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

CLOUDOVÉ ŘEŠENÍ PRO ZPRACOVÁNÍ 3D MODELŮ

CLOUD SOLUTION FOR 3D MODELS PROCESSING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB KLEMENS**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL ŠPANĚL, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Klemens Jakub**
Program: Informační technologie
Název: **Cloudové řešení pro zpracování 3D modelů**
Cloud Solution for 3D Models Processing
Kategorie: Počítačová grafika

Zadání:

1. Prostudujte problematiku reprezentace a zpracování 3D modelů (typické operace, apod.). Seznamte se s existujícími knihovnami pro zpracování 3D modelů.
2. Prostudujte dostupné cloudové technologie a jejich možnosti.
3. Vyberte vhodné technologie a navrhněte cloudové řešení, které bude poskytovat operace nad 3D modely.
4. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace metod.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručný plakát nebo video prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Španěl Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 2. dubna 2019

Abstrakt

Tato práce se zabývá možnostmi zpracování 3D modelů v prostředí cloudových aplikací. Na základě získaných znalostí vznikla knihovna Cloud3D implementována v jazyce C++. Výsledná knihovna slouží k rychlému vytváření klient-server aplikací. Knihovna je rozdělena do tří samostatných částí: klient, poskytovatel služeb a Load Balancer. Poskytovatel služeb pracuje v cloudu a umožňuje zpracovávání 3D modelů, které mu zašle klient. Největší výhodou knihovny Cloud3D je jednoduchost vytváření nových aplikací s její pomocí. Mezi další výhody patří škálovatelnost, zajištěna implementací look-a-side Load Balanceru, a bezpečnost, zajištěna použitím SSL certifikace.

Abstract

This thesis deals with the possibilities of processing 3D models in cloud applications. A C++ library called Cloud3D has been designed and implemented. The resulting library is used to quickly create client-server applications. The library is divided into three separate parts: Client, Service Provider and Load Balancer. The service provider runs in the cloud and provides 3D model processing services to client applications. The biggest advantage of Cloud3D is the ease of creating new applications with its help. Other benefits include scalability, assured implementation of look-a-side Load Balancer, and security ensured by the use of SSL certification.

Klíčová slova

3D, modely, grafika, cloud, knihovna, klient, poskytovatel služeb, load balancer, protoc, gRPC

Keywords

3D, models, graphic, cloud, library, client, service provider, load balancer, protoc, gRPC

Citace

KLEMENS, Jakub. *Cloudové řešení pro zpracování 3D modelů.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Španěl, Ph.D.

Cloudové řešení pro zpracování 3D modelů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Klemens
22. května 2019

Poděkování

Děkuji panu Ing. Michalu Španělovi Ph.D. za jeho odborné vedení práce, dále bych pak chtěl poděkovat svým rodičům za obrovskou podporu a v neposlední řadě své přítelkyni za neskonalou trpělivost.

Obsah

Kapitola 1

Úvod

V dnešní době se setkáváme s počítačovou grafikou na každém kroku. A zatímco cena a výkon nových počítačových grafik stoupá každým rokem, nutnost zpracovávat velké množství složitých trojrozměrných dat roste každým dnem. Z tohoto důvodu se v praxi stále častěji přechází k ponechání složitých výpočtů do cloudu.

V této práci bude zpracována problematika reprezentace a zobrazování trojrozměrných objektů v počítačové grafice. Dále se bude věnováno nejčastějším operacím, které mohou být s prostorovými objekty prováděny a několika algoritmům pro jejich zjednodušení. Na základě získaných znalostí bude následně navržena a implementována knihovna pro tvorbu klient-server aplikací zpracovávající trojrozměrné modely. Knihovna se od ostatních volně dostupných knihoven odlišuje svojí kombinací práce s 3D modely a možností provádět složité výpočty vzdáleně na serveru.

Text práce je členěn do kapitol. V 2. kapitole je popsána reprezentace trojrozměrných modelů v počítačové grafice. V kapitole 3 jsou popsány běžné geometrické operace, jenž jsou s objekty prováděny, a některé vybrané algoritmy pro zjednodušení sítě trojúhelníků. Kapitola 4 se zaobírá problematikou vyvažování zátěže. Jsou zde popsány základní přístupy, jejich výhody a nevýhody. V kapitole 5 jsou popsány vybrané knihovny umožňující tvorbu cloudových řešení a operace s 3D modely. Kapitola 6 popisuje příklady řešení, jenž by bylo možné vytvářet za použití řešení z této práce. Kapitola 7 se věnuje návrhu samotné knihovny vytvořené v rámci této práce, na což je navázáno v kapitole 8, která se věnuje popisu implementace. V kapitole 9 jsou popsány experimenty, které proběhly s knihovnou vytvořenou v rámci této práce a mají za úkol ověřit úspěch implementace. V závěrečné kapitole 10 budou shrnuty dosažené výsledky a nastíněny budoucí plány s touto prací.

Kapitola 2

Reprezentace prostorových objektů

V této kapitole bude popsáno, jak jsou trojrozměrné modely reprezentovány v počítačové grafice. Bude zde popsáno vytváření celých těles. Těleso v tomto případě značí spojitý útvar, tvořený jedním celkem. Mezi nejznámější a nejpoužívanější reprezentaci těles patří sítě trojúhelníků. Mimo to se ale budeme věnovat i hraniční reprezentaci těles a konstruktivní geometrii těles. Většina informací v této kapitole byla čerpána z knihy [12].

Aby bylo možné vyobrazovat 3D modely, muselo se začít u mnohem jednoduších reprezentací. Základy novodobého zobrazování křivek položil Pier de Casteljau v roce 1959, který vymyslel matematický model pro jednoduchou práci s nimi, zatímco pracoval pro firmu Citroen. Tento model byl později formalizován a zpopularizován Pierrem Bézierem, podle nějž také nesou název *Bézierovy křivky*.

Dalším výrazným pokrokem v této oblasti byl začátek používání racionálních B-spline křivek a ploch s neuniformní parametrizací zvaných *NURBS* (Non-Uniform Rational B-Spline).

Díky těmto základům můžeme dnes zobrazovat složité útvary a scény na vysoké úrovni[19, str. 108-111].

2.1 Sítě trojúhelníků

Jak už název napovídá, *sítě trojúhelníků* jsou složeny z trojúhelníků. Ty se používají pro svoje dobré vlastnosti jako zaručení konvexnosti, grafická podpora procesorem nebo dobrá optimalizace pro mnoho algoritmů. Aby trojúhelníky mohly tvořit síť, musí každý trojúhelník sdílet minimálně dvě své hrany.

Sítě rozdělujeme na dvě logické části:

1. Geometrickou
2. Topologickou

V geometrické části jsou uloženy všechny souřadnice vrcholů trojúhelníků, které se v tělese vyskytují a topologická část obsahuje údaje o tom, kterými vrcholy jsou tvořeny jednotlivé trojúhelníky.

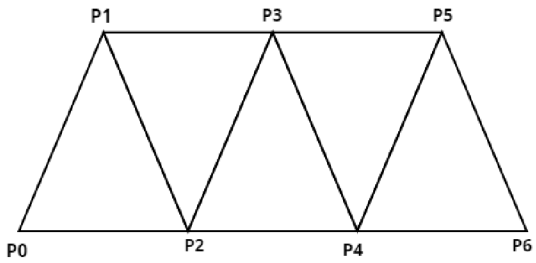
Při práci se sítí trojúhelníků je kladen důraz zejména na dvě vlastnosti:

1. Přesné a úsporné vyjádření tvaru

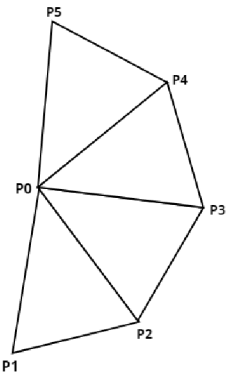
2. Vhodně navržené uspořádaní pro další použití

Přesné a úsporné vyjádření se týká především geometrické části. Z důvodu, že trojúhelníková síť není vhodná k modelování, se obvykle modelování provádí pomocí jiné reprezentace. Proto je nutné, aby se převod z této reprezentace snažil o co nejmenší počet trojúhelníků při zachovaní co nejvěrnějšího tvaru.

Druhá vlastnost souvisí s její topologií, protože rozdělení na geometrickou a topologickou část nemusí být vždy vhodné. Někdy je potřeba popsat objekt pouze pomocí jedné lineární datové struktury. Abychom zamezili redundantním operacím nad vrcholy trojúhelníků, tak trojúhelníky uložíme do posloupnosti tzv. pruhu trojúhelníků. Díky této posloupnosti docílíme, že každý vrchol bude zpracován pouze jednou. Stejného výsledku lze dosáhnout i použitím tzv. vějíře trojúhelníků. Příklad uspořádaní do pruhů lze pozorovat na obrázku 2.1 a příklad uspořádaní do vějíře lze vidět na obrázku 2.2.



Obrázek 2.1: Uspořádání sítě trojúhelníků do pruhu.



Obrázek 2.2: Uspořádání sítě trojúhelníků do vějíře.

Mimo rozdělení na geometrickou a topologickou část, můžeme síť trojúhelníků vyjádřit ještě explicitně. V takovém případě je každý trojúhelník vyjádřen svými vrcholy. Toto vyjádření je nevhodné, protože je neúsporné, neboť společné vrcholy jsou zde uloženy dvakrát až třikrát. Další problém spočívá v neurčitém vyjádření, a tím může dojít ke ztrátě při zaokrouhlování. V neposlední řadě je při každé změně nutné procházet všechny trojúhelníky a hledat všechny společné vrcholy[4, str. 473-476].

2.2 Hraniční reprezentace těles

Jedná se o nejpřirozenější způsob popisu objektu. Objekt je popsán pomocí množiny hraničních bodů (*boundary representation*). Vnitřní body jsou buďto ignorovány nebo lze informace o nich zjistit pomocí popisu hranice. Hraniční reprezentace je vhodná pro geometrické výpočty. Také dovoluje popsat širokou paletu těles, a to dokonce takové, které nejdou vyrobit. Proto vznikly pojmy manifold a nonmanifold. Manifold symbolizuje těleso, které lze skutečně vyrobit. Opakem je nonmanifold, jenž popisuje objekty, které nelze skutečně vy-

robit. Příkladem mohou být objekty, které se dotýkají pouze v jednom bodě, což v reálném světě nelze vyrobit. Za manifold považujeme těleso, jehož hrana inciduje se dvěma hranami a jehož strany neprotínají jiné plochy. Dále musí platit, že osamocený vrchol nesmí spojovat dvě části tělesa.

Eulerova rovnost

Eulerova rovnost udává, jestli je mnohostěn hranicí uzavřeného objemu. Vyjadřuje vztah

$$F + V = E + 2, \tag{2.1}$$

kde F je počet stěn, V je počet vrcholů a E je počet hran. Splnění této rovnosti ještě samo o sobě nezaručuje, že tato množina tvoří mnohostěn. Ještě musí platit, že každá hrana propojuje dva vrcholy a stěny a hrany se nesmějí protínat.

Pokud má těleso otvor, je nutné použít zobecněnou Eulerovou rovnost. Ta je určena vztahem

$$F + V = E + 2(C - H) + R, \tag{2.2}$$

kde R symbolizuje počet vnitřních smyček hran, C je počet samostatných komponent tělesa a H je počet otvorů, jenž prochází tělesem.

Hranová reprezentace

Modely v hranové reprezentaci jsou popsány pouze hranami a vrcholy. Tyto modely občas bývají označovány jako drátové modely (*wire-frame model*).

Drátový model, podobně jako síť trojúhelníků, je tvořen seznamem vrcholů a seznamem hran. Každá položka v seznamu hran potom odkazuje právě na dva vrcholy. Ač je tento zápis paměťově úsporný, neposkytuje dostatek informací o generovaném tělese.

Plošková reprezentace

Modely v ploškové reprezentaci jsou popsány pouze svými plochami. Jedná se tak o rozšíření původní hranové reprezentace o plochy. Ty mohou být tvořeny obecnými polygony, ovšem výpočetně nejvýhodnější varianta je použití trojúhelníků.

Ploškovou reprezentaci můžeme rozdělit na dvě skupiny:

1. Jednoduchá
2. Strukturovaná

V jednoduché reprezentaci se obvykle používá jednoduchá datová struktura, ve které je uchován počet vrcholů a odkazy do seznamu vrcholů. V této reprezentaci má také smysl uchovávat pořadí vrcholů na obvodu. Pokud budeme vrcholy uchovávat proti směru hodinových ručiček, lze z nich odvodit směr normálového vektoru. Jednoduchá plošková reprezentace umožňuje oproti hranové reprezentaci uchovávat více informací a umožňuje vykreslování zohledňující viditelnost.

Strukturovaná plošková reprezentace neboli *okřídlená hrana* (*winged-edge*) obsahuje mimo jiné ukazatele na sousední geometrické elementy. Objekt v této reprezentaci sestává ze tří seznamů v hierarchickém uspořádání. Na nejvyšší úrovni je seznam ploch, uprostřed se nachází seznam okřídlených hran a na nejspodnější úrovni je seznam vrcholů. Okřídlené hrany, kromě informace o hraně samotné, obsahují ukazatele na sousední plochy, informace

o dalších hranách a ukazatel na další hranu. Seznamy vrcholů a ploch se potom pouze odkazují do seznamu okřídlených hran. Největší předností této reprezentace je, že umožňuje jednoduše nalézt sousedící plochy, plochy incidující s danou hranou nebo vrcholy a hrany dané stěny.

Kapitola 3

Operace s 3D modely

V této kapitole budou popsány základní operace, které lze s trojrozměrnými modely provádět. Nejprve se zaměříme na základní geometrické transformace a následně si popíšeme některé složitější algoritmy pro různé operace.

Většina algoritmů pro práci s trojrozměrnými modely vychází z rovnic pro práci s dvourozměrnými obrazy, které zde nebudou popsány, neboť nejsou primárním cílem této práce.

3.1 Geometrické transformace

Geometrické transformace s trojrozměrnými objekty jsou pouze zobecněním dvourozměrných transformací a pro tyto operace se používá čtvercová matice čtvrtého stupně[10, str. 151-168].

Tyto výpočty nejsou dostatečně složité, aby se oplatilo je řešit na vzdáleném zařízení, ale jsou naprostým základem práce s 3D modely.

3.1.1 Posunutí

Posunutí trojrozměrného objektu je určeno vektorem posunutí $\vec{p}(X_t, Y_t, Z_t)$ a transformační maticí T, která je definována jako

$$T = \begin{bmatrix} 1 & 0 & 0 & X_t \\ 0 & 1 & 0 & Y_t \\ 0 & 0 & 1 & Z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.1}$$

3.1.2 Otočení

Otáčení v trojrozměrném prostoru lze rozdělit na dva podproblémy:

1. Otočení kolem souřadnicových os
2. Otočení kolem obecné osy

Operace otočení kolem osy x je definována úhlem α a $maticR_x$, jenž má tvar

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.2}$$

Matice pro osu y má podobný tvar

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.3}$$

respektive pro osu z

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.4}$$

Otáčení kolem obecné osy se realizuje o poznání hůř než kolem souřadnicových os. Tato problematika se řeší skládáním jednotlivých transformací, ovšem nalezení správných rotačních úhlů není triviální záležitost. Z tohoto důvodu se používají *Rodriguesovy formule*, jenž tento problém převádí na promítání a skládání vektorů. Díky tomu lze rotaci kolem obecné osy definovat polohovým vektorem $\vec{x}'$ jako

$$\vec{x}' = \cos\alpha.\vec{x} + (1 - \cos\alpha)(\vec{a}.\vec{x})\vec{a} + \sin\alpha(\vec{a} \times \vec{x}), \tag{3.5}$$

kde $\vec{a}$ je jednotkový vektor, $\vec{x}$ označuje polohový vektor transformovaného bodu X a α určuje úhel otočení kolem osy otáčení. Důležitou podmínkou je, že osa otáčení je určena počátkem souřadnicového systému. Tento vztah lze přepsat do maticového zápisu

$$X' = R(\vec{a}, \alpha)X, \tag{3.6}$$

kde

$$R(\vec{a}, \alpha) = \cos\alpha.I + (1 - \cos\alpha) \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z & 0 \\ a_x a_y & a_y^2 & a_y a_z & 0 \\ a_x a_z & a_y a_z & a_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \sin\alpha \begin{bmatrix} 0 & -a_z & a_y & 0 \\ a_z & 0 & -a_x & 0 \\ -a_y & a_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.7}$$

a kde I označuje jednotkovou matici.

Pokud bychom chtěli provést obecnou rotaci mimo počátek souřadnicového systému, bude nejprve potřeba objekt posunout do počátku, provést rotaci a nakonec posunout zpět na původní pozici[13, str. 78-80]. V takovém případě bude transformační matice A složená ze tří transformací a bude mít tvar

$$A = T(P_x, P_y, P_z)R(\vec{a}, \alpha)T(-P_x, -P_y, -P_z). \tag{3.8}$$

3.1.3 Škálování

Změna měřítka je definována koeficienty s_x, s_y a s_z, které určují změnu ve směru příslušné souřadnicové osy a pro které platí $s_x \neq 0$, $s_y \neq 0$ a $s_z \neq 0$. Transformační matice má tvar

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.9}$$

Pomocí této operace lze při použití správných koeficientů provést transformaci středové souměrnosti, souměrnosti roviny a osové souměrnosti.

3.1.4 Zkosení

Operace zkosení je definována třemi koeficienty sh_x, sh_y a sh_z, které určují míru zkosení v jednotlivých směrech. Také určujeme tři matice zkosení, a to pro každou rovinu zvlášť, jenž mají tvar

$$Sh(y,z) = \begin{bmatrix} 1 & sh_y & sh_z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$Sh(x,z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ sh_x & 1 & sh_z & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.10}$$

$$Sh(x,y) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ sh_x & sh_y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

3.2 Algoritmy pro zjednodušení sítě trojúhelníků

Redukce počtu trojúhelníků v síti trojúhelníků má velký význam pro zobrazování trojrozměrných modelů v počítačové grafice. Například nemá význam generovat všechny trojúhelníky pro objekt, jenž je příliš vzdálený od kamery, a odebrání těchto trojúhelníků nemá žádný vliv na výsledný obraz. Proto se zde setkáváme s pojmem LOD (*level of details*), který nám udává úroveň a množství detailů v modelu.

Mezi nejjednodušší a zároveň nejrychlejší algoritmy patří metoda *shlukování vrcholů* nebo *decimace sítě trojúhelníků*. Mezi ty výpočetně náročnější, avšak dosahující lepších výsledku, řadíme algoritmy *quadric error metrics* a *image-driven simplification*[14, str. 122-136].

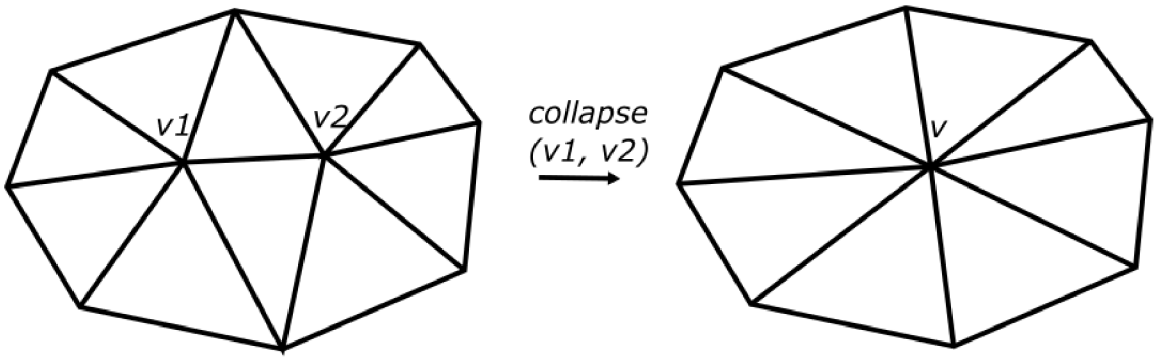
3.2.1 Shlukování vrcholů

Metoda shlukování vrcholů (*vertex clustering*) nabízí jednoduché a rychlé zjednodušení sítě trojúhelníků, která je však velice komplexní. Původní algoritmus byl navržen v roce 1992 Jarkem Rossignacem a Paulem Borrelem. Od té doby byl několikrát upraven a vylepšen dalšími lidmi.

Algoritmus funguje na principu ohodnocování vrcholů. Každému vrcholu je přiřazena důležitost na základě toho, zda se nachází na velké stěně nebo v oblasti, která je hodně zakřivena. Následně se celý model obklopí trojrozměrnou mřížkou a v každé buňce se vrcholy shluknou do toho nejdůležitějšího. Čím menší je rozlišení mřížky, tím je výsledný model jednodušší, a naopak při použití velice jemné mřížky se model pozmění minimálně. Všechny trojúhelníky, jejichž vrcholy se během procesu shlukování zřítí do sebe, jsou vymazány[15].

Přednosti tohoto algoritmu je, že pracuje s každým trojúhelníkem zvlášť, tudíž nezachovává topologii. Proto nepotřebuje žádnou validní topologii a poradí si s náročnými a složitými modely, které by jiné algoritmy špatně zvládaly. Další výhodou je, že je velice jednoduchý na implementaci a patří k nejrychlejším algoritmům v této oblasti.

K jeho největším nevýhodám patří vizuálně ošklivější výsledný obraz, protože není zachována topologie. Další nevýhodou je, že dopředu nelze odhadnout, kolik trojúhelníků



Obrázek 3.1: Princip metody zhroucení hran (*edge collapse*).

bude mít výsledný zjednodušený model a jedinou možností jak to zjistit, je provést samotnou operaci.

3.2.2 Decimace sítě trojúhelníků

Algoritmus decimace sítě trojúhelníků byl poprvé publikován v roce 1992 Williamem Schroederem, Jonathanem Zargem a Williamem Lorensonem. Největší úspěch tento algoritmus získal ve vědeckých a lékařských kruzích.

Algoritmus decimace sítě trojúhelníků funguje na principu zhroucení hran[14, str. 21-24] (*edge collapse*). Princip jednoduchého zhroucení hran lze vidět na obrázku 3.1. Algoritmus několikrát prochází všechny vrcholy a v každém průchodu se podle daných kritérii rozhoduje, jestli daný vrchol odstraní nebo ne. Vrcholy zvažované pro odstranění jsou rozděleny do pěti kategorií, jak lze vidět na obrázku 3.2. Vrcholy, které nesplňují zadaná kritéria, jsou vymazány společně se všemi trojúhelníky ke kterým náleží, a nově vznikla mezera je zaplněna pomocí triangulace. Každá skupina má vlastní kritéria, která jsou porovnána s hodnotou prahu nastavenou uživatelem. Algoritmus prochází objekt stále dokola a na konci každého průchodu zvedne práh, dokud není splněna požadovaná úroveň zjednodušení[16].

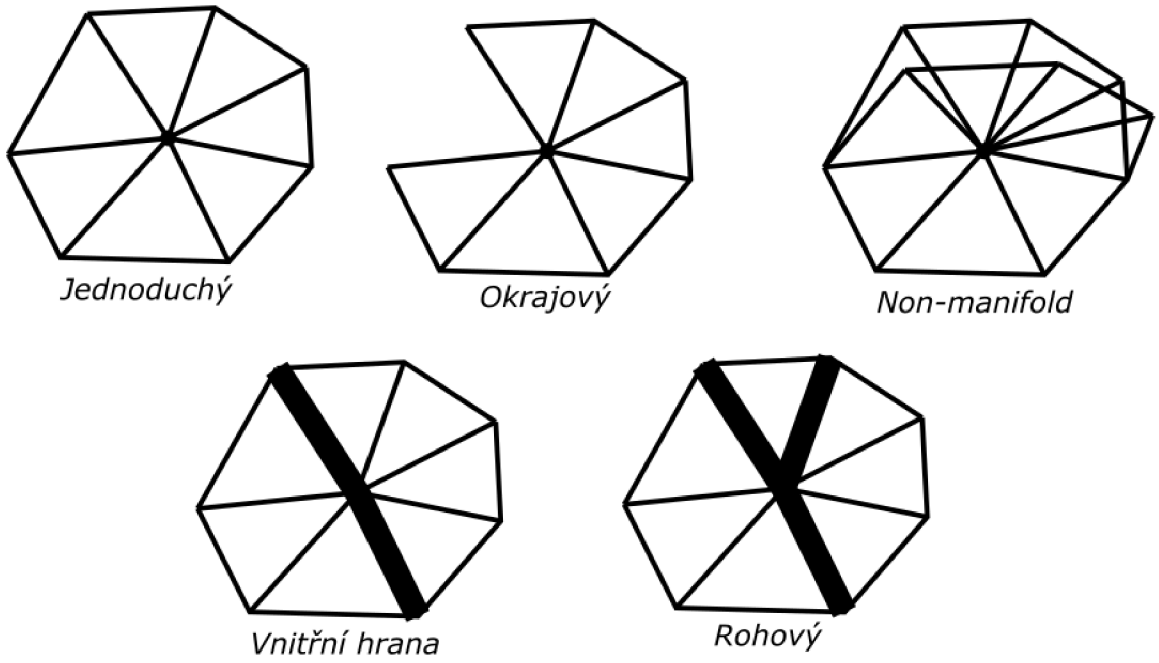
Hlavní výhodou je jeho rychlost a schopnost odstranit velké množství nadbytečných trojúhelníků vygenerovaných při některé triangulační metodě. Tato metoda také zachovává topologii, a proto vrcholy výsledného modelu jsou podmnožinou vrcholů původního.

Mezi jeho nevýhody patří méně přesný výsledný model. Kvůli tomu, že algoritmus prochází model několikrát, je těžké dopředu odhadnou výsledný počet trojúhelníků. Taky špatně zvolený práh může vést k přílišnému zjednodušení modelu, popř. k neúměrně dlouhému trvání běhu algoritmu.

3.2.3 Quadric Error Metrics

Algoritmus *Quadric Error Metrics* nabízí skvělý kompromis mezi rychlostí a věrností výsledného modelu a stále je poměrné jednoduchý na implementaci. Poprvé byl publikován v roce 1997 Michaelem Garlandem a Paulem Heckbertem.

Algoritmus funguje na principu souhrnného měření zakřivení plochy. Nejprve jsou nalezeny všechny páry vrcholů. Ty jsou následně vloženy do prioritní fronty, kde jsou seřazeny podle jejich kvadrické chyby (*quadric error*) a slučovány v pořadí od nejmenší chyby po největší. Sloučení jednoho páru ovlivní i všechny sousední. Tyto sousední páry jsou znovu



Obrázek 3.2: Rozdělení vrcholů podle typu pro metodu decimace sítě trojúhelníků.

vyhodnoceny a opětovně umístěny do fronty na odpovídající místo. Algoritmus takto pokračuje, dokud není dosaženo požadované úrovně zjednodušení.

Kvadrika je čtvercová matice čtvrtého stupně, která uchovává informaci o jedné nebo více plochách. Díky této informaci lze jednoduše spočítat vzdálenost vrcholu ke každé ploše, jenž kvadrika reprezentuje. V případě že jsou dva vrcholy sloučeny, se nová kvadrika spočte jako součet kvadrik těchto dvou vrcholů[6].

Kapitola 4

Vyvažování zátěže

Tato kapitola se bude věnovat vyvažováni zátěže (load balancing). Nejprve bude popsán jeho účel a následně probrány jednotlivé přístupy se kterými se běžně můžeme setkat.

Vyvažování zátěže slouží k správné distribuci jednotlivých zdrojů a zajišťuje rovnoměrné vytížení všech komponent. Tato metodika se týká velkého množství počítačových komponent, jako je procesor nebo disková jednotka. V téhle práci ovšem budeme pohlížet na vyvažování zátěže z pohledu počítačových sítí.

Load balancery můžeme rozdělit do tří kategorií podle toho jak fungují:

- Load Balancer na staně klienta
- Load Balancer na straně serveru
- Look-a-side Load Balancer

4.1 Load Balancer na straně klienta

Load Balancer implementovaný v klientovi je nejjednodušší přístup k vyvažování zátěže. Funguje na principu, že klient zná adresy vzdálených serverů a sám si z nich vybírá. Dále je zde předpoklad, že klient zná zátěž těchto serverů, ale tento předpoklad nemusí být splněn. Klient se potom sám podle nějakého algoritmu[3] nebo například podle plánovacího algoritmu Round-robin rozhodne, ke kterému serveru se připojí. Při připojení se obvykle aktualizují informace o vytížení serveru.

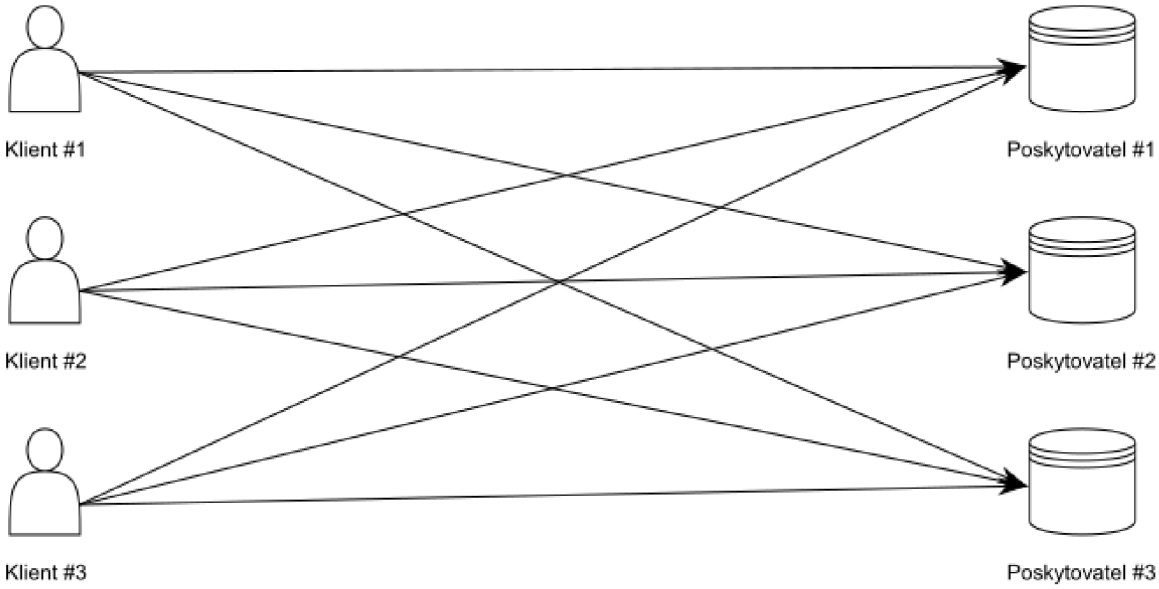
Největším přínosem tohoto přístupu je jeho výkonnost. Ta je takto vysoká z důvodu, že odpadá jakákoliv další režie na navazování dalších spojení.

Jeho největší nevýhodu je nutnost implementovat složitého klienta. Takový klient si musí uchovávat informace o všech serverech, jejich vytíženosti a možnostech, jenž nabízí. Dále je potřeba, aby tento klient sám implementoval nějaký vyvažovací algoritmus. Další problémy jsou špatná přenositelnost kódu a komplikace se zabezpečením.

Obrázek tohoto přístupu lze nalézt 4.1.

4.2 Load Balancer na straně serveru

Vyvažování zátěže na straně serveru, někdy též zvané *proxy load balancing*, patří dnes k nejhojněji používanému přístupu. Tento koncept funguje následovně. Existuje jeden server, který se pro všechny uživatele tváří jako koncový. Ve skutečnosti se jedná o Load Balancer,



Obrázek 4.1: Koncept Load Balanceru implementovaného na straně klienta.

který přeposílá všechny požadavky od klientů na skutečné koncové servery, vyčká na jejich odpověď a následně ji zašle zpět klientovi. Load Balancer si uchovává seznam všech aktivních serverů a pomocí nějakého vyvažovacího algoritmu[2] distribuuje všechny požadavky. Proxy Load Balancery lze ještě rozdělit podle toho, zda pracují na transportní vrstvě (L3/4) nebo aplikační (L7). Load Balancer pracující na transportní vrstvě má výhodu v tom, že má mnohem menší latenci. Děje se tak, protože spojení od klienta je jednoduše přerušeno, příchozí data jsou zkopírovány a je otevřeno nové spojení ke koncovému serveru.

Naproti tomu Load Balancer pracující na aplikační vrstvě nabízí možnost data zpracovat a připojit klienta k nejvíce vyhovujícímu serveru. Vyvažování zátěže na transportní vrstvě se nejvíce hodí, pokud se snažíme docílit velice nízké latence, popř. se snažíme o co nejmenší vytížení zdrojů. Naopak Load Balancer na aplikační vrstvě nabízí skvělé možnosti využití, pokud se jednotlivé požadavky od sebe příliš odlišují nebo pokud potřebujeme specificky rozdělit jednotlivé požadavky.

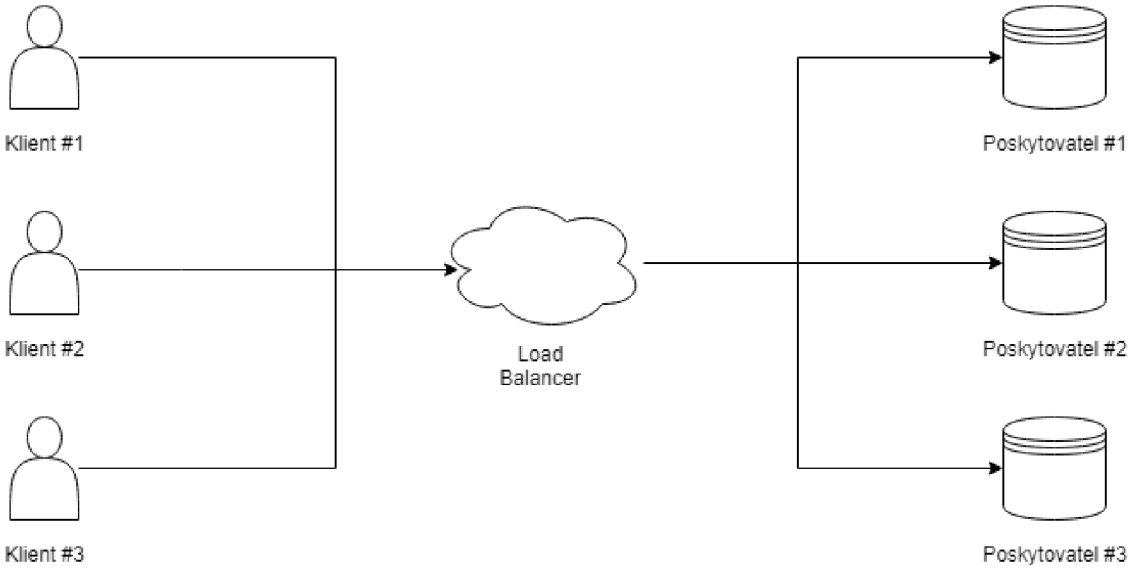
Obecně velkou výhodou Load Balancerů na straně serveru je jejich schopnost odstínit klienta od celkové infrastruktury serverů a taky jednodušší implementace ověřovaní klientů.

Nevýhodami tohoto přístupu je vyšší zpoždění, protože vše prochází skrz Load Balancer. Dále je zde také problém s omezením, protože škálovatelnost systému je omezena maximální propustností Load Balanceru.

Koncept vyvažování zátěže na straně serveru je vyobrazen 4.2.

4.3 Look-a-side Load Balancer

Look-a-side Load Balancer, občas také označovaný jako *external Load Balancer* nebo *one-arm Load Balancer*, je speciální typ vyvažování zátěže, při kterém stojí Load Balancer bokem mimo základní infrastrukturu. Základním principem tohoto přístupu je, že se klienti dotazují Load Balanceru na nejlepší možný server. Ten jim odpoví adresou serveru nejvíce odpovídajícímu požadavku klienta. Klient se následně sám připojí na koncový server. Load



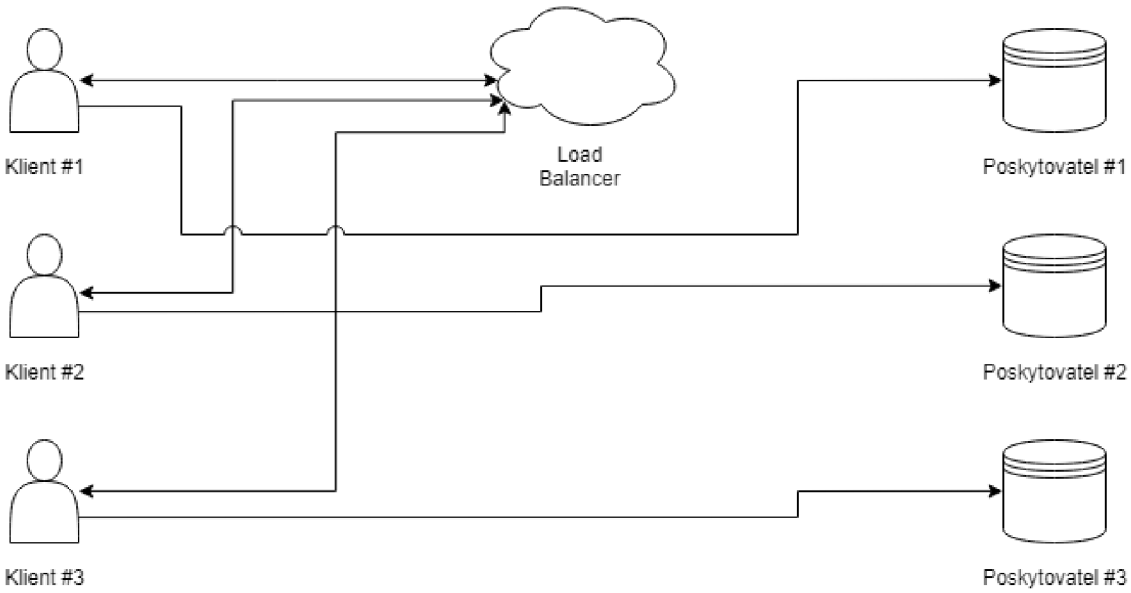
Obrázek 4.2: Koncept Load Balanceru implementovaného na straně serveru.

Balancer má v sobě implementovaný algoritmus pro vyvažování zátěže a také si udržuje aktuální seznam všech serverů.

Na rozdíl od serverového Load Balanceru, není look-a-side Load Balancer omezen svou propustností, takže se hodí zejména pro velice objemné přenášení dat mezi klientem a serverem.

Nevýhodou je náročnější implementace ověřování klientů, protože všechny strany komunikují mezi sebou.

Tento koncept je lépe vysvětlen na obrázku 4.3.



Obrázek 4.3: Koncept Look-a-side Load Balanceru.

Kapitola 5

Existující nástroje pro implementaci grafických a cloudových řešení

V této kapitole budou popsány jednotlivé knihovny a aplikace, které se zabývají stejnou problematikou jako táto práce. Každý nástroj bude popsán, budou uvedeny jeho největší přednosti a zápory. Smyslem je nalézt vhodné nástroje pro tvorbu vlastní aplikace.

Z pohledu této práce můžeme rozdělit následující řešení do dvou skupin. První se zaobírá tvorbou cloudových systému a druhá zpracováním trojrozměrných modelů.

5.1 Nástroje na tvorbu cloudových aplikací

5.1.1 gRPC

gRPC je open source nástroj pro tvorbu aplikací umožňujících RPC. Umožňuje tvorbu multiplatformních řešení a poskytuje rozhraní pro velké množství programovacích jazyků (C++, C#, Java, Python, Objective-C, PHP a další)[8].

RPC (*remote procedure call* neboli *vzdálené volání procedur*) je technologie umožňující vykonávat procedury v jiném adresářovém prostoru, než ve kterém je procedura volána. Typickým příkladem této služby je výpočet složitých matematických operací na vzdáleném serveru.

Metoda RPC umožňuje uživateli volat procedury stejně, jako kdyby byly naprogramovány lokálně. V objektově orientovaném programování se tento přístup nazývá RMI (*remote method invocation*).

RPC funguje v následující posloupnosti:

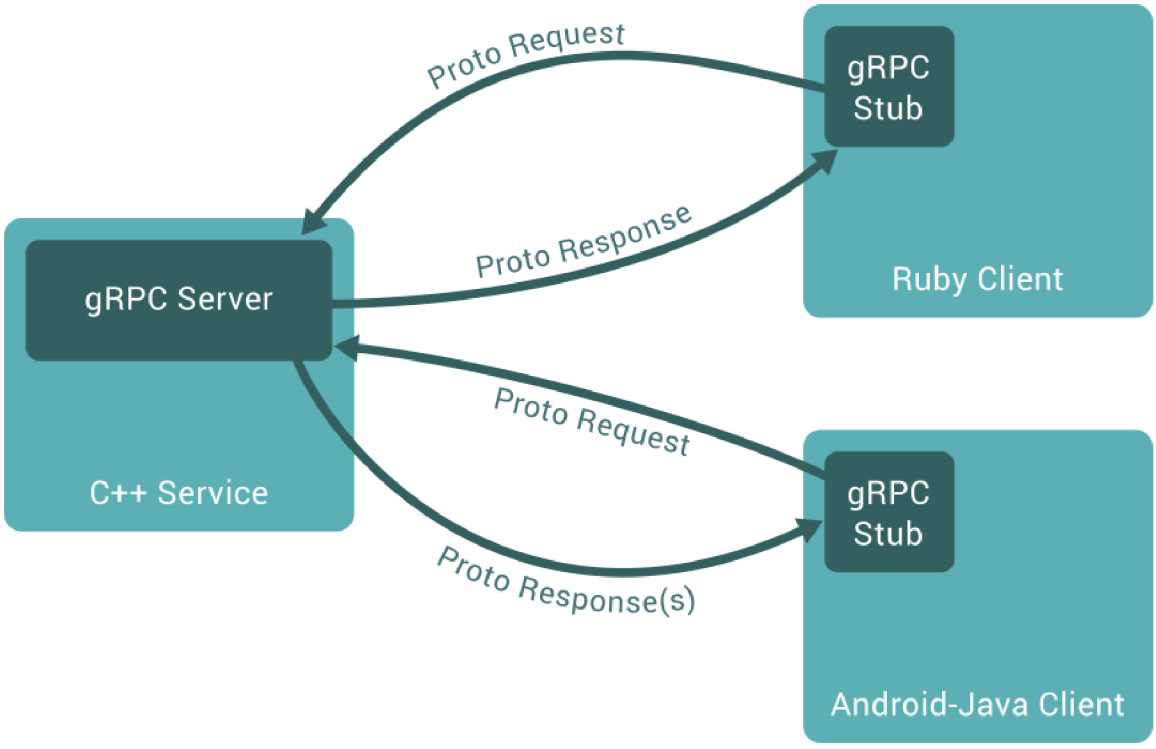
1. Identifikátor procedury a její parametry jsou převedeny do formy vhodné pro přenos. Tento proces se nazývá *marshalling*.
2. Operační systém klienta zašle zprávu operačnímu systému serveru.
3. Operační systém klienta odešle zabalená data operačnímu systému serveru.
4. Server data zpracuje do původní formy. Tento proces se nazývá *unmarshalling*.
5. Sever provede s přijatými daty požadovanou proceduru.

6. Server data odešle zpět klientovi použitím stejných kroků.

gRPC bylo vytvořeno společnosti Google. Pro přenos dat využívá protokol HTTP/2. Pro reprezentaci přenášených datových struktur a procedur využívá jazyk Protocol Buffers (*Protobuf*).

Protobuf byl také vytvořen společností Google, jako interně používaný nástroj pro serializaci a deserializaci strukturovaných dat nezávisle na zvoleném programovacím jazyku. Jeho nejširší využití lze nalézt v přenosu po síti nebo ukládání strukturovaných dat. Cílem Protocol Bufferu je být rychlým a kompaktním řešením pro proces serializace.

Největší přínosy, jenž gRPC přináší jsou možnosti obousměrného streamování dat a integrované ověřování za pomocí SSL nebo Token-based autentitazace. Další výhodou je možnost implementovat klienta a server v různých programovacích jazycích dle potřeby. Tato struktura potom následně spolu komunikuje díky Protocol Bufferu. Příklad této infrastruktury lze vidět na obrázku 5.1.



Obrázek 5.1: Ukázka propojení služeb za pomocí frameworku gRPC. Převzato z [9].

5.1.2 ZeroMQ

ZeroMQ (někdy psáno jako ØMQ) je knihovna sloužící pro tvorbu klient-server aplikací pracující pod licencí LGPL. ZeroMQ je napsané v programovacím jazyky C++, ale i přesto nabízí rozraní pro mnoho ostatních jazyků. Mezi ty nejznámější patří C, Java, Python, ale třeba i Haskell nebo Lua[11].

ZeroMQ nabízí nadstavbu nad klasickými BSD sokety a zároveň přináší mnoho vylepšení a zjednodušení. Nabízí totiž asynchronní odesílání zpráv, jejich zpracování na pozadí a abstrakci nad použitými komunikačními protokoly.

Knihovna ØMQ umožňuje tvorbu tzv. komunikačních strategií. Strategie lze různě kombinovat, aby bylo docíleno vždy nejlepšího výsledku. Můžeme je rozdělit do 4 částí:

1. PAIR
2. REQ-REP
3. PUB-SUB
4. PUSH-PULL

PAIR strategie umožňuje obousměrné, popř. jednosměrné propojení dvou procesů, přičemž každý z těchto procesů může být spuštěn na jiném zařízení. Svým způsobem se nejvíce blíží klasickým BSD soketům.

REQ-REP neboli *request-response* (požadavek-odpověď) strategie popisuje klasický přístup k vytváření serverů. Klient posílá požadavky, server je obsluhuje a zasílá odpovědi.

PUB-SUB neboli *publish-subscribe* (vydání-odebírání) je strategie, kdy server (Publisher) neposílá zprávu přímo klientům, ale publikuje kategorizovanou zprávu. Klienti (subscribers) se mohou přihlásit k odběru těchto zpráv nebo si je filtrovat na základě vybraných kategorií.

PUSH-PULL je rozšíření předchozí strategie PUB-SUB, kdy servery jsou schopné zprávy aktivně zasílat všem klientům (Push) a klienti jsou schopni si žádat neustále o nové zprávy (Pull).

5.1.3 Apache Kafka

Apache Kafka je open-source platforma sloužící pro zpracování toku dat. Platforma byla vytvořena společností LinkedIn a byla implementována v programovacích jazycích Java a Scala[5].

Apache Kafka má tři klíčové vlastnosti, které jsou nezbytné pro zpracování toku dat. Jsou to:

- Implementace PUB-SUB strategie pro tok záznamů
- Uložení záznamů do odolné struktury
- Zpracování toku záznamů, jakmile se objeví

Díky těmto vlastnostem se tato platforma hodí především pro tvorbu pipeline (*potrubí*) pro spolehlivé streamování záznamů mezi aplikacemi nebo systémy a pro tvorbu aplikací umožňujících streamování v reálném čase, které reagují nebo přenášejí stream dat.

Apache Kafka funguje na principu *clusterů* (*shluku*), kdy jeden nebo více serverů tvoří cluster, který může obsahovat několik datacenter. Každý cluster ukládá streamy záznamů. Ty jsou ukládány podle kategorií, které jsou nazvány *topics*. Každý záznam je tvořen klíčem, hodnotou a časovou značkou.

Kafka se skládá ze čtyř API. Jedná se o:

1. Producer API
2. Consumer API
3. Streams API

4. Connector API

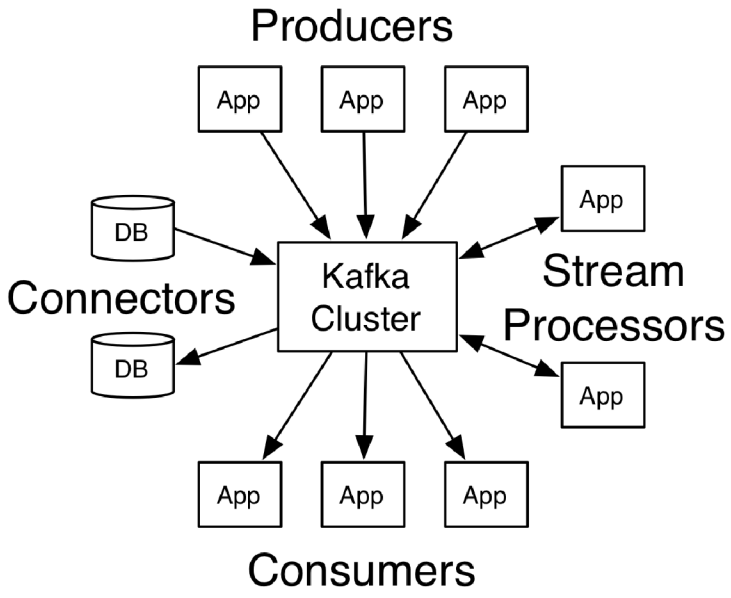
Producer API slouží k publikování streamů záznamů do topiků.

Consumer API nabízí možnost aplikacím přihlásit se k jednotlivým topikům. Uživatelé (*consumer*) jsou rozdělení do skupin a každý publikovaný záznam je zaslán jednomu uživateli ve skupině.

Streams API slouží ke zpracovávání stremů. Umožňuje aplikaci zpracovat vstupní stream z jednoho nebo více topiků a zpracovat ho na výstupní stream, který může být zařazen do jednoho nebo více topiků.

Connector API umožňuje tvorbu a běh producentů (*producer*) a uživatelů (*consumer*), kteří spojují topiky a existující aplikace nebo datacentra.

Příklad propojení jednotlivých rozhraní a náhled na fungování celého systému lze vidět na obrázku 5.2.



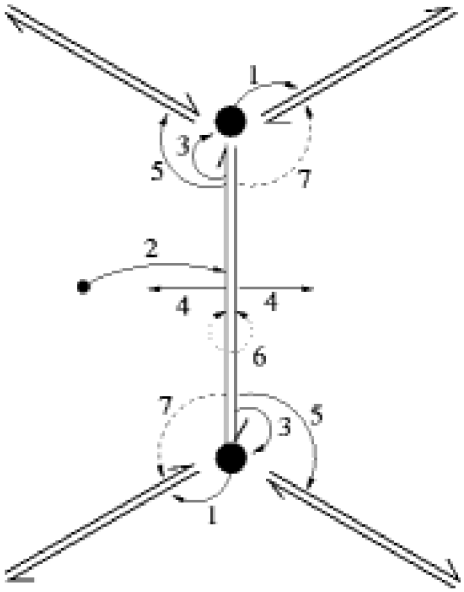
Obrázek 5.2: Schéma fungování systému Apache Kafka. Převzato z [5].

5.2 Nástroje pro práci s 3D modely

5.2.1 OpenMesh

OpenMesh je open-source knihovna nabízející datovou strukturu pro reprezentaci trojrozměrných objektů za pomocí sítě polygonů. Knihovna je implementovaná v programovacím jazyku C++ a nabízí rozhraní pro jazyky C++ a Python. Za jejím vývojem stojí skupina Computer Graphics Group z německé univerzity RWTH z Cách. Software je distribuován pod licencí BSD-3[18].

Cílem knihovny OpenMesh je vytvořit flexibilní, efektivní a jednoduše použitelnou datovou strukturu. Z tohoto důvodu nabízí struktury a algoritmy pro obecné sítě polygonů, ale i specializovanou strukturu pro reprezentaci za pomoci trojúhelníkové sítě. Dále také nabízí explicitní reprezentaci vrcholů, hran a stěn nebo třeba rychlý přístup k okolním prvkům.



Obrázek 5.3: Popis half-edge struktury. Převzato z [1].

Knihovna OpenMesh reprezentuje 3D modely pomocí tzv. *half-edge* struktury. Její princip lze nejlépe demonstrovat na obrázku 5.3. Každý vrchol odkazuje na půl-hranu, která z něj vychází (1). Každá stěna odkazuje na jednu půl-hranu, která je s ní svázána. Každá půl-hrana obsahuje odkaz na vrchol, do kterého směřuje (2), na stěnu, která ji náleží (4), na další half-edge (5), na identický obracený half-edge (6) a v některých případech i na předchozí half-edge ve stěně (7).

Díky způsobu, jakým je OpenMesh implementován, nepřináší žádnou nadbytečnou zátěž ve formě tabulky virtuálních metod nebo dynamického linkování. Dále nemá vysoké nároky na paměť a režii za běhu programu. Výhodou také je, že vstupní data jsou šablony, takže je vše dekódováno už při překladu programu.

5.2.2 OpenCTM

OpenCTM je open-source knihovna nabízející rozhraní formátu pro reprezentaci 3D objektů pomocí sítě trojúhelníků. Její hlavní snahou je vytvořit lehce uchopitelný, avšak univerzální formát. Knihovna OpenCTM je napsaná v programovacím jazyku C a zároveň přináší rozhraní pro spoustu jiných jazyků[7].

Největší předností OpenCTM je efektivita. Knihovna je schopna ve velice malém čase zpracovávat obrovskou síť trojúhelníků. Dále také nabízí kompresi jednotlivých modelů v případě, kdy je to potřeba.

Knihovna OpenCTM obecně není vhodná, pro cokoliv jiného než sítě trojúhelníků. Nedokáže si poradit s uchováním scény, není schopna zvládnout vícero modelů zároveň, neuchovává informace o nasvícení, materiálech, apod.

Kapitola 6

Příklady existujících webových řešení

V této kapitole budou popsány příklady existujících webových řešení pro zpracováni 3D modelů v cloudu. Budou zde popsány možnosti, jaké nabízí a jak fungují. Tato kapitola by měla sloužit hlavně jako příklad užití zamýšlené knihovny.

6.1 Blend4Web

Blend4Web[17] je open-source projekt, který umožňuje vytváření a zpracovávání interaktivních trojrozměrných webových aplikací v prohlížeči a vytváří nadstavbu nad webovými technologiemi jako je WebGl, WebAudio nebo WebVR.

Tento projekt byl vytvořen společností Triumph v roce 2014 jako nadstavba pro jejich předchozí projekt *Blender*. Blender je sada různých nástrojů pro tvorbu 3D modelů, animovaných filmů, vizuálních efektů a videoher.

Blend4Web pro svou práci používá modely vytvořené v editoru Blender. Ty lze vyexportovat přímo ve formátu HTML, popř. JSON a vložit je přímo na své webové stránky. Mezi podporované možnosti patří vytváření jednoduchých počítačových her, interaktivních scén apod. Blend4Web přináší také podporu virtuální reality díky integraci knihoven WebVR a ARToolKit.

Tento framework běží na operačních systémech Windows, Linux a MacOS a byl napsán v programovacích jazycích Python a C.

Kapitola 7

Návrh knihovny pro zpracování 3D modelů v cloudu

Cílem této kapitoly bude popsat návrh knihovny pro vytváření cloudových aplikací na bázi klient-server pracujících s 3D objekty. Do návrhu byla zahrnuta většina možných případů užití, se kterými by se budoucí uživatel mohl setkat, a zároveň bylo myšleno na snadnou rozšiřitelnost v případě, že by se vyskytly nové požadavky. Na základě následujících specifikací byla navržena knihovna Cloud3D.

7.1 Požadavky

Hlavním cílem této práce bylo umožnit jednoduše vytvářet klient-server aplikace pro zpracování 3D modelů. Základní myšlenkou takové aplikace je poskytovatel služeb zpracovávající požadavky vzdáleně na serveru či v cloudu. K tomuto poskytovateli by se následně připojovali klienti s požadavky na zpracování trojrozměrných modelů, které jsou příliš náročné na jejich výpočetní výkon.

Kromě základní myšlenky, návrh knihovny, umožňující tvorbu takových aplikací, vychází z následujících předpokladů:

- Intuitivní a jednoduché API pro rychlou tvorbu nových aplikací
- Zátěžová škálovatelnost výsledného produktu
- Zabezpečení spojení

Intuitivní API je nesmírně důležité, aby uživatelé výsledné knihovny mohli rychle a jednoduše vytvářet nové aplikace. Na druhou stranu je podstatné, aby jednoduchost nebyla vynucena na úkor účelnosti. Nejdůležitější a zároveň nejsložitější částí je proto poskytovatel služeb, u kterého je nutno dát uživateli naprostou svobodu v definování vlastních služeb, avšak stále pomocí jednoduchého rozhraní.

Dalším nezbytným prvkem je škálovatelnost vytížení jednotlivých poskytovatelů služeb. Protože se předpokládá, že k poskytovateli můžou v jednu chvíli přistupovat desítky až stovky klientů současně, je nezbytné, aby bylo možno jednoduše rozložit zátěž mezi jednotlivé poskytovatele.

Velký důraz při návrhu byl také kladen na zabezpečené spojení. Protože není žádoucí, aby byl server napaden a klientská data zneužita, je nezbytné umožnit navázat šifrované spojení mezi jednotlivými stranami.

Mezi další požadavky, na které bylo při návrhu nahlíženo patří:

- Multi-platformnost
- Efektivní práce s 3D modely

7.2 Rozložení zátěže mezi poskytovatele

Klientská část aplikace komunikuje s poskytovateli služeb, kteří za ni provádějí veškeré požadované operace. Aby komunikace proběhla co možná nejhladším způsobem, je důležité správně rozložit zatížení mezi jednotlivé poskytovatele služeb. K tomuto účelu slouží technika tzv. vyvažování zátěže (Load Balancing).

Existují tři hlavní přístupy k vyvažování zátěže. Tyto přístupy jsou detailněji diskutovány v kapitole 4. Pro účely této práce byl zvolen a implementován Look-a-side Load Balancer.

Oproti vyvažování na straně klienta nabízí jednoduší možnost ověřování serverů a klientů. To je důležité, protože jedním z hlavních požadavků na knihovnu Cloud3D je právě zabezpečení.

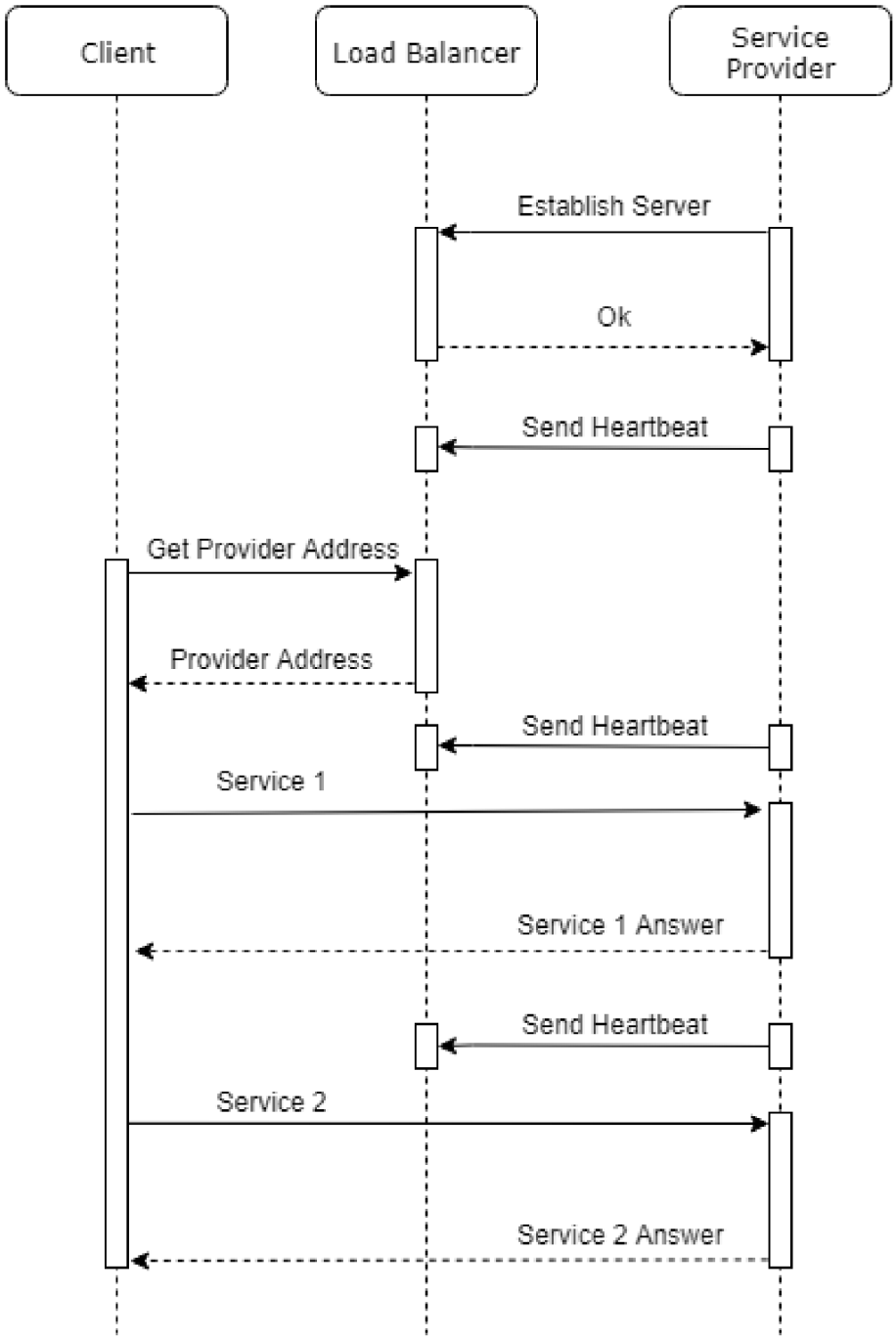
Na rozdíl od Load Balanceru na straně serveru, není nutné na Look-a-side Load Balancer zasílat 3D model, který chceme zpracovat. Protože se předpokládá práce s opravdu velkými soubory, bylo by možné při vysokém vytížení jednoduše Load Balancer zahltit. Díky tomu je model přenesen pouze dvakrát (klient $\rightarrow$ poskytovatel $\rightarrow$ klient), na rozdíl od implementace na straně serveru, kdy jej bylo nutné přenášet čtyřikrát (klient $\rightarrow$ LB $\rightarrow$ poskytovatel $\rightarrow$ LB $\rightarrow$ klient).

Samotné fungování Load Balanceru je následující: Load Balancer si udržuje seznam všech poskytovatelů, kteří právě běží. V případě, že je mu zaslán dotaz od klienta, vybere poskytovatele s nejmenším vytížením, jenž splňuje všechny požadavky, a zašle jeho adresu zpět klientovi. Pokud existuje více poskytovatelů služeb, kteří splňují všechny podmínky a mají stejné vytížení, je server vybrán náhodně. Požadavky, které klienta mohou zajímat, jsou služby, které poskytovatel nabízí a také operace, které umožňuje provádět.

7.3 Proces komunikace

Způsob, jakým mezi sebou jednotlivé komponenty komunikují, je podstatnou částí celého systému. V procesu se vyskytují tři účastníci. Klient, jenž zasílá požadavky. Poskytovatel služby, který tyto požadavky vyřizuje. A Load Balancer, jehož úkolem je rovnoměrně vyvážit zátěž mezi poskytovatele. Komunikace probíhá mezi všemi účastníky navzájem.

Komunikaci zahajuje poskytovatel zasláním zprávy na Load Balancer, že je v provozu, že je připraven přijímat požadavky, jaké služby poskytuje a v jaké verzi. Následně začne periodicky zasílat zprávy o tom, že je v provozu a jaké je jeho vytížení. Od tohoto okamžiku může kdykoliv přijít požadavek od klienta na Load Balancer, jenž obsahuje seznam služeb, o které má klient zájem a číslo verze, v jaké by měl poskytovatel pracovat. Load Balancer vše vyhodnotí a zašle zpět klientovi odpovídající adresu poskytovatele. Klient následně zašle požadavek na adresu poskytovatele, kterou obdržel. Poskytovatel zpracuje požadavek a odpoví klientovi požadovaným způsobem. Celý proces komunikace lze nalézt na přiloženém sekvenčním diagramu 7.1.



Obrázek 7.1: Sekvenční diagram komunikace služeb knihovny Cloud3D.

7.4 Zabezpečení

Protože aplikace využívající knihovnu Cloud3D budou komunikovat převážně přes internet, je potřeba umožnit uživateli tuto komunikaci zabezpečit. K tomu je využit protokol SSL, sloužící k šifrování komunikace a autentizaci komunikujících stran. Zabezpečení se vyskytuje u všech komunikujících objektů, tj. klient, poskytovatel služeb a Load Balancer. Poskytovatel služeb a Load Balancer využívají stejných autorizačních prostředků, proto se pro klienta tváří jako dva totožné subjekty. Ve své práci jsem se rozhodl, že je potřeba, aby autentizace proběhla na obou komunikujících stranách, tedy jak na straně klienta, tak na straně serveru. Aby komunikace mohla být zahájená šifrovaným způsobem, je potřeba, aby Load Balancer a poskytovatel služeb znaly následující údaje:

- Osobní certifikát serveru vydaný Certifikační Autoritou (CA)
- Privátní klíč serveru
- CA certifikát pro klientovou Certifikační Autoritu

V případě, že se klient chce připojit na takovýto server, je potřeba, aby měl k dispozici následující údaje:

- Osobní certifikát klienta vydaný Certifikační Autoritou (CA)
- Privátní klíč klienta
- CA certifikát pro Certifikační Autoritu serveru

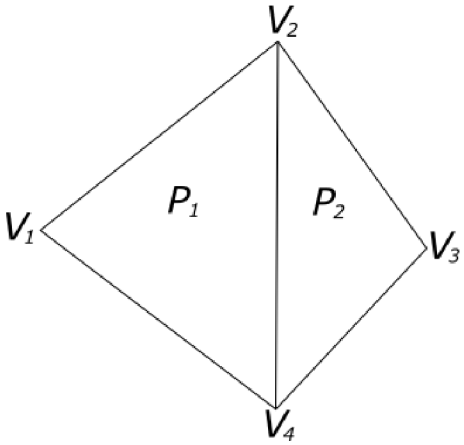
Pokud jsou všechny údaje platné, digitální podpisy jsou vzájemně zkontrolovány a obě strany autentizovaný, je zahájeno šifrované spojení mezi nimi.

7.5 Zpracování 3D objektů

Aby bylo možné vytvořit vysoce efektivní aplikaci na zpracování 3D objektů, je nutná jejich rychlá serializace a deserializace. Avšak zároveň je potřeba myslet, aby výsledný objekt byl co možná nejmenší, neboť se bude přenášet přes síť. Mimo jiné je potřeba myslet na to, aby takový objekt byl zachován ve své nezměněné podobě a v případě deserializace zpět nebyla ztracena žádná informace.

Serializace 3D modelů probíhá tímto způsobem. Nejprve jsou uloženy všechny vrcholy daného objektu do seznamu vrcholů. Následně se projdou všechny polygony a uloží všechny ukazatele do seznamu vrcholů, které polygon obsahuje. Takto může být 3D model jednoduše přenesen a zase opětovně složen. Výhodou tohoto přístupu je malá velikost a naprostá jednoznačnost. Ukázku této reprezentace lze vidět na obrázku 7.2.

V případě, že by se přenášely konkrétní vrcholy pro každý polygon, razantně by narostla výsledná velikost. Dále by také mohlo dojít k nejednoznačnosti, v případě že by dva vrcholy měly podobné souřadnice, popř. by tento rozdíl zanikl během přenosu. Pokud by se požila reprezentace polygonů pomocí seznamu hran, vzrostla by velikost přenášeného modelu oproti seznamu vrcholů. Více je tato problematika popsána v části 2.1.



$$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), ..., (x_4, y_4, z_4))$$
$$P_1 = (1, 2, 4)$$
$$P_2 = (2, 3, 4)$$

Obrázek 7.2: Příklad reprezentace objektu za pomocí seznamu vrcholů.

Kapitola 8

Implementace

V této kapitole budou popsány použité externí knihovny a vlastní způsob implementace jednotlivých problémů. Pro vytvoření knihovny Cloud3D byl použit programovací jazyk C++. Z externích knihoven byly použity zejména knihovny *gRPC* a *OpenMesh*. Pro správu verzí byl použit verzovací systém GitHub. Dále byl použit program CMake pro možnosti multiplatformní kompilace a přenositelnosti.

C++

Pro implementaci byl zvolen programovací jazyk C++. Jeho počátek se datuje do roku 1983 a autorem je Bjarne Stroustrup. C++ vzniklo jako rozšíření programovacího jazyka C. Na rozdíl od svého předchůdce umožňuje objektově orientované programování, vytváření šablon, zpracování výjimek a další užitečné koncepty moderních programovacích jazyků. Jazyk C++ byl zvolen pro implementaci především pro svou vysokou rychlost během chodu programu, kompatibilitu s většinou operačních systémů a v neposlední řadě pro znalost daného jazyka.

gRPC

Na základě znalostí získaných v kapitole 5.1, byl pro implementaci cloudové služby zvolen framework gRPC. Je to především z důvodu, že systém RPC přesně vystihuje problematiku, řešenou v této práci. Dále gRPC nabízí integrovanou podporu STL nebo obousměrný stream dat. A díky použití struktury ProtoBuf, umožňuje vytvářet klienty a poskytovatele v různých programovacích jazycích při jednoduše zachovatelné kompatibilitě.

OpenMesh

Knihovna OpenMesh byla vybrána především pro svoji rychlost a relativně jednoduché rozhraní. Pokud se uživatel nerozhodne jinak, tak je formát knihovny OpenMesh velice vhodný pro implementaci většiny algoritmů a operací, které by s modely mohly být prováděny. Díky tomu odpadá režie na další manipulaci s 3D objekty.

8.1 Vlastní implementace

8.1.1 Definice služeb

Protože každý uživatel, který přijde do styku s knihovnou Cloud3D bude mít od ní lehce odlišné požadavky, je důležité mu umožnit je implementovat. Z tohoto důvodu vznikla třída Service, respektive její specializace ModelToModelService a ModelToNumbersService. Jejich základních premisou je umožnit nadefinovat jakoukoliv službu, která bude zpracovávat příchozí model a umožní vrátit požadovanou odpověď klientovi. Aby tato služba byla validní je potřeba nadefinovat její přesný název a konkrétní funkci, která bude zpracovávat příchozí modely a vytvářet požadované odchozí objekty.

Všechny příchozí a odchozí informace jsou umístěny v kontejnerech třídy vektor. To se děje z důvodu, že uživateli je umožněno přenášet neomezený počet 3D modelů najednou.

Příklad konkrétního nadefinování služby lze nalézt ve výpisu 8.1.

```
//Sluzba musi byt definovana jako funkce s navratovou hodnotou void a jedinym
argumentem
//Tato konkretni sluzba prijima mesh modely a vraci mesh modely
void meshOperation(ModelToModelService *service)
{
    for (auto mesh : service->getIncomingMeshModels())
    {
        /***
        * Zde bude operace s modelem
        ***/
        service->addOutgoingMeshModel(outMesh);
    }
}
```

Výpis 8.1: Příklad definice nové služby pomocí knihovny Cloud3D

8.1.2 Rozhraní

Knihovna Cloud3D je tvořena třemi dílčími součástmi. Jedná se o knihovny CloudClient, LoadBalancer a ServiceProvider. Důvodem, proč jsou tyto části rozděleny a nejsou spojeny v jednu obří knihovnu je předpoklad, že uživatel implementující služby pomocí knihovny Cloud3D, bude vytvářet jednotlivé součástí odděleně. Tudíž je mu zabráněno v chybném použití některých funkcí a zbytečně není zatížen rozhraním, které v danou chvíli nepotřebuje.

Definice poskytovatele

Poskytovatel je vytvořen pomocí třídy ServiceProvider. Ta potřebuje ke svému spuštění znát ip adresu a port na kterém má poskytovatel běžet, ip adresu a port na kterém běží Load Balancer, číslo verze a informaci, jak často má být odeslána zpráva pro Load Balancer ohledně vytížení stavu poskytovatele. Pokud se uživatel rozhodne využít šifrovaného spojení,

Metoda	Popis
Run	Spustí poskytovatele služeb
setModelsToModelsService	Přidá novou službu přijímající model a vracející model
setModelsToNumberService	Přidá novou službu přijímající model a vracející sadu čísel
setUsageFunction	Nastaví funkci pro výpočet zátěže poskytovatele
Service::getName	Vrátí název konkrétní služby
Service::getIncomingModel	Vrátí vektor všech příchozích modelů
Service::getIncomingMeshModel	Vrátí vektor všech příchozích modelů ve formátu mesh
Service::setOutgoingModel	Nastaví vektor modelů k odeslání zpět klientovi
Service::setOutgoingMeshModel	Nastaví vektor modelů ve formátu mesh k odeslaní zpět klientovi
Service::setOutgoingPoints	Nastaví vektor číselných hodnot k odeslání zpět klientovi
addOutgoingModel	Přidá model do vektoru modelů určených k odeslání klientovi
addOutgoingMeshModel	Přidá model ve formátu mesh do vektoru modelů určených k odeslání klientovi
addOutgoingPoint	Přidá číselnou hodnotu do vektoru určených k odeslání klientovi

Tabulka 8.1: Rozhraní pro poskytovatele služeb.

je nutné dále nadefinovat cestu k certifikátu daného poskytovatele, jeho privátní klíč a nakonec CA certifikát klienta.

Dalším potřebným krokem je definice všech služeb, které bude poskytovatel nabízet. Tento krok je více popsán v kapitole 8.1.1.

Mimo to, má uživatel možnost nadefinovat vlastní funkci pro výpočet vytížení poskytovatele. Tento údaj je pravidelně zasílán na Load Balancer, který díky tomu vybírá nejméně zatíženého poskytovatele. Je pouze na uživateli, jakou metriku zvolí. Pokud se tuto možnost rozhodne nevyužít, jsou poskytovatele vybíráni náhodně.

Celé veřejné rozhraní pro poskytovatele služeb a jeho popis lze nalézt v tabulce 8.1. Příklad jednoduchého vytvoření poskytovatele služeb lze nalézt ve výpisu 8.2. Diagram tříd reprezentující strukturu poskytovatele služeb lze nalézt na obrázku 8.1.

Definice služby Load Balancer

Load Balancer slouží k rovnoměrnému rozložení zátěže mezi poskytovatele služeb. Jeho funkcionalita je definována ve třídě LoadBalancer. Pro správné fungování musí uživatel nadefinovat adresu a port na kterém služba poběží a také údaj, jak často má očekávat zprávy od poskytovatelů, aby mohl vyloučit ty neaktivní. Pokud se uživatel rozhodne využít šifrovanou formu komunikace je dále potřeba zadat cestu k certifikátu pro Load Balancer, jeho privátní klíč a CA certifikát klienta.

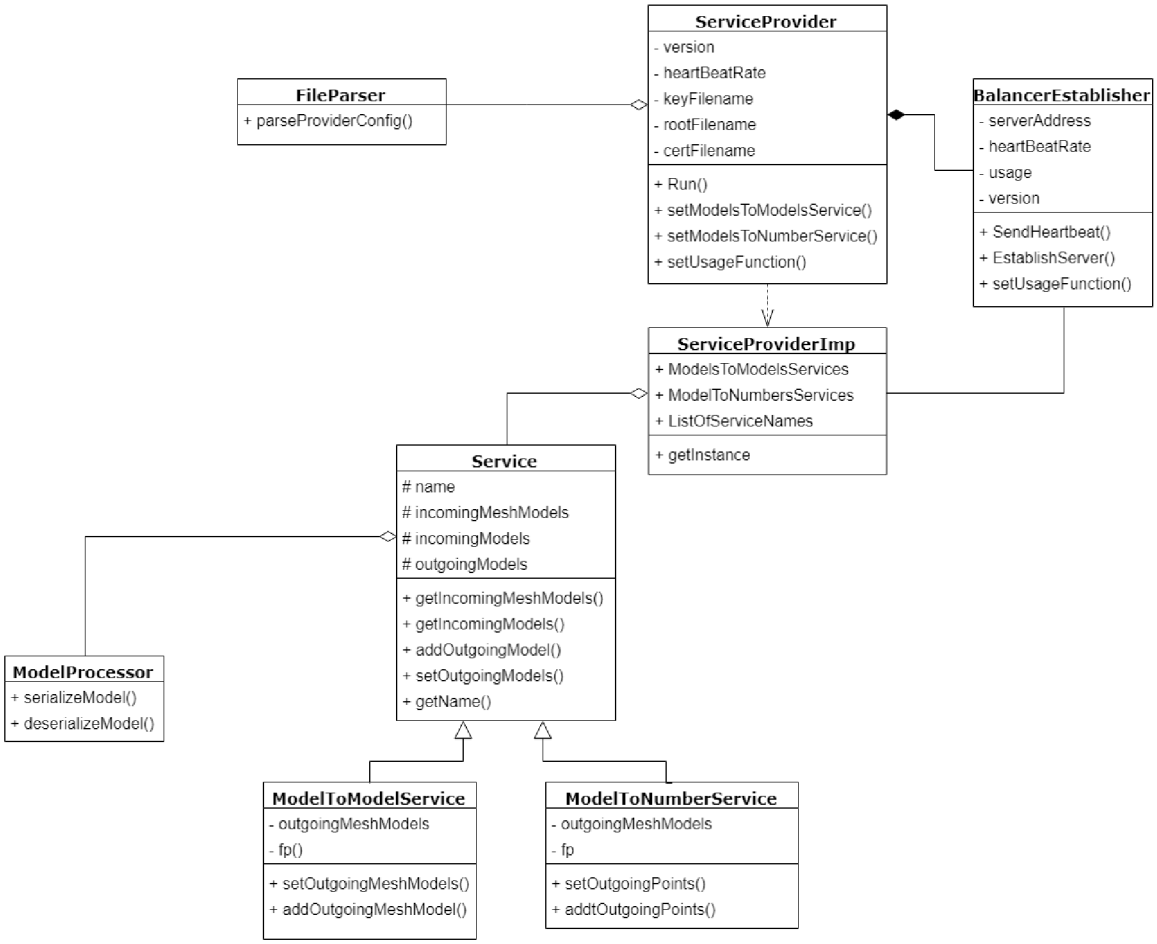
Veškeré metody dostupné pro Load Balancer a jejich popis lze nalézt v tabulce 8.2. Příklad jednoduchého vytvoření služby Load Balancer lze nalézt ve výpisu 8.3. Diagram tříd reprezentující strukturu Load Balanceru lze nalézt na obrázku 8.2.

```
int main(int argc, char *argv[])
{
    /**
    Je nutne zadat cestu ke konfiguracnimu souboru, pokud se rozhodneme vytvaret poskyt
    **/
    std::string configFile = "/path/to/configfile";

    ServiceProvider* provider = new ServiceProvider(configFile);

    //Kazda sluzba musí mit unikatni identifikator
    std::string serviceName1 = "MeshSmooth";
    std::string serviceName2 = "VertexClustering";
    std::string serviceName3 = "FeaturesExtraction";

    //Vytvoreni nove sluzby vyhlazeni, ktera je definovana ve funkci
meshSmooth, etc.
    ModelToModelService service1(serviceName1, meshSmooth);
    ModelToModelService service2(serviceName2, vertexClustering);
    ModelToNumberService service3(serviceName3, extracFeatures);

    //Prirazeni nove sluzby do poskytovatele
    provider->setModelsToModelsService(service1);
    provider->setModelsToModelsService(service2);
    provider->setModelsToNumberService(service3);

    /**
    Nastaveni funkce pro vypocet vyuziti poskytovatele sluzeb
    Tento krok je nepoviny, ale pomaha lepe distribuovat zatez
    **/
    provider->setUsageFunction(UsageFunction);

    //Spusteni poskytovatele sluzeb
    provider->Run();

    return 0;
}
```

Výpis 8.2: Příklad vytvoření a spuštění služby poskytovatele služeb pomocí knihovny Cloud3D

Metoda	**Popis**
Run	Spustí Load Balancer

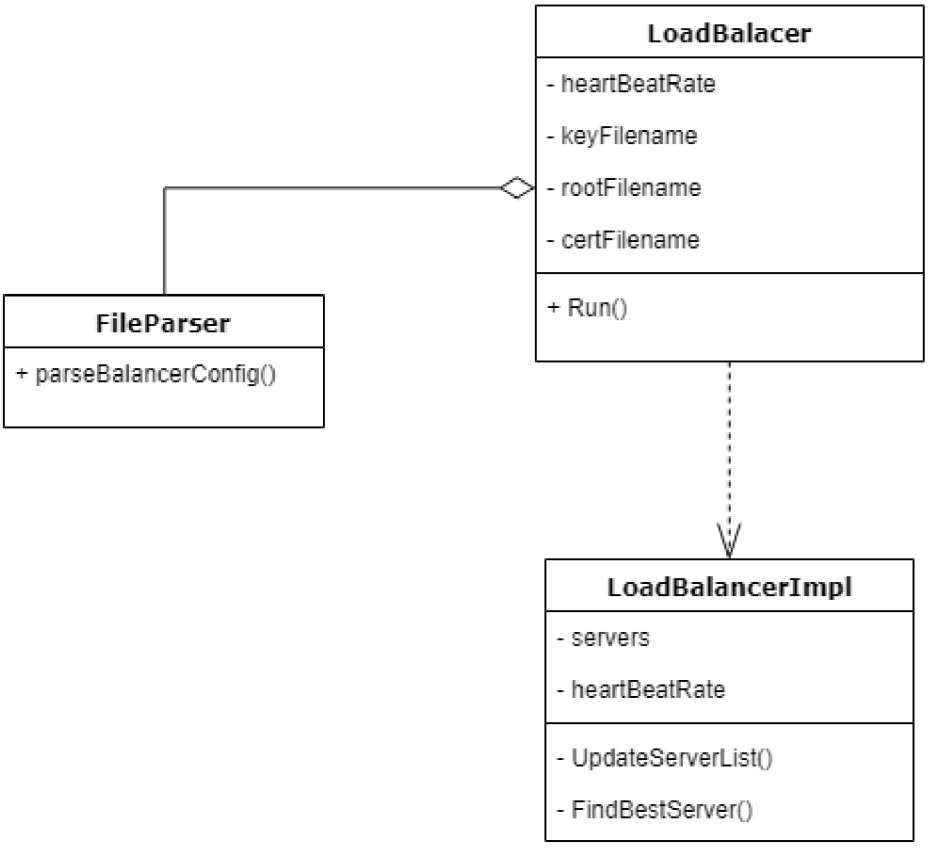
Tabulka 8.2: Rozhraní pro Load Balancer.



Obrázek 8.1: Třídní diagram poskytovatele služeb.

```
int main(int argc, char *argv[])
{
    /**
    Je nutne zadat cestu ke konfiguracnimu souboru, pokud se rozhodneme vytvaret Load Ba
    **/
    std::string configFile = "path/to/configfile";

    LoadBalancer balancer(configFile);

    //Load Balancer je spusten metodou Run
    balancer.run();
    return 0;
}
```

Výpis 8.3: Příklad vytvoření a spuštění služby Load Balancer pomocí knihovny Cloud3D



Obrázek 8.2: Diagram tříd popisující Load Balancer.

Metoda	Popis
performOperation	Pokusí se připojit k poskytovateli služeb a vykonat požadovanou operaci, jejímž vstupem je vektor modelů a výstupem je vektor modelů, popř. vektor double hodnot
loadMeshFromFile	Načte uložený soubor s modelem do struktury Mesh
saveMeshToFile	Uloží strukturu mesh do souboru

Tabulka 8.3: Rozhraní pro Klienta.

Název	Popis	Povinný
balancerAddress	IP adresa Load Balanceru	Ano
version	Minimální verze služby, se kterou chce klient pracovat	Ne
certFilename	Cesta k souboru s osobním certifikátem klienta	Ne
keyFilename	Cesta k souboru s privátním klíčem klienta	Ne
rootFilename	Cesta k souboru s CA certifikátem serveru	Ne

Tabulka 8.4: Seznam argumentů pro spuštění klienta.

Definice klienta

Rozhraní klienta je popsáno ve třídě CloudClient. Klient pro své správné fungování potřebuje znát ip adresu a port na kterém běží služba Load Balancer. Dále je potřeba zadat minimální verzi v jaké by měl poskytovatel služeb běžet. A nakonec seznam všech služeb, které by měl poskytovatel služeb nabízet. Seznam služeb je potřeba, aby se klient nemusel neustále připojovat na Load Balancer s každým dotazem, ale mohl následně už komunikovat pouze s poskytovatelem. V případě, že se uživatel rozhodne komunikovat se servery v šifrované podobě, je potřeba dále nadefinovat cestu k osobnímu certifikátu klienta, jeho privátnímu klíči a CA certifikátu serverů. Klient následně pouze volá požadované procedury konkrétním názvem, jakým jsou nadefinovány v poskytovateli služeb.

Kompletní seznam veřejného rozhraní dostupného klientu lze nalézt v tabulce 8.3. Příklad jednoduchého vytvoření klienta lze nalézt ve výpisu 8.4. Diagram tříd reprezentující strukturu klienta lze nalézt na obrázku 8.3.

8.2 Načítání argumentů

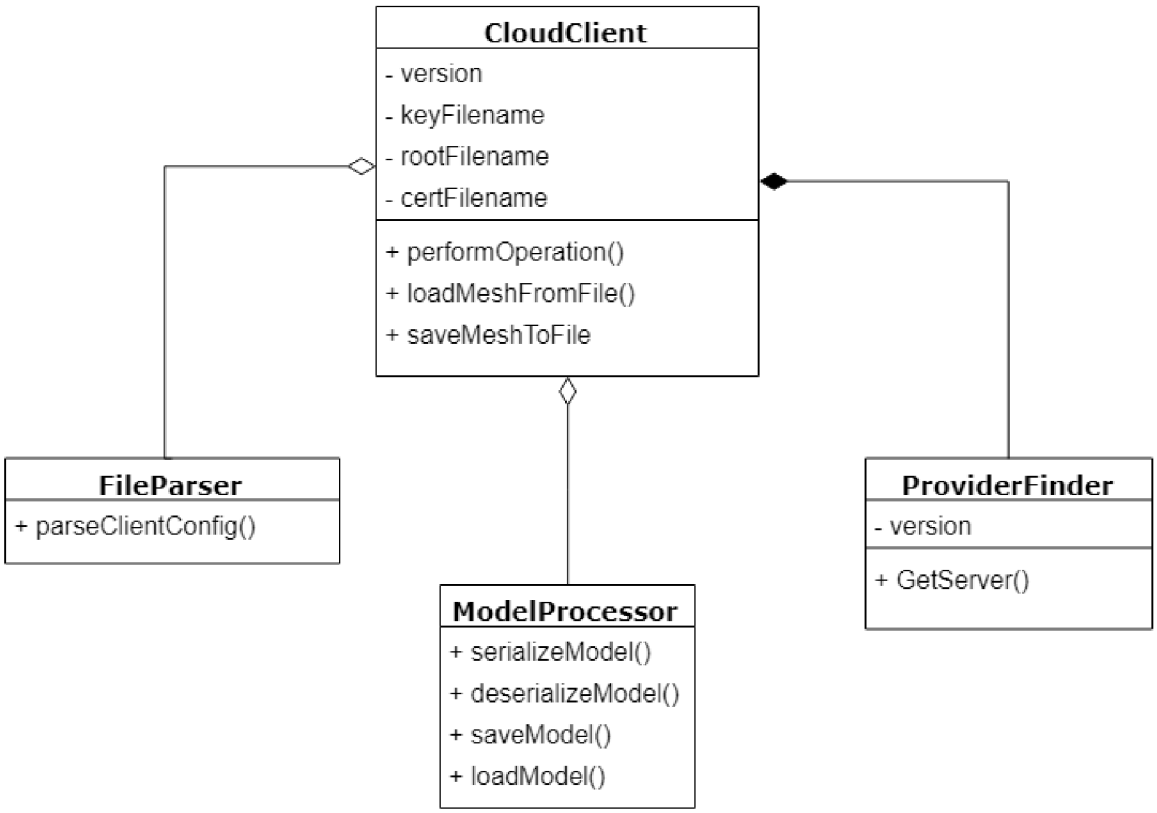
Protože v dnešní době jsou všechny služby nasazovány na servery automaticky pomocí skriptů, bylo potřeba jednoduchou modifikovatelnost. Takže kromě možností zadat všechny potřebné informace ručně, má uživatel možnost vytvářet konfigurační soubory, které mu umožní celý proces zautomatizovat. Pro potřeby knihovny Cloud3D byl vytvořen nový formát pro definování konfiguračních souborů. Ten vypadá následovně:

název=hodnota

Jako oddělovač slouží nový řádek. Každý ze 3 subjektů má vlastní argumenty, které jsou povinné a které jsou volitelné. Seznam všech argumentů pro klientskou část lze nalézt v tabulce 8.4, pro poskytovatele v tabulce 8.5 a pro Load Balancer v tabulce 8.6.

```
\\Priklad vytvoreni klienta a provedeni operace Vertex Decimation v cloudu
int main(int argc, char *argv[])
{
    /**
    Nejprve je nutne zadat seznam operaci,
    ktere od poskytovatele pozadujeme
    **/
    std::vector<std::string> vecOfOperation{"VertexDecimation", "MeshSmooth"};
    std::string inputFile = "/path/to/model.obj";
    std::string outputFile = "/path/to/new_model.obj";

    /**
    Je nutne zadat cestu ke konfiguracnimu souboru, pokud se rozhodneme
    vytvaret klienta touto cestou
    **/
    std::string configFile = "path/to/configfile";
    std::vector<CloudClient::CloudMesh> oldMesh;

    //V dobe sveho vzniku si klient zazada Load Balancer o adresu serveru
    CloudClient client(configFile, vecOfOperation);

    CloudClient::CloudMesh loadedMesh;
    //Klient sam umoznuje nacitani a ukladani souboru v potrebnem formatu
    client.loadMeshFromFile(inputFile, loadedMesh);
    oldMesh.push_back(loadedMesh);
    std::vector<CloudClient::CloudMesh> newMesh;

    //Volani vzdalene procedury "VertexDecimation"na serveru.
    client.performModelsToModelsOperation("VertexDecimation",
                                          oldMesh,
                                          newMesh);

    client.saveMeshToFile(outputFile, newMesh.front());

    return 0;
}
```

Výpis 8.4: Příklad vytvoření klienta s využitím knihovny Cloud3D



Obrázek 8.3: Diagram tříd znázorňující stavbu klienta.

Název	Popis	Povinný
balancerAddress	IP adresa Load Balanceru	Ano
providerAddress	IP adresa poskytovatele služeb	Ano
heartBeat	Udává, s jakou periodou se má zasílat informace pro Load Balancer	Ne
version	Verze poskytovatele služeb	Ne
certFilename	Cesta k souboru s osobním certifikátem poskytovatele služeb	Ne
keyFilename	Cesta k souboru s privátním klíčem klienta	Ne
rootFilename	Cesta k souboru s CA certifikátem klienta	Ne

Tabulka 8.5: Seznam argumentů pro spuštění poskytovatele služeb.

Název	Popis	Povinný
balancerAddress	IP adresa Load Balanceru	Ano
heartBeat	Udává, jak často má přijít informace od poskytovatele	Ne
certFilename	Cesta k souboru s osobním certifikátem Load Balanceru	Ne
keyFilename	Cesta k souboru s privátním klíčem klienta	Ne
rootFilename	Cesta k souboru s CA certifikátem klienta	Ne

Tabulka 8.6: Seznam argumentů pro spuštění Load Balanceru.

8.3 Ověřování verze

Protože knihovna Cloud3D umožňuje tvorbu aplikací umožňujících vzdálené volání procedur, mohou být klienti a poskytovatelé služeb upravováni nezávisle na sobě bez toho, aby se pro ně cokoliv změnilo v komunikačním procesu. Bohužel to přináší i nechtěnou vlastnost, kdy poskytovatel může nabízet zastaralou verzi některé služby. Z tohoto důvodu byla implementována možnost verzování.

Poskytovatel služeb při svém úvodním kontaktu s Load Balancerem udává, na jaké konkrétní verzi pracuje. Load Balancer si tuto informaci uchovává v paměti a porovnává ji s požadavky klienta, které na něj přichází. Klient zasílá minimální číslo verze, se kterou chce pracovat. Pokud je verze na které poskytovatel služeb běží nižší než verze o kterou si žádá klient, není klientovi tento poskytovatel služeb nabídnut. A to ani v případě, kdyby byl jediným poskytovatelem, který splňuje všechny ostatní požadavky.

Verze služeb lze zadat dvěma způsoby. První možností je přímo v konstruktoru poskytovatele služeb, popř. klienta. Druhá varianta umožňuje zadání verze pomocí konfiguračního souboru.

Pokud uživatel tuto funkcionalitu nepotřebuje, je možné ji naprosto vynechat. V takovém případě se všechny zúčastněné strany chovají, jako kdyby pracovaly všechny ve stejné verzi.

Kapitola 9

Experimenty s knihovnou Cloud3D

Tato kapitola popisuje experimenty, jenž byly provedeny s knihovnou Cloud3D. Cílem těchto experimentů bylo zjistit, zda knihovna, vytvořena v rámci této práce, opravdu funguje zamyšleným způsobem. Postupně byly provedeny experimenty v oblasti efektivity, spolehlivosti a škálovatelnosti.

Všechny experimenty byly provedeny za pomoci služby Google Cloud, která umožňuje správu virtuálních strojů na vzdálených serverech. Aby byly výsledky experimentů co nejpřesnější, byly použity dvě různé konfigurace. První konfigurace slouží pro simulaci klientského zařízení, druhá slouží pro simulaci poskytovatele služeb a Load Balancer.

Všechny konfigurace pracují s operačním systémem Ubuntu verze 18.04 LTS. Klientská část je tvořena jedno jádrovým sdíleným virtuálním procesorem s paměti 0,6 GB s 10GB pevným diskem. Konfigurace obsluhující poskytovatele služeb a Load Balancer jsou tvořeny čtyř jádrovým virtuálním procesorem běžícím na platformě Intel Skylake s 15 GB paměti. Dále má k dispozici grafický procesor NVIDIA Tesla P100 a SSD pevný disk s 20 GB paměti. Tyto konfigurace si nedávají za úkol simulovat současný stav klientských a serverových zařízení, jakožto spíše simulovat poměr mezi jejich výkony.

Aby experimenty simulovaly realitu co nejvěrněji, bylo každé zařízení umístěno na jiný server. Poskytovatele služeb se nachází v těchto oblastech:

- Jižní Karolína - Download: 394.08 Mbit/s, Upload: 55.26 Mbit/s
- Oregon - Download: 737.88 Mbit/s, Upload: 87.10 Mbit/s
- Montreál - Download: 336.99 Mbit/s, Upload: 42.18 Mbit/s

Load Balancer se nachází v oblasti:

- Iowa - Download: 389.22 Mbit/s, Upload: 114.27 Mbit/s

Klienti se vyskytují v následujících oblastech:

- Iowa - Download: 389.22 Mbit/s, Upload: 114.27 Mbit/s
- Nizozemí - Download: 385.99 Mbit/s, Upload: 26.15 Mbit/s
- Hong Kong - Download: 92.67 Mbit/s, Upload: 24.96 Mbit/s

9.1 Experiment 1 - Měření režie při použití knihovny Cloud3D

Cílem prvního experimentu bylo zjistit, zda využití knihovny Cloud3D bude v praxi efektivnější oproti řešením bez využití této knihovny. Experiment proběhl na cvičné operaci, která prováděla vyhlazování pomocí Laplaceho vyhlazovací metody. Tato metoda byla zvolena z důvodu své proveditelnosti na konfiguraci s nižším výkonem, avšak rychlost provedení je dostatečně rozdílná na všech typech konfigurací.

Experiment probíhal s třemi 3D modely různých velikostí. Jednalo se o tři modely aut o velikosti 108 kB s počtem trojúhelníků 1 172, 5,8 MB s počtem trojúhelníků 34 126 a 14,1 MB s počtem trojúhelníků 125 407. Všechny modely byly uloženy ve formátu OBJ. Jejich náhled lze vidět na obrázku 9.1.

Obrázek 9.1: Náhled modelů použitých v experimentu. Model o velikosti 108kB s počtem trojúhelníků 1 172 (vlevo), model o velikosti 5.8MB s počtem trojúhelníků 34 126 (vprostřed) a model o velikosti 14.1.MB s počtem trojúhelníků 125 407 (vpravo).

Nejprve byly získány referenční hodnoty při provedení dané operace bez použití knihovny Cloud3D. Měření proběhlo na konfiguracích pro klienta a pro poskytovatele. Výsledky naměřených hodnot lze nalézt v tabulce 9.1. V tabulce 9.1 jsou uvedené pouze časy, po které trvá samotná operace vyhlazení bez ostatní režie, např. načtení modelů do paměti programů.

Z tabulky lze vyčíst, že daná operace při práci s nejmenším objektem trvá přibližně stejně dlouho. To lze tvrdit i o středně velkém objektu, kde rozdíl mezi oběma konfiguracemi už je patrnější, ovšem ne zcela zásadní. Největšího rozdílu je dosaženo při práci s nejsložitějším a zároveň největším modelem. Zde je čas potřebný k vykonání požadované operace na straně klienta dvojnásobný oproti času, který k tomu potřebuje konfigurace poskytovatele služeb.

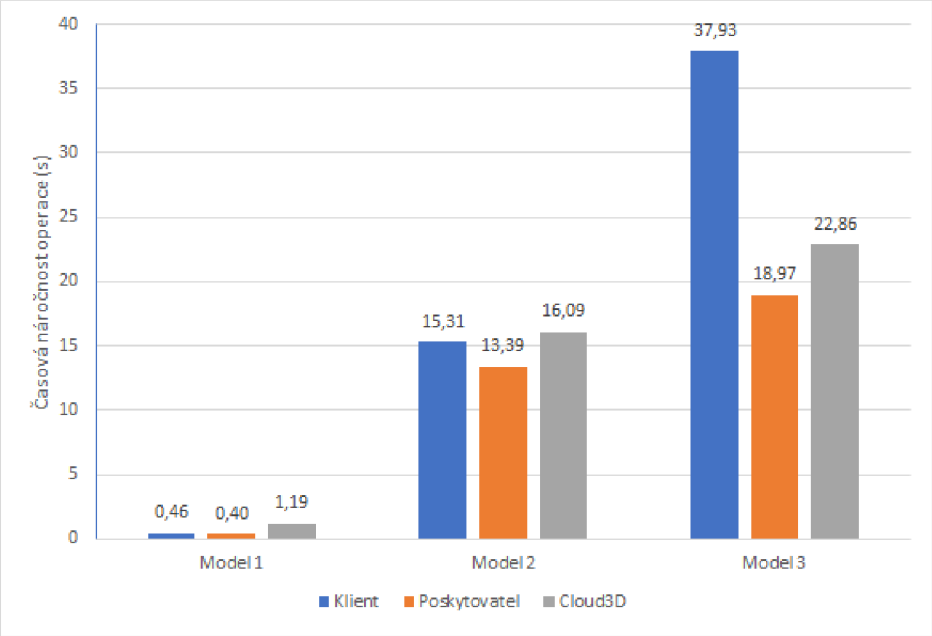
Velikost souboru	Klient	Poskytovatel
108kB	0.459s	0.401s
5.8MB	15.311s	13.386s
14.1MB	37.926s	18.967s

Tabulka 9.1: Časová náročnost operace vyhlazení na jednotlivých konfigurací bez použití systému Cloud3D.

Následně proběhlo měření s využitím všech prvků knihovny Cloud3D, tzn. že byl použit Load Balancer, klientská část aplikace a poskytovatele služeb. Výsledky tohoto experimentu lze nalézt v tabulce 9.2. V tabulce je uvedeno pět různých informací: jak dlouho trvalo připojit se na Load Balancer a zajistit adresu poskytovatele služeb, serializace a deserializace modelu, odeslání modelu z klienta na poskytovatele služeb a zpět, vykonaní požadované operace a celkový čas. Ve výsledných časech je opět zanedbáno načítání do paměti a ukládání výsledného modelu, neboť to klient provede vždy stejně, ať je operace prováděna vzdáleně nebo lokálně.

	Model 1	Model 2	Model 3
Velikost modelu	108kb	5.8MB	14.1MB
Připojení na Load Balancer a získání adresy	0.219s	0.207s	0.212s
Serializace a Deserializace	0.008s	0.263s	0.669s
Odeslání a příjem modelu	0.681s	2.35s	3.198s
Operace s modelem	0.211s	13.265s	18.782s
Celkový čas	1.191s	16.085s	22.861s

Tabulka 9.2: Výsledky časů naměřených při použití knihovny Cloud3D.



Obrázek 9.2: Graf srovnávající délku výpočtu na jednotlivých konfiguracích se systémem Cloud3D.

Jak z naměřených hodnot vyplývá, použití knihovny Cloud3D není vhodné pro použití na malých souborech přibližně do 8 MB. V takovém případě režie na zakódování souboru a jeho přenesení je větší než průběh samotné operace na pomalejším zařízení. Jak vyplývá z tabulky 9.2, čas na získání adresy poskytovatele služeb trvá konstantní čas. Nároky na serializaci a deserializaci rostou poměrově rychleji než čas na přenesení modelu a zpracování operace nad ním, avšak čas na serializaci je zanedbatelný v poměru s trváním operace. Z toho plyne, že knihovna Cloud3D je vhodná především pro práci s modely většími než 8 MB, popř. pro výpočetně náročnější operace, což bylo také jejím záměrem. Přímé porovnání systému Cloud3D proti lokálním výpočtům lze nalézt v grafu 9.2.

9.2 Experiment 2 - Ověřování robustnosti knihovny

Druhý experiment měl za úkol zjistit, jak se zachová systém v případě výpadku jedné nebo více jeho částí. Protože knihovna Cloud3D by měla být používána ve složitější síťových

strukturách, bylo nezbytné zjistit, zda v případě výpadku jedné nebo více konkrétních služeb nedojde k výpadku celé infrastruktury. Hlavní důraz byl kladen především na poskytovatele služeb a Load Balancer. Klientská část aplikace pro tento experiment nebyla zásadní.

Bylo sestaveno několik krizových scénářů, které byly postupně vyzkoušeny:

A Výpadek poskytovatele služeb

B Výpadek Load Balanceru

C Výpadek poskytovatele služeb a Load Balanceru

V případě scénáře A, tedy poskytovatel služeb se ukončí, popř. se přeruší internetové připojení, dojde k jeho vymazání z nabídky dostupných poskytovatelů, jenž Load Balancer nabízí. Tento jev nastane po výchozí jedné vteřině nebo po uplynutí doby, kterou si sám uživatel nastaví. Klienti, kterým se nechtěně podaří získat adresu daného poskytovatele, se na něj nepřipojí nebo se od něj odpojí a nevykonají požadovanou operaci. V takovém případě je nutné vytvořit novou instanci klienta a získat novou adresu poskytovatele služeb.

Poskytovatel služeb se po svém opětovném zprovoznění připojí na Load Balancer normálním způsobem. V případě scénáře B, tedy když není možné se připojit na Load Balancer, může dojít k několika následujícím událostem. Klient, jenž se pokouší získat adresu poskytovatele služeb bude neúspěšný a bude se o to muset pokusit znovu později. Klient, který adresu poskytovatele služeb již má, s ním udržuje neustále spojení, a tedy není výpadkem nijak postižen.

Nově vytvořený poskytovatel služeb se pokusí vytvořit záznam v Load Balanceru a bude neúspěšný. I přesto začne přijímat příchozí požadavky od klientů a periodicky se bude snažit zahájit komunikaci s Load Balancerem. Pokud již existující poskytovatel zjistí, že nemůže zaslat informaci o svém stavu na Load Balancer, začne se periodicky pokoušet znovu vytvořit spojení. Poskytovatel služeb ovšem v takovém případě nepřeruší příjem nových požadavků, ani nezruší ty stávající.

V případě scénáře C, tedy výpadku všech částí systému, budou všichni stávající klienti odpojeni a jejich požadavky skončí neúspěchem. Noví klienti nebudou moci získat žádnou adresu poskytovatele. V obou případech bude služba nefunkční a bude potřebovat opětovné spuštění všech komponent. V tomto experimentu bylo zjištěno, že služba nabízená knihovnou Cloud3D je připravena na výpadky jednotlivých částí systému v omezené míře. Load Balancer je stavěný na to, že jednotliví poskytovatelé se mohou v libovolné míře připojovat a odpojovat a nemělo by to způsobovat jakékoliv potíže. Pokud by se stalo, že by byl Load Balancer kompletně nedostupný, byly by zachovány alespoň stávající klienti. V případě výpadku celého systému bude služba kompletně nedostupná.

9.3 Experiment 3 - Škálovatelnost systému

Třetí experiment si dál za cíl zjistit škálovatelnost systému Cloud3D. Při experimentu byl postupně testován systém s jedním, následně pěti a nakonec dvaceti poskytovateli služeb. Během experimentu bylo měřeno, kolik operací je schopen vykonat celý systém za minutu. Do systému bylo zasíláno 200 požadavku za minutu z různých klientů. Testovací operace by Laplaceho vyhlazování s modelem o největší velikosti 14,1 MB a počtem trojúhelníků 125 407 z prvního experimentu (viz 9.1).

V tabulce 9.3 lze vidět naměřené hodnoty. Všechny hodnoty byly měřeny v časovém úseku deseti minut a následně zprůměrovány. Z hodnot vyplývá, že s počtem klientů roste

i maximální počet obsloužených operací a klientů. Avšak s počtem serverů, neroste přímo úměrně také počet vykonaných operací.

Pokud bude počet poskytovatelů pětinásobný, tak výkonost celého systému vzroste pouze dvojnásobně. To je bohužel zapříčiněno zvýšenou režií na fungování celého systému.

Počet poskytovatelů	**Počet operací za minutu**
1	22.3
5	48.6
20	88.7

Tabulka 9.3: Průměrný počet provedených operací v systému Cloud3D při zaslání 200 požadavku za minutu.

Z tohoto experimentu plyne, že systém Cloud3D je velice vhodný ve chvílích, kdy počet klientů a jejich požadavků převyšuje výpočetní výkonost jednoho poskytovatele. Lze si taky povšimnout, že problémy spojené se škálováním velkých cloudových aplikací se vyskytují i v tomto systému.

Kapitola 10

Závěr

V této práci byla popsána problematika zobrazování a zpracování trojrozměrných modelů v počítačové grafice. Byly popsány základy tvorby těchto objektů a nejběžnější postupy při jejich zobrazování. Dále byly nastíněny základní algoritmy pro transformaci 3D objektů a jejich zjednodušování.

Na základě těchto znalostí byla navržena a implementována knihovna Cloud3D v programovacím jazyce C++. Výsledná knihovna nabízí možnost jednoduše vytvářet klient-server aplikace pro práci s 3D modely. Cílem takových aplikací je umožnit zpracování časově náročných operací v cloudu, bez výpočetního výkonu klienta. Klientská část knihovny je schopna rychle a efektivně odesílat a přijímat grafické modely. Část pro poskytovatelé služeb nabízí jednoduché rozhraní k vytváření a zpracování složitých časově náročných operací s trojrozměrnými modely. Mimo jiné knihovna Cloud3D také implementuje službu Load Balancer, která se stará o rozložení zátěže mezi jednotlivé poskytovatele. Knihovna Cloud3D je distribuována pod licencí MIT a volně dostupná pod odkazem[1].

S knihovnou byly provedeny tři experimenty mající za cíl ověřit použitelnost knihovny v praxi a funkčnost implementace. Z dosažených výsledků bylo zjištěno, že knihovna Cloud3D splňuje účel za jakým byla vytvořena. Experimentálně bylo ověřeno, že knihovna je dobře použitelná pro aplikace, kdy počet klientů a jejich požadavků převyšuje výpočetní možnosti jednoho serveru. Dále bylo zjištěno, že knihovna nabízí urychlení při práci se složitějšími modely a operacemi s nelineární časovou složitostí. Se snižujícím se výkonem klientů stoupá výsledná efektivita celého systému.

I přesto, že knihovna Cloud3D splňuje vše, k čemu byla vytvořena, existuje řada možností, kam ji stále vylepšovat. Autor má v plánu přinést plnou podporu pro operace modelů s texturami a vytvářet další rozšíření pro další typy operací.

[1] `https://github.com/Tasarak/3DCloud`

Literatura

[1] Computer Graphics Group, R. A.: *OpenMesh: The Halfedge Data Structure*. [Online; navštíveno 15.02.2019].
URL `https://www.openmesh.org/media/Documentations/OpenMesh-6.3-Documentation/a00010.html`

[2] Cortés, A.; Ripoll, A.; Cedó, F.; aj.: An asynchronous and iterative load balancing algorithm for discrete load model. *Journal of Parallel and Distributed Computing*, ročník 62, č. 12, 2002: s. 1729–1746, ISSN 0743-7315.

[3] Couturier, R.; Giersch, A.; Hakem, M.: Best effort strategy and virtual load for asynchronous iterative load balancing. *Journal of Computational Science*, ročník 26, 2018: s. 118–127, ISSN 1877-7503.

[4] Foley, D. J.: *Computer Graphics : principles and practice*. Boston: Addison-Wesley, druhé vydání, 1996, ISBN 0-201-84840-6.

[5] Foundation, A. S.: *Apache Kafka*. [Online; navštíveno 15.02.2019].
URL `https://kafka.apache.org/intro`

[6] Garland, M.; Heckbert, P.: Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings Visualization '98 (Cat. No.98CB36276)*, IEEE, 1998, ISBN 081869176X, s. 263–269.

[7] Geelnard, M.: *OpenCTM - About*. [Online; navštíveno 15.02.2019].
URL `http://openctm.sourceforge.net/?page=about`

[8] Google: *gRPC*. [Online; navštíveno 15.02.2019].
URL `https://grpc.io/`

[9] Google: *Guides - gRPC*. [Online; navštíveno 15.02.2019].
URL `https://grpc.io/docs/guides/`

[10] Gregory, J.: *Game engine architecture*. Natick: A K Peters, 2009, ISBN 978-1-56881-413-1.

[11] iMatix: *Distributed Messaging - zeromq*. [Online; navštíveno 15.02.2019].
URL `http://zeromq.org/`

[12] Žára Jiří: *Moderní počítačová grafika*. Brno: Computer Press, vyd 1. vydání, 2004, ISBN 80-251-0454-0.

[13] Lengyel, E.: *Mathematics for 3D : game programming and computer graphics*. Massachusetts: Charles River Media, vyd. 1 vydání, 2004, ISBN 1-58450-277-0.

[14] Luebke, D.: *Level of detail for 3D graphics*. Boston: Morgan Kaufmann, vyd. 1. vydání, 2003, ISBN 1-55860-838-9.

[15] Rossignac, J.; Borrel, P.: Multi-resolution 3D approximations for rendering complex scenes. Technická zpráva, Yorktown Heights, NY 10598, Feb. 1992, iBM Research Report RC 17697. Also appeared in *Modeling in Computer Graphics*, Springer, 1993.

[16] Schroeder, W. J.; Zarge, J. A.; Lorensen, W. E.: Decimation of triangle meshes. *ACM SIGGRAPH Computer Graphics*, ročník 26, č. 2, 1992: s. 65–70, ISSN 00978930.

[17] Triumph: *Blend4Web | Unleashing the Power of 3D Internet*. [Online; navštíveno 15.02.2019].
URL `https://www.blend4web.com`

[18] University, R.-A.: *OpenMesh*. [Online; navštíveno 15.02.2019].
URL `https://www.openmesh.org/`

[19] Watt, A.: *Advanced animation and rendering techniques : theory and practice*. New York: Addison-Wesley, vyd. 1. vydání, 1992, ISBN 0-201-54412-1.

Příloha A

Instalace

V této kapitole bude popsána instalace knihovny Cloud3D a včetně všech jejich prerekvizit.

Zdrojové kódy této práce jsou dostupné z repozitáře verzovacího systému GitHub. Zdrojové kódy lze stáhnout přímo z webové stránky nebo pomocí terminálového příkazu:

$$git\ clone\ https://github.com/Tasarak/3DCloud$$

Před samotným překladem knihovny Cloud3D je potřeba nainstalovat všechny prerekvizity. Ty jsou následující:

- Protocol Buffer 3
- gRPC
- OpenMesh
- Log4Cplus

Dalším předpokladem je nainstalovaný nástroj CMake a jakýkoliv překladač jazyku C++.

Po nainstalování všech prerekvizit je možné přeložit knihovnu pomocí příkazů

$$cmake\ CMakeList.txt$$

$$make$$

Překladem vzniknou tři dynamické knihovny

- CloudClientlib
- ServiceProviderlib
- LoadBalancerlib

a několik samostatně spustitelných programů, jejichž zdrojové kódy lze nalézt ve složce *Samples*.

Příloha B

Obsah CD

- xkleme11.pdf - elektronická verze textu této práce
- xkleme11.zip - zdrojové soubory textu této práce
- sources.zip - zdrojové soubory implementované knihovny Cloud3D