Cube Craft for Mesh Stylization

Haoda Li
haoda_li@berkeley.edu
University of California Berkeley
Berkeley, California, USA

Puyuan Yi
yipuyuan@berkeley.edu
University of California Berkeley
Berkeley, California, USA

Weiji Li
weiji@berkeley.edu
University of California Berkeley
Berkeley, California, USA

Zhen Jiang
zhen_jiang16@berkeley.edu
University of California Berkeley
Berkeley, California, USA

Figure 1: Cubic Craft turns triangle meshes (grey) into cubic-styled meshes (green)

ABSTRACT

We present a stylization tool to automatically manipulate triangle meshes into a cubic style. Our tool uses a cubic stylization algorithm[Liu and Jacobson 2019] to cubify the user's provided meshes. The algorithm extends the as-rigid-as-possible energy [Sorkine and Alexa 2007] with an additional L1 regularization, hence can work seamlessly with ADMM optimization. Cubic stylization works only on the vertex positions, hence preserving the geometrical details and topology. In addition, we implemented the algorithm with GPU acceleration and achieves real-time interactive editing. We also created a user-friendly interaction surface to let users easily change the algorithm's hyperparameter and cubify their own mesh. With our tool, 3D artists can create Minecraft-styled objects with ease.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Computing methodologies** → **Mesh geometry models**; Parallel algorithms; • **Human-centered computing** → *Visualization toolkits.*

KEYWORDS

mesh stylization, mesh deformation, geometry processing, visualization tools

ACM Reference Format:
Haoda Li, Puyuan Yi, Weiji Li, and Zhen Jiang. 2023. Cube Craft for Mesh Stylization. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

With the increasing availability of image stylization filters and non-photorealistic rendering techniques, creating artistic images has become much more accessible to non-professional users. However, the direct stylization of 3D shapes and non-realistic modeling has not yet been given as much attention. Despite the advancements in technology, professional industries like visual effects and video games still rely on trained modelers to meticulously create non-realistic geometric assets. This is because exploring geometric styles

presents a greater challenge, as it involves dealing with arbitrary topologies, curved metrics, and non-uniform discretization. While image stylization tools have made it easier to generate artistic imagery, there is still a lack of effective tools for generating artistic geometry, which remains a major obstacle to the development of geometric stylization.

The focus of this paper is on a specific style of sculpture, namely the cubic style. This style has been prevalent throughout art history (ancient sculptures) and modern game history (Minecraft). In this work, we have developed a stylization tool *cubic stylization* based on GPU that takes a 3D shape as input and outputs a deformed shape that has the same style as cubic sculptures. This tool is aimed at helping artists and designers achieve the cubic style more easily and efficiently, while also providing a new way to explore and experiment with this timeless artistic tradition.

Our implemented method *cubic stylization* formulates the task as an energy optimization problem, which preserves the geometric details of a shape while transforming it into a cubic form. Specifically, this energy function combines an as-rigid-as-possible (ARAP) energy with a special L1 regularization. This energy can be minimized efficiently using the local-global approach with the Alternating Direction Method of Multipliers (ADMM). This method has strong flexibilities that allowing artists and designers to achieve a wide range of stylistic variations within the cubic style, providing them with greater creative freedom and expressive potential.

Our main contributions are summarized as follows:

- We implemented GPU-based *cubic stylization* function, which allows real-time cubic style object generation and rendering.
- We created a user-friendly GUI to interact with our algorithm. Users can easily change the hyperparameter of the algorithm and observe different results.

2 RELATED WORKS

In this section, our primary focus is on exploring methods for processing geometry. Specifically, we will be discussing various techniques for studying geometric styles and deformation methods that share common technical similarities. Our aim is to provide a comprehensive overview of these methods and their applications and to highlight their significance in the field of geometry processing.

2.1 shape deformation

2.2 Different geometric style

3 METHOD

Our method is based on cubic stylization [Liu and Jacobson 2019], a method to deform the input mesh into a cubic stylized mesh. Generally, the method adds a new L1 regularization on the deformation with (As-rigid-as-possible) ARAP[Sorkine and Alexa 2007]energy optimization. By regularizing each vertex's normals to align with the axis, the mesh can have a cubic style, while maintaining the local shape.

The problem description is illustrated as follows: given a triangle mesh S as a set of vertices $V \in \mathbb{R}^{n\times 3}$ and a set of faces $F \in \mathbb{R}^{m\times 3}$. We want to output a deformed shape $\tilde{V}$. The output shape will have each sub-component in the style of axis-aligned cubes and will retain the geometric details of the original mesh.

We will describe our method and implementation in the following sections: In section 3.1, we will talk about the As-rigid-as-possible (ARAP) Deformation[Sorkine and Alexa 2007], and elaborate on its energy functions. Then we will talk about cubic Stylization [Liu and Jacobson 2019] in section 3.2. The implementation part is in section 4.

3.1 As-rigid-as-possible Deformation

The thought of ARAP deformation is very intuitive: Given the cell C_i corresponding to vertex i, and its deformed version $\tilde{C}_i$, ARAP defines the approximate rigid transformation between the two cells by observing the edges emanating from the vertex i in S and $\tilde{S}$, where S and $\tilde{S}$ denote the original triangle mesh and the deformed triangle mesh. Note that $\tilde{S}$ should have the same connectivity as S. If the deformation $C_i \rightarrow \tilde{C}_i$ is rigid, there must exists a rotation matrix R_i such that:

$$\tilde{V}_i - \tilde{V}_j = R_i(V_i - V_j), \forall j \in \mathcal{N}(i) \tag{1}$$

$\mathcal{N}(i)$ denotes the set of vertices connected to vertex i, also called the one-ring neighbors.

When the deformation is not rigid, we can still find the best approximating rotation matrix R_i that fits the above equations in a weighted least squares sense, i.e., minimizes

$$E(C_i, \tilde{C}_i) = \sum_{j\in\mathcal{N}(i)} w_{ij} \|(\tilde{V}_i - \tilde{V}_j) - R_i(V_i - V_j)\|^2 \tag{2}$$

w_{ij} is the cotangent weight between vertice i and vertice j. What we need to do is to solve for vertex position $\tilde{V}_i$ of the deformed triangle mesh $\tilde{S}$ that minimizes the energy function above.

3.2 Cubic Stylization

In this section, we will illustrate the cubic stylization algorithm. Following the As-rigid-as-possible Deformation, the cubic problem is also viewed as an energy optimization problem. In addition to the ARAP energy, the energy term has an additional L1 regularization term. The full energy term is listed as follows:

$$\begin{aligned} E(C_i, C_i') = \quad & \sum_{i\in V} \sum_{j\in\mathcal{N}(i)} \frac{w_{ij}}{2} \|R_i d_{ij} - \tilde{d}_{ij}\|_F^2 + \lambda a_i \|R_i \tilde{n}_i\|_1 \\ \text{subject to} \quad & \tilde{V}, R_1, ..., R_n \end{aligned}$$

where $d_{ij} = V_i - V_j$ and $\tilde{d}_{ij} = \tilde{V}_i - \tilde{V}_j$. In the L1 regularization term, $\hat{n}_i$ denotes the area-weighted unit normal vector of v_i and a_i is the barycentric area of v_i and λ is a tuning parameter.

Generally, we follow the local global step as ARAP energy optimization, i.e. for each iteration, we first optimize the rotations with constant vertex positions, and then optimize vertices with constant rotations. The local step involves finding the rotation matrix $R_1, ..., R_n$, i.e. For each vertex i, we are to optimize F:

$$R_i^* = \arg\min_{R_i \in SO(3)} \sum_{j\in\mathcal{N}(i)} \frac{w_{ij}}{2} \|R_i d_{ij} - \tilde{d}_{ij}\|_F^2 + \lambda a_i \|R_i \tilde{n}_i\|_1 \tag{3}$$

note that the ARAP energy can be expressed in matrix formations

$$\frac{1}{2}(R_iD_i - \tilde{D}_i)^T W_i(R_iD_i - \tilde{D}_i)$$

where $D_i, \tilde{D}_i \in \mathbb{R}^{3\times|\mathcal{N}(i)|}$ are stacked rim/spoke edge vectors and W_i is the diagonal matrix of $w_1, ..., w_n$. Then, write $z = R_i\hat{n}_i$, we can turn the formation into

$$\begin{aligned} \text{minimize}_{z,R_i} \quad & \frac{1}{2}(R_iD_i - \tilde{D}_i)^T W_i(R_iD_i - \tilde{D}_i) + \lambda a_i\|z\|_1 \\ \text{subject to} \quad & z - R_i\hat{n}_i = 0 \end{aligned}$$

Now We can solve the local step using the alternating direction method of multipliers (ADMM) updates. Applying ADMM, the update steps are

$$\begin{aligned} R_i^{k+1} &= \arg\min \frac{1}{2}(R_iD_i - \tilde{D}_i)^T W_i(R_iD_i - \tilde{D}_i) + \frac{\rho^k}{2}\|R_i\hat{n}_i - z^k + u^k\|_2^2 \\ z^{k+1} &= \arg\min \lambda a_i\|z\|_1 + \frac{\rho^k}{2}\|R_i^{k+1}\hat{n}_i - z + u^k\|_2^2 \\ \tilde{u}^{k+1} &= u^k + R_i^{k+1}\hat{n}_i - z^{k+1} \\ \rho^{k+1}, u^{k+1} &= \text{update}(\rho^k) \end{aligned}$$

Then, consider each update, The rotation update can be viewed as

$$\begin{aligned} R_i^{k+1} &= \arg\max tr(R_iM_i) \\ M_i &= \begin{bmatrix}[D_i] & [\hat{n}_i]\end{bmatrix}\begin{bmatrix}[W_i] & 0 \\ 0 & \rho^k\end{bmatrix}\begin{bmatrix}[\tilde{D}_i] \\ [(z^k - u^k)^T]\end{bmatrix} \end{aligned}$$

This becomes an Orthogonal Procrustes problem, and the solution is given by $M = U\Sigma V^T$ through single value decomposition, and then

$$R = UV^T$$

up to $det(R) > 0$ by alternating the sign of U's column. The z update is an instance of lasso problem, which can be solved with a shrinkage step

$$z^{k+1} = S_{\lambda a_i/\rho^k}(R_i^{k+1}\hat{n}_i + u^k)$$

where the shrinkage is defined as

$$S_\chi(x_j) = (1 - \frac{\chi}{|x_j|}) + x_j$$

Now all the local step is solved.

4 IMPLEMENTATION

We implement the cubic stylization [Liu and Jacobson 2019] algorithm using Python and libigl [Jacobson et al. 2018]. In addition, we observe that the local step updates each vertex independently, providing opportunities for parallelization. We use Taichi [Hu et al. 2019] to implemented a GPU accelerated version. Compared to the CPU implementation [Liu and Jacobson 2019], our implementation gradually accelerated the local step computation. We tested our implementation on an AMD R9 5900HS CPU with a NVIDIA 3050ti GPU, and listed the performance in Table 1.

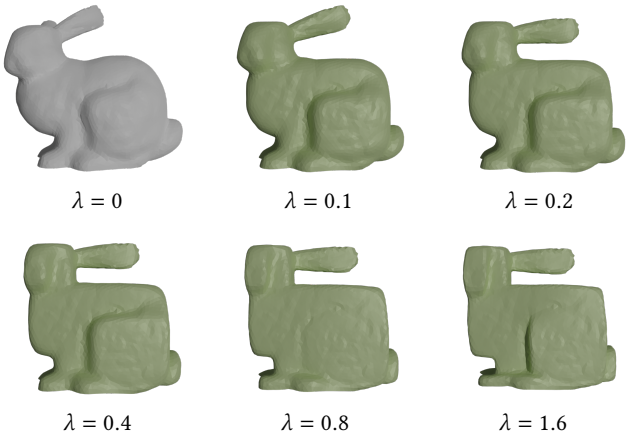

Figure 2: Meshes with different cubeness

Figure 3: Meshes with different cube orientation

5 USER INTERFACE

We provide a graphical interface for the users to visualize and easily edit the meshes. The graphical interface is based on the GUI system provided by Taichi. Given a triangle mesh, our graphical interface allows the user to change the parameters in the algorithm, visualize the deformations, and save the resulting mesh. The example of different cubic stylization parameters that work differently on bunny.obj is shown in Figure 2.

In addition to the cubeness parameter, we notice that cube stylization is orientation dependent. The cubeness is achieved by forcing all vertex normals to align with the three standard axes. If we rotate the input mesh, the output shape will be different. Note that the

Mesh name	|V|	CPU time (s)	GPU time (s)
homer	6002	16.03	1.77
bunny	6172	43.23	1.96
armadillo	49990	370.64	7.49

Table 1: Running time for Cubic Stylization

same effect can be achieved by applying a coordinate transformation on all vertex normals. Therefore, we add the coordinate rotation parameters (θ, ϕ) so that users can have different cube orientations. The experiments of different orientation on cubic stylization are presented in Figure 3.

Similar to Sorkine and Alexa's approach, we can put constraints on vertex positions. Currently, the vertex index and positions are hard-coded. In the future, the users will be able to left-click the mesh and place constraints on the deformation.

6 CONCLUSION

In conclusion, our work presents a powerful tool for cubic stylization that enables 3D artists to create Minecraft-styled objects with ease. Our algorithm, which extends the as-rigid-as-possible energy with an L1 regularization, works seamlessly with ADMM optimization and preserves the underlying geometrical details and topology of the mesh. Furthermore, our implementation with GPU acceleration allows for real-time interactive editing, making the tool both efficient and intuitive to use.

Overall, our work contributes to the growing field of geometry processing by presenting a novel approach to stylization. The ability to manipulate and transform meshes in a cubic style has significant potential for a range of applications, including architectural design, game development, and animation. We believe that our tool will be particularly valuable to 3D artists who wish to create unique and visually striking objects quickly and efficiently.

REFERENCES

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 201.

Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

Hsueh-Ti Derek Liu and Alec Jacobson. 2019. Cubic Stylization. *ACM Transactions on Graphics* (2019).

Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, Vol. 4. 109–116.