VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Geometrické modelování šíření polygonálního čela vlny

Geometrical modelling of a polygonal wavefront scattering

2016 Jakub Jaroš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání bakalářské práce

Student: **Jakub Jaroš**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 1103R031 Výpočetní matematika

Téma: Geometrické modelování šíření polygonálního čela vlny
Geometrical modelling of a polygonal wavefront scattering

Jazyk vypracování: čeština

Zásady pro vypracování:

Důležitou kvalitou při modelování vlnových rovnic, např. v ultrazvukové defektoskopii letadel, je doba, kterou vlna putuje od zdroje přes překážky (trhliny) k cíli (k senzoru). Známou metodou pro určení této doby je sledování paprsků (ray tracing). Cílem této bakalářské práce je modifikovat ray tracing na případ, kdy víme, že tvary zdroje i překážek jsou polygony. Metodu nazveme polygonal tracing. Práce zahrnuje:
1. studium eikonální rovnice popisující čelo vlny,
2. studium metody ray tracing,
3. analýza geometrie odrazu polygonální vlny,
4. srovnání metod ray a polygonal tracing v defektoskopii.

Seznam doporučené odborné literatury:

[1] B. Enquist, O. Runborg, Computational high frequency wave propagation, Acta Numerica 2003, str. 181-266.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Dalibor Lukáš, Ph.D.**

Datum zadání: 01.09.2014
Datum odevzdání: 29.04.2016

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

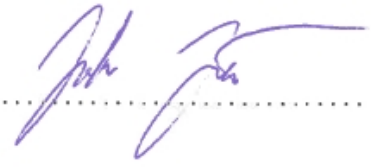
prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 29. dubna 2016

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016



Rád bych na tomto místě poděkoval doc. Ing. Daliboru Lukášovi, Ph.D., za odborné vedení bakalářské práce, rady a náměty a především za jeho trpělivost.

Abstrakt

Práce se zabývá úpravou metody pro sledování šíření vysokofrekvenčních vln ray tracing (sledování paprsků) v ultrazvukové defektoskopii, kde emitor i trhliny jsou jednoduché geometrické útvary, nejlépe polygony. Upravená metoda spočívá v analýze prostředí šíření vlny a následné optimalizaci počtu sledovaných paprsků. Tuto metodu jsme nazvali polygonal tracing.

Klíčová slova: ray tracing, ultrazvuková defektoskopie, eikonálová rovnice, bakalářská práce

Abstract

This thesis is devoted to modification of the wave propagation method ray tracing in ultrasonic defectoscopy, where emitter and flaws are simple geometrical objects ideally polygons. The principle of modified method is in analysis of enviroment of wave propagation and subsequently in optimalization of number of traced rays. We named this method polygonal tracing.

Key Words: ray tracing, ultrasonic defectoscopy, eikonal equation, bachelor thesis

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V současné době je jedním z nejpoužívanějších algoritmů v oblasti modelování šíření vlny ray tracing. Nevýhodou tohoto algoritmu je jeho vysoká výpočetní náročnost, neboť pro co nejpřesnější modelování vlny je potřeba sledovat propagaci tisíců, ne-li miliónů paprsků, což znemožňuje realtimeové nasazení a na výsledek modelování je potřeba velmi dlouho čekat.

Tato práce je motivována ultrazvukovou defektoskopií trhlin. Informace o trajektorii odražených vln umožnuje identifikovat přítomnost a často i polohu trhliny.

Cílem této bakalářské práce je nalézt takovou modifikaci dvourozměrného ray tracingu, která by pro speciální případy kdy emitor vlny i její překážky jsou polygony, dokázala snížit počet potřebných paprsků na minimum a přitom nesnížit přesnost výsledku, a tím rapidně zvýšit efektivitu algoritmu.

2 Čelo vlny a eikonálová rovnice

Tato kapitola slouží k nastínění problému numerického modelování vysokofrekvenčních vln, která je důležitá v mnoha odvětvích jako je například počítačová grafika, seismologie nebo právě ultrazvuková defektoskopie. Nebudeme zabíhat do příliš velkých detailů, protože to není předmětem této bakalářské práce. Pro bližší informace ohledně této problematiky, včetně odvození v této kapitole zmiňovaných vztahů, viz [1].

Šíření akustické vlny v materiálu popisuje vlnová rovnice (1), kde $u(t, x)$ je akustický tlak a $c(x)$ je lokální rychlost šíření vlny v médiu. [2]

$$u_{tt} - c(x)^2 u_{xx} = 0 \tag{1}$$

Přesnost numerického řešení simulace šíření vlny za pomocí vlnové rovnice závisí na počtu bodů mřížky na vlnovou délku. V případech, kdy je frekvence relativně vysoká a vlnové délky jsou v porovnání se sledovanou oblastí malé, by přímá simulace s využitím standardní vlnové rovnice byla příliš výpočetně náročná. Při požadavku na zachování numerické přesnosti roste výpočetní náročnost algebraicky s frekvencí, a pro dostatečně vysoké frekvence už není proveditelná. Proto se pro výpočet vlny musí použít aproximační modely. Tyto modely jsou asymptotické aproximace pro frekvence blížící se nekonečnu. Namísto oscilujícího vlnového pole u, jsou neznámými fáze ϕ a amplituda A, ani jedna z těchto proměnných nezávisí na frekvenci ω a jsou typicky méně proměnlivé než u. Z toho důvodu by měly být i vhodnější pro numerické výpočty. Tyto požadavky splňuje eikonálová rovnice (2), kde $|\cdot|$ značí Eukleidovskou normu v R^n a $\nabla\phi$ je gradient ϕ. [1]

$$\phi_t + c(x)|\nabla\phi| = 0 \tag{2}$$

S eikonálovou rovnicí úzce souvisí i metoda ray tracing, na kterou je možné nahlížet jako na řešení eikonálové rovnice pomocí metody charakteristik (3), kde $p = \nabla\phi(x(t))$ je proměnná hybnosti obvykle nazývaná jako vektor pomalosti[1] a ∇p, ∇x jsou gradienty s respektem k p respektive x. [1]

$$\begin{aligned} \frac{dx}{dt} = \nabla_p H(x,p),\ \frac{dp}{dt} = \nabla_t H(x,p) \\ H(x,p) = c(x)|p|,\ x,p \in R^n \end{aligned} \tag{3}$$

Ray tracing se obvykle nesnaží řešit kompletní Cauchyho úlohu s libovolnými počátečními a hraničními daty, ale najít čas, za který paprsky dosáhnou všech bodů sledované oblasti nebo jen omezeného počtu senzorů.

[1] Anglicky "slowness vector".

V případě, že je médium, ve kterém se paprsky šíří, homogenní jsou paprsky přímky a splňují zákony odrazu a lomu. Řešení se tak redukuje na geometrický problém hledání bodů, ve kterých se paprsky odráží a lámou. [1]

3 Ray tracing

Ray tracing je tradiční metodou pro výpočet cestovních časů vysokofrekvenčních vln. Paprsek $r(t)$ sleduje trajektorii jednoho bodu čela vlny.

3.1 Geometrie ray tracingu

Pro potřeby této bakalářské práce budeme předpokládat, že se paprsky šíří ve dvourozměrném prostředí v rovině, médium, ve kterém se paprsky šíří, je homogenní a trhliny, o které se budou paprsky odrážet a lámat, jsou jednoduché dvourozměrné geometrické útvary, ideálně polygony.

Poloha paprsku r v čase t je dána rovnicí (4), kde u je normovaný směrový vektor a $r(0)$ je zdrojový bod paprsku. [3]

$$r(t) = r(0) + u \cdot t \tag{4}$$

Naší úlohou tedy bude sledovat paprsky a nacházet jejich průsečíky s polygony těchto trhlin. V těchto bodech se paprsky rozdvojí, jedna část paprsků se odrazí zpět do prostředí, druhá se láme směrem do polygonu. Pro jednoduchost budeme takto lámané paprsky ignorovat, a proto budeme tyto útvary v tomto textu dále označovat jako zrcadla.

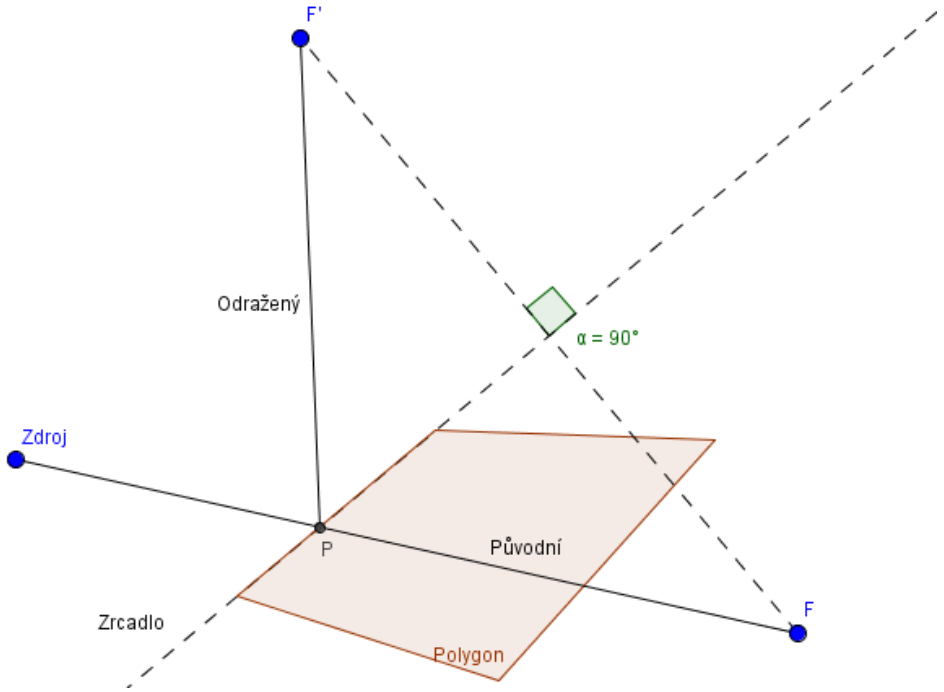
Odražený paprsek se může po své cestě odrážet znovu a znovu, proto budeme celý proces na odražené paprsky aplikovat rekurzivně, dokud se paprsky již dále nebudou odrážet nebo nepřekročíme nějaký námi stanovený limit maximálního počtu odrazů s ohledem na výkon celého algoritmu.

3.1.1 Hledání průsečíku paprsku s polygonem

V případě, že trhlina je polygon, rozložíme polygon na jednotlivé strany, ty pak budeme považovat za samostatná zrcadla, se kterými budeme hledat průsečík paprsku. Nejdříve nalezneme průsečík přímek paprsku a zrcadla. Poté ověříme, že nalezený bod skutečně patří, jak paprsku, tak úsečce zrcadla. Výpočet průsečíků $P = (x, y)$ přímek p_1 a p_2 s normálovými vektory (a_1, b_1) a (a_2, b_2) vede na soustavu obecných rovnic přímek (5), kterou můžeme řešit například Cramerovým pravidlem.

$$\begin{aligned} p_1 &: a_1 x + b_1 yx + c_1 = 0 \\ p_2 &: a_2 x + b_2 yx + c_2 = 0 \end{aligned} \tag{5}$$

Musíme si však dát pozor na některé speciální případy. V případě, že má soustava právě jedno řešení, pak řešení této soustavy je průsečíkem těchto dvou přímek a zbývá rozhodnout, zda průsečík leží zároveň na paprsku a zrcadle. V případě, že soustava nemá řešení, průsečík



Obrázek 1: Odraz paprsku od polygonu

neexistuje (přímky jsou rovnoběžné), a konečně v případě, že má soustava nekonečně mnoho řešení, jsou přímky shodné a paprsek v takovém případě nemusíme odrážet.

V cestě paprsku obvykle stojí více než jedno zrcadlo, o které by se mohl odrážet, proto před samotným zrcadlením musíme určit, od kterého zrcadla se paprsek odrazí nejdříve. Od tohoto zrcadla budeme následně paprsek zrcadlit. Ostatní body odrazu jsou ve stínu tohoto zrcadla a pro další výpočty je nebudeme potřebovat.

3.1.2 Zrcadlení paprsku

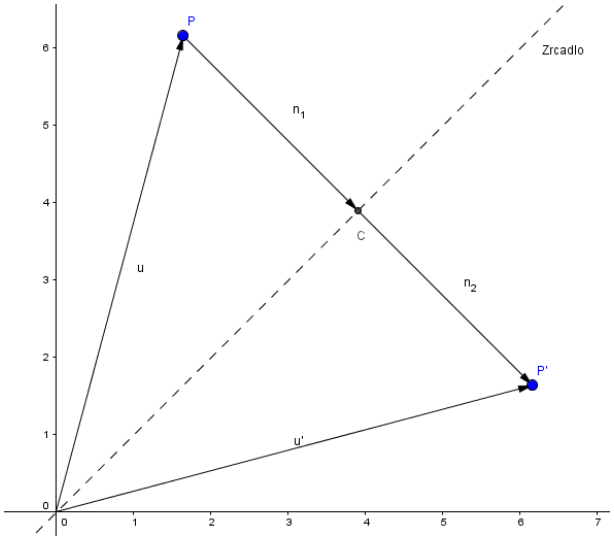
Po nalezení bodu odrazu paprsku zbývá určit, kam paprsek odrazit. K tomu můžeme využít Householderovu matici zrcadlení.

3.1.3 Householderova transformace

Za předpokladu, že přímka zrcadlení prochází počátkem souřadnicového systému, můžeme k zrcadlení paprsku využít Householderovy transformace. Princip transformace je znázorněn na obrázku 2.

Householderova matice zrcadlení (dále jen Householderova matice) má tvar

$$H(n) = I - \frac{2nn^T}{\|n\|^2}, \tag{6}$$



Obrázek 2: Householderova transformace

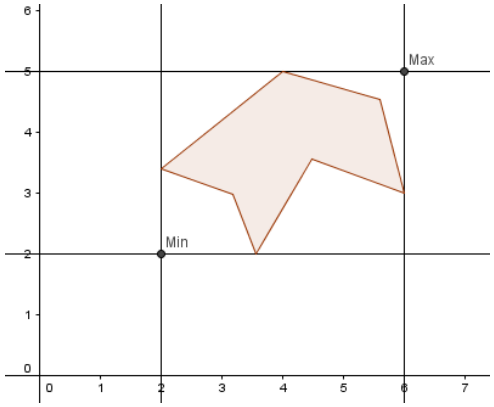
kde I je jednotková matice a n je normála k přímce zrcadlení, kterou získáme například z obecné rovnice přímky zrcadla. Vektor u' zrcadlený podle přímky zrcadla procházející počátkem souřadnicové soustavy pak můžeme spočítat jako $u' = H(n)u$. [4] V případě, že přímka zrcadla neprochází skrze počátek souřadnicového systému, musíme před použitím Householderovy transformace nejprve provést vhodnou translaci souřadnic tak, aby přímka počátkem procházela. Výhodou využití Householderovy transformace je v tom, že matici zrcadlení stačí pro každé zrcadlo spočítat pouze jednou.

3.1.4 Optimializace hledání průsečíků a zrcadel

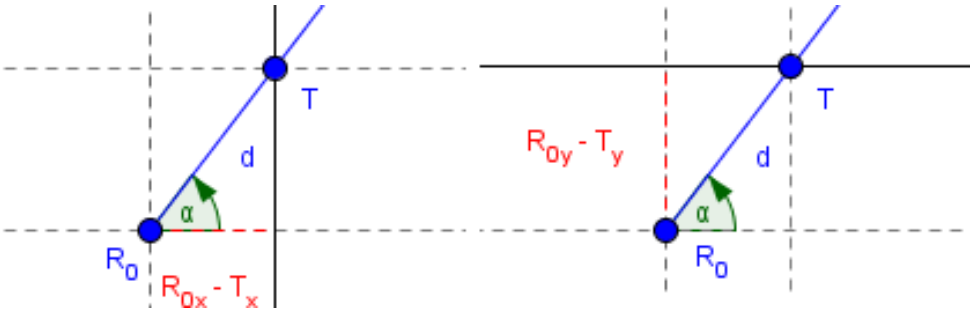
Ověřování, zda paprsky neprotínají celou množinu zrcadel, by bylo příliš výpočetně náročné, proto je vhodné zavést nějaký heuristický prvek, který pro každý paprsek omezí počet zrcadel, se kterými budeme průsečík paprsku hledat. Nejjednodušší metodou je využití bounding boxu, tedy obdélníku s minimálním obsahem, do kterého se vejde celé zrcadlo a jehož strany jsou rovnoběžné s osami souřadnic (obr. 3). Ověření, zda paprsek neprochází tímto obdélníkem je mnohem rychlejší, než ověřování skutečného průsečíku. Pokud paprsek bounding boxem neprochází, je zbytečné dále počítat skutečné průsečíky. Bounding box je jednoznačně definován jen za použití dvou souřadnic, a to souřadnice levého dolního a pravého horního vrcholu. [3]

Vysoce efektivní metodou pro ověření, zda paprsek prochází skrz bounding box, je metoda plátů[2]. Plát je prostor mezi dvěma paralelníma přímkama a průsečík dvou plátů tvoří bounding box. Metoda nalezne pro každý pár paralelních přímek tvořících plát blízký a vzdálený průsečík T_1 a T_2 a jejich orientované vzdálenosti od zdroje paprsku. Orientovanou vzdálenost průsečíku od zdroje paprsku R_0 spočítáme s využitím vlastností jeho normovaného normálového vektoru $n = (x, y) = (\cos\alpha, \sin\alpha)$, kde α je úhel paprsku svíraný s osou x.

[2]Anglicky "slab method".



Obrázek 3: Bounding box polygonu



Obrázek 4: Výpočet d

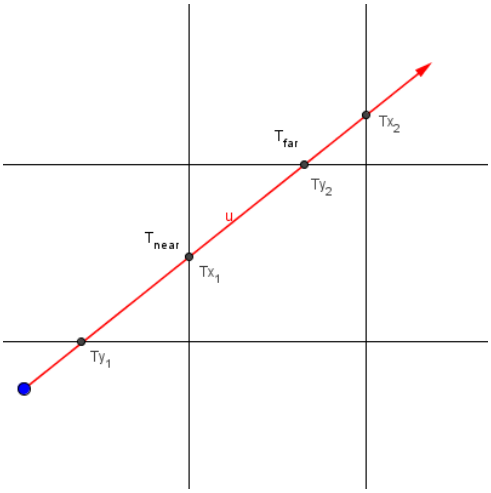
Z obrázku 4 je patrné, že vzdálenost d průsečíků od zdroje paprsku vypočteme pro $n_x, n_y \neq 0$ dle rovnic (7).

$$d_i = \begin{cases} \frac{R_{0x} - T_{ix}}{n_x} & \text{pro průsečíky rovnoběžnými s osou y} \\ \frac{R_{0y} - T_{iy}}{n_y} & \text{pro průsečíky rovnoběžnými s osou x} \end{cases} \tag{7}$$

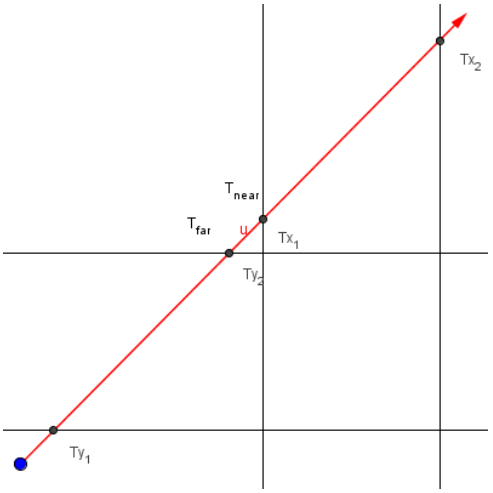
Označme T_{far} jako orientovanou vzdálenost nejbližšího ze vzdálených průsečíků od R_0 a T_{near} jako orientovanou vzdálenost nejvzdálenější z blízkých průsečíků od R_0. Pokud pro daný paprsek platí, že $T_{far} \geq T_{near}$ (obr. 5), pak paprsek bounding boxem prochází. V opačném případě, kdy je $T_{far} < T_{near}$, paprsek bounding box míjí (obr. 6). [3]

Speciálně je potřeba ošetřit případy, kde je některý paprsek rovnoběžný s některou ze souřadnicových os. Implementace je popsána ve výpisu 2.

```
bool intersection(Box box, Ray ray) {
    double t_near = -INFINITY, t_far = INFINITY;

    if (ray.direction.x != 0.0) {
        double tx1 = (box.min.x - ray.origin.x) / ray.direction.x;
        double tx2 = (box.max.x - ray.origin.x) / ray.direction.x;
```



Obrázek 5: Paprsek prochází bounding boxem $T_{far} > T_{near}$



Obrázek 6: Paprsek neprochází bounding boxem $T_{near} > T_{far}$

```
        t_near = min(tx1, tx2);
        t_far = max(tx1, tx2);
    }

    if (ray.direction.y != 0.0) {
        double ty1 = (box.min.y - ray.origin.y) / ray.direction.y;
        double ty2 = (box.max.y - ray.origin.y) / ray.direction.y;

        t_near = max(t_near, min(ty1, ty2));
        t_far = min(t_far, max(ty1, ty2));
    }

    return t_far >= t_near;
}
```

Výpis 1: Implementace ověření existence průsečíku paprsku s bounding boxem

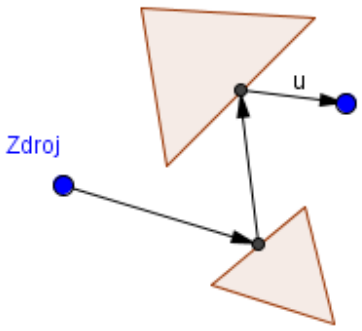
V algoritmu se několikrát vyskytuje náročná operace dělení složkami směrového vektoru paprsku. Počet dělení můžeme snížit na polovinu pomocí násobení obrácenou hodnotou těchto složek, označme je jako inv_x a inv_y. Pokud se můžeme spolehnout na vlastnosti IEEE 754 [5] čísel s plovoucí desetinnou čárkou, elegantně tím eliminujeme i ověřování rovnoběžnosti paprsku s osami – v případě, že některá ze složek směrového vektoru je nulová, a paprsek bude uvnitř jednoho z plátů, výsledné obrácená hodnota této složky bude $infinity$, a tudíž T_{x1} a T_{x2} respektive T_{y1} a T_{y2} $infinity$ opačných znamének. Tím pádem nebudou mít vliv na výsledné T_{far} a T_{near}. V opačném případě, kdy je paprsek vně plátu a hodnoty T_{x1} a T_{x1}, respektive T_{y1} a T_{y2}, budou $infinity$ stejného znaménka, změní hodnoty T_{near} na $+infinity$ nebo T_{far} na $-infinity$, což

v každém případě nesplňuje podmínku na $T_{far} >= T_{near}$. [6] Protože paprsek obecně testujeme proti mnoha bounding boxům dává smysl si inv_x a inv_y předpočítat pro každý paprsek a v dalších výpočtech je použít znovu. [7]

```
bool intersection(Box box, Ray ray) {
    double tx1 = (b.min.x - ray. origin.x) * ray.inv_x;
    double tx2 = (b.max.x - ray. origin.x) * ray.inv_x;

    double tmin = min(tx1, tx2);
    double tmax = max(tx1, tx2);

    double ty1 = (b.min.y - ray.origin.y) * ray.inv_y;
    double ty2 = (b.max.y - ray. origin.y) * ray.inv_y;

    double t_near = max(tmin, min(ty1, ty2));
    double t_far = min(tmax, max(ty1, ty2));

    return t_far >= t_near;
}
```

Výpis 2: Optimalizovaný algoritmus využitím vlastností IEEE 754

Zbývá rozhodnout, jakým způsobem reprezentovat paprsek v paměti počítače. Pokud potřebujeme znát celou trajektorii paprsku, která bude vypadat jako lomená čára (obr. 7), bude vhodné ukládat všechny krajní body dílčích segmentů paprsků, směrový vektor posledního segmentu a pro optimalizaci ověřování existence průsečíku paprsku s bounding boxem i převrácené hodnoty složek směrového vektoru.



Obrázek 7: Trajektorie paprsku

Pokud nás jen zajímá, jakého místa paprsek za určitý čas dosáhl, stačí společně se směrovým vektorem a jeho převrácenými složkami ukládat jen poslední dvě známé polohy paprsku, protože

k výpočtu dalšího odrazu a polohy paprsku stačí vždy pouze poslední segment paprsku. Tím můžeme šetřit operační paměť počítače.

3.2 Použití v počítačové grafice

Protože je pravděpodobnost, že paprsek vyslaný ze světelného zdroje skutečně protne plátno obrazovky velmi malá, byla by implementace reálného fyzikálního modelu zbytečně náročná – drtivá většina paprsků by nikdy neprotnula plátno a my bychom zbytečně plýtvali výpočetním časem na nezajímavé paprsky. Mnohem efektivnější je celý proces obrátit a sledovat pouze paprsky procházející plátnem. Tím docílíme toho, že každý zpracovávaný paprsek je pro naše potřeby zajímavý a neplýtváme výpočetním výkonem. Navíc nás v každém bodu odrazu bude zajímat i barva povrchu, na který paprsek dopadá.

4 Modifikace ray tracingu

V této kapitole se budeme zabývat tím, jak vybrat paprsky, které jsou pro nás zajímavé z pohledu modelování polygonálního čela vlny. Protože předpokládáme, že zdroj i trhliny v prostředí jsou polygony, nazveme tuto modifikaci polygonal tracing.

Myšlenka polygonal tracingu je založena na pozorování, že rovnoběžné paprsky vycházející z jedné úsečky se odrážejí opět na jednu úsečku (obr. 8). Ze zdrojové úsečky tedy stačí spočítat pouze dva paprsky odrážející se od zrcadla, abychom získali informaci o tvaru čela vlny, tím oproti nemodifikované verzi ray tracingu ušetříme spoustu výpočetního výkonu.

V praxi však situace obvykle nebude tak jednoduchá jako na obrázku 8 a segmenty zdroje paprsku budeme muset odrážet obecně proti několika polygonům. To nás staví před následný problém. Svazek paprsků vyslaný ze zdrojové úsečky se může odrážet o více než jednu stranu polygonu. Musíme tedy naleznout body na segmentu zdroje, od kterých se svazky paprsků budou odrážet podle různých zrcadel (obr. 9), což se vizuálně projeví jako lámání čela vlny na více menších segmentů.

Zaveďme proto pojem body podezřelé z lomu čela vlny. Množina těchto bodů bude obsahovat všechny body, o kterých se můžeme domnívat, že se o ně čelo vlny rozlomí na více segmentů. Tyto body však mohou být ve stínu jiných stran polygonů, proto je budeme zatím nazývat právě jen body podezřelými. Tyto body jsou znázorněny na obrázku 10.

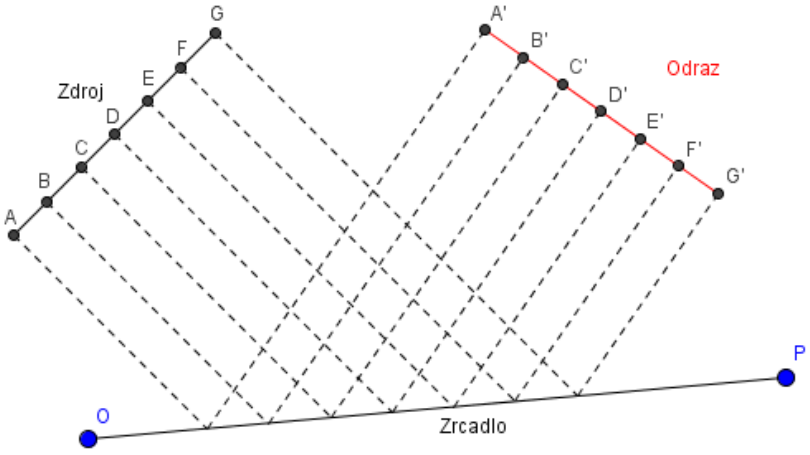
4.1 Nalezení bodů podezřelých z lomu čela vlny

Z obrázku 10 můžeme odvodit, o jaké body se bude jednat. Jsou to

- krajní body čela vlny v čase t,
- průsečíky krajních paprsků se stranami polygonů,
- průsečíky čela vlny se stranami polygonů
- a vrcholy polygonů uvnitř obdélníku definovaného zdrojovým segmentem, krajními paprsky a čelem vlny.

Nalezení první množiny bodů je triviální a nalezení dalších dvou množin je totožné s problémem nalezení průsečíků paprsku a polygonu řešeným v podkapitole 3.1.1, proto se jimi už nebudeme dále zabývat a zaměříme se na nalezení bodů z poslední množiny.

Z obrázku 10 je patrné, že svazek paprsků vycházející ze segmentu zdroje tvoří obdélník A, B, A', B'. Tento obdélník snadno rozdělíme na dva trojúhelníky A, B, A' a A', B', B, pokud vrchol leží uvnitř jednoho z těchto trojúhelníku, leží také uvnitř svazku paprsků tvořících obdélník. K určení, zda vrchol polygonu leží uvnitř jednoho z těchto trojúhelníků, můžeme využít barycentrických souřadnic. [8]



Obrázek 8: Odraz paprsků vycházejících z jedné přímky

Barycentrické souřadnice $(x_0, x_1, \ldots, x_n)$ bodu X vzhledem k simplexu $P_0, P_1, \ldots, P_n$ definujeme předpisem

$$x = \sum_{i=1}^{n} x_i P_i, \text{ s podmínkou} \sum_{i=1}^{n} x_i = 1. \tag{8}$$

Barycentrické souřadnice (a, b, c) bodu X vzhledem k trojúhelníku P_0, P_1, P_2 tedy určíme pomocí následujícího systému lineárních rovnic (9).

$$\begin{aligned} X = &\quad a \cdot P_0 + b \cdot P_1 + c \cdot P_2 \\ 1 = &\quad\quad a + b + c \end{aligned} \tag{9}$$

S přihlédnutím k tomu, že $c = 1 - a - b$ můžeme soustavu rovnic upravit na

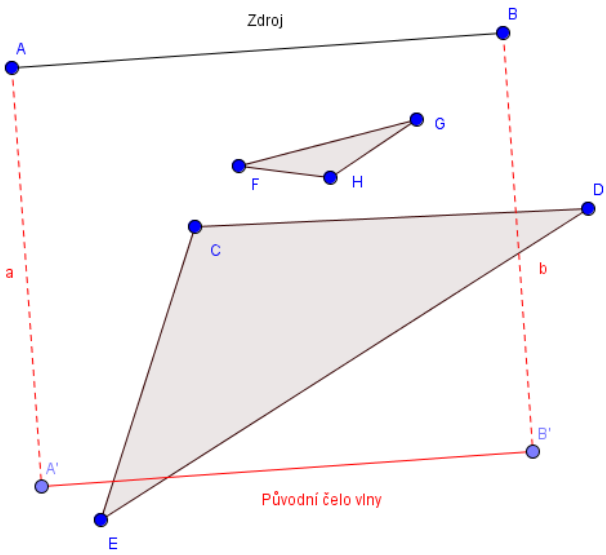
$$X - P_2 = a \cdot (P_0 - P_2) + b(P_1 - P_2)$$

a řešit ji na příklad pomocí Cramerova pravidla. Bod X pak leží uvnitř trojúhelníku P_0, P_1, P_2 právě když platí, že

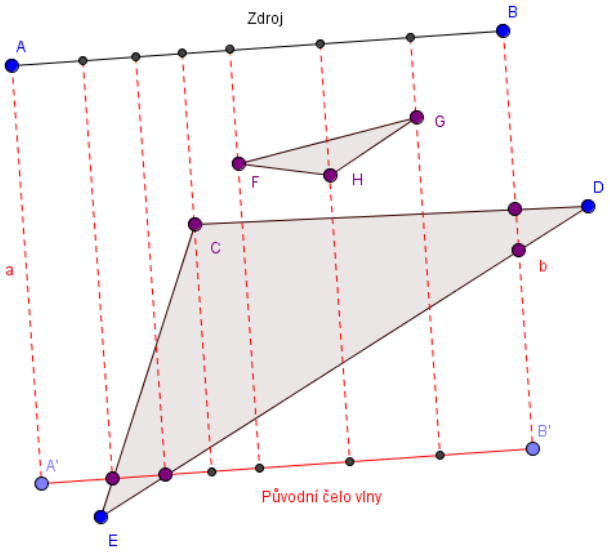
$$0 \leq a \leq 1 \wedge 0 \leq b \leq 1 \wedge 0 \leq c \leq 1.$$

Pseudokód algoritmu pro ověření zda bod X leží uvnitř trojúhelníku P_0, P_1, P_2 je popsán ve výpisu (3).

```
bool insideTrinagle(Vector X, Vector P0, Vector P1, Vector P2) {
```



Obrázek 9: Odraz vlny o dva polygony



Obrázek 10: Body podezřelé z lomu čela vlny (fialově)

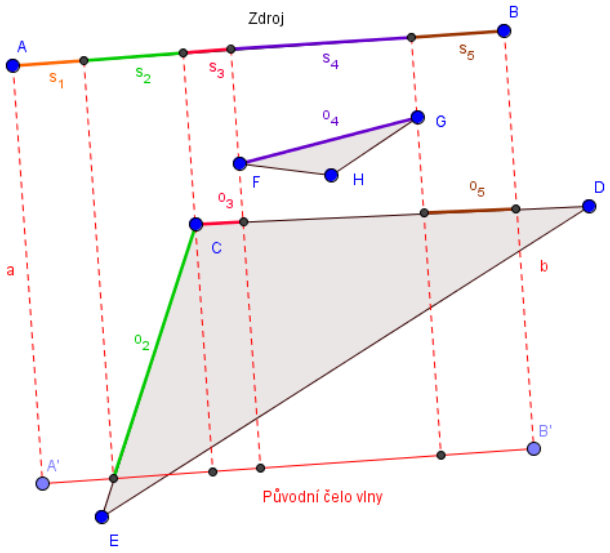
```
    double d = 1/(P0.x*(P1.y-P2.y)+P1.x*(P2.y-P0.y)+P2.x*(P0.y-P1.y));
    double a = (X.x*(P1.y - P2.y) + P1.x*(P2.y - X.y) + P2.x*(X.y - P1.y)) * d;
    double b = (P0.x*(X.y - P2.y) + X.x*(P2.y - P0.y) + P2.x*(p0.y - X.y)) * d;
    return 0 <= a && a <= 1 && 0 <= b && b <= 1 && a + b <= 1;
}
```

Výpis 3: Ověření zda bod leží uvnitř trojúhelníku

4.2 Nalezení bodů lomu čela vlny a odrazových zrcadel

Nyní, když už umíme nalézt všechny body podezřelé z lomu čela vlny, zbývá určit, které body čelo vlny opravdu lámou, a od kterých zrcadel se vlastně budou svazky paprsků odrážet. Jednoduchý způsob, jak toho docílit, je pomocí kolmého průmětu promítnout body podezřelé z lámání čela vlny na zdrojovou úsečku a čelo vlny. Tyto body rozdělí zdrojovou úsečku a čelo vlny na menší úsečky. Každé dvě tyto menší úsečky ohraničují svazek paprsků, které se odrážejí od stejného zrcadla. Zrcadlo nalezneme vysláním paprsku ze středu zdroje tohoto svazku do středu čela vlny svazku, první průsečík po cestě paprsku pak určuje zrcadlo, o které se bude svazek odrážet. Pokud mají dva sousední svazky stejné zrcadlo, můžeme tyto svazky sjednotit. Svazky paprsků vyslané z těchto úseček se odrážejí podle stejného zrcadla. Tato metoda je znázorněna na obrázku 11.

Svazky paprsků, se podobně jako tomu je u paprsků ray tracingu, mohou dále odrážet, proto je potřeba na jednotlivé odražené svazky paprsků celý proces odrážení opět aplikovat rekurzivně, dokud se už svazky nebudou dále odrážet.



Obrázek 11: Lámání čela vlny

S využitím všech vědomostí z této a předchozích kapitol lze naprogramovat jednoduchý polygonal tracer. Příklad čela vlny by vypadal jako na obrázku 12. Všimneme si, že nemáme celou informaci o čelu vlny, protože ztrácíme informaci o difrakci (ohybu) vlny o vrcholy polygonu.

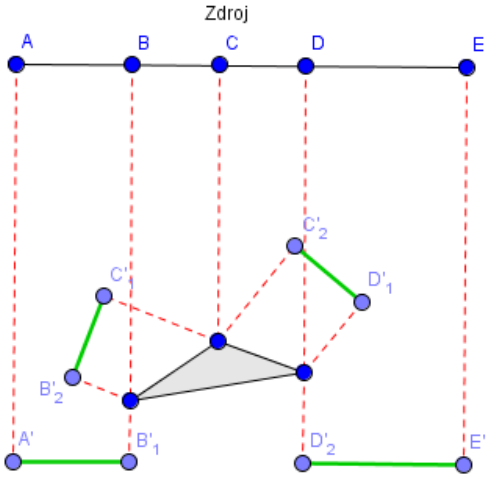
4.3 Difrakce vlny o vrcholy polygonu

Jedním s vlnových fenoménů, které nemodifikovaný ray tracing neřeší je difrakce vlny. Paprsky dopadající na nějaký konvexní vrchol polygonu se odrážejí do všech možných směrů (obr. 14). [9] Do ray tracingu se proto často přidává difrakce paprsků o vrchol tak, že v případě, že paprsek dopadne na vrchol (nebo do jeho těsné blízkosti), vygenerují se z vrcholu nové paprsky [1], které směřují do všech směrů od vrcholu mimo vnitřní oblast polygonu. Protože je jedním z předpokladů použití polygonal tracingu homogenní médium, je rychlost šíření paprsků konstantní. Čelo vlny formované paprsky odražené od konvexního vrcholu polygonu tvoří kruhový oblouk se středem ve vrcholu. Krajní body tohoto oblouku leží na stranách polygonu (obr. 14). Protože nás zajímá jen čelo vlny, můžeme tento oblouk sjednotit s okolními svazky paprsků. Krajní body oblouku pak budou krajními body svazků paprsků lámaných o tento vrchol. Výsledek je vidět na obrázku 13, ze kterého je patrné, že přidání difrakce paprsků o vrchol dává mnohem lepší představu o tvaru čela vlny než jednoduchá implementace polygonal tracingu z obrázku (12).

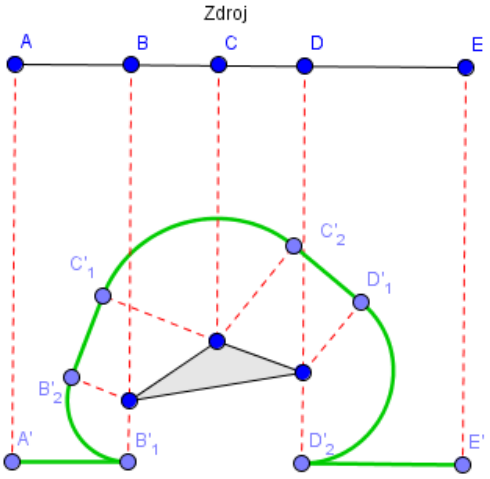
Difrakce paprsků o vrchol však není úplně zadarmo. Svazky difraktovaných paprsků se totiž mohou dále odrážet od dalších polygonů, a tak musíme řešit z cela nový problém.

4.3.1 Nalezení bodů podezřelých z lomu kruhového čela vlny

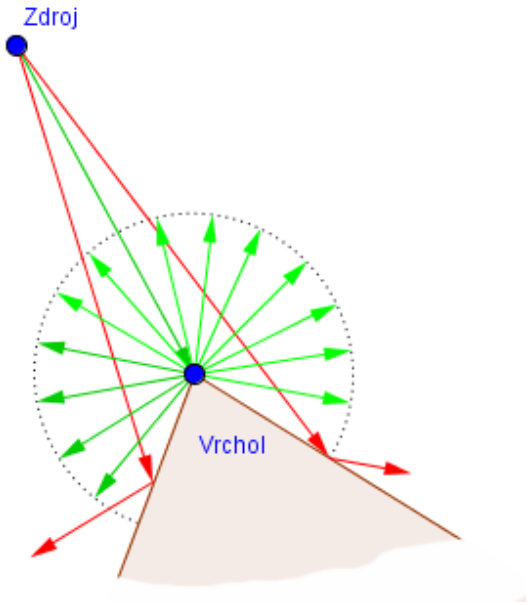
Podobně jako v kapitole 4.1 budeme hledat body, u kterých se dá předpokládat, že se o ně bude kruhové čelo vlny lámat. Mezi tyto bodu budou patřit



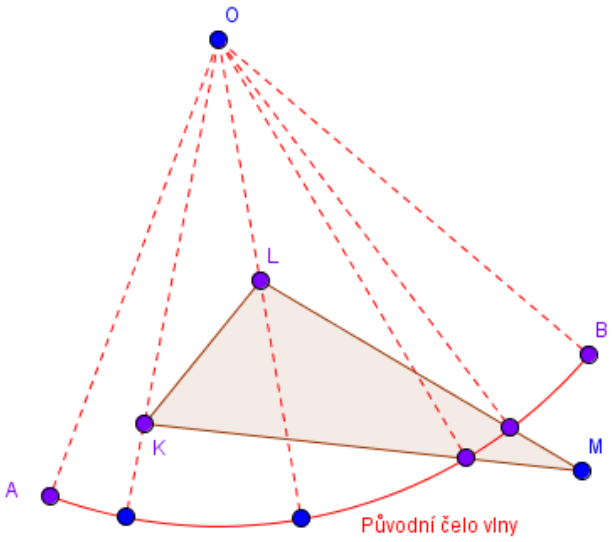
Obrázek 12: Polygonal tracer bez difrakce



Obrázek 13: Polygonal tracer s difrakcí



Obrázek 14: Difrakce paprsků o vrchol polygonu



Obrázek 15: Body podezřelé z lomu oblouku čela vlny

- krajní body čela vlny v čase t,
- průsečíky polygonů s úsečkami definovanými zdrojem paprsků a krajními body kruhového oblouku čela vlny,
- průsečíky kruhového oblouku čela vlny se stranami polygonů
- a vrcholy polygonů uvnitř kruhové výseče definované zdrojem paprsků a čelem vlny.

Tyto body jsou znázorněny na obrázku 15.

Nalezení první množiny je triviální a nalezení množiny z druhého bodu jsme již řešili v kapitole **??**. Pro určení třetí množiny nejprve nalezneme průsečíky přímek stran polygonů s kružnicí. K nalezení průsečíku kružnice k se středem v (m, n) a poloměrem r s přímkou p určenou její obecnou rovnicí využijeme soustavy rovnic (10).

$$\begin{aligned} k: &\quad (x^2 - m)^2 + (y^2 - n) = r^2 \\ p: &\quad ax + by + c = 0 \end{aligned} \tag{10}$$

Ve výpočtu můžeme postupovat tak, že nejprve z rovnice přímky vyjádříme x nebo y, vybereme tu souřadnici, která není nulová nebo blízká nule. Po dosazení do rovnice kružnice obdržíme kvadratickou rovnici, která může mít v R^2 nula, jedno nebo dvě řešení. V případě, že kvadratická rovnice nemá řešení, průsečíky neexistují, jinak dopočítáme druhou souřadnici průsečíků dosazením zpátky do rovnice přímky.

Pokud je $|a| \geq |b|$ dosadíme $x = \frac{-by-c}{a}$ do rovnice kružnice a řešíme kvadratickou rovnici

$$(\frac{-by-c}{a} - m)^2 + (y-n)^2 = r^2.$$

Dosadíme do vzroce pro výpočet kvadratické rovnice

$$y_{1,2} = \frac{2a^2n - 2abm + 2bc \pm \sqrt{(2abm + 2bc - 2a^2n)^2 - 4(a^2+b^2)(a^2m^2 + a^2n^2 - a^2r^2 + 2acm + c^2)}}{2(a^2+b^2)}$$

a za předpokladu, že (a, b) je normovaný normálový vektor přímky p upravíme na

$$y_{1,2} = a^2n - abm + bc \pm \sqrt{(abm + bc - a^2n)^2 - (a^2m^2 + a^2n^2 - a^2r^2 + 2acm + c^2)}.$$

Pro $|a| < |b|$ postupujeme analogicky.

Po nalezení průsečíků zbývá určit, zda průsečíky leží, jak na úsečce strany polygonu, tak na kruhovém oblouku. Ověření, zda průsečík leží na úsečce jsme už řešili v předchozí kapitole. Pro ověření, zda průsečík leží na kruhovém oblouku můžeme využít polární soustavy souřadnic. Nejprve provedeme translaci souřadnic, aby střed kruhového oblouku ležel v počátku kartézských souřadnic, poté převedeme krajní body kruhového oblouku a průsečíku do polární soustavy souřadnic podle převodního vzorce (11).

$$\begin{aligned} r &= \sqrt{x^2 + y^2} \\ \phi &= \arctan \tfrac{y}{x} \end{aligned} \tag{11}$$

Kvůli vlastnostem funkce arkus tangens by však převodní vzorec (11) fungoval pouze na intervalu $\langle 0, \pi/2)$, pro jiné intervaly bychom museli brát v potaz znaménka jednotlivých souřadnic, navíc vztah y/x není definován pro $x = 0$. Z těchto důvodu se často zavádí funkce $\text{atan2}(y, x)$ (12), jejíchž implementace je v dnešní době součástí jádra mnoha programovacích jazyků včetně Javy. [10]

$$\text{atan2}(y,x) = \begin{cases} \arctan \frac{y}{x} & \text{pro} \quad x > 0 \\ \arctan \frac{y}{x} + \pi & \text{pro} \quad x < 0 \wedge y \geq 0 \\ \arctan \frac{y}{x} - \pi & \text{pro} \quad x < 0 \wedge y < 0 \\ \frac{\pi}{2} & \text{pro} \quad x = 0 \wedge y > 0 \\ -\frac{\pi}{2} & \text{pro} \quad x = 0 \wedge y < 0 \end{cases} \tag{12}$$

```
// prevod z kartezskych do polarnich souradnic
Vector toPolarCoordinates(Vector cartesian) {
    double r = sqrt(cartesian.x * cartesian.x + cartesian.y * cartesian.y);
    double fi = atan2(cartesian.y, cartesian.y);
    return new Vector(r, fi);
```

```
}

// prevod z polarnich do kartezskych souradnic
Vector toCartesianCoordinates(Vector polar) {
    double x = polar.x * cos(polar.y);
    double y = polar.x * sin(polar.y);
    return new Vector(x, y);
}
```

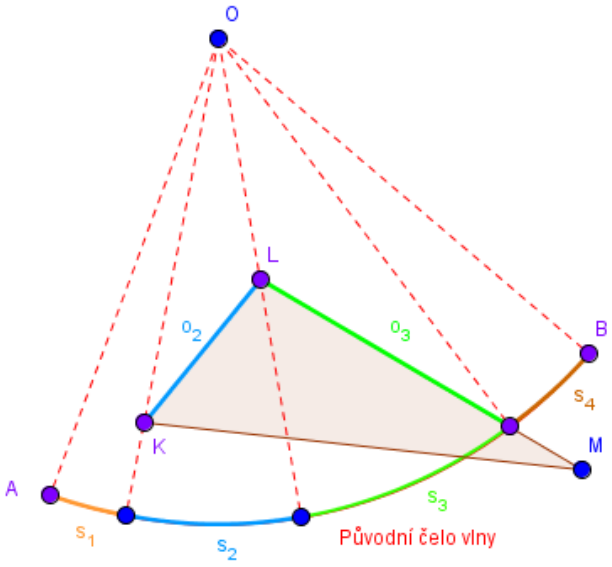
Výpis 4: Převody souřadnic

Na konec stačí ověřit, zda úhel průsečíku ϕ_p leží v intervalu úhlů začátku ϕ_{start} a konce kruhového oblouku ϕ_{end}, zároveň musí být poloměr r_p průsečíku p menší nebo roven poloměru r kruhové výseče, což je vzhledem k tomu, že průsečík leží na kružnici k určitě splněno. Je třeba brát zřetel na případy, kdy je $\phi_{start} > \phi_{end}$, v takovém případě musíme k ϕ_{end} a ϕ_p přičíst 2π.

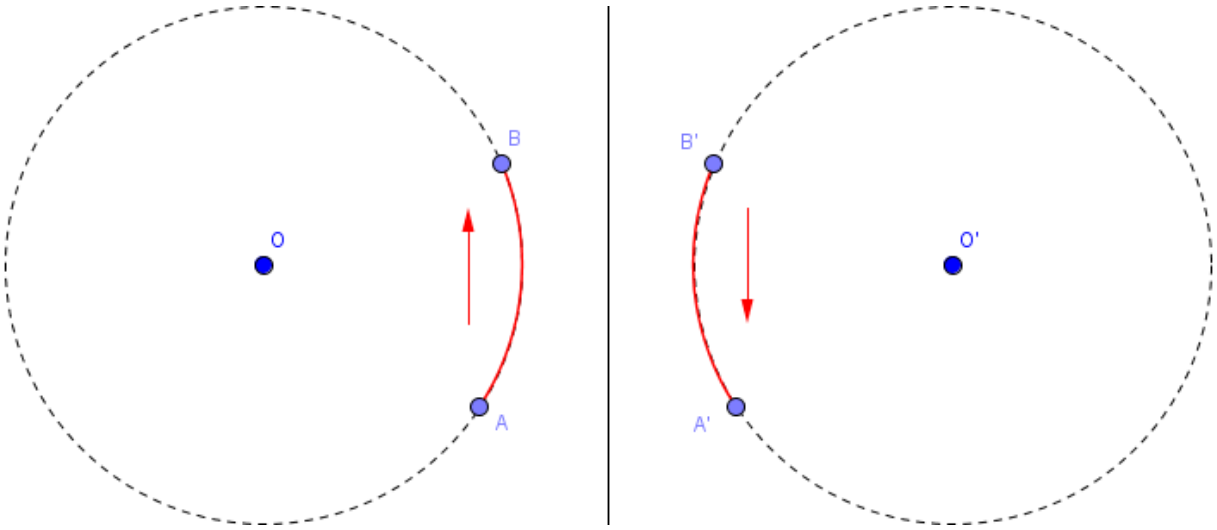
Pro nalezení vrcholů uvnitř kruhové výseče můžeme opět využít převodu kartézských souřadnic na souřadnice polární, tentokrát už ale musíme kromě úhlu ϕ_p ležícího v intervalu $\langle \phi_{start}, \phi_{end} \rangle$ ověřovat i zda je $r_p < r$.

4.3.2 Nalezení bodů lomu kruhového čela vlny a odrazových zrcadel

Podobně jako v kapitole 4.2 nejprve nalezneme projekce bodů podezřelých z lomu čela vlny na kruhový oblouk, čímž ji rozdělíme na několik menších oblouků, zrcadlo odrazu tohoto oblouku určíme tak, že nalezneme průsečíky paprsku vyslaného ze středu kružnice do středu tohoto oblouku (obr. 16).



Obrázek 16: Lámání oblouku čela vlny



Obrázek 17: Změna orientace oblouku při zrcadlení

Po nalezení všech částí oblouku a jejich odrazových úseček, můžeme jednotlivé části oblouku zrcadlit. Zrcadlíme vždy dva krajní paprsky a střed kruhového oblouku. Při zrcadlení dochází ke změně orientace kruhového oblouku Jestli původní kruhový oblouk vedl od A do B proti směru hodinových ručiček, u odraženého oblouku vede A' do B' po směru hodinových ručiček (obr. 17). Toto můžeme ošetřit uchováváním stavu orientace nebo můžeme jednoduše u odražené části čela vlny prohodit A' a B'.

4.4 Reprezentace částí čela vlny v algoritmu

Pro část čela vlny, která je úsečkou stačí uchovávat krajní paprsky vlny. Pro část vlny, která je kruhovým obloukem uchováveme i střed kruhového oblouku a orientaci. Pokud to zvolený programovací jazyk umožňuje, můžeme využít dědičnosti tříd a umístit stejné části kódu do společného předka.

```
// spolecny predek
class WaveSegment {
    // krajni paprsky
    Ray startRay;
    Ray endRay;
    // nasleduji obecne metody pro praci s casti cela vlny
}
// useckova cast cela vlny
class LineSegment extends WaveSegment {
    // nasleduji metody pro praci s useckovou casti cela vlny
}
// kruhova cast cela vlny
```

```
class ArcSegment extends WaveSegment {
    Vector origin; // stred kruznice
    boolean orientation; // orientace kladna = true, zaporna = false
    // nasleduji metody pro praci s obloukovou casti cela vlny
}
```

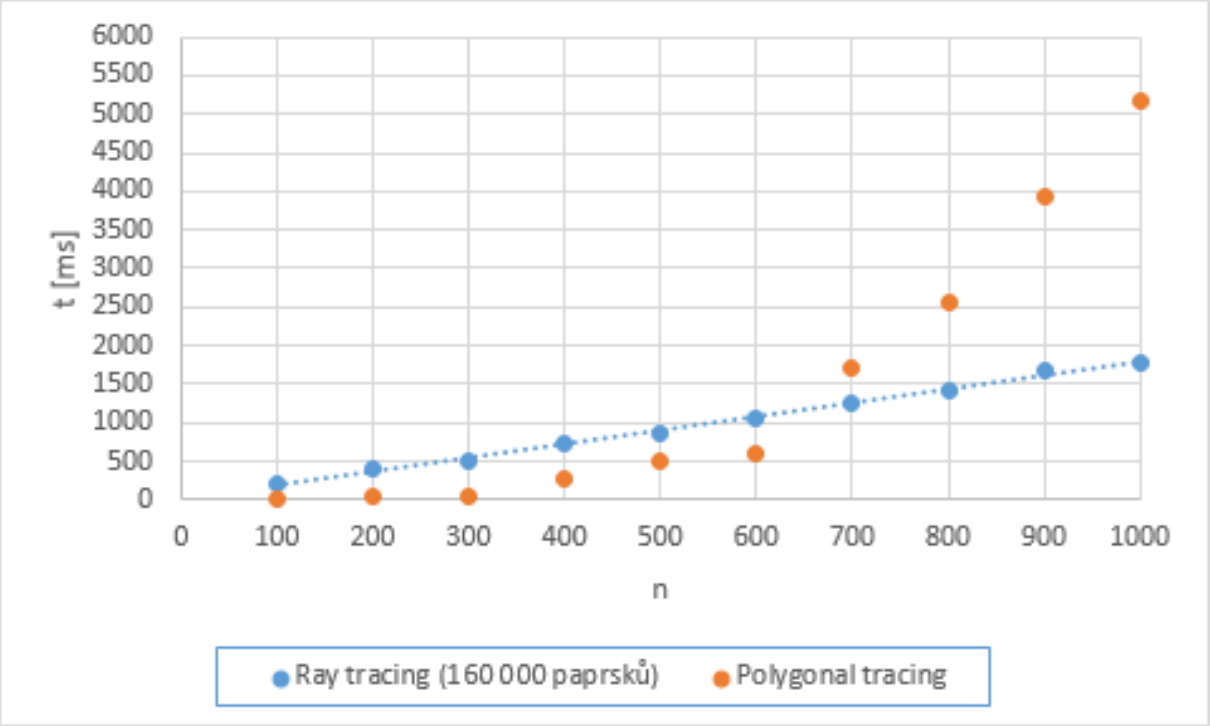
Výpis 5: Hierarchie tříd čela vlny

5 Porovnání metod

Pro porovnání ray tracingu a polygonal tracingu byl naprogramován program v jazyce Java, jenž je společně s pseudokódem a zdrojovými kódy přílohou této bakalářské práce. Jak implementace ray tracingu, tak implementace polygonal tracingu neřeší difrakci paprsku o vrcholy polygonů.

5.1 Porovnání rychlosti metod

V tabulkách 1 a 2 jsou výsledky měření výpočetního času ray tracingu a polygonal tracingu, v závislosti na počtu vrcholů trhlin v cestě vlny. Výsledky měření jsou graficky znázorněny na obrázku 18. Při pokusných měřeních byly paprsky vysílány proti řadám trhlin ve tvaru čtverců, které měly dohromady n vrcholů (obr. 19), při každém pokusu se umístění čtverců mírně měnilo a pro každé n byla měření obou metod pětkrát zopakována. V případech, kdy byl počet trhlin malý byla naše implementace polygonal tracingu výrazně rychlejší než ray tracing. Pro $n = 100$ byl dokonce průměrně šestnáctkrát rychlejší, pro $n = 200$ a $n = 300$ přibližně osmkrát rychlejší, ale pro $n = 1000$ byl už průměrně třikrát pomalejší než ray tracing. Což je očekávatelný výsledek. Ray tracing v každém případě vyslal 160 000 paprsků a tento počet zůstával po celý cyklus výpočtu konstantní. Počet paprsků vyslaných polygonal tracingem závisel na počtu bodů podezřelých z lámání čela vlny. V nejhorším případě rozdělí n vrcholů v cestě čelo vlny na $n + 1$ částí a pro každou část čela vlny bude potřeba vyslat tři paprsky, jeden pro určení odrazového zrcadla a dva tvořící krajní paprsky čela vlny. Každá z těchto částí čela vlny se poté může znovu odrážet a lámat. To znamená, že počet paprsků během výpočtu roste z každým odrazem svazku paprsků a od určitého počtu trhlin začne polygonal tracing vysílat více paprsků než ray tracing a tím pádem začíná být výrazně pomalejší.



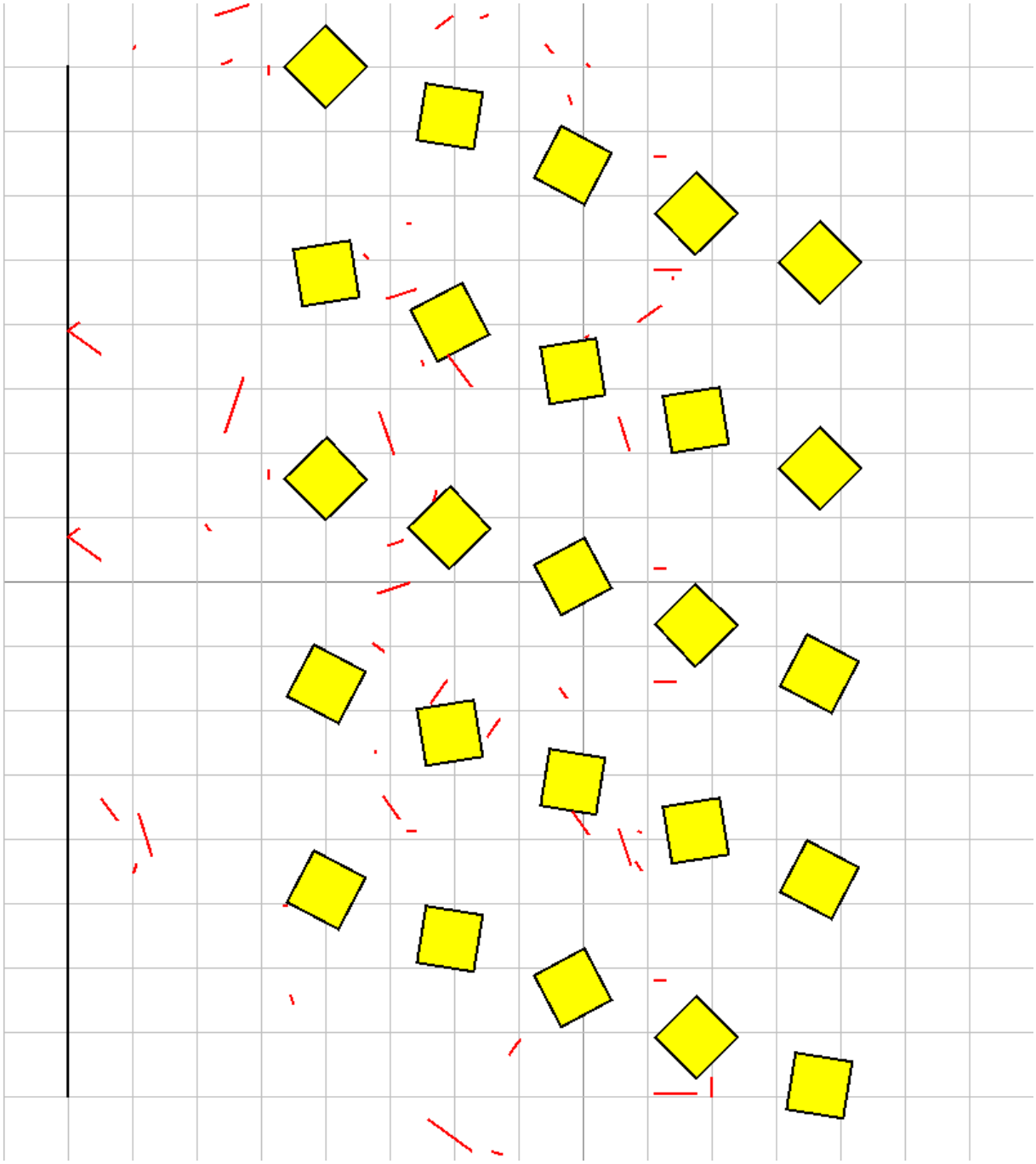
Obrázek 18: Změna orientace oblouku při zrcadlení

Tabulka 1: Měření výpočetního času [ms] ray tracingu pro n vrcholů

n	100	200	300	400	500	600	700	800	900	1000
Měření 1	174	383	663	718	851	863	1202	1608	1831	1999
Měření 2	191	406	437	693	906	1028	1176	1308	1790	1801
Měření 3	196	396	466	737	781	1161	1186	1406	1552	1769
Měření 4	237	396	517	805	825	1079	1347	1354	1691	1692
Měření 5	175	416	467	669	915	1089	1330	1425	1556	1573
Průměr	194,6	399,4	510	724,4	855,6	1044	1248,2	1420,2	1684	1766,8

Tabulka 2: Měření výpočetního času [ms] polygonal tracingu pro n vrcholů

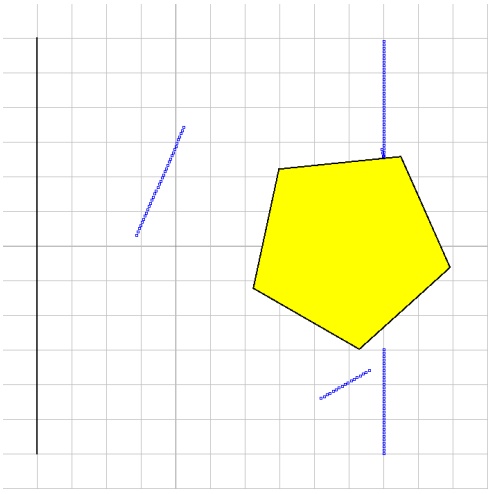
n	100	200	300	400	500	600	700	800	900	100
Měření 1	9	52	57	398	562	1048	2012	3883	3418	5867
Měření 2	10	47	52	219	431	458	1058	2215	3755	4360
Měření 3	15	46	48	212	550	517	2241	1590	3168	5748
Měření 4	8	60	52	243	583	582	1402	2390	4420	3991
Měření 5	13	46	51	215	376	461	1876	2669	4947	5929
Průměr	11	50,2	52	257,4	500,4	613,2	1717,8	2549,4	3941,6	5179



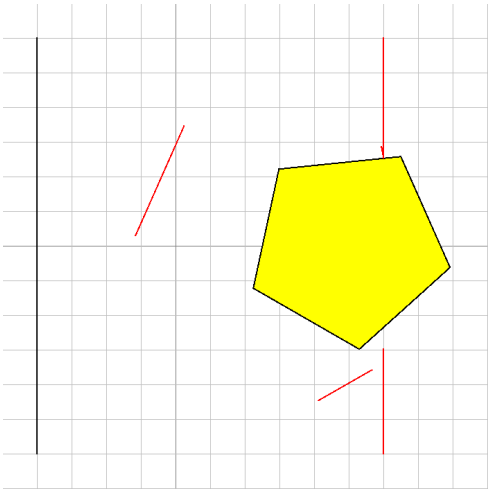
Obrázek 19: Testovací scénář se 100 vrcholy

5.2 Vizuální porovnání

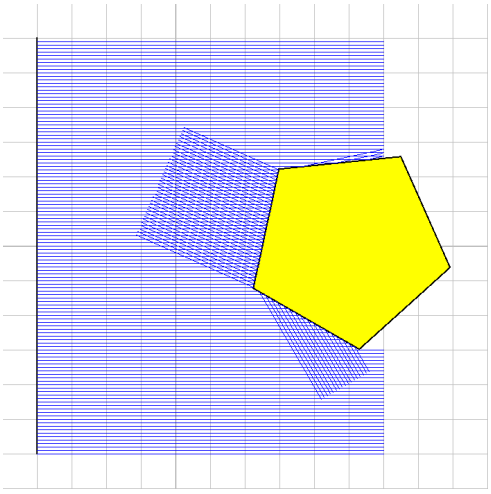
Následují ukázky ray tracingu a polygonal tracingu při modelování vlny odrážené o pravidelný pětiúhelník a pěticípou hvězdu.



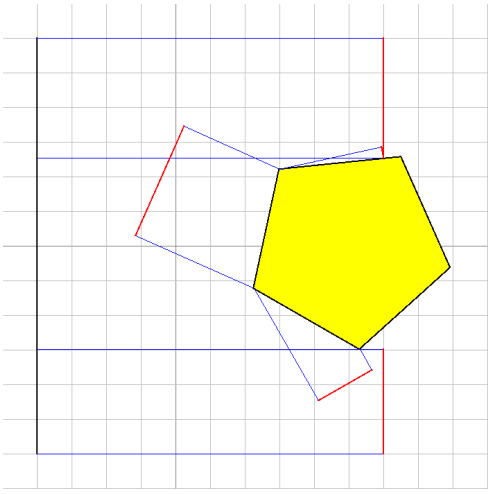
Obrázek 20: Ray tracing pětiúhelníku



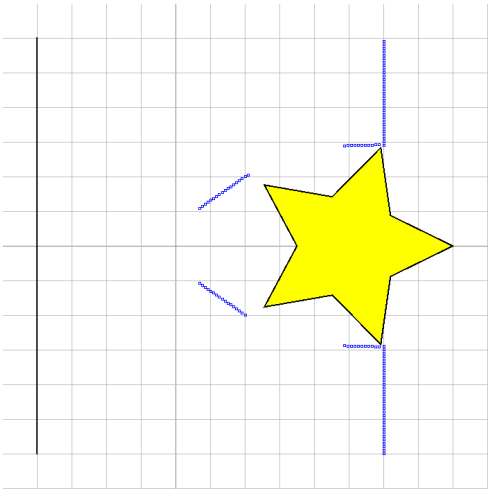
Obrázek 21: Polygonal tracing pětiúhleníku



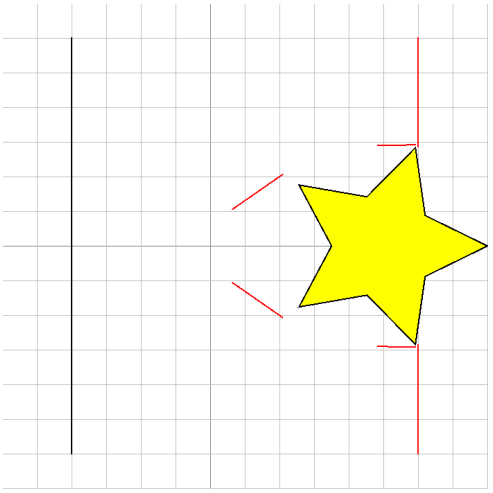
Obrázek 22: Ray tracing pětiúhelníku celé paprsky



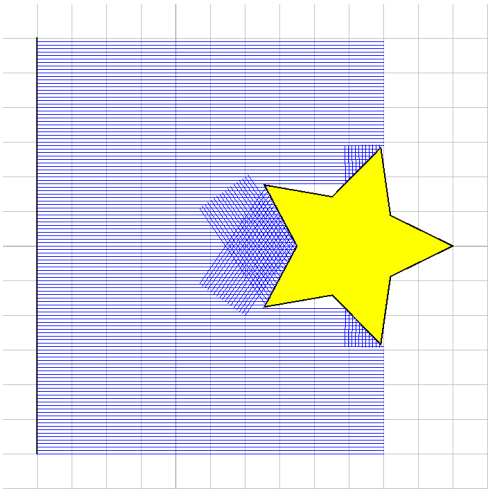
Obrázek 23: Polygonal tracing pětiúhleníku celé paprsky



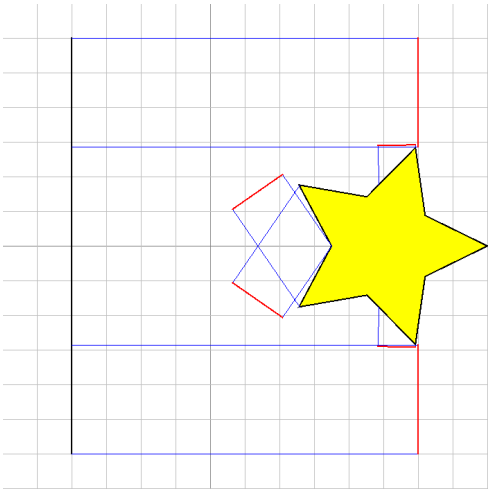
Obrázek 24: Ray tracing hvězdy



Obrázek 25: Polygonal tracing hvězdy



Obrázek 26: Ray tracing hvězdy celé paprsky



Obrázek 27: Polygonal tracing hvězdy celé paprsky

6 Závěr

Cílem práce bylo nalézt takovou modifikaci ray tracingu, která by byla optimalizována pro specifické potřeby ultrazvukové defektoskopie materiálů, kde vysílač a trhliny v materiálu jsou polygony a zároveň by byla rychlejší než její původní verze.

Na úvod práce jsme se seznámili s matematickými základy modelování čela vlny včetně vlnové a eikonálové rovnice. Poté byla představena metoda ray tracing, její teoretické základy a postup implementace. Z ray tracingu a některých vlastností odražených rovnoběžných paprsků byla odvozena metoda polygonal tracing, která byla rovněž implementována. V této bakalářské práci byla také navržena a teoreticky vypracována difrakce paprsků o vrcholy polygonů pro polygonal tracing, není však součástí implementace metody.

Při porovnávání obou algoritmů jsme zjistili, že efektivita polygonal tracingu přímo závisí na počtu trhlin v materiálu. Vyvinutý algoritmus polygonal tracing je skutečně mnohem rychlejší než ray tracing, ale pouze v případech, kdy je počet trhlin dostatečně nízký. Pro velké množství trhlin je naopak mnohem pomalejší. Je nasnadě se domnívat, že další rozvoj této metody by mohl přinést ještě lepší výsledky.

Literatura

[1] ENGQUIST Björn, RUNBORG Olof. Computational high frequency wave propagation. In *Acta Numerica 2003*. Volume 12. Cambridge University Press, 2003. Kapitola 3, s. 181-266.

[2] FEYNMAN Richard. Lectures in Physics. Sound. The wave equation. In Lectures in Physics. Volume 1. Caltech, 1963, 2006, 2013. [online]. Dostupné z: http://feynmanlectures.caltech.edu/I_47.html.

[3] HAINES Eric. Essential Ray Tracing Algorithms. In *An introduction to ray tracing*. Volumne 3. San Diego: Academic Press, 1990. Kapitola 1 s. 35, kapitola 4, s. 65.

[4] KOZUBEK Tomáš, BRZOBOHATÝ Tomáš, JAROŠOVÁ Marta, HAPLA Václav, MARKOPOULOS Alexandros. Lineární algebra s Matlabem. [online]. Dostupné z: http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/linearni_algebra_s_matlabem.pdf. 2012.

[5] IEEE Standards Association. IEEE standard for binary floating-point arithmetic. IEEE Report (New York), 1985. ANSI/IEEE Std 754-1985

[6] SMITS Brian. Efficiency issues of ray tracing. In *Journal of Graphics Tools*. 3(2):1-14, 1998.

[7] WILLIAMS Amy, BARRUS Steve, MORLEY R. Keith, SHIRLEY Peter. An Effecient and Robust Ray-Box Intersection Algorithm. University of Utah. [online]. Dostupné z http://www.cs.utah.edu/ awilliam/box/box.pdf. (nedatováno).

[8] HAINES Eric. Point in Polygon Strategies. In *Graphics Gems IV*. Editoval HECKBERT, Paul. Academic Press. s. 24-46. [online]. Dostupné z: http://erich.realtimerendering.com/ptinpoly/. 1994.

[9] JIN Shi. Computational high frequency wave diffraction by a corner via the Liouville equation and geometric theory of diffraction. In *Kinetic and Related Models*. Volume 4. S. 295 - 316. 2011.

[10] Math (Java Platform SE 7). [online]. Poslední revize 11.1.2016. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html#atan2(double,%20double).

A Pseudokód ray tracingu

```
// trida definujici paprsek
class Ray {
    Vector* travelPoints; // seznam bodu kterymi paprsek prochazi
    Vector direction; // smerovy vektor paprsku
    Vector inv_direction; // prevraceny smerovy vektor

    // Funkce pro aktualizaci paprsku.
    // speed - rychlost sireni paprsku v prostredi
    // time - cas sireni paprsku
    void update(float speed, float time) {
        // ziskej posledni pozici paprsku
        Vector lastPoint = travelPoints.last();
        // vypocitej novou pozici paprsku
        Vector newPoint = lastPoint * speed * time;
        // pridej novou pozici paprsku do seznamu
        travelPoints.add(newPoint);
    }
}
// struktura uchovavajici informace o pruseciku
struct Intersection {
    Vector point; // prusecik
    Mirror mirror; // zrcadlo
}
// rozhrani zrcadla
Interface Mirror {
    // funkce overujici existenci pruseciku paprsku se zrcadlem
    Intersection intersects(Ray ray);
    // funkce overujici zda paprsek prochazi bounding-boxem
    boolean boundingBoxIntersects(Ray ray);
    // funkce zrcadlici bod o zrcadlo
    Vector doMirror(Vector point);
}
// Hlavni funkce a vstupni bod programu.
void rayTracing() {
    Ray* rays; // set paprsku
    Mirror* mirrors; // set zrcadel (trhlin)
    float speed; // rychlost sireni paprsku v prostredi
```

```
    float time; // cas sireni paprsku
    for each ray in rays {
        ray.update(speed, time); // aktualizuj paprsek
        trace(ray, mirrors, speed, time);
    }
}
// Vlastni sledovani paprsku.
// ray - sledovany paprsek
// mirrors - set zrcadel (trhlin)
// speed - rychlost sireni paprsku v prostredi
// time - cas
void trace(Ray ray, Mirror* mirrors, float speed, float time) {
    Intersection* intersections; // set pruseciku

    // pro vsechna zrcadla
    for each mirror in mirrors {
        // over zda prapsek protina bounding-box
        if ray intersects bounding-box of mirror {
            Intersection i = findIntersection(ray, mirror);
            // pokud prusecik existuje pridej jej do setu
            if i exists {
                intersections.add(i);
            }
        }
    }
    // jestli byl nalezen nejaky prusecik
    if intersections not empty {
        // nalezni prvni prusecik
        Intersection firstIntersection = firstIntersection(ray, intersections);
        // odraz paprsek
        Ray mirroredRay = mirror(ray, firstIntersection);
        // sleduj odrazeny paprsek rekurzivne
        trace(mirroredRay);
    }
}
// Funkce, ktera provede zrcadleni paprsku.
// ray - paprsek
// intersection - prusecik
void mirror(Ray ray, Intersection intersection) {
```

```
    Vector lastPoint = ray.travelPoints.last();
    Vector directionVector = ray.direction;
    Vector intersectionPoint = intersection.point;
    Mirror mirror = intersection.mirror;
    // zrcadli posledni bod paprsku
    Vector newLastPoint = mirror.doMirror(lastPoint);
    // zrcadli smerovy vektor paprsku
    Vector newDirection = mirror.doMirror();
    // odstran puvodni posledni bod a pridej do paprsku nove body
    ray.travelPoints.removeLast();
    ray.travelPoints.add(intersectionPoint);
    ray.travelPoints.add(newLastPoint)
    // zmen smerovy vektor paprsku
    ray.directionVector = newDirection;
}
```

Výpis 6: Pseudokód ray tracingu

B Pseudokód polygonal tracingu

```
// Hlavni funkce a vstupni bod programu
void polygonalTracing() {
    WaveLineSegment waveSegment; // celo vlny
    Mirror* mirrors; // set zrcadel (trhlin)
    float speed; // rychlost sireni paprsku v prostredi
    float time; // cas sireni paprsku

    waveSegment.update(speed, time); // aktualizuj vlnu (jeji paprsky)

    trace(waveSegment, mirrors); // sleduj vlnu
}

// Sleduj celo vlny.
// waveSegment - cast cela vlny
// mirrors - set zrcadel (trhlin)
void trace(WaveLineSegment waveSegment, Mirror* mirrors) {
    // najdi body podezrele z lomu vlny
    Vector* breakPoints = waveSegment.findBreakPoints(mirrors);

    // rozdel vlnu podle bodu podezrelych z lomu cela vlny
    WaveLineSegment* splittedWaves = waveSemgnet.split(breakPoints);

    // projdi vsechny nove casti vlny
    for each splittedWaveSegment in splittedWaveSegemnts {
        // odraz vlnu
        bool mirrored = splittedWaveSegment.mirror();
        if (mirrored) {
            // pokud byla vlna odrazena, rekurzivne ji sleduj
            trace(splittedWaveSegment, mirrors, speed, time);
        }
    }
}

// trida useckoveho cela vlny
class WaveLineSegment {

    Ray startRay; // paprsek na zacatku svazku
```

```
Ray endRay; // paprsek na konci svazku

Interesction intersection; // zrcadlo

// Najde a vrati vsechny body podezrele z lomu cela vlny.
Vector findBreakPoints(Mirror* mirrors);

// Rozdeli vlnu podle bodu podezrelych z lomu cela vlny.
// breakPoints - body podezrele z lomu cela vlny serazeny podle vzdalenosti
    jejich kolmeho prumetu na celo vlny od jejiho zacatku
WaveLineSegment* split(Vector* breakPoints) {
    WaveLineSegment* waveSegments = Empty list;

    // projdi body lomy v poradi vzdalenosti jejich kolmeho prumetu na celo
        vlny od zacatku cela
    for (int i = 0; i < breakPoints.size() - 1; i++) {
        Ray aStartRay = getNewRay(breakPoints[i]); // paprsek na zacatku
            svazku A
        Ray aEndRay = getNewRay(breakPoints[i+1]); // paprsek na konci
            svazku A
        Ray aTestRay = getTestRay(breakPoints[i], breakPoints[i+1]); //
            testovaci paprsek svazku A

        // najdi prvni pruseciku svazku A
        Intersection aIntersection = findFirstIntersection(aTestRay, mirrors
            );

        // prochazej vedlejsi paprsky dokud nenarazis na prpsek s jinym
            zrcadlem
        for (int j = i + 1; j < breakPoints.size() - 1; j++) {
            Ray bStartRay = getNewRay(breakPoints[j]); // paprsek na zacatku
                svazku B
            Ray bEndRay = getNewRay(breakPoints[j+1]); // paprsek na konci
                svazku B
            Ray bTestRay = getTestRay(breakPoints[j], breakPoints[j+1]); //
                testovaci paprsek svazku B

            // najdi prvni prusecik svazku B
```

```
            Intersection aIntersection = findFirstIntersection(bTestRay,
                mirrors);

            // over zda jsou pruseciky na stejnych zrcadlech
            if (aIntersection.mirror == bIntersection.mirror) {
                // svazky paprsku se odrazeji o stejna zrcadla, spojujeme
                aEndRay = bEndRay;
            } else {
                // svazky paprsku se odrazeji o jina zrcadla, nespojujeme
                i = j - 1;
                break;
            }
        }

        // vytvor novy segment cela vlny
        WaveLineSegment newSegment = new WaveLineSegment(aStartRay, aEndRay,
             aIntersection);

        // pridej celo vlny do seznamu
        waveSegments.add(aIntersection);

        i++
    }

    // vrat rozdelene casti cela
    return waveSegments;
}

// Odrazi paprsek o zrcadlo, na zaklade pruseciku.
// vraci true pokud dojde k zrcadleni
bool mirror() {
    // pokdu existuje prusecik se zrcadlem
    if (this.intersection != null) {
        Mirror mirror = intersection.mirror;

        // najdi pruseciky paprsku se zrcadlem
        Intersection startIntersection = findIntersection(this.startRay,
            mirror);
```

```
            Intersection endIntersection = findIntersection(this.endRay, mirror)
                ;

            // zrcadli paprsky
            mirror(this.startRay, startIntersection);
            mirror(this.endRay, endIntersection);

            return true;
        }

        return false;
    }

    // Ziska novy paprek prochazejici break pointem.
    // breakPoint - bod podezrely z lomu cela vlny
    Ray getNewRay(Vector breakPoint);

    // ziska testovaci paprsek pro nalezeni pruseciku se zrcadlem
    // breakPoint1 - bod podezrely z bodu cela vlny
    // breakPoint2 - bod podezrely z bodu cela vlny
    Ray getTestRay(Vector breakPoint1, Vector breakPoint2);
}
```

Výpis 7: Pseudokód polygonal tracingu

C Ukázkový program

Součástí této bakalářské práce je i ukázkový program s implementací ray tracingu a polygonal tracingu v jazyce Java. Program se společně se zdrojovými kódy nachází na CD, které je přiloženo k této práci.

C.1 Spuštění programu

Pro spuštění programu je potřeba mít nainstalovanou Javu ve verzi 1.6 (`https://java.com/en/download/`). Aplikace se spouští přes příkazový řádek příkazem `java -jar PolygonalTracing.jar` z adresáře umístění programu.

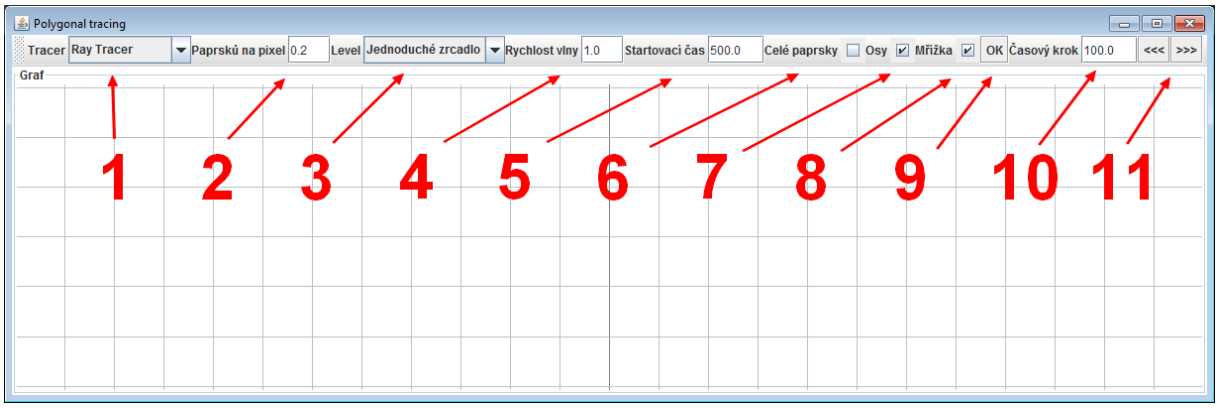
C.2 Sestavení programu ze zdrojových kódů

Pro sestavení programu ze zdrojových kódu je potřeba mít nainstalovaný Apache Maven ve verzi alespoň 2.2.1 (`https://maven.apache.org/download.cgi`). Sestavení probíhá přes příkazový řádek a spouští se příkazem `mvn clean package` z kořenového adresáře zdrojových kódů.

C.3 Návod k použití

Na obrázku 28 je obrazovka s jednoduchým ovládacím panelem:

1. Výběr metody modelování vlny. Ray tracing nebo polygonal tracing.
2. Hustota vyslaných paprsků ray tracingu na jeden pixel.
3. Volba testovacího scénáře.
4. Rychlost vlny c. Desetinné číslo.
5. Startovací čas t_0 modelování čela vlny. Desetinné číslo.
6. Zatrhávací políčko určující, zda se má zobrazovat celá trajektorie paprsku.
7. Zatrhávací políčko určující, zda se mají zobrazovat souřadnicové osy.
8. Zatrhávací políčko určující, zda se má zobrazovat souřadnicová mřížka.
9. Tlačítko pro potvrzení nastavení a spuštění modelování.
10. Výběr časového kroku Δt.
11. Tlačítka posunující čas o Δt zpět nebo vpřed. Desetinné číslo.



Obrázek 28: Kontrolní panel programu