TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

DIPLOMOVÁ PRÁCE

LIBEREC 2011

JAN QUAISER

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: M2612 - Elektronika a informatika

Studijní obor: N2612 - Informační technologie

VIZUALIZACE OZÁŘENÍ TVAROVĚ SLOŽITÉ FORMY PRO VÝROBU UMĚLÝCH KŮŽÍ

VISUALISATION OF ARTIFICAL SKIN COMPLICATEDLY SHAPED MAKING FORMS ILLUMINATION

Diplomová práce

Autor: ..... Bc. Jan Quaiser

Vedoucí práce: ... Ing. Jiří Hnídek

Konzultant: ..... Ing. Martin Živný

V Liberci 28.3.2011

Zadání

Prohlášení

Byl jsem seznámen stím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména §60 - školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřbu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL. V tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

1.4.2011

Podpis .....

Poděkování

Velmi rád bych v úvodu této práce poděkoval a vyslovil uznání všem, kteří mi pomáhali při vzniku této práce. Především Ing. Jiřímu Hnídkovi, vedoucímu mé diplomové práce za trpělivé vedení a množství praktických rad. Dále Ing. Martinu Živnému a celé společnosti Lenam s.r.o. za jejich trpělivost a podporu.

Abstrakt

Diplomová práce je součástí projektu, který se zabývá inovací technologií na výrobu umělých kůží. Jde o spolupráci tří subjektů a to Technické univerzity v Liberci, společnosti Lenam s.r.o. a společnosti Magna.

Cílem diplomové práce je vytvořit program, který urychlí proces rozmisťování infračervených (IR, infrared) zářičů pro ohřev forem pro umělou kůži. Pro dosažení tohoto cíle bylo potřeba implementovat nové funkce do vhodně zvoleného programu. Po provedení multikriteriální analýzy nakonec vzešel Blender jako nejvhodnější kandiát. Po implementaci těchto rozšíření již není zapotřebí několika programů a specialistů na práci v těchto programech, ale vše je spojeno v jeden program se všemi nezbytnými funkcemi.

Mezi základní rozšiřující funkce patří funkce pro detekci kolizí zářičů a dále pak funkce pro zviditelnění stop zářičů. Implementovány jsou tři základní funkce pro zobrazování stop. Jsou to geometrická metoda (GEOM), metoda založená na funkci definovaném rozložení vyzářené energie na ploše (FUNC) a metoda založená na výpočtu radiosity (RAD.) Dále byly vytvořeny funkce pro měření vzdáleností jednotlivých objektů, zjišťování různých statistik apod. Pro tyto funkce bylo potřeba také vytvořit ovládací prvky a upravit grafické uživatelské rozhraní tak, aby bylo co nejvíce podobné jiným, které se používají v profesionálních počítačem podporovaných projektech (CAD, coputer aided design).

Jednotlivé kapitoly jsou členěny do tří částí. První část je vždy věnována teoretickým poznatkům. Tyto poznatky jsou zužitkovány v druhé části, která se již zabývá konkrétními postupy. Na konci každé kapitoly jsou pak uvedeny výsledky a stručné zhodnocení nové funkce. Všechny popisované rozšíření jsou naprogramovány v jazyce Python. V závěru jsou uvedeny výstupy, kterých lze pomocí výsledné aplikace dosáhnout.

Klíčová slova - Blender, Python, radiační metody, kolize, umělá kůže

Abstract

The thesis is a part of a project which deals with innovation of technologies for production of imitation leather. The project is the cooperation of three subjects, the Technical University of Liberec, Lenam Ltd. Company and Magna Company.

The aim of this thesis is to create a program which accelerates the process of arranging of IR emitters for imitation leather forms heating. To reach this aim it was necessary to implement new functions into Blender program. Thanks to these new functions there are not needed several programs and specialists for work with these programs but everything is united into one program with all indispensable functions.

Among the basic extending functions there belongs the function for detection of emitters collisions and also the function for making emitters tracks visible. There are implemented three basic functions, GEOM, FUNC and RAD into the program. Then there were created functions for measuring distances of single objects, for surveying statistics etc. For these functions it was also necessary to create control components and adjust graphic user interface to be the most resembling another ones used in professional CAD systems.

The first part of this thesis deals with theoretical information which is utilized in the second part that deals with particular procedures. These led to successful realization of the whole program. Programming language Python is used for Blender program and that is why all mentioned extending modules are programmed in this language. In the conclusion there are stated results which are possible to achieve thanks to the application.

Keywords - Blender, Python, radiating methods, collision, artificial skin

Obsah

Seznam použitých zkratek

BMW	Bavarian Motor Works
OS	Operating System
GSS	Globální souřadný systém
LSS	Lokální souřadný systém
IGS	Initial Graphics Exchange Specification
NAS	Nastran
VRML	Virtual Reality Modeling Language
GPL	General Public License
LGPL	Lesser General Public License
SAT	Separating Axis Test
OBB	Orientable Bounding Box
AABB	Axis Align Bounding Box
2D	Two-Dimensional
3D	Three-Dimensional
STL	Standard Library
IRE	Infrared Emitter
IR	Infrared
GEOM	Geometrická metoda
FUNC	Metoda založená na funkcí definovaném rozložení vyzářené energie
RAD	Metoda založená na výpočtu radiosity

Seznam obrázků

Seznam tabulek

1 Úvod

Konkurence nikdy nespí a obzvlášť to platí v automobilovém průmyslu. Společnost Magna Cartech je jedním z největších dodavatelů dílů pro automobilový průmysl. Na českém trhu působí od roku 1996 a jejími zákazníky jsou například BMW, Škoda, Volkswagen a další. Magna spolupracuje s Libereckou společností Lenam, která ve specializovaných softwarech realizuje některé úkoly spojené s výrobou umělých kůží. Protože vývoj jde neustále kupředu a je třeba neustále zrychlovat, zkvalitňovat a zlevňovat výrobu, byl spuštěn projekt pro inovaci technologie výroby umělých kůží. Díky spolupráci firmy Lenam a Technické univerzity v Liberci jsem se dostal k této zajímavé diplomové práci.

Mezi zaměření firmy Lenam patří provádění výpočtů pro rozmístění IR zářičů kolem tvarově velmi komplikované formy, jakou je například palubní deska automobilu. Umělá kůže, která je výsledkem celého procesu, se následně používá k potahování vnějšího povrchu palubní desky, aby dostala svoji konečnou podobu, tak jak ji známe z interiéru automobilu.

Celý proces začíná nanesením práškového elastického elastomeru na kovovou formu, na níž se prášek nataví a slinuje do tenké kompaktní vrstvy. Po ochlazení je celý výrobek sejmut a díky formě dostane žádoucí tvar a po celém povrchu jemný reliéfní dezén. K dosažení výstupní kvality výrobku je však potřeba dodržet určitou teplotu slinování, která se pohybuje kolem 220 C v úzkém tolerančním intervalu. To vyžaduje co nejrovnoměrnější rozložení teploty kolem celé formy, která je obvykle velice tvarově složitá, a proto tento požadavek není snadné dodržet.

K nalezení optimálního rozmístění zářičů kolem formy se používá několika programů, jejichž vstupy a výstupy jsou na sobě závislé. Úkolem této diplomové práce bylo vytvořit jeden program, který by zastoupil všechny tyto programy, jejich funkce a případně doplnil nějaké další. Výsledný program by měl vést k urychlení procesu hledání optimálního rozmístění zářičů a měl by obsahovat pouze jeden vstup v podobě modelu formy a výstup obsahující quaterniony pro robota umisťujícího na výrobní lince IR zářiče.

Model formy se do programu importuje pomocí skriptů, které řešil v rámci své bakalářské práce Zbyněk Hlava. Kolem modelu následně uživatel rozmístí předdefinované typy zářičů. Následně lze v jakékoliv fázi rozmisťování zářičů vyvolat nově napsané funkce. Základem jsou funkce pro zobrazení ozáření formy, nebo kolidujících zářičů. Uživatel má také k dispozici některé další pomocné funkce, které budou podrobně popsány v dalších částech této práce.

Jako základní stavební kámen byl pro provedení multikriteriální analýzy vybrán program Blender, který umožňuje uživatelům přidat různé rozšiřující funkce v podobě skriptů. Tyto skripty lze psát v jazyce Python a po přidání do programu Blender se uživateli zpřístupní nově napsané funcke. Vyvolávat je následně může pomocí příkazového řádku, či ovládacích prvků, které si některé skripty sami generují.

První skript řeší vkládání objektů, které reprezentují zářiče, jež se budou rozmisťovat kolem formy. Dále byly vytvořeny další funkce pro detekci kolizí a obarvení formy. Všechny tyto funkce byly vytvořeny na základě konzultací s odborníky nebo dat, která se získala měřením. Poté byly z funkcí sestaveny skripty, které lze použít ve výsledném programu tak, aby chování programu a jeho výsledky co nejvíce odpovídaly požadavkům definovaným v počátcích projektu.

2 Výběr programu

Prvním úkolem byl výběr vhodného programu pro implementaci potřebných rozšíření. V zadání diplomové práce je již zmíněn program Blender, ale bylo potřeba toto rozhodnutí ještě řádně argumentovat a prověřit i jiné alternativy, které navrhovali zaměstnanci společnosti Lenam.

První diskuze se vedly nad dvěma variantami. Jednou z nich bylo využití nějakého programu jako základu s možností implementovat do něj potřebná rozšíření. Druhá varianta pak spočívala v tvorbě úplně nového programu. Volba padla na využití nějakého programu, který bude sloužit jako základ a pomocí rozšiřujících modů se do něj doplní potřebné funkce. Hlavním argumentem pro využití již existujícího programu jako základ, byla časová omezenost projektu, v rámci kterého vznikla tato diplomová práce. Začít tvořit celý program od začátku by znamenalo velkou časovou náročnost na programování a odladění, tak aby fungoval obstojně a splňoval všechny požadavky, jež vyžadovala společnost Lenam. Základními funkcemi, které měl program splňovat, byly například ovládání pohledu do 3D scény, manipulace s objekty a podobně. Ovládáním zobrazení je myšlena například změna pohledu na scénu, posun, otáčení, zoomování, zobrazení celé scény a předdefinované pohledy. Mezi další nezbytné funkce patří manipulace s objekty. S objekty je potřeba posouvat ve směru os manuálně, nebo pomocí nastavení konkrétní délky posunu, dále transformace vektorem a otáčení kolem os. Všechny tyto funkce již obsahuje spousta existujících programů. Proto by bylo zbytečné začínat programovat od začátku, když již existuje velké množství programů, které jsou odladěné, a spousta lidí je již nějakou dobu úspěšně používá a také podporují možnost jejich rozšiřování pomocí vestavěných modulů.

2.1 Navrhované alternativy

Následující kapitola popisuje několik programů, které navrhovali pracovníci společnosti Lenam. Jedná se o výběr několika neplacených programů, splňujících základní požadavky na funkcionalitu programu. Mezi navrhovanými alternativami byly programy Blender, Free CAD, K-3D, Salome a Heekscad. Programy a jejich funkce jsou zevrubně popsány a následuje multikriteriální analýza Na základě jejich výsledků byl vybrán program nejlépe splňující všechny stanovené požadavky.

2.1.1 Blender

Jedná se o multiplatformní open source aplikaci. Je zaměřená převážně na vytváření 3D modelů, tvorbu animací, rendering, ale díky implementaci game enginu je v něm možné vytvářet i interaktivní aplikace. Díky tomu, že je Blender open source, je uživateli umožněno stáhnout si kompletní zdrojové kódy. Ty může libovolně upravovat a kompilovat.

Blender je neustále ve vývoji a v horizontu 3-4 měsíců jsou představovány nové verze, nebo uváděny rozšiřující funkce a různá vylepšení. Tato rozšíření a vylepšení si každý uživatel také může vytvořit sám. Stačí, aby zvládal jazyk Python, ve kterém jsou napsány přídavné moduly. Uživatel si tak může sám napsat různé funkce, jež mu ušetří práci, nebo obohatí základ programu o další užitečné funkce. Celou řadu těchto rozšíření lze také najít na internetu.

V době vzniku této diplomové práce se vyvíjela verze 2.5, která přinesla uživateli vítanou změnu v podobě uživatelsky modifikovatelném grafickém rozhraní. Uživatelé si tak mohou jednoduše pomocí skriptů kompletně předělat vzhled a rozložení prvků nebo si přidat nové.

O velké oblíbenosti tohoto programu svědčí i obrovské množství tutoriálů, které neustále přibývají. Dalším příkladem jsou nesčetné stránky, zabývající se tvorbou v tomto programu a obsáhlá fóra. Tyto fóra obsahují velké množství užitečných informací pomáhající všem zkušeným, či začínajícím uživatelům při řešení problémů, se kterými se při své práci potýkají. [1]

[1] Blender [online]. 2005 [cit. 2010-11-05]. Dostupné z WWW: http://blender.org.

2.1.2 FreeCAD

Jde o 3D CAD program s pokročilou simulací pohybu. Slovo free v jeho názvu indikuje, že se jedná o open source aplikaci dostupnou pod licencemi GNU GPL a LGPL. Program uživateli dovoluje vytvářet a manipulovat s různými částmi, které jsou tvořeny jednoduchými 3D tělesy. Ty lze dále skládat do větších celků. Dále mohou na tato spojení působit různé síly, omezení a podobně. Jednotlivé části, jejich spojení a mezi nimi působící síly společně tvoří výslednou strukturu, která je dále předmětem zkoumání. Díky pokročilým pohybovým simulacím podává program přesné informace o chování objektu všem, kteří s ním pracují.

Aplikace běží pod operačními systémy Windows, Linux a Macintosh. Data vzniklá činností programu v rámci různých platforem jsou přenositelná a není nutné je nijak transformovat. Program dovoluje také velice dobrou rozšiřitelnost díky modulům psaným v Pythonu. Stejně jako v Blenderu i zde lze upravovat ovládací prvky. Bohužel jen velmi omezeně. Fóra s malým počtem příspěvků a nepříliš obsáhlá dokumentace mohou vypovídat o oblíbenosti tohoto programu. I přes to je vhodný pro každého, kdo se chce zdarma naučit v 3D CAD programu, před tím, než začne používat propracovanější programy. [2]

2.1.3 HeeksCAD

HeeksCAD patří mezi CAD aplikace. Podporuje import STEP a IGES souborů, kreslení konstrukční geometrie, vkládání primitivních těles, nebo vytváření nových pomocí vytažení náčrtu tělesa. Tělesa mohou být modifikována například pomocí booleanovských operací. Po provedení potřebných úprav se výsledek dá opět uložit jako IGES, STEP, nebo STL soubor. Pokud se jedná pouze o 2D geometrii, je možné ji přímo z programu bez dalších úprav vytisknout.

[2] FreeCAD [online]. 2011 [cit. 2010-11-05]. Dostupné z WWW: http://sourceforge.net/projects/free-cad/.

Program je možné spustit pod operačními systémy Windows a Linux. Stejně jako u předchozích programů i u tohoto si může uživatel doprogramovat různá rozšíření. Mezi jeho nevýhody patří malá dokumentace a přímo v programu není integrován žádný help. [3]

2.1.4 K-3D

Tento program je obdobně jako Blender zaměřen na 3D modelování a animaci. Mezi jeho hlavní rysy patří robustnost, dále pak objektově orientovaná pagin architektura, navržená pro potřeby profesionálních umělců. Od základu je navržen pro tvorbu kvalitních animací. K dosažení výsledné podoby animací jsou zde použity velmi kvalitní renderovací nástroje.

Program se dá používat pod systémy Linux, Windows, Solaris a Mac OS. K-3D obsahuje inovativní systém tutoriálů, které uživateli ukážou základní ovládací prvky a práci s nimi. Uživatel také sám může návody nahrávat a pak je sdílet na internetu. K rychlému seznámení s prostředím přispívá také obsáhlá a srozumitelná dokumentace. Program opět disponuje možností rozšíření pomocí skriptů, ale oproti ostatním se tyto skripty musí psát a spouštět v integrovaném editoru. [4]

2.1.5 Salome

Jedná se open-source program spustitelný na všech platformách. Jeho hlavní zaměření jsou příprava a následná zpracování při numerických simulacích. Salome se dá používat jako standardní aplikace pro generování CAD modelů, jejich přípravu pro numerické simulace a následné zpracování těchto výsledků. Opět je možné ho rozšířit pomocí nových komponent naprogramovaných v Pythonu. Poskytuje také uživatelsky přívětivé a efektivní rozhraní. Toto rozhraní pak pomáhá redukovat uživatelův čas, který by jinak potřeboval ke zvládnutí ovládání programu.

[3] heekscad - Free CAD based on Open CASCADE [online]. 2011 [cit. 2010-11-05]. Heekscad . Dostupné z WWW: http://code.google.com/p/heekscad/. [webová stránka]

[4] K-3D [online]. 2010 [cit. 2010-11-05]. K-3d.org. Dostupné z WWW: http://www.k-3d.org/wiki/Main_Page.

Do programu se dají importovat IGES, SEP a BREP soubory a následně provést jejich úprava. Jednoduché Mesh modely se dají v programu tvořit kompletně od začátku. Následně je potřeba provést nastavení modelu, jako například fyzikální vlastnosti částí objektu. Výsledné práce se dají libovolně ukládat a opětovně nahrávat pro další zpracování. [5]

2.2 Multikriteriální analýza

Z navrhovaných alternativ nebylo možné jen tak náhodně vybrat jeden z programů. Proto bylo potřeba vhodným způsobem porovnat vlastnosti a funkce jednotlivých kandidátů. Jako nejvhodnější se pro tento druh srovnání vlastností jevila multikriteriální analýza.

Jedná se o analýzu, zabývající se vyhodnocováním jednotlivých alternativ podle zvolených kritérií. Tato kritéria se vhodně zvolí a je jim přiřazena váha, která vyjadřuje důležitost jednotlivých kritérií vůči ostatním. Výstupem je pak tabulka s normalizovanými hodnotami. Součet těchto hodnot tvoří konečné skóre jednotlivých alternativ a nejvhodnější z nich se stává ta s nejvyšším bodovým ohodnocením.

2.3 Zvolená kritéria

Následující body popisují jednotlivá kritéria, která byla použita při vyhodnocování jednotlivých programů. Všechna hlediska pro posuzování byla volena podle požadavků pracovníků společnosti Lenam na funkcionalitu programu. Jedná se o licenci, pod kterou je program dostupný, kvalitu kódu, dostupnou dokumentaci, platformu, využitelné funkce, kterými program disponuje, kontakt s vývojáři a rozšiřitelnost o nové funkce.

2.3.1 K1-Licence

Jako nejlépe hodnocené byly zvoleny ty licence, které vývojáře a uživatele méně svazují. Jelikož všechny porovnávané subjekty patří mezi svobodný software, liší se známky jen minimálně.

[5] SALOME Platform [online]. 2005 [cit. 2010-11-05]. Dostupné z WWW: http://www.salome-platform.org/.

2.3.2 K2-Kvalita kódu

Důležitým aspektem při rozšiřování stávajících programů může být také jazyk, ve kterém se program a jeho moduly píší. Dále zda je v takto napsaných modulech k dispozici komentář, který by zjednodušil orientaci v kódu a jeho lepší pochopení.

2.3.3 K3-Dokumentace

Pod tímto kritériem je zahrnuta, jak napovídá název, dokumentace k programu. Dále zde bude zahrnuto i několik dalších velmi významných aspektů, které se nedají opomenout při vývoji programu. Jde například o oblíbenost mezi uživateli, z čehož plynou diskuze na různých fórech, kde si lze najít spoustu užitečných informací.

2.3.4 K4-Platforma

Významným aspektem z hlediska hodnocení je také přenositelnost programu mezi různými platformami. Zvoleny byly hlavní tři ovlivňující velkou měrou výsledek hodnocení. Jedná se konkrétně o tyto platformy:

- Windows
- Linux
- Mac OS

2.3.5 K5-Kontakt s vývojáři

Jde o velmi jednoduché, ale přitom významné kritérium. Kontakt s vývojáři programu může být pro výsledek celého projektu zásadní. Jde například o možnost zasahovat do částí programu, ke kterým by jinak nebylo možné se dostat vůbec nebo jen velice obtížně.

2.3.6 K6-Využitelné vlastnosti

Toto kritérium hodnotí funkce a vlastnosti programů, které jsou v nich implementovány a zároveň jsou požadovány zadavateli projektu. Mezi tyto funkce patří zejména:

- Ovládání zobrazení: změna pohledu na scénu: posun, otáčení, zoomování, zobrazení celé scény, předdefinované pohledy, uživatelsky definované pohledy, rozdělení plochy do více pohledů
- Manipulace s objekty: Posouvání ve směru os manuálně, relativně, absolutně, posouvání a otáčení ke globálnímu souřadnému systému (GSS), lokálnímu souřadnému systému (LSS) a normále obrazovky
- Kontrola průniků
- Geometrické informace: Měření vzdáleností, ploch, úhlů, průmět v definovaném směru, informace v GSS a LSS, uložení informací do entity
- Import modelu: Initial Graphic Exchange Specification (IGS), Nastran (NAS), Standard Library (STL)
- Export dat: NAS, obrázků, Virtual Reality Modeling Language (VRML)

2.3.7 K7-Rozšiřitelnost

Cílem je ohodnotit, jak snadno či nikoli se program rozšiřuje. Jedná se hlavně o to, co vše je potřeba udělat, aby uživatel mohl do programu přidat svou nově naprogramovanou funkci.

2.4 Vyhodnocení

Výstupem je soubor čtyř tabulek. První obsahuje názvy zvolených kritérií. Dále pak rozmezí bodového ohodnocení kritéria a určení optimální hodnoty podle toho, zda chceme její maximum či minimum. V posledním sloupci první tabulky jsou uvedeny váhy jednotlivých kritérií.

V druhé tabulce jsou přiděleny jednotlivým kritériím konkrétní hodnoty. Třetí obsahuje normalizované hodnoty těchto kritérií. V závěrečné tabulce je pak uveden celkový součet bodů a vyhodnocení výsledného pořadí.

Tabulka 1: Stanovení kritérií pro hodnocení programů

Kritérium		**Jednotky**	**Optimum**	**Váha-V(j)**
K1	Licence	Známka 0-1	Maximum	6
K2	Kvalita kódu	Známka 0-1	Maximum	8
K3	Dokumentace	Známka 0-1	Maximum	18
K4	Platforma	Známka 0-1	Maximum	8
K5	kontakt s vývojáři	0-Ne, 1-Ano	Maximum	25
K6	Využitelné funkce a vlastnosti	Známka 0-1	Maximum	15
K7	Rozšiřitelnost	Známka 0-1	Maximum	20

Tabulka 2: Hodnoty vybraných kritérií

Program	**K1**	**K2**	**K3**	**K4**	**K5**	**K6**	**K7**
Blender	0.70	1.00	0.95	1.00	1	0.59	1.00
FreeCAD	0.90	0.90	0.70	0.82	0	0.49	0.70
K-3D	0.75	0.80	1.00	0.95	0	0.39	0.60
Salome	0.85	0.50	0.85	0.25	0	0.72	0.40
Heekscad	1.00	0.50	0.15	0.50	0	0.47	0.50

2.4.1 Normalizace

Jelikož jednotlivá kritéria většinou nejsou poměřována stejnými jednotkami, je potřeba provést standardizaci matice na normalizovaný tvar. Maticí je v tomto případě předchozí tabulka, která je tvořena prvky S_{ij} kde i = 1, ..., I alternativ a j = 1,..., J kritérií. Vyhodnocovací matice je tedy:

$$S = \begin{matrix} S_{11} & ... & S_{1J} \\ ... & & \\ S_{I1} & ... & S_{IJ} \end{matrix} \quad (1)$$

$$e_{ij} = \frac{S_{ij} - min_i S_{ij}}{max_i Sij - min_i S_{ij}} \quad (2)$$

$$e_{ij} = \frac{max_i Sij - S_{ij}}{max_i S_{ij} - min_i S_{ij}} \quad (3)$$

Dále mohou nastat dva případy. V prvním případě znamená vyšší hodnocení kritéria lepší bodový zisk (2). Případem opačným je, když vyšší hodnocení znamená méně získaných bodů pro jednotlivé kritérium (3).[6]

Tabulka 3: Znormalizované hodnoty kritérií

Program	**K1**	**K2**	**K3**	**K4**	**K5**	**K6**	**K7**
Blender	0.00	1.00	0.94	1.00	1	0.61	1.00
FreeCAD	0.67	0.80	0.65	0.76	0	0.30	0.50
K-3D	0.17	0.60	1.00	0.93	0	0.00	0.33
Salome	0.50	0.00	0.82	0.00	0	1.00	0.00
Heekscad	1.00	0.00	0.00	0.33	0	0.24	0.17

[6]SMEP [online]. 2003 [cit. 2010-11-10]. Vícekriteriální rozhodování. Dostupné z WWW: http://etext.czu.cz/php/skripta/skriptum.php?titul_key=79. [webová stránka]

Tabulka 4: Vyhodnocení

Program	**K1**	**K2**	**K3**	**K4**	**K5**	**K6**	**K7**	**Body**	**Pořadí**
Blender	0.00	0.08	0.17	0.08	0.25	0.09	0.20	0.87	**1**
FreeCAD	0.04	0.06	0.12	0.06	0.00	0.05	0.10	0.43	**2**
K-3D	0.01	0.05	0.18	0.07	0.00	0.00	0.05	0.36	**3**
Salome	0.03	0.00	0.15	0.00	0.00	0.15	0.00	0.33	**4**
Heekscad	0.06	0.00	0.00	0.03	0.00	0.04	0.03	0.16	**5**

2.4.2 Závěr

Jako nejvhodnější program pro zvolený cíl vyšel po provedení multikriteriální analýzy Blender. Za první místo vděčí z velké části kritériu ,,kontakt s vývojáři", u kterého byla váha zvolena dvacet pět procent. Nicméně i bez této položky by se výsledné pořadí neměnilo, jelikož Blender patří z vybrané množiny mezi nejoblíbenější a to díky své flexibilitě.

3 Úprava uživatelského rozhraní

Po zvolení programu vhodného pro rozšiřující moduly, přišly na řadu úpravy uživatelského rozhraní. Nově naprogramované moduly se dají otevřít ve vestavěném textovém editoru a následně spustit. Tím se moduly zaregistrují a dále je lze spouštět pomocí příkazového řádku. Toto řešení je ale zdlouhavé a pro nezasvěceného uživatele těžko proveditelné.

Díky novým vlastnostem programu Blender se dají všechny tyto kroky implementovat pod jedno tlačítko. Podle požadavků pracovníků společnosti Lenam tak byla dotvořena potřebná menu. Tyto menu obsahují všechny potřebné ovládací prvky, jako jsou tlačítka, zaškrtávací pole a pole pro zadávání vstupních hodnot. Kromě těchto změn se upravilo také celkové rozmístění dalších prvků.

3.1 Vlastnosti Blenderu

Výše zmiňované úpravy by nebyly možné bez vlastností, kterými disponuje program Blender v nové verzi 2.5. Základem pro veškeré úpravy je programovací jazyk Python a jeho podpora. Pomocí tohoto jazyka lze přeprogramovat vzhled a rozmístění většiny ovládacích prvků a také vytvořit nové funkce a navázat je na ovládací prvky.

Dále je zde možnost upravit si vzhled i bez znalosti programování, ale samozřejmě pouze do určité míry. To je možné pomocí takzvaného window manageru. Jde o jednoduchý systém, díky němuž si každý uživatel přizpůsobí vzhled a rozmístění oken podle svého uvážení. Jedná se o obdobu kliknutí pravým tlačítkem myši a příkazu split area v nižších verzích programu. Zde jde o nenápadný trojúhelníček v každém rohu okna. Pokud na něj uživatel najede myší, kurzor se změní na křížek. Pomocí křížku lze okna rozdělovat a znovu sjednocovat. V levém dolním rohu nově vzniknuvšího okna je pak tlačítko s nabídkou. Pomocí této nabídky je možné si zvolit, jaké okno se má zobrazovat. Novinkou v této verzi je možnost oddělení do zvláštního okna, které se dá snadno provést při podržení tlačítka SHIFT. Následující kapitoly popisují již zmiňovanou podrobnější úpravu uživatelského rozhraní pomocí programovaní.

3.2 Skripty pro úpravu UI

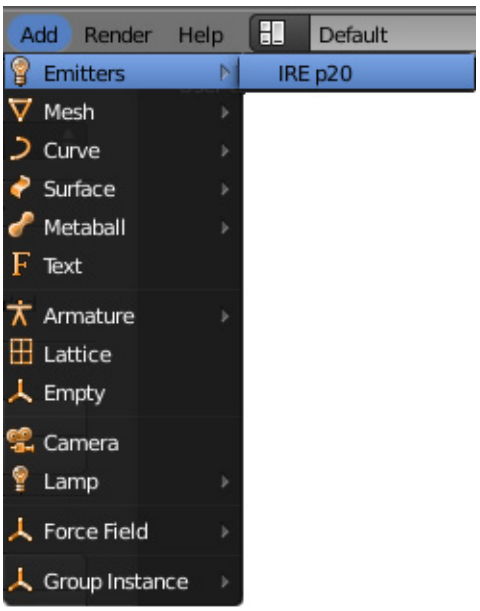
Pokud se uživatel nespokojí pouze s úpravami rozhraní, které mu dovoluje window manager, pak má možnost pustit se do úprav skriptů, které jsou dostupné ve složce ui. Tato složka obsahuje kolem čtyř desítek skriptů. Tyto skripty odpovídají jednotlivým menu Blenderu, jako jsou například Textový editor, Node editor, Editor video sekvencí, panel tvořící základní menu pro ukládání souborů, nebo vkládání nových objektů. Názvy souborů jsou odvozeny od toho, jaké menu se pomocí nich dá upravit. Nejvíce upravované byly dva s názvy space_info.py který jsem v rámci diplomové práce rozšířil o možnost vkládat objekty zářičů a space_view3d_toolbar.py jež poskytuje uživateli rozhraní pro ovládání nově vytvořených funkcí. Tyto dva byly pouze nejpoužívanější, nikoliv však jediné, které prošly úpravami. Modifikací prošel také skript properties_object.py, jež slouží k vizualizaci barevné škály reprezentující jednotlivé teplotní rozsahy dosažené na galvanu.

3.3 Dosažené výsledky

Pomocí prvního menu, tvořeného v rámci této práce, se do scény přidávají objekty reprezentující zářiče. Jako nevhodnější bylo vybráno menu Add, které slouží pro přidávání primitiv jako je například koule, kvádr, nebo objekty kamer a lamp do scény. **Nabídka na následujícím obrázku byla vytvořena vložením několika řádek kódu do skriptu, který slouží k vložení objektu zářiče do scény. Kód, pomocí kterého docílíme vytvoření nové nabídky, je následující**:

layout.menu(def menu_func(self, context):
self.layout.operator(addIREp20.bl_idname, text="IRE P20")

V druhém řádku jsou uvedeny dva parametry. První z nich je jméno funkce, která se má spustit. Druhým je text, který se bude u nabídky zobrazovat. Dále je možné uvést třetí nepovinný parametr. Tento parametr vloží zvolenou ikonku k textu v nabídce. Aby se zobrazil požadovaný výsledek, je třeba ještě učinit poslední krok, kterým je zaregistrování námi vytvořené třídy.



Obrázek 1: Výsledná nová nabídka pro přidání objektu zářiče do scény

Druhé menu je o poznání komplikovanější. Nejedná se pouze o přidání jednoho prvku, ale hned několika. Pro větší přehlednost bylo potřeba tyto prvky rozčlenit podle funkcionality do různých částí.

Na následujícím obrázku (2) je panel, skládající se z několika skupin. První má pouze informativní účel a ulehčuje uživateli práci. Informuje, zda spuštěná metoda proběhla bez chyb, či nikoliv. Pod textem Info se objevují hlášení, předávané metodami. Ty jsou trojího typu. První je informační, začínající textem INF: a následuje sdělení metody. Většinou se jedná o hlášení sdělující uživateli, že metoda proběhla bez obtíží, nebo provedla ten či onen úkon. Druhým typem jsou varování, uvozené textem WAR: a opět následuje text definující upozornění. Jde například o situace, kdy jsou dvě metody na sobě závislé, a vykonání druhé je podmíněno spuštěním první. Pokud se tedy uživatel pokusí spustit druhou metodu, program mu v panelu zobrazí varovný text a nasměruje ho k napravení chyby. Posledním typem je chybové hlášení začínající ERR: a opět následuje text objasňující nastalou chybu. Ve většině případů jde ale o nevykonání metody. Nastalá chyba v metodě může být způsobena například špatným programováním a neodchycením výjimky, která se následně musí řešit zpětnou vazbou na programátora.

Obrázek 2: Menu funkcí

Jako další se na panelu nachází ovládací prvek, sloužící k definování objektu galvana. Pokud chce uživatel používat další funkce dostupné na panelu, musí nejprve provést zmiňované definování galvana. Ta je základem, jelikož tím uživatel řekne programu, který z objektů ve scéně je galvano. Galvano je tedy objekt, na kterém se budou vizualizovat stopy zářičů. Uživatel to provede ve dvou krocích a to tak, že si označí objekt, reprezentující galvano. Poté klikne na tlačítko s názvem Define Galvano.Tlačítko pak změní svůj text na název objektu. To je vidět i na obrázku a uživatel je tak hned informován, jaký objekt zvolil. Na začátku projektu zde byly ještě některé další ovládací prvky, které se postupem času zredukovaly na minimum.

Možnost testovat kolize mezi zářiči je dostupná v následujícím úseku. Jedná se o jednu z hlavních částí, programované v rámci této práce. Tato funkce je založena na algoritmu oddělených os (SAT, separating axis theorem) pro testování kolizí mezi dvěma orientovanými boxy. Detekce kolizí se pak provádí pouze pro objekty reprezentující infračervené zářiče (IRE, infrared emitter). Názvy všech zářičů vložených do scény mají na svém začátku zkratku IRE, čímž se skriptu identifikují. Výsledkem činnosti skriptu jsou barevně odlišené kolidující a volné zářiče.

Poslední skupinou jsou radiační metody. Ty slouží k vizualizaci stop zářičů na modelu galvana. Jedná se o funkci GEOM, která je tou jednodušší z této dvojice a slouží pouze k vizualizaci ozáření pomocí jedné barvy. Uživatel tak zjistí pouze oblast, kterou je schopen zářič efektivně ovlivnit. Druhá metoda s názvem FUNC je o poznání složitější a vizualizuje i jednotlivé teplotní rozsahy, jichž se dá na modelu galvana dosáhnout.

3.3.1 Příklad tvorby informačního panelu

Následující část kódu je k nalezení ve skriptu se jménem spece_view3d_toolbar.py. Tento kód generuje již zmíněný informační panel. Tvorba panelu se dá zjednodušeně popsat následujícím způsobem. Uživatel si vytvoří novou třídu s odpovídajícím názvem. Blender si ji automaticky zaregistruje. Následují řádky s určením kontextu a popisu, který se bude u nabídky zobrazovat. Dále se zde nachází několik neměnných řádek kódu.

```
class VIEW3D_PT_tools_informations(View3DPanel, bpy.types.Panel):
    bl_context = 'objectmode'
    bl_label = 'Info'
    def draw(self, context):
        layout = self.layout
        col = layout.column(align=True)
        col.label(text=Fn.infoText[len(Fn.infoText)-1])
```

Poslední řádek z ukázky je ale o poznání zajímavější. Obsahuje text, který se bude vypisovat. Na panel lze tedy vypsat jakýkoliv řetězec, stačí jej vepsat do uvozovek za identifikátor text. V tomto konkrétním případě se vypisuje poslední prvek pole, které obsahuje zprávy funkcí, jako GEOM, FUNC apod. Dalším hojně používaným prvkem je tlačítko, to lze vytvořit následujícím kódem:

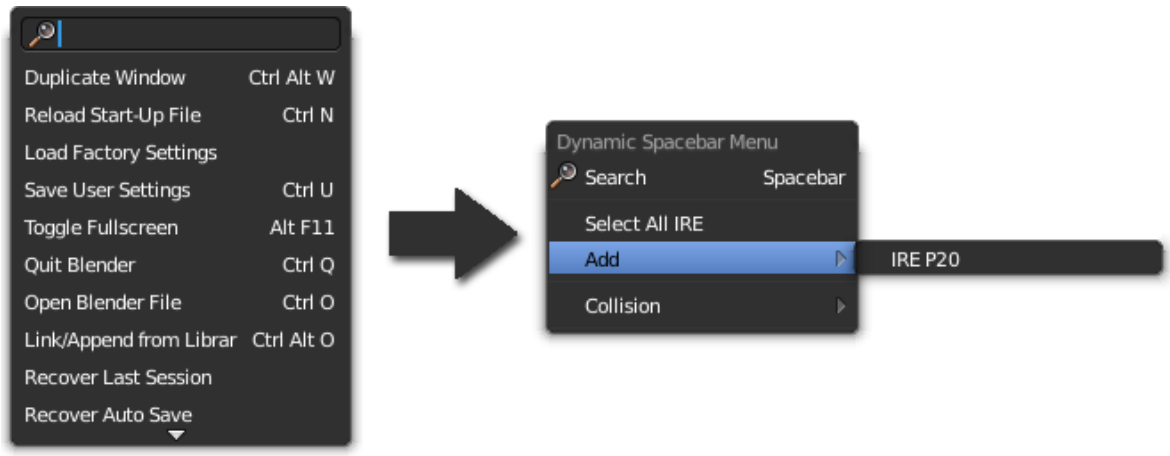
```
col.operator(checkCollision.bl_idname, text='Check Collision')
```

Funkce tvořící tlačítko přebírá dva parametry. Prvním je jméno funkce, pod kterým je zaregistrována a druhým je popisek tlačítka. V dokumentaci lze najít i další nepovinné parametry. Jedná se například o možnost přidat ikonu na tlačítko, či zobrazení informačního textu, pokud uživatel najede na tlačítko a po nějakou dobu na něm ponechá kurzor.

Tlačítko a textové pole jsou pouze dva základní stavební prvky, patřící mezi dva nejvíce využívané v této práci. Pokud by to uživateli nestačilo a chtěl by použít některé další, jako jsou checkBox, pole pro zadávání textu, číselné hodnoty, nebo výběrové pole s pevně danými hodnotami, nebude to v Blenderu žádný problém. Stačí si pouze dohledat v dokumentaci, jak se patřičný prvek používá.

3.3.2 Další úpravy a výsledná podoba programu

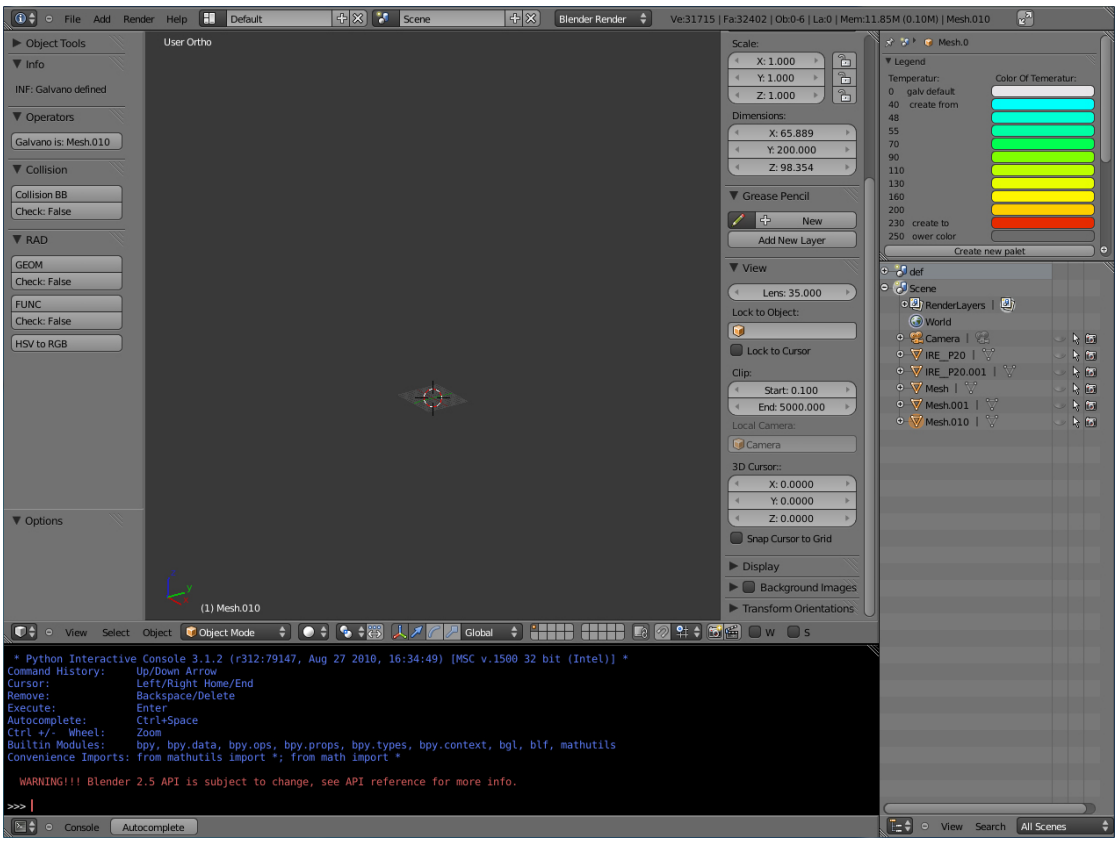
Úpravami prošlo také dynamické menu, které se vytvoří na pozici kurzoru myši při stisknutí mezerníku. Obrázek (3) ukazuje klasické menu, dostupné ve všech verzích Blenderu a menu upravené. To obsahuje nejčastěji využívané položky při práci se zářiči. Jedná se o selekci všech objektů typu zářič, přidání zářiče do scény a detekce kolizí. Podle potřeb uživatelů se pak dají přidávat či odebírat různé funkce pomocí jednoduchých zásahů do skriptu, pomocí kterého se toto menu upravuje.



Obrázek 3: Dynamické menu

Jak již bylo napsáno výše, nebylo potřeba pouze vytvořit nové panely s ovládacími prvky, ale také kompletně upravit jejich rozmístění v okně programu. To je celé rozděleno na šest sekcí. První z nich je horizontální, ve kterém uživatel najde všechny základní funkce dostupné v jakýchkoliv jiných programech. Mezi tyto funkce paří například tvorba nového projektu, uložení, načtení, vkládání objektů a další funkce specifické pro Blender, mezi které patří přepínání mezi scénami, nebo různými uskupeními ovládacích prvků. Nejvíce vpravo jsou nad sebou umístěny dva panely. Vrchní zobrazuje barevnou škálu. Podmínkou zobrazení škály je definice galvana, pokud není definované, barevná škál se nezobrazí. Spodní z pravé dvojice panelů zobrazuje strom objektů dostupných ve scéně. Dále je možné pomocí klávesy N zobrazit či skrýt panel obsahující některé další hojně využívané funkce. Mezi ně patří například transformace, nebo změna měřítka vybraného objektu. Uprostřed okna programu se nachází pracovní plocha, na které je možné si prohlédnout výsledky uživatelovi práce. Předposlední je pak panel z obrázku (2), nacházející se na obrázku vlevo. Na něm najdeme největší koncentraci nově vytvořených ovládacích prvků. Poslední panel se nachází pod levým panelem a pracovní plochou. Zde je umístěna konzole. Uživatel samozřejmě může obsah jednotlivých oken libovolně měnit podle svých potřeb a práce, kterou právě provádí.

Celé rozložení všech panelů a ovládacích prvků, jak je popsáno výše, bylo vytvořeno podle požadavků pracovníků společnosti Lenam. Jak toto výsledné okno vypadá, ilustruje následující obrázek (4).



Obrázek 4: Okno programu Blender

4 Vkládání zářičů

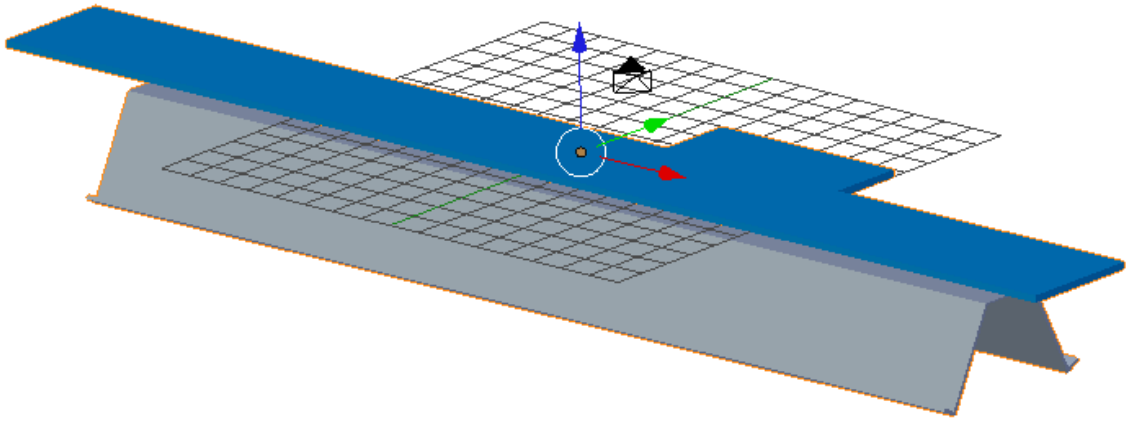
Na začátek zde uvedu trochu teorie k pochopení nutnosti detekovat kolize mezi zářiči. Ty budou ve scéně reprezentovat jejich zjednodušené modely, které mají hned několik funkcí. Uživatel samozřejmě potřebuje vidět, jak zářič vypadá, kam zasahuje, jestli se některé jeho části nepřekrývají s jiným a proto bylo potřeba model vytvořit co nejpřesněji.

Velká přesnost zde hraje podstatnou roli. Jedná se především o to, že na skutečné výrobní lince rozmisťuje zářiče kolem galvana robot, kterému se zadají quaterniony určující, kam je zářiče třeba umístit. Poté přijde na řadu mechanik, jenž pomocí speciálních držáků upevní zářič na místo určené robotem. Pokud by v programu zářiče jen nepatrně kolidovaly, což je celkem pravděpodobné díky jejich velkému množství ve scéně, robot by mohl o již upevněné zářiče zavadit a poškodit je. To je vzhledem k jejich ceně nežádoucí.

Data k vytvoření zářiče byla dodána pracovníky společnosti Lenam. Jednalo se o model importovaný do programu Blender. Model zářiče byl značně komplikovaný a jeho síť se skládala z velkého počtu polygonů. Model zářiče se tedy nedal pro potřebné záměry téměř použít. Síť modelu se musela značně zjednodušit a upravit ji tak, aby bylo možné na jednotlivé části objektu nasadit co možná nejpřesněji různě rotované kvádry, které budou v pozdějších částech sloužit k detekci kolizí.

Nakonec vznikla poměrně jednoduchá síť, tvořená šesti částmi, na které se dají s velkou přesností nasadit takzvané bounding boxy. Jak je celý objekt členěný a jaký je konečný výsledek, je vidět na obrázcích (5) a (6). I když je zářič značně zjednodušený, je i tak plně dostačující pro potřebné účely a díky jednoduchosti nijak nezpomaluje program i při vložení velkého počtu zářičů.

Jednotlivé části pak slouží k vytvoření datové struktury, použité pro výpočty kolizí. Celý objekt hraje velkou roli při provádění funkcí GEOM a FUNC. Pomocí jeho pozice a rotace se určují různé vektory potřebné k chodu funkcí GEOM a FUNC. Jedná se například o výpočet průsečíku paprsků směřujících od zářiče ke galvanu. Tyto výpočty zjednodušují některé vestavěné moduly, díky nimž není nutné psát například funkce pro práci s vektory a různé matematické funkce.



Obrázek 5: Model zářiče vkládaný do scény

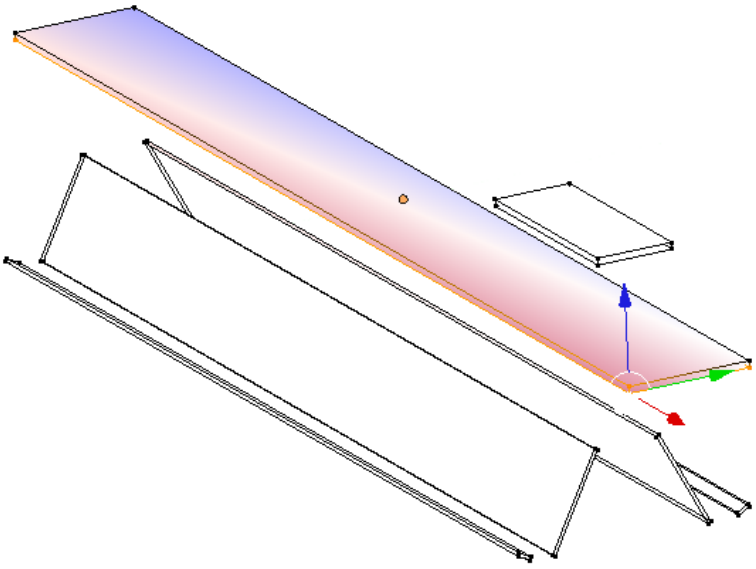
4.1 Použité moduly

Python poskytuje možnost využívat moduly. Jedná se o samostatné soubory obsahující zdrojové kódy Pythonu, ale také často kompilované objektové soubory jazyků C nebo C++. Moduly se dají k našemu kódu připojit pomocí příkazu import a pak již můžeme využívat všech metod a atributů, které jsou v něm obsaženy. Pokud bychom chtěli zjistit, jaké metody a atributy máme k dispozici, můžeme s výhodou použít funkci dir(). Jako parametr příkazu dir se pak uvádí jméno modulu který nás zajímá.

Mezi nejvíce využívané moduly patří například math, nebo mathutils. Ty obsahují hlavně matematické funkce a funkce pro práci s vektory. K podrobnějšímu popisu jsem vybral dva moduly. První z nich má název IRELab_Fn a byl vytvořen speciálně pro tuto práci. Druhý je modul Pickle, který výrazně pomáhá programátorům s urychlením jejich práci.

4.1.1 IRELab_fn

Jedná se o modul, který vznikl speciálně pro využití v nově psaných skriptech. Jsou zde vytvořeny funkce, které se ve skriptech často opakovaly, a tudíž by bylo zbytečné je opakovaně psát. S výhodou zde byla využita možnost vytvořit si vlastní modul, obsahující všechny potřebné a často se opakující části kódu.



Obrázek 6: Model zářiče rozčleněný na jednotlivé kvádry

Hlavní funkcí modulu je práce se zprávami, kterými nově napsané skripty informují uživatele o své činnosti. Modul se stará o jejich shromažďování a předávání na informační panel. Dále obsahuje funkci, jejímž úkolem je tvoření nových materiálů. Ta se zavolá a předají se jí čtyři parametry. Tyto parametry informujícími Blender o difúzní barvě, její intenzitě, a spekulární barvě a její intenzitě. Často se využívá pro tvoření barevné škály tepelného záření na galvanu, nebo barev jednotlivých zářičů.

4.1.2 Pickle

Jednou z velkých výhod jazyka Python je možnost ukládat objekty do souborů. Pomocí Pythonu můžeme tedy zapsat do souboru libovolné datové struktury a poté je opětovně načíst a vytvořit a stačí mu k tomu pouze několik příkazů. Jde o velice užitečnou funkci, která nejednomu programátorovi ušetří spoustu zbytečného kódu, který by nedělal nic jiného, než četl data ze souboru a upravoval je tak, aby odpovídaly uloženým strukturám.

Python poskytuje tuto možnost uložení struktur a jejich opětovného načtení pomocí modulu pickle. Uložení do souboru se provádí pomocí příkazu dump. Uložit se takto dá jakákoliv proměnná. Ta může být pouze jednoduchý datový typ, nebo i seznam slovníků obsahující instance tříd definovaných uživatelem. Příklad použití je následující:

```
data = a: 'ahoj'
Import pickle
soubor=open(data.txt,'wb')
pickle.dump(data, soubor)
soubor.close()
```

Libovolná data, která byla předtím uložena v proměnné, budou opět ze souboru načtena a znovu do této proměnné uložena příkazem load. S takto načtenou proměnnou se v programu dá opět pracovat jako při jejím vytvoření, tedy používat na ní stejné funkce a procedury bez jakýchkoliv dodatečných úprav. Postup, jak načíst obsah proměnné, uložené předešlým kódem, je následující:

```
Import pickle
soubor=open(data.txt,'rb')
data = pickle.load(soubor)
soubor.close()
```

Jednou z podmínek, kterou musíme při používání modulu pickle splnit, je otevření souboru jako binárního. Pokud bychom to nedodrželi, nebylo by možné data ukládat a při provádění kódu by nastala chyba. [7]

[7] HARMS, Daryl; MCDONALD, Kenneth. Začínáme programovat v jazyce Python. Brno : Computer Press, 2003. Nakládání objektů do souboru, s. 456. [část knihy]

4.2 Virtuální model zářiče

Jak již bylo řečeno v předchozích kapitolách, kolem modelu galvana se budou rozmisťovat zářiče, které ho budou ozařovat a zahřívat na požadovanou teplotu. V programu Blender neexistuje žádné primitivum, jenž by alespoň vzdáleně mohlo reprezentovat zářič. Proto bylo potřeba vytvořit objekt, který by ve scéně zářič reprezentoval. Model se musel vytvořit co nejpřesněji, protože se ve scéně používá k detekci kolizí mezi zářiči. Program Blender naštěstí obsahuje funkce, dovolující programátorovi vytvořit libovolně vypadající nový objekt.

4.2.1 Tvorba nového objektu

Základem pro tvorbu nového objektu je nadefinování pole vertexů a následně jejich propojení do plošek. Vertex neboli bod se skládá ze tří složek, kterými jsou pozice na ose x, y a z. Pokud si uživatel definuje svou množinu bodů, je dále potřeba říct programu o bodech, jež jsou spolu v prostoru spojeny hranami, nebo tvoří plošku, což je spojení tří až čtyř bodů. Následující ukázka kódu popisuje, jak se vytvoří 3 body a následně se spojí v plošku.

```
verts = []
verts.extend([ 18.000, 1.200, 0.200])
verts.extend([-18.000, 1.200, 0.200])
verts.extend([-18.000,-2.500, 0.200])
faces = []
faces.extend([ 0, 1, 2])
```

Pokud bychom chtěli tuto plošku vložit do scény, musí se provést ještě několik dalších kroků. Nejdříve je nutné vytvořit mesh objekt a přiřadit mu jméno. Dále se mesh objektu přidají připravená data. Těmi jsou vertexy a jejich spojení v polygon. Takto vytvořený mesh se přiřadí konkrétnímu objektu, který je opět nutné pojmenovat. Nakonec se výsledný objekt vloží do scény. Následující řádky provádějí všechny úkony popsané v tomto odstavci.

```
scene = bpy.data.scenes['def']
mesh = bpy.data.meshes.new("P20_def")
obj = bpy.data.objects.new("P20_def", mesh)
mesh.vertices.add(len(verts) // 3)
mesh.faces.add(len(faces) // 4)
mesh.vertices.foreach_set("co", verts)
mesh.faces.foreach_set("vertices_raw", faces)
mesh.update()
scene.objects.link(obj)
```

Výsledkem ve scéně je pak objekt trojúhelníku s výchozí šedou barvou, umístěný na souřadnicích 0,0,0. V dokumentaci je možné najít další metody, pomocí kterých se dá objektu například přiřadit barva, pokud bychom se nespokojili pouze s výchozí šedivou, nebo objektu můžeme nastavit konkrétní umístění ve scéně a podobně.

4.2.2 Zářič

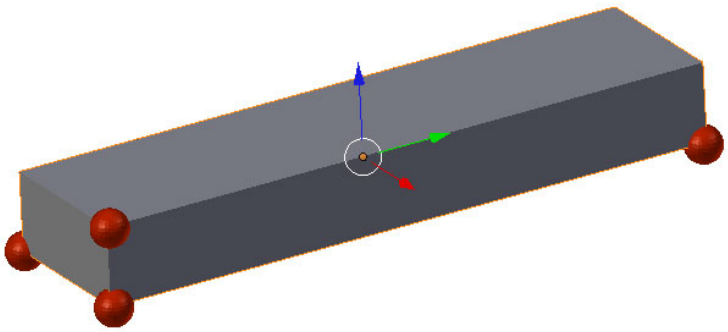
Při tvorbě modelu zářiče, použitého ve scéně pro výpočty, bylo zapotřebí provést další úpravy. Samotný zářič je o něco složitější než pouze jednoduchý objekt, vytvořený pro ilustraci v předešlé kapitole. Objekt zářiče je tvořen okolo padesáti body a o něco méně ploškami. Zářič bylo potřeba nějak odlišit, proto byla pro objekt zářiče vytvořena specifická barva, podle níž se dají jednotlivé typy zářičů rozlišit. V dalších verzích programu se totiž předpokládá použití více druhů zářičů. Dalším důvodem využití barev jsou kolize mezi zářiči. Pokud zářiče kolidují, změní se jejich barva. Kolidujícím se přiřadí speciální barva, která je pro jednoznačnost zvolena jako červená, ostatním zůstane barva původní. Pokud již zářič nebude v kolizi, přiřadí se mu opět jeho výchozí barva.

Další data nesoucí zářičem, jsou určena pro zmiňované kolize. Jedná se o souhrn informací, které potřebuje metoda SAT, používaná při detekci kolizí ke své funkčnosti. Tato data jsou uložena do souboru pomocí knihovny pickle. Při kontrole kolizí se vyberou patřičná data podle typu zářiče a použijí se jako vstupní hodnoty při provádění skriptu.

4.2.3 Členění kvádru a data pro kolize

Tato část je zaměřena na ozřejmění struktury objektu, používané při detekci kolizí mezi jednotlivými kvádry. Samotné vysvětlení, jak ona metoda SAT funguje, bude následovat v kapitole o kolizních metodách. Datová struktura, vytvořená při vložení prvního zářiče do scény, obsahuje informace o rozměrech kvádru, umístění středu a souřadnice čtyř jeho bodů. Pokud se tedy do scény vloží první exemplář zářiče, ve složce data se vytvoří soubor, nesoucí název zářiče, pro který byl vytvořen a obsahující potřebná data.

Do souboru se ukládají následující data. Souřadnice čtyř bodů kvádru. Jde o vrcholy, z nichž jeden musí sousedit s ostatními třemi. Jaké vrcholy to například mohou být, je vyznačeno na následujícím obrázku.



Obrázek 7: Model ilustrující rozmístění bodů vkládaných jako data pro kolize

Body používané pro detekce kolizí jsou vyznačeny na obrázku červeně. Dále je k detekci potřeba souřadnice středu kvádru. Poslední složkou jsou poloviny délek hran kvádru, určující jeho rozměry. Všechny souřadnice, potřebné pro funkci metody, se zadávají v lokálním souřadném systému kvádru. Proto je potřeba provést jejich přepočet do globálního souřadného systému. To se provede vynásobením transformační matice vektorem tvořeným souřadnicemi bodu.

Metoda SAT a také většina dalších kolizních metod pracuje s kvádry. Proto bylo potřeba složitější objekt zářiče rozčlenit na jednotlivé kvádry. Jakým způsobem se toto rozčlenění provedlo, je vidět na obrázku (6). Pokud by se jednalo o nějaký složitější objekt,

provedlo by se členění podobně. Použilo by se více kvádrů, popřípadě by se dala použít některá další obalová tělesa, lépe aproximující objekt. V tomto konkrétním případě, jak je zřejmé, bylo nejvýhodnější použít pouze kvádrů jako obalových těles.

Samotná datová struktura, o které je řeč a která nese všechna potřebná data, má následující podobu. Jedná se o třídu, které se předávají při vytvoření následující parametry. Jméno objektu, pro který je vytvořena, jeho transformační matice, počáteční body, dále pak střed a poloviny délek hran. Všechna tato data musí být uvedena pro každý jednotlivý kvádr. Tudíž, skládá-li se celé těleso z šesti kvádrů, musejí se do výsledného souboru zapsat datové struktury pro všechny tyto kvádry. Kód, který se stará o vytvoření této struktury, je následující:

```
class strucOfEmitter:
    def __init__(self,name=None, mat=None, bp=None, ce=None, ha=None):
        self.name = name
        self.mat = mat
        self.bp = bp
        self.ce = ce
        self.ha = ha
```

Tento souhrn informací se týká pouze části, která se tvoří při vkládání nových objektů zářičů do scény. Všechny podrobnější informace, jak se tato data využívají, k čemu slouží jejich jednotlivé složky a jak se s nimi počítá při detekci kolizí, budou popsány v následující kapitole, která se bude týkat obalových těles a detekcí kolizí mezi nimi.

5 Kolizní metody a jejich využití v programu

V předchozí kapitole byla poprvé řeč o kolizích a o tom, proč je potřeba je detekovat. Jedná se o hojně používané funkce, hlavně pokud jde o herní průmysl. Těžko bychom si dnes dokázali představit hru, ve které by neprobíhaly testy kolizí. Přestože tyto testy probíhají v menší či větší míře téměř v každé hře. Jedná se o celkem složité operace, hlavně pokud jde o případy detekce v 3D prostoru a mezi složitými objekty. Tyto detekce se často provádějí v reálném čase. Aby toho bylo možno docílit, musejí se funkce, starající se o detekce, optimalizovat či zjednodušovat objekty, mezi kterými se detekce provádějí.

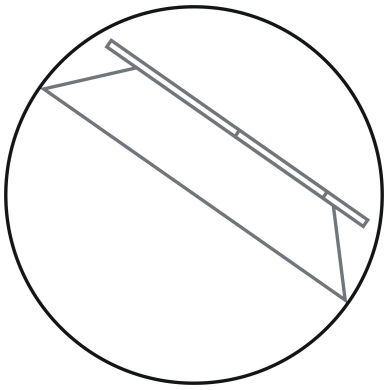
Detekce kolizí v tomto programu se nebudou provádět v reálném čase. Přesto je potřeba provést určité optimalizace, jelikož se ve scéně bude nacházet velké množství objektů, které bude nutné prověřit na kolize. Jednou ze základních technik při detekci kolizí je použití zjednodušení objektů za pomocí obalových těles.

Obalové těleso tvoří jakousi schránku, uvnitř které je celý objekt schován a výpočet kolize se pak provádí pouze pro tuto schránku. Existuje několik druhů obalových těles a také požadavky, které musí splňovat. Prvním z nich je, aby těleso co nejtěsněji obklopovalo objekt. Čím těsnější obal bude, tím více klesá pravděpodobnost, že při detekci kolize spolu kolidují obaly ale tělesa nikoliv. Další požadavek vyplývá ze samotné podstaty obalového tělesa. Tím je rychlost testu mezi dvěma obalovými tělesy,která by měla být samozřejmě co největší. Dále je potřeba vytvořit datovou strukturu, reprezentující obalové těleso. Struktura musí být naopak co nejmenší, aby nezabírala místo v paměti. [8]

5.1 Koule

Koule je nejjednodušší obalové těleso. Její jednoduchost je ale vykoupena netěsností pro většinu objektů. Velkou výhodou je velmi rychlý test kolize dvou koulí a také skutečnost, že je invariantní vůči rotaci. Kouli reprezentují pouze dva parametry a těmi jsou její střed a poloměr.

[8] DVOŘÁKOVÁ, Marta. Kolizní systémy. Brno, 2007. 26 s. Bakalářská práce. Masarykova univerzita. [akademická práce]

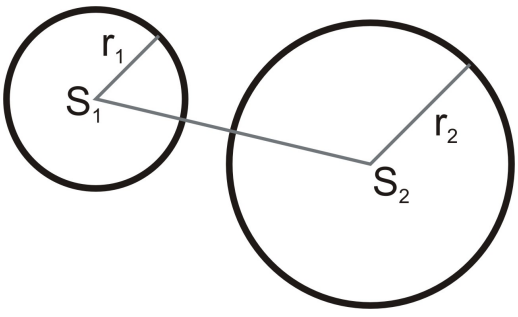


Obrázek 8: Obalová koule

5.1.1 Kolize koule-koule

Tento algoritmus je velmi jednoduchý. Jelikož se jedná o opravdu elementární a rychle vyhodnocený test, používá se v prvních úrovních testování. Pokud dojde k nalezení kolize, pak se může přistoupit k další úrovni testování. To může například znamenat i použití jiných obalových těles než koulí, aby bylo dosaženo přesnějšího vyhodnocení kolizí. Při testu máme dvě koule reprezentované svými středy S1, S2 a poloměry R1 a R2. Pokud koule nekolidují, je vzdálenost jejich středů větší než součet jejich poloměrů, musí tedy splnit podmínku (5).[9]

$$|s_1 - s_2| > r_1 + r_2 \tag{4}$$

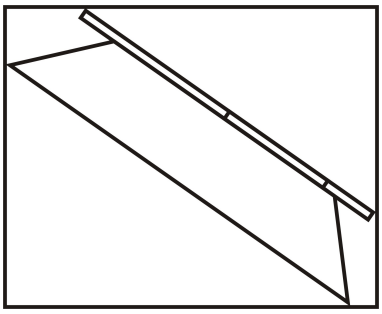


Obrázek 9: Test kolize koulí

[9]DVOŘÁKOVÁ, Marta. Kolizní systémy. Brno, 2007. 26 s. Bakalářská práce. Masarykova univerzita. [akademická práce]

5.2 Osově orientovaný box

Toto těleso bývá často používáno u objektů, které jsou osově zarovnané, dále se s nimi nepohybuje a nerotuje. Typickým příkladem objektu pro použití osově orientovaného boxu (AABB, axis-aligned bounding boxu) je například dům. Důvodem použití je fakt, že takový objekt obepne velice těsně. Nespornou výhodou je také rychlost detekce kolize dvou AABB, protože v nejhorším případě stačí pro zjištění kolize šest porovnání.



Obrázek 10: Osově orientovaný box (AABB)

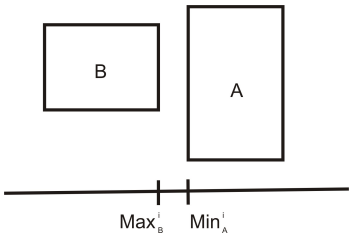
5.2.1 Kolize dvou osově orientovaných boxů

Dva osově orientované kvádry A, B určené svými maximálními a minimálními vrcholy MaxA, MaxB, MinA, MinB spolu nekolidují, pokud platí:

$$\exists i(1 \leq i \leq 3)((Min_i^B > Max_i^A)||(Min_i^A > Max_i^B)) \tag{5}$$

Kolize lze vyloučit, pokud se nepřekrývají intervaly, tedy průměty kvádrů alespoň na jedné z os. [10]

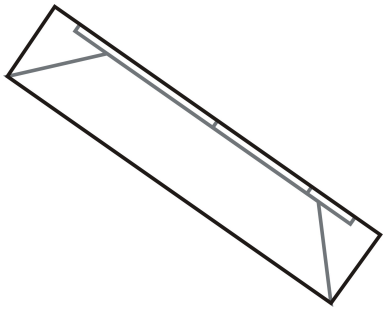
[10]DVOŘÁKOVÁ, Marta. Kolizní systémy. Brno, 2007. 26 s. Bakalářská práce. Masarykova univerzita. [akademická práce]



Obrázek 11: Test kolize dvou AABB

5.2.2 Orientovaný kvádr

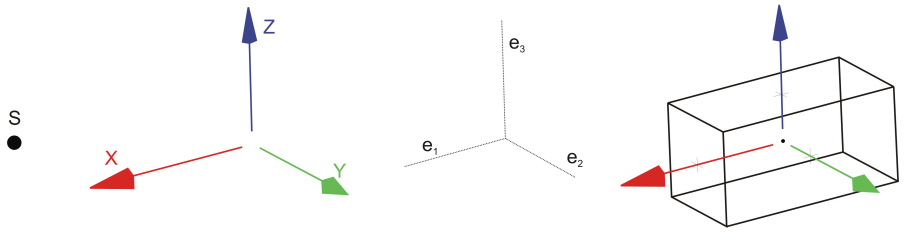
Název napovídá, že jde o kvádr, který je v prostoru natočený podle objektu, který obepíná. Proto se dá dosáhnout větší těsnosti než u předchozího, a je pro tuto vlastnost také často využíván. Předností je také rychlost zjištění, zda spolu dva orientované boxy (OBB, orineted boundingbox) kolidují či nikoliv. V nejhorším případě je potřeba provést patnáct jednoduchých testů, které jsou popsány v tabulce (5).



Obrázek 12: Orientovaný box (OBB)

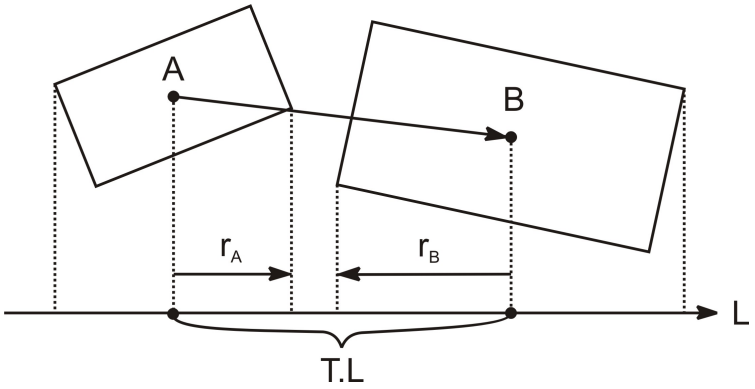
5.2.3 Kolize dvou orientovaných boxů

Algoritmus, který je použit k detekci kolizí mezi dvěma OBB se nazývá SAT. Aby mohla být detekce proveditelná, je potřeba znát některé údaje o kvádru. Prvním z nich je střed kvádru, následují vektory, které reprezentují jednotlivé osy kvádru v jeho lokálním souřadném systému. Dále potřebujeme znát rozsahy kvádru na jednotlivých osách. Jedná se o vzdálenost středu kvádru od středu stěny na všech osách. Pro názornost jsou na obrázku (13) ukázány všechny složky, které tvoří vstupní parametry algoritmu.



Obrázek 13: Reprezentace OBB ve 3D prostoru

Díky orientaci boxů není jednoduché tento test provést. Jde o spoustu aritmetických operací a tak je mnohonásobně pomalejší, než například test kolize dvou koulí. Zmíněný algoritmus SAT, ve volném překladu metoda oddělených os, porovnává vzdálenosti na určité ose $\overrightarrow{L}$, na kterou provede projekci středů a vrcholů a porovnává vzdálenosti. Tuto myšlenku ilustruje obrázek (14).



Obrázek 14: 2D algoritmus SAT pro dvě obálky OBB

Obecná podmínka, která platí v případě, že jsou dvě obálky OBB odděleny, je definována jako (6). Metoda je založena na předpokladu, že na počátku ke kolizi došlo a provádí porovnání dle podmínky (6), aby indikovala bezkolizní výsledek. Nejvíce porovnání se tedy provede, pokud ke kolizi nedošlo. Těchto porovnání bude maximálně 15 a tento počet závisí na počtu os, kterých je třeba v testu. Tabulka (5) je přehledem všech potenciálních porovnání.[11]

[11]KORBA, Lukáš. Simulace řízení vozidla. Praha, 2008. 63 s. Diplomová práce. České vysoké učení technické v Praze.

$$|\overrightarrow{T} \cdot \overrightarrow{L}| > r_A + r_B \tag{6}$$

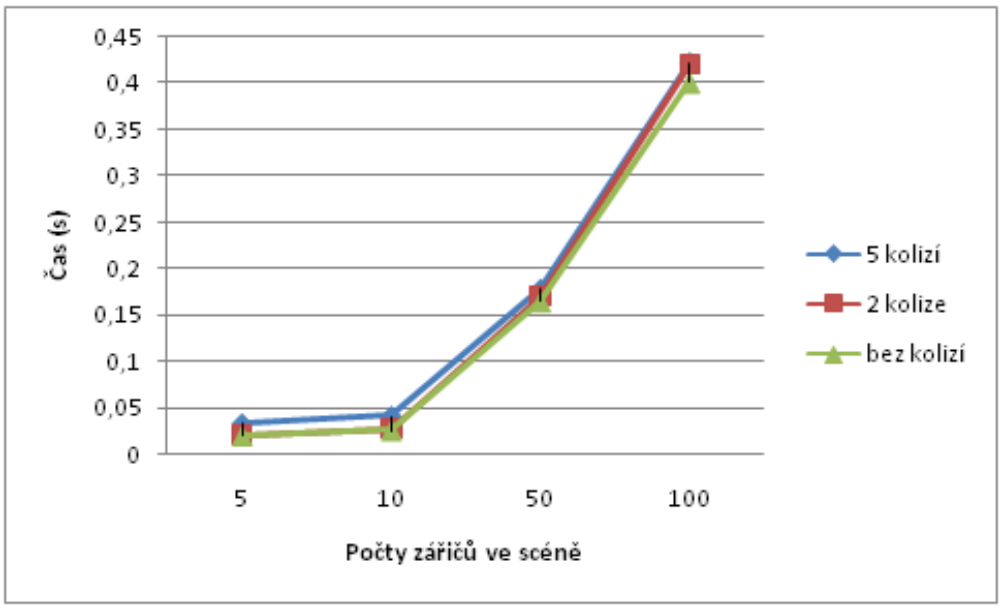
Tabulka 5: SAT porovnání k určení bezkolizního stavu dvou OBB

$\overrightarrow{L}$	$\lvert\overrightarrow{T} \cdot \overrightarrow{L}\rvert$	r_A	r_B
$a_0^{\overrightarrow{A}}$	$\lvert t_0\rvert$	e_0^A	$e_0^B\lvert r_{00}\rvert + e_1^B\lvert r_{10}\rvert + e_2^B\lvert r_{20}\rvert$
$a_1^{\overrightarrow{A}}$	$\lvert t_1\rvert$	e_1^A	$e_0^B\lvert r_{01}\rvert + e_1^B\lvert r_{11}\rvert + e_2^B\lvert r_{21}\rvert$
$a_2^{\overrightarrow{A}}$	$\lvert t_2\rvert$	e_2^A	$e_0^B\lvert r_{02}\rvert + e_1^B\lvert r_{12}\rvert + e_2^B\lvert r_{22}\rvert$
$a_0^{\overrightarrow{B}}$	$\lvert t_0r_{00} + t_1r_{10} + t_2r_{20}\rvert$	$e_0^A\lvert r_{00}\rvert + e_1^A\lvert r_{10}\rvert + e_2^A\lvert r_{20}\rvert$	e_0^B
$a_1^{\overrightarrow{B}}$	$\lvert t_0r_{01} + t_1r_{11} + t_2r_{21}\rvert$	$e_0^A\lvert r_{01}\rvert + e_1^A\lvert r_{11}\rvert + e_2^A\lvert r_{21}\rvert$	e_1^B
$a_2^{\overrightarrow{B}}$	$\lvert t_0r_{02} + t_1r_{12} + t_2r_{22}\rvert$	$e_0^A\lvert r_{02}\rvert + e_1^A\lvert r_{12}\rvert + e_2^A\lvert r_{22}\rvert$	e_2^B
$a_0^{\overrightarrow{A}} \times a_0^{\overrightarrow{B}}$	$t_2r_{10} - t_1r_{20}$	$e_1^A\lvert r_{20}\rvert + e_2^A\lvert r_{10}\rvert$	$e_1^B\lvert r_{02}\rvert + e_2^B\lvert r_{01}\rvert$
$a_0^{\overrightarrow{A}} \times a_1^{\overrightarrow{B}}$	$t_2r_{11} - t_1r_{21}$	$e_1^A\lvert r_{21}\rvert + e_2^A\lvert r_{11}\rvert$	$e_0^B\lvert r_{02}\rvert + e_2^B\lvert r_{00}\rvert$
$a_0^{\overrightarrow{A}} \times a_2^{\overrightarrow{B}}$	$t_2r_{12} - t_1r_{22}$	$e_1^A\lvert r_{22}\rvert + e_2^A\lvert r_{12}\rvert$	$e_0^B\lvert r_{01}\rvert + e_1^B\lvert r_{00}\rvert$
$a_1^{\overrightarrow{A}} \times a_0^{\overrightarrow{B}}$	$t_0r_{20} - t_2r_{00}$	$e_0^A\lvert r_{20}\rvert + e_2^A\lvert r_{00}\rvert$	$e_1^B\lvert r_{12}\rvert + e_2^B\lvert r_{11}\rvert$
$a_1^{\overrightarrow{A}} \times a_1^{\overrightarrow{B}}$	$t_0r_{21} - t_2r_{01}$	$e_0^A\lvert r_{21}\rvert + e_2^A\lvert r_{01}\rvert$	$e_0^B\lvert r_{12}\rvert + e_2^B\lvert r_{10}\rvert$
$a_1^{\overrightarrow{A}} \times a_2^{\overrightarrow{B}}$	$t_0r_{22} - t_2r_{02}$	$e_0^A\lvert r_{22}\rvert + e_2^A\lvert r_{02}\rvert$	$e_0^B\lvert r_{11}\rvert + e_1^B\lvert r_{10}\rvert$
$a_2^{\overrightarrow{A}} \times a_0^{\overrightarrow{B}}$	$t_1r_{00} - t_0r_{10}$	$e_0^A\lvert r_{10}\rvert + e_1^A\lvert r_{00}\rvert$	$e_1^B\lvert r_{22}\rvert + e_2^B\lvert r_{21}\rvert$
$a_2^{\overrightarrow{A}} \times a_1^{\overrightarrow{B}}$	$t_1r_{01} - t_0r_{11}$	$e_0^A\lvert r_{11}\rvert + e_1^A\lvert r_{01}\rvert$	$e_0^B\lvert r_{22}\rvert + e_2^B\lvert r_{20}\rvert$
$a_2^{\overrightarrow{A}} \times a_2^{\overrightarrow{B}}$	$t_1r_{02} - t_0r_{12}$	$e_0^A\lvert r_{12}\rvert + e_1^A\lvert r_{02}\rvert$	$e_0^B\lvert r_{21}\rvert + e_1^B\lvert r_{20}\rvert$

5.3 Výsledky

Algoritmus SAT popsaný výše je tedy základem funkce, která byla naprogramována v rámci práce k detekci kolizí. V první fázi byly všechny části potřebné pro práci tohoto algoritmu programovány zcela od základu bez jakýchkoliv optimalizací. Jednalo se například o vektorový součin, odčítání vektorů atd. Celý algoritmus detekce fungoval spolehlivě a detekoval veškeré i sebemenší kolize. Velkou výhodou použití tohoto algoritmu je možnost zadat si toleranci při detekci kolizí. To se provede připočtením zvolené velikosti tolerance k rozměrům kvádrů, které obalují jednotlivé části zářiče.

V pozdějších fázích projektu přišlo na řadu testování kolizí mezi větším počtem zářičů. Pro provedení měření se ale ukázalo, že rychlost provedení skriptu je poměrně malá a při větším počtu zářičů ve scéně výrazně klesá. Proto bylo potřeba přistoupit k optimalizacím skriptu. Jako první se nahradily všechny funkce pro práci s vektory psané ručně. Byly nahrazeny optimalizovanými funkcemi, které jsou obsaženy v modulech mathutils a math. To vedlo k velmi výraznému urychlení skriptu. V současné době je tedy možné zapnout automatickou detekci, a pokud uživatel pohne zářičem do polohy, kdy koliduje s jiným a pustí tlačítko myši, dojde k okamžité vizualizaci kolize. To splňuje požadavky, které si kladla společnost Lenam v prvních fázích projektu. Na následujícím grafu je vidět závislost délky vykonávání skriptu na počtu zářičů ve scéně.

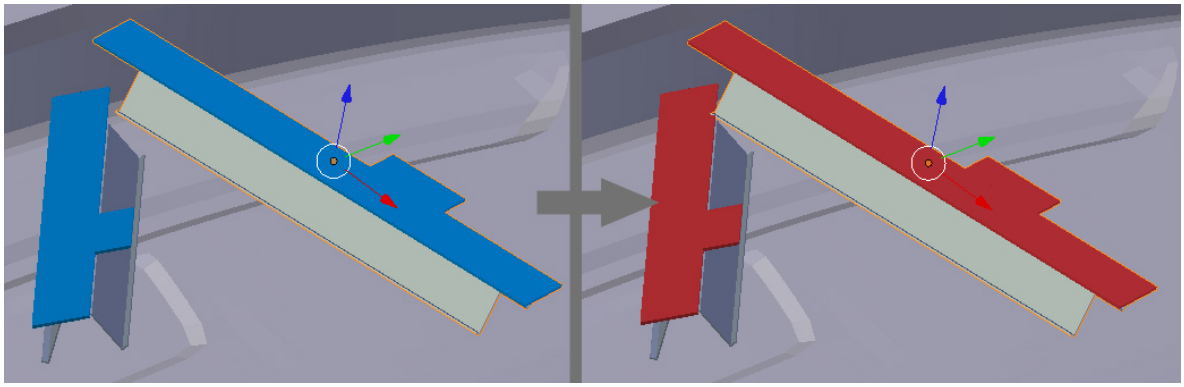


Obrázek 15: Graf závislosti času provádění skriptu na počtu zářičů

Graf je vytvořen pro pět až sto zářičů ve scéně, což je pro testování rychlosti provádění skriptu dostačující. Čas se měří v případě, že nekolidují žádné zářiče, dva a pět zářičů. Z grafu je viditelná značná závislost času provádění na počtu zářičů ve scéně. Tato doba ale není nikterak výrazná a při použití sta zářičů se doba provedení skriptu pohybuje do půl sekundy. Jedná se o uspokojivý výsledek vzhledem k tomu, že v praxi by neměl být tento počet výrazně vyšší. Rozdíly v časech při různém počtu kolidujících zářičů jsou pak zanedbatelné.

Celkové ohlasy na tuto kolizní metodu byly pozitivní. Její použití je flexibilní a dá se aplikovat i na další zářiče, které se mohou lišit rozměry, či počtem kvádrů, ze kterých jsou složeny. Jádro zůstává vždy stejné a jedná se pouze o změnu čtyř čísel pro jeden kvádr, které se dají snadno odečíst z modelu zářiče. Pokud se jedná o rozšíření zářiče o další části (kvádry), přidá se do skriptu pouze jeden řádek opět obsahující čtyři čísla a skript sám pak provádí detekce i pro tyto další části objektu. Nespornou výhodou je také možnost přidat toleranci, která zajistí určitou míru vůle mezi zářiči.

Jediné poznámky byly směřovány opět k optimalizacím a ještě většímu urychlení skriptu. Ve skriptu je ještě možné provést optimalizace například v různých částech algoritmu SAT, případně v některých dílčích částech skriptu. Zásahy by však nevedly k výraznějšímu zrychlení skriptu. Proto budou tyto optimalizace provedeny někdy v budoucnu. Na závěr této kapitoly je uveden ilustrační obrázek, zobrazující průběh a vizualizaci kolidujících zářičů.



Obrázek 16: Zobrazení volných a kolidujících zářičů

6 Geometrická vizualizační metoda

První z metod, sloužících k vizualizaci stopy zářiče, má zkratku GEOM. Důvodem volby uvedené zkratky je podstata této metody, která je založena na reprezentaci kužele pomocí geometrických objektů. Hlavním požadavkem na zmíněnou metodu byla rychlost provedení vizualizace. Dále má funkce zobrazovat pouze jednobarevnou stopu. Uživatel tedy bude vidět pouze oblast, kterou pokryje záření, ale nebude mít žádné informace o teplotách, kterých galvano pod zářičem může dosáhnout. Díky tomu, že na tuto metodu nebyly kladeny žádné další požadavky, bylo možné předpokládat, že se bude jednat o poměrně jednoduchou metodu. Bylo tedy zapotřebí nějakým způsobem definovat kužel světla pod zářičem a následně ověřit, zda tento kužel zasahuje na galvano. Zpočátku se vedly diskuse nad několika nápady a postupy, jak toho dosáhnout. Nakonec se ujaly pouze dva. První definoval kužel pomocí geometrických objektů, kde by se následně kontrolovalo, zda polygony leží vně, či uvnitř takto definované obálky. Druhý byl založen na představě vysílání velkého množství paprsků z objektu zářiče směrem ke galvanu a následně měl kontrolovat, zda paprsky projdou některým z polygonů.

Následující kapitoly popisují různé způsoby, jak najít průsečík paprsku a geometrického útvaru. Nalezení průsečíků je nutné k vytvoření druhého způsobu obarvování galvana. Metody potřebné k vytvoření obálky a hledání, zda obálka zasahuje galvano, jsou v podstatě detekce kolizí, které byly popsány výše.

6.1 Průsečík paprsku s objektem

V úvodu této kapitoly byly definovány principy, na kterých tato metoda funguje. Základem druhého postupu hledání obálkou zasažených polygonů je tedy hledání průsečíku paprsku s ploškou. S těmito ploškami se nadále pracuje. Jedná se především o změnu její barvy. Následující kapitoly popisují způsoby nalezení průsečíků.

6.1.1 Paprsek a polygon

Budeme předpokládat,že e polygon je rovinný. Dále je potřeba definovat si vstupy, kterými jsou paprsek R a polygon. Nejprve se spočítá normálový vektor roviny. Dále se určí průsečík ri paprsku s rovinou plošky a nakonec se otestuje, zda průsečík r_i leží uvnitř plošky.

Máme tedy k dispozici vstupy v podobě polygonu definovaného N vrcholy (X_i, Y_i, Z_i), kde $0 < i < N$ a paprsku R, jehož počátek je v bodě $R_0(X_0, Y_0, Z_0)$. Směr paprsku určuje vektor $R_d(X_d, Y_d, Z_d)$. Vstupy dosadíme do rovnice přímky (7), která nám určuje paprsek.

$$R(t) = R_0 + R_d \cdot t, \ kde\ t > 0 \tag{7}$$

Dále bude potřeba znát implicitní tvar rovnice roviny (8) s normalizovanými koeficienty A,B a C

$$a_i \cdot x + b_i \cdot y + c_i \cdot z + d_i = 0 \tag{8}$$

Jde tedy o to, nalézt koeficienty A,B,C a D. První tři jsou koeficienty normálového vektoru roviny. Proměnné x,y,z jsou souřadnice bodu ležícího v rovině. Poslední koeficient di získáme tak, že dosadíme souřadnice některého bodu roviny do obecné rovnice roviny. [12]

K rozhodnutí, zda paprsek rovinu protnul či nikoliv, potřebujeme provést další výpočty. Jde o dosazení rovnice přímky (7) do rovnice roviny (8). Po několika úpravách se dostaneme k rovnici (9) s vyjádřeným parametrem t, který potřebujeme zjistit.

$$t = \frac{-(A \cdot X_0 + B \cdot Y_0 + C \cdot Z_0 + D)}{A \cdot X_d + B \cdot Y_d + C \cdot Z_d} \tag{9}$$

[12] Wikipedie [online]. 2010 [cit. 2010-12-10]. Rovina - Wikipedie. Dostupné z WWW: http://cs.wikipedia.org/wiki/Rovina. [webová stránka]

Po dopočítání parametru t již můžeme přistoupit k dosazení do rovnice (7), čímž se dostaneme k nalezení souřadnic průsečíku. Posledním krokem, když jsme již nalezli souřadnice průsečíku, je zjistit, zda bod skutečně leží uvnitř polygonu. Následující odstavce ve stručnosti popisují tři základní metody, jak toho docílit.

Crossing test spočívá ve vyslání paprsku z místa průsečíku v kladném směru osy X a počítání, kolikrát paprsek protne hranu polygonu. Pokud je počet nalezených průsečíků lichý, pak bod leží uvnitř polygonu. Dále je možné použít zjednodušení, kterým je posunutí polygonu tak, aby nalezený průsečík ležel v počátku souřadného systému.

Angle summation test je založen na principu sčítání orientovaných úhlů mezi dvojicemi hran sousedních vrcholů a průsečíkem. Pokud tento součet vyjde 360°, potom tento bod leží uvnitř polygonu, pokud naopak vyjde 0, leží vně. Bohužel jde o velice pomalou metodu, neboť se počítá arkustangens velikosti jejich hran.

Barycentric algorithm je obdobný jako crossing test. Rozdíl spočívá v nahlížení na polygon, kdy jej bereme jako množinu trojúhelníků a pro každý trojúhelník testujeme, zda má bod barycentrické souřadnice, nacházející se uvnitř tohoto trojúhelníku. [13]

6.1.2 Paprsek a trojúhelník (ray cast)

Nalezení průsečíku objektu a paprsku je složitá operace. Lze ji ale zjednodušit. Ať je objekt jakkoliv složitý, dá se rozčlenit na síť trojúhelníků. Ze zdánlivě neřešitelného problému nalezení průsečíku objektu a paprsku se pak stane jednodušší test paprsek - trojúhelník. Existují testy, které jsou rychlejší, pokud je výsledek pozitivní (paprsek zasáhne trojúhelník), nebo naopak negativní. Některé potřebují ke své práci velké množství paměti pro předpočítávání nejrůznějších mezivýsledků, jiné naopak nic dopředu nepočítají. Všechny algoritmy mají ale jedno společné a to jsou dva základní vstupy. Prvním jsou vrcholy trojúhelníku (V_0, V_1, V_2) orientované proti směru hodinových ručiček a druhým paprsek obvykle popsaný parametrickou rovnicí. Existuje několik metod, pomocí kterých se dá průsečík spočítat, většina z nich používá k nalezení průsečíku barycentrických souřadnic (u, v).

[13] KAŇOVSKÝ, Tomáš. Základní geometrické operace v počítačové grafice. Brno, 2006. 29 s. Bakalářská práce. Masarykova univerzita.

Nejdříve je zapotřebí najít průsečík paprsku s rovinou trojúhelníku P. Postupuje se stejně jako při hledání průsečíku paprsku s polygonem. Nalezený průsečík se označí r_i. Dále se dekomponuje barycentrická rovnice pro bod v trojúhelníku (10) na systém tří rovnic (14).

$$r_i = (1 - u - v) \cdot V_0 + u \cdot V_1 + v \cdot V_2 \tag{10}$$

$$P - V_0 = u \cdot (V_1 - V_0) + v \cdot (V_2 - V_0) \tag{11}$$

$$\begin{aligned} x_p - x_0 &= u \cdot (x_1 - x_0) + v \cdot (x_1 - x_0) \\ y_p - y_0 &= u \cdot (y_1 - y_0) + v \cdot (y_1 - y_0) \\ z_p - z_0 &= u \cdot (z_1 - z_0) + v \cdot (z_1 - z_0) \end{aligned} \tag{12}$$

Následující krok je projekce trojúhelníku na jednu ze souřadných rovin (xy, xz, nebo yz). Nalezneme souřadnici i_0 s největší magnitudou, kde N_x, N_y, aN_z jsou jednotlivé složky normálového vektoru N a následně se podle ní provede projekce.

$$i_0 = \begin{cases} 0 - pokud |N_x| = max(|N_x|, |N_y|, |N_z|) \\ 1 - pokud |N_y| = max(|N_x|, |N_y|, |N_z|) \\ 2 - pokud |N_z| = max(|N_x|, |N_y|, |N_z|) \end{cases} \tag{13}$$

Poznamenejme, že $i_1 a i_2 (i_1 a i_2 \in 0, 1, 2)$, jsou indexy různé od . Reprezentují primární rovinu (rovinu, na kterou se bude provádět projekce). Pokud je např. $i_0 = 0$, potom nulujeme x-ové souřadnice.

$$\begin{aligned} u_0 = P_{i_1} - V_{0_{i_1}}, u_1 = V_{1_{i_1}} - V_{0_{i_1}}, u_2 = V_{2_{i_1}} - V_{0_{i_1}} \\ v_0 = P_{i_2} - V_{0_{i_2}}, v_1 = V_{1_{i_2}} - V_{0_{i_2}}, v_2 = V_{2_{i_2}} - V_{0_{i_2}} \end{aligned} \tag{14}$$

Rovnice se pak zjednoduší takto:

$$\begin{aligned} u_0 &= u \cdot u_1 + v \cdot u_2 \\ v_0 &= u \cdot v_1 + v \cdot v_2 \end{aligned} \tag{15}$$

Konečně se dostáváme k závěrečnému řešení u a v, které má tvar (16)

$$u = \frac{det\begin{pmatrix} u_0 & u_2 \\ v_0 & v_2 \end{pmatrix}}{det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}},\ v = \frac{det\begin{pmatrix} u_1 & u_0 \\ v_1 & v_0 \end{pmatrix}}{det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}} \tag{16}$$

Algoritmus je celkem jednoduchý, ale v porovnání například s algoritmem Moller-Trumbone je ve složitých trojúhelníkových sítích pomalejší, protože je nutné ukládat normálový vektor, tím pádem pro větší sítě roste objem dat nutných k uložení. [14]

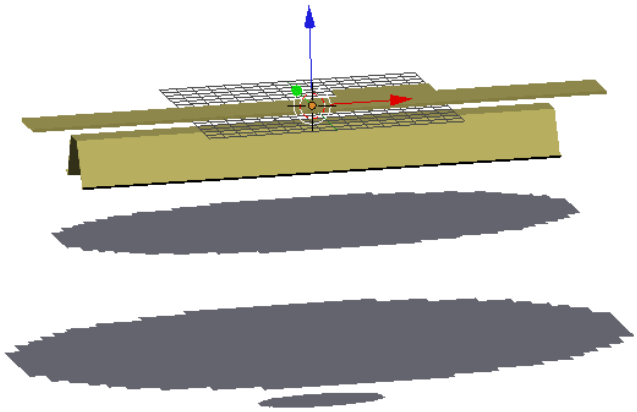
6.2 Postup při realizaci geometrické funkce

K uspokojivým výsledkům při řešení této funkce nevedla hned první cesta, ale bylo zapotřebí vyzkoušet několik různých alternativ. Následující podkapitoly popisují dva postupy. První z nich je založen na detekci kolizí a druhý na hledání průsečíku definovaného paprsku a objektu. Druhý ze jmenovaných pak vykazoval velmi pěkné výsledky a byl tedy použit ve finální verzi programu.

6.2.1 Řešení geometrické metody pomocí detekce kolizí

Jelikož detekce kolizí se již řešila v podobě metody testující střet dvou zářičů, vedla první myšlenka k vyřešení této metody opět k detekování kolizí. Myšlenkou, na které je celá metoda založena, je ověřování, zda se bod nachází uvnitř tělesa, které reprezentuje kužel světla vycházející ze zářiče. Základním předpokladem je tedy vytvoření obálky, jež by co nejpřesněji aproximovala světelné záření vycházející od zdroje. První představou, jak toho docílit, byly zkušenosti pracovníků společnosti Lenam. Ti si představovali tuto obálku jako kužel rozšiřující se do určité vzdálenosti od zářiče a opět v určité vzdálenosti se znovu zužující. Pro vytvoření této obálky bylo potřeba zjistit některé údaje.

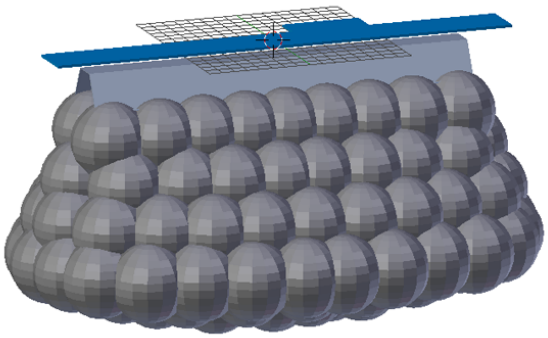
[14] KAŇOVSKÝ, Tomáš. Základní geometrické operace v počítačové grafice. Brno, 2006. 29 s. Bakalářská práce. Masarykova univerzita.



Obrázek 17: Vrstvy vymezující působení zářiče

Po několika diskuzích se přistoupilo k měření vybraných charakteristik zářiče. Z nich se dále vytvořilo několik vrstev, které určují meze působení záření vycházejícího ze zářiče. Zpracovaná data jsem následně obdržel jako soubor pro program Blender, jež můžete vidět na obrázku (17). Pod objektem se nachází několik vrstev elips, jež ukazují vliv zářiče v určitých vzdálenostech. Vytvoření obálky přesně reprezentující tyto vrstvy by bylo komplikované. To samé platí o ověření, zda se body na galvanu nacházejí uvnitř obálky. Proto se přistoupilo ke kompromisu a obálka se vytvořila pomocí velkého počtu kuliček. Ty se rozmístí tak, aby co nejpřesněji aproximovaly kužel světla, vycházejícího ze zářiče. Samotná detekce, zda se bod nachází uvnitř kuličky, je pak velice snadná a rychlá. Jak vypadá výsledná struktura obálky složené z kuliček, je uvedeno na obrázku (18).

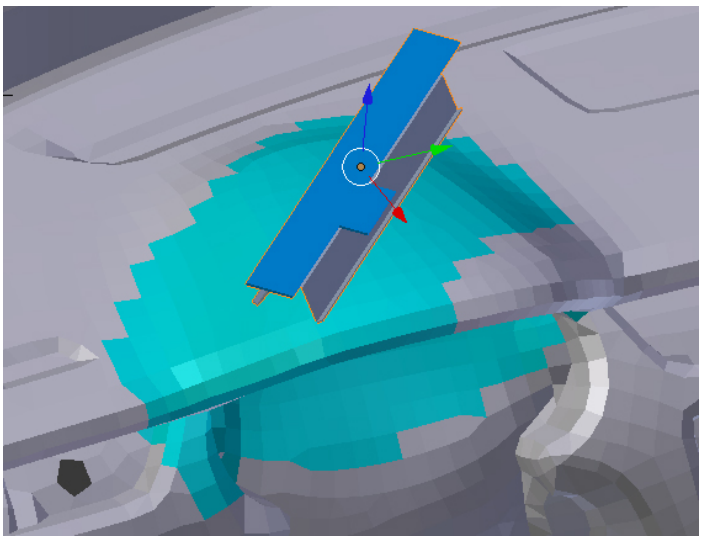
Výhodou obálky složené z kuliček, je invariantnost koule vůči rotacím a tudíž stačí v případě změny polohy zářiče přepočítat pouze souřadnice středu koule. Pokud by byla obálka sestavena z několika kvádrů, nad čím se také uvažovalo, výpočtů by se muselo provést větší množství a tím by se zvýšila i celková časová náročnost výpočtu. Reprezentace obálky je tedy hotova. Dále se musí definovat body, u kterých budeme testovat, zda se nacházejí pod zářičem, či nikoliv.



Obrázek 18: Výsledná podoba obálky

V Blenderu je vše dostupné jako objekt, nad kterým se dají provádět různé operace. Dále je možné o objektu zjistit spousty informací. Každý objekt je složen z bodů, propojených v plošky, tvořících výsledný objekt. K jednotlivým ploškám se dá přistupovat a dále o nich zjišťovat různé informace. Například indexy vertexů tvořící plošku, normálový vektor, či souřadnici středu plošky. Poslední zmiňovaná informace je pro tuto metodu velice dobře využitelná, jelikož použijeme tento bod pro testování, zda se nachází pod světelným kuželem. Pokud se bude bod nacházet v zasažené oblasti, pomocí jednoduché metody se dohledá jeho index. Plošce s tímto indexem se nastaví určitá barva, odlišná od barvy neozářeného galvana. Výsledky této funkce po jejím průběhu ilustruje obrázek (19). V prvních chvílích se zdálo, že výsledky jsou uspokojivé a celá metoda probíhala vcelku rychle. Následně se ale ukázalo, a je to zřejmé i z obrázku (19), že metoda obarvuje i plošky, které jsou zastíněny a tudíž by na ně zářič neměl mít žádný vliv.

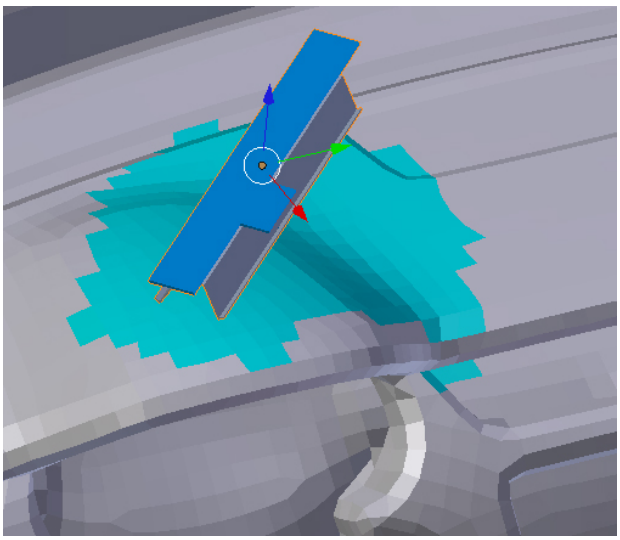
Nad odstraněním tohoto problému se několikrát vedly diskuse a nakonec se našlo řešení v podobě použití normál plošek. Řešení spočívá v nalezení normály směřující k zářiči. Tato normála se určí pomocí paprsku, jehož směr je určen natočením zářiče a jeho počátek je ve středu zářiče. Z plošky, kterou paprsek protne, vezmeme normálový vektor. Z jeho složek se pak vybere ta s největší magnitudou. Následně se určí toleranční okruh, který se volí v rozmezí 0.1 až 0.2. Dalším úkonem je porovnávání zvolených složek normál plošek. Porovnává se to, zda je vybraná složka normály plošky větší než zvolená složka určující normály. Pokud je podmínka splněna, tak plošku obarvíme. Výsledky jsou patrné z obrázku (20).



Obrázek 19: Výsledek průběhu geometrické metody

Na obrázku (20) je již patrné výrazné zlepšení problému s obarvováním zastíněných plošek, ale stále je zde několik problémů. Prvním je volba hlavní normály. Při jejím určování by mohlo dojít k zasažení výrazně natočeného plošky v poli jinak kolmo k zářiči orientovaných, což by vedlo k špatnému výsledku.

Dalším je opět obarvování zastíněných. Pokud bude orientace zastíněných plošek stejná jako orientace určující plošky a obálka by je zasahovala, opět by došlo k jejich nesprávnému obarvení. Tento problém se dá částečně vyřešit určením vzdálenosti mezi určující ploškou a zářičem. Následně se bude kontrolovat, zda se plošky se stejnou orientací příliš neliší od hlavní ve vzdálenosti od zářiče. To by ale vedlo k nutnosti provádět další výpočty a zpomalení skriptu. Po provedení několika dalších analýz byl stanoven závěr, který klade za úkol zkusit navrhnout tuto funkci úplně jinak a to s využitím funkce rayCast integrované v modulu programu Blender.



Obrázek 20: Výsledek průběhu funkce GEOM s použitím normály

6.2.2 Řešení pomocí funkce rayCast

Tato funkce funguje na principu hledání průsečíku paprsku a trojúhelníku, jejíž popis je možné najít v kapitole 6.1.2. Pomocí tohoto popisu vznikla nová metoda, která po zadání vstupních parametrů vrátila index plošky protnuté paprskem. Dále bylo potřeba definovat obálku. Ta se skládá z velkého množství paprsků vysílaných do určité vzdálenosti od zářiče.

Tyto paprsky je možné volitelně definovat podle toho, jak složitá je síť galvana, tak aby se zajistilo, že všechny plošky pod zářičem budou zasaženy alespoň jedním z paprsků. Počáteční body paprsků se definují na kružnicích, rozmístěných pod zářičem. Tyto kružnice jsou vždy ve dvojicích. První se nachází ihned pod zářičem a definuje počáteční body paprsků. Druhá je v určité vzdálenosti od zářiče, případně se posune, aby tvar obálky co nejlépe odpovídal naměřeným charakteristikám, a body na této kružnici jsou konečnými body pro paprsky. Dále funkce umožňuje definovat si počet kružnic, rozmístěných pod zářičem a také počet bodů, který se na obvodu této kružnice vytvoří. Díky těmto vlastnostem je možné odladit si funkci tak, aby při použití na konkrétní síti dávala dobré výsledky a její provedení netrvalo zbytečně dlouho.

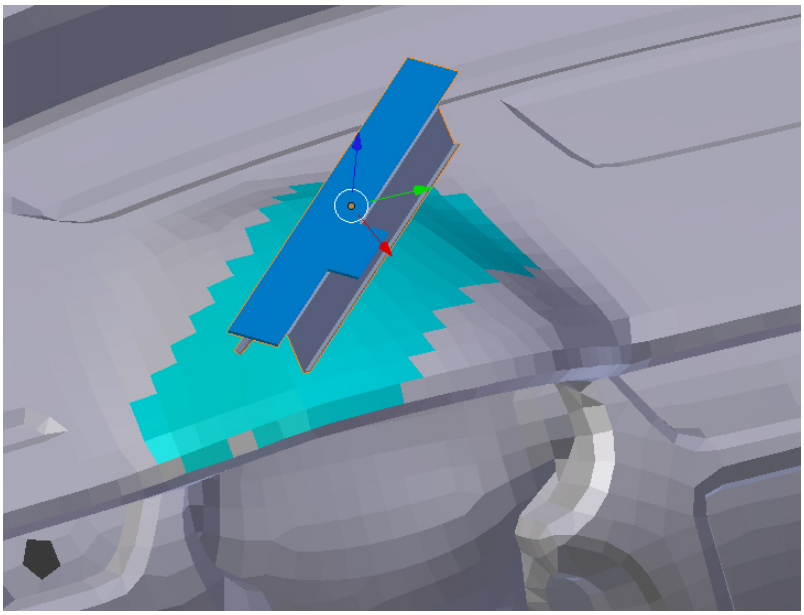
I přes možnost doladit si funkci podle potřeb trvalo její provádění o 1/3 déle než v předcházejícím případě. Proto se opět přistoupilo k optimalizacím. Funkce pro hledání průsečíků, která byla napsána, se ukázala jako nejslabší článek. Po konzultaci s vedoucím diplomové práce jsem se dozvěděl o existenci funkce ray_cast, která pracuje stejně jako mnou naprogramovaná, ale daleko rychleji a je součástí vestavěných modulů Blenderu. Proto byla celá funkce předělána a výsledkem bylo podstatné urychlení, po kterém se výsledná rychlost přiblížila rychlostem, kterých dosahovala funkce řešená pomocí kolizí. Další problém s trváním výpočtu nastal při použití většího počtu zářičů. Když se ve scéně nacházel zářič, pro který již bylo vše vypočteno a stopa zobrazena, provedl se výpočet znovu a pro velké množství zářičů se stala metoda pomalá. Proto se provedla optimalizace, ve které se výpočet provádí vždy jen pro nově přidané zářiče či zářiče, se kterými se od posledního výpočtu nějakým způsobem manipuluje.

Princip této optimalizace je následující. V prvním kroku se načtou data o zářičích, která obsahují následující informace o zářiči. Jeho jméno, údaj o jeho existenci, rotaci, pozici a ozářených ploškách. Ozářené plošky jsou uloženy v seznamu, jehož klíč je index plošky a hodnotou je intenzita ozáření. Jelikož se jedná o metodu GEOM, je hodnota ozáření vždy nulová. Po načtení dat se všem zářičům nastaví index u existence na nulu. Dále se začnou procházet objekty ve scéně. Pokud se narazí na objekt, který má v názvu zkratku IRE, tak zkontrolujeme, zda je obsažen v načtených datech. Pokud není, vygenerujeme pro něj potřebné atributy a uložíme je. Jestli již objekt v datech je, porovnáme jeho pozici a rotaci obsaženou v datech s jeho aktuální. Když se budou lišit, provedeme výpočet ozářených plošek a opět uložíme jeho aktuální pozici a rotaci, plus existenci nastavíme na jedničku. Pokud by se v datech nacházel nějaký zářič s existencí nastavenou na nulu, což znamená, že ho uživatel smazal, provedlo by se jeho odstranění z dat. Následuje zobrazení všech ozářených plošek a uložení aktuálních dat.

Tímto postupem se zredukuje velké množství výpočtů, které jsou nutné k nalezení průsečíků paprsků a galvana. Výpočty se pak provádí jen pro nově přidané zářiče, nebo zářiče, s kterými se manipulovalo. Proto výsledná doba provedení metody závisí pouze na počtu provedených změn mezi zářiči. Metodu je možné vyvolat pomocí ovládacího prvku, nebo je možné spustit její automatické provádění, které je založeno na změnách polohy či rotace zářiče. Pokud je změna detekována, metoda se automaticky spustí.

6.3 Výsledky metody GEOM

Na obrázku (21) je vidět podstatné zlepšení výsledného obarvení galvana. Obarví se pouze plošky pod zářičem, které zasáhnou paprsky. Dále je úplně odstraněn problém se zastíněnými ploškami díky použití funkce rayCast, jež vrací vždy pouze index první zasažené plošky. Prezentované výsledky nejlépe odpovídaly skutečnosti a také představám pracovníku společnosti Lenam, proto je tato metoda použita ve finální verzi programu.



Obrázek 21: Výsledek průběhu funkce GEOM s použitím funkce rayCast

7 Metoda FUNC

Jedná se o druhou z vizualizačních metod. Požadavky na tuto metodu byly následující. Při jejím vyvolání metoda zobrazí barevnou stopu zářiče. Jednotlivé barvy pak budou reprezentovat dosažené teploty. Pokud by barevná škála byla příliš rozsáhlá a nebylo by jasné přiřazení barev na galvanu, je možné na panelu barvu změnit a změna se promítne i na galvano.

7.1 Postup při realizaci

Princip této metody řešili pracovníci společnosti Lenam. Základem je soubor dat, získaných měřením na speciálním zařízení. To bylo složeno z několika set tepelných čidel rozmístěných na ploše, nad níž byl umístěn zářič. Jeho vzdálenost od čidel se dala pomocí tohoto zařízení nastavit. Následně byla naměřena potřebná data. Tyto data tvoří koordináty X, Y a Z. X a Y jsou čísla udávající polohu konkrétního čidla na ploše a Z je vzdálenost od zářiče. Data bylo potřeba vhodně uspořádat tak, aby práce s nimi byla co nejjednodušší a nemusela se složitě transformovat. Tabulka (6) je malou ukázkou části naměřených dat. Vybralo se toto uspořádání dat. V prvním sloupci je hodnota reprezentující vzdálenost zářiče od plochy. Druhou položkou v tabulce je pozice x, následuje y a posledním údajem je naměřená teplota. Hodnoty jsou uvedeny v milimetrech. Výjimku tvoří pouze poslední sloupec, kde jsou uvedeny hodnoty ve stupních Celsia. Tato data jsou pak uložena v souborech pojmenovaných podle vzdálenosti, pro kterou byly naměřeny.

Metoda pak probíhá dle následujícího scénáře. První se opět provede načtení objektu galvana. Následuje načtení dat, která této metodě předá funkce GEOM. Jedná se o indexy plošek, jež jsou obarveny pod jednotlivými zářiči. K tomuto řešení se přistoupilo po několika pokusech, kdy se zkoumala doba přiřazení všech naměřených dat jednotlivým ploškám. To se ukázalo velmi časově náročné a opět zde vystal problém s obarvováním zastíněných plošek. Jelikož to již bylo vyřešeno v GEOM, s výhodou se tak využila data, která nám tato metoda poskytuje.

Tabulka 6: Výsledná tabulka naměřených hodnot

z	x	y	T [C]
-110	-390	-300	38,72
-110	-390	-270	38,78
-110	-390	-240	38,85
-110	-390	-210	38,89
-110	-390	-180	38,92
-110	-390	-150	39,02

Poté je na řadě načtení dat, získaných měřením. Ta byla rozdělena do šesti souborů a jsou pojmenovány podle vzdáleností, ve kterých se měřily charakteristiky zářiče. Z těchto souborů se pak pomocí následujícího regulárního výrazu vezmou a upraví data do požadované formy.

rv = re.compile('S[0-9.0-9]+')

Následuje již samotná hlavní metoda. V té se opět na začátku projdou všechny objekty ve scéně, a pokud se narazí na objekt zářiče, vloží se do pole. To se následně projde a pro každý objekt se pomocí funkce ray_cast provede zjištění průsečíku s galvanem. Tento průsečík se pak bere jako referenční bod se souřadnicemi (0,0,0). Dále se vezmou indexy plošek, které tento zářič zasahuje, a od jejich souřadnic středů se odečtou souřadnice bodu, který byl určen jako referenční. Tím se docílí sjednocení souřadnic středů faců galvana a souřadnic, na kterých se měřila data. Následuje přiřazení odpovídající teploty jednotlivým plošek nacházejícím se v poli. Tyto informace se uloží do datové struktury. Jedná se o seznam, jehož klíče tvoří jména zářičů a hodnoty tvoří datová struktura s informacemi o pozici, rotaci, existenci a již zmiňovaný další seznam s klíči, tvořenými jednotlivými indexy plošek a hodnotou, kterou tvoří teplota.

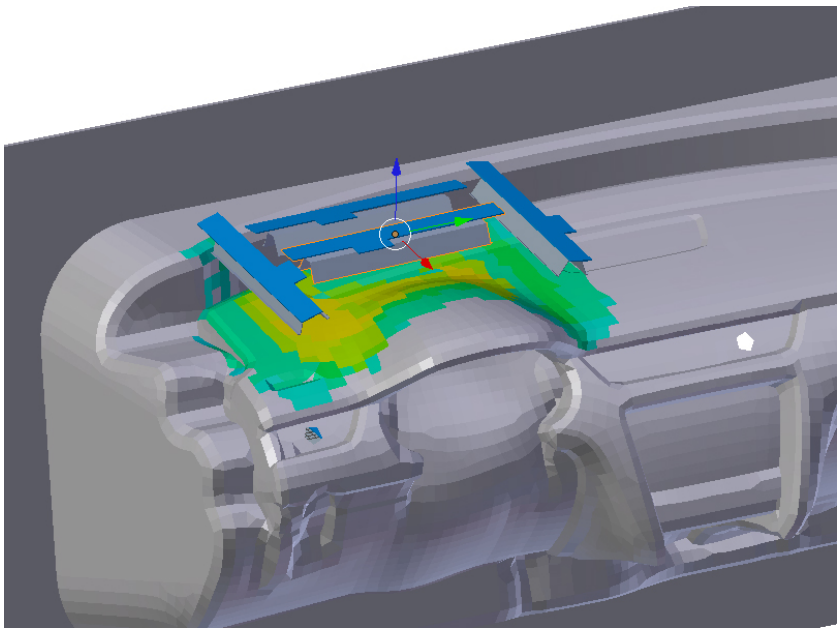
Dále je nutné ošetřit působení více zářičů na jednu plošku. To se provede tak, že se opět vezmou všechny objekty zářičů a postupně se procházejí. Procházejí se ale již jen indexy plošek a ukládají se do jednoho velkého seznamu. Jeho plnění probíhá následovně. Přijde se na objekt zářiče. Vezme se index první plošky a dotážeme se, zda je již v seznamu obsažena. Pokud nikoliv, vložíme ji a přiřadíme jí hodnotu, kterou měla v datové struktuře zářiče. Pokud je již v seznamu obsažena, například když vezmeme další zářič a ten je umístěn blízko jiného, je velká pravděpodobnost, že zářiče ovlivňují stejné plošky. Pak pouze sečteme hodnotu teploty plošky ve výsledném seznamu a plošky v datové struktuře zářiče. Nakonec zbývá pouze obarvit všechny plošky podle výsledné teploty.

Po odzkoušení metody přišla na řadu její optimalizace. Jelikož se optimalizace provedená v metodě GEOM ukázala jako velmi dobrá, provedly se tytéž optimalizace i zde. Přiřazování odpovídajících teplot se tedy provádí jen v případě, že se se zářičem od posledního výpočtu hnulo nebo rotovalo. Dosáhlo se tak dobré rychlosti provedení metody.

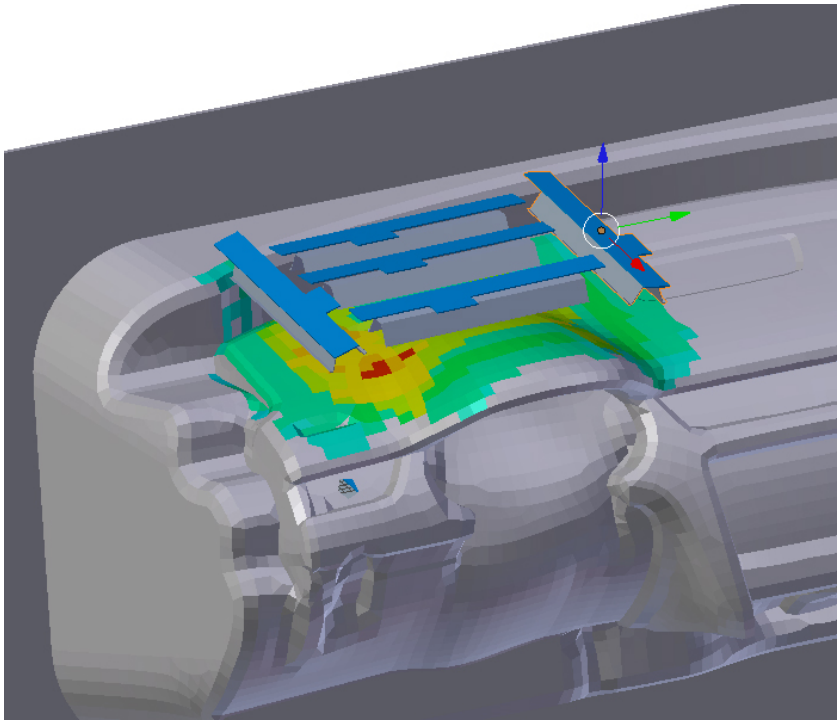
7.2 Výsledky průběhu metody FUNC

Tato metoda měla být podle požadavků pracovníků společnosti Lenam založena na funkci definování rozložení vyzářené energie na ploše. Toho se docílilo pomocí měření teplot pod zářičem v určitých vzdálenostech. Tyto naměřené hodnoty následně skript přiřazuje odpovídajícím ploškám na galvanu. Metoda pracuje na vyžádání, nebo se při provedení změny polohy či rotace zářiče automaticky vyvolá. Jedním z požadavků bylo také zohlednění orientace ploch, na které záření dopadá. Data, která byla k dispozici při vzniku této práce, byla naměřena pouze pro plošky kolmé k zářiči. Proto v ní není zohledněno natočení plošek vůči zářiči. Metoda je ale navržena tak, aby po naměření těchto dat bylo možné metodu nepatrně upravit a využít tak i data naměřená pro plošky, které nejsou k zářiči kolmé.

Závěrem této kapitoly bych rád uvedl dva ilustrační obrázky, ukazující jakých výsledků se dá dosáhnout metodou FUNC. Jde o ilustrace jen s malým počtem zářičů. Cílem je ukázat změnu ohřevu galvana při přidání zářiče. První obrázek ukazuje ohřev při použití čtyř zářičů. Následně je přidán jeden zářič a ihned se dostáváme do vyšších teplot, jež reprezentuje oranžová a červená barva.



Obrázek 22: Zahřátí galvana při použití čtyř zářičů



Obrázek 23: Zahřátí galvana při použití pěti zářičů

8 Závěr

Tato práce si kladla za cíl vytvořit jeden program s jednoduchou obsluhou, který by umožňoval zastoupit programy dosud používané a zvládal všechny požadované funkce. Ke konečnému výsledku nevedla vždy první cesta, ale bylo potřeba vyzkoušet více možností a vybrat tu, která nejlépe odpovídala požadavkům a dávala nejlepší výsledky.

Hned na počátku projektu se diskutovalo o tom, zda použít již hotový program jako základ nebo začít programovat novou aplikaci. Nakonec se ukázalo že, vzhledem k časové omezenosti projektu, jako vhodnější použití již hotového a odzkoušeného programu jako základu pro další rozšiřování o potřebné funkce. Základním stavebním kamenem byl zvolen program Blender. Ten obsahoval velké množství potřebných funkcí již v základní verzi a poskytoval možnosti modifikace uživatelského rozhranní a doplňování nových metod pomocí modulů napsaných v jazyce Python.

Následovala implementace požadovaných metod. První byla metoda pro detekci kolizí mezi zářiči. Při zvolení vhodného obalového tělesa je tato metoda velice přesná a dovoluje nastavit tolerance. Je také velmi rychlá, což splňuje požadavek na okamžitou vizualizaci kolizí při provedení změny polohy, či rotace zářiče. Druhou podstatnou metodou, sloužící k vizualizaci stopy zářičů, je funkce GEOM. Požadavky, na ní kladené byly opět rychlost jejího provedení a jednoduchost. Po testování dvou možností definice kužele světla pod zářičem, z níž jedna byla založena na detekci kolizí a druhá na vysílání paprsků, byla vybrána druhá zmiňovaná. Ta dosahovala velmi dobrých výsledků a odstraňovala hlavní nedostatek první metody, což bylo obarvování i zastíněných plošek. K volbě druhé možnosti také přispěla její rychlost. Na dokončení této metody navazovala tvorba další a to FUNC. Ta bere jako základ pro svůj průběh informace z GEOM o ozářených ploškách pod jednotlivými zářiči. Samozřejmě není nutné nejdříve volat GEOM a poté FUNC, vše probíhá automaticky. Výsledky jsou pak ihned viditelné na galvanu, kde jsou barevně odlišená místa s různými teplotami. Barvenou škálu zobrazenou na galvanu je také možné si definovat podle požadavků uživatele.

Výstupy programu pak tvoří renderované obrázky ilustrující ohřev galvana. Dále je zde možnost uložit informace o rozestavení zářičů do souboru, který by se dal použít po úpravách jako vstup pro robota rozmisťujícího na lince zářiče. Závěrem se dá říci, že byly splněny všechny požadavky stanovené na začátku projektu a program je připraven pro testování. Stále však ještě zůstává prostor pro další optimalizace, které by vedly k zrychlení provádění skriptů a také pro implementaci dalších funkcí, zjednodušujících práci v tomto programu.

Reference

[1] ŽÁRA, J. - BENEŠ, B. - SOCHOR, J. - FELKEL, P. Moderní počítačová grafika. 2.vyd. Brno: Computer Press, 2005, ISBN 80-251-0454-0.

[2] POKORNÝ, Pavel. - Blender - naučte se 3D grafiku. 2.vyd. Praha: BEN - technická literatura, 2009, ISBN 80-7300-244-2.

[3] HARMS, D. - MCDONALD, K. - Začínáme programovat v jazyce Python. Brno: Computer Press, 2008, ISBN 978-80-251-2161-0.

[4] DVOŘÁKOVÁ, M. Kolizní systémy. Brno: Masarykova univerzita. Fakulta informatiky, 2007. 22 s. Vedoucí bakalářské práce: doc. Ong. Jiří Sochor, CSc.

[5] VAŠÍČEK, J. Detekce kolizí. Plzeň: Západočeská univerzita v Plzni. Fakulta aplikovaných věd, 2008. 42 s. Vedoucí diplomové práce: doc. Dr. Ing. Ivana Kolingerová.

[6] KORBA, L. Simulace řízení vozidla. Praha: České vysoké učení technické v Praze. Fakulta elektrotechnická, 2008. 61 s. Vedoucí diplomové práce: Ing. Vlastimil Havran, Ph.D.

[7] KAŇOVSKÝ, Tomáš. Základní geometrické operace v počítačové grafice. Brno, 2006. 29 s. Bakalářská práce. Masarykova univerzita.

[8] Blender [online]. 2005, [cit. 2010-11-05]. Dostupné z: <http://www.blender.org/>.

[9] Python Documentation [online]. 1990, [cit. 2010-10-04]. Dostupné z: <http://www.python.org/doc/>.

[10] SMEP [online]. 2003 [cit. 2010-11-10]. Vícekriteriální rozhodování. Dostupné z: <http://etext.czu.cz/php/skripta/skriptum.php?titul_key=79>. [webová stránka]

[11] heekscad - Free CAD based on Open CASCADE [online]. 2011 [cit. 2010-11-05]. Heekscad . Dostupné z WWW: <http://code.google.com/p/heekscad/>. [webová stránka]

[12] K-3D [online]. 2010 [cit. 2010-11-05]. K-3d.org. Dostupné z WWW: <http://www.k-3d.org/wiki/Main_Page>. [webová stránka]

[13] SALOME Platform [online]. 2005 [cit. 2010-11-05]. Dostupné z WWW: <http://www.salome-platform.org/>. [web]

[14] FreeCAD [online]. 2011 [cit. 2010-11-05]. Dostupné z WWW: <http://sourceforge.net/projects/free-cad/>.

[15] Wikipedie [online]. 2010 [cit. 2010-12-10]. Rovina - Wikipedie. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Rovina>. [webová stránka]

A Uživatelská příručka

A.1 Instalace aplikace

- Aplikace se instaluje na počítač klasicky, jako jakákoliv jiná. Stači pouze spustit instalaci a postupovat podle pokynů.
- Pro správnou funkci programu je třeba mít na PC naistalovaný Python, který je ke stažení na stránkách www.python.org.
- Alternativou je možnost zkopírovat pouze nově vytvořené skripty do určitých složek Blenderu. Uživatel pak bude mít přístupné všechny funkce, stejně jako kdyby postupoval podle předchozích kroků.

A.2 Ovládání aplikace

Aplikace se ovládá za pomoci nově vytvořených menu. Není tedy potřeba psát jakékoliv příkazy do ovládací konzole.

Import modelu galvana. Jako první krok je potřeba importovat si do programu model galvana pomocí importního skripu, který řešil ve své práci Zbyněk Hlava.

Definice galvana. Poté je třeba galvano označit a pomocí tlačítka *Define Galvano* ho uvést jako objekt, na kterém se budou provádět všechny operace.

Vkládání zářičů a detekce kolizí. Základními prvky jsou zářiče, které se vkládají pomocí menu Add > Mesh > IRE_P20. Alternativou je vkládání pomocí menu vyvolaného mezerníkem a po jeho vyvolání lze zářič vložit pomocí Add > IRE_P20. Kolize mezi zářiči se detekují stiskem tlačítka Collision BB, případně lze zapnout automatickou detekci kolizí pomocí tlačítka Check: True. Detekce lze opět zapnout i z menu vyvolaného po stisku mezerníku.

Ovládání funkcí GEOM a FUNC. Ovládací prvky pro tyto funkce se nacházejí v skupině menu s názvem RAD. Jejich vyvolání je možné pomocí tlačítek nesoucích jejich názvy a opět lze zapnout jejich autoamtické provádění. To lze ale vždy jen pro jednu z nich.

Další funkce. Poslední dvě tlačítka slouží k vytvoření výstupů z programu. Pomocí prvního z nich lze exportovat quaterniony a uložit je jako textový soubor, který bude dohledatelný v kořenovém adresáři Blenderu pod názvem quaterniony.

Poslední tlačítko Save view to image slouží k uložení aktuální obrazovky do souboru s koncovkou png. Ten je pak umístěn na disku C v adresáři tmp.

B Obsah přiloženého CD

[*Text*] - Elektronická verze diplomové práce.

[*BlenderLenam*] - Instalační balík výsledné aplikace.

[*Scripts*] - Veškeré skripty, které byly vytvořeny v této diplomové práci jsou umístěné ve složkách. Tyto složky nesou stejná jména, jako mají složky v Blenderu, ve kterých by měly být skripty umístěny.