WORCESTER POLYTECHNIC INSTITUTE

# “Meng” - Live Coding in Virtual Reality

by

Chunzhen Jiang
Kai Yan
Jian Liu

A thesis submitted in partial fulfillment for the
degree of Master of Science
in
Interactive Media and Game Development

May 2021

APPROVED:

Professor Charles Davis Roberts, Major Thesis Advisor

Professor Mark L. Claypool, Major Thesis Reader

# ABSTRACT

We discuss live coding inside of a virtual reality environment using specialized virtual reality equipment. While prior research on live coding has primarily focused on using keyboard and mouse input, we argue this is not appropriate for virtual reality environments, where users cannot see a physical keyboard. We have created a new environment, named “Meng”, that focuses on embodied gestures, voice recognition, and in-world graphic user interfaces to enable users to program in a more embodied style while immersed in the virtual world they are creating. We conducted a user study to evaluate our environment; however, due to COVID-19 restrictions participants used an alternative version of our software meant for use with typical desktop computers. Participants indicated that Meng is a live coding system that has immersive visual effects, and that most readily adapted to using the new input mechanisms we placed in the environment. We released Meng—and two associated plugins for voice input and rendering ray marchers—as open-source plugins for Unreal Engine 4.

# 1 INTRODUCTION

Our project combines two rapidly emerging fields: live coding and virtual reality. As its name suggests, in live coding programmers/performers allow other people to watch their programming process in real-time during live performance. Most live coding performers combine music, images, and code on screen to create an interactive experience. While many live coders perform for audiences, live coding is also a novel method for creatively exploring software development; the near-instant feedback it provides to programmers encourages play and experimentation in a way that differs from traditional software development[1].

In virtual reality (VR) worlds[2], people see and experience completely different scenes from the physical world. We believe that live coding while immersed in VR provides performers an exciting opportunity to create new worlds and new types of live coding experiences. We chose the Oculus Quest[3], a recently introduced VR device, as our research focus for live coding. Because the Quest is a wireless VR device its mobility is better than other equipment, enabling users to experience virtual worlds more freely and more immersively.

Most VR devices, unlike typical computers, do not make heavy use of a physical keyboard for typing. But most prior research on programming in VR environments focused on creating virtual keyboards, and letting performers use input controllers to click these graphical widgets in VR environments (see detailed information in Section 2). However, with these affordances, programmers cannot input commands as quickly as they would in the real world; in our experience, the input speed is very slow when comparing virtual keyboards to physical ones. In live coding shows, there is an unwritten rule that performers cannot enter instructions too slowly, otherwise the resulting music and graphics become repetitive and boring as development takes too long; therefore we argue that virtual keyboards are an inappropriate input mechanism for live coding while immersed in VR. To remedy this, Meng combines GUI panels, voice commands, and gesture recognition as input functions in our live coding system. In this way, users will not need a traditional keyboard for programming while using VR equipment, and will be able to program in a more embodied fashion.

To make these novel input mechanisms work in Meng, we abstract control commands into a programming language that can be immediately evaluated. We also built a ray marching[4] plugin to render complex geometries in the virtual world and simulate flocking behaviors for the virtual agents while enriching their visual expression. Thus, users are able to build their world by creating different agents with this embodied programming language while navigating the 3D world through first person perspective which is—to us—like a dream. And that is why our project is called “Meng”, the Mandarin word for “dream”.

Meng provides users with many interactive affordances for creating virtual worlds:

- Creating agents with voice commands, which are close to natural language. For example, users can simply say “New red cube” to create a red cube in the virtual world.

- Selecting agents with VR controllers and then using GUI or voice commands to change their properties.
- Querying agents with specific tags by using speech. For instance, by simply saying “Query ray marching”, all the agents based on ray marching rendering will be selected.
- Drawing arbitrary paths for agents to follow and scaling the size of paths by stretch gesture.

Figure 1 shows an example scene created in our system, featuring flocks of agents, ray marched geometries, and volumetric fog.

**Figure 1**: A demo scene in Meng

Due to Covid-19, we ported our project from a VR-specific version to one that runs on regular desktop computers, so that participants could experience it remotely as part of a user study. This study focused on helping us improve Meng by collecting and analyzing users’ feedback about experiencing a new live coding environment with such different input mechanisms. We also compared Meng with a traditional web-based live coding environment using the Cognitive Dimensions of Notation framework[5].

The outline of this paper is as follows: In the next section we discuss the background of some VR projects that can allow players to program and a traditional live coding website. Section 3 is a detailed introduction and the methodology of creating Meng. We describe our experimental evaluation result in Section 4 before concluding the paper and discussing future work.

# 2 BACKGROUND AND MOTIVATION

The rapid maturation of VR technologies has led to a corresponding interest in researching methods for programming while immersed in virtual reality. Alive[6] takes the advantage of AlloSphere[7]—an immersive, spherical, virtual reality environment at the University of California, Santa Barbara—to explore the combination of multi user collaboration and live coding in a large-scale virtual reality environment. It provides users with a sense of immersion by applying spatialized audio and projecting across a three-story stereoscopic display while users develop the sonic and visual behaviors of agents in a virtual world.

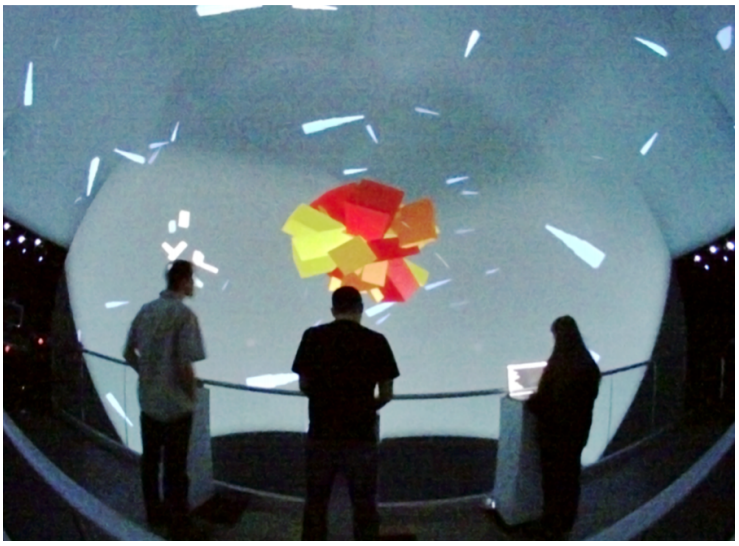

**Figure 2:** Fish-eye photograph of three performers standing on the bridge of the AlloSphere and live coding the world that surrounds them

Inception[8] is another system exploring live coding with VR devices. It is a system for digital artists to create artwork in a virtual reality environment, which takes both a standard QWERTY keyboard and VR controllers as input devices, providing a live coding text editor in the virtual reality interface in addition to a novel feature for viewing design alternatives. However, in this system, VR controllers are only used to manipulate rendered 3D objects, not to input source code.

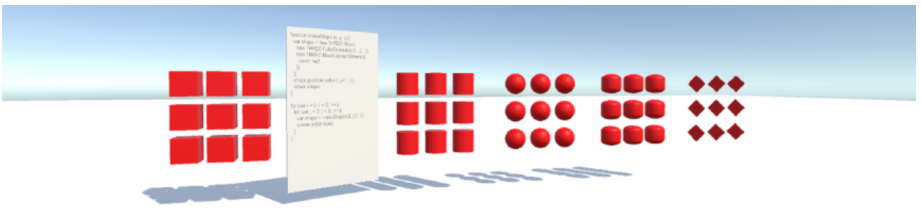

**Figure 3**: Inception creates an immersive creative-coding environment for digital artists

In regards to VR support for programming tasks, VR-FTC[9] contributes an immersive VR approach for visualizing program structures. Users can interactively visualize, navigate and

communicate program code information in a VR environment by using controllers and a virtual keyboard. Notable features it contains include both searching and tagging of code.

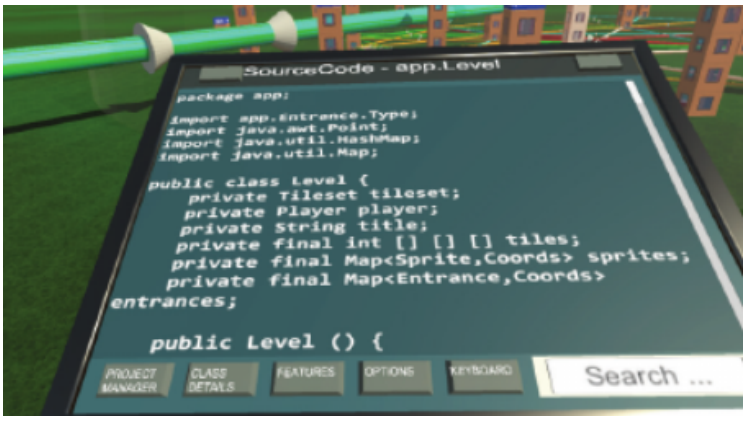


**Figure 4**: The code editor window in VR-FTC

**Figure 5**: A virtual keyboard supports text input for searching, filtering, and tagging in VR-FTC

In regards to game-like systems exploring VR programming, Cubely[10] presents puzzles in a virtual world and uses its game-like appeal to foster experimentation and creativity. By leveraging the embodied experience provided by VR devices, the 3D-VPL[11] has made its contribution in the field of education. It supports block-based programming systems that can be edited in the VR environment, which also provides an overview and survey of programming concepts visualized in a virtual world.

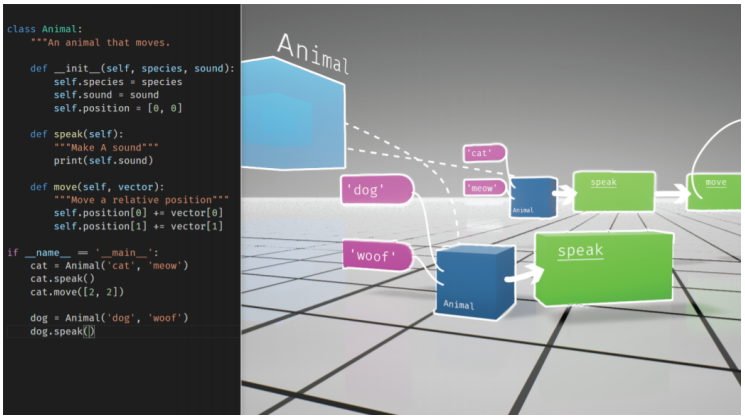


**Figure 6**: The code editor window in 3D-VPL

As to non-VR live coding software, Livecodelab[12] is a browser-based platform for live coding that supports both audio and graphics programming (as shown in Figure 7). The Improcess project[13] aims to create new tools for performing improvised electronic graphics in a live setting; its authors provide an in-depth discussion and evaluation of the abstraction of structure for developing a new language for live coding.


**Figure 7**: Livecodelab running inside of Google Chrome.

We compare the advantages and disadvantages of using a physical keyboard, virtual keyboard, or voice and gestures as the input method in the VR environment in Table 1.

| Input Methods | Advantages | Disadvantages |
|---|---|---|
| Physical keyboard | It is the most familiar and generic interface.<br><br>The input speed when using a physical keyboard is fast. | Keyboards would either have to be mounted on a harness for users to wear or it would only be usable when facing a single direction. |
| Virtual keyboard | It is also a familiar and generic interface, widely used in touch devices and VR systems.<br><br>Input speed with a virtual keyboard is relatively fast.<br><br>Easy to apply into a VR environment. | Virtual keyboard might occlude the virtual world when used in a VR environment.<br><br>Virtual typing could be difficult in VR. The input speed could be very low when typing on the keyboard with VR controllers. |
| Voice + Gestures + GUI | It is potentially more embodied in the VR environment.<br><br>It does not occlude the virtual world.<br><br>This interaction is very easy and interesting, so it might attract those people who do not have any background in programming. | It will be an unfamiliar interface for users accustomed to “traditional” programming.<br><br>Voice can be difficult to accurately recognize. So the voice system has high requirements in terms of both the players’ pronunciation and a quiet physical environment.<br><br>It can be difficult for the players to draw what they want, because drawing in a 3D world using VR controllers does not have the same haptic feedback as drawing on a plane. |

**Table 1**: Comparison of using different input methods in VR

Because of the disadvantages of using a physical keyboard and virtual keyboard in a VR environment, we believe that trying to bring traditional programming interfaces like keyboards into VR is the wrong approach. Instead, we hypothesize that programming in VR can take better advantage of the medium through the use of voice commands and embodied gestures. Thus, we explored combining GUI panels, gesture recognition, and voice commands as input methods instead of physical or virtual keyboards, providing what we feel is a more intuitive way to interact with the virtual live coding system.

# 3 METHODOLOGY

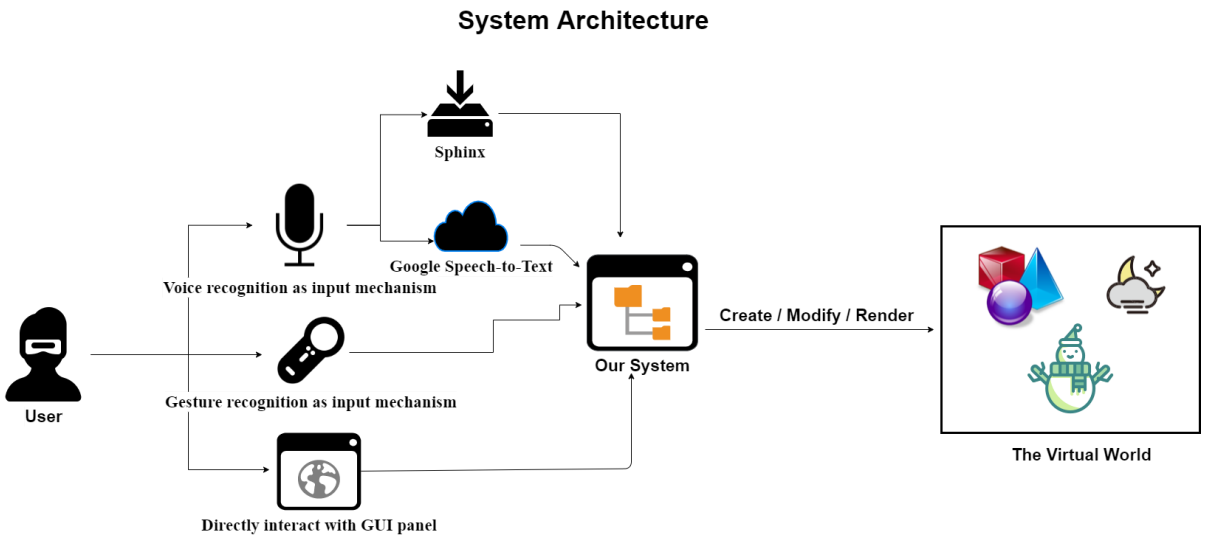


**Figure 8**: System Architecture

In this section, we discuss how we design and implement three input mechanisms, and our use of experimental rendering techniques like ray marching. Figure 8 shows a high level overview of the architecture of Meng.

## 3.1 Input mechanisms

Meng enables users to create virtual agents and control their movement in a virtual world. To create and program these agents, users employ three different input mechanisms: voice, gesture, and GUI. Agents can also be tagged to create groups; we can then query for all agents with a given tag to select a group of agents for modification.

### 3.1.1 Voice recognition as input mechanism

We combined Sphinx[14] and Google Speech-to-Text[15] to get the best of both systems: immediate responses to our grammar using Sphinx running on local machines, and the flexibility of natural language recognition when required (with some added latency) by using Google Speech-to-Text in the cloud.

With Meng's voice input system, users can create agents, adjust agents' parameters, tag agents and query agents. For example, after summoning the GUI panel by pressing the left trigger on a VR controller, users can speak to change the shape and size of the selected object or create new agents with some basic parameters such as color and shape using Sphinx. They can also tag agents or query agents using existing tags through Google Speech-to-Text.

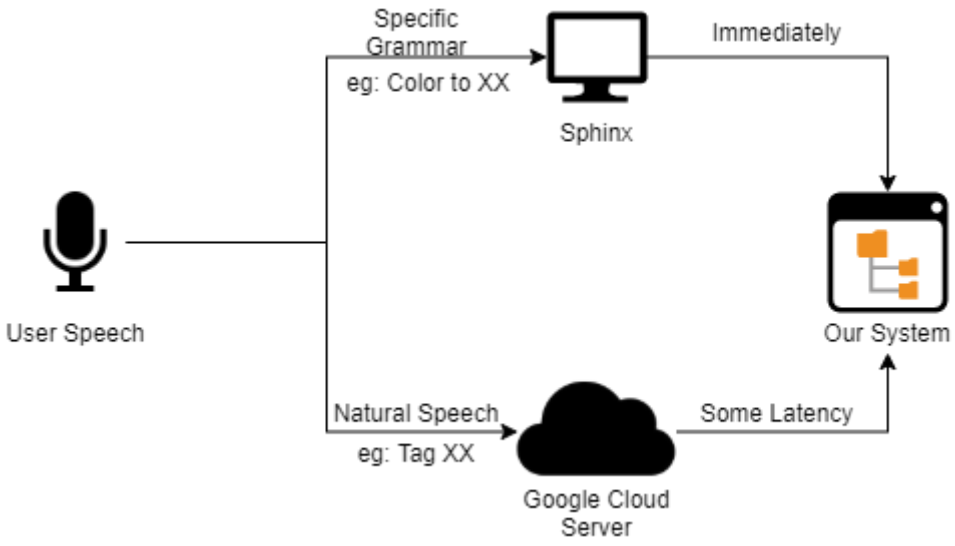


**Figure 9**: Combine Sphinx and Google Speech-to-Text

- **Sphinx:**

  To create a more intuitive method for users to input voice commands, we first brought the Sphinx-UE4[16] plugin into our project. This plugin enables the majority of the voice commands in Meng, and performs voice recognition on the local VR device or connected computer.

  Sphinx enables us to customize the grammar that we want it to recognize by providing a configuration file in the JSpeech Grammar Format (JSGF)[17]. JSGF is a platform-independent, vendor-independent textual representation of grammars for use in speech recognition. Figure 10 shows a simple JSGF grammar supporting statements such as “new red sphere”. Its ease of use and flexibility makes it easy for us to define the syntax that needs to be recognized. This in turn enables users of Meng to input both more complex and natural statements.

```
public <final_rule> =  new [ <colors> ] <meshBasedObjects>;

<colors> = ( red | blue | white | black | yellow | green );

<meshBasedObjects> = ( cube | sphere | pillar | elephant );
```

**Figure 10**: Grammar example that would recognize, as one example, “new red elephant”

  By switching grammar files according to the state of users, it is easy to react to different commands in different situations.

- **Google Speech-to-Text:**

  Although Sphinx’s grammar mode is very flexible, it is not as free as natural speech recognition; only words in the grammar file can be recognized. While the majority of

commands in Meng can be captured by a JSGF grammar, we also wanted to enable users to apply “tags” to virtual agents so that these tags could subsequently be used to query for agents possessing a particular tag; this is a common technique found in game engines (including Unreal Engine 4). To break out of the limit of word lists included in grammars and let users use any tags of their choosing, we tried to find a way to recognize natural speech. However, the accuracy of the natural speech mode of Sphinx—which runs purely on the user’s local machine—is too low. Our research found that Google Speech-to-Text offers very good natural language recognition service in comparison[18]. It uses a model trained on a huge speech data set, and can accurately convert speech into text.

To use Google Speech-to-Text in our project, we embedded a C++ REST SDK in Unreal Engine and used it to build a Google cloud client. Then, we created an asynchronous drop-in node for the Unreal Blueprints scripting language enabling easy configuration of the service.

When using web services like Google Speech-to-Text, smaller data sizes lead to faster responses. Google recommends capturing audio with a sampling rate of 16000 Hz and one channel. We used RtAudio—a cross-platform audio library included in UE4—to implement a voice capture function that allows us to set both sampling rate and the number of channels used for input. After recording speech, the recorded audio is converted to a Base64 string, so that we can pack the speech and associated configuration data into a JSON message and send it to Google Cloud through a HTTP request.

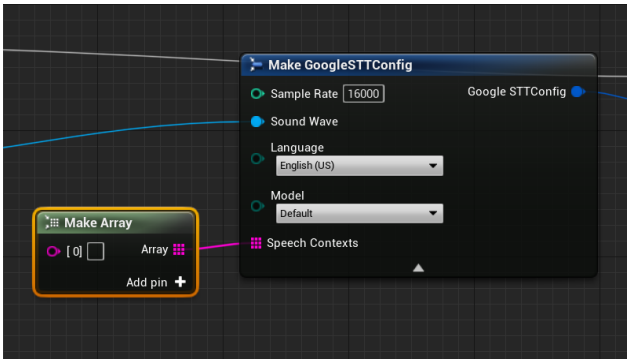


**Figure 11**: Google Speech-to-Text config

With Google Speech-to-Text, users can easily tag agents using whatever the word they want and query them later.

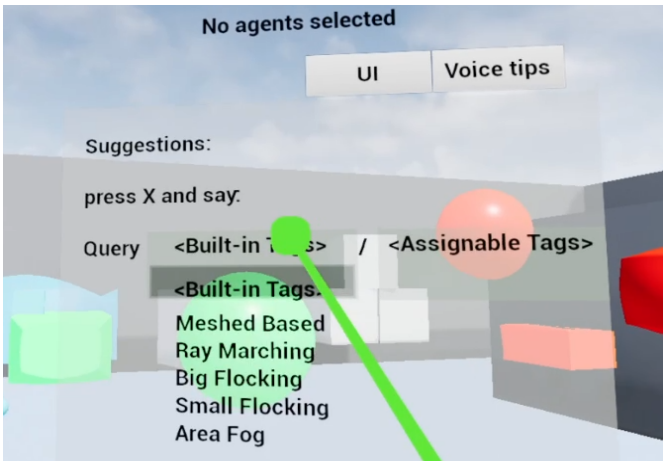


**Figure 12**: Query GUI

### 3.1.2 Gesture recognition as input mechanism

We integrate RunebergVRPlugin[19] in our project, which can record the movement of the VR controllers when the agent is grabbed by the left controller and the Y button on the right controller is pressed. By using this movement data, we enable users to draw arbitrary paths for selected objects to follow; the handheld VR controllers can also stretch these paths and change their scale.

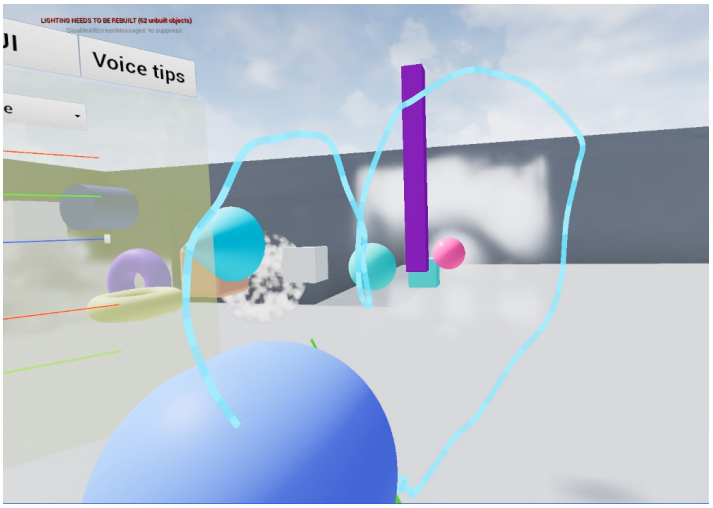


**Figure 13**: Draw a heart-shape path, and the selected cube will start to move and follow it

### 3.1.3 Directly interact with GUI panel

Two concerns prompted us to create Meng’s GUI system. First, it is hard to use voice or gesture to precisely adjust some agents’ parameters. For example, to precisely change a color to a desired hue, you might need to experiment with changing the value of the color’s red channel many times. This will be difficult if every command is executed via voice command; dragging a slider will often be much quicker. Second, memorizing all the commands in our project is difficult, especially for those using our project for the first time. The GUI system

also should act as a complement to the voice and gesture input systems, and provide hints to users about commands they can explore. So we created a GUI system which can allow users to directly interact with agent parameters.

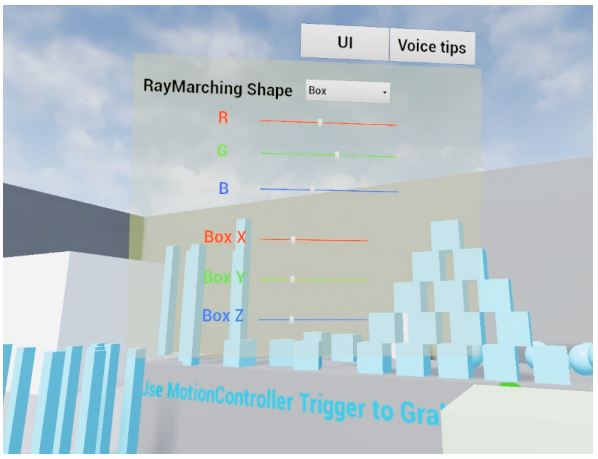


**Figure 14**: Using the GUI to change the color and shape of the selected object

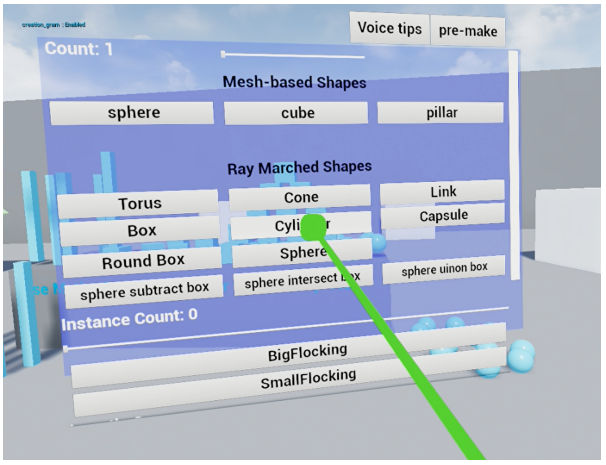


**Figure 15**: Using the UI to create a new agent

We designed the GUI layout based on the characteristics of the virtual reality world and our project. In general, the GUI of traditional video games is designed in screen space. However, we cannot use the same approach in the virtual reality world because the VR equipment is worn on the head of the player which means the distance between the GUI and the player’s eyes is so close that all the player will see is the GUI. To avoid this, we decided to attach the GUI to the left controller and adjusted its distance, position and size to ensure that players can interact with it comfortably.

As we said before, the goal for our GUI is for it to supplement the voice and gesture inputs, so it should not be too complex or it will negatively affect the user’s sense of immersion. To keep our GUI simple, all our graphical interfaces adopt the same style and format. We hide as much of the GUI as we can whenever possible, displaying it only when the user needs it. For example, initially the layout of the GUI only shows example commands (see Figure 16). But

if users need some tips about a particular command, they can use controllers to point the command and then the associated tips will be shown (see Figure 17).

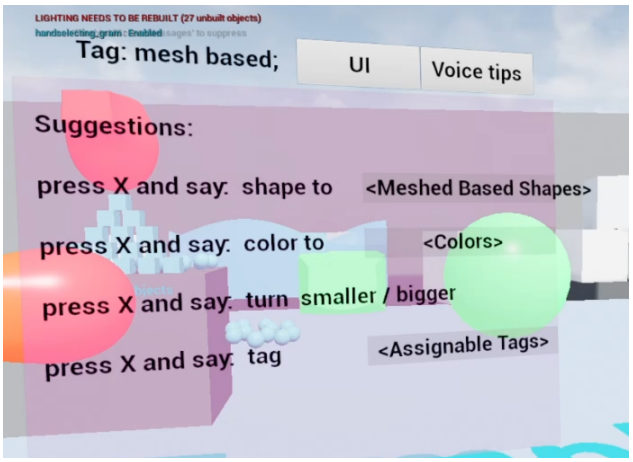


**Figure 16**: The GUI interface in our project without pointing at “Assignable Tags”

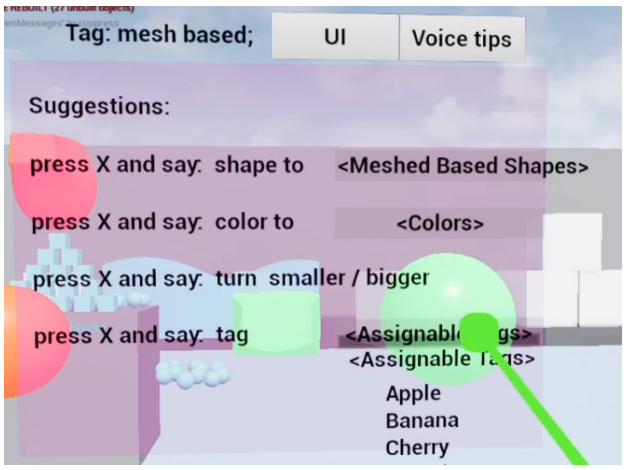


**Figure 17**: The GUI interface in our project after pointing at “Assignable Tags”

This method avoids creating multi-level hierarchical GUI, making interaction relatively simple.

## 3.2 Content creation

Meng uses ray marching to render agents and blend primitives. To simulate grouping behavior with agents, we implemented the Boids algorithm[20] and use GPGPU methods to run it more efficiently on the GPU. For rendering thousands of agents on the screen at the same time, we take the advantage of the powerful auto-instancing feature in Unreal Engine 4.

In the remainder of this section, we will discuss these techniques in detail to show how we use them to create and render agents in Meng.

### 3.2.1 Ray Marching and procedural shape generation

Meng uses a graphics rendering technique called ray marching. Ray marching enables us to combine geometries in novel ways and create forms like fractal geometries and area fog that are not possible using more traditional rendering techniques. Ray marching has also been explored in the live coding community. By using the ray marcher in a live coding system, programmers can generate many complex scenes without needing to know details about the underlying ray marching or lighting algorithm[21]. Thus, we implemented a ray marcher inside Meng which abstracts away the complexities of writing shader code and makes our system more accessible to people who do not have much graphics programming experience.

Ray marching is similar to traditional ray tracing in that a ray is cast into the scene. In ray tracing, the algorithm uses a set of equations that determine the intersection point of the ray and the object to be rendered. In this way, the object where the rays intersect can be found. Using this intersection information, we can decide what the color should be for each pixel based on how light in the scene strikes the intersection.

Ray marching uses a different algorithm to deal with the ray/object intersection problem. In contrast, we “move” a point along the ray until we find that the point intersects the object rather than trying to directly calculate the intersection analytically. Sampling this point along the ray is a relatively simple and cheap operation and has high real-time performance, although not as good as the raster algorithm used in most 3D graphics applications.

For the purpose of using UE4’s built-in lighting system for geometries we generated by ray marching, we were not doing full-screen ray marching, which is the way ray marching is most commonly used. Instead, we do ray marching in each local mesh. To be more specific, we are doing ray marching inside a sphere mesh for every single object. This is more efficient than full-screen ray marching, as the origin of each ray is not the camera, but rather the surface of the sphere mesh, as shown in Figure 18.

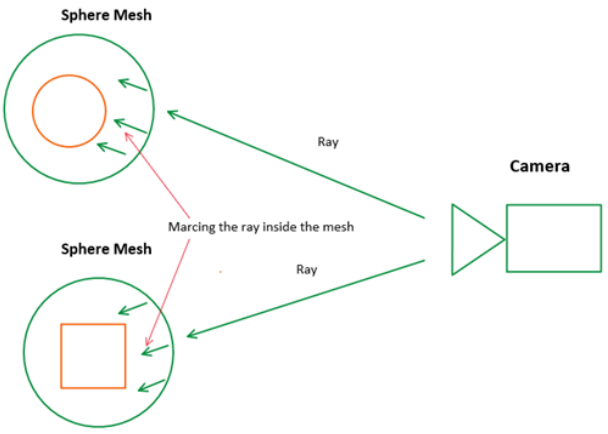


**Figure 18**: Ray Marching in Unreal Engine 4

- **Signed distance function and sphere marching:**

  To achieve better performance, we use signed distance functions and sphere marching optimizations in Meng’s ray marchers.

  Signed distance functions, or SDFs for short, are functions that take a point as input and return the shortest distance between the point and some surface. The sign of the return value shows whether the point is inside or outside the surface. We can use this return value as the step length of the ray to speed up the intersection process; this technique is also called sphere marching and is much quicker than advancing a uniform distance for every step.

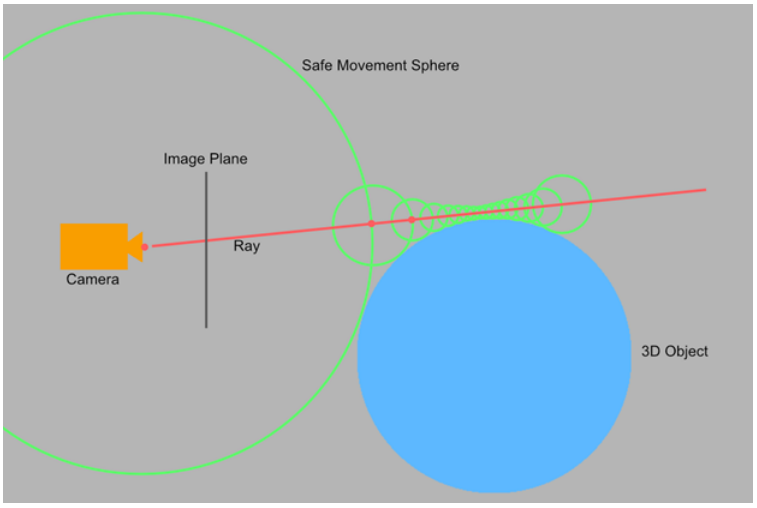


**Figure 19**: Using Sphere Marching as optimization

As long as we have the signed distance function of a certain shape, it means we can easily calculate the intersection between each ray and use this to render a variety of different shapes. We currently support creating spheres, boxes, round boxes, torus, links, cylinders, cones, and capsules in our project.

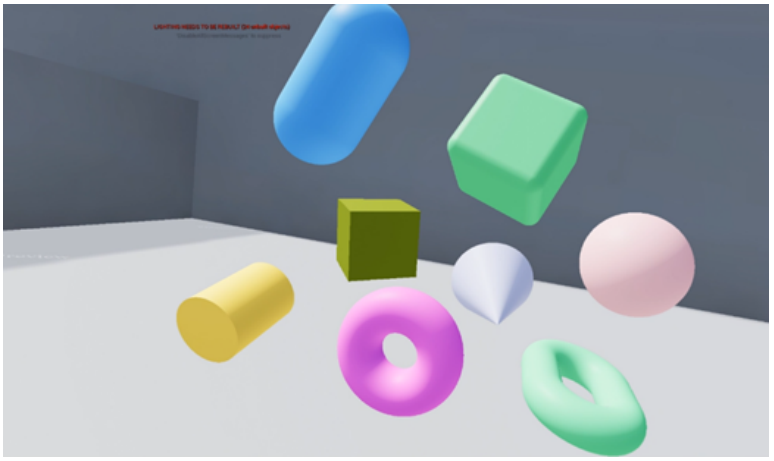

**Figure 20**: Geometries rendered by Ray Marching in Meng

- **Boolean operations and primitive blending:**

  We also implemented three operations for combining geometries created by ray marching. They correspond to the three basic Boolean operations (Union, Subtraction and Intersection) and enable us to achieve primitive combinations and even the blending of primitives without the geometric seams.

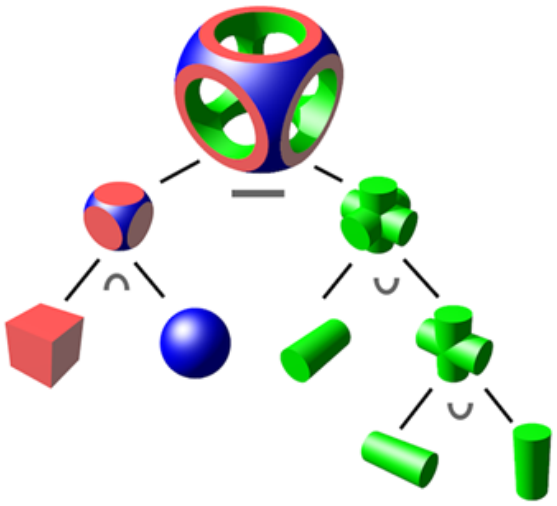

**Figure 21**: Boolean operations between shapes from "Constructive solid geometry" on Wikipedia[22]

These three operations can be expressed concisely when two geometries are expressed as SDFs.

We have distA as the return value of the SDF of geometry A and distB represent the return value of the SDF of geometry B. Then, we can use the formula below to blend them.

$$Union\ SDF\ =\ Min(distA,\ distB);$$

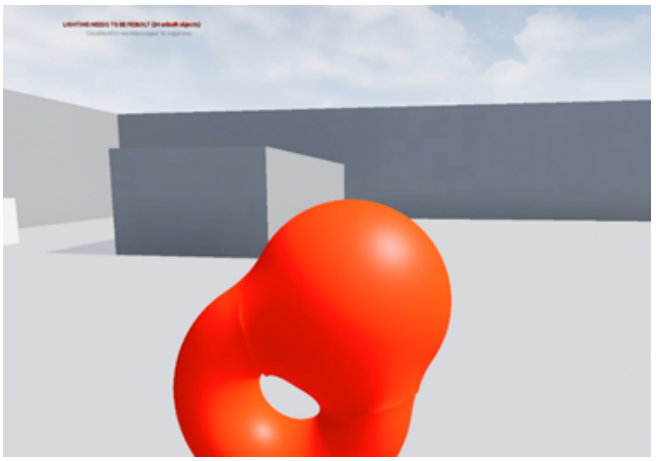

**Figure 22**: Blending primitives using “Union” operation

$$Subtraction\ SDF\ =\ Max(distA,\ -\ distB);$$

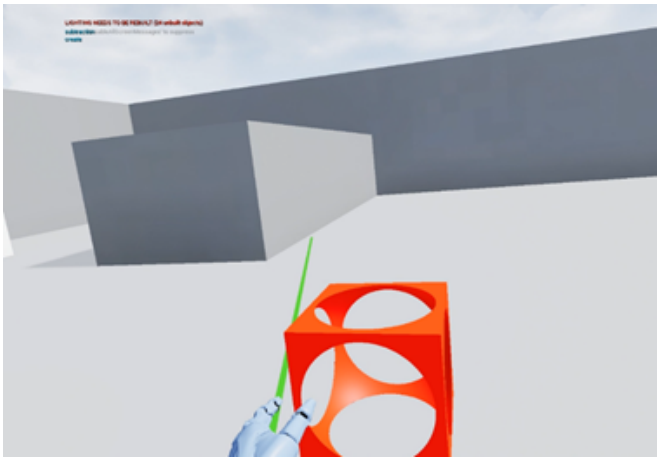

**Figure 23**: Blending primitives using “Subtraction” operation

$$Intersection\ SDF\ =\ Max(distA,\ distB);$$

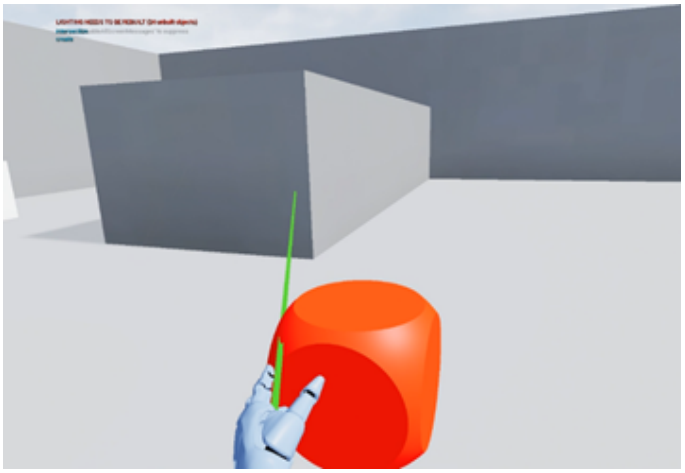

**Figure 24**: Blending primitives using “Intersection” operation

- **Fractal:**

  SDFs are excellent for rendering geometries that with bounds that cannot be computed analytically; this enables Meng to render more experimental forms. Here is an example of rendering Mandelbulb fractals with different parameters in Meng.

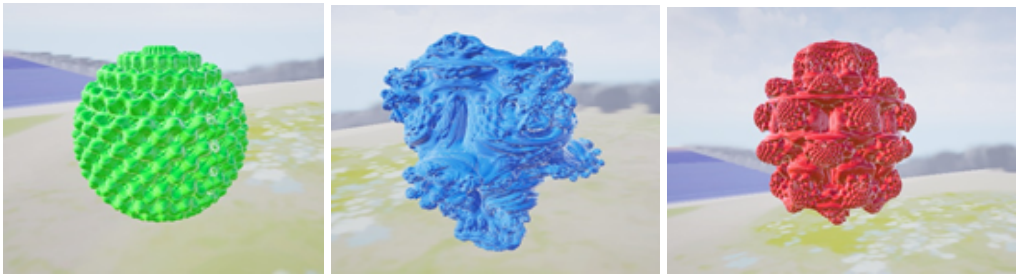

**Figure 25**: Mandelbulb fractal with different parameters

### 3.2.2 Agent grouping behavior

In Meng, we not only provide the function of using gestures to customize agents’ moving path, but also we also allow users to create a group of agents that move together. Using this feature, users can create a group of agents that exhibit flocking behavior.

- **Boids:**

  To implement Meng’s flocking simulation, we implemented Boids algorithm which is a classic algorithm to simulate grouping behavior within a swarm of agents.

  “The complexity of Boids arises from the interaction of individual agents (the boids, in this case) adhering to a set of simple rules. The rules applied in the Boids world are as follows:

  - Separation: steer to avoid crowding local flockmates
  - Alignment: steer towards the average heading of local flockmates
  - Cohesion: steer to move towards the average position (center of mass) of local flockmates”[23]

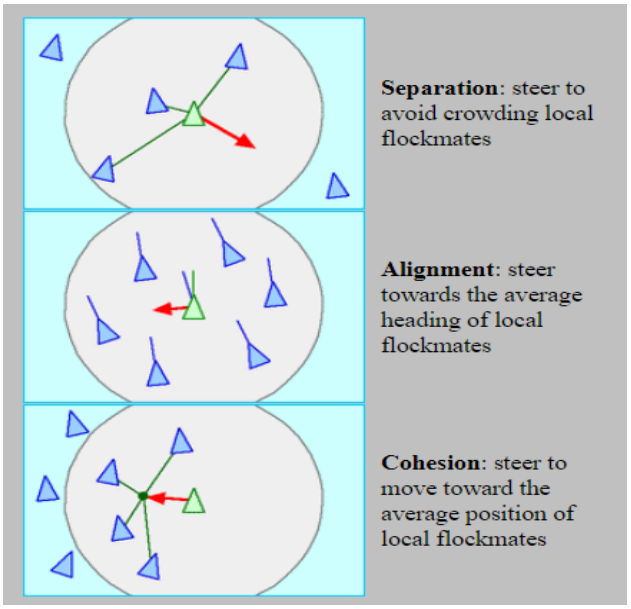


**Figure 26**: Boids Algorithm

- **Using the parallel computing power of the GPU**

  If we look closely at the Boids algorithm, it is not difficult to find that when we are calculating the behavior of an agent, we need to consider all the agents around it. This means that the time complexity of this algorithm is O (n*n). When the number of agents increases, the running time of the algorithm will also increase exponentially. But what if we use parallel computing? If we can perform calculations for each agent in parallel on different cores at the same time, the efficiency of the algorithm will be greatly improved. Therefore, we introduced GPGPU technology to solve this problem.

  GPGPU is an abbreviation of General-purpose computing on graphics processing units. We used this technology to run Boids algorithms on the GPU side that is better at doing parallel computing, using Compute Shader.

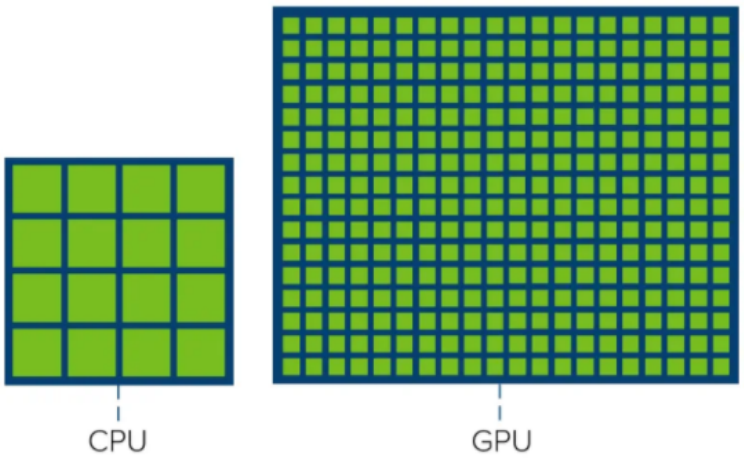


**Figure 27**: Comparison number of cores on CPU and GPU

  We dispatched Compute Shaders every frame and used it to calculate position and velocity for every agent and then save the result to textures. Such processing allows us to make full use of the computing power of GPU as well as free up the computing power of CPU and makes a balance to achieve better performance.

  We are able to run the flocking simulation for thousands of agents on the screen at the same time with this GPGPU method.

- **Rendering thousands of agents:**

  How to efficiently render thousands of objects in each frame of video has always been a problem that real time applications are committed to solving. Realtime rendering programs are always required to run at a frame rate of at least 30 FPS. To render as

many objects as possible in such a short time slice, it is necessary to give full play to the rendering capabilities of the GPU and perform specific optimizations. In order to solve this problem, our project uses two different ways to render the agents in the group.

(1) Rendering agents by Static Mesh

For the flock flying across the scene, we use Unreal engine’s built-in static mesh component to render them. Leveraging the auto-instancing function provided by the unreal engine, we can render thousands of agents with just one draw call. The cost is that the group of agents we render share the same material. In this way, the unreal engine will automatically merge the rendering of these same agents into one rendering batch to greatly improve the rendering efficiency.

**Figure 28**: Thousands of Boids flying

(2) Rendering by Instanced Static Mesh.

UE4 provides a component called the instanced Static Mesh Component for GPU instancing. As its name suggests, it will batch all the instances and render them by one draw call. For the flock in which agents fly around in a small area, we use this method to finish the rendering task.

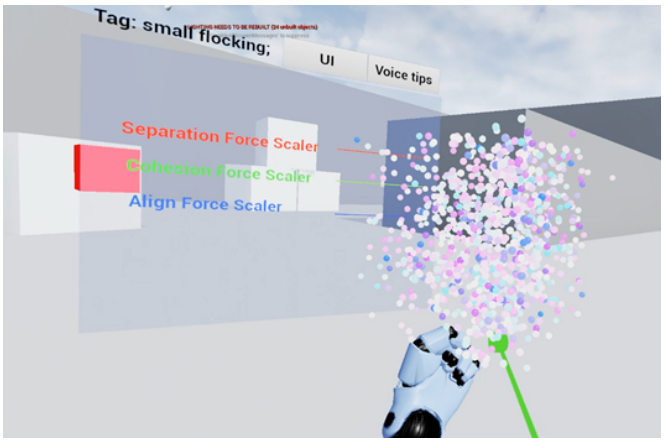


**Figure 29**: Thousands of particles flying near hand

# 4 EVALUATION

The results of the experimental evaluation are split into two parts. The first part discusses the results that came from a user study we led. In the second part, based on the results we got from the first part, we used the Cognitive Dimensions of Notation framework, or CDN for short, to compare our system with a traditional 2D web-based live coding environment and analyze the differences and similarities.

## 4.1 User study

COVID-19 changed our planned evaluation, as we felt it was no longer safe to conduct in-person user studies on shared VR equipment. This meant that user study participants would have to run our system on their personal computers, and that they might not have access to VR equipment. Accordingly, to conduct our evaluation we ported our project from running on VR hardware to running on standard commodity desktops running Windows. While this is imperfect given Meng’s emphasis on using embodied gesture, it does still enable users to create worlds using both voice and GUI input, and we conducted our tests with participants using these two input mechanisms.

### 4.1.1 Protocol Setup

Instead of requiring VR controllers, we enabled users to use keyboard and mouse to control the PC version of our system in a classic FPS control mode (using WADS on keyboard to move and the mouse to rotate the camera). We retain the system logic and almost all system features of the original VR version in this PC version—except for using VR control devices to draw paths—to ensure the effectiveness of evaluation. We distributed the PC version through the Internet, so that playtesting could be conducted remotely.

To help users get familiar with our project quickly, we created a tutorial which includes three parts. The first part teaches users how to interact with the UI system; the second part teaches users about the voice commands; and the third part is a mini task that requires users to complete without tips. In the third part, users can choose the GUI system or their voice as the main input mechanism, using their choice of GUI or voice commands to create an object with some special properties. Also, if participants forget necessary commands, they can press T on their keyboard to get hints. The purpose was to examine the acceptance of the new input mechanism in this new live coding environment and the difficulty of learning this project.

In this evaluation process, participants should try their best to finish the tutorial. Theoretically, after following all the steps of the tutorial system, users should have a basic understanding of our project. To obtain information so that we can improve the design of our system, we asked users to complete a survey containing questions about their experience (see Appendix 1).

After all preparations were complete, the IRB approved our project (IRB 21-0485, see Appendix 2, Appendix 3 and Appendix 4) and we invited participants from WPI IMGD major via an email that included a link to an executable of our project and the survey.

### 4.1.2 Testing Results

In the total we received 19 replies; of these about 50% had previously used a live coding system. The participants were asked to give their graphics cards information and rate the overall performance of running Meng on their computers. Based on their answers, most of them use dedicated graphics cards and stated that Meng ran smoothly on their computers (see Figure 30).

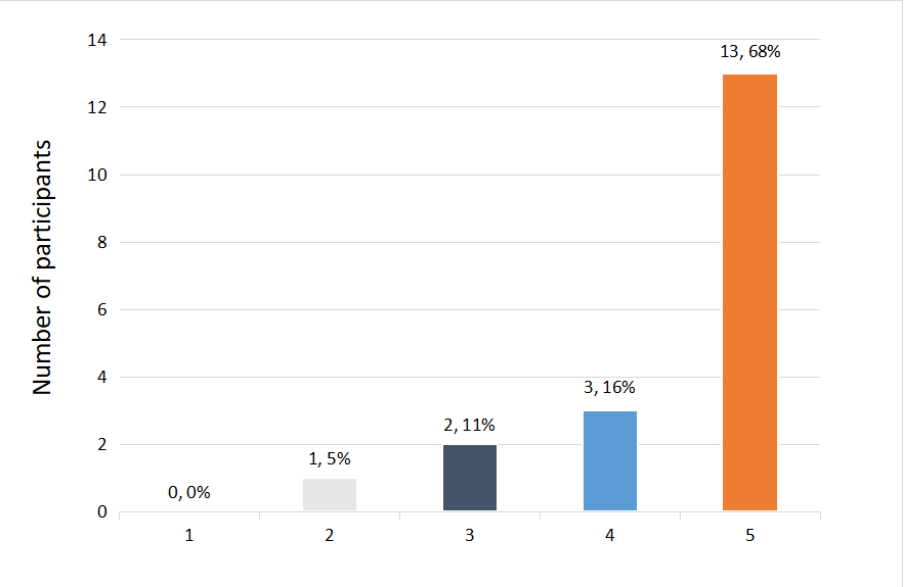


**Figure 30**: Value of 5 means “High (Running smoothly)” and value of 1 means “Low ( Stuttering / Frame Drops can be clearly identified)”.

78.9% participants tried to experiment with other ideas that were not covered in the task (see Figure 31), which means most of participants could infer many commands they were not explicitly taught by inference. Except for 7 participants who did not get to the third part of the tutorial, we knew that more than half participants finished the third part without using hints, as shown in Figure 32. All of these indicate that learning how to interact with the environment in our project is not difficult.

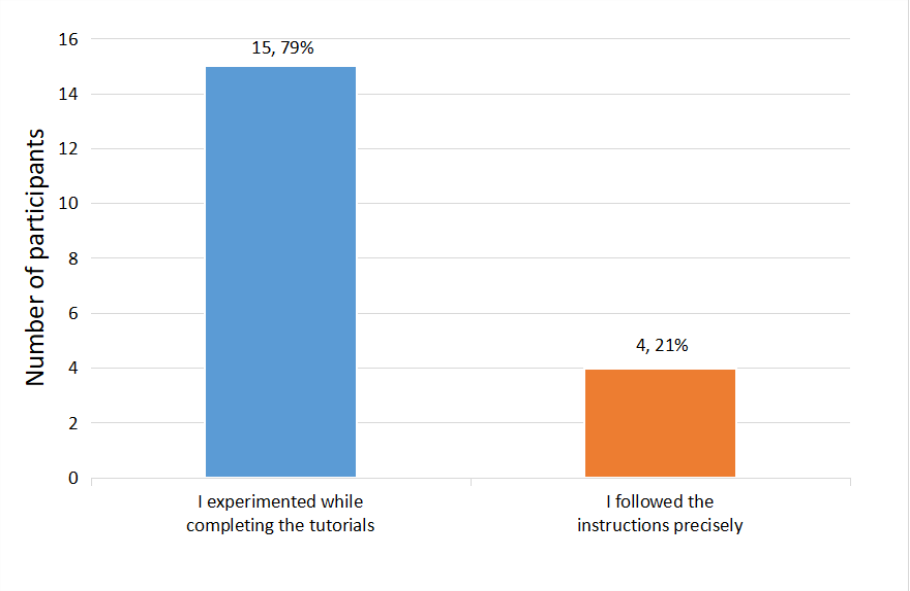


**Figure 31**: Distribution of participants following the instructions

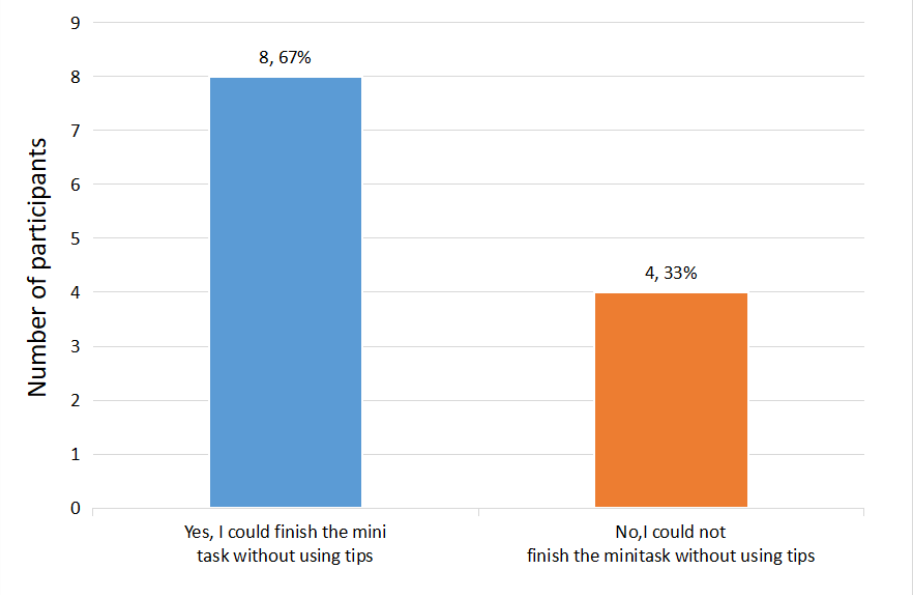


**Figure 32**: Distribution of participants who did not use tips in the mini task

As noted before, using traditional input mechanisms seems inappropriate for programming in VR, especially for live coding shows which need quick commands and feedback. Hence, an important question for this user study is whether the new input mechanisms that we created are acceptable for users. From the results, half participants preferred using the voice input system. And they stated, in the survey, that the reasons why their first choice was voice commands were that it was “more convenient”, and “was more interesting than using GUI system”. The rest of participants stated that the reason they chose the GUI system was that they were accustomed to using GUIs. As Figure 33 shows, many participants complained about the recognition rate of the voice input system, but they still showed high acceptance of

this new input mechanism. As expected, most participants were quite satisfied with combining voice and GUI to be the main input mechanisms (see Figure 34).

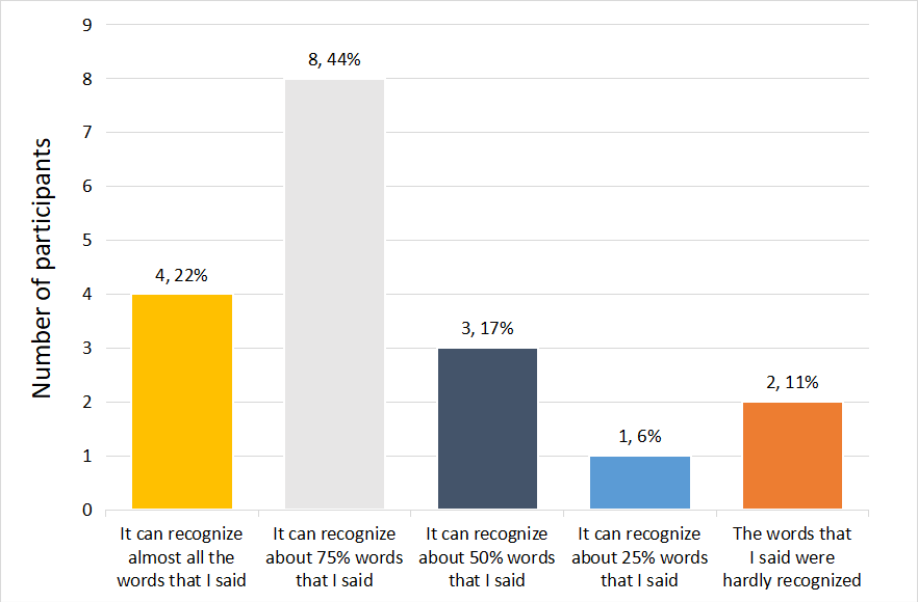


**Figure 33**: Distribution of recognition rate of voice input

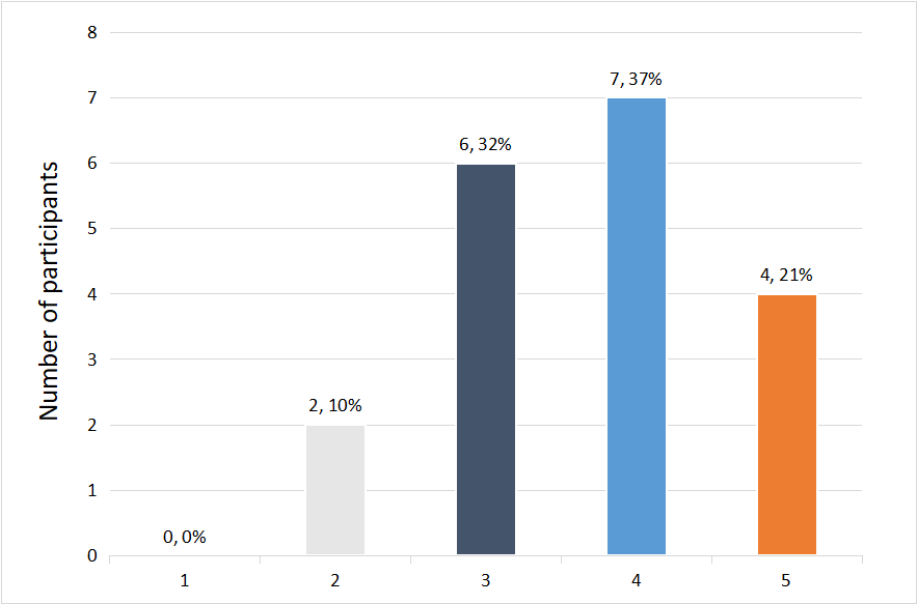


**Figure 34**: Value of 5 means “I enjoy it very much” and a value of 1 means “I did not like it”.

## 4.2 Compared to Livecodelab with CDN

Livecodelab is a web-based live coding environment. The language it created is simple but can support some complex graphics and audio programming. The developers first released Livecodelab nine years ago, and it has evolved into a mature and widely-used live coding environment.

Cognitive Dimensions of Notation framework is a common framework in the psychology of the programming community. The whole framework has 13 dimensions but analysis often foreground a subset of these. We chose four of them to use in our analysis by selecting the four that seemed most relevant to live coding.

- **Progressive evaluation**:

  Progressive evaluation refers to the ability of a programming environment to support “partially-finished” programs so that programmers—especially novices—can evaluate their own problem-solving progress at frequent intervals and obtain feedback on “How am I doing?”.

  In Livecodelab, each line of code represents a command and is a part of a whole. Any edits to code are evaluated immediately, without requiring additional user interaction. So users, even novices, can see problems in their programs very quickly and then modify code as needed.

  The collection of all the commands for agent creation, modification and operation can be regarded as a mini program in Meng. Similar to Livecodelab, each command executed for creating/modifying/operating agents in Meng can be evaluated immediately. For easy debugging, we implemented a log system and log interface to help users check each command. For example, our project has a command called “New Cube”. If users only input “New”, the command will be considered as an incomplete command. The log interface will display error-message to help users debug and encourage progresive evaluation.

- **Consistency:**

  The authors of CDN take consistency as a particular form of guessability: when some of the language has been learnt, how much of the rest can be inferred? They believe that a consistent language usually can be learned more readily.

  As we mentioned before, live coding performers should enter instructions quickly. Lots of live coding systems create their new languages to fit this rule and Livecodelab is no exception. The commands in Livecodelab are similar. For example, if users know how to create a cube and change its color, they will know how to create a ball with different colors.

  When trying to create commands for our project, we considered the integrity and consistency. For all voice commands, we used Sphinx grammars to enforce a consistent structure; we made GUI adopt the same structure and ordering.. Thus, users can learn how to interact with Meng quickly and infer many commands they were not explicitly taught.This has been shown from results of the user study (see Figure 31 and Figure 32 above).

- **Viscosity:**

  CDN defines Viscosity in programming languages (and other information structures) as how much effort is required to perform a single change. Though it depends on the precise change and languages/environments,  it seems that in live coding performance, reduced viscosity is often preferred.

  Livecodelab needs users to enter the exact number when adjusting agents’ parameters. For example, when changing colors, users should search the RGB number, and then enter it in the right place. Sometimes they need to enter different numbers multiple times to adjust to the appropriate color. This process wastes much time.

  We have introduced the GUI system to our project for adjusting agents’ parameters. Basically, users can just hold the slider and adjust its parameters. This way seems more convenient because users not only do not need to enter instructions repeatedly, but also do not need to search for any other information.

- **Premature commitment:**

  It is hard for writers to write a table of contents list before writing a book. Similarly, forcing users to make a decision before the information is available can be difficult for them especially those live coding performers.

  To avoid it, in Livecodelab, users do not need to do system design before coding. They are allowed to do live coding shows impromptu and add any sentences to the internal structure or delete some of them during performance.

  Our system is dedicated to providing an embodied way to interact with the virtual live coding system. We do not require our users to have any prior experience of programming or using our system. And we do not require users to decide what they need to create for the project before exploring it.

This comparison indicates that the new interactive modes provided by our system can meet the needs of live coding—the language is simple enough to learn, all commands can be executed immediately and it is not a fluid system (mentioned in CDN).

# 5 CONCLUSION

Live coding environments enable users to create audiovisual works that modify a running system, without requiring the system to be restarted. They are often used in live performances. Our project aims to take advantage of the visual display of VR to make the performance more immersive, and the controllers of typical VR environments to make live coding more embodied. Thus, we present a programming environment in virtual reality for live coding performers. In order to support more intuitive and more appropriate interactions in VR, we introduce voice and gesture recognition combined with the GUI system to Meng and believe this is preferable to using virtual keyboards for programming in VR.

Meng has different aesthetics from other live coding systems. The major visual features in Meng come from the ray marcher we implemented, which supports primitive combination and blending, allowing users to combine existing geometries into new shapes; it also supports more advanced features such as rendering volumetric fog and fractal geometries.

Our user study indicated a high degree of satisfaction with the input methods of Meng. However, still many participants are dissatisfied with the voice recognition rate; improving this will be an important component of our future work. Other areas we plan to work on in the future include:

- The ray marcher in our system now supports using boolean operations to combine two geometries. We will add support for blending multiple geometries using boolean operations.
- We run the boids algorithm on GPUs that are good at doing parallel computing for better performance in Meng. In fact, the boids algorithm still has room for optimization. We are considering using the spatial search algorithm to speed up the agent's query process for its surrounding agents.
- We plan to develop a GUI to visualize the tags of all agents in the scene. In this way, the user can conveniently use the voice to query the corresponding agent according to the displayed tags.
- We plan to implement undo functionality in a future version to allow users to easily reverse earlier commands.
- We plan to conduct a long term study to observe whether users are still satisfied with combining voice and GUI as inputs after using them for longer periods of time. We will also attempt to evaluate if the worlds users make with the features Meng provides meets their goals and expectations.

During the development of Meng we have contributed a number of open-source libraries:

- **“Meng” - Live Coding in Virtual Reality:**
  https://github.com/aceyan/Live-Coding-Virtual-Environment
  The Github repository of our system including the evaluation version.

- **UE4 Raymarching Plugin:**
  https://github.com/aceyan/UE4_RayMarchingPlugin
  A plugin enables users using ray marching in Unreal Engine 4.

- **UE4 GPGPU Flocking/Boids:**
  https://github.com/aceyan/UE4_GPGPU_flocking
  A plugin enables simulting flocking behavior using GPGPU in Unreal Engine4.

- **UE4 Google Speech to Text Plugin:**
  https://github.com/Vakarian15/UE4-Google-Speech-to-Text-Plugin
  A plugin enables users to record their speech and recognize it through Google Speech-to-Text.

# 6 REFERENCES

[1] Tanimoto, S. L. (2013). A perspective on the evolution of live programming. 2013 1st International Workshop on Live Programming (LIVE). doi:10.1109/live.2013.6617346

[2] Cruz-Neira, C., Sandin, D. J., & DeFanti, T. A. (1993, September 1). Surround-screen projection-based virtual reality: the design and implementation of the CAVE. Retrieved from https://dl.acm.org/doi/10.1145/166117.166134

[3] Oculus Quest: All-in-One VR Headset. (n.d.). Retrieved from https://www.oculus.com/quest/

[4] Tuy, Heang K, and Lee Tan Tuy. 1984. “Direct 2-d Display of 3-d Objects.” IEEE Computer Graphics and Applications 4 (10). IEEE: 29–34.

[5] Green, T., & Petre, M. (1996). Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework. *Journal of Visual Languages & Computing, 7*(2), 131-174. doi:10.1006/jvlc.1996.0009

[6] Wakefield, Graham & Roberts, Charlie & Wood, Timothy & Yerkes, Karl. (2014). Collaborative Live-Coding Virtual Worlds with an Immersive Instrument. In Proceedings of the International Conference on New Interfaces for Musical Expression Conference.

[7] J. Kuchera-Morin, M. Wright, G. Wakefield, C. Roberts, D. Adderton, B. Sajadi, T. H¨ollerer, and A. Majumder. Immersive full-surround multi-user system design. Computers & Graphics, 2014.

[8] Fischer, M. H. (2016). Inception. Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology - VRST 16. doi: 10.1145/2993369.2996354

[9] Oberhauser, R., & Lecon, C. (2017). Virtual Reality Flythrough of Program Code Structures. Proceedings of the Virtual Reality International Conference - Laval Virtual 2017 on - VRIC 17. doi: 10.1145/3110292.3110303

[10] Vincur, J., Konopka, M., Tvarozek, J., Hoang, M., & Navrat, P. (2017). Cubely. Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology. doi: 10.1145/3139131.3141785

[11] Ortega, F. R., Bolivar, S., Bernal, J., Galvan, A., Tarre, K., Rishe, N., & Barreto, A. (2017). Towards a 3D Virtual Programming Language to increase the number of women in computer science education. 2017 IEEE Virtual Reality Workshop on K-12 Embodied Learning through Virtual & Augmented Reality (KELVAR). doi: 10.1109/kelvar.2017.7961558

[12] LiveCodeLab, D., Casa, D., LiveCodeLab, LiveCodeLab, G., John, G., Leeds, U., . . . Authors: Davide Della Casa LiveCodeLab. (2014, September 01). LiveCodeLab 2.0 and its LANGUAGE LIVECODELANG. Retrieved May 04, 2021, from https://dl.acm.org/doi/10.1145/2633638.2633650

[13] Aaron, Samuel, Blackwell, Alan, Hoadley, Richard, & Regan, Tim. (2011). A Principled Approach to Developing New Languages for Live Coding. In Proceedings of the International Conference on New Interfaces for Musical Expression (pp. 381–386). Zenodo. http://doi.org/10.5281/zenodo.1177935

[14] LEE, K., HON, H., & REDDY, R. (1990). An overview of the sphinx speech recognition system. Readings in Speech Recognition, 600-610. doi:10.1016/b978-0-08-051584-7.50056-5

[15] Google Speech-to-Text. https://cloud.google.com/text-to-speech/

[16] Shane, Colbert. https://github.com/shanecolb/sphinx-ue4

[17] Jspeech grammar format. (n.d.). Retrieved April 29, 2021, from https://www.w3.org/TR/jsgf/

[18] Këpuska, V. (2017). Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx). International Journal Of Engineering Research And Applications, 07(03), 20-24. doi: 10.9790/9622-0703022024

[19] Rune Berg. https://github.com/1runeberg/RunebergVRPlugin

[20] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH 87. doi: 10.1145/37401.37406

[21] Charlie Roberts. (2019). Live Coding Ray Marchers with Marching.js. In Proceedings of the Fourth International Conference on Live Coding (p. 353). Madrid, Spain: Medialab Prado / Madrid Destino. http://doi.org/10.5281/zenodo.3946267

[22] Constructive solid geometry. (2021, February 02). Retrieved April 29, 2021, from https://en.wikipedia.org/wiki/Constructive_solid_geometry

[23] Boids. (2021, January 13). Retrieved April 29, 2021, from https://en.wikipedia.org/wiki/Boids

# 7 APPENDICES

[1] Survey Questions

[2] IRB Application

[3] User Study Protocol

[4] User Study Consent Form