ISSN 2464-4617(print)
ISSN 2464-4625(CD)
Computer Science Research Notes
CSRN 2801
Full Papers Proceedings
http://www.WSCG.eu

3D Object Classification and Parameter Estimation based on Parametric Procedural Models

Roman Getto Kenten Fina Lennart Jarms
Technische Universität Darmstadt
Fraunhoferstr. 5 64283 Darmstadt, Germany
{firstname.lastname}@gris.tu-darmstadt.de

Arjan Kuijper Dieter W. Fellner
Technische Universität Darmstadt & Fraunhofer IGD
Fraunhoferstr. 5, 64283 Darmstadt, Germany
arjan.kuijper@mavc.tu-darmstadt.de
dieter.fellner@gris.tu-darmstadt.de

ABSTRACT

Classifying and gathering additional information about an unknown 3D objects is dependent on having a large amount of learning data. We propose to use procedural models as data foundation for this task. In our method we (semi-)automatically define parameters for a procedural model constructed with a modeling tool. Then we use the procedural models to classify an object and also automatically estimate the best parameters. We use a standard convolutional neural network and three different object similarity measures to estimate the best parameters at each degree of detail. We evaluate all steps of our approach using several procedural models and show that we can achieve high classification accuracy and meaningful parameters for unknown objects.

Keywords

Procedural model, parametric model, parameterization, 3D object classification, deep learning.

1 INTRODUCTION

The most widely accepted approach for 3D Object Classification is the database-approach. A class is learned by having all types of example objects within a database. However, this approach is not applicable to all domains. A large database with all examples for the desired classes is not always available. In research environment there are several big databases that provide enough data to learn different classes and then evaluate the performance of a classification algorithm. In real applications we have actual classes of objects in mind which do not fit to the classes offered in the test databases. The amount of 3D data is often not available and the cost and time effort to produce such a database is tremendous. For 2D (image) classifications the data-approach is more affordable since images are available for literally everything. Many approaches tried to make 3D data more available and delivered environments to easily create new 3D objects, even for non-expert users. However, in direct comparison to image data, 3D data is still near non-existent. When insufficient data is available, one approach is to represent a class directly by a procedural model. The procedural model is a more abstract description of an object by representing the object implicitly by a parameterizable object construction algorithm. Therefore, the procedural model corresponds to a blueprint of an object class. Creating a single procedural model includes some effort but can then be used as a complete data foundation for a desired class. In many cases it is more affordable to create a blueprint instead of gathering a vast amount of example objects.

Our contribution is a complete processing pipeline to achieve classification and parameter estimation using procedural models as basis. Also, the pipeline contains three separate contributions: The algorithm to (semi-)automatically generate parameters for a procedural model, a scheme to use procedural models for deep learning, and the parameter estimation technique using three different similarity measures.

In the following Section we review related work. Section 3 presents our methodology including the procedural model definition, the classification with deep learning, and the parameter estimation. In Section 4 we evaluate and discuss each step of the pipeline individually. Finally, we conclude and outline future work.

2 RELATED WORK

Procedural models are often referred to as grammars [Tal11] or L-systems [Šťb10]. In general a 3D procedural model is a description of building scheme for a class of 3D objects, which allows to easily generate many different variations. Therefore, procedural models excel in content generation. Instead of an implicit grammar representation, a procedural model can also be represented by a concatenation of parameterized procedures. The

ISBN 978-80-86943-40-4
https://doi.org/10.24132/CSRN.2018.2801.2

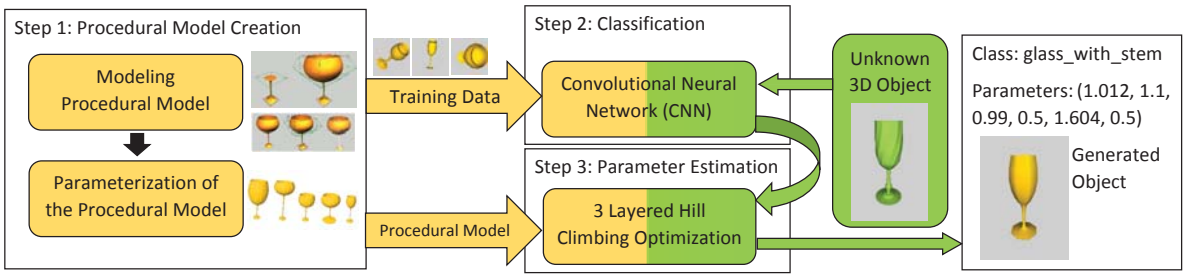

Figure 1: The full pipeline of our system: procedural models are created to represents blueprints of an objects class. These are used to classify and estimate the parameters of an unknown database object.

sequence describes the building process and the parameterization allows the variation of the building process.

Bokeloh et al. [Bok12] propose a procedural modeling algorithm which works on regular structured polygon meshes. A procedural model is automatically generated, so that parameters change the shape of the object while preserving the regular patterns optimally. This approach shows that procedural models are generally very powerful in terms of flexibility. Still, this approach is only suited for cases with regular structured objects.

Other approaches tackle the problem of variation generation by recombining several objects. Jain et al. [Jai12] create variations by part-based recombinations. Yumer et al. [Yum15] define variations with terms like 'luxurious' 'sporty' or 'expensive'. Wang et al. [Wan11] use a symmetry hierarchy to vary objects. Other approaches use box templates [Ave14] to represent a blueprint of a class. Generally, all these approaches are limited to the already available 3D objects. In terms of flexibility procedural models are vastly superior.

In a previous work [Get17] we proposed a definition of procedural models as a concatenation of procedures by using modeling operations. We use this framework for our work to define our initial procedural models.

Ullrich et al. [Ull11] presented a work with a concept similar to ours. They define a 3D object procedurally and compare the procedural model to a query object to estimate the parameters. However, in their approach, the procedural model is designed and parameterized manually and the similarity is only based on a surface difference measure. Our approach includes semi-automatization of the creation of the procedural model, deep learning of the class and a more reliable layered parameter estimation.

To measure the difference between two 3D Objects many so-called descriptors have been proposed. These are focused on different aspects, using histograms [Osa02], topology graphs [Mar07] or image properties of rendered images [Vra05]. For our initial parameter estimation we use the panorama descriptor [Pap10] which is considered to be one of the best geometrical descriptors [Li15] .

For the 3D object classification the descriptors have also been used to directly learn single classes of 3D objects [Wes08, Wan15]. However, deep learning mostly outclassed previous approaches. Maturana et al. [Mat15] and Wu et al. [Wu15] proposed convolutional neural network (CNN) approaches directly learning on voxel representations. Su et al. [Su15] developed a multi-view CNN learning on rendered 2D Images of 3D objects. With this approach they achieve higher accuracy than any comparable approach. The authors reason that currently the relative efficiency using 2D data is higher than using 3D representations. For this reason we also use a CNN approach learning on 2D rendered images.

3 METHODOLOGY

We propose a system based on procedural models. We train a Convolution Neural Network (CNN) with the procedural model and propose a technique to estimate the values of all parameters of the procedural model. We present the concept of our pipeline in Figure 1.

The procedural model itself consists of a concatenation of parameterized procedures. In contrast to an explicit surface representation like a polygon mesh, the procedural model is an implicit object representation. The procedures describe a construction algorithm. When the procedures are executed subsequently an instance of the procedural model is generated. The instance of the procedural model is a 3D polygon mesh itself. When the parameters of the procedures are changed, the resulting mesh changes. Therefore, the procedural model offers the possibility to generate infinite variations by varying the parameters.

For the initial creation of a procedural model we use the tool and the algorithm of Getto et al. [Get17]. The concept of the tool is that a procedural model is automatically generated during the modeling of a single object. The modeling operations are transformed to procedures

ISBN 978-80-86943-40-4

of the procedural model obeying several rules, e.g. the rule of locality, so that parameter changes only have local effects. The boundary representation is a sparse control mesh of a subdivision surface. The edges can be marked as smooth or sharp. It offers basic operations, allowing to insert, remove, drag and connect vertices, edges and faces. Additional a path of face extrusions can be performed by sketching a line from a face.

3.1 Semi-Automatic Parameter Insertion

While the procedural model is modeled manually, we propose a semi-automatic parameter insertion technique to enhance the process of creating the fully parameterized procedural model. Our goal is to compute several possible variations and show them to the user, so that the user can decide which variations make sense. Therefore, the user can define all parameters with a few clicks. As the user cannot inspect every possible parameter, we order the possible variations by 'importance' and furthermore automatically group related parameters together. Table 1 shows the complete overview of the relevant operations. Operations only including ids (e.g. connect faces) are not relevant for the parameterization.

Procedure Parameters: An extrusion and a drag is described with cylindrical coordinates ρ, ϕ and z. A rotation-extrude is defined by the width w and length l. The insert vertex operation defines the the position of the new vertex on an edge as barycentric coordinate λ_1 ($\lambda_2 = 1 - \lambda_1$). A scale has a relative size σ and a rotate has a rotation angle α.

Automatic Variations: For all these operations, we define parameter variations to evaluate the importance. These are shown in Table 1. The automatic variations mostly include doubling, halving or inverting the parameters as suitable.

Importance Evaluation Measures: To measure the importance of an operation we follow a simple rule: The bigger a change the higher the importance. We generate a single mesh for every variation and compare the varied mesh to the original mesh taking into account 5 measures. The overview Table 1 shows the composition of the evaluation measure for each parameter variation. For all measures the base mesh is normalized, so that the centroid is at the origin and the mesh is within a radius of 1. The volume is computed with the method of Zhang et al. [Zha01]. The surface area is the sum of all polygon areas. The bounding sphere has its center at the origin and its radius is the distance to the furthest vertex of the mesh. The coordinate plane projection difference is computed by projecting the surface of the mesh on to the three coordinate plane. We conduct this projection by creating an image of 64x64 pixels for each plane. A pixel is set to true if any part of the object is projected onto this pixel. The average distance is the average Euclidean distance of a vertex of the original mesh and the respective vertex of the varied mesh. The final value of the difference of the two meshes is calculated by the following equations:

$$\delta_p = \frac{p(v) - p(b)}{p(b)} \tag{1}$$

$$\delta_{projection} = \frac{\sum_{i=0}^{m} xor(v_{pixel\_i} - b_{pixel\_i})}{m} \tag{2}$$

$$\delta_{vertexdistance} = \frac{\sum_{i=0}^{n} |v_{vertex\_i} - b_{vertex\_i}|}{n} \tag{3}$$

$$v = variation\ mesh, b = base\ mesh,$$
$$p \in \{volume, surface, BSvolume\},$$
$$m = number\ of\ pixels\ in\ projection\ planes,$$
$$n = number\ of\ vertices\ in\ the\ mesh$$

Parameter Grouping: Groups of parameters are new parameters themselves. When the group parameter is changed all underlying operations are changed respectively. Groups of parameters are formed by finding related operations with related parameters. This is generally the case if two operations are similar. Operations are similar if their values are similar. Therefore, we first define the similarity of two values x and y and two angles α and β:

$$similarity(x,y) = 1 - \frac{|x-y|}{max(1,|x|,|y|)} \tag{4}$$

$$similarity(\alpha,\beta) = 1 - \frac{|\alpha-\beta|}{c} \quad c \in \{45,90\} \tag{5}$$

For the similarity of angles we need to cover additional special cases since two angles of related operations should be considered similar if the one angle is the mirrored version of the other. The angles are defined in a local plane in u-v-space. We consider 4 different angles: the original, mirrored on the u-axis, mirrored on the v-axis and mirrored on both. Furthermore, we check 4 additional angles: the original angle rotated by 90 degrees and all 3 mirrored version of this angle. For this 4 angles the criteria for the similarity are more tight: we half the range of similarity, which is achieved by exchanging the 90 by a 45 within the equation.

To calculate the similarity of two operations we multiply all similarities of their parameters. Finally, we set a threshold for the similarity of each operation as some operations are much more likely to have a higher similarity (e.g. insert vertex) than others.

After finding all similar operations we need to further process them to identify actual groups. We can deduct the relation of operations by their relative position in the sequence of operations. We identified that related instructions are either present in the pattern AA or with the pattern ABAB. This means they are not only similar but also subsequent, as in pattern AA. Or a combination of operations AB is subsequent, forming the pattern ABAB. Finally, we build groups of similar operations which are present in one of these two patterns within the sequence of operations of the procedural model.

User based parameter choice: A parameter is considered to be important if the importance value is bigger or

 ISBN 978-80-86943-40-4

Operation	Procedure Parameters	Automatic Variation	Importance Evaluation Measure	New Inserted Parameter x	Initial Range of x	Similarity Threshold
Extrude (Sketching)	(ρ, ϕ, z)	$(2\rho, \phi, 2z)$	$0.1\delta_{volume} + 0.25\delta_{surface} + 0.1\delta_{BSvolume} + 0.2\delta_{projection}$	$(x\rho, \phi, xz)$	[0.125,8]	0.3
		$if\ \rho > 0.15z\ and\ \phi \in [90,270): (\rho, \phi + 180(\sqrt{0.5} - 1), z)$	$0.75\delta_{projection} + 0.1\delta_{vertexdistance}$	$(\rho, \phi + 180(x-1), z)$	[0,2.0]	
		$if\ \rho > 0.15z\ and\ \phi \in [270,90): (\rho, \phi + 180(\sqrt{1.5} - 1), z)$				
Rotation-Extrude (Sketching)	(w, l)	$(w, 2l)$	$0.15\delta_{volume} + 0.3\delta_{surface} + 0.1\delta_{BSvolume} + 0.2\delta_{projection}$	(w, xl)	[0.125,8]	0.45
		$(0.5w, l)$	$0.7\delta_{volume} + 0.15\delta_{surface} + 0.05\delta_{projection}$	(xw, l)	$[-2,1]$	
Insert Vertex	(λ_1) $\lambda_2 = 1 - \lambda_1$	$if\ \lambda_1 \geq 0.5 : (0.5 + 0.5\lambda_1)$	$0.1\delta_{surface} + 0.3\delta_{projection} + 0.15\delta_{vertexdistance}$	$(0.5 + x\lambda_1)$	[1,1.9]	0.95
		$if\ \lambda_1 < 0.5 : (0.5\lambda_1)$		$(x\lambda_1)$	[0.1,1.0]	
Drag	(ρ, ϕ, z)	$(2\rho, \phi, 2z)$	$0.1\delta_{surface} + 0.5\delta_{projection} + 0.15\delta_{vertexdistance}$	$(x\rho, \phi, xz)$	$[-8,8]$	0.85
Scale	(σ)	$if\ \sigma > 1 : (2\sigma)$	$0.5\delta_{volume} + 0.25\delta_{surface} + 0.1\delta_{BSvolume} + 0.15\delta_{projection}$	$(x\sigma)$	$[-3,3]$	0.9
		$if\ \sigma < 1 : (0.5\sigma)$				
Rotate	(α)	$(-\alpha)$	$0.7\delta_{projection} + 0.1\delta_{vertexdistance}$	$(x\alpha)$	[0.125,5]	0.7

ρ = radial distance, ϕ = angular coordinate, z = height , w = width, l = length, λ = baryzentric coordinate, σ = relative scale, α = rotation angle
δ_{volume} = volume difference, $\delta_{surface}$ = surface area difference, $\delta_{BSvolume}$ = bounding sphere volume difference,
$\delta_{projection}$ = coordinate plane projection difference, $\delta_{vertexdistance}$ = average vertex distance difference

Table 1: Overview of all relevant operations used to automatically parameterize the procedural model.

equal than 1. Additionally, we include the user in this step and offer a simple interface to inspect all parameters and choose all important parameters. This interface is shown in Figure 2. The parameters are ordered by their importance. The user can check or uncheck any individual parameter or parameter group. He can create new groups and rename parameters.

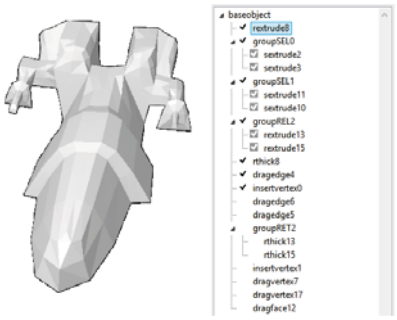

Figure 2: The user interface of the parameter insertion.

Range Estimation: For the random generation of variations we need to additionally define a valid range for the parameter x. The overview Table 1 shows the initial range estimations for every parameter type. For each parameter we generate a mesh with the maximal and minimal value for the specific parameter and measure the difference to the base mesh using the panorama distance [Pap10]. If the panorama distance is bigger than a threshold $C \cdot t$ the range is diminished and reevaluated. t is obtained by measuring the panorama distance to all generated variations of all inserted parameters.

$$Threshold = C \cdot t \tag{6}$$

$$C = constant\ multiplier (default\ C = 1)$$
$$t = max \in \{panorama\ distance\ to\ all\ generated\ meshes\}$$

3.2 Classification with Deep Learning

We propose to use the very deep Convolutional Neural Network (CNN) Inception [Sze15] to directly learn the 3D object with rendered images of the object. The last fully connected layer of the inception network can be retrained with a relative small amount of 3D data. Also the retraining is tremendously faster than training a network from scratch. We retrain the last fully connected layer with a randomized set of images of rendered views of the 3D object. Also, we additionally generate random variations of the 3D object within these images.

Each procedural model represents an object class. For each class we generate 1000 variations (3D mesh). We vary each parameter of the procedural model randomly within the parameter range. For each of the 1000 variations we generate 10 images. In sum, we use 10 000 images per class to train the network.

Figure 3: Examples of generated learning images

Image Generation: We use rendered images of the generated 3D objects with random perspectives. Like [Su15] we noticed that different illumination setups did not make significant differences in the results. In Figure 3 we show example images of our setup. Before generating the images the 3D object is first normalized. The center of mass of the mesh is translated to the origin. The object is scaled, so that all coordinate values are within -1 and 1. We include a random rotation and scaling of the object for each image. The scaling is limited to the range $[0.8, 2]$.

Object Classification: With the retrained network we can classify images. The output of the network for a single image is a class probability for each trained class. To classify a new 3D object we generate 10 images and average the classification values for each class. The 3D object is then put in the class with the highest value.

ISBN 978-80-86943-40-4

3.3 Parameter Estimation

The procedural model has several parameters to generate variations. We estimate those parameters for a new 3D object having the same class as the procedural model. The parameters can either be labeled by the user, e.g. 'wing length', or the influence of the parameters can be shown visually to the user by generating exemplary objects for different values. In both cases estimating the parameters for an unknown object gives valuable information to the user.

The parameter estimation is based on geometrical similarities of the unknown 3D object and the objects generated by the procedural model. It is important to note that the procedural models cannot reproduce any object perfectly in full detail. We use 3 different measures with different degrees of detail. The panorama distance [Pap10], the surface distance and a z-buffer distance.

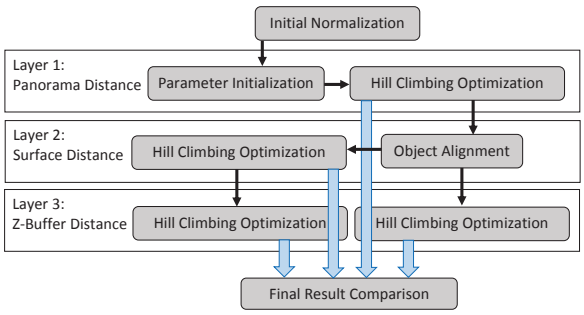

Figure 4: The parameter estimation consists of 3 layers using 3 different levels of distance measures. The final result is the best result of 4 different optimizations.

Our algorithm includes 3 layers for these 3 measures. We show an overview of the parameter estimation in Figure 4. We use all measures subsequently in a hill climbing optimization to refine the estimated optimal parameters step by step. Additionally we set thresholds for the measures, so that the result of the preceding layer is taken if the object cannot be represented precisely on a layer. As a result we do not only estimate the best parameters but actually are able to tell how well the parameters of the procedural model can represent the unknown 3D object.

Panorama Distance: The panorama distance is defined by the panorama descriptor [Pap10], which is a hybrid descriptor based on geometrical features and image features of panoramic views of the object.

Surface Distance: The surface distance, also known as point-to-surface-distance is based on the distance between the actual surface polygons of both meshes. To compute the surface distance between an instance of the procedural model and the unknown object we generate a set of points for both meshes. For the unknown object we use the Poisson disk sampling [Cor12] with 2000 points. For the mesh generated by the procedural model we take all vertices of the mesh after 2 iterations of the subdivision. For each set of points the surface distance of a single point is the distance to the nearest point of the other set. We average this distance over all points.

Z-Buffer Distance: We compare the z (depth) information of both objects pixel wise. For this distance we generate a total of 14 views with 256x256 pixels. We use an orthogonal projection with $[-2,2]$ for all boundary planes. The 14 views are the 6 views directly from the positive and negative coordinate axes and the 8 views from the corners of a cube around the origin. We Present the distance calculation in Algorithm 1. Note that we penalize an undersizing of the generated object more than an oversizing. This reinforces the initial growing into all regions.

Algorithm 1 Z-Buffer Distance

```
1:  procedure (Original Obj. O, Generated Obj. G)
2:      Generate 14 views V_O for O and V_G for G
3:      Distance d ← 0
4:      for all Views v_O ∈ V_O do
5:          for all Pixels p_O ∈ v_O do
6:              if p_O = background ∧
                   p_G != background then
7:                  d = d + 1
8:              else if p_O != background ∧
                   p_G = background then
9:                  d = d + 2
10:             else if p_O != background ∧
                   p_G != background then
11:                 d = d + |z(p_O) − z(p_G)|
12:             end if
13:         end for
14:     end for
15:     d is the final distance
16: end procedure
```

Initial normalization: At the start we bring both objects into a shared coordinate system. The average position of the vertices of the mesh (center of mass) is translated to the origin. The object is scaled, so that all coordinate values are within -1 and 1.

Hill climbing algorithm: Each layer includes a hill climbing search with one of the distance measures. We show the hill climbing search for a distance measure D in Algorithm 2. All parameters are optimized with diminishing step sizes. Note that to measure the distance with D and parameters P we generate a new object using the procedural model with the parameters P.

For the surface distance (layer 2) and the z-buffer distance (layer 3) an additional intermediate step has to be inserted: When parameters of the procedural model change and the object shape changes, the position of the object is shifted respectively. Therefore after each change of a parameter value the objects have to be realigned before the distance measure is computed. Just

ISBN 978-80-86943-40-4

Algorithm 2 Hill Climbing Optimization

```
1:  procedure (Distance Measure D, Parameters P)
2:      Distance d ← D(P)
3:      do
4:          Step size s ← 1.0
5:          do
6:              for all p ∈ P do
7:                  for all ⊕ ∈ {+, −} do
8:                      p_new = p ⊕ s
9:                      clamp p_new to [min, max] of p
10:                     Distance d_new ← D(p_new)
11:                     if d_new < d then
12:                         d = d_new
13:                         p = p_new
14:                     end if
15:                 end for
16:             end for
17:         while no parameter has been changed
18:         s = next s ∈ {1.0, 0.5, 0.25, 0.1, 0.01, 0.0}
19:     while s! = 0.0
20: end procedure
```

like the parameter adjustment, we perform a greedy search for the best translation, rotation and scaling. The step size is fixed for this realignment: 0.01 for the translation and scaling, and $0.01 \cdot 180^\circ$ for the rotation. The scaling is limited to a minimum of 0.5 and a maximum of 1.5 of the original scale.

Layer 1 - panorama distance: Before the first layer the initial normalization (scaling and translation) is performed. In the first layer the parameter initialization and hill climbing optimization with the panorama distance is performed (See Figure 4). The initialization of the parameters of the procedural model is of major importance since the following greedy hill climbing algorithms can get stuck in a local extremum. The panorama distance generally measures the distance between two 3D objects and is optimally suited to fulfill this task. We generate a total of 10 000 objects from the procedural model with random parameterization, covering a large range of possible initialization values. We compute the panorama distance of each generated object to the unknown object. The parameters of the generated object with the smallest panorama distance are taken as our starting point. Then we perform a hill climbing optimization of all parameters using the panorama distance.

Layer 2 - surface distance: In the second layer the object alignment and hill climbing optimization with the surface distance is performed (See Figure 4). For the surface distance measure and the following z-buffer distance measure we need to optimize the alignment of both objects. We optimize the translation, rotation and scaling in a greedy search like introduced in the hill climbing description. However, this greedy search only remedies small misalignments. Since the initial orientation can be majorly flawed we additional consider 24 possible coordinate system rotations. The 24 rotations include all main rotation possibilities: the x-axis can be rotated to match one of the 6 possible positive or negative coordinate system axis and can be rotated around itself by $0, 90, 180$ or 270 degrees. Giving a total of $6 \cdot 4 = 24$ possibilities. For each of the 24 possibilities we perform the alignment optimization and evaluate the case with surface distance. The case with the lowest surface distance is taken as the initial alignment. Finally, we perform the hill climbing optimization of all parameters using the surface distance.

Layer 3 - z-buffer distance: In the final layer two hill climbing optimizations with the z-buffer distance are performed (See Figure 4). In the first case we use the output of layer 2 as input and in the second case we use the output of layer 1 (after the alignment in layer 2) as input. Hence, we compute optimal parameters for 2 different starting point. Then we compare the z-buffer distance of the two final optimization results and take the better solution as final result.

At the end of our parameter estimation we decide which layer result is the most adequate representation. The procedural model might not be able to represent every object to a pixel wise degree, hence we set thresholds for the final results to decide to which extent the procedural model represents the unknown object. We analyzed several objects, results and distance measures values and identified shared thresholds for the distance measures. A z-buffer distance of lower than 0.7 and a surface distance of lower than 0.04 represents an adequate match. The final parameters correspond to the result of layer 3 if the z-buffer distance is below 0.7. Else it corresponds to the result of layer 2 if the surface distance is below 0.04. If both are not the case than the result of layer 1 gives the final parameters.

4 EVALUATION AND DISCUSSION

In this section we evaluate our approach, including our 3 steps: the parameter insertion, classification with deep learning and the parameter estimation. For each step we separately show results and discuss the results.

4.1 Parameter Insertion

To evaluate the correctness of the proposed semi-automatic parameter insertion technique we evaluate the importance evaluation measure and the parameter grouping. We created several example models (See Figure 5) with the modeling tool and manually annotated important parameters and appropriately grouped related parameters. Then we retrieved the proposed important parameters and groups automatically detected by the default threshold of 1. We present our

 ISBN 978-80-86943-40-4

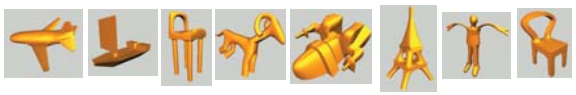

Figure 5: Models of the parameter insertion evaluation.

	Importance					Grouping			
	CP	FP	FN	CN	Accuracy	CG	FG	MG	Accuracy
Airplane	9	1	1	11	90.91%	7	0	0	70.00%
Ship	7	3	0	8	83.33%	3	1	0	66.67%
Stool	6	3	0	12	85.71%	8	1	0	87.50%
Animal	11	12	1	18	69.05%	7	0	0	100.00%
Spaceship	6	1	0	6	92.31%	3	0	0	100.00%
Tower	11	4	1	7	78.26%	4	0	1	80.00%
Humanoid	10	8	1	12	70.97%	9	0	1	90.00%
Chair	6	6	0	8	70.00%	7	1	0	85.71%
Average	9.42	5.43	0.57	11.71	80.07%	6.00	0.38	0.63	84.98%
CP = Correct Positive FP = False Positive, FN = False Negative CN = Correct Negative						CG = Correct Group FG = False Group MG = Missed Group			

Table 2: The accuracy of the importance evaluation measure and the accuracy of the parameter grouping

results in Table 2. The most relevant parameters are automatically categorized as important in 80% of the cases. 84% percent of the groups are correctly identified by the algorithm.

Discussion: Table 2 shows that false negatives are seldom. Our algorithm rather finds too many important parameters, so that false positives occur. In several cases parameters are found to be important even though the change of the parameter does not lead to a semantically consistent outcome. For this reason, we include a user phase where the user can refine parameters. Therefore, the user is able to correct semantic inconsistencies.

The parameter grouping also has erroneous cases. The majority of the missing groups and false groups are caused by the insert vertex operation. The insert vertex only includes a single parameter and the values are mostly within $[0.25, 0.75]$. Even though we have set a tight threshold with 0.95 for this operation, the similarity computation is less reliable for this operation. We highlight the insert vertex groups for the user, so that he can decide in these cases.

4.2 Classification with Deep Learning

To evaluate our classification approach with deep learning on rendered images, we constructed 10 procedural models. Figure 6 shows the models. The 10 procedural models represent 10 different classes. To evaluate our approach we use the NIST database [Fan08] and the Princeton shape benchmark (PSB) [Shi04] together. Table 3 shows the number of objects of each class in the databases and also show properties of the procedural models.

We took the preset classes within the given databases (since the NIST database only has single a class with spiders and insects our spider class includes both). We trained our network with a total of 100 000 images (10

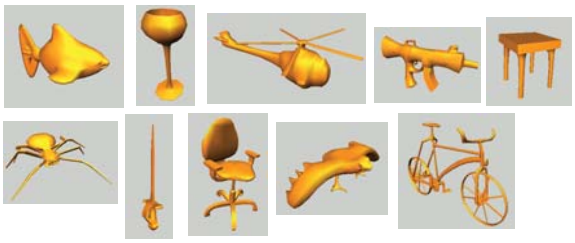

Figure 6: All models that were used to evaluate the classification and parameter estimation step.

	Database			Procedural Model			
	NIST	PSB	Total	po	to-ops	par-ops	par
fish	18	17	35	1348	150	68	13
glass_with_stem	18	9	27	332	44	11	6
helicopter	18	35	53	1560	294	184	12
gun	36	39	75	888	161	54	15
table	36	63	99	614	87	51	6
spider	18	16	34	2976	197	137	9
sword	18	31	49	568	64	30	7
office_chair	18	15	33	1724	193	86	8
bird	18	21	39	2040	241	134	13
bicycle	18	7	25	4966	510	189	10
others	504	1562	2066				
total within classes	216	253	469				
total	720	1815	2535				
po = number of polygons to-ops = total number of operations par-ops = parameterizable operations par = number of parameters							

Table 3: The number of objects for each class in the PSB and the NIST database, and the properties of the used procedural models

classes,1000 variations with 10 images). The retraining took about 10 hours on a casual PC. We used our algorithm for 2 different scenarios: the classification of database objects and a 3D object retrieval scenario.

Classification: We classified every object of the databases that belong into one of the 10 learned classes and measured the overall classification accuracy. Figure 7 (right) shows the accuracy for all 10 classes resulting in the average accuracy of 86.14% (404 of 469 objects are classified correctly).

3D object retrieval: The output of a 3D object retrieval query is a list of retrieved objects, ordered from the most similar to the least similar. We directly use the class probabilities given by the neural network to sort the list of retrieved objects. A class label is the query itself and the first object of the retrieval list is the 3D object with the highest class probability for this class. Here we include all 2535 objects of all databases.

Figure 7 shows the precision recall curve for the 3D object retrieval. We compare this result with the panorama distance by using the default instance of the procedural model as query for the database. The panorama distance from this object to all objects in the database is calculated and the retrieval list is sorted respectively. We also show the result of our approach without vari-

ISBN 978-80-86943-40-4

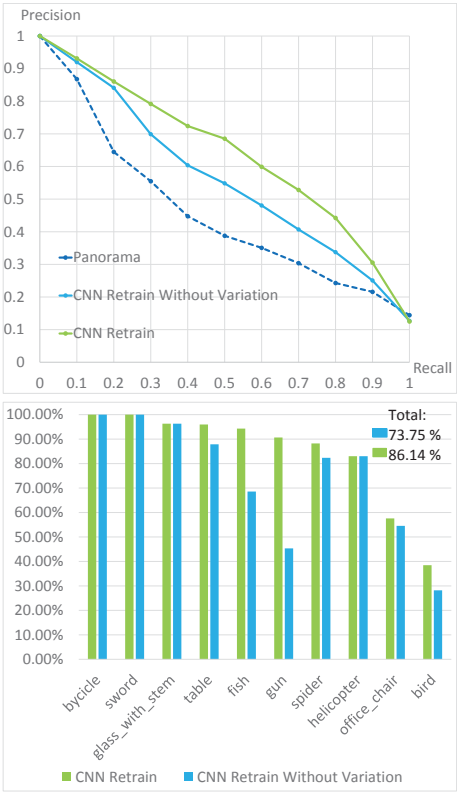

Figure 7: The 3D object retrieval precision-recall curve and the classification accuracy.

ations: we generate all images without changing any parameters and retrain the network with these images.

Discussion: The classification and the 3D object retrieval illustrate several properties of the approach. An important insight is that including variations into the learning process leads to improvements. This is not as trivial as it might seem at first glance. We tested several other possibilities of image generation, including random translations and higher variations of scaling and found out that it is easier for the neural network to learn the class when the images are more consistent. At the same time a good amount of variability is needed in the images to prevent overfitting and promote generalizability. However, our results clearly show that object variations enhance the results in all cases.

The average classification accuracy is 86%. This is comparable to state-of-the-art approaches like [Su15] achieving 83-90% accuracy on the classification task. Only the office chair and bird class achieved a lower accuracy. The database objects are not sufficiently similar to the initial procedural model. In Figure 8 we show falsely classified objects. The parameters did not compensate very exceptional variations of the objects.

In the precision-recall curve (Figure 7) our approach also outperforms the panorama distance, even though the panorama distance is among the best geometrical distance measures. In sum, our deep learning retraining

Figure 8: Images of falsely classified objects.

approach with rendered images of variations is fast and works with less data than a full network learning and still generates comparable results.

4.3 Parameter Estimation

For the final step we took all correctly classified examples of our 10 classes and estimated the parameters of the procedural model for every unknown database object. In total 66.09% of the final parameter estimations origin from layer 3. 10.15% from layer 2 and 23.76% from layer 1. Figure 9 shows the distribution of the surface distance and z-buffer distance for the 4 different results in the 3 layers. Figure 10 presents several exemplary parameter estimations for all classes.

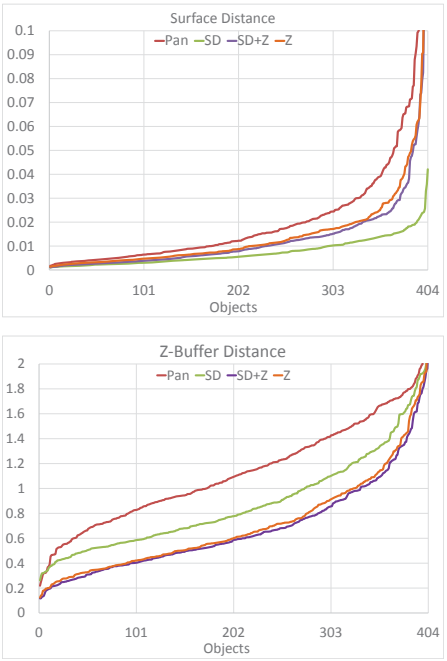

Figure 9: The distribution of the surface distance and the z-buffer distance for all objects for the 4 different results from the 3 layers.

Discussion: In Figure 9 we can detect a general advantage of the layered optimization system. In the plots we present the two different results from layer 3 separately: using the surface distance with z-Buffer distance(SD+Z) and only the z-buffer distance(Z). Here, we can see that the distribution of the z-buffer distance in the final layer is better on average when the output of the 2nd layer is taken as input (SD+Z). The hill climbing algorithm naturally profits from a good initialization. We can see that not only the initial setup improves the results, but also the intermediate optimization of layer 2 improves the results of the final layer 3.

ISBN 978-80-86943-40-4

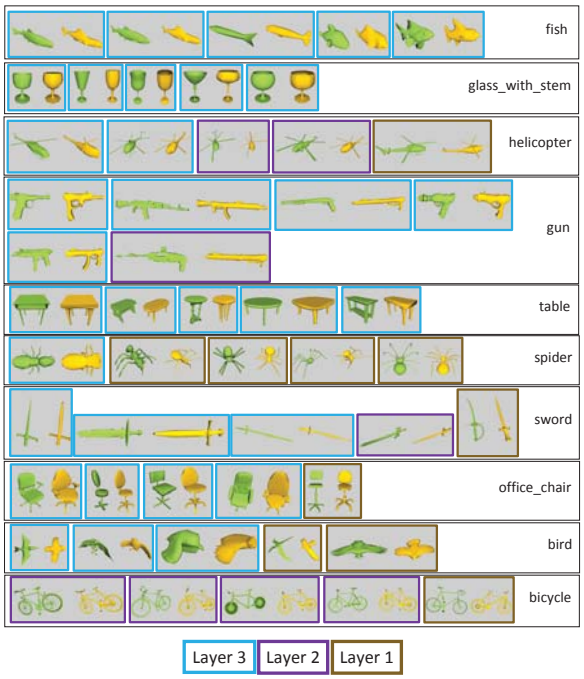

Figure 10: Exemplary results for all classes. The colored borders show from which layer the result origins.

Figure 11: Ten different glasses of the database sorted by the ratio of stem length to the bowl length.

The examples presented in Figure 10 show that the parameter estimations lead to generated objects with similar overall appearance compared to the unknown database objects. Most objects could be estimated on layer 3 (z-buffer). However, the bicycle, spider and helicopter class did not have enough flexibility to represent most of the objects on layer 3. Especially the rotors of the helicopter, the legs of the spiders and the thin spokes and connection bars of the bicycle could not be matched pixel-wise. The user can improve the estimations for the classes by adding additional parameters to increase the flexibility. Nonetheless, our system is able to provide meaningful results from layer 2 (surface distance) and layer 1 (panorama distance) for the cases where the procedural model is not suitable enough for the objects.

Figure 11 presents an object characteristic derived by the parameters. Here we order the objects by the ratio of the stem length to the bowl length. Important to note in this context is that ratios and differences between parameters are more meaningful than the comparison of values of a single parameter. This is the case because the database objects have to be normalized and the instances of the procedural models have to be scaled and aligned accordingly. Therefore, the actual values itself are less comparable when the coordinate systems of different objects do not match. In the use case of having scanned objects as input, no normalization is needed since the values are related to real millimeter values. In this case the actual values of single parameters are also completely comparable.

Figure 12 shows two types of errors that we found in the results. The bird is mostly symmetrical, so that the instance of the procedural model happens to be misaligned. The head and the tail are facing in the wrong direction. These cases happened at some symmetrical objects of the bird, fish and gun class. In the future we will have to integrate an additional symmetry detection to handle these cases explicitly.

The second error type is represented by the glasses with stem in Figure 12. The database object does not have a real stem. The bowl is directly connected to the base. The procedural model does not include the case of a stem having 0 length. Even though this result comes from layer 3, the final parameters are distorted by the falsely estimated stem length.

Figure 12: The bird is falsely aligned. The glass has an estimation of the stem length even though the glass of the database has no stem.

5 CONCLUSION & FUTURE WORK

We proposed a new approach including a system to model and parameterize complete procedural models, train a convolutional neural network solely with the procedural models and finally classify an unknown object from a database and additionally estimate all parameters of the procedural model for the unknown object. Hence, our system does not only classify unknown objects but also retrieve additional information.

The proposed system has a very high potential when suitable procedural models can be created. Therefore, the currently biggest drawback is the need to model the initial model with the modeling tool. We will further investigate the possibilities of automatizing this step. Creating a method that can automatically construct a procedural model from a single object in mesh representation would highly enhance the ease and usability of our system.

Our learning method shows a clear enhancement of the results by using the variations of the objects. A further investigation of the exact mechanisms leading to this effect should be performed. This would enable advanced possibilities of enforcing this mechanisms.

The accuracy of the final parameter estimation step is directly dependent on the provided procedural models. Therefore, the final estimation will improve by further enhancing the creation of the procedural model itself.

ISBN 978-80-86943-40-4

6 REFERENCES

[Ave14] Averkiou M., Kim V. G., Zheng Y., Mitra N. J. Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis. In Computer Graphics Forum, vol. 33, Wiley Online Library, pp. 125–134, 2014.

[Bok12] Bokeloh M., Wand M., Seidel H.-P., Koltun V. An algebraic model for parameterized shape editing. ACM Transactions on Graphics 31, No. 4, pp. 1–10, 2012.

[Cor12] Corsini M., Cignoni P., Scopigno R. Efficient and flexible sampling with blue noise properties of triangular meshes. IEEE Transactions on Visualization and Computer Graphics 18, No. 6, pp. 914–924, 2012.

[Fan08] Fang R., Godil A., Li X., Wagan A. A new shape benchmark for 3d object retrieval. Advances in Visual Computing, pp. 381–392, 2008.

[Get17] Getto R., Merz J., Kuijper A., Fellner D. W. 3d meta model generation with application in 3d object retrieval. In Proceedings of the Computer Graphics International Conference, ACM, p. 6, 2017.

[Jai12] Jain A., Thormählen T., Ritschel T., Seidel H.-P. Exploring Shape Variations by 3d-Model Decomposition and Part-based Recombination. In Computer Graphics Forum, vol. 31, Wiley Online Library, pp. 631–640, 2012.

[Li15] Li B., Lu Y., Li C., Godil A., Schreck T., Aono M., Burtscher M., Chen Q., Chowdhury N. K., Fang B., et al. A comparison of 3d shape retrieval methods based on a large-scale benchmark supporting multimodal queries. Computer Vision and Image Understanding 131, pp. 1–27, 2015.

[Mar07] Marini S., Spagnuolo M., Falcidieno B. Structural shape prototypes for the automatic classification of 3d objects. IEEE Computer Graphics and Applications, No. 4, pp. 28–37, 2007.

[Mat15] Maturana D., Scherer S. Voxnet: A 3d convolutional neural network for real-time object recognition. In Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, IEEE, pp. 922–928, 2015.

[Osa02] Osada R., Funkhouser T., Chazelle B., Dobkin D. Shape distributions. ACM Transactions on Graphics (TOG) 21, No. 4, pp. 807–832, 2002.

[Pap10] Papadakis P., Pratikakis I., Theoharis T., Perantonis S. Panorama: A 3d shape descriptor based on panoramic views for unsupervised 3d object retrieval. International Journal of Computer Vision 89, No. 2, pp. 177–192, 2010.

[Shi04] Shilane P., Min P., Kazhdan M., Funkhouser T. The princeton shape benchmark. In Shape modeling applications, 2004. Proceedings, IEEE, pp. 167–178, 2004.

[Šťava10] Šťava O., Beneš B., Měch R., Aliaga D. G., Krištof P. Inverse procedural modeling by automatic generation of l-systems. In Computer Graphics Forum, vol. 29, Wiley Online Library, pp. 665–674, 2010.

[Su15] Su H., Maji S., Kalogerakis E., Learned-Miller E. Multi-view convolutional neural networks for 3d shape recognition. In Proceedings of the IEEE international conference on computer vision, pp. 945–953, 2015.

[Sze15] Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9, 2015.

[Tal11] Talton J. O., Lou Y., Lesser S., Duke J., Měch R., Koltun V. Metropolis procedural modeling. ACM Transactions on Graphics (TOG) 30, No. 2, p. 11, 2011.

[Ull11] Ullrich, Torsten, Fellner, Dieter W. Generative Object Definition and Semantic Recognition. 2011.

[Vra05] Vranic D. V. Desire: a composite 3d-shape descriptor. In Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on, IEEE, pp. 4–pp, 2005.

[Wan11] Wang Y., Xu K., Li J., Zhang H., Shamir A., Liu L., Cheng Z., Xiong Y. Symmetry Hierarchy of Man-Made Objects. In Computer graphics forum, vol. 30, Wiley Online Library, pp. 287–296, 2011.

[Wan15] Wang Y., Liu Z., Pang F., Li H. Boosting 3d model retrieval with class vocabularies and distance vector revision. In TENCON 2015-2015 IEEE Region 10 Conference, IEEE, pp. 1–5, 2015.

[Wes08] Wessel R., Baranowski R., Klein R. Learning distinctive local object characteristics for 3d shape retrieval. In VMV, pp. 169–178, 2008.

[Wu15] Wu Z., Song S., Khosla A., Yu F., Zhang L., Tang X., Xiao J. 3d shapenets: A deep representation for volumetric shapes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1912–1920, 2015.

[Yum15] Yumer M. E., Chaudhuri S., Hodgins J. K., Kara L. B. Semantic shape editing using deformation handles. ACM Transactions on Graphics 34, No. 4, pp. 86:1–86:12, 2015.

[Zha01] Zhang C., Chen T. Efficient feature extraction for 2d/3d objects in mesh representation. In Image Processing, 2001. Proceedings. 2001 International Conference on, vol. 3, IEEE, pp. 935–938, 2001.

 ISBN 978-80-86943-40-4