Towards Data-centric Artificial Intelligence with Flexible Photorealistic Simulations

Mehdi Mousavi
Georgia State University

Follow this and additional works at: https://scholarworks.gsu.edu/cs_diss

Recommended Citation
doi: https://doi.org/10.57709/28830665

This Dissertation is brought to you for free and open access by the Department of Computer Science at ScholarWorks @ Georgia State University. It has been accepted for inclusion in Computer Science Dissertations by an authorized administrator of ScholarWorks @ Georgia State University. For more information, please contact scholarworks@gsu.edu.
Machine Learning requires data. Without the availability of large, high-quality datasets, the success of deep learning in recent years would not have been possible. Data is the fundamental building block in developing AI pipelines. However, due to the limitations in measurement tools, lack of control and immutability of real-life datasets, the general approach to developing machine learning solutions has evolved to be model-centric. This Dissertation explores the possibility of Data-centric AI by looking at the development of a novel technology — flexible photorealistic simulations — that can generate labeled datasets for use in lieu of real data in various fields of deep-learning accelerated computer vision. In each chapter of this work, we’ll follow a major phase shift that represents a forward
step in the applications of this field. From proof of concept, Improving existing methods, Applications on hard tasks, to achieving state-of-the-art performance.

TOWARDS DATA-CENTRIC ARTIFICIAL INTELLIGENCE WITH
FLEXIBLE PHOTOREALISTIC SIMULATIONS

by

MEHDI MOUSAVID

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy
in the College of Arts and Sciences
Georgia State University
2022
TOWARDS DATA-CENTRIC ARTIFICIAL INTELLIGENCE WITH
FLEXIBLE PHOTOREALISTIC SIMULATIONS

by

MEHDI MOUSAVI

Committee Chairs : Rolando J. Estrada
                Ashwin Ashok

Committee: Robert Harrison
           Zhisheng Yan
           Ying Zhu

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
March 2022
DEDICATION

This dissertation is dedicated to my Parents, in acknowledgement of their unconditional love and emotional support through this long and trying journey. I honor all my family, my incredible siblings, my nephews and nieces for their kindness, patience and support.
I would like to thank my advisors for their guidance and sponsorship. Their valuable presence and constant support, immensely helped me in coping with pressures of undertaking a terminal degree in computer science. I would like to thank Dr. Estrada for his patience and his ability to clarify the complicated path of research in our many conversations. I would like to thank Dr. Ashok for his incredible support during the toughest of times, his wisdom, and for providing valuable teamwork opportunities. I would like to thank Dr. Harrison for his contagious enthusiasm and supporting my ambitions to conduct research in the synthetic simulations field. I would like to thank Dr. Yan and Dr. Zhu for agreeing to be in my committee, for helping me see their unique perspectives and for actively contributing to the betterment of all aspects of this dissertation.

I would like to thank my Parents for their unconditional support throughout the years, and for their incredible bravery in the face of the unknown, and sacrifice in enduring the long-term distance between us, and for sending me out to the United States in pursuit of knowledge. I would like to thank my Sister, If were not for her determination, encouragement and interest in my future, I would have never attended graduate school. I thank her and my parents for believing in my abilities even during the times that even I did not.

Lastly, I would like to thank my friends, whose presence and support kept me going through very hard times. For our many conversations that lasted long into the night, for your incredible perspectives on handling the pressures of graduate school. For our laughter, joy and happiness in times when I needed them most.
# TABLE OF CONTENTS

[ACKNOWLEDGMENTS](#) ........................................... v

[List of Tables] .................................................. xi

[List of Figures] .................................................. xii

Chapter 1  INTRODUCTION ........................................ 1

1.1 SYNTHETIC DATA FOR MACHINE LEARNING .................... 1

1.1.1 Data-Centric vs Model-Centric AI .......................... 2

1.1.2 Using Synthetic Images as Training Data .................. 4

1.1.3 Synthesizing Images: importance and the role of Photorealism 4

   Synthesizing ground-truth: getting the most accurate readings 5

1.1.4 Types of Image data for Deep Learning .................... 7

1.1.5 Data Ablation: Enabling A Flexible Approach to Simulations . 9

1.1.6 Phases of Synthetic Data Adoption ........................ 10

   Phase 1, Proof of Concept ................................. 10

   Phase 2, Improving existing methods ...................... 11

   Phase 3, Applications on difficult tasks .................. 11
Chapter 2 PHASE ONE: PROOF OF CONCEPT

2.1 AI PLAYGROUND: UNREAL ENGINE BASED DATA ABLATION TOOL FOR DEEP LEARNING

2.1.1 Generating Ground-truth annotations

- Depth estimation: .......................... 16
- Surface normals: .......................... 16
- Semantic segmentation: .................... 17
- Data ablation controls ......................... 18

2.1.2 Experiments and Results .................. 19

- Experimental Setup ....................... 21
- Monocular depth estimation experiments ............. 23
- Surface normal estimation experiments ............... 25
- Semantic segmentation experiments ................. 26

2.1.3 Data Ablation diagnostic analysis ........... 27

2.1.4 Contributions .......................... 30

Chapter 3 PHASE TWO: IMPROVING EXISTING METHODS ............ 32

3.1 SUPERCAUSTICS: REAL-TIME, OPEN-SOURCE SIMULATION OF TRANSPARENT OBJECTS FOR DEEP LEARNING APPLICATIONS .......... 32

3.1.1 Why a flexible system is needed? ......... 37

3.1.2 SuperCaustics: a Flexible Simulation ........ 40
3.1.3 Evaluation .................................................. 44
Experimental Setup .............................................. 45
Hardware .......................................................... 45
Image acquisition .................................................. 45
Deep Neural Networks .......................................... 46
Results & Discussion ............................................ 46
3.1.4 Contributions ............................................... 51

Chapter 4 PHASE THREE: APPLICATIONS ON DIFFICULT TASKS 53

4.1 OPENWATERS: PHOTOREALISTIC SIMULATIONS FOR UNDERWATER COMPUTER VISION 53
4.1.1 Related Work .............................................. 55
4.1.2 Evaluation .................................................. 60
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experimental Setup</td>
<td>61</td>
</tr>
<tr>
<td>Simulation Instance</td>
<td>61</td>
</tr>
<tr>
<td>4.1.3 Results &amp; Discussion</td>
<td>63</td>
</tr>
<tr>
<td>4.1.4 Contributions</td>
<td>70</td>
</tr>
<tr>
<td>Chapter 5 PHASE FOUR: ACHIEVING STATE OF THE ART PERFORMANCE</td>
<td>71</td>
</tr>
<tr>
<td>5.1 IDEHAZE: SUPERVISED UNDERWATER IMAGE ENHANCEMENT &amp; DEHAZING</td>
<td>71</td>
</tr>
<tr>
<td>V IA PHYSICALLY ACCURATE PHOTOREALISTIC SIMULATIONS</td>
<td>71</td>
</tr>
<tr>
<td>5.1.1 Related Work</td>
<td>74</td>
</tr>
<tr>
<td>5.1.2 Methods</td>
<td>76</td>
</tr>
<tr>
<td>Simulation of Underwater Environments</td>
<td>77</td>
</tr>
<tr>
<td>Dehazing vs Color Transfer</td>
<td>79</td>
</tr>
<tr>
<td>5.1.3 Experiments</td>
<td>80</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>80</td>
</tr>
<tr>
<td>Data acquisition</td>
<td>80</td>
</tr>
<tr>
<td>Experimental Setup</td>
<td>81</td>
</tr>
<tr>
<td>Metrics</td>
<td>81</td>
</tr>
<tr>
<td>Datasets</td>
<td>82</td>
</tr>
<tr>
<td>5.1.4 Results</td>
<td>83</td>
</tr>
<tr>
<td>Discussion</td>
<td>90</td>
</tr>
<tr>
<td>Future Work</td>
<td>92</td>
</tr>
</tbody>
</table>
5.1.5 Contributions ................................................. 92

Chapter 6 CONCLUSIONS ........................................ 93

REFERENCES ......................................................... 95
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>AI Playground: Experimentation scenarios</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>AI Playground: Data Ablation results for Depth Estimation</td>
<td>22</td>
</tr>
<tr>
<td>2.3</td>
<td>AI Playground: Results on real datasets</td>
<td>23</td>
</tr>
<tr>
<td>2.4</td>
<td>AI Playground: Data Ablation results for Surface Normal Estimation</td>
<td>25</td>
</tr>
<tr>
<td>2.5</td>
<td>AI Playground: Data Ablation results for Semantic Segmentation</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>SuperCaustics: results on novel scenarios</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td>SuperCaustics: caustic segmentation quantitative results</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>OpenWaters: Data Ablation diagnostics on same domain</td>
<td>64</td>
</tr>
<tr>
<td>4.2</td>
<td>OpenWaters: Data Ablation Diagnostic experiments</td>
<td>65</td>
</tr>
<tr>
<td>4.3</td>
<td>OpenWaters: semi-quantitative analysis</td>
<td>65</td>
</tr>
<tr>
<td>5.1</td>
<td>iDehaze: Performance vs. Leading Learning based methods</td>
<td>87</td>
</tr>
<tr>
<td>5.2</td>
<td>iDehaze: Performance vs. Leading GAN-Based methods</td>
<td>88</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Imperfections in real-data measurements</td>
<td>3</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Taxonomy of different sources of deep learning data</td>
<td>7</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>Image examples for Data Ablation</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>AI Playground: System architecture</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>AI Playground: perspective projection vs. Orthographic projection</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>AI Playground: Virtual Environments</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>AI Playground: Fidelity Comparison</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>AI Playground: Sample results</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>AI Playground: Data Ablation on Semantic Segmentation example</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>SuperCaustics: Example rendered images</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>SuperCaustics: system capabilities</td>
<td>36</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>SuperCaustics: system overview</td>
<td>37</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>SuperCaustics: controlling simulation variables</td>
<td>41</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>SuperCaustics: world-space normals vs camera-space normals</td>
<td>43</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>SuperCaustics: network output results on cleargrasp-novel real image dataset</td>
<td>47</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>SuperCaustics: caustics segmentation network output example</td>
<td>50</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>OpenWaters: Simulation architecture</td>
<td>58</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>OpenWaters: Ground-Truth Annotations</td>
<td>59</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>OpenWaters: physical experimentation setup</td>
<td>60</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>4.4</td>
<td>OpenWaters: Neural Network architecture</td>
<td>62</td>
</tr>
<tr>
<td>4.5</td>
<td>OpenWaters: Qualitative Analysis for real images</td>
<td>63</td>
</tr>
<tr>
<td>4.6</td>
<td>OpenWaters: Comparison between ZED camera and OpenWaters prediction</td>
<td>66</td>
</tr>
<tr>
<td>4.7</td>
<td>OpenWaters: Synthetic Data Tests</td>
<td>67</td>
</tr>
<tr>
<td>4.8</td>
<td>OpenWaters: Data Ablation experiment examples</td>
<td>68</td>
</tr>
<tr>
<td>5.1</td>
<td>iDehaze: Results on UIEB Dataset</td>
<td>72</td>
</tr>
<tr>
<td>5.2</td>
<td>iDehaze: Proposed Pipeline</td>
<td>77</td>
</tr>
<tr>
<td>5.3</td>
<td>iDehaze: Example synthetic data</td>
<td>79</td>
</tr>
<tr>
<td>5.4</td>
<td>iDehaze: Qualitative Comparisons</td>
<td>85</td>
</tr>
<tr>
<td>5.5</td>
<td>iDehaze: Qualitative Comparisons with Leading methods</td>
<td>86</td>
</tr>
<tr>
<td>5.6</td>
<td>iDehaze: Qualitative analysis on the EUVP dataset</td>
<td>89</td>
</tr>
<tr>
<td>5.7</td>
<td>iDehaze: Qualitative analysis on the UFO-120 dataset</td>
<td>90</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 SYNTHETIC DATA FOR MACHINE LEARNING

Machine learning requires data. Being the fundamental building block for solving AI problems, the vast availability of data plays an integral part in the immense surge of AI and computer vision, and the success of deep learning in recent years would not have been possible without large, high-quality datasets [1]. Deep neural networks have thousands or even millions of parameters, which require vast numbers of training examples to tune.

For supervised, deep-learning-accelerated computer vision, there are tasks that require pixel-wise labels for every image example in the training dataset. Such tasks like monocular depth estimation, or robotic grasping of transparent objects are reliant upon vast amounts of accurate sensor data as ground-truth. Slightest inaccuracy in the process of acquiring labels can result in errors cascading down the pipeline, which causes further inaccuracies in the deep learning inference.

Modern computer graphics can achieve near-photorealism in real time. Game engines like Unreal Engine 5 and Unity are adopting novel real-time rendering methods that can produce life-like images, capable of creating experiences almost indistinguishable from reality. With the rapid rise of deep learning, the need for more data — specifically labelled image data — has been evermore pressing. Since many computer vision systems and deep learning models
require more and more data to tune on, photorealistic simulations can be a replacement for
gathering real images in tasks where availability of real data is scarce, or where its difficult,
expensive or laborious to acquire labelled data.

1.1.1 Data-Centric vs Model-Centric AI

Role of data quality is often neglected in AI and ML Projects. As low-quality, generic and
often immutable datasets are used in custom-designed neural network and algorithm
pipelines. The vast majority of effort is concentrated on making sure the model performs
well on a generic dataset. Thus, the current state-of-affairs in machine learning projects can
be dubbed as model-centric. In a model-centric AI paradigm, training the model is the most
expensive part of the project (and mainly its primary focus), and the data is immutable. As
a result, the only option is either going back to edit the model, or use another set of training
data to tune the system further. If there are any errors or shortcomings in the training data
(As is it the case with many real-world datasets, see Figure 1.1), Its almost impossible to re-
capture the data-set. Producing a high-quality dataset of real data is very challenging. First,
one has to acquire the raw data, which must either be labeled manually—a slow, subjective
process that may require significant expertise—or with expensive, specialized equipment.
Finally, errors can occur in both the acquisition and labeling phases. More subtly, real data
is very difficult to control before acquisition and nearly impossible to change afterwards.
For instance, once an image has been taken, one cannot change its illumination from day to
night or replace one object for another$^1$. To change the color of a couch in an image, one

$^1$Photo-manipulation techniques can be used to alter images, but their effects are either non-specific (e.g.,
reducing brightness) or introduce unwanted artifacts. They also require significant human effort.
would need to swap out two otherwise identical couches and place them in the same, exact location. Aside from its difficulty, this approach is simply not feasible for natural scenes or crowd-sourced data.

![Image and Ground-truth examples from DIODE Dataset](image.png)

*Figure 1.1: Image and Ground-truth examples from DIODE Dataset [2], black pixels in the images represent measurement errors in the ground-truth.*

By definition, A Data-Centric approach is a highly iterative process where there are no constant parts in the training procedure. For example, if there is a specific problem to solve, It’s possible to synthesize a dataset of image examples and perfectly correct ground-truth annotations, train a deep-learning model on the data and assess its performance on various characteristics of the target domain using Data Ablation. When the short-comings are discovered, on top of the regular approach to editing the model, its possible to synthesize a new dataset to cover the gaps in training. Compared to real data, Flexible synthetic data
is very cheap to acquire, does not suffer from human error or labeling issues and can be meaningfully customized or changed before or after acquisition.

1.1.2 Using Synthetic Images as Training Data

1.1.3 Synthesizing Images: importance and the role of Photorealism

The question of photorealism and the influence of computational approximation on the performance of a model on real-world data has been the subject of numerous studies [3–5]. Specifically, photorealism is at the heart of [6], where researchers attempt to definitively answer when and where synthetic data can be most reliable. While photorealism is essential to the usefulness of a synthetic dataset, there is a point of diminishing returns where the time required to render a single frame becomes increasingly less justifiable when compared to the performance gain from the increased fidelity, in other words, the rendering time required to improve performance can be better spent elsewhere. This is further confirmed in following studies [7–9], that show models that use more photo-realistic data generally perform better than those trained on lower-fidelity data, however, photorealism is proven to be not the only factor when evaluating a data-set for deep learning purposes; as lighting, structure and variety can also play an important role in how a synthetic dataset performs in real-life scenarios.

Generally, in order for a synthetic dataset to be usable, there has to be at least some level of photorealism that brings the images into the realm of real-life (i.e. no stylization, or use of obsolete rendering techniques), but extreme photorealism is generally not required [6].
Finally, there has been significant advances in modern computer graphics in the past few years, where near-photorealistic images can be synthesized in real-time [10,11]. As such, synthetic data has become a viable source of training data in situations where acquiring or labeling real data is difficult. Synthetic data has been proven to be useful in complex computer vision tasks, such as depth estimation [7,8,12], surface normal estimation [8], robotic grasping [8,13] and object segmentation [9] by multiple independent researchers [4–8].

**Synthesizing ground-truth: getting the most accurate readings**  At their core, photorealistic simulations are a mathematical representation of geometric shapes in 3D space. Much like the real world, simulations are 3D environments where entities of different abstract classes can interact with each other. Light, Cameras and World Objects (Their position, rotation, scale, geometry and materials) all follow rules defined by the programmer, which can follow the same rules as the real-world and be modeled to be physically accurate. Examples of such designs are Euclidean game engines, such as Unreal Engine [11] and rendering algorithms that model light propagation like the real-world, such as ray-tracing.

Because of the nature of these simulations, Ground-truth data in many cases is of a mathematical nature, thus can be accurately calculated. Take the example of depth estimation, which represent the distance between every pixel and the camera; Before an image can form in a virtual camera, the distance between every point in the environment and every pixel on the sensor of the virtual camera can be accurately calculated. This operation can be done in constant time on a parallel processor such as a Graphics Processing Unit (GPU). Similarly, there are many other tasks where the ground-truth can be mathematically calculated using a single or a combination of calculated ground-truth passes, such as Image
segmentation, surface normal estimation in camera-space and world-space and object outlines (trimaps). Mathematically calculated ground-truth annotations are the most reliable and accurate available annotations for any image.

However, there are other tasks where ground-truth can be a lot more subjective. For example, detecting emotions of a randomly generated digital human cannot be easily calculated by a simple mathematical equation without further context and information. In such cases, there is a need for a human operator (likely the simulation engineer or the 3D artist) to add meta-data to the different components used in the simulation to allow for a rule-based synthesis of ground-truth annotations. For example, if our approach to synthesizing a face is choosing from a random set of eyes, mouths and noses, we could label the general expression of the resulting face depending on the labels of its chosen components.

It’s important to note that ground-truth synthesized in rule-based approaches can be fundamentally less accurate than mathematically calculated ground-truth. Since there is a human operator involved in the creation of the rule-set, the possibility for human-error in devising a correct rule-set is significant, and the simulation engineer has to be extra cautious about the devised rules in such cases.
1.1.4 Types of Image data for Deep Learning

![Diagram of data accuracy and flexibility]

*Figure 1.2: Taxonomy of different available sources of data for deep learning applications*

For the purposes of this document, we assess the quality of a dataset based on its accuracy and flexibility, and categorize traditional datasets on a accuracy/flexibility scale. Traditionally, crowd-sourced data is the moderately flexible, as post-processing and photo-manipulation techniques can be used on them to alter the images. However, errors in crowd-sourced data labels are not uncommon. Conversely, expertly labelled data is often very accurate, but any changes in the data may invalidate the labels, making them very rigid. Arguably, the least accurate type of data is often manually gathered datasets, where the probability of measurement errors can be high due to inexpensive, inaccurate measuring tools or human error. While manually gathered data might not be as accurate as expertly labelled datasets, it can be flexible enough to fit the specific properties of the problem its intended for, which makes them very useful in novel tasks. However, as mentioned before not all tasks have reliable real data available; either because of the immense difficulty of gathering image data (e.g. underwater images) or difficulty in obtaining accurate labels (e.g.
depth photos of transparent objects). In such cases, we see the merits in using synthetic data, in which the image and ground-truth are synthesised from scratch using computer graphics. This can solve the problem of image availability and accuracy, making synthetic datasets significantly more accurate than even expertly labelled data due to the fact that ground-truth annotations can be generated in pixel-wise level with no calculation errors. However, there is still a difference in flexibility between synthetic datasets (See Figure 1.2). Arguably, flexibility is an important property of synthetic data, and one of its main strengths that highlights its advantage over real data. By definition, A flexible dataset must allow for rapid iteration in how it is set up, rendered and shaded. More importantly, it should allow for fine control over most (if-not-all) variables in the simulation. In real-time systems, the data can be adjusted for the weaknesses in the performance of a model, recaptured and fed to the model for training to patch out any weaknesses or sensitivities. A flexible simulation also means the final dataset can be smaller and more dense, reducing training costs and computational time needed to train the system.

Another advantage of synthetic data is the ability to perform domain targeting. For very specific use-cases — such as robotics — where we know the details of the final product, we can tailor the properties of the simulation to match the real-world parameters the system will be encountering, thus tuning training data more towards the target domain. For instance, imagine if we were tuning the navigation system for a household robot that is equipped with a specific image sensor and optics. In capturing the training (or fine-tuning) data, we can simulate the exact camera optics and sensor size when acquiring the image data to come as close as possible to the final images in the real domain.
1.1.5 Data Ablation: Enabling A Flexible Approach to Simulations

Data ablation is the study of change in the performance of a neural network, given isolated changes in its input data. For example, if we trained a network that labels a couch in a well-lit room and we wanted to test how ambient lighting affects its performance, we would have to keep our camera, the couch and every other object in the frame exactly the same, and assume our camera performs exactly the same. Apart from being extremely time-consuming, there will likely be presence of hard-to-control variables like noise in the camera or presence of optical artifacts. In the real world, Data ablation experiments can go from being easy but laborious to extremely difficult and virtually impossible. For example, If we wanted to do the same object segmentation experiment on an object underwater, controlling the patterns of caustics would be prohibitively difficult. Conversely, Data Ablation is possible in synthetic simulations where there is real-time access to rendering parameters. One can freeze the patterns of a water wave, or change the illumination of a room to take two identical images with isolated changes.

Applying the scientific method on specific features of data is the main strength of Data Ablation. When the only difference between two images is an isolated feature, conclusions can be confidently drawn about difference in performance (See Figure 1.3) given the change in the input data. If a segmentation model completely breaks down when encountering a new couch texture, or a new lighting scenario, it would be safe to say that its very sensitive to the texture of that couch, or change in the lighting of the room.
Figure 1.3: Data Ablation: Changing the texture of a couch inside a photorealistic simulation.

If a model were to perform poorly when a couch texture is changed, we could draw a conclusion about its sensitivities when encountering new data, and adjust training data accordingly.

1.1.6 Phases of Synthetic Data Adoption

Phase 1, Proof of Concept In the conception of every new technology, there are phases in which the possibilities and bounds of said technology will be explored. This phase is often the very starting point in scholarly research and in industry implementations. For synthetic data, Proof of Concept phase started around 2015. Studies around that time were focusing more on the applicability of synthetic data for training or diagnostics of neural networks, its limitations and novel implementations. There were studies that explored synthetic data in the context of pedestrian detection, depth estimation and pixel-wise labelling. During this time, it was demonstrated that synthetic data can in-fact be used to train artificial neural networks for real-world tasks, the importance of photorealism was demonstrated, and use of the pixel-perfect ground-truth annotations to run accurate tests on a convolutional neural network’s performance were explored [3,6,14]. This phase continued until late 2019 [7],
where the benefits of virtual worlds and photorealism were no longer debated; and the field was more focused on more applications aimed towards improving existing methods [8].

**Phase 2, Improving existing methods** When the potential of a new technology is demonstrated in its proof of concept phase, momentum is directs the field to its next phase, where scholarly research is focused on leveraging the new technology’s strengths to improve upon current standards and state-of-the-art. In this phase, fields that were driven to stagnation due to sheer difficulty or a lack of reliable data suddenly see a surge of activity. This was the case in Phase 2 of Synthetic data, where it was applied in a wide variety of domains in both academia and in industry research. This phase started in early 2019, and synthetic data was applied on a wide variety of applications, including monocular depth estimation, surface normals estimation, robotic grasping for transparent objects, autonomous vehicles, autonomous warehouse and factory equipment, various fields of reinforcement learning, and even mobile phone notifications [7,8,15–17].

**Phase 3, Applications on difficult tasks** The use of a new technology in various different fields and applications drives its movement to Phase 3 of adoption, where a few of the problems turn out to be extremely difficult to solve using the traditional methods, and experience immense improvements by the application of the new technology. For Synthetic data, one such task is underwater computer vision. Deep learning enables the ability to solve a wide variety of visual tasks due to the availability of specialized, standardized, massive and high quality datasets. However, that was generally not the case in underwater computer vision, because there were many challenges in acquiring image and ground-truth pairs from
underwater environments. Problems like difficulty in accessing accurate equipment, logistics of manually gathering underwater data and lack of standardization in the available datasets make applying deep learning to this field quite challenging. However, synthetic data does not require physically diving in order to gather high-quality images. As such, synthetic data has been used in the past to fuel deep learning approaches in the underwater domain [18–23].

Phase 4, Achieving State-of-the-art performance After a certain period of research, there’s generally a phase where a matured technology starts achieving state-of-the-art results, beating traditional methods in performance, or efficiency. For deep learning accelerated computer vision via synthetic data, that’s generally because of the accuracy and availability of high-quality datasets. Currently, there are still tasks which training with real data will result in better performance, however, these are generally tasks in which generating a 3D simulation is often more difficult than gathering real data. For other types of problems, where gathering labelled data is challenging, researchers are beginning to enter the final phase of Adoption, and achieve state-of-the-art results using majority synthetic data for training. As an example, fields such as Robotics, Warehouse autonomy, Underwater image enhancement and dehazing, or Autonomous vehicles are seeing extended use of simulations as the primary source of training data and ground-truth annotations [8,9,23,24].
In this dissertation, we follow the development and adoption of photorealistic simulations over the aforementioned four phases. In phase one, we’ll explore the broad question of using photorealistic simulations in lieu of real data for training and diagnosis of deep learning pipelines. We explore the applicability of synthetic data for indoor vision tasks, and attempt to answer questions about learned features and robustness to changes for various deep learning architectures. Secondly, we introduce the concept of Data Ablation, a robust procedure to run accurate diagnostics on deep learning systems.

In phase two, we explore the use of photorealistic simulations for applying deep learning in the field of transparent object detection. We introduce massive improvements to the state-of-the-art method by devising a more efficient training procedure that reduces the amount of required training data by 90%, and substantially reduces the computational costs. Achieved by creating a procedural scene generation system with a higher degree of domain representation, and a denser, more high-quality dataset.

In phase three, we explore the feasibility of training underwater computer vision pipelines for depth estimation entirely in simulations, and demonstrate the usefulness of simulations in difficult, hard-to-control environments. Secondly, we show that in case of 1-to-1 recreations where simulations are modelled after their real-life counterparts, simulations can be used to make ground-truth annotations for real images that lack labels.

In phase four, we introduce a novel deep learning pipeline for underwater image enhancement and dehazing, trained on majority synthetic data generated in physics-based, photorealistic 3D simulations. We then demonstrate the state-of-the-art performance of the deep learning pipeline by performing quantitative and qualitative analysis on various underwater image datasets.
2 PHASE ONE: PROOF OF CONCEPT

2.1 AI PLAYGROUND: UNREAL ENGINE BASED DATA ABLCATION TOOL FOR DEEP LEARNING

![Diagram of AI Playground components]

**Figure 2.1: AI Playground:** AIP two main modules: the AIP Core within UE4 and Probe, a Python module that communicates with the Core. Probe receives instructions generated by the Command module, and saves its state in its own dedicated memory. This allows changing settings inside the engine while AIP is running. Manually changing components is also possible via the GUI.

AIPlayground is a UE4-based tool for data ablation studies in computer vision. As illustrated in Fig. 2.1, AI system has four components: (1) high-resolution 3D environments; (2) multiple ground-truth annotations; (3) data ablation controls; and (4) a user-friendly,
graphical interface. We use Blueprint, Unreal Engine’s visual scripting language, for the ground-truth annotations and data ablation controls. We use a separate Python interaction module—Probe—for data collection. The key contributions of AIP are: 1. Portable data ablation tools for UE4 Environments, 2. Multimodal, accurate ground-truth generation for visual tasks, and 3. Python and UE4 interplay for data acquisition.

![Image](image_url)

**Figure 2.2: Depth estimation: AIP uses perspective projection (first row), which is more accurate than orthographic projection (second row). The third column uses color banding to highlight the differences between these two approaches. The bottom rows show examples from the DIODE and NYUv2 datasets. Note the lack of artifacts in the virtual ground truth.**
2.1.1 Generating Ground-truth annotations

One of the main advantages of virtual environments is that obtaining ground-truth annotations is trivial relative to real-world environments. Specifically, we use Unreal Blueprint (an internal scripting language) to calculate the ground-truth properties listed below. AIP includes Blueprint scripts for estimating depth, surface normals, and object classes, and can be readily extended by adding additional scripts. We use post-processing shaders, called materials in UE4, to overlay these properties over the image, enabling pixel-perfect alignment between the data and the ground-truth labels (see Fig. 2.3 for examples).

**Depth estimation:** We calculate the normalized distance between each pixel that belongs to a specific object and the camera. We set the real-life range of depth to 10 meters, which covers the entire environment and does not clip between any corners of the room. We define the depth using *perspective projection* relative to the viewer’s POV, which is significantly more accurate than orthographic methods. In perspective depth, each light ray is traced to the exact pixel from the object its coming from; in orthographic depth, on the other hand, light-rays are assumed to be coming from *infinity* (see Fig. 2.2). In real-world datasets, e.g., NYUv2 [25] and DIODE [2], depth is registered based on orthographic projection because of physical limitations in the sensor.

**Surface normals:** We estimate the normal vector w.r.t to each 3D surface, then color each pixel to indicate the vector’s direction. We use 6 main colors to show 6 axis of direction (positive and negative xyz, as shown in Fig. 2.3).
Figure 2.3: Virtual environments: Sample screenshots from our annotated virtual environments. From left to right: depth, surface normals, and semantic labels.

Semantic segmentation: In UE4, it is easy to map visible pixels to their corresponding 3D objects. Our Blueprint script uses this mapping to overlay pixel-perfect semantic labels on the various objects in the scene (e.g., couch, table, lamp, etc.).
Each image snippet of Low Fidelity indicates the difference in Texture resolution, Reflections quality, Render Scaling and Shadow quality. The amount of change in each of these settings is customizable through AIP’s Core.

**Data ablation controls** Similar to the ground-truth, we use Blueprint to dynamically alter properties of the environment. We can access and isolate specific properties in different objects. For example, we can isolate metallic objects or rough surfaces with a pixel-perfect binary ground truth. We can also change the fidelity of reflections, lighting, mesh level of detail (LOD), render resolution (either localized to an object or globally), anti-aliasing algorithms (or toggle on and off), or render scaling. Figure 2.4 illustrates the same scene rendered under different fidelity settings. Our scripts are reusable, in the sense that they do not require adaptation to other environments and are also easily portable to other UE projects.
2.1.2 Experiments and Results

We carried out multiple experiments to validate the usefulness of our proposed system. Specifically, we tested AIP in two ways. First, we verified its viability as a data ablation tool. As we detail below, we captured the same images under different fidelity and lighting settings (which we refer to as a scenario), then trained deep neural networks on each scenario to assess the impact of the various environmental features. We carried out both same- and cross-scenario testing (e.g., a Brown/Day/High network on Brown/Night/High). Table 2.1 summarizes the scenarios used. For each scenario, we tested our networks on (1) monocular depth and (2) surface normal estimation, as well as (3) semantic segmentation.

Second, to validate that our virtual data is realistic enough, we tested networks trained with AIP on real-world depth-estimation datasets, achieving results comparable to training on real data alone. Below, we first detail our experimental setup, then discuss each experiment.
Table 2.1: Scenarios used in experiments

<table>
<thead>
<tr>
<th>Default</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps</td>
<td>Lighting</td>
</tr>
<tr>
<td>Brown Room Day</td>
<td>High</td>
</tr>
<tr>
<td>Brown Room Night</td>
<td>High</td>
</tr>
<tr>
<td>Brown Room Day</td>
<td>Low</td>
</tr>
<tr>
<td>Brown Room Night</td>
<td>Low</td>
</tr>
<tr>
<td>Blue Room Day</td>
<td>High</td>
</tr>
<tr>
<td>Blue Room Night</td>
<td>High</td>
</tr>
<tr>
<td>Blue Room Day</td>
<td>Low</td>
</tr>
<tr>
<td>Blue Room Night</td>
<td>Low</td>
</tr>
<tr>
<td>Abstract Shapes Day</td>
<td>High</td>
</tr>
<tr>
<td>Unlit brown Room N/A</td>
<td>High</td>
</tr>
<tr>
<td>Unlit Blue Room N/A</td>
<td>High</td>
</tr>
</tbody>
</table>

*shows settings used, not indicative of all settings available. *diffuse shading.

Figure 2.5: Sample results: Sample images, ground truth, and predictions for semantic segmentation (first three columns), depth estimation (middle columns), and surface normal estimation (last three columns). Figure best viewed onscreen.
**Experimental Setup**  **Image acquisition:** Our Probe script can control the viewpoint by simulating keystrokes. It can move and look freely (yaw and pitch) in the environment. Probe can also send specific commands and can gather images with high overlap (in groups) or low overlap (completely random). Probe’s step size, look sensitivity, randomness of image acquisition (group capture), and number of images to gather are all customizable and can be saved for reproduction across all different scenarios. For our depth estimation experiments, we randomly collected 8265, 640×480 synthetic color images. We collected the same images, by replicating the same camera positions and rotations, across different lighting and fidelity scenarios (Tbl. 2.2). We split these images into 80% for training, and 20% for testing. Similarly, for semantic segmentation and surface normal estimation, we gathered 3000 images for each scenario and split in the same ratio.

**Deep neural networks:** We used a encoder-decoder architecture, and loss function from [25] for depth estimation, and an implementation of U-net [26] from [27] for surface normal estimation and semantic segmentation. We use smooth L1 loss function for Surface Normal Estimation, and Cross-Entropy loss for segmentation task. We use a mini-batch size of 16, learning rate of 0.001, and trained for 51 epochs for all experiments.
Table 2.2: **Depth estimation**: Data ablation test results. Metrics are threshold accuracy ($\delta_i < 1.25^i$), average relative error (REL), root mean squared error (RMS), and average ($\log10$) error. Arrows indicate if higher or lower values are better. For space, we included only some of the conducted experiments; results shown are indicative of the behavior of the trained models in other scenarios.

**SC:** Sanity Check. **L:** Change in Lighting. **M:** Change in Maps. **F:** Positive Change in Fidelity

<table>
<thead>
<tr>
<th>Training Scenario / Fidelity</th>
<th>Test / Fidelity</th>
<th>Goal</th>
<th>$\delta_1 \uparrow$</th>
<th>$\delta_2 \uparrow$</th>
<th>$\delta_3 \uparrow$</th>
<th>REL↓</th>
<th>RMS↓</th>
<th>$\log10↓$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown / Day / High</td>
<td>Brown / Day / High</td>
<td>SC</td>
<td>0.7992</td>
<td>0.9113</td>
<td>0.9474</td>
<td>0.1426</td>
<td>0.0278</td>
<td>0.0740</td>
</tr>
<tr>
<td>Blue / Day / Low</td>
<td>Blue / Day / Low</td>
<td>SC</td>
<td>0.7609</td>
<td>0.8980</td>
<td>0.9278</td>
<td>0.1643</td>
<td>0.0366</td>
<td>0.0858</td>
</tr>
<tr>
<td>Brown / Night / High</td>
<td>Brown / Night / High</td>
<td>SC</td>
<td>0.8333</td>
<td>0.9248</td>
<td>0.9509</td>
<td>0.1327</td>
<td>0.0278</td>
<td>0.0689</td>
</tr>
<tr>
<td>Brown / Day / Low</td>
<td>Brown / Day / Low</td>
<td>SC</td>
<td>0.7719</td>
<td>0.8945</td>
<td>0.9388</td>
<td>0.1544</td>
<td>0.0289</td>
<td>0.0798</td>
</tr>
<tr>
<td>Brown / Day / High</td>
<td>Brown / Night / High</td>
<td>L</td>
<td>0.7616</td>
<td>0.8928</td>
<td>0.9315</td>
<td>0.1711</td>
<td>0.0398</td>
<td>0.0875</td>
</tr>
<tr>
<td>Brown / Night / High</td>
<td>Brown / Day / High</td>
<td>L</td>
<td>0.7366</td>
<td>0.8942</td>
<td>0.9420</td>
<td>0.1904</td>
<td>0.0351</td>
<td>0.0939</td>
</tr>
<tr>
<td>Blue / Day / Low</td>
<td>Blue / Day / High</td>
<td>F</td>
<td>0.7817</td>
<td>0.9062</td>
<td>0.9329</td>
<td>0.1587</td>
<td>0.0370</td>
<td>0.0822</td>
</tr>
<tr>
<td>Brown / Day / Low</td>
<td>Brown / Day / High</td>
<td>F</td>
<td>0.8010</td>
<td>0.9113</td>
<td>0.9475</td>
<td>0.1426</td>
<td>0.0273</td>
<td>0.0731</td>
</tr>
<tr>
<td>Brown / Night / High</td>
<td>Blue / Night / High</td>
<td>M</td>
<td>0.5959</td>
<td>0.8632</td>
<td>0.9079</td>
<td>0.3415</td>
<td>0.0671</td>
<td>0.1193</td>
</tr>
<tr>
<td>Brown / Day / High</td>
<td>Blue Day / High</td>
<td>M</td>
<td>0.6420</td>
<td>0.8528</td>
<td>0.9223</td>
<td>0.2220</td>
<td>0.0433</td>
<td>0.1067</td>
</tr>
</tbody>
</table>
Table 2.3: **Depth estimation:** Results on real-world datasets.

<table>
<thead>
<tr>
<th>Train / Fidelity</th>
<th>Test</th>
<th>$\delta_1$</th>
<th>$\delta_2$</th>
<th>$\delta_3$</th>
<th>REL↓</th>
<th>RMS↓</th>
<th>log10↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown / Day / High</td>
<td>NYUv2</td>
<td>0.3666</td>
<td>0.6012</td>
<td>0.7586</td>
<td>0.5044</td>
<td>0.2014</td>
<td>0.1938</td>
</tr>
<tr>
<td>Brown / Day / Low</td>
<td>NYUv2</td>
<td>0.3720</td>
<td>0.6062</td>
<td>0.7627</td>
<td>0.5010</td>
<td>0.2010</td>
<td>0.1921</td>
</tr>
<tr>
<td>Brown / Night / High</td>
<td>DIODE</td>
<td>0.3563</td>
<td>0.5948</td>
<td>0.7945</td>
<td>0.7659</td>
<td>3.6897</td>
<td>0.2148</td>
</tr>
<tr>
<td>Brown / Night / Low</td>
<td>DIODE</td>
<td>0.3163</td>
<td>0.5647</td>
<td>0.7345</td>
<td>0.7743</td>
<td>3.7898</td>
<td>0.2149</td>
</tr>
<tr>
<td>Brown / Night / High</td>
<td>DIODE - Filtered</td>
<td>0.6546</td>
<td>0.7725</td>
<td>0.8371</td>
<td>0.6608</td>
<td>2.9765</td>
<td>0.1458</td>
</tr>
<tr>
<td>Brown / Day / High</td>
<td>NYUv2 - Filtered</td>
<td>0.5996</td>
<td>0.8405</td>
<td>0.9308</td>
<td>0.2835</td>
<td>0.1232</td>
<td>0.1054</td>
</tr>
<tr>
<td>DIODE/Indoor 28</td>
<td>DIODE/Indoor</td>
<td>0.4919</td>
<td>0.7159</td>
<td>0.8256</td>
<td>0.3306</td>
<td>1.6948</td>
<td>0.1775</td>
</tr>
<tr>
<td>NYUv2 29</td>
<td>NYUv2</td>
<td>0.895</td>
<td>0.980</td>
<td>0.9960</td>
<td>0.1030</td>
<td>0.390</td>
<td>0.0430</td>
</tr>
<tr>
<td>NYUv2 29</td>
<td>DIODE/Indoor</td>
<td>0.2869</td>
<td>0.5097</td>
<td>0.6730</td>
<td>0.6599</td>
<td>2.8854</td>
<td>0.2573</td>
</tr>
</tbody>
</table>

**Monocular depth estimation experiments**  **Data ablation:** Table 2.2 shows a representative sample of the data ablation experiments we conducted using our depth ground truth. For these experiments, we initialized our deep networks using the weights from a network trained on NYUv2. For evaluation, we used the same metrics as those used in [30]: average relative error (REL), root mean squared error (RMS), average log10 error, and threshold accuracy ($\delta_i < 1.25^i$ for $i = [1, 2, 3]$). As we discuss further below, models trained in higher fidelity data generally tend to yield higher scores, even on lower-fidelity scenarios.

**Real-world validation:** To demonstrate the transferability of learned features from a synthetic dataset, we tested our best-performing models on the real-world DIODE and NYUv2 datasets. In addition to the full test set, we also evaluated our networks on a filtered subset that only contained scenes structurally similar to our virtual environments,
i.e., indoor scenes of a living room, with objects such as couches, beds, TVs, etc. As Tbl. 2.3 shows, our high-fidelity trained model had better threshold accuracy on DIODE than a model trained only on NYUv2 [28], confirming that the features learned on our environments are transferable to real-world data. In addition, our model trained on Night lighting, high-fidelity settings achieved 31% $\delta_1$ vs 28% $\delta_1$ of NYUv2 model — 59% $\delta_2$ vs 50% $\delta_2$ of NYUv2 model — 79.4% $\delta_3$ vs 67.3% of $\delta_3$ of NYUv2 model. These results further confirm that our photo-realistic data can match and even exceed real-life training. Furthermore, these models achieved a much higher score in our filtered test set, suggesting that depth estimation is more sensitive to the structure of the input image than to lighting or fidelity. We also believe our models would have performed even better had DIODE used perspective depth (Fig. 2.2).
Table 2.4: **Surface normal estimation:** Metrics are percentage of pixels that differ by 11.5, 22.5, and 30 from the true normal, and mean and median errors. Mean and median are higher than \[31\] because our loss function did not implement hybrid measures to reduce them. This wasn’t necessary since our ground-truth data does not suffer from the problem mentioned in \[31\].

**SC:** Sanity Check. **L:** Change in Lighting. **M:** Change in Maps. **F:** Positive Change in Fidelity

<table>
<thead>
<tr>
<th>Scenario / Fidelity</th>
<th>Test / Fidelity</th>
<th>Goal</th>
<th>11.5↑</th>
<th>22.5↑</th>
<th>30↑</th>
<th>Mean↓</th>
<th>Median↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown / Day / High</td>
<td>Brown / Day / High</td>
<td>SC</td>
<td>0.9014</td>
<td>0.9566</td>
<td>0.9727</td>
<td>24.4575</td>
<td>88.2878</td>
</tr>
<tr>
<td>Blue / Day / Low</td>
<td>Blue / Day / Low</td>
<td>SC</td>
<td>0.9274</td>
<td>0.9746</td>
<td>0.989</td>
<td>30.5607</td>
<td>94.9516</td>
</tr>
<tr>
<td>Blue / Night / High</td>
<td>Blue / Night / High</td>
<td>SC</td>
<td>0.865</td>
<td>0.9224</td>
<td>0.9401</td>
<td>28.2409</td>
<td>69.2181</td>
</tr>
<tr>
<td>Brown / Day / Low</td>
<td>Brown / Day / Low</td>
<td>SC</td>
<td>0.8883</td>
<td>0.9443</td>
<td>0.961</td>
<td>25.3718</td>
<td>81.4871</td>
</tr>
<tr>
<td>Brown / Day / High</td>
<td>Brown / Night / High</td>
<td>L</td>
<td>0.052145</td>
<td>0.2238</td>
<td>0.3464</td>
<td>106.70</td>
<td>121.26</td>
</tr>
<tr>
<td>Brown / Night / High</td>
<td>Brown / Day / High</td>
<td>L</td>
<td>0.050291</td>
<td>0.2135</td>
<td>0.4253</td>
<td>115.82</td>
<td>119.86</td>
</tr>
<tr>
<td>Blue / Day / Low</td>
<td>Blue / Day / High</td>
<td>F</td>
<td>0.195269</td>
<td>0.2683</td>
<td>0.3015</td>
<td>97.832</td>
<td>113.57</td>
</tr>
<tr>
<td>Brown / Day / Low</td>
<td>Brown / Day / High</td>
<td>F</td>
<td>0.028247</td>
<td>0.2102</td>
<td>0.368</td>
<td>109.14</td>
<td>118.08</td>
</tr>
</tbody>
</table>

**Surface normal estimation experiments** We carried out a similar set of data ablation experiments as above, but using surface normal data as the ground truth. Here, we trained each model from scratch, i.e., without pre-trained weights, and used the same evaluation metrics as in \[31\]: mean (average L1 loss), median (average L2 loss), and percentage of pixels that differ by 11.5, 22.5, and 30 relative to the true surface normal. Surface normal estimation is a promising use case for AIP because it is very challenging to capture surface
normal information for real scenes. One needs expensive equipment to measure the angles, and these sensors are extremely hard to calibrate. As Tbl. 2.4 shows, we can successfully train deep networks using AIP (see Fig. 2.5). Overall, surface normal models seem to be less sensitive to photo-realistic features and higher fidelity settings compared to depth estimation or segmentation. Models trained on high fidelity settings perform 2% better than ones trained on low fidelity.

*Table 2.5: Semantic segmentation:* Mean intersection over union (IOU) of all classes for different scenarios. Higher values are better.

<table>
<thead>
<tr>
<th>Scenario / Fidelity</th>
<th>Test / Fidelity</th>
<th>Goal</th>
<th>Global IOU↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown / Day / High</td>
<td>Brown / Day / High</td>
<td>SC</td>
<td>0.8984</td>
</tr>
<tr>
<td>Blue / Day / Low</td>
<td>Blue / Day / Low</td>
<td>SC</td>
<td>0.4119</td>
</tr>
<tr>
<td>Blue / Night / High</td>
<td>Blue / Night / High</td>
<td>SC</td>
<td>0.8335</td>
</tr>
<tr>
<td>Brown / Day / Low</td>
<td>Brown / Day / Low</td>
<td>SC</td>
<td>0.4714</td>
</tr>
<tr>
<td>Brown / Day / High</td>
<td>Brown / Night / High</td>
<td>L</td>
<td>0.6932</td>
</tr>
<tr>
<td>Brown / Night / High</td>
<td>Brown / Day / High</td>
<td>L</td>
<td>0.6418</td>
</tr>
<tr>
<td>Blue / Day / Low</td>
<td>Blue / Day / High</td>
<td>F</td>
<td>0.3862</td>
</tr>
<tr>
<td>Brown / Day / Low</td>
<td>Brown / Day / High</td>
<td>F</td>
<td>0.4188</td>
</tr>
</tbody>
</table>

*Semantic segmentation experiments* Semantic segmentation involves assigning a class label to every pixel on the image. The built-in environments in AIP have fifteen classes, all of which corresponds to regular household objects, e.g., *wall, couch, table, TV,*
plant, etc. We use a label of other for miscellaneous items. As with the surface normals, we trained different networks from scratch on each scenario. We used mean intersection over union (IOU) of all classes as our evaluation metric. As we can see in Tbl. 2.5, model performance is directly linked to a scenario’s fidelity (see Fig. 2.5). Semantic segmentation seems to depend heavily on the render scaling and resolution. At lower settings, borders of the objects are blurry, as is their texture. This causes the model to label them as other since it cannot surely ascertain their object class, thus lowering the global IOU (see Fig. 2.6 for an example).

2.1.3 Data Ablation diagnostic analysis

One of the strengths of AIP is the ability to perform data ablation analysis to diagnose weak and sensitive points in the image, and draw conclusions about the training data and the performance of the model. Below, we mention some of the main sensitivities we found in our models during the data ablation diagnostic analysis.

Sensitivity to lighting: Changes in lighting are a result of the environment, so they cannot be "fixed" by a better acquisition device. As such, a general-purpose model should be robust to them. However, objects can appear in drastically different ways under different lighting conditions, which did affect performance across all experiments. More specifically, segmentation models are particularly sensitive to differences in lighting. In Fig. 2.6 both models labeled the top part of the TV as Wall since they have almost the same color. However, the model trained on a Day setting was much less accurate on the Night image than its counterpart, presumably because the Night setting is darker overall and has more
pronounced reflections. The opposite effect is visible in the reverse case (bottom Fig. 2.6), where the reflection in the lamp confused the model because that level of reflection from sunlight does not exist in the Night lighting.

Our surface normal models are also sensitive to changes in lighting. However, for depth estimation, performance drops only slightly when the lighting is changed, suggesting that local contrast is less important for this problem.

**The impact of fidelity on surface normals vs. segmentation:** Semantic segmentation is very sensitive to changes in fidelity. When objects are blurred due to lower rendering resolution and lower texture clarity, the model appears to be indecisive about picking an object’s class in its border regions. As shown in Fig. 2.5, we see that the model incorrectly classified border regions as *Other*.

In contrast, surface normal estimation is more robust to these kinds of changes. This difference between these two problems highlights the importance of using data ablation tools. Previous studies, e.g., [4, 5, 32], mainly focus on the effects of fidelity on their segmentation experiments. Our findings with surface normals, on the other hand, suggest that fidelity as a general feature of the image might not be enough to draw conclusions about the quality of the data. AIP’s tools allow us to study other aspects of data, such as texture, structure complexity, lighting and more.
Figure 2.6: **Semantic segmentation:** (Image, Ground Truth, Prediction). Top: A model trained on Brown Day High (DH) images segmenting a Brown Night High (NH) image. Bottom: a model trained on Brown Night High tested on Brown Day High. Note the impact of lighting on the final result.

**Perspective vs orthographic depth:** Orthographic depth projection is when light-rays coming to the camera are assumed to be coming from infinity. In calculating the depth ground-truth, this simplification introduces errors to the measurement. We have seen the effects of this assumption on the NYUv2 and DIODE dataset (Fig. 2.2). Specifically, our models’ performance on DIODE was lower in part due to them being trained on perspective depth, which is different from the GT used in DIODE. Although orthographic measurements are currently widely used, we argue that perspective depth, which AIP supports, is the correct way to measure depth.
Impact of fidelity on depth estimation: Generally, the performance of models trained on higher fidelity settings are better than those trained in lower fidelity settings (Table 2.2). However, one exception is when the lower fidelity setting in training better matches the features of the target domain. In Tbl. 2.3, our low fidelity model does slightly better on NYUv2 than the high-fidelity one. We argue this is due to the blur present in NYUv2, which is also present in our low fidelity settings training set due to its lower render settings, making them visually similar. The DIODE dataset, on the other hand, is much more recent, so the depth ground truth was measured with a more accurate sensor. Due to the lack of blur and fuzz on the ground-truth, we did not observe the same kind of performance gain on this dataset.

2.1.4 Contributions

Using AIP, we generated different image datasets and conducted experiments that are nearly impossible with real data, thus demonstrating that AIP is a viable tool for data ablation studies in computer vision. We also verified that our high-fidelity trained models can match or exceed the scores achieved by training with real-data. As suggested by other studies [4, 5, 32, 33], we found that higher-fidelity data is linked to better performance in segmentation, but we also found that sensitivity to scene structure, fidelity and lighting scenario of training data varies from task to task. For example, our surface normal and depth estimation models were not as sensitive to fidelity as our segmentation models were. AIP enables us to change individual features, e.g., quality of shadows, quality of reflections, quality of lighting or resolution of textures, and assess their impact on different models based on the current task. More generally, AIP can help researchers find sensitive points in their
models and aid them in creating high-quality data for training neural networks for a specific computer vision task.
3 PHASE TWO: IMPROVING EXISTING METHODS

3.1 SUPERCAUSTICS: REAL-TIME, OPEN-SOURCE SIMULATION OF TRANSPARENT OBJECTS FOR DEEP LEARNING APPLICATIONS

Detecting transparent objects is one of the most challenging problems in computer vision, because these objects do not form disjoint boundaries with their environment. In addition, different types of transparent objects have unique characteristics based on the materials they’re made of, and they come in varied thicknesses and densities that affects their appearance. Light passing through a transparent object will bounce multiple times before it reaches our eyes (or a camera), causing effects such as refraction, specular reflections, caustics and dispersion; and the dynamic nature of light makes them appear in radically different ways when exposed to light from different angles.
Figure 3.1: Images rendered in SuperCaustics. Changing image features in real time is trivial in SuperCaustics. Note the changes in lighting, sharpness of caustics, realistic backdrop and specular reflections on the transparent objects.

There has been relatively little work on applying deep learning to transparent object detection in part because gathering and labeling a sufficiently large dataset for this problem is cumbersome and difficult. Most recent publications on transparent object detection have utilized small, custom-made datasets of real data, gathered and labelled by the authors themselves to validate or test their algorithm or DCNN model. The actual training data consists primarily of large volumes of synthetic images created in 3D modelling software (e.g., Blender [34]).

Rigidity of these image datasets is a significant problem. For instance, once an image has been generated, one cannot change its illumination, or replace the reflection maps in the
environment. For real images, labeling the ground truth is also a significant challenge, especially for cameras with high pixel counts. Depth, in particular, is almost impossible to label by hand, so one is limited to active depth sensing (e.g., RGB-D sensors) or passive methods like disparity matching. However, depth cameras often register errors when encountering transparent objects due to their reflective and refractive properties [8]. In computer vision applications where transparent objects are involved —such as robotic grasping— precise labeling is an integral part of the entire pipeline, and having fault-free data is crucial to the function of a robotics grasp algorithm. For example, the Cleargrasp pipeline [8], the current state-of-the-art for deep-learning based transparent object detection, uses a segmentation map to complement and remove faulty depth signals from a depth image.

Moreover, no matter how extensive and large a pre-made training dataset is, it can only cover so many lighting conditions. The positions of objects, type of surfaces, camera angles, and placement of light sources will only cover a small subset of all possible combinations. This is a challenge for transparent objects because their appearance is highly dependent on how they are lit. For example, they might cast shadows, caustics, or a combination of the two simply by varying the angle or intensity of the light sources. As such, models trained on synthetic datasets with only a small range of lightning scenarios suffer performance degradation when deployed on real-world data.

More specifically, clever deep learning solutions are often stifled by rigid datasets that cannot represent the full possibility space of the target domain, are difficult to change after acquisition, require significant human effort to produce or use proprietary technology for which the source-code and project files used to generate them are rarely released for free use by the public.
To address this gap, we propose SuperCaustics, an easy to use, highly customizable, easily extendable, open-source real-time simulation tool designed for generating dynamic, massive datasets in highly complex scenarios. SuperCaustics is a modular set of customizable classes developed within NVIDIA’s RTX Branch of Unreal Engine 4 (Epic Games, USA) [10,11], allowing even those researchers who are unfamiliar with Unreal Engine to create, maintain and iterate on their own photorealistic customized datasets for vision tasks involving transparent objects. SuperCaustics features four modules:

1. A stochastic scene generation system that creates the geometry of a scene based on parameters like object type, number of objects, camera type, lighting, and background.

2. A prop manager that adds or removes customizable background items to increase scene complexity and variety.

3. A ground truth core that automatically overlays a wide variety of accurate, pixelwise ground-truth annotations.

4. A data ablation core [7] that enables changing characteristics of the objects, background, environment, lighting, or rendering without affecting scene composition.
Our proposed system allows for fine granular control over small details of the scene before and after generation. Each module has a set of customizable and extendable controls, including tools for data ablation. As detailed in [7], data ablation is the study of the effects of isolated changes in the features of data on the performance of a deep learning network (e.g. having the exact same image under a different lighting condition). Using the Data Ablation Core, we are able to change the characteristics of a scene in real-time, e.g., different lighting color, presence of sharp or soft caustics, light source angle, texture or shape of the backdrop, or changes in roughness/color of the glass surfaces (see Fig. 3.4). The tools in the Data Ablation Core can also be used to increase variety in the captured images.

A synthetic dataset generated from SuperCaustics contains the features of the target domain, free from acquisition errors or labeling bias. More importantly, as our experiments
confirm, a small, targeted, carefully crafted data-set can efficiently match the performance achieved by state-of-the-art methods that train their systems on massive datasets.

Figure 3.3: As illustrated, our system has four components: (1) Scene Generator Module; (2) Prop Manager Module; (3) Ground-truth Core; (4) Data Ablation Core. Each of these modules comes with its own sub-modules that enable generation of a customizable scene in real-time.

3.1.1 Why a flexible system is needed?

The old and the new - A problem that persists: Recognizing transparent objects has been a difficult problem to solve in vision. Early solutions used general features of transparent objects, e.g., specular reflections, for object recognition [35]. More sophisticated approaches include estimating bounding boxes based on the refraction distortions that transparent objects’ edges create on the background [36]. Newer research trains deep convolutional neural networks on massive datasets to reach a pixel-wise labeling of transparent objects [8, 37]. Also, some researchers have devised clever methods to exploit sensor failures in depth images to localize transparent objects [38]. Regardless, often times researchers are forced to
use low-quality real data for testing their algorithm or training a neural network \footnote{36}, and other times researchers resort to creating rigid virtual datasets to train large models, such as a DCNN. \footnote{37,38}.

**The state of current Synthetic Data:** Synthetic data has proven useful in various computer vision tasks, such as depth estimation \footnote{8,12}, surface normal estimation \footnote{7,8}, robotic grasping \footnote{13}, and object segmentation \footnote{6} by multiple independent researchers \footnote{4–7}. Modern computer graphics can achieve near-photorealism, so synthetic data has become a viable alternative in situations where acquiring or labeling real data is difficult. However, there are still few datasets that meet the image quality required to be considered photo-realistic \footnote{37}, and those that do, rarely include transparent objects because of the complexities they introduce in rendering and generating correct ground-truth labels. More importantly, to the best of our knowledge, there is no photorealistic, synthetic dataset that includes the tools and project files needed to generate the data itself. More often, these synthetic datasets are static and use proprietary code and technologies that does not allow for reproduction or change in the dataset. We believe flexibility in creating the image data is crucial since it allows for greater degrees of freedom when evaluating a specific method, and allows researchers to make corrections in the data and address weak spots in the model’s performance.

**The need for reliable data:** One common factor in transparent object detection is the lack of reliable data. In older research, we see the use of personally-gathered real data that suffers from lack of variety, noise, or lack of labels. To get past this obstacle, we see various works generate their own synthetic datasets using tools like Blender \footnote{8,34} or NVIDIA Om-
niverse. As a result, these datasets consist of huge collections of static images. In addition, the developers of these datasets rarely release the project files or source code used to generate the data. Nevertheless, in most cases the target domain of the final algorithm is the real world. As such, to be able to generalize well into the target domain these datasets have to have tens of thousands of images, at least, making the training very expensive and time consuming. Despite all this data, we still see significant performance degradation when shifting from crisp, high-resolution synthetic data to blurry, grainy real images that are lit entirely differently. These synthetic datasets are non-changeable (static) and more often than not, they do not include graphically intense phenomena that happen in real-world scenarios (e.g., dispersion or caustics), which makes them very narrow in domain and makes the trained algorithms sensitive to these lighting conditions.

**SuperCaustics vs. Cleargrasp:** Arguably, the most significant effort in modeling transparent objects is Cleargrasp. In this study, researchers trained deep learning models to estimate depth, surface normals, and occlusion boundaries using a mix of synthetic and real data. Their results indicate that in addition to a significant performance loss when shifting from a synthetic to a real domain, the models consistently misclassify caustics. In particular, they wrongly classify caustics as separate transparent objects. The authors of Cleargrasp speculated that this was because the same amount of sharp caustics were not present in the synthetic dataset due to limitations in the rendering software. A more flexible simulation tool, such as our proposed system, allows for addition or removal of such features. In particular, SuperCaustics can produce images with sharp caustics. More generally, our system can incorporate features of the real domain in the training data to make trained models more
robust to such features in the target domain.

Overall, prior work on transparent object detection has been hampered by a lack of reliable, flexible data. To address this gap, we have developed SuperCaustics, an extendable, open-source, user-friendly simulation tool that allows for fine controls over every variable in the simulation. Our system can generate millions of unique images out of the box. In addition, it allows researchers to import 3D models of their choice, set their desired parameters, and capture synthetic data from fully customizable virtual environments. We describe our system in more detail, below.

3.1.2 SuperCaustics: a Flexible Simulation

SuperCaustics is a simulation tool made in Unreal Engine for generating massive computer vision datasets that include transparent objects. Unreal Engine is the engine of choice for projects with high-resolution, real-time 3D graphics. It is free for both commercial and non-commercial use and its source code is publicly available and extended by the community. We use one such extended version created for hardware raytracing by NVIDIA RTX graphics cards [10].
Figure 3.4: SuperCaustics allows fine controls over fundamental characteristics of every variable in the simulation. e.g., controlling the softness of the light-source and its caustics.

Left: Very soft caustics, Middle: Moderately soft, Right: Very Sharp.

The key contributions of SuperCaustics are: 1. Open-source, free set of user-friendly, extendable tools for creating custom datasets. 2. Customizable photorealistic scenes using real-time hardware ray-tracing, 3. Automatic, multimodal, accurate ground-truth generation for visual tasks, 4. Python and UE4 interplay for data acquisition and processing.

Below, we detail each major module of SuperCaustics, and provide explanations of the different ground-truth annotations available in SuperCaustics.

**Generator Module** The Generator Module creates a scene based on given parameters like item type, shape, number of items and/or range of items. It leverages the physics engine inside UE4 to drop the items on the scene within a customizable distance offset. A random physics impulse can be added to the objects upon spawning to increase randomness and add variety to how they come to rest. The intensity of the impulse is also customizable.

**Props Manager** Similar to the Generator Module, a separate module manages the background items in the simulation. This module accepts different 3D objects as prop inputs,
and manages their position and rotation within the first few frames of the simulation until they come to rest by the physics engine.

**Data Ablation Core**  Data Ablation is a powerful method for determining weak points and improving robustness in a neural network’s performance [7]. The Data Ablation Core in SuperCaustics allows for such experiments within the simulation. Additionally, the features of the Data Ablation Core can be used to further increase the variety in the final dataset. We can adjust the cameras (camera matrix), lighting, backdrops, reflection profiles (HDRI maps), and properties of the glass material (color, specular, thickness, roughness, texture). Additionally, all properties of the ray-tracing engine are exposed for experimentation within the Data Ablation module.

**Ground Truth Core**  SuperCaustics also includes scripts for automatically estimating depth, surface normals (world space and camera space), object masks, outlines, occlusion boundaries and caustics segmentation. The Ground Truth Core can be readily extended by adding additional scripts (or modifying the available scripts). We use post-processing shaders (dubbed *materials* in UE4), to overlay these properties over the image, enabling pixel-perfect alignment between the data and the ground-truth labels. (See Figure 3.2) We detail each ground-truth mode below:

**Depth Ground Truth**  We calculate the normalized distance between each pixel that belongs to a specific object and the camera. By default, we set the real-life range of depth to 10 meters, which covers the entire environment. This range is customizable in the Ground Truth Core.
Figure 3.5: Comparison between world-space surface normals and Camera-space surface normals.

Surface Normals Ground Truth  We estimate the normal vector w.r.t to each 3D surface, then color each pixel to indicate the vector’s direction. We use 6 main colors to show 6 axis of direction (positive and negative xyz). Surface normals can be calculated in 2 different spaces: camera-space and world-space. Normals of a pixel in world-space are defined with respect to the world Cartesian coordinate system. In world-space, each normal vector points towards a fixed axis in space, regardless of where it is being observed from. Camera-space is the space in which points are defined with respect to the camera coordinate system. Each normal vector has the transformation of the camera itself applied to it, making the camera the origin point of the coordinate system. This means the transformations of the camera in world-space is implicitly applied to every normal vector in the image. This is a simplification compared to world-space surface normals. (See Figure 3.2 and 3.5.)
Object Mask Ground Truth  In Unreal Engine, it is easy to map visible pixels to their corresponding 3D objects. Our Blueprint script uses this mapping to automatically detect and overlay pixel-perfect semantic labels on the objects in the scene.

Outlines Ground Truth  The edges of transparent objects are sometimes hard to detect visually. In this Ground Truth pass, we can highlight the edges of objects with a customizable thickness (e.g., 1px or 20px).

Occlusion Boundaries Ground Truth  This Ground Truth pass shows surfaces where transparent objects are touching another surface (like the backdrop). Specifically, this pass shows where there is a depth discontinuity. The logic of this ground truth pass is adapted from ClearGrasp [8].

Local & Non-local Caustics Mask Ground Truth  In SuperCaustics, we have fine control over the characteristics of the image, and we can create the exact same image under different settings. We use the difference in the luminosity between two rendered images (one with caustics, and another without) to reach a pixel-wise segmentation of local caustics (inside transparent object) and non-local caustics (projected unto another surface) in the image.

3.1.3 Evaluation

To validate the usefulness of SuperCaustics, we trained a deep convolutional neural network (DCNN) to do a pixel-wise labeling of transparent objects with both soft and sharp caustics. We chose pixel-wise labeling because it was more suitable given the limited hard-
ware we had available. To demonstrate the flexibility of SuperCaustics, we aimed to test our model on the real test set released by Cleargrasp. To do that, we set the angles of the simulated Intel real-sense cameras to resemble similar perspectives and based the parameters of our simulation on the characteristics of the Cleargrasp’s real-world dataset (i.e., occasional sharp caustics from hard lights, general soft lighting, presence of a tote box and wooden and cloth backdrops among synthetic backgrounds). Below, we compare our network’s performance to the network proposed in Cleargrasp \cite{8}. Our results indicate that our network achieved comparable results with much fewer training data.

**Experimental Setup**

**Hardware** We conducted all our experiments in a Dell Precision 7920R server with two Intel Xeon Silver 4110 CPUs, two GeForce GTX 1080 Ti graphics cards, and 128 GBs of RAM.

**Image acquisition** We generated 9000 1920×1080 synthetic images from 650 randomized scenarios. In each scenario, a reflection map and a backdrop was randomly chosen from a bank of 33 HDRI mappings and 33 backdrops. Then, a number of transparent objects were dropped into the scene from off-camera, so that they came to rest naturally using UE4’s physics engine. Then, the Prop Manager module added the input props in random locations inside the scene. For gathering the images, 12 Intel Realsense cameras were simulated in various locations and angles. In every camera angle, the Data Ablation Core rotated the main light of the scene to cast shadows and caustics at various angles before capturing an
image and its ground-truth values. For training, the images were split in the following ratio: 8000 for training and 1000 for validation.

**Deep Neural Networks** We used an implementation of U-net [40] as our segmentation neural network. In particular, we used the tools provided by the EasyTorch Library to prototype and run the experiments. EasyTorch is a Pytorch-based deep learning library used in [27, 41, 42]. We evaluated our performance on mean intersection-over-union (IOU) for the test set. For the loss function, we used the negative log likelihood loss (NLL). We trained our network for 30 epochs on purely synthetic data from SuperCaustics. Afterwards, we used the SuperCaustics model as a pre-trained starting point (labeled SuperCaustics-R in our results), which was trained on Cleargrasp Known Objects for an additional four epochs.

**Results & Discussion** Table 3.1 shows the object segmentation performance of our network trained on SuperCaustics data, and Table 3.2 shows the equivalent results when segmenting caustics directly. In particular, the first table shows that our SuperCaustics-R model (which was also trained on a subset of real data from Cleargrasp) generalizes well to novel objects. Our network trained only with synthetic data achieves 88% IOU on the same domain, and it is robust to challenging image features like sharp caustics and segmenting multiple overlapping objects. Moreover, our SuperCaustics-R model achieves a performance of 53% IOU (comparable to Cleargrasp’s 58%) on the Novel Objects test set. Below, we analyze these results in more detail.
Table 3.1: Models trained with SuperCaustics Data generalize well to novel scenarios

<table>
<thead>
<tr>
<th>Training</th>
<th>Testing</th>
<th>Accuracy</th>
<th>F1 Score</th>
<th>Precision</th>
<th>Recall</th>
<th>IOU</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperCaustics</td>
<td>SuperCaustics (Known Objects)</td>
<td>0.9924</td>
<td>0.9361</td>
<td>0.9975</td>
<td>0.8818</td>
<td>0.8801</td>
</tr>
<tr>
<td>SuperCaustics-R</td>
<td>ClearGrasp-Real (Novel)</td>
<td>0.9560</td>
<td>0.6941</td>
<td>0.5627</td>
<td>0.9056</td>
<td>0.5316</td>
</tr>
<tr>
<td>ClearGrasp</td>
<td>ClearGrasp-Real (Novel)</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
<td>0.5800</td>
</tr>
</tbody>
</table>

Figure 3.6: Image (left), Prediction (middle), Ground-truth (right) First two rows show SuperCaustics Model on SuperCaustics data. Second two rows show SuperCaustics-R on ClearGrasp-Novel.
Performance, Efficiency and Scalability: Compared to [8], our training procedure is much more efficient. By producing higher quality, wider-domain data, we can achieve roughly the same performance on the Novel Objects test set using only 10% of the amount of data and in a fraction of the training time used by Cleargrasp (34 epochs on about 8000 images, compared to 100 epochs on 80000 images). Moreover, our training setup did not require 8 Tesla v100 cards as used by Cleargrasp. Achieving comparable results more efficiently supports our assertion that for difficult, complex tasks like labeling shiny transparent objects with caustics, using a rigid, massive dataset is not ideal, and one can always benefit from the availability of a better curated, higher quality dataset. That is why open-source, user-friendly simulations like SuperCaustics are important, because for every use-case scenario, a well-made custom dataset is going to achieve comparable results quicker and cheaper than traditional, massive datasets.

Robustness to Caustics: Fig. 3.6 shows examples of images with sharp caustics that were produced by our SuperCaustics system. Our trained model does not confuse caustics with separate transparent objects and is quite robust to caustics from various angles and light sources. This is due to the presence of natural caustics in the dataset.

Performance on the Real Domain: As Fig. 3.6 shows, our trained model seems to have learned the general characteristics of transparent objects from the SuperCaustics synthetic data. However, as opposed to Cleagrasp where known objects were present in the synthetic data, our model was not exposed to the shape of these novel objects. Performance in the real domain was also affected by the grainy, noisy nature of the real-camera. Prior work
has shown that segmentation models are very sensitive to fidelity [7], thus, a change in the
general fidelity of the domain had a significant, but expected impact on the performance of
the model.

**A Solution to Cleargrasp’s main problem — Segmenting Non-local Caustics:** In
Cleargrasp’s original experiments, their trained models falsely classify caustics as separate
transparent objects [8]. One possible solution for this problem is having a separate system
that only classifies caustics, and because predictions in these regions tend to be invalid,
masking the caustics out of the predictions. This setup is straightforward with SuperCaust-
tics. To test this, we generated 2300 $1280 \times 720$ images using the caustics segmentation mask
ground-truth. We then trained a segmentation model to identify sharp, refractive, and re-
reflective non-local caustics for 50 epochs. We then generated 60 images using novel objects
(not seen before by the model) to test the generalization of the model. As Table 3.2 shows,
our model was able to segment caustics cast by novel objects at 68% IOU rate. As such, we
believe that training a model with our SuperCaustics system will prove a feasible solution
for masking out the erroneous pixels in Cleargrasp.
Figure 3.7: Image (left), Ground-truth (middle), Prediction (right), SuperCaustics Model on Segmenting Caustics cast by Novel transparent objects.
### Table 3.2: Caustics Segmentation

<table>
<thead>
<tr>
<th>Training</th>
<th>Testing</th>
<th>Accuracy</th>
<th>F1 Score</th>
<th>Precision</th>
<th>Recall</th>
<th>IOU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caustic Segmentation</td>
<td>Caustic Segmentation (Known Objects)</td>
<td>0.9981</td>
<td>0.9386</td>
<td>0.9541</td>
<td>0.9237</td>
<td>0.8844</td>
</tr>
<tr>
<td>Caustic Segmentation</td>
<td>Caustic Segmentation (Novel Objects)</td>
<td>0.9958</td>
<td>0.8102</td>
<td>0.8682</td>
<td>0.7594</td>
<td>0.6809</td>
</tr>
</tbody>
</table>

**Sensitivity to Brightness:** Our trained model seems to be very sensitive to brightness in the features of the objects. In our tests with synthetic data, where the camera’s exposure underexposes shadows, our model seems to make mistakes in segmenting transparent objects.

**Side-Effect of the Patch Based Approach:** In some of the images, there is some noise visible in the labeled pixels. We believe this is due to the patch-based approach used by the U-Net neural network; specifically, the noise is a result of the artifacts from stitching these patches together in the final image. In most cases, patches are surrounded by uniform neighboring pixels, which could be used as information in a context-based post processing algorithm to get rid of the noise.

### 3.1.4 Contributions

SuperCaustics is an extendable, open-source, and user-friendly simulation tool for transparent objects that allows for fine controls over every variable in the simulation. Our system allows users to create customized datasets using its flexible simulation tools. With SuperCaustics, we hope to address the dearth of high-quality data for transparent objects, particularly data that can be modified post-acquisition. We demonstrated that by using a
better curated data, a network can achieve performance comparable to when training with much larger rigid datasets. In particular, we were able to achieve comparable performance to the state-of-the-art networks trained on Cleargrasp’s data using a dataset of only 9000 RGB images. Specifically, we found that high-quality data is linked to better performance in segmentation, as our SuperCaustics-R model was able to approach the performance of state-of-the-art (cleargrasp) with only 34 epochs of training and 10% of the data, on far more reasonable hardware.
4 PHASE THREE: APPLICATIONS ON DIFFICULT TASKS

4.1 OPENWATERS: PHOTOREALISTIC SIMULATIONS FOR UNDERWATER COMPUTER VISION

Computer vision can turn noticeably more difficult when applied to under water images. Light transport in a dense medium like water creates phenomena such as caustics, refraction, dense scattering and absorption. These effects can dramatically change how objects or environments appear in images. This is attributed to the loss in optical signal energy due to the high absorption in dense water medium (compared to air) and scattering. With less light energy arriving at the camera lens, the image formation undergoes various types of artifacts or distortions. Such characteristics cannot be easily fixed through image filtering mechanisms and are often very complex and difficult to control after image acquisition. Compared to the land domain, underwater images require significantly more effort to gather for any type of visual task. This additionally makes preparation of underwater data for machine/deep learning analysis very challenging due to the manual effort required for labeling (supervised) and data cleanup (unsupervised). Availability of large volume datasets has enabled deep learning methods to achieve state-of-the-art results in computer vision depth estimation. However, these datasets are immutable, and modification of imaging characteristics is not possible after acquisition.
For example, imagine that we want to assess how an environment’s texture affects the system’s ability to estimate depth (distance between camera and image). In this case, we would need to compare the system’s output across different environments and hope that the impact of other features, (e.g., lighting or shape) cancels out across the samples. This makes certain diagnostic experiments like data ablation nearly impossible. Since such features are usually expensive to change, it can make introducing variety into an image dataset and producing data prohibitively difficult. The only way to achieve such effects is by recording every step, and manipulating the source of the data before acquisition; requiring a tightly controlled environment and precise measurements.

Manual labeling of depth in underwater images is very challenging, so one is limited to active depth sensing (e.g., RGB-D sensors) or passive methods like disparity matching. Gathering data from underwater sources often requires expensive, delicate equipment that need to be enclosed in watertight containers, which can particularly affect depth estimation cameras, by blocking depth sensing rays in active depth sensing cameras; or making it difficult to track points in passive stereo vision cameras. All assuming that there is access to a controlled, physical underwater setup (i.e. a pool) to gather images from. As such, underwater vision problems rarely have large amounts of labelled data with accurate ground-truth annotations.

The limitations in resource availability, imperfection in measurement tools and difficulty in gathering real-world underwater data can be prohibitive for researchers. Since solving many vision tasks with machine learning require massive amounts of labelled data, simulations can be a viable resource for solving underwater computer vision problems. However,
underwater simulations can be outdated, not taking advantage of modern hardware, or have a different focus than computer vision.

**OpenWaters.** To address the fundamental limitations in data collection for underwater computer vision, in this paper, we have designed OpenWaters, an underwater simulation kit designed to be highly customizable, easily extendable, run in real-time and open-source. An OpenWaters simulation can be shaped for any underwater visual task, including visible light communications, depth estimation and image enhancement. OpenWaters is capable of generating dynamic, massive image datasets in highly complex scenarios. OpenWaters is a modular set of customizable abstract classes developed within NVIDIA’s RTX Branch of Unreal Engine 4 (Epic Games, USA) [10][11]. OpenWaters is built with compatibility and low entrance barrier in mind, allowing even those researchers who are unfamiliar with Unreal Engine to create, maintain and iterate on their own photorealistic customized datasets for any vision task underwater. OpenWaters contains three key modules: (1) A stochastic scene generation system that creates the geometry of a scene based on parameters like objects, objectives, camera type, lighting, and backgrounds; (2) A ground truth core that automatically calculates and overlays a wide variety of accurate, pixelwise ground-truth annotations; (3) A data ablation core that enables changing characteristics of the simulation (i.e water, environment, lighting, timescale or rendering) in real-time without affecting scene composition.

### 4.1.1 Related Work

**Underwater Simulations.** Currently, there are open-source underwater simulations available. Prior simulations exclusively developed for underwater uses are UWSim (published in
These tools generally allow for visualization of virtual underwater scenarios, and provide simulations of rigid body dynamics or sensors such as sonar or pressure sensors. However, these simulation tools haven’t been recently updated or developed to support modern hardware. More importantly, these simulations do not focus on real-time, realistic image rendering with hardware-accelerated ray tracing, nor they are designed for modern diagnostic methods such as data ablation. In comparison to other simulation tools, our system — OpenWaters — uses hardware ray tracing to accurately generate Photorealistic images in real time, its designed to be user friendly, and has integrated tools for data ablation experiments.

**Underwater Depth Estimation.** Scattering and absorption make underwater depth estimation more challenging in comparison to on-land scenarios. Prior work on Depth estimation has produced state-of-the-art results for in-land datasets such as NYUv2 or DIODE-Depth. Using deep learning, pixelwise accurate measurements can be achieved that all but eliminate the need for active depth sensing or stereo cameras for indoor scenes. The indoor data-sets gathered with active depth sensing cameras can use the structure of the scene to infer a pixel-wise depth map. Although these laser cameras work on indoor or outdoor scenes, submerging them underwater will interfere with their depth sensing capabilities. This has led to studies using unsupervised methods for underwater depth estimation, or bringing images from other domains (Style transfer) using methods such as Generative Adversarial Networks (GANs), so they resemble underwater images. Generally, these studies seem to be hampered by the lack of reliable, flexible data.

**Using Synthetic Data.** Modern computer graphics can achieve near-photorealism, so synthetic data has become a viable alternative in situations where acquiring or labeling real
data is difficult. Synthetic data has been proven to be useful in complex computer vision
tasks, such as depth estimation [7,8,12], surface normal estimation, robotic grasping [8,13]
and object segmentation [9] by multiple independent researchers [5,7,8].

Image Enhancement and Style Transfer. A common approach in obtaining labelled
data for underwater vision purposes is to enhance currently available underwater images,
or use style transfer methods to make underwater data out of images captured on in land
environments [51,52]. Enhancement in this context refers to removing "unwanted" features
such as haziness from underwater images. For instance, researchers in [54] use a Convolu-
tional Neural Network (CNN) to estimate the ambient light and dehaze underwater images.
Similarly, in [55] a CNN learns cross-domain features that exist between air and underwater
images. The challenge in either Enhancement or Style transfer approaches is by altering
the image, some features that could be useful in training a neural network are also affected.
By removing (Enhancement) or adding features (Style transfer) we are effectively modifying
images into a state that might not have sufficient overlap with our target images, and since
the data is already captured, isolating such features may not be entirely under our control.
Some of these image features like moderate murkiness can be useful in predicting depth maps
in underwater images.

The key contributions of OpenWaters are: (i) Open-source, free set of user-friendly,
extendable tools for creating customized datasets of underwater scenarios, (ii) A set of cu-
rated, customizable photorealistic scenes for underwater visual communications using real-
time hardware ray-tracing, (iii) Extendable, Automatic, multi-modal, accurate ground-truth
generation for visual tasks, and (iv) Compatible Python scripts for data acquisition and
processing.
Figure 4.1: OpenWaters Simulation Architecture

Figure 4.1 depicts the architecture of our OpenWaters simulation toolkit. Below, we describe the key elements of the system.

**Environments Core:** This module contains the over-all code for generating novel scenes. It also contains several extendable abstract classes that can be used to target a wide range of domains. This covers the environment geometry and objects (e.g. pool or light transmitters), lighting (direct sunlight, diffused skylight, pool lights), water (clarity, color, surface level, surface agitation), rendering features (caustics, ray-tracing settings, resolution), Scene Materials and time (dilation or freezing).

**Data Ablation Core:** This module translates control signals from a human operator or a python script. These control signals apply changes in the environments generated by the environments core. We can adjust the cameras (camera matrix), lighting, Items, Rendering (e.g. reflection profiles or HDRI maps), Geometry and properties of the materials in the
scene. Additionally, all of the ray-tracing engine parameters are exposed for experimentation within the Data Ablation module. The features of the data ablation core are adapted from [7,9].

**Ground-truth Core:** OpenWaters includes scripts for automatically estimating depth, surface normals (world space and camera space), object masks and caustics segmentation. The Ground Truth Core is easily extendable by adding or modifying the existing scripts. We overlay these properties directly over the image with pixel-perfect alignment between the data and the ground-truth labels. In this work we explore only the depth ground truth core. This core annotates the calculated and normalized distance between each pixel that belongs to a specific object and the camera. By default, we set the real-life range of depth to 1000 centimeters, which covers the entire environment. This range is customizable in the GT Core.

![Figure 4.2: Ground-truth annotations from OpenWaters - First row: Real-time Generated Image, Image without water caustics, Caustics segmentation, World-space surface normals, Object Mask segmentation, Depth Ground truth](image)
4.1.2 Evaluation

To validate our OpenWaters Simulation, we use it to generate a dataset of 13,000 640×480 RGB and Depth images, then we train a deep convolutional neural network (DCNN) from scratch to estimate depth in underwater scenarios. We chose depth since our real ZED camera did not produce reliable depth images underwater (See Figure 4.6.) Our images feature moderate to high levels of scattering and absorption (normal and murky water). In particular, since we aim to see the transference of learned features into the real domain for underwater depth estimation. Since reliable, accurate ground-truth data is not available for underwater depth estimation in real domain, we exclusively use synthetic data for training. Our experiments show the model learns transferable features from the synthetic domain, that can be applied to real setup and infer depth from monocular images. Our data ablation experiments show the model attempting to extract perspective clues from the structure and texture, and using absorption and scattering as clues in predicting depth.

Figure 4.3: Our physical experimentation setup. We use this setup to model an underwater visual communications simulation on OpenWaters. We also gather real-world test RGB images from our real underwater setup.
**Experimental Setup** To gather real underwater images, we constructed a physical underwater setup. As Figure 4.3 shows, our physical setup is an Intex pool of size 20’ × 10’ × 5’, with custom steel rails held together by cinder blocks that allow for horizontal movement of the camera and LED capsule. Our transmitter and the receiver (ZED stereo camera) are submerged and mounted on the mentioned rails inside Blue robotics watertight enclosure capsules. Our Simulation instance is modeled after this physical setup.

**Simulation Instance** To model the physical setup, we create a set of custom 3D meshes. For the transmitter capsules, we model them inside Unreal Engine as per the technical details provided by their manufacturer. We also model the Intex pool’s dimensions in the physical setup. These meshes are available for public use with the released open-source code. The virtual transmitter and receiver are mounted similarly to our physical setup on virtual rails that allow for horizontal movement. The environments core generates a random location along the mounting bar to place the transmitter, and pseudo-random transform (location, pitch, yaw, roll) for the receiver. In each scenario, the transmitter and receiver are automatically mounted on random locations along the rails, and the objects are moved horizontally (similar to the physical setup) to introduce depth variety. Additionally, the main light source (sun) is randomly rotated in every frame to introduce lighting variety in the data. For setting up the environment generation, we create an instance of the environments core, and add our 3D meshes as its child components. We then set up the general geometry of the static objects (rails) in the scene. For the camera, we use the cinematic camera class available in Unreal Engine to replicate the camera matrix of the ZED Camera. For data ablation tests, we create 3 instances of the data ablation core, each controlling an exclu-
sive aspect in the simulation: 1) movement of the light sources, camera and transmitter 2) murkiness of the water, texture of the pool 3) controlling time, ground-truth core and image utilities.

**Metrics.** For evaluation, we used the same metrics as those used in [48]: average relative error (REL), root mean squared error (RMS), average log10 error, and threshold accuracy ($\delta_i < 1.25^i$ for $i = [1, 2, 3]$).

**Hardware.** We conducted all our experiments on a Computer with an AMD Ryzen 5 5600x CPU, a GeForce RTX 2080 Ti graphics card, and 16 GBs of RAM.

**Deep Neural Networks.** We used the encoder-decoder architecture, and loss function from [25], with implementation from [14]. We train our model on purely synthetic data generated by OpenWaters simulation for 50 epochs. Dataset size is 13000 Images, split 80% (10400) for Training and 20% (2600) for validation.

![Figure 4.4: Neural Network architecture. Figure adapted from [14] Rectangles represent convolutional layers, and change in their size indicates pooling and up-sampling.](image)
4.1.3 Results & Discussion

To test our model, we generate 80 images in various scenarios using the data ablation utilities in the simulation. For each scenario, camera and lighting are set at random positions, then the data ablation core freezes time to prevent changes in the caustics pattern, scattering and water particles. Then, it captures ground-truth and the test image. To accurately measure the effects of change in the isolated features, the test images are exactly the same, with only the specified parameters changing. (See Figure 4.8).
Performance on Real Domain. To showcase the flexibility of OpenWaters, we set the parameters of our simulation to model the physical setup. In particular, we simulate the camera matrix of the ZED camera, set the scattering and attenuation parameters to similar values, and implement the structure and dimensions of the physical environment.

To get a better grasp of the model’s performance, we recreated images from the physical setup in OpenWaters, where we have access to accurate depth ground-truth. These recreations are featured in figure 4.5. We then manually matched the real images and synthetic ground-truth by overlaying them pixel-by-pixel in Adobe Photoshop [59]. This allows us to run a semi-quantitative evaluation on the performance of the model. As seen in Table 4.3 performance of the model in real images is consistent with our data ablation diagnostic analysis on synthetic images. Overall, our model can generate depth maps in real underwater images without being trained on real data. Based on our qualitative and semi-quantitative analysis, it seems to have picked up useful features from its training with synthetic data. Figure 4.5 also shows predictions of our model vs. DenseDepth [14] in our real underwater setup.

Table 4.1: To evaluate the robustness of the model, we run a set of data Ablation tests in the same domain. As indicated by the results, Our model is robust to changes in lighting or murkiness. Arrows indicate if higher or lower values are better.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Goal</th>
<th>(\delta_1\uparrow)</th>
<th>(\delta_2\uparrow)</th>
<th>(\delta_3\uparrow)</th>
<th>REL(\downarrow)</th>
<th>RMS(\downarrow)</th>
<th>log10(\downarrow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Intex Pool Sanity Check (Same domain test set)</td>
<td>0.9699 0.9895 0.9949 0.0444 0.0214 0.0193</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Murky Effect of heavy scattering in same domain</td>
<td>0.9671 0.9885 0.9945 0.0486 0.0256 0.0216</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Sunlight Effect of sunlight in same domain</td>
<td>0.9652 0.9877 0.9941 0.0529 0.0264 0.0234</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4.2: Data Ablation tests. In each experiment, image features are isolated to determine the result of experiment goal. Arrows indicate if higher or lower values are better.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Goal</th>
<th>δ₁ ↑</th>
<th>δ₂ ↑</th>
<th>δ₃ ↑</th>
<th>REL ↓</th>
<th>RMS ↓</th>
<th>log₁₀ ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Intex Pool</td>
<td>Sanity Check</td>
<td>0.9099</td>
<td>0.9895</td>
<td>0.9949</td>
<td>0.0444</td>
<td>0.0214</td>
<td>0.0193</td>
</tr>
<tr>
<td>Procedural Tiles</td>
<td>Effect of change in pool texture</td>
<td>0.5878</td>
<td>0.6533</td>
<td>0.7198</td>
<td>0.1027</td>
<td>0.1255</td>
<td>0.2273</td>
</tr>
<tr>
<td>Procedural Tiles: Murky</td>
<td>Effect of heavy scattering + pool texture</td>
<td>0.6053</td>
<td>0.6698</td>
<td>0.7043</td>
<td>0.7968</td>
<td>0.1076</td>
<td>0.2723</td>
</tr>
<tr>
<td>Clay bottom</td>
<td>Effect of change in pool texture</td>
<td>0.5921</td>
<td>0.6781</td>
<td>0.7283</td>
<td>0.4378</td>
<td>0.1178</td>
<td>0.2567</td>
</tr>
<tr>
<td>Clay bottom: Murky</td>
<td>Effect of heavy scattering + pool texture</td>
<td>0.6007</td>
<td>0.6855</td>
<td>0.7408</td>
<td>0.6138</td>
<td>0.1423</td>
<td>0.2166</td>
</tr>
<tr>
<td>Ceramic Tiles</td>
<td>Effect of change in pool texture</td>
<td>0.3098</td>
<td>0.4458</td>
<td>0.5372</td>
<td>0.5971</td>
<td>0.2645</td>
<td>0.4428</td>
</tr>
<tr>
<td>Ceramic Tileds: Murky</td>
<td>Effect of heavy scattering + pool texture</td>
<td>0.5885</td>
<td>0.6437</td>
<td>0.6727</td>
<td>0.4417</td>
<td>0.1125</td>
<td>0.3465</td>
</tr>
<tr>
<td>Irregular Tiles</td>
<td>Effects of Non-Cubic texture</td>
<td>0.4184</td>
<td>0.5678</td>
<td>0.6734</td>
<td>0.5317</td>
<td>0.2171</td>
<td>0.2707</td>
</tr>
<tr>
<td>Irregular Tileds: Murky</td>
<td>Effects of Non-Cubic texture + Scattering</td>
<td>0.6540</td>
<td>0.7385</td>
<td>0.7887</td>
<td>0.4067</td>
<td>0.0847</td>
<td>0.1887</td>
</tr>
<tr>
<td>Dark Pattern</td>
<td>Effects of Non-Cubic texture</td>
<td>0.1641</td>
<td>0.3483</td>
<td>0.5089</td>
<td>0.6850</td>
<td>0.3116</td>
<td>0.3574</td>
</tr>
<tr>
<td>Dark Pattern: Murky</td>
<td>Effects of Non-Cubic texture + Scattering</td>
<td>0.6077</td>
<td>0.7201</td>
<td>0.7806</td>
<td>0.4193</td>
<td>0.0982</td>
<td>0.1909</td>
</tr>
<tr>
<td>Cobblestone</td>
<td>Effects of Non-Cubic texture</td>
<td>0.2794</td>
<td>0.4357</td>
<td>0.5108</td>
<td>0.5724</td>
<td>0.2603</td>
<td>0.5011</td>
</tr>
<tr>
<td>Cobblestone: Murky</td>
<td>Effects of Non-Cubic texture + Scattering</td>
<td>0.5590</td>
<td>0.6081</td>
<td>0.6288</td>
<td>0.4181</td>
<td>0.1226</td>
<td>0.3798</td>
</tr>
</tbody>
</table>

Table 4.3: Semi-Quantitative analysis of model performance on Real Images. Performance drop is consistent with our data ablation diagnostic results. These numbers serve as a rough representation of model performance on real images, and not its actual performance. Arrows indicate if higher or lower values are better.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Goal</th>
<th>δ₁ ↑</th>
<th>δ₂ ↑</th>
<th>δ₃ ↑</th>
<th>REL ↓</th>
<th>RMS ↓</th>
<th>log₁₀ ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Intex Pool</td>
<td>Sanity Check (Same domain test set)</td>
<td>0.9099</td>
<td>0.9895</td>
<td>0.9949</td>
<td>0.0444</td>
<td>0.0214</td>
<td>0.0193</td>
</tr>
<tr>
<td>Real Setup</td>
<td>Semi-quantitative analysis</td>
<td>0.3862</td>
<td>0.5709</td>
<td>0.7195</td>
<td>0.4215</td>
<td>0.2653</td>
<td>0.2556</td>
</tr>
</tbody>
</table>
Figure 4.6: Comparison between ZED camera raw depth output and OpenWaters depth prediction. The ZED Camera output is unreliable in underwater scenarios.

Unreliable Depth Ground-truth in Real images. As seen in Figure 4.5, our model can generate a depth map by from a monocular real image without prior training on real data. We confirm the findings of other studies that depth cameras do not yield reliable results when submerged underwater [43]. Passive depth sensing cameras like the ZED Camera [56] use disparity matching to calculate a depth map. In underwater images, there are fewer structural key points to track for depth. Furthermore, lens-like curvature of the blue robotics capsule [57] used to submerge the camera setup underwater introduces an uneven pincushion distortion around the edges of the frame that further interferes with disparity matching in camera. As such, our ZED camera does not produce an accurate depth map in the real
underwater setup. As seen in Figure 4.6, our depth estimation neural network can predict depth where the real camera produces an unreliable depth image.

![Image of depth estimation](image)

*Figure 4.7: Synthetic data tests. Top two rows: Blue Intex pool, Two bottom rows: Ceramic Tiles, Cobblestone.*

Table 4.1 shows the performance of our network, trained entirely on synthetic data generated by OpenWaters Simulation. In particular, the first table shows that our Depth estimation model achieves close to 97% ($\delta < 1.25^3$) accuracy in the test set, which contains
challenging features such as water caustics, scattering and lighting variety. Our depth estimation model is quite robust to changes in lighting or scattering amount (murkiness). We argue this is due to these features being present in the training data. This table further confirms the feasibility of training deep learning models on synthetic data. In particular, it shows the model can learn to estimate depth from monocular synthetic images.

Table 4.2 shows the results for our data ablation tests (sample images in Figure 4.8) on the effects of pool texture and scattering on the performance of the depth estimation model. Below, we go over our insights from the Data Ablation diagnostic analysis.

**Figure 4.8:** Data Ablation: Isolating specific features (in these images, murkiness and pool material) inside the exact same image, to evaluate their effects on the performance of a neural network. *(Figure best viewed on screen)*

**Effect of change in pool texture.** Using the data ablation core in OpenWaters, we perform experiments that are extremely difficult to do in real physical setups. One of which
is introducing a change in the pool texture, while keeping all else exactly the same. The texture of the pool is directly related to the environment, since an object’s surface material will affect its interactions with light, and it can dramatically change how it appears in an image. We see an expected performance drop similar to a dataset shift problem when switching to other types of texture materials for the pool.

**Looking for Perspective clues.** One reason for trying different pool textures is seeing the effect of perspective clues in the performance of the model. Many pools come with uniform, uni-color, often cubic tiles that can easily be tracked by the model as an easy perspective clue. In fact, this effect can be seen in our real images test, where the model guesses depth on the pool bottom almost perfectly. However, in case of a more complex texture like procedural tiles or non-cubic textures like cobblestone, we see a sharp decrease in performance. (see table 4.2) We argue this is because it is much harder for the model to extract perspective information from such irregular textures, compared to uniform tiles. We might be able to improve this by using the data ablation core to add more variety to the training data.

**Scattering can be a perspective clue.** An interesting find in our data ablation experiments is the positive effect of scattering in the performance of the model, specifically in out-of-domain scenarios. As mentioned earlier, it is evident that our model searches for perspective clues when attempting to predict depth. One such perspective clue is the murkiness of the water, since light from certain distance gets absorbed and scattered, the model uses that information to infer depth from the image. This can be a useful insight when developing underwater depth estimation models. This behaviour is most visible in the ”dark pattern” pool texture, where adding scattering hugely improves model performance.
4.1.4 Contributions

In this paper, we introduced OpenWaters: an open-source, extendable and user friendly simulation kit that enables fine control over every variable, including realistic rendering parameters and ground-truth labels. OpenWaters can be shaped to support any type of underwater visual task, and allows users to create and iterate over their own modifiable, customized datasets and interactive underwater environments. We used OpenWaters to train a deep neural network, predicting depth from single underwater images. We demonstrated our model’s ability to learn from synthetic images generated from OpenWaters, and provided semi-quantitative and qualitative analysis on how this model can perform in real images. In addition, we showcased the diagnostic tools provided with OpenWaters by performing data ablation diagnostic analysis on the depth estimation model to determine its weak points and infer insights for training neural networks for underwater computer vision. These kinds of diagnostic tools can be of utmost importance in fine-tuning deep learning models, understanding their behavior and discovering sensitive points. We posit that our analysis serves as an example of the types of experiments OpenWaters simulation kit enables in the future.
5 PHASE FOUR: ACHIEVING STATE OF THE ART PERFORMANCE

5.1 IDEHAZE: SUPERVISED UNDERWATER IMAGE ENHANCEMENT & DEHAZING VIA PHYSICALLY ACCURATE PHOTOREALISTIC SIMULATIONS

There is an increasing need for underwater imagery in applications ranging from unmanned underwater vehicles (UUVs) to oceanic engineering and marine biology research. However, water is denser and more dielectric than air, so capturing clear images underwater is much more challenging than on land. Specifically, since water absorbs and scatters more light than air, a submerged image sensor can capture less information about the surrounding environment, leading to a hazy, blurry image.
Underwater image enhancement and restoration (UIER) seeks to remedy this by using image processing techniques to enhance the images after they’ve been captured, specifically by applying dehazing (to remove scattering effects) and color correction (to reduce absorption effects) to the raw image. The state of the art for most forms of image analysis is currently deep learning due to its unmatched ability to learn task-relevant features. However, applying deep learning to UIER is challenging due to the dearth of data in this domain. As it’s well known, deep neural networks require vast amounts of (mostly labeled) data to achieve good results. Specifically, deep learning’s impressive results on challenging computer vision tasks, such as depth estimation, surface normal estimation and segmentation have leveraged (mostly free) high-quality datasets. In contrast, underwater image data is expensive to acquire due
to the equipment and transportation costs involved. Capturing underwater images also requires specialized skills, and the data is far more time-consuming to label. As such, free, high quality underwater images are scarce.

In this paper, we propose a novel, two-step supervised method for underwater image dehazing and color correction. Our approach combines state-of-the-art deep learning with synthetic data generation, the latter to address the aforementioned dearth of real image data. In more detail, we use Unreal Engine 4 [11] to model an underwater environment, which we then use to capture synthetic images for training our deep neural networks. This UE4 environment has physically accurate absorption and scattering dynamics that control the amount of signal loss in the rendered image. We can also adjust the level of simulated molar concentration and scattering wavelengths to model different levels of haze in an image. Importantly, by turning off these effects, we can obtain a ground-truth image with no haze, which we can use to train a model for image dehazing. For color correction, we use a subset of the UIEB dataset [60] to train a simple model that transforms the colors into the target domain. As we show in Section 5.1.2, our two-step approach achieves state-of-the-art results in underwater image enhancement. In particular, it improves clarity and can restore small details in distant parts of the underwater image.

The key contributions of this paper are as follows:

1. Design and implementation of a unique photorealistic 3D simulation system modelled after real-world conditions for underwater image enhancement and dehazing.

2. A deep convolutional neural network (CNN) for underwater image dehazing, trained on pure synthetic data.
3. A deep convolutional neural network trained on a subset of the UIEB dataset for underwater image color transfer.

4. A customizable synthetic dataset/simulation toolkit for training and diagnostics of underwater image enhancement systems with robust ground-truth and evaluation metrics.

5.1.1 Related Work

Below, we review key prior work on underwater image enhancement and synthetic data generation for computer vision tasks.

**Underwater Image Enhancement:** Restoring an underwater image is often labelled as "dehazing" or "enhancement" and presented as a cumulative process in which the colors of the image are enhanced through a color correction pass, and local and global contrast is altered to yield the enhanced final image. Such pipelines are often collections of linear and non-linear operations through algorithms that break-down images into regions \[61\], or estimate attenuation and scattering \[62\] to approximate real scattering and correct it accordingly. However, for reasons we’ll explore further in Section 5.1.2, color correction and dehazing are two different problems that require their own separate solutions.

**GANs and Synthetic Data Generation:** Use of synthetic data has been the topic of several recent publications, and the application of synthetic data varies greatly depending on the method of data generation. In this context, synthetic data mostly refers to making underwater images from in-land images using different methods. One might apply a scattering filter
effect\textsuperscript{63}, or make use of color transformations to reach the look of an underwater image. Most notably, Li \textit{et al.} converted real indoor on-land images into underwater-looking images via the use of a GAN (Generative Adversarial Network)\textsuperscript{21}, which sparked an avalanche of GAN-based underwater image enhancement methods.\textsuperscript{18, 22, 23, 64, 66} GANs remain a subject of interest for underwater image enhancement and restoration (UIER) due to the fact that labelled, high quality data is rare in UIER, as discussed above. While such methods can be helpful, there are caveats and challenges to GAN-based synthetic data generation and image enhancement models. GANs in image enhancement are typically finicky in the training process as they are very sensitive to hyper-parameters, a very common issue in GAN-based approaches\textsuperscript{23, 67, 69}. In comparison, CNNs are feed-forward models that are far more controllable in training and testing. Secondly, features of the underwater domain might differ from the features learned and generated by the GAN, which can cause further inaccuracies in the supervised image enhancement models that learn from this generated data. Therefore, the most accurate method of generating synthetic data for underwater scenarios is to use 3D photorealistic simulations that allow for granular control over every variable, can be modelled after many different environments, and allow for diagnostic methods and wider ground-truth annotations\textsuperscript{20}.

**Lack of Standardized Data:** Underwater image enhancement suffers from a lack of high-quality, annotated data. While there have been numerous attempts at gathering underwater images from real environments\textsuperscript{23, 70}, the inconsistencies between image resolution, amount of haze, and imaged objects makes testing and training of deep learning models significantly more challenging. For example, the EUVP dataset\textsuperscript{23} contains small-size images of $256 \times 256$. 
resolution, while the UFO-120 dataset \cite{71} contains 320 × 240 and 640 × 480 images, and the UIEB Dataset \cite{60} contains images of various resolutions, ranging from 330 × 170 up to 2160 × 1440 pixels. This variance between image samples, especially the variance in image quality, haze, and imaged objects is an issue with many learning based systems, both in training and in evaluation.

**Underwater Simulations:** Currently, there are a handful of open-source underwater simulations available. Prior simulations exclusively developed for underwater uses include UWSim (published in 2012) \cite{44} and UUV (Unmanned Underwater Vehicle) Simulator (published in 2016) \cite{45}. These softwares provide tools for simulations of rigid body dynamics or sensors such as sonar or pressure sensors. However, these simulation tools haven’t been recently updated or developed to support modern hardware. More importantly, these simulations do not focus on real-time, realistic image rendering with ray tracing, nor they are designed for modern diagnostic methods such as data ablation \cite{7,9,19}. In contrast, our simulation supports real-time ray-tracing, physics based attenuation and scattering, and allows for dynamic modifications to the structure of the scenes and captured images.

**5.1.2 Methods**

As demonstrated in Figure 5.2, our proposed method separates dehazing from color correction in a two-step process. First, we reconstruct a dehazed image from the input, then feed the resulting dehazed image to a color-transfer model to obtain a final image. As we detail in Section 5.1.3 our dehazing model is trained on 5000 synthetic images that physically model light scattering and attenuation in water. Our model restores pixel information
in areas affected by this attenuation and scattering, and the color model —trained on a subset of the UIEB dataset— matches the color space of the dehazed image with the target domain, finishing the process. By splitting the image enhancement task into dehazing and color transfer, we can effectively train deep learning models, independently control how they process the input image, and quickly update our pipeline to match a new target domain for color transfer without the need to retrain the dehazing model.

In this section, we will go over the various parts of the iDehaze pipeline. Below, we first review the 3D simulation used to gather training images for the Dehazing model.

Figure 5.2: The two-step approach of iDehaze. The input image is first dehazed by a specialized Dehazing model, trained on synthetic data. The resulting colors are then transferred unto a target domain by the Color model. iDehaze can reproduce greater detail from real images compared to the reference images in the UIEB dataset.

Simulation of Underwater Environments  Compared to the land domain, underwater images require significantly more time and effort to gather for any type of visual task. This, in turn, makes preparation of underwater data for machine/deep learning analysis very challenging due to the manual effort required for labeling (supervised) and data cleanup (unsupervised). In addition, these datasets are immutable; it is not possible to modify them after
acquisition. We address this dearth of real data by generating photorealistic data using a 3D simulation environment. We then use this generated data to train our deep neural network to dehaze underwater images. Our simulation is made in Unreal Engine 4 \cite{11} using real-time ray-tracing. We modelled an underwater environment to match the properties of the target domain. In particular, our environment contains dynamic swimming fish, inanimate objects, dynamic aquatic plants, wreckage and boulders. The lighting of the simulation is entirely achieved by real-time global illumination via ray-tracing, this makes our underwater scenes very realistic since objects are shaded and lit realistically for every pixel in the image. We use a global pixel shader for modelling the attributes of light propagation underwater. Specifically, we model the exponential signal decay based off of the Beer-Lambert law \cite{72}, where light is exponentially scattered away depending on the optical depth and attenuation coefficient:

\[
FragColor(r, g, b) = \text{Lerp}\left(\lambda(r, g, b), 1 - \theta(r, g, b), e^{\Delta(r, g, b) \cdot \mu}\right)
\] (5.1)

In Eq. 5.1, $\lambda$ is the ray-traced normal image, $\theta$ is the scattered wavelengths, and $\Delta$ is the pixelwise optical depth in each channel (r,g,b) and $\mu$ is the molar concentration of the dielectric material (water). in beer-lambert law the term $\Delta \cdot \mu$ is called the attenuation coefficient. The \text{Lerp} function interpolates between the scattered image and the ray-traced image rendered at the GPU frame-buffer, using the Beer-Lambert law as the interpolation key. The wavelengths term $\theta$ allows us to control which wavelengths of light are scattered
and which can reach the camera, hence enabling realistic modelling of different types of murky waters (see Fig. 5.3).

Figure 5.3: The manipulation of the attenuation coefficient in the global pixel shader allows for creating a supervised dataset and train a deep learning model to learn to reverse the effects of attenuation and scattering in underwater images. Images with a high attenuation coefficient are hazy—matching the light characteristics under water—while those with a low coefficient resemble images taken on land (ground truth).

**Dehazing vs Color Transfer**  Similar to some earlier studies [73, 74], we split the overall task of image enhancement into two distinct tasks, (1) image dehazing and (2) color transfer. By splitting these two fundamentally different tasks, we are able to train specialized deep learning models to dehaze and transfer the color to a target domain. We note that, due to the lack of accurate measurements at image capture time in the UIEB and many other underwater image enhancement datasets, we usually do not know the ”true” color space of the underwater images. Hence, rather than color correction, we believe that this step is best called *color transfer*. We address this issue in our simulation by including a customizable color checker chart with which to accurately gauge the correctness of color-transfer methods.
We expect that this tool will prove useful for training underwater color correction algorithms in the future.

5.1.3 Experiments

**Neural Networks** We use a version of U-NET \[40\] with a modified final layer to generate a three channeled image. The patch-based approach in U-NET allows for the use of variously sized images when training the neural network. We set the patch size to 384 \times 384 and we slid the patches by 300 pixels to cover the image. To make sure that the model learned both the image structure and to reduce outlier prediction values and image reproduction noise, we used a hybrid loss function that allows controlling the amount of processing for each image:

$$HybridLoss = (\lambda \times (1 - ssim)) + ((1 - \lambda) \times (mse))$$ \hspace{1cm} (5.2)

In Eq. 5.2 \(\lambda\) is a weight hyperparameter that controls how much the model can deviate from the original image, \(ssim\) is the structural similarity index \[75\], and \(mse\) is the mean squared error. Both models were trained with a batch size of 128, and learning rate of 0.001. The color model was trained for 100 epochs, and the dehazing model was trained for 50 epochs. The training procedure for each network took approximately 4 hours on Two Nvidia GTX 1080ti cards.

**Data acquisition** For acquiring the image data, we used a Python module to move and rotate in 6 directions in the 3D space and gather image samples. This module automatically changed the scattering wavelengths in each captured image to cover a wide array of
scattering color patterns. These wavelengths were chosen by analyzing the infinity regions (i.e., where light reaching the camera is completely scattered) of the UIEB Dataset [60] with the eyedropper tool in Photoshop. We gathered 5000 RGB images of varying angles and scattering wavelengths, which we then used to train our dehazing model. These images were split to a 80% ratio for training, 10% validation and 10% for the test set.

**Experimental Setup**  We conducted all our experiments in a Dell Precision 7920R server with two Intel Xeon Silver 4110 CPUs, two GeForce GTX 1080 Ti graphics cards, and 128 GBs of RAM. As noted above, we trained the neural networks for 100 epochs for each task (Dehaze, Color Transfer) and the \( \lambda \) hyperparameter in Eq. 5.2 was set to 0.6 for each neural network. The Dehazing Synthetic Dataset was generated in Unreal Engine 4.26 using a Windows 10 machine with 16GBs of RAM, an RTX 2080ti and AMD Ryzen 5600x CPU.

**Metrics**  We quantitatively evaluated the output of iDehaze using the most common metrics used in prior work [23][73][76][77], namely Underwater Image Quality Measure (UIQM), Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM). The UIQM metric is a non-reference measure that considers three attributes of the final result: 1) UICM — image colorfulness 2) UISM — image sharpness and 3) UIConM — image contrast. Here, each attribute captures one aspect of image degradation due to signal path loss in underwater images. In Eq. 5.3, the UIQM is measured by adding UISM, UICM, and UIConM using a ratio defined by three constants:

\[
UIQM = c_1 \times UICM + c_2 \times UISM + c_3 \times UIConM
\]  

(5.3)
The $c_1$, $c_2$ and $c_3$ constant values are set to the same values as the original publication’s suggested values \[76\] and are the same across the comparisons drawn in section \[5.1.4\] more specifically: $c_1 = 0.0282$, $c_2 = 0.2953$, $c_3 = 3.5773$.

The PSNR metric measures the approximate reconstruction quality of image $x$ compared to ground-truth image $y$ based on the mean squared error ($mse$):

$$PSNR(x, y) = 10 \log_{10} \left[ \frac{255^2}{mse(x, y)} \right]$$  \hspace{1cm} (5.4)

The $SSIM$ metric, on the other hand, compares the image patches based on luminance, contrast and structure. In Eq. \[5.5\] $\mu$ denotes the mean, and $\sigma$ denotes the variance, while $\sigma_{xy}$ denotes the cross-correlation between $x$ and $y$. In addition, the constants $c_1 = (255 \times 0.01)^2$ and $c_2 = (255 \times 0.03)^2$ are present to ensure numeric stability \[23,75\].

$$SSIM(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \cdot \frac{2\sigma_{xy} + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}$$  \hspace{1cm} (5.5)

Datasets  We use a subset of the UIEB dataset \[60\] for training our color model, specifically, we discard low resolution images, and only use images that have at least 384 pixels in each dimension. The UIEB dataset is a set of 890 real underwater images that were captured under different lighting conditions and have diverse color range and contrast. We chose UIEB since its reference images were obtained without synthetic techniques. We reserve 80 images for evaluating the iDehaze pipeline. For benchmarks, we chose the EUVP dataset \[23\] and the UFO-120 dataset \[71\]. EUVP is a large collection of lower resolution
underwater images, manually captured by the authors during oceanic explorations. We evaluate our method on 515 paired test images in the EUVP dataset. Note that due to the low resolution of the EUVP test images (256 × 256), we pad the samples with empty pixel values when feeding them into our pipeline. The UFO-120 dataset is a collection of 1500 640 × 480 underwater images and 120 test samples for evaluation. We use the 120 test samples from the UFO dataset to evaluate and compare iDehaze with other systems.

5.1.4 Results

In this section, we analyse the results of our proposed approach both qualitatively and quantitatively. Table 5.1 shows the performance of the iDehaze system against four state-of-the-art methods on three aforementioned underwater image datasets, while Table 5.2 compares the performance of iDehaze against the most recent GAN-based models on the UFO-120 and EUVP datasets. The UIEB metrics were measured with a separate test-set of 80 randomly chosen images, unseen by the color model in the iDehaze pipeline.

As shown on these tables, iDehaze performs higher than all of the other methods based on the UIQM metric, achieving state-of-the-art performance on the UIEB, UFO-120 and EUVP datasets. Note that our deep neural networks were not trained on each of these three datasets separately. Instead, our dehaze model was solely trained on synthetic data obtained from our UE4 simulation, and our color model was trained only on a subset of the UIEB dataset. As such, the result on the UFO-120 and EUVP prove that (1) our 3D simulation is able to generate realistic data that matches real-world data and (2) that our two-step pipeline is able to learn features that can generalize to a wide variety of data with no additional training. Finally, we note that iDehaze also achieved state-of-the-art SSIM on
the EUVP dataset. In more detail, in Table 5.1, we compare iDehaze to various state-of-the-art models in underwater image enhancement. The WaterNet model [60] was trained on the entire UIEB dataset, the [23] FUnIE-GAN method was trained on the EUVP dataset, the deep SESR was trained on the UFO-120 dataset [71], and Shallow-UWNet used the pre-trained weights of the Deep SESR model and was re-trained on a subset of the EUVP dataset [73]. iDehaze showed superior performance and stability on the EUVP dataset and the UFO-120 dataset despite not being trained on them, and it also outperformed all methods in the UIEB test set in the underwater image quality (UIQM) metric. For the SSIM metric, iDehaze narrowly beat Shallow-UWNet, but it is less stable with a relatively higher standard deviation. Conversely, iDehaze outperformed WaterNet on the UIEB dataset with more accurate SSIM and higher stability.

iDehaze performs relatively poorly on the EUVP and UFO-120 datasets on the PSNR metric. We postulate that such performance drop could be due to the iDehaze pipeline trying to reconstruct every detail present in the image. Since the Dehazing model is trained on clean, noiseless images, some pixel values reconstructed in the real images are noise captured by the underwater camera, irrelevant particulates in the water and compression artifacts (that shouldn’t be reconstructed) and thus hurt the PSNR score. Another reason for the low PSNR value is the presence of visual artifacts in the unseen structures of the images, we explain more about this in Section 5.1.4.

Finally, to see how iDehaze (a CNN-based method) compares to the state-of-the-art GAN based models, we ran evaluations on the UFO-120 and EUVP dataset. As Table 5.2 shows, iDehaze outperformed all GAN-based models in the UIQM metric, and seems to have a higher but less stable SSIM on the EUVP dataset.
Figure 5.4: Qualitative comparison between iDehaze (rightmost image) and RedChannel [78], GDCP [79], Bluriness and Light Absorption (UIBLA) [80], Fusion Based [81], FunIE-GAN Method [23] and UWCNN [82] images were obtained from [74].
Figure 5.5: Sample inputs and outputs from various datasets (UIEB \cite{60}, EUVP \cite{23}, UFO-120 \cite{71}). Features learned from the dehazing simulation transfer well to other domains in image restoration, as iDehaze brings back lost information in scattered areas and dark parts of the images.
Table 5.1: Performance of iDehaze in comparison to WaterNet [60], FUnIE-GAN [23], Deep SESR [64], Shallow-UWNet [73]. Metrics: Peak Signal-to-Noise Ratio (PSNR), Structural similarity index (SSIM), Underwater Image Quality Measure (UIQM) - higher is better

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>WaterNet</th>
<th>FUnIE-GAN</th>
<th>Deep SESR</th>
<th>Shallow-UWNet</th>
<th>iDehaze (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR ↑</td>
<td>19.11 ± 3.68</td>
<td>19.13 ± 3.91</td>
<td><strong>19.26 ± 3.56</strong></td>
<td>18.99 ± 3.60</td>
<td>17.96 ± 2.79</td>
</tr>
<tr>
<td>UIEB</td>
<td>SSIM ↑</td>
<td>0.79 ± 0.09</td>
<td>0.73 ± 0.11</td>
<td>0.73 ± 0.11</td>
<td>0.67 ± 0.13</td>
<td>0.80 ± 0.07</td>
</tr>
<tr>
<td></td>
<td>UIQM ↑</td>
<td>3.02 ± 0.34</td>
<td>2.99 ± 0.39</td>
<td>2.95 ± 0.39</td>
<td>2.77 ± 0.43</td>
<td><strong>3.28 ± 0.33</strong></td>
</tr>
<tr>
<td></td>
<td>PSNR ↑</td>
<td>24.43 ± 4.64</td>
<td>26.19 ± 2.87</td>
<td>25.30 ± 2.63</td>
<td><strong>27.39 ± 2.70</strong></td>
<td>23.01 ± 1.97</td>
</tr>
<tr>
<td>EUVP</td>
<td>SSIM ↑</td>
<td>0.82 ± 0.08</td>
<td>0.82 ± 0.08</td>
<td>0.81 ± 0.07</td>
<td>0.83 ± 0.07</td>
<td><strong>0.84 ± 0.09</strong></td>
</tr>
<tr>
<td></td>
<td>UIQM ↑</td>
<td>2.97 ± 0.32</td>
<td>2.84 ± 0.46</td>
<td>2.95 ± 0.32</td>
<td>2.98 ± 0.38</td>
<td><strong>3.11 ± 0.36</strong></td>
</tr>
<tr>
<td></td>
<td>PSNR ↑</td>
<td>23.12 ± 3.31</td>
<td>24.72 ± 2.57</td>
<td><strong>26.46 ± 3.13</strong></td>
<td>25.20 ± 2.88</td>
<td>17.55 ± 1.86</td>
</tr>
<tr>
<td>UFO-120</td>
<td>SSIM ↑</td>
<td>0.73 ± 0.07</td>
<td>0.74 ± 0.06</td>
<td><strong>0.78 ± 0.07</strong></td>
<td>0.73 ± 0.07</td>
<td>0.72 ± 0.07</td>
</tr>
<tr>
<td></td>
<td>UIQM ↑</td>
<td>2.94 ± 0.38</td>
<td>2.88 ± 0.41</td>
<td>2.98 ± 0.37</td>
<td>2.85 ± 0.37</td>
<td><strong>3.29 ± 0.26</strong></td>
</tr>
</tbody>
</table>
Table 5.2: Quantitative analysis on the EUVP and UFO-120 across various GAN-based models. Best values are highlighted in bold. Sample outputs from the mentioned models are found in figures 5.6 and 5.7.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>Deep SESR</th>
<th>FUnIE-GAN</th>
<th>FUnIE-GAN-UP</th>
<th>UGAN</th>
<th>UGAN-P</th>
<th>iDehaze (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR↑</td>
<td>25.30 ± 2.63</td>
<td>26.19 ± 2.87</td>
<td>25.21 ± 2.75</td>
<td>26.53 ± 3.14</td>
<td><strong>26.53 ± 2.96</strong></td>
<td>23.01 ± 1.97</td>
</tr>
<tr>
<td>EUVP</td>
<td>SSIM↑</td>
<td>0.81 ± 0.07</td>
<td>0.82 ± 0.08</td>
<td>0.78 ± 0.06</td>
<td>0.80 ± 0.05</td>
<td>0.80 ± 0.05</td>
<td><strong>0.84 ± 0.09</strong></td>
</tr>
<tr>
<td></td>
<td>UIQM↑</td>
<td>2.95 ± 0.32</td>
<td>2.84 ± 0.46</td>
<td>2.93 ± 0.45</td>
<td>2.89 ± 0.43</td>
<td>2.93 ± 0.41</td>
<td><strong>3.11 ± 0.36</strong></td>
</tr>
<tr>
<td></td>
<td>PSNR↑</td>
<td><strong>26.46 ± 3.13</strong></td>
<td>24.72 ± 2.57</td>
<td>23.29 ± 2.53</td>
<td>24.23 ± 2.96</td>
<td>24.11 ± 2.85</td>
<td>17.55 ± 1.86</td>
</tr>
<tr>
<td>UFO-120</td>
<td>SSIM↑</td>
<td><strong>0.78 ± 0.07</strong></td>
<td>0.74 ± 0.06</td>
<td>0.67 ± 0.07</td>
<td>0.69 ± 0.07</td>
<td>0.69 ± 0.07</td>
<td>0.72 ± 0.07</td>
</tr>
<tr>
<td></td>
<td>UIQM↑</td>
<td>2.98 ± 0.37</td>
<td>2.88 ± 0.41</td>
<td>2.60 ± 0.45</td>
<td>2.54 ± 0.45</td>
<td>2.59 ± 0.43</td>
<td><strong>3.29 ± 0.26</strong></td>
</tr>
</tbody>
</table>
Figure 5.6: Qualitative analysis on the EUVP dataset [23]. Other methods include UGAN and UGAN with gradient penalty (UGAN-P) [23], Deep SESR [64] and FUnIE-GAN [23].
Discussion
Below, we discuss the main takeaways of our experimental results.

**Image Artifacts:** As shown above, iDehaze can reproduce detail and color in datasets never-before encountered in its training. However, iDehaze seems to be sensitive to previously unencountered structures in the image. For example in the EUVP dataset, there are novel structures that aren’t present in our synthetic dataset, therefore the dehazing model introduces image artifacts to those areas. Image artifacts are generally more present in higher resolution images, as well as new datasets unseen by the models. However, we believe...
the presence of artifacts can be easily remedied by training the model for a few epochs on the new data.

**Strengths and Weaknesses of the Patch-Based Approach:** U-NET, the CNN used in the iDehaze pipeline typically breaks down images into patches and reconstructs them together at output time. Because of this, iDehaze is not sensitive to input image size and can accept various sizes and image qualities as input, an important feature to have when the availability of high-resolution, labelled real underwater images is limited. However, stitching the image patches together can create a patchwork texture in some images, which appears from time-to-time in iDehaze image outputs. This can be fixed with the use of bigger patch sizes, or a potential post-processing pass that aims to clear out the stitching marks without introducing blur to the final image.

**On the use of Compressed Images:** A frustrating fact about the available image datasets in the UIER field is the use of compressed image formats. More specifically, the JPG and JPEG file formats use lossy compression to save disk space. Image compression can introduce artifacts that while invisible to the human eye, can and will have effects on a Neural Network's performance. Hopefully, as newer and more sophisticated image datasets are gathered in the UIER field, the use presence of compressed images will eventually fade away. To take a step in the correct direction, the iDehaze synthetic dataset uses lossless 32-bit PNG images, and will be available freely for public use.
**Future Work**  In many cases, the two-step approach of iDehaze can work in tandem to bring out details and lost information in underwater images, particularly in images that suffer from significant scattering, images that cover a wide optic depth, and images that have a relatively uniform optic depth. However, in some cases the full iDehaze pipeline might qualitatively look "over-processed" to the human eye. In such cases, the color model alone produces a more aesthetically pleasing result. One interesting future direction is devising an automatic system that could choose between the output of the color model or the output of the full pipeline based on psychometric criteria.

**5.1.5 Contributions**

In this paper, we presented iDehaze, a state-of-the-art image dehazing and color transfer pipeline. Our proposed system includes a 3D simulation toolkit capable of generating millions of customizable, unique photorealistic underwater images with physics-based scattering and attenuation enabled by real-time ray-tracing. In our pipeline, we break down the larger task of underwater image enhancement into two steps: dehazing and color transfer. Our experiments demonstrate that iDehaze is capable of reconstructing clear images from raw, hazy inputs, achieving a state-of-the-art SSIM score on the EUVP dataset and state-of-the-art UIQM scores for the UIEB, UFO-120 and EUVP datasets *despite not being trained on the latter two datasets at all*. These results showcase the strengths of a carefully curated, physically modelled synthetic dataset made by 3D digital content creation tools. Our synthetic dataset, benchmarks and code will be released upon publication. We believe the image enhancement research community will greatly benefit from the availability of free, high quality, high resolution labelled training data for underwater image dehazing and restoration tasks.
Throughout this dissertation, we explored the applicability of photorealistic simulations in a variety of computer vision tasks. More specifically, we saw its applications in indoor depth estimation, semantic segmentation and surface normal prediction. We then demonstrated the strengths of simulations in more nuanced applications, such as depth completion and semantic segmentation of transparent objects via exclusive training in photorealistic simulations. We saw the benefits of flexible simulations in extreme environments and data-deprived tasks in Underwater Computer visions.

Our studies span all phases of synthetic data adoption, from proof of concept in exploring the bounds of features that can be learned from simulated data, improving existing methods by introducing flexibility enabled by real-time graphics and data-ablation methods, applications on difficult underwater computer vision tasks, and achieving state-of-the-art performance by training with majority synthetic data in underwater image enhancement and dehazing.

Throughout our studies, the fourth phase of synthetic data adoption has been achieved by a number of researchers in various vision related tasks. In our case, we were able to achieve state-of-the-art performance in Underwater Image Enhancement and Restoration, which further cements the strengths of flexible photorealistic simulations. However, since
vision tasks can span a wide variety of domains, there are still fields where applications of synthetic data will fall in phase two category.

An important and overlooked property of synthetic data is its flexibility for meaningful change even after data acquisition. Using data ablation we can isolate and change every aspect of a flexible simulation. This will allow for fast iteration during testing and evaluation and allow for generation of more data to amend a training dataset depending on the performance of a given deep learning architecture on the target domain. This flexibility will be compounded by using real-time solutions for rendering the scenes, and generating the synthetic datasets. Modern graphics processing units (GPUs) allow for real-time ray-tracing, that enables immense fidelity in a much faster time than traditional rendering techniques.

When the author of this dissertation started his work on synthetic data and use of simulations in lieu of real data, this field technically did not exist as an explored field with a robust body of research. The hope is the contents of this dissertation will be of small help in the work of those who tread this path in the future.
REFERENCES


[78] A. Galdran, D. Pardo, A. Picón, and A. Alvarez-Gila, “Automatic red-channel under-


