Exploring Generative 3D Shapes Using Autoencoder Networks

Nobuyuki Umetani
Autodesk Research, Toronto, Canada

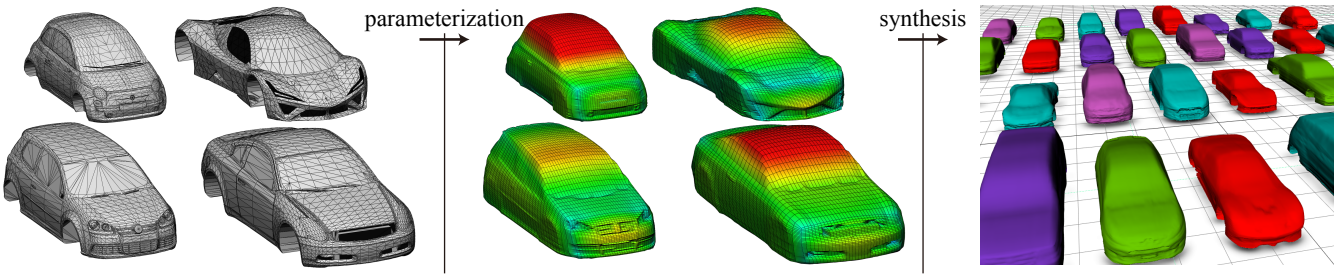


Figure 1: From unstructured triangle mesh (*left*), our approach can efficiently and robustly construct a quad mesh with a consistent topology (*middle*) that is compactly parameterized as a height map (shown in color contour). The autoencoder constructs a low-dimensional representation of the set of shapes to synthesize new shapes (*right*). Our interface allows the user to interactively guide the synthesis by directly manipulating on the shapes.

ABSTRACT

We propose a new algorithm for converting unstructured triangle meshes into ones with a consistent topology for machine learning applications. We combine the orthogonal depth map computation and the shrink wrapping approach to efficiently and robustly parameterize the triangle geometry regardless of imperfections such as inverted faces, holes, and self-intersections. The converted mesh is consistently and compactly parameterized and thus is suitable for machine learning. We use an autoencoder network to extract the manifold of shapes in the same category to explore and synthesize a variety of shapes. Furthermore, we introduce a direct manipulation interface to navigate the synthesis. We demonstrate our approach with over one thousand car shapes represented in unstructured triangle meshes.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Shape modeling**; • **Applied computing** → **Computer-aided design**;

KEYWORDS

machine learning 3D shapes, interactive shape exploration

ACM Reference Format:
Nobuyuki Umetani . 2017. Exploring Generative 3D Shapes Using Autoencoder Networks. In *Proceedings of SIGGRAPH Asia Technical Brief, Bangkok, Thailand, November 2017,* 4 pages.
https://doi.org/10.1145/3145749.3145758

SIGGRAPH Asia Technical Brief, November 2017, Bangkok, Thailand

ACM ISBN 0730-0301/2017/November…$15.00
https://doi.org/10.1145/3145749.3145758

1 INTRODUCTION

Recent advances in machine learning have seen the introduction of various applications such as classification, style transferring and generation, which target media such as images and audio. However, 3D shapes have not gained full benefit from machine learning, despite the vast number of 3D shapes now available on the internet. This is mainly because the machine learning algorithms require the consistent representation of input and output data such as an orthogonally aligned grid (i.e., pixels in the images). Unstructured triangle meshes are the most popular surface representation in the computer graphics, but their topological structures are usually different from one another, hindering the use of machine learning.

In this paper, we present a new parameterization technique for efficiently converting a given unstructured mesh into one with a manifold mesh with consistent connectivity using depth information. Our parameterization is robust to deficiencies such as holes, gaps, and inverted triangles. We achieve compact and explicit parameterization of a 3D shape by representing the shape as a *hight field*, which is elevated from the subdivision of a simple primitive polygon. We demonstrate the robustness of our approach by the parameterization of over one thousand car shapes.

The main benefits of our parameterization are the generation of input and output data that is ready for machine learning (Figure 1-middle). From many shapes in the same category, our autoencoder network constructs a manifold of these shapes. Using the low dimensional representation from the autoencoder, we can generate and explore a variation of the 3D shapes at the interactive rate (see Figure 1-right). We also present an interface to interactively manipulate the 3D shape synthesis result, allowing the user to directly specify the location of a vertex of a generated shape. Our contributions are summarized as follows:

- A compact and efficient parameterization of 3D shapes.
- An autoencoder to construct a manifold of 3D shapes.
- A direct manipulation interface to explore generative shapes.

2 PARAMETERIZATION OF 3D SHAPES

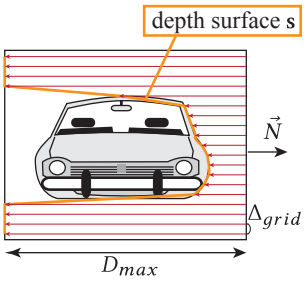
The neural network, which is one of the most common approaches in machine learning, constructs a map that approximates the input/output relationship in the training data. While the neural network can approximate highly nonlinear functions, their input and output dimensions are usually predetermined, and thus we need to represent the training data via fixed-sized *feature vectors*. In computer graphics, 3D geometries are often available as *polygon soup* which are usually non-manifold and un-oriented triangle meshes which may contain self-intersections. It is very challenging to construct a consistent representation of such unstructured data.

As deep learning became a hot topic in computer science, many graphics researchers started to study 3D shape representation for neural networks. We refer to the survey [Masci et al. 2016] for various previous studies. Voxel approaches consume a lot of memory space and thus it is difficult to increase resolution. The multi-view projection approaches represent a shape with projected images taken from multiple viewpoints. However, it is difficult to use this implicit 2D parameterization to synthesize 3D shapes because generated 2D images may contradict each other. Approaches based on descriptors extracted from 3D geometry do not usually contain complete information of the original shapes and thus it is also difficult to synthesize 3D shapes.

Our parameterization constructs a quad mesh with consistent topology to explicitly represent 3D shapes (Sec. 2.2). The quad mesh is efficiently computed from the depth map from the multi-view projection (Sec. 2.1). We are interested in machine learning exterior shapes that are genus zero and not very concave (no point is occluded completely) and shaped similarly to the cuboid in the coarsest resolution such that the depth map contains enough information on the 3D surface.

2.1 Depth Map

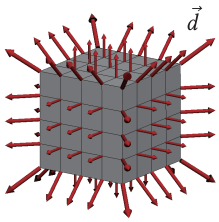
First, we set up a bounding box that encloses all the training shapes. Then, we divide a face of the bounding box to construct a Cartesian grid. From each center of the grid cell, we shoot a ray in the inward direction $-\vec{N}$, where $\vec{N}$ is the normal of the bounding box face. For all the grid cells, we record the depth, i.e., the distance the rays traveled before intersecting any of the triangles in the object. Since the training shapes are always inside bounding box, the depth takes value in the range $(0, D_{max}]$, where the D_{max} is the maximum depth of the bounding box for the grids whose ray does not intersect with the shape. For the car shapes, we use the bounding box that has the dimensions 2 m×2 m×6 m. Each face of the bounding box is divided in the resolution where the grid size Δ_{grid} is 1 cm. We used a pixel shader to efficiently compute the depth map with 4-byte accuracy. Hereafter, we call the surface prescribed by the projected cell-centered points as *depth surface* **s**.



2.1.1 relaxation. The general non-manifold triangle meshes often have many holes (e.g., the gap between the doors, the air intake of the front grill). If a ray goes through such holes, it appears as a sharp spike in the depth surface. Such high-frequency artifacts often undermine the machine learning by making it focus on the noise. Hence, we filter out such noise if it exceeds a certain curvature threshold. We first extract the silhouette of the projection by marking the outside grids by flood-filling grids with the maximum depth D_{max}. Then, for the cell inside the silhouette and not adjacent to its boundary, we compute the Laplacian of the depth using the central differential scheme. If the magnitude of the Laplacian at a cell is larger than $\alpha D_{max}/\Delta_{grid}^2$, we apply the Laplacian smoothing for that cell. Note the α (here we use $\alpha = 0.1$) is the threshold factor indicating the magnitude of the Laplacian comparing its maximum possible value. We iterate this relaxation process several times until reaching convergence. Note that this operation converges quickly as it does not change the silhouette and only reduces the magnitude of the Laplacian.

2.2 Shrink Wrapping Parameterization

We propose to use the shrink wrapping approach [Kobbelt et al. 1999] to consistently parameterize the 3D shapes for machine learning. Shrink wrapping is a technique used to construct a subdivision connectivity mesh by projecting the vertices onto the target mesh while iteratively subdividing a coarse base mesh. In this paper, we use a cube as the base mesh because most of the cars have a box-like geometry in a coarse resolution. In short, how the subdivided cube is deformed to fit the target 3D shape gives the parameterization of the shape.



2.2.1 projection directions. We predefine the projection direction $\vec{d}$ for each subdividing vertex of the cube such that only the projection height h determines the positions of the vertices. This constant direction projection instead of variable directions helps to reduce the number of parameters to encode the movement of a vertex from three variables (XYZ displacement) to a single variable (scalar height). Given an axis-aligned cube, we choose projection directions for each subdivision vertex by adding up the normals of the faces to which the vertex belongs (see the inset figure). For example, the projection direction is $\vec{d} = (0, 0, +1)$ if the vertex is inside a face with +z normal. If the vertex is on the edge between the faces with normal -x and +z, the projection direction would be $\vec{d} = (-1, 0, +1)$. If the vertex is at the corner surrounded by the three faces with normals +x, -y, and -z, the projection direction will be $\vec{d} = (+1, -1, -1)$. The integer values of the XYZ components of the projection direction help to accelerate the computation of the following ray intersection method.

2.2.2 corner points. Given the set of depth surfaces $\mathcal{S}$ for an object, we first move the eight corner vertices of the base cube onto the object's surface. The corner vertex should be placed such that the base cube approximates the object as much as possible. Let a corner vertex have a projection direction $\vec{d}$; we simply place the vertex where the depth surface **s** is tangent to the plane with normal $\vec{d}$ (see Fig. 2-top left). We ignore the depth surface $\mathbf{s} \in \mathcal{S}$ if its depth projection direction is opposite to the vertex projection

direction $\vec{N} \cdot \vec{d} < 0$ because the tangent point is usually not on the object surface. For the rest of the tangent points, we compute the average to determine the final corner points location. This corner vertex placement is somewhat heuristic, but gives a desirable result for an approximately convex shape such as that of most cars.

2.2.3 ray intersection. In each subdivision iteration, we subdivide the cube mesh in half by adding vertices at the center of the edges and quad faces (see Fig. 2). From each newly added vertex, we shoot a ray in the direction $\vec{d}$ to find the first intersection point $\mathbf{p}_s$ against all the depth surfaces $\mathbf{s} \in \mathcal{S}$. If the ray does not intersect with a depth surface, we shoot a ray in an opposite direction $-\vec{d}$. For the intersection points, we then compute their average with the weight w_s as

$$\mathbf{p} = \sum_{\mathbf{s}\in\mathcal{S}} w_s \mathbf{p}_s / \sum w_s, \quad w_s = \vec{N} \cdot \vec{n}_s, \tag{1}$$

where $\vec{n}_s$ is the the normal of the depth surface at the intersection point. We trust the intersection point where the normal $\vec{n}_s$ is similar to the projection direction $\vec{N}$ because the depth surface better represents the object's surface and it is less affected by the relaxation step. Again we ignore the intersection point if the depth projection direction faces opposite to the ray direction as $\vec{d} \cdot \vec{N} < 0$. Since the XYZ components of the ray direction $\vec{d}$ are either 0,+1 or -1, we can efficiently find the set of grid cells where the ray may intersect by traveling the adjacent cells.

2.2.4 projection height. We denote δh the projection height, i.e., how far the original subdivision point is projected to reach the intersection point $\mathbf{p}$ in the direction of the ray $\vec{d}$. For visualization purposes, we compute the total height h for each subdivision step by adding its projection height δh to the averaged total heights of the points of the subdivided element (edge or quad). The total heights visualize how much in total the subdivision points are lifted from the base mesh. Fig. 2-middle shows the color contour visualization of the total height at different levels of the subdivision.

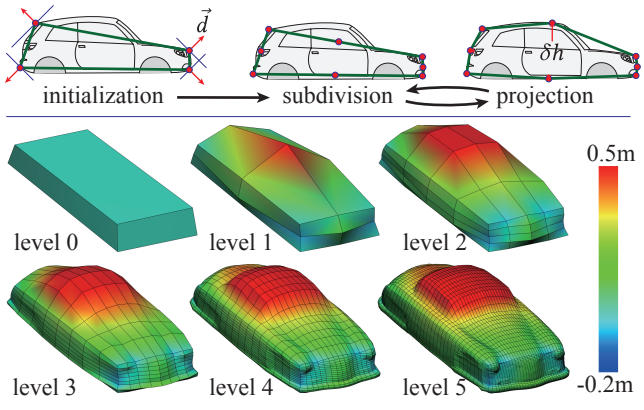


Figure 2: Shrink wrapping parameterization. (*top*) 2D schematic illustration. (*bottom*) Example of parameterization in different subdivision levels.

3 MACHINE LEARNING

3.1 Training Data

We subdivide the base cube five times to create a quad mesh with 6146 vertices. The shape of the quad mesh is parameterized with the XYZ coordinate of the eight corner points and the projection height δh of the 6138 subdivision points. We also included three additional parameters to encode the location of the front and rear tire axis and its radius. After all, we have a parameter vector with $8 \times 3 + 6138 + 3 = 6165$ dimensions.

We use car shapes in the ShapeNet [Chang et al. 2015] data set for our machine learning. There are approximately 3.5 k shapes under the "car" category. However, some of them are highly concave if, for example, their windows are open. We manually selected realistic and reasonably convex 1243 cars. These cars include various styles such as sedans, wagons, pickup trucks, sports cars and classic cars. For these cars, we manually removed sharp non-convex details such as tires, side-mirrors, spoilers, and antennas. We compute the depth surface from five projection directions (top, front, back, right and left). We do not project depth from the below because the underbody of the car is sometimes missing or is very complicated (with pipes or shafts). The shrink wrap vertex projection for the underbody completely relies on depth surfaces from side views.

3.2 Autoencoder Network

So far, we have described how a 3D shape is parameterized in a fix-sized high dimensional vector. This high dimensional space is very difficult to explore manually because there are too many parameters to tweak. In this section, we describe a way to extract a much lower number of parameters that parameterize various shapes in the same category. In other words, we extract a *shape manifold*, which is represented in few parameters, from the high dimensional parameter space.

There are many techniques used to reduce the number of parameters such as principal component analysis. Here, we use the autoencoder technique to construct nonlinear mapping between a reduced number of parameters to high-dimensional parameters. The autoencoder is a neural network that has a hidden layer with a small number of neurons. For the technical details of the autoencoder, please refer to the survey [Schmidhuber 2015].

Fig. 3 illustrates the configuration of our neural network. Our neural network is a feed-forward neural network with four fully connected layers. The output layer has a linear activation function,

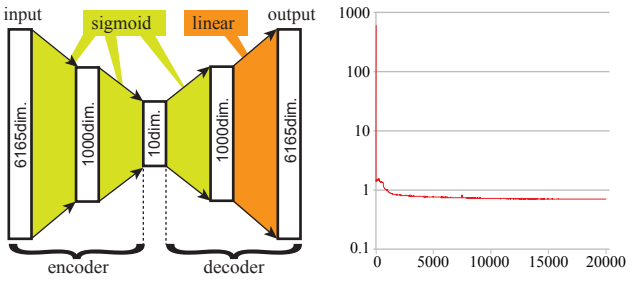


Figure 3: (*left*) Configuration of our autoencoder network. (*right*) Convergence of the sum of squares in the training.

while the other layers use a sigmoid function for the activation functions to express nonlinearity.

We train our neural network such that the input values and the output values are as similar as possible for all the training shapes in the sum of squares measure. We used the GPU implementation of the Tensor flow™ to train the network. The dropout rate is set to be 20% for all the layers to reduce the over-fitting problem. We used the stochastic gradient descent using an Adam optimizer with batch size 50.

4 RESULT

Our parameterization of 3D shapes can robustly parameterize all of the input triangle meshes of the cars without serious distortion such as flipped faces. To encourage further use of this dataset, we attached entire quad meshes obtained through our parameterization as supplemental material. The parameterization is fast enough to be interactive. Conversion from the input triangle mesh (which contains 11.2k triangles on average) into hight-map parameters takes 0.33 second on average.

It takes about 20 minutes to iterate the optimization 200,000 times to obtain the convergence in the network training (see Fig. 3-right). Thanks to our consistent parameterization, we could train the autoencoder network with relatively small errors. The root mean squared error of the neural network averaged over all the training set and all the vertices is 0.01 m, which indicates the input and output shape is only different in the order of few centimetre by average. Fig. 4-top shows some of the examples of the input and output of the autoencoder network. The autoencoder preserves the feature of the input shapes well.

The middle layer of our autoencoder network has ten neurons, which is the smallest among all the four layers. The neural networks before and after the smallest layer are called the encoder and decoder, respectively. Since the output of the network is fully determined by the input of these ten neurons, we can synthesize new shapes by changing input values $\mathbf{q}$ for the decoder between zero and one, which is the range of the sigmoid function (see Fig. 4-bottom for some of the synthesis results). The synthesized car shapes have detailed features such as the front grills or the rear visors. The number of input parameters is small enough for the user to directly manipulate them with slide bars. The average shape synthesis from the low-dimensional parameters runs roughly 300 FPS.

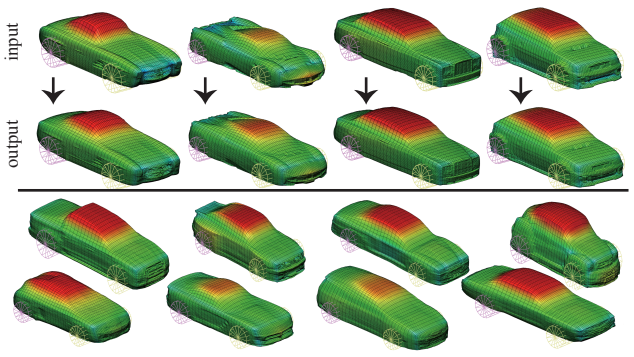
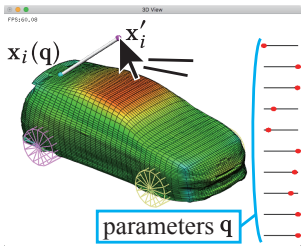


Figure 4: (*top*) Examples of input and output of our autoencoder network. (*bottom*) Synthesized car shapes

4.1 Interactive Exporation



So far, we have described the method used to consistently parameterize shapes and reduce their dimensions using the autoencoder. However, it is sometimes difficult to determine how to manipulate the parameters $\mathbf{q}$ to obtain a desirable shape since the relationship between parameters and the resulting shape is not obvious. Hence, our interface allows the user to interactively specify $\mathbf{x}'_i$ which is the target position for a vertex i to steer the synthesis results (see the inset figure). The interface runs the optimization of the input of the decoder parameter $\mathbf{q}$ such that the position of the output shape's vertex $\mathbf{x}_i(\mathbf{q})$ is as close as possible by minimizing the following error

$$E(\mathbf{q}) = ||\mathbf{x}_i(\mathbf{q}) - \mathbf{x}'_i||_2. \quad (2)$$

Since our subdivision and projection approach provides a closed form relationship between the input projection heights and coordinates of the vertices, we could analytically compute the gradient of the error E concerning the low-dimensional parameter $\mathbf{q}$. Once the gradient is computed, we update the parameter using the Newton-Raphson iterations as $\mathbf{q} := \mathbf{q} - E(\mathbf{q})\,(\partial E/\partial \mathbf{q})\,/||\partial E/\partial \mathbf{q}||^2$. For each update, we project each element of $\mathbf{q}$ between zero and one if it exceeds that range. The optimization works at 17 FPS, which is sufficiently fast to allow the interactive manipulation.

5 CONCLUSION

We introduced a parameterization approach that combines the depth map and shrink wrapping to efficiently and robustly construct a consistent parameterization for machine learning of 3D shapes. We further demonstrated the construction of a shape manifold using an autoencoder and presented an interface to directly manipulate the generation of 3D shapes.

Our method is not suitable for highly concave shapes such as characters' bodies because the coarse geometry is far from a cube. This is not a fundamental limitation because we can change the base coarsest mesh to other than a cube (such as coarse quad mesh) or segment the shape into several convex parts that are parameterized individually. We are also considering using our manifold representation of car shapes to improve the object detection framework for self-driving cars.

REFERENCES

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. *ShapeNet: An Information-Rich 3D Model Repository.* Technical Report arXiv:1512.03012 [cs.GR].

Leif P. Kobbelt, Jens Vorsatz, and Ulf and Labsik. 1999. A Shrink Wrapping Approach to Remeshing Polygonal Surfaces. *Computer Graphics Forum* 18, 3 (1999), 119–130.

Jonathan Masci, Emanuele Rodolà, Davide Boscaini, Michael M. Bronstein, and Hao Li. 2016. Geometric Deep Learning. In *SIGGRAPH ASIA 2016 Courses (SA '16).* ACM, New York, NY, USA, Article 1, 50 pages.

J. Schmidhuber. 2015. Deep Learning in Neural Networks: An Overview. *Neural Networks* 61 (2015), 85–117. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].