VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PARAMETRICKÉ 3D MODELY

PARAMETRIC 3D MODELS

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE — Bc. MICHAL ONDREJÓ
AUTHOR

VEDOUCÍ PRÁCE — prof. Dr. Ing PAVEL ZEMČÍK
SUPERVISOR

BRNO 2020

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání diplomové práce

22905

Student: **Ondrejó Michal, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Parametrické 3D modely**
Parametric 3D Models
Kategorie: Počítačová grafika
Zadání:

1. Prostudujte dostupnou literaturu, dostupná řešení i software pro 3D modelování s parametrickým vstupem.
2. Navrhněte možnosti "geometrického provázání" objektů v parametrickém modelu a možnosti animace takového modelu v závislosti na nastavených parametrech. Prostudujte též možnost "interface" s uživatelským software.
3. Navrhněte způsob implementace systému včetně návaznosti na aplikační programy (může být i nadstavbou nad vhodným CAD systémem).
4. Implementujte a demonstrujte na vhodném příkladě, například na výsledcích simulace.
5. Zhodnoťte výsledky a diskutujte možnost další práce.

Literatura:

- Dle pokynů vedoucího

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Zemčík Pavel, prof. Dr. Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 6. listopadu 2019

Abstrakt

Cieľom tejto práce je navrhnúť možnosti previazania objektov v parametrickom modeli. Jednotlivé možnosti sú implementované v systéme na tvorbu parametrických trojrozmerných modelov. Systém umožňuje tvorbu modelov pomocou rôznych geometrických operácií, zmenu parametrov v ľubovoľnom čase, animovanie vytvoreného modelu a uloženie parametrického modelu v špeciálnom formáte, ktorý je čitateľný pre človeka. Navrhované riešenie bolo realizované a overené na príklade.

Abstract

The aim of this work is to propose possibilities of interconnection of objects in parametric model. Individual options are implemented in the parametric three-dimensional modeling system. This system allows the creation of models using various geometric operations, change parameters at any time, animate the created model, and save the parametric model in a human-readable format. The proposed solution was implemented and evaluated on simple example.

Kľúčové slová

parametrické modely, 3D, grafika, parametre

Keywords

parametric models, 3D, graphics, parameters

Citácia

ONDREJÓ, Michal. *Parametrické 3D modely.* Brno, 2020. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Dr. Ing Pavel Zemčík

Parametrické 3D modely

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána prof. Dr. Ing. Pavla Zemčíka. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michal Ondrejó
3. júna 2020

Poďakovanie

Týmto by som sa chcel poďakovať vedúcemu práce prof. Dr. Ing. Pavlovi Zemčíkovi za odbornú pomoc a nápady pri tvorbe tejto práce.

Obsah

1 Úvod **3**

2 Parametrické trojrozmerné modely **4**
2.1 Kategorizácia parametrických nástrojov 4
2.2 História parametrických modelov 5
2.3 Existujúce systémy pre tvorbu parametrických modelov 6
2.4 Existujúce knižnice na tvorbu trojrozmerných modelov 11

3 Geometrické objekty a operácie v 3D modelovaní **13**
3.1 Operácie vytvárajúce bod 13
3.2 Operácie vytvárajúce úsečky 19
3.3 Operácie vytvárajúce plošné objekty 22
3.4 Operácie vytvárajúce objemové objekty 24

4 Analýza a koncept práce **26**
4.1 Analýza súčasného stavu vo svete 26
4.2 Koncept práce 27
4.3 Technické parametre vytváraného softvéru 27
4.4 Návrh vytvárania geometrických objektov 28

5 Implementácia práce **33**
5.1 Bloková schéma 33
5.2 Skladanie operácií 34
5.3 Pridávanie a odoberanie objektov 35
5.4 Výrazy v operáciách 37
5.5 Grafické rozhranie 41

6 Vyhodnotenie vytvoreného riešenia **45**

7 Záver **52**

Literatúra **53**

A Obsah priloženého média **55**

B Rozhranie navrhnutého systému **56**

C Zoznam geometrických operácií **58**

D Množina operácii, tvoriace model hlavy. 62

E Operácie tvoriace model motora 64

Kapitola 1

Úvod

Objekty vo svete sú rôzne. Líšia sa tvarom aj veľkosťou. Každý si pod názvom nejakého objektu môže prestaviť trochu iný tvar a mohol by ho trochu inak vytvoriť. Na to slúži návrh objektu, ktorý obsahuje základné informácie o objekte. Na zdieľanie a uchovanie tohto návrhu je potrebné ho zaznamenať, zvyčajne zakresliť na papier pomocou ceruzky. Kedysi museli návrhári stráviť aj hodiny práce kreslením na papier a pri každej zmene tento návrh pracne prekresľovať. Počítače sa ale stále zdokonaľujú, a preto nie je prekvapením, že v dnešnej dobe nahradili ceruzky a papier, a stále viac pomáhajú s návrhom modelov.

Úlohou tejto práce je vytvoriť systém na tvorbu trojrozmerných parametrických modelov a prácu s nimi. Parametrické modelovanie využíva rôzne geometrické operácie na tvorbu modelov a umožňuje zmenu ich parametrov v ľubovoľnom čase.

Aplikácia by mala umožňovať prácu programátorského a grafického rozhrania. Programátorské rozhranie by malo slúžiť na využitie parametrických modelov v aplikáciach a na nastavovanie hodnôt parametrov modelu. Grafické rozhranie by malo umožniť návrhárovi jednoduchú tvorbu modelov, a tiež zobrazenie modelu a jeho animáciu v čase. Hodnota parametrov v parametrickom modeli sa musí dať ľubovoľnom čase meniť.

Následujúca kapitola sa zaoberá problematikou parametrických modelov. V tejto kapitole sú popísané súčasné systémy pre tvorbu parametrických modelov. V ďalšej, tretej kapitole sú opísané geometrické objekty a geometrické operácie. Tieto operácie sú rozdelené na štyri druhy, podľa toho aký typ objektu vytvárajú. Sú to bodové, úsečkové, plošné a objemové geometrické operácie. V štvrtej kapitole je opísaná analýza, koncept práce a návrh skladania parametrických modelov. Piata kapitola obsahuje implementačnú časť práce. Popis a postup vytvárania geometrických modelov pomocou skladania geometrických operácií a použitie rôznych možností pri zápise výrazu v parametroch geometrických operácií. V šiestej kapitole je popísané experimentálne vyhodnotenie vytvoreného riešenia. Posledná kapitola obsahuje zhrnutie práce a rôzne možnosti pre ďalšie rozšírenia a vylepšenia.

Kapitola 2

Parametrické trojrozmerné modely

Táto časť práce sa zaoberá parametrickými modelmi. Obsahuje zhrnutie súčasného stavu problematiky parametrických trojrozmerných modelov.

2.1 Kategorizácia parametrických nástrojov

Typickým návrhovým médiom je ceruzka a papier. Presnejšie je to ceruzka, guma a papier. Ceruzka pridáva a guma odoberá. Po pridaní niekoľkých nástrojov, ako pravítko, kružidlo a uhlomer sa kresby stanú presnejšími a precíznejšími modelmi navrhovanej idey. Dizajnéri tieto značky pridávajú, odoberajú a spájajú ich [17].

Konvenčné návrhové systémy fungujú práve na tomto princípe. Parametrické modelovanie predstavuje zásadnú zmenu. Značky, ktoré sú základom návrhu spolu súvisia a vzájomne menia svoju pozíciu. Dizajnéri už nemusia iba pridávať a odoberať, ale môžu vytvárať medzi bodmi vzťahy a upravovať navrhnutý model [17].

Modelovanie sa rozdeľuje na tri druhy [14]:

Povrchové modelovanie (Surface) Tento typ modelovania je založený na systéme NURBS (Non-uniform rational basis spline). Je to technika, ktorá umožňuje prirodzenejšie tvary z kriviek. Je to vynikajúca metóda pre hladké tvary, ako sú karosérie automobilov alebo lopatky turbín.

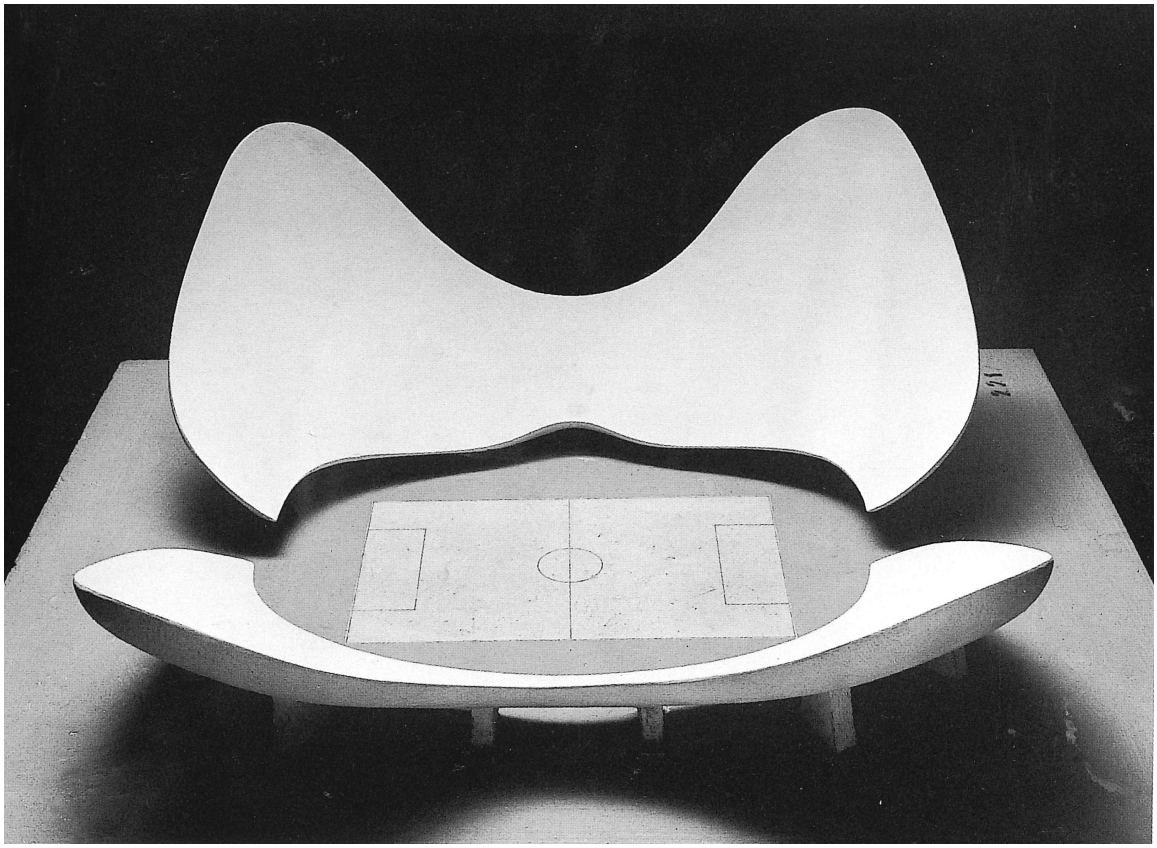
Parametrické modelovanie (Parametric) Najpoužívanejší typ modelovania u profesionálnych dizajnérov, kvôli potrebe vysokej presnosti mechanických častí. Princípom je definovať parametre komponentov a rôznych funkcií. Komponenty sú potom parametricky riadené a ľahko upravované vďaka stromu histórie. Umožňujú matematicky definovať vzťahy pre parametre, čo umožňuje zmenu konštrukcie objektu pomocou zmeny niekoľkých hodnôt.

Priame modelovanie (Direct) Je jednoduchšie ako parametrické modelovanie. Odstránenie alebo presunutie niektorej časti modelu je jednoduché, stačí s ňou pohnúť. Narozdiel od parametrického modelovania, nie je reprezentovaný stromom operácií, a teda sa netreba obávať, že sa celý model rozbije.

2.2 História parametrických modelov

Pôvod slova *parameter* pochádza z matematiky. Popisuje matematickú metódu, ktorá používa nezávislé premenné, zvané parametre. Debatuje sa ale ohľadom toho, kedy začali dizajnéri používať toto slovo [3]. David Gerber v svojej doktorskej práci *Parametric Practice* (2007) pripisoval Maurice Reiter prvé použitie tohto slova na papieri s názvom *Parametric Design* z roku 1988. V roku 1988 bol tiež vydaný prvý komerčne úspešný softvér pre parametrické modelovanie Pro/ENGINEER firmou Parametric Technology Corporation založenou matematikom Samuelom Geisbergom v roku 1985.

Robert Stiles ale dokázal, že skutočný pôvod tohto slova bol už o niekoľko dekád skôr pomocou zápisov architekta Luigi Morettiho z rokov 1940. Moretti písal o parametrickej architektúre, ktorú definoval ako architektonické systémy s cieľom definovať vzťahy medzi rozmermi závislými od rôznych parametrov. Moretti v roku 1960 použil ako príklad dizajn štadióna 2.1, kde vysvetlil, ako sa môže štadión meniť pomocou devätnástich parametrov, ktoré zahŕňali pozorovacie uhly a cenové náklady na betón. Tento model vytvoril už za pomoci počítača 610 od spoločnosti IBM[5]. O pár rokov neskôr Moretti navrhol Watergate Complex 2.2, ktorý je prvou veľkou stavebnou prácou, na ktorú boli využité počítače [3].



Obr. 2.1: Model štadióna N od Luigi Moretti - parametrický model pozostáva z devätnástich parametrov (tento model bol vystavený na výstave parametrickej architektúry v Twelfth Milan Triennial v roku 1960) [3]

Obr. 2.2: Watergate Complex navrhnutý Luigi Morettim - prvý veľký stavebný projekt, na ktorý boli využité parametrické modely [1]

2.3 Existujúce systémy pre tvorbu parametrických modelov

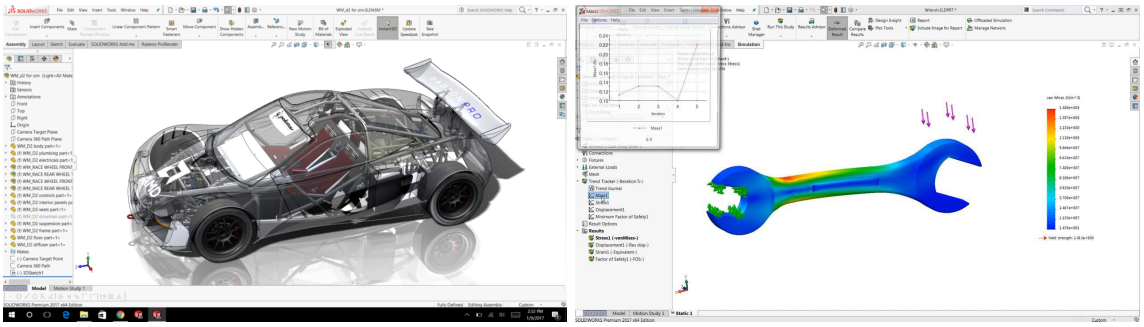
Existuje niekoľko systémov pre tvorbu parametrických modelov. Zväčša sú platené, ale sú aj také, ktoré umožňujú vytváranie modelov zdarma. Väčšinou tieto parametrické modelovacie systémy nie sú navrhnuté pre domáceho používateľa ale skôr pre profesionálnych 3D dizajnérov.

Solidworks

Solidworks je jedným z najlepších softvérov na tvorbu mechanických častí. Tento parametrický softvér je profesionálny nástroj pre inžinierov a dizajnérov. Umožňuje vykonať nad

[1] Google Earth, earth.google.com/web/

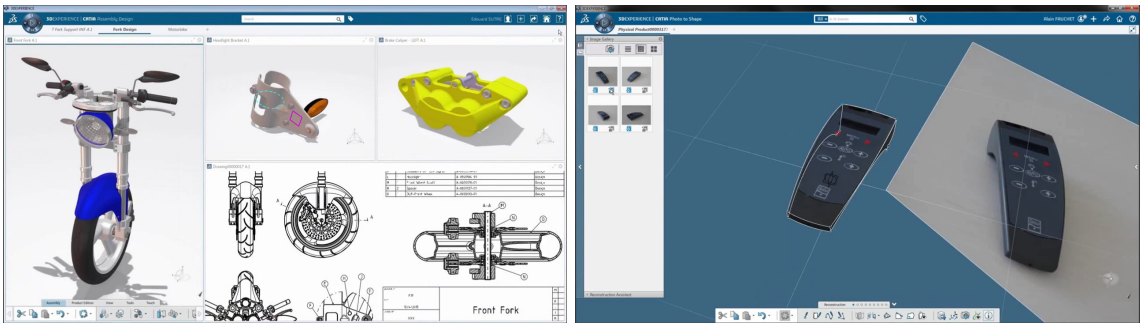
vytvoreným modelom aj rôzne simulácie, napríklad záťažové alebo tepelné ako je vidieť na obrázku 2.3 [6].



Obr. 2.3: Prostredie programu Solidworks - vpravo je záťažová simulácia [2]

Catia

Tento softvér je náročnejší pre menej zdatných užívateľov. Je to komplexný nástroj určený hlavne pre profesionálov a umožňuje množstvo pokročilejších nástrojov, ako je prevedenie dvojrozmerného obrázku do trojrozmerného modelu alebo generovanie organických tvarov, ktoré spĺňajú pevnostné požiadavky, ale sú od pôvodného modelu značne odľahčené [6].

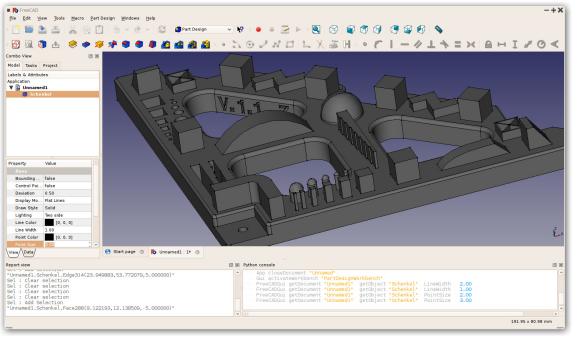


Obr. 2.4: Prostredie programu Catia -vpravo je ukážka prevodu z 2D obrázku do 3D modelu[3]

[2] https://blogs.solidworks.com/solidworksblog/2017/02/solidworks-world-partner-pavilion-preview-part-v.html
https://hawkridgesys.com/blog/solidworks-simulation-package-right
[3] https://www.youtube.com/watch?v=1NYHYrbqPZI

FreeCAD

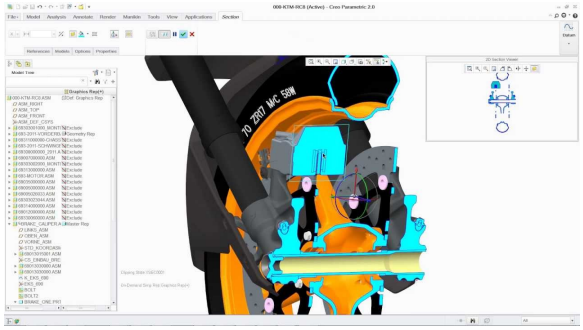
FreeCad je voľne dostupný softvér s intuitívnym grafickým rozhraním. Umožňuje množstvo podobných nástrojov ako Catia a SolidWorks. Je zameraný na strojárstvo a návrh výrobku [7].



Obr. 2.5: Prostredie programu FreeCAD [7]

Creo Parametric

Jedná sa o softvér pre priemyselný dizajn. Umožňuje vytvárať zložité trojrozmerné modely. Poskytuje veľa účinných nástrojov prispôsobených priemyselnému výrobnému prostrediu. Tento softvér sa často používa napríklad v automobilovom priemysle [6].



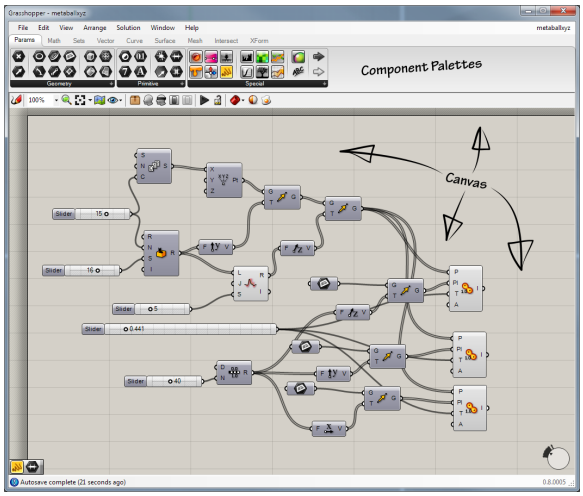
Obr. 2.6: Prostredie programu Creo Parametric [4]

Rhino s plugiom Grasshopper

Rhino je profesionálny 3D CAD softvér používaný v množstve spoločností. Pre prácu s parametrickými modelmi je potrebný plugin Grasshopper [6].

[4] https://www.youtube.com/watch?v=kQYCmq8EmNk

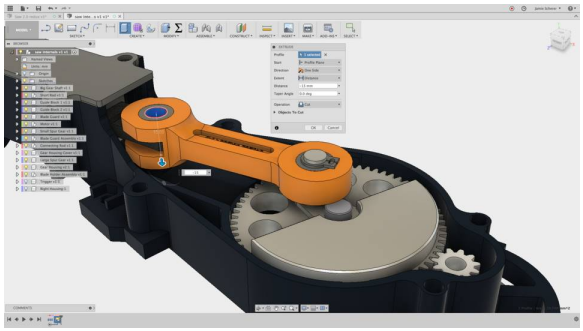
Grasshopper je vizuálny programovací jazyk, znázornený na obrázku 2.7. Umožňuje tvorbu parametrických modelov pre konštrukčné inžinierstvo, architektúru a výrobu.



Obr. 2.7: Prostredie programu Grasshopper [5]

Fusion 360

Fusion 360 nie je iba parametrický modelovací systém. Podporuje aj priame modelovanie a umožňuje prechod medzi týmito typmi modelovania. Keďže pri priamom modelovaní nie je model vytváraný pomocou stromu operácií, je tento strom pri prenose odstránený. Má dobré simulačné a modelovacie nástroje, ktoré pomáhajú pri návrhu [6].

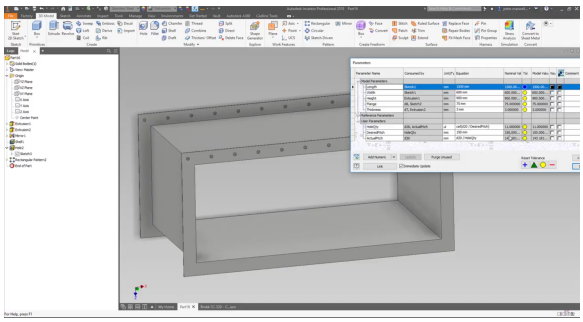


Obr. 2.8: Prostredie systému Fusion 360 [6]

[5] https://en.wikipedia.org/wiki/Grasshopper_3D

Inventor

Inventor je rovnako ako Fusion 360 vytvorený spoločnosťou Autodesk. Tento program je často používaný v strojárstve. Inventor ponúka množstvo parametrických možností, ktoré pomáhajú vytvárať 3D modely a hlavne mechanické konštrukcie [6].



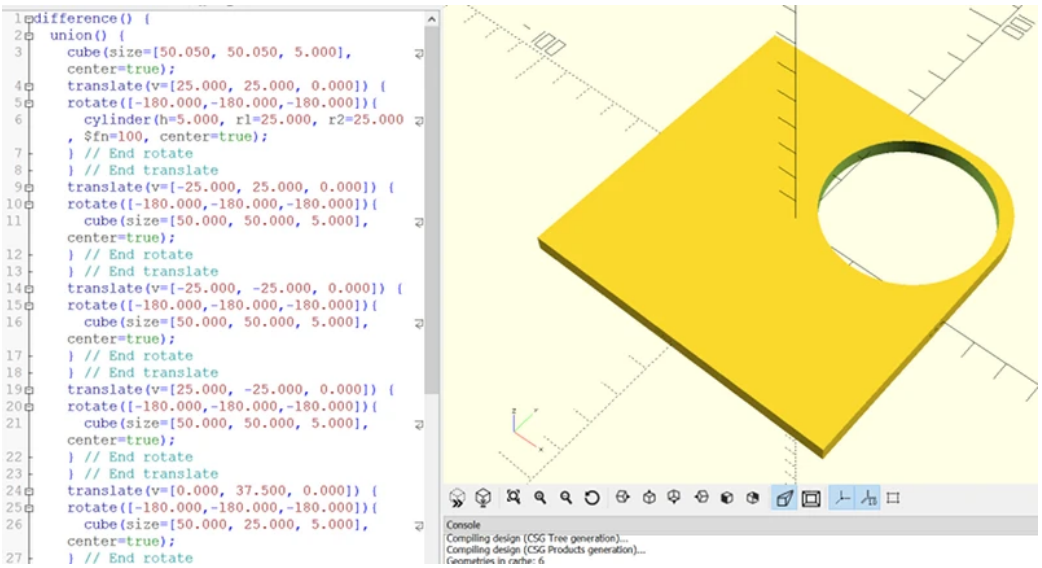
Obr. 2.9: Prostredie systému Inventor [6]

2.4 Existujúce knižnice na tvorbu trojrozmerných modelov

Existuje niekoľko knižníc, ktoré umožňujú vytváranie trojrozmerných modelov. V tejto časti je opísaných niekoľko z nich.

OpenSCAD

OpenSCAD je softvér na vytváranie solídnych trojrozmerných objektov. Je to bezplatný a multiplatformový softvér. Tento nástroj nie je vhodný na vytváranie veľmi komplikovaných objektov, aké umožňujú vytvárať vyššie spomenuté systémy, ale je vhodný na tvorbu trojrozmerných modelov častí strojov. OpenSCAD nie je interaktívny modelovací systém. Skôr je to 3D-kompilátor, ktorý číta skript opisujúci objekt a tento trojrozmerný model vykresľuje. Týmto spôsobom má dizajnér úplnú kontrolu nad modelovacím procesom a umožňuje jeho jednoduchú úpravu. Jazyk, ktorý OpenSCAD využíva na tvorbu modelov je podobný rodine jazykov C/Java/Python [8].



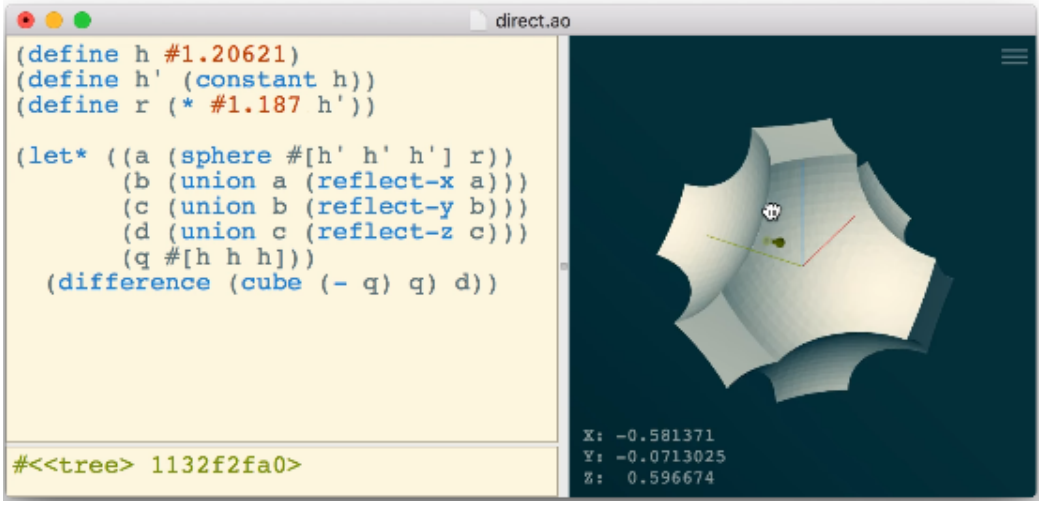
Obr. 2.10: Prostredie OpenSCAD [7]

Libfive

Libfive je knižnica a sada nástrojov na modelovanie pevných látok, zvlášť vhodná na parametrické a procedurálne navrhovanie. Tvorba modelov je podobná ako pri knižnici OpenSCAD. Jazyk, ktorý táto knižnica používa na modelovanie objektov má prefixovú notáciu,

[6] https://www.youtube.com/watch?v=KAeaonxJ5Es
[7] cite https://www.openscad.org/documentation.html

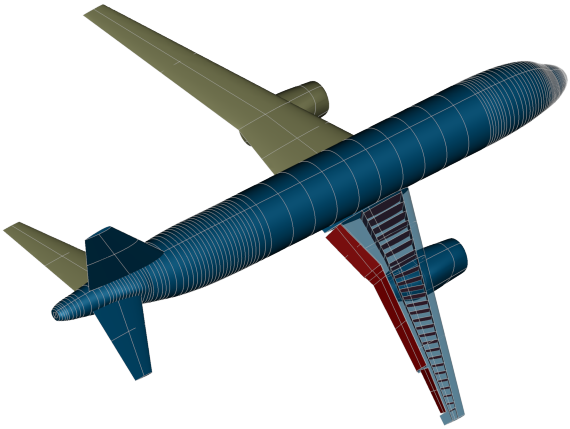
čím sa podobá jazyku lisp. Rovnako ako OpenSCAD je Libfive bezplatná knižnica. Na zobrazovanie modelov využíva aplikáciu Studio, ktorá je zobrazená na obrázku 2.11 [10].



Obr. 2.11: Aplikácia Studio, používajúca knižnicu libfive [10]

Tigl

TiGL je knižnica, ktorá je určená hlavne pre geometrické modelovanie lietadiel a helikoptér. Je to spojenie medzi parametrami CPACS lietadla a simulačnými nástrojmi. CPACS (The Common Parametric Aircraft Configuration Schema) je definícia údajov pre systém leteckej dopravy. TiGL počíta CAD modely z týchto parametrov. Tvorba modelu v knižnici Tigl je založená hlavne na vytváraní plôch pomocou kriviek NURBS. [15]



Obr. 2.12: Model vytvorený pomocou knižnice Tigl [15]

Kapitola 3

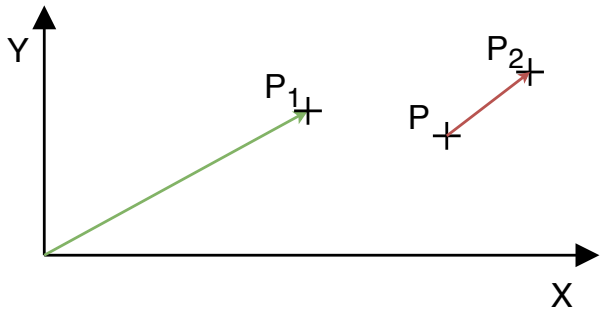
Geometrické objekty a operácie v 3D modelovaní

Táto kapitola je súčasťou zhrnutia súčasného stavu. Sú v nej zhrnuté matematické a geometrické vlastnosti objektov a operácií ktorými sa práca zaoberá.

Geometrické objekty môžeme rozdeliť do 4 kategórií podľa dimenzií. Body, úsečky, plochy a objemové objekty. Spájaním objektov z nižších dimenzií, vznikajú zložitejšie objekty z vyšších dimenzií. Táto kapitola opisuje jednotlivé geometrické objekty a operácie, ktoré ich vytvárajú, a to od jednoduchších až k zložitejším.

3.1 Operácie vytvárajúce bod

Bod je základnou stavebnou jednotkou všetkých objektov. Všetky geometrické útvary sa dajú definovať ako množina bodov. Je to bezrozmerný geometrický útvar, teda nemá šírku, výšku ani hrúbku. Jeho úlohou je označiť pozíciu v priestore. Pozíciu bodu v trojrozmernom priestore udáva vzdialenosť na jednotlivých osiach ortogonálneho súradnicového systému X, Y a Z. Táto pozícia môže byť v absolútnom tvare, teda od stredu súradnicového systému alebo v relatívnom tvare, kedy je závislá na pozícii iného bodu.

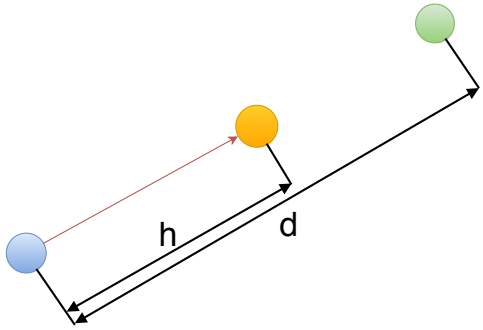


Obr. 3.1: Bod P_1 s absolútnou pozíciou a bod P_2 s relatívnou pozíciou od bodu P

Lineárna interpolácia

Lineárna interpolácia umožňuje získať bod, ktorý je na rovnakej priamke ako dva zadané body. Na obrázku 3.2 sú tieto body zobrazené modrou a zelenou farbou, výsledný bod

je zobrazený žltou farbou. Pozícia bodu závisí od zadanej vzdialenosti od počiatočného bodu. Táto vzdialenosť môže byť zadaná dĺžkou alebo percentuálne, kde 50% vytvorí bod uprostred počiatočného a koncového bodu.



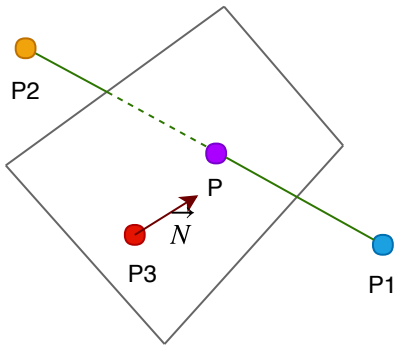
Obr. 3.2: Lineárna interpolácia medzi bodmi, kde `d` označuje vzdialenosť medzi zadanými bodmi a `h` označuje vzdialenosť v akej sa má vykonať interpolácia

Ak je zadaná vzdialenosť pomocou dĺžky, použije sa vzorec 3.1. Pri percentuálnej vzdialenosti sa použije obdobný vzorec, ale vektor medzi bodmi bod1 a bod2 sa nenormalizuje [18].

$$bod = bod1 + norm(bod2 - bod1) * vzdialenosť; \tag{3.1}$$

Priesečník plochy a úsečky

Pri tejto operácii sa používa ľubovolný plošný objekt ako rovina a úsečka ako priamka. Táto geometrická operácia vytvorí bod v mieste, kde sa priamka pretína s rovinou [2].



Obr. 3.3: Pretnutie plochy priamkou

Rovnica pre rovinu, ktorá je tvorená bodom $\mathtt{P}_3$ nachádzajúcom sa na rovine a normálou N, sa dá zapísať ako 3.2.

$$\mathbf{N} \cdot (\mathbf{P} - \mathbf{P3}) = 0 \tag{3.2}$$

Rovnica priamky 3.3, ktorá je určená bodmi $\mathtt{P}_1$ a $\mathtt{P}_2$

$$\mathbf{P} = \mathbf{P1} + u(\mathbf{P2} - \mathbf{P1}) \tag{3.3}$$

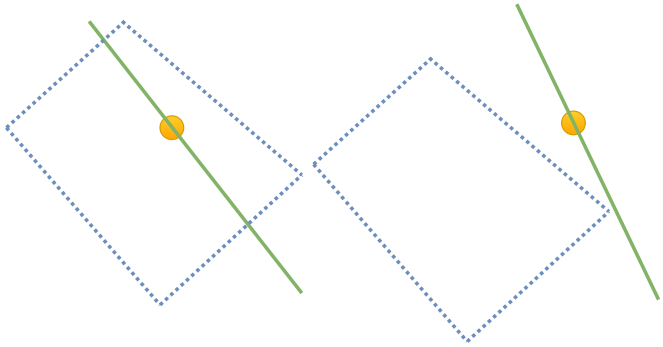
Bod `P` označuje priesečník medzi rovinou a priamkou. Pomocou substitúcie získame rovnicu 3.4.

$$\mathbf{N} \cdot (\mathbf{P1} + u(\mathbf{P2} - \mathbf{P1}))) = \mathbf{N} \cdot \mathbf{P3} \tag{3.4}$$

Po vyriešení tejto rovnice dostaneme rovnicu 3.5. Výslednú pozíciu bodu dostaneme dosadením u do rovnice pre priamku 3.3.

$$u = \frac{\mathbf{N} \cdot (\mathbf{P3} - \mathbf{P1})}{\mathbf{N} \cdot (\mathbf{P2} - \mathbf{P1})} \tag{3.5}$$

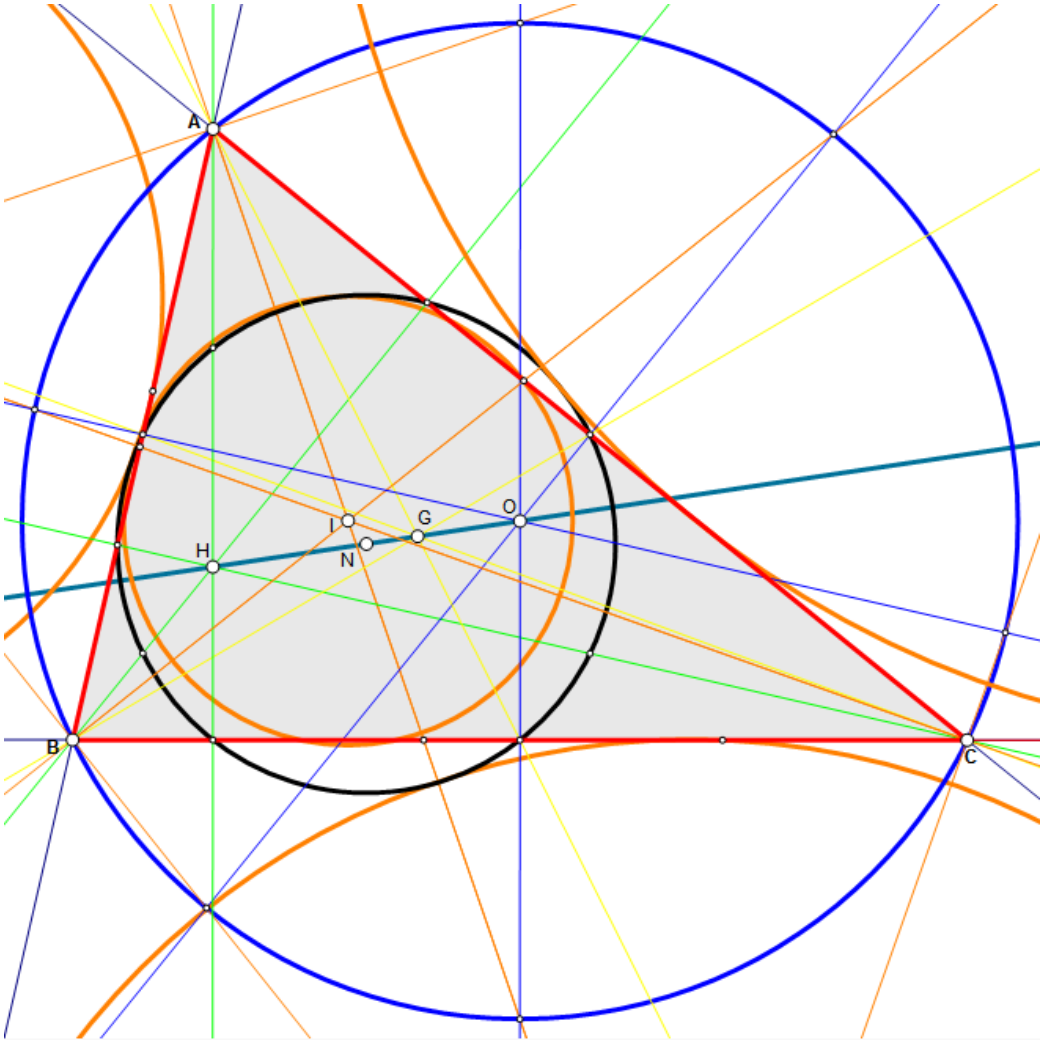
Ako je vidieť na obrázku 3.4, pomocou tejto operácie sa vytvorí bod aj mimo zadaných objektov. Problém nastáva, ak je zadaná úsečka paralelná s plochou, a teda je kolmá na normálu plochy N. Skalárny súčin v menovateli je potom rovný 0, teda úsečka je s plochou rovnobežná. V tomto prípade priesečník buď neexistuje, alebo je priesečníkov nekonečne veľa, ak úsečka leží na rovine [2].



Obr. 3.4: Priesečník plochy a priamky, vpravo je zobrazený prípad, ak sa priesečník nachádza na rovine, ale je mimo plochy objektu

Stred trojuholníka

Aj pre trojuholník existuje množstvo typov stredu. V súčastnosti je podľa encyklopédie stredov trojuholníkov známych až 30 714 trojuholníkových centier [12]. Tento počet každým dňom narastá. Pre porovnanie, v roku 1994 bolo známych 101, v roku 1998 bolo 360 a v decembri 2004 bolo známych 3053 [16].



Obr. 3.5: Najznámejšie stredy trojuholníka sú ťažisko (G), ortocentrum (H), stred vpísanej kružnice (I), opísanej kružnice (O) aj stred kružnice deviatich bodov (N) [1]

Medzi najznámejšie patria ťažisko, ortocentrum, stred vpísanej kružnice, opísanej kružnice aj stred kružnice deviatich bodov. Tieto body sú zaznačené na obrázku 3.5.

Ťažisko (angl. Centroid)

Ťažisko trojuholníka sa nachádza v priesečníku troch mediánov trojuholníka. Na nájdenie jeho pozície stačí vypočítať aritmetický priemer vrcholov trojuholníka v jednotlivých osiach 3.6 [1].

$$\frac{A+B+C}{3} \tag{3.6}$$

[1] https://en.wikipedia.org/wiki/Triangle_center

Stred vpísanej kružnice (angl. Incenter)

Z každého bodu trojuholníka urobíme priamku tak, aby uhly na oboch stranách priamky boli rovnaké. Táto priamka sa nazýva tiež bisektor [1]. Stred vpísanej kružnice je na priesečníku týchto priamok.

Stred vpísanej kružnice sa dá vypočítať aj pomocou vzorca 3.7, kde A, B, C sú vrcholy trojuholníka a a, b, c sú dĺžky strán protiľahlých k vrcholom A, B, C [13].

$$O = \frac{a * A + b * B + c * C}{a + b + c} \tag{3.7}$$

Stred opísanej kružnice (angl. Circumcenter)

U jednotlivých strán trojuholníka zistíme stred a z tohto bodu urobíme kolmice. Tam kde sa tieto kolmice stretnú, vznikne stred vpísanej kružnice. Veľkosť kružnice je vzdialenosť od stredu k ľubovoľnému vrcholu trojuholníka. Táto vzdialenosť je pre všetky vrcholy rovnaká [11].

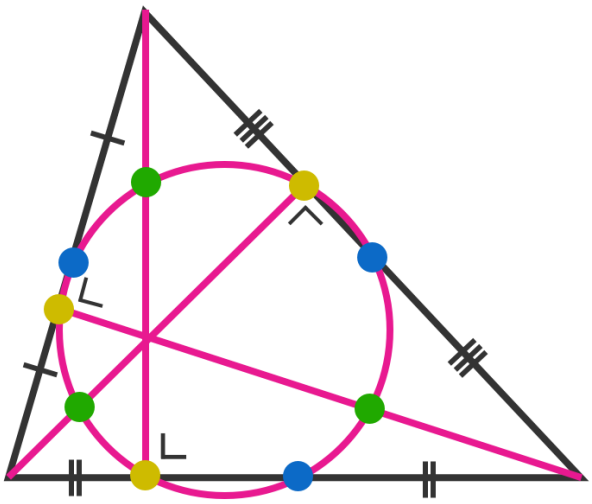
Ortocentrum (angl. Orthocenter)

Ortocentrum sa nachádza na priesečníku kolmíc, ktoré prechádzajú cez protiľahlý vrchol. Tieto kolmice sa tiež nazývajú výškou trojuholníka. Ak je trojuholník tupý, ortocentrum sa nachádza mimo trojuholníka, ak je trojuholník v niektorom vrchole kolmý, nachádza sa v takomto vrchole aj ortocentrum trojuholníka. Pre získanie pozície ortocentra zistíme aspoň dve kolmice pomocou operácie 3.2. Pozícia ortocentra sa nachádza v mieste, kde sa tieto kolmice pretínajú [11].

Stred kružnice deviatich bodov (angl. NinePointCenter)

Kružnica deviatich bodov, tiež známa ako Feuefbachova kružnica, po nemeckom matematikovi Karl Wilhelm Feuerbach, ktorý ako prvý dokázal, že sa kružnica deviatich bodov dotýka vpísanej kružnice a pripísaných kružníc [4].

Teorém 1 ([4] Teorém kružnice deviatich bodov) *Nech ABC je všeobecný trojuholník, P,Q,R nech sú päty jeho výšok, K,L,M nech sú stredy jeho strán, O nech je priesečník výšok a T,U,V nech sú postupne stredy úsečiek AO,BO,CO. Potom 9 bodov P, Q, R, K, L, M, T, U, V leží na jednej (tzv. Feuerbachovej) kružnici.*



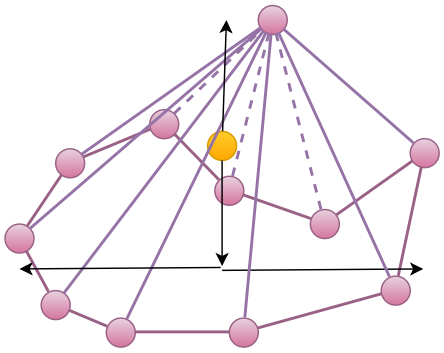
Obr. 3.6: Kružnica deviatich bodov - modré body označujú stredy strán, žlté body označujú päty výšok a zelené sú stredy medzi vrcholmi a priesečníkom výšok [2]

Stred objektu

Rovnako ako pri dvojrozmerných objektoch, je aj u trojrozmerných objektoch viacero variánt získania stredu objektu. Zvolil som dve metódy, a to metódu ohraničujúceho kvádra a metódu priemerného stredu všetkých bodov.

Stred pomocou ohraničujúceho kvádra

Pri tejto metóde sa prejdú všetky body a zoberie sa maximálna a minimálna hodnota v osiach X, Y a Z. Takto dostaneme ohraničujúci kváder (Bounding box) a ako výsledný bod sa zoberie stred tohto kvádra.



Obr. 3.7: Stred ohraničujúceho kvádra

[2] https://brilliant.org/wiki/nine-point-circle/

3.2 Operácie vytvárajúce úsečky

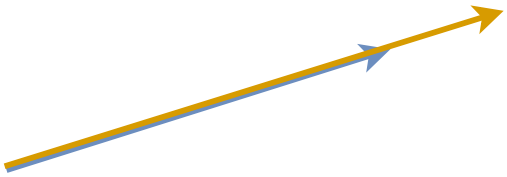
Úsečka je časť priamky medzi dvoma koncovými bodmi. Aby bolo možné používať v geometrických operáciách smerový vektor z úsečky, môžeme tieto body označiť ako počiatočný a koncový, ako je zobrazené na obrázku 3.8 [18].

$$P = P1 + u(P2 - P1) \tag{3.8}$$

Obr. 3.8: Zobrazenie úsečky s počiatočným a koncovým bodom

Zmena dĺžky úsečky

Vytvorenie úsečky so zadanou dĺžkou. Dĺžka môže byť zadaná vzdialenosťou alebo percentuálne od veľkosti zadanej úsečky. Výsledná úsečka je v rovnakom smere a začína v rovnakom bode ako zadaná úsečka. V prípade, ak veľkosť výslednej úsečky sa rovná jedna, táto geometrická operácia sa nazýva aj normalizácia [18].



Obr. 3.9: Zmena veľkosti úsečky

Najkratšia úsečka medzi bodom a priamkou

Priamka je definovaná dvoma bodmi, bodom $P1(x1 - y1)$ a bodom $P2(x2, y2)$. Rovnica priamky je 3.9.

$$P = P1 + u(P2 - P1) \tag{3.9}$$

Na získanie najkratšej vzdialenosti je potrebné nájsť kolmicu od bodu $P3(x3, y3)$ na priamku. Z toho vyplýva, že skalárny súčin medzi nimi musí byť rovný 0, tento vzťah je vidieť v rovnici 3.10.

$$(P3 - P) \cdot (P2 - P1) = 0 \tag{3.10}$$

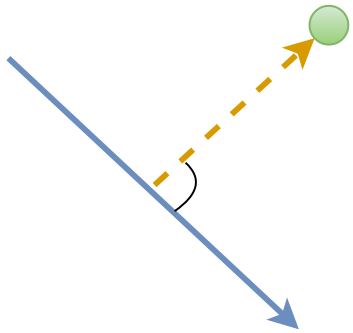
Bod P označuje najbližší bod na priamke k bodu $P3$. Substitúciou rovnice 3.9 do 3.10 získame rovnicu 3.11.

$$[P3 - P1 - u(P2 - P1)] \cdot (P2 - P1) = 0 \tag{3.11}$$

Vyriešením tejto rovnice získame rovnicu 3.12, ktorú následne môžeme dosadiť do rovnice pre priamku 3.9 a získať pozíciu bodu P. Ak by sme chceli, aby sa bod vytváral iba na úsečke a nie mimo nej, bolo by potrebné testovať vzdialenosť u, či je v rozmedzí (0,1). Ak je hodnota u záporná alebo väčšia ako jedna, najbližší bod na priamke k bodu $P3$ sa nachádza mimo zadanej úsečky.

$$u = \frac{(x3 - x1)(x2 - x1) + (y3 - y1)(y2 - y1) + (z3 - z1)(z2 - z1)}{\|P2 - P1\|^2} \tag{3.12}$$

Výsledná úsečka je tvorená bodmi P a $P3$ [2].



Obr. 3.10: Najkratšia vzdialenosť medzi úsečkou a bodom

Najkratšia úsečka medzi dvomi priamkami

Keďže v trojrozmernom priestore často nenastáva pretnutie dvoch úsečiek v jednom bode, je práve najkratšia úsečka medzi dvoma priamkami používaná ako priesečník priamok v trojrozmernom priestore [2].

Hľadáme najkratšiu úsečku s bodmi P_a a P_b, kde P_a leží na priamke definovanou bodmi P_1 a P_2, a P_b leží na priamke, ktorá je definovaná bodmi P_3 a P_4.

Pre bod P_a môžeme napísať rovnicu 3.13.

$$P_a = P_1 + u_a(P_2 - P_1) \tag{3.13}$$

Podobne pre bod P_b rovnicu 3.14.

$$P_b = P_3 + u_b(P_4 - P_3) \tag{3.14}$$

Pri hľadaní najkratšej úsečky medzi týmito priamkami si stačí uvedomiť, že najkratšia úsečka bude na tieto priamky kolmá, čo nám umožňuje zapísať následovné rovnice 3.15.

$$\begin{aligned}(P_a - P_b) \cdot (P_2 - P_1) = 0 \\ (P_a - P_b) \cdot (P_4 - P_3) = 0\end{aligned} \tag{3.15}$$

Po doplnení P_a a P_b do týchto rovníc získame rovnice 3.16.

$$\begin{aligned}((P_1 + u_a(P_2 - P_1)) - (P_3 + u_b(P_4 - P_3))) \cdot (P_2 - P_1) = 0 \\ ((P_1 + u_a(P_2 - P_1)) - (P_3 + u_b(P_4 - P_3))) \cdot (P_4 - P_3) = 0\end{aligned} \tag{3.16}$$

Keďže by tieto rovnice boli veľmi rozsiahle, je vhodné si pre riešenie tejto rovnice definovať nasledovnú substitúciu 3.17.

$$d_{mnop} = (x_m - x_n)(x_o - x_p) + (y_m - y_n)(y_o - y_p) + (z_m - z_n)(z_o - z_p) \tag{3.17}$$

Pomocou tejto substitúcie sa dajú rovnice 3.16 zapísať nasledovne 3.18.

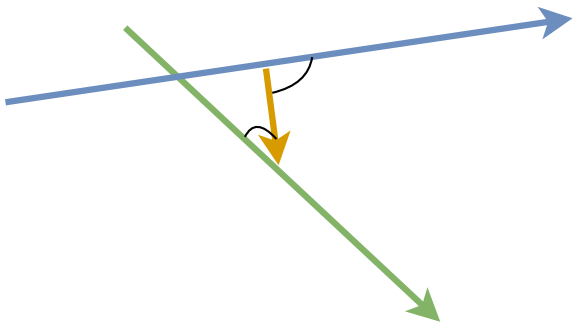
$$\begin{aligned}d_{1321} + u_a d_{2121} - u_b d_{4321} = 0 \\ d_{1343} + u_a d_{2143} - u_b d_{4343} = 0\end{aligned} \tag{3.18}$$

Vyriešením týchto rovníc pre u_a získame rovnicu 3.19.

$$u_a = \frac{d_{1343}d_{4321} - d_{1321}d_{4343}}{d_{2121}d_{4343} - d_{2143}d_{2143}} \tag{3.19}$$

Keď už poznáme u_a, pomocou dosadenia do rovnice 3.20 získame u_b.

$$u_b = \frac{d_{1321} + u_a d_{2121}}{d_{4321}} \tag{3.20}$$



Obr. 3.11: Najkratšia vzdialenosť medzi dvomi úsečkami

Najkratšia úsečka medzi plochou a bodom

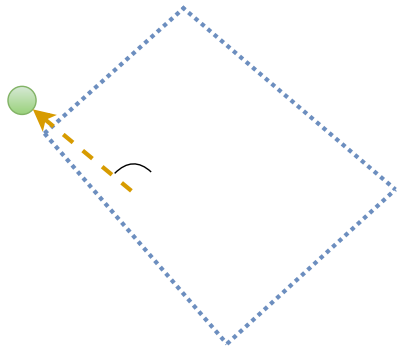
Rovina je definovaná pomocou normály $\overrightarrow{N} = (A, B, C)$ a bodom na nej ležiacom $P_a = (x_a, y_a, z_a)$. Hľadaná úsečka má rovnaký smer ako normála plochy. Vzdialenosť medzi touto rovinou a bodom $P_b = (x_b, y_b, z_b)$ dostaneme pomocou vzorca 3.21, kde premietneme vektor medzi bodom P_b a bodom P_a na normálu roviny $\overrightarrow{N}$ pomocou skalárneho súčinu.

$$distance = (P_b - P_a) \cdot \overrightarrow{N} \tag{3.21}$$

Bod P získame vynásobením vektora $\overrightarrow{N}$ touto vzdialenosťou a následným odčítaním od bodu P_b 3.22.

$$P = P_b - (\overrightarrow{N} * distance) \tag{3.22}$$

Výsledná úsečka má počiatočný bod na ploche a koncový bod P_b [2].

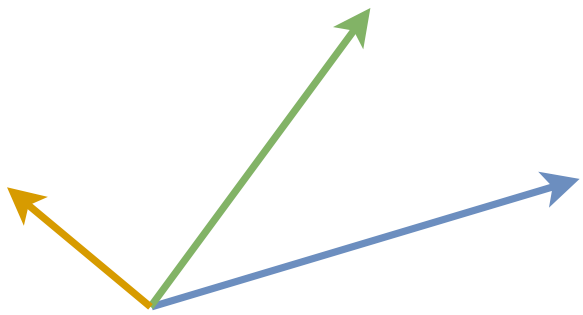


Obr. 3.12: Najkratšia vzdialenosť medzi plochou a bodom

Vektorový súčin

Vektorový súčin (cross product) vytvára úsečku, ktorá je kolmá na dané úsečky (priamky). Zadané úsečky sa najprv prevedú na vektory (*koncový_bod* − *začiatočný_bod*). Veľkosť úsečky závisí od veľkosti zadaných úsečiek a od uhla, ktorý zvierajú [18].

$$L_1 \times L_2 = \{y_{l1}z_{l2} + y_{l2}z_{l1}, z_{l1}x_{l2} + z_{l2}x_{l1}, x_{l1}y_{l2} + x_{l2}y_{l1}\} \tag{3.23}$$



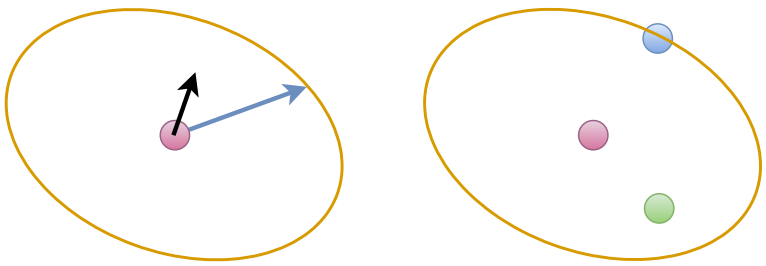
Obr. 3.13: Vektorový súčin

Táto operácia sa často používa aj v počítačovej grafike, kde sa používa pre výpočet normály plošných útvarov. Výsledný vektor sa musí normalizovať, teda vydeliť jeho dĺžkou.

3.3 Operácie vytvárajúce plošné objekty

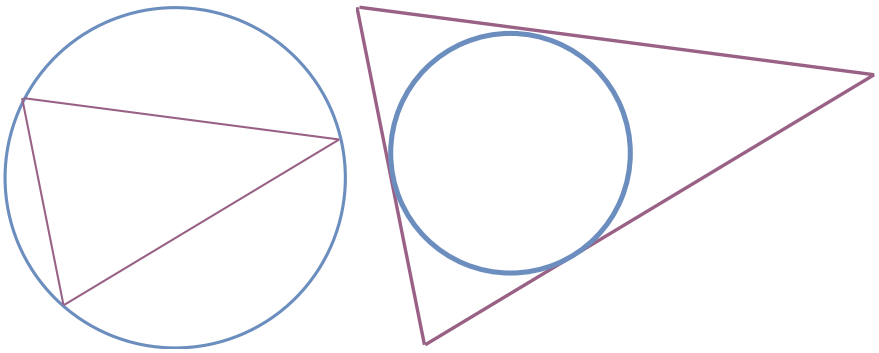
Kruh

Je viac metód ako zadávať kruh. Základným spôsobom pre vytvorenie kruhu je zadanie stredového bodu a priemeru. Keďže chceme aby bol kruh v trojrozmernom priestore, je potrebné zadať aj normálu . Ďalším spôsobom je zadanie troch bodov, kde jeden z bodov označuje stred kruhu, druhý bod leží na obvode kruhu a tretí bod ležiaci na ploche kruhu tak, aby nebol s ostatnými bodmi na jednej priamke, označuje natočenie kruhu [18].



Obr. 3.14: Vľavo je kruh pomocou priemeru a normály a v pravo je kruh tvorený troma bodmi, kde jeden bod udáva stred a pomocou ďalších dvoch bodov sa určí priemer a smer normály

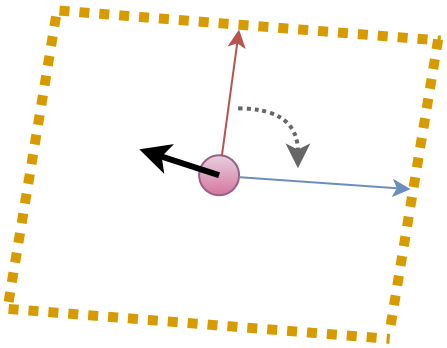
Alternatívne môžeme vyjadriť kruh pomocou opísanej (Circumscribed) alebo vpísanej (Inscribed) kružnice trojuholníka (obrázok 3.15). Postup pre nájdenie stredu opísanej a vpísanej kružnice trojuholníka je v časti 3.1.



Obr. 3.15: Opísaná a vpísaná kružnica trojuholníka

Obdĺžnik

Často používaným geometrickým objektom v modelovaní je aj obdĺžnik. Základnými parametrami obdĺžnikov je výška a šírka. Ďalším parametrom je často aj natočenie okolo normály. Keďže sa zaoberáme objektami v trojrozmernom priestore, obdĺžnik musí definovať aj normála, ako je vidieť na obrázku 3.16.



Obr. 3.16: Obdĺžnik

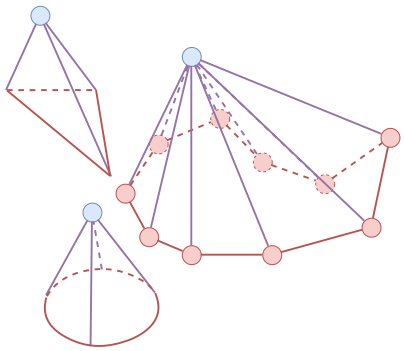
Polygón

Keďže v reálnom svete nie sú základom objektov len dokonalé objekty ako kruh, trojuholník alebo štvorec, zvolil som pre túto prácu aj tvorbu polygónov. Polygóny pozostávajú z množiny bodov a množinou hrán, ktoré tieto body spájajú a vytvárajú obvod polygónu. Keďže polygón je plošný útvar, musí byť tvorený minimálne 3 bodmi. Táto operácia má ako jediná premenlivý počet bodov[18].

Všetky primitívne objekty, ako štvorec, kruh alebo trojuholník, sú konvexné, teda všetky vnútorné uhly sú menšie alebo rovné 90 stupňov. Proces triangulácie je v tomto prípade veľmi jednoduchý. Stačí spojiť jeden ľubovoľný bod so všetkými ostatnými bodmi. Pri zadávaní polygónov pomocou bodov sa ale môže stať, že niektorý vnútorný uhol polygónu bude tupý, teda väčší ako 90 stupňov. Klasická metóda triangulácie teda nepostačuje a je potrebné vybrať niektorú inú metódu, napríklad *Ear clipping method*[18].

3.4 Operácie vytvárajúce objemové objekty

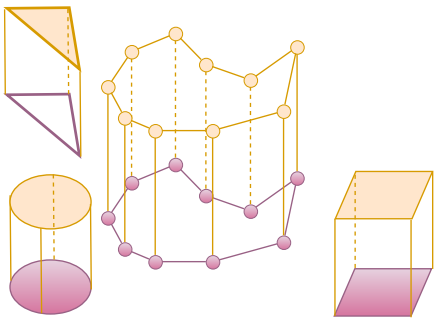
Medzi základné geometrické objemové telesá patrí guľa, kváder, valec, a ihlan. Nad týmito geometrickými telesami je možné vykonávať rôzne geometrické operácie.



Obr. 3.17: Ihlany s rôznymi základmi.

Extrudovanie

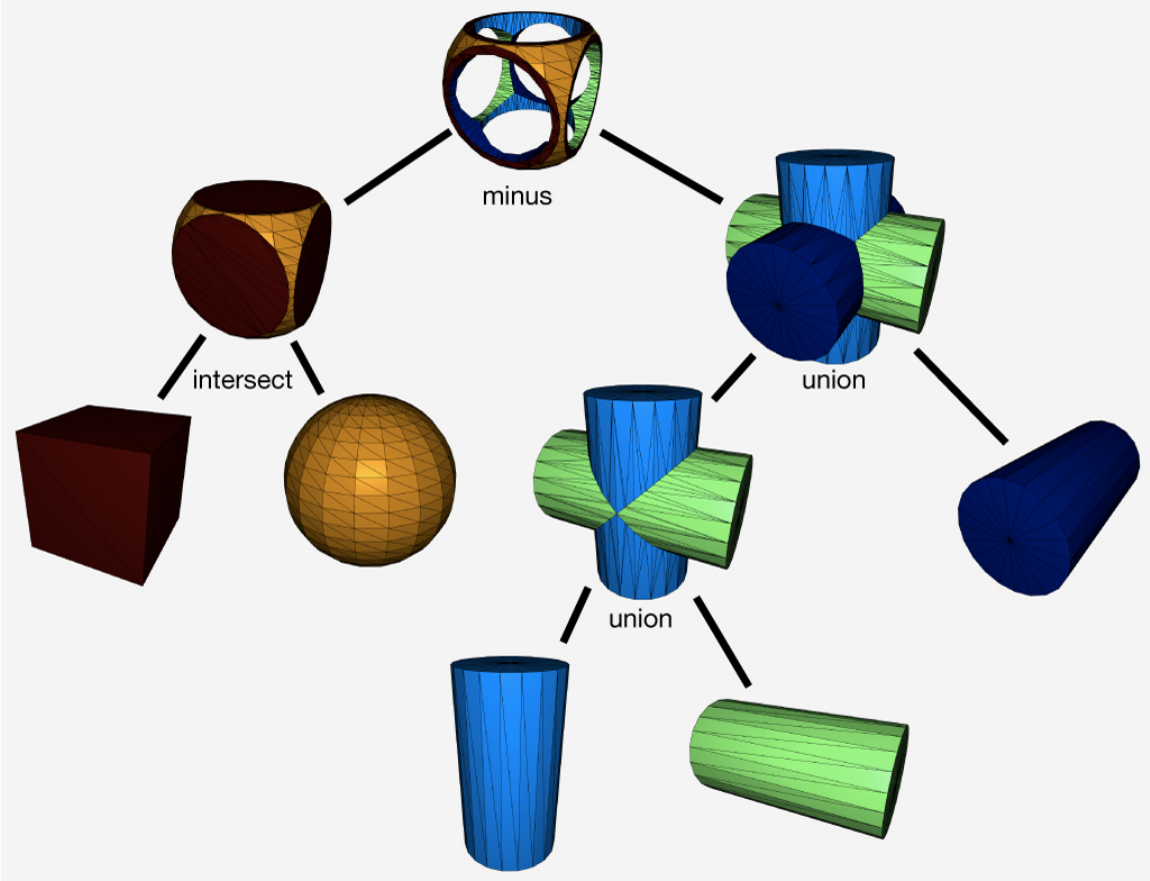
Natiahne plochu do priestoru v smere normály, čím získa na objeme. Pomocou tejto operácie sa dá vytvoriť aj napríklad kváder alebo valec, ak je základ štvorcového, respektívne kruhového tvaru. Táto operácia je zobrazená na obrázku 3.18, kde je fialovým zobrazená počiatočná plocha a žltým je zobrazená extrudácia. Extrudáciu je možné spraviť tak, že sa vytvorí kópia zadanej plochy a posunie sa o zadaný offset. Tieto dve plochy sa následne prepoja. Častou modifikáciou je extrudovanie po krivke a/alebo postupná zmena veľkosti [18].



Obr. 3.18: Extrudovanie

Booleovské operácie nad objemovými telesami

Pre vytvorenie zložitejších objektov je potrebné jednoduché objekty skladať pomocou booleovských operácií ako zjednotenie (union), prienik (intersection) a rozdiel medzi množinami (minus). Tieto operácie sú zobrazené aj na obrázku 3.19, kde tvoria CSG (constructive solid geometry) strom. Keďže sú tieto operácie náročné na spracovanie, rozhodol som sa využiť niektorú z voľne dostupných knižníc. Našiel som niekoľko dostupných knižníc, ktoré by sa dali použiť. Jednou z nich je knižnica `libigl`.



Obr. 3.19: Skladanie objektov pomocou booleovských operácii [9]

Knižnica libigl je používaná aj veľkými spoločnosťami ako je Activision, EA, Adobe, Google, Epic Games, Microsoft, Pixar, UBISOFT a ďalšie. Táto knižnica je distribuovaná pod licenciou Mozilla Public License (MPL). Obľúbená je hlavne pre svoju jednoduchosť. Pre tvorbu booleovských operácií nad mesh stačí zavolať funkciu `mesh_boolean`. Táto funkcia pracuje s bodmi (V) a trojuholníkmi (F). Na výpočet zjednotenia (typ `MESH_BOOLEAN_TYPE_UNION`) pre mesh $A = (VA, FA)$ a mesh $B = (VB, FB)$ je nasledovná funkcia. Výsledok tejto funkcie sa uloží do mesh $C = (VC, FC)$ [9]:

```
igl::copyleft::cgal::mesh_boolean(VA,FA,VB,FB,MESH_BOOLEAN_TYPE_UNION,VC,FC);
```

Kapitola 4

Analýza a koncept práce

Touto kapitolou začína popis vlastnej práce. Táto kapitola obsahuje analýzu a koncept práce. Tiež obsahuje aj návrh, ako sa budú geometrické objekty vytvárať.

4.1 Analýza súčasného stavu vo svete

Existuje niekoľko systémov a knižníc pre tvorbu trojrozmerných modelov. Niekoľko z nich je opísaných v časti 2.3 a 2.4. Systémy opísané v časti 2.3 umožňujú návrh rôznych komplexných modelov. Tieto modelovacie systémy sú veľmi kvalitné a často používané aj profesionálnymi grafickými dizajnérmi. Tieto systémy vytvárajú modely pomocou skladania rôznych geometrických operácií.

Ak ale užívateľ chce prepojiť takýto parametrický model s programovacím jazykom, napríklad kvôli nastaveniu parametrov z výsledkov simulácie alebo na jeho dynamickú úpravu, má často iba obmedzené možnosti. Niektoré modelovacie systémy obsahujú aplikačné programovacie rozhranie (API). Modelovacie systémy zväčša umožňujú prácu s modelmi iba pomocou skriptov alebo je potrebné pridať doplnky (angl. add-in), na prepojenie modelovacích systémov s externou aplikáciou. Rozhranie takéhoto systému je ale často veľmi komplikované na používanie.

Názov	Skripty	API	Cena
Solidworks	áno	Add-in	$4k-$8k
Catia	áno		$11k
FreeCAD	áno		Zadarmo
Creo Parametric		áno	$2,200
Rhino s Grasshopper	áno		$995
Fusion 360	áno	Add-in	Zadarmo - $1,500
Inventor		Add-in	$5,000

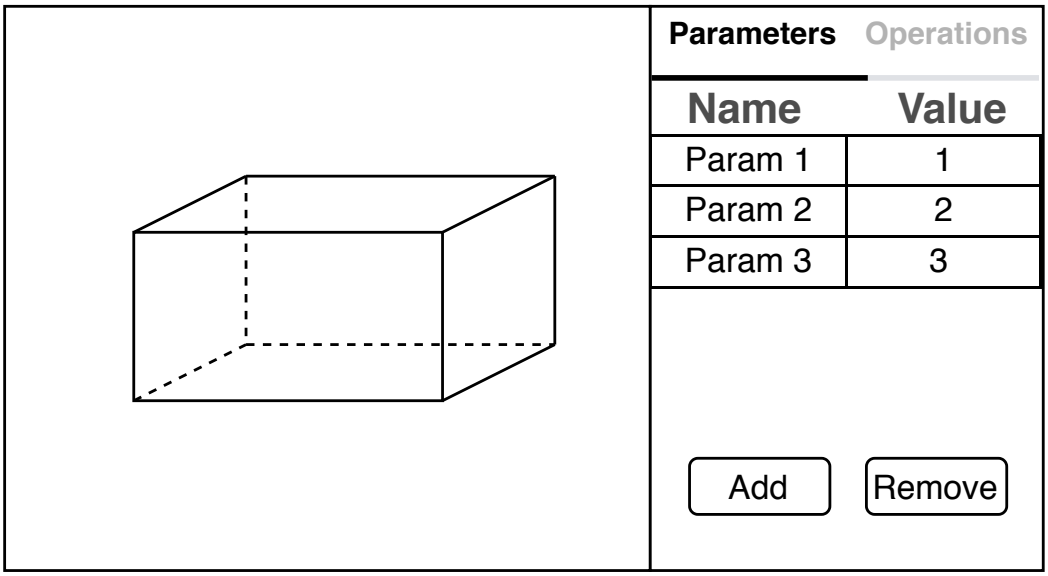
Tabuľka 4.1: Tabuľka vlastností existujúcich systémov

Pri použití knižnice na tvorbu trojrozmerných modelov sa nerozširuje komplexný modelovací systém, ale sa rozširuje programovací jazyk o možnosť tvorby geometrických modelov. Existujúce knižnice, ktoré umožňujú tvorbu geometrických modelov sú opísané v časti 2.4. Tieto knižnice nie sú také komplexné ako modelovacie systémy, a preto je ich rozhranie podstatne jednoduchšie.

4.2 Koncept práce

Z vykonanej analýzy vyplýva, že existujúce systémy pre tvorbu parametrických modelov, buď umožňujú pripojenie s externými systémami pomocou API (Application Programming Interface), ktoré sú ale zväčša komplikované na používanie alebo obsahujú iba skripty, ktoré sú určené len na tvorbu modelov, ale neumožňujú pripojenie k externým systémom. Preto som sa rozhodol vytvoriť systém, ktorý bude obsahovať knižnicu, ktorá bude mať jednoduché rozhranie pre tvorbu a nastavovanie modelu a aplikáciu, ktorá bude využívať túto knižnicu a uľahčovať ich tvorbu.

Takáto knižnica by mala byť založená na vytváraní objektov pomocou geometrických operácií, ktoré na seba budú nadväzovať. Tiež by mala mať jednoduché rozhranie pre tvorbu modelov a umožňovať ich vykresľovanie do grafického okna aplikácie. Súčasťou realizačnej časti musí byť aj vytvorenie grafického rozhrania, pre návrh parametrických modelov v trojrozmernom priestore. Návrh základnej koncepcie rozhrania je uvedené na obrázku 4.1.



Obr. 4.1: Návrh grafického užívateľského rozhrania

Takéto grafické užívateľské rozhranie by malo umožňovať zobrazenie modelu, nastavovanie jednotlivých parametrov, a tiež pridávanie a úpravu jednotlivých operácií, ktoré vytvárajú parametrický model.

Rozhranie knižnice by malo umožňovať aj ukladanie takéhoto modelu do súboru a jeho opätovné načítanie.

Rozhodol som sa, že táto knižnica bude využívať externú knižnicu libigl, ktorá bude použitá na konštruktívnu geometriu a knižnice, na ktorých je knižnica libigl závislá.

4.3 Technické parametre vytváraného softvéru

Na základe analýzy a koncepcie by mal softvér zahrňovať tieto vlastnosti:

- vytváranie objektov pomocou geometrických operácií, ktoré sú závislé od iných objektov,
- podpora všetkých geometrických objektov a operácií, ktoré sú opísané v kapitole 3,

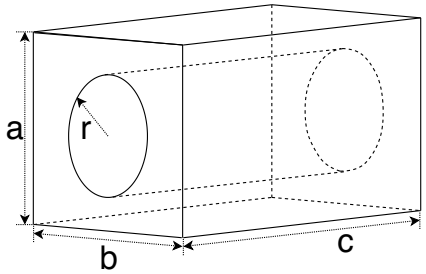
- vykresľovanie trojrozmerného modelu pomocou OpenGL,
- univerzálne vykresľovanie knižnice, pre použitie s rôznymi knižnicami pre tvorbu grafického okna, ako je Qt alebo GLFW,
- ukladanie modelu v parametrickom súborovom formáte,
- ukladanie objektu v polygonálnom formáte OBJ,
- animácie modelu v čase,
- dynamické modelovanie,
- možnosť získavania hodnôt z objektov,
- softvér napísaný v jazyku C++.

4.4 Návrh vytvárania geometrických objektov

Vytváranie trojrozmerných parametrických modelov musí byť založené na skladaní rôznych operácií. Knižnica musí byť schopná spracovať následujúce príklady úloh.

Príklad 1. Vytvorenie kvádra s kruhovým výrezom

Cieľovým objektom tohoto príkladu je kváder s kruhovým výrezom, viď. obrázok 4.2. Základné parametre tohoto útvaru sú zhrnuté v tabuľke 4.2 a sú zobrazené na obrázku 4.2.

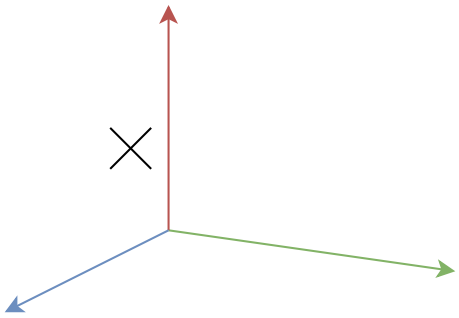


Obr. 4.2: Ukážka vytvoreného kvádra s kruhovým výrezom

Parameter	Popis parametru
a	Výška kvádra
b	Šírka kvádra
c	Hĺbka kvádra
r	Priemer výrezu

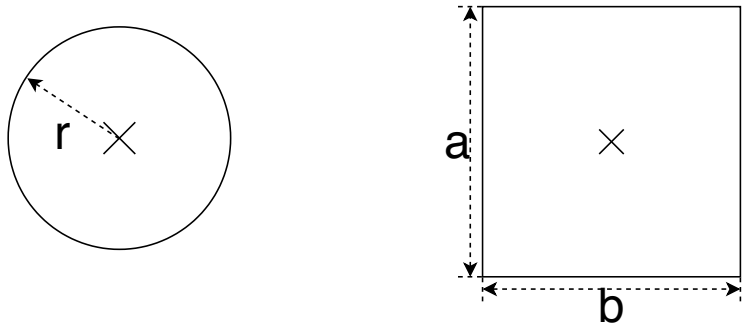
Tabuľka 4.2: Parametre modelu - kváder s kruhovým výrezom

Počiatočný bod, od ktorého bude model vytváraný. Na pozícii tohoto bodu závisí pozícia celého objektu.



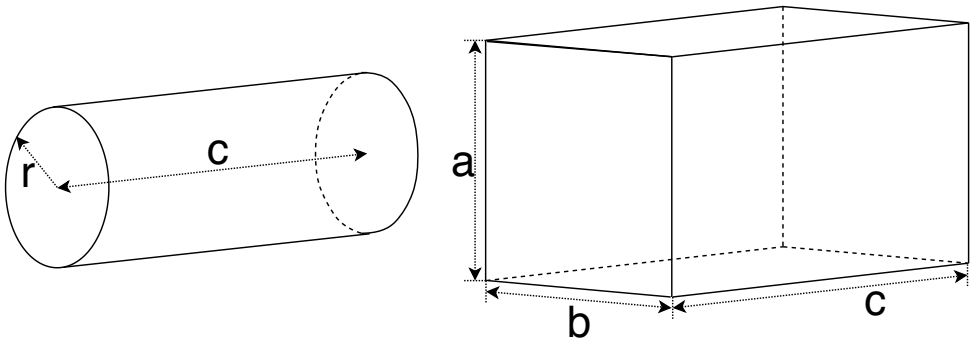
Obr. 4.3: Počiatočný bod

Vytvorenie kruhu a štvorca. Šírka a výška objektu je udávaná podľa parametrov modelu a a b. Prierez kruhu udáva parameter r. Tieto objekty majú rovnakú orientáciu normály v priestore.



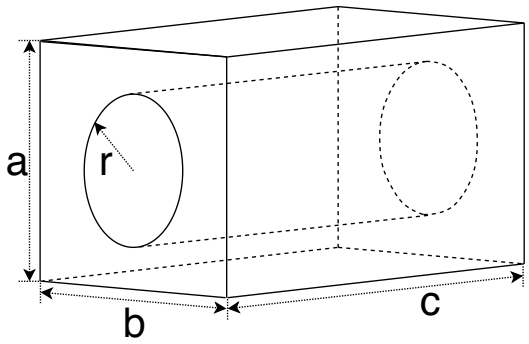
Obr. 4.4: Kruh a štvorec, z ktorých sa vytvoria trojrozmerne objekty

Pridaním hĺbky c ku kruhu a štvorcu, vznikne kváder a valec.



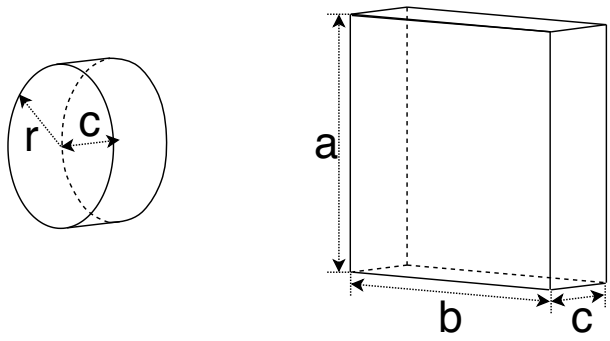
Obr. 4.5: Kváder a valec, vytvorené pomocou natiahnutia kruhu a štvorca do priestoru

Rozdielom kruhu od kvádra vznikne kváder s kruhovou dierou.



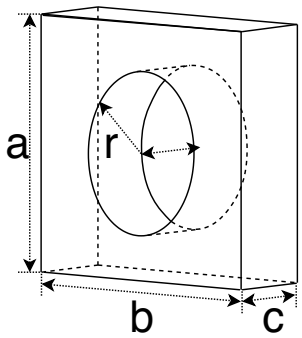
Obr. 4.6: Vytvorený kváder s kruhovým výrezom

Pri modelovaní ale často nie je hneď zrejmé, aký rozmer má mať objekt. Na to slúžia v parametrickom modeli parametre. Pri zmene hodnoty parametru c na menšiu hodnotu bude objekt vyzerať následne:



Obr. 4.7: Kváder a valec

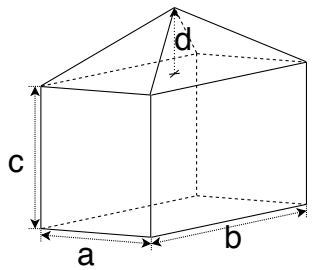
Rozdielom týchto objektov vznikne následný objekt:



Obr. 4.8: Kváder s valcovým výrezom

Príklad 2 - Vytvorenie modelu domu

Cieľovým objektom tohoto príkladu je model domu, viď obrázok 4.9. Základné parametre tohoto útvaru sú zhrnuté v tabuľke 4.3 a sú zobrazené na obrázku 4.9 .

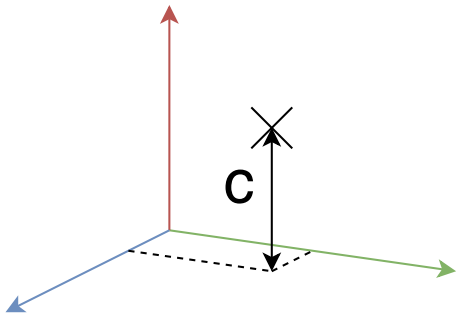


Obr. 4.9: Ukážka modelu domu

Parameter	Popis parametru
a	Šírka domu
b	Dĺžka domu
c	Výška domu
d	Výška strechy

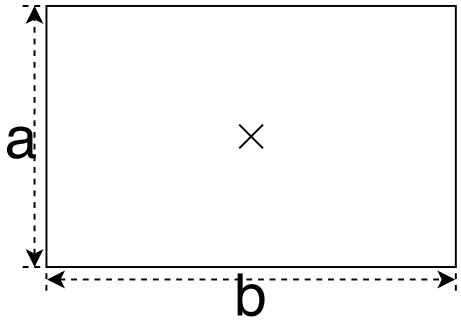
Tabuľka 4.3: Parametre modelu - dom

Ako prvé je potrebné stanoviť počiatočný bod, z ktorého sa bude model domu vytvárať. Na tomto bode je závislá pozícia domu. Keďže z tohto bodu sa bude vytvárať aj strecha, jeho výška je stanovená parametrom c.



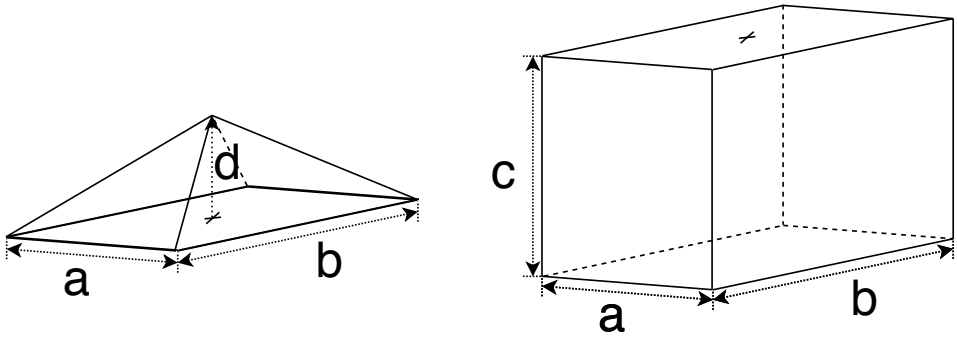
Obr. 4.10: Počiatočný bod domu

Na bode sa vytvorí obdĺžnik, so šírkou zadanou parametrom a a dĺžkou parametrom b. Tento obdĺžnik leží na vodorovnej rovine.

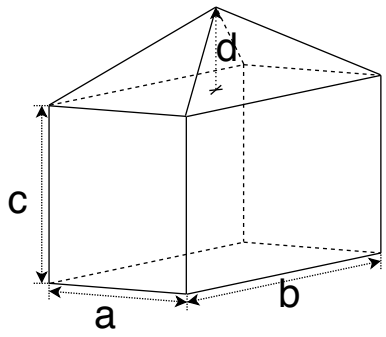


Obr. 4.11: Rovina domu

Pridaním výšky k rovine sa vytvorí blok domu. Z roviny vytvorenej v predchádzajúcom kroku sa dá vytvoriť aj strecha domu, a to vytvorením ihlanu s výškou zadanou parametrom d.



Obr. 4.12: Vľavo: Strecha domu. Vpravo: Kváder domu



Obr. 4.13: Model domu

Spojením týchto objektov, kvádra domu a strechy domu, vznikne kompletný model domu, zobrazený na obrázku 4.13.

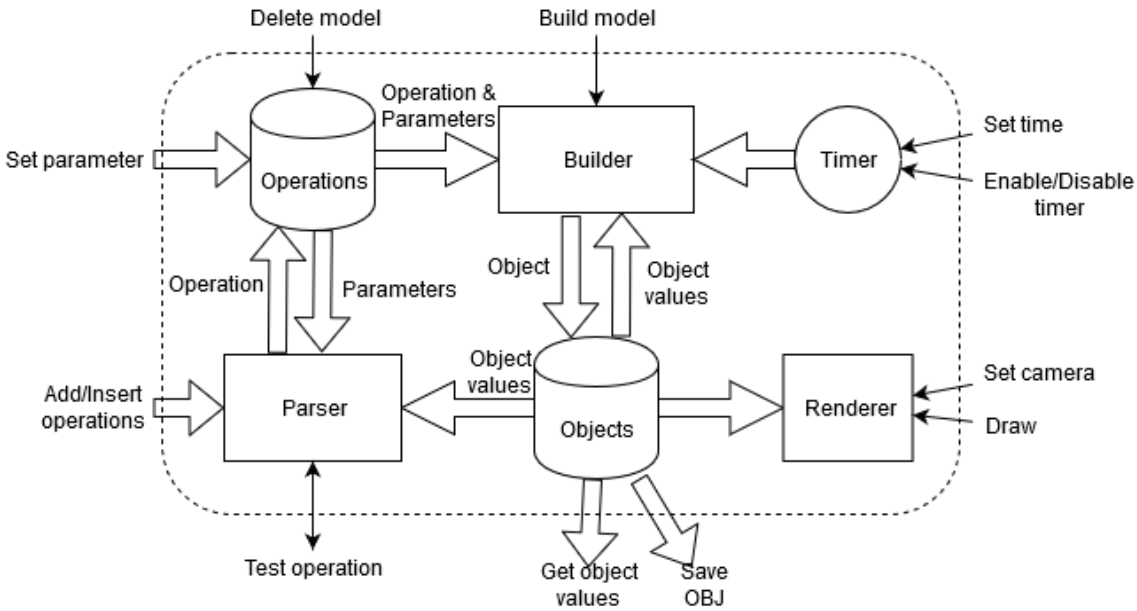
Kapitola 5

Implementácia práce

V tejto časti je opísaný spôsob implementácie softvéru, návrh možností previazania objektov v parametrickom modely a implementácia aplikácie a knižnice na tvorbu trojrozmerných parametrických modelov, kde sú tieto možnosti previazania implementované.

5.1 Bloková schéma

Vytvorený softvér by mal mať jednoduché rozhranie, aby bol ľahko použiteľný pre používateľa. Toto rozhranie je vidieť aj na blokovom diagrame 5.1. Operácie by mali byť ukladané v zozname operácií. Po zavolaní metódy pre vyskladanie objektu sa všetky operácie vyhodnotia a vytvoria objekt. Operácie môžu obsahovať aj hodnoty parametrov modelu a aj hodnoty objektov, ktoré boli vytvorené z predchádzajúcich operácií. Výsledný model, by malo byť možné animovať. Na to je potrebné zahrnúť pri vyhodnocovaní operácií aj časovú premennú. Tú by sa malo dať ľubovoľne nastavovať a aj pozastaviť.



Obr. 5.1: Bloková schéma softvéru

Vytvorené objekty sa budú ukladať do zoznamu objektov a ak užívateľ chce, malo by sa dať vytvorený model aj vizualizovať.

5.2 Skladanie operácií

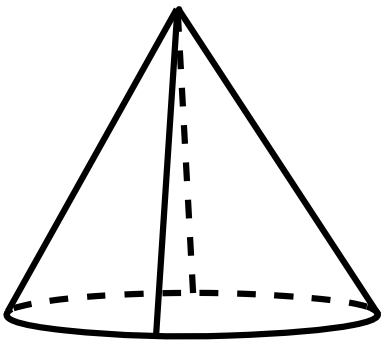
Všetky geometrické operácie sa dajú zapisovať pomocou textovej podoby. Textová podoba geometrickej operácie sa začína s názvom použitej geometrickej operácie a parametrami uvedenými v zátvorkách `(` a `)`, oddelenými čiarkou. U jednotlivých parametrov sa medzery na začiatku a konci ignorujú.

Keďže pri niektorých objektoch existuje viac možností ako objekt vytvoriť, inšpiroval som sa preťažovaním funkcií v jazyku C++ a umožnil som, aby mali niektoré operácie rovnaký názov, ale rozdielny typ argumentov. Parametre môžu byť dvoch typov, a to buď názov objektu alebo v tvare výrazu. Viac o možnostiach výrazov je v časti 5.4. Operácie, ktoré táto knižnica umožňuje využiť, sú uvedené v prílohe C, s ich textovým zápisom a s požadovanými parametrami, potrebnými na zostavenie týchto operácií.

Parametrom operácií, ktoré sú typu *Expression* (zadávané výrazom), je možné vytvoriť názov, pomocou ktorého sa bude dať na daný parameter neskôr pristupovať. Tento názov sa zadáva za hodnotou parametra oddelený dvojbodkou.

Jednotlivé operácie sa oddeľujú bodkočiarkou (`;`). Táto textová podoba geometrických operácií sa používa pri vytváraní modelu a aj pre uloženie do súboru, a tiež jeho následné načítanie.

Pre predstavu ako sa jednotlivé objekty vytvárajú, uvádzam príklad pre vytvorenie kužeľu s kruhovou podstavou s polomerom 5 a výškou 10. Tento model je zobrazený na obrázku 5.2



Obr. 5.2: Ukážka vytváraného kužeľa

- Na vytvorenie kužeľu je potrebné najprv vytvoriť bod na pozícii XYZ, napr. $[1, 2, 3]$, ktorý bude slúžiť ako stred kruhovej podstavy.

```
Point(Stred podstavy, 1, 2, 3 ,00000000);
```

- Na vytvorenie kruhu (podstavy kužeľa) je potrebný polomer a normála. Ako normálu je možné využiť smer ľubovoľnej úsečky, ale keďže zatiaľ žiadna úsečka vytvorená nie je, je potrebné ju vytvoriť. Úsečka sa skladá z dvoch bodov, a to počiatočného a koncového. Ako normála sa použije normalizovaný vektor medzi bodom počiatočným a koncovým 5.1.

$$\overrightarrow{normal_vector} = \frac{end_point - begin_point}{\|end_point - begin_point\|} \tag{5.1}$$

```
Point(Počiatočný bod, 0, 0, 0, 00000000);
Point(Koncový bod, 1, 1, 1, 00000000);
Line(Normála, Počiatočný bod, Koncový bod, 00000000);
```

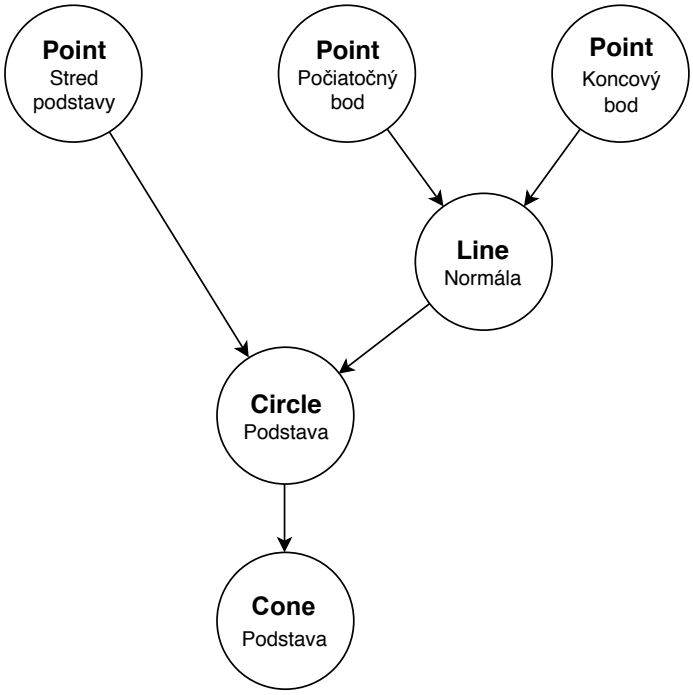
- Keď už je normála vytvorená, je možné vytvoriť kruh s polomerom 5.

```
Circle(Podstava, Stred podstavy, 5, Normála, 00000000)
```

- Ostáva vytvoriť samotný kužeľ s výškou 10.

```
Cone(Kužeľ, Podstava, 10, 00FF00FF)
```

Tento sled operácií je možné zobraziť pomocou orientovaného grafu 5.3.



Obr. 5.3: Sled operácií pomocou orientovaného grafu

Operácie používajú iba už vytvorené objekty a ich hodnoty, teda žiadny objekt nemôže byť závislý od iného objektu, ktorý je závislý na tomto objekte. Takýto graf sa tiež nazýva aj acyklický.

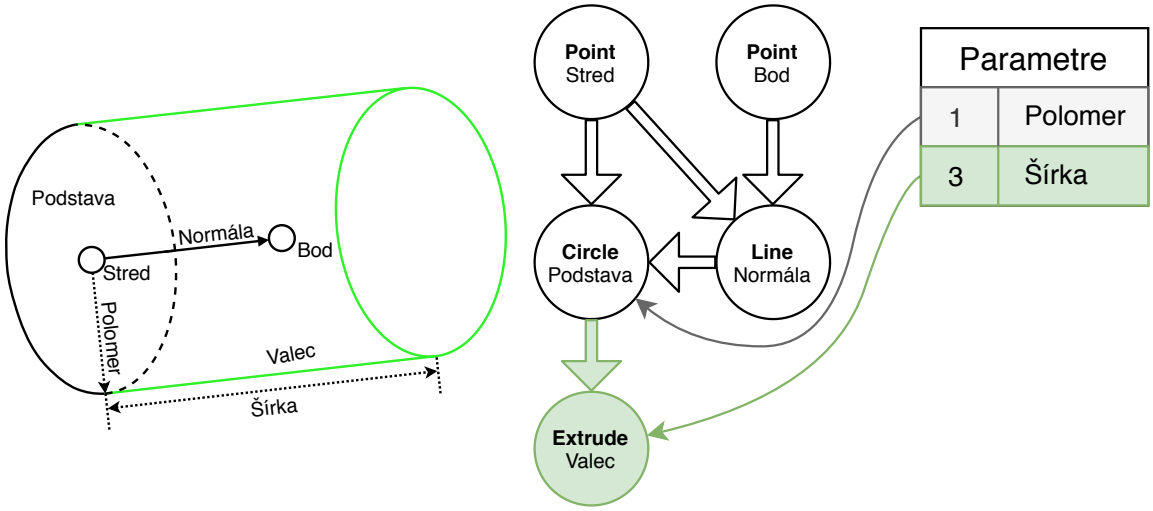
5.3 Pridávanie a odoberanie objektov

Jedným z cieľov bolo, aby sa dal model dynamicky upravovať. Na to vo vytvorenej knižnici slúži niekoľko metód. Pre pridávanie operácií do modelu slúžia metódy `AddOperation` a `AddOperations`, ktoré umožňujú pridávanie operácii v `String`-ovej forme v tvare ako je

napísané v časti 5.2. Na obrázku 5.4 je zobrazené, ako sa prejaví po vložení následujúcej operácie pomocou metódy `AddOperations`:

`AddOperations("Extrude(Valec,Podstava,3:Šírka,00FF00FF)").`

Po zavolaní tejto metódy, sa do zoznamu operácií vloží operácia `Extrude`, ktorá z kruhu `"Podstava"`, vytvára valec. Táto operácia tiež pridáva aj parameter `"Šírka"`.



Obr. 5.4: Ukážka, ako sa zmení model a graf modelu, pri vkladaní operácie

V zobrazenej ukážke je kruh vytváraný pomocou stredového bodu, normály a polomeru kruhu. Operácia Extrude vytvára objekt pretiahnutím plošného objektu kolmo, teda smerom jeho normály.

Metóda `AddOperations` umožňuje pridanie množiny operácií, kde jednotlivé zápisy operácií sú oddelené znakom bodkočiarky (;). Operácie je tiež možné zapísať aj v tvare štruktúry `Operation`, kde sa zapisujú jednotlivé zložky operácie, ako sú typ operácie, názov vytváraného objektu a parametre operácie. Pomocou metódy `InsertOperation` je možné vkladať operácie do už vytvoreného modelu na ľubovoľnú pozíciu, prípadne pomocou metódy `ReplaceOperation` nahradiť ľubovoľnú operáciu. Metóda `RemoveOperation` umožňuje odstránenie jednej operácie na zadanej pozícii. Pre vymazanie celého modelu slúži metóda `DeleteModel`.

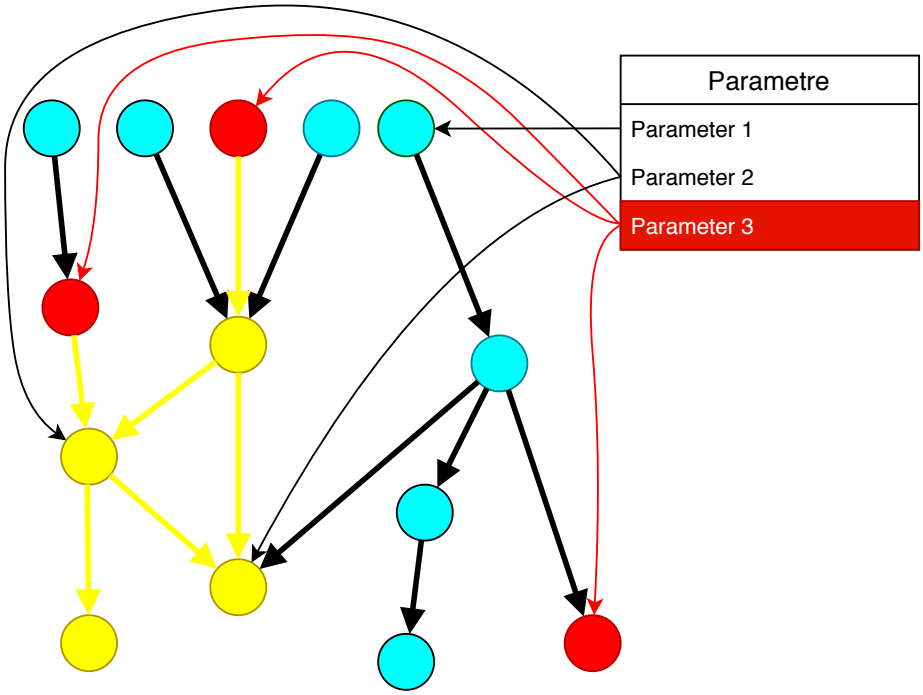
Knižnica tiež umožňuje uloženie parametrického modelu do súboru a jeho následné načítanie. Aby bolo možné použiť vytvorený objekt aj v iných aplikáciach, je možné tento objekt exportovať do súborového formátu OBJ, ktorý kvôli svojej jednoduchosti patrí medzi najviac podporované geometrické súborové formáty.

Pre otestovanie, či v zapísanom reťazci nie je chyba, či už syntaktická alebo sémantická, slúži testovanie. Na začiatku každého testovania je potrebné testovanie resetovať a následne je možné pridávať operácie pomocou metódy `TestOperation`. Takto pridané operácie sa nezapisujú do samotného modelu a budú vymazané pri ďalšom zavolaní metódy `resetTest`. Metóda TestOperation vracia hodnotu True, ak sa nevyskytla žiadna chyba, a je teda možné operáciu do modelu bezchybne vložiť.

Aby sa urýchlil výpočet modelu, spracovanie sa nezačína hneď po každej úprave modelu, ale je potrebné, aby bola zavolaná metóda `BuildModel`. Pri skladaní iba jednoduchých operácií stačilo celý model pred generovaním zahodiť a vytvoriť znova celý model od začiatku. Pri operáciach, ktoré trvajú niekoľko sekúnd, ako sú operácie konštruktívnej geometrie, je

takýto postup generovania príliš pomalý. Preto som navrhol algoritmus tak, aby sa znova generovali iba objekty, ktoré sa nejakým spôsobom modifikovali, či už pridaním novej operácie alebo zmenou hodnoty niektorého parametra operácie, alebo ak sa zmenila niektorá operácia, na ktorej je táto operácia tvoriaca daný objekt závislá. Taktiež je potrebné kontrolovať, či niektorá premenná v zadanom výraze nezmenila svoju hodnotu (napríklad časová premenná alebo parameter niektorého objektu - viac v časti 5.4) a je teda potrebné znovu vypočítať zadaný výraz a podľa nových hodnôt vytvoriť objekt.

Táto optimalizácia je zobrazená aj na grafe 5.5. V tomto grafe sú vyznačené operácie, ktoré je potrebné znovu vyhodnotiť, pri zmene jedného z parametrov.



Obr. 5.5: Graf operácii, pri zmene jedného parametra

Červenou farbou sú vyznačené operácie, ktorým sa zmenil parameter. Žltou farbou sú vyznačené operácie, ktoré sú závislé od výsledku pozmenenej operácie. Tieto operácie je potrebné tiež znovu vyhodnotiť. Modrou farbou sú vyznačené operácie, ktorým sa žiadna hodnota nezmenila a nezmenila sa ani žiadna operácia, na ktorej sú závislé. Výsledok týchto operácií by bol rovnaký, ako pred zmenou hodnoty parametra. Preto tieto operácie nie je potrebné znova spracovať, a tým sa dá ušetriť výpočtový čas na znovupostavenie modelu.

5.4 Výrazy v operáciách

Zo začiatku umožňovala knižnica v parametroch operácií iba číselnú hodnotu, s presnosťou dátového typu *double*. To ale často nebolo dostatočné, preto som sa rozhodol prerobiť túto hodnotu na výrazy so základnou aritmetikou a s použitím aj iných hodnôt z objektov. Neskôr som pridal aj trigonometrické operácie a zaokrúhľovanie. Pridal som aj časové premenné, aby sa dal model animovať v čase. Parametre operácií môžu byť zapísané v tvare výrazu. Tieto výrazy sa počas zápisu overia a spracúvajú. Vyhodnocovanie prebieha až v čase vytvárania modelu.

Príklady zápisu rôznych výrazov v operáciách vytvárajúcich body:

```
Point(P1,1+2:param1,0-5:param2,1.5:param3,FF0000FF);
Point(P2,param1+param2, sin(param3*time),P1.X,00FF00FF);
```

Výrazy môžu obsahovať rôzne prvky. V tejto knižnici sú podporované nasledujúce prvky:

- čísla (0; 1.5; -5.5)
- parametre modelu
- hodnoty z objektov
- časové premenné
  - time - časová hodnota v milisekundách
  - time_seconds - časová hodnota v sekundách
- unárne operácie
  - round - zaokrúhlenie na celé číslo
  - ceil - zaokrúhlenie na vyššie celé číslo
  - floor - zaokrúhlenie na nižšie celé číslo
  - trunc - odstráni desatinnú časť čísla
  - sin - sínus
  - cos - kosínus
  - tan - tangens
  - asin - arkus sínus
  - acos - arkus kosínus
  - atan - arkus tangens
  - sqrt - odmocnina
- binárne operácie
  - + sčítanie
  - - odčítanie
  - * násobenie
  - / delenie
  - % modulo
  - ^ exponent
- zátvorky
  - (
  -)

Precedenčná tabuľka

Pre vyhodnocovanie výrazov v realizovanom systéme je potrebné dbať na precedenciu operátorov. Preto je potrebná precedenčná tabuľka. Tá označuje, ktorá operácia má vyššiu prioritu pri výpočte. Zvislé popisky udávajú, aký znak je na vrchole zásobníka a vodorovné popisky udávajú, aký znak práve prišiel (vstupný token). Znak $ na zásobníku určuje dno zásobníka a $ na vstupe určuje koniec. Ak je výsledná hodnota z precedenčnej tabuľky znak X, znamená to, že zadaný výraz je zapísaný v chybej forme, a teda nie je možné takýto výraz ďalej riešiť.

Pre vyhodnocovanie výrazov sa v knižnici využíva nasledovná precedenčná tabuľka:

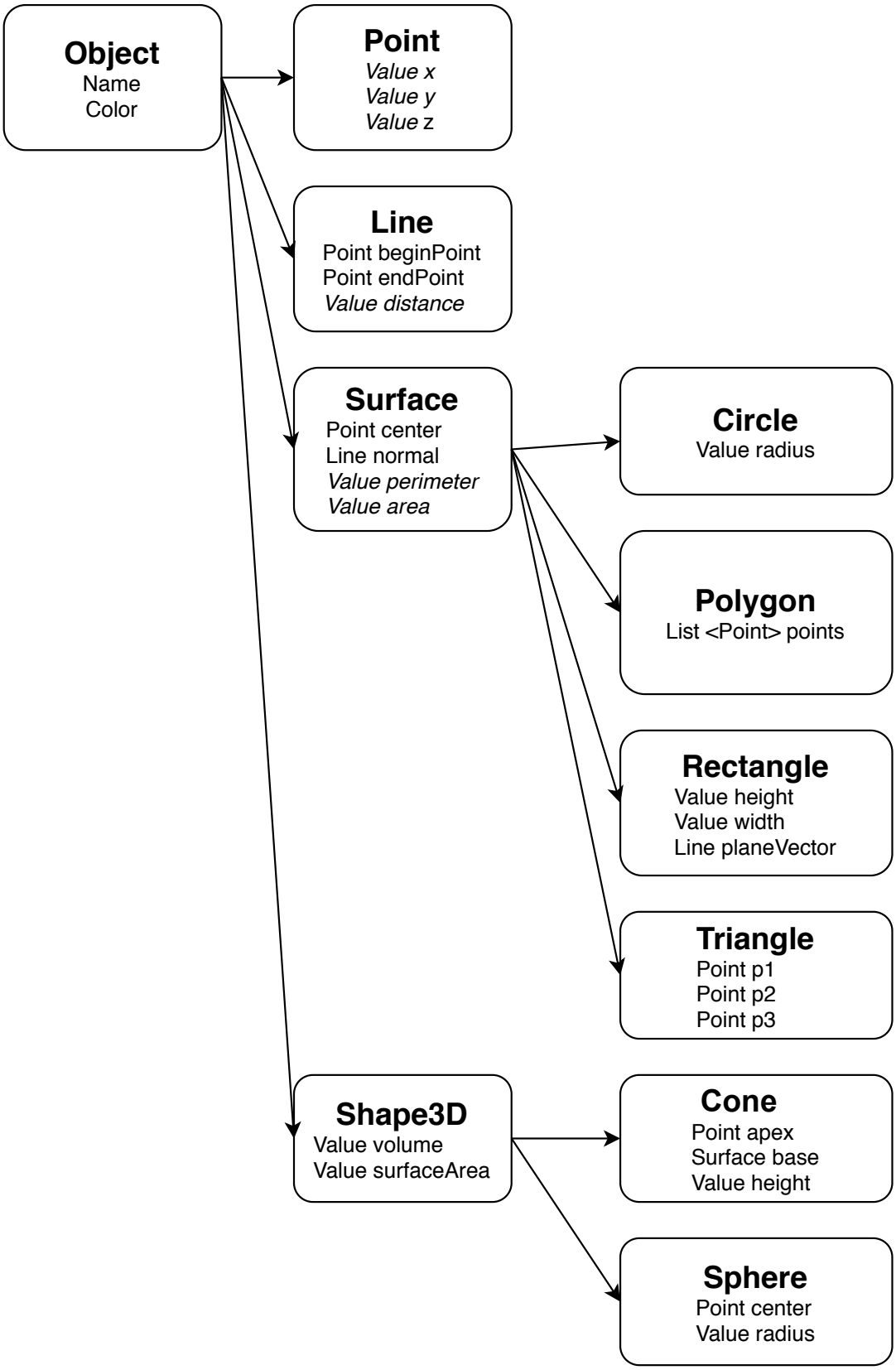
	+	-	*	/	%	i	(	)	^	unary	$
+	>	>	<	<	<	<	<	>	<	<	>
-	>	>	<	<	<	<	<	>	<	<	>
*	>	>	>	>	>	<	<	>	<	<	>
/	>	>	>	>	>	<	<	>	<	<	>
%	>	>	>	>	>	<	<	>	<	<	>
i	>	>	>	>	>	X	X	>	>	>	>
(	<	<	<	<	<	<	<	=	<	<	X
)	>	>	>	>	>	X	X	>	>	X	>
^	>	>	>	>	>	<	<	>	<	<	>
unary	>	>	>	>	>	<	<	>	>	>	>
$	<	<	<	<	<	<	<	X	<	<	O

Tabuľka 5.1: Precedenčná tabuľka

Ak nastane situácia, kedy nie je možné niektorý výraz vyriešiť, napríklad delenie nulou, geometrická operácia nevytvorí takýto objekt a taktiež operácie, ktoré sú na tomto objekte závislé nevytvoria svoje objekty.

Parametre objektov

Niektoré geometrické objekty majú podobné parametre. Všetky plošné útvary majú normálu, stred a aj plochu a obvod. Tieto parametre pre geometrické objekty sa dajú zobraziť pomocou stromu dedičnosti. Ten je zobrazený na obrázku 5.6. Jednotlivé geometrické objekty obsahujú priame hodnoty, teda parametre, na ktorých je objekt závislý, ako výška alebo šírka objektu a hodnoty, ktoré je potrebné vypočítať, ako sú napríklad obvod alebo objem objektu. Napríklad kužeľ s kruhovou podstavou obsahuje stred a polomer základne a výšku kužeľu. Vypočítanými hodnotami tohto objektu sú obvod a obsah základne alebo objem a plocha kužeľu.



Obr. 5.6: Štruktúry objektov vytvárajú strom dedičnosti (Pri každom objekte je uvedené, aké dáta uchováva)

Vo vytvorenej knižnici je možné pristupovať k jednotlivým hodnotám objektom pomocou `GetObjectValue`. Použitie tejto metódy je vidieť na následujúcich príkladoch:

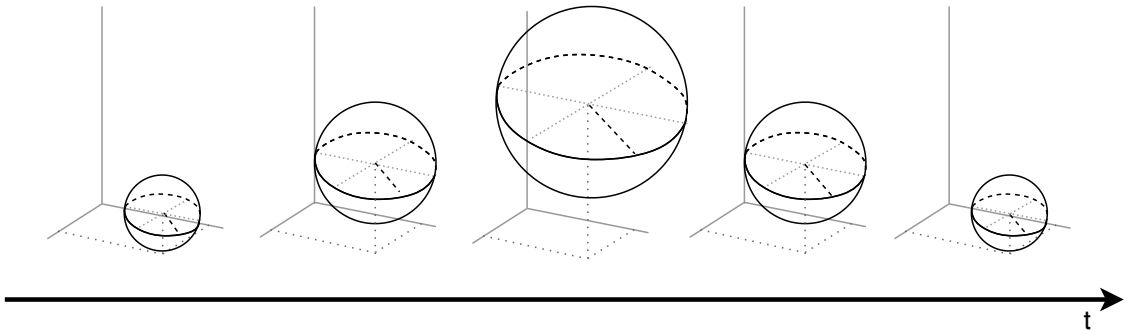
`GetObjectValue("Guľa.center.x")` - Hodnota pozície stredu kruhu na ose X,
`GetObjectValue("Trojuhoľník.perimeter")` - Obvod trujuhoľníka s názvom "Trojuhoľník",
`GetObjectValue("Kužeľ.volume")` - Objem kužeľa "Kužeľ".

Alternatívne sa dá pristupovať k štruktúram objektov priamo, pomocou funkcie `GetObject`, ktorý berie ako argument názov požadovaného objektu.

`((Object::Sphere*) paramModel.GetObject("Guľa"))->center.Position.X` - Hodnota pozície stredu kruhu na ose X
`((Object::Triangle*) paramModel.GetObject("Trojuhoľník"))->GetPerimeter()` - Obvod trujuhoľníka s názvom "Trojuhoľník",
`((Object::Shape3D*) paramModel.GetObject("Kužeľ"))->GetVolume()` - Objem kužeľa "Kužeľ".

Časové premenné

Knižnica by mala umožňovať aj vytváranie animácií. Pri tvorbe modelu s animáciou, je vhodné využiť časovú hodnotu a na tejto hodnote založiť operácie. Na obrázku 5.7 je zobrazená animácia skákajúcej guli, ktorej priemer sa mení podľa výšky. Čím je táto guľa vyššie, tým je jej priemer väčší.



Obr. 5.7: Animácia skákajúcej guli, ktorej polomer sa mení podľa jej výšky

Pre možnosti animácie modelu som vytvoril časové premenné time a time_seconds pre časovú hodnotu v milisekundách, prípadne v sekundách. Táto časová hodnota je po zapnutí aplikácie nastavená na 0 a behom aplikácie sa zvyšuje. V prípade, ak je potrebné nastaviť túto hodnotu na inú hodnotu je možné ju nastaviť manuálne pomocou metódy `setTime`.

V prípade ak užívateľ chce čas pozastaviť alebo preferuje nastavovanie času manuálne, je možné pozastaviť časovač pomocou nasledujúcej metódy `useManualTimer`.

5.5 Grafické rozhranie

Vytvorená knižnica umožňuje vytvorený model aj vizualizovať. Na to je potrebné vytvoriť okno, do ktorého sa bude model vykresľovať a k nemu grafický kontext. Vytvorená knižnica

neobsahuje metódy na vytvorenie grafického okna, ale iba metódy na vykresľovanie pomocou GLEW (The OpenGL Extension Wrangler Library)[1].

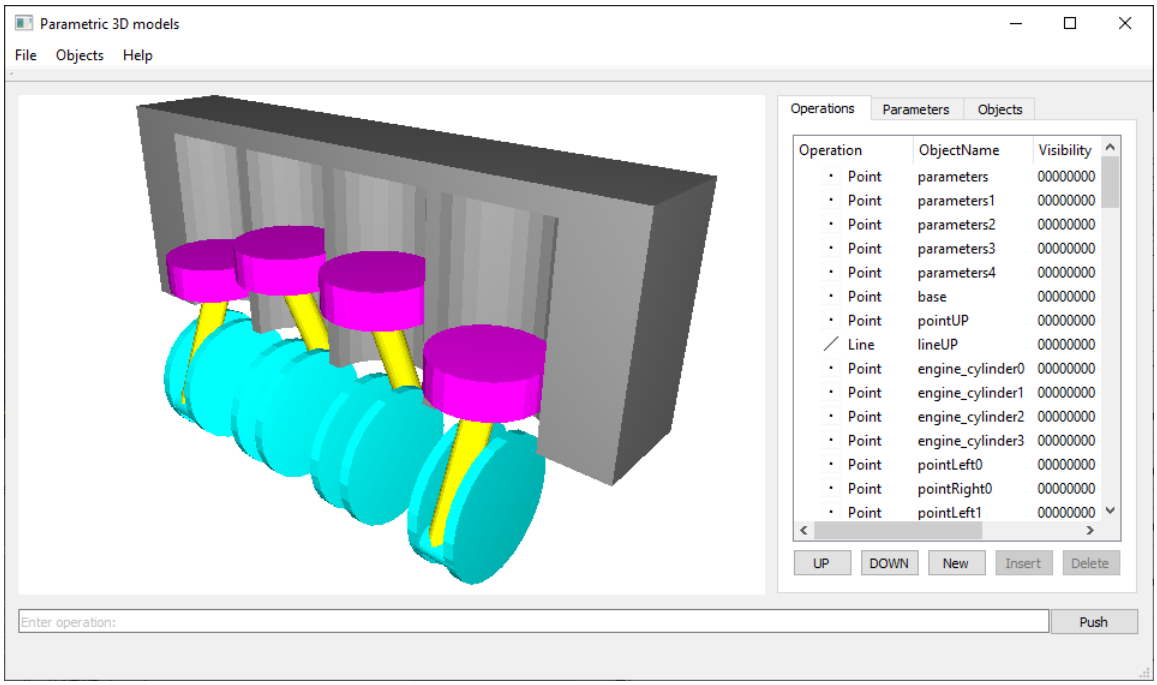
Pre použitie grafického rozhrania je potrebné, aby používateľ po vytvorení kontextu inicioval renderer. Minimálna požiadavka pre spustenie grafického rozhrania je podpora OpenGL vo verzii 3.3, v ktorom sú napísané shadery. Túto verziu som zvolil kvôli podpore starších a aj súčasných grafických kariet. Následne je potrebné volať vo vykresľovacej slučke metódu Draw, ktorá ako parameter berie pozíciu pre vykresľovanie, veľkosť zorného poľa a pomer strán okna, v ktorom sa má vykresľovať.

Následne je pre prácu s obrazom možné:

- nastavenie pozície kamery,
- natočenie kamery,
- nastavenie sýtosti ambientného osvetlenia v rozmedzí <0;1>.

Grafický editor

Práca obsahuje aj jednoduché grafické rozhranie, ktoré návrhárovi pomáha, pri vytváraní parametrického modelu a uľahčuje prácu s geometrickými operáciami, a zároveň pri tom zobrazuje aj internú štruktúru modelu, teda jednotlivé operácie, objekty a parametre a zároveň tento model aj vizualizuje. Táto aplikácia je vytvorená pomocou knižnice QT[2] a jej grafické rozhranie je zobrazené na obrázku 5.8.



Obr. 5.8: Grafický editor na tvorbu parametrických trojrozmerných modelov

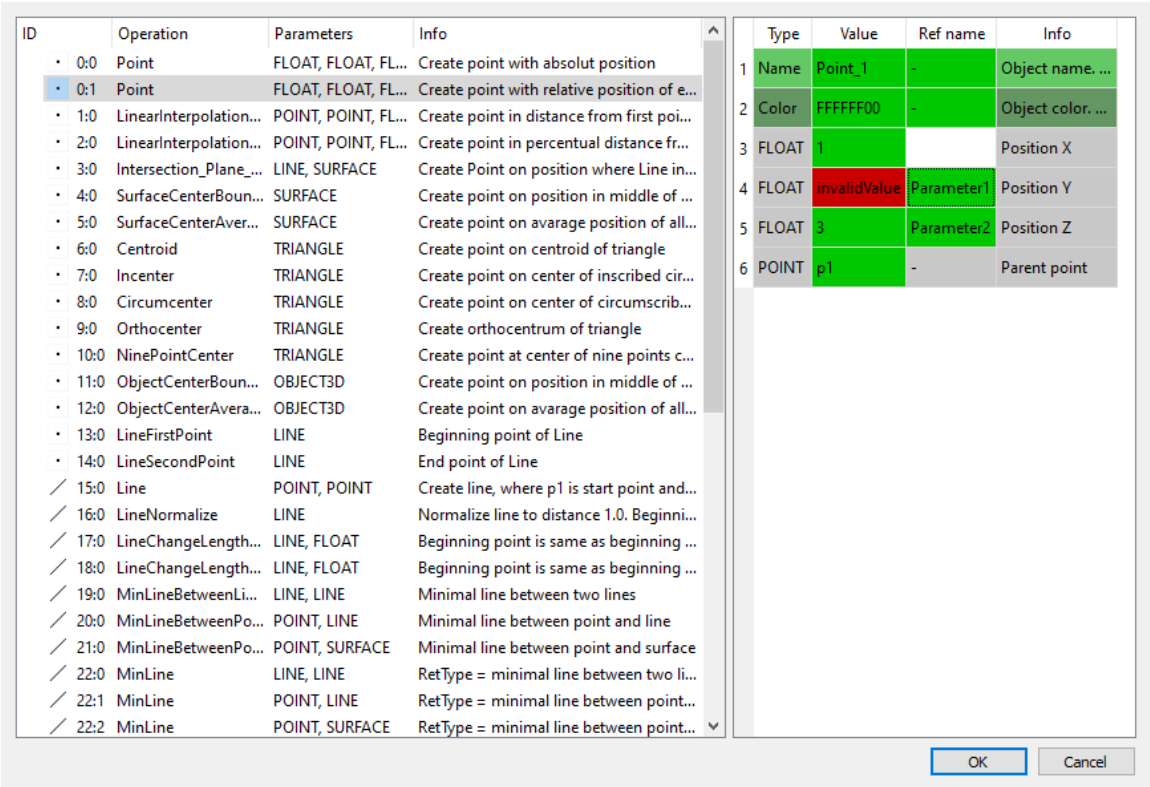
Aby bolo jednoduchšie vytvárať objekty, grafické rozhranie umožňuje geometrické operácie pridať, upraviť, mazať a aj vložiť na ľubovoľné miesto. Tieto operácie je tiež možné

[1] http://glew.sourceforge.net/
[2] https://www.qt.io/

ľubovoľne presúvať. Operácie sú zoradené zhora dole v poradí, v akom sa vyhodnocujú. Parametre je možné zapisovať rovnako, ako v programátorskej časti, teda aj vo forme výrazu s odkazmi na hodnoty iných objektov a parametrov modelu. Keďže parametre operácií môžu odkazovať iba na skôr vytvorené objekty, je nutné pri upravovaní, presúvaní a mazaní tieto parametre kontrolovať. Grafické rozhranie následne zobrazí, v ktorých operáciách sa nachádzajú nevyhovujúce parametre a tieto objekty (a objekty na nich závislé) sa nebudú vykresľovať.

Pridanie, vloženie a úprava geometrických operácií

Dialógové okno umožňuje výber operácie, ktorá sa má použiť, zo zoznamu geometrických operácií. V zozname sa nachádza názov geometrickej operácie, parametre, ktoré operácia potrebuje a nápoveda, ktorá informuje užívateľa čo daná operácia robí. Po vybraní operácie zo zoznamu, sa zobrazia v pravej časti dialógového okna parametre vybranej operácie. Ak je dialógové okno v režime úpravy (okno sa zobrazilo po dvojkliku na zozname geometrických operácií), sú pri vybraní rovnakého typu operácie tieto parametre už predvyplnené hodnotami zvolenej operácie na úpravu.



ID	Operation	Parameters	Info
0:0	Point	FLOAT, FLOAT, FL...	Create point with absolut position
0:1	Point	FLOAT, FLOAT, FL...	Create point with relative position of e...
1:0	LinearInterpolation...	POINT, POINT, FL...	Create point in distance from first poi...
2:0	LinearInterpolation...	POINT, POINT, FL...	Create point in percentual distance fr...
3:0	Intersection_Plane_...	LINE, SURFACE	Create Point on position where Line in...
4:0	SurfaceCenterBoun...	SURFACE	Create point on position in middle of ...
5:0	SurfaceCenterAver...	SURFACE	Create point on avarage position of all...
6:0	Centroid	TRIANGLE	Create point on centroid of triangle
7:0	Incenter	TRIANGLE	Create point on center of inscribed cir...
8:0	Circumcenter	TRIANGLE	Create point on center of circumscrib...
9:0	Orthocenter	TRIANGLE	Create orthocentrum of triangle
10:0	NinePointCenter	TRIANGLE	Create point at center of nine points c...
11:0	ObjectCenterBoun...	OBJECT3D	Create point on position in middle of ...
12:0	ObjectCenterAvera...	OBJECT3D	Create point on avarage position of all...
13:0	LineFirstPoint	LINE	Beginning point of Line
14:0	LineSecondPoint	LINE	End point of Line
15:0	Line	POINT, POINT	Create line, where p1 is start point and...
16:0	LineNormalize	LINE	Normalize line to distance 1.0. Beginni...
17:0	LineChangeLength...	LINE, FLOAT	Beginning point is same as beginning ...
18:0	LineChangeLength...	LINE, FLOAT	Beginning point is same as beginning ...
19:0	MinLineBetweenLi...	LINE, LINE	Minimal line between two lines
20:0	MinLineBetweenPo...	POINT, LINE	Minimal line between point and line
21:0	MinLineBetweenPo...	POINT, SURFACE	Minimal line between point and surface
22:0	MinLine	LINE, LINE	RetType = minimal line between two li...
22:1	MinLine	POINT, LINE	RetType = minimal line between point...
22:2	MinLine	POINT, SURFACE	RetType = minimal line between point...

	Type	Value	Ref name	Info
1	Name	Point_1	-	Object name. ...
2	Color	FFFFFF00	-	Object color. ...
3	FLOAT	1		Position X
4	FLOAT	invalidValue	Parameter1	Position Y
5	FLOAT	3	Parameter2	Position Z
6	POINT	p1	-	Parent point

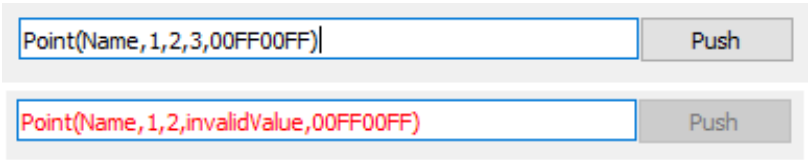
Obr. 5.9: Dialógové okno na pridanie, vloženie a úpravu operácií (Ukážka pridania bodu s názvom $Point_1$ s jednou chybne zadanou hodnotou)

Ako prvý parameter je názov takto vytvoreného objektu, ktorý je pri novo vytváraných objektoch (režim okna pridanie alebo vloženie) predvyplnený, ale je možné ho upraviť na ľubovoľnú hodnotu. Po názve objektu nasleduje farba objektu v tvare RRGGBBAA zapísaná hexadecimálne. Ďalej nasledujú samotné parametre operácie typu $Expression$, $Point$, $Line$, $Surface$ a ďalšie. Parametrom typu $Expression$ sa dá nastaviť názov, podľa ktorého sa na

ne bude odkazovať. Tento názov ale nie je povinný a pre parametre stačí nechať políčko prázdne. Parametre typu *Expression*, ktoré sú týmto názvom pomenované, sa následne zobrazia aj v zozname parametrov modelu. V poslednom stĺpci sa nachádza nápoveda k danému parametru.

Po zadaní hodnoty sa overuje, či je táto hodnota valídna pre daný parameter. To zahŕňa testovanie, či hodnota zadaná parametru *Expression* je platný výraz a pri ostatných parametroch sa testuje, či existuje objekt s rovnakým názvom ako bol zadaný, a či je tento objekt správnym typom prípadne podtypom (viac o typoch objektov v kapitole 3). Tiež sa testuje aj názov objektu a názov referencovaného parametra na unikátnosť. Ak hodnota parametra nesplňuje niektorú požiadavku, je toto políčko označené červenou farbou pozadia, čo upozorňuje užívateľa na chybu. Ak je hodnota valídna, políčko sa označí zeleným pozadím. Prázdne políčko je označené bielym pozadím.

V hlavnom okne aplikácie je možné zadať operácie v textovej podobe. Tento text sa pri zadaní testuje a zafarbuje podľa validity, ako je vidno na obrázku 5.10.



Obr. 5.10: Textový zápis operácii, v hlavnom okne aplikácie

Ak je zadaný text valídny, zafarbí sa na čierno. Naopak ak je v niektorej operácii chyba, či už syntaktická alebo sémantická, zafarbí sa na červeno. Viac o možnostiach pridávania, úpravy a odoberanie objektov je v časti 5.3.

Kapitola 6

Vyhodnotenie vytvoreného riešenia

Vytvorený systém bolo treba po jeho implementácii overiť. K tomuto účelu som vytvoril niekoľko príkladov. Na tieto príklady som vzal niekoľko geometrických tvarov od jednoduchších až po zložitejšie. S každým geometrickým tvarom som vykonal následujúce operácie:

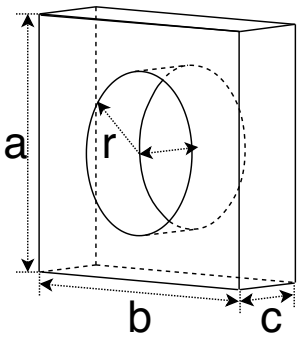
- Vytvorenie parametrického modelu daného tvaru
- Overenie funkčnosti parametrizácie daného tvaru
- Overenie zobrazovania daného tvaru
- Overenie že systém nespadne
- Obecný test softvéru, či pracoval korektne z pohľadu využitia prostriedkov a kompatibility

Pre jednoduchosť som testy vykonal iba na jednom druhu počítača:

- OS: Windows 10 (1909)
- CPU: i5-8300H
- RAM: 8GB
- GPU: GTX 1050

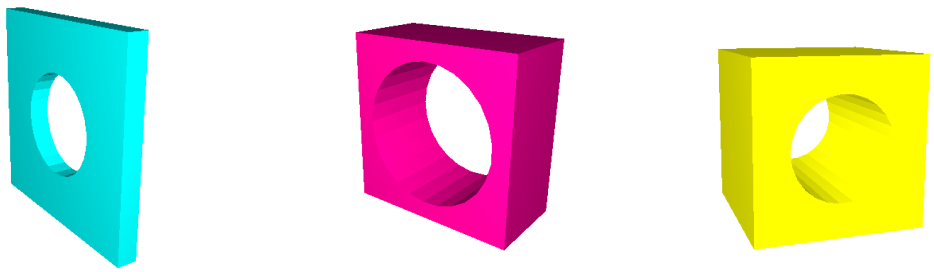
Príklad 1 - Kváder s kruhovým výrezom

Základom pre tento príklad bol model kvádra s dierou, zobrazený aj na obrázku 6.1.



Obr. 6.1: Model kvádra s dierou

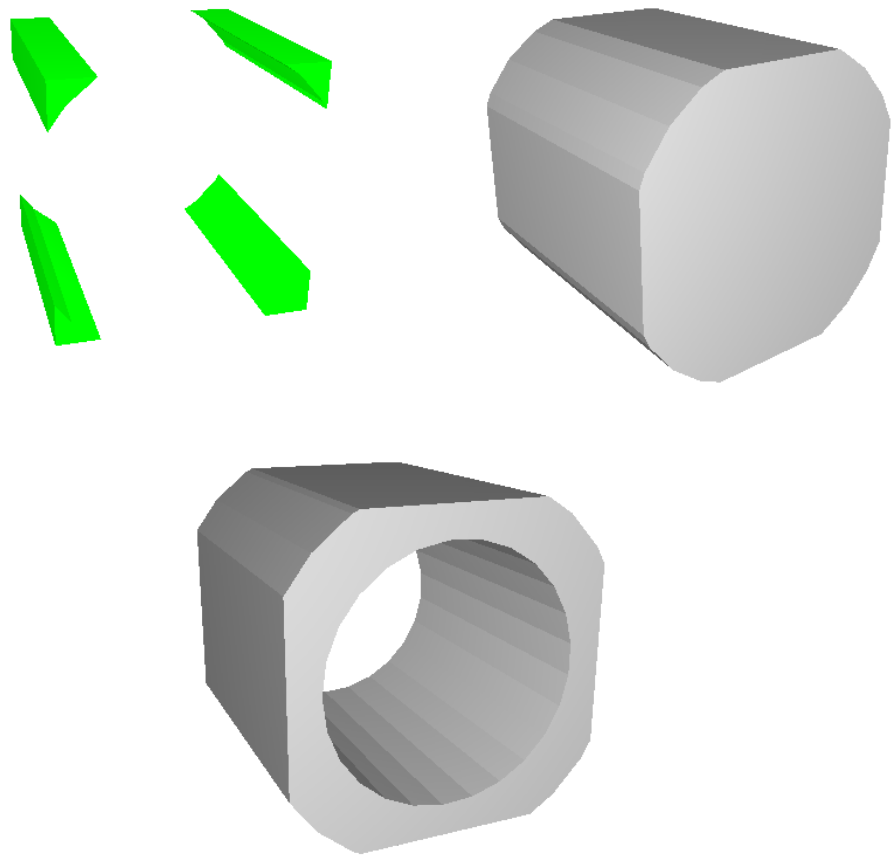
Výsledný model kvádra s dierou, vytvorený pomocou implementovaného systému je zobrazený na obrázku 6.2.



Obr. 6.2: Model "kváder s dierou"pri rôznych hodnotách parametrov

Tento navrhnutý model je vytvorený pomocou nasledujúcej množiny operácii:

```
Point(center,0,0,0,00000000);
Point(endPoint,0,0,1,00000000);
Line(vector,center,endPoint,00000000);
LinearInterpolationDist(centerWithOffset,center,endPoint,-0.5:offset
    ,00000000);
Circle(kruh,centerWithOffset,2.5:polomer kruhu,vector,00FF0000);
Rectangle(stvorec,center,5,5,0,vector,FF000000);
Extrude(kvader,stvorec,1:sirka,00000000);
Extrude(diera,kruh,sirka-offset*2,00000000);
BooleanMinus(kvader s~kruhovou dierou,kvader,diera,0000FFFF);
```

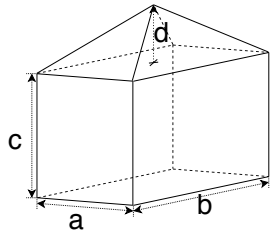


Obr. 6.3: Modifikácie vytvoreného modelu - a)Diera prechádza cez okraje kvádra, b)Kváder z pôvodného modelu minus `a`, c)Model `b` minus kruh s menším polomerom

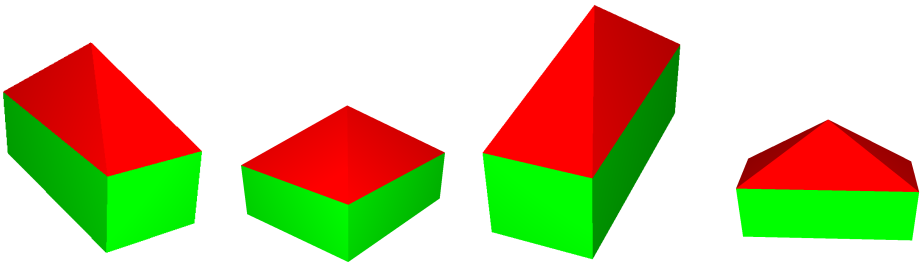
Navrhnutý model je možné ďalej modifikovať. Ďalšie vytvorené modifikácie sú zobrazené na obrázku 6.3.

Príklad 2 - Model domu

Tento príklad je implementovaný na priloženom médiu a zobrazuje vytvorený model s použitím grafického okna vytvoreného pomocou knižnice glfw. Model použitý v tejto aplikácii je model domu, ktorý je navrhnutý v časti 4.4.



Obr. 6.4: Návrh modela domu



Obr. 6.5: Model "Dom"pri rôznych hodnotách parametrov

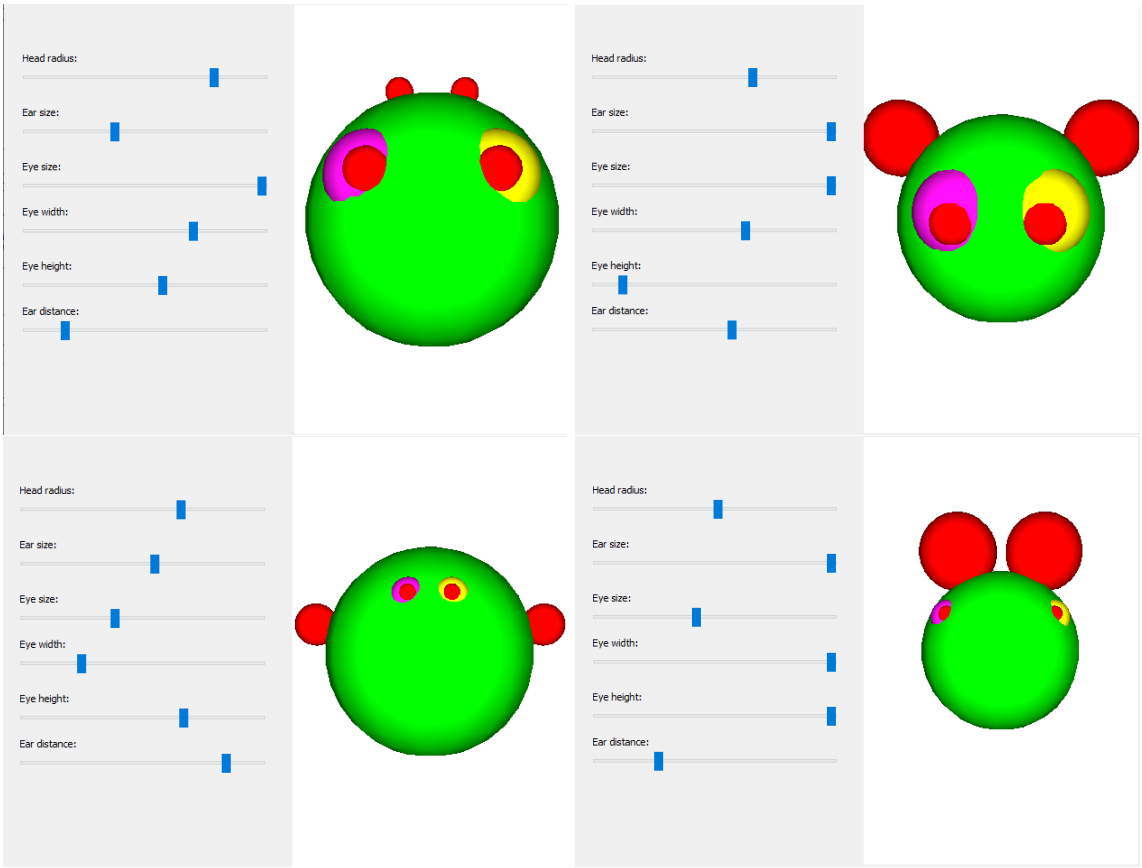
V návrhu vyplýva, že sa musí dať vytvorený model aj uložiť a po načítaní ďalej upravovať, napríklad pridávaním ďalších operácií. To je zobrazené na nasledovnej ukážke:

```
1   ParametricModel paramModel;
2
3   paramModel.AddOperations(R"(
4         Point(HouseTopCenter,0,2:houseHeight,0,00000000);
5         Point(PointUP,0,1,0,HouseTopCenter,00000000);
6         Line(LineUP,HouseTopCenter,PointUP,00000000);
7         Rectangle(Top,HouseTopCenter,3:houseX,2+sin(time/1500):houseY,
             sin(time/1000)-0.2:rotation,LineUP,00000000);
8         Extrude(House,Top,0-houseHeight,00FF00FF))"); //pridanie množiny
             operácii na tvorbu základov domu (zelená časť domu)
9   paramModel.BuildModel();
10
11  paramModel.Save("Output.parammodel"); //model je možné uložiť
12  paramModel.DeleteModel(); //model je možné zmazať
13  paramModel.Load("Output.parammodel"); //uložený model je možné znova
       načítať
14  paramModel.AddOperation("Cone(Roof,Top,2:roofHeight,FF0000FF)");
       //pridanie operácie do načítaného modelu (strecha-červená časť domu)
15  paramModel.BuildModel();
```

Hore uvedený kód aplikácie ukazuje vytvorenie modelu, pridávanie operácií, uloženie a načítanie modelu. Vo vykresľovacej slučke sú následne volané metódy `BuildModel` a `Draw`.

Príklad 3. Aplikácia s použitím Qt

Táto ukážka zobrazuje použitie vytvorenej knižnice s aplikáciou vytvárajúcou okno pomocou knižnice Qt.



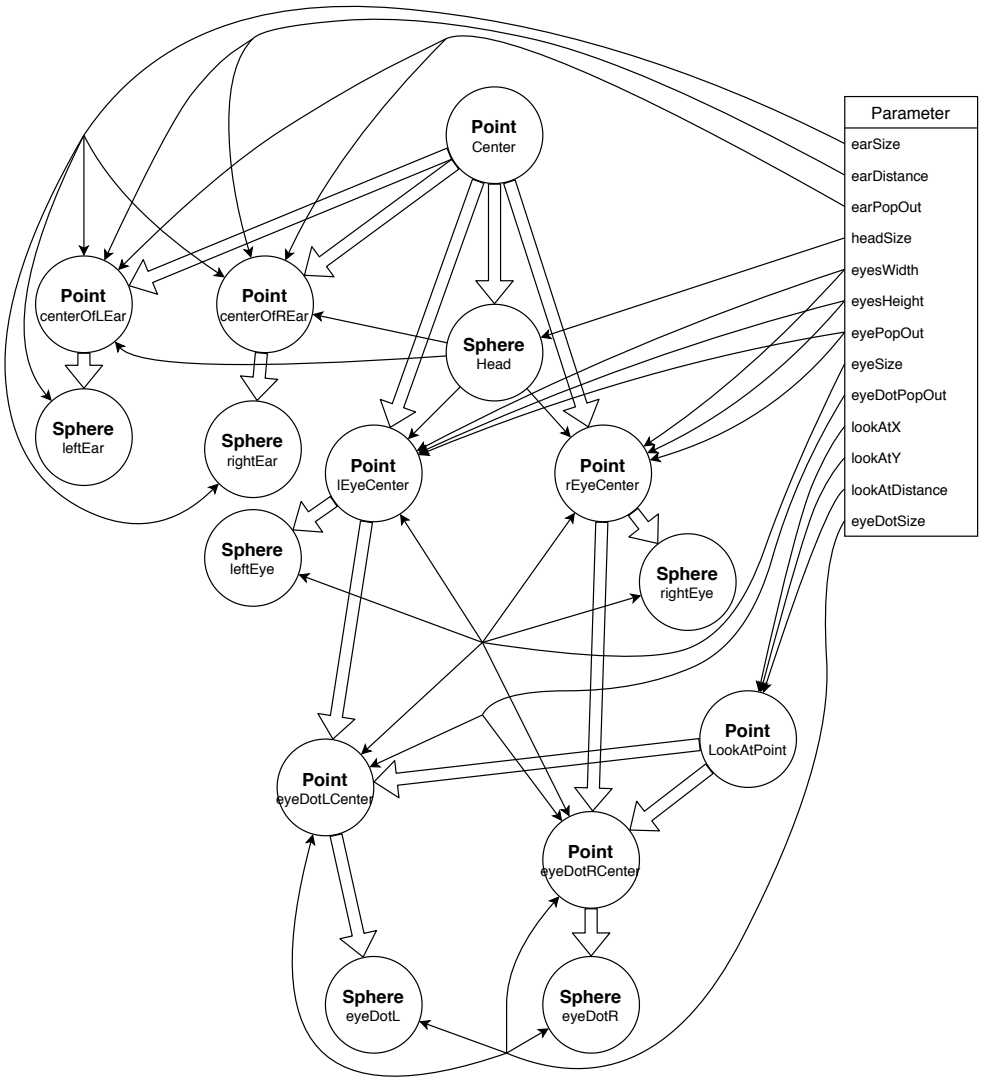
Obr. 6.6: Model "Hlava"pri rôznych hodnotách parametrov

Okno aplikácie obsahuje niekoľko sliderov na manipuláciu s modelom. Pomocou týchto sliderov sa v modele "Hlava", dá upravovať niekoľko parametrov hlavy. Dá sa upraviť napríklad veľkosť hlavy, veľkosť uší alebo veľkosť očí. Taktiež sa dá upravovať vzdialenosť medzi ušami alebo pozíciu očí. Oči modelu sú tiež pohyblivé, teda sa dá nastaviť pozíciu, na ktorú sa bude model pozerať. To je v aplikácii ovládané pomocou ťahu pravého tlačidla myši, po okne aplikácie.

Grafické rozhranie Qt, je tvorené pomocou widgetov. V kóde aplikácie, je v inicializácii vykresľovacieho widgetu QOpenGLWidget, volaná inicializácia `Rendereru` knižnice a nastavenie farby pozadia. Pri každom snímku vykresľovania, sa model znovu postaví a nastaví sa pozícia kamery a jej natočenie okolo modelu. Následne je tento model vykreslený do grafického widgetu aplikácie.

```
void initializeGL() {
   paramModel->InitRenderer();
   paramModel->SetBackgroundColor(1, 1, 1, 1);
   ...
}
void paintGL() {
  paramModel->BuildModel();
  paramModel->SetRendererCameraRotation(Pitch, Yaw, 0);
  paramModel->SetRendererCameraPosition(positionX, positionY, distance);
  paramModel->Draw(0,0,45,width(), height());
}
```

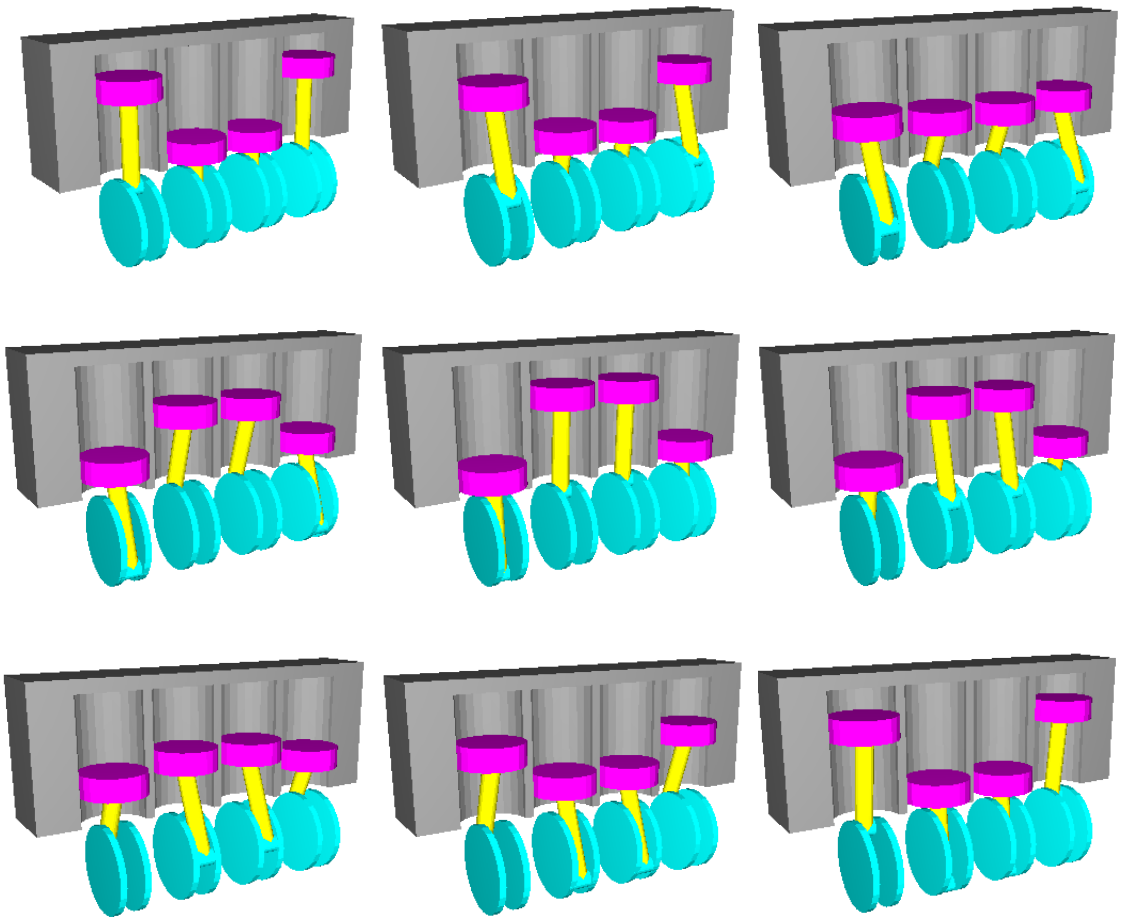
Parametrické modely sú tvorené rôznymi operáciami, ktoré na seba nadväzujú. Tieto operácie sa dajú zobraziť aj pomocou acyklického grafu operácií. Množina operácií, tvoriace model "Hlava", je v prílohe D. Grafové zobrazenie tohto modelu, je na obrázku 6.7.



Obr. 6.7: Grafické zobrazenie operácii - Hlava

Príklad 4 - Animácia motora

Najzložitejší model, ktorý som vytvoril pomocou tohoto systému, je model motora. Tento model je navrhnutý tak, aby sa dal animovať v čase a obsahuje rôzne parametre ako je veľkosť hlavy, rádius hriadeľa a aj rýchlosť motora.



Obr. 6.8: Animácia modelu motora

Tento model je vytvorený pomocou 93 operácií, ktoré sú zhrnuté v prílohe E.

Výsledok vyhodnotenia

Pri všetkých štyroch príkladoch, či už boli príklady zložité alebo jednoduché, model objektu sa podarilo vytvoriť, bol správne zobrazený, overil som jeho parametrizáciu a neobjavili sa problémy s kompatibilitou. Pri tržnom využití by bolo potrebné vyskúšať na viacerých zariadeniach s rôznymi grafickými kartami a bolo by potreba vyskúšať na viacerých modeloch.

Kapitola 7

Záver

Cieľom práce bolo navrhnúť a vytvoriť systém na tvorbu parametrických modelov. Tento cieľ práce sa podarilo splniť.

Naštudoval som literatúru na tému parametrických modelov, existujúce riešenia a geometrické operácie. Výsledok tejto štúdie je v kapitole 2 a 3. Navrhnutý systém obsahuje knižnicu na tvorbu parametrických modelov a aplikáciu využívajúcu vytvorenú knižnicu, ktorá uľahčuje návrh parametrických trojrozmerných modelov. Vytvorená knižnica umožňuje vytváranie modelov pomocou geometrických operácií. Väzby medzi jednotlivými objektami sú vytvárané priamo, teda operácie naviazané na iné objekty, alebo aj pomocou prepojenia vo výrazoch. Vytvorené modely je možné voľne upravovať, nastavovať parametre modelu, a tiež zobraziť v grafickom rozhraní. Príklady využitia tohoto implementovaného systému sú opísané v kapitole 6.

Výsledný softvér, ktorý som vytvoril, obsahuje 8500 riadkov zdrojového kódu, umožňuje spracovávať štyri druhy geometrických objektov a obsahuje 40 geometrických operácií. Jazyk, pomocou ktorého systém vytvára parametrický model, je zameraný na vytváranie objektov z operácií, kde každá operácia vytvára jeden objekt.

Tento jazyk by som do budúcna chcel rozšíriť o možnosť cyklenia. Do budúcna by som sa chcel zamerať aj na dokončení priehľadnosti objektov. Táto možnosť je už z časti implementovaná. Je možné jednotlivým objektom nastaviť farbu aj s priehľadnosťou, ale pre zameranie práce na iné časti, zatiaľ nie sú tieto objekty vykresľované ako priehľadné. Tiež je vhodné pridanie textúr na vytvorené objekty. Ďalej by som chcel pridať nahrávanie animácie, kde by si užívateľ mohol nastaviť rôzne nastavenia, ako je rozmer, kódek alebo dĺžka nahrávania, a tiež by mohol medzi jednotlivými snímkami nastavovať parametre modelu alebo model úplne prerobiť pridaním alebo odobraním operácií.

Literatúra

[1] BOSE, S.: *MATHEMAICS FORMULAE*. V&S Publishers, 2015, ISBN 9789352151202.
URL https://books.google.sk/books?id=k6E3DwAAQBAJ

[2] Bourke, P.: Point, Line, Plane. 1988–1998.
URL http://paulbourke.net/geometry/pointlineplane/

[3] Davis, D.: *Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture*. Dizertačná práca, RMIT University, 2013.

[4] Feuerbach, K. W.; Buzengeiger, C. H. I.: Eigenschaften einiger merkwürdigen Punkte des geradlinigen Dreiecks und mehrerer durch sie bestimmten Linien und Figuren. http://resolver.sub.uni-goettingen.de/purl?PPN512512426, 1822.

[5] Frazer, J.: Parametric Computation: History and Future. *Architectural Design*, ročník 86, č. 2, 2016: s. 18–23, doi:10.1002/ad.2019, https://onlinelibrary.wiley.com/doi/pdf/10.1002/ad.2019.

[6] Gaget, L.: Top 8 of the best parametric modeling software. 2018.
URL https://www.sculpteo.com/blog/2018/03/07/top-8-of-the-best-parametric-modeling-software/

[7] van Havre, Y.: *About FreeCAD*. Creative Commons Attribution, 2015.
URL https://www.freecadweb.org/wiki/About_FreeCAD

[8] Horvath, J.; Cameron, R.: *3D Printed Science Projects: Ideas for your classroom, science fair or home*. Apress, 2016, ISBN 9781484213230.
URL https://books.google.sk/books?id=dJsqDAAAQBAJ

[9] Jacobson, A.; Panozzo, D.; aj.: libigl: A simple C++ geometry processing library. 2018.
URL http://libigl.github.io/libigl/

[10] Keeter, M.: libfive. 2019.
URL https://libfive.com/about/

[11] Kelley, W.: *The Humongous Book of Geometry Problems*. Humongous Books, DK Publishing, 2013, ISBN 9781615646982.
URL https://books.google.sk/books?id=3XSjAQAAQBAJ

[12] Kimberling, C.: ENCYCLOPEDIA OF TRIANGLE CENTERS. 2019.
URL http://faculty.evansville.edu/ck6/encyclopedia/ETCPart16.html

[13] Page, J.: Incenter coordinates of a triangle given the vertex coordinates (Coordinate Geometry) - Math Open Reference. 2011.
URL https://www.mathopenref.com/coordincenter.html

[14] Recrosio, E.: Save Time with Parametric Design for 3D Printing. 2017.
URL https://www.sculpteo.com/blog/2017/05/10/save-time-with-parametric-design-for-3d-printing/

[15] Siggel, K. J. S. T. e. a., M.: *TiGL: An Open Source Computational Geometry Library for Parametric Aircraft Design.* Math.Comput.Sci., 2019.
URL https://doi.org/10.1007/s11786-019-00401-y

[16] Weisstein, E. W.: Kimberling Center – from Wolfram MathWorld. 2004.
URL http://mathworld.wolfram.com/KimberlingCenter.html

[17] Woodbury, R.: *Elements of Parametric Design.* Routledge, 2010, ISBN 9780415779876.
URL https://books.google.sk/books?id=Zho5QAAACAAJ

[18] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika.* Brno: Computer Press, 2004, ISBN 9788025104545.
URL https://books.google.cz/books?id=USQnAAAACAAJ

Príloha A

Obsah priloženého média

- **Models/** - Vytvorené modely a obrázky
- **Release/** - Spustiteľné súbory
  - ☐ **ThesisLibrary/**
    - **Thesis.lib** - Knižnica na tvorbu parametrických modelov
  - ☐ **ParametricModel_Editor/...** - Aplikácia navrhnutá na tvorbu parametrických modelov
  - ☐ **GLFW_Application/...** - Aplikácia s použitím glfw s modelom z príkladu 6.2
  - ☐ **QtProgram/...** - Aplikácia s použitím Qt z príkladu 6.3
- **Source/** - Zdrojové kódy
  - ☐ **ExternalLibraries/..** - Použité externé knižnice
  - ☐ **Thesis/..** - Zdrojový kód navrhnutého riešenia
- **latex/...** Projekt textovej časti
- Parametricke_3D_modely.pdf
- README.txt

Príloha B

Rozhranie navrhnutého systému

Programátorské rozhranie navrhnutého systému:

```
1  std::vector <Operation*> OperationsVec; //Operations stack
2  std::map<std::string, Operation*> OperationMap; //Map for easier acces to
       operations
3
4  std::vector <Object::GeometricObject*> Objects; //Generated objects of
       parametric model
5  std::map<std::string, Object::GeometricObject*> ObjectMap; // Map for
       objects of parametric model
6
7  ParamRef paramRef //Model parameter referencies
8  ParametricModel() //Constructor
9  ~ParametricModel() //Destructor
10
11 Object::GeometricObject* GetObject(std::string objectName)
12 void BuildModel() //Generate model objects from operations. Call this to
       apply modifications
13 void RemoveOperation(size_t index) //Remove operation from specific index
14 void AddOperation(Operation *c) //Add operation to model
15 void InsertOperation(size_t index,Operation *c) //Insert operation to
       specific position
16 void DeleteModel() //Reset model to default state
17 bool AddOperations(std::string s) //Add multiple operations to model
18 bool AddOperation(std::string s) //Add operation to model
19
20 void resetTest() //Test initializer. Need to be called before every test
21 bool TestOperation(std::string s) //Test if operation is valid
22
23 bool AddReferenceParam(std::string ParamRefName, std::string refObjectName,
        size_t paramIndex) //Add model parameter
24 bool AddReferenceParamWithoutValidCheck(std::string ParamRefName, std::
       string refObjectName, size_t paramIndex)
25 bool TestRefParam(std::string ParamRefName, std::string refObjectName,
       size_t paramIndex) //Test if parameter can be added to object
```

```
void RemoveReferenceParam(std::string ParamRefName) //Remove model parameter
void RemoveReferenceParam(std::string refObjectName, size_t paramIndex) //Remove model parameter from specific object
void ReplaceOperation(size_t index, Operation *c) //Replace operation

void SetRefValue(std::string ObjectName, size_t paramindex, std::string value) //Set value of operation parameter
void SetRefValue(std::string RefName,float value) //Set value of parameter

double GetObjectValue(std::string s,bool* Err = NULL) //Get value in string form

void InitRenderer() //Initialize renderer
void Draw(int x, int y, float fov, int width, int height) //Draw with OpenGL

void SetRendererCameraPosition(float X, float Y, float Z)
void SetRendererCameraRotation(float Pitch, float Yaw, float
void setRendererAmbientStrength(float ambientStrength)

void Save(std::string filePath) //Save parametric model to file
bool Load(std::string filePath) //Load parametric model to file
void SaveOBJ(std::string filePath) //Export model to OBJ file

void setTime(unsigned long miliseconds) //Set timer to selected time
void useManualTimer(bool enable) //Enable or disable timer
void SetBackgroundColor(float R, float G, float B, float A)
void SetBackgroundColor(signed char R, signed char G, signed char B, signed char A)
```

Príloha C

Zoznam geometrických operácií

Geometrické operácie, z ktorých sa skladajú parametrické 3D modely. Tieto operácie sa začínajú názvom operácie, nasledované parametrami v zátvorkách, ktoré sú oddelené čiarkou.

Bodové operácie

- **Point(string name, expression X, expression Y, expression Z, RRGGBBAA color)** - Bod s absolútnou pozíciou
- **Point(string name, expression X, expression Y, expression Z, Point parent, RRGGBBAA color)** - Bod s relatívnou pozíciou of zadaného bodu
- **LinearInterpolationDist(string name, Point from, Point to, expression distance, RRGGBBAA color)** - Bod medzi dvomi bodmi, so vzdialenosťou od prvého bodu
- **LinearInterpolationPerc(string name, Point from, Point to, expression percentage, RRGGBBAA color)** - Bod medzi dvomi bodmi, s percentuálnou vzdialenosťou od prvého bodu
- **Intersection_Plane_Line(string name, Line l, Sufrace s, RRGGBBAA color)** - Bod v pozícii, kde úsečka pretína plochu
- **SurfaceCenterBoundingSquare(string name, Surface s, RRGGBBAA color)** - Bod v centre plochy
- **SurfaceCenterAverage(string name, Surface s, RRGGBBAA color)** - Bod roviny vytvorený spriemerovaním všetkých bodov plochy
- **Centroid(string name, Triangle t, RRGGBBAA color)** - Bod v ťažisku trojuholníka
- **Incenter(string name, Triangle t, RRGGBBAA color)** - Bod v centre vpísaného kruhu trojuholníka
- **Circumcenter(string name, Triangle t, RRGGBBAA color)** - Bod v centre opísaného kruhu
- **Orthocenter(string name, Triangle t, RRGGBBAA color)** - Ortocentrum trojuholníka

- **NinePointCenter(string name, Triangle t, RRGGBBAA color)** - Stred kruhu deviatich bodov trojuholníka
- **ObjectCenterBoundingBox(string name, Object3D o, RRGGBBAA color)** - Stred ohraničujúceho kvádra objektu
- **ObjectCenterAverage(string name, Object3D o, RRGGBBAA color)** - Vážený stred objektu, tvorený spriemerovaním všetkých bodov objektu
- **LineFirstPoint(string name, Line l, RRGGBBAA color)** - Počiatočný bod úsečky
- **LineSecondPoint(string name, Line l, RRGGBBAA color)** - Koncový bod úsečky

Úsečkové operácie

- **Line(string name, Point p1, Point p2, RRGGBBAA color)** - Vytvorenie úsečky, kde p1 je počiatočný bod a p2 je koncový bod
- **LineNormalize(string name, Line l, RRGGBBAA color)** - Normalizujte čiaru na vzdialenosť 1,0. Počiatočný bod je rovnaký ako počiatočný bod zadanej čiary, konečný bod je vo vzdialenosti 1,0 v smere koncového bodu čiary.
- **LineChangeLengthDist(string name, Line l, expression distance, RRGGBBAA color)** - Počiatočný bod je rovnaký ako počiatočný bod zadaného riadku, koncový bod je v zadanej vzdialenosti v smere k koncovému bodu riadku
- **LineChangeLengthPerc(string name, Line l, expression percent, RRGGBBAA color)** - Počiatočný bod je rovnaký ako počiatočný bod zadaného riadku, koncový bod je v zadanej percentuálnej vzdialenosti v smere k cieľovému bodu riadku
- **MinLineBetweenLineAndLine(string name, Line l1, Line l2, RRGGBBAA color)** - Najkratšia úsečka medzi dvomi priamkami
- **MinLineBetweenPointAndLine(string name, Point p, Line l, RRGGBBAA color)** - Najkratšia úsečka medzi bodom a priamkou
- **MinLineBetweenPointAndSurface(string name, Point p, Surface s, RRGGBBAA color)** - Najkratšia úsečka medzi bodom a plochou
- **SurfaceNormal(string name, Surface s, RRGGBBAA color)** - Normála plochy
- **LineRelocationByPoint(string name, Line l, Point p, RRGGBBAA color)** - Presun úsečky na novú pozíciu. Začiatočný bod je zadaný bod a koncový bod je vo vzdialenosti zadanej čiary rovnakým smerom.
- **CrossProduct(string name, Line l1, Line l2, RRGGBBAA color)** - Úsečka v smere vektorového súčinu dvoch priamok

Plošné operácie

- **RectangleFromLine(string name, Line l, expression width, Point pointOnSurface, type, RRGGBBAA color)** - Vytvorenie obdĺžniku podľa úsečky, šírky obdĺžniku a bodu na ploche
- **RectangleFromLine(string name, Line l, expression width, Line normal, type, RRGGBBAA color)** - Vytvorenie obdĺžniku podľa úsečky, šírky obdĺžniku a normály od plochy
- **Circle(string name, Point center, expression radius, Line lineNormal, RRGGBBAA color)** - Vytvorenie kruhu podľa zadaného stredu, priemeru a normály
- **Circle(string name, Point center, Point outlinePoint, Point planePoint, RRGGBBAA color)** - Vytvorenie kruhu od stredu, bodu na okraji kruhu a bodu na ploche
- **Triangle(string name, Line l, Point p, RRGGBBAA color)** - Vytvorenie trojuholníku z úsečky a bodu
- **Triangle(string name, Point p1, Point p2, Point p3, RRGGBBAA color)** - Vytvorenie trojuholníku z troch bodov
- **Rectangle(string name, Point center, expression X, expression Y, expression Roll , Line normal, RRGGBBAA color)** - Vytvorenie obdĺžniku
- **Polygon(string name, Point p1, Point p2, Point p3, ..., RRGGBBAA color)** - Vytvorenie polygónu z ľubovoľného počtu bodov
- **Circumscribed(string name, Triangle t, RRGGBBAA color)** - Vytvorenie opísanej kružnice trojuholníka
- **Inscribed(string name, Triangle t, RRGGBBAA color)** - Vytvorenie vpísanej kružnice trojuholníka

Objemové operácie

- **Cone(string name, Surface s, expression distance, RRGGBBAA color)** - Vytvorenie kužeľu s podstavou z ľubovoľného plošného útvaru so zadanou výškou
- **Cone(string name, Surface s, Point p, RRGGBBAA color)** - Vytvorenie kužeľu s podstavou z ľubovoľného plošného útvaru a zadaným vrcholom
- **Extrude(string name, Surface s, expression distance, RRGGBBAA color)** - Vysunutie plochy do priestoru
- **Sphere(string name, Point center, expression radius, RRGGBBAA color)** - Guľa zo zadaného bodu a priemeru
- **BooleanUnion(string name, Object o1, Object o2, RRGGBBAA color)** - Zjednotenie objektov

- **BooleanIntersection(string name, Object o1, Object o2, RRGGBBAA color)** - Prienik objektov
- **BooleanMinus(string name, Object o1, Object o2, RRGGBBAA color)** - Rozdiel objektov
- **BooleanXOR(string name, Object o1, Object o2, RRGGBBAA color)** - Symetrická diferencia objektov

Príloha D

Množina operácii, tvoriace model hlavy.

Z týchto operácii je tvorený model hlavy, ktorý je zobrazený na obrázku 6.6.

```
Point(center,0,0,0,00000000);
Sphere(head,center,2:headSize,00FF00FF);
Point(param1,0.5:earSize,3.14/4:earDistance,0.7:earPopOut,00000000);
Point(centerOfLEar,
        0-(sin(earDistance)*(head.radius+earSize*earPopOut)),
        cos(earDistance)*(head.radius+earSize*earPopOut),
        0,center,0000FFFF);
Point(centerOfREar,
        sin(earDistance)*(head.radius+earSize*earPopOut),
        cos(earDistance)*(head.radius+earSize*earPopOut),
        0,center,0000FFFF);
Sphere(leftEar,centerOfLEar,earSize,FF0000FF);
Sphere(rightEar,centerOfREar,earSize,FF0000FF);
Point(paramEyes,3.14/8:eyesWidth,3.14/8:eyesHeight,0.7:eyePopOut,00000000);
Point(paramEyes2,0.2:eyeSize,0,0,00000000);
Point(lEyeCenter,
        sin(eyesWidth)*(head.radius+eyeSize*(0-eyePopOut)),
        cos(eyesWidth)*(head.radius+eyeSize*(0-eyePopOut))*sin(eyesHeight),
        cos(eyesHeight)*(head.radius+eyeSize*(0-eyePopOut))*cos(eyesWidth),
        FF00FFFF);
Point(rEyeCenter,
        0-sin(eyesWidth)*(head.radius+eyeSize*(0-eyePopOut)),
        cos(eyesWidth)*(head.radius+eyeSize*(0-eyePopOut))*sin(eyesHeight),
        cos(eyesHeight)*(head.radius+eyeSize*(0-eyePopOut))*cos(eyesWidth),
        0000FFFF);
Sphere(leftEye,lEyeCenter,eyeSize,FFFF00FF);
Sphere(rightEye,rEyeCenter,eyeSize,FF0FF0FF);

Point(param2,0.7:eyeDotPopOut,eyeSize/2:eyeDotSize,0,00000000);
Point(LookAtPoint,0:lookAtX,0:lookAtY,10:lookAtDistance,00000000);
```

```
LinearInterpolationDist(eyeDotLCenter,lEyeCenter,LookAtPoint,eyeSize-
    eyeDotSize*eyeDotPopOut,00000000);
Sphere(eyeDotL,eyeDotLCenter,eyeDotSize,FF0000FF);
LinearInterpolationDist(eyeDotRCenter,rEyeCenter,LookAtPoint,eyeSize-
    eyeDotSize*eyeDotPopOut,00000000);
Sphere(eyeDotR,eyeDotRCenter,eyeDotSize,FF0000FF);
```

Príloha E

Operácie tvoriace model motora

Z týchto operácii je tvorený model hlavy, ktorý je zobrazený na obrázku 6.8.

```
Point(parameters,1.7:offset,0.5:cylinder_height,0.75:cylinder_radius
    ,00000000);
Point(parameters1,2:piston_height,0.2:piston_radius,0,00000000);
Point(parameters2,0.3:Upperdeadcenterpiston_height,2,0.5:A_radius,
    00000000);
Point(parameters3,50:speed,2:ASDF,0.15:circle,00000000);
Point(parameters4,speed*(0-time)/10000:rotation,2,0,00000000);
Point(base,0,-1,0,00000000);
Point(pointUP,0,1,0,base,00000000);
Line(lineUP,base,pointUP,00000000);
Point(engine_cylinder0,offset*1.5,0,0,base,00000000);
Point(engine_cylinder1,offset*0.5,0,0,base,00000000);
Point(engine_cylinder2,offset*-0.5,0,0,base,00000000);
Point(engine_cylinder3,offset*-1.5,0,0,base,00000000);
Point(pointLeft0,piston_radius,0,0,engine_cylinder0,00000000);
Point(pointRight0,0-piston_radius,0,0,engine_cylinder0,00000000);
Point(pointLeft1,piston_radius,0,0,engine_cylinder1,00000000);
Point(pointRight1,0-piston_radius,0,0,engine_cylinder1,00000000);
Point(pointLeft2,piston_radius,0,0,engine_cylinder2,00000000);
Point(pointRight2,0-piston_radius,0,0,engine_cylinder2,00000000);
Point(pointLeft3,piston_radius,0,0,engine_cylinder3,00000000);
Point(pointRight3,0-piston_radius,0,0,engine_cylinder3,00000000);
Line(vectorRight,pointLeft0,base,00000000);
Line(vectorLeft,base,pointLeft0,00000000);
Circle(Base_circle0,pointLeft0,piston_height/2,vectorLeft,00000000);
Circle(Base_circle1,pointLeft1,piston_height/2,vectorLeft,00000000);
Circle(Base_circle2,pointLeft2,piston_height/2,vectorLeft,00000000);
Circle(Base_circle3,pointLeft3,piston_height/2,vectorLeft,00000000);
Circle(Base_circleR0,pointRight0,piston_height/2,vectorRight,00000000);
Circle(Base_circleR1,pointRight1,piston_height/2,vectorRight,00000000);
Circle(Base_circleR2,pointRight2,piston_height/2,vectorRight,00000000);
Circle(Base_circleR3,pointRight3,piston_height/2,vectorRight,00000000);
Extrude(BaseR0,Base_circle0, circle,00FFFFFF);
```

```
Extrude(BaseR1,Base_circle1, circle,00FFFFFF);
Extrude(BaseR2,Base_circle2, circle,00FFFFFF);
Extrude(BaseR3,Base_circle3, circle,00FFFFFF);
Extrude(BaseL0,Base_circleR0,circle,00FFFFFF);
Extrude(BaseL1,Base_circleR1,circle,00FFFFFF);
Extrude(BaseL2,Base_circleR2,circle,00FFFFFF);
Extrude(BaseL3,Base_circleR3,circle,00FFFFFF);
Point(wing0,0,cos(rotation)*(piston_height/2-piston_radius),sin(rotation)*(
    piston_height/2-piston_radius),engine_cylinder0,000000FF);
Point(wing1,0,cos(rotation+3.141598)*(piston_height/2-piston_radius),sin(
    rotation+3.141598)*(piston_height/2-piston_radius),engine_cylinder1
    ,000000FF);
Point(wing2,0,cos(rotation+3.141598)*(piston_height/2-piston_radius),sin(
    rotation+3.141598)*(piston_height/2-piston_radius),engine_cylinder2
    ,000000FF);
Point(wing3,0,cos(rotation)*(piston_height/2-piston_radius),sin(rotation)*(
    piston_height/2-piston_radius),engine_cylinder3,000000FF);
Point(CylinderBase0,0,(sqrt(piston_height^2-(wing0.Z-engine_cylinder0.Z)^2)
    )+wing0.Y-engine_cylinder0.Y,0,engine_cylinder0,00FF00FF);
Point(CylinderBase1,0,(sqrt(piston_height^2-(wing1.Z-engine_cylinder1.Z)^2)
    )+wing1.Y-engine_cylinder1.Y,0,engine_cylinder1,00FF00FF);
Point(CylinderBase2,0,(sqrt(piston_height^2-(wing2.Z-engine_cylinder2.Z)^2)
    )+wing2.Y-engine_cylinder2.Y,0,engine_cylinder2,00FF00FF);
Point(CylinderBase3,0,(sqrt(piston_height^2-(wing3.Z-engine_cylinder3.Z)^2)
    )+wing3.Y-engine_cylinder3.Y,0,engine_cylinder3,00FF00FF);
Circle(CylinderHead0_circle,CylinderBase0,cylinder_radius,lineUP,00000000);
Circle(CylinderHead1_circle,CylinderBase1,cylinder_radius,lineUP,00000000);
Circle(CylinderHead2_circle,CylinderBase2,cylinder_radius,lineUP,00000000);
Circle(CylinderHead3_circle,CylinderBase3,cylinder_radius,lineUP,00000000);
Extrude(CylinderHead0,CylinderHead0_circle,cylinder_height,FF00FFFF);
Extrude(CylinderHead1,CylinderHead1_circle,cylinder_height,FF00FFFF);
Extrude(CylinderHead2,CylinderHead2_circle,cylinder_height,FF00FFFF);
Extrude(CylinderHead3,CylinderHead3_circle,cylinder_height,FF00FFFF);
Line(piston_line0,wing0,CylinderBase0,00000000);
Line(piston_line1,wing1,CylinderBase1,00000000);
Line(piston_line2,wing2,CylinderBase2,00000000);
Line(piston_line3,wing3,CylinderBase3,00000000);
Circle(piston_circle0,wing0,piston_radius,piston_line0,00000000);
Circle(piston_circle1,wing1,piston_radius,piston_line1,00000000);
Circle(piston_circle2,wing2,piston_radius,piston_line2,00000000);
Circle(piston_circle3,wing3,piston_radius,piston_line3,00000000);
Extrude(piston0,piston_circle0,piston_height,FFFF00FF);
Extrude(piston1,piston_circle1,piston_height,FFFF00FF);
Extrude(piston2,piston_circle2,piston_height,FFFF00FF);
Extrude(piston3,piston_circle3,piston_height,FFFF00FF);
Point(crankshaft0__point,piston_radius+circle,0,0,wing0,00000000);
Point(crankshaft1__point,piston_radius+circle,0,0,wing1,00000000);
Point(crankshaft2__point,piston_radius+circle,0,0,wing2,00000000);
```

```
Point(crankshaft3__point,piston_radius+circle,0,0,wing3,00000000);
Circle(crankshaft0_circle,crankshaft0__point,piston_radius,vectorLeft,0000FF00);
Circle(crankshaft1_circle,crankshaft1__point,piston_radius,vectorLeft,0000FF00);
Circle(crankshaft2_circle,crankshaft2__point,piston_radius,vectorLeft,0000FF00);
Circle(crankshaft3_circle,crankshaft3__point,piston_radius,vectorLeft,0000FF00);
Extrude(crankshaft0_cylinder,crankshaft0_circle, 0-((piston_radius+circle)*2),00FFFFFF);
Extrude(crankshaft1_cylinder,crankshaft1_circle, 0-((piston_radius+circle)*2),00FFFFFF);
Extrude(crankshaft2_cylinder,crankshaft2_circle, 0-((piston_radius+circle)*2),00FFFFFF);
Extrude(crankshaft3_cylinder,crankshaft3_circle, 0-((piston_radius+circle)*2),00FFFFFF);
Point(blockCenter,0,piston_height/2,cylinder_radius,base,00000000);
Rectangle(blockBase,blockCenter,offset*2.5,cylinder_radius,0,lineUP,00000000);
Extrude(block,blockBase, cylinder_height+piston_height+Upperdeadcenterpiston_height*2,FFFFFF00);
Circle(CutOff0_circle,engine_cylinder0,cylinder_radius,lineUP,00000000);
Circle(CutOff1_circle,engine_cylinder1,cylinder_radius,lineUP,00000000);
Circle(CutOff2_circle,engine_cylinder2,cylinder_radius,lineUP,00000000);
Circle(CutOff3_circle,engine_cylinder3,cylinder_radius,lineUP,00000000);
Extrude(CutOff0,CutOff0_circle,cylinder_height+piston_height*1.5+Upperdeadcenterpiston_height,00000000);
Extrude(CutOff1,CutOff1_circle,cylinder_height+piston_height*1.5+Upperdeadcenterpiston_height,00000000);
Extrude(CutOff2,CutOff2_circle,cylinder_height+piston_height*1.5+Upperdeadcenterpiston_height,00000000);
Extrude(CutOff3,CutOff3_circle,cylinder_height+piston_height*1.5+Upperdeadcenterpiston_height,00000000);
BooleanUnion(union01,CutOff0,CutOff1,00000000);
BooleanUnion(union23,CutOff2,CutOff3,00000000);
BooleanUnion(union0123,union01,union23,0000FF00);
BooleanMinus(blockMinus,block,union0123,888888FF);
```