

Cuprins

| | | |
|-------|---|----|
| 1 | Cuvinte cheie | 3 |
| 2 | Introducere | 4 |
| 3 | Informații preliminare | 5 |
| 4 | Graf | 6 |
| 4.1 | Multigraf | 7 |
| 4.2 | Graf planar | 8 |
| 4.3 | Aplicare | 10 |
| 5 | Algoritmi force-directed de desenare a grafurilor | 12 |
| 5.1 | Aplicare | 13 |
| 5.2 | Avantaje | 14 |
| 5.3 | Dezavantaje | 15 |
| 5.4 | Metode de desenare | 15 |
| 5.4.1 | Metoda baricentrică | 15 |
| 5.4.2 | Algoritmul Fruchterman–Reingold | 16 |
| 6 | Proiecția ortografică | 18 |
| 6.1 | Aplicare | 19 |
| 7 | Tehnologii și limbaje folosite | 20 |
| 7.1 | C++ | 20 |
| 7.2 | Qt | 21 |
| 7.2.1 | Graphics view framework | 22 |
| 7.3 | Aplicare | 25 |
| 8 | Grafica turtle | 26 |
| 8.1 | Aplicare | 26 |
| 9 | Setul de date | 30 |
| 9.1 | XML | 30 |
| 9.2 | Aspecte | 30 |
| 9.3 | Terminologie specifică XML | 31 |
| 9.4 | Utilizabilitate XML | 32 |

| | |
|--|----|
| 10 Simcenter | 33 |
| 11 Funcționalități | 38 |
| 11.1 Încarcarea fișierelor | 39 |
| 11.2 Modificarea diagramei | 40 |
| 11.3 Salvarea scenei modificate | 41 |
| 11.4 Afișarea sub forma unei perspective ortografice | 42 |
| 11.5 Afișarea sub formă de graf | 43 |
| 11.6 Căutarea elementelor | 44 |
| 11.7 Compararea modelelor | 46 |
| 11.8 Informații adiționale | 47 |
| 11.9 Legenda | 48 |
| 12 Anexa | 50 |
| 12.1 XML | 50 |
| 12.1.1 Motion body | 50 |
| 12.1.2 Joint | 50 |
| 12.1.3 Connector | 51 |
| 12.2 Turtle graphics | 52 |
| 12.2.1 Desenarea unui cilindru | 52 |
| 13 Bibliografie | 54 |

1 Cuvinte cheie

- Motion body - corp rigid, componentă a modelului 3D
- Joint - legătură într-un punct fix dintre două elemente Motion body
- Connector - element flexibil care leagă două elemente Motion body, ex: un arc
- Mecanism - întregul model format din elemente Motion body, Joint și Connector
- .mdef - motion definition file, extensia fișierului care conține datele prelucrate de aplicație
- XML - extensible markup language
- C++ - limbaj de programare
- Qt - bibliotecă grafică pentru C++
- Force-directed - clasă de algoritmi pentru desenarea grafurilor

2 Introducere

În cartografie sau geologie, o hartă topologică este un tip de diagramă care conține doar informații esențiale. Numele este derivat din topologie, o ramură a matematicii care studiază proprietățile unui obiect care nu se schimbă dacă obiectul în sine este deformat, fie prin întindere, îndoire, dar nu și rupere sau lipire. Aceste hărți nu au scară, iar distanța și direcția sunt supuse schimbării și variației, având în prim plan evidențierea relațiilor dintre anumite puncte esențiale. Un exemplu ar fi harta unui metrou care conține doar date importante precum numele stațiilor și ordinea lor, însă distanța dintre ele nu este întocmai cea din realitate.

Acest concept este aplicat pentru crearea unei diagrame având la baza un model 3D format din mai multe elemente. Modelul 3D este generat de aplicația de simulare Simcenter sub forma unui fișier .mdef care conține doar informații esențiale. Aplicația prezentată în această lucrare are scopul de a vizualiza doar noțiuni semnificative precum relațiile dintre elemente și tipul legăturilor lor. Această aplicație este necesară în cazurile când nu se dorește deschiderea întregului model în Simcenter (lucru ce se dovedește ineficient în unele cazuri din punct de vedere al timpului de încărcare), ci doar crearea unei diagrame simple conținând caracteristicile prezentate anterior.

3 Informații preliminare

Topologia, ca și o ramură bine definită a matematicii, își are originile de la începutul secolului XX, însă au fost descoperite și cazuri izolate în aplicarea acestui concept chiar și cu câteva secole în urmă. Leonhard Euler este considerat ca fiind primul care abordează acest domeniu prin lucrarea sa scrisă în 1736, Cele șapte poduri din Königsberg, care se referă la crearea unui traseu prin orașul Königsberg trecând peste fiecare pod o singură dată, lucru ce s-a dovedit imposibil. (figura 1)

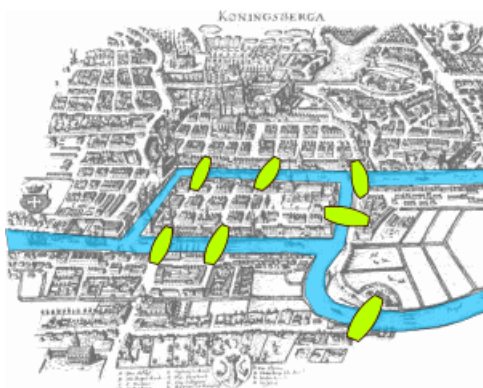


Fig. 1: Cele șapte poduri din Königsberg ¹

La 14 noiembrie 1750, Euler a scris unui prieten că a realizat importanța muchiilor unui poliedru. Aceasta a condus la formula lui poliedrală [1]:

$$V - E + F = 2$$

unde V , E și F indică numărul de vârfuri, muchii și fețe ale poliedrului. Această analiză este considerată drept prima teoremă, semnalizând nașterea topologiei.

Alte contribuții au fost făcute de Augustin-Louis Cauchy, Ludwig Schläfli, Johann Benedict Listing, Bernhard Riemann și Enrico Betti. Listing a introdus termenul "Topologie" în "Vorstudien zur Topologie", scris în limba germană.

¹https://en.wikipedia.org/wiki/Topology/media/File:Königsberg_bridges.png

4 Graf

Foarte multe probleme pot fi descrise printr-o diagrama formata dintr-un set de puncte și linii care conecteaza anumite perechi de puncte. De exemplu, punctele pot reprezenta o multime de persoane, iar liniile dintre ele relatii de rudenie, sau punctele pot fi centre de comunicare iar liniile legăturile dintre ele. Se poate observa că în astfel de diagrame importanța este pusa pe modul în care sunt conectate acele puncte, aspectul fiind nesemnificativ. O abstracție matematica a situațiilor de acest tip este conceptul de graf [8].

Un graf este o structură formată din obiecte în care sunt puse în evidență legăturile dintre ele. Obiectele corespund unor abstracții matematice numite, într-un graf, noduri/vârfuri (puncte) și fiecare legătură dintre perechile de obiecte asociate se numește muchie (arc sau linie). De obicei, un graf este reprezentat în formă schematică ca un set/grup de puncte care reprezintă nodurile, iar acestea sunt unite două câte două de linii sau curbe care corespund mulțimii muchiilor. Muchiile pot fi orientate/directe sau neorientate/indirecte.

Grafurile sunt numite astfel datorită reprezentării grafice, și datorită acestei reprezentări putem înțelege proprietățile lor mult mai ușor. Nu există un mod unic de desenare a unui graf; pozițiile relative ale punctelor care reprezintă nodurile și ale liniilor care reprezintă muchiile nu au importanță.

Din punct de vedere matematic un graf G este un triplet ordonat $(V(G), E(G), \psi_G)$ care conține un set nevid $V(G)$ de noduri, un set $E(G)$, disjunct $V(G)$, de muchii și o funcție de incidență ψ_G care asociază fiecărei muchii din G o pereche neordonată (dar nu neaparat distinctă) de noduri din G . Dacă e este o muchie și avem nodurile u și v astfel încât $\psi_G(e) = uv$, atunci se spune că e unește u și v ; nodurile u și v sunt numite capetele lui e . [8]

Majoritatea definițiilor și conceptelor în teoria grafurilor sunt sugerate de reprezentarea grafică. Capetele unei muchii sunt incidente cu muchia, și vice versa. Două noduri care sunt incidente cu o muchie comună sunt adiacente, la fel pentru două muchii care sunt incidente cu un nod comun. O muchie cu capete identice se numeste buclă.

Un graf este finit dacă ambele seturi, cel de noduri și cel de muchii, sunt finite. Totodata un graf cu un singur nod se numește trivial, iar restul grafurilor

netriviale. Un graf este simplu dacă nu are bucle și nu există doua muchii care unesc aceleași două noduri.

Doua noduri u și v sunt conexe dacă există un drum (u, v) în G , adică o secvență de noduri unice legate prin muchii care începe cu u și se termină cu v . Conectivitatea este o relație de echivalență pe setul de noduri V . Astfel există o partiționare a lui V în subseturi nevide V_1, V_2, \dots, V_n astfel încât nodurile u și v sunt conexe dacă și numai dacă sunt din același set V_i . Subgrafurile $G[V_1], G[V_2], \dots, G[V_n]$ se numesc componente ale lui G . Dacă G are o singură componentă, atunci G este conex, altfel este neconex.

Ordinul unui graf reprezintă numărul de noduri, notat cu $|V|$. Mărimea unui graf este numărul de muchii $|E|$. Gradul unui nod este numărul de muchii care sunt incidente cu el. Într-un graf de ordin n , gradul maxim al oricărui nod este de $n - 1$, iar numărul maxim de muchii este $n(n - 1)/2$.

4.1 Multigraf

Un multigraf este o generalizare în care oricare doua noduri pot avea mai multe muchii între ele. Acele muchii sunt numite și muchii paralele sau muchii multiple. Există doua noțiuni distincte despre muchii paralele [10]:

- Muchii fără identitate: identitatea unei muchii ține de cele două noduri de care aparține. În unele cazuri nevoia de a distinge muchii multiple dintre doua noduri poate lipsi, iar acele muchii sunt considerate ca o singură entitate.
- Muchii cu identitate: în acest caz fiecare muchie este considerată ca fiind o primitivă, la fel ca nodurile, iar în cazul în care între două noduri există mai multe muchii, fiecare dintre ele este considerată fiind o entitate distinctă.

Definiție matematică:

- Multigraf neorientat, cu muchii fără identitate: $G = (V, E)$ unde:

- V este un set de noduri
- E este un multiset de perechi de noduri, numite muchii.

■ Multigraf neorientat, cu muchii cu identitate: $G = (V, E, r)$, unde:

- V este un set de noduri
- E este un set de muchii
- $r : E \rightarrow \{\{x, y\} : x, y \in V\}$

4.2 Graf planar

Un graf planar este un graf care poate fi încorporat într-un plan, adică un graf care poate fi desenat astfel încât marginile sale să se intersecteze doar în noduri. Cu alte cuvinte, muchiile grafului să nu se suprapună. [11]

Matematicianul polonez Kazimierz Kuratowski a caracterizat idea de graf planar prin următoarea teoremă:

Un graf finit este planar dacă și numai dacă nu conține un subgraf care este o subdiviziune a grafului complet K_5 sau a grafului complet bipartit $K_{3,3}$. O subdiviziune a unui graf rezultă din inserarea a oricâte noduri într-o muchie.

O altă modalitate de a descrie un graf planar este prin teorema lui Wagner, exprimată în legătura cu noțiunea de graf minor:

Un graf finit este planar dacă și numai dacă nu conține grafurile K_5 sau $K_{3,3}$ ca graf minor. (figura 2) Un minor al unui graf rezultă prin contractarea unei muchii într-un nod, fiecare vecin al nodului original devenind vecin cu nodul nou.

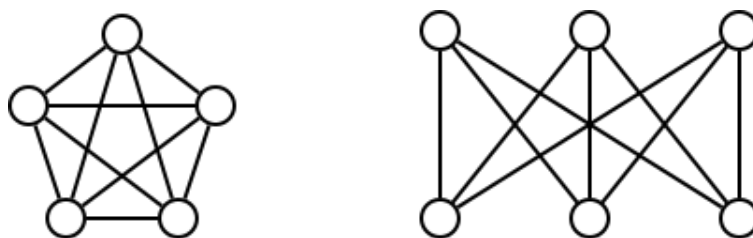


Fig. 2: Grafurile K_5 și $K_{3,3}$ ²

Alte criterii de planaritate:

În practică este dificil să folosim teorema lui Kuratowski pentru a decide într-un mod eficient dacă un graf este planar sau nu. Există și alți algoritmi pentru rezolvarea acestei probleme, pentru un graf cu n noduri cu o complexitate de $O(n)$.

Un graf simplu cu $v \geq 3$ noduri, e muchii și f fețe, trebuie să îndeplinească următoarele condiții pentru a fi planar:

- $e \leq 3v - 6$
- dacă nu sunt cicluri de lungime 3, atunci $e \leq 2v - 4$
- $f \leq 2v - 4$

Formula lui Euler:

Formula lui Euler afirmă că dacă un graf planar, finit, conectat, este desenat într-un plan și v este numărul de vârfuri, e este numărul de muchii și f este numărul de fețe (regiuni marcate de muchii, inclusiv regiunea exterioară, infinit de mare)

$$v - e + f = 2$$

²<https://www.nomachetejuggling.com/assets/k33andk5.png>

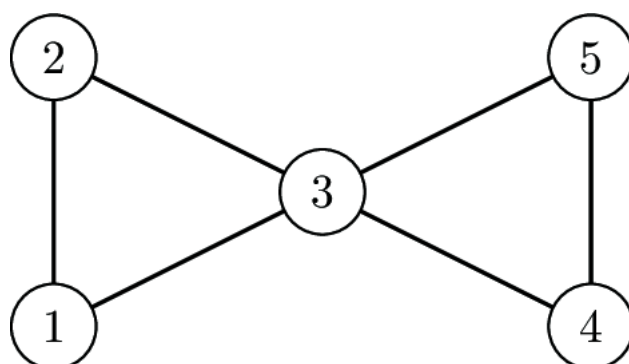


Fig. 3: Graf fluture³

Ca o ilustrare, în graful fluture dat mai sus, (figura 3) $v = 5$, $e = 6$ și $f = 3$. În general, dacă proprietatea este validă pentru toate grafurile planare cu f fețe, orice modificare a grafului care ar crea o față suplimentară, graful fiind în continuare planar ar fi și $v - e + f$ invariant. Din moment ce proprietatea este validă pentru toate grafurile cu $f = 2$, prin inducție matematică este valabilă pentru toate cazurile. Formula lui Euler poate fi demonstrată și în modul următor: dacă graful nu este un arbore, șterge o muchie care completează un ciclu. Astfel scade atât e , cât și f cu unul, lăsând $v - e + f$ constantă. Repetați până când graful rămas este un arbore; copacii au $v = e + 1$ și $f = 1$, producând $v - e + f = 2$.

4.3 Aplicare

Grafurile pot fi folosite pentru modelarea relațiilor și proceselor într-un sistem fizic, biologic, social sau informatic. Multe probleme practice pot fi reprezentate prin grafuri. În domeniul informaticii grafurile pot fi folosite pentru a reprezenta rețele de comunicare, organizarea datelor, dispozitive computerizate, fluxul de calcul, etc. De exemplu și un site web poate fi reprezentat printr-un graf, paginile web fiind nodurile iar legăturile dintre ele, link-urile, fiind muchiile. O abordare similară poate fi proiectată pentru multe alte domenii, precum ingineria

³<https://www.researchgate.net/publication/328308578/figure/fig1/AS:68235221692006501539696854150/The-butterfly-graph-five-nodes-with-two-node-pairs-connected-by-the-bridging-node-3.png>

mecanică unde putem exprima relațiile dintre mai multe elemente mecanice printr-un graf. Dezvoltarea algoritmilor pentru a gestiona grafuri este, prin urmare, de interes major în domeniul informaticii.

5 Algoritmi force-directed de desenare a grafurilor

Este o categorie de algoritmi de desenare a grafurilor într-un mod plăcut din punct de vedere estetic. Printr-un astfel de algoritm se poziționează nodurile grafului într-un spațiu bidimensional sau tridimensional astfel încât lungimea muchiilor este mai mult sau mai puțin egală, iar în structura grafului există un număr minim de suprapuneri între muchii, iar în cazuri favorabile nici o suprapunere. Desenarea grafurilor poate fi o problema grea de abordat dar prin algoritmii de desenare force-directed, se exclud noțiuni ce țin de planaritatea grafurilor, întreaga operație de desenare fiind mai degrabă o simulare cu aspecte ce se regăsesc în fizică.

Prin asignarea anumitor forțe setului de noduri și setului de muchii din graf acestea fie se apropie fie se depărtează după fiecare iterație a algoritmului, scopul algoritmului fiind de a minimiza deplasarea nodurilor sau a muchiilor până când se ajunge la o stare de echilibru. De obicei forțe elastice bazate pe legea lui Hooke sunt atribuite nodurilor unei muchii pentru ca acestea să se atragă, în schimb ce simultan forțe de repulsie, similare cu cele ale particulelor cu sarcina electrică bazate pe legea lui Coulomb, sunt folosite pentru a crea o repulsie între nodurile grafului. În stările de echilibru ale acestui sistem de forțe, muchiile nodurilor care au un părinte comun au de obicei lungimea aproape egală (datorită forței elastice de atracție), iar nodurile care nu sunt conectate printr-o muchie se resping între ele (datorită forței de repulsie). Forțele de atracție și repulsie pot fi definite și prin funcții care nu sunt bazate pe legile fizice ale elasticității sau sarcinii electrice, de exemplu unele sisteme pot folosi funcții de atracție logaritmice în loc de funcții liniare.

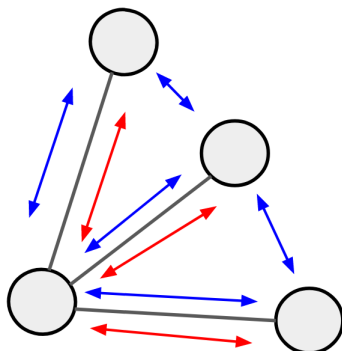


Fig. 4: Exemplu pentru aplicarea forțelor. Forța de repulsie între toate nodurile (cu albastru) și forța de atracție între nodurile adiacente (cu roșu).

O forță similară cu cea a gravitației poate fi folosită pentru a atrage nodurile către un punct fix în planul de desenare, astfel pentru un graf cu mai multe componente conexe acestea vor sta relativ în aceeași zonă și nu se vor mai depărta constant una de cealaltă datorită forței continue de repulsie și a lipsei de atracție între ele. Forțe similare cu cele magnetice pot fi aplicate într-un sistem, atât pe noduri cât și pe muchii în sine, pentru a evita suprapunerea muchiilor în întreaga simulare. [13]

5.1 Aplicare

Odată ce forțele au fost atribuite nodurilor sau muchiilor dintr-un graf, întregul comportament al elementelor este simulat ca într-un sistem fizic unde nodurile se deplasează la fiecare iterație. Acest proces este repetat iterativ până când se ajunge la o stare de echilibru mecanic, adică pozițiile relative ale elementelor nu se mai schimbă de la o iterație la alta. Structura finală după întregul proces este folosită pentru a reprezenta graful ce trebuie desenat.

Este posibilă și adăugarea unui mecanism de calculare a stării de echilibru pe lângă simularea fizică, acestea fiind metode de optimizare globală, precum algoritmi genetici.

5.2 Avantaje

■ Rezultate calitative

Pentru grafuri de mărimi medii (50-500) de noduri de multe ori rezultatul aplicării unui algoritm force-directed este unul bun și inteligibil din perspectiva omului. Rezultatul fiind catalogat după următoarele criterii:

- lungimea uniformă a muchiilor
- distribuirea uniformă a nodurilor
- număr minim de suprapuneri
- simetrie

Simetria este un criteriu greu de îndeplinit și depinde de structura internă a grafului.

■ Flexibilitate

Algoritmul ales poate fi adaptat pentru a îndeplini criterii adiționale de desenare. Exemple de extensibilitate ar fi desenarea de: grafuri orientate, grafuri 3D, grafuri împărțite în clustere, multigrafuri, grafuri ponderate

■ Intuitivitate

Având la bază analogii din fizică, precum elasticitatea, comportamentul algoritmului este ușor de prezis și înțeles.

■ Simplitate

De obicei astfel de algoritmi sunt ușor de implementat, sau au fost deja implementați în anumite biblioteci.

■ Interactivitate

Prin desenarea unor stagii intermediare ale grafului pe care este aplicat algoritmul, utilizatorul poate urmări evoluția procesului, observând cum se dezvoltă dintr-o structura încurcătă cu multe suprapuneri de muchii, într-un graf echilibrat și ușor de urmărit.

■ Fundație teoretică

Înca din anii 1960 metode de desenare ale grafulor au fost cercetate, o primă implementare fiind făcută prin algoritmul lui Tutte în anul 1963

folosind o reprezentare baricentrică (prin care se setează un grup de noduri de baza în jurul cărora celelalte noduri vor fi plasate), folosind doar forte de atracție. În următorii ani alte metode de desenare au fost create de către Eades(1984), Fruchterman și Reingold (1991), Kamada și Kawai (1989).

5.3 Dezavantaje

■ Timp de rulare mare

De obicei complexitatea unui algoritm de desenare force-directed este $O(n^3)$, unde n este numărul de noduri din graf. Acest lucru rezultă deoarece numărul iterațiilor este estimat ca fiind $O(n)$ iar pentru fiecare iterație fiecare pereche de noduri trebuie vizitată pentru calcularea forței de repulsie mutuale.

■ Minim local

Prin algoritmii force-directed se ajunge la o structură în care nodurile se află într-o stare de echilibru, întreg procesul poate fi considerat ca fiind rezolvarea unei probleme de optimizare. De cele mai multe ori starea rezultată se poate afla într-un minim local al funcției ce trebuie optimizate, dar care nu coincide neapărat cu un minim global. Întreg procesul depinde de starea inițială a nodurilor care este aleasă aleatoriu. Această problemă crește odată cu numărul de noduri din graf. O modalitate de rezolvare ar fi combinarea a mai multor algoritmi force-directed pentru obținerea unui rezultat mai bun. De exemplu folosirea algoritmului Kamada–Kawai pentru crearea unei așezări în plan inițiale, iar apoi îmbunătățirea structurii cu ajutorul algoritmului Fruchterman–Reingold.

5.4 Metode de desenare

5.4.1 Metoda baricentrică

Metoda baricentrică de desenare a lui Tutte din 1963 este considerată prima versiune a unui algoritm force-directed pentru obținerea unei structuri cu muchii drepte, fără suprapuneri pentru un graf planar 3-conex. În comparație cu alte metode de desenare această varianta garantează că fețele grafului planar sunt convexe. Ideea în spatele algoritmului constă în faptul că dacă o față a grafului planar este fixată în plan, atunci pozițiile potrivite ale celorlalte noduri

sunt găsite rezolvând un set de ecuații liniare, unde fiecare poziție a unui nod este reprezentată ca o combinație convexă a pozițiilor nodurilor vecine lui. În acest model forța unei muchii (u, v) este direct proporțională cu distanța dintre nodul u și v în plan. Astfel forța la un nod v este descrisă prin

$$F(v) = \sum_{(u,v) \in E} p_u - p_v$$

unde p_u și p_v sunt pozițiile nodurilor. Deoarece această funcție are un minim cu toate nodurile așezate în aceeași locație, setul de noduri trebuie împărțit în noduri fixe și noduri libere, care pot fi deplasate. [12]

Algorithm 1 Metoda baricentrică de desenare [12]

Input: $G = (V, E)$;

partiționarea lui V într-un set V_0 cu cel puțin 3 noduri fixe și un set V_1 cu noduri libere, $V = V_0 \cup V_1$;

un poligon convex P cu $|V_0|$ vârfuri.

Output: o poziție p_v pentru fiecare nod din V astfel încat nodurile fixe formeaza un poligon.

- 1: Plasează fiecare nod $u \in V_0$ la un varf al lui P , și fiecare nod liber în origine
 - 2: repeat
 - 3: for each $v \in V_1$ do
 - 4: $x_v = \frac{1}{deg(v)} \sum_{(u,v) \in E} x_u$
 - 5: $y_v = \frac{1}{deg(v)} \sum_{(u,v) \in E} y_u$
 - 6: end for
 - 7: until x_v și y_v converg pentru toate nodurile libere v
-

5.4.2 Algoritmul Fruchterman–Reingold

Algoritmul Fruchterman–Reingold de desenare a grafurilor introduce noțiunea de temperatură. Această variabilă are o valoare setată la început iar ea scade la fiecare iterație până la 0. Temperatura controlează deplasarea nodurilor iar cu cât ne apropiem de un minim, cu atât temperatura va fi mai mică pentru a nu schimba cu mult rezultatul.[12]

Algorithm 2 Fruchterman si Reingold [12]

$arie = W * L$; W și L sunt lățimea și lungimea suprafeței de desenare

$G = (V, E)$; nodurile au poziții inițiale aleatorii

$k = \sqrt{arie/|V|}$

$f_a(x) = x^2/k$

$f_r(x) = k^2/x$

```
1: for  $i = 1$  to  $iteratii$  do
2:   calcularea forțelor de repulsie
3:   for each  $v \in V$  do
4:     fiecare nod are doi vectori  $pos$  și  $disp$ 
5:      $v.disp = 0$ 
6:     for each  $u \in V$  do
7:       if  $u \neq v$  then
8:          $\delta$  este vectorul diferență dintre pozițiile celor două noduri
9:          $\delta = v.pos - u.pos$ 
10:         $v.disp = v.disp + (\delta/|\delta|) * f_r(|\delta|)$ 
11:      end if
12:    end for
13:  end for
14:  calcularea forțelor de atracție
15:  for each  $e \in E$  do
16:    fiecare muchie este o perche de noduri  $u$  și  $v$ 
17:     $\delta = e.v.pos - e.u.pos$ 
18:     $e.v.disp = e.v.disp - (\delta/|\delta|) * f_a(|\delta|)$ 
19:     $e.u.disp = e.u.disp + (\delta/|\delta|) * f_a(|\delta|)$ 
20:  end for
21:  deplasarea maximă se limitează în funcție de temperatura  $t$ 
22:  for each  $v \in V$  do
23:     $v.pos = v.pos + (v.disp/|v.disp|) * \min(v.disp, t)$ 
24:     $v.pos.x = \min(W/2, \max(-W/2, v.pos.x))$ 
25:     $v.pos.y = \min(L/2, \max(-L/2, v.pos.y))$ 
26:  end for
27:  reducerea temperaturii
28:   $t = cool(t)$ 
29: end for
```

6 Proiecția ortografică

Proiecția ortografică (sau proiecția ortogonală) este o modalitate de a reprezenta un model tridimensional într-un spațiu bidimensional. Este o forma de proiecție paralelă, în care toate liniile de proiecție sunt ortogonale cu planul de proiecție. (figura 5)

Termenul ortografic este uneori rezervat în mod specific pentru reprezentări ale obiectelor în care axele principale sau planurile obiectului sunt de asemenea paralele cu planul de proiecție, dar acestea sunt mai bine cunoscute ca proiecții multiview. Mai mult, atunci când planurile sau axele principale ale unui obiect într-o proiecție ortografică nu sunt paralele cu planul de proiecție, dar sunt înclinate mai degrabă pentru a descoperi mai multe laturi ale obiectului, proiecția se numește o proiecție axonometrică. Subtipurile de proiecție multiview includ planuri, elevații și secțiuni. Subtipurile de proiecții axonometrice includ proiecții izometrice, dimetrice și trimetrice. [5]

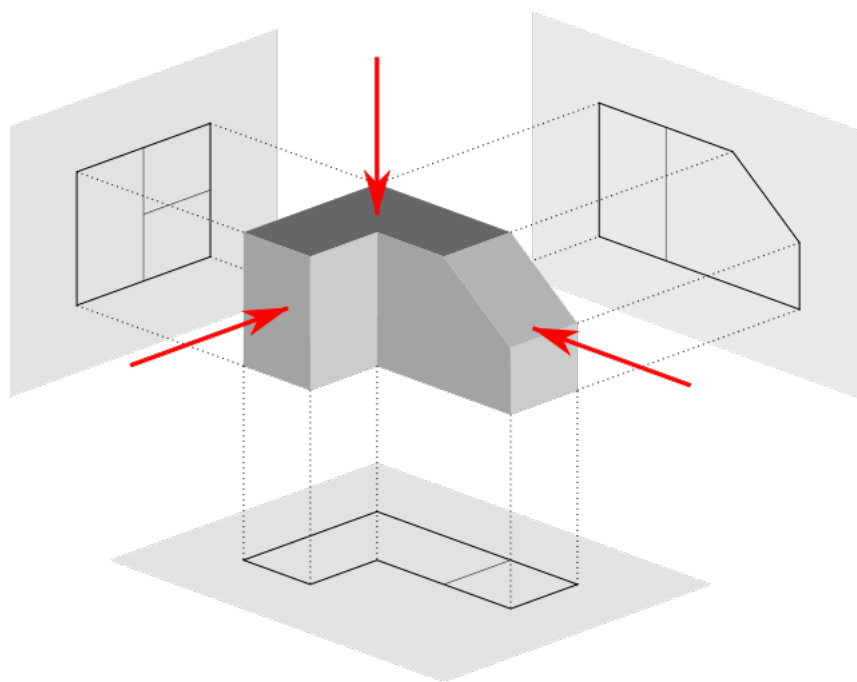


Fig. 5: Proiecția ortografică a unui corp ⁴

O proiecție ortografică simplă, pe planul $z = 0$ poate fi definită de următoarea matrice [4]:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Pentru fiecare punct $v = (v_x, v_y, v_z)$, punctul transformat ar fi:

$$P_v = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix}$$

6.1 Aplicare

Acest concept a fost aplicat pentru reprezentarea elementelor care formează integral model 3D. Setul de date nu conține dimensiunile exacte ale elementelor ci doar punctele de legare cu late elemente și centrul de greutate. Prin calcularea punctelor simetrice ale punctelor de legare la centrul de greutate și prin încadrarea unui dreptunghi în jurul lor obținem forma care v-a reprezenta elementul în sine.

Aplicația conține trei moduri de afișare a modelului, fiecare perspectivă folosind coordonatele (x, y, z) din setul de date eliminând una din ele pentru reprezentarea în 2D:

- Perspectiva de deasupra, folosind (x, y)
- Perspectiva laterală, folosind (x, z)
- Perspectiva frontală, folosind (y, z)

⁴<https://www.kisspng.com/png-graphical-projection-orthographic-projection-multi-1362397/download-png.html>

7 Tehnologii și limbaje folosite

Fiind o aplicație software am folosit limbajul de programare C++ împreună cu biblioteca Qt pentru dezvoltare. Deoarece aplicația are la bază date din exterior, anume fișiere .mdef care pot conține multe elemente diferite, C++ este eficient pentru procesarea lor dar și pentru anumiți algoritmi care se află în spatele vizualizării datelor⁵. Limbajul C++ neavând suport nativ pentru interfețe grafice, folosirea unei biblioteci grafice, Qt, s-a dovedit esențială. Totodată Qt conține clase pentru lucrul cu fișiere cu structură de XML.

7.1 C++

C++ este un limbaj de programare creat de Bjarne Stroustrup ca o extensie a limbajului C, inițial fiind numit "C cu clase". Limbajul s-a extins semnificativ, C++ modern având caracteristici orientate pe obiect, de programare generica și funcțională precum și accesul la manipularea memoriei. Este de obicei implementat ca limbaj compilat, iar mulți furnizori oferă compilatoare C++, inclusiv LME, LLVM, Microsoft, Intel și IBM, astfel încât limbajul să poată fi compilat pe mai multe platforme.

A fost conceput cu o înclinare către programarea sistemelor și programarea embeded, pentru software limitat de resurse și sisteme mari, punând în prim plan performanța, eficiența și flexibilitatea ca și caracteristici esențiale ale limbajului. C++ s-a dovedit fiind util și în alte contexte precum aplicații desktop, servere, centrale telefonice.

C++ este standardizat de către Organizația Internațională pentru Standardizare (ISO), cu cea mai recentă versiune standard publicată de ISO în decembrie 2017 ca ISO / IEC 14882: 2017 (informal cunoscut sub numele de C++17). Înainte de standardizarea inițială din 1998, C++ a fost dezvoltat de Bjarne Stroustrup la Bell Labs din 1979, ca extensie a limbajului C; el dorea un limbaj eficient și flexibil similar C, care să ofere și caracteristici de nivel înalt pentru organizarea programelor. [15]

⁵Algoritmul de desenare force-directed necesită un număr considerabil de iterații.

7.2 Qt

Qt este un toolkit open-source și gratis folosit la crearea de interfețe grafice pentru aplicații cross-platform care pot fi rulate pe diverse platforme hardware și software precum Linux, Windows, macOS, Adroid sau sisteme embedded, cu puține sau fără schimbări în codul de baza dar păstrând funcționalitățile native. Qt este momentan dezvoltat de The Qt Company sub Qt Project prin guvernare open-source, în care sunt implicate mai multe organizații și dezvoltatori individuali cu scopul de aduce noi funcționalități bibliotecii Qt.

Majoritatea programelor GUI dezvoltate cu Qt au o interfață nativă similară cu cea a sistemului pe care este rulat. Aplicații precum unelte ce pot fi rulate din linia de comandă sau console pentru servere, fără interfață grafică pot fi dezvoltate în Qt. Un exemplu fiind framework-ul web Cutelyst.

Qt suportă diverse compilatoare precum GCC C++ și MSVC de la Visual Studio. Qt conține și Qt Quick, care include propriul limbaj de scriptare numit QML, care a facilitat dezvoltarea rapidă a aplicațiilor mobile păstrând opțiunea de a scrie logică în C++ pentru cele mai bune rezultate în legătură cu performanța.

Alte funcționalități includ acces la baze de date prin SQL, cititor de XML, cititor de JSON, managementul thread-urilor. [16]

Qt este construit pe aceste concepte:

■ Abstractizarea interfeței grafice

La început Qt folosea propriul motor de desenare pentru a emula aspectul diferitelor platforme de pe care rula. Astfel doar câteva clase depindeau de platforma făcând portarea mult mai ușoară, însă emularea nu era mereu perfectă. Versiunile mai noi de Qt folosesc API-uri native de pe platforma de rulare, pentru a rezolva problemele din trecut.

■ Signals și slots

Este un concept introdus în Qt pentru comunicarea între obiecte, având la baza conceptul șablonului de proiectare observer evitând cod boilerplate. Obiectele ce țin GUI pot transmite semnale ce conțin informații relative la un eveniment care pot fi captate de alte obiecte care conțin socluri.

■ Compilator de metaobiecte

Numit și moc, este o unealtă care rulează pe codul unei aplicații Qt. Interpretează anumite macro-uri scrise în codul sursă C++ și le folosește pentru a genera cod C++ cu ajutorul unor informații legate de clasele din aplicație. Acest concept este folosit pentru a adăuga funcționalități care nu sunt prezente nativ în limbajul C++: semnale și socluri, introspecție, apelări asincrone de funcții.

■ Legături între limbaje

Qt poate fi folosit și de alte limbaje precum Python, Javascript, C# sau Rust.

7.2.1 Graphics view framework

Graphics View pune la dispoziție un mediu în care se pot controla și se poate interacționa cu un număr mare de elemente grafice 2D. Framework-ul include și o arhitectura de propagarea a evenimentelor care facilitează interacțiunea cu elementele din scena. Elementele suportă evenimente legate de taste, mișcarea mouse-ului, evenimente de dublu click, etc.

Graphics View folosește un arbore BSP (Binary Space Partitioning) care oferă o accesare rapidă la oricare element, astfel se poate vizualiza și interacționa cu scene cu milioane de elemente grafice în timp real.

Graphics View furnizează o abordare bazată pe elemente a programării model-view. Mai multe view-uri pot observa o scena, iar scena conține elementele grafice compuse dintr-o varietate de forme geometrice. [14]

■ Scena

Clasa QGraphicsScene oferă următoarele funcționalități:

- O interfață pentru controlarea unui număr mare de elemente grafice.
- Propagarea evenimentelor din exterior către elemente.
- Setarea stării unui elemente, ex: selecție, focus.

Scena reprezintă un container pentru obiecte de tip `QGraphicsScene`. Elementele sunt adăugate prin funcția `addItem()`, iar apoi pot fi accesate folosind una din multele funcții de returnare. Funcția `items()` și supraincarcările ei, returnează toate elementele grafice care se intersectează sau care sunt conținute de un punct, un dreptunghi, un poligon sau un vector. `itemAt()` returnează elementul de la cel mai înalt nivel de suprapunere într-un anumit punct. Toate funcțiile returnează elementele într-o ordine descendentă în legătura cu eventuale suprapuneri.[14]

```
QGraphicsScene scene;  
QRectF rectangle(0, 0, 100, 100);  
QGraphicsRectItem *rectItem = scene.addRect(rectangle);  
QGraphicsItem *item = scene.itemAt(50, 50);
```

Arhitectura de propagare a evenimentelor a clasei `QGraphicsScene` asigură transmiterea evenimentelor către elemente dar și propagarea evenimentelor între ele. Dacă scene primește un eveniment legat de mouse la o anumită poziție, scena transmite mai departe evenimentul către toate elementele care se afla în acel punct.

Scena totodată poate controla starea elementelor, de exemplu dacă elementele sunt selectate, sau sunt în focus relativ cu evenimente de la tastatură. Mai multe elemente pot fi selectate folosind `setSelectionArea()`. Pentru a accesa toate elementele selectate se poate folosi `selectedItems()`. O altă stare tratată este cea de focus pe un anumit element. Se poate seta prin apelarea funcției `setFocusItem()` sau `setFocus()`, iar pentru a accesa elementul curent în focus `focusItem()`.

■ View

`QGraphicsView` furnizează un widget prin care se poate vizualiza conținutul unei scene. Se pot atasa mai multe view-uri la aceeași scena dacă este nevoie. [14]

```
QGraphicsScene scene;  
myPopulateScene(&scene);  
  
QGraphicsView view(&scene);
```

```
view . show ( ) ;
```

View-ul poate primi evenimente din exterior de la tastatură sau mouse și le trimite mai departe la scene (făcând conversia la coordonatele scenei dacă este nevoie, deoarece în unele cazuri când scene este mai mare decât fereastra view-ului, view-ul nu poate afișa toată scena, ci doar o porțiune).

Folosind matricea de transformare, `transform()`, view-ul poate scala sau roti scene. Pentru conveniență `QGraphicsView` oferă funcții pentru convertirea la, sau de la, coordonatele scenei la coordonatele view-ului. `mapToScene()` și `mapFromScene()`.

■ Elementul

`QGraphicsItem` este clasa de baza pentru elementele grafice dintr-o scena. `Graphics View` oferă câteva elemente grafice standard pentru forme simple, precum dreptunghi (`QGraphicsRectItem`), elipsa (`QGraphicsEllipseItem`), dar și text (`QGraphicsTextItem`). De obicei utilizatorul specializează clasa de baza prin moștenire pentru a obține elemente grafice de forme diferite.

`QGraphicsItem` suportă următoarele functionalitati:

- evenimente de mouse (click, hover, scroll-wheel, etc.)
- evenimente de tastatură
- evenimente meniu de context
- grupare, fie prin relație părinte-copil, fie prin clasa `QGraphicsItemGroup`
- detecție de coliziuni

Toate elementele se afla într-un sistem de coordonate local, și la fel ca `QGraphicsView`, conțin funcții pentru convertirea la coordonatele scenei. Totodată la fel ca și `QGraphicsView` sistemul de coordonate poate fi transformat folosind o matrice, `QGraphicsItem::transform()`. Acest lucru este folositor la scalarea sau rotirea individuala.

Elementele pot conține și alte elemente (copii). Transformările părintelui sunt moștenite de toți copiii.

`QGraphicsItem` suportă și detecție de coliziuni, prin funcțiile `QGraphicsItem::shape()` și `QGraphicsItem::collidesWith()`, ambele funcții virtuale care pot fi suprascrise.

7.3 Aplicare

Majoritatea acestor noțiuni au fost folosite la dezvoltarea aplicației, interfața grafică a aplicației având la bază Qt. Fiecare tab deschis conține un `QGraphicsView` cu o scenă cu toate elementele. View-ul este dinamic iar utilizatorul poate interacționa cu el, prin zoom sau prin schimbarea întregii scene, selectând diferite perspective.

Fiecare element grafic din aplicație, moștenește clasa de bază `QGraphicsItem`, fiind specializat pentru diferitele tipuri care trebuie afișate. De exemplu obiectele de tip `Motion body` sunt afișate ca dreptunghiuri albastre de diferite forme în funcție de datele din fișierul sursă. Aceste elemente pot fi mutate cu mouse-ul. Elementele de tip `Motion body` pot fi conectate de alte elemente `Motion body` fie prin printr-un `Joint` sau `Connector`. La fel pentru `Joint` și `Connector` s-au folosit implementări separate pentru afișarea diferitelor tipuri.

Utilizatorul poate interacționa cu toate elementele prin click-dreapta pentru a deschide un meniu de context. Prin acest meniu el poate schimba culoarea, reseta culoarea sau deschide un panou în care sunt afișate informații adiționale elementului.

8 Grafica turtle

Este o modalitate de desenare care se bazează pe câteva rutine simple. Ne putem imagina ca fiind un cursor care desenează linii drepte într-un plan cartezian. Cursorul are trei atribute: o locație, o orientare (sau o direcție) și un pix. Pixul are de asemenea, atribute: culoare, lățime și stare(desenare / inchis).

Cursorul se muta cu comenzi care sunt relative la poziția sa, cum ar fi "înainte 10 pixeli" și "roțiți cu 90 de grade". Pixul purtat de cursor poate fi de asemenea controlat, având posibilitatea de a îi schimba culoarea sau lățimea.

Un sistem complet de grafica necesita un flux de control, proceduri si recursivitate. Din aceste metode se pot construi forme mai complexe, cum ar fi pătrate, triunghiuri, cercuri și alte figuri compuse. Ideea de grafica turtle, de exemplu, este utilă într-un sistem Lindenmayer pentru generarea de fractali.

Geometria graficii turtle funcționează oarecum diferit față de geometria carteziană, fiind bazată în primul rând pe noțiunea de vector (direcția relativă și distanța față de punctul de pornire). [7]

8.1 Aplicare

```
class Turtle
{
public:
    Turtle(double x, double y, double dir);
    void rotate(double angle);
    void forward(double length, bool draw = true);
    Path getPath() const;
private:
    double x, y, dir;
    Path path;
};
```

De exemplu pentru a desena un triunghi echilateral cu lungimea laturilor de 100 de unități.[6]

```
forward(100);
rotate(PI/3);
forward(100);
rotate(PI/3);
forward(100);
```

Fiecare comanda schimba starea Turtle-ului si a traseului desenat. Cele trei comenzi de forward desenează laturile triunghiului de lungime 100, adăugând latura la traseul total parcurs. Comenzile de rotate schimba direcția de desenare cu 60° ⁶. După ultima comandă de forward pixul se afla în punctul inițial de desenare, dar datorita lipsei unei comenzi rotate de 60° , direcția de desenare este diferită de cea inițială.

Folosind structuri repetitive putem abstractiza metoda de desenare. De exemplu pentru desenarea unui poligon regulat convex[6]:

```
polygon(nrOfEdges){
  for(i=1; i<nrOfEdges; i++){
    forward(2*PI/nrOfEdges);
    rotate(2*PI/nrOfEdges);
  }
}
```

Pentru `nrOfEdges=3` avem din nou un triunghi echilateral, pentru `nrOfEdges=4` avem un pătrat, și așa mai departe. Folosind un `nrOfEdges` destul de mare, lungimea laturii scade si putem ajunge la o forma care într-un spațiu discret este similară cu un cerc.

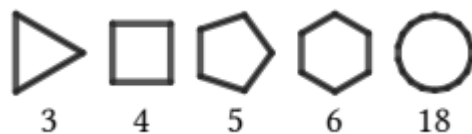


Fig. 6: Desenarea unei forme similare cu un cerc ⁷

Extinzând aceste metode elementare de desenare putem ajunge la modalități de desenare pentru simboluri mai complexe formate din forme geometrice

⁶in exemplul pentru desenare am folosit *PI* ca reprezentand 180° , astfel $180^\circ \div 3 = 60^\circ$

⁷imagine preluata din [6]

simple. (ex desenarea proiecției unui cilindru într-un plan 2D, anexa 12.2).

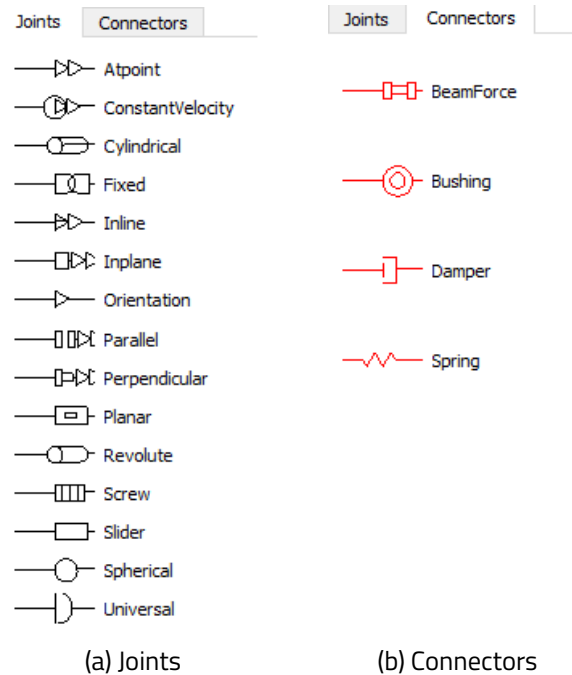


Fig. 7: Simbolurile desenate prin turtle graphics pentru fiecare tip de connector sau joint

Pentru implementarea mea a acestui concept am adăugat funcțiile `save()` și `load()` pentru a eficientiza procesul de desenare. Funcția `save()` salvează starea curentă a clasei, și anume coordonatele x, y și direcția în acel moment. Astfel dacă vrem să ne întoarcem într-un punct precedent nu mai este nevoie să facem pașii necesari să ne întoarcem în acel punct, ci putem apela direct funcția `load()`.

Secvența de cod corespunzătoare implementării structurii de stare:

```
struct State
{
    State(double x, double y, double dir)
    double m_x;
    double m_y;
    double m_dir;
};
```

Clasa adaptată:

```
class Turtle
{
public:
    Turtle(double x, double y, double dir);
    Turtle(QPointF start, double dir);
    void rotate(double angle);
    void forward(double length, bool draw = true);

    void save();
    void load();

    QPainterPath getPath() const;
private:
    QPainterPath m_path;
    State m_current;
    State m_saved;
};
```

9 Setul de date

Setul de date este un fișier cu extensia .mdef dar care structura identica cu cea de XML. Mdef provine de la "motion definition file", fișierul conținând date precum submecanisme, elementele care compun întregul model, legăturile dintre ele și modul cum interacționează în întregul sistem. (anexa 12.1) În aplicația dezvoltată sunt evidențiate aceste date, scopul aplicației fiind de a ușura înțelegerea modelului de către utilizator fără a fi nevoit să citească direct fișierul .mdef de bază.

9.1 XML

Extensible Markup Language (XML) este un meta-limbaj de marcare care definește un set de reguli de scriere astfel încât să poată fi citit și de oameni dar și de calculator într-un mod eficient. X-ul din XML vine de la eXtensibil, ceea ce înseamnă că acest limbaj poate fi adaptat și proiectat în funcție de nevoia utilizatorului. Trebuie remarcat faptul că, în ciuda numelui său XML în sine nu este un limbaj de marcare, este un set de reguli prin care utilizatorul își poate dezvolta propriul limbaj.[2]

XML este un standard aprobat de W3C pentru documentele de marcare. Acesta oferă o sintaxă generică folosită pentru a marca date din document cu tag-uri, într-un mod flexibil care poate fi adaptat pentru a satisface anumite cerințe din domeniul în care este aplicat.

9.2 Aspecte

1. Descriptiv

XML oferă utilizatorilor libertatea de a-și crea propriul limbaj de marcare pentru un scop specific. Astfel se pot crea un număr nelimitat de limbaje pentru a satisface o anumită necesitate.

2. Meta-date

Meta-datele se referă la "date care descriu date". Acest lucru este esențial deoarece degeaba avem date dacă nu știm cum să le interpretăm. XML are o metoda standard și sintaxa pentru a expune atât datele cât și meta-datele. De exemplu pentru <country>Romania</country>, "country" reprezintă meta-data, iar "Romania" este data în sine. Astfel putem avea mai multe date atribuite aceleiași meta-date [3]:

```
<country>Romania</country>  
<country>France</country>
```

3. Portabilitate

XML oferă potențialul de partajare a datelor pe diferite platforme. Scopul de baza din spatele XML este scrierea documentelor într-un mod care poate fi transmis de la un mediu la altul păstrând integritatea datelor. Acest lucru este facilitat prin faptul ca documentele XML sunt text și astfel orice instrument ce poate citi un document text poate interpreta un document XML.

4. Structura neambigua

Chiar dacă XML este flexibil în definirea elementelor este strict în alte aspecte, iar utilizatorii trebuie sa urmeze un set de reguli predefinite. Aceste reguli restricționează modul în care un document este scris astfel încât sa nu exista ambiguitate în interpretarea numelor, ordinii elementelor sau ierarhiei de elemente. Astfel se minimizează eventuale erori ce pot apărea și complexitatea textului, iar cititorii de XML pot interpreta datele cu ușurință fără apariția de erori pe parcurs.

Utilizatorii sunt, de asemenea, liberi să creeze reguli privind modul în care ar trebui să arate documentele. Definirea tipului de document (DTD) și schemele XML sunt instrumentele care ajută la acest proces.

9.3 Terminologie specifică XML

1. Caracter - un document XML este format din string-ului de caractere. Aproape orice caracter unicode poate apărea într-un document.

2. Marcaj - documentul este împărțit în marcaje și conținut care pot fi observate după reguli sintactice. În general textul care formează în marcaj începe cu caracterul "<" și se termina cu ">".

3. Tag - reprezintă un marcaj și poate fi :

- de început : <sectiune>
- de sfârșit : </sectiune>
- fără conținut : <line-break/>

4. Element - este o componenta a documentului care începe cu un tag de început și se termina cu un tag de sfârșit corespunzător. Datele între tag-uri con-

stituie conținutul elementul care poate cuprinde și alte elemente.

ex: `<greeting>Hello, world!</greeting>`

5. Atribut - un atribut constă într-o pereche de tip nume-valoare care se poate afla într-un tag de început sau într-un tag fără conținut.

Un exemplu este ``, unde numele atributului este `src` iar valoarea este `image.jpg`. Un atribut poate avea o singură valoare și poate apărea cel mult o dată în tag-ul unui element.

6. XML prolog - Un prolog este linia inițială într-un document XML. Conține de obicei versiunea de XML și tipul de encoding.

De exemplu: `<?xml version="1.0" encoding="UTF-8">`

7. Arborele XML Structura unui document XML poate fi considerată ca fiind un arbore. Documentul începe cu o rădăcină și se ramifică în mai multe frunze. Elementul rădăcină conține toate elementele documentului și nu are părinte, ca restul elementelor.

9.4 Utilizabilitate XML

Aplicația lucrează cu trei tipuri de elemente: Motion body, Joint, Connector. Elementele Motion body sunt obiecte care sunt conectate între ele fie cu elemente Joint fie cu elemente Connector. Joint reprezintă un punct fix între două elemente Motion body, iar Connector este un element flexibil care face legătura dintre două elemente de tip Motion body. Fișierul de bază `.mdef` conține toate informațiile necesare pentru a defini modul prin care interacționează aceste elemente. Buna funcționare a aplicației ține de structura documentului.

10 Simcenter

CAE(Computer-aided engineering) este un termen care definește un tip de aplicație folosit în domeniul ingineriei. Aceste aplicații sunt folosite pentru a rezolva diferite probleme ce țin de arii științifice precum: analiza finită a elementelor (FEA), mecanica fluidelor numerică (CFD), durabilitatea materialelor și optimizare. Acest tip de aplicație este folosit pentru a analiza rezistența și performanța anumitor componente sau ansambluri de componente, eficientizând timpul și materia necesară în întregul proces, în special datorită aspectului digital. Este folosit în multe domenii precum industria de auto-motive, aviație și aeronautică. [17]

Ariile acoperite de CAE sunt:

- analiza stresului anumitor componente folosind FEA
- analiza fluidelor prin CFD
- cinematica
- simularea formării unei componente prin turnarea sau presarea materialelor de construcție
- optimizarea produsului
- optimizarea procesului de creare

În general sunt trei etape care formează întregul proces de analiză:

- pre-procesare: se definește modelul și factorii de mediu
- analiză: proces care este rulat pe o unitate de calculare performantă
- post-procesarea: se afișează rezultatele utilizând aplicații de vizualizare

Acest ciclu este rulat iterativ de mai multe ori fie manual fie prin aplicații de optimizare.

Aplicațiile CAE sunt folosite foarte des în industria de automotive. Folosirea lor a dat posibilitatea de a reduce costul și timpul de producție în timp ce au îmbunătățit siguranța, confortul și durabilitatea vehiculelor produse. S-a ajuns în

punctul în care se preferă testarea produselor într-un mediu digital decât pe un prototip fizic.[17]

Simcenter Motion este o aplicație software CAE folosită pentru animarea și analizarea mecanismelor cinematice și dinamice în legătură cu puncte de design critice, forțe, viteze, accelerații. [18]

Într-o simulare cinematică:

- sarcinile externe și forțele de inerție afectează forțele de reacție ale corpurilor dar nu afectează mișcarea
- corpurile și legăturile dintre ele sunt rigide
- nu există bușe sau alte elemente care se pot deforma

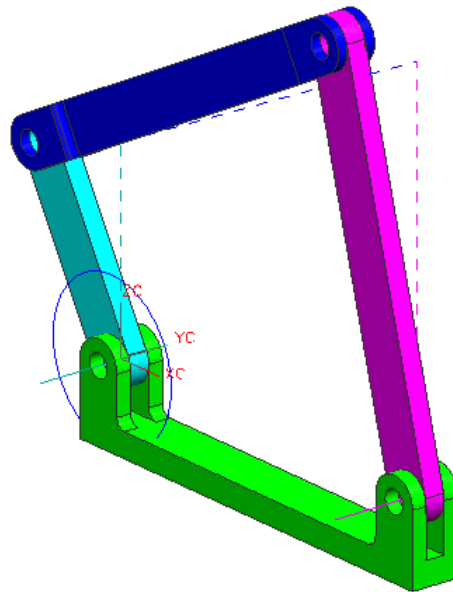


Fig. 8: Model cinematic⁸

⁸ imagine preluată din [18]

Într-o simulare dinamică:

- sarcinile externe și forțele pot genera mișcare
- există bucșe pentru simularea legăturilor de supunere
- există efecte ce țin de forțe de frecare și contact
- se poate rula o analiză de echilibru static în care toate forțele externe și interne sunt în echilibru

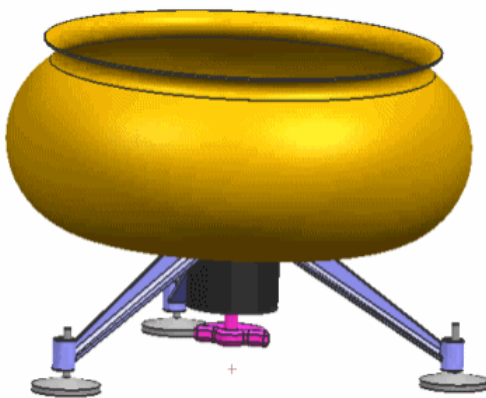


Fig. 9: Model dinamic ⁹

În aplicație un mecanism se definește ca fiind un ansamblu de componente rigide care se mișcă coeziv. Pentru a crea un mecanism este necesară:

- specificarea corpurilor de mișcare și a corpurilor statice
- constrângerea mișcării dintre corpuri, prin definirea mișcării unui corp în legătură cu celelalte corpuri. Acestea se creează prin legături dintre corpurile de mișcare.
- setarea întregii mișcări ale mecanismului

⁹ imagine preluată din [18]

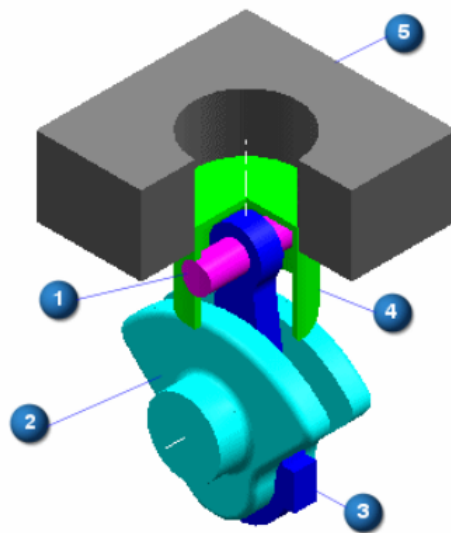


Fig. 10: Model mecanism ¹⁰

Corpurile de mișcare reprezintă elemente rigide în mecanism. Orice componentă din model ce se poate mișca trebuie inclusă într-un corp de mișcare. Un corp de mișcare poate fi reprezentat fie printr-o componentă de asamblare fie prin corpuri geometrice simple solide, linii curbe sau puncte. De obicei este mai ușor în procesul de creare a mecanismului să fie definit inițial prin elemente simple precum linii curbe și puncte pentru a verifica un comportament elementar ca apoi detaliile să fie adăugate.

¹⁰ imagine preluată din [18]

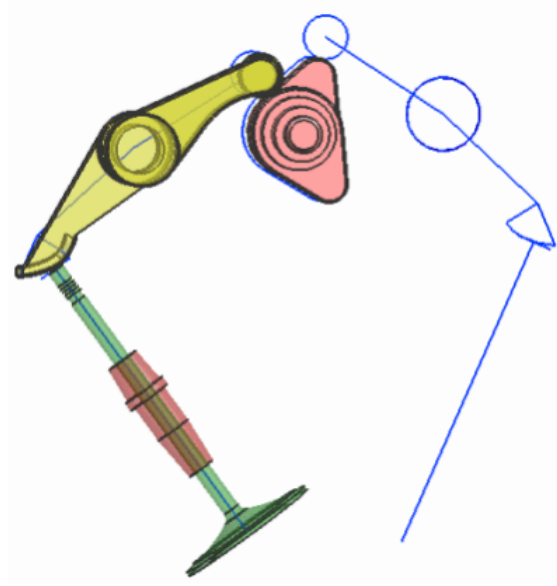


Fig. 11: Model mecanism format din forme simple¹¹

Într-un mecanism un grad de libertate descrie cum un corp de mișcare se poate deplasa relativ la celelalte corpuri. Toate gradele de libertate ale unui mecanism descriu toate mișcările independente posibile ale lui. Fără constrângeri un corp dintr-un mecanism plutește în spațiu având șase grade libertate: trei grade de translație (direcția X, Y sau Z) și trei grade de rotație (în jurul axelor X, Y sau Z). O îmbinare ale corpurilor constrânge unu sau mai multe grade libertate. [18]

Mișcarea unui corp este definită ca fiind mișcarea lui relativă la un corp de baza de care este legat, sau la pământ dacă nu exista.

Aplicația creată are rolul de vizualiza relațiile dintre elementele prezente în model, ținând cont de tipul lor și de tipul de legătura dintre ele.

¹¹ imagine preluată din [18]

11 Funcționalități

În acest capitol sunt prezentate funcționalitățile implementate în aplicația dezvoltată. Aplicația afișează o diagramă care are la baza un model 3D format din mai multe elemente, punând accentul pe relațiile de legătura dintre elemente.

Funcționalități de bază:

- încărcarea oricărui fișier .mdef și crearea diagramei
- modificarea diagramei create
 - mutarea elementelor
 - scalarea întregii diagrame
 - schimbarea culorii anumitor elemente
- salvarea modificărilor într-un fișier separat
- încărcarea automată a fișierului salvat, dacă există
- afișarea modelului din mai multe perspective ortografice
- afișarea modelului sub formă de graf
- căutarea unui element după un cuvânt cheie și evidențierea lui
- compararea dintre diferite versiuni de modele
- afișarea unui panou de tip legendă care prezintă diferitele tipuri de elemente
- afișarea unui panou de informații în care sunt afișate informații adiționale legate de elementele selectate

În continuare sunt prezentate caracteristicile aplicației în detaliu. Pentru demonstrarea funcționalităților am folosit un fișier .mdef pe baza modelului următor. Modelul a fost ales datorita structurii simetrice și a tipurilor diferite de legături din model.

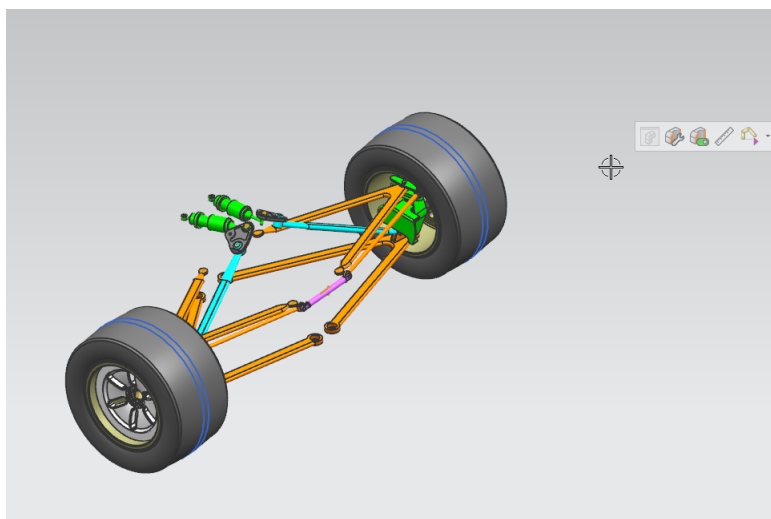


Fig. 12: Porțiunea de suspensii de la o mașină

11.1 Încarcarea fișierelor

Se pot încărca oricâte fișiere .mdef în aplicație, fiecare fișier fiind deschis într-un tab diferit. Acest lucru se face din meniul *File* de unde se alege opțiunea *Open*, care v-a deschide o nouă fereastră de unde utilizatorul își poate alege fișierul. Odată ales un fișier el este deschis într-un tab nou care conține o scenă cu mai multe elemente de tip *Motion body*, *Joint* sau *Connector*.

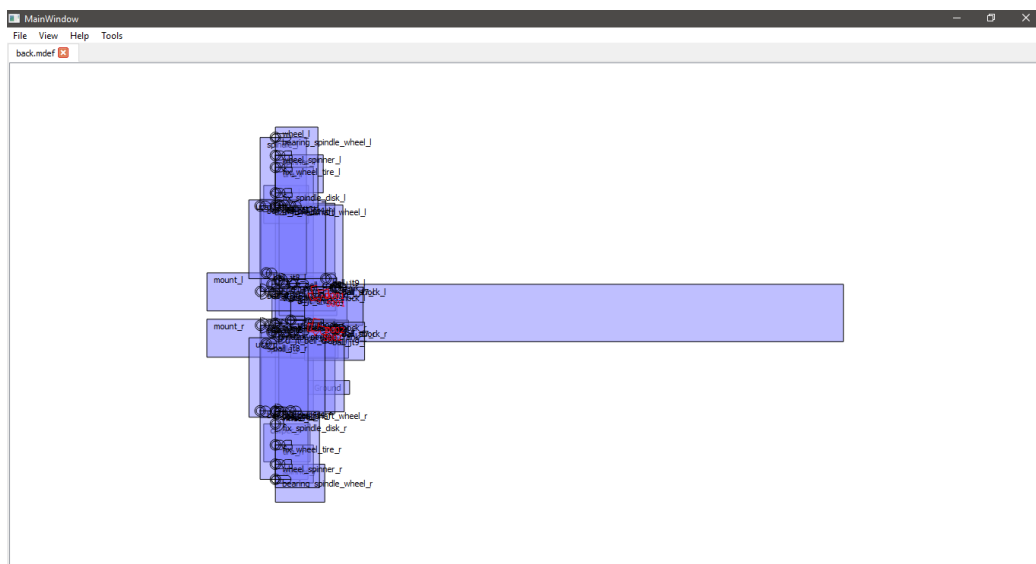


Fig. 13: Modelul menționat anterior deschis în aplicație

11.2 Modificarea diagramei

Diagrama creată se poate modifica prin:

- schimbarea perspectivei ortografice
- afișarea sub formă de graf
- scalare
- modificarea pozițiilor elementelor
- modificarea culorii elementelor

Una dintre problemele întâmpinate în procesul de dezvoltare a aplicației a fost suprapunerea elementelor în anumite perspective. Această problemă a fost rezolvată prin două metode: prin scalare, care păstrează pozițiile relative ale elementelor în model, și prin afișarea modelului sub forma unui graf, care nu ia în considerare aspecte legate de poziții sau forme.

Scalarea este necesară pentru modele mari având scopul de a pune în evidență detalii care s-ar observa greu dacă întregul model este afișat în fereastră. Totodată pentru modul de afișare sub forma unei perspective ortografice,

11.4 Afișarea sub forma unei perspective ortografice

Fișierul reprezentând un model 3D, pentru a afișa structura de bază și relațiile dintre elemente în sine într-un plan 2D am ales folosirea noțiunii de perspective ortografice. Modelul este desenat în 2D, având la bază coordonatele modelului 3D din fișierul .mdef, pentru fiecare perspectivă folosindu-se doua dintre cele trei coordonate. Astfel se obțin trei perspective individuale pentru afișarea modelului:

- perspectiva laterala (side)
- perspectiva frontala (front)
- perspectiva de deasupra (top)

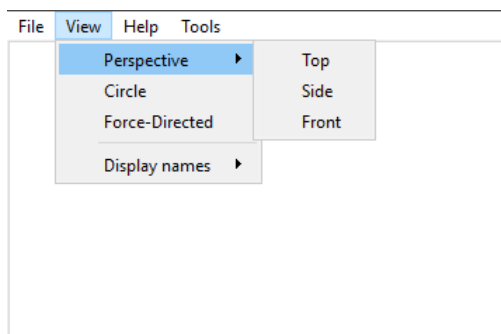


Fig. 15: Meniul view

Când un model este încărcat el este deschis automat în perspectiva laterală. Formele elementelor din această opțiune au la bază coordonate reale citite din fișier. De exemplu mărimea dreptunghiului care reprezintă un Motionbody este calculată în funcție de pozițiile conexiunilor cu alte elemente și centrul de greutate. Aceste perspective pot fi alese de utilizator selectând Perspective din meniul View.

Această abordare este folositoare, din perspectiva utilizatorului, doar pentru modele cu un număr mic spre mediu de elemente deoarece pentru modele mai mari s-a observat că datorită structurii lor, în multe cazuri existau suprapuneri în oricare dintre cele trei perspective, făcând înțelegerea modelului mult

mai grea.

11.5 Afișarea sub formă de graf

Pentru cazuri în care apar multe suprapuneri s-a implementat o structură de graf pentru reprezentarea modelului. În comparație cu perspectiva ortografică nu se ține cont de date precum puncte de conexiune sau centru de greutate și se bazează strict pe relațiile dintre elemente. Folosind un algoritm force-directed de desenare a unui graf obținem o structură mult mai organizată și mai inteligibilă, înșă renunțăm la concepte legate de forma modelului sau pozițiile elementelor în model.

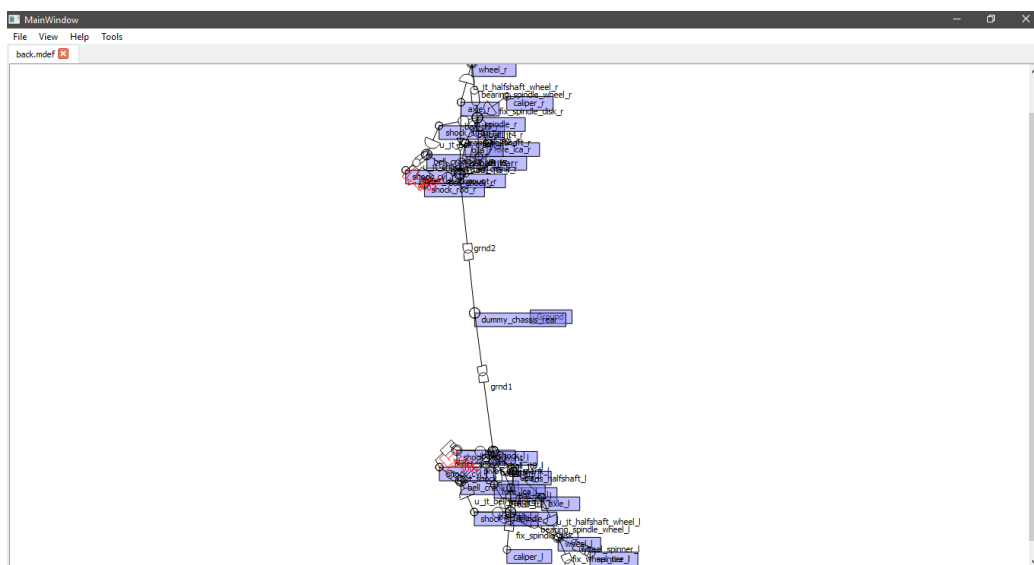


Fig. 16: Modelul menționat sub structura de graf

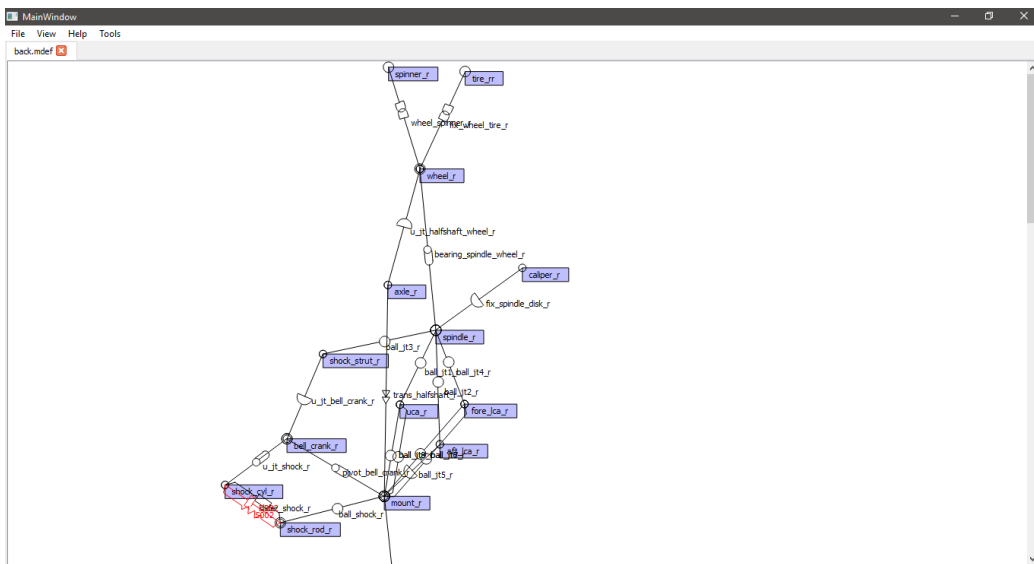


Fig. 17: Același graf scalat

Această funcționalitate a fost adăugată având în gând un concept prezent în algoritmi force-directed și anume cel care garantează ca dacă un graf este planar, desenarea lui nu va conține suprapuneri între muchii, și cel mai important, între noduri. Totodată și pentru grafuri care nu sunt planare desenarea lui conține un număr minim de suprapuneri între muchii, de multe ori neglijabil, modelul fiind în continuare ușor de înțeles.

Opțiunea de graf poate fi aleasă alegând Force-directed din meniul View. În această perspectivă se pot evidenția și tipurile de elemente Joint mai bine deoarece nu mai sunt reprezentate ca un punct fix dintre doua elemente Motion body ci dintr-o muchie cu un simbol pentru tipul elementului. Totodată din meniul de view se poate alege dacă este necesară afișarea fiecărui nume ale elementelor, acest lucru se face din meniul View alegând Show names, de unde se poate deselecta câte o căsuță pentru fiecare tip de element.

11.6 Căutarea elementelor

În continuare, pentru înțelegerea mai buna a modelului au fost implementate niște funcționalități de ajutor. O funcție de căutare pune în evidență

anumite elemente din model. Se poate accesa din meniul **Tools** de unde este deschisa o nouă fereastră cu mai multe opțiuni de cautare. Căutarea după un sub-string al cuvântului cheie sau căutarea după întreg cuvântul. Este prezentă și o opțiune de regex prin care utilizatorul poate cauta elemente după un anumit șablon. Din aceeași fereastră se poate alege dacă vrem să cautăm doar după anumite tipuri de elemente, precum și culoarea de selectare. După ce apăsăm **search**, dacă căutarea a avut succes elementele sunt evidențiate în scena cu o culoare aleasă. Utilizatorul poate efectua mai multe căutări cu diferite culori, lucru folositor pentru organizarea elementelor.

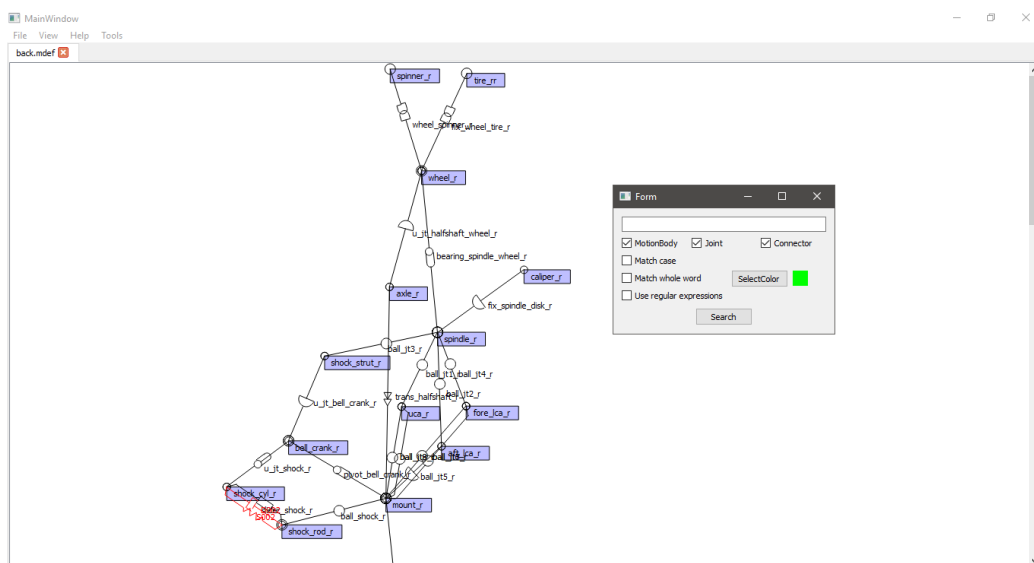


Fig. 18: Fereastra de cautare

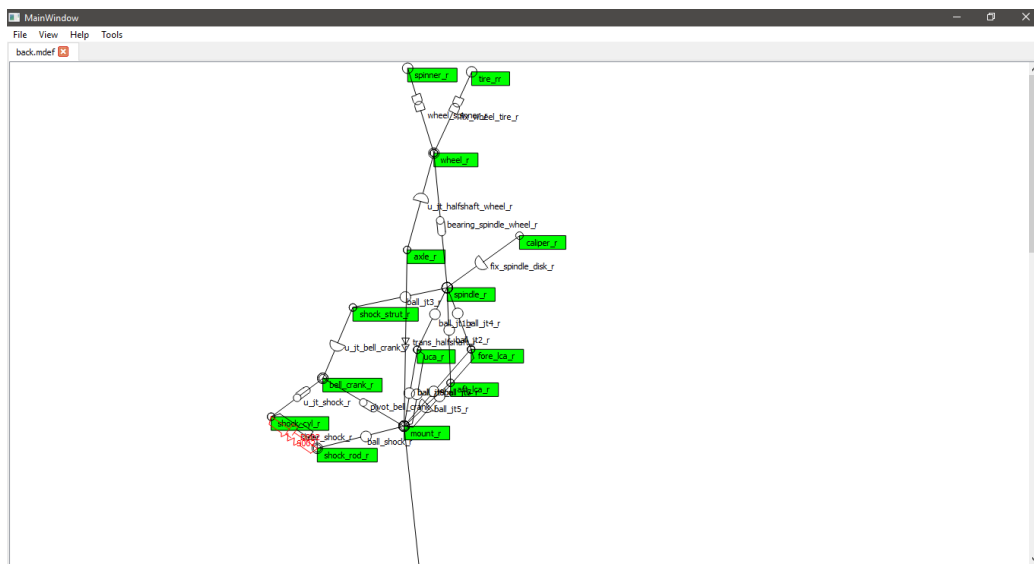


Fig. 19: Elementele care se afla pe partea dreapta a modelului, evidențiate prin funcția de căutare

11.7 Compararea modelelor

Pentru evidențierea diferențelor dintre versiuni diferite de fișiere .mdef a fost implementată funcționalitatea `Compare models`. Se poate accesa din meniul `Tools`, opțiunea `Compare models` și va deschide o nouă fereastră de unde utilizatorul poate alege ce modele vrea sa compare dintre cele deschise. Diferențe precum date diferite sunt evidențiate cu galben, iar lipsa de elemente este evidențiată prin colorarea rosu, în modelul care conține elementul lipsă din modelul de comparare.

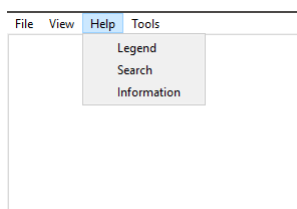


Fig. 22: Meniul Help

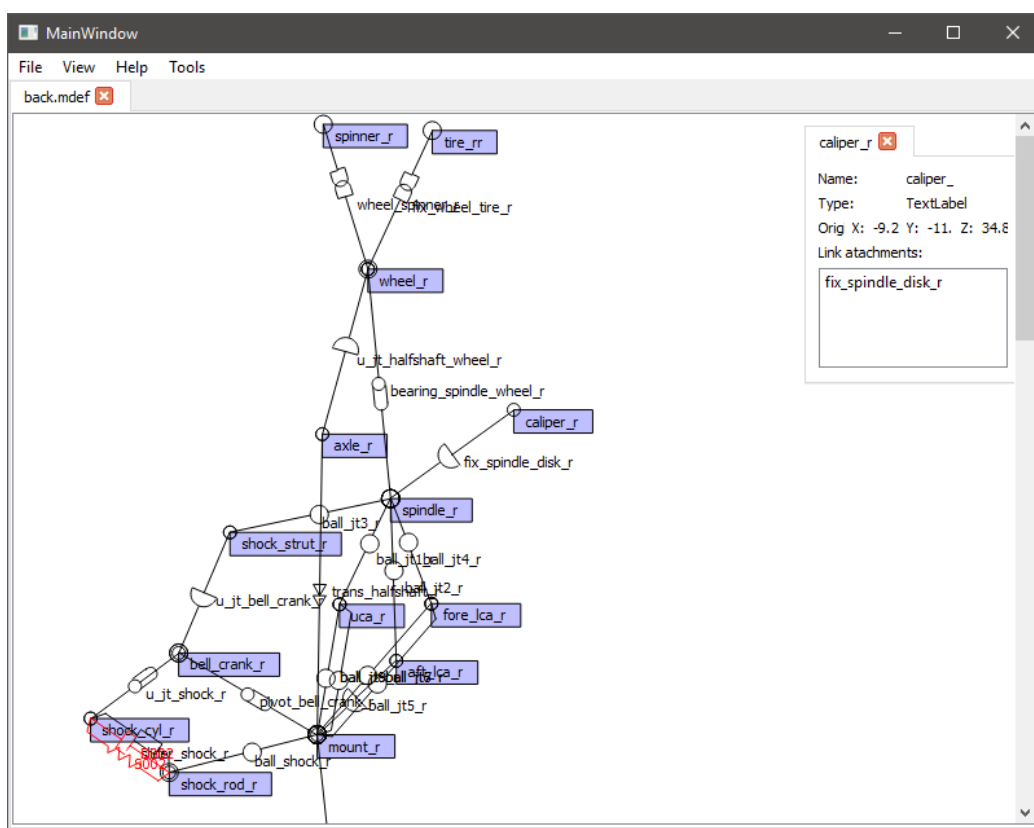


Fig. 23: Panoul cu informații despre un element Motion body

11.9 Legenda

O alta opțiune de ajutor este cea de legenda. Din help, legend se deschide un nou panou care afișează toate simbolurile de elemente Joint și ele-

mente Connector pentru ca utilizatorul sa înțeleagă mai bine structura modelului.

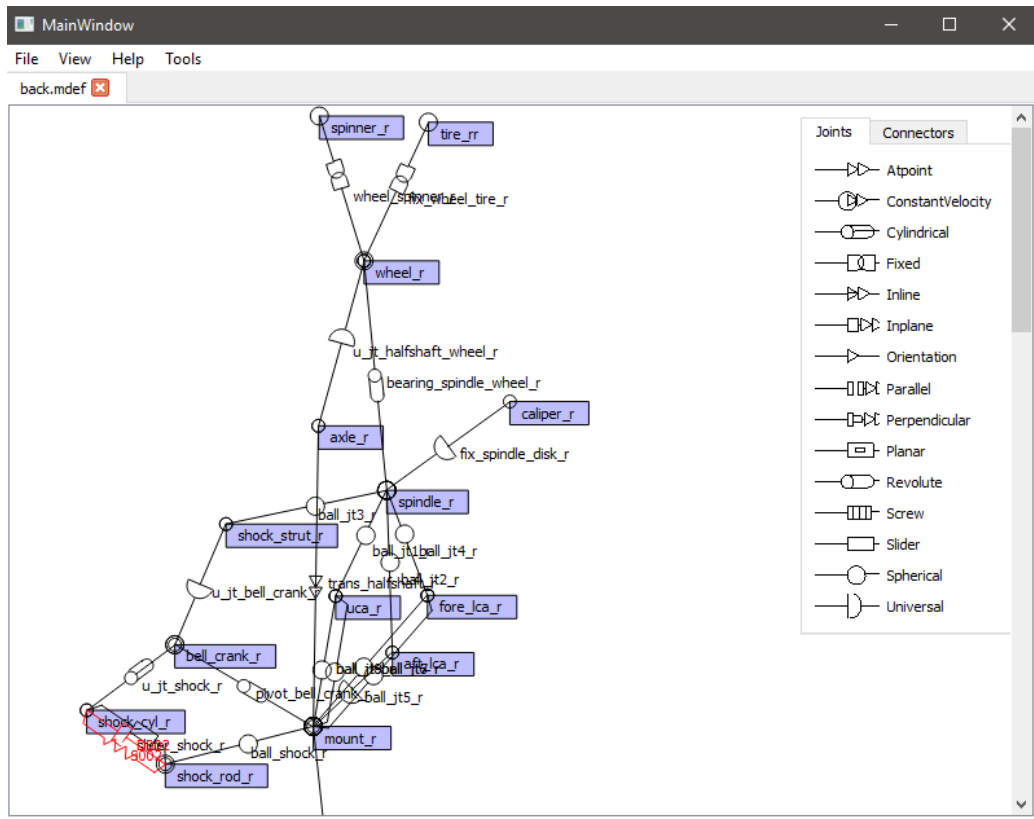


Fig. 24: Legenda

12 Anexa

12.1 XML

12.1.1 Motion body

Exemplu simplificat de element care reprezintă un corp de mișcare în fișierul .mdef:

```
<MotionBody Name="m1">
  <MassAndInertia>
    <TransformationMatrix>
      <Origin>
        <X Unit="in" Value="-7.58042829409448870592e+00"/>
        <Y Unit="in" Value="4.23044650011811071977e+01"/>
        <Z Unit="in" Value="3.44301163838582695575e+01"/>
      </Origin>
    </TransformationMatrix>
  </MassAndInertia>
</MotionBody>
```

12.1.2 Joint

Exemplu simplificat de element care reprezintă o legătură între două corpuri de mișcare:

```
<Joint Name="u_jt_shock_l" Type="Cylindrical">
  <Action>
    <SelectedMotionBody Name="shock_cyl_l"/>
    <TransformationMatrix>
      <Origin>
        <X Unit="in" Value="-4.14178975641839031141e+00"/>
        <Y Unit="in" Value="2.03613060274620565338e+01"/>
        <Z Unit="in" Value="4.14651875259415945152e+01"/>
      </Origin>
    </TransformationMatrix>
  </Action>
  <Base>
    <SelectedMotionBody Name="bell_crank_l"/>
  </Base>
</Joint>
```

```

    <TransformationMatrix>
      <Origin>
        <X Unit="in" Value="-4.14178975641839031141e+00"/>
        <Y Unit="in" Value="2.03613060274620565338e+01"/>
        <Z Unit="in" Value="4.14651875259415945152e+01"/>
      </Origin>
    </TransformationMatrix>
    <SnapMotionBodies Value="false"/>
  </Base>
  <DisplayScale Value="1.00000000000000000000e+00"/>
</Joint>

```

12.1.3 Connector

Exemplu de element Connector de tip arc (spring) dintre doua corpuri de miscare:

```

<Spring Name="S001" Type="MotionBody">
  <Action>
    <SelectedMotionBody Name="shock_rod_l"/>
    <TransformationMatrix>
      <Origin>
        <X Unit="in" Value="4.97282173372801139521e+00"/>
        <Y Unit="in" Value="2.22172095074280626648e+01"/>
        <Z Unit="in" Value="3.98239507981948648307e+01"/>
      </Origin>
    </TransformationMatrix>
  </Action>
  <Base>
    <SelectedMotionBody Name="shock_cyl_l"/>
    <TransformationMatrix>
      <Origin>
        <X Unit="in" Value="1.13657023776269738846e-01"/>
        <Y Unit="in" Value="2.10778191616462606817e+01"/>
        <Z Unit="in" Value="4.04163897920995651702e+01"/>
      </Origin>
    </TransformationMatrix>
  </Base>

```

</Spring>

Tag-urile Action si Base de la elementul Joint și Connector arată corpurile de miscare legate.

Toate elementele au o legătura strânsă iar buna funcționalitate a aplicației tine de structura fișierului.

12.2 Turtle graphics

12.2.1 Desenarea unui cilindru

```
void JointPainterPathCreator::drawRevolutePath(  
    Turtle & turtle ,  
    double length  
    ) const  
  
    const double radius = 5;  
  
    turtle.forward((length / 2) - radius);  
  
    turtle.save();  
  
    //drawing first circle  
    turtle.rotate(M_PI / 2);  
    const int nrOfSegments = 32;  
    const double angle = 2 * M_PI / nrOfSegments;  
    const double segmentLength = radius*sqrt(2.0 - 2.0*cos(angle));  
  
    for (int i = 0; i < nrOfSegments; i++) {  
        turtle.forward(segmentLength);  
        turtle.rotate(-angle);  
    }  
  
    turtle.load();  
    turtle.forward(2 * radius , false);  
  
    //drawing cilinder upper line  
    const double cilinderLength=10;
```

```

    turtle.forward(cilinderLength , false);

    turtle.rotate(-M_PI / 2);
    turtle.forward(radius , false);

    turtle.rotate(-M_PI / 2);
    turtle.forward(cilinderLength + 3);
    turtle.rotate(M_PI);
    turtle.forward(cilinderLength + 3, false);

    turtle.save();
    for (int i = 0; i < nrOfSegments / 2; i++) {
        turtle.forward(segmentLength);
        turtle.rotate(angle);
    }

    turtle.load();
    turtle.rotate(M_PI / 2);
    turtle.forward(radius*2.0, false);
    turtle.rotate(M_PI / 2);

    //drawing cilinder lower line
    turtle.forward(cilinderLength + 3);
    turtle.rotate(M_PI);
    turtle.forward(cilinderLength + 3, false);
    turtle.rotate(-M_PI);

    turtle.rotate(M_PI / 2);
    turtle.forward(radius , false);

    turtle.rotate(M_PI / 2);
    turtle.forward(radius , false);
}
}

```

13 Bibliografie

Referințe

- [1] Wikipedia contributors. Topology. Wikipedia, The Free Encyclopedia, 2019.
<https://en.wikipedia.org/wiki/Topology>
- [2] Bidyottama Koirala. DEVELOPING AN XML-BASED APPLICATION. Turun Ammattikorkeakoulu, Turku University Of Applied Sciences, 2014.
- [3] Wikipedia contributors. XML. Wikipedia, The Free Encyclopedia, 2019.
<https://en.wikipedia.org/wiki/XML>
- [4] Dr. Akhilesh Kumar Maurya. ORTHOGRAPHIC PROJECTIONS. Indian Institute Of Technology Guwahati, 2016
- [5] Wikipedia contributors. Orthographic projection. Wikipedia, The Free Encyclopedia, 2019.
https://en.wikipedia.org/wiki/Orthographic_projection
- [6] Matthias Graf. Direct Manipulation of Turtle Graphics. Otto-von-Guericke University Magdeburg, 2014
- [7] Wikipedia contributors. Turtle graphics. Wikipedia, The Free Encyclopedia, 2019.
https://en.wikipedia.org/wiki/Turtle_graphics
- [8] J. A. Bondy, U. S. R. Murty. GRAPH THEORY WITH APPLICATIONS. Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada, 1976
- [9] Wikipedia contributors. Graph theory. Wikipedia, The Free Encyclopedia, 2019.
https://en.wikipedia.org/wiki/Graph_theory
- [10] Wikipedia contributors. Multigraph. Wikipedia, The Free Encyclopedia, 2019.
<https://en.wikipedia.org/wiki/Multigraph>
- [11] Slavic Sorin Grafuri Planare Facultatea de Matematică și Informatică, Universitatea din București, 2010

- [12] Stephen G. Kobourov Force-Directed Drawing Algorithms University of Arizona.
- [13] Wikipedia contributors. Force-directed graph drawing. Wikipedia, The Free Encyclopedia, 2019.
https://en.wikipedia.org/wiki/Force-directed_graph_drawing
- [14] Qt Company. Graphics View Framework
<https://doc.qt.io/archives/qt-4.8/graphicsview.html>
- [15] Wikipedia contributors. C++. Wikipedia, The Free Encyclopedia, 2019.
<https://en.wikipedia.org/wiki/C%2B%2B>
- [16] Wikipedia contributors. Qt (software). Wikipedia, The Free Encyclopedia, 2019.
[https://en.wikipedia.org/wiki/Qt_\(software\)](https://en.wikipedia.org/wiki/Qt_(software))
- [17] Alexander Kolbasin, Oksana Husu. Computer-aided design and Computer-aided engineering. Moscow State University of Civil Engineering, 2018
- [18] Motion Simulation, Student guide, SC12. Siemens PLM Software, 2018