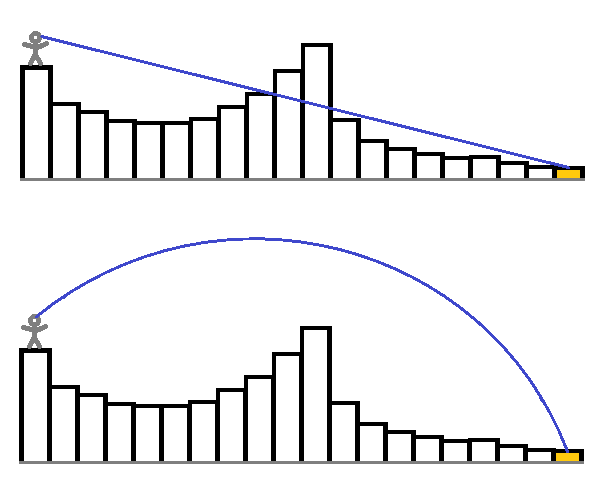
Syntax a význam programu „Throwshed“

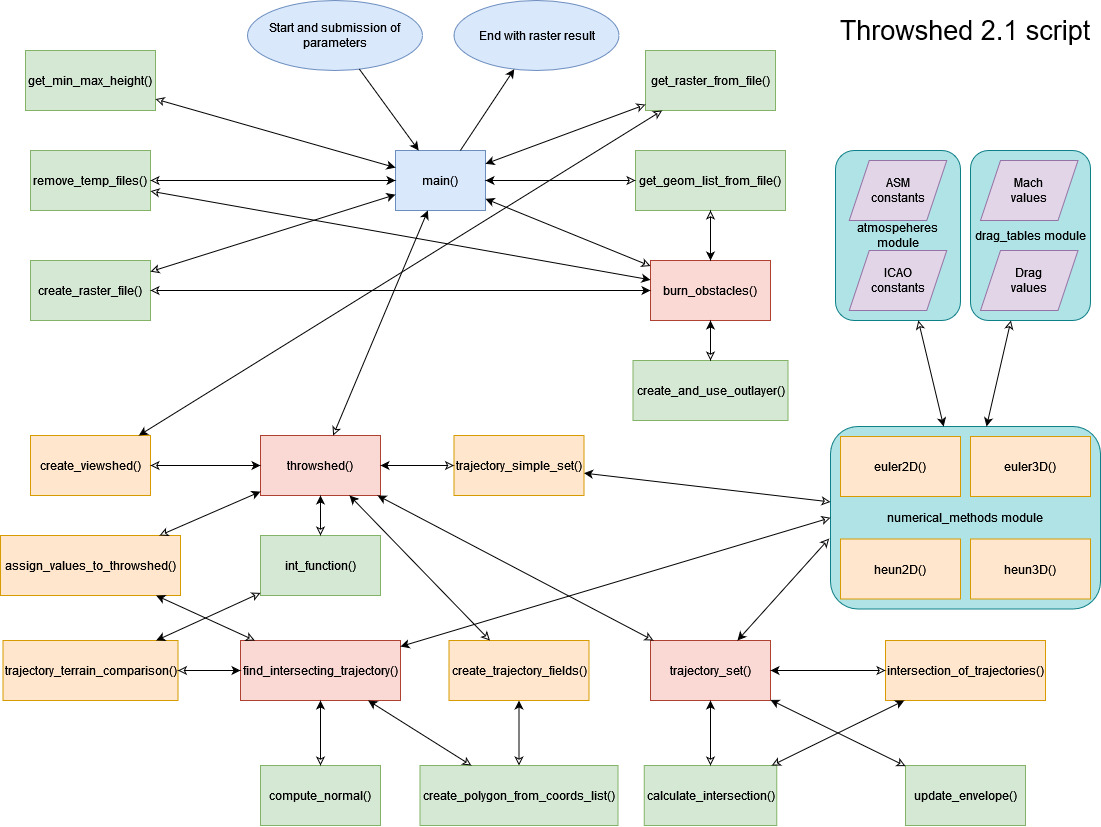
Úvod

Program „Throwshed“ je vyvíjanou priestorovou analýzou, ktorej účelom je pre vybrané miesto výstrelu zhodnotiť prístupnosť okolitého terénu, resp. povrchu pre konkrétny typ vystreleného projektilu. Analýza je veľmi podobná analýze viditeľnosti, pretože tiež využíva vybranú georeferencovanú pozíciu (pozorovateľa, resp. strelca), digitálny model reliéfu (DMR) pre reprezentáciu reálneho terénu okolia a krivky, ktorými sú spájané pozícia s jednotlivými bunkami rastra, aby boli hodnotené, či nepretínajú terén skôr ako v cieľovej bunke, čo rozhodne o viditeľnosti, resp. prístupnosti danej bunky. Rozdielom analýz je, že zatiaľ čo v analýze viditeľnosti sú používané analyticky vypočítané línie pohľadu k bunkám, v analýze „throwshedu“ sú generované balistické krivky numerickými metódami. Obrázok 1 naznačuje tento rozdiel na terénnom profile.



Obr. 1: Prístupnosť cieľovej bunky vo „viewshede“ (hore) a „throwshede“ (dole)

Táto práca detailne opisuje skript analýzy „throwshed“, ktorý bol naprogramovaný v jazyku Python. Tento program zatiaľ nedisponuje vlastným grafickým rozhraním, ani nie je súčasťou iného softvéru. Možno ho spúšťať priamo v editoroch príslušného jazyka ako skript vo formáte \*.py, ktorý využíva aj funkcie, konštanty a hodnoty z pridružených modulov tiež vo formáte \*.py. Všetky súbory programu sú súčasťou verzionovaného repozitára postaveného na systéme Git, prístupného na stránke vývojárskej platformy GitHub, <https://github.com/Tadeo98/throwshed2>. Celá štruktúra vzájomnej komunikácie vlastných funkcií všektých súborov programu je znázornená na obrázku č. 2.



Obr. 2: Schéma vzájomnej komunikácie funkcií   
(legenda: elipsa – spustenie/ukončenie programu, obdĺžnik – funkcia, obdĺžnik bez rohov – modul, kosoštvorec – séria konštánt/hodnôt, obojsmerná šípka – volanie funkcie, kde plná šípka je na strane volajúcej funkcie a prázdna šípka na strane volanej funkcie, modrá – kostra programu, červená – najvyťaženejšie funkcie, oranžová – volané funkcie volajúce iné funkcie, zelená – len volané funkcie)

Nasledujúce kapitoly budú zamerané na opis príslušného skriptu a modulov s podrobným popisom funkcií a konštánt v podkapitolách.

1 Skript throwshed2.py

Skript po spustení definuje a ukladá zadané vstupné hodnoty a nastavenia, aby boli hneď odovzdané hlavnej funkcii *main()*, ktorá je srdcom celého programu. Parametre sú v skratke popísané pri definovaní.

Zadefinované sú cesty k rastru vo formáte \*.tif obsahujúcemu DMR, na ktorom bude analýza vykonaná. Raster môže mať akékoľvek rozlíšenie. Podmienkou je, aby bol riadne georeferencovaný a dimenzovaný v jednotkách dĺžky [m]. Nasleduje určenie cesty k bodovej vrstvy vo formáte \*.shp, ktorá môže obsahovať jeden alebo viacero bodov s polohovými súradnicami (2D) v rámci územia prislúchajúcom DMR. V rámci DMR sa musia nachádzať aj línie, ktoré predstavujú umelú pozdĺžnu prekážku. Ak má byť rešpektovaná, je nutné zadať cestu k líniovej vrstve vo formáte \*.shp. Dôležitá je aj destinácia uloženia a názov výsledku analýzy, ktorý bude vytvorený ako georeferencovaný raster vo formáte \*.tif v rovnakom rozlíšení ako je vstupné DMR.

Hodnoty prislúchajúce nastaveniam určujú aký mód analýzy má byť použitý – jednoduchý „throwshed“ zobrazujúci územie v rámci bezpečnostnej zóny alebo klasický znázorňujúci už aj zákryty v rámci bezpečnostnej zóny. Nasleduje možnosť využitia analýzy viditeľnosti, ktorou je výsledok analýzy „throwshed“ orezaný. Nechýba ani možnosť využitia spomínanej líniovej vrstvy pre vpálenie línií do DMR ako prekážok, čím vznikne digitálny model povrchu (DMP). Dôležité je číslo pásma rastra, z ktorého sú výšky DMR čerpané (zvyčajne rovné 1). Interpoláciu využívanú pri rôznych procesoch v programe možno zvoliť ako metódu najbližšieho suseda alebo lineárnu. Program ďalej disponuje možnosťou zvolenia súčtového (kumulatívneho) výsledku, v ktorom pri viacerých bodoch (miestach výstrelu) môžu bunky nadobúdať namiesto binárnych 0 a 1 celkový počet prekrytov individuálnych výsledkov „throwshedu“, teda že 0 – n (n = počet bodov) strelcov dokáže zasiahnuť konkrétnu bunku. Hodnota EPSG definuje súradnicový systém výsledného rastra ako aj medzivýsledných dočasných súborov. Tiež je nutné vybrať medzi modelmi atmosféry, s ktorými uvažujú výpočty funkcií generujúcich zoznamy súradníc bodov naprieč balistickými trajektóriami. Nasleduje výber jednej z numerických metód výpočtu trajektórií a nakoniec zostáva určiť, či neskôr zadaný inkrement (krok) trajektórie zamýšľaný v horizontálnom alebo šikmom smere.

Nakoniec fyzikálne parametre v jednotkách sústavy SI obsahujú počiatočnú výšku výstrelu nad terénom/povrchom, minimálny a maximálny uhol výstrelu určujúci interval, v ktorom je generovaná sieť trajektórií (základný predpoklad generovania trajektórií, ktoré pretínajú všetky možné bunky rastra DMR: všetky vstupné parametre sú nemenné, mení sa len uhol výstrelu). Tiež treba zadať veľkosť tiažového zrýchlenia pre danú oblasť, počiatočnú (úsťovú) rýchlosť projektilu pri výstrele, teplotu a hustotu vzduchu v mieste výstrelu, hodnotu aerodynamického koeficientu ako konštantu dátového typu integer alebo float (postačuje pri podzvukovej rýchlosti projektilu), alebo ako funkciu odporu k prislúchajúcemu číslu mach dátového typu list (treba definovať hodnoty) alebo string, teda reťazec podľa katalogizovaných funkcií odporu v module *drag\_tables.py*, ktorá bude vybraná pre použitie. Definícia pokračuje priemerom a hmotnosťou projektilu, veľkosťou kroku trajektórie, výškou pozorovateľa a cieľa pre prípad použitia analýzy viditeľnosti a končí definíciou výšky a šírky prekážok, ktoré reprezentujú línie.

1.1 Funkcia main()

Celý program disponuje niekoľkými opatreniami na vyvarovanie sa nesprávneho chodu algoritmov a upozornenia používateľa na opravu zadaných hodnôt. Hlavná funkcia sa uistí, či minimálny uhol výstrelu neprevyšuje maximálny. Nasleduje definícia globálnych premenných, do ktorých sú ukladané zadané aj odvodené hodnoty, aby mohli byť tieto používané naprieč všetkými funkciami. Premenné, ktorých využitie sa vyskytuje len v jednej alebo dvoch funkciách, sú definované len ako lokálne premenné a sú odovzdávané ako argumenty samotných funkcií. Vďaka zadanému EPSG kódu je definovaný súradnicový systém funkciami *SpatialReference()* a *ImportFromEPSG()* z modulu *OSR* (balíčka *osgeo*), ktorý je určený na prácu s projekciami a súradnicovými systémami. Tento súradnicový systém je priraďovaný k tvoreným vektorovým a rastrovým vrstvám v kóde.

Funkciou *getattr()* používanou v objektovo orientovanom programovaní možno zadefinovať funkciu na výpočet balistickej trajektórie z modulu *numerical\_methods.py* na základe poskytnutého reťazcového názvu funkcie. Vlastnou funkciou *get\_raster\_from\_file()* sú získané parametre rastra DMR. Ďalej je určená veľkosť kroku, ktorá ak je zadaná ako *None*, tak je jej pridelená hodnota rozlíšenia rastra (používateľ chce, aby mala trajektória rovnaké rozlíšenie ako raster). Môže však byť nastavená používateľom ako menšia hodnota (používateľ vyžaduje detailnejšiu trajektóriu). Ak by bola nastavená väčšia ako rozlíšenie DMR, mohlo by to viesť k zlyhaniu niektorých algoritmov, preto je v takomto prípade hodnota zrazená na veľkosť rozlíšenia DMR, ktorá je tiež získaná z parametra definujúceho raster.

Funkciou *get\_geom\_list\_from\_file()* sú zadefinované parametre vektorovej bodovej vrstvy a primárne zoznam (list) geometrií jednotlivých bodov obsahujúcich ich 2D súradnice. V prípade, že majú byť využité líniové prekážky, je volaná funkcia *burn\_obstacles()*. Funkciou *get\_min\_max\_height()* sú pridelené hodnoty najväčšej a najmenšej výšky DMR. Na základe zadaných okrajov intervalu je vytvorený zoznam uhlov výstrelu pevne stanoveným počtom hodnôt zoznamu, ktorý odpovedá kroku 5° pri plnom rozsahu výškových uhlov, čo je výpočtovo optimálne. Zadefinované sú i 2 matice s rozmermi rastra DMR ako nulové, ktoré budú neskôr aktualizované pripočítavaním 1 vždy, keď konkrétna bunka bude vyhodnotená ako trajektórii prístupná či bude aspoň v dostrele (záleží od zvolených nastavení). Každá matica korešponduje typu trajektórie (a jednému pásmu rastra vo výsledku), ktorá môže bunku pretínať, teda trajektória penetrujúca bunku pod nižším uhlom výstrelu (napriamo) a trajektória pretínajúca bunku pod vyšším uhlom výstrelu.

Príslušný cyklus generuje raster „throwshedu“ individuálne pre každý bod výstrelu funkciou *throwshed()* a každý vytvorený dočasný raster analýzy viditeľnosti vo vnorených funkciách je tu vždy vymazaný, ak bol vytvorený. Funkciou *create\_raster\_file()* je vygenerovaný výsledok analýzy „throwshed“ ako dvojpásmový raster a nakoniec, ak bol generovaný aj DMP, tento je tiež odstránený z pamäte disku funkciou *remove\_temp\_files()*.

1.2 Funkcia get\_raster\_from\_file()

Táto vlastná funkcia preberá argument s cestou k rastru (DMR alebo viewshed), načíta tento raster ako dátový zdroj pomocou *Open()* a funkciou *GetRasterBand()* získava jeho pásmo obsahujúce údaje, z neho hodnoty rastra pomocou *ReadAsArray()* v podobe matice, ako aj geotransformačné údaje (zoznam s veľkosťami bunky v smere oboch osí, súradnice ľavého horného rohu rastra atď.) vďaka *GetGeoTransform()* a hodnotu prázdnych buniek rastra pomocou *GetNoDataValue()*. Použité funkcie sú z modulu *GDAL* (balíka *osgeo*) slúžiaceho na manipuláciu s rastrovými dátami. Tieto parametre sú vrátené do volajúcej funkcie všetky. Hoci nemusia byť všetky ďalej použité (napr. dátový zdroj), je nutné ich vrátiť a ponechať alokované, pretože inak by pri ďalších dátach čerpaných z nich (napr. pásmo) dochádzalo k takzvanému „osiroteniu“ a stali by sa tak nepoužiteľnými, čo by viedlo k zrúteniu sa programu. Tento krok je nevyhnutný potom aj pri načítaní vektorových vrstiev.

1.3 Funkcia get\_min\_max\_height()

Úlohou funkcie je vrátiť minimálnu a maximálnu hodnotu rastra DMR funkciami *GetMinimum()* a *GetMaximum()*. V prípadoch niektorých rastrov môže dôjsť k získaniu týchto extrémov z pásma rastra v podobe bezdátového *None,* preto je algoritmus zaopatrený prídavnou funkciou *ComputeStatistics()*, ktorá prepočíta aj tieto hodnoty, aby boli neskôr čerpateľné. Podstatné je, že tieto hodnoty sú v prípade DMP čerpané z rastra až po možnom zakomponovaní líniových prekážok, ktoré majú šancu zmeniť maximálnu hľadanú hodnotu.

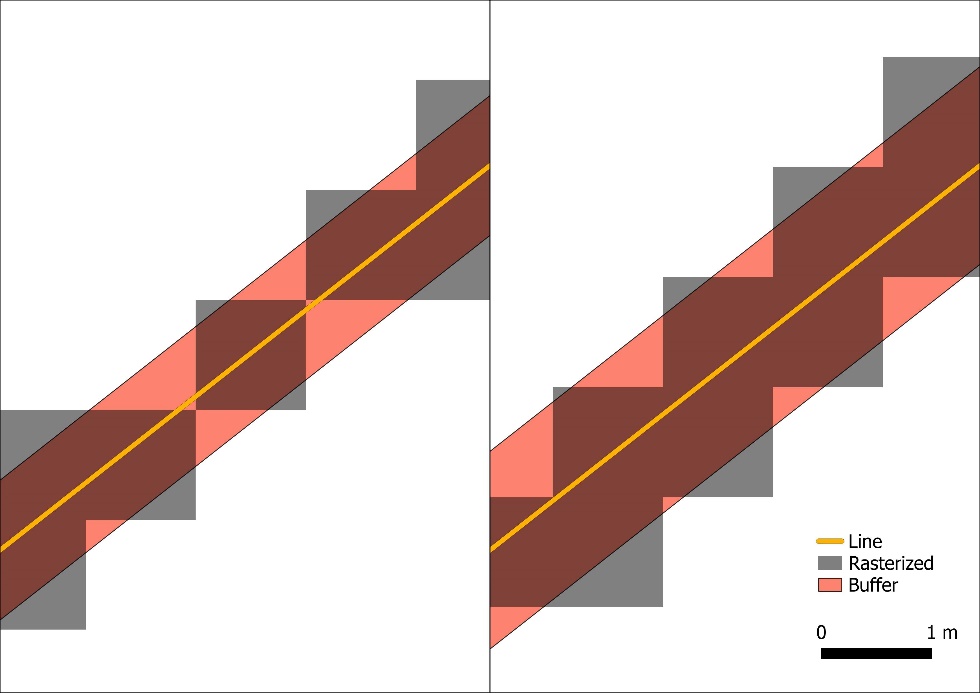
1.4 Funkcia get\_geom\_list\_from\_file()

Funkcia preberá argument s cestou k vektorovej vrstve (bodovej, prípadne líniovej), vytvára premennú odkazujúcu na dátovy zdroj vrstvy, funkciou *GetLayer()* je získaná konkrétna vrstva zo zdroja a metódou zoznamovej komprehenzie je získaný zoznam prvkov vektorovej vrstvy s využitím funkcie *GetFeature()*, a zoznam geometrií týchto prvkov funkciou *GetGeometryRef()*. Všetky vzniknuté odkazy na objekty sú vrátené.

1.5 Funkcia burn\_obstacles()

Tento algoritmus je za predpokladu zvolenia možnosti využitia umelých prekážok určený k úprave hodnôt buniek rastra DMR, cez ktoré majú prekážky prechádzať. Ich výška je pripočítaná k prislúchajúcim výškam buniek, čím vznikne nový raster DMP, ktorý je samostatne dočasne uložený a využitý v analýze, dokým celá neprejde, po čom sa dočasný súbor odstráni. Zoznam geometrií línií je získaný funkciou *get\_geom\_list\_from\_file()*. Z línií je nutné vygenerovať buffer, ktorý je treba vďaka plošnému charakteru rasterizovať a v podobe rastra tak pripočítať k originálnemu rastru DMR.

Definovaný je najprv parameter definujúci veľkosť buffera, resp. jeho polomer, ktorý je rovný polovici zadanej šírky prekážky. Ak je táto hodnota veľmi malá, je vyčíslený na veľkosť polovice diagonály bunky rastra, aby pri rasterizácii buffera nikdy nestala situácia, kedy by zapísané bunky reprezentujúce prekážku susedili len diagonálne. Obrázok č. 3 znázorňuje porovnanie použitia parametra buffera menšieho a rovného polovici diagonály. Prípad diagonálne susediacich buniek by znamenal, že v mieste stretu týchto buniek je šírka prekážky nulová. V kritickej situácii, kedy by trajektória prechádzala práve blízko takéhoto miesta, by jej body, ktorých výšky sú predmetom porovnávania s príslušnými výškami buniek rastra, mohli pripadnúť mimo vnútro bunky, resp. prekážky a hoci by trajektória pretínala prekážku, prekážka by bola ignorovaná.

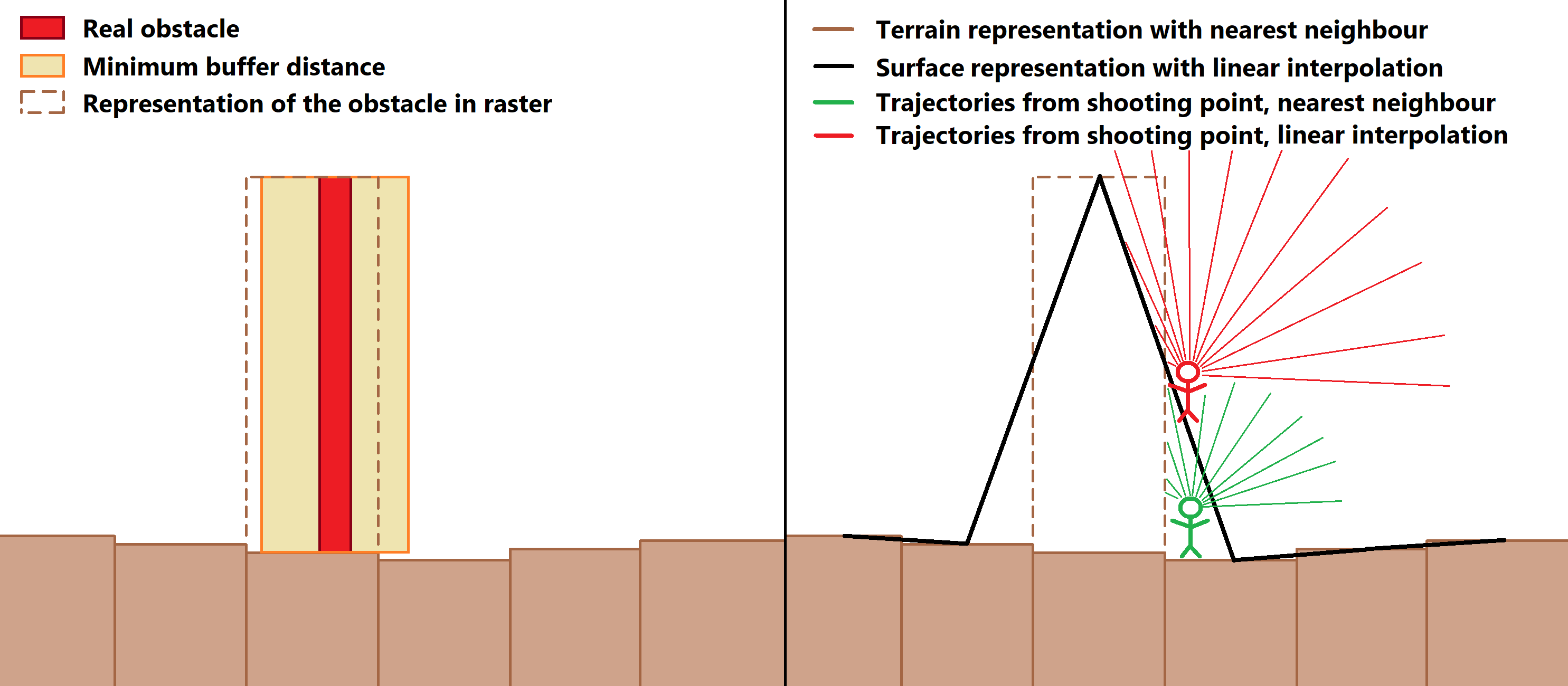


Obr. 3: Výsledok rasterizácie (použitie parametra buffera menšieho ako polovica diagonály vľavo a rovného polovici diagonály vpravo)

Takto rasterizovaná prekážka bude mať vždy šírku minimálne rovnú rozlíšeniu rastra, čo pri veľmi úzkych prekážkach, napr. palisády, môže predstavovať nereálnu reprezentáciu, ale z hľadiska charakteru algoritmu hľadajúceho prienik trajektórie je to nevyhnutné.

Funkciou *Buffer()* volanej na geometrii líniového prvku je vytvorená geometria bufferu a ak sa v líniovej vrstve vyskytuje viac ako 1 prvok, sú geometrie bufferov ostatných prvkov postupne zjednotené do jednej viacprvkovej geometrie. Funkciou *create\_and\_use\_outlayer()* je vytvorená nová vrstva obsahujúca geometriu buffera a funkciou *create\_raster\_file()* je vytvorené pásmo s dátovým zdrojom rastra buffera, do ktorého sa buffer rasterizuje funkciou *RasterizeLayer()* a to s hodnotami zadanej výšky prekážky. Pásmo rastra potom možno načítať ako maticu funkciou *ReadAsArray()*. K pôvodnej globálnej premennej odkazujúcej na maticu rastra DMR je pripočítaná matica rasterizovanej prekážky, čím sa pôvodná globálna prepíše. Taktiež sa prepíše aj globálne definované pásmo rastra DMR na DMP, ktoré je vytvorené funkciou *create\_raster\_file()*, čím sa v pamäti disku vytvorí dočasný súbor s DMP. Tento je vymazaný až na konci funkcie *main()*, ale ostatné dočasné súbory s vektorovou vrstvou a rastrom buffera sú vymazané už na konci aktuálnej funkcie funkciou *remove\_temp\_files()*.

Za zmienku stojí úvaha, že prekážky by bolo možno lepšie považovať za samostatné 3D objekty, ktoré by neboli zatavené do originálu DMR. Dôvodom je fakt, že takto sú prekážky deformované prvýkrát pri ich rasterizácii a druhýkrát pri algoritme porovnávajúcom výšky bodov trajektórie s výškami prislúchajúcich buniek získanými lineárnou interpoláciou, čím sa zo zvislej steny prekážky stane šikmá so sklonom závislým od výšky susednej terénnej bunky pri prekážke. Následkom môže tiež byť v ojedinelej situácii priradenie nesprávnej výšky strelcovi, ktorý je umiestnený tesne pri prekážke (výška, na ktorej strelec stojí, je prevzatá z bunky, na ktorej sa bod miesta výstrelu nachádza). Preto je v tomto prípade bezpečnejšie používať interpoláciou metódou najbližšieho suseda. Tieto deformácie a ojedinelú situáciu znázorňuje obrázok č. 4. Je možné, že na prekážku sa bude v budúcnosti prihliadať ako na 3D objekt a bude sa posudzovať príslušnosť bodu trajektórie vzhľadom na terén a zvlášť na tento objekt, teda či v ňom bod leží alebo nie.



Obr. 4: Deformácie prekážky rasterizáciou a interpoláciou a ojedinelá situácia pridelenia nesprávnej výšky miesta výstrelu

1.6 Funkcia create\_and\_use\_outlayer()

Funkcia slúži na vytvorenie novej vektorovej vrstvy, kde je najprv nutné funkciou *GetDriverByName()* definovať ovládač a funkciou *CreateDataSource()* vytvoriť dátový zdroj, potom vygenerovať vrstvu funkciou *CreateLayer()* a prvok funkciou *Feature()* (funkcie modulu *OGR*, balíka *osgeo*), aby bola doň vložená geometria funkciou *SetGeometry()*. Prvok s geometriou je nakoniec vložený do vrstvy funkciou *CreateFeature()* a táto vrstva je s dátovým zdrojom vrátená.

1.7 Funkcia create\_raster\_file()

Táto funkcia vytvára rastrový súbor vo formáte \*.tif a funguje v dvoch módoch. Buď vytvára prázdne pásmo pripravené na zápis hodnôt špecifickou funkciou mimo tejto funkcie (napr. rasterizácia v *burn\_obstacles()*) alebo priamo zapisuje matice s hodnotami rastra do pásma (tvorba DMP v *burn\_obstacles()* alebo tvorba finálneho rastra „throwshedu“ v *main()*). Taktiež dátovým typom rozlišuje, či sa jedná o zápis hodnôt desatinných čísel (*float*, pri tvorbe DMP) alebo celých čísel (*integer*, pri zápise počtu zásahov vo výslednom rastri analýzy).

Najprv je vytvorený ovládač funkciami *Create()* a *GetDriverByName()*, kde sa argumentami definujú niektoré parametre rastra ako aj funkciou *SetGeoTransform()* a súradnicový systém pomocou *SetProjection()* tak, aby boli totožné s rastrom DMR. Funkcia je schopná zápisu matíc do viacerých pásiem rastra, kde najprv je pásmo získané funkciou *GetRasterBand()*, potom je nastavená hodnota pre prázdne bunky funkciou *SetNoDataValue()* a nakoniec je zapísaná matica do príslušného pásma. Pásmo je vrátené, no nie vždy je využité.

1.8 Funkcia remove\_temp()

Funkcia vykonáva odstránenie všetkých dočasných súborov, ktoré sa ukladajú do priečinka, kde bude uložený aj súbor s výsledkom analýzy. Algoritmicky je postavená tak, aby zistila formáty všetkých prítomných súborov na základe konkrétneho názvu (kvôli vektorovým vrstvám, ktoré okrem súboru vo formáte \*.shp ešte disponujú aj inými formátmi, napr. \*.prj). Tu je použitá funkcia *listdir()* knižnice *os* a nakoniec funkcia *remove()* z tejto knižnice vymaže dočasné súbory.

1.9 Funkcia throwshed()

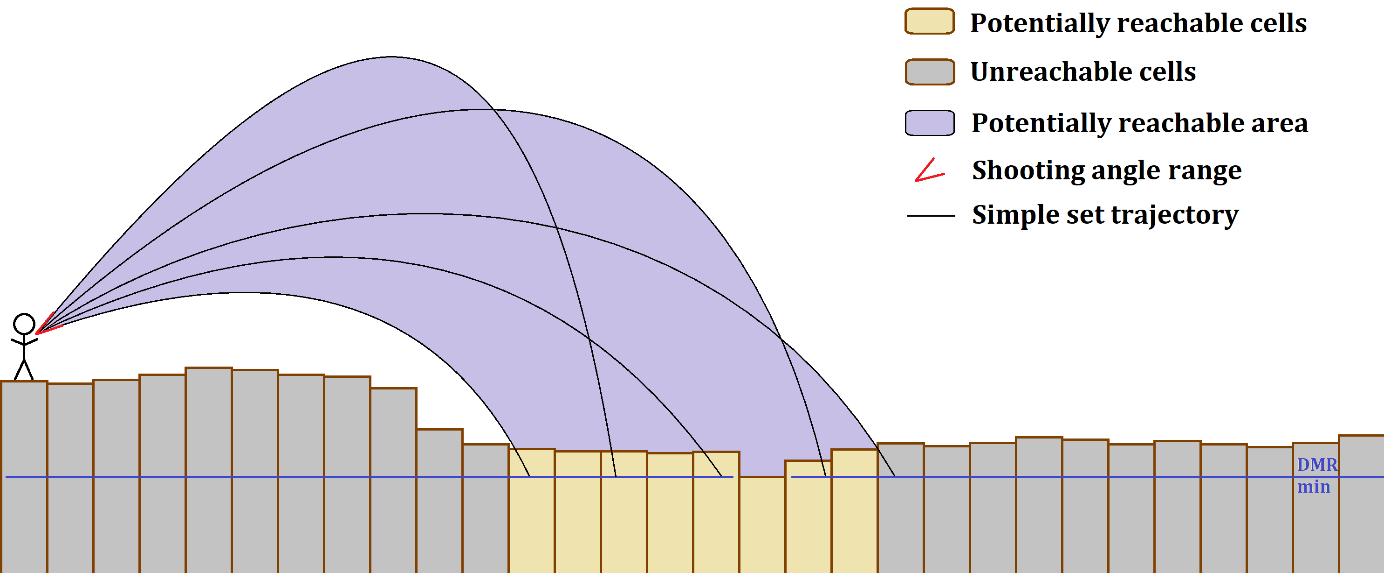
Funkcia slúži na analýzu „throwshed“ pre jeden bod – jedno miesto výstrelu. Vytvára geometriu bodu, ktorému je priradená aj výška terénu získaná interpoláciou vo funkcii *int\_function()*. Pre tento bod je vytvorená jednoduchá séria trajektórií funkciou *trajectory\_simple\_set()*, táto séria je dodatočne zhustená funkciou *trajectory\_set()*. Na základe série sú funkciou *create\_trajectory\_fields()* vytvorené vertikálne plochy ohraničujúce oblasti, v ktorých bunky rastra môžu zasiahnuť trajektórie s menším uhlom výstrelu a väčším uhlom výstrelu. V prípade využitia analýzy viditeľnosti je funkciou *create\_viewshed()* vygenerovaný raster viditeľnosti. Funkciou *assign\_values\_to\_throwshed()* sú bunkám rastra „throwshedu“ priradené hodnoty na základe ich príslušnosti k spomínaným poliam, terénnym zákrytom a viditeľnosti.

1.10 Funkcia int\_function()

Funkcia vracia interpolované výšky z DMR, resp. DMP, na základe argumentu funkcie - 2D polohy bodu. Momentálne je možná interpolácia bilineárna a metódou najbližšieho suseda. Uplatnenie má funkcia pri získaní výšky miesta výstrelu a výšok terénu pod bodmi trajektórií, pri hodnotení, či trajektórie nepretínajú terén skôr ako v cieľových bunkách.

1.11 Funkcia trajectory\_simple\_set()

V rámci tejto funkcie je vytvorená globálna premenná, do ktorej budú v podobe zoznamu ukladané všetky trajektórie s údajmi o uhle výstrelu a súradnicach bodov trajektórií od miesta výstrelu ku koncu trajektórií vo vertikálnej rovine. Definícia základného setu trajektórií v rozsahu minimálneho a maximálneho uhla výstrelu (podľa zoznamu uhlov výstrelu danom vo funkcii *main()*) má podobu zoznamovej komprehenzie. Účel celej série spočíva v generovaní trajektórií, ktoré z oboch strán obklopujú cieľovú bunku, pretože len na základe pozície bunky v rámci dvoch susedných trajektórií s malým rozdielom v uhle výstrelu je možné iteračnou metódou vyhľadať taký uhol výstrelu, s ktorým bude vytvorená trajektória pretínať cieľovú bunku. Nakoľko sú trajektórie generované numericky a nie analyticky, nie je možné na základe vzájomnej polohy miesta výstrelu a bunky analyticky nájsť vstupné parametre do výpočtu trajektórie, ktorá by pretínala cieľovú bunku. Preto bol zvolený iteračný princíp. Obrázok 5 zobrazuje jednoduchú sieť trajektórií, ktorá ohraničuje určitú plochu. Bunky spadajúce do tejto plochy sú považované za potenciálne zasiahnuteľné (potenciálne, pretože hodnotená trajektória môže pretínať terén skôr, ako v cieľovej bunke rastra) a bunky mimo nej budú považované za vždy nedostupné (mimo dostrel).

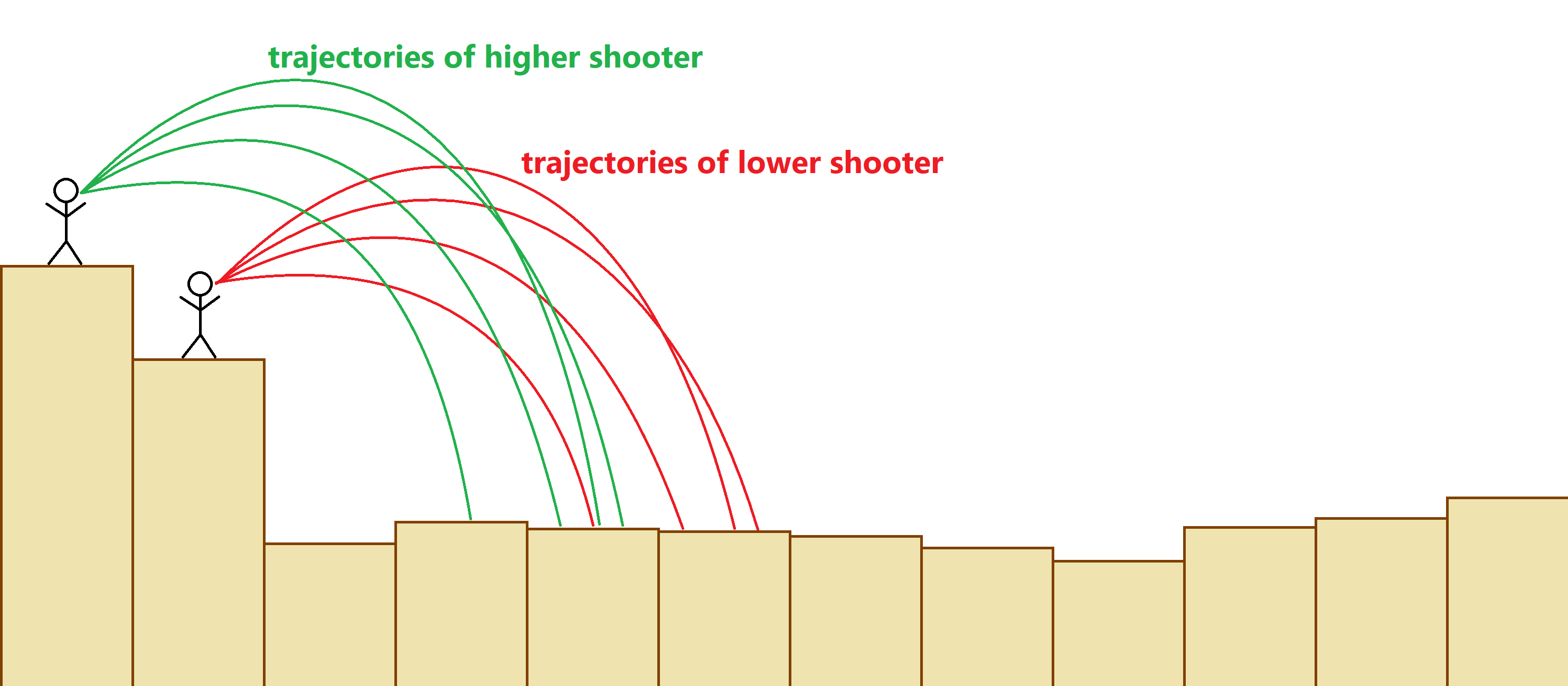


Obr. 5: Potenciálne zasiahnuteľná oblasť v rámci jednoduchej siete trajektórií

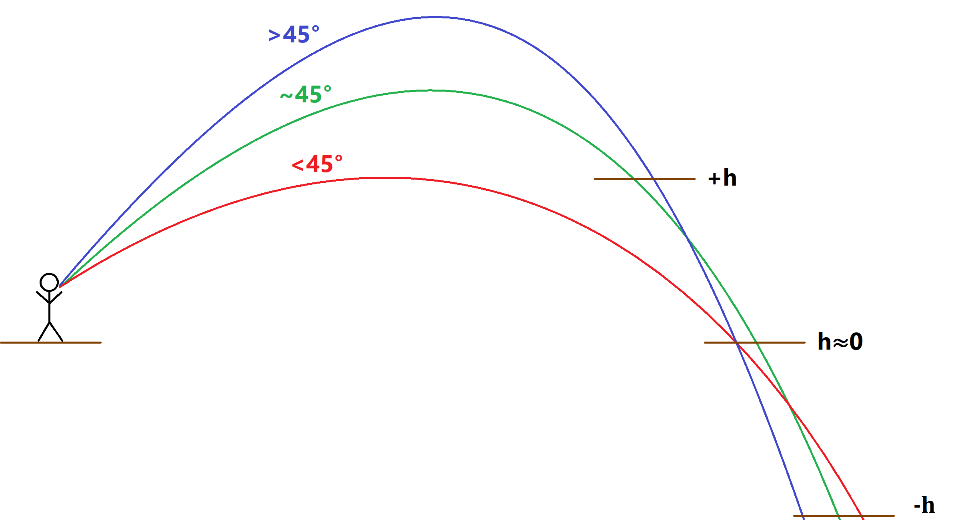
1.12 Funkcia trajectory\_set()

Táto funkcia má niekoľko častí. Najprv vyhľadá trajektóriu s najväčším dosahom na úrovni minimálnej výšky rastra. Od tejto trajektórie je po trajektóriu s maximálnym uhlom výstrelu zhustená séria trajektórií tak, aby nedošlo k opomenutiu buniek na okraji série vplyvom jej nedostatočného rozlíšenia. Nakoniec je vytvorená obálka setu trajektórií, ktorá definuje bezpečnú zónu v okolí miesta výstrelu.

Pri hľadaní trajektórie s najväčším dosahom sa vykonáva kontrola, či neexistuje uhol výstrelu, ktorý je blízky uhlu výstrelu trajektórie s aktuálne najväčším dosahom a to na úrovni najnižšej výšky rastra DMR. Na najnižšiu výšku DMR je kladený dôraz z toho dôvodu, že je to úroveň, pod ktorou sa už bunky nemôžu nachádzať a trajektórie, ktoré sú predmetom porovnávania s terénom, nemá zmysel generovať nižšie. Všetky trajektórie sú teda generované od miesta výstrelu len po túto výšku. Keďže sa miesta výstrelu môžu nachádzať v rôznych výškach, čo mení prevýšenie medzi najmenšou výškou DMR a miestom výstrelu, mení to aj charakter celej série trajektórií a preto nemožno predávať ten istý set trajektórií medzi dvomi a viacerými bodmi (miestami výstrelu) na skrátenie výpočtov. Pre predstavu je táto skutočnosť znázornená na obrázku 6. A pretože sa miesto výstrelu môže nachádzať na najnižšej bunke DMR alebo vyššie, hraničná výška generovania trajektórie bude vždy relatívne nižšie. To zmenšuje uhol výstrelu od klasickej hodnoty 45°, čo je uhol, pod ktorým je možné dosiahnuť najväčší dostrel (vo vákuu) na rovine, pretože s klesajúcou relatívnou výškou medzi miestom výstrelu a cieľovým územím klesá aj uhol výstrelu pri zabezpečení čo najväčšieho dostrelu. Pre zaujímavosť, miesta nachádzajúce sa vyššie ako miesto výstrelu vyžadujú naopak väčší uhol výstrelu ako 45° pre najväčší dosah. Jedná sa len o jeden z predpokladov lepšieho pochopenia správania sa balistických trajektórií pri zmene ich parametrov a terénnych pomerov. Obrázok 7 ilustruje tento predpoklad s malou vzorkou siete trajektórií, kde každá má najväčší dosah v rôznej výške cieľa.

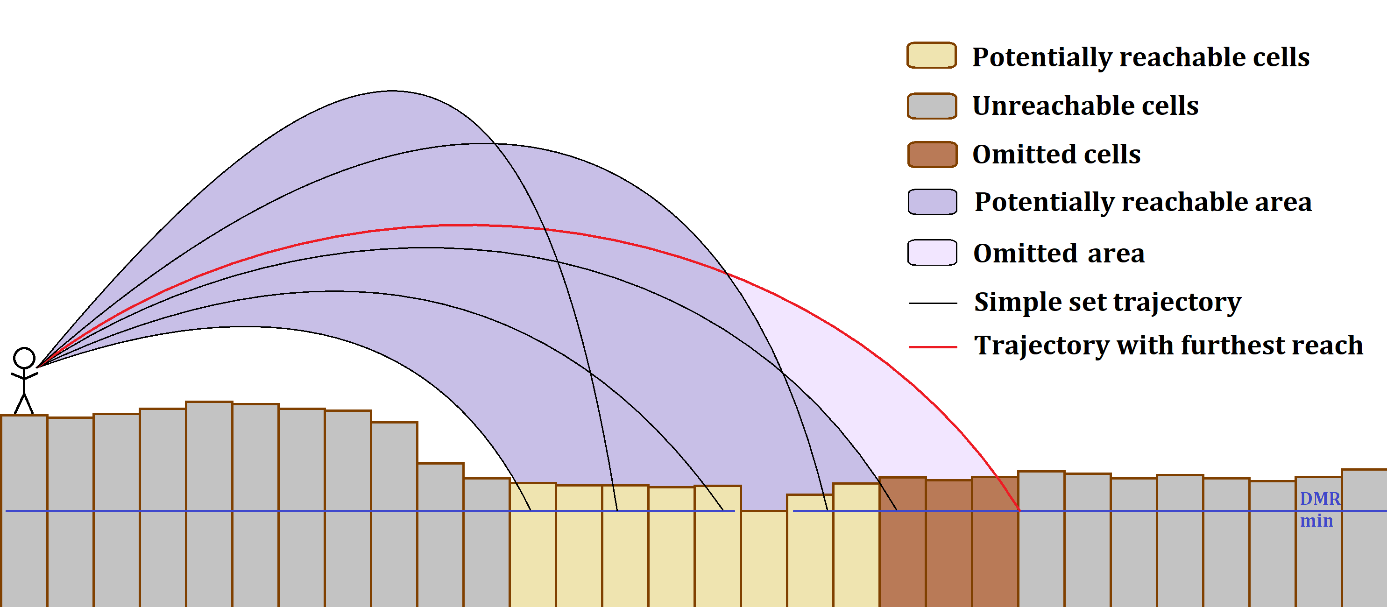


Obr. 6: Porovnanie sietí trajektórií strelcov s rozdielnou výškou



Obr. 7: Najširšie trajektórie prislúchajúce rôznym relatívnym výškam cieľa

V blízkosti trajektórie s aktuálne najväčším dosahom, trajektórie, ktorá je súčasťou len základného setu trajektórií, sa bude nachádzať trajektória, ktorá bude mať najväčší možný dosah (najširšia). Je nutné túto trajektóriu nájsť, aby k nej bolo možné priraďovať príslušnosť buniek rastra DMR, pretože ak by bola ponechaná najširšia trajektória len z prvotne vygenerovaného setu, veľmi pravdepodobne by existovala ešte širšia trajektória, len s mierne odlišným uhlom výstrelu, ktorá by mala dosah väčší o niekoľko-násobok veľkosti bunky rastra, a tak by došlo k opomenutiu niekoľkých buniek v smere od miesta výstrelu, pretože by boli považované automaticky za nedosiahnuteľné. Obrázok 8 znázorňuje, ako môže nastať táto situácia. Opomenutie nemá smerový charakter, ale skôr plošný, pretože ak by povrch bunky pripadol práve do plochy, o ktorú najširšia možná, no neznáma trajektória obohatila základnú plochu potenciálne dosiahnuteľnej oblasti, bola by príslušná bunka považovaná za nedostupnú. Ak by sa vyskytoval povrch všetkých možných buniek v tejto malej oblasti, v horizontálnom smere by bolo opomenutých viac buniek ako na obrázku.



Obr. 8: Opomenutie buniek pri použití najširšej trajektórie zo základnej siete

Preto prvý algoritmus tejto funkcie hľadá (aktuálne) najširšiu trajektóriu zo základnej siete a medzi ňu a susedné trajektórie vkladá trajektórie nové s počiatočným uhlom výstrelu nastaveným ako priemer uhlov najširšej a susednej trajektórie. Vzniknú dve nové trajektórie a zisťuje sa vzťah dosahu tej širšej s dosahom predošlej širšej. Toto sa opakuje, dokým rozdiel dosahu trajektórie z tejto a pedošlej iterácie nie je menší, ako polovica veľkosti bunky, čo je v celom programe nastavená dovolená priestorová odchýlka (nastavená charakteristika prestnosti výsledku). Takto je to nastavené, aby nadbytočnými iteráciami pri ďalšom spresňovaní nedochádzalo k predlžovaniu výpočtového času a pretože samotné rozlíšenie analyzovaného rastra DMR nevyžaduje podrobnejší detail.

Algoritmus je po syntaktickej stránke riešený zavedením dvojprvkového zoznamu, kde sa bude ukladať vzdialenosť (súradnica X) posledného bodu najširšej trajektórie z aktuálneho a predošlého opakovania cyklu, ktorý nasleduje po definícii zoznamu. Do zoznamu je najprv vložená maximálna súradnica X najširšej trajektórie vygenerovanej ešte v predošlej funkcii *trajectory\_simple\_set()* a táto je ponásobená konštantami tak, aby nasledujúci cyklus prebehol minimálne v dvoch opakovaniach (aby sa hodnotou líšili o jeden násobok dosahu). Úlohou cyklu je v prípade aktuálne najširšej trajektórie (nájdenej funkciou *max()* s využitím vyhľadávania podľa kľúča), ktorej nepripadá uhol výstrelu rovný maximu alebo minimu rozsahu, aktualizovať globálnu premennú so setom trajektórií o nové dve vložené do stredu medzi akuálne najširšiu a jej susedné trajektórie v zmysle ich spriemerovaných uhlov výstrelu. Ak pripadá uhol výstrelu najširšej trajektórie na okraj rozsahu, doplnená je len jedna trajektória smerom dovnútra dovoleného rozsahu uhlov výstrelu. Preto sa v cykle nachádzajú až tri podmienky. Dosah novej trajektórie (v prípade dvoch nových trajektórií - tej širšej) prepíše prvú z hodnôt v zozname a pôvodná prvá je zapísaná do druhej hodnoty zoznamu. Na konci cyklu sú novovygenerované trajektórie zapísané do setu podľa veľkosti ich uhla výstrelu, aby boli zoradené. Cyklus sa opakuje dokým nebude rozdiel týchto dvoch hodnôt zoznamu menší ako polovica veľkosti bunky rastra DMR. V nasledujúcich opakovaniach by bol každý ďalší rozdiel o polovicu menší, čo znamená že by pri akumulácii stále menších horizontálnych krokov smerom k teoreticky najširšej trajektórii došlo nasčítaniu približne polovice šírky bunky rastra DMR. Jedná sa ale o maximálnu možnú hodnotu, v skutočnosti je väčšina takýchto nasčítaných rozdielov medzi najširšou realizovanou a teoretickou trajektóriou ešte menšia. Obrázok 9 graficky opisuje práve situáciu najväčšej odľahlosti algoritmickej a teoretickej trajektórie v dvoch prípadoch, kedy vzájomný posun siete trajektórií a rastra o polovicu šírky bunky predstavuje rozdiel medzi opomenutí a uznaním bunky, ktorá by na základe teoreticky najširšej trajektórie uznaná mala byť.

A diagram of a person's face

Description automatically generated with medium confidence

Obr. 9: Prípad opomenutia a uznania bunky, ktorá by uznaná mala byť

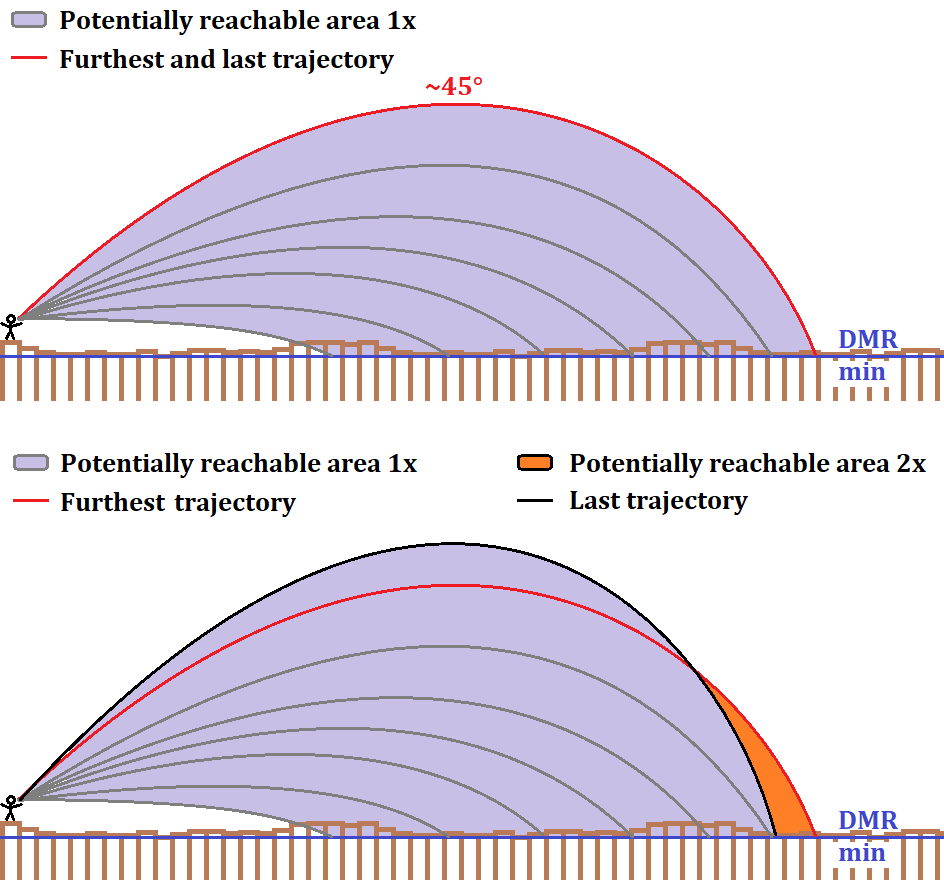
Dôvodom minimálne dvoch opakovaní cyklu je, že existuje pravdepodobnosť splnenia podmienky dovoleného rozdielu dosahu novej a predošlej trajektórie už po prvom prejdení cyklu s tým, že po druhom opakovaní cyklu by tento rozdiel náhle narástol. Napovedá to o komplikovanom správaní trajektórií. Dokonca aj obrázok 10 znázorňujúci postupne približujúce sa trajektórie k tej teoreticky najširšej zobrazuje len najjednoduchšiu situáciu. V iných prípadoch dochádza napríklad k výberu širšej trajektórie z vygenerovanej dvojice – tej s menším uhlom výstrelu, k vygenerovaniu novej dvojice, kde sú obe trajektórie kratšie ako najširšia trajektória zo základného setu alebo k postupnému približovaniu k najširšej trajektórii zo základného setu, pretože táto náhodou pripadla do pozície teoreticky najširšej trajektórie. So všetkými týmito prípadmi dokáže algoritmus pracovať a vyhnúť sa tak nesprávnemu chodu zvyšku kódu. Takto je splnený krok nájdenia najširšej trajektórie s prihliadaním na rozlíšenie rastra DMR.

A screenshot of a diagram

Description automatically generated

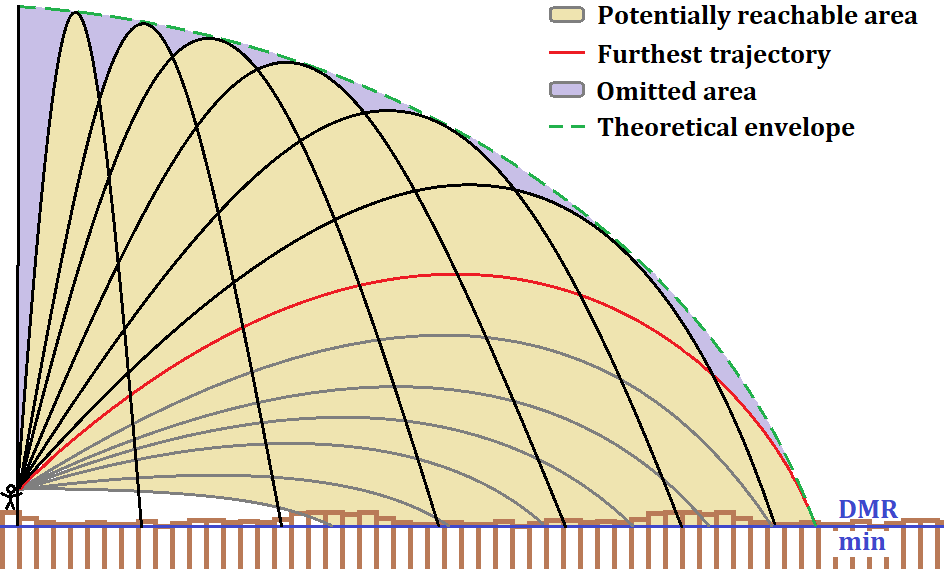
Obr. 10: Proces postupného priblíženia k najširšej možnej trajektórii

Neskoršie funkcie programu vytvárajú 2 polia, vertikálne polygóny vymedzujúce oblasti, v ktorých sa očakáva zásah projektilom obomi typmi trajektórie, aby sa zistil priestorový vzťah bunky ako bodu a zasiahnuteľného poľa ako polygónu. Ak bunka leží v tomto poli, považuje sa za potenciálne zasiahnuteľnú a podľa zvoleného módu analýzy sa buď považuje za zasiahnuteľnú alebo sa môže pristúpiť k iteračnej metóde hľadania jej pretínajúcej trajektórie. Ak neleží v tomto poli, tak sa bunka považuje za nezasiahnuteľnú. Po vyhľadaní najširšej trajektórie sa vykoná kontrola, či nie je náhodou aj poslednou trajektóriou s najväčším počiatočným uhlom výstrelu. Ak by sa tak preukázalo, neskoršie algoritmy by vyhodnocovali len pole zasiahnuteľné trajektóriami s nižším uhlom výstrelu, pretože žiadna bunka v takejto oblasti nemôže byť zasiahnutá druhýkrát. Obrázok 11 znázorňuje príklad pre túto situáciu a aj pre situáciu, kde je poslednou trajektóriou tá, ktorá má o niečo väčší uhol výstrelu ako najširšia trajektória, čo dáva priestor vzniknúť aj malej oblasti, kde sú bunky zasiahnuteľné dvakrát.



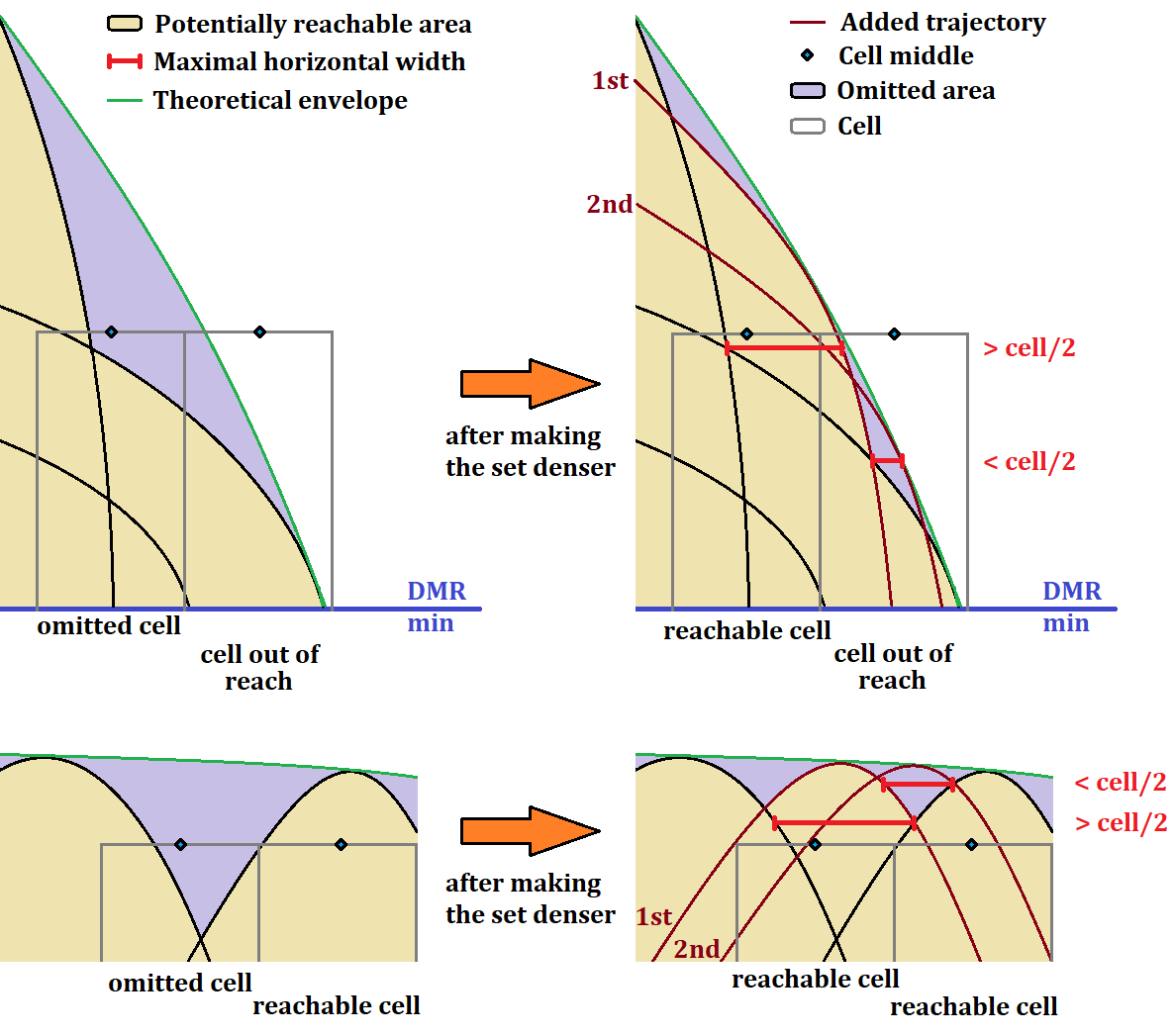
Obr. 11: Prípad najširšej trajektórie ako poslednej v sete a predposlednej

Nasleduje zhustenie siete trajektórií od najširšej trajektórie prislúchajúcej najmenšej výške rastra po poslednú trajektóriu v celom sete. Toto zhusťovanie sa vykonáva taktiež z dôvodu, aby neboli opomenuté bunky, ktoré by sa mohli nachádzať v oblastiach nedostatočne pokrytých vygenerovanými trajektóriami. Obdobne ako obrázok 8 zobrazuje túto situáciu obrázok 12.



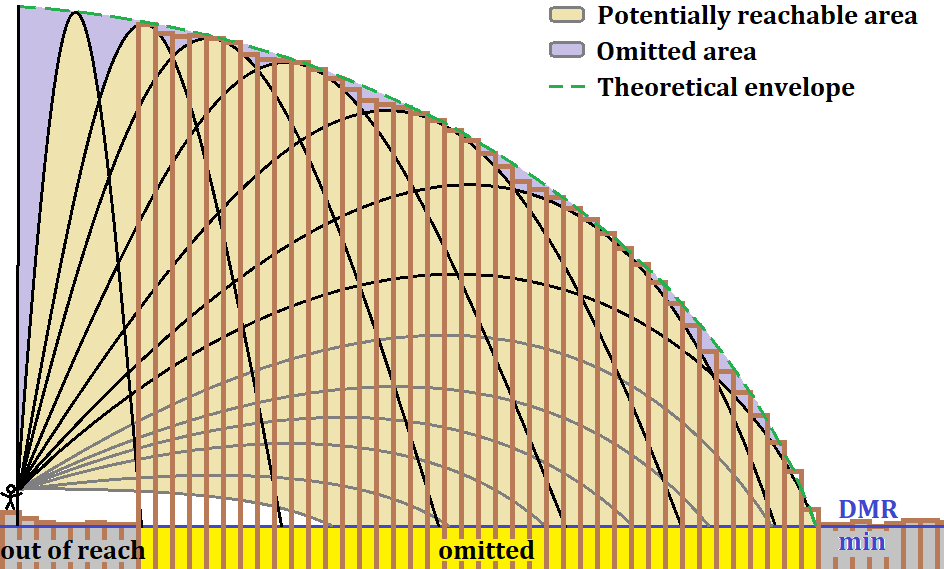
Obr. 12: Opomenuté oblasti pri nedostatočne zhustenej základnej sieti trajektórií

Ideálny prípad by bol, keby algoritmom vytvorené potenciálne zasiahnuteľné územie pokrývalo celú oblasť teoretickej obálky. Teoretická obálka ohraničuje skutočnú bezpečnú zónu a v prípade parabolických trajektórií vo vákuu by táto obálka bola vyjadriteľná rovnicou (Eugene Butikov, 2003). Avšak v prípade uvažovania faktorov ako je napr. odpor vzduchu je možné sa k teoretickej obálke len numericky priblížiť. Takéto približovanie sa vykonáva algoritmicky spomínaným zhusťovaním siete trajektórií. Zhusťovaním sa zmenšujú opomenuté oblasti pripomínajúce trojuholníky. Síce nie je možné ich zmenšovať do nekonečna, aby nedošlo k opomenutiu buniek spadajúcich sem, je možné nastaviť vhodné zastavovacie kritérium pri splnení určitej presnosti vykreslenia výsledku analýzy za predpokladnu priaznivých terénnych pomerov. Ak by bol terén vodorovnou rovinou a uvažujeme šírku bunky, nastavenie maximálnej najväčšej šírky každej oblasti opomenutia na polovicu rozlíšenia rastra by predstavovalo vo výsledku opomenutie maximálne jednej bunky na okraji „throwshedu“. Obrázok 13 znázorňuje túto situáciu v dvoch prípadoch – okraj najširších a najužších trajektórií. Vidieť situáciu pred zhustením a po zhustení trajektórií, kde ich okraje predstavujú algoritmickú obálku, ktorá sa blíži k teoretickej obálke. Aby bola bunka považovaná za potenciálne zasiahnuteľnú, musí sa jej stred nachádzať vnútri algoritmickej obálky. Môže sa stať, že stred bunky bude tesne mimo hranice obálky a jej polovica vnútri obálky. Metodika však považuje za zasiahnuteľnú bunku takú, ktorej aspoň polovica je prístupná. Je dôležité, aby potom vedľa takejto bunky tesne za hranicou vygenerovanej obálky nemohla byť ďalšia bunka, ktorej časť je ešte vnútri teoretickej obálky. Cieľom je v grafickom výsledku analýzy na okraji zasiahnuteľnej oblasti uvažovať len s maximálne jednou ďalšou celou bunkou smerom von z oblasti, ktorá ešte teoreticky mohla byť zasiahnutá, ale ani zlomok druhej nasledujúcej bunky už by zasiahnutý nemohol byť.



Obr. 13: Vyhnutie sa opomenutiu buniek na okraji širších a užších trajektórií

Takto vyjadrená charakteristika presnosti nie je typická, ale pri uvažovaní spôsobu výpočtu je pochopiteľná. Často sa dokonca vyskytujú situácie, kde terén nebýva vodorovný a môže sa stať, že za sebou niekoľko buniek je opomenutých lebo akurát pripadli do oblasti opomenutia. Toto znázorňuje obrázok 14. Mierna variácia vstupných údajov by mohla priniesť úplne iný výsledok. Tu môže byť výhodný výpočet pravdepodobného „throwshedu“ čo i len s minimálnym počtom opakovaní.



Obr. 14: Ojedinelá situácia opomenutia niekoľkých buniek za sebou

Z obrázku 14, ktorý stavia situáciu z obrázku 12 do extrému, vidieť, že može existovať rozsiahla časť rastra, ktorá sa automaticky vyhodnotí ako nezasiahnuteľná, pretože stredy buniek sa dostali mimo algoritmickú obálku, hoci by podľa teoretickej obálky by mali byť považované za potenciálne zasiahnuteľné a mali by byť odovzdané algoritmom posudzujúcim, či cieľové bunky pretínajúce trajektórie nepretínajú terén skôr alebo priamo v cieľovej bunke (rozhodnutie o zasiahnuteľnosti). Nezáleží na veľkosti opomenutých oblastí, vždy bude existovať šanca, že stredy buniek pripadnú práve sem.

Programovo sa zhustenie siete trajektórií rieši tak, že sa najprv zadefinije premenná, ktorá bude obsahovať súradnice X,Y bodov algoritmickej obálky. Do jej polí sa hneď zadefinuje prvý bod, ktorým je posledný bod najširšej trajektórie. Tu začína obálka a postupne bude aktualizovaná o ďalšie body okrajových častí jednotlivých trajektórií, ktoré sa najviac približujú k skutočnej obálke a to po dostatočnom zhustení siete. V tomto bode sa zadefinujú už aj súradnice vnútorného priesečníka z predošlého cyklu zhusťovania. Jedná sa o vnútorný priesečník v rámci oblasti/trojuholníka, ktorým obálku aktualizujeme, resp. uznáme, že už nie je potrebné ju o tento úsek doplniť. Vnútorný priesečník vznikne pretnutím aktuálnej a druhej nasledujúcej (už v sieti existujúcej) trajektórie je len nastavený na hodnoty súradníc posledného bodu najširšej trajektórie, teda prvého bodu algoritmickej obálky, pretože si to prvý priebeh nasledujúceho cyklu vyžaduje, ale v následných opakovaniach cyklu už bude zastávať legitímnu úlohu vnútorného priesečníka z predošlého cyklu.

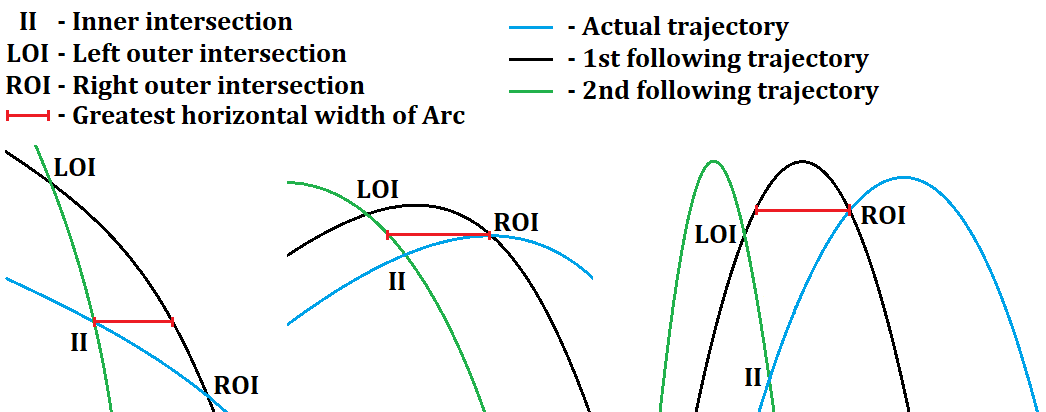
Zvyšok funkcie pokrýva cyklus, ktorý bude jednoduchú sieť trajektórií dopĺňať o nové trajektórie, dokým nebude v celom rozsahu sieť dostatočne hustá na to, aby vytvorila vhodnú obálku. V cykle budú vždy vystupovať 3 trajektórie, a to aktuálna, prvá nasledujúca a druhá nasledujúca. Aktuálnou sa bude považovať v prvom priebehu cyklu najširšia trajektória a za druhú nasledujúcu sa považuje paradoxne prvá nasledujúca z jednoduchého setu. Medzi ne sa vloží nová trajektória, ktorej počiatočný uhol výsrelu sa vypočíta ako priemer počiatočných uhlov zvyšných dvoch trajektórií. Cyklus potom buď vyhodnotí túto novú trajektóriu za nepotrebnú alebo potrebnú. Ak nepotrebnú, tak celý úsek medzi aktuálnou a druhou nasledujúcou trajektóriou sa považuje za dostatočne zhustený, vložená nepotrebná trajektória sa vymaže a cyklus sa spustí znova už s novou aktuálnou trajektóriou nastavenou na druhú nasledujúcu trajektóriu z predošlého cyklu a tak sa hodnotí hustota série trajektórií presne od bodu, v ktorom sa hodnotenie úspešne prestalo v predošlom cykle. Ak sa vložená trajektória považuje sa potrebnú, tak tento úsek setu nie je dostatočne zhustený a tak sa táto nová trajektória uloží do globálnej premennej so zoznamami súradníc bodov trajektórií a cyklus sa spustí znova s ponechanou aktuálnou trajektóriou, ale druhou nasledujúcou trajektóriou už bude nová trajektória z predošlého cyklu. Medzi týmito sa opäť vygeneruje nová, prvá nasledujúca trajektória a bude sa hodnotiť, či je potrebná alebo nie. Takto sa cyklus opakuje dokým každý úsek obálky nie je dostatočne zhustený až po poslednú trajektóriu v celom sete.

V cykle teda po výpočte uhla výstrelu novej trajektórie nasleduje samotná generácia trajektórie pomocou funkcie zvolenej metódy výpočtu a nová trajektória sa zapíše za aktuálnu trajektóriu v zozname globálnej premennej obsahujúcej informácie o ostatných trajektóriách. Existuje malá šanca, že v prvom priebehu cyklu, teda keď je aktuálnou trajektóriou tá najširšia trajektória, že nová vygenerovaná trajektória bude mať väčší dosah a bude novou najširšou trajektóriou. Preto je táto ojedinelá sitácia ošetrená podmienkou, v ktorej tele sa zvýši index aktuálnej trajektórie o 1, počiatočný bod obálky sa zmení na posledný bod novej najširšej trajektórie a takisto sa tieto súradnice prepíšu aj v premennej vnútorného priesečníka z predošlého cyklu. Takto je aktualizovaná najširšia trajektória a cyklus sa spustí odznovu.

V opačnom prípade cyklus pokračuje výpočtom súradníc pravého vonkajšieho priesečníka v aktuálnom cykle, teda priesečníka aktuálnej a prvej nasledujúcej trajektórie a za podmienky že druhá nasledujúca trajektória nemá počiatočný uhol 90°, tak sa vypočíta aj ľavý vonkajší priesečník ako priesečník prvej a druhej nasledujúcej trajektórie a vnútorný priesečník ako priesečník aktuálnej a druhej nasledujúcej trajektórie. V prípade úseku, kde je druhá nasledujúca trajektória práve poslednou v sete s počiatočným uhlom výstrelu 90°, nemožno vypočítať priesečník predposledných lomených trajektórií s poslednou, ktorá smeruje zvislo hore a vracia sa zvislo dole. Na výpočty priesečníkov sa používa vlastná funkcia *calculate\_intersection()*, ktorej argumenty sú polia súradníc X,Y jednej a druhej trajektórie. Vstupujú súradnice všetkých bodov trajektórií okrem prvého, ktorý je v mieste výstrelu a ktorý je rovnaký pre všetky trajektórie, preto je taktiež ich vzájomným, nežiaducim priesečníkom.

Nasleduje opäť kontrola, či druhá nasledujúca trajektória nemá počiatočný uhol výstrelu 90°. Ak nie, cyklus pokračuje v tele. Ak áno, skontroluje sa, či už nová vložená trajektória dostatočne zhustila posledný úsek obálky. Ak nie, tak sa cyklus začne odznova. Ak áno, vykoná sa niekoľko krokov. Ide o individuálny prístup, ktorý si situácia za daných podmienok vyžaduje. Tento úsek bude dostatočne zhustený, ak súradnica X posledného bodu vloženej trajektórie bude menšia ako rozlíšenie rastra. To znamená, že najvyšší bod tejto trajektórie bude od najvyššieho bodu poslednej zvislej trajektórie vzdialený približne polovicu veľkosti bunky, čo metodicky postačuje. V tele vnorenej podmienky je v globálnej premennej série trajektórií doplnená aktuálna trajektória o indexy začiatočných a koncových bodov úseku trajektórie, o ktorý bola obálka v tu volanej funkcii *update\_envelope()* doplnená. Doplnené sú aj indexy bodov na obálke, ktoré boli z priesečníkov doplnené navyše a obklopujú regulérne body trajektórie. Jedná sa o úsek aktuálnej trajektórie od vnútorného priesečníka z predošlého cyklu po priesečník aktuálnej a prvej nasledujúcej trajektórie. Tá istá funkcia potom znova doplní obálku o body predposlednej trajektórie od pravého vonkajšieho priesečníka po jej najvyšší bod. Takisto sú k predposlednej trajektórii v globálnej premennej doplnené indexy bodov trajektórie a indexy bodov obálky, ktoré obklopujú body trajektórie (tu je však koncový bod trajektórie totožný s koncovým obaľujúcim bodom obálky). Nakoniec je obálke pripojený posledný bod, ako najvyšší bod poslednej zvislej trajektórie a samotnej poslednej trajektórii sú doplnené indexy začiatočného a koncového bodu prispievajúceho obálke (keďže ide o jeden bod, sú indexy rovnaké) a začiatočného a koncového bodu obálky, ktorým prislúchajú (keďže ide o jeden bod, je to rovnaký index a to posledný bod obálky).

Cyklus pokračuje výpočtom najväčšieho horizontálneho rozmeru v útvare, ktorý tri trajektórie vytvárajú. Čiže v mieste, kde je útvar najširší, sa vypočíta táto šírka – horizontálna, pretože sa kladie dôraz na rozmer bunky rastra, ktorý je tiež v horizontálnom smere. Šírka útvaru musí potom byť menšia ako polovica bunky. V závislosti od konštelácie jednotlivých priesečníkov v útvare sa počíta táto šírka. Obrázok 15 zachytáva tri situácie, ku ktorým môže dôjsť. Na pravom okraji obálky pri najširších trajektóriách, kedy je pravý vonkajší priesečník nižšie, ako vnútorný priesečník, je najširšie miesto útvaru medzi vnútorným priesečníkom a prvou nasledujúcou trajektóriou. Potom na okraji trajektórií s vyšším uhlom výstrelu, kedy sa pravý vonkajší priesečník dostane vyššie ako vnútorný priesečník, je najširší rozmer medzi pravým vonkajším priesečníkom a druhou nasledujúcou trajektóriou. V treťom prípade, keď pri veľmi úzkych trajektóriách prevýši pravý vonkajší priesečník aj ten ľavý, je maximálna šírka útvaru medzi pravým vonkajším priesečníkom a prvou nasledujúcou trajektóriou. Algoritmicky je to riešené tak, že najprv prebehne podmienka, ktorá kontroluje, či pravý vonkajší priesečník nie je nižšie ako vnútorný priesečník. Ak áno, funkciou *calculate\_intersection()* sa vypočítajú a vrátia súradnice priesečníka prvej nasledujúcej trajektórie s horizontálnou úsečkou vystupujúcou od vnútorného priesečníka. Šírka tejto úsečky sa nastaví na horizontálnu odľahlosť vnútorného priesečníka a pravého vonkajšieho priesečníka, čo je dosť na to aby algoritmus vyhľadal relevantný priesečník. Funkcii sú teda predané body trajektórie a dva body úsečky, s čím funkcia dokáže pracovať. Potom je vypočítaná maximálna šírka úseku ako rozdiel súradníc X nového priesešníka a vnútorného priesečníka Ak sa preukáže, že pravý priesečník je vyššie ako vnútorný priesečník, ďalšou podmienkou sa zistí, či nie je aj vyšší ako ľavý priesečník. Ak nie, obdobne sa vypočíta priesečník horizontálnej úsečky prechádzajúcej pravým vonkajším priesečníkom s druhou nasledujúcou trajektóriou. Ak je vyššie, tak sa funkcii *calculate\_intersection()* predajú body prvej nasledujúcej trajektórie a body horizontálnej úsečky vo výške pravého vonkajšieho priesečníka s počiatočnou súradnicou X ľavého vonkajšieho priesečníka a konečnou súradnicou X priemeru tejto súradnice oboch vonkajších priesečníkov. Úsečka nevychádza priamo z pravého vonkajšieho priesečníka, pretože tento samotný bod leží na trajektórii a mohol by byť vyhodotený ako nežiadúci priesečník. Pre obidva prípady ľavého priesečníka, ktorý je vyššie ako vnútorný priesečník, sa rozdielom súradníc X nového a pravého vonkajšieho priesečníka vypočíta maximálna šírka úseku.



Obr. 15: Situácie vyhľadania najväčšieho horizontálneho rozmeru opomenutého útvaru

Po získaní maximálneho rozmeru úseku sa pristúpi ku kontrole tohto rozmeru, pretože slúži ako zastavovacie kritérium opakovania zhusťovania jedného úseku. Ak je väčší ako veľkosť polovice bunky, dôjde k novému opakovaniu cyklu, kde sa aktuálna trajektória zachová a prvá nasledujúca trajektória nadobudne úlohu druhej nasledujúcej a medzi ne bude vkladaná nová trajektória. Ak je už rozmer úseku menší ako polovica veľkosti bunky, vymaže sa prvá nasledujúca trajektória, pretože sa považuje za nadbytočnú. Funkciou *update\_envelope()* sa obálka aktualizuje o ďalšie body už dostatočne zhusteného úseku série trajektórií a aktuálnej trajektórii v prislúchajúcej globálnej premennej sa pripoja indexy začiatočného a posledného bodu trajektórie, ktorým sa obálka aktualizovala a začiatočný a koncový bod obálky, ktoré obklopujú využité body. Obálka je doplnená o body aktuálnej trajektórie od vnútorného priesečníka z predošlého cyklu po vnútorný priesečník aktuálneho cyklu. Keďže sa telo cyklu chýli ku koncu a teda je tesne pred novým opakovaním, do súradnicových parametrov vnútorného priesečníka predošlého cyklu zapíšu údaje z aktuálneho vnútorného priesečníka a index aktuálnej trajektórie sa zvýši o 1. Ak by prislúchala trajektória s týmto novým idexom poslednej trajektórii v sete (ktorá nemá počiatočný uhol výstrelu 90°), telo podmienky zabezpečí doplnenie obálky o body poslednej trajektórie po jej najvyšší bod a pre trajektóriu samotnú opäť pridanie indexov relevantných bodov trajektórie a obálky (koncový bod z trajektórie a koncový bod obálky sú tu totožné, resp. indexy odkazujú na totožný bod). Tu sa cyklus zastaví a program sa vráti do funkcie *throwshed()*.

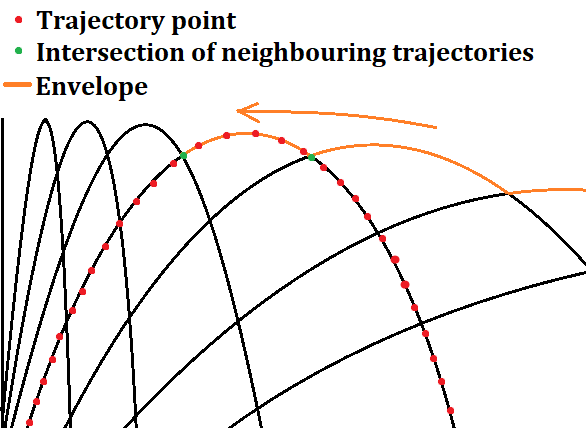
1.13 Funkcia calculate\_intersection()

Táto funkcia úzko spolupracuje s funkciou *trajectory\_set()*, v ktorej tele je volaná. Je to funkcia, ktorá berie za argumenty súradnice X a Y bodov dvoch línií ako zoznamy a vracia X a Y súradnicu priesečníka týchto dvoch línií. Može sa jednať o lomené línie s viac než dvomi bodmi (trajektórie) alebo úsečky len s dvomi bodmi.

Pomocou metódy *Geometry()* a argumentu *wbkLinearString* z modulu *OGR* je vytvorená prázdna geometria prvej línie a cyklom *for* sú priradené do tejto geometrie jednotlivé body zo zoznamu súradníc bodov prvej línie. Obdobne sa vytvorí aj geometria druhej línie. Funkciou *Intersection()* tiež z modulu *OGR* sa vyhľadá priesečník línií a nakoniec sú z geometrie bodu funkciami *GetX()* a *GetY()* získané a vrátené jednotlivé súradnice priesečníka.

1.14 Funkcia update\_envelope()

Táto funkcia taktiež vystupuje len v materskej funkcii *trajectory\_set()* a jej úlohou je pomocou niekoľkých argumentov (metóda výpočtu, trajektória, súradnice príslušných priesečníkov) aktualizovať obálku, ktorá spočiatku disponuje len jedným začiatočným bodom (posledný bod najširšej trajektórie), o body jednotlivých okrajových úsekov trajektórií, ktoré boli vyhodnotené za dostatočne zhustené až po najvyšší bod poslednej trajektórie. Body sú ukladané do globálnej premennej a hotová obálka bude neskôr použitá napríklad na vygenerovanie polygónov bezpečnej zóny. Pretože bude v neskorších algoritmoch potrebné poznať, ktoré trajektórie prispeli ktorými vlastnými bodmi k vytvoreniu obálky, funkcia taktiež vracia indexy začiatočného a koncového obálke odovzdaného bodu a indexy bodov z obálky, ktoré uzatvárajú body trajektórie z oboch strán (predtým boli priesečníkmi trajektóorií). Táto obálka má opačný priebeh ako trajektória, teda sprava doľava. Obrázok 16 naznačuje, ako dochádza k budovaniu obálky pomocou úsekov jednotlivých trajektórií.



Obr. 16: Tvorba algoritmickej obálky pomocou úsekov príslušných trajektórií

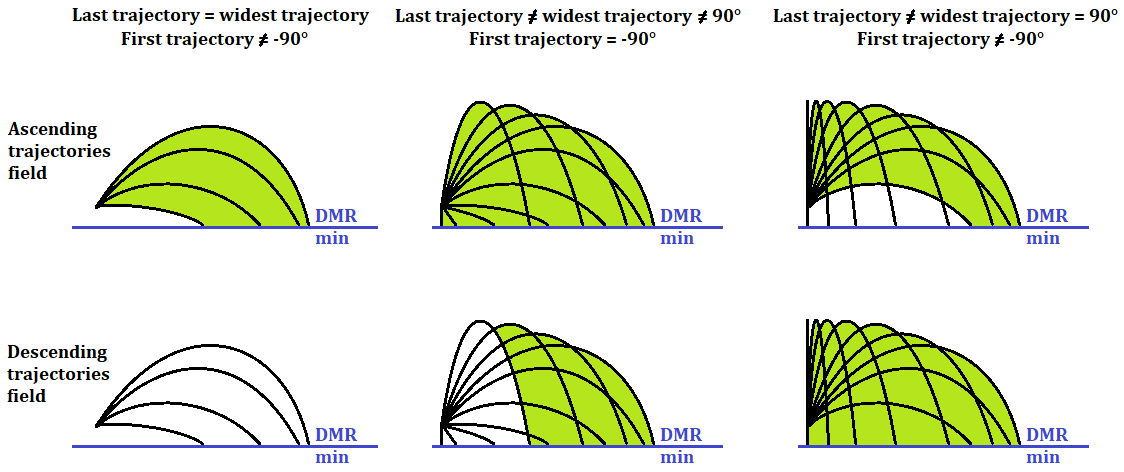
Telo funkcie definuje reverzný zoznam súradníc X a Y vstupnej trajektórie, ktorej body prispejú k budovaniu obálky. Nasledujúce cykly budú prechádzať jednotlivými bodmi zoznamu. V približne rovinatom teréne končia trajektórie asi vo výške miesta výstrelu. Ak je nejaká časť trajektórie, ktorá prispeje obálke, je to jej pravá strana (oddialenejšia časť trajektórie, úzke trajektórie prispejú svojím stredom) a navyše vplyvom odporu vzduchu je pravá strana trajektórie nesymetrická voči ľavej a má tendenciu veľmi rýchlo smerovať dolu. Preto je pravá strana trajektórie kratšia a disponuje menším počtom bodov. Preto sa urýchlili nasledujúce cykly, pristupuje sa k trajektóriám odzadu.

Nasleduje kontrola metódy, ktorá sa posiela ako vstupný parameter o hodnote 0 alebo 1. Rozlišuje sa ňou, či sa jedná o aktualizáciu obálky bodmi regulérnej časti setu trajektórií, alebo sa jedná o poslednú (pri uhle výsrelu poslednej trajektórie <90°), resp. predposlednú trajektóriu (pri uhle výsrelu poslednej trajektórie =90°). Ak sa jedná o všednú trajektóriu, spustí sa cyklus *for*, ktorý prechádza jednotlivé úsečky tvoriace trajektóriu odkonca a zisťuje, na ktorom z nich leží vnútorný priesečník z predošlého cyklu, teda bod, po ktorý sa obálka aktualizovala v predošlom úseku pomocou predošlej trajektórie. Keď sa zistí, tak sa zapíše index ľavého bodu úsečky. Potom nasleduje obdobný cyklus, ktorý prechádza body trajektórie znova odzadu, ale tentokrát až od bodu, ktorý predchádza bodu s uloženým indexom v predchádzajúcom cykle. Tentokrát sa hľadá, na ktorej úsečke leží aktuálny vnútorný priesečník. Zapíše sa opäť index bodu tentokrát pravého z úsečky, na ktorej priesečník leží. Tento cyklus sa nezačína od konca trajektórie, pretože sa dá predísť zbytočnému prechádzaniu cyklu cez časť trajektórie, v ktorej je zrejmé, že sa vnútorný priesečník nachádzať nebude. Začína sa už od bodu trajektórie, ktorý sa nachádza pred bodom, ktorého index sa uložil, pretože sa jedná o rovnakú úsečku, pri ktorom sa predchádzajúci cyklus zastavil. Je to tak preto, lebo pri raritnej konštelácii priesečníkov a nízkom rozlíšení trajektórie môže nastať situácia, kde oba priesečníky budú ležať na rovnakej úsečke trajektórie. Tu by sa obálka aktualizovala len o aktuálny priesečník a žiadne body trajektórie. Následne sa zapíše index posledného bodu obálky ako začiatočný index priesečníka z tejto trajektórie. Predtým získané indexy bodov trajektórie, ktoré sa nachádzajú medzi priesečníkmi, sú hneď aj využité v nasledujúcom cykle na doplnenie obálky o tieto body. Cyklus sa však nachádza v podmienke, kde musí byť splnená postupnosť indexov. Práve v spomínanej situácii priesečníkov ležiacich na jednej úsečke by indexy otočili svoje číslovanie a preto by sa cyklus doplnenia bodov do obálky nevykonal. K obálke sa ešte samostatne pridá aj súradnica aktuálneho priesečníka a zapíše sa index tohto bodu v rámci obálky. Takto boli vygenerované 4 indexy, z čoho 2 sú indexy začiatočného a koncového bodu v rámci trajektórie, ktorej body boli vložené do obálky a 2 sú indexy predošlého a aktuálneho priesečníka v rámci obálky, ktoré obklopujú použité body trajektórie. Tieto indexy sú vrátené a vo funkcii *trajectory\_set()* sú pridané na koniec premennej každej príslušnej trajektórie.

V prípade druhej metódy, teda aktualizácie obálky o body poslednej, resp. predposlednej trajektórie, sa definuje index začiatočného bodu tohto posledného úseku v rámci obálky. Predbežne sa zadefinuje aj hodnota indexu začiatočného bodu úseku v rámci trajektórie na 0, ale táto hodnota bude v cykle zmenená. Cyklus prechádza celou trajektóriou odzadu po bod, ktorý prevýši priesečník, po ktorý bola obálka aktualizovaná v predošlom cykle. Index tohto bodu prepíše hodnotu 0 spomínaného definovaného indexu a to podmienene tak, že v ďalšom opakovaní cyklu sa už táto hodnota nemôže prepísať. Od tohto bodu až ponajvyšší bod trajektórie budú body pridávané do obálky. Potom sa cyklus ukončí. Zadefinuje sa index posledného, najvyššieho bodu v rámci trajektórie a aj index koncového bodu úseku v rámci obálky. Jedná sa o rovnaký bod, ale indexy majú rozdielne hodnoty. Nakoniec sú indexy vrátené.

1.15 Funkcia create\_trajectory\_fields()

Táto funkcia generuje vertikálne polygóny bezpečnej zóny pre vzostupné aj zostupné trajektórie a ukladá ich do globálnych premenných. Najprv sa definujú prázdne premenné, ktoré neskôr poslúžia na uloženie vygenerovaných polygónov. Následne sa zadefinujú prázdne zoznamy týchto polí, ktoré budú naplnené súradnicami X a Y bodov, ktoré ohraničujú polia a poslúžia na generovanie geometrií polygónov. Nasleduje podmienka, ktorá zisťuje, či nie je obálka prázdna. Prázdna obálka bola zadefinovaná vo funkcii *trajectory\_set()* v prípade, kedy najširšia trajektória bola aj tou poslednou v sete. To znamená že je aj obálkou. V tele podmienky sa teda do poľa vzostupných trajektórií uložia body od začiatku po koniec prvej trajektórie, ktorá ohraničuje pole z jednej strany a ďalej sa sem pridajú body poslednej trajektórie od konca po začiatok, aby bola zachovaná postupnosť bodov proti smeru hodiných ručičiek. Dôležité je, že takto vytvorený zoznam súradníc X a Y začína aj končí v tom istom bode. Funkciou *create\_polygon\_from\_coords\_list()* sa zo zoznamu súradníc bodov vytvorí geometria polygónu poľa vzostupných trajektórií. Následne by sa funkcia ukončila a pole zostupných trajektórií by zostalo prázdne, pretože neexistuje v takejto situácii. V opačnej situácii, kedy najširšia trajektória nie je poslednou, sa spustí druhá časť podmienky, kde sa zadefinuje index najvyššieho bodu trajekórie, kde končí obálka. Do poľa vzostupných trajektórií sú ukladané body ohraničujúcej prvej trajektórie, obálky a naopak zoradených bodov poslednej trajektórie od bodu, kde obálka končí, na čo sa použije spomínaný index. Ďalej pri poli zostupných trajektórií platí, že ak nie je počiatočný uhol výstrelu poslednej trajektórie rovný 90°, hodnota indexu sa zachová a do poľa by vstúpili body obálky, poslednej trajektórie od bodu, kde končí obálka (smerom doprava po trajektórii) a nakoniec ešte raz prvý bod obálky, aby sa útvar uzavrel. V prípade počiatočného uhla výstrelu 90° pri poslednej trajektórii by sa hodnota indexu zmenila na -2, čo by následne s pripočítanou jednotkou zapríčinilo, že z poslednej trajektórie sa uloží len jej posledný bod, čo úplne postačuje, pretože celá táto posledná trajektória má zvislý priebeh. Nakoniec by sa funkciou *create\_polygon\_from\_coords\_list()* zo zoznamov súradníc bodov vygenerovali geometrie polygónov oboch polí. Obrázok 17 zachytáva vzhľad polí s niekoľkými nastaveniami minimálneho a maximálneho uhla výstrelu a ich kombinácie. Polia sú zvlášť vykreslené pre jednotlivé situácie.



Obr. 17: Vzhľad polí s kombináciou niekoľkých rozsahov uhla výstrelu

1.16 Funkcia create\_polygon\_from\_coords\_list()

Úlohou tejto funkcie je zo vstupného zoznamu súradníc X a Y vygenerovať geometriu polygónu a vrátiť ju do volajúcej funkcie. Využitá je vo funkcii *create\_trajectory\_fields()* a *cell\_availability()*. Najprv je vytvorená prázdna geometria prstenca funkciou *Geometry()* s argumentom definujúcim typ geometrie *wbkLinearRing* (obe z modulu *OGR*). Cyklom *for* sú pridávané postupne jednotlivé body zo zoznamu súradníc a to pomocou funkcie *AddPoint()*. Vytvorí sa prázdna geometria polygónu s argumentom *wbkPolygon* vo funkcii *Geometry()* a následne sa funkciou *AddGeometry()* vloží dopolygónu geometria prstenca. Polygón je vrátený.

1.17 Funkcia assign\_values\_to\_throwshed()

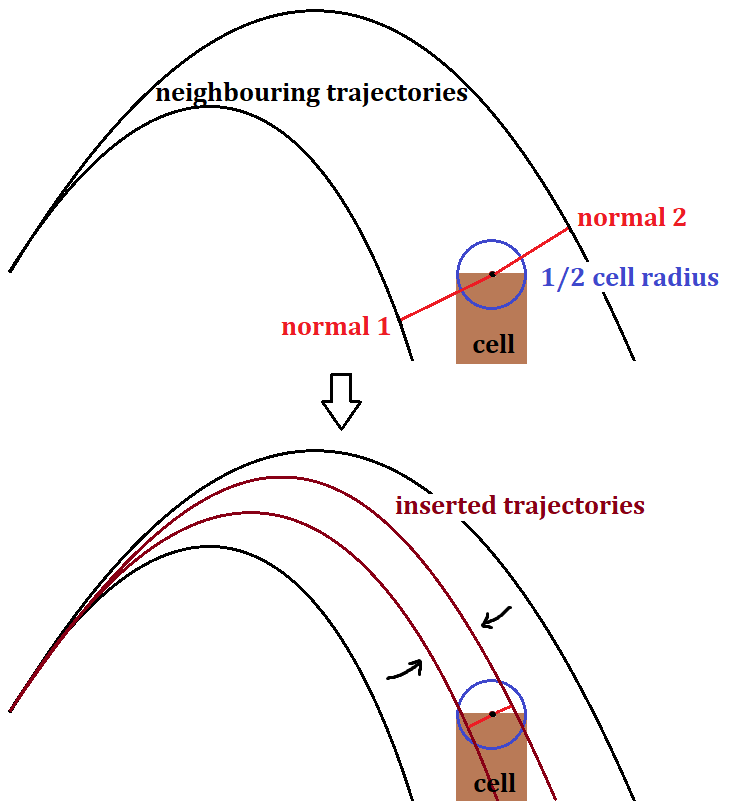
Táto funkcia mení hodnoty prvkov polí reprezentujúcich matice prislúchajúce k pásmam výsledného rastra „throwshedu“. Prvky, teda bunky rastra môžu nadobúdať hodnoty 1 až n, kde n je počet miest výstrelu a dve pásma rastra zodpovedajú dvom možným zásahom jednej bunky z jedného miesta výstrelu pomocou trajektórie s nižším a vyšším uhlom výstrelu. Hodnotí sa priestorový vzťah stredu každej bunky vzhľadom na predtým vygenerované vertikálne polia a podľa potreby sa hodnotí, či sa bunka nenachádza v zákryte.

Funkcia začne cyklom *for*, ktorý bude prechádzať všetkými riadkami poľa výsledného rastra a v tomto cykle je vnorený ďalší cyklus, ktorý bude prechádzať všetkými stĺpcami, teda vo výsledku – bunku po bunke podľa aktuálneho indexu. Vo vnorenom cykle skript začína podmienkou, či sa aktuálna hodnota bunky výsledného rastra nerovná hodnote *NoDataValue*, čo bude mať zmysel od výpočtu individuálneho throwshedu druhého miesta výstrelu. Ďalej pokračuje podmienka, že pre prípad, že prislúchajúca bunka DMR obsahuje hodnotu *NoDataValue* a jedná sa o výpočet „throwshedu“ pre prvé miesto výstrelu, tak sa zapíše táto hodnota aj do bunky rastra „throwshedu“ v oboch pásmach, pretože nemožno posudzovať príslušnosť bunky bez údaju výšky k sieti trajektórií. A preto je na mieste prvá podmienka, ktorá od nasledujúcich výpočtov „throwshedov“ pre danú bunku posunie cyklus na ďalšiu bunku ako aj táto podmienka, aby sa ponechala hodnota *NoDataValue*. Nasleduje podmienka pre prípad od druhého miesta výstrelu, kedy je možnosť kumulatívneho „throwshedu“ vypnutá a kedy už bunka z predošlého miesta výstrelu bola posúdená ako zasiahnuteľná, teda má hodnotu 1, čo posunie cyklus na novú bunku, pretože bunku z tohto miesta výstrelu už netreba posudzovať. Pre prípad vypočítaného rastra z analýzy viditeľnosti, ak je prislúchajúca bunka neviditeľná, nemusí sa zisťovať jej zasiahnuteľnosť a tak cyklus prejde k hodnoteniu ďalšej bunky. Po zohľadnení všetkých týchto situácií program vypočíta súradnice X a Y stredu aktuálnej bunky v súradnicovom systéme a tiež je vypočítaná horizontálna vzdialenosť stredu bunky od miesta výstrelu. Bunke je vytvorená geometria bodu v relatívnej a absolútnej variante. Relatívna geometria dostane súradnice spomínanej horizontálnej vzdialenosti a výšky stredu bunky v DMR – jedná sa o X,Y súradnice vo vertikálnej rovine trajektórií. Absolútna geometria dostane súradnice X,Y súradníc bunky v súradnicovom systéme DMR, teda v rovine kartografického zobrazenia.

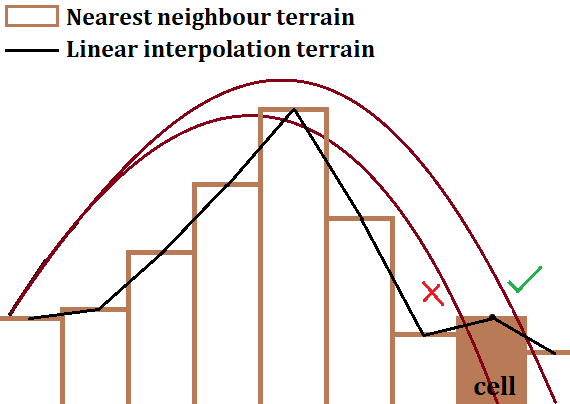
Nasleduje hodnotenie priestorovej príslušnosti geometrie relatívnej bunky k poliam trajektórií. Najprv sa kontroluje, či bod stredu bunky leží v polygóne poľa vzostupných trajektórií a to isté sa zopakuje pre pole zostupných trajektórií, len v tomto prípade predchádza ešte kontrola, či polygón tohto poľa nie je prázdny, čo nastane v situácii, kedy posledná trajektória setu je aj najširšou trajektóriou (žiadna bunka nie je zasiahnuteľná druhýkrát). Použitá je funkcia *Intersect()*, ktorá pozitívne vyhodnotí aj bunku, ktorej stred leží presne na obvode poľa. Ak tam neleží cyklus pokračuje k hodnoteniu ďalšej bunky, teda aktuálna bunka už viac nezaťažuje program, pretože je mimo dosah strelca. Keď pole obsahuje bunku, podľa módu „throwshedu“ sa buď funkciou *cell\_availability()* pristúpi k hľadaniu pretínajúcej trajektórie tejto bunky a hodnoteniu prístupnosti tejto bunky alebo sa rovno pripočíta hodnota 1 k bunke. Ak sa rovno pripočíta 1 bez hľadania pretínajúcej trajektórie, výsledný raster bude obsahovať všetky bunky z bezpečnej zóny, aj tie, ktoré sú neprístupné kvôli terénnym či iným prekážkam. Spomínaná funkcia vracia hodnotu *True* alebo *False*, čo vyjadruje prístupnosť bunky, teda či nie je náhodou zakrytá terénnou prekážkou, hoci sa v bezpečnej zóne nachádza. V prípade *True* sa pripočíta k hodnote bunky 1. V opačnom prípade prejde cyklus k novej bunke. Funkcii hodnotiacej prístupnosť bunky sú predávané geometrie bunky v relatívnej a absolútnej podobe a taktiež parameter, ktorý rozlišuje, či sa jedná o vzostupné alebo zostupné trajektórie.

1.18 Funkcia cell\_availability()

Úlohou funkcie je vygenerovať takú trajektóriu, ktorá pretína príslušnú bunku, pomocou inej funkcie zistiť, či nájdená trajektória nepretína terén skôr ako v cieľovej bunke a na základe toho vrátiť booleanovskú hodnotu *True* alebo *False*. Najprv sa v sete trajektórií nájdu tie trajektórie, ktoré obklopujú cieľovú bunku. Následne sa začnú posudzovať najkratšie kolmé vzdialenosti susedných trajektórií (normály) od bunky a keď budú dostatočne krátke, tzn. trajektórie budú prechádzať dostatočne blízko stredu bunky na to, aby boli považované za pretínajúce, tak sa ďalšou funkciou vyhodnotí, či pretínajúce trajektórie z jednej či druhej strany bunky nepretínajú terén skôr ako v tejto cieľovej bunke. Ak bude aspoň jedna susedná trajektória považovaná za pretínajúcu terén až v cieľovej bunke, tak sa cieľová bunka bude považovať za zasiahnuteľnú. Ak obe pretínajúce trajektórie pretnú terén skôr, ako v cieľovej bunke, tak za bude považovať za neprístupnú. Je málo pravdepodobné, aby normály k bunke boli hneď dostatočne krátke. Preto sa medzi tieto susedné trajektórie vkladajú nové, generované trajektórie, ktoré sa postupne z oboch strán približujú k stredu bunky, dokým ich normály nebudú dostatočne krátke alebo sa bunka nepreukáže byť prístupná na základe jednej z trajektórií. Najvhodnejšie by bolo poznať takú trajektóriu, ktorá pretína bunku presne v strede. Možné je však len postupne sa približovať k bunke v niekoľkých iteráciách, kde za zastavovacie krítérium tohto približovacieho procesu sa považuje opäť veľkosť polovice bunky, aby sa tak výpočet nezdržiaval príliš dlho v tejto časti skriptu. O susedných trajektóriách, ktoré sa vyhodnotia za dostatočne blízke, a teda pretínajúce, je isté len toľko, že sa nachádzajú len na opačných stranách bunky, teda delí ich stred bunky. Nie je známe, ako ďaleko od stredu sa nachádzajú a teda po vyhodnotení zasiahnuteľnosti, resp. neprístupnosti bunky, aká veľká časť bunky reálne zodpovedá výsledku. Napríklad, ak len percento oblasti bunky je zasiahnuteľné kvôli terénnej prekážke a približujúca sa trajektória pretne práve túto časť, celá bunka sa vyhodnotí za prístupnú. Nebezpečné to môže byť pri priebehu terénu, ktorý kopíruje priebeh trajektórie, kedy môžu vo výsledku vzniknúť napríklad striedajúce sa pásy prístupných a neprístupných oblastí, alebo veľká oblasť sa bude považovať za neprístupnú, pritom jednotlivé bunky môžu byť reálne z väčšej časti prístupné. Aby sa tomuto vyhlo, hodnotia sa až dve pretínajúce trajektórie. Avšak ani 10 pretínajúcich trajektórií by nedokázalo priniesť presný výsledok a tak, aby sa čas výpočtu skrátil, použije sa len štatisticky slabá vzorka dvoch trajektórií za predpokladu považovania bunky za prístupnú, keď čo i len jej zlomok je reálne zasiahnuteľný. Prístupov k tejto metodike však môže byť niekoľko, ale vybraný bol práve tento. Pre ukážku, obrázok 18 vykresľuje jednoduchý krok priblíženia v jednom kroku, kedy sú nájdené obe pretínajúce susedné trajektórie naraz. Hodnotenie prístupnosti bunky na základe týchto trajektórií znázorňuje obrázok 19. Jedna z pretínajúcich trajektórií pretína terén skôr, ale druhá až v cieľovej bunke a tak sa cieľová bunka vyhlási za prístupnú. Na obrázku taktiež vidieť dve rôzne reprezentácie terénu – z interpolácie metódou blízkeho suseda a lineárnej interpolácie.

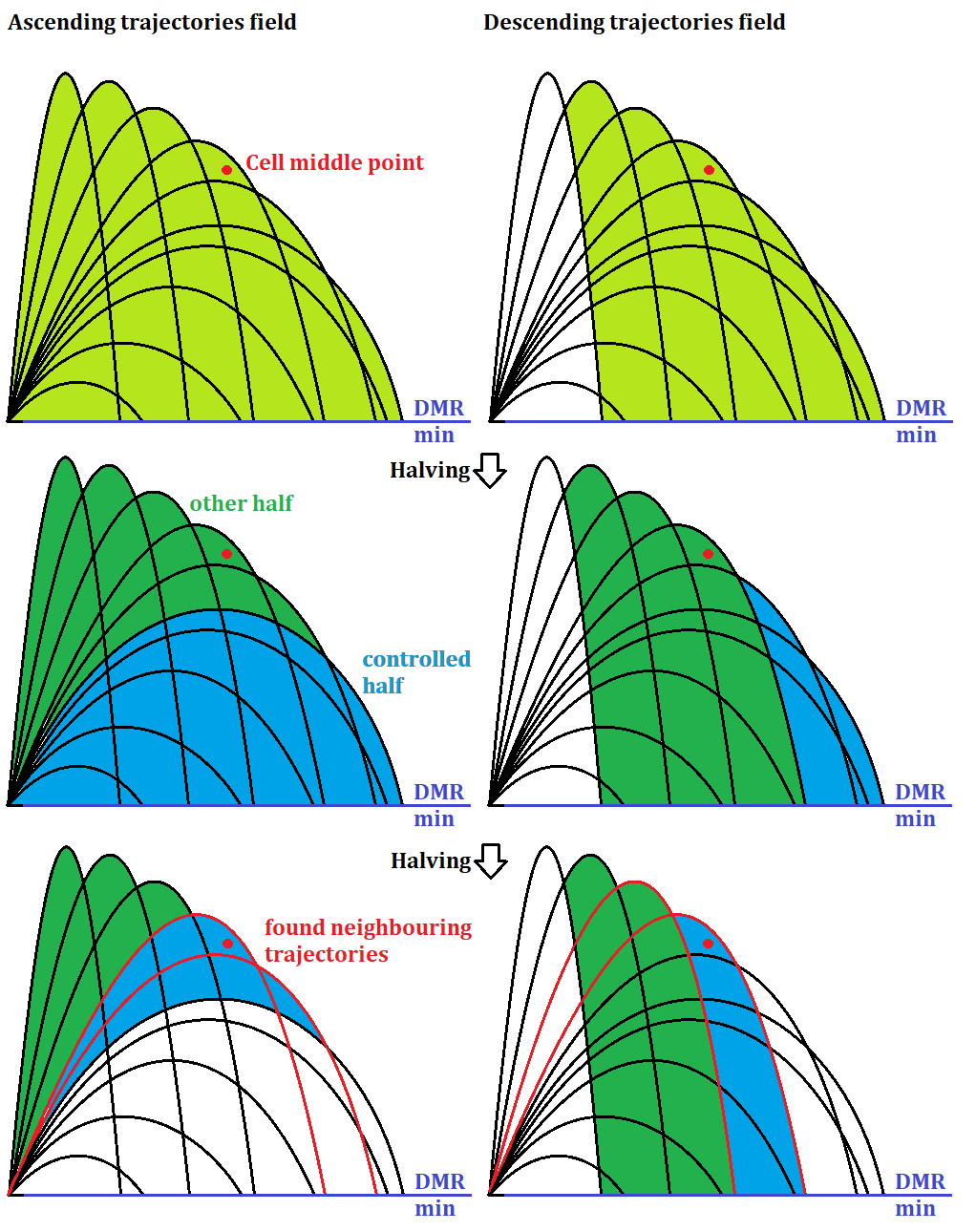


Obr. 18: Ukážka priblížovacieho procesu pri hľadaní pretínajúcej trajektórie



Obr. 19: Ukážka porovnania pretínajúcich trajektórií s terénnym profilom vedúcim k bunke

Funkcia najprv zadefinuje index najširšej trajektórie. Na základe parametra určujúceho, či sa hľadá vzostupná alebo zostupná pretínajúca trajektória bunky, sa určí krátky zoznam troch indexov určujúcich prvú a poslednú a medzi nimi prostrednú možnú trajektóru zo setu. Pre vzostupné trajektórie je to od prvej trajektórie po poslednú, pre zostupné trajektórie najširšiu až poslednú trajektóriu. V nasledujúcom cykle *while* sa budú hľadať dve susediace trajektórie, medzi ktorými bunka leží. Rieši sa to vytváraním polygónu, ktorého obvod tvoria dve trajektórie, prípadne aj obálka. Pre čo najrýchlejšie nájdenie najbližších susedných trajektórií sa k tomu pristupuje zbavením sa polovice nepotrebných trajektórií setu v každom opakovaní cyklu. Kontroluje sa najprv prvá polovica setu od prvej možnej po prostrednú trajektóriu, či medzi nimi bunka neleží. Ak áno, druhá polovica sa neposudzuje, ale prvá polovica sa rozpolí na dve časti a zisťovanie príslušnosti bunky a postupné priblíženie sa vykonáva, dokým rozdiel indexov susedných trajektórií nie je 1. Ak neleží bunka v prvej polovici poľa, musí automaticky ležať v druhej polovici a tak sa rovno rozpolí druhá polovica. V cykle *while* sa podľa typu hladanej trajektórie vygeneruje polygón funkciou *create\_polygon\_from\_coords\_list()*, do ktorej vstupuje vybudovaný zoznam bodov tvoriacich kontrolovanú polovicu oblasti. Tu okrem bodov okrajových trajektórií môžu vystupovať aj body obálky, ktoré je nutné zakomponovať. V prípade vzostupných trajektórií, kedy je druhá trajektória tou najširšou alebo tou s menším uhlom výstrelu sa polygón vygeneruje osobitne bez použitia bodov obálky. Program ďalej rozlišuje, či pri vzostupných trajektóriách najširšia trajektória nepredchádza použité trajektórie alebo nie je práve prvou trajektóriou. Individuálny prístup budovania polygónu si vyžaduje aj prípad, kedy najširšia trajektória je v prípade vzostupných trajektórií niekde medzi hraničnými trajektóriami. Po vytvorení polygónu sa funkciou *Intersects()* zistí, či bod stredu bunky neleží v polygóne polovice poľa trajektórií. Ak áno, zoznam indexov použitých trajektórií sa zmení na použitie prvej a prostrednej trajektórie z predošlého opakovania cyklu a medzi ne vloží novú. Ak neleží bunka v prvej polovici, automaticky musí ležať v druhej, čomu sa tiež prispôsobia indexy. Ešte sa skontroluje, či náhodou prvá a tretia trajektória v zozname nie sú už susedné, teda s rozdielom indexov 1, a ak áno, zadefinuje sa index predchádzajúcej susednej trajektórie. Obrázok 20 približuje tento proces.



Obr. 20: Ukážka vyhľadania susedných trajektórií bunky zo setu trajektórií

Nasledujúci algoritmus bude vkladať medzi nájdené trajektórie nové trajektórie postupným približovaním sa, dokým nepretnú bunku, aby boli následne porovnané s priebehom terénu, na základe čoho za zhodnotí prístupnosť bunky. Najprv sú zadefinované pomocné parametre, ktoré budú neskôr upravované. Spustí sa cyklus *while*, v ktorom sa najprv začne s vygenerovaním polygónu ohraničeného aktuálnymi susednými trajektóriami bunky a následne kontrolou, či bunka leží v tomto polygóne. V prvom priebehu cyklu sa môže zdať tvorba polygónu nadbytočná, pretože s využitím trajektórií zo setu a z výsledku predošlého algoritmu je jasné, kde bunka leží, ale v niektorých situáciách sa môže predošlý algoritmus skončiť tak, že síce index predchádzajúcej najbližšej trajektórie je správny, polygón vytvorený v predchádzajúcom algoritme môže byť z prvej polovice troch trajektórií a keďže tu bunka neležala, tak sa druhá polovica určila ako obsahujúca bunku a tomu bol zadefinovaný príslušný index trajektórie. Ale polygón sa zachoval nesprávny, z prvej polovice. Preto ho tu treba znova vygenerovať s indexom správnej predchádzajúcej trajektórie. Pre tento prípad je aj koeficient udávajúci index poslednej trajektórie pred vkladanými trajektóriami zadefinovaný až v tele podmienky. Vkladané trajektórie sa totižto po posúdení prístupnosti bunky vymažú, pretože pri veľmi veľkých rastroch DMR s vysokým rozlíšením by vzniklo priveľa trajektórií, z ktorými by si musela pamäť PC poradiť. Tento koeficient zostane nemenný pozvyšok opakovaní cyklu. Nasleduje kontrola založená na využití dvoch pomocných parametrov. Prvým sa zisťuje, či už náhodou nebola vypočítaná z predchádzajúcej trajektórie normála, ktorá by mohla v ďalšom opakovaní cyklu zostať rovnaká, ak nová vložená trajektória zaujme úlohu nasledujúcej trajektórie. Predchádzajúca trajektória zostane rovnaká a teda aj jej normála a tak sa v takom prípade nebude znova počítať. Druhým parametrom sa zisťuje, či už predchádzajúca trajektória nepretla jednu stranu bunky v predošlom cykle, čo robí túto polovicu bunky už zhodnotenú a viac sa tu generovať pretínajúce trajektórie nebudú. Ak nebola nájdená ešte predchádzajúca pretínajúca trajektória a jej normála počítaná ešte nebola, pretože bola trajektória práve vložená, tak v tele podmienky sa vypočíta veľkosť normály z aktuálne predchádzajúcej trajektórie pomocou funkcie *compute\_normal()*. Ak je normála menšia ako polovičná veľkosť bunky, predchádzajúca trajektória sa považuje za pretínajúcu a funkciou *trajectory\_terrain\_comparison()*, ktorá vracia hodnotu *True* alebo *False*, sa zistí, či je bunka prístupná. Ak áno, pomocou určených koeficientov sa vymažú vložené trajektórie a funkcia vráti hodnotu *True* a teda už sa nehľadá ani nasledujúca pretínajúca trajektória, ak ešte nájdená nebola. V opačnom prípade sa v pomocnom koeficiente zaeviduje, že predchádzajúca pretínajúca trajektória už nájdená bola, ale pretína terén skôr, aby sa v nasledujúcom opakovaní cyklu už neposudzovala. Toto celé sa opakuje aj pre prípad nasledujúcej trajektórie od kontroly na základe dvoch pomocných koeficientov až po zaevidovanie situácie neprístupnosti bunky ani v jej druhej polovici v pomocnom koeficiente. Ak už predchádzajúca aj nasledujúc pretínajúca trajektória nájdené boli, ale obe vyhodnotili bunku za neprístupnú, podmienka to na základe evidujúceho pomocného koeficientu zistí, vymaže všetky vložené trajektórie a funkcia sa ukoční s navrátenou hodnotou *False*. V bežných opakovaniach ale ďalej cyklus pokračuje výpočtom pomeru na základe normál alebo sa zadefinuje ako jedna polovica. Bunka sa totižto nemusí nachádzať presne v strede medzi susednými trajektóriami a vygenerovanie novej trajektórie prechádzajúcej približne v strede medzi aktuálnymi susednými trajektóriami nemusí byť najrýchlejší spôsob hľadania pretínajúcej trajektórie. Preto sa do počiatočného uhla novej vloženej trajektórie pretaví pomer vypočítaný z normál, ktorý definuje kde asi približne sa bunka medzi susednými trajektóriami nachádza. V prípade ale, že už jedna pretínajúca trajektória nájdená bola, sa vygeneruje nová trajektória s pomerom jednej polovice, teda v strede obklopujúcich trajektórií, aby sa tak predišlo vzniku novej trajektórie na tej strane bunky, kde už pretínajúca trajektória existuje. Jedná sa o opatrnejšie približovanie. Nová trajektória je vložená medzi trajektórie setu a pomocné koeficienty sú prispôsobené. Mimo tela hlavnej podmienky, či bunka leží v polygóne, sa ešte na konci cyklu zvýši index prislúchajúci predchádzajúcej trajektórie, čo je potrebné, keď nová vložená trajektória rozdelí polygón na dve časti a kontroluje sa najprv, či neleží bunka v prvej časti. Ak áno, pôvodná predchádzajúca trajektória si ponechá svoju úlohu a vložená trajektória zaujme úlohu nasledujúcej trajektórie. Ak nie, index sa zvýši a vložená trajektória zaujme úlohu predchádzajúcej trajektórie a pôvodne nasledujúca trajektória sa zachová.

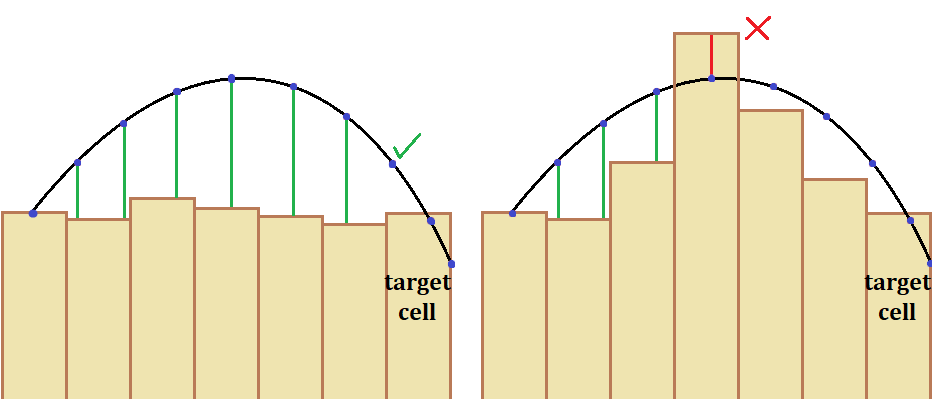
1.19 Funkcia compute\_normal()

Táto funkcia hľadá najkratšiu vzdialenosť medzi záujmovou trajektóriou a cieľovou bunkou. Hľadaná je teda veľkosť kolmice alebo normály z úseku trajektórie, ktorý je najbližšie k stredu bunky a zistený údaj vracia funkcii *cell\_availability()*.

Najprv je zoznamovou komprehenziou vytvorený list vzdialeností každého bodu trajektórie od stredu bunky a najkratšou vzdialenosťou sa určí index definujúci najbližší bod trajektórie. Pred definíciou indexu sa však ešte skontroluje, či jedna zo vzdialeností zo zoznamu nie je menšia ako polovica bunky, čo by znamenalo, že trajektória sa môže automaticky bez počítania kolmice považovať za pretínajúcu a tak sa vráti hodnota tejto vzdialenosti. V opačnom prípade ak najbližší bod trajektórie nie je v dostatočne blízkom okruhu stredu bunky, môže stále samotný úsek, teda línia spájajúca dva najblišie body trajektórie, pretínať blízky okruh stredu bunky a nájdenie veľkosti kolmice od tohto úseku po stred bunky bude vhodným parametrom rozhodnutia o tom, či trajektória pretína bunku. Preto sa nájde aj druhý najbližší bod k bunke, aby dotvoril spomínaný úsek a podľa toho, či je v poradí pred alebo za najbližším bodom sa pozmení index, ktorý musí ukazovať na bod úseku bližšie k začiatku trajektórii. Ak index neodkazuje na prvý ani posledný bod trajektórie, tak sa spustí ďalšia podmienka, kde sa skontroluje vzdialenosť bunky od predošlého a nasledujúceho bodu trajektórie. Ak je bližšie predošlý bod, tak sa určí ako hľadaný úsek od tohto bodu po najbližší bod a preto sa index zníži o 1, aby sa tak odkazovalo na ľavý bod úseku. Ak je bližšie nasledujúci bod, tak je cieľový úsek od najbližšieho bodu po tento druhý najbližší v smere po trajektórii a tak sa index nemení. Ak by sa jednalo o prípad najbližšieho bodu ako posledného z trajektórie, tak sa index zníži o 1, pretože len predposledný bod trajektórie bude druhým najbližším a teda ľavým bodom úseku. V ojedinelom prípade najbližšieho prvého bodu trajektórie sa index ponechá pretože len nasledujúci bod bude spolu s týmto vytvárať najbližší úsek. Index sa použije na zadefinovanie vzdialeností od najbližších dvoch bodov trajektórie a na výpočet dĺžky samotného úseku, čo dotvára trojuholník tvorený najbližšími bodmi a bodom stredu bunky. Herónovým vzťahom sa vypočíta obsah trojuholníka a z obsahu a dĺžky úseku je vypočítaná výška trojuholníka, teda normála, ktorá je aj vrátená a funkcia sa ukončí.

1.20 Funkcia trajectory\_terrain\_comparison()

Úlohou funkcie je vytvoriť terénny profil vedúci od miesta výstrelu po hodnotenú bunku a porovnať ho s priebehom trajektórie, ktorá pretína bunku. Ak táto trajektória pretne terén skôr, ako v cieľovej bunke, tak sa bunka považuje za neprístupnú, hoci sa nachádza v bezpečnej zóne. Terénne pomery v podobe prekážok nedovoľujú zasiahnutie bunky. V opačnm prípade sa bunka považuje za prístupnú. Porovnávať terén v podobe výškového modelu a trajektóriu v bodobe množiny bodov možno však len v určitom rozlíšení a to práve v rozlíšení trajektórie. Teda pre horizontálnu polohu každého bodu od začiatku trajektórie po bunku sa určí výška terénu a tá sa porovnáva s výškou bodu trajektórie.



Obr. 21: Detailné porovnanie trajektórie s terénnym profilom

Síce nie je možné vygenerovať cieľovej bunke trajektóriu, ktorá by prechádzala jej stredom, je možné aspoň na základe polohy miesta výstrelu a stredu cieľovej bunky nasmerovať trajektóriu pod správnym azimutom presne na stred bunky. Azimut sa počíta zo súradnicových rozdielov súradníc X a Y strelca a bunky a v prípade, kedy by bunka ležala presne na východ alebo západ od miesta výstrelu, je kód opatrený náležitou kontrolou. Taktiež je hodnota azimutu prispôsobená kvadrantu, v ktorom sa cieľová bunka nachádza. Najprv sa ale musia súradnicové rozdiely vypočítať. Hneď po ich výpočte nasleduje aj kontrola, či nenadobúdajú hodnotu menšiu ako polovica bunky, čo by znamenalo, že miesto výstrelu sa nachádza na cieľovej bunke. V takomto prípade sa bunka automaticky považuje za zasiahnuteľnú a funkcia vráti hodnotu *True*. V opačnom prípade po výpočte azimutu nasleduje cyklus *for*, ktorý bude prechádzať súradnicami každého bodu trajektórie od začiatku. Cyklus začína podmienkou, či už nie je relatívna súradnica X vo vertikálnej rovine kontrolovaného bodu trajektórie väčšia alebo rovná súradnici X stredu bunky, od ktorej je odčítaná polovica veľkosti bunky, teda odkazuje to na bližší okraj bunky. Ak sa ukáže, že bod trajektórie sa už nachádza súradnicou X v úrovni plochy bunky, funkcia sa ukončí s navrátením hodnoty *True*. Inak pokračuje cyklus na základe azimutu, relatívnej súradnice X bodu bunky (vzdialenosť) a polohy miesta výstrelu výpočtom absolútnej polohy kontrolovaného bodu terénu, kde jeho výška je interpolovaná funkciou *int\_function()* podľa zvolenej metódy interpolácie. Nakoniec cyklu sa kontroluje, či táto interpolovaná výška bodu terénu neprevyšuje výšku aktuálneho bodu. Ak nie, tak cyklus pokračuje na ďalší bod, ale ak áno, tak sa funkcia ukončí s vrátením hodnoty *False*. To znamená, že trajektória pretla terén skôr ako v cieľovej bunke a nemá zmysel porovnávať ostatné body trajektórie.

1.21 Funkcia create\_viewshed()

Táto posledná funkcia je volaná vo funkcii *throwshed()* v prípade, ked bolo využitie analýzy viditeľnosti aktivované. Zadefinujú sa globálne premenné, do ktorých sa uložia všetky potrebné parametre rastra viditeľnosti, ktorý je vytvorený *GDAL* funkciou *ViewshedGenerate()* a načítaný vlastnou funkciou *get\_raster\_from\_file()*. Pri volaní funkcie, ktorá ukladá raster v pamäti PC, sa použijú zadané parametre polohy a výšky očí strelca ako aj výška cieľa či hodnota *NoDataValue* prebratá z rastra DMR.

2 Modul numerical\_methods.py

Tento modul obsahuje funkcie generujúce a vracajúce zoznamy súradníc X a Y bodov trajektórií v ich vertikálnej rovine. Na výpočet sa využívajú numerické metódy s využitým diferenciálnych rovníc, kde podľa zvoleného rozmeru sú generované trajektórie eulerovou alebo heunovou metódou vo vertikálnej rovine, teda 2D, alebo 3D. 2D metódy zohľadňujú efekt tiažovej sily alebo odporu vzduchu, zatiaľ čo 3D metódy už aj vplyv vetra či Coriolisov a Etvosov efekt, ktoré majú význam pri streľbe na veľké vzdialenosti. Zohľadnený je napríklad aj faktor ako zmena teploty a hustoty vzduchu vplyvom narastajúcej nadmorskej výšky a to podľa jednej z vybraných typov atmosfér, ktoré sa v balistike používajú. 3D úlohy zatiaľ však využitie nemajú, pretože takto vyprodukovaným trajektóriám ešte nie je prispôsobená metodika posudzovania zasiahnuteľnosti buniek. Okrem knižnice *numpy* využíva tento modul ešte konštanty z modulov *atmospheres.py* a *drag\_tables.py*.

2.1 Funkcia interpolate\_C\_D()

Táto krátka funkcia počíta koeficient aerodynamického odporu na základe argumentu ukazujúceho na tabuľku koeficientov odporu a argumentu čísla Mach. Pri sonických a supersonických projektiloch je koeficient odporu vyjadriť funkciou namiesto konštanty. Vstupný parameter udávajúci koeficient alebo jeho funkciu môže byť tejto funkcii predaný ako reťazec znakov, ktorý odkazuje na názov konkrétnej tabuľky koeficientov pre jednotlivé čísla Mach, čiže odpovedá špecifickej zbrani, ktorej koeficienty boli experimentálne získané v laboratóriách na to určených. V tomto prípade je z modulu *drag\_tables.py* prevzatý zoznam koeficientov k príslušným číslam Mach. Tento parameter môže vstupovať aj ako zoznam zadaný používateľom. V oboch prípadoch sa koeficient interpoluje pre vstupnú hodnotu Mach. Tento parameter môže pri subsonických projektiloch vstupovať aj ako jedno číslo, celé alebo desatinné, vtedy sa táto hodnota ponechá a interpolácia sa nevykoná. Toto všetko je ošetrené systémom podmienok a prípadnou interpoláciou a funkciu uzatvára navrátenie hodnoty koeficientu odporu.

2.2 Funkcia euler2D()

Táto funkcia využíva Eulerovu numerickú metódu postupného budovania trajektórie od miesta výstrelu až pokým zastavovacie kritérium neukončí generovanie (keď body trajektórie nadobudnú výšku menšiu ako minimálna výška rastra výškového modelu). Používa sa v niekoľkých funkciách skriptu *throwshed2.py*, kde je vložený vrátený zoznam súradníc X a Y vo vertíkálnej rovine do globálnej premennej k prislúchajúcej trajektórii.

Funkcia začína výpočtom plochy kolmého prierezu projektilu z jej priemeru, ak nebola individuálne zadaná táto plocha. Následne sa definuje zoznam bodov trajektórie, ktorému sa rovno pridelí prvý bod so súradnicou X rovnou 0 a Y rovnou výške výstrelu. Nulové hodnoty sa predbežne pridelia aj parametru prejdeného času a krivočiarej vzdialenosti, ktorú projektil naprieč trajektóriou prejde. Vypočítaný je aj nekompletný odpor, kde vzťah neobsahuje koeficient odporu a ani rýchlosť, pretože tieto sa naprieč trajektóriou budú meniť, ale ostatné vystupujúce sa meniť nebudú, a tak aby sa navzájom v každom segmente trajektórie nenásobili, sú ponásobené tu do pomocného koeficientu, ktorý sa bude ďalej používať. Je to pre šetrenie výpočtového času. Vypočítané sú aj zložky rýchlosti z počiatočnej rýchlosti v smere osí X a Y vertikálnej roviny streľby. Nastavená je šírka segmentu trajektórie na základe používateľom zadanej hodnoty a spustí sa cyklus budovania nových bodov trajektórie na základe diferenciálnych rovníc.

Cyklus začína predefinovaním šírky segmentu v prípade, že zadávaná veľkosť dielika bola zamýšľaná ako šikmá a nie horizontálna. Nasleduje pripojenie nového bodu trajektórie do zoznamu, ktorého súradnice sú vypočítané na základe zložiek rýchlosti, horizontálnej šírky segmentu a súradníc predošlého bodu. Čiže cyklom je vytvorený druhý a ďalšie body trajektórie. Nasleduje kontrola, či sa nejedná o kmitajúci, resp. rotujúci projektil, kedy treba aktualizovať čas a prejdenú vzdialenosť po trajektórii pripočítaním zlomku času, resp. dĺžky segmentu. Ak projektil nekmitá a považujú sa jeho charakteristické parametre za nemenné, čas ani krivočiara vzdialenosť nezohrávajú rolu. Nasleduje kontrola, či Y súradnica posledného vytvoreného bodu trajektórie ešte nie je menšia ako minimálna výška rastra výškového modelu, pod ktorou už nie je potrebné generovať trajektóriu. V tele podmienky sa potom nachádza ďalšia, ktorá zisťuje či tento bod ležiaci v alebo pod efektívnou výškou nie je len druhým bodom v zozname, čo môže nastať pri veľmi malých a záporných uhloch výstrelu a pri malej výške výstrelu nad terénom. Normálne by sa posledný bod na poslednom segmente trajektórie preinterpoloval na minimálnu výšku rastra, ale v tomto prípade sa druhý bod ponechá, ajkeď je pod danou výškou a celý cyklus sa ukončí. Orezanie trajektórie na minimálnu výšku modelu má zmysel pri neskorších trajektóriách, obzvlášť pri hľadaní najširšej trajektórie. Ak sa bod nachádza ale nad touto výškou, pokračuje výpočet veľkosti vektora rýchlosti na základe jeho zložiek, ďalej sa počíta prevýšenie medzi aktuálnym bodom trajektórie a miestom výstrelu, ktoré slúži na výpočet teploty v tejto relatívnej výške. Na to sú použité aj zadaná teplota v mieste výstrelu, či faktor poklesu teploty s prevýšením z modulu *atmospheres.py*. Faktor rýchlosti zvuku vo vzduchu je potom s touto teplotou použitá na výpočet rýchlosti zvuku v daných podmienkach a táto rýchlosť poslúži s rýchlosťou projektilu k výpočtu čísla Mach. Na základe tohto čísla a zadanej tabuľky koeficientov aerodynamického odporu je vyinterpolovaný koeficient odporu funkciou *interpolate\_C\_D()* pre aktuálne podmienky. Samozrejme, koeficient môže byť pre subsonické projektily zadaný ako konštanta. Vtedy sa funkciou vráti tá istá hodnota. Nasleduje kontrola, či je nejaká frekvencia oscilácie projektilu. Ak nie, znamená to, že projektil neosciluje a vypočíta sa parameter, kde je ponásobený pomocný koeficient už aj koeficientom odporu vzduchu a rýchlosťou projektilu. Nejedná sa však úplne o hodnotu zrýchlenia, resp. spomalenia, vplyvom odporu vzduchu, pretože rýchlosť nie je umocnená na druhú. Taktiež je táto hodnota prepočítaná podľa faktora poklesu hustoty so zmenou prevýšenia ku konkrétnej výške a nakoniec je celá hodnota podelená zložkou rýchlosti v smere osi X, aby sa tak určila zmena tohto koeficientu na jednotku vzdialenosti, v tomto prípade na 1 meter horizontálnej vzdialenosti. Na konci cyklu sa táto hodnota ešte ponásobí pre smer oboch osí zložkou rýchlosti v smere osi X, resp. Y, čím sa docieli tá druhá mocnina rýchlosti, hoci tu v dvoch zložkách. Pre smer Y je ešte zohľadnený vplyv tiaže na základe tiažového zrýchlenia a nakoniec sa pre nové zložky rýchlosti, ktoré sa použijú v ďalšom segmente, ponásobia tieto pomocné hodnoty šírkou aktuálneho dielika, čím sa určí zmena rýchlosti v aktuálnom segmente a po pripočítaní tejto zmeny k zložkám rýchlosti v aktuálnom segmente vzniknú hodnoty zložiek pre nový segment. Ak sa ale preukáže, že existuje frekvencia oscilácie projektilu, ktorú používateľ zadal, v poslednej podmienke sa počíta goniometrický koeficient na základe frekvencie a času. Tento udáva číslo od 0 po 1 v závislosti od pozície alebo fázy kmitu, v ktorej sa projektil nachádza. 0 znamená, že projektil je v základnej polohe s minimálnym odporom vzduchu a 1 znamená, že projektil je v jednom alebo druhom maxime kmitu (napr. stav najväčšieho ohnutia vystreleného šípu). V závislosti od zadanej vzdialenosti, po ktorú kmitanie slabne až úplne zanikne sa ešte tento koeficient môže ponásobiť lineárnym koeficientom, ktorý len udáva pomer prejdenej vzdialenosti a maximálnej vzdialenosti kmitania. Hodnota koeficientu klesá od 1 lineárne po maximálnu vzdialenosť. Ak by bola maximálna vzdialenosť vyjadrená nulou, program ráta s nemenným efektom kmitania až po koniec trajektórie. Ak prejdená vzdialenosť po trajektórii prevýši túto nenulovú maximálnu vzdialenosť kmitania, počítajú sa fyzikálne parametre ako klasicky bez kmitania. V opačnom prípade po ponásobení goniometrického a lineárneho koeficientu vznikne oscilačný koeficient a ním sa názobí rozdiel plochy kolmého prierezu projektilu v základnej a maximálnej polohe kmitu, čím sa určia príspevky k základným hodnotám koeficientu odporu a plochy kolmého prierezu. K základným hodnotám sa pripočítajú a príslušné fyzikálne parametre sú počítané pomocou týchto hodnôt až po nové zložky rýchlosti mimo tela podmienky.

2.3 Funkcia heun2D()

Táto funkcia taktiež generuje a vracia zoznam bodov trajektórie. Táto trajektória je ale budovaná Heunovou metódou, čo je vylepšená Eulerova metóda, druhého rádu, alebo tiež prediktor-korektor metóda. Rozdiel je v počítaní diferenciálnych rovníc pre aktuálny a aj nasledujúci segment Eulerovou metódou (prediktor), ktoré sa po spriemerovaní aplikujú pre výpočet aktuálneho segmentu (korektor). Táto metóda je skôr určená pre streľbu na veľké vzdialenosti, pretože presnejšie vystihuje reálnu trajektóriu. Obyčajná Eulerova metóda bohate postačuje pre streľbu na krátke vzdialenosti predovšetkým subsonickými projektilmi.

V tejto Heunovej metóde nemožno uvažovať s oscilujúcim projektilom a tak funkcia začína informačným opatrením, kedy je používateľ oboznámený s touto skutočnosťou, ak aj napriek použitiu Heunovej metódy zadal frekvenciu oscilácie. Inak je začiatok kódu funkcie takmer totožný s kódom funkcie *euler2D()*. Pred spustením cyklu je ešte predbežne zadefinovaná šírka nasledujúceho elementu, ktorá bude tiež v cykle použitá a menená a treba ju osobitne zadefinovať, pretože šírka aktuálneho elementu nie je so šírkou nového elementu totožná.

V cykle po možnom prepočítaní šírky dielika na základe šikmej dĺžky je vypočítaná aktuálna rýchlosť na základe jej zložiek a pokračuje sa výpočtom od prevýšenia po koeficient aerodynamického odporu, a vypočítajú sa neúplné parametre čakajúce na úpravu zložiek rýchlosti pre nasledujúci bod v aktuálnom segmente. Tieto zložky aj upravené sú a definujú predbežný vektor rýchlosti z nového bodu smerom do nasledujúceho segmentu. Z nich sa vypočíta aj veľkosť vektora rýchlosti, číslo Mach, interpolovaný koeficient odporu až po fyzikálne parametre čakajúce na aktualizáciu vektora rýchlosti. Tu sa ešte prípadne prepočíta šírka nového segmentu, ak bolo zadané rozlíšenie trajektórie v horizontálnom smere. Čiže, akoby dvakrát za sebou vykonaná Eulerova metóda aktuálneho a nasledujúceho segmentu. Vplyv neúplných parametrov z aktuálneho a nasledujúceho segmentu sa spriemeruje pri výpočte novej rýchlosti pravého bodu segmentu, čím sa uzatvorí aktuálny segment. Keďže sa jedná o zložky rýchlosti pravého bodu a poznáme aj zložky rýchlosti z ľavého bodu z predošlého segmentu trajektórie, pri výpočte súradníc pravého bodu sa aplikuje priemer týchto rýchlostí a zložky rýchlosti ľavého bodu v nasledujúcom riešenom segmente sa použijú tie z pravého bodu aktuálneho segmentu. Výpočtu súradníc predchádza niekoľko opatrení, kde sa prepočítava šírka segmentu alebo sa pri situácii, kde vertikálna zložka rýchlosti ľavého a pravého bodu majú opačný smer, zamietne aplikácia korektoru a súradnica pravého bodu sa určí ako pri Eulerovej metóde. Tento prípad môže nastať v mieste trajektórie, kde projektil prestáva stúpať a začne klesať. Je to tak, pretože aplikácia neúplných fyzikálnych parametrov je prispôsobená zadanej veľkosti segmentu, ktorá musí byť splnená pre zabezpečenie jednotného rozlíšenia trajektórie. Pred zadefinovaním zložiek vektora rýchlosti pre nový segment na konci cyklu sa vykoná kontrola, či už trajektória nepresiahla minimálnu výšku výškového modelu, ako to bolo aj v prípade funkcie *euler2D()*.

2.4 Exprerimentálne funkcie

Predmetný modul obsahuje aj funkcie *euler3D()*, *heun3D()* prispôsobené na generovanie zoznamov súradníc X,Y a Z bodov trajektórie. Teda súradnice X a Y tvoriace vertikálnu rovinu ešte dopĺňa súradnica Z, ktorá udáva odchýlenie sa projektilu od vertikálnej roviny. Pri odchýlení sa doprava od roviny nadobúda Z kladné hodnoty. Tieto funkcie sú teda 3D ekvivalentom 2D funkcií. K odchýleniu projektilu môže dôjsť za vplyvu Coriolisovho efektu, ktorý zohľadňuje rotáciu Zeme alebo za vplyvu bočného vetra, ktorý je tu tiež zohľadnený v smere všetkých troch osí. Funkcia *heun\_i()* je obdobou funkcie *heun2D()*, ktorá je prispôsobená ale pre imperiálne jednotky. Všetky tieto tri funkcie nájdu využitie až po prispôsobení kódu novej metodike zohľadňujúcej trajektóriu ako priestorovú, nie rovinnú krivku a po pridaní možnosti pracovať aj s imperiálnymi jednotkami.

3 Modul atmospheres.py

Tento modul obsahuje konštanty niekoľkých parametrov vždy po dvoch hodnotách resp zoznamoch hodnôt, kde prvá reprezentuje hodnotu používanú v metodológii atmosféry Army Standard Metro a druhá hodnoty z atmosféry ICAO, čo sú štandardné typy atmosfér používané v armáde a konkrétne v balistike. Hodnoty obsiahnuté v tomto module sú používané pri budovaní trajektórií funkciami v module *numerical\_methods()*. Konštanty zahŕňajú faktor poklesu teploty s prevýšením *K*, faktor rýchlosti zvuku vo vzduchu *a0f*, faktor poklesu hustoty vzduchu s prevýšením *h*. Tieto konštanty sú vyjadrené pre metrický aj imperiálny systém jednotiek.

4 Modul drag\_tables.py

Modul obsahuje slovník, kde k jednotlivým kľúčom v podobe názvu zbrane ako reťazec znakov prislúcha zoznam hodnôt koeficientu odporu vzduchu pre niekoľko čísel Machu podľa prislúchajúceho typu zbrane. Funkcie koeficientov odporu, resp. tabuliek odporu k Machu, reprezentujú hlavné typy projektilov G1 – G8, ale aj niektoré vybrané špecifické zbrane, resp. projektily do nich. Tieto zoznamy sa budú v budúcnosti aktualizovať o tabuľky ďalších známych projektilov, čo uľahčí používateľovi, ktorý chce použiť špecifickú zbraň, prácu s nástrojom. Na základe názvov kľúčov sú vyberané funkcie koeficientu pre následnú interpoláciu vo funkciách modulu *numerical­\_methods()*, kde sa generujú trajektórie projektilov.

Záver

Analýza „thowshed“ momentálne poskytuje rad funkcionalít, ktoré sú v použitom jazyku Python opatrené niekoľkými stupňami optimalizácie (využívanie vstavaných funkcií namiesto vlastných), čo má za následok rýchlejší výpočet výsledku analýzy a menšie nároky na výpočtovú techniku. Nakoľko je už len z algoritmického hľadiska celá analýza veľmi komplikovaná, s aktiváciou viacerých funkcionalít, predovšetkým však hodnotenia prieniku trajektórií s terénnymi profilmi (klasický „throwshed“), nemožno porovnávať rýchlosť výpočtu s bežne používanými priestorovými analýzami. Opatrenia na zmenšenie objemu kódu boli tiež aplikované, teda pre časti, kde sa vyskytovala rovnaká alebo podobná časť výpočtu, bola zostavená vlastná funkcia s príslušnými argumentami a podmienkami v tele funkcie rozlišujúcimi konkrétne využitie funkcie, ak to algoritmus vyžadoval. Program sa tiež niekoľkými spôsobmi snaží vyhnúť ojedinelým situáciám, či už z pohľadu algoritmického (raritné konštelácie trajektórií) alebo zlého nastavenia vstupných hodnôt používateľom (minimálny uhol výstrelu nastavený väčší ako maximálny), aby nedošlo k spadnutiu programu alebo naopak nekonečnému zacykleniu. Všetky tieto faktory vyzdvihujú spôsobilosť celého programu a aj napriek jedinečnosti samotnej analýzy, ktorá v takomto rozsahu ešte vyvíjaná nebola, porovnateľnosť s programami iných existujúcich priestorových analýz.

Predsa je ďalšie skracovanie kódu veľmi pravdepodobné ako aj ďalšia nevyhnutná optimalizácia. Po algoritmickej stránke vyžaduje program niekoľko aktualizácií ako aj pridaní nových funkcií odpovedajúcim novým funkcionalitám programu.