MACQUARIE UNIVERSITY

Development of Interactive Virtual Environments Using AI-Enhanced Photogrammetry

Benjamin Yee (45425108)

Supervisor: Dr David Payne

Bachelor of Engineering, Honours in Software

6th June 2024

Acknowledgements

I would like to thank Dr David Payne for his mentorship and guidance throughout this thesis project. His insight and advice have been invaluable to the creation of this document.

I'd also like to thank my family for their support throughout this project, and for putting up with me when the conversations go off-topic.

Statement of Work

I, Benjamin Yee, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the School of Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment an any academic institution.

I also declare that AI tools were not used in the writing of this paper.

Student's Name: Benjamin Yee

Student's Signature: Benjamin Yee (electronic) Date: 2/6/24

Abstract

The traditional method of teaching in a student/mentor capacity has not changed for hundreds of years. This method has merit for direct instruction, a structured learning environment, and basic memorisation. However, there are often scenarios where traditional methods face limitations - for example, a student can be shown how to react in a dangerous workplace situation, but *practising* how to react can be limited in a classroom environment in the absence of perceived danger. Advancements in technology over the past few decades have opened up newer and better ways of delivering content, especially in the digital space. The development of new technology to aid in interactivity and delivery of content can be used to complement existing teaching methods to improve the retention of knowledge and increase the safety of the learning environment of students.

Virtual Reality (VR) technology is quickly being recognised as a valuable tool for Workplace Health and Safety (WHS) training due to its ability to provide true-to-life and immersive simulations of real-life environments. Interactive and hands-on experiences with virtual objects and equipment, as well as the ability to receive instant feedback, can strengthen the learning process and retention of knowledge. However, the process of creating a realistic virtual environment is often very complex. While many tools exist to facilitate the process, creating a digital environment that closely simulates real-world scenarios can be expensive, time consuming, and can require the expertise of skilled professionals.

This solution, hereafter referred to as the DIVE-AIP pipeline, will explore a novel method of quickly creating virtual environments that simulates real-world scenarios from a single image. In this process, we are able to create a 3D scene that not only matches the scale of the source scene, but also produces independent placeholder assets that represent objects in the source scene. The DIVE-AIP pipeline aims to fast-track the development of virtual environments by reducing the time spent in developing 3D assets.

Contents

List of Figures

List of Tables

1 Introduction

Virtual reality (VR), where users are immersed in a computer-generated simulation, has become increasingly popular in recent years. As technology has improved, advancements in computing power and graphics processing have allowed for the creation of more realistic virtual environments that enhance user immersivity, allowing for more life-like experiences while removing exposure to real-world risks. As a result of these developments, VR technology has been implemented in various industries including gaming, entertainment, architecture design, hospitality, and others. The ability to simulate presence and interactivity while minimising risk makes VR an attractive solution for businesses looking to provide experiences that cannot be replicated through traditional methods [14].

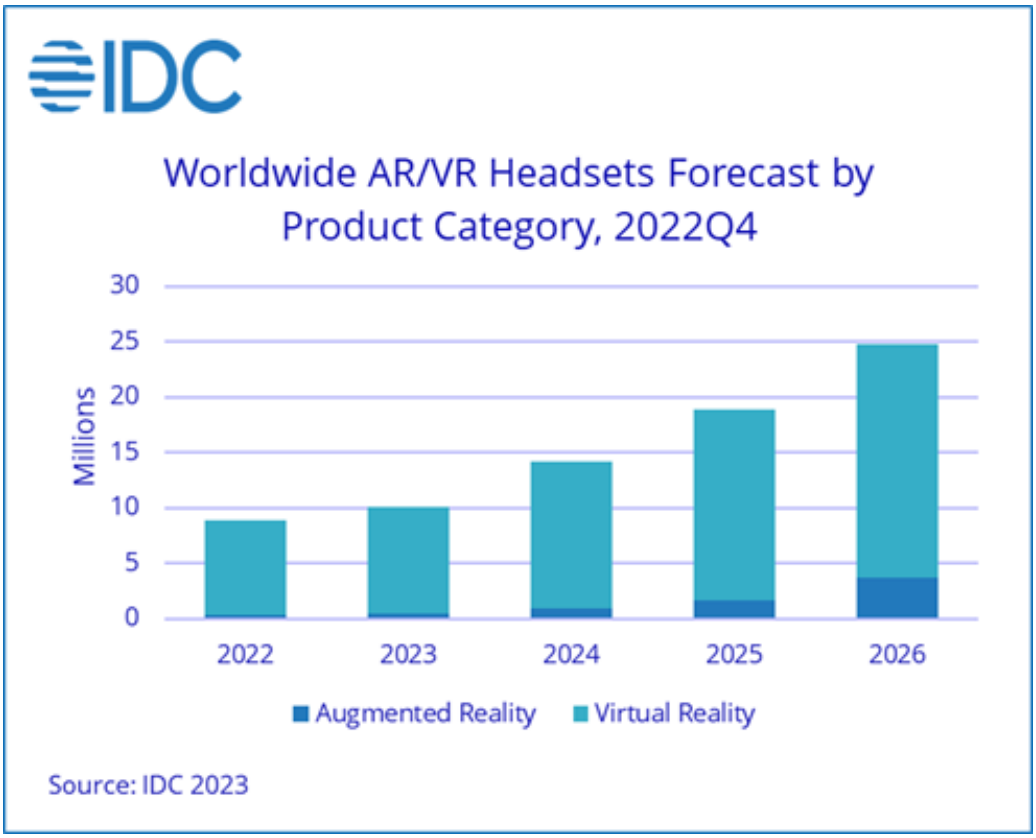

Figure 1: Worldwide shipments forecast of VR/AR headsets, Source: [1]

Workplace Health and Safety (WHS) training is an extremely important aspect in maintaining a safe work environment and in ensuring the well-being of employees. Training workers to accurately identify, prevent, and respond to potential hazards decreases accident rates, injuries, and fatalities. Traditionally, WHS training has been delivered through various methods

including: lectures, videos, printed materials, and classroom-based sessions with instructors or trainers. While these methods are tried and proven, they can sometimes lack the hands-on, practical experiences that are crucial for retention and effective learning, especially in high-risk scenarios.

VR technology has emerged as a valuable tool for WHS training due to the ability to allow for hands-on and interactive training experiences. The advantage of being able to recreate high-risk or dangerous situations in a safe and controlled virtual setting allows trainees to practise their responses to potential hazards while minimising real-world risk. Using VR, trainees are able to receive immediate feedback on their actions, which can aid the learning process and enhance the retention of taught content. Digital VR scenarios also allow for repeated practice sessions, so trainees are able to practise their tasks as many times as needed until they are confident, without fear of negative consequences. In this way, trained workers are able to be better prepared to handle real-world situations safely.

Advancements in computational power over the past decade have significantly contributed to growth in the machine learning (ML) and artificial intelligence (AI) fields. The development of faster and larger Graphical Processing Units (GPUs) have provided researchers with enough processing capability to train larger and more complex machine learning models. This acceleration in development has been followed by a culture of collaboration and knowledge sharing, and so many open-source projects exist that provide frameworks and tools that can be used by anyone. Machine learning applications range from text generation and sequence-to-sequence translation to object recognition and depth estimation, and are quickly becoming adopted by many fields of research.

1.1 Problem Statement

The integration of virtual reality into educational settings creates an environment that is more conducive to learning - immersing students in realistic training scenarios enhances the understanding and retention of knowledge. The ability to create safe and controlled virtual environments with minimal risk or real-world consequences allows for more complex learning experiences that would otherwise be difficult or impossible to recreate in a traditional classroom setting. Through VR students are able to actively participate in hands-on experiences that allow them to apply theoretical knowledge to practical situations, which aids in preparing students for real-life scenarios. In addition, studies suggest that students training with VR achieve better pass rates than those using traditional training methods [15].

The development of realistic virtual environments that are comparable to real-world scenarios usually demand significant resources. The development process can be a time-consuming process that involves several stages including conceptualization, prototyping, testing, and refining. The creation of high quality content, such as 3D models, textures, lighting, and animation usually requires the expertise of skilled professionals who possess the necessary technical knowledge and creativity to accurately recreate real-world scenarios in the digital space.

The process of the development stage of building a virtual environment to be used as an educational tool generally follows this structure (assuming the scope of the project has already been decided):

1. Conceptualisation of Environment

(a) Planning of the virtual environment that will be presented to the player involves the consideration of key elements - this involves deciding on the assets that the player will be able to interact with, and deciding how the target learning concepts can be conveyed

2. Gathering Reference Materials

   (a) Collecting reference materials of the real-world scenario to be virtualized can involve compiling data from photos, videos, architectural plans, or other visual references to build up a dataset that developers can draw upon during the creation of assets

3. Creation of Assets

   (a) The process of creating elements to be used in the 3D space. 3D modelling is used to create digital representations of physical objects, such as buildings, terrain, and other interactive objects where scale and proportion are of high importance. Other assets include audio, shaders and textures, and realistic lighting to best match the target environment.

4. Programming

   (a) Once an asset has been created, each element in the virtual environment must be programmed to correctly interact with the player and other objects. Programming usually involves scripting and the development of simulations to best emulate real-world scenarios.

The high amount of resources required throughout this traditional development process means that it is generally unsuited for small-scale projects or for rapid prototypes.

1.2 Project Goal

The aim of the DIVE-AIP pipeline is to develop an approach for the rapid creation of interactive virtual environments, while leveraging the power of machine learning models. The main goal is to improve the efficiency of using VR as an educational tool by reducing the time associated with the development of a virtual environment, specifically by combining and streamlining the *Conceptualise environment* to *Creation of assets* steps as outlined in the Project Statement. This paper aims to integrate ML models and utilise existing frameworks in order to create a pipeline to expedite this process.

2 Background Research

This Background Research section provides historical backgrounds on the development of Virtual Reality and Artificial Intelligence. This section will explore the historical development, key milestones, influential figures, and relevant events that have led up to the architectures that are available today.

2.1 Virtual Reality: Improving on Immersion

2.1.1 Historical Milestones in the Development of VR

Attempts at creating immersive environments are evident from as early as the nineteenth century, with 360-degree murals intended to fill the viewer's field of vision. These artworks often weighed several tons [16] and took many months and sometimes years to complete. Intended to be viewed by the masses, the panoramas usually required the construction of specialised exhibition buildings to accommodate for the size of the artwork [16].

In the 1830s, researcher Charles Wheatsone began investigations into how the brain processed three-dimensional images. His work focused on how the use of two distinct two-dimensional images could be used to perceive depth and spatial awareness, and led to the development of the Stereoscope: a device that presented viewers with side-by-side images captured at slightly different viewpoints, and the viewer's brains would interpret this as a unified three-dimensional scene. Modern head-mounted virtual reality devices leverage the same fundamental concept of presenting distinct images to each eye in order to simulate depth and increase immersion [17].

Morton Heilig's Sensorama, developed in the 1950s, is regarded as one of the earliest examples of virtual reality. The Sensorama provided a multi-sensory experience through three-dimensional colour film, sounds, smell, and a wind machine, but did not include user interaction with the provided films [18]. Heilig would then go on to design the Telesphere Mask, which was the first example of a head-mounted display (HMD) with stereoscopic vision and stereo sound. However, the Telesphere Mask did not have motion tracking [19].

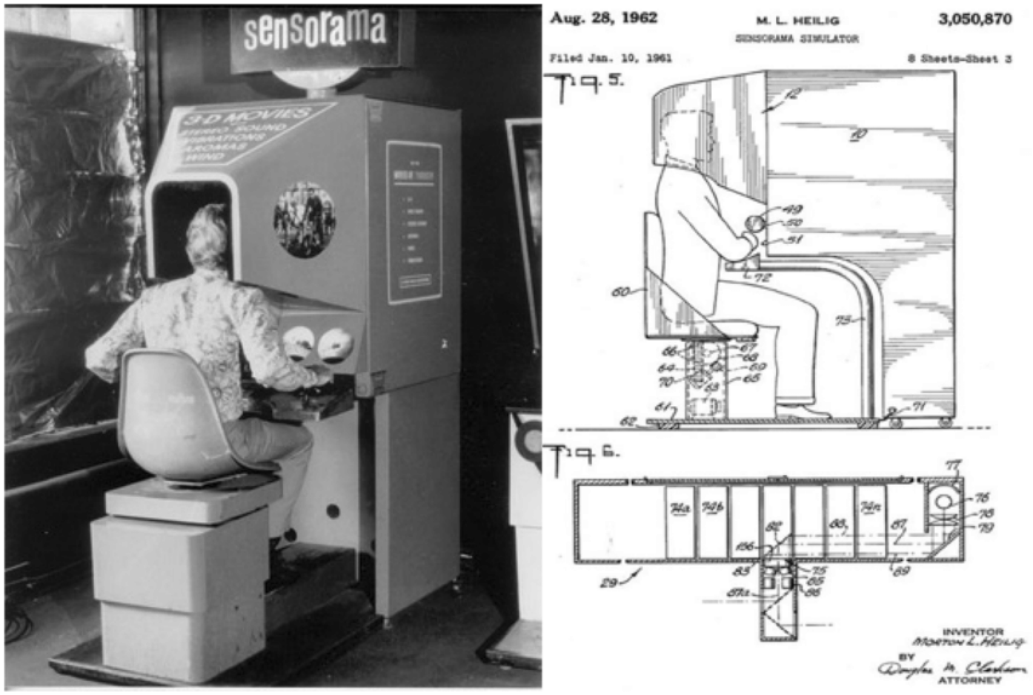

Figure 2: Heilig's Sensorama (left), and the 1962 patent (right), Source: [2]

The Sword of Damocles system, developed by Ivan Sutherland in the 1960s, is widely considered to be one of the earliest "true" head-mounted virtual reality systems. Named after the ancient Roman parable where a sword was suspended above a throne by a horsehair, the Sword of Damocles system needed to be suspended from the ceiling due to its weight. Sutherland's system was different to other movement tracking systems at the time in that it eschewed external sensors (cameras) in favour of internal mechanisms, such as gyroscopes and accelerometers. When tethered to the user via cables and a robotic arm, the Sword of Damocles allowed for better immersion and realism to the user by allowing for dynamic changes in the user's field of view when they moved. [20].

Figure 3: Sutherland's boom-mounted "Sword of Damocles", Source: [3]

The Stereoscope, Sensorama, and The Sword of Damocles represent important milestones in the evolution of modern head-mounted virtual reality systems. The Stereoscope introduced the concept of stereoscopic vision by presenting viewers with paired images to simulate depth, Heilig's Sensorama integrated stereo audio alongside visual stimuli to enhance the immersive experience, and The Sword of Damocles introduced the concept of movement and position tracking to allow users to interact with the virtual environment. These features have formed the basis of the technologies used in modern VR systems.

2.1.2 VR For the Masses: Oculus Rift and Others

The modern age of VR has mainly been heralded by the diffusion of VR technology into the public market and into private households. This trend began with the introduction of the Oculus Rift in 2012, where the Kickstarter campaign raised 974% of the stated goal [21]. The Oculus Rift offered better immersion than other existing HMDs at the time, including a wider Field of View (FOV), lower latency in head tracking via the use of gyroscopes, accelerometers, and magnetometers, and a much lighter weight at around 450g [21]. The Oculus Rift was quickly followed by other commercial HMDs, including the HTC VIVE, PlayStation VR, Samsung's Gear VR, and Google Cardboard [22].

Advancements in computing power have seen the release of standalone systems including the Oculus Quest and HTC VIVE Focus, which no longer need to be connected to a separate computer to process and render 3D environments [22]. Current research includes the development of haptic systems, such as VR gloves, treadmills, and suits to further enhance user immersion [23].

2.2 3D Graphics: From Lines to Landscapes

Early research into 3D graphics can be traced back to the 1960s and 1970s, where researchers focused on wireframe models - representing 3D models using lines and vertices. This technique laid the foundation for the visualisation of complex geometries, and primarily focused on representing objects from engineering and industrial design fields. As models became more complex, research focused on hidden line removal algorithms such as Painter's and Z-buffer [24] to determine which lines were visible to the viewer, and which lines were supposed to be obscured by other parts of the model. The implementation of these techniques improved the visual understanding of 3D models.

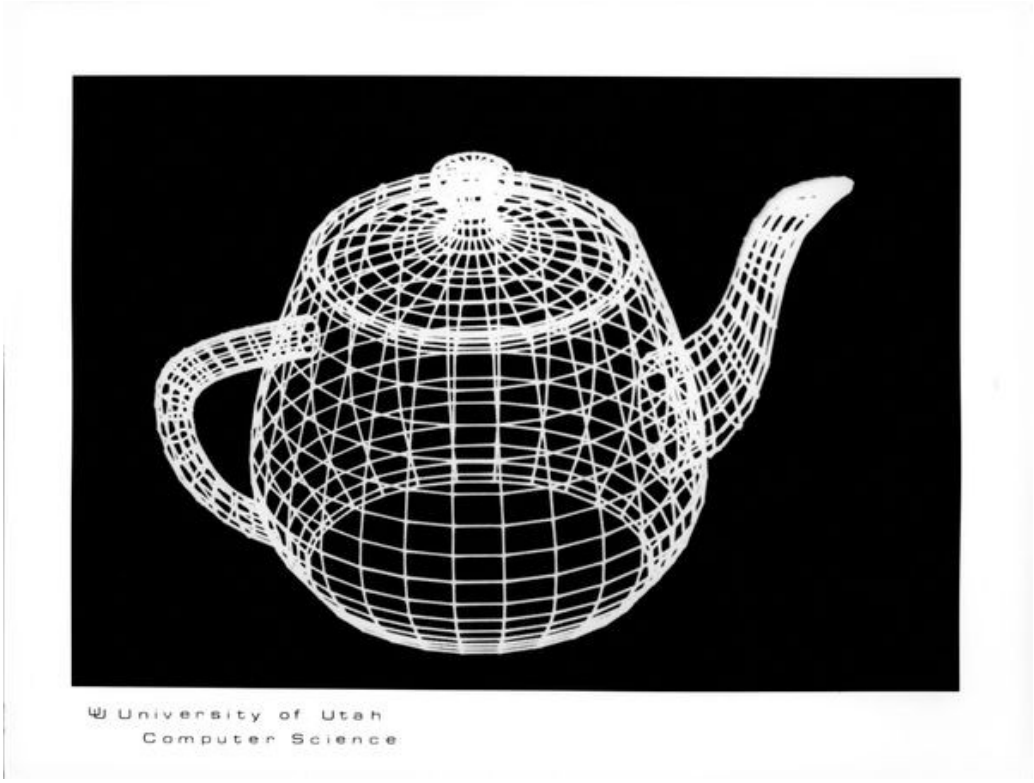

Figure 4: Wire-frame model of the Utah Teapot, Source: [4]

Research quickly moved into adding surface shading to the wireframe models in the early 1970s. Shading techniques such as Phong and Gourand allowed for the simulated appearances of smooth surfaces by visibly replacing the sharp creases between polygons with continuous changes in tone or colour [25]. Gourand and Phong are still widely used in modern graphics systems [26], and almost all current graphics architectures support Gouraud shading [26].

Research into ray tracing in the 1980s enabled more realistic lighting and reflections by tracing the path of light rays throughout a 3D scene. Ray tracing algorithms enabled the simulation of shadows, reflections, and refraction by calculating the interactions between rays of light and objects in the scene, and the resulting intensity of each pixel in an image [27]. However, ray tracing was computationally expensive, and optimisations such as bounding volume hierarchies and spatial partitioning techniques were developed to accelerate the ray tracing process.

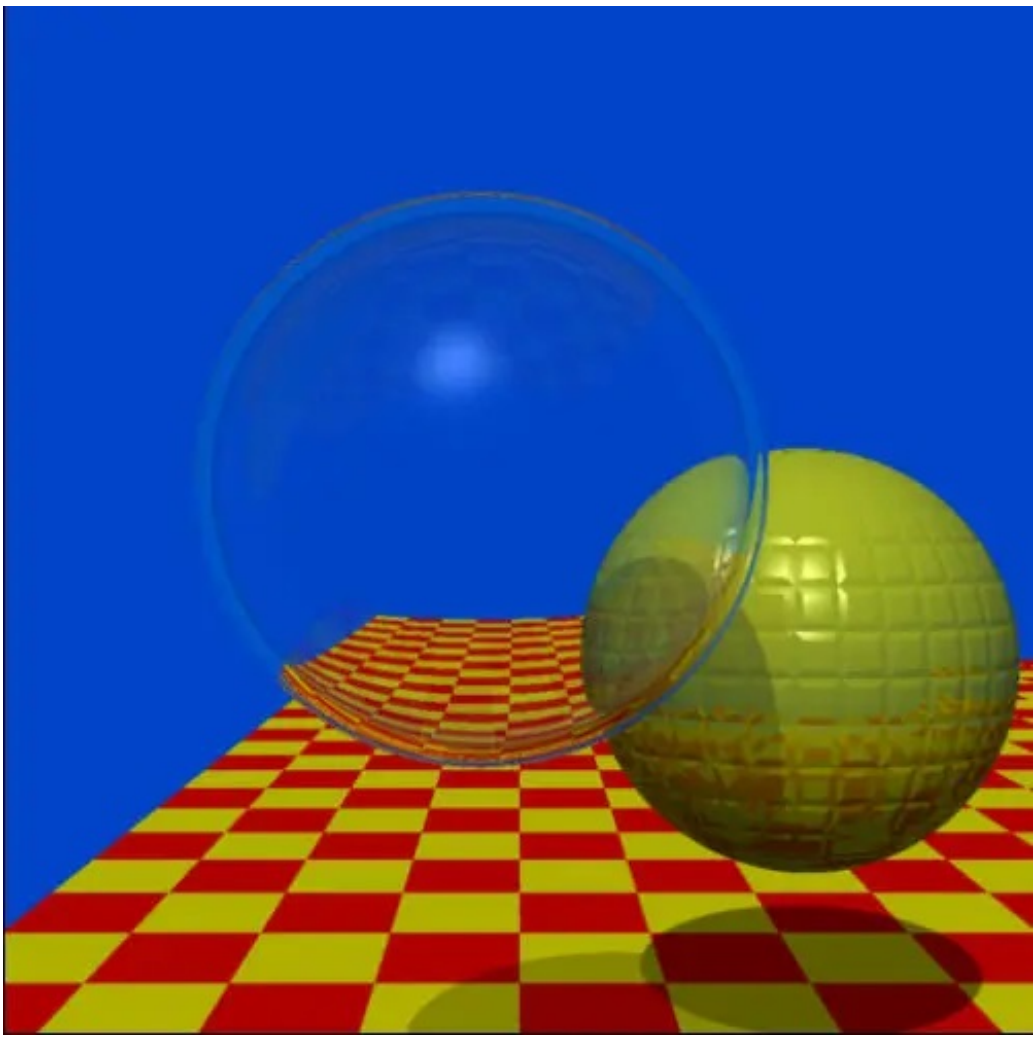

Figure 5: Whitted's image of inter-reflecting spheres to demonstrate global illumination via ray tracing, Source: [5]

The introduction of texture mapping and bump maps were the next steps in enhancing the visual realism of 3D-rendered objects. The development of procedural texture generation in the 1980s to mathematically create complex patterns, such as wood grain, marble, or clouds, and image-based texturing in the 1990s allowed for more realistic representations of surface details and improved the visual quality of rendered objects. The introduction of bump mapping in the 1980s allowed for the simulation of surface details without the need to alter existing geometry, which both improved realism and also increased the efficiency and performance of 3D rendering engines.

Advancements in hardware, parallel computing, and Graphics Processing Units (GPUs) have had significant impacts on the development of realistic 3D environments - real-time ray tracing and the ability to render scenes with dynamic lighting and shadows are now accessible to the general public and allow for the creation of more complex, more realistic 3D environments. Advancements in this field have been vastly contributed to by the rise in popularity and accessibility of game engines.

2.3 VR and Game Engines: Simulated 3D Environments

Modern game engines are best suited for developing VR environments. These engines provide a framework with toolsets for visual scripting, physics simulations, animation editors and more to streamline the process for developers to create interactive VR experiences. Most modern game engines provide VR-specific features, plugins, and optimizations to simplify the process of integrating VR, such as foveated rendering or dynamic resolution scaling to maximise performance on the platform.

2.3.1 Assets

Assets are a vital component in VR environments. These can include the 3D models and animations, textures, and audio files used to contribute to the immersiveness and realisticness of the environment presented to the player. Game engines provide asset management systems that allow developers to import and organise assets within the development environment.

2.3.2 Rendering

The creation of visual imagery and graphics in real-time is a fundamental aspect in 3D environments. Rendering takes into account material properties, light sources, environmental reflections, textures, and shaders in order to create a visual environment. Techniques such as physically-based rendering (PBR) to simulate real-world lighting conditions, hardware-accelerated graphics via dedicated GPUs, advancements in display hardware refresh rates, and accurate stereoscopic rendering are all crucial components in creating a convincing, realistic, and immersive virtual environment.

2.3.3 Physics

Realistic physics in a virtual environment are a fundamental aspect in improving the realism and immersion of VR. Modern game engines are capable of complex physics interactions, including gravity and inertia, cloth and fluid simulations, and realistic animations. Other techniques include collision and rigid body dynamics, constraint solving, and dynamic object behaviours. Modern game engines utilise dedicated physics engines such as Havok and PhysX in order to replicate the physical properties and behaviours of objects in the digital space; however, approximations and optimisations are often used to ease the computational strain of perfectly realistic simulations.

2.3.4 Audio

Audio plays a major role in enriching the sense of presence within a virtual environment by shifting the player's auditory awareness from their physical environment to the digital environment. Spatial audio and simulating how sound behaves in the real world enhances immersion by assisting players in perceiving depth and direction. Audio also serves as a vital feedback mechanism by providing players with sensory cues in the absence of physical touch.

2.3.5 Input and User Interaction

User input and interaction are fundamental components of interactive virtual environments. Allowing players to make decisions and perform actions provides a sense of control and agency, which deepens immersion. The advent of VR has led to the development of dedicated VR controllers that provide hand tracking and motion sensing capabilities, which allow users to manipulate virtual objects using more natural gestures than can be achieved on a gamepad or using a mouse. Modern game engines have integrated support for these controllers and allow developers to map user gestures and actions to environment events.

2.4 Artificial Intelligence: Software to Think Like A Human

Research into Artificial Intelligence, and in creating a machine able to perform human tasks, can be traced back as early as the 1940s: Alan Turing's The Bombe was able to break the Enigma code and is generally considered to be the first working electro-mechanical computer [28]. Turing 's work and paper on "Computing Machinery and Intelligence" gave rise to the notion of a machine that can "learn from experience", where "[t]he possibility of letting the machine alter its own instructions provides the mechanism for this" [29].

Early machine learning programs in the 50s and 60s were limited by the size and speed of memory processors, and were largely based on symbolic manipulation languages such as List, IPL, and POP [30]. Nevertheless, these still led to the creation of significant milestones, such as Arthur Samuel's checkers-playing program (1956), an example of a program improving its own ability through experience [30];the Perceptron (1957), a neural network model capable of recognizing the letters of an alphabet using analog and discrete signals [31]; and ELIZA (1964), a natural language processing tool capable of conversation with a human [28].

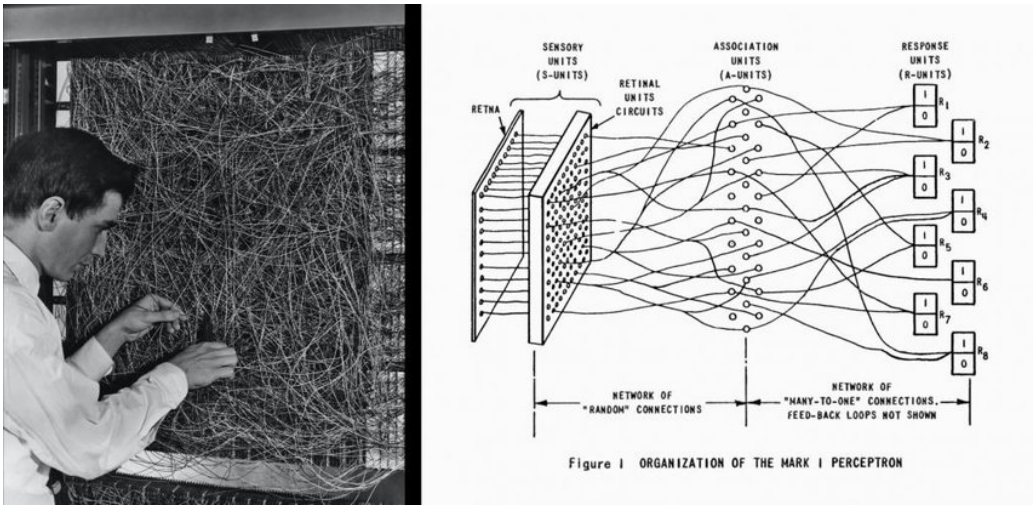

Figure 6: Frank Rosenblatt's Mark I Perceptron (left), and a graphical representation (right), Source: [6]

Despite advancements in the field, progress in AI stagnated from the late 1960s to the 1980s [31]. ELIZA, the Perceptron, and other systems of the time were Expert Systems, that is, the architecture was based on the assumption that human intelligence could be formalised

and reconstructed logically [28]. While Expert Systems performed proficiently in their field of specialisation, they performed poorly outside of that specialisation; work by M.Minsky and S.Papert emphasised that Expert Systems were not able to realise the logical relationship of some logical functions like XOR or NXOR [31][32] .

Research and development of AI received a jumpstart in the 2010s, and can be explained by three synchronous trends: the concept of Big Data (that is, datasets that are too large or complex to be handled by traditional data-processing techniques), reduction in costs of parallel computing and memory, and the development of deep machine learning algorithms [31]. Deep machine learning, coupled with artificial neural networks, form the basis of most AI applications around today.

3 Existing Research

This Related Work section will explore existing research and development in the fields of VR and game engines, and AI object recognition and depth estimation. This section will provide an overview on current methodologies and applications within these domains.

3.1 The COLLADA Format

The Collaborative Design Activity (COLLADA) filetype, developed by the Khronos Group, was developed with the intention of interoperability and accessibility for 3D content creation. COLLADA was designed as an open standard in order to allow for the exchange of 3D assets across different software platforms, in order to free developers from constraints imposed by proprietary file formats. COLLADA's Digital Asset Exchange (DAE) files serve as containers that capture models, textures, and animations in a single file. This approach both simplified asset management and offered a human-readable file format, as DAE files are XML-based. Currently COLLADA has been adopted by many industry-leading 3D tools, including Blender, Unity, and Autodesk Maya.

The inherent features of the COLLADA format have made it an ideal candidate for integration in the DIVE-AIP pipeline. Firstly, COLLADA's open-source status allows developers to freely access, modify, and extend the format of DAE files. The human-readable format of DAE files means that it can be parsed and processed by natural language processing models, and lastly, COLLADA's compatibility with many common 3D platforms means that the resulting generated content can be used without fear of compatibility issues.

3.2 AI Technology

3.2.1 Object Recognition

Overview of Object Recognition

Research into object detection and computer vision can be divided into two main periods. Prior to 2014, research predominantly focused on conventional detection methods, such as Histogram of Oriented Gradients (HOG) and Deformable Part Models (DPM). These approaches, while mostly effective, were limited by their reliance on handcrafted features devised by research teams, and were generally not able to adapt to diverse, real-world scenarios. Post-2014, researchers began shifting towards the use of deep neural networks (DNNs) and convolutional neural networks (CNNs). The ability to extract features or patterns from raw input data, with-

out explicit programming or guidance, allowed for significant growth in object detection tasks, and has since expanded to include object classification, detection, and image segmentation [33].

Convolutional Neural Networks

Convolutional Neural Networks form the backbone of many object detection algorithms. CNNs typically consist of several interconnected "layers", where operations are performed on data in order to recognise salient features from an input image. These frameworks operate within a hierarchical manner, where early layers can be used to detect basic features like edges or corners, and deeper layers combine features from earlier layers to detect more complex features, such as shapes or complete objects. CNNs generally comprise of several layers, including an input layer, a convolutional layer where the image is split into smaller fields in order to detect specific patterns, a Rectified Linear Unit (ReLU) layer that allows the CNN to operate on non-linear data (i.e. an image), a pooling layer that condenses detected feature maps while preserving essential features, and a fully connected layer that allows the network to generate a prediction based on previously extracted features [7].

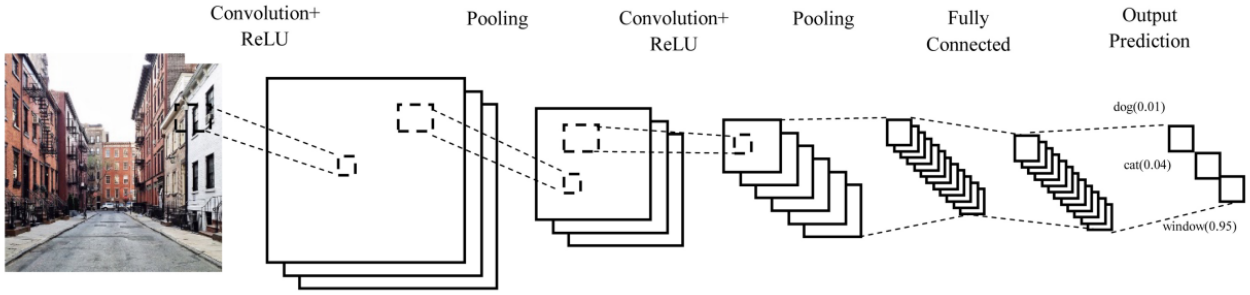

Figure 7: Architecture of a Convolutional Neural Network, Source: [7]

3.2.2 Object Detection Algorithm Types

Two-Stage Algorithms

Two-stage detection algorithms, also known as region-based detectors, detect objects through a hierarchical process of feature extraction and refinement. In particular, anchor-based two-stage detectors use a Region Proposal Network (RPN), which divides the input image into distinct areas of interest. These areas are then used as candidates for subsequent object detection processes across further layers, by placing anchor boxes (bounding boxes) around each area in the source image. These anchor boxes are progressively refined by each layer in the RPN, which eventually generates a region of interest (RoI) for each anchor box that matches the dimensions of the detected feature. Examples of anchor-based two-stage detection algorithms include Region-based Convolutional Neural Networks (R-CNNs), Fast (and Faster) R-CNNs, Spatial Pyramid Pooling Networks (SPPNets), Region-based Fully Convolutional Networks (R-FCNs), and SpineNet. Each of these detection algorithms utilise different architectural elements and optimisation techniques in order to improve performance and efficiency.

One-Stage Algorithms

The development of one-stage object detection algorithms represents a significant advancement in computer vision, and has enabled the development of real-time object detection tasks. Unlike their two-stage counterparts, one-stage detectors remove the region proposal and refinement process and instead perform operations directly on the source image, in a single pass. In this method, anchor boxes are directly classified and modified without the need to generate RoIs, which leads to faster inference speeds and lower-latency performance compared to two-stage detectors - this means that they are much more suitable for real-time classification tasks. Examples of one-stage detectors include Single-Shot MultiBox Detector (SSD), You Only Look Once (YOLO), RetinaNet, and EfficientDet. Again, each of these detection algorithms utilise different optimisation techniques in order to improve performance and detection speed and accuracy.

Two-Stage Algorithms vs One-Stage Algorithms

	Pros	**Cons**
Two-Stage	**Accuracy** The generation of region proposals and anchor boxes, then classification and regression on these boxes makes two-stage detection suitable where precision is more important than speed.	**Speed** The extra step of creating region proposals means that two-stage detectors are generally slower than one-stage detectors.
One-Stage	**Speed** One-stage detectors do not generate region proposals, so are therefore generally faster than two-stage detectors and are more suitable for real-time applications.	**Accuracy** Without region proposal and the associated object classification, one-stage detectors are not as accurate when detecting small or complex objects.

Table 1: Pros and Cons: Two-Stage vs One-Stage Algorithms

The main difference between the two algorithms is the trade-off between accuracy and processing speed. Real-time object detection will not be required for the DIVE-AIP pipeline, therefore, a two-stage object detection algorithm may be ideal for scene reconstruction.

3.2.3 Examples of Some Current Object Recognition Algorithms

Single-Shot Detector (SSD) Algorithm

THe SSD algorithm is based on the VGG-16 architecture and developed by Google, and is significantly faster for high-accuracy detection compared to prior models such as mAP, R-CNN, and YOLO.

The Single Shot MultiBox Detector (SSD) method detects objects in images using a single deep neural network. SSD training involves defining a collection of bounding boxes of different aspect ratios that detected items may fit into. In operation, SSD uses different layers of a convolutional neural network to predict a fixed set of bounding boxes, known as "anchor" boxes. In this "feature map" of anchors, SSD then iterates over the multiple bounding boxes and calculates its confidence for what object is contained in a box, which are then used to produce adjustments to the bounding box to better match the shape of the object [8]. This approach of multiple bounding boxes allows SSD to detect objects of various sizes in a single pass, hence making SSD one of the faster object detection algorithms.

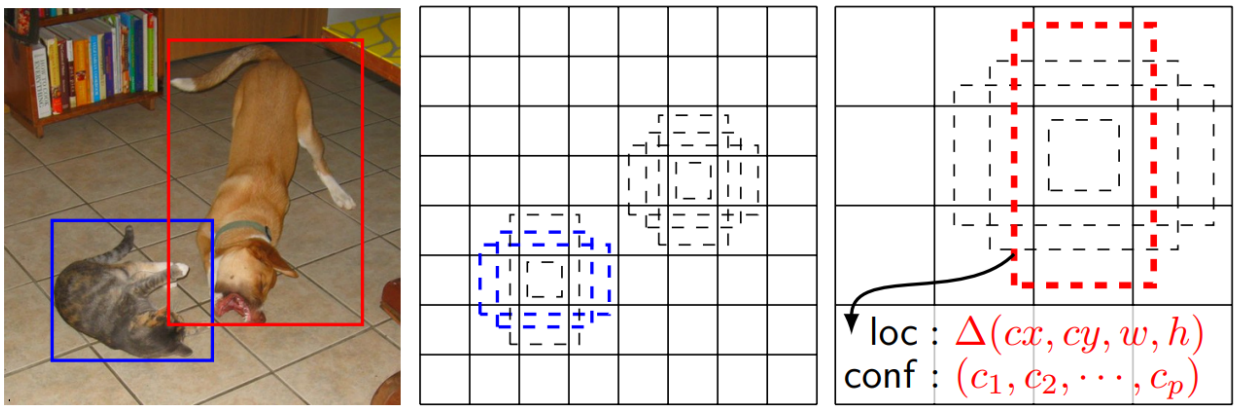

Figure 8: Sample feature maps produced by the SSD framework, Source: [8]

You Only Look Once (YOLO) Algorithm

The You Only Look Once (YOLO) family of algorithms, developed since 2016, are a popular choice for real-time object detection. YOLO has been used across autonomous vehicle systems, agriculture, medical, security, and traffic systems. The YOLO algorithm divides an input image into a grid, where each grid predicts multiple bounding boxes depending on class (category of object). Each bounding box prediction consists of five values, including the confidence score of the box, x and y coordinates of the centre of the box relative to the grid cell, and the height and width of the box relative to the image [9].

Figure 9: Use of Non-Maximum Suppression (NMS) in YOLO. a) Shows the typical output of an object detection modelcontaining multiple overlapping boxes. b) Shows the output after NMS., Source: [9]

DEtection TRansformer (DETR) Model

The DETR model utilises Transformer-based encoder-decoder architectures in order to detect objects, as opposed to anchor-based techniques. In this model, the Transformer architecture is used to first identify objects directly from the source image, by using a CNN to extract features. Then, a set of learned "object queries" are used to create predictions for each object detected by the CNN backbone. In each iteration of the refinement process, these "object queries" are adapted to better align with the characteristics of the detected objects, which results in an accurate set of final predictions [34]

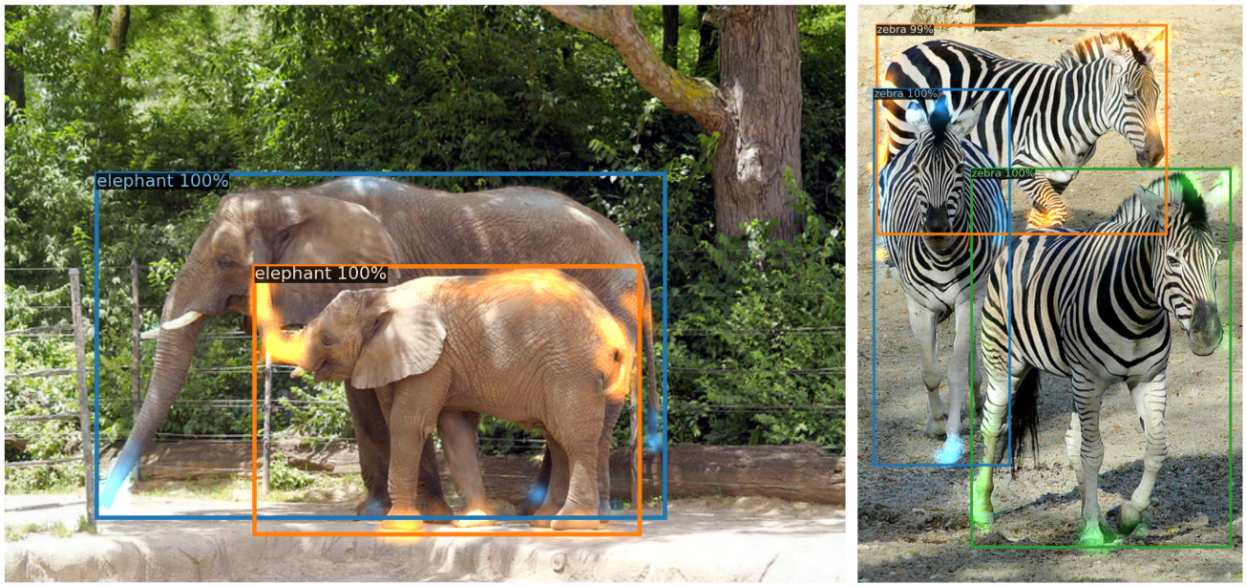

Figure 10: Visualising decoder attention for every predicted object using DETR, Source: [10]

3.2.4 Depth Estimation

Overview of Depth Estimation

In computer vision, depth estimation refers to predicting the spatial relationships between objects in a scene, relative to the camera's viewpoint. Early depth estimation involved manual intervention from researchers, where geometric principles were applied by hand in order to infer distances. The development of CNNs has greatly advanced the speed of depth estimation, as the networks are able to autonomously learn depth features and visual cues based on hierarchical representations. Presently depth estimation has uses in autonomous navigation, 3D reconstruction, and augmented reality.

3.2.5 Depth Estimation Techniques

Stereo Vision

Stereo vision calculates the depth of objects within a scene by processing the subtle differences, known as disparity, between two images taken at slightly different viewpoints. This approach is based on the principles of human binocular vision, where the brain is able to perceive depth by comparing the visual inputs from the eyes. Stereo vision usually relies on the assumption that the key camera parameters, such as field of view and aperture size, are already known. Typically, stereo vision setups use two cameras that are positioned to capture overlapping views of the same scene, mimicking binocular vision. Computer vision systems are then able to infer depth information based on pixel-level disparities between corresponding points in the two images, which can then be used to calculate the relative distances between objects in the source scene.

Monocular Depth Estimation

Monocular depth estimation involves inferring depth information from a single image, that is, a single viewpoint. Unlike stereo vision where depth is calculated from the disparity of different viewpoints, monocular depth estimation relies on visual cues including perspective, texture gradient (the density of texture depending on distance from the camera), shading and shadows, and occlusion. Use of CNNs have greatly improved the efficiency of monocular depth estimation models, as CNNs can learn to extract depth-related features from images faster and more accurately than traditional manual estimation methods.

Transformers

The Transformers framework, originally developed for natural language processing tasks, has found application in computer vision and depth estimation. Transformers holds many advantages over processing with a CNN alone, including being able to process tasks in parallel, handling multiple modalities (images, videos, text, speech), and the ability to recognise and model relations between input sequences through its self-attention mechanism. In depth estimation, this key advantage allows Transformer models to apply, analyse, predict and reapply weights to various parts of an input image to significantly improve accuracy and efficiency.

3.2.6 Examples of Some Current Depth Estimation Models

There are currently several AI models available for depth estimation within a scene.

Global-Local Path Neworks (GLPN) (monocular DE):

The GLPN architecture generates an estimated depth map by recognising global and local features in a source image, by using a hierarchical transformer encoder. [11]. The model captures the global context of the image and recognises the relations between different parts of a scene in order to generate a depth map. In addition, the GLPN model is able to achieve state-of-the-art performance for the NYU Depth V2 dataset [11]

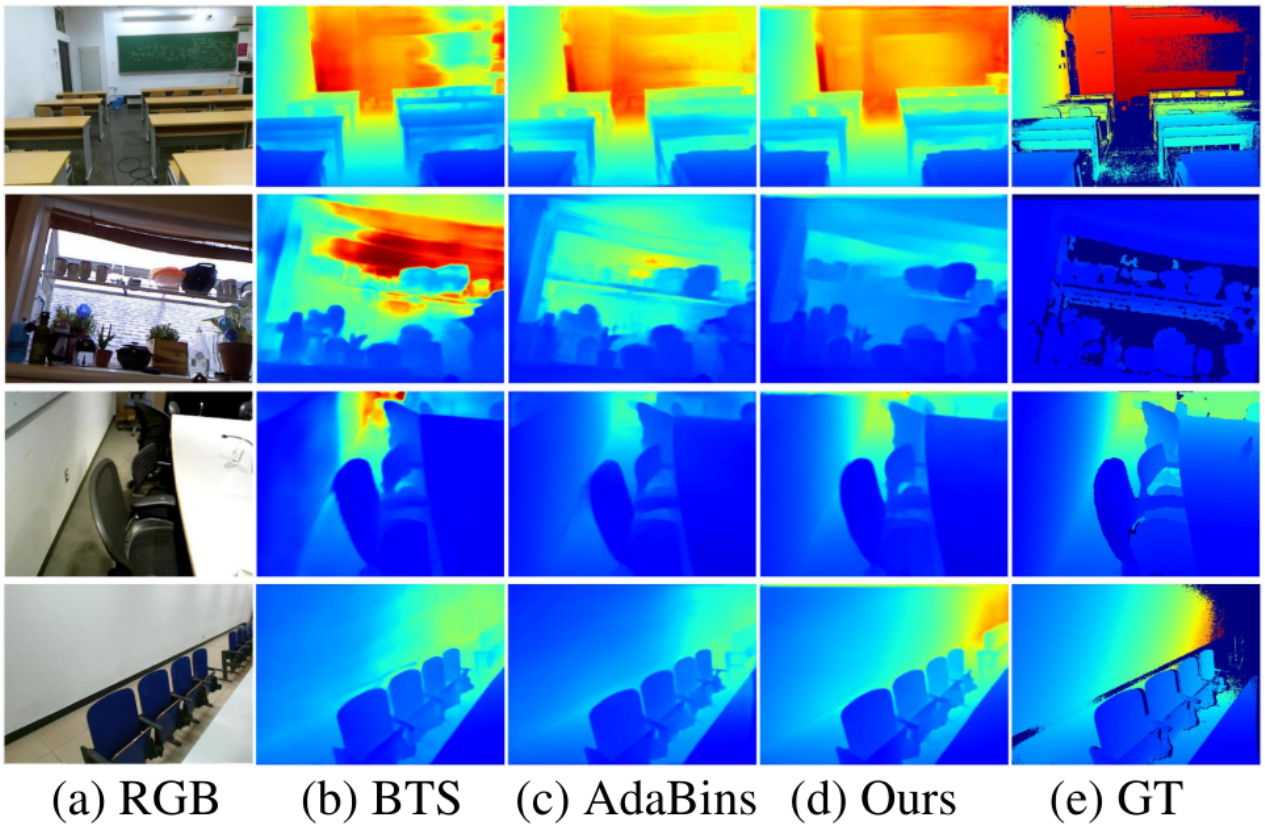

Figure 11: Examples of estimated depth maps on the SUB RGB-D dataset, using GLPN. Source: [11]

Dense Prediction Transformer (DPT) (monocular DE):

The DPT architecture utilises a vision transformer (ViT) backbone. In this model, features in the source image are extracted and then passed through the ViT, which encodes and forms relationships between the extracted features. The encoded features are then passed to a decoder, which generates a depth map from the image. This process allows DPT to provide finer-grained and more coherent predictions [12].

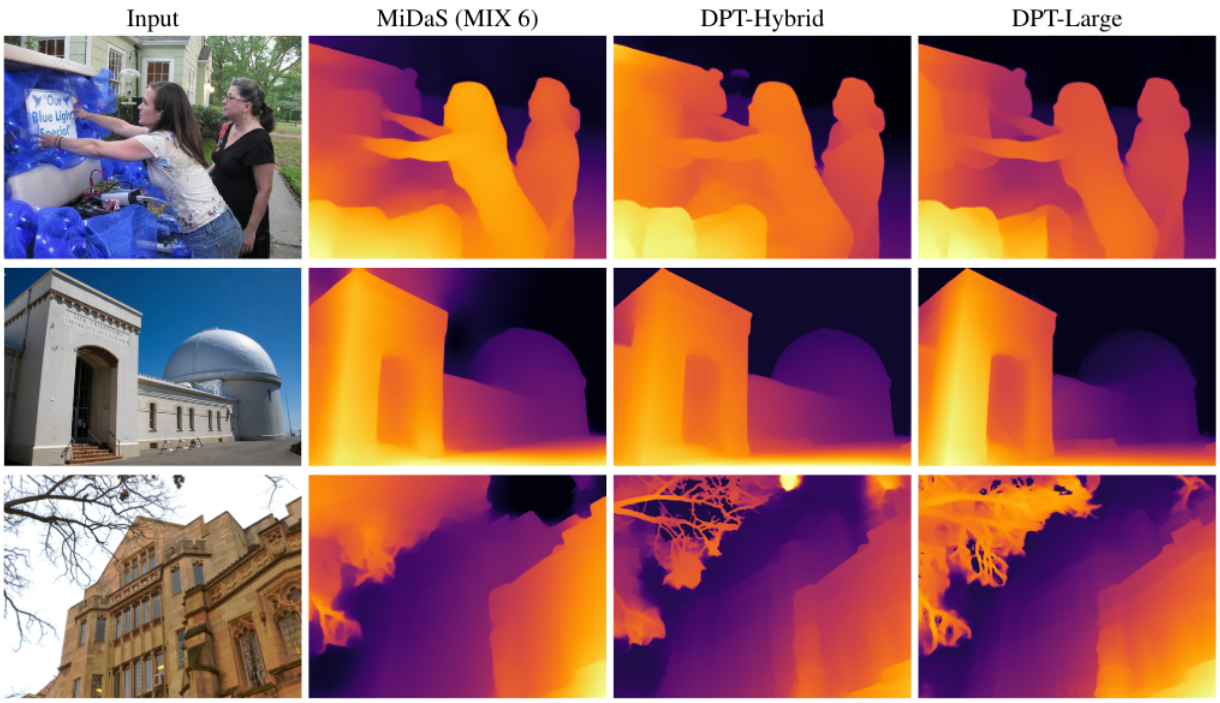

Figure 12: Sample results for monocular depth estimation, using DPT. Source: [12]

3.2.7 Examples of Some Seq2seq NLP Models

There are currently several AI models available for sequence-to-sequence translation using natural language processing:

Text-To-Text Transfer Transformer (T5):

The T5 family of models by Google reframe all NLP tasks into a text-to-text format, where the input and output sequences are always strings; this is in contrast to BERT-style models that output class labels or input spans [35]. This framework allows T5 models to perform any type of NLP task, including translation, summarisation, and question answering. T5's text-to-text interface for input and output is ideal as it simplifies the implementation of downstream tasks, without the need for additional processing from other data types.

Bidirectional Encoder Representations from Transformers (BERT):

The BERT language models by Google was designed to analyse input text bidirectionally, meaning that it is able to understand the context of a word based on its surroundings on both sides of the word (as opposed to unidirectional models, which analyse text either left-to-right or right-to-left). Google offers pre-trained BERT models which were trained on a large corpus of English data in a self-supervised fashion, where the raw texts in the training set were not labelled by humans. This makes the BERT models useful for extracting features in downstream tasks, such as language inference and question answering [36]

3.2.8 Examples of Methods to Create 3D Scenes From Images

There are several existing methods of creating a 3D scene from a single image:

Make3D

Make3D uses Markov Random Fields (MRF) to detect monocular cues in a single image, e.g. texture variations and gradients, interposition, occlusion, light and shading, and defocus [10]. In the algorithm, an image is passed through a segmentation algorithm in order to divide it into many small regions known as superpixels. Then, the algorithm attempts to infer the 3D position and orientation of the surface that the superpixel lies on by inferring meaningful boundaries within the image, such as occlusion boundaries or folds. Once the position and orientation have been calculated, the model is then able to build a 3D mesh of the scene and apply the original image as a texture.

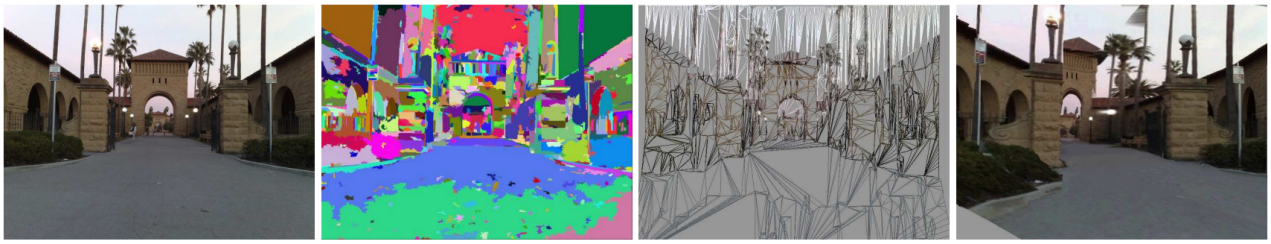

Figure 13: Make3D sample: (a) An original image. (b) Oversegmentation of the image to obtain "superpixels". (c) The 3-d model predicted by the algorithm. (d) A screenshot of the textured 3-d model, Source: [10]

Approximate Differentiable One-Pixel Point Rendering (ADOP)

ADOP takes a collection of calibrated camera images and a point cloud to generate a high quality render in real-time. In order to generate a view, ADOP learns feature vectors for each point in a given point cloud, and applies a neural network to fill in gaps in the mesh. The model also uses a point-based rendering pipeline that is able to self-optimise input parameters such as camera positions and lens distortions to reduce inconsistencies caused by an imperfect reconstruction or calibration [13].

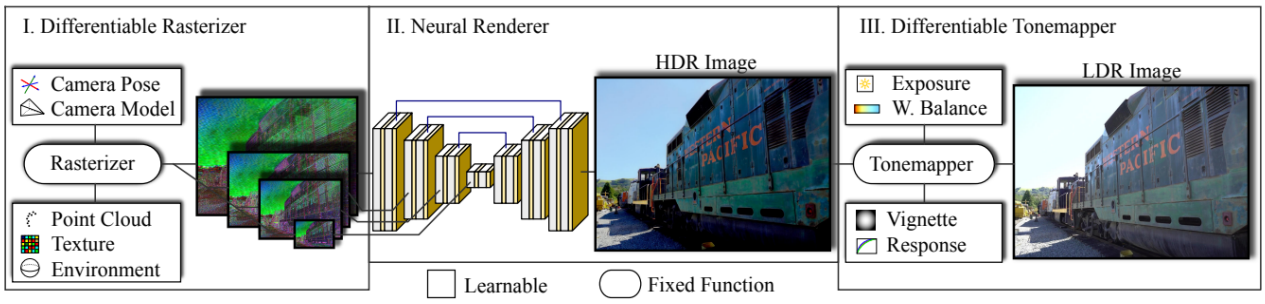

Figure 14: Overview of ADOP's point-based HDR neural rendering pipeline, Source: [13]

Make3D vs ADOP

Make3D and ADOP are both able to construct 3D models from 2D images. However, where Make3D focuses on creating a mesh from a single input image, ADOP focuses more on the refinement of an existing proxy mesh or point cloud. Neither model is able to create a scene with independent objects - they both result in a single mesh with all objects in the scene encompassed.

	Pros	**Cons**
Make3D	**Single Image** Make3D is able to reconstruct scenes from a single image, having been trained on pairs of RGB and depth map images.	**Complexity of a scene** The model may struggle to reconstruct items outside of its trained dataset
ADOP	**Variability in input images** ADOP is able to achieve a high render quality even with disparities between input images, e.g. varying exposure, imperfect camera calibrations.	**Computation** ADOP requires multiple (ideally) precisely positioned images, as well as a proxy input 3D mesh in order to recreate a model with high render quality.

Table 2: Pros and Cons: Make3D vs ADOP

4 Proposed Approach

This section will describe the high-level approach taken to develop the DIVE-AIP pipeline. It will also contain the justifications for the assets used in the pipeline.

4.1 Assets To Be Used:

- Blender
  - The 3D creation suite Blender was chosen for the creation and viewing of 3D models. This program was chosen as it is free, open-source, and widely used by the 3D modelling community.
- Facebook/detr-resnet-50 model: Object Detection
  - The detr-resnet-50 model was chosen for the purpose of object detection in the DIVE-AIP pipeline. This model was chosen as it is open-source, and one of the most used object detection models provided by the HuggingFace platform
- Intel/dpt-hybrid-midas model: Depth Estimation
  - The dpt-hybrid-midas model was chosen for the purpose of depth estimation in the DIVE-AIP pipeline. This model was chosen as it is open-source, and one of the most used depth estimation models provided by the HuggingFace platform
- LiheYoung/depth-anything-small-hf (Depth Estimation), and hustvl/yolos-base (Object Detection)
  - These supplementary models were chosen in order to test the usage of different models in the DIVE-AIP pipeline. These models were selected as they are both open-source, and had high download numbers on the HuggingFace platform.
- Salesforce/CodeT5-small
  - The CodeT5-small model will be used as the base model for fine-tuning the Code Generation model. This model was chosen as it was based on Google's T5 family of models, but was pretrained on a corpus of programming code, which makes it ideal for translating code from one language to another. The CodeT5 family of models is also open-source.

4.2 Proposed Pipeline

The proposed pipeline will include three machine learning steps:

1. Object detection
2. Depth estimation
3. Sequence-to-sequence code generation

4.2.1 Proposed Process to Developing the Pipeline

The proposed process to developing the pipeline will be as follows:

1. Collecting Data for Training:
   (a) In order to train the sequence-to-sequence code generation model, a collection of 3D models with description pairing must be curated. This collection will consist of two data types: a 3D model file, in COLLADA file format, and an accompanying code description of the file. In this approach, both the 3D models and code descriptions will be generated using a combination of Python scripting and the 3D modelling program Blender.
2. Training the Code Generation Model:
   (a) The Salesforce/CodeT5-small model will be fine-tuned on the 3D model/description pairings, in order to develop a model that can translate description files into COLLADA files.
3. Object Detection Implementation:
   (a) The Facebook/detr-resnet-50 model will be used for object detection, and imported via the HuggingFace Transformers pipeline. The expected output is a list of objects within the scene.
4. Depth Estimation Implementation:
   (a) The Intel/dpt-hybrid-midas model will be used for depth estimation, and imported via the HuggingFace Transformers pipeline. The expected output is a depth map of the source image, from which the depth of objects in the scene can be inferred.
5. Code Generation:
   (a) Generated information from the Object Detection and Depth Estimation Implementation steps will be used to create a description file, which will then be fed into the Code Generation model. This description file must include the estimated dimensions and location of each object detected in the source image. The output should be a COLLADA 3D scene file that contains individual representations of each object detected in the source image.

5 Methodology

The process of converting a single image into an interactive 3D environment began with the construction of a pipeline, where the image was fed into several components that extract the necessary information to construct a 3D scene. In our approach, this pipeline consists of an object detection model, a depth estimation model, and a code conversion model.

1. The image was first processed by the object detection model, which identifies various objects within the image
2. The image was then passed through a depth estimation model, which identifies the placement of objects in the scene relative to the camera
3. Information from steps 1) and 2) were combined: Objects in the scene identified in step 1) were used to isolate the depth information of each object from step 2)
4. Depth and placement information of each object was converted to a .json file
5. Each object's corresponding .json file was passed to a code conversion model, which created a COLLADA code fragment for each object
6. The generated code fragments were then combined into a useable 3D COLLADA file

The pipeline was constructed using Google Colaboratory, a hosted Jupyter Notebook service that offers free access to computing resources.

The following sections will further detail each component within the DIVE-AIP pipeline.

5.1 Sample Data

Sample images were procured in order to test the accuracy of the DIVE-AIP pipeline. These images included photography of real-life scenarios and 3D renderings. The following conditions were used in the selection and development of these images:

1. $4 \leq x \leq 8$ furniture objects clearly visible (i.e. tables, chairs)
2. A single ground plane (no multiple elevations)
3. All furniture objects on the same ground plane
4. Defined rear wall in the scene (image must be of an enclosed space)

5.2 The Code Conversion Model

The first major hurdle involved devising a method to convert data between two distinct formats: JSON and COLLADA - JSON files represent data in a hierarchical key-value pair structure, where COLLADA files follow a non-hierarchical XML-based format. Our approach was to custom-train a sequence-to-sequence (seq2seq) model to facilitate this conversion process, by training the model on a dataset of paired JSON and COLLADA code fragments. This was accomplished through the following process:

5.2.1 Procuring the Dataset

We used a combination of Python scripting and the 3D modelling program Blender to automate the creation of a diverse 3D dataset to be used to train the code conversion model. Using the Blender API the Python script created a series of files, each featuring a number of objects: a large cube to represent a room, and smaller cubes to represent objects (furniture, etc) within the room.

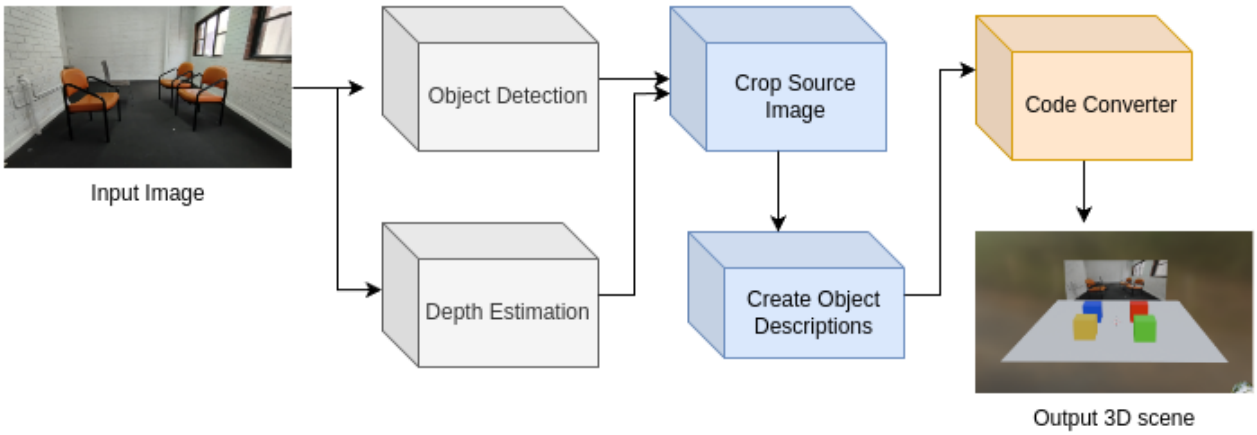

Figure 15: Visualisation of the DIVE-AIP pipeline

The size of the room cube, and number and placement of objects within the room cube were randomised with each generation. The generated scenes were exported by Blender to their own COLLADA files, and the Python script saved the generation data to a separate JSON file, thus creating the JSON/COLLADA pairings to be used in training.

A dataset of 100 JSON and COLLADA pairings was created to train the code conversion model on. This number was chosen as it is recommended by multiple sources to train seq2seq models on greater than 50 data samples.

5.2.2 Training the Code Conversion Model

The Google Colaboratory platform and HuggingFace Transformers library were used to facilitate the task of fine-tuning a seq2seq model to perform code conversion. We used the existing CodeT5 model by Salesforce as the base model for training, as the pre-trained model was intended for use in code summarisation, generation, and refinement. The process of fine-tuning

the model was achieved by training the model to recognise the relationships between the given COLLADA and JSON files. The process was as follows:

1. The COLLADA files were stripped of their default code structure. This was done in order to isolate the code fragment that specifically referred to 3D object within the code, as well as to reduce the amount of processing required by the model trainer
2. The COLLADA code fragments and JSON files were combined into a single Python dataset with two columns: "GENERATED_DATA" containing the COLLADA code fragments, and "GENERATED_DESCRIPTION" containing the JSON description files
3. This dataset was split into training, testing, and validation subsets for the training process
4. The dataset was converted into tokens that could be understood by the model
5. The Trainer library included in the HuggingFace Transformers library was used to train the CodeT5 on the tokenized data. It is in this step that the CodeT5 model analyses and forms relationships between the input JSON description files and COLLADA output files.
6. The resulting fine-tuned model was saved locally, to be used in later steps.

```
dataframe = pd.DataFrame(data_list, columns=["GENERATED_DESCRIPTION", "GENERATED_DATA"])
dataframe
```

	GENERATED_DESCRIPTION	GENERATED_DATA
0	{'location': '<Vector (0.0000, 0.0000, 12.5000...	b'<library_geometries>\n <geometry id="Obje...
1	{'location': '<Vector (-4.0000, -4.0000, 0.500...	b'<library_geometries>\n <geometry id="Obje...
2	{'location': '<Vector (-8.0000, -2.0000, 0.500...	b'<library_geometries>\n <geometry id="room...
3	{'location': '<Vector (9.0000, 0.0000, 0.5000)...	b'<library_geometries>\n <geometry id="Obje...
4	{'location': '<Vector (6.0000, 6.0000, 0.5000)...	b'<library_geometries>\n <geometry id="Obje...
...	...	...
1243	{'location': '<Vector (-1.5000, 5.5000, 0.5000...	b'<library_geometries>\n <geometry id="Obje...
1244	{'location': '<Vector (2.5000, -6.5000, 0.5000...	b'<library_geometries>\n <geometry id="room...
1245	{'location': '<Vector (3.5000, 9.5000, 0.5000)...	b'<library_geometries>\n <geometry id="Obje...
1246	{'location': '<Vector (0.5000, -9.5000, 0.5000...	b'<library_geometries>\n <geometry id="Obje...
1247	{'location': '<Vector (-1.5000, 4.5000, 0.5000...	b'<library_geometries>\n <geometry id="Obje...

1248 rows × 2 columns

Figure 16: Sample of the dataset used to train the Code Conversion model, as seen in Google Colab

5.3 Pipeline Architecture

5.3.1 Pipeline Backbone

The HuggingFace Transformers library was used as the underlying structure that handled the various file types, data conversions, and high-level abstractions used by the natural language processing (NLP) models in the pipeline. The library simplified the processes of importing pretrained AI models and piping data between models by providing a high-level interface, while still allowing customizability of imported models.

5.3.2 Object Detection

The core process of the DIVE-AIP pipeline is the function to discern various objects within a single image, from which 3D representations are generated. For this purpose, we decided to use the DEtection TRansformer model with ResNet-50 backbone (detr-resnet-50), by Facebook. This object detection model was chosen as it was free, open-source, and most importantly trained on the COCO 2017 object detection dataset, which contains 5k+ labelled samples of chairs, tables, and other furniture. The detr-resnet-50 model is also capable of detecting up to 100 separate objects within a scene.

The DIVE-AIP pipeline first passed an input image through the detr-resnet-50 model. This produced a set of bounding boxes and labels for each object detected in the image, represented as a Python dictionary containing three key-value pairs:

1. Scores: A tensor variable containing the confidence scores of each detected object, represented between 0 and 1
2. Labels: A tensor variable containing the class labels, which correspond to the classification labels from the COCO 2017 dataset
3. Boxes: A tensor variable containing the bounding box coordinates of each detected object, relative to the source image

Figure 17: Sample of the Object Detection process, with confidence scores

5.3.3 Depth Estimation

The depth estimation step in the DIVE-AIP pipeline allowed for the inference of 3D distance from a single (monocular) 2D source image. In this step we used the Dense Prediction Transformer with ViT-hybrid backbone (dpt-hybrid-midas) model by Intel. This depth estimation model was chosen as it was free and open-source, and allowed for depth estimation from a single image (as opposed to other models which required two or more images of a scene taken at different perspectives).

In this step, the source image was passed through the dpt-hybrid-midas model. This produced a Python dictionary containing the following variables:

1. `predicted_depth`: A tensor variable containing the estimated depth of the image at each pixel
2. `depth`: An image that represented the estimated depth, shown as a gradient from white (closest) to black (furthest)

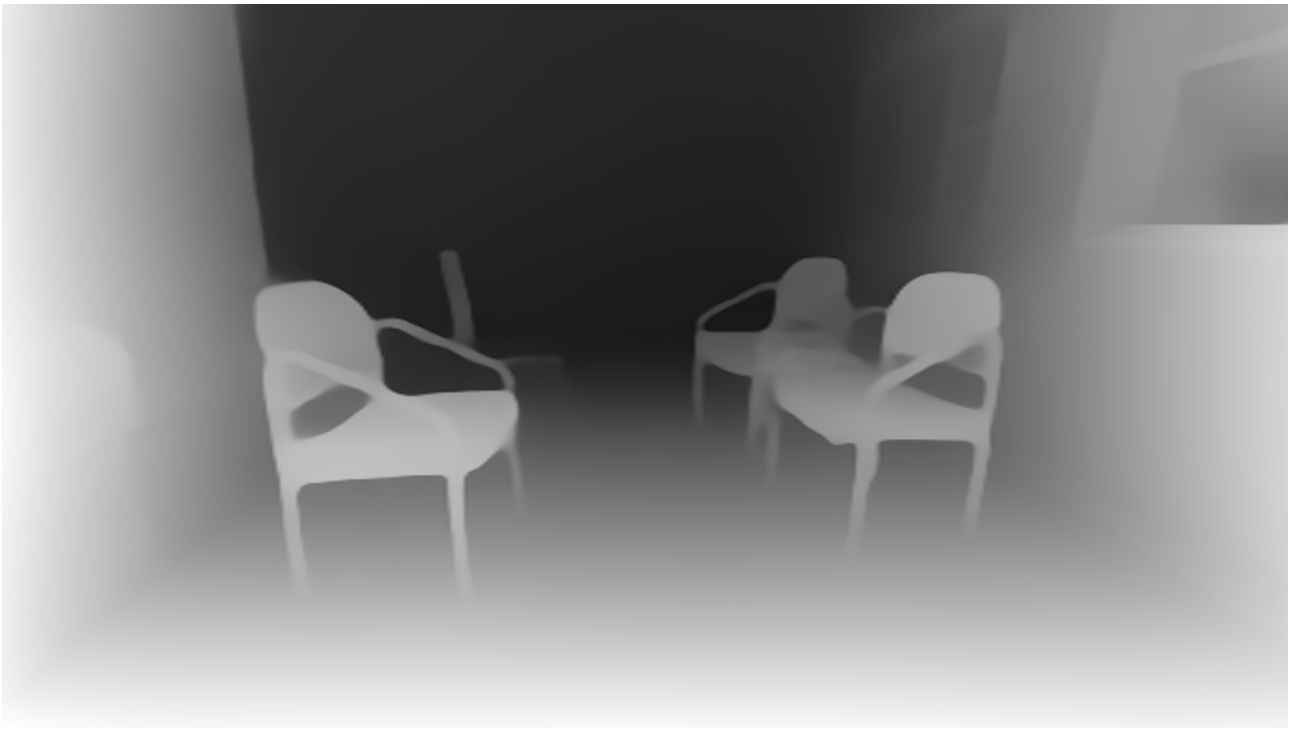

Figure 18: Sample of the Depth Estimation process

The estimated depth information was then used in the Isolating Objects step.

5.3.4 Isolating Objects

Once the bounding boxes and depth estimation maps of objects within the source image have been generated, the information was passed through a Python function that isolated each object within the depth estimation map. The Python PIL library was used for this process, and involved generating new images by cropping the depth estimation map, based on the coordinates of the generated bounding boxes. This resulted in an array of images, where each image represented the depth information of each object. Since the depth was represented by the brightness value of each pixel (represented as an RGB value between 0-255), the average value of each image was calculated to represent the average depth of the object - this was used to determine how "deep" the object was in the image (relative to the camera). This average depth value was then mapped between 0 (the placement of the camera) and a user-defined depth of the scene, in metres.

The depth information of each object in the scene was then used in the Code Conversion Preprocessing step.

5.3.5 Code Conversion Preprocessing

The data generated in the Isolating Objects step was then concatenated into a data format that could be understood by the code conversion module. A key was created for each object, with a nested dictionary created for each key containing the predicted location and a fixed size of [1, 1, 1] (xyz values, i.e. a cube). Once the dictionary has been built that holds the converted

objects, it is exported as a JSON file to be processed by the code conversion module.

5.3.6 Code Conversion Module

When loaded into the code conversion module, the JSON file is read line-by-line (and hence, object by object) and tokenized into a format understandable by the fine-tuned CodeT5 model. The fine-tuned model is then instructed to translate the JSON code into a COLLADA code fragment. The resulting code fragments are reinserted back into a COLLADA file structure using Python.

6 Results

We were able to successfully train the code conversion module to translate code from a JSON description file to a COLLADA 3D environment file. When using an Nvidia A100 GPU provided by the Google Colaboratory service we were able to train the model in 7.5 minutes, with a final training loss of 0.4452 and a final validation loss of 0.0129 (lower is better, range from 0 to positive infinity). Training the model consumed 21.8GB of VRAM out of the 40GB available, and the final model was 231MB in size.

```
trainer.train()
#A100 GPU == 21.8/40GB used, 7.5min
```

[900/900 07:21, Epoch 3/3]

Step	Training Loss	Validation Loss
100	No log	0.316607
200	No log	0.113534
300	No log	0.058390
400	No log	0.026888
500	0.445200	0.019753
600	0.445200	0.016668
700	0.445200	0.015266
800	0.445200	0.013299
900	0.445200	0.012917

Figure 19: Results of training the Code Conversion model

The DIVE-AIP pipeline was tested on several images. In each case, the pipeline was able to generate a 3D environment with individual objects to represent furniture within the scene. With each generation, the estimated position of objects closely matched the placement in the real-life scenario, where the maximum deviation in the testing dataset was 1.5 metres. The DIVE-AIP pipeline took on average 19s to convert a single JSON description file into a COLLADA code fragment. The time taken to generate a complete environment increased linearly with the number of objects within the scene. The DIVE-AIP pipeline performed poorly when estimating the depth of objects that were concealed from the camera behind other objects.

Using different object detection models yielded no change in the number of objects detected. In contrast, different depth estimation models produced slightly different object placements in the axis pointing away from the camera. Different depth estimation models also estimated a different maximum scene depth, therefore requiring a user-added offset value in order to produce an accurate depth estimation.

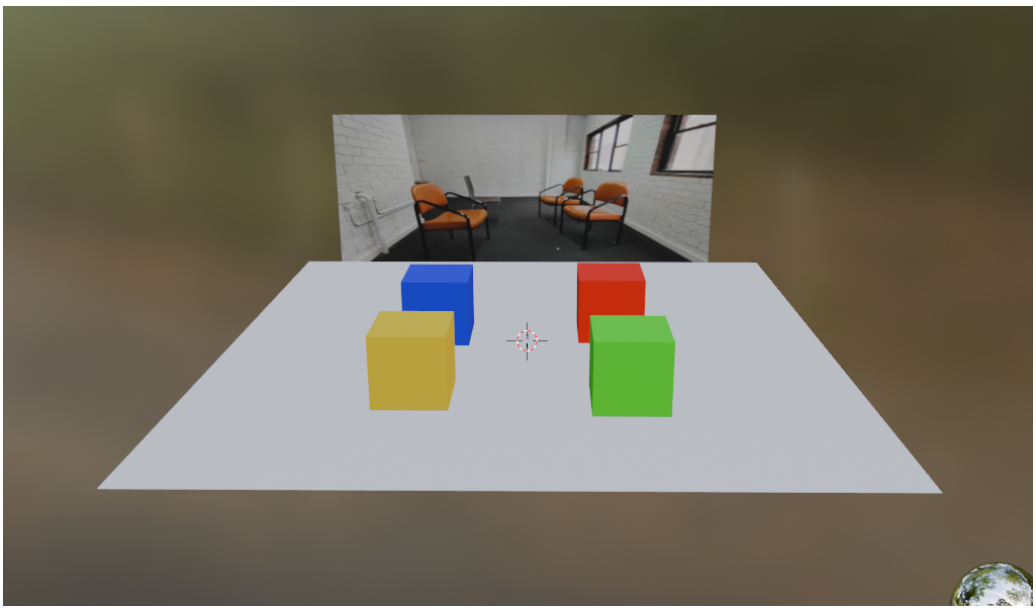

Figure 20: Demonstration of the DIVE-AIP pipeline in Blender, with original image for reference. Objects were coloured in post-processing.

Figure 20 shows an example of a scene that was recreated using the DIVE-AIP pipeline. In this scene, each of the coloured boxes represents a chair that was detected in the source image. In the example shown, the cube representing the room has been replaced with a single plane, and each of the generated cubes have been coloured post-generation for the sake of clarity.

6 Discussion

This section will explore the implications of the findings from the Results section, and will provide insight to possible use cases of the results.

7.1 Key Findings

The DIVE-AIP pipeline is able to create virtual environments with interactive objects from a single image, and is able to do so in a much shorter time frame compared to traditional methods. However, currently the generated environments are very basic but can be used as a guideline on which traditional methods can build upon; the DIVE-AIP pipeline still reduces the time spent in the Conceptualise Environment and Creation of Assets stages of traditional development. The accuracy of a predicted environment may depend on the quality of the source image, including the resolution of the taken image, noise of the camera sensor, and lens distortion. Due to the use of the HuggingFace Transformers pipeline, different object detection and/or depth estimation models can be used to produce different inferences.

7.2 Comparison With Existing Methods

In comparison to existing methods to create 3D environments from images:

Make3D	**DIVE-AIP pipeline**
Can apply textures to the created scene	Cannot apply textures to the created scene
Cannot create independent objects to represent objects within the scene	Can create independent objects to represent objects within the scene

Table 3: Make3D vs DIVE-AIP pipeline

ADOP	**DIVE-AIP pipeline**
Can operate in real time	Cannot operate in real time
Can produce a high-quality mesh	Cannot create a high-quality mesh
Continued on next page	

	Continued from previous page
Can apply textures to the created scene	Cannot apply textures to the created scene
Requires an initial estimate of the point cloud, camera parameters, and a set of camera poses	Only requires a single input image
Cannot create independent objects to represent objects within the scene	Can create independent objects to represent objects within the scene

Table 4: ADOP vs DIVE-AIP pipeline

7.3 Limitations

This section will discuss some of the limitations faced by the DIVE-AIP pipeline.

7.3.1 Input Sequences in Training

Many Transformer-based models have a maximum length of input sequences that can be processed - typically a cap of 512 tokens. We found that a tokenized JSON scene description file contained sequences that were between 200 tokens long for scenes with two objects (room and single object), and 1039 tokens long for scenes with twenty objects. The cap of input tokens by the Transformer-based models meant that input sequences were truncated - this resulted in information loss and an incomplete contextual understanding of the entire file to be created. While the model used in this project could have been trained on input sequences larger than 512 tokens, the amount of VRAM used The solution to this limitation was to employ a chunking strategy to the input files. This involved removing all boilerplate code from the generated COLLADA files, so that each file represented a code fragment that captured the actual geometry of each object. Similarly, the JSON description files were stripped so that each description file pertained to exactly one COLLADA file - that is, one description for each object in a scene. These description code fragments were significantly smaller and within the 512 character limit, at an average of 57 tokens.

7.3.2 Computational Resources

Training an NLP model is a computationally intensive task. The process usually requires substantial memory resources in order to store the model parameters and data structures generated during the training process. As mentioned earlier the code conversion model was trained using a Nvidia A100 GPU, with 40GB of RAM. In addition, we were able to utilise existing deep learning frameworks including TensorFlow, as well as CUDA libraries in order to manage memory allocation on the GPU. However, A100 GPUs, priced at around $25,000 AUD (at the time of writing) may fall well beyond the reach of most consumers if they do not have access to Google Colaboratory (or its paid plans). In this instance, training an NLP model may take significantly longer, or may even fail if a GPU with insufficient VRAM is used.

7.3.3 Inaccurate Position of Generated Objects

Currently the x-position (relative to the camera) of predicted objects is estimated by inferring the position of the bounding box relative to the source image. While this method works for objects that are closer to the camera, it does not provide the true x-position because it does not take into account depth of field. This feature will be explored further in the Future Work section.

7.3.4 Model Discrepancies

Through the HuggingFace Transformers library, the DIVE-AIP pipeline is able to use various depth estimation models in the depth estimation step. However, due to differences in the training of each of these models, each depth estimation model will produce slightly different inference results, but moreover will produce different maximum predicted depths. Currently this means that a maximum real-world depth (in metres) must be provided in order for the DIVE-AIP pipeline to correctly gauge the depth and distance of each predicted object.

7.3.5 Scale of Generated Objects

Currently, objects detected within an image are represented as a 1x1x1m cube. As such, this does not represent the true scale of the detected objects, which may lead to confusion when the generated file is imported into a 3D workspace. As a result, developers using the DIVE-AIP pipeline will still need to reference the original scene or image when developing assets in later stages of development, and may need further information such as the dimensions of objects in the scene.

7.3.6 Input Image Limitations

The DIVE-AIP pipeline was designed to operate on a single (monocular) input image. As a result, the pipeline is not able to detect and estimate objects that are obstructed from view, for example partially hidden behind another object, or behind corners. A solution to this issue is suggested in the Future Work section.

7.3.7 Limitations of Scenes

One major limitation of the DIVE-AIP pipeline is the inability to accurately process scenes that are not in an enclosed area, for example, outdoors in a field. When the furthest point in an unenclosed scene (the horizon) is very far away, the depth estimation model's maximum depth prediction tends to read closer, and therefore the generated depth position of objects will be incorrect. In the same vein, objects that are too far away will appear too small in the source image, and thus may not be correctly identified by the object detection models.

8 Future Work

This section will explore future work that can expand the capabilities of the project, as well as improvements to existing methods.

8.1 Improving the Position of Generated Objects

The current method used by the DIVE-AIP pipeline to estimate the x-position of objects within the scene is inaccurate as it does not take into account field of view, nor depth of field. Future work may include additional libraries with parameters that take into account information such as camera position and height, characteristics such as lens distortion and focal length. Future work may also include the use of a real-world calibration object with fixed properties that can then be recognised and translated into virtual coordinates by a computer vision system.

8.2 Naming of Generated Objects

Currently the DIVE-AIP pipeline generates nondescript names for each detected object in a scene, such as "Object1" or "Object2". Future work may involve fine-tuning the object detection model, or adding more classifiers. This would allow the DIVE-AIP pipeline to generate more specific names for objects based on the results of the object detection step. In turn, this will aid the process of developing a 3D environment, especially in scenes that contain many objects.

8.3 Generating High-Quality Meshes

In its current state, the DIVE-AIP pipeline represents detected objects as 1x1x1m cubes in the generated virtual scene. Future work may investigate a database of 3D models that can be imported into the scene as better placeholders, based on the categories of objects detected by the pipeline, and where scale is based off of the depth estimation model. This will afford developers a better sense of scale when building upon the generated 3D environment, and can shorten the development process further. In addition to this, if 3D models can be generated based on detecting geometry from the source image, the generated objects can then be textured based on the source image, in a similar fashion to other existing image-to-3D models.

8.4 Improving the Pipeline's Input Capabilities

The DIVE-AIP pipeline was designed to operate on a single (monocular) image. As discussed above, this means that the pipeline is not able to recognise or generate objects that are obscured from the camera, or are behind other objects. One solution may be to modify the pipeline to accept stereoscopic input - that is, a scene captured from multiple cameras or viewpoints so that depth of field can be inferred; this process may also allow obstructed objects to be more clearly visible. Another solution could be to allow the pipeline to accept a series of images, similar to existing image-to-3D models discussed earlier - viewpoints taken from various locations within the scene may help to remove the issue of objects hidden from view.

8 Conclusion

In conclusion, the DIVE-AIP pipeline presents a successful concept in the rapid creation of virtual environments. The pipeline improves over existing methods by offering the capability to create a 3D scene with independent, interactive objects that are detected from the source image. While the current output may be considered limited compared to existing methods, its basis on open-source elements mean that the DIVE-AIP pipeline can be used as a foundation for further development.

Appendix

File repository, including Jupyter Notebooks, can be found here:

https://github.com/MrYeeOus/DIVE-AIEP

References

[1] URL https://blogs.nvidia.com/blog/2018/08/01/ray-tracing-global-illumination-turner

[2] Horton heilig's concept, "the sensorama" (sensorama simulator 1962). URL https://www.researchgate.net/figure/Horton-Heiligs-concept-The-Sensorama-Sensorama-simulator-1962_fig9_333855439.

[3] URL http://etsanggarp.blogspot.com/2016/03/.

[4] Wire-frame model of the utah teapot. URL https://www.computerhistory.org/revolution/computer-graphics-music-and-art/15/206/554.

[5] URL https://blogs.nvidia.com/blog/2018/08/01/ray-tracing-global-illumination-turner

[6] Frank rosenblatt's mark 1 perceptron. URL https://data-science-blog.com/blog/2020/07/16/a-brief-historyof-neural-nets-everything-you-should-know-before-learning

[7] Y. Wang J. Ren. Overview of object detection algorithms using convolutional neural networks. *Journal of Computer and Communications*, 10:pp.115–132, 2022. URL https://doi.org/10.4236/jcc.2022.101006.

[8] et al. W. Liu, D. Anguelov. Ssd: Single shot multibox detector. *Computer Vision and Pattern Recognition*, pages pp.21–37, 2016. URL https://doi.org/10.1007%2F978-3-319-46448-0_2.

[9] D. Cordova-Esparza J. Terven. A comprehensive review of yolo: From yolov1 and beyond. *Computer Vision and Pattern Recognition*, pages pp.1–34, 2023. URL https://arxiv.org/abs/2304.00501.

[10] et al. A. Saxena, M. Sun. Make3d: Depth perception from a single still image. *AAAI*, 3: pp.1571–1576, 2008. URL https://cdn.aaai.org/AAAI/2008/AAAI08-265.pdf.

[11] et al. D. Kim, W. Ka. Global-local path networks for monocular depth estimation with vertical cutdepth. *Computer Vision and Pattern Recognition*, 2022. URL https://arxiv.org/pdf/2201.07436.

[12] V. Koltun R. Ranftl, A. Bochkovskiy. Vision transformers for dense prediction. *Computer Vision and Pattern Recognition*, 2021. URL https://doi.org/10.48550/arXiv.2103.13413.

[13] et al. D. Rückert, L. Franke. Adop: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (ToG)*, 41(4):pp.1–14, 2022. URL https://dl.acm.org/doi/abs/10.1145/3528223.3530122.

[14] F. P. Brooks. What's real about virtual reality? *IEEE Computer Graphics and Applications*, 19(6):pp.16–27, 1999. URL https://doi.org/10.1109/38.799723.

[15] et al. G. Zhao, M. Fan. The comparison of teaching efficiency between virtual reality and traditional education in medical education: a systematic review and meta-analysis. *Annals of Translational Medicine*, 9(3):pp.1–8, 2020. URL http://dx.doi.org/10.21037/atm-20-2785.

[16] M. Dodge. St. nicholas. 70(4):pp.102–112, 1943. URL https://original-ufdc.uflib.ufl.edu//UF00065513/00179.

[17] Guest editorial. Charles wheatstone (1802 - 1875). *Perception*, 31:pp.265–272, 2002. URL http://dx.doi.org/10.1068/p3103ed.

[18] J. Lowe. The oculus rift and virtuix omni. *Computers for Everyone*, 1:pp.102–103, 2014. URL https://search.iczhiku.com/paurl=/GIhcbqY7IoPP5Fex.pdf.

[19] Virtual Reality Society. History of virtual reality. 2017. URL https://www.vrs.org.uk/virtual-reality/history.html.

[20] S. Berry. The evolution of virtual reality. *Computers for Everyone*, 1:pp.15–17, 2014. URL https://search.iczhiku.com/paurl=/GIhcbqY7IoPP5Fex.pdf.

[21] T. Cornall. The oculus rift. *Computers For Everyone*, 1:pp.45–48, 2014. URL https://search.iczhiku.com/paurl=/GIhcbqY7IoPP5Fex.pdf.

[22] et al I. Wohlgenannt, A. Simons. Virtual reality. *Business & Information Systems Engineering*, 62:pp.455–461, 2020. URL https://doi.org/10.1007/s12599-020-00658-9.

[23] et al V. Mach, J. Valouch. Virtual reality - level of immersion within the crime investigation. *23rd International Conference on Circuits, Systems, Communications and Computers (CSCC 2019)*, 292:pp.1–4, 2019. URL https://doi.org/10.1051/matecconf/201929201031.

[24] Nisha. Visible surface detection algorithms: A review. *International Journal of Advanced Engineering Research and Science*, 4(2):pp.147–150, 2017. URL https://dx.doi.org/10.22161/ijaers.4.2.29.

[25] A. Glassner. Situation normal (gourand and phong shading). *IEEE Computer Graphics and Applications*, 17(2):pp.83–87, 1997. URL https://doi.org/10.1109/38.574687.

[26] et al. A. Romanyuk, A. Hast. The methods of the fast shading. *9th International Conference on Development and Application Systems*, pages pp.175–179, 2008. URL http://www.dasconference.ro/papers/2008/D10.pdf.

[27] J. T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):pp.343–349, 1980. URL https://dl.acm.org/doi/10.1145/358876.358882.

[28] A. Kaplan M. Haenlein. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, 61(4):pp.5–14, 2019. URL https://doi.org/10.1177/0008125619864925.

[29] A. M. Turing. Intelligent machinery. pages pp.107–127, 1948. URL https://hashingit.com/elements/research-resources/1948-intelligent-machinery.pdf.

[30] B. G. Buchanan. A (very) brief history of artificial intelligence. *AI Magazine*, 26(4):pp.53–60, 2005. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1848.

[31] A. L. Fradkov. Early history of machine learning. *21st IFAC World Congress*, pages pp.1385–1391, 2020. URL https://doi.org/10.1016/j.ifacol.2020.12.1888.

[32] J.W. Feng Y.C. Wu. Development and application of artificial neural network. *Wireless url =sonal Communications*, 102:pp.1645–1656, 2018. URL https://doi.org/10.1007/s11277-017-5224-x.

[33] et al. P.F. Felzenszwalb, R.B. Girshick. Object detection with discriminately trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9): pp.1627–1645, 2010. URL 10.1109/TPAMI.2009.167.

[34] et. al N. Carion, F. Massa. End-to-end object detection with transformers. *Computer Vision and Pattern Recognition*, 2020. URL https://arxiv.org/pdf/2005.12872.

[35] et al. C. Raffel, N. Shazeer. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research 21 (2020)*, pages pp.1–67, 2020. URL https://jmlr.org/papers/volume21/20-074/20-074.pdf.

[36] et al. J. Devlin, M. Chang. Bert: Pre-training of deep bidirectional transformers for language understanding. *Computation and Language*, 2018. URL https://arxiv.org/pdf/1810.04805.