Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Simulátor difraktivní optiky v paprskovém modelu světla

Plzeň, 2016 Matěj Berka

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Matěj BERKA**
Osobní číslo:	**A13B0276P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Název tématu:	**Simulátor difraktivní optiky v paprskovém modelu světla**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování:

1. Seznamte se s paprskovým modelem světla a jeho rozšířením pro difraktivní optiku.
2. Vyberte vhodný nástroj pro implementaci simulátoru; ten by měl být spustitelný prostřednictvím moderního webového prohlížeče.
3. Implementujte simulátor s vhodně vybranou funkcionalitou. Simulátor by měl umožňovat simulaci nejběžnějších optických soustav používaných v holografii.
4. Otestujte, jak se uživatelům simulátor používá a jak jim pomáhá v pochopení simulovaného jevu.

Rozsah grafických prací:	**dle potřeby**
Rozsah kvalifikační práce:	**doporuč. 30 s. původního textu**
Forma zpracování bakalářské práce:	**tištěná**

Seznam odborné literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce:	**Ing. Petr Lobaz** Katedra informatiky a výpočetní techniky
Datum zadání bakalářské práce:	**12. října 2015**
Termín odevzdání bakalářské práce:	**5. května 2016**

Doc. RNDr. Miroslav Lávička, Ph.D.
děkan

L.S.

Doc. Ing. Přemysl Brada, MSc. PhD.
vedoucí katedry

V Plzni dne 15. října 2015

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 27. dubna 2016

Matěj Berka

Abstract

In this graduation bachelor work was created new holographic simulator and it purpose is to simulate the most often used optical systems in holography. It is able to visualize behaviour of light in optical system in ray model of light in 2D. New version is based on knowledge from previous versions and removes some of their drawbacks. Newly brings support for mobile devices and is written in programming language JavaScript.

Abstrakt

V této bakalářské práci vznikl nový holografický simulátor vytvořený k simulaci nejběžnějších optických soustav používaných v holografii. Je schopný vyobrazit chování světla při průchodu optickou soustavou v paprskovém modelu světla a to ve 2D. Nová verze vychází ze zkušeností předchozích verzí a odstraňuje některé jejich nedostatky. Nově přináší podporu mobilních zařízení a oproti předchozím verzím je realizovaný v programovacím jazyce JavaScript.

Poděkování

Tímto bych chtěl velice poděkovat panu Ing. Petru Lobazovi, vedoucímu projektu. Za jeho pomoc a rady při tvorbě bakalářské práce a za jeho velice vstřícný přístup.

Obsah

1 Úvod

Cílem této práce je vytvořit výukový program, který uživatelům dovolí v interaktivní podobě vytvářet optické soustavy a přitom sledovat, jak se chová světlo, které touto soustavou prochází ve 2D. Uživatel bude mít po spuštění výukového programu (simulátoru) k dispozici pracovní plochu, na kterou bude moci rozmisťovat různé optické členy (komponenty) a zdroje světla a přitom sledovat, jak se světlo chová při střetu s jednotlivými členy. Zda se od členu odráží, nebo jím částečně prochází a pod jakým úhlem atd.

Simulátor bude zobrazovat ve 2D a bude možné jej používat jak při výuce, tak při práci. Při výuce bude například možné vysvětlit rozdíl mezi chováním ideální tenké čočky a reálné čočky, ukázat princip teleskopu nebo demonstrovat vznik a rekonstrukci hologramu. Při práci se bude moci využívat zejména jako pomůcka, která uživateli dovolí si nahrubo vyzkoušet, zda je daná soustava vůbec realizovatelná, zda neskrývá nějaké nedostatky, jaká je citlivost optické soustavy na nedokonalost rozestavení optických členů a další.

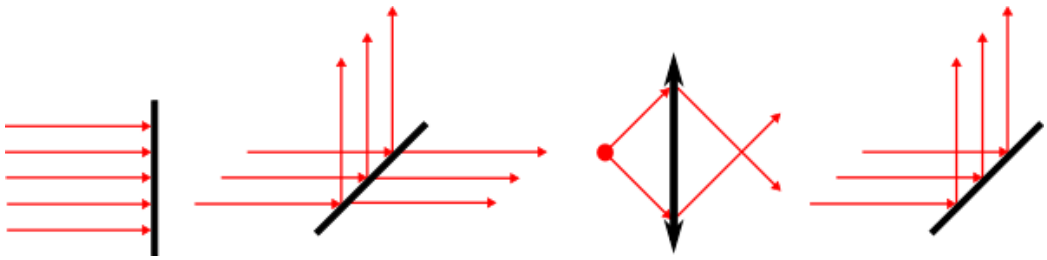
Tato práce není první, která se vývojem takového simulátoru zabývala. V minulosti již několik simulátorů vzniklo v podobě bakalářských a diplomových prací, které se stále udržují. Všechny předchozí verze byly napsány jako aplikace v jazyce Java, které se mohly spouštět jak na desktopu, tak jako applety v okně webového prohlížeče. Tato práce je opět zaměřena na webovou verzi. S vedoucím práce jsme se dohodli, že by stálo za to vyzkoušet i jinou cestu, než jsou Java applety. Navrhl jsem využít HTML5 a JavaScript, který je nejrozšířenějším jazykem schopným běžet ve webovém prohlížeči. Možnosti simulátoru budou vycházet z předchozích verzí, aby se zamezilo opakování chyb a kvalita simulátoru spíše neupadala. Informace jsem tedy čerpal především z předcházejících verzí simulátoru a částečně také z podobných simulačních programů, které popsal ve své diplomové práci Kamil Rendl [4].

Výše popsané možnosti použití jsou také důvodem, proč se začalo s vývojem vlastního simulátoru a nepoužívá se již nějaké existující řešení. Požadavky jsou velmi specifické a i když podobné programy existují, většina jsou komerční produkty, které bývají drahé a navíc jsou pro studenty složité. Také většina z nich neposkytuje výhodu v podobě webové aplikace, kdy pouze stačí navštívit adresu, na které se simulátor nachází a není potřeba program instalovat.

2 Chování světla

2.1 Paprskový model světla a optické členy

Simulované světlo bude kvůli názornosti reprezentované v paprskovém modelu. Tedy zdroje světla budou opouštět svazky paprsků (polopřímek), které mohou narazit do dalšího optického členu. Přitom každý optický člen na příchozí paprsek reaguje specifickým způsobem. Může jej pohltit, odrazit, částečně propustit atd. Záleží na tom, s jakým optickým členem se paprsek střetne. Pokud optický člen paprsek odráží či propouští, tak vytváří modifikovaný paprsek a posílá jej dále. Ten se zase může střetnout s jiným členem. Na následujícím obrázku je příklad, jak se takový paprsek (polopřímka) může při střetu s optickým členem zachovat.



Obrázek 2.1: Příklady střetů světla s optickými členy. Na prvním obrázku je možné sledovat, že světelné paprsky jsou po střetu s optickým členem zcela pohlceny. Optickým členem je v tomto případě stínítko. Na druhém obrázku je vidět, že se paprsek rozdělí na dva, kde část se odrazí a část projde skrz. Zde se jedná o polopropustné zrcátko (dělič paprsků). Na třetím obrázku je vidět, že paprsek projde skrz, ale už nemá ten samý směr, jako měl při vstupu. Dojde tedy k lomu světla a v tomto případě k lomu světla na ideálně tenké čočce. Na posledním obrázku je vidět, že se paprsek od členu jednoduše odráží. V tomto případě se jedná o klasické zrcátko.

Jak je z obrázku 2.1 vidět, chování světla je závislé na tom, s jakou komponentou se střetne. Těchto základních komponent může mít simulátor velké množství. V první verzi jsme se rozhodli pro stínítko, zrcátko, polopropustné zrcátko, ideálně tenkou čočku, zdroj světla a materiál zaznamenávající interferenci světla („holografickou desku“). Většinu optických členů není nutné představovat. Komponenta, která by však mohla zarazit je „holografická deska“ a tu si představíme v následující sekci Hologram.

Každá z výše zmíněných komponent bude mít také sadu parametrů, které jim bude možné nastavovat a upřesňovat tak jejich chování. Například u čočky bude možné určit její ohniskovou vzdálenost. Tedy to, jak se bude světlo chovat při střetu s optickým členem. Nebude určovat pouze typ členu, ale také jeho konkrétní nastavení.

2.2 Hologram

Komponenta nazývaná „holografická deska“, nebo také zjednodušeně záznamový materiál, je odlišná od předchozích uvedených komponent (optických členů). Umožňuje totiž vytvořit záznam interference světla, které jí osvětluje. Při příslušném nastavení jejího osvětlení je simulován záznam holografického optického členu, resp. difrakční mřížky, kterou je možné po „vyvolání“ osvětlovat a pozorovat ohyb světla. V případě simulátoru se jedná o jednotlivé paprsky (polopřímky), které na desku dopadají. Tato komponenta nám dovolí vytvářet něco, čemu se říká hologram.

Pod pojmem hologram si asi každý představí různé prostorové obrázky či animace a nebude přitom daleko od pravdy. Avšak iluzi prostorového obrázku či animace je možné realizovat mnoha způsoby a hologramy jsou pouze jedním z nich. Pod pojmem hologram je tedy nutné chápat právě záznam interference světla na záznamový materiál a jeho následnou rekonstrukci která opět zobrazí zaznamenané světlo.

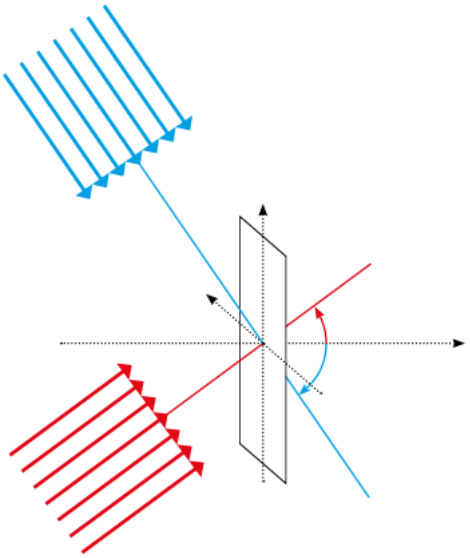
Kdybychom tedy chtěli zaznamenat holograficky nějaký předmět, tak musíme vytvořit záznam světla, které se od předmětu odráží. Při rekonstrukci hologramu bychom viděli to samé světlo, které se odráželo od předmětu a zdálo by se nám jako by byl předmět opět přítomen. Tedy viděli bychom obraz původního předmětu.

Tato komponenta má oproti ostatním zmiňovaným komponentám dvě základní odlišnosti. Za prvé mění své chování podle dopadajícího světla a za druhé používá k popisu chování vlnový model světla, respektive paprskový model světla rozšířený o jisté vlnové aspekty. Jelikož se tato část optiky v počítačové grafice příliš nepoužívá, seznámíme se s ní v další kapitole podrobněji.

2.3 Záznam a rekonstrukce světla

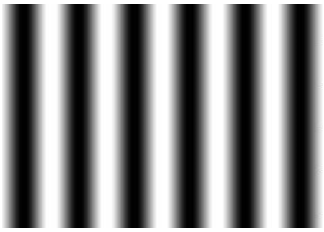
K tomu, aby „holografická deska“ (záznamový materiál) mohla zaznamenat a následně rekonstruovat světelné paprsky, je potřeba, aby na záznamový materiál dopadaly alespoň dva svazky paprsků. Jeden z nich

představuje takzvané referenční světlo, které je využito i při rekonstrukci záznamu. Nepředstavuje tedy světlo, které chceme zaznamenat. Druhý svazek je takzvané objektové světlo reprezentující světlo (objekt), které chceme zaznamenat.



Obrázek 2.2: Dva svazky paprsků přivedené na záznamový materiál [6]

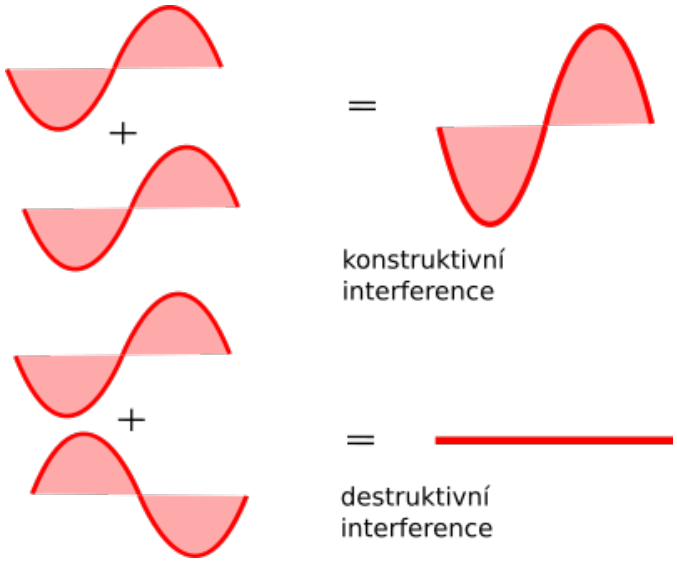
Tyto dva svazky budou po dopadu na desku navzájem interferovat a vytvoří interferenční vzor proužků, ve kterém se budou střídat maxima a minima světelné intenzity. Právě interferenční vzor se zaznamenává na „holografickou desku“ (záznamový materiál). Jak takový interferenční vzor může vypadat, je znázorněno na následujícím obrázku 2.3.



Obrázek 2.3: Interference světla

Z příkladu je vidět, že ve vzoru interference se postupně střídají světlé a tmavé proužky. Tmavá místa ukazují, že na některých místech se paprsky

navzájem vyruší a vzniká tam tma. Naopak na světlých místech se paprsky navzájem sčítají a vznikají tam světelná maxima. Tento efekt vychází z vlnových vlastností světla a lze ho ilustrativně popsat jako sčítání dvou periodických funkcí. Obrázek 2.4.



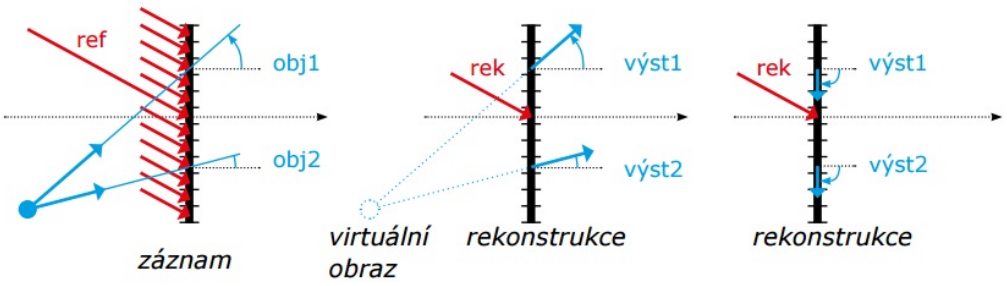
Obrázek 2.4: Vznik interference

Na obrázku 2.4 je vidět jak se osově převrácené vlny navzájem vynulují. V těchto místech nastane tma. V jiných místech se naopak sečtou a vzniknou tam světelná maxima.

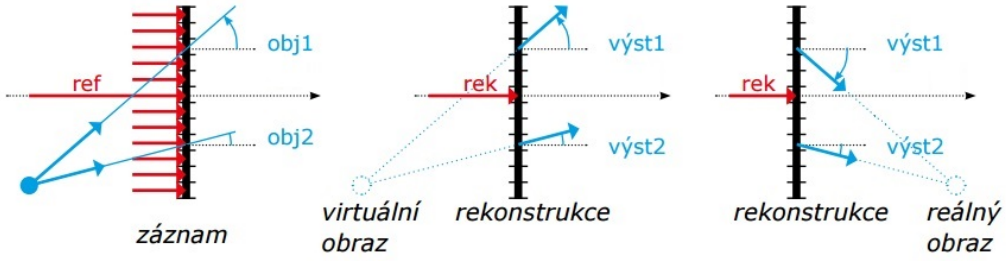
Tento interferenční vzor na „holografickou desku“ (záznamový materiál) zaznamenáme. Když po provedení záznamu odebereme zdroj objektových paprsků a zaznamenaný vzor nasvítíme, začne propuštěné světlo vykazovat ohyb (difrakci). Zaznamenaný vzor proto můžeme chápat jako difrakční mřížku. Je-li difrakční mřížka vyrobena uvedeným optickým způsobem a nasvítíme-li ji kopií referenčního světla, vytvoří některé z difraktovaných paprsků dokonalou kopii objektového světla. Speciálně vyrobené difrakční mřížce, kterou jsme právě popsali, říkáme hologram. Pokud se budeme dívat na mřížku tak, aby nám do očí dopadaly difraktované paprsky, tak se nám bude zdát, jako kdyby paprsky vycházely právě z původního objektového svazku paprsků, který osvětloval náš záznamový materiál. Tato situace je znázorněna na obrázku 2.5.

Pokud by svazek referenčních paprsků při záznamu dopadal kolmo na záznamový materiál, tak bychom při opětovném nasvícení pozorovali, že se nám vytvoří dvě kopie objektových paprsků. Jedna kopie vytváří virtuální obraz, druhá vytváří reálný obraz. Tato situace je znázorněna na obrázku 2.6. Jen pro upřesnění: na obrázku 2.5 se rovněž vytvářejí dvě kopie, ale je vidět, že paprsky druhé kopie jsou natolik stočené, že je nebude možné pozorovat.

Pokud se paprsky sbíhají do jediného bodu, říkáme mu reálný obraz. Příklad reálného obrazu je zakreslen do obrázku 2.6. Ten by se pozorovateli, stojícímu na stejné straně jako je reálný obraz, jevil jako vystupující z roviny. V tomto případě bychom také pozorovali, že reálný obraz má převrácené hloubky. To znamená, že pokud by objektové paprsky reprezentovaly dva body A a B, kde A je dále od záznamového materiálu než B, pak při pohledu na reálný obraz by se zdálo, že je tomu přesně naopak. Kromě reálného obrazu je vytvářen také virtuální, který vzniká na pomyslném průsečíku rozbíhavých paprsků.



Obrázek 2.5: Příklad záznamu hologram (Off-axis hologram) [6]



Obrázek 2.6: Příklad záznamu hologram 2 [6]

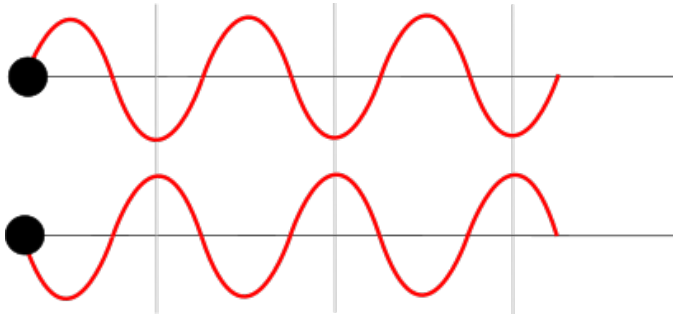
Poznámka: na obrázku 2.5 i 2.6 červené paprsky představují referenční paprsky (při rekonstrukci též nazývané jako rekonstrukční paprsky). Tyto paprsky jsou pouze pomocné a nepředstavují světlo které chceme zaznamenat. Modré pak objektové paprsky. Ty reprezentují světlo které chceme zaznamenat. Kvůli zjednodušení obrázků, na kterých jsou vyobrazeny rekonstrukce, je na nich uveden pouze jeden reprezentant červených paprsků. Jedná se však o stejné světlo, jako v případě referenčních paprsků. Také je vidět, že rekonstrukce světla je vyobrazena dvakrát, aby bylo zdůrazněno, že paprsky se při difrakci světla šíří do obou stran. Malé čárky na záznamovém materiálu představují rozdělení na malé úseky, kde v každém úseku vzniká trochu jiný interferenční vzor. Ten určují úhly dopadajících paprsků. Tečkované modré čáry ukazují, že světelné paprsky, které vznikají při rekonstrukci, opouštějí záznamový materiál ve stejném směru, ve kterém na něj dopadaly paprsky původní.

2.4 Simulace a realita

V předchozí části jsme si popsali princip záznamu a rekonstrukce hologramu. Musíme si však uvědomit, že v reálných podmínkách je nutné splnit mnoho dalších předpokladů, aby se nám povedlo záznam vytvořit a rekonstruovat. Simulátor bude v tomto směru zjednodušen a od těchto problémů oproštěn. Částečně také proto, že některé předpoklady by v simulaci nebylo možné splnit. Pro představu bude dobré si tyto problémy alespoň krátce popsat.

2.4.1 Délka koherence

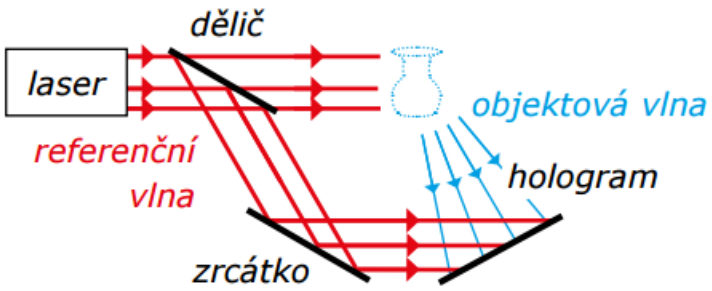
Zmínil jsem, že je potřeba alespoň dvou svazků paprsků, aby se vytvořil interferenční vzor, který chceme zaznamenat. Je však nutné aby tyto svazky byly také navzájem koherentní a to po dostatečně dlouhou dobu. Když si světlo opět představíme jako vlnění, tak je koherentní, pokud má stejný směr kmitání, stejnou frekvenci a konstantní fázový rozdíl. Délka koherence je právě jedním z důvodů proč běžně nepozorujeme interferenci u denního světla. V holografii se tedy převážně používají lasery, které mívají délku koherence řádově desítky centimetrů až desítky metrů, zatímco u denního světla jsou to řádově mikrometry.



Obrázek 2.7: Koherentní vlny (jejich fázový rozdíl je konstantní)

Pokud chceme vytvořit hologram nějakého předmětu, je postup následující. Svazek světla z laserového zdroje se přivede na dělič svazku, který jej rozdělí na dva. Svazkem A nasvítíme předmět, který chceme zaznamenat. Z každého bodu předmětu se začne šířit kulová vlnoplocha odraženého světla a takto odražené světlo necháme dopadat na záznamový materiál. Svazek B také přivedeme na záznamový materiál. Například pomocí zrcátka. Pokud je rozdíl vzdáleností obou svazků paprsků od světelného zdroje k záznamovému

materiálu menší než je délka koherence, tak se na záznamovém materiálu vytvoří interferenční vzor, který bude věrohodně reprezentovat původní zdroje světla (svazek referenčních paprsků a světlo odražené od našeho předmětu). Popsaný postup je vyobrazen na následujícím obrázku 2.8.



Obrázek 2.8: Záznam objektu [6]

Takto vytvořený hologram se nazývá transmisní hologram. U něj platí, že jak objektové paprsky, tak referenční paprsky svítí na záznamovou desku z jedné strany. Existují i další typy hologramů, jako je například reflexní hologram, u kterého objektové paprsky svítí na záznamový materiál z druhé strany, ale těmi se dále zabývat nebudeme. V simulátoru si bude možné vyzkoušet pouze transmisní hologramy. Délka koherence použitého světla je tedy důležitým faktorem při tvorbě hologramů a v simulátoru bude nutné dodržet délku koherence. Pokud délku koherence nedodržíme, nebude možné vytvořit záznam interferenčního vzoru.

2.4.2 Kvalita záznamu

Během pořizování záznamu je v simulaci uvažován ideální stav. To znamená, že paprsky se šíří nerušeně jako ve vakuu, rovinné zrcátko je skutečně rovné a bez prachových zrnek, stínítko pohltí veškeré dopadající světlo atd. Je však nutné si uvědomit, že v reálných podmínkách se střetáváme s řadou potíží na které je nutné brát zřetel. Například během záznamu by měla být naprostá tma, aby na záznamový materiál nedopadalo ještě i jiné světlo a nenarušovalo interferenční vzor, který se snažíme zaznamenat. Přesto se však ideálnímu stavu pouze blížíme. Světlo se rozptyluje ve vzduchu, i od té nejčernější plochy se část světla odrazí, žádné zrcátko není dokonale rovinné a dokonale čisté apod.

Kromě naprosté tmy se také snažíme zamezit jakýmkoliv vibracím, protože obzvláště experimenty s interferencí světla jsou velmi citlivé na

změny rozestavení komponent. Tyto změny mohou být i těžko ovlivnitelné. Například nasvícené zrcátko se může vlivem světla zahřívat a tím měnit svůj tvar, nebo zdroj světla může ohřívá vzduch a měnit tak jeho index lomu. V simulaci je naproti tomu opět uvažována ideální situace.

Chceme tedy vytvořit stabilní a ničím nerušené prostředí. Kvalitu záznamu neurčuje pouze interferenční vzor, ale také kvalita materiálu na který se snažíme interferenční vzor zaznamenat. Kvalitní záznamový materiál je například schopný zaznamenat 1000 linek interferenčního vzoru na mm. Tomu také odpovídá fakt, že reálné vzory interference bývají opravdu mikroskopické a spíš vypadají jako nečitelný šum, než aby připomínaly obrázek 2.3. Jako záznamový materiál je někdy používán fotografický film, který je však schopný zaznamenat řádově 100 linek na mm. Mnohem častěji se používá materiál vyrobený ze speciální emulze. Při výběru záznamového materiálu je také důležité zvažovat životnost hologramu, protože ta závisí na vybraném materiálu.

3 Výběr prostředků

V úvodu jsem zmínil, že při tvorbě se bude používat programovací jazyk JavaScript. Hlavním důvodem je, že v době psaní této práce je JavaScript nejrozšířenějším jazykem schopným běžet ve webovém prohlížeči a také široce podporovaným na desktopových a mobilních verzích webových prohlížečů. Z pohledu uživatele není tedy potřeba instalovat žádné dodatečné moduly a aplikaci takto vytvořenou je možné začít ihned používat. Bohužel tato podpora není jednotná a co prohlížeč, to trochu jiný JavaScript. Abych to upřesnil, existují přesné specifikace, které definují, co by měla jaká verze JavaScriptu obsahovat, ale už záleží na výrobcích prohlížečů, jestli se jimi budou řídit. To se naštěstí děje, ale běžně si každý výrobce přidá něco navíc.

Příkladem mohou být webové stránky [8] společnosti Mozilla, které ukazují jaké rozšíření JavaScriptu je dostupné ve webovém prohlížeči Firefox. Častým problémem bývá využívání pokročilých funkcí při asynchronní výměně dat s webovým serverem (AJAX). Dalším problémem je implementace nových specifikací. Některé webové prohlížeče neposkytují automatické aktualizace a uživatelé zůstávají na starších verzích. To je například velký problém u prohlížeče Internet Explorer, který používá velké procento uživatelů. V praxi to znamená, že je nutné hlídat i verze webových prohlížečů, když chceme používat nové konstrukce jazyka. Jednotlivé specifikace vydává konsorcium World Wide Web Consortium (W3C). Při navrhování simulátoru jsme se s vedoucím projektu domluvili, že by měl simulátor podporovat široce používané desktopové prohlížeče. Konkrétně se jedná o prohlížeče: Chrome, Firefox, Internet Explorer od verze 10 (nově Edge), Operu a Safari. Také by měl alespoň částečně podporovat mobilní zařízení s operačními systémy Android a iOS.

Kromě problému rozdílných implementací JavaScript přináší ještě jeden problém a to, že není zcela připravený na vývoj aplikací, které již vyžadují nějakou složitější logiku a specifickou architekturu (například architekturu Model-view-controller). V ten okamžik začneme narážet na různé problémy. Například jak strukturovat kód? JavaScript nemá tradiční konstrukce pro definici tříd a balíčků, které je možné vidět například v programovacím jazyce Java. Jak si hlídat datové typy? JavaScript je dynamicky typovaný jazyk. Nebo jak vytvářet privátní metody? Z počátku bylo možné využívat JavaScript pouze na přidávání interaktivních prvků textovému dokumentu, nebylo tudíž potřeba žádných složitějších konstrukcí. Situace se naštěstí lepší a některé tradiční konstrukce, které je možné vidět ve vyspělých jazycích, jsou nyní možné i v JavaScriptu. Přesto většinou konstrukce zatím není možné reálně používat, protože je podporují jen nejnovější verze webových

prohlížečů a u mobilních prohlížečů je podpora téměř nulová. Je tedy nutné si na tyto konstrukce ještě nějaký čas počkat.

Vývoj v čistém JavaScriptu tedy přináší jisté problémy. Aby byl programátor alespoň částečně osvobozen od těchto problémů je zapotřebí přidat nějakou abstraktní vrstvu, která by práci usnadnila. Stanovil jsem si 3 základní požadavky, které jsem chtěl aby hledané řešení splňovalo. Hledal jsem řešení se statickou typovou kontrolou, alespoň částečným zakrytím výše zmíněných implementačních rozdílů a návrhem strukturování kódu. V dalším textu se zmiňuji o existujících cestách, jenž alespoň částečně tento problém řeší.

3.1 Výběr vhodných nástrojů

Jedním z přístupů je využívání jiného programovacího jazyka. V tom danou aplikaci napsat a kód přeložit do JavaScriptu. Mezi reprezentanty této třídy patří například TypeScript, CoffeeScript a Google Web Toolkit (GWT). Mnoho dalších je možné nalézt na webové stránce [7].

TypeScript a CoffeeScript jsou si velmi podobné. Oba jazyky vznikly proto, aby usnadnily vývoj JavaScriptových aplikací. Snaží se přinést nové konstrukce, které v JavaScriptu není možné vytvářet, jako je například tradiční definice tříd klíčovým slovem class. TypeScript je v tomto směru o něco dále a navíc například přináší statickou typovou kontrolu. TypeScript vytvořila firma Microsoft a autorem CoffeeScriptu je Jeremy Ashkenas. GWT je trochu opakem předchozích dvou. Nesnaží se JavaScript rozšířit, ale zcela se mu vyhnout. GWT je sada nástrojů, které vytvořila firma Google, a jejich účelem je dovolit vývojářům psát webové aplikace v programovacím jazyce Java a následně aplikace přeložit do JavaScriptu a HTML.

Pokud bych měl shrnout, proč jsou tyto jazyky vytvářeny a používány, tak jsou to právě různé nové konstrukce těchto jazyků, které v JavaScriptu není možné vytvářet a které mají zpříjemnit (usnadnit) vývoj. Dalším důvodem je také řešení problému, kdy mám existující aplikaci a tu bych rád co nejrychleji předělal na JavaScriptovou aplikaci. Na webové stránce [7] je možné se dočíst, že již existuje řada jazyků, které lze pomocí různých nástrojů do JavaScriptu převádět.

Tento přístup se mi však nejevil vhodný. Znamenalo by to psát v jiném programovacím jazyce než byl původně zamýšlený JavaScript. To by mohla být v případě GWT výhoda, ale stále po překladu vzniká JavaScriptový kód. Dále v textu se zmiňuji, že existují i čistě JavaScriptová řešení, která rovněž poskytují hledanou abstrakci. Tudíž je trochu nadbytečné používat jiný jazyk. Navíc v době psaní této práce nic nenasvědčovalo tomu, že by

měl JavaScript v nejbližší budoucnosti z webových prohlížečů vymizet a být nahrazen některým ze zmíněných jazyků. Od tohoto přístupu jsem tedy odstoupil a dále jsem hledal pouze řešení, která by mi umožňovala psát přímo v JavaScriptu.

Jako další možnost jsem zkoumal využití nějakého existujícího frameworku (či knihovny) napsaného v čistém JavaScriptu, který by pouze poskytoval potřebnou abstrakci. JavaScriptových frameworků existuje opravdu ohromné množství a většina se liší jen v drobných detailech. Z toho důvodu v následujícím textu popíši pouze ty, u kterých jsem zvažoval jejich použití.

jQuery
: jQuery je to dobře známá a široce používaná knihovna, kterou vytvořil John Resig. Je to jedna z prvních knihoven, jenž se snažily vytvořit abstraktní vrstvu, která by překlenula rozdílné implementace JavaScriptu ve webových prohlížečích. Má dobře definované jednotlivé metody (API) a přidává mnoho užitečných funkcí, které například usnadňují práci s objektovou reprezentací webové stránky (DOM), vytváření různých animací a přidávání nových událostí. Řeší tedy problém rozdílností webových prohlížečů, ale už dále neřeší jak strukturovat kód a ani nepřináší statickou typovou kontrolu.

AngularJS
: AngularJS je nový framework, který vytvořili zaměstnanci firmy Google. Tento framework podobně jako jQuery přináší abstraktní vrstvu, která zakrývá rozdíly jednotlivých prohlížečů. Není však tak obsáhlá jako jQuery, která má mnoho funkcí navíc. Také obsahuje návrh, jak strukturovat kód, ale bohužel opět neobsahuje statickou typovou kontrolu.

React + Flow
: React společně s Flow jsou projekty, které zveřejnila firma Facebook. React je JavaScriptová knihovna, která opět stejně jako AngularJS řeší rozdíly webových prohlížečů a navrhuje jak strukturovat kód, ale také společně s Flow přidává statickou typovou kontrolu. Řeší tedy hlavní požadavky.

Closure Tools
: Closure Tools je JavaScriptová knihovna a sada nástrojů, které vytvořila firma Google již před několika lety a s úspěchem je používá dodnes v mnoha svých produktech. Mezi tyto produkty patří například Gmail či Google mapy. Je tedy velmi dobře odzkoušena na již

existujících aplikacích. Kromě toho, že řeší tři zásadní problémy o kterých jsem se zmiňoval v předcházejícím textu, přináší ještě mnohem více, například vlastní šablonovací systém, rozšíření definice CSS, statický analyzátor kódu kontrolující dodržování konvencí a také optimalizační nástroj kontrolující syntaxi kódu, odstraňuje nedosažitelný kód programu, odebírá komentáře a minimalizuje kód do kompaktní nečitelné podoby. Další výhodou je, že není nutné využívat vše co Closure Tools nabízí, ale pouze některou část.

Pro porovnání výše zmíněných frameworků, knihoven přidávám přehledovou tabulku.

	jQuery	AngularJS	React + Flow	Closure Tools
Statická typová kontrola	Ne	Ne	Ano	Ano
Návrh, jak strukturovat kód	Ne	Ano	Ano	Ano
Alespoň částečné zakrytí rozdílů prohlížečů	Ano	Ano	Ano	Ano

Tabulka 3.1: Porovnání rozdílů

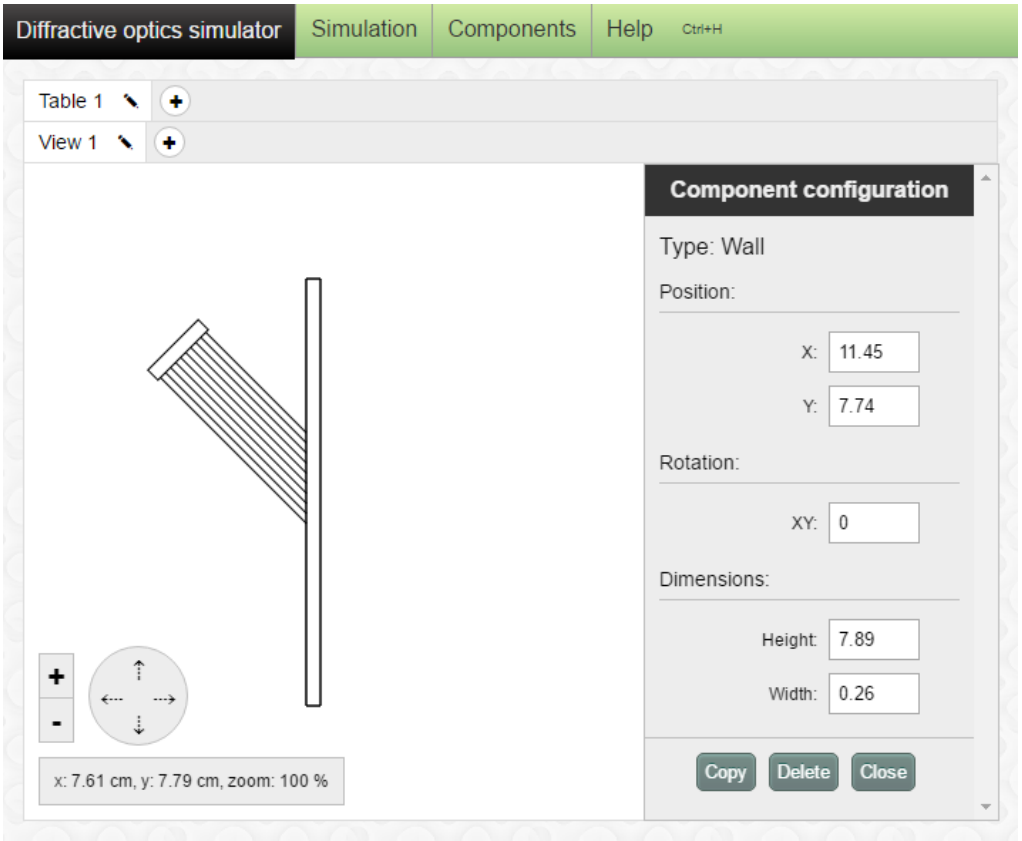
Po porovnání jsem se rozhodl pro Closure Tools, které obsahují vše, co jsem hledal a také přinášejí mnoho dalších užitečných nástrojů, které jsou dobře odzkoušené na velkých projektech. Co mi nevyhovovalo na AngularJS a Reactu je jejich zaměřenost na práci s objektovou reprezentací webové stránky (DOM) a jeho častými změnami. To je v tomto projektu až druhotná věc a většinu programu bude tvořit kód pracující s kreslícím plátnem. Stáří těchto dvou projektů naznačovalo, že v budoucnu pravděpodobně dojde ještě k velkým změnám a tedy k problému s přechodem na novější verzi. React je v době psaní této práce teprve dva roky starý projekt, stále ve verzi 0.14.2 a AngularJS 6 let ve verzi 1.4.7 s chystanou verzí 2.0. Aplikaci jsem se rozhodl vyvíjet přímo v JavaScriptu, ale přitom využít nástrojů a knihovny, které poskytují Closure Tools.

3.2 Vykreslování simulace

Velmi důležitou součástí simulátoru je také plátno, na které se bude simulace vykreslovat. Na to má HTML5 speciální element označený jako canvas, který umožňuje vykreslovat bitmapovou grafiku. K tomu ještě existuje rozšíření v podobě webového API nazvaného WebGL, které přináší podporu interaktivní 3D grafiky. Vzhledem k tomu, že simulátor bude pracovat s 2D grafikou, tak se od tohoto API pro tuto verzi upustilo. Alternativou by bylo ještě využít SVG, které dovoluje vykreslovat vektorovou grafiku. Tato volba by však nebyla vhodná, protože se v simulaci může vyskytovat velké množství paprsků, tedy mnoho úseček a SVG si vytváří pro všechny své elementy objektovou reprezentaci a to se u komplexních obrázků může znatelně projevit na výkonu. U canvasu klesá výkonnost pouze s rostoucími rozměry plátna.

4 Implementace simulátoru

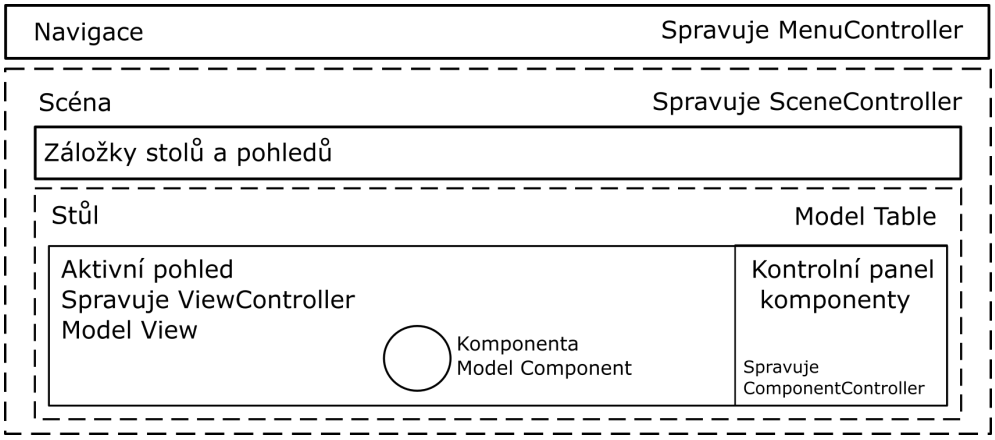
Nyní si detailně popíšeme již implementovaný simulátor. Nejprve si představíme základní strukturu simulátoru. Popíšeme si části, ze kterých je složen a vztahy mezi nimi. Pak bude následovat část popisující implementaci jednotlivých tříd, metod a jejich účelu. Nakonec bude popsáno, jak je možné simulátor dále rozšiřovat a upravovat. Simulátor se stává aktivní ihned po dokončení načítání stránky. Neobsahuje žádnou úvodní stránku ani nějaké další stránky, na které by se dalo z hlavní stránky přecházet. Obsahuje pouze jednu stránku, jejíž obsah je podle voleb uživatele pozměňován pomocí JavaScriptu. Jedná se tedy o tzv. Single-page aplikaci (SPA). Náhled do uživatelského rozhraní aplikace je vidět na následujícím obrázku 4.1.



Obrázek 4.1: Ukázka uživatelského rozhraní

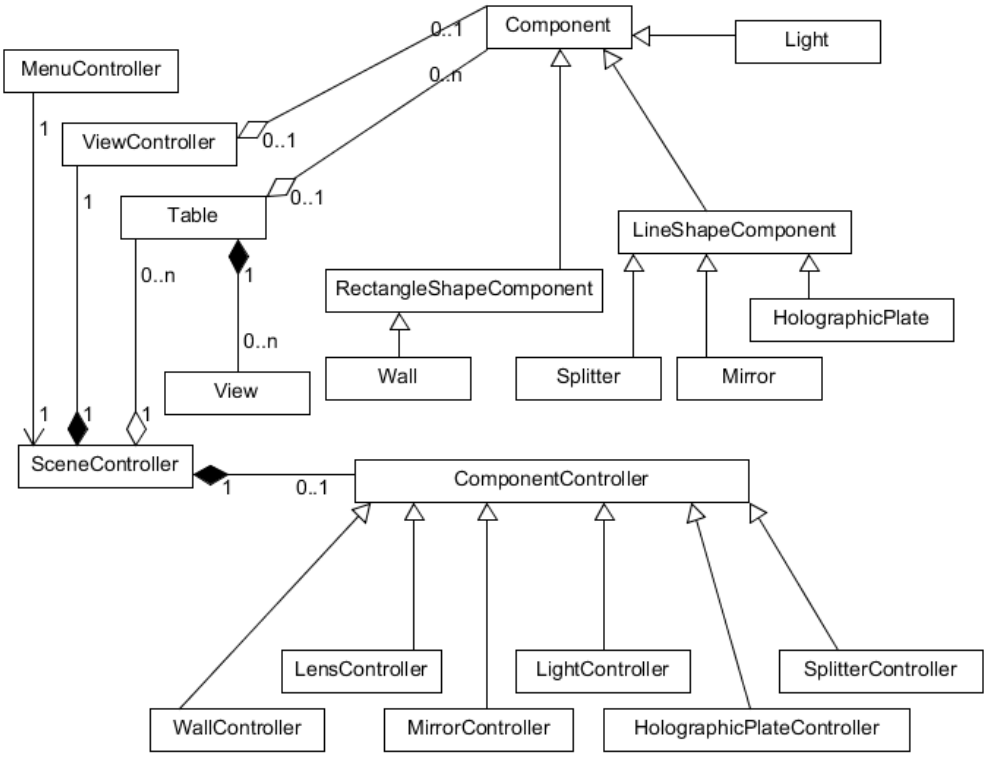
Implementace simulátoru je založena na architektuře Model-view-controller (MVC) a stránka zobrazující simulátor je logicky členěna na několik oblastí,

kde každou oblast má na starost jiný kontroler, který odchytává a zpracovává události generované v této oblasti. Jednotlivé oblasti si nyní popíšeme. První oblastí je horní navigace, která umožňuje nastavovat globální parametry simulace, vyvolat přidání nové komponenty do simulace a provádět import a export existující simulace. Na události generované horní navigací reaguje kontroler `MenuController` ve spolupráci s kontrolerem `SceneController`. Další je oblast, která je pojmenovaná jako scéna a zobrazuje se zde samotná simulace. Tuto oblast spravuje kontroler `SceneController` a stará se o to, aby poskytoval intuitivní rozhraní pro práci se simulací. Vytváří si přitom další dva pomocné kontrolery `ViewController` a `ComponentController`, které scénu dále rozdělují a spravují specifické prvky scény, o kterých si povíme za chvíli. Uvnitř scény se také pracuje se třemi důležitými modely a to `Table`, `View` a `Component`. Tyto modely tvoří základ celé simulace a vztahy mezi nimi jsou následující. Model `Table` reprezentuje jednu konkrétní simulaci, kterou si v simulátoru vytváříme. Do `Table` modelu přidáváme jednotlivé komponenty (potomky třídy `Component`), které vykreslujeme v simulaci a také zde ukládáme pohledy (`View` modely), které nad `Table` modelem vytváříme. Pohledy nám zprostředkovávají náhled do `Table` modelu a definují si nad ním vlastní transformace scény, díky kterým můžeme například obsah simulace přibližovat či oddalovat. Výše zmiňovaný kontroler `ViewController` zpracovává události generované pohledem a také se stará o překreslování pohledu. Kontroler `ComponentController` se pak stará o zpracování událostí přicházejících z konfiguračního panelu komponenty a zařizuje aktualizace `Component` modelu, které se propagují do celé scény a způsobují její překreslování. Pro lepší pochopení struktury simulátoru je ještě uveden následující obrázek 4.2. Celý model simulátoru je také znázorněn na následujícím obrázku 4.3.



Obrázek 4.2: Ukázka struktury programu

Poznámka: přerušované čáry ukazují další zapouzdření, které je vidět pouze při pohledu na zdrojové kódy a není viditelné běžnému uživateli.



Obrázek 4.3: UML diagram simulátoru

4.1 Model komponenty

Každá komponenta, kterou je v simulátoru možné vytvořit, má vlastní model, který ji reprezentuje. Třída `Component` je pouze abstraktní třída, od které jednotlivé komponenty dědí. Některé dědí přímo, jiné skrze další třídu, která přidává atributy a metody, které jsou společné pro komponenty se stejným tvarem. Příkladem jsou třídy `LineShapeComponent` a `RectangleShapeComponent`. Příkladem přímého dědění je třída `Light`, která se podle vybraného typu světla vykresluje jako obdélník, nebo jako kruh.

Tvary komponent jsou definovány sadou bodů. V případě komponenty `Wall` se pak jedná o čtyři rohové body. Tyto body se využívají při aplikaci transformací, které jsou komponentě nastavené. Například pootočení komponenty znamená, že se musí přidat nová transformace. Aby bylo vykreslování komponent efektivní, každá komponenta si udržuje dvě sady bodů. První sada bodů udává polohu komponenty v počátku souřadného systému. Tedy ve stavu kdy na komponentu nejsou aplikovány ještě žádné transformace. Druhá sada bodů pak udává pozici komponenty po aplikování všech transformací.

První sada bodů se využívá v případě, kdy chceme ověřit, zda uživatel vybral komponentu. Aplikují se reverzní transformace pouze na souřadnice kliku myši a ověří se, zda se souřadnice nachází uvnitř komponenty. Při ověřování se ještě komponentě přidává na rozměrech 5 px, aby zde byla malá tolerance nepřesného kliknutí. Druhá sada bodů se pak využívá při pravidelném překreslování scény, kdy již není nutné při každém překreslení počítat transformace bodů. Zde lze namítnout, že dvě sady bodů pro každou komponentu znamenají vyšší nároky na paměť. Získaná rychlost však tuto nevýhodu značně převažuje. Pokud si například vezmeme situaci, kde uživatel uchopí komponentu a bude jí přesouvat po plátně, začne docházet k neustálému překreslování. Během tohoto překreslování by se musely neustále počítat transformované body i pro komponenty, které pouze stojí na místě. Tato úspora je pak znát především na mobilních zařízeních.

Všechny komponenty obsahují kromě metod zajišťujících výše popsané transformace také metody, které se využívají například k interakci s ostatními komponentami simulace a je nutné si tyto metody představit, abychom získali základní představu o vnitřní implementaci komponenty. První je metoda `generateShapePoints()`, ta se volá pokaždé, když manipulujeme s rozměry komponenty a stará se o vytvoření sady bodů reprezentující objekt v počátečním stavu. Dále metoda `isSelected()`, které se předává bod, na který uživatel klikl a ověřuje se, zda se nachází uvnitř komponenty. Před ověřováním se na bod aplikují výše zmíněné reverzní transformace komponenty, aby bylo možné provést jednoduchou kontrolu, kdy je

komponenta v základní poloze. Metoda `copy()` se stará o vytváření klonu komponenty, který pak vrací jako návratovou hodnotu. Využívá se, když chceme v uživatelském rozhraní kopírovat komponentu z jednoho stolu na jiný. Další je metoda `importComponentData()`. Ta na vstupu přijme objekt, který obsahuje data již dříve vytvořené komponenty a tyto data následně do komponenty nakopíruje. Další důležité metody jsou `isIntersection()`, `draw()` a `intersects()`, které si popíšeme až v sekci Vykreslování, kde bude přímo popsáno jejich použití. V kapitole Rozšiřitelnost simulátoru je pak možné se dočíst, jak lze přidávat nové komponenty.

4.1.1 Holografická deska

Výše zmíněné metody obsahuje každá komponenta, ale konečná implementace se ve většině komponent liší. Rozdíly jsou však u většiny komponent minimální. Komponenta, která se velmi liší od všech ostatních, je holografická deska, když pomineme komponentu světlo. Jako jediná funguje jako záznamový materiál. Všechny ostatní komponenty fungují na principu přijmi paprsek a popřípadě vygeneruj výstupní paprsek. Při záznamu dochází k výpočtu frekvence interferenčního vzoru, který zde dopadající paprsky vytvářejí (viz vzorec 4.1). Tato hodnota je následně využita k výpočtu výstupního paprsku (viz vzorec 4.2), když na desku po provedení záznamu znovu dopadají světelné paprsky. Protože při vytváření záznamu nelze dosáhnout nekonečné přesnosti, je tato hodnota počítána na malých viditelných úsecích desky a je v celém úseku stejná. Velikost úseku lze měnit skrze nastavitelný parametr komponenty nazývaný `resolution`.

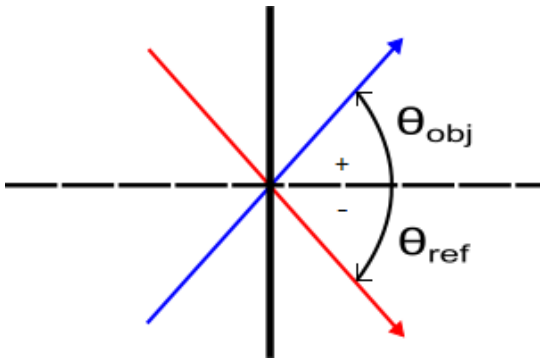
$$f = \frac{(\sin\theta_{\text{obj}} - \sin\theta_{\text{ref}})}{\lambda_1} \qquad \begin{aligned} \sin\theta_{\text{obj}} &- \text{sinus úhlu objektového paprsku} \\ \sin\theta_{\text{ref}} &- \text{sinus úhlu referenčního paprsku} \\ \lambda_1 &- \text{vlnová délka referečního světla} \end{aligned} \tag{4.1}$$

$$\sin\theta_{\text{vys}} = mf\lambda_2 + \sin\theta_{\text{rek}}$$

$\sin\theta_{\text{vys}}$ – sinus úhlu výstupního paprsku
m – řád difrakčního maxima
λ_2 – vlnová délka rekonstrukčního světla
$\sin\theta_{\text{rek}}$ – sinus úhlu rekonstrukčního paprsku

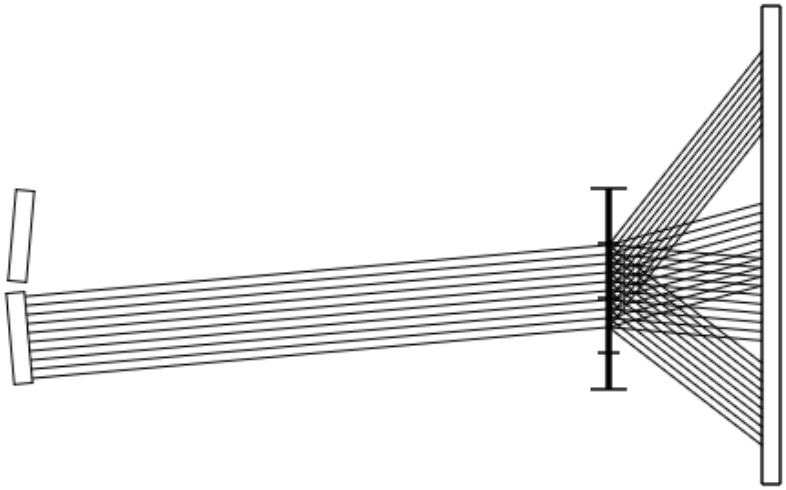
(4.2)

Úhly uváděné ve vzorcích jsou počítány směrem od kolmice holografické desky k dopadajícímu paprsku, tak jak je naznačeno na následujícím obrázku 4.4.



Obrázek 4.4: Úhel je určován směrem od kolmice k paprsku

Vytváření záznamu na holografické desce je pro uživatele dvoukrokový proces. Nejprve je nutné desku nasvítit světly, která chceme zaznamenat a následně provést záznam. Poté je možné desku znovu osvětlovat a sledovat výstupní paprsky. Příklad takového záznamu je znázorněn na následujícím obrázku 6.1.



Obrázek 4.5: Ukázka rekonstrukce zaznamenaného světla

Poznámka: Malé obdélníčky představují světla. Komponenta uprostřed je holografická deska a velký obdélník představuje komponentu nazývanou zeď. Světlo, které nyní negeneruje žádné paprsky, bylo použito jako objektové a druhé jako referenční. Malé čárky na holografické desce představují jednotlivé oddíly, ve kterých se záznam provádí. V tomto případě se zobrazuje plus první a čtvrté maximum a mínus první a čtvrté maximum. Protože je úhel dopadu paprsků velmi nízký, v tomto případě 5°, je možné pozorovat jak první, tak čtvrtá difrakční maxima. S rostoucím úhlem však bude počet maxim ubývat. To si lze jednoduše ověřit. Například dosazením úhlů 45° a -45° do prvního vzorce 4.1 a vyzkoušet si výpočet pro různá maxima druhého vzorce 4.2.

Při pohledu na implementaci je však záznam světla a jeho rekonstrukce tříkrokový proces. V okamžiku, kdy uživatel vybere referenční světlo a potvrdí vytvoření záznamu, následuje první krok, ve kterém dojde k překreslení scény a holografická deska si během překreslování vytváří záznam o světle, které na ní dopadá. Během tohoto kroku si holografická deska v každém úseku pro každý světelný zdroj, který tento úsek osvětluje, ukládá jeden reprezentativní paprsek a úhel, pod kterým na desku dopadá. Pokud se v daném úseku vyskytuje několik paprsků od stejného zdroje, ukládá se paprsek, který má úhel dopadu vypočtený jako průměrnou hodnotu ze všech paprsků stejného zdroje. Tento záznam je vytvářen metodou `recordRay()` a je ukládán do parametru `groups`.

Poté následuje druhý krok, během kterého dochází k výpočtu frekvencí podle prvního vzorce 4.1. Výpočet se opět provádí pro každý úsek desky a

před začátkem výpočtu se vždy kontroluje, zda mají paprsky stejnou vlnovou délku a také rozdíl uražených vzdáleností od světelných zdrojů nesmí být vyšší než nastavený limit. Pokud některá z podmínek není splněna, výpočet se neuskuteční a uživatel je na tyto nedostatky upozorněn. Výpočet se provádí v metodě `createRecord()` a výsledek se opět ukládá do atributu `groups`.

Posledním krokem je vykreslování výstupních paprsků při opětovném nasvícení holografické desky. V okamžiku, kdy je deska znovu nasvícena, dochází k výpočtu výstupních paprsků. Při tomto výpočtu se využívá druhý vzorec 4.2 a podle úseku do kterého paprsek dopadá, se také používá zaznamenaná frekvence, která byla vypočítána v předchozím kroku. Výsledkem tohoto výpočtu je pak úhel, pod kterým bude odchozí paprsek opouštět holografickou desku. Pokud si uživatel vybere zobrazení hned několika maxim najednou, bude se hodnota maxima ve vzorci 4.2 měnit a pro každé zvolené maximum se bude počítat jeden výstupní paprsek.

4.2 Vykreslování

Během vykreslování simulace se kromě výše zmíněných datových modelů pracuje ještě s dvěmi dalšími objekty a to: bodem a paprskem. V obou případech se jedná o jednorozměrná pole obsahující číselné hodnoty. V případě bodu jsou to jeho souřadnice a u paprsku je to v pořadí: počáteční bod, směrový vektor, identifikátor světelného zdroje, délka paprsku měřená od světelného zdroje a vlnová délka světla. Vlnová délka je u paprsku uvedena proto, aby nebylo nutné pro každý paprsek zjišťovat jeho vlnovou délku od světelného zdroje. Jak počáteční bod, tak směrový vektor obsahují tři složky a poslední je vždy nulová. Poslední položka je nulová proto, že se do budoucna uvažuje simulátor upravit tak, aby pracoval ve 3D. Pole paprsku má tedy celkem 9 položek.

Vykreslovaní začíná ve třídě `SceneController` voláním `redrawAll()`. Tato metoda je volána pokaždé, když dochází ke změně, která vyžaduje aktualizaci scény. Například přidání komponenty. Metoda projde všechny pohledy, které právě zobrazený stůl má, postupně je předává třídě `ViewController` a volá nad nimi z této třídy metodu `draw()`, která se již stará o překreslování samotného pohledu. Metoda `draw()` před začátkem vykreslování požádá `View` model (pohled) o poskytnutí grafického plátna, na které se bude vykreslovat. Model `View` pak vrací toto plátno s již aplikovanými transformacemi, které tento pohled má vůči stolu nastavené.

Následně metoda začne procházet všechny komponenty, které stůl obsahuje, a začne nad nimi volat metodu `draw()` a jako parametry jí předává ukazatel na grafické plátno a ukazatel na pole světelných paprsků.

Pole světelných paprsků se využívá k uskladnění paprsků, které některé komponenty generují (například komponenta světlo) a jsou dále zpracovány po vykreslení všech komponent. Výsledek volání této metody pak záleží na typu komponenty. Komponenty, které světlo negenerují, využívají pouze první parametr a na předané plátno se vykreslují. U komponent, které světlo generují, se pak využije i druhý parametr a komponenta během vykreslování přidává paprsky, které sama generuje do pole připravených paprsků.

Po vykreslení všech komponent následuje třetí krok a to: zpracování vygenerovaných paprsků. Paprsky jsou postupně odebírány z pole připravených paprsků a jsou porovnávány vůči všem komponentám, aby se ověřilo, zda nějakou komponentu protínají. Tato kontrola probíhá voláním metody `isIntersection()` nad danou komponentou a předává se jí paprsek pro který kontrolu provádíme. Pokud paprsek komponentu protíná, metoda navrací vzdálenost od počátku paprsku až do bodu průniku. V opačném případě navrací hodnotu `Infinity`. Během této kontroly si zapamatujeme komponentu s nejkratší vzdáleností nad kterou po dokončení kontroly voláme metodu `intersects()` a předáváme jí ukazatel na pole paprsků, které právě zpracováváme. Výsledek volání této metody se bude opět lišit podle typu komponenty. Komponenty, které světlo pouze pohlcují, předávaný parametr ignorují a pouze navracejí bod průniku paprsku a komponenty tak, aby bylo možné určit koncový bod paprsku. Naopak komponenty, které s paprsky dále pracují, jako například zrcátko, předávaný parametr využívají k uložení odchozího paprsku, který příchozí paprsek vygeneruje, aby mohl být následně zpracován. Pokud paprsek žádnou komponentu neprotíná, nevykresluje se. Poté co jsou všechny paprsky vykresleny, metoda končí a volání se navrací opět do třídy `SceneController`, kde je vybrán další pohled a volání se znovu opakuje.

4.3 Podpora mobilních zařízení

Při vývoji simulátoru byla brána v potaz i podpora mobilních zařízení. Simulátor se automaticky přizpůsobuje rozměrům zařízení a jeho funkčnost na mobilních zařízeních není nijak limitována. Limitem jsou pouze hardwarové parametry zařízení, jako je velikost displeje a výpočetní výkon zařízení. Slabší výkon zařízení se může projevit při vytváření složitějších simulací, které jsou náročné na překreslování. Při ověřování funkčnosti jsem používal 7palcový tablet s Androidem s prohlížečem Chrome a iPad 3 s prohlížečem Safari. Menší 7palcový tablet, který má dvoujádrový procesor Intel Atom Z2560 s taktem 1.6 GHz, byl schopen překreslovat o něco vyšší počet paprsků než iPad 3, ale zase 7palcová obrazovka nebyla na práci se

simulátorem příliš vhodná. V obou případech však bylo možné v omezené míře pracovat i se světlem, které generovalo okolo tisíce paprsků, které se dále střetávaly s dalšími komponentami.

4.4 Omezení a nároky

Simulátor nemá žádné specifické požadavky na hardware či na jeho výkon. Je však nutné vzít v potaz náročnost simulace. Pokud budu chtít vytvářet složité simulace, ve kterých se budou vykreslovat tisíce paprsků, porostou nároky na výpočetní výkon a paměť. U slabšího hardware se může stát, že bude v těchto případech simulace reagovat velmi pomalu. Softwarovým nárokem je pak použití moderního webového prohlížeče Firefox, Chrome, Safari, Opera, Microsoft Edge, nebo Internet Exploreru verze 10, nebo 11. Jednoznačným doporučením je webový prohlížeč Chrome, ve kterém simulace pracuje o něco rychleji v porovnání s ostatními prohlížeči.

5 Testování

Při ověřování funkčnosti simulátoru jsem používal sadu simulací, které jsem po skončení úprav do simulátoru vždy naimportoval a ověřoval, zda se stále zobrazují korektně. Tato sada obsahovala všechny základní komponenty, které byly různě natočené a osvětlené světly pod různými úhly.

5.1 Testování s dobrovolníky

Posledním bodem zadání bakalářské práce bylo otestování simulátoru na skupině uživatelů. Předložit jim sadu jevů, které mají nasimulovat, a analyzovat přitom, jak dobře se uživatelům simulátor používá a jak dobře jim pomáhá pochopit simulovaný jev. Během provádění testu byla uživatelům předložena uživatelská dokumentace a seznam úkolů, které mají v simulátoru vykonat. Při tvorbě seznamu úkolů jsem čerpal z [5], kde jsou zmíněny některé běžně používané optické soustavy. Poté, co uživatelé úspěšně dokončili seznam úkolů, tak jim byla položena sada otázek, které měli za úkol zodpovědět a poskytnout tak zpětnou vazbu o kvalitě simulátoru. Uživatelská dokumentace, seznam úkolů a sada předkládaných otázek jsou přiloženy jako přílohy k bakalářské práci.

Testu se zúčastnilo celkem 5 osob a odpovědi dobrovolníků jsou shrnuty v následující tabulce 5.1.

Otázky	Uživatele				
	1.	2.	3.	4.	5.
Použití dokumentace	Ano	Ano	Ano	Ne	Ano
Spokojenost s návrhem simulátoru	Ano	Ano	Ano	Ano	Ano
Porozumění simulovaného jevu	Částečně	Částečně	Ano	Omezeně	Ano
Chybějící komponenta	Ano	Ne	Ano	Ne	Ne

Tabulka 5.1: Odpovědi dobrovolníků

Někteří dobrovolníci se také vyjádřili k nedostatkům simulátoru a chybějícím komponentám. První dobrovolník chtěl do simulátoru přidat zvýraznění právě vybrané komponenty. To bylo také následně přidáno.

V simulátoru mu pak chyběla kulová zrcadla. Druhý dobrovolník chtěl aby se vizuálně odlišil dělič svazku a zrcátko. Třetímu dobrovolníku v simulátoru chyběly tlusté čočky a poslední dobrovolník chtěl do simulátoru přidat komponentu, která by fungovala jako roztažitelné pravítko.

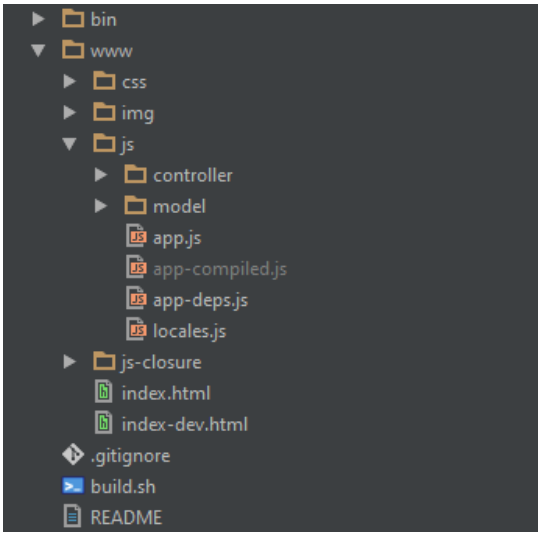
Jelikož v této práci nešlo o implementaci mnoha typů komponent, odkládá se zapracování těchto připomínek do další verze simulátoru.

6 Rozšiřitelnost simulátoru

Při návrhu simulátoru bylo také dbáno na to, aby jej bylo možné v budoucnu jednoduše rozšiřovat. A to například přidáváním nových komponent, úpravou popisků, rozšiřování nabídek menu a přidáváním překladu pro další jazyky.

6.1 Adresářová struktura a překlad programu

Předtím než začneme s rozšiřováním simulátoru, je nejprve nutné se seznámit se simulátorem jako takovým, s jeho adresářovou strukturou a se způsobem, jakým se vytváří produkční verze. Při otevření složky projektu je možné vidět, že obsahuje několik souborů a složek.



Obrázek 6.1: Obsah adresáře projektu

Soubor `README` obsahuje odkazy na užitečné zdroje, které je možné při rozšiřování simulátoru využít. Soubor `.gitignore` a složka `.git` byly vytvořeny verzovacím systémem Git a mohou být smazány v případě, že je vyžadováno využití jiného verzovacího systému. Důležitou složkou je adresář `bin`, ve kterém se nachází `compiler.jar`, který se využívá při sestavování produkční verze.

Kompilátor dělá hned několik věcí najednou. Tou nejdůležitější je, že na vstupu přijímá všechny zdrojové soubory projektu a vytváří jeden výstupní

minifikovaný soubor `app-compiled.js`, který se využívá v produkční verzi. Kompilátor však neprovádí pouze minifikaci, ale také například provádí statickou typovou kontrolu a to pomocí datových typů uvedených v komentářích. Komentáře s datovými typy představují podmnožinu JSDoc anotací, které je kompilátor schopen rozpoznat. O tom, jaké anotace kompilátor podporuje, je možné se dozvědět v oficiální dokumentaci ke Closure kompileru od firmy Google [9]. Dále také umí odstraňovat z kódu mrtvé úseky, které by se nikdy nevykonaly, a mnoho dalších věcí. Přestože je kompilátor přibalenou součástí projektu, je vhodné zkontrolovat před začátkem jakékoliv úpravy, zda není dostupná nová verze kompilátoru. Autoři dělají často aktualizace a sám jsem během tvorby narazil na problém, který vyřešilo stažení nové verze kompilátoru.

Kromě adresáře `bin` obsahuje projekt ještě adresář `www`, ve kterém jsou zdrojové soubory simulátoru. V adresáři `css` se nachází soubor, který definuje vzhled celému simulátoru. Adresář `img` pak obsahuje použité obrázky. Dále jsou zde dva soubory `index.html` a `index-dev.html`, které obsahují HTML kód a definují základní strukturu simulátoru. Jejich obsah by měl být vždy až na malé rozdíly totožný. Rozdíl byl měl být pouze v tom, jaké JavaScriptové soubory se načítají. Soubor `index.html` by měl sloužit jako produkční verze, ve které se již načítá pouze jeden JavaScriptový soubor `app-compiled.js`. Soubor `index-dev.html` by měl sloužit jako vývojová verze, ve které se již načítají všechny JavaScriptové soubory projektu. Při tomto rozdělení na `index-dev.html` a `index.html` není nutné neustále vytvářet minifikovaný soubor a vývoj je o něco rychlejší.

Další důležitou složkou je adresář `js-closure`, ve kterém se nachází knihovna Closure Library, která se v simulátoru používá například k definici jmenných prostorů, či k vytváření tříd a interaktivních dialogů. Stejně jako tomu je u Closure kompileru, i zde je vhodné před začátkem úprav zkontrolovat, jestli nedošlo k aktualizaci knihovny. Odkazy na různé dokumentace a návody ke knihovně lze nalézt v již zmiňovaném souboru `README`. Poslední složkou je adresář `js`. Tento adresář již obsahuje samotné zdrojové kódy simulátoru tak, jak jsou popsány v kapitole Implementace. Soubory jsou dále členěny do podsložek, aby vytvořily logickou strukturu, jak je naznačeno v kapitole Implementace simulátoru na obrázku 4.3.

Kromě logiky programu se zde ještě nachází jeden pomocný soubor `app-deps.js`. Ten je využíván při vývoji a obsahuje mapování závislostí programu. Pokaždé, když v souboru požadujeme nějakou třídu či jmenný prostor voláním `goog.require()`, se tato závislost následně generuje právě do tohoto souboru. Tento soubor nám také pomáhá při načítá JavaScriptových souborů v souboru `index-dev.html`. Není nutné ručně uvádět všechny soubory, které projekt má, ale pouze načíst tento soubor

závislostí. Jeho obsah se však nevytváří automaticky. Vždy, když přidáváme novou závislost, je nutné tento soubor aktualizovat. O aktualizaci se stará soubor `depswriter.py`, který je možné nalézt mezi soubory knihovny. Již nyní se ale celý proces překladu a vývoje stává dosti složitý. Souborům `compiler.jar` a `depswriter.py` je nutné zadávat sadu parametrů, aby správně fungovaly a parametry se těžko pamatují. Proto má projekt ještě jeden soubor `build.sh`, který celý tento proces usnadňuje. Pokud chceme vytvořit novou verzi produkční aplikace, voláme script `build.sh` s argumentem `build`, a pokud pouze přidáváme novou závislost a pracujeme s vývojovou verzí, tak voláme script s argumentem `deps`. K hlubšímu pochopení, jak celý překlad funguje, je vhodné si prohlédnout obsah scriptu `build.sh`. Já však od dalšího popisu upustím a nyní popíši konkrétní příklady rozšíření simulátoru.

Aplikace začíná v souboru `app.js` voláním funkce `start()`. Tu lze přirovnat k volání funkce `main()` v programovacím jazyce Java. Zařizuje spuštění hlavních kontrolerů `MenuControler` a `SceneController`, které již načtou zbytek aplikace a začnou naslouchat na příchozí události aplikace, které pak dále delegují. Soubor `app.js` dále obsahuje pomocné funkce, které jsou používány v celé aplikaci a zajišťují například překlad aplikace, o kterém si povíme nyní.

6.2 Jazykový překlad

Jazyk, ve kterém se simulátor po spuštění zobrazuje, je definován v proměnné `app.locale`. Tuto hodnotu měníme výběrem jazyku v horní nabídce a změna spouští funkci `app.util.translate()`, která provádí překlad. Hodnota, která je ukládána do proměnné `app.locale`, představuje klíč, který je použit k získaní sady popisků, které se nachází v souboru `locales.js`. Při pohledu do souboru `locales.js` je vidět, že každý popisek má svůj identifikátor. Tento identifikátor lze využít dvěma způsoby. Prvním z nich je dát stejný identifikátor HTML elementu, který bude obsahovat překládaný text. Funkce `app.util.translate()` během překladu tyto elementy vyhledá a automaticky přeloží. Druhým použitím je pak přímé volání identifikátoru uvnitř aplikace ve tvaru `app.translation['jmeno-identifikátoru']`. Atribut `app.translation` obsahuje vždy sadu popisků právě vybraného jazyka.

Pokud chceme přidat překlad pro další jazyk, je nutné udělat následující tři kroky. Prvním krokem je v souborech `index.html` a `index-dev.html` přidat novou položku menu s požadovaným jazykem. Druhým krokem je úprava `MenuControlleru` a to konkrétně rozšířením listeneru, který reaguje

na událost výběru jazyka, a přidáním klávesové zkratky pro nový jazyk do metody `addKeyboardShortcuts()`. Posledním krokem je přidání samotného překladu do souboru `locales.js`.

6.3 Přidání komponenty

Přidání nové komponenty opět vyžaduje několik kroků.

1. Vytvoření modelu komponenty.
2. Vytvoření kontroleru pro komponetu.
3. Registrace nového modelu a kontroleru.
4. Přidání komponety do GUI.

Model nové komponenty se přidává do adresáře `www/js/model/elements`. Každý model komponenty musí ať už přímo, nebo skrze nějakou další třídu dědit od třídy `app.model.Component`. Tato třída obsahuje základní metody a atributy společné pro všechny komponenty a také definuje několik abstraktních metod, které je nutné doimplementovat. Jmenovitě se jedná o metody `isIntersection()`, `draw()`, `isSelected()`, `generateShapePoints()`, `importComponentData()`, `copy()`, `intersect()`. Jejich použití již bylo popsáno v kapitole Implementace simulátoru. Během implementace je nutno brát v potaz, že metody jako `isIntersection()`, `draw()` a `intersect()` se spouští pokaždé, když dojde k nějaké aktualizaci scény a některé i několikrát během jednoho překreslení. Je tedy vhodné logiku uvnitř těchto metod ponechat co možná nejjednodušší, aby nedocházelo ke zbytečnému zpomalování. Je také vhodné při implementaci využít metody `normalize2DVector()`, `rotatePoint()`, `reverseTransformPoint()`, `transformPoint()`, které jsou definované ve třídě `Component` a nevytvářet vlastní implementaci těchto metod.

Po dokončení implementace modelu následuje implementace kontroleru komponenty. Podobně jako tomu bylo u modelu, i zde je nutné dědit od obecného kontroleru `ComponentController`, který poskytuje základní akce a obsluhu, které jsou pro všechny komponenty společné. Tyto kontrolery se využívají v okamžiku, kdy uživatel vybral komponentu ve scéně a nechal si zobrazit postranní konfigurační panel komponenty. Právě obsah panelu a události nad ním definované se vytvářejí v těchto kontrolerech. Při implementaci kontroleru je nutné implementovat dvě metody a to: `showComponentControlPanel()` a `addPanelListeners()`.

Metoda `showComponentControlPanel()` se stará o přidávání kontrolek do panelu a metoda `addPanelListeners()` pak přidává události, které je možné nad kontrolkami spouštět. Obě tyto metody jsou již v základní formě implementovány ve třídě `ComponentController` a přidávají například kontrolu polohy komponenty a množina tlačítek, které ovlivňují existenci komponenty. Nesmí být proto přímo překryty novou implementací, ale měly by je dále rozšiřovat, stejně jako to dělají všechny ostatní komponenty.

V okamžiku, kdy máme hotovou implementaci modelu a kontroleru, je nutné tyto dvě třídy zaregistrovat ve třídě `app.SceneController`. Nový model v metodě `createComponentModel()` a kontroler v metodě `setSelectedComponent()`.

Posledním krokem je přidat komponentu do nabídky v uživatelském rozhraní. Obdobně jako u překladu je tedy nutné rozšířit soubory `index.html` a `index-dev.html` a přidat novou položku reprezentující komponetu. Také je nutné pro tuto komponentu přidat překlad do souboru `locales.js` a vytvořit pro ně obslužnou událost ve třídě `MenuController`.

7 Závěr

Cílem bylo vytvořit interaktivní výukový program, který by uživatelům dovolil vytvářet různé optické soustavy a sledovat přitom chování světla, které touto soustavou prochází. Nebylo cílem vytvořit dokonalý simulační program difraktivní optiky, ale především program, který dokáže zjednodušeně vizualizovat chování světla v optických soustavách. Tento cíl se také podařilo splnit a vznikl simulátor realizovaný v HTML5 a JavaScriptu. Simulátor je tedy možné pustit v moderním webovém prohlížeči a až na drobná omezení (viz sekce Omezení a nároky v kapitole Implementace) jej lze plně využívat i na mobilních zařízeních. Oproti předchozím verzím tedy přináší podporu mobilních zařízení a také odstraňuje potřebu instalace dodatečných modulů do webového prohlížeče, stejně jako tomu je například u Java Appletů.

Text této práce obsahuje seznámení s paprskovým modelem světla a základy holografie, popis výběru nástrojů použitých k realizaci programu a odůvodnění výběru. Dále také popis implementace simulátoru a jeho omezení, způsob ověřování funkčnosti, testování s dobrovolníky a na závěr možnosti rozšíření programu. S ohledem na rozsah programu nejsou v textu nikde uvedeny zdrojové kódy programu a implementace je pouze slově popsána.

Do budoucna by mělo dojít ještě k rozšiřování simulátoru a během implementace na to byl brán zřetel. Možnostem rozšíření je rovněž věnována jedna kapitola textu. Pro autory budoucích rozšíření bude pravděpodobně nesložitější se seznámit s nástroji Google Closure. Text ale obsahuje odkazy na užitečné návody, které autorům pomohou začít. Vhodným rozšířením by bylo také přidání testů ověřujících kvalitu software, na které vzhledem k rozsahu práce již nezbyl čas.

Literatura

[1] BOLIN, Michael. Closure: The Definitive Guide. Beijing: O'Reilly Media, 2010. ISBN: 1-4493-8187-1.

[2] ŽÁRA, Jiří. Moderní počítačová grafika. 2. vyd. BENEŠ, Bedřich, SOCHOR, Jiří, FELKEL, Petr. Brno: Computer Press, 2004. ISBN: 80-251-0454-0.

[3] RENDL, Kamil. Vizualizace principu hologramu.
Plzeň, 2012. Bakalářská práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd. Vedoucí práce Petr LOBAZ.

[4] RENDL, Kamil. Simulátor optických jevů.
Plzeň, 2015. Diplomová práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd. Vedoucí práce Petr LOBAZ.

[5] HADAČEK, Michael. Vizualizace principu hologramu.
Plzeň, 2014. Bakalářská práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd. Vedoucí práce Petr LOBAZ.

[6] LOBAZ, Petr. Počítačem generovaná holografie. [online]. Západočeská univerzita v Plzni, Fakulta aplikovaných věd. Poslední změna 1.6.2015 [cit. 12.11.2015]. Dostupné z: http://holo.zcu.cz

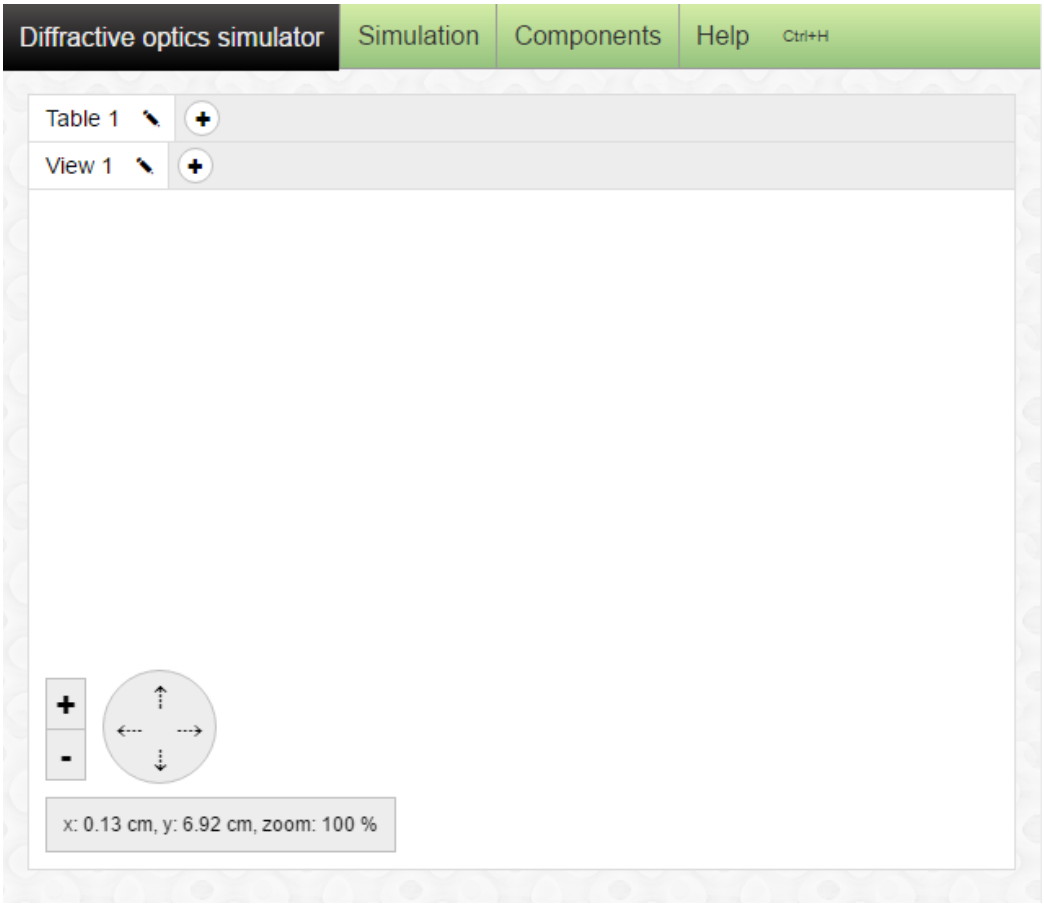
[7] ASHKENAS, Jeremy. Seznam jazyků, které je možné překládat do JavaScriptu. *GitHub* [online]. GitHub, Inc. Poslední změna 18.11.2015 [cit. 23.11.2015]. Dostupné z: https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js

[8] Webové API. *Mozilla Developers* [online]. Mozilla Developer Network and individual contributors. Poslední změna 10.3.2016 [cit. 23.11.2015]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API

[9] Closure Tools. *Google Developers* [online]. Google Inc. Poslední změna 16.11.2015 [cit. 2.2.2015]. Dostupné z: https://developers.google.com/closure

A Uživatelská dokumentace

Simulátor lze spustit z přiloženého CD a to otevřením souboru `index.html` ve svém webovém prohlížeči. K zobrazení je možné použít libovolný široce používaný webový prohlížeč (Chrome, Firefox, Microsoft Edge, Opera, Safari). Po zobrazení stránky uvidíte prázdný simulátor (viz obrázek A.1), který má vytvořený pouze jeden prázdný stůl, který zobrazuje jeden pohled. Vizuálně lze simulátor rozdělit na horní nabídku a scénu starající se o vykreslování simulace.



Obrázek A.1: Úvodní obrazovka

A.1 Globální parametry

Simulátor má řadu globálních parametrů, které ovlivňují vykreslování celé simulace. Tyto parametry je možné nastavovat skrze horní nabídku na záložce Simulation/Settings. První parametr Count of reflections udává, kolikrát se může paprsek odrazit, než je jeho šíření zastaveno. Tento parametr je zde hlavně proto, aby nedocházelo k zacyklení paprsku. Například pokud by se paprsek dostal mezi dvě zrcátka. Drobnou nevýhodou je pak nutnost si tuto hodnotu hlídat. Mohlo by se vám totiž stát, že si bude například myslet, že na holografické desce záznam nevznikl, protože při osvětlení žádné světlo z desky nevychází. Přitom je pouze nutné zvýšit tento globální parametr o jedna.

Dalším parametrem je parametr Screen diagonal, který je spíše pomocný a je možné jej využít v případě, kdy chceme, aby velikosti uváděné a zobrazované v centimetrech skutečně odpovídali reálným hodnotám. Pak je možné do tohoto parametru zadat úhlopříčku vašeho monitoru v palcích a program si již správné velikosti přepočítá.

Parametr Coherence length nyní úmyslně přeskočím a popíši ho až při popisu holografické desky. Poslední parametr pak slouží k výběru jazyka simulátoru.

A.2 Manipulace s komponentou

Dostupné komponenty si lze zobrazit na záložce Components. Po kliknutí na vybranou komponentu budete vyzváni, abyste kliknutím na plátno komponentu přidali. Váš kurzor se přitom změní na černý kříž, který upozorňuje, že jste ve stavu přidávání komponenty. Alternativně je také možné využívat klávesové zkratky, které jsou u jednotlivých komponent definované.

Poté, co komponentu na plátno přidáte, se nám na levé straně objeví konfigurační panel komponenty. Tento panel vám dovoluje nastavovat komponentám různé vlastnosti a v závislosti na typu komponenty se jeho obsah mění. Co je pro všechny komponenty společné, je možnost nastavit pozici a natočení komponenty. Také mají všechny komponenty na konfiguračním panelu tři tlačítka, která vám dovolují panel znovu zavřít, smazat komponentu a zkopírovat komponentu na jiný dostupný stůl. Při překopírování na nový stůl se komponenta objeví na stejných souřadnicích. Dále si popíšeme specifické parametry každé komponenty.

Světlo

U světla lze měnit jeho typ a to na: bodové, nebo plošné. V základním nastavení je plošné. U plošného světla se všechny paprsky šíří jedním směrem. U bodového ještě s využitím parametru rádius se paprsky šíří do všech stran podle velikosti rádia. Parametr vlnová délka světla se nám hodí v okamžiku, kdy pracujeme s holografickou deskou, o které si povíme v následujícím textu. Parametr udávající rozměr nám v případě plošného světla určuje jeho výšku a u bodového jeho poloměr.

Zeď

Zeď je komponenta primárně určená k zakrývání paprsků a kromě rozměru neobsahuje žádné další parametry. Obdobně je tomu u zrcátka a děliče. Opět kromě rozměrů již neobsahují žádný další parametr.

Čočka

Čočka má parametry, které určují její rozměr, typ a ohniskovou vzdálenost.

Holografická deska

Holografická deska je nejzajímavější komponentou v celém simulátoru. Důležitým parametrem, který holografická deska má, je rozlišení. Tímto parametrem nastavujeme, na kolik částí bude holografická deska rozdělena, tedy na kolika místech se bude počítat frekvence interference podle vzorečku A.1. Tento výpočet probíhá v okamžiku, kdy provedeme záznam kliknutím na tlačítko `Make record` a vybereme referenční světlo. Při výběru referenčního světla je možné si všimnout, že v některých případech se nám nezobrazují pouze ID světelných zdrojů, ale například ID ve tvaru 1-2S-1M a 1-2S-2M. Tato ID nám říkají, že světlo pochází ze stejného zdroje, to nám říká první číslo, ale po cestě se paprsky rozdělily. První paprsek šel nejprve prošel děličem s ID 2 a pak zrcátkem s ID 1. Druhý prošel opět děličem s ID 2 a následně zrcátkem s ID 2. Možné hodnoty jsou: M pro zrcátko, L pro čočku, S pro dělič a H pro holografickou desku.

Po provedení záznamu je možné desku znovu osvětlovat a sledovat výstupní paprsky. Úhel výstupních paprsků se pak počítá podle vzorečku A.2. A podle toho, kolik si vybereme difrakčních maxim, tolik uvidíme i výstupních paprsků. Je však nutné si uvědomit, že ne vždy uvidíme všechny paprsky, protože v některých případech by byl úhel příliš velký. Důležitým globálním parametrem je Coherence length. Tento parametr říká, jak velký může být rozdíl ve vzdálenostech paprsků od světelných zdrojů dopadajících na holografickou desku, aby

byly ještě přijaty a vytvořil se v daném úseku záznam.

$$f = \frac{(\sin\theta_{\text{obj}} - \sin\theta_{\text{ref}})}{\lambda_1}$$

$\sin\theta_{\text{obj}}$ – sinus úhlu objektového paprsku
$\sin\theta_{\text{ref}}$ – sinus úhlu referenčního paprsku
λ_1 – vlnová délka referečního světla

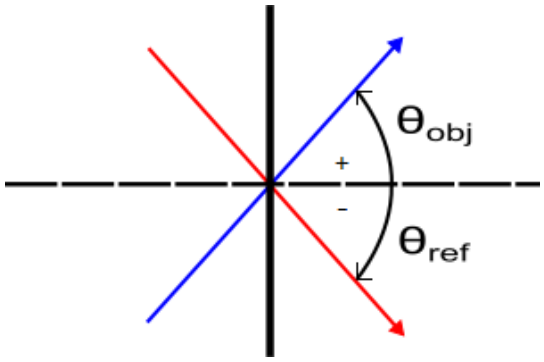
(A.1)

$$\sin\theta_{\text{vys}} = mf\lambda_2 + \sin\theta_{\text{rek}}$$

$\sin\theta_{\text{vys}}$ – sinus úhlu výstupního paprsku
m – řád difrakčního maxima
λ_2 – vlnová délka rekonstrukčního světla
$\sin\theta_{\text{rek}}$ – sinus úhlu rekonstrukčního paprsku

(A.2)

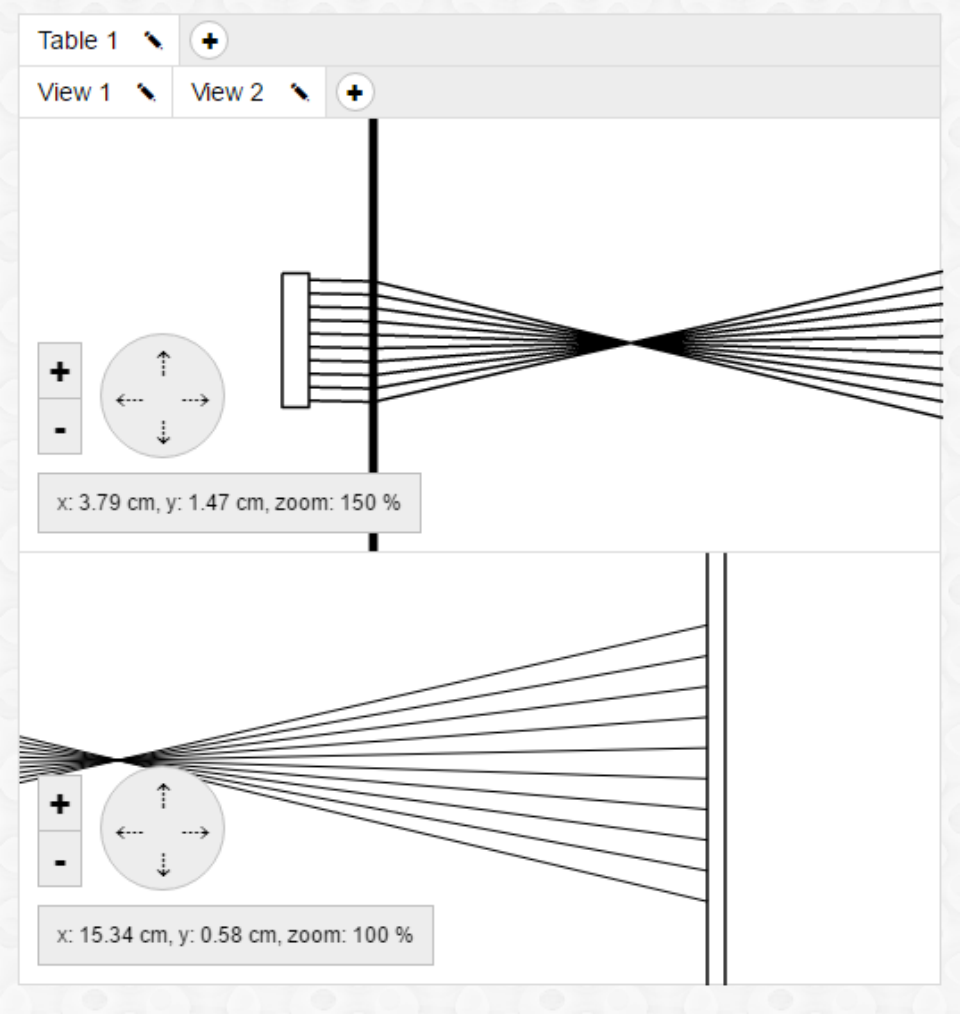
Úhly uváděné ve vzorcích jsou počítány směrem od kolmice holografické desky, tak jak je naznačeno na následujícím obrázku A.2.



Obrázek A.2: Úhel je určován směrem od kolmice k paprsku

A.3 Pohledy a stoly

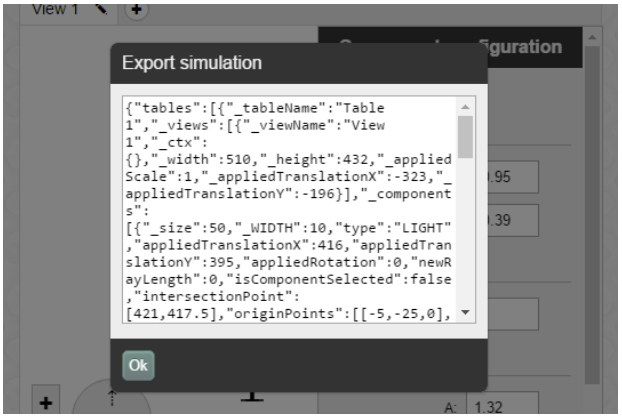
Zatím jsme si pouze popsali jednotlivé komponenty, ale k tomu abychom mohli začít z komponent skládat optické soustavy, je ještě potřeba se seznámit s pohledy a stoly. Stoly si lze představit jako opravdové fyzické stoly, na které stavíme jednotlivé komponenty. Jednotlivé komponenty jsou tedy umisťovány absolutně vzhledem ke stolu, na kterém se nachází. Pohledy si pak lze představit okna, skrze která se na stůl díváme. Pohledem se lze po stole různě posouvat a také si jeho obsah přibližovat a oddalovat, viz následující obrázek A.3.



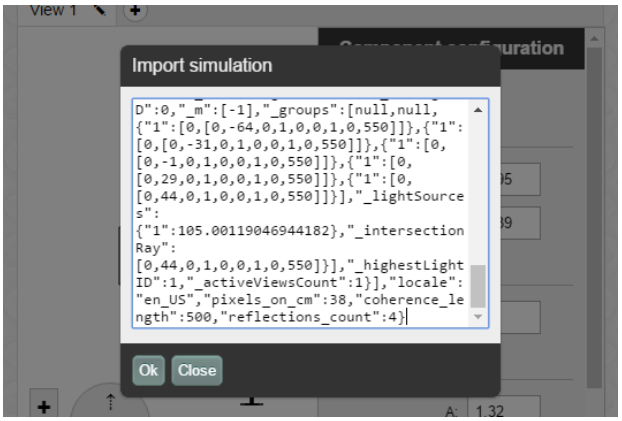
Obrázek A.3: Ukázka práce s pohledy

A.4 Export a import

V případě, že si chceme rozdělanou simulaci uchovat, nebo jí například poslat známému, je možné využít import a export funkcí, které jsou v simulátoru dostupné. Na záložce Simulation je položka Export simulation. Ta po kliknutí vyvolá dialogové okno obsahující data simulace ve formátu JSON (JavaScript Object Notation). Tato data je možné zkopírovat do textového souboru a uchovat je tak na pozdější použití. Simulaci je možné znovu načíst kliknutím na záložku Import simulation. Ta opět vyvolá dialogové okno, do kterého je možné dříve uložená data simulace znovu nakopírovat. Po stisknutí tlačítka Ok dojde k zobrazení simulace.



Obrázek A.4: Ukázka exportu dat

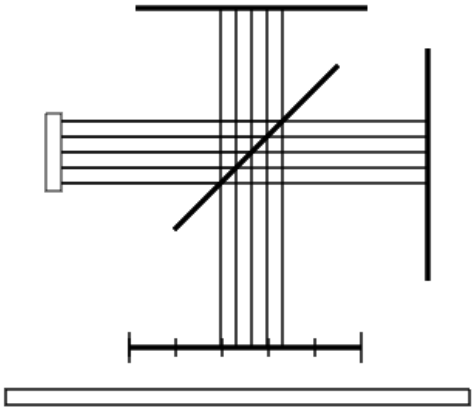


Obrázek A.5: Ukázka importu dat

B Seznam úkolů

V simulátoru se pokusíme nasimulovat dva interferometry a vytvoříme si příklad tzv. Off-axis hologramu. Princip Off-axis hologramu je vysvětlen u úkolu. Interferometr je zařízení, které se využívá k velmi přesnému měření. Jeho princip je založen na interferenci světla. Měří se pomocí něho například index lomu u plynů a kapalin.

1. Jako první si vyzkoušíme sestavit Michelsonův interferometr. Přejmenujte první stůl z Table 1 na Michelsonův interferometr. K jeho sestavení je potřeba mít světelný zdroj, dvě zrcátka, dělič svazku a materiál k záznamu interference. V případě simulátoru je to holografická deska. Dále potřebujeme zeď, na kterou promítneme vystupující paprsky zaznamenané na holografické desce. Jednotlivé komponenty rozestavíme podle obrázku B.1.

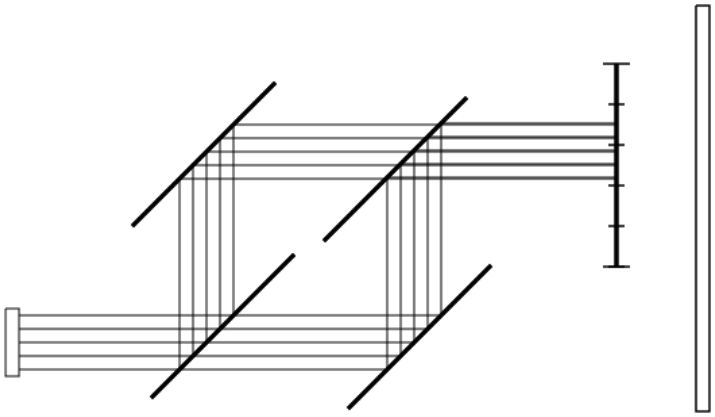


Obrázek B.1: Michelsonův interferometr

Poté, co jsou komponenty rozestaveny, se pokuste několikrát vytvořit záznam na holografické desce a postupně přitom oddalovat jedno ze zrcátek, dokud nenarazíte na limit délky koherence světla. V okamžiku, kdy se již záznam nepovede provést, je nutné použít světlo, které má vyšší délku koherence světla. V případě simulátoru je možné tuto vzdálenost zvýšit a to úpravou globálního parametru Coherence length v menu programu.

2. Nyní si vyzkoušíme ještě jeden interferometr a to: Mach-Zehnderův interferometr. V holografii se tento interferometr používá především

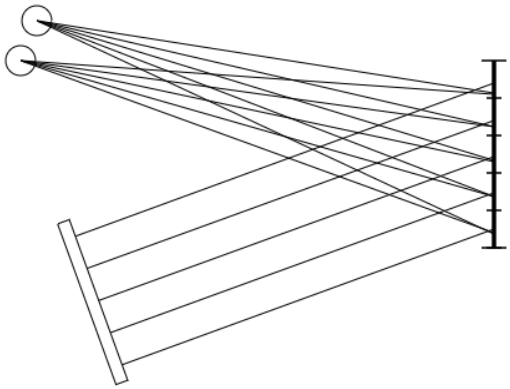
pro zjištění odolnosti optické soustavy vůči otřesům. Vytvoříme si tedy nový stůl a pojmenujeme ho Mach-Zehnderův interferometr. Zde lze s výhodou využít kopírování komponent z jednoho stolu na druhý, protože k sestavení tohoto interferometru je potřeba mít opět světelný zdroj, dvě zrcátka, dva děliče svazků, holografickou desku a zeď. Také je možné využít přidávání pohledů, pokud se vám sestava na monitor nevejde. Komponenty rozmístíme podle následujícího obrázku B.2.



Obrázek B.2: Mach-Zehnderův interferometr

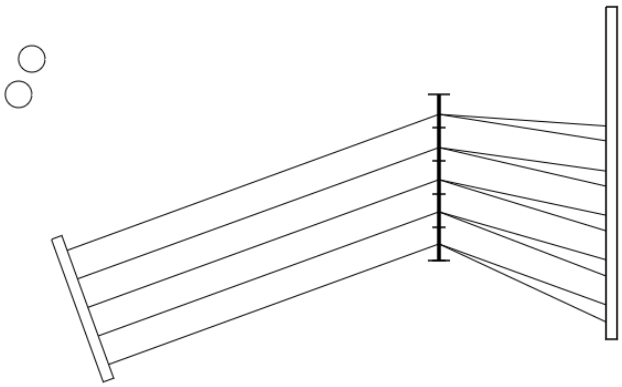
Opět je možné si vyzkoušet provést záznam na holografickou desku. Především je však vidět, že posuvem děličů zpřesňujeme šíření paprsků v soustavě. I drobná změna se může výrazně projevit.

3. Nakonec si vyzkoušíme vytvořit Off-axis hologram. Při této metodě záznamu jsou jak objektové, tak referenční paprsky umístěny mimo osu (Off-axis). Tyto hologramy se používají například k záznamu neprůhledných objektů (viz obrázek 2.8). K tvorbě tohoto hologramu namísto objektu použijeme dva bodové zdroje světla, které budou reprezentovat náš objekt. Dále budeme opět potřebovat holografickou desku, zeď a další světlo, které bude představovat referenční světlo. Komponenty rozmístíme opět podle následujícího obrázku B.3.



Obrázek B.3: Příprava Off-axis hologramu

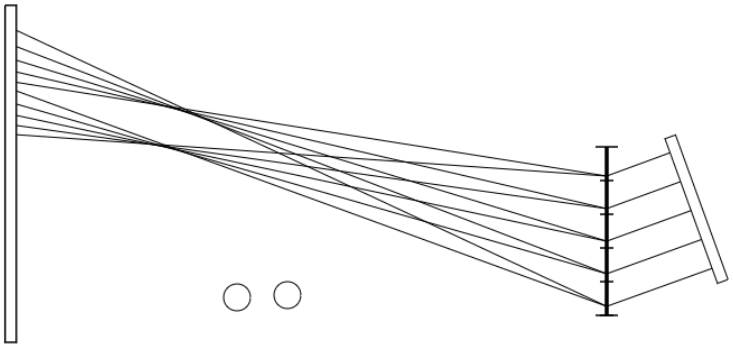
Paprsky se nemusí na holografické desce přímo střetávat, ale je nutné, aby v každém úseku desky byl od každého světelného zdroje jeden reprezentant (jak je naznačeno na obrázku). Po provedení záznamu vypneme objektová světla (nastavíme počet paprsků na 0) a necháme si zobrazit plus první difrakční maximum. Zde vidíme, že referenční světlo (nyní již rekonstrukční) osvětlující holografickou desku vytváří výstupní paprsky ve směru původních objektových světel.



Obrázek B.4: Rekonstrukce objektového světla

Pokud po vytvoření záznamu desku ještě nasvítíte z opačné strany a necháte si zobrazit mínus první difrakční maximum, je možné určit

pozici původních objektových světel. V tomto případě musí desku osvětlovat rekonstrukční paprsky ve stejném směru, ve kterém by jí opouštěly paprsky referenčního světla. Pouze jsou opačně orientované.



Obrázek B.5: Určení pozic objektových světel

4. Na závěr otevřete ještě jednu záložku se simulátorem v prohlížeči a naimportujte sem hotovou simulaci, abyste si vyzkoušeli práci s import a export funkcemi programu.

C Dotazník

1. Museli jste při používání simulátoru využít uživatelskou dokumentaci?
   (a) ANO
   (b) NE

2. Jste spokojeni s rozložením prvků v simulátoru? Případně je něco, co byste rádi na simulátoru změnili?
   (a) ANO
   (b) NE

   Změnil/a bych: _____

3. Pomohl vám simulátor porozumět simulovanému jevu?
   (a) ANO
   (b) ANO, ALE JEN ČÁSTEČNĚ
   (c) JEN VELMI OMEZENĚ
   (d) NE

4. Chyběla vám v simulátoru nějaká komponenta? Jaká?
   (a) ANO
   (b) NE

   V simulátoru chybí: _____