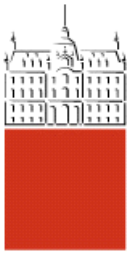
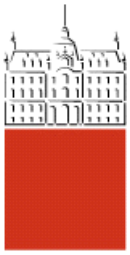
Univerza
v Ljubljani
Fakulteta
za gradbeništvo
in geodezijo



Jamova cesta 2
1000 Ljubljana, Slovenija
http://www3.fgg.uni-lj.si/

University
of Ljubljana
Faculty of
Civil and Geodetic
Engineering



Jamova cesta 2
SI – 1000 Ljubljana, Slovenia
http://www3.fgg.uni-lj.si/en/

Univerza
v Ljubljani

Fakulteta za
gradbeništvo in
geodezijo

Jamova 2
1000 Ljubljana, Slovenija
telefon (01) 47 68 500
faks (01) 42 50 681
fgg@fgg.uni-lj.si

UNIVERZITETNI ŠTUDIJ
PRVE STOPNJE
GRADBENIŠTVA

Kandidat:

JURE ČESNIK

PARAMETRIČNE DINAMIČNE KONSTRUKCIJSKE KOMPONENTE ZA PREFABRICIRANE AB ELEMENTE

Diplomska naloga št.: 2/B-GR

PARAMETRIC DYNAMIC STRUCTURAL COMPONENTS FOR PREFABRICATED REINFORCED CONCRETE ELEMENTS

Graduation thesis No.: 2/B-GR

Mentor:
doc. dr. Tomo Cerovšek

Predsednik komisije:
izr. prof. dr. Janko Logar

Član komisije:
prof. dr. Darko Beg

Ljubljana, 18. 09. 2012

STRAN ZA POPRAVKE

Stran z napako	Vrstica z napako	Namesto	Naj bo

IZJAVE

Podpisani Jure Česnik izjavljam, da sem avtor diplomske naloge z naslovom »Parametrične dinamične konstrukcijske komponente za prefabricirane AB elemente«.

Izjavljam, da je elektronska različica v vsem enaka tiskani različici.

Izjavljam, da dovoljujem objavo elektronske različice v repozitoriju UL FGG.

Ljubljana, 6.9.2012 Jure Česnik

BIBLIOGRAFSKO – DOKUMENTACIJSKA STRAN IN IZVLEČEK

UDK: **004:624.01:624.07(043.2)**
Avtor: **Jure Česnik**
Mentor: **doc. dr. Tomo Cerovšek**
Naslov: **Parametrične dinamične konstrukcijske komponente za prefabricirane AB elemente**
Tip dokumenta: **diplomska naloga – univerzitetni študij**
Obseg in oprema: **33 str., 2 pregl., 19 sl., 3 pril.**
Ključne besede: **parametrične dinamične komponente, BIM, CAD, prefabricirani elementi, SketchUp, vtičnik, računalniški model 3D**

Izvleček

Z vedno bolj razširjeno uporabo naprednih 3D modelirnikov ob računalniških programih za izdelavo gradbenih projektov, se med gradbenimi inženirji in arhitekti na tem področju gradbene informatike vedno bolj uveljavlja izdelava in uporaba parametričnih dinamičnih komponent, ki jih uporabljajo informacijski modelirniki stavb BIM (angl. *Building Information Modelling*).

Prvi del diplomske naloge podaja opis razvoja in uporabe BIM programov na področju parametričnih dinamičnih komponent s poudarkom na prefabriciranih konstrukcijskih elementih. Podrobneje je opisana tudi integracija komponent v uporabniških okoljih BIM in kako je ta integracija izvedena v različnih sistemih BIM. V nadaljevanju je bolj podrobno preučena notranja zgradba komponent in sistemi rabe komponent ter prednosti in slabosti modeliranja s komponentami v primerjavi s tradicionalnim postopkom modeliranja.

Drugi del naloge se poglobi v izdelavo parametričnih dinamičnih konstrukcijskih komponent in izdelavo podpornih vtičnikov, ki dopolnjujejo uporabo programa SketchUp, ki take funkcije do sedaj še nima izdelane. Zbirka izdelanih komponent armiranobetonskih konstrukcijskih prefabrikatov in vtičnikov je v nadaljevanju uporabljena za izdelavo trodimenzionalnega modela montažnega industrijskega objekta, na katerem so prikazane prednosti uporabe komponent (lažje, hitrejše, bolj natančno modeliranje). Na koncu so predstavljene še možnosti razvoja in prihodnosti uporabe konstrukcijskih komponent.

BIBLIOGRAPHIC – DOCUMENTALISTIC INFORMATION AND ABSTRACT

UDC: **004:624.01:624.07(043.2)**
Author: **Jure Česnik**
Supervisor: **Assist. Prof. Tomo Cerovšek, Ph. D.**
Title: **Parametric dynamic structural components for prefabricated reinforced concrete elements**
Document type: **Graduation Thesis – University studies**
Notes: **33 p., 2 tab., 19 fig., 3 ann.**
Key words: **parametric dynamic structural components, BIM, CAD, prefabriced elements, SketchUp, plug-in, computer 3D model**

Abstract

Advanced 3D modelling computer programs for authoring of building projects are gaining on popularity among civil engineers and architects and are thus developing with lightning speed. One of the main areas of development are parametric dynamic components and their use in BIM environments.

The first part of the presented work deals with the development and use of parametric dynamic components in BIM software, focusing on prefabricated construction elements. It also explains how such components are integrated into the BIM environment and what types of approach are used by different BIM packages. Futhermore, it focuses on the inner composition of components, systems of their usage and the description of both advantages and disadvantges of their usage.

The second part of the graduation work dives deeper into the creation of parametric dinamic construction components and development of plug-ins for 3D modelling software SketchUp, which does not yet include such a set of parametric components. We created a database of components for prefabricated concrete elements and SketchUp plug-ins, which are then used to create a 3D model of an industrial facility, that serves as an example for all the advantages of component use (easier, faster and more precise modelling). The work concludes with a presentation of possible future development of component usage.

ZAHVALA

Za pomoč in nasvete pri izdelavi diplomske naloge se iskreno zahvaljujem mentorju doc. dr. Tomu Cerovšku.

Posebna zahvala gre tudi mojima staršema, bratu in Anji, ki so verjeli vame in mi v težkih trenutnih z veliko mero razumevanja stali ob strani.

Nenazadnje bi se rad zahvalil še sošolcem, s katerimi sem skupaj prejadral zadnja tri leta tako v sončnem kot tudi nevihtnem vremenu.

KAZALO VSEBINE

KAZALO PREGLEDNIC

KAZALO SLIK

OKRAJŠAVE IN SIMBOLI

AEC	angl. Architectural Service Industry, slo. industrija arhitekturnih uslug
API	angl. Application Programming Interface, slo. programski vmesnik aplikacije
BIM	angl. Building Information Modeling, slo. informacijsko modeliranje stavb
CAD	angl. Computer Aided Design, slo. računalniško podprto načrtovanje
DC	angl. Dynamic Component, slo. dinamična komponenta
GDL	angl. Geometric Description Language, slo. geometrijsko opisni jezik
UI	angl. User Interface, slo. uporabniški vmesnik
UML	angl. Unified Modeling Language, slo. poenoten jezik modeliranja

1 UVOD

Načrti za gradbene projekte so že več stoletij nepogrešljiv izdelek, od kakovosti katerih je v veliki večini odvisna kakovost izvedenega projekta. Izdelava načrtov je s prihodom računalniške tehnologije in vedno bolj dostopnih osebnih računalnikov že pred desetletji presegla le risanje po papirju ter se neverjetno hitro prilagodila najprej dvodimenzionalnim risarskim računalniškim programom, nato pa še trodimenzionalnim programom za modeliranje.

Prvi koncept, ki je v preteklosti omogočil uporabo risarskih programov, je računalniško podprto načrtovanje (angl. *Computer Aided Design*, kratica *CAD*). Njegova slabost je, da je v večini le nadomestil papir z računalniškim modelom, saj za svoje delovanje uporablja le osnovne geometrijske elemente (črte, liki, telesa). Sledil mu je nadgrajeni koncept informacijskih modelov stavb (angl. *Building Information Modeling*, kratica *BIM*), ki za svoje delovanje ne uporablja le geometrijskih elementov, temveč le-te združi v skupine oziroma objekte modela, jih obogati z dodatnimi informacijami in s tem ustvari objektno orientirane komponente. Ravno po zaslugi slednjih je koncept BIM v primerjavi s konceptom CAD bolj napreden, prav tako pa se je delo z modeli stavb zaradi uporabe komponent spremenilo za vedno.

1.1 Opis problema

Z vedno večjo uporabo programov BIM se pojavlja tudi vedno večja potreba po razširjeni funkcionalnosti teh programov, saj s tem povečamo hitrost in natančnost izdelave reprezentativnega modela bodočega objekta. Večja natančnost pa vodi k zmanjšanemu številu napak in stroškov izdelave gradbenega objekta samega. Hitrost izdelave takih modelov je v veliki večini odvisna od števila in univerzalnosti konstrukcijskih, fasadnih in ostalih komponent, zato je izdelava le-teh ključna za dobro uporabo BIM programov.

Najbolj pomembne izmed vseh tipov komponent so ravno parametrične konstrukcijske komponente, saj ustvarijo geometrijsko osnovo za celotno nadaljnjo modeliranje, zato je dobra zbirka takih komponent ključna za uspešnost okolja BIM. Največjo zmogljivost dosežemo s tem, da poleg same parametrizacije komponentam dodamo še dinamičnost, to je zmožnost dinamičenga spreminjanja parametrov, in s tem ustvarimo parametrične dinamične konstrukcijske komponente.

Primarni cilj diplomske naloge je podrobneje preučiti, kako so take komponente sestavljene, kako delujejo in kakšna je njihova uporaba v okolju BIM. Sekundarni cilj pa je prikaz vključitve konstrukcijskih komponent v program za modeliranje, ki le-teh do sedaj izrecno še ne vključuje.

1.2 Metodologija

Izdelava diplomske naloge je zahtevala preučitev koncepta parametričnih dinamičnih komponent in kako se z njihovo izdelavo in uporabo spopadajo različna programska orodja, ki so na razpolago (SketchUp, ArchiCAD, Revit, Tekla...) ter učenje izdelave parametriziranih dinamičnih komponent v enem izmed teh programov. Med vsemi temi ponudniki sem se za izdelavo in uporabo odločil izbrati program SketchUp, saj je enostaven za uporabo, ima velik potencial za razvoj, ker je odprto koden, in še nima razvite zbirke konstrukcijskih komponent, čeprav ima možnost izdelave dinamičnih komponent. S tem sem poskusil prikazati ne le samo izdelavo komponent, temveč tudi izdelavo nadgradnje programa z zbirko teh komponent in vtičniki, ki optimizirajo njihovo uporabo.

Poleg samega postopka izdelave dinamičnih komponent v programu SketchUp sem se naučil tudi programskega jezika Ruby, saj izdelava vtičnikov za ta program poteka ravno v tem jeziku, in poenotenega jezika modeliranja (angl. *Unified Modeling Language*, kratica *UML*), s katerim so opisani sestava in sistemi rabe komponent.

1.3 Struktura naloge

Diplomska naloga je sestavljena iz štirih glavnih sklopov, ki jih pokriva šest poglavij vključno z uvodnim. Na začetku obrazložen razvoj okolja BIM, nato pa še splošen razvoj komponent in uporaba komponent v BIM programih. V nadaljevanju je podrobneje preučena notranja sestava komponente, opisane so zahteve, ki jih mora taka komponenta izpolnjevati, sistem izdelave in uporabe parametričnih dinamičnih komponent ter prednosti in slabosti modeliranja s komponentami v primerjavi s tradicionalnim pristopom k modeliranju.

Uporaba tega osnovnega znanja je prikazana v četrtem in petem poglavju z implementacijo sistema konstrukcijskih komponent v program SketchUp. Le-ta je izvedena s pomočjo dinamičnih komponent (angl. *Dynamic Components*, kratica *DC*) prefabriciranih konstrukcijskih elementov in programskih Ruby vtičnikov. Poleg tega diplomska naloga vključuje še kratka navodila za izdelavo enostavne dinamične komponente in programskega vtičnika ter kako se taka orodja uporabi na realnem primeru montažnega industrijskega objekta. V zaključku so opisane še možnosti nadaljnjega razvoja BIM programov na področju komponent in njihove uporabe v stroki.

2 OKOLJA BIM

2.1 Razvoj BIM-a

V šestdesetih let se je v računalniškem svetu razvil koncept CAD, ki je še sedaj eden najbolj uspešnih in revolucionarnih konceptov na področju gradbene informatike. Slabost tega koncepta je njegov osnovni pristop k modeliranju, to je da le nadomestimo papir z računalniškim modelom, gradniki modelov pa so ostali na nivoju geometrije brez dodatnih informacij. S takim pristopom se je sicer v preteklosti proces izdelave dokaj pospešil, a sta se ohranila problema ročnega modeliranja in bogatenja objektov z informacijami. Poleg se je ohranil tudi problem medsebojnega sodelovanja različnih strok, saj CAD ne nudi skupne platforme, znotraj katere bi lahko delovali vsi udeleženci izdelave projekta od razvoja do končne izvedbe.

Dobrih dvajset let po prvih korakih koncepta CAD se je v namene gradbeništva in arhitekture razvilo novo mišljenje imenovano industrija arhitekturnih uslug (angl. *Architectural Services Industry*, kratica *AEC Industry*), katerega vizija je bila povezati vse usluge arhitekture in gradbeništva z enim samim razvojnim okoljem. S takim razmišljanjem se je iz koncepta CAD hitro razvil nadgrajen koncept BIM, ki je trodimenzionalno geometrijo nadgradil z novimi negeometrijskimi informacijami in je za svoje delovanje kot prvi uporabil informacijsko obogatene objekte. Taki objekti imajo poleg trodimenzionalne geometrije vključene še dodatne sklope informacij oziroma dimenzije kot so čas, stroški, parametri, vezi in v vedno večjem obsegu tudi informacije proizvajalcev. Poleg dopolnitve geometrije z informacijami je namen BIM modelov tudi skupna kolaboracija vseh udeležencev izdelave projekta na isti platformi, ki obsega vsa področja in nudi celosten pogled na projekt brez menjave programskega okolja.

2.2 Razvoj komponent v BIM-u

Modelirni predmeti, ki se jih uporablja v BIM programih, se delijo na dve podskupini: objekti izdelani s strani uporabnika, ki so vsak zase unikatni, in vnaprej pripravljeni objekti oziroma komponente. Komponenta je definirana kot parametričen modularen del računalniškega modela, ki je venomer ponovljiv in se lahko preko uporabniškega vmesnika ali medsebojnih povezav dinamično spreminja. Glavna značilnost komponent je, da se v celotnem modelu ponovijo večkrat v različnih oblikah, ravno zaradi tega pa je v osnovi njihova izdelava veliko bolj zahtevna. Ravno po zaslugi komponent je koncept BIM zelo pohitril in poenostavil izdelavo računalniških modelov stavb. Ta diplomska naloga

v celoti pokriva področje komponent v okoljih BIM in poskusi kar najbolj nazorno prikazati njihovo sestavo in rabo.

Čeprav se je BIM razvil šele proti koncu osemdesetih let (ArchiCAD, 1987) je bila ideja za parametrične komponente v svoji osnovni, primitivni obliki razvita že v okolju CAD nekaj let prej vzporedno z razvojem t.i. polnih teles (angl. *true solids*). Pravi razvoj parametričnih komponent, kot jih poznamo danes, se je pričel v začetku devetdesetih let in se je nadaljeval z mnogimi razširitvami do danes. Največja razširitev je bila vključitev vezi med določenimi komponentami, ki omogoči spremembo geometrije vezane komponente zgolj s spremembo geometrije obravnavane komponente, brez da bi uporabnik vezano komponento karkoli spreminjal.

Zaradi takega pristopa in zaradi vnaprej izdelane osnovne geometrije, ki jo le prilagajamo potrebam, je uporaba parametričnih dinamičnih komponent hitro postala zelo priljubljena, saj je tak pristop veliko bolj optimiziran kot tradicionalen pristop k modeliranju. Največja slabost komponent je dokaj zapleten postopek izdelave takih komponent, a ko je enkrat komponenta narejena, se čas izdelave modelov, ki vključujejo to komponento, drastično zmanjša.

2.3 Uporaba komponent v BIM-u

Komponente, ki so vgrajene v okolja BIM, so lahko najrazličnejših vrst in vključujejo vse od konstrukcijskih elementov do elementov stavbnega pohištva in dvodimenzionalnih inženirskih oznak. Z razvojem koncepta BIM se je na svetu razvilo veliko programov, ki uporabniku nudijo delo z najrazličnejšimi komponentami, vse od svetovno uveljavljenih pa do manjših, brezplačnih programov. Čeprav je osnova celotnega sistema komponent opisana kasneje v tretjem poglavju, se bom v nadaljevanju osredotočil na nekaj najbolj znanih BIM programov in za vsakega posebej na kratko opisal, kakšna je njegova raba in pristop na področju komponent. Za lažjo primerjavo različnih sistemov komponent, ki jih ti programi uporabljajo, je na koncu poglavja v Preglednica 1 izdelan kratek pregled glavnih značilnosti teh sistemov.

ArchiCAD. Graphisoft, podjetje iz sosednje Madžarske, se lahko ponaša z naslovom pionirja BIM aplikacij, saj je kot prvo CAD svetu postavilo izziv s svojim BIM programom ArchiCAD. Le-ta komponente uporablja za parametrične konstrukcijske komponente (stene, stebri, nosilci...) in objekte ustvarjene z geometrijsko opisnim jezikom (angl. *Geometric Description Language*, kratica *GDL*), ki so napisani na osnovi programskega jezika BASIC. Prvi tip komponent je integriran v sam program in je sestavljen iz večih kategorij, vsaka kategorija pa ima svoje orodje za uporabo njenih komponent.

ArchiCAD podpira osnovno uporabo vezanih komponent, saj se konstrukcijske komponente dinamično odzivajo na svojo okolico preko uporabe stopenj prioritete.

Drugi tip objektov, to so GDL objekti, ki večinoma pokrivajo področja prefabrikatov, stavbnega pohištva in inženirskih oznak, je shranjen v zunanjih knjižnicah na pomnilniku ali medmrežju, od kjer jih program prebere ter pozicionira instance objektov v prostor modela. Prednosti ArchiCAD-a sta njegov inovativen in zmogljiv uporabniški vmesnik (angl. *User Interface*, kratica *UI*), ki ga je mogoče po želji modificirati, in možnost parametrične definicije ne samo trodimenizionalne predstave komponente, temveč tudi njenega dvodimenzionalnega simbola.

Allplan. Nemško podjetje Nemetschek, ki je leta 2008 prevzelo ArchiCAD, ima na BIM področju lasten proizvod pod imenom Allplan, ki pa komponente uporablja večinoma le za modeliranje armiranobetnoskih prefabrikatov in se je šele letos podal v vode pravih parametričnih komponent vseh vrst z dodatkov SmartParts. Komponente prefabrikatov so integrirane v samo delovanje Allplan-a in so ustvarjene na enak način kot sam program, medtem ko se bodo nove komponente lahko po želji modificirale in ustvarjale s posebnim poenostavljenim programskim jezikom SmartPart Script, ki ga je ravno v ta namen razvil Nemetschek.

Revit. Revit je 3D modelirni program, katerega razvoj je temeljil na mišljenju AEC industrije in ki ga je leta 2002 odkupil Autodesk ter ga dogradil v pravo okolje BIM. Njegova uporaba parametričnih komponent je zelo podobna pristopu uporabljenen v ArchiCAD-u, saj so vse komponente združene v različne družine (angl. *families*). Od njega pa se razlikuje po tem, da je pristop do vseh komponent ne glede na družino enak. To pomeni, da so vse družine eksterno shranjene na disku ali medmrežju in da je mogoče vse komponente spreminjati in ustvariti na enak način. Poleg tega omogoča Revit zelo dobro definiranje tro- in dvodimenzionalnih predstav komponent ter njihovo stopnjo detajlov glede na merilo in tip načrta. Slabost Revit-ovih komponent je v tem, da jih ni mogoče enostavno programirati in so načeloma omejene na funkcionalnost vgrajenega uporabniškega vmesnika.

Tekla. Program Tekla je bil razvit leta 2000 v podjetju Tekla Coropration in je od takrat postal eden najbolj uspešnih BIM programov, ki uporablja parametrične dinamične konstrukcijske komponente, še posebej na področjih jeklenih in armiranobetnoskih konstrukcij. Komponente obravnava čisto drugače kot večina ostalih BIM programov, ker uporablja vnaprej parametrično izdelane gradnike in jih nato sestavlja v najrazličnejše komponente. Največje prednosti takega pristopa so možnost uporabe vseh možnih kombinacij gradnikov preko uporabniškega vmesnika, zelo velika dinamičnost in povečana uporaba vezanih komponent, kar nam da zelo detajlne modele konstrukcije in je s tega vidika najbolj

dovršen program na trgu. Po drugi strani pa je izdelava novih sestavnih gradnikov zelo kompleksna naloga in je uporabnik dokaj omejen v smislu izdelave svojih komponent iz niča.

MicroStation. Podjetje Bentley je dokaj hitro uvedlo parametrične komponente v svoj program MicroStation, čeprav je ta temeljil na konceptu CAD. Uvedlo jih je s pomočjo generativnih komponent (angl. *Generative Components*), ki pa se za razliko od ArchiCAD-a, katerega komponente večinoma temeljijo na parametrizaciji pravilnih geometrijskih oblik, bolj nagibajo k uporabi čisto parametriziranih komponent, ki temeljijo na matematičnih funkcijah in operacijah.

Generativne komponente se lahko ustvarijo na tri različne načine: geometrijsko (geometrijske operacije na osnovnem modelu), skriptno (s pomočjo prirejenega C programskega jezika imenovanega GCScript) ali programsko (z implementacijo čiste programske kode jezika C#). Najbolj zahtevne komponente nastanejo s hibridno uporabo vseh treh načinov, ker ima vsaka svoje prednosti in slabosti.

Preglednica 1: Primerjava sistemov komponent v različnih programih, ki delujejo v okolju BIM

BIM program	Sistem komponent	Uporabljen jezik	Glavne posebnosti
ArchiCAD	Dva sistema:		- uporaba razredov - delna uporaba vezanih komponent
	konstrukcijske komponente	C++	
	GDL komponente	GDL, BASIC	
Allplan	Dva sistema:		- možnost dinamične izdelave modelov armature znotraj komponent
	komponente armiranobetonskih prefabrikatov	C++	
	SmartParts	SmartPart Script	
Revit	konstrukcijske in arhitekturne komponente v istem sistemu	C++	- uporaba družin - predstave komponent vezane na stopnjo natančnosti
Tekla	konstrukcijske komponente sestavljene iz nabora gradnikov	C#	- izboljšana uporaba vezanih komponent
MicroStation	Generative Components	GCScript in C#	- možnost kombiniranja različnih načinov izdelave komponent

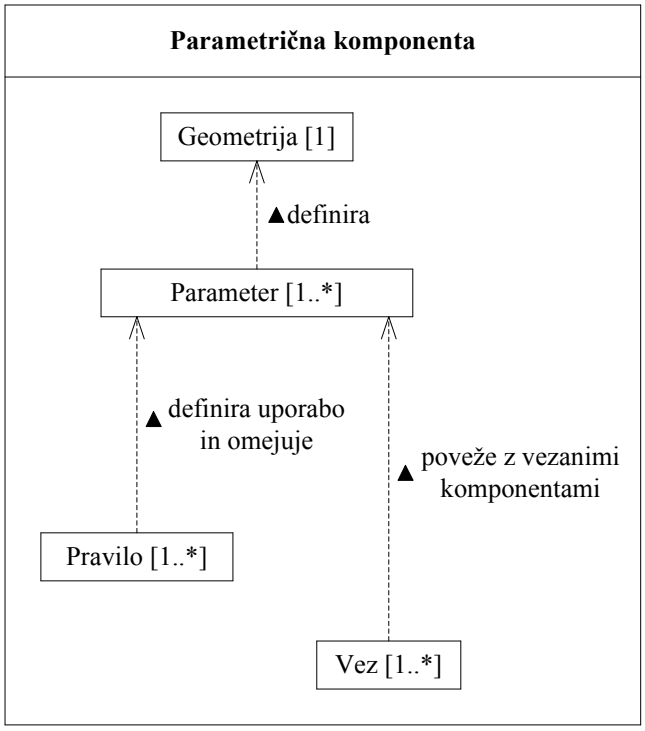
3 SISTEM RABE KOMPONENT

V tem poglavju je opisano delovanje celotnega sistema rabe komponent. Ne glede na to, da vsak izmed prej opisanih BIM programov uporablja komponente na svoj način, mora interni sistem še vedno slediti določenim smernicam sestave, izdelave in uporabe komponent, brez katerih komponent ne morejo obstajati.

3.1 Sestava komponent

Na Slika 1 je z UML diagramom kompozitne strukture prikazana notranja sestava parametrične komponente, kjer lahko vidimo, da je geometrija komponente, ki je ena sama, definirana in odvisna od vsaj enega parametra komponente. Parametri so z geometrijo povezani na podlagi notranjih pravil, ki definirajo, kako naj se komponenta obnaša v prostoru računalniškega model in jo omejujejo.

Poleg pravil so parametri lahko odvisni tudi od ene ali več vezi, ki te parametre povežejo s t.i. vezanimi komponentami oziroma bolj natančno s parametri vezanih komponent. Te vezi spremljajo spremembe komponente in na podlagi le-teh izvajajo spremembe parametrov vezanih komponent.



Slika 1: Diagram notranje sestave parametrične komponente

3.2 Zahteve za komponente

Samo znanje, kako so komponent zgrajene, za njihovo učinkovito rabo nikakor ni dovolj, zato moramo poznati, kakšne so zahteve za to zgradbo in njene elemente. Poleg tega pa si moramo zastaviti tudi zahteve za rabo komponent, to je izdelavo in uporabo, ki sta zelo pomembna sistema znotraj implementacije komponent. Če hočemo, da je ta implementacija uspešna, mora vsaka parametrična dinamična komponenta slediti naslednjimi zahtevam:

- **Parametričnost.** Da lahko za komponento rečemo, da je parametrična, mora vsebovati vsaj en parameter (notranji atribut geometrije komponente). Če parametra ni, potem imamo opravka zgolj z navadnim objektom in ne s pravo komponento.

- **Povezava parametrov in geometrije.** Za vsak parameter mora obstajati vsaj eno pravilo, ki ga poveže z geometrijo. Če ni pravil, potem se sprememba parametra ne odraža na spremembi geometrije, saj ravno pravila ustvarijo povezavo med parametri in geometrijo, kar pomeni, da so ta pravila obvezen element parametričnih komponent.

- **Dinamičnost.** Za doseg dinamičnosti je potreben uporabniški vmesnik ali grafična orodja, s katerimi je mogoče spreminjati vrednosti parametrov komponete z normalnim znanjem uporabe modelirnega programa. Dodatno dinamičnost dosežemo z uporabo vezi, če le-te omogočajo, da se sprememba parametra trenutno aktivne komponente odraža v spremembi vsaj enega parametra vezane komponente. Vezi za razliko od pravil niso obvezne, a zelo izboljšajo delovanje komponent, saj je v nasprotnem primeru nivo dinamičnosti dokaj nizek, ker lahko spremembo parametrov izvede le uporabnik preko uporabniškega vmesnika.

- **Univerzalnost.** Celoten namen komponent je njihova večkratna uporaba v različnih projektih z različnimi parametri, kar pa je mogoče le če so komponente izdelane univerzalno. S tem je mišljeno, da niso izdelane za zadostitev potreb enega samega projekta, temveč je njihova uporaba nemoteno omogočena v vseh potencialnih projektih. To zagotovimo z uporabo dobro definiranih pravil obnašanja in parametrov, ki upoštevajo vse možne situacije, v katerih se bo komponenta uporabljala. Če imamo na primer komponento točkovnega temelja, mora biti ta dovolj univezalna, da se lahko ne glede na različne dimenzije uporabi v različnih oblikah v različnih projektih.

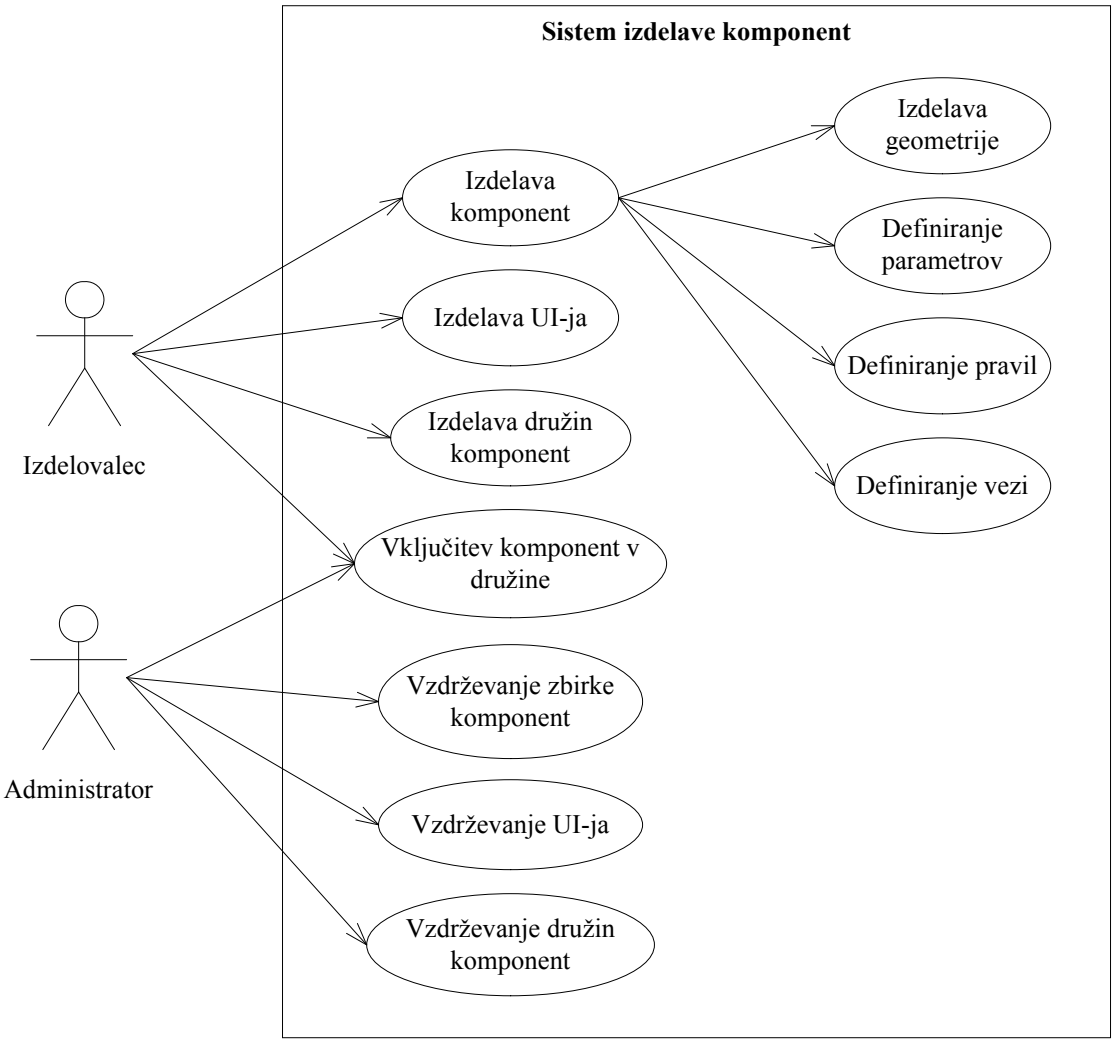
- **Reprezentativnost.** Zavedati se moramo, da je v komponente skoraj nemogoče prenesti celotno sliko realnega stanja, zato je nujno potrebno izdelati take komponente, da

zadovoljujejo nivoju zahtevane natančnosti modela, a ga ne presegajo. S tem je ohranjen velik del enostavnosti komponent in posledično zmanjšana potreba po računski moči.

3.3 Sistema rabe komponent

3.3.1 Sistem izdelave vzdrževanja komponent

Proces izdelave in implementacije vsake komponente je ne glede na program v osnovi enak in je prikazan na Slika 2 s primerom uporabe, prikazanim z diagramom UML. Celoten sistem je zgrajen iz komponent in družin komponent, v katere združujemo po namenu, uporabi ali funkciji podobne komponente. Za obstoj in optimizirano uporabo komponent potrebujemo izdelovalce, ki izdelajo komponente, programski vmesnik za uporabo komponent in same komponente vključijo v izdelane družine. Poleg izdelovalcev potrebujemo tudi administratorje, ki sistem vzdržujejo in nadzorujejo ter ga po potrebi tudi prilagajajo uporabniku.



Slika 2: Diagram sistema izdelave in vzdrževanja komponent

Ko razmišljamo o vzpostavitvi parametrov, pravil in vezi, moramo vedno imeti v mislih zahteve komponent in splošne smernice izdelave komponent. V principu to pomeni, da mora formulacija notranjih pravil slediti vedno istim smernicam, saj le s tem omogočimo poenoten način izdelave in uporabe vseh komponent.

3.3.2 Sistem uporabe komponent

Kot že prej rečeno so komponente in njihova uporaba ene izmed najboljših kvalitet BIM programov, zavedati pa se moramo, da sama zbirka komponent in razdelitev po družinah ni dovolj. Komponente morajo biti v BIM program integrirane brez vidnega prehoda, saj se s tem zelo poenostavi tako uporaba kot tudi poveča hitrost dela s komponentami. Da pa lahko izvedemo tako integracijo moramo razumeti, kaj vse lahko uporabnik dela s komponentami. Na Slika 3 je prikazan sistem uporabe komponent z diagramom primera uporabe, ki prikazuje možne operacije nad komponentami, ki jih lahko izvaja uporabnik.

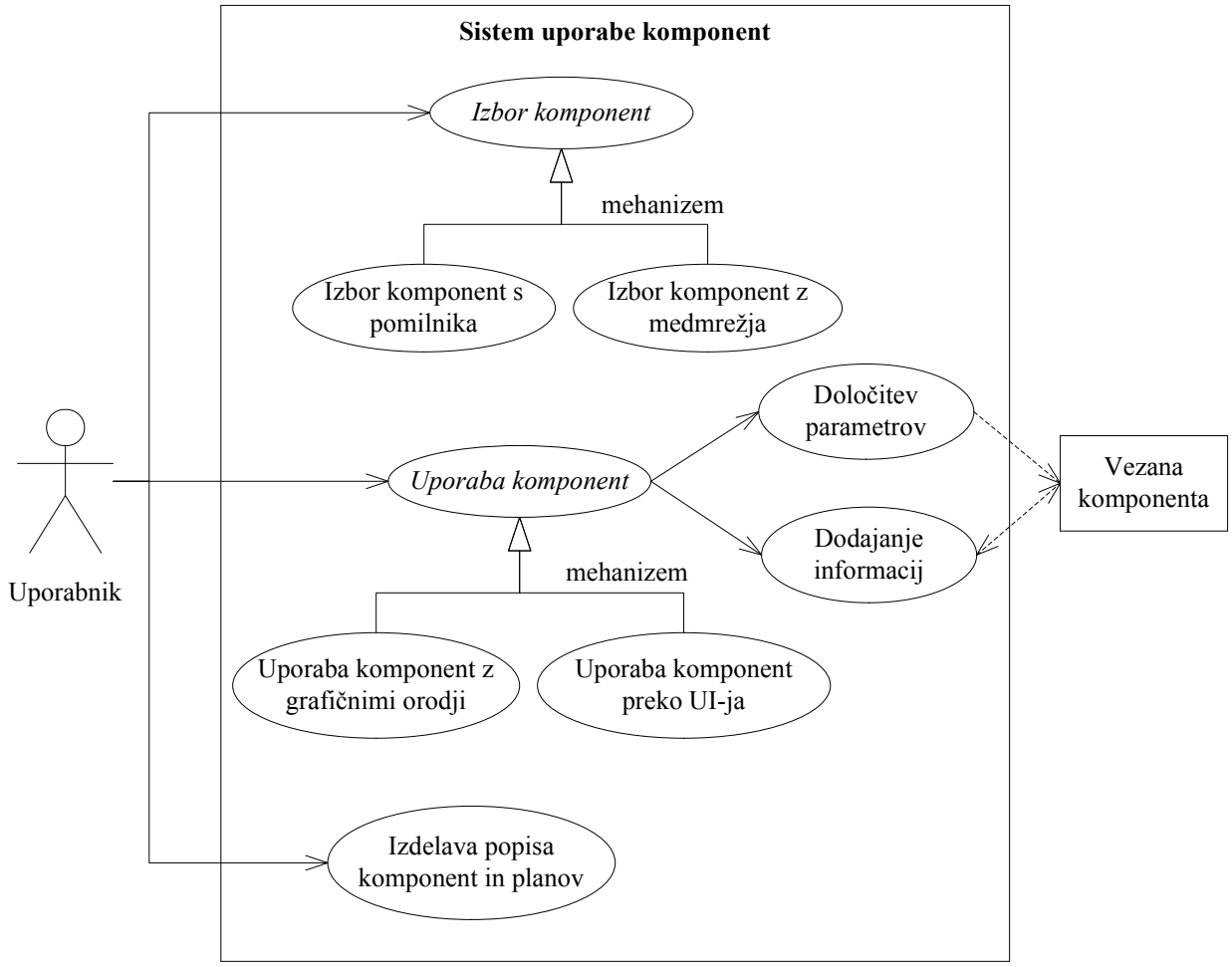
Ta sistem je ločen od sistema izdelave komponent, čeprav se na prvi pogled pri večini programov zdi, da lahko uporabnik sam izdeluje komponente. To sicer drži, a pri tem uporabnik programa izgubi funkcijo uporabnika komponent in postane izdelovalec komponent. Zavedati se moramo razlike med vlogami izdelovalca, ki razvija in uporabnika, ki izključno uporablja. S tako ločitvijo lahko hitro razberemo, da je pristop do komponent pri uporabniku zelo drugačen, ker le-tega ne zanima notranja sestava in delovanje komponent, temveč izključno hitra, učinkovita in enostavna uporaba komponent. Zato lahko na Slika 3 opazimo, da uporabnik izvaja na tem področju tri operacije, to so izbor komponent, uporaba komponent in izdelava popisa komponent in planov, ki so ločene od sistema izdelave komponent.

Izbor komponent je v večini osnovnih BIM programov možen le, če je zbirka komponent naložena na pomnilnik v točno določeni datoteki, ki jo program uporabi, da uporabniku ponudi celoten nabor. Bolj napredni programi imajo možnost povezave v računalniške mreže in celo povezave preko interneta na oddaljene strežnike, kjer je možen izbor komponent, izdelanih s strani različnih izdelovalcev, ali pa komponent ponudnika programa, ki se še razvijajo in dopolnjujejo. V vsakem primeru pa je uporabnik tisti, ki mora določeno komponento izbrati, program mu lahko le predlaga družino komponent, ki je za določeno funkcijo najbolj primerna.

Po izboru komponente pride na vrsto uporaba komponente, ki se navadno izvede preko uporabniškega vmesnika, kjer uporabnik ročno vnese točno določene parametre. Ker so BIM programi v osnovi še vedno risarski programi, je možna tudi uporaba komponent z grafičnimi orodji, s katerimi posredno spreminjamo parametre. V splošnem velja, da je uporaba grafičnih orodij lažja, uporaba

uporabniškega vmesnika pa bolj natančna. Ti parametri so preko vezi lahko vezani na parametre vezanih komponent, ki se dinamično spreminjajo in prilagajajo glede na uporabnikov vnos (parametri vezanih komponent so odvisni od parametrov komponente, katero uporabnik trenutno obdeluje). Poleg parametrov uporabnik vnese tudi dodatne informacije, za kar pa je nujno potreben uporabniški vmesnik. Posebnost je, da izmenjava teh informacij preko vezi lahko teče v smeri od uporabljene proti vezanim komponentam in obratno.

Uporabnik navadno na koncu uporabi še interna orodja programa za izdelavo popisov komponent in izdelavo terminskih, finančih in ostalih planov, ki pomagajo optimizirati gradnjo.



Slika 3: Diagram sistema uporabe komponent

3.4 Prednosti in slabosti uporabe komponent

Z vedno večjo uporabo komponent na področju izdelave računalniških modelov stavb se pojavlja tudi vedno večja razlika med modeliranjem na tradicionalen način (ročno izdelavo modela) in novim

načinom z uporabo komponent. Vsak način ima svoje prednosti in slabosti ter je optimalen za uporabo na določenih tipih konstrukcij stavb, kar je lepo razvidno v Preglednica 2.

Preglednica 2: Značilnosti in razlike med tradicionalnim načinom izdelave modela in načinom izdelave z uporabo komponent

Način izdelave	Tradicionalen/ročen	Uporaba komponent
Univerzalnost	Izdelamo lahko vse.	Uporabimo lahko le podloge, ki pa niso vsesplošno univerzalne.
Način začetne izdelave	Le izdelava geometrije.	Izdelava geometrije, dopolnitev s parametri in omejitev s pravili.
Zahtevnost začetne izdelave	Odvisna od geometrije.	Odvisna od geometrije in stopnje univerzalnosti.
Čas začetne izdelave	Potrebujemo manj časa.	Potrebujemo več časa.
Zahtevnost uporabe	Večja, saj moramo vedno ročno izrisati oziroma spremeniti.	Majhna, saj je potrebna le ročna postavitev, uvedba sprememb je polavtomatska.
Čas uporabe	Potrebujemo veliko časa.	Potrebujemo malo časa.
Pripis dodatnih informacij	Zahtevnejši.	Enostavnejši, že integriran.
Priporočena uporaba	Unikatne konstrukcije s posebnimi oblikami.	Konstrukcije, ki se izdelajo z uporabo prefabricianih polizdelkov.
Možnost nadaljnjega razvoja	Majhna, le izdelava novih orodij ročnega modeliranja.	Velika, saj so komponente vedno bolj zapletene in dovršene, njihovo medsebojno sodelovanje pa večje.

S tem v mislih primerjajmo čas izdelave modela montažne hiše v okolju CAD na tradicionalen način in te iste hiše v okolju BIM z uporabo parametričnih komponent. Opazimo, da je delo veliko prej opravljeno v okolju BIM, saj je komponente potrebno le postaviti in definirati, medtem ko je v okolju CAD potrebno celotno strukturo izrisati iz niča. V kolikor imamo opravka z montažnimi, ponavljajočimi, neunikatnimi gradbenimi elementi je delo s komponentami edina smiselna rešitev, saj je geometrija elementov definirana s točno določenimi pravili. Nasprotno pa je s komponentami skoraj nemogoče izdelati unikatne izdelke oziroma je delo z njimi nesmiselno, zato tu še vedno prevladuje CAD pristop. V gradbeništvu imamo večinoma opravka s konstrukcijami in stavbami, ki so sicer same po sebi unikatni (zunanji izgled, tip in razporeditev konstrukcije in prostorov), a so ti v veliki večini izdelani iz serijskih neunikatnih polizdelkov (prefabrikatov), zato je BIM pristop do izdelave modelov teh stavb veliko lažji in hitrejši.

4 IMPLEMENTACIJA KOMPONENT

V prejšnjem poglavju so opisani sestava, sistemi, slabosti in prednost ter uporaba komponent, v tem poglavju pa je predstavljena implementacija takega sistema v modelirni program, ki v le-tega še ne pozna in uporablja, to je program SketchUp. Tekom poglavja bomo spoznali uporabo njegovih dinamičnih komponent in funkcij, ker je v SketchUp okolju možno zelo nazorno prikazati, kako okolja BIM delujejo na področju komponent, čeprav sam ne deluje po poncipih BIM-a.

4.1 Osnovno o SketchUp-u

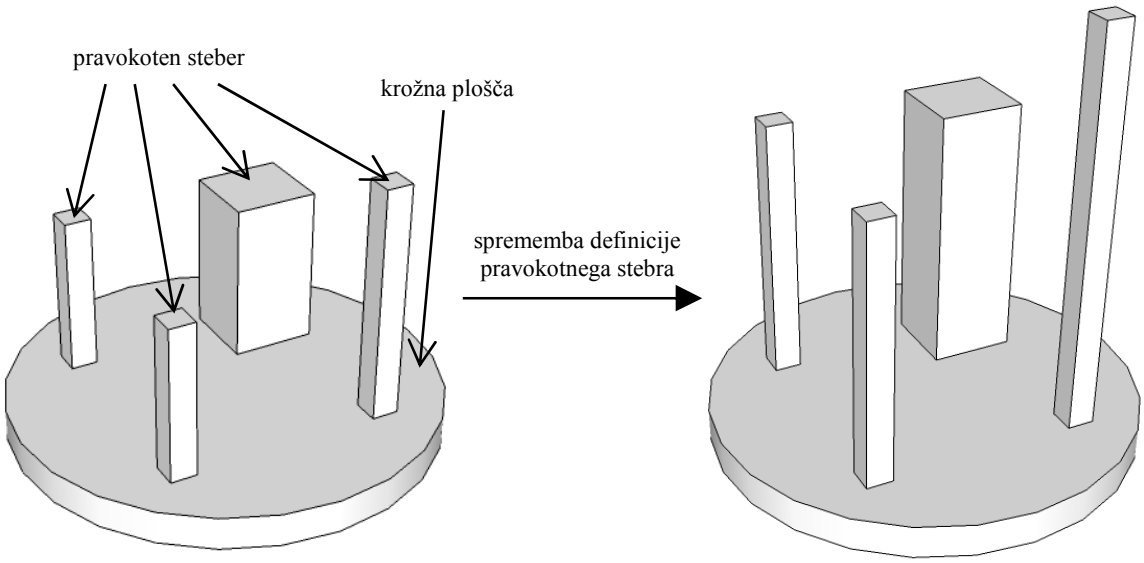
SketchUp je 3D modelirni program, ki je izšel leta 2000 izpod rok podjetja LastSoftware, leta 2003 pa je vajeti nadaljnjega razvoja v roke prevzel Google. Ta je pravice do programa letos spomladi prodal družbi Trimble, ki ima že kar nekaj izkušenj z razvojem programske opreme v namene gradbeništva (letos je družba poleg SketchUp-a prevzela še program Tekla). SketchUp že od samega začetka stremi k enostavni uporabi in zmožnosti prilagajanja novim situacijam s pomočjo vtičnikov, zato je zelo primeren kot uvodni program tako za učenje osnov trodimenzionalnega modeliranja, kot tudi za učenje izdelave vtičnikov, ki povečajo njegovo funkcionalnost.

Program je na voljo v dveh različicah: brezplačni in profesionalni, ki vključuje nekatere dodatne, napredne funkcije in dva dodatna programa (LayOut in Style Builder). Med dodatnimi funkcijami je tudi možnost izdelave dinamičnih parametriziranih komponent, ki je bila v izdelavi diplomske naloge najbolj uporabljena.

4.2 Dinamične komponente

4.2.1 Osnovno o SketchUp dinamičnih komponentah

Dinamične komponente so v SketchUp-u uporabljene na podlagi dveh pojmov: definicij in primerkov oziroma instanc. Definicija komponente je nosilka geometrije in pravil neke vrste komponente in je ne moremo fizično predstaviti, medtem ko je instanca fizična reprezentacija določene definicije komponente. Instance iste komponente se lahko med seboj razlikujejo, kar pomeni, da imajo različno določene parametre. Za lažjo predstavo poglejmo primer na Slika 4.



Slika 4: Prikaz razlike med definicijo in instanco neke komponente

Na levi polovici Slika 4 imamo štiri instance, ki imajo za definicijo pravokoten steber, in eno instanco z definicijo krožne plošče. Dve instanci pravokotnega stebra sta različni od ostalih instanc iste definicije, prav tako sta različni med seboj. V kolikor bi spremenili definicijo komponente pravokotnega stebra in stanje osvežili, bi dinamično spremenili vse instance, ki so osnovane na tej definiciji, kot je to razvidno na desni polovici Slika 4, kjer je višina definicije povečana za faktor dva.

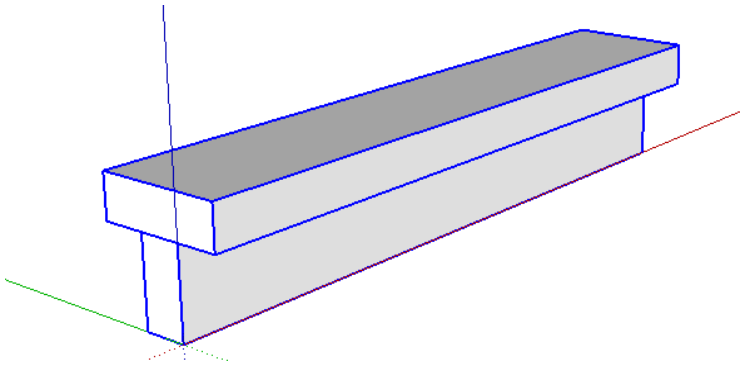
Če hočemo preprečiti spremembo neke instance s spremembo njene definicije, lahko to instanco naredimo unikatno (ukaz *Make Unique*), s čimer SketchUp ustvari novo definicijo komponente, ki temelji na tej instanci, in ji jo pripiše. Sprememba je razvidna pri unikatnem imenu definicije, saj se novo ustvarjenim definicijam po zgornjem postopku na koncu imena pripiše znak # in zaporedna številka, npr: Pravokoten steber → Pravokoten steber #1. S to definicijo lahko naprej operiramo enako kot z ostalimi prej obstoječimi definicijami.

Kot sem že omenil, profesionalna različica programa SketchUp dovoljuje izdelavo in prirejanje dinamičnih komponent, ki jim lahko pripišemo parametre, katere lahko določijo ali modificirajo uporabniki ali pa so z internimi funkcijami dinamično vezani na druge parametere. Za lažjo predstavo je recimo višina komponente nekega stebra lahko vezana na njegovo širino prečnega prereza ali obratno. Na žalost pa SketchUp ne dovoljuje povezave parametrov med dvemi ločenimi komponentami, to je ustvarjanja vezi med komponentami.

4.2.2 Primer izdelave dinamične komponente

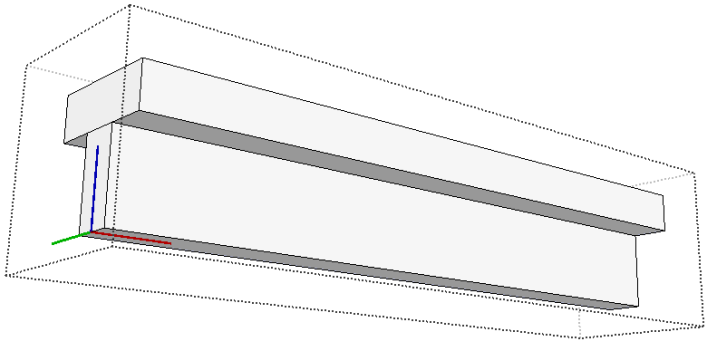
V nadaljevanju je opisan postopek izdelave dinamične komponente »T« nosilca, ki je definiran na podlagi dimenzij prečnega prereza in dolžine razpona. Če pogledamo nazaj na Slika 2, lahko vidimo, katere korake moramo pokriti v vlogi izdelovalca komponente. Prvi korak je izdelava osnovne geometrije sestavnih delov nosilca, kot sta stojina in zgornja pasnica, in vključitev vsakega osnovnega dela v posamične komponente z ukazom *Make Component* (Slika 5).

Pri tem je zelo pomembna pazljiva in premišljena definicija koordinatnega sistema vsake komponente, ki se v splošnem razlikuje od koordinatnega sistema modela.



Slika 5: Sestavne komponente T nosilca: Zgornja pasnica in Stojina

Komponenti združimo v eno starševsko komponento in njen koordinatni sistem pozicioniramo na mestu, kjer je to najbolj smiselno, v primeru nosilcev na sredino spodnjega roba stojine enega izmed koncev nosilca (Slika 6). Opozorilo: čeprav je komponenta deljena na več podkomponent, je geometrija celotne komponente ena sama.

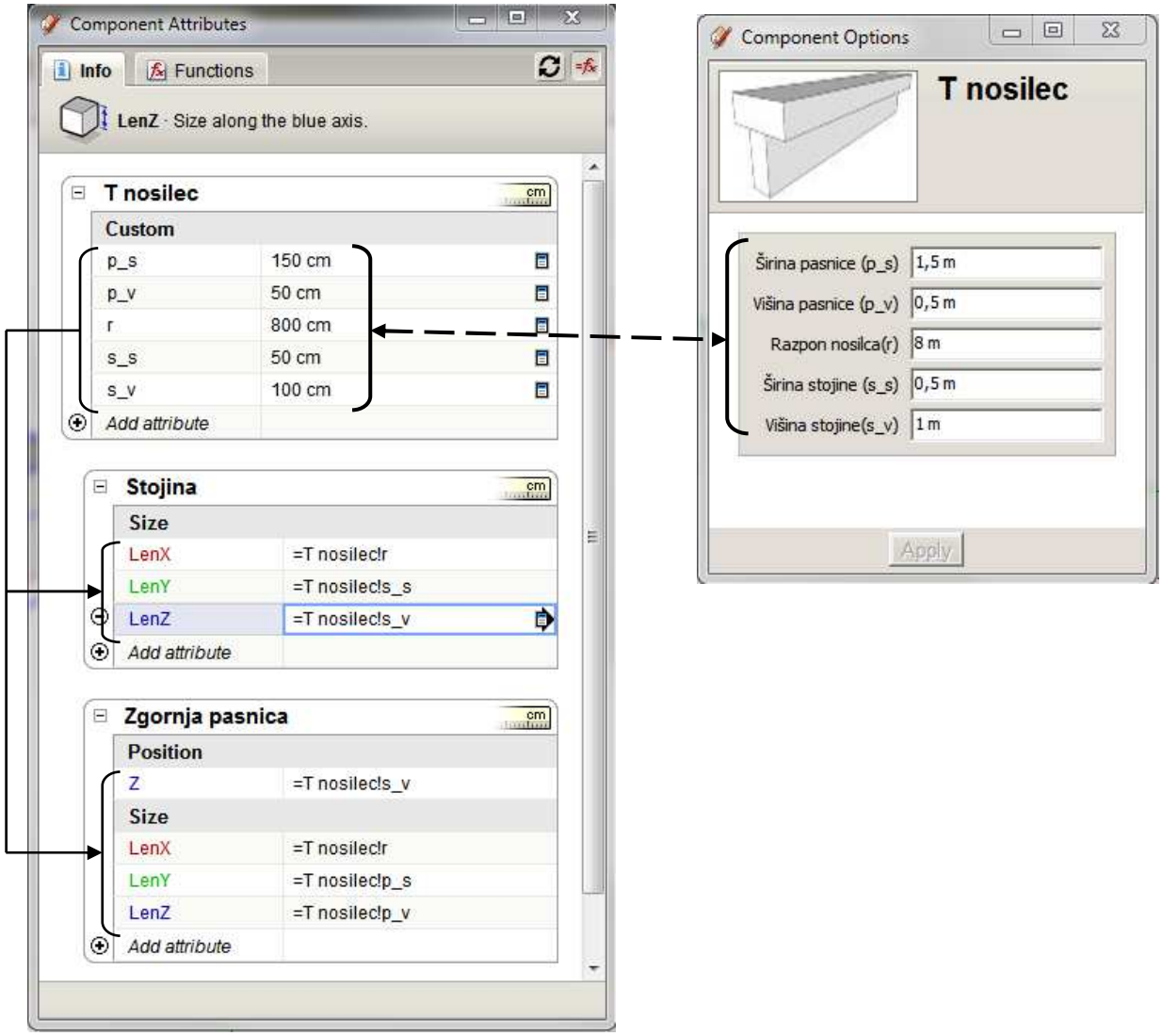


Slika 6: Starševska komponenta (T nosilec) in pozicija njenega koordinatnega izhodišča

Sedaj lahko pričnemo definirati parametre oziroma atribute vsake podkomponente in jih povezovati s parametri starševske komponente. Za ta korak ima SketchUp izdelan poseben uporabniški vmesnik

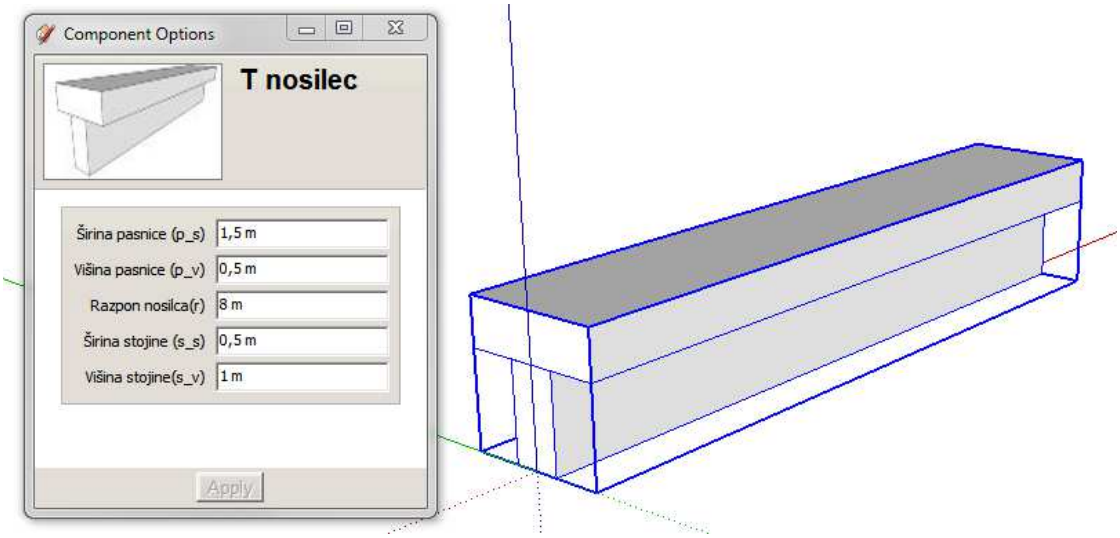
pod imenom Component Attributes (levo okno na Slika 7), kjer lahko definiramo svoje parametre, jim podamo začetne vrednosti, določimo ali jih uporabnik lahko vidi, pod kakim imenom in ali jih lahko spreminja. Posebnost SketchUp pristopa h komponentam je, da so lahko na uporabniškemu vmesniku za upravljanje s komponento vidni le parametri starševske komponente (desno okno na Slika 7), ki so v primeru komponente T nosilca dimenzije prečnega prereza in razpona: p_s, p_v, r, s_s, s_v (črtkana povezava med okni).

Te parametre je potrebno preko funkcij oziroma pravil povezati s parametri podkomponent, ki so vnaprej definirani v SketchUp-u in lahko vključujejo dimenzije, pozicije, vidnost in ostale atribute (polna povezava na Slika 7).

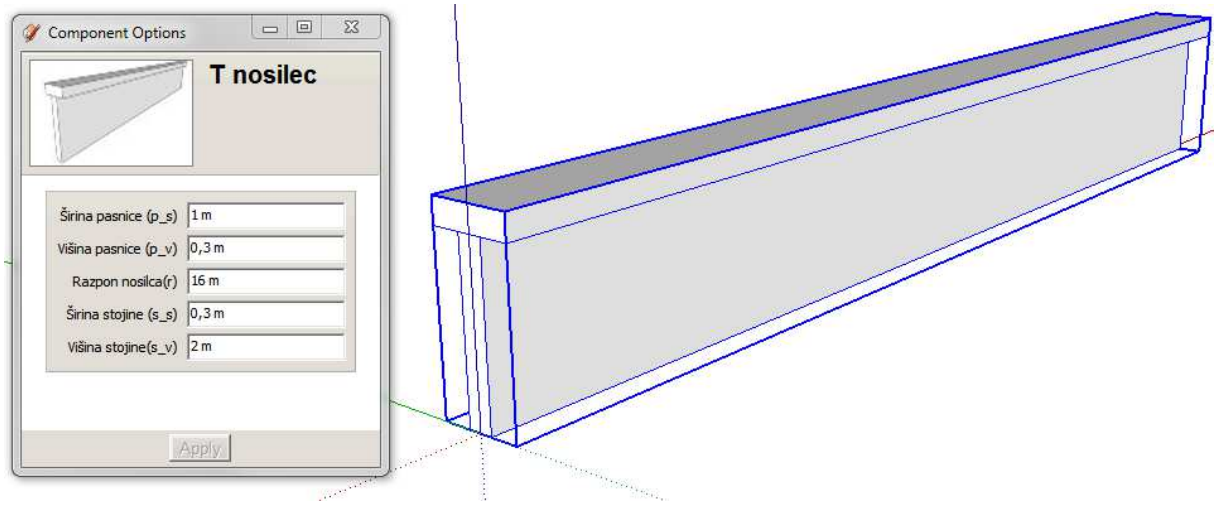


Slika 7: Uporabniška vmesnika za izdelavo in upravljanje s komponento in njune medsebojne povezave

S tem smo definirali vsa pravila, ki jim mora parametrična dinamična komponenta slediti, da se bo obnašala po pričakovanjih. Uporabnik lahko nato poljubno spreminja vrednosti parametrov v uporabniškem vmesniku in komponenta se po potrditvi le-teh sama dinamično spremeni (Slika 8 in Slika 9).



Slika 8: T nosilec z originalno podanimi parametri



Slika 9: Nosilec s poljubno spremenjenimi parametri s strani uporabnika

Sedaj ko imamo izdelano komponento jo lahko shranimo na pomnilnik računalnika, kjer imamo shranjene ostale konstrukcijske komponente. Če je teh komponent več, je smiselno izdelati družine komponent in nato umestiti vse komponente v zbirki v te družine glede na njihovo funkcijo. T nosilec bi v tem primeru umestili v družino nosilcev.

4.2.3 Zbirka parametričnih dinamičnih konstrukcijskih komponent

V prejšnjem poglavju je opisan najbolj enostaven primer izdelave konstrukcijske komponente, ki vključuje samo spremembe dimenzij podkomponent. Z bolj zahtevnimi funkcijami je možno izdelati veliko bolj zapletene komponente, pri katerih se lahko spreminjajo tudi pozicije, vidnost in število delov komponente ter še mnogo drugega. S temi bolj naprednimi funkcijami sem za predstavitev uporabnosti komponent v SketchUp-u izdelal zbirko dinamičnih komponent, ki zastopajo prefabricirane armiranobetonske konstrukcijske elemente, to so stebri, temelji, plošče in nosilci, ter jih razporedil v pripadajoče družine.

Parametrične dinamične konstrukcijske komponente so še posebej zanimive ravno pri uporabi prefabriciranih konstrukcijskih elementov, ker imajo ti točno definirane dimenzije in pravila obnašanja in se velikokrat ponavljajo v različnih velikostih, a vedno v istih oblikah. V prilogi A je zajeta zbirka vseh teh komponent in njihovih opisov, prav tako pa so točno določena poimenovanja vseh parametrov in grafični prikazi dinamičnih komponent.

4.3 Nadgradnja SketchUp-a z vtičniki

4.3.1 Programski jezik Ruby

Ruby je kot programski jezik izdelal Yukihiro Matsumoto, ki je prvo, alfa verzijo izdal leta 1995. Cilj avtorja je bil izdelati jezik, ki je bolj zmogljiv kot Perl in bolj objektno usmerjen kot Python, s katerima si tudi deli nekatere značilnosti. Ruby je objektno usmerjen jezik, ki privzame, da je vsaka stvar objekt, ne glede na to ali je to beseda, številka ali datoteka. Ruby se drži načela, da obstaja več načinov, kako neko stvar narediti, kar ima za posledico, da ga lahko uporabljamo podobno kot bolj poznane programske jezike, to je bolj statično, ali pa programsko kodo pišemo v popolnoma unikatnem Ruby stilu.

Ta stil temelji na tem, da se Ruby bere skoraj kot pravi, živ jezik, saj se s tem poveča čitljivost in razumljivost celotne kode, kar je razvidno tudi na naslednjem primeru enostavne programske Ruby kode.

```
3.times do print "Hello, world!" end
```

Z osnovnim znanjem angleščine takoj postane jasno, kaj koda zahteva od računalnika, saj lahko dobesedno preberemo, da naj trikrat prikaže poved »Hello, world!«. Če primer kode poskusimo zagnati, prejmemo naslednji rezultat, ki je enak našim pričakovanjem.

```
Hello, world!Hello, world!Hello, world!
```

Ker Ruby vse smatra kot objekt, je nujno razumevanje, kako programski jezik med različnimi objekti sploh loči in kako nad njimi izvaja operacije. Vsak objekt v Ruby-ju ima definiran svoj razred (angl. *class*), na primer bolj poznani so Float, File, Hash, Array. Nad objekti v določenemu razredu se lahko izvajajo metode (angl. *method*), ki so opreacijsko srce Ruby-ja, ker izvajajo operacije nad objekti.

4.3.2 Ruby vtičniki za SketchuUp

Ena izmed glavnih prednosti SketchUp-a je ravno zmožnost razširitve ali prilagoditve obstoječih operacij in funkcij z vtičniki, ki so napisani v Ruby-ju. SketchUp ima vgrajen programski vmesnik za Ruby aplikacije (angl. *Application Programming Interface*, kratica *API*), ki je v resnici knjižnica Ruby razredov in metod, ki so značilne za SketchUp in omogočajo nemoteno komunikacijo med Ruby-jem in SketchUp-om. Za lažjo predstavo naj naštejem nekaj teh razredov: Edge, Layer, Menu, Model, Vector3d in tako naprej.

V SketchUp-u vtičnik uporabimo tako, da ga napišemo v enem izmed množice programov za izpis in ureditev kode (moja izbira je bil program Notepad++), ga shranimo s končnico .rb v datoteko Plugins, kjer je nameščen SketchUp in SketchUp zaženemo. Program sam zazna, da je bil dodan nov vtičnik in ga takoj ob zagonu aktivira, nato pa vtičnik čaka na poziv uporabnika, da prične s svojimi operacijami.

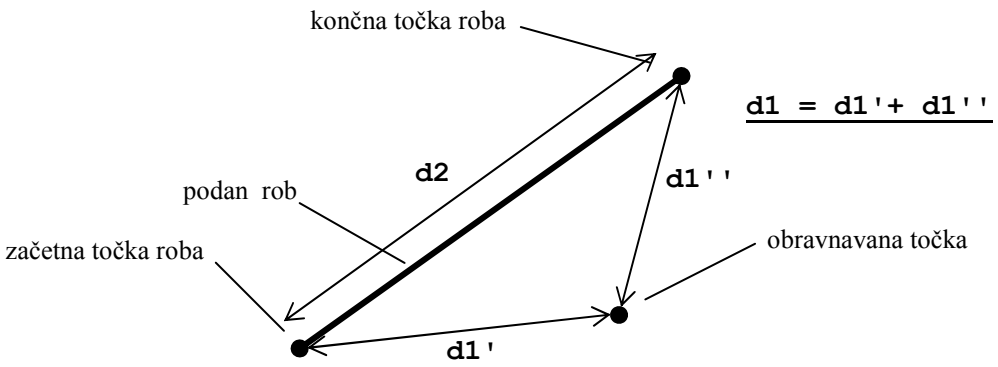
4.3.3 Izdelava Ruby vtičnika

Za predstavitev izdelave Ruby vtičnika sem izbral zelo kratek vtičnik z imenom *Point on edge.rb*, kateri nam pove ali neka določena točka (razred `Point3d`) leži na izbranem robu (razred `Edge`). Programska koda vtičnika je predstavljena spodaj, za boljšo geometrijsko predstavo določenih neznank v kodi sami, pa ja priložena še Slika 10.

```
1     class Geom::Point3d
2       def on_edge?(edge)
3         pt_start = edge.start.position
4         pt_end = edge.end.position
5         d1 = (self.distance(pt_start) + self.distance(pt_end))
6         d2 = pt_start.distance(pt_end) + 0
7         if (d1 <= d2) or (d1-d2 < 1e-10)
8           return true
9         else
10          return false
```

```
11        end
12        return UI.messagebox("Something is wrong.")
13      end
14    end
```

Najprej odpremo razred `Point3d` in vanj dodamo novo metodo imenovano `on_edge?`, ki zahteva vnos nekega roba. V vrsticah 3 in 4 sta definirani začetna in končna točka roba v prostoru (`pt_start` in `pt_end`), sledi pa še izračun dolžin `d1` in `d2`, katerih geometrijska predstava je prikazana na Slika 10. Koda se v vrstici 7 nadaljuje z logičnimi vrati, ki določijo, ali je `d1` manjši ali enak `d2` z zelo majhnim odstopanjem in če to drži, metoda vrne `true`, drugače pa `false`. Če se v metodi zgodi karkoli nepričakovanega in se zgornja koda ne more izvesti, se odpre opozorilno okno (`UI.messagebox`), ki nam pove, da je prišlo v metodi do napake. Na koncu se metoda in razred zapreta z dvojno uporabo kode `end`.



Slika 10: Geometrijska predstava dolžin `d1` in `d2`

4.3.4 Vtičniki v povezavi z dinamičnimi komponentami

Kljub temu, da dinamične komponente zelo poenostavijo izdelavo trodimenzionalnih modelov konstrukcij, pa same po sebi niso dovolj, da bi se taka razširitev smatrala kot zametek BIM-a. Za tako obsežno funkcionalnost potrebujemo nekaj več opcij in orodij v samem SketchUp-u, ki jih implementiramo v obliki Ruby vtičnikov. SketchUp API za komponente uvede dva nova razreda ComponentDefinition in ComponentInstance, ki vključujeta metode značilne le za definicije komponent in instance komponent.

Funkcionalnost SketchUp-a se lahko razširi z možnostjo določitve in izrisom geometrijskih osi ter pozicioniranja komponent na sečiščih osi, implementacijo vezanih komponent in formiranja vezi med komponentami. Možno bi bilo tudi izdelati popise za izdelavo potrebnih prefabrikatov in njihovih

dimenzij. Ta seznam se z vedno večjo pozornostjo na detajle lahko daljša v nedogled in z vedno večimi vtičniki bi se vedno bolj približevali pravemu konceptu BIM.

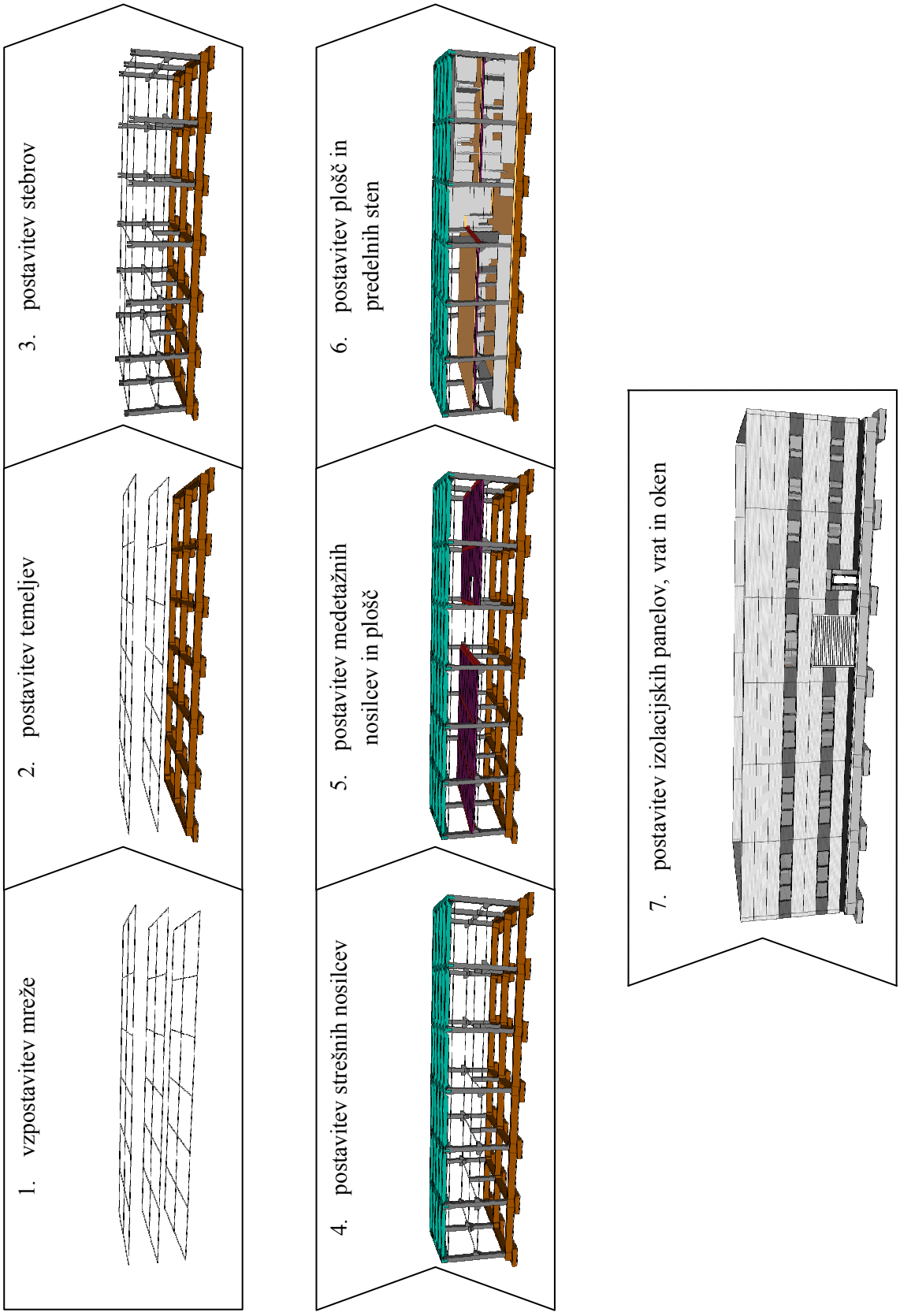
V tej diplomski nalogi sem sam izdelal le nekaj vtičnikov, ki zadevajo avtomatično izdelavo mreže na podlagi uporabnikovega vnosa dimenzij in pozicioniranje komponent na sečiščih te mreže, saj bi izdelava ostalih presegla obseg diplomske naloge in to tudi ni njen glavni cilj. Izdelani vtičniki so predstavljeni v prilogi B, kjer je izpisana njihova programska koda, komentarji in kratke obrazložitve funkcionalnosti vsakega vtičnika.

5 UPORABA NADGRADITEV

V sledečem poglavju je opisan postopek uporabe izdelanih vtičnikov in zbirke dinamičnih komponent v programu SketchUp na primeru dvoetažnega industrijskega objekta nazivnih dimenzij 42,5 m × 15,5 m × 8,5 m, katerega konstrukcija je izdelana iz armiranobetonskih montažnih elementov. Objekt je v lasti podetja IRIO d.o.o. in se nahaja na področju industrijske cone Veliki Otok pri Postojni, izgradnja pa je bila zaključena v začetku leta 2007, v njemu pa se nahajajo proizvodni, pisarniški in skladiščni prostori.

Izdelava računalniških modelov je ena izmed najbolj odgovornih projektantskih nalog in zahteva izredno natančnost, saj se na podlagi teh modelov izdela končne načrte za izgradnjo. Poleg tega ima izdelovalec opravka z zelo velikim številom podrobnosti in zato je obvezno, da si pred samo izdelavo modela ovirno zastavi potek modeliranja po korakih. Ti si morajo slediti v logičnem zaporedju tako, da vsak naslednji korak dopolni model na osnovi trenutnega koraka. Nesmiselno in tudi zelo narobe bi bilo na primer v model najprej postaviti fasadne izolacijske panele in šele nato izdelati konstrukcijo, ki te panele nosi, saj obstaja velika verjetnost, da bomo morali panele še spreminjati in prilagajati, verjetnost za napake pa se s tem eksponentno poveča.

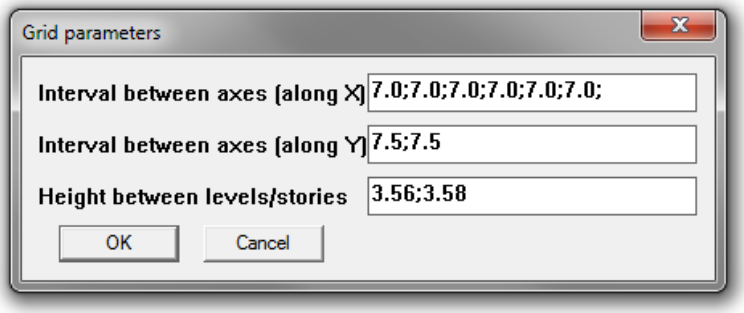
Celoten postopek izdelave računalniškega modela armiranobetonskega montažnega objekta lahko razdelimo na naslednjih sedem faz: 1 – vzpostavitev mreže, 2 – postavitev temeljev, 3 – postavitev stebrov, 4 – postavitev strešnih nosilcev, 5 – postavitev medetažnih nosilcev in plošč, 6 – postavitev predelnih sten in plošč, 7 – postavitev izolacijskih panelov. Postopek modeliranja je po fazah oziroma korakih je za lažjo vizualizacijo predstavljen tudi grafično na Slika 11. Če primerjamo ta potek z realnim potekom izgradnje montažnega objekta, lahko opazimo zelo veliko podobnost, kar nikakor ni naključje, saj je eno izmed glavnih načel BIM-a ravno reprezentativnost, kar vključuje tudi zaporedje izdelave modela zgradbe.



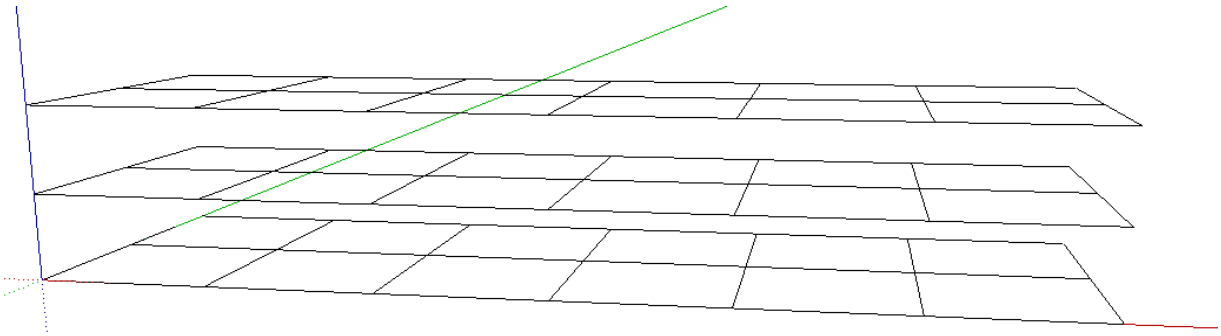
Slika 11: Potek izdelave modela

5.1 Uporaba vtičnikov za izdelavo mreže konstrukcijskih osi

Na podlagi obstoječega stanja in končnih načrtov konstrukcije objekta sem odčital razmake med konstrukcijskimi osmi v obeh smereh in razmake med etažami ter nivojem naleganja primarne strešne konstrukcije. Iz teh podatkov sem pomočjo vtičnika *Draw a 3D grid* (Slika 13), ki je dostopen preko menijev Plugins → Grid setup, ustvaril trodimenzionalno mrežo konstrukcijskih osi, ki je vidna na Slika 13.



Slika 12: Definicija osi s pomočjo vtičnika draw a 3D grid

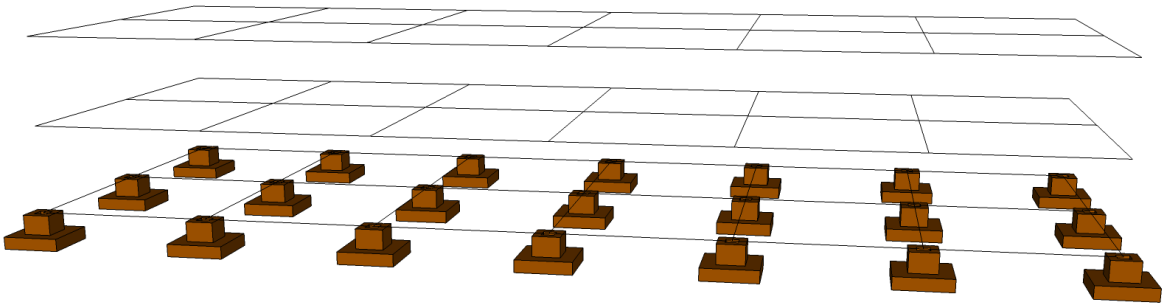


Slika 13: Prikaz 3D mreže konstrukcijskih osi montažnega objekta

5.2 Uporaba komponent točkovnih elementov

Mreža nam poleg geometrijske orientacije pomaga tudi pri pozicioniranju komponent točkovnih konstrukcijskih elementov, ki se nahajajo na sečiščih konstrukcijskih osi. To opravilo je zelo poenostavljeno z uporabo drugega vtičnika imenovanega *Position at grid intersections*, ki je prav tako dostopen preko menijev Plugins → Grid setup. Za njegovo uporabo je potrebno označiti željene konstrukcijske mreže, ki smo jih malo prej izdelali ali pa poljubno izrisali z orodji, ki jih ponuja na razpolago SketchUp, in zagnati vtičnik.

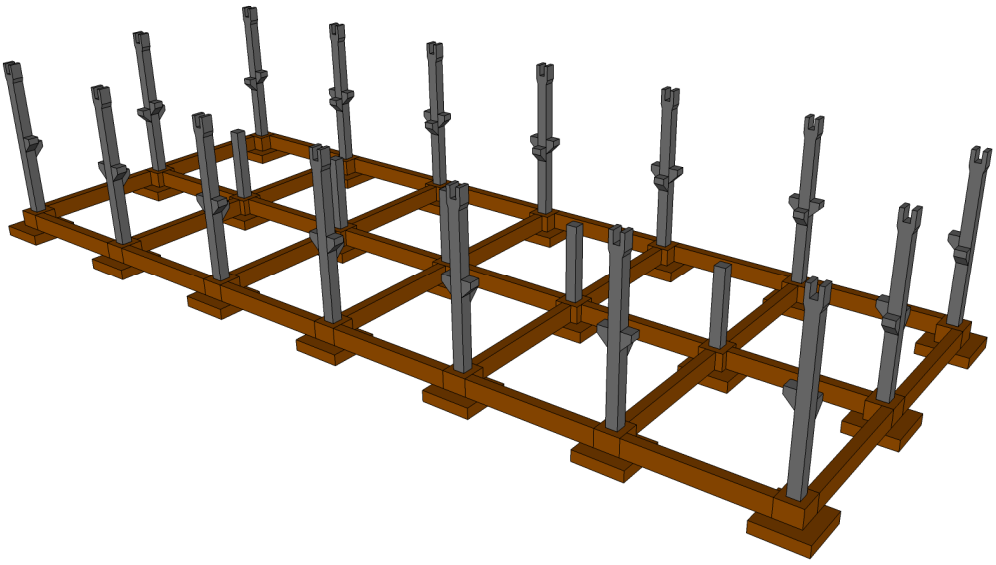
Po zagonu nam vtičnik odpre okno z zbirko dinamičnih konstrukcijskih komponent opisanih v prilogi A in da uporabniku možnost izbire definicije komponente, katere instance se bodo pozicionirale na sečišča mrež. To izpolni zahtevo po uporabnikovi izbiri komponente s pomilnika, ki smo jo opisali na Slika 3. Uporaba tega vtičnika s tem ni omejena le na določene komponente, saj lahko uporabnik sestavi svojo zbirko komponent in jih uporablja, prav tako pa bi se lahko zbirka teh komponent nahajala na medmrežju. Tako operacijo lahko uporabimo za postavitev instanc točkovnih komponent, kot so točkovni temelji in stebri, kar je v primeru temeljev razvidno na Slika 14.



Slika 14: Postavitev temeljev na sečiščih spodnje mreže konstrukcijskih osi

Naslednji korak je izbira in postavitev stebrov ter določitev končnih dimenzij komponent elementov z vnosom parametrov preko uporabniškega vmesnika, katerega uporaba je prikazana v četrtem poglavju na Slika 8 in Slika 9. Na žalost SketchUp pri uporabi dinamičnih komponent ne dopušča definiranja parametrov iste komponente tako preko uporabniškega vmesnika kot tudi na grafični način, zato sem se zaradi zadovoljitve potrebe po natančnosti odločil za uporabo uporabniškega vmesnika. S tem je izpolnjena druga operacija sistema uporabe komponent (Slika 3). Končni produkt tega postopka je viden na Slika 15, na kateri so zaradi večje preglednosti komponente obarvane po plasteh.

Na Slika 15 je opazen tudi prvi primer ročne postavitve instanc vnaprej izdelane komponente. To so temeljne grede, ki povezujejo točkovne temelje med seboj in dajejo podporo nenosilnim stenam ter fasadnim panelom. Zaradi vnaprej izdelanih podlog oziroma komponent je tak postopek še vedno veliko hitrejši kot pa ročna izdelava elementov iz niča, saj moramo iz zbirke le izbrati komponento, jo pozicionirati in vnesti njene parametre.



Slika 15: Postavitev stebrov in temeljnih gred ter določitev parametrov vsem instancam

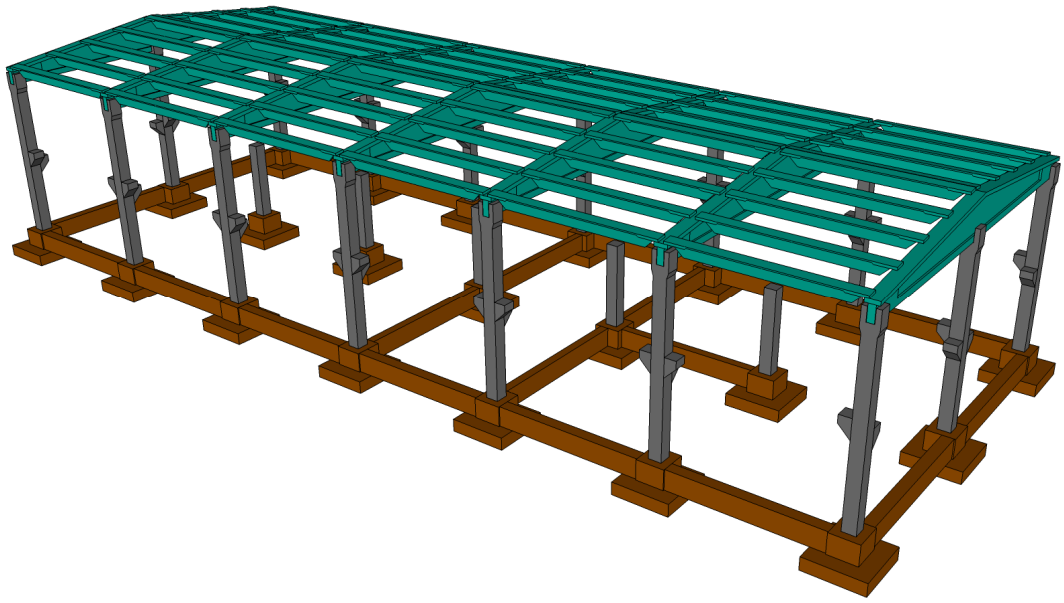
V tem poglavju sem podrobno opisal način uporabe komponent in dopolnitev v programu SketchUp, zato v nadaljevanju ni več podrobnih obrazložitev uporabe in je samo še predstavljen postopek dokončanja modela.

5.3 Uporaba komponent linijskih elementov

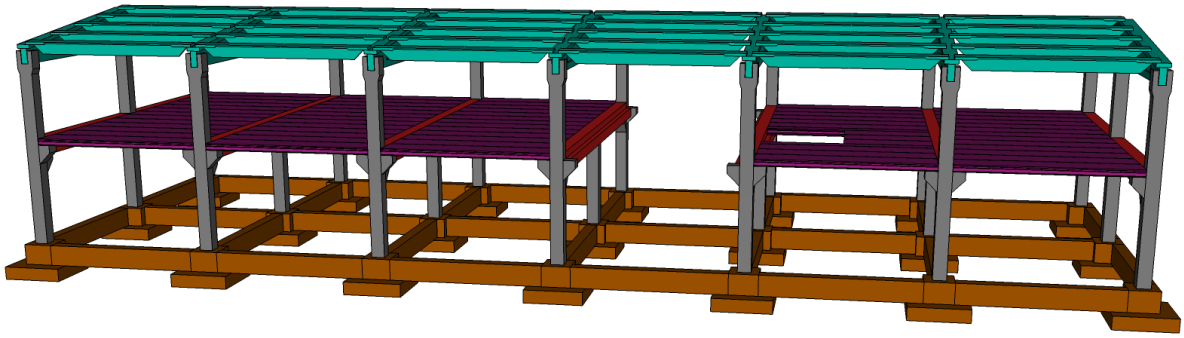
Na enak način kot v prejšnjem poglavju v model vnesemo in postavimo na izbrano mesto tudi primarne strešne nosilce in na te pozicioniramo nosilce kritine. Tudi tem instancam določimo parametre in tako dinamično spremenimo geometrijo nosilcev (Slika 16).

Po končani izdelavi strešne konstrukcije se lahko posvetimo še izdelavi medetažne konstrukcije, ki jo sestavljajo L nosilci in pa adhezijsko prednapete votle plošče, ki nalegajo na L nosilce. Po istem postopku kot prej tudi te elemente umestimo v model konstrukcije in pri tem pazimo na vertikalne preboje (Slika 17).

Pri vseh dinamičnih komponentah v zbirki je možen ročni vnos pozicije instance kot primer dodajanja informacij. Lahko bi dodali tudi druge dodatne informacije kot so na primer, koda elementa v skladišču, cena elementa, čas postavitve in tako naprej.



Slika 17: Ročna postavitev instanc elementov strešne konstrukcije



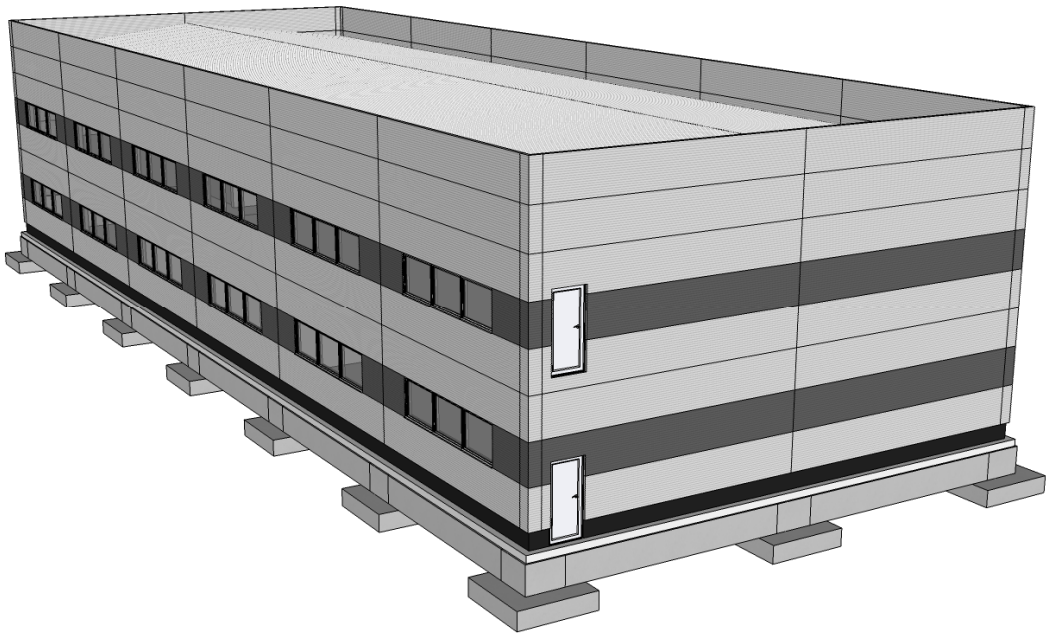
Slika 16: Postavitev medetažnih nosilcev in votlih plošč na konzole stebrov

Z zadnjim korakom smo zaključili izdelavo modela osnovne montažne konstrukcije objekta in to v le nekaj minutah v treh enostavnih korakih. Čas potreben za ročno izdelavo take konstrukcije brez vnaprej izdelanih komponent je lahko tudi en dan ali več. Če upoštevamo še to, da unikatni izdelki niso parametrični in je njihovo spreminjanje zelo zahteven in dolgotrajen proces, medtem ko spreminjanje parametrične konstrukcije zahteva le nekaj minut, je prednost uporabe parametričnih dinamičnih konstrukcijskih komponent na področju hitrosti in enostavnosti izdelave več kot razvidna. To je najbolj očitno na primeru primarnega strešnega nosilca, kjer bi izdelava in spreminjanje unikatnega trodimenzionalnega modela lahko zahtevala več ur namesto le nekaj minut.

5.4 Dopolnitev modela objekta

Ko imamo enkrat izdelano glavno geometrijsko podlago, to je v našem primeru montažna konstrukcija, je nadaljnjo ročno modeliranje nenosilnih in predelnih sten, plošč, fasadnih panelov in vertikalnih komunikacij dokaj enostavno, saj te elemente le dodajamo na konstrukcijo. V primerjavi z dosedanjim delom s komponentami je ročno modeliranje zelo počasno, a zaradi enostavnosti uporabe SketchUp-a ta postopek ni preveč dolgotrajen in se na dolgi rok izplača, saj model dopolnimo z vso potrebno osnovno geometrijo in si ga lahko vizualiziramo v celoti.

Ker ročno modeliranje ni namen te diplomske naloge, ta diplomska naloga ne vključuje razlage uporabe internih SketchUp orodij in funkcij za izdelavo dodatne geometrije, temveč je le prikazan končni produkt (Slika 18) in izvedena primerjava z realnim stanjem postavljenega objekta (Slika 19). Za bolj podroben vpogled v točne dimenzije objekta so v prilogi C zajeti vsi načrti konstrukcije, tlorisi, prerezi in pogledi industrijskega objekta.



Slika 18: Vizualizacija celotnega objekta v SketchUp-u

Slika 19: Fotografija objekta (vir: lasten)

6 ZAKLJUČEK

6.1 Povzetek diplomske naloge

V samem začetku naloge smo ugotovili, da je tradicionalen pristop do modeliranja premalo zmogljiv in da potrebujemo nekaj hitrejšega, boljšega in bolj avtomatiziranega, kar nas je hitro pripeljalo do uporabe parametričnih dinamičnih komponent. Spoznali smo, kaj je to okolje BIM, kaj so to komponente, kako se razlikujejo od navadnih geometrijskih objektov in osnovne principe delovanja parametričnih dinamičnih komponent v okviru okolja BIM. Predstavil sem tudi njihovo rabo, kako se ta razlikuje od tradicionalnega načina modeliranja in kakšne so prednosti in slabosti obeh pristopov. Na kratko so tudi opisane smernice, ki jim morajo sistemi komponent slediti. Ker so le-te okvirne, jih vsak od BIM programov drugače interpretira in integrira v svoje programsko okolje, zato je spekter tipov komponent zelo pisan.

Z izdelavo nadgradnje programa SketchUp sem prikazal integracijo konstrukcijskih komponent v okolje, ki tega še ne pozna, in vsaj v osnovi pojasnil način nastanka parametričnih dinamičnih komponent in spremljevalnih vtičnikov. S tem sem poskušal pokazati osnovne principe, ki jih uporablja vsak BIM modelirni program in kako lahko uporaba komponent spremeni način izdelave modelov na bolje. Čeprav je primer industrijske montažne hale, na katerem sem to nadgradnjo preizkusil, razmeroma enostaven, je prihranek na času zelo očiten, s čimer je uporaba komponent upravičena ne samo po časovni, temveč tudi po finančni plati.

6.2 Razvojne možnosti komponent

Z do sedaj pridobljenim znanjem lahko sklenemo, je delo s komponentami, ko enkrat osvojimo njihova osnovna načela, zelo enostavno in hitro. Tudi izdelava samih komponent za izkušenega inženirja z dobro podlago v geometriji (predvsem translacije, rotacije in transformacije) in programiranju ne predstavlja večje težave ne glede na izbiro programskega orodja. Zato je v zadnjih dveh desetletjih postala uporaba parametričnih komponent tako močno integrirana v postopke 3D modeliranja stavb, da jih uporabljamo brez večjega zavedanja. Na področju uporabe so komponente z uporabniku prijaznimi vmesniki načeloma že dosegle želeno stopnjo enostavnosti in vidnega napredka v tej smeri ni več pričakovati.

Razvoj komponent se vedno bolj usmerja v njihovo boljšo in lažjo izdelavo, s čimer se veča zbirka najrazličnejših komponent in s tem tudi obseg področij, ki jih pokrivajo, saj že nekaj časa nismo

omejeni izključno le na konstrukcijske elemente in inženirske oznake. Komponente s tem postajajo tudi geometrijsko vedno bolj svobodne in univerzalne, saj prihaja do razvoja funkcij, o katerih pred desetimi leti projektanti niso niti sanjali.

Največjo možnost razvoja osebno vidim v dinamičnosti komponent, kjer zelo izstopa področje povezovanja komponent med seboj in izdelave t.i. vezanih komponent. Z vedno večjo in bolj integrirano uporabo teh vezi postajajo komponente vedno bolj dinamične in odzivne na okolje okoli sebe, kar pomeni, da ima projektant vedno manj dela z ročnim vnašanjem in spreminjanjem parametrov. To v prvi vrsti zmanjša potrebo po času potrebnem za modeliranje, saj se s spremembo enega samega elementa v konstrukciji spremeni tudi okoliška konstrukcija brez človeških posegov. Poleg tega se v veliki meri odpravi tudi možnosti za človeške napake, saj ima projektant vedno manjšo vlogo izdelovalca in vedno večjo vlogo nadzornika.

Zadnja večja razvojna niša je tudi vedno bolj izpopolnjeno sodelovanje med komponentami in ostalimi funkcijami BIM programov. Narava komponent sledi določenim pravilom, vključuje pa tudi možnost pripisa dodatnih atributov in je zato zelo pripravna za integracije v druge sisteme oziroma funkcije. To vključuje terminske, materialne in finančne plane, ki prav tako sledijo strogo določenim pravilom in uporabljajo atribute komponent za lastno orientacijo.

V splošnem je razvoj komponent še daleč od svojega viška tako po funkcionalnosti kot tudi razširjenosti uporabe med uporabniki, saj starejši projektanti še vedno v veliki meri prisegajo na tradicionalne načine modeliranja. Na drugi strani pa mlajše generacije projektantov, ki so ciljna populacija 3D modelirnikov, vedno bolj posegajo po vseh možnih informacijskih napredkih, kamor spadajo tudi parametrične dinamične komponente, ker z njihovo uporabo vidijo priložnost konkuriranja starejšim generacijam kljub temu, da so njihove izkušnje na področju projektiranja manj obsežne. V prihodnosti lahko z vedno bolj razširjeno in poglobljeno uporabo BIM orodij pričakujemo tudi vedno večjo uporabo in razvoj komponent.

6.3 Zaključek

Ne glede na to, na kakšen način obravnavamo temo parametričnih dinamičnih komponent, naj bo to z vidika sestave, izdelave ali pa uporabe, vedno opazimo, da se ponavlja načelo univerzalnosti. To načelo nagovarja k enkratni, kakovostni izdelavi neke stvari, ki jo lahko nato venomer znova uporabljamo v različnih oblikah ter s tem pridobimo na času, denarju in izboljšamo samo kvaliteto končnega izdelka, to je grajenega okolja.

VIRI

Begrad. Ponudba armiranobetonskih montažnih elementov. 2012.
http://www.begrad.si/montazne-konstrukcije/ponudba/montazni-elementi/ (Pridobljeno 23. 7. 2012.)

Chonoles, M., Schardt J. 2003. UML 2 For Dummies. Indianapolis, Wiley: 434 str.

Cooper, P. 2009. Beginning Ruby. From Novice to Professional, Second Edition. Berkeley, Apress: 658 str.

Dynamic Components Self Paced Tutorials. 2012.
http://sketchup.google.com/3dwarehouse/cldetails?clid=56584c8086c0357ede650492e45fb14f
(Pridobljeno 12. 7. 2012.)

GDL Reference Guide. 2011. Budimpešta, GRAPHISOFT: 398 str.

Gradbeno podjetje Grosuplje. TOZD Gradbeni polizdelki – proizvodnja. 2009. Ljubljana, Gradbeno podjetje grosuplje: 58 str.

Klinc, L. 2010. Razvoj orodja za izdelavo integriranih modelov v programu SketchUp. Diplomska naloga. Ljubljana, Fakulteta za gradbeništvo in geodezijo, Oddelek za gradbeništvo, Konstrukcijska smer (samozaložba L. Klinc): 91 str.

Leban-Meza, A. 2007. Proizvodno poslovni objekt v industrijski coni Veliki Otok. Projektna dokumentacija. Postojna, IRIO d.o.o.: 250 f.

Metric Tutorials. Revit Architecture 2010 Families Guide. 2009. San Rafael, Autodesk: 308 str.

MicroStation GenerativeComponents V8*i* Help. 2012.
http://docs.bentley.com/en/GenerativeComponents/index.php (Pridobljeno 30. 7. 2012.)

Object Management Group – UML. 2012.
http://www.uml.org/ (Pridobljeno 20. 8. 2012.)

Parametric Building Modeling: BIM's Foundation. 2007. San Rafael, Autodesk: 6 str.

Roskes, B. 2009. Google SketchUp Cookbook. Sebastopol, O'Reilly Media: 384 str.

Ruby. Help and documentation for the Ruby programming language. 2012.
http://www.ruby-doc.org/ (Pridobljeno 15. 4. 2012.)

Sacks, R., Eastman, C. M., Lee, G. 2004. Parametric 3D modeling in building construction with examples from precast concrete. Automation in Construction 13: 291-312.

Scarpino, M. 2010. Automatic SketchUp. Creating 3-D Models in Ruby. Hanover, Eclipse Engineering LLC: 456 str.

SketchUcation Community Forums. 2012.
http://www.sketchucation.com/ (Pridobljeno 25. 6. 2012.)

SketchUp Ruby API Documentation. 2012.
https://developers.google.com/sketchup/ (Pridobljeno 10. 6. 2012.)

SketchUpBIM. 2012.
http://www.sketchupbim.com/ (Pridobljeno 23. 7. 2012.)

Stavbar IGM. Katalog montažnih elementov. 2008. Hoče, Stavbar IGM: 37 str.

Tekla Structures 12. Release Notes. 2006. Espoo, Tekla Corporation: 128 str.

SEZNAM PRILOG

PRILOGA A: Zbirka dinamičnih konstrukcijskih komponent

PRILOGA B: Ruby vtičniki

- vtičnik *Grid*;
- vtičnik Create 3D g*rid*;
- vtičnik *Grid positioning*;
- vtičnik *Point on edge*.

PRILOGA C: Načrti konstrukcije montažne hale

Na zgoščenki so priložene naslednje datoteke:

- »Česnik, Jure; Parametrične dinamične konstrukcijske komponente za prefabricirane AB elemente«, diploma v elektronski obliki;
- »Grid.rar«, Ruby vtičniki za program SketchUp, opisani v prilogi B, skupaj z zbirko dinamičnih konstrukcijskih komponent, opisanih v prilogi A;
- »Model montažne hale.skp«, trodimenzionalni model montažne armiranobetonske hale izdelane v poglavju 5.

PRILOGA A: Zbirka dinamičnih konstrukcijskih komponent

Vse parametrične dinamične konstrukcijske komponente so izdelane na podlagi armiranobetonskih prefabriciranih polizdelkov, ki jih proizvajajo podjetja Stavbar IGM, Gradbeno podjetje Grosuplje in Begrad.

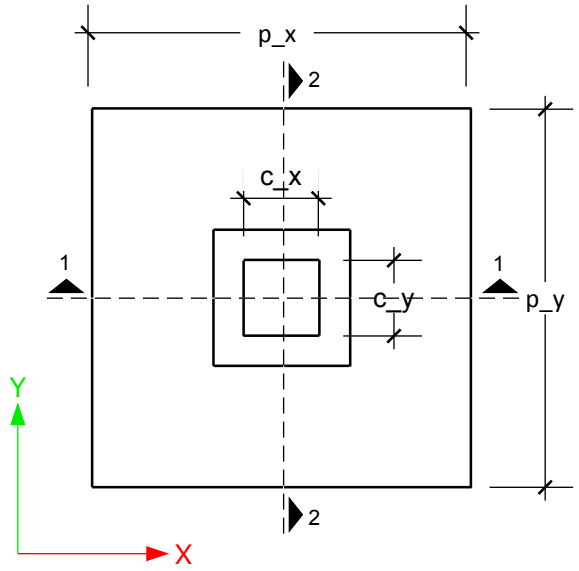
Določevanje parametrov je pri skoraj vseh komponentah prepuščeno uporabnikovim željam, zato je pri uporabi le-teh potreben razmislek o geometrijski smiselnosti in ali je tak element sploh na razpolago pri katerem od ponudnikov.

V nadaljevanju so geometrijsko in opisno definirani vsi parametri naslednjih dinamičnih konstrukcijskih komponent:

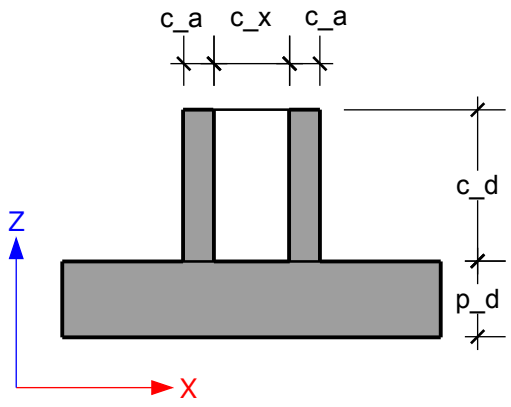
- čašast temelj (tem_čaš);
- temeljna greda (tem_gr);
- steber brez konzol (steb_brez);
- steber z utorom (steb_utor);
- steber z navadnimi konzolami (steb_navad);
- steber s »T« konzolami (steb_t);
- primarni strešni nosilec (nos_str);
- nosilec kritine (nos_krit);
- T nosilec (nos_t);
- pravokoten nosilec (nos_prav);
- L nosilec (nos_l);
- Π plošča (pl_pi);
- votla plošča (pl_vot).

ČAŠAST TEMELJ (tem_čaš):

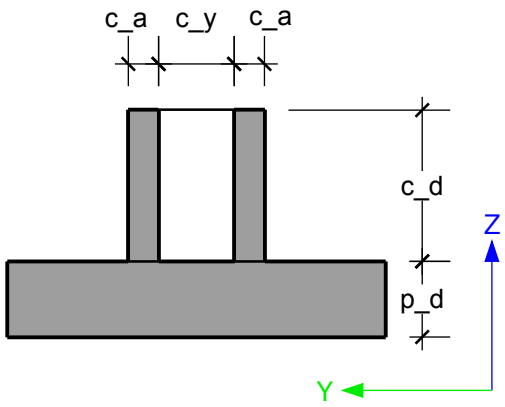
Tloris:



Prerez 1-1:



Prerez 2-2:

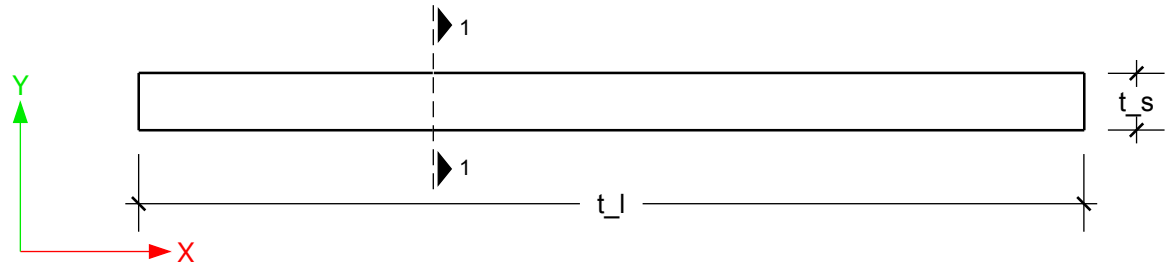


Dimenzije:

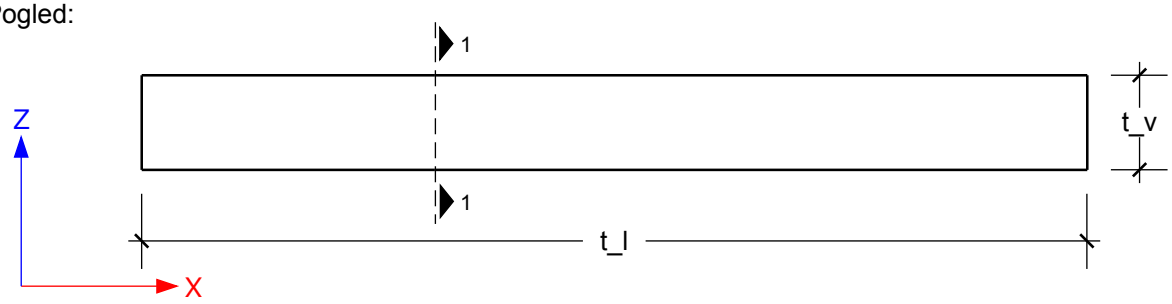
- c_x - notranja dimenzija čaše v smeri x, odvisna od dimenzij stebra.
- c_y - notranja dimenzija čaše v smeri y, odvisna od dimenzij stebra.
- c_d - višina čaše, odvisna od statičnega izračuna.
- c_a - debelina stene čaše, odvisna od obremenitve.
- p_x - dimenzija pete v smeri x, odvisna od nosilnosti tal.
- p_y - dimenzija pete v smeri y, odvisna od nosilnosti tal.
- p_d - višina pete, odvisna od statičnega izračuna.

TEMELJNA GREDA (tem_gr):

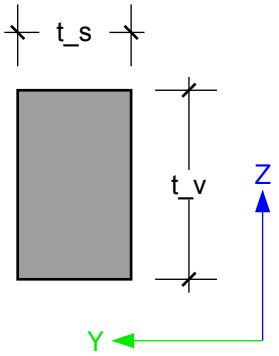
Tloris:



Pogled:



Prerez 1-1:

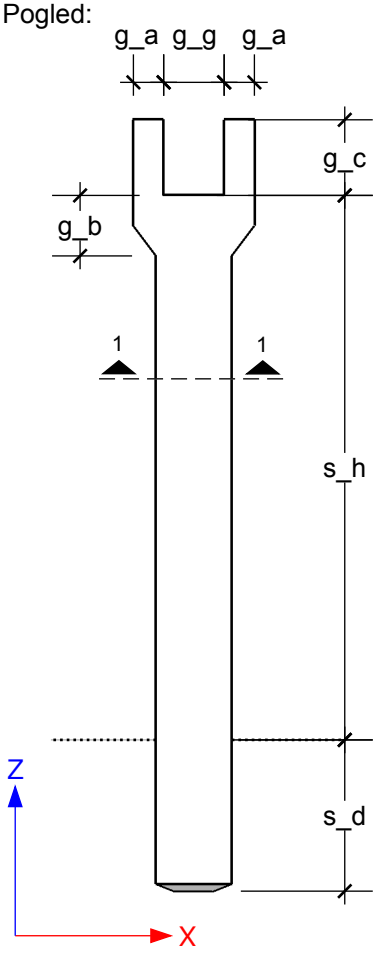


Dimenzije:

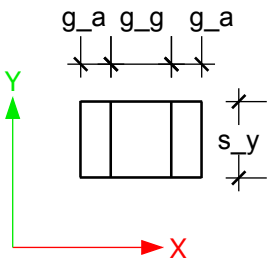
t_l - dolžina temeljne grede.
t_s - širina prereza temeljne grede.
t_v - višina prereza temeljne grede.

STEBER BREZ KONZOL (steb_brez):

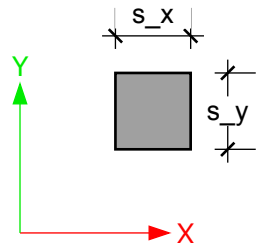
Pogled:



Tloris:



Prerez 1-1:

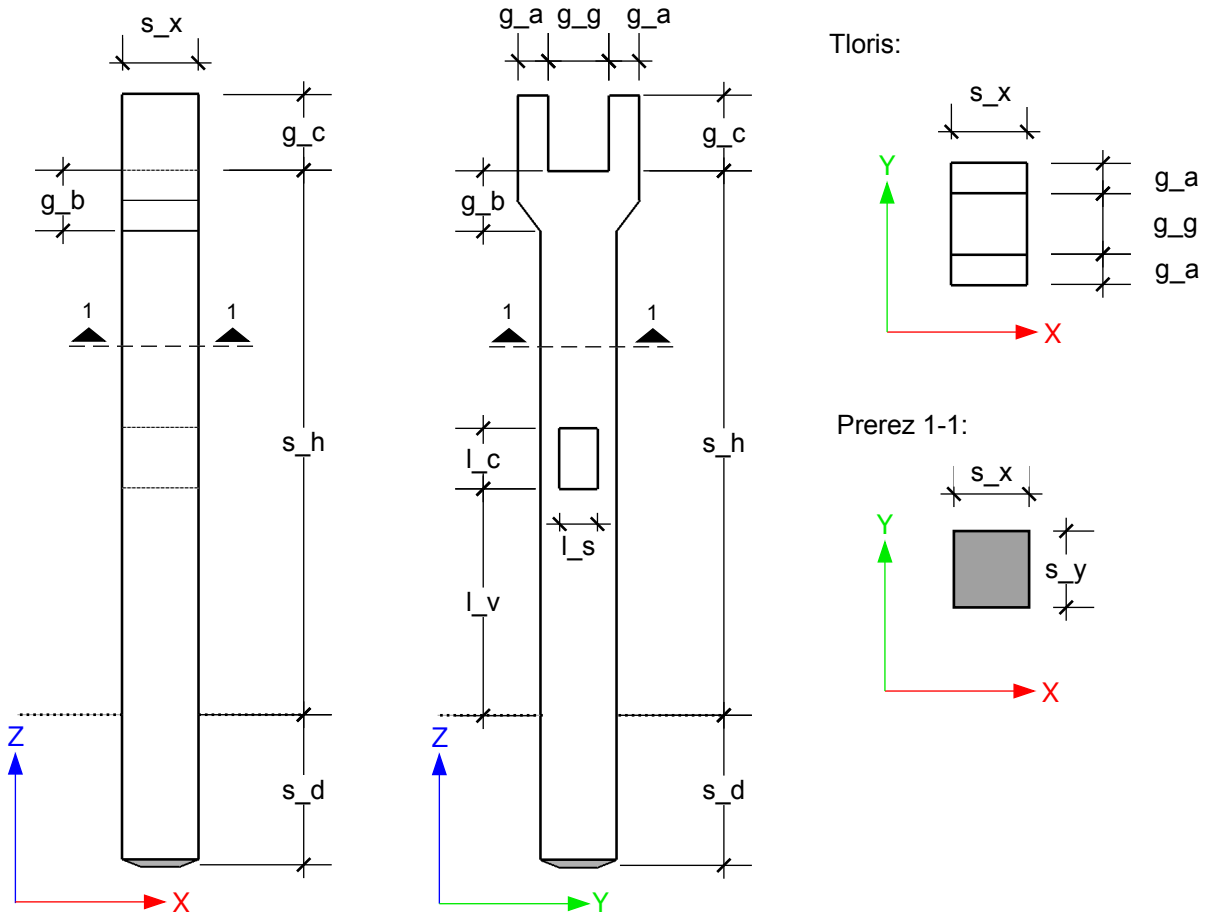


Dimenzije:

- s_x - prečna dimenzija stebra v smeri x.
- s_y - prečna dimenzija stebra v smeri y.
- s_h - višina stebra.
- s_d - globina vpetja stebra, odvisna od statičnega izračuna.
- g_g - širina stojine nosilca, ki nalega na glavo.
- g_a - debelina čaše, odvisna od obremenitve.
- g_c - višina čaše, odvisna od nosilca, ki nalega na glavo.
- g_b - višina prehoda iz glave na steber, odvisna od obremenitve.

STEBER Z UTOROM (steb_utor):

Pogleda (usmerjenost utora: vzdolžno):

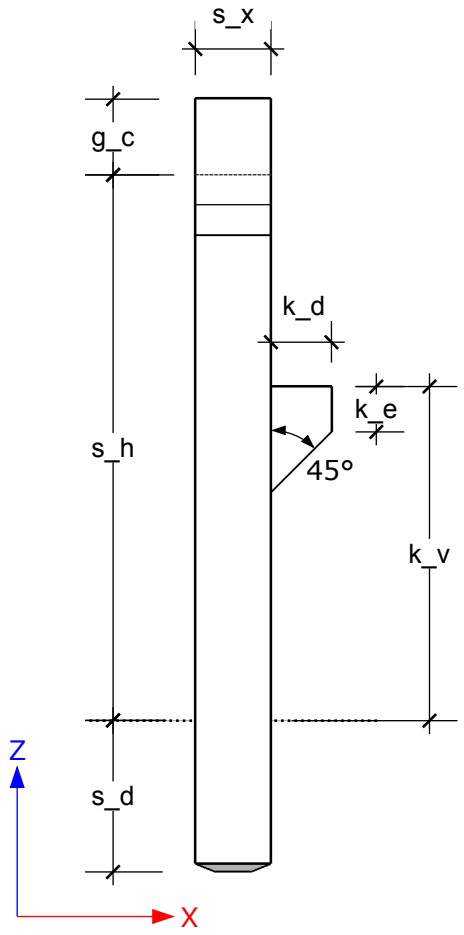


Dimenzije:

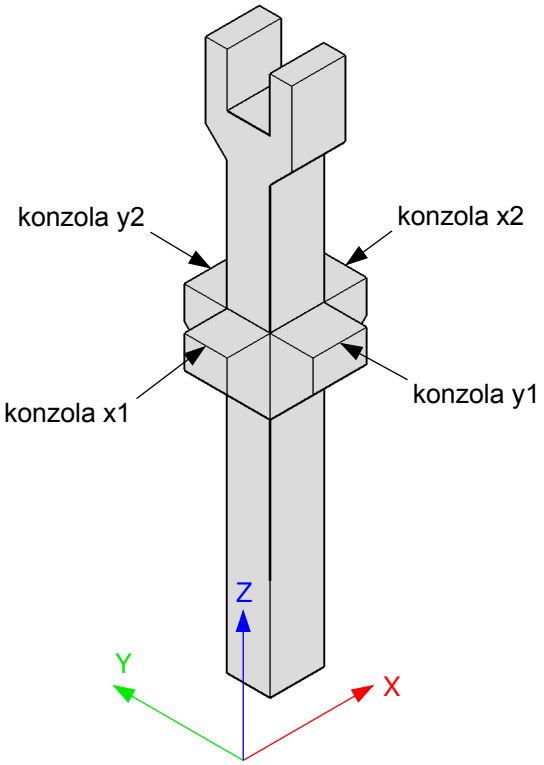
- s_x - prečna dimenzija stebra v smeri x.
- s_y - prečna dimenzija stebra v smeri y.
- s_h - višina stebra.
- s_d - globina vpetja stebra, odvisna od statičnega izračuna.
- g_g - širina stojine nosilca, ki nalega na glavo.
- g_a - debelina čaše, odvisna od obremenitve.
- g_c - višina čaše, odvisna od nosilca, ki nalega na glavo.
- g_b - višina prehoda iz glave na steber, odvisna od obremenitve.
- l_v - višina naleganja nosilca na utor.
- l_v - višina utora, odvisna od nosilca.
- l_s - širina utora, odvisna od nosilca.

STEBER Z NAVADNIMI KONZOLAMI (steb_navad):

Pogled (le konzola x2):



Aksonometrija nosilca:



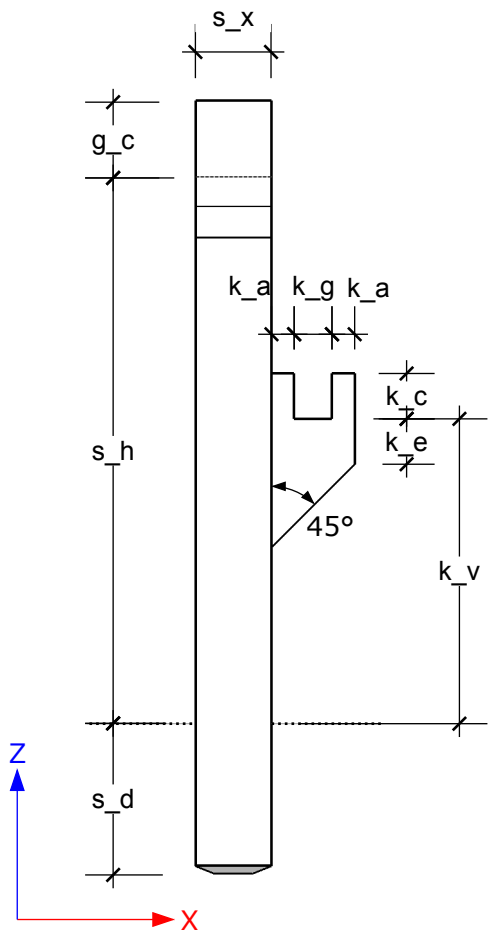
Za vse glavne dimenzije stebra glej Steber brez konzol.

Dimenzije:

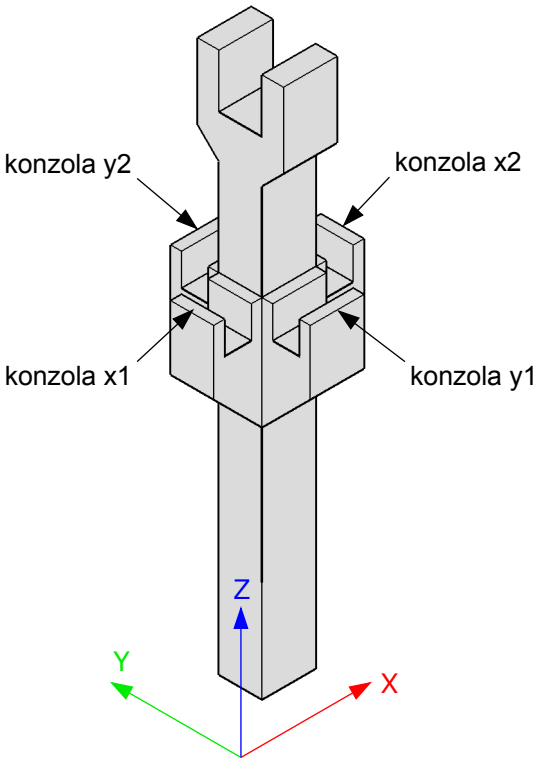
- k_d - globina konzole, odvisna od naleganja nosilca.
- k_e - debelina konzole, odvisna od obremenitve.
- k_v - višina konzole, različna za vsako pozicijo konzole.

STEBER S "T" KONZOLAMI (steb_t):

Pogled (le konzola x2):



Aksonometrija nosilca:



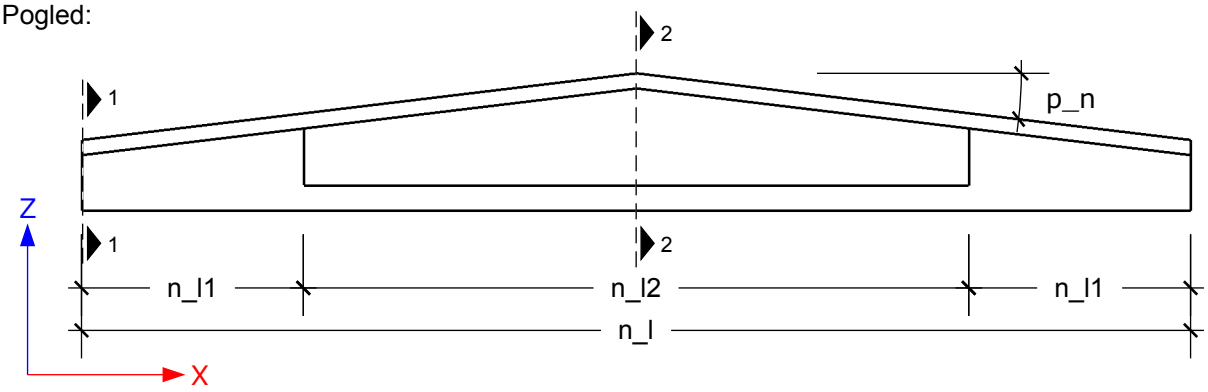
Za vse glavne dimenzije stebra glej Steber brez konzol.

Dimenzije:

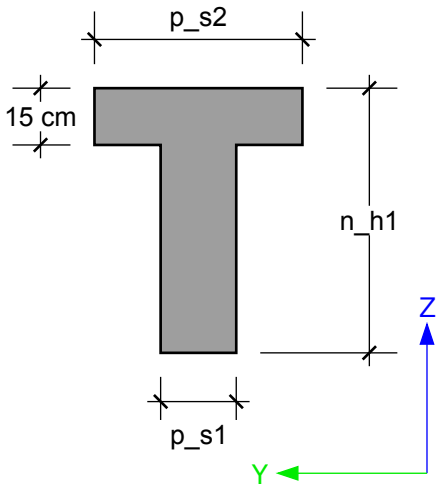
- k_a - debelina čaše konzole, odvisna od obremenitve.
- k_c - višina čaše konzole, odvisna od nosilca, ki nalega.
- k_e - debelina konzole, odvisna od obremenitve.
- k_v - višina konzole, različna za vsako pozicijo konzole.

PRIMARNI STREŠNI NOSILEC (nos_str):

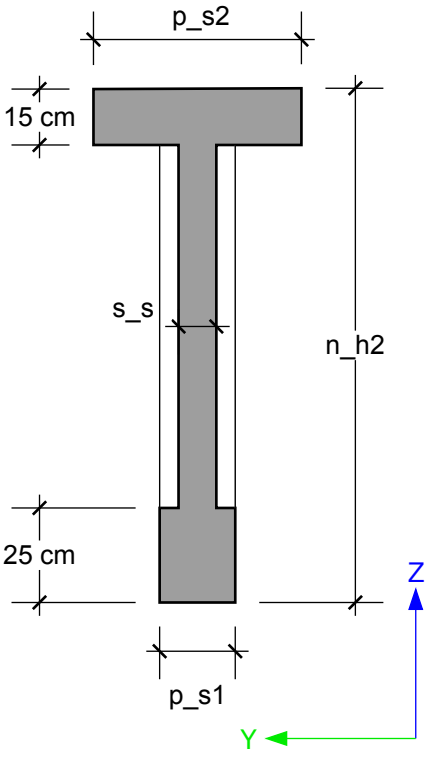
Pogled:



Prerez 1-1:



Prerez 2-2:

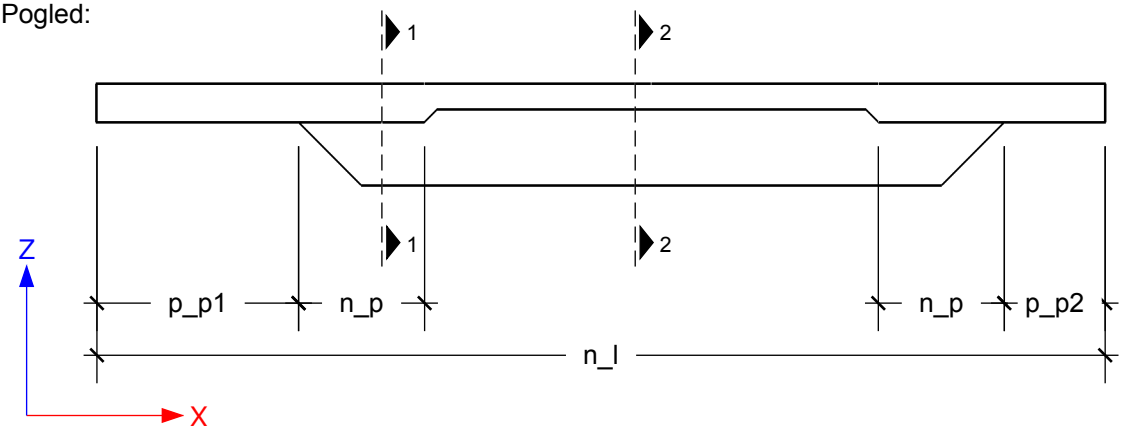


Dimenzije:

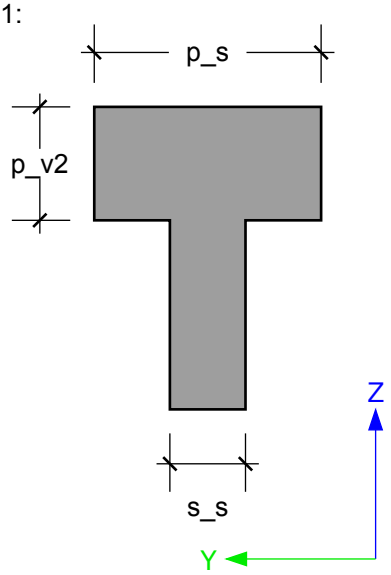
- n_l - dolžina nosilca.
- n_l1 - dolžina nosilca v polju, odvisna od n_l2.
- n_l2 - dolžina naleganja nosilca, odvisna od obremenitve.
- n_h1 - najmanjša višina nosilca (prerez 1-1).
- n_h2 - največja višina nosilca (prerez 2-2).
- p_n - naklon zgornje pasnice v %.
- p_s1 - širina spodnje pasnice v polju in stojine nad podporo.
- p_s2 - širina zgornje pasnice.
- s_s - širina stojine v polju.

NOSILEC KRITINE (nos_krit):

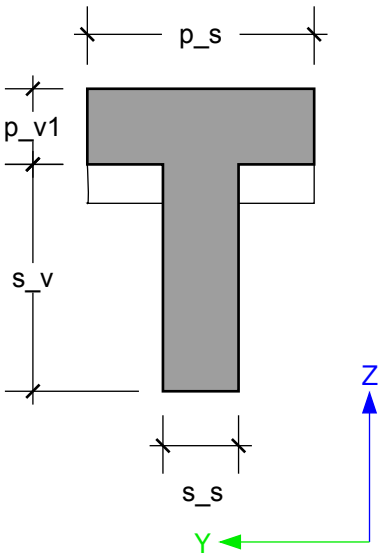
Pogled:



Prerez 1-1:



Prerez 2-2:

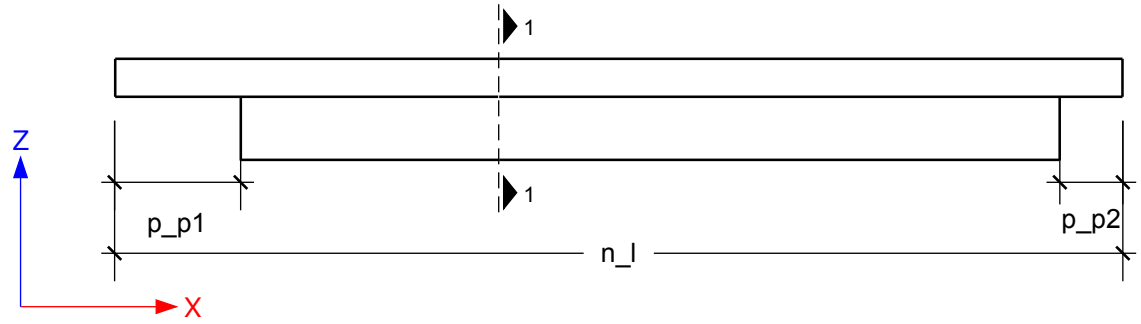


Dimenzije:

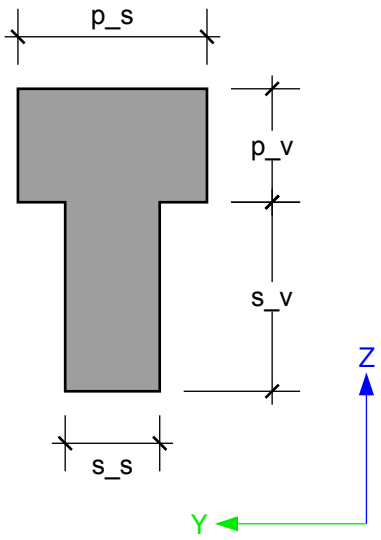
- n_l - dolžina nosilca.
- n_p - dolžina prehoda med poljem in podporo.
- p_s - širina pasnice.
- p_v1 - višina pasnice v polju.
- p_v2 - višina pasnice nad podporo.
- p_p1 - dolžina previsa pri podpori 1.
- p_p2 - dolžina previsa pri podpori 2.
- s_s - širina stojine.
- s_v - višina stojine v polju.

T NOSILEC (nos_t):

Pogled:



Prerez 1-1:

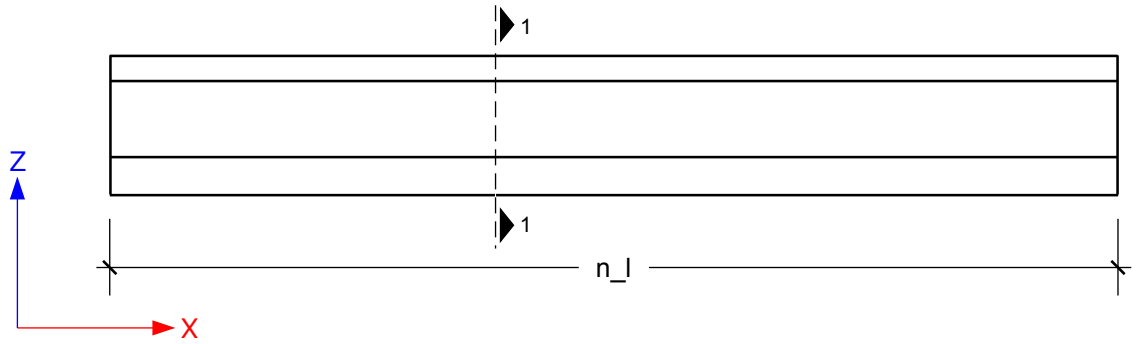


Dimenzije:

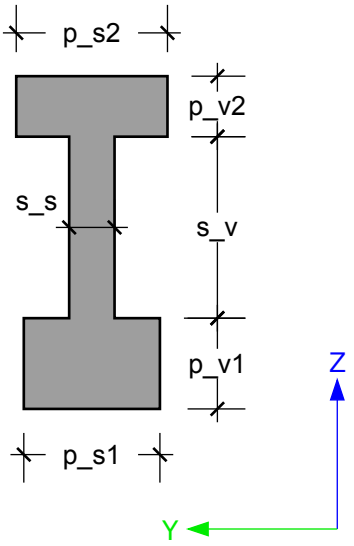
- n_l - dolžina nosilca.
- p_s - širina pasnice.
- p_v - višina pasnice.
- p_p1 - dolžina previsa pri podpori 1.
- p_p2 - dolžina previsa pri podpori 2.
- s_s - širina stojine.
- s_v - višina stojine.

I NOSILEC (nos_i):

Pogled:



Prerez 1-1:

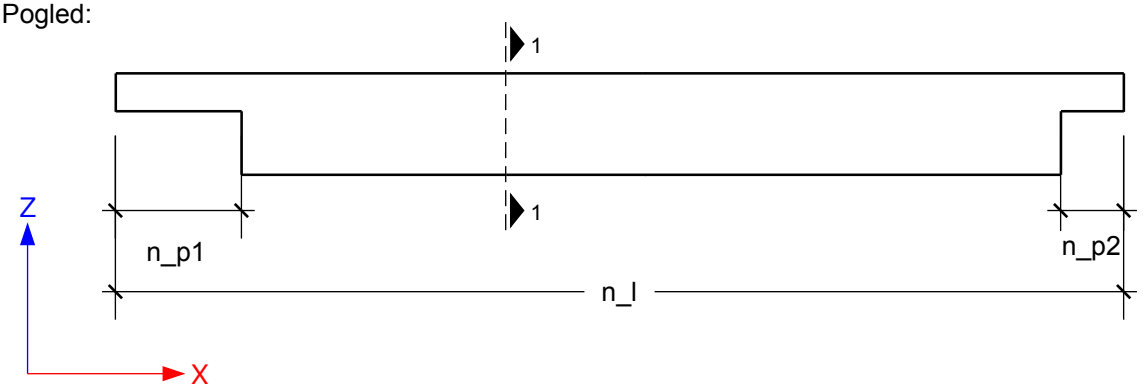


Dimenzije:

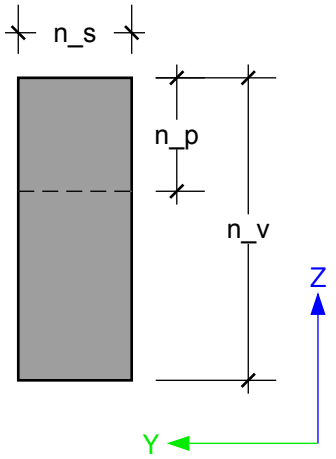
- n_l - dolžina nosilca.
- p_s1 - širina spodnje pasnice.
- p_s2 - širina zgornje pasnice.
- p_v1 - višina spodnje pasnice.
- p_v2 - višina zgornje pasnice.
- s_s - širina stojine.
- s_v - višina stojine v polju.

PRAVOKOTEN NOSILEC (nos_prav):

Pogled:



Prerez 1-1:

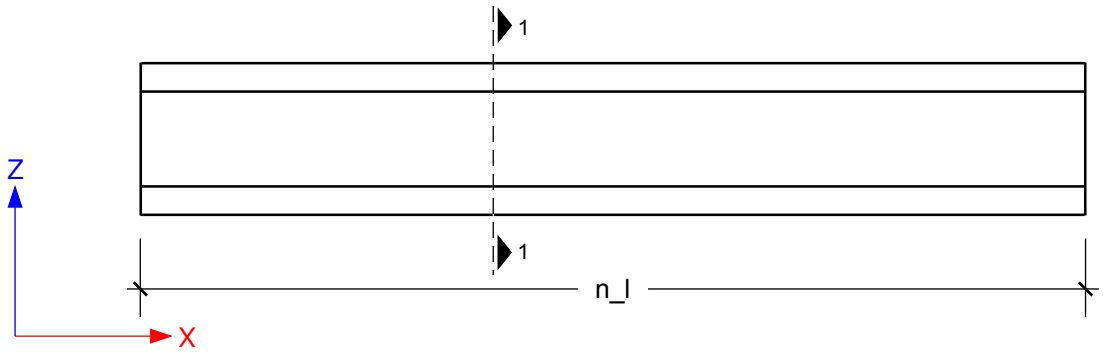


Dimenzije:

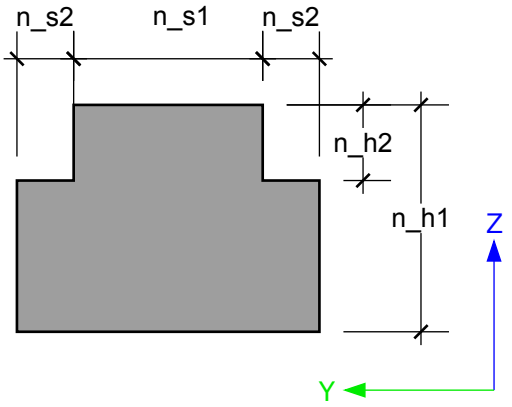
- n_l - dolžina nosilca.
- n_s - širina nosilca.
- n_v - višina nosilca.
- n_p - višina previsa nosilca.
- p_p1 - dolžina previsa pri podpori 1.
- p_p2 - dolžina previsa pri podpori 2.

L NOSILEC (nos_l):

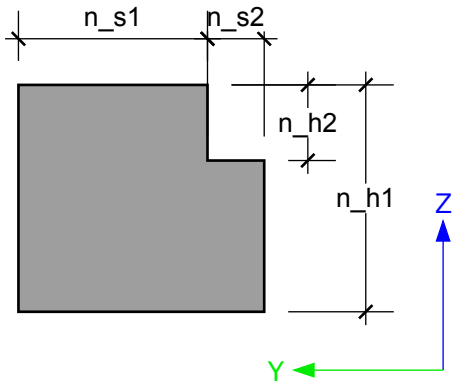
Tloris (tip nosilca: dvostranski):



Prerez 1-1 (tip nosilca: dvostranski):



Prerez 1-1 (tip nosilca: enostranski):

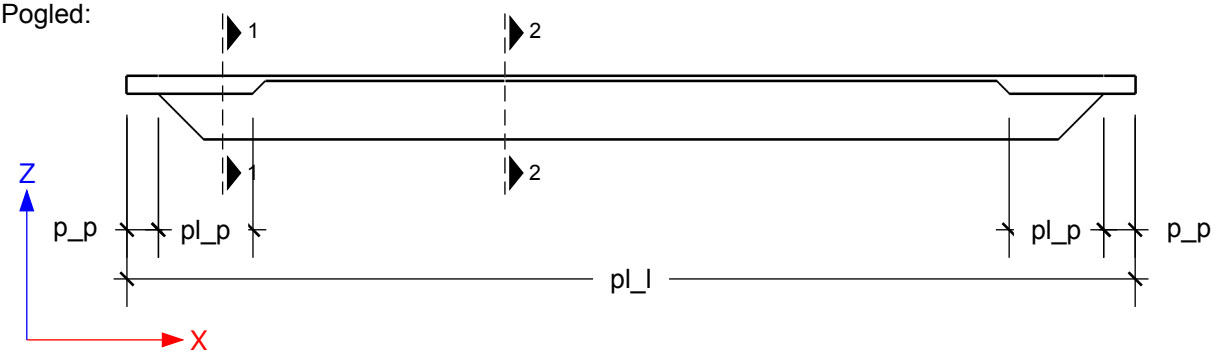


Dimenzije:

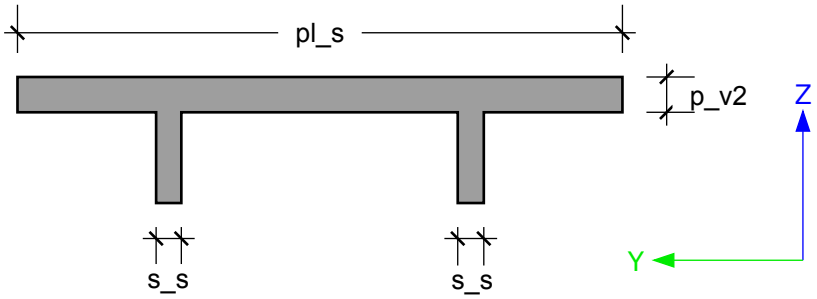
- n_l - dolžina nosilca.
- n_s1 - širina stojine.
- n_s2 - širina roba, odvisna od globine naleganja.
- n_h1 - višina nosilca.
- n_h2 - višina roba, odvisna od višine elementa, ki nalega.

π PLOŠČA (pl_pi):

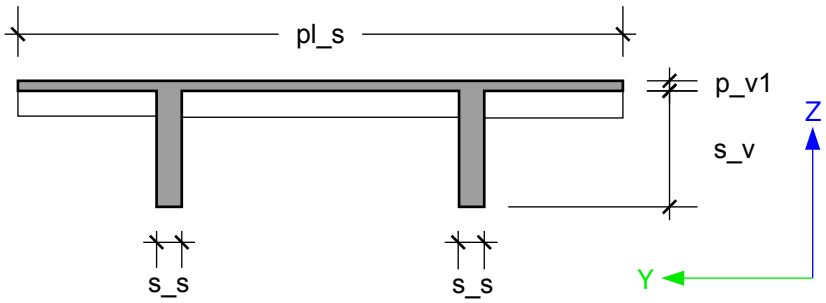
Pogled:



Prerez 1-1:



Prerez 2-2:

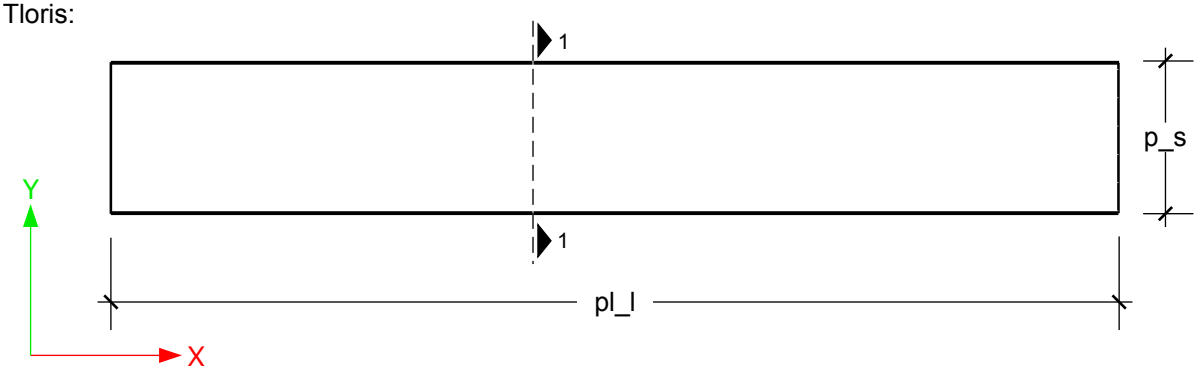


Dimenzije:

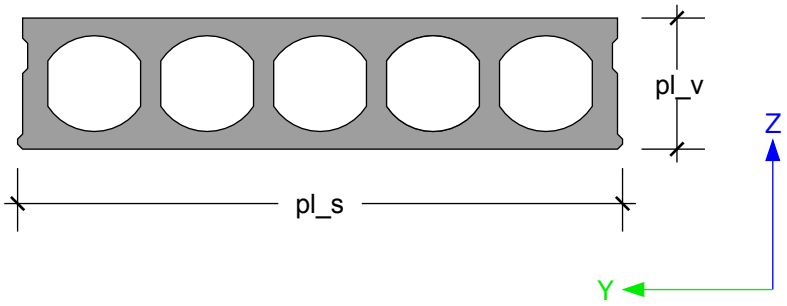
- pl_l - dolžina plošče.
- pl_p - dolžina prehoda od podpore na polje.
- pl_s - širina plošče.
- p_v1 - višina pasnice v polju.
- p_v2 - višina pasnice nad podporami.
- s_s - širina stojin.
- s_v - višina stojin v polju.

VOTLA PLOŠČA (pl_vot):

Tloris:



Prerez 1-1:



Dimenzije:

pl_l - dolžina plošče.
pl_s - širina plošče.
pl_v - višina plošče.*

*Večina ponudnikov nudi le plošče višine 16, 20, 26, 32, 40 in 50 cm, zato je možna izbira samo teh dimenzij. Vse ostale dimenzije razen širine in dolžine plošče so vezane na višino plošče in so dinamično spremenljive.

PRILOGA B: PROGRAMSKA KODA RUBY VTIČNIKOV

Vtičnik *Grid*:

```
require "sketchup.rb"

# Requires Grid positioning.rb and Create 3D grid.rb via a relative path.
$main_path = File.dirname(__FILE__)
require_all("#{$main_path}/Grid setup")


# Adds User Interface to the menus in SketchUp.
if( not file_loaded?("Grid positioning.rb") or not file_loaded?("Create 3D
grid.rb") ) then

  grid = Grid.new

  # Adding the tree structure to Plugins menu.
  plugins_menu = UI.menu "Plugins"
  grid_menu = plugins_menu.add_submenu("Grid setup")
  grid_menu.add_item("Draw a 3D grid") do
   grid.create_3d_grid
  end
  grid_menu.add_item("Position at grid intersections") do
   grid.grid_positioning(Sketchup.active_model.selection)
  end


  # Adding the tree structure to the selection options.
  UI.add_context_menu_handler do |context_menu|
    context_menu.add_separator
    grid_popup = context_menu.add_submenu("Grid setup")
    grid_popup.add_item("Draw a 3D grid") do
      grid.create_3d_grid
    end
    grid_popup.add_item("Position at grid intersections") do
      grid.grid_positioning(Sketchup.active_model.selection)
    end
  end

end

file_loaded("Create 3D grid.rb")
file_loaded("Grid positioning.rb")
```

Vtičnik *Create 3D Grid*:

```
class Grid

  def unit(num)
    # 4=m,3=cm,2=mm,1=feet,0=inches
    @unit = Sketchup.active_model.options["UnitsOptions"]["LengthUnit"]

    case
    when @unit == 0
      num.inch
    when @unit == 1
      num.feet
    when @unit == 2
      num.mm
    when @unit == 3
      num.cm
    when @unit == 4
      num.m
    else
    end

  end

  def create_3d_grid

    model = Sketchup.active_model
    model.start_operation "Create 2D Grid"
    entities = model.active_entities

    definitions_list = model.definitions
    grid_2d_comp_def = definitions_list.add("2D Grid")

   # Prompts the user to insert values and defines default values.
      prompts = ["Interval between axes (along X)", "Interval between axes
      (along Y)", "Height between levels/stories"]
      def_values = ["8.0;13.0;6.0", "12.0;15.0", "5.0"]
      results = inputbox prompts, def_values, "Grid parameters"
      interval_x_str, interval_y_str, interval_z_str = results

    if interval_x_str == nil or interval_y_str == nil or interval_z_str ==
     nil

    return UI.messagebox("Insert values.")

    end

    intervals_x = []
    intervals_y = []
    intervals_z = []

    # Writes floats from intervals strings to intervals arrays, the pattern
      is ; .
    intervals_x = interval_x_str.split(/\;/).map {|int| Float(int)}
    intervals_y = interval_y_str.split(/\;/).map {|int| Float(int)}
    intervals_z = interval_z_str.split(/\;/).map {|int| Float(int)}
```

```
# Checks if each array element is smaller or equal to 0.0 and if any
  is, pops up a message and end the method.
if intervals_x.each {|int| int <= 0.0} or intervals_y.each {|int| int
 <= 0.0} or intervals_z.each {|int| int <= 0.0} then
else
return UI.messagebox("Values must be positive.")
end

nu_x = intervals_x.length
nu_y = intervals_y.length
nu_z = intervals_z.length


# Scales the intervals to user-used units and calculates the dimensions
  of a cube which contains the 3D grid.
n = 0
length_x = 0
intervals_x.each do |interval_x|
intervals_x[n] = unit(interval_x)
length_x = length_x+intervals_x[n]
n += 1
end

n = 0
length_y = 0
intervals_y.each do |interval_y|
intervals_y[n] = unit(interval_y)
length_y = length_y+intervals_y[n]
n += 1
end

n = 0
length_z = 0
intervals_z.each do |interval_z|
intervals_z[n] = unit(interval_z)
length_z = length_z+intervals_z[n]
n += 1
end

pts = []
step = -1
x = 0.0
y = 0.0

# Finds start and finish points for edges and draws them into a
  component definition grid_2d_comp_def.
while x < length_x
  if step == -1 then
  pts[0] = Geom::Point3d.new(0, 0, 0)
  pts[1] = Geom::Point3d.new(0, length_y, 0)
  else
  x = x+intervals_x[step]
  pts[0] = Geom::Point3d.new(x, 0, 0)
  pts[1] = Geom::Point3d.new(x, length_y, 0)
  end
  line_y = grid_2d_comp_def.entities.add_edges(pts[0], pts[1])
  step += 1
end
```

```
    step = -1

    while y < length_y
      if step == -1 then
      pts[0] = Geom::Point3d.new(0, 0, 0)
      pts[1] = Geom::Point3d.new(length_x, 0, 0)
      else
      y = y+intervals_y[step]
      pts[0] = Geom::Point3d.new(0, y, 0)
      pts[1] = Geom::Point3d.new(length_x, y, 0)
      end
      line_y = grid_2d_comp_def.entities.add_edges(pts[0], pts[1])
      step += 1
    end

    model.commit_operation

    model.start_operation "Duplicate 2D grid"

    step = -1
    z = 0.0

    # Positions grid_2d_comp_def instances on different heights.
    while z < length_z
      if step == -1 then
      grid_2d_instance = entities.add_instance grid_2d_comp_def, [0, 0, 0]
      else
      z = z+intervals_z[step]
      grid_2d_instance = entities.add_instance grid_2d_comp_def, [0, 0, z]
      end
      step += 1
    end

    model.commit_operation

    # Zooms to include the entire model (grid).
    view = Sketchup.active_model.active_view
    view_all = view.zoom_extents

  end

end
```

Vtičnik *Grid positioning*:

```
require "Point on edge.rb"

class Grid

  # Takes the @intersection_points array and positions the chosen component
    at all the array points.
  def position_at_intersection(intersection_points)

    # Opens up an explorer panel and lets you choose the component
      definition and saves its path.
    component_path = UI.openpanel("Open Component File", "#{$main_path}/DC
     Library", "*.skp")
    component_def = Sketchup.active_model.definitions.load(component_path)

    intersection_points.each do |point|
      Sketchup.active_model.active_entities.add_instance(component_def,
      point)
    end

  end

  # Gets edges array, the number of edges n_end and the instance in which
    the edges are located. Processes them depending
  # on the instance class, finds points where edges intersect and writes
    these points into @intersection_points.
  def find_intersection_points(edges, n_end, instance)

    intersection_points = []

    # Takes one edge at a time and finds intersections with all other
      lines.
    edges.each do |current_edge|
      n = 0
      i = edges.index(current_edge)

      until n == n_end do

        # Checks if the current_edge is different from the edge it should
           intersect.
        # If they are the same edge, does nothing, else finds their
           intersection point.
        if i == n  then
          # Do nothing.
        else
          intersection_point = Geom.intersect_line_line(current_edge.line,
           edges[n].line)

          # Checks if such an intersection already exists or if there is no
            intersection at all (nil).
          # If its a new, not-nil intersection, pushes it into the
            intersection_points array.
          if (intersection_points.any? {|previous_point| previous_point ==
             intersection_point} == true or intersection_point == nil)
             then
            # Do nothing
```

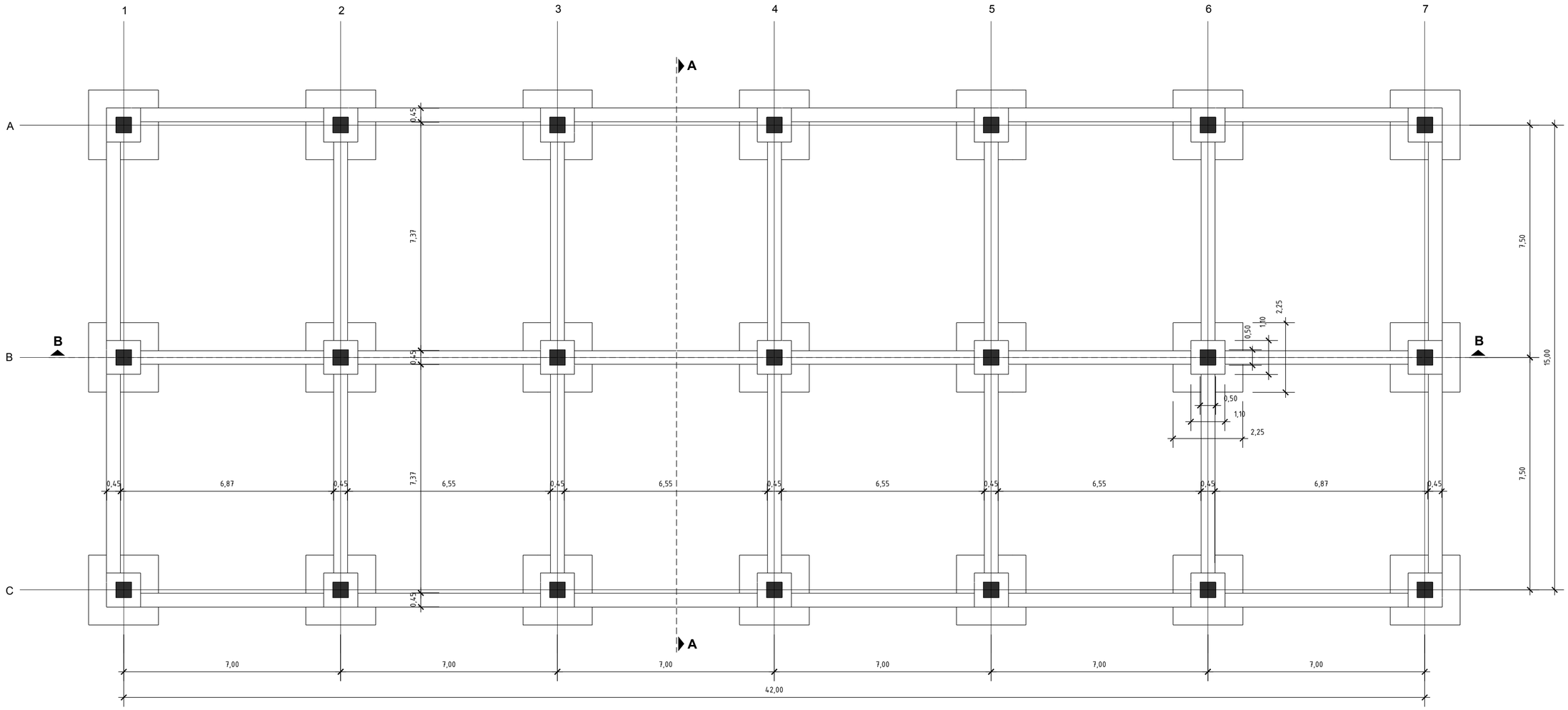
```
        else

          # Checks if intersection_point is on current_edge and on
            edges[n], which are the two addressed edges, with
          # on_edge? method from Point on edge.rb.
          # If the point is on both edges, writes it into
            intersection_point array.
          if intersection_point.on_edge?(current_edge) == true and
          intersection_point.on_edge?(edges[n]) == true then
            intersection_points.push intersection_point
          else
            # Do nothing
          end
        end
      end

      n += 1

    end
  end

  # Examines the instance's class and if it's a Group or a
    ComponentInstance, moves/transforms all the points with the
  # trans_vector.inverse vector. trans_vector is a vector from the
    group/component instance origin to the global model origin.
  intersection_points.each do |point|
    case
    when instance.class == Sketchup::Group
      trans_vector =
      Geom::Transformation.translation(instance.transformation.origin.vec
      tor_to(ORIGIN))
      point_trans = point.transform(trans_vector.inverse)

      @intersection_points.push point_trans

    when instance.class == Sketchup::ComponentInstance
      trans_vector =
      Geom::Transformation.translation(instance.transformation.origin.vec
      tor_to(ORIGIN))
      point_trans = point.transform(trans_vector.inverse)

      @intersection_points.push point_trans

    else
      @intersection_points.push point
    end
  end
end

# Goes through the entire everything array, defines each instance's class
   and sends its edges to find_intersection_points.
# After find_intersection_points processes the edges this method recieves
   the @intersection_points array and informs the user
# if there are no edges in the selection or if these edges don't
   intersect.
def find_edges_and_points(everything)

  # Defines all the necessary arrays which are used in this plugin.
```

```
normal_edges = []
normal_edges_n = 0
group_edges = []
group_edges_n = 0
component_edges = []
component_edges_n = 0

@intersection_points = []

# Goes through the entire everything array and processes each instance
  according to its class.
# Regardless the class every single edge array is sent to
  find_intersection_points.
everything.each do |instance|
  case
  when instance.class == Sketchup::Edge
    normal_edges.push instance
    normal_edges_n += 1

  when instance.class == Sketchup::Group
    group_entities = instance.entities

    group_edges = []
    group_edges_n = 0

    group_entities.each do |group_entity|
      if group_entity.class == Sketchup::Edge
        group_edges.push group_entity
        group_edges_n += 1
      end
    end
    # Sends all the edges from this group to find_intersection_points.
    find_intersection_points(group_edges, group_edges_n, instance)

  when instance.class == Sketchup::ComponentInstance
    component_entities = instance.definition.entities

    component_edges = []
    component_edges_n = 0

    component_entities.each do |component_entity|
      if component_entity.class == Sketchup::Edge
        component_edges.push component_entity
        component_edges_n += 1
      end
    end
    # Sends all the edges from this component instance to
       find_intersection_points.
    find_intersection_points(component_edges, component_edges_n,
    instance)

  end
end

# If no edges were selected, pops up messagebox and ends
  find_edges_and_points method.
if normal_edges.empty? and group_edges.empty? and
 component_edges.empty?
```

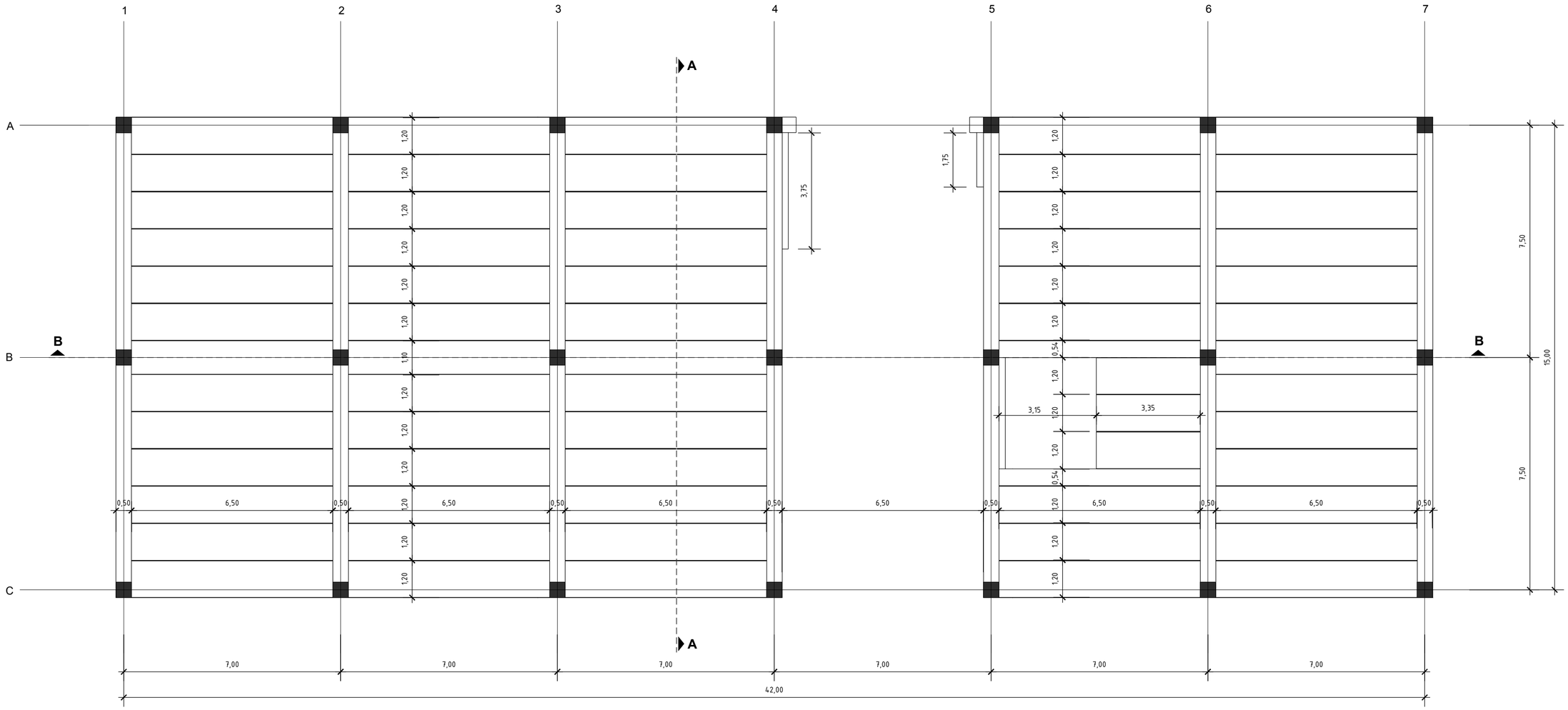
```
      return UI.messagebox("No edges were selected.")
    end

    # Sends all the "normal" edges in the model to
      find_intersection_points.
    find_intersection_points(normal_edges, normal_edges_n, nil)


    # If no edges in the model/same group/same component instance
      intersect, pops up messagebox and ends find edges and points method.
    if @intersection_points.empty?
      return UI.messagebox("Edges do not intersect.")
    end

  end


  # Sends selection_everything to find_edges_and_points, recieves
     @intersection_points array from it and sends
  # it to position_at_intersection.
  def grid_positioning(selection_everything)

    find_edges_and_points(selection_everything)

    position_at_intersection(@intersection_points)

  end
end
```

Vtičnik *Point on edge*:

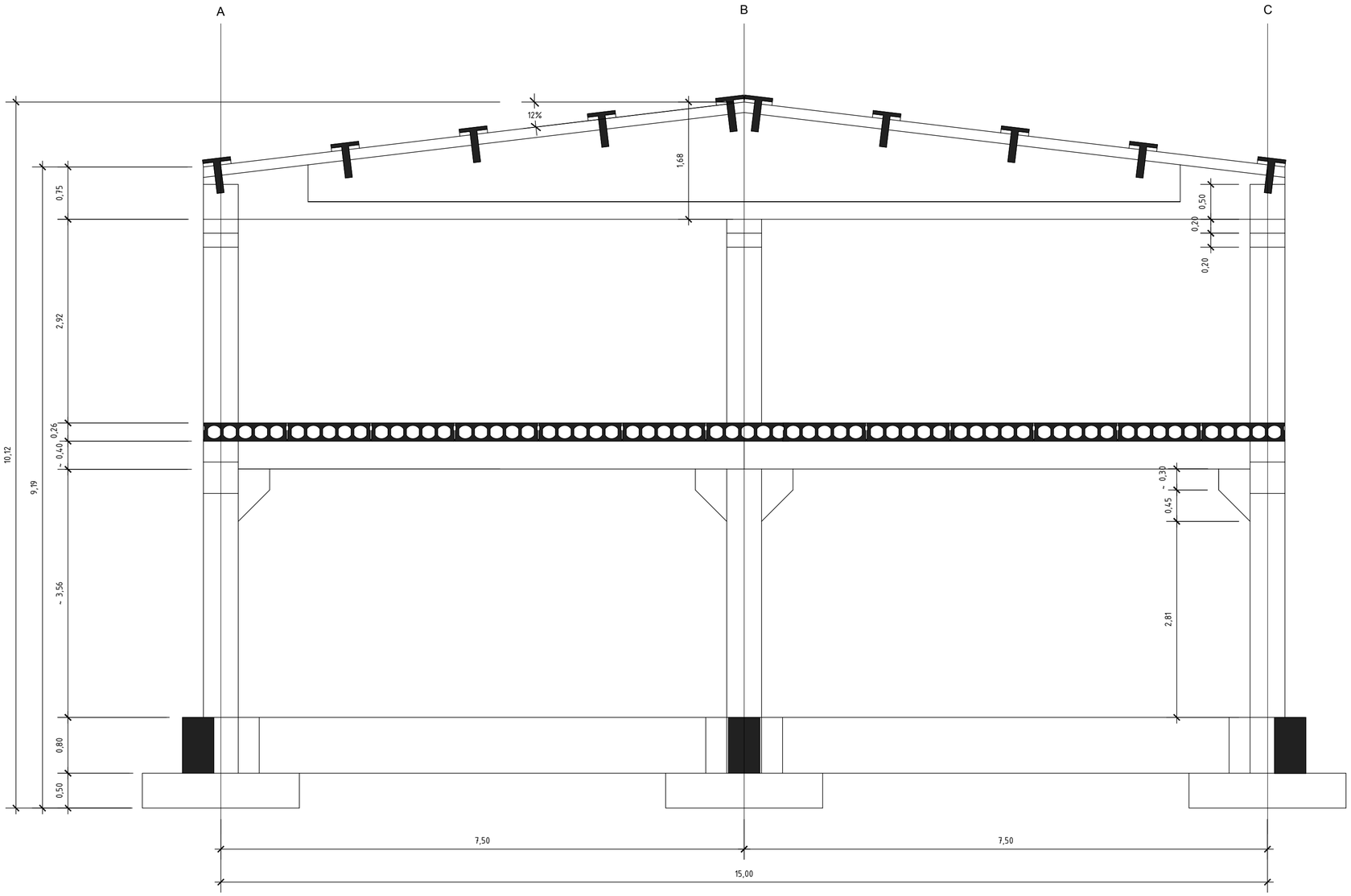
```
# Opens class Geom::Point3d.
class Geom::Point3d

  # Adds a method called on_edge? to the Geom::Point3d class, which returns
     true if our point
  # is on the edge and false if not.
  def on_edge?(edge)

    # Starting and ending point3d of the edge.
    pt_start = edge.start.position
    pt_end = edge.end.position

    # self is a point3d which we are testing.
    # self.distance(pt_start) is the distance from our point to the start
      point of the edge.
    # self.distance(pt_end) is the distance from our point to the end point
      of the edge.
    d1 = (self.distance(pt_start) + self.distance(pt_end))

    # d2 is the length of the edge.
    d2 = pt_start.distance(pt_end) + 0

    # Tests if our point is on the edge (tolerates a really small offset).
    if (d1 <= d2) or (d1-d2 < 1e-10)
      return true
    else
      return false
    end

    # Safe-lock.
    return UI.messagebox("Something is wrong.")

  end
end
```



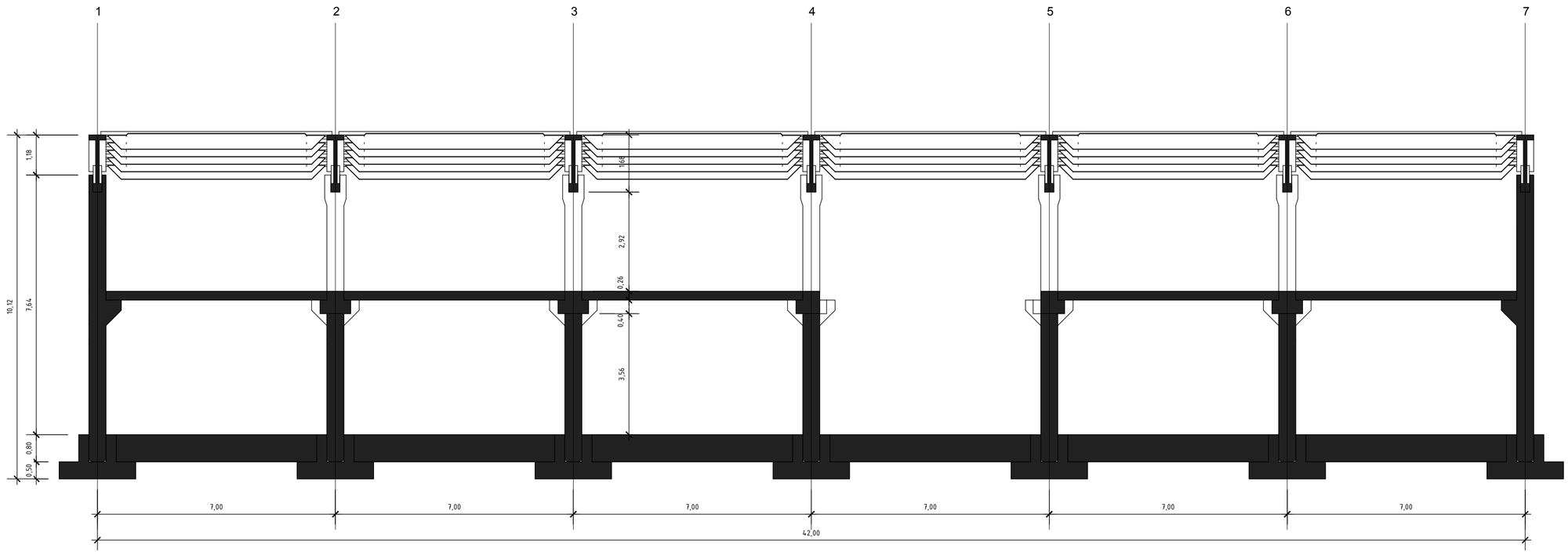
Projekt: Industrijski montažni objekt; IRIO	Izdelal: Jure Česnik	
Vsebina načrta: Tloris pritličja; montažna konstrukcija	Datum izdelave: 29.8.2012	
Univerza v Ljubljani Fakulteta za gradbeništvo in geodezijo	Merilo: M 1:100	Stran načrta: 1/5



Projekt: Industrijski montažni objekt; IRIO	Izdelal: Jure Česnik	
Vsebina načrta: Tloris nadstropja; montažna konstrukcija	Datum izdelave: 29.8.2012	
Univerza v Ljubljani Fakulteta za gradbeništvo in geodezijo	Merilo: M 1:100	Stran načrta: 2/5

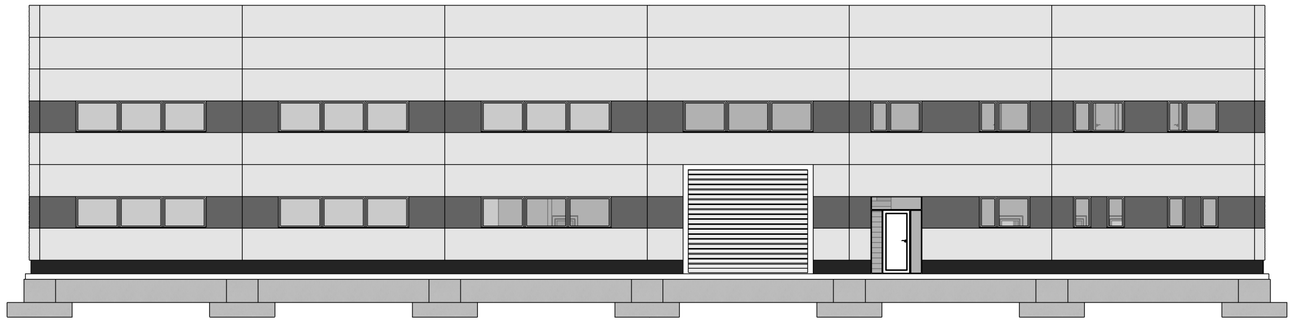


Projekt: Industrijski montažni objekt; IRIO	Izdelal: Jure Česnik	
Vsebina načrta: Prerez A-A; montažna konstrukcija	Datum izdelave: 29.8.2012	
Univerza v Ljubljani Fakulteta za gradbeništvo in geodezijo	Merilo: M 1:50	Stran načrta: 3/5

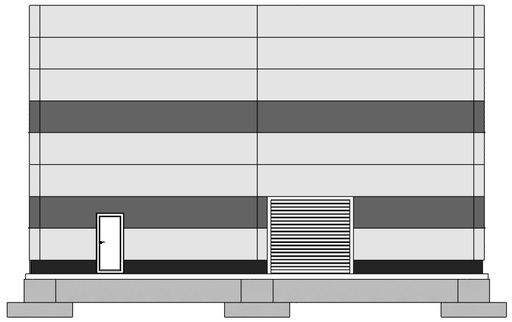


Projekt: Industrijski montažni objekt; IRIO	Izdelal: Jure Česnik	
Vsebina načrta: Prerez B-B; montažna konstrukcija	Datum izdelave: 29.8.2012	
Univerza v Ljubljani Fakulteta za gradbeništvo in geodezijo	Merilo: M 1:100	Stran načrta: 4/5

Vzdolžna vhodna fasada:

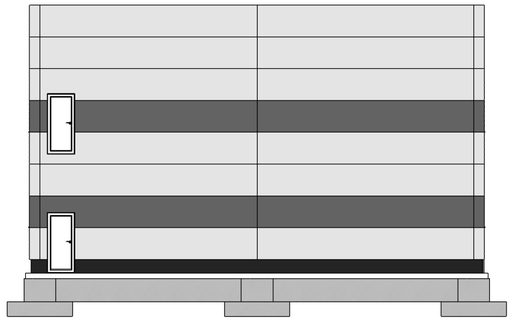


Prečna fasada 1:

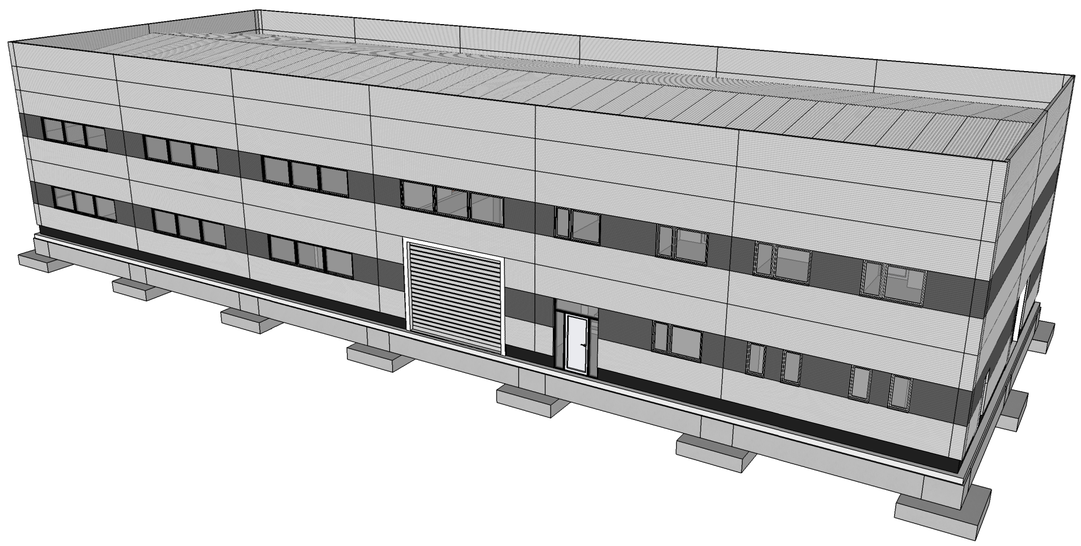


Vzdolžna zadnja fasada:

Prečna fasada 2:



3D vizualizacija:



Projekt: Industrijski montažni objekt; IRIO	Izdelal: Jure Česnik	
Vsebina načrta: Fasade, vizualizacija	Datum izdelave: 29.8.2012	
Univerza v Ljubljani Fakulteta za gradbeništvo in geodezijo	Merilo: M 1:150	Stran načrta: 5/5