VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta strojního inženýrství

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Jakub Karas

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MATEMATIKY
INSTITUTE OF MATHEMATICS

ČÁSTICOVÝ SYSTÉM GARTICLE ENGINE
GARTICLE ENGINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jakub Karas**
AUTHOR

VEDOUCÍ PRÁCE — **Mgr. Aleš Návrat, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav matematiky
Student:	**Jakub Karas**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Matematické inženýrství
Vedoucí práce:	**Mgr. Aleš Návrat, Ph.D.**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Částicový systém Garticle engine

Stručná charakteristika problematiky úkolu:

Úkolem tohoto projektu na pomezí matematiky a informatiky je navrhnout a implementovat částicový systém na bázi geometrických algeber. Využitím PGA (projektivní geometrická algebra), což je moderní bezsouřadnicový jazyk pro geometrické počítání, se zjednoduší složité geometrické operace. Versory této algebry (nahrazující transformační matice) jsou izomorfní duálním kvaternionům a tzv. bivektory přímo reprezentují generátory Lieovy grupy SE(3), grupy všech translací a rotací v 3D.
Použití PGA versorů (8 typů float) oproti maticím (16 typů float) znamená zdvojnásobení částic, které se vejdou do jedné stopy. Stabilita integrování v Lieově grupě zase znamená, že tato implementace umožní delší časové kroky v simulaci a poskytne tak podstatný extra výkon bez ztráty přesnosti. PGA zdrojové kódy jsou k dispozici pro c++, c#, python, rust a javascript.

Cíle bakalářské práce:

1. Pochopení reprezentace rotací pomocí kvaternionů a translací pomocí duálních kvaternionů.
2. Nastudování PGA, zejména reprezentace transformací v PGA.
3. Implementace vlastního částicového systému využívajícího PGA v jakémkoli programovacím jazyce.

Seznam doporučené literatury:

GUNN, Charles G. Course notes Geometric Algebra for Computer Graphics, SIGGRAPH, 2019. Dostupný z: https://bivector.net/PROJECTIVE_GEOMETRIC_ALGEBRA.pdf

DORST, L., FONTIJNE, D., MANN, S. Geometric Algebra for Computer Science: An

Object-Oriented Approach to Geometry. 1st edn. Morgan Kaufmann Publishers Inc., 2007.

ABSTRAKT

V této práci je vytvořen funkční částicový systém, který na rozdíl od klasických implementací částicových systémů využívá k výpočtům moderní bezsouřadnicový jazyk – projektivní geometrickou algebru (PGA). Využití tohoto jazyka umožňuje efektivně nahradit body v částicovém systému tuhými tělesy, snížit paměťové nároky na počítač a v ideálních případech i urychlit výpočet. V teoretické části této práce je představena projektivní geometrická algebra a popsán způsob, jak v ní reprezentovat Euklidovské transformace a zformulovat rovnice pohybu tuhého tělesa, které tvoří základ výpočetní části systému.

KLÍČOVÁ SLOVA

Euklidovské transformace, kvaterniony, duální kvaterniony, projektivní geometrická algebra, pohyb tuhého tělesa, částicový systém

ABSTRACT

The main goal of this thesis is creation of a particle engine. Unlike classical implementations of particle engines this one uses a modern coordinate-free language – Projective Geometric Algebra (PGA). PGA allows us to replace points in the engine with rigid bodies. Furthermore usage of geometric algebra could reduce both space complexity and computational complexity. In theoretical part of the thesis is presented PGA, a representation of Euclidean transformations in PGA and formulation of equations of rigid body motion in PGA which are basis of the computational part of the engine.

KEYWORDS

Rigid transformations, Euclidean transformations, quaternions, dual quaternions, Projective Geometric Algebra, Rigid body motion, Particle system, Particle engine

KARAS, Jakub. *Částicový systém Garticle engine*. Brno, 2021, 59 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav matematiky. Vedoucí práce: Mgr. Aleš Návrat, PhD.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Částicový systém Garticle engine“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Mgr. Aleši Návratovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod

Hlavní náplní této práce je vytvoření tzv. *částicového systému*. To je nástroj pro sledování pohybu jakýchsi elementárních prvků (běžně bodů) v prostoru se silovým působením. Můj systém využívá k popisu pohybu místo klasické maticové reprezentace reprezentaci v projektivní geometrické algebře (PGA). Ta umožňuje tento částicový systém zdokonalit, ve smyslu užití těles s momentem setrvačnosti a hmotností jako částic.

Nejprve ukážeme reprezentaci rotací pomocí kvaternionů a Euklidovských transformací užitím duálních kvaternionů, protože ty jsou izomorfní motorům, které reprezentují transformace v PGA. K tomu zmíním komplexní a duální čísla, protože kvaterniony a duální kvaterniony jsou jejich zobecněním do prostoru.

Dále se budu věnovat reprezentaci v samotné projektivní geometrické algebře. Uvedu definice základních operací a reprezentaci objektů, které PGA umožňuje. Nakonec přidám popis transformací pomocí PGA.

Ve třetí kapitole popisuji pohyb tuhého tělesa. Uvádíme klasickou notaci, stejně jako notaci v PGA, a ukážeme, že rovnice pohybu tuhého tělesa v reprezentaci PGA jsou ekvivalentní zápisu, který známe z lekcí fyziky.

Následně uvádím pár slov k částicovým systémům jako takovým a v poslední kapitole přikládám dokumentaci k mému programu a výsledky, které jsem získal.

1 Různé způsoby popisu rotací a translací

V literatuře se můžeme setkávat s popisem Euklidovských transformací pomocí typicky komplexních čísel pro popis rotací v rovině, duálních čísel pro popis translací po přímce, kvaternionů pro rotace v prostoru a duálních kvaternionů pro obecné rotace a translace v prostoru. Nyní se budu postupně věnovat každé z těchto algeber. V rámci této kapitoly budu více či méně parafrázovat [1].

1.1 Komplexní čísla

Definice 1. Nechť $a, b \in \mathbb{R}, \boldsymbol{i} \notin \mathbb{R}, \boldsymbol{i}^2 = -1$, pak číslo $z = a + \boldsymbol{i}\, b$ nazveme *komplexní číslo*, $\boldsymbol{i}$ nazýváme *komplexní jednotka*. Množinu všech komplexních čísel značíme $\mathbb{C}$.

Popis rotací v komplexní množině je známý fenomén. Pro další použití a také pro vytvoření spojitostí především s PGA je užitečný popis rotací pomocí násobení exponenciální funkcí. Další analogií s PGA je reprezentace vektoru $(a, b) \in \mathbb{R}^2$ prvkem algebry – zde komplexním číslem $a + \boldsymbol{i}b \in \mathbb{C}$.

Věta 1. *Nechť $\varphi \in \langle 0; 2\pi \rangle$, pak $e^{\boldsymbol{i}\,\varphi}$ vyjadřuje rotaci o úhel φ kolem počátku v následujícím smyslu. Komplexní číslo $z\, e^{\boldsymbol{i}\,\varphi} \in \mathbb{C}$ je obrazem rotace libovolného $z = a + \boldsymbol{i}b \in \mathbb{C}$ o tento úhel.*

Důkaz. Nejprve vyjádříme operátor $e^{\boldsymbol{i}\,\varphi}$ pomocí součtu mocninné řady a následně tuto mocninnou řadu upravíme a využijeme vlastností komplexní jednotky pro převod do známého goniometrického tvaru komplexního čísla:

$$e^{\boldsymbol{i}\,\varphi} = \sum_{n=0}^{\infty} \frac{(\boldsymbol{i}\,\varphi)^n}{n!} = \sum_{n=0}^{\infty} \frac{(-1)^n\,(\varphi)^{2n}}{(2n)!} + \sum_{n=0}^{\infty} \frac{(-1)^n\,\boldsymbol{i}\,(\varphi)^{2n+1}}{(2n+1)!} = \cos(\varphi) + \boldsymbol{i}\,\sin(\varphi).$$

Druhou část důkazu ukážeme přepsáním operátoru do získané formy a roznásobením:

$$\begin{aligned} z\, e^{\boldsymbol{i}\,\varphi} &= (a + \boldsymbol{i}\, b)\, e^{\boldsymbol{i}\,\varphi} = (a + \boldsymbol{i}\, b)\,(\cos(\varphi) + \boldsymbol{i}\,\sin(\varphi)) \\ &= (a\,\cos(\varphi) - b\,\sin(\varphi)) + \boldsymbol{i}\,(a\,\sin(\varphi) + b\,\cos(\varphi))\,. \end{aligned} \tag{1.1}$$

□

V této formě jde jasně vidět, že tento operátor je ve skutečnosti také pouze jen další komplexní číslo, jehož koeficienty jsou závislé na parametru φ. V získaném výsledku rotování libovolného bodu je možno rozeznat, co bychom dostali z násobení dvourozměrného vektoru maticí rotace. Je tedy vidět analogie s touto operací. Jako poslední by se dalo ukázat, že skládání rotací je postupné násobení exponenciál a tedy sčítání úhlů v argumentu, což je to, co bychom čekali.

1.2 Duální čísla

Definice 2. Nechť $a, b \in \mathbb{R}, \boldsymbol{\varepsilon} \notin \mathbb{R}, \boldsymbol{\varepsilon}^2 = 0$, pak číslo $z = a + \boldsymbol{\varepsilon} b$ nazveme *duální číslo*, $\boldsymbol{\varepsilon}$ nazýváme *duální jednotkou*. Množinu všech duálních čísel značíme $\mathbb{D}$.

Ze stejných důvodů, které jsem uvedl u komplexních čísel, by bylo dobré najít nějakou reprezentaci translace pomocí duálních čísel ve formě násobení exponenciální funkcí. Analogicky ke komplexním číslům ztotožníme číslo $b \in \mathbb{R}$ s duálním číslem $1 + \boldsymbol{\varepsilon} b \in \mathbb{D}$.

Věta 2. *Nechť $t \in \mathbb{R}$, pak $e^{\boldsymbol{\varepsilon} t}$ vyjadřuje translaci o t ve směru imaginární osy bodu $z = 1 + \boldsymbol{\varepsilon} b \in \mathbb{D}$ v následujícím smyslu. Bod $z\, e^{\boldsymbol{\varepsilon} t} \in \mathbb{D}$ v duální rovině, který získáme posunutím bodu z ve směru osy $\boldsymbol{\varepsilon}$ o vzdálenost t.*

Důkaz. Opět se využívá rozvoj do mocninné řady a následná úprava pomocí vlastností duální jednotky.

$$e^{\boldsymbol{\varepsilon} t} = \sum_{n=0}^{\infty} \frac{(\boldsymbol{\varepsilon} t)^n}{n!} = 1 + \boldsymbol{\varepsilon} t.$$

$$z e^{\boldsymbol{\varepsilon} t} = (1 + \boldsymbol{\varepsilon} b)(1 + \boldsymbol{\varepsilon} t) = 1 + \boldsymbol{\varepsilon}(b + t)$$

□

Co se týká skládání posunutí je opět jednoduché ukázat, že je ekvivalentní postupnému násobení exponenciál. Skutečně,

$$(1 + \boldsymbol{\varepsilon} t)(1 + \boldsymbol{\varepsilon} s) = 1 + \boldsymbol{\varepsilon}(t + s) + \boldsymbol{\varepsilon}^2 ts = 1 + \boldsymbol{\varepsilon}(s + t).$$

1.3 Kvaterniony

Definice 3. Nechť $a, b, c, d \in \mathbb{R}, \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k} \notin \mathbb{R}, \boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1, \boldsymbol{ij} = -\boldsymbol{ji}$, potom číslo $\boldsymbol{p} = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$ nazveme *kvaternion* a čísla $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ nazýváme *komplexní jednotky*. Množinu všech kvaternionů značíme $\mathbb{H}$.

Součinem na množině komplexních a duálních čísel bylo standardní komutativní násobení, jaké známe z reálných čísel. To již u kvaternionů nefunguje, komplexní jednotky jsou navzájem antikomutativní. To je v pořádku, jelikož jimi budeme chtít reprezentovat rotace v $\mathbb{R}^3$, které také nejsou komutativní.

Konjugovaným kvaternionem pak nazýváme $\bar{\boldsymbol{p}} = a - b\boldsymbol{i} - c\boldsymbol{j} - d\boldsymbol{k}$. Pro popis rotace se využívá norma kvaternionu $\|\boldsymbol{p}\| = \sqrt{\boldsymbol{p}\bar{\boldsymbol{p}}} = \sqrt{a^2 + b^2 + c^2 + d^2}$. V kvaternionech ztotožňujeme $\boldsymbol{n} = (b, c, d) \in \mathbb{R}^3$ s tzv. *ryzím kvaternionem* $\boldsymbol{n} = b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$. Kvaterniony, které splňují $\|\boldsymbol{p}\| = 1$, se nazývají *jednotkové*, lze je zapsat ve tvaru $\boldsymbol{p} = \cos(\alpha) + \boldsymbol{n}\sin(\alpha) = e^{\alpha \boldsymbol{n}}$, kde $\boldsymbol{n} \in \mathbb{R}^3$, $\|\boldsymbol{n}\| = 1$, $\alpha \in \langle -\pi, \pi \rangle$.

Věta 3. *Nechť* $\boldsymbol{p} = \cos(\frac{\alpha}{2}) + \boldsymbol{n}\sin(\frac{\alpha}{2})$ *je jednotkový kvaternion,* $\boldsymbol{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$*, pak* $\boldsymbol{x}' = \boldsymbol{p}\,\boldsymbol{x}\bar{\boldsymbol{p}}, \boldsymbol{x}' \in \mathbb{H}$ *je obraz rotace bodu* $\boldsymbol{x}$ *kolem osy se směrovým vektorem* $\boldsymbol{n}$ *o úhel* α*.*

Důkaz. Tato věta je zobecněním Věty 1 do prostoru $\mathbb{R}^3$. Lze ji dokázat opět přímým výpočtem, analogicky důkazu zmíněné Věty 1. Kompletní důkaz lze najít v [2]. □

Dalším, pro geometrické algebry typickým, je tzv. *sandwich součin*, tedy násobení prvku zleva a zprava, jako to vidíme právě zde. To je důsledkem antikomutativity komplexních jednotek. Kvaterniony je možné zapsat v exponenciálním tvaru jako $\|\boldsymbol{p}\|e^{\boldsymbol{n}\alpha}$. Vychází to právě z možnosti zápisu jednotkového kvaternionu pomocí sinu a kosinu, libovolný kvaternion pak dostaneme jako násobek nějakého jednotkového.

1.4 Duální kvaterniony

Duální kvaterniony jsou dalším logickým zobecněním. Pokud vezmeme v úvahu, že jsme „vybaveni“ duálními čísly, reprezentujícími translaci, a kvaterniony reprezentujícími rotace, ale pouze kolem os procházejících počátkem. Bylo by záhodno je zkombinovat, abychom získali algebru, která bude schopna reprezentovat jak translace v prostoru, tak rotace. A to rotace kolem libovolné osy. V rámci této podkapitoly také čerpám z [4].

Definice 4. Nechť $\boldsymbol{q}, \boldsymbol{q}_\varepsilon \in \mathbb{H}$, $a_d, b_d, c_d, d_d \in \mathbb{D}$, $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}, \boldsymbol{\varepsilon} \notin \mathbb{R}$, $\boldsymbol{\varepsilon}^2 = 0$, $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$, $\boldsymbol{ij} = -\boldsymbol{ji}$, potom $\boldsymbol{p} = \boldsymbol{q} + \boldsymbol{\varepsilon}\boldsymbol{q}_\varepsilon = a_d + b_d\boldsymbol{i} + c_d\boldsymbol{j} + d_d\boldsymbol{k}$ nazveme *duálním kvaternionem*, množinu všech duálních kvaternionů značíme $\mathbb{H}_D$.

Duální jednotka $\boldsymbol{\varepsilon}$ je komutativní s komplexními jednotkami. *Duálně konjugovaný duální kvaternion* definujeme jako $\boldsymbol{p}^* = \boldsymbol{q} - \boldsymbol{\varepsilon}\boldsymbol{q}_\varepsilon$. Ovšem budeme potřebovat i „ryze kvaternionovou“ konjugaci, tu značíme $\bar{\boldsymbol{p}} = a_d - b_d\boldsymbol{i} - c_d\boldsymbol{j} - d_d\boldsymbol{k}$. Zavádíme normu duálního kvaternionu

$$\|\boldsymbol{p}\| = \sqrt{\boldsymbol{p}\bar{\boldsymbol{p}}} = \sqrt{(a + \boldsymbol{\varepsilon}a_\varepsilon)^2 + (b + \boldsymbol{\varepsilon}b_\varepsilon)^2 + (c + \boldsymbol{\varepsilon}c_\varepsilon)^2 + (d + \varepsilon d_\varepsilon)^2}.$$

Podobně jako v případě kvaternionů, pokud nebude řečeno jinak, s prostorovým vektorem $\boldsymbol{n} = (n_1, n_2, n_3)$ ztotožňujeme také kvaternion $\boldsymbol{n} = n_1\,\boldsymbol{i} + n_2\,\boldsymbol{j} + n_3\,\boldsymbol{k}$. Oproti kvaternionům se změní reprezentace bodů. Nyní bod s vektorem souřadnic $\boldsymbol{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$ reprezentujeme jako $\boldsymbol{X} = 1 + \boldsymbol{\varepsilon}\boldsymbol{x} = 1 + \boldsymbol{\varepsilon}(x_1\,\boldsymbol{i} + x_2\,\boldsymbol{j} + x_3\,\boldsymbol{k})$.

Popis translací je přímočarý, jde o přímé zobecnění duálních čísel. *Translátor* $\boldsymbol{T}$ vyjádříme jako:

$$\boldsymbol{T} = 1 + \boldsymbol{\varepsilon}\frac{\boldsymbol{t}}{2} = \exp\left(\boldsymbol{\varepsilon}\frac{\boldsymbol{t}}{2}\right),$$

kde vektor $\boldsymbol{t}$ je vektor posuvu. Rotorem je tzv. *jednotkový kvaternion*, totiž jednotkový duální kvaternion s nulovou duální částí

$$\boldsymbol{R} = \cos(\frac{\alpha}{2}) + \boldsymbol{n}\sin(\frac{\alpha}{2}) = \exp\left(\alpha\frac{\boldsymbol{n}}{2}\right).$$

Tedy kvaterniony, jak je známe z Podsekce 1.3. Rotovat tedy umíme stále pouze kolem os procházejících počátkem. Pro rotaci kolem osy, která počátkem neprochází, použijeme složení s translací. Nejprve přeneseme soustavu bodu a osy tak, aby osa procházela počátkem, následně provedeme požadovanou rotaci, a poté vše přeneseme zpět.

Věta 4. *Nechť $\boldsymbol{T}$ je translátor, $\boldsymbol{R}$ je rotor, $\boldsymbol{X} = 1 + \boldsymbol{\varepsilon x}$ je reprezentace bodu v duálních kvaternionech. Potom*

a) $\boldsymbol{T}\,\boldsymbol{X}\,\overline{\boldsymbol{T}^*}$ je vyjádřením translace bodu $\boldsymbol{X}$ v prostoru,

b) $\boldsymbol{R}\,\boldsymbol{X}\,\overline{\boldsymbol{R}^*}$ je vyjádřením rotace bodu $\boldsymbol{X}$ v prostoru,

c) rotaci pomocí rotoru $\boldsymbol{R}$ kolem obecné osy, procházející bodem $\boldsymbol{A} = 1 + \boldsymbol{\varepsilon a}$, získáme jako $\boldsymbol{T}_A\,\boldsymbol{R}\overline{\boldsymbol{T}}_A$, kde $\boldsymbol{T}_A = 1 + \boldsymbol{\varepsilon}\frac{\boldsymbol{a}}{2}$.

Důkaz. a) Důkaz je analogický důkazu Věty 2 u duálních čísel. Nicméně není nijak zvlášť dlouhý, proto ho zde vypíši. Kombinaci duální konjugace a „ryze kvaternionové“ konjugace lze přepsat

$$\overline{\boldsymbol{T}^*} = 1 - \boldsymbol{\varepsilon}\frac{1}{2}\boldsymbol{\dot{t}} = 1 + \boldsymbol{\varepsilon}\frac{1}{2}\boldsymbol{t} = 1 + \frac{1}{2}(t_1\boldsymbol{i} + t_2\boldsymbol{j} + t_3\boldsymbol{k}) = \boldsymbol{T}.$$

Celý vztah z věty vyjádříme jako

$$\boldsymbol{T}(1+\boldsymbol{\varepsilon x})\boldsymbol{T} = (1+\boldsymbol{\varepsilon}\frac{\boldsymbol{t}}{2})(1+\boldsymbol{\varepsilon x})(1+\boldsymbol{\varepsilon}\frac{\boldsymbol{t}}{2}) = (1+\boldsymbol{\varepsilon}\frac{\boldsymbol{t}}{2}+\boldsymbol{\varepsilon x})(1+\boldsymbol{\varepsilon}\frac{\boldsymbol{t}}{2}) = 1+\boldsymbol{\varepsilon}(\boldsymbol{x}+\boldsymbol{t}).$$

b) Viz Větu 3.

c) Poslední část věty lze dokázat přímým výpočtem. Jak již bylo zmíněno v úvodu k této větě, jde o složení translace soustavy do počátku souřadného systému, následné provedení rotace a přenesení soustavy zpět. Pro kompletní důkaz mohu odkázat na [3].

□

Příklad 1. Vypočtěte, do jakého bodu se posune bod $X = [0, 0, 1]$, když provedeme rotaci kolem osy zadané směrovým vektorem $\boldsymbol{n} = (0, 1, 1)$ a bodem $A = [0, 1, 0]$ o úhel $\alpha = \pi$ a následně translaci, ve směru vektoru $\boldsymbol{t} = (1, 0, 0)$ o vzdálenost $d = 3$.

Řešení 1. Nejdříve přepíšeme bod X do reprezentace pomocí duálních kvaternionů a vyjádříme všechny operátory.

$$\boldsymbol{X} = 1 + \varepsilon\boldsymbol{k}, \boldsymbol{T}_A = 1 + \varepsilon\frac{\boldsymbol{j}}{2}, \boldsymbol{T} = 1 + \varepsilon\frac{3}{2}\boldsymbol{i}$$
$$\boldsymbol{R} = \cos\left(\frac{\pi}{2}\right) + \sin\left(\frac{\pi}{2}\right)\left(\frac{\sqrt{2}}{2}\boldsymbol{j} + \frac{\sqrt{2}}{2}\boldsymbol{k}\right) = \frac{\sqrt{2}}{2}(\boldsymbol{j} + \boldsymbol{k})$$

$$\boldsymbol{X}_1 = \overline{\boldsymbol{T}}_A\boldsymbol{X}\overline{\left(\overline{\boldsymbol{T}}_A\right)^*} = \overline{\boldsymbol{T}}_A\boldsymbol{X}\overline{\boldsymbol{T}}_A = \left(1 - \varepsilon\frac{\boldsymbol{j}}{2}\right)(1 + \varepsilon\boldsymbol{k})\left(1 - \varepsilon\frac{\boldsymbol{j}}{2}\right)$$
$$= \left(1 - \varepsilon\frac{\boldsymbol{j}}{2} + \varepsilon\boldsymbol{k}\right)\left(1 - \varepsilon\frac{\boldsymbol{j}}{2}\right) = 1 - \varepsilon(-\boldsymbol{j} + \boldsymbol{k})$$

$$\boldsymbol{X}_2 = \boldsymbol{R}\boldsymbol{X}_1\overline{\boldsymbol{R}^*} = \boldsymbol{R}\boldsymbol{X}_1\boldsymbol{R}^* = \left(\frac{\sqrt{2}}{2}\boldsymbol{j} + \frac{\sqrt{2}}{2}\boldsymbol{k}\right)(1 - \varepsilon(-\boldsymbol{j} + \boldsymbol{k}))\left(-\frac{\sqrt{2}}{2}\boldsymbol{j} - \frac{\sqrt{2}}{2}\boldsymbol{k}\right)$$
$$= \left[\frac{\sqrt{2}}{2}\boldsymbol{j} + \frac{\sqrt{2}}{2}\boldsymbol{k} + \varepsilon\left(\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}\boldsymbol{k}\boldsymbol{j} + \frac{\sqrt{2}}{2}\boldsymbol{j}\boldsymbol{k} - \frac{\sqrt{2}}{2}\right)\right]\left(-\frac{\sqrt{2}}{2}\boldsymbol{j} - \frac{\sqrt{2}}{2}\boldsymbol{k}\right)$$
$$= \left(\frac{\sqrt{2}}{2}\boldsymbol{j} + \frac{\sqrt{2}}{2}\boldsymbol{k} + \varepsilon\sqrt{2}\boldsymbol{j}\boldsymbol{k}\right)\left(-\frac{\sqrt{2}}{2}\boldsymbol{j} - \frac{\sqrt{2}}{2}\boldsymbol{k}\right)$$
$$= -\frac{1}{2}\boldsymbol{j}^2 - \frac{1}{2}\boldsymbol{k}\boldsymbol{j} - \frac{1}{2}\boldsymbol{j}\boldsymbol{k} + \frac{1}{2} + \varepsilon\left(-\boldsymbol{j}\boldsymbol{k}\boldsymbol{j} - \boldsymbol{j}\boldsymbol{k}^2\right)$$
$$= 1 + \varepsilon\left(\boldsymbol{j} - \boldsymbol{k}\right)$$

$$\boldsymbol{X}_3 = \boldsymbol{T}_A\boldsymbol{X}_2\overline{\boldsymbol{T}_A^*} = \boldsymbol{T}_A\boldsymbol{X}_2\boldsymbol{T}_A = \left(1 + \varepsilon\frac{\boldsymbol{j}}{2}\right)\left[1 + \varepsilon\left(\boldsymbol{j} - \boldsymbol{k}\right)\right]\left(1 + \varepsilon\frac{\boldsymbol{j}}{2}\right)$$
$$= \left[1 + \varepsilon\left(\frac{3}{2}\boldsymbol{j} - \boldsymbol{k}\right)\right]\left(1 + \varepsilon\frac{\boldsymbol{j}}{2}\right) = 1 + \varepsilon\left(2\boldsymbol{j} - \boldsymbol{k}\right)$$

$$\boldsymbol{X}' = \boldsymbol{T}\boldsymbol{X}_3\overline{\boldsymbol{T}^*} = \boldsymbol{T}\boldsymbol{X}_3\boldsymbol{T} = \left(1 + \varepsilon\frac{3}{2}\boldsymbol{i}\right)\left[1 + \varepsilon\left(2\boldsymbol{j} - \boldsymbol{k}\right)\right]\left(1 + \varepsilon\frac{3}{2}\boldsymbol{i}\right)$$
$$= \left[1 + \varepsilon\left(\frac{3}{2}\boldsymbol{i} + 2\boldsymbol{j} - \boldsymbol{k}\right)\right]\left(1 + \varepsilon\frac{3}{2}\boldsymbol{i}\right) = 1 + \varepsilon\left(3\boldsymbol{i} + 2\boldsymbol{j} - \boldsymbol{k}\right)$$

2 Projektivní geometrická algebra

Geometrické algebry využívají objektový přístup. To znamená, že prvky prostoru a transformace jsou reprezentovány přímo pomocí prvků této algebry. Geometrická algebra jsou duální kvaterniony spolu s reprezentacemi objektů. Podle typu objektů a samozřejmě dimenze prostoru, který uvažujeme, rozlišujeme jednotlivé typy geometrických algeber.

Pro modelování vektorového prostoru nám stačí samotný vektorový prostor. My ovšem potřebujeme reprezentovat další objekty, ty získáváme zvýšením dimenze vektorového prostoru, který geometrickou algebru generuje (viz níže). Pokud nebude řečeno jinak, při zmínění PGA bude myšlena PGA pro modelování 3D prostoru generovaná 4-rozměrným vektorovým prostorem. Dalším z běžně používaných typů je tzv. konformní geometrická algebra (CGA) s reprezentací kružnic, resp. sfér. Pro více informací k CGA je možno nahlédnout do [5].

Kompletní zavedení geometrických algeber jako takových spolu se všemi příslušnými operacemi lze najít v učebních textech opět například v [5]. Nejdříve okomentuji vytvoření gradovaného prostoru, přestože v něm budu zmiňovat některé operace, i když jejich přehled uvedu až posléze. A to proto, že tyto dvě oblasti jsou spolu úzce spojeny a považuji za přehlednější mít výčet operací kompletní v jedné podsekci.

2.1 Gradovaný prostor vnějších součinů

Gradovaným prostorem vnějších součinů (nebo také vnější algebrou) obecně rozumíme prostor generovaný vektorovým prostorem pomocí tzv. *vnějšího součinu*. To je asociativní, antisymetrická a bilineární operace (viz 2.2). Vektorový prostor generující vnější algebru budeme zde značit V, uvažujeme dimenzi $\dim V = n$ a jeho bázové vektory budeme značit $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n$. Vnějším součinem k různých bázových vektorů generujícího vektorového prostoru V získáme tzv. *k-blade* (říkáme také, že blade je *stupně* k, angl. grade). Vektorový prostor lineárních kombinací k-bladů označujeme $\bigwedge^k(V)$, je dimenze $\dim\left(\bigwedge^k(V)\right) = \binom{n}{k}$ a jeho prvky nazýváme *k-vektor*. Pro ně pak z antisymetrie plyne

$$\boldsymbol{e}_i \wedge \boldsymbol{e}_i = 0 \text{ a } \boldsymbol{e}_i \wedge \boldsymbol{e}_j = \boldsymbol{e}_{ij}, \text{ pro } i \neq j. \tag{2.1}$$

Zápis k-bladů se také pomocí bázových vektorů V běžně zkracuje

$$\boldsymbol{e}_1 \wedge \boldsymbol{e}_2 \wedge \cdots \wedge \boldsymbol{e}_k = \boldsymbol{e}_{12\cdots k}.$$

Existuje tedy blade maximálního řádu n, označujeme ho I a nazýváme *pseudoskalár*. Vnější algebrou pak myslíme

$$\bigwedge(V) = \bigwedge^0(V) + \bigwedge^1(V) + \cdots + \bigwedge^n(V),$$

její dimenze je $\dim(\bigwedge(V)) = 2^n$. Kde $\bigwedge^0(V)$ jsou skaláry, $\bigwedge^1(V)$ jsou vektory, ..., $\bigwedge^n(V)$ jsou pseudoskaláry. Z tohoto zápisu je vidět, proč se také označuje gradovaným prostorem. Pod pojmem *multivektor* rozumíme libovolnou lineární kombinaci k-vektorů.

Vektorovým prostorem generujícím PGA je vektorový prostor dimenze 4. Má tedy čtyři bázové ortogonální vektory, které značíme: $\boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$. Je zde tedy nutné uvažovat obecně k-vektory. Rozdíl je v tom, že například obecný bivektor se skládá ze součtů 2-bladů, ale nemusí být nutně možno ho zapsat pomocí vnějšího součinu dvou vektorů (např. bivektor $\boldsymbol{e}_0 \wedge \boldsymbol{e}_1 + \boldsymbol{e}_2 \wedge \boldsymbol{e}_3$). Analogicky to platí pro trivektory. Celý gradovaný prostor PGA je v Tabulce 2.1.

2.2 Operace v PGA

Nyní zavedu různé operace v PGA. U binárních operací nejdříve uvedu definici na vektorech a následně zobecnění pro obecné blady. U každé operace uvedu v případě odlišnosti notaci použitou v implementaci mého programu v závorce.

Vnější součin $\boldsymbol{a} \wedge \boldsymbol{b}$

Vnější součin (angl. *wedge*) vektorů $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in V, \beta \in \mathbb{R}$ je asociativní operace, která splňuje tyto tři vlastnosti:

1. $\boldsymbol{a} \wedge \boldsymbol{b} = -\boldsymbol{b} \wedge \boldsymbol{a}$,
2. $\boldsymbol{a} \wedge (\beta \boldsymbol{b}) = \beta(\boldsymbol{a} \wedge \boldsymbol{b})$,
3. $\boldsymbol{a} \wedge (\boldsymbol{b} + \boldsymbol{c}) = (\boldsymbol{a} \wedge \boldsymbol{b}) + (\boldsymbol{a} \wedge \boldsymbol{c})$.

Poslední dvě vlastnosti se nazývají linearita, dohromady s první vlastností (antisymetrií) máme bilinearitu. Vnější součin je, jak jsem již řekl, způsob přechodu do vyšší dimenze a wedge dvou vektorů splňuje vlastnost 2.1. Pokud bychom uvažovali obecný k-blade $\boldsymbol{e}_A$, kde A je množina indexů taková, že $|A| = k$, pak

$$\boldsymbol{e}_i \wedge \boldsymbol{e}_A = \begin{cases} \boldsymbol{e}_{iA}, & \text{jestliže } i \notin A \\ 0, & \text{jestliže } i \in A \end{cases}. \tag{2.2}$$

Zobecnění na vnější součin dvou obecných bladů plyne přímo z asociativity vnějšího součinu. Tedy pokud $\boldsymbol{e}_A$ a $\boldsymbol{e}_B$ sdílejí nějaký vektor, pak je roven nule, jinak provedeme vnější součin všech obsažených vektorů. Vlastnost 2.2 zaručí, že stupeň vytvořeného bladu nepřekročí dimenzi prostoru V. Geometrická interpretace vnějšího součinu v PGA je průnik.

Vnitřní součin $a \cdot b$ $(a|b)$

Vnitřní součin pro vektory, tj. prvky stupně 1 v PGA, je běžný euklidovský, jak ho známe z lineární algebry, řídí se následujícími pravidly:

$$\boldsymbol{e}_i \cdot \boldsymbol{e}_j = \begin{cases} 0, & \text{jestliže } i = j = 0, \text{ nebo } i \neq j \\ 1, & \text{jestliže } i = j \in \{1, 2, 3\} \end{cases}.$$

Je potřebné ovšem zmínit, že zde dochází ke kombinaci dvou skutečností. Vektory $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ tvoří ortonormální bázi a vektor $\boldsymbol{e}_0$ je nulový vektor kolmý k ostatním vektorům. Právě dle druhé mocniny bázových vektorů ve vnitřním součinu (později ukáži, že to platí i pro geometrický součin) se určuje typ geometrické algebry. Kupříkladu CGA má čtyři vektory, které jsou v druhé mocnině rovny 1, a jeden vektor, který se umocní na -1.

Zobecněním vnitřního součinu vektoru a bladu je tzv. *levá a pravá kontrakce*. Uvedu zde rekurentní vzorec pro obecný vektor $\boldsymbol{a}$ a blade $\boldsymbol{e}_A = \boldsymbol{e}_B \wedge \boldsymbol{e}_C$, ze kterého lze vyjádřit, ale detailní odvození uvádět nebudu. Nicméně jej lze najít např. v [5].

$$\boldsymbol{a} \cdot \boldsymbol{e}_A = \boldsymbol{a} \cdot (\boldsymbol{e}_B \wedge \boldsymbol{e}_C) = (\boldsymbol{a} \cdot \boldsymbol{e}_B) \wedge \boldsymbol{e}_C + (-1)^{|B|} \boldsymbol{e}_B \wedge (\boldsymbol{a} \cdot \boldsymbol{e}_C)$$

Zobecnění levé kontrakce na operaci na bladech $\boldsymbol{e}_A = \boldsymbol{e}_B \wedge \boldsymbol{e}_C$ a $\boldsymbol{e}_D$ uvedu opět v implicitní formě:

$$\boldsymbol{e}_A \cdot \boldsymbol{e}_D = (\boldsymbol{e}_B \wedge \boldsymbol{e}_C) \cdot \boldsymbol{e}_D = \boldsymbol{e}_B \cdot (\boldsymbol{e}_C \cdot \boldsymbol{e}_D).$$

Geometrický součin $a\,b$ $(a * b)$

Geometrický součin je označován jako fundamentální součin geometrické algebry. Pomocí geometrického součinu můžeme definovat všechny ostatní operace, ale i naopak geometrický součin můžeme pro vektory definovat pomocí vztahu

$$\boldsymbol{a}\,\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{b} + \boldsymbol{a} \wedge \boldsymbol{b}.$$

Výsledek je tedy ze své podstaty multivektor, protože skalární součin vektorů je skalár a vnější součin vektorů je bivektor. Protože platí $\boldsymbol{a} \wedge \boldsymbol{a} = 0$, musí platit rovnost $\boldsymbol{a}^2 = \boldsymbol{a} \cdot \boldsymbol{a}$. Pokud se setkáme v literatuře s mocninou obecného multivektoru, je toto mocnění myšleno pomocí geometrického součinu.

Při zobecňování geometrického součinu na obecné blady využijeme vlastností asociativity geometrického součinu a již odvozených operací. Pro vektory $\boldsymbol{a}, \boldsymbol{b}$ a blade $\boldsymbol{e}_B$ platí

$$\begin{aligned} \boldsymbol{a}\,\boldsymbol{e}_B &= \boldsymbol{a} \wedge \boldsymbol{e}_B + \boldsymbol{a} \cdot \boldsymbol{e}_B, \\ (\boldsymbol{a}\,\boldsymbol{b})\,\boldsymbol{e}_B &= \boldsymbol{a}\,(\boldsymbol{b}\,\boldsymbol{e}_B). \end{aligned}$$

Poincarého[1] dualita e_A^* ($!e_A$)

Poincarého dualita je unární lineární operace. Díky linearitě je možné ji definovat na bázových bladech vztahem $\boldsymbol{e}_A \wedge \boldsymbol{e}_A^* = \mathrm{I}$. To umožňuje identifikovat PGA*, kterým značíme duální vektorový prostor prostoru PGA, tedy všechna zobrazení z PGA do reálných čísel, s vlastním prostorem PGA pomocí bázových bladů $\boldsymbol{e}^A \leftrightarrow \boldsymbol{e}_A^*$, kde $\boldsymbol{e}^A$ je bázový blade PGA* daný tak, že $\boldsymbol{e}^A(\boldsymbol{e}_A) = 1$, jinak 0. Například pro bázový 2-blade platí $\boldsymbol{e}^{01} \leftrightarrow \boldsymbol{e}_{01}^* = \boldsymbol{e}_{23}$. Pro multivektor s koeficienty $a, \ldots, p$, je vyjádření Poincarého duality v Tabulce 2.1.

Regresivní součin $\boldsymbol{e}_A \vee \boldsymbol{e}_B$ ($\boldsymbol{e}_A \& \boldsymbol{e}_B$)

Regresivní součin definujeme vztahem $\boldsymbol{e}_A \vee \boldsymbol{e}_B = (\boldsymbol{e}_A^* \wedge \boldsymbol{e}_B^*)^*$, je to tedy duál vnějšího součinu duálů. Tuto definici nám umožňuje právě ztotožnění PGA* s PGA, díky kterému můžeme použít vnější součin na prvcích duálního prostoru. Jeho geometrickým významem je konstrukce objektů. Například regresivním součinem dvou bodů získáme v PGA přímku, která těmito body prochází. Regresivní součin, na rozdíl od vnějšího součinu, snižuje stupeň bladů, to také plyne z definice Poincarého duality, protože pokud roste stupeň prvku v PGA*, musí klesat stupeň prvku k němu duálnímu v PGA.

Reverze $\widetilde{\boldsymbol{e}}_A$ ($\sim \boldsymbol{e}_A$)

Uvažujme blade $\boldsymbol{e}_A = \boldsymbol{e}_1 \wedge \boldsymbol{e}_2 \wedge \cdots \wedge \boldsymbol{e}_{k-1} \wedge \boldsymbol{e}_k$ tvořený ortogonálními vektory $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_{k-1}, \boldsymbol{e}_k$. Pak jeho reverze $\widetilde{\boldsymbol{e}}_A$ vznikne převrácením pořadí vektorů ve vnějším součinu

$$\widetilde{\boldsymbol{e}}_A = \boldsymbol{e}_k \wedge \boldsymbol{e}_{k-1} \wedge \cdots \wedge \boldsymbol{e}_2 \wedge \boldsymbol{e}_1.$$

Protože vnější součin je antikomutativní, můžeme reverzi vyjádřit jako

$$\widetilde{\boldsymbol{e}}_A = (-1)^{\frac{1}{2}k(k-1)} \boldsymbol{e}_A.$$

Reverze bladů obecného multivektoru lze najít v Tabulce 2.1.

Stupňové projekce $\langle \mathbf{A} \rangle_k$

Stupňovou projekcí multivektoru $\langle \mathbf{A} \rangle_k$ je projekce $\mathbf{A}$ na $\bigwedge^k(V)$.

Shrnutím tedy PGA je 16D prostor pro modelování 3D prostoru. Běžné značení tohoto prostoru je $\mathbb{R}^*_{3,0,1}$, hvězdička značí, že reprezentace bodu je duální k homogenní reprezentaci vzhledem k Poincarého dualitě, viz níže. Část v dolním indexu dává informaci o druhých mocninách jednotlivých bázových vektorů v geometrickém

[1] Jules Henri Poincaré, 1854–1912

součinu (3 kladné, žádný záporný, 1 nulový). CGA pro 3D prostor se zase značí $\mathbb{R}^{4,1,0}$. Můžeme tedy vystavět celý gradovaný prostor, který zapsán v Tabulce 2.1. Protože jsme v duálním prostoru, vektory reprezentují roviny, bivektory přímky a trivektory body.

	Skaláry	Vektory				Bivektory						Trivektory				Ps.skaláry
	$\mathbf{1}$	$\boldsymbol{e}_0$	$\boldsymbol{e}_1$	$\boldsymbol{e}_2$	$\boldsymbol{e}_3$	$\boldsymbol{e}_{01}$	$\boldsymbol{e}_{02}$	$\boldsymbol{e}_{03}$	$\boldsymbol{e}_{12}$	$\boldsymbol{e}_{31}$	$\boldsymbol{e}_{23}$	$\boldsymbol{e}_{021}$	$\boldsymbol{e}_{013}$	$\boldsymbol{e}_{032}$	$\boldsymbol{e}_{123}$	$\boldsymbol{e}_{0123}$
$\boldsymbol{A}$	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
$\boldsymbol{A}^*$	p	o	n	m	l	k	j	i	h	g	f	e	d	c	b	a
$\tilde{\boldsymbol{A}}$	a	b	c	d	e	$-f$	$-g$	$-h$	$-i$	$-j$	$-k$	$-l$	$-m$	$-n$	$-o$	p
		Roviny				Přímky						Body				

Tab. 2.1: Struktura gradovaného prostoru a některé operace (převzato z [6])

2.3 Objekty v PGA

Na rozdíl od klasické notace, kde hlavním objektem je vektor, v PGA jím je bod. Bod A o souřadnicích $[x, y, z]$ je v PGA reprezentován trivektorem

$$\boldsymbol{A} = x\boldsymbol{e}_{032} + y\boldsymbol{e}_{013} + z\boldsymbol{e}_{021} + \boldsymbol{e}_{123}.$$

Lze si všimnout, že jde o duální reprezentaci k homogenní reprezentaci ($\boldsymbol{A}^* = \boldsymbol{e}_0 + x\boldsymbol{e}_1 + y\boldsymbol{e}_2 + z\boldsymbol{e}_3$). Reprezentaci dalších objektů, konkrétně v PGA se jedná o přímky a roviny, definujeme pomocí vztahu:

$$A \in X \iff \boldsymbol{A} \vee \boldsymbol{X} = 0, \text{ kde } A \text{ je bod a } X \text{ je přímka nebo rovina.} \tag{2.3}$$

Zde ztotožňujeme dvě různé reprezentace, na levé straně ekvivalence se nachází geometrické objekty, zatímco vpravo je reprezentace v PGA.

Pro účely následující věty se nyní budu věnovat Plückerovým souřadnicím (parafrázuji [9]). Přímku l lze definovat pomocí dvou vektorů $\boldsymbol{d}, \boldsymbol{m} \in \mathbb{R}^3$. Vektor $\boldsymbol{d}$ je směrovým vektorem přímky a vektor $\boldsymbol{m} = \boldsymbol{d} \times \boldsymbol{x}$ je tzv. momentový vektor, získáme ho vektorovým součinem směrového vektoru $\boldsymbol{d}$ a vektoru $\boldsymbol{x}$, který vede z počátku do libovolného bodu na přímce. Plückerovými souřadnicemi pak myslíme $(\boldsymbol{d} : \boldsymbol{m}) = (d_1 : d_2 : d_3 : m_1 : m_2 : m_3)$.

Věta 5. *Reprezentace přímek a rovin v* PGA *se řídí následujícími vztahy.*

a) *Přímku l, definovanou jejími Plückerovými souřadnicemi $(\boldsymbol{d} : \boldsymbol{m})$, reprezentujeme v* PGA *pomocí bivektoru $\boldsymbol{l} = m_1\boldsymbol{e}_{01} + m_2\boldsymbol{e}_{02} + m_3\boldsymbol{e}_{03} + d_3\boldsymbol{e}_{12} + d_2\boldsymbol{e}_{31} + d_1\boldsymbol{e}_{23}$.*

b) *Rovinu ρ, definovanou rovnicí $\rho : ax + by + cz + d = 0$, reprezentujeme v* PGA *pomocí vektoru $\boldsymbol{\rho} = d\boldsymbol{e}_0 + x\boldsymbol{e}_1 + y\boldsymbol{e}_2 + z\boldsymbol{e}_3$.*

Důkaz. Důkaz obou částí lze provést přímým výpočtem.

a) Uspořádání Plückerových souřadnic je ve vyjádření bivektoru přeházené, protože PGA je duálním prostorem. Přímým výpočtem

$$\begin{aligned}\boldsymbol{A} \vee \boldsymbol{l} =&(-z \cdot m_3 - y \cdot m_2 - x \cdot m_1)\boldsymbol{e}_0 + \\ &(+z \cdot d_2 - y \cdot d_3 + 1 \cdot m_1)\boldsymbol{e}_1 + \\ &(-z \cdot d_1 + x \cdot d_3 + 1 \cdot m_2)\boldsymbol{e}_2 + \\ &(+y \cdot d_1 - x \cdot d_2 + 1 \cdot m_3)\boldsymbol{e}_3\end{aligned}$$

získáme 4 rovnice a srovnáním s [9] zjistíme, že jsme došli ke stejnému výsledku.

b) Přímým výpočtem lze získat $\boldsymbol{A} \vee \boldsymbol{\rho} = zc + yb + xa + d$. Toto je rovnice roviny přesně, jak ji známe. Rovnice je opravdu rovna nule právě tehdy, když $A \in \rho$.

□

Jak jsem zmínil v Sekci 2.2, regresní součin slouží ke konstrukci objektů a vnější součin reprezentuje jejich průnik. Regresním součinem dvou různých bodů vznikne přímka, tři různé body dají rovinu a pomocí čtyř různých bodů získáme celý prostor. Naopak vnějším součinem dvou rovin s různými normálovými vektory je přímka. Další příklady lze najít v [5].

2.4 Transformace v PGA

Transformace v PGA zajišťují tzv. *motory*, ty získáme součinem rotoru a translátoru, které se na objekt $\boldsymbol{X}$ aplikují tzv. *sandwich součinem*. Nový objekt vyjádříme jako $\boldsymbol{X}_1 = \boldsymbol{m}\,\boldsymbol{X}\,\widetilde{\boldsymbol{m}}$. Motory jsou izomorfní duálním kvaternionům, transformace tedy musí fungovat stejným způsobem. Izomorfismus $\mathbb{H}_D \to$ PGA můžeme vypsat zobrazením bázových prvků:

$1 \mapsto 1$	$\boldsymbol{i} \mapsto \boldsymbol{e}_{23}$	$\boldsymbol{j} \mapsto \boldsymbol{e}_{31}$	$\boldsymbol{k} \mapsto \boldsymbol{e}_{12}$
$\boldsymbol{\varepsilon} \mapsto I$	$\boldsymbol{i\varepsilon} \mapsto \boldsymbol{e}_{01}$	$\boldsymbol{j\varepsilon} \mapsto \boldsymbol{e}_{02}$	$\boldsymbol{k\varepsilon} \mapsto \boldsymbol{e}_{03}$

Věta 6. *Translace o vzdálenost* d *ve směru jednotkového vektoru* $\boldsymbol{n} = (n_1, n_2, n_3)^T$ *vyjádříme translátorem* $\boldsymbol{t} = 1 + \frac{d}{2}(n_1\boldsymbol{e}_{01} + n_2\boldsymbol{e}_{02} + n_3\boldsymbol{e}_{03})$. *Rotaci o úhel* φ *kolem jednotkové osy, procházející počátkem, se směrovým vektorem* $\boldsymbol{n} = (n_1, n_2, n_3)^T$, *vyjádříme rotorem* $\boldsymbol{r} = cos(\frac{\varphi}{2}) + sin(\frac{\varphi}{2})(n_1\boldsymbol{e}_{23} + n_2\boldsymbol{e}_{31} + n_3\boldsymbol{e}_{12})$. *Rotory a translátory můžeme skládat pomocí geometrického součinu.*

Důkaz. Jak jsem již řekl, motory jsou izomorfní duálním kvaternionům. Důkaz je tedy analogický důkazu Věty 4. □

Příklad 2. Vypočtěte, do jakého bodu se posune bod $X = [0, 0, 1]$, když provedeme rotaci kolem osy zadané směrovým vektorem $\boldsymbol{n} = (0, 1, 1)$ a bodem $A = [0, 1, 0]$ o úhel $\alpha = \pi$, a následně translaci ve směru vektoru $\boldsymbol{t} = (1, 0, 0)$ o vzdálenost $d = 3$.

Řešení 2. Přepíšeme vše do reprezentace PGA:

$$\begin{aligned} \boldsymbol{X} &= \boldsymbol{e}_{021} + \boldsymbol{e}_{123} \\ \text{Rotor } \boldsymbol{r} &= -\frac{\sqrt{2}}{2}\boldsymbol{e}_{01} - \frac{\sqrt{2}}{2}\boldsymbol{e}_{12} + \frac{\sqrt{2}}{2}\boldsymbol{e}_{13} \\ \text{Translátor } \boldsymbol{t} &= 1 + \frac{3}{2}\boldsymbol{e}_{01} \end{aligned}$$

Za povšimnutí stojí, že rotor $\boldsymbol{r}$ není ve tvaru, jaký jsem uvedl ve větě 6. Obsahuje člen $\boldsymbol{e}_{01}$. To je důsledek toho, že jde o rotaci kolem osy neprocházející počátkem souřadného systému. Je to tedy složení rotace a translace.

Výsledný bod $\boldsymbol{X}_n$ pak vypočteme následovně:

$$\begin{aligned} \boldsymbol{X}_n &= \boldsymbol{t}\,\boldsymbol{r}\,X\,(\widetilde{\boldsymbol{r}})\,(\widetilde{\boldsymbol{t}}\,) \\ &= \boldsymbol{t}\,(\frac{\sqrt{2}}{2}\boldsymbol{e}_{01} + \frac{\sqrt{2}}{2}\boldsymbol{e}_{12} + \frac{\sqrt{2}}{2}\boldsymbol{e}_{31})\,(\boldsymbol{e}_{021} + \boldsymbol{e}_{123})\,(-\frac{\sqrt{2}}{2}\boldsymbol{e}_{01} - \frac{\sqrt{2}}{2}\boldsymbol{e}_{12} - \frac{\sqrt{2}}{2}\boldsymbol{e}_{31})\,(\widetilde{\boldsymbol{t}}\,) \\ &= (1 - \frac{3}{2}\boldsymbol{e}_{01})\,(2\boldsymbol{e}_{013} - \boldsymbol{e}_{021} + \boldsymbol{e}_{123})\,(1 + \frac{3}{2}\boldsymbol{e}_{01}) \\ &= 3\boldsymbol{e}_{032} + 2\boldsymbol{e}_{013} - \boldsymbol{e}_{021} + \boldsymbol{e}_{123}. \end{aligned}$$

Je to stejný výsledek, který vyšel v Příkladu 1 v Kapitole 1. Tento příklad lze jednoduše vyřešit v rámci mého programu, který popisuji v Kapitole 5. Spouštěcí kód a výsledek jsou na Obrázcích 2.1 a 2.2. Ve výsledku na Obr. 2.2 se objevují numerické chyby, které je potřeba zanedbávat.

```
var X = new PGAPoint(0, 0, 1);
var osaTranslace = new PGAPoint(1, 0, 0) & new PGAPoint(0, 0, 0);
var t = PGATranslator(3, osaTranslace.Normalized().Ideal());
var osaRotace = new PGAPoint(0, 2, 1) & new PGAPoint(0, 1, 0);
var r = PGARotor(MathF.PI, osaRotace.Normalized());
Console.WriteLine("X: " + X.ToPGA().ToString());
Console.WriteLine("t: " + t.ToPGA().ToString());
Console.WriteLine("r: " + r.ToPGA().ToString());
Console.WriteLine("r*X*(~r): " + X.Move(r).ToPGA().ToString());
Console.WriteLine("t*r*X*(~r)*(~t): " + X.Move(t * r).ToPGA().ToString());
```

Obr. 2.1: Spouštěcí kód Příkladu 2

```
C:\WINDOWS\system32\cmd.exe
X: 1e021 + 1e123
t: 1 + -1.5e01
r: -4.371139E-08 + 0.70710677e01 + 0.70710677e12 + 0.70710677e31
r*X*(~r): -1e021 + 2e013 + 1.2363449E-07e032 + 1e123
t*r*X*(~r)*(~t): -1e021 + 2e013 + 3.0000005e032 + 1e123
Press any key to continue . . .
```

Obr. 2.2: Výsledky Příkladu 2

3 Pohyb tuhého tělesa

Pohyb tuhého tělesa je téma velmi obsáhlé a ve fyzice dobře rozvedené a popsané pomocí maticově-vektorového zápisu. V řeči PGA neexistuje téměř žádný materiál. Jedním z mála je například [7]. Nicméně tento text jde svou precizností na úkor srozumitelnosti. Pokusím se tedy vybrat části důležité pro cíl mojí práce, kterým je částicový systém založený na PGA, a ukázat, že se výpočty v PGA shodují s poznatky z „učebnicové“ fyziky.

Jednou z výhod popisu pohybu pomocí PGA je fakt, že výpočet probíhá v souřadném systému pevně svázaným s tělesem (angl. body frame). Oproti nějakému obecnému souřadnému systému (angl. space frame). V označování veličin se budu držet konvence v Tabulce 3.1. Dále budu používat horní index s (space frame) a b (body frame) pro upřesnění souřadného systému, ve kterém se objekt uvažuje (např. $\mathbf{v}^s$).

Další výhodou je snížení paměťové náročnosti pro počítač. Matice polohy a rotace $G \in \mathbb{R}^{4\times4}$, zatímco motor v PGA $\mathbf{g} \in \mathbb{R}^8$. Tedy snížíme paměťové nároky na polovinu, a tím pádem zdvojnásobíme možný počet částic uvažovaných ve výpočtu. Při použití výpočetní techniky optimalizované pro PGA by měl být výpočet zároveň i rychlejší než použití maticově-vektorové notace, na kterou je dnešní hardware optimalizován.

Vlastnosti tělesa

Potřebujeme definovat hmotnost a inerciální tenzor tělesa, které vystupují v Newtonových – Eulerových dynamických rovnicích. Zde se budu držet klasické konvence, hmotnost budu značit m a inerciální tenzor

$$I = \begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{bmatrix}.$$

Například $I = m\frac{s^2}{6}\mathbb{I}$ je inerciální tenzor pro krychli o hmotnosti m a délce strany s ([17]). Při sjednocení lineárních a úhlových složek do jednoho objektu, jak je tomu například i v PGA, je vhodné využít symetrické zobrazení $A : \mathbb{R}^6 \to \mathbb{R}^6$.

$$A = \begin{pmatrix} m\mathbb{I} & \mathbb{0} \\ \mathbb{0} & I \end{pmatrix}$$

Kde $\mathbb{I} \in \mathbb{R}^{3\times3}$ je jednotková matice. Při takovém sjednocení je třeba uvažovat hybnosti, síly a momenty v duálním prostoru. V PGA popisu je A zobrazení z bivektorů do duálních bivektorů.

3.1 Klasický zápis

Pozici a orientaci částice v maticově-vektorové notaci reprezentujeme vektorem rotace $\boldsymbol{R}$ a vektorem translace $\boldsymbol{r}$. Tato transformace představuje transformaci z obecného souřadného systému do souřadného systému tělesa. Zde se přirozeně objevuje otázka, jestli bychom nebyli schopni sestrojit zobrazení mezi těmito dvěma systémy s pomocí této transformace. Toto zobrazení existuje, je jím tzv. *adjungovaná akce*. Její kompletní odvození lze najít v [10]. Často se u klasické reprezentace využívá zobrazení

$$\widehat{\ }: \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \mapsto \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}.$$

Vektorové veličiny jsou v běžném zápisu sloupcových vektorů z $\mathbb{R}^3$. Například $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^T$ je úhlová rychlost. Vztah mezi rychlostmi ve space framu a body framu určuje

$$\begin{pmatrix} \boldsymbol{v}^s \\ \boldsymbol{\omega}^s \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{R}} & \hat{\boldsymbol{r}}\widehat{\boldsymbol{R}} \\ \boldsymbol{0} & \widehat{\boldsymbol{R}} \end{pmatrix} \begin{pmatrix} \boldsymbol{v}^b \\ \boldsymbol{\omega}^b \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{R}}\boldsymbol{v}^b + \hat{\boldsymbol{r}}\widehat{\boldsymbol{R}}\boldsymbol{\omega}^b \\ \widehat{\boldsymbol{R}}\boldsymbol{\omega}^b \end{pmatrix}.$$

Sílu i točivý moment reprezentujeme opět vektory v $\mathbb{R}^3$. Protože se ovšem nachází v duálním prostoru, transformují se pomocí tzv. *koadjungované akce*. Tedy

$$\begin{pmatrix} \boldsymbol{F}^s \\ \boldsymbol{\tau}^s \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{R}}^T & \boldsymbol{0} \\ \widehat{\boldsymbol{R}}\hat{\boldsymbol{r}} & \widehat{\boldsymbol{R}}^T \end{pmatrix} \begin{pmatrix} \boldsymbol{F}^b \\ \boldsymbol{\tau}^b \end{pmatrix} = \begin{pmatrix} \widehat{\boldsymbol{R}}^T\boldsymbol{F}^b \\ \widehat{\boldsymbol{R}}\hat{\boldsymbol{r}}\boldsymbol{F}^b + \widehat{\boldsymbol{R}}^T\boldsymbol{\tau}^b \end{pmatrix}.$$

Rovnice popisující pohyb tuhého tělesa mají, s použitím klasického zápisu, tvar (viz [10]):

$$\dot{\boldsymbol{r}} = \widehat{\boldsymbol{R}}\boldsymbol{v}^b \tag{3.1}$$

$$\dot{\widehat{\boldsymbol{R}}} = \widehat{\boldsymbol{R}}\hat{\boldsymbol{\omega}}^b \tag{3.2}$$

$$\dot{\boldsymbol{v}}^b = \frac{\boldsymbol{F}^b}{m} - \hat{\boldsymbol{\omega}}^b\boldsymbol{v}^b \tag{3.3}$$

$$\dot{\boldsymbol{\omega}}^b = I^{-1}(\boldsymbol{\tau}^b - \hat{\boldsymbol{\omega}}^b I\boldsymbol{\omega}^b) \tag{3.4}$$

3.2 Zápis pomocí PGA

V zápisu, využívajícím PGA, je na první pohled vidět zjednodušení v sjednocení lineárních a úhlových částí objektů. Pozici a orientaci reprezentuje motor $\mathbf{g}$. Vyjadřuje vlastně přenos obecného souřadného systému do systému souřadnic vlastního tělesu. Pokud uvažujeme lineární a úhlovou rychlost ve složkách v_i, ω_i, je bivektor rychlosti tvaru

$$\mathbf{v} = \frac{1}{2}(v_1\boldsymbol{e}_{01} + v_2\boldsymbol{e}_{02} + v_3\boldsymbol{e}_{03} + \omega_3\boldsymbol{e}_{12} + \omega_2\boldsymbol{e}_{31} + \omega_1\boldsymbol{e}_{23}).$$

Transformaci mezi space a body framem vyjádříme jako $\mathbf{v}^s = \mathbf{g}\,\mathbf{v}^b\,\widetilde{\mathbf{g}}$. Zde je vidět, jak se v PGA reprezentuje adjungovaná akce, kterou jsem zmínil výše.

Hybnost lze jednoduše získat jako

$$\boldsymbol{A}\,\mathbf{v} = 2(mv_3\boldsymbol{e}_{12} + mv_2\boldsymbol{e}_{31} + mv_1\boldsymbol{e}_{23} + \sum_{i=1}^{3} I_{1,i}\omega_i\boldsymbol{e}_{01} + \sum_{i=1}^{3} I_{2,i}\omega_i\boldsymbol{e}_{02} + \sum_{i=1}^{3} I_{3,i}\omega_i\boldsymbol{e}_{03}),$$

proto ji nebudu označovat dalším symbolem. Jak bylo zmíněno, uvažujeme hybnost v PGA*, ale díky Poincarého dualitě (viz Podsekce 2.2) ji můžeme ztotožnit s prvky PGA. Například blade $\frac{1}{2}\boldsymbol{e}_{01} \leftrightarrow 2\boldsymbol{e}^{01} = 2\boldsymbol{e}_{23}$. Koeficient $\frac{1}{2}$ se změní ve 2, aby se vzájemně vyrušily ve vnějším součinu v definici duality a 2-blady obsahující $\boldsymbol{e}_0$ nyní obsahují členy momentu hybnosti, zatímco u rychlosti obsahují lineární složku. Sílu budeme stejně jako hybnost uvažovat v duálním prostoru, což je jen logické, jelikož 2. Newtonův zákon říká, že síla je derivací hybnosti podle času. Bivektor silových působení vyjádříme jako

$$\mathbf{f} = 2(\tau_1\boldsymbol{e}_{01} + \tau_2\boldsymbol{e}_{02} + \tau_3\boldsymbol{e}_{03} + f_3\boldsymbol{e}_{12} + f_2\boldsymbol{e}_{31} + f_1\boldsymbol{e}_{23}).$$

Složky síly v příslušném směru jsou f_i, τ_i jsou složky točivého momentu v dané rovině. Opět vidíme, že točivý moment je reprezentován 2-blady obsahujícími $\boldsymbol{e}_0$, tedy 2-blady, ve kterých je v bivektoru rychlosti lineární složka. Dalším důsledkem toho, že uvažujeme sílu v PGA*, je, že transformace mezi space a body framem je tvaru $\mathbf{f}^s = \widetilde{\mathbf{g}}\,\mathbf{f}^b\,\mathbf{g}$. Poznamenám, že zde vidíme reprezentaci koadjungované akce v PGA.

Název	Klasický zápis	PGA
Pozice / orientace	$\boldsymbol{r}/\boldsymbol{R}$	$\mathbf{g}$
Rychlost (lineární / úhlová)	$\boldsymbol{v}$ / $\boldsymbol{\omega}$	$\mathbf{v}$
Silové účinky (lineární / točivé)	$\boldsymbol{F}$ / $\boldsymbol{\tau}$	$\mathbf{f}$

Tab. 3.1: Označení veličin ve výpočtech

3.3 Rovnice pohybu tuhého tělesa

Pohyb tuhého tělesa v PGA lze popsat dvěma obyčejnými diferenciálními rovnicemi 1. řádu. Infinitesimální verzi adjungované akce $\mathrm{ad}_{\mathbf{g}}\,\mathbf{v}$ nazýváme *komutátor*. Je definována vztahem

$$\mathbf{g} \times \mathbf{v} = \frac{1}{2}(\mathbf{g}\,\mathbf{v} - \mathbf{v}\,\mathbf{g}). \tag{3.5}$$

Jak je vidět v tomto vztahu, budu komutátor značit $\times$.

Věta 7. *Nechť* $\mathbf{g}$ *je motor přenášející z obecného souřadného systému do souřadného systému tělesa,* $\mathbf{v}^b$ *je rychlost tělesa,* A *je inerciální matice tělesa a* $\mathbf{f}^b$ *je silové působení na těleso. Pak rovnice*

$$\dot{\mathbf{g}} = \mathbf{g}\,\mathbf{v}^b, \tag{3.6}$$

$$\dot{\mathbf{v}}^b = A^{-1}\mathbf{f}^b - A^{-1}(\mathbf{v}^b \times A\,\mathbf{v}^b) \tag{3.7}$$

popisují pohyb tuhého tělesa v prostoru.

Důkaz. Důkaz lze najít v [7] a [10]. V [10] je ovšem rovnice 3.7 rozdělena do dvou – pro lineární a úhlové zrychlení tak, jak jsem to uvedl v předchozí části.

Rovnice 3.6 vychází přímo z definice rychlosti $\mathbf{v}^b = \widetilde{\mathbf{g}}\dot{\mathbf{g}}$. Ukáži, jak bychom převedli rovnici 3.7 na rovnice 3.3 a 3.4. Využiji k tomu Tabulku 3.2 níže. Vyjádření komutátoru v rovnici 3.7 vyjde jako:

$$\begin{aligned}
\mathbf{v}^b \times A\,\mathbf{v}^b = &(-\cancel{mv_2v_3} + \cancel{mv_2v_3} + \omega_3 \sum_{i=1}^{3} I_{2,i}\omega_i - \omega_2 \sum_{i=1}^{3} I_{3,i}\omega_i)\boldsymbol{e}_{01} + \\
&(\cancel{mv_1v_3} - \cancel{mv_1v_3} - \omega_3 \sum_{i=1}^{3} I_{1,i}\omega_i + \omega_1 \sum_{i=1}^{3} I_{3,i}\omega_i)\boldsymbol{e}_{02} + \\
&(-\cancel{mv_1v_2} + \cancel{mv_1v_2} + \omega_2 \sum_{i=1}^{3} I_{1,i}\omega_i - \omega_1 \sum_{i=1}^{3} I_{2,i}\omega_i)\boldsymbol{e}_{03} + \\
&m(\omega_2 v_1 - \omega_1 v_2)\boldsymbol{e}_{12} + \\
&m(-\omega_3 v_1 + \omega_1 v_3)\boldsymbol{e}_{31} + \\
&m(\omega_3 v_2 - \omega_2 v_3)\boldsymbol{e}_{23}.
\end{aligned}$$

Odsud je již vidět, že pokud vezmeme duální bivektor a vynásobíme ho inverzí k A, dostaneme přesně vztahy 3.3, 3.4. □

$\times$	$\boldsymbol{e}_{01}$	$\boldsymbol{e}_{02}$	$\boldsymbol{e}_{03}$	$\boldsymbol{e}_{12}$	$\boldsymbol{e}_{31}$	$\boldsymbol{e}_{23}$
$\boldsymbol{e}_{01}$	0	0	0	$\boldsymbol{e}_{02}$	$-\boldsymbol{e}_{03}$	0
$\boldsymbol{e}_{02}$	0	0	0	$-\boldsymbol{e}_{01}$	0	$\boldsymbol{e}_{03}$
$\boldsymbol{e}_{03}$	0	0	0	0	$\boldsymbol{e}_{01}$	$-\boldsymbol{e}_{02}$
$\boldsymbol{e}_{12}$	$-\boldsymbol{e}_{02}$	$\boldsymbol{e}_{01}$	0	0	$\boldsymbol{e}_{23}$	$-\boldsymbol{e}_{31}$
$\boldsymbol{e}_{31}$	$\boldsymbol{e}_{03}$	0	$-\boldsymbol{e}_{01}$	$-\boldsymbol{e}_{23}$	0	$\boldsymbol{e}_{12}$
$\boldsymbol{e}_{23}$	0	$-\boldsymbol{e}_{03}$	$\boldsymbol{e}_{02}$	$\boldsymbol{e}_{31}$	$-\boldsymbol{e}_{12}$	0

Tab. 3.2: Komutátor na 2-bladech

4 Částicový systém

Hlavní výsledkem této bakalářské práce je vytvoření tzv. „částicového systému“ (angl. *Particle engine/system*). Rozdíl proti drtivé většině používaných systémů (i komerčně využívaných) bude spočívat v implementaci geometrické algebry PGA pro všechny výpočty v systému. Nejprve obecněji k částicovým systémům. Budu parafrázovat z [8].

Částicový systém je technika pro simulace výpočetně složitých situací, případně jejich modelování. Pod těmito si můžeme představit různé výbuchy, kde se částice rozlétají do všech možných stran. Dále simulace pohybu tekutin, tvoření vírů, plamene ohně atp. Animace těchto jevů se běžně využívají ve filmech, počítačových hrách a jinde. Pro vytvoření těchto simulací sledujeme pohyb námi určených elementárních prvků (částic) v čase, ten zaznamenáváme, a nakonec jej vykreslíme. Tím docílíme toho, že není třeba jev počítat pokaždé, když k němu dojde, jen přehrajeme předem vypočítanou animaci.

Pro zajímavost, částicové systémy se užívají i pro výpočet pohybu tuhého tělesa složitého tvaru (nelze ho nahradit jednoduchými geometrickými tvary). To můžeme aproximovat sítí uzlů, např. získané triangularizací 3D modelu, kterým se přidělí část váhy tělesa.

Prostředí výpočtu

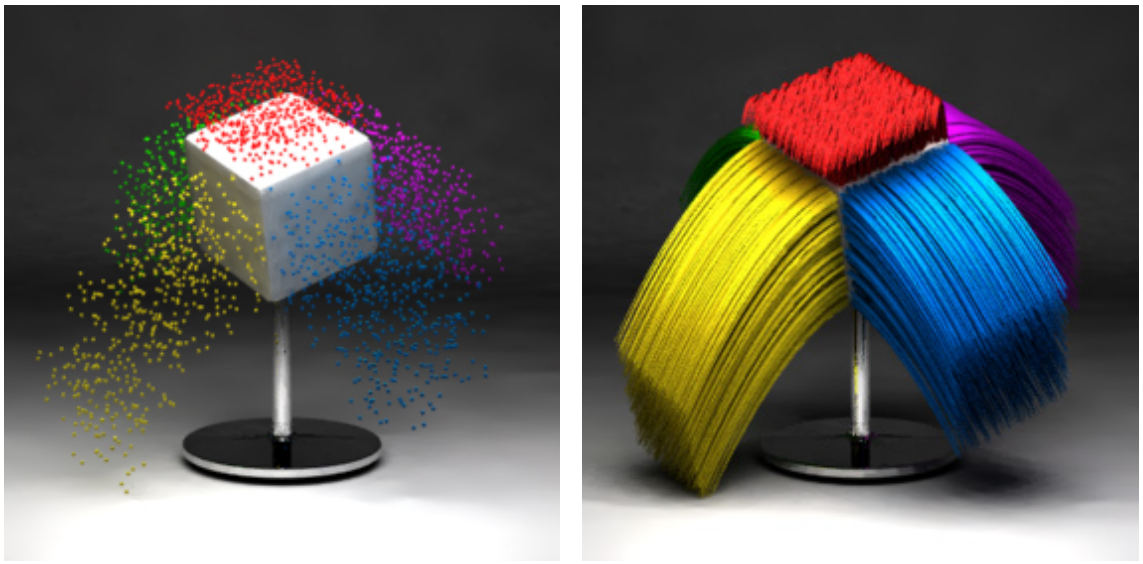
V prostředí se pochopitelně vyskytují částice, jejich počáteční poloha se běžně uvažuje na tzv. „emitoru“. Tím se běžně myslí část roviny, ale lze jej pochopitelně uvažovat jako libovolnou plochu v prostoru, případně přímku, nebo její část, nebo i bod. Já ve svém příkladu budu uvažovat část roviny $y = 0$.

Dále uvažujeme, že částice se pohybují v nějakém vnějším silovém poli, typicky v gravitačním poli Země. Na závěr je možno zahrnout do prostředí překážky, přes které částice nemohou prostupovat. Ty uvažovat nebudu, nicméně by se daly implementovat pomocí detektoru kolizí a následné korekci směru pohybu a rychlosti částice.

Vlastnosti částic

Tím se dostávám k tomu, jaké vlastnosti u každé částice sledujeme. Zejména, jak jsem již uvedl, nás zajímá poloha a orientace v prostoru v nějakých časových okamžicích.

K určení změny polohy a orientace potřebujeme i rychlost částice. Dále můžeme uvažovat různé délky životnosti pro jednotlivé částice, tedy jak dlouho budeme počítat jejich pohyb, než zaniknou. Příkladem, proč toto uvažovat, mohou být jiskry



(a) Vykreslení částic (b) Vykreslení vláken

Obr. 4.1: Metody vykreslování (převzato z [11])

vylétající z ohně, které se po určité době ochladí, takže v noci „zmizí“. Nebudu uvažovat kolize částic navzájem.

Průběh výpočtu

Výpočet se skládá ze dvou kroků. Fáze vlastního výpočtu pohybu a následně fáze vykreslení, nebo animace. Oběma fázím se budu více věnovat později v dokumentaci mé vlastní implementace.

Vykreslení

Částicové systémy se většinou používají k výpočtu tzv. snímků (angl. *frames*). Tím dostáváme dvě možnosti – vykreslením jednotlivých snímků postupně získáme animaci (Obr. 4.1a), nebo můžeme jednotlivé polohy částic proložit křivkou a získat jakési vlákno (Obr. 4.1b). V prvním případě se pak běžně volí frekvence 24 *snímků za sekundu – frames per second* (FPS).

5 GArticle Engine

Název této kapitoly, a i celé práce, je upozorněním na fakt, že výpočty budou probíhat v geometrické algebře (GA – Geometric Algebra).

Jde o, v jistém smyslu, dokonalejší systém. Většina částicových systémů uvažuje jako částici bod. To je jednodušší, ale u bodu je bezpředmětné uvažovat jeho orientaci. Můj systém umí počítat pohyb při „rozhození“ těles do prostoru, u kterých uvažujeme moment setrvačnosti a orientaci v prostoru. Tento scénář je pochopitelně mnohem složitější a pro velké množství těles je nutné kód optimalizovat. Právě zde je PGA velmi vhodným nástrojem.

Jako tělesa reprezentující částice jsem vybral krychle pro jednoduchost jejich inerciálního tenzoru a vykreslování. Nicméně implementace umožňuje použít libovolný tvar, u kterého budeme znát hmotnost a inerciální tenzor.

Vybral jsem si pro řešení jazyk C# a projekt je ve formátu Visual Studio Solution. Snažil jsem se program psát ve filosofii objektově orientovaného programování (OOP). Je tedy členěn do několika souborů a jmenných prostorů.

Také bych chtěl zmínit, že vlastní implementace se od odvozených rovnic výše liší v jistých znaménkových konvencích, které jsou dány reprezentací rovnice 3.7 a zobrazení A. U lineárních složek sil a rychlostí je potřebné otočit znaménko a u sil nefiguruje koeficient 2, ale $\frac{1}{2}$.

5.1 Program

Nejprve tedy popíši soubory, ze kterých se můj program skládá.

pga3d.cs

Tento soubor jsem původně získal z [12]. Nacházela se v něm třída PGA3D spolu s ukázkovým použitím. Tato třída by sama o sobě stačila pro správné výpočty. Nicméně cílem bylo zároveň použité operace optimalizovat pro jednotlivé prvky PGA. Reprezentace celého gradovaného prostoru má 16 bladů. Pokud bych používal PGA3D pro všechny objekty, paměťová náročnost by se, v závislosti na konkrétní klasické implementaci, se kterou bychom PGA implementaci srovnávali, v lepším případě nezvýšila. To proto, že bych používal reprezentaci pomocí 16 koeficientů pro objekty, stejně jako tomu je u klasické reprezentace.

Vytvořil jsem tedy další čtyři třídy PGAPlane, PGALine, PGAPoint, PGAMotor. Jde o optimalizovanou reprezentaci vektorů, bivektorů, trivektorů a motorů.

```
0 references
static void Main(string[] args)
{
    // Motor a
    PGA3D a = new PGA3D();
    a._mVec = new string[] { "a0", "", "", "", "", "a1", "a2", "a3", "a4", "a5", "a6", "", "", "", "", "a7" };

    // Bivektor b
    PGA3D b = new PGA3D();
    b._mVec = new string[] { "", "", "", "", "", "b0", "b1", "b2", "b3", "b4", "b5", "", "", "", "", "" };

    // Soucin
    PGA3D res = (a*b);
    Console.WriteLine(res.ToString());
}
```

(a) Zadání pro výpočet

```
C:\WINDOWS\system32\cmd.exe
(-b3*a4-b4*a5-b5*a6) +
(+b0*a0-b3*a2+b4*a3+b1*a4-b2*a5-b5*a7)e01 +
(+b1*a0+b3*a1-b5*a3-b0*a4+b2*a6-b4*a7)e02 +
(+b2*a0-b4*a1+b5*a2+b0*a5-b1*a6-b3*a7)e03 +
(+b3*a0+b5*a5-b4*a6)e12 +
(+b4*a0-b5*a4+b3*a6)e31 +
(+b5*a0+b4*a4-b3*a5)e23 +
(+b5*a1+b4*a2+b3*a3+b2*a4+b1*a5+b0*a6)e0123 +

Press any key to continue . . .
```

(b) Výsledek symbolického výpočtu

Obr. 5.1: Program pro symbolické výpočty

Obsahují pole pouze tak dlouhé, z kolika bladů se skládají. Z toho plyne, že musím také optimalizovat jednotlivé operace. V první řadě je třeba zmínit, že jsem optimalizoval jen určitý výčet operací v každé třídě.

Pozn. Po ruční optimalizaci několika operací, jsem vytvořil verzi pga3D.cs pracující se znakovými řetězci plné délky. To jsem následně použil k optimalizaci dalších operací, včetně sendwich součinu pro pohyb. Na Obr. 5.1 je ukázkové použití pro výpočet násobení motoru $\boldsymbol{a}$ a bivektoru $\boldsymbol{b}$. Tento program jsem používal i při hledání správného tvaru rovnice 3.7. Přikládám tedy i tento program v příloze B pro volné užití a modifikaci.

Kromě optimalizace operací jsem také přidal metodu *ToPGA*(). Je to v podstatě přetypování do obecné třídy PGA3D, které používám především pro výpis na konzoli, který byl v této třídě již vytvořen.

Mohla by vyvstat otázka, proč jsem zde nepoužil dědičnost, když jde o jasnou hierarchii – objekt PGA obsahuje všechny podmnožiny svých bladů. To především z praktického důvodu. Každý objekt v mnou definovanými třídami by totiž implicitně zdědil pole koeficientů rodiče, čemuž je žádoucí se vyhnout.

Environment.cs

Zde se zatím vyskytuje jen jedna metoda *Forces*(), která vrací bivektor reprezentující síly $\mathbf{f}^s$ v obecném souřadnicovém systému. Pokud bychom uvažovali překážky, definovali bychom je v tomto souboru.

Particles.cs

Soubor Particles.cs obsahuje jedinou třídu Particles. Z názvu je očividné, že se jedná o třídu částic. Začnu popisem atributů.

Statické atributy *m, I, det, IInv, A* a *AInv* definují, jaké těleso uvažujeme za částici. Snažil jsem se držet notace z Kapitoly 3. I reprezentuje inerciální tenzor a m hmotnost tělesa. Tyto dva atributy pak využívám k sestavení matice A a její inverze *AInv*. Ve skutečnosti tuto 6×6 matici reprezentuji jako vektor a pomocí speciálně definovaného násobení v třídě PGALine dosáhnu stejného výsledku, jako při výpočtu $\boldsymbol{y} = A\boldsymbol{x}$, kde $A \in \mathbb{R}^{6\times6}, \boldsymbol{x}\,, \boldsymbol{y} \in \mathbb{R}^6$.

Pole *cube* obsahuje vrcholy krychle s jednotkovou stranou, průsečíkem tělesových úhlopříček v počátku obecného souřadného systému a normálovými vektory stěn mířícími ve směru jedné ze souřadnicových os.

Dále každá částice obsahuje svoje jedinečné ID, motor *position* pozice a orientace v prostoru a bivektor rychlosti *velocity*. Poslední dva atributy *lifetime* a *framesAlive* jsou přípravou pro rozdílné životnosti každé částice. Tento scénář v příkladech uvažovat nebudu, nicméně v programu je tato možnost.

Metoda *vDot*() je reprezentace rovnice 3.7. Symbol % je značení pro komutátor (viz vztah 3.5). Tuto rovnici používám v numerickém řešení, k tomu se dostanu v následující podsekci. Protože síly v souboru Environment.cs jsou vyjádřené v obecném systému, musíme je přesunout do systému tělesa $\mathbf{f}^b = \mathbf{g}\,\mathbf{f}^s\,\widetilde{\mathbf{g}}$.

Metoda *ToPoint*() vrací bod $O = [0, 0, 0]$ transformovaný pomocí motoru částice a *ToPlane*() vrací transformovanou rovinu $\rho : z = 1$.

ODESolver.cs

V tomto souboru se nachází metoda pro numerické řešení rovnic 3.6,3.7. Rovnici 3.7 řeším Runge–Kuttovou metodou 4. řádu (RK4, [13]). Řešení rovnice 3.6 můžeme napsat následujícím způsobem:

$$\dot{\mathbf{g}} = \mathbf{g}\,\mathbf{v}^b \Rightarrow \mathbf{g}(t) = e^{\int_0^t \mathbf{v}^b(\tau)d\tau}. \tag{5.1}$$

Exponenciálu aproximuji součtem několika členů její nekonečné řady. K aproximaci integrálu v argumentu exponenciály jsem využil to, že v RK4 používám

iterace. V rámci každé iterace vypočítám aproximaci tohoto integrálu levobodovou obdélníkovou formulí ([14]) a následně vypočítám aproximaci celé exponenciály.

Návratovým typem této metody je dynamické pole motorů vypočítaných v iteracích.

ColourGenerator.cs

V souboru ColourGenerator.cs se nachází generátor dobře rozlišitelných barev v hexadecimálním formátu. Převzal jsem ho z [15], využívám jej v rámci samotného částicového systému, kde pro každou částici je vygenerována barva pro vykreslení.

GarticleEngine.cs

Zde se nachází třída GarticleEngine. Obsahuje definici vlastní implementace mého částicového systému.

Atribut *particles* obsahuje částice systému, *positions* jsou vypočítané motory metodou *Solve*() pro každou částici. Parametry *framesToCalculate* a *numOfParticles* se zadávají v konstruktoru. Nesou informaci, kolik částic bude engine obsahovat a kolik iterací proběhne v numerickém řešiči.

Metoda *Initialize*(PGALine *initVelocity*) generuje částice. Zde předávám všem částicím stejnou počáteční rychlost, zadanou jako parametr této metody, a stejný počet kroků ve výpočtu. Mohli bychom vygenerovat náhodné hodnoty, nebo i hodnoty normálně rozdělené se střední hodnotou v těchto uživatelem zadaných hodnotách, ale pro jednoduchost uvažuji, jak jsem již řekl, stejné hodnoty počáteční rychlosti a kroků pro všechny částice. Metoda *Run*(float[] *tRange*, float *eps*) počítá aproximaci pohybu částic.

Metody *ToCube*(PGAMotor *g*, string *col*) a *ToString*(int *N*) slouží k vykreslení vypočteného pohybu částic v rámci experimentální platformy Stevena de Kenicka([16]). Výstupem *ToString*(int *N*) je výpis pohybu barevných krychlí, který je možno zkopírovat do skriptu v příloze C a spustit na této stránce. Výsledkem je interaktivní prostředí, ve kterém je scéna vykreslena.

5.2 Výsledky programu

Nejprve ukáži, jak program počítá na příkladech s jednou částicí. Abych eliminoval vzájemnou interakci lineárních a rotačních působení, rozdělím zvlášť tyto dva případy. Důvod, proč budu eliminovat interakci těchto dvou druhů působení, je, že pak

lze velmi jednoduše zkontrolovat výsledky při řešení na intervalu $t \in \langle 0, 1\rangle$. Počáteční pozice je stejná, jak ji vyjadřuje pole *cube* popsané v 5.1. Výsledky okomentuji pomocí metod *ToPoint*() a *ToPlane*() (opět viz 5.1).

Lineární působení

Jak jsem již předeslal, zde budu uvažovat točivý moment $\boldsymbol{\tau} = (0, 0, 0)^T$ Nm a úhlovou rychlost $\boldsymbol{\omega}(0) = (0, 0, 0)^T$ rad $\cdot$ s^{-1}.

Příklad 3. $\boldsymbol{F} = (-1, 0, 0)^T$ N, $\boldsymbol{v}(0) = (1, 0, 0)^T$m $\cdot$ s^{-1}.

Řešení 3. V PGA bychom sílu a počáteční rychlost vyjádřili $\mathbf{f}^s = \frac{1}{2}\boldsymbol{e}_{23}$ N a $\mathbf{v}^b(0) = -\frac{1}{2}\boldsymbol{e}_{01}$ m$\cdot$s^{-1}. Očekávaný výsledek je posunutí bodu $\boldsymbol{X} = \boldsymbol{e}_{123}$ do $\boldsymbol{X}_1 = \frac{1}{2}\boldsymbol{e}_{032} + \boldsymbol{e}_{123}$. Rovina $\boldsymbol{\rho} = \boldsymbol{e}_3 - \boldsymbol{e}_0$ by se pohybem neměla změnit, protože pohyb je ve směru vektoru, který je kolmý k její normále. Příkladné spuštění je na Obrázku 5.2a, výsledek na Obrázku 5.2b. Až na zaokrouhlovací chyby vyšlo přesně to, co jsme očekávali.

Příklad 4. $\boldsymbol{F} = (-1, -1, 0)^T$ N, $\boldsymbol{v}(0) = (0, 0, 1)^T$m $\cdot$ s^{-1}.

Řešení 4. V PGA bychom sílu a počáteční rychlost vyjádřili $\mathbf{f}^s = \frac{1}{2}\boldsymbol{e}_{23} + \frac{1}{2}\boldsymbol{e}_{31}$ N a $\mathbf{v}^b(0) = -\frac{1}{2}\boldsymbol{e}_{03}$ m$\cdot$s^{-1}. Očekávaný výsledek je posunutí bodu $\boldsymbol{X} = \boldsymbol{e}_{123}$ do $\boldsymbol{X}_1 = -\frac{1}{2}\boldsymbol{e}_{032} - \frac{1}{2}\boldsymbol{e}_{013} + \boldsymbol{e}_{021} + \boldsymbol{e}_{123}$. Rovina $\boldsymbol{\rho} = \boldsymbol{e}_3 - \boldsymbol{e}_0$ by se pohybem měla posunout. Výsledná rovina je $\boldsymbol{\rho}_1 = \boldsymbol{e}_3 - 2\boldsymbol{e}_0$. Příkladné spuštění je na Obrázku 5.3a, výsledek je na Obr. 5.3b. Opět při zanedbání zaokrouhlovacích chyb vychází požadované hodnoty.

Rotační působení

Nyní budu uvažovat $\boldsymbol{F} = (0, 0, 0)^T$ N a $\boldsymbol{v}(0) = (0, 0, 0)^T$ m $\cdot$ s^{-1}. Bod $\boldsymbol{X}$ by se nyní pohybovat neměl, protože jde o rotace kolem os procházejících počátkem.

Příklad 5. $\boldsymbol{\tau} = (3\pi, 0, 0)^T$ Nm, $\boldsymbol{\omega}(0) = \left(\frac{\pi}{2}, 0, 0\right)^T$ rad $\cdot$ s^{-1}.

Řešení 5. V PGA bychom sílu a počáteční rychlost vyjádřili $\mathbf{f}^s = \frac{3\pi}{2}\boldsymbol{e}_{01}$ Nm a $\mathbf{v}^b(0) = \frac{\pi}{4}\boldsymbol{e}_{23}$ rad$\cdot$s^{-1}. Očekávaný výsledek je rotace roviny $\boldsymbol{\rho} = \boldsymbol{e}_3 - \boldsymbol{e}_0$ do $\boldsymbol{\rho}_1 = -\boldsymbol{e}_3 - \boldsymbol{e}_0$. Příkladné spuštění a výsledek jsou na Obr. 5.4a, 5.4b. Zde je vidět, že rotace jsou náchylnější na numerické chyby, ale stále je chyba v motoru v řádu setin.

Příklad 6. $\boldsymbol{\tau} = (0, 0, 0)^T$ Nm, $\boldsymbol{\omega}(0) = (\pi, \pi, 0)^T$rad $\cdot$ s^{-1}.

Řešení 6. V PGA bychom sílu a počáteční rychlost vyjádřili $\mathbf{f}^s = 0$ Nm a $\mathbf{v}^b(0) = \frac{\pi}{2}\boldsymbol{e}_{12} + \frac{\pi}{2}\boldsymbol{e}_{31}$ rad$\cdot$s^{-1}. Příkladné spuštění je na Obrázku 5.5a. Výsledek je na Obr. 5.5b. Při této kombinací rotací už není tak jednoduché zjistit, jak přesně by měla vypadat

rovnice otočené roviny. Můžeme ovšem využít znalosti rotací kolem jednotlivých os a spočítat výslednou rotaci jako jejich složení. Tímto způsobem jsem dostal přesnou hodnotu otočené roviny $\boldsymbol{\rho}_{1,přesná} = -\boldsymbol{e}_0 + \frac{1}{2}\boldsymbol{e}_1 + \frac{1}{2}\boldsymbol{e}_2 + \frac{\sqrt{2}}{2}\boldsymbol{e}_3$. Rovina vypočtená numericky je $\boldsymbol{\rho}_1 = -1.0000033\boldsymbol{e}_0 + 0.68158484\boldsymbol{e}_1 + 0.63312835\boldsymbol{e}_2 + 0.36687496\boldsymbol{e}_3$. Zde už se přesné a numerické řešení rozchází v řádu desetin.

Výsledky GArticle Enginu

Na závěr přidám příklady vypočtené pro větší počet krychlí najednou. Budu uvažovat silové působení $\boldsymbol{F} = (0, 0, -1)^T$ N, jelikož si jej lze představit jako obdobu gravitační síly. Nebudu přikládat spouštěcí kódy, protože je lze všechny získat změnou bivektoru rychlosti ve Výpisu 5.1. Všechny příklady budu řešit na intervalu $t \in \langle 0, 1.5 \rangle$.

Příklad 7. $\mathbf{v}^b(0) = \frac{1}{2}\boldsymbol{e}_{01} + \frac{1}{2}\boldsymbol{e}_{02}$

Řešení 7. Vykreslení je na Obr. 5.6. Jsou vyobrazeny 3 pozice každé krychle – počáteční, uprostřed časového intervalu a konečná. Deformace krychlí je způsobena stylem, jakým pracuje [16] s perspektivou.

Příklad 8. $\mathbf{v}^b(0) = \frac{\pi}{4}\boldsymbol{e}_{23}$

Řešení 8. Vykreslení je na Obr. 5.7. Opět se projevuje jistá deformace způsobená perspektivou a také jsem pro lepší přehlednost vykreslil jen počáteční a konečný stav.

Příklad 9. $\mathbf{v}^b(0) = -\frac{1}{2}\boldsymbol{e}_{01} - \boldsymbol{e}_{02} + \frac{\pi}{4}\boldsymbol{e}_{23}$

Řešení 9. Zde provedu výpočet pro 25 a pro 36 krychlí a vykreslím každou opět ve 3 pozicích. Už pro 25 krychlí je vykreslení dosti nepřehledné (Obr. 5.8a) a pro 36 se už v něm opravdu nedá dobře vyznat (Obr. 5.8b). Zároveň se u okrajových krychlí znatelně projevuje zkreslení kvůli perspektivě. Dalším logickým krokem by tedy bylo vytvoření vlastní zobrazovací techniky.

```
GarticleEngine ge = new GarticleEngine(1000, 9);                    1
PGALine v = new PGALine(new float[] { 0, 0, 0, 0, 0, 0 });          2
ge.Initialize(v);                                                   3
ge.Run(new float[] { 0, 1.5f }, (float)1E-10);                      4
Console.WriteLine(ge.ToString(3));                                  5
```

Výpis 5.1: Příklad spuštění GArticle Enginu

Závěr

GArticle Engine je pokročilým nástrojem pro počítání pohybu tuhých těles. Ze znalosti hmotnosti a inerciálního tenzoru tělesa jsme nyní schopni relativně úsporně počítat pohyb takových těles. V průběhu práce bylo ukázáno, že geometrické algebry jsou velmi vhodnou pomůckou pro aplikaci fyzikálních poznatků. Patrné to je ve zjednodušení rovnic pohybu 3.1 – 3.4 na dvě rovnice 3.6 a 3.7.

V průběhu psaní práce jsem se neustále potýkal s nedostatkem materiálů, ze kterých bych mohl čerpat. Například rovnice zrychlení tuhého tělesa v [6] se neshoduje s poznatky, které jsem ukázal zde, nicméně v závislosti na reprezentaci může být i tento zápis rovnice správný. Ovšem nebylo možné dohledat, jaká reprezentace je v [6] myšlena. Podobná nejednotná nebo nekompletní označení vystupovala na povrch v rámci celé práce. Bylo by tedy dobré, aby celá problematika byla kompletně sjednocena v rámci jednoho dokumentu, kterým by mohla být i tato práce.

Již v průběhu popisu programu jsem narazil na několik oblastí, kde by bylo možné program rozšířit. Kromě vytvoření vlastního zobrazovacího nástroje, a tím pádem čistšího výstupu, by bylo možné inicializovat částice s různými počátečními hodnotami bivektoru rychlosti. Ruku v ruce s tím by ale bylo potřebné začít uvažovat kolize mezi částicemi navzájem, jelikož se nejedná o hmotné body. I k tomuto je ovšem PGA vybavena, viz příklady v [16]. S vystavěným aparátem pro kontrolu kolizí by se pak dalo jednoduše program obohatit o překážky v prostoru.

Bylo by vhodné zmínit paralelizaci výpočtu, zde narážíme na problém. U běžných částicových systémů neuvažujeme vzájemnou interakci mezi částicemi. Je tedy možné je jednoduše paralelizovat. V našem případě by paralelizace vyžadovala množství práce a optimalizace kódu.

Obrázky

```
PGAMotor g = new PGAMotor(new float[] { 1, 0, 0, 0, 0, 0, 0, 0 });
PGALine v = new PGALine(new float[] { -0.5f, 0, 0, 0, 0, 0 });
Particles P = new Particles(1, g, v);

float[] Tint = new float[] { 0, 1 };

Console.WriteLine("f: " + (Environment.Forces()).ToPGA().ToString());
Console.WriteLine("v: " + (P.Velocity).ToPGA().ToString());
Console.WriteLine("g: " + (P.Position).ToPGA().ToString());
Console.WriteLine("~g: " + (~P.Position).ToPGA().ToString());
Console.WriteLine("point: " + P.ToPoint());
Console.WriteLine("plane: " + P.ToPlane());
Console.WriteLine("");

ODESolver.Solve(P, Tint, 1000, (float)1E-10);

Console.WriteLine("v: " + (P.Velocity).ToPGA().ToString());
Console.WriteLine("g: " + P.Position.ToPGA().ToString());
Console.WriteLine("~g: " + (~P.Position).ToPGA().ToString());
Console.WriteLine("point: " + P.ToPoint());
Console.WriteLine("plane: " + P.ToPlane());
Console.WriteLine("");
Console.WriteLine("");

GarticleEngine ge = new GarticleEngine(1000, 1);
ge.Initialize(v);
ge.Run(Tint, (float)1E-10);
Console.WriteLine(ge.ToString(2));
```

(a) Spouštěcí kód Příkladu 3

```
C:\WINDOWS\system32\cmd.exe
v: -0.5e01
f: 0.5e23
g: 1
~g: 1
point: 1e123
plane: -1e0 + 1e3

v: -4.662317E-06e01
f: 0.5e23
g: 1 + -0.25025296e01
~g: 1 + 0.25025296e01
point: 0.5005059e032 + 1e123
plane: -1e0 + 1e3
```

(b) Výsledky Příkladu 3

Obr. 5.2: Výstup k Příkladu 3

```
PGAMotor g = new PGAMotor(new float[] { 1, 0, 0, 0, 0, 0, 0, 0 });
PGALine v = new PGALine(new float[] { 0, 0, -0.5f, 0, 0, 0 });
Particles P = new Particles(1, g, v);

float[] Tint = new float[] { 0, 1 };

Console.WriteLine("f: " + (Environment.Forces()).ToPGA().ToString());
Console.WriteLine("v: " + (P.Velocity).ToPGA().ToString());
Console.WriteLine("g: " + (P.Position).ToPGA().ToString());
Console.WriteLine("~g: " + (~P.Position).ToPGA().ToString());
Console.WriteLine("point: " + P.ToPoint());
Console.WriteLine("plane: " + P.ToPlane());
Console.WriteLine("");

ODESolver.Solve(P, Tint, 1000, (float)1E-10);

Console.WriteLine("v: " + (P.Velocity).ToPGA().ToString());
Console.WriteLine("g: " + P.Position.ToPGA().ToString());
Console.WriteLine("~g: " + (~P.Position).ToPGA().ToString());
Console.WriteLine("point: " + P.ToPoint());
Console.WriteLine("plane: " + P.ToPlane());
Console.WriteLine("");
Console.WriteLine("");

GarticleEngine ge = new GarticleEngine(1000, 1);
ge.Initialize(v);
ge.Run(Tint, (float)1E-10);
Console.WriteLine(ge.ToString(2));
```

(a) Spouštěcí kód Příkladu 4

```
C:\WINDOWS\system32\cmd.exe
v: -0.5e03
f: 0.5e31 + 0.5e23
g: 1
~g: 1
point: 1e123
plane: -1e0 + 1e3

v: 0.49999535e01 + 0.49999535e02 + -0.5e03
f: 0.5e31 + 0.5e23
g: 1 + 0.24974833e01 + 0.24974833e02 + -0.49999535e03
~g: 1 + -0.24974833e01 + -0.24974833e02 + 0.49999535e03
point: 0.9999907e021 + -0.49949667e013 + -0.49949667e032 + 1e123
plane: -1.9999907e0 + 1e3
```

(b) Výsledky Příkladu 4

Obr. 5.3: Výstup k Příkladu 4

```
PGAMotor g = new PGAMotor(new float[] { 1, 0, 0, 0, 0, 0, 0, 0 });
PGALine v = new PGALine(new float[] { 0, 0, 0, 0, 0, MathF.PI / 4 });
Particles P = new Particles(1, g, v);

float[] Tint = new float[] { 0, 1 };

Console.WriteLine("f: " + (Environment.Forces()).ToPGA().ToString());
Console.WriteLine("v: " + (P.Velocity).ToPGA().ToString());
Console.WriteLine("g: " + (P.Position).ToPGA().ToString());
Console.WriteLine("~g: " + (~P.Position).ToPGA().ToString());
Console.WriteLine("point: " + P.ToPoint());
Console.WriteLine("plane: " + P.ToPlane());
Console.WriteLine("");

ODESolver.Solve(P, Tint, 1000, (float)1E-10);

Console.WriteLine("v: " + (P.Velocity).ToPGA().ToString());
Console.WriteLine("g: " + P.Position.ToPGA().ToString());
Console.WriteLine("~g: " + (~P.Position).ToPGA().ToString());
Console.WriteLine("point: " + P.ToPoint());
Console.WriteLine("plane: " + P.ToPlane());
Console.WriteLine("");
Console.WriteLine("");

GarticleEngine ge = new GarticleEngine(1000, 1);
ge.Initialize(v);
ge.Run(Tint, (float)1E-10);
Console.WriteLine(ge.ToString(2));
```

(a) Spouštěcí kód Příkladu 5

```
C:\WINDOWS\system32\cmd.exe
f: 4.712389e01
v: 0.7853982e23
g: 1
~g: 1
point: 1e123
plane: -1e0 + 1e3

v: 29.059877e23
g: -0.6970727 + 0.71699023e23
~g: -0.6970727 + -0.71699023e23
point: 1e123
plane: -0.99998534e0 + -0.9995886e2 + -0.0281
64655e3
```

(b) Výsledky Příkladu 5

Obr. 5.4: Výstup k Příkladu 5

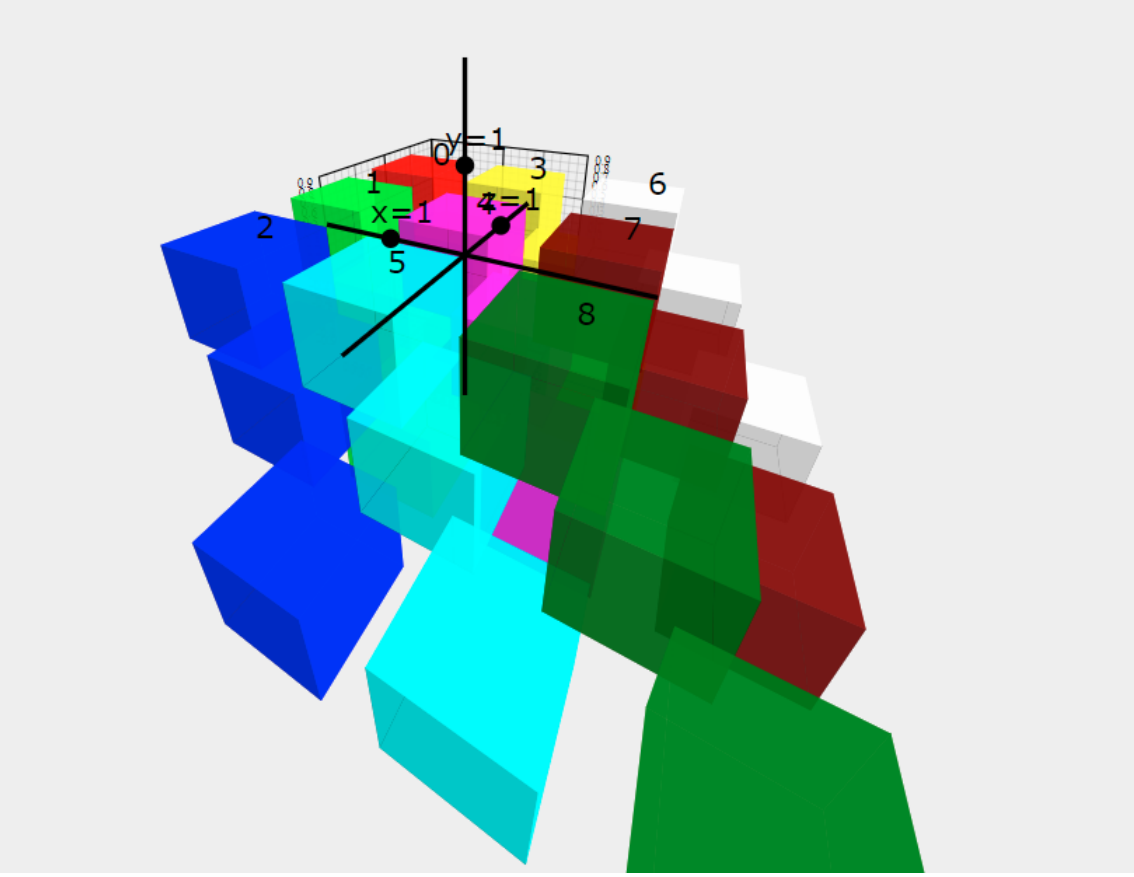
```
PGAMotor g = new PGAMotor(new float[] { 1, 0, 0, 0, 0, 0, 0, 0 });
PGALine v = new PGALine(new float[] { 0, 0, 0, MathF.PI / 2, MathF.PI / 2, 0 });
Particles P = new Particles(1, g, v);

float[] Tint = new float[] { 0, 1 };

Console.WriteLine("f: " + (Environment.Forces()).ToPGA().ToString());
Console.WriteLine("v: " + (P.Velocity).ToPGA().ToString());
Console.WriteLine("g: " + (P.Position).ToPGA().ToString());
Console.WriteLine("~g: " + (~P.Position).ToPGA().ToString());
Console.WriteLine("point: " + P.ToPoint());
Console.WriteLine("plane: " + P.ToPlane());
Console.WriteLine("");

ODESolver.Solve(P, Tint, 1000, (float)1E-10);

Console.WriteLine("v: " + (P.Velocity).ToPGA().ToString());
Console.WriteLine("g: " + P.Position.ToPGA().ToString());
Console.WriteLine("~g: " + (~P.Position).ToPGA().ToString());
Console.WriteLine("point: " + P.ToPoint());
Console.WriteLine("plane: " + P.ToPlane());
Console.WriteLine("");
Console.WriteLine("");

GarticleEngine ge = new GarticleEngine(1000, 1);
ge.Initialize(v);
ge.Run(Tint, (float)1E-10);
Console.WriteLine(ge.ToString(2));
```

(a) Spouštěcí kód Příkladu 6

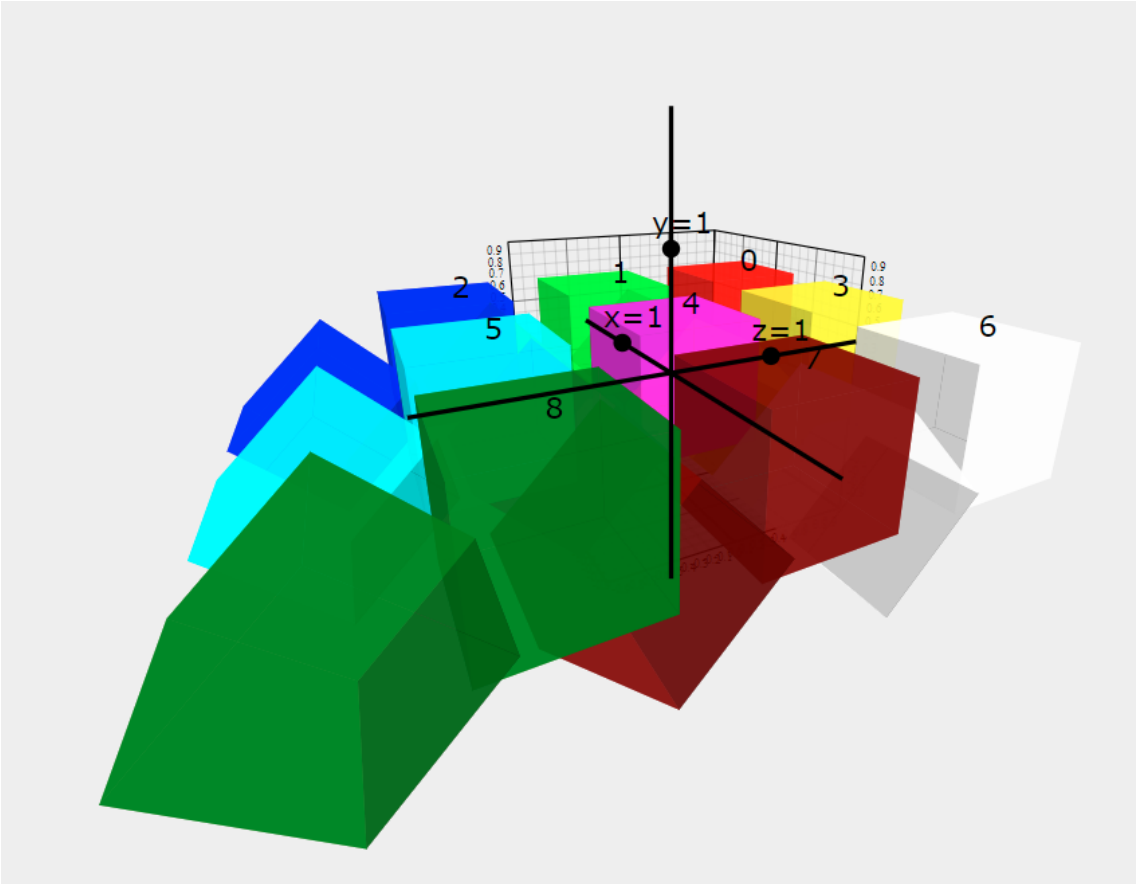
```
C:\WINDOWS\system32\cmd.exe
f: 0
v: 1.5707964e12 + 1.5707964e31
g: 1
~g: 1
point: 1e123
plane: -1e0 + 1e3

v: 1.5707964e12 + 1.5707964e31
g: -0.60570204 + 0.56264037e12 + 0.56264037e31
~g: -0.60570204 + -0.56264037e12 + -0.56264037e31
point: 1e123
plane: -1.0000033e0 + 0.68158484e1 + 0.63312835e2
+ 0.36687496e3
```

(b) Výsledky Příkladu 6

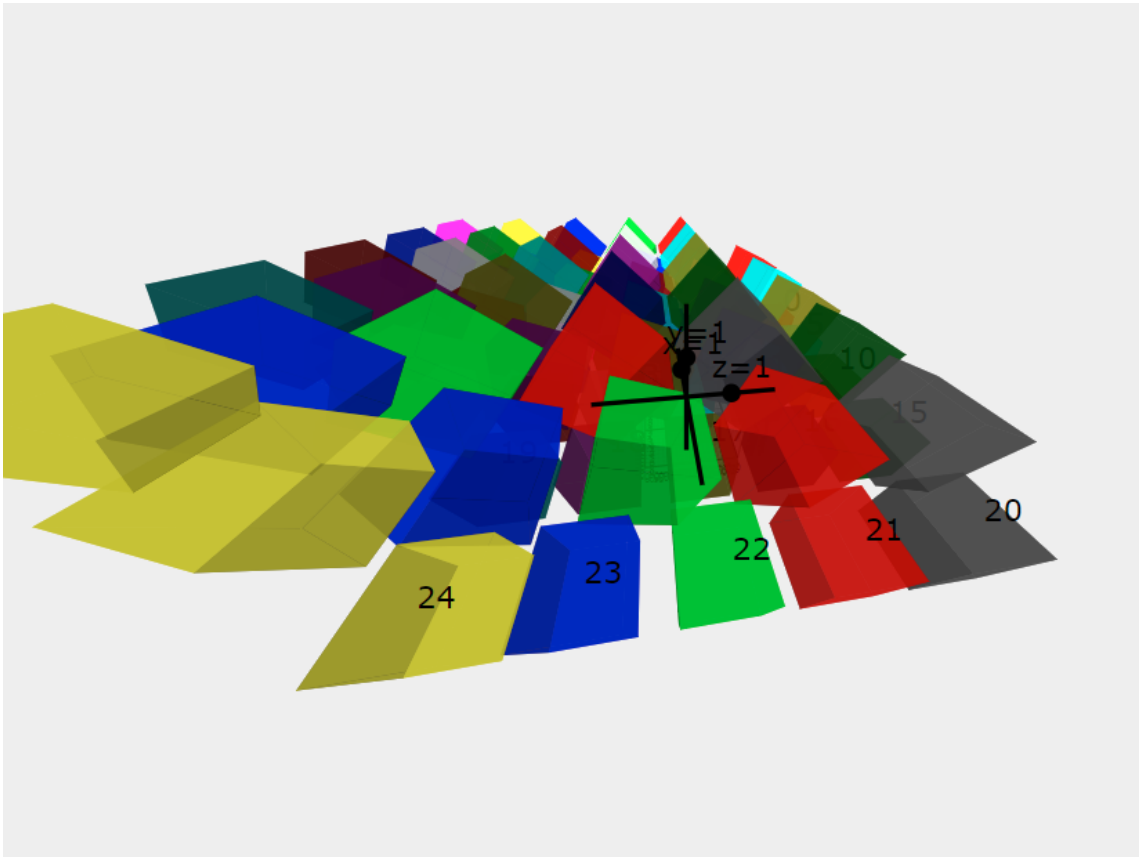
Obr. 5.5: Výstup k Příkladu 6



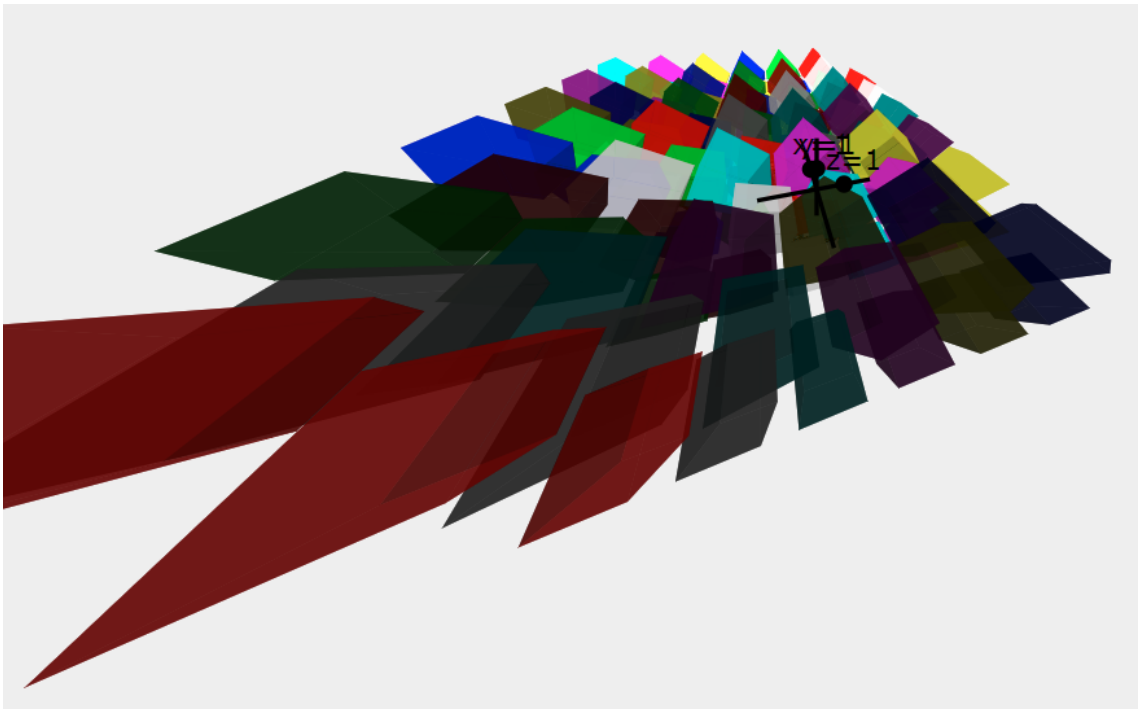
Obr. 5.6: Vykreslení Příkladu 7



Obr. 5.7: Vykreslení Příkladu 8



(a) Vykreslení Příkladu 9 pro 25 krychlí



(b) Vykreslení Příkladu 9 pro 36 krychlí

Obr. 5.8: Vykreslení Příkladu 9

Seznam obrázků

Seznam tabulek

Literatura

[1] PROŠKOVÁ, Jitka: *Duální kvaterniony a jejich aplikace* Praha: Vydavatelství ČVUT v Praze, 2010.

[2] PROŠKOVÁ, Jitka: *Kvaterniony a jejich využití v geometrii* Plzeň, 2006. Bakalářská práce (Bc.). Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra matematiky.

[3] PROŠKOVÁ, J. Kvaterniony, duální kvaterniony a jejich aplikace. Diplomová práce. Plzeň, 2009.

[4] LECLERCQ, Guillaume; LEFÈVRE, Philippe; BLOHM, Gunnar. 3D kinematics using dual quaternions: theory and applications in neuroscience. Frontiers in behavioral neuroscience, 2013, 7: 7.

[5] DORST, Leo; FONTIJNE, Daniel; MANN, Stephen. Geometric algebra for computer science: an object-oriented approach to geometry. Elsevier, 2010.

[6] DE KENINCK, Steven a Charles GUNN. 3D Cheat Sheet. BiVector.net [online]. [cit. 2021-02-02]. Dostupné z: https://bivector.net/doc.html

[7] GUNN, Charles. Geometry, kinematics, and rigid body mechanics in Cayley-Klein geometries. 2011. PhD Thesis. Universitätsbibliothek der Technischen Universität Berlin.

[8] Particle system. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-4-25]. Dostupné z: https://en.wikipedia.org/wiki/Particle_system

[9] JIA, Yan-Bin. Plücker coordinates for lines in the space. Problem Solver Techniques for Applied Computer Science, Com-S-477/577 Course Handout. Iowa State University. http://web. cs. iastate. edu/ cs577/handouts/plucker-coordinates. pdf, 2018.

[10] MURRAY, Richard M., et al. A mathematical introduction to robotic manipulation. CRC press, 1994.

[11] Particle system. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-4-25]. Dostupné z: https://en.wikipedia.org/wiki/Particle_system

[12] /tools. BiVector.net [online]. [cit. 2021-4-29]. Dostupné z: https://bivector.net/tools.html

[13] Rungeova–Kuttova metoda. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-4-30]. Dostupné z: https://cs.wikipedia.org/wiki/Rungeova-Kuttova_metoda

[14] Left, Right, and Midpoint Riemann Sums. Expii [online]. [cit. 2021-4-30]. Dostupné z: https://www.expii.com/t/left-right-and-midpoint-riemann-sums-244

[15] MELDRUM, Sam. Generate distinctly different RGB colors in graphs. Https://stackoverflow.com/ [online]. 21.11.2008 [cit. 2021-4-30]. Dostupné z: https://stackoverflow.com/questions/309149/generate-distinctly-different-rgb-colors-in-graphs

[16] The CoffeeShop [online]. [cit. 2021-4-30]. Dostupné z: https://enkimute.github.io/ganja.js/examples/coffeeshop.html

[17] BRANSON, Jim. Example: The Inertia Tensor for a Cube. Physics 110B [online]. 21.10.2012 [cit. 2021-5-2]. Dostupné z: https://hepweb.ucsd.edu/ph110b/110b_notes/node26.html

Seznam zkratek

PGA projektivní geometrická algebra – Projective Geometrical Algebra

FPS snímků za sekundu – frames per second

OOP objektově orientované programování

RK4 Runge–Kuttova metoda 4. řádu

Přílohy

A ConsoleGarticleEngine

Obsahuje projekt GArticle Enginu a složku se soubory popsanými v 5.1. Formát projektu je Microsoft Studio Solution. Spouští se pomocí zdarma dostupné aplikace Microsoft Visual Studio.

B SymbolicPGA3D

Projekt s programem pro symbolické výpočty v PGA.

C Vykresleni.txt

Textový dokument se skriptem pro zkopírování do konzole v [16]. Po zkopírování výstupu z GArticle Enginu pod komentář „Insert cubes“ je možné si nechat výstup vykreslit.