УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Владимир Н. Батоћанин

УНАПРЕЂЕЊЕ АЛГОРИТМА CLEARGRASP ПРИМЕНОМ ФИЛТЕРА ЗА ОДСЈАЈ ТРАНСПАРЕНТНИХ ОБЈЕКАТА

мастер рад

Београд, 2024.

Ментор:

др Александар Картељ, ванредни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

проф. др Младен Николић, ванредни професор
Универзитет у Београду, Математички факултет

др Иван Чукић, доцент
Универзитет у Београду, Математички факултет

Датум одбране: 15. септембар 2024.

Наслов мастер рада: Унапређење алгоритма ClearGrasp применом филтера за одсјај транспарентних објеката

Резиме: Један од главних циљева компјутерске визије и роботике је омогућавање интеракције са простором на основу минималне количине визуелних информација. Пошто савршена перцепција простора захтева велику количину улазних информација, да бисмо сузили скуп неопходних информација, морамо се ослонити на вештачку интелигенцију.
Модерна решења за просторну перцепцију у реалном времену најчешће користе камере које паралелно снимају обичну слику и информације о дубини. Такав приступ је адекватан за већину објеката, јер нам дубинске информације дају добар осећај положаја објекта у простору. Мана овог приступа је то што се технологије мерења дубине базирају на принципу одбијања ласера, односно снопова светлости, што имплицира да овакав принцип не можемо користити на транспарентним објектима. Највећи проблем код транспарентних објеката је то што не одбијају правилно светло ка његовом извору, већ га распршују на различите начине у зависности од облика самог транспарентног објекта.
Одговор на овај проблем је био алгоритам *ClearGrasp*, који је користећи неколико дубоких неуронских мрежа процењивао позицију и облик транспарентних објеката. Мана овог алгоритма је то што комплексне спекуларне одсјаје, односно каустике (енг. caustics), третира у одређеном броју случајева исто као транспарентне објекте. Овај рад ће се бавити овим типом грешке, као и предлогом модификације оригиналног алгоритма којим би се смањио број грешака овог типа.
Рад ће анализирати ClearGrasp алгоритам и покушаће да поједностави поступак самосталног генерисања синтетичких података за његов улаз и тестирање, као и будуће претренирање. Бавиће се и развојем и тренирањем дубоке неуронске мреже за сегментацију каустика; и имплементацију те мреже у ClearGrasp пројекат као и његовом поновном евалуацијом.

Кључне речи: рачунарски вид, дубоке неуронске мреже, вештачка интелигенција, транспарентни објекти, оптика, просторна перцепција, камере, филтер за одсјај, ClearGrasp, спекуларни одсјаји, каустици, сегментација

Садржај

Глава 1

Увод

1.1 Мотивација

Просторно мапирање објеката постаје све важнији задатак у савременом свету, због развоја и унапређења бројних система, технологија и научних дисциплина, као што су аутономна вожња, проширена реалност, роботика, здравство, и многе друге.

У области аутономне вожње, просторно мапирање је кључно за навигацију и безбедност возила. Самовозећи аутомобили користе тродимензионалне мапе за прецизно детектовање и избегавање препрека, као и за планирање путања у реалном времену. Ово значајно повећава ниво безбедности у саобраћају и смањује ризик од несрећа.

Здравство такође има велике користи од просторног мапирања, посебно у хирургији и дијагностици. Прецизне тродимензионалне мапе омогућавају хирурзима да боље планирају и изводе сложене оперативне захвате, док дијагностичари могу да добију детаљније слике унутрашњих структура пацијената, што побољшава тачност дијагноза.

И у развоју савремених технологија виртуелне и проширене реалности, просторно мапирање се користи за креирање реалистичних и интерактивних окружења.

За овај рад је најбитнија примена просторног мапирања у производној роботици где се користи за навигацију и манипулацију објеката. Роботи користе тродимензионалне мапе да би прецизно хватали и обрађивали производе, чиме се повећава ефикасност и продуктивност производних процеса.

У свим овим дисциплинама, а нарочито у роботској аутоматизацији и

проширеној реалности, може се јавити случај да програм мора да препозна транспарентни објекат као на пример чашу, флашу или прозор.

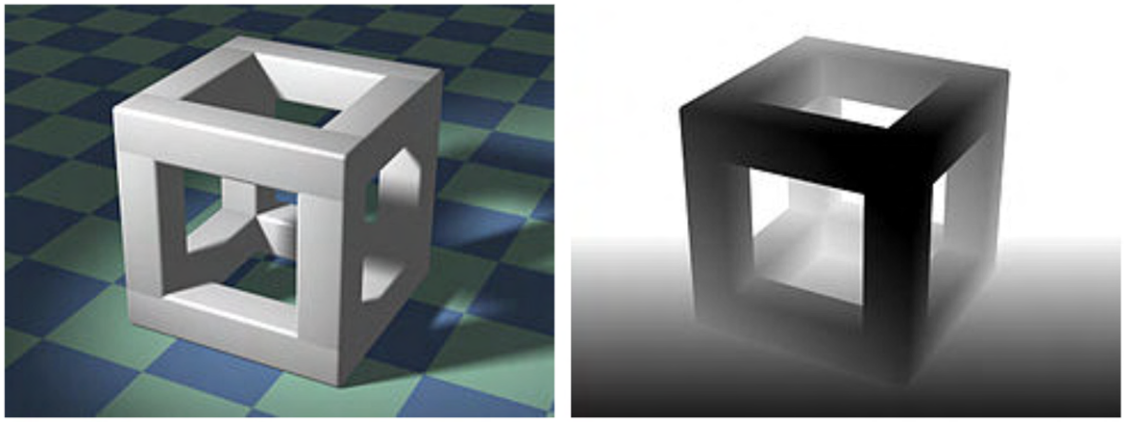
У таквим моментима настаје озбиљан проблем јер информације о дубини прозирних објеката нису веродостојне.

У разумевању овог проблема помоћи ће нам основни принципи мерења дубине и просторног мапирања.

1.2 Мерења дубине

Мерење дубине представља апроксимирање раздаљине садржаја сваког појединачног пиксела од позиције камере. Те вредности представљају вредности z-координате сваког пиксела у сопственом координатном систему камере.

Прикупљене информације о дубини се пакују у црно-белу слику на којој нијансе боје одговарају удаљености објекта од камере. Слика 1.1 приказује пример колор слике и њене одговарајуће дубинске слике.

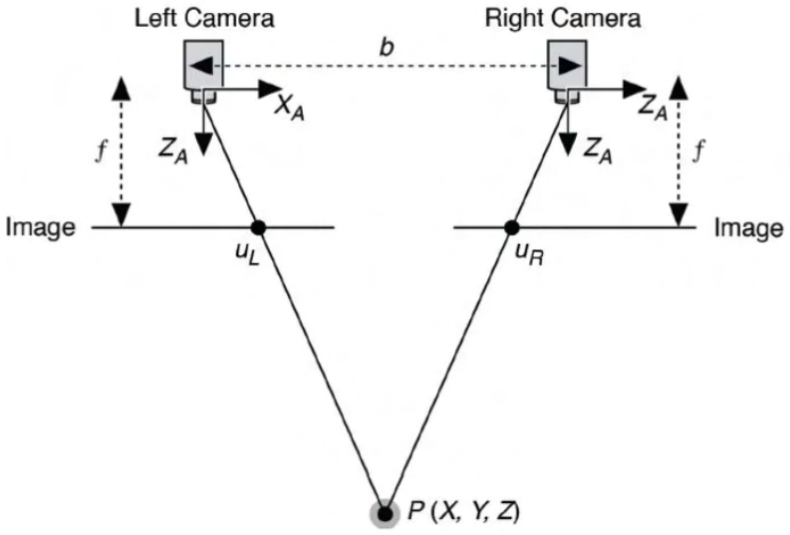


Слика 1.1: Слика коцке у боји (лево) и одговарујуће слике са дубинским информацијама (десно). На овој дубинској слици тамнији пиксел означава већу близину камери.

Стерео вид

Један од начина мерења дубине је *стерео вид*. Ова техника користи две камере постављене на фиксној удаљености једна од друге. На сличан начин функционишу људске очи. Налажењем неколико истих пиксела из обе

перспективе можемо проценити дубину кроз процес триангулације[6]. Овај процес је приказан на слици 1.2.



Слика 1.2: Стерео вид са две камере

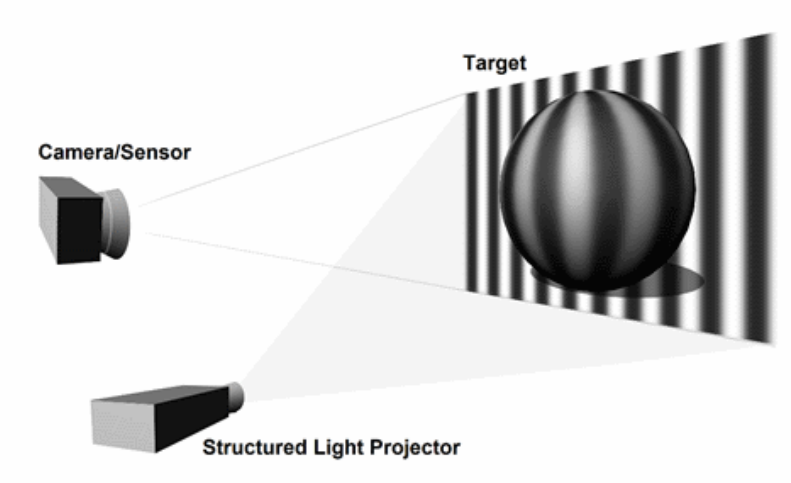
Иако је ова метода природна и интуитивна, она има велике проблеме са транспарентним објектима. Недостатак видљивих текстура и компликована рефлексија отежава проналажење истих тачака на обе слике, што значајно компликује прецизно одређивање дубине.

Структурирана светлост

Дубину можемо мерити и помоћу *структуриране светлости* (енг. *Structured Light*). Код овог приступа се познати светлосни образац пројектује на сцену или објекат. Тај образац може бити у облику тачака, линија или мреже, и он се емитује из извора светлости који је део камере или је одвојен уређај.

Када светлост удари у објекат, образац се деформише у складу са обликом и удаљеношћу површине од извора светлости. На пример, ако светлост удари у избочину на објекту, тај део обрасца ће бити више деформисан у односу на део обрасца који удари у равну површину (види слику 1.3).

Камере уређаја снимају деформисани образац светлости који се враћа са објекта. Ове снимке затим анализира софтвер који је дизајниран да препозна и интерпретира деформације у обрасцу. Софтвер користи ове информације да би израчунао удаљеност сваке тачке на објекту од камере. На овај начин



Слика 1.3: Илустрација структуриране светлости са одвојеном камером и извором структурираног светла

се добија тродимензионална мапа објекта, која верно представља његов облик и контуре.

1.3 Основе просторног мапирања

Просторно мапирање омогућава реконструкцију тродимензионалне структуре сцене од једне или више дводимензионалних слика. Овај процес захтева подешавање камере и примену алгоритама за детекцију и праћење карактеристичних тачака. Коришћење више перспектива и углова снимања доприноси прецизности реконструкције сцене.

Први корак у просторном мапирању је **калибрација камере**. Овај корак подразумева одређивање унутрашњих параметара камере, као што су жижна даљина и тачка пресека, и спољашњих параметара, који обухватају положај и оријентацију камере у односу на сцену.

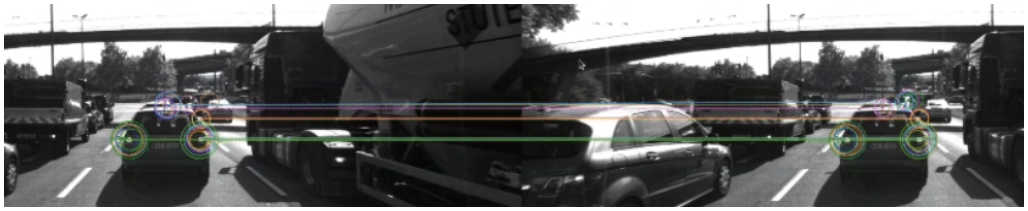
Пример једне поставке више камера у сврху просторног мапирања је приказан на слици 2.3.

Следећи корак је **детекција карактеристичних тачака**. Слично као у стерео виду, те тачке су лако препознатљиве на слици и играју кључну улогу у праћењу кретања. Пример слике са обележеним караткеристичним тачкама је приказан на слици 1.4.

Циљ обележавања ових тачака је омогућавање алгоритму да разуме како

се тачке померају у односу на камеру. Праћење тачака између слика може да буде корисно и у случају праћења померања тачака у серији слика ухваћених из истог угла са малим временским размаком, али и у случају симултаног снимања истих тачака из различитих перспектива. Слика 1.5 приказује праћење тачака у серији слика, док слика 2.3 приказује праћење из више симултаних перспектива.

Слика 1.4: Пример карактеристичних тачака



Слика 1.5: Праћење карактеристичних тачака

На основу података о померању карактеристичних тачака и параметара камере, могуће је израчунати просторне координате тачака у сцени. Овај процес се ослања на принципе пројективне геометрије, која је основа за многе прорачуне у просторној реконструкцији.

Прорачун за просторну перцепцију користи пројективну матрицу P за пројекцију тродимензионалне тачке $\mathbf{X} = (X, Y, Z, 1)^T$ на дводимензионалну слику $\mathbf{x} = (u, v, 1)^T$. Ова пројекција се описује формулом

$$\mathbf{x} = P\mathbf{X}$$

где је P матрица димензија 3×4, која садржи параметре унутрашње и спољашње калибрације камере. Пројективна матрица се може поделити на два дела: K, који представља матрицу унутрашње калибрације, и $[R|t]$,

који представља матрицу спољашње калибрације, укључујући ротацију R и транслацију камере t у односу на сцену.

Реконструкција тродимензионалних тачака коришћењем *стерео визије* подразумева решавање система једначина који произилазе из пројективних трансформација. За две камере са пројективним матрицама P_1 и P_2, и одговарајуће дводимензионалне тачке $\mathbf{x}_1$ и $\mathbf{x}_2$, систем једначина гласи:

$$\mathbf{x}_1 \times (P_1\mathbf{X}) = 0$$

$$\mathbf{x}_2 \times (P_2\mathbf{X}) = 0$$

Овај систем линеарних једначина може се решити коришћењем *SVD* декомпозиције, чиме би се добиле тродимензионалне координате тачке $\mathbf{X}$.

Иако је процес просторног мапирања веома моћан, он се суочава са бројним препрекама. Шум и нетачности у подацима могу значајно утицати на детекцију и праћење карактеристичних тачака. Сцене са мало текстура или обрасцима који се понављају, као што су зидови или плочице, могу довести до неодређености у праћењу тачака. Промене у осветљењу и сенке могу такође значајно утицати на детекцију и праћење тачака, и могу бити узрок грешака у реконструкцији. Присуство покретних објеката у сцени може додатно ометати процес мапирања, јер ови објекти не прате исту геометрију као статични делови сцене. На крају, просторно мапирање захтева значајну рачунарску моћ, посебно код обраде великих сетова слика и сложених сцена.[11]

Због оваквих пројективних принципа, немогуће је направити верну репрезентацију неког објекта без барем два извора информација, а то је основа стерео перцепције. Уколико је доступна само једна слика, реконструкција простора постаје неодређена јер недостаје информација о дубини. Овај недостатак можемо надоместити коришћењем додатних извора информација као што су очитавање дубине помоћу ласерских или инфрацрвених сензора. Ови сензори мере време које је потребно светлости да се врати до њих након одбијања од објекта, чиме обезбеђују прецизне информације о дубини и омогућавају верније реконструкцију простора.

1.4 Оптичка својства транспарентних објеката

Прикупљање информација о дубини највише зависи од хардвера који користимо. У овом раду ћемо се фокусирати на камеру *Intel RealSense D415* због тога што се користи у алгоритму *ClearGrasp*[10], и зато што користи два поуздана метода израчунавања дубине паралелно:

1. стерео вид (види поглавље 1.2)
2. структурирана светлост (види поглавље 1.2)

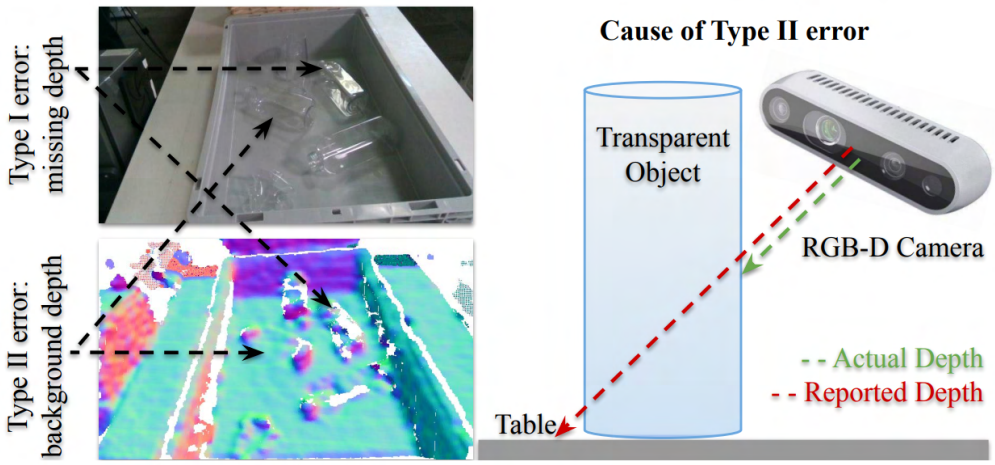
Транспарентни материјали, као на пример стакло, имају две карактеристичне особине које отежавају лако очитавање дубине: прозирност и преламање светлости.

- Прозирност одређује колико светлости може проћи кроз стакло, и изузетно је висока у случају транспарентних материјала као што је стакло
- Преламање светлости је промена правца светлосног зрака приликом проласка кроз одређени материјал

Прозирност изазива привид бесконачне удаљености приликом очитавања дубине. То је последица чињенице да се светло које напусти извор никада не врати јер просто прође кроз транспарентни објекат и настави своју путању, осим у случају када посматрани зрак светла не одбије други објекат. Тако да, иако добијемо некакво очитавање, оно не описује облик транспарентног објекта, већ његове позадине.

Прозирност изазива грешке типа 2 приликом очитавања дубине, где се уместо дубине површине посматраног објекта очита дубина позадине иза објекта[10]. Ова појава је илустрована на слици 1.6.

Преламање светлости је узрок дијаметрално супротне грешке, јер изазива скупљање светлосних зрака у облике који су визуелно слични спекуларној рефлексији нетранспарентних објеката. Због тога што ти облици представљају области повећаног светлосног флукса, дубинска очитавања оваквих области нису прецизна јер се много различитих зракова може очитати са истог места, што доводи до јављања „слепе тачке" (преоптерећење сензора) у дубинским очитавањима[10].



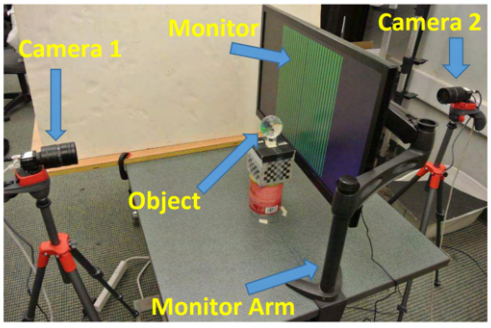
Слика 1.6: Грешке приликом очитавања дубине типа 1 и типа 2

Слепе тачке престављају грешке типа 1. Њихов изглед се може видети на слици 1.6.

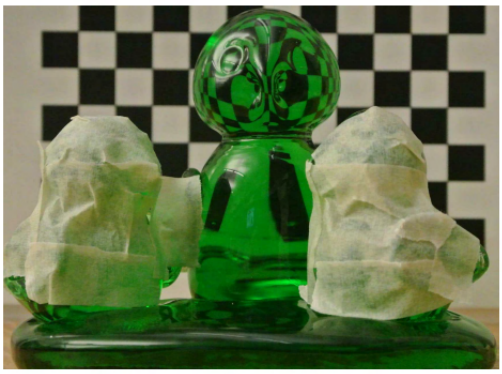
Оба ова проблема можемо решити бацањем сенки. Сенке пружају додатне информације о облику и релативној позицији објеката у сцени. Када је објекат осветљен из одређеног правца, сенке се могу користити за одређивање облика и положаја објекта. Различити примери примене овог концепта су приказани на слици 1.7.

Слика 1.7: Скенирање облика помоћу бацања сенки: (a) подешавање камере са тачкастим извором светлости (стона лампа без рефлектора) и штап који се држи у руци и баца сенку; (b) објекти који се скенирају испред две планарне позадине и (c) мапа дубине у реалном времену добијена коришћењем система пулсирајућег осветљења.

Ова метода се користи код мапирања непрозирних објеката, али је могуће модификовати је да би функционисала и за прозирне објекте. Уколико направимо калибрациони дијаграм, који је искључиво састављен од правих углова и предвидивих облика, и ставимо га иза камере и прозирног објекта, можемо веома сличним процесом израчунати облик прозирног објекта[8].



Слика 1.8: Снимање облика стакленог предмета посматрањем деформације правих линија пројектованих на *LCD* екран



Слика 1.9: Снимање облика стакленог предмета посматрањем деформације статичног шаблона који је постављен у његовој позадини

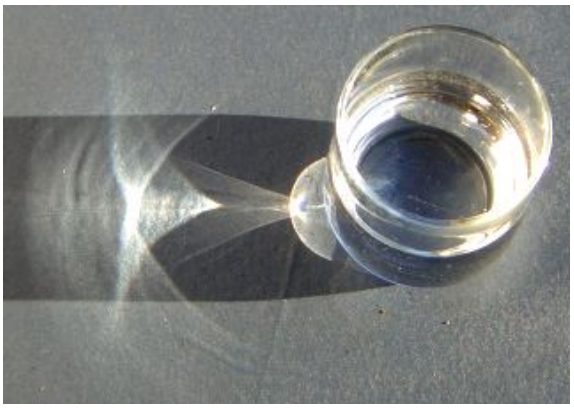
Ова модификација је врло слична структурираној светлости. Једина разлика је у томе да ли се посматра светлост која се емитује из камере или се посматра предефинисани шаблон који постављамо у позадини, као на сликама 1.8 и 1.9. Овај процес ће касније бити детаљније обрађен.

Глава 2

Просторно мапирање транспарентних објеката

2.1 Каустици

За објашњење оптике транспарентних објеката значајан је појам каустици (енг. *caustics*). Каустици су светлосне шаре које настају када светлост пролази кроз транспарентне материјале. Ове светлосне шаре се често могу видети као криве или мреже светлости које се пројектују на површинама. Оне настају због фокусирања светлосних зрака и могу бити врло сложене.



Слика 2.1: Каустици који се стварају када светлост прође кроз стаклену чашу

На пример, када сунчева светлост пролази кроз чашу воде, она се прелама и фокусира у различитим тачкама, стварајући светле, таласасте обрасце на дну или поред чаше као на слици 2.1.

Каустично инжењерство представља математичко одређивање контуре објекта који производи одређене каустике. Конкретно се бави проучавањем и применом карактеристичних облика каустика који настају услед рефлексије или преламања кроз закривљене објекте, стварајући концентрисане светлосне линије или криве.

Ова дисциплина се не бави само одређивањем облика објеката који могу да произведу дате каустике, већ се бави и дизајном и пројектовањем објеката који могу да произведу одсјај жељеног облика.

Природа каустика и предвидивост њиховог облика су нам битне информације за њихово визуелно препознавање.

2.2 Методе мапирања које захтевају фиксне предуслове

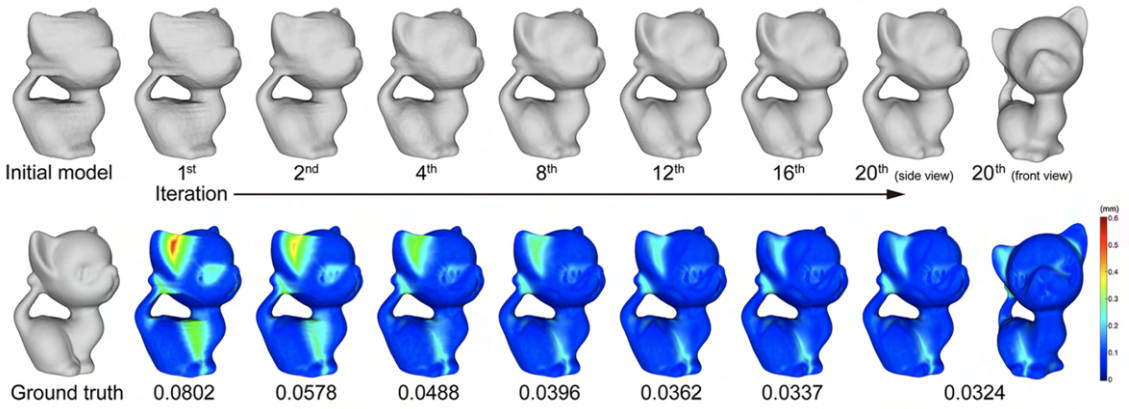
За одређивање облика транспарентних објеката користи се њихова особина да преламају и преусмеравају светлосне зраке.

Ако поставимо објекат на ротирајуће постоље испред екрана са усправним линијама које емитују светло и ако снимимо профил објекта током ротације, можемо реконструисати његов тродимензионални облик (види слику 2.3). Камере бележе силуете и путеве рефракције светлости, што омогућава креирање иницијалног грубог модела објекта.

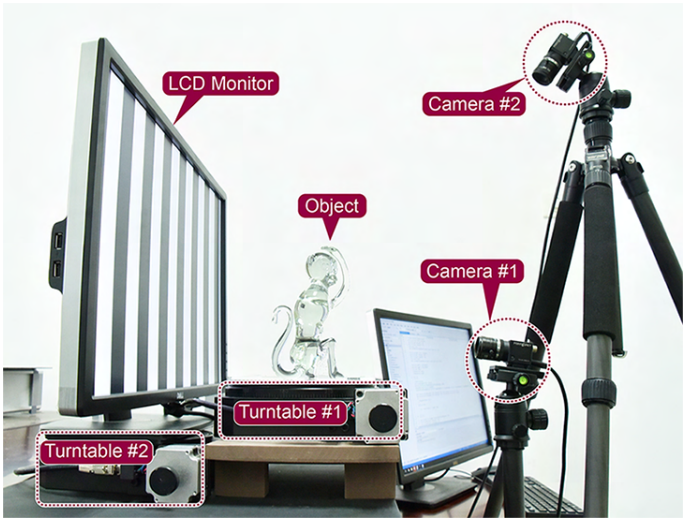
Овај модел можемо оптимизовати ако узмемо у обзир конзистентност површинских нормала и рефракције, као и конзистентност пројекције површине и силуете, чиме се осигурава глаткоћа површине. На овај начин добијамо тачан и детаљан 3D модел транспарентног објекта[12].

Овакав приступ се може користити и за глатке и за објекте са оштрим ивицама, ово је могуће јер алгоритам у свакој итерацији тежи да одржи паралелно и силуету објекта са свих страна као и да одржи глатке површинске нормале (види слику 2.2). Ова два параметра терају оптимизацију да тежи у супротним смеровима, што нам даје равнотежу између прецизности у дефинисању ивице и глаткоће површине, омогућавајући постизање добре репрезентације како за глатке тако и за оштре делове објекта.

За потпуну математичку тачност, уводи се ротација и самог екрана око своје осе. Овим можемо да упаримо конфигурације позиције камере и екрана



Слика 2.2: Процес итеративне минимизације грешке одступања модела од референтног модела. Кроз итерације се прецизније приказују доље и узвишења.

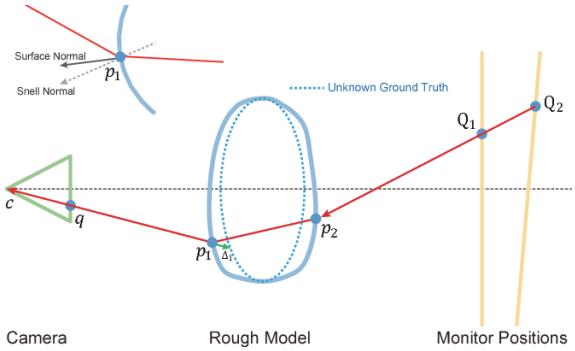


Слика 2.3: Поставка камера, екрана и окретних сталака.

тако да добијемо исти зрак односно рефлексију у два случаја као на слици 2.4, чиме можемо комплетно да одредимо облик објекта.

Једно од великих ограничења овог метода је то што ради само под претпоставком да се светло пре враћања у камеру преломи само два пута. Ово није случај код компликованијих геометријских тела. Да би ова метода функционисала и за њих, потребно је далеко више мерења, камера и сензора.

Друго, много важније ограничење је спорост алгоритма, што га чини неприкладним за коришћење у реалном времену. Ово је значајно јер у многим случајевима није потребна потпуна тачност, већ само оквирна претпоставка о облику транспарентног објекта. На пример, у индустријским апликацијама



Слика 2.4: Израчунавање тачке контакта зрака са транспарентним објектом $p1$ и $p2$ који производи илуминацију пиксела на слици q, одређено уз помоћ две конфигурације екрана као извора светла Q_1 и Q_2.

где је брза процена облика важна за контролу квалитета или аутоматизоване процесе.

Због потребе за брзим резултатима, многе методе мапирања су прешле на коришћење модела машинског учења, који могу да обезбеде потребну брзину и прихватљив ниво тачности.

2.3 Сегментација слика

Сегментација слике је процес дељења слике на саставне делове и издвајања објеката који нас занимају. Улаз сваког алгоритма за сегментацију је слика, док је излаз иста та слика са додатим лабелама (или вероватноћама могућих лабелирања) за сваки појединачни пиксел. У процесу тренирања модела за сегментацију одреди се скуп могућих вредности лабела. На пример, код детекције објеката у саобраћају, различите класе могу бити „возило”, „пешак” или „пут” (види слику 2.5)[3].

Бинарна маска (енг. *binary mask*) је матрица Булових променљивих исте величине као и улазна слика процеса сегментације, она нам одређује којим пикселима слике припада одређена класа. Бинарне маске се користе за издвајање делова слике који су нама релевантни. Множењем колор слике и одговарајуће бинарне маске добија се означени део слике (види слику 2.6).

Слика 2.5: Резултат сегментације слике улице. Боје означавају пикселе којима је додељена иста лабела.

Слика 2.6: Процес изолације дела слике уз помоћ бинарне сегментационе маске

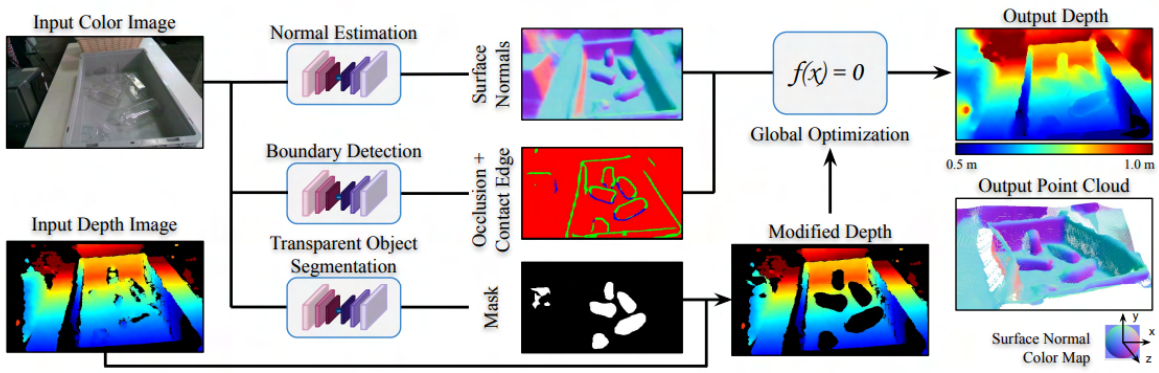
2.4 ClearGrasp алгоритам

Један од најбољих примера методе која користи моделе машинског учења је алгоритам *ClearGrasp*.

Аутори овог алгоритма су покушали да реше проблем перцепције простора са прозирним објектима.

Користили су камеру за детекцију дубине *Intel RealSense D415*, која производи неисправне информације за прозирне објекте, али тачне за непрозирне објекте. Развили су дубоке конволуционе мреже за предвиђање: нормала објеката на слици, бинарне сегментације прозирних објеката и за ободе објеката као и њихове додирне тачке. Искористили су предвиђену сегментацију да би избацили фаличне пикселе са оригиналног очитавања дубине и тиме добили слику дубине са рупама где би требало да буду прозирни објекти. Укомбиновали су модификовану дубину са рупама, предвиђене нормале и предвиђене ободе у функцију глобалне оптимизације (види слику 2.7). Као

резултат свега овога добили су исправљену дубину.



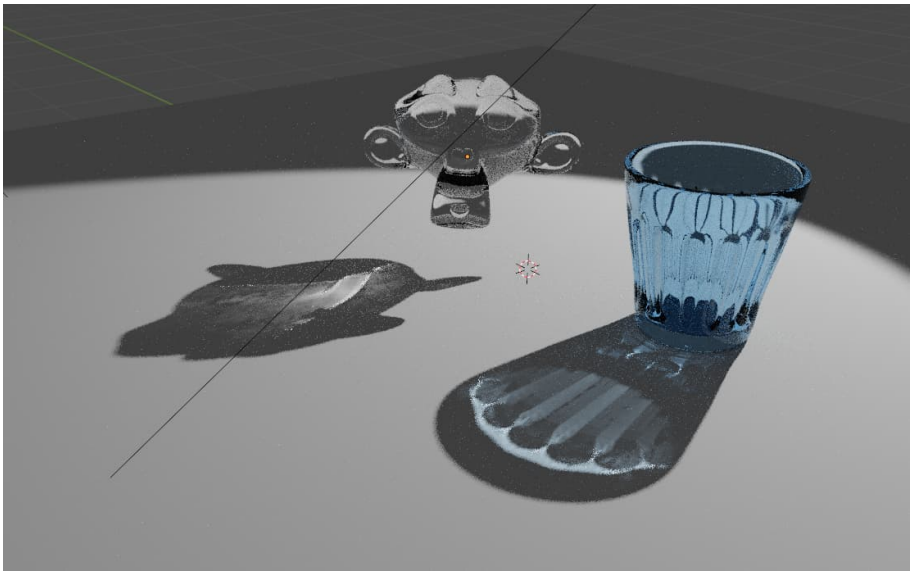
Слика 2.7: Дијаграм обраде података у алгоритму *ClearGrasp*

Алгоритам *ClearGrasp* је решио грешке очитавања дубинских информација типа 1 и 2 (види слику 1.6) тиме што је третирао податке очитане са сензора као фаличне, и накнадно их исправљао уз помоћ конволутивних мрежа. Илустрација овог процеса је приказана на слици 2.7.

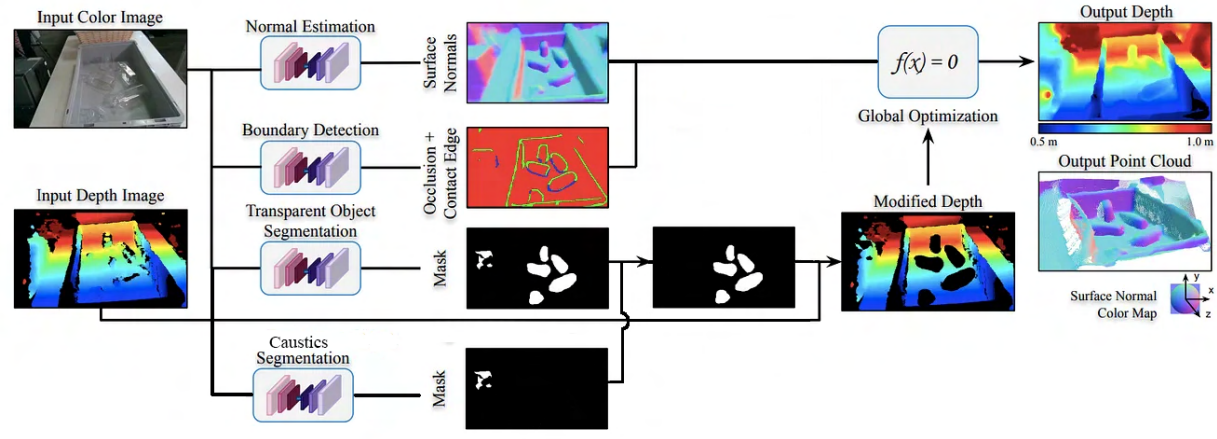
Али, остала су ограничења у случајевима када постоје каустици на слици. И спекуларна рефлексија и каустици представљају већу концентрацију одбијеног светла на релативно правилној кривој, што доводи до тога да их алгоритам не разазнаје и препознаје каустике као ободе транспарентних објеката[10]. Овај проблем настаје јер је облик каустика врло уско везан са обликом одговарајућег транспарентног објекта (пример на слици 2.8). Ову појаву објашњава *каустично инжењерство* (обрађено у секцији 2.1)

У овом раду се разматра решење овог проблема кроз оптимизацију која би укључивала додатну дубоку конволуциону мрежу за прецизнију сегментацију каустика и прозирних објеката. Ова додатна мрежа би генерисала маске које јасно раздвајају каустичке ефекте од стварних прозирних површина (види слику 2.9).

Такве маске би се укомбиновале са постојећим маскама прозирних објеката, чиме бисмо добили доста прецизније оивичене прозирне објекте. Овим се омогућава боља корекција дубинских података и побољшање укупне тачности модела у сценама са сложеним светлосним условима.



Слика 2.8: Визуелна сличност каустика и одговарајућих транспарентних објеката



Слика 2.9: Дијаграм обраде података у модификованом алгоритму

Глава 3

Имплементација дубоке неуронске мреже за препознавање каустика

3.1 Генерисање синтетичких података

За унапређење алгоритма *ClearGrasp* велику препреку представља генерисање података за тренирање, валидацију и евалуацију.

Аутори алгоритма *ClearGrasp* су комбиновали праве фотографије са синтетичким сликама које су добили од *Synthesis AI*, компаније која се бави генерисањем база знања за тренирање модела машинског учења.

Синтетичке податке су генерисали уз помоћ софтвера за моделовање и симулацију 3D објеката *Blender*, док су само исцртавање слика као и симулацију реалистичног светла вршили у његовом интегрисаном програму за исцртавање и праћење зракова (енг. *ray tracing*) *Cycles*.

Овај рад ће се фокусирати на генерисање реалистичних синтетичких слика иако су аутори алгоритма *ClearGrasp* напоменули да је генерализација модела боља у случају да се користе и синтетички подаци и фотографије пошто прављење референтне истине за локацију каустика изискује две идентичне слике, једну са каустицима, другу без каустика. Ово би било немогуће у случају фотографија, и захтевало би ручно лабелирање слика. Са друге стране, синтетички подаци добијени уз помоћ 3D модела имају много метаподатака о околини и сцени, и тиме могу да генеришу мноштво корисних слика за тренирање модела, као што су:

1. Прецизне контуре објеката
2. Тачне површинске нормале
3. Тачне дубинске информације

3.2 Аутоматизација генерисања података

Као почетну тачку за процес генерисања слика употребићемо рад *SuperCaustics* [5], који се базира на генерисању слика преко окружења за развој *Unreal Engine* и софтвера за симулацију светла и рефлексија *Lumen*.

Главна предност софтвера *Lumen* је то што је оптимизован за исцртавање реалистичног светла у реалном времену. Он је оптимизован да може да подржи неколико светлосних извора концентрисаних на конкретан део сцене. Ово ће много олакшати и убрзати процес синтетисања података.

Генерисање новог синтетичког скупа слика подразумева прикупљање 3D модела који су по облику и карактеристикама довољно слични моделима коришћеним у ClearGrasp алгоритму — чаше и шоље за прозирне објекте, пластичне посуде, металне кугле, као и неколико нових објеката са комплекснијим контурама.

Након тога је дефинисан стаклени материјал у *Unreal Engine* формату којим су дефинисане светлосне интеракције са прозирним објектима. Дефинисано је и неколико различитих непрозирних материјала за остале објекте за тренирање, са варирајућим мерама спекуларне рефлективности, храпавости и боје ради боље генерализације. Одређени су и материјали за подлогу.

Подешена је гравитација за сцену као и насумично постављање објеката који ће падати са висине, да би пали у сцену и слетели у насумичне позиције. Изабрано је неколико углова из којих ће се снимати тако да модел буде инваријантан на перспективу и релативну величину објеката.

Дефинисане су команде за укључивање и искључивање напредног исцртавања светла, приказивања нормала, контура и дубине.

Прилагођена је скрипта за генерисање базе знања тако да исту сцену ухвати са различитим светлосним условима и из више различитих углова. Скрипта се покреће и за сваку расподелу објеката генерише следеће слике:

1. Слика са укљученим праћењем трајекторије светла али искљученим каустицима

2. Слика са укљученим праћењем трајекторије светла и каустицима
3. Слика површинских нормала
4. Слика са дубинским информацијама
5. Слика само са означеним ободима и ивицама објеката
6. Маска као резултат сегментације транспарентних објеката

3.3 Претпроцесирање слика за проблем сегментације

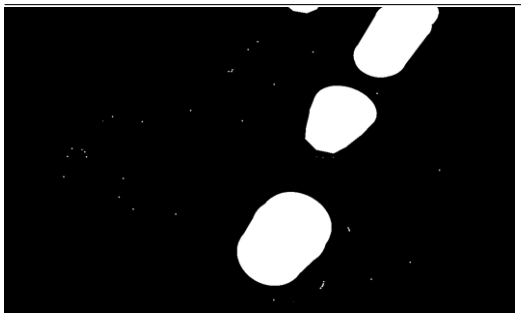
Након генерисања слика морамо да изолујемо информације о каустицима, илити да направимо сегментационе маске које ће нам обележити на ком делу слике се налазе каустици.

Због природе каустика, вредности осветљености пиксела каустика ће увек бити веће од вредности истих пиксела без каустика. Ово важи из једначине за израчунавање осветљености пиксела на основу његових RGB вредности:

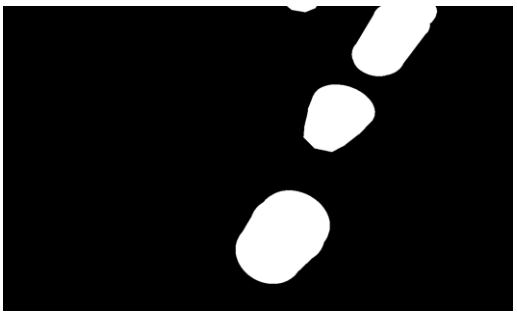
$$Y = 0.2126 \cdot R_{\text{lin}} + 0.7152 \cdot G_{\text{lin}} + 0.0722 \cdot B_{\text{lin}}$$

Ова формула представља начин израчунавања релативне осветљености користећи линеарне компоненте RGB спектра. R_{lin}, G_{lin} и B_{lin} су линеаризоване вредности црвене, зелене и плаве компоненте боје. Коефицијенти 0.2126, 0.7152 и 0.0722 одражавају колико доприноси свака од ових боја у људском виду. Због тога што људско око највише осећа зелену светлост, коефицијент за зелену компоненту је највећи, док је коефицијент за плаву компоненту најмањи[7].

Што је боја ближа белој, осветљеност је већа, па можемо изоловати каустике одузимањем тих двеју слика на нивоу пиксела. Резултат ове операције је разлика осветљености коју треба претворити у сегментациону маску процесом бинаризације. Ту маску треба и очистити филтерима да би се изглачале вредности маске (види имплементацију 3.1). Глачање је неопходно због тога што су нам битне веће повезане области на маски, а не разлике на нивоу индивидуалних пиксела (види слике 3.1 и 3.2).



Слика 3.1: Сегментациона маска пре глачања



Слика 3.2: Сегментациона маска после глачања

```
import PIL.Image as Image
import PIL.ImageChops as ImageChops
import PIL.ImageFilter as ImageFilter
def extract_caustics_segmentation(
    path_img_with_caustic,
    path_img_no_caustic,
    path_segmentation,
    threshold=10
):
    WHITE_PIXEL = 255
    BLACK_PIXEL = 0

    img_with_caustic = Image.open(path_img_with_caustic).convert('RGB')
    img_no_caustic = Image.open(path_img_no_caustic).convert('RGB')

    # Convert image to a binary black/white image
    img_segmentation = Image.open(path_segmentation).convert('1')

    # Subtract the images
    acc_image = ImageChops.subtract(img_with_caustic, img_no_caustic)

    # Function for binarizing pixels
    fn = lambda x: WHITE_PIXEL if x > threshold else BLACK_PIXEL

    # Convert the result to grayscale an binarize it
    r = acc_image.convert('L').point(fn, mode='1')

    # Remove noise with min then max filtering
    p = ImageChops.subtract(r, img_segmentation)
```

```
30      p = p.filter(ImageFilter.MinFilter(5))
31      p = p.filter(ImageFilter.MaxFilter(5))
32
33      return p
```

Имплементација 3.1: Процес извлачења маски са каустицима

Није потребно чувати сегментационе маске у пуном квалитету (у *RGB* формату). Маске семантички само садрже једну бинарну информацију, да ли је нешто одређени тип објекта или не, док колор слике чувају 3 информације јер се обично користе за приказ боја, тако да ћемо их претворити у црно-бели формат како бисмо уштедели меморијски простор (види имплементацију 3.2).

```
1  import os
2  import numpy as np
3  import cv2
4  from PIL import Image
5
6  def compress_and_save_image(
7      input_path,
8      output_path,
9      process_function
10 ):
11     img = Image.open(input_path).convert('L')
12     arr = np.array(img)
13     processed_arr = process_function(arr)
14     cv2.imwrite(output_path, processed_arr)
15     print(os.path.basename(input_path) + ' Processed.', end='\r')
16
17 def process_segmentation(arr):
18     return np.where(arr >= 220, 255, -1)
19
20 def process_outline(arr):
21     return np.floor(arr / 126)
```

Имплементација 3.2: Функција за компресију слика

3.4 Манипулисање и аугментација базе знања

За манипулацију подацима и тренирање новог модела за сегментацију ћемо користићемо *Python* библиотеку за машинско учење *PyTorch*.

Када се ради са великом количином слика уско грло је иницијално учитавање сваке појединачне слике са дуготрајне меморије. Стога се препоручује да се све слике учитају у радну меморију и мапирају у хеш табели. Након овога им се у константном времену $O(1)$ може приступити без учесталог чекања.

Аугментација

Приликом сваког приступа сликама за тренирање имплементираћемо да се примене одређене визуелне трансформације да би се повећала робусност модела, односно да бисмо добили модел који је независтан од оштрине, оријентације и увећања слике. Аугментација ће нам помоћи да варирамо свако својство слике које нам је небитно, тако модел може да научи да се искључиво фокусира на битна својства слике, чиме добијамо модел који боље генерализује[2].

У процесу аугментације (види имплементацију 3.3), коришћене су следеће трансформације:

- Динамичко одређивање величине и насумично сечење слике
- Наношење насумичног замућења
- Насумично хоризонтално окретање слика са вероватноћом од 50%, чиме се модел учи на варијантама објеката који могу бити различито оријентисани

```
from torch.utils.data import Dataset
from torchvision import transforms
from torchvision.transforms.functional import crop
from PIL import Image, ImageFilter
import os

class SegmentationDataset(Dataset):
    def __init__(
        self,
        image_dir,
        mask_dir,
        indices=None,
        transform=None,
        crop_size=(256, 256),
```

```
        max_blur_radius=2,
        is_train=True
    ):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
        self.crop_size = crop_size
        self.max_blur_radius = max_blur_radius
        self.is_train = is_train

        # Load images and masks into a dictionary
        self.data_dict = self._load_data()
        self.images = list(self.data_dict.keys())

        # Subset the dataset based on provided indices
        if indices is not None:
            self.images = [self.images[i] for i in indices]

    def _load_data(self):
        data_dict = {}
        image_files = os.listdir(self.image_dir)

        for img_name in image_files:
            img_path = os.path.join(self.image_dir, img_name)
            mask_path = os.path.join(self.mask_dir, img_name)

            if os.path.exists(mask_path):
                image = Image.open(img_path).convert("RGB")
                mask = Image.open(mask_path).convert("L")
                data_dict[img_name] = (image, mask)

        return data_dict

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_name = self.images[idx]
        image, mask = self.data_dict[img_name]

        if self.is_train:
            # Determine the crop size dynamically
```

```
57              original_width, original_height = image.size
58              crop_width = random.randint(int(original_width * 0.25),
    original_width)
59              crop_height = random.randint(int(original_height * 0.25),
    original_height)
60
61              # Apply random crop
62              i, j, h, w = transforms.RandomCrop.get_params(image,
    output_size=(crop_height, crop_width))
63              image = transforms.functional.crop(image, i, j, h, w)
64              mask = transforms.functional.crop(mask, i, j, h, w)
65
66              # Apply random blur to the image
67              blur_radius = random.uniform(0, self.max_blur_radius)
68              image = image.filter(ImageFilter.GaussianBlur(blur_radius)
    )
69
70              # Apply random horizontal flip with 50/50 chance
71              if random.random() > 0.5:
72                  image = transforms.functional.hflip(image)
73                  mask = transforms.functional.hflip(mask)
74
75          if self.transform:
76              image = self.transform(image)
77              mask = self.transform(mask)
78
79          return image, mask
```

Имплементација 3.3: Класе за читање и аугментацију слика

Током иницијалног тестирања коришћена је имплементација 3.4 за креирање посебних датотека за сваки од скупова за тренирање, валидацију и тестирање. Ова функција ће у будућим итерацијама алгоритма бити неопходна у случају претрениравања модела у режиму са више нити. Физичко раздвајање и умножавање података омогућавају паралелну обраду истих података на више нити, чиме се побољшава ефикасност процеса.

```
1 def create_split_directories(
2     full_dataset,
3     base_dir,
4     train_indices,
5     val_indices,
6     test_indices
```

```
):
    """Create directories for train, val, and test splits and copy images and masks."""
    for split, indices in zip(['train', 'val', 'test'], [train_indices, val_indices, test_indices]):
        split_dir = os.path.join(base_dir, split)
        img_split_dir = os.path.join(split_dir, 'imgs')
        mask_split_dir = os.path.join(split_dir, 'masks')

        os.makedirs(img_split_dir, exist_ok=True)
        os.makedirs(mask_split_dir, exist_ok=True)

        for idx in indices:
            img_name = full_dataset.images[idx]
            img_src = os.path.join(image_dir, img_name)
            mask_src = os.path.join(mask_dir, img_name)

            img_dst = os.path.join(img_split_dir, img_name)
            mask_dst = os.path.join(mask_split_dir, img_name)

            shutil.copy(img_src, img_dst)
            shutil.copy(mask_src, mask_dst)

# Split dataset into train, val, and test sets
train_val_indices, test_indices = train_test_split(range(len(full_dataset)), test_size=0.2, random_state=42)
train_indices, val_indices = train_test_split(train_val_indices, test_size=0.25, random_state=42)  # 0.25 * 0.8 = 0.2

# Create split directories and copy data
base_dir = '../Data/split'
create_split_directories(full_dataset, base_dir, train_indices, val_indices, test_indices)
```

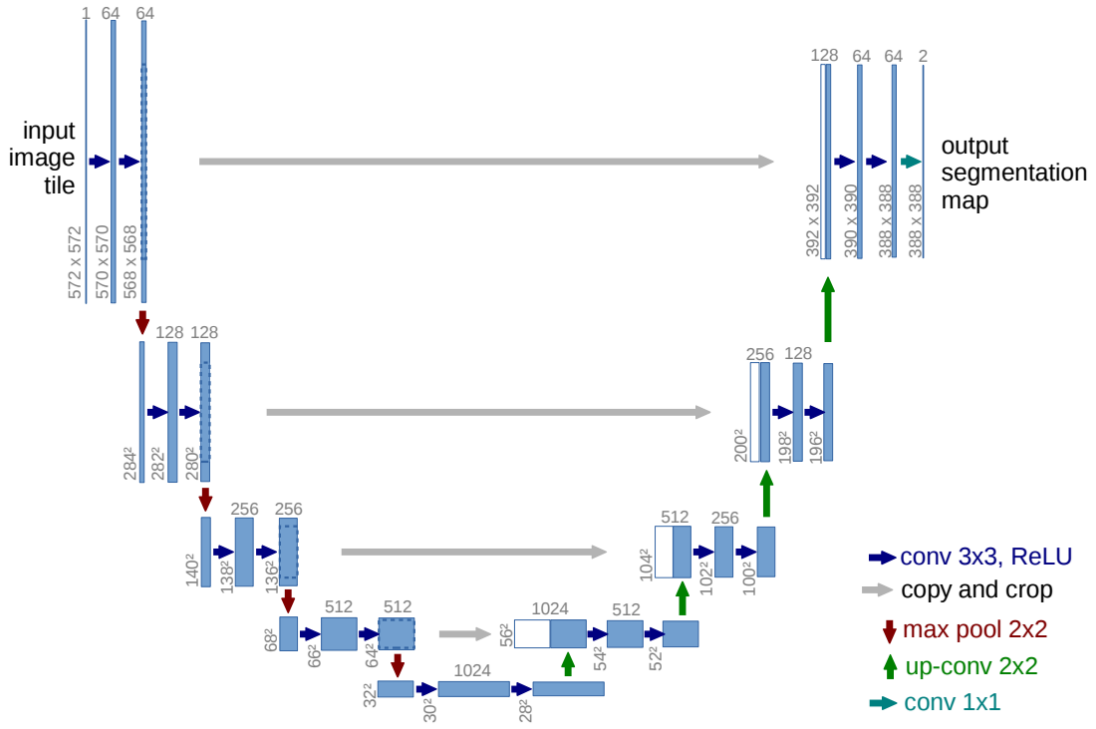
Имплементација 3.4: Имплементација функције за дељење података

3.5 Моделирање конволутивне мреже

Конволутивна мрежа *U-Net*

Као базичну структуру конволутивне мреже користићемо *U-Net*[9] јер је доказано врло ефективан код проблема визуелне сегментације. *U-Net* архи-

тектура личи на латинично слово *U* и састоји се од секвенце конволуција које повећавају резолуцију слике, и истог броја агрегативних слојева који смањују резолуцију слике (види слику 3.3).



Слика 3.3: Визуелна репрезентација *U-Net* архитектуре

Додатно, сваки конволутивни слој је повезан са одговарајућим агрегативним слојем на истом нивоу хијерархије. То омогућава директан пренос информација са конволутивних на агрегативне слојеве истог нивоа (види сиве линије на слици 3.3). Ова структура помаже у очувању детаља слике тако што омогућава прескакање преко других слојева, чиме се задржавају важни атрибути. Ова архитектура је заснована на принципима учења на више скала (енг. *Multi-scale learning*).

Multi-scale learning је ефикасан приступ за тренирање конволутивних неуронских мрежа јер омогућава мрежи да користи информације са различитих нивоа детаља. Коришћењем различитих нивоа детаља и различитих апстракција у репрезентацији података, модел може да научи инваријантне особине објеката које су отпорне на промене у контексту слике. Ова метода помаже

мрежи да генерализује боље на невиђене домене, побољшавајући укупну тачност и робусност класификације[9].

Структура конволутивне мреже

На основу основне структуре *U-Net* мреже имплементираћемо једну њену варијанту.

```
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)

class DownSampleLayer(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownSampleLayer, self).__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)

class UpSampleLayer(nn.Module):
    def __init__(self, in_channels, out_channels, bilinear=True):
        super(UpSampleLayer, self).__init__()
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        else:
```

```
33              self.up = nn.ConvTranspose2d(in_channels // 2, in_channels
    // 2, kernel_size=2, stride=2)
34
35          self.conv = DoubleConv(in_channels, out_channels)
36
37      def forward(self, x1, x2):
38          x1 = self.up(x1)
39          diffY = x2.size()[2] - x1.size()[2]
40          diffX = x2.size()[3] - x1.size()[3]
41          x1 = nn.functional.pad(x1, [diffX // 2, diffX - diffX // 2,
   diffY // 2, diffY - diffY // 2])
42          x = torch.cat([x2, x1], dim=1)
43          return self.conv(x)
44
45  class OutConv(nn.Module):
46      def __init__(self, in_channels, out_channels):
47          super(OutConv, self).__init__()
48          self.conv = nn.Conv2d(in_channels, out_channels, kernel_size
   =1)
49
50      def forward(self, x):
51          return self.conv(x)
52
53  class UNet(nn.Module):
54      def __init__(self, n_channels, n_classes):
55          super(UNet, self).__init__()
56          self.inc = DoubleConv(n_channels, 64)
57          self.down1 = DownSampleLayer(64, 128)
58          self.down2 = DownSampleLayer(128, 256)
59          self.down3 = DownSampleLayer(256, 512)
60          self.down4 = DownSampleLayer(512, 512)
61          self.up1 = UpSampleLayer(1024, 256)
62          self.up2 = UpSampleLayer(512, 128)
63          self.up3 = UpSampleLayer(256, 64)
64          self.up4 = UpSampleLayer(128, 64)
65          self.outc = OutConv(64, n_classes)
66
67      def forward(self, x):
68          x1 = self.inc(x)
69          x2 = self.down1(x1)
70          x3 = self.down2(x2)
71          x4 = self.down3(x3)
```

```
72          x5 = self.down4(x4)
73          x = self.up1(x5, x4)
74          x = self.up2(x, x3)
75          x = self.up3(x, x2)
76          x = self.up4(x, x1)
77          logits = self.outc(x)
78          return logits
```

Имплементација 3.5: Имплементација *U-Net* конволуционе мреже

Функција грешке

Приликом тренирања модела за проблем сегментације, важно је да што више пиксела буде обележено тачном вредношћу циљне променљиве (или тачном лабелом).

Да бисмо подстакли овакав приступ, као функцију грешке уводимо функцију која третира маске као бинарне скупове и рачуна прецизност сегментације тако што рачуна релативну покривеност уније обележених пиксела у оригиналној маски и генерисаној маски.[4]

Прецизност модела у овом случају дефинишемо као однос површине пресека и површине уније између предвиђене и стварне маске сегментације:

$$IoU(Mask, Pred) = \frac{P(Mask) \cap P(Pred)}{P(Mask) \cup P(Pred)}$$

Област пресека је број пиксела који су правилно предвиђени као део објекта, док је област уније укупан број пиксела који су означени као део објекта у било којој од маски (и предвиђеној и стварној).

```
1  def calculate_iou(preds, masks):
2      preds = preds.int()
3      masks = masks.int()
4      intersection = (preds & masks).float().sum((1, 2))
5      union = (preds | masks).float().sum((1, 2))
6      iou = (intersection + 1e-6) / (union + 1e-6)
7      return iou.mean().item()
8
9  def evaluate_model_with_iou_loss(model, dataloader):
10     model.eval()
11     total_iou = 0.0
12     total_samples = 0
13     with torch.no_grad():
```

```
        for inputs, masks in dataloader:
            inputs = inputs.to(device)
            masks = masks.to(device)
            outputs = model(inputs)
            preds = torch.sigmoid(outputs) > 0.5

            iou = calculate_iou(preds, masks)
            total_iou += iou * inputs.size(0)
            total_samples += inputs.size(0)

    avg_iou = total_iou / total_samples
    avg_iou_loss = 1 - avg_iou  # IoU loss
    return avg_iou_loss
```

Имплементација 3.6: Имплементација функције грешке

Угњеждена унакрсна валидација

Пошто је генерисани скуп релативно мали, да бисмо оптимизовали поделу података, као хиперпараметре, имплементирана је угњеждена унакрсна валидација (види имплементацију 3.7).

Пре процеса унакрсне валидације ручно је одвојено *6* слика (уз пропратне маске и дубине) у посебну датотеку. Оне се понашају као фиксни скуп за **тестирање** како не би дошло до цурења података у скуп за тренирање.

Пре него што је имплементирана угњеждена унакрсна валидација, резултати су доста варирали од једне тестиране инстанце до друге, али су се сегментационе маске доста стабилизовале након имплементиране аугментације заједно са унакрсном валидацијом.

```
import logging
from tqdm import tqdm
from sklearn.model_selection import ParameterGrid, KFold
from torch.utils.data import Subset, DataLoader

def train_model(model, criterion, optimizer, dataloader, num_epochs
    =10):
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for inputs, masks in dataloader:
            inputs = inputs.to(device)
```

```
12              masks = masks.to(device)
13
14              optimizer.zero_grad()
15
16              outputs = model(inputs)
17              loss = criterion(outputs, masks)
18              loss.backward()
19              optimizer.step()
20
21              running_loss += loss.item() * inputs.size(0)
22
23          epoch_loss = running_loss / len(dataloader.dataset)
24          logging.info(f'Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss
    :.4f}')
25
26 def cross_val_grid_search(model, criterion, full_dataset, param_grid,
    n_splits=5):
27      best_loss = float('inf')
28      best_params = None
29      param_combinations = list(ParameterGrid(param_grid))
30
31      kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
32
33      for params in tqdm(param_combinations, desc="Hyperparameter
    combinations"):
34          logging.info(f"Testing with parameters: {params}")
35          fold_losses = []
36
37          for train_indices, val_indices in tqdm(kf.split(full_dataset),
     desc="Folds", leave=False):
38              train_subset = Subset(full_dataset, train_indices)
39              val_subset = Subset(full_dataset, val_indices)
40
41              train_dataloader = DataLoader(train_subset, batch_size=
    params['batch_size'], shuffle=True, num_workers=0)
42              val_dataloader = DataLoader(val_subset, batch_size=params[
    'batch_size'], shuffle=False, num_workers=0)
43
44              optimizer = get_optimizer(params['optimizer'], model.
    parameters(), params['lr'], params['weight_decay'])
45
46              # Train the model
```

```
47            train_model(model, criterion, optimizer, train_dataloader,
    num_epochs=params['num_epochs'])
48
49            # Evaluate the model using IoU loss
50            val_loss = evaluate_model_with_iou_loss(model,
    val_dataloader)
51            fold_losses.append(val_loss)
52
53        avg_loss = sum(fold_losses) / len(fold_losses)
54        logging.info(f"Average IoU Loss for parameters {params}: {
    avg_loss:.4f}")
55
56        // Check if the current loss is the best (lowest)
57        if avg_loss < best_loss:
58            best_loss = avg_loss
59            best_params = params
60
61    logging.info(f"Best parameters: {best_params}, Best IoU Loss: {
    best_loss:.4f}")
62    return best_params, best_loss
```

Имплементација 3.7: Функције угњеждене унакрсне валидације

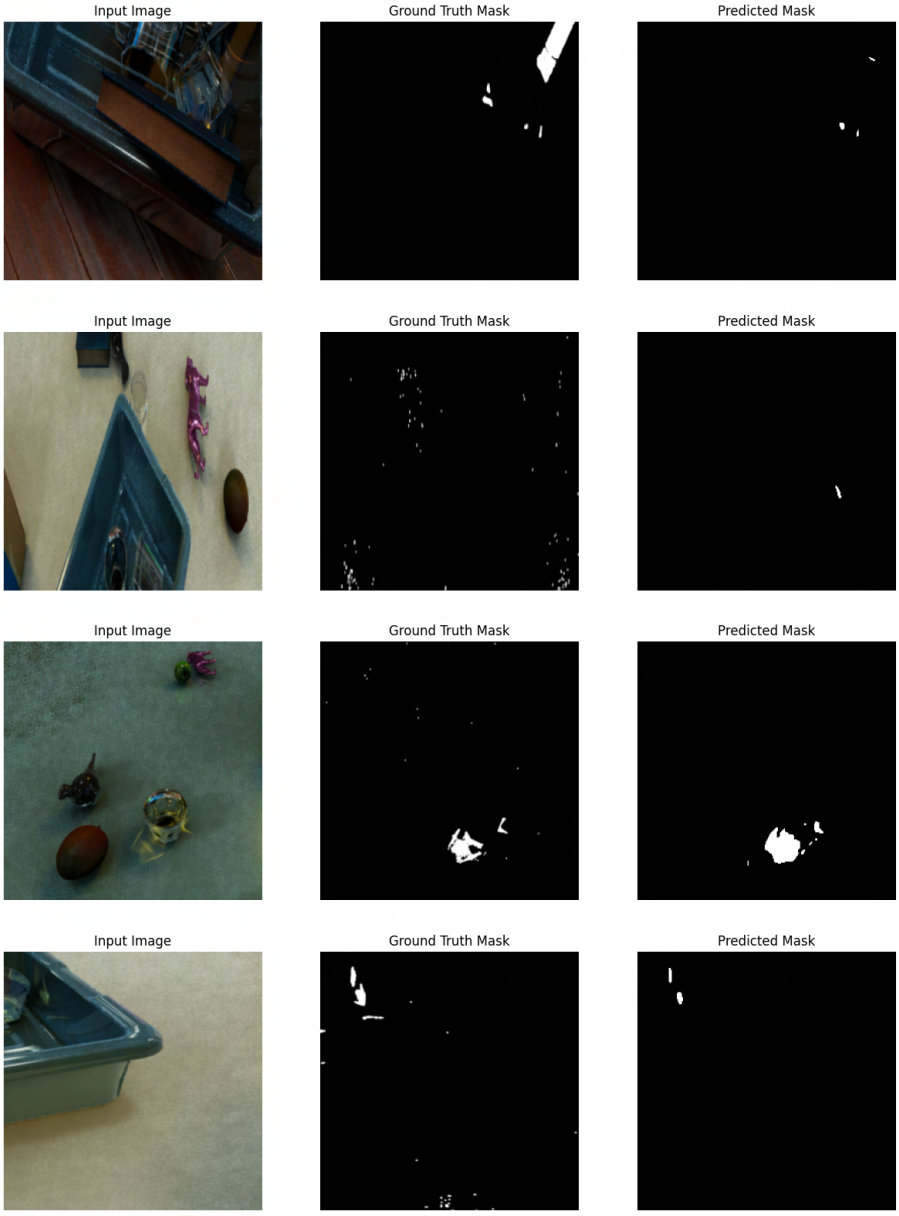
Тренирање

Модел је трениран на следећој конфигурацији:

- *Aorus Nvidia RTX 3080 10GB*
- *Ryzen 9 5900X*
- *Ubuntu 24.04/Windows 11*
- *64GB* радне меморије

Након тренирања добија се модел који производи резултате који се могу видети на слици 3.4.

Модел препознаје веће површине које су недвосмислено комплексне рефлексијеи тако избегава погрешно лабелирање прозирних објеката од стране алгоритма *ClearGrasp*.



Слика 3.4: Резултати примене модела за сегментацију каустика. По колонама су приказане колор слике (лево), очекиване сегментационе маске (средина) и добијене сегментационе маске (десно)

Глава 4

Интеграција у ClearGrasp алгоритам

Манипулација слика

ClearGrasp користи специјални формат слика за тренирање и тестирање који се зове *OpenEXR*. Иако се овај рад се неће бавити поновним тренирањем *ClearGrasp* модела, мора се имплементирати конверзија између генерисаних слика и *OpenEXR* формата због инспекције резултата.

OpenEXR је формат за дигиталне слике високог динамичког опсега развијен од стране *ILM* (en *Industrial Light & Magic*). Овај формат је посебно дизајниран за потребе визуелних ефеката и анимација у филмској индустрији, али је такође користан у разним другим областима где је потребна велика прецизност и флексибилност у обради слике [1].

Интеграција

Процес предикције у ClearGrasp алгоритму функционише тако што се за дату дубину и колор слику предикције генеришу секвенцијално од неколико модела, од којих је нама најбитнији модел за генерисање маски.

Да бисмо нови модел за маскирање интегрисали у *ClearGrasp* алгоритам:

- додали смо класу која дефинише модел за сегментацију каустика коју смо ми истренирали у инфраструктуру неуралних мрежа;
- додали смо класу за предикцију каустика која користи наш модел и класу за препознавање каустика;

```
class InferenceCaustics:
    ...
    def segmentCausticsOnNumpyImage(
        self,
        img,
        dilate_mask=False,
        inverse_mask=False
    ):
        with torch.no_grad():
            // Resize Image and convert to tensor
            det_tf = self.transform.to_deterministic()
            img = det_tf.augment_image(img)
            inputs = transforms.ToTensor()(img)
            inputs = torch.unsqueeze(inputs, 0)
            inputs = inputs.to(self.device)

            outputs = self.model(inputs)
            predictions = torch.max(outputs, 1)[1]
            predictions = predictions.squeeze(0).cpu().numpy()

            mask = np.zeros(predictions.shape, dtype=np.uint8)
            if inverse_mask:
                mask[predictions != 255] = 1
            else:
                mask[predictions == 255] = 1

            if dilate_mask:
                kernel = np.ones((5, 5), np.uint8)
                mask = cv2.dilate(mask, kernel, iterations=1)

        return mask


```

Имплементација 4.1: Класа за предикцију сегментационе маске каустика

- додали смо корак који од оригиналне сегментационе маске одузима нашу сегментациону маску која предвиђа саме каустике, и тиме добијамо сегментациону маску за искључиво транспарентне објекте без њихових одсјаја.

```
def _modify_depth_delete_masks(
    self,
```

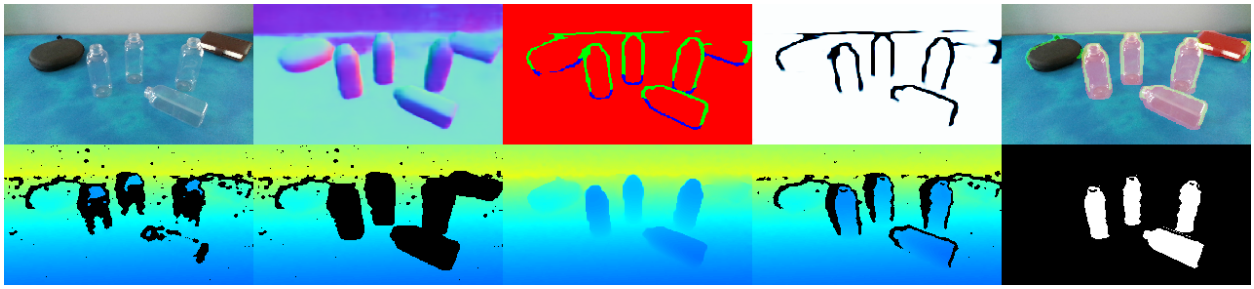
```
3     rgb_image,
4     input_depth,
5     dilate_mask=True):
6     // Run inference to get mask of transparent objects
7     self.mask_predicted = self.inferenceMasks.runOnNumpyImage(
    rgb_image, dilate_mask=True)
8
9     // Run caustics inference to get caustic segmentation mask
10    self.mask_predicted = inference_models.
    subtract_caustics_from_mask(rgb_image,
11
          self.mask_predicted,
12
          self.inferenceCaustics)
13
14    self.mask_predicted = cv2.resize(self.mask_predicted,
15                                     (self.outputImgWidth, self.
    outputImgHeight),
16                                     interpolation=cv2.
    INTER_NEAREST)
17
18    // Mask depth of all transparent objects (detected depth of
    transparent objects is unreliable)
19    modified_input_depth = input_depth.copy()
20    modified_input_depth[self.mask_predicted > 0] = 0.0
21
22    return modified_input_depth
23
```

Имплементација 4.2: Интеграција модела за предикцију каустика у алгоритам *ClearGrasp*

Евалуација новог алгоритма

Покретањем алгоритма пре и после додавања новог корака у израчунавање сегментационих маски добијамо упоредиве резултате. Стари алгоритам је покренут над сликама које су у оквиру *ClearGrasp* базе знања, које немају реалистичну депикцију рефлексија прозирних објеката. Са друге стране, нови алгоритам је покренут са сликама са реалистичним рефлексијама. Важно је напоменути да постоји разлика у улазним дубинским сликама. Пошто приступ истим дубинским камерама као оригинални аутори рада није био

доступан, валидационе слике су ручно обрађене тако да уопште не препознају прозирне објекте. Овакав сценарио је најнеповољнија ситуација за алгоритам, јер мора ни из чега да претпоставља дубину.



Слика 4.1: Излаз старог алгоритма, редом: Улазна колор слика, предвиђене нормале, предвиђене границе, предвиђене тежине граница оклузије, преклоп маски и граница, улазна дубина, очишћена улазна дубина, излазна дубина, стварна дубина, маска важећих транспарентних пиксела (преко којих се рачунају метрике)

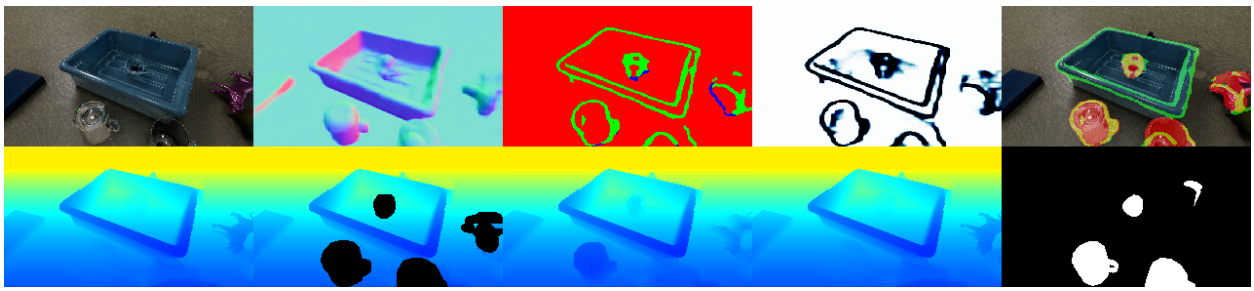
Као што можемо видети на слици 4.2, резултати су врло упоредиви иако нови алгоритам има стартни хендикеп код дубинских слика. Такође можемо приметити да су транспарентни објекти много оштрије приказани на сегментационим маскама.

Гледајући функције грешака, можемо приметити смањење грешака у односу на немодификовани алгоритам када су каустици присутни на сликама (види табелу 4). Са друге стране, ако се алгоритми тестирају на сликама које немају каустике, оригинални алгоритам прави мању грешку у односу на модификовану верзију.

На основу резултате можемо закључити да модификовани алгоритам показује побољшање у присуству каустика, док оригинални алгоритам боље функционише на сликама без каустика.

Табела 4.1: Резултати грешака редом са лева на десно: *ClearGrasp* алгоритам тестиран на подацима без каустика, модификовани *ClearGrasp* тесиран на подацима са каустицима и *ClearGrasp* алгоритам тестиран на подацима са каустицима

Метрика	***ClearGrasp***	**Модификовани *ClearGrasp***	***ClearGrasp* + каустици**
RMSE	0.038	0.049	0.052
REL	0.048	0.056	0.061
MAE	0.027	0.036	0.038



Слика 4.2: Излаз новог алгоритма, редом: улазна колор слика, предвиђене нормале, предвиђене границе, предвиђене тежине граница оклузије, преклоп маски и граница, улазна дубина, очишћена улазна дубина, излазна дубина, стварна дубина (у овом случају непостојећа јер није дата), маска важећих транспарентних пиксела (преко којих се рачунају метрике)

Глава 5

Закључак

Модификација алгоритма *ClearGrasp* има доста простора за побољшање. На основу табеле 4 можемо закључити да додавање детекције каустика повешава грешку алгоритма када нема каустика. Један од начина да се ово избегне је да се уведе динамичко увођење филтера за каустике само у случају када су они присутни на више од $N\%$ пиксела улазне слике.

Алтернативно решење би било тренирати модел за детекцију каустика на далеко већем скупу синтетичких података у комбинацији са реалним фотографијама. Овај приступ би побољшао генерализацију модела, чинећи га способним да прецизније детектује и разликује каустике у различитим ситуацијама. Осим тога, требало би повећати различитост облика 3D модела који представљају транспарентне објекте, као и дефинисати више варијанти прозирних материјала. Ово би омогућило алгоритму да препозна прозирне објекте независно од њиховог облика и количине шума на слици.

Коначна опсервација је везана за алгоритам пре и после модификације. Током израде овог рада пун процес израчунавања резултата од полазних слика трајао је, реда величине, 2-3 минута. За брже извршавање алгоритма могуће је покретати алгоритам на више нити паралелно, тако да би за његово идеално убрзање било неопходно да се убрза сваки индивидуални искоришћени модел.

Парцијално убрзање бисмо могли да постигнемо паралелизацијом, додељивањем више процесорских нити сваком моделу. Свакој нити би се доделио конкретан део слике над којим би вршила израчунавања. Сличне методе исцртавања се примењују у модерним имплементацијама исцртавања компјутерске графике у реалном времену, где свака нит има алоцирани део екрана

за чије исртавање је надлежна, доприносећи брзом исцртавању слика.

Уз сва ова побољшања, модификовани алгоритам би могао достићи нове нивое ефикасности и прецизности у детекцији прозирних објеката.

Библиографија

[1] Openexr documentation, 2024. Accessed: 2024-07-26.

[2] MM El-Gayar, H Soliman, et al. A comparative study of image low level feature extraction algorithms. *Egyptian Informatics Journal*, 14(2):175–181, 2013.

[3] Dragan Lazarević, Milan Mišić, and Bogdan Ćirković. Postojeće tehnike za segmentaciju slike. *Festival kvaliteta*, 2014.

[4] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[5] Mehdi Mousavi and Rolando Estrada. Supercaustics: Real-time, open-source simulation of transparent objects for deep learning applications, 2021.

[6] Oliver Ozvačić. *Procjena dubine podudaranjem stereo slika.* PhD thesis, Fakultet elektrotehnike i računarstva, 2024.

[7] Charles Poynton. *Digital Video and HD: Algorithms and Interfaces.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition, 2012.

[8] Yiming Qian, Minglun Gong, and Yee Hong Yang. 3d reconstruction of transparent objects with position-normal consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4369–4377, 2016.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[10] Shreeyak Sajjan, Matthew Moore, Mike Pan, Ganesh Nagaraja, Johnny Lee, Andy Zeng, and Shuran Song. Clear grasp: 3d shape estimation of transparent objects for manipulation. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 3634–3642. IEEE, 2020.

[11] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[12] Bojian Wu, Yang Zhou, Yiming Qian, Minglun Gong, and Hui Huang. Full 3d reconstruction of transparent objects. *arXiv preprint arXiv:1805.03482*, 2018.

Биографија аутора

Владимир Батоћанин рођен је 3. августа 1997. у Београду. Након завршетка Тринаесте београдске гимназије у Београду 2016. године, уписао је Математички факултет Универзитета у Београду на модулу Информатика. Дипломирао је на Математичком факултету Универзитета у Београду 2021. године. Тренутно је студент мастер програма на Математичком факултету Универзитета у Београду.