TED UNIVERSITY

FALL 2024

CMPE 492

Low Level Design Report

Project Name

3DifyMe

Project Supervisor

Gökçe Nur Yılmaz

Project Team Members

Hamza Emin Hacıoğlu

Deniz Caner Akdeniz

Bengisu Tuğrul

Elif Arslan

Project Jury Members

Müslim Bozyiğit

Tolga Kurtuluş Çapın

Contents

1.Introduction

3DifyMe is a desktop application that transforms 2D videos and live camera feeds into 3D experiences. By using color theory and artificial intelligence (AI) techniques, the application enhances depth perception and creates the illusion of three-dimensional space.

Designed with a user-friendly interface, 3DifyMe offers users an experience in transforming traditional 2D media into visually stunning 3D content. The system combines advanced technology with accessibility, providing an easy way to explore new dimensions in media.

1.1.Object Design Trade-Offs

- Performance vs. Visual Quality:

  The primary trade-off lies in ensuring smooth performance without sacrificing the quality of the converted 3D content. The system leverages GPU-accelerated processing for computationally heavy tasks, such as generating depth maps and applying filters, which significantly reduce the time taken to convert each frame. However, this reliance on GPU processing may create compatibility issues for users with lower-end devices or those without dedicated graphics hardware. Instead of altering the algorithms, the system will address these issues through the following methods:

- Algorithm Complexity vs. Usability:

  Advanced algorithms like deep learning algorithms and depth estimation techniques can produce more realistic depth maps, improving the overall 3D effect. However, these algorithms are computationally expensive, which could lead to increased latency or frame drops during real-time processing, affecting usability. To address this, the system provides the ability to scale down the complexity of these algorithms depending on conditions. Users can choose for a lower-fidelity conversion in real-time to ensure smooth performance.

- Energy Efficiency vs. Continuous Processing:

  For devices like laptops or mobile systems, continuous real-time processing of live streams can lead to high energy consumption and battery drain, particularly when GPU acceleration is involved. To mitigate this, the system includes adaptive processing, which adjusts the conversion intensity based on the current battery level or power mode. This trade-off ensures that users can maintain a balance between processing quality and energy efficiency, especially for extended usage.

In the design of 3DifyMe, various trade-offs must be considered to achieve a balance between performance, usability, and energy efficiency. By leveraging techniques such as dynamic resolution scaling and adaptive processing, the system ensures smooth real-time 3D conversion while maintaining visual quality across different hardware configurations. Additionally, energy-efficient processing modes allow for extended usage on mobile devices without compromising functionality. These design decisions ensure that 3DifyMe delivers an optimized and adaptable user experience without requiring significant changes to its core algorithms.

1.2. Interface Documentation Guidelines

The documentation for 3DifyMe follows best practices to ensure that developers and users can interact effectively with both the front-end interface and the back-end system. The guidelines include:

- Clear Naming Conventions:
  All functions, methods, and UI elements are clearly named to reflect their purpose. For example, buttons like "Start Live 3D Conversion" or "Adjust Depth" provide immediate clarity to users and developers working on the project.

- Comprehensive API Documentation:
  For developers, each API endpoint and function is documented with clear descriptions of inputs, outputs, and expected behavior. This includes specifying data types, handling exceptions, and including example requests and responses.

- Graphical User Interface (GUI) Documentation:
  The GUI elements, such as buttons, sliders, and menus, are documented with descriptions of their function and intended use. For each component, documentation includes:
  - Functionality description.
  - Required user inputs and the expected behavior.
  - User permissions or roles associated with accessing certain features.
- Tooltips and On-screen Help:
  Integrated tooltips within the GUI offer users quick explanations of various controls. Additionally, a "How to Use" section provides detailed guidance on converting videos and adjusting settings, allowing users to understand the process step-by-step.

The interface documentation for 3DifyMe ensures a seamless experience for both developers and users by adhering to clear naming conventions, providing comprehensive API documentation, and detailing the functionality of the GUI elements. The inclusion of tooltips and a “How to Use” section enhances user understanding, while the well-documented back-end and front-end interactions ensure maintainability and ease of development. These guidelines enable efficient collaboration and improve user engagement with the system.

1.3. Engineering Standards

In creating this low-level design report, we aimed for clarity, simplicity, and adherence to industry best practices in software engineering. To achieve these goals, we followed UML (Unified Modeling Language) guidelines and key IEEE standards.

UML, often seen as the blueprint of software systems, was used extensively throughout our design process to maintain consistency and clarity. It allowed us to visually represent the architecture and class-level interactions, ensuring that all components of the software are well-structured and easy to understand. UML is also widely taught and recognized in academia and industry for its user-friendly approach, making it an essential tool in our documentation.

Additionally, we adhered to several IEEE standards to guide the various aspects of our design. Specifically, we followed IEEE 830-1998 for software requirements specifications and IEEE 1016-1998 for software design descriptions, ensuring that our design process aligns with broader software engineering practices. These standards helped us structure our documentation in a way that is both rigorous and accessible, contributing to the overall quality and comprehensibility of the low-level design.

1.4. Definitions, acronyms, and abbreviations

This section provides key definitions, acronyms, and abbreviations used throughout the 3DifyMe project to ensure clarity and consistency. By defining important technical and project-specific terms, such as "2D", "3D", and "Monocular Depth Estimation," this section serves as a reference for developers, users, and stakeholders. Understanding these terms is essential for interpreting the system's functionalities, methodologies, and design principles, as they play a critical role in both the technical and user-facing aspects of the project.

- 2D (Two-Dimensional):

A media format that has only height and width dimensions. 2D media lacks a depth dimension, typically resulting in a flat appearance. Examples include photographs and traditional videos.

- 3D (Three-Dimensional):
  A media format that includes height, width, and depth dimensions. 3D content provides viewers with a sense of depth, offering a more realistic and engaging experience. Content created using 3D modeling and animation techniques falls into this category.

- GUI (Graphical User Interface):
  A visual interface that allows users to interact with electronic devices through graphical elements such as buttons, sliders, and menus. The GUI in the 3DifyMe application enables users to perform tasks such as uploading videos and starting transformations.

- API (Application Programming Interface):
  A set of functions and protocols that allow interaction between the application and other software components. APIs help developers extend the functionality of the application and integrate with other systems.

- Real-Time Processing:
  The capability to process and output data almost instantaneously as it is received. The 3DifyMe application performs real-time processing of live camera streams, providing instant results to users.

- Depth Layers:
  Additional layers added to an image to create the illusion of depth. Depth layers enhance the three-dimensional appearance of 2D images.

- Color Theory:
  The study of how colors relate to each other and how they affect human perception. Color theory is utilized in 3D transformation to enhance depth perception through strategic color placement and manipulation.

- Deep Learning:
  A subset of machine learning that involves neural networks with multiple layers, enabling the model to learn from large amounts of data. 3DifyMe employs deep learning techniques to convert 2D content into 3D.

- User-Friendly:

Refers to an interface or application that is easy to understand and navigate, promoting a positive user experience. The 3DifyMe application is designed to be user-friendly, allowing users to interact effortlessly.

- Event-Driven Architecture:
  A software design pattern where events trigger actions within the system. The 3DifyMe application adopts this approach, defining user interactions and background image processing steps as events.

- Global Software Control:
  A central mechanism that manages the flow of data and coordinates actions between different parts of the software system. This control mechanism ensures smooth operation and responsiveness of the application.

- Optical Flow:
  A technique for estimating the motion of objects in a video sequence. This is useful in the 3D transformation process for tracking moving objects and ensuring a consistent depth effect.

- Camera Control:
  A feature that allows users to start and stop live video streams, as well as toggle mirror views. This enhances the user experience within the application.

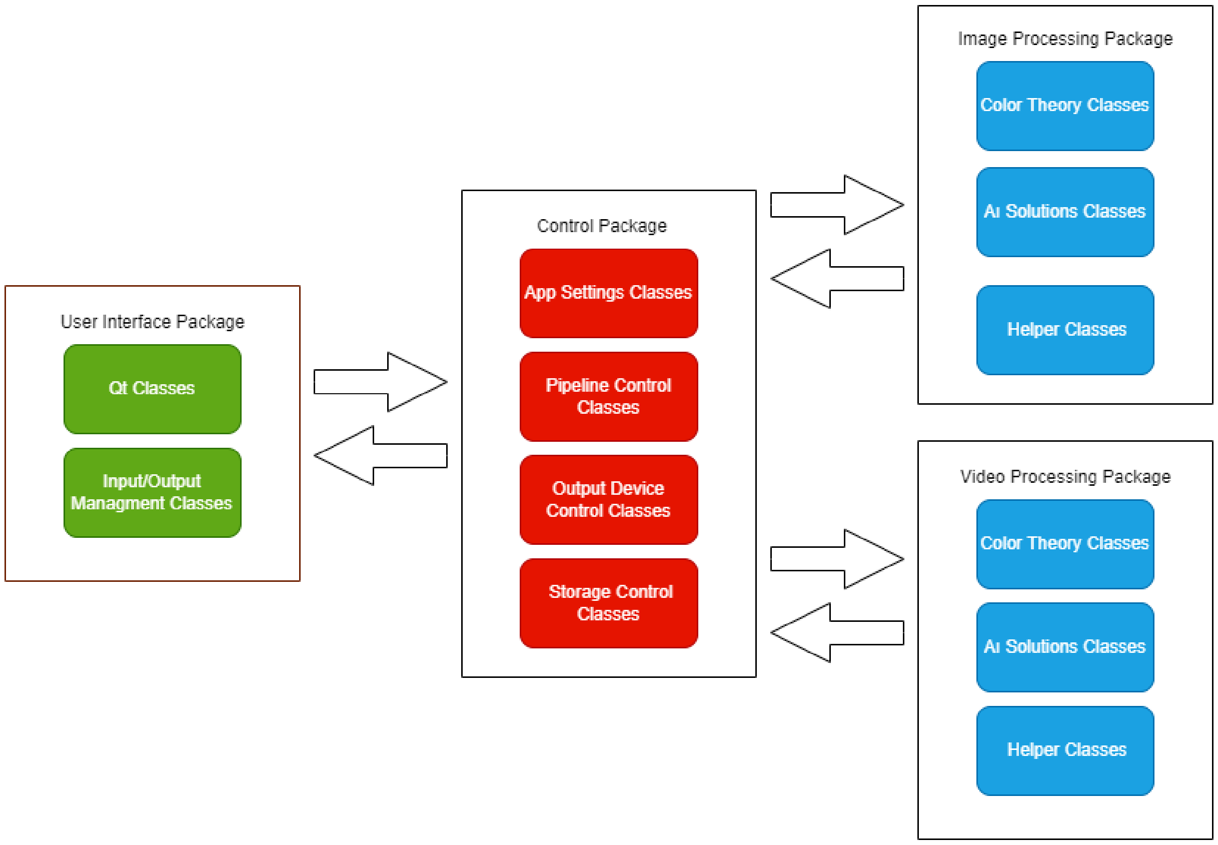
- Offline Processing:
  A feature that allows users to convert 2D videos to 3D without needing an internet connection. This ensures that users can utilize the application even with limited or no internet access.

- Settings Management:
  The ability to save user preferences and application settings, ensuring a consistent user experience across sessions. This feature allows users to maintain their preferred configurations.

- Performance Optimization:
  Improvements made to ensure the application runs efficiently on various hardware configurations. This includes adaptive quality settings that automatically adjust based on the user's device capabilities.

2. Packages

The 3DifyMe application is organized into several packages, each responsible for distinct aspects of the system's functionality. These packages ensure modularity, facilitate maintenance, and provide a clear separation of concerns across different components of the application.



Explanation of the image:

1. User Interface Package

This package is responsible for managing the application's front end, including input and output handling.

Subcomponents:

Qt Classes: These classes handle the user interface using the Qt framework, providing the layout and interaction logic for the application.

Input/Output Management Classes: These classes manage the flow of data between the user and the system, handling tasks like reading video or image inputs and displaying processed outputs.

2. Control Package

This package manages the core control logic of the application, governing the pipeline, settings, and connected devices.

Subcomponents:

App Settings Classes: Responsible for handling the configuration and management of application settings, including resolution, frame rate, and output format.

Pipeline Control Classes: These classes oversee the flow of data from input to output, ensuring that all processes are executed in the correct sequence.

Output Device Control Classes: Responsible for managing output devices like cameras, speakers, or displays, ensuring that the processed media is presented correctly.

Storage Control Classes: These classes handle the saving and retrieval of processed data, managing storage locations and formats.

3. Video Processing Package

This package focuses on transforming 2D video inputs into 3D using various techniques, including color theory and AI-based methods.

Subcomponents:

Color Theory Classes: These classes implement traditional 3D visualization techniques based on color separation (e.g., red-cyan filtering).

AI Solution Classes: Leverages deep learning models to enhance 3D video processing, offering more advanced and accurate depth estimation and transformation.

Helper Classes: Contains utility functions and additional methods that support the core video processing tasks, including video frame manipulation and pre/post-processing.

4. Image Processing Package

Similar to the Video Processing Package, this package applies 3D transformation techniques to images. Given the lower computational complexity, this package also includes custom algorithms to enhance the 3D effects in images.

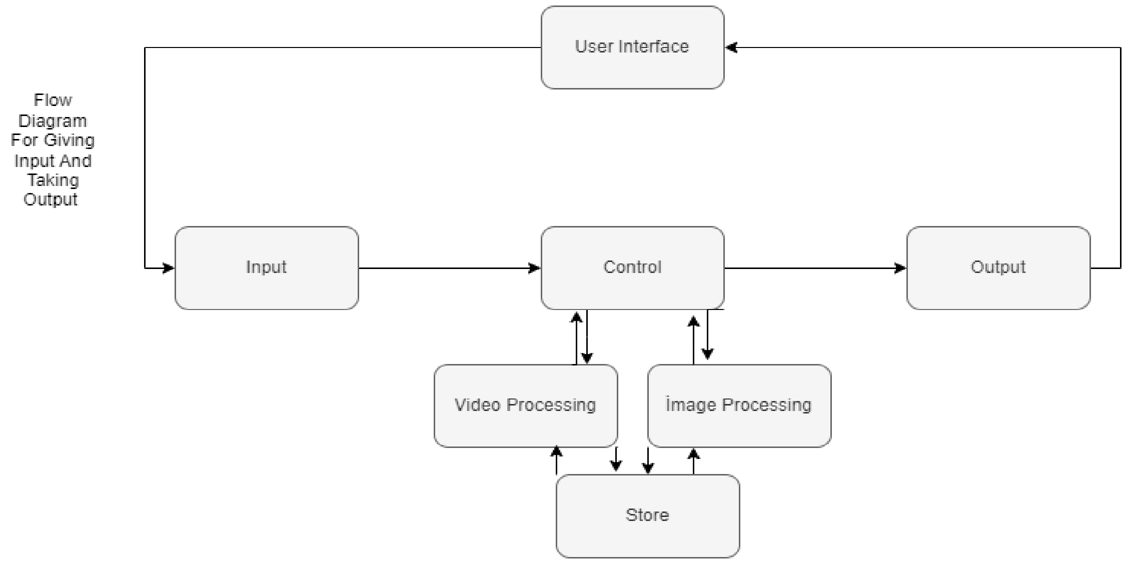
Subcomponents:

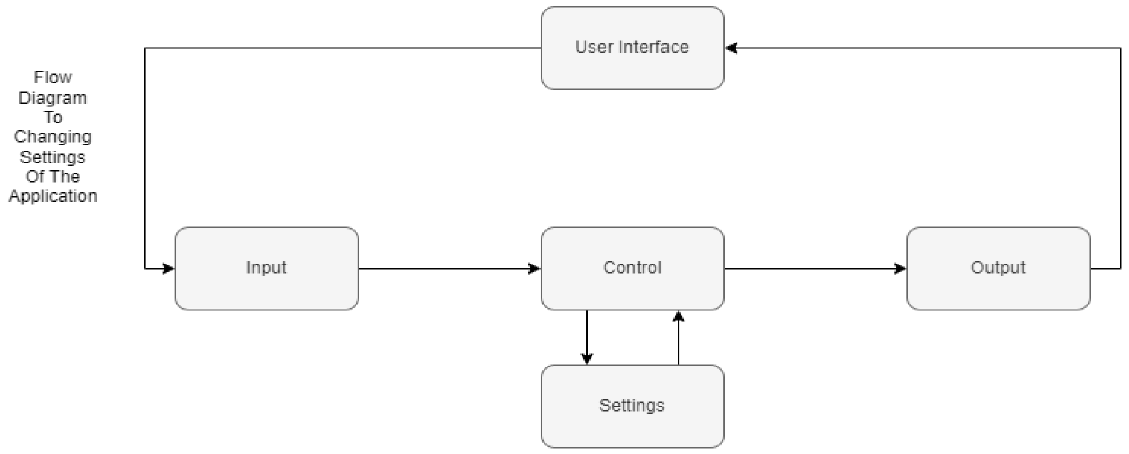
Color Theory Classes: Implements classical 3D effects through color manipulation, similar to the video processing package but optimized for static images.

AI Solution Classes: Uses AI-driven models to estimate depth and improve the realism of 3D image outputs.

Helper Classes: Contains additional utility methods, specifically designed for image processing tasks like resizing, color correction, and format conversion.

Also to clarify our packages and show how our application works with the help of these packages, we are sharing some flow diagrams.





Explanation of the image: The image contains two flow diagrams representing key processes within the application.

First Diagram:
The first diagram illustrates the flow when the user provides input to the system. Once the input is received, it is managed by the Control Package, which verifies and handles the input.

Depending on the type of input, the system will determine whether it is a video or an image. Based on this decision, the relevant processing pipeline (either video or image processing) is applied. After the processing is complete, the system offers the user an option to save the processed output. Finally, the output is displayed to the user via the User Interface.

Second Diagram:
The second diagram demonstrates what happens when the user modifies the application settings. The user input is received, and after passing through the necessary control checks, the system applies the changes to the settings. Once the settings have been updated, the modified output is displayed to the user through the User Interface.

!!! There is an important note that we need to clarify before moving on to the class interface part.

You will see that in the class interface section, the class names are clear and specific. After completing the implementation of our project, we will organize those classes into their respective subpackages according to their purposes, as shown above. Therefore, please do not consider the classes below irrelevant to what we have presented earlier.

3. Class Interfaces

3.1. Class 1: FrameProcessor

The FrameProcessor class handles the extraction and preparation of video or image frames. It ensures that each frame is correctly formatted and suitable for further processing in the 3D transformation pipeline. This component is crucial, as the subsequent steps rely on having the input in an optimal form, ensuring no data is lost or distorted during the initial extraction.

Attributes:

Frame: This attribute stores the current frame from the input video or image. It is essential to hold the frame in memory during processing so that operations like depth analysis or color transformation can be applied efficiently.

formatType: This attribute defines the format of the current input (e.g., JPG, PNG, MP4). It ensures that the processor knows how to handle the file correctly, converting the frame into a usable format for later steps, such as converting raw video streams to image arrays.

frameIndex: This attribute stores the index of the current frame being processed, allowing for sequential and time-stamped handling of frames, which is especially important for video inputs.

Methods:

extractFrame(input): This method is responsible for extracting individual frames from a video or retrieving an image from a static input. It also converts the frame to a standardized format, ensuring compatibility with subsequent processes.

validateFrameFormat(): This method checks the format of the input and ensures it matches the system requirements for processing. If the format is invalid, the system either converts it or raises an exception.

preprocessFrame(): This method performs any necessary preprocessing on the frame, such as resizing, color normalization, or noise reduction, ensuring that the frame is in optimal condition before deeper analysis.

getNextFrame(): For video inputs, this method moves to the next frame in the sequence and updates the Frame and frameIndex attributes accordingly.

3.2. Class 2: SettingsManager

The SettingsManager class allows the user to define and customize how the frames should be processed and how the final output should appear. It provides flexibility for different outputs, including resolution adjustments and color space configurations.

Attributes:

resolution: This attribute stores the user-defined resolution for the output frames or videos (e.g., 4K, 1080p). Higher resolutions may require more computational power, but they offer greater image detail.

frameRate: Specifies the frame rate for video outputs. The frame rate impacts how smooth the final video will appear, with higher frame rates offering smoother motion at the cost of increased processing.

outputFormat: Stores the format (e.g., MP4, JPG) of the final processed output. Different formats offer varying compression and quality levels.

colorDepth: Defines the color depth (e.g., 8-bit, 16-bit) for the output, impacting the richness of colors in the final output.

customSettings: A dictionary or map containing any additional settings (e.g., custom file paths, watermark options).

Methods:

setResolution(resolution): Allows the user to select the desired resolution for the output, dynamically updating how frames are processed (e.g., upscaling or downscaling).

configureFrameRate(rate): Adjusts the frame rate for the final output. The system adapts its processing to ensure the desired frame rate is met.

setOutputFormat(format): Configures the format for the output file. Each format (e.g., MP4, AVI, PNG) has different characteristics, and this method allows the user to select the most appropriate one.

applyCustomSettings(customSettings): This method takes user-defined settings and applies them to the system, offering full control over the output's parameters, such as aspect ratio, bit rate, or even encoding profiles.

validateSettings(): Ensures that the combination of settings selected is feasible for processing, avoiding scenarios where, for example, a high frame rate is selected alongside an incompatible output format.

3.3. Class 3: RedCyanProcessor

The RedCyanProcessor class is responsible for generating an anaglyph 3D image from the input frame by manipulating the red and cyan channels of the image. This process helps create a 3D effect that can be viewed with red-cyan glasses.

Attributes:

redChannel: This attribute stores the red channel of the image, which is used as the left-eye perspective in the final 3D output.

cyanChannel: Stores the cyan channel, which represents a combination of the blue and green channels of the image, used as the right-eye perspective in the 3D output.

anaglyphImage: Holds the final anaglyph image, a combination of the red and cyan channels designed to be viewed with red-cyan glasses.

Methods:

separateChannels(videoFrame): This method separates the video frame into its individual color channels, specifically extracting the red and cyan components.

applyRedCyanShift(): This method applies a shift to the red and cyan channels to create the illusion of depth when combined.

combineChannels(): Combines the red and cyan channels into a single anaglyph image, producing the final 3D effect.

renderAnaglyph(): Generates the final anaglyph image, combining the red and cyan shifted channels for 3D viewing.

previewAnaglyph(): Provides a preview of the generated anaglyph image for real-time adjustments before final output.

3.4. Class 4: MidasDepthEstimator

The MidasDepthEstimator class uses the Midas model to estimate the depth of objects within a video or image frame. This component produces a depth map, which is essential for adding realistic depth effects to 2D media.

Attributes:

Frame: Stores the current frame being analyzed for depth estimation.

depthMap: A matrix or image that represents the depth information for each pixel in the frame, with closer objects having smaller depth values and distant objects having larger values.

model: The Midas model used for depth estimation. It holds the pre-trained model data for generating depth maps.

Methods:

generateDepthMap(videoFrame): This method applies the Midas model to the video frame, generating a depth map that represents the spatial distance of objects in the frame.

refineDepthMap(colorData): This method further refines the depth map by considering color data from the frame, which helps in distinguishing objects in complex scenes.

applyPostProcessing(depthMap): Enhances the depth map by applying smoothing or interpolation techniques to remove noise or sharp transitions between depth values.

exportDepthMap(): Exports the generated depth map as an image or data file for use in further processing.

3.5 Class 5: DepthBasedReconstruction(Reimagining)

The DepthBasedReconstruction class is responsible for reconstructing a 3D image or video from the depth information and applying specific algorithms to manipulate the depth and perspective using RedCyanProcessor.

Attributes:

depthMap: The depth data generated by Midas, stored as a matrix where each pixel has a corresponding depth value.

angleMap: Stores the angles assigned to different parts of the image or video based on their depth, allowing for perspective shifts as the depth increases or decreases.

reconstructedImage: Holds the final reconstructed 3D image or video, which incorporates depth and perspective shifts.

Methods:

applyDepthBasedAngles(depthMap): Based on the depth information from the depthMap, this method assigns different angles to each part of the image. The further away an object is, the more its angle is adjusted to simulate 3D depth.

generateParallaxEffect(): Simulates the parallax effect by adjusting the positions of objects in the image based on their depth, giving the illusion of 3D movement when viewed from different angles.

compose3DImage(): Combines the depth information and angle adjustments into a single, cohesive 3D image or video.

finalize3DRendering(): This method performs the final rendering of the 3D image or video, ensuring that all depth and angle adjustments have been properly applied and that the output is ready for export.

4.Glossary

AI Solution	Deep learning-based methods for enhancing 3D processing.
App Settings	Configurations like resolution, frame rate, and output format.
Color Depth	The number of bits used to represent the color of each pixel, affecting the range of colors displayed.
Color Theory	Techniques for creating 3D effects using color separation.
Cyan Channel	A combination of blue and green channels used for the right-eye view in 3D anaglyphs.
Databases	Organized collections of data (e.g., 3D video, image, or live streams) used for training or processing in machine learning and 3D rendering systems.
Deep Learning Model:	A type of neural network model that uses multiple layers of processing units to learn from large amounts of data, used for complex tasks like 3D image predictions.
Depth Map	A representation of depth in an image, indicating how far each pixel is from the viewer.
Flow Diagram	Visual representation of how the system processes input and displays output.
Frame	A single still image in a sequence that makes up video content.
Frame Rate	The number of frames displayed per second in a video, determining motion smoothness.
Helper Classes	Utility methods for tasks like frame manipulation and pre-processing.
Image Processing Package	Applies 3D transformation techniques to images, similar to video processing.
Optical flow	Refers to optical flow, which tracks the movement of objects between consecutive frames in a video, capturing motion and changes in position.
Output Device Control	Handles hardware like cameras or screens.

Parameters	Variables in a model that are learned from the data during training, which influence the performance and behavior of the model.
Parallax Effect	The perceived movement of objects at different depths when the viewpoint changes, simulating 3D.
Pipeline Control	Ensures correct data flow from input to output.
Post-Processing	Techniques applied after initial processing to enhance or refine the image or video output.
Pre-Trained Model	: A machine learning model that has been previously trained on a large dataset and is used to perform tasks like classification or prediction without retraining.
Qt Framework	A cross-platform framework used to build the graphical user interface.
Red Channel	The part of an image representing red color information, used for the left-eye view in 3D anaglyphs.
Resolution	The number of pixels in an image or video, affecting detail and clarity (e.g., 4K, 1080p).
Stereo Video	A video format that contains two separate images, one for the left eye and one for the right eye, to create a 3D effect when viewed through a stereoscopic device
Stereoscopic	Refers to the technique of creating the illusion of depth in an image or video by presenting two offset images separately to the left and right eye (3D effect).
Storage Control	Manages saving and retrieving processed data.
User Interface (UI)	Manages user interactions, built with the Qt framework.
Video Processing Package	Transforms 2D videos into 3D using color theory and AI methods.

5. References

- IEEE. "IEEE Homepage." Available: https://www.ieee.org. Accessed: Oct. 13, 2024.
- Wikipedia. "2D to 3D Conversion." Available: https://en.wikipedia.org/wiki/2D_to_3D_conversion. Accessed: Oct. 13, 2024.
- Wikipedia. "Parallax." Available: https://en.wikipedia.org/wiki/Parallax. Accessed: Oct. 13, 2024.
- Wikipedia. "Stereoscopy." Available: https://en.wikipedia.org/wiki/Stereoscopy. Accessed: Oct. 13, 2024.
- Wikipedia. "Depth Map." Available: https://en.wikipedia.org/wiki/Depth_map. Accessed: Oct. 13, 2024.
- Leibe, B., Matas, J., Sebe, N., & Welling, M. (Eds.). (2016). 14th European Conference on Computer Vision, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV (Vol. 9908, Lecture Notes in Computer Science). Springer. https://doi.org/10.1007/978-3-319-46493-0.
- Mehl, L., Bruhn, A., Gross, M., & Schroers, C. (Year). "Stereo Conversion with Disparity-Aware Warping, Compositing, and Inpainting." Institute for

Visualization and Interactive Systems, University of Stuttgart; Computer Graphics Lab, Department of Computer Science, ETH Zurich; Disney Research | Studios, Switzerland.