Ministry of Science and Higher Education of the Russian Federation
NATIONAL RESEARCH
TOMSK STATE UNIVERSITY (TSU)
Institute of Applied Mathematics and Computer Science

ADMISSION TO PROTECTION IN GEC
Head of the OPOP
Dr. tech. sciences, professor
_____ Y.L. Kostyuk
«_____»_____2022 г.

FINAL QUALIFICATION WORK OF THE MASTER
(MASTER'S DISSERTATION)

# DEVELOPING A VIRTUAL RENOVATION APP

in the direction of preparation 02.04.02 Fundamental informatics and information technologies, focus (profile) "Computer science and technology"

Sakhapov Daniil Albertovich

Supervisor
_____ A.M. Babanov
«_____»_____2022 г.

Author
Student of group № 932007
_____D.A. Sakhapov
«_____»_____2022 г.

Tomsk – 2022

# ANNOTATION

During the renovation, many tasks arise: repaint the walls, replace the old sofa with a new one, buy a new wardrobe ... It is not always possible to imagine how the result of a particular decision will look like. To simplify this representation, you can resort to visualizing the result. For this, there are various applications for virtual renovation, but all of them are highly specialized and are not suitable for a comprehensive solution to the problem of renovation. This paper describes the development of an application that can cover a wide range of tasks related to virtual renovation.

The object of research is virtual renovation.

The subject of the study is an application for virtual renovation.

The goal is to design and develop a virtual renovation application.

Tasks:

• identify and draw up requirements;

• study analogues;

• study the theoretical foundations of neural networks;

• design an application, neural networks;

• develop an application and neural networks;

• train neural networks;

• test the application and neural networks.

The result of the work is the analysis of requirements, the design and development of a web application for virtual renovation.

# TABLE OF CONTENTS

# INTRODUCTION

During the renovation of an apartment, questions often arise: “how will the new sofa look in this place?”, “Will this parquet fit into the current design of the room?”; The best answer to these questions is visualization.

But visualizing a sofa, especially parquet or wallpaper, is a difficult task for the human brain. To simplify this task, nowadays there are a large number of programs that allow you to do “virtual renovation”: insert 3D objects into the augmented reality of a room (Figure 1), replace wallpaper, floor, etc. These programs often require sophisticated technology to operate. There are also simpler programs where a person himself can build a model of his room, but only in an impersonal form, due to which the resulting 3D model loses its value, since it is difficult to imagine how this or that object (for example, a sofa) will look in a real apartment (Figure 2).

The purpose of this work is to develop a program that could simplify this visualization process by creating a 3D model of a real room and other “virtual renovation” processes. And in order for the process not to be tedious, it is proposed to answer all questions using only one photograph, which can be easily taken using any camera.

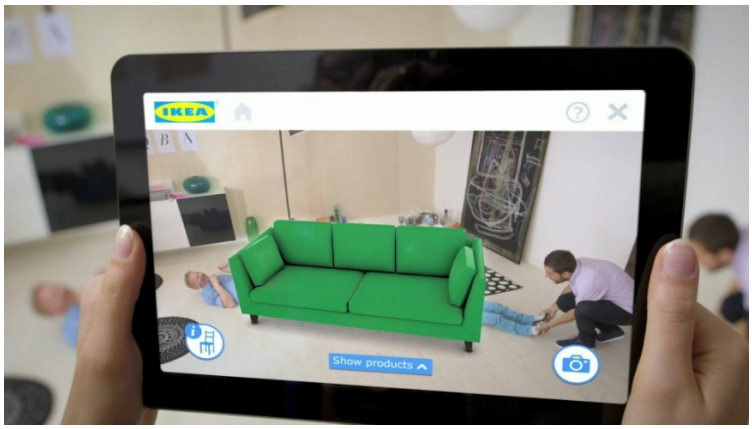



Figure 1 - An example of augmented reality created in one of the existing programs

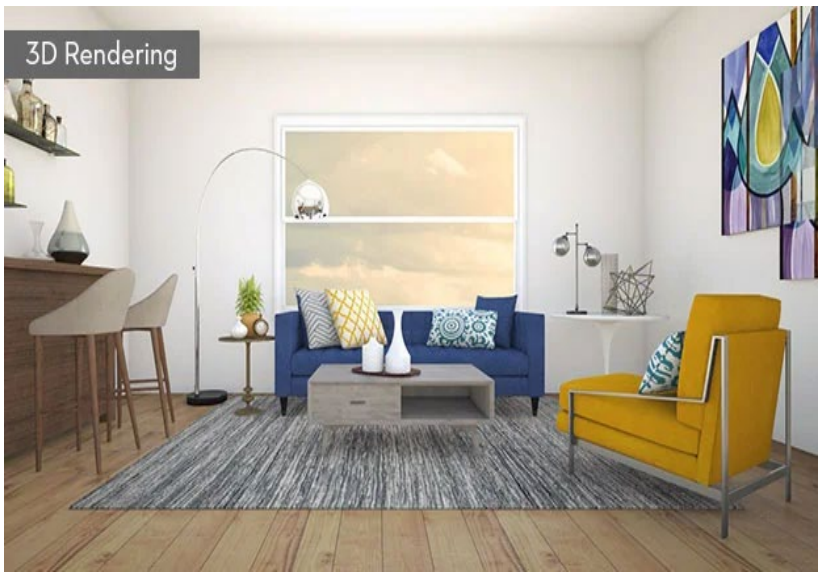



Figure 2 - An example of a 3D model created in one of the existing programs

In addition to the user's desire to see how an object will look in their real room, it may be necessary to paint over an image of an old object (for example, the user wants to replace an old sofa with a new one). To do this, you need to solve the problem of deleting an object. Also, the user may want to change the floor or wallpaper.

Thus, the objectives of this work are:

• identify and draw up requirements;

• study analogues;

• study the theoretical foundations of neural networks;

• design an application, neural networks;

• develop an application and neural networks;

• train neural networks;

• test the application and neural networks.

## 1 Drafting requirements

In this work, two types of actors can be distinguished.

The first type is the user. The main consumer of the application. Based on the questions that the user wants answered during the renovation, the following functional requirements were drawn up:

• build a 3D model based on a real photo – using the user's input photo, you need to estimate its depth and build a 3D model based on this information;

• visualize 3D model – show the user the constructed 3D model;

• add 3D objects to a 3D model – enable the user to add 3D objects (for example, a sofa) to a 3D model built from his photo for visualization;

• remove objects from a photo – give the user the ability to paint over objects in a photo and delete objects that have been painted over;

• identify and replace floor, wallpaper, ceiling – find the floor (wallpaper, ceiling) in a photo and replace it with a pattern selected by the user;

• like objects – from the catalog of all 3D models, the user can select those that he likes to add them to his 3D model;

• order the necessary objects - those objects that fit the user in his 3D model, he can order.

In addition to the user, you can also highlight companies that want to sell their products. If the user likes the way, for example, the sofa in his room looks, he may immediately want to order it. Based on this, the company can:

• add your own objects to the application (wallpaper, parquet, sofa);

• receive orders for objects.

Non-functional requirements:

• Web application.

Consider possible use cases.

1) Replace the old sofa with a new one and order.

Main Actor: User
Main success scenario:

1. The user selects a new sofa in the catalog and adds it to himself.
2. The user takes a photo of an old sofa and uploads it to the app.
3. The application offers to remove the object from the photo or immediately build a 3D model.
4. The user chooses to delete the object.
5. The application opens the interface for removing the object from the photo.
6. The user paints over the old sofa and presses the process button.
7. The application removes the old sofa and builds a 3D model.
8. The user adds a new sofa model to their 3D model.
9. The user likes how the new sofa looks in his room, and he orders it.
10. The application generates an order.

Extensions:

9.a. The user does not like how the new sofa looks in his room.
9.a.1. The user selects a new sofa from the catalog and adds it to the 3D model.
9.a.2. The user likes the look of the new sofa and orders it.

2) Download objects.
Main Actor: Company
Main success scenario:

1. The company enters the file upload form in the application, uploads objects, adds a description.
2. The application loads objects with a description.

## 2 Overview of existing solutions

At the time of writing, there was no single application that has the full functionality that this work pursues. There are analogues that implement individual functions.

For example, the hama.app application allows you to remove objects from a photo (Figure 3 and Figure 4), but does not allow you to do anything further with the photo. And the Design a room application (Figure 5) allows you to create a highly realistic rendering of a room, but only from pre-existing objects, it is impossible to add an existing room to it. There are also a large number of AR applications (Figure 6) that allow you to see an object on the gadget screen in real time, but such applications require special equipment and efforts.

Figure 3 - hama.app, an example of deleting an object

Figure 4 - hama.app, an example of deleting an object

Figure 5 - Design a room, an example of a photorealistic render

Figure 6 - Amazon, View in your room, AR application example

As you can see, there are a large number of different applications that implement individual functions, but there is no one that combines them all. The purpose of this work is precisely to implement such an application.

## 3 Theoretical foundations

Before you start designing an application, it is worth dwelling in more detail on the tasks facing it, and understand the formal statement of tasks and how to solve them. Indeed, in addition to classical programming, most of this work consists in the implementation of functionality that requires the use of neural networks.
So, first we will consider the possibility of removing an object from an image.

### 3.1 Image inpainting

Image inpainting refers to the process of filling in missing data in a designated visual input area. The goal of the process is to restore missing parts or damaged images in such a way that the shaded area cannot be detected by the observer. Applications range from the removal of imperfections such as cracks, scratches and dirt in scanned photographs and digitized images of works of art, to the removal/introduction of image objects such as logos, date stamps, text, faces and special effects on stage. As a rule, after the user selects an area to restore, the shading algorithm automatically restores the damaged area.
More formally, the task posed in the framework of this work is as follows: given an input image and a mask that paints some part of it, it is required to produce an image in which the shaded regions are filled with something meaningful, suitable for the surrounding context. In this work, often the walls and floor will be such an ambient context, since in almost all cases when the sofa or bed is removed, the wall and floor are hidden behind them.

There are various approaches to solving this problem, the most successful of them are based on the use of neural networks. But all of these approaches experience difficulties when the mask size becomes large. In this section, we will consider the key ideas on which the neural network architecture for image shading will be built in the future.

The most popular approach to solving the problem of shading an image is the use of generative adversarial networks (GANs). They learn to generate images so that the generated example cannot be distinguished from the example from the

training dataset. To do this, a generator network is trained that generates data from input random noise, which is the hidden distribution parameters of a data set that cannot be estimated, and a discriminator network that tries to understand whether the input data is generated or real. With such a design, a competition occurs between networks, as a result of which they come to an equilibrium, as a result of which the generator network has learned to generate such data that the discriminator network can distinguish whether the data is real or not with a probability of 0.5. Such an example is an example of a GAN without limitation, since the input noise alone determines the output image.

The potential of GANs is huge as they mimic any distribution of data. GANs are taught to create structures eerily similar to the entities from our world in the field of images, music, speech, prose. Generative adversarial networks are, in a sense, robot artists, and the result of their work is impressive.

Let's consider how the discriminator and generator algorithms work.

Let's start with the discriminator. Discriminatory algorithms try to classify the input. Given the characteristics of the data obtained, they try to determine the category to which they belong.

For example, by running through all the words in a letter, a discriminating algorithm can predict whether a message is spam or not spam. Spam is a category, and a pack of words collected from e-mail are images that make up the input data. Mathematically, categories represent y and images represent x. The notation p(y|x) is used to mean "probability of y given x", which means "the probability that an email is spam given the given set of words".

So, discriminative functions match images with a category. They are only concerned with this correlation.

At the same time, generative algorithms do the opposite. Instead of predicting a category from available images, they try to fit images to a given category.

While discriminative algorithms care about the relationship between y and x, generative algorithms care about “where x comes from”. They allow you to find p(x|y), the probability of x given y, or the probability of patterns given a class

(Generative algorithms can also be used as classifiers. They can do more than classify input.)

One more idea about the operation of generative algorithms can be obtained by separating discriminatory models from generative ones in this way:

• Discriminatory models study the boundary between classes;

• Generative models model the distribution of individual classes.

Let's talk about how GANs work. One neural network, called the generator, generates new data instances, and the other, the discriminator, evaluates them for authenticity; those. the discriminator decides whether each data instance it considers belongs to the training dataset or not.

Let's say we're trying to do something a little more banal than recreating the Mona Lisa. We will generate handwritten numbers similar to those in the MNIST dataset. The purpose of the discriminator is to recognize genuine instances from a set.

Meanwhile, the generator creates new images, which it passes to the discriminator. He does this in the hope that they will be accepted as authentic, even though they are fake. The purpose of the generator is to generate handwritten digits that will be skipped by the discriminator. The purpose of the discriminator is to determine if an image is genuine.

The steps a GAN goes through are:

• The generator receives a random number and returns an image.

• This generated image is fed into the discriminator along with a stream of images taken from the actual data set.

• The discriminator accepts both real and fake images and returns probabilities, numbers between 0 and 1, with 1 representing the genuine image and 0 representing the fake one.

So we have a double feedback loop:

• The discriminator is in a valid image loop.

• The generator is in a loop along with the discriminator

Figure 7 shows the GAN diagram.

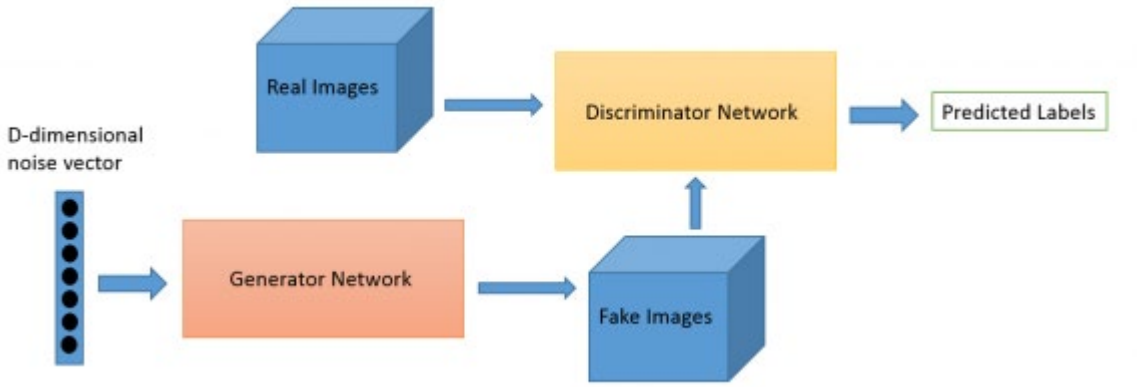



Figure 7 – GAN diagram

You can think of GAN as a counterfeiter and a cop playing cat and mouse, where the counterfeiter learns to make fake bills and the cop learns to spot them. Both are dynamic; i.e. the cop is also training (perhaps the central bank is marking missed bills), and each side comes to learn the other's methods in a constant escalation.

The discriminator network is a standard convolutional network that can classify images fed to it using a binomial classifier that recognizes images as real or fake. The generator is, in a sense, an inverse convolutional network: while a standard convolutional classifier takes an image and reduces its resolution to get a probability, the generator takes a vector of random noise and transforms it into an image. The former culls the data using downsampling techniques such as maxpooling and the latter generates new data.

Both networks try to optimize the objective or loss function in a zero-zum game. This is essentially the actor-critic model. When the discriminator changes its behavior, then the generator changes, and vice versa.

Due to this approach, the resulting network has a large variability, which is required in the problem of shading an object. But, on the other hand, the quality of the generated data suffers due to the large variability, since the noise is not limited by anything, due to which the output image is unpredictable, and, for example, instead of a painted refrigerator in the kitchen, a floor or a shower can be generated. This can be fixed by making the GAN limited (Conditional GAN). For example, you can specify the class of the dataset from which we want to generate the result: if the dataset contains animals and there are labels for each picture, which animal is shown

in the picture, then you can explicitly tell the GAN to generate an elephant or a corgi. And you can look at it from another angle - the input image is constrained: pixels that are not affected by the mask should not change. These pixels set the style context for our neural network, because it is required that the generated result fit the context of the original image. Thus, if we combine the variance of the unconditional GAN and the constraints of the Conditional GAN, then the resulting model should behave well on large masks, where unconditional models perform poorly due to high variance, and Conditional GAN, on the contrary, lacks it.

The proposed combination of variability and limitations is a new approach to the shading problem. Further in the chapter on choosing the architecture of neural networks, we will talk about the implementation of this approach.

Now let's talk about the specifics of the problem considered in this paper. As already mentioned, when removing an object (sofa, table, bed), there will most often be walls and / or floors behind it, and, almost always, there will be a line of contact between them. But the neural network cannot always make a straight line, since it is not explicitly indicated that the line needs to be continued, which means that the behavior of the generated line cannot be predicted and it can go very crooked, or there may not be a line at all.

In order to overcome this problem, we can explicitly pass the lines that we expect there to the neural network. To do this, it is proposed to predict the boundaries of objects in the original image and use another GAN that would predict their continuation on the areas filled with the mask.

Edge detection includes a variety of mathematical techniques aimed at detecting edges, curves in a digital image, where the brightness of the image changes dramatically or, more formally, has discontinuities. The same problem of detecting discontinuities in one-dimensional signals is known as step detection, and the problem of detecting discontinuities in signals over time is known as change detection. Edge detection is a major tool in image processing, machine vision and computer vision, especially in the areas of feature detection and extraction.

Thus, first, the existing lines of objects are found from the input image, then the lines are completed by the first GAN in the areas that turned out to be filled with a mask, after which the input image and the restored lines are transferred to the final GAN, which paints the object taking into account the predicted lines.

Details of the implementation and training of the final approach will be discussed later.

### 3.2 Image Depth Estimation

When solving the problem of building a 3D model from one photo from the camera, the question may arise: “where can I get the third coordinate?”, Because the photos are two-dimensional. Since the goal of this work is to make the construction of a 3D model and the entire virtual renovation as simple and accessible as possible, no additional sensors or real-time video recording (as, for example, for an AR application) is required. It turns out that the task of estimating the depth of the image arises. Neural networks have been used for this task for a long time. Since this work is quite specific, some assumptions can be made that can improve the results of the assessment. For example, it can be assumed that almost all rooms in the world are Manhattan, that is, they consist of parallel and perpendicular planes. This information can be used to obtain more accurate results. Since intuitively, if we, as humans, use this information, then the neural network can use it to improve results. Thus, the proposed approach for estimating the depth of an image will be as follows: first, feature maps will be extracted from the image, for which any convolutional neural network (CNN) is suitable, after which maps will be obtained using the Feature Pyramid Network (to improve work with small objects). special signs of different sizes. These maps will be passed on to the block, which will simultaneously evaluate the depth and plane masks (as well as the unique number of each plane). This approach will allow the depth extraction branch to use the plane information and improve its estimation.

Details of training and architecture will be outlined later in this paper.

### 3.3 Image segmentation

In digital imaging and computer vision, image segmentation is the process of dividing a digital image into multiple image segments, also known as image regions or image objects (sets of pixels). The purpose of segmentation is to simplify and/or change the representation of an image into something more meaningful and easily parsable. Image segmentation is commonly used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to each pixel in an image so that pixels with the same label share common characteristics.

The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image. Each of the pixels in an area is like some characteristic or computed property, such as color, intensity, or texture. Neighboring regions vary significantly in color for the same characteristic(s). When applied to an image stack typical of medical imaging, the resulting contours after image segmentation can be used to generate 3D reconstructions using interpolation algorithms such as marching cubes.

Image segmentation means assigning a class to each pixel. Within the framework of this task, this is necessary: firstly, to determine where the floor, wall, ceiling is located, for the subsequent replacement of the texture; secondly, object masks can be used to simplify the removal of objects - instead of manually painting the object, the user can simply select the proposed object mask.

The segmentation problem is well studied and modern approaches give excellent results, so for this problem you can use existing solutions, rather than invent your own, taking into account the specifics of the problem, as in previous cases.

Details of training and architecture will also be outlined later in this paper.

# 4 Application design

## 4.1 Domain model

Figure 8 shows the domain model of the application being developed, which describes the main entities and relationships between them.

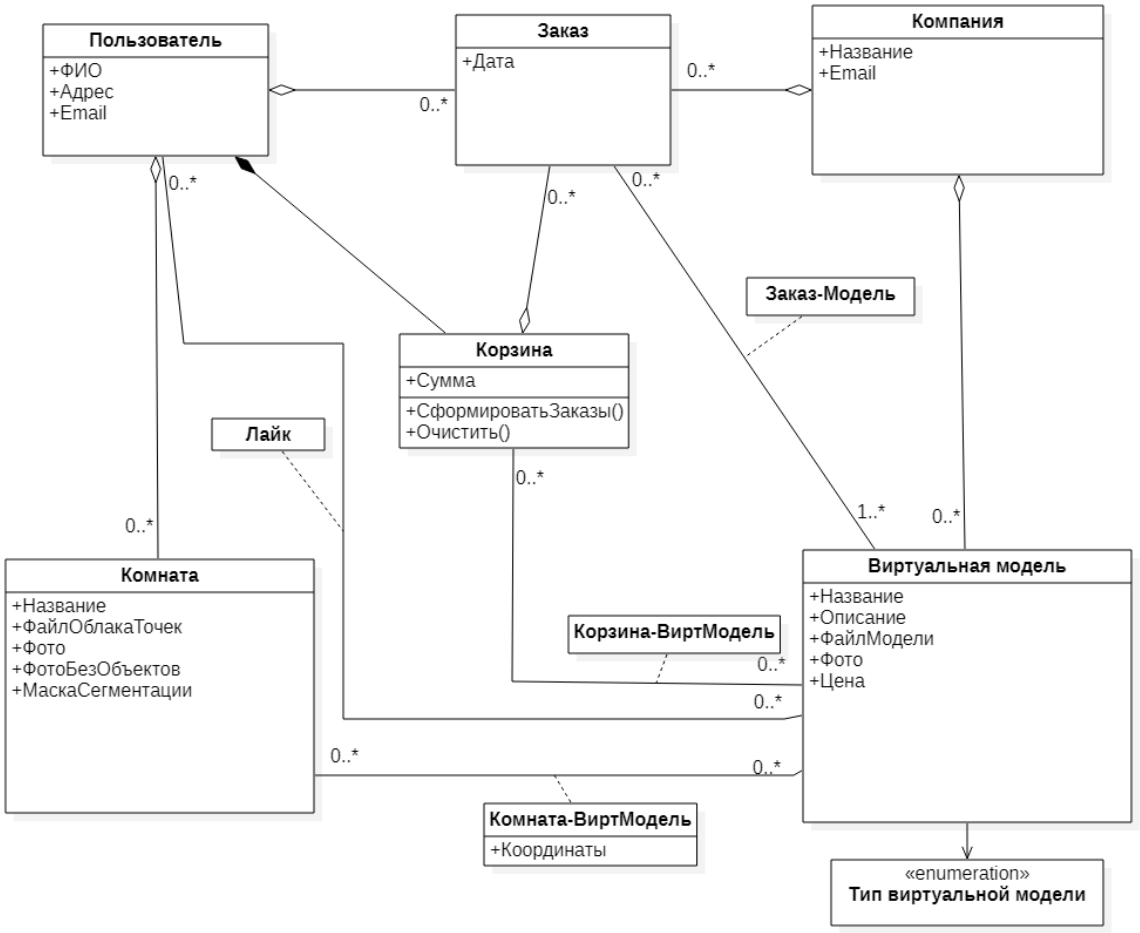



Figure 8 - Domain Model

Let's take a closer look at each class:

• User - a class representing a user and storing information about him.

• Company - a class representing a company and storing information about it.

• Room – a class that represents a room that the user loads into the system and whose photo is used for the main actions. Initially, this class stores the user's name and photo. After processing, a point cloud file and a segmentation mask will appear. A photo without objects may also appear if the user wants to remove the object from the photo.

• Virtual Model – a class that represents all kinds of virtual objects – wall, floor, sofa, table… All options are listed in the Virtual Model Type class and will be described later.

• Cart - a class that represents the user's current cart. The user adds items to the cart that he would like to purchase. When buying, the shopping cart generates orders, in each of which the objects are grouped by company. After placing an order, the basket is cleared.

• Order - a class representing an order and storing information about it. In particular, an order stores information about the user, the company, and the products that were ordered.

• Like - a class that reflects the fact that the user liked this or that object. The user can add objects that he likes to his 3D room model.

• Type of virtual model – a class representing an enumeration of all available types of objects. At the time of writing this is:

o parquet;

o linoleum;

o laminate;

o paint;

o sofa;

o table;

o chair;

o armchair;

o closet;

o shelf;

o rack;

o bed;

o bench;

o stool;

o cradle;

o pouf;

o bureau;

o couch;

o wallpaper;

o sideboard;

This list may be expanded at any time.

### 4.2 Choice of application architecture

Consider now the application architecture shown in Figure 9. Since the non-functional requirements dictate the need for the developed application to be a web application, it would be most reasonable to choose the MVC architecture (Model-View-Controller).

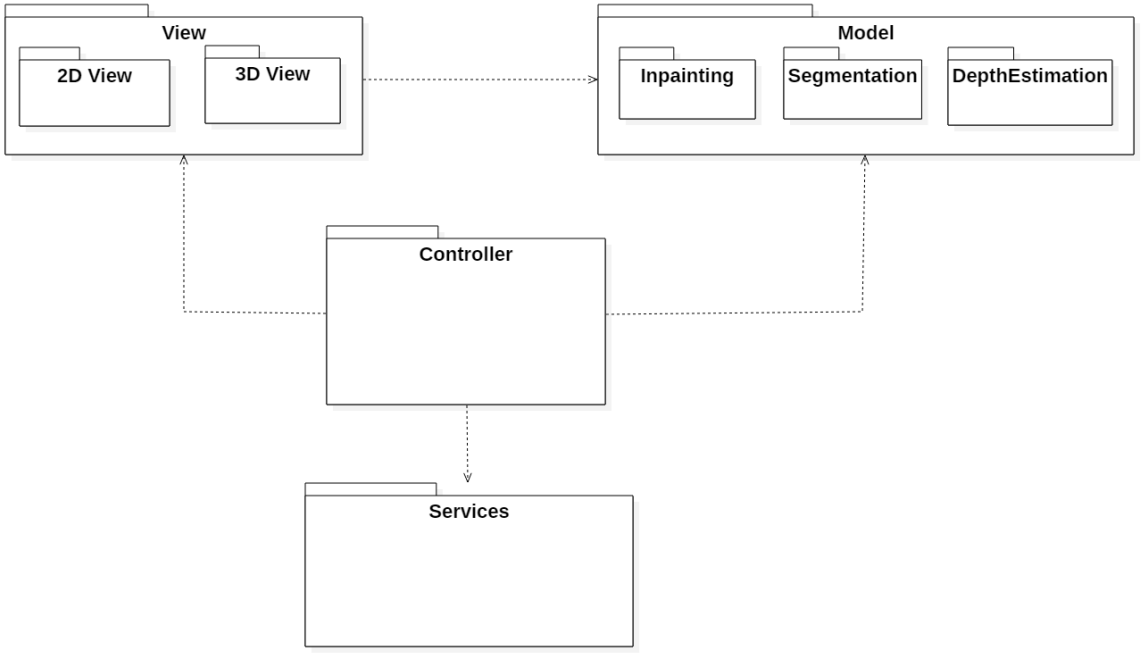



Figure 9 - Application architecture

In this architecture, the logic of the application is concentrated in the Controller package, which contains various controllers for processing requests to the application.

The View package contains the pages that the user sees, as well as the 2D View and 3D View packages, which are visualizers of 2.5D photographs and 3D

models. Learn more about 2D and 3D View in the Application Implementation chapter.

The Model package stores the normal application classes, as well as packages that contain classes that are responsible for processing the user's input photo. They were placed in the Model package because the View package needs to receive a 3D model of the room and other representations to display to the user.

The Inpainting package contains the InpainterWrapper class (Figure 10), which is a wrapper over a neural network that is responsible for painting an object from a photo.

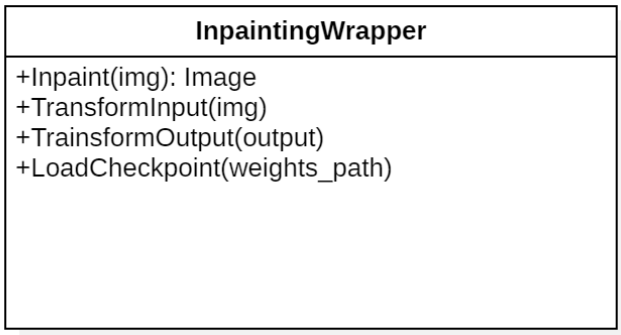



Figure 10 - Class InpaintingWrapper

This class loads the weights of the neural network, performs the necessary transformations of the input image and mask, paints the object using the neural network, performs the transformation of the result, and returns the image with the object removed.

The DepthEstimation package contains the DepthEstimationWrapper class (Figure 11), which is a wrapper over a neural network that is responsible for estimating the depth of a photo.

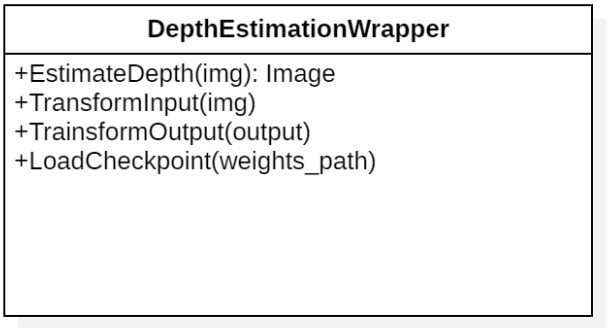



Figure 11 – Class DepthEstimationWrapper

This class loads the weights of the neural network, performs the necessary transformations of the input image and mask, uses the neural network to estimate the depth of the input image, performs the transformation of the result, and returns the estimated depth.

The Segmentation package contains the SegmentationWrapper class (Figure 12), which is a neural network wrapper that is responsible for segmenting the input image.

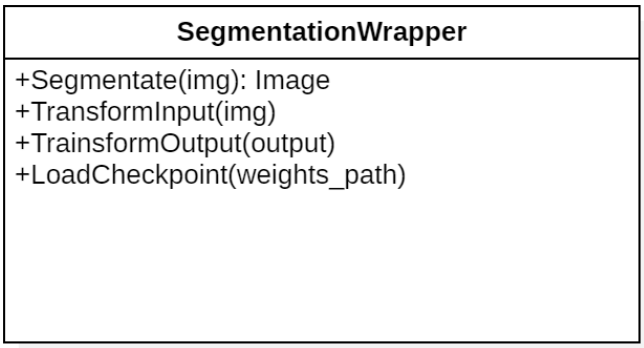



Figure 12 – Class SegmentationWrapper

This class loads the weights of the neural network, performs the necessary transformations of the input image and mask, segments the input image using the neural network, transforms the result, and returns the segmentation mask.

The Services package (Figure 13) contains various services that help in the implementation of the application. The most important of these are the user authorization service (AuthService), the broker service (BrokerService) and the worker service (WorkerService).

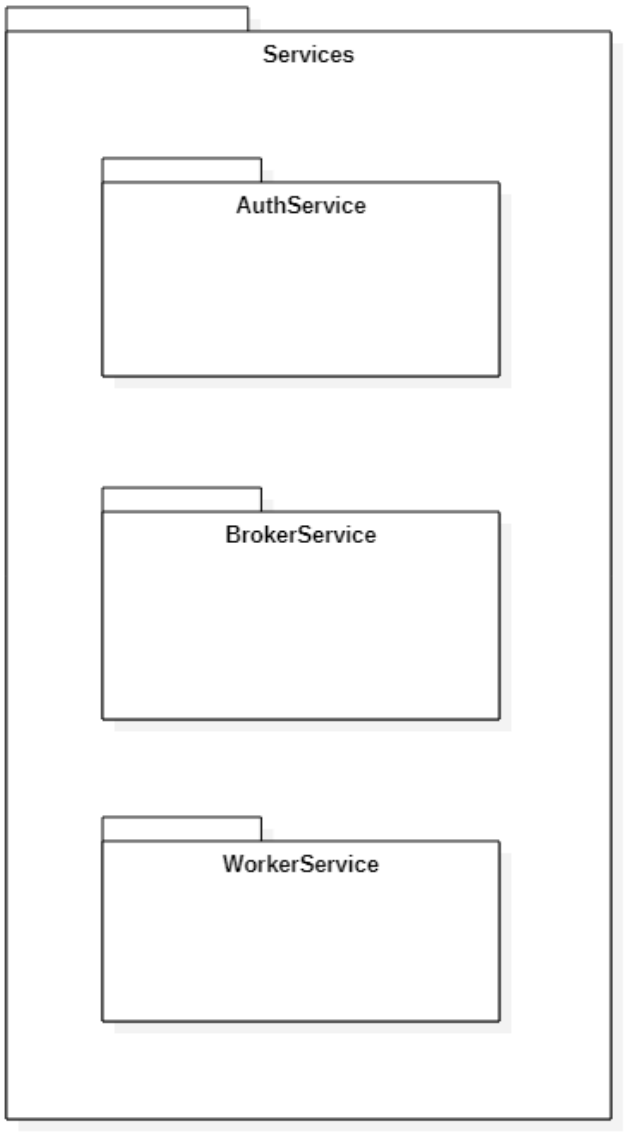



Figure 13 – Package Services

Processing by neural networks takes some time, so a situation may arise when two users request to paint an object at the same time, but the video memory resource in the system is enough to serve only one of them. To cope with such situations, we need the services BrokerService and WorkerService, which will allow us to implement the task mechanism.

When a user wants to create a 3D model from a photo or remove an object from a photo, the controller will create a task and the BrokerService will store it. After that, the WorkerService will pick up this task and execute it, put the result in

the right place and notify the user that the task has been completed. An example of this work can be seen in Figure 14.

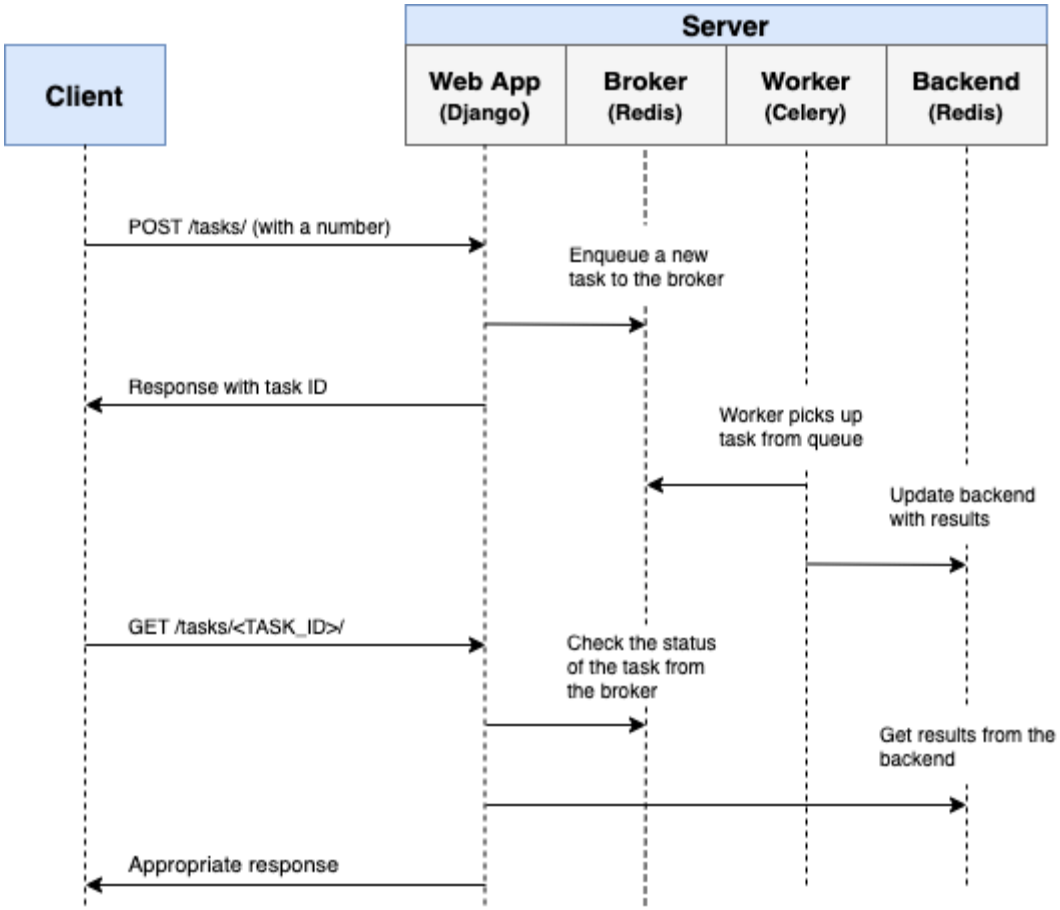



Figure 14 – An example of how the task mechanism works

### 4.3 Designing the architecture of neural networks

Let us consider in more detail the architecture of neural networks for each task.

Let's start with a neural network for painting an image. Since StyleGAN2 is the most powerful model for unconditional generation of images, the StyleGAN2 architecture shown in Figure 15 will serve as the framework for the created model.

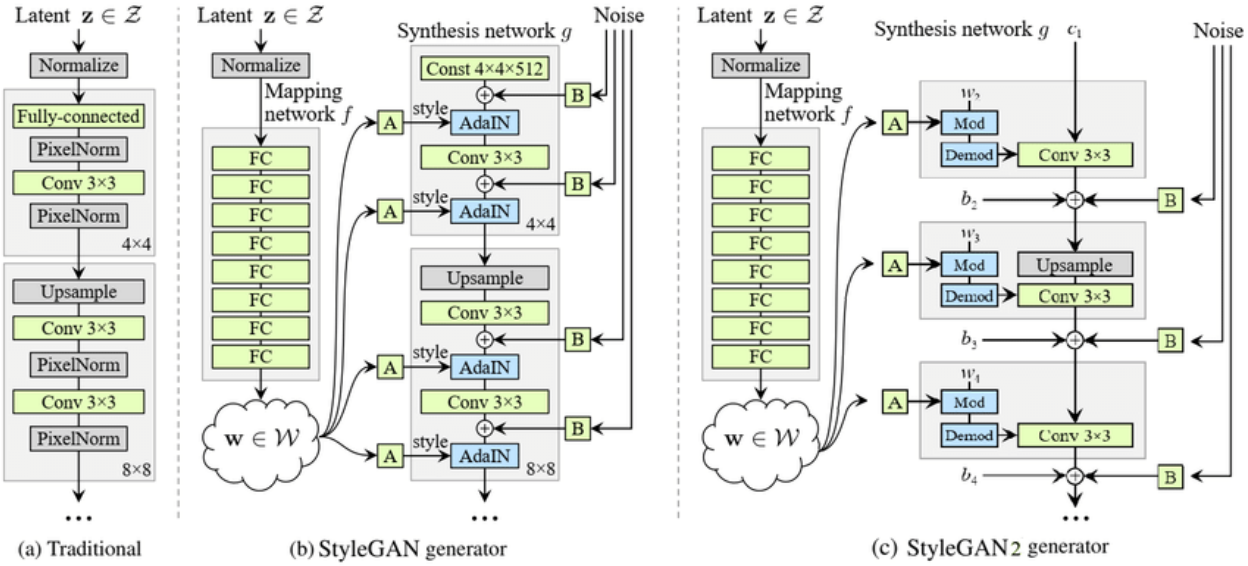
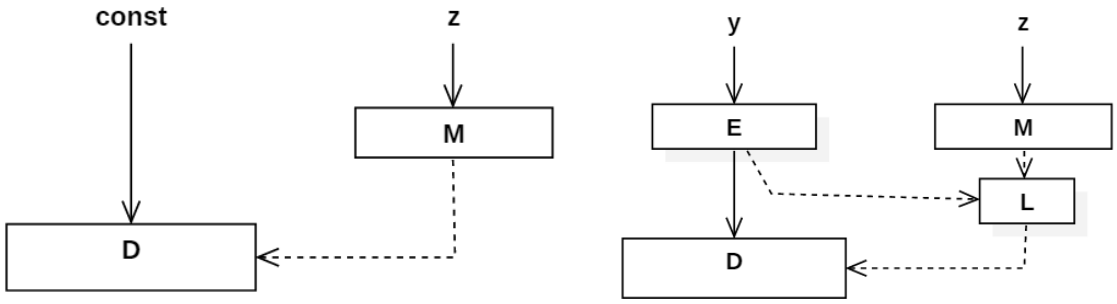



Figure 15 - (a) Original architecture (b) StyleGAN architecture (c) StyleGAN2 architecture

In the case of StyleGAN2, the model starts the synthesis with a constant that sets the base of the image, after which the image is gradually increased in size, noise is added to it to change the image content and style. The most interesting place is the modulation block for modulating the weights. This method allows you to apply a style to the input, in this case an image.

This is where the stochastic properties of the unconditional GAN and the limitations of the Conditioned GAN will be mixed. Figure 16 shows a diagram of combining the two approaches. First, the input image is encoded into a feature vector, which is the input image context responsible for the content of the output image; also, the input noise responsible for the stochastics is encoded into a vector. Next, they are combined using a linear combination and the combined style is applied using weight modulation.




(а) (б)

Figure 16 - (a) StyleGAN2 unconditional modulation; (b) modulation in the developed architecture

Thus, the developed architecture is the StyleGAN2 architecture with a modified modulation block to add a constraint on the input image, as well as another network that encodes the input image.

Let's talk about the network for depth estimation. As mentioned earlier, due to the nature of the work, finding the planes simultaneously with the depth estimate should improve the accuracy of the estimate. Therefore, the following architecture was developed.

First, the input image is passed through a convolutional neural network, thus obtaining feature maps. After that, these maps are transferred to the Feature Pyramid Network (Figure 17), which allows you to work better with objects of different scales. The output from this network is used to predict planes in the input image. To do this, it is driven through a series of convolutional layers, after which the segmentation head segments by class and entity number. The penultimate layer of this network is fed to the input of another branch, which originates from the feature maps coming out of the convolutional network. Thus, the depth estimation branch uses information from feature maps and plane estimation to improve the depth estimation of the input image. This information is concatenated, run through a layer of convolutions and a network prediction is obtained.

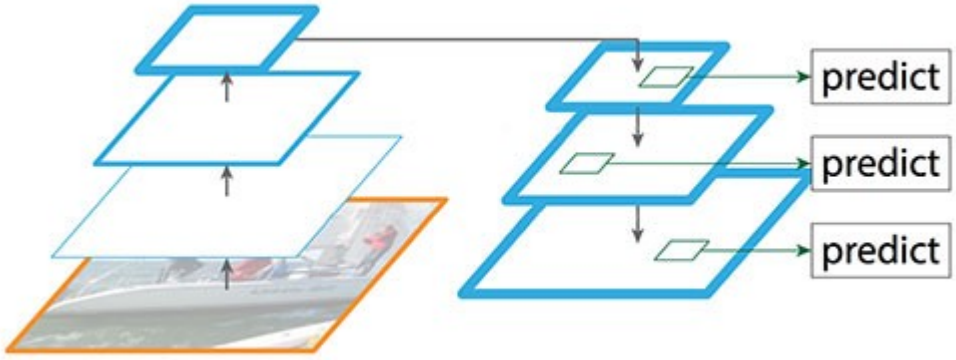



Figure 17 – Feature Pyramid Network

All that remains is a network for image segmentation. As stated earlier, already implemented networks that are in the public domain can be used for this task. In this work, the HRNetV2 network was chosen, the architecture of which is shown in Figure 18.

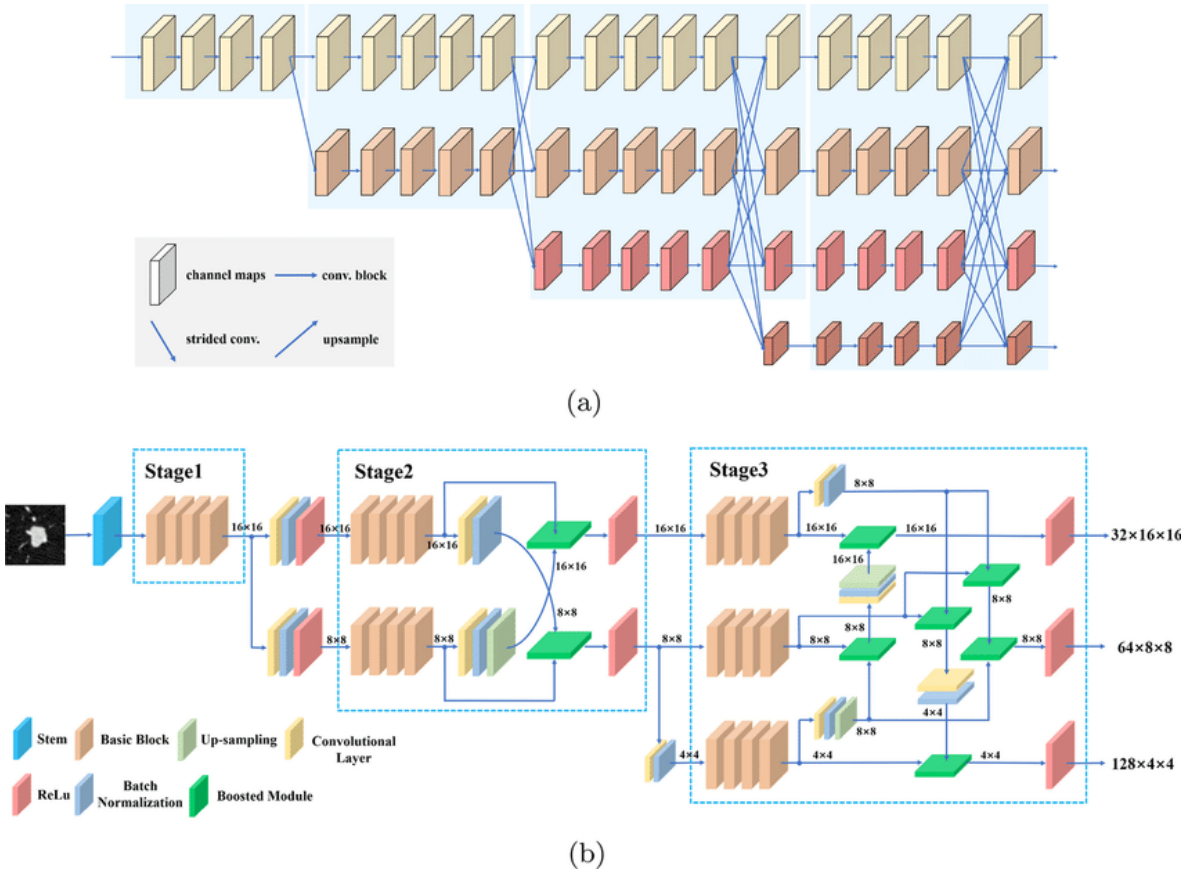



Figure 18 – HRNetV2

# 5 Application Implementation

## 5.1 Choice of technologies

To implement neural networks, the Python programming language was chosen, since neural networks are, as a rule, programs that are not very large in volume, but which often have to be changed in order to select the optimal architecture, data preprocessing, and so on. Therefore, there are requirements for rapid development. Python does this better than C++ or Java. There are also a large number of frameworks that simplify writing your own neural networks due to the fact that they take on the implementation of automatic differentiation, memory management, and more. Also, Python has recently become the standard for writing neural networks.

To speed up development, the PyTorch framework was chosen.
PyTorch is an open source machine learning framework for the Python language based on Torch. It is used to solve various problems: computer vision, natural language processing. Developed primarily by Facebook's artificial intelligence group. PyTorch provides two main high-level models:

• Tensor computing (similar to NumPy) with advanced support for GPU acceleration.

• Deep neural networks based on the autodiff system.

PyTorch is also the standard in deep learning lately.

PyPCL, a Point Cloud Library (PCL) bind for the Python language, is used to post-process the results of neural networks.

Point Cloud Library is an open source algorithm library for point cloud and 3D geometry processing tasks, such as in 3D computer vision.

To implement the web application itself, it was decided to use Python for easier compatibility and work with neural networks. Django was chosen as the framework.

Django is a free Python web application framework that uses the MVC design pattern. A Django site is built from one or more applications, which are recommended to be alienable and pluggable. This is one of the significant architectural differences of this framework from some others (for example, Ruby on Rails). One of the basic principles of the framework is DRY (Don't repeat yourself). Also, unlike other frameworks, Django's URL handlers are explicitly configured using regular expressions.

To work with the database, Django uses its own ORM, in which the data model is described by Python classes, and the database schema is generated from it. Thus, it is ideal for developing the required application: it allows easy integration with neural networks, implements its own ORM and MVC design pattern, and also has a powerful templating system for writing web pages and allows you to easily embed the Bootstrap CSS framework to improve the appearance. pages.

The PyTest framework was used for testing.

For 2.5D and 3D visualizations, the Three.js library is used.

Three.js is a lightweight cross-browser JavaScript library used to create and display animated 3D computer graphics in web application development. Three.js scripts can be used in conjunction with the HTML5 CANVAS element, SVG or WebGL.

This library makes it easy to embed 3D graphics on the Web.

Also, Redis and Celery, respectively, are used to implement the BrokerService and WorkerService services.

Redis is an open source NoSQL resident database management system that works with key-value data structures. It is used both for databases and for the implementation of caches, message brokers. Focused on achieving maximum performance on atomic operations.

Celery is an open source asynchronous task queue or job queue based on distributed message passing. While it supports scheduling, it focuses on real-time operations.

Finally, the PostgreSQL database was chosen to store the information.

PostgreSQL is a free object-relational database management system (DBMS).

Thus, the technology stack is:

•Python:

o PyTest;

o Django;

o PyTorch;

o PyPCL;

•JS:

o Three.js

• CSS:

o Bootstrap;

• Celery;

• Redis;

• PostgreSQL;

### 5.2 Preparing datasets

Neural networks need data to train. For this job you need:

• images of interiors and corresponding depths - for training a neural network to estimate the depth of the image;

• images of interiors - for training neural networks for shading objects;

• images of interiors and segmentation masks - for training neural networks for image segmentation.

Fortunately, there are a large number of datasets that contain photographs of interiors. Consider those that were chosen for this work.

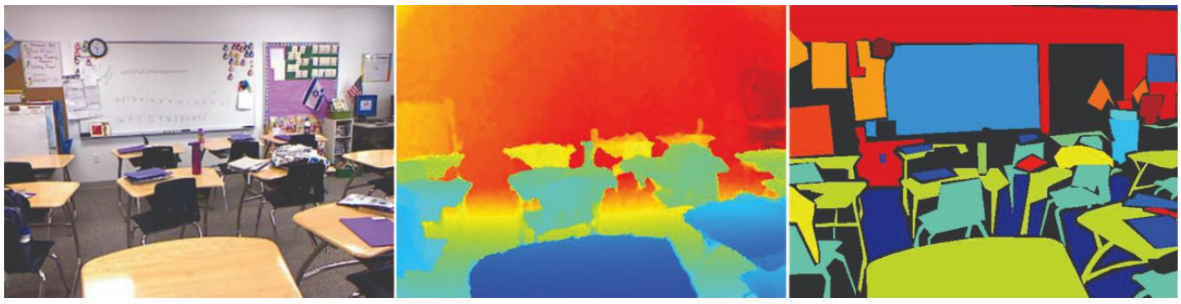
Places365 is a dataset containing a large number of different images of interiors and information about the appearance of the scene. Figures 19-20 show sample images from this dataset.

Figure 19 - Image from the Places 365 dataset

Figure 20 - Image from the Places 365 dataset

Places 365 contains almost 7 million images of various scenes, which gives greater variability in training neural networks, improving their generalization.

To train neural networks to estimate the depth of an image, it is necessary to prepare a data set in the form of a “picture - depth map”. The NYU-Depth2 dataset is well suited for this. It contains 1449 pictures of interiors, corresponding depth maps made on Microsoft Kinect, and a segmentation mask. Figure 21 contains an example of data from this set.



Output from the RGB camera (left), preprocessed depth (center) and a set of labels (right) for the image.

Figure 21 - Image from the NYU-Depth2 dataset

Also in the idea of depth prediction in the chapter on theoretical foundations, it was proposed to simultaneously predict the depth of an image and the planes represented on it. The ScanNet data set is well suited for this task. It contains marked-up interiors in the form of 3D scans, which are ideal for teaching the developed model. Figures 22 and 23 provide sample data.

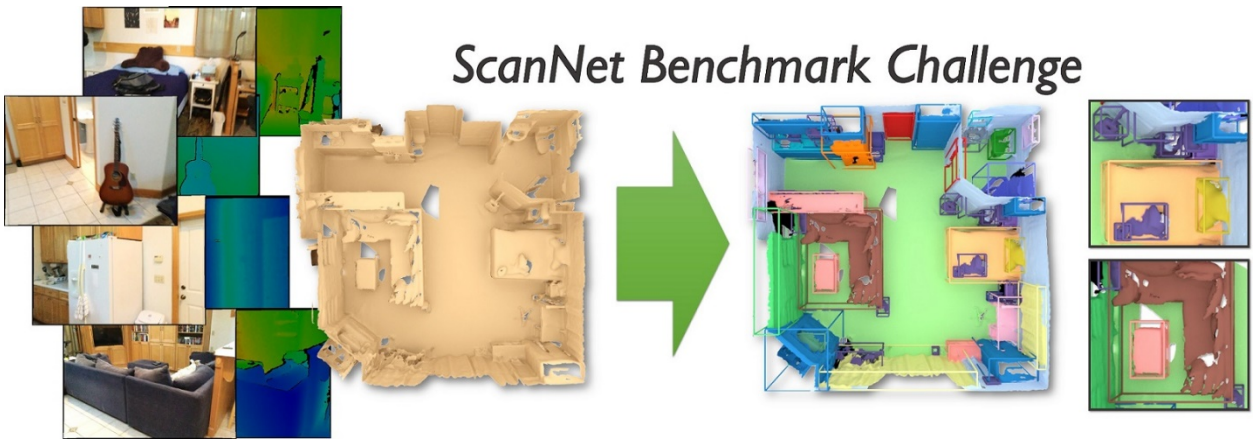



Figure 22 - Example from the ScanNet dataset

It is worth noting that in order to get photos from a 3D model, we had to pre-process the dataset, after which the photos were obtained.

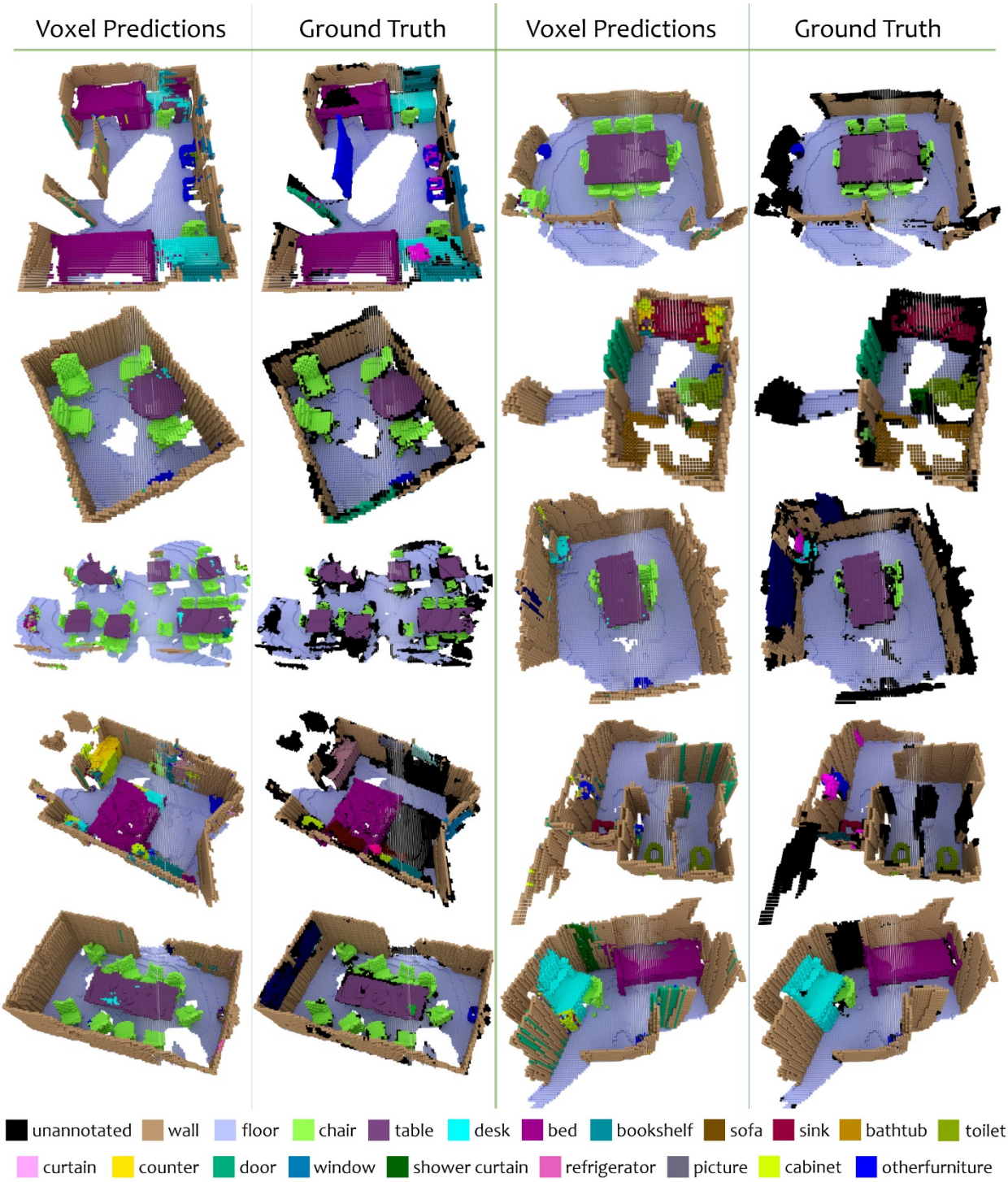



Figure 23 - Example from the ScanNet dataset

To train segmentation models, you can use the well-known MIT ADE20K dataset. This dataset contains 20 thousand pictures of interiors and segmentation masks for them. Figure 24 contains sample data.

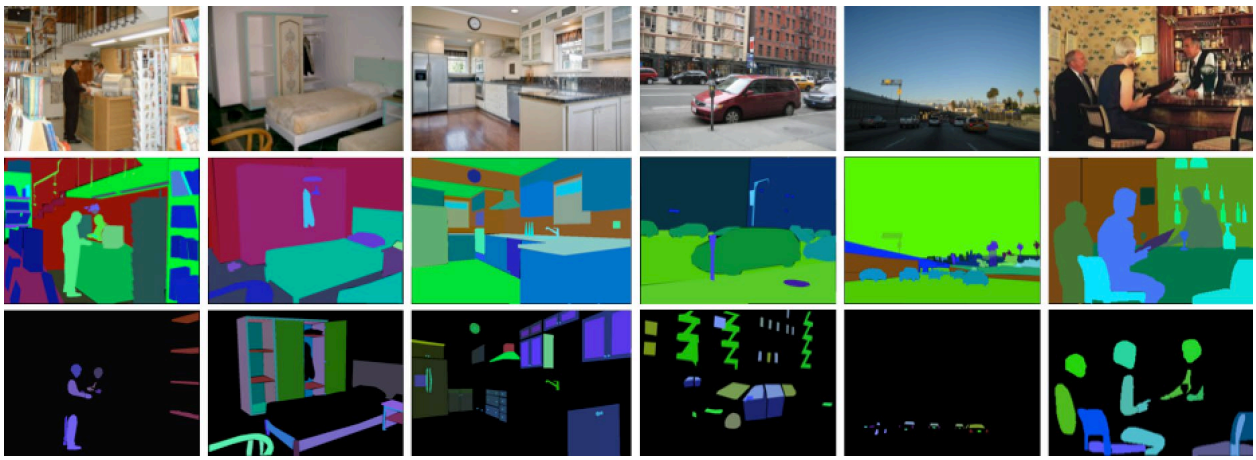


Figure 24 - Example from the MIT ADE20K dataset

Thus, all data sets were downloaded, preprocessed and prepared for training neural networks. A dataset of various photographs of interiors was also prepared by scraping. There is only one question left.

To train the network to remove objects, you need a set of masks that mimic the masks that the user makes. For such a simulation, the following approach was developed.

Firstly, some of the masks for deleting an object were taken from images that have segmentation masks. After pre-processing, all the masks on the image were discarded except for one representing the object of interest, for example, a sofa. And if there was a pillow on the sofa in the photo, then its mask was combined with the sofa mask, and thus the final mask was obtained to remove the sofa.

Secondly, to simulate masks with fuzzy contours, and with a free form, an approach was developed using the random walk algorithm (Figure 25).

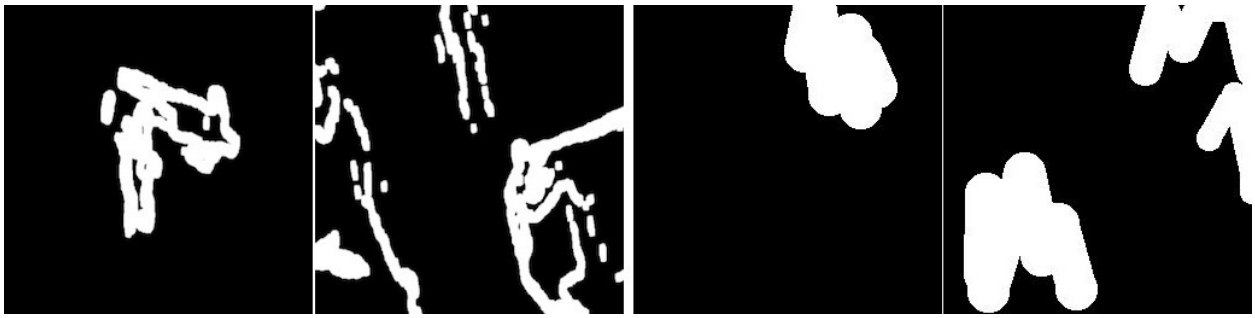


Figure 25 - An example of freeform masks

As a result of all these actions, all the data has been prepared, you can begin to implement neural networks and train them.

### 5.3 Implementation of neural networks

The PyTorch framework was used to implement neural networks. After the networks were implemented and trained, they were converted to the TensorRT format for faster execution when using the application. Correct writing and debugging architecture and training is the most time-consuming and difficult part of the work. Often training had to be started from scratch after it became clear that the experiment had failed.

### 5.4 Training of neural networks

The neural networks were trained on an NVIDIA 2060 Max-Q Design graphics card. Since the video card is located on a laptop, training had to be carried out intermittently. The total duration of network training was almost a month. Due to resource constraints, the training was not completed.

To train the neural network for painting objects, non-saturating logistic loss and L1 regularization were used.

Guidance Loss was used to train the neural network for depth estimation.

Cross Entropy Loss was used to train the neural network for segmentation.

### 5.5 Implementation of Neural Network Results Improvement

As a result of studying the results of the work of neural networks, it was noticed that the predicted depth does not always make the planes flat, because of which, for example, the floor does not always appear flat. To solve this problem, a post-processing algorithm was developed. It consists in the fact that, knowing the segmentation mask, you can determine which image points belong to the floor, and then, using the plane equation, perform optimization to bring the floor to a flatter

form with the restriction that the points that are articulated with the points walls cannot be changed so as not to spoil the appearance of the 3D model.

### 5.6 Application Implementation

The application framework is the server part, written in Django, it contains the main application logic.

The brain of the system are neural networks that are responsible for processing images and 3D models. The main difficulties and labor costs in the development were associated with them. Open3D and PCL libraries were used for post-processing.

To visualize 3D models, the Three.js engine was used, which simplifies working with 3D graphics. However, we had to implement translation, rotations, collisions, selection of objects and other things separately. Object inpainting is also implemented using JavaScript.

To ensure that image processing does not make the user wait, processing was moved to a background process using task queues implemented using Redis and Celery.

Consider an example of a sequence diagram describing the implementation of the scenario for creating a 3D room in Figure 26.

The user adds a photo of the room, after which the information about the room is added to the database, and the task broker creates a task to create a 3D model of the room. The user receives an immediate response. After some time, the worker requests a task from the broker, accesses the wrapper over the neural network to estimate the depth, receives a 3D model of the room, then accesses the wrapper over the neural network to segment the image, receives a segmentation mask, improves the 3D model using this mask using the developed algorithm and saves the information to the database.

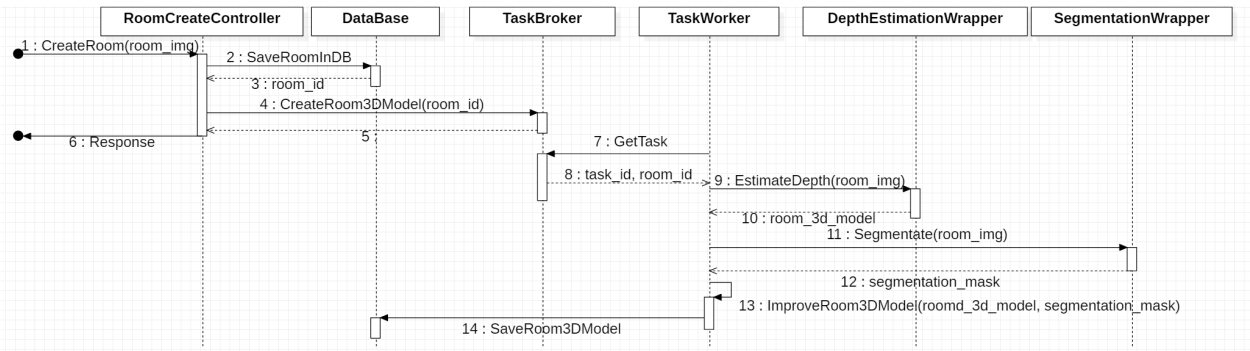



Figure 26 - Sequence Diagram for Room Creation Scenario

The object is painted in the same way, the only difference is that the input is a mask drawn by the user, and only the neural network is called to paint the image.

## 5.7 Application testing

In the best traditions of software development, the application being developed is covered with tests. The tests were developed using the PyTest framework. Tests include:

• unit tests that test individual functions and classes;

• integration tests that check whether separate application modules work well with each other, as well as check work with the database;

• functional tests to check the adequacy of the application's behavior;

• tests of neural networks;

• tests on users.

## 6 Application performance analysis

Let's look at some examples of how the application works.

Figures 27-29 contain examples of the results of neural networks. First, the original image and its segmentation mask are shown, then the image with the object removed and its segmentation mask, and then an example of image depth estimation. Figure 30 contains an example of a 3D model.

Figure 27 - Example of segmentation network operation

Figure 28 - An example of the segmentation network

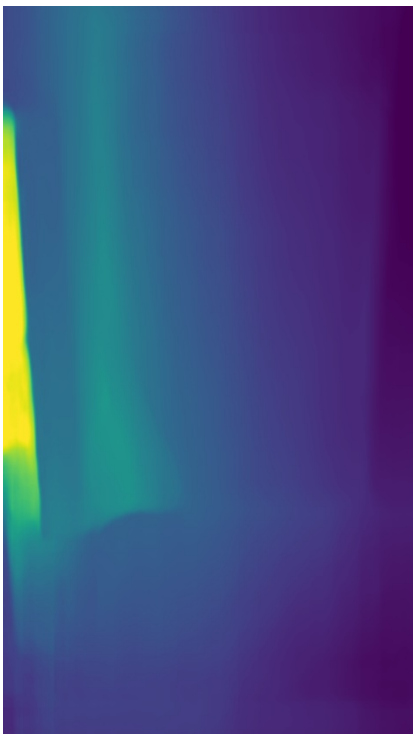


Figure 29 - An example of the operation of the network for depth estimation

Figure 30 - An example of the constructed 3D model

Similar steps are shown in Figures 31-32. You can see that due to the presence of photo wallpapers, neural networks are experiencing difficulties. You can see that

the bridge on the photo wallpaper sticks out, as a result of the fact that the network perceives it as a bridge, and not as a picture on the wall.

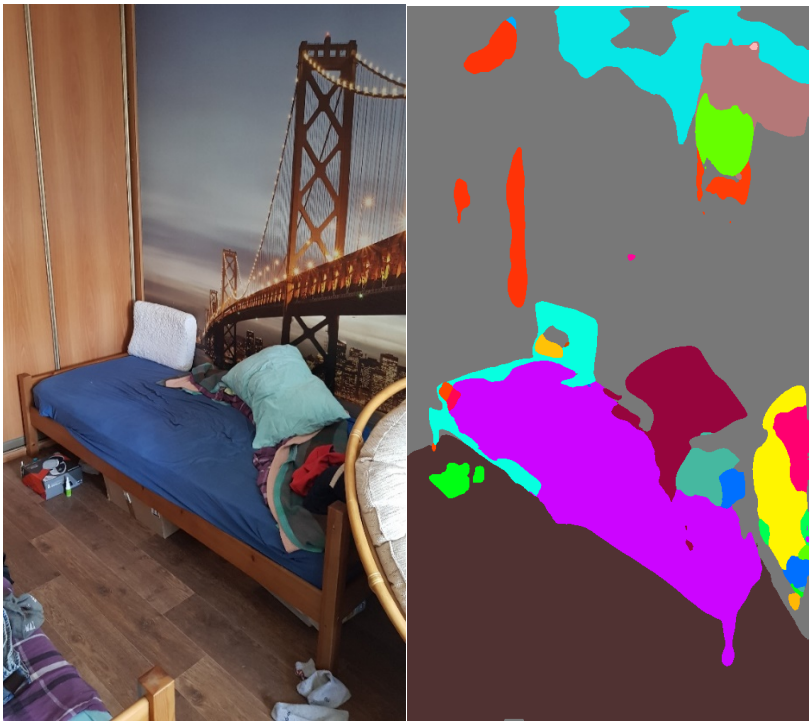


Figure 31 - An example of the segmentation network

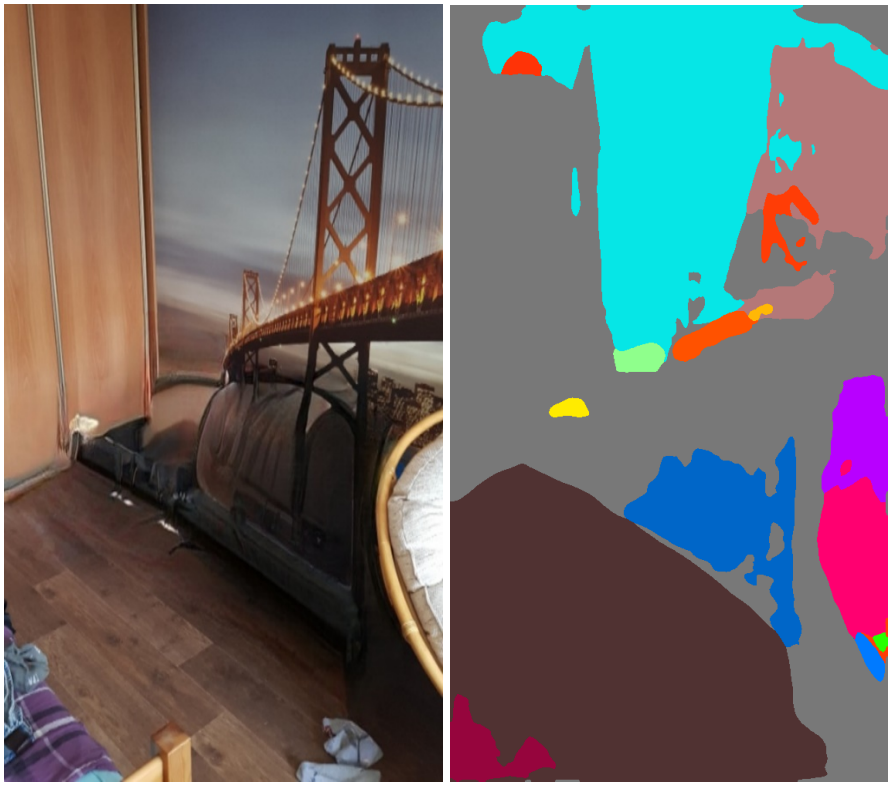


Figure 32 - An example of the operation of a segmentation network

Figure 33 - An example of the constructed 3D model

Figure 34 - An example of the constructed 3D model

It can be seen that neural networks cope with the tasks assigned to them even in rather extreme conditions. Built in 3D models, you can add objects and visualize how they will look in the room.

Figures 35-54 show a typical user-side usage scenario in screenshots.

The user enters the product catalog, selects the one he likes, creates his own room model from the photo, deletes the old object from the room, and adds the product to his room model.

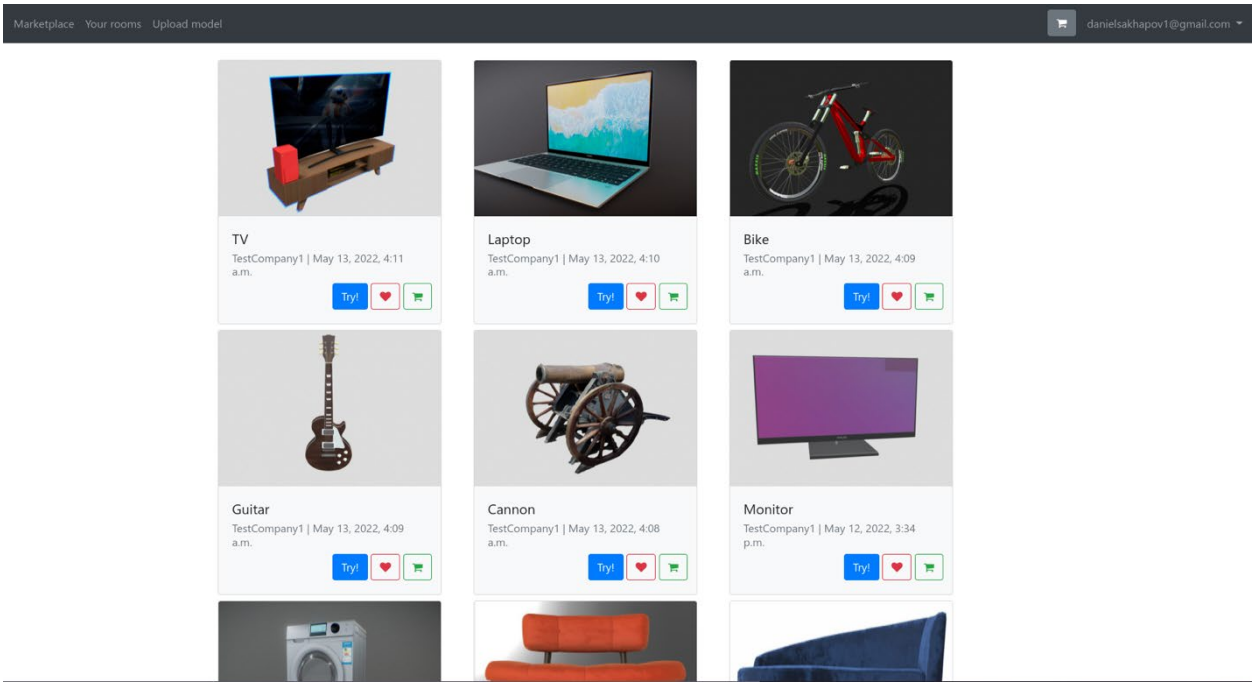



Figure 35 - Product catalog

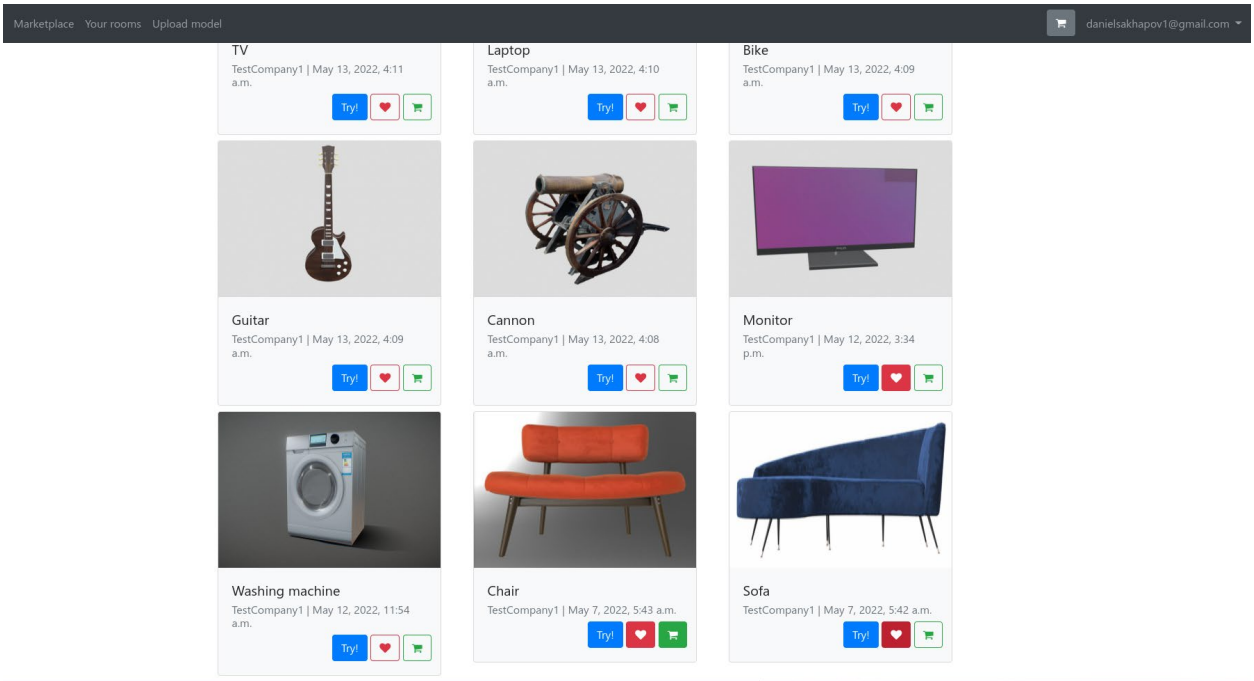



Figure 36 - The user marks the products he likes

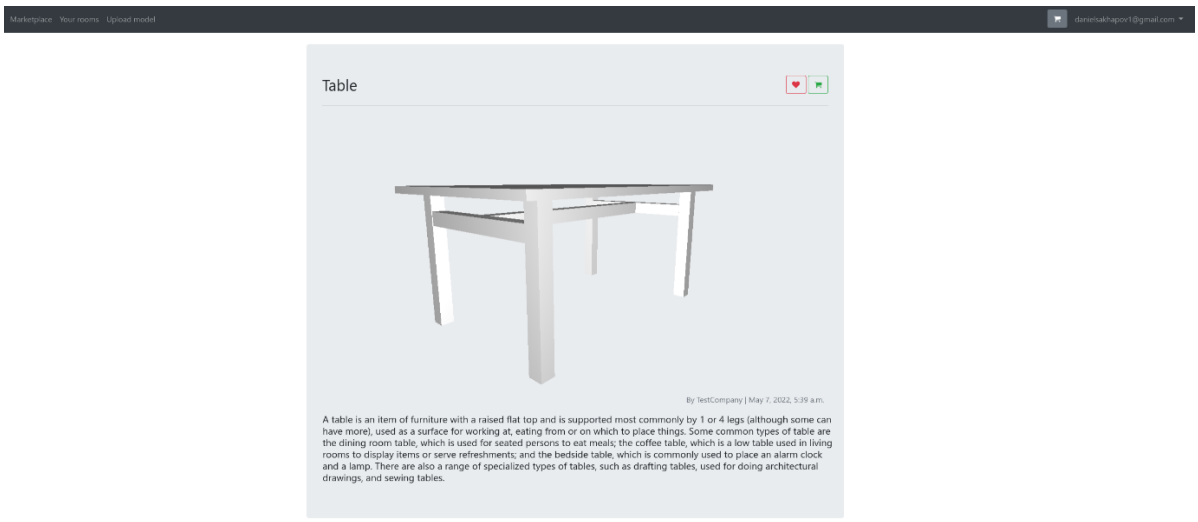



Figure 37 - Model description and 3D preview

The user likes the model he likes and uploads a photo of the room, after which the photo turns into a 3D model after background processing.

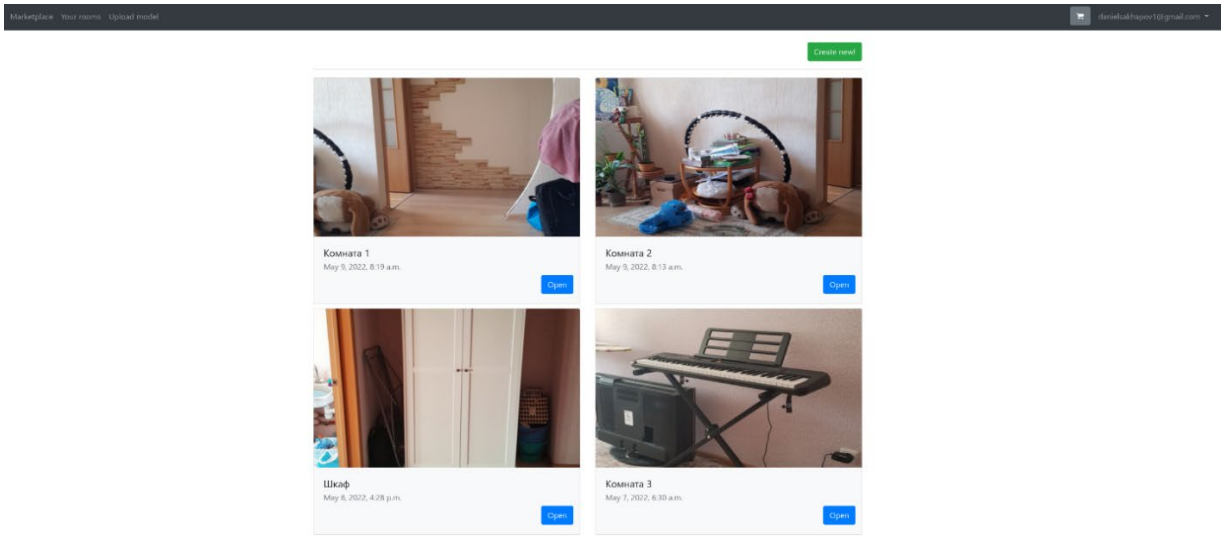



Figure 38 - User rooms

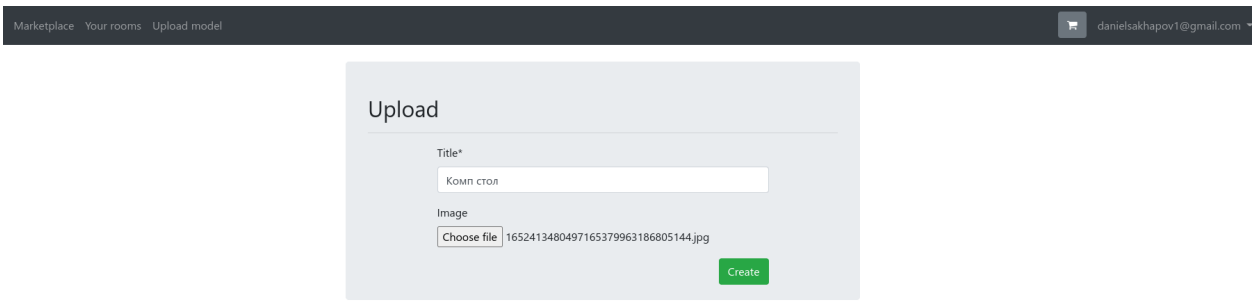



Figure 39 - Room loading window

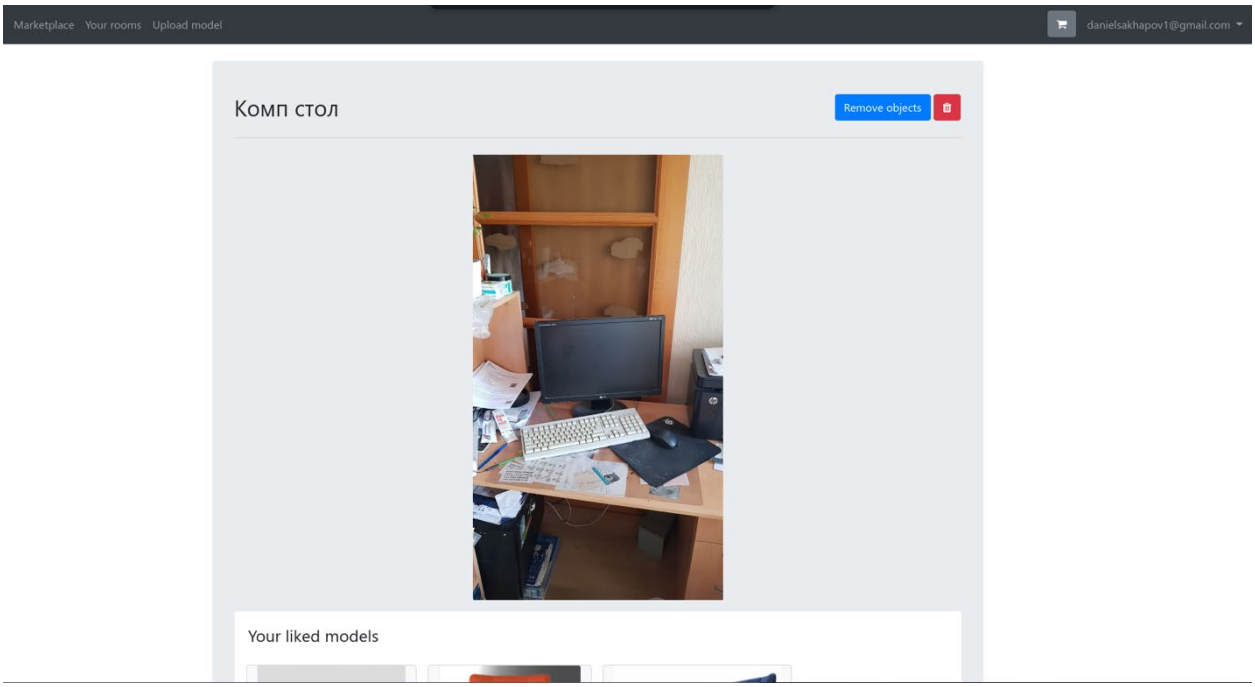



Figure 40 - User room after image upload

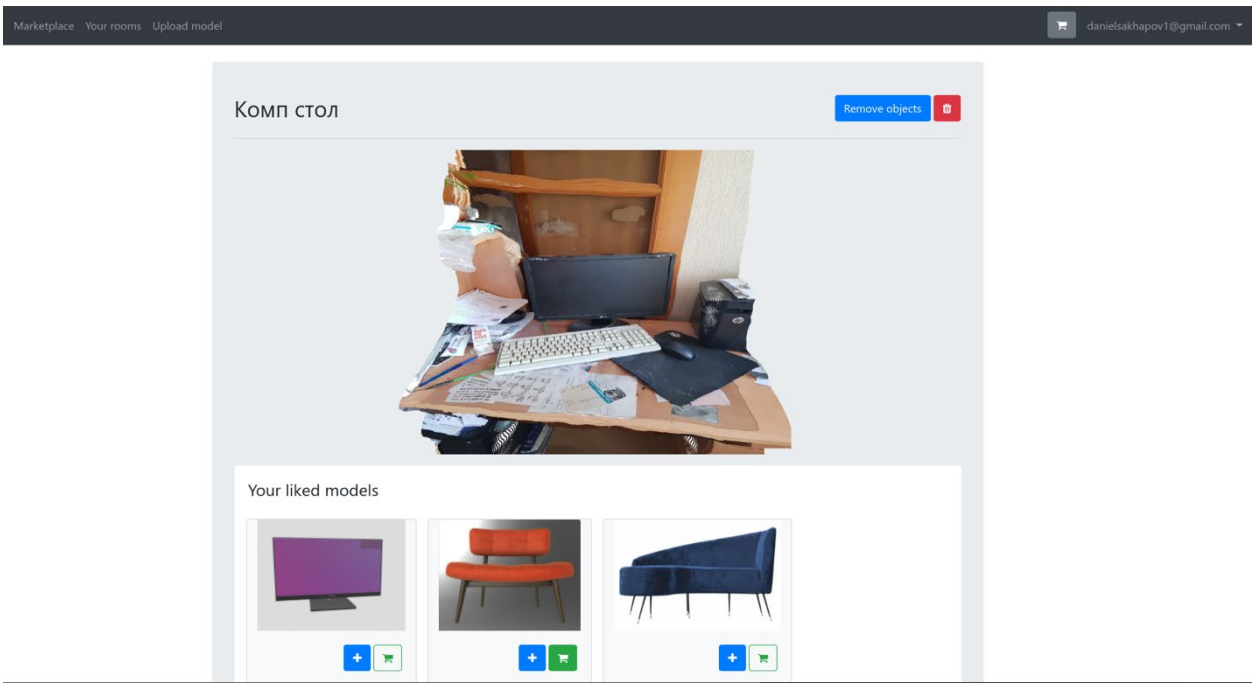



Figure 41 - The user's room in the form of a 3D model after background processing

The user paints over the old object in the photo, after which, in the background, the object is removed from the photo and a new model of the room is built without the object.

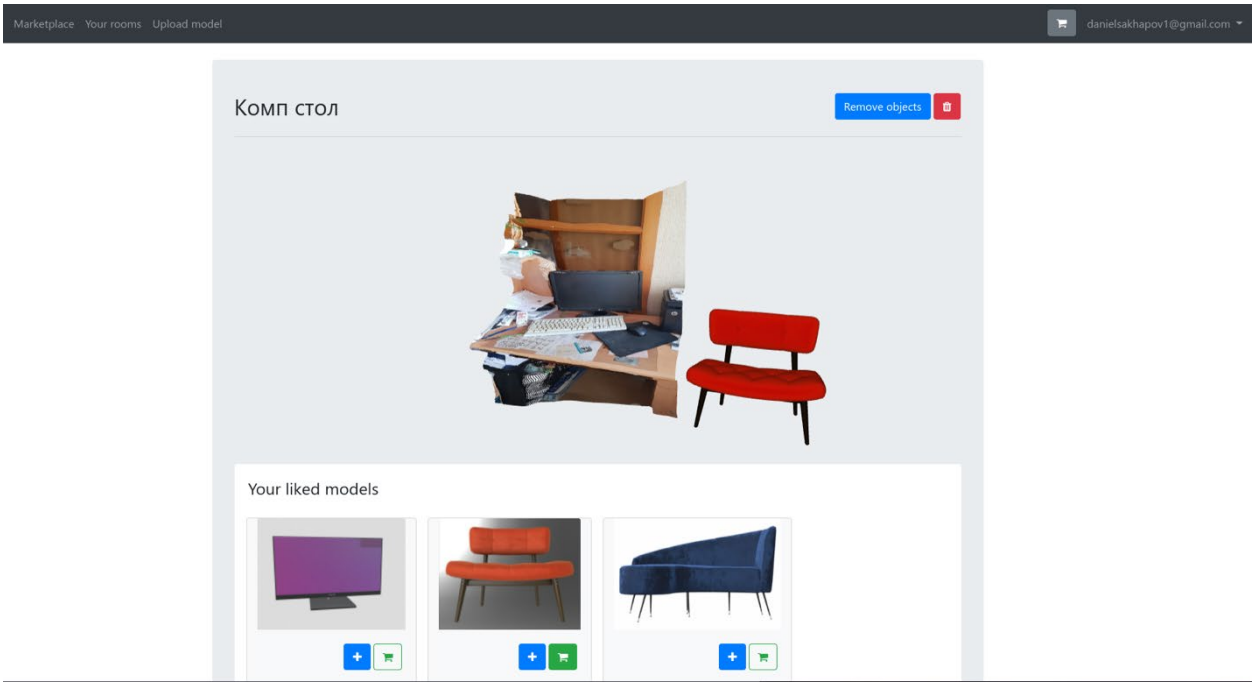



Figure 42 - The user adds a chair to the 3D model

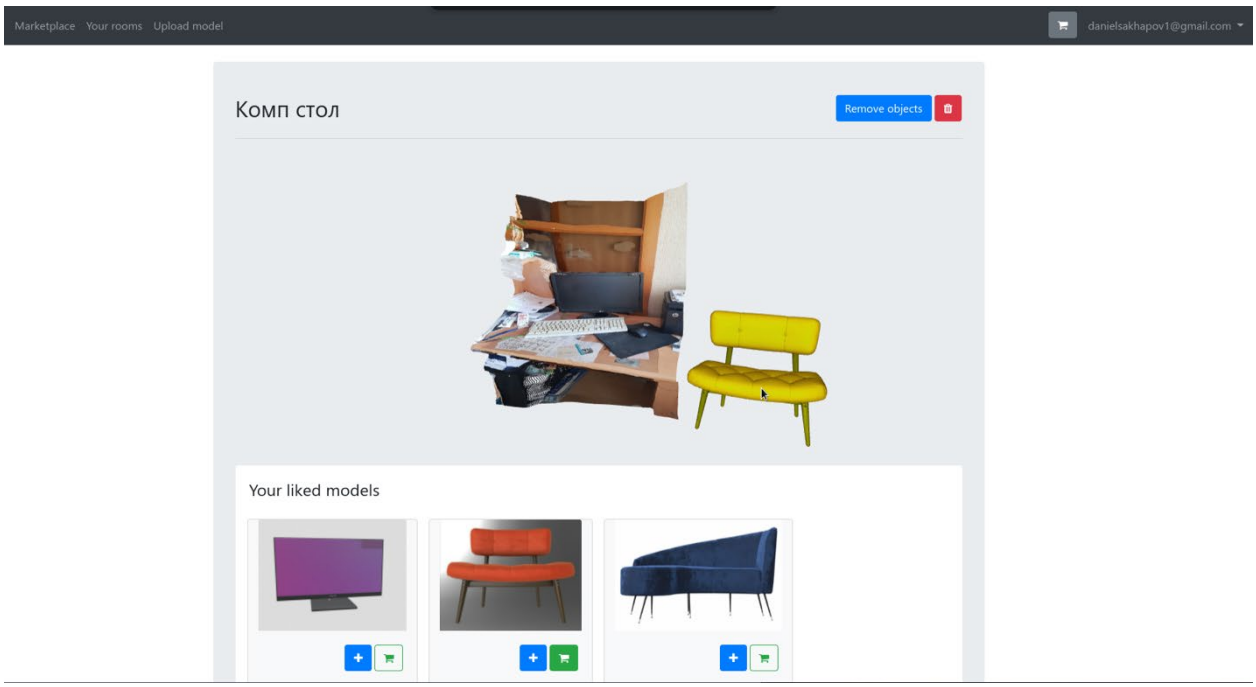



Figure 43 - The user selects a chair in the 3D model in order to move it

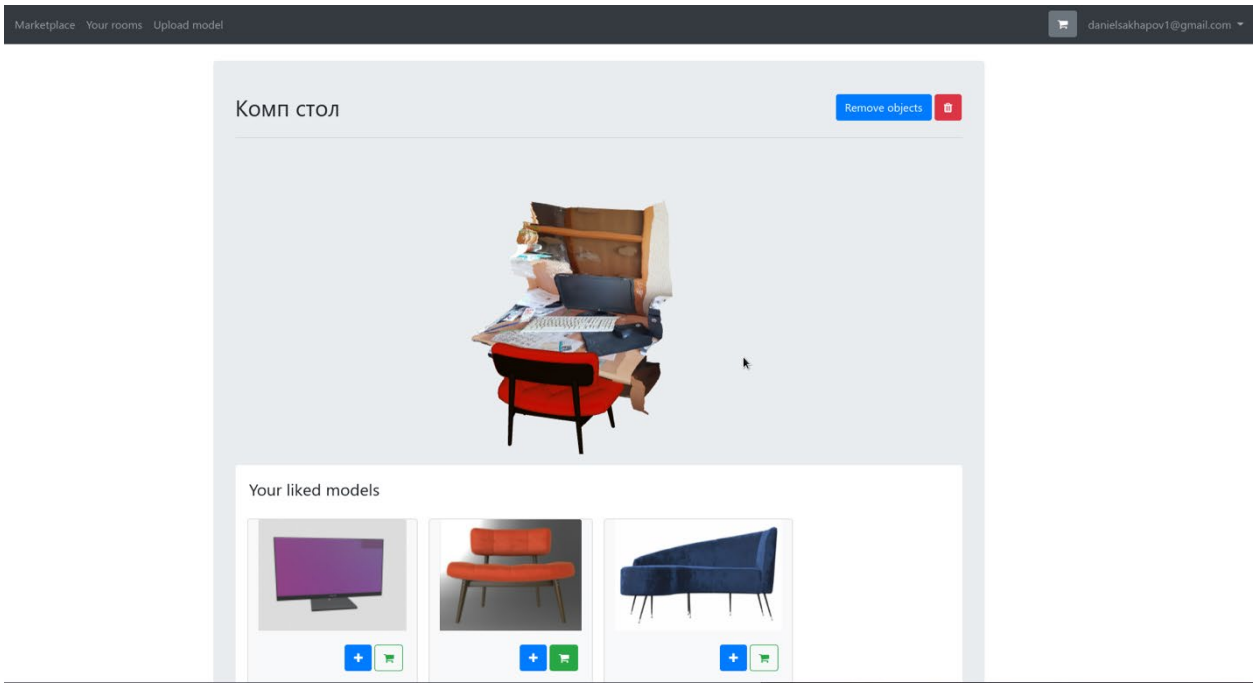



Figure 44 - The user moves the chair as he likes

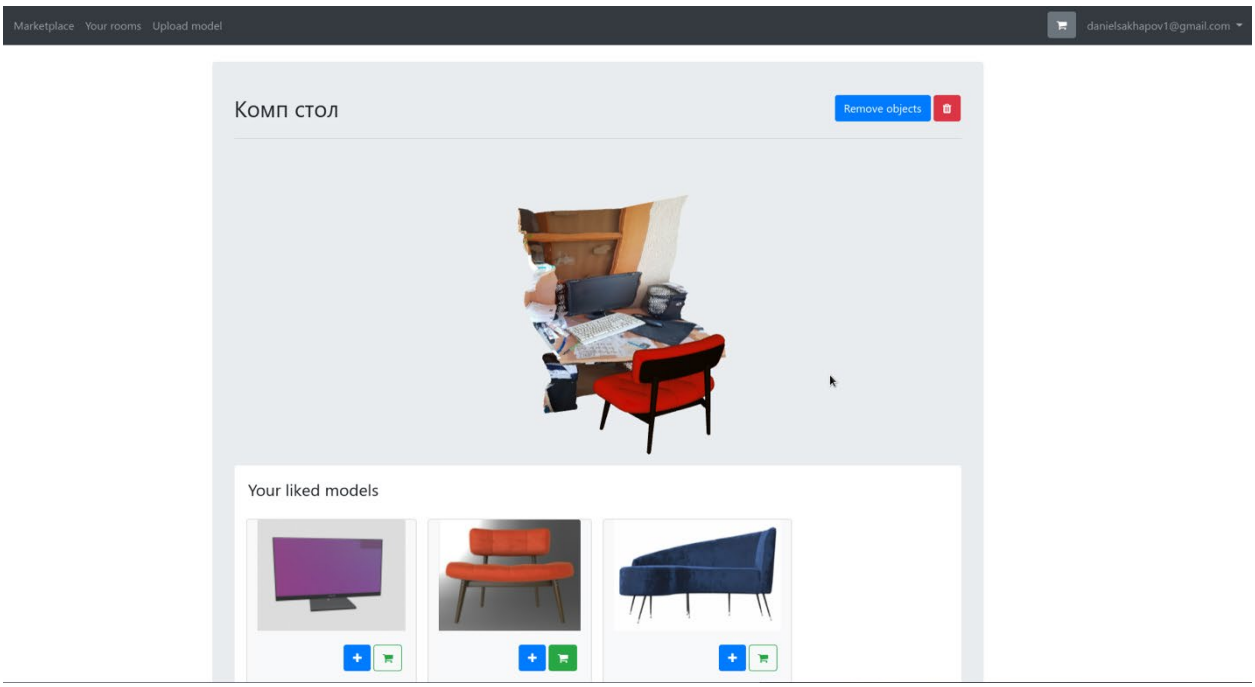



Figure 45 - Different angles

The user can decide what he wants, for example, to replace the monitor, in addition to installing a chair. To do this, he must first remove it.

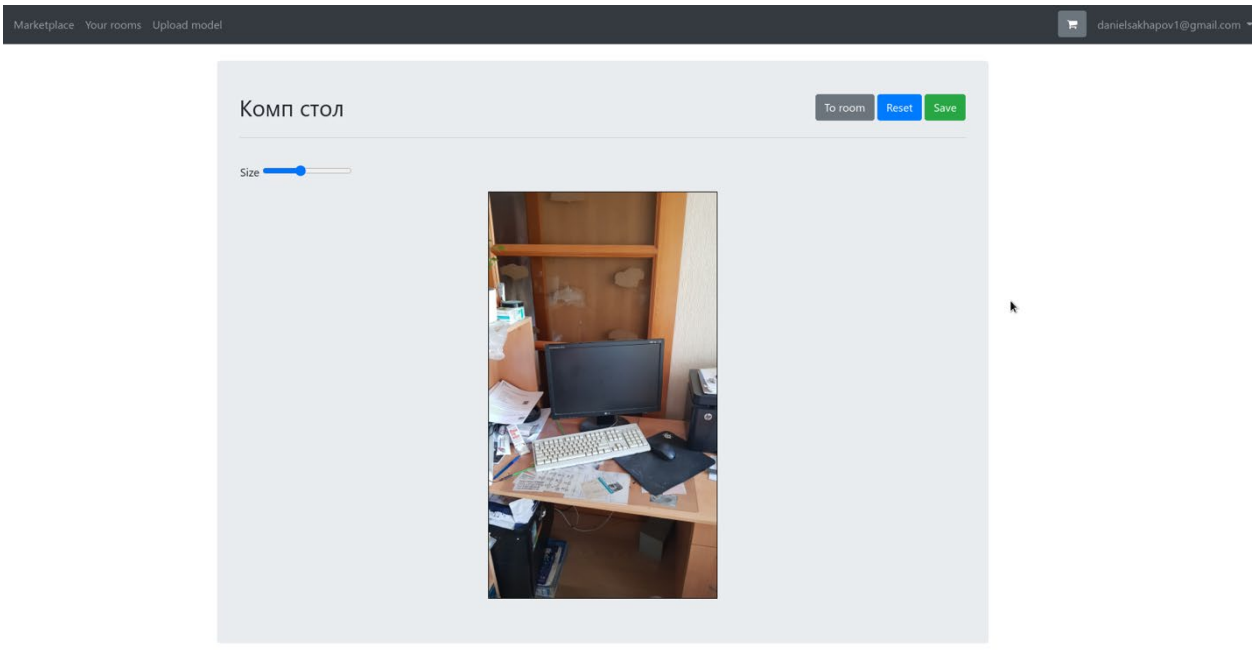



Figure 46 - Screen for inpainting objects

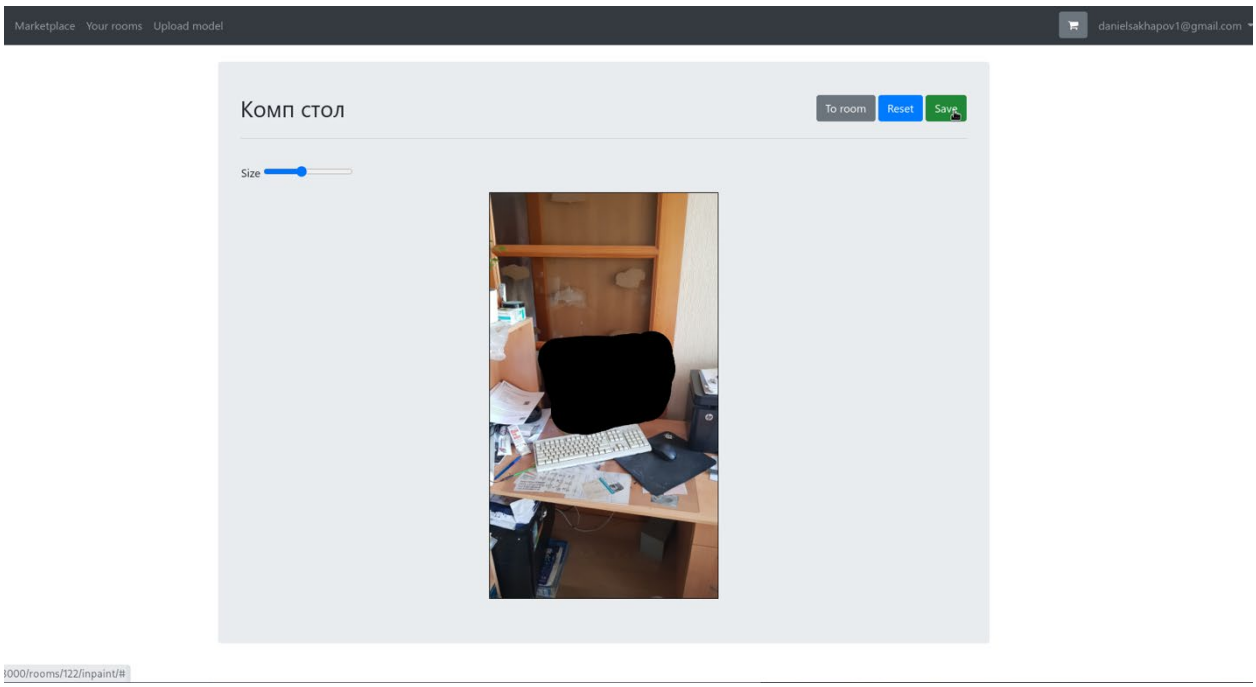



Figure 47 - The user removes the object from the photo

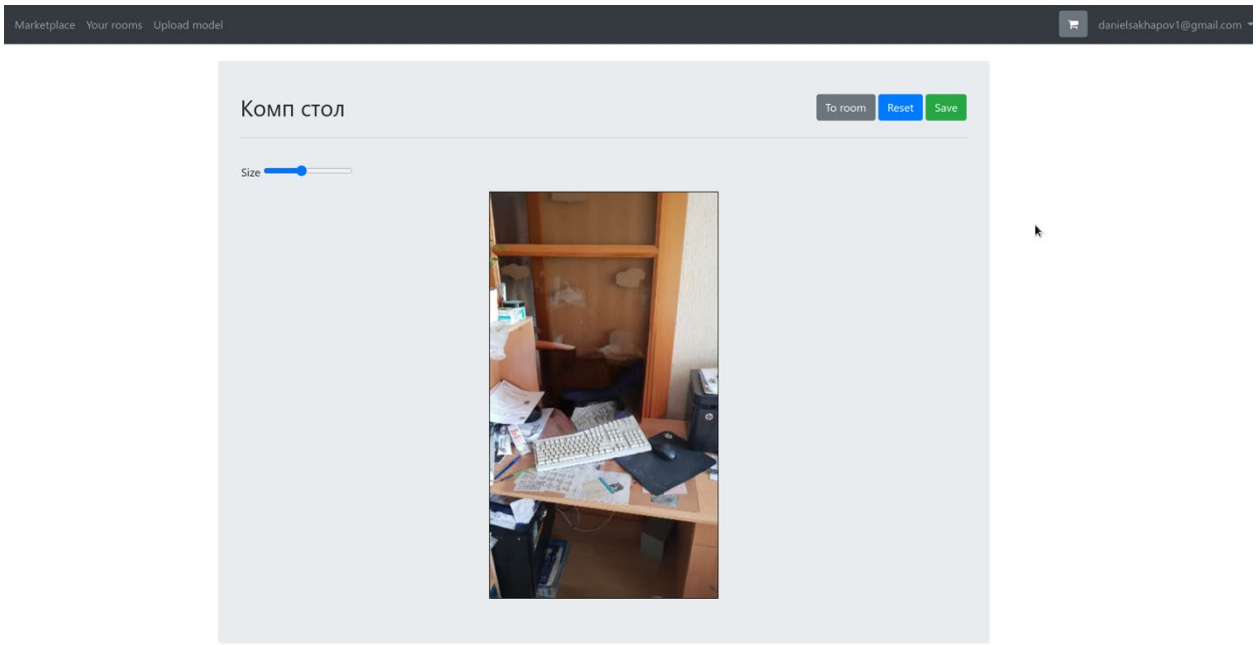



Figure 48 - Photo without the object

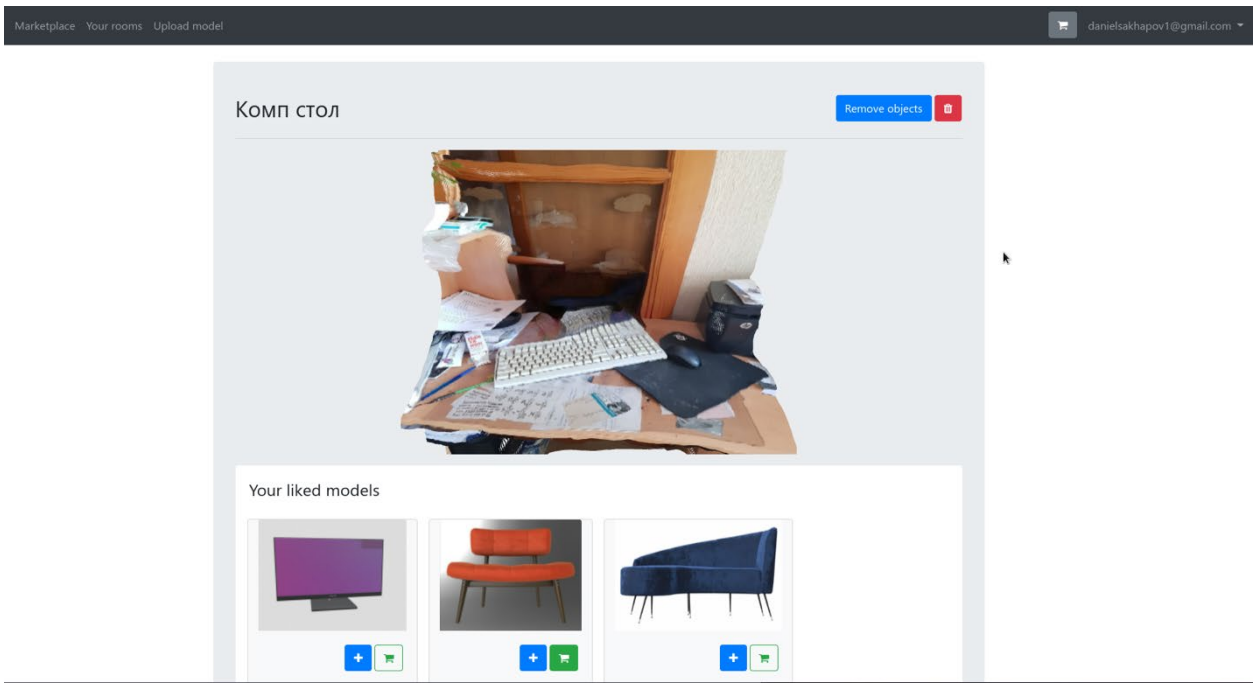



Figure 49 - Room model without the object

Now the user adds the 3D object they like to their room.

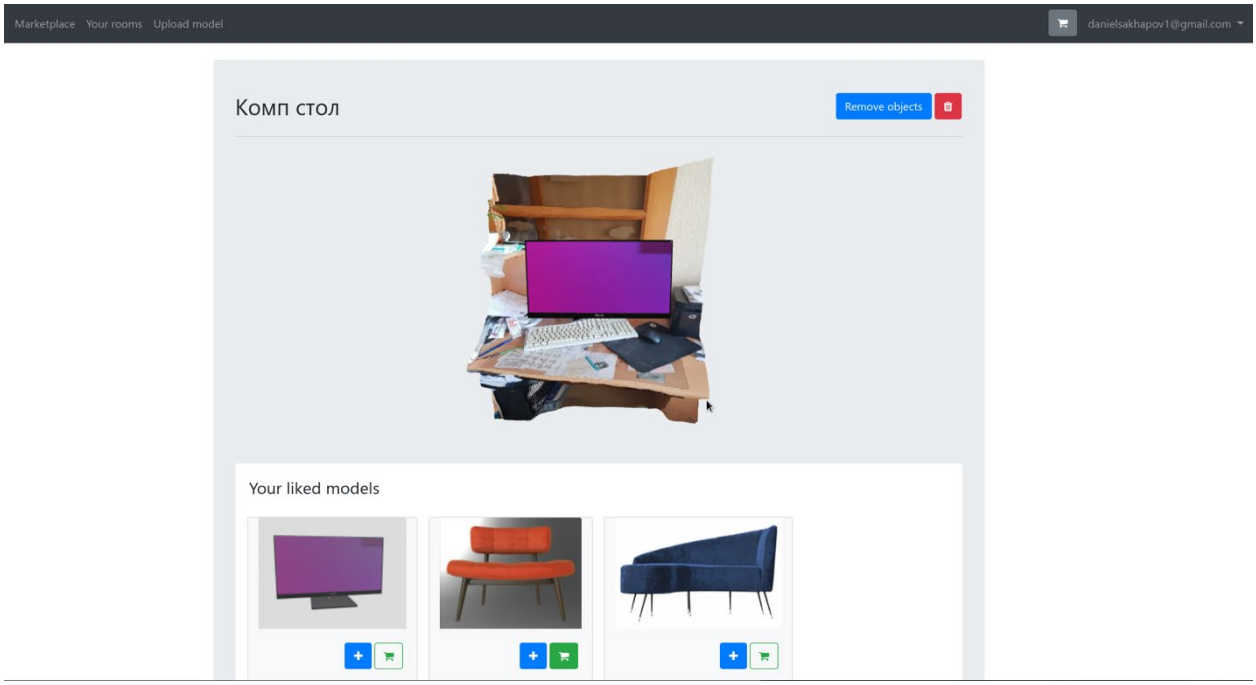



Figure 50 - The user has added a monitor to his room model

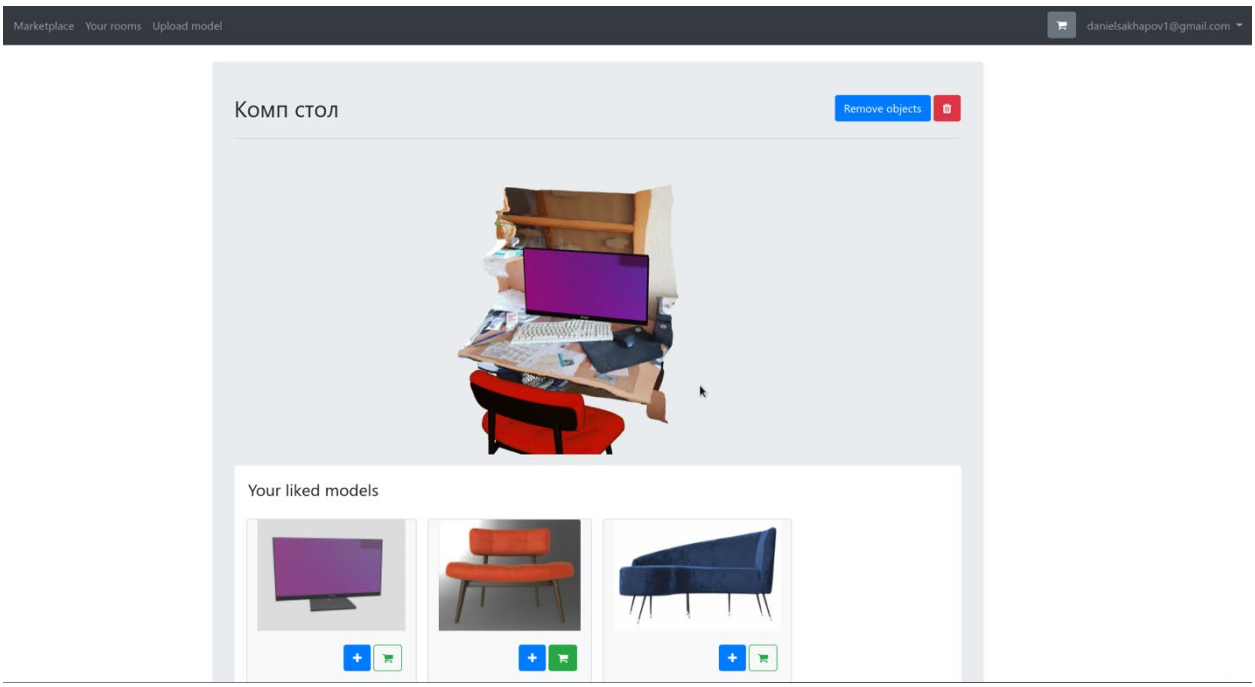



Figure 51 - The user also added a chair

Let's consider one more example with deleting an object and adding others.

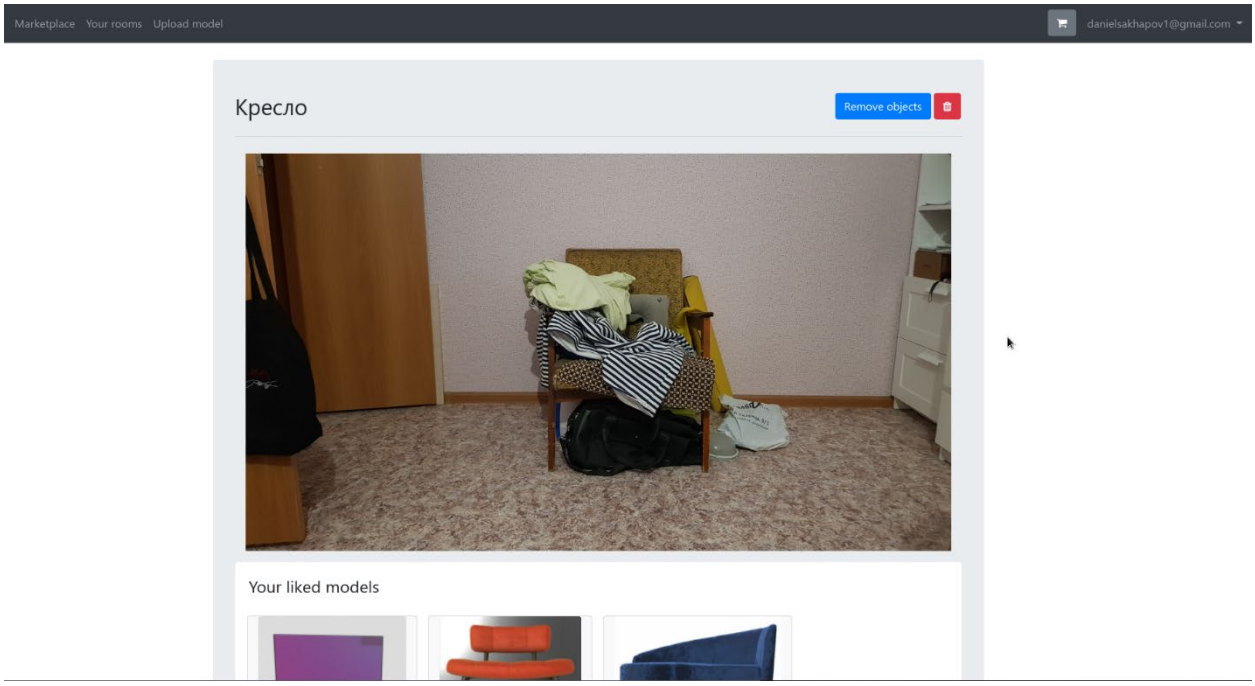



Figure 52 - Room with an armchair

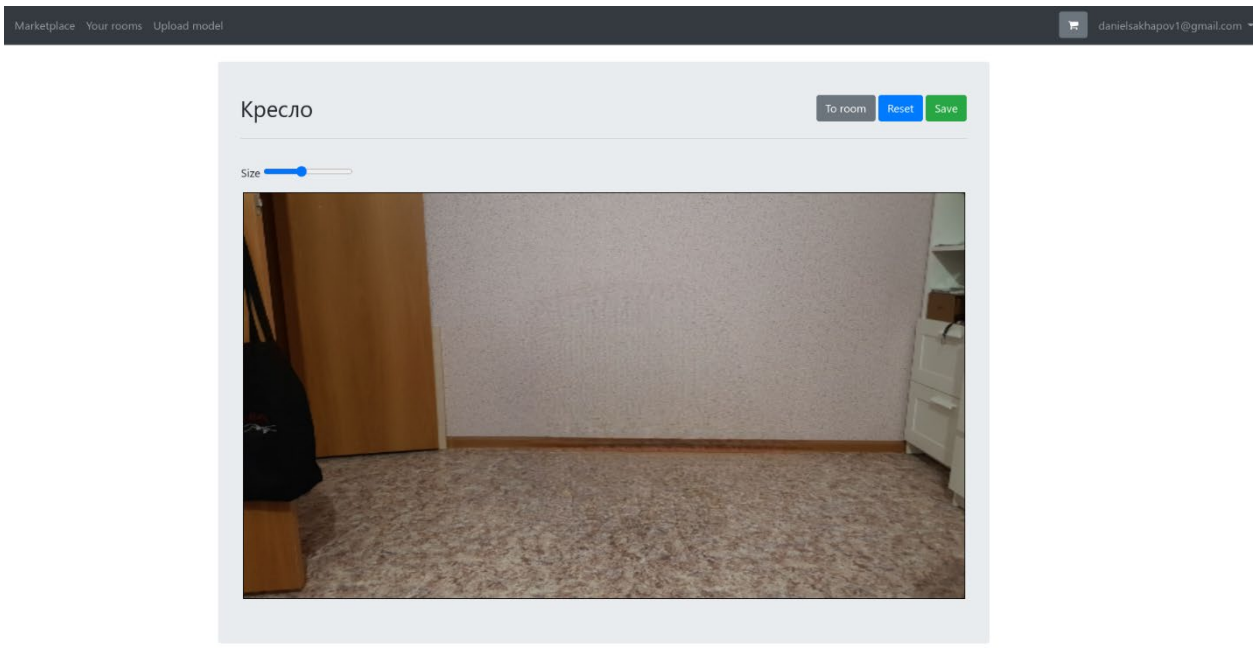



Figure 53 - Armchair removed by neural network

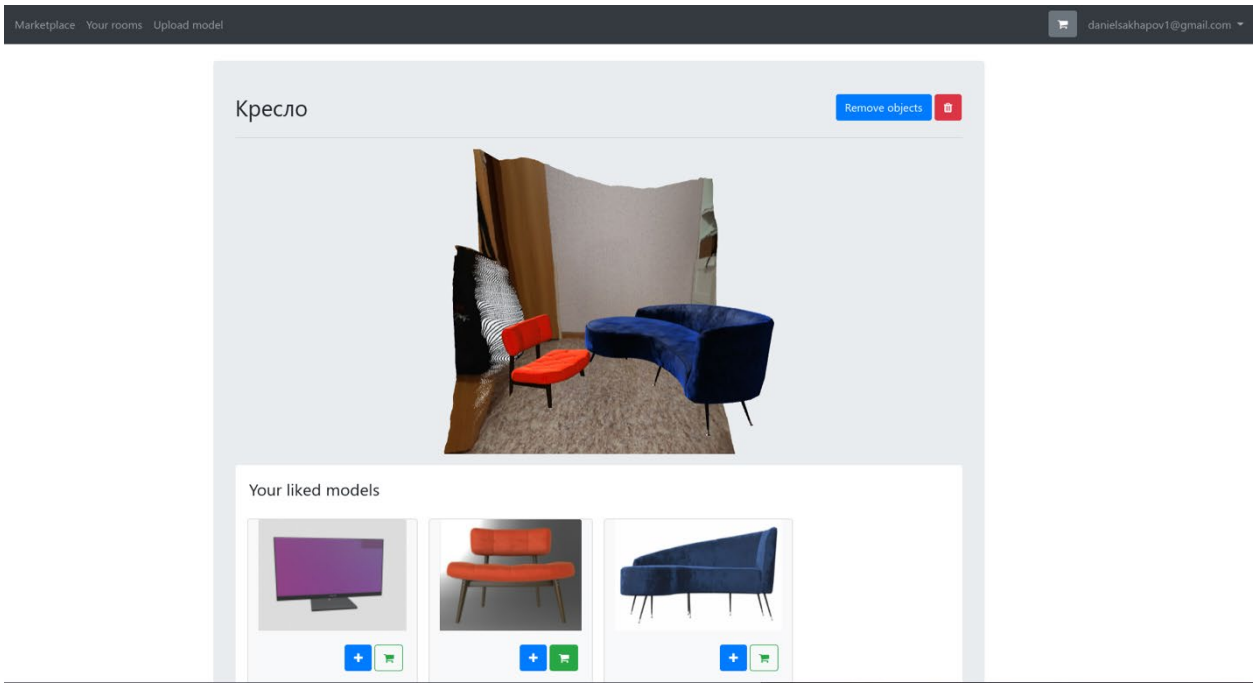



Figure 54 - Other 3D objects added to the room model

The developed application is also adaptive, which is shown in Figures 55-59. All the functionality of a full-size application is preserved.

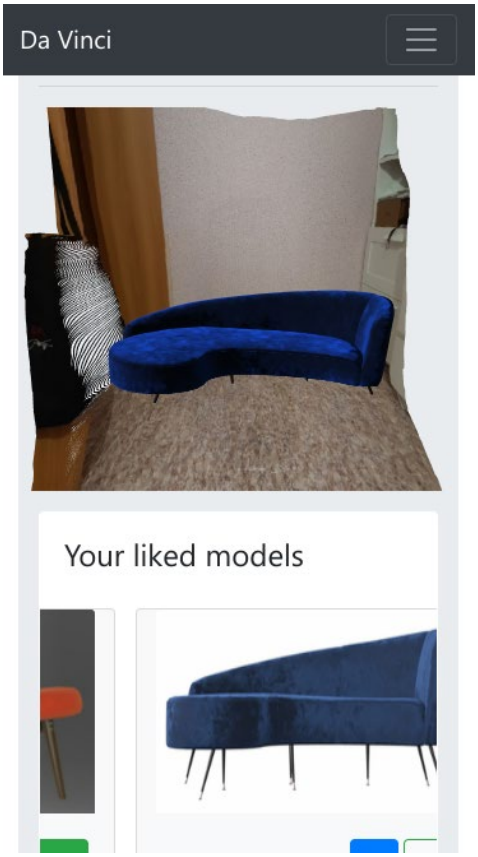



Figure 55 - Application adaptability example

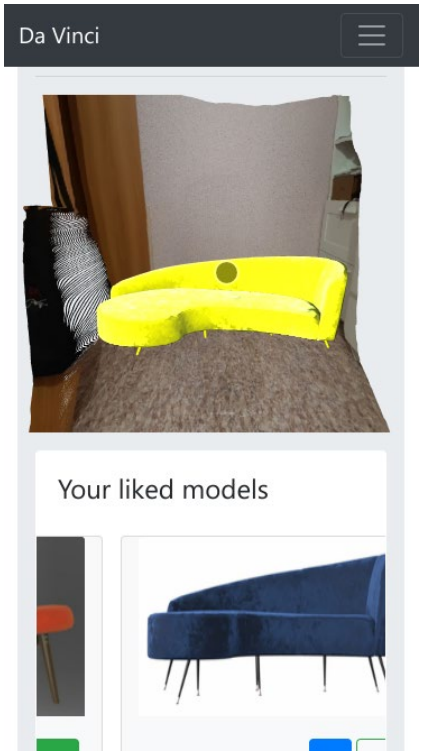



Figure 56 - Application adaptability example

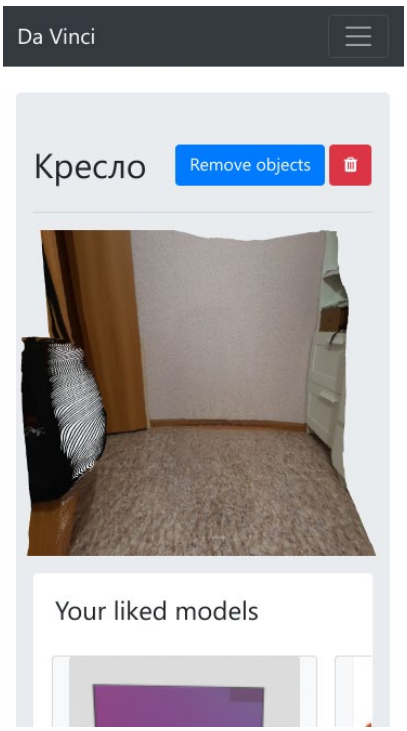



Figure 57 - Application adaptability example

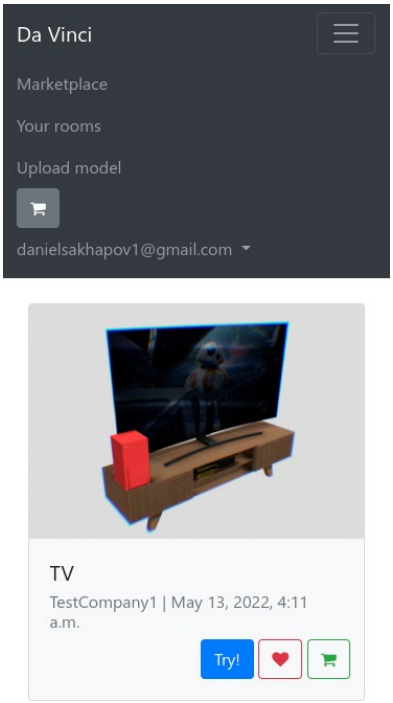



Figure 58 - Application adaptability example

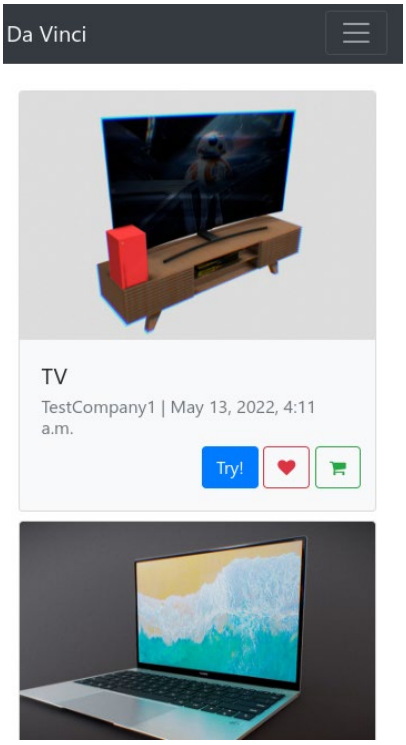



Figure 59 - Application adaptability example

# CONCLUSION

As a result of the work done, the goal was achieved - an application for virtual renovation was developed, and all tasks:

- requirements are identified and drafted;
- studied analogues;
- studied the theoretical foundations of neural networks;
- designed application, neural networks;
- application and neural networks developed;
- trained neural networks;
- app and neural networks tested.

The developed application is completely ready for public use.

# LITERATURE

1. Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European conference on computer vision (ECCV), pages 801–818, 2018.
2. Saumitro Dasgupta, Kuan Fang, Kevin Chen, and Silvio Savarese. Delay: Robust spatial layout estimation for cluttered indoor scenes. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 616–624, 2016.
3. Chen-Yu Lee, Vijay Badrinarayanan, Tomasz Malisiewicz, and Andrew Rabinovich. Roomnet: End-to-end room layout estimation. In Proceedings of the IEEE International Conference on Computer Vision, pages 4865–4874, 2017.
4. Giovanni Pintore, Marco Agus, and Enrico Gobbetti. Atlantanet: Inferring the 3d indoor layout from a single 360 degree image beyond the manhattan world assumption. In European Conference on Computer Vision, pages 432–448. Springer, 2020.
5. Jian Zhang, Chen Kan, Alexander G Schwing, and Raquel Urtasun. Estimating the 3d layout of indoor scenes and its clutter from depth sensors. In Proceedings of the IEEE International Conference on Computer Vision, pages 1273–1280, 2013.
6. Weidong Zhang, Wei Zhang, and Yinda Zhang. Geolayout: Geometry driven room layout estimation based on depth maps of planes. In European Conference on Computer Vision, pages 632–648. Springer, 2020.
7. Chuhang Zou, Alex Colburn, Qi Shan, and Derek Hoiem. Layoutnet: Reconstructing the 3d room layout from a single rgb image. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2051–2059, 2018.
8. Xiaohu Lu, Jian Yaoy, Haoang Li, Yahui Liu, and Xiaofeng Zhang. 2-line exhaustive searching for real-time vanishing point estimation in manhattan world. In 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 345–353. IEEE, 2017.
9. Arun Mallya and Svetlana Lazebnik. Learning informative edge maps for indoor scene layout prediction. In Proceedings of the IEEE international conference on computer vision, pages 936–944, 2015.
10. Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering the spatial layout of cluttered rooms. In 2009 IEEE 12th international conference on computer vision, pages 1849-1856. IEEE, 2009.

11. Yuqing Ma, Xianglong Liu, Shihao Bai, Lei Wang, Aishan Liu, Dacheng Tao, and Edwin Hancock. Region-wise generative adversarial imageinpainting for large missing areas. arXiv preprint arXiv:1909.12507, 2019.
12. Qi Mao, Hsin-Ying Lee, Hung-Yu Tseng, Siwei Ma, and Ming-Hsuan Yang. Mode seeking generative adversarial networks for diverse image synthesis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1429–1437, 2019.
13. Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? arXiv preprint arXiv:1801.04406, 2018.
14. Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. Edgeconnect: Generative image inpainting with adversarial edge learning. arXiv preprint arXiv:1901.00212, 2019.
15. Augustus Odena, Jacob Buckman, Catherine Olsson, Tom B Brown, Christopher Olah, Colin Raffel, and Ian Goodfellow. Is generator conditioning causally related to gan performance? arXiv preprint arXiv:1802.08768, 2018.
16. Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatiallyadaptive normalization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2337–2346, 2019.
17. Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2536–2544, 2016.
18. Pengda Qin, Weiran Xu, and William Yang Wang. Dsgan: Generative adversarial training for distant supervision relation extraction. arXiv preprint arXiv:1805.09929, 2018.
19. Jimmy SJ Ren, Li Xu, Qiong Yan, and Wenxiu Sun. Shepard convolutional neural networks. In Advances in Neural Information Processing Systems, pp. 901–909, 2015.
20. Yurui Ren, Xiaoming Yu, Ruonan Zhang, Thomas H Li, Shan Liu, and Ge Li. Structureflow: Image inpainting via structure-aware appearance flow. In Proceedings of the IEEE International Conference on Computer
21. Vision, pp. 181–190, 2019. Mark Sabini and Gili Rusak. Painting outside the box: Image outpainting with gans. arXiv preprint arXiv:1808.08483, 2018.
22. Min-cheol Sagong, Yong-goo Shin, Seung-wook Kim, Seung Park, and Sungjea Ko. Pepsi: Fast image inpainting with parallel decoding network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 11360–11368, 2019.

23. Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 9157–9166, 2019.
24. Alejo Concha, Muhammad Wajahat Hussain, Luis Montano, and Javier Civera. Manhattan and piecewise-planar constraints for dense monocular mapping. In Robotics: Science and systems, 2014.
25. David R Cox. The regression analysis of binary sequences. Journal of the Royal Statistical Society: Series B (Methodological), 20(2):215–232, 1958.
26. Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5828–5839, 2017.
27. Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun.ACM, 24(6):381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL https://doi.org/10.1145/358669.358692.
28. Alex Flint, David Murray, and Ian Reid. Manhattan scene understanding using monocular, stereo, and 3d features. In 2011 International Conference on Computer Vision, pages 2228–2235. IEEE, 2011.
29. Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2(11):559–572, 1901. doi: 10.1080/14786440109462720.
30. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
31. Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.
32. Lam Huynh, Phong Nguyen-Ha, Jiri Matas, Esa Rahtu, and Janne Heikkilä. Guiding monocular depth estimation using depth-attention volume. In European Conference on Computer Vision, pages 581–597. Springer, 2020.
33. Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703, 2019.
34. Xiaojuan Qi, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia. Geonet: Geometric neural network for joint depth and surface normal estimation. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 283–291, 2018.
35. Xiaojuan Qi, Zhengzhe Liu, Renjie Liao, Philip HS Torr, Raquel Urtasun, and Jiaya Jia. Geonet++: Iterative geometric neural network with edge-aware refinement for joint depth and surface normal estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020.
36. Yiming Qian and Yasutaka Furukawa. Learning pairwise inter-plane relations for piecewise planar reconstruction. In European Conference on Computer Vision, pages 330–345. Springer, 2020.
37. Michael Ramamonjisoa and Vincent Lepetit. Sharpnet: Fast and accurate recovery of occluding contours in monocular depth estimation. The IEEE International Conference on Computer Vision (ICCV) Workshops, 2019.
38. Jason Rambach, Paul Lesur, Alain Pagani, and Didier Stricker. Slamcraft: Dense planar rgb monocular slam. In 2019 16th International Conference on Machine Vision Applications (MVA), pages 1–6. IEEE, 2019.
39. Pierluigi Zama Ramirez, Matteo Poggi, Fabio Tosi, Stefano Mattoccia, and Luigi Di Stefano. Geometry meets semantics for semi-supervised monocular depth estimation. In Asian Conference on Computer Vision, pages 298–313. Springer, 2018.
40. Akshay Rangesh and Mohan Manubhai Trivedi. Ground plane polling for 6dof pose estimation of objects on the road. IEEE Transactions on Intelligent Vehicles, 5(3): 449–460, 2020.
41. Fangwen Shu, Yaxu Xie, Jason Rambach, Alain Pagani, and Didier Stricker. Visual slam with graph-cut optimized multi-plane reconstruction. arXiv preprint arXiv:2108.04281, 2021.
42. Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In International Conference on Machine Learning, pages 9120–9132. PMLR, 2020.
43. Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. Demon: Depth and motion network for learning monocular stereo. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5038–5047, 2017.
44. C. O. Ancuti, C. Ancuti, R. Timofte, L. Van Gool, L. Zhang, and M.-H. Yang. Ntire 2019 image dehazing challenge report. In CVPR Workshops, June 2019.
45. A. Arnab, S. Jayasumana, S. Zheng, and P. H. S. Torr. Higher order conditional random fields in deep neural networks. In ECCV, pages 524–540, 2016.

46. V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell., 39(12):2481–2495, 2017.
47. T. Baltrusaitis, P. Robinson, and L. Morency. Constrained local neural fields for robust facial landmark detection in the wild. In ICCVW, pages 354–361, 2013.
48. P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar. Localizing parts of faces using a consensus of exemplars. IEEE Trans. Pattern Anal. Mach. Intell., 35(12):2930–2940, 2013.
49. G. Bertasius, C. Feichtenhofer, D. Tran, J. Shi, and L. Torresani. Learning temporal pose estimation from sparsely-labeled videos. CoRR, abs/1906.04016, 2019.
50. A. Bulat and G. Tzimiropoulos. Human pose estimation via convolutional part heatmap regression. In ECCV, pages 717–732, 2016.
51. A. Bulat and G. Tzimiropoulos. Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources. In ICCV, pages 3726–3734, 2017.
52. A. Bulat and G. Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230, 000 3d facial landmarks). In ICCV, pages 1021–1030, 2017.
53. X. P. Burgos-Artizzu, P. Perona, and P. Dollar. Robust face landmark estimation under occlusion. In ICCV, pages 1513–1520, 2013

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

# Отчет о проверке на заимствования №1

**Автор:** Сахапов Даниил Альбертович
**Проверяющий:** Сахапов Даниил

Отчет предоставлен сервисом «Антиплагиат» - http://users.antiplagiat.ru

## ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 3
Начало загрузки: 03.06.2022 08:48:22
Длительность загрузки: 00:00:02
Имя исходного файла: master.pdf
Название документа: РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ВИРТУАЛЬНОГО РЕМОНТА
Размер текста: 60 кБ
Тип документа: Выпускная квалификационная работа
Символов в тексте: 61729
Слов в тексте: 7647
Число предложений: 519

## ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 03.06.2022 08:48:24
Длительность проверки: 00:00:02
Корректировка от 03.06.2022 08:55:42
Комментарии: не указано
Модули поиска: Интернет Free

| ЗАИМСТВОВАНИЯ | САМОЦИТИРОВАНИЯ | ЦИТИРОВАНИЯ | ОРИГИНАЛЬНОСТЬ |
|---|---|---|---|
| 8,13% | 0% | 0% | 91,87% |

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

| № | Доля в отчете | Источник | Актуален на | Модуль поиска |
|---|---|---|---|---|
| [01] | 2,56% | Thinking Outside the Box: Generation of Unconstrained 3D Room Layouts http://arxiv.org | 09 Мая 2022 | Интернет Free |
| [02] | 1,86% | Shallow2Deep: Indoor Scene Modeling by Single Image Understanding http://arxiv.org | 09 Мая 2022 | Интернет Free |
| [03] | 0,68% | HorizonNet: Learning Room Layout with 1D Representation and Pano Stretch Data Augmentation http://arxiv.org | 09 Мая 2022 | Интернет Free |

Еще источников: 7
Еще заимствований: 3,04%

С результатом ознакомлен _____ Бабанов А.М.