Generated Planets

20171678 이찬희

Target Users

- Game developer who needs randomly generated planet.
- Gamers whose choice generates a virtual planet.

Features

- A random planet is generated with several factors user can change.
- User can define planet's size, complexity, and color with sliders.
- User can randomly reset noise seed when key is pressed.
- Planet rotates within mouse position.

Concepts and Ideas

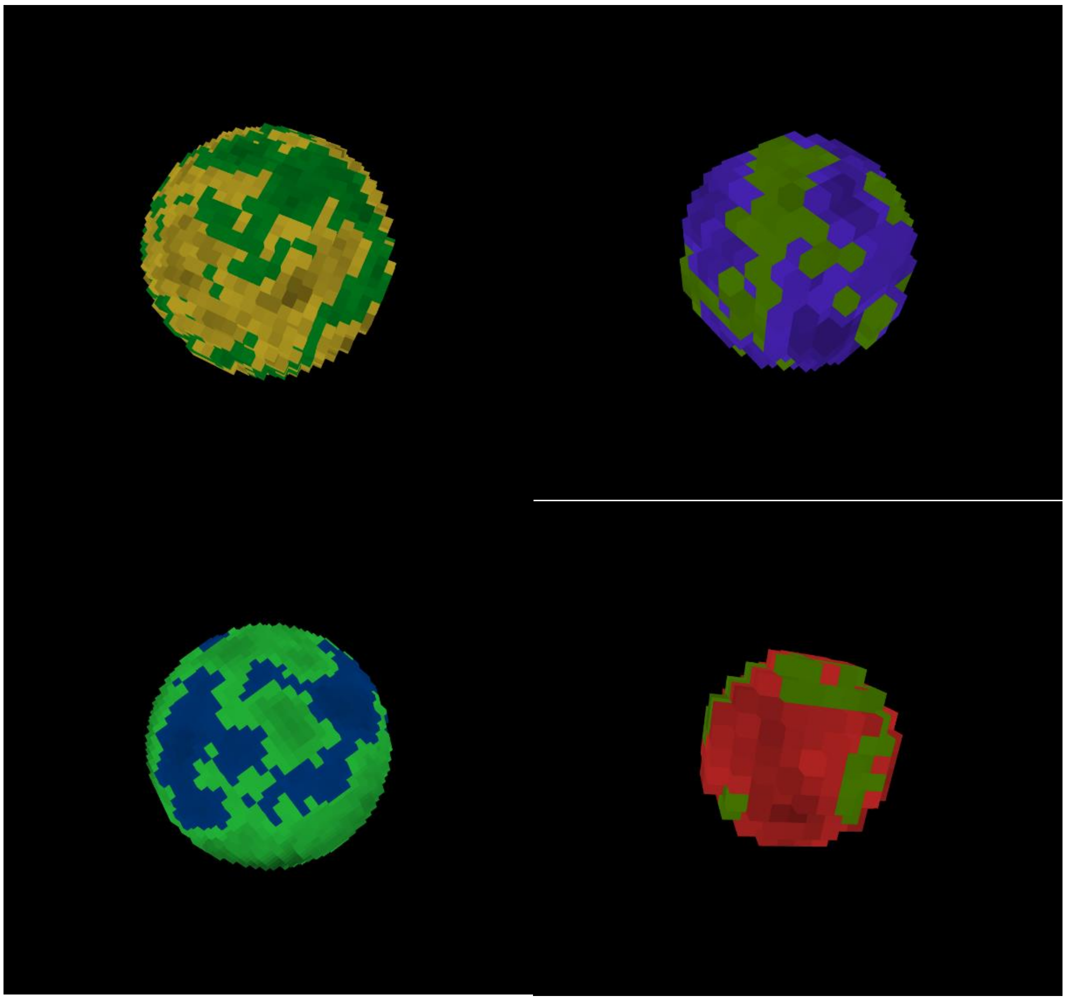
- Randomly generated virtual world in games
  - **MineCraft**

  - **No Man's Sky**

-

These games gave me idea of randomly generated planet with different biomes. By borrowing a cube block, a representative element of Minecraft, I came up with the idea of a spherical planet using voxels.

Screen Shots



Algorithms

■ **Setting Camera position**

```
    // set camera position
    zTranslate = -boxSz_sl.value() * boxSz_sl.value() / 320;
    translate(0, 0, zTranslate);

    let xp = (mouseX - windowWidth / 2);
    let yp = (mouseY - windowHeight / 2)
    rotate(sqrt(pow(xp, 2) + pow(yp, 2)) * 0.01, [-yp, xp, 0])
```

Using mouse position, rotation of camera is calculated.

The position of the z-axis of the camera was calculated in relation to the size of the planet to fit the window, even if the size of the planet increases.

■ **Drawing Voxel Sphere**

```
function VoxelShpere(boxSz) {
    var radius = boxSz - (boxSz / 10);

    noStroke();
    hue_land = hue_sl.value()
    hue_ocean = (hue_sl.value() + 84) % 255;

    for (var x = -boxSz + gridSz; x <= boxSz - gridSz; x += gridSz) {
        for (var y = -
boxSz + gridSz; y <= boxSz - gridSz; y += gridSz) {
            for (var z = -
boxSz + gridSz; z <= boxSz - gridSz; z += gridSz) {

                var d = dist(0, 0, 0, x, y, z);
                if ((d > radius - gridSz) && (d < radius)) {
                    push();
                    // set color
                    var nos = noise(
                        (x + maxBoxSz) * noiseSz_sl.value() * 0.0001,
                        (y + maxBoxSz) * noiseSz_sl.value() * 0.0001,
                        (z + maxBoxSz) * noiseSz_sl.value() * 0.0001)
;

                    if (nos < 0.5)
                        fill(hue_land, 81, 20 + nos * 100);
                    else
                        fill(hue_ocean, 100, 70 - nos * 50);
                    translate(x, y, z);
                    box(gridSz);
                    pop();
                }
            }
        }
    }
}
```

Basic Algorithm is from shaggy on CodePen, and little tweaks were made to set the color of each voxel. Each voxel has its 'nos' value, Perlin noise calculated by its x, y, z axis position and value of 2nd slider. If the value is under 0.5, the voxel is considered as land, and detailed color is calculated using then nos value and vice versa.

Web Link

https://editor.p5js.org/chan2ie/present/hfu2cESR5