PA081: Programování numerických výpočtů

2. Numerická stabilita (nejen) elementárních výpočtů

Aleš Křenek

jaro 2013

Kvadratické rovnice

- rovnice tvaru

$$ax^2 + bx + c = 0$$

- školní vzorec

$$x_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Kvadratické rovnice

- rovnice tvaru
$$ax^2 + bx + c = 0$$
- školní vzorec
$$x_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
- je-li $b^2 \gg ac$, počítáme x_+ z rozdílu dvou velmi blízkých čísel
  - problém katastrofálního zrušení
- „$\gg$“ neznamená až příliš velký rozdíl koeficientů
  - typ `float` – přesnost na 7–8 dekadických číslic
  - při $a = 1$ tedy stačí $b \sim 10^3 c$

Kvadratické rovnice

- konkrétně pro $b = 2000$, $c = 1$
- „přesné“ hodnoty
  - $\sqrt{D} = 1999.998999999749999874999921874945 3\ldots$
  - $x_+ = -.000500000125000062500039062527343 77\ldots$
  - $x_- = -1999.999499999874999937499960937 4\ldots$
- pro `float`
  - $\sqrt{D} = 1999.999$
  - $x_+ = -0.00048828125$, chyba 2.5 %
  - $x_- = -1999.9995$, chyba odpovídá přesnosti typu

Kvadratické rovnice

- numericky stabilní řešení pro x_+
- vlastnosti kořenů rovnice:

$$ax^2 + bx + c = (x - x_+)(x - x_-) \quad \text{tedy} \quad c = x_+ x_-$$

  a lze počítat $x_+ = c/x_-$;
- pro předchozí příklad dostáváme
  - $x_+ = -0.00050000014$, chyba odpovídá přesnosti typu
- analogicky pro $b < 0$ je nepřesné x_-, vypočte se symetricky

Kvadratické rovnice

- numericky stabilní řešení pro x_+
- vlastnosti kořenů rovnice:

$$ax^2 + bx + c = (x - x_+)(x - x_-) \quad \text{tedy} \quad c = x_+ x_-$$

  a lze počítat $x_+ = c/x_-$;
- pro předchozí příklad dostáváme
  - $x_+ = -0.00050000014$, chyba odpovídá přesnosti typu
- analogicky pro $b < 0$ je nepřesné x_-, vypočte se symetricky

- i v triviálním výpočtu může vzniknout problém
- řešení může být docela jednoduché

Elementární funkce

- goniometrické, hyperbolické, exponenciální, logaritmické, odmocniny, ...

Elementární funkce

- goniometrické, hyperbolické, exponenciální, logaritmické, odmocniny, ...
- pohled matematika
  - zaběhaný intuitivní aparát
  - příjemné analytické vlastnosti (spojitost, derivace, ...)
  - většina matematických modelů se bez nich neobejde

Elementární funkce

- goniometrické, hyperbolické, exponenciální, logaritmické, odmocniny, ...
- pohled matematika
  - zaběhaný intuitivní aparát
  - příjemné analytické vlastnosti (spojitost, derivace, ...)
  - většina matematických modelů se bez nich neobejde
- pohled programátora
  - samy o sobě numericky stabilní
  - optimalizované implementace v knihovnách
  - problematické chování v okrajových případech vede na numerickou nestabilitu
  - netriviální výpočetní náročnost

Elementární funkce

Co je špatně?

- celý aparát vede na iracionální čísla ($\pi, e, \sqrt{2}, \ldots$)
  - ve většině případů zdroj nepřesnosti
- e^x
  - tendence k přetečení – $e^{88} = 1.65 \times 10^{38}$
- $\tan x$
  - přetečení pro $x \to \frac{\pi}{2}$ (a perioda) roste nade všechny meze
- $\sin x$ a $\cos x$
  - pro $x \to \frac{\pi}{2}$ resp. $x \to 0$ vedou na *špatně podmíněné* rovnice
  - malá změna vstupu vede k velké změně výstupu
  - např. $\sin x = t$ pro $t \to 1$

Co s tím?

- uvědomit si možné problematické chování
- potřebuji skutečně tyto funkce pro svůj výpočet?
  - mnoho problémů má i jiné řešení
- provést transformace, kterými se vyhneme numerické nestabilitě
  - podobné $(a+b)^2 = a^2 + 2ab + b^2$
- podle kontextu zvolit vhodnou implementaci

Algebraické úpravy

- výraz $\sqrt{1+x} - \sqrt{1-x}$
- pro malá x odčítání velmi blízkých čísel
- transformace

$$\begin{aligned}\sqrt{1+x} - \sqrt{1-x} &= \\ &= \frac{(\sqrt{1+x} - \sqrt{1-x})(\sqrt{1+x} + \sqrt{1-x})}{\sqrt{1+x} + \sqrt{1-x}} \\ &= \frac{(1+x) - (1-x)}{\sqrt{1+x} + \sqrt{1-x}} \\ &= \frac{2x}{\sqrt{1+x} + \sqrt{1-x}}\end{aligned}$$

- sčítání velmi blízkých čísel – dostatečně přesné

Algebraické úpravy

- výraz $\ln\sqrt{x+1} - \ln\sqrt{x}$
- pro velká x rozdíl blízkých čísel
- transformace

$$\ln\sqrt{x+1} - \ln\sqrt{x} =$$
$$= \ln(x+1)^{\frac{1}{2}} - \ln x^{\frac{1}{2}} = \frac{1}{2}(\ln(x+1) - \ln x)$$
$$= \frac{1}{2}\ln\frac{x+1}{x} = \frac{1}{2}\ln(1+\frac{1}{x})$$

- možná ztráta přesnosti, ale i tak lepší

Mocninné řady

- Taylorův rozvoj základních funkcí

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{2 \cdot 4} + \frac{3x^3}{2 \cdot 4 \cdot 6} - \dots$$

$$\frac{1}{\sqrt{1+x}} = 1 - \frac{x}{2} + \frac{3x^2}{2 \cdot 4} - \frac{3 \cdot 5x^3}{2 \cdot 4 \cdot 6} + \dots$$

Mocninné řady

Demonstrační příklad

- výraz
$$\frac{1-\cos x}{x^2}$$
se pro $x \to 0$ blíží $\frac{1}{2}$
- pro malá x odečtení dvou blízkých čísel
- konkrétně pro $x = 0.0005$ ve `float`:
$$\begin{aligned} \cos x &= 0.99999988 \\ 1-\cos x &= 1.1920928 \times 10^{-7} \quad \text{(falešná přesnost)} \\ \frac{1-\cos x}{x^2} &= 0.47683709 \end{aligned}$$
- správný výsledek je 0.49999999 (na 8 míst)

Mocninné řady

Demonstrační příklad

- náhrada $\cos x$ Taylorovým rozvojem

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

- pro výpočet na 8 cifer stačí do řádu x^4

$$\frac{1 - \cos x}{x^2} = \frac{1 - 1 + \frac{x^2}{2} - \frac{x^4}{12}}{x^2} = \frac{1}{2} - \frac{x^2}{12}$$

- výsledek výpočtu pro $x = 0.0005$ je 0.49999997
- podstatně lepší přesnost
- řádově méně aritmetických operací

Mocninné řady

Nedostatky

- rychlá konvergence pro malá x, pomalá pro větší
  - přímý důsledek Taylorovy věty
- vyplatí se použít vlastní vzorec pro malé x
  - resp. blízko problematického bodu numerické stability
- v ostatních případech zůstat u knihovní funkce
- jak poznáme, co je „malé x“?

Mocninné řady

Kdy použít

- známe požadovanou přesnost
  - bezpečné je použít přesnost daného typu
  - pro `float` a výsledky ~ 1 je to 10^{-7}
  - při méně přesných vstupních datech méně striktní
- nerovnost $x < c$ musí zajistit, že největší zanedbaný člen řady je menší než požadovaná přesnost

Mocninné řady

Kdy použít

- známe požadovanou přesnost
  - bezpečné je použít přesnost daného typu
  - pro `float` a výsledky ~ 1 je to 10^{-7}
  - při méně přesných vstupních datech méně striktní
- nerovnost $x < c$ musí zajistit, že největší zanedbaný člen řady je menší než požadovaná přesnost
- konkrétně v předchozím příkladu

$$\frac{x^4}{6!} < 10^{-7} \quad \text{tedy} \quad x < 0.09212$$

- zkontrolujeme, zdali dává smysl pro problematickou funkci

$$\cos 0.09212 \doteq 0.99999871$$

je ještě v toleranci

Mocninné řady

Kdy použít

- výsledný kód pro

$$\frac{1-\cos x}{x^2}$$

```
float x,y;
...
if (x < 0.09212)
      y = 0.5 + x*x/12.0;
else
      y = (1-cos(x))/(x*x);
```

Odmocnina

- ve většině používána k normalizaci vektorů
- silové působení, např. dva bodové náboje **a** a **b**
  - velikost síly

$$F = k_c \frac{q_a q_b}{r^2}$$

    kde $r = \|\mathbf{a} - \mathbf{b}\|$
  - silový vektor

$$\mathbf{F} = F \cdot \frac{\mathbf{a} - \mathbf{b}}{r}$$

- snadný výpočet

$$r^2 = \sum (a_i - b_i)^2$$

- r už vyžaduje odmocninu

Odmocnina

Masivní použití

- počítačová grafika – osvětlení plochy
  - zpravidla interpolací počítáme normály k zakřivené ploše
  - vektory mají správný směr ale nejsou jednotkové
  - pro správný výpočet osvětlení potřebná normalizace
- ke všem těmto výpočtům nepotřebujeme striktně $\sqrt{x}$
- $\frac{1}{\sqrt{x}}$ se hodí mnohem více
  - násobení je lepší než dělení
- navíc zpravidla stačí velmi hrubá aproximace
  - citlivý je hlavně směr vektoru, ten zůstává
- přímo Taylorův rozvoj $\frac{1}{\sqrt{1+x}}$
- iterační metody (Newtonova atd.)

Odmocnina

Fast inverse square root

- hra Quake III na konci 90. let
- převratně realistická grafika
- vděčila velmi rychlému výpočtu $\frac{1}{\sqrt{x}}$
- kód zřejmě pochází z grafických knihoven SGI

Odmocnina

Fast inverse square root

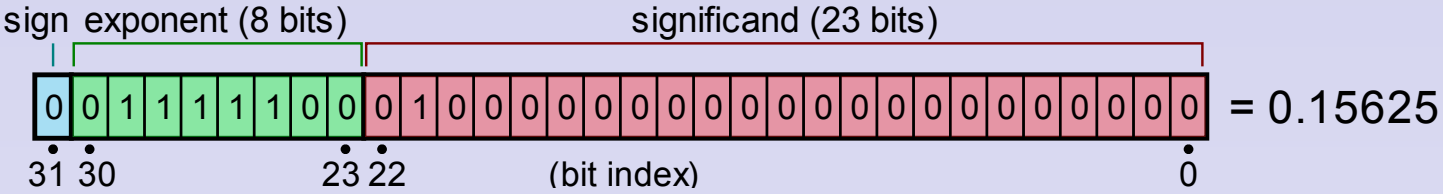
- hra Quake III na konci 90. let
- převratně realistická grafika
- vděčila velmi rychlému výpočtu $\frac{1}{\sqrt{x}}$
- kód zřejmě pochází z grafických knihoven SGI

```
float invSqrt(float x) {
        float xhalf = 0.5f*x,y;
        union {
                float x;
                int i;
        } u;
        u.x = x;
        u.i = 0x5f3759df - (u.i >> 1);
        y = u.x * (1.5f - xhalf * u.x * u.x);
        return y;
}
```

Odmocnina

Fast inverse square root

- funguje na základě IEEE reprezentace čísla $x = (1 + m_x) \times 2^{e_x}$



- přitom $m_x = M_x/2^{23}$ a $e_x = E_x - 127$
- základ výpočtu

$$y = \frac{1}{\sqrt{x}}$$

$$\log_2 y = -\frac{1}{2}\log_2 x$$

$$\log_2(1 + m_y) + e_y = -\frac{1}{2}\log_2(1 + m_x) - \frac{1}{2}e_x$$

Odmocnina

Fast inverse square root

- pro $m \in [0, 1)$ si lze dovolit aproximaci

$$\log_2(1 + m) = m + \sigma$$

- dosazením a úpravami dostaneme

$$E_y \times 2^{23} + M_y = R - \frac{1}{2}(E_x \times 2^{23} + M_x)$$

  tj. červený řádek v kódu s empiricky stanovenou konstantou R
- více viz `http://en.wikipedia.org/wiki/Fast_inverse_square_root`

Odmocnina

Fast inverse square root

- poslední krok je jedna iterace Newtonovy metody
- pro vstup x hledáme takové y aby $y = \frac{1}{\sqrt{x}}$
- tj. hledáme kořen funkce $f(y) = \frac{1}{y^2} - x$
- Newtonovou metodou

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} = y_n - \frac{\frac{1}{y_n^2} - x}{\frac{-2}{y_n^3}} = y_n + \frac{y_n(1 - x y_n^2)}{2}$$

což už odpovídá poslednímu výrazu v kódu

Odmocnina

Fast inverse square root

- aproximace je překvapivě přesná
  - v Quake III je zakomentována další Newtonova iterace
  - nebyla potřeba
- cca. 4× rychlejší než dělení
- dnes překonána instrukcí `rsqrtss`

Odmocnina

Fast inverse square root

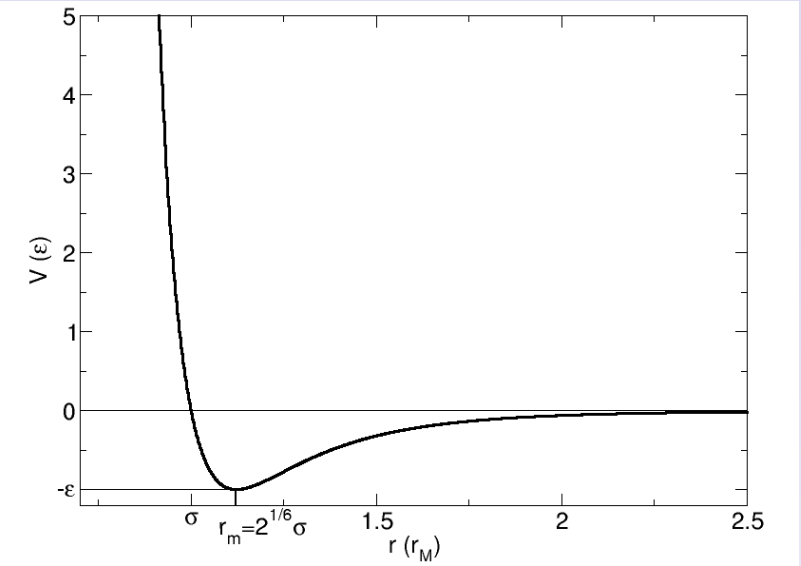
- aproximace je překvapivě přesná
  - v Quake III je zakomentována další Newtonova iterace
  - nebyla potřeba
- cca. 4× rychlejší než dělení
- dnes překonána instrukcí `rsqrtss`
- přesné pochopení konkrétního účelu výpočtu
  - včetně zhodnocení „má smysl tvrdě optimalizovat“
- volba adekvátní metody
  - přispěla k velkému komerčnímu úspěchu
  - pro jiné účely by byla zcela nevhodná

Odmocnina

Lze se jí i vyhnout

- Lenard-Jonesuv potenciál
  - nevazebná interakce dvou atomů
  - významná pro modelování biochemických dějů
- vzorec pro *energii* (potenciál)

$$E_{LJ} = \frac{A}{r^{12}} - \frac{B}{r^6} \quad \text{kde } r \text{ je vzdálenost atomů}$$



Odmocnina

Lze se jí i vyhnout

- výpočet velikosti síly

$$F = \frac{dE}{dr} = -\frac{12A}{r^{13}} + \frac{6B}{r^7}$$

- liché exponenty ⇒ bude potřeba odmocnina

Odmocnina

Lze se jí i vyhnout

- výpočet velikosti síly

$$F = \frac{dE}{dr} = -\frac{12A}{r^{13}} + \frac{6B}{r^7}$$

- liché exponenty ⇒ bude potřeba odmocnina
- zajímá mě většinou silový vektor

$$\mathbf{F} = F\frac{\mathbf{a}-\mathbf{b}}{r} = (\mathbf{a}-\mathbf{b})\left(\frac{6B}{r^8} - \frac{12A}{r^{14}}\right)$$

- tedy vystačím s r^2

Odmocnina

Lze se jí i vyhnout

- implementace

```
float n,r2,r4,r8,r14,F[3];
int i;
for (n=0,i=0; i<3; i++) {
    F[i] = a[i] - b[i];
    n += F[i]*F[i];
}
r2 = 1.0/n;
r4 = r2*r2; r8 = r4*r4;
r14 = r8*r4*r2;

n = 12*A*r14 - 6*B*r8;
for (i=0; i<3; i++) F[i] *= n;
```

Parametrizace rotace

Aneb jak se vyhnout goniometrickým funkcím

- ve 2D jeden úhel ϕ
- složení součtem, inverze $-\phi$
- aplikace násobením maticí

$$R = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

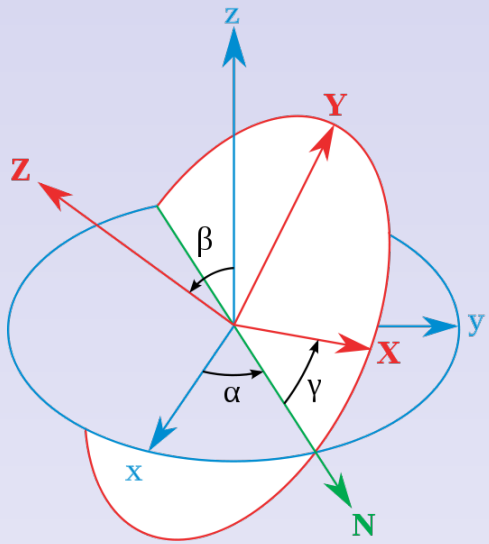
- goniometrickým funkcím se můžeme vyhnout

$$\sin\phi = \frac{2t}{1+t^2} \qquad \cos\phi = \frac{1-t^2}{1+t^2}$$

- numericky příjemné, nevyjádříme $\phi = \pm\frac{\pi}{2}$

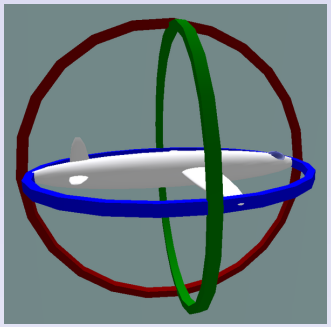
Parametrizace rotace

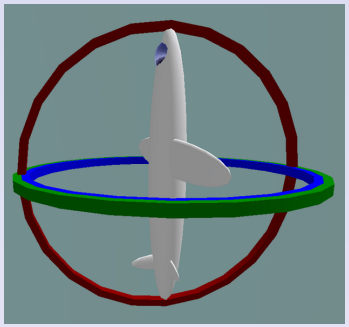
- ve 3D eulerovské úhly
- intuitivní ale numericky a programátorsky nevýhodné
  - značně nepřehledné
  - záleží na pořadí
  - singularity – „gimbal lock“

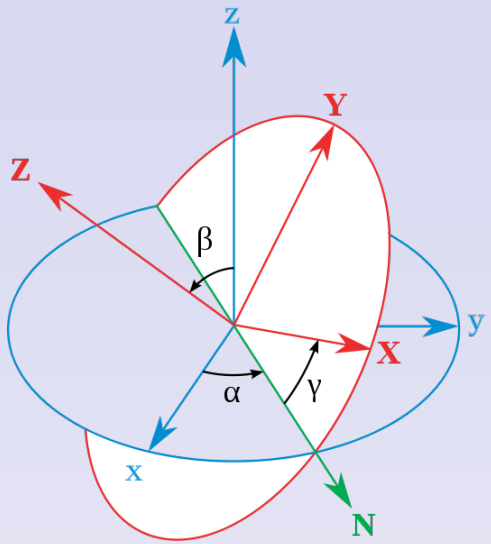


Parametrizace rotace

- ve 3D eulerovské úhly
- intuitivní ale numericky a programátorsky nevýhodné
  - značně nepřehledné
  - záleží na pořadí
  - singularity – „gimbal lock“







Parametrizace rotace

- maticové vyjádření
  - složení 3 rotací podél osy

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{pmatrix}$$

- aplikovatelná stejná substituce
  - vede na komplikované polynomy

Parametrizace rotace

- přímo maticí 3×3
- numericky samo o sobě stabilní, pokrývá všechny případy
- příliš mnoho „parametrů navíc“
- musí být ortogonální
- snadno degeneruje
  - nepřesností výpočtu ztrácí ortogonalitu
  - korekce není jednoduchá
- zabere více paměti

Kvaterniony

- více ekvivalentních definic
- nejjednodušší $\mathbb{H} = \mathbb{R}^4$ s kanonickou bází $(1, i, j, k)$ a násobením
$$i^2 = j^2 = k^2 = ijk = -1$$
- chápeme $\mathbb{R}^3 \subset \mathbb{H}$ jako ryze imaginární kvaterniony
- potom pro $|q| = 1$ je zobrazení $x \mapsto qx\bar{q}$ ortogonální lineární transformace

Kvaterniony

- více ekvivalentních definic
- nejjednodušší $\mathbb{H} = \mathbb{R}^4$ s kanonickou bází $(1, i, j, k)$ a násobením
$$i^2 = j^2 = k^2 = ijk = -1$$
- chápeme $\mathbb{R}^3 \subset \mathbb{H}$ jako ryze imaginární kvaterniony
- potom pro $|q| = 1$ je zobrazení $x \mapsto qx\bar{q}$ ortogonální lineární transformace
- nakrytí 2:1 (q a $-q$ reprezentují stejnou transformaci)
- skládání rotací je násobení
- inverze je $\bar{q}$
- korekce degenerace normalizací

Interpolace rotací

- použití pro animace ve 3D
- lineární interpolace nestačí

$$lerp(q_0, q_1, t) = (1 - t)q_0 + tq_1$$

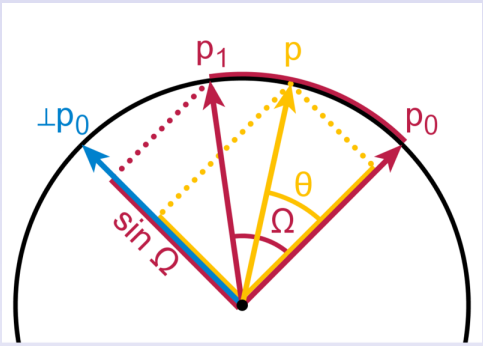
Interpolace rotací

- použití pro animace ve 3D
- lineární interpolace nestačí

$$lerp(q_0, q_1, t) = (1 - t)q_0 + tq_1$$

- sférická lineární interpolace (SLERP)

$$slerp(q_0, q_1, t) = \frac{\sin(1-t)\Omega}{\sin\Omega} q_0 + \frac{\sin t\Omega}{\sin\Omega} q_1$$



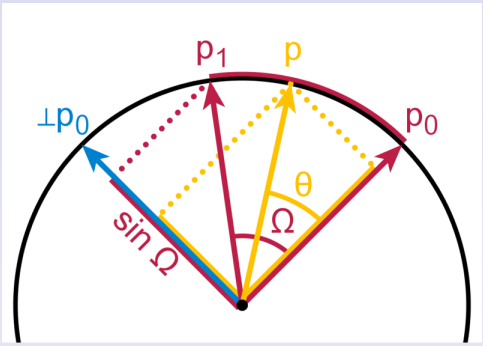
Interpolace rotací

- použití pro animace ve 3D
- lineární interpolace nestačí

$$lerp(q_0, q_1, t) = (1 - t)q_0 + tq_1$$

- sférická lineární interpolace (SLERP)

$$slerp(q_0, q_1, t) = \frac{\sin(1-t)\Omega}{\sin\Omega} q_0 + \frac{\sin t\Omega}{\sin\Omega} q_1$$



- v kvaternionech lze vyjádřit

$$slerp(q_0, q_1, t) = q_0(q_0^{-1}q_1)^t$$

Shrnutí

- i v triviálním výpočtu může dojít k numerickému problému
  - školní řešení kvadratické rovnice
  - zpravidla lze obejít
- elementární funkce
  - samotná implementace numericky stabilní
  - použití ve výrazech může vést na numerické problémy
  - explicitní Taylorův rozvoj pro okrajové případy
- odmocnina a reciproká odmocnina
- kvaterniony jako aparát pro práci s rotacemi