Term Project Technical Note

Computer Graphics

| 2021.11.21 |

201533631 김도균
201835439 김태희
201835455 박윤재
201835527 조민영

Table of Content

Brick Craft

1. Flutter Framework 3p
2. Progressive Web App 7p
3. Exporting 3D Object 11p
4. Voxel Environment 16p
5. Ray Cast Picking 26p
6. Camera Movement 29p
7. Appendix 38p

Ⅰ. Flutter Framework

I . Flutter Framework

Flutter is the cross-platform framework based on Dart language.

Flutter is developed by Google, and latest version is support various platform such as Android, iOS, Windows, Linux, even macOS.

Flutter also support web front-end page build to make SPA(Single Page Application).

This web building features is very powerful to make web front-end page easier such as Vue.js and React.js

Flutter Framework

Flutter-powered Google apps shipping now include:

Google One · G Pay · Nest Hub · STADIA

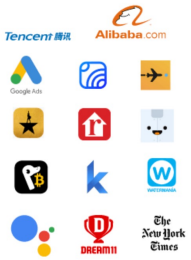
Family Link · Google Ads · Google Shopping

Google Analytics · Cloud Search · Google AdMob

Toyota + Flutter

Toyota is taking a completely new approach to building next-gen vehicle infotainment systems powered by Flutter.

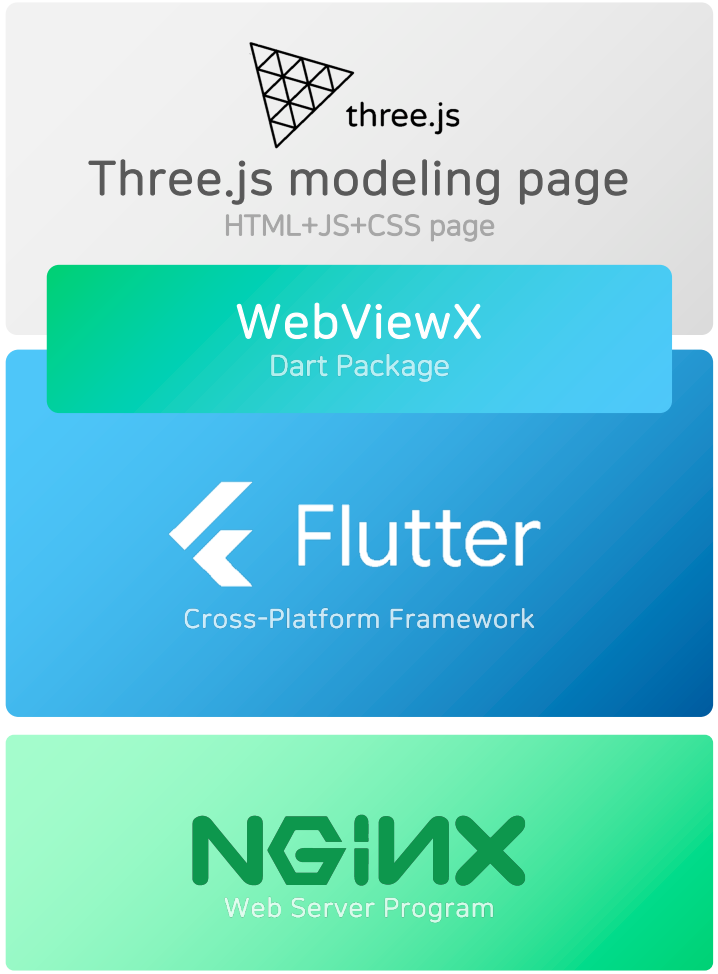
TOP 32 Famous Mobile Apps Developed Using FLUTTER



Flutter's future is very bright. Nowadays, some of Google's major services are made by Flutter such as Stadia(Real-time cloud computing system for gamming), Google Pay, and services related with Google's Advertisement.

Flutter framework also built-in to vehicle's infotainment system of Toyota and lot of other services such as The New York Times, use Flutter framework for service their app.

I . Flutter Framework



This is the BrickCraft's layered architexture. We made the service as a web app. For this, we needed to server program like NGINX to provide web page assets through the internet.

We need to bond flutter and modeling page made by three.js. To this, we used WebViewX package provided by dart package repository(pub.dev).

II . Progressive Web App

II . Progressive Web App

PWA (Progressive Web App) is the type of web app that has advantage of native app and web app. This web page can be installed and can use sensors and other components what web client has such as camera and vibration motors through the PWA app.

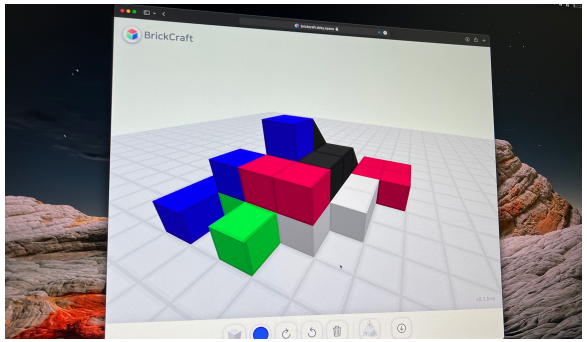
Reference: https://www.prestashop.com/forums/topic/1025852-module-pwa-mobile-app-launch-a-pwa-mobile-app-for-your-prestashop-store/

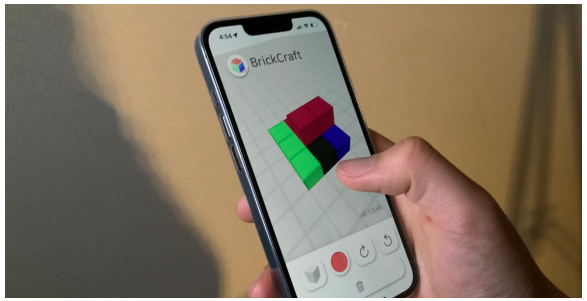
Ⅱ. Progressive Web App

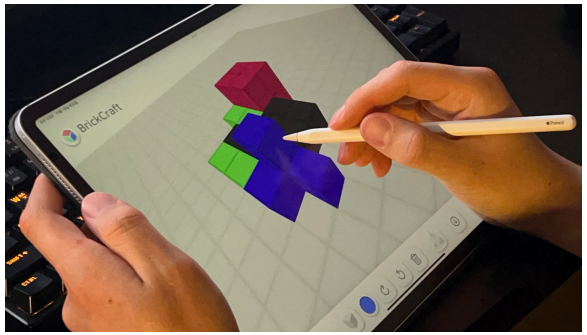
PWA technique was introduced by Google I/O 2016 and Flutter was introduced by same event at the following year.

Flutter web support PWA and this can make PWA app easier than develop PWA in vanilla environment.

II . PWA Support







PWA App can be installed various environments, and this works like native app.

We tested BrickCraft's PWA techniques in MacBook, iPhone, iPad and this works well.

Ⅲ. Exporting 3D Object

III. Exporting 3D Object

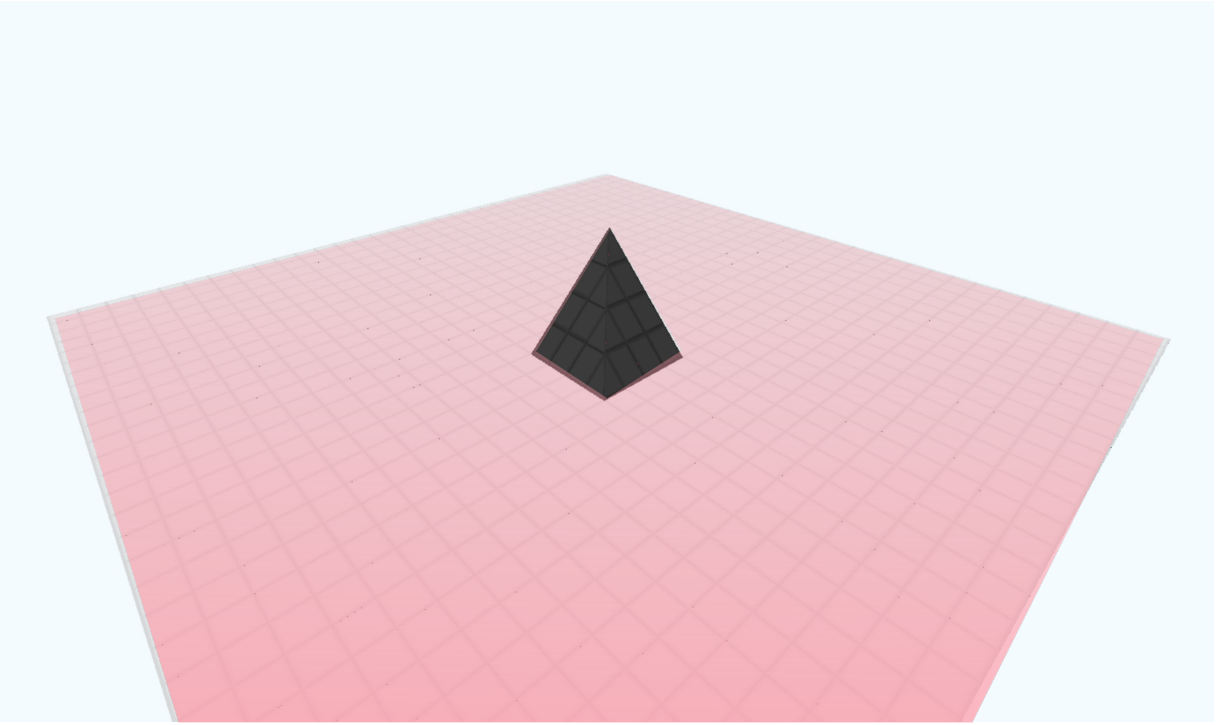
```
import {OBJExporter} from './examples/jsm/exporters/OBJExporter.js';

function objExportFromScene(targetScene) {
    let objCode = (new OBJExporter()).parse(targetScene);
    return objCode
}
```

Three.js support to exporting obj file through the OBJExporter.js.
This function make scene instance to 3D data file (object format) what is printable.

III. Exporting 3D Object

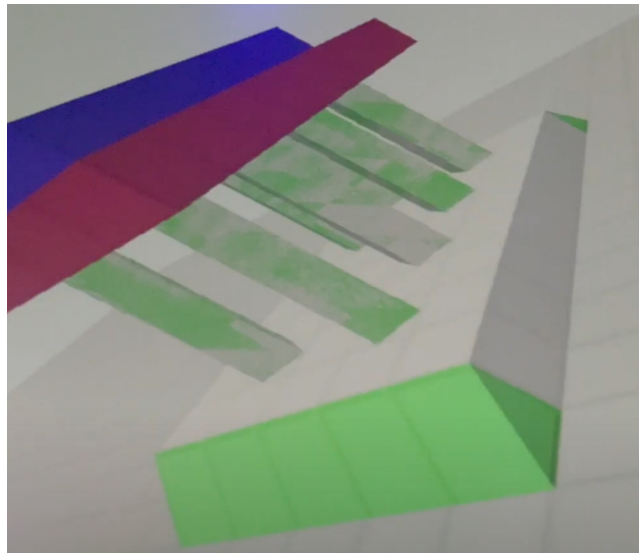
Manipulating bottom plate what is added in scene instance
for extracting scene to OBJ file.

```
// Create(add) pannel to scene
function createBottomPlate() {
    for(let i = 0; i < paletteSize; i++)
        for(let j = 0; j < paletteSize; j++)
            palette.setBrickState(i, 0, j, 1);
}


// Delete(remove) pannel to scene
function removeBottomPlate() {
    for(let i = 0; i < paletteSize; i++)
        for(let j = 0; j < paletteSize; j++)
            palette.setBrickState(i, 0, j, 0);
}
```



Ⅲ. Exporting 3D Object

Process of printing 3D object file

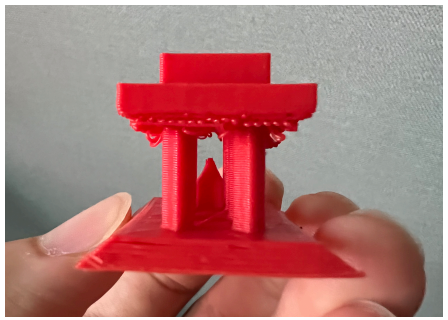


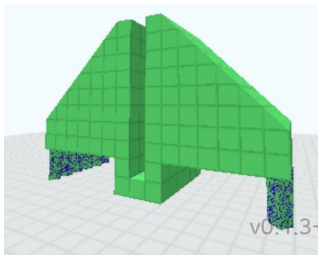
BrickCraft Model

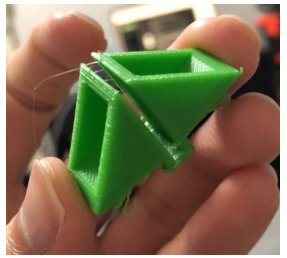
obj file

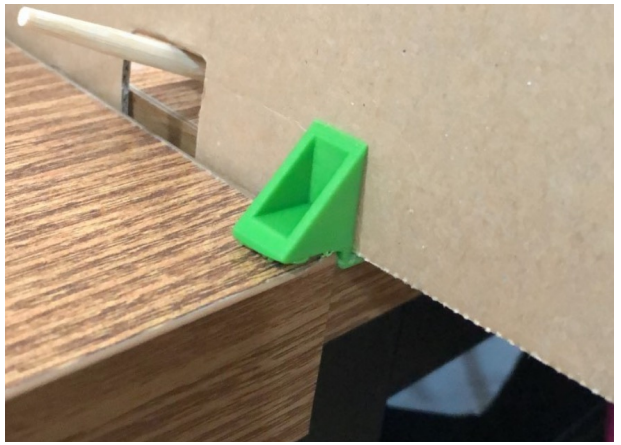
Printed object

Ⅲ. Exporting 3D Object









The size of bricks is 1x1x1mm. This can make the printed model to use practical area such as a part of furniture or utilities in living room like hanger.

Ⅵ. Voxel Environment

Ⅳ. Voxel Environment

Set shape and color variables that selected by the user.

```
// Set functions to export globally(globalThis)
// for Flutter framework
let control_create = () => {
    deleteMode = false;
    changeBrick()
}

let control_delete = () => {
    deleteMode = true;
    currentBrickShape = 0;
}

let control_selectColor = (index) => {
    selectedColor = index
    changeBrick()
}

let control_selectShape = (index) => {
    selectedShape = index
    changeBrick()
}
```

deleteMode = false

deleteMode = true

5 different shapes

5 different colors

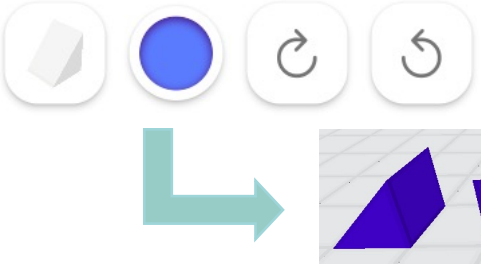
and 2 additional texture

Ⅳ. Voxel Environment

ChangeBrick() function calculates all possible numbers depending on the shape and color.

```
// Set functions to export globally(globalThis)
// for Flutter framework
let control_create = () => {
    deleteMode = false;
    changeBrick()
}

let control_delete = () => {
    deleteMode = true;
    currentBrickShape = 0;
}

let control_selectColor = (index) => {
    selectedColor = index
    changeBrick()
}

let control_selectShape = (index) => {
    selectedShape = index
    changeBrick()
}
```

```
// Change block by selectecd value (selectedColor, selectedShape)
function changeBrick() {
    selectedColor = Number.parseInt(selectedColor)
    selectedShape = Number.parseInt(selectedShape)

    currentBrickShape = selectedColor + (selectedShape - 1) * numberOfTexture
    currentBrickShape += 1
}
```



This button rotates the shape.
The shape of the shape to be made depends on the direction of the shape, so we implemented like this.

Ⅳ. Voxel Environment

```
// Add listener for mouse events (Place / Remove bricks)
canvas.addEventListener('pointerdown', (event) => {
    event.preventDefault();

    // Right click / delete mode to delete
    if (event.button == 2 || deleteMode)
        currentBrickShape = 0;
    // Left click to create
    else if(event.button == 0)
        changeBrick()

    // Check mouse was moved between mousedown and mouseup event
    // Record mouse position
    mousedownCoordinate = event.clientX + event.clientY
    mousedownCoordinate = mousedownCoordinate * mousedownCoordinate
});
```

```
window.addEventListener('pointerup', (event) => {
    // Check mouse was moved between mousedown and mouseup event
    let mouseupCoordinate = event.clientX + event.clientY
    mouseupCoordinate = mouseupCoordinate * mouseupCoordinate
    // Tolerate error to ignore a little bit moved
    const error = 2000
    if(mouseupCoordinate + error >= mousedownCoordinate && mousedownCoordinate >= mouseupCoordinate - error)
        placeBrick(event);
});
```

Divided into 'mousedown' & 'mouseup'

If the difference exceeds a certain number,
it is judged that the mouse moved while clicking.

If we moved after pressing the mouse, we had to judge with the intention of turning the camera, so we implemented it separately when the mouse was pressed and when it was implemented. Therefore, the action of adding and removing blocks was implemented only when the mouse was pressed and not moved.

Reference: https://threejsfundamentals.org/threejs/lessons/threejs-voxel-geometry.html

Ⅳ. Voxel Environment

```
// Place brick with ray cast picking
function placeBrick(event) {

    const rect = canvas.getBoundingClientRect();
    const pos = {
        x: (event.clientX - rect.left) * canvas.width  / rect.width,
        y: (event.clientY - rect.top ) * canvas.height / rect.height,
    }

    const x = (pos.x / canvas.width ) *  2 - 1;
    const y = (pos.y / canvas.height) * -2 + 1;  // flip Y

    const start = new THREE.Vector3();
    const end = new THREE.Vector3();
    start.setFromMatrixPosition(camera.matrixWorld);
    end.set(x, y, 1).unproject(camera);

    const intersection = palette.rayCastPicking(start, end); //rayCastPicking
    if (intersection) {

        const pos = intersection.position.map((v, ndx) => {
            return v + intersection.normal[ndx] * (currentBrickShape > 0 ? 0.5 : -0.5);
        });

        // Prevent deleting bottom of palette
        if(pos[1] > 1){
            palette.setBrickState(Math.floor(pos[0]), Math.floor(pos[1]), Math.floor(pos[2]), currentBrickShape);
            updateGeometry(Math.floor(pos[0]), Math.floor(pos[1]), Math.floor(pos[2]));
            requestRender();
        }
    }
}
```

```
// Add block to palette
// setBrickState( x, y, z, textureIdx )
setBrickState(x, y, z, v) {
    this.cell[x][y][z] = v
}
```

Canvas -> Clip Space
→ Change the coordinates of the Canvas clicked by the user to the coordinates of the Clip Space.

Clip Space -> rayCastPicking() -> setBrickState()
→ Block is installed with picking implemented with ray cast technology, rayCastPicking().

setBrickState() -> updateGeometry()
→ Block displayed on the screen using updateGeometry()

Ⅳ. Voxel Environment

```
        if(pos[1] > 1){
            palette.setBrickState(Math.floor(pos[0]), Math.floor(pos[1]), Math.floor(pos[2]), currentBrickShape);
            updateGeometry(Math.floor(pos[0]), Math.floor(pos[1]), Math.floor(pos[2]));
            requestRender();
        }
```

```
let mesh;

function updateGeometry(x, y, z) {
    const cellX = Math.floor(x / paletteSize);
    const cellY = Math.floor(y / paletteSize);
    const cellZ = Math.floor(z / paletteSize);

    const geometry = mesh ? mesh.geometry : new THREE.BufferGeometry();

    const {positions, normals, uvs, indices} = palette.generateGeometryData();


    const positionNumComponents = 3;
    geometry.setAttribute('position', new THREE.BufferAttribute(new Float32Array(positions), positionNumComponents));
    const normalNumComponents = 3;
    geometry.setAttribute('normal', new THREE.BufferAttribute(new Float32Array(normals), normalNumComponents));
    const uvNumComponents = 2;
    geometry.setAttribute('uv', new THREE.BufferAttribute(new Float32Array(uvs), uvNumComponents));
    geometry.setIndex(indices);
    geometry.computeBoundingSphere();


    if (!mesh) {
        mesh = new THREE.Mesh(geometry, material);
        scene.add(mesh);
        mesh.position.set(cellX * paletteSize, cellY * paletteSize, cellZ * paletteSize);
    }
}
```

Get the information of shape, color, and direction set by the user.

The block is added to the scene using mesh data information.

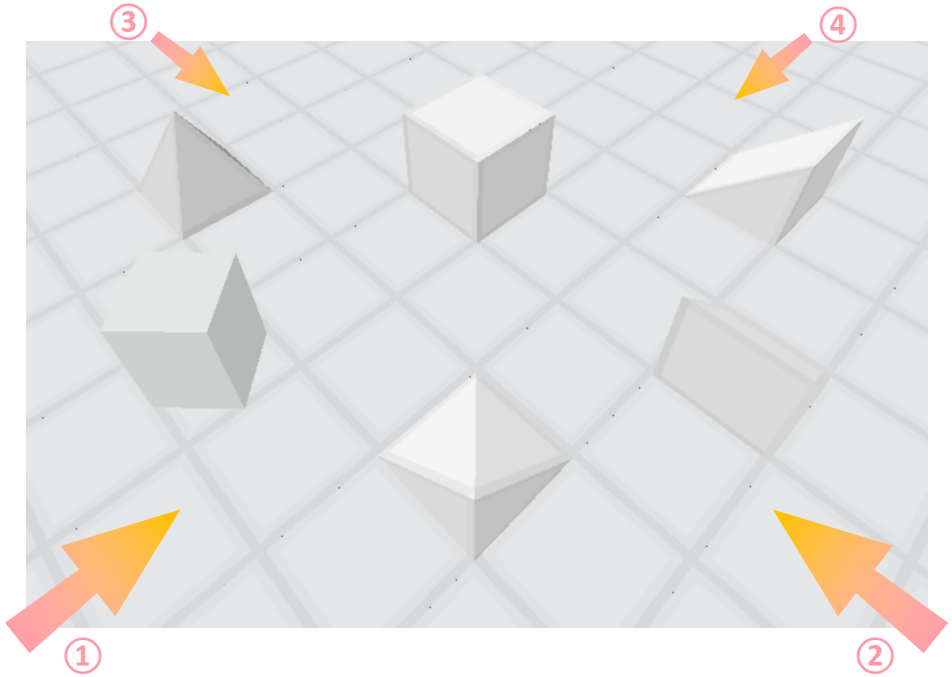
Ⅳ. Voxel Environment

Light from four directions

```
// Light
addLight(0, 10,  -30, 0.9); ①
addLight(-30, 10,  0, 0.8); ②

addLight(30, 10, 0, 0.7);   ③
addLight(0, 10,  30, 0.6);  ④
```

```
// Lighting
function addLight(x, y, z, intensity) {
    const color = 0xFFFFFF;
    const light = new THREE.DirectionalLight(color, intensity);
    light.position.set(x, y, z);
    scene.add(light);
}
```

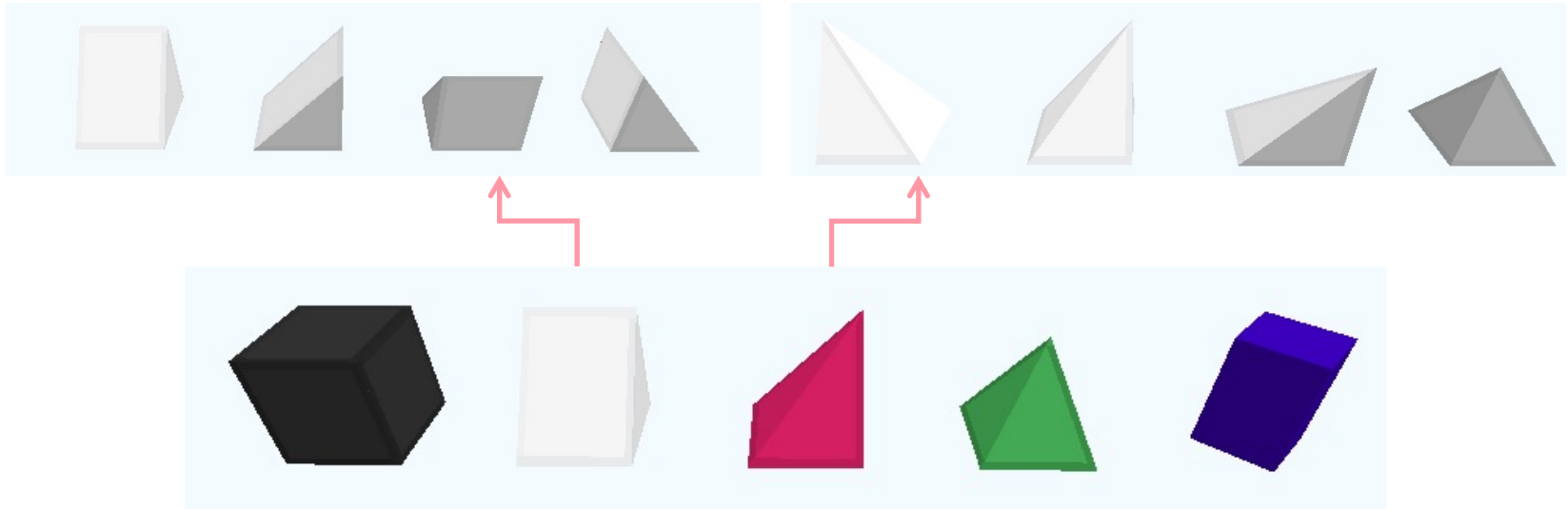
Brightness difference for each side



Ⅳ. Voxel Environment - Texture

Load Texture through **Three.TextureLoader()**
and use **Three.mesh** to **texture the blocks**

Ⅳ. Voxel Environment - Texture

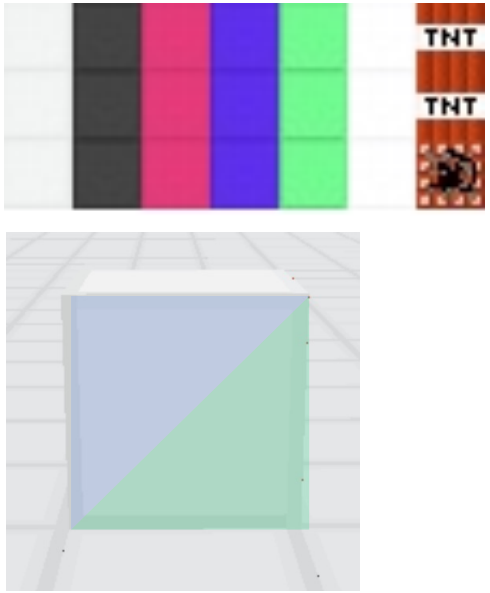


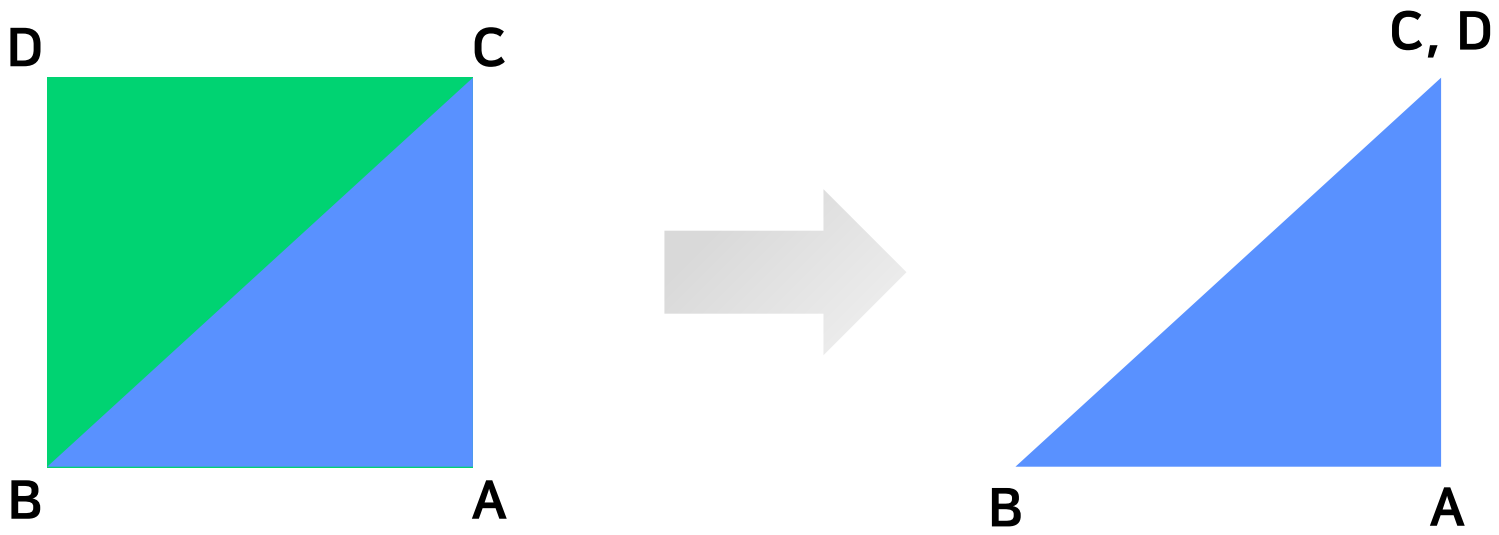
Brick State	facesCube [1]	facesHypotenuse [2~5]	facesTriangle [6~9]	facesPyramid [10]	facesColumn [11]

There are 11 cases in which texturing is required, including rotating blocks. In each case, a texture is applied to the block.
The data about the block loads the blocks defined in object.js.

Ⅳ. Voxel Environment - Texture

When a rectangle texture is loaded and the four vertices are A, B, C, D, the positions of C and D are overlapped to create a side with four coordinates with two coordinates overlapping each other.





V. Ray Cast Picking

V. Ray Cast Picking

```
// Picking with using ray cast
rayCastPicking(start, end) {

    let dx = end.x - start.x;
    let dy = end.y - start.y;
    let dz = end.z - start.z;
    const lenSq = dx * dx + dy * dy + dz * dz;
    const len = Math.sqrt(lenSq);

    dx /= len;
    dy /= len;
    dz /= len;
```

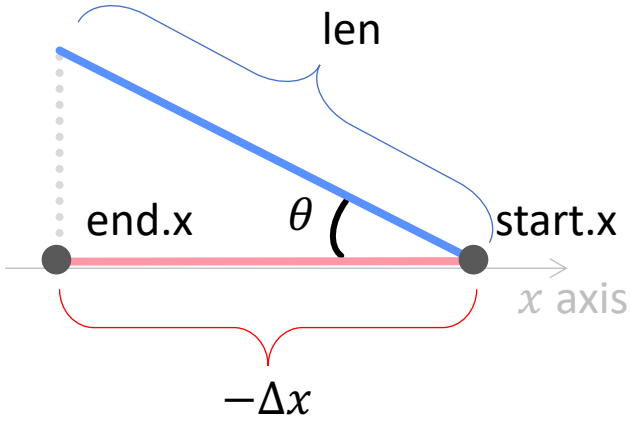
```
const stepX = (dx > 0) ? 1 : -1;
const stepY = (dy > 0) ? 1 : -1;
const stepZ = (dz > 0) ? 1 : -1;
```

```
let t = 0.0;
let ix = Math.floor(start.x);
let iy = Math.floor(start.y);
let iz = Math.floor(start.z);
```

$$dx = \frac{end.x - start.x}{\sqrt{dx^2 + dy^2 + dz^2}}$$

$$dy = \frac{end.y - start.y}{\sqrt{dx^2 + dy^2 + dz^2}}$$

$$dz = \frac{end.z - start.z}{\sqrt{dx^2 + dy^2 + dz^2}}$$



When ray cast picking is called with the start and end parameters in Placebrick, subtract each x,y,z value of end and start, and put it in dx, dy, dz.
And then square each of the dx, dy, and dz and put the square root value in len.
As a result, len represents the distance between two points on 3d because $\sqrt{dx^2 + dy^2 + dz^2}$

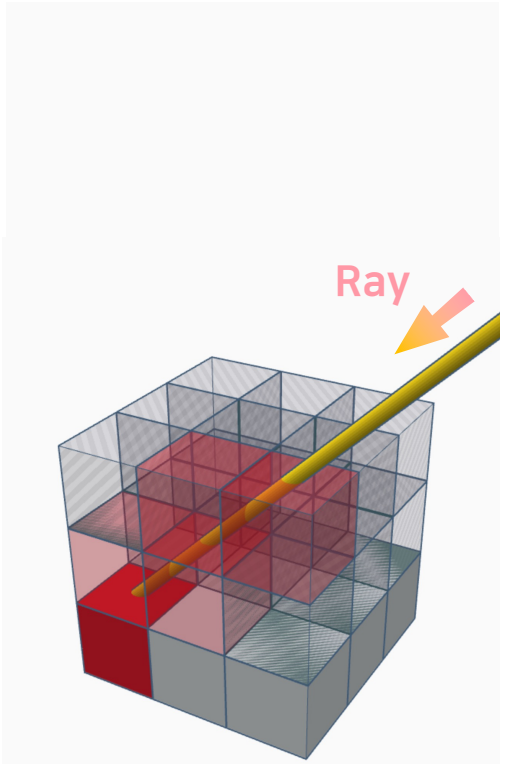
Now we divide dx,dy,dz by that len value.

Then, dx is $dx = \frac{end.x - start.x}{\sqrt{dx^2 + dy^2 + dz^2}}$

From stepX,Y,Z to the cos value, the concept of taking a minus on delta x was applied because it was end-start, not start-end, when calculating the slope.

V. Ray Cast Picking

```
// main loop along raycast vector
while (t <= len) {
    const voxel = this.getBrickState(ix, iy, iz);

    if (voxel) {
        return {
            position: [
            start.x + t * dx,
            start.y + t * dy,
            start.z + t * dz,
            ],
            normal: [
            steppedIndex === 0 ? -stepX : 0,
            steppedIndex === 1 ? -stepY : 0,
            steppedIndex === 2 ? -stepZ : 0,
            ],
            voxel,
        };
    }

// advance t to next nearest voxel boundary
if (txMax < tyMax) {
    if (txMax < tzMax) {
        ix += stepX;
        t = txMax;
        txMax += txDelta;
        steppedIndex = 0;
    } else {
        iz += stepZ;
        t = tzMax;
        tzMax += tzDelta;
        steppedIndex = 2;
    }
} else {
    if (tyMax < tzMax) {
        iy += stepY;
        t = tyMax;
        tyMax += tyDelta;
        steppedIndex = 1;
    } else {
        iz += stepZ;
        t = tzMax;
        tzMax += tzDelta;
        steppedIndex = 2;
    }
}
```



While the t-value is less than or equal to len, receive a brickstate in the voxel and put it in. In BrickState, there is a transparent block, which is an empty block, in which case the value is 0.

And if the state of the voxel is not 0, return the position, normal, and voxel value information of the voxel in the if statement.

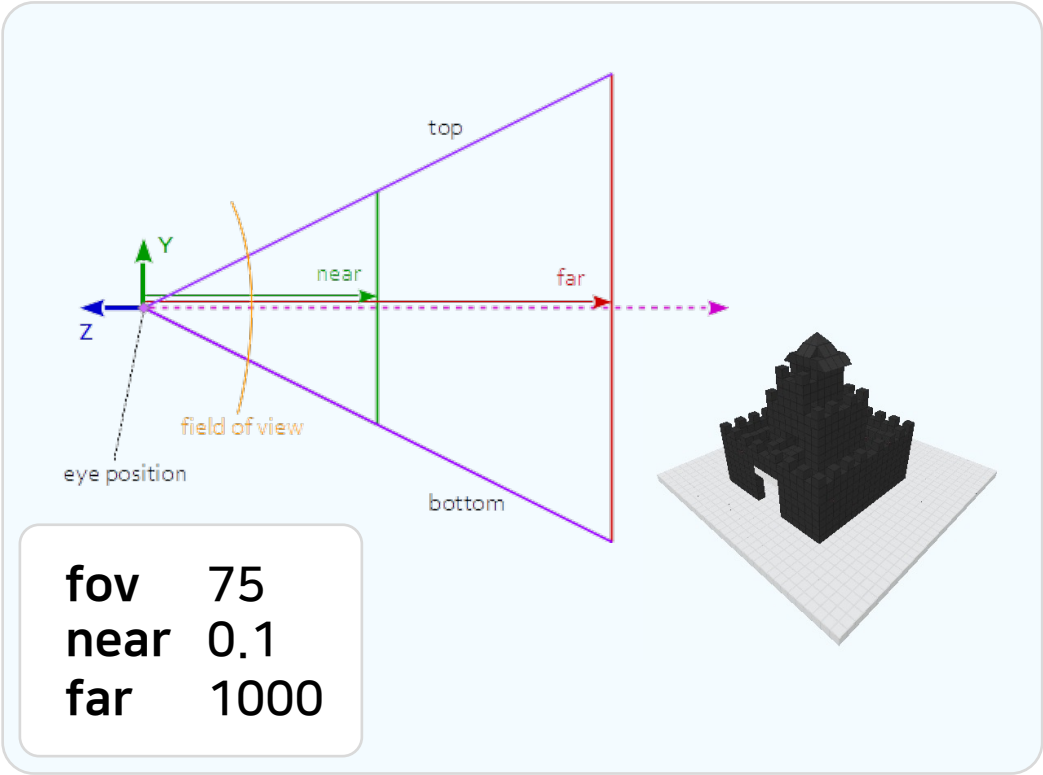
Next, the process of advancing to the nearst voxel boundary proceeds. In each case, step values are added to the ix, iy, and iz values, and the stepped Index values are substituted.

Ⅵ. Camera

Ⅵ. Camera

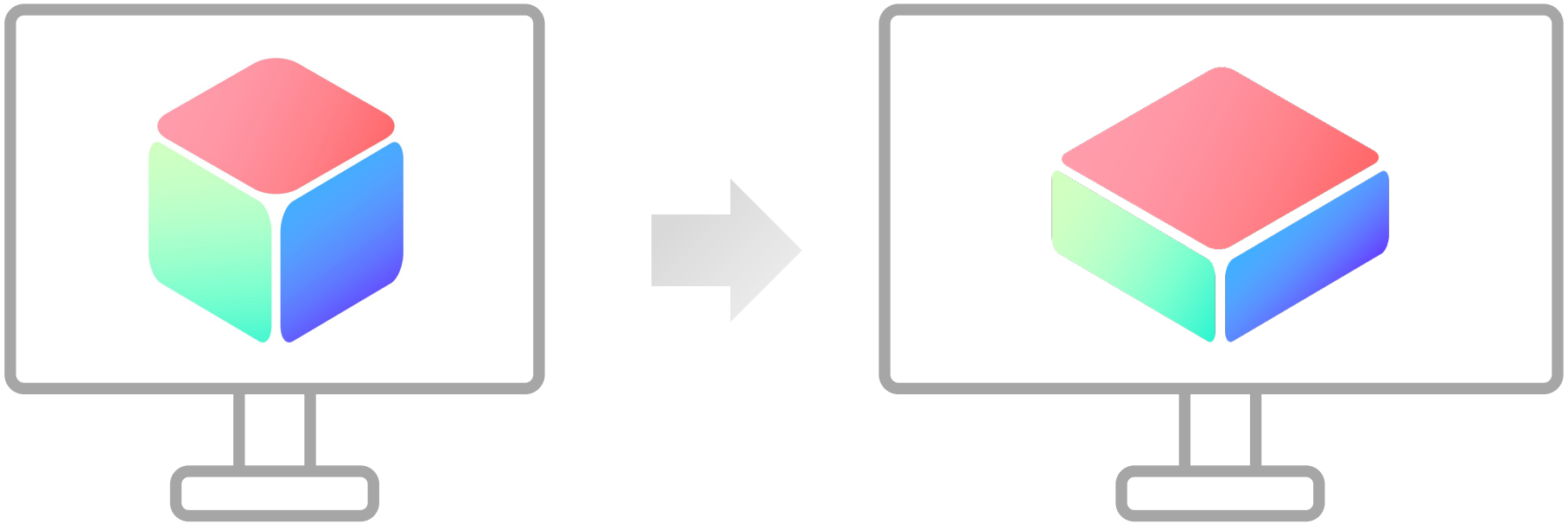
Use PerspectiveCamera()

- **fov** Camera frustum vertical field of view.
- **aspect** Camera frustum aspect ratio.
- **near** Camera frustum near plane.
- **far** Camera frustum far plane.



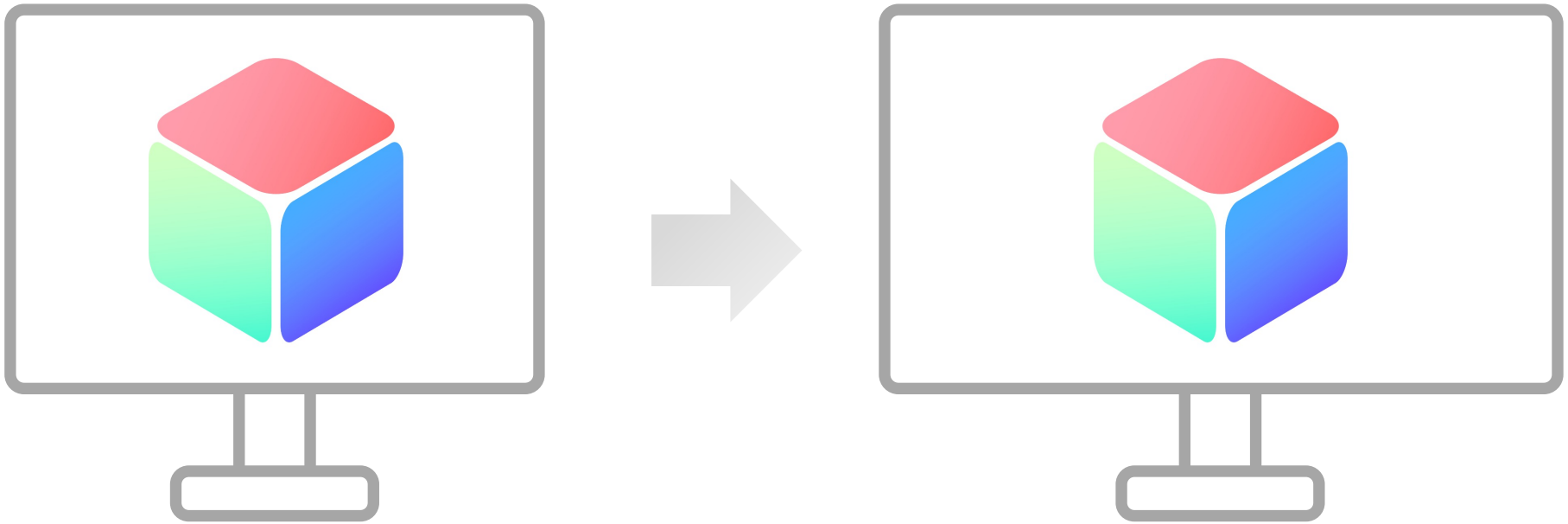
Ⅵ. Camera - Aspect

If the Aspect Ratio value is not optimized,
the object's size ratio will change as the canvas size changes.



VI. Camera - Aspect

Even if the aspect ratio is changed,
the size ratio of the object should be constant.



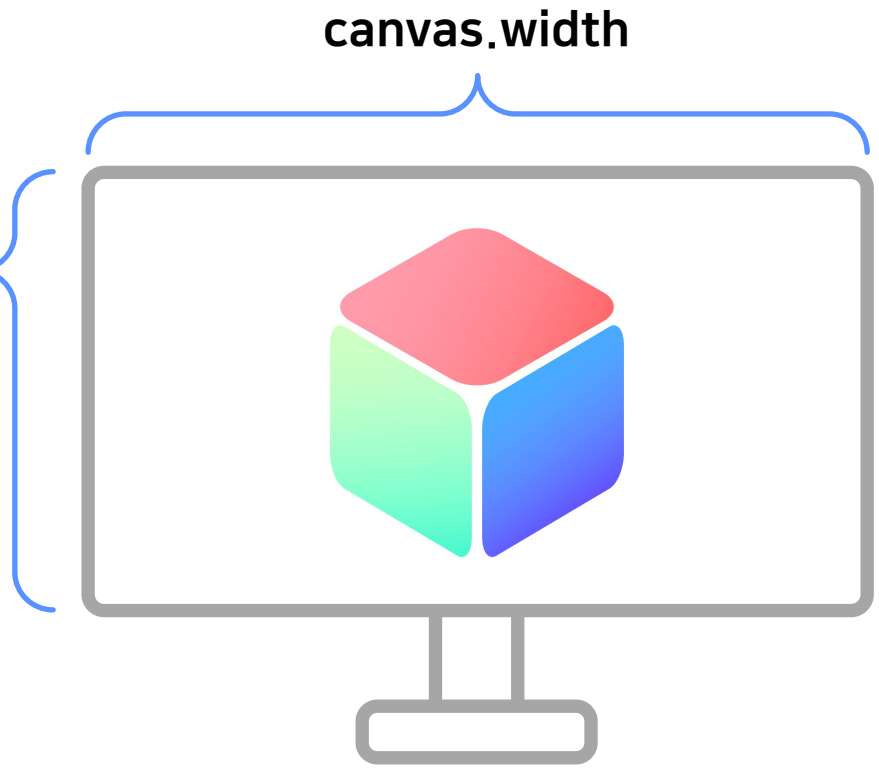
Ⅵ. Camera - Aspect

resizeRendererToDisplaySize()

· Check if the canvas size has changed
· Setsize to the changed canvas size.

· When rendering, the value of the aspect will be canvas.clientWidth / canvas.clientHeight.

· clientWidth and clientHeight get the pixel values of the screen displayed.



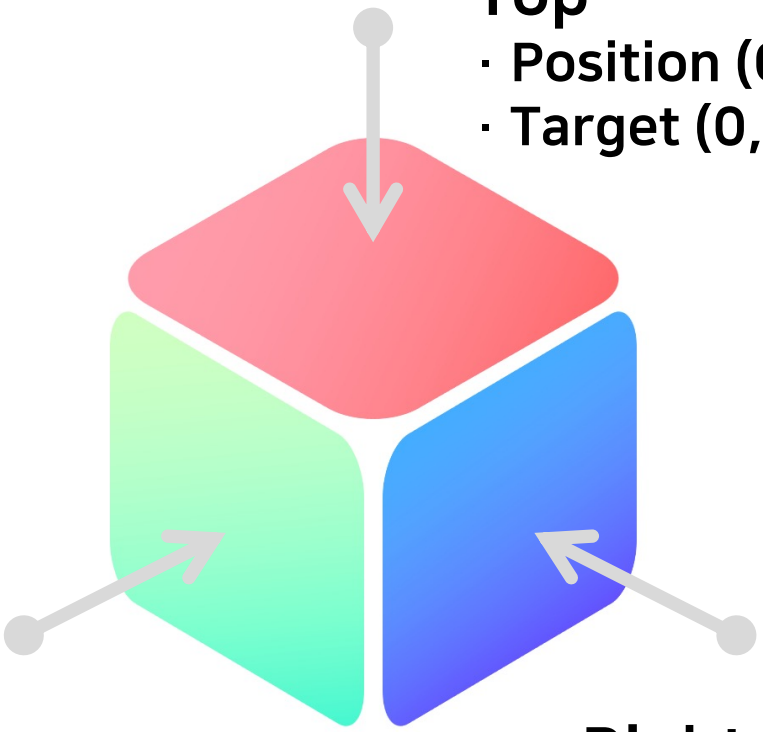
Ⅵ. Camera – Point of View

View Target from Position

Camera Vector = Target - Position

The camera looks at the target, which is the specified coordinate, in Position, which is the camera's position coordinate. Therefore, the camera vector value is the target-position.

By using this, the target coordinates are fixed and the position value is changed so that you can see the front view, right side view, and top view by pressing the viewpoint button.



Top
· Position (0,Y,0)
· Target (0,0,0)

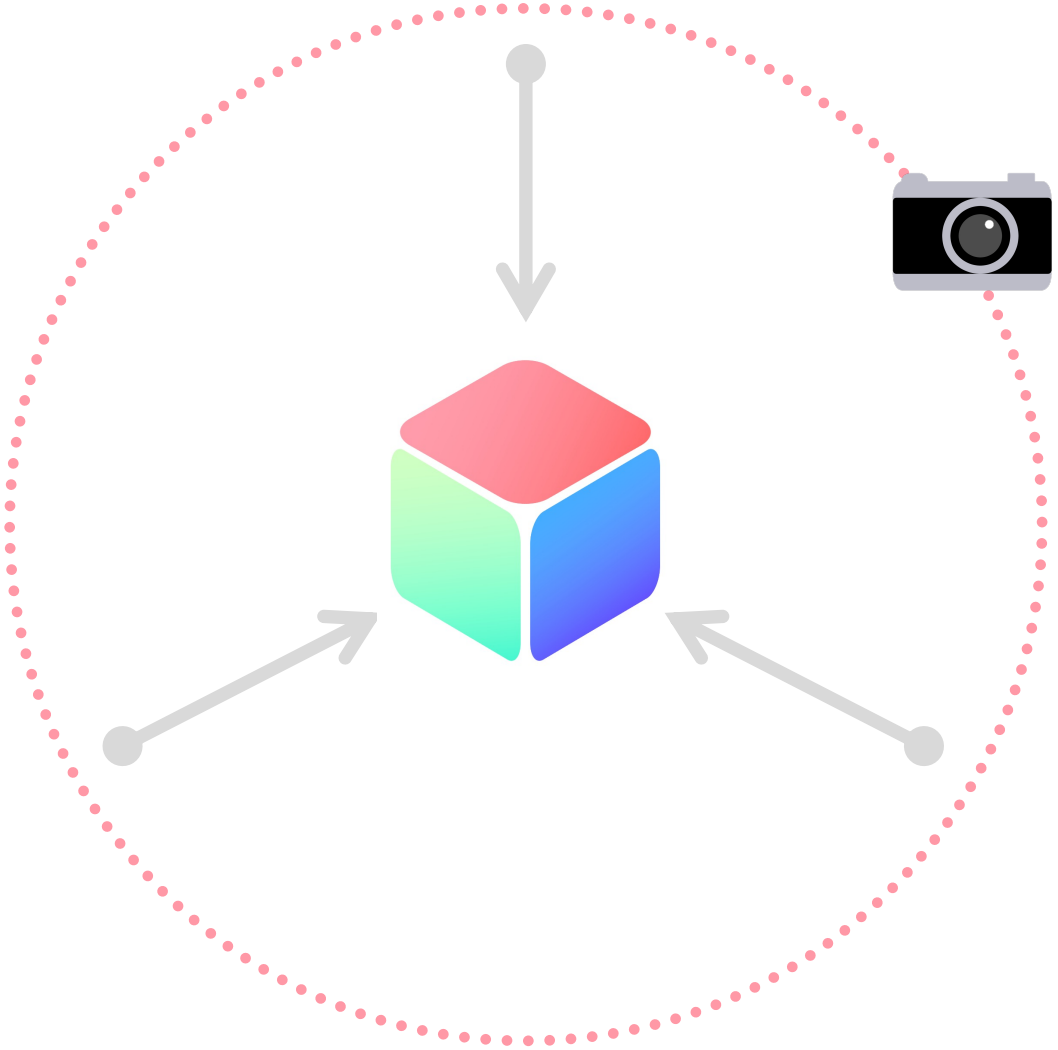
Front
· Position (X,0,0)
· Target (0,0,0)

Right Side
· Position (0,0,Z)
· Target (0,0,0)

VI. Camera – Orbit Control

Camera Rotation

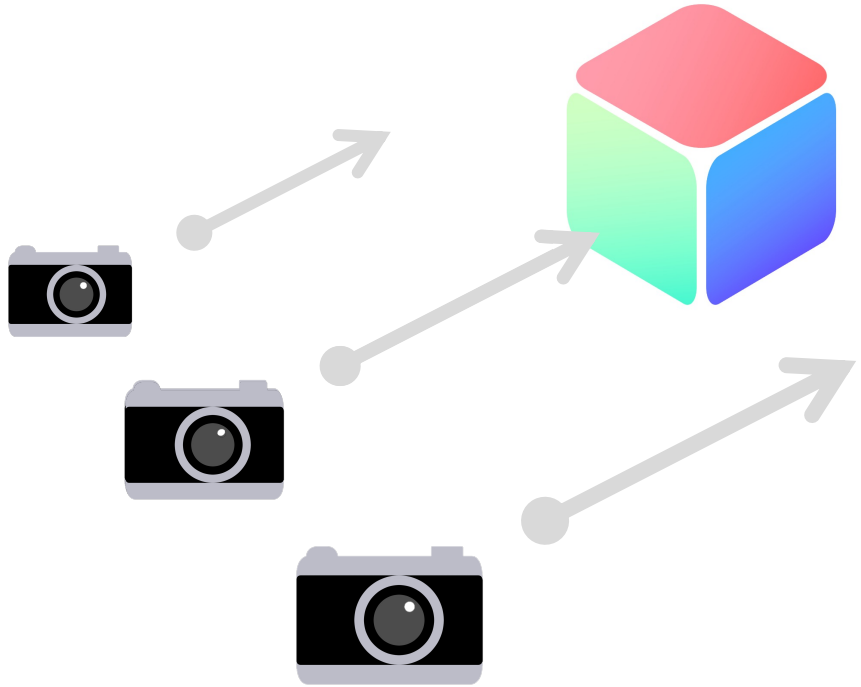
Hold left click and rotate the camera relative to the target.



VI. Camera – Orbit Control

Camera Movement

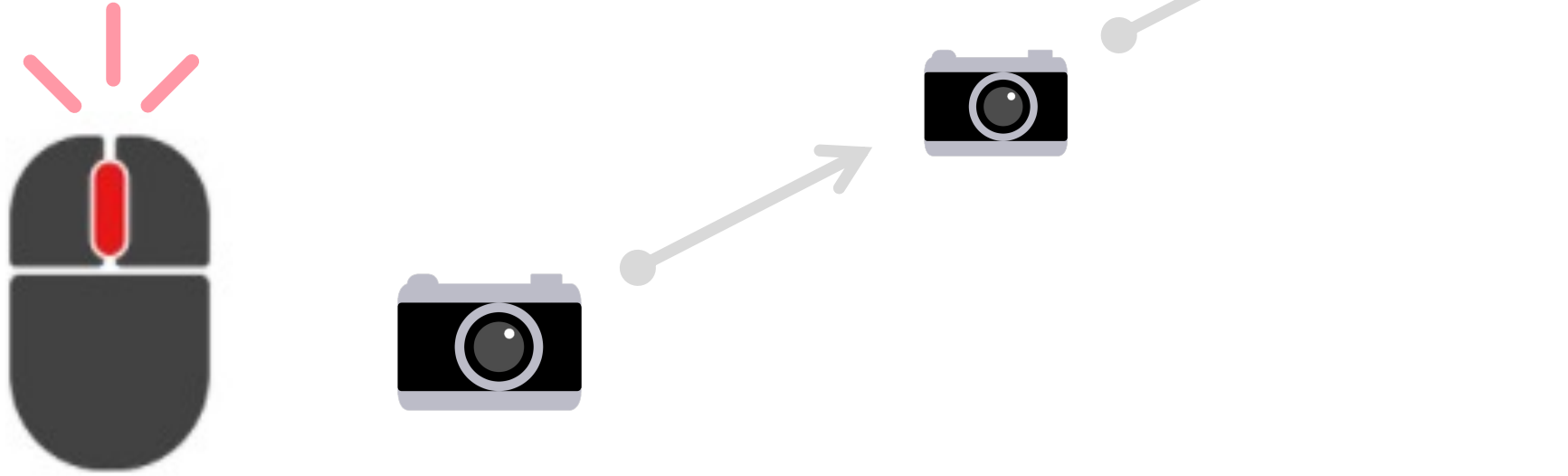
Move the camera up, down, left, or right
while holding the right-click.



Ⅵ. Camera – Orbit Control

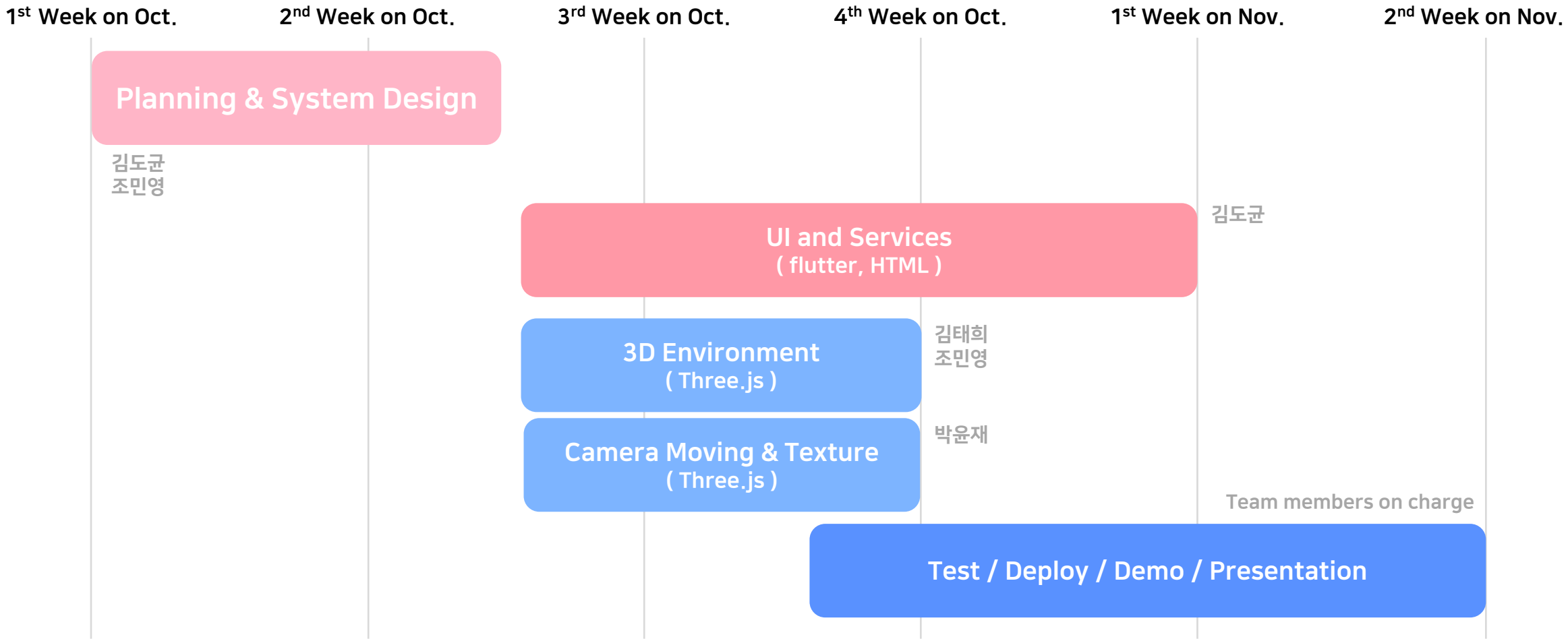
Camera Zoom

Roll the mouse wheel to adjust
the camera zoom.



Ⅶ. Appendix

Ⅶ. Schedule and Roles



Ⅶ. Team Member

ID	201533631
Name	김도균
GitHub	DokySp
Email	uhug@gachon.ac.kr

System Design, 3D modeling, UIX

ID	201835439
Name	김태희
GitHub	taehui530
Email	swanna18@gachon.ac.kr

3D Environment (Picking)

ID	201835455
Name	박윤재
GitHub	Lab00700
Email	rnjsxorl2085@gachon.ac.kr

Camera Moving & Texture

ID	201835527
Name	조민영
GitHub	miiiin0
Email	alsdud5766@gachon.ac.kr

3D Environment

Ⅶ. Additional Information

Youtube

BrickCraft Presentation Video

https://youtu.be/q-H44HPy658

Github

BrickCraft Repository

https://github.com/GC212CG/brickcraft

Youtube

BrickCraft Demo Video

https://youtu.be/q-H44HPy658?t=91

Website

Service URL

https://brickcraft.doky.space

Github

Organization of Team A

https://github.com/GC212CG

Term Project Technical Note

Computer Graphics

| 2021.11.21 |

201533631 김도균
201835439 김태희
201835455 박윤재
201835527 조민영