

SURFACE TRACE :

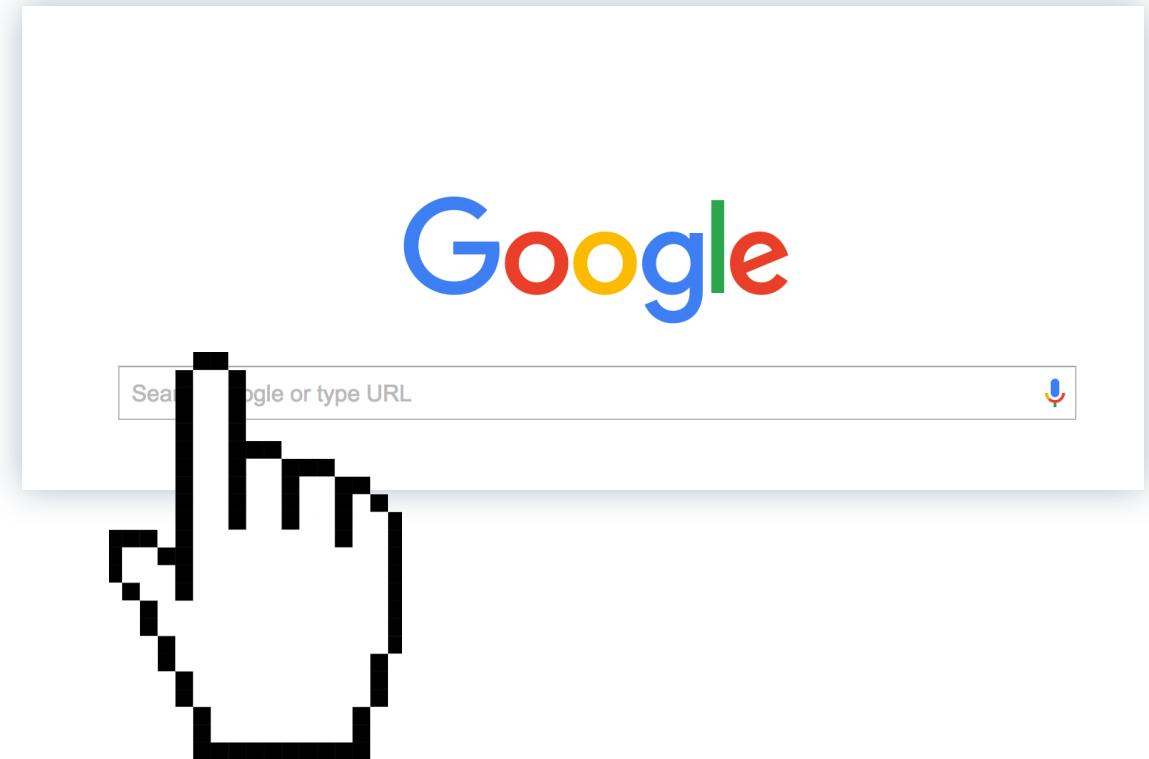
**AN EXPERIMENTAL SEARCH
INTERFACE**

TYPOGRAPHY AND INTERFACES

The typography that most of us consume most frequently is coded typography linked to computational systems. The type that constitutes our digital world watches and consumes our behaviour and habits.

“BEING ON THE INTERNET MEANS BEING TRACEABLE”

- Critical Atlass of the Internet



TYPOGRAPHY AND INTERFACES

ONE SIZE FITS NONE : TOWARDS CRITICAL INTERFACE DESIGN BY ROSA LLOP

- Interfaces should be challenging rather than accessible

Interfaces should be challenging for advanced users and for the mainstream, so that interaction with the system is an intellectually motivating experience.

- Interfaces should be tolerant but also sceptical

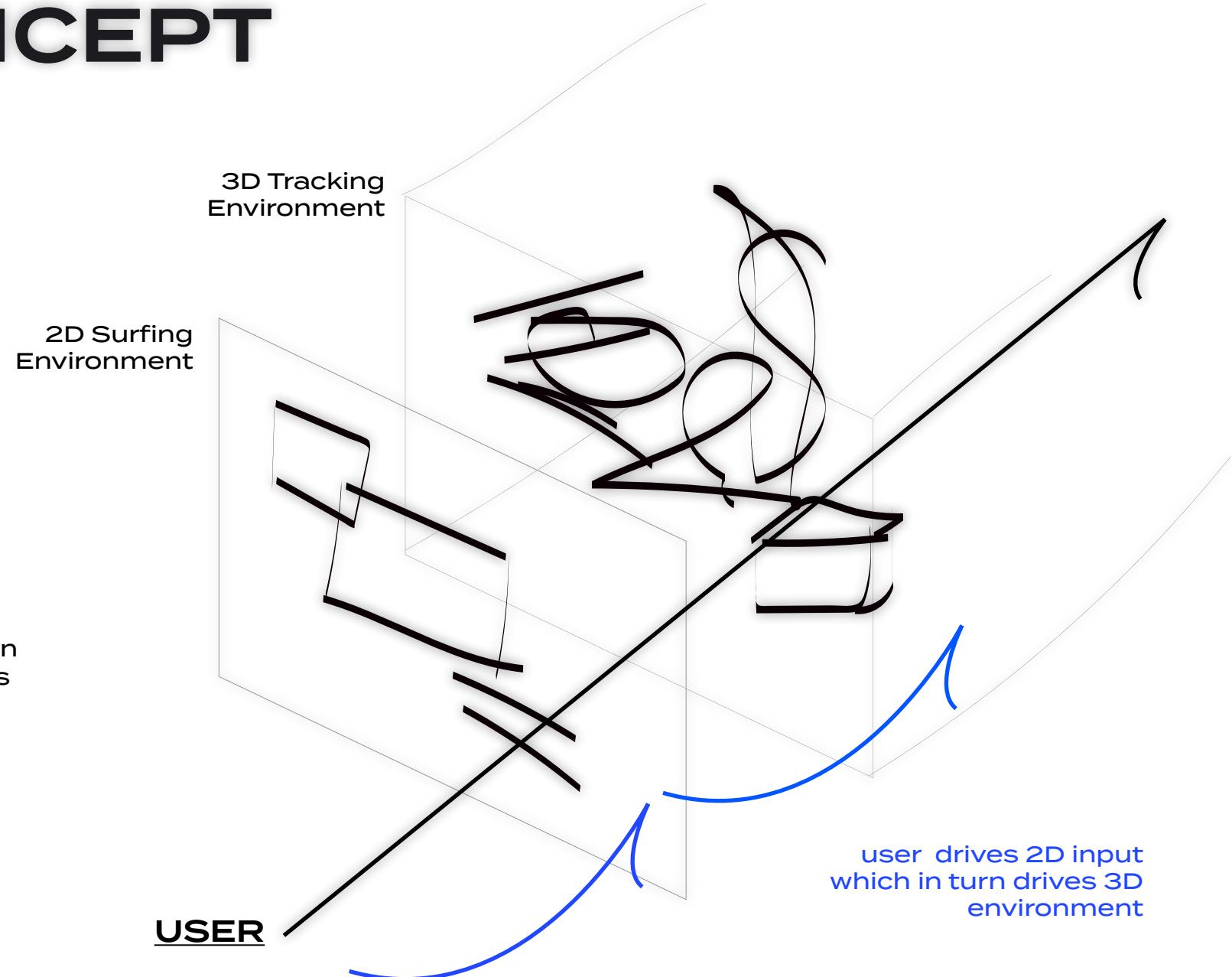
I propose that interfaces be tolerant, but also sceptical. That they encourage users to be curious about how they interact with them, help to disclose the values that arise from their use, and promote more conscious and responsible behaviour.

- Interfaces should offer a sense of empathy rather than control

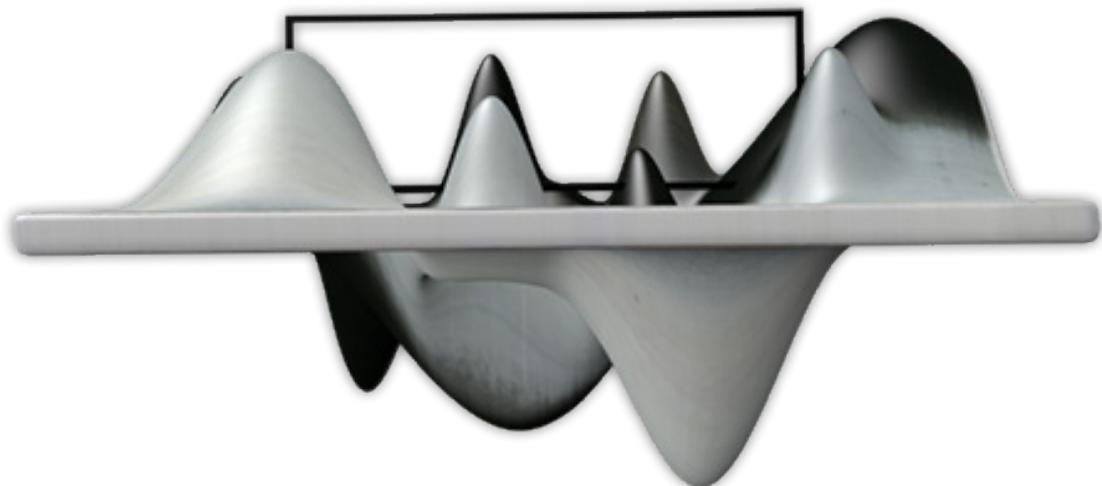
That instead of giving signs that they are executing the requested action they send signs that show the complexity of the system, generate curiosity, and encourage an attitude of exploration and learning.

INITIAL CONCEPT

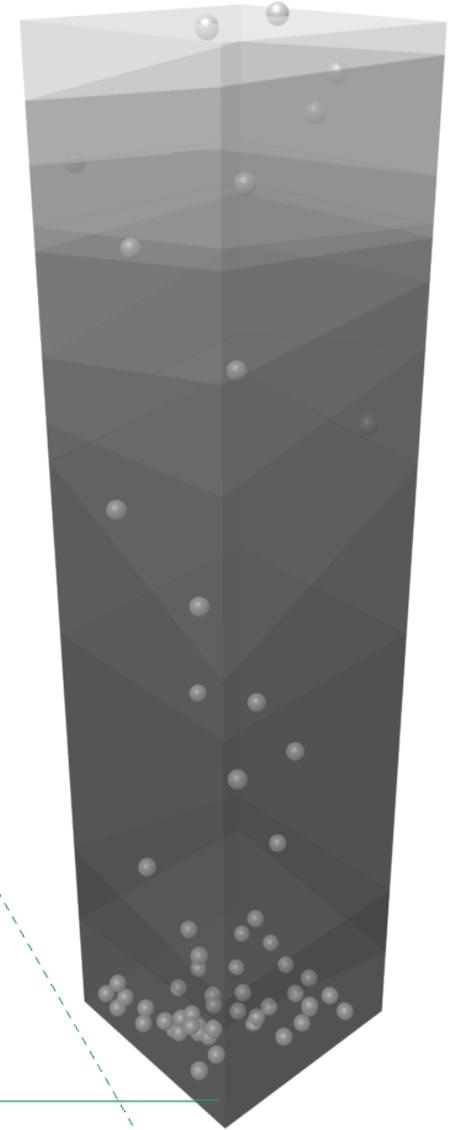
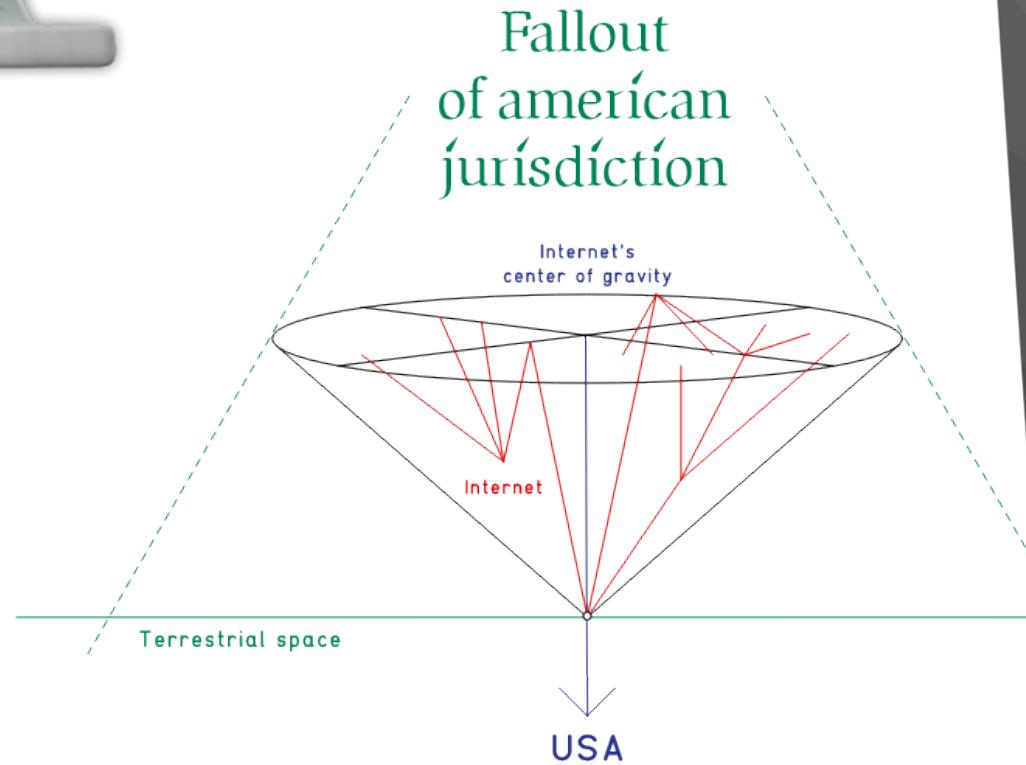
An experimental generative search interface that encourages user interaction and questions user interaction and stores the metadata of this interaction to be displayed.



MAPPING AS A TOOL FOR UNDERSTANDING



Representing ideas and processes in a visual or spacial way helps us to connect to those things and understand them in a deeper way.



PROCESS

EXPERIMENTING WITH METHODS OF TRACKING

```
50 |     this.y = y;
51 |     this.z = z;
52 |
53 |
54 |     this.history = [];
55 |
56 ▼ this.update = function() {
57 |     this.y = mouseY - windowHeight/2;
58 |     this.x = mouseX - windowWidth/2;
59 |     angle += 0.3;
60 |     this.z = sin(frameCount*0.001)*800;
61 |
62 |     var v = createVector(this.x, this.y, this.z);
63 |     this.history.push(v);
64 |
65 ▼ if (this.history.length > 10000) {
66 |     this.history.splice(0, 1);
67 |
68 }
69 |
70 ▼ this.show = function() {
71 |     noStroke();
72 |     fill(0);
73 |     strokeWeight(10);
74 |     beginShape();
75 |
76 |     fill(0, 0, 255);
77 |     stroke(0);
78 ▼     for (var j = 0; j < 10; j++) {
79 ▼         for (var i = 0; i < this.history.length; i++) {
80 |             var pos = this.history[i];
81 |             vertex(pos.x + 2, pos.y, pos.z);
82 |
83 |
84 }
```

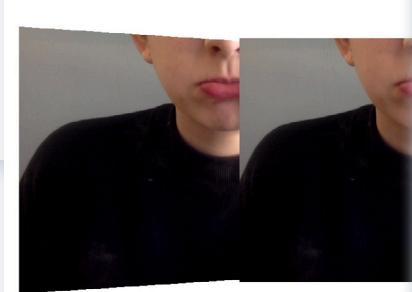
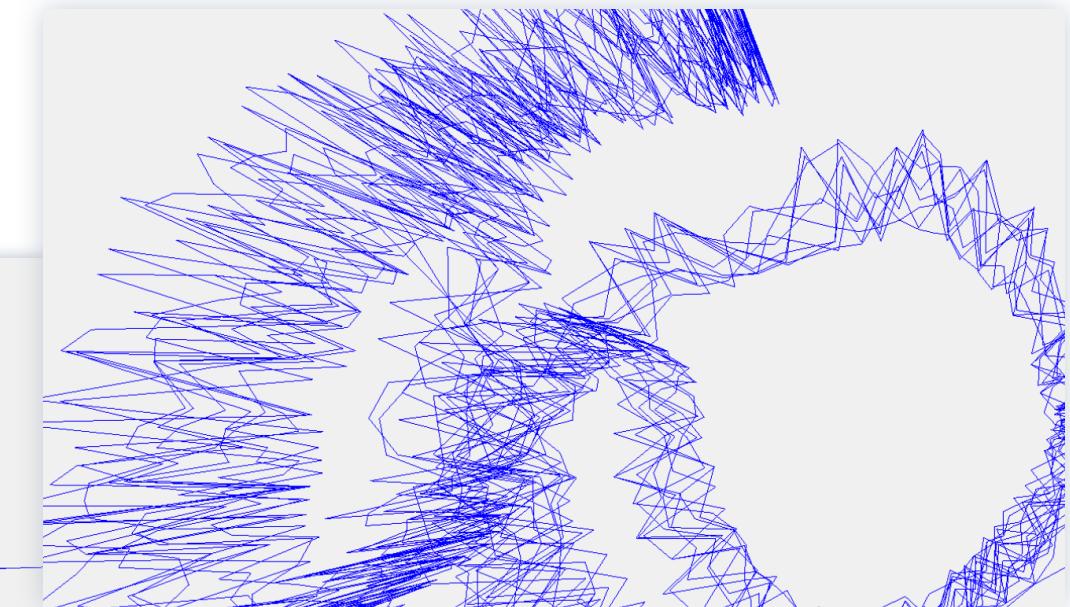


Photo taken and displayed each time mouse clicked:

Mouse position logged in 3D space:



PROCESS

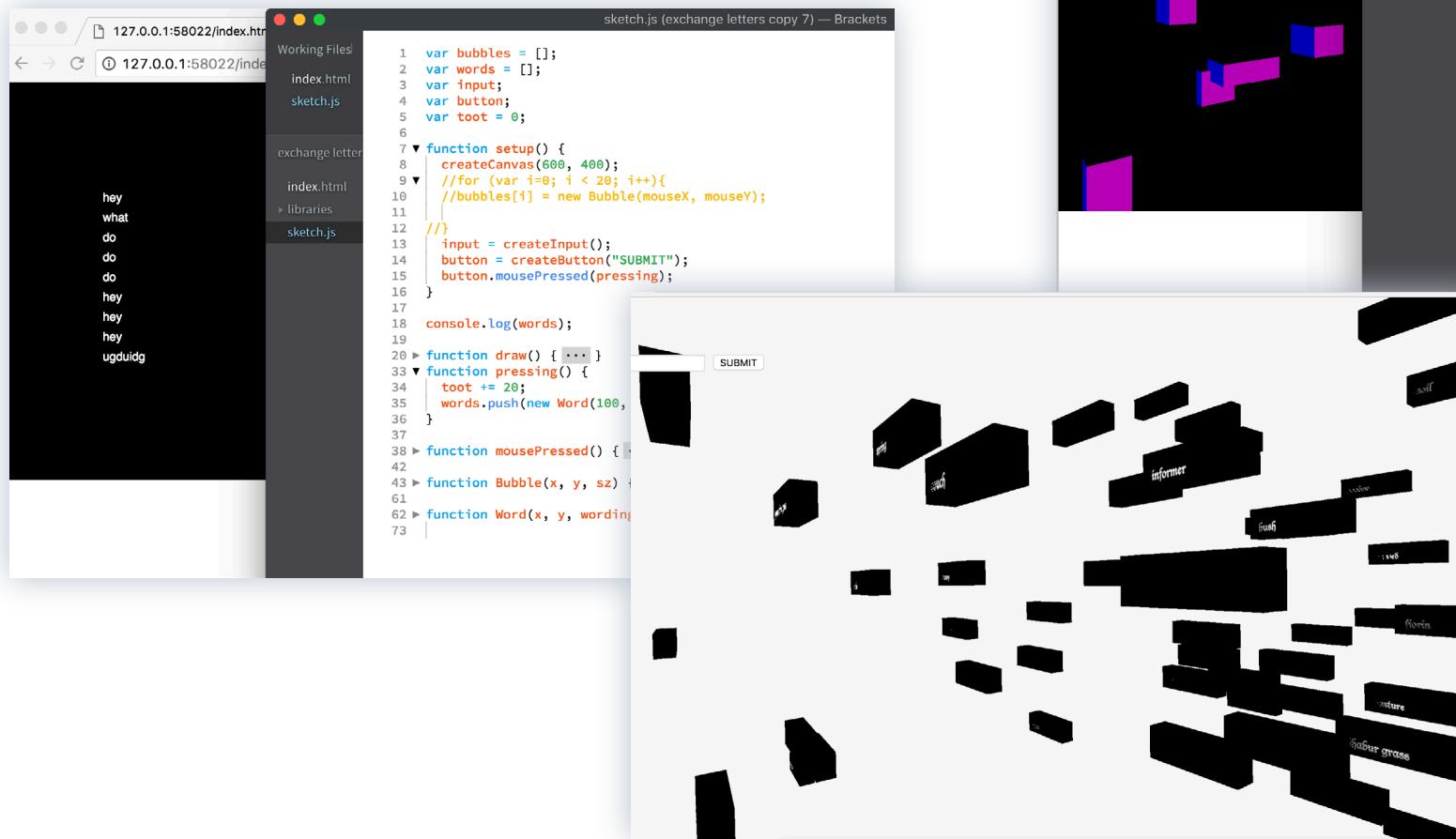
METHODS OF TRACKING

- Mouse Position
Mouse position logged and mapped in 3D space as well as mouse position affecting camera position / angle.
- Images (giphy)
Returned and displayed on objects
- Network of extra data
related words (from wordnik) produced and displayed in sky
- Searched Terms
Displayed on ground, constructing the walls of the environment

My design involved a number of different complex parts so I constructed each individually, gradually combining them to form the whole.

PROCESS

NETWORK OF RELATED DATA

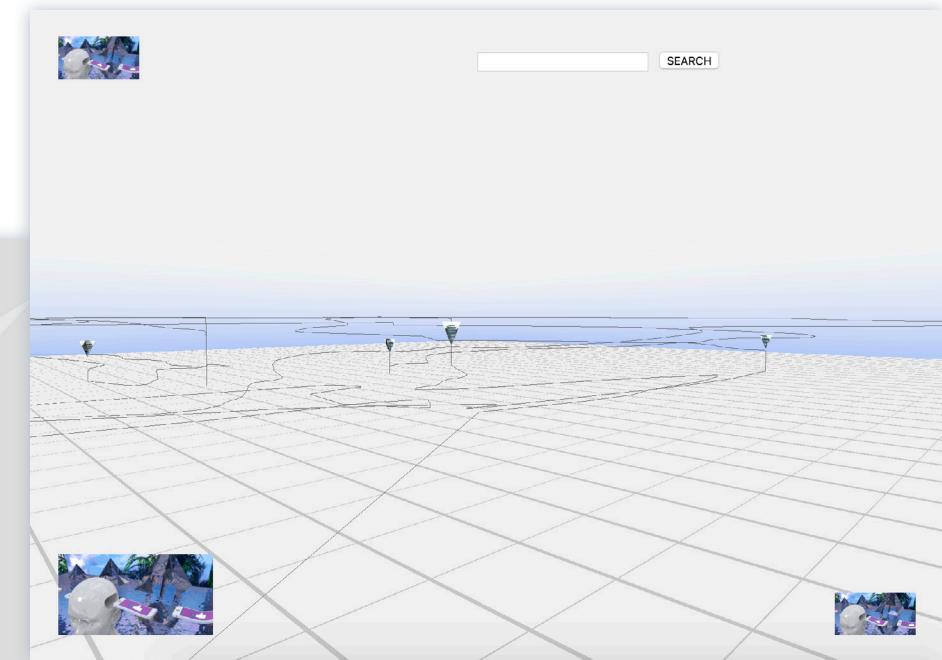
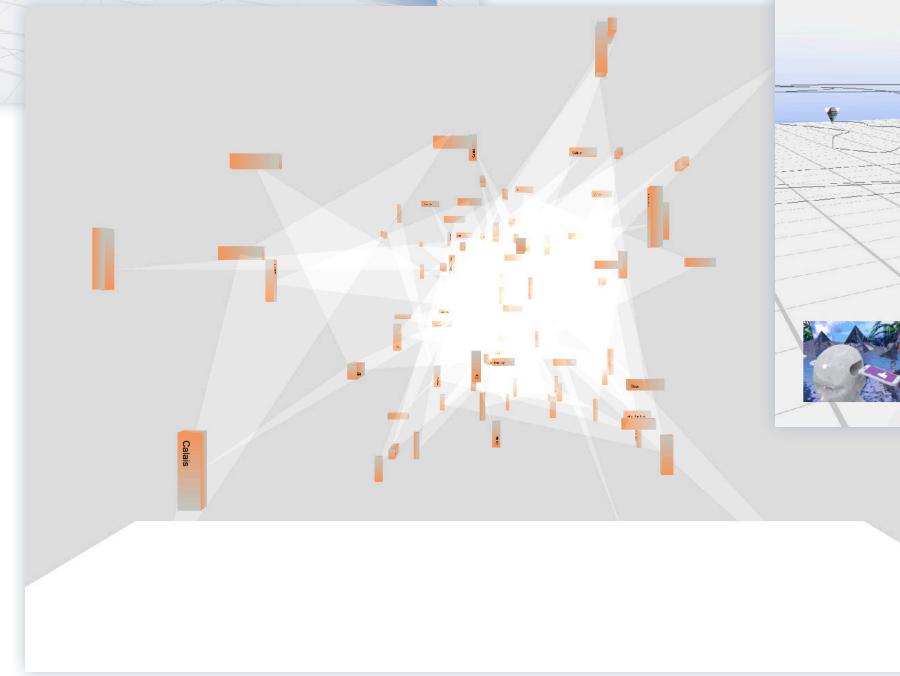
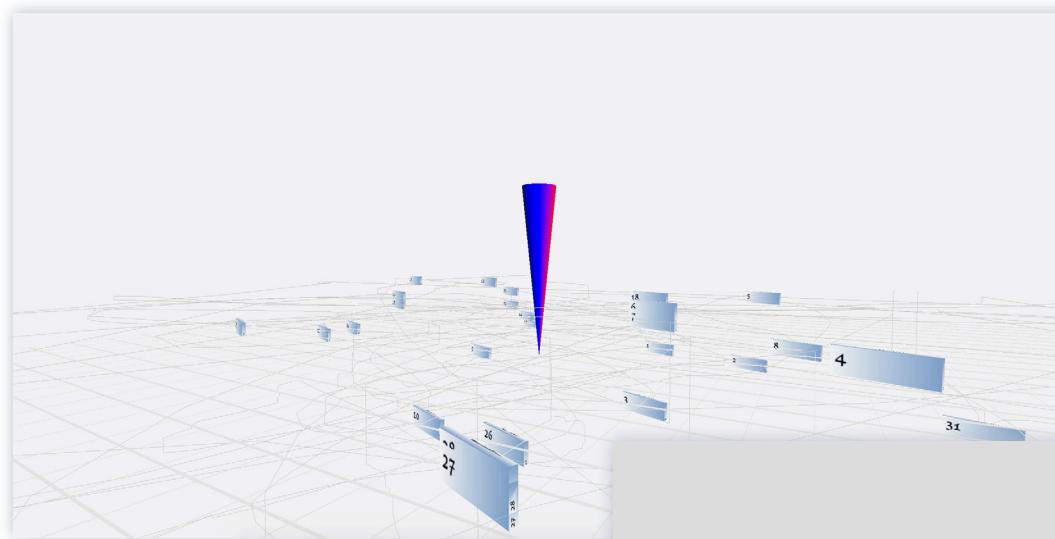


The image shows a Brackets code editor with a sketch.js file open. The code has been modified to create a 3D word cloud. It includes functions for setup, draw, pressing, and mousePressed. The draw function contains logic to move and display bubbles. The pressing function adds words to the array. The mousePressed function is currently empty. The code editor also shows a preview of the browser window at 127.0.0.1:49545/index.html, which displays a 3D word cloud with various words like 'hey', 'what', 'do', 'informer', 'culture', 'nature', etc., as 3D blocks.

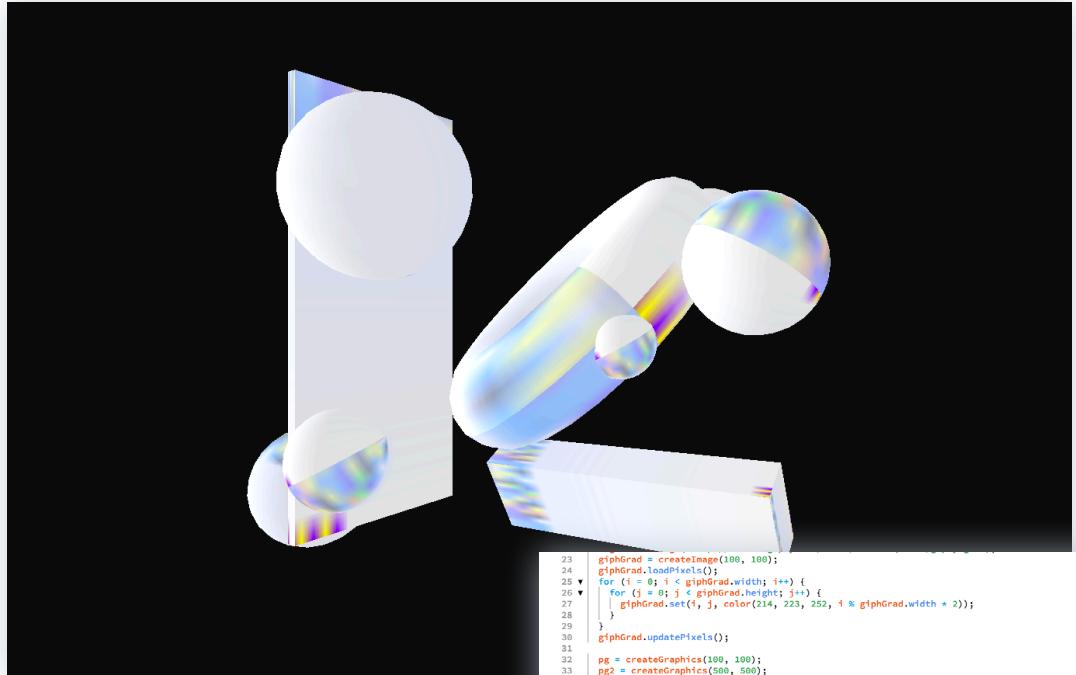
```
sketch.js (exchange letters copy 11) — Brackets
Working Files
sketch.js
index.html
exchange letter
index.html
libraries
sketch.js

1 function setup() {
2   createCanvas(600, 400, WEBGL);
3   input = createInput();
4   button = createButton("SUBMIT");
5   button.mousePressed(pressing);
6
7
8   console.log(words);
9
10 function draw() {
11   background(0);
12
13   /*for (var i = 0; i < bubbles.length; i++) {
14     bubbles[i].move();
15     bubbles[i].display();
16   }*/
17
18   for (var i = 0; i < words.length; i++) {
19     words[i].display();
20   }
21
22 }
23
24 function pressing() {
25   words.push(new Word(random(-width/2, width/2), random(-height/2, height
26     /2), random(10, 50), random(10, 50)));
27 }
28
29
30 function Word(x, y, w, h, d) {
31   this.x = x;
32   this.y = y;
33   this.w = w;
34   this.h = h;
35   this.d = d;
36
37   this.update();
38   this.show();
39 }
```

PROCESS

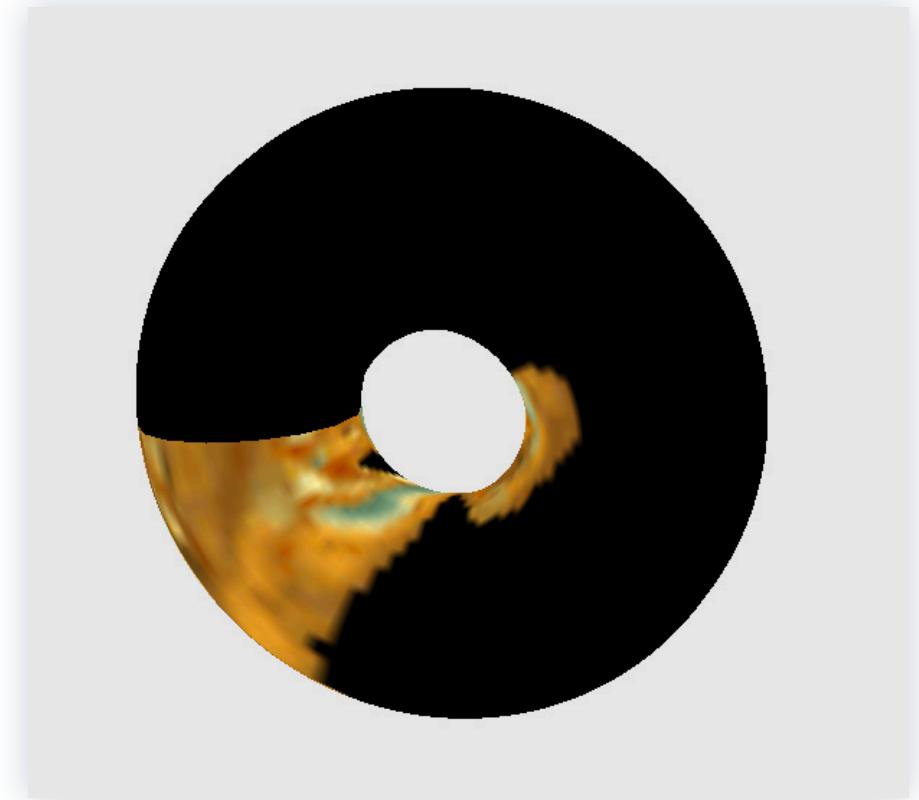


PROCESS



```
23 giphGrad = createImage(100, 100);
24 giphGrad.loadPixels();
25 for (i = 0; i < giphGrad.width; ++i) {
26   for (j = 0; j < giphGrad.height; ++j) {
27     | giphGrad.set(i, j, color(214, 223, 252, i % giphGrad.width * 2));
28   }
29 }
30 giphGrad.updatePixels();
31
32 pg = createGraphics(100, 100);
33 pg2 = createGraphics(500, 500);
34 input = createInput();
35 button = createButton("SUBMIT");
36 button.mousePressed(asKiphy);
37 }
38 // function gotData(giphy) { ... }
39 function askGiphy() {
40   var url = api + apiKey + input.value();
41   loadJSON(url, gotData);
42 }
43
44 function draw() {
45   ambientLight(250);
46   directionalLight(60, 60, 60, -1, 0, 0);
47   background(10);
48   orbitControl();
49   angleMode(DEGREES);
50   if (gifs.length <= 0) {
51     pg.background(243);
52     pg.imageMode(CENTER);
53     pg.image(img1, 0, 0, 1600, 1600);
54     pg.imageMode(CORNER);
55     pg.image(giphGrad, 0, 0);
56     giphyShapes(0, 0, -180, pg, pg, pg, pg, pg, pg, pg);
57   } else if (gifs.length > 0) {
58     pg.background(238);
59     pg.imageMode(CENTER);
60     pg.image(img1, 0, 0, 1600, 1600);
61     pg.imageMode(CORNER);
62     pg.image(giphGrad, 0, 0);
63     giphyShapes(0, 0, -180, pg, pg, pg, pg, pg, pg, pg);
64   }
65 }
66
67 function giphyShapes(x, y, z, giph1, giph2, giph3, giph4, giph5, giph6, giph7, giph8, giph9) { ... }
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
```

IMAGE REPSONSE TO INPUT AS
TEXTURE



PROCESS

INPUTS AS ARCHITECTURAL STRUCTURE



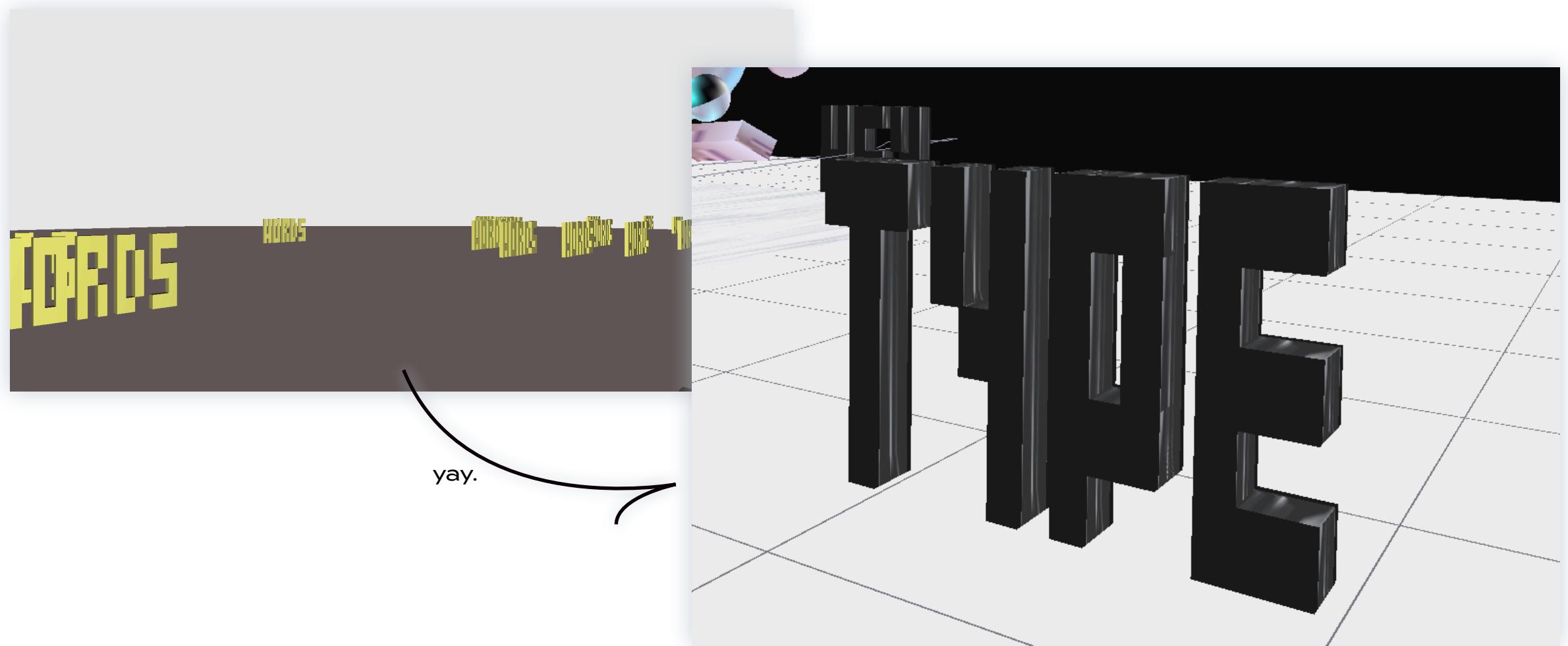
```
inputCollect.push(input.value()); //add inputs to an array for 3D text  
  
words.push(new InputsData(random(-boxSize/2.3, boxSize/2.3 - 400), random(0, 0), random(-boxSize/2.3, boxSize/2.3), inputCollect[inp  
l], rotations[floor(random(0, rotations.length))]));
```

```
//-----  
for (var j = 0; j < words.length; j++) {  
  
    var char = words[j].wording.split("");  
    for (var i = 0; i < char.length; i++) { //split all arrays into array of individual letters  
        if (words[j].x > -300 && words[j].x < 300 && words[j].z > -300 && words[j].z < 300) {  
            var wordX = words[j].x + 500;  
            var wordZ = words[j].z + 500;  
        } else {  
            var wordX = words[j].x;  
            var wordZ = words[j].z;  
        }  
    }  
}
```

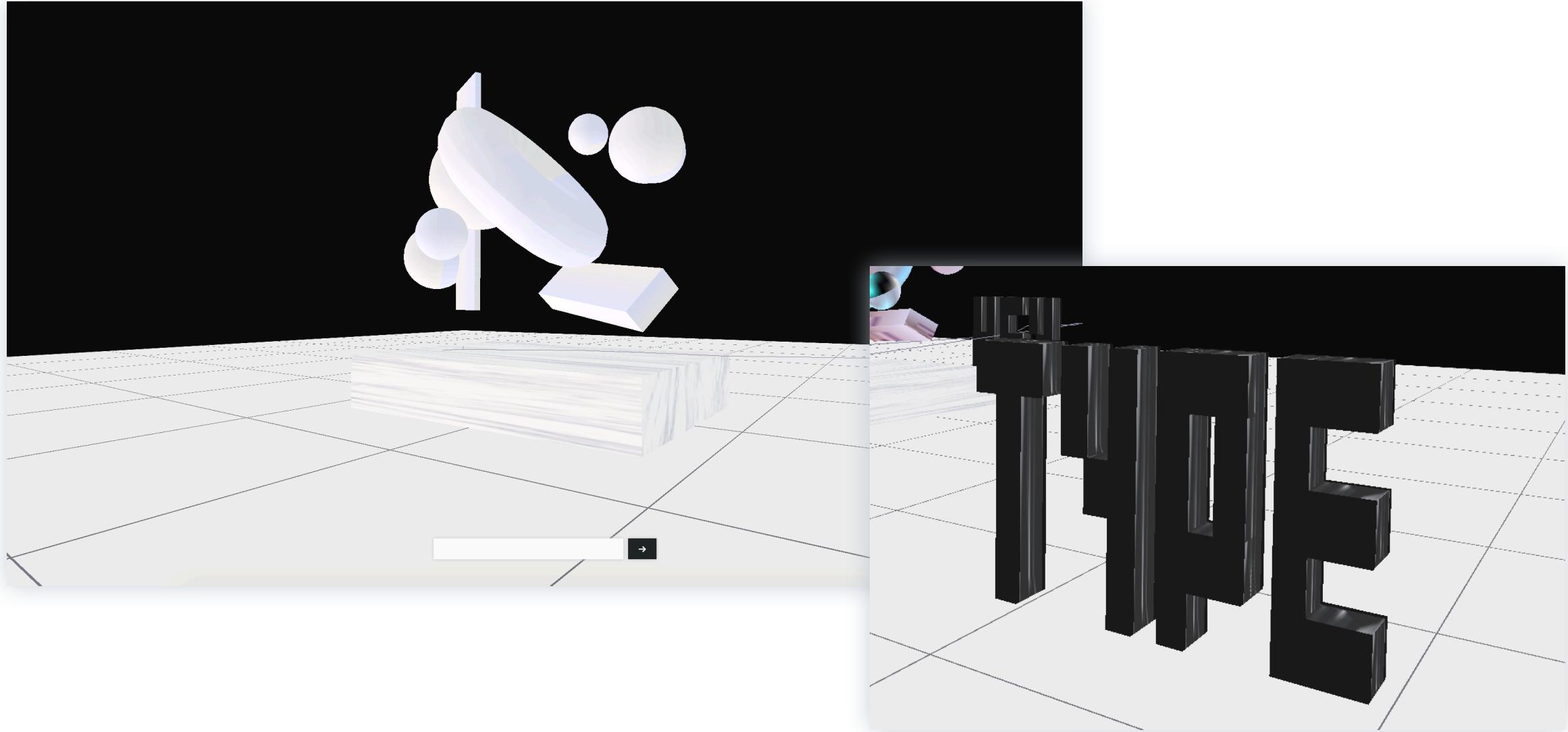
```
switch(char[i].toLowerCase()) { //change all letters to lowercase  
//switching text for designed 3D letters  
case 'a':  
    for (var lA = 0; lA < 40; lA++) {  
        if (words[j].wording.charAt(lA) == 'a') {  
            var xa = letterSpacing * lA;  
            drawA(wordX + xa, 0 + words[j].y + height/2 - heightOffset, 0 + wordZ, stretch, words[j].ystretch, letterScale, words[j].letterRo  
            marbleTextureB);  
        }  
    }  
    break;
```

PROCESS

SETTING THE TYPE



FINAL OUTCOME



FINAL OUTCOME

