Cubic Stylization

HSUEH-TI DEREK LIU, University of Toronto, Canada
ALEC JACOBSON, University of Toronto, Canada

1 INTRODUCTION

The availability of image stylization filters and non-photorealistic rendering techniques has dramatically lowered the barrier of creating artistic imagery to the point that even a non-professional user can easily create stylized images. In stark contrast, direct stylization of 3D shapes or non-realistic modeling has received far less attention. In professional industries such as visual effects and video games, trained modelers are still required to meticulously create non-realistic geometric assets. This is because investigating geometric styles is more challenging due to arbitrary topologies, curved metrics, and non-uniform discretization. The scarcity of tools to generate artistic geometry remains a major roadblock to the development of geometric stylization.

In this paper, we focus on the specific style of cubic sculptures. The cubic style is prevalent across art history, for instance the ancient
sculptures from the post-classic era (900-1250 CE), Maya sculptures, block statues in Egypt, and modern abstract sculptures such as the ones from Constantin Brâncuși and Ju Ming (Fig. 2). In addition, the cubic style is a popular digital art, such as the award-winning Anicube by Aditya Aryanto (Fig. 3). Complementing their presence in art, cubic shapes also present themselves in fabrication and furniture purposes (Fig. 4). We contribute to the rich history of cubic sculpting by providing a stylization tool that takes a 3D shape as input and outputs a deformed shape that has the same style as cubic sculptures.

We present cubic stylization which formulates the task as an energy optimization that naturally preserves geometric details while cubifying a shape. Our proposed energy combines an as-rigid-as-possible (ARAP) energy with an \( \ell^1 \) regularization. This energy can be minimized efficiently using the local-global approach with alternating direction method of multipliers (ADMM). This variational approach affords the flexibility of incorporating many artistic controls, such as applying constraints, non-uniform cubeness, and different global/local cube orientations (Sec. 4). Moreover, our method requires no remeshing (Fig. 5) and generalizes to polyhedral stylization (Fig. 24). Our proposed tool for non-realistic modeling goes beyond the 2D stylization and opens up the possibility of, for instance, creating non-realistic 3D worlds in virtual reality (Fig. 1).

2 RELATED WORK

Our work shares similar motivations to a large body of work on image stylization [Kyprianidis et al. 2013], non-photorealistic rendering [Gooch and Gooch 2001], and motion stylization [Hertzmann et al. 2009]. While their outputs are images or stylized animations, we take a 3D shape as input and output a stylized shape. Thus we focus our discussion on methods for processing geometry, including the study of geometric styles and deformation methods that share technical similarities.

Discriminative Geometric Styles. The growing interest in understanding geometric styles has been inspiring recent works on building discriminative models for style analysis. One of the main challenges is to define a similarity metric aligned with human perception. Many works propose to compare projected feature curves [Li et al. 2013; Yu et al. 2018], sub-components of a shape [Hu et al. 2017; Lun et al. 2015; Xu et al. 2010], or using learned features [Lim et al. 2016]. These models enable users to synthesize style compatible scenes [Liu et al. 2015] or transfer style components across shapes [Berkinen et al. 2017; Lun et al. 2016; Ma et al. 2014]. However, these methods are designed for discerning and transferring styles, instead of generating 3D stylized shapes directly.

Generative Geometric Styles. Direct 3D stylization has been an important topic in computer graphics. Many generative models have been proposed for producing specific styles, without relying on identifying and transferring style components from other shapes. This includes creating the collage art [Gal et al. 2007; Theobalt et al. 2007], voxel/lego art [Luo et al. 2015; Testuz et al. 2013], neuronal homunculus [Reinert et al. 2012], the manga style shapes [Shen et al. 2012], shape abstraction [Kratt et al. 2014; Mehra et al. 2009; Yumer and Kara 2012], and bas-relief sculptures [Bian and Hu 2011; Kerber et al. 2009; Schüller et al. 2014; Song et al. 2007; Weyrich et al. 2007]. While not pitched as stylization techniques, many geometric flows and filters can also be used for creating stylized geometry, such as creating edge-preserving smoothing geometry [Zhang et al. 2018], piece-wise planar [He and Schaefer 2013; Stein et al. 2018b] or developable shapes [Stein et al. 2018a], and stylized shapes prescribed by image filters [Liu et al. 2018] (see Fig. 6). Our method contributes to the field of direct 3D stylization, focusing on the style of cubic sculptures (Fig. 7).

Shape Deformation. Many works deal with the question of how to deform shapes given modeling constraints. One of the most popular choices is the ARAP energy [Chao et al. 2010; Igarashi et al. 2005; Liu et al. 2008; Sorkine and Alexa 2007], which measures local rigidity of the surface and leads to detail-preserving deformations. Not just
We take advantage of the main instruments in the construction of polycube maps [Tarini et al. 2010] and [Garland 1999; Zorin 2006]. This class of accelerations shares similar characteristics to multiresolution modeling (see [Garland 1999; Zorin 2006]). However, having nearly interactive performance on highly detailed meshes still remains a major challenge. An alternative strategy to speed it up is to use the hierarchical deformation which optimizes ARAP on a low resolution model and then recover the original details back afterwards [Manson and Schaefer 2011]. This class of accelerations shares similar characteristics to multiresolution modeling (see [Garland 1999; Zorin 2006]). We take advantage of the ARAP energy for detail preservation and adapt the method of Manson and Schaefer [2011] to accelerate our cubic stylization to meshes with millions of faces.

**Axis-Alignment in Polycube Maps.** Axis-alignment is an important property for many geometry processing tasks, such as [Muntoni et al. 2018; Stein et al. 2019]. Especially, this concept is one of the main instruments in the construction of polycube maps [Tarini et al. 2004], including defining polycube segmentations [Fu et al. 2016; Livesu et al. 2013; Zhao et al. 2018] and the cost function for polycube deformations [Gregson et al. 2011; Huang et al. 2014]. Although polycube methods can obtain cubic geometry, they fail to preserve detail (Fig. 8) because they are not desirable for intended applications such as parameterization and hexahedral meshing [Cherchi et al. 2016; Fang et al. 2016; Garcia Fernández et al. 2013; He et al. 2009; Lin et al. 2008; Wang et al. 2007, 2008; Yu et al. 2014].

One tempting direction of creating cubic geometry is to use voxelization. However, voxelization fails to capture the details depicted by the artists and cannot capture the wide spectrum of cubeness across cubic sculptures. Another tempting direction is to recover geometric features from the polycube results. This would lead to a multi-step algorithm and suffer from limitations of particular detail encoding schemes (e.g., bump maps). Even if we stop the polycube algorithm earlier such as the method of [Gregson et al. 2011] to maintain details, it does not provide a satisfactory solution (see the inset for a comparison with Fig. 5 in [Gregson et al. 2011]). More importantly, many artistic controls in Sec. 4 would be nontrivial to add on. Building stylization on top of polycube methods would also suffer from slow performance. For instance, Huang et al. [2014] propose a polycube method that minimizes the $\ell^1$-norm of the normals on the deformed tetrahedral mesh with ARAP for regularization. Their formulation involves minimizing a complicated non-linear function and requires minutes to hours to optimize. Thus a stylization built on top of this method would be even slower. In contrast, our formulation is a single energy optimization which can easily incorporate many artistic controls (Sec. 4). Our energy is similar to the polycube energy of [Huang et al. 2014] in that we also minimize the ARAP energy with a $\ell^1$ regularization, but the key difference is that we define the $\ell^1$-norm on the rotated normals of the original mesh instead. This allows us to optimize our energy much faster using the local-global approach with ADMM in only a few seconds (Table 1).

**3 METHOD**

The input of our method is a manifold triangle mesh with/without boundaries. Our method outputs a cubified shape where each sub-component has the style of an axis-aligned cube. Meanwhile, our stylization will maintain the geometric details of the original mesh. Let $V$ be a $|V| \times 3$ matrix of vertex positions at the rest state and $\hat{V}$ be a $|V| \times 3$ matrix containing the deformed vertex positions. We denote $d_{ij} = [v_j - v_i]^T$ and $d_{ij} = [v_j - \hat{v}_i]^T$ be the edge vectors between vertices $i,j$ at the rest and deformed states respectively. The energy for our cubic stylization is as follows

$$
\text{minimize} \quad \hat{V} \in \{R_i\} \sum_{i \in V} \sum_{j \in N(i)} \frac{w_{ij}}{2} \left\| R_i d_{ij} - \bar{d}_{ij} \right\|^2 + \lambda \left\| R_i \hat{n}_i \right\|_1.
$$

The first term is the arap energy [Sorkine and Alexa 2007], where $R_i$ is a 3-by-3 rotation matrix, $w_{ij}$ is the cotangent weight [Pinkall and Polthier 1993], and $N(i)$ denotes the “spokes and rims” edges of the $i$th vertex [Chao et al. 2010] (see the inset). In the second term, $\hat{n}_i$ denotes the unit area-weighted normal vector of a vertex $i$ in $\mathbb{R}^3$. The $a_i \in \mathbb{R}^3$ is the barycentric area of vertex $i$, which is crucial for $\lambda$ to exhibit the similar cubeness across different mesh resolutions. Intuitively, minimizing the $\ell^1$-norm of the rotated normal encourages $R_i \hat{n}_i$ to align with one of coordinate axes because $\ell^1$-norm encourages sparsity. Combining the two, the optimal rotation ($R_i^*$) would simultaneously preserve the local structure (arap) and encourage axis alignment (cubeness).

We adapt the standard local-global update strategy to optimize our energy [Sorkine and Alexa 2007] (see Alg. 1). Our global step, updating $V_i$, is achieved by solving a linear system, the same as the Equation 9 in Sorkine and Alexa [2007]. Our local step, finding the optimal rotation, is however different from the previous literature due to the $\ell^1$ term.

### 3.1 Local-Step

Our local step for each vertex $i$ can be written as

$$R^*_i = \arg \min_{R_i \in SO(3)} \frac{W_i}{2} \| R_iD_i - \bar{D}_i \|_F^2 + \lambda a_i \| R_i \hat{n}_i \|_1, \quad (2)$$

where $W_i$ is a $|N(i)| \times |N(i)|$ diagonal matrix of cotangent weights, $D_i$ and $\bar{D}_i$ are $3 \times |N(i)|$ matrices of rim/spoke edge vectors at the rest and deformed states respectively. By setting $z = R_i \hat{n}_i$, we can rewrite Eq. 2 as

$$\min_{z, R_i \in SO(3)} \frac{W_i}{2} \| R_iD_i - \bar{D}_i \|_F^2 + \lambda a_i \| z \|_1 \quad (3)$$

subject to $z - R_i \hat{n}_i = 0$.

Eq. 3 is a standard ADMM formulation. We solve this local step using the scaled-form ADMM updates [Boyd et al. 2011]:

$$R^{k+1}_i \leftarrow \arg \min_{R_i \in SO(3)} \frac{W_i}{2} \| R_iD_i - \bar{D}_i \|_F^2 + \frac{\rho}{2} \| R_i \hat{n}_i - z^k + u^k \|_2^2 \quad (4)$$

$$z^{k+1} \leftarrow \arg \min_z \lambda a_i \| z \|_1 + \frac{\rho}{2} \| R_i^{k+1} \hat{n}_i - z + u^k \|_2^2 \quad (5)$$

$$u^{k+1} \leftarrow u^k + R_i^{k+1} \hat{n}_i - z^{k+1} \quad (6)$$

$$\rho^{k+1}, u^{k+1} \leftarrow \text{update}(\rho^k) \quad (7)$$

where $\rho \in \mathbb{R}^+_0$ is the penalty and $u$ is the scaled dual variable.

Eq. 4 is an instance of the orthogonal Procrustes [Gower et al. 2004]

$$R^{k+1}_i \leftarrow \arg \max_{R_i \in SO(3)} \text{Tr}(R_i M_i)$$

where $M_i = [D_i \hat{n}_i] W_i \rho \begin{bmatrix} \bar{D}_i^T \\ (z^k - u^k)^T \end{bmatrix}$. One can derive the optimal $R_i$ from the singular value decomposition of $M_i = \mathcal{U}_i \Sigma_i \mathcal{V}_i^T$:

$$R^{k+1}_i \leftarrow \mathcal{V}_i \mathcal{U}_i^T,$$

up to changing the sign of the column of $\mathcal{U}_i$ so that $\det(R_i) > 0$.
We update the penalty with zeros, and set the initial

Applying different rotations to the mesh lead to different results.

and elaborate stopping criteria, such as the method of [Zhu et al. 2018], for the convergence plots). More

It is lower than $3$ relative per vertex displacements, the infinity norm of

$\epsilon$ that for extremely large $z$ can significantly speed up the optimization. Specifically, we initialize

Starting the local-step parameters from the previous iteration can

iteratively until the norm of primal/dual residuals are small. Warm

shrinkage [Shapiro 1996], which can be solved with a

Eq. 5 is an instance of the lasso problem [Boyd et al. 2011; Tibshirani 1996], which can be solved with a

shrinkage step:

We update the penalty $\rho$ (Eq. 7) according to Sec. 3.4.1 in [Boyd et al.

In short, local fitting is performed by running Eq. 8, 9, 6, and 7

iteratively until the norm of primal/dual residuals are small. Warm

starting the local-step parameters from the previous iteration can

significantly speed up the optimization. Specifically, we initialize $z, u$

with zeros, and set the initial $\rho = 10^{-4}, \epsilon^\text{abs} = 10^{-5}, \epsilon^\text{rel} = 10^{-3}, \mu = 10,$ and $\nu^\text{incr} = 2$ (the same notation as used in Sec. 3 of [Boyd et al. 2011]). Then $z, u, \rho$ are reused in consecutive iterations. Note that for extremely large $\lambda$ one may need to increase the initial value of $\epsilon^\text{abs}$ accordingly in order to avoid bad local minima. We stop the optimization when the relative displacement, the infinity norm of relative per vertex displacements, is lower than $3 \times 10^{-3}$ (see Fig. 12 for the convergence plots). More elaborate stopping criteria, such as the method of [Zhu et al. 2018], could also be used.

At this point we have completed the cubic stylization algorithm summarized in Alg. 1, enabling us to efficiently create cubified shapes (see Fig. 10). In Fig. 11 and 14 we show that this formulation is applicable to meshes with boundaries and non-orientable surface respectively. As the cubeness is dependent to the orientation of the mesh, one can apply different rotations to control how the stylization runs (Fig. 13). We expose the weighting $\lambda$ to be a design parameter controlling the cubeness of a shape (Fig. 9).

However, the “vanilla” cube stylization shares the same caveat as other distortion minimization algorithms: having slow runtime on high resolution meshes.

3.2 Affine Progressive Meshes

Manson and Schaefer [2011] propose a hierarchical approach to accelerate ARAP deformations. The main idea is to deform a low-resolution model and recover the details back after convergence.

Specifically, Manson and Schaefer [2011] propose a progressive mesh [Hoppe 1996] representation which first simplifies a given mesh via a sequence of edge collapses, and then represents the mesh as its coarsest form together with a sequence of vertex splits. After applying some deformations to the coarsest mesh, each “deformed” vertex split is computed by fitting the best local rigid transformation. This approach is suitable for deformations that are locally rigid (e.g., ARAP), but our cubic stylization is less rigid for larger $\lambda$.

So we fit the best affine transformation in each vertex split,
We implement the cubic stylization in C++ using libigl [Jacobson et al. 2018] and evaluate our runtime on a MacBook Pro with an Intel i5 2.3GHz processor. Table 1 lists the parameters and the runtime of our stylization in Fig. 10 (top) and Fig. 16. The results show that we can obtain stylized geometry within a few seconds. This is important for users to receive quick feedback on their parameter choices and iterate on their designs, such as the cubeness $\lambda$ in Fig. 9 and the level of details $m$ in Fig. 17.

**User study.** We prototype a user interface (see the inset) to conduct an informal user study with six participants (4 male, 2 female) between the ages of 24 and 29. Participant 3D modeling experience ranged from none (complete novice) to three years of hobbyist use. Each participant was instructed for three minutes on how to use our software to load a mesh and control the cubeness parameter $\lambda$. Then we asked them to cubify a shape of their choosing from a collection of ten shapes. The results of their work is shown in Fig. 18. All users reported that they were satisfied with the cubeness of their resulting shape. One user said their work is shown in Fig. 18. All users reported that they were satisfied with the cubeness of their resulting shape. One user said

Table 1. For each example in Fig. 10 and Fig. 16, we report the number of faces in the original model ($|F|$), $\lambda$ weight, number of faces of the coarsest mesh ($m$), number of iterations (Iters.), pre-processing time (Pre.), and runtime at the online stage (Runtime).

| Model     | $|F|$ | $\lambda$ | $m$ | Iters. | Pre. | Runtime |
|-----------|------|-----------|-----|-------|------|---------|
| Fig. 10, left | 39K  | 0.20      | n/a | 106   | n/a  | 5.08s   |
| Fig. 10, mid. | 41K  | 0.20      | n/a | 93    | n/a  | 4.50s   |
| Fig. 10, right | 21K  | 0.40      | n/a | 86    | n/a  | 2.26s   |
| Fig. 16, left | 2018K | 0.20     | 20K | 83    | 64.19s | 3.93s   |
| Fig. 16, mid. | 346K | 0.40      | 20K | 222   | 10.69s | 4.59s   |
| Fig. 16, right | 811K | 0.30      | 40K | 173   | 30.44s | 8.38s   |

Fig. 15. Affine progressive meshes allow us run cubic stylization on a low-resolution model and then recover original details when converged.

Fig. 16. With the affine progressive meshes, we can scale the cubic stylization to meshes with millions of faces.

Fig. 17. The number of faces $m$ used in the decimated mesh not only controls the runtime but also the frequency level of details that get preserved.

An interesting observation is that the number of faces $m$ in the coarsest mesh not only controls the runtime, but implicitly controls the frequency level of geometric details that gets preserved. In Fig. 17 we show that, under the same $\lambda$, a smaller $m$ keeps details across a wider frequency range; in contrast, a larger $m$ only keeps details at higher frequencies. Therefore one can manipulate the level of preserved features by playing with $m$.

3.3 Implementation

We implement the cubic stylization in C++ using libigl [Jacobson et al. 2018] and evaluate our runtime on a MacBook Pro with an Intel i5 2.3GHz processor. Table 1 lists the parameters and the runtime of our stylization in Fig. 10 (top) and Fig. 16. The results show that we can obtain stylized geometry within a few seconds. This is important for users to receive quick feedback on their parameter choices and

rather than rigid transformations. Specifically, in each edge collapse we store the displacement vectors from the newly inserted vertex $p_j$ to the endpoints $p_j, p_k$ (see the inset) together with a matrix $A$:

$$A = (Q_i Q_i^T)^{-1} Q_i,$$

$Q_i$ is a $3 \times |N(i)|$ matrix where each column is the vector from $p_i$ to one of its one-rings neighbors $N(i)$. If $(Q_i Q_i^T)$ is singular (e.g., in planar regions), we remedy the issue with the Tikhonov regularization [Tikhonov et al. 2013]. Then $A$ is used to computed the deformed displacements for each vertex split as

$$\tilde{p}_j - \tilde{p}_i = \tilde{Q}_i A (p_j - p_i),$$

where $\tilde{p}_i$ denotes the position of vertex $i$ in the cubified coarsened shape, and $\tilde{Q}_i$ is a $3 \times |N(i)|$ matrix containing vectors from $\tilde{p}_i$ to its one-rings neighbors.

Affine progressive meshes allows us to losslessly recover the original meshes undergoing affine transformations. For smooth non-affine transformations such as our cube stylization, it could still be approximately recovered (see Fig. 15). We summarize our cubic stylization with the affine progressive mesh in Alg. 2. Note that the edge collapses is just a pre-processing step. In the online stage, one only needs to run cubic stylization on the coarsest mesh and then apply a sequence of vertex splits to visualize the result on the original resolution. This offers a huge speed-up when interacting the parameter $\lambda$ on highly detailed models (see Fig. 16).

We prototype a user interface (see the inset) to conduct an informal user study with six participants (4 male, 2 female) between the ages of 24 and 29. Participant 3D modeling experience ranged from none (complete novice) to three years of hobbyist use. Each participant was instructed for three minutes on how to use our software to load a mesh and control the cubeness parameter $\lambda$. Then we asked them to cubify a shape of their choosing from a collection of ten shapes. The results of their work is shown in Fig. 18. All users reported that they were satisfied with the cubeness of their resulting shape. One user said...
that controlling the cubeness of their resulting shape is very easy because it only requires tuning a single parameter.

4 ARTISTIC CONTROLS

In addition to the two parameters $\lambda, m$, we expose many variants of our stylization to incorporate artistic controls. As a non-realistic modeling tool, this is important for users to realize their creativity.

We first focus our discussion on a variety of artistic controls that are related to the cubeness parameter $\lambda$. Although Eq. 1 only has a single $\lambda$ for an entire shape, we can actually specify different $\lambda_i$ for each vertex independently to have non-uniform cubeness, which leads to the expression $\lambda_i a_i \| R_i \hat{n}_i \|_1$. In Fig. 19, we use this approach to make the back of the sheep much more cubic than the rest of the shape to create an ottoman-like geometry. We can also specify the non-uniform cubeness $\lambda_i$ in a different way, instead of painting on the surface directly. In Fig. 20 we paint a function on the Gauss map in which the surface normal pointing towards left has higher cubeness. When we map this function back to the surface, we can have a cubified owl that is more cubic when initial normals pointing towards the left and less cubic when pointing towards the right. Similarly, we can have different $\lambda_x, \lambda_y, \lambda_z$ for different axes. In Fig. 21, we replace the cubeness in Eq. 1 with $a_i (\lambda_x (\| R_i \hat{n}_i \|_x) + \lambda_y (\| R_i \hat{n}_i \|_y) + \lambda_z (\| R_i \hat{n}_i \|_z))$ and specify different values for each $\lambda_x, \lambda_y, \lambda_z$ to have the style of a rectangular prism.

If one wants to fix certain parts of the shape, we can easily add constraints in the global step, the same way as the method of Sorkine and Alexa [2007]. In Fig. 4 we add the parts constraint by fixing the position of some vertices when solving the linear system; we add the points constraint by specifying some deformed vertices $V_i$ at user-desired positions. We can also use the same methodology to constrain some parts of the geometry lying on certain planes. For instance, setting $(V_i)_x = 0$ can force vertex $i$ lying on the yz-plane. In Fig. 22 we use this plane constraint to create a table clinger.

In addition, one can utilize the property of the $\ell^1$-norm to have different artistic effects. Because the cubeness term is orientation dependent, in Fig. 13 we can apply different rotations to the mesh.
Fig. 23. We can define the $\ell^1$-norm on different coordinate systems for different parts of the shape, instead of using the world coordinates. In the figure the hands and the body use different coordinate systems (left). By changing them, we can vary the cube orientations for different parts.

Fig. 24. We apply a coordinate transformation inside the $\ell^1$-norm to generalize cubic stylization to polyhedrons.

Fig. 25. We apply non-symmetric coordinate transformations inside the $\ell^1$-norm to generalize stylization to irregular polyhedrons.

before the stylization to control the results. Rather than rotating the mesh, another way is to encode the normal vector in a different coordinate system $\lambda a_i \|R_i \hat{n}_{local}\|_1$, where we use $\hat{n}_{local}$ to denote the user-desired coordinate system for vertex $i$. This perspective allows us to define the $\ell^1$-norm on different coordinate systems for different parts of the shape to obtain different cube orientations (Fig. 23). Beyond the cubic stylization, in Fig. 24, 25 we apply a coordinate transformation $B$ inside the $\ell^1$-norm $\lambda a_i \|BR_i \hat{n}_i\|_1$ to achieve polyhedral stylization, for which we provide the details in App. A. Once we obtain the stylized shapes, they are ready to be used by standard deformation techniques in animations (Fig. 26).

5 LIMITATIONS & FUTURE WORK
Accelerating the stylization to real-time would enable faster iterations between designs. Developing a more robust stylization technique for bad quality triangles, non-manifold meshes, or even point cloud could be useful for stylizing real-world geometric data. Guaranteeing results to be self-intersection free would be desirable for downstream tasks. Extending our energy to be invariant to discretizations could achieve more consistent results across different resolutions (see Fig. 27). Extending to quadrilateral meshes and NURBS surfaces could benefit existing modeling or engineering design softwares. Generalizing to volumetric meshes could have a better volume preservation. Exploring different deformation energies and $\ell^p$-norm could lead to novel stylization tools for non-realistic modeling. Beyond generating stylized shapes, the mathematical expression of the cubic geometry could offer insights toward understanding more intricate styles. For instance, Cubism has been considered as a revolutionized artistic style for paintings and sculptures. Cubism has appeared since the early 20th century. Since then, several attempts have tried to describe [Henderson 1983] and generate Cubist art [Corker-Marin et al. 2018; Wang et al. 2011], but more efforts still required to offer scientific explanations to a wide variety of Cubist art. Our cubic stylization only focuses on a specific style. We hope this could inspire future attempts to capture different sculpting styles such as those presented in African art, or even a generic approach to create different styles in an unified framework.

ACKNOWLEDGMENTS
Our images and 3D models are used under Creative Commons. We obtain the 3D assets in Fig. 1 from sketchfab.com under CC Attribution. In particular, we cunate the textured meshes created by Katerina Novakova, 3dlogicus, Frédéric Cambon, akennedy007, germymd, and Jesús Orgaz. The images in Fig. 2 are from wikimedia.com and The Kiss (middle) was created by Constantin Brâncuși and photographed by Art Poskanzer under CC BY 2.0. The Nefertiti mesh in
Fig. 16 was scanned by Nora Al-Badri and Jan Nikolai Nelles from the Nefertiti bust. Images in Fig. 4 are from Antiques & Artifacts LLC. and wikimedia.com. The image in Fig. 7 is modified from the photo by puffin11k under CC BY-SA 2.0. We thank Aditya Aryanto for sharing the images of Anicube in Fig. 3. Other 3D meshes are obtained from the Thing10K [Zhou and Jacobson 2016] and cleaned with the method of [Hu et al. 2018].

Our research is funded in part by New Frontiers of Research Fund (NFRFE–201), the Ontario Early Research Award program, NSERC Discovery (RGPIN2017–05235, RGPAS–2017–507938), the Canada Research Chairs Program, the Fields Centre for Quantitative Analysis and Modelling and gifts by Adobe Systems, Autodesk and MESH Inc. We thank members of Dynamic Graphics Project at the University of Toronto; Michael Tao and Wen-Hsiang Tsai for project motivations; David J.W. Levin and Yotam Gingold for ideas on the artistic controls and the user study; Oded Stein for sharing results; Rahul Arora for fabricating stylized shapes; Leonardo Sach and Silvia Sellan for proofreading; Omid Pour saeed, Rahul Arora, Whitney Chiu, Yang Zhou, Yifan Wang, Youssef Alami Mejjati, and Zhicong Lu for participating the user study.

REFERENCES


A POLYHEDRAL GENERALIZATION

Fig. 28. By specifying different coordinate transformations \( B \) inside the \( \ell^1 \)-norm, we can encourage polyhedral style.

Simply applying a coordinate transformation \( B : \mathbb{R}^n \rightarrow \mathbb{R}^m \) inside the \( \ell^1 \)-norm can encourage polyhedral results, instead of cubic results (see Fig. 28). The \( \ell^1 \)-norm of a vector is defined as the summation of its magnitudes along each basis vector. Thus applying a coordinate transformation inside the \( \ell^1 \)-norm changes its behavior because the basis vectors are different. Following the notation in Eq. 1, polyhedron energy can be written as

\[
\min_z \sum_{i \in T} \sum_{j \in \mathcal{N}(i)} \frac{\|B_{ij} \cdot z_t - \tilde{D}_i\|_2^2 + \lambda \|B_{\hat{R}i\hat{n}_i}\|_1}{2}.
\]

In our case, \( B \) is a \( m \times 3 \) coordinate transformation matrix for shapes embedded in \( \mathbb{R}^3 \). Again by setting \( z = R_t \hat{n}_t \) we can reach almost the same optimization procedures, except the Eq. 5 now becomes (we ignore the iteration superscript for clarity)

\[
z^{k+1} = \arg \min_z \lambda \|Bz\|_1 + \frac{\rho}{2} \|R_t \hat{n}_t - z + u\|_2^2.
\]

Similar to common techniques for solving the Basis Pursuit problem, we introduce a variable \( t \geq \|Bz\|_1 \) to transform Eq. 10 into a small quadratic program subject to equality constraints

\[
\min_{z,t} \begin{bmatrix} z^T & t^T \end{bmatrix} \begin{bmatrix} \rho/2 & 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} z \\ t \end{bmatrix} + \left[-\rho(R_t \hat{n}_t + u)^T \lambda \hat{a}_1 \hat{t}_m \right] \begin{bmatrix} z \\ t \end{bmatrix} \leq 0,
\]

subject to \( \begin{bmatrix} B & -B \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z \\ t \end{bmatrix} \leq 0, \)

where \( I_3 \) and \( I_4 \) denote the identity matrix with size \( x \) and a column vector of 1 with size \( x \) respectively. We then solve this efficiently using CVXGEN [Mattingley and Boyd 2012]. Note that the results in Fig. 24 and Fig. 25 use \( m = 4 \).