ДОДАТОК А

Слайди презентації

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

АТЕСТАЦІЙНА РОБОТА МАГІСТРА

Дослідження та реалізація методів 3D моделювання з використанням 2D зображень

Виконав:
Ст. Гр. іПЗМ-18-4
Федосенко Н.В.

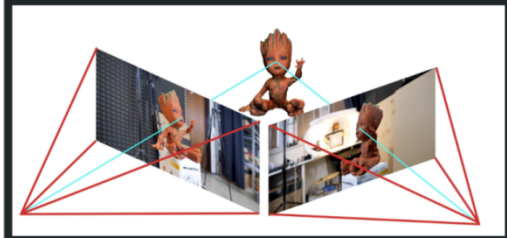
Науковий керівник:
проф. Білоус Н.В.

Мета роботи

- Реалізація методів 3D моделювання, для подальшого дослідження та вдосконалення існуючих методів.
- Програмна реалізація веб-додатку для створення 3D моделей з зображень.

Аналіз предметної галузі

Методи, засновані на створенні моделі з набору зображень



Методи, засновані на алгоритмах машинного навчання

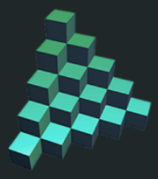
Постановка задачі

- Дослідження методів машинного навчання для створення 3D моделей з зображень
- Реалізувати веб-додаток, що надасть змогу користувачам завантажувати зображення та отримувати 3D модель
- Інтегрувати методи машинного навчання

Існуючі рішення

INSIGHT3D

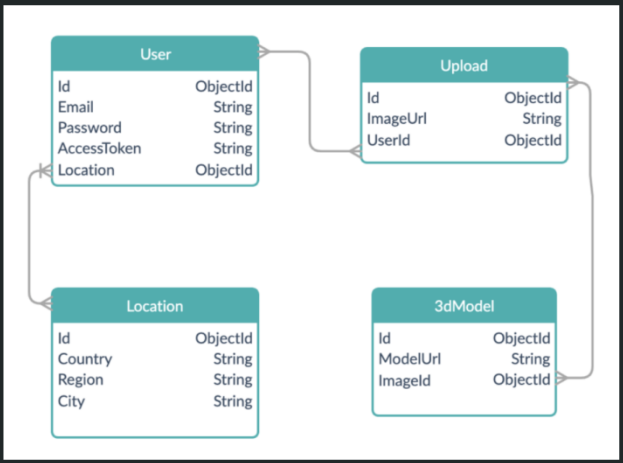
SMOOTHIE 3D



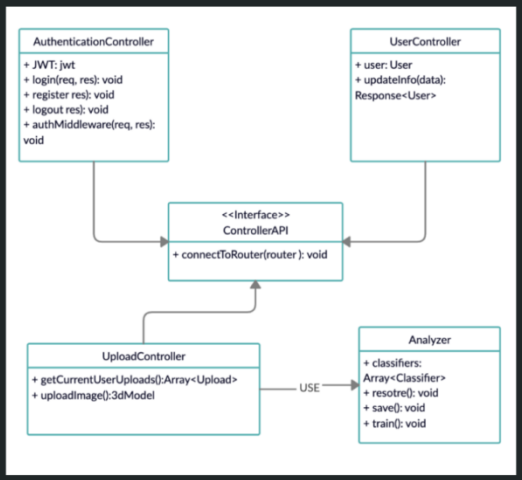
Стек технологій

- Javascript
- React.js
- Node.js
- Python, Keras

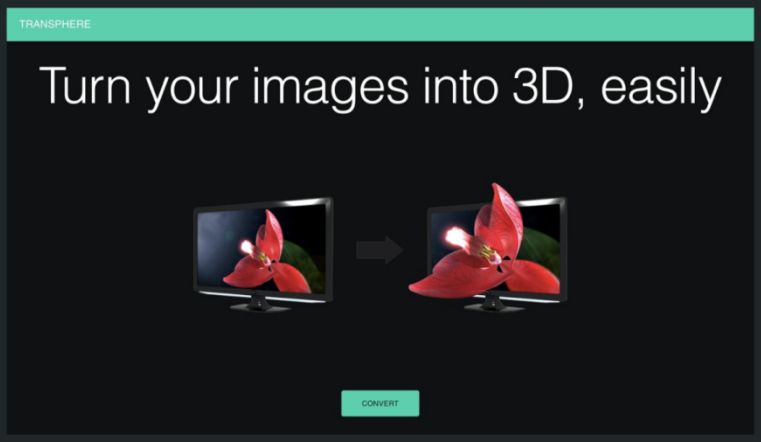
ER бази даних



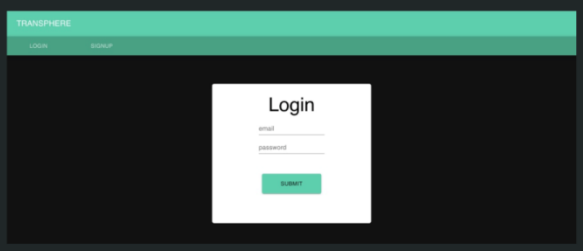
Діаграма класів

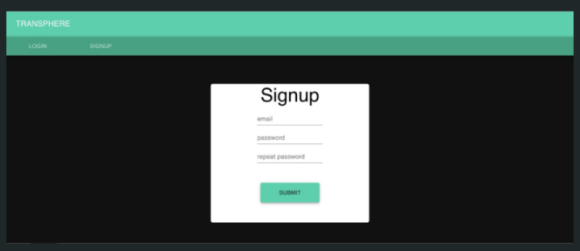


Інтерфейс системи (Landing)

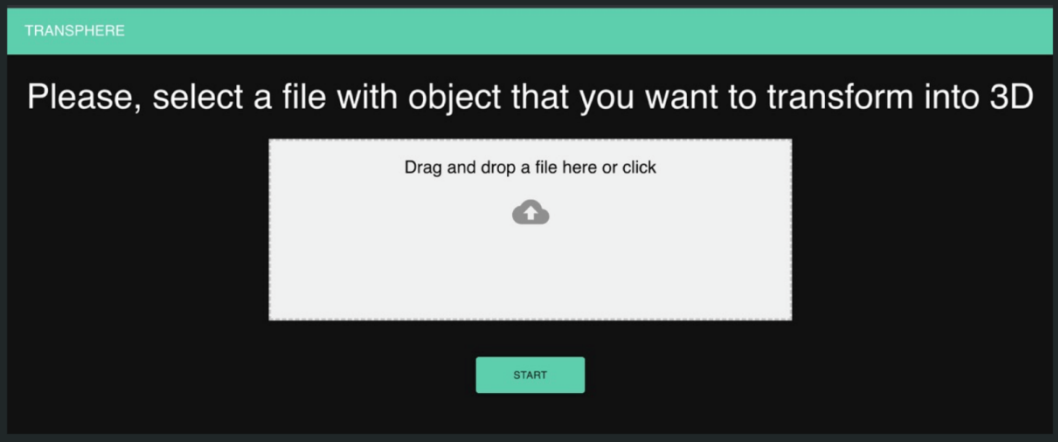


Інтерфейс системи (Log in/ Sign up)





Інтерфейс системи (Upload page)



Висновки

- Проведено аналіз предметної галузі та виявлені існуючі проблеми
- Спроектовано веб-додаток для створення 3D моделей з зображень
- Розроблено програмний продукт згідно до виставлених вимог
- Досліджено та реалізовано методи 3D моделювання з 2D зображень

ДОДАТОК Б

Фрагменти коду програми

```
import chainer
import cupy as cp


def _voxelize_sub1(faces, size, dim=2):
    assert (0 <= dim)
    bs, nf = faces.shape[:2]
    if dim == 0:
        i = cp.array([2, 1, 0])
        faces = faces[:, :, :, i]
    elif dim == 1:
        i = cp.array([0, 2, 1])
        faces = faces[:, :, :, i]
    faces = cp.ascontiguousarray(faces)
    voxels = cp.zeros((faces.shape[0], size, size, size), 'int32')
    chainer.cuda.elementwise(
        'int32 j, raw T faces, raw int32 bs, raw int32 nf, raw int32 vs',
        'raw int32 voxels',
        '''
            int y = j % vs;
            int x = (j / vs) % vs;
            int bn = j / (vs * vs);
            //
            for (int fn = 0; fn < nf; fn++){
                float* face = &faces[(bn * nf + fn) * 9];
                float y1d = face[3] - face[0];
                float x1d = face[4] - face[1];
                float z1d = face[5] - face[2];
                float y2d = face[6] - face[0];
                float x2d = face[7] - face[1];
                float z2d = face[8] - face[2];
                float ypd = y - face[0];
                float xpd = x - face[1];
                float det = x1d * y2d - x2d * y1d;
                if (det == 0) continue;
                float t1 = (y2d * xpd - x2d * ypd) / det;
                float t2 = (-y1d * xpd + x1d * ypd) / det;
                if (t1 < 0) continue;
                if (t2 < 0) continue;
                if (1 < t1 + t2) continue;
                int zi = floor(t1 * z1d + t2 * z2d + face[2]);
                int yi, xi;
                yi = y;
                xi = x;
                if ((0 <= yi) && (yi < vs) && (0 <= xi) && (xi < vs) && (0
<= zi) && (zi < vs))
                    voxels[bn * vs * vs * vs + yi * vs * vs + xi * vs + zi]
= 1;
                yi = y - 1;
                xi = x;
```

```
                if ((0 <= yi) && (yi < vs) && (0 <= xi) && (xi < vs) && (0
<= zi) && (zi < vs))
                    voxels[bn * vs * vs * vs + yi * vs * vs + xi * vs + zi]
= 1;
                yi = y;
                xi = x - 1;
                if ((0 <= yi) && (yi < vs) && (0 <= xi) && (xi < vs) && (0
<= zi) && (zi < vs))
                    voxels[bn * vs * vs * vs + yi * vs * vs + xi * vs + zi]
= 1;
                yi = y - 1;
                xi = x - 1;
                if ((0 <= yi) && (yi < vs) && (0 <= xi) && (xi < vs) && (0
<= zi) && (zi < vs))
                    voxels[bn * vs * vs * vs + yi * vs * vs + xi * vs + zi]
= 1;
            }
        ''',
        'function',
    )(cp.arange(bs * size * size).astype('int32'), faces, bs, nf, size,
voxels)
    voxels = voxels.swapaxes(dim + 1, -1)
    return voxels


def _voxelize_sub2(faces, size):
    bs, nf = faces.shape[:2]
    faces = cp.ascontiguousarray(faces)
    voxels = cp.zeros((faces.shape[0], size, size, size), 'int32')
    chainer.cuda.elementwise(
        'int32 j, raw T faces, raw int32 bs, raw int32 nf, raw int32 vs',
        'raw int32 voxels',
        '''
            int fn = j % nf;
            int bn = j / nf;
            float* face = &faces[(bn * nf + fn) * 9];
            for (int k = 0; k < 3; k++) {
                int yi = face[3 * k + 0];
                int xi = face[3 * k + 1];
                int zi = face[3 * k + 2];
                if ((0 <= yi) && (yi < vs) && (0 <= xi) && (xi < vs) && (0
<= zi) && (zi < vs))
                    voxels[bn * vs * vs * vs + yi * vs * vs + xi * vs + zi]
= 1;
            }
        ''',
        'function',
    )(cp.arange(bs * nf).astype('int32'), faces, bs, nf, size, voxels)
    return voxels


def _voxelize_sub3(voxels):
    # fill in
    bs, vs = voxels.shape[:2]
    voxels = cp.ascontiguousarray(voxels)
    visible = cp.zeros_like(voxels, 'int32')
```

```
    chainer.cuda.elementwise(
        'int32 j, raw int32 bs, raw int32 vs',
        'raw int32 voxels, raw int32 visible',
        '''
            int z = j % vs;
            int x = (j / vs) % vs;
            int y = (j / (vs * vs)) % vs;
            int bn = j / (vs * vs * vs);
            int pn = j;
            if ((y == 0) || (y == vs - 1) || (x == 0) || (x == vs - 1) || (z
== 0) || (z == vs - 1)) {
                if (voxels[pn] == 0) visible[pn] = 1;
            }
        ''',
        'function',
    )(cp.arange(bs * vs * vs * vs).astype('int32'), bs, vs, voxels, visible)

    sum_visible = visible.sum()
    while True:
        chainer.cuda.elementwise(
            'int32 j, raw int32 bs, raw int32 vs',
            'raw int32 voxels, raw int32 visible',
            '''
                int z = j % vs;
                int x = (j / vs) % vs;
                int y = (j / (vs * vs)) % vs;
                int bn = j / (vs * vs * vs);
                int pn = j;
                if ((y == 0) || (y == vs - 1) || (x == 0) || (x == vs - 1)
|| (z == 0) || (z == vs - 1)) return;
                if (voxels[pn] == 0 && visible[pn] == 0) {
                    int yi, xi, zi;
                    yi = y - 1;
                    xi = x;
                    zi = z;
                    if (visible[bn * vs * vs * vs + yi * vs * vs + xi * vs +
zi] != 0) visible[pn] = 1;
                    yi = y + 1;
                    xi = x;
                    zi = z;
                    if (visible[bn * vs * vs * vs + yi * vs * vs + xi * vs +
zi] != 0) visible[pn] = 1;
                    yi = y;
                    xi = x - 1;
                    zi = z;
                    if (visible[bn * vs * vs * vs + yi * vs * vs + xi * vs +
zi] != 0) visible[pn] = 1;
                    yi = y;
                    xi = x + 1;
                    zi = z;
                    if (visible[bn * vs * vs * vs + yi * vs * vs + xi * vs +
zi] != 0) visible[pn] = 1;
                    yi = y;
                    xi = x;
                    zi = z - 1;
```

```
                    if (visible[bn * vs * vs * vs + yi * vs * vs + xi * vs +
zi] != 0) visible[pn] = 1;
                    yi = y;
                    xi = x;
                    zi = z + 1;
                    if (visible[bn * vs * vs * vs + yi * vs * vs + xi * vs +
zi] != 0) visible[pn] = 1;
                }
            ''',
            'function',
        )(cp.arange(bs * vs * vs * vs).astype('int32'), bs, vs, voxels,
visible)
        if visible.sum() == sum_visible:
            break
        else:
            sum_visible = visible.sum()
    return 1 - visible


def voxelize(faces, size, normalize=False):
    faces = cp.copy(faces)
    if normalize:
        faces -= faces.min((0, 1, 2), keepdims=True)
        faces /= faces.max()
        faces *= 1. * (size - 1) / size
        margin = 1 - faces.max((0, 1, 2))
        faces += margin[None, None, None, :] / 2
        faces *= size
    else:
        faces *= size

    voxels0 = _voxelize_sub1(faces, size, 0)
    voxels1 = _voxelize_sub1(faces, size, 1)
    voxels2 = _voxelize_sub1(faces, size, 2)
    voxels3 = _voxelize_sub2(faces, size)
    voxels = voxels0 + voxels1 + voxels2 + voxels3
    voxels = (0 < voxels).astype('int32')
    voxels = _voxelize_sub3(voxels)

    return voxels

import argparse
import os
import random

import chainer
import cupy as cp
import neural_renderer
import numpy as np
import skimage.io

import models

RANDOM_SEED = 0
GPU = 0
DIRECTORY = './data/models'
```

```
def tile_images(images):
    rows = int(images.shape[0] ** 0.5)
    cols = int(images.shape[0] ** 0.5)
    image_size = images.shape[2]
    if images.ndim == 3:
        image = np.zeros((rows * image_size, cols * image_size))
    else:
        image = np.zeros((rows * image_size, cols * image_size,
images.shape[1]))
        images = images.transpose((0, 2, 3, 1))
    for i in range(rows):
        for j in range(cols):
            image[i * image_size:(i + 1) * image_size, j * image_size:(j +
1) * image_size] = images[i * cols + j]
    return image


def run():
    # arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('-eid', '--experiment_id', type=str)
    parser.add_argument('-d', '--directory', type=str, default=DIRECTORY)
    parser.add_argument('-i', '--input_image', type=str)
    parser.add_argument('-oi', '--output_image', type=str)
    parser.add_argument('-oo', '--output_obj', type=str)
    parser.add_argument('-s', '--seed', type=int, default=RANDOM_SEED)
    parser.add_argument('-g', '--gpu', type=int, default=GPU)
    args = parser.parse_args()
    directory_output = os.path.join(args.directory, args.experiment_id)

    # set random seed, gpu
    random.seed(args.seed)
    np.random.seed(args.seed)
    cp.random.seed(args.seed)
    chainer.cuda.get_device(args.gpu).use()

    # load dataset
    image_in = skimage.io.imread(args.input_image).astype('float32') / 255
    if image_in.ndim != 3 or image_in.shape[-1] != 4:
        raise Exception('Input must be a RGBA image.')
    images_in = image_in.transpose((2, 0, 1))[None, :, :, :]
    images_in = chainer.cuda.to_gpu(images_in)

    # setup model & optimizer
    model = models.Model()
    model.to_gpu()
    chainer.serializers.load_npz(os.path.join(directory_output,
'model.npz'), model)

    # reconstruct .obj
    vertices, faces = model.reconstruct(images_in)
    neural_renderer.save_obj(args.output_obj, vertices.data.get()[0],
faces.get()[0])
```

```
    # render reconstructed shape
    ones = chainer.cuda.to_gpu(np.ones((16,), 'float32'))
    distances = 2.732 * ones
    elevations = 30. * ones
    azimuths = chainer.cuda.to_gpu(np.arange(0, 360, 360. /
16.).astype('float32')) * ones
    viewpoints = neural_renderer.get_points_from_angles(distances,
elevations, azimuths)
    images_out =
model.reconstruct_and_render(chainer.functions.tile(images_in, (16, 1, 1,
1)), viewpoints)
    image_out = tile_images(images_out.data.get())
    image_out = (image_out * 255).clip(0, 255).astype('uint8')
    skimage.io.imsave(args.output_image, image_out)


if __name__ == '__main__':
    run()
```

ДОДАТОК В

Апробація результатів роботи. Наукова стаття

Comparing 2D to 3D conversion methods

Nikita Fedosenko
Kharkiv National University of Radio
Electrtonics
Kharkiv, Ukraine
nikita.fedosenko@nure.ua

***Abstract*—With the growth of technologies related to augmented and virtual reality, the problem of creating quality 3D models is becoming more relevant. In this paper are describe and compared the main algorithms for generating 3D models from a series of 2D images or a single 2D image.**

Keywords—3D reconstruction, machine learning, 2D to 3D conversion, image analysis.

Introduction

Currently, the area of multimedia systems, based on extending real world with some virtual objects is growing rapidly. Furthermore, the technology of 3D printers is becoming popular nowadays. Even e-commerce platforms are tend to use 3D models of products for a better user experience. As a result, a variety of modern areas are required to use 3D modelling to fit their needs. Currently, the technologies level allow us to automate the process of generating 3D models, instead of creating them manually.

The problem of 3D reconstruction from images is one of the most challenging. Basically, we can group methods for 3D reconstruction from images into two groups: the ones, based on using of several images of an object from different angles and the ones based on machine learning and computer vision techniques. The last ones tend to be more complicated, but can provide more precise and fast solution. Also, some of machine learning solutions can reconstruct 3D model even from single image.

In this paper, we will explore the fundamentals and implementation details of each group of methods for 3D reconstruction, along with pros and cons of them.

Camera-based methods

There are different approaches to 3D reconstruction, which are based on various optical systems used to capture images. Typically, an optical system can be implemented either as a specialized depth camera (for example, an active depth sensor or stereo camera), or as a traditional monocular camera. Most of the methods use depth maps data for 3D reconstruction from images. The main principle of this method is to create a copy of initial image with indication of depth of object presented there [1]. On Figure 1 you can see an example of a depth map made from image.

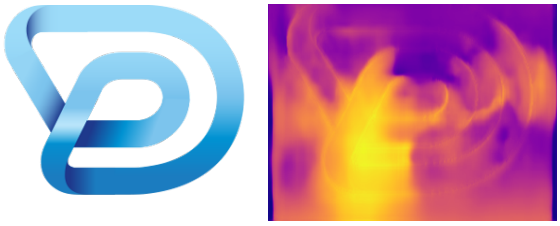


Figure 1. Example of a depth map, generated for a 2D image

A specialized depth camera can capture depth maps (either directly in the case of an active depth sensor, or after processing a pair of adjusted stereo images in the case of a stereo camera) in real time. Each pixel in depth maps corresponds to a discrete measurement of distance obtained by a camera from a 3D point. This approach has two drawbacks: firstly, it requires expensive hardware to implement a depth sensor used to capture depth maps, and secondly, the processing algorithm is computationally complex and requires a powerful graphics engine for processing real-time data.

In approaches with a monocular camera, a 3D reconstruction based on silhouettes can be used. In this method, images captured by the camera are used to extract silhouettes that then form a three-dimensional structure, together with their intersections [2]. The disadvantage of this method is its computational complexity with respect to volume combining and silhouettes extraction, which limits its implementation only on powerful pcs or cloud computation. Another disadvantage is that this method cannot restore concavities that are not visible in silhouette images.

There are two approaches to solving the three-dimensional reconstruction problem: active and passive. The active method uses depth sensors that directly interact with the object. Such systems are complex and require additional equipment. In the passive method, a camera is used as a sensor, which captures images of an object from all parties. This approach does not require specialized depth gauges and can be applied in any conditions. However, the accuracy of such a three-dimensional reconstruction substantially depends on the quality obtained photos and reconstruction algorithm.

One of the promising approaches in the construction of three-dimensional objects is the stereo reconstruction of an object based on it stereo images [3]. Stereo reconstruction is depth map restoration task observed scene where depth is

estimated for each pixel of the image. Stereo image object - a picture or video using two separate images of the object obtained from two cameras that watch the same scene from one horizontal offset between cameras.

The process of obtaining a three-dimensional model object can be divided into several stages.

Camera calibration step or projecting scene points onto an image. For this must be mathematically removed radial and tangential distortions. Such a process called distortion removal (undistortion) or correction. At this stage receive undistorted images from each cameras.

The calibration phase of a pair of cameras. For this need to adjust the angle and distance between cameras providing stereo images. This process is called rectification.

The stage of selection of control points on multiple images and finding matches of these points. The step of determining three-dimensional coordinates points. The step of determining three-dimensional coordinates points.

The camera calibration step consists of definitions of internal parameters regarding photo forming systems and external regarding the configuration of various angles view. Internal camera settings include camera focal length, location point and radial distortion introduced the lens. External camera settings include camera position relative to a specific coordinate system. These settings include determination of the center of the camera and the angles of rotation, describing the orientation of the camera with respect to main coordinate system. In addition, the camera position may be computed by matching between points in multiple images of a scene without direct calibration.

Camera calibration is usually done in account of multiple shooting some calibration template on the image of which you can easily highlight key points for of which their relative positions in space. Further systems of equations connecting projection coordinates, camera arrays and position dots pattern in space.

There are public implementations of calibration algorithms e.g. Matlab Calibration toolbox. Also a computer-vision library called OpenCV includes camera calibration and search algorithms for calibrating pattern on the image.

As a result, in the next step there are two calibrated cameras set by their matrices P and P' in some coordinate system. If the centers of the cameras do not match, then this pair of cameras can be used to define three-dimensional coordinates of observed points [4]. For stereo matching images obtained from two cameras, need to translate these images into same plane with vertically oriented lines. When the cameras tuned that way say a couple image is fixed.

Existing Correction Algorithms stereo images use epipolar geometry for image alignment. The most common is Boget's algorithm.

This method of creating 3D models is based on the list of images of object, made from different angles. Having multiple images could help us to find the intersection between them. This process is known as triangulation (Figure 2) and it helps to find the relations between images for sculpting the final 3D model.

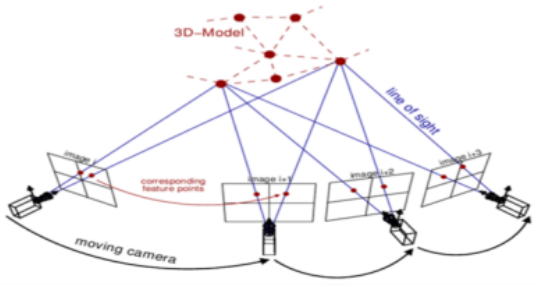


Figure 2. Capturing 3D object from different angles

Deep Learning Methods

In recent years, Deep Learning (DL) has demonstrated outstanding capabilities in solving 2D-image tasks such as image classification, object detection, semantic segmentation, etc. Not an exception, DL has showed tremendous progresses in applying it to 3D graphic problems. Most of the deep learning algorithms, suitable for extending data from images and creating self-learning networks are based on using convolutional neural networks.

Convolutional Neural Network is a neural network which consists of convolutional layers. Usually in convolution neural networks there is also a layer of subsampling and fully connected layer. Convolutional neural networks apply for optical pattern recognition, image classification, object detection, semantic segmentation and other tasks.

The foundations of the modern architecture of convolutional neural networks were laid in one of the first widely known convolutional neural network - LeNet-5 Yana LeCun, which architecture is shown in Figure 3.

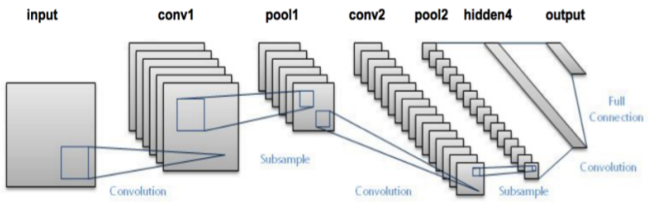


Figure 3. Architecture of LeNet-5 convolutional neural network

In convolutional neural networks, convolution and subsampling layers consist of several “levels” of neurons, called feature maps, or channels. Each neuron of this layer is connected to a small section of the previous layer, called receptive by the field [5]. In the case of an image, the feature map is a two-dimensional array of neurons, or simply a matrix. Other measurements can be used if another type of data is received at the input, for example, audio data (one-dimensional array) or volumetric data (three-dimensional array).

In the convolution layer, each feature map corresponds to one convolution core, also called a filter. Each neuron, as its output value, performs the operation of convolution or cross-correlation with its receptive layer.

It is worth noting that these two operations in the context of training convolutional neural networks are interchangeable,

as a result of which, in many software implementations, the operation “Convolution” is actually a cross-correlation operation.

Since the convolution core is the same for each feature map, this allows the neural network to learn how to distinguish features outside depending on their location in the input image and also leads to a significant reduction in the number of parameters. According to established notation, they say that the convolution layer uses filter W x H, if each filter in this layer has number W x H x C where, C dimension channels in the previous layer [6].

The down sampling layer implements compaction of signs cards of the previous layer and does not change the number of cards. Each layer feature map connected to appropriate feature card previous layer, each neuron performs “compression” of its receptive fields through any function. The most popular types of this layer are Max Pooling (from the receptive layer the maximum value is selected), Average
Pooling (select average) and L2 Pooling (L2 norm is selected). Using the down sampling layer resistance to small input image shifts as well the dimension of subsequent Layers. Fully Connected Layer - Normal Hidden multilayer perceptron layer, connected to all neurons of the previous layer. Thus, images are fed to the input of the convolutional neural network, and the output to which is the class to which belongs to the image.

A single image is only a projection of 3D object into a 2D plane, so some data from the higher dimension space must be lost in the lower dimension representation. Therefore, from a single-view 2D image, there will never be enough data construct its 3D component.

A method to create the 3D perception from a single 2D image therefore requires prior knowledge of the 3D shape in itself. In 2D Deep Learning, a Convolutional AutoEncoder is a very efficient method to learn a compressed representation of input images. Extending this architecture into learning a compact shape knowledge is the most promising way to apply Deep Learning to 3D data [7].

Unlike a 2D image that has only one universal representation in computer format (pixel), there are many ways to represent 3D data in in digital format. They come with their own advantages and disadvantages, so the choice of data representation directly affected the approach that can be utilized.

Voxel, in short for volumetric pixel, is the direct extension of spatial-grid pixels into volume-grid voxels. The locality of each voxels together define the unique structure of this volumetric data, so the locality assumption of ConvNet still hold true in volumetric format. However, this representation is sparse and wasteful. The density of useful voxels decreases as the resolution increases.

Polygonal mesh is a collection of vertices, edges and faces that defines the objects’ surface in 3 dimensions. It can capture granular details in a fairly compact representation.

Point Cloud is a collection of points in 3D coordinate (x, y, z), together these points form a cloud that resemble the shape of object in 3 dimension. The larger the collection of points, the more details it gets. The same set of points in different order still represents the same 3D object. You can see the representation of these layers on Figure 4.

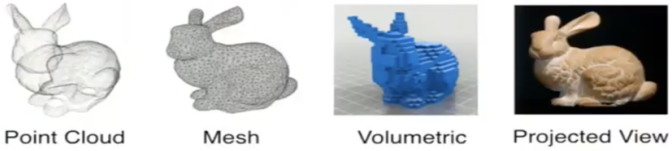


Figure 4. Visualization of different methods of rendering 3D model, depending on 3D layer type

Conclusions

To sum up, currently there are a variety of methods for reconstructing 3D models from a set of images or even from a single image. The universal solution for generating 3D models from images doesn’t exist. Although, many methods requires using of cameras and a complicated process of capturing object from different angles, machine learning techniques have more potential. In this paper we have explored the main methods for 2D to 3D conversion, based on both depth maps and capturing object from cameras and the ones based on machine learning and using of convolutional neural networks.

References

[1] Redert, A.,Patent ID: WO2005083630 A2, WO2005083630 A2, WO2005083631 A2, “Creating a Depth Map”, Royal Philips Electronics, the Netherlands, 2005.

[2] Kao, M.A., Shen, T.C. “A novel real time 2D to 3D conversion technique using depth based rendering”. IDW'09, p. 203, 2009.

[3] Wong, K.T.; Ernst, F., Master thesis “Single Image Depth-fromDefocus”, Delft university of Technology & Philips Natlab Research, Eindhoven, The Netherlands, 2004.

[4] J. Arvo. Linear-time voxel walking for octrees. Ray Tracing News, 1(2), 1988

[5] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015.

[6] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. arXiv preprint arXiv:1604.03265, 2016.

[7] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In IROS, 2015.

ДОДАТОК Г

Акт прийому наукової статті

Submission Summary

Conference Name
2020 IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT)

Track Name
Artificial intelligence

Paper ID
81

Paper Title
Comparing 2D to 3D conversion methods

Abstract
With the growth of technologies related to augmented and virtual reality, the problem of creating quality 3D models is becoming more relevant. In this paper are described and compared the main algorithms for generating 3D models from a series of 2D images or a single 2D image.

Created on
30.04.2020, 14:25:47

Last Modified
30.04.2020, 14:25:47

Authors
Nikita Fedosenko (Kharkiv National University of Radio Electronics) <nikita.fedosenko@nure.ua>

Submission Files
диплом-публикация.pdf (1.1 Mb, 30.04.2020, 14:25:41)