Frame Generation in the Web Browser for Alternative Object Angles

Victor Vlad
Faculty of Computer Science
Alexandru Ioan Cuza University
Iaşi, Romania
0009-0005-8504-1964

Sabin C. Buraga
Faculty of Computer Science
Alexandru Ioan Cuza University
Iaşi, Romania
0000-0001-9308-0262

***Abstract*—Modern Web development, particularly in the domain of multimedia processing and video analysis, increasingly emphasizes leveraging general JavaScript programming and, more specifically, the TensorFlow platform to transition traditionally server-based machine learning applications to the Web browser. This approach places a strong emphasis on optimizing real-time performance and front-end responsiveness. The research presents a pipeline that recognizes objects in video footage and seamlessly replaces them with corresponding 3D objects, enabling the scene to be efficiently re-rendered from alternative angles. Additionally, performance considerations are discussed.**

***Index Terms*—Video Frame Analysis, TensorFlow Video Processing, Video Editing, Machine Learning, JavaScript, Web Browser**

I. INTRODUCTION

In certain instances of (digital) filmmaking [1], it may only become apparent during the video post-production phase that the desired angle was not fully captured. Traditional methods of addressing such limitations often involve re-filming, which can be both costly and impractical, especially in scenarios where the original setup, lighting, or actors are unavailable. Our paper presents a pipeline for replacing an object in an existing video footage with a 3D model and rendering the modified scene from a new camera angle.

In this study, we employ advanced computer vision techniques [2] to identify and isolate target objects within source footage. This is achieved utilizing mainly TensorFlow, a widely recognized platform for machine learning (ML) and artificial intelligence (AI) [7], which is trained on labeled images. Subsequently, we apply various ML algorithms for depth estimation, enabling us to accurately position objects within a three-dimensional spatial framework. Additionally, we leverage 3D rendering methods to incorporate new models into the final scene, adjusting the compositional perspective while preserving textures from the replaced object.

Our approach has been subjected to rigorous testing on a diverse range of tree footage scenarios, encompassing both static and dynamic environments. This comprehensive evaluation has demonstrated the robustness and performance of the proposed pipeline. However, it is important to acknowledge certain limitations that may arise in specific scenarios, such as high-motion sequences, intricate shapes, or highly textured surfaces.

Paper organization: The following four sections provide detailed insights into the key components of this research.

- The next section discusses object tagging and model training using freely available solutions such as Keras[1], YOLOv11[2] and TensorFlow[3], focusing on the preparation and optimization of a ML model for detecting objects in video footage.
- The section II explores depth estimation and 3D object placement, highlighting techniques for integrating detected objects into a three-dimensional space.
- In the following section, texture replacement is presented, where identified objects are replaced with corresponding 3D textures to enable realistic rendering.
- Finally, the reminding section covers rendering the scene from another angle, showcasing the process of rendering modified video footage to achieve dynamic perspectives.

Moreover, our study delves into an analysis of performance for each section enumerated above.

The paper ends with related approaches (see Section VI) followed by several conclusions.

II. OBJECT TAGGING AND MODEL TRAINING WITH KERAS, YOLOV11 AND TENSORFLOW

To initiate the research, we assembled a comprehensive corpus of images that depicted instances of the target objects. For experimental purposes, we trained and validated the model on 50 images of trees. The experiments were predominantly conducted utilizing Google Chrome Canary[4] version 125.0.6379.3 on macOS. The execution environment comprised a Mac M1 Max equipped with 64 GB of RAM.

[1]Keras, an open-source library providing a Python interface for artificial neural networks – *keras.io* (Last accessed: May 23rd, 2025).

[2]Ultralytics YOLO11, a versatile solution for a wide range of computer vision tasks – *https://docs.ultralytics.com/models/yolo11/* (Last accessed: May 23rd, 2025).

[3]TensorFlow.js, a library for machine learning in JavaScript – *www.tensorflow.org/js* (Last accessed: May 23rd, 2025).

[4]Google Chrome Canary – *www.google.com/chrome/canary/* (Last accessed: May 23rd, 2025).

The performance of these studies may fluctuate based on factors such as the Web browser version, the underlying hardware, and the system's available memory.

Using Label Studio[5], we carefully configured a labeling interface that allowed for the precise annotation of bounding boxes around each tree (see also Figure 1). This meticulous process was repeated for every image in the dataset.

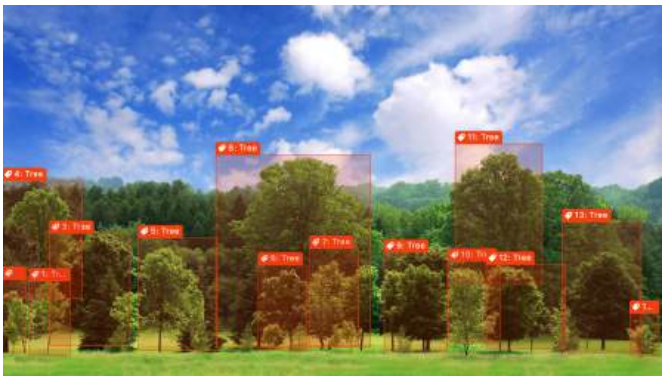

Fig. 1: Bounding box annotations in Label Studio for tree dataset

After finalizing the annotations, we exported them in a format compatible with the TensorFlow object detection pipeline.

Next, we parsed these annotations to construct the tensors containing the input images resized and normalized alongside their labels. In order to perform this, we defined a Keras-based architecture that combined a pre-trained feature extractor MobileNetV2 [8] with specialized output layers for bounding box regression and classification.

We subsequently trained the model using standard Keras training procedures, setting the learning rate to 1×10^{-4} and batch size of 2 as key parameters and reserving a portion of the dataset for validation to track and mitigate overfitting.

Finally, once the training process converged to stable loss and accuracy metrics, we saved the Keras model in the `.h5` format[6]. This export facilitated subsequent conversion to TensorFlow.js, enabling real-time inference within a Web browser.

Utilizing Python as a back-end technology, the integration of advanced data handling capabilities was achieved without compromising the responsive nature of the JavaScript-driven video interface. This approach enabled near-real-time inference, with typical latency remaining below 35ms, directly on the front-end (i.e. within the client's Web browser). A JavaScript program rendered video frames onto a `<canvas>` HTML element[7] and promptly transmitted them to the loaded model for immediate detection.

The model demonstrated a limitation in its recognition capacity, as it consistently identified only two trees within the video frames, even when the confidence threshold was set as low as 0.5.

Several factors could contribute to this limitation. The training dataset may have included insufficient examples of overlapping or closely spaced trees, resulting in reduced sensitivity for multiple detections in complex scenes. Furthermore, the architectural design of the model – particularly its bounding box regression component – may lack optimization for generating multiple candidate bounding boxes in scenarios involving closely positioned objects.

Fig. 2: Real time bounding box tree object detection

To advance the research, we investigated the application of YOLOv11 [3], [4] for tree detection. The model's performance was significantly improved in terms of both precision and inference speed through the augmentation of the annotated dataset. Specifically, we expanded the dataset to include 100 images, representing a substantial increase from the 50 images utilized in the prior experiment.

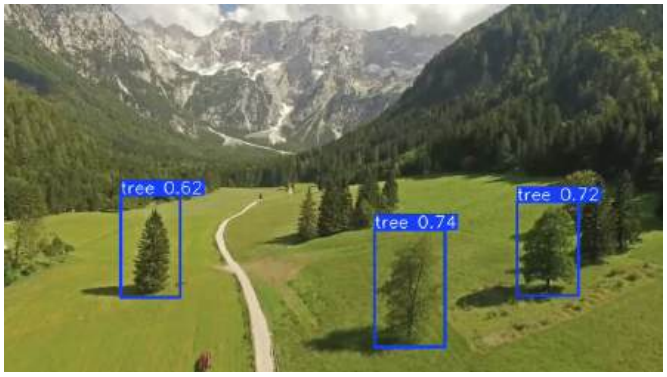

Fig. 3: YOLOv11 enhanced tree detection using bounding boxes

We initially trained the model using YOLOv11s, followed by YOLOv11m, to evaluate their respective performances with respect to tree detection. The two variants differ primarily in their architectural complexity and computational demands. `YOLOv11s` features fewer parameters and reduced depth

[5]Label Studio, a data labeling platform to fine-tune Large Language Models (LLMs), prepare training data or validate AI models – *labelstud.io* (Last accessed: May 23rd, 2025).

[6]`h5`, (Hierarchical Data Format 5), a versatile format, used by various libraries, able to store the entire model, including the architecture, learned parameters, and additional data used during the training – *www.hdfgroup.org/solutions/hdf5/* (Last accessed: May 23rd, 2025).

[7]`<canvas>`: the graphics canvas element – *developer.mozilla.org/docs/Web/HTML/Element/canvas* (Last accessed: May 23rd, 2025).

compared to its larger counterparts. Larger variants in the YOLOv11 family – such as `YOLOv11l` and `YOLOv11x` – were not employed due to the relatively small size of the dataset. While YOLOv11m demonstrated superior detection accuracy and recall, its larger architecture and increased capacity led to overfitting, resulting in a higher rate of false positives. In contrast, YOLOv11s delivered a more balanced performance, achieving a lower false positive rate and faster inference speeds (see Table I).

Model	Precision	Recall	False Positives
YOLOv11m	85.2%	94.8%	High
YOLOv11s	92.5%	85.3%	Low

TABLE I: Comparative performance of YOLOv11m and YOLOv11s in terms of precision, recall, and false positives.

These characteristics make YOLOv11s particularly suitable for real-time video deployment in Web browser-based applications, where efficiency is paramount.

The demonstrated low false positive rate and accelerated inference speeds of YOLOv11s render it particularly adept for integration into real-time video analytics systems. In the context of browser-based Web applications, as highlighted in the current study, the model's efficiency ensures seamless user experiences without compromising on detection reliability.

Frame	#Trees Found	Processing Time (ms)	Pre-process (ms)	Post-process (ms)
1	3	30.48	1.2	0.4
2	3	30.00	1.1	0.5
3	3	25.86	0.9	0.3
4	3	23.58	1.0	0.3
5	3	28.20	0.9	0.3
6	3	29.46	1.0	0.3

TABLE II: Frame processing time comparison showing trees detected, processing time, and breakdown of pre-processing and post-processing times for first six frames.

The performance evaluation of the frame processing pipeline demonstrates efficient inference capabilities, resulting in an average processing time of approximately 28.3 milliseconds across a subset of the initial six frames (see Table II). Pre-processing times remained minimal, averaging around 1.0 ms, which indicates a streamlined initial data handling process. Post-processing durations were equally stable, with an average of 0.4 ms.

These metrics highlight the pipeline's ability to deliver rapid and dependable performance, essential for real-time video-processing Web applications.

III. Depth Estimation

Depth estimation can be carried out at multiple levels of granularity, ranging from computing a dense map across the entire scene to focusing exclusively on the region defined by a previously detected bounding box. For performance purposes, we restrict depth analysis to a localized bounding box that concentrates resources on the object of interest.

To accomplish this task, we employed MiDaS, a convolutional neural network architecture trained on a comprehensive collection of depth datasets [5]. Initially, we converted the MiDaS model to the Open Neural Network Exchange (ONNX) format [6]. This conversion enables the model to be executed within a Web browser, thereby utilizing the client's local CPU resources without the need for a separate server-side pipeline. Subsequently, once a frame is captured from the video, the pixel data are transformed into tensors that conform to the MiDaS input specification.

Bounding boxes were annotated with depth heatmaps localized to the detected regions (Figure 4). A vertical gradient ruler was introduced adjacent to the video, visually encoding the depth range while providing corresponding numeric annotations for interpretability. Furthermore, the mean depth for each detected object was explicitly calculated and displayed, providing a precise statistical representation of object distances in the scene.

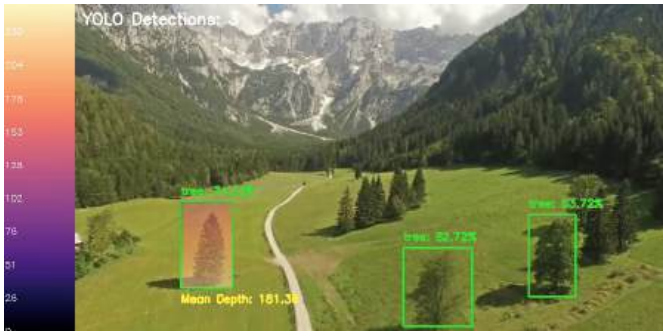

Fig. 4: A video frame showing YOLO-detected objects with labeled bounding boxes, depth heatmaps overlaying the detections, and a gradient ruler displaying depth scale on the side.

The process of rendering depth information for YOLO-detected objects is computationally intensive due to the use of FFmpeg library[8] for reconstructing video frames analyzed with MiDaS.

This workflow involves repeated decoding and encoding of frames, which imposes a significant processing load. We chose to do depth estimation to only one of the three YOLO model detected trees to optimize performance for the Web browser-based execution.

Pre-processing entails transforming raw video frames into a format compatible with model requirements, including resizing and normalization. Post-processing, in contrast, encompasses tasks such as bounding box refinement, overlaying depth heatmaps, and integrating semantic labels (consult also Table III).

[8]FFmpeg, an open-source cross-platform solution to record, convert and stream audio and video – *www.ffmpeg.org* (Last accessed: May 23rd, 2025).

Frame	#Trees Found	Processing Time (ms)	Pre-process (ms)	Post-process (ms)
1	3	45.2	1.5	0.8
2	3	43.4	1.1	0.4
3	3	49.4	1.1	0.8
4	3	53.8	2.8	0.4
5	3	57.8	3.4	0.7
6	3	84.1	1.0	0.3

TABLE III: Frame processing time comparison showing trees detected, processing time, and breakdown of pre-processing and post-processing times.

The system initiates with a video source and continuously captures individual frames in a loop (see Algorithm 1). For each successfully retrieved frame, the YOLO architecture is executed to perform precise object detection. Concurrently, the MiDaS model is utilized to generate a corresponding depth map, providing spatial context and depth information for the detected objects. The iterative process continues until frame capture is completed.

Input: Video source
Output: Processed frames with object detection and depth estimation
while *True* **do**
  ret, frame ← captureframe.read();
  if *not ret* **then**
    // If we cannot extract the frame, then we stop
    break
  else
    // Perform YOLO object detection
    objects ← YOLO.detect(frame);
    // Estimate depth using MiDaS
    depthMap ← MiDaS.estimateDepth(objects);
    // Display the results
    display(frame, objects, depthMap);
  end
end

Algorithm 1: Main Loop: YOLO object detection and MiDaS depth estimation

IV. Texture Replacement and 3D Objects Placement

In this approach, the three most dominant colors within the bounding boxes of detected objects are extracted using a frequency analysis of pixel color values. By reshaping the pixel data into a uni-dimensional array, the most frequent RGB (Red-Green-Blue) values are identified and ranked. These colors, represented in hex code, are then used as visual indicators alongside the bounding box (see Figure 5). The extracted color palette serves as a foundation for augmenting detected objects in 3D environments, enabling the replacement or mapping of the identified colors with textures in 3D models. This process bridges 2D video analysis with 3D reconstruction.

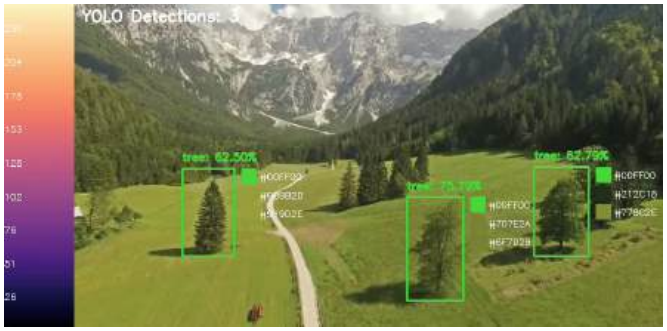

Fig. 5: Detected objects with top three dominant colors displayed as hex codes

We extend this approach by constructing a representative texture for each detected tree within the scene. This process involves reshaping the pixel data within each bounding box into a one-dimensional array, enabling the computation of frequency distributions for each unique pixel value (Figure 6). The most frequently occurring pixels are then systematically organized into a compact 10 by 10 pixels grid, effectively extracting the dominant texture characteristics of the detected tree. The construction of the 10×10 grid involves selecting the top 100 most frequent pixel values from the flattened one-dimensional array derived from the bounding box region of each detected tree.

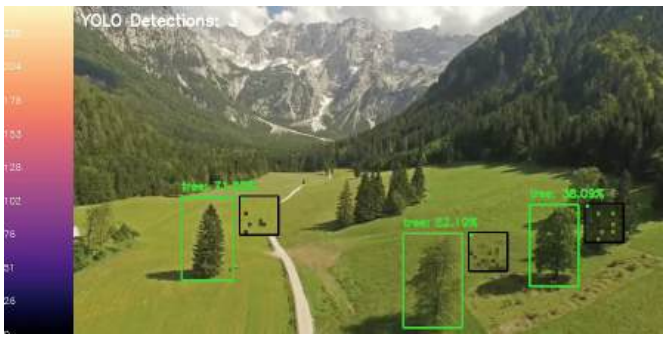

Fig. 6: Detected objects alongside their most frequent texture patterns displayed in a black border

The 3D objects are placed by first anchoring each tree's lowest Z-coordinate in the 3D scene. The bounding-box location and MiDaS depth estimate then set the virtual camera's distance. Finally, through alpha compositing, the rendered tree is blended into the video frame, creating semi transparent overlay that respects depth and placement of the real-world context and the visual appearance.

In our initial approach, we utilized the Three.js library[9] to perform real-time 3D rendering directly within the Web browser, but the computational overhead introduced to much latency. Consequently, we offloaded the complex 3D rendering

[9]Three.js: an open-source JavaScript 3D library – *threejs.org/manual/* (Last accessed: May 23rd, 2025).

workflow to a Python based back-end that leverages PyRender[10] for high fidelity RGB frames.

In addition to this, we transmitted an average depth value for each detected entity. This value serves as input to the virtual camera's distance parameter in PyRender, enabling the positioning of the 3D mesh. Furthermore, the material properties, lighting conditions, and camera orientation are also sent. On the client side, these pre-rendered frames are composited with the video stream, thereby enhancing performance. For instance, Figure 7 depicts a single frame from a real time stream, in which a 3D tree model is rendered according to the bounding box's depth and is composited onto the video.

Fig. 7: Frame representing a 3D tree model seamlessly blending into the video frame, depth aligned and color matched.

Most computationally intensive frames exhibit processing times ranging from approximately 106 ms to 167 ms. These particularly high latency frames introduce perceptible delays within the video processing pipeline, thereby increasing the probability of frame skipping. Performing the entirety of these operations on the front-end would further magnify this issue. The pre-processing and post-processing values remain comparatively modest, suggesting that neither data augmentation steps constitute major performance limitations (see Table IV).

Frame	#Trees Found	Processing Time (ms)	Pre-process (ms)	Post-process (ms)
19	3	167.4	7.1	0.4
2	3	158.5	6.5	0.5
21	3	127.6	1.3	1.1
17	3	127.0	2.1	0.4
27	3	114.1	1.2	3.4
9	3	106.6	1.9	0.8

TABLE IV: Most computationally intensive frames.

In contrasting these new measurements with those documented in Table III, it becomes apparent that the inclusion of the 3D rendering pipeline significantly increases overall latency. While the earlier dataset reported per-frame latencies largely in the range of 40-80 ms, the updated results now include frames that exceed 100 ms and, in some instances, approach 170 ms.

[10]PyRender, a pure Python library for physically-based rendering and visualization – *pyrender.readthedocs.io* (Last accessed: May 23rd, 2025).

V. Rendering Scene from Another Angle

By capturing a single RGB image and generating a corresponding depth map via MiDaS, we obtained a rough estimate of the scene's three dimensional structure. From this depth map, each pixel is reconstructed into a three dimensional space, creating an approximate 3D mesh. To simulate a different viewpoint, we introduced a virtual camera and adjust its translation and rotation relative to the reconstructed geometry. The mesh is then rendered from this virtual camera position, producing a new view that emulates a second, physically repositioned camera.

Fig. 8: Reconstructing a virtual camera from an RGB image using depth mapping and viewpoint generation.

Furthermore, we relocated the load to a Python back-end to accelerate the frame rendering. Despite these efforts, the overall performance continues to be a latency bottleneck, particularly when dealing with high-resolution images.

In our opinion, future optimizations, such as utilizing GPU acceleration or implementing a more efficient rendering pipeline, could substantially enhance the system's responsiveness.

The observed increase in pre-processing, processing, and post-processing times (Table V) is directly attributed to the computational demands introduced by MiDaS frame analysis.

Frame	#Trees Found	Processing Time (ms)	Pre-process (ms)	Post-process (ms)
1	3	152.40	2.40	0.60
2	3	150.00	2.20	0.75
3	3	129.30	1.80	0.45
4	3	117.90	2.00	0.45
5	3	141.00	1.80	0.45
6	3	147.30	2.00	0.45

TABLE V: Frame processing time comparison showing newly generated viewpoint

VI. Related Work

The generation of alternative object angles in video sequences has gained significant attention in recent years [10], with applications spanning virtual reality, augmented reality and video editing.

Object recognition, essential for replacing objects in video with 3D models, relies on models like YOLO [3] for accurate detection and segmentation.

Several pragmatic considerations regarding 3D frame generation are discussed in [11], but that research does not concern Web applications. A recent study introduces a novel framework tailored for mobile devices, but is not directly focused on frame generation in the Web browser. This framework seamlessly integrates frame generation with super resolution to significantly enhance real-time rendering performance [12].

Research by [13] introduced a hybrid model combining depth-based synthesis with neural rendering, achieving impressive results in video reconstruction tasks, although computational demands remain high. Also, the modeling and re-rendering dynamic 3D scenes by using artificial neural networks is studied by [9].

Depth-based methods for view modification utilize the geometric structure of a scene to simulate alternative perspectives. Works focused on stereo magnification [14], [15] and layered depth video [16] leverage estimated depth maps to reconstruct intermediate views.

Efforts in improving real-time performance have also been explored with focus on real-time 3D scene reconstruction from monocular video streams [17]. These methods emphasize efficient rendering pipelines to bridge the gap between offline and real-time processing for virtual reality and augmented reality applications. Recent complex approaches, although not yet suitable for efficiency on the Web browser side, include – among other proposals – self-supervised 3D scene graph learning [18], multi-modal feature fusion and global–local attention [19], or multi-camera unified pre-training [20].

VII. Conclusion and Further Work

This research study presents evidence that object segmentation and depth estimation can be integrated with three-dimensional models to modify video frames, thereby generating novel perspectives. While these advancements have been made, challenges persist, particularly in managing intricate textures and high-motion scenes, which can compromise the accuracy of object reconstruction and scene rendering.

Expanding the scope of the training data can enhance adaptability across diverse scenarios. Additionally, reducing latency remains a critical challenge, and improving depth reconstruction by incorporating information from multiple frames can lead to greater spatial consistency. Optimizing the rendering pipeline is another crucial area for further development. More efficient scene reconstruction methods can significantly reduce computational overhead.

Beyond performance enhancements, integrating this pipeline into immersive environments presents a promising direction for future research, particularly in applications such as augmented reality (AR) and virtual reality (VR) on the Web. Facilitating real-time manipulation of scene composition would significantly enhance creative control, enabling users to dynamically adjust object placement, modify textures, and alter perspectives in an interactive manner.

In summary, this paper demonstrates the practicality of generating diverse object viewpoints within video frames, which presents significant opportunities for applications in video editing and augmented content creation within a Web browser environment.

References

[1] D. Case, *Film Technology in Post Production*. Routledge, 2013.

[2] G. Stockman, and L. G. Shapiro. *Computer Vision*. Prentice Hall PTR, 2001.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once (YOLO): Unified, real-time object detection", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), IEEE Press, 2016, pp. 779–788.

[4] R. Khanam, M. and Hussain, M., "Yolov11: An overview of the key architectural enhancements", *arXiv preprint*, arXiv:2410.17725, 2024.

[5] R. Birkl, D. Wofk, and M. Müller, "MiDaS v3.1 – a model zoo for robust monocular relative depth estimation", *arXiv preprint*, arXiv:2307.14460, 2023.

[6] P. Jajal et al., "Interoperability in deep learning: a user survey and failure analysis of ONNX model converters". in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ACM Press, 2024, pp. 1466–1478.

[7] Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems", *arXiv preprint*, arXiv:1603.04467, 2016.

[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, 'MobileNetV2: Inverted Residuals and Linear Bottlenecks', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), IEEE Press, 2018, pp. 4510–4520.

[9] A. Cao, A. and J. Johnson, J. "Hexplane: A fast representation for dynamic scenes". in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (CVPR), IEEE Press, 2023, pp. 130–141.

[10] X. Li, X. et al., "Advances in 3D generation: A survey", *arXiv preprint*, arXiv:2401.17807, 2024.

[11] N. Ray, D. Sokolov and B. Lévy, "Practical 3D frame field generation", *ACM Transactions on Graphics*, Volume 35, Issue 6, ACM Press, 2016, pp.1–9.

[12] S. Yang, et al. "Mob-FGSR: Frame generation and super resolution for mobile real-time rendering". *ACM SIGGRAPH 2024 Conference Papers*, ACM press, 2024, pp. 1–11.

[13] A. Chen et al., "MVSNeRF: fast generalizable radiance field reconstruction from multi-view stereo", in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (CVPR), IEEE Press, 2021.

[14] T. Zhou, P. P. Srinivasan, J. Flynn, J. T. Barron, A. A. Efros, and J. Malik, "Stereo magnification: Learning view synthesis using multiplane images", *ACM Transactions on Graphics*, Volume 37, ACM Press, 2018. pp. 1–12.

[15] X. Wang et al., "Multi-view stereo in the deep learning era: A comprehensive review", *Displays*, Volume 70, Springer, 2021, pp.102–102.

[16] Y.-Y. Shih, W.-S. Chiu, R. Garg, H. Wang, M. Nguyen, and J.-B. Wang, "Layered depth video for novel view synthesis", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (CVPR), IEEE Press, 2020. pp. 6196–6205.

[17] L. Wang et al., "DistillNeRF: Perceiving 3D Scenes from Single-Glance Images by Distilling Neural Fields and Foundation Model Features", *arXiv preprint*, arXiv:2406.12095, 2024.

[18] S. Koch, P. Hermosilla, N. Vaskevicius, M. Colosi T. and Ropinski, "Sgrec3d: Self-supervised 3d scene graph learning via object-level scene reconstruction". in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, IEEE Press, 2024, pp. 3404–3414.

[19] D. Liu, Z. Wang and P. Chen, P. "DSEM-NeRF: Multimodal feature fusion and global–local attention for enhanced 3D scene reconstruction". *Information Fusion*, Volume 115, Elsevier, 2025, pp. 102–752.

[20] C. Min, L. Xiao, D. Zhao, Y. Nie and B. Dai, "Multi-camera unified pre-training via 3d scene reconstruction", *IEEE Robotics and Automation Letters*, Volume 9, Issue 4, IEEE Press, 2024, pp. 3243–3250.