UDC 681.327.12.001.362
DOI https://doi.org/10.32782/IT/2024-1-8

Borys MOROZ
Doctor of Technical Sciences, Professor of Department of Software Engineering, Dnipro University of Technology, 19, Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49000, moroz.boris.1948@gmail.com
ORCID: *0000-0002-5625-0864*
Scopus Author ID: *57202222055*

Leonid KABAK
PhD, Associate Professor of Department of Software Engineering, Dnipro University of Technology, 19, Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49000, kabak.leo@gmail.com
ORCID: *0000-0001-6267-1772*
Scopus Author ID: *5720222205*

Dmytro MOROZ
PhD, University Lecturer of Department of Software Engineering, Dnipro University of Technology, 19, Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49000, dmitriy@moroz.cc
ORCID: *0000-0003-2577-3352*
Scopus Author ID: *57369936300*

Yevhenii RUKSOV
Master, Dnipro University of Technology, 19, Dmytra Yavornytskoho ave., Dnipro, Ukraine, 49000, euruksov@gmail.com

To cite this article: Moroz, B., Kabak, L., Moroz, D., Ruksov, E. (2024). Metody predstavlennia 3D-obiektiv dlia navchannia heneratyvnykh neironnykh merezh [Methods of representation of 3D objects for learning generative neural networks]. *Information Technology: Computer Science, Software Engineering and Cyber Security,* 1, 61–72, doi: https://doi.org/10.32782/IT/2024-1-8

METHODS OF REPRESENTATION OF 3D OBJECTS FOR LEARNING GENERATIVE NEURAL NETWORKS

The paper considered various methods of three-dimensional objects generating by using of neural networks. Several key elements of the methodologies of this type of synthesis of new information were singled out, on the basis of which a new way of three-dimensional objects representing was proposed for its use for typical generative models of neural networks training.

The purpose of the work is *to develop a method of representing of three-dimensional objects that would satisfy the criterion of high density of information useful for a generative model. The minimization of the redundancy of the information generated together with the minimization of the losses associated with the process of transition from a three-dimensional way of representing of an object to a two-dimensional one (with which the existing generative models can cope quite well) are the key aspects of the proposed method of representation.*

The methodology *for solving of the problem given consists in building of a mathematical model of a new type of representation of three-dimensional objects; development of a software algorithm that implements a mathematical model; and testing this representation based on a typical generative neural network model.*

The scientific novelty *is that for the first time such a type of representation of a three-dimensional object was proposed, which could be used for typical generative models training.*

Conclusions. *The proposed method of representing of a three-dimensional object showed its viability even in the context of training of a small typical generative model DCGAN. Prospects for further research of the proposed method for training of other typical generative models were also determined, because this method could be quite easily adapted to representations of input and output data of a wide range of neural network architectures.*

Key words: *three-dimensional objects, DCGAN models, generative neural network models, generative competitive networks, variational autoencoders.*

Борис МОРОЗ
доктор технічних наук, професор кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Яворницького, 19, м. Дніпро, Україна, 49005
ORCID: *0000-0002-5625-0864*
Scopus Author ID: *57218242332*

Леонід КАБАК
кандидат технічних наук, доцент, доцент кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Яворницького, 19, м. Дніпро, Україна, 49005
ORCID: *0000-0001-6267-1772*
Scopus Author ID: *5720222205*

Дмитро МОРОЗ
доктор філософії, викладач програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Яворницького, 19, м. Дніпро, Україна, 49005
ORCID: *0000-0003-2577-3352*
Scopus Author ID: *57369936300*

Євгеній РУКСОВ
студент кафедри програмного забезпечення комп'ютерних систем, Національний технічний університет «Дніпровська політехніка», просп. Яворницького, 19, м. Дніпро, Україна, 49005

Бібліографічний опис статті: Мороз, Б., Кабак, Л., Мороз, Д., Руксов, Є. (2024). Методи представлення 3D-об'єктів для навчання генеративних нейронних мереж. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 1, 61–72, doi: https://doi.org/10.32782/IT/2024-1-8

МЕТОДИ ПРЕДСТАВЛЕННЯ 3D-ОБ'ЄКТІВ ДЛЯ НАВЧАННЯ ГЕНЕРАТИВНИХ НЕЙРОННИХ МЕРЕЖ

У роботі були розглянуті різні методи генерування тривимірних об'єктів за допомогою нейронних мережі. Було виокремлено ряд ключових елементи методологій такого типу синтезу нової інформації, на основі чого була запропонований новий спосіб представлення тривимірних об'єктів для його застосування в навчанні типових генеративних моделей нейронних мереж.

Метою роботи є *розробка такого методу представлення тривимірних об'єктів, який задовольняв би критерію високої щільності корисної для генеративної моделі інформації. Мінімізація надлишковості генерованої інформації разом з мінімізацією втрат, пов'язаних з процесом переходу з тривимірного способу представлення об'єкта в двовимірний (з яким існуючі генеративні моделі здатні впоратись доволі якісно), є ключовими аспектами запропонованого методу представлення.*

Методологія *вирішення поставленої задачі полягає в побудові математичної моделі нового типу представлення тривимірних об'єктів; розробці програмного алгоритму, який реалізує математичну модель; та тестуванні цього представлення на базі типової генеративної моделі нейронної мережі.*

Наукова новизна *полягає в тому, що вперше був запропонований такий тип представлення тривимірного об'єкта, який можливо використовувати для навчання типових генеративних моделей.*

Висновки. *Запропонований метод представлення тривимірного об'єкта показав свою життєздатність навіть в контексті навчання маленької типової генеративної моделі DCGAN. Також були визначені перспективи подальших досліджень запропонованого методу для навчання інших типових генеративних моделей, адже цей метод може бути досить легко адаптований до представлень вхідних та вихідних даних великого спектру архітектур нейронних мереж.*

Ключові слова: *тривимірні об'єкти, моделі DCGAN, генеративної моделі нейронної мережі, генеративні змагальні мережі, варіаційні автокодувальники.*

Urgency of the problem. In recent years, generative models based on deep learning have attracted increasing interest due to significant advancements in this field. Leveraging vast amounts of data, well-designed network architectures, and intelligent training methods, deep generative models have demonstrated incredible ability to create highly realistic content across various domains, such as images, texts, and sounds. Among these deep generative models, two main groups stand out and deserve special attention: Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs).

Analyses of Recent Research and Publications. The Generative Adversarial Network (GAN), developed by Ian Goodfellow and colleagues in 2014 (Goodfellow, 2014), represents a machine learning method where two networks – the generator and discriminator – engage in a «zero-sum game». The generator aims to replicate the distribution of the training data, while the discriminator assesses whether an input sample is real or synthesized. Training the generator model involves maximizing the probability of the discriminator misclassifying samples. These networks are multi-layer perceptrons trained using the standard backpropagation method. An important aspect of such networks is also normalizing input data to effectively handle large images, particularly through convolutional neural networks (CNNs).

The Variational Autoencoder (VAE) is a type of autoencoder that regulates the distribution of encodings during training to preserve properties of the latent space that facilitate the generation of new data. This approach is based on the principles of variational inference, allowing VAE to effectively reduce the dimensionality of data through encoding and decoding processes. A variational autoencoder learns to encode input as a distribution in the latent space rather than a single point, aiding in preserving the structure of data during the generation of new samples. Training a VAE involves minimizing a loss function consisting of reconstruction and regularization terms, with the latter ensuring the organization of the latent space using the Kullback-Leibler divergence (Description of the working principle of the VAE model, 2019).

Methods of representing 3D objects

The next important aspect of this work is the construction of three-dimensional (3D) objects. The field that deals with creating objects of this type of graphics is called 3D modelling. There are several methodologies for creating three-dimensional models:

1. Polygon Mesh – a set of vertices, edges, and faces that form surfaces of an object in 3D space. The Polygon Mesh is one of the most common methods for representing 3D models. It utilizes polygons (typically triangles or quadrilaterals) to approximate the surfaces of the object. This method allows for representing various details of an object but can be demanding in terms of the number of polygons, especially for complex models.
2. Solid Modelling – method that uses mathematical primitives (such as cubes, spheres, cones, etc.) to create 3D objects is known as procedural modelling.
3. NURBS (Non-Uniform Rational B-Splines) – a mathematical method for representing curves and surfaces in 3D space.
4. Point Clouds – collections of 3D coordinates that represent the surface of an object or environment. They are used in scanning real-world objects to create accurate 3D models.
5. Voxel Grids – three-dimensional counterparts to pixels in two-dimensional space. Voxel Grids divide 3D space into evenly spaced volumetric elements, known as voxels.
6. Spline Surfaces – a method that uses mathematical splines (patches or pieces) to create surfaces.
7. Constructive Solid Geometry – a method that uses addition, subtraction, and intersection operations to create complex 3D objects by combining simple geometric shapes such as cubes, cylinders, and cones.

To accomplish the set task, Polygon Mesh method was chosen. There are several types of Polygon Mesh, including Triangle Mesh, Quadrilateral Mesh, Polyhedron Mesh, Quad-Dominant Mesh, Mixed-Topology Mesh, Delaunay Triangulation, Regular Grid. In the proposed representation method, the triangular version will be used.

Also, there are several forms of storing information about vertices, edges, and surfaces, among the most common ones:

1. Vertex List – this method is quite simple, but it does not provide fast access to neighboring vertices.
2. Edge List – this method provides detailed information about the mesh topology but consumes a lot of memory and is inefficient for processing.
3. Face List – this method is typically used for storing surfaces in triangle meshes, but its efficiency is questionable.
4. Adjacency List – this method allows for quickly finding neighbors for each vertex, which is useful for certain operations.
5. Adjacency Matrix – large matrices can be memory-inefficient but provide fast access to neighbours.
6. Half-Edge Structure – often used to optimize mesh operations but requires more memory for storage.
7. Prism Structure – this structure is based on dividing the mesh into layers, where each layer has vertices and edges. It is useful for representing objects with internal structure.

8. Edge Mesh – in this structure, edges are stored as separate objects, and faces are represented by pointers to edges. It can be useful for handling different types of faces without the need for conversion (A general description of the existing methods of presenting three-dimensional models, 2012).

It is also important to note the main file formats for 3D models:

1. STL (Stereolithography) is a format that is approximated by large triangles and is used for exchanging finished 3D models. It is very simple and suitable for 3D printing but does not contain any information about textures or the colour of the model.

2. OBJ (Wavefront Object) is a text format that can contain geometric data, textures, materials, and normals. It is supported by many 3D modelling and visualization programs and is widely used in computer graphics.

3. FBX (Filmbox) is an open format developed by Autodesk that supports the exchange of information between various programs for 3D modelling and animation. It can contain geometric data, textures, animations, and other attributes.

4. Collada is an open XML format for exchanging information between programs for 3D modelling, animation, and rendering. It supports geometric data, textures, materials, cameras, lights, and animation.

5. 3DS (3D Studio Max) is a format developed for the 3D Studio Max software. It is supported by many programs and can contain geometric data, textures, materials, and animation. However, this format has limited support for modern features.

6. PLY (Polygon File Format) is a format used for storing triangle meshes. It has a simple structure and can contain textures, colours, and normals.

7. DXF (Drawing Exchange Format) is a text format developed by Autodesk, aimed at exchanging geometric data. It is widely used in CAD programs to exchange data between programs from different manufacturers.

8. IGES (Initial Graphics Exchange Specification) is a standard format for exchanging geometric data between different CAD and CAM systems. It is not limited to triangles and can contain many types of geometric objects (Description of the most popular file formats for 3D objects, 2022).

9. USD (Universal Scene Description) is an open and high-performance format for representing and exchanging 3D scenes and animations. It is developed by Pixar and widely used in renowned film studios and in the visualization industry. USD supports various types of geometric and animated objects, materials, lights, and other scene attributes. It allows for efficient management of complex scenes and assets, providing high performance during data exchange between programs and workflows (Official documentation for Universal Scene Description, 2021).

The USD format was specifically chosen for this research, because it has a convenient textual structure that is easily processed using the Java programming language.

The application of generative models for creating 3D objects

The Variational Autoencoder (VAE) is highly valued for its elegance, theoretical appeal, and ease of implementation. It is considered one of the leading methods in generative modelling, producing excellent results. However, a limitation of VAEs trained on images is that the generated samples may be blurry, and the precise reasons for this phenomenon remain unknown.

Nevertheless, the structure of VAE can be extended to various types of architectures, making it suitable for building a wide range of probabilistic models. One notable advantage of VAE is that training both the encoder and decoder simultaneously encourages the model to learn a coherent coordinate system that the encoder can capture, making it effective for diverse learning tasks.

The fundamental principle of Generative Adversarial Networks (GANs) is to focus on models that are not restricted solely by maximum likelihood, thus eliminating destructive divergences between different models. In GAN training, it is critically important to strike a balance between the power of the generator and the discriminator. In cases where the discriminator proves to be extremely efficient, the generator may face difficulties in detecting gradients. Conversely, an advantage for the generator may lead to the detection of weak points in the discriminator, resulting in false recognitions. Adjusting the learning rates of both networks can address these issues.

Although GANs were initially proposed for unsupervised learning, they have found wide applications in semi-supervised learning, supervised learning, and reinforcement learning. A plethora of GAN variations has been developed, including DCGAN, SRGAN, VAE-GAN, WGAN, CycleGAN, and StyleGAN, each characterized by its unique features and applications.

These two technologies also have applications for generating three-dimensional models, not just two-dimensional images.

Specifically, in the article dedicated to the 3D-GAN model (Jiajun Wu, 2017), the possibility of applying a Generative Adversarial Network for creating three-dimensional objects is explored for the first time. This approach exhibits a sufficiently high quality of generated objects. Essentially, it is a classic DCGAN model (Goodfellow, 2014), as described in the previous section, but with an additional dimension in the image to represent the three-dimensional shape (without colour) of the object. The authors of the article apply all the classic tools and methodologies to create such a generative model. Additionally, the authors provide a description and test results of extending their 3D-VAE-GAN model, which allows generating its three-dimensional version based on a two-dimensional image of an object. Here, a combination of VAE and GAN technology is applied on the scale of 3D objects.

Overall, the article provides a fairly broad overview of a new type of generative models. It has served as the foundation for further research in this field. However, the 3D-GAN model has several drawbacks:

1. A large volume of information is generated and processed through the folding-unfolding process, much of which is redundant, as this model operates in voxel space (in other words, 3D pixels).

2. The focus on filling a certain volume of space rather than the shape of the object. In other words, the model generates volume, not shape. This results in a loss of the integrity of the form in some generated samples.

In another article dedicated to the ShapeVAE model (Nash, 2017), a somewhat different approach to generating three-dimensional objects is described. The central element of this model is the VAE technology. Unlike the previously discussed work, this model generates a so-called «point cloud» at the output instead of a voxel space. The authors provide a fairly detailed description of the proposed model's operation, and the results demonstrate a quite decent quality of the generated objects. The authors also make assumptions about the future improvement of the model using GAN (but without specifics).

However, the ShapeVAE model also has some drawbacks:

1. The VAE technology itself has the drawback of blurriness in the generated image, and in this model, the authors did not manage to avoid this issue, as they also note in their article.

2. Generating a «point cloud» leads to an excess of output data, although it is somewhat less than in voxel space.

The described models are early applications of VAE and GAN technologies for generating three-dimensional objects. However, further works have proposed more refined versions of these approaches. Additionally, some studies have suggested alternative representations of the output data, improving parameters such as data redundancy, processing speed, learning efficiency, and the quality of generated 3D object samples. Let's explore some of these advancements.

There is a model called AtlasNet (Groueix, 2018), which applies an interesting approach to generating a mesh based on a set of two-dimensional patches that are assembled into a three-dimensional object according to certain rules. In this model, one can clearly see the authors' inspiration from UV representations of surfaces, commonly used to shape the texture topology in various formats for representing three-dimensional objects. The authors themselves acknowledge this in their paper: «Inspired by the formal definition of a surface as a topological space locally reminiscent of a Euclidean plane, we aim to locally approximate the target surface by mapping a set of squares onto the surface of the 3D shape» (Groueix, 2018). This approach is quite similar to atlases in geography.

This model has many advantages compared to the previous two. Here, it operates directly on the shape surface rather than the filled volume, significantly improving the quality of object generation. However, this model has one drawback – the use of somewhat outdated and less efficient neural network topology, specifically a multi-layer perceptron (MLP). This significantly affects the training speed of such a model and carries the risk of overfitting if hyperparameters are not properly tuned.

Another model is the so-called «Multi-Resolution Tree-like Network» (MRTNet) (Gadelha, 2018), capable of generating «point clouds». This model is based on VAE, so it is quite similar to the ShapeVAE model. It can be considered as an improved and optimized version of ShapeVAE, operating on a regular point list. Overall, the drawbacks of this model are similar to ShapeVAE but with less significant manifestations.

Regarding the representation of «point clouds», there is another notable model – PointFlow (Guandao Yang, 2019). This model proposes representing the «point cloud» as a probability distribution of values along three coordinates. The underlying network topology is still based on VAE, but the principle of generation (and accordingly the model training algorithm) of a three-dimensional object differs somewhat from ShapeVAE and MRTNet. Here, the authors applied a series of complex algebraic and probabilistic manipulations with the data, improving the quality and accuracy of generation and mitigating the negative effect of VAE technology. However, some other GAN-based models still show better results on certain training data sets.

Overall, there are many different approaches to generating «point clouds». In a meta-study (Achlioptas, 2018) that compares various existing methods based on different models and technologies, the authors concluded that the best technology for generating three-dimensional objects is the Gaussian Mixture Model (GMM), which was to some extent applied in the PointFlow model. In general, generative models based on the representation of «point clouds» have been thoroughly investigated and have a wide representation of various ideas.

The development of research in the field of generative models for representing three-dimensional objects in the form of a mesh has not been as rapid as the advancements in technologies related to «point clouds». However, there are some developments in this area, such as the PolyGen model (Nash, 2020). It consists of two components: an unconditional vertex generator and a conditional surface generator. The model is based on recurrent neural network architectures.

Advantages of the PolyGen model include:

1. Generation of mesh, providing a reasonably acceptable level of correspondence between the model and the shape of the 3D object.

2. The ability to simultaneously generate different types of polygons in the mesh.

Disadvantages of this model include:

1. Complex architecture with recurrent connections, leading to difficulty in implementation and deployment of the model.

2. Complex training algorithm for the model.

Among all of the models considered, this model is the closest to the proposed idea of representing a 3D object for use in generative models.

In general, the complexity of working with a mesh is primarily related to the high level of heterogeneity of the generated elements, complicating the construction of an efficient way to represent mesh data without loss.

Let's consider another more exotic way of representing 3D objects, using the example of Occupancy Networks (Mescheder, 2019). This is a novel approach to representing a 3D object, involving the direct mapping of the object in space as a function of multiple arguments that a neural network must approximate. This function is called the occupancy function. In such a model, there is no discrete division of space (as in voxel space), or a discrete set of points (as in a «point cloud»), or a discrete set of surface elements (as in mesh); instead, the authors of the Occupancy Networks propose a continuous representation of the surface. Moreover, they ensure that such a representation method does not significantly impact memory load and algorithm speed.

In general, this method of representing a 3D object has broad prospects, as the main advantage is the relatively high quality of surface approximation. Although the authors of the PolyGen model (Nash, 2020), who compared their model with Occupancy Networks, note a decrease in the quality of such a model in representing smooth surfaces (such as a table).

The purpose of the article is to develop a method for representing 3D objects that satisfies the criterion of high density of information useful for generative models. Minimizing the excess of generated information along with minimizing the losses associated with the transition from a 3D representation of the object to a 2D one (with which existing generative models can cope quite qualitatively) are key aspects of the proposed POLY-IMAGE method.

Presentation of the main material.

DESCRIPTION OF THE POLY-IMAGE METHOD

Justification of the chosen methods

Based on the developments in the field of generating three-dimensional objects, this task can be divided into several important aspects:

1. Generative Model Architecture.

2. Representation Method of Three-dimensional Object.

Various architectures have been considered for this task, including GAN (and its variations), VAE (and its variations), GMM, and various recurrent architectures. Each approach has its own advantages and disadvantages and can be applied with varying levels of efficiency for different representation methods. These two aspects are undoubtedly closely related. In the field of neural network architecture, there is already a sufficient foundation of research and solutions. Therefore, this work will not propose something entirely new in terms of model architecture.

Instead, in terms of the representation method, there are many different directions that are still not sufficiently explored. Most techniques are based on memory consumption efficiency, which is the main criterion for performance quality. From the perspective of surface approximation quality, each method has its application depending on the topology of the surface itself. In general, the triangular mesh is considered the most optimal method, taking into account both memory aspects and the quality of approximating various types of surfaces.

Moreover, the mesh representation format in USD files is quite convenient for computations. This format consists of simple arrays of numbers that have a specific relationship with each other (this will be described in more detail in the next section). Another advantage of this format is that it is simply a text file, easily readable from any programming language. Generating such a file based on only a certain part of it (such as a sequence of point indices) is a relatively trivial task for a regular deterministic linear algorithm.

To verify the application of the proposed mesh representation method for the training model, a DCGAN model will be used to generate two-dimensional images, similar to what was applied in my previous work dedicated to generating paintings (Ruksov, 2022).

In general, the proposed mesh representation method can be used with other generative models. In theory, recurrent neural networks might be the best architecture for such representation, as they allow generating and recognizing data of any dimension. This work will consider a limited version of the proposed mesh representation, but it can be expanded and unified.

To implement the set task, the programming languages Java and Julia were chosen. The reasons for this choice are detailed in the first section.

Mathematical model

The main feature of the proposed model lies in a new way of representing data input for the discriminator and output for the generator. Most existing models (as detailed in the previous section) suggest overly redundant representations of generated data. Specifically, voxel space, point clouds, vector representations of meshes, occupancy functions, and others all require neural networks to process unnecessary information, potentially complicating the organization of efficient training for large models of this type.

Additionally, such representations do not focus directly on the shape of three-dimensional objects. To draw an analogy on a smaller scale and with a simple type of model, consider a basic classifier for two-dimensional images. After training, a neural network can recognize certain shapes and colors of objects in two-dimensional space, even though the object may have different sizes, positions in space, may be modified, and so on. Thus, the model finds a specific pattern of object characteristics in two-dimensional space, realizing the ability to classify images. Extrapolating this representation to higher-dimensional data, the shape of the three-dimensional object's surface is crucial for effective classification. The internal content (what is beneath the surface of the object) obviously does not carry any useful information for the neural network. Moreover, the surrounding space is also redundant.

In the previous section, several arguments were presented in favor of using mesh as a sufficiently effective way to approximate the surface. It's worth adding that mesh itself has redundant data, which can be processed and transformed into the final «consumer» state using conventional algorithms and classical mathematical transformations.

The method of simplifying the representation of the mesh was invented based on the structure of the file format for 3D scenes USD. In this format, the representation of the mesh consists of four main elements:

1. Arbitrary (unsorted) set of points with coordinates in 3D space, which are the vertices of the mesh.
2. Array of dimensions for each mesh patch.
3. Array of sequential numbers (indices) of vertex points of the mesh (has a clear order of values in correspondence with the previous two elements).
4. Array of points corresponding to the normals of the mesh (not considered in this work).

From such a simple way of describing a mesh, you can create the following model for representing the shape of a three-dimensional object. The first important element of the model is the discrete space, bounded by values for all three coordinates in the range [-1; 1]. Let's denote this space as an indexed set:

$$P = \{A_i\}_{i \in I}, \tag{1}$$

where I is the set of indices for which $I \subseteq N$;

A is the indexed discrete set of points.

For the levels of discretization within a single cube, it can be arbitrary, and it determines the quality and accuracy of the generated object.

Let's specify the essence of the surface f within the indexed space:

$$f = \{(a, b, c) : a, b, c \in I\}, \tag{2}$$

where $f \in F$, F – is the set of all possible surfaces.

In scope of this work, a triangular mesh is considered, so the surface in it is an ordered set of three indices.

Next, let's specify the essence of the mesh within the proposed model:

$$m = F_m, \tag{3}$$

where $F_m \subset F$ – is a subset of surfaces.

Now, based on this model of mesh representation, we can build a scheme for mapping this representation to a two-dimensional image, which will be generated by the neural network.

Two-dimensional images are matrices of pixels, each of which represents an ordered set of three brightness values in the RGB scheme. Each pixel can be considered as a point in three-dimensional space. In this case, the entire image can be represented as a point cloud. However, with such a representation, some information about the arrangement of each pixel in the image matrix is lost. To address this issue, the point cloud needs to be indexed, and it will have two indices:

$$U = \{X_{r,k}\}_{r \in R, k \in K}, \quad (4)$$

where X – is point cloud;

R – is a set of indices per row of the image matrix;

K – is a set of indexes on the column of the matrix.

Now, based on such a representation of a two-dimensional image, we can replace the standard RGB scheme for pixels with a model representing mesh surfaces, as indicated in expression (2). Thus, the point cloud X will be a set of tuples f, so that f ∈ X, and X = F.

To determine the sets of indices R and K, additional attributes for each surface need to be defined.

Firstly, it is necessary to determine the average point (or center of mass Of) for each surface. As known, the center of mass of a triangle is the point of intersection of all three medians of the triangle. To calculate this, the following formulas are applied:

$$O_f(x_f, y_f, z_f) = \begin{cases} x_f = \dfrac{x_a + x_b + x_c}{3} \\ y_f = \dfrac{y_a + y_b + y_c}{3}, \\ z_f = \dfrac{z_a + z_b + z_c}{3} \end{cases} \quad (5)$$

where x_f, y_f, z_f – are the coordinates of the centre of mass of the surface f.

The next step is the transition from Cartesian coordinates to spherical coordinates, where a point is represented as a tuple (ρ, φ, θ), where: ρ is the radius; φ is the angle between the positive x-axis and the projection of the segment drawn from the pole to the point Of onto the XY plane; θ is the angle between the positive z-axis and the segment drawn from the pole to the point P. To perform this transition, the following formulas are applied:

$$O_f(\rho, \varphi, \theta) = \begin{cases} \rho = \sqrt{x_f^2 + y_f^2 + z_f^2} \\ \varphi = \arctan\dfrac{\sqrt{x_f^2 + y_f^2}}{z_f}, \\ \theta = atan2(y_f, x_f) \end{cases} \quad (6)$$

where atan2 is a special function for determining the angle θ (a standard function in many programming languages; the details of its operation will not be discussed in this work).

The next step is to determine the values of r and k for the surface f. To do this, we take the angular values of the spherical coordinate of the center of mass Of of the surface, as these values are uniform in value and form a sphere (i.e., a globe) that can be conventionally unfolded and represented as a two-dimensional matrix, and the radius ρ has a depth value (or in terms of the RGBA colour scheme – transparency). Therefore, the radius value will be added as another attribute to the formula (2). Thus, we obtain the following expression for the model of a single surface:

$$f_{r,k} = \{(a,b,c,d) : a,b,c \in I; d = \rho\}_{r=\varphi, k=\theta}, \quad (7)$$

where d – is a fourth value that is equivalent to the radius value.

Thus, we have obtained a method for representing the surface of a three-dimensional object as a two-dimensional image with four-component pixel tuples.

It only remains to determine the order of vertices a, b, c in the tuple of the surface f. For a clearly deterministic determination, we will apply the method of numbering vertices by the distance between the centre of mass point and the vertex, in increasing order of this value. The neural network can potentially generate any order of these vertices, but this is not a problem for the reverse transformation into a mesh. To implement this method, we will apply the classic formula for the distance between points:

$$op_j = \sqrt{(x_j - x_f)^2 + (y_j - y_f)^2 + (z_j - z_f)^2}, \quad (8)$$

where j ∈ {1,2,3} – is the sequence number of the vertex.

Next step requires jsut sort the obtained values in ascending order and number the resulting array. The vertices corresponding to these distances will be applied to order the values a, b, c for the formula (7). There is no sense in providing a mathematical description of these simple operations, as they are purely algorithmic operations that can be easily implemented in any programming language, and they are not related to the analytical description of the proposed representation model.

Thus, we have obtained a complete mathematical description of the proposed model. The schematic representation of the bidirectional converter of PolyImage representations will be presented next.

Program Implementation Description

For greater clarity in understanding the operation of the developed PolyImage converter (also known as MeshImage), a diagram is provided, as shown in fig. 1.

The role of the triangle distance ρ from the centre is not depicted in this diagram, as described in the previous section. In fact, this parameter is not used in the training dataset since it has no real impact on the generated data. Only for the visual representation of the converter's results as an image can this parameter be enabled.

As mentioned earlier, to implement the mathematical model of the PolyImage converter, the Java 17 programming language was used. The converter is a console application with two functions (the functionality can be easily extended as the code follows SOLID principles):

1. Generating a training dataset based on a single instance of a three-dimensional object: This object, which needs to be trained, should be in.usda format. The result will be a set of.bson files, which are compressed representations of PixelImage (as shown in diagram 1). These files do not store zero pixels and will be filled automatically by the Julia algorithm (described below).

2. Reverse conversion from a set of.bson files to.usda files of three-dimensional objects: This function is needed to verify the generated objects since the model generates only PixelImage representations.

Finally, a class diagram (abbreviated) of this converter is presented in Fig. 2.

It's cleary can be seen that some classes participate in interpreting the USD format, and some in converting UsdFile to PixelImage, and another part performs the function of generating instances for the training dataset. The algorithm for generating instances is based on the operation of rotating points in three-dimensional space around the unit vectors x, y, z (the mathematical component of this operation will not be described in detail as these are well-known linear operations).

Next, let's present a class call diagram, which will provide additional insight into the mechanism. This diagram is shown in Fig. 3.

It seems there was an issue, and the code snippet or further description about the bidirectional conversion from the compressed representation PixelImage to numerical tensors used in the PolyImageGen model training algorithm is missing. Here is the relevant code snippet:

```
function parse_meshImg(bson::Dict)
    filledPixels = [bson[:pixels][i] for i
in 1:size(bson[:pixels], 1)]
    meshImg = zeros(Float32, bson[:width],
bson[:height], 3)
    for pixel in filledPixels
        meshImg[pixel[:x]+1, pixel[:y]+1, 1]
= pixel[:a]
        meshImg[pixel[:x]+1, pixel[:y]+1, 2]
= pixel[:b]
        meshImg[pixel[:x]+1, pixel[:y]+1, 3]
= pixel[:c]
    end
    return meshImg
end
function serialize_img(img)
    pixels = vec([Dict(:a=>Float64(
img[x,y,1]), :b=>Float64(img[x,y,2]),
:c=>Float64(img[x,y,3])) for x in
1:size(img,1), y in 1:size(img,2)])
    pixels = filter(x -> x[:a]!=0 ||
x[:b]!=0 || x[:c]!=0 , pixels)
    return Dict(:pixels => pixels, :width
=> size(img,1), :height => size(img,2))
end
for i in 1:size(a,4)
    s_img = serialize_img(a[:, :, :, i])
    BSON.bson("bsons/" * string(i) *
".bson", s_img)
End
```

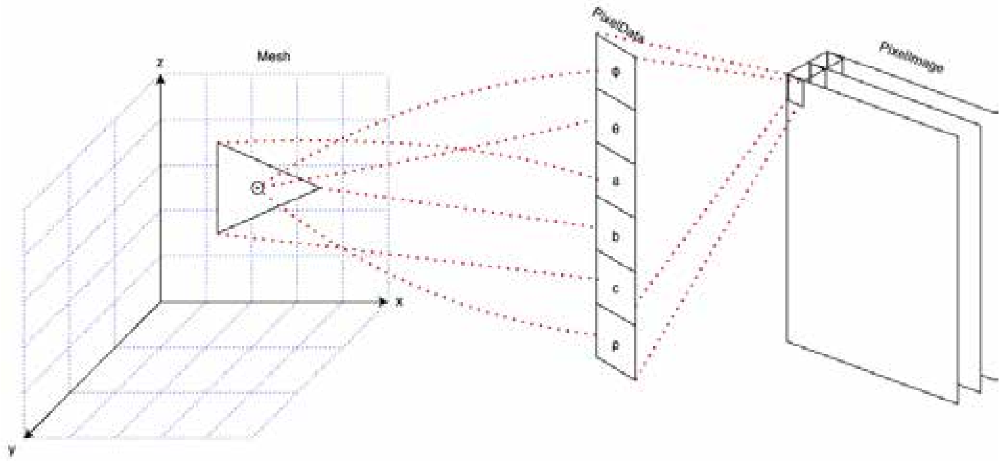

Fig. 1. PolyImage converter scheme

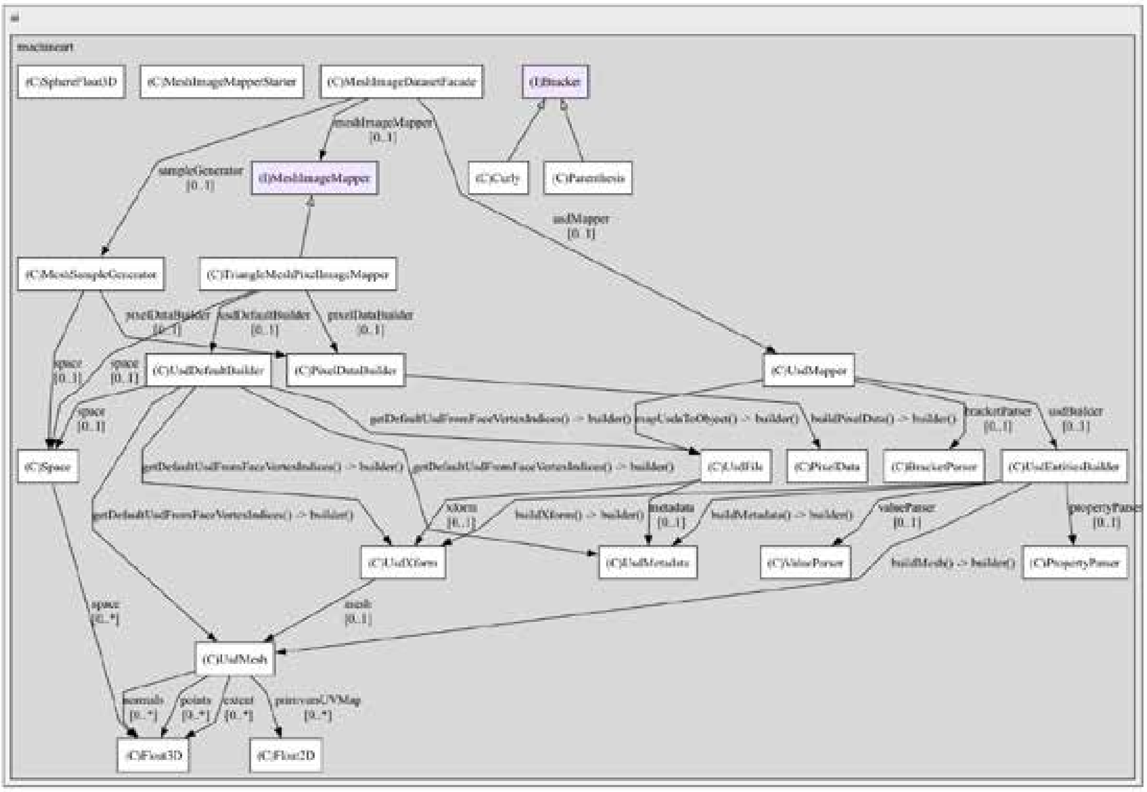

Fig. 2. Converter classes diagram

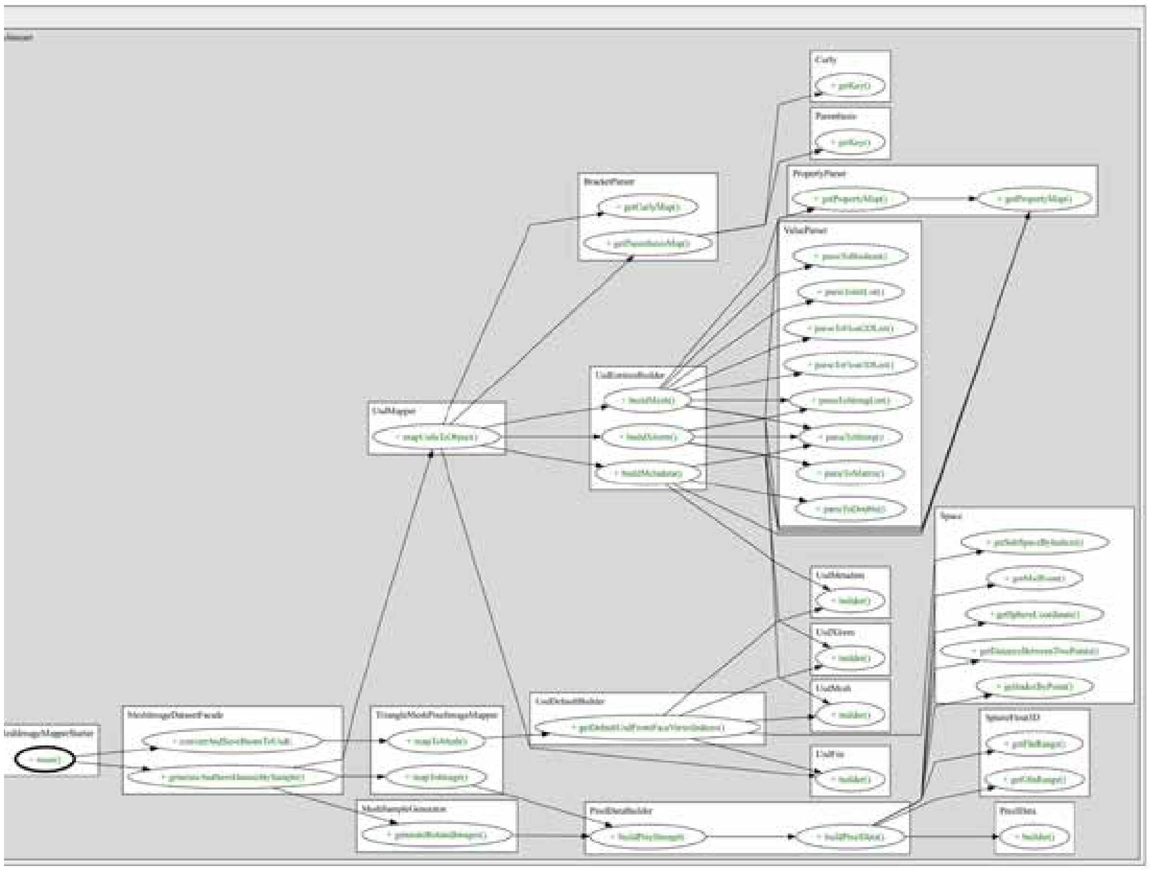

Fig. 3. Converter call diagram

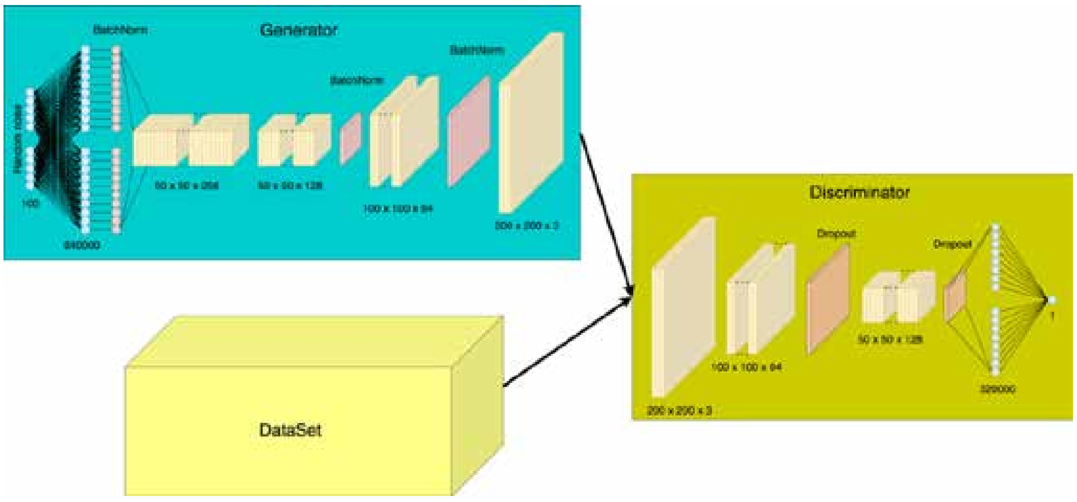

Fig. 4. DCGAN Model

As we can see, operations with vectors, matrices, and tensors are quite easily and concisely implemented in the Julia programming language.

Further, in Fig. 4, the schema of the DCGAN model used for testing PolyImage is presented. The diagram also illustrates the logical connection between the discriminator and the generator, as well as a set of training instances.

The diagram includes several additional elements necessary to improve the training process of the standard GAN model, including BatchNorm (to strengthen the generator) and Dropout (to slow down the discriminator). The ADAM optimization method with different combinations of learning rates was used to optimize the backpropagation algorithm. Most implementations of these methods rely on the Flux framework for the Julia programming language.

Summary and Conclusion. Conclusions from this research and prospects for further exploration in this direction. The scientific novelty lies in proposing a new type of representation for a three-dimensional object that can be used for training typical generative models.

The proposed method of representing a three-dimensional object demonstrated its viability even in the context of training a small typical DCGAN generative model. Perspectives for further research into the proposed method for training other typical generative models were also identified, as this method can be easily adapted to representations of input and output data in a wide range of neural network archite.

BIBLIOGRAPHY:

1. Ian, J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. Archive of scientific articles arXiv, 9 p. arXiv: 2014. 1406.2661.

2. Description of the working principle of the VAE model. Medium WebSite. URL: https://towardsdatascience.com/understanding-variational -autoencoders-vaes-f70510919f73.

3. A general description of the existing methods of presenting three-dimensional models. Carlow Institute of Technology website, Ireland. URL: https://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes /meshes/polygon_meshes.html.

4. Description of the most popular file formats for 3D objects. SelfCAD project site. URL: https://www.selfcad.com/blog/8-best-common-3d-file-formats.

5. Official documentation for Universal Scene Description. URL: https://openusd.org/release/index.html.

6. Jiajun, Wu, Chengkai, Zhang, Tianfan, Xue, William, T. Freeman, Joshua B. Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. Archive of scientific articles arXiv, 2017. 11 p. arXiv: 1610.07584.

7. Charlie, Nash. The shape variational autoencoder: A deep generative model of part-segmented 3D objects. Charlie Nash, C. K. I. Williams. – GitHub website of article author Charlie Nash, 2017. 11 p. URL: http://charlienash.github.io /assets/docs /sgp2017.pdf.

8. AtlasNet: A Papier-Mache Approach to Learning 3D Surface Generation. Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, Mathieu Aubry. – Archive of scientific articles arXiv, 2018. – 16 p. arXiv: 1802.05384.

9. Matheus Gadelha. Multiresolution Tree Networks for 3D Point Cloud Processing. / Matheus Gadelha, Rui Wang, Subhransu Maji. – Archive of scientific articles arXiv, 2018. – 17 p. arXiv: 1807.03520.

10. PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows. Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, Bharath Hariharan. – Archive of scientific articles arXiv, 2019. – 15 p. arXiv: 1906.12320.

11. Learning Representations and Generative Models for 3D Point Clouds. Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, Leonidas Guibas. – Archive of scientific articles arXiv, 2018. – 18 p. arXiv: 1707.02392.

12. PolyGen: An Autoregressive Generative Model of 3D Meshes. Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, Peter W. Battaglia. – Archive of scientific articles arXiv, 2020. – 16 p. arXiv: 2002.10880.

13. Occupancy Networks: Learning 3D Reconstruction in Function Space. Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, Andreas Geiger. – Archive of scientific articles arXiv, 2019. – 11 p. arXiv: 1812.03828.

14. Comparison of different configurations of the GAN model based on the AWS cloud computing service. Information Technology: Computer Science, Software Engineering and Cyber Security. B. Moroz., L. Kabak, K. Rodna, E. Ruksov, 2, 2022. p. 61–78. https://doi.org/10.32782/IT/2022-2-7.

REFERENCES:

1. Ian, J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. (2014). Generative Adversarial Nets. Archive of scientific articles arXiv, 9 p. arXiv: 1406.2661.

2. Description of the working principle of the VAE model. Medium WebSite. Retrieved from: https://towardsdatascience.com/understanding-variational -autoencoders-vaes-f70510919f73.

3. A general description of the existing methods of presenting three-dimensional models. Carlow Institute of Technology website, Ireland. Retrieved from: https://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes /meshes/polygon_meshes.html.

4. Description of the most popular file formats for 3D objects. SelfCAD project site. Retrieved from: https://www.selfcad.com/blog/8-best-common-3d-file-formats.

5. Official documentation for Universal Scene Description. Retrieved from: https://openusd.org/release/index.html.

6. Jiajun, Wu, Chengkai, Zhang, Tianfan, Xue, William, T. Freeman, Joshua B. Tenenbaum. (2017). Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. Archive of scientific articles arXiv, 11 p. arXiv: 1610.07584.

7. Charlie, Nash. (2017). The shape variational autoencoder: A deep generative model of part-segmented 3D objects. / Charlie Nash, C. K. I. Williams. – GitHub website of article author Charlie Nash, – 11 p. Retrieved from: http://charlienash.github.io /assets/docs /sgp2017.pdf.

8. Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, Mathieu Aubry. (2018). AtlasNet: A Papier-Mache Approach to Learning 3D Surface Generation. – Archive of scientific articles arXiv, 16 p. arXiv: 1802.05384.

9. Matheus Gadelha, Rui Wang, Subhransu Maji. (2018). Matheus Gadelha. Multiresolution Tree Networks for 3D Point Cloud Processing. Archive of scientific articles arXiv, 17 p. arXiv: 1807.03520.

10. Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, Bharath Hariharan. (2019). PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows. Archive of scientific articles arXiv, – 15 p. arXiv: 1906.12320.

11. Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, Leonidas Guibas. (2018). Learning Representations and Generative Models for 3D Point Clouds. Archive of scientific articles arXiv, 18 p. arXiv: 1707.02392.

12. Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, Peter W. Battaglia. (2020). PolyGen: An Autoregressive Generative Model of 3D Meshes. Archive of scientific articles arXiv,16 p. arXiv: 2002.10880.

13. Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, Andreas Geiger. (2019). Occupancy Networks: Learning 3D Reconstruction in Function Space. / – Archive of scientific articles arXiv, 11 p. arXiv: 1812.03828.

14. Moroz, B., Kabak, L., Rodna, K. & Ruksov, E. (2022). Comparison of different configurations of the GAN model based on the AWS cloud computing service. Information Technology: Computer Science, Software Engineering and Cyber Security, 2 p. 61–78. https://doi.org/10.32782/IT/2022-2-7.