PM4VR: A Scriptable Parametric Modeling Interface for Conceptual Architecture Design in VR

Wanwan Li
University of South Florida
Tampa, Florida, USA

Figure 1: This figure shows the results of interactive parametric modeling on conceptual architecture designs using PM4VR. Users are able to use simple code (shown in the text editor) to program the conceptual architecture designs with a simplified programming language called Java♭ which is developed by us. Through PM4VR, users can interactively tune the design parameters using VR controllers in immersive VR environments. In this figure, the virtual urban environment is automatically loaded through OpenStreetMap API which is matching with the real world located nearby the Zhujiang New City Tower, Tian He Qu, Guang Zhou Shi, China. Through our integrated interface of PM4VR, users can create conceptual architecture parametric designs with immersive interactive experiences to tune the parameters in a realistic virtual environment.

ABSTRACT

In this paper, we propose PM4VR, a novel scriptable parametric modeling interface for the Unity3D game engine which can be applied to VR-driven parametric modeling designs. By simplifying prevailing advanced programming languages such as C# and Java, we propose another programming language, named Java♭, to simplify the grammar and lower the programmer's learning curve. By implementing a series of advanced parametric modeling techniques, we integrate our Java♭ compiler virtual machine with those functionalities which can facilitate interactive parametric modeling design process on the Unity3D game engine within immersive SteamVR environments. More specifically, in this paper, we introduce the Java♭ programming language, explain the implementation details of Java♭ compiler virtual machine, and discuss the experimental results of the interactive parametric modeling on conceptual architecture designs using PM4VR. Besides, a Supplementary Material with Java♭ programming examples is included.

VRCAI '22, December 27–29, 2022, Guangzhou, China

ACM ISBN 979-8-4007-0031-6/22/12…$15.00
https://doi.org/10.1145/3574131.3574442

CCS CONCEPTS

• **Computing methodologies** → **Shape modeling**.

KEYWORDS

Parametric Modeling, Virtual Reality, Conceptual Architecture Design, Programming Language, Compiler and Virtual Machine

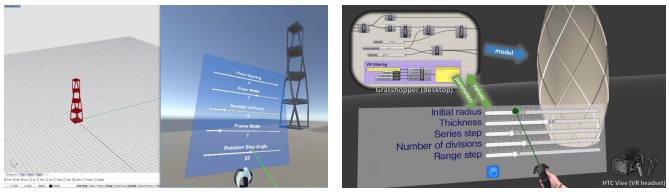
ACM Reference Format:
Wanwan Li. 2022. PM4VR: A Scriptable Parametric Modeling Interface for Conceptual Architecture Design in VR. In *The 18th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry (VRCAI '22), December 27–29, 2022, Guangzhou, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3574131.3574442

1 INTRODUCTION

With the rapid development of conceptual architectural design, CAGD (Computer Aided Graphics Design) [Shivegowda et al. 2022] has become an indispensable application tool for architects. Since Robert McNeel & Association first released Rhinoceros 3D [Omid and Golabchi 2020], a 3D development software with NURBS surface parametric modeling [Stavric and Marina 2011] as its main feature, such innovation has been revolutionary. First of all, it introduces the parametric equation representation of NURBS surfaces as an alternative to the traditional 3D modeling approaches which are based on dragging and dropping NURBS control points. Secondly, by writing the RhinoScript programs through the Kangaroo plug-in [Asefi and Bahremandi-Tolou 2019], developers can efficiently develop complex 3D surfaces by adjusting parameter settings. Since then, script-based parametric modeling approaches have gain more popularity within the conceptual architecture design industry.

However, due to the complex grammar of RhinoScript, its learning curve for beginners becomes extremely high. Sure enough, such a problem was noticed by David Rutte, who developed a visual programming plug-in called Grasshopper 3D [Fink and Koenig 2019] through which designers can program easily by dragging "battery"-like controls and connecting those controls with different logic flows. In this way, developers only need to care about the function of each control without requiring too many programming skills. However, when reading and debugging a large-scale visual program including hundreds of controls, understanding each control and the logical relationship between components becomes an extremely complicated thing. Firstly, mastering the functions of a large number of controls is actually not simpler than learning an easy programming language; Secondly, reading the program logic in a way of "batteries" and "circuits" is far less straightforward than reading program code directly. Therefore, although Grasshopper 3D is a powerful tool, it does not simplify the programming either.

On the other hand, as Virtual Reality (VR) technologies are gaining popularity among this generation, directly adjusting the parametric conceptual architecture designs in the virtual immersive environment is more inspiring for creative design. Figure 2 shows some existing works, which include Hawton et al. [Hawton et al. 2018], Coppens et al. [Coppens et al. 2019], Castelo et al. [Castelo-Branco et al. 2019; Castelo-Branco and Leitão 2022], and Archimatix Pro [Productions 2021], etc., that are building bridges between the existing parametric modeling tools such as Grasshopper 3D with the virtual immersive VR environments provided on the Unity 3D game engine. For addressing the limitations in the existing parametric modeling toolkit and emphasizing the advanced feature of VR, we propose the PM4VR, an integrated and simplified design package on the Unity 3D game engine, for facilitating the VR-driven conceptual architecture parametric modeling and designing experiences of architects. The main contributions of our work include:



(a) [Hawton et al. 2018] (b) [Coppens et al. 2019]

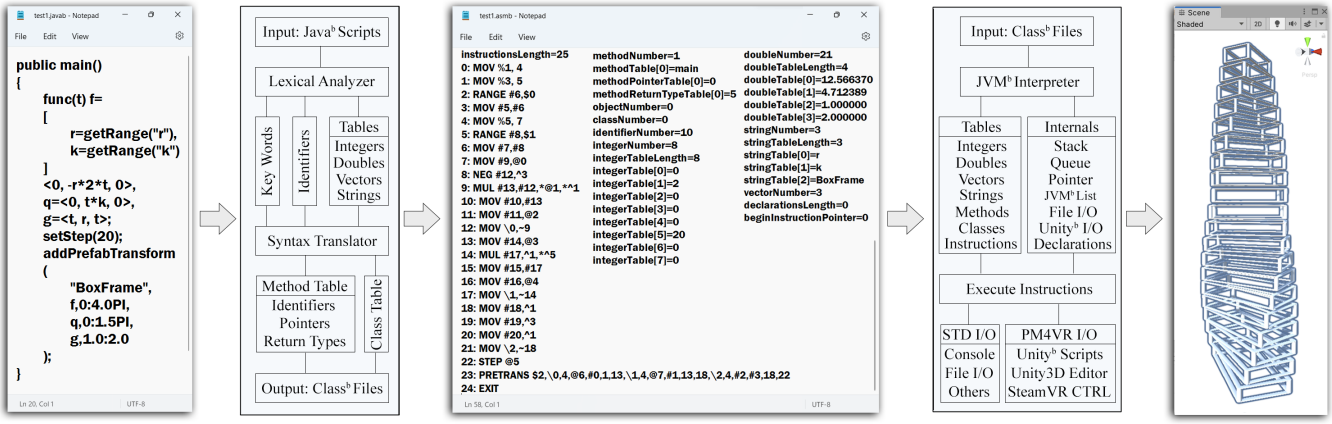
(c) [Castelo-Branco and Leitão 2022] (d) Archimatix Pro [2021]

Figure 2: Some related works that connect existing parametric modeling tools with immersive virtual environments.

- Designing a new Java-like and object-oriented programming language called Java$^\flat$ with simpler grammars that are specially proposed for efficient parametric modeling.
- Developing the Java$^\flat$ compiler using C language programming, developing a virtual machine for Unity 3D Game Engine called JVM$^\flat$, and a novel assembly language instructions set called ASM$^\flat$ which is executed on JVM$^\flat$.
- Developing the VR feature by connecting the JVM$^\flat$ with the SteamVR plugin and test our proposed interactive parametric modeling interface of PM4VR in virtual reality.

2 OVERVIEW

In this paper, we propose the PM4VR, a scriptable integrated design toolkit on Unity 3D game engine, for facilitating the VR-driven conceptual architecture parametric modeling and designing experiences. More specifically, we simplify the programming by proposing a novel programming language (called Java$^\flat$) with simple grammar but powerful parametric modeling capabilities. After the designer writes the Java$^\flat$ script for architecture designs, the Java$^\flat$ compiler compiles the scripts and generates a set of specially designed assembly language instructions (called ASM$^\flat$), we develop a Unity 3D Game Engine-based virtual machine (called JVM$^\flat$). By executing the ASM$^\flat$ instructions, the JVM$^\flat$ can exchange parametric modeling information between the Unity3D Editor and the Java$^\flat$ script so as to achieve the interactive parametric modeling feature on Unity3D in a virtual reality environment through SteamVR plug-in. Java$^\flat$ language is an object-oriented programming language with the programming style of Java. It avoids many complex syntax definations and also include some new features such as 3D vector operations, function operations, geometric transformation, etc. This language has a lot of flexibility that RhinoScript language does not have, also, it encapsulates many graphics modules with strong applicability, such as curve and surface modeling based on parametric equations and connection with Unity3D assets. Therefore, the Java$^\flat$ language will potential have a valuable impact among existing parametric modeling interfaces for conceptual architecture design in VR.



(a) Javab Script (b) Javab Compiler (c) ASMb Instructions in Classb Files (d) JVMb + PM4VR (e) Unity Output

Figure 3: Overview of our approach.

3 TECHNICAL APPROACH

3.1 Javab Compiler

Javab language is a 3D programming language designed by us. As shown in Figure3(a), the overall grammar of Javab is similar to Java language but is significantly simplified. For more details about Javab language programming, please refer to the Supplementary Material (Section 1). The implementation of the Javab compiler is using C language and is based on three main parts including the definition of language grammar structures, the design of a lexical analyzer, and the implementation of an object-oriented syntax translator based on a top-down recursive descent semantic analysis algorithm. Besides, there are techniques involved in symbol or identifier table management, runtime data storage organization, and information exchange with external resources such as the console, file system, and the Unity3D interface. Javab's lexical analyzer implements those functionalities including scanning the Javab script line by line from left to right, translating valid words into keywords or identifiers, creating constant data segment tables (including integers, doubles, strings, and vectors), etc. The Javab's syntax translator can identify the semantics of the grammatical components, obtain the attributes of identifiers, semantic checking, static binding of method pointers, calculate the relative addresses of instructions, bindings static variables, binding relative addresses of identifier tables, etc. After being compiled through the Javab compiler, the Javab script (*.javab) can be translated into the ASMb instructions (*.asmb) and written into binary class files (*.classb) called Classb. Javab compiler's main components are shown in Figure3(b).

3.2 ASMb Language

Assembly Language Flat (ASMb) is a high-level machine language that we have specially designed for the Javab Virtual Machine (JVMb). JVMb can interpret and execute ASMb instructions to implement the stand I/O, human-computer interaction in VR, and parametric modeling on Unity3D. As shown in Figure3(c), there are many similarities between ASMb language and standard assembly languages, e.g., both of them contain the data segment and code segment. But the ASMb instruction set is more advanced than the standard ASM instruction set because of ASMb instructions' flexibility. For example, ASMb can directly store integers, doubles, and strings in the data segment. But in the standard ASM language, only byte/word data can be stored. Moreover, the standard ASM language only supports fixed-length instructions, but ASMb supports variable-length instructions such as ADD, MUL, AND, SWITCH, etc. Therefore, the multiple instructions in standard ASM language can be completed by one instruction in ASMb, which improves the execution efficiency. For more details about ASMb language, please refer to the Supplementary Material (Section 2).

3.3 Javab Virtual Machine

Javab Virtual Machine (JVMb) is a software developed by us in C language. JVMbis similar to Java Virtual Machine (JVM). Typically, virtual machines can be used for separating the software from hardware. Advantages of virtual machines include high execution efficiency, cross-platform, easy for software development, etc. Given this observation, JVMb has a unique instruction set: ASMb(Assembly Language Flat), a powerful variable-length instruction set, which makes JVMb's execution efficiency very high. At the same time, the ASMb instruction set enables the user to generate 3D graphics on Unity3D regardless of the execution platform. The JVMb's execution process includes: loading the class file, allocating memory, loading the data segment and code segment, initializing the instruction pointer, and executing the instructions one by one until the instruction pointer points to the instruction of EXIT. In the end, exiting the whole virtual machine and finished running. The main components of the JVMbare shown in Figure3(d).

3.4 Unityb Script and PM4VR

The main function of the JVMb is to provide an efficient programming interface on Unity Editor (as shown in Figure3(e)), thus becoming a practical programming platform for parametric modeling in VR (which is called PM4VR in our work). Once users import our PM4VR interface package into Unity Editor, four C# script files will be imported including **UnityUtils.cs**, **MeshGeometry.cs**, **JavabCompiler.cs**, and **JavabScriptBehaviour.cs**. Among those C#

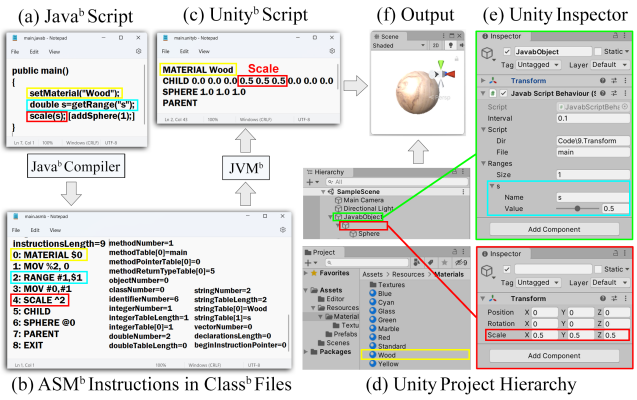


Figure 4: Example of PM4VR's execution process.

scripts, **JavabCompiler.cs** is used to connect JVM^b with Unity Editor through temporary files called Unity^b Scripts (*.unityb). More specifically, every time when user wants to reload updated Java^b scripts or wants to tune the parametric design through the range controls in Unity Editor or SteamVR, Unity^b scripts that contain the parameter settings will be written automatically. Then, JVM^b can read those parameters from Unity^b Script and output 3D geometrical calculations results into Unity^b Script again. In the end, **JavabScriptBehavior.cs** will load those results from Unity^b Script into Unity Editor or VR environments in real-time for users' interactions on parametric molding in VR. Figure4 shows an example of using PM4VR to add a wooden sphere with a radius specified by a range control named "s" whose current value is set to 0.5. For more detailed explanation for this running example, please refer to the Supplementary Material (Section 3).

3.5 Java^b Parametric Modeling

In our PM4VR package, **MeshGeometry.cs** implements some important parametric modeling algorithms discussed in this section.

Curve. According to the curve's parametric equation, any 3D curve can be defined as the trajectory of a moving 3D point or vector with is changing along with the time parameter t such that $\mathbf{f}(t) = \langle x(t), y(t), z(t) \rangle$, where $\mathbf{f}(t) \in \mathbb{R}^3, t \in [t_0, t_1], t_0 < t_1 \in \mathbb{R}$. Given this representation, any 3D curve can be mapped from a 1D parametric space into the 3D space. However, only a 3D curve can not present a meshed surface in parametric modeling. Therefore, in our proposed PM4VR, one additional 1D scalar parametric equation $g(t)$ where $t \in [t'_0, t'_1]$ and $t'_0 < t'_1 \in \mathbb{R}$ is considered in our approach to represent the radius of the moving points at time t. By combining point position of $\mathbf{f}(t)$ and point radius $g(t)$, user can construct the pipe-like 3D shape using the function call **addPipe** $(\mathbf{f}(t), t_0 : t_1, \mathbf{g}(t), t'_0 : t'_1)$ in the Java^b script. Figure 5 shows an example of pipe geometry generated in Unity with Java^b script, for more details of the equation described in the script, please refer to Supplementary Material (Section 3).

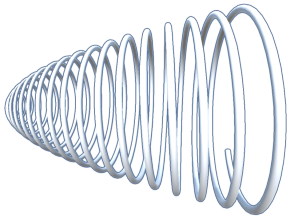


Figure 5: Curve

Surface. Surface's parametric equation can be defined through two parameters u and v as $\mathbf{f}(u, v) = \langle x(u, v), y(u, v), z(u, v) \rangle$ where $u \in [u_0, u_1]$, $v \in [v_0, v_1]$, $u_0 < u_1 \in \mathbb{R}$ and $v_0 < v_1 \in \mathbb{R}$. Through this representation, any 3D surface can be mapped from a 2D parametric space. In the Java^b script, user can construct the 3D surface using function call **addSurface** $(\mathbf{f}(u, v), u_0 : u_1, v_0 : v_1)$. Figure 6 shows an example of Klein surface generated with Java^b script, for more details of the equation described in the script, please refer to the experiment result of Building 1 in Figure 13, or, refer to Supplementary Material (Section 3). Indeed, Klein surface is very famous among parametric modeling designers and it has been widely used in conceptual architecture designs. Besides, using the same Java^b function call, 1D scalar field equation $h(u, v) \in \mathbb{R}$ is also supported by our interface though an autoconversion from 1D scalar function $h(u, v)$ into 3D vector $\mathbf{f}(u, v) =< u, h(u, v), v >$. This function type can be employed to represent the terrain's heightmap.

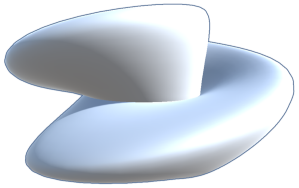


Figure 6: Surface

Prefab Surface. Given arbitrary parametric equation of surface $\mathbf{f}(u, v) = \langle x(u, v), y(u, v), z(u, v) \rangle$ and a pre-defined 3D prefab in Unity3D Editor or using Java^b function of **addPrefab** ("prefab name", $\langle$ scale $\rangle$), we provide another useful Java^b function called **addPrefabSurface** ("prefab name", $\langle$ scale $\rangle$, $\mathbf{f}_(u, v), u_0 : u_1, v_0 : v_1$). This function call automatically adds and places prefab onto every vertex in the surface mesh and rotates such prefab by the transformation such that the prefab's local space coordinates $\langle \hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}} \rangle$ are aligned with the surface's tangent - normal - bitangent space $\langle \hat{\mathbf{t}}, \hat{\mathbf{n}}, \hat{\mathbf{b}} \rangle$ coordinates such that $\langle \hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}} \rangle \rightarrow \langle \hat{\mathbf{t}}, \hat{\mathbf{n}}, \hat{\mathbf{b}} \rangle$. Figure 7 shows an example of prefab surface geometry generated in Unity with Java^b script, in this case, the prefab is a manually created 3D model called "AxisIcon". For more details of the equation described in the script, please refer to Supplementary Material (Section 3).

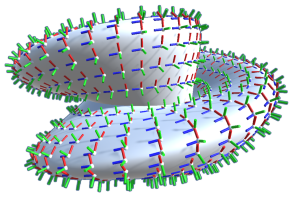


Figure 7: Prefab Surface

Wire Surface. Wireframes are commonly used in conceptual architectural design. Typically, wire surface is an important skill for architecture parametric modeling. To provide such technical support in PM4VR, we design a Java^b function called **addWireSurface** $(\mathbf{f}_(u, v), u_0 : u_1, v_0 : v_1,$ radius). After specifying the radius of the wires and the function $\mathbf{f}_(u, v)$, the wire surface is automatically generated through this function call by placing line pipes whose endpoints align with every two adjacent vertices in the surface mesh. Figure 8 shows an example of wire surface geometry generated in Unity with Java^b script, for more details of the equation described in the script, please refer to Supplementary Material (Section 3).

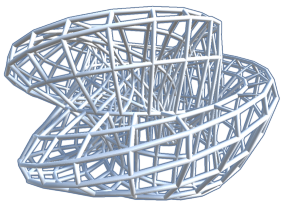


Figure 8: Wire Surface

Crust Surface. In most of cases, surface function $h(u,v)$ has no volume. However, the crust does. By extruding any surface along its normal direction at a specific distance, crust geometry can be generated from surface geometry. Therefore, in order to convert the arbitrary surface into crust, a Java♭ function called **addCrust**($\mathbf{f}_(u,v)$, $u_0 : u_1$, $v_0 : v_1$, $g_(u,v)$, $u_0' : u_1'$, $v_0' : v_1'$, $q_(u,v)$, $u_0'' : u_1''$, $v_0'' : v_1''$) is developed by us. Mathematically, we construct two surfaces including $\mathbf{f}^+(u,v)$ as the outer side surface and $\mathbf{f}^-(u,v)$ as the inner side surface, then a crust geometry is generated by $\mathbf{f}^+(u,v)$ and $\mathbf{f}^-(u,v)$. Given $g_(u,v)$ and $q_(u,v)$ specifying the thickness of the outer side and the inner side, mathematically, we have $\mathbf{f}^{+/-}(u,v)$ defined through the following equations:

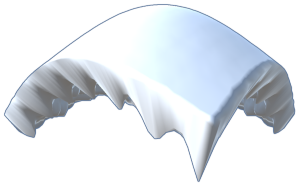


Figure 9: Crust Surface

$$\begin{cases} \mathbf{f}^+(u,v) = \mathbf{f}(u,v) + \mathbf{n}(u,v)g(u,v) \\ \mathbf{f}^-(u,v) = \mathbf{f}(u,v) - \mathbf{n}(u,v)q(u,v) \end{cases} \tag{1}$$

where the normal of surface $\mathbf{n}(u,v)$ is calculated as:

$$\mathbf{n}(u,v) = \frac{\mathbf{f}_u(u,v) \times \mathbf{f}_u(u,v)}{\left|\mathbf{f}_u(u,v) \times \mathbf{f}_u(u,v)\right|} \tag{2}$$

Figure 8 shows an example of wire surface geometry generated in Unity with Java♭ script, for more details of the equation described in the script, please refer to Supplementary Material (Section 3).

Coons Surface. We provide another functionality to generate surfaces from four borders which are defined with four curve equations which are $\mathbf{p}_{U0}(u)$, $\mathbf{p}_{U1}(u)$, $\mathbf{p}_{0V}(v)$, and $\mathbf{p}_{0V}(v)$. According to the definition of Coons Surface which is calculated by the linear interpolation between two opposite borders subtracted by the bilinear interpolation between four borders. Through function **addCoonsSurface**($\mathbf{p}_{U0}(u)$, $u_0 : u_1$, $\mathbf{p}_{U1}(u)$, $u_0' : u_1'$, $\mathbf{p}_{0V}(v)$, $v_0 : v_1$, $\mathbf{p}_{1V}(v)$, $v_0' : v_1'$) in the Java♭ developed by us, Coons surface can be automatically constructed through the following equation:

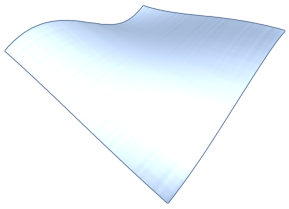


Figure 10: Coons Surface

$$\begin{aligned} \mathbf{f}(u,v) = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{p}_{0V}(v) \\ \mathbf{p}_{1V}(v) \end{bmatrix} + \begin{bmatrix} \mathbf{p}_{U0}(u) & \mathbf{p}_{U1}(u) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix} \\ - \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{p}_{00} & \mathbf{p}_{01} \\ \mathbf{p}_{10} & \mathbf{p}_{11} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix} \end{aligned} \tag{3}$$

where $\mathbf{p}_{00}$, $\mathbf{p}_{01}$, $\mathbf{p}_{10}$, $\mathbf{p}_{11}$ are calculated through the intersections between every two pairs of adjacent border curves. Figure 10 shows an example of a Coons surface geometry generated in Unity with Java♭ script, for more details of the equation described in the script, please refer to Supplementary Material (Section 3). Due to the simplicity of such a definition, Coons surface is widely used in architectural design and CAD designs. Especially, this functionality is able to help designers to build 3D surfaces directly from the curve equations that define the border shapes of arbitrary surface patches.

Isosurface. In geometric modeling, any isosurface can be defined from a 3D scalar field $f(x,y,z) \in \mathbb{R}$ through an isovalue C, where $x \in [x_0, x_1]$, $y \in [y_0, y_1]$, $z \in [z_0, z_1]$, $x_0 < x_1 \in \mathbb{R}$ and $y_0 < y_1 \in \mathbb{R}$, and $z_0, z_1 \in \mathbb{R}$. Mathematically, the isosurface is constructed from those points satisfying the equation: $f(x,y,z) = C$. In Java♭ script, user can define any isosurface through function call **addIsoSurface** ($f(x,y,z) == C$, $x_0 : x_1$, $y_0 : y_1$, $z_0 : z_1$). In our proposed interface, we automatically construct the surface mesh through the marching cube algorithm. Figure 11 shows an example of isosurface geometry generated in Unity with Java♭ script, In this case, a hyperboloid isosurface (a quadric surface) is constructed as a Unity Prefab at the beginning. After then, it is loaded nine times along a circle. For more details of the equation described in the script, please refer to Supplementary Material (Section 3).

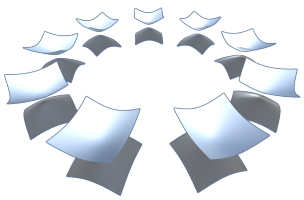


Figure 11: Isosurface

Prefab Transform. With our proposed PM4VR interface, the user can load a Unity prefab repeatedly through a series of transformations defined through three different vector parametric equations which are translation equation $\mathbf{f}(t)$, rotation equation $\mathbf{q}(t)$, and scale equation $\mathbf{g}(t)$. After calling the Java♭ function which is **addPrefabTransform** ("prefab name", $\mathbf{f}(t)$, $t_0 : t_1$, $\mathbf{q}(t)$, $t_0' : t_1'$, $\mathbf{g}(t)$, $t_0'' : t_1''$), prefab is loaded according to the position specified in $\mathbf{f}(t)$, the rotation Euler angles specified in $\mathbf{q}(t)$ and the scale specified in $\mathbf{g}(t)$. This function is very helpful for parametric architecture designs, especially for those tilted building structures design. Figure 12 shows an example of the transforming prefab geometry generated in Unity with Java♭ script, in this case, the prefab is a manually created 3D model called "BoxFrame". According to this design, as the "BoxFrame" is moving downwards, it zooms out and at the same time and rotates along the y-axis. For more details of the equation described in the script, please refer to Supplementary Material (Section 3). A similar design is also shown in the experimental result of Building 11 in Figure 14.

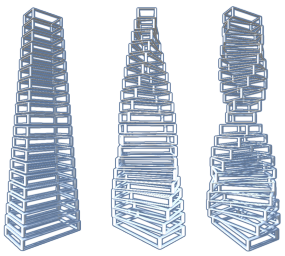


Figure 12: Tilt Prefab

4 EXPERIMENT RESULTS

We conducted a series of experiments to validate the functionalities of our proposed PM4VR interface on Java♭ parametric modeling for conceptual architectural design. Given fifteen different parametric equations as design plans, we program those conceptual designs using Java♭ scripts and execute those scripts in Unity3D Editor. Experiment results are shown in Figure 13-15, the virtual urban environment is automatically loaded through OpenStreetMap (OSM) API which is matching with the real world located nearby the Zhujiang New City Tower, Tian He Qu, Guang Zhou Shi, China. According to the original map loaded from OSM API, there is plenty of grass or empty space for placing those conceptual architecture designs. Fifteen architectural designs are labeled from Building 1

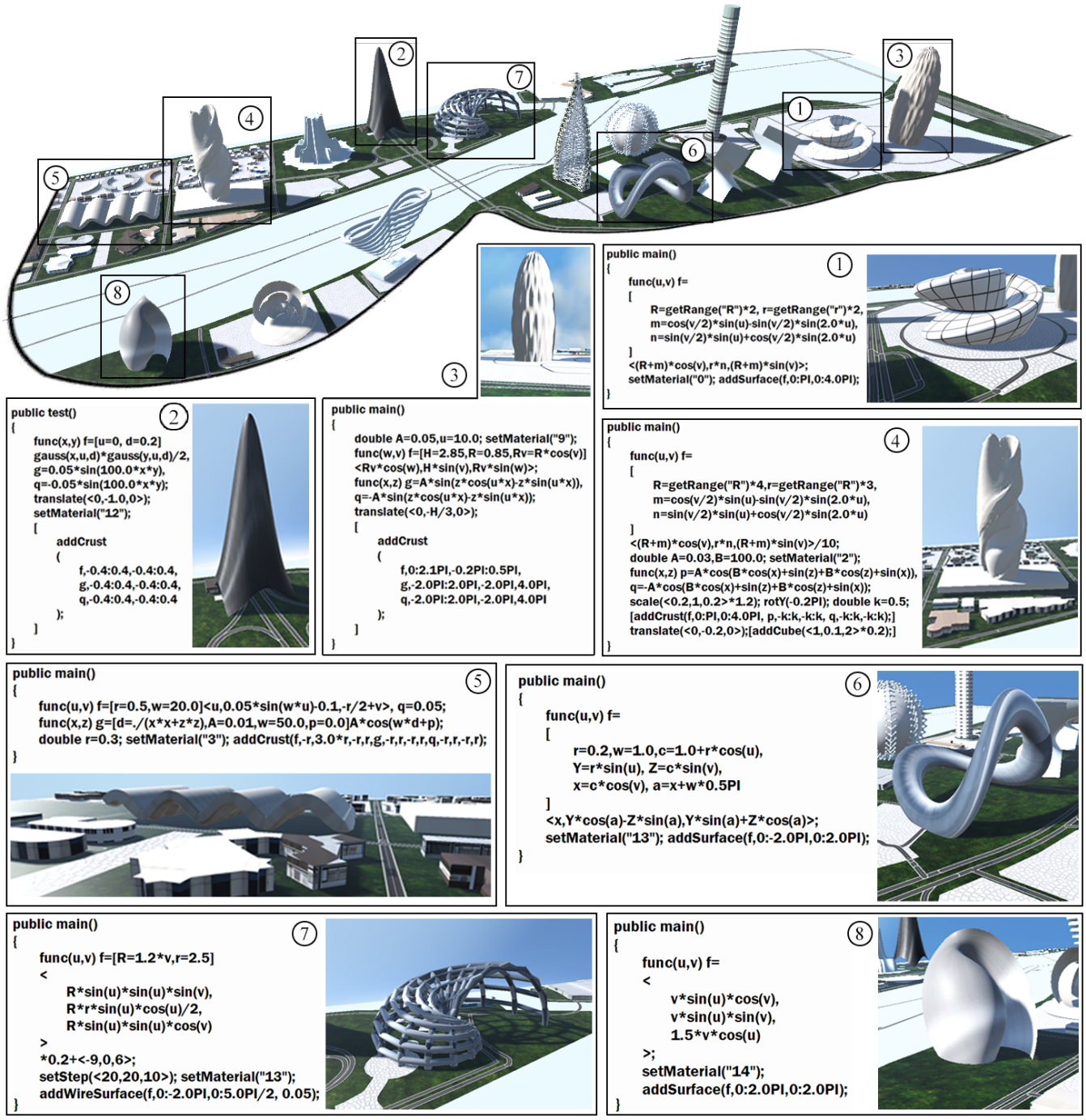


Figure 13: Experiment Results (Part 1). This figure shows 1st to 8th architecture designs using PM4VR Java♭ programming.

to Building 15 whose labels can be found correspondingly in those subfigures. Each subfigure shows the Java♭ script and its corresponding conceptual architecture design result rendered with a closer view in the virtual environment.

Building 1 is generated using a standard Klein surface parametric equation with a white grid texture. Two parameters for Building 1 include *R* and *r* used to adjust the radius and height of this building respectively. *R* and *r* are acquired through two range controls named "R" and "r" respectively. The white grid texture is defined in a material whose filename is "0.mat". Building 2 is generated using a 2D Gaussian equation extruded with sine equations. Its appearance uses a dark material texture which is defined in a material whose filename is "12.mat". Building 3 is generated using a 3D oval sphere equation extruded by a sine wave equation as a "bumpy" oval crust. Its appearance uses a white marble texture which is defined in a material whose filename is "9.mat". Building 4 is generated using

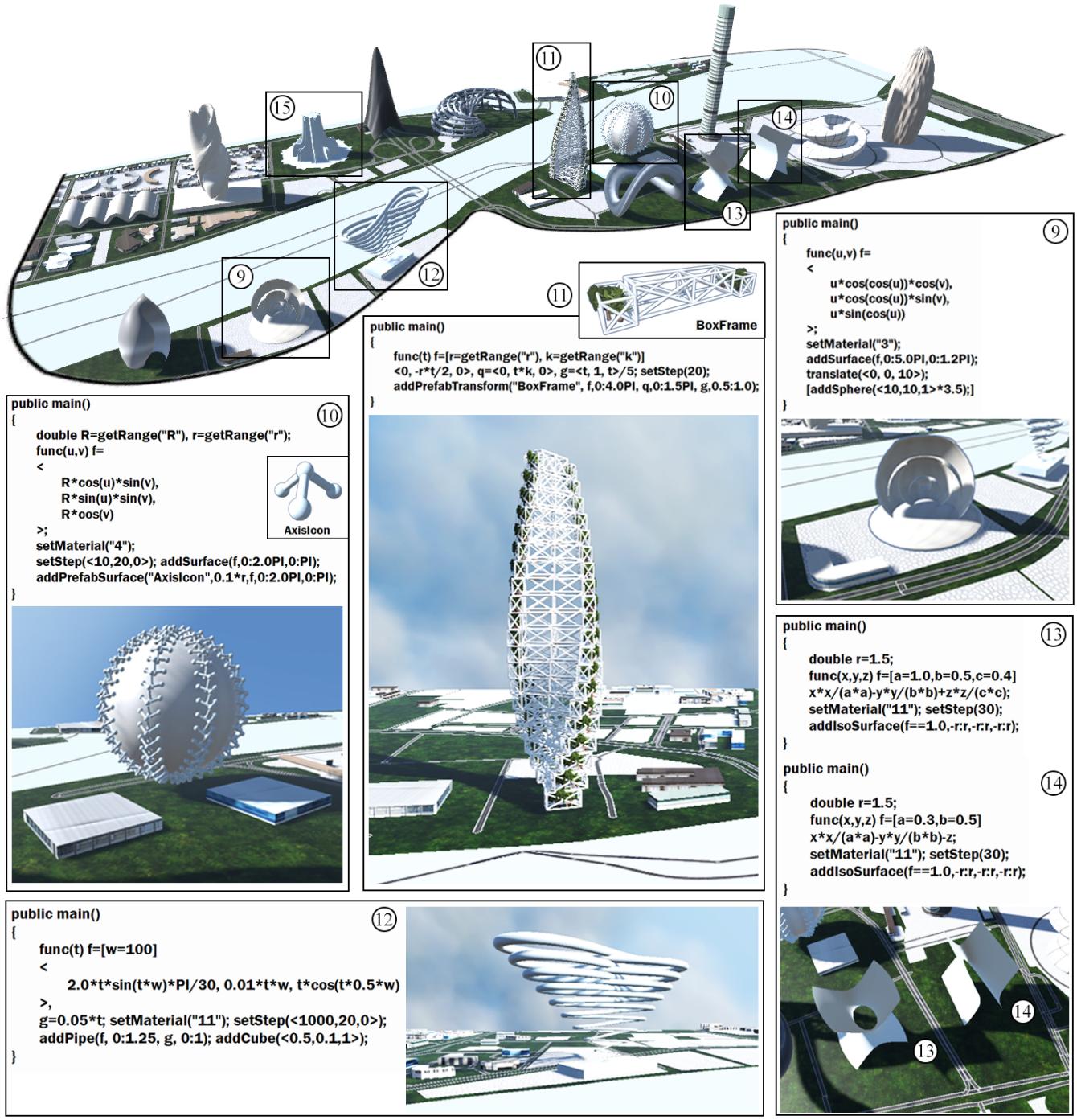


Figure 14: Experiment Results (Part 2). This figure shows 9^{th} to 14^{th} architecture designs using PM4VR Java$^{\flat}$ programming.

an extruded spiral shape crust that looks like a deformed version of the "double helix". At the bottom of the design, a flat box is added as a base of this building. Its appearance uses a white stone texture which is defined in a material whose filename is "2.mat". Building 5 is generated using a crust extruded from a 2D sine equation that looks like a waved roof. Its appearance uses a gray texture which is defined in a material whose filename is "3.mat". Building 6 is generated using a Möbius ring surface equation. Its appearance uses a shining gray metallic texture which is defined in a material whose filename is "13.mat". Building 7 is generated as a wired surface using the spiral shell equation. Its appearance uses the same material as Building 6. Building 8 is generated using a Scotch Bonnet shell surface equation. Its appearance uses a gradient gray texture which is defined in a material whose filename is "14.mat". Building 9 is generated using a nested sphere surface equation along with a flat sphere as a base. Its appearance uses same material as Building 5.

```
double F(double X,double Z,double H,double G)
{
    double a=1.3,b=1.4,c=0.1;
    double y=a*X*X*Z-b*X*X*X*X-c*Z*Z*Z;
    if(y<G)y=G; else if(y>H)y=H; return y;
}

void CreatePrefab()
{
    double r=13.5,h=24.5,g=-10.0;
    func(x,z) f=F(x,z,h,g),
    g=0.0005*sin(100.0*x*z),
    q=-0.0005*sin(100.0*x*z);
    setStep(200); setMaterial("11");
    newPrefab("Group1")
    {
        translate(<0,0.0,r>);
        [
            addSurface(f,-r:r,-r:r);
        ]
    }
}

public main()
{
    CreatePrefab(); int length=10;
    double w=2.0PI/(length-1);
    translate(<0,-1.0,-r>);
    [
        for(int i=0;i<length;i++)
        {
            rotate(<0,1,0>,-w*i);
            [
                addPrefab("Group1", 1);
            ]
        }
    ]
}
```

15

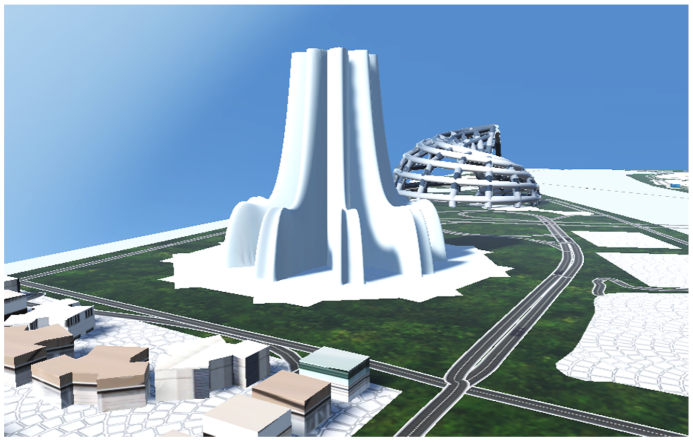


Figure 15: Experiment Results (Part 3). This figure shows the 15^{th} architecture design using PM4VR Java$^{\flat}$ programming.

Building 10 is generated by placing the prefabs called "AxisIcon" along a sphere surface equation. According to the step settings for $u = 10$ and $v = 20$, prefabs are distributed denser along latitude than longitude, and two parameters R and r are used for adjusting the sphere radius and the Axis Icon size respectively. Building 11 is generated using a tilt structure using the "BoxFrame" prefab. As tiling the prefab up, it is rotating along the y-axis while zooming out. Two parameters of r and k are used for adjusting the building's height and tilting degree respectively. Building 12 is generated using a pipe curve structure defined from a progressing figure "8" track. Its appearance uses a bright white texture which is defined in a material whose filename is "11.mat". Building 13 and Building 14 are generated from the isosurface of hyperboloid equations. Their appearance uses the same material as Building 12. Building 15 is generated using an "F" function that constructs a surface shape that looks like a "waterfall" and saves it as a prefab called "Group1" though Java$^{\flat}$ function call **newPrefab**("prefab name"). In main function, "Group1" is loaded ten times while rotating along y-axis.

5 USER STUDY

We conducted a preliminary user study to ask a user to test our PM4VR interface by tuning the architectural parametric design within a virtual environment. Figure 16 shows the photo capture during the study where the left part shows the user's view captured in VR and the right part shows the user who is interacting with the range control using a VR controller. As shown in the result, we develop this VR range control identical to the range control in Unity Editor. Real-time update on architecture design is rendered in Oculus Quest 2 VR headset while the user tunes parametric value via VR range control, this results in an immersive design experience.

6 CONCLUSION

In this paper, we propose a novel easy-to-write parametric modeling language Java$^{\flat}$ along with a VR interactive interface called PM4VR that connects Java$^{\flat}$ parametric modeling with virtual reality. By using PM4VR which implements some parametric modeling algorithms commonly used in architecture concept design, the user can write simple Java$^{\flat}$ script to create and update the conceptual architecture design in an immersive virtual environment through

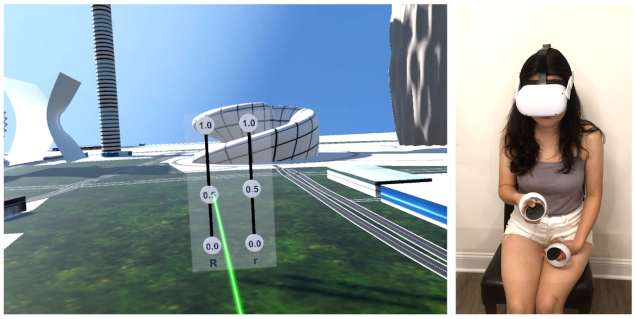


Figure 16: User is tuning architecture parameters via PM4VR.

an Oculus Quest 2 VR headset and VR controllers. In future work, we will conduct larger-scale user studies to validate our proposed PM4VR platform from more aspects of analysis including the Java$^{\flat}$ coding efficiency comparison test, user interaction test, and large-scale architecture conceptual designs application test, etc.

REFERENCES

Maziar Asefi and Mahnaz Bahremandi-Tolou. 2019. Design challenges of reciprocal frame structures in architecture. *Journal of building Engineering* 26 (2019), 100867.

Renata Castelo-Branco et al. 2019. Immersive Algorithmic Design-Live Coding in Virtual Reality. (2019).

Renata Castelo-Branco and António Leitão. 2022. Algorithmic Design in Virtual Reality. *Architecture* 2, 1 (2022), 31–52.

Adrien Coppens, Tom Mens, and Mohamed-Anis Gallas. 2019. Parametric modelling within immersive environments: building a bridge between existing tools and virtual reality headsets. *arXiv preprint arXiv:1906.05532* (2019).

Theresa Fink and Reinhard Koenig. 2019. Integrated Parametric Urban Design in Grasshopper/Rhinoceros 3D-Demonstrated on a Master Plan in Vienna. (2019).

Dominic Hawton, Ben Cooper-Wooley, Jorke Odolphi, Ben Doherty, Alessandra Fabbri, Nicole Gardner, and M Hank Haeusler. 2018. Shared immersive environments for parametric model manipulation-evaluating a workflow for parametric model manipulation from within immersive virtual environments. (2018).

Hanie Omid and Mahmood Golabchi. 2020. Survey of parametric optimization plugins in Rhinoceros used in contemporary architectural design. In *Proceedings of the Fourth International Conference on Modern Research in Civil Engineering, Architecture, Urban Management and Environment, Karaj, Iran*, Vol. 17.

Roaring Tide Productions. 2021. Unity Asset Store: ArchimatixPro.

Monika Dyavenahalli Shivegowda, Pawinee Boonyasopon, Sanjay Mavinkere Rangappa, and Suchart Siengchin. 2022. A review on computer-aided design and manufacturing processes in design and architecture. *Archives of Computational Methods in Engineering* (2022), 1–8.

Milena Stavric and Ognen Marina. 2011. Parametric modeling for advanced architecture. *International journal of applied mathematics and informatics* 5, 1 (2011), 9–16.