Sadržaj

Uvod

Zupčanicima i sličnim mehaničkim komponentama moguće je sastaviti analogno računalo. Ovaj rad opisuje implementaciju programa koji sastavlja virtualno analogno mehaničko računalo bilo koje racionalne matematičke funkcije. Osim generiranja mehanizma, program ga također simulira tako da se na virtualnom crtaču iscrtava graf te matematičke funkcije.

U prvom poglavlju opisuju se mehaničke komponente koje predstavljaju jednostavne matematičke operacije. U sljedećem poglavlju opisuje se sintaksno parsiranje koje je potrebno za parsiranje upisane matematičke funkcije. Treće poglavlje opisuje relevantne teme iz područja računalne grafike koje su potrebne za implementaciju ovog programa.

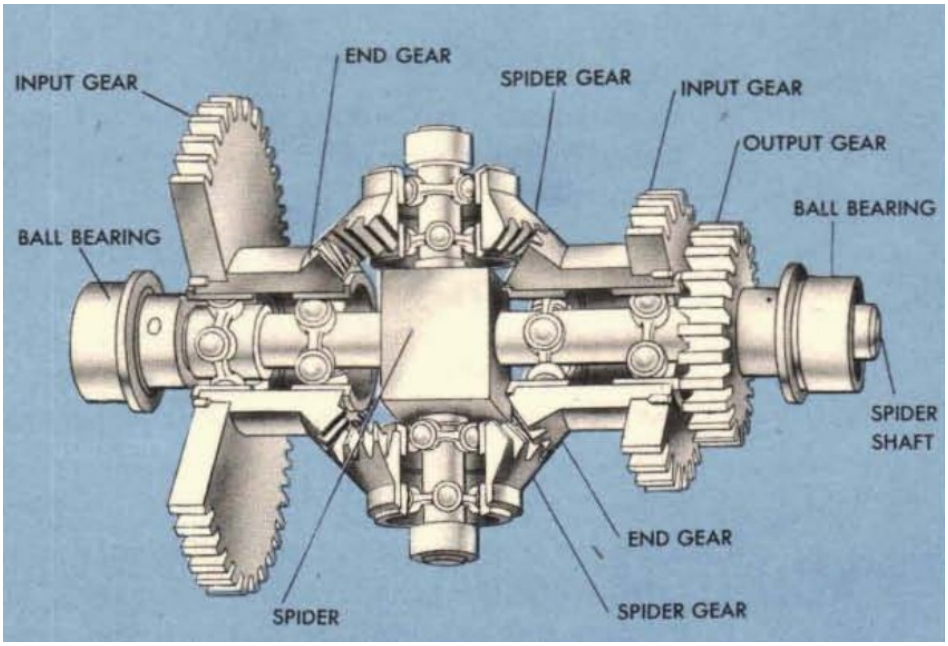
U četvrtom poglavlju opisana je implementacija proceduralnog generiranja 3D modela zupčanika i letvica varijabilnih veličina. Zatim je u sljedećem poglavlju opisano simuliranje i interakcija zupčanika i letvica. U šestom poglavlju opisane su virtualne komponente iz prvog poglavlja, te kako su one implementirane. Sedmo poglavlje opisuje implementaciju sintaksnog parsera matematičke funkcije, te potrebne transformacije dobivenog sintaksnog stabla za jednostavnije generiranje konačnog mehanizma koje je opisano u osmom poglavlju. Na kraju, u devetom poglavlju, prikazan je rad programa za tri različita primjera matematičke funkcije.

1. Mehaničke komponente matematičkih operacija

Postoji više načina za kodiranje vrijednosti u zupčanicima. Za potrebe ovog rada vrijednost se kodira kao broj okretaja zupčanika od početne rotacije. Za modeliranje i izračun racionalnih funkcija, potrebno je mehanički realizirati pet matematičkih operacija. To su zbrajanje, oduzimanje, množenje, dijeljenje i potenciranje. Sve mehaničke komponente iskorištene su iz [1]. Operaciju zbrajanja moguće je realizirati diferencijalom. Istu mehaničku komponentu se može iskoristiti za oduzimanje ako se desni pribrojnik pomnoži s minus jedan. Za množenje i dijeljenje iskorišteni su mehanički množitelji (engl. *multiplier*). Pošto su racionalne funkcije sastavljene od polinoma, a polinomi imaju samo cjelobrojne potencije varijable x, onda se potenciranje može predstaviti nizom množenja.

1.1. Diferencijal

Diferencijal se koristi za zbrajanje rotacije dva zupčanika, pa tako i za zbrajanje dvije vrijednosti. Sastoji se od: (Slika 1) dva ulazna zupčanika (engl. *input gear*), izlaznog zupčanika (engl. *ouput gear*), dva prijenosna zupčanika (engl. *end gears*), dva planetarna konična zupčanika (engl. *spider gears*) i osovine oblika križa (engl. *spider shaft*).



Slika 1 Dijagram diferencijala [1]

Input zupčanici su čvrsto povezani s njihovim *end* zupčanicima, odnosno okretanjem *input* zupčanika direktno se okreče i *end* zupčanik. Planetarni *spider* zupčanici su zaduženi za

okretanje *spider* osovine, odnosno planetarnim okretanjem *spider* zupčanika okreće se i *spider* osovina. *Output* zupčanik čvrsto je povezan sa *spider* osovinom, odnosno okretanjem *spider* osovine, direktno se okreće i *output* zupčanik.

Način rada diferencijala

Rad diferencijala lakše je shvatiti pojednostavljenjima. Ako su se oba ulaza okrenula isti broj puta (okreću se istom brzinom), *spider* zupčanici se neće okretati oko svojih osi, odnosno neće se kotrljati po *end* zupčanicima nego će biti zaključani za njih i okretati se planetarno oko *spidera*, istom brzinom kao i ulazni zupčanici. Tako će se *spider* osovina, pa i *output* zupčanik okrenuti isti broj puta kao i ulazni zupčanici. Važno je primijetiti da se *output* zupčanik okrene duplo manje puta nego zbroj koji se traži, pa je na njega potrebno dodati još jedan zupčanik s duplo manje zubaca da se dobije točan zbroj.

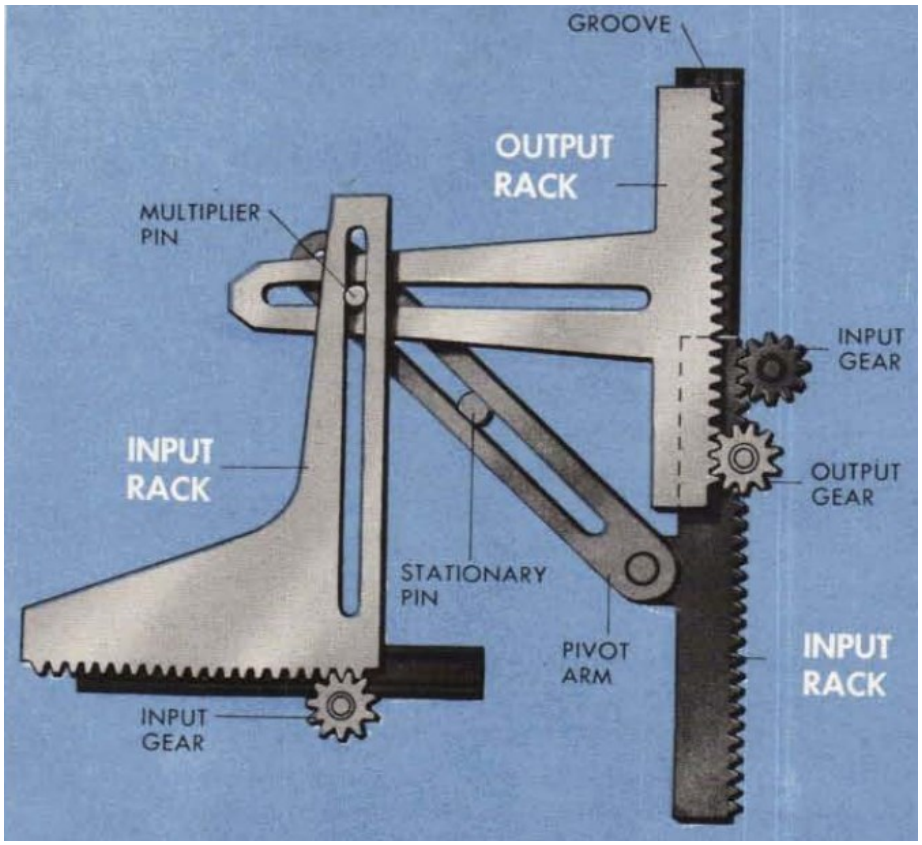
U slučaju da se oba ulaza okreću istom brzinom ali u suprotnim smjerovima, *spider* zupčanici će se okretati oko svojih osi, ali će planetarno stajati na mjestu, odnosno neće okretati *spider* osovinu, pa će rezultat biti nula.

Vidljivo je sada kako okretanjem jednog ulaza brže ili sporije od drugog okreće *spider* zupčanike i oko svoje i oko planetarne osi, te se tako dobije točan zbroj na izlaznom zupčaniku.

1.2. Množitelj i djelitelj

Množitelj se koristi za računanje operacije množenja i dijeljenja. Sastoji se od (Slika 2): dvije ulazne zupčaste letve (engl. *input racks*), izlazne zupčaste letve (engl. *output rack*), stacionarnog *pina* (engl. *stationary pin*), zakretne ruke (engl. *pivot arm*) i *pina* za množenje (engl. *multiplier pin*).

Pivot arm se slobodno može okretati i pomicati oko *stationary pina*, a povezan je s desnim *input rackom* koji mu kontrolira rotaciju. *Multiplier pin* prolazi kroz tri utora, dva na *rackovima* i jedan na *pivot armu*. Bit *multiplier pina* je da pomiče *output rack* kako se pomiču dva *input racka*.



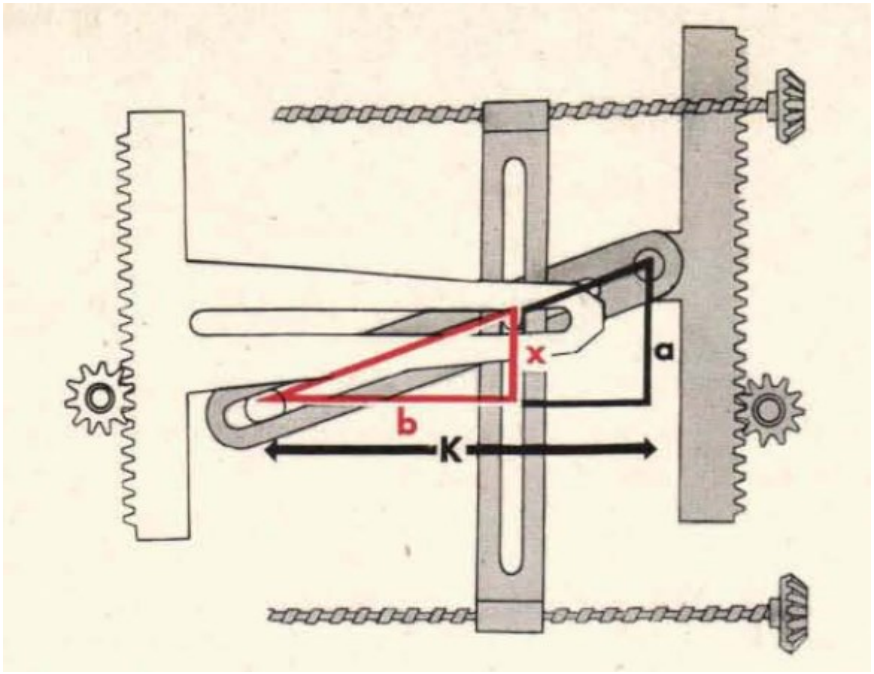
Slika 2 Dijagram množitelja [1]

Način rada množitelja i djelitelja

Množitelj računa operaciju množenja uz pomoć sličnih trokuta. Trokuti su pravokutni trokuti između *stationary pina* i stožera *pivot arma* kao što je prikazano na dijagramu (Slika 3). Duljina donje katete manjeg trokuta je pomak donjeg *input racka* od *stationary pina*, a duljina desne katete je pomak *output racka* od *stationary pina*. Duljina donje katete većeg trokuta je konstanta, odnosno udaljenost *stationary pina* i stožera *pivot arma*, a duljina desne katete je pomak bočnog *input racka* od *stationary pina*. Sada se po poučku proporcionalnosti stranica sličnih trokuta dobiva formula (1).

$$\frac{x}{b} = \frac{a}{K} \Rightarrow x = \frac{ba}{K} \tag{1}$$

Vidi se da je rezultat (pomak *output racka* - *x*) umnožak ulaza (pomak *input rackova* - *a* i *b*), ali je podijeljen s konstantom *K*, tako da je potrebno na *output rack* staviti zupčanik određene veličine da se izlazna vrijednost smanji za faktor *K*.



Slika 3 Dijagram načina rada množitelja [1]

Također je važno primijetiti da množitelj ima fizička ograničenja na veličinu vrijednosti ulaza jer je ograničen dužinom *rackova*, za razliku od diferencijala, a rješenje tog problema objašnjeno je kasnije.

Dijeljenje je postignuto tako da se zamjene jedan ulazni *rack* i izlazni *rack*. S tim da je zbog jednostavnosti prvo izračunata recipročna vrijednost nazivnika, pa je zatim rezultat toga pomnožen s brojnikom. Dijeljenje odnosno računanje recipročne vrijednosti ima još veća fizička ograničenja, osim apsolutnog gornjeg limita ima i apsolutni donji limit, te nije moguće dijeliti s nulom, odnosno proći kroz nju tijekom mijenjanja predznaka.

2. Sintaksno parsiranje

Da bi se generirao mehanizam zupčanika s prije spomenutim komponentama, potrebno je parsirati dobiveni matematički izraz odnosno funkciju. Izraz je potrebno pretvoriti u sintaksno stablo gdje su čvorovi matematičke operacije, a listovi konstante ili varijabla *x*.

Postoje različite tehnike sintaksnog parsiranja [2]:

- odozgo prema dolje (engl. *top-down*)
  - prediktivno (engl. *predicting parsing*) – LL(1)
  - **metoda rekurzivnog spusta** (engl. *recursive descent parsing*)
- odozdo prema gore (engl. *bottom-up*)

Razlika *top-down* i *bottom-down* parsiranja je što *top-down* generira stablo s vrha, odnosno od korijenskog čvora, dok *bottom-up* generira stablo počevši od listova.

Gramatika sintaksnog parsera sastoji se od jedne ili više produkcija. Produkcija se sastoji od nezavršnog znaka gramatike koji prelazi u niz završnih ili nezavršnih znakova. Primjer jedne produkcije je:

$$P \rightarrow F \text{ "^" } P \tag{2}$$

U produkciji (2) nezavršni znakovi su *P* i *F*, dok je završni znak "^". Završni znak predstavlja list u sintaksnom stablu, a nezavršni znakovi predstavljaju čvorove, te se stablo generira po produkcijama gramatike.

Metoda rekurzivnog spusta parsira ulazni niz tako da za svaki nezavršni znak postoji funkcija koja primjenjuje njegove produkcije. Za završne znakove produkcije funkcija konzumira znak na ulazu, dok za nezavršne znakove rekurzivno poziva funkcije koje ih predstavljaju.

Za parsiranje matematičkih izraza potrebno je sastaviti gramatiku odnosno niz produkcija koji točno predstavljaju sintaksu matematičkih izraza.

Pošto neki matematički operatori imaju prednost nad drugim te mogu biti lijevo ili desno asocijativni, važno je točno sastaviti produkcije gramatike da to i poštuju. Tako je Theodore Norvell definirao sljedeća pravila po kojima je sastavio gramatiku [3]:

- zagrade imaju prednost nad svim ostalim operatorima
- potenciranje ima prednost nad unarnim - i binarnim operatorima /, *, - i +
- * i / imaju prednost nad unarnim - i binarnim - i +

- unarni - ima prednost nad binarnim - i +
- potenciranje je desno asocijativno dok su svi ostali binarni operatori lijevo asocijativni

Zatim je poštivajući ta pravila sastavio gramatiku koju je moguće implementirati metodom rekurzivnog spusta:

$$E \rightarrow T\ \{\ (" + " \mid " - ")\ T\ \} \tag{3}$$

$$T \rightarrow P\ \{\ (" * " \mid "/" \mid "")\ P\ \} \tag{4}$$

$$P \rightarrow F\ "\hat{}"\ P \mid F \tag{5}$$

$$F \rightarrow " - "\ T \mid "("\ E\ ")" \mid a \tag{6}$$

gdje {} zagrade označavaju da se izraz u njima ponavlja nula ili više puta, a izraz *a* označava konstantu, odnosno decimalni broj. Produkcija (3) je početna produkcija te označava *expression* koji se sastoji od niza zbrajanja ili oduzimanja *termova*. Pošto je to prva produkcija, bit će i najplića u sintaksnom stablu pa će tako imati najmanju prednost. Također je pravilo lijeve asocijativnosti poštovano pri implementaciji rekurzivne metode tog nezavršnog znaka. Na isti način je definiran i *term* (4) koji se sastoji od niza množenja i dijeljenja *powera*. Moguće je i izostaviti znak operatora pa se to smatra kao množenje tako da je moguće parsirati izraz $2x$. Nezavršni znak *power* (5) definiran je desnom rekurzijom, tako da poštuje pravilo lijeve asocijativnosti. Na kraju je definirana produkcija za *factor* (6) koji može biti negirani *term* (unarni -), *expression* između zagrada ili konstanta.

3. Relevantne teme iz područja računalne grafike

3.1. Modeliranje i digitalni zapis virtualnih predmeta

Za modeliranje predmeta u memoriji računala postoji više metoda, a sve se nalaze u spektru između dvije krajnosti [4]: parametarskih metoda i metoda jediničnih elemenata.

Parametarske metode za digitalni zapis koriste predefinirane funkcije ili oblike, koje se onda preciznije oblikuju parametrima. Na primjer definirana je funkcija kugle čiji su parametri radijus i centar u 3D prostoru.

Metoda jediničnih elemenata koristi poligone ili male 3D ćelije (engl. *volume elements, voxels*), smještene u trodimenzionalnom prostoru koji zajedno čine neki veći model.

Prikaz geometrije poligonima najčešći je način zapisa virtualnih predmeta u računalu [4]. Danas je većina sklopovlja optimizirana za iscrtavanje virtualnih scena iz niza poligona i to najčešće trokuta. Modeli su sastavljeni od niza trokuta, a svaki trokut se sastoji od 3 vrha (engl. *vertex*) i 3 brida (engl. *edge*). Točan zapis u memoriji je da se prvo za sve vrhove zapišu njihove tri koordinate, a zatim se za svaki trokut zapišu 3 indeksa prije zapisanih vrhova koji čine taj trokut.

Pošto je nezgodno ručno zapisivati koordinate vrhova, najčešće se koriste pomoćni alati koji to rade automatski uz navođenje dizajnera. No nekad je to potrebno raditi parametarski odnosno proceduralno u stvarnom vremenu izvođenja programa (engl. *real time*) i to se zove proceduralna generacija (engl. *procedural generation*). U ovom radu je ta tehnika potrebna za generiranje modela zupčanika bilo koje zadane veličine (radijusa, debljine, broja zubaca, veličine zubaca…).

3.2. Programski alat Unity

Unity je *game engine* koji podržava niz platformi poput Windows, Mac OS, Android, iOS i sl. Unity je napisan u C++-u, ali kao skriptni jezik koristi C#. Glavna komponenta simulacije je scena koja sadrži hijerarhiju objekata, svaki objekt može sadržavati više komponenti koje definiraju ponašanje tog objekta [5]. Komponente mogu biti već postojeće poput *Transform* komponente koju ima svaki objekt te mu ona opisuje poziciju, rotaciju i veličinu u odnosu na roditelja, ili mogu biti skripte koje je korisnik sam isprogramirao.

3.2.1. Digitalni zapis virtualnih predmeta u alatu Unity

Za zapis virtualnih predmeta u memoriji unutar programskog alata Unity koristi se klasa `Mesh` [6]. Glavni atributi klase su `vertices` i `triangles`. Atribut `vertices` je niz koordinata vrhova, a atribut `triangles` je niz indeksa vrhova koji sačinjavaju trokute, te njegova veličina treba biti djeljiva s 3 jer se svaki trokut sastoji od tri vrha.

`Mesh` klasa također sadrži još neke atribute poput `normals`, `colors`, `uv`, koji su također nizovi iste dužine kao i atribut `vertices`, jer i-ti element tih nizova opisuje i-ti vrh u nizu `vertices`. Atribut `normals` sadrži vektore koji definiraju smjer normale (smjer orijentacije) svakog vrha, a ta se informacija koristi tijekom sjenčanja trokuta koji sadrže taj vrh. Atribut `colors` sadrži boje vrhova, a atribut `uv` sadrži koordinate teksture svakog vrha.

3.2.2. Simulacijska petlja u alatu Unity

U Unityu svaka komponenta ima mogućnost pretplate na događaje koji se pozivaju u različitim trenutcima [7].

Pri kreiranju objekta, prvo se na svim njegovim komponentama pozivaju `Awake` metode, a zatim `Start` metode koje služe za inicijalizaciju stanja komponenti.

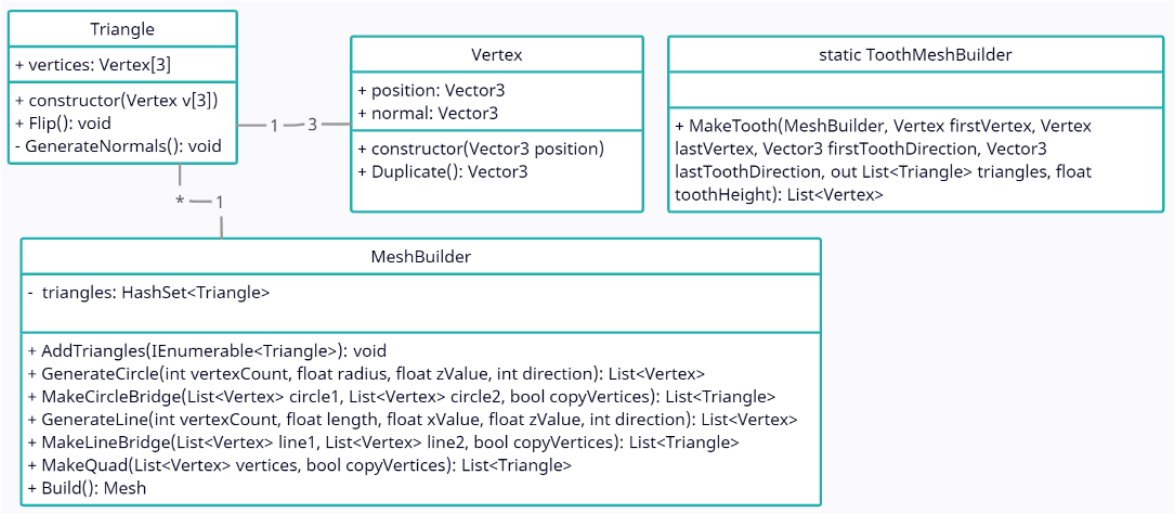
Zatim, dok je simulacija pokrenuta, prije iscrtavanja svake slike, pozivaju se metode `Update` svake komponente, u kojima se implementira ponašanje i logika simulacije. Metoda `Update` poziva se brzinom iscrtavanja slika na ekran, odnosno što je brže moguće ako nije postavljena granica na maksimalni broj slika u sekundi. Koristeći `Time.deltaTime` moguće je dobiti informaciju koliko je sekundi (i milisekundi) prošlo od zadnjeg poziva `Update` metode.

Također postoji i `FixedUpdate` metoda koja se ne poziva brzinom iscrtavanja slika nego se poziva određen broj puta u sekundi (npr. 50), tako da postoji mogućnost da se neće pozvati prije svakog poziva `Update` (ako je broj slika u sekundi veći od 50) ili će se pozvati više puta prije poziva `Update` (ako je broj slika u sekundi manji od 50). U metodi `FixedUpdate` najbolje je implementirati fiziku jer će biti deterministička jer je `Time.deltaTime` konstantan i kontroliran broj.

4. Proceduralno generiranje 3D modela mehaničkog crtača matematičkih funkcija

Za modeliranje matematičkih funkcija potrebno je imati mogućnost napraviti zupčanik bilo koje veličine, znači nije moguće imati skup prije izmodeliranih zupčanika i letvica, nego ih je potrebno izgenerirati tijekom izvođenja programa ovisno o zadanoj matematičkoj funkciji. Za to se koristi tehnika proceduralne generacije.

Pošto se pri generiranju zupčanika i letvica koriste slične funkcije dobro je napraviti dvije pomoćne klase `MeshBuilder` i `ToothMeshBuilder` koje prikazuje Slika 4.



Slika 4 UML dijagram klasa MeshBuilder i ToothMeshBuilder

Klasa `MeshBuilder` se instancira za generiranje novog modela, ona sadrži listu razreda `Triangle` koji predstavljaju pojedinačne trokute. Konstruktor klase `Triangle` prima tri `Vertexa`, te poziva metodu `GenerateNormals` koja generira normale sva tri *vertexa* da pokazuju u smjeru normale trokuta. Metoda `Flip` okreće te normale u suprotan smjer.

Klasa `MeshBuilder` ima niz metoda za lakše generiranje modela. Prva je `GenerateCircle` koja generira niz *vertexa* koji čine kružnicu. Kružnica se generira na x i y osima, dok je z os konstantna i postavljena na `zValue`, a `direction` je smjer (kazaljka na satu -1, suprotan 1).

Druga metoda je `MakeCircleBridge` koja prima dva niza *vertexa* koji zatvaraju kružnicu, te kreira trokute između te dvije 'kružnice' spajajući ih. Argument

`copyVertices` određuje hoće li se *vertexi* svakog trokuta duplicirati prije kreiranja trokuta.

Treća metoda `GenerateLine` generira niz *vertexa* koji čine liniju. Parametri `xValue` i `zValue` određuju x i z koordinate, a linija se stvara u smjeru y osi. Argument `direction` je isti kao kod metode `GenerateCircle`.

Sljedeća metoda `MakeLineBridge` radi istu stvar kao `MakeCircleBridge`, osim što spaja dvije linije umjesto kružnice.

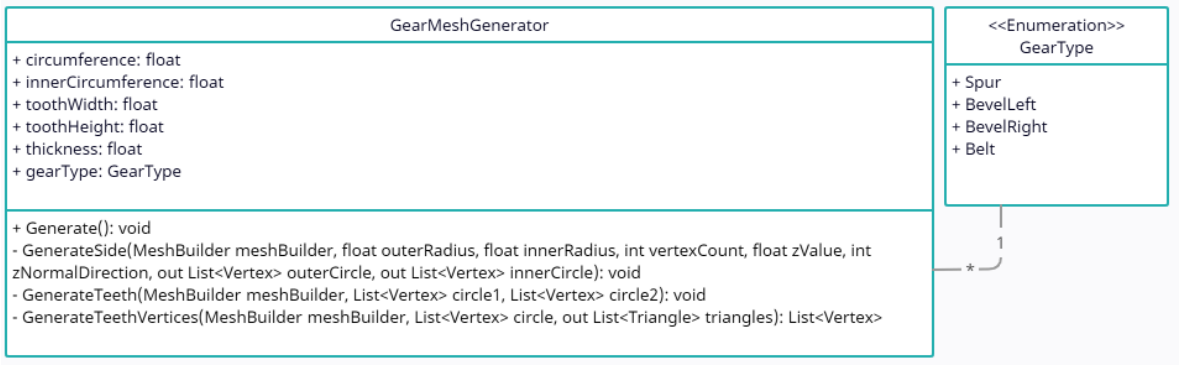
Metoda `MakeQuad` prima 4 *vertexa* te generira dva trokuta koji ih spajaju tako da čine četverokut.

Metoda `Build` kreira sami `Mesh`, iz svojih trokuta, koji će se predati grafičkoj kartici na iscrtavanje.

Klasa `ToothMeshBuilder` je pomoćna klasa za generiranje zubaca. Metoda `MakeTooth` kao argument prima dva *vertexa* iz kojih generira zubac dodavanjem 4 nova *vertexa* te generiranjem 2 četverokuta između njih i dobivenih *vertexa*.

4.1. Generiranje modela zupčanika

Proceduralno generiranje modela zupčanika implementirano je u klasi `GearMeshGenerator` čiji UML dijagram prikazuje Slika 5.



Slika 5 UML dijagram klase GearMeshGenerator

`GearMeshGenerator` sadrži javne atribute koji opisuju parametre zupčanika koji će se izgenerirati, a to su:

- `circumference` – opseg zupčanika bez zubaca
- `innerCircumference` – opseg rupe za osovinu

- `toothWidth` – širina pojedinog zupca po opsegu zupčanika
- `toothHeight` – visina zubaca
- `thickness` – debljina zupčanika
- `gearType` – tip zupčanika koji označava je li to običan zupčanik (`Spur`), zupčanik nagnut na lijevo ili desno (`BevelLeft` ili `BevelRight`) ili zupčanik bez zubaca namijenjen za remen (`Belt`)

Za generiranje zupčanika potrebno je pozvati javnu metodu `Generate`, koja koristi prije opisane klase `MeshBuilder` i `ToothMeshBuilder`, kao i svoje privatne metode za pomoć pri generiranju modela kojeg onda predaje radnom okviru za iscrtavanje na ekran.

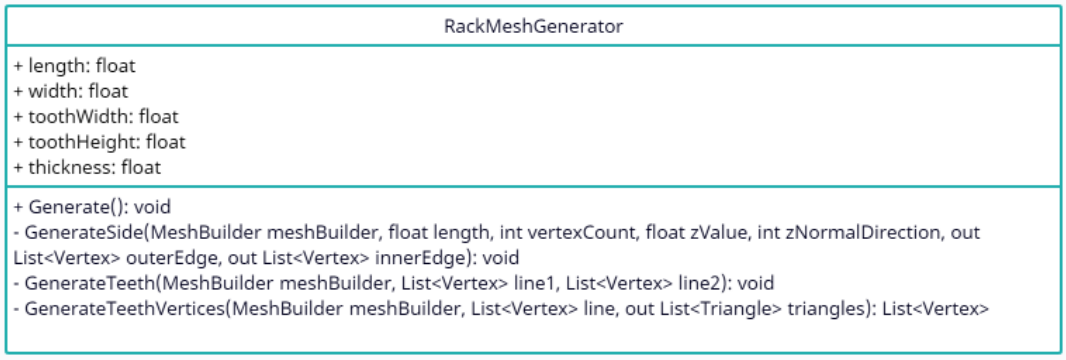
Metoda `GenerateSide` generira jednu stranicu zupčanika, tako da generira vanjsku i unutrašnju kružnicu pomoću `MeshBuilder.GenerateCircle` te ih spoji koristeći `MeshBuilder.MakeCircleBridge`.

Metoda `GenerateTeeth` dodaje dodatne *vertexe* za zupce na obje strane zupčanika pomoću metode `GenerateTeethVertices`, te ih spaja koristeći `MeshBuilder.MakeCircleBridge`.

Metoda `GenerateTeethVertices` prolazi kroz sve *vertexe* vanjske kružnice zupčanika te za svaki drugi par *vertexa* poziva funkciju `ToothMeshBuilder.MakeTooth` za generiranje zupca.

4.2. Generiranje modela letvice

Klasa za proceduralno generiranje letvice je skoro identična klasi za generiranje zupčanika, osim što umjesto `MeshBuilder.GenerateCircle` koristi `GenerateLine`. UML dijagram klase `RackMeshGenerator` prikazuje Slika 6.

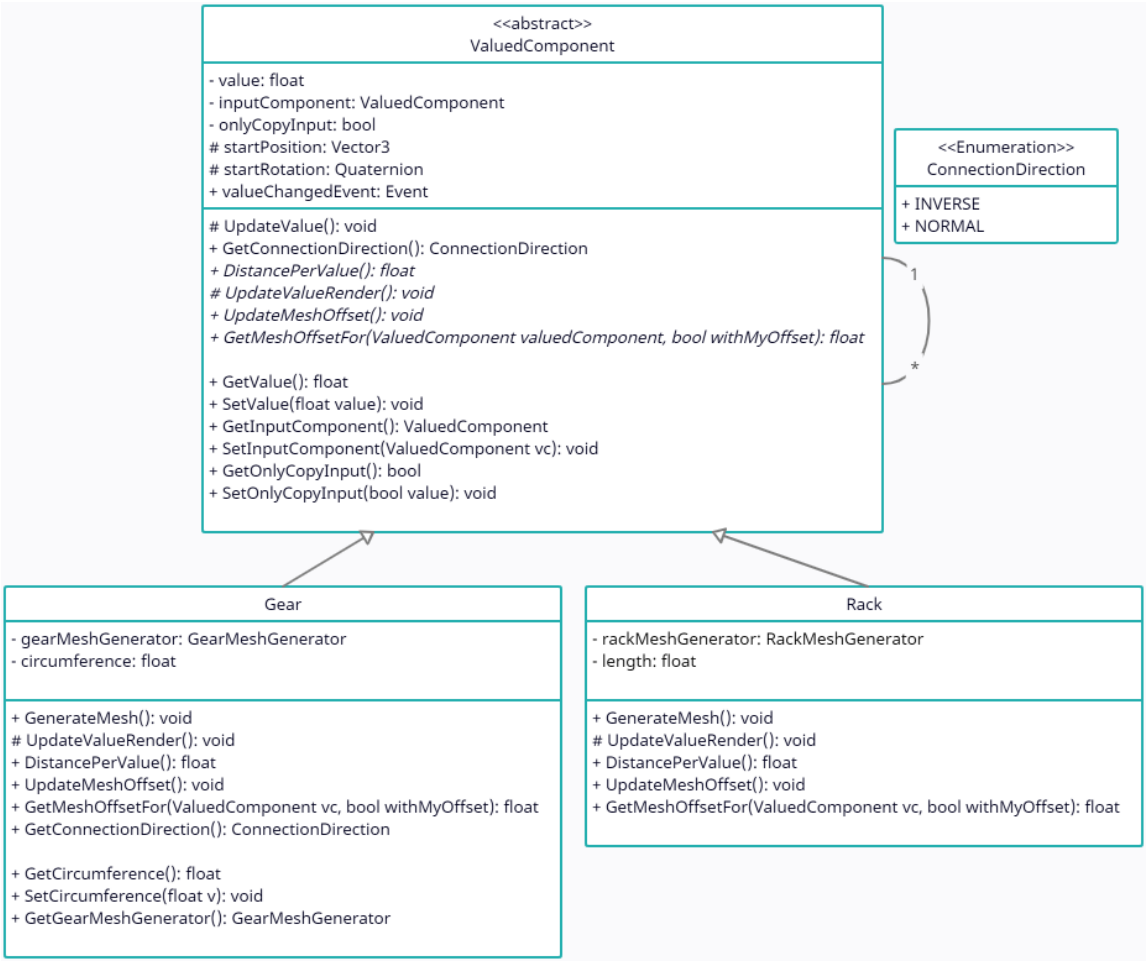


Slika 6 UML dijagram klase RackMeshGenerator

5. Simuliranje mehaničkih elemenata

Svaki mehanički element (zupčanik ili letvica) je objekt u sceni, te se njegovo ponašanje simulira baznom klasom `ValuedComponent` iz koje su derivirane klase `Gear` i `Rack`. UML dijagram tih klasa prikazuje Slika 7. `ValuedComponent` klasa sadrži atribute:

- `value` – vrijednost ovog elementa (broj okretaja zupčanika ili pomak letvice)
- `inputComponent` – element na kojeg je spojen trenutni element
- `onlyCopyInput` – ako je element spojen na `inputComponent` čvrstom vezom (umjesto zupcima), onda se vrijednost `inputComponente` ne skalira nego samo direktno kopira
- `startPosition` i `startRotation` – početna pozicija i rotacija elementa u 3D prostoru
- `valueChangedEvent` – događaj koji se odašilje pri promjeni `valuea`



Slika 7 UML dijagram klasa ValuedComponent, Gear i Rack

Metoda `UpdateValue` se pozove svaki put kad `inputComponent` pozove događaj `valueChangedEvent`, odnosno kad se promjeni vrijednost elementa na kojeg je trenutni element spojen. U toj metodi je potrebno postaviti novu vrijednost atributa `value`. Vrijednost se samo kopira u slučaju da je postavljen `onlyCopyInput`, a inače se računa po formuli (7).

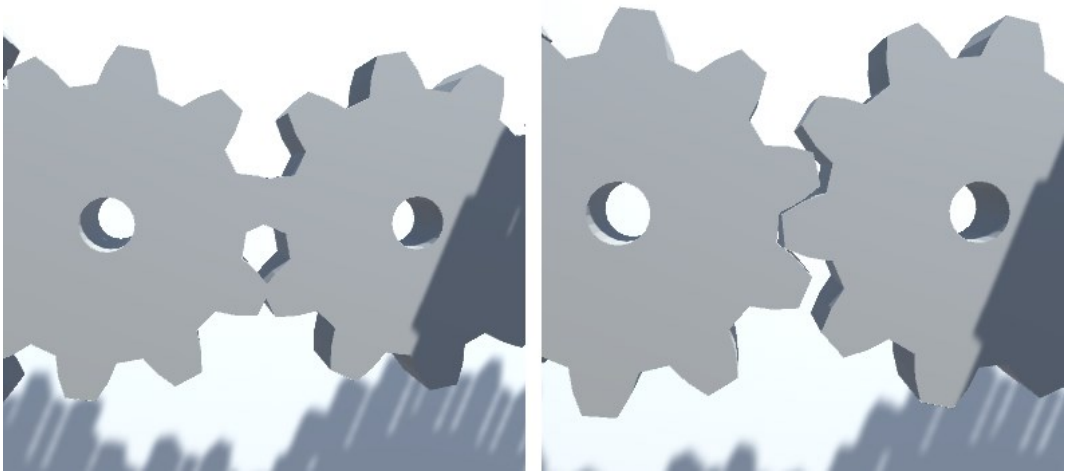
$$this.Value = inputComponent.Value * \frac{inputComponent.DistancePerValue()}{this.DistancePerValue()} \quad (7)$$

Gdje su `DistancePerValue` metode koje vraćaju prijeđeni put zubaca te komponente po promjeni njene vrijednosti za 1. Točnije za zupčanik je to njegov opseg, a za letvicu je 1. Također je potrebno vrijednost pomnožiti s -1 ako je `GetConnectionDirection` jednak `INVERSE`. To se koristi u slučaju da su zupčanici povezani remenom, te se onda okreću u istom smjeru umjesto u suprotnom.

Metodu `UpdateValueRender` je potrebno zvati iz metode `SetValue`. Ona postavlja izgled elementa ovisno o njenoj trenutnoj vrijednosti (rotacija zupčanika ili pomak letvice).

5.1. Problem poklapanja zubaca u prikazu

Ako se ne doda nikakva funkcionalnost osim gore navedene, dogodit će se da zupci dva elementa koji su spojeni prolaze jedni kroz druge, odnosno neće se točno poklopiti. Zato je potrebno odvojiti sami 3D model elementa u odvojeni objekt, te ga postaviti kao njegovo dijete. Tako dobijemo mogućnost da pomaknemo ili rotiramo 3D model elementa bez da utječemo na njegovu vrijednost i na taj način točno poklopimo zupce. Slika 8 prikazuje prije i poslije.



Slika 8 Prije i poslije implementacije poklapanja zubaca u prikazu elemenata

Metoda `UpdateMeshOffset` treba postaviti pomak/rotaciju djeteta tako da zupci budu točno poklopljeni. Metoda je apstraktna te se implementira u podrazredima Gear i Rack. U klasi Gear metoda zbraja *offset* vrijednosti dobivene pozivom metode `GetMeshOffsetFor` za sebe u odnosu na `inputComponentu`, te za `inputComponentu` u odnosu na sebe, ali u prvom slučaju za argument `onlyCopyInput` šalje `true`. Zatim djetetu koje sadrži 3D model postavlja rotaciju na dobiveni zbroj pomnožen s kutom koji prekriva jedan zubac.

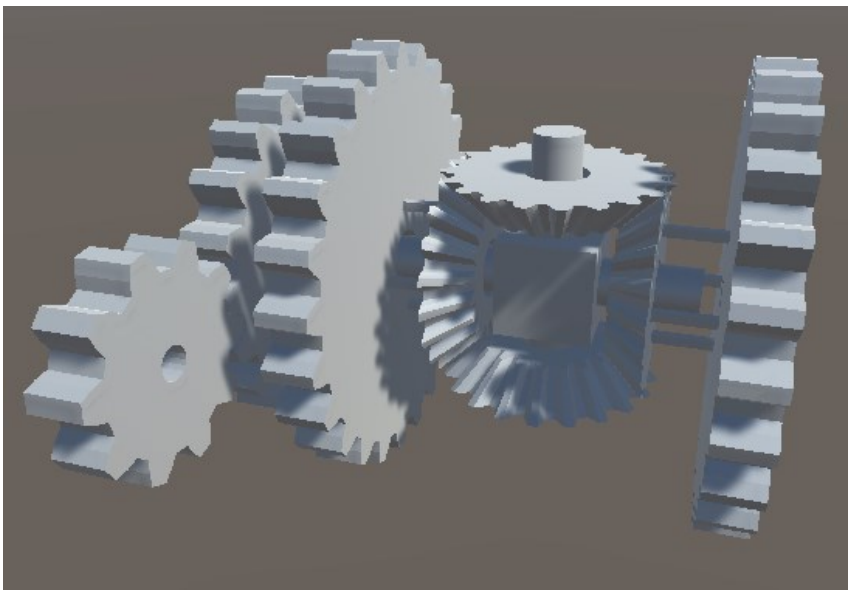
Metoda `GetMeshOffsetFor` vraća vrijednost između -1 i 1 koja označava koliko zubaca je 3D model trenutnog elementa krivo poklopljen u odnosu na 3D model danog elementa. U slučaju da je argument `withMyOffset` postavljen na `true` onda se konačnom rezultatu pridodaje pomak/rotacija i trenutnog 3D modela, inače se taj već primijenjen pomak/rotacija zanemaruju.

6. Virtualne mehaničke komponente

Pomoću implementiranih zupčanika i letvica moguće je sastaviti i same komponente koje će obavljati matematičke operacije. Svaku komponentu najbolje je kreirati kao šablonu tako da ih je jednostavno kreirati pri generiranju cijelog mehanizma.

6.1. Virtualni diferencijal

Virtualni diferencijal sastavljen je od zupčanika kako je pojašnjeno u prvom poglavlju. Na *end* i *spider* zupčanicima je potrebno postaviti `gearType` u klasi `GearMeshGenerator` na `BevelLeft` i `BevelRight`, te je cijeli *spider shaft*, zajedno sa *spider* zupčanicima, potrebno postaviti kao djecu *output* zupčanika, tako da se oni okreću zajedno s njim. Potrebno je kreirati i dodatni *output* zupčanik koji će biti duplo manji od početnog *output* zupčanika tako da mu pomnoži vrijednost s dva. Slika 9 prikazuje konačni izgled diferencijala.

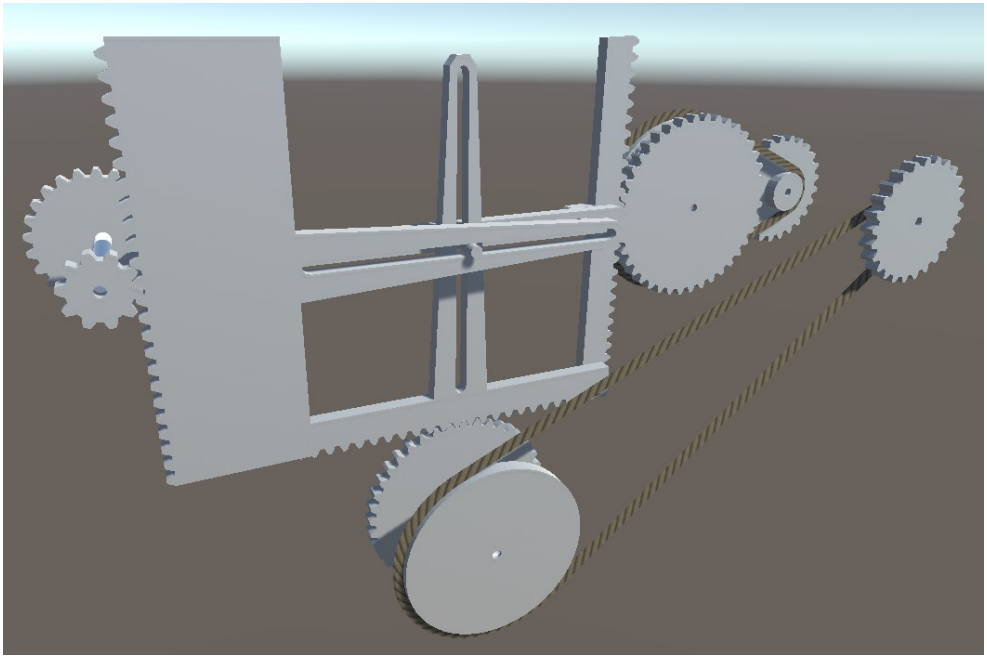


Slika 9 Virtualni diferencijal

Klasa `Differential` upravlja diferencijalom, tako da se pretplati na `valueChanged` događaj oba *end* zupčanika, te postavi vrijednost *output* zupčanika na njihov zbroj podijeljen s 2. Također promjeni i vrijednost *spider* zupčanika.

6.2. Virtualni množitelj

Virtualni množitelj je također sastavljen kako je opisano u prvom poglavlju, koristeći zupčanike i letvice. Ulazni zupčanici množitelja postavljeni su na istu visinu, te im je pristup s desne strane, tako je olakšan problem slaganja cijelog mehanizma koji je opisan u kasnijim poglavljima. Izlazni zupčanik također je postavljen na istu visinu te mu je pristup s lijeve strane. Također su dodani zupčanici prije ulaznih letvica koji skaliraju ulaz tako da je maksimalna vrijednost koju je moguće pomnožiti jednaka 1 a minimalna -1, prije nego li letvica dođe do kraja. Slika 10 prikazuje konačni izgled množitelja.



Slika 10 Virtualni množitelj

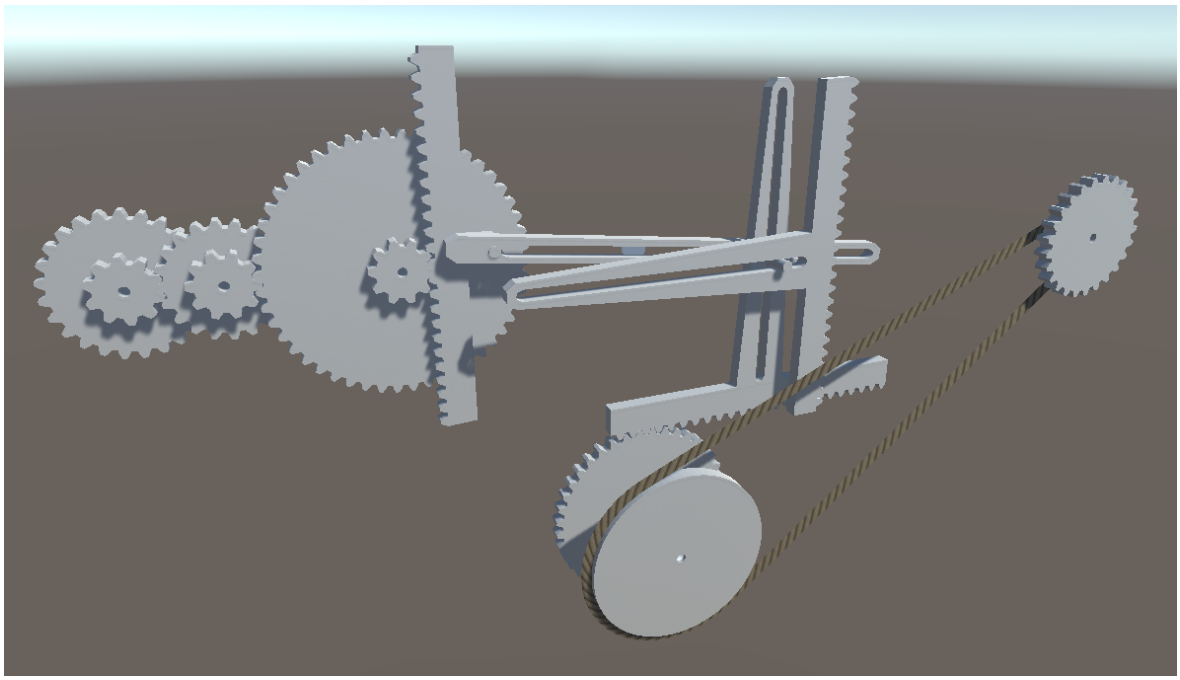
Klasa `Multiplier` upravlja množiteljem. Na početku, pri stvaranju množitelja izračunava se konstanta K odnosno udaljenost *stationary pina* i stožera *pivot arma*. Zatim se pri svakoj promjeni ulaznih letvica računa vrijednost izlazne letvice po formuli (1). Zatim se još treba izračunati rotacija zakretne ruke, te njen pomak u odnosu na stacionarni *pin*, te se još postavlja pozicija *multiplier pina*.

Ne postoji provjera jesu li letvice došle do svog kraja, odnosno je li *multiplier pin* još uvijek unutar svojih utora, tako da kada apsolutne vrijednosti ulaznih letvica prijeđu vrijednost 1 množitelj nastavlja sa svojim radom normalno, osim što vizualno izađe iz fizičkih granica mogućnosti, pa se to ne bi trebalo dozvoliti. Taj problem je riješen u kasnijim poglavljima.

6.3. Virtualni djelitelj

Djelitelj je sastavljen od dva dijela, prvo je komponenta za izračun recipročne vrijednosti nazivnika, a zatim je ta vrijednost pomnožena s brojnikom koristeći množitelj.

Djelitelj je konstruiran na isti način kao i množitelj osim što su mu zamijenjene izlazna letvica i jedna ulazna letvica. Također, sadrži samo jedan ulaz jer se računa samo recipročna vrijednost. Ulaz je još uvijek s desne strane, a izlaz s lijeve. Za letvicu na koju nije spojen ulaz (brojnik u operaciji dijeljenja) je postavljena vrijednost 1. Još je potrebno i dodati niz zupčanika koji će skalirati vrijednost izlazne letvice tako da se dobije točna vrijednost na izlazu. Slika 11 prikazuje konačan izgled djelitelja.



Slika 11 Virtualni djelitelj

Klasa `Divider` koja upravlja djeliteljem skoro je identična klasi `Multiplier`.

Osim gornjeg apsolutnog ograničenja, kao množitelj, djelitelj ima i donje apsolutno ograničenje od 0.1, no svejedno može računati recipročne vrijednosti negativnih brojeva samo što ulazna vrijednost ne smije proći kroz zabranjeno područje u rasponu <-0.1, 0.1> nego smije biti samo pozitivna ili samo negativna. Rješenje tog problema opisano je u kasnijem poglavlju.

6.4. Virtualni množitelj s konstantom

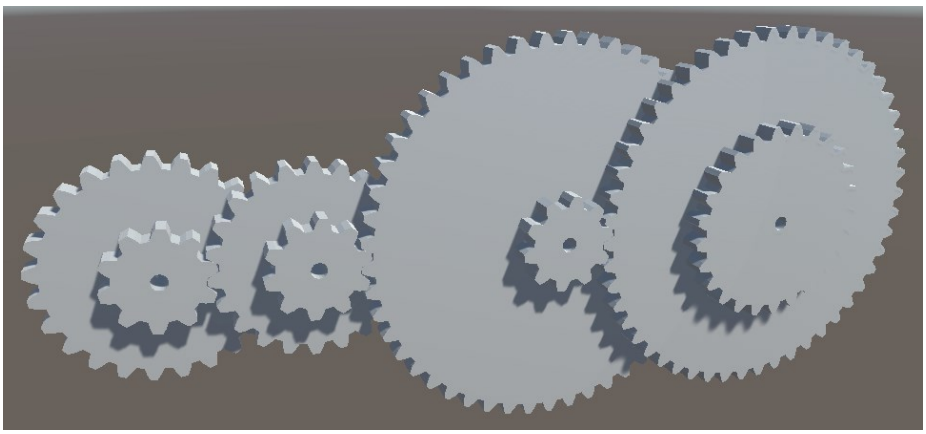
Množenje s konstantom moguće je izvesti bez korištenja letvica, pa tako i bez ulaznih ograničenja. Ideja je da se izgenerira niz zupčanika koji imaju konačni omjer zubaca kao i tražena konstanta.

Klasa `ConstantMultiplier` sadrži jedinu metodu `GenerateGears` koja kao argument prima konstantu za koju treba generirati zupčanike. Algoritam za generiranje niza zupčanika opisan je sljedećim pseudokodom.

```
GenerateGears(constant):
      lastGear = KreirajNoviZupcanik();
      isNegativeCurrent = true;

      if (constant < 0):
            isNegative = true;
            constant *= -1;

      if (constant < 1):
            isReciprocal = true;
            constant = 1 / constant;

      while (constant > 1):
            if (constant > MAX_RATIO) ratio = MAX_RATIO;
            else ratio = constant;

            firstGearCircumference = ratio * MIN_CIRCUMFERENCE;
            secondGearCircumference = MIN_CIRCUMFERENCE;

            isWholeRatio = IsInt(firsGearCircumference);

            if (isReciprocal)
                  Swap(firstGearCircumference, secondGearCircumference);

            firstGear = KreirajNoviZupcanik(c = firstGearCircumference);
            firstGear.PlaceNextTo(lastGear);
            firstGear.InputComponent = lastGear;
            firstGear.OnlyCopyInput = true;

            secondGear = KreirajNoviZupcanik(c = secondGearCircumferenc);
            secondGear.PlaceOn(firstGear);
```

```
        secondGear.InputComponent = firstGear;

        if (!isWholeRatio):
              SpojiSRemenom(leftGear, rightGear);

        if (isWholeRatio):
              isNegativeCurrent = !isNegativeCurrent;

        lastGear = secondGear;
        constant /= ratio;
end while;

if (isNegative != isNegativeCurrent)
        DodajJosJedanZupcanikDaSeOkrenePredznak();
```

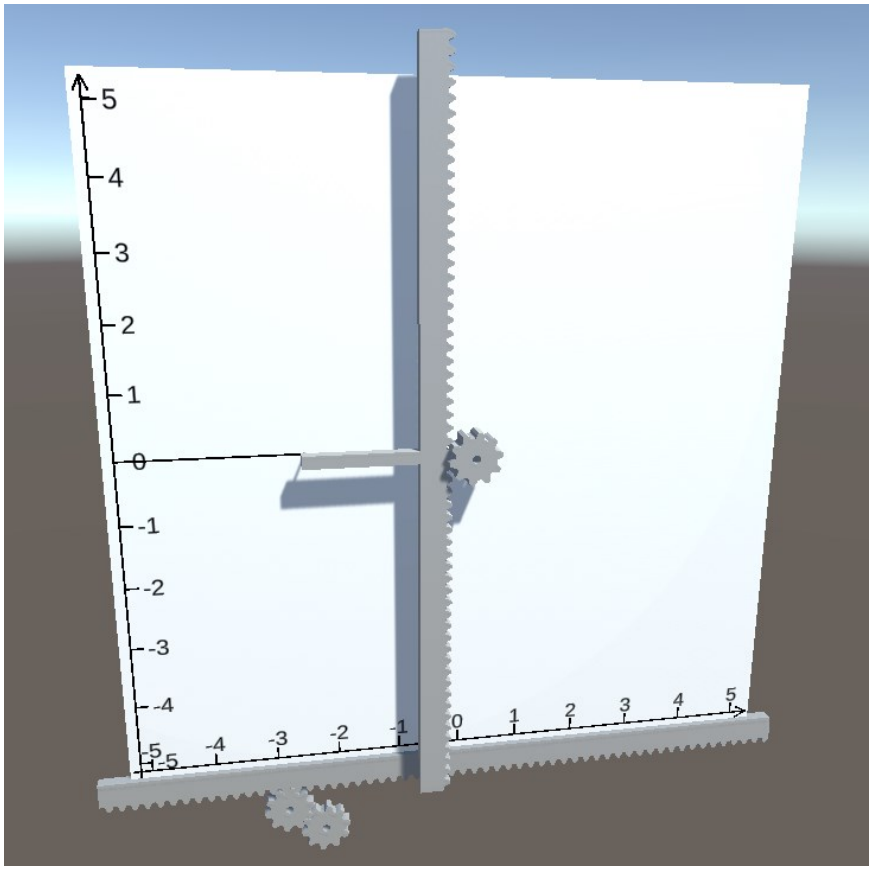
Konstanta `MAX_RATIO` predstavlja maksimalni omjer dva zupčanika koja su međusobno spojena, dok konstanta `MIN_CIRCUMFERENCE` predstavlja minimalni opseg najmanjeg zupčanika koji će se generirati. Metoda `PlaceNextTo` postavlja trenutni zupčanik pored danog zupčanika u 3D prostoru, a metoda `PlaceOn` ga postavlja tako da se njihovi zupčanici spajaju. Slika 12 prikazuje zupčanike koje je taj algoritam izgenerirao za konstantu 55 uz parametre `MAX_RATIO = 5` i `MIN_CIRCUMFERENCE = 10`.



Slika 12 Množitelj s konstantom za konstantu 55

6.5. Crtač

Crtač se sastoji od bijele površine koja predstavlja papir i dvije letvice, jedna koja pomiče papir u lijevo i predstavlja x os, te druge koja pomiče olovku gore i dolje te predstavlja y os. Na papiru se nalaze i označene koordinatne osi s ucrtanim vrijednostima, te su na letvice spojeni ulazni zupčanici. Slika 13 prikazuje konačni izgled crtača.



Slika 13 Crtač

Klasa `Plotter` upravlja crtačem, odnosno prije svakog iscrtavanja slike na ekran, dodaje trenutnu točku olovke u listu točci, te iscrtava liniju prolazeći kroz sve dosad zapisane točke u listi. Također sadrži i metodu `GenerateCoordinateSystem(float xFrom, float xTo, float yFrom, float yTo)` koja inicijalizira koordinatne osi.

6.6. Ulazna ručica

Klasa `Crank` upravlja ulaznim zupčanikom cijelog mehanizma te ga okreće odnosno povećava mu vrijednost prije iscrtavanja svake slike na ekran.

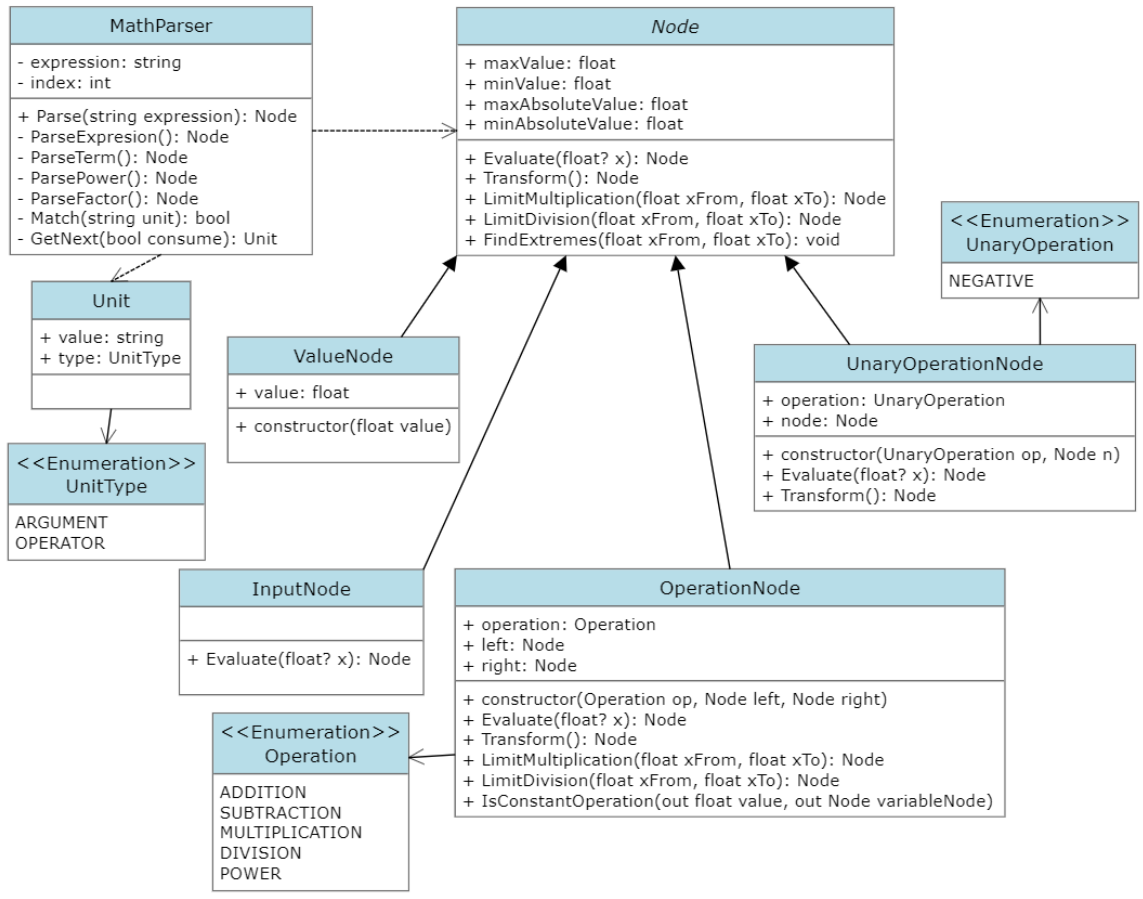
7. Parsiranje i transformacija matematičke funkcije

Prije generiranja konačnog mehanizma, potrebno je ulaznu matematičku funkciju parsirati u stablo matematičkih operacija, gdje je svaki čvor operacija, a djeca čvora su operandi.

7.1. Parsiranje metodom rekurzivnog spusta

Klasa `MathParser` sadrži implementaciju sintaksnog parsiranja metodom rekurzivnog spusta. Ulazna metoda je `Parse(string expression)` koja vraća korijenski čvor tipa `Node`. Slika 14 prikazuje njihov UML dijagram. Klasa `Node` je apstraktna klasa, a njezine podklase su:

- `ValueNode` – predstavlja konstantu
- `InputNode` – predstavlja ulaznu varijablu *x* u matematičkoj funkciji
- `OperationNode` – matematička operacija nad čvorovima `left` i `right`
- `UnaryOperationNode` – unarna operacija nad čvorom `node`



Slika 14 UML dijagram klasa MathParser i Node

Pri sintaksnom parsiranju potrebno je leksički odvajati ulazne znakove. Metoda `GetNext` vraća sljedeći `Unit` u ulaznom izrazu, to je moguće implementirati pomoću jednostavnog regularnog izraza. Ako je argument `consume` postavljen na *true* taj `Unit` se konzumira odnosno `index` se pomiče za dužinu učitanog `Unita`.

Metoda `Match` je pomoćna metoda koja preko argumenta prima očekivani *unit*, te koristeći metodu `GetNext` vraća *true* ako je sljedeći Unit očekivani *unit* a inače vraća *false*. Također konzumira *unit* jedino ako je jednak očekivanom.

Ulazna metoda `Parse` prvo sanira ulazni izraz brišući sve razmake, te zatim poziva metodu prve produkcije `ParseExpression`. Implementirane produkcije su opisane u poglavlju 2.

Metoda `ParseExpression` prvo poziva metodu `ParseTerm`, te zatim poziva `Match`("+" ili "-") dok on vraća *true*, te svaki put poziva `ParseTerm` i spaja ga sa zadnjim `Nodeom` preko `OperationNodea`.

Metoda `ParseTerm` izgleda skoro isto kao `ParseExpression` samo što spaja množenja i dijeljenja umjesto zbrajanja i oduzimanja, te poziva metodu `ParsePower`.

Pošto je produkcija (5) desna rekurzija, za razliku od ostalih produkcija, nju je moguće jednostavnije implementirati. Metoda `ParsePower` prvo poziva `ParseFactor` te zatim ako `Match("^")` vraća *true* rekurzivno poziva `ParsePower` te ta dva rezultata vraća spojena `OperationNodom` potenciranja.

Metoda `ParseFactor` prvo čita sljedeći *unit* metodom `GetNext`, te ovisno o dobivenoj vrijednosti primjenjuje drugu produkciju. Ako je vrijednost "-" onda poziva metodu `ParseTerm` te rezultat vraća omotan u `UnaryOperationNode`. Ako je vrijednost "(" onda poziva metodu `ParseExpression` te vraća dobiveni rezultat. Ako je tip pročitanog *unita* argument, umjesto operator, i vrijednost mu je "*x*" onda vraća `InputNode`, a inače vraća `ValueNode`.

7.2. Pojednostavljenje funkcije

Parsirani izraz poželjno je pojednostavniti, odnosno primijeniti matematičke operacije nad izrazima koji ne sadrže ulaznu varijablu *x*. Apstraktna klasa `Node` sadrži metodu `Evaluate` koja ima mogućnost primanja vrijednosti ulazne varijable *x*.

Podklase `OperationNode` i `UnaryOperationNode` implementiraju tu metodu tako da prvo rekurzivno pozivaju `Evaluate` svojim operandima, te zatim ako su operandi tipa `ValueNode` izračunaju vrijednost operacije te je vrate kreirajući novi `ValueNode`.

Podklasa `InputNode` implementira metodu `Evaluate` na način da vraća `ValueNode` ako je dobivena vrijednost argumenta `x`, a inače vrati `InputNode`.

7.3. Transformacija funkcije

Hodanjem po sintaksom stablu moguće je sada raditi jednostavne transformacije stabla koja će kasnije, pri generiranju cijelog mehanizma, pojednostavniti implementaciju.

Apstraktna klasa `Node` sadrži metodu `Transform` koja transformira dobiveni izraz u željeni, te je njene podklase implementiraju.

Prva transformacija je pretvaranje oduzimanja u zbrajanje po formuli:

$$a - b = a + (-1 * b) \tag{8}$$

Na taj način više nije potrebno implementirati mehaničku komponentu za oduzimanje nego samo za množenje s konstantom i zbrajanje.

Druga transformacija je pretvaranje unarne operacije negacije u množenje s minus jedan, te tako opet iskorištavamo mehaničku komponentu za množenje s konstantom.

Sljedeća transformacija je pretvaranje potencije u niz množenja. Na taj način umjesto da implementiramo mehaničku komponentu za potenciranje, koristimo komponentnu za množenje. Pri ovoj transformaciji također provjeravamo da potencija može biti samo pozitivan cjelobrojni broj.

Zadnja transformacije je pretvaranje dijeljenja u množenje s recipročnim nazivnikom po formuli:

$$\frac{a}{b} = a * \frac{1}{b} \tag{9}$$

Tako primjenjujemo djelitelj, opisani u poglavlju 6.3, koji računa recipročnu vrijednost.

7.4. Limitiranje operanada množenja i dijeljenja

Kao što je spomenuto u poglavljima 1.2, 6.2 i 6.3, množitelj i djelitelj imaju fizička ograničenja kolika je apsolutna maksimalna i minimalna vrijednost koju oni mogu dobiti za računanje. Množitelj ima samo gornju granicu, odnosno nema donju jer može množiti s nulom, te je pri implementaciji množitelja u poglavlju 6.2 ta granica podešena da bude 1. Odnosno množitelj radi u rasponu $[-1, 1]$.

Djelitelj ima istu gornju granicu kao i množitelj, ali ima i donju apsolutnu granicu koja je pri implementaciji u poglavlju 6.3 podešena da bude 0.1. Tako da djelitelj radi u rasponu $[-1,\ -0.1] \cup [0.1, 1]$.

Maksimalna vrijednost operanda limitira se tako da se pronađe maksimalna vrijednost koju će taj operand zaprimiti tijekom računanja funkcije u danom rasponu, te se taj operand podijeli tom vrijednošću, a rezultat množenja pomnoži istom tom vrijednošću. Važno je primijetiti da množenje s konstantom nema fizička ograničenja. To je potrebno ponoviti za oba operanda, te je transformacija izraza prikazana sljedećom formulom:

$$a * b = c \Rightarrow a * \frac{1}{max_a} * b * \frac{1}{max_b} * max_a * max_b = c \tag{10}$$

Na sličan način se limitira i dijeljenje. U poglavlju 7.3 smo sva dijeljenja razdvojili u množenje i računanje recipročne vrijednosti, tako da je sada potrebno samo limitirati nazivnik. Transformacija za limitiranje nazivnika prikazana je sljedećom formulom:

$$\frac{1}{b} = c \Rightarrow \frac{1}{\frac{b}{max_b}} * \frac{1}{max_b} = c \tag{11}$$

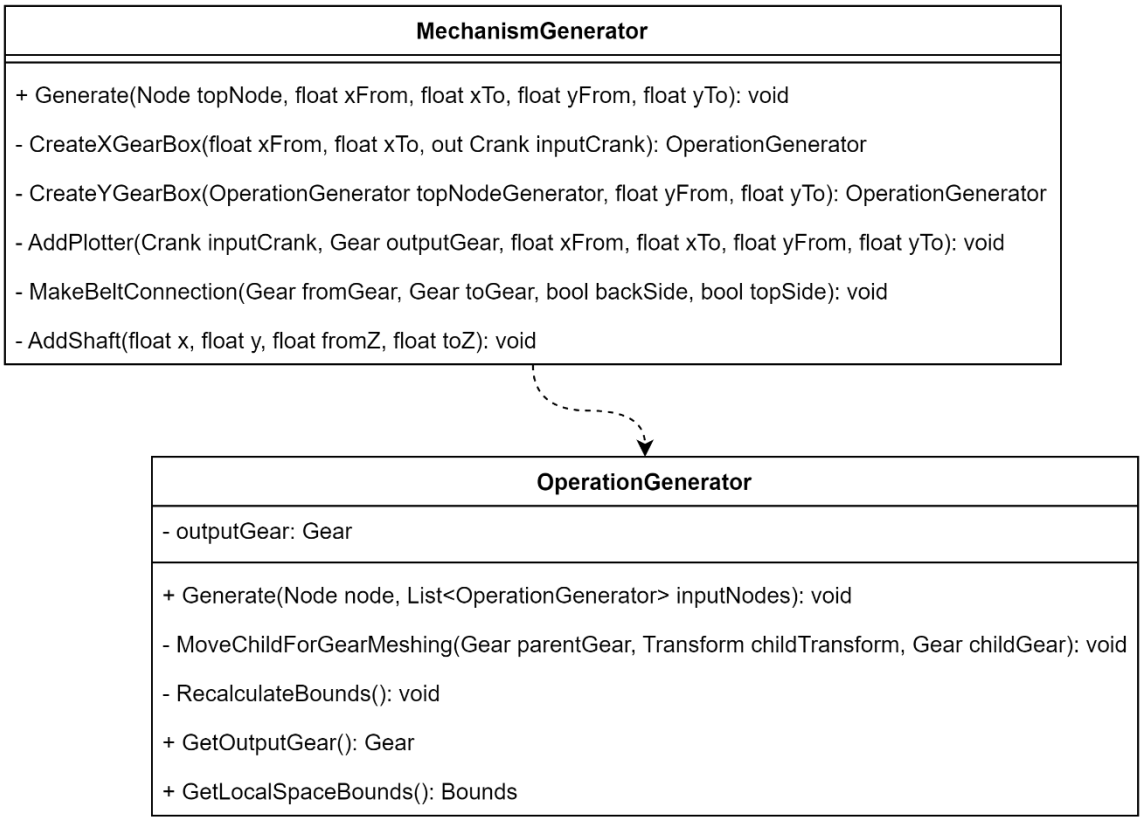
Za pronalazak ekstrema izraza na određenom rasponu koristi se metoda `FindExtremes` u klasi `Node`, koja poziva metodu `Evaluate` da bi dobila vrijednost izraza za određeni ulazni parametar *x*, te pronađene ekstreme sprema u atribute `maxValue`, `minValue`, `minAbsoluteValue` i `maxAbsoluteValue`.

Metoda `LimitMultiplication` rekurzivno prolazi kroz sintaksno stablo, te u čvorovima tipa `OperationNode` kojima je operacija množenje prvo poziva `FindExtremes` za oba operanda, te zatim provodi transformaciju kao u (10), gdje je max_i `maxAbsoluteValue`.

Metoda `LimitDivision` radi na isti način kao i `LimitMultiplication` osim što radi dodatne provjere. Prva provjera je ako je `minValue` manji od 0, a `maxValue` veći od 0, onda vrijednost nazivnika prolazi kroz 0 što nije izvedivo te se prikazuje odgovarajuća poruka korisniku. Zatim provjerava ako je `minAbsoluteValue` / `maxAbsoluteValue` manji od 0.1 onda također prikazuje poruku s greškom korisniku. Metoda na kraju vraća novi `OperationNode` s transformacijom kao u formuli (11).

8. Generiranje konačnog mehanizma

Nakon parsiranja i transformacije dobivenog izraza, sintaksno stablo se predaje klasi `MechanismGenerator` koja uz pomoć klase `OperationGenerator` kreira konačni mehanizam. Slika 15 prikazuje njihov UML dijagram.



Slika 15 UML dijagram klasa MechanismGenerator i OperationGenerator

Klasa `OperationGenerator` kreira mehanizam jednog čvora. Metoda `Generate` je ulazna metoda kojoj se predaje `Node` i lista u kojoj su zapisani svi `OperationGeneratori` koji predstavljaju `InputNode`. Sljedeći pseudokod opisuje rad `Generate` metode:

```
if (node is InputNode):
      outputGear = KreirajZupcanik();
      inputNodes.Add(this);
else if (node is ValueNode):
      outputGear = KreirajZupcanik();
      outputGear.Value = node.Value;
else if (node is OperationNode):
      if (node.IsConstantMultiplication()):
```

```
            opGen = KreirajOperationGeneratorZa(node.left);
            outputGear = KreirajConstantMultiplier(node.right, opGen);
      else if (node.Operation == DIVISION):
            opGen = KreirajOperationGeneratorZa(node.right);
            outputGear = KreirajDivider(opGen);
      else:
            leftOp = KreirajOperationGeneratorZa(node.left);
            rightOp = KreirajOperationGeneratorZa(node.right);

            if (node.Operation == ADDITION):
                  outputGear = KreirajDifferential(leftOp, rightOp);
            else if (node.Operation == MULTIPLICATION):
                  outputGear = KreirajMultiplier(leftOp, rightOp);

            if (leftOp.bounds.Intersects(rightOp.bounds)):
                  Pomakni_leftOp_u_smjeru_Z_osi();
                  Dodaj_dodatni_zupcanik_i_shaft();

RecalculateBounds();
```

Metoda `Generate` rekurzivno za svaki čvor sintaksnog stabla kreira još jedan `OperationGenerator`, te ovisno o operaciji čvora kreira komponentu koja predstavlja tu operaciju, i na kraju provjerava je li se dobivena djeca `OperationGeneratori` sječu u 3D virtualnom prostoru, te u tom slučaju pomiče lijevo dijete od desnog, te dodaje dodatan zupčanik na kojeg ga spaja, a taj zupčanik spaja s originalnim ulaznim zupčanikom pomoću osovine. Na kraju, nakon što su sva djeca čvorovi izgenerirani, poziva se metoda `RecalculateBounds`.

Metoda `RecalculateBounds` rekurzivno prolazi kroz stablo scene, te računa obujmicu (engl. *bounds*) koja predstavlja kvadar koji u potpunosti opisuje cijeli sadržaj tog objekta i njegove djece.

Metoda `MoveChildForGearMeshing` pomiče objekt dijete tako da se izlazni zupčanik tog djeteta spaja sa ulaznim zupčanikom trenutnog `OperationGeneratora`.

Klasa `MechanismGenerator` kreira cjelokupni završni mehanizam koji se sastoji od ulaznog *gearboxa*, mehanizma za računanje funkcije, izlaznog *gearboxa* i crtača. Ulazni i izlazni *gearboxovi* služe za translaciju i skaliranje vrijednosti na raspon koordinatnih osi koje je korisnik upisao.

Metoda `AddPlotter` kreira crtač u sceni te mu jedan ulaz spaja pomoću remena na ulaznu ručicu, a drugi ulaz mu spaja na izlazni zupčanik izlaznog *gearboxa*.

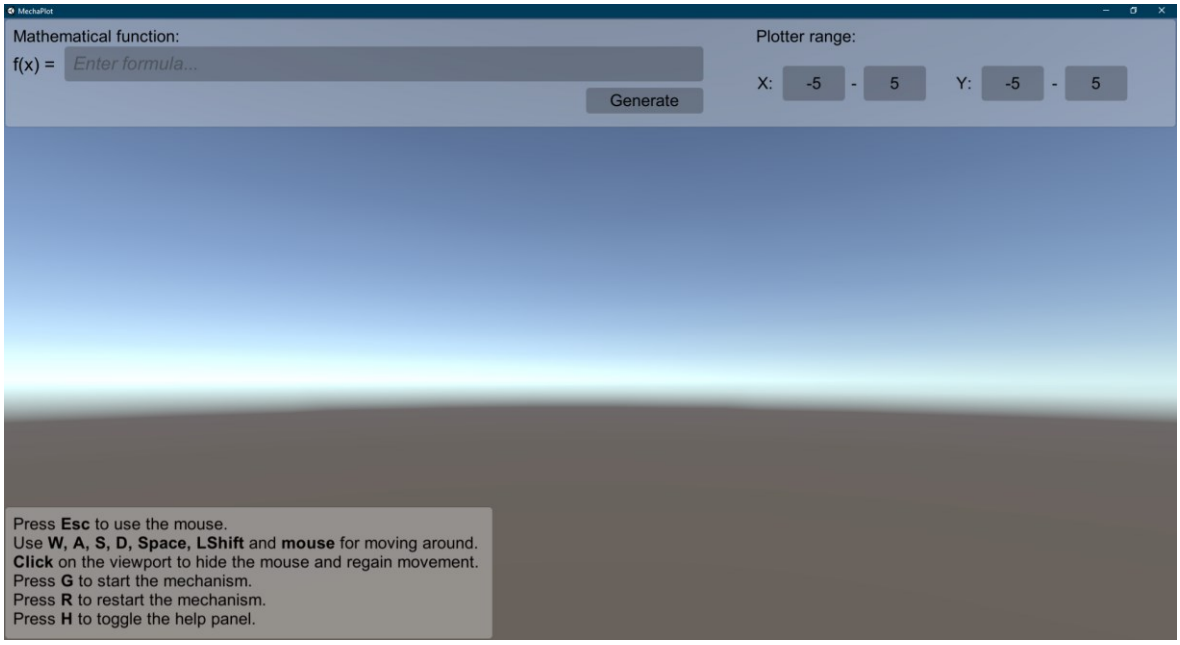
Metoda `MakeBeltConnection` je pomoćna metoda koja kreira dodatan zupčanik preko kojeg spoji `fromGear` i `toGear` pomoću remena. Argumenti `backSide` i `topSide` određuju s koje strane `fromGeara` će se kreirati dodatni zupčanik.

Metoda `AddShaft` je također pomoćna metoda koja dodaje u scenu cilindar koji predstavlja osovinu.

Metoda `Generate` koristi prije opisane metode za generiranje konačnog mehanizma. Prvo kreira korijenski `OperationGenerator` kojem predaje korijenski čvor sintaksnog stabla. Zatim poziva metodu `CreateYGearBox`. Nakon toga prolazi kroz sve `OperationGeneratore` koji su u `inputNodes` listi, te ih spaja na nove zupčanike preko remena, a te zupčanike spoji osovinom. Zatim poziva metodu `CreateXGearBox`, te na kraju metodu `AddPlotter`.

9. Rezultati i diskusija

Pokretanjem konačnog programa dobiva se početni prozor kojeg prikazuje Slika 16. Na vrhu su ulazne kontrole, s lijeve strane za upis matematičke funkcije, a s desne za raspon koordinatnog sustava na kojem će se iscrtati funkcija. U donjem lijevom kutu nalaze se uputstva za korištenje.



Slika 16 Početni prozor programa

Prvi primjer

Za prvi primjer rada programa korištena je sljedeća funkcija:

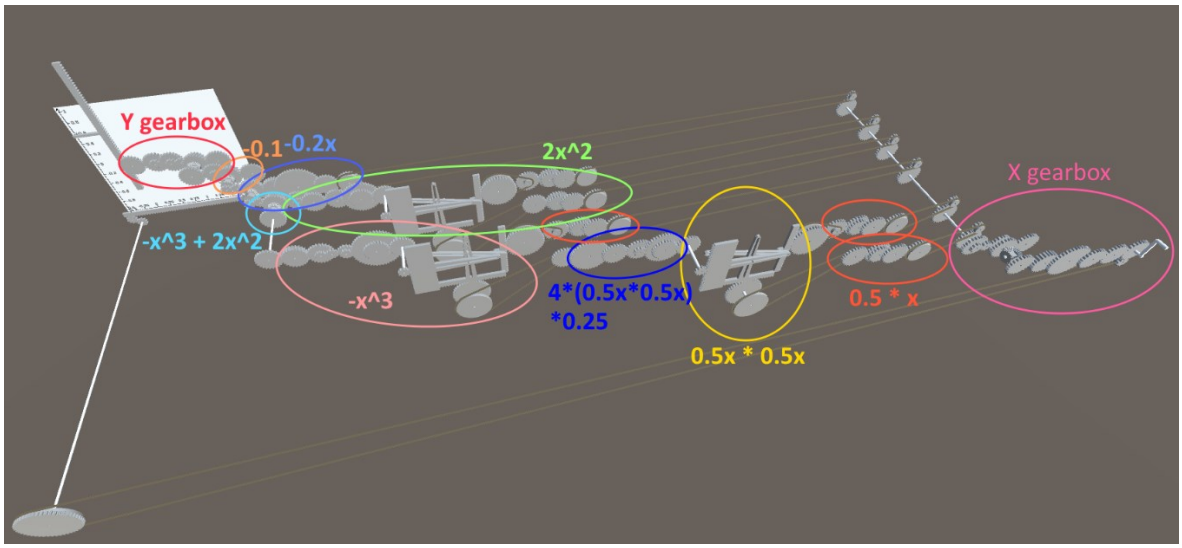
$$f(x) = -x^3 + 2x^2 - 0.2x - 0.1 \tag{12}$$

Odnosno kad je napisana bez korištenja *superscripta* glasi: -x^3 + 2x^2 - 0.2x - 0.1.

Za raspon *x* osi odabran je [-0.5, 2], a za *y* raspon [-1, 1]. Dobiveni konačni mehanizam prikazuje Slika 17.

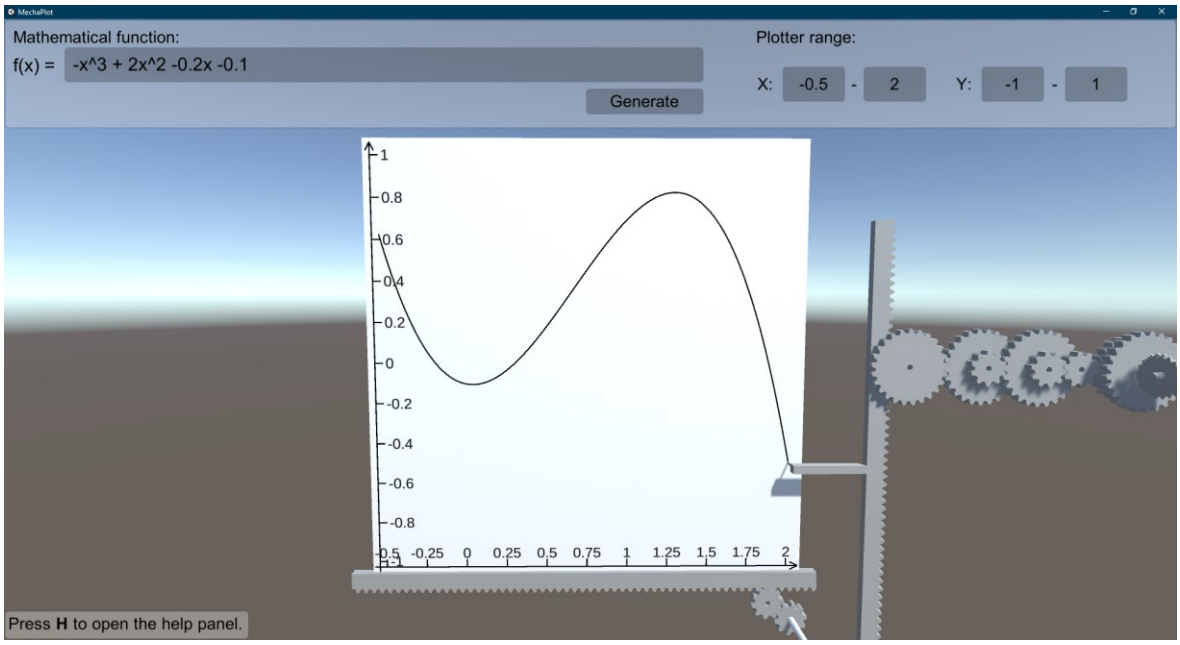
Dobiveni izraz program transformira i limitira operande kako je opisano u poglavlju 7, te se mehanizam generira po sljedećem izrazu: ((((-1 * (8 * (((4 * ((x * 0.5) * (x * 0.5))) * 0.25) * (x * 0.5)))) + (2 * (4 * ((x * 0.5) * (x * 0.5))))) + (-1 * (0.2 * x))) + -0.1).

Slika 17 također sadrži oznake koje opisuju koji dijelovi mehanizma računaju koji dio izraza.



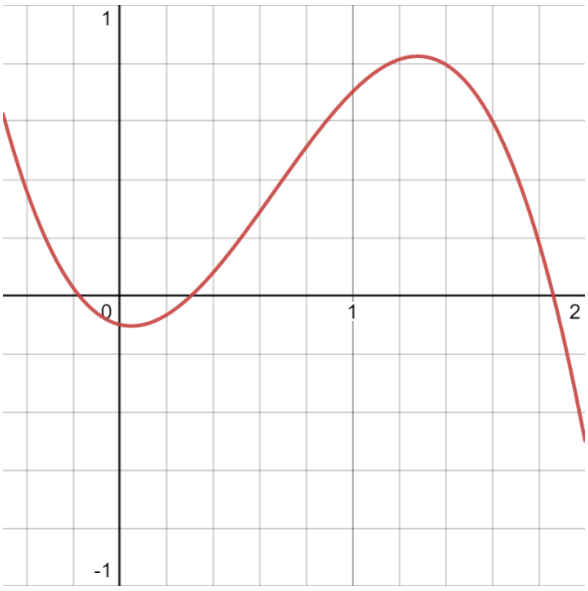
Slika 17 Konačni mehanizam prvog primjera

Nakon pokretanja ulazne ručice, crtač iscrtava funkciju koju prikazuje Slika 18.



Slika 18 Izlaz crtača prvog primjera

Slika 19 prikazuje graf iste te funkcije nacrtanog pomoću alata desmos.com. Usporedbom s izlazom programa vidi se da program točno iscrtava traženu funkciju.



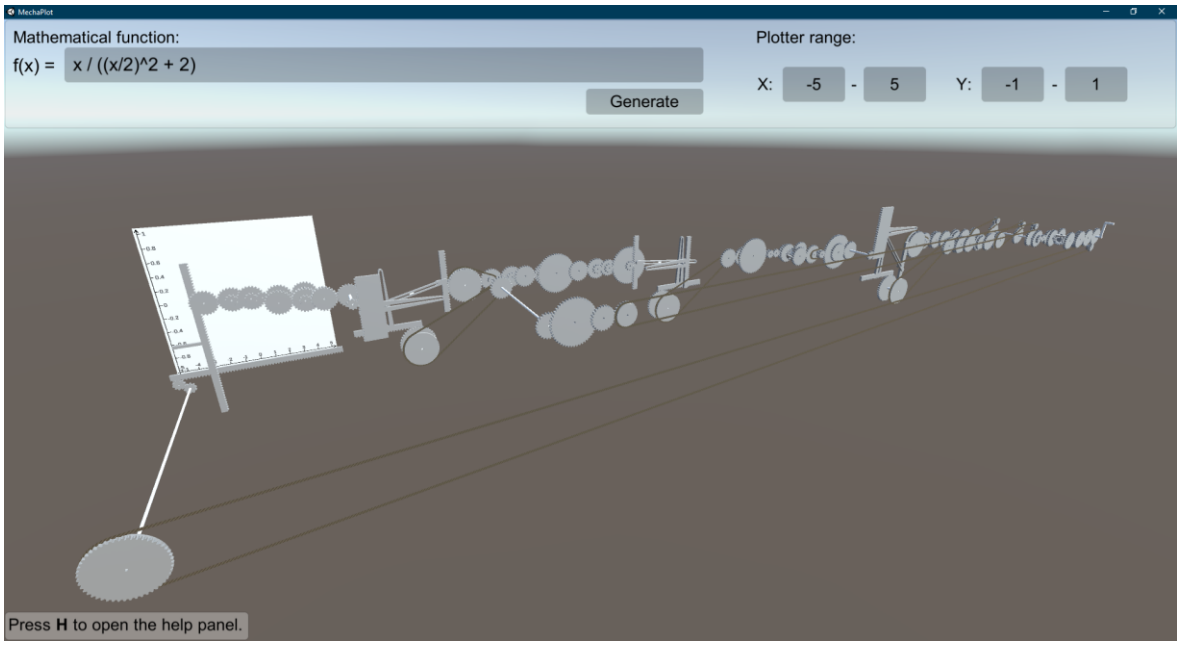
Slika 19 Graf prvog primjera iscrtan pomoću alata desmos.com

Drugi primjer

Za drugi primjer koristi se sljedeća funkcija:

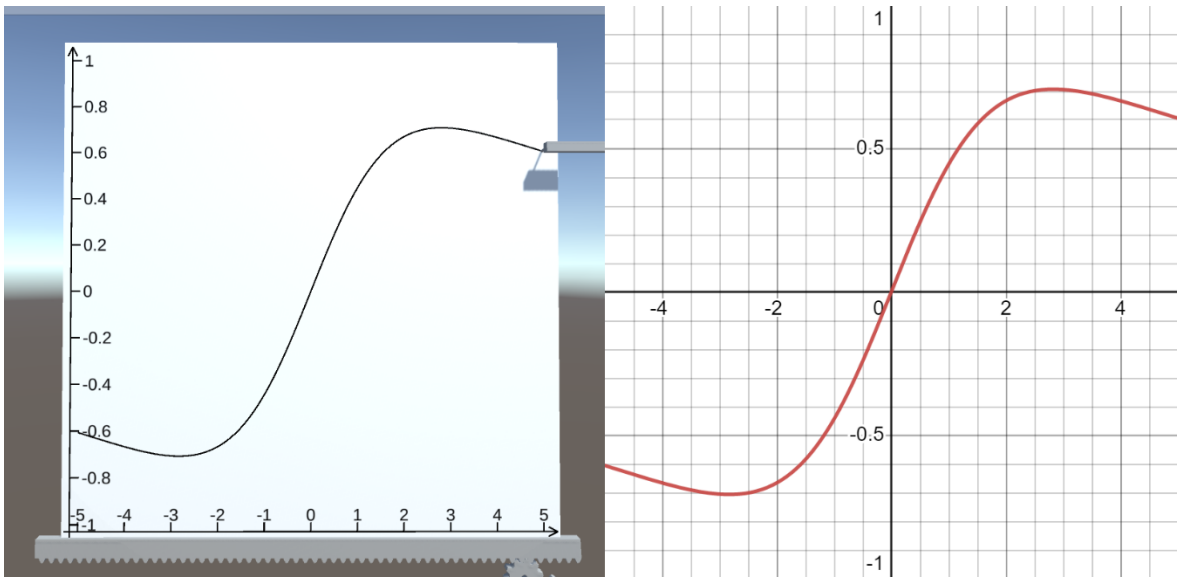
$$f(x) = \frac{x}{\frac{x^2}{2}+2} \tag{13}$$

S rasponom na *x* osi [-5, 5], a na *y* osi [-1, 1]. Slika 20 prikazuje konačni mehanizam koji je program izgenerirao za funkciju (13).



Slika 20 Konačni mehanizam drugog primjera

Slika 21 prikazuje izlaz programa kraj grafa iste funkcije iscrtanog alatom desmos.com na kojoj je vidljivo da je iscrtana krivulja ispravna.



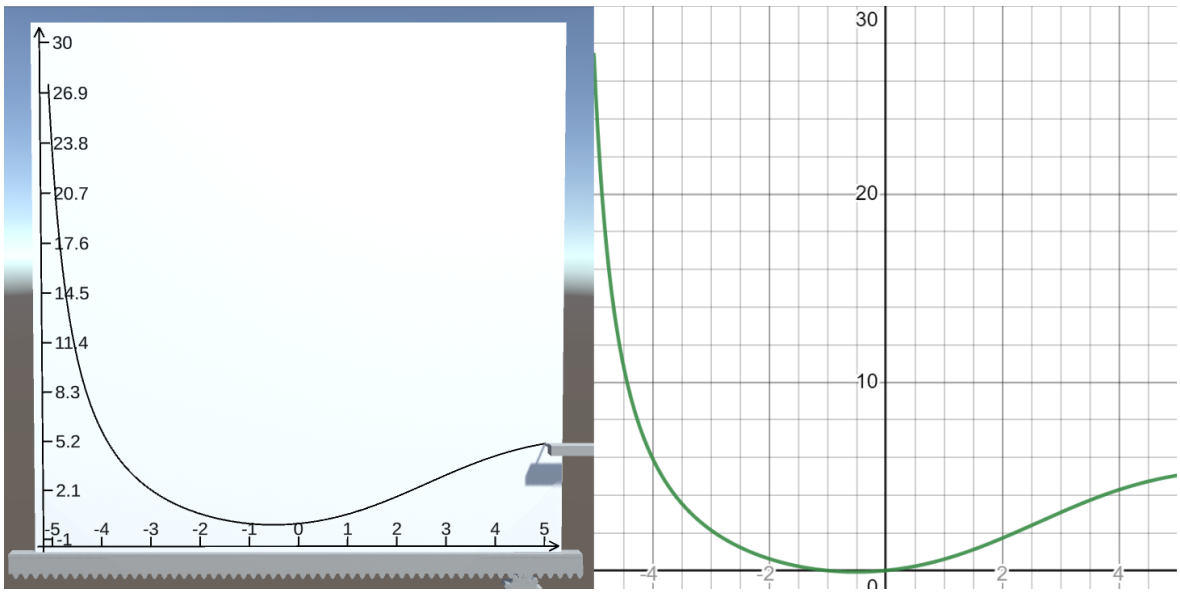
Slika 21 Usporedba izlaza programa i grafa funkcije iscrtanog alatom desmos.com drugog primjera

Treći primjer

Za treći primjer korištena je sljedeća funkcija:

$$f(x) = \frac{6x^2+6x}{\frac{x^3}{2}+20} \tag{14}$$

S rasponom na *x* osi [-5, 5], a na *y* osi [-1, 30]. Slika 22 prikazuje dobiveni izlaz.



Slika 22 Usporedba izlaza programa i grafa funkcije iscrtanog alatom desmos.com trećeg primjera

Zaključak

Program prikazuje mogućnost zupčanika da izvršavaju analogno računanje, i to skoro bilo koje racionalne funkcije. Skoro bilo koje jer postoje fizička ograničenja (duljina letvice) koja ograničavaju veličine brojeva s kojima mehanizam može računati. Također, mehanizam nije generiran da zauzima što je manje prostora moguće (kao što bi se radilo u stvarnom svijetu), nego samo prikazuje mogućnost generiranja takvog mehanizma bez prostornog ograničavanja. Osim toga, analogna računala su poznata po gomilanju grešaka, no pošto je ovo digitalna simulacija takvih grešaka nema, tako da većina izgeneriranih mehanizama ne bi precizno radila u stvarnom svijetu.

Cilj ovog rada nije bio simuliranje stvarnog mehanizma sa svim njegovim ograničenjima i nesavršenstvima, nego simuliranje teoretskog savršenog mehanizma koji prikazuje teoretske mogućnosti zupčanika.

Literatura

[1] Navy Department Bureau of Ordnance. *Basic Fire Control Mechanisms*. Washington, D.C.: rujan 1944.

[2] John Smith, *Syntax Analysis: Compiler Top Down & Bottom Up Parsing Types*, 14. travnja 2022., https://www.guru99.com/syntax-analysis-parsing-types.html, 11. svibnja 2022.

[3] Theodore Norvell, *Parsing Expressions by Recursive Descent*, 1999., https://www.engr.mun.ca/~theo/Misc/exp_parsing.htm, 11. svibnja 2022.

[4] Igor S. Pandžić, Tomislav Pejša, Krešimir Matković, Hrvoje Benko, Aleksandra Čereković, Maja Matijašević, *Virtualna okruženja: interaktivna 3D grafika i njene primjene*, 1. izdanje, Zagreb: Element, 2011.

[5] Unity Documentation, *The GameObject-Component Relationship*, 2015., https://docs.unity3d.com/510/Documentation/Manual/TheGameObject-ComponentRelationship.html, 31. svibnja 2022.

[6] Unity Documentation, *Mesh*, 2022., https://docs.unity3d.com/ScriptReference/Mesh.html, 31. svibnja 2022.

[7] Unity Documentation, *Order of execution for event functions*, 2022., https://docs.unity3d.com/Manual/ExecutionOrder.html, 31. svibnja 2022.

Sažetak

Naslov: Simulacija mehaničkog crtača matematičkih funkcija realiziranog zupčanicima

Sadržaj:

Za generiranje virtualnog mehanizma za računanje racionalnih matematičkih funkcija prvo je opisana implementacija proceduralnog generiranja 3D modela zupčanika i letvica. Zatim je opisan model za simulaciju tih elemenata tako da svaki element ima svoju vrijednost, te varijablu koja predstavlja pomak njegovih zubaca promjenom njegove vrijednosti za jedan, tako je realiziran točan omjer okretanja spojenih zupčanika. Sada je s tim virtualnim zupčanicima i letvicama moguće sastaviti mehaničke komponente koje predstavljaju matematičke operacije zbrajanja, množenja i računanja recipročne vrijednosti. Zatim je opisana implementacija sintaksnog parsera metodom rekurzivnog spusta za parsiranje ulazne matematičke funkcije, koji generira sintaksno stablo nad kojim se rade transformacije za pretvaranje oduzimanja u zbrajanje i sl. Također su limitirane maksimalne vrijednosti operanada množenja i računanja recipročne vrijednosti da bi se zadovoljila fizička ograničenja komponenti. Na kraju se po tom sintaksnom stablu generira konačan mehanizam na koji se spaja ulazna ručica i mehanički crtač.

Ključne riječi: simulacija, mehanički crtač, racionalne funkcije, zupčanici, sintaksno parsiranje, analogna računala, proceduralno generiranje

Summary

Title: Simulation of a mechanical plotter of mathematical functions realized with gears

Content:

For generating virtual mechanism for computing rational mathematical functions, firstly the implementation of procedural generation of 3D models of gears and racks is described. Then, a model for simulating these elements is described so that each element has its own value, and a variable which represents the change of position of its teeth when its value changes by one, in that way correct gear ratio is realized. Now with these virtual gears and racks it is possible to assemble mechanical components which represent mathematical operations of addition, multiplication and reciprocal value calculation. Then, the implementation of the syntax parser is described using recursive descent method for parsing the input mathematical function, which generates a syntax tree over which transformations are performed to convert subtractions to additions, etc. Also, the maximum values of operands of multiplication and reciprocal value calculation are limited so that physical constraints of the components are satisfied. Finally, using the syntax tree, the final mechanism is generated which is then connected to an input crank and a mechanical plotter.

Keywords: simulation, mechanical plotter, rational functions, gears, syntax parsing, analog computers, procedural generation