Generative Design applied to Cloud Modeling

Carlos Eduardo Vaisman Muniz
Institute of Computing
Federal Fluminense University
Niterói, Brazil
carloseduardomuniz@id.uff.br

Wagner Luiz Oliveira dos Santos
Institute of Computing
Federal Fluminense University
Niterói, Brazil
wagner_luiz@id.uff.br

***Abstract*—Geometric modeling has recently leveraged the power of Generative Design. Generative Design can be seen as a framework in which models can be generated by systematically exploring a space of shapes generated by recombination of parametric shape descriptors. In this work, we explore the use of Generative Design for modeling 3D clouds. Clouds are usually modeled in Computer Graphics as voxelized models by using a combination of implicit and procedural techniques. Hence, they are described by a large number of parameters. Although these parameters usually include parameters that define a combination of scalar fields, noise functions and affine transforms, the controlled use of such parameters is rather complex. How to tune them up to obtain a plausible result is not obvious. We propose a method based on generative design combined with Machine Learning to produce families of cloud shapes automatically. Our generative design method is based on an evolutionary approach that generates instances of plausible 3D cloud shapes by optimizing a fitness function that measures the likelihood of a shape be a cloud. As the manual design of a fitness function is also quite complex, we propose using a Convolutional Neural Network to learn the fitness of arbitrary 2D views of the generated clouds. We perform several experiments that confirm the viability of the proposed method compared to manually modeled clouds.**

***Index Terms*—Cloud Modeling, Generative Design, Procedural Modeling, Machine Learning, Convolutional Neural Networks**

I. Introduction

Modeling natural objects is a hard task because the processes that guide their formation are extremely complex and sometimes not completely known. The most accurate way to model natural objects is to simulate the underlying physical and biological mechanisms of creation. Nevertheless, in many cases, the simulation of such processes is overly expensive in computational terms. Hence the need for alternative ways of computationally modeling natural objects.

Clouds are one of such objects. They are a visible group of small droplets of liquid combined with other particles suspended in the air. Many computer graphics, extended reality, and games include clouds as part of the rendered scene for realistic experiences. Thus, it makes cloud modeling and visualization one of the most relevant topics in these areas.

Although some works deal with physically based 3D cloud modeling, in Computer Graphics and affine areas, clouds are modeled using alternative techniques. The first ones used impostors, that is, small images of real clouds or synthetic 2D clouds oriented to the observer as a way of cloud representation. Later, more geometrically inspired techniques became more popular. Among such techniques, we find the use of procedural-based methods.

Procedural-based 3D cloud modeling approaches model a cloud by combining two different processes. The first one defines an overall cloud shape via an implicit density field function limiting its occupancy volume. In the second process, this initial shape is then deformed or altered by adding different noise levels. These processes use many parameters to define the shape of the cloud, such as the number of primitives, the primitive function used, the position, orientation, scale of each primitive, and all the parameters that rule the noise addition. Specifying all such parameters for controlling the cloud generation process is at the least a very cumbersome task. This problem scales when there is the requirement of generating many different shapes.

To face the problem of generating multiples cloud instances via procedural modeling, we propose using Generative Design techniques. Generative Design is an umbrella term that includes many methods to facilitate many aspects of design: innovation and diversification by efficiently exploring the design space, design optimization, cost reduction, computer-automated design, etc. Such techniques have been successful in helping designers find solutions for design problems in the areas of Computer-Aided Design, urban layout, architecture, manufacture, and others.

The main contribution of this paper is a Generative Design approach for 3D Cloud Modeling. We propose the use of an evolutionary mechanism that explores the space of procedural cloud specification parameters to produce a set of cloud instances that are plausible and diverse. The evolutionary procedure consists of a Genetic Algorithm that creates a cloud population evolution guided by a fitness function. The fitness function measures how credible is a 3D voxel model of a cloud-based on a sample 2D picture taken from an arbitrary point of view.

One of the challenges of our solution was to define what is a plausible cloud. Some works in the literature avoid dealing with this question by proposing methods that directly model clouds from real pictures. To solve this problem, we use a machine learning (ML) approach that learns what seems to be a plausible model from real cloud pictures and a not credible candidate from synthetic clouds previously labeled. The ML approach consists of a Convolutional Neural Network that receives synthetic images from 3D cloud models and returns a probabilistic measure of its plausibility to be a cloud.

Experiments were performed to show that the method can automatically select an appropriate set of parameters that produce a diversified and plausible set of 3D cloud instances. The results were compared with fine-tuned models created by humans by visual inspection. We could not perceive any abnormalities between the results produced by a human modeler and the automatic generation method proposed here.

This paper is organized as follows: Section 2 presents a brief background on the cloud modeling techniques used here, generative design concepts, and CNN concepts. Section 3 describes some of the works found in the literature that are related to ours. Section 4 presents the proposed method. Section 5 describes the experiments that were performed and their analysis. Finally, in section 6, some final considerations about our work are presented.

II. Background

A. Cloud Modeling

There are different ways to model clouds aiming at games, virtual reality, and computer graphics applications. Two of the most common approaches include Physically Based Modeling and Procedural Modeling. The former produces the most convincing results, whereas the latter is the most used because of its effectiveness. Recently hardware started to perform real-time physically-based modeling and animation. However, procedural approaches are still relevant because the computational effort is still quite cumbersome, considering that GPUs and CPUs still have many other tasks to perform. In this work, we will focus on Procedural Modeling, closely following the work from [1] for generating cumulus clouds and replicating some of its concepts here for comprehensiveness.

A cloud is represented as a volumetric shape $V \in R^3$ where a density value is calculated by a function $\rho(p)$ for each $p \in V$. To define the general aspect of the cloud, we use an implicit density field $g(nd_i)$, combined with a noise function $n(p)$ that defines its details. The parameter ndx_i is a normalized distance to a primitive that could be a simple point or a skeleton primitive (a poly-line or polygon chain). In the case of a point primitive, the distance function can be computed as $nd_i = d_i/l_i$ where d_i is the distance of a given point $p \in R^3$ to the primitive and l_i is the size of the region of influence (see Figure 1).

The implicit density field $g(nd_i)$ is defined as a combination of the field functions $g(nd_i)$ of each of the n primitives. The result of the field function is 1 if p is inside the primitive or the implicit primitives $b(nd_i)$ given by the Wyvill-1 function [2] defined in the equation 1 if p is its region of influence:

$$b_i(nd_i) = -\frac{4}{9}nd_i^6 + \frac{17}{9}nd_i^4 - \frac{22}{9}nd_i^2 + 1, 0 \leq nd_i \leq 1 \quad (1)$$

The combination based on the Set Theory proposed by [3] produces a coarse density field $c\rho(p)$ expressed as the following recursive equation:

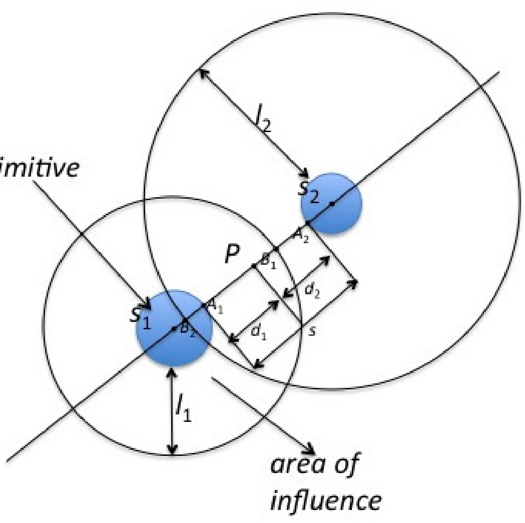

Fig. 1. Field function. Picture drawn based on [3].

$$\begin{aligned} c\rho_1(p) &= g_1(nd_i) \\ c\rho_i(p) &= c\rho_{i-1}(p) + g_i(nd_i) - \frac{1}{w}c\rho_{i-1}(p)g_i(nd_i), \\ i &= 2, \ldots, n, \\ w &= w_1 = w_2 = \cdots = w_n \end{aligned} \quad (2)$$

The final step is to disturb the overall shape by adding a noise function defined at each point in the space. The noise function that produced the best results was the *Turbulence Function* proposed by Ken Perlin [4].It can be controlled via four parameters: *amplitude* (a), *frequency* (fr), *gain* (ga) and *lacunarity* (lc). Lacunarity and gain define, respectively, the rate of change of the frequency and amplitude of the noise per octave, regulating the fractal behavior of the noise.

$$n(p) = \sum_i a \left| \frac{(lc_i) * fr * p}{ga_i * a} \right| \quad (3)$$

The noise function is defined as the absolute value of $N(p) = fBm3D(x, y, z, n, fr, a, la, ga)$, where x, y, z are the coordinates of a given point primitive p, n is the number of octaves and fr, a, la and ga are the turbulence parameters. Taking the modulus of fBm3D is appropriate for generating *cummuliform* clouds [5].

There are two ways to apply noise on top of the computed density fields. The first one is to apply noise the combined coarse noise field as shown in equation 4. The second is applying different noise configurations to each primitive individually as seen in equation 5. In equation (5), we set $g'(x_i) = N(g(x_i))$ so that the combination is done on the primitives to which noise is added.

$$\bar{\rho}(p) = (1 - np) \cdot \rho(p) + (np) \cdot \rho(p) \cdot N(p) \quad (4)$$

$$
\begin{aligned}
g'(nd_i) &= N(g(nd_i)) \\
\rho_1(p) &= g'_1(nd_i) \\
\rho_i(p) = \rho_{i-1}(p) + g'_i(nd_i) &- \frac{1}{w}\rho_{i-1}(p)g'_i(nd_i), \\
i &= 2, \dots, n,
\end{aligned}
$$

$$
\text{w=w}_1 = w_2 = \cdots = w_n \tag{5}
$$

The second approach, used in this work, allows the creation of details in different scales, amplitudes, and frequencies at each local part of the model under the influence of a given primitive.

Our cloud modeling tool uses the techniques described on the algorithms 1 and 2 to calculate the density from a voxel. Algorithm 1 blends the noise function on the combined implicit density field, while Algorithm 2 does it per primitive, allowing further. More details about the method associated to both algorithms, including paralelization and differentiability discussion can be seen in [1].

```
Data: Set of primitives P
la - lacunarity, fr - frequency, a - amplitude, ga - gain,
s - scale, n - numOctaves, p - noise percentage
width, height, depth - dimensions of the volume array
Result: Density volume V
  for $V_{ijk} \in V$ do
    $V_{ijk} \leftarrow 0$
    $\rho' \leftarrow 0$
    $x, y, z \leftarrow i + rand()(s), j + rand()(s), k + rand()(s$
    for $h = 0$ to $|P|$ do
      $nd_h \leftarrow distance(i, j, k, P_h.x, P_h.y, P_h.z)$
      if $nd_h < P_h.radius$ then
        $\rho' \leftarrow \rho' + 1$
      else if $nd_h < P_h.radius + P_h.influence$ then
        $dist \leftarrow (nd_h - P_h.radius)/P_h.influence$
        $g \leftarrow Wyvill(dist)$
        $\rho' \leftarrow \rho + g - (1/P_h.weight)(\rho)g$
      end if
    end for
    $noise \leftarrow fBm3D(x, y, z, n, fr, a, la, ga)$
    $V_{i,j,k} \leftarrow max((1 - n)\rho' + (n * noise * \rho'), 1)$
  end for
```
Algorithm 1: Algorithm 1

B. Generative Design

According to [6], Generative Design (GD) is a label encompasses a group of techniques that enables designers to efficiently explore the conceptual design space, automate design generation, achieve design cost reduction, optimization, accuracy and consistency. Generative Design has applications in many areas including architecture, manufacturing, CAD, structural optimization, human computer interface design and etc.

```
Data: Set of primitives P
la - lacunarity, fr - frequency, a - amplitude, ga - gain,
s - scale, n - numOctaves, p - noise percentage
width, height, depth - dimensions of the volume array
Result: Density volume V
  for $V_{ijk} \in V$ do
    $V_{ijk} \leftarrow 0$
    $\rho' \leftarrow 0$
    $x, y, z \leftarrow i + rand()(s), j + rand()(s), k + rand()(s$
    for $h = 0$ to $|P|$ do
      $nd_h \leftarrow distance(i, j, k, P_h.x, P_h.y, P_h.z)$
      if $nd_h < P_h.radius$ then
        $noise \leftarrow$
        $fBm3D(x, y, z, P_h.n, P_h.fr, P_h.a, P_h.la, P_h.ga$
        $g \leftarrow (1 - P_h.n) + P_h.n * noise$
        $\rho' \leftarrow \rho' + g - (1/P_h.w) * g * \rho'$
      else if $nd_h < P_h.radius + P_h.influence$ then
        $noise \leftarrow$
        $fBm3D(x, y, z, P_h.n, P_h.fr, P_h.a, P_h.la, P_h.ga$
        $dist \leftarrow (nd_h - P_h.radius)/P_h.influence$
        $g \leftarrow Wyvill(dist)$
        $g \leftarrow (1 - P_h.n) * g + P_h.n * noise * g$
        $\rho' \leftarrow \rho' + g - (1/P_h.w) * g * \rho'$
      end if
    end for
    $V_{i,j,k} \leftarrow max(\rho', 1)$
  end for
```
Algorithm 2: Algorithm 2

There are many techniques that support computational generative design. The most basic ones [7] include shape grammars (SG) [8], L-systems (LS) [9], cellular automata (CA) [10] [12], genetic algorithms (GA) [11], and swarm intelligence (SI) [13]. Such techniques have different intents and can be used for:

- search space exploration and design optimization - GA
- shape and style generation - SG and LS
- context-sensitive and behaviour-driven design processes - CA
- form driven design process - SG and LS
- design by self-organization - CA and SI
- evaluation of design usability - SI

Many design problems are solved using GD by combining such techniques as integrated frameworks. An comprehensive description of each of these techniques is out of the scope of this paper. Nevertheless, we describe in a few more details, in section V the use o GAs in Generative Design which was the approach used to solve part of the problem of cloud modeling here presented.

C. Deep Image Classification

According to [14], Image classification, which can be defined as the task of categorizing images into one among many classes predefined. Although it can be considered second nature for humans, it is much more challenging for an automated system. It forms the basis for other computer vision tasks such as localization, detection, segmentation, and generative models.

Convolutional Neural Networks (CNNs) [15] have drastically changed the computer vision landscape by considerably improving the performance on most image benchmarks. A key characteristic of CNNs is that the deep(-based) representation, used to perform the classification, is generated from the data rather than being engineered.

This two-stage approach is traditionally applied to classification or location problems. In the first step, the initial layers of the network perform the feature extraction process. In the second step, the final layers perform the classification or location step according to the nature of the problem.

III. Related Works

Many works have investigated the problem of cloud modeling. Some use more computationally inspired techniques as procedural modeling and cellular automata whereas others are more inclined to the use of physically based approaches.

The first works usually modeled clouds using impostors, that is, a set of images, carefully positioned in the scene space that are able to track the observer position giving an illusion of a tridimensional perception. One of the most representative works in this class is the work of Harris et al. [17] which is focused in the rendering aspects of clouds in games and computer graphics in general.

Ebert et al. was one of the first to investigate how to model steam, smoke, clouds and similar phenomena [18]. Similarly to the way we generate cloud instances in this work, his approach is also based on a procedural modeling where implicit functions define the overall shape that is modified by noise.

Later, Schpok et al. [5] modified Ebert's techniques so that it could be used more efficiently by leveraging the power of programmable graphics hardware. He proposed a complete realtime system for cloud modeling and animation.

Lipuš et al. [3] analysed the behaviour of implicit function blending and verifyed that its direct summation produced artificial effects in the cloud density values. They proposed a different way of blending implicit primitives based on set Theory. The work in [1] extended this technique by enabling it add different noise functions on different primitives before blending.

Man et al. [19] proposed a work that represents clouds a set of metaballs that approximates the original density function of a cloud build with procedural noise. The metaballs parameters are determined using a radial basis function real neural network.

One important example of cellular automata applied to cloud modeling is the work of Dobashi et al. [21] aimed at cloud animation. In Dobashi's work, the dynamics of the fluid is described by transition rules guide define the simulation of complex motion inside the clouds. Jiangbin Xu et al. also proposes the use of probabilistic cellular automata [23]. In his work, probability fields define a fractal Brownian motion function that guides the movements of the particles in the cloud.

Physically based cloud modeling is a very complex task. Dobashi et al. in 2008 investigated how to control the parameters of a fluid dynamic model to produce cummuliform clouds [22]. His work also uses a first step where the overall shape of the cloud is defined so that later the parameters can be adjusted to give the cloud its final visual aspect conforming to the predefined shape.

Dobashi et al. proposed a methods for cloud modeling based on photos in [20] . In such works, cloud modeling is formulated as an inverse problem. The inout photo of a cloud serves as a guide to an optimization method estimate the parameters that define a non-uniform density function.

IV. Problem Definition

The problem we solve in this paper is how to compute a family of cloud models $\mathcal{C}$ which are consistent with what we understand as a shape and texture of a cloud using procedural modeling. Besides, we wish that $\mathcal{C}$ presents a high level of variability.

Our hypothesis is that by considering this problem as a shape design problem and using Generative Design techniques we are able to:

- effectively search the cloud shape space aiming at finding optimum shape alternatives
- be able to create a high level of variability in the set of shape variants

V. Proposed Method

The methodology used to solve the problem previously defined uses a Generative Design pipeline that combines procedural modeling, GAs and CNNs.

A. Method overview

The procedural modeling it the part of the method that produces instances of volumetric volumes from parametric specifications. In part we could understand that it plays the role of Shape Grammars and L-Systems in classical Generative Design approaches. The GA proposed here is the module that searches the shape space aiming at finding good cloud shape alternatives. The measure of how "good" a cloud is considered is captured by a fitness function used by the GA optimizer. Instead of devising such fitness function manually, we use a CNN to learn it from training examples including previously tagged synthetic models and real cloud pictures.

B. Cloud instance generation

The generation of instances is done using the method described in section Cloud Modeling. Each cloud instance is characterized by a set of parameters defining its set of density field primitives P_h, $h = 1 \ldots |P|$ and their associated additive noise parameters. Each primitive has its center (x, y, z), radius r, dimensions (sx, sy, sz), and weight w. Besides, for each primitive we define the corresponding noise parameters as defined in equation 3. In our implementation, all required

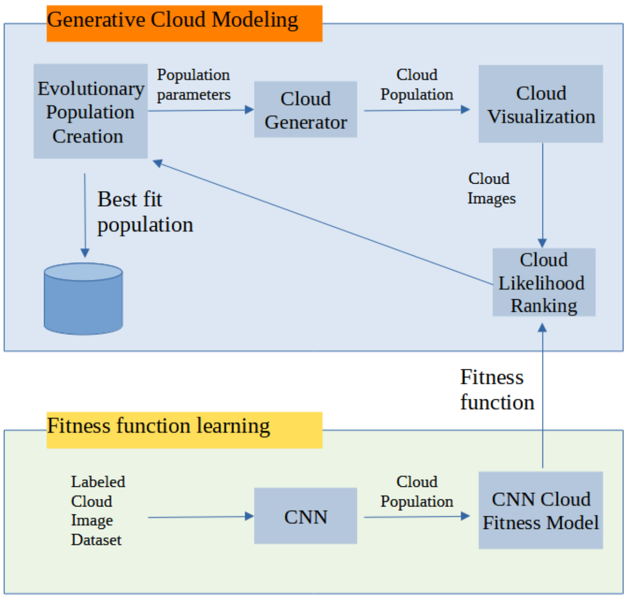

Fig. 2. Method overview

parameters of all primitives are grouped in an array properly indexed. The structure of such array is detailed in the next section where the evolution of a population of cloud instances according to evolutionary rules are presented.

C. *Cloud population evolution*

The evolutionary process works with a fixed population number of cloud candidates and a pre-determined number of epochs saved in a settings file. The initial cloud population is either mutation of the cloud that is being modeled by the user or a set of recorded clouds. At the end of each epoch, the screenshots of the clouds are sent to the CNN classifier. It returns the fitness value, which is the probability of each candidate be a cloud. The candidates are ordered according to the fitness value. Half of the population with the highest fitness value is kept untouched, while the other half is discarded and replaced by mutations or crossover of the surviving population. The whole process is displayed in figure 2. The optimal number of epochs depends on the quality of the input candidates and the expectations of the user of the cloud modeling tool.

The cloud parameters are stored in a one-dimensional array. The global parameters come on the first positions, and then the parameters of each primitive come next. Every parameter is stored as double, regardless of its data type. Such structure facilitates the process of mutating a cloud candidate since it allows it to change a random amount of parameters, choose the parameters randomly and randomly select the intensity of the change and its signal. In our evolutionary process, we have restricted a maximum of 10% of the parameters to be changed in a cloud candidate per epoch. We have also capped the intensity of each change to a maximum of 6 increments. There are maximum and minimum limits previously defined by the user for the shape space. It is possible to restrict these limits for specific contexts due to the varied behavior of the cloud parameters. Some parameters may make others behave inconsistently in different contexts. One example is the choice of the algorithm that calculates the density of the voxel, which may turn primitive frequency related parameters useless depending on which algorithm is chosen.

The crossover between two cloud candidates consists of randomly choosing, for each parameter, which of the two clouds will transfer its property. Due to its higher chance of convergence compared to mutations, we have prioritized mutations over crossovers, setting an 80% chance of mutations to happen and only 20% for crossovers.

D. *Cloud ranking*

Building a CNN model that aptly represents a cloud is a task that requires hundreds or thousands of images with the many forms of this object in all its natural variations. The main variations of the class or object are localization, viewpoint, colors, and scale.

The dataset used in this work is composed of positive and negative cloud images. They come from three different sources:

- 1) photos from real clouds,
- 2) synthetic clouds,
- 3) data augmentation techniques.

We have used a total of 2550 images, with a minimum resolution of 240x 240 pixels. Photos taken from clouds, which represented just over 10% of the database, were used in the set of real clouds. Most of them were taken from the ground or buildings by the authors, although some were adapted from public datasets such as [24], [26] and [25]. All the photos were hand edited to remove features that are not covered by the cloud generator, such as birds, sunlight, airplanes, artificially generated smokes, rainbow, and types of clouds that are far too different from cumulus clouds. A larger group of images created with our cloud generator has been included in the class of non-clouds. We were far more concerned at teaching the CNN what a cloud could not be since the probability of generating non-clouds with the cloud modeling tool is much higher than obtaining a good-looking cloud. Besides that, the approval of a bad-looking cloud could ruin the whole evolution process, while skipping a good looking cloud does not do the same damage. The validation images consist of 49 photos taken from clouds and additional 600 non-clouds created by the cloud modeling tool used in this work. We have also used extra 600 synthetic clouds that we perceived as feasible clouds to analyze the CNN's ability to classify synthetic clouds created by the cloud modeling tool.

The unbalanced dataset drove us to make a sampling adjustment of the classes within each batch, already applying some data augmentation techniques, artificially expanding the dataset, and further reducing overfitting on training. In this data pre-processing step, we use image flip operations, rotation transformations, and brightness and contrast intensity changes.

After testing some CNN architectures like AlexNet, VGG1, ResNet-50 [15] considering our cloud training database, we opted for the following architecture displayed on table I without performing the transfer learning operation since resources extracted from the cloud are very singular and some of them are too thin and do not resemble commonly used objects such as the ImageNet base [16].

TABLE I
CNN ARCHITECTURE

Layer	Shape
Input	[144X144X1]
Conv2D	138X138X64
ReLu	138X138X64
BatchNorm2D	138X138X64
MaxPool2D	69X69X64
Conv2D	65X65X128
ReLu	65X65X128
BatchNorm2D	65X65X128
MaxPool2D	32X32X128
Conv2D	30X30X128
ReLu	30X30X128
BatchNorm2D	30X30X128
MaxPool2D	15X15X128
Conv2D	13X13X64
ReLu	13X13X64
BatchNorm2D	13X13X64
MaxPool2D	6X6X64
Flatten	2304
Linear	1024
ReLu	1024
Dropout	1024
Linear	1024
ReLu	1024
Dropout	1024
Linear	1
Sigmoid	1

VI. RESULTS

In order to produce plausible clouds, our fitness function must be able to return a high probability when the candidate looks similar to a real cloud, and it must also be able to filter non-clouds. If the fitness function works, good synthetic clouds will not be replaced by bad-looking clouds. Thus the odds of generating the variations of good-looking clouds increases considerably.

We have conducted a couple of tests with our CNN, such as the k-fold cross-validation technique and a confusion matrix. Most of the other experiments had their results analyzed visually, and, therefore, the results will be shown as pictures.

To assess the generalizability of a model from a set of data, we use the k-fold cross-validation technique, which is widely used to estimate how accurate this model is in practice, that is, its performance for a new set of data, in special on a limited data sample or small. This procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. This work we choose k = 5.

To evaluate the efficiency of the test, the confusion matrix of the considered model is used, being applied over a set of 500 images. The confusion matrix on table II and table III contains the values true positive, true negative, false positive and false negative. The highest diagonal values in the confusion matrix show the model's accurate predictions.

TABLE II
CONFUSION MATRIX

	Non-Cloud (prediction)	Cloud (prediction)
Non-Cloud (real)	305	0
Cloud (real)	0	49

TABLE III
CONFUSION MATRIX

	Non-Cloud (prediction)	Cloud (prediction)
Non-Cloud (synthetic)	601	0
Cloud (synthetic)	355	282

The classifier was able to detect all true clouds. However, it is confused with some synthetic clouds. This behavior was somewhat expected since the synthetic clouds might have different features, lighting details, contrast, and resolution differences. It is also subject to our concept of what a cloud is. But since we want the genetic algorithm to generate clouds close to the real ones, in this context, it is less problematic to wrong classifying what we perceive as being a cloud than the other way around.

The table IV shows the fitness scores of 10 candidates on the figure 3, where half are considered clouds and the other half non-clouds. It is not trivial to pickup the details that makes some of these candidates be more cloud than others.It's a matter of shape, illumination, texture and bad use of Perlin noise that may affect the classification.

TABLE IV
FITNESS SCORE

Cloud	Score	Prediction
Candidate (0, 0)	0.104416	Non-cloud
Candidate (1, 0)	0.468740	Non-cloud
Candidate (2, 0)	0.019677	Non-cloud
Candidate (0, 1)	0.010033	Non-cloud
Candidate (1, 1)	0.049599	Non-cloud
Candidate (2, 1)	0.999962	Cloud
Candidate (0, 2)	0.999385	Cloud
Candidate (1, 2)	0.974813	Cloud
Candidate (2, 2)	0.720757	Cloud
Candidate (2, 3)	0.577683	Cloud

Finally, the figure 4 shows what kind of clouds you can get after 15 epochs using the cloud candidate in the top-left corner as input. The clouds in the first two lines uses the settings described on section V-C. The last three candidates is what happens when we allow stronger mutations.

Further results and resources used by our work will be shared at [27].

VII. CONCLUSION AND FUTURE WORKS

This work proposes using an evolutionary mechanism to explore the space of parameters for the Generative Design

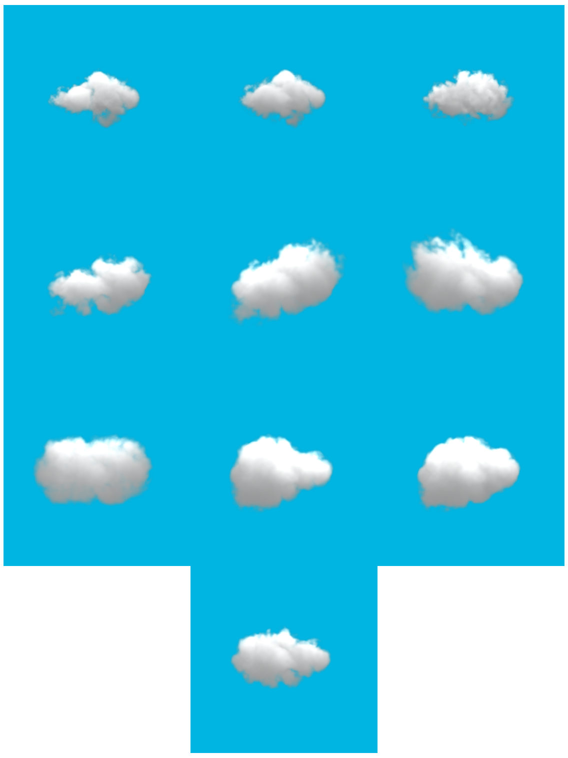
Fig. 3. Cloud Candidates for table IV

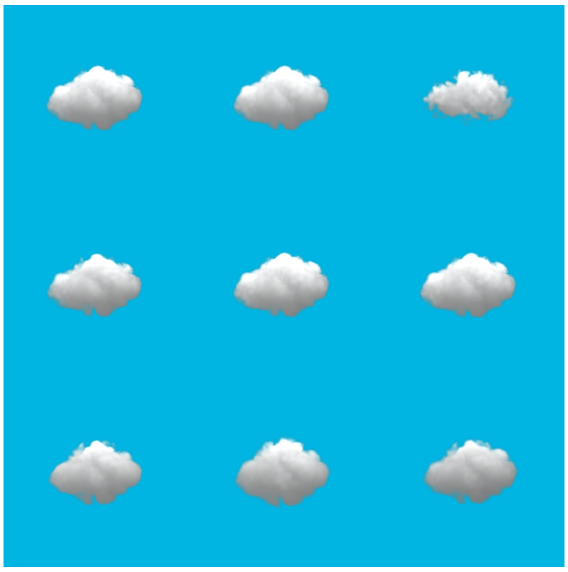
Fig. 4. Clouds obtained after 15 epochs

approach for procedural Cloud Modeling. It uses a Convolutional Neural Network to calculate the fitness function of the evolutionary technique. It generates several different clouds that are similar to the one that is being designed by the user and that have strong possibilities of look like a cloud. The proposed technique is a brute-force mutation-based approach whose effectiveness relies on the accuracy of the fitness function.

The CNN trained for this work is more effective at rejecting bad-looking clouds than detecting good clouds. We have prioritized the rejection of bad-looking clouds, because the strictness of the CNN towards cloud candidates reduces the chances of the evolutionary mechanism to allow bad clouds to replace good ones. Although it is not bad at the detecting realistic clouds, it may occasionally reject clouds that look good for humans. One of the reasons is that the training dataset is unbalanced towards non-clouds training images, even with the sampling adjustment done at each batch. Obtaining synthetic images is an easier task than taking pictures from real-life clouds. Some aspects of the synthetic images generated by the cloud modeling tool differ from real clouds.

The evolutionary mechanism is calibrated by several parameters that could be relaxed to allow stronger changes to the cloud mutations, allowing a larger variation of clouds to be built with the proposed method. However, it requires a better trained CNN.

As future works, we see a lot of room to better calibrate the existing solution from both CNN and evolutionary perspectives. Or even use the evolutionary tool to calibrate other parts of the modeling tool such as the rendering algorithm, illumination settings, etc.

Acknowledgment

We thank Anselmo Antunes Montenegro for providing the synthetic cloud generator used in his work [1], for helping to improve its rendering code, and for supporting the development of our work. We also thank Adriana Ossaille Beltrame for many of the real cloud pictures used on our CNN.

References

[1] A. Montenegro, Í. Baptista, B. Dembogurski and E. Clua, ”A New Method for Modeling Clouds Combining Procedural and Implicit Models,” In *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2017, pages 173-182, doi: 10.1109/SBGames.2017.00027.

[2] B. Wyvill and G. Wyvill. Field functions for implicit surfaces. In *New Trends in Computer Graphics*, pages 328-338. Springer, 1988.

[3] B. Lipuš and N. Guid. A new implicit blending technique for volumetric modelling. *The Visual Computer*, 21(1-2):83–91, 2005.

[4] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

[5] J. Schpok, J. Simons, D. S. Ebert, and C. Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 160–166. Eurographics Association, 2003.

[6] S. Khan, M. J. Awan. A generative design technique for exploring shape variations, In *Advanced Engineering Informatics*, Volume 38, 2018, pages 712-724, ISSN 1474-0346, https://doi.org/10.1016/j.aei.2018.10.005.

[7] V. Singh and N. Gu. Towards an integrated generative design framework, In *Design Studies*, Volume 33, Issue 2, 2012, pages 185-207, ISSN 0142-694X, https://doi.org/10.1016/j.destud.2011.06.001.

[8] G. Stiny and J. Gips. Shape Grammars and the Generative Specification of Painting and Sculpture. In *IFIP Congress*. pages 71. 1460-1465, 1971.

[9] A. Lindenmayer. Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs, In *Journal of Theoretical Biology*, Volume 18, Issue 3, 1968, pages 300-315,ISSN 0022-5193, https://doi.org/10.1016/0022-5193(68)90080-5.

[10] J. Von Neumann. The General and Logical Theory of Automata. In: *Jeffress*, L.A., Ed., Cerebral Mechanisms in Behavior: The Hixon Symposium, John Wiley & Sons, New York, 1-41, 1951.

[11] Holland, J. H. (1992). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press.

[12] S. Wolfram and A. Mallinckrodt. Cellular Automata And Complexity. In *Computers in Physics 9*, page 55, 1995, doi: 10.1063/1.4823369

[13] J. L. Deneubourg. Application de l'ordre par fluctuations á la descriptions de certaines étapes de la construction du nid chez les termites. In *Insect. Soc. 24*, pages 117-139, 1977.

[14] W. Rawat, Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. In *Neural computation*, v. 29, n. 9, pages 2352-2449, 2017.

[15] N. Van Noord, E. Postma. Learning scale-variant and scale-invariant features for deep image classification. In *Pattern Recognition*, v. 61, pages 583-592, 2017.

[16] A. Krizhevsky, I. Sutskever, G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, v. 25, pages 1097-1105, 2012.

[17] M. J. Harris and A. Lastra. Real-time cloud rendering for games. In Proceedings of Game Developers Conference, pages 21–29, 2002.

[18] D. S. Ebert. Texturing & modeling: a procedural approach. Morgan Kaufmann, 2003.

[19] P. Man. Generating and real-time rendering of clouds. In *Central European seminar on computer graphics*, pages 1–9. Citeseer, 2006.

[20] Y. Dobashi,W. Iwasaki, A. Ono, T. Yamamoto, Y. Yue, and T. Nishita. An inverse problem approach for automatically adjusting the parameters for rendering clouds using photographs. In *ACM Trans. Graph.*, 31(6):145, 2012.

[21] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 19–28. ACM Press/Addison-Wesley Publishing Co., 2000.

[22] Y. Dobashi, K. Kusumoto, T. Nishita, and T. Yamamoto. Feedback control of cumuliform cloud formation based on computational fluid dynamics. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3), 2008.

[23] J. Xu, C. Yang, J. Zhao, and L. Wu. Fast modeling of realistic clouds. In Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on, pages 1–4. IEEE, 2009.

[24] ”Cloud image classification Dataset — Kaggle”. https://www.kaggle.com/nakendraprasathk/cloud-image-classification-dataset. (accessed Oct 4th, 2021)

[25] ”Public Domain Pictures, 2021”. https://www.publicdomainpictures.net/. (accessed Oct 4th, 2021)

[26] ”Cumulus — Wikipedia a Free Encyclopedia”. https://en.wikipedia.org/wiki/Cumulus_cloud. (accessed Oct 4th, 2021)

[27] ”Generative Design Applied to Cloud Modeling”. https://cloudsbgames2021.carloseduardomuniz.com/. (accessed Oct 4th, 2021)