Pacific Graphics 2023
R. Chaine, Z. Deng, and M. H. Kim
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 42 (2023), Number 7

Data-guided Authoring of Procedural Models of Shapes Supplementary Material

Ishtiaque Hossain[1] I-Chao Shen[2] Takeo Igarashi[2] Oliver van Kaick[1]

[1] Carleton University, Canada
[2] The University of Tokyo, Japan

1. Additional result of our method on 3D shapes

Since the procedural model used in our paper for the main results on 3D shapes is too complex for illustration purposes, in this section, we show an example of a complete result of our method obtained with a simplified procedural model. The procedural model used here combines primitive shapes such as cubes and cylinders to create tables. The primitive shapes are created with the Blender API for Python. Figure 1 shows the cube and the cylinder functions, which are parameterized by the location and the scale of the corresponding primitive.

For the set of reference shapes, we selected 20 shapes from the table category in ShapeNet. Figure 2 shows the selected shapes. The figure also shows how the clustering step of our method collects similar-looking shapes into groups for the user to inspect. Each group is assigned a different color. In this example, after looking at the groups in Figure 2, the user decided to write an initial procedural model that creates a rectangular table-top and four straight legs, which can be created using cubes at various locations and scales. Figure 3 shows the initial procedural model.

Figure 4 shows the result of one iteration of our method after using the initial procedural model written by the user. We see how our method identifies the compatible shapes, i.e. the reference shapes that can be replicated well using the initial procedural model, as well as the incompatible shapes. Good replications are colored green and other approximations are colored yellow. Again, similar shapes are grouped together and looking at these groups, the user decided to add an additional parameter and code to create rectangular or rounded table-tops. Figure 5 shows the second version of the procedural model after this change.

Figure 6 shows the second iteration of our method. While inspecting the groups of incompatible shapes, the user noticed that some tables have a support structure among the four legs and decided to introduce another parameter and code to specify the type of leg to be constructed. Figure 7 shows the revised procedural model that takes this change into account.

The next iteration of our method is shown in Figure 8. Among the groups of incompatible shapes, there are tables that have only one leg with a rounded base. The user then included this type of leg into the model and revised the procedural model accordingly. Figure 9 shows the fourth version of the procedural model.

Figure 10 shows the next iteration, where the only remaining incompatible shapes have another type of leg. Then, the user modifies the procedural model to accommodate the additional leg type. Figure 11 shows the fifth version of the procedural model.

Figure 12 shows the final iteration of our method, where all the reference shapes are replicated well. At this point, the procedural model has grown in both complexity and its ability to generate variations of tables similar to the reference shapes.

```python
import bpy


def cube(location=(0.0, 0.0, 0.0), scale=(1.0, 1.0, 1.0)):
    bpy.ops.mesh.primitive_cube_add(size=1.0, location=location, scale=scale)


def cylinder(location=(0.0, 0.0, 0.0), scale=(1.0, 1.0, 1.0)):
    bpy.ops.mesh.primitive_cylinder_add(radius=0.5, depth=1.0, location=location, scale=scale)
```

Figure 1: *Methods for creating cube and cylinder primitives.*

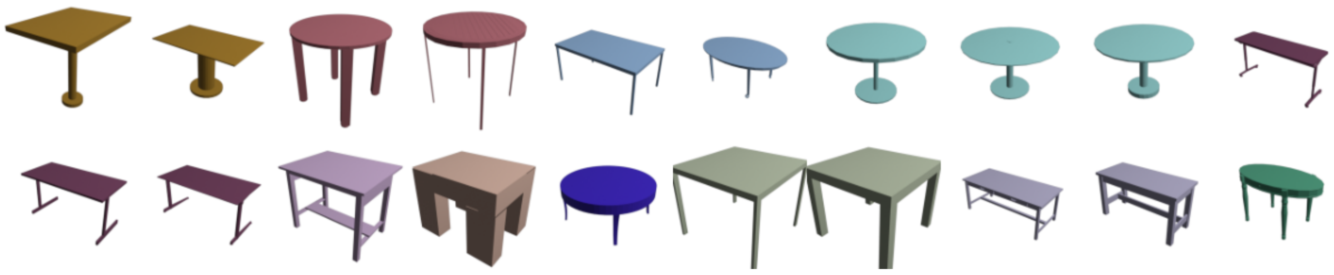

Figure 2: *Initial grouping performed automatically by our method on selected shapes from ShapeNet.*

```
def create_table(height, breadth, top_thickness, leg_thickness):
    h, b, t_th, l_th = height, breadth, height * top_thickness, breadth * leg_thickness
    # create table-top
    top_loc, top_scl = (0.0, 0.0, (h - t_th) / 2.0), (1.0, b, t_th)
    cube(location=top_loc, scale=top_scl)
    leg_scl = (l_th, l_th, h)
    # create four legs
    x = (1.0 - l_th) / 2.0
    y = (b - l_th) / 2.0
    leg_locs = [(x, y, 0.0), (x, -y, 0.0), (-x, y, 0.0), (-x, -y, 0.0)]
    for leg_loc in leg_locs:
        cube(location=leg_loc, scale=leg_scl)
```

Figure 3: *Initial procedural model.*

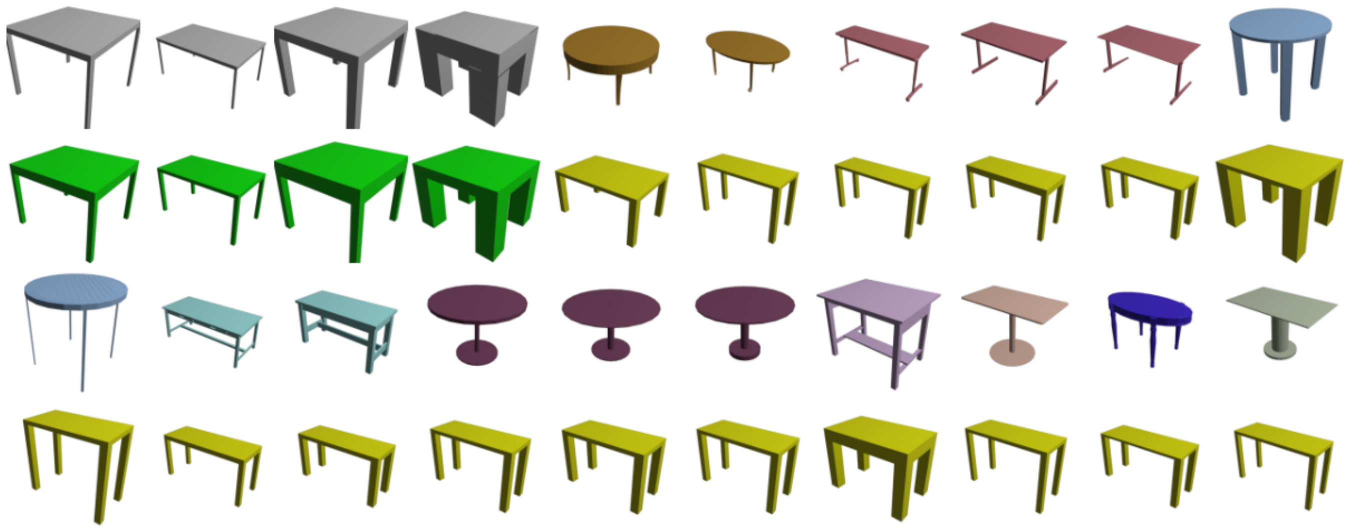

Figure 4: *Replication result using the initial version of the procedural model.*

```python
def create_table(height, breadth, top_thickness, leg_thickness, roundtop):
    h, b, t_th, l_th = height, breadth, height * top_thickness, breadth * leg_thickness
    rt = roundtop
    # create table-top
    top_loc, top_scl = (0.0, 0.0, (h - t_th) / 2.0), (1.0, b, t_th)
    if rt:
        cylinder(location=top_loc, scale=top_scl)
    else:
        cube(location=top_loc, scale=top_scl)
    leg_scl = (l_th, l_th, h)
    # create four legs
    a = 1.41 if rt else 1.0
    x = (1.0 - a * l_th) / (2 * a)
    y = (b - a * l_th) / (2 * a)
    leg_locs = [(x, y, 0.0), (x, -y, 0.0), (-x, y, 0.0), (-x, -y, 0.0)]
    for leg_loc in leg_locs:
        cube(location=leg_loc, scale=leg_scl)
```

Figure 5: *Second version of the procedural model.*

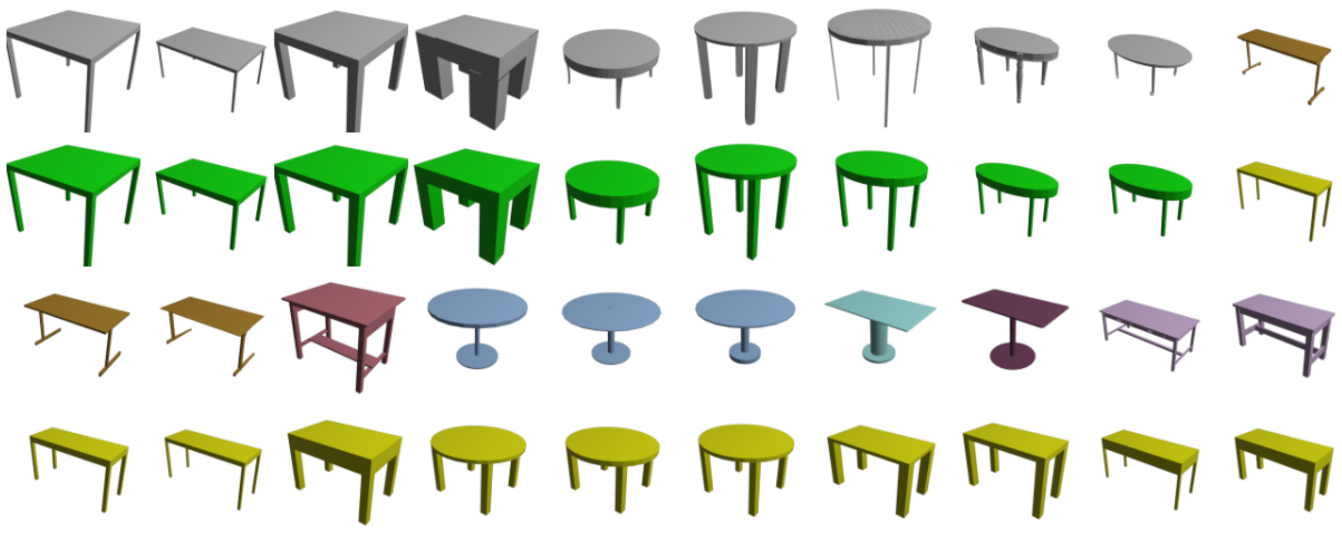

Figure 6: *Replication result using the second version of the procedural model.*

```python
def create_table(height, breadth, top_thickness, leg_thickness, roundtop, leg_type):
    h, b, t_th, l_th  = height, breadth, height * top_thickness, breadth * leg_thickness
    rt, l_tp = roundtop, leg_type
    # create table-top
    top_loc, top_scl = (0.0, 0.0, (h - t_th) / 2.0), (1.0, b, t_th)
    if rt:
        cylinder(location=top_loc, scale=top_scl)
    else:
        cube(location=top_loc, scale=top_scl)
    leg_scl = (l_th, l_th, h)
    # create four legs
    a = 1.41 if rt else 1.0
    x = (1.0 - a * l_th) / (2 * a)
    y = (b - a * l_th) / (2 * a)
    leg_locs = [(x, y, 0.0), (x, -y, 0.0), (-x, y, 0.0), (-x, -y, 0.0)]
    for leg_loc in leg_locs:
        cube(location=leg_loc, scale=leg_scl)
    # create support between legs
    if l_tp == 'support':
        if rt:
            leg_sprt_scl = (l_th, b / a, l_th)
        else:
            leg_sprt_scl = (l_th, b, l_th)
        cube(location=(x, 0.0, -h / 3.0), scale=leg_sprt_scl)
        cube(location=(-x, 0.0, -h / 3.0), scale=leg_sprt_scl)
        cube(location=(0.0, 0.0, -h / 3.0), scale=((1.0 - a * l_th) / a, l_th, l_th))
```

Figure 7: *Third version of the procedural model.*

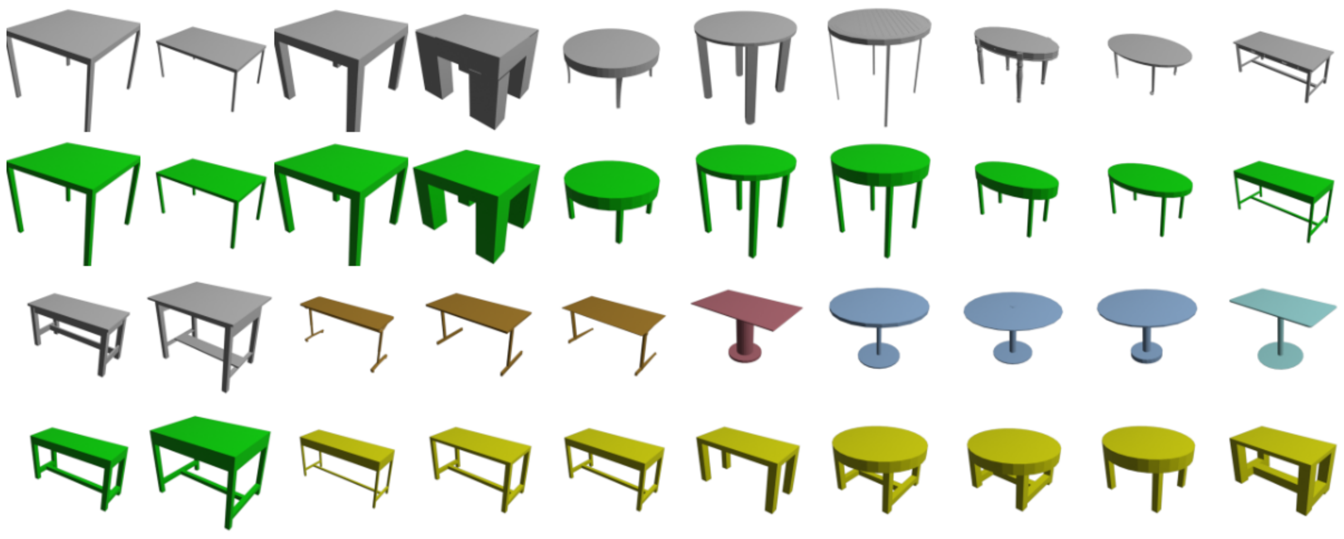

Figure 8: *Replication result using the third version of the procedural model.*

```python
def create_table(height, breadth, top_thickness, leg_thickness, roundtop, leg_type):
    h, b, t_th, l_th  = height, breadth, height * top_thickness, breadth * leg_thickness
    rt, l_tp = roundtop, leg_type
    # create table-top
    top_loc, top_scl = (0.0, 0.0, (h - t_th) / 2.0), (1.0, b, t_th)
    if rt:
        cylinder(location=top_loc, scale=top_scl)
    else:
        cube(location=top_loc, scale=top_scl)
    leg_scl = (l_th, l_th, h)
    if l_tp == 'round':
        # create rounded leg
        cylinder(location=(0.0, 0.0, 0.0), scale=leg_scl)
        cylinder(location=(0.0, 0.0, -(19.0 * h) / 40.0), scale=(0.5, 0.5, h / 20.0))
    else:
        # create four legs
        a = 1.41 if rt else 1.0
        x = (1.0 - a * l_th) / (2 * a)
        y = (b - a * l_th) / (2 * a)
        leg_locs = [(x, y, 0.0), (x, -y, 0.0), (-x, y, 0.0), (-x, -y, 0.0)]
        for leg_loc in leg_locs:
            cube(location=leg_loc, scale=leg_scl)
        # create support between legs
        if l_tp == 'support':
            if rt:
                leg_sprt_scl = (l_th, b / a, l_th)
            else:
                leg_sprt_scl = (l_th, b, l_th)
            cube(location=(x, 0.0, -h / 3.0), scale=leg_sprt_scl)
            cube(location=(-x, 0.0, -h / 3.0), scale=leg_sprt_scl)
            cube(location=(0.0, 0.0, -h / 3.0), scale=((1.0 - a * l_th) / a, l_th, l_th))
```

Figure 9: *Fourth version of the procedural model.*

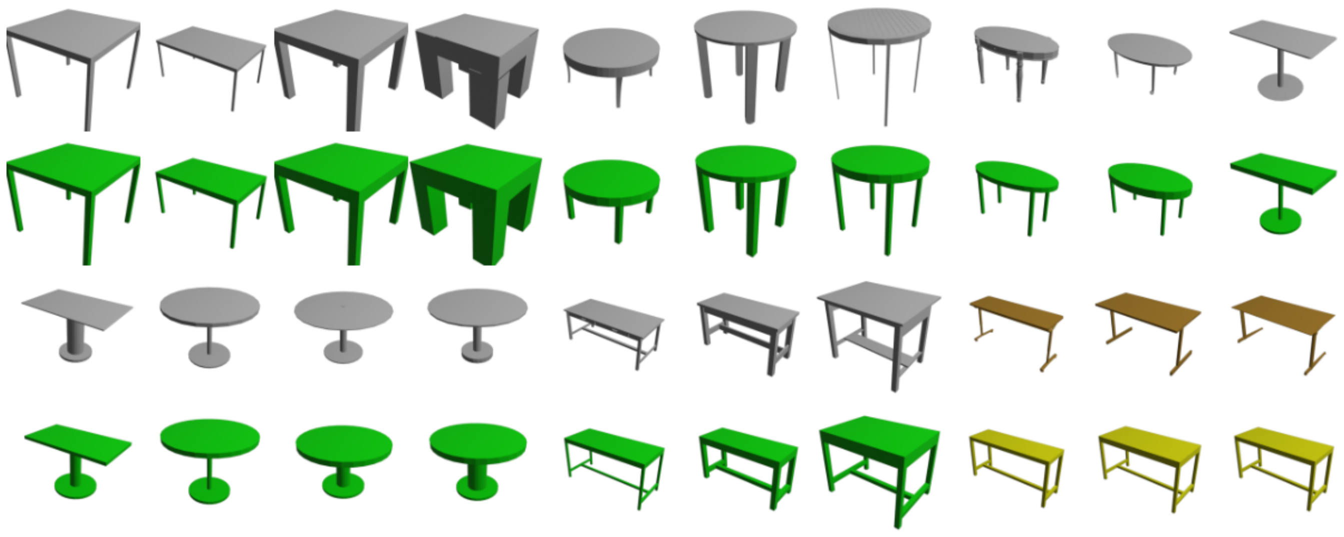

Figure 10: *Replication result using the fourth version of the procedural model.*

```
def create_table(height, breadth, top_thickness, leg_thickness, roundtop, leg_type):
    h, b, t_th, l_th  = height, breadth, height * top_thickness, breadth * leg_thickness
    rt, l_tp = roundtop, leg_type
    # create table-top
    top_loc, top_scl = (0.0, 0.0, (h - t_th) / 2.0), (1.0, b, t_th)
    if rt:
        cylinder(location=top_loc, scale=top_scl)
    else:
        cube(location=top_loc, scale=top_scl)
    leg_scl = (l_th, l_th, h)
    if l_tp == 'round':
        # create rounded leg
        cylinder(location=(0.0, 0.0, 0.0), scale=leg_scl)
        cylinder(location=(0.0, 0.0, -(19.0 * h) / 40.0), scale=(0.5, 0.5, h / 20.0))
    elif l_tp == 'split':
        # create split legs
        for x in [0.5 - l_th / 2.0, l_th / 2.0 - 0.5]:
            cube(location=(x, 0.0, 0.0), scale=(l_th, l_th, h))
            cube(location=(x, 0.0, (l_th - h) / 2.0), scale=(l_th, b, l_th))
    else:
        # create four legs
        a = 1.41 if rt else 1.0
        x = (1.0 - a * l_th) / (2 * a)
        y = (b - a * l_th) / (2 * a)
        leg_locs = [(x, y, 0.0), (x, -y, 0.0), (-x, y, 0.0), (-x, -y, 0.0)]
        for leg_loc in leg_locs:
            cube(location=leg_loc, scale=leg_scl)
        # create support between legs
        if l_tp == 'support':
            if rt:
                leg_sprt_scl = (l_th, b / a, l_th)
            else:
                leg_sprt_scl = (l_th, b, l_th)
            cube(location=(x, 0.0, -h / 3.0), scale=leg_sprt_scl)
            cube(location=(-x, 0.0, -h / 3.0), scale=leg_sprt_scl)
            cube(location=(0.0, 0.0, -h / 3.0), scale=((1.0 - a * l_th) / a, l_th, l_th))
```

Figure 11: *Fifth version of the procedural model.*

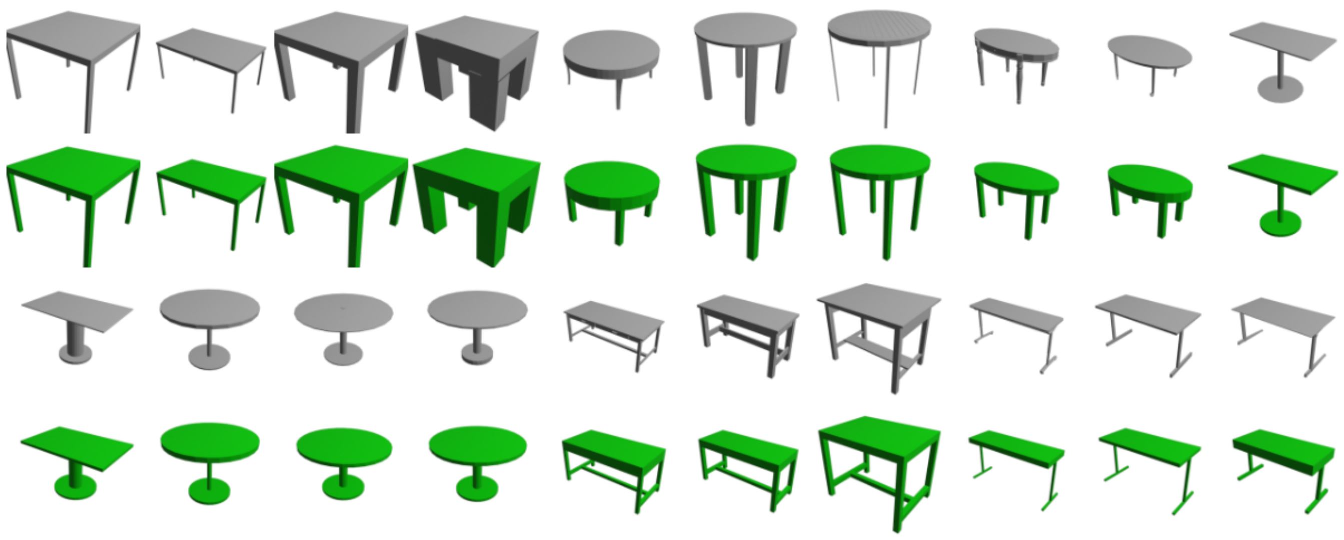

Figure 12: *Replication result using the fifth version of the procedural model.*