Daniel Ávila – 6000975

Vista perspectiva y ortogonal y su relación con three.js

- Vista perspectiva:

La vista de perspectiva es una técnica utilizada en dibujo y diseño para representar objetos tridimensionales en una superficie bidimensional, como un papel o una pantalla de ordenador. La perspectiva se utiliza para crear la ilusión de profundidad y distancia en una imagen, lo que permite al espectador sentir como si estuviera mirando la escena en el espacio real, la vista de perspectiva se basa en la idea de que los objetos parecen más pequeños a medida que se alejan y que las líneas paralelas parecen converger en un punto de fuga. Hay varios tipos de perspectiva, incluyendo la perspectiva de un punto, la perspectiva de dos puntos y la perspectiva de tres puntos, cada una de las cuales se utiliza para representar diferentes ángulos y posiciones de los objetos.

En Three.js, una biblioteca de gráficos en 3D para la web, la vista de perspectiva se puede aplicar utilizando una cámara que utiliza la proyección de perspectiva. La cámara define la posición del observador y la dirección en la que se está mirando, así como la relación de aspecto y la distancia de visualización. La escena en sí misma se define utilizando objetos como mallas y luces, que se colocan en coordenadas tridimensionales.

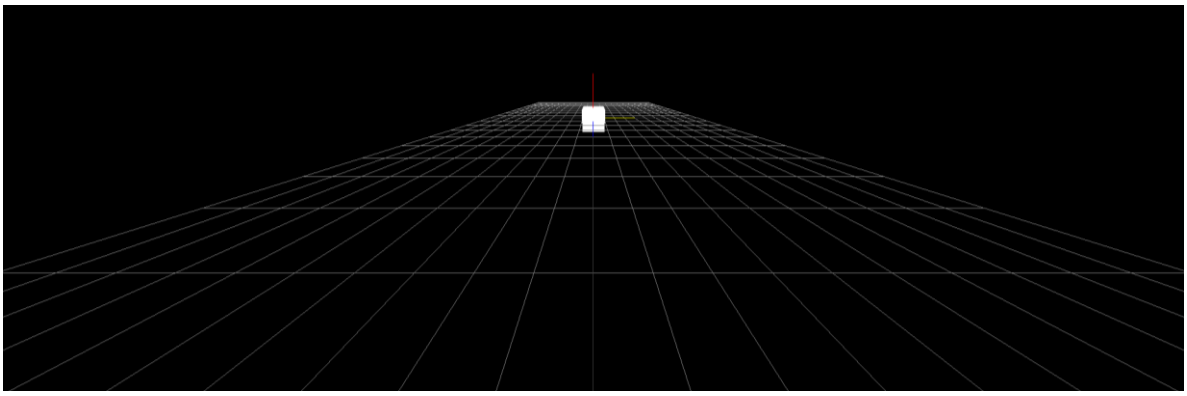
Con Three.js y JavaScript, es posible crear una amplia variedad de escenarios en 3D utilizando la vista de perspectiva. Algunos ejemplos podrían incluir:

- Modelos arquitectónicos: se puede crear una maqueta en 3D de un edificio o una casa, utilizando la vista de perspectiva para dar una idea de cómo se vería en la vida real.
- Juegos: con Three.js, es posible crear juegos en 3D, donde la vista de perspectiva puede utilizarse para dar una sensación de profundidad y distancia en el mundo del juego.
- Visualizaciones científicas: se pueden crear visualizaciones en 3D de datos científicos, utilizando la vista de perspectiva para mostrar la relación entre diferentes variables y permitir que los usuarios exploren los datos desde diferentes ángulos.
- Escenarios de realidad virtual: con la ayuda de dispositivos de realidad virtual, se pueden crear escenarios inmersivos en 3D utilizando la vista de perspectiva, lo que permite a los usuarios sentir como si estuvieran en un entorno real.

Un ejemplo del uso de la vista de perspectiva en three.js podría ser el siguiente:

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>My first three.js app</title>
        <style>
            body { margin: 0; }
        </style>
    </head>
    <body>
        <script type="module">
            import * as THREE from 'https://unpkg.com/three/build/three.module.js';


            // Creamos la escena
            var scene = new THREE.Scene();

            //Grid Helper: Rejilla para establecer un piso
            const size = 10;
            const divisions = 20;

            const gridHelper = new THREE.GridHelper( size, divisions );
            scene.add( gridHelper );

            //Axes Helper: Linea en el eje x, y, z, ayuda a entender la perspectiva
            const axesHelper = new THREE.AxesHelper( 2 );
            axesHelper.setColors( 0xffff00,  0xff0000, 0x0000ff);
            scene.add( axesHelper );

            // Creamos la geometría y el material para una caja
            var geometry = new THREE.BoxGeometry(1, 1, 1);
            var material = new THREE.MeshBasicMaterial({ color: 0xffffff });

            // Creamos la malla para la caja y la agregamos a la escena
            var cube = new THREE.Mesh(geometry, material);
            scene.add(cube);

            // Creamos la cámara con una proyección de perspectiva
            var camera = new THREE.PerspectiveCamera(500, window.innerWidth / window.innerHeight, 0.1, 1000);

            // Posicionamos la cámara en un punto en el espacio y la apuntamos hacia la caja
            camera.position.x = 0;
            camera.position.y = 1.5;
            camera.position.z = 5;
            camera.lookAt(cube.position);

            // Creamos el renderizador y lo agregamos a la página
            var renderer = new THREE.WebGLRenderer();
            renderer.setSize(window.innerWidth, window.innerHeight);
            document.body.appendChild(renderer.domElement);

            // Renderizamos la escena y la cámara
            renderer.render(scene, camera);
        </script>
    </body>
</html>
```

Con este código podemos apreciar cómo se deforma la caja que se agrega a la escena según los valores de perspectiva que le demos a la cámara.



Como vemos no solo es la caja que se deforma sino también el grid helper, o la rejilla que simula el piso, pues de tener una vista ortogonal, los cuadros del piso serian del mismo tamaño, mientras en la foto ocurre que parecen más rectángulos que reducen su tamaño entre más a fondo están en el escenario.

- Vista ortogonal:

La vista ortogonal es un tipo de vista en la que los objetos se representan sin perspectiva, es decir, sin la distorsión de la perspectiva que ocurre cuando los objetos están más lejos. En lugar de eso, los objetos se representan con un tamaño constante, independientemente de su distancia desde la cámara. La vista ortogonal es útil cuando se necesita una vista precisa y detallada de los objetos, especialmente en aplicaciones como el diseño técnico o la representación de planos, en la vista ortogonal, los objetos se representan en relación con una dirección particular, que puede ser cualquiera de los tres ejes cartesianos.

En Three.js, la vista ortogonal se puede crear utilizando una cámara de proyección ortográfica. La cámara ortográfica se define utilizando un volumen de vista ortográfica, que define la sección rectangular del mundo que se representará. La cámara ortográfica es útil para la representación de objetos en una vista plana y sin distorsión, lo que permite al usuario visualizar y manipular objetos con precisión.

La vista ortogonal en Three.js con JavaScript tiene varios usos. Aquí hay algunos ejemplos:

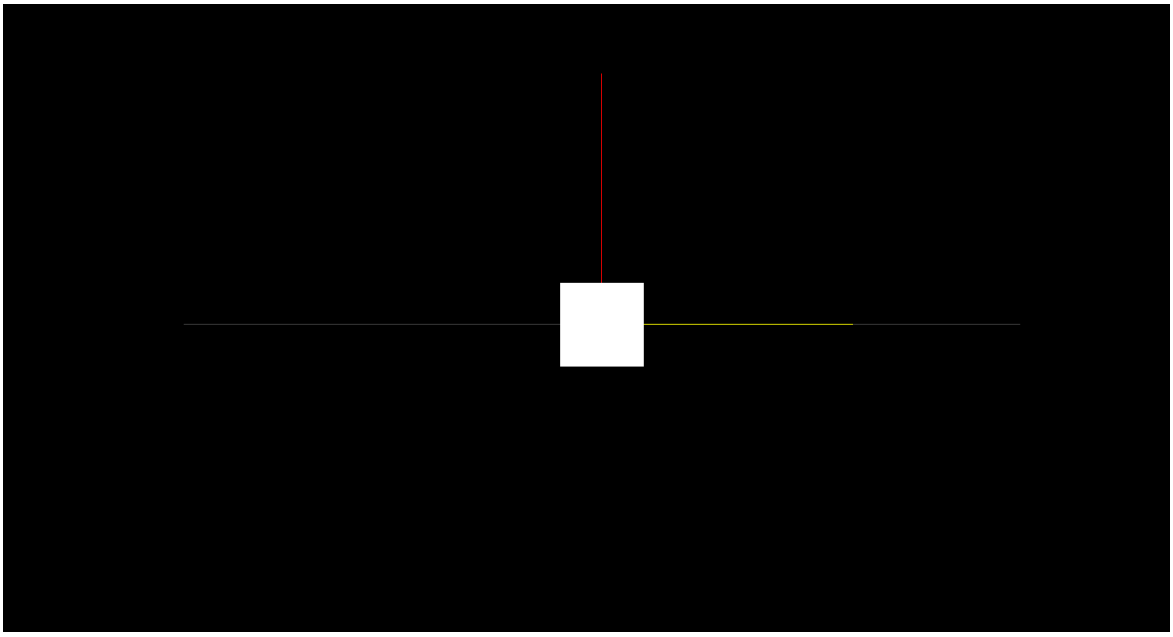
- Representación precisa de objetos en 2D: La vista ortogonal se utiliza comúnmente para la representación de objetos en 2D, como planos arquitectónicos, diseños mecánicos, gráficos vectoriales y diagramas de flujo. La cámara ortográfica en Three.js puede ayudar a representar estos objetos con precisión y en una vista frontal sin distorsiones.
- Interfaces de usuario 2D: La vista ortogonal también se puede utilizar para crear interfaces de usuario 2D, como botones, menús y barras de herramientas. Estas interfaces de usuario se pueden colocar en una capa separada del contenido 3D, lo que permite una representación clara.
- Animación de personajes 2D: La vista ortogonal se utiliza a menudo para la animación de personajes 2D, como juegos de plataformas y juegos de rol. Al representar los personajes y el entorno en una vista frontal, se puede animar a los personajes con mayor facilidad.

- Mapas 2D y sistemas de información geográfica: La vista ortogonal también se utiliza en sistemas de información geográfica y mapas en 2D, ya que permite una representación precisa de los datos geográficos en una vista plana.

Un ejemplo de la cámara ortográfica en three.js seria el siguiente:

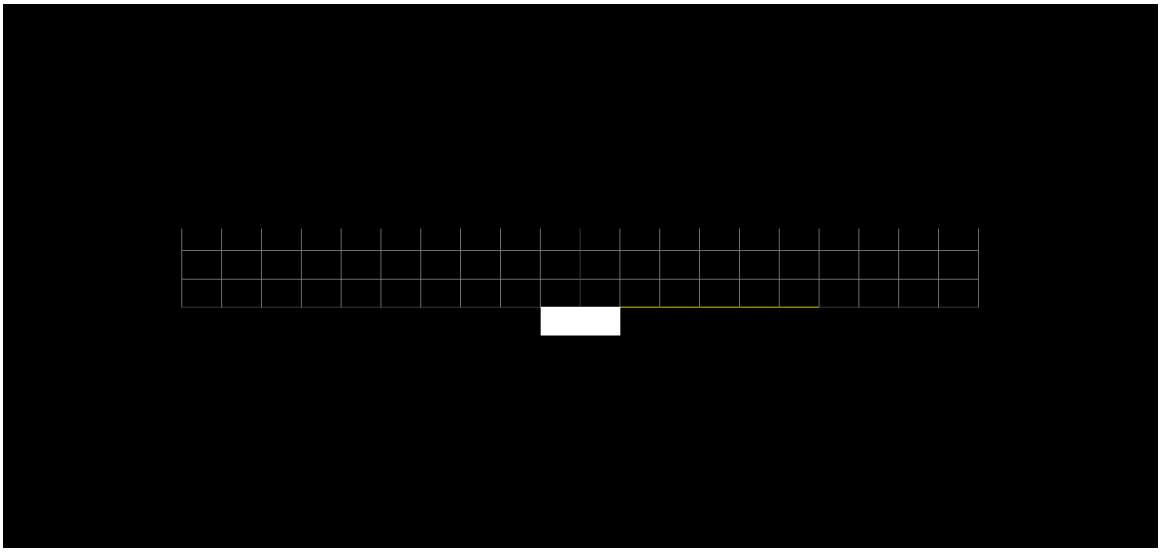
```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>My first three.js app</title>
        <style>
            body { margin: 0; }
        </style>
    </head>
    <body>
        <script type="module">
            import * as THREE from 'https://unpkg.com/three/build/three.module.js';
            // Creamos una escena
            var scene = new THREE.Scene();

            //Grid Helper: Rejilla para establecer un piso
            const size = 1000;
            const divisions = 20;
            const gridHelper = new THREE.GridHelper( size, divisions );
            scene.add( gridHelper );

            //Axes Helper: Linea en el eje x, y, z, ayuda a entender la perspectiva
            const axesHelper = new THREE.AxesHelper( 300 );
            axesHelper.setColors( 0xffff00,  0xff0000, 0x0000ff);
            scene.add( axesHelper );

            // Creamos una geometría de un plano
            var planeGeometry = new THREE.PlaneGeometry(100, 100);

            // Creamos un material de color blanco
            var planeMaterial = new THREE.MeshBasicMaterial({ color: 0xffffff });

            // Creamos una malla a partir de la geometría y el material
            var planeMesh = new THREE.Mesh(planeGeometry, planeMaterial);

            // Agregamos la malla a la escena
            scene.add(planeMesh);

            // Creamos una cámara ortográfica
            var camera = new THREE.OrthographicCamera(
            window.innerWidth / -2, // Left
            window.innerWidth / 2, // Right
            window.innerHeight / 2, // Top
            window.innerHeight / -2, // Bottom
            1, // Near
            100 // Far
            );

            // Establecemos la posición de la cámara
            camera.position.set(0, 0, 1);

            // Establecemos la dirección de la cámara
            camera.lookAt(scene.position);

            // Agregamos la cámara a la escena
            scene.add(camera);

            // Creamos un renderizador de Three.js
            var renderer = new THREE.WebGLRenderer();

            // Establecemos el tamaño del renderizador
            renderer.setSize(window.innerWidth, window.innerHeight);

            // Agregamos el renderizador al documento
            document.body.appendChild(renderer.domElement);

            // Renderizamos la escena con la cámara ortográfica
            renderer.render(scene, camera);

        </script>
    </body>
</html>
```

Al ejecutar el código obtenemos lo siguiente:



Si ajustamos los valores de la cámara esta puede mostrar la vista ortogonal de objeto desde otro punto de vista, sin embargo, al tratarse de esta cámara comprender la profundidad puede llegar a ser un tanto más complejo que con la de perspectiva, pues aquí todos los elementos de la escena conservan su tamaño.

Por ejemplo, si ajustamos la cámara en eje y de 0 a 1:



Vemos como la rejilla, la caja y el axes helper se muestran de tamaños equivalentes, pero se torna difícil comprender lo que ocurre en la escena.

En la computación gráfica, el cálculo de una vista en perspectiva se realiza mediante una proyección perspectiva de los objetos 3D en un plano 2D. Esta proyección se realiza utilizando una cámara virtual que simula la forma en que un ojo humano percibe la escena. Los parámetros que se utilizan en el cálculo de la perspectiva incluyen:

1. Ángulo de visión (FOV): El ángulo de visión define cuánto de la escena es visible desde la cámara. Especifica cuánto se extiende el cono de vista, es decir, la porción de la escena que será visible en la imagen. Un ángulo de visión más amplio capturará más de la escena, mientras que un ángulo de visión más estrecho capturará menos.
2. Relación de aspecto: La relación entre la anchura y la altura de la imagen que se está generando. Este parámetro es importante para asegurarse de que la imagen final se ajuste correctamente al tamaño de la pantalla.
3. Distancia de visualización (f): Es la distancia entre el plano de la cámara y el punto focal de la cámara. La distancia de visualización determina la profundidad de campo de la imagen, es decir, la cantidad de la escena que aparecerá enfocada en la imagen final.
4. Posición de la cámara: La posición de la cámara se define en términos de su ubicación (x, y, z) en la escena y su orientación (pitch, yaw, roll). Estos valores determinan el punto de vista desde el que se verá la escena.

Estos parámetros se utilizan en la matriz de proyección que se aplica a los vértices de los objetos 3D antes de ser renderizados en la pantalla. Al cambiar estos parámetros, se puede ajustar la apariencia de la vista en perspectiva, permitiendo al usuario ver la escena desde diferentes puntos de vista y con diferentes grados de profundidad y detalle.

En THREE.js, varios elementos intervienen en la configuración de las vistas en perspectiva y ortogonales. Aquí están algunos de los elementos clave y su significado:

1. Cámara (Camera): En THREE.js, la cámara define el punto de vista desde el cual se visualiza la escena. La cámara puede ser una cámara en perspectiva (PerspectiveCamera) o una cámara ortogonal (OrthographicCamera).
2. Escena (Scene): La escena es el contenedor para todos los objetos 3D que se están renderizando. La escena puede contener objetos, luces, cámaras y cualquier otro elemento que se quiera visualizar.
3. Renderer: El renderer es el objeto que renderiza la escena y la cámara. THREE.js proporciona varios tipos de renderers, como WebGLRenderer para renderizar en WebGL, CanvasRenderer para renderizar en 2D y SVGRenderer para renderizar en SVG.
4. Geometría (Geometry): En THREE.js, una geometría es un objeto que describe la forma y la posición de un objeto 3D. Las geometrías pueden definirse de muchas maneras diferentes, como una malla de triángulos o un conjunto de curvas.
5. Material: El material define cómo se verá una geometría en la escena. Los materiales pueden ser simples, como un color sólido, o más complejos, como una textura.
6. Luz (Light): Las luces se utilizan para iluminar la escena. En THREE.js, hay varios tipos de luces, como luces direccionales, puntuales o de punto.

Estos elementos se utilizan conjuntamente para crear y configurar la vista en perspectiva u ortogonal. Por ejemplo, se crea una cámara, se añade a la escena y se configuran los parámetros de la cámara como el ángulo de visión, la relación de aspecto y la posición de la cámara. Luego se crean

geometrías y materiales, se añaden a la escena y se configuran las luces para iluminar la escena. Finalmente, el renderer se utiliza para renderizar la escena y la cámara, y se muestra el resultado en la pantalla.

Referencias:

1. OpenAI. (2023). GPT-3: Language Models are Few-Shot Learners. arXiv:2005.14165.
2. Three.js. (2023). Home. https://threejs.org/
3. Three.js. (2023). Camera. https://threejs.org/docs/#api/en/cameras/Camera
4. Three.js. (2023). OrthographicCamera. https://threejs.org/docs/#api/en/cameras/OrthographicCamera
5. Three.js. (2023). PerspectiveCamera. https://threejs.org/docs/#api/en/cameras/PerspectiveCamera
6. W3Schools. (2023). JavaScript. https://www.w3schools.com/js/default.asp
7. Mantri, S. (2019). Orthographic projection in computer graphics. GeeksforGeeks. https://www.geeksforgeeks.org/orthographic-projection-in-computer-graphics/
8. Williams, R. (2013). The Animator's Survival Kit. Faber & Faber.
9. Lengyel, E. (2010). Mathematics for 3D Game Programming and Computer Graphics. Cengage Learning.