Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 3, No. 2, 2023

Procedural Modeling Method of Pipe Network Based on Spline Grammar

Xiaolong Wang*

School of Software Engineering, Chengdu University of Information Technology, Chengdu 610225, China
* **Corresponding author Email:** microlone@163.com

Abstract: The shape grammar is usually used to describe the three-dimensional model with certain regularity, but in the process of constructing the three-dimensional model of industrial pipeline network or urban underground pipeline network, due to the complex wiring characteristics of the pipeline, the pipeline with different changing trajectories is often unable to be described by fixed shape. Therefore, the construction process for 3D pipe networks often requires art developers to spend a lot of time and effort to develop models. Arbitrarily shaped pipes can be effectively described in procedural modeling construction processes by incorporating splines that are inherently variable in nature and combining them as terminator shapes into shape grammar. Based on this theory, this paper proposes that the procedural modeling method based on spline grammar can efficiently complete the procedural construction of pipeline networks, and this method can quickly complete the splicing of pipeline networks. Experimental results show that our method is suitable for pipe network construction projects in different fields.

Keywords: Pipe network; Spline grammar; Programmatic modeling.

1. Introduction

The procedural modeling method of parametric spline grammar proposed in this paper is similar to the traditional parametric modeling technique, which is a technique for constructing three-dimensional models by graphical algorithm programming. It allows users to define model parameters and relationships by dragging and connecting different components, and the type description of the shape of the pipeline is defined by the input of parameters without additional editing code.

However, unlike the traditional procedural modeling method based on parametric shape grammar, this paper proposes to add splines as a special shape to the grammar rules and derive the parametric formal grammar as a terminator. At the same time, after removing the spline terminator in the grammar structure, the remaining terminators adaptively calculate their pending parameters (such as intermediate filling node coordinates and compact coefficients) according to the generation of spline terminators and generate pipe fiettors, and finally complete the splicing of pipe body and pipe fittings. This method can reduce the workload of users to adjust secondary parameters after the initial generation of the pipe network model through the procedural logic determination of spatial geometry, which greatly simplifies the operation process and improves the work efficiency.

The main contributions of this paper are as follows:

i. A procedural pipeline modeling method based on spline grammar is proposed. In this paper, the pipe network can be atomized by combining the basic geometry of parametric shape grammar, straight line segment, and different types of spline curves. A parametric description of the operation.
ii. Design two general-purpose pipe caps as shape terminators and introduce them into the grammatical structure. The cap grid is programmed and automatically constructed without manual adjustment by the user.

2. Related Work

2.1. Shape Grammar Based Modeling Methods

Wonka et al. [1] proposed a split grammar for modeling buildings of different styles through automatic derivation. Zmugg et al. [2] improved the split grammar so that it can be applied to surface architectural modeling, and the authors generated pagoda and Great Wall models based on the former method. Edelsbrunner [3] further improved the splitting grammar by specifying each part of the generated building model as an independent coordinate system, and then applying the splitting operation along these coordinate systems. The researchers also added Boolean operations to split the grammar to describe more complex building shapes. Muller et al. [4] proposed CGA shapes, which extend the split grammar. The CGA-based modeling system can automatically generate architectural models of various shapes according to the rules written by users. Schwarz et al. [5] introduced the CGA++ grammar language, which adds context sensitivity to CGA shapes so that combinations of multiple buildings can yield better results. In addition, the inverse process modeling method [6-10] has also been widely used. Wu et al. [10] proposed a method to automatically derive a split grammar for a given facade layout (segmented facade images), and then generate new buildings based on the derived grammar. Demir et al. [7] proposed a guided inverse procedural modeling approach to analyze existing input building models to obtain a set of shape grammars, and then synthesize new building models based on these shape grammars under the guidance of the user. Shape grammar-based modeling methods can produce results that meet user expectations, but their shortcomings are also evident when used to model pipeline networks. First of all, the shape grammar is better at generating blocky architectural models that lack details, and the wiring of the pipeline network is extremely complex, and the method based on the shape grammar is usually unable to generate a large number of models that satisfy the pipeline curve expression. Second, non-professional users are usually unable to create and modify complex modeling grammar rules.

2.2. Spline-Based Modeling Methods

Different from the above methods, Huang et al. [11] proposed a novel spline-based interactive and procedural modeling method. The method first provides users with a standardized spline-based frame, and users edit the frame through two types of control points: Position Control Points (Position Control Points) and Bending Control Points (Bending Control Points). Once the user finishes editing, the modeling system instantiates prefabricated components from the component library and places them one by one on the frame to obtain the target model. Huang et al. [12] continued to improve their method by making walls and roofs, multi-storey buildings and composite buildings on the basis of cutting and combining. However, the improved system still has significant limitations. Users can only edit frames, not components. Also, since the components in the component library are still grid-based, the file size of the library is still large. Some auxiliary structures that are common in architectural models, such as fences, column brackets, etc., are not defined in the above system.

3. Modeling Method

Parametric spline grammar is a method of generating shapes using parametric splines. A parametric spline is a vector-valued function defined by a set of control points and a function, which can describe complex curved shapes. However, different from the traditional way of describing the target curve by approximation and fitting by a single spline object, this paper combines the rational splicing of several basic geometries, such as parametric shape grammar, straight line segments, and different types of spline curves. A parametric description of the pipe network that can be atomically operated.

First of all, the method proposed in this paper based on the parametric spline grammar to generate the pipe network model is divided into two categories: one is the pipe body (the skeleton category includes straight line, Bézier spline, Catmull-Rom spline and B-Spline spline); the other type is connecting pipe fittings (the s-keleton category includes straight line and B-Spline spline); the above two parts are combined by the user, and the construction of the procedural pipe network is completed through parametric grammar.

The basic overall process of modeling based on parametric spline grammar is as follows:

i. Predefine the initial shape parameters of the pipe body and fittings;
ii. Read the node information of the pipe network and generate relevant basic skeleton splines;
iii. The connection node between the pipe body and the pipe fitting completes the automatic adsorption connection through the collision detection;
iv. The pipe body and pipe fittings complete the model generation of the pipe network according to the configuration parameters of the extruder.

3.1. Spline Grammar

In this paper, the system divides the pipe network into two categories, namely pipe body (Pipe Body) and pipe fitting (Pipe Fitting). The pipe body is subdivided into four types of terminators including Linear, Bézier, Catmull-Rom, and B-Spline; similarly, pipe fittings are also subdivided into two types of terminators including Linear and B-Spline. Therefore, the grammar structure based on parametric spline grammar is shown in Fig. 1. The execution process of the spline grammar is: according to the input parameter rules, use the specified shape to deduce the parent structure until all non-terminal symbols are converted into terminal symbols.

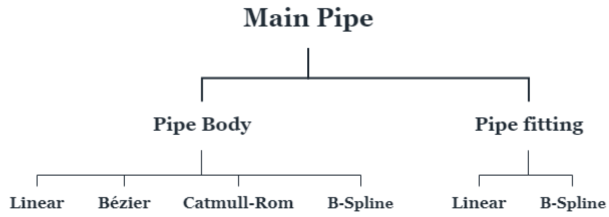


Figure 1. Parametric spline shape hierarchy

The grammatical structure involved in this article is inspired by the shape grammar, so the grammatical structure is similar to the shape grammar, and it is described by the quaternion expression G, as shown in Exp.1.

$$G = \{I, N, T, R\} \tag{1}$$

For the above formula combined with the spline-based grammar structure proposed in this paper, a detailed elaboration is given:

i. where I is the initial shape formed by the input starting point set;
ii. N represents the shape of a non-terminal, N = {B, T}. Contains pipe structure (Body) and pipe element (Fitting);
iii. T represents the shape of the terminator, which represents the final pipe spline (straight line, curve) and accessory elements (basic pipe fittings, ring fasteners, pipe supports, valves, etc.);
iv. R is a finite set of production rules. The rules have the general form $A \rightarrow B$: cond, $A \in N$, $B \in V$, where V is a grammatical vocabulary defined as $N \cup T$. A condition is a logical expression that must be true for a rule to be invoked. This rule can be understood as shape A being replaced by shape B under the condition cond.

In the initial stage of construction, the terminal shape P = {B | F} is formed by the unit spline; where B is the type of the spline, and F is the type of the pipe fitting to which the spline belongs. Spline types include straight line (Linear), curve (Bezier, B-Spline, Catmull-Rom). Fitting types include Linear and B-Spline.

3.2. Rules for Pipe Body

According to the grammatical rules, the initial shape I is transformed to derive the expression P containing non-terminal symbols. The conversion rules are shown in Formula .2:

$$R_{\text{transf}} : I \rightarrow N\text{:cond}(\mathrm{H} \neq 0) \tag{2}$$

The variable h represents the compactness coefficient, which represents the distance parameter between the pipe body and the pipe body that can accommodate the pipe fittings. The final shape defined by the terminal is further derived by the above formula.

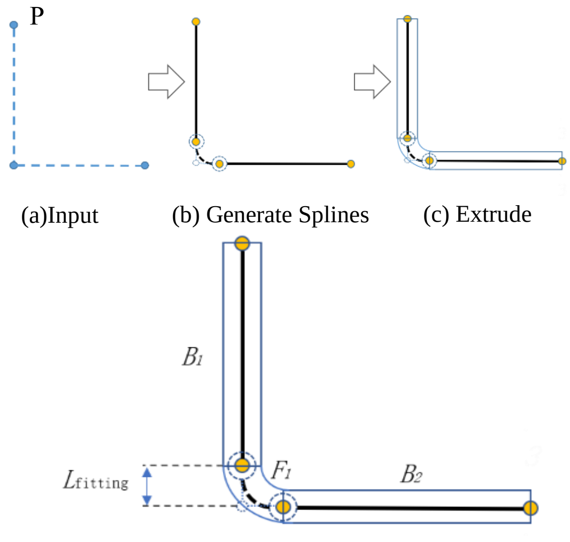
The terminal shape Presul represents the alternate splicing and combination of the pipe body and the pipe fitting structure, where 1 : B and 1 : F represent the number of elements of the pipe body and pipe fittings; the symbol * indicates that the shape based on this expression is continuously copied and extended in space according to the terminal shape, until all input shapes read have been transformed and derived.

$$Presul = \{1 : B \mid 1 : F\}* \tag{3}$$

$$Pos = P_2 + h * Drec * Lengh_{linear} \tag{4}$$

As shown in Fig. 2, execute after the user enters key point data:

i. Add two new nodes to the remaining key points except the starting point and the ending point;
ii. According to the compact coefficient h input by the user, the E_1 node generated in step (i) is offset along the trajectory ($\overrightarrow{P_2P_1}$) direction, as shown in (b) in Fig. 2; the S_1 node is along the trajectory ($\overrightarrow{P_2P_3}$) direction The offset value is h * Lenghlinear; the spatial coordinates of Si and Ei are related to the compactness coefficient $h \in (0,1)$. Among them, h is defined by the user, and the position of any coordinate axis after the offset can be obtained as shown in formula. 4. The direction vector Drec is directed from the intermediate node P_2 to the edge node (P_1 or P_3) of the current spline. Through this method, the spatial coordinates of the newly generated nodes S_3 and E_1 are calculated respectively.
iii. Input the calculated two groups of control points P_1, E_1, S_2, and P_3 into the extruder according to the pipe body terminator predefined by the user to extrude the Presul based on the section contour projection algorithm, and finally complete the pipe body model generation as shown in Fig. 3 shown.



(a)Input (b) Generate Splines (c) Extrude

(d) Terminal shape example

Figure 2. Schematic diagram of derivation conversion process

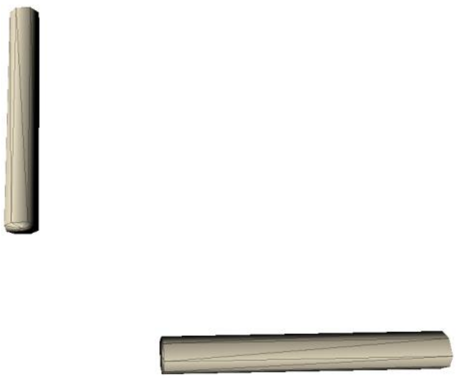


Figure 3. Example of pipe model

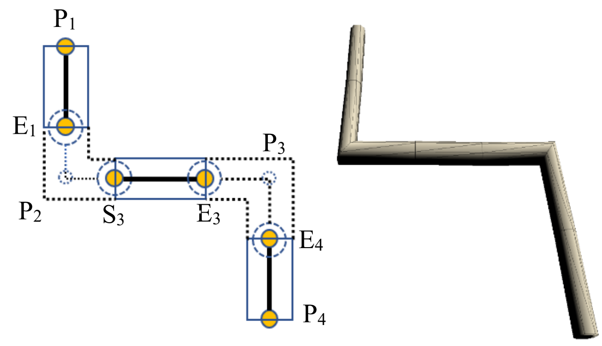
When users need other types of pipe bodies, they can manually switch the spline type of pipe body objects in the editor mode. In most cases, the system generates straight pipe bodies by default, which is suitable for pipe network construction in most environments.

3.3. Rules for Pipe Fitting

The pipe body uses the spline skeleton as the basic skeleton. In order to enable the pipe fittings to automatically generate a reasonable straight line or curved skeleton after the pipe body spline is generated, B-Spline (B-Spline) and straight line (Linear) are introduced as the skeleton of the pipe fittings. Input to the extruder subdivision point basis. With the help of BVH collision detection, the adjacent nodes are collided and adsorbed to complete the splicing of the pipe fittings and the adjacent nodes of the pipe body. Next, this section will give a detailed description of the generation of the expression describing the pipe skeleton based on the pipe body spline object.

3.3.1. Straight Fittings

In this paper, the effective part of the key points input by the user is used as the intermediate control point of the linear pipe, and the extrusion algorithm based on the profile projection is also used to generate the pipe model. The intermediate nodes {E1, P2, S3}, {E3, P3, S4} obtained after the previous derivation can be directly used to generate straight pipe fittings, and the extruder curve type used here is consistent with that used for the pipe body, as shown in Fig. 4.



(a) Node Adaptive Construction (b) Linear Tube

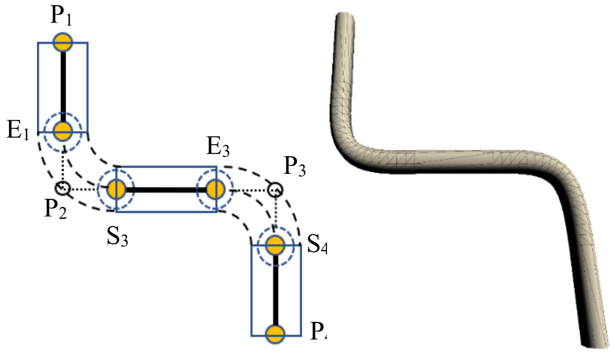
Figure 4. Examples of straight fittings

3.3.2. B-Spline Fittings

To use the profile projection extruder to generate B-spline pipe fittings, analyze it from the perspective of reverse engineering, we know that the final output is the subdivision point set of B-spline geometric trajectory. We need to analyze the B-spline geometry that can be matched by the generated pipe body, so the problem is transformed into the coordinates of the starting point, middle point and end point of the B-spline space based on the known middle node coordinates P_2 and P_3. B-spline curve trajectory subdivision point set.

According to the derivation result of the spline grammar, as shown in Figurg. 5, input {E1, P2, S3}, {E3, P3, S4} into the extruder and set the CurveType to B-Spline. The final model effect is shown in (b).

In summary, although the skeleton splines used by the two pipe fittings are different, they both extrude the pipe fitting model through the filling node in the middle as the control point.



(a) Node Adaptive Construction (b) B-Spline fitting

Figure 5. Arc pipe generation

4. Cap Mesh

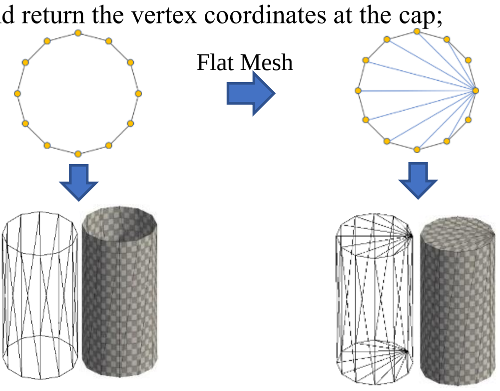
In actual project development, pipe meshes are often used to occupy as few resources as possible. During the model design and construction phase, meshes are often designed for single-sided rendering. After completing the generation of the basic mesh of the pipeline, in order to make the pipeline more reasonable visually, it is necessary to cap the first and last ends of the pipeline. Among them, for industrial pipe network or urban underground pipe network, according to the geometric shape of the head, it is divided into two common head types: flat head and convex head. Among them, the convex head is subdivided into spherical, elliptical, dish-shaped, and spherical-shaped heads. Next, the generation methods of these two types of head grids will be described in detail.

4.1. Flat Cap

The flat head is a relatively common head type, and the caping area is a closed polygon in the plane, and its shape is consistent with the pipe section. Similar to the pipe extrusion method, the calculation of vertices in the cap plane is also based on the section profile profile.

The mesh generation of the flat head is relatively simple. It only needs to use the vertex of the profile contour as the head polygon point and generate the triangle index array and other mesh attributes in order. The general steps are as follows:

i. Read the starting point and ending point of the currently operated tube object to determine the position coordinates of the caping center point;
ii. Read the configuration file of the section profile, use the center point of the cap as the vertex offset calculation and return the vertex coordinates at the cap;



(a) primitive tube (b)Flat cap tube

Figure 6. Constructing a flat-capped pipe mesh

iii. Generate a triangle array in order from the returned vertices;
iv. Complete the caping UV mapping calculation.

The overall construction process is shown in Fig. 6, and the relevant codes are shown in Table 1 below.

Table 1. Flat Cap mesh generation code

Description: Generate code for flat cap mesh

```
void GenerateFlatCaps()
{
    //Cap processing
    GetSample(index, ref evalResult); // Get the coordinates of the center point of the head and tail section
    for (int i = 0; i < _sides+1; i++)
   {
        // The cap vertices are assigned according to the contour points of the section;
        int index = bodyVertexCount + i;
        _tsMesh.vertices[index] = _tsMesh.vertices[i];
        _tsMesh.normals[index] = -evalResult.forward;
        _tsMesh.colors[index] = _tsMesh.colors[i];
        _tsMesh.uv[index] = Quaternion.AngleAxis(_revolve * (((float)i) / (_sides - 1)), Vector3.forward) * Vector2.right * (0.5f * capUVScale) + Vector3.right * 0.5f + Vector3.up * 0.5f;//Calculate UVs
    }
    _tsMesh.triangles = GenerateTrians();//Generate flat cap triangular index array}
```

4.2. Convex Cap

The general process of mesh calculation of convex head is similar to that of flat seal, but it needs to calculate the vertex coordinates by uniform interpolation from the outside to the inside of the section profile, and at the same time, in order to get a good visual effect, it is necessary to calculate the vertices on the adjacent triangles. Calculate the average value of the line.

First define the parameter _roundCapLatitude (Sides) of the degree of subdivision of the hemisphere and the diameter of the pipe to calculate the offset of the convex grid vertices and the grid UV. Here, the regular dodecagon is used as the profile of the section as an example. The convex head is For hemispherical type, the head vertex is calculated according to the two levels of Latitude and Longitude. Among them, lat∈[1, Latitude+1] indicates the number of layers of the current latitude line; as shown in Formula (5), $angle_{lat}$ indicates the clamp formed by the offset vertex, the vector of the center point of the head, and the bottom plane of the head Angle size; as shown in Formula (6), lon∈[0, Sides] indicates the number of layers of the current meridian; $angle_{lon}$ indicates the projection formed by the offset vertex and the vector of the center point of the head and the bottom plane of the head The size of the angle formed by the vector and the zero vector;

$$angle_{lat} = \frac{lat}{\text{Latitude+1}} * 90 \tag{5}$$

$$angle_{lon} = \frac{lon}{\text{Sides}} * 360 \tag{6}$$

Next, the rotation function Rot is defined by the quaternion-based rotation formula to rotate the target vertex. The $Rot(\vec{u}, \theta)$ function is used to calculate the quaternion after rotating θ around $\vec{u}$. Based on formulas (5) and (6), the formula (7) for calculating the target vertex vector $\vec{p}$ can be deduced, where $(\overrightarrow{P_{center}})$ is the coordinates from the origin of the space to the cap center point; $\vec{R}$, $\vec{F}$ and $\vec{U}$ represent three-dimensional space vector constants Vector3 (1, 0, 0), Vector3 (0,0,1) and Vector3 (0, 1, 0) respectively.

$$\vec{P} = \overrightarrow{P_{center}} + Rot(-\vec{F}, angle_{lon} + 180) * Rot(\vec{U}, angle_{lat}) * \vec{R} \quad (7)$$

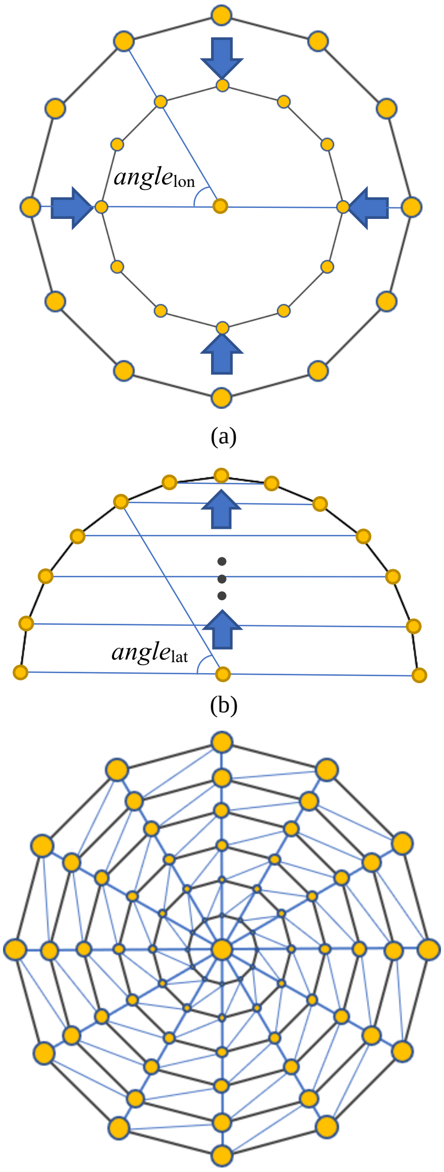


Figure 7. An example diagram of the principle of constructing a convex head grid, (a) means to return to each lat layer vertex by inward offset, (b) means to return to each lon layer vertex by offsetting to the far end of the head, (c) means to traverse lat * Vertex and triangle calculation results after lon times

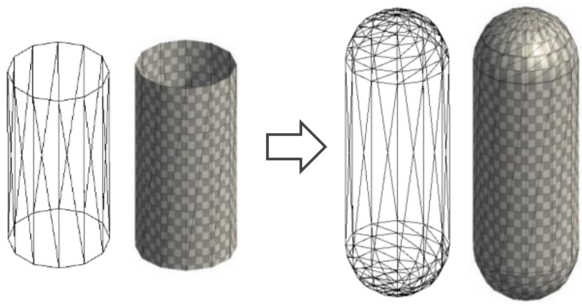


Figure 8. Convex cap mesh

Finally, the convex head grid as shown in Fig. 8 is obtained, and the relevant code is as follows:

Table 2. Convex Cap mesh generation code

Description: Generate code for convex cap mesh

Input: spline subdivision point evalResult; section node; subdivision parameter _roundCapLatitude; path size

Output: Cap Mesh _tsMesh

```
void GenerateRoundCaps()
{
    GetSample(index, ref evalResult);
    Computes the interpolated position of the convex start point in the spline startV;
    capLengthPercent = size * 0.5f;
    for (int lat = 1; lat < _roundCapLatitude+1; lat++)// Loop latitude
    {
        float latitudePercent = ((float)lat / _roundCapLatitude);// Current latitude ratio
        float latAngle = 90f * latitudePercent; // Current latitude
        for (int lon = 0; lon <= sides; lon++)// Loop longitude
        {
            float anglePercent = (float)lon / sides; // current longitude ratio
            int index = bodyVertexCount + lon + (lat-1) * (sides + 1);
            Quaternion rot = Quaternion.AngleAxis(_revolve * anglePercent + rotation + 180f, -Vector3.forward) * Quaternion.AngleAxis(latAngle, Vector3.up);// Calculate the offset quaternion
            _tsMesh.vertices[index] = center + lookRot * rot * -Vector3.right * (size * 0.5f * evalResult.size);
            _tsMesh.normals[index] = (_tsMesh.vertices[index] - center).normalized;
            float baseV = startV + capLengthPercent * latitudePercent;
            Vector2 baseUV = new Vector2(anglePercent * uvScale.x - baseV * _uvTwist, baseV * uvScale.y) - uvOffset;
            _tsMesh.uv[index] = Vector2.one * 0.5f + (Vector2)(Quaternion.AngleAxis(uvRotation + 180f, Vector3.forward) * (Vector2.one * 0.5f - baseUV));
        }
        _tsMesh.triangles = GenerateTrians();//Generate flat cap triangular index array
    }
}
```

5. Conclusion

A pipe network modeling method based on parametric spline grammar is proposed. Firstly, the grammar structure and derivation rules based on parametric spline grammar are proposed and designed. The replacement derivation relationship between non-terminal symbols and terminal symbols is detailed elaborate. Then it introduces how to use the key point set input by the user to generate the combined splicing model of ordered pipe body and pipe fittings by deriving rules, and expounds the logic of its generation algorithm. Secondly, the cap mesh algorithm is designed and implemented for the pipe body and pipe fittings to realize the automatic generation of the cap mesh, as shown in Fig. 9.

This method is a new application of the idea of shape grammar, which can properly simplify the operation process of manual modeling and has certain scalability.

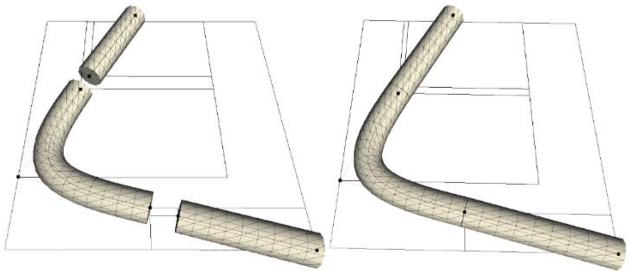


Figure 9. Pipeline Model Generated Based on Spline Grammar

References

[1] Peter W, M Wimmer, FX Sillion, W Ribarsky (2003) Instant architecture. ACM T Graphic 22(4):669- 677.

[2] R Zmugg,W Thaller,U Krispel,J Edelsbrunner,S Havemann,D W. Fellner. Procedural architecture using deformation-aware split grammars[J]. The Visual Computer,2014,30(9).

[3] J Edelsbrunner,S Havemann,A Sourin, W Dieter. Fellner. Procedural modeling of architecture with round geometry[J]. Computers & Graphics,2017,64.

[4] P Müller,P Wonka,S Haegler,A Ulmer,L V Gool. Procedural modeling of buildings[J]. ACM Transactions on Graphics (TOG), 2006, 25(3).

[5] M Schwarz,P Müller. Advanced procedural modeling of architecture[J]. ACM Transactions on Graphics (TOG), 2015, 34(4).

[6] S Barroso,G Besuievsky,G Patow. Visual copy & paste for procedurally modeled buildings by ruleset rewriting[J]. Computers & Graphics,2013,37(4).

[7] İ Demir,D G. Aliaga. Guided proceduralization: Optimizing geometry processing and grammar extraction for architectural models[J]. Computers & Graphics,2018,74.

[8] I Martin,G Patow. Ruleset-rewriting for procedural modeling of buildings[J]. Computers & Graphics,2019,84(C).

[9] Martinovic, A Van Gool, L. Bayesian Grammar Learning for Inverse Procedural Modeling[],2013.

[10] F Wu,D.-M Yan,W Dong,X.-P Zhang,P Wonka. Inverse procedural modeling of facade layouts[J]. ACM Transactions on Graphics (TOG),2014,33(4).

[11] C.-Y Huang,W.-K Tai. Ting tools: interactive and procedural modeling of Chinese ting[J]. The Visual Computer, 2013, 29(12).

[12] C.-Y Huang, W.-K Tai. Interactive and Procedural Modeling of Featured Chinese Architectures [],2015.