České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačové grafiky a interakce

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Václav Pavlovec

Studijní program: Softwarové technologie a management
Obor: Web a multimedia

Název tématu: Výpočet rozmístění popisků kolem 3D objektů

Pokyny pro vypracování:

Seznamte se s algoritmem pro výpočet rozmístění popisků kolem konvexní obálky 3D objektů [1] a modifikujte tento algoritmus tak, aby umožňoval rozmístění popisků přímo kolem siluet 3D objektů. Dále se seznamte s algoritmem pro výpočet rozmístění popisků pro panorama [2] a použijte tento algoritmus pro vylepšení rozmístění popisků v případě, že popisky jsou umístěny pouze nad a/nebo pod siluetou 3D objektů. Vaši implementaci otestujte alespoň na čtyřech 3D modelech, které obsahují alespoň 10 částí, pro které budou popisky rozmístěny. Pro každý z 3D modelů otestujte implementaci rozmístění popisků pro alespoň 4 různé pohledy na 3D model. Vaše výsledky porovnejte se stávajícím algoritmem [1].

Seznam odborné literatury:

[1] Čmolík, Ladislav, and Jiří Bittner. 'Layout-aware optimization for interactive labeling of 3D models.' Computers & Graphics 34.4 (2010): 378-387.

[2] Gemsa, Andreas, Jan-Henrik Haunert, and Martin Nöllenburg. 'Boundary-labeling algorithms for panorama images.' Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 289-298, ACM, 2011.

Vedoucí: Ing. Ladislav Čmolík, Ph.D.

Platnost zadání: do konce zimního semestru 2018/2019


prof. Ing. Jiří Žára, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 4.4.2017

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Bakalářská práce

# Výpočet rozmístění popisků kolem 3D objektů

# External labeling of 3D objects

Václav Pavlovec

Vedoucí práce: Ing. Ladislav Čmolík, Ph.D.

Studijní program: Softwarové technologie a management

Studijní obor: Web a multimédia

26. května 2017

## Poděkování

Rád bych poděkoval vedoucímu práce Ing. Ladislavu Čmolíkovi, Ph.D. za pravidelné konzultace naplněné cennými radami a připomínkami.

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 26.5.2017 ……

## Abstrakt

Tato bakalářská práce se zabývá problematikou automatického umístění popisků k objektu, konkrétně popisky externími (angl. External Labeling). Dále se tato práce věnuje optimalizaci rozmístění těchto popisků pro případy, kdy se popisky nacházejí nad nebo pod popisovaným objektem (angl. Panorama Labeling). Cílem této práce je seznámit čtenáře s algoritmy, které jsou pro tyto účely využity.

Pro každý z těchto dvou případů je nejprve provedena jeho analýza společně s představením algoritmu, kterým bude řešen. Na základě analýzy je realizována implementace požadované funkčnosti v projektu dodaném vedoucím práce. Výsledky této implementace jsou otestovány na čtyřech modelech a poskytují tak čtenáři vhled do rozdílů oproti původním verzím algoritmů.



## Klíčová slova

Externí popisky, automatické umístění popisků, panoramatické umístění popisků.



## Abstract

Main focus of this thesis is automatic labeling, namely external labeling. Furthermore, panorama labeling is utilized to improved results obtained from the algorithm for label positions above or below the labeled model. The aim of this theses is to present a reader with algorithms utilized for these purposes and to give an overview of obtained results.

At first both of these two techniques are analyzed. Subsequent implementation is carried out in a project provided by the thesis supervisor. Results of the implementation are tested on four models of different complexity and thus provide a reader with an insight into impact of modifications made to these algorithms.



## Key words

External labeling, automatic label placement, panorama labeling.

# Obsah


1 Úvod ... 1
1.1 Cíle práce ... 2
1.2 Struktura práce ... 2
2 Rozmístění popisků - analytická část ... 3
2.1 Terminologie ... 3
2.2 Techniky používané k umísťování popisků ... 4
2.2.1 Fixní kotva, fixní popisek ... 4
2.2.2 Plovoucí kotva, fixní popisek ... 4
2.2.3 Fixní kotva, plovoucí popisek ... 4
2.2.4 Plovoucí kotva, plovoucí popisek ... 4
2.3 Umísťování popisků kolem nekonvexních oblastí ... 5
2.4 Algoritmus automatického umísťování popisků ... 6
2.4.1 Kritéria umístění externích popisků ... 7
2.4.2 Simplifikace kritérií automatického umístění popisků ... 7
2.4.3 Výpočet vodící čáry ... 8
2.4.4 Pořadí zpracování popisovaných oblastí ... 10
2.4.5 Druhy rozložení popisků ... 11
2.4.6 Přehled algoritmu ... 11
2.5 Summed area table ... 12
2.5.1 Výpočet summed area table ... 12
2.5.2 Update summed area table ... 13
2.6 Technologie pro implementaci ... 14
2.6.1 OpenGL (Open Graphics Library) ... 14
2.6.2 GLSL (OpenGL Shading Language) ... 14
2.6.3 JOGL (Java OpenGL) ... 15
2.6.4 Tiger (Toolkit for Interactive Graphics rendERing) ... 15
3 Rozmístění popisků - návrh a implementace ... 17
3.1 Vstupní obraz pro SAT ... 18
3.2 Výpočet SAT ... 19
3.3 Pořadí zpracování jednotlivých oblastí ... 19
3.4 Diskvalifikace bodů siluety ... 19
3.5 Přepočítání SAT po umístění popisku ... 20
3.6 Umístění popisků do scény ... 21
4 Panorama – analytická část ... 23

4.1 Algoritmus panoramatického rozmístění popisků ..... 24
4.1.1 Optimalizace jedné řádky ..... 24
4.1.2 Panoramatické rozmístění popisků ..... 25
4.2 Detekce volného místa ..... 26
5 Panorama – návrh a implementace ..... 29
5.1 Získání obsazení scény ..... 29
5.2 Reprezentace scény pro panoramatické rozmístění popisků ..... 30
5.3 Rozmístění popisků ..... 31
5.3.1 Umístění popisků pod popisovaným objektem ..... 32
5.3.2 Umístění popisků nad a pod popisovaným objektem ..... 32
5.4 Rozmístění popisků s horizontem ..... 33
6 Testování ..... 35
6.1 Testování rozložení popisků kolem nekonvexní obálky ..... 36
6.2 Testování panoramatického rozložení popisků ..... 41
6.3 Rychlost vykreslení scény ..... 46
7 Závěr ..... 49
7.1 Možné pokračování práce ..... 50
8 Zdroje ..... 51

# Seznam obrázků

Obrázek 2.1:(a) Voroného diagram bodů na siluetě $A_I$. (b) Stohování popisků podél siluety. Zdroj [5]. ..... 5
Obrázek 2.2: Nemožnost umístění popisku do nekonvexní části objektu při nedostatku prostoru pro jeho akomodaci. Zdroj [8]..... 6
Obrázek 2.3: překryv popisku a objektu ..... 6
Obrázek 2.4: (a) Barvou vyjádřená id oblastí $Ai$. (b-e) Vizualizace členských funkcí, tmavší pixel reprezentuje méně pravděpodobného kandidáta na kotvu vodící čáry. (b) Délka vodících čar. (c) Významnost kotev. (d) Vzdálenost kotev po umístění jedné vodící čáry (zelenou barvou). (e) Vzdálenost koncových bodů $ei$ po umístění jedné vodící čáry (zelenou barvou). Zdroj [5]..... 10
Obrázek 2.5: Summed area table. Zdroj [7]. ..... 12
Obrázek 2.6: Algoritmus rekurzivního zdvojování v 1D. Zdroj [7]..... 13
Obrázek 2.7: Update summed area table při změně hodnot v odélníkové oblasti původního obrázku, zde vlivem umístění popisku. Pro body (a) napravo od oblasti, (b) napravo a nad oblastí, (c) nad oblastí, (d) pod nebo nalevo od oblasti, (e) v oblasti. Šedé plochy určují oblast, pro kterou je modifikace summed area table shodného druhu. Modré plochy znázorňují množinu určující velikost penalizace bodů příslušných šedých oblastí. ..... 14
Obrázek 3.1:(a)Textura počtu objektů, ze které byl získán (b) vstupní obraz pro SAT. ..... 18
Obrázek 3.2: (a) Integrální obraz - SAT, (b) zpětná rekonstrukce zdrojového obrazu ..... 19
Obrázek 3.3: Místo obsazené popisky, jak zaneseno v SAT ..... 21
Obrázek 3.4:Třídy balíčku tiger.effects.labeling.layout ..... 22
Obrázek 4.1: Maximalizace počtu popisků pro 3 řádky. Zdroj [19]. ..... 23
Obrázek 4.2: Popisky nad a pod při využití horizontů..... 23
Obrázek 4.3: Rozložení popisků nad objektem vzhledem k jeho hranici. ..... 24
Obrázek 4.4: Detekce volného místa ..... 26
Obrázek 4.5: Vyhodnocení obsazeného místa..... 27
Obrázek 5.1: Textura volného místa pro množinu koulí..... 30
Obrázek 5.2: Třídy balíčku tiger.effects.labeling.panorama.data..... 31
Obrázek 5.3: Třídy balíčku tiger.effects.labeling.layout..... 31
Obrázek 5.4: Protínání popisků a vodících čar ..... 32
Obrázek 6.1: Testované modely. (a) hlava, (b) trávící soustava, (c) sada koulí, (d) vrtačka, (e) vidlice kola..... 35
Obrázek 6.2: Torus. (A)- původní algoritmus, (B) nový algoritmus..... 36
Obrázek 6.3: Koule. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus ..... 36
Obrázek 6.4: Koule. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus ..... 37
Obrázek 6.5: Umístění popisku s uživatelem povoleným překryvem. ..... 37
Obrázek 6.6: Hlava.(a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus. ..... 38
Obrázek 6.7: Hlava. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus. ..... 38
Obrázek 6.8: Trávící soustava. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus. ..... 39
Obrázek 6.9: Trávící soustava. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus. ..... 39
Obrázek 6.10: Vidlice. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus. ..... 40
Obrázek 6.11: Vidlice. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus. ..... 40
Obrázek 6.12: Koule. (a) a (c) - původní algoritmus, (b) a (d) - panoramatické rozmístění novým algoritmem..... 41

Obrázek 6.13:: Koule. (a) a (c) - původní algoritmus, (b) a (d) - panoramatické rozmístění novým algoritmem ..... 42
Obrázek 6.14: Hlava. (a), (c), (e), (g) - původní rozmístění, (b), (d), (f), (h) - panoramatické rozmístění ..... 42
Obrázek 6.15: Trávící soustava. (a), (c) - původní rozmístění, (b), (d) - panoramatické rozmístění ..... 43
Obrázek 6.16: Trávící soustava. (a), (c) - původní rozmístění, (b), (d) - panoramatické rozmístění ..... 43
Obrázek 6.17: Vrtačka. (a), (c) - původní rozmístění, (b), (d) - panoramatické rozmístění ..... 44
Obrázek 6.18: (a), (c) - původní rozmístění, (b), (d) - panoramatické rozmístění..... 45
Obrázek 6.19: Upravená verze panoramatického rozmístění popisků. (a) hlava, (b) trávící soustava, (c) koule, (d) vrtačka. ..... 46
Obrázek 6.20: Závislost FPS na FrameTime. Zdroj [21]. ..... 46
Obrázek 6.21: Čas vykreslení scény ..... 47
Obrázek 6.22: Hardware. CPU (vlevo), RAM (uprostřed), GPU (vpravo) ..... 47

# Kapitola 1

# Úvod

V širokém spektru oborů se lze setkat s řadou komplexních objektů. Pro pochopení jejich funkce je nutná jejich vizuální reprezentace. Tímto způsobem však lze získat poznatky omezené na prostorové uspořádání jednotlivých částí objektu. Způsob, jakým spolu tyto části spolupracují, je pak nutné detailně popsat. Propojení této textové informace s informací vizuální je zajištěno pomocí popisků.

Popisky zvyšují porozumění studovanému objektu, jeho struktuře či funkci. Umístění popisků ovlivňuje správnost a snadnost pochopení daného popisovaného předmětu. Automatické umístění popisků v kombinaci s interaktivním prohlížením popisovaného modelu skýtá mnohé výhody. Možnost studia takovéhoto modelu z různých pohledů je mnohem ilustrativnější než pouhé statické obrázky.

Dle umístění lze popisky klasifikovat jako interní a externí. Tato práce se zabývá popisky externími. S externími popisky se lze setkat v široké škále případů a napříč spektrem oborů. Zástupnými příklady jsou případy, kdy pro umístění interních popisků na popisovaný objekt není dostatek místa, či by takovéto umístění snížilo míru přenášené vizuální informace.

Jelikož je problém umístění popisků NP-složitý [6], je tato úloha pojata jako optimalizace výsledku integrace textových elementů do rámce vizuální informace na základě stanovených kritérií. Definujícími prvky výsledného rozložení popisků jsou jejich pozice a pozice jim příslušných kotev – bodů v objektu, které slouží k propojení objektu s popiskem. Přístupy k této úloze se liší ve stupni volnosti definujících prvků a kritériích na ně kladených.

Existují algoritmy, které provádějí velmi rychlé optimalizační výpočty na GPU, přičemž uvažují flexibilní kotvy i popisky [5]. Popisky umísťují kolem konvexní obálky siluety 2D průmětu 3D objektu na obrazovku. V současné době však neexistuje algoritmus zahrnující flexibilní polohu kotev a popisků, který by byl realizovaný na GPU a umísťoval popisky přímo na hranici objektu. Umístěním přímo na hranici objektu lze dosáhnout zkrácení vzdálenosti mezi popiskem a popisovaným předmětem, díky čemuž lze snáze správně asociovat popisek s jeho objektem. Tato hranice je obecně nekonvexní. To má za důsledek, že nejsou splněny předpoklady, které potřebuje stávající algoritmus pro svou správnou funkci.

## 1.1 Cíle práce

Cílem této práce je analýza algoritmu automatického umísťování popisků [5] a jeho následná modifikace takovým způsobem, aby umožňoval umístění popisků přímo na hranici objektu. Tato hranice je obecně nekonvexní. Pro tento účel se seznámíme s datovou strukturou summed area table a zhodnotíme možnosti jejího použití při řešení prezentovaného problému. Na základě této analýzy bude provedena implementace v jazyku java za využití java knihoven JOGL a Tiger. Implementace bude mít formu úpravy již existujícího projektu, který využívá výše zmíněný algoritmus.

V navazující části práce si představíme algoritmus panoramatického umístění popisků, který prezentovali ve své práci Gemsa a kol. [19]. Zhodnotíme možnosti úpravy algoritmu pro scény, kdy je možné umístit popisky nad a/nebo pod popisovaný objekt s respektem k jeho hranici. Implementace využije získané poznatky pro optimalizaci rozložení popisků nad a/nebo pod popisovaným objektem.

## 1.2 Struktura práce

Práce je rozdělena do sedmi kapitol. V kapitole 1 se nachází uvedení do problematiky umísťování popisků společně s vytyčením cílů bakalářské práce a nastíněním struktury této práce. V kapitole 2 je analyzován algoritmus automatického umísťování popisků a je představena datová struktura summed area table, které bude dále využito k řešení potřebných úprav v algoritmu. Kapitola 3 přináší přehled a způsob implementace rozdílných přístupů oproti původní verzi algoritmu. V kapitole 4 je analyzován algoritmus panoramatického umístění objektů, který bude využit k úpravě rozmístění popisků nad a/nebo pod objektem. Kapitola 5 pak prezentuje způsob zapojení tohoto algoritmu do výpočtu rozmístění popisků. Důvodem přítomnosti dvou analytických částí (kapitola 2, 4) a dvou implementačních částí (kapitola 3, 5) je rozdílnost témat, které řeší. Tato organizace usnadňuje pochopení řešených problémů. V kapitole 6 jsou prezentovány výsledky, kterých bylo v rámci bakalářské práce dosaženo, skrze testování algoritmů na čtyřech modelech. Srovnáním nově dosažených výsledků s výsledky, které dávaly původní verze algoritmu, je poskytnut vhled do rozdílnosti výstupu obou algoritmů. Kapitola sedmá, a zároveň poslední, pak přináší zhodnocení celé práce a její možnou návaznost.

## Kapitola 2

# Rozmístění popisků - analytická část

V této kapitole jsou analyzovány komponenty a přístupy potřebné k dosažení vytyčených cílů práce. Pro tyto účely je nejprve zmapován stav oblasti zabývající se automatickým umísťováním popisků. V sekci 2.2 je provedena klasifikace utilizovaných přístupů v závislosti na fixnosti polohy kotev a popisků. Sekce 2.3 analyzuje důvody, proč nelze algoritmus popsaný ve zdroji [5] použít k automatickému umístění popisků přímo na hranici objektu. Sekce 2.4 popisuje modifikovanou verzi algoritmu automatického umísťování popisků [5]. Tato modifikace je hlavním fokusem této práce. Dále je v této části navrhnuto možné řešení. V sekci 2.5 je představena summed area table s využitím které je možné řešit nové problémy prezentované modifikovanou verzí algoritmu sekce 2.4. V sekci 2.6 jsou představeny technologie, které budou použity k implementaci.

## 2.1 Terminologie

Tato sekce definuje základní termíny, které jsou dále v textu používány. Porozumění těmto termínům je důležité pro následné pochopení popisovaných problémů.

Pro účely popisu je uvažován model M sestávající z n objektů $O_i$, $i \in \hat{n}$, $M = \bigcup_{i \in \hat{n}} O_i$, přičemž každému z objektů je přidělen unikátní popisek. Označení $\hat{n}$ představuje množinu $\hat{n} = \{1, 2, \dots, n-1, n\}$. Projekce objektu $O_i$ na obrazovku je zvána $A_i$. Vnitřní oblast projekce modelu M na obrazovku je definována jako $M_I = \bigcup_{i \in \hat{n}} A_i$. Dále je definována konvexní oblast obalující $M_I$.

$$A_I^{\kappa} = \{x | x \in A_s \wedge \varrho\left(x, \overline{M_I^{\kappa}}\right) < \alpha\}, \tag{2.1}$$

kde $A_s$ je oblast celého okna, $\varrho$ je metrika, $\alpha$ je parametr určující míru extruze oblasti $M_I^{\kappa}$ a

$$M_I^{\kappa} = \bigcap_{M_I \subset K \wedge K \text{ je konvexní množina}} K \tag{2.2}$$

Oblast $A_I^{\kappa}$ tedy představuje nejmenší konvexní množinu obsahující množinu $M_I$, rozšířenou o její α-okolí. Obdobně je definována oblast $A_I$, kdy místo oblasti $M_I^{\kappa}$ figuruje ve vztahu (2.1) oblast $M_I$. Hranice množiny $\partial A_I$ je zvána **siluetou** modelu M. Komplementární oblast množině $A_I$ vůči celkové oblasti obrazovky $A_S$ definujeme jako $A_E = A_S \setminus A_I$.

**Popisek** $p_i$ je textový prvek popisující objekt $O_i$. **Kotva** $a_i \in A_i$ je označení pro bod, který leží uvnitř projekce objektu $O_i$ na obrazovku. **Vodící čára** $l_i$ je spojnice kotvy $a_i$ a popisku $p_i$. **Koncový bod** této spojnice je označen jako $e_i, e_i \neq a_i$.

Popisky a kotvy jsou zvány jakou **plovoucí**, pokud jejich pozice není před výpočtem známa a je určena až algoritmem, který popisky či kotvy na obrazovku umisťuje. V případě že jsou pozice kotev či

popisků předem známy, algoritmus tedy určuje pouze permutaci popisků na předem známých místech, nazýváme je jako **fixní**.

## 2.2 Techniky používané k umísťování popisků

Přístupy výzkumných týmů k problematice automatického umísťování popisků jsou rozdílné. Čmolík [20] klasifikuje metody umísťování popisků na základě flexibility polohy kotev a popisků. Existují i přístupy, které nemohou být zařazeny do žádné z kategorií této sekce, avšak z jejich povahy jsou zaměřeny na řešení specifických problémů. Následující text této sekce mapuje různé, dříve dosažené, výsledky dle Čmolíka [20].

### 2.2.1 Fixní kotva, fixní popisek

Případ, kdy je kotva i popisek fixní, je nejméně komplikovaný z možných kombinací. Jelikož je umístění kotvy i popisku již předem známo, je dimenze příslušejícího prostoru řešení nižší než v případě opaku.

Bekos a kol. [2] definují ve své práci problém umístění popisků, kdy jsou popisky umístěny kolem obdélníkové oblasti. Oblast obaluje popisovaný objekt a již předem známé fixní kotvy. Permutace popisků a minimalizace součtu délek čar vedoucích od popisku ke kotvě pro různá uspořádání a velikosti popisků jsou stěžejním předmětem jejich optimalizace.

Další přístup představují ve své práci Benkert a kol.[1]. Pohlížejí na problém jako na úlohu dynamického programování, ve které optimalizují několik kritérií. Předmětem jejich optimalizace je délka vodících čar, počet jejich ohybů a jejich vzdálenost od kotev. Popisky jsou umísťovány jednostranně kolem obdélníkové oblasti. Oboustranné umístění popisků je řešeno zavedením dělicí čáry a rozpadem na dva problémy jednostranného umístění popisků.

### 2.2.2 Plovoucí kotva, fixní popisek

Bekos a kol. [17] rozšířili problém umístění popisků. Obdobně jako ve své předchozí práci je kvalita výsledného hodnocení určena součtem délek vodících čar. V tomto případě se však kotva může pohybovat uvnitř polygonální oblasti.

### 2.2.3 Fixní kotva, plovoucí popisek

Stein a Décoret [18] prezentovali hladový algoritmus realizovaný na GPU, kterým se snaží nalézt optimální rozmístění popisků ve scéně. K určení pořadí popisků využívají Apolloniův diagram, což je vlastně Voroného diagram s vahou iniciální množiny bodů. K zamezení překryvu vodících čar a popisků je využita summed area table.

### 2.2.4 Plovoucí kotva, plovoucí popisek

Případ plovoucí kotvy a plovoucího popisku je nejflexibilnější z možných kombinací. Umístění kotev a popisků zde může být chápáno jako jeden provázaný problém, či jako dva problémy, které se mohou jednosměrně ovlivňovat.

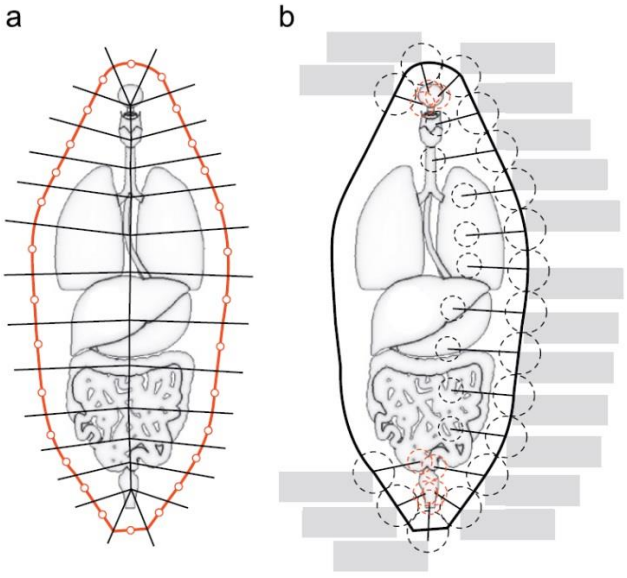
Hartman a kol. [3] představili metodu, kdy jsou nejprve spočteny polohy kotev a následně jsou dopočteny pozice jim náležících popisků. Pro určení kotvy objektu je objekt neustále zmenšován a limita této posloupnosti je její polohou. Ve výpočtu jsou použita pole dynamických potenciálů přitažlivých a odpudivých sil, přičemž popisky jsou umísťovány do bodů ekvilibria mezi nimi.

Ali a kol [4] představili algoritmus umísťování popisků počítaný v reálném čase. Řešení je opět dosaženo nalezením kotev a dopočtením jím příslušejících popisků. Algoritmus se řídí řadou kvalitativních kritérií na výsledek a umožňuje volbu stylu rozložení popisků kolem popisovaného objektu.

Čmolík a Bittner [5] prezentovali algoritmus operující v reálném čase, který komplexně přistupuje k nalezení kotev a popisků. Tento problém nedělí na nalezení kotev a následné dopočtení popisků, avšak oba prvky hledá současně a výsledné rozložení zpětně optimalizuje. Důraz je kladen na koherenci mezi po sobě následujícími snímky z hlediska rozložení popisků.

## 2.3 Umísťování popisků kolem nekonvexních oblastí

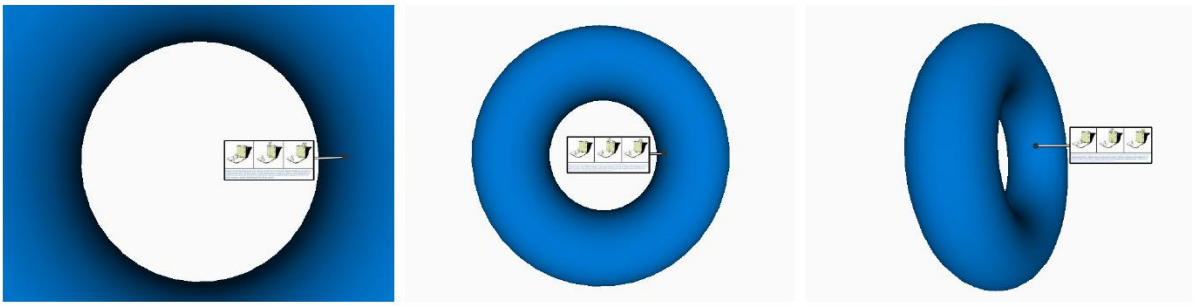
Cílem této práce je modifikace algoritmu umísťování popisků, utilizovaného balíčkem tiger.labeling tak, aby byly umísťovány na hranici objetu. Tato hranice je obecně nekonvexní. Problém je v mnohém podobný problému řešenému algoritmem zdroje [5], avšak přináší nutnost změny algoritmu s respektem k požadovaným výsledkům. Současná verze algoritmu [5] umísťuje popisky na siluetu popisovaného objektu. Díky konvexnosti této siluety a stohování popisků (viz obrázek 1b) kolem ní nemusí být řešen problém místa a případného zásahu popisku do objektu, či překryv popisků.




*Obrázek 2.1:(a) Voroného diagram bodů na siluetě $A_i$. (b) Stohování popisků podél siluety. Zdroj [5].*

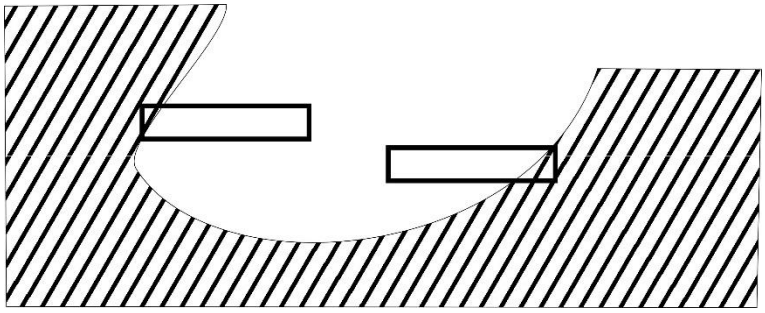
Stávající algoritmus tedy není schopen umístit popisek na nekonvexní hranici objektu z důvodu absence schopnosti determinovat prázdnost jeho průniku s popisovaným objektem, případně již

umístěnými popisky, jelikož není možné aplikovat přístup stohování popisků kolem siluety [4]. Tento problém je zachycen na obrázku 2.2 a 2.3.



*Obrázek 2.2: Nemožnost umístění popisku do nekonvexní části objektu při nedostatku prostoru pro jeho akomodaci. Zdroj [8].*

Na obrázku 2.2 je zachyceno umístění popisku do vnitřní části objektu s detekcí volného místa. K nevhodnému umístění však může dojít i při umístění popisku vně objektu. Tento případ je zachycen na obrázku 2.3, kdy obdélník reprezentující popisek je sice umístěn vně objektu na jeho hranici, avšak stále zasahuje svou malou částí do šrafované oblasti – objektu.



*Obrázek 2.3: překryv popisku a objektu*

Integrací kritéria, které by bylo schopné určit obsazenost oblasti, kam má být popisek umístěn, by byl tento nedostatek odstraněn. Pro tento účel je vhodné využití datové struktury nazývané summed area table.

## 2.4 Algoritmus automatického umísťování popisků

Tato sekce popisuje zobecnění algoritmu, který prezentovali ve své práci Čmolík a Bittner [5], na nekonvexní objekty. Algoritmus operuje s plovoucími kotvami a jejich příslušejícími, taktéž plovoucími, popisky. Pro správnou funkčnost publikované verze algoritmu musí být popisky umisťovány kolem konvexní obálky popisovaného objektu. Důsledkem tohoto přístupu je vyšší vzdálenost popisků od kotev, a tedy i popisovaného objektu. Cílem této práce je odstranění nutnosti konvexní obálky. V této sekci je popsána modifikovaná verze algoritmu. Vzhledem k velké shodě s původní verzí je na místech, kde se algoritmy liší, tak uvedeno. Nevyhnutelné je tedy použití velké části uveřejněného algoritmu [20], k čemuž udělil autor, Ing. Čmolík Ph.D., svolení. Tato sekce čerpá z výše uvedeného zdroje.

### 2.4.1 Kritéria umístění externích popisků

Rozložení popisků by mělo vykazovat čtyři základní vlastnosti – čitelnost, jednoznačnost, estetičnost a kompaktnost [20]. Jelikož umístění popisků a kotev představuje optimalizační problém, jsou nutná kritéria definující metriku pro srovnávání získaných řešení. Kritéria zavedená v popisovaném algoritmu [20] jsou následující:

(I) **Protínání vodících čar**: Vodící čáry se neprotínají. Opak by mohl působit konfuzi v tom, který popisek přísluší které kotvě.

(II) **Vzdálenost vodících čar**: Vodící čáry nejsou příliš blízko u sebe. Jejich přílišná blízkost by ztěžovala jejich asociaci se správným popiskem.

(III) **Uspořádání vodících čar**: Vodící čáry jsou uspořádány ve směru hlavních směrů, tj. předem definovaných směrů určujících validitu vodící čáry.

(IV) **Délka vodících čar**: Vodící čáry jsou tak krátké, jak je to jen možné. Čím delší jsou, tím obtížnější je přiřazení popisku správnému objektu.

(V) **Vzdálenost popisků**: Popisky jsou blízko k objektům, s kterými korespondují. Důsledkem je vyšší porozumění studovanému modelu.

(VI) **Překryv popisků**: Popisky se navzájem nepřekrývají. Překryv by omezil jejich čitelnost.

(VII) **Významnost kotev**: Kotvy jsou významné body oblasti, ke které příslušejí. Za významný bod je považován takový bod, jehož příslušnost dané oblasti je zřejmá. Typicky se jedná o body, jejichž vzdálenost od hranice oblasti $A_i$ je co nejvyšší

(VIII) **Vzdálenost kotev**: Kotvy $a_i$ nejsou příliš blízko sebe. Opak by ztěžoval jejich rozpoznání a asociaci s korektním popiskem.

(IX) **Vzdálenost koncových bodů**: Koncové body $e_i$ nejsou příliš blízko sebe. Tento požadavek koresponduje s požadavky na vzdálenost a překryv popisků.

S přihlédnutím k faktu, že si některá kritéria vzájemně odporují, je nutné nalézt jejich ekvilibrium.

### 2.4.2 Simplifikace kritérií automatického umístění popisků

Popisovaný algoritmus [20] determinuje současně pozice kotev a popisků. Tento přístup je složitější než alternativa určující tyto pozice postupně. Dimenze řešeného problému je tedy čtyři. Definicí jednoznačné vodící čáry pro každou kotvu je však možné dimenzi řešení snížit. Vodící čára je, dle kritérií (III) a (IV), určena jako nejkratší čára v hlavním směru, která spojuje kotvu a popisek. Pro vodící čáru $l_i = (a_i, e_i)$ je bod $e_i$ nejbližším bodem na siluetě $A_I$ bodu $a_i$ takový, že vektor $a_i - e_i$ je rovnoběžný s hlavním směrem. Tento přístup redukuje počet možných směrů vodící čáry na jediný (viz obrázek 2.1a), a tím také snižuje dimenzi problému na dva.

Takto definované vodící čáry se nikdy neprotnou, a proto může být kritérium (I) vypuštěno. Kritérium (IV) vyžaduje minimální délku vodících čar. Pokud jsou však vodící čáry krátké, implikuje tento fakt blízkost popisku a kotvy. Tím je tedy kritérium (V) již pokryto a může být samostatně vypuštěno. Jelikož vzdálenost vodících čar $l_i$ a $l_j, i \neq j$ je v tomto případě určena jako $\varrho(l_i, l_j) = \min\{\varrho(e_i, e_j), \varrho(a_i, a_j)\}$, může být kritérium (II) vypuštěno. Původní verze algoritmu umožňovala při splnění podmínky $\varrho(e_i, e_j) >$ výška popisku umístění popisků kolem konvexní siluety $\partial A_I^{\kappa}$ bez překryvu (viz obrázek 2.1 b) a vypuštění kritéria (VI). Tato simplifikace v modifikované verzi algoritmu již možná není.

Simplifikovaná množina kritérií tedy obsahuje všechna původní kritéria s výjimkou kritérií (I), (II), (V) a (VI), která jsou v uvažovaném přístupu redundantní.

**Simplifikovaná množina kritérií**

(1) **Uspořádání vodících čar**: Vodící čáry jsou uspořádány ve směru hlavních směrů, tj. předem definovaných směrů určujících validitu vodící čáry.
(2) **Délka vodících čar**: Vodící čáry jsou tak krátké, jak je to jen možné. Čím delší jsou tím obtížnější je přiřazení popisku ke správnému objektu.
(3) **Významnost kotev**: Kotvy jsou významné body oblasti, ke které příslušejí. Za významný bod je považován takový bod, jehož příslušnost dané oblasti je zřejmá. Typicky se jedná o body, jejichž vzdálenost od hranice oblasti $A_i$ je co nejvyšší
(4) **Vzdálenost kotev**: Kotvy $a_i$ nejsou příliš blízko sebe. Opak by ztěžoval jejich rozpoznání a asociaci s korektním popiskem.
(5) **Vzdálenost koncových bodů**: Koncové body $e_i$ nejsou příliš blízko sebe. Tento požadavek koresponduje s požadavky na vzdálenost a překryv popisků.
(6) **Překryv popisků**: Popisky se navzájem nepřekrývají. Dále popisky nezasahují do popisovaného objektu, jelikož by došlo k omezení jejich čitelnosti.

### 2.4.3 Výpočet vodící čáry

Pro každou oblast $A_i$ je nutné nalézt vodící čáru, přičemž každému bodu této oblasti přísluší unikátní kandidát na ni. Pro nalezení optimální vodící čáry je použita fuzzy optimalizace. Technika, která umožňuje simultánní naplnění několika i navzájem protichůdných kritérií.

Fuzzy teorie množin je rozšířením teorie množin [10]. Prvky ve fuzzy množině mají stupně členství. Fuzzy množina je dvojice $(U, m)$, kde $U$ je množina a $m: U \rightarrow [0,1]$ je funkce určující míru členství prvku v množině. Dále v textu je nazývána členská funkce.

Fuzzy optimalizace v popisovaném algoritmu [20] uvažuje prostor řešení $X$. Každé kritérium $C_i$ je modelováno jako fuzzy množina na tomto prostoru, s členskou funkcí $f_i$, která popisuje uspokojení kritéria (členství ve fuzzy množině) prvkem $x \in X$. Agregací kriteriálních funkcí pomocí fuzzy konjunkce je získána agregovaná funkce členství:

$$f(x) = \bigcap_{1 \le i \le 6} f_i(x). \tag{2.3}$$

Hodnota funkce $f(x)$ představuje realizovatelnost řešení $x$. Operátor průniku zaručuje současné naplnění všech kritérií. Neuspokojení jednoho kritéria tedy nemůže být kompenzováno uspokojením jiného. Nalezením globálního maxima funkce $f(x)$ je nalezeno nejlépe realizovatelné, tj. nejlépe kritéria respektující, řešení. V případě popisovaného algoritmu je prostorem řešení prostor kandidátů vodících čar oblasti $A_I$. Kritéria jsou vyhodnocována následujícím způsobem, přičemž je uvažována simplifikovaná množina kritérií:

**(1) - uspořádání vodících čar**: Kritérium (1) je implicitně splněno, jelikož kandidáty na vodící čáru jsou pouze čáry rovnoběžné s hlavními směry.

**(2) - délka vodících čar**: Délka vodící čáry $l_i = (a_i, e_i)$ je spočtena jako vzdálenost jejího koncového bodu od kotvy. Tato hodnota je následně normalizována největší délkou kandidáta vodící čáry v příslušné oblasti:

$$f_2(l) = 1 - \frac{|a - e|}{d_{max}}, \tag{2.4}$$

kde $|a - e|$ je vzdálenost bodů $a$ a $e$ a $d_{max}$ je délka nejdelšího kandidáta na vodící čáru. Průběh funkce je zachycen na obrázku 2.4(b).

**(3) - významnost kotev**: Významnost kotvy je určena jako její vzdálenost od hranice oblasti $\partial A_i$, normalizovaná délkou nejdelšího kandidáta na vodící čáru:

$$f_3(l) = \frac{dist_{sil}(a)}{d_{max}}, \tag{2.5}$$

kde $dist_{sil}$ je vzdálenost kotvy od hranice $\partial A_i$ a $d_{max}$ je nejdelší kandidát na vodící čáru. Grafická reprezentace této funkce je zachycena na obrázku 2.4(c). Je z něj patrné, že vhodnými kandidáty na kotvy jsou body ve středech příslušných oblastí.

**(4) - vzdálenost kotev**: Vzdálenost kotvy od ostatních kotev je určena jako její vzdálenost od již umístěných vodících čar $p \in P$. Vzdálenost je normalizovaná prahovou hodnotou $d_a$, která vyjadřuje nejmenší požadovanou vzájemnou vzdálenost dvou kotev. Vzdálenost kotvy kandidáta na vodící čáru $l$ a kotvy již umístěné vodící čáry $p$ je určena jako:

$$dist_a(l,p) = \min\{\frac{|a_l - a_p|}{d_a}, 1\}, \tag{2.6}$$

kde $|a_l - a_p|$ je vzdálenost mezi kotvou kandidáta na vodící čáru a kotvou již umístěné vodící čáry. Členská funkce, vizualizovaná na obrázku 2.4(d), je určena jako konjunkce vzdáleností od všech již umístěných kotev:

$$f_4(l) = \bigwedge_{p \in P} dist_a(l,p). \tag{2.7}$$

**(5) - vzdálenost koncových bodů**: Vzdálenost koncového bodu kandidáta na vodící čáru je určena jako jeho vzdálenost od koncových bodů již umístěných vodících čar $p \in P$. Dále je normalizovaná prahovou hodnotou $d_e$, která vyjadřuje minimální požadovanou vzájemnou vzdálenost dvou koncových bodů. Je-li dán kandidát na vodící čáru $l$ a již umístěná vodící čára $p$, pak je vzdálenost jejich koncových bodů určena jako:

$$dist_e(l,p) = \min\{\frac{|e_l - e_p|}{d_e}, 1\}, \tag{2.8}$$

kde $|e_l - e_p|$ je vzdálenost mezi koncovým bodem kandidáta na vodící čáru a koncovým bodem již umístěné vodící čáry. Členská funkce, vizualizovaná na obrázku 2.4(e), je určena jako konjunkce vzdáleností od všech již umístěných kotev:

$$f_5(l) = \bigwedge_{p \in P} dist_e(l,p). \tag{2.9}$$

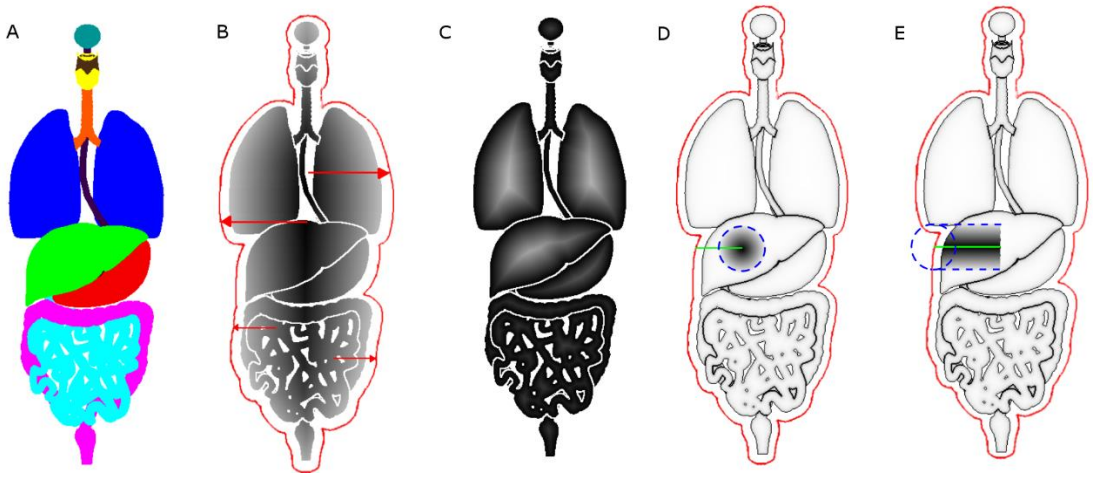
**(6) – překryv popisků**: Překryv popisků představuje kritérium, které penalizuje takové kandidáty na vodící čáry, jejichž umístěním do scény by došlo k překryvu jim příslušného popisku s popisovaným objektem či dříve umístěným popiskem.

$$f_6(l) = \begin{cases} \frac{1}{1+o_l} & o_l < t \\ 0 & jinak \end{cases}, \tag{2.10}$$

kde $o_l$ představuje průnik s neprázdným místem ve scéně a hodnota t prahovou hodnotu, nad kterou již kandidát není uvažován. V algoritmu [5] nebylo toto kritérium přítomno.

Agregovaná členská funkce je získána použitím fuzzy konjunkce jako operátoru průniku v rovnici (2.3):

$$f(l) = \bigwedge_{2 \le i \le 6} f_i(l). \tag{2.11}$$




*Obrázek 2.4: (a) Barvou vyjádřená id oblastí $A_i$. (b-e) Vizualizace členských funkcí, tmavší pixel reprezentuje méně pravděpodobného kandidáta na kotvu vodící čáry. (b) Délka vodících čar. (c) Významnost kotev. (d) Vzdálenost kotev po umístění jedné vodící čáry (zelenou barvou). (e) Vzdálenost koncových bodů $e_i$ po umístění jedné vodící čáry (zelenou barvou). Zdroj [5].*

Pro zavedení vah jednotlivých kritérií jsou použity funkce reflektující váhu jednotlivých členských funkcí ve výsledné agregované členské funkci:

$$F_i(l, w_i) = f_i(l)^{w_i}, \tag{2.12}$$

kde $f_i$ je členská funkce a $w_i \in [0,1]$ je její váha. Rovnice (2.3) je tedy upravena na:

$$F(l) = \bigwedge_{2 \le i \le 6} F_i(l, w_i). \tag{2.13}$$

### 2.4.4 Pořadí zpracování popisovaných oblastí

Algoritmus používá hladovou optimalizaci bez zpětného sledování. Kvalita výstupu je tedy závislá na pořadí, ve kterém jsou vodící čáry umístěny do ilustrace. Umístění vodící čáry do ilustrace omezuje možnosti umístění dalších vodících čar. Algoritmus operuje na předpokladu, že umístění vodící čáry do ilustrace ovlivňuje oblast s malým počtem kandidátů na vodící čáry více než oblast, která má těchto kandidátů mnoho. Je tedy definována priorita oblasti jako:

$$p_i = \sum_{l \in A_i} (1 - f(l)) \tag{2.14}$$

Pro váženou konjunkci pak:

$$P_i = \sum_{l \in A_i} (1 - F(l)) \tag{2.15}$$

Nejprve jsou vyhodnoceny priority jednotlivých oblastí a vodící čára je umístěna k oblasti s nejvyšší prioritou. Následuje rekalkulace priorit nepopsaných oblastí a umístění vodící čáry k ní. Tento cyklus se opakuje, dokud nejsou všechny oblasti popsány.

### 2.4.5 Druhy rozložení popisků

Rozložení popisků je určeno tvarem oblasti $A_I$ a množinou hlavních směrů $D$. Vodící čára je nejkratší spojnice kotvy a hranice $A_I$, která je rovnoběžná s některým z hlavních směrů $d \in D$. Rozložení popisků je tedy určeno tvarem hranice $A_I$ a směry vodících čar jsou určeny hlavními směry. Pokud není množina D omezena, tedy libovolný směr je směrem hlavním, je vodící čárou spojnice kotvy a hranice $A_I$. Rozložení popisků i směry vodících čar jsou v tomto případě určeny pouze tvarem hranice $A_I$.

### 2.4.6 Přehled algoritmu

Vstupem algoritmu [20] je 3D scéna sestávající z n 3D objektů $O_i$, $i \in \hat{n}$, sada hlavních směrů $D$, parametry $d_a, d_b, o_l, t$ a váhy $w_2, \ldots, w_6$. Výstupem algoritmu je množina vodících čar $L = \{l_i, \ldots, l_n\}$. Algoritmus postupuje následujícím způsobem:

Algoritmus 1: Automatické umístění popisků

1. Spočtením viditelnosti každého objektu ve scéně S získej množinu oblastí A = {$A_1$,…,$A_n$}.
2. Pro každý bod **a** uvnitř oblasti $M_I$ najdi nejbližší bod **e** na siluetě $A_I$ takový, že **e-a** je kolineární s hlavním směrem **d**∈D. Tímto je určena vodící čára a funkce $f_2$(**l**) je spočtena pro všechny body **a**∈$A_i$.
3. Spočti délku nejdelšího kandidáta na vodící čáru $d_{max}$.
4. Pro každý bod **a** oblasti $A_i$, $i \in \hat{n}$, vypočti $f_3$(**l**), tedy vzdálenost bodu **a** od hranice $A_i$.
5. Vytvoř množinu vodících čar L={}.
6. Vypočti přípustnost F(**l**)každé vodící čáry **l**=(**a**,**b**).
7. Vypočti summed area table pro scénu
8. Dokud není množina oblastí A prázdná, dělej:
   (a) Spočti prioritu $P_I$ pro všechny oblasti z A.
   (b) Vyber oblast s nejvyšší prioritou $A_{max}$.
   (c) Vyber nejlepšího kandidáta na vodící čáru $\mathbf{l}_{max}$ s maximální F(**l**).
   (d) Vlož kandidáta na vodící čáru $\mathbf{l}_{max}$ do množiny vodících čar L.
   (e) Proveď update summed area table
   (f) Odeber oblast $A_{max}$ z množiny A.
   (g) Proved update $f_3$ a $f_4$ pomocí $\mathbf{l}_{max}$ a znovu vyhodnoť F(**l**) pro všechny kandidáty na vodící čáry všech oblastí z A.

Algoritmus [5] v případě chybného rozložení popisků toto rozložení opravil. V modifikované verzi algoritmu již, díky znalosti volného místa na pozici kam je popisek umísťován, tato situace nenastává.

## 2.5 Summed area table

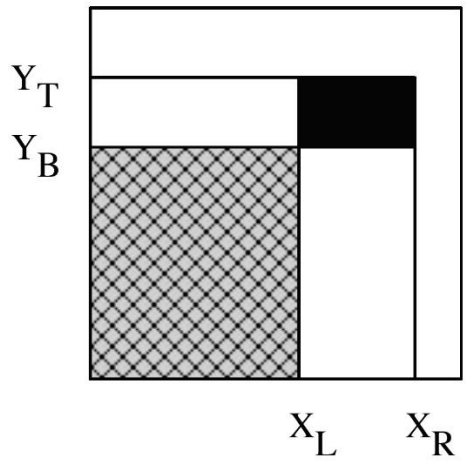
Summed area table, zkráceně SAT, je datová struktura a algoritmus pro efektivní nalezení součtu hodnot obdélníkové podmnožiny obdélníkové sítě. Frank Crow [9] ji poprvé představil v roce 1984 jako alternativu k mipmapám. Vstupem pro algoritmus summed area table je obdélníková síť, typicky se jedná o obrázek. Pro potřeby této práce bude dále v textu nazván jako vstupní obraz. Výstupem algoritmu SAT je obdélníková síť stejných rozměrů jako síť vstupní. Každý bod výstupu obsahuje součet hodnot obdélníku určeného souřadnicemi bodu samotného a levým dolním rohem vstupního obrázku. Z tohoto důvodu je SAT také nazýván jako integrální obraz. Hodnota bodu o souřadnicích (x, y) v summed area table je dána jako:

$$S(x,y) = \sum_{x'=0}^{x} \sum_{y'=0}^{y} i(x',y'), \tag{2.17}$$

kde $i(x,y)$ představuje hodnotu pixelu vstupního obrazu se souřadnicemi $(x,y)$.

Výpočet součtu hodnot obdélníkové oblasti (na obrázku 2.5 černý obdélník) je při znalosti summed area table zkonstruovanou nad příslušným obrázkem jednoduchou operací:

$$\sum_{\substack{X_L \le x \le X_R \\ Y_B \le y \le Y_T}} i(x,y) = S(X_R,Y_T) - S(X_R,Y_B) - S(X_L,Y_T) + S(X_L,Y_B) \tag{2.18}$$

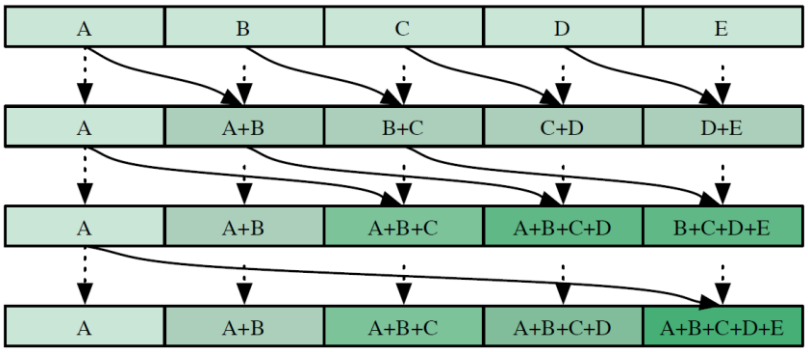



*Obrázek 2.5: Summed area table. Zdroj [7].*

### 2.5.1 Výpočet summed area table

Z definice summed area table je patrné, že její konstrukce nad daným obrázkem nepředstavuje implementačně složitý úkol. Pro potřeby této práce je však nutné, aby implementace byla co nejrychlejší z hlediska času potřebného k její konstrukci. Toho lze dosáhnout využitím vhodného algoritmu realizovaného na GPU.

Hensley a kol. [7] představili algoritmus umožňující generování SAT v $O(\log n)$ čase. K dosažení této rychlosti využili techniku známou jako rekursivní zdvojování. Na obrázku 2.6 je zachycen postup výpočtu pro 1D problém. V prvním kroku je ke každému elementu přičtena hodnota elementu nalevo od něj. Ve druhém kroku je k současné hodnotě přičtena hodnota elementu dva prvky doleva. Délka kroku je s každou další iterací zdvojnásobena. Pro 2D problém funguje algoritmus ve dvou fázích – vertikální a horizontální. Nejprve v horizontální fázi vytvoří součty řádků, následně pak ve fázi vertikální sloupcově sečte hodnoty získané z předchozí fáze. Tyto iterace probíhají do té doby, než je získáno řešení. Pro urychlení běhu algoritmu je využito dvou textur, které jsou po každém kroku prohozeny. Jedna z nich vždy slouží pro čtení a druhá pro zápis.




*Obrázek 2.6: Algoritmus rekurzivního zdvojování v 1D. Zdroj [7].*

### 2.5.2 Update summed area table

Změna hodnot obdélníkové oblasti, v případě této práce přidáním popisku, v původním obrázku má dopad pouze na část hodnot summed area table. Opětná kalkulace celé tabulky by tedy představovala časově redundantně nákladnou operaci. Obrázek (2.7d) zachycuje, že podstatná část tabulky být přepočítána nemusí. Výpočtem střední hodnoty podílu části tabulky, která musí být přepočtena, je možné získat pohled na výhodnost navrhovaného způsobu rekalkulace:

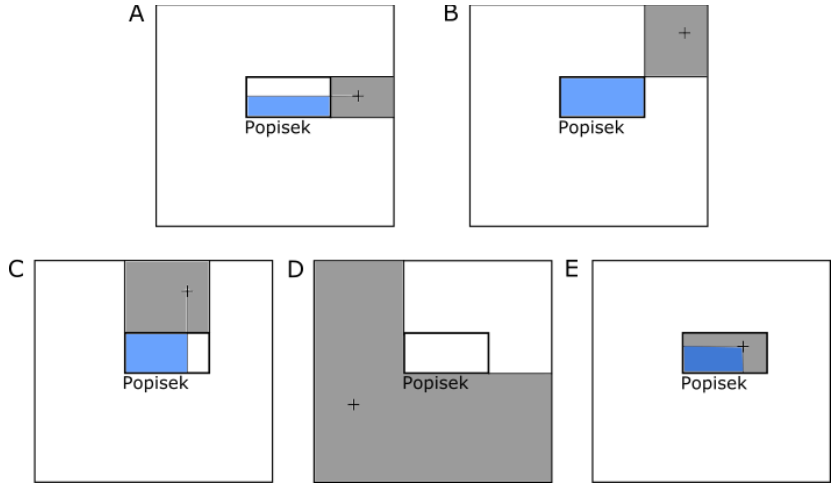
$$E(x * y) = E(x) * E(y) = \int_0^1 x dx * \int_0^1 y dy = 0.25, \tag{2.19}$$

kde $x$ a $y$ jsou souřadnice levého dolního rohu obdélníkové oblasti v obrázku a $E(x * y)$ představuje střední hodnotu plochy znázorněné bílou barvou na obrázku (2.7d) v poměru k ploše celého obrázku. Úprava hodnot SAT po změně hodnot v obdélníkové oblasti se tedy v průměru týká pouze její jedné čtvrtiny.

Dle polohy pixelu v SAT se způsob rekalkulace hodnoty rozpadá na pět možných případů, určující je přitom spolu s polohou pixelu $\boldsymbol{p} = (\boldsymbol{x}, \boldsymbol{y})$ pouze poloha levého spodního rohu $\boldsymbol{BL} = (\boldsymbol{BL_x}, \boldsymbol{BL_y})$ a pravého horního rohu $\boldsymbol{TR} = (\boldsymbol{TR_x}, \boldsymbol{TR_y})$ uvažované obdélníkové oblasti (viz obrázek 2.7):

$$\begin{aligned} S'(x,y) = S(x,y) &- S\big(\min\{TR_x, x\}, \min\{TR_y, y\}\big) \\ &+ S\big(\min\{TR_x, x\}, \min\{BL_y, y\}\big) \\ &+ S\big(\min\{BL_x, x\}, \min\{TR_y, y\}\big) \\ &- S(\min\{BL_x, x\}, \min\{BL_y, y\}) \\ &+ \sum_{x'=BL_x}^{\min\{TR_x,x\}} \sum_{y'=BLy}^{\min\{TR_y,y\}} i'(x', y'), \end{aligned} \tag{2.20}$$

kde $i'(x', y')$ představuje novou hodnotu pixelu v obrázku na pozici $(x', y')$, $S'(x, y)$ reprezentuje novou hodnotu pixelu v SAT a $S(x, y)$ jeho starou hodnotu. Vztah (2.20) je dále vhodnou volbou reprezentace místa ve scéně značně zjednodušen. Operace v tomto případě představuje jediný průchod shader programem.




*Obrázek 2.7: Update summed area table při změně hodnot v odélníkové oblasti původního obrázku, zde vlivem umístění popisku. Pro body (a) napravo od oblasti, (b) napravo a nad oblastí, (c) nad oblastí, (d) pod nebo nalevo od oblasti, (e) v oblasti. Šedé plochy určují oblast, pro kterou je modifikace summed area table shodného druhu. Modré plochy znázorňují množinu určující velikost penalizace bodů příslušných šedých oblastí.*

## 2.6 Technologie pro implementaci

V této sekci jsou popsány technologie, které budou využity při implementaci. Hlavní je knihovna Tiger, která pro svou funkčnost vyžaduje všechny další tři zmíněné technologie.

### 2.6.1 OpenGL (Open Graphics Library)

OpenGL je multiplatformní, jazykově nezávislé rozhraní (API) pro tvorbu 2D a 3D počítačové grafiky [11]. Jeho implementace existují prakticky pro všechny počítačové platformy. Kromě hardwarových implementací existují i implementace softwarové, které však zákonitě mají nižší výkon. OpenGL je vysoce oblíbená také díky podrobné dokumentaci, která je dostupná na jejích oficiálních stránkách. Základní funkcí OpenGL je vykreslování do framebufferu. Činnost OpenGL je řízená vydáváním příkazů pomocí volání funkcí a procedur.

### 2.6.2 GLSL (OpenGL Shading Language)

GLSL je vyšší programovací jazyk, vycházející ze syntaxe jazyka C. Ve skutečnosti se jedná o několik sobě blízkých jazyků. GLSL je použito k vytváření shaderů pro každou programovatelnou část zobrazovacího řetězce OpenGl, především vertex shader a fragment shader. Podobně jako jazyk C

podporuje cykly, větvení a programátorem definované funkce. Rekurze zde však není povolena. OpenGL poskytuje shader programům část svých stavů [16].

### 2.6.3 JOGL (Java OpenGL)

JOGL je obalová knihovna, která umožňuje použití OpenGL v Jave. K OpenGL přistupuje pomocí volání JNI (Java Native Interface). Nabízí přístup ke standardním GL a GLU (OpenGL Utility). Knihovna GLUT (OpenGL Utility Toolkit), která zaštiťuje volání oken, však dostupná není [13]. Důvodem je, že Java má pro tento účel své vlastní knihovny, a sice AWT, Swing a SWT. Dále poskytuje svoji vlastní NEWT (native windowing toolkit) [14].

### 2.6.4 Tiger (Toolkit for Interactive Graphics rendERing)

Tiger je Java knihovna, která usnadňuje vývoj víceprůchodového vykreslování v OpenGL. Knihovna přistupuje k OpenGl pomocí obalové knihovny JOGL [15].

## Kapitola 3

# Rozmístění popisků - návrh a implementace

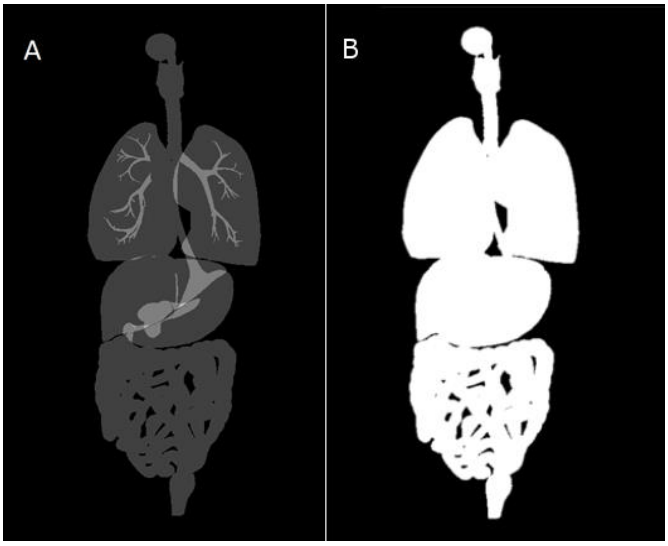
Tato kapitola popisuje přístupy k jednotlivým problémům, jejichž vyřešení je nutné pro správnou modifikaci algoritmu. Vždy je nejprve navrhnut způsob přístupu k danému problému a následně je popsána implementace realizovaná v rámci kódu projektu. Jelikož řešeným problémem je modifikace algoritmu pro automatické umísťování popisků, není zapotřebí, a není ani žádoucí, změna struktury tříd projektu. Majoritní část změn je omezena na modifikace kódu třídy ExternalLabeling.java umístěné v balíčku tiger.effects.labeling, společně s úpravou některých stávajících a vytvořením nových shader programů. V algoritmu 1 se jedná o kroky zvýrazněné tučně. Implementace byla realizována s verzí JOGL 2.2.2 a OpenGL 3.1.

Algoritmus 1: Automatické umístění popisků

1. Spočtením viditelnosti každého objektu ve scéně S získej množinu oblastí $A = \{A_1,\dots,A_n\}$.
2. Pro každý bod **a** uvnitř oblasti $M_I$ najdi nejbližší bod **e** na siluetě $A_I$ takový, že **e-a** je kolineární s hlavním směrem **d**∈D. Tímto je určena vodící čára a funkce $f_2(l)$ je spočtena pro všechny body a∈$A_i$.
3. Spočti délku nejdelšího kandidáta na vodící čáru $d_{max}$.
4. Pro každý bod **a** oblasti $A_i$, $i \in \hat{n}$, vypočti $f_3(l)$, tedy vzdálenost bodu **a** od hranice $A_i$.
5. Vytvoř množinu vodících čar L={}.
6. Vypočti přípustnost F(**l**)každé vodící čáry **l**=(**a**,**b**).
7. **Vypočti summed area table pro scénu**
8. Dokud není množina oblastí A prázdná, dělej:
   - **(h) Spočti prioritu $P_I$ pro všechny oblasti z A.**
   - **(i) Vyber oblast s nejvyšší prioritou $A_{max}$.**
   - **(j) Vyber nejlepšího kandidáta na vodící čáru $l_{max}$ s maximální F(l).**
   - (k) Vlož kandidáta na vodící čáru $\mathbf{l}_{max}$ do množiny vodících čar L.
   - **(l) Proveď update summed area table**
   - (m) Odeber oblast $A_{max}$ z množiny A.
   - (n) Proved update $f_3$ a $f_4$ pomocí $\mathbf{l}_{max}$ a znovu vyhodnoť F(**l**) pro všechny kandidáty na vodící čáry všech oblastí z A.

## 3.1 Vstupní obraz pro SAT

Vhodným nastavením hodnot vstupního obrazu pro algoritmus SAT lze docílit značného zjednodušení dotazů, které nad následně vytvořenou datovou strukturou budou prováděny. SAT slouží jako prostředek k zodpovězení otázky týkající se velikosti místa na dané obdélníkové oblasti scény. Na základě z ní získaných informací je učiněno rozhodnutí o diskvalifikaci či ponechání bodu siluety, který byl kandidátem na koncový bod vodící čáry. Dále jsou popisovány pouze hodnoty složky **r** barevných vektorů, jelikož informace jimi obsažená je pro všechny potřebné výpočty dostačující. Definicí prázdného pixelu hodnotou 0.0f a obsazeného pixelu jako 1.0f, tj. pixelu na který zasahuje zobrazovaný model, je získán vstupní obraz zachycený na obrázku (3.1b).




*Obrázek 3.1:(a)Textura počtu objektů, ze které byl získán (b) vstupní obraz pro SAT.*

Tento obraz lze získat implementací velmi jednoduchého algoritmu.

Algoritmus 2: Vstupní obraz pro SAT

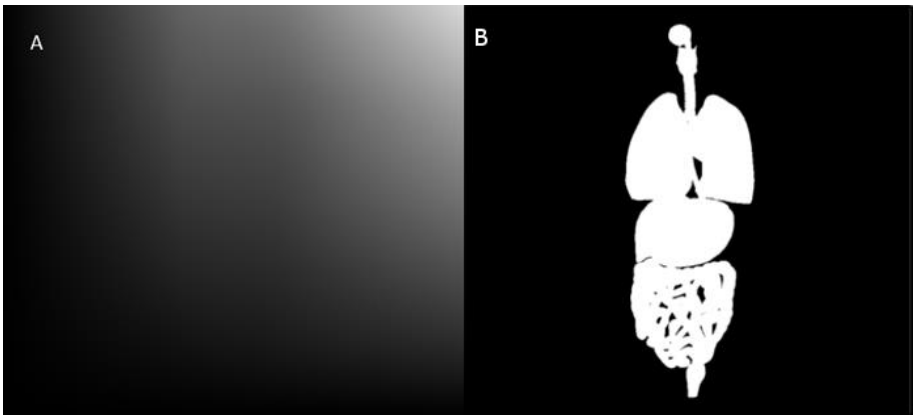
```
pokud je vstupní pixel neobsazený
 nastav hodnotu výstupního pixelu jako černá.
jinak
 nastav hodnotu výstupního pixelu jako bílá.
konec podmínky
```

### Implementace

Tato funkčnost je implementována jako shader SATDataPrepare.frag. Vstupem pro tento shader je textura countTexture (viz obrázek (7a)). Výstup je textura zachycená na obrázku (7b).

## 3.2 Výpočet SAT

Knihovna Tiger již ve svém balíčku tiger.util.sat obsahuje implementované třídy umožňující výpočet summed area table. Jako vstup pro SAT slouží textura popsána v sekci 3.1. Byla zvolena třída Sat2DJF, jelikož při testech na texturách inicializovaných na bílou barvu vykazovala nejvyšší rychlost výpočtu. Tato třída k výpočtu SAT používá algoritmus [7]. Po aplikaci Algoritmu SAT byla získána textura zachycená na obrázku 3.2.




*Obrázek 3.2: (a) Integrální obraz - SAT, (b) zpětná rekonstrukce zdrojového obrazu*

## 3.3 Pořadí zpracování jednotlivých oblastí

Původní verze algoritmu určovala pořadí zpracování popisovaných oblastí modelu dle váženého množství kandidátů na vodící čáru, jak zachycuje vztah (2.15). To bylo možné díky faktu, že popisek mohl být do scény vždy umístěn. Vlivem nedostatku volného místa však může být kandidát na vodící čáru mylně považován za platného. Je tedy nutné do součtu kandidátů dané oblasti zahrnout pouze ty, pro které je umístění vodící čáry s jejím popiskem možné. Oblast, ke které má být popisek umístěn v následující iteraci, je určena jako ta mající nejmenší součet dle vztahu (2.15).

Po umístění popisku je nutné provést rekalkulaci těchto sum, jelikož umístěním popisku do scény dojde ke změně volného místa v ní. Tím však může být pozměněno pořadí, ve kterém by byly oblasti popsány. V krajním případě může pak dojít až k nemožnosti umístění popisku pro některou z oblastí.

**Implementace**

Tato funkčnost je implementována v shaderech SphereAreaSum.vert a SphereAreaSum.frag.

## 3.4 Diskvalifikace bodů siluety

Diskvalifikace bodů z hranice modelu $\partial A_I$ pro popisky, které jsou příliš velké a zasahovaly by tak do modelu, je stěžejním problémem modifikovaného algoritmu automatického umísťování popisků. Původní verze algoritmu [5] uvažovala konvexní obálku, okolo které umísťovala popisky. Díky ní byla

podmínka dostatku místa pro umístění popisku implicitně splněna. K určení, zda může být bod na siluetě modelu koncovým bodem $e_i$ vodící čáry $l_i = (a_i, e_i)$, je využita SAT. Směr vodící čáry ovlivňuje polohu popisku vůči jejímu koncovému bodu (viz obrázek 2.1b). Směr vodící čáry je určen jako:

$$dir(l_i) = a_i - e_i. \tag{3.1}$$

Společně s rozměry popisku je jednoznačně určena oblast, kterou by popisek zabíral, pokud by do ní mohl být umístěn. Dotaz do textury SAT dle vztahu (2.18) vrací volné místo v této oblasti. Hodnota pixelu obrazu, ze kterého je konstruována SAT, je 0.0 právě tehdy, když je volný. Celá obdélníková oblast je tedy neobsazená, pokud je součet jejich pixelů roven nule. Uživatel může ovlivnit míru překryvu popisků s dalšími objekty ve scéně nastavení hodnoty slideru, konkrétně float slideru. Jeho hodnota je předána fragment shader programu a představuje prahovou hodnotu diskvalifikace (threshold) kandidáta na vodící čáru. Tento algoritmus zachycený pseudokódem vypadá následovně:

Algoritmus 3: Diskvalifikace bodů ze siluety

1. Pro vodící čáru $l_i(e_i,a_i)$ urči její směr jako $l_{dir} = e_i - a_i$
2. Podle polohy bodu $e_i$ a vektoru $l_{dir}$ urči polohu popisku s výškou **h** a šířkou **w** danou jeho levým spodním bl =(bl.x, bl.y) a pravým horním tr=(tr.x,tr.y) rohem.
3. Získej ze SAT ***s*** hodnoty a vypočti **x** = s(tr.x,tr.y)−s(tr.x,bl.y) − s(bl.x,tr.y)+ s(bl.x,bl.y)
4. **pokud x**<threshold
   ponech $l_i$ jako kandidáta na vodící čáru
   **jinak**
   diskvalifikuj $l_i$ jako kandidáta na vodící čáru

Po vyhodnocení přípustnosti všech kandidátů je vybrán nejlepší z nich a do scény je umístěn příslušný popisek.

### Implementace

Tato funkčnost je implementována jako součást shaderu max.vert a max.frag. Do shaderu max.frag vstupují uniform float proměnné reprezentující výšku a šířku popisku. Dále jsou jeho vstupem uživatelem nastavitelné uniform float proměnné, které určují prahovou hodnotu diskvalifikace fragmentu a míru penalizace kandidáta na vodící čáru, jejíž popisek by vykazoval překryv s dalšími objekty scény.

## 3.5 Přepočítání SAT po umístění popisku

Při umístění popisku k modelu se mění místo ve scéně, které je považováno jako volné, a tedy připadá v úvahu k umístění popisku. Pro následné korektní umísťování popisků, tj. bez jejich vzájemných překryvů, je tedy nutné přepočítat SAT. S přihlédnutím k faktu, že velká část SAT textury zůstává po této operaci stejná (viz sekce 2.5.2), je vhodné pouze aktualizovat tu část, která je umístěním popisku ovlivněna. Vzhledem tomu, že hodnota pixelu zdrojového obrazu pro SAT je 1.0 nebo 0.0, nastává při umístění popisku pouze změna hodnot ovlivněných pixelů z 0.0 na 1.0. Výhodou

těchto hodnot je jednoduché přepočítání SAT. Vztah (2.20) se značně zjednodušuje a nová SAT textura má hodnoty:

$$S'(x,y) = S(x,y) + \max\{0, (\min\{TR_x, x\} - BL_x) * (\min\{TR_y, y\} - BL_y)\}, \quad (3.2)$$

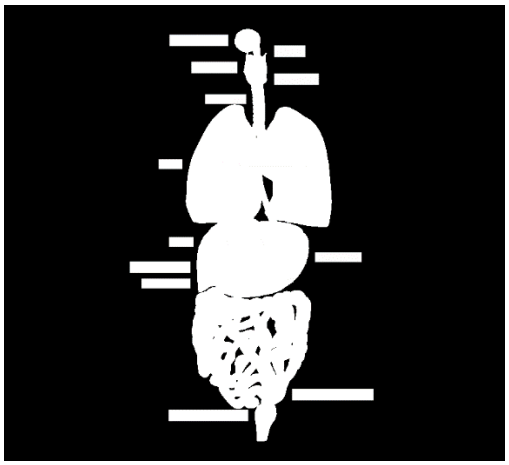
kde je $\boldsymbol{TR} = (\boldsymbol{TR_x}, \boldsymbol{TR_y})$ pravý horní roh, $\boldsymbol{BL} = (\boldsymbol{BL_x}, \boldsymbol{BL_y})$ levý dolní roh umístěného popisku a $p = (x, y)$ je souřadnice pixelu v SAT textuře. Souřadnice jsou uvažovány v pixelech. Hodnota $\max\{0, (\min\{TR_x, x\} - BL_x) * (\min\{TR_y, y\} - BL_y)\}$ představuje plochu modré oblasti na obrázku 2.7. Algoritmus přepočítání SAT vypadá následovně:

Algoritmus 4: Aktualizace SAT

```
1. Načti pro pixel p=(x,y) jeho hodnotu value ze SAT
2. Spočti polohy bodů levého spodního rohu popisku bl = (bl.x, bl.y)
   a pravého horního rohu popisku tr=(tr.x,tr.y).
3. Spočti vztah (3.2)
       delkaY = min (y,tr.y)-bl.y
       delkaX = min (x,tr.x)-bl.x
       newValue = value - max(0,delkaX*delkaY)
4. Zapiš hodnotu pixelu p=newValue v SAT textuře
```

**Implementace**

Tato funkčnost je implementována v shaderu SatUpdate.frag. Podoba summed area table pro průchodu shaderem je zachycena na obrázku 3.3.

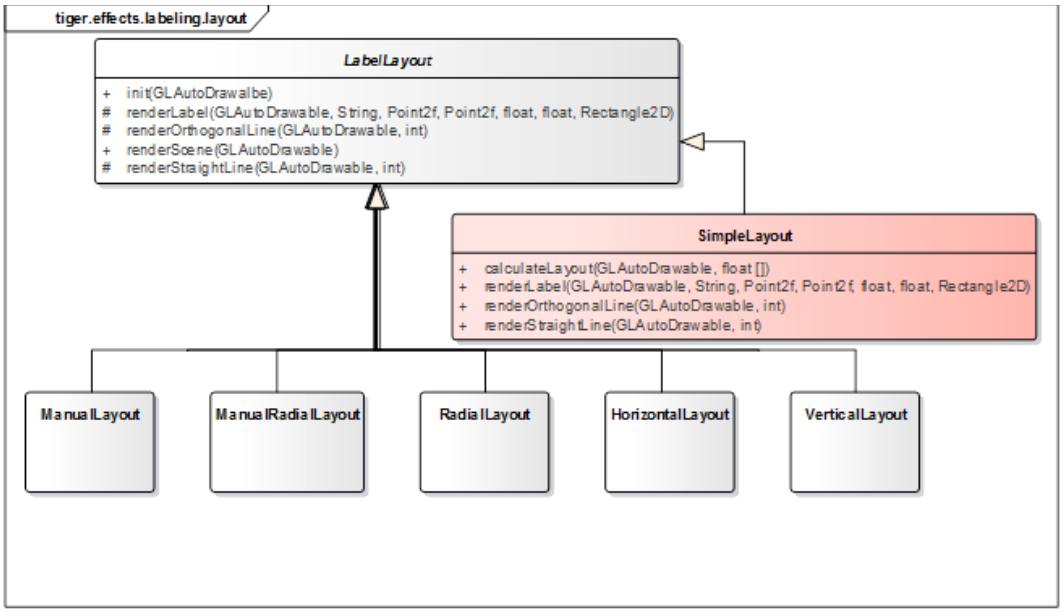


*Obrázek 3.3: Místo obsazené popisky, jak zaneseno v SAT*

## 3.6 Umístění popisků do scény

Množina hlavních směrů, se kterými algoritmus pracuje, určuje, jakým způsobem bude popisek do scény umístěn. Pro každou množinu je však pozice popisku vůči vodící čáře rozdílná. Je tedy nutné

zajistit, aby byl popisek vykreslován do scény způsobem, který je uvažován v shaderech SphereAreaSum.frag a max.frag při zjišťování volného místa. Pro tento účel je vytvořena třída SimpleLayout, která neovlivňuje výpočty na grafické kartě a pouze zajišťuje správné vykreslení popisku. Tato třída je součástí balíčku tiger.effects.labeling.layout a je potomkem abstraktní třídy LabelLayout. Překrývá její metody zajišťující vykreslení vodících čar a popisků. Na obrázku 3.4 je zachycen balíček tiger.effects.labeling.layout. Metody ostatních tříd v něm nejsou zachyceny, protože nejsou pro tento popis podstatné.




*Obrázek 3.4:Třídy balíčku tiger.effects.labeling.layout*

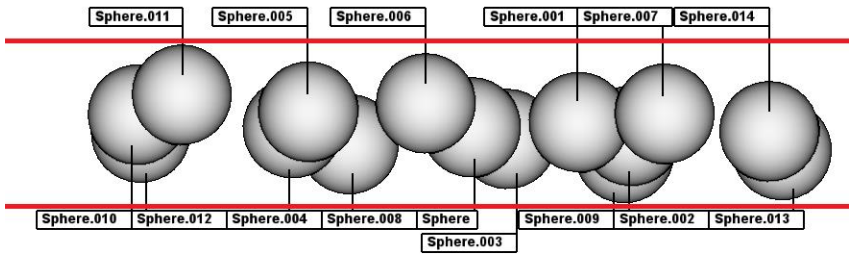
## Kapitola 4

# Panorama – analytická část

V této kapitole je analyzován algoritmus panoramatického umístění popisků, který ve své práci prezentovali Gemsa a kol. [19]. Algoritmus řeší problém umístění popisků na hranici obrázku. Na rozdíl od prací, které mu předcházely, však uvažuje takové množství popisků, které není umístitelné pouze do jedné řady. Tato úloha tedy představuje optimalizační problém, přičemž předmětem optimalizace je minimalizace počtu řádků, do kterých jsou popisky umístěny, či maximalizace počtu popisků umístěných do n řádků. Na obrázku 4.1 je zachycen výstup algoritmu pro případ maximalizace počtu popisků umístěných do tří řádků.

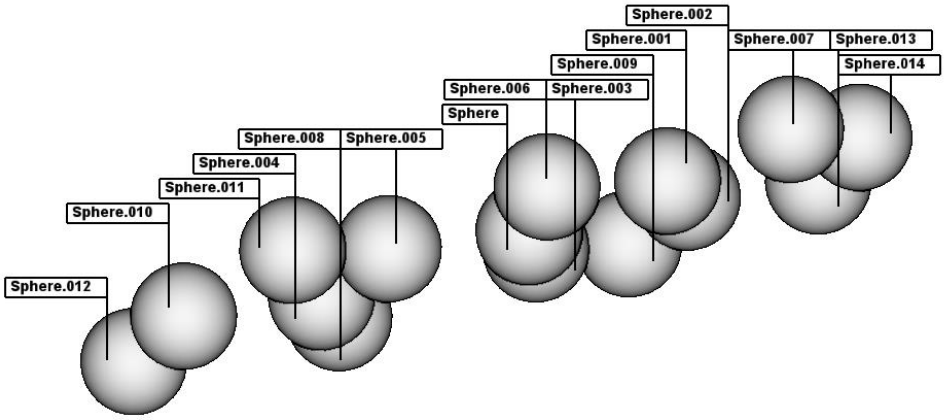

*Obrázek 4.1: Maximalizace počtu popisků pro 3 řádky. Zdroj [19].*

Pro účely této práce je možné algoritmus využít k optimalizaci umístění popisků v případě, že jsou umístěny nad či pod popisovaným objektem. Jeho využitím lze zvýšit kompaktnost rozložení a počet umístěných popisků ve zmíněných případech. Algoritmus lze použít v nezměněné formě, pokud budou popisky umístěny mimo popisovaný objekt, jak je zachyceno na obrázku 4.2. V tomto případě jsou uvažovány horizonty (červené čáry na obrázku 4.2) jako v algoritmu [19].




*Obrázek 4.2: Popisky nad a pod při využití horizontů*

Vzhledem k charakteru zobrazovaných dat je vhodné algoritmus upravit takovým způsobem, aby umísťoval popisky do scény s respektem ke hranici objektu, jak je zachyceno na obrázku 4.3, místo umístění všech popisků nad objekt.




*Obrázek 4.3: Rozložení popisků nad objektem vzhledem k jeho hranici.*

## 4.1 Algoritmus panoramatického rozmístění popisků

V této sekci je popsána původní verze algoritmu panoramatického umístění popisků pro minimalizaci počtu řádků, který prezentovali Gemsa a kol. [19]. Jeho cílem je rozmístění popisků do co nejmenšího počtu řad takovým způsobem, aby se žádné dva popisky neprotínaly. Dále se s popisky nesmí protínat žádná z, vertikálních, vodících čar.

Algoritmus uvažuje množinu bodů (kotev popisků) $P = \{p_1, \dots, p_n\}$, kde $p_i = (x_i, y_i)$. Též uvažuje, že je tato množina vzestupně uspořádána dle souřadnice x, tedy $x_i < x_{i+1}\ \forall i \in \{1, \dots, n\}$. Algoritmus dále uvažuje, že $y_i < 0\ \forall i \in \{1, \dots, n\}$, přičemž hodnota y = 0 vyznačuje horizontální čáru zvanou horizont. Je tedy patrné, že se žádné dvě vodící čáry nemohou protínat, jelikož jsou vertikální a mají rozdílné souřadnice x. Je nutné nalézt množinu $L = \{L_1, \dots, L_n\}$, kde $L_i = (X_i, Y_i)$, přičemž $y_i > 0\ \forall i \in \{1, \dots, n\}$. $L_i$ představuje pravý dolní roh popisku a díky kladné souřadnici y leží nad horizontem. Dále algoritmus uvažuje dva virtuální popisky $p_0 = (x_0, y_0)$ a $p_{n+1} = (x_{n+1}, y_{n+1})$ $x_0 < x_1, x_{n+1} < x_n$, které leží mimo scénu a neovlivňují rozložení ostatních popisků. Jejich přítomnost je však nutná pro správný chod algoritmu.

### 4.1.1 Optimalizace jedné řádky

První fází algoritmu je výpočet rozložení popisků pro případ, pokud by byla k dispozici pouze jedna řada, do které mohou být umístěny. Tento problém je řešen hladovou optimalizací. Každý popisek je umístěn na nejlevější možnou pozici. Tedy $X_i = max\{X_{i-1} + W_i, x_i\}$, kde $W_i$ představuje šířku i-tého popisku. Nastane-li situace, kdy $X_i > x_i + W_i$, pak není možné popisek bez překryvu umístit. Řešení, kdy by všechny popisky byly umístěny v jedné řadě, v tomto případě neexistuje. Dále jsou spočteny hodnoty $X_{i,j,1}, i \leq j$, které představují polohu okraje j-tého popisku při umístění popisků

v rozmezí **i** až **j** stejným způsobem, jak je popsáno výše. V případě, že řešení tohoto problému neexistuje, je $X_{i,j,1} = \infty$.

### 4.1.2 Panoramatické rozmístění popisků

Algoritmus konstruuje třírozměrnou tabulku $T$, kde $T[i,j,1] = X_{i,j,1}$. V první fázi algoritmu je tedy proveden výpočet pro jednu řadu, jak je popsáno v sekci 4.1.1. Tato první fáze je nutná, protože v dalším běhu algoritmus využívá dříve vypočítaných hodnot uložených v tabulce. První řada tedy plní tabulku iniciálními daty, která jsou použita pro další výpočet. Následně algoritmus konstruuje tabulku $T$ kde $T[i,j,k] = \min \theta_{i,j}^{k}, k > 1$ a kde

$$\theta_{i,j}^{k} = \{\max\{x_j, T[i,l,k] + W_j\} \mid i \le l < j, T[i,l,k] < x_j, T[l,j,k-1] < \infty\}. \quad (4.1)$$

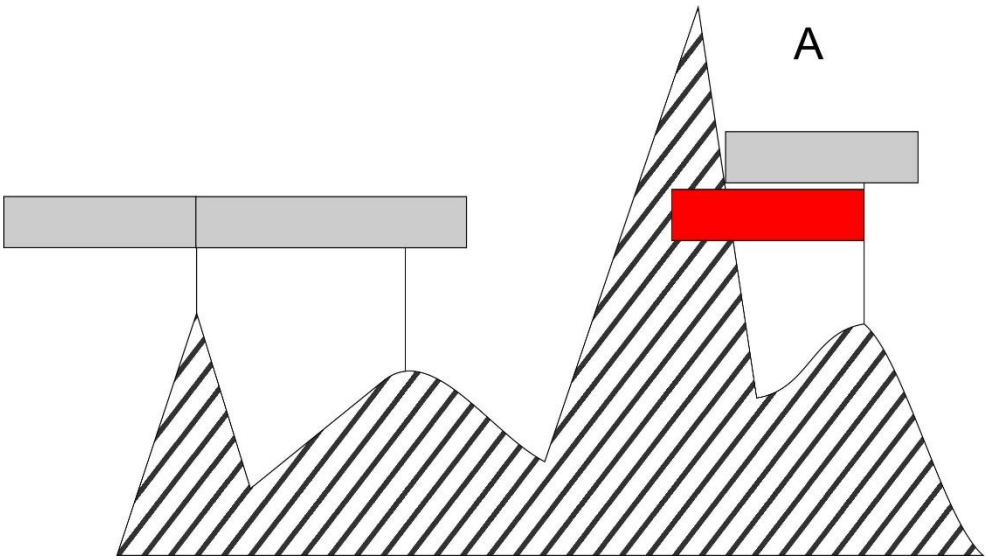
Algoritmus tedy pro každé dva popisky **i** a **j** na řádce **k** hledá třetí popisek **l**, který se nachází mezi nimi, a existuje řešení umístění popisků **l** až **j** na řádce **k-1**. Množina $\theta_{i,j}^{k}$ představuje polohy pravých okrajů popisku **j** pro všechny přípustné dělící popisky **l**. Do tabulky je pak zapsáno minimum z této množiny, které představuje nejkompaktnější rozmístění popisků. Zde je také patrný význam virtuálních popisků, které umožňují umístění popisku **1** do nižší úrovně a popisku **n** do vyšší úrovně v příslušných množinách $\theta_{i,j}^{k}$. Tabulka T je pak naplněna algoritmem 5. Po jejím naplnění následuje fáze zpětného vyhledávání (angl. backtracking), při které je rekonstruováno rozložení popisků. Aby existovalo přípustné rozložení všech popisků, musí pro řádku **k** nastat situace, kdy $T[0, n+1, k] < \infty$. V takovém případě bylo nalezeno přípustné rozložení popisků a algoritmus končí.

Algoritmus 5: Výpočet tabulky T

```
pro  k od 2 do počet popisků/2{
     pro j od 1 do počet popisků+1{
          pro i od 0 do j-1{
               T[i,j,k]= min θ^k_{i,j}
          }
     T[j,j,k]=x_j
     }
pokud T[0,n+1,k]< ∞ pak vrať k
}
```

## 4.2 Detekce volného místa

Algoritmus panoramatické rozmístění popisků může být rozšířen o detekci volného místa. Verze, kterou publikovali Gemsa a kol. [19], uvažuje umístění popisků nad horizont. Jediné další objekty, se kterými by mohl popisek mít neprázdný průnik, jsou pak další popisky. V případě odstranění požadavku na horizont, nad kterým je veškeré místo volné, je nutné algoritmus obohatit o možnost detekce volného místa. Je nutné poukázat, že v případě, kdy nebudou přítomny další objekty, které by možnosti umístění popisků redukovaly, bude činnost algoritmu totožná s jeho původní verzí [19].




*Obrázek 4.4: Detekce volného místa*

Na obrázku 4.4 je zachycen případ, kdy by popisky (šedé obdélníky) nebyly umístěny nad horizont, který se v případě obrázku 4.4 nalézá na nejvyšším vrcholem u písmena „A“. Původní verze algoritmu [19] by v případě umístění horizontu pod úroveň vrcholu nebyla schopna rozpoznat kolizi s popisovaným objektem. Tento případ zachycuje červený obdélník, jehož pozice očekávaná algoritmem je nesprávná. Přípustnou pozicí v modifikované verzi algoritmu je poloha šedého obdélníku u písmene „A“.

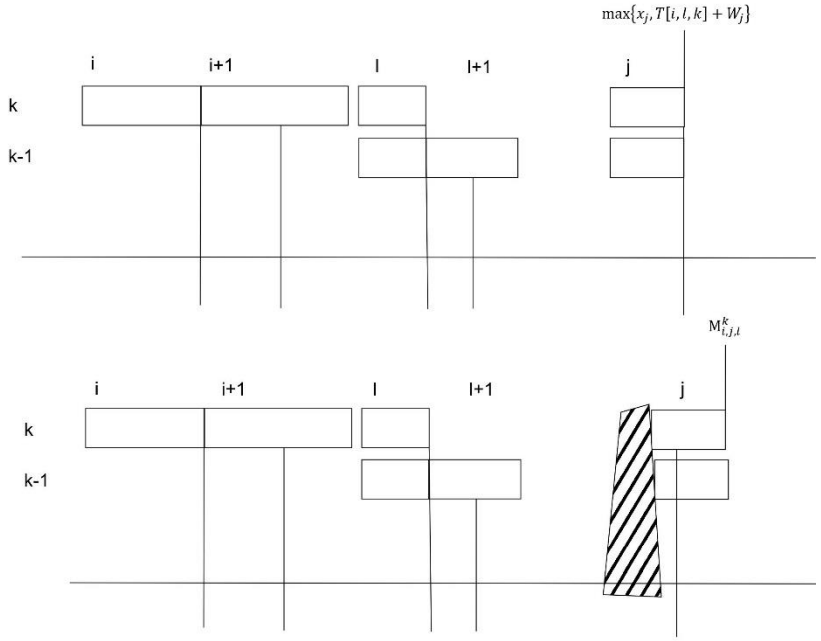
Změnou množiny $\theta_{i,j}^{k}$ lze docílit správného chování algoritmu pro situaci popsanou výše. Hodnota $\max\{x_j, T[i,l,k] + W_j\}$ reprezentuje polohu pravého okraje popisku **j** při umístění popisků s indexy v rozmezí **i** až **l**. Tato hodnota je považována za výchozí pro modifikovanou verzi algoritmu. Popisek s pravým okrajem je však podroben testu, kterým je zjištěn jeho případný nenulový průnik se scénou. Hodnota $\mathrm{M}_{i,j,l}^{k}$ pak představuje polohu pravého okraje tohoto popisku s respektem k objektům ve scéně:

$$\mathrm{M}_{i,j,l}^{k} = \min\{x \mid \boldsymbol{j} - t\acute{y}\ popisek\ v\ \check{r}ad\check{e}\ \boldsymbol{k}\ s\ pravým\ okrajem\ v\ bodu\ na\ \acute{u}rovni\ \boldsymbol{x}\ m\acute{a}\ nulový\ pr\mathring{u}nik\ se\ scénou,\ \max\{x_j, T[i,l,k] + W_j\} \le x \le x_j + W_j\} \tag{4.2}$$

Množina $\theta_{i,j}^{k}$ pak přechází do tvaru:

$$\theta_{i,j}^{k}{}' = \{\mathrm{M}_{i,j,l}^{k} \mid i \leq l < j, T[i,l,k] < x_j, T[l,j,k-1] < \infty\} \tag{4.3}$$

Chod algoritmu s množinou $\theta_{i,j}^{k}{}'$ je již stejný jako v případě algoritmu 5. Je zde také patrné omezení kladené na tvar volného místa ve scéně. Řešení pro rozmístění popisků **i** až **j** je hledáno jako rozklad na dvě částí s dělícím popiskem **l**, kde první část je na úrovní **k** a druhá na úrovni **k-1**. Existence řešení je tedy zjišťována pouze o úroveň níže. Obsazené místo ve scéně na úrovni **k** musí být podmnožinou obsazeného místa na úrovni **k-1**. V takovém případě $T[i,l,k-1] < \infty \Rightarrow T[i,l,k] < \infty$. Toto je obecně splněno pro konvexní oblasti. V případě libovolného tvaru obsazeného místa by muselo být řešení vyhledáváno na úrovních $\widehat{k-1}$. Dalším problematickým aspektem je, že při nevhodném nastavení úrovně horizontu nebude možno umístit žádné popisky. V takovém případě však $T[i,j,0] = \infty \; \forall i \forall j, j \leq i$ a řešení diverguje jelikož $\theta_{i,j}^{k}{}', k > 0$ je vždy prázdná množina. Ke stejnému problému dojde v případě existence obsazené úrovně.




*Obrázek 4.5: Vyhodnocení obsazeného místa*

Obrázek 4.5 zachycuje vztah mezi $\max\{x_j, T[i,l,k] + W_j\}$ v původní verzi algoritmu a $\mathrm{M}_{i,j,l}^{k}$ v upravené verzi. Na obrázku 4.5 nahoře je zachycen případ, kdy jsou popisky umístěny do volného prostoru. Na obrázku 4.5 dole jsou popisky umístěny do scény s překážkou. Výchozí pozice popisku **j** je hodnota $\max\{x_j, T[i,l,k] + W_j\}$, následně je popisek algoritmem posunut na pozici $\mathrm{M}_{i,j,l}^{k}$, kde nedochází k jeho průniku s překážkou.

## Kapitola 5

# Panorama – návrh a implementace

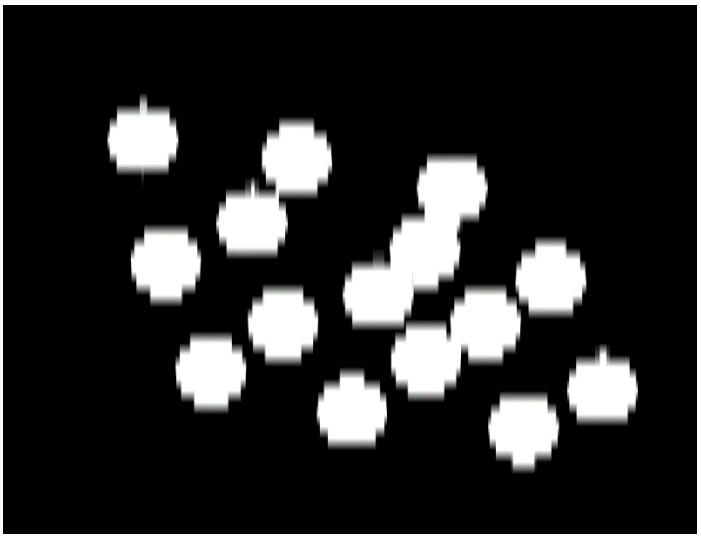
Tato kapitola prezentuje implementační část práce spojené s panoramatickým umístěním popisků. Toho je využito pro umísťování popisků do scény v případě, že se nacházejí nad a/nebo pod siluetou objektu. Je zde popsána datová struktura, která je využita k výpočtu hodnoty $\mathrm{M}_{i,j,l}^{k}$. V případě, že jsou popisky umístěny pod siluetou objektu, je pro správný výstup algoritmu nutná transformace vstupních dat – zrcadlení kotev a scény podél horizontální osy. Tento přístup je popsán v sekci 5.3.1. Samotný algoritmus nebyl upravován.

Při panoramatickém umístění popisků do scény jsou nejprve spočteny vodící čáry pomocí algoritmu umísťování popisků tak, jak je popsáno v kapitole 2. Pozice kotev popisků, společně s texty popisků a informací o objektech ve scéně jsou dále předány instanci třídy PanoramaOneWayLayout pro umístění popisků nad nebo pod objektem, případně instanci třídy PanoramaTopBottomLayout při jejich umístění nad i pod objektem. Ta tyto data dále vhodně předá instanci třídy MultirowLayout, která zaštiťuje panoramatické rozmístění popisků.

Úprava algoritmu, tak jak byla popsána v sekci 4.2, není ještě zcela funkční. Je zde uvedena její implementace společně se základní variantou, kdy je uvažován ve scéně horizont, nad který jsou popisky umístěny.

## 5.1 Získání obsazení scény

Ještě před tím, než mohou být vypočteny pozice popisků, je nutné získat informaci o objektech ve scéně. Jelikož algoritmus panoramatického umístění popisků operuje s řádky, není nutné a není ani žádoucí přenášet z grafické karty na procesor data v plném rozlišení okna scény. Ve třídě ExternalLabeling balíčku tiger.effects labeling je tedy přidána nová textura, která má šířku scény, ale její výška odpovídá počtu řádků, který se do scény vejde. Výška řádku je získána pomocí instance třídy TextRenderer. Poté, co jsou textuře a příslušnému framebuferu nastaveny správné rozměry, je do nich vykreslena scéna. Výsledek je zachycen na obrázku 5.1



*Obrázek 5.1: Textura volného místa pro množinu koulí*

Textura je přenesena na procesor a předána objektu, který ze získaných dat vytvoří datovou strukturu reprezentující obsazení místa ve scéně. Tato datová struktura je popsána v následující sekci 5.2.

**Implementace:**

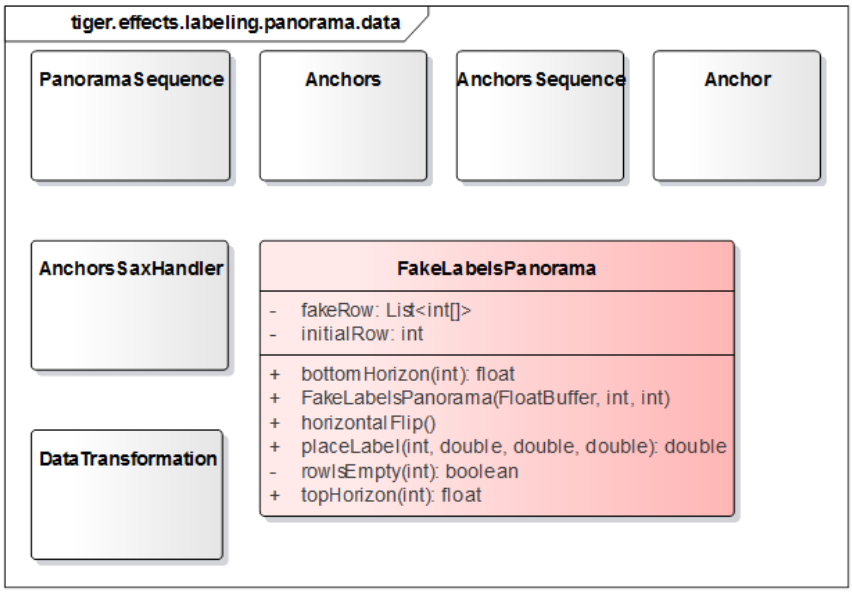
Tato funkčnost je implementována shadery PanoramaLines.vert a PanoramaLines.frag.

## 5.2 Reprezentace scény pro panoramatické rozmístění popisků

Scéna se pro účely panoramatického umístění popisků skládá z řádků. Každý řádek má šířku shodnou se šířkou scény. Každý bod ve vstupní textuře může nabývat pouze dvou hodnot, které vyjadřují, jestli je místo na jeho souřadnicích obsazené, či nikoli. Pro reprezentaci scény lze tedy použít dvojrozměrné pole. Řádky tohoto pole reprezentují řádky scény a sloupce reprezentují jednotlivé pixely těchto řádků. V algoritmu panoramatického umístění popisků je popisek umístěn vždy nalevo od nějakého bodu. Se znalostí volného místa nalevo od tohoto bodu je tedy možné rozhodnout, jestli je do něj možné popisek umístit, či ne. Scénu je možné reprezentovat jako dvojrozměrné pole, kdy každý prvek pole drží hodnotu volného místa nalevo od něj. Při umístění popisků pod objekt je nutné scénu zrcadlit podél horizontální osy. Z tohoto důvodu je vhodné použít ArrayList, kdy jednorozměrné pole – řádek, představuje záznam v něm.

Tato struktura je reprezentována třídou FakeLabelsPanorama balíčku tiger.effects.labeling.panorama.data. Třída obsahuje konstruktor FakeLabelsPanorama (FloatBuffer data, int width, int height), kde FloatBuffer data je načtená textura obsazení scény ze sekce 5.1 a int hodnoty widh a height představují její rozměry. Stěžejní metodou, která určuje, jestli je popisek umístitelný je metoda Double placeLabel (int rowNumber1, double labelWidth, float labelXAnchor, double presumedLabelXPos), kde rowNumber1 je číslo řádku, labelWidth je šířka popisku, labelXAnchor je pozice kotvy popisku na ose x a presumedLabelXPos je předpokládaná pozice popisku. Předpokládaná pozice je hodnota $\max\{x_j, T[i,l,k] + W_j\}$. Metoda placeLabel tedy slouží k určení

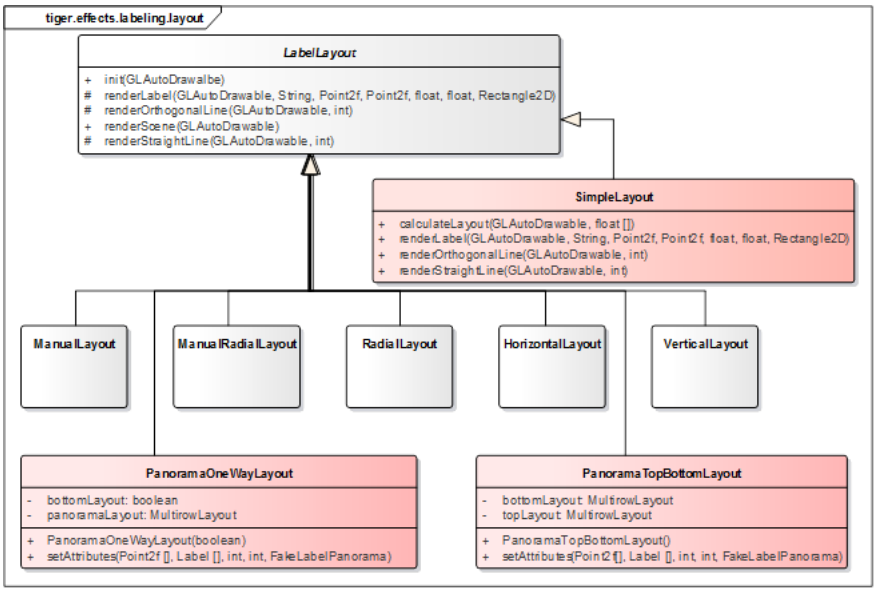
hodnoty $M_{i,j,l}^{k}$ ze vztahu (4.2) a naplnění množiny $\theta_{i,j}^{k}{}'$ ze vztahu (4.3). Metody a atributy této třídy jsou zachyceny na obrázku 5.2.




*Obrázek 5.2: Třídy balíčku tiger.effects.labeling.panorama.data*

## 5.3 Rozmístění popisků

Polohy kotev popisků jsou získány pomocí algoritmu automatického umístění popisků, tak jak je popsáno v kapitole 2. Množina hlavních směrů algoritmu je odvislá od požadavku na polohu popisků. V případě panoramatického rozmístění popisků je cílem umístit co nejvíce popisků. Prahová hodnota překryvu je tedy nastavena tak, aby algoritmus do scény umístil všechny popisky bez ohledu na případné překryvy s dalšími objekty. Polohy popisků budou upraveny algoritmem panoramatického umístění popisků. V tomto novém rozložení se již překryvy vyskytovat nebudou.



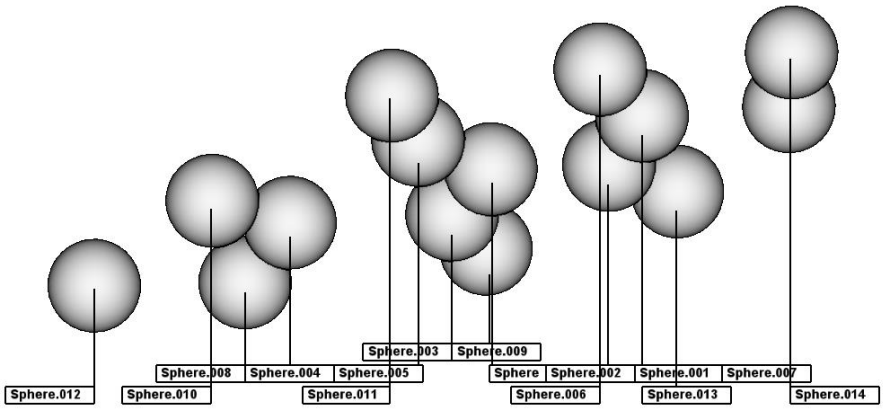

*Obrázek 5.3: Třídy balíčku tiger.effects.labeling.layout*

Instance třídy MultirowLayout jsou obaleny novým objektem – instancí třídy PanoramaOneWayLayout nebo PanoramaTopBottomLayout. Naplní je potřebnými daty – polohami kotev, textem popisků a informací o místě ve scéně a zprostředkovává volání metod třídy LabelLayout. Obalový objekt se tedy chová jako instance třídy LabelLayout a může být použit ve třídě ExtrenalLabeling pro vykreslení rozložení popisků do scény. Struktura balíčku tiger.effects.labeling.layout je zachycena na obrázku 5.3.

Pokud jsou popisky rozmístěny původní verzi algoritmu [19], je využito instance třídy FakeLabelsPanorama k určení horizontů scény, jak je zachyceno na obrázku 4.2. Popisky jsou pak rozmístěny nad a/nebo pod tyto horizonty.

### 5.3.1 Umístění popisků pod popisovaným objektem

Algoritmus panoramatického umísťování popisků není schopen umístit popisky pod popisovaný objekt. Popisky jsou jím rozmísťovány do řad, jejichž souřadnice **y** ve scéně roste s číslem řady. Popisky by tedy byly umístěny do řad v opačném pořadí a protínaly by se s vodícími čarami jiných popisků, jak je zachyceno na obrázku 5.4.




*Obrázek 5.4: Protínání popisků a vodících čar*

Pro správnou funkci algoritmu je tedy nutné zrcadlit popisovanou scénu podél horizontální osy, provést výpočet rozložení popisků a výslednou scénu opět zrcadlit podél její horizontální osy. Tato funkčnost je implementována v třídě MultiRowLayout.java balíčku tiger.effects.panorama.layout.

### 5.3.2 Umístění popisků nad a pod popisovaným objektem

Případ, kdy mají být popisky umístěny nad i pod popisovaný objekt, je řešen třídou PanoramaTopBottomLayout zachycenou na obrázku 5.3. Popisky jsou rozděleny do dvou skupin. Jedna skupina je vykreslena nad objekt a druhá skupina pod objekt. Každé toto vykreslení je zajištěno vlastní instancí třídy MultiRowLayout.

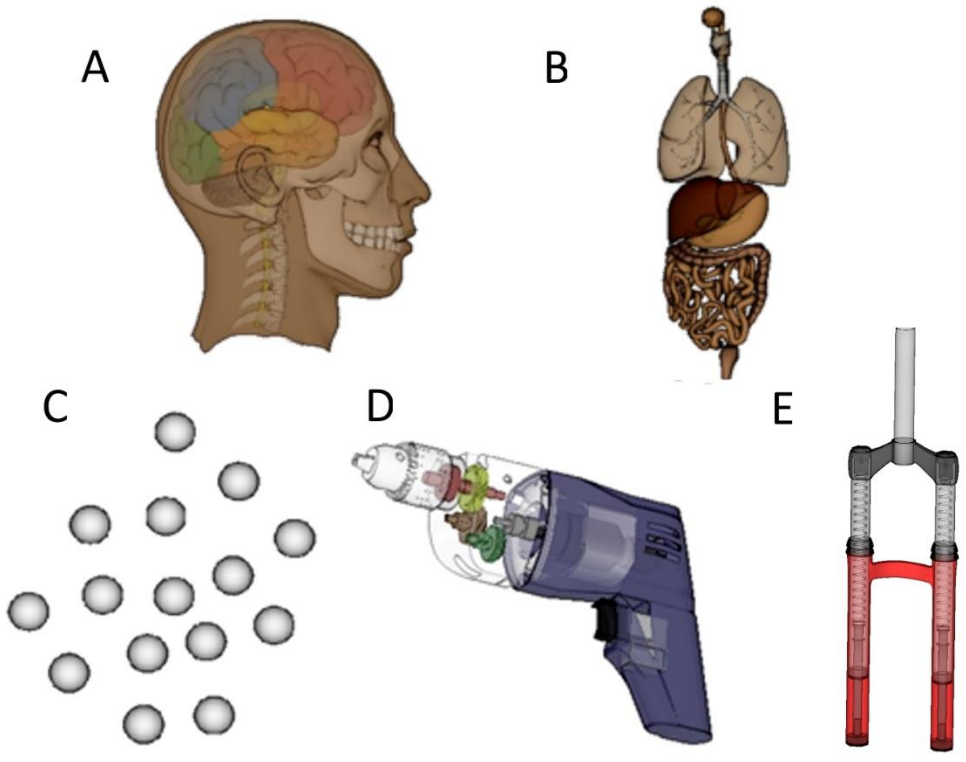
## 5.4 Rozmístění popisků s horizontem

Výpočet rozložení popisků pomocí algoritmu panoramatického umístění popisků lze realizovat také v jeho triviální podobě. V případě umístění popisků nad model je ve scéně nalezen horizont, nad kterým je všechno místo volné a rozmístění popisků nic nelimituje. K nalezení horizontu je využita instance třídy FakeLabelsPanorama. Její metoda topHorizon() vrací polohu horizontu. Rozmístění popisků je následně vypočteno pomocí původní verze algoritmu panoramatického umístění popisků [19]. V případě umístění popisků pod nebo nad a pod model je použit stejný přístup jako v sekci 5.3.1, resp. 5.3.2.

## Kapitola 6

# Testování

V této kapitole jsou otestovány výsledky implementace na modelech dodaných vedoucím práce. Výsledky jsou zobrazeny párově s výstupem původní verze algoritmu, aby bylo možné jejich srovnání a byla možná diskuze nad výhodami a nevýhodami nově implementované verze algoritmu. Pro účely demonstrace výhod implementace modifikované verze algoritmu jsou zahrnuty také modely, který nereprezentují fyzické objekty. Na všech obrázcích sekce 6.1, na kterých jsou zobrazeny modely, je taktéž zobrazena jejich hranice, respektive hranice konvexní obálky pro původní verzi algoritmu. Tato hranice je na obrázcích zachycena úzkou červenou čárou. Předmětem testování byly modely zobrazené na obrázku 6.1.

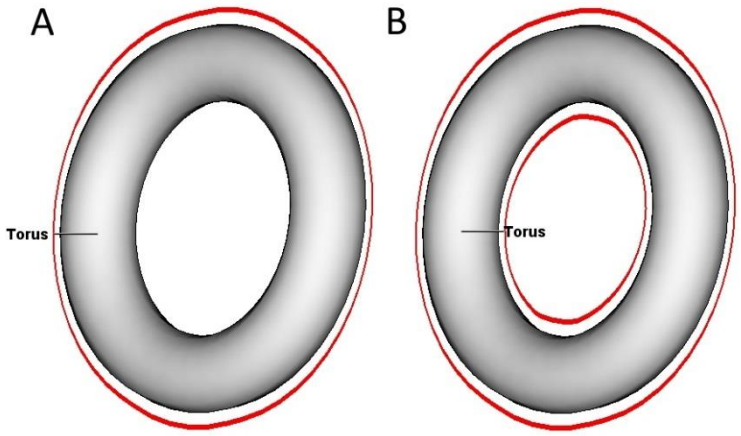



*Obrázek 6.1: Testované modely. (a) hlava, (b) trávící soustava, (c) sada koulí, (d) vrtačka, (e) vidlice kola*

## 6.1 Testování rozložení popisků kolem nekonvexní obálky

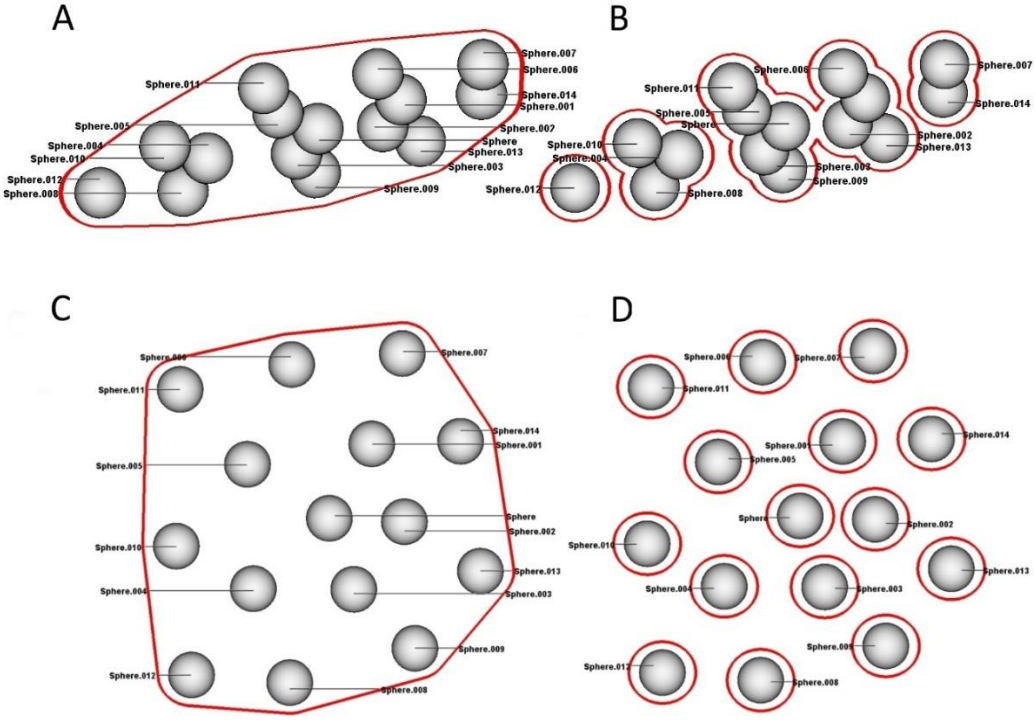
Nejprve jsou otestovány výstupy algoritmu pro modely zachycené na obrázku 6.1 za použití algoritmu pro umístění popisků na hranici objektu.

Na úvod je uveden příklad umístění popisku do torusu. Jedná se o reprezentativní příklad, který ukazuje nemožnost umístění popisku uvnitř torusu v původní verzi algoritmu. Tento nedostatek je novou verzí odstraněn.

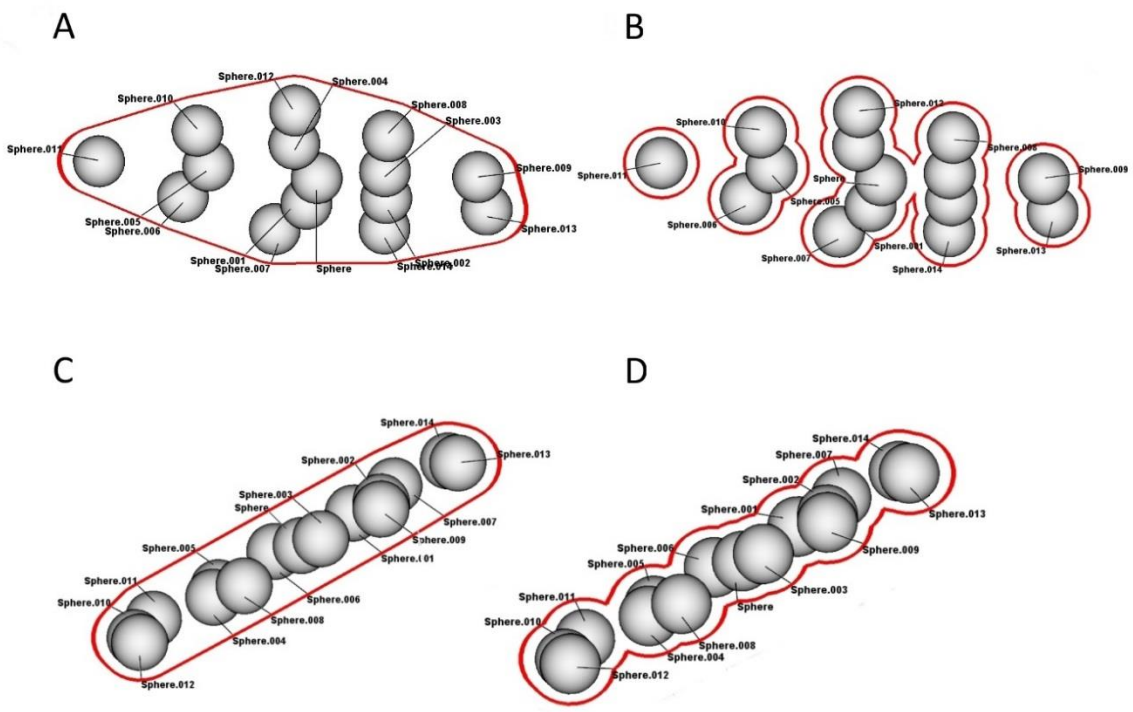



*Obrázek 6.2: Torus. (A)- původní algoritmus, (B) nový algoritmus*

Další příklad je opět modelový, přičemž jsou znovu demonstrovány hlavní výhody umísťování popisku přímo na hranici objektu. Jak zachycují obrázky 6.3 a 6.4, konvexní obálka množiny koulí obklopuje také velké množství volného prostoru, kam není problém umístit příslušné popisky.

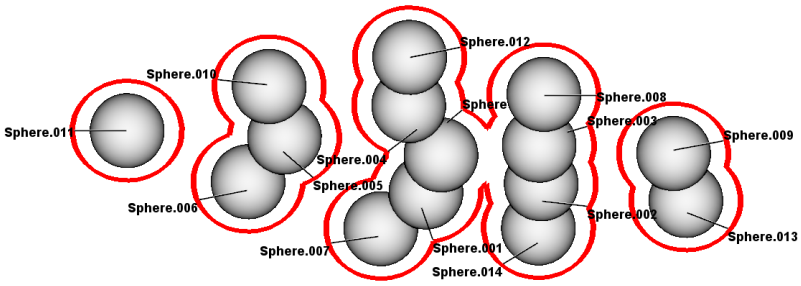



*Obrázek 6.3: Koule. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus*



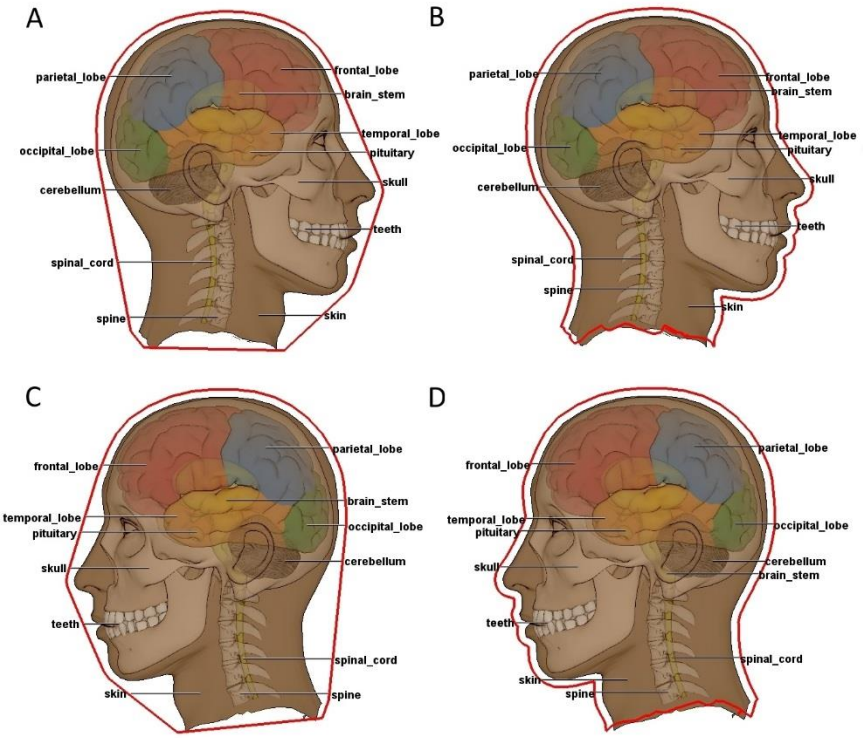

*Obrázek 6.4: Koule. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus*

Z obrázků 6.3 a 6.4 je patrné, že uspořádání popisků u nového algoritmu je více kompaktní, což je jeden z hlavních požadavků na něj. Dále je však zřejmé, že ne všechny popisky byly umístěny, jak je případem na obrázku 6.3(b) a 6.4(b). Stalo se tak proto, že na hranici objektu, kam se algoritmus popisek pokouší umístit, není pro něj dostatek volného místa. Původní verze algoritmu tímto problémem netrpí a všechny popisky byly umístěny. Tento problém lze u nové verze také částečně řešit. Uživatel může měnit prahovou hodnotu překryvu popisku s dalšími objekty ve scéně, při které je popisek ještě umístěn. Tento případ je zachycen na obrázku 6.5.

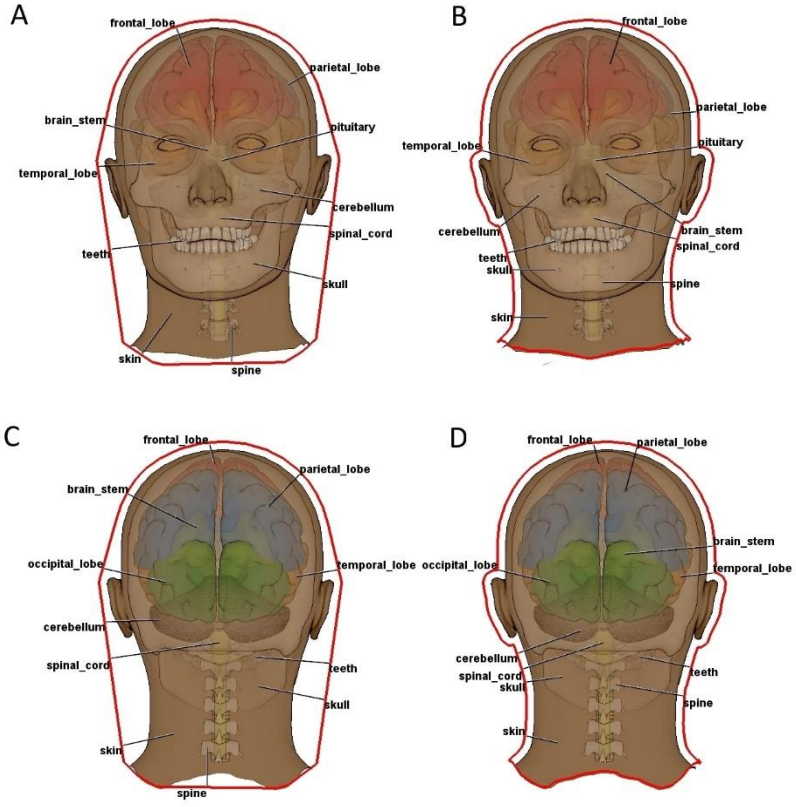


*Obrázek 6.5: Umístění popisku s uživatelem povoleným překryvem.*

Dalším testovaným modelem je model lidské hlavy. Výsledky rozložení popisků jsou zachyceny na obrázcích 6.6 a 6.7. Jak je z obrázků patrné, jedná se o model, který je téměř konvexní. Tento fakt způsobuje malé změny v rozložení popisků vůči původní variantě algoritmu.

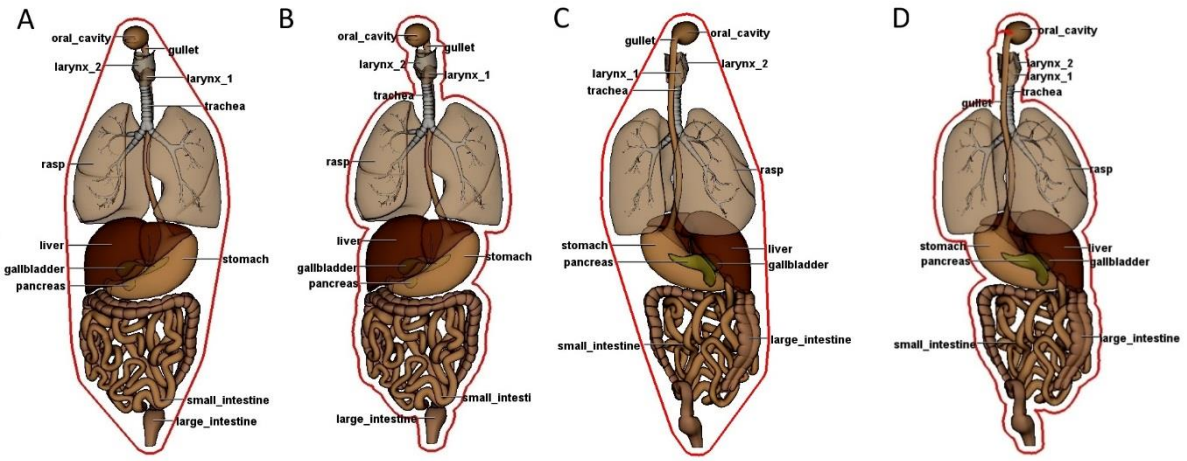



*Obrázek 6.6: Hlava.(a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus.*

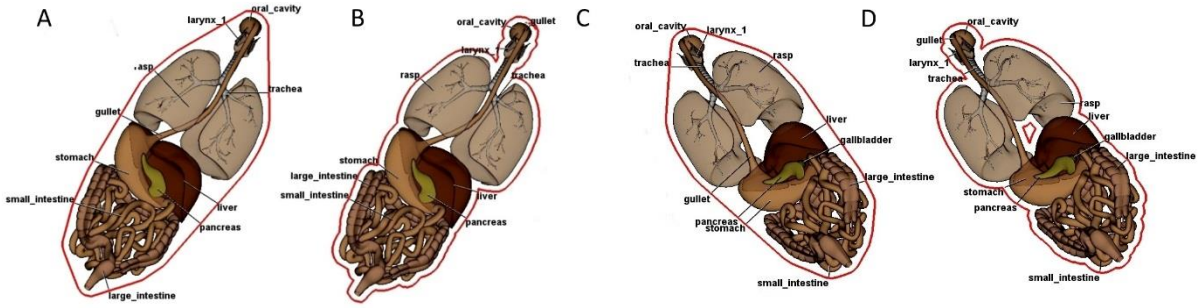



*Obrázek 6.7: Hlava. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus.*

Rozmístění popisků u modelu trávící soustavy, jak je zachyceno na obrázcích 6.8 a 6.9, vykazuje vyšší kompaktnost. Vzhledem k podobnosti konvexní a nekonvexní obálky jsou si však výsledná rozložení podobná.

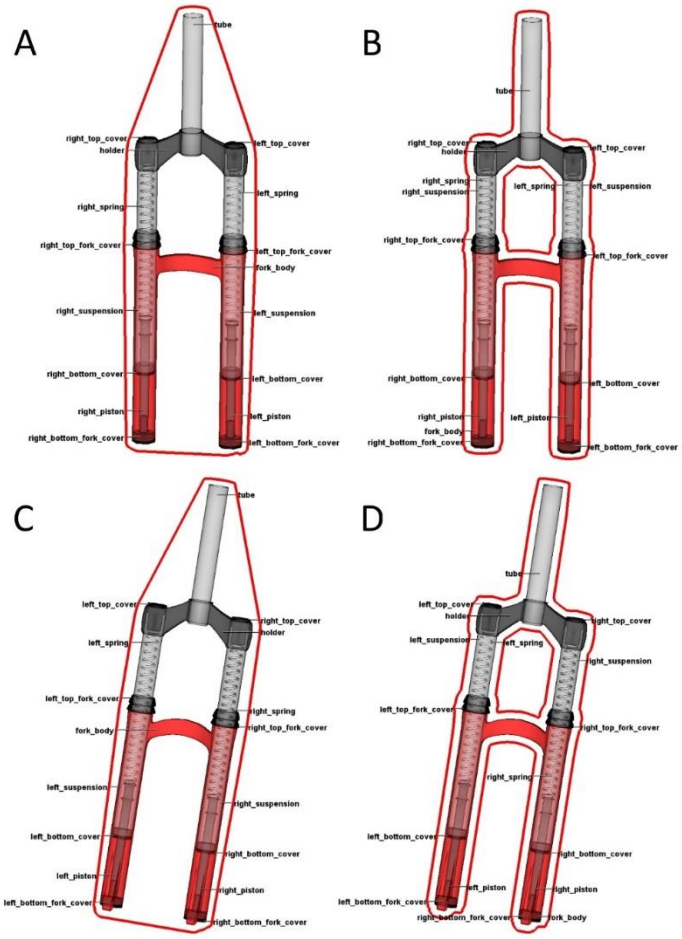



*Obrázek 6.8: Trávící soustava. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus.*

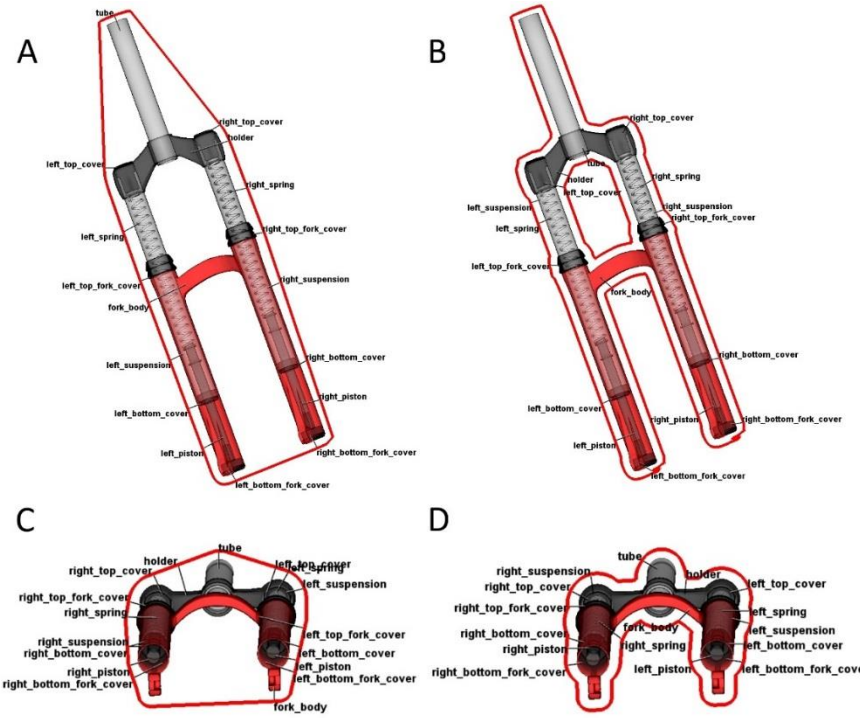



*Obrázek 6.9: Trávící soustava. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus.*

Model vidlice kola byl dalším testovaným. Rozmístění popisků na testovaných pohledech je zachyceno na obrázcích 6.10 a 6.11. Obě rozmístění popisků jsou si navzájem podobná. Modifikovaná verze algoritmu zde vykazuje vyšší kompaktnost a čitelnost. Popisky umísťuje také do míst, kam jsou původní verzí algoritmu neumístitelné.




*Obrázek 6.10: Vidlice. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus.*

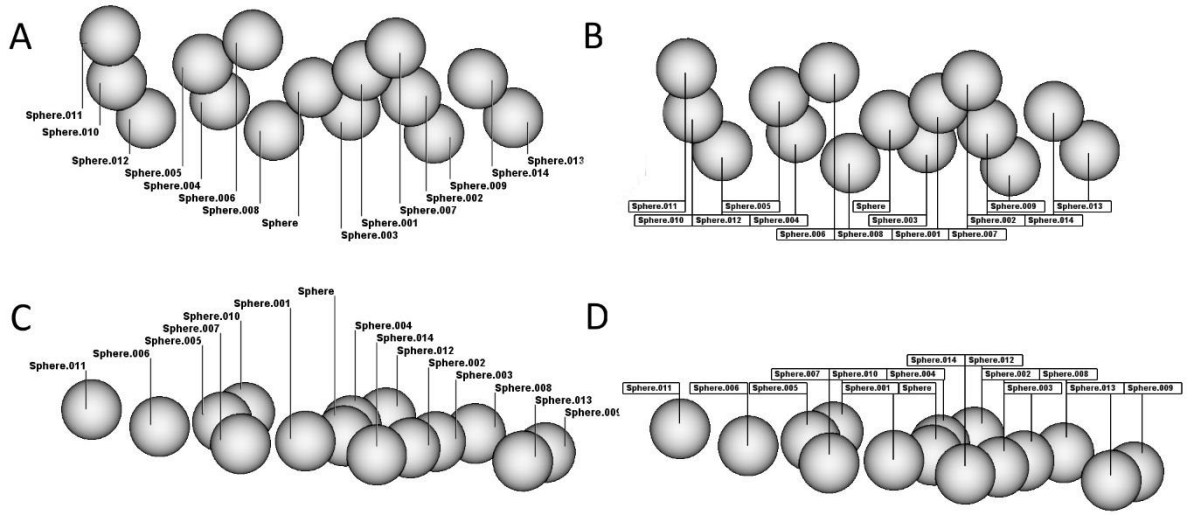



*Obrázek 6.11: Vidlice. (a) a (c) - původní algoritmus, (b) a (d) - nový algoritmus.*

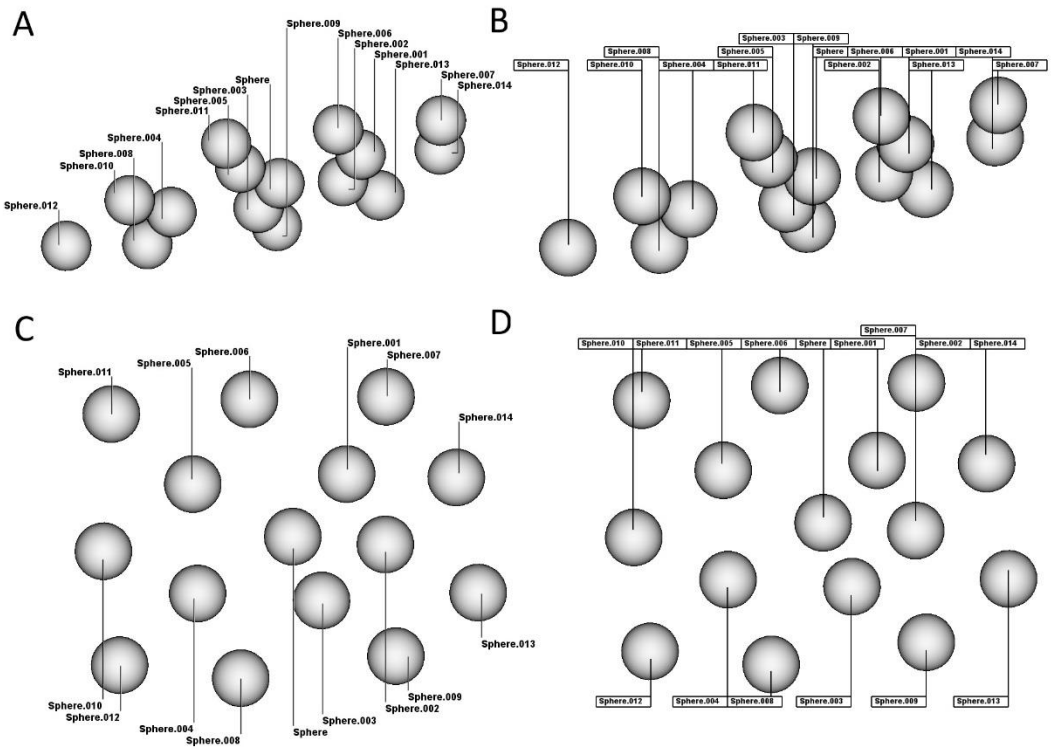
## 6.2 Testování panoramatického rozložení popisků

V této sekci je testováno rozložení popisků, pokud jsou umístěny nad a/nebo pod objektem. Vzhledem k tomu, že upravený algoritmus panoramatického rozmístění popisků nebyl v době testování plně funkční, je testování provedeno za využití dodané verze algoritmu [19]. Popisky jsou tedy umístěny pouze nad a/nebo pod modelem. Algoritmus neuvažuje volné místo ve scéně mezi a v okolí objektů a popisky umísťuje pouze do oblastí, kde se žádné objekty nenacházejí. Na obrázcích jsou zachyceny také výsledky rozložení popisků tak, jak bylo vyhodnocováno dříve.

Prvním testovaným modelem je sada koulí. Výsledky testování jsou zachyceny na obrázcích 6.12 a 6.13. Jedná se o případ, kdy rozložení popisků vychází lépe za použití panoramatického rozložení. Výsledek je vyšší kompaktnost tohoto rozmístění a výrazně menší počet řádků než při rozložení původním.

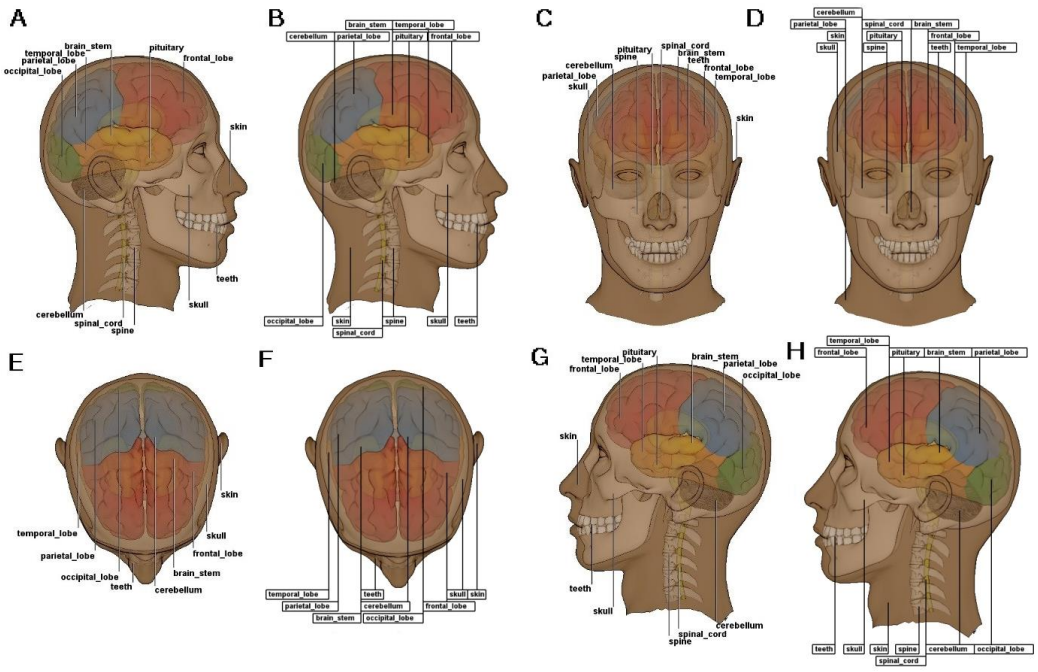



*Obrázek 6.12: Koule. (a) a (c) - původní algoritmus, (b) a (d) - panoramatické rozmístění novým algoritmem*

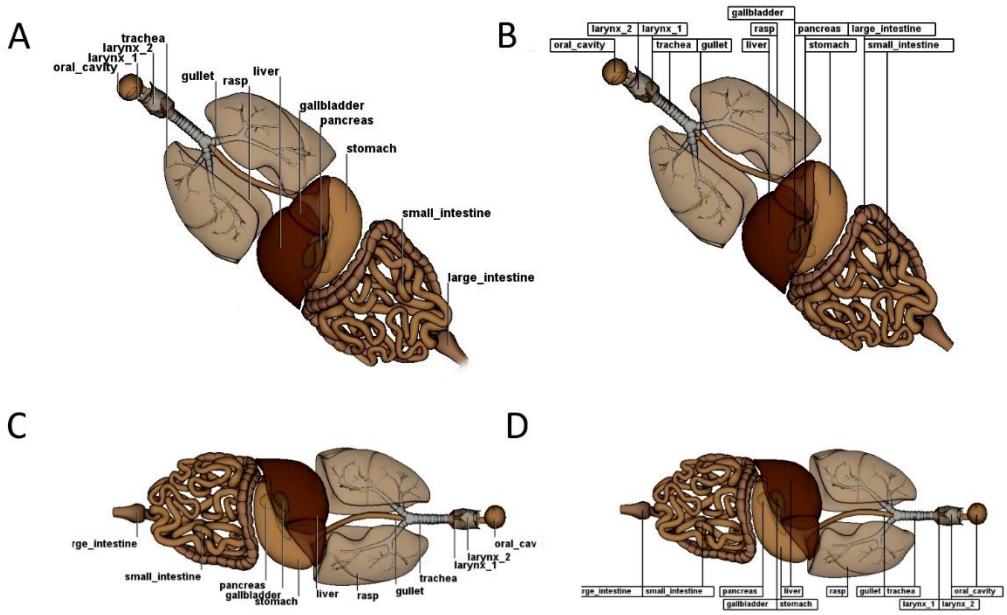



*Obrázek 6.13:: Koule. (a) a (c) - původní algoritmus, (b) a (d) - panoramatické rozmístění novým algoritmem*

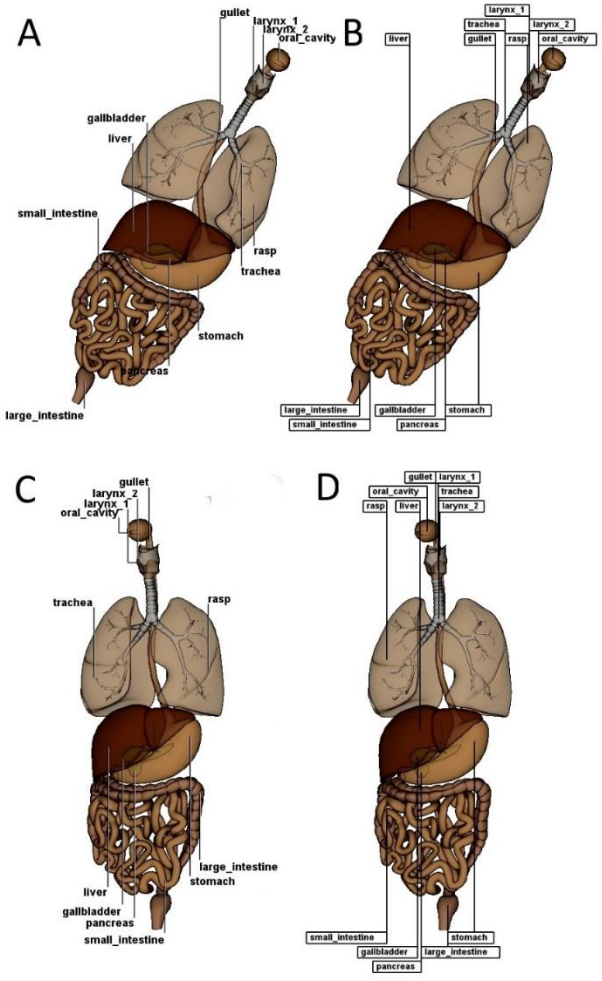
Dalším testovaným modelem byl model hlavy. Výsledky testování jsou zachyceny na obrázku 6.14. Z obrázků je patrné, že původní rozmístění popisků vykazuje vyšší kompaktnost a je lépe čitelné, než je tomu v případě panoramatického rozmístění. Obzvláště je tomu tak v případě, že jsou popisky rozmístěny nad nebo pod modelem. Panoramatické rozložení popisků zabírá menší počet řádků, než rozložení získané původním algoritmem.




*Obrázek 6.14: Hlava. (a), (c), (e), (g) - původní rozmístění, (b), (d), (f), (h) - panoramatické rozmístění*



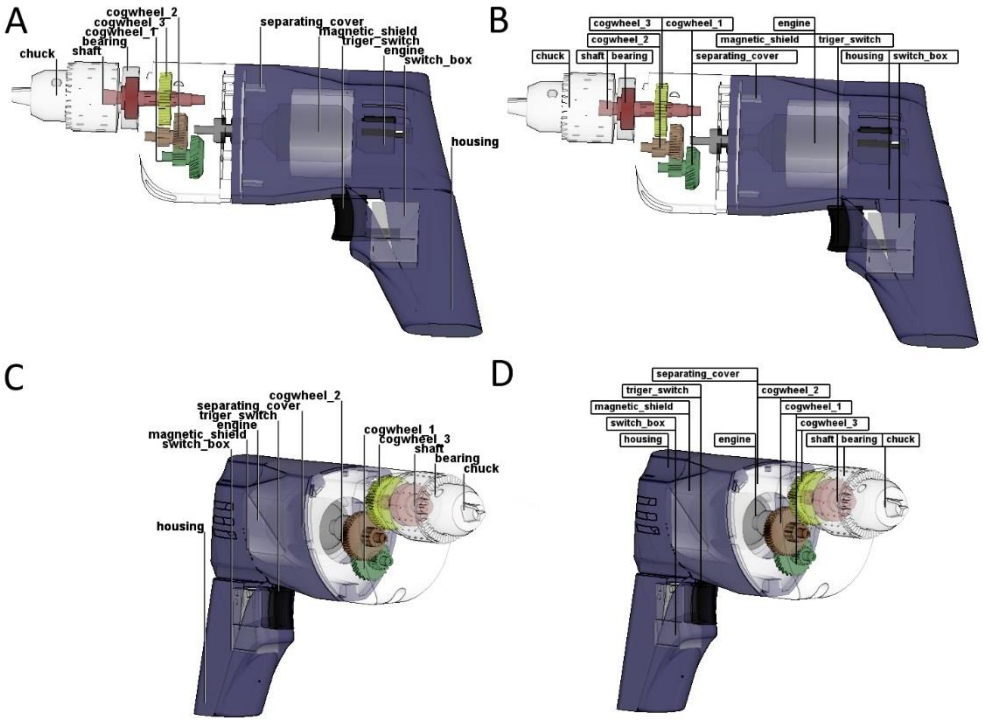

*Obrázek 6.15: Trávící soustava. (a), (c) - původní rozmístění, (b), (d) - panoramatické rozmístění*



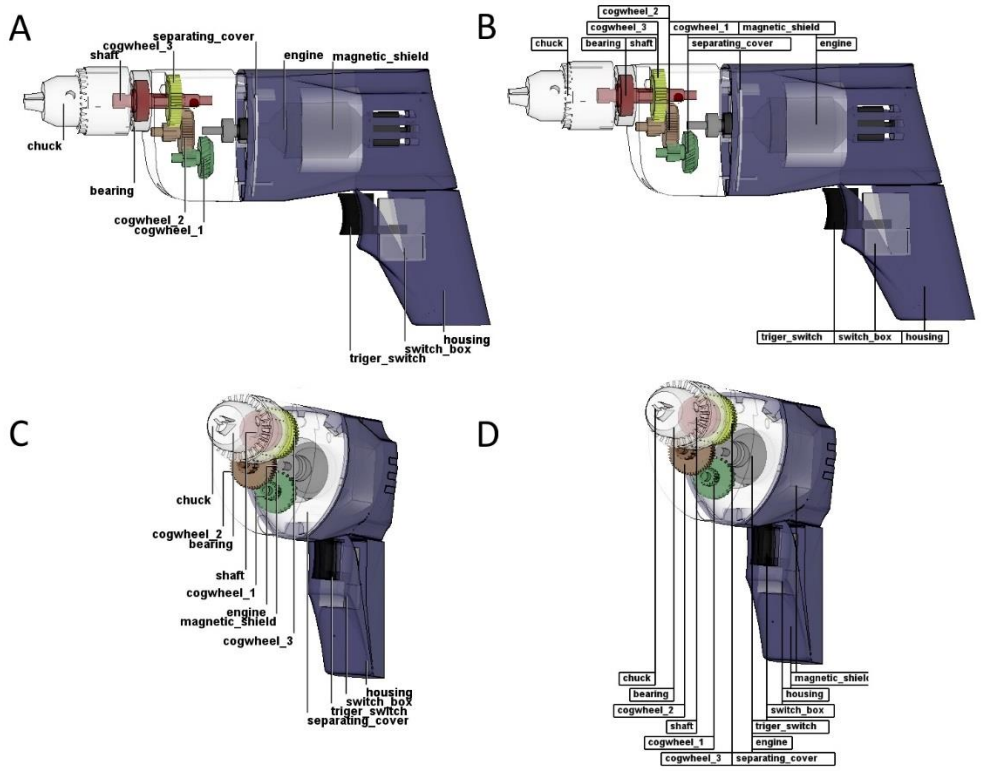

*Obrázek 6.16: Trávící soustava. (a), (c) - původní rozmístění, (b), (d) - panoramatické rozmístění*

Výsledky rozložení popisků pro trávící soustavu, jak zachycena na obrázcích 6.15 a 6.16 nevykazují velké rozdíly. Původní rozložení popisků je kompaktnější. To je zejména patrné ze srovnání obrázků 6.15(a) a 6.15(b).

Vrtačka byla poslední testovaným modelem. Výsledky rozložení popisků jsou zachyceny na obrázcích 6.17 a 6.18. Až na poslední testovaný pohled, zachycený na obrázku 6.18 (c) a (d), kdy původní rozmístění popisků vykazuje vyšší kompaktnost, se dosažené výsledky příliš neliší.

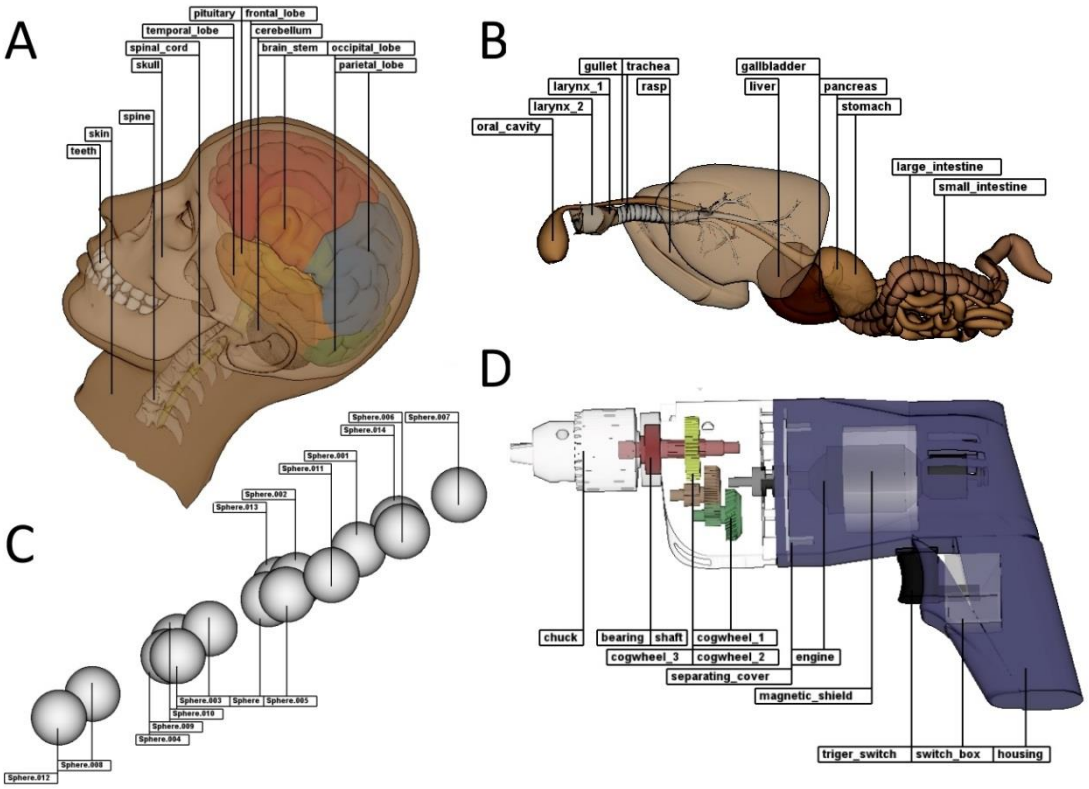



*Obrázek 6.17: Vrtačka. (a), (c) - původní rozmístění, (b), (d) - panoramatické rozmístění*




*Obrázek 6.18: (a), (c) - původní rozmístění, (b), (d) - panoramatické rozmístění*

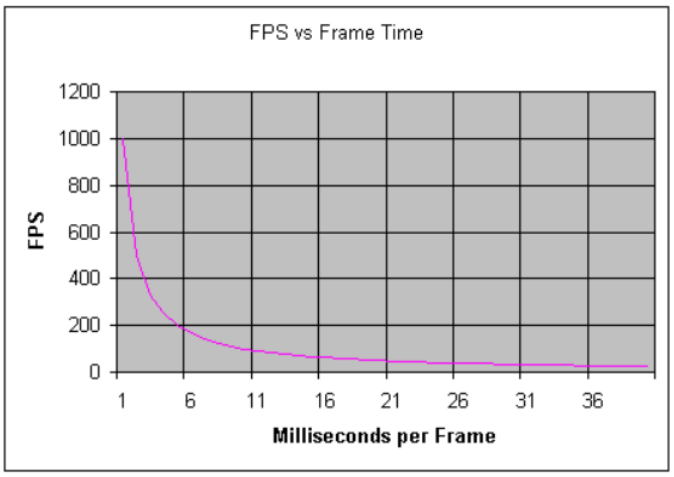
Na obrázku 6.19 jsou uvedeny výsledky rozmístění popisků získané s využitím modifikované verze algoritmu panoramatického rozmístění popisků. Na uvedených příkladech výsledné rozložení popisků kopíruje hranici daných modelů. Tento algoritmus však ještě nefunguje zcela správně, kvůli protichůdným požadavkům na něj kladených.




*Obrázek 6.19: Upravená verze panoramatického rozmístění popisků. (a) hlava, (b) trávící soustava, (c) koule, (d) vrtačka.*

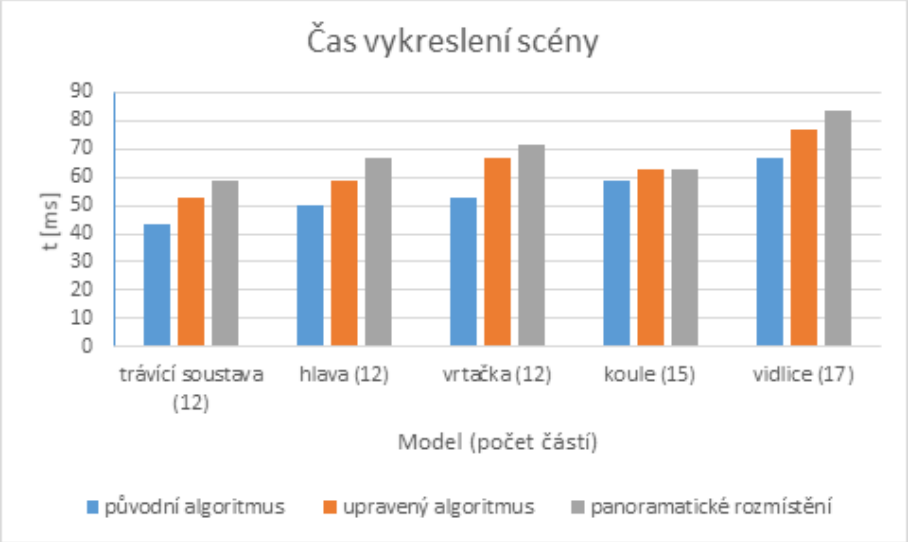
## 6.3 Rychlost vykreslení scény

V knihovně Tiger je možný výpis rychlosti vykreslování scény ve snímkové frekvenci (dále FPS - frames per second). Tato jednotka je však nelineární a jako taková se dle [21] nehodí k porovnávání rychlostí vykreslení scény. Pro tyto účely je dle [21] vhodné použít čas vykreslení snímku, který se již lineárně chová. Vztah mezi FPS a časem snímku zachycuje obrázek 6.20.




*Obrázek 6.20: Závislost FPS na FrameTime. Zdroj [21].*

Zhodnocením rychlosti výsledné implementace je možné učinit si představu o dopadu změn provedených v algoritmech na rychlost vykreslení scény. Čas vykreslení scény pro jednotlivé modely je zachycen na obrázku 6.21.

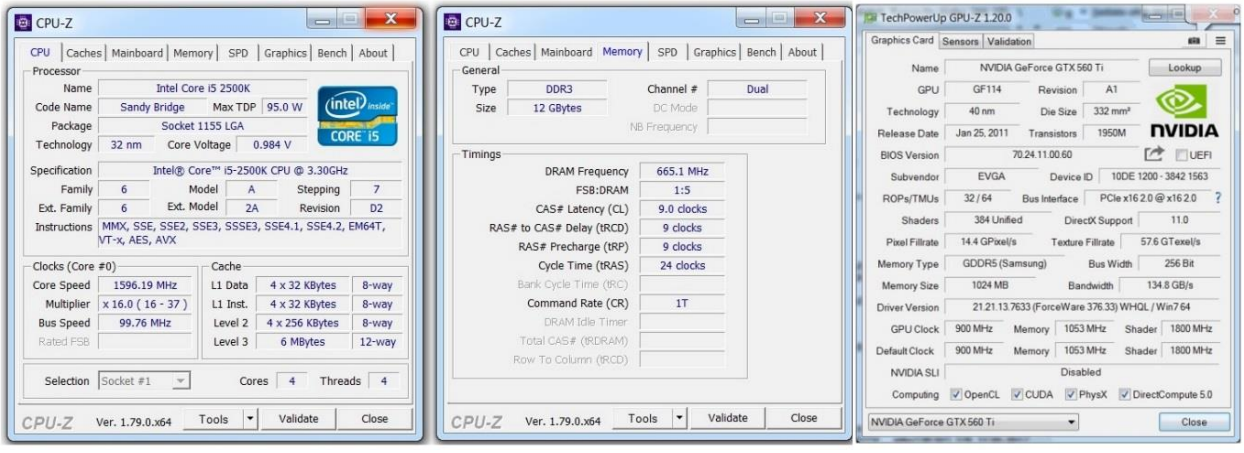



*Obrázek 6.21: Čas vykreslení scény*

Vertikální osa zachycuje střední hodnotu času potřebného k vykreslení scény společně s umístěním popisků. Samotný čas vykreslení scény je řádově jednotky ms a většina času tedy připadá na výpočet rozmístění popisků. Čísla v závorce u modelů jsou počty částí příslušného modelu (a tedy i popisků). Z grafu vyplývá existence vztahu mezi počtem popisků a časem vykreslení scény. Čím více popisků, tím déle trvá vykreslení scény. Toto tvrzení však neuvažuje podíl modelu na prostoru scény, který také bude ovlivňovat dobu výpočtu. Pro rozdílné modely se stejným počtem částí (trávící soustava, hlava, vrtačka) trvá tento proces jinak dlouho.

Zapojení summed area table do nového algoritmu se tedy projevilo ve zpomalení výpočtu rozmístění popisků. Vyšší kompaktnost rozložení popisků je na úkor doby jeho výpočtu. Při zapojení, na CPU vyhodnocovaného, panoramatického rozmístění popisků je výpočet ještě pomalejší.

Testování proběhlo pod operačním systémem Windows 7, při rozlišení okna 832x756 na hardwaru, jehož specifikace jsou zachyceny na obrázku 6.22




*Obrázek 6.22: Hardware. CPU (vlevo), RAM (uprostřed), GPU (vpravo)*

# Kapitola 7

# Závěr

Tato bakalářská práce měla dva hlavní cíle. Prvním cílem byla úprava algoritmu automatického umísťování popisků [5] takovým způsobem, aby umožnoval jejich umístění přímo na, obecně nekonvexní, hranici objektu. Druhým cílem bylo seznámení se s algoritmem panoramatického umístění popisků [19] a jeho zapojení do výpočtu rozložení popisků pro případy, kdy se popisky nacházejí nad a/nebo pod objektem.

První z těchto cílů naplňují kapitoly dva a tři. V kapitole druhé jsme si nejprve popsali problém spojeným s umístěním popisků přímo na hranici objektu a následně jsme si představili řešení v podobě detekce volného místa za využití summed area table. V sekci 2.4 jsme si popsali modifikovanou verzi algoritmu, která byla k tomuto účelu využita. V sekci 2.5 jsme se zabývali summed area table a analyzovali si způsob, kterým by bylo možné urychlit aktualizaci jejích hodnot a tím se vyhnout jejímu opětovnému výpočtu.

Kapitola tři přinesla přehled implementovaných změn. Každá změna byla nejdříve navrhnuta s odůvodněním její potřeby s následným uvedením způsobu implementace. Výsledkem této části práce je modifikovaný algoritmus automatického umísťování popisků, který je schopný popisky umístit přímo na hranici objektu. Při umístění nedochází k překryvům s objektem ani s jinými popisky. Toho bylo docíleno využitím summed area table a jejím updatem po každém umístění popisku do scény.

Druhý cíl naplňují kapitoly čtyři a pět. V kapitole čtvrté jsme si představili algoritmus panoramatického umístění popisků do scény [19]. Byly navrhnuty způsoby jeho zapojení do rozmístění popisků. První přístup spočíval v pouhém stanovení horizontu, nad kterým by se nevyskytovaly žádné další objekty, a využití původní verze algoritmu [19]. Druhý přístup při výpočtu rozmístění popisků zohledňoval i místo ve scéně. Dalé byl analyzován způsob zapojení informace o volném místu do tohoto algoritmu. Kapitola pátá prezentovala návrh reprezentace volného místa a způsob jeho získání s ohledem na potřeby algoritmu [19]. Výsledná modifikace algoritmu panoramatického umístění popisků však není zcela funkční a vyžaduje další práci na ní. Při výpočtu rozmístění popisků je tedy použita původní verze algoritmu způsobem popsaným v sekci 5.4.

V kapitole šesté byly oba algoritmy testovány na čtyřech modelech rozdílné složitosti a byla demonstrována jejich funkčnost. Zjistili jsme, že modifikovaný algoritmus automatického umístění popisků dosahuje při umístění popisků přímo na hranici objektu vyšší kompaktnosti jejich rozložení. Nejvíce rozdílné je rozmístění popisků v případě množiny koulí (viz obrázky 6.3 a 6.4). V případě dalších testovaných objektů je pak konvexní obálka objektu velmi podobná samotné hranici objektu a rozdíl v umístění popisků není tak značný.

Pro testování rozmístění popisků nad a/nebo pod objektem byl použit algoritmus panoramatického rozmístění popisků způsobem popsaným v sekci 5.4. Popisky jsou tedy umístěny nad

horizontem. Z výsledků, prezentovaných v sekci 6.2, vidíme, že rozmístění s použitím algoritmu panoramatického rozmístění popisků je výrazně lepší především v případě množiny koulí (viz obrázek 6.12). Zde dochází k výraznému snížení počtu řádek potřebných k umístění popisků. Na závěr této sekce jsou uvedeny také výsledky implementace modifikace tohoto algoritmu (viz obrázek 6.19), jak je popsána v sekci 4.2.

Testováním času potřebného k vykreslení scény je zhodnoceno zpomalení algoritmu, které přichází výměnou za rozšířenou funkcionalitu. Vykreslení scény s modifikovanou verzí algoritmu automatického umístění popisků je oproti verzi původní [5] na testovaných objektech přibližně o 10ms pomalejší. S využitím panoramatického rozmístění popisků se tento rozdíl ještě prohlubuje. Toto zachycuje obrázek 6.21.

## 7.1 Možné pokračování práce

Použitím jiného přístupu k zakomponování panoramatického rozmístění popisků do stávajícího projektu by mohlo být dosaženo lepších výsledků. Současný postup, jak navržen v sekci 4.2, má jisté limitace, které mohou být pravděpodobně odstraněny pouze částečně.

# Zdroje


[1] M. Benkert, H. Haverkort, M. Kroll, and M. Nöllenburg. Algorithms for multi-criteria one-sided boundary labeling, 2008.

[2] M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. Computational Geometry, 36(3):215–236, 2007.

[3] K. Hartmann, K. Ali, and T. Strothotte. Floating labels: Applying dynamic potential fields for label layout. In Smart Graphics, volume 3031/2004 of LNCS, pages 101–113. Springer, 2004.

[4] K. Ali, K. Hartmann, and T. Strothotte. Label layout for interactive 3d illustrations. Journal of the WSCG, 13(1):1–8, 2005.

[5] L. Čmolík and J. Bittner. Layout-aware optimization for interactive labeling of 3d models. Computers & Graphics, 34(4):378–387, 2010.

[6] M. Formann, F. Wagner, A packing problem with applications to lettering of maps, in: Proc. 7th Annual ACM Symposium on Computational Geometry (SoCG'91), 1991, pp. 281–288.

[7] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra. Fast summed-area table generation and its applications. Computer Graphics Forum, 24:547–555, 2005.

[8] T. Stein and X. D Décoret. Dynamic label placement for improved interactive exploration. In Proc. of NPAR '08, pages 15–21, New York, NY, USA, 2008. ACM.

[9] CROW F. C.: Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), ACM Press, pp. 207–212.

[10] Fuzzy set. *Wikipedia* [online]. [cit. 2017-01-15]. Dostupné z: https://en.wikipedia.org/wiki/Fuzzy_set

[11] OpenGL. *Wikipedia, the free encyclopedia* [online]. [cit. 2017-01-19]. Dostupné z: https://en.wikipedia.org/wiki/OpenGL

[12] OpenGL. *Wikipedie, otevřená encyklopedie* [online]. [cit. 2017-01-19]. Dostupné z: https://cs.wikipedia.org/wiki/OpenGL

[13] Java OpenGL. *Wikipedia, the free encyclopedia* [online]. [cit. 2017-01-20]. Dostupné z: https://en.wikipedia.org/wiki/Java_OpenGL

[14] JOGL - Java Binding for the OpenGL API - JogAmp [online]. [cit. 2017-01-20]. Dostupné z: http://jogamp.org/jogl/www/

[15] ČMOLÍK, Ladislav. [online]. [cit. 2017-01-20]. Dostupné z: http://dcgi.felk.cvut.cz/home/cmolikl/projects.html

[16] OpenGL® Registry. *OpenGL - The Industry Standard for High Performance Graphics* [online]. [cit. 2017-01-20]. Dostupné z: https://www.opengl.org/registry/

[17] Bekos MA, Kaufmann M, Symvonis A, Wolff A. Boundary labeling: models and efficient algorithms for rectangular maps. Computational Geometry 2007;36(3):215–36.

[18] STEIN, Thierry a Xavier DÉCORET. 2008. Dynamic label placement for improved interactive exploration. *Proceedings of the 6th international symposium on Non-photorealistic animation and rendering - NPAR '08*. New York, New York, USA: ACM Press

[19] Gemsa, Andreas, Jan-Henrik Haunert, and Martin Nöllenburg. *'Boundary-labeling algorithms for panorama images.*'Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems,pp. 289-298, ACM, 2011.

[20] Cmolik L. *Interactive Illustrative Visualization of 3D Models*. *PhD thesis*, Czech Technical University in Prague, 2011

[21] DUNLOP, Robert, FPS: A common yet flawed metric of game performance [online]. [cit. 2017-05-24]. Dostupné z: https://www.mvps.org/directx/articles/fps_versus_frame_time.htm