Scientific Visualization, 2025, volume 17, number 1, pages 65 - 85, DOI: 10.26583/sv.17.1.06

Application of PyTorch3D and NERF Computer Vision Tools for Building a Point Cloud of a Three-Dimensional Model and Determining the Camera Position of Still Images in Space

V.V. Konkov[1], A.B. Zamchalov[2]

Institute of Intelligent Cybernetic Systems, National Research Nuclear University MEPhI, Moscow, Russian Federation

[1] ORCID: 0009-0005-1197-2248, vlad.konkov.7145@gmail.com
[2] ORCID: 0009-0006-0955-1062, andreizam@yandex.ru

Abstract

Recently, computer graphics plays a key role in solving computer vision problems. The problem of converting 2D images into 3D models continues to be urgent, as it requires precise determination of camera position and construction of accurate 3D models of objects. Traditional methods are often limited in application and do not offer a comprehensive solution. This study examines the use of PyTorch3D and NERF libraries to determine the camera position in 3D space and create a 3D model of an object from a single 2D image. As a method of data preparation, a hardware and software system was used, including a stepper motor control device that provides manual and sequential positioning of the camera and its return to the initial position, a shooting control system to generate a comprehensive set of photos at each camera position, and a mechanism for sending data to a remote computer for further processing. The PyTorch3D library was selected during the study to explore the possibilities of converting 2D images into 3D models or determining the position of an object in the photos. The processing process included several steps: building a point cloud to generate a 3D volumetric model of the object, determining the camera position in 3D space from a single 2D image using inverse problem algorithms, and constructing a 3D object using differentiable rendering, creating 3D voxels and 3D meshes. The results of this study showed successful determination of camera position in 3D space and construction of a 3D object model from a single 2D image, demonstrating the advantages of using the PyTorch3D library over other existing models. These findings can be applied in the development of software and hardware systems for creating 3D images from 2D photographs. The study confirmed the relevance and effectiveness of using PyTorch3D library to solve the problems of converting 2D images into 3D models. Further work will be aimed at expanding the functionality of the system and its use in various areas of computer vision.

Keywords: computer vision; PyTorch3D; NERF; 3D modeling; camera positioning, point cloud; deep learning; 3D reconstruction; differentiated rendering.

1. Introduction

Computer vision is one of the most dynamically developing areas of artificial intelligence, finding application in many areas from medical diagnostics to autonomous driving. In recent years, the problem of converting two-dimensional (2D) images into three-dimensional (3D) models has become especially urgent. This is due to the fact that 3D models allow for a more accurate analysis and interpretation of the environment. In this context, determining the exact camera position and building high-quality 3D models of objects are key tasks. Traditional methods used for these purposes often face limitations related to accuracy and scalability.

The introduction of new technologies and approaches, such as the PyTorch3D library, opens up new possibilities for solving these problems. This research aims to explore the potential of using PyTorch3D to determine camera position and create 3D models of objects based on a single 2D image. As part of the work, a hardware and software system was used, including stepper motor control for precise camera positioning, a shooting control system, and a mechanism for transferring data to a remote server for further processing.

The main objective of this study is to evaluate the accuracy and efficiency of the proposed method in comparison with traditional approaches, as well as to identify potential areas of application of the developed solution.

Purpose of the work: study of 2D and 3D image processing methods using the capabilities of PyTorch3D library.

Additionally, the following tasks were solved:

In terms of preparation of initial data (photos) of the object without using synthetic datasets

- task of stepper motor control with the possibility of manual and sequential positioning of the camera near the object and returning the camera to the initial position;
- task of shooting control to form a complex set of raw data (photos) at each camera position near the object for further processing;
- The task of sending to a remote computer for processing;
- Generation of .obj model of the object.

In terms of object detection on 2D image based on OBJ model using PyTorch3D (inverse problem):

- building a point cloud to generate a 3D volumetric model;
- determining the camera position in 3D space from a single photo;
- determining the position of a 3D object in a photo using differentiated rendering;

2. Literature Review

In recent years, there has been considerable interest in developing computer vision techniques used to analyze the spatial arrangement of the camera in images in order to create three-dimensional models. A growing number of studies focus on the use of PyTorch3D, a library that provides tools for three-dimensional modeling in computer vision. These tools allow processing and integrating data from photographs into accurate three-dimensional point clouds. While there is a significant amount of literature on computer vision techniques, the use of PyTorch3D to accurately capture camera position in photographs has been less widely investigated. This review aims to analyze the existing research on this topic and identify the potential of PyTorch3D in creating accurate three-dimensional models.

In the field of computer vision, especially concerning Neural Radiance Fields (NeRF), the authors of [1] proposed Depth-supervised Neural Radiance Fields (DS-NeRF) to solve the problem of obtaining accurate geometries with a limited number of input images. By implementing a depth-guided loss function that utilizes data extracted from the motion structure, DS-NeRF not only improves image quality but also accelerates learning [2].

Advancing developments in 3D technology, the authors of [3] developed PyTorch3D, a toolkit designed to process complex 3D data and promote efficient differentiable graphical operations. This toolkit supports applications ranging from autonomous driving to virtual reality, emphasizing the ongoing evolution in deep 3D learning and the potential for improvement by NeRF-enhanced techniques.

Considering cultural heritage preservation, where traditional methods often face obstacles due to the expensive equipment required for high-quality 3D scanning, the authors of [1] also proposed a cost-effective solution using PyTorch3D aimed at art museum restorers. This approach enables the detection of fine surface details of artworks using affordable devices such as tablets, demonstrating PyTorch3D's improved capabilities in the precise camera positioning required for detailed visualization. This integration not only increases the accuracy and

availability of high-quality 3D modeling technologies, but also highlights PyTorch3D as a potentially key tool for 3D reconstruction in contexts requiring accuracy with limited resources.

By reviewing these advances and taking into account additional research, such as also the results of the authors of [4], who study various 3D reconstruction methods, including NeRF, a complex narrative emerges. In particular, the authors of the paper provide evidence that NeRF outperforms traditional photogrammetry methods in processing non-textured objects or reflective surfaces. This discovery suggests that future applications of PyTorch3D may benefit from incorporating NeRF methodologies to improve the quality of reconstructions.

Based on what was described in studies [1] and [2], the authors of [5] presented a modified NeRF-- model that eliminates the need for known camera parameters in NeRF settings. This model emphasizes simplicity in rendering 3D scenes and extends accessibility. Their method, validated with the Blender Forward-Facing Dataset, demonstrates high-quality synthesis and dynamic optimization of camera parameters during training.

Current methodologies used to create new views of scenes from a single image have seen continuous challenges, especially in terms of achieving realism without extensive 3D data. Confronting these obstacles, the authors of [6] developed a novel model that is capable of synthesizing new scene perspectives from just a single input image. The importance of this model in the field of visualization and 3D reconstruction is highlighted by the fact that it is trained directly on real images and functions independently of pre-defined 3D data. At the core of its functionality is a differentiable point cloud visualizer that efficiently converts latent 3D point clouds to the desired viewpoint. It also includes a refinement network that decodes projected features to eliminate gaps and create more realistic images.

The authors of [6] presented an innovative approach using progressive rendering to efficiently handle point clouds, which gives greater flexibility and scalability compared to traditional photogrammetry or the technique discussed in [4]. This flexibility facilitates the visualization of high-resolution scenes, opening new possibilities in animation and interactive augmented and virtual reality applications. The Synsin model described in their study is related to advances in neural radiation field (NeRF) technology, as noted by the authors [5], and reduces the dependence on pre-known camera parameters. In addition, the method of [6] finds echoes in the PyTorch3D initiatives mentioned in [3] and [1], which aim to simplify 3D reconstruction and visualization with minimal initial data.

The study [6] marks a significant step forward in the development of 3D rendering and reconstruction algorithms, reflecting trends corresponding to the application of NeRF and PyTorch3D in fields ranging from museum exhibitions to industrial chemistry. The integration of Synsin methods mentioned in previous articles can open up new opportunities to automate visual content, increase interactivity, and improve the accuracy of 3D models. This ongoing technological development opens new horizons for understanding and reproducing 3D space from a minimal amount of input data, while achieving maximum realism in the scenes created.

Despite these advances in 3D modeling with limited data, a wide range of questions arise as to whether techniques such as NeRF and Synsin, discussed in [5] and [6], can consistently provide accurate and realistic models under different conditions. These techniques greatly enhance the applications of 3D technology, but the consistency of their results under different conditions remains controversial. Moreover, techniques such as PyTorch3D, as pointed out by [3] and [1], push the boundaries of fast and cost-effective 3D scanning and material analysis. Nevertheless, their performance in different operational environments is still surrounded by uncertainty. Studies [4] and [7] suggest an urgent need for further research to develop new methodologies that improve the accuracy and realism of 3D reconstructions. Although steps have been taken, the problem remains insufficiently addressed. Furthermore, these innovative approaches not only facilitate scene reconstruction but also support real-time manipulation and animation of objects in 3D space, which presents ongoing research challenges regarding the feasibility and reliability of these methods in practical scenarios.

In conclusion, despite significant advances in computer vision and 3D reconstruction, many questions regarding the feasibility, reliability, and application of these techniques in various settings remain unresolved. This emphasizes the continued need for further innovation and critical evaluation of these techniques in various fields.

3. Data Processing Methods

As a result of the literature review, the following data processing tools were identified:

3.1 PyTorch3D

PyTorch3D is a 3D data library developed by Facebook AI Research (FAIR). PyTorch3D is a popular choice for several reasons:

- PyTorch3D is tightly integrated with PyTorch, allowing you to take full advantage of this popular machine learning framework. This is especially useful for developers who are already familiar with PyTorch and want to add 3D data processing to their projects.
- PyTorch3D supports differentiable rendering, which allows optimization techniques such as gradient descent to be used for 3D-related tasks [8].

Differential rendering is an important tool for tasks involving the reconstruction of 3D models from 2D images, as well as other operations that may require gradient descent or other optimization techniques. When we generate an image, it can be thought of as a process of mapping 3D models into a 2D plane. For example, if we have two 3D spheres, their position in space determines how they will look in the 2D image. Different configurations of the spheres in 3D will result in different 2D pictures.

Many 3D computer vision tasks require the inverse task: reconstructing 3D structures from 2D images. For example, if you have a 2D image of a scene, you need to determine the 3D structure of the object in order to obtain such an image. The solution to this problem can be thought of as an optimization problem, where the variables are parameters that describe 3D objects (e.g., the location of the center of a sphere). We are looking for values of these parameters such that the generated image is as close as possible to the given 2D image. [1]

To do this, we define a cost function that measures how similar the generated image is to the real image. In this case, for example, we can use the RMS error per pixel. Next, we need to compute the gradient of this cost function with respect to the 3D object parameters. Knowing these gradients allows us to iteratively change the parameters in a direction that reduces the cost, which ultimately leads to better image matching. [8] To do this, the function that maps the parameters into a cost function must be differentiable. This means that the entire rendering process must be differentiable. This allows optimization techniques such as gradient descent to be used to efficiently find parameters that minimize the cost function and provide the best match between the 3D model and the 2D image. [8]

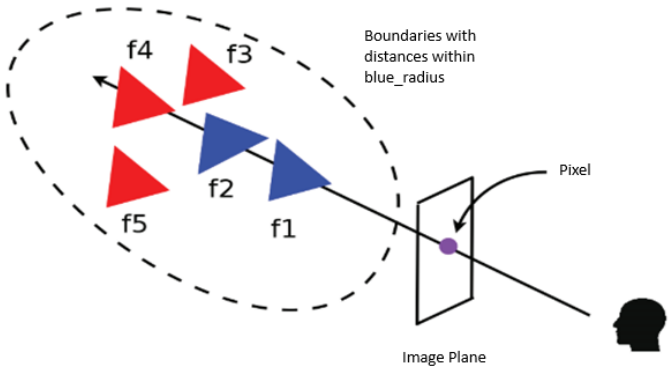

Figure 1. Differentiated rendering by weighted averaging of all matching faces of a polygonal mesh

Traditional screening algorithms face problems when calculating color gradients. The main reason is that the verification process is a separate process. In it, a ray is created for each pixel in the image, which passes through the pixel and intersects with different polygons of the 3D scene. Since this algorithm only selects the grid region closest to the camera, the process is essentially a step function and therefore not differentiable.

PyTorch3D solves this problem using the principles described in the Soft Rasterizer approach [9]. The main idea is to make the rasterization process "soft", allowing to take into account several potentially suitable faces of the polygonal mesh when determining the color of each pixel.

Instead of selecting one nearest edge, all edges whose distance from the ray is less than some threshold are considered. In PyTorch3D this threshold is set through the blur_radius attribute in RasterizationSettings.

For each selected edge, a probability is calculated representing the likelihood of that edge intersecting the ray. The formula includes a hyperparameter that controls the "blur". In PyTorch3D, this parameter can be customized through BlendParams.sigma. [8] Next, the renderer must calculate the probability associated with each face of the polygonal mesh as shown below (1). Here dist represents the distance between the facet and the ray, and σ (sigma) is a hyperparameter. Simply put, the probability for each edge denotes the probability that this polygonal mesh edge covers a particular pixel of the image. In this case, the distance can be negative if the ray intersects a polygonal grid edge.

$$D_{j=sigmoid\ (\frac{-dist_j}{\sigma})} \tag{1}$$

weight, w_b, – is the low weight of the background color. Here, the parameter γ (gamma) is a hyperparameter.

$$W_j = D_j \exp\left(\frac{z_j}{\gamma}\right); W_b = \exp(\frac{\in}{\gamma}) \tag{2}$$

Thus, the final color of a pixel can be determined using the following equation below:

$$I = \frac{(\sum_j w_j c_j) + w_b c_b}{(\sum_j w_j) + w_b} \tag{3}$$

When implementing differentiable rendering in the PyTorch3D library, an additional α (alpha) value is calculated for each image pixel. This value represents the probability that the image pixel is in the foreground and that the ray intersects at least one edge of the polygonal mesh. [8]

In the soft rasterizer, the value of α is calculated from the corresponding facet probabilities as shown below.

$$a = 1 - \prod_J (1 - Dj) \tag{4}$$

The color of a pixel is determined by weighted averages of the shading values of all selected faces, with the weights depending on the calculated probabilities.

Differentiated rendering greatly enhances computer vision and machine learning capabilities by enabling the efficient use of optimization techniques such as gradient descent to solve complex problems, including the reconstruction of 3D structures from 2D images. PyTorch3D implements differentiable rendering using a soft rasterization approach that makes the process smooth and mathematically differentiable. [8]

Using PyTorch3D's differentiable renderer, we can solve the problem of object pose estimation. The task will be to estimate the pose of an object from one single image obtained from observation. In addition, we will assume that we have a three-dimensional mesh model of the object.

Additionally, PyTorch3D offers many modules for handling various aspects of 3D data, including rendering, mesh deformation, 3D model reconstruction, etc. This allows developers to easily assemble their own processing pipelines. And it also supports working with meshes (polygonal meshes), point clouds and voxels, making it versatile for various applications.

- Pixels in 2D are used to render flat images or sprites in 3D scenes. This is useful for creating the illusion of depth at minimal computational cost.
- Polygons (Triangles and Squares) are used to model the surfaces of 3D objects. This approach allows you to create complex shapes with high detail with a relatively small amount of data.
- Voxels: are used to model volumetric data, including the internal structure of objects. Voxel models consist of many small cubes that fill 3D space.

Voxels, short for “volume element”, are the analog of pixels in 3D space. While pixels represent 2D elements of an image, voxels are volumetric cubic elements that fill 3D space. Voxel representations are often used for visualization of volumetric data, for example, in medical visualization or in game applications. [8]

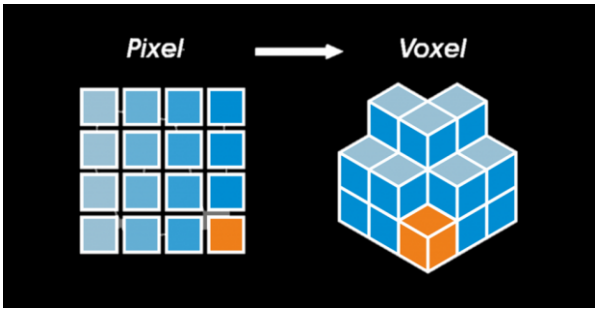

Figure 2. Voxel representation

Voxels can represent not only the surface of an object but also its internal structure, and voxels are easy to discretize and process, making them convenient for volume visualization algorithms. But voxel representations can take up a lot of memory, especially for high resolutions and a huge number of voxels are required to store details, making such representations inefficient for highly detailed models. [8]

Additionally, OBJ and MTL files are used in the study. The OBJ file and the associated MTL file are often used to store 3D models:

- The OBJ file describes the geometry of the object. It contains a list of vertices (points) and planes (faces) that form a polygonal mesh. It is one of the most common formats for 3D models because of its simplicity and support by various 3D modeling programs.
- MTL file contains information about materials such as color, textures, reflective properties, etc. An MTL file is usually referenced in an OBJ file and allows you to assign materials to polygons.

3.2 Neural Radiance Fields

NeRF is a method that uses neural networks to represent and render 3D objects based on a series of 2D images. The basic idea is to represent a continuous scene using a density and color function in a 5-dimensional space, which interprets the visible scene in three spatial dimensions and two angular dimensions. [4]

The concept of Neural Radiance Fields (NeRF) is an approach to modeling 3D scenes using neural networks. The main essence of the task is to synthesize new scene angles based on a limited number of available 2D images. This is a challenging task as many factors such as object artifacts, light sources, reflections, opacity of materials, texture of surfaces and occlusions by other objects need to be taken into account. These aspects can significantly affect the final image when changing the perspective. [5]

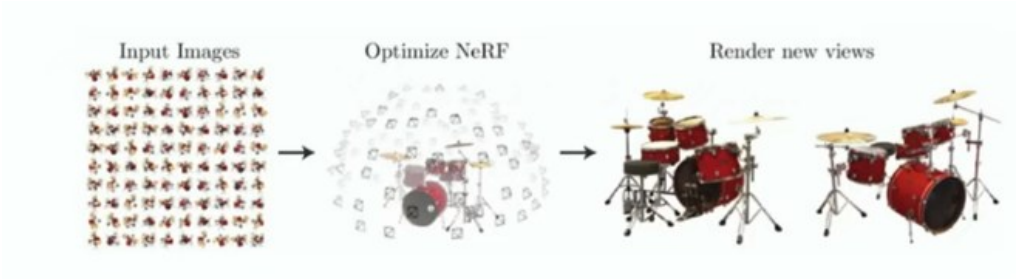

Figure 3. Algorithm of NeRF operation

The successful application of NeRF (Neural Radiance Fields) requires specific input data that includes scene images, camera parameters (internal and external), and possibly masks or other auxiliary data. [8] Below is how these data structures are typically organized.

Images - These can be color or black and white images of the scene from different angles. They are usually in PNG or JPEG format.

- Format: .png or .jpg
- Content: A set of scene images, each with its own unique name.

Camera intrinsics - these parameters describe the internal calibration of the camera, such as focal length, distortion coefficients and optical center coordinates.

- File format: text file (e.g., intrinsics.txt)
- Contents: may include focal length (fx, fy), main point coordinates (cx, cy) and distortion parameters, if any.

The NeRF method uses neural networks to deeply and accurately model a scene using a non-conventional approach. This method was proposed by a research team from the University of California, Berkeley, Google Research, and the University of California, San Diego. [8] With this unique use of neural networks and the high accuracy of the learned models, the NeRF method has unleashed a number of new inventions in image synthesis, depth sensing and 3D reconstruction. Thus, mastering this concept becomes critical to further explore the mentioned topics.

The “physics” is reduced to a formula for volume rendering:

$$\mathrm{C(r)} = \int_{t_n}^{t_f} \mathrm{T(t)}\sigma\big(\mathrm{r(t)}\big)\mathrm{c(r(t),d)dt}, \text{ where } \mathrm{T(t)} = \exp\left(-\int_{t_n}^{t} \sigma\big(\mathrm{r(s)}\big)\mathrm{ds}\right) \quad (5)$$

here (C) - the resulting color of the beam, (T) - transmittance, (c) - local color, (σ) - it's density, (r) - the coordinate of the ray, $\mathrm{a}(d)$ - direction.

NERF encodes a continuous volumetric feature that gives good quality and requires little storage space.

$$5D(x, y, z, theta, phi) \rightarrow 3D(r, g, b) \quad (6)$$

The luminance field underlying NeRF describes the distribution of light in 3D space and allows for detailed modeling of light interactions with objects in the scene. Neural networks serve to capture and represent this complex information, making NeRF capable of dealing with incomplete and noisy information, producing an accurate representation of the scene. [8] This concept offers a powerful tool for solving the problems of synthesizing new perspectives and modeling complex 3D scenes, making it important for all professionals working in computer vision.

The brightness field of radiation describes the distribution of light energy in space and time. Radiation brightness is understood as the intensity of light at a given point of space when observed in a certain direction. This parameter is measured in units of light intensity per unit area per unit solid angle. In the context of computer vision and graphics, radiant luminance is most often represented in the RGB system, where the three components (red, green, and blue) describe color information. [8]

It is important to realize that the brightness of the radiation is determined by a number of factors. First, these factors include the light sources that illuminate a point in space. The position, power, and color of the light sources affect how the scene is illuminated and perceived. Second is the presence of surfaces or volumes that can change direction, reflect or absorb

light. These objects can create shadows, glare, or diffuse reflections, contributing to the overall luminance pattern. Third, surface texture and material also play an important role: smooth or shiny surfaces will reflect light differently than rough or matte surfaces.

The radiance field is a key concept underlying 3D scene modeling and rendering techniques such as NeRF. It allows us to model how light propagates in a complex scene, taking into account the interactions of light with objects and materials. Storing and utilizing this information allows us to create photorealistic images and accurate reconstructions of 3D objects, which is the goal of such methods.

Representing radiance fields using neural networks such as NeRF (Neural Radiance Fields) profoundly changes the approach to 3D scene creation and rendering tasks. Unlike traditional methods, NeRF uses neural networks not to classify or generate images directly, but to represent the scene in the form of a volume function. [8] The following is a brief description of this approach.

NeRF uses a multilayer perseptron (MLP), which is essentially a fully-connected neural network, to model the scene. This network is not a convolutional neural network (CNN), which makes it noticeably different from most current models in computer vision.

The input data of the network includes five coordinates:

- Three spatial coordinates (x, y, z) to define a point in 3D space.
- Two viewing angles (θ, Ø) that can be converted to a unit direction vector d in a Cartesian coordinate system. This helps to specify the direction under which the point is observed.

For each point and direction, the network predicts:

- Volume density σ at the point in question (x, y, z), which determines how much light can be absorbed or scattered by this volume;
- Color (r, g, b), which determines the color of the emitted light at that point at a given viewing direction (θ, Ø). This color serves as an indirect indicator of the brightness of the emitted light.

Each input point is a 5-dimensional vector. It was found that training the model directly on these inputs does a poor job of representing high frequency variations in color and geometry. This is because neural networks tend to learn low frequency features better. An effective solution to this problem is to transform the input space into a higher dimensional space and use it for training. This transformation is a set of sinusoidal functions with fixed but unique frequencies.

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \dots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)) \quad (7)$$

The NeRF model is trained on multiple images of the same scene taken from different angles. This means that each individual model is optimized for only one scene, and yet it can generalize to new viewpoints within the scene.

The model uses a volume rendering technique to produce the final image. This involves integrating the predicted densities and colors along rays coming from the viewpoint to create a two-dimensional image of the scene.

For each pixel of the image, the released ray from the camera crosses the scene, and points are selected along the length of this ray. Each point is assigned spatial coordinates and the corresponding camera direction vector.

At each selected point (x, y, z) and viewing angle (θ, Ø), the neural network predicts the density σ and color (r, g, b).

Using volumetric rendering, density and color values are aggregated along each ray in a specific way to produce the final image. This involves weighting the predicted colors based on the densities and further summing these weighted colors to obtain the image pixel. [8]

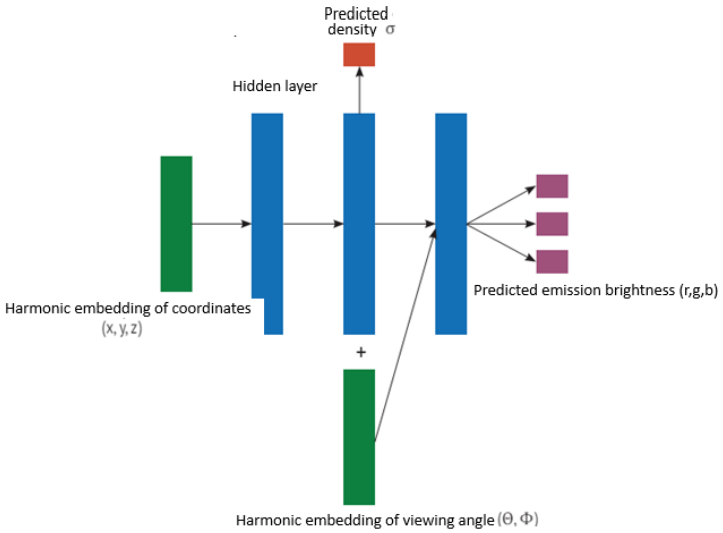

Figure 4. Simplified architecture of the NERF model

3.3 Mesh R-CNN и Mask R-CNN

This section describes the Mesh R-CNN model, which combines two important tasks into one end-to-end model: image segmentation and 3D structure prediction. The Mesh R-CNN model combines the well-known Mask R-CNN, designed for instance segmentation based on object detection, with a new model that predicts 3D structures. [12]

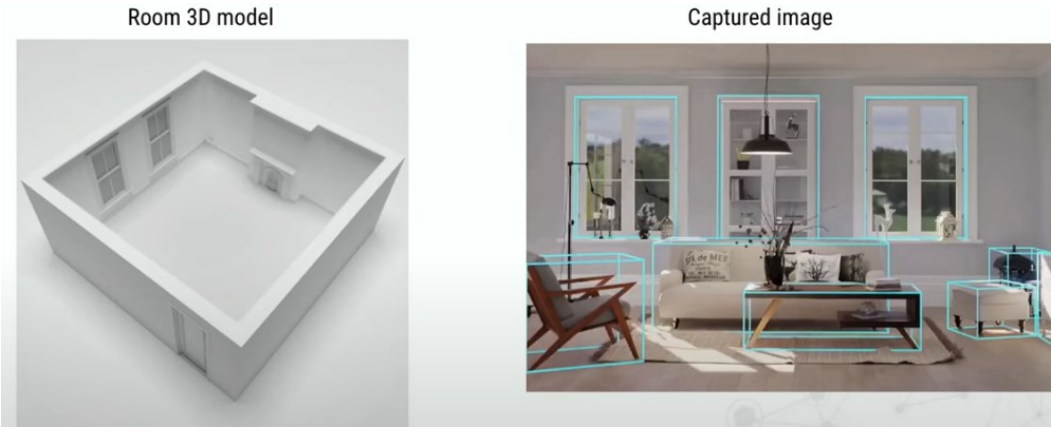

Figure 5. Mesh R-CNN processing result in terms of object recognition

Mask R-CNN is based on an algorithm that provides high precision in benchmarks, and acts within the R-CNN family as a two-stage model for object detection. However, Mesh R-CNN extends this functionality by offering not only 2D segmentation, but also the ability to generate 3D polygonal meshes for detected objects. Thus, Mesh R-CNN aims to mimic human perception, which perceives the world in 3D space, and takes it a step further by rendering objects in 3D. [8]

The Mask R-CNN model processes an RGB image as input and produces bounding boxes, category labels and instance segmentation masks as output. The image is first passed through a frame network, typically based on ResNet, such as ResNet-50-FPN. This network generates a feature map, which is then passed to a plot suggestion network (RPN). The RPN produces sentences that are processed by the feature classification and mask prediction branches, resulting in output classes and masks. [8]

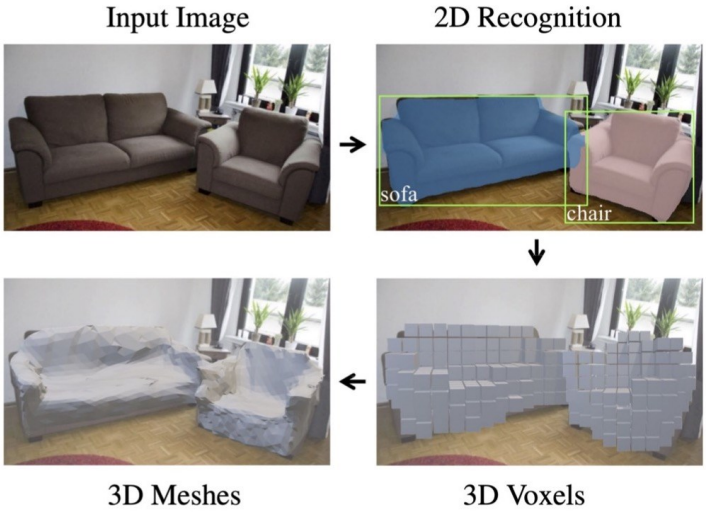

Figure 6. Result of the Mesh R-CNN model in terms of voxel creation.

This Mask R-CNN architecture is retained in the Mesh R-CNN model, but with the addition of a polygonal mesh predictor. This new module includes two branches: the voxel branch and the polygonal mesh refinement branch.

The voxel branch processes the proposed and aligned features, producing rough voxel predictions as output. These coarse predictions are then passed to the input of the polygonal mesh refinement branch, which generates the final polygonal mesh. The losses of the voxel branch and the polygonal mesh refinement branch are added to the frame and mask losses, and the entire model is trained end-to-end from start to finish. [12] Mesh R-CNN not only retains the segmentation and classification capabilities of Mask R-CNN for 2D objects, but also adds the ability to predict their 3D structure, making the model more comprehensive and consistent with the real world perception in 3D.

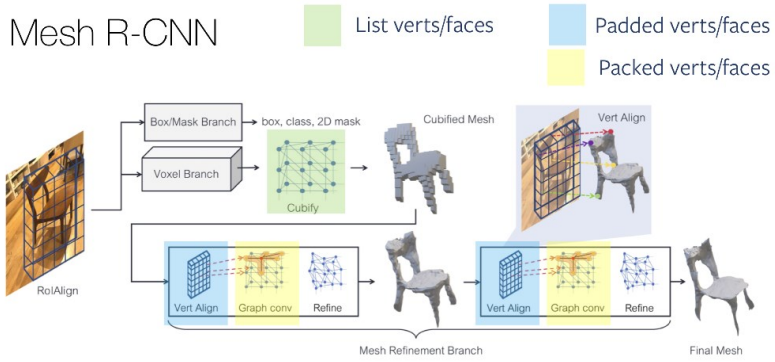

Figure 7. Architecture of the Mesh R-CNN model

The Polygonal Mesh Prediction module is designed to detect the 3D structure of an object. It is a logical development of the RoI align module and is responsible for predicting and outputting the final polygonal mesh.

Since we need to obtain three-dimensional polygonal meshes from real images, it is not possible to use fixed mesh templates with specific topologies. Therefore, the polygonal mesh predictor consists of two branches. The joint use of the voxel branch and the polygonal mesh refinement branch helps to cope with the problem of fixed topologies.

Voxel loss is a binary cross entropy that minimizes the predicted voxel occupancy probabilities by comparing them to the true occupancy values.

The polygonal mesh refinement branch includes a sequence of three operations: vertex alignment, graph convolution, and vertex refinement. Vertex alignment is similar to ROI alignment; it finds features for each vertex of the polygonal mesh aligned to the image.

For this purpose, the following method of selecting points from the polygonal mesh is used: given vertices and faces, points are selected uniformly from the probability distribution of the polygonal mesh surface. The probability of each face is proportional to its area.

Using these selection methods, a point cloud Q is formed from the valid data and a point cloud P is formed from the prediction. Next, DP,O, which is the set of pairs (p, q), where q is the nearest neighbor of point p in Q, is computed.

Then, the facet distance between P and Q is computed.

$$L_{\text{cham}}(P,Q) = |P|^{-1} \sum_{(p,q)\in\Delta_{P,Q}} |p-q|^2 + |Q|^{-1} \sum_{(q,p)\in\Delta_{Q,P}} |p-q|^2 \quad (8)$$

Then the absolute normal distance is calculated:

$$L_{\text{norm}}(P,Q) = -|P|^{-1} \sum_{(p,q)\in\Delta_{P,Q}} |u_p \cdot u_q|^2 - |Q|^{-1} \sum_{(q,p)\in\Delta_{Q,P}} |u_p \cdot u_q|^2 \quad (9)$$

3.4 Proprietary data preprocessing solutions

Based on the literature review, it was found that almost every study uses a ready-made data set. Within the framework of this study, a hardware and software package that generates an image of an object from different angles in the form of photographic images was used for data preparation. As a result, there is a need to obtain metadata for processing independently, for which it was decided to use the software listed below:

3DF Zephyr is an advanced photogrammetry software that allows users to automatically convert photo series into 3D models. This tool uses powerful algorithms to process images and create accurate and detailed 3D reconstructions. [10] The program has a wide range of features, including the ability to edit models, measure distances and areas, and export to various formats. 3DF Zephyr is used in many fields such as architecture, engineering, cultural heritage, and the gaming industry due to its flexibility and accuracy in recreating physical objects digitally.

COLMAP is a powerful open-source photogrammetry tool that provides users with comprehensive capabilities for creating three-dimensional models from photographs. The program combines automated Structure from Motion (SfM) and dense stereo matching algorithms to accurately recreate 3D scenes and objects. [11] COLMAP is ideal for researchers and 3D modeling enthusiasts, offering features such as automatic photo alignment, geometry optimization, and model texturing. The tool is widely used in fields that require high fidelity recreations of reality, including archaeology, architecture, and virtual reality content creation

4. Research methodology

In the course of the research it was decided to collect and prepare photos of the object instead of using ready-made datasets, to convert 2D photos of the object into a 3D model using proprietary solutions Colmap and 3D Zephyr, as well as to perform work processing using methods previously listed in the article, in particular PyTorch3D and evaluate the results obtained.

4.1 Data preparation

Instead of pre-prepared data, which are usually used in NERF, an integrated hardware-software complex, described in detail by the authors in [13], which is specialized for receiving, transmitting and storing a series of photographic images, was used for data creation and preparation. This complex provides the possibility of comprehensive photography of the object, which allows to fix it from all sides and form a complete dataset. The obtained images can then be downloaded to a computer, where they are used to create an accurate three-

dimensional model of the object, providing a high degree of detail and realistic visualization. Figure 8 shows the installation of the hardware and software system. It is based on:

- A rig capable of 360 degrees of rotation, with a movable carriage that can move 90 degrees;
- Stepper motors and associated drivers, with 450 mm diameter toothed perforated rotor parts, providing movement on the toothed rails of the rig;
- Board and camera, mounted on a movable carriage and protected by an enclosure (Figure. 9);
- The camera was a Raspberry Pi Camera v2.1 This camera has a Sony IMX219 Exmor sensor, a resolution of 5 megapixels, a maximum photo resolution of 2592×1944 pixels, and has a fixed focal length [14];
- Chromakey, which significantly improves the quality and accuracy of the created model, reducing processing time and ensuring the creation of high-quality backgrounds for models that can be used later.

Figure 8. Setup for capturing photo images

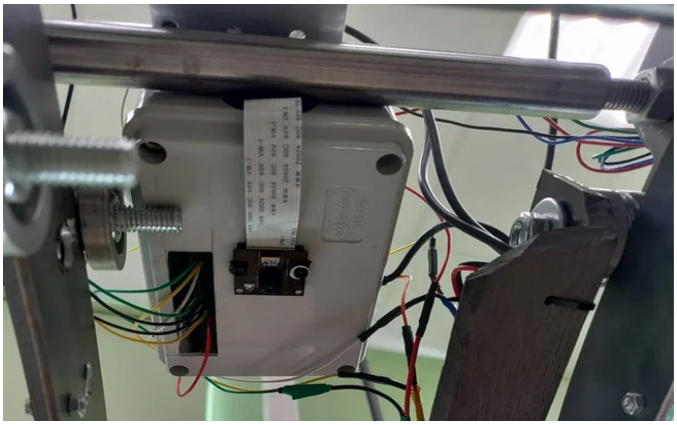

Figure 9. Movable carriage with installed camera and board in the housing

Figure 10 shows the user interface for working with this hardware-software complex, within which it is possible to specify the number of stops (positions for taking photos), and on 11 the ready dataset (collection of photos of one particular object). The necessary and sufficient number of photos was chosen empirically of each object in the amount of 50.

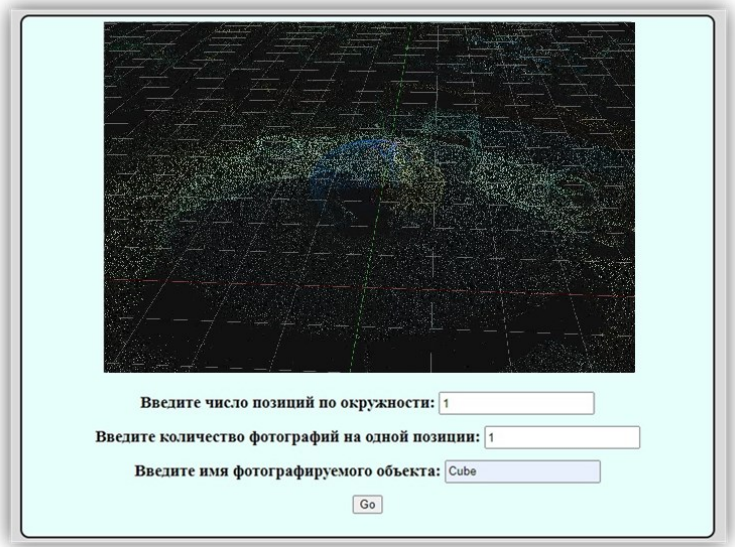

Figure 10. User interface of the hardware-software complex

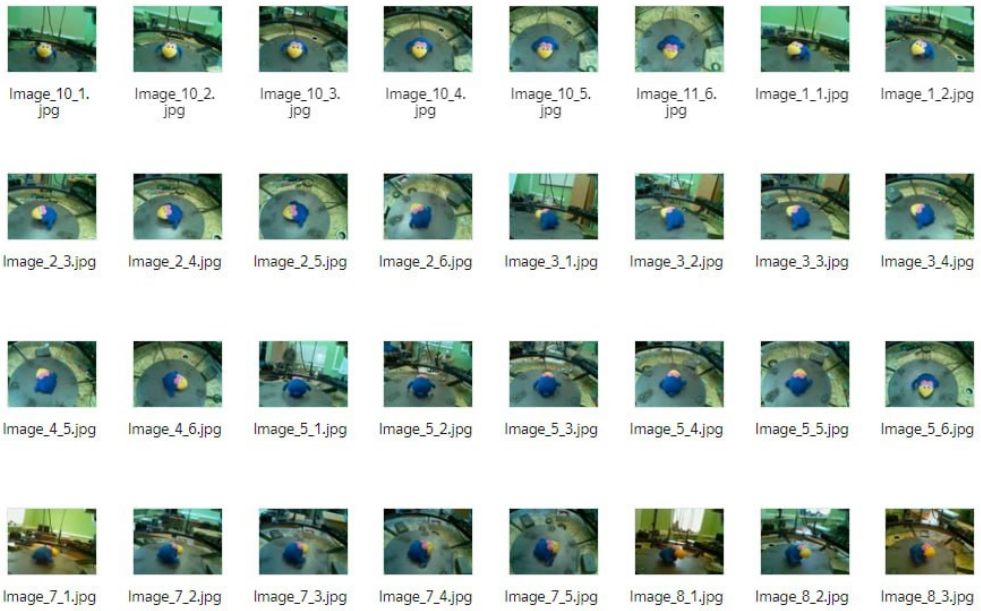

Figure 11. Example dataset

4.2 Convert to 3D model

The next step is to convert the dataset of photos into a 3D model of .obj format. For this purpose, 3DF Zephyr and COLMAP programs were used.

As part of the study, data preparation was performed for several objects, among which were the simplest geometric shapes and more complex ones, among which were objects with complex geometry and mirrored surface. The result of visualization of each of the listed objects is provided in Figure 12.

Figure 12. Example of processed 3D objects in 3DF Zephyr software

An example of the target 3D object that was selected to continue the study is shown in Figure 13. An example of a processed 3D object is presented in Figure 14.

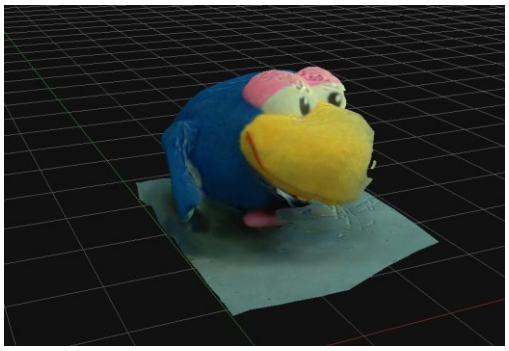

Figure 13. Example of a processed 3D object in 3DF Zephyr software

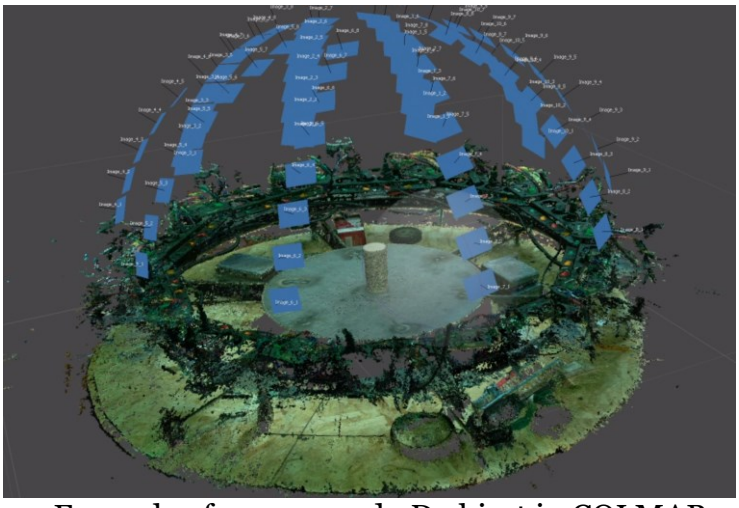

Figure 14: Example of a processed 3D object in COLMAP software

5. Results of the study

At the previous stages of the study, 50 images of objects from different angles were obtained using the hardware-software complex. One of the objects was empirically selected as a test object (Figure 15).

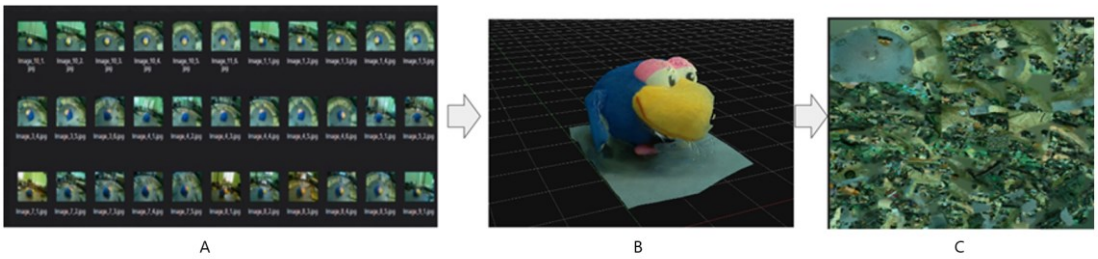

Figure. 15. Prepared dataset. A. - Prepared dataset, B. - 3D model created in 3D Zephyr from the dataset, C. - Object texture in .mtl format.

Processing 3D files involves several steps: creating a model, texturing, and then processing that model with different programs to get the final result. Various file formats are used in this process, each with its own purpose and structure:

- The .obj format is widely used to store information about 3D models. It is a text file that contains data about vertices, texture coordinates, normals and indices.
- The .mtl format is used to describe the material for an .obj object. It defines properties such as material color, texture, reflectance, etc.
- The .npz format is a container for storing multiple NumPy data arrays, which is a standard for scientific and numerical computing in Python. It stores large amounts of numerical data sets such as point clouds, which is convenient for data processing and analysis in scientific and technical applications.

With 50 photographs of the same object from different angles, we used 3D Zephyr and COLMAP to extract metadata from these photographs, including obj, mtl, and npz file formats. (Figure 16).

Figure 16. Translating dataset to .npz-format

After that, we proceeded to solve the object pose estimation problem. In the process of solving the object pose estimation problem, differentiable rendering was applied, whereby the 3D model was compared with a single image from a single perspective. The single image from a single perspective was obtained from the original dataset data. The comparison was performed using mean square error. In this section, we have demonstrated a concrete example of

using differentiable rendering for 3D computer vision tasks. Our goal is to estimate the pose of an object from a single image obtained from observation. Assuming that we have a three-dimensional model of the object, we can perform a comparison between an RGB image of the object and a silhouette image - Figure 17. The task is to estimate the orientation and pose of the object at the time these images were captured.

Figure 17. 3D object transformation. A. - Point clouds of the object (3D model), B. - Single images of the object from different angles.

Since it is difficult to rotate and move the polygonal meshes, we instead fixed their orientation and location by deciding to optimize the camera orientations and locations. Under the assumption that the camera is always facing the polygonal meshes, the task can be simplified to optimizing the camera location.

Thus, we formulated an optimization problem where the variables are camera coordinates. Using differentiable rendering, synthetic RGB images and silhouette images of the polygonal mesh of the object can be obtained. These synthetic images are then compared to the observed images, allowing the loss functions between them to be calculated. RMS errors are used as the loss function. Since the process is differentiable, the gradients of the loss functions with respect to the optimization variables can be computed. The gradient descent algorithm can then be used to find the optimal camera positions at which the synthetic images maximally match the observed images. As a result, the camera position was determined - Figure 18.

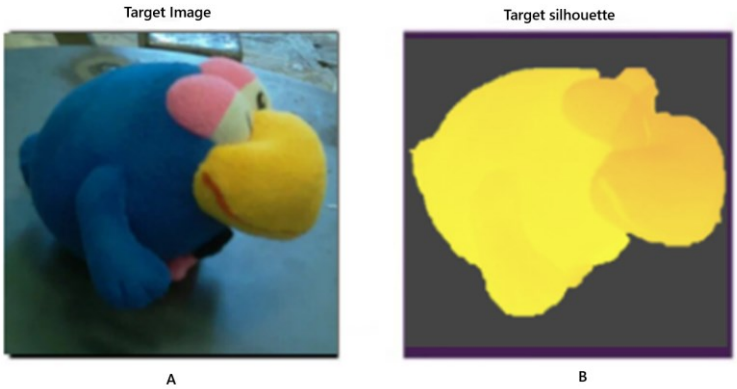

Figure 18: Comparison of 3D model. A - Single image from one perspective. B. - Target image of the object orientation.

Figure 19 shows the RMS error:

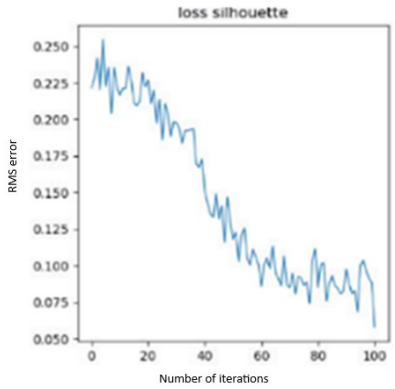

Figure 19. Root mean square error

In further stages of the research, it is planned to use NERF to solve the inverse recognition problem - creation of 3D models and an improved implementation of Mesh R-CNN to form a polygonal mesh to identify objects in photographs with detailed prediction of geometry and topology.

In the task of reconstructing a 3D object and the entire scene from multiple images taken from different angles, there are several key steps and techniques that can make it much easier.

First, we start with the data collection phase. The rig takes images of the object from different angles, and the first step here is to properly segment the object in these images. For this, we can use Mask R-CNN, which is known for its accuracy in the task of image segmentation. This will allow us to automatically find and select the object, which is critical for the subsequent reconstruction steps. After all, the segmentation accuracy determines how "clean" the data passed to the next stage of the algorithm will be. Next, we move on to the reconstruction of the object itself. There are several methods, but MESH R-CNN looks like a particularly promising choice when it comes to 3D reconstruction from 2D images. This technique allows us to obtain detailed mesh models that can be used for further analysis or visualization. However, it is worth remembering that such methods often require significant computational power and can be quite difficult to implement. But if the target criterion is high accuracy of the model, it is quite justified. After successful reconstruction of the object, we move on to the important aspect - reconstruction of the whole scene. NeRF (Neural Radiance Fields) can help us a lot here. This technology is remarkable because it allows us to reproduce not only shape, but also lighting and texture, which makes it ideal for creating photorealistic scene models. NeRF models a scene based on multiple images, creating a three-dimensional representation that can be easily integrated into virtual and augmented realities. But again, we pay for the high quality by the fact that NeRF requires significant resources to train the model. Now that we have the object and scene models, the question of integrating and visualizing them arises. This is where PyTorch3D comes to the rescue. It supports a variety of formats and visualization capabilities, which makes it easy to combine the results and get the final product. The ability to integrate is also important for the final stage of verification of the results and their subsequent use, whether in game engines or in engineering CAD programs.

As a result, for the task of reconstructing a 3D object and scene from images taken from different angles, several technologies need to be integrated to achieve the best results. Mask R-CNN will help in initial segmentation, MESH R-CNN will provide accurate reconstruction of the object, NeRF will create a detailed 3D scene, and PyTorch3D will allow for model integration and visualization. A cumulative comparative analysis of the data processing methods is listed in Table 1.

Table 1. Comparative analysis of data processing methods.

Characteristic	PyTorch3D	NeRF (Neural Radiance Fields)	MESH R-CNN	Mask R-CNN
Data Type	3D data	2D images for 3D scenes	2D images for creating 3D models	2D images
Goal in This Task	Visualization and processing of 3D models	Photorealistic reconstruction of 3D scenes	Reconstruction of 3D models from images	Object detection and segmentation
Main Task	Reconstruction of 3D objects	Generation of photorealistic 3D scenes	Segmentation and creation of 3D object models	Object extraction for further processing
Advantages	Model visualization, fast processing	High detail and accuracy	Accurate reconstruction of shapes and contours	Fast object extraction, data preparation
Limitations	Requires PyTorch knowledge, complex setup	High computational load	Dependency on the quality of mesh data	Limited to 2D segmentation
Complexity	Medium	High	High	Medium
Application Area	Processing and visualization of 3D models	Virtual/Augmented Reality, high-level reconstructions	Computer vision, medicine	Preliminary image segmentation
Optimal Use	Post-processing and data visualization	Tasks requiring photorealistic visualization	Precise reconstruction focusing on shapes	Data preparation for subsequent 3D reconstruction

To summarize, the following results were obtained during the study:

- a literature review of 3D image processing using PyTorch3D, NERF and MESH R-CNN tools was performed;
- as a result of modification of the hardware-software complex for automated measurements of relatively small objects from the outside and premises from the inside using both photofixation and lidar scanning, chromakey was added to improve image quality and preparation of photos of different types of objects for further analysis was performed;
- photographs of the object from different angles were obtained and processed by 3D Zephyr and COLMAP to obtain photo metadata, in particular .obj, .mtl and .npz files;
- the acquired data were processed using the PyTorch3D library to determine the position of the object in the images;
- PyTorch3D, NERF and MESH R-CNN, Mask R-CNN methods as image processing tools were analyzed through literature analysis and review of data processing techniques; and, conclusions were drawn on the use of MESH R-CNN implementation for further research to determine the position of objects in the photographs.

Conclusion and discussion

The introduction of new technologies and approaches, such as the PyTorch3D library, opens up new possibilities for solving the above problems. This research was aimed at exploring the potential of using PyTorch3D to determine camera position and create 3D models of objects based on a single 2D image. As part of the work, a hardware-software complex was

used, including stepper motor control for accurate camera positioning, a shooting control system and a mechanism for data transfer to a remote server for further processing. The main objective of the study was to evaluate the accuracy and efficiency of the proposed method compared to traditional approaches, as well as to identify potential areas of application of the developed solution. The aim of the work was to investigate methods of 2D and 2D image processing using PyTorch3D library.

In the process of work the following tasks were solved: preparation of initial data (photos) of the object using stepper motor control and sequential positioning of the camera, generation of a complex set of initial data at each camera position, data transfer to a remote computer for processing, and generation of .obj model of the object. The next step involved the task of object detection in the 2D image based on the OBJ model using PyTorch3D. This involved building a point cloud to generate a 3D volumetric model, determining the position of the camera in 3D space from a single photo, determining the position of the 3D object in the photo using differentiable rendering, and creating 3D voxels and 3D meshes.

In the course of the research, a literature review was performed in terms of 3D image processing using PyTorch3D, NERF and MESH R-CNN tools. A hardware and software system was modified for automated measurements of relatively small objects from the outside and from the inside, using both photofixation and lidar scanning. Photographs of the object from different angles were obtained and processed using 3D Zephyr and COLMAP to produce photo metadata, specifically .obj, .mtl and .npz files. The resulting data were processed using the PyTorch3D library to determine the position of the object in the imagery. The capabilities and limitations of NERF and MESH R-CNN were identified. Conclusions are drawn to continue the research in terms of applying NERF to solve the inverse recognition problem, which will allow the creation of 3D models of objects. An improved implementation of Mesh R-CNN will be used for polygonal mesh generation and detailed prediction of the geometry and topology of objects in photographs.

1. Cossairt, O., Willomitzer, F., Yeh, C. K., & Walton, M. (2020, November). Low-budget 3D scanning and material estimation using PyTorch3D. 2020 54th Asilomar Conference on Signals, Systems, and Computers (pp. 1316-1317).
2. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
3. Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W. Y., Johnson, J., & Gkioxari, G. (2020). Accelerating 3d deep learning with pytorch3d. arXiv preprint arXiv:2007.08501.
4. Remondino, F., Karami, A., Yan, Z., Mazzacca, G., Rigon, S., & Qin, R. (2023). A critical analysis of NeRF-based 3d reconstruction. Remote Sensing, 15(14), 3585.
5. Wang, Z., Wu, S., Xie, W., Chen, M., & Prisacariu, V. A. (2021). NeRF--: Neural radiance fields without known camera parameters. arXiv preprint arXiv:2102.07064.
6. Wiles, O., Gkioxari, G., Szeliski, R., & Johnson, J. (2020). Synsin: End-to-end view synthesis from a single image. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 7467-7477).
7. Wu, R., Mildenhall, B., Henzler, P., Park, K., Gao, R., Watson, D., ... & Holynski, A. (2023). Reconfusion: 3d reconstruction with diffusion priors. arXiv preprint arXiv:2312.02981.
8. Ma K., Hegde W., Yolan L. M12 Three-dimensional deep learning in Python / translated by A. V. Logunov. - Moscow: DMK Press, 2023. - 226 p.: ill.
9. Liu S. et al. Soft rasterizer: A differentiable renderer for image-based 3d reasoning //Proceedings of the IEEE/CVF international conference on computer vision. – 2019. – C. 7708-7717.
10. Miyake K. Evaluating the Reliability of Three-dimensional Models Constructed Photogrammetry Software 3DF Zephyr by Measuring Joint Angles of Fingers: A Comparison to a

Conventional Goniometer //Journal of Plastic and Reconstructive Surgery. – 2024. – Т. 3. – №. 1. – С. 34-38.

11. Fisher A. et al. COLMAP: A memory-efficient occupancy grid mapping framework //Robotics and Autonomous Systems. – 2021. – Т. 142. – С. 103755.

12. Gkioxari G., Malik J., Johnson J. Mesh r-cnn //Proceedings of the IEEE/CVF international conference on computer vision. – 2019. – С. 9785-9795.

13. Konkov, V.V., Zamchalov, A.B., & Zhabitsky, M.G. (2023). Complex software and hardware for IIoT-based acquisition of photographic images and analysis of accuracy of different algorithms for digital generation of 3D models based on the principle of photogrammetry. International Journal of Open Information Technologies, 11 (8), 32-51.

14. Camera for Raspberry Pi “Model D”. - Text: electronic // Amperk: [electronic resource]. - URL: http://wiki.amperka.ru/products:camera-raspberry-pi-model-d (address date: 15.04.2023).