obsahuje shadery, potřebné k jejich spuštění.

Parametry

Parametry, které nejčastěji využívá OpenGL a které bychom chtěli mít možnost ovládat, se nacházejí v gleem.linalg.

V knihovně se také nachází abstraktní třída GlslProgramParameter, která se rozšiřuje konkrétními implementacemi pro různé datové typy. Tyto parametry jsou důležité, chovají se jako prostředníci mezi ovládacími prvky, shadery a scénou. Zjednodušeně řečeno jsou to takové trezory, do kterých ukládáme hodnoty k pozdějšímu výběru, a při jakékoliv změně na ni upozorní.

Parametry v gleem.linalg

Vektory najdeme v tomto balíčku jako třídy Vecf, Vec2f, Vec3f, Vec4f,Vec3d a Veci, kde číslo určuje počet prvků vektoru, bez čísla není počet předem určen. Konečné písmenko určuje, jaký datový typ vektor obsahuje, f pro float, d pro double a i pro integer. Hodnoty jsou ukládány v poli příslušného datového typu. Třídy obsahují metody pro práci s vektory - sčítání, odčítání, násobení vektorem i konstantou atd. Vecf lze za pomocných metod zkopírovat a vytvořit s jeho hodnotami instanci vektoru s předem určenou délkou, tedy Vec2f, Vec3f a Vec4f. Lze použít, pouze pokud má vektor instance Vecf délku jako jedna z konkrétnějších implementací, tzn. Vecf s třemi prvky lze převést pouze do Vec3f. Vec2f, Vec3f a Vec4f lze zpět převést do obecnějšího Vecf pomocí metody toVecf. Třídy se vzájemně nijak nerozšiřují.

Pro znázornění matice jsou zde třídy Matf, která zastupuje matici obecně, a Mat2f, Mat3f a Mat4f, určené pro matice o velikosti n*n, kde n je číslo následující Mat a předcházející f. Písmenko f znamená, že hodnoty v matici jsou typu float. Matf má metody toMat2f, toMat3f a toMat4f, která vrátí novou instanci požadované třídy (velikosti). Při zavolání dané metody se zkontroluje, že matice, na které je metoda volána, je požadovaných velikostí, například při zavolání toMat2f matice musí být rozměrů 2x2, a pokud není, hodí program chybu DimensionMismatchException. Hodnoty se do nově vytvořené matice z originálu zkopírují. Naopak třídy Mat2f, Mat3f, Mat4f mají metodu toMatf, která vrátí novou instanci obecné matice, velikostí i hodnotami kopírující originál. Třídy se vzájemně nijak nerozšiřují. Hodnoty jsou v maticích reprezentovány polem floatů.

Rotf reprezentuje rotaci jako kvaternion. Kvaternion se skládá ze čtyř čísel typu float. Podle dokumentace nultý prvek reprezentuje skalár a první až třetí vektor. Tato rotace se využívá v kameře.

GlslProgramParameter

Instance GlslProgramParameteru vyžaduje při vytváření jako vstup v textovém řetězci zadané jméno, které musí být shodné se jménem proměnné, kterou zastupuje v shaderu. Pokud není parametr využitý v shaderu, může mít jméno libovolné. GlslProgramParameter dále obsahuje metody pro

předávání do shaderu, metodu init a abstraktní metodu initValue, která poté u každého parametru vyžaduje korektní implementaci; a také metody na přidání listenerů. Listenery jsou pro parametr důležité, aby bylo možné při změně jejich hodnoty aktualizovat hodnoty v jejich ovládacích prvcích.

Konkrétní implementace GlslProgramParametru mohou být poté rozšířeny ještě o sadu metod pracující s parametry a jejich hodnotami, jako například getValue a setValue, které vrátí, popřípadě nastaví hodnotu parametru, nebo nějaké složitější, jako například násobení vektorů.

Nejdůležitější prvky v mé práci jsou parametry, které dokážou v nich uloženou hodnotu na zažádání předat shaderům, a je možné jejich hodnoty za běhu programu měnit. Rozhodl jsem se, že všechny tyto parametry budou rozšiřovat třídu GlslProgramParameter.

Rozšiřováním třídy GlslProgramParameter získají parametry tyto vlastnosti:

- možnost pojmenování - důležité pro předávání do shaderů
- množinu listenerů a možnost do ní libovolně listener přidat a opětovně odebrat - důležité pro upozorňování ovládacích prvků a skupin závislých parametrů na změny
- implementují metodu initValue, která předává hodnoty shaderu

Návrh parametrů

Mnou vytvořené parametry musí fungovat jako prostředníci, místo, do kterého je možné kdykoliv zapsat hodnotu, nebo ji z něj přečíst. Řekl bych, že toto je důležité pro napojení parametru na ovládací prvek. Proto jsem se rozhodl o implementace parametrů pomocí třídy GlslProgramParameter.

Každý ovládací prvek bude vyžadovat při vytvoření parametr, který bude upravovat, a při zaregistrování změny na ovládacím prvku se patřičná změna podle konkrétní implementace ovládacího prvku poté projeví na v něm uloženém parametru.

V místech, kde se parametry využívají, se musí poté pracovat s jejich metodami getValue a setValue. Důraz je kladen obzvláště metodu setValue, kterou použijeme, pokud chceme změnit hodnotu parametru. Důležité je využít tuto metodu, protože nesmíme přepsat odkaz na parametr novým. Pokud bychom přepsali odkaz, ztratilo by se pouto mezi ovládacím prvkem a v programu využívanou hodnotou. Ovládací prvek by poté upravoval jemu předaný parametr a, zatímco program by se snažil pracovat s hodnotami parametru b, který by již nic ze strany uživatele používající uživatelské rozhraní neovlivňovalo.

Základem všeho je vytvoření parametru pro float (FloatParameter) a integer (IntegerParameter). Tyto třídy přímo rozšiřují GlslProgramParameter a mají v sobě proměnnou jménem value patřičného

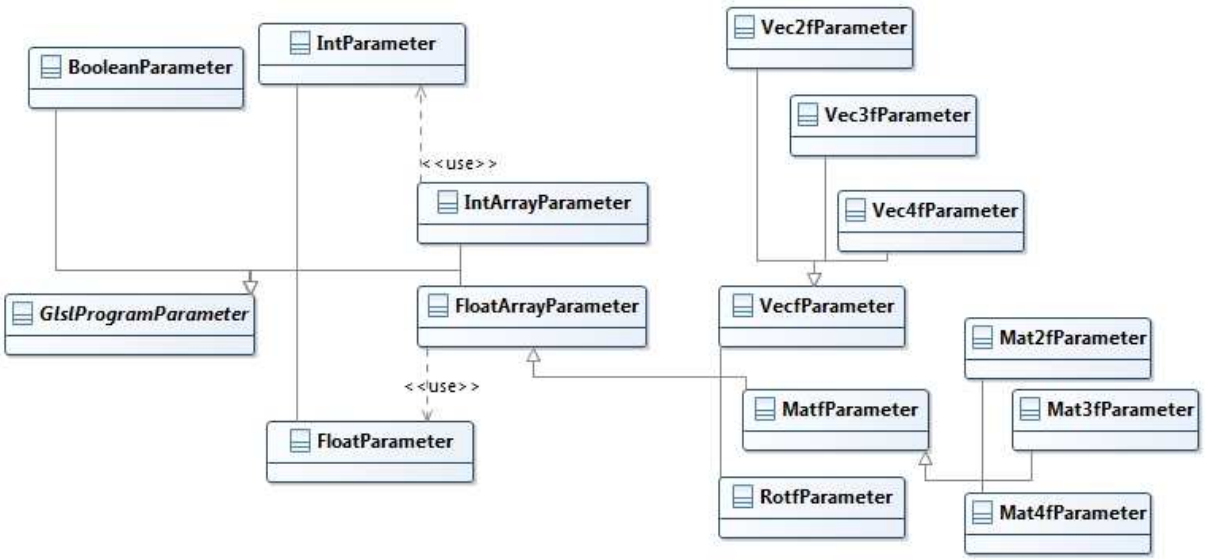
datového typu pro uložení hodnoty. Při použití metody setValue dojde k přepsání hodnoty v proměnné value, potom dojde k ohlášení změny postupně na všech listenerech, přidaných k parametru metodou addChangeListener.

Jakousi nástavbou jsou pole zmíněných parametrů FloatArrayParameter a IntArrayParameter. Tyto parametry sice také přímo rozšiřují GlslProgramParameter, ale hodnoty neuchovávají v prostém poli floatů nebo integerů, nýbrž v poli FloatParameterů a IntParameterů. Toto nám dá možnost manipulovat s každým prvkem tohoto pole jako samostatným parametrem. Pokud se přidá listener na parametr reprezentující pole, přidá se na všechny parametry v něm, protože změna kteréhokoliv z nich ovlivní celé pole, ale je možné zažádat si o určitý parametr v poli a na něj samotný přidat vlastní listener, ať už z jakéhokoliv důvodu. Tato vlastnost je určená především pro závislosti mezi parametry. Pokud by vám šlo o vzájemný vztah každé hodnoty v parametru, tak jej to převede na úroveň vztahu mezi jednotlivými parametry.

Vektory, matice a rotace jsou konkrétními případy využití pole floatů. Z tohoto důvodu jsem se rozhodl, že VecfParameter, MatfParameter a RotfParameter budou rozšiřovat FloatArrayParameter a pouze ho obohatí o specifické metody pro práci s nimi, jako například vzájemné násobení vektorů, a v případě matic upraví metodu initValue pro předávání shaderům.

Ve vzoru k implementacím vektorů a matic konkrétních velikostí jsou implementovány i takovéto parametry, na rozdíl od originálů však rozšiřují obecné parametry, tudíž Mat2fParameter, Mat3fParameter a Mat4fParameter rozšiřují MatfParameter.

BooleanParameter je poněkud jednoduchý, rozšiřuje GlslProgramParametr, uchovává boolean s jeho hodnotou. Jeho zvláštností je, že boolean se předává do shaderů jako integer s hodnotou 0 pro nepravdu (false) nebo 1 pro pravdu (true).



Obrázek 1 - Návrh implementace parametrů

Na obrázku (viz. Obrázek 1) můžete vidět UML diagram mého návrhu implementace parametrů a vztahy mezi nimi.

Rozšiřitelnost parametrů

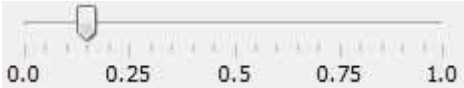
V balíčku se nacházely i další třídy - Line, Plane, PlaneUV a IntersectionPoint. Ve své práci se ale spíše snažím rozšířit stávající stav, ve kterém se knihovna nachází, a vzhledem k tomu, že tyto třídy se v projektu nikde nepoužívají, nejsou příliš důležité pro implementaci. Je pro mne lepší se zaměřit na používané prvky než vytvářet situace, ve kterých bych mohl využít nové. Tím nechci říct, že jsou tyto parametry nepoužitelné.

V rámci práce se musím zamýšlet i na možností snadné rozšiřitelnosti a řekl bych, že tyto parametry jsou dobrou ukázkou možné rozšiřitelnosti implementací mých parametrů. Například Line využívá tří vektorů, takže pokud vytvořím parametr pro zastoupení vektoru, může budoucí uživatel, který by potřeboval třídu Line, už využít mých implementací vektorů, které by mu měly práci značně usnadnit.

Návrh ovládacích prvků

Každý parametr, který bychom chtěli za chodu programu upravovat, by měl být spárovaný s nějakým ovládacím prvkem. Základní ovládací prvky jsou takové, která pracují s parametry IntParameter, FloatParameter, IntArrayParameter a FloatArrayParameter v takové podobě, že uživatel upravuje přímo jejich hodnoty. Je dobré podotknout, že ovládací prvky pro FloatArrayParameter fungují i pro parametry, které ho rozšiřují, tudíž VecfParameter, MatfParameter, RotfParameter a jejich potomci.

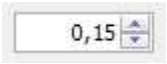
Pro číselné hodnoty je možné použít třeba posuvník. Ten je nejvhodnější pro parametry s předem určeným rozsahem, kde při manipulaci s ním nepotřebujete naprosto přesnou hodnotu, ale chcete mít možnost rychle ji měnit, bez zadávání hodnot nebo nekonečného klikání na šipky. Implementace využívá javax.swing.JSlider (viz. Obrázek 2).



Obrázek 2 - javax.swing.JSlider

Uživatel ale s největší pravděpodobností bude chtít zadat některé hodnoty v určitých případech s větší přesností než například desetiny či setiny, a proto je důležité vytvořit ovládací prvek, který mu umožní zadat hodnotu na klávesnici přesně. Jde o implementaci javax.swing.JSpinner (viz. Obrázek 3), a i v tomto případě je důležité dodržovat minima a maxima pro parametr.

Možnost zadávat hodnoty ručně by měla být dostupná jak pro floaty, tak pro integery.



Obrázek 3 - javax.swing.JSpinner

Pro booleany je nejjednodušší vytvořit ovládací prvek v podobě checkboxu (zaškrtávátko, viz. Obrázek 4), neboť boolean nabývá pouze hodnot pravda, nepravda, stejně jako může být políčko checkboxu zaškrtnuto nebo prázdné. Alternativa by byl optionbox, ale ten v lidech vzbuzuje v uživatelském rozhraní dojem, v případě, že jich je v jednom uskupení ovládacích prvků více, že vždy bude zaškrtnut právě jeden takovýto prvek, což většinou nebude pravda, pouze pokud by šlo o nějaké na sobě závislé parametry.

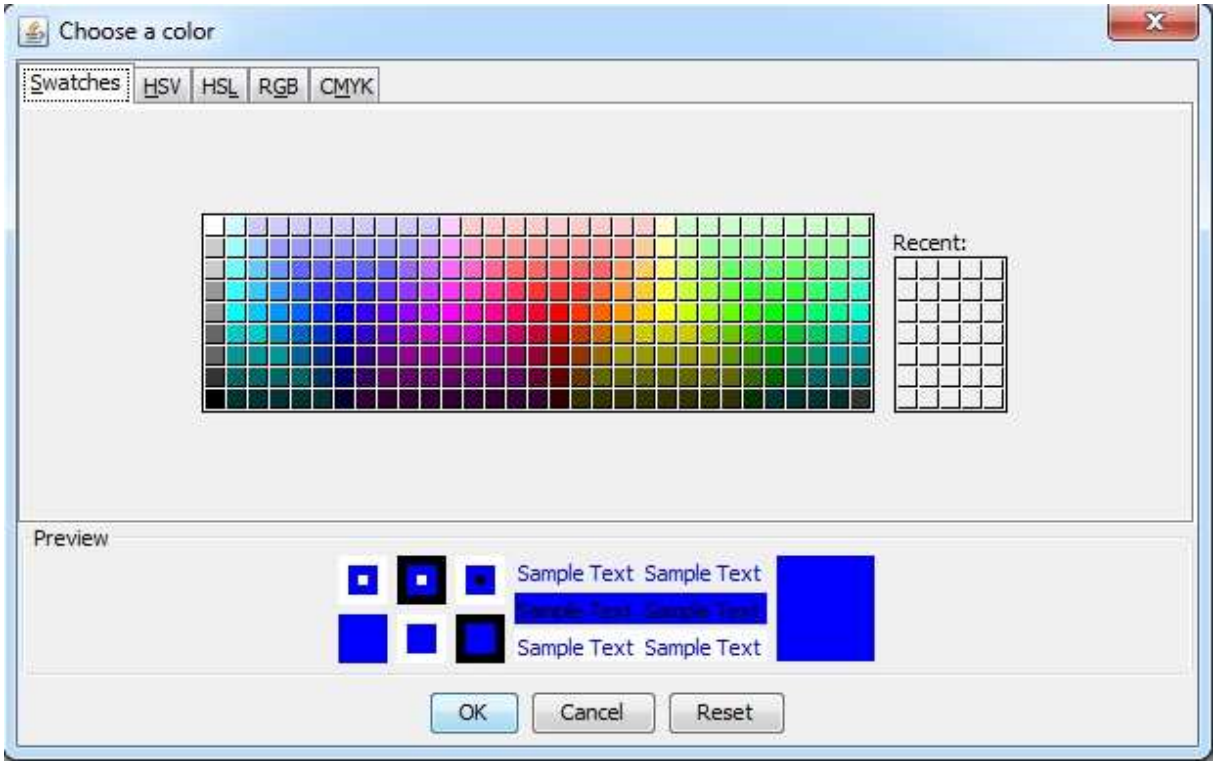
Obrázek 4 - JCheckBox

Pokročilé ovládací prvky

Pokročilé ovládací prvky ovládají tytéž parametry jako základní, s tím rozdílem, že je u nich složitější si představit, jak se změny hodnot v parametrech projeví ve výsledku. V případě mých konkrétních implementací se jedná o ovládání barvy a kamery. Speciální je časová osa, která na zažádání, například od okna, upraví hodnoty požadovaných parametrů.

Barva

Barva je reprezentována v OpenGL polem tří nebo čtyř floatů, lépe řečeno vektorem o délce tří nebo čtyř prvků, kde jedna složka zastupuje jednu složku z RGB, popřípadě RGBA - red, green, blue, alpha. Ovládat ji stejně jako normální floaty je možné, ale poměrně nepraktické, proto je pro výběr barvy vytvořen prvek, který umožní vybrat barvu z palety, a to pomocí javax.swing.JColorChooser. Pro výběr barvy stačí stisknout tlačítko a v samostatném okně vyskočí paleta, kde se dá různými způsoby vybrat barvu. Po potvrzení výběru se barva pomocí metody Color.getComponents přepočítá na čtyři floaty a je jimi vyplněn parametr.



Obrázek 5 - Vyskakovací okno pro výběr barvy

Jak je vidět na Obrázku 5, vyskakovací okno má více záložek a opravdu poskytuje širokou paletu barev.

Kamera

Ovládání kamery je složitější. Vzhledem k tomu, že všechny údaje o ní jsou reprezentovány vektory, maticemi a rotacemi, je možné ji ovládat ručními vstupy, ale někdo, kdo by ji jimi byl schopný ovládat, by musel být extrémně trpělivý nebo mít výbornou představivost a znalost tématu. Pro obyčejného uživatele je proto potřeba vytvořit k jejímu ovládání něco intuitivního, a proto jsem se rozhodl pro ideu mého vedoucího - pokusit se vytvořit pro ovládání rotace kamery trackball a její pozice trackpad.

Pro ovládání kamery jsem vybral ExaminerViewer. Popsat jak přesně funguje, není cílem mé práce, ale bylo pro ni poměrně důležité, abych ho pochopil. Co stojí za představení, jsou jeho složky Vec3 dolly, Vec3f center, Rotf orientation a CameraParameters params. Center jsou souřadnice xyz, ve kterém se nachází fokální bod, dolly je vzdálenost, z jaké se díváme na fokální bod, a orientation je orientace (natočení) kamery. CameraParameters slouží k ukládání vypočítaných dat potřebných pro finální zobrazení, konkrétně modelovou a projekční matici.

Aby bylo možné ovládat parametry kamery, je potřeba předělat i viewer používaný v projektu. Vytvořil jsem nové třídy, pojmenované TigerExaminerViewer a TigerCameraParameters, obě po funkční stránce shodné s normálním ExaminerViewerem a CameraParameters z knihovny, s tou výjimkou, že veškeré parametry v nich byly upraveny na danou implementaci GlslProgramParametrů.

Toto umožňuje přiřazení ovládacích prvků jednotlivým parametrům a zároveň vytvoření ovládacích prvků pro celý TigerExaminerViewer.

CameraControls je abstraktní třída pro plánované ovládací prvky. Rozšiřuje obyčejný javax.swing.JPanel a obohacuje ho o na něj automatické přidání mouseListenerů.

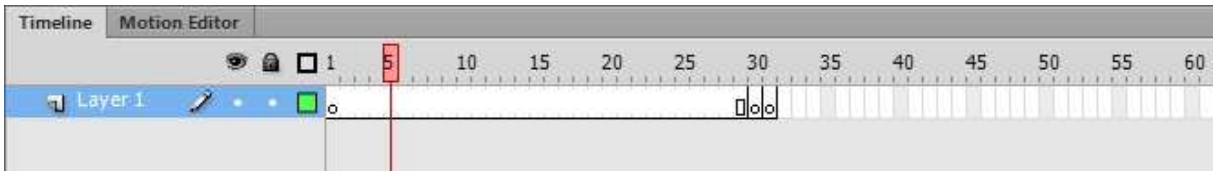
Časová osa

Součástí zadání je vytvořit časovou osu, která umožní definovat a přehrát změnu hodnot parametrů v čase.

Plánuji, že časová osa bude rozdělena na jednotlivé snímky (frame), uživatel před spuštěním určí, kolik by chtěl, aby jeho osa měla snímků, za chodu pak může určit FPS (frames per second, snímky za vteřinu) a opakování určitého úseku. Na ose bude možnost vytvořit klíčový snímek (keyframe), na kterém bude při vybrání parametru možné jeho hodnotu upravovat.

Samozřejmě nesmí chybět možnost přehrávání kdykoliv spustit a zastavit, typická tlačítka play/stop.

Inspiraci pro vzhled, k nahlédnutí na Obrázku 6, a chování osy jsem převzal z Adobe Flash Proffesional CS6.



Obrázek 6 - Výřez z uživatelského rozhraní Adobe Flash Proffesional CS6

Časová osa potřebuje největší přípravu od uživatele před spuštěním programu. Získat všechny proměnné z programu je nemožné, proto se mi zdá nejrealizovatelnější, aby parametry, které v čase bude potřeba upravovat, vybral programátor před spuštěním programu a předal je časové ose.

Protože je možné, že uživatel bude chtít různé implementace chování parametru v čase, je nejlepší, aby klíčové snímky na sebe měly vzájemné odkazy, a způsob, jakým se hodnota na snímku f dopočítá. Z tohoto důvodu je důležité, aby pro každý parametr nebo alespoň typ byl vytvořen klíčový snímek, který toto vše zajistí.

Proto je v knihovně vytvořena abstraktní třída TigerParameterKeyframe, jejíž konkrétní implementace se předávají časové ose. Zároveň je potřeba, aby se při vytvoření nového klíčového snímku vytvořila instance správné implementace, proto se časové ose k jednotlivým parametrům přiřazuje i továrna na správné keyframy. Pro tyto továrny je vytvořeno rozhraní KeyframeFactory. Správnou továrnu musí k parametru opět přiřadit programátor před spuštěním.

Konkrétní implementaci továren a klíčových snímků si ukážeme v kapitole implementace.

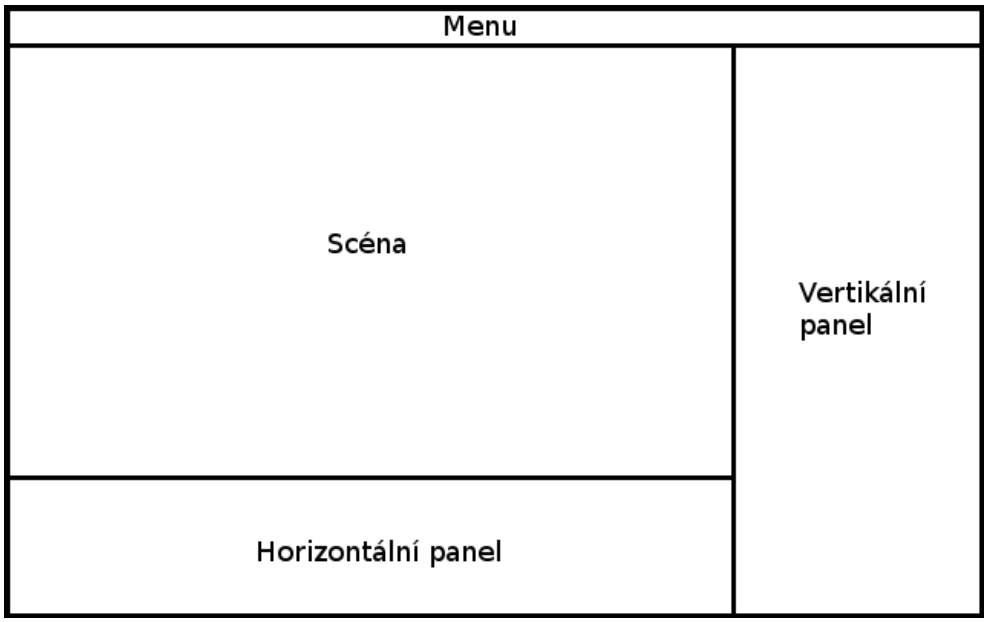
Závislosti mezi parametry

Někdy je potřeba nebo alespoň žádoucí vytvořit mezi parametry nějaký typ závislosti, například jen jeden boolean může mít hodnotu true, nebo součet skupiny floatů musí být roven jedné. Implementace něčeho takového je poměrně složitá, nejrozumnější řešení je vytvořit jakési pouzdro pro skupinu prvků, které na každý z nich navěsí listener, kontrolující jejich stav a případně upravující ostatní prvky skupiny. Takovýto listener ale bude muset uživatel pokaždé naimplementovat sám, pro svůj konkrétní příklad, neboť není možné vymyslet jeden univerzální.

V rámci výsledků vytvořím jeden instruktivní příklad na tyto závislosti.

Návrh uživatelského rozhraní

Základní požadavky na uživatelské rozhraní projektu vymezují podmínky, že v okně se musí nacházet místo na plátno, na které se vykresluje scéna, a nějaký panel, nebo více, na které se umístí ovládací prvky. Vzhledem k možnosti vytáhnout panely z okna jsem se rozhodl, že v základě stačí pevné panely v okně dva, jeden horizontální a jeden vertikální. Přibližný náčrt na Obrázku 7

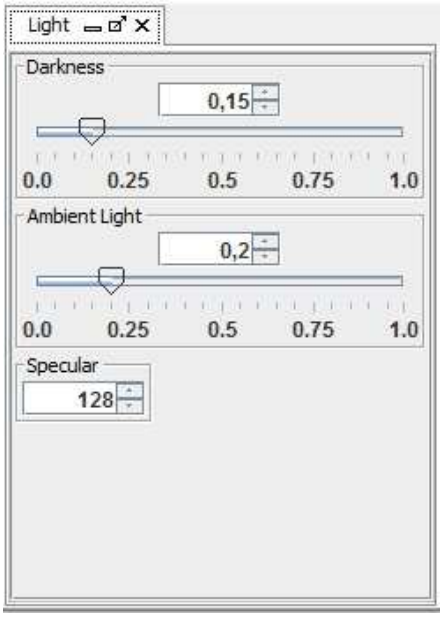


Sdružení ovládacích prvků

V uživatelském rozhraní by také měla existovat možnost sjednotit určené ovládací prvky. Základní roztřídění bude na záložky, ale i v rámci záložek může být potřeba kvůli přehlednosti prvky rozdělit do skupin.

Toto je realizováno obyčejným JPanelem s tenkými, ale viditelnými hranami. Pro zpřehlednění kódu je toto implementováno ve třídě ComponentGroup, kterou poté uživatel používá místo JPanelu, ale je možné na něm volat jakékoliv jeho metody a upravit si ho pro vlastní použití.

Skupinu prvků je také možné pojmenovat. Obrázek 8 je náhled na již implementovaný výsledek takového seskupení, vytvořeného v rámci jednoho z příkladů.



ení ovládacích prvků do
a na záložku

Výběr knihovny pro práci s dokovatelnými panely

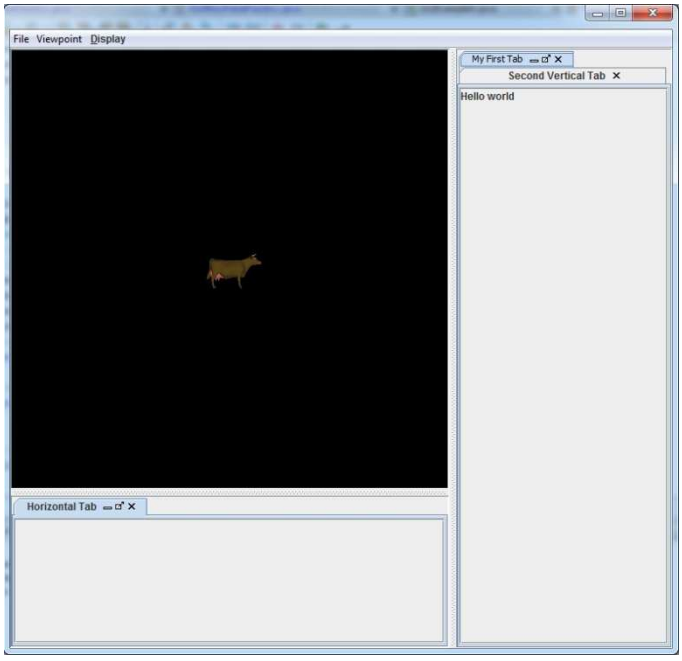
Jedna z částí projektu je vytvoření dokovatelných panelů pro UI, aby bylo možné si za chodu programu ovládací prvky přemístit podle potřeby, popřípadě zavřít s možností pozdějšího opětovného otevření.

Při pokusech o napsání vlastní implementace a hledání možností realizace narazíte už na hotové knihovny, které vám obdobné věci umožní, proto jsem se rozhodl, že budu využívat už nějakou z vytvořených knihoven, neboť výběr byl poměrně bohatý.

Od knihovny v rámci projektu chci především, aby byla pod licencí, která mi pro nekomerční účely nabídne své služby zdarma (GNU General Public License, GPL), a umožňovala mi vytvářet přesunovatelné dokovatelné panely nebo záložky, s možností uzavření nebo odepnutí z hlavního okna.

Pod těmito požadavky jsem našel nejpřijatelnější 3 knihovny fungující v Javě - MyDoggy[1], InfoNode[2] a Sanaware Java Docking[3]. S každou knihovnou jsem si zkusil vytvořit jednoduché uživatelské rozhraní podle ukázek poskytnutých autory, abych zjistil, jak těžké je s nimi pracovat a co mi knihovny dovolí.

Nakonec jsem pro své účely shledal nejlepší ze jmenovaných knihovnu Sanaware Java Docking (ukázka z implementace mého návrhu je na Obrázku 9), která je poměrně jednoduchá k použití, poskytuje uživateli dobrou sadu ovládacích prvků a u dokovacích panelů se dá omezit, kde všude je můžete umístit, což se mi například u MyDoggy nedaří implementovat, a i když to nezní moc jako nevýhoda, není nejlepší dávat uživateli naprostou volnost nad UI. Navíc se mi zdály ukázky a dokumentace JavaDockingu nejsrozumitelnější a vždy demonstrovaly různé situace, některé obdobné těm mým.



Obrázek 9 - Vzhled uživatelského rozhraní využívající Sanaware Java Docking

[1] http://mydoggy.sourceforge.net/
[2] http://www.infonode.se/index.html?idw
[3] http://www.javadocking.com/

Implementace

Rozhodl jsem se navrhnout strukturu tak, aby nevzniklo příliš redundantního kódu. Nejdůležitější je propojení mezi ovládacími prvky a parametry, takže prvním úkolem je, jak mají být udělané parametry.

Parametry

Abstraktní třídu GlslProgramParameter jsem obohatil o abstraktní metodu clone.

Pokud přímo rozšiřujete GlslProgramParameter, je důležité naimplementovat obdobnou metodu, jako naleznete pod tímto odstavcem. Zde je uvedený setValue, kde PRIMITIVE je v tomto případě jakékoliv primitivum. Není nutné, aby byl nový parametr založený na primitivu, ale kód, jak jsem ho uvedl v Javě, bude fungovat pouze pro ně. Tato metoda zajistí, že všechny listenery navěšené na parametru budou upozorněny na změnu jeho hodnoty, proto je potřeba zároveň měnit hodnotu parametru vždy metodou, která přesně toto udělá.

```
public void setValue(PRIMITIVE value) {
        this.value = value;
        ChangeEvent e = new ChangeEvent(this);
        for(ChangeListener l : listeners) {
            l.stateChanged(e);
        }
    }
```

IntParameter

Parametr uchovávající hodnotu integer (celé číslo). Konstruktor vyžaduje dvě vstupní proměnné, první je povinná pro GlslProgramParameter, textový řetězec obsahující jméno (name), druhá je hodnota, která má být při inicializaci do parametru uložena.

Vzhledem k možnosti ovládání jednoho parametru více ovládacími prvky je nutné, aby parametr při každé změně upozornil všechny v něm uložené listenery, že nastala změna. Toto se děje při zavolání metody setValue po přiřazení nové hodnoty do parametru, proto je důležité vždy měnit hodnotu v parametru touto metodou!

BooleanParameter

Parametr reprezentující boolean - logický datový typ. Stejně jako v případě IntParametru je vstupem jméno a hodnota. Opět je nutné měnit hodnotu metodou setValue. Zvláštnost na booleanu je, že se do shaderů předává jako integer, hodnota 1 zastává pravdu (true), 0 zase nepravdu (false).

FloatParameter

Parametr zprostředkovávající hodnotu jednoho floatu. Metody ve FloatParameteru jsem od obdržení neupravoval, pouze jsem přidal implementaci metody clone. Vstupní parametry konstruktoru jsou jméno a hodnota.

IntArrayParameter

Uchovává pole IntParametrů, i když velikost pole je neomezená, do shaderu se předávají vždy maximálně čtyři prvky, a pokud se uživatel rozhodne v parametru ukládat větší pole než o délce čtyř, musí poté nastavit délku (length) metodou setLength a číslo prvního prvku (offset) metodou setOffset, aby Parametr věděl, kolik prvků a odkud předat. Length může nabývat hodnot mezi 1 a 4 včetně a offset jako ukazatel v poli začíná na 0 pro první prvek.

IntArrayParametr nabízí dva konstruktory, jeden pouze se jménem a polem integerů obsahujícím hodnoty. Tato možnost si doplní délku na délku vloženého pole a offset na 0, vhodné pro vytváření parametrů, pokud máte pole s počtem prvků splňující zmíněné podmínky, a také konstruktor požadující jméno, hodnoty, číslo prvního prvku a délku.

Metoda setValue se také dělí podle vstupu. Pokud zadáte pouze pole integerů s prvky, přepíší se hodnoty v parametru, ale pouze pokud je vstupní pole stejně dlouhé, jako pole s vnitřními parametry. Alternativou je stejnojmenná metoda, jejímž vstupem je pole prvků, číslo prvního prvku a délka. Tuto metodu nedoporučuji používat, pokud se Parametr nepoužívá pouze vnitřně, protože ovládací prvky jsou vázány na vnitřní parametry a tato metoda všechny vnitřní parametry nahradí novými.

Metoda getValue(int position) vrátí IntParameter uložený v poli na pozici position. Pozice v poli začíná číslem 0.

FloatArrayParameter

Uchovává pole FloatParameterů. Stejně jako IntArrayParameter je jeho velikost neomezená, maximální počet předaných prvků jsou čtyři, vymezený délkou length a číslem prvního prvku offset.

Metoda getValueAtPosition(int position) vrátí FloatParameter na pozici s číslem position, začínajícím na 0 pro první prvek, getValueAtPositionAsFloat(int position) vrátí z pozice pouze hodnotu typu float. Pokud chcete z nějakého důvodu hodnoty rovnou jako pole floatů, je tu pro vás metoda getValueAsFloatArray(), která přesně toto udělá.

Tento parametr dále rozšiřují třídy MatfParameter, RotfParameter a VecfParameter.

VecfParameter

Rozšiřuje FloatArrayParameter a využívá i jeho metodu na předávání do shaderů, neboť neexistuje speciální předávání pro vektory. Vektory se tedy sice předávají jako pole floatů, v shaderu se už ale o nich mluví jako o vektorech.

Konstruktor má čtyři alternativy, s tím, že ho můžete vytvořit zadáním délky požadovaného vektoru nebo jiného Vec3fParametru, přičemž jsou zkopírovány všechny jeho vlastnosti. Další alternativy jsou stejné jako předchozí, jen je jako vstupní parametr přidán řetězec pro jméno.

Metoda get(int i) vrátí hodnotu na i-té pozici jako float, set(int i, float val) zapíše hodnotu val na pozici i. Pokud chcete délku, v tomto případě to znamená jako u FloatParameteru počet prvků vektoru.

VecfParameter je možné převést podle potřeby do jiných vektorových parametrů pomocí metod toVec(počet prvků vektoru)(Parameter), kde konverze nemusí být do implementace GlslProgramParametru. Výsledný vektor je kopií, která neobsahuje žádné listenery originálu.

MatfParameter

MatfParameter rozšiřuje FloatArrayParameter. Minimální požadovaný vstup pro vytvoření jsou dva integery nRow a nCol, počet řádek a počet sloupců. Hodnoty se uchovávají v poli FloatParametrů o velikosti nRow * nCol. Matici je možné přiřadit jméno.

Kromě setterů a getterů z FloatArrayParameteru má tato třída i možnost získání či zapsání hodnoty pomocí metod set a get, pracujícími se souřadnicemi řádků a sloupců v matici.

Metoda transpose transponuje hodnoty přímo v matici, createTranspose vytvoří novou, transponovanou matici.

Matici je možné vynásobit, dělá se to metodou mul, která má dvě alternativy. První možnost je se vstupem jiné matice, kde výstupem je úplně nová matice s hodnotami matice A*B, kde A je matice, na které byl mul zavolán, a B je matice uvedená jako vstupní parametr. Druhá možnost je násobení vektorem, výstup je opět vektor. Pro násobení matice platí všechny podmínky jako v lineární algebře.

MatfParameter je dále rozšiřován konkrétními implementacemi Mat2fParameter, Mat3fParameter a Mat4fParameter.

U této třídy bylo nutné přepsat initValue, protože matice se předává do shaderů trochu jinak než obyčejné pole floatů.

RotfParameter

Rozšiřuje FloatArrayParameter. Rotace je reprezentována polem FloatParameterů o délce čtyř prvků, s výchozí hodnotou nastavenou na (1, 0, 0, 0), jedná se o kvaternion.

Konstruktorů naleznete v RotfParameru opravdu mnoho. Vytvořit ho můžete bez jakéhokoliv vstupního parametru za pomoci jiné rotace, polem floatových hodnot (o délce 4), vektorem o třech prvcích a úhlem nebo dvěma vektory o třech prvcích, a to počáteční a koncový vektor. Existují i obdoby těchto konstruktorů se vstupem textového řetězce pro jméno. Předávání do shaderů je stejné jako u FloatArrayParameterů.

Třída obsahuje i několik setterů, prakticky všechny jsou odvozené z konstruktorů, takže je ve skutečnosti jedno, jestli vytvoříte rotaci s prázdným vstupem a poté ji nastavíte, nebo rovnou vytvoříte se všemi vstupními parametry. Jediným takovým bonusovým setterem je metoda fromMatrix, se dvěma možnými vstupy, buď Mat4f, nebo Mat4fParameter, která nastaví rotaci podle Matice 4*4. Z této matice se ve skutečnosti využije pouze levá horní část o velikosti 3x3. Je také naopak možné z každé rotace vytvořit novou matici 4*4.

Metoda length funguje u rotace trochu jinak, nevrací počet prvků (což je mimo jiné poměrně zbytečné, protože vždy bude tvořena čtyřmi prvky), ale místo toho délku vektoru.

Rotace nabízí dvě možnosti násobení, times(RotfParameter b), která vrátí novou rotaci, ve které je výsledek po vynásobení rotace, na které je metoda volána, rotací b (v tomto pořadí), a metoda mul(RotfParameter a, RotfParameter b), která do rotace, na které je zavolána, uloží výsledek a*b (opět v tomto pořadí).

Metoda slerp(RotfParameter B, double t) slouží k interpolaci dvou rotací. Interpoluje se mezi rotací, na které je slerp zavolán, a rotací B. Double t je číslo mezi 0 a 1, které udává podíl rotací na interpolaci. Kdyby se t rovnalo 0, je výsledek interpolace rotace, na které je metoda volána, pokud 1, je výsledek rotace B.

Závislosti

Závislost jsem nedokázal vymyslet tak, aby nevyžadovala poměrně velkou spolupráci od programátora.

V projektu jsem vytvořil třídu ParameterGroup a rozhraní ParameterGroupAction s jednou metodou performGroupOperation se vstupy GlslProgramParameter source a List<GlslProgramParameter> groupMembers.

Základní ideou je, že programátor vytvoří závislost následujícím způsobem. Vytvoří vlastní instanci třídy ParameterGroup, do které poté přidá metodou addParameter GlslProgramParameter libovolný počet parametrů, které chce v závislosti. Poté musí vytvořit vlastní implementaci ParameterGroupAction a její metody performGroupOperation, která nějak implementuje vztah mezi parametry. Ukázku takovéto implementace naleznete v kapitole Implementace. Tuto implementaci poté předá metodou addGroupAction své instanci ParameterGroup.

Vnitřně třída funguje tak, že na každý parametr vložený do ParameterGroup se přidá instance vnitřní třídy ParamChangeListener (implementace ChangeListeneru), která zajistí, že pokud se nějaký parametr změní, zavolá se metoda performGroupOperation instance uživatelem implementované třídy ParameterGroupAction.

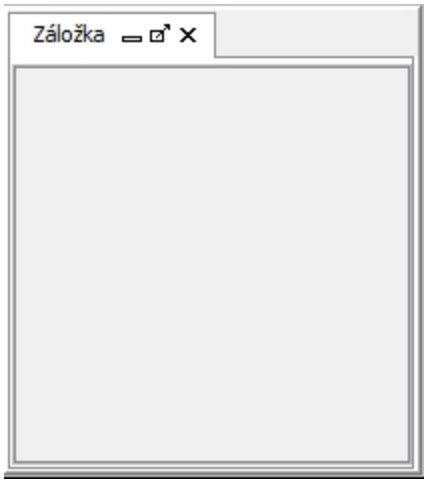
Uživatelské rozhraní a ovládací prvky

Tato podkapitola se zaměřuje na výsledné GUI, ovládací prvky a k čemu slouží. Veškeré ovládací prvky pro nějaký GlslProgramParameter mají vnořenou třídu ParamChangeListener, implementující ChangeListener. Při vytvoření ovládacího prvku se na GlslProgramParameter, pro který je určen, přidá tento ParamChangeListener, takže pokud se změní hodnota v parametru, zavolá se metoda stateChanged instance tohoto listeneru a provede se, co je v jejím těle. V těle této metody je vždy kód, který nastaví ovládací prvek na hodnotu uloženou v parametru. Toto je důležité, aby se jakákoliv změna v parametru projevila i na ovládacím prvku, protože parametr může být ovládán několika prvky, nebo je možné ho měnit i v kódu. Zároveň má každý ovládací prvek svůj vlastní listener, který mu naslouchá, a při jeho změně změní hodnotu parametru. V kódu jsou i určitá zaopatření proti zacyklení.

DraggableTab

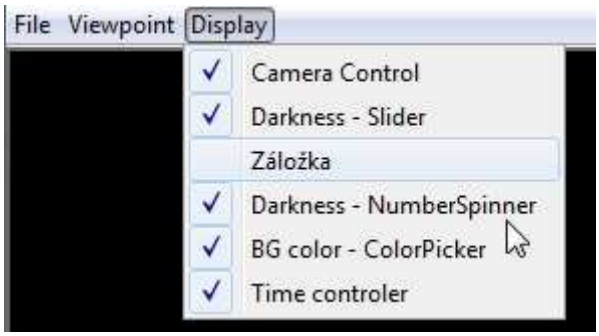
Základním prvkem uživatelského rozhraní jsou záložky. Třída DraggableTab (viz. Obrázek 10) rozšiřuje JPanel

a implementuje DraggableContent z knihovny JavaDocking. Záložku můžete pojmenovat, dokonce je to doporučeno kvůli orientaci. Pokud ji nepojmenujete, bude pojmenována „Untitled“ za vás. Záložka jinak funguje jako obyčejný JPanel, proto je možné ji upravit podle vašich představ. Prázdnou záložku, na které jsou nastaveny pouze jméno a rozměry, je vidět na obrázku.



Obrázek 10 - Záložka

Záložky je možné úplně zavřít nebo minimalizovat. Zavřené záložky lze obnovit z menu, kde naleznete seznam všech záložek pod položkou Display (viz Obrázek 11). Minimalizované můžete maximalizovat z lišty, která se objeví ve spodní části okna (viz Obrázek 12).



Obrázek 11 - Menu

Obrázek 12 - Lišta s minimalizovanými prvky

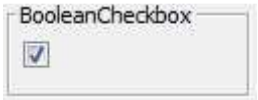
ComponentGroup

ComponentGroup je třeba nezaměňovat s ParameterGroup. Slouží k uskupení a pojmenování vašich ovládacích prvků na záložce pro přehlednost. Jedná se o obyčejný JPanel s přednastavenými hranami a popiskou, kterou při vytváření zadává uživatel. Prázdný ComponentGroup je vidět na Obrázku 13.

Obrázek 13 - ComponentGroup

BooleanCheckbox

Jednoduchý checkbox pro hodnoty typu boolean - pravda, nepravda. Při vytváření musíte zadat BooleanParameter, který bude ovládací prvek upravovat. Na Obrázku 14 je vidět BooleanCheckbox s hodnotou nastavenou na pravdu (true, zaškrtnuto) v ComponentGroup s názvem BooleanCheckbox.

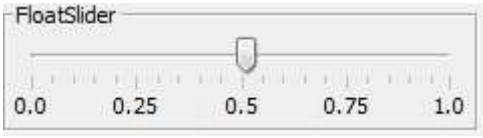


Obrázek 14 - BooleanCheckbox

IntSlider a FloatSlider

Posuvníky pro parametry IntParameter a FloatParameter. Tyto ovládací prvky jsou užitečné, pokud potřebujete nebo chcete rychle změnit hodnotu parametru, ale nejde vám o přesné číslo, spíše o nějaký výsledný dojem efektu na obrazovce. Zároveň je určen pro parametry, kde počítáte s určitým minimem a maximem.

Vstupem je parametr, minimum, maximum a v případě FloatSlideru boolean, zda chcete FloatParameter normalizovat.



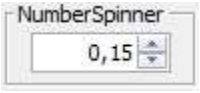
Obrázek 15 - FloatSlider

Obrázek 16 - IntSlider

NumberSpinner

Umožní naprosto přesný číselný vstup. Vhodné, pokud vám záleží na přesnosti, nebo vám nezáleží na minimální a maximální hodnotě.

NumberSpinner (viz Obrázek 17) je možné vytvořit pro IntParameter nebo FloatParameter, podle toho, jaký při vytváření zadáte. Dále můžete zadávat boolean, zda chcete parametr normalizovat, minimum, maximum a velikost kroku, kterou udělá jedno stisknutí šipky na ovládacím prvku. Pokud ho nenastavíte sami, bude minimum a maximum nastaveno na minimum a maximum daného datového typu.



Obrázek 17 - NumberSpinner

FloatArrayInput

Pro ovládání parametru typu FloatArrayParameter jde o seskupení několika NumberSpinnerů v rámci jednoho JPanelu. Lze použít i na vektory a rotace. Nastavení, lépe řečeno vytvoření, probíhá se stejnými parametry jako NumberSpinner. Obrázek 18 ukazuje FloatArrayInput pro FloatArrayParameter se čtyřmi parametry.

Obrázek 18 - FloatArrayInput

Pokud chcete, můžete pro zpřehlednění pojmenovat každý spinner metodou setLabels(String[] labels), kde vstup je pole textových řetězců obsahující jména. Například na Obrázku 19 jsou nastaveny popisky na x, y, z.

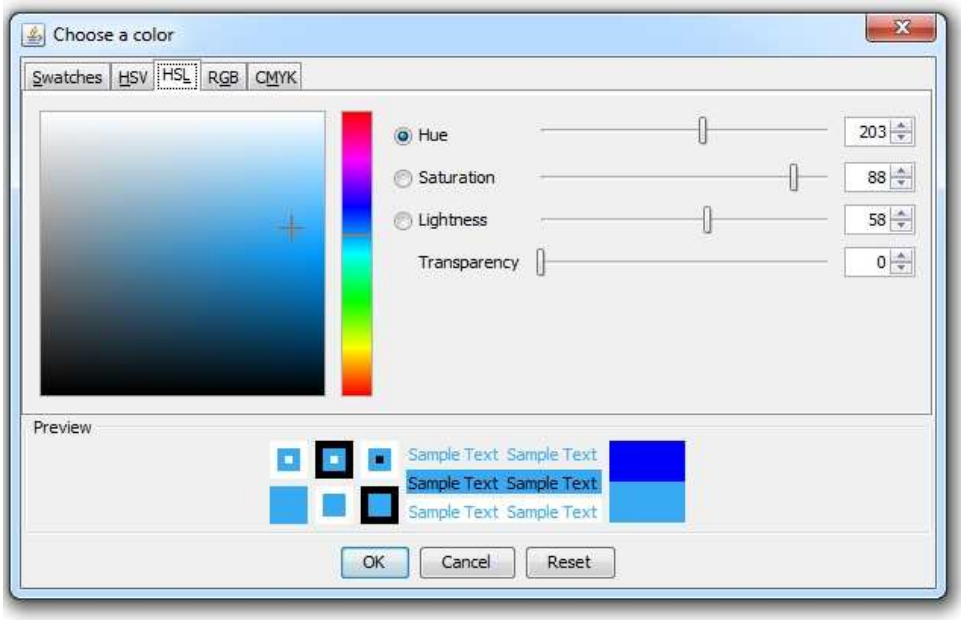
Obrázek 19 - FloatArrayInput s nastavenými popiskami

ColorPicker

Barva je reprezentována vektorem, ale není to zrovna nejsympatičtější způsob zadávání, proto byl vytvořen ovládací prvek ColorPicker. Jako vstup stačí FloatArray se čtyřmi prvky, nebo pokud chcete, je možné použít i Vec4fParameter. V GUI je ovládací prvek obyčejné tlačítko s aktuální barvou (viz Obrázek 20).

Obrázek 20 - Tlačítko ColorPickeru

Po stisknutí tlačítka vyskočí okno pro výběr barvy (viz Obrázek 21). Můžete vybrat jednu ze základní palety, nebo úplně vlastní za pomocí různých barevných modelů: HSV (Hue, Saturation, Value), HSL (Hue, Saturation, Lightness), RGB (Red, Green, Blue) a CMYK (Cyan, Magenta, Yellow a blacK)



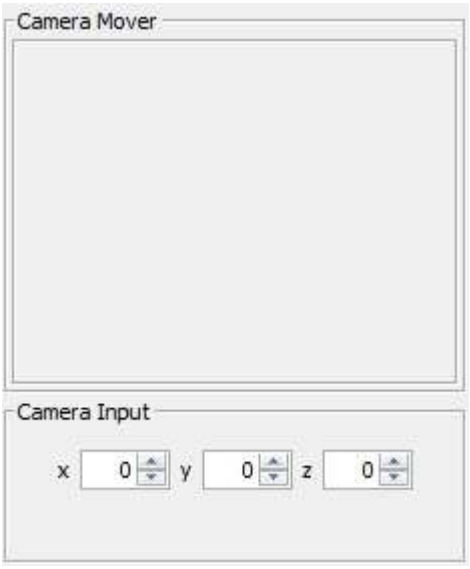
Obrázek 21 - Vyskakovací okno ColorPickeru

Ovládání kamery

Trackpad je implementován třídou CameraMover, trackball zase třídou CameraTrackball. Obě tyto třídy rozšiřují abstraktní třídu CameraControl, která rozšiřuje obyčejný javax.swing.JPanel obohacený o mouseListener a jeho metody.

CameraMover

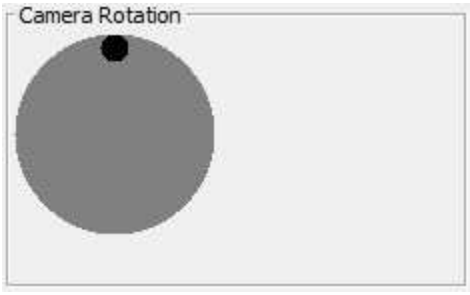
CameraMover (Obrázek 22) je naprosto primitivní trackpad. Při stisknutí pravého nebo levého tlačítka myši se posouváte po ose x a y vašeho pohledu, když stisknete obě, pohybujete se po ose z vašeho pohledu (do hloubky). Pro vytvoření musíte ovládacímu prvku předat TigerExaminerViever. Pokud byste chtěli zadávat pozici kamery ručně, zavoláním metody getCameraInput() na CameraMover vám vrátí, popřípadě pokud nebyla volána, ještě vytvoří JPanel obsahující ovládací prvek pro kameru jako Vec3fParametru pozice kamery (také na Obrázku 22).



Obrázek 22 - CameraMover a CameraInput

CameraTrackball

Ovládání natočení kamery. Pro pochopení následujících instrukcí se podívejte na Obrázek 23. Při stisknutí a táhnutí v tmavě šedé části prvku se otáčí kamera po svých osách x a y. Táhnutím a otáčením černého kolečka otáčíte kameru okolo její osy z. Nezapomeňte, že kamera v TigerExaminerVieweru se dívá na fokální bod z nějaké vzdálenosti, a v tomto případě otáčení kamery znamená spíše chození okolo fokálního bodu s kamerou.



Obrázek 23 - CameraRotation

TimeLine

TimeLine (viz Obrázek 24) se skládá z několika prvků. První řádek obsahuje tlačítka Play a Stop, které pravděpodobně nepotřebují představit. Do NumberSpinneru s popiskou FPS se zadává číslo, okolo kterého se bude FPSanimator držet při spuštění s vykreslovanými snímky za sekundu.

Samotná časová osa je pak tvořena řádkem vyšších políček, kde jedno políčko je jeden snímek. Při kliknutí pravým tlačítkem myši na prázdné políčko je možné na něm vytvořit klíčový snímek. Při kliknutí pravým tlačítkem myši na klíčový snímek můžeme vybrat, který v programu předem zadaný parametr chceme na daném snímku upravit, nebo klíčový snímek odstranit.

Řádek čtvercových políček nad časovou osou reprezentuje opakovač. Políčka, nad kterými je vybarven opakovač šedě, se budou přehrávat při spuštění animace. Kliknutím na políčko v opakovači levým tlačítkem myši nastaví počáteční snímek, pravým konečný. Poté proběhne kontrola a popřípadě se prohodí.

Pro zjištění na jakém snímku se nacházíte nechejte nad políčkem myš, po krátké chvíli by se u kurzoru mělo zobrazit informační okénko s číslem. Vytvoření jiného způsobu číslování se mi nepovedlo.

Obrázek 24 - TimeLine

Klíčové snímky

Jak již bylo zmíněno, klíčové snímky určují, jakým způsobem se parametr bude v čase chovat. V mém případě jsem naimplementoval lineární interpolaci. Klíčový snímek je označen na časové ose klíčkem (viditelné na Obrázku 24).

Pokud by chtěl tedy někdo vytvořit alternativní chování, jeho práce by nebyla nejtěžší. Například pro novou implementaci Keyframu pro FloatParameter stačí udělat nový třeba MyKeyframe.

```
MyKeyframe extends TigerParameterKeyframe<FloatParameter>
```

A naimplementovat vlastní průběh v metodě setParameterAtFrame(int frame).

Poté vytvořit ještě vlastní továrnu na keyframy, která bude také skoro identická s FloatArrayKeyframeFactory, pouze zamění řádek

```
return new FloatArrayKeyframe(parameter, frameNumber);
```

za řádek

```
return new MyArrayKeyframe(parameter, frameNumber);
```

Toto prozrazuje, jak funguje časová osa. Uchovává v sobě parametry, které se v čase mění, ke každému parametru se přiřadí jedna továrna. Je možné přiřadit jednu továrnu k více parametrům. Když je vše hotovo a uživatel spustí přehrávání, ve vykreslovacím cyklu se bude předávat číslo snímku, které časová osa vezme, a všechny parametry pomocí metody TigerParameterKeyframů setParameterAtFrame(int frame) nastaví na správnou hodnotu, se kterou je poté scéna vykreslena.

Programátor předá časové ose parametr pro změnu v čase metodou addTimeVaryingParameter(GlslProgramParameter parameter), který poté spáruje s továrnou metodou setFactoryForParameter(GlslProgramParameter parameter, KeyframeFactory factory).

Uživatel může v kódu přidat vlastní předem vytvořený keyframe na časovou osu metodou addParameterKeyframe(TigerParameterKeyframe keyframe). Pro lepší pochopení doporučuji názorný příklad kódu v kapitole Výsledky a ve zdrojových kódech projektu.

Výsledky

Ve výsledcích jsou představeny názorné ukázky ovládacích prvků a jejich použití. Příklady budou představeny spíše obecně, s několika obrázky, jak daný příklad vypadá. Více naleznete ve zdrojovém kódu a spustitelných souborech na CD. Zdrojový kód obsahuje i komentáře a slouží jako jakýsi tutoriál pro uživatele, aby byl schopen si naimplementovat vlastní uživatelské rozhraní. Každý příklad odkazuje svým jménem na stejnojmenný zdrojový kód v balíčku tiger.example.GUI.

V každém dalším příkladu navazuji svým způsobem na předchozí, budu tedy předpokládat, že v případě nejasností nějakého z kroků tvorby GUI by mělo být vysvětlení v jednom z předchozích příkladů.

GUIExample1

První příklad se zabývá vytvářením GUI. V kódu se nejprve inicializují potřebné věci pro vykreslování scény, jedná se o postup pracující s JOGL a knihovnou Tigr, do kterého jsem prakticky nezasahoval. Některé části, se kterými je potřeba pracovat při napojení ovládacích prvků nebo efektů, si představíme později.

Prvním krokem je vytvořit okno, předat mu předem načtenou scénu k vykreslení, a efekt, který vykreslení zpracovává.

```
Window w = new Window(scene, 512, 512, true);
w.setEffect(effect);
```

K ovládání kamery ve scéně vytvoříme TigerExaminerViewer, kterému předáme plátno a scénu, a nakonec ho nastavíme jako ovládání okna.

```
TigerExaminerViewer viewer = new TigerExaminerViewer(
                             java.awt.MouseInfo.getNumberOfButtons());
viewer.attach(w.canvas, scene);
w.setControler(viewer);
```

Poté jsou základní prvky GUI nastaveny a můžeme vytvářet záložky a ovládací prvky, které pak oknu předáme. Při vytváření záložky nezapomeňte uvést její jméno, jinak bude pojmenována Untitled.

```
DraggableTab firstTab = new DraggableTab("My First Tab");
DraggableTab secondTab = new DraggableTab("Second Vertical Tab");
DraggableTab thirdTab = new DraggableTab("Horizontal Tab");
```

Na záložku je poté možné přidat libovolnou komponentu metodou add, v tomto případě obyčejný JLabel s textem.

```
firstTab.add(new JLabel("Hello world"));
```

Záložky se poté předají oknu, podle toho, kde mají být při spuštění umístěné, metodou addVecticalTab, pokud na vertikální panel, a metodou addHorizontalTab na horizontální. Záložka přidaná na panel jako poslední bude po spuštění na vrchu tohoto panelu.

```
w.addVerticalTab(secondTab);
w.addHorizontalTab(thirdTab);
w.addVerticalTab(firstTab);
```

Nakonec spustíme příkazem `w.start();`

GUIExample2

Druhý příklad je zaměřený na vytváření FloatParametrů a NumberSpinner pro ně.

Nejprve tedy vytvoříme FloatParametry, vzhledem k tomu, že tyto parametry budeme chtít předat do shaderů, musíme je pojmenovat, jméno parametru nemusí být stejné jako jméno instance objektu v Javě.

```
FloatParameter darkness = new FloatParameter("darkness", 0.15f);
FloatParameter specularExp = new FloatParameter("specularExp",
                                                128.0f);
FloatParameter ambientLight = new FloatParameter("ambientLight",
                                                 0.2f);
```

V nepředstaveném kódu je část, která vytváří průchody vykreslováním, ke každému průchodu můžete přiřadit vlastní vertex a fragmet shadery pro způsob vykreslení. Tyto průchody jsou představovány v knihovně tiger Třídou Pass. V shaderech můžete využívat vlastních proměnných, které se jim předají právě přes jejich Pass. Předávání vypadá následovně:

```
pass1.glslVaryingParameters.add(darkness);
pass1.glslVaryingParameters.add(specularExp);
pass1.glslVaryingParameters.add(ambientLight);
```

Následuje vytvoření ovládacích prvků pro každý parametr. Vytváříme NumberSpinner, a jak již bylo zmíněno v implementaci, v tomto případě jsou vstupem parametr, false, když nechceme, aby byl normalizován, minimální hodnota, kterou lze ovládacím prvkem zadat, maximální hodnota a velikost jednoho kroku při používání šipek na Spinneru.

```
NumberSpinner darknessSpinner = new NumberSpinner(darkness, false,
                                    0f, 1f, 0.01f);
NumberSpinner specularExpSpinner = new NumberSpinner(specularExp,
                                    false, 0f, 180.0f, 1.0f);
NumberSpinner ambientLightSpinner = new NumberSpinner(ambientLight,
                                    false, 0f, 1f, 0.01f);
```

Nakonec opět vytvoříme záložku, na kterou ovládací prvky umístíme, a tu vložíme na vertikální panel.

```
DraggableTab firstTab = new DraggableTab("Light");
firstTab.add(darknessSpinner);
firstTab.add(specularExpSpinner);
firstTab.add(ambientLightSpinner);
w.addVerticalTab(firstTab);
```

GUIExample3

Třetí příklad ukazuje využití seskupení prvků pomoci ComponentGroup a další ovládací prvek floatů, FloatSlider. V GUI předchozího příkladu při spuštění nebylo příliš jasné, který ovládací prvek co ovládal. K zpřehlednění je dobré přidávat k prvků JLabely, nebo prvky třídit do vhodně pojmenovaných ComponentGroup. K vytvoření ComponentGroupy stačí libovolný název.

```
ComponentGroup darknessGroup = new ComponentGroup("Darkness");
ComponentGroup specularGroup = new ComponentGroup("Specular");
ComponentGroup ambientGroup = new ComponentGroup("Ambient Light");
```

Poté vytvoříte NumberSpinnery z předchozího příkladu a přidáte je do korespondujících ComponentGroup. V jedné ComponentGroupě může být libovolný počet prvků, doporučuji ale seskupovat pouze relevantní ovládací prvky.

```
darknessGroup.add(darknessSpinner);
specularGroup.add(specularExpSpinner);
ambientGroup.add(ambientLightSpinner);
```

Vytvoření FloatSliderů jako alternativní ovládání FloatParametrů. Při vytváření se zadává parametr, boolean, jestli má být normalizován, informace, zda je slider horizontální nebo vertikální, minimum a maximum na posuvníku.

```
FloatSlider darknessSlider = new FloatSlider(darkness, false,
JSlider.HORIZONTAL, 0f, 1f);
FloatSlider ambientSlider = new FloatSlider(ambientLight, false,
JSlider.HORIZONTAL, 0f, 1f);
```

Následuje přidání sliderů do ComponentGroupy již obsahující odpovídající spinner.

```
darknessGroup.add(darknessSlider);
ambientGroup.add(ambientSlider);
```

Přidání ComponentGroup na záložku probíhá jako přidání jakýchkoliv komponent.

```
lightTab.add(darknessGroup);
lightTab.add(ambientGroup);
lightTab.add(specularGroup);
```

GUIExample4

Příklad čtyři přináší ukázku vytváření závislostí mezi parametry. Nejprve musíme vytvořit skupinu parametrů, kterých se bude závislost týkat.

```
ParameterGroup contributionGroup = new ParameterGroup();
```

K této skupině poté musíme přidat vlastní implementaci ParameterGroupAction, která určuje chování parametrů ve skupině při změně kteréhokoliv z nich. Ukázka mé implementace zajistí, že součet všech parametrů ve skupině bude vždy 1.

```
contributionGroup.addGroupAction(new ParameterGroupAction() {
    @Override
    public void performGroupOperation(GlslProgramParameter source,
List<GlslProgramParameter> groupMembers) {
        float change = 1.0f;
        float sum = 0.0f;
        for (GlslProgramParameter param : groupMembers) {
            if (param != source) {
                sum += ((FloatParameter) param).getValue();
            }
        }
        change -= (sum + ((FloatParameter) source).getValue());
        if (sum == 0.0f) {
            for (GlslProgramParameter param : groupMembers) {
                if (param != source) {
                    ((FloatParameter)
param).setValue(((FloatParameter) param).getValue() + (change /
(groupMembers.size()-1)));
                }
            }
        } else {
            change /= sum;
            for (GlslProgramParameter param : groupMembers) {
                if (param != source) {
                    ((FloatParameter)
param).setValue(((FloatParameter) param).getValue() +
((FloatParameter) param).getValue() * change);
                }
            }
        }
    }
});
```

Nakonec do skupiny přidáme parametry, které mají být závislé.

```
contributionGroup.addParameter(ambientContribution);
contributionGroup.addParameter(diffuseContribution);
contributionGroup.addParameter(specularContribution);
```

GUIExample5

V pátém příkladu vytvářím FloatArrayParameter, jeho ovládání pomocí FloatArrayInput a možnost ovládání každé komponenty zvlášť. Vytvoření FloatArrayParametru, který je potřeba v tomto příkladu, bylo v příkladech již předtím, nebyla mu však věnována pozornost, protože doposud nebylo potřeba vysvětlení. V tomto příkladu využíváme poněkud neprakticky vstup pro pole k zadávání barvy, jako názorný příklad to ale stačí.

```
FloatArrayParameter background = new FloatArrayParameter
                    ("background", new float[]{0f, 0f, 0f, 1f});
```

Přiřazení parametru jako výchozí barvu pozadí RenderStatu.

```
rs.setClearColor(background);
```

Vytvoření FloatArrayInputu požaduje parametr, minimum a maximum a velikost kroku. Tyto vlastnosti jsou pro každý ovládací prvek společné.

```
ComponentGroup backgroundGroup = new ComponentGroup("Background
                                                    Color");
FloatArrayInput backgroundInput = new FloatArrayInput(background,
                                                    0f, 1f, 0.01f);
```

Přiřazení popisek ke vstupům je možné pomocí metody setLabels a zadání pole řetězců pro každou popisku.

```
backgroundInput.setLabels(new String[] {"R","G","B","A"});
backgroundGroup.add(backgroundInput);
```

Zároveň je možné z FloatArrayParametru vytáhnout jednotlivé složky a pro ně samostatně vytvořit libovolný ovládací prvek. V tomto případě jsem vytvořil pro názornou ukázku 4 slidery.

```
FloatSlider r = new FloatSlider(background.getValueAtPosition(0),
true, JSlider.HORIZONTAL, 0f, 1f);
FloatSlider g = new FloatSlider(background.getValueAtPosition(1),
true, JSlider.HORIZONTAL, 0f, 1f);
FloatSlider b = new FloatSlider(background.getValueAtPosition(2),
true, JSlider.HORIZONTAL, 0f, 1f);
FloatSlider a = new FloatSlider(background.getValueAtPosition(3),
true, JSlider.HORIZONTAL, 0f, 1f);

ComponentGroup colorSliders = new ComponentGroup("Color Sliders");
colorSliders.add((new ComponentGroup("R")).add(r));
colorSliders.add((new ComponentGroup("G")).add(g));
colorSliders.add((new ComponentGroup("B")).add(b));
colorSliders.add((new ComponentGroup("A")).add(a));

backgroundGroup.add(colorSliders);
```

GUIExample6

Vytvoření ColorPickeru jako alternativy k zadávání barvy je naprosto primitivní. Ve skutečnosti doporučuji pro zadávání barvy využívat spíše ColorPicker, v předchozím příkladu šlo pouze o názornou ukázku zadávání nějakého pole floatů, což pole hodnot uchovávající barvu splňovalo. ColorPickeru stačí předat FloatArrayParameter, používejte prosím pole o délce tří nebo čtyř prvků (čtyři jsou doporučené).

ColorPicker backgroundPicker = new ColorPicker(background);

GUIExample7

Příklad sedmý provádí jednoduché nastavení ovládání kamery, toto ovládání slouží jako rozšíření ovládání TigerExaminerVieweru. CameraMover i CameraTrackball potřebují jako vstup TigerExaminerViewer.

CameraMover je trackpad, pokud držíte pouze levé, nebo pravé tlačítko myši, pohybujete se po ose x a y aktuálního pohledu. Na rozdíl od ovládání TigerExaminerVieveru, kde při stisknutí obou tlačítek na myši měníte hodnotu dolly, která určuje, co bude v záběru, CameraMover vás bude posunovat po ose z aktuálního pohledu.

```
CameraMover cm = new CameraMover(viewer);
```

CameraRotation je trackball, který otáčí kameru v prostoru okolo fokálního bodu po jeho osách pohledu.

```
CameraRotation cr = new CameraRotation(viewer);
ComponentGroup CamGroup1 = new ComponentGroup("Camera Mover");
CamGroup1.add(cm);
```

CameraMover vám poskytuje možnost vytáhnout z něj i manuální vstup polohy kamery, a to metodou getCameraInput(), která vrátí už hotový JPanel s ovládáním, který pouze přidáte, kam chcete.

```
CamGroup1.add(cm.getCameraInput());
```

Zbytek kódu je pouze přidávání prvků na jednu záložku.

```
ComponentGroup CamGroup2 = new ComponentGroup("Camera Rotation");
CamGroup2.add(cr);
DraggableTab tabCam = new DraggableTab("Camera Control");
tabCam.add(CamGroup1);
tabCam.add(CamGroup2);
w.addVerticalTab(tabCam);
```

GUIExample8

Vytváření časové osy a přidávání parametrů do seznamu umožňující její změnu v čase si ukážeme v tomto osmém příkladu.

Časovou osu nejprve vytvoříme a předáme jí odkaz na naše okno.

```
TimeLine tl = new TimeLine();
tl.setWindow(w);
```

Poté přidáme jakoukoliv instanci nějaké implementace GlslProgramParametru metodou addTimeVaryingParameter. K tomuto parametru poté musíme přiřadit továrnu na klíčové snímky. V tomto případě moji FloatKeyframeFactory, vytvářející obyčejné FloatKeyframy, které v čase provádí lineární interpolaci mezi dvěma hodnotami.

```
tl.addTimeVaryingParameter(darkness);
tl.setFactoryForParameter(darkness, new FloatKeyframeFactory());
```

Tento krok musíme provést pro každý parametr přidávaný do časové osy. V tomto příkladu to bylo poněkud zbytečné, ale pokud by vám to vyhovovalo, jednu továrnu je možné používat pro více parametrů, které jsou stejného typu a od kterých požadujete stejné chování.

```
tl.addTimeVaryingParameter(background);
tl.setFactoryForParameter(background,
                          new FloatArrayKeyframeFactory());
tl.addTimeVaryingParameter(viewer.getCenter());
tl.setFactoryForParameter(viewer.getCenter(),
                          new FloatArrayKeyframeFactory());
tl.addTimeVaryingParameter(viewer.getOrientation());
tl.setFactoryForParameter(viewer.getOrientation(),
                          new RotfKeyframeFactory());
```

Nakonec přidáme časovou osu na některou záložku, v případě tohoto ovládacího prvku doporučuji záložku poté přidat na horizontální panel.

```
DraggableTab timelineTab = new DraggableTab("Time controler");
timelineTab.add(tl);
w.addHorizontalTab(timelineTab);
```

GUIExample9

V příkladu 9 si ukážeme, jak zacházet s časovou osou a jak na ni přidávat z kódu klíčové snímky. Opakovač na časové ose se nastavuje přes časovou jednotku okna, požadovaný počet snímků za sekundu se ale nastavuje přes časovou osu.

```
Window.timeUnit.setStartFrame(5);
Window.timeUnit.setEndFrame(50);
tl.setFPS(20);
```

Klíčové snímky je možné vytvářet i z kódu, jediné co musíte splnit, jsou požadavky, které vaše TigerParameterKeyframy požadují jako vstup, a pak jim nastavit požadovanou hodnotu. V tomto případě jsou vytvořeny dva klíčové snímky pro parametr darkness, jeden na snímku 10, kde hodnota bude nastavena na 1, druhý na snímku 40, kde už má být hodnota rovna 0.

```
FloatKeyframe darknessFrame1 = new FloatKeyframe(darkness, 10);
FloatParameter value = new FloatParameter("darknesValueAtFrame10",
1.0f);
darknessFrame1.setValueAtKeyframe(value);

FloatKeyframe darknessFrame2 = new FloatKeyframe(darkness, 40);
value = new FloatParameter("darknesValueAtFrame40", 0.0f);
darknessFrame2.setValueAtKeyframe(value);
```

Vytvořené časové snímky přidáme na osu metodou addParameterKeyframe, ta si je poté vnitřně zpracuje (vytvoří mezi nimi vzájemné odkazy a zanese si je na ovládací prvek časové osy).

```
tl.addParameterKeyframe(darknessFrame1);
tl.addParameterKeyframe(darknessFrame2);
```

Úplně stejně fungují i ostatní klíčové snímky, na ukázku zde máme vytvoření FloatArrayKeyframu pro změnu barvy pozadí, a RotfKeyframu pro změnu natočení kamery. Jejich kód by byl však příliš dlouhý a až na detaily stejný, proto pokud vás zajímá, podívejte se prosím na zdrojové kódy.

GUIExample10

Příklad pro znázornění vytvoření ovládacího prvku matice zároveň pracuje s průchody vykreslováním. V tomto příkladu se pokouším o vytvoření konvoluční matice.

Nejprve vytvoříme Parametr pro matici 3*3. Poté mu nastavíme jméno na convolutionMatrix, se kterým se pracuje v shaderu, a hodnotu prostředního prvku na 1, protože nová matice má vždy všechny prvky nastavené na 0.

```
Mat3fParameter matrix = new Mat3fParameter();
matrix.setName("convolutionMatrix");
matrix.set(1, 1, 1.0f);
```

Efekt konvoluční matice se aplikuje až při druhém průchodu, proto načteme nový fragment shader, který zpracovává tento efekt. Vzhledem k tomu, že první průchod vykresluje přes FrameBuffer do textury texture1, uvedeme jako vstup do druhého průchodu nově načtený shader a texturu, ve které je vykreslena scéna jako bitmapa. Průchodu potom ještě přidáme parametr matice.

```
fragmentStream =
ClassLoader.getSystemResourceAsStream("tiger/example/GUI/convolution
.frag");
Pass pass2 = new Saq(fragmentStream, texture1);
pass2.glslVaryingParameters.add(matrix);
pass2.renderState = rs;
```

Efekt pro konvulenci je poté v shaderu.

Závěr

Mým úkolem bylo vytvořit rozšíření knihovny Tigr, které umožní uživateli v čase měnit hodnoty parametrů.

Analyzoval jsem stav knihovny, přičemž jsem navrhl možné řešení pro parametry schopné změny v čase, a to implementací abstraktní třídy GlslProgramParameter. Implementace této třídy umožňuje předávání parametru do shaderu, pokud bude uživatel chtít, ale především možnost přidání listenerů, které při korektní implementaci slouží k upozornění na změny v parametru.

Dále jsem analyzoval možné vstupy pro různé typy parametrů a navrhl pro ně ovládací prvky.

Při implementaci jsem postupoval podle návrhů a pokusil se o možnou rozšiřitelnost kódu.

Na závěr jsem sestrojil několik ukázkových příkladů vytvoření uživatelského rozhraní, které zároveň pro čtenáře slouží jako návod postupu vlastní implementace.

Bibliografie

Olšák, P. (2007). *Úvod do algebry, zejména lineární.* Praha: Fakulta elektrotechnická ČVUT.

Wright, R. S., Lipchak, B., & Haemel, N. (2007). *OpenGL SuperBible Fourth Edition Comprehensive Tutorial and Reference.* Addison-Wesley.

Příloha

Obsah přiloženého DVD

Na kořenovém adresáři odevzdaného DVD naleznete

doc/	Složka obsahující dokumentaci knihovny Tigr
executables /	Spustitelné ukázkové příklady, více v README.TXT
lib/	Knihovny potřebné k projektu
src/	Zdrojové kódy
thesis_netusji1.pdf	Bakalářská práce
README.txt	Informace o obsahu DVD