Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Predikce energetické náročnosti budovy na základě jejího CAD nákresu

2024

Bc. Václav Procházka

Vedoucí práce:
doc. Mgr. Jan Outrata, Ph.D.

Studijní program: Aplikovaná informatika,
Specializace: Počítačové systémy a technologie

Bibliografické údaje

Autor:	Bc. Václav Procházka
Název práce:	Predikce energetické náročnosti budovy na základě jejího CAD nákresu
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2024
Studijní program:	Aplikovaná informatika, Specializace: Počítačové systémy a technologie
Vedoucí práce:	doc. Mgr. Jan Outrata, Ph.D.
Počet stran:	40
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Bc. Václav Procházka
Title:	Prediction of buildings energy efficiency based on its CAD
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2024
Study program:	Applied Computer Science, Specialization: Computer Systems and Technologies
Supervisor:	doc. Mgr. Jan Outrata, Ph.D.
Page count:	40
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Diplomová práce představuje možnost konverze CAD nákresu budovy do voxelové reprezentace, v níž nadále ukazuje, jak volumetricky simulovat vedení tepla. Na základě tohoto výpočtu pak stanoví jak predikovat energetickou náročnost budovy pomocí účinnosti zdroje tepla a další spotřebované energie. Nakonec jsou diskutovány výsledky predikce.

Synopsis

Master's thesis shows how to convert CAD to voxel representation and how to simulate heat conduction in the representation. Then it shows how to predict energy efficiency of the building using efficiency of the heat source and other consumed energy. In the end the prediction results are discussed.

Klíčová slova: CAD; vedení tepla; volumetrická simulace; voxelová reprezentace; energetická náročnost budovy;

Keywords: CAD; heat conduction; volumetric simulation; voxel representation; energy efficiency;

Děkuji vedoucímu práce doc. Mgr. Janu Outratovi, Ph.D. za odborné rady při zpracovávání této práce. Dále děkuji celé své rodině za její podporu.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Návrhy budov, strojů a dalších produktů jsou v dnešní době stále častěji prováděny za pomocí specializovaných softwarů, jež se nazývají *CADy*. *CADy* obvykle obsahují informaci o materiálech, a tedy je možné je snadno využít pro *fyzikální simulace*, jež mohou pomoci s návrhem co nejvhodnějších produktů zaměřených na potřeby konkrétního zákazníka.

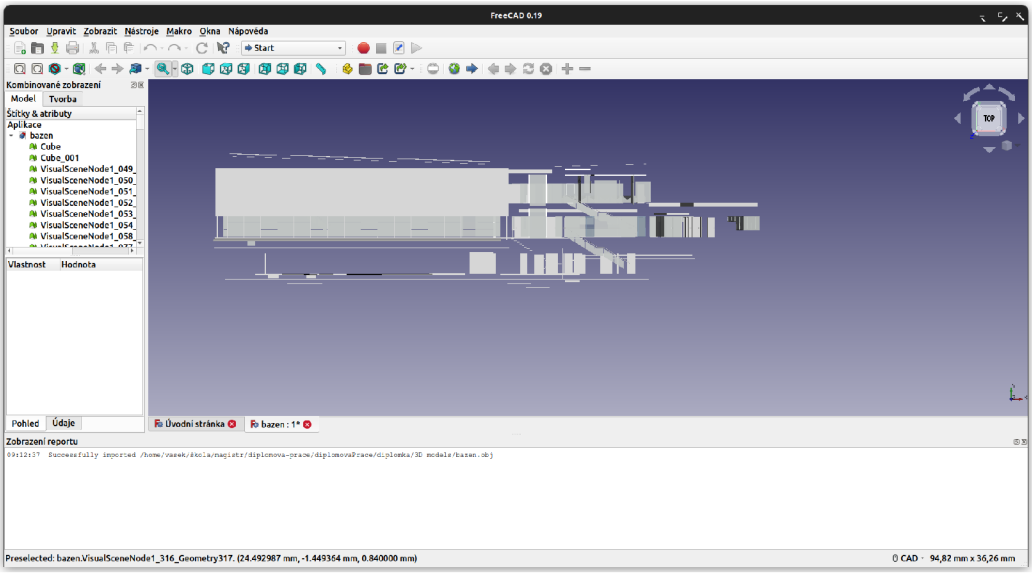
Fenoménem dnešní doby je snaha snižovat spotřebu *energie*, jak z důvodu energetických krizí, tak strategické energetické bezpečnosti a ochrany klimatu. V našich klimatických podmínkách se naprostá většina *energie* rezidenčních budov využívá na *vytápění*, jímž se detailně zabývá i tato diplomová práce.

Primárním cílem této diplomové práce je vytvořit *REST aplikaci*, jež bude poskytovat vhodné rozhraní pro nahrání *CAD souborů* a jejich *simulaci*. *Simulace* by především měla obsahovat *šíření tepla vedením*, jež je hlavním zdrojem *úniků tepla* přes zdi. *Simulace* by také měla být vytvořena ve *voxelové reprezentaci*, jež je obecná, rozšiřitelná a umožňuje rychlé výpočty. Nakonec by mělo být možné na základě této simulace a případně dalších dodatečných informací predikovat *energetickou náročnost* budovy.

Práce je rozdělená do čtyř hlavních částí. Nejprve se zaměří na konverzi CADu do voxelové reprezentace, kde čtenáře seznámí se základními pojmy CAD a voxelová reprezentace a popíše řešený problém. V druhé části se práce zaměří na popsání základních znalostí z fyziky šíření tepla potřebných pro pochopení kontextu práce, zejména se pak zaměří na šíření tepla vedením a zářením. Ve třetí části bude vysvětlen pojem energetická náročnost budovy z pohledu energetiky v souvislosti se vztažnou veličinou *EnPi*. Zde autor využil své znalosti nabyté ve firmě *ENSYTRA s.r.o.* poskytující energetický informační systém *Energybroker*. V poslední popisné části bude představena výsledná aplikace, jež je výsledkem této diplomové práce. Aplikace bude popsána z hlediska použitých technologií, její celkové architektury, obecného fungování a použití pomocí *API*. Na závěr bude práce diskutována, budou zde popsány její limity a možná rozšíření do budoucna a budou shrnuty konkrétní výsledky práce, tedy představení vlastností simulace a srovnání s běžnými bazény.

CAD model bazénu, využívaný pro demonstrační účely této práce, byl poskytnut architektem Alešem Burianem, dle reálně stojícího bazénu ve Znojmě. Pouze pro nekomerční a testovací účely této diplomové práce byly využita data o materiálech, jež jsou dostupná z https://thermtest.com/thermal-resources/materials-database.

Původním záměrem práce bylo vytvořit ji pro potřeby firmy *ENSYTRa*, ale vzhledem k autorovu odchodu z firmy, již nebylo možné získat nezávislé vyjádření představitelů této firmy.



Obrázek 1: Ukázka FreeCADu

2 Konverze CADu do voxelové reprezentace

V této kapitole je nejprve popsáno, co je to *CAD* a v jakých formátech jej lze ukládat. Dále je zde popsáno co je to *voxelová reprezentace*. A nakonec je celkově vysvětlen tento problém a jsou popsány možné přístupy k jeho řešení.

2.1 CAD

CAD, nebo také *Computer aided design*, je specializovaný software využívaný pro design strojů, budov a dalších produktů. Příklady *CADů* mohou být například ArchiCAD, jež se využívá pro potřeby architektů. Nebo například *Siemens NX*, jež umožňuje vytváření produktů a dále usnadňuje jejich uvedení do výroby, to se označuje jako CAM - *Computer aided manufacturing*. Na obrázku 1 můžete vidět příklad open-sourcového *CADu* - *FreeCAD*.

*F*ormáty, v nichž se CAD ukládá, lze obecně rozdělit na dvě kategorie:

1. *závislé na konkrétním softwaru* - tyto formáty lze spustit pouze v daném softwaru, příkladem takového formátu je *.pln*, jež je využívaný pouze v *ArchiCADu*. U tohoto typu formátů je obvyklé, že je potřeba za jejich specifikaci zaplatit.

2. *přenositelné* - tyto formáty lze spustit nezávisle na využitém softwaru, ale může dojít ke ztrátě některých informací při konverzi z předchozí kategorie. Přenositelné formáty jsou přesně specifikované. Příkladem takového formátu je *.obj*, nebo *.dae*.

Obrázek 2: Bazén zobrazený ve voxelové reprezentaci za použití voxelového enginu Minetest

2.2 Wavefront obj

Wavefront obj je textový formát vytvořený firmou *Wavefront technologies*. Je určený pro popis 3D objektů a spolu s rozšířením .mtl umožňuje i přenos informace o materiálech použitých v modelu.

Formát popisuje 3D objekty za pomocí pozicí *bodů* a *povrchů*. Dále může ukládat další data, jako pozice textur nebo normálové vektory. Materiály popisuje pomocí barev, odrazivosti a dalších optických vlastností.

2.3 Voxelová reprezentace

Voxel je 3D obdoba *pixelu*. Každý *voxel* může obsahovat různé *sémantické informace* - například o typu materiálů, barvě, teplotě a podobně dle domény, pro kterou se aktuálně tato reprezentace využívá. *Voxelová reprezentace* se využívá v počítačové grafice, počítačových hrách, modelování měst, geologii a dalších oblastech. Její *diskrétní* povaha umožňuje tvorbu rychlých simulací na *GPU*.

Obrázek 2 ukazuje jak může vypadat bazén ve *voxelové reprezentaci*.

2.4 Řešený problém

Na vstupu je dokument v *CAD formátu* a na výstupu je *voxelová reprezentace* obsahující sémantické informace o zvoleném materiálu pro potřebu naší oblasti využití. Existují dva hlavní způsoby, jak k problému přistupovat - vykreslit trojúhelníky nebo pro každý voxel zjistit, zda se nachází uvnitř oblasti.

Pseudokód 1 ukazuje, jak vypadá možné řešení problému pomocí vykreslení trojúhelníků. Hlavním nedostatkem tohoto řešení je, že výsledná voxelová repre-

zentace neobsahuje voxely, jež jsou uvnitř oblastí určených trojúhelníky. Časová složitost tohoto řešení je určená pouze počtem trojúhelníků, tedy $\Theta(T)$, kde T je počet trojúhelníků. V případě řešení pomocí voxelů, představeném v pseudokódu 2, je časová složitost závislá, jak na počtu voxelů, tak na počtu trojúhelníků. Časová složitost v tomto případě závisí na zvolené datové struktuře pro uložení trojúhelníků.

```
1   function triangle_conversion(triangles, voxel_size) {
2      let voxels = new VoxelRepresentation(voxel_size);
3      for(let triangle of triangles) {
4         draw_on_voxels(triangle, triangle.material, voxels);
5      }
6      return voxels;
7   }
```

Zdrojový kód 1: Pseudokód řešení problému pomocí vykreslení trojúhelníků

```
1   function voxel_conversion(triangles, voxel_size) {
2      let voxels = new VoxelRepresentation(voxel_size);
3
4      for(let [x,y,z] of voxels.indexes()) {
5         let [inside, material] = is_inside([x,y,z], triangles);
6         if(inside) {
7            voxels[x][y][z] = material;
8         }
9      }
10
11     return voxels;
12  }
```

Zdrojový kód 2: Pseudokód řešení problému pomocí voxelů

3 Fyzikální model šíření tepla

Teplota je průměrná kinetická energie vibrujících a kolidujících *atomů* v rámci látky a *teplo* je změna *energie* v rámci látek způsobená rozdílem v *teplotě*. *Teplo* se může *šířit* třemi základními způsoby - *vedením*, *zářením* a *prouděním*. V této kapitole budou popsány tyto základní typy *šíření tepla*. Obecně k *vedení* dochází v rámci jedné *látky* pomocí *atomových vazeb*. K *záření* dochází na rozhraní materiálů, kde materiál vyzařuje *elektromagnetické záření*. A k *proudění* dochází v *tekutinách* (*kapalinách* a *plynech*).

3.1 Vedením

K *šíření tepla vedením* dochází v rámci *jedné látky*. Vždy se teplo šíří od části látky s vyšší teplotou k části, kde je teplota nižší. Základními parametry, jež ovlivňují rychlost vedení u pevných látek jsou:

1. *síla vazeb* mezi *atomy*
2. *pravidelnost atomové struktury* - například *krystalická struktura*
3. přítomnost *volných elektronů*

Tedy například diamanty mají *vysokou vodivost* díky tomu, že mají *pravidelnou krystalickou strukturu*. A *kovy* mají vysokou *vodivost* díky tomu, že obsahují *volné elektrony*. U *tekutin* (*plynů* a *kapalin*) k *vedení* dochází kvůli *kolizím* jednotlivých *částic*. Z toho důvodu mají *tekutiny* obvykle nízkou *vodivost* - takové látce se říká *izolant*. Této skutečnosti se využívá při tvorbě materiálů, jež jsou izolanty - například *aerogel* nebo *skelná vata*.

Základní veličinou *vedení tepla* je *tepelná vodivost*, jež je definována pomocí základní podoby *Fourierova zákona*:

$$q = -k * \frac{T_2 - T_1}{L} \tag{1}$$

V rovnici značí q *tepelný tok* s jednotkou $\frac{W}{m^2}$, jež je odváděn při rozdílu teplot $T2 - T1$, délce prostoru s látkou, jež odděluje rozdíl teplot L a *tepelnou vodivostí* této látky k s jednotkou $\frac{W}{m*K}$. Zobecněná verze *Fourierova zákona* zohledňuje různé rozložení teplot v potenciálně vícedimenzionálním prostoru:

$$q(r,t) = -k * \nabla T(r,t) \tag{2}$$

v tomto případě je *tepelný tok* definován v závislosti na čase t a pozici r. T je rozložení tepla v látce na pozici r v čase t. k je zde opět *tepelná vodivost*.

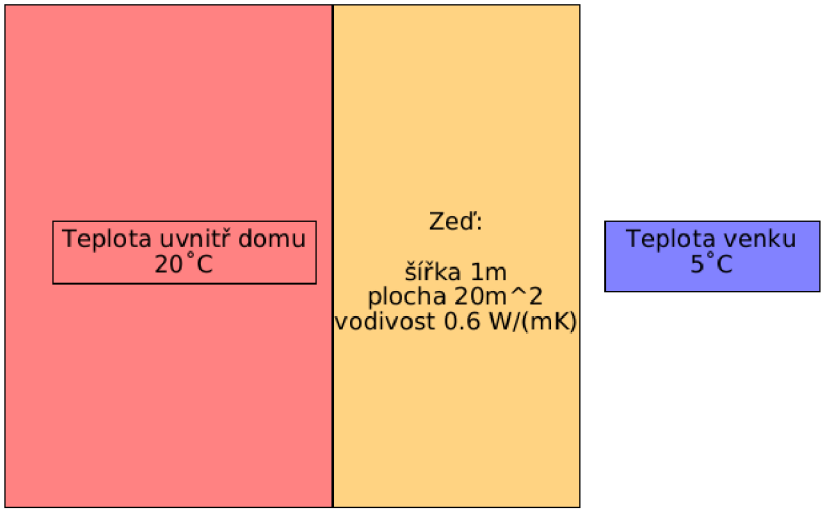
V praxi je možné využít *základní podobu Fourierova zákona* například pro výpočet *úniků tepla* přes zeď, při znalosti *tepelné vodivosti materiálů* z nichž se skládá. Tedy, kdybychom jako v ukázce na obrázku 3 měli cihlovou zeď o ploše $A = 20m^2$ širokou jeden metr s tepelnou vodivostí $k = 0.6$, přičemž teplota uvnitř budovy je $T_1 = 20°\text{C}$ a venkovní teplota je $T_2 = 5°\text{C}$, pak lze spočítat tepelný tok, jako

$$q = -0.6 * \frac{5 - 20}{1} = -0.6 * -15 = 9\frac{W}{m^2} \tag{3}$$

Z tohoto můžeme dále spočítat rychlost, jakou teplo uniká přes zeď vynásobení její plochou:

$$P = 9 * 20 = 180W \tag{4}$$

Toto v praxi může znamenat, že z budovy za hodinu, přes tuto zeď při konstantních vstupních podmínkách, unikne $E = 180 * 3600 = 648000J = 0.18kWh$ energie.



Obrázek 3: Vstupní podmínky ukázkového příkladu tepelné vodivosti

3.2 Zářením

Tepelné záření je *elektromagnetické záření*, jež má *vlnovou délku* $\lambda \in [10^{-1}, 10^{2}]\, m$, tedy od *ultrafialového záření* přes *viditelné spektrum* až po *infračervené záření*. Při pokojové teplotě je většina *záření* v *infračerveném spektru*, a tedy nelze vidět. Oproti tomu například hvězdy, jež mají velmi vysokou teplotu *vyzařují* i značnou část v *UV spektru* a *viditelném spektru*.

Základním modelem pro zkoumání *šíření tepla zářením* je tzv. *černé těleso*. *Černé těleso* se vyznačuje tím, že veškerou energii pohltí. Množství *vyzářené energie* za jednotku obsahu *černým tělesem* je určeno vzorcem:

$$E_b = \sigma * T^4 \tag{5}$$

Ve vzorci je T teplota v *Kelvinech*, σ je *Stefan-Boltzmanova konstanta* a E_b je *vyzářená energie* s jednotkou $\frac{W}{m^2}$. S rostoucí teplotou velmi rychle roste *vyzářená energie*. Pro aproximaci *šíření tepla* reálných těles se využívá zobecněný model *šedé těleso*, jehož *vyzářená energie* je určena vzorcem:

$$E_g = \epsilon * \sigma * T^4 \tag{6}$$

V reálném světě je situace ještě komplikovanější, protože ϵ závisí i na teplotě.

Pokud spolu interagují dvě černá tělesa, pak pro výměnu energie platí rovnost:

$$Q_{net1-2} = A_1 F_{1-2} \sigma (T_1^4 - T_2^4) \tag{7}$$

Při situaci, kdy elektromagnetické záření koliduje s tělesem může dojít ke 3 případům:

1. těleso přijme energii - procento záření, jež těleso přijme je určeno absorbtivitou α

2. záření se odrazí - procento záření, jež těleso odrazí je určeno reflektivitou ρ

3. záření projde tělesem - procento záření, jež tělesem projde je určeno transmisivitou τ

Pro tyto případy platí rovnice:

$$\alpha + \rho + \tau = 1 \tag{8}$$

Pokud vezmeme v potaz tyto případy, pak je černým tělesem takové těleso, jež má $\alpha = 1$, tedy přijme veškerou energii ze záření.

V případě, že budeme počítat i s odrazy mezi dvěma šedými tělesy, pak lze rovnost pro výměnu energie zobecnit na:

$$Q_{net1-2} = A_1 F_{1-2}(B_1 - B_2) \tag{9}$$

V rovnici B_1 a B_2 značí *radiozity* těchto těles (*tepelný tok*, jež je vyzářen) a ta je definována jako:

$$B = \rho H + \epsilon E_b \tag{10}$$

Zde H značí *irradianci* - *tepelný tok*, jež dopadá na plochu.

4 Energetická náročnost budovy

Spotřeba budovy je celková energie (obvykle se udává v MWh), jež tato budova využije na splnění svých cílů, během časového období. Do *spotřeby energie* se počítá i energie, jíž budova sama generuje, tedy například elektřinu za pomoci *solárních panelů*, nebo teplo za pomoci *tepelných čerpadel*.

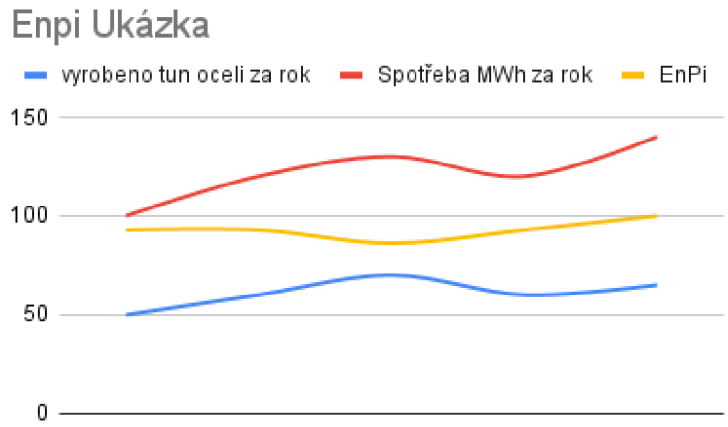
Od spotřeby se odvozuje vztažná veličina *EnPi*. V obecném případě se jedná o podíl $\frac{spotřeba}{veličina}$. Za *veličinu* je možné dosadit jakoukoliv jednotku, jež je pro danou budovu relevantní - tedy například pro hospodu by se mohlo za veličinu dosadit počet obsloužených zákazníků, nebo pro továrnu vyrábějící ocel by za veličinu bylo možné dosadit vyprodukované množství oceli.

Pro běžné budovy se v praxi používá *EnPi* nastavené na $\frac{spotřeba}{m^2}$ a tomuto *EnPi* se říká *energetická náročnost budovy*. Hlavním faktorem, jež ovlivňuje *energetickou náročnost* je spotřeba tepla a také se může značně lišit dle typu budovy:

1. *historické budovy* - tyto již obvykle nelze zateplit, a tedy mají velmi vysokou energetickou náročnost

2. *lázně, bazény, atp.* - tyto budovy spotřebovávají velké množství energie na vytápění, a tedy mají velkou energetickou náročnost

3. *školy, školky, atp.* - tyto budovy se využívají pouze v určitou část dne a bývají zateplené. Tedy mají nízkou energetickou náročnost



Obrázek 4: Příklad využití EnPi pro firmu exportující ocel (poznámka EnPi je posunuté a zvětšené pro ilustraci)

4. *bytové domy, domovy pro seniory, atp.* - tyto budovy jsou obývány po celý den a energetická náročnost se typicky liší dle úrovně zateplení

Obrázek 4 ilustruje jak může vypadat vývoj *EnPi* při dané spotřebě oceli a *spotřebě energie* v *MWh* za rok. *EnPi* zůstává poměrně neměnné oproti tomu ostatní veličiny spolu korelují. Díky tomu se *EnPi* může například využít ve strategii pro snížení *spotřeby*.

U běžných budov je také poměrně obvyklé, že se *EnPi* udává jako $\frac{denostupně}{spotřeba}$, v případě, že jsou k dispozici data o *spotřebách* alespoň na měsíční bázi. *Denostupeň* je definován jako:

$$D(t_{is}, T_{out}) = \begin{cases} t_{is} - T_{out} & \text{pokud } t_{is} - T_{out} \geq 0 \\ 0 & \text{jinak} \end{cases}$$

V definici je T_{out} venkovní teplota a t_{is} je vyžadovaná teplota uvnitř budovy.

Tedy kdyby teplota venku byla například $T_{out} = 5°\text{C}$ a teplota uvnitř by byla $t_{is} = 20°\text{C}$, pak by $D(20, 5)$, bylo rovno $20 - 5$, a tedy $D(20, 5) = 15$. Oproti tomu, když by teplota venku byla vyšší než teplota uvnitř, pak by byl výsledek roven 0. Tedy například $D(20, 30) = 0$.

Pro budovu, jež má dubnovou spotřebu $500kWh$, při průměrné venkovní teplotě 12°C, jež je vytápěna na 22°C a má velikost $100m^2$, pak je její *energetická náročnost* za duben rovna:

$$\frac{500}{100}\frac{kWh}{m^2} = 0.2\frac{kWh}{m^2} \tag{11}$$

a její EnPi vztažené k denostupni je rovno:

$$\frac{500}{22-12} = \frac{500}{10} = 50\frac{kWh}{K} \tag{12}$$

5 Aplikace

V této kapitole je celkově shrnuto fungování aplikace a její použití v praxi. Nejprve kapitola popisuje, které technologie byly použity pro vývoj aplikace. Dále se zabývá, jak se nahrávají soubory na server a transformují se do *voxelové reprezentace*. Nakonec představuje fungování *predikce energetické náročnosti* budov.

Aplikaci lze nasadit příkazem *docker-compose up*[1] z terminálu ve složce s aplikací. Automatizované testování lze pak spustit z *dockerového kontejneru* pomocí příkazu *npm test*.

5.1 Použité technologie

V této kapitole jsou stručně popsány technologie, jež jsou využity v aplikaci. V rámci aplikace je primární využitou technologií *Node.js* spolu s programovacím jazykem *JavaScript*. Na testování se využívá testovací framework *Mocha* spolu s *Should.js*. Pro tvorbu *Rest API* byl zvolen *Fastify*. Náročné výpočty se provádějí na grafické kartě za pomoci *Gpu.js*. Aplikace se nasazuje za pomoci *dockeru* v kombinaci s *docker-compose*. Data animací se ukládají do databáze *SQLite* za pomoci ORM *Sequelize*. Pro testovací účely zobrazení se využívá *voxelový engine Minetest*.

5.1.1 Node.js

Node.js je *open-sourcové* běhové prostředí pro *JavaScript*, jež je možné spustit na všech běžných platformách. *Node.js* běží na *JavaScriptovém* enginu *V8*, jež je využíván také prohlížečem *Google Chrome*. Výpočty v *Node.js* probíhají v rámci jednoho *procesu* a *standardní knihovny* jsou psány tak, aby nebyly *blokující*. *Node.js* podporuje v různých verzích různé verze standardu *ECMAScript* na němž je založen *JavaScript*. Validní soubor v *Node.js* lze spustit pomocí příkazu **node jmeno_souboru.mjs**. Pro spuštění s úpravou některých *parametrů* běhu lze přidat k příkazu *proměnné* tedy například: **node −− flag1 −−flag2 −−var1=X −−var2=Y jmeno_souboru.mjs**.

5.1.2 Mocha, Should.js

Mocha.js je testovací *framework* pro *prohlížeče* a *Node.js*. Základní syntax *mochy* je zobrazena v kódu 3.

Should.js je *deklarativní framework* pro tvorbu *tvrzení* o *testovacích případech* a je nezávislý na *testovacím frameworku*. *Should.js* funguje na principu úpravy *Object.prototype* v *JavaScriptu*. Jednoduchý případ použití je představen v kódu 4.

[1] Pro správné fungování aplikace musí být správně nainstalován a nastaven docker, dle oficiální dokumentace

```
describe("test1", () => {
   it("should ... testovaci_pripad1", () => {
      // kód testu
   })

   //dalsi testovaci pripady ...
})
```

Zdrojový kód 3: Ukázka základní syntaxe mochy

```
let pes = {
   barva: "hnědá"
};
pes.should.have.property("barva", "hnědá"); // pokud pes nebude m
    ít hnědou barvu, pak vrátí chybu
```

Zdrojový kód 4: Jednoduchá ukázka Should.js

5.1.3 Fastify

Fastify je server v *Node.js*, jež je zaměřený na vysokou rychlost *zpracování požadavků* ze strany uživatelů. Důležitou vlastností tohoto *serveru* je *rozšiřitelnost* a možnost *validace požadavků* pomocí *JSON Schema*. *Fastify* dále umožňuje využívání *logů* pomocí knihovny *Pino*. V neposlední řadě se zabývá *testováním API*. Kód 5 představuje, jak by mohl vypadat velmi jednoduchý *server* ve *Fastify*, jež při zadání *adresa-serveru:8080/ahoj* vypíše řetězec *svete*.

```
import Fastify from "fastify";
const fastify = Fastify();
fastify.get("/ahoj", async function (request, reply) {
   return "svete"
})
await fastify.listen({ port: 8080 })
```

Zdrojový kód 5: Hello world ve Fastify

5.1.4 Gpu.js

Gpu.js je *JavaScriptová knihovna*, určená pro *prohlížeče* i *Node.js*, pro *GPGPU*, tedy obecné programování na *grafické kartě*. Knihovna *transpiluje JavaScriptový kód* do *GLSL*, tedy *shaderového programovacího jazyka* běžně podporovaného *grafickými kartami*. Jednoduché využití na vytvoření pole s daty od 0 po 64 je ukázáno ve zdrojovém kódu 6.

```
1  import { GPU } from 'gpu.js';
2  const gpu = new GPU();
3
4  const kernel = gpu.createKernel(function() {
5     return this.thread.x              // index aktuálního kernelu
6  }).setOutput([64]);
7
8  console.log(kernel());               // vypíše pole od 0 po 64
```

Zdrojový kód 6: Ukázkový kód v gpu.js

5.1.5 Docker, Docker-compose

Docker je otevřená platforma pro vývoj, deployment a spouštění aplikací. Pomocí *dockeru* je možné oddělit aplikaci od softwaru nainstalovaného na serveru. Tedy je možné výrazně rychleji nasadit software na produkční prostředí, bez potřeby dalšího testování.

Docker-compose je nástroj pro spouštění více *Dockerových kontejnerů* pro usnadnění microservicové architektury. Dále výrazně zjednodušuje jejich nasazení. Kód 7 představuje, jak může vypadat velmi jednoduchý soubor pro tvorbu obrazu. Tuto aplikaci a databázi by pak bylo možné spustit pomocí *docker-compose*, jak je ukázáno v kódu 8.

```
1  FROM node:18.16.0          # běhové prostředí, v~němž aplikace
      poběží, včetně verze
2  COPY . /home/app/          # zkopírování aktuální aplikace do
      kontejneru
3  WORKDIR /home/app/         # nastavení aktuální složky
4  RUN npm install            # dodatečná instalace závislostí
5  ENTRYPOINT node run_server.mjs # vstupní aplikace
```

Zdrojový kód 7: Ukázka jednoduchého Dockerového souboru

5.1.6 Sequelize, SQLite

Sequelize je *ORM* (*Objektově relační mapování*) pro *Node.js* založené na *příslibech* a určené pro různé typy *databází* - jako například *SQLite* nebo *PostgreSQL*. Kód 9 představuje, jak vypadá vytvoření *SQLite databáze* v *Sequelize*. Jak v něm lze vytvářet tabulky a následně jak se na ně dotazovat.

5.1.7 Minetest

Minetest je *open-sourcový voxelový herní engine*. Hlavní předností je, že umí zobrazovat velké *voxelové mapy*. Dále umožňuje snadnou editovatelnost pomocí *módů* napsaných v *LuaAPI*. Pro účely *parsování JSONu* z *dotazu na API* je

```
services:
  app:
    build: .      # sestaví Dockerfile v~tomto adresáři
    ports:
      - 8080:8080 # namapování portů z~aplikace na port zařízení
    depends_on:
      - db        # počká na spuštění databáze
  db:
    image: postgres:latest   # sestaví obraz z~dockerového repozitáře
    environment:             # proměnné, jež je možné do dockeru vložit
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: database
    ports:
      - 5432:5432
```

Zdrojový kód 8: Ukázka nasazení databáze a vlastní aplikace v docker-compose

využívána dodatečná *knihovna* pro *luu - lua-cjson*. Obrázek 5 ukazuje, jak vypadá prázdný plochý svět v *minetestu*.

Obrázek 5: Prázdný plochý svět v minetestu

5.2 Architektura

V této kapitole je popsána celková *architektura* aplikace. Obecně se skládá z pěti částí, jež zajišťují různé úlohy. Obrázek 6 schématicky představuje, jak *architektura* vypadá.

V nejnižší *vrstvě architektury* jsou dvě části *DATA IMPORT* a *DB*. *DATA IMPORT* zajišťuje schopnost *importu* různých *formátů* do *voxelové reprezentace* spolu s *metadaty materiálů* využívanými v těchto formátech, nebo odhadem jejich

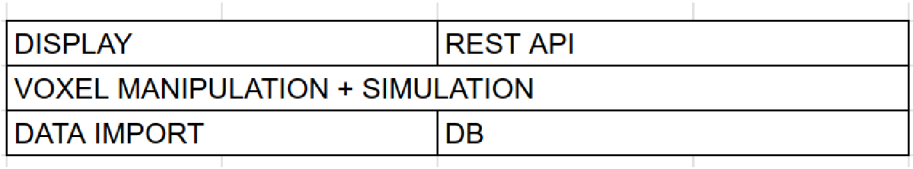
```
import { Sequelize, DataTypes } from "sequelize";

const sequelize = new Sequelize({
   dialect: "sqlite",
   storage: "./db.sqlite",
   logging: false
}); // vytvoří novou SQLite databázi v~aktuálním adresáři

const Dog = sequelize.define("dog",
   name: {
      type: DataTypes.STRING(20) //jméno je řetězec délky 20
   } // nový sloupec se jménem
) // vytvoření nové tabulky pro psy

await sequelize.sync() // synchronizuje tabulky v~kódu s~těmi v~
    databázi
await Dog.findAll() // vrátí všechny psy v~databázi
```

Zdrojový kód 9: Ukázkový kód Sequelize spolu s SQLite



Obrázek 6: Architektura aplikace

vlastností. *DB* umožňuje *perzistentní uložení dat*, za účelem zobrazení animace simulace a testování. Pro *DB* je použito již dříve zmíněné *Sequelize* a *SQLite*.

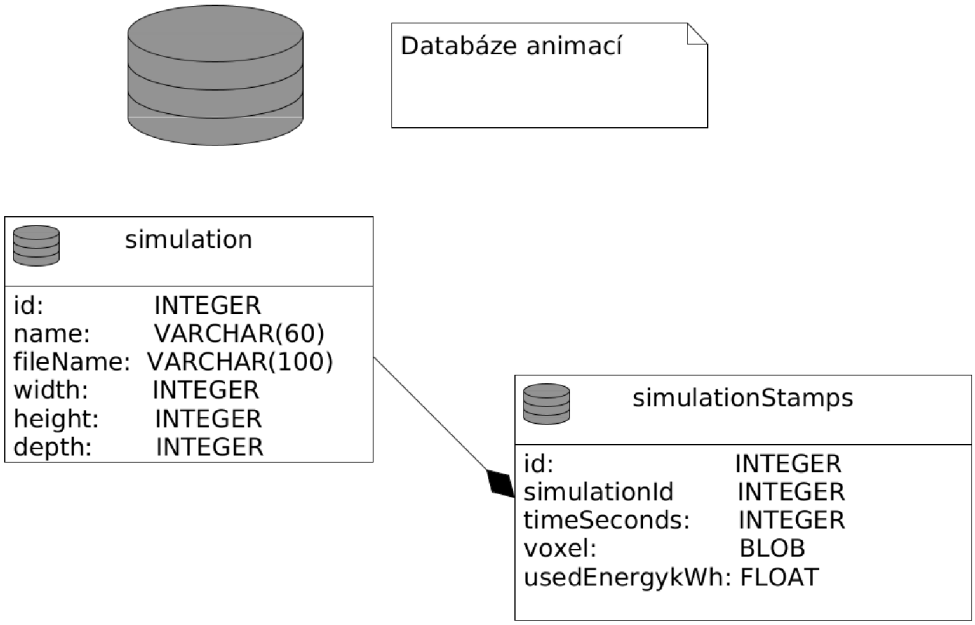
V druhé *vrstvě* je zajištěna *třída* pro obecné zpracování *voxelových dat* na *grafické kartě*, nebo *procesoru* pokud *grafická karta* není dostupná. Tato *třída* vytváří abstrakci nad *GPU.js*. Dále se zde zajišťuje celková *simulace šíření tepla*, přičemž tato diplomová práce se zabývá primárně o *šíření tepla vedením*.

Nejvyšší vrstva zajišťuje možnost *integrace* dalších aplikací pomocí *REST API* v *JSONu*. *REST API* je implementována pomocí dříve zmíněného *Fastify*. Další částí této vrstvy je *DISPLAY*, jež umožňuje zobrazení *nasimulovaných dat*, nebo nahraných souborů ve *voxelové reprezentaci*.

5.2.1 Databáze

Databáze je aktuálně určená primárně pro ukládání mezivýsledků simulace, jež slouží pro účely zobrazení simulovaných dat, jak je ukázáno například na obrázku 9. Databáze je připravená, aby bylo simulace možné znovu obnovit v případě náhlého pádu aplikace.

Schéma databáze je naznačeno na obrázku 7. Databáze se dělí na dvě tabulky *simulation* a *simulationStamps*. Tabulka *simulation*, modeluje jednotlivou nahra-



Obrázek 7: Schéma databáze pro uložení mezivýsledků simulace

nou simulaci, každá simulace musí být nahrána s unikátním jménem, uloženým v sloupci *name*. Pro každou simulaci je znám soubor, z něhož byla vytvořena, ve sloupci *fileName*. Dále jsou uloženy rozměry souboru, jež byl využit k simulaci ve sloupcích *width*, *height* a *depth*.

Tabulka *simulationStamps*, modeluje stav simulace v čase simulace *timeSeconds* sekund. Tabulka obsahuje odkaz na tabulku simulation v podobě cizího klíče *simulationId*. V tabulce je uložený *BLOB* dat, jež představuje *voxelovou reprezantaci* s teplotou v dané chvíli simulace. Posledním sloupcem je *usedEnergykWh*, který představuje kumulativní spotřebovanou energii za dobu simulace.

Kromě výše zmíněných *tabulek* a *sloupců databáze* obsahuje další, jež jsou vytvořeny automaticky pomocí *Sequelize*. *Sequelize* pomocí *triggerů* automaticky ukládá dobu, kdy byl prvek přidán, a kdy byl upraven. Také obsahuje *tabulku* pro implementaci *sekvencí*.

5.3 Nahrání souboru na server ve voxelové reprezentaci

Tato kapitola je zaměřená na technické řešení nahrání souboru a převod do *voxelové reprezentace*. Nejprve se kapitola zaměří na řešení načítání trojúhelníků z *CAD nákresu*, dále podrobně vysvětlí transformaci těchto trojúhelníků do *voxelové reprezentace*. Nakonec ukáže, jak lze tato řešení využít pomocí *REST API*. Protože toto načtení souboru může chvíli trvat, tak využívá *cachování* na zapamatování si výsledků pro případné znovuvyužití.

5.3.1 Načítání trojúhelníků

Načítání trojúhelníků je rozdělené na dvě části - nejprve se načtou trojúhelníky a *materiály* ze souborů, poté se odhadnou vlastnosti *materiálů*, pokud nebyly přiřazeny už ze souboru. Autor zvolil jako *CAD formát* pro testovací účely této práce přenositelný formát *Wavefront obj*, protože je často využíván pro jeho univerzální podporu napříč různorodými *CAD systémy*.

Pro testovací účely byla využita data o *materiálech* z *thermtestu*[2]. Zde se na základě názvů *materiálů* odhadnou vlastnosti materiálu, dle názvu *materiálu* z *CADu*, v případě, že *materiál* není nalezen, je zapotřebí ruční manipulace se souborem obsahujícím *materiály*.

Pseudokód 13 ukazuje, jak se obecně načítají trojúhelníky, kdy se nejprve rozhodne zda systém umí přečíst daný *formát* a poté se načtou trojúhelníky a *materiály*. Z trojúhelníků se poté odstraní *body*, a *setřídí* se sestupně podle velikosti, kvůli dalším výpočtům. Nakonec se *materiálům* přiřadí *tepelné vlastnosti* a vrátí se výsledek.

```
1   const supported_formats = {
2       obj: loadObj // načte obj dle jeho specifikace
3   }
4
5   function load_triangles(path) {
6       if(path.extension() in supported_formats && path.exists()) {
7           let [triangles, materials] = supported_formats[path.
                extension()](read_file(path));
8           triangles = triangles.filter(triangle => !is_point(triangle
                ))
9                           .sort(desc(triangle_size))
10          materials = assign_thermal_properties(materials);
11          return [triangles, materials];
12      }
13  }
```

Zdrojový kód 10: Pseudokód načítaní trojúhelníků

5.3.2 Transformace trojúhelníků do voxelové reprezentace

Autor nejprve vyzkoušel přístup dle pseudokódu 1. Pro *voxelovou reprezentaci* bylo využito *3D pole* a pro trojúhelníky *pole*, přičemž jednotlivé trojúhelníky byly reprezentovány jako *pole* obsahující 3 *body* a tyto *body* jsou pak reprezentovány taktéž jako *pole* o délce 3 obsahující číselné *souřadnice bodů*. Pro případy z reálného světa, kdy *modely budov* mohou obsahovat i miliardy trojúhelníků, se toto řešení neosvědčilo, protože při převodu na realistickou *voxelovou reprezentaci* o velikosti 256^3 s miliardou trojúhelníků může výpočet trvat značně dlouho dobu.

[2]https://thermtest.com/thermal-resources/materials-database

Při následujícím pokusu autor využil přístupu dle pseudokódu 2 se stejnými *implementacemi* jako v předchozím případě. Zde se objevil objektivní problém při *simulaci*, kdy se *vnitřní prostory zdí* chovaly jako *vzduch*. Za řešení autor zvolil kombinaci těchto přístupů, kdy jsou nejprve vykresleny trojúhelníky, poté se spočítají všechny *oblasti* na jejichž základě se vyberou vnitřní oblasti, jež jsou posléze *vyplněny materiálem*. V běžných budovách je poměrně malé množství těchto ucelených oblastí, a tedy je provedení kódu výrazně kratší. Tento přístup je ukázán v pseudokódu 11.

```
1   function conversion(triangles, voxel_size) {
2       let voxels = triangle_conversion(triangles, voxel_size);
3       let areas = find_areas_on_voxels(voxels);
4       for(let area of areas) {
5           let [inside, material] = is_inside(area[0], triangles);
6           if(inside) {
7               area.fill(material);
8           }
9       }
10      return voxels;
11  }
```

Zdrojový kód 11: Finální řešení konverze trojúhelníků do voxelové reprezentace

Implementace funkce is_inside je uskutečněna za pomocí jednoduchého *řešení soustavy lineárních rovnic* s využitím *Gausovy eliminace*. Vychází ze *vzorce pro trojúhelník*, kde A je *bod trojúhelníku* a $\vec{u_1}$, $\vec{u_2}$ jsou *vektory definující strany* a k, j jsou *skaláry*:

$$A\ + k\ \vec{a} + j\vec{b} \text{ t.ž.: } k\ + j <= 1 \wedge k\ > 0 \wedge j > 0 \tag{13}$$

a *rovnice přímky*, jež je definovaná se stejným značením, jako:

$$L + l\vec{v} \tag{14}$$

Na základě těchto *definic* lze vytvořit *rovnici* za účelem *získání průsečíku*:

$$A\ + k\ \vec{a} + j\vec{b} = L + l\vec{v} \tag{15}$$

Tuto *rovnici* lze převést do tvaru vhodnějšího pro *Gaussovu eliminaci*[3]:

$$k\ \vec{a} + j\vec{b} + l\vec{v} = L - A \tag{16}$$

Z této rovnice lze vytvořit soustavu tří rovnic o třech neznámých:

$$\begin{aligned} k\ \vec{a_1} + j\vec{b_1} + l\vec{v_1} &= L_1 - A_1 \\ k\ \vec{a_2} + j\vec{b_2} + l\vec{v_2} &= L_2 - A_2 \\ k\ \vec{a_3} + j\vec{b_3} + l\vec{v_3} &= L_3 - A_3 \end{aligned}$$

[3]u vektoru $\vec{v}$ nezáleží na směru, tedy je možné změnit znaménko

Tato *soustava rovnic* je automatizovaně řešitelná *Gaussovou eliminací*. Nakonec se pouze zjistí zda *rovnice* splňuje původní *podmínku* $k + j <= 1 \wedge k >= 0 \wedge j >= 0$ a pokud ano, pak existuje průnik, určený dosazením k a j do původní rovnice trojúhelníku. Na základě tohoto, pak *algoritmus* umí zjistit, které *voxely* jsou uvnitř, a které jsou venku, pomocí *lichosti/sudosti* počtu trojúhelníků, jež jsou před *voxelem* ve směru *přímky*. Toto postačuje k sestavení finálního pseudokódu 12 pro *funkci is_inside*.

```
1   function is_inside(coordinate, triangles) {
2       let line_vector = [0, 1, 0]; // vektor ve směru Y
3       return is_odd( triangles
4           .map(triangle => triangle_line_intersection(...triangle,
                coordinate, line_vector))
5           .filter(solution => solution && solution[1] > coordinate
                [1])
6           .length
7       ) // otestuje zda je počet průsečíků v~jednom směru lichý
            pokud ano, pak je uvnitř
8
9   }
10  function triangle_line_intersection(t_pt1, t_pt2, t_pt3,
        line_origin, line_vector) {
11      let [[X, Y, Z], [a1, a2, a3], [b1, b2, b3]] =
12          [t_pt1, t_pt2 - t_pt1, t_pt3 - t_pt1];
13      let [Lx, Ly, Lz] = line_origin;
14      let [vx, vy, vz] = line_vector;
15
16      let [k, j, l] = solve_linear_equations([
17          [a1, b1, vx, Lx - X],
18          [a2, b2, vy, Ly - Y],
19          [a3, b3, vz, Lz - Z]
20      ])
21      if(k + j > 1 || k~< 0 || j < 0) {
22          return undefined;
23      }
24
25      return [X, Y, Z] +
26            [k~* a1, k~* a2, k~* a3] +
27            [j * b1, j * b2, j * b3]
28  }
```

Zdrojový kód 12: Pseudokód implementace is_inside

Implementace *find_areas_on_voxels* je naznačena v pseudokódu 13. Zde se nejprve převede *voxelová reprezetace* na *grafovou reprezentaci*, dle *sousedních voxelů* na základě jejich *materiálu*. Tedy pokud má soused stejný *materiál*, pak bude existovat *hrana* od aktuálního *voxelu* k tomuto *sousedovi*, jinak ne. A každý *voxel* má přiřazený *jeden vrchol* ohodnocený jeho *indexem*. Výsledkem jsou *části grafu* mezi nimiž neexistuje žádná *hrana*. A tyto disjunktní části grafu tvoří

oblasti se stejným materiálem voxelů.

```
function find_areas_on_voxels(voxels) {
   let graph = new Graph(voxels)
               .on(voxel_neighbourghood)
               .from(voxel => voxel.material);
   return graph.disjunct_parts();
}
```

Zdrojový kód 13: Pseudokód implementace find_areas_on_voxels

5.3.3 API

Pro nahrávání souborů na server aplikace využívá *endpoint adresa:port/context/upload*. V praxi je možné jej využít například pomocí utility *cURL*, jak je ukázáno v kódu 14, kde se na *server* na adrese *localhost:8080* nahraje soubor umístěný v */home/uzivatel/Dokumenty/nazev_souboru.txt* a pojmenovaný jako *nazev_souboru.txt*.

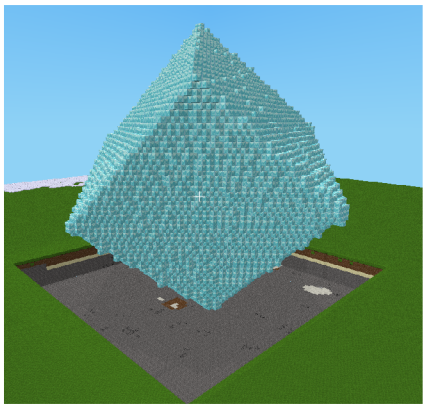
```
curl --request POST \
    --url http://localhost:8080/context/upload \
    --header 'Content-Type: multipart/form-data' \
    --form 'nazev_souboru.txt=@/home/uzivatel/Dokumenty/
       nazev_souboru.txt' \
```

Zdrojový kód 14: Uploadování souboru na server

Pro *transformaci* uloženého *CAD souboru* do *voxelové reprezentace* je použit *endpoint adresa:port/context/import*. Formát *JSONu*, jež je zapotřebí na něj zaslat je popsán v kódu 15. Po zaslání tohoto *JSONu* na *server metodou POST*, se do aktuálního kontextu načte soubor *nazev-souboru.obj*, jež byl nahrán na server a implicitně se načte i soubor *nazev-souboru.mtl* s *materiály* pokud je dostupný. Tato *voxelová reprezentace* má velikost $256 * 256 * 256$, tedy obsahuje 2^{24} voxelů. Reálné rozměry nahrané budovy jsou $20m * 20m * 20m$, tedy $8000m^3$.

```
{
  "file": "nazev-souboru.obj",
  "size": 256,
  "real_object_size": [20, 20, 20]
}
```

Zdrojový kód 15: Importování souboru do voxelové reprezentace a jeho zpřístupnění



Obrázek 8: Transformovaná 3D reprezentace rotované krychle do voxelové reprezentace

Pro získání výstupu v *JSONu* vhodného pro zobrazení, je možné dotázat se na server *GET metodou adresa:port/context/simple*. Tato *metoda* vrátí aktuální *kontext*. Tohoto *výstupu* pak využívá zobrazování v *minetestu*, kdy takto dotáže na server, a na jeho základě, pak vykreslí voxelovou reprezentaci. Možný příklad zobrazení rotované krychle v *minetestu* je ukázán na obrázku 8.

5.4 Predikce energetické náročnosti

Tato kapitola detailně představuje, jak funguje predikce energetické náročnosti, kdy se nejprve zabývá technickými detaily a následně ukazuje, jak vypadá její webové rozhraní a použití simulace.

5.4.1 Fungování

Predikce *energetické náročnosti budovy* funguje v několika krocích. Nejprve se zjistí, které *oblasti* jsou uvnitř budovy, a které jsou vně. Poté se ve smyčce podle počtu požadovaných sekund provádí *šíření tepla vedením*. Po uběhnutí tohoto počtu sekund proběhne ukládání mezivýsledků do *databáze* za účelem testování a zobrazení. Dále je v tomto kroku vytápěna na požadovanou teplotu.

Pro identifikaci vnitřku budovy se využívá algoritmus *find_areas_on_voxels* z pseudokódu 13, kdy algoritmus pro každou *oblast* zjistí, zda je její *materiál* vzduch a zda obsahuje nějaký *hraniční voxel*, a pokud ano, pak uvnitř budovy oblast není, jinak je. Pseudokód 16 shrnuje, jak by bylo možné implementovat tuto *funkci*.

Pro *simulaci šíření tepla* vedením autor nejprve vyzkoušel implementovat ji na klasickém procesoru ve více vláknech. To bylo pomalé pro běžné velikosti *voxelového kontextu* (256^3), a tedy se rozhodl implementovat algoritmus pro *grafickou kartu*. Obecně algoritmus funguje, jak je naznačen v pseudokódu 17, kdy se nejprve na *grafické kartě* provádí v určitém množství iterací určených *parametrem options.dev.gpu_iterations*, poté se uloží do *databáze* dle parametru *op-*

```
function find_inside_areas(voxels){
   return find_areas_on_voxels(voxels)
         .filter(area => area.material !== air ||
                        !area.some(voxel => voxel.includes(0)) ||
                        !area.some(voxel => voxel.includes(voxel.
                           dimensions - 1)));
}
```

Zdrojový kód 16: Ukázka pseudokódu možné implementace hledání vnitřní části budov

tions.animation, následně proběhne *vytápění budovy* a nakonec se okolí budovy ochladí z důvodu *proudění vzduchu*.

```
function simulate(building, seconds, options) {
   init_building(options.environment);
   let consumption = 0;
   do_times(seconds / options.dev.gpu_iterations, () => {
      do_times_on_gpu(options.dev.gpu_iterations, conduction);
      save_progress_to_db(building, options.animation);
      consumption += heat_building(building, options.heating);
      cool_outside(building, options.environment);
   })
   return consumption;
}
```

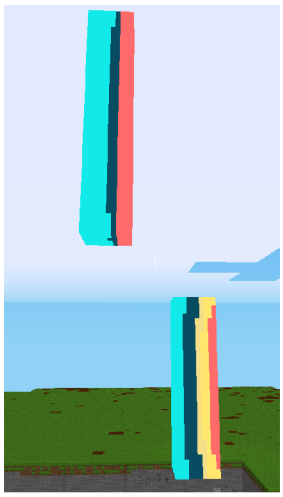
Zdrojový kód 17: Ukázka algoritmu simulace na grafické kartě

Simulace *vedení* je prováděna na *přímém okolí voxelu*, kdy se spočítá na základě *rozdílu teplot* a *velikosti voxelu tepelný tok*, a na základě *volumetrické tepelné kapacity* se zvýší nebo sníží *teplota aktuálního voxelu*. V případě, že *okolí voxelu* není definované, pak se využije teplota z okolí definovaná *parametrem options.environment.outside_temperature*.

Vytápění budov funguje primárně na principu *šíření tepla prouděním* a *vedením*. V rámci této simulace se předpokládá, že *vytápění* je *rovnoměrné*, a tedy vyhřeje všechny vnitřní prostory na požadovanou teplotu. Proto se na základě *volumetrické tepelné kapacity* a rozdílu teplot spočítá požadovaná *energie* na *vytápění vnitřních prostor budovy*. *Venkovní teplota* se ochlazuje z důvodu *šíření tepla prouděním* a funkčně je řešená stejně, jako *vytápění budov*.

Obrázek 9 ukazuje, jak může vypadat simulace *vedením*, v případě, že oba objekty jsou zprava zahřáty na vysokou teplotu. Oba objekty mají různou *tepelnou vodivost*, a tedy se jimi *šíří teplo* různou rychlostí. Teploty se postupně vyrovnávají v objektech směrem od teplé části k chladné.

Na základě získané *spotřeby z vytápění*, znalosti reálné *velikosti budovy*, *účinnosti vytápění* a ostatní využívané *energie* je pak možné jednoznačně určit *energetickou náročnost budovy*.



Obrázek 9: Šíření tepla vedením, u dvou materiálů s různou tepelnou vodivostí

5.4.2 API

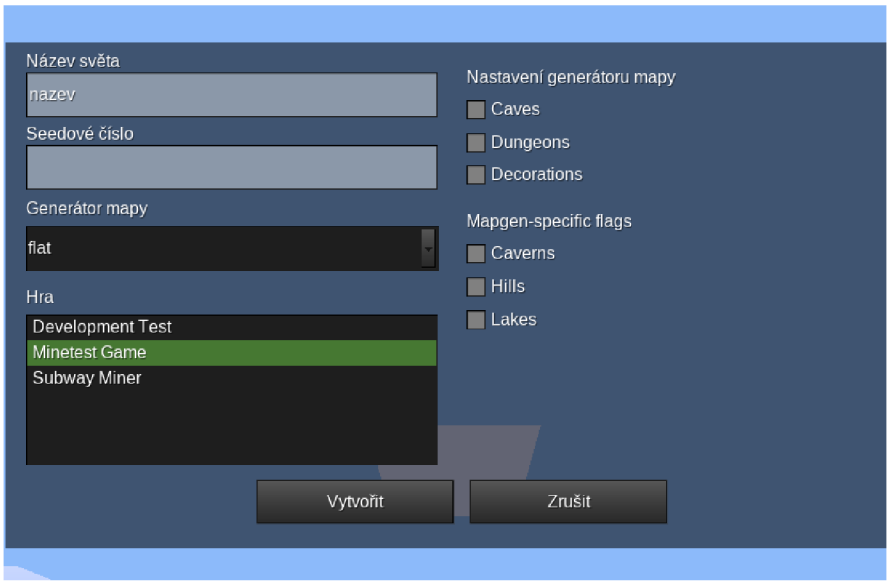
Pro simulaci aktuálního *kontextu* se využívá *endpoint* dostupný na *adresa:port/simulation/run*. Zde pak lze zaslat pomocí metody *POST* dotaz ve formátu *JSON*, jak je popsaný v kódu 18. Tento kód spustí simulaci pro jednu hodinu šíření tepla a vytápění. Volitelným parametrem *JSONu* je *animation*, jež zajistí, že se po každé iteraci, v níž se kód dostane zpět do procesoru, uloží aktuální teploty voxelů do *databáze*. Volitelný parametr *heating* umožňuje predikci spotřeby energie vytápění, v aktuálním případě se vytápí vnitřek budovy na 20°C při *efektivitě vytápění* 90% a spotřebě ostatní energie *1000 Joulů za hodinu*. Simulace začíná s počátečním stavem určeným parametrem environment, a tedy vnitřní teplota je 10°C a vnější teplota je 0°C. Nakonec je možné zadat nepovinný parametr *dev* a *gpu_iterations*, jež zařídí, že se každých 100 sekund simulace, dostane aktuální informace o teplotách z *gpu* na *procesor*.

Výsledek takového kódu je ve formátu naznačeném v pseudokódu 19, kdy je vrácena *spotřeba energie* v různých jednotkách. Dále je vrácena *energetická náročnost* v $\frac{MWh}{m^2}$ a *EnPi* vztažené k *denostupni*.

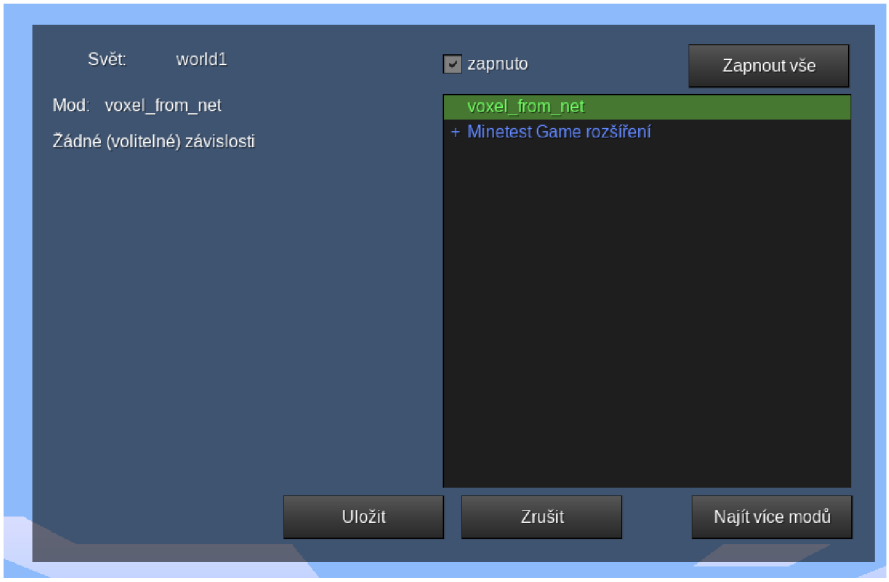
5.5 Zobrazení simulace a nahraného souboru

Pro zobrazení simulace je potřeba nejprve spustit *minetest*, kde je nainstalovaný mód *voxels_from_net*, dle dokumentace v *README.md*. Poté je potřeba vytvořit nový placatý svět, jak je ilustrováno na obrázku 10.

Poté je potřeba zvolit mód *voxels_from_net*, jak je ilustrováno na obrázku 11. Po spuštění nového světa se správně nastaveným *módem* a *běžícím serverem* s načteným *CAD modelem* by se měl po chvíli zobrazit tento voxelizovaný model, v případě bazénu by pak měla zobrazená *voxelová reprezentace* vypadat jako na obrázku 2. Pro zobrazení snímku uložené simulace stiskněte klávesu *F10*, poté zadejte příkaz */next_frame nazev-simulace*.



Obrázek 10: Vytvoření nového placatého světa v minetestu



Obrázek 11: Přidání módu voxels_from_net do již existujícího světa

```
{
   "hours": 1,
   "animation": {
      "name": "animace",
      "save_progress_after_iterations": 1
   },
   "heating": {
      "inside_temperature": 20,
      "efficiency": 0.9,
      "other_energy_consumption_per_hour_J": 1000
   },
   "environment": {
      "outside_temperature": 0,
      "inside_temperature": 10
   },
   "dev": {
      "gpu_iterations": 100
   }
}
```

Zdrojový kód 18: Rozhraní pro simulaci

6 Diskuze

V této kapitole jsou nejprve zhodnocena očekávaná chování simulace oproti jejím reálným výstupům. Dále se zhodnotí predikovaná energetická náročnost oproti očekávané. Nakonec jsou naznačena možná budoucí rozšíření projektu.

6.1 Očekávaná chování simulace

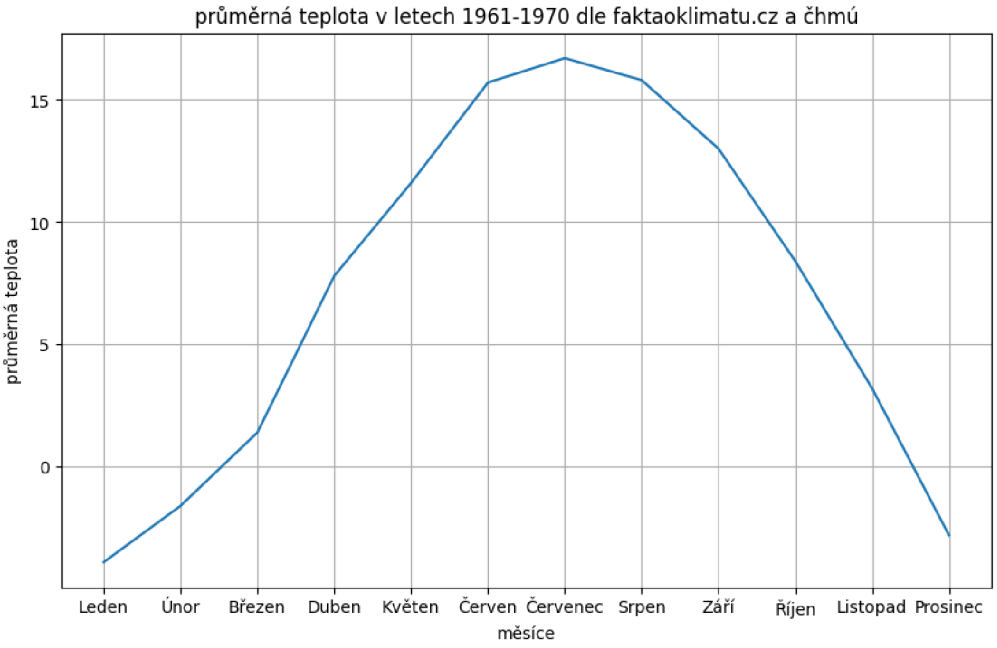
Od této simulace lze očekávat tři základní vlastnosti, jež by měla splňovat. První je, že čím vyšší je teplotní rozdíl tím větší by měla být spotřeba energie (a tedy i energetická náročnost budovy). Druhým předpokladem je, že čím je budova větší tím větší by měla být i její spotřeba. Poslední vlastností je, že zateplenější budova by měla mít menší spotřebu než nezateplená budova.

Pro testy těchto vlastností simulace autor využil teplotní průměry na území ČR během let 1961-1970, dle [1]. Data jsou představena na obrázku 12.

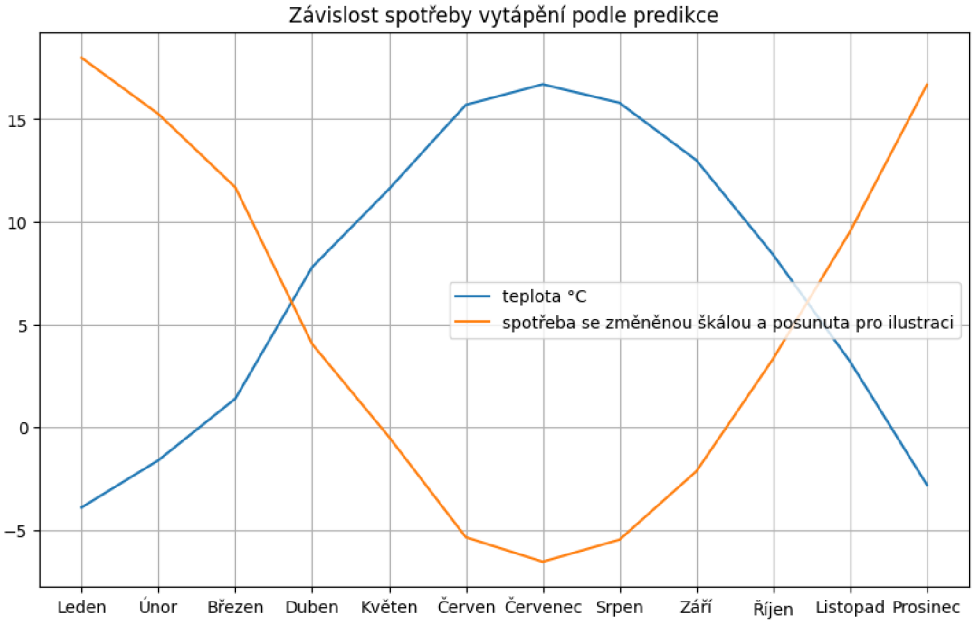
Závislost spotřeby na teplotě[4] je zobrazena na obrázku 13. Obrázek ilustruje, že simulace zřejmě splňuje první vlastnost, protože v měsíce, kdy je nižší teplota, pak je spotřeba vyšší a obráceně, když je teplota vyšší, pak je spotřeba nižší.

Pro otestování modelu na rozdílu spotřeb při různých reálných velikostech stejného modelu, byly použity dva identické modely bazénu se špatným zateplením. Jak lze vidět na obrázku 14, tak pokud je bazén menší, tak se výrazně snižuje spotřeba.

[4]spotřeba je upravená kvůli vhodnější vizualizaci, a tedy neodpovídá ose y



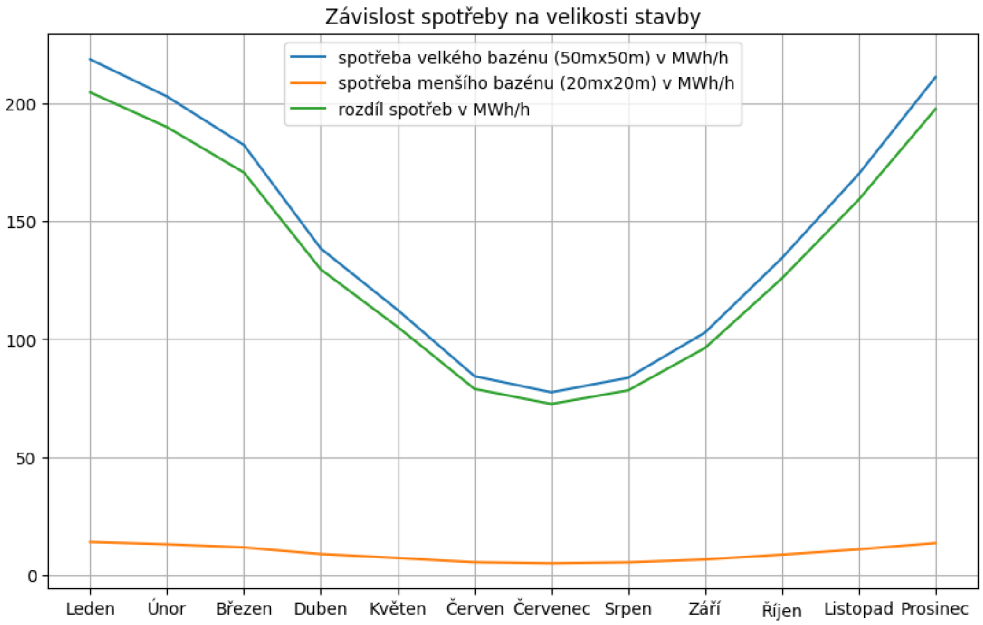
Obrázek 12: Průměrná teplota v letech 1961-1970 dle faktaoklimatu.cz



Obrázek 13: Závislost spotřeby na aktuálních teplotách

```
{
  "message": "success",
  "statistics": {
    "energy_consumption_MWh": 1.1798111508791602,
    "energy_consumption_kWh": 1179.8111508791603,
    "energy_consumption_MJ": 0.4247320143164977,
    "energy_efficiency_MWh_m2": 0.0029495278771979007,
    "energy_per_degree_day_MWh": 0.05899055754395801
  }
}
```

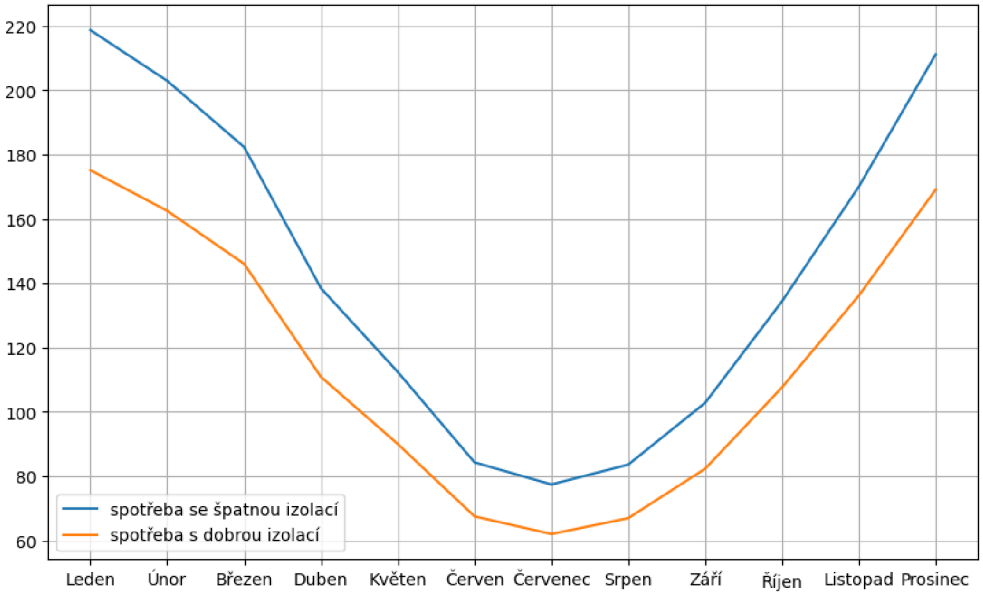
Zdrojový kód 19: Možný výsledek určité simulace



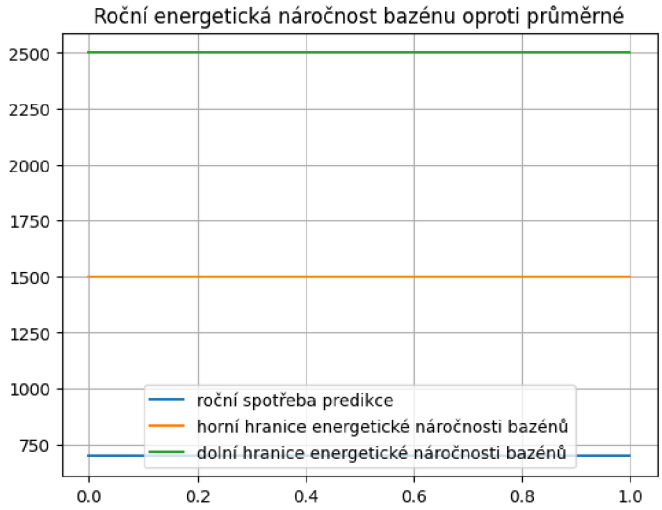
Obrázek 14: Závislost spotřeby na velikosti stavby

Pro otestování modelu na různých materiálech zateplení byly zvoleny dvě izolace: ocel a NIST 1450. V případě oceli jsou výsledky simulace o více než 40MWh za měsíc vyšší než bez vhodné izolace. Výsledky testu jsou zobrazeny na obrázku 15.

Nakonec je simulace porovnána se skutečnými energetickými náročnostmi běžných veřejných bazénů a aquaparků dle [2]. Toto srovnání ukazuje obrázek 16, tedy predikce je výrazně podhodnocena - cca. 2x. Autor toto přisuzuje tomu, že model nepočítá s vyššími ztrátami ze země, kdy předpokládá, že vše okolo stavby je vzduch. Dále je možné, že do spotřeby energie mohou v případě bazénů vstupovat i další významné přístroje, jež spotřebovávají značné množství energie. V neposlední řadě je možné, že účinnost tvorby tepelné energie může být nižší.



Obrázek 15: Odlišné izolace ve stejné budově



Obrázek 16: Porovnání predikované energetické náročnosti bazénů oproti běžné

6.2 Zhodnocení a možná rozšíření

V rámci diplomové práce byla vytvořena aplikace, jež umí načíst CAD soubory ve voxelové reprezentaci. Dále byla vytvořena simulace, jež ve voxelové reprezentaci umí na grafické kartě simulovat šíření tepla vedením. Simulace je díky využití voxelové reprezentace obecná a rozšiřitelná.

Aktuálními omezením pro využití v praxi je absence možnosti nahrát více formátů CADů, obzvláště ArchiCADu, jež je často využíván pro tvorbu architektury. Za další omezení autor považuje velmi pomalé načítání CADu do voxelové reprezentace, kdy nahrání testovacího CAD bazénu může trvat i hodinu. Za poslední nedostatek autor považuje absenci otestování aplikace na větším množství dat.

Do budoucna by bylo možné zrychlit voxelizaci CADu, jak je navrženo například v [3]. Pro použití v praxi by dále bylo vhodné zakoupit licenci formátu ArchiCADu a umožnit jeho nahrání do aplikace. V neposlední řadě by bylo vhodné vše otestovat na reálných datech. Nakonec by bylo možné využít nového standardu *WebGPU*, jež byl představen v době tvorby této práce a je obecnější než aktuální řešení pomocí *Gpu.js*. Možnými rozšířeními simulace by mohla být implementace *šíření tepla zářením*, jež způsobuje vyšší teploty uvnitř budov primárně v letních měsících a implementace *šíření tepla prouděním*, jež způsobuje snížení teploty ve venkovní části a je aktuálně implementována naivně, dále by byla využitelná pro simulaci větrání v budovách.

Závěr

Diplomová práce nejprve popsala problematiku načtení *CADu* do *voxelové reprezentace*. Poté se zaměřila na popis *fyzikálního modelu šíření tepla*, kdy se zabývala zejména *šířením tepla vedením* a *zářením*. Dále se zaměřila na popis *Energetické náročnosti budovy* v kontextu *energetiky*. Nakonec popsala aplikaci z technologického hlediska i z hlediska jejího použití ze strany vývojáře. Dále byly diskutovány nedostatky aplikace, její predikované výstupy a její možná rozšíření.

Aplikaci, jež je výstupem této práce, je možné snadno využívat díky *REST API* implementované ve *Fastify*. Aplikace je rozdělená do dvou hlavních částí, z nichž první umožňuje nahrání souboru s budovou ve voxelové reprezentaci z CAD formátu. Další část zajišťuje simulaci šíření tepla vedením v nahrané voxelové reprezentaci a vytápění této části. Nakonec se využijí nasimulované výsledky k predikci energetické náročnosti budovy.

Conclusions

Master's thesis has described the problem of loading *CAD files* to *voxel representation*. Then it described the physical model of *heat transfer*, more specifically - *heat conduction* and *radiation*. After that it characterized *energy efficiency* of buildings from the *energetics* point of view. In the end it shown what the application can do and how it is used. Then there were discussed correctness and possible future extensions to the app.

Application that is result of the thesis is possible to easily use thank to *REST API* implemented in *Fastify*. Application is divided into two main parts, the first allows people to upload a building in *CAD format* and convert it to *voxel representation* and the second allows simulation of heat transfer in uploaded in *voxel representation*. In the end the simulated values are used for energy efficiency prediction.

A Obsah elektronických dat

`text/`
: Složka s textem práce

`README.md`
: Dokumentace ke zprovoznění softwaru

`source/3D models/`
: Složka obsahující 3D modely využité v práci pro testovací účely

`source/DATA_LAYER/`
: Složka obsahující soubory s výpočty pro import do voxelové reprezentace

`source/DATABASE/`
: Složka obsahující schéma databáze v *Sequelize* a databázi samotnou v *SQLite*

`source/minetest_integration/`
: Složka obsahující mód pro minetest

`source/OPEN_DATA_SCRAPING/`
: Složka obsahující skript získání otevřených dat pro testovací účely a jejich výstup, pro případ změny stránky

`source/showcase`
: Složka obsahující obrázkové ukázky softwaru

`source/tests`
: Složka obsahující automatizované testy

`source/VOXEL_MANIPULATION`
: Složka obsahující simulaci a server ve fastify

Literatura

[1] *Fakta o klimatu — faktaoklimatu.cz*. [Accessed 29-04-2024]. Dostupný také z: ⟨faktaoklimatu.cz⟩.

[2] *Energy saving for public swimming pools — sunnyshark.com*. [Accessed 07-05-2024]. Dostupný také z: ⟨https://sunnyshark.com/en/energy-saving-public-pool/⟩.

[3] *arxiv.org*. [Accessed 17-04-2024].

[4] *ArchicCAD website*. [Accessed 03-04-2024]. Dostupný také z: ⟨https://graphisoft.com/solutions/archicad⟩.

[5] *NX software | Siemens Software — plm.sw.siemens.com*. [Accessed 03-04-2024]. Dostupný také z: ⟨https://plm.sw.siemens.com/en-US/nx/⟩.

[6] contributors, Wikipedia. *Wavefront .obj file*. 2023. Dostupný také z: ⟨https://en.wikipedia.org/wiki/Wavefront_.obj_file⟩.

[7] Aleksandrov, M.; Zlatanova, S.; Heslop, D. J. Voxelisation and voxel management options in UNITY3D. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2022, roč. X-4/W2-2022, s. 13–20. Dostupný také z: ⟨http://dx.doi.org/10.5194/isprs-annals-x-4-w2-2022-13-2022⟩.

[8] Lienhard IV, John H.; Lienhard IV, John H. *A Heat Transfer Textbook, 5/e*. Dostupný také z: ⟨https://ahtt.mit.edu/wp-content/uploads/2020/08/AHTTv510.pdf⟩.

[9] contributors, Wikipedia. *Heat*. 2024. Dostupný také z: ⟨https://en.wikipedia.org/wiki/Heat⟩.

[10] contributors, Wikipedia. *Temperature*. 2024. Dostupný také z: ⟨https://en.wikipedia.org/wiki/Temperature⟩.

[11] contributors, Wikipedia. *Thermal conductivity and resistivity*. 2024. Dostupný také z: ⟨https://en.wikipedia.org/wiki/Thermal_conductivity_and_resistivity⟩.

[12] *Node.js website*. Dostupný také z: ⟨https://nodejs.org/en/learn/getting-started/introduction-to-nodejs⟩.

[13] *Mocha - the fun, simple, flexible JavaScript test framework — mochajs.org*. [Accessed 13-04-2024]. Dostupný také z: ⟨https://mochajs.org/⟩.

[14] *should — npmjs.com*. [Accessed 13-04-2024]. Dostupný také z: ⟨https://www.npmjs.com/package/should⟩.

[15] *Fast and low overhead web framework, for Node.js | Fastify — fastify.dev*. [Accessed 13-04-2024]. Dostupný také z: ⟨https://fastify.dev/⟩.

[16] *Get Docker — docs.docker.com*. [Accessed 13-04-2024]. Dostupný také z: ⟨https://docs.docker.com/get-docker/⟩.

[17] *Docker Compose overview — docs.docker.com*. [Accessed 13-04-2024]. Dostupný také z: ⟨https://docs.docker.com/compose/⟩.

[18] *Getting Started | Sequelize — sequelize.org*. [Accessed 13-04-2024]. Dostupný také z: ⟨https://sequelize.org/docs/v6/getting-started/⟩.