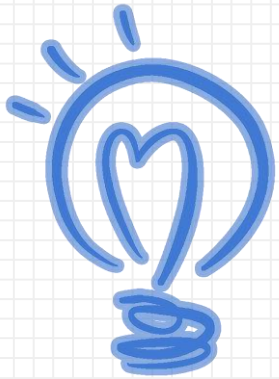


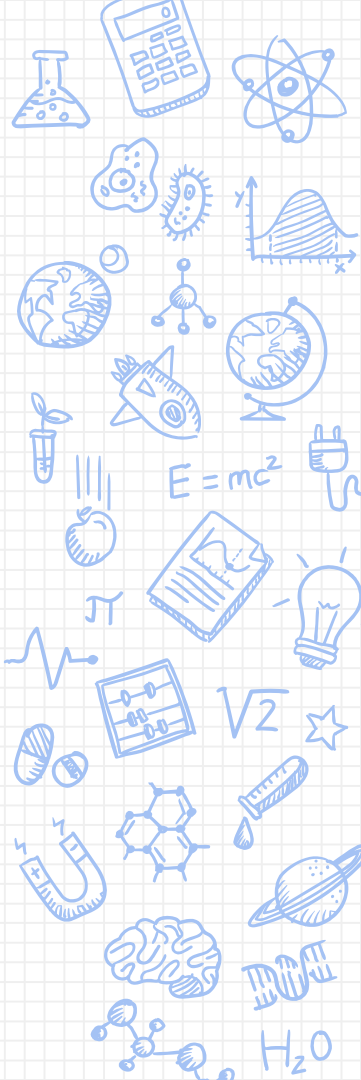
Naruhodo!

A laboratory on your desk!

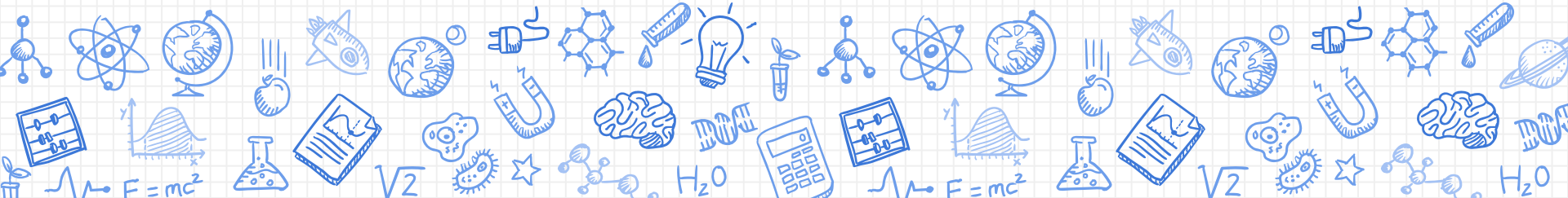
Designed & developed by Bowei Zhou

Outline

- ✗ What is Naruhodo?
- ✗ What can we do?
- ✗ How does it work?

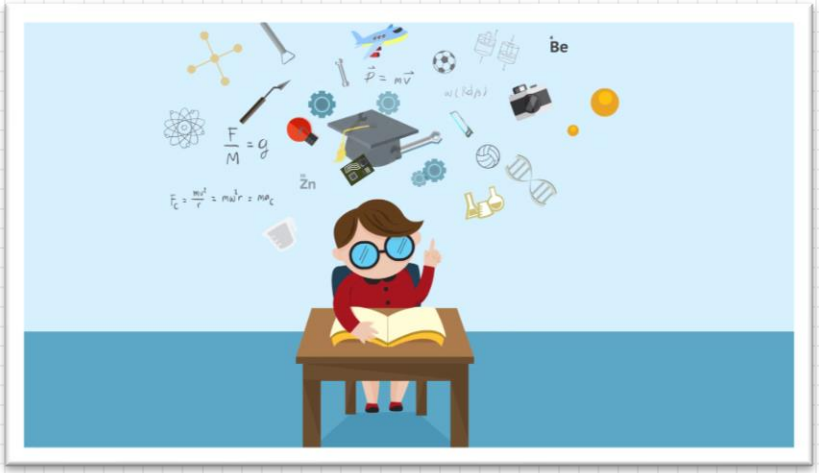


What is Naruhodo?



Inequality of education opportunity

The inequality of education opportunity is a serious problem. It is necessary for a government to give everyone a quality education.



A true story about me

In my school days, I enjoyed doing experiments at home…

$$2Cu + O_2 \triangleq 2CuO$$

A true story about me

In my school days, I enjoyed doing experiments at home…

$$CuO + 2CH_3COOH = H_2O + Cu(CH_3COO)_2$$

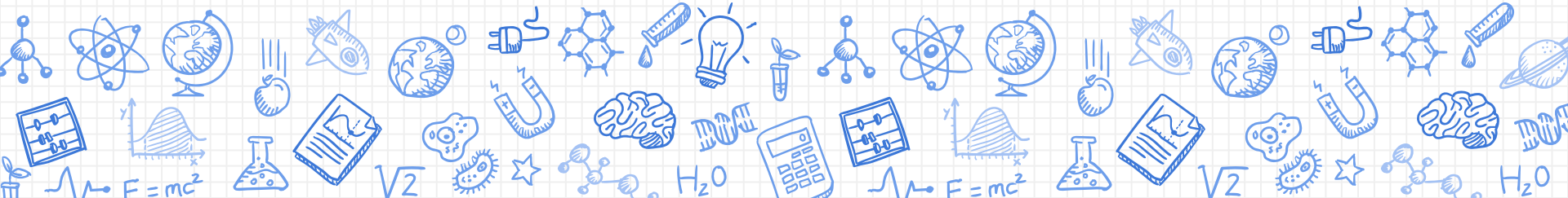
What is Naruhodo?

Naruhodo is a Japanese word, which means

"I see!" or "A-ha!"

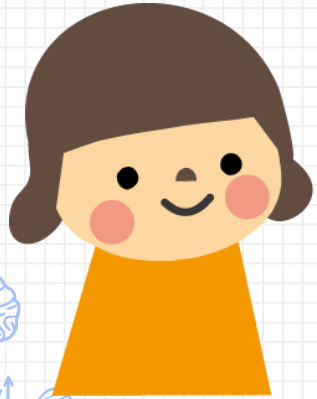
It is necessary to do experiments by oneself, in order to improve scientific literacy.

What can we do?



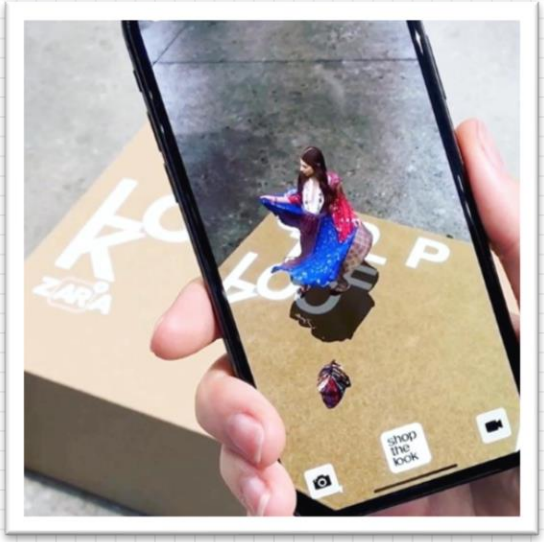
I want to do experiments at home!

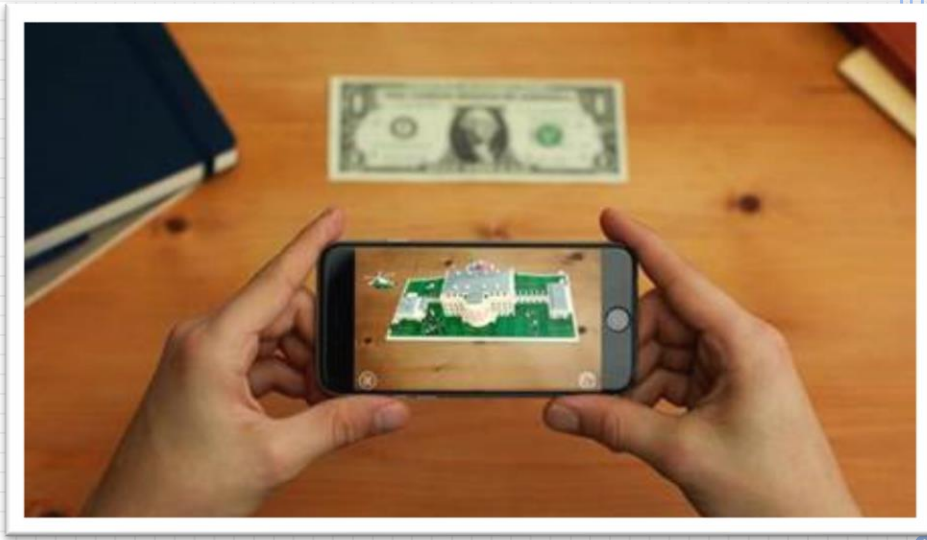
Let AR technology help you.



AR (Augmented Reality) technology

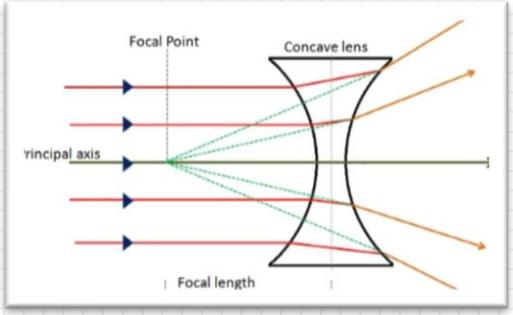
AR is a technology that allows the virtual world on the screen to combine and interact with real-world scenes.

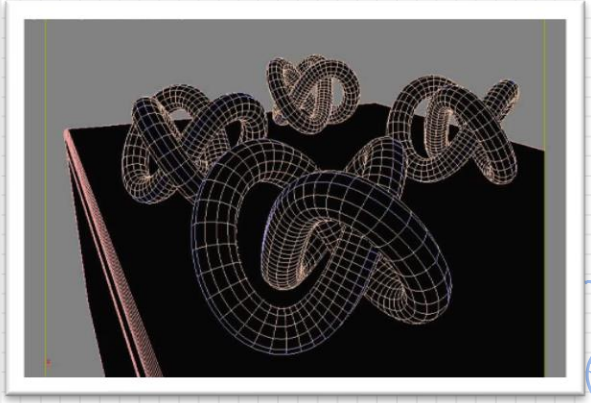




Naruhodo

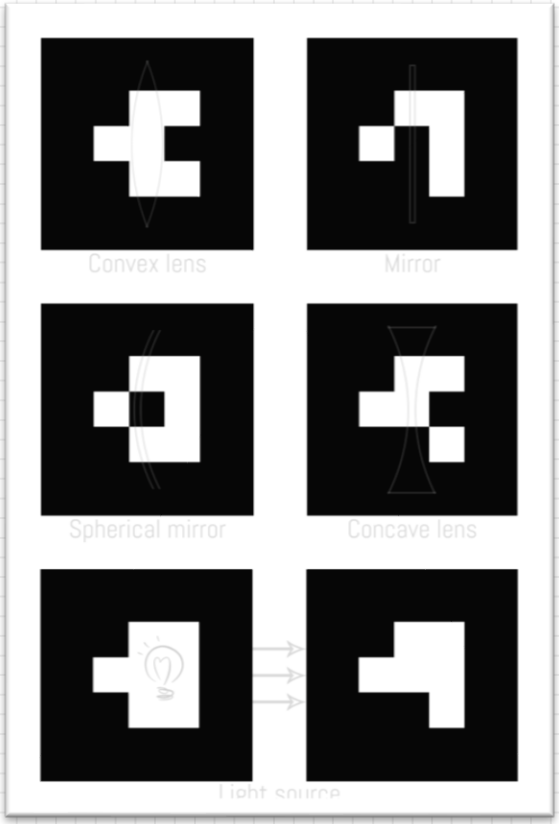
=

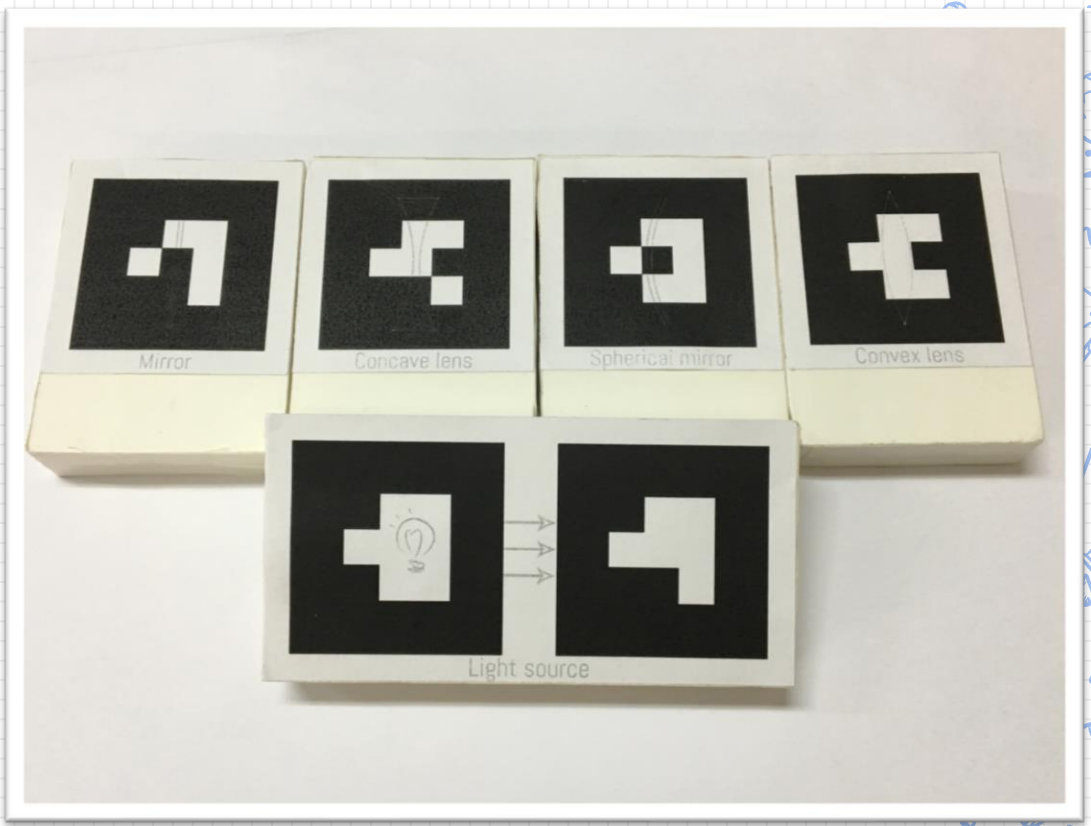




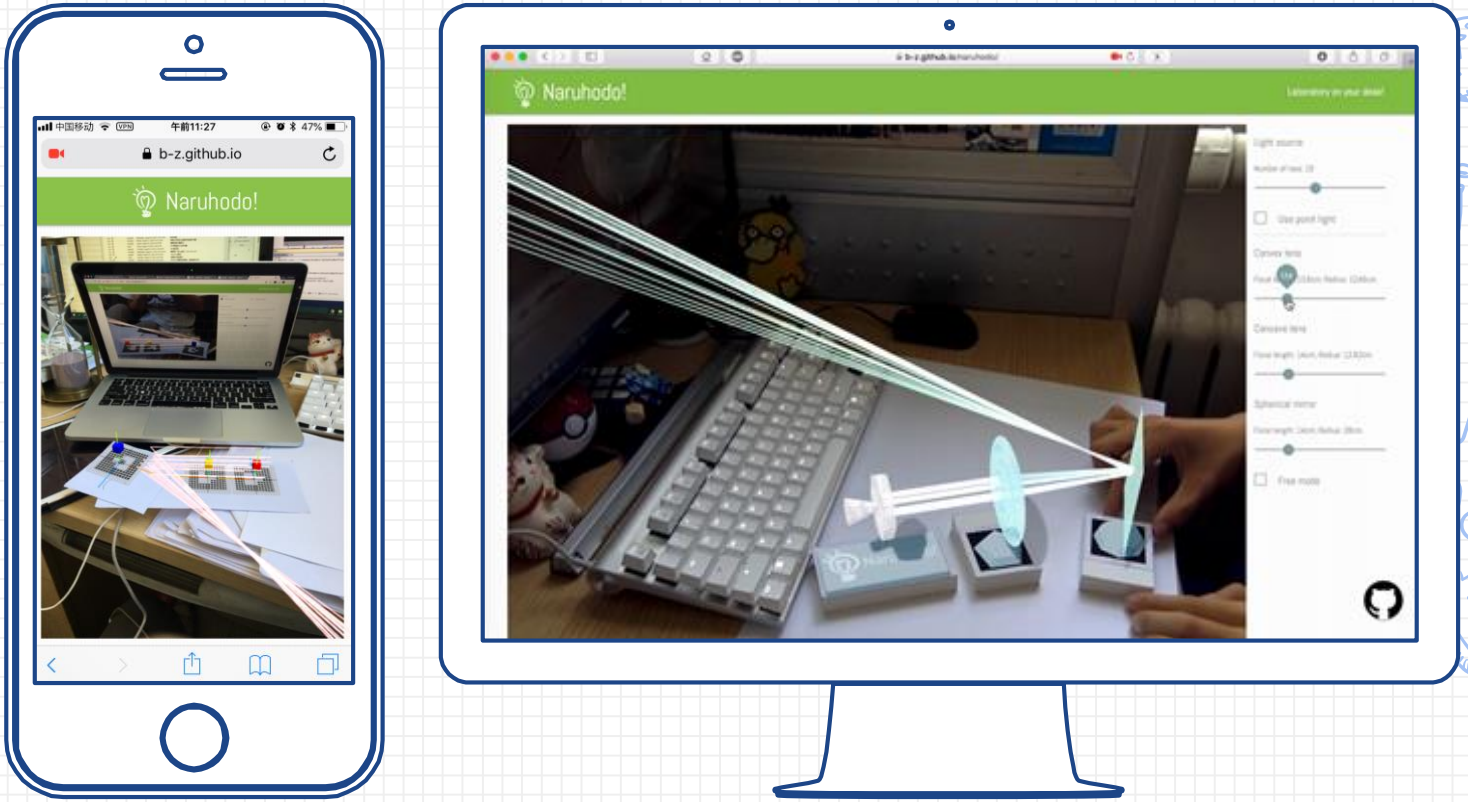
AR + Physics + 3D

Print the markers

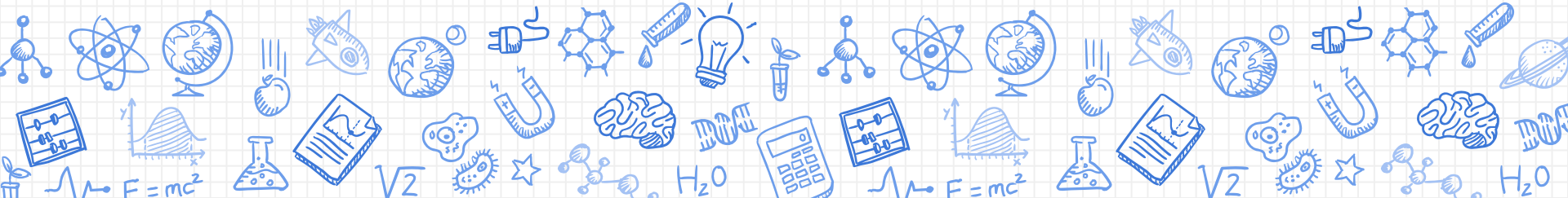




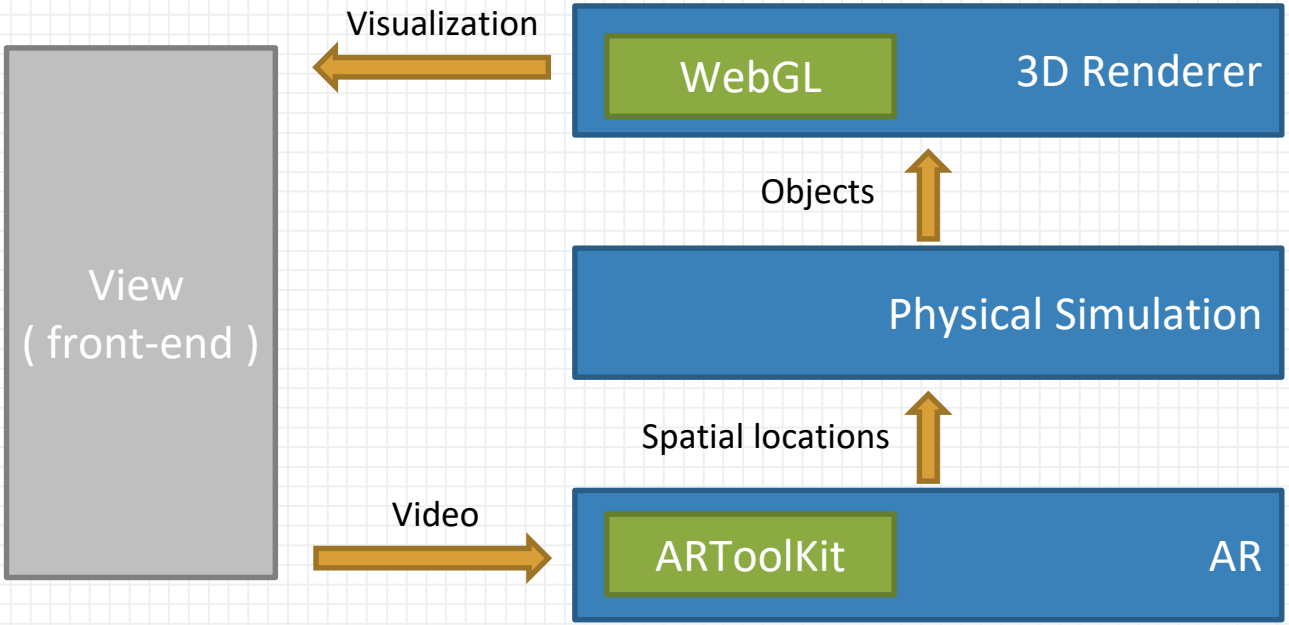
Place the markers on your desk



How does it work?



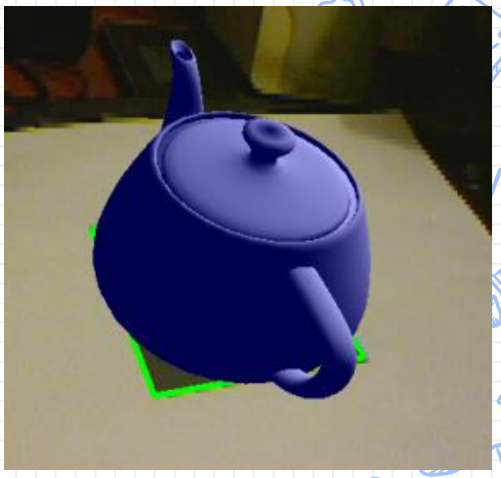
System structure



Layer-3: AR

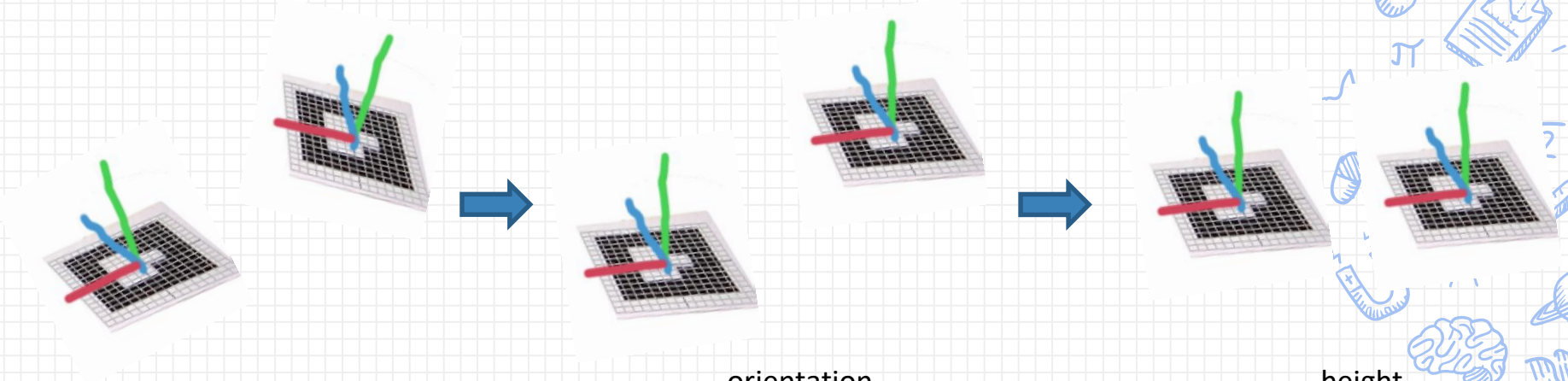
Core:

Video + ARToolKit → 3D Position and orientations of the markers



Layer-3: AR

Need to balance the height and orientation of the markers



orientation

height

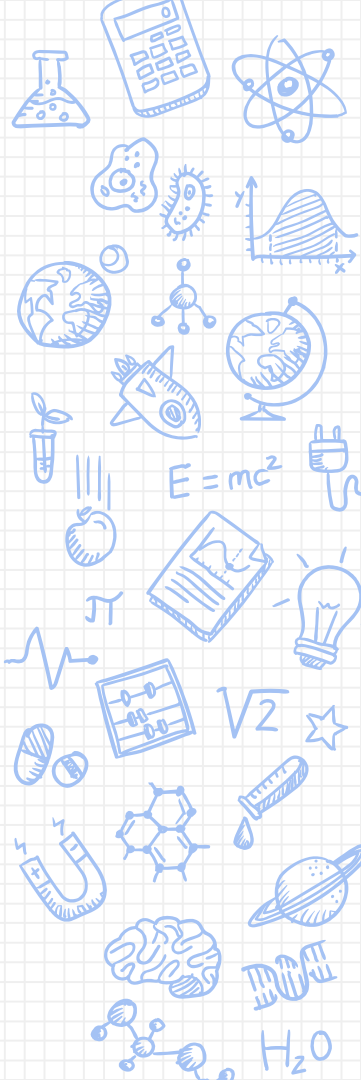
Layer-3: AR

Balance the orientations:

1. Calculate the world coordinate of (0, 1, 0) vector in each marker's world.
2. Calculate the mean orientation.
3. Rotate the markers' worlds.

Balance the heights:

1. Solve the inverse matrix of a matrix formed by orthogonal vectors of respective marker's world.
2. Use matrix multiplication to get the spatial height of each marker.
3. Balance their heights.



Layer-3: AR

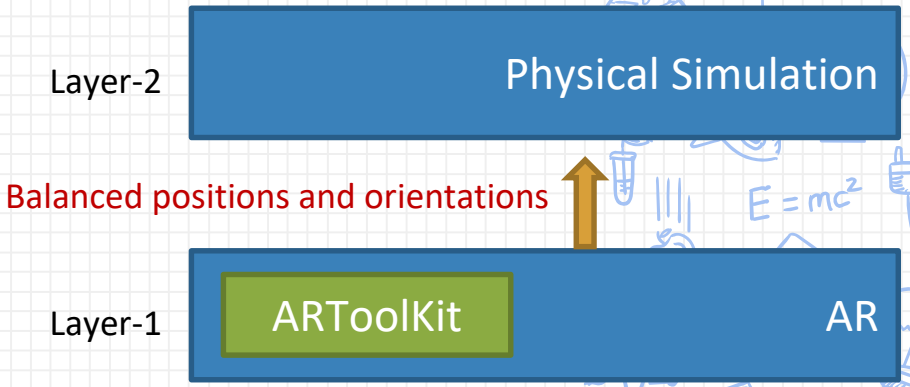
```
function adjustMarkers(markers, groups) {
    // return;
    var m = [];
    var g = [];
    for (var i = 0; i < markers.length; i++) {
        if (markers[i].visible) {
            m.push(markers[i]);
            g.push(groups[i]);
            groups[i].position.copy(markers[i].position);
            groups[i].quaternion.copy(markers[i].quaternion);
            groups[i].updateMatrixWorld(true);
        }
        groups[i].visible = markers[i].visible;
    }
    // if (markers[0].visible && !markers[1].visible) {
    if (!data.coplanar) return;
    // first, rotate to a same rotation
    if (!m.length) return;
    var dir = g.reduce(function(p, obj) {
        var v = new THREE.Vector3(0, 1, 0);
        v.applyMatrix4(obj.matrixWorld).sub(obj.position).normalize().add(p)
        return v;
    }, new THREE.Vector3(0, 0, 0));
    dir.normalize();
    g.forEach(function(obj) {
        var q = new THREE.Quaternion();
        var v = new THREE.Vector3(0, 1, 0);
        v.applyMatrix4(obj.matrixWorld).sub(obj.position).normalize();
        // q.setFromUnitVectors(v, dir).normalize();
        // obj.quaternion.normalize().multiply(q).normalize();
        // obj.quaternion.setFromUnitVectors(v, dir.clone().normalize());
        var axis = new THREE.Vector3();
        axis.crossVectors(v, dir);
        if (axis.length() < epsilon) return;
        axis.normalize();
        var angle = v.angleTo(dir);
```

```
    // then move to a same level
    var x = new THREE.Vector3(1, 0, 0);
    var y = new THREE.Vector3(0, 1, 0);
    var z = new THREE.Vector3(0, 0, 1);
    x.applyMatrix4(g[0].matrixWorld).sub(g[0].position);
    y.applyMatrix4(g[0].matrixWorld).sub(g[0].position);
    z.applyMatrix4(g[0].matrixWorld).sub(g[0].position);
    var a1 = x.x;
    var a2 = x.y;
    var a3 = x.z;
    var b1 = y.x;
    var b2 = y.y;
    var b3 = y.z;
    var c1 = z.x;
    var c2 = z.y;
    var c3 = z.z;
    var det = a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) + a3 * (b1 * c2 - c1 * b2);
    // if (Math.abs(det) < epsilon) return null;
    var px = new THREE.Vector3(b2 * c3 - c2 * b3, c1 * b3 - b1 * c3, b1 * c2 - c1 * b2);
    var py = new THREE.Vector3(c2 * a3 - a2 * c3, a1 * c3 - c1 * a3, c1 * a2 - a1 * c2);
    var pz = new THREE.Vector3(a2 * b3 - b2 * a3, b1 * a3 - a1 * b3, a1 * b2 - b1 * a2);

    var kys = [];
    g.forEach(function(obj) {
        var pos = obj.position;
        var ky = py.dot(pos) / det;
        // kx * x + ky * y + kz * z
        // should average the ky;
        kys.push(ky);
    });
    var avg = 0;
    for (var k of kys) avg += k;
    avg /= m.length;

    for (var i = 0; i < g.length; i++) {
        var obj = g[i];
        var pos = obj.position;
```

Layer-2: Physical simulation

Physical simulation

1. Refraction and reflection
2. Test intersections with
   - ✗ Sphere
   - ✗ Plane



$$(x_0, y_0, z_0) + t(a, b, c), t > \varepsilon, (x_0 + ta - x_b)^2 + (y_0 + tb - y_b)^2 + (z_0 + tc - z_b)^2 = R^2,$$

$$(a^2 + b^2 + c^2)t^2 + 2(a(x_0 - x_b) + b(y_0 - y_b) + c(z_0 - z_b))t + ((x_0 - x_b)^2 + (y_0 - y_b)^2 + (z_0 - z_b)^2 - R^2) = 0$$

$$src + t \cdot dir = c + a \cdot x + b \cdot y \quad \text{s.t.} -1 \leq a \leq 1, \quad -1 \leq b \leq 1, \quad t > \varepsilon$$

Layer-2: Physical simulation

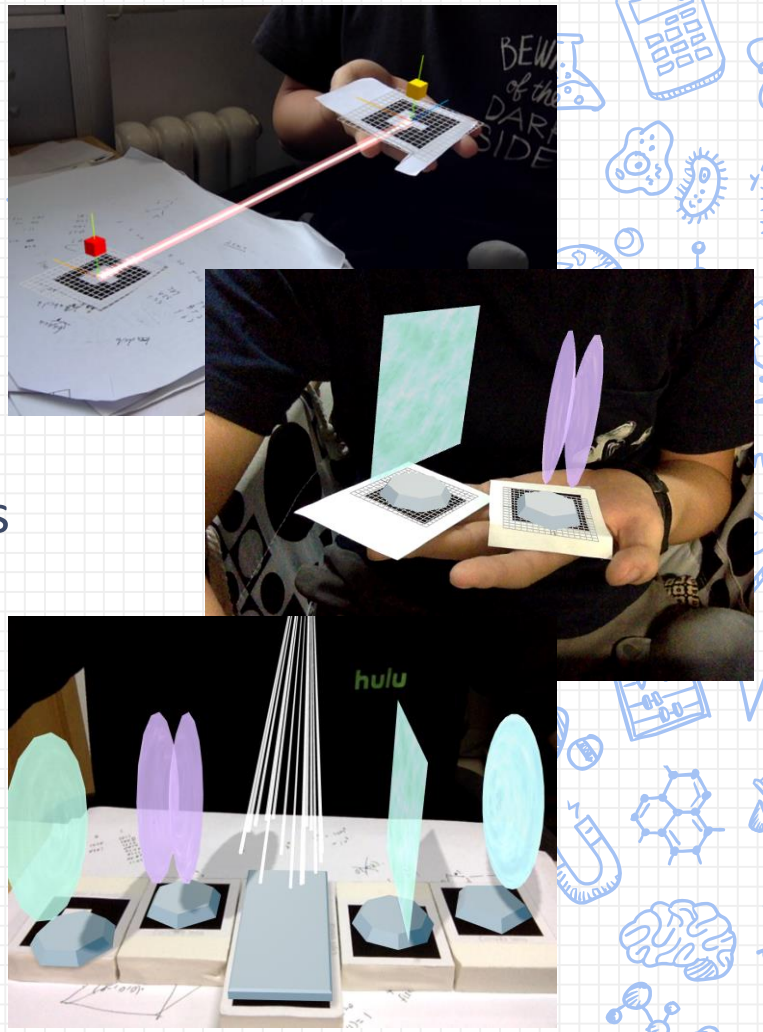
```
function testIntersectionToSphere(src, dir, center, R) {
    // 射线到球求交点
    var a = dir.dot(dir);
    var sub = src.clone().sub(center);
    var b = 2 * dir.dot(sub);
    var c = sub.dot(sub) - R * R;

    var delta = b * b - 4 * a * c;
    var result = [];
    if (delta >= 0) {
        delta = Math.sqrt(delta);
        var t1 = (-b - delta) / a / 2;
        var t2 = (-b + delta) / a / 2;
        if (t1 > epsilon) result.push(src.clone().add(dir.clone().mu
        if (t2 > epsilon) result.push(src.clone().add(dir.clone().mu
    }
    return result;
}
```

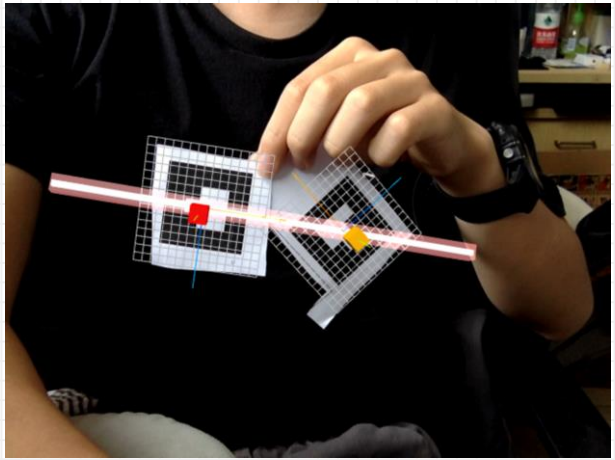
```
function testIntersectionToPlanePart(src, dir, c, x, y, norm) {
    var a1 = x.x;
    var a2 = x.y;
    var a3 = x.z;
    var b1 = y.x;
    var b2 = y.y;
    var b3 = y.z;
    var c1 = -dir.x;
    var c2 = -dir.y;
    var c3 = -dir.z;
    var det = a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) + a3 * (b1 * c2 - c1 * b2);
    if (Math.abs(det) < epsilon) return null;
    var pa = new THREE.Vector3(b2 * c3 - c2 * b3, c1 * b3 - b1 * c3, b1 * c2 - c1 * b2);
    var pb = new THREE.Vector3(c2 * a3 - a2 * c3, a1 * c3 - c1 * a3, c1 * a2 - a1 * c2);
    var pt = new THREE.Vector3(a2 * b3 - b2 * a3, b1 * a3 - a1 * b3, a1 * b2 - b1 * a2);
    var s = src.clone().sub(c);
    var a = pa.dot(s) / det;
    var b = pb.dot(s) / det;
    var t = pt.dot(s) / det;
    if (t < epsilon) return null;
    if (a > 1 || b > 1 || a < -1 || b < -1) return null;
    // console.log(t);
    var pos = c.clone().add(x.clone().multiplyScalar(a)).add(y.clone().multiplyScalar(b));
    return {
        pos: pos, //src.clone().add(dir.clone().multiplyScalar(t)),
        norm: norm
    }
```

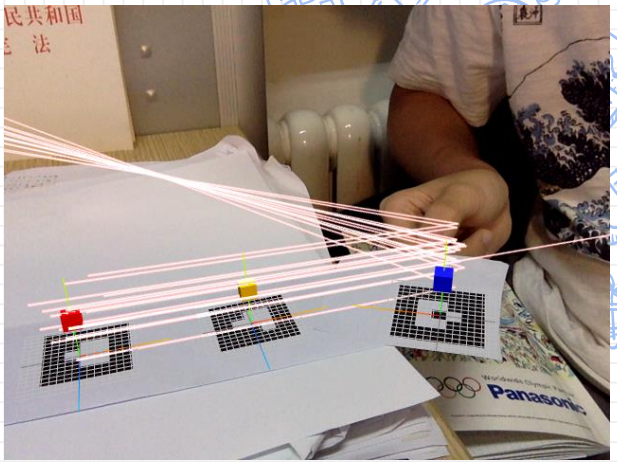
Layer-1: 3D renderer

In Layer-1, I should:

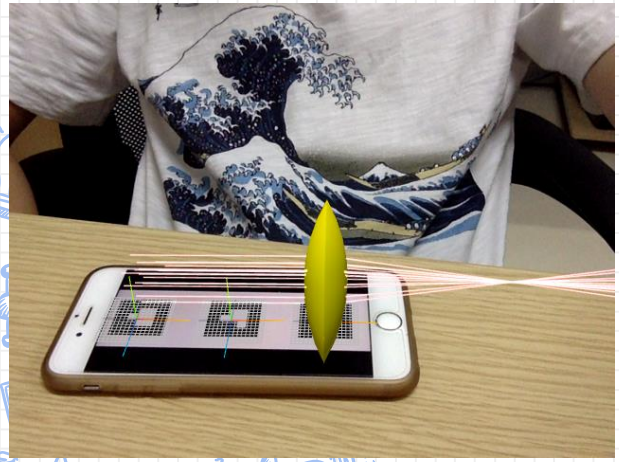
- ✗ Design & draw the objects
- ✗ Design the material of the objects
- ✗ Add some light sources
- ✗ Draw the shadows

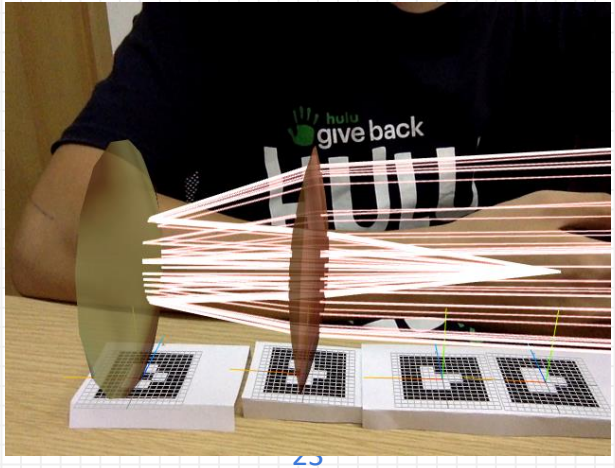


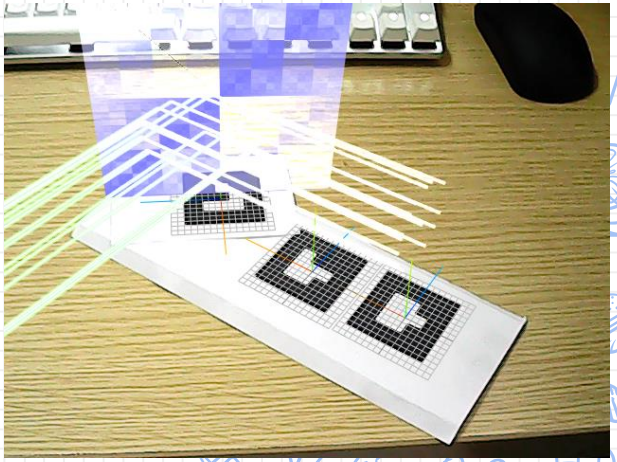




The two weeks of development







THANKS!

Designed & developed by Bowei Zhou

https://b-z.github.io/naruhodo