UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE
IN INFORMACIJSKE TEHNOLOGIJE
KOPER

Izvedba vmesnika za porazdeljeno in vzporedno upodabljanje 3D modelov

(Development of Plug-in for Distributed and Parallel Rendering of 3D Models)

ZAKLJUČNA NALOGA

Ime in priimek: Domen Petrič

Študijski program: Računalništvo in informatika 1. stopnja

Mentor: doc. dr. Janez Žibert

Koper, avgust 2012

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE
IN INFORMACIJSKE TEHNOLOGIJE
KOPER

Zahvala

Zahvaljujem se staršem in prijateljem za podporo. Posebna zahvala pa je namenjena mentorju doc. dr. Janezu Žibertu za pomoč in nasvete med pisanjem zaključne naloge.

Povzetek

V zaključni nalogi smo se ukvarjali z upodabljanjem tehničnih načrtov 3D modelov. Pregledali smo najbolj pogosta programska orodja za tehnično risanje ter postopke za upodabljanje, opisali smo postopke za izvajanje porazdeljenega upodabljanja med več procesorji ter izdelali vmesnik za programsko orodje AutoCAD, ki nam omogoča da lahko tehnične načrte 3D modelov glede na trenutni pogled sprotno upodabljamo na strežniku ter nato upodobljeno sliko pošiljamo nazaj k uporabniku. Vmesnik je zasnovan na principu odjemalec-strežnik, kar nam omogoča, da lahko izvajamo proces upodabljanja porazdeljeno, če je tako zasnovana arhitektura strežnika in sam sistem za upodabljanje tehničnih načrtov. Tako izvedeno upodabljanje slik smo tudi testirali na porazdeljenem sistemu in ugotovili znatno izboljšanje hitrosti delovanja upodabljanja na različno zahtevnih testnih slikah.

Ključne besede

Upodabljanje, vtičnik za upodabljanje, AutoCAD, porazdeljeno upodabljanje, postopki upodabljanja.

Summary

The final project assignment focuses on rendering technical designs for 3D models. It includes a review of the most common drafting software tools and rendering procedures, a description of the procedures for executing distributed rendering among several processors and an elaboration of the AutoCAD plug-in that enables the real-time rendering of technical designs as 3D models on the server-side on the basis of the current view. The plug-in is designed using the client-server principle, which enables the distributed execution of the rendering process, if the server architecture and the system for rendering technical designs are designed in such a manner. Image rendering executed in this way was also tested on a parallel system and a considerable increase in the rendering speed on test images of various levels of complexity was determined.

Keywords

Rendering, render plug-in, AutoCAD, parallel rendering, rendering techniques

Kazalo

Kazalo slik

Kazalo tabel

Kazalo izvornih kod

1. Uvod

Upodabljanje (*ang. rendering*) predstavlja proces preslikovanja tehničnih in drugih načrtov, ki so izvedeni v različnih dimenzijah in na različne načine, v očesu prijazno sliko. Običajno se upodabljanje uporablja v primeru načrtov 3D modelov, kjer se je tudi najprej razvilo, vendar se je kasneje začelo uporabljati tudi na ostalih področjih.

1.1. Opredelitev področja in opis problema

3D upodabljanje slik je problem, ki se uporablja na številnih področjih človekovega življenja. V zaključni nalogi se bomo usmerili predvsem problemu upodabljanja slik iz programov za arhitekturo in strojništvo, področje animacij ter filma pa bomo pustili ob strani. Tako npr. arhitekt, ki uporablja nek programski paket za arhitekturno risanje načrtuje model in ko je model narejen, bi rad ta model predstavil. Model je najlažje predstaviti, če ga lahko pokažeš čim bolj realistično, saj si večina ljudi iz narisanega načrta ne zna predstavljati dejanskega objekta. Arhitektu tako lahko preostane fizična izdelava prototipa, ki je v mnogih primerih predraga, oziroma jo je nemogoče narediti. Pri izdelavi pomanjšanega modela pa lahko naletimo tudi na problem natančnosti izdelave ter vizualizacije notranjosti modela. Tretja možnost, ki se nam ponuja in je cenejša od prejšnjih dveh, pa je upodabljanje načrtov z računalnikom. Pomanjkljivost te metode pa je, da se predstavljenega modela ne da fizično dotikati, vendar pa lahko ta metoda prikaže tako notranjost kot zunanjost modelov ter je cenovno ugodnejša. Problem upodabljanja je v času potrebnem za izdelavo slik, saj je upodabljanje računsko zelo zahteven proces, ki za posamezno sliko zaposli računalnike tudi za več ur ali celo dni. V pričujoči nalogi smo čas upodabljanja poskušali z vzporednimi in porazdeljenimi metodami zmanjšati. Cilj naloge je bil zgraditi vmesnik, ki bi prenesel upodabljanje na ustrezen računalnik ali računalniško arihtekturo, kjer bi bilo mogoče izvajati upodabljanje vzporedno in/ali porazdeljeno, da bi tako pohitrili proces upodabljanja in nudil rezultate hitreje.

1.2. Namen in cilji naloge

Namen zaključne naloge je bil spoznati programske pakete za tehnično risanje, preučiti možnosti teh programskih orodij za razširitev dodajanja vtičnikov, ki nam bi zagotovili samostojno upodabljanje slik iz načrtov 3D modelov. S tem bi lahko proces upodabljanja preselil na računalniške arhitekture, ki nam bi zagotavljale paralelizacijo procesa upodabljanja. V ta namen smo preučili tudi možne izvedbe algoritmov za upodabljanje ter njihovo razširitev za vzporedno delovanje. Končni cilj naloge je bil narediti vtičnik, ki bi skrajšal čas izvedbe upodabljanja, tako da načrt uporabnika pošlje na zmogljivejši računalnik, ki izdela sliko ter jo pošlje nazaj uporabniku v krajšem času, kot bi ga za to potreboval uporabnik na svojem računalniku.

1.3. Pregled vsebine zaključne naloge

V nalogi bomo pregledali potek upodabljanja. Pogledali bomo najbolj uporabljene programske pakete za tehnično in arhitekturno risanje. Nekateri izmed njih že vsebujejo algoritme za upodabljanje, drugi ne. Nekateri izmed algoritmov v samih programskih paketih so pri uporabnikih manj zaželeni, zato mnogokrat uporabniki prenašajo načrte na druge algoritme in jih tam upodabljajo. V ta namen bomo v zaključni nalogi pregledali algoritme za upodabljanje slik, tako samostojne, kot tiste vključene v programske pakete za tehnično risanje.

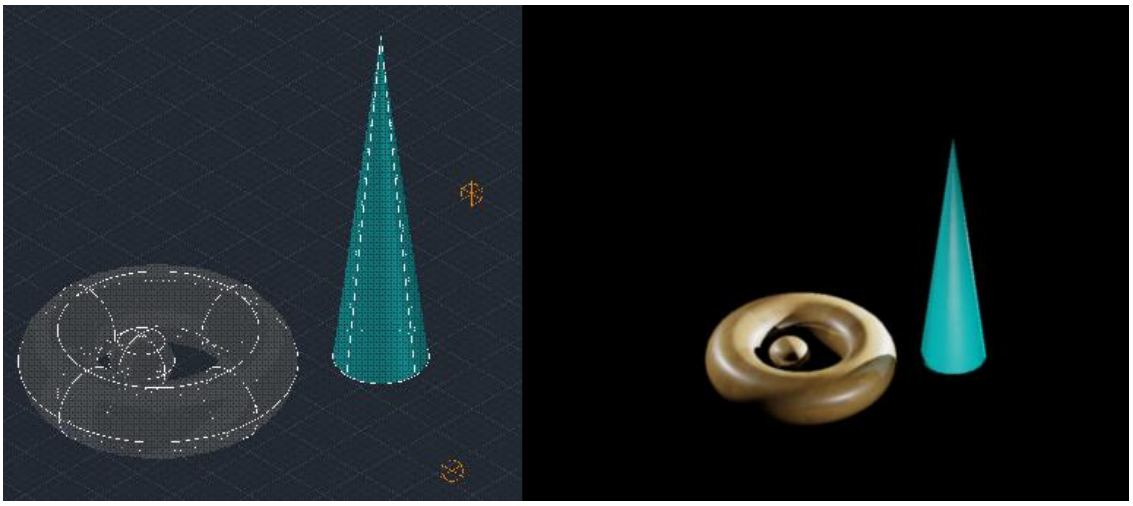
Predstavili bomo tudi izdelavo vtičnika za enega izmed programskih paketov za tehnično risanje, ki smo ga testirali z različno zahtevnimi tehničnimi slikami na vzporedni računalniški arhitekturi. Izvedena je bila tudi primerjava z obstoječim algoritmom upodabljanja, ki se je izvajal znotraj programskega paketa na računalniški arhitekturi, ki ni omogočala vzporednega izvajanja upodabljanja.

2. Upodabljanje slik

2.1.Splošno o upodabljanju slik

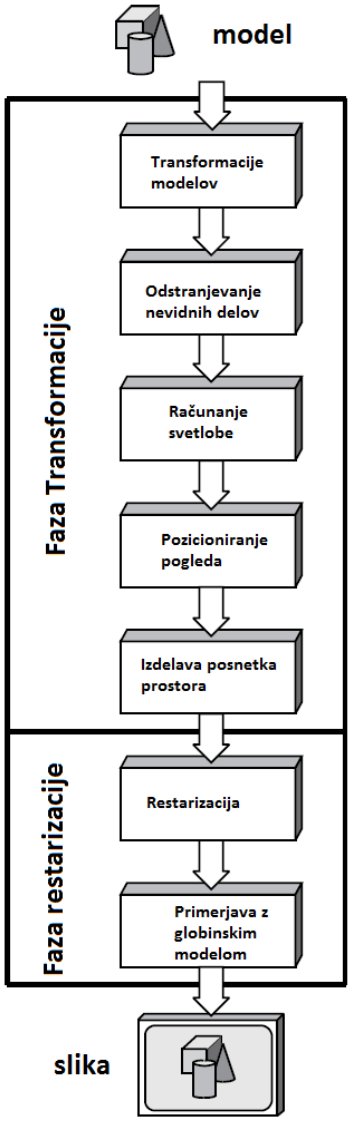
Upodabljanje je proces izdelave slike iz žičnega modela. Samo upodabljanje si lahko predstavljamo kot slikanje s fotoaparatom na gledališki predstavi. Rezultat upodabljanja je slika scene, na katero vplivajo predmeti postavljeni v ta navidezni prostor, material predmetov, smer pogleda, svetloba in posledično tudi sence. Slika 1 prikazuje primer začetne scene ter končne upodobitve. Slike so izdelane s pomočjo algoritmov za upodabljanje, ki uporabljajo računske moči grafičnih procesorskih enot (GPE). Grafični procesorji ne zmorejo vedno vsega dela, včasih pa računalniki uporabnikov le teh sploh nimajo, zato pri računanju pomaga centralna procesorska enota (CPE) ali več njih.

Upodabljanje se uporablja v zelo širokem spektru področij. Največ se uporablja v arhitekturi, strojništvu, video igrah, simulatorjih, filmih ter televizijskih efektih. Zaradi padanja cene izdelave animacij pa se vedno bolj pogosto uporablja tudi na področju trženja. Veliko povpraševanje ter različna področja uporabe sta razvila in še razvijata najrazličnejše algoritme za upodabljanje. Nekateri so vsebovani v isto namenskih programih, drugi obstajajo sami zase. Nekatere izmed njih razvijajo večja podjetja, običajno so to tisti algoritmi, ki so vsebovani v večjih programskih paketih z načrtovanjem, drugi so odprtokodni. Zaradi zahtevnosti postopka upodabljanja delimo algoritme za upodabljanje na tiste, ki delajo v realnem času ter druge, za katere je potrebna pred-upodobitev za kasnejši prikaz.



Slika 1: Slika iz načrtovalskega programa levo in upodobljena slika desno

Slika 2 prikazuje postopek od modela do slike. Najprej postopek umesti objekte v prostor s koordinatami. Nato postopek odstrani vse površine, ki na končni sliki ne bodo vidne. V nadaljevanju se izračuna svetloba po postopkih, ki bodo predstavljeni v nadaljevanju. V koordinatni sistem se doda še točko pogleda ter se izdela posnetek scene. Sceno se nato rasterizira in dobljeno sliko primerja z globinsko sliko modela.

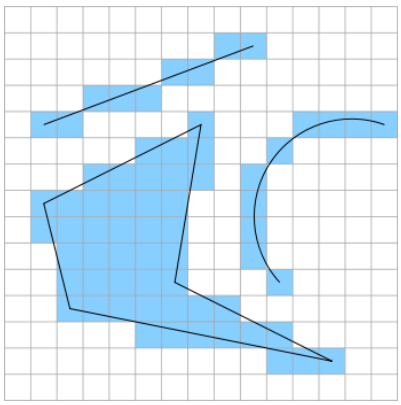


Slika 2: Shema poenostavljenega postopka upodabljanja [9].

Sledenje vsakemu žarku svetlobe, njegovemu odboju v predmetih in podobno, je zaradi časovne kompleksnosti skoraj nemogoče izvesti celo na današnjih računalnikih. Zaradi tega problema so se razvile štiri osnovne metode za računanje poti svetlobe. Te so rasterizacija (*ang. rasterization*), žarkovno zlivanje (*ang. ray casting*), žarkovno sledenje (*ang. ray tracing*) in razpršena osvetljenost (*ang. radiosity*).

2.1.1. *Rasterizacija*

Načrti predstavljajo v osnovi skupek matematičnih objektov, ravnin, točk in krivulj. Predstavljati krivulje v 2D sliki je zahtevno opravilo, saj se mora ob vsaki spremembi preračunati celotna slika. V ta namen tehnika rasterizacije nadomesti vektorsko sliko z rastersko. Rasterska slika je sestavljena iz točk, ki vsebujejo informacijo o barvi. Skupek teh barvnih točk nam tvori sliko. V nasprotju z vektorsko sliko se kvaliteta rasterske slike pri povečanju zmanjša, vendar je samo preračunavanje spremembe slike hitro v nasprotju z vektorsko sliko, ki je skupek matematičnih objektov katerim je treba prišteti rastezni faktor. Na sliki 3 imamo najprej črne črte, ki predstavljajo vektorsko sliko, nato pa sliko postavimo na mrežo in slikovne elemente mreže, v katerih leži vektorska slika pobarvamo. Dobili smo rastersko sliko. V kolikor bi vzeli začetno mrežo, ki bi imela manjše slikovne elemente, bi bila slika po rasterizaciji bolj podobna originalu.



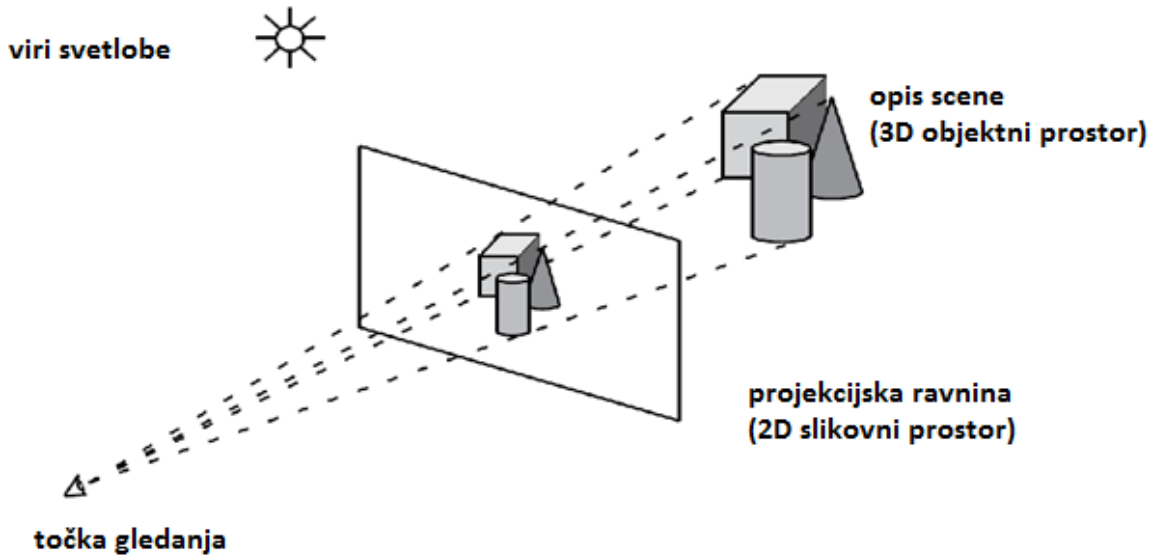
Slika 3: Primer postopka rasterizacije.[1]

Postopek rasterizacije je hiter, zato je zelo razširjen pri algoritmih v realnem času. Za predstavo si lahko vzamemo okolja v računalniških igrah. Okolja in predmeti v računalniških igrah so narejeni vektorsko, saj ustvarjalci postavljajo matematične objekte v prostor ter jim dodajajo barve in ostale lastnosti. Zaradi toka igre računalnik ne more ob vsakem premiku igralca preračunati vseh objektov in jih tako vektorsko povečati oziroma zmanjšati. V ta namen se uporabi metoda rasterizacije, ki objekte spravi v mrežo barvnih točk in jih prikaže igralcu. Za neopazno spreminjanje slik je poskrbljeno s količino slik v sekundi. Najmanjša količina slik v sekundi, kjer človeško oko ne zazna sprememb, je 24.

[1] Vir: http://iloveshaders.blogspot.com/2011/05/how-rasterization-process-works.html

2.1.2. *Žarkovno zlivanje*

Pri žarkovnem slikanju se scena, ki se upodablja, razdeli na slikovne točke. Ravnino slikovnih točk na sliki 4 prikazuje projekcijska ravnina. Nato se iz točke pogleda skozi vsako slikovno točko na tej ravnini izstreli snop svetlobe. Svetloba potuje skozi prostor, vendar se v primeru, da naleti na ploskvo objekta ne odbije, zaradi tega je postopek hitrejši od postopka žarkovnega sledenja. Ob trku žarka ob ploskev objekta se izračuna barva za posamezno barvno piko. Pomembno je, da se svetlobo izstreli iz točke pogleda, saj drugače slika ne bi predstavljala 3D objektov. Metoda se veliko uporablja za upodabljanje v realnem času, kjer so potrebne velike količine slik na sekundo. Z žarkovnim zlivanjem se tudi odkriva, katere ploskve so v ospredju modelov in katere v ozadju, ter se s tem ugotovi, katere objekte s scene je potrebno prikazati. Glede na čas potreben za potovanje potovanje posameznega žarka pa lahko izračunamo tudi globino glede na ostale objekte in jih zato lahko upodobimo v 3D prostoru.



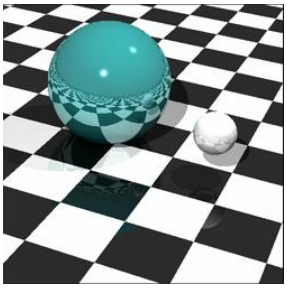
Slika 4: Projeciranje objektov na ravnino. [8]

Problem tehnike žarkovnega zlivanja je izbira prave ločljivosti projekcijske ravnine. V kolikor izberemo premajhno ločljivost, bodo objekti na sliki nepravilnih oblik, saj se barva vedno določi za celotno barvno piko. Če izberemo preveliko, s tem povečamo računsko breme algoritma.

Slike s pomočjo žarkovnega zlivanja so uporabljale prve igre s 3D grafiko.

2.1.3. *Žarkovno sledenje*

Žarkovno sledenje je podobno žarkovnemu zlivanju. Proces teče na podoben način, saj imamo prav tako projekcijsko ravnino in točko pogleda, s katere pošljemo žarek skozi vsako barvno točko na projekcijski ravnini. Žarek potuje skozi prostor scene in prileti na ploskev objekta na sceni. Preračuna se še svetloba, ki jo oddajajo viri v sceni ter se glede na barvo, material in teksturo izračuna barva te točke. Od ploskve se svetloba odbije naprej in tako ustvari bolj realističen pogled. V naravi je svetloba elektromagnetno valovanje, zato prenaša tudi informacijo o barvah. Prav tako mora odbojni žarek nositi tudi informacijo o barvi, od kjer se je odbil. Na ta način vidimo v odbojih lahko tudi odseve. Vidnost odsevov je odvisna od materiala. Primer žarkovnega sledenja je viden na sliki 5, kjer v kroglah vidimo odseve talne plošče in virov svetlobe nad kroglama.



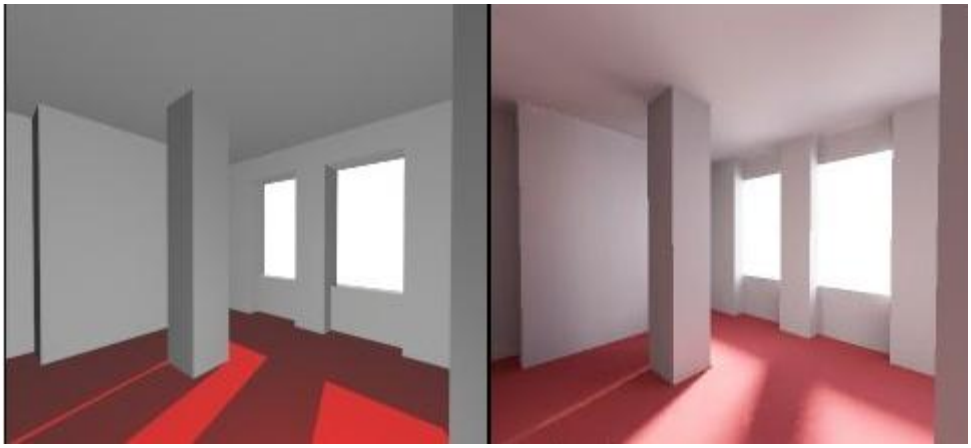
Slika 5:Primer žarkovnega sledenja.[2]

2.1.4. *Razpršena osvetljenost*

Metoda razpršene osvetljenosti je računsko najbolj intenzivna od vseh opisanih metod, a je tudi upodobljena slika najbližje sliki realnega sveta. Metoda temelji na ideji, da za razliko od prejšnjih metod svetloba ne prihaja iz točke pogleda, ampak vsaka površina oddaja svetlobo. Nekatere ploskve oddajajo močnejšo, druge pa šibkejšo, odvisno od materiala. Delo te upodobitve traja veliko več časa od ostalih, a za razliko od vseh ostalih se tukaj preračuna celoten prostor, zato je računanje svetlobe, če spremenimo pogled takojšnje. Algoritem poskrbi tudi za sence različnih odtenkov.

Na sliki 6 vidimo primer prostora kjer se je uporabil algoritem za razpršeno svetlobo in algortem brez te metode. Prostor je videti bolj realističen in ima celo različne odtenke senc.

[2] Vir: http://ajaxian.com/archives/javascript-ray-tracing



Slika 6: Slika brez (desno) in slika z (levo) uporabo metode razpršene osvetljenosti.[3]

[3] Vir: http://en.wikipedia.org/wiki/Radiosity_(3D_computer_graphics)

2.2. Upodabljanje slik v realnem času

Upodabljanje v realnem času se uporablja v namene, kjer je potreben takojšnji prikaz rezultatov in tam, kjer slika ni določena statično. Primeri takih uporab so video in računalniške igre ter simulacije. Igre in simulacije ustvarjajo navidezno premikanje s tvorjenjem niza slik, katerih mora biti za uspešno ustvarjeno iluzijo od 24 do 30 na sekundo. Sodobne igre nam postrežejo z 60 do 160 slikami na sekundo, v simulatorjih pa še več, zato se algoritmi za upodabljanje v zadnjem času osredotočajo na realizem videnega. Izdelava realnega pogleda je izjemno zahteven proces in je bil do nedavnega zaradi omejenih računalniški zmogljivosti praktično neizvedljiv. Danes so te omejitve presežene z grafičnimi procesorji, pa tudi centralno procesne enote, ki omogočajo vzporedno izvajanje procesov lahko prevzamejo del računskega bremena. Vendar je potrebno v tem primeru proces upodabljanja prilagoditi vzporednemu delovanju.

2.3. Pred-upodobljene slike

Pred-upodobljene slike se izdeluje tako, da se slike najprej upodobi, nato sestavi skupaj in šele nato se dokončan produkt lahko predstavi. Večinoma so to filmi in animacije. V to kategorijo spada tudi upodabljanje slik za predstavitev arhitekturnih in strojniških izdelkov. Običajno se posamezna slika upodablja od nekaj ur do nekaj dni. Stremi se k čim bolj realističnemu okolju prikazanih slik. Pozornost je na efektih kot so dim, dež, ogenj, megla, poudarek je tudi na odsevnosti tekočin in njihovi transparentnosti ter na potovanju svetlobe skozi predmete iz stekla in podobnih materialov. Tudi tu je potrebna velika računalniška moč, zato se pojavlja potreba po vzporednem in porazdeljenem računanju in v nekaterih primerih je potrebno uporabiti celo superračunalnike za upodabljanje posameznih zahtevnih prizorov.

2.4. Zahtevnejši deli upodabljanja

Pri upodabljanju, ki stremi k prikazovanju čim bolj realnih slik, je potrebno dodati najrazličnejše učinke, ki pa so računsko zahtevni. V nadaljevanju so našteti in opisani nekateri od njih.

-senčenje (*ang. shadows*)

Senčenje na področjih, kjer padajo sence objektov glede na vir svetlobe.

-odboji (*ang. reflection*)

Odboji se lahko pojavijo na površini tekočin, v ogledalih, steklu, …

-ukloni (*ang. diffraction*)

Uklone je potrebno preračunati, ko se svetloba lomi na robu nekega objekta in nadalje pada pod drugačnim kotom.

-posredna osvetljenost (*ang. indirect illumination*)

Posredna osvetljenost je odboj svetlobe od objekta, na katerega je prej padla svetloba iz vira.

-prosojnost (*ang. transparency*)

Prosojnost na slikah lahko pomeni tako prepuščaje svetlobe skozi nek prosojen objekt, kot tudi izris objekta v ozadju.

-upodabljanje teksture (*ang. texture –mapping*)

Upodabljanje ne-gladkih površin , kot so npr. zidovi, travnate površine, itd.

-upodabljanje premikanja (*ang. motion blur*)

Premikanje kamere ali objekta se rešuje z zameglitvijo in kasnejšim ostrenjem objektov na sliki.

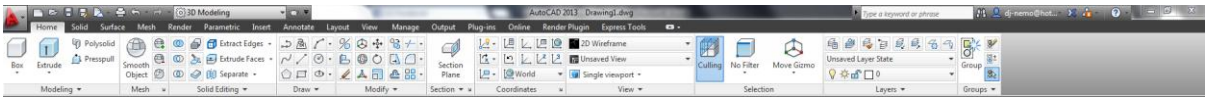
3. Pregled orodij za izdelavo tehničnih risb

3.1. AutoCAD

AutoCAD je orodje za tehnično načrtovanje v 2D in 3D perspektivi. Programski paket AutoCAD razvija podjetje Autodesk Inc., ustanovljeno leta 1982 s strani programerja Johna Walkerja [3] . Od prve verzije programa AutoCAD je sledilo še 18 verzij in 26 izdaj. Zadnja izdaja programa je AutoCAD 2013, ki je izšla 27. marca 2012. Različico lahko dobimo brezplačno za trideset dnevno testiranje in je pripravljena za operacijske sisteme Windows in Mac OS X. Prodajna cena osnovne različice je 5873,25 €.

AutoCAD ponuja možnost dodajanja vtičnikov za povečanje funkcionalnosti osnovnega programa. Sam AutoCAD je napisan v programskem jeziku C, vtičnike pa je priporočeno programirati na Autodesku v programskem jeziku Visual Basic .NET. V zadnjih izdajah je poenostavljeno tudi dodajanje vtičnikov v sam AutoCAD, saj le-te prenesemo v namestitveno mapo in ob ponovnem zagonu se vtičnik samodejno namesti.

Osnovni format zapisa datotek je DWG podpira pa tudi DXF in nekatere druge.



Slika 7: Slika uporabniškega vmesnika za program AutoCAD.

Na sliki 7 je uporabniški vmesnik programa AutoCAD. Odprt je le en od zavihkov in vidimo lahko, da AutoCAD omogoča ogromno funkcij, zato so proizvajalci razvili različne rešitve za prilagajanje uporabniškega vmesnika tako, da bo prijazen uporabniku, hkrati pa obstaja tudi ukazna vrstica za bolj izkušene uporabnike. Pomembn dodatek grafičnemu vmesniku so prednastavljene delovne površine. Izbiramo lahko med dvo in tro dimenzionalno delovno površino ter nekaterimi drugimi vnaprej pripravljenimi delovnimi okolji. Svoje okolje si lahko tudi nastavimo in shranimo. Funkcije, ki jih ne uporabljamo si lahko skrijemo, ter na njihovo mesto postavimo druge bolj uporabne, prav tako lahko skrivamo in prikazujemo posamezne zavihke.

3.2. SketchUp

SketchUp je prav tako program za risanje načrtov in modelov, tako arhitekturnih kot tehničnih. Prvotni snovalec programa je bilo podjetje Last Software, ki si je leta 2000 zamislilo program za načrtovanje, ki bo preprost za uporabo in enostaven za učenje v primerjavi z dotedanjimi programi za načrtovanje. Pospremili so ga s sloganom »*3D za vse*«. Leta 2006 je podjetje Google kupilo Last Software in s tem prevzelo tudi razvoj SketchUp-a. Google je razvil dve različici programa, prvo imenovano SketchUp, drugo pa SketchUp Pro. Slednja je bila dodelana različica prve in je bila namenjena za profesionalno uporabo. Profesionalna različica je bila plačljiva. Aprila 2012 je Google prodal SketchUp podjetju Trimble. Objavili pa so skupni nadaljnji razvoj.

Program je dostopen uporabnikom operacijskih sistemov Windows, Mac OS X ter uporabnikom Linuxa preko programa Wine. Cena različice SketchUp Pro je 495,04 € [6].

SketchUp prav tako kot AutoCAD podpira formata DWG in DXF za zapis datotek.

Slika 8: Uporabniški vmesnik SketchUpa.

Prav tako kot AutoCAD tudi SketchUp ponuja možnost vtičnikov. Program je napisan v programskem jeziku Ruby. Vtičniki v SketchUp-u tako predstavljajo le nove metode v programu in se jih zato prav tako implementira v Rubyu.

SketchUp prav tako kot AutoCAD podpira formata DWG in DXF za zapis datotek.

Na sliki 8 vidimo, da je osnovni meni SketchUpa bolj enostaven. V kolikor posamezna funkcija ponuja še več možnosti se nam te odprejo v manjšem oknu, kjer jih lahko urejamo. Za razliko od AutoCADa je vmesnik narejen tudi tako, da imamo funkcije razporejene v spustnih seznamih.

3.3. CATIA

Slika 9: Logotip podjetja Dassault in programa CATIA [7].

CATIA je programski paket namenjen predvsem risanju modelov za avtomobilsko, letalsko in ladjedelniško industrijo. Na pa primeren za arhitekturo. Razvijati so ga začel leta 1977 v letalskem podjetju Avisions Marcel Dassault. Podjetje je program najprej razvijalo za lastne potrebe. Leta 1981 se je razvilo podjetje Dassault, ki se je začelo ukvarjati s prodajo programskega paketa CATIA ter ga prilagajati za podjetja v avtomobilski in ladijski industriji zadnja izdaja je CATIA V6 2013. Na voljo je uporabnikom operacijskega sistema Windows. Cena za programski paket je spremenljiva glede na zahteve in želje uporabnika . Brezplačne poizkusne izdaje ni mogoče dobiti.

CATIA je napisana v programskem jeziku C++.

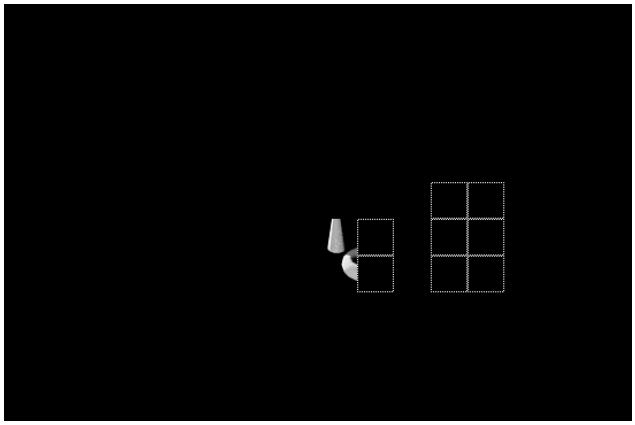
4. Algoritmi za upodabljanje

Algoritmi za upodabljanje izdelajo sliko iz načrta oziroma scene. Kakovost končnega izdelka je odvisna od posameznega algoritma in od časa, ki ga ima na voljo. Zaradi hitrejšega upodabljanja se za večja upodabljanja najema računske farme za upodabljanje slik (*ang. render farms*). Računske farme za upodabljanje so skupina več računalnikov, včasih superračunalnikov, ki si delijo delo pri upodabljanju slike. Takšen način upodabljanja je nujen, ker se bi sicer proces upodabljanja izvajal izjemno dolgo. Tako bi npr. za risani film *Madagascar: Escape 2 Africa*, na računalniku z eno-procesorsko enoto potrebovali kar 30 milijonov ur oziroma več kot 3400 let za upodobitev celotnega filma [8].

Zato je potrebno upodabljanje izvajati vzporedno in/ali porazdeljeno. Na področju vzporednega upodabljanja lahko uporabimo več različnih oblik paralelizma. To so podatkovni, časovni in postopkovni paralelizem [8]. Nekateri so bolj uporabni za ene aplikacije, medtem ko so drugi primerni za druge. Nekateri algoritmi pa vsebujejo tudi kombinacijo vseh pristopov.

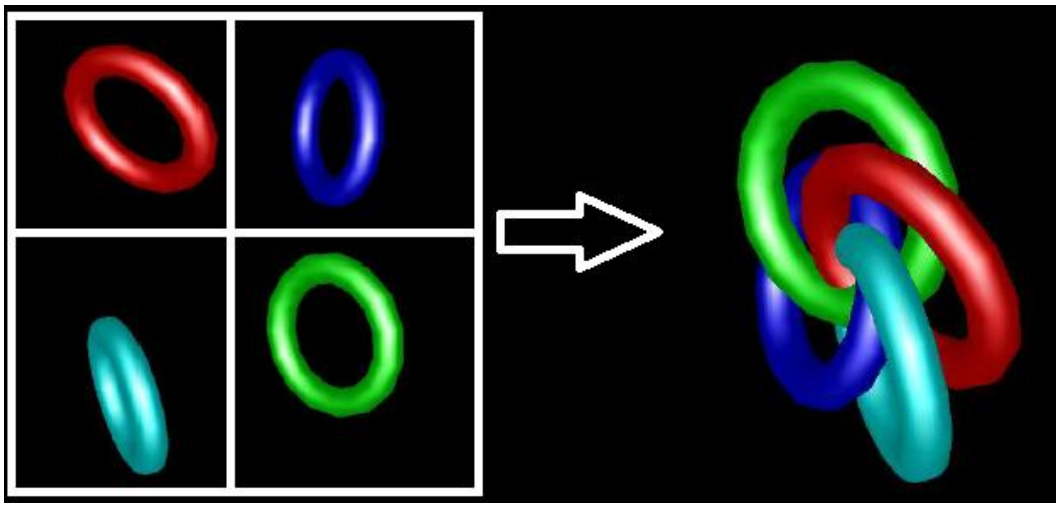
4.1. Podatkovni paralelizem

Podatkovni paralelizem pomeni porazdelitev dela na več procesov tako, da sliko razdelimo in vsakemu procesu določimo en del za upodobitev. Pri tem mora vsak proces opraviti celotni postopek upodabljanja za svoj del slike. Za učinkovito upodabljanje je potrebno število procesov omejiti s številom procesnih enot, ki jih imamo na voljo. Ker je omejitev števila procesov na število jeder, ki jih imamo na voljo, sprejemljiva, se ta oblika paralelizma pogosto uporablja za upodabljanje. Običajno se namreč zahtevno upodabljanje izvaja na več procesnih enotah, sploh če gre za kompleksne scene. Število procesorjev se lahko s strani algoritma tudi omeji glede na resolucijo končne slike, kompleksnost scene ali želenega časovnega okvirja, v katerem naj bi bilo delo končano. Slika 10 prikazuje podatkovni paralelizem v AutoCADu. Algoritem upodablja osem poddelov slike, ki so označeni.



Slika 10: Podatkovni paralelizem v AutoCADu.

Algoritmi, ki za upodabljanje uporabljajo podatkovni paralelizem, slonijo na dveh metodah. Uporabljajo lahko objektni ali slikovni paralelizem. Pri objektnem paralelizmu se slika razdeli glede na objekte, ki so na sceni. Vsak del slike nato vzporedno prehaja skozi začetne faze upodabljanja. Proces objektnega paralelizma je prikazan na sliki 11. Tu mislimo na izdelavo senc in računanje svetlobe, transformacije modelov, medtem ko se faza rasterizacije opravi nedeljivo. Slikovni paralelizem temelji na razdelitvi slikovnih elementov na območja, ki se jih nato upodablja vsakega v svojem procesu. Ta paralelizem deluje vzporedno predvsem v drugi fazi upodabljanja – v fazi rasterizacije, medtem ko je prvi del računanja svetlobe in podobne operacije izveden zaporedno.



Slika 11: Objektni paralelizem [10].

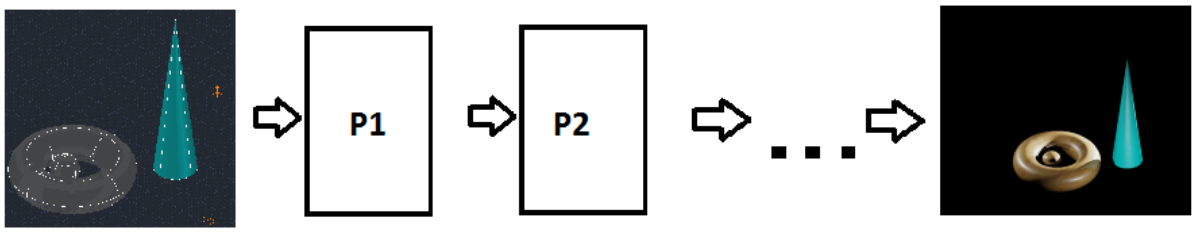
Običajno je lahko stopnja paralelizma zelo visoka, saj imajo običajni modeli od nekaj sto do nekaj milijonov objektov, ter od tisoč do nekaj sto milijonov slikovnih točk [8]. V praksi se izkaže, da so običajno modeli razdeljeni na skupine slikovnih elementov in objektov, saj se tako zmanjša breme komunikacije pri podatkovnem paralelizmu. Današnji algoritmi so kombinacija tako objektnega kot slikovnega paralelizma.

4.1.1. *Časovni paralelizem*

Časovna vzporednost se uporablja predvsem v animacijah in filmih, oziroma kjer je potrebno narediti več slik. Vsak proces izvaja celoten postopek upodabljanja, vzporednost pa se doseže s porazdelitvijo različnih časovnih okvirjev med procesi. Na koncu sestavimo animacijo z združitvijo slik . Ta oblika paralelizma je podvrsta postopkovnega paralelizma, saj namesto delitve slike izvajamo delitev po času.

4.2. Postopkovni paralelizem

Druga možnost za vzporedno upodabljanje je postopkovni paralelizem. Prikazuje ga slika 9. Postopek upodabljanja razdelimo na posamezne funkcije in vsakega izmed procesov določimo za opravljanje ene ali več funkcij. Ko proces opravi svojo funkcijo, preda podatke procesa naslednjim procesom, sam pa sprejme podatke od predhodnega procesa in ponovno opravi svojo funkcijo. Primer postopkovnega paralelizma je na sliki 12, kjer model predamo prvemu procesu (P1), ki preračuna kako potuje svetloba po sceni. Ob zaključku dela preda podatke naprej procesu dva (P2), ki preračuna padanje senc ter tako naprej do končne slike. Nekateri procesi lahko tečejo vzporedno zato na ta način pride do pohitritev.



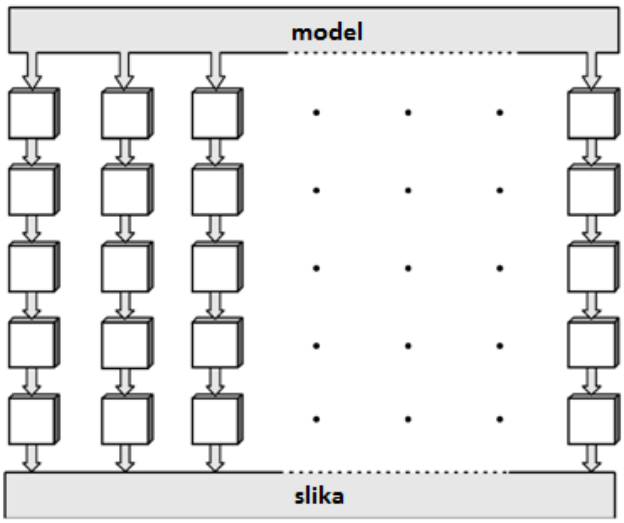
P1 = Preračunavanje poti svetlobe
P2 = Preračunavanje senc

Slika 12: Primer postopkovnega paralelizma.

Pri postopkovnem paralelizmu se pojavita predvsem dva problema. Prvi izmed problemov je obstoj ozkega grla pri funkciji, ki potrebuje največ procesorskega časa za obdelavo. Posledično morajo vsi prihajajoči kosi čakati na prost dostop do niti. Drugi problem, ki se pojavi, pa je stopnja paralelnosti. Ta je namreč lahko največ taka, kot je število nedeljivih funkcij v procesu upodabljanja.

4.3. Kombinacija različnih pristopov

V želji po še boljšem paralelizmu se v sodobne algoritme vgrajujejo hibridni tipi paralelizmov, ki združujejo oba tipa paralelizma. Pogosto je uporabljen hibrid med postopkovnim in podatkovnim paralelizmom, kot ga prikazuje slika 13. Končna slika je v spodnjem primeru razdeljena na več delov po principu podatkovnega paralelizma. Vsaka puščica, ki gre iz modela predstavlja en tak del. Nato je vsak izmed teh delov razdeljen na procese tako, da vsak proces prevzame posamezen del upodabljanja po principu postopkovnega paralelizma. Razdelitev na procese na sliki prikazujejo procesi postavljeni v vrste.



Slika 13: Hibridni(podatkovni in postopkovni) paralelizem.

4.4. Komercialne storitve za porazdeljeno in vzporedno upodabljanje

Rešitev za pohitritev upodabljanja z uporabo vzporednega in porazdeljenega procesiranja je že kar nekaj. Pregledali bomo dve komercialni storitvi.

4.4.1. *Autodesk 360 Rendering*

Podjetje Autodesk, ki je razvijalec programskega orodja AutoCAD, ponuja svojo storitev za upodabljanje tehničnih načrtov s pomočjo superračunalnika na spletu oziroma za uporabnike njihovega programa. Za uporabo se je potrebno prijaviti na spletni strani Autodeska 360. Za upodabljanje lahko naložimo datoteko preko spletnega brskalnika in jo tako upodobimo, na voljo pa je tudi vmesnik v AutoCADu. Z nakupom AutoCADa pridobimo 100 upodabljanj na leto brezplačno. Za več upodabljanj trenutno ni cene, saj uporabnik ne more izvesti več kot 100 upodabljanj. [11]. V primeru brez nakupa je uporabniku, ki se registrira na voljo 75 upodabljanj za testiranje.

4.4.2. *Rebus farm*

Ena izmed največjih računskih farm za upodabljanje je Rebus Farm, ki ga ponuja podjetje RebusMedia Ta ponuja storitev za upodabljanje, ki se zaračunava za 3,9 dolarskih centov na GHz [12] ali drugače za vsako milijardo operacij, ki jih izvede njihov računalnik, se plača 3,9 dolarskih centov. Program moramo namestiti na naš računalnik, nato pa preko programa naložimo sliko na njihov strežnik, ki nato sliko upodobi. Sliko lahko nato prenesemo na svoj računalnik. Program za upodabljanje lahko upodobi datoteke več različnih programov za animacije in filme. Rešitev z uporabo zunanjega programa za upodabljanje je uporabniku manj prijazna.

5. Izdelava vtičnika

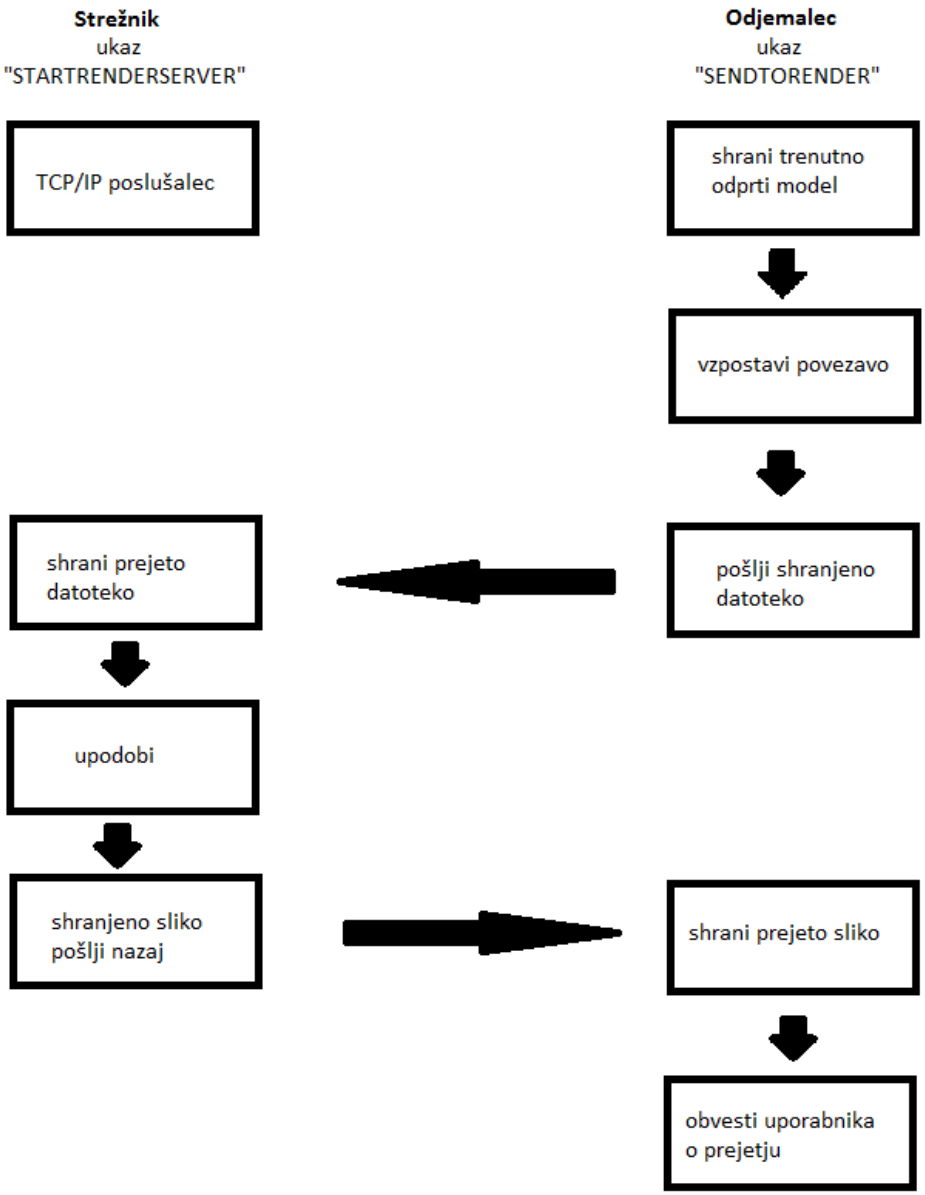
Glavni problem pri upodabljanju je gotovo čas, ki ga proces porabi za izdelavo upodobljene slike. V našem primeru smo ta problem reševali z izgradnjo vtičnika v programskemu paketu za arhitekturno in tehnično risanje, ki bi omogočal ločeno upodabljanje tehničnih načrtov in risb na namenskih računalniških arhitekturah, kjer bi se lahko zagotovilo vzporedno upodabljanje slik. S stališča pogostosti uporabe ter širokega spektra uporabe smo si izbrali programski paket AutoCAD. Vtičnik je narejen tako, da uporabniku omogoča avtomatizirano upodabljanje slik s klicem enega ukaza, ki izvede upodabljanje tako, da prenese trenutno sliko na strežniški računalnik z več procesorji in tam upodobil sliko ter jo prenese nazaj do uporabnika. Pri izdelavi vtičnika smo stremeli k čim enostavnejši uporabi.

5.1. Delovno okolje

Vtičnike v AutoCADu se izdeluje v programskem jeziku Visual Basic.NET (VB.NET). Vtičnik smo izdelali v programskem okolju Microsoft Visual Basic 2010 Express, ki je sicer brezplačno za uporabo, vendar lahko to orodje nadomestimo tudi s plačljivim okoljem Visual Studio. Visual Basic je objektno naravnan programski jezik, ki temelji na .NET ogrodju. Razvija ga podjetje Microsoft in je trenutno v različici 10.0. Prva objava programskega jezika je bila leta 2001. Namestili smo tudi okolje ObjectARX, ki nam je omogočalo dostop do funkcij AutoCADa. ObjectARX je sicer orodje, ki se uporablja kot vmesnik za vse Autodeskove produkte. Seveda smo potrebovali tudi programski paket AutoCAD, kjer smo dokončno implementirali in preizkusili vtičnik. Uporabili smo izdajo programskega paketa z letnico 2013.

5.2. Zgradba vtičnika

Vtičnik je sestavljen iz dveh procesov – procesa strežnika in procesa odjemalca. Ob zagonu odjemalca ta shrani trenutno odprti model ter ga pošlje strežniku. Strežnik prejeto datoteko shrani na svoji strani, nato pa jo odpre in upodobi. Sliko, katero je upodobil pošlje nazaj odjemalcu. Odjemalec sliko shrani in o tem obvesti uporabnika. Opisani postopek je prikazan na sliki 14.



Slika 14: Podatkovni tok

Orodje ObjectARX ne podpira dela z več procesi na enkrat, zato ne moremo v programu AutoCAD izvajat ničesar v času, ko vtičnik deluje.

5.2.1. *Proces strežnika*

Ob prejetju ukaza »STARTRENDERSERVER« AutoCAD zažene metodo, ki se izvaja v neskončni zanki. V zanki se kličejo preostale metode potrebne za delovanje.

```
<CommandMethod("StartRenderServer")>
Public Sub StartServer()
    While (True)
        ListenAndReceive()
        Render(dwgFileName, imageFileName)
        ResponseStreamToClient(imageFileName)
        DisconnectTCP()
    End While
End Sub
```

Izvorna koda 1: Strežnikova glavna metoda.

Strežniški del programa gre najprej skozi metodo »ListenAndReceive«, ki aktivira TCP poslušalca za sprejemanje klientov. Na poziv odjemalca se sproži in shrani tehnično risbo v formatu .dwg, ki mu jo odjemalec pošlje. Nato program začne z izvedbo metode »Render« ki ima dva parametra in to sta ime tehnične risbe ter ime slike, katero bo metoda upodobila. Jedro metode za upodabljanje je precej enostavno. Najprej tehnično risbo odpremo, nato v ukazno vrstico pošljemo niz ukazov, ki nam risbo upodobijo v sliko in zaprejo risbo. Postopek je prikazan v izvorni kodi 2.

```
Dim docCol As DocumentCollection = Application.DocumentManager
Dim doc As Document = DocumentCollectionExtension.Open(docCol, filePath, False)

doc.SendStringToExecute("_RENDER SAVEALL CLOSE ", True, False, True)
```

Izvorna koda 2: Jedro metode za upodabljanje.

Program nato pošlje upodobljeno sliko iz izvorne kode 2 nazaj uporabniku, ter zapre povezavo z njim. Ta postopek se ponavlja v neskončni zanki.

Omeniti velja tudi posebno strukturo, v kateri se na strežniški strani hranijo podatki za posameznega odjemalca. Struktura je bila vpeljana z mislijo na nadaljnji razvoj, ko bo vtičnik sprejemal več odjemalcev naenkrat ter jih shranjeval v čakalno vrsto. V izvorni kodi 3 lahko vidimo to strukturo. V strukturi imamo dve lastnosti in sicer datoteko, ki smo jo prejeli ter sliko, ki. Datoteki sta shranjeni v obliki, ki dovoljujejo možnost pošiljanja. Za lažje dostopanje do lastnosti imamo še metodi za branje in pisanje.

```
Public Structure ClientWrapper
    Public Property File() As MemoryStream
        Get
            Return m_File
        End Get
        Set(value As MemoryStream)
            m_File = value
        End Set
    End Property
    Private m_File As MemoryStream
    Public Property Result() As FileStream
        Get
            Return m_Result
        End Get
        Set(value As FileStream)
            m_Result = value
        End Set
    End Property
    Private m_Result As FileStream
End Structure
```

Izvorna koda 3: Struktura za shranjevanje uporabnikovih podatkov.

5.2.2. *Proces odjemalca*

Proces odjemalca je izveden v uporabniškem vtičniku. Zažene se z ukazom »SENDTORENDER«. Ob klicu tega ukaza se začne izvajanje metode prikazane v izvorni kodi 4.

```
<CommandMethod("SendToRender")>
Public Sub SendToRenderer()
    SaveCurrentDrawingToFile(inputFileName)
    ConnectTCP()
    SendStreamToServer(inputFileName)
    ReceiveStreamFromServer(outputFileName)
    DisconnectTCP()
    editor.WriteMessage(vbLf + "Slika se nahaja: " + ClientFolder + vbLf)
End Sub
```

Izvorna koda 4: Odjemalčeva glavna metoda.

Na uporabnikovi strani si vtičnik najprej shrani trenutno odprto tehnično risbo, nato pa se poveže na strežnik. V primeru, da je povezava uspela se začne prenos risbe na strežnik ter se nato počaka, da se začne polniti kanal med odjemalcem in strežnikom z upodobljeno sliko. V tej fazi prejemanja odjemalec tudi shrani sliko v svojo mapo na trdem disku. Ob koncu prenosa se povezava prekine, vtičnik pa izpiše pot do slike.

5.3. Izdelava

Postopek izdelave smo začeli z izdelavo metode, ki bo shranjevala trenutni model. Metoda sama kreira mapo na disku, v katero shrani model. Model se shrani v format .dwg verzije trenutno odprtega AutoCADa. Zaradi komunikacije med strežnikom in odjemalcem smo morali nato narediti TCP/IP povezavo ter preko nje poslati datoteko modela odjemalčeve strani na stran strežnika. Strežnik prejeto datoteko shrani v svojo mapo, ki jo tudi ustvari, če je potrebno. Strežnik nato odpre model in požene ukaz za upodabljanje v AutoCADu. Tu je potrebno poudariti, da smo za postopek upodabljanja uporabili AutoCAD-ov algoritem, ki pa je narejen tako, da omogoča vzporedno izvajanje upodabljanja, če je to mogoče. AutoCADov algoritem za upodabljanje nato shrani sliko v prej narejeno mapo na disku. Potem se na strežniškem koncu zapre odprti model ter pošlje sliko iz datoteke nazaj uporabniku. Ta se na uporabnikovi strani shrani v mapo na disk ter v uporabniški vmesnik izpiše, kje se slika nahaja. S pomočjo AutoCADovega orodja za urejanje uporabnikovega vmesnika smo izdelali tudi uporabniški vmesnik. Prikaz vmesnika je na sliki 12.

5.4. Namestitev vmesnika

Vmesnik je zapakiran v arhivu, ki ga je potrebno razširiti. V njem imamo mapo z imenom, oblike »ime.bundle«, ki predstavlja standardno obliko zapisa imen vtičnikov v AutoCADu. V kolikor imamo odprt program AutoCAD, ga moramo najprej zapreti. Mapo vtičnika je potrebno premestiti v mapo »ApplicationPlugins«, ki se nahaja v namestitveni mapi Autodeska. Tako je vtičnik nameščen. Ob naslednjem zagonu AutoCADa se bo vtičnik samodejno naložil v AutoCAD.

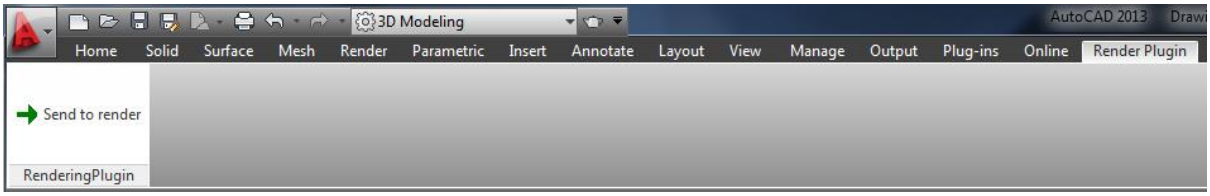
5.5. Uporaba vmesnika

5.5.1. *Vtičnik strežnika*

Strežniški vtičnik se samodejno naloži ob zagonu AutoCADa. Za zagon strežnika je potrebno v ukazno vrstico AutoCADa vpisati »STARTRENDERSERVER«. Trenutno sliko, ki se upodablja vtičnik shrani v mapo »ServerFolder«, ki jo ob prvi uporabi ustvari.

5.5.2. *Vtičnik odjemalca*

Vtičnik odjemalca se prav tako kot strežniški samodejno naloži ob zagonu AutoCADa. Uporabniški vmesnik, ki je prikazan na sliki 15, se naloži v zavihek »Render Plugin«. Za isto funkcijo lahko tudi v ukazno vrstico vpišemo »SENDTORENDER«. Ta ukaz nam bo poslal in shranil trenutno sliko ter jo poslal na strežnik. Za uspešno pošiljanje je v zavihku za upodabljanje potrebno označiti, da se rezultati shranjujejo na disk ter v tekstovno polje za pot do datoteke vstaviti »C:\ServerFolder\Picture.png«.



Slika 15: Uporabniški vmesnik.

Ob prejetju slike se v konzolo izpiše, da je slika sprejeta in izpisana je pot do mape, kjer se slika nahaja.

5.6. Testiranje in premerjava vtičnika z običajnim upodabljanjem v AutoCADu

Test je potekal na dveh prenosnih računalnikih. Vlogo strežnika je prevzel HP Envy 15-3040nr, ki ima 4 jedra s frekvenco 2,2 GHz. Vlogo odjemalca pa je prevzel prenosnik Lenovo 4446 – 25G, ki ga poganja dvojedrni procesor z taktom 2 GHz.

Slike, ki smo jih testirali, so se razlikovale v pogledu katerega upodabljamo, v ločljivosti končne slike, v številu odbojev svetlobe, načinu senčenja ter drugih atributih. Slike, ki so potrebovale več časa so po številu barvnih pik večje, kar postopek podaljša zaradi več preračunavanj. Nekatere slike imajo različno stopnjo natančnosti algoritma za upodabljanje, tiste z manjšo natančnostjo se hitreje upodobijo, kot tiste, ki imajo natančnost maksimalno. Pomemben atribut pri upodabljanju testnih vzorcev je imela tudi nastavitev odbojev svetlobe. Prve slike so imele število odbojev nastavljeno na 9, medtem ko je imelo zadnjih šest modelov nastavljeno na 19 odbojev. Posledično se z vsakim odbojem svetlobe veča porabljen čas za upodabljanje. V slikah se uporablja tudi različne tipe senčenja. V nekaterih slikah so vse sence enake v drugih pa so sence, ki se prekrivajo temnejše od drugih. Takšno senčenje potrebuje ponovno več preračunavanja. Modeli so v tabelah rezultatov označeni z zaporednimi številkami od 1 do 20, težavnost upodabljanja pa se z višanjem zaporedne številke veča.

Opravili smo tri vrste testiranj:

-Testiranje časa upodabljanja posamezne slike na prenosnem računalniku Lenovo,

-Testiranje časa upodabljanja posamezne slike na prenosnem računalniku HP,

-Testiranje časa upodabljanja posamezne slike s pomočjo vtičnika, kjer je vlogo strežnika prevzel računalnik znamke HP vlogo odjemalca pa računalnik Lenovo.

Za pridobitev rezultatov smo vsak test ponovili petkrat, nato pa izračunali povprečje. V tabelah 1 do 3 so prikazani posamični rezultati upodabljanja za vse tri teste. Časi, ki so manjši pomenijo lažje probleme za upodabljanje, medtem ko so zahtevnejši problemi potrebovali več časa.

Računalnik Lenovo						
Model	**Test 1 [s]**	**Test 2 [s]**	**Test 3 [s]**	**Test 4 [s]**	**Test 5 [s]**	**Povprečje [s]**
1	33	29	29	28	30	**29,8**
2	19	21	18	22	24	**20,8**
3	22	24	21	20	24	**22,2**
4	63	60	62	61	63	**61,8**
5	182	183	190	184	184	**184,6**
6	9	9	8	10	9	**9**
7	9	9	10	8	14	**10**
8	22	23	22	21	25	**22,6**
9	21	25	22	21	22	**22,2**
10	20	25	25	23	23	**23,2**
11	32	31	32	33	34	**32,4**
12	20	20	19	22	22	**20,6**
13	54	43	43	42	43	**45**
14	519	441	451	458	452	**464,2**
15	528	489	478	515	492	**500,4**
16	600	502	489	508	522	**524,2**
17	563	502	531	532	521	**529,8**
18	546	471	470	474	477	**487,6**
19	548	486	476	474	478	**492,4**
20	655	522	513	513	517	**544**

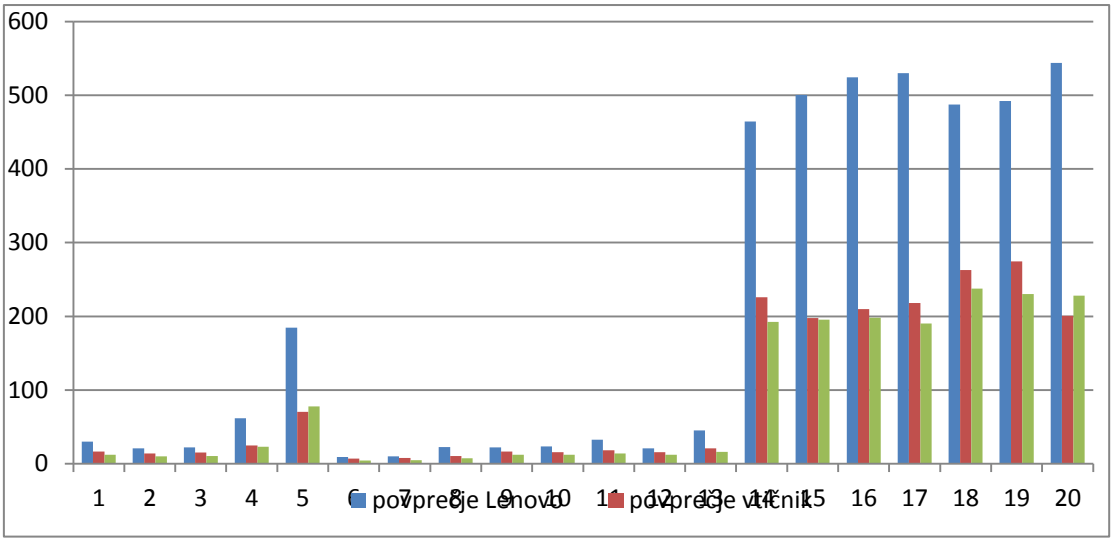
Tabela 1: Rezultati testiranj na prenosnem računalniku Lenovo.

Računalnik HP						
Model	**Test 1 [s]**	**Test 2 [s]**	**Test 3 [s]**	**Test 4 [s]**	**Test 5 [s]**	**Povprečje [s]**
1	11	12	12	11	14	**12**
2	10	9	9	10	11	**9,8**
3	10	12	10	10	10	**10,4**
4	20	21	18	21	34	**22,8**
5	64	78	72	101	73	**77,6**
6	5	4	4	4	4	**4,2**
7	5	5	5	5	4	**4,8**
8	7	7	7	7	8	**7,2**
9	12	12	12	12	12	**12**
10	12	12	12	12	12	**12**
11	14	14	14	14	14	**14**
12	12	12	12	12	12	**12**
13	16	16	17	16	16	**16,2**
14	244	167	181	197	174	**192,6**
15	171	168	175	247	217	**195,6**
16	174	190	246	193	188	**198,2**
17	221	179	210	221	120	**190,2**
18	236	273	264	229	185	**237,4**
19	259	274	208	216	193	**230**
20	236	240	213	247	205	**228,2**

Tabela 2: Rezultati testiranj na prenosnem računalniku HP.

Pošiljanje preko vtičnikov						
Model	**Test 1 [s]**	**Test 2 [s]**	**Test 3 [s]**	**Test 4 [s]**	**Test 5 [s]**	**Povprečje [s]**
1	15	17	16	18	16	**16,4**
2	12	16	14	14	14	**14**
3	13	15	15	17	16	**15,2**
4	28	23	22	25	26	**24,8**
5	76	72	73	68	63	**70,4**
6	6	7	7	7	7	**6,8**
7	7	8	8	8	8	**7,8**
8	9	11	10	11	11	**10,4**
9	14	17	16	19	16	**16,4**
10	13	16	17	16	17	**15,8**
11	16	18	18	19	19	**18**
12	14	16	16	16	16	**15,6**
13	18	23	21	22	21	**21**
14	267	275	216	173	198	**225,8**
15	185	163	191	238	212	**197,8**
16	202	208	175	246	218	**209,8**
17	189	204	224	237	236	**218**
18	278	259	268	243	267	**263**
19	259	260	275	296	283	**274,6**
20	243	186	190	189	195	**200,6**

Tabela 3: Testiranje časa upodabljanja posamezne slike s pomočjo vtičnika, kjer je vlogo strežnika prevzel računalnik znamke HP vlogo odjemalca pa računalnik Lenovo.



Slika 16: Povprečne vrednosti posameznih modelov.

Iz grafa na sliki 16 lahko jasno vidimo pohitritev procesa upodabljanja s pomočjo vtičnika in računalnika z več jedri. Odjemalcu modre barve smo postopek upodabljanja skrajšali na čas, ki ga prikazujejo rdeči stolpci.

V tabeli 4 so zbrane povprečne vrednosti iz vseh treh poskusov skupaj. V četrtem stolpcu tabele je izračunana absolutna razlika časa, ki ga je uporabnik privarčeval s tem ko je uporabil vtičnik. V petem stolpcu pa je izračunana relativna pohitritev kot odstotek časa absolutne razlike glede na čas izračuna upodobitve brez uporabe vtičnika.

povprečje Lenovo [s]	povprečje vtičnik [s]	povprečje HP [s]	časovna razlika med upodabljanjem na računalniku Lenovo ter z uporabo vtičnika [s]	realtivna pohitritev [%]
29,8	16,4	12	13,4	**45,0**
20,8	14	9,8	6,8	**32,7**
22,2	15,2	10,4	7	**31,5**
61,8	24,8	22,8	37	**59,9**
184,6	70,4	77,6	114,2	**61,9**
9	6,8	4,2	2,2	**24,4**
10	7,8	4,8	2,2	**22,0**
22,6	10,4	7,2	12,2	**54,0**
22,2	16,4	12	5,8	**26,1**
23,2	15,8	12	7,4	**31,9**
32,4	18	14	14,4	**44,4**
20,6	15,6	12	5	**24,3**
45	21	16,2	24	**53,3**
464,2	225,8	192,6	238,4	**51,4**
500,4	197,8	195,6	302,6	**60,5**
524,2	209,8	198,2	314,4	**60,0**
529,8	218	190,2	311,8	**58,9**
487,6	263	237,4	224,6	**46,1**
492,4	274,6	230	217,8	**44,2**
544	200,6	228,2	343,4	**63,1**

Tabela 4: Zbrane povprečne vrednosti testiranj in pohitritev.

Iz rezultatov iz grafa na sliki 16 in tabele številka 4 smo opazili, da je pri upodabljanjih, ki so manj zahtevna, razlika manjša. Pri zahtevnejših upodabljanjih pa se v večini primerov zmanjša čas upodabljanja za več kot 50 %. Povprečna pohitritev na vseh primerih znaša 44,8%.

S testiranji smo pokazali, da se z našim vtičnikom, lahko izrazito zmanjša čas upodabljanja. Čas, ki ga vzame prenos datotek pri sistemu vtičnika, je zanemarljivo majhen glede na čas izračunavanja upodabljanja.

Čas zagona algoritma za upodabljanje (približno 1 sekunda) je vštet v čase, ki so merjeni z vtičnikom, kar v časih upodabljanja na prenosniku Lenovo in HP ni všteto.

5.7. Problemi vtičnika

Problem samega vmesnika izvedenega v okolju ObjectARX je gotovo večnitenje. Želeli smo si, da bi lahko strežnik sprejemal in obdeloval več odjemalcev hkrati, kar pa brez poganjanja strežnika v večnitnem načinu ne moremo doseči. Prav tako bi bilo zelo priročno, da bi lahko algoritme za upodabljanje v strežniku nadomestili tudi z drugimi postopki upodabljanja in ne bi uporabljali algoritem iz programskega paketa AutoCAD. To nam med drugim predstavlja težavo tudi pri zaustavljanju procesa strežnika, saj ga lahko izklopimo le z zapiranjem AutoCADa, ki pa je lahko prav tako zaradi delovanja strežnika v eni niti v primeru izvajanja upodabljanja neodzivna.

V namene trženja vtičnika npr. za delovanje na superračunalnikih bi bilo potrebno v vtičnik implementirati še dostop do podatkovne baze in sistem zaračunavanja upodabljanja uporabniku. Potrebno bi bilo seveda rešiti tudi problem enonitnega obdelovanja podatkov.

6. Zaključek

V zaključni nalogi smo izdelali vmesnik za programsko orodje AutoCAD, ki omogoča avtomatizirano in porazdeljeno upodabljanje tehničnih načrtov in risb. Vmesnik je sestavljen iz dveh vtičnikov, ki jih registriramo v programu AutoCAD, pri čemer en prevzame vlogo strežnika, ki izvaja upodabljanje, in drugi vlogo odjemalca, ki ustrezno pripravi risbe za upodabljanje in izvede komunikacijo s strežnikom. Vmesnik je bil ustrezno testiran, kjer smo pokazali smiselnost in učinkovitost takšne rešitve za upodabljanje. Polna uporabnost vmesnika pa se bo pokazala šele z namestitvijo vtičnikov na visoko zmogljivem računalniku ali ustrezni računalniški arhitekturi, ki bi omogočala visoko stopnjo paralelizacije problema upodabljanja. V tem primeru bi lahko izvajali upodabljanje kot storitev predvsem za samostojne oblikovalce, študente in manjše arhitekturne pisarne, ki si ne morejo privoščiti zmogljivih računalniških sistemov za procese upodabljanja.

Z vtičnikom smo naredili prvi korak k razvoju učinkovitega in uporabniku prijaznega vzporednega in porazdeljenega upodabljanja. Vmesnik je namreč izdelan tako, da lahko z enim klikom oziroma enim ukazom pošljemo tehnični načrt ali risbo na ločen računalniški sistem za upodabljanje, jo tam upodobimo in upodobljeno sliko dobimo nazaj. Pri ostalih obstoječih orodjih je potrebno namestiti namenske programe ali preko spletnih vmesnikov izvajati upodabljanje, kar ni najbolj praktično s stališča uporabnika takšnih storitev. Drugi namen izvedbe vmesnika pa je bil ravno v tem, da smo ločili proces načrtovanja tehničnih načrtov od procesa upodabljanja, s čimer smo omogočili samostojno uporabo postopkov za upodabljanje slik in ob podpori ustrezne računalniške arhitekture tudi izvedbo teh postopkov na vzporeden način.

7. Literatura in viri

[1] M. Watt in A.Watt. Advanced animation and rendering techniques, New York, 10. 11. 1992

[2] M. Howison in E. W. Bethel in H. Childs. MPI-hybrid parallelism for volume rendering on large multi-core systems, Lawrence Berkeley national laboratory, 13. 7. 2010

[3] fundinguniverse.com. http://www.fundinguniverse.com/company-histories/autodesk-inc-history , 10. 9. 2012

[4] autodesk.com. http://usa.autodesk.com , 12. 9. 2012

[5] Thomas Funkhouser. Advanced computer graphics, Princeton university , 2002

[6] sketchup.com. www.sketchup.com , 12. 9. 2012

[7] 3ds.com www.3ds.com/products/catia/ , 12. 9. 2012

[8] Reuters. Animated field vies for Oscar nominatons, 5. 1. 2009

[9] Thomas W Crockett. Parallel rendering, Hampton, April 1995

[10] Parallel rendering, Austin, 2004

[11] 360.autodesk.com. https://360.autodesk.com , 4. 9. 2012

[12] rebusfarm.net. http://www.rebusfarm.net , 4. 9. 2012

[13] http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=15039

[14] vb.net-informations.com. http://vb.net-informations.com/communications/vb.net_socket_programming.htm , 27. 8. 2012

[15] forums.autodesk.com. http://forums.autodesk.com/t5/NET/bd-p/152 , 3. 9. 2012

8. Priloge

V nadaljevanju podajamo izvorno kodo za izvedbo strežnika in odjemalca vtičnika za programsko orodje AutoCAD.

8.1. Izvorna koda strežnika

```
Imports System.Net.Sockets
Imports System.Text
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports System.Net
Imports System.IO
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD
Imports Autodesk.AutoCAD.GraphicsSystem
Imports Autodesk.AutoCAD.Runtime.Interop
Imports System.Drawing

Namespace serverACDrendering
    Public Class Command
        Dim portNum As Integer = 8000
        Dim ipServer As String = "192.168.1.3"
        Private Const BUFFER_SIZE As UInt16 = 4096
        Private editor As Editor = Application.DocumentManager.MdiActiveDocument.Editor
        Private tcpClient As System.Net.Sockets.TcpClient
        Private Const ServerFolder As String = "C:\ServerFolder\"
        Private client As ClientWrapper
        Dim dwgFileName As String = "ServerDrawing.dwg"
        Dim imageFileName As String = "Picture.png"

        <CommandMethod("StartRenderServer")>
        Public Sub StartServer()
            While (True)
                ListenAndReceive()
                Render(dwgFileName, imageFileName)
                ResponseStreamToClient(imageFileName)
                DisconnectTCP()
            End While
        End Sub
```

```
        Private Sub ListenAndReceive()
            Dim ipAddress__1 As IPAddress = IPAddress.Parse(IpServer)
            Dim tcpListener = New TcpListener(ipAddress__1, PortNum)
            tcpListener.Start()
            Try
                tcpClient = tcpListener.AcceptTcpClient()
                If tcpClient.Connected Then
                    client = New ClientWrapper() With { _
                     .File = ReadStreamFromTCP() _
                    }
                    editor.WriteMessage(vbLf + "Received file ..." + vbLf)
                End If
            Catch ex As System.Exception
                editor.WriteMessage(ex.Message + vbLf + ex.StackTrace)
            End Try
            tcpListener.[Stop]()
        End Sub

        Private Sub Render(dwgFileName As String, imageFileName As String)
            Try
                Dim filePath As String = ServerFolder + dwgFileName
                Dim responseStream As MemoryStream = client.File
                If Not Directory.Exists(ServerFolder) Then
                    Directory.CreateDirectory(ServerFolder)
                End If
                If File.Exists(filePath) Then
                    File.Delete(filePath)
                End If
                If File.Exists(ServerFolder + imageFileName) Then
                    File.Delete(ServerFolder + imageFileName)
                End If
                Dim fileStream As FileStream = File.OpenWrite(filePath)
                responseStream.WriteTo(fileStream)
                fileStream.Close()

                Dim docCol As DocumentCollection = Application.DocumentManager
                Dim doc As Document = DocumentCollectionExtension.Open(docCol,
filePath, False)

                doc.SendStringToExecute("_RENDER SAVEALL CLOSE ", True, False, True)

            Catch ex As System.Exception
                editor.WriteMessage(ex.Message + vbLf + ex.StackTrace)
            End Try
        End Sub

        Private Sub ResponseStreamToClient(fileName As String)
            Try
                Dim fileStream = New FileStream(ServerFolder + fileName,
FileMode.Open, FileAccess.Read)
                WriteStreamToTCP(fileStream)
                fileStream.Close()
            Catch ex As System.Exception
                editor.WriteMessage(ex.Message + vbLf + ex.StackTrace)
            End Try
        End Sub
```

```
        Private Function ReadStreamFromTCP() As MemoryStream
            Dim readBuffer = New Byte(BUFFER_SIZE) {}
            Dim requestStream As NetworkStream = tcpClient.GetStream()
            Dim reader = New BinaryReader(requestStream)
            Dim streamLength As Integer = reader.ReadInt32()
            Dim responseStream = New MemoryStream()
            While streamLength > 0
                Dim lData As Integer = requestStream.Read(readBuffer, 0, BUFFER_SIZE)
                responseStream.Write(readBuffer, 0, lData)
                streamLength -= lData
            End While
            Return responseStream
        End Function

        Private Sub WriteStreamToTCP(stream As Stream)
            Dim reader = New BinaryReader(stream)
            Dim requestStream As NetworkStream = tcpClient.GetStream()
            Dim writer = New BinaryWriter(requestStream)
            Dim length As Integer = Convert.ToInt32(stream.Length)
            writer.Write(length)
            Dim buffer As Byte()
            Do
                buffer = reader.ReadBytes(BUFFER_SIZE)
                writer.Write(buffer)
            Loop While buffer.Length = BUFFER_SIZE
            writer.Flush()
            reader.Close()
        End Sub

        Private Sub DisconnectTCP()
            tcpClient.Close()
        End Sub

    End Class

    Public Structure ClientWrapper
        Public Property File() As MemoryStream
            Get
                Return m_File
            End Get
            Set(value As MemoryStream)
                m_File = value
            End Set
        End Property
        Private m_File As MemoryStream
        Public Property Result() As FileStream
            Get
                Return m_Result
            End Get
            Set(value As FileStream)
                m_Result = value
            End Set
        End Property
        Private m_Result As FileStream
    End Structure
End Namespace
```

8.2. Izvorna koda odjemalca

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports System.Net.Sockets
Imports System.Text
Imports System
Imports System.IO
Imports Autodesk.AutoCAD.DatabaseServices

Namespace ClientSide

    Public Class Commands
        Dim ipServer As String = "192.168.1.2"
        Dim portNum As Integer = 8000
        Private Const BUFFER_SIZE As Integer = 4096
        Private Const ClientFolder As String = "C:\ClientFolder\"
        Private ReadOnly editor As Autodesk.AutoCAD.EditorInput.Editor =
Application.DocumentManager.MdiActiveDocument.Editor
        Private tcpClient As System.Net.Sockets.TcpClient
        Dim inputFileName As String = "ClientDrawing.dwg"
        Dim outputFileName As String = "Picture.png"

        <CommandMethod("SendToRender")>
        Public Sub SendToRenderer()
            Dim time1 As DateTime = DateTime.Now
            SaveCurrentDrawingToFile(inputFileName)
            ConnectTCP()
            SendStreamToServer(inputFileName)
            ReceiveStreamFromServer(outputFileName)
            DisconnectTCP()
            Dim timer As TimeSpan = DateTime.Now - time1
            editor.WriteMessage(vbLf + "Slika se nahaja: " + ClientFolder + vbLf +
timer.TotalSeconds.ToString)
        End Sub

        Private Sub ConnectTCP()
            Try
                tcpClient = New System.Net.Sockets.TcpClient()
                tcpClient.Connect(ipServer, portNum)
            Catch ex As System.Exception
                editor.WriteMessage("Streznik ne obstaja ali pa je zaseden. Poskusite
kasneje " + ipServer.ToString)
            End Try
        End Sub

        Private Sub DisconnectTCP()
            tcpClient.Close()
        End Sub
```

```
        Private Sub SaveCurrentDrawingToFile(fileName As String)
            Dim doc As Document = Application.DocumentManager.MdiActiveDocument
            Dim db As Database = doc.Database
            Using db
                If Not Directory.Exists(ClientFolder) Then
                    Directory.CreateDirectory(ClientFolder)
                End If
                db.SaveAs(ClientFolder + fileName, False, DwgVersion.Current,
doc.Database.SecurityParameters)
            End Using
        End Sub

        Private Sub SendStreamToServer(fileName As String)
            Try
                Dim fileStream = New FileStream(ClientFolder + fileName, FileMode.Open,
FileAccess.Read)
                WriteStreamToTCP(fileStream)
                fileStream.Close()
            Catch ex As System.Exception
                editor.WriteMessage(ex.Message + vbLf + ex.StackTrace)
            End Try
        End Sub


        Private Sub ReceiveStreamFromServer(fileName As String)
            Try
                Dim responseStream = ReadStreamFromTCP()

                Dim filePath As String = ClientFolder + fileName
                If File.Exists(filePath) Then
                    File.Delete(filePath)
                End If
                Dim fileStream As FileStream = File.OpenWrite(filePath)
                responseStream.WriteTo(fileStream)
                fileStream.Close()
            Catch ex As System.Exception
                editor.WriteMessage(ex.Message + vbLf + ex.StackTrace)
            End Try
        End Sub


        Private Function ReadStreamFromTCP() As MemoryStream
            Dim readBuffer = New Byte(BUFFER_SIZE) {}
            Dim requestStream As NetworkStream = tcpClient.GetStream()
            Dim reader = New BinaryReader(requestStream)
            Dim streamLength As Integer = reader.ReadInt32()
            Dim responseStream = New MemoryStream()
            While streamLength > 0
                Dim lData As Integer = requestStream.Read(readBuffer, 0, BUFFER_SIZE)
                responseStream.Write(readBuffer, 0, lData)
                streamLength -= lData
            End While
            Return responseStream
        End Function
```

```
        Private Sub WriteStreamToTCP(stream As Stream)
            Dim reader = New BinaryReader(stream)
            Dim requestStream As NetworkStream = tcpClient.GetStream()
            Dim writer = New BinaryWriter(requestStream)
            Dim length As Integer = Convert.ToInt32(stream.Length)
            writer.Write(length)
            Dim buffer As Byte()
            Do
                buffer = reader.ReadBytes(BUFFER_SIZE)
                writer.Write(buffer)
            Loop While buffer.Length = BUFFER_SIZE
            writer.Flush()
            reader.Close()
        End Sub
    End Class
End Namespace
```

8.3. Izjava o avtorstvu

MA-RIN-MF *Obr. Izjava o avtorstvu zaključne naloge*

UNIVERZA NA PRIMORSKEM
UNIVERSITÀ DEL LITORALE / UNIVERSITY OF PRIMORSKA

FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE
FACOLTÀ DI SCIENZE MATEMATICHE NATURALI E TECNOLOGIE INFORMATICHE
FACULTY OF MATHEMATICS, NATURAL SCIENCES AND INFORMATION TECHNOLOGIES

Glagoljaška 8, SI – 6000 Koper
Tel.: (+386 5) 611 75 70
Fax: (+386 5) 611 75 71
www.famnit.upr.si
info@famnit.upr.si

UNIVERZA NA PRIMORSKEM
UNIVERSITÀ DEL LITORALE
UNIVERSITY OF PRIMORSKA

Titov trg 4, SI – 6000 Koper
Tel.: + 386 5 611 75 00
Fax.: + 386 5 611 75 30
E-mail: info@upr.si
http://www.upr.si

IZJAVA O AVTORSTVU ZAKLJUČNE NALOGE

Spodaj podpisani/a Domen Petrič, z vpisno številko 89081038,
vpisan/-a v študijski program Računalništva in informatike,
sem avtor/-ica zaključne naloge z naslovom:

Izvedba vmesnika za porazdeljeno in vzporedno upodabljanje 3D modelov

S svojim podpisom zagotavljam, da je predložena zaključna naloga izključno rezultat mojega lastnega dela. Prav tako se zavedam, da je predstavljanje tujih del kot mojih lastnih kaznivo po zakonu.

Soglašam z objavo elektronske verzije zaključne naloge v zbirki »Dela FAMNIT« ter zagotavljam, da je elektronska oblika zaključne naloge identična tiskani.

V Kopru, dne 19.9.2012 Podpis avtorja/-ice: Domen