2023 年度 実世界情報実験２ 「CG」テキスト

第 2～7 週：Three.js を使った 3 次元 CG プログラミング

立命館大学 情報理工学部 実世界情報コース

1. 実験の目的と手順

本実験は，(1) Three.js を用いた CG プログラミングの手法を学ぶ，(2) プログラミングを通して CG の基礎技術を体験的に学ぶ，(3) インタラクティブな 3 次元 CG アニメーションの作成方法について学ぶことを目的としている．テキスト中に課題があるので，その指示にしたがってプログラムを作成・実行し，実行結果を確認すること．**実行結果が確認できたら TA のチェックを受けること**．課題ソースコードは，下記 OneDrive からダウンロードできる．

https://ritsumei365-my.sharepoint.com/:f:/g/personal/f-naka_fc_ritsumei_ac_jp/Ei98FadYUf9BtBHtagY-MP8BugQlC7lVmQMR6gSuajlhBA?e=4HS5sA

2. Three.js

2.1. WebGL とは

WebGL（ウェブジーエル）は，ウェブブラウザで 3 次元コンピュータグラフィックスを表示させるための標準仕様．非営利団体の Khronos Group で管理されている．WebGL 1.0 は，ブラウザ上で利用できる OpenGL ES 2.0 の派生規格であるが，細部に違いがある [1]. WebGL 2.0 は，ブラウザ上で利用できる OpenGL ES 3.0 の派生規格であるが，細部に違いがある [2]. WebGL は HTML5 の canvas 要素に描画する．

2.2. Three.js とは

手軽に 3D コンテンツを制作できる商用利用可能な JavaScript ライブラリ. WebGL だけで 3D 表現をするには, 立方体 1 つ表示するだけでも多くの JavaScript や GLSL コードを書く必要があり専門知識も必要となる．Three.js を使えば JavaScript の知識だけで簡単に 3D コンテンツが作成できるため, 手軽に扱えるようになる．Three.js は，ウェブブラウザ上でリアルタイムレンダリングによる 3 次元コンピュータグラフィックスを描画する，クロスブラウザ対応の軽量な JavaScript ライブラリ及びアプリケーションプログラミングインタフェースである．HTML5 の canvas 要素, Scalable Vector Graphics, WebGL との組み合わせが可能である．ソースコードは GitHub でホストされている．WebGL という Web 標準技術の登場により [3] 商用のブラウザ拡張機能に頼る必要がなく，HTML ファイル内に埋め込まれた JavaScript を介して GPU アクセラレーションによる動的表現を描画することが可能になった [4]. Three.js は，WebGL の API を簡略化するためのラッパである．

2.3. プログラムのダウンロード

任意のパソコンで任意の場所に jikken2-CG という作業フォルダを作成する．以降は，このフォルダ以下で課題を作成する. そして, サンプルプログラムを先ほど作成した「jikken2-CG」にダウンロードして保存する．

https://ritsumei365-my.sharepoint.com/:f:/g/personal/f-naka_fc_ritsumei_ac_jp/Ei98FadYUf9BtBHtagY-MP8BugQlC7lVmQMR6gSuajlhBA?e=4HS5sA

上記 URL にアクセスして「ソースコード」フォルダをダウンロード，解凍し，中身を全て「jikken2-CG」フォルダに移す．課題を行う場合は，作業フォルダ「jikken2-CG」内に課題ファイルを作成することとし，他の場所には作成しないようにする．（ファイル紛失防止のため）

2.4. 実行方法

index.html をブラウザで実行する．WebGL 対応ブラウザは幅広く，例として GoogleChrome，Firefox，Opera，Safari，InternetExproler， MicrosoftEdge の各最新版が対応している．本実験では GoogleChromeを
その他にも対応しているブラウザは以下のリンク先で確認できる．

https://caniuse.com/webgl

また，ファイルを実行ではなく編集したい時は，ブラウザではなくエディタで開くことで編集ができる．例として，Windows では標準搭載のメモ帳，Mac では標準搭載のテキストエディットで開くことができる．その他に使いたいエディタあれば，それを用いて編集しても良い．

2.5. プログラムの構成

HTML ファイル内から JavaScript を呼び出し，ブラウザ上で WebGL を動作させている．下の例では index.html から three.js と index.js を呼び出し，動作させている．

/*index.html*/

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" />
    <!-- three.js を読み込む -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/101/three.min.js"></script>
    <!-- index.js を読み込む -->
    <script src="index.js"></script>
</head>
<body>
    <canvas id="myCanvas"></canvas>
</body>
</html>
```

3. 3 次元 CG プログラミング

3.1. ウィンドウを開く

課題 1 sample1.js は，500×500 ドットで，背景が黒のウィンドウを画面左上にオープンするプログラムである（太字の行は主な新しい学習内容を示す）．index.html に sample1.js を読み込ませるように編集し，実行しなさい．また，ウィンドウサイズや背景色を任意に変更したプログラム kadai1.js を作成せよ．（※この課題は TA チェック不要）

```
1  /* sample1.js */
2  window.addEventListener("DOMContentLoaded", init);
3
4  function init() {
5      const width = 500;
```

```
    const height = 500;

    // レンダラーを作成
    const renderer = new THREE.WebGLRenderer({
        canvas: document.querySelector("#myCanvas")
    });
    renderer.setSize(width, height); /* ウィンドウサイズの設定 */
    renderer.setClearColor(0x000000); /* 背景色の設定 */

    // シーンを作成
    const scene = new THREE.Scene();

    // カメラを作成
    const camera = new THREE.PerspectiveCamera(45, width / height);
    camera.position.set(0, 0, +1000);

    // 初回実行
    renderer.render(scene, camera);
}
```

デバック方法（Chrome）
「F12」キーを押すと，デバック用の Window が出ます

3.2. 立方体を描いてみる

課題 2　sample2.js は，sample1.js に，立方体を描くコードを追加したものである．index.html に sample2.js を読み込ませるように編集し，実行しなさい．次に 3.3 節を参考にして，sample2.js を基に，中心(0.0, 0.0, 0.0), 半径 5 の球を描くプログラム kadai2.js を作成しなさい．

```
/* sample2.js */

/*■kadai1.js と同じ■*/

    // カメラを作成
    const camera = new THREE.PerspectiveCamera(45, width / height);
    camera.position.set(0, 0, 50);

    // 平面を作成
    const geometry = new THREE.BoxGeometry(10, 10, 10);
    const material = new THREE.MeshStandardMaterial({
        color: 0xffffff
    });
```

```
27      const box = new THREE.Mesh(geometry, material);
28      scene.add(box);
29
30      // 平行光源
31      const directionalLight = new THREE.DirectionalLight(0xffffff);
32      directionalLight.position.set(1, 1, 1);
33
34      // シーンに追加
35      scene.add(directionalLight);
36
37      // 初回実行
38      renderer.render(scene, camera);
39  }
```

3.3. 様々なオブジェクトの描画

①ジオメトリの作成　②マテリアルの作成　③メッシュの作成という 3 つの手順を踏むことでオブジェクトを作成できる.

3.3.1.　ジオメトリ

ジオメトリ（形状）を作成する方法について説明する．Three.js では以下に示すように定義する.

```
const geometry = new THREE.PlaneGeometry(100, 100);
```

このジオメトリの関数には以下のようなものがある．図形によって引数の数が異なり，デフォルト値が指定されているものは省略可である．（デフォルト値は“=1”のように表記している）

●平面

四角形：PlaneGeometry(横幅，縦幅，横の分割数=1，縦の分割数=1)

円形：CircleGeometry(半径,分割数=8，弧の始点=0，弧の長さ=Math.PI*2)

輪形 : RingGeometry(穴の半径,半径，弧の分割数=8，幅の分割数=8，弧の始点=0，弧の長さ= Math.PI*2)

●立体

直方体：BoxGeometry(横幅,高さ,奥行き，横幅の分割数=1,高さの分割数=1,奥行きの分割数=1)

球体：SphereGeometry(半径,縦の分割数=8,横の分割数=6，横の弧の始点=0，横の長さ= Math.PI*2,
縦の弧の始点=0，縦の弧の長さ= Math.PI*2)

円錐 : ConeGeometry(半径,高さ，面の分割数=8，高さの分割数=1，端が開いている=false，弧の始点=0,
弧の長さ= Math.PI*2)

円柱：CylinderGeometry(上面の半径,底面の半径,高さ，面の分割数=8，高さの分割数=1,
端が開いている=false，弧の始点=0，弧の長さ= Math.PI*2)

トーラス：TorusGeometry(円の半径=1，管の半径=0.4，円の分割数=8，管の分割数=6,
弧の長さ= Math.PI*2)

3.3.2. CG におけるシェーディング

マテリアル作成のためにシェーディングについて説明する．シェーディングとは，陰影付け処理である．物体表面で反射して視点に到達する光の強度を計算することにより，物体の色の濃淡と質感を再現する．物体表面で反射する光の成分は，おおむね 3 つの成分で表すことができる．シェーディングでは，視点に到達するこれらの成分の強度を別々に計算し，重ね合わせることで物体の一点の輝度を求める．

(1) 拡散反射成分

拡散反射成分とは，物体表面の非常に薄い層で起こる光の反射である．拡散反射光は，すべての方向に一様に散乱されるため，入射光の方向と強度および表面の向きのみで計算され，視点の方向には依存しない． 拡散反射成分は，物体の材質が持つ色を反映しており，面の傾きにより生成される濃淡の違いが陰影として知覚される．

(2) 鏡面反射成分

物体表面に細かな凹凸がなく滑らかであると，入射光は特定の方向に強く反射され，ハイライトと呼ばれる輝点を生ずる．このような反射を鏡面反射と呼び，物体表面に光沢感を与える．鏡面反射の方向は，面の法線方向に対して入射光の方向と対称となる角度（正反射方向）付近に分布している．したがってハイライトの強度は視点位置によって異なり，視点方向が正反射方向と一致したときに最大となる．

(3) 環境光反射成分

われわれの周囲の空間は，光源からの直接光の他に空気による散乱光や他の物体表面からの副次的な反射によりすべての方向から弱く照らされている．このため，物体の影の部分もある程度の明るさを持つ．CG では，このような間接光の成分を環境光と呼び，それによる反射成分を考慮する．環境光成分を考慮しない画像は，宇宙空間で撮影した写真のように影の部分が真っ黒になり，不自然な印象を与える．

Three.js では，(1), (2) を物体が持つマテリアルとして設定し，(3) をライトとして設定する．（旧バージョンでは，(3) もマテリアルに含まれていた）

3.3.3. マテリアル

マテリアル（材質）を作成する方法について説明する．前述した物体表面の持つ光の反射特性を設定することで色を指定することができる．Three.js では，以下に示すようにマテリアルを定義する．

```
const material = new THREE.MeshStandardMaterial({
    color: 0xffffff, specular: 0x666666, shininess: 30 })
```

ここで，“color” や “specular” などの名前付き引数をプロパティと呼び，反射特性を直接指定することができる．色は RGB 値を 16 進数で表したカラーコードを使用する．主なプロパティは次の通り．（参考：https://akros-ac.jp/4107/）

color : 拡散反射光の色
　　設定値はカラーコードで指定（デフォルト値は 0xffffff）

specular : 鏡面反射光の色
　　設定値はカラーコードで指定（デフォルト値は 0x666666）

shininess : 鏡面反射光の強さ
　　設定値は Float で指定（デフォルト値は 30）

emissive : 発光体の場合の発光色

設定値はカラーコードで指定（デフォルト値は 0x222200）

emissiveIntensity : 発光の強さ

設定値は Float で指定（デフォルト値は 1）

map : カラーマップ

設定値は texture で指定（デフォルト値は null）

また，マテリアルの関数には以下のものがある．

MeshStandardMaterial() : 標準的なマテリアル
MeshBasicMaterial() : ライトの影響を受けないマテリアル
MeshNormalMaterial() : 法線ベクトルを RGB カラーで可視化するマテリアル
MeshPhongMaterial() : 光沢のあるマテリアル
MeshLambertMaterial() : 光沢のないマテリアル
MeshToonMaterial() : トゥーン調のマテリアル
MeshPhysicalMaterial() : MeshStandardMaterial を拡張したマテリアル

3.3.4. テクスチャ

Three.js では，あらかじめテクスチャとして読み込んでおいた画像を，マテリアルの map プロパティに設定することでテクスチャを貼るためのマテリアルを作成できる．テクスチャに使用する画像は，2 の累乗の高さ・幅が推奨されている．

```
const loader = new THREE.TextureLoader();
const texture = loader.load('画像のパス');          /*  画像を読み込む  */
const material = new THREE.MeshStandardMaterial({
      map: texture
});
```

画像やオブジェクトファイルなどを使用する場合は，同一のオリジンのリソースであるか別オリジンからのリソースを使用することを許可する必要がある．ここでは一例として GoogleChrome および Safari のセキュリティを無効にすることで方法を記載する．（実験以外では使用しない）

・GoogleChrome の場合

ショートカットを作成し，以下を指定する．

"chrome.exe へのパス" --disable-web-security --user-data-dir="C://Chrome dev session"

作成したショートカットから起動する．

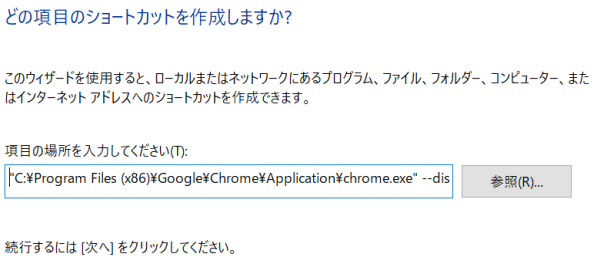

図 1　Chrome の設定

・Safari の場合

Develop（開発）タブから Dissable Cross-Origin Restrictions（クロスオリジンの制限を無効にする）を選択する．

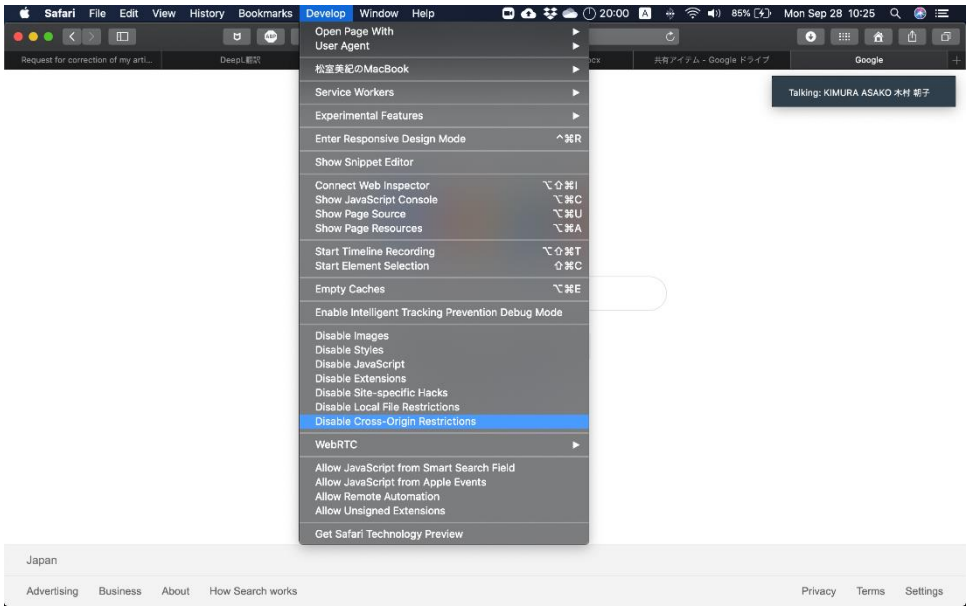

図 2　Safari の設定

3.3.5.　メッシュ

メッシュを作成する方法について説明する．Three.js ではジオメトリとマテリアルを結合しメッシュを作成することで表示可能な 3D オブジェクトが得られる．以下に示すようにメッシュを定義する．

```
const mesh = new THREE.Mesh(geometry, material);
```

また，オブジェクトはシーンに配置されたカメラの画角内に配置されることで初めて描画される．以下に示すようにシーンに配置する．

```
mesh.position.set(0, 0, 0);    /* オブジェクトの中心座標（原点の場合は省略可）*/
scene.add(mesh);
```

3.3.6.　マテリアルの変更

課題 3　sample2.js を基にして，マテリアルを MeshBasicMaterial() 及び MeshNormalMaterial() に変更したトーラスを描くプログラム kadai3-1.js, kadai3-2.js をそれぞれ作成し，比較しなさい．

```
sample2.js を改変すること
```

3.4. ライティング

ここでは光源の種類と設定法について述べる．Three.js では光源を以下のように定義する．

```
const light = new THREE.DirectionalLight(0xffffff, 1);
scene.add(light);
```

Three.js の光源には以下の 6 種類が存在する．

(1) 平行光源

太陽光のように無限遠点に光源が存在するような光源を平行光源という．このような場合は，光は平行に到達するため，光の強度と方向は場所によらず一定とみなすことができる．

```
DirectionalLight(光源色=0xffffff, 光強度=1)
```

(2) 点光源

裸電球のように方向性をもたない比較的小さな光源が局所的に空間を照らすような光源を点光源という．点光源は光源からの距離が離れるに従って到達する光の強度が低下する．

```
PointLight(光源色=0xffffff, 光強度=1, 最大照射距離=0, 光の減衰率=1)
```

(3) スポット光源

方向性をもった点光源をスポット光源という．懐中電灯やスポットライトのように円錐状に光が放射される．光軸はデフォルトで原点(0, 0, 0)を向くようになっている．

```
SpotLight(光源色=0xffffff, 光強度=1, 最大照射距離=0, 照射角, ボケ具合=0, 光の減衰率)
```

(4) 矩形光源

長方形の面に均一に光が放射されるような光源を矩形光源という．明るい窓やストリップライトのようなものを表現するときに使用できる．

```
RectAreaLight(光源色=0xffffff, 光強度=1, 横幅=10, 縦幅=10)
```

(5) 環境光

空気による散乱や周囲の物体との相互反射によって生じる間接光源を環境光という．3D 空間全体に光が当たるため，この光源だけでは陰影や影による立体感を表現することはできない．

```
AmbientLight(光源色=0xffffff, 光強度=1)
```

(6) 半球光源

半球光源はより現実に近い環境光のようなものであり，上からの太陽光と地面からの反射光を分けて設定することができる．

```
HemisphereLight(上からの光源色=0xffffff,下からの光源色=0xffffff, 光強度=1)
```

また，Three.js で影を付ける場合は，以下の 4 つの設定を行う．

・レンダラーで影を有効にする

```
renderer.shadowMap.enabled = true;
```

・光源の影を有効にする

```
light.castShadow = true;
```

・影を落とすオブジェクトを設定する

```
object.castShadow = true;
```

・影を受けるオブジェクトを設定する

```
receivedObject.receiveShadow = true;
```

課題 4　sample3.js は sample2.js に床面となるオブジェクトを追加したものである．また，視点を (0, 20, -40) から原点 (0, 0, 0) を見下ろすように変更している．sample3.js を基に，平行光源を以下のような光源に変更したプログラム kadai4.js を作成せよ．

・スポット光源と環境光を使用

・スポット光源の位置は (0, 20, 0)，照射角は 30 度，最大照射距離は 100，減衰率は 2

・環境光の光強度は 0.5

```
1  /* sample3.js */
2  window.addEventListener("DOMContentLoaded", init);
3
4  function init() {
5      const width = 500;
6      const height = 500;
7
8      // レンダラーを作成
9      const renderer = new THREE.WebGLRenderer({
10         canvas: document.querySelector("#myCanvas")
11     });
12     renderer.setSize(width, height); /* ウィンドウサイズの設定 */
13     renderer.setClearColor(0x000000); /* 背景色の設定 */
14
15
16     // シーンを作成
17     const scene = new THREE.Scene();
18
19     // カメラを作成
20     const camera = new THREE.PerspectiveCamera(45, width / height);
21     camera.position.set(0, 20, -40);
22     camera.lookAt(new THREE.Vector3(0,0,0));
23
24     const boxGeometry = new THREE.BoxGeometry(10,10,10);
25     const boxMaterial = new THREE.MeshStandardMaterial({
26         color: 0xffffff
27     });
28     const box = new THREE.Mesh(boxGeometry, boxMaterial);
29
30     scene.add(box);
31
32     const planeGeometry = new THREE.PlaneGeometry(50,50);
33     const planeMaterial = new THREE.MeshStandardMaterial({
```

```
34          color: 0xaaaaaa
35      });
36      const plane = new THREE.Mesh(planeGeometry,planeMaterial);
37      plane.position.set(0,-5,0);
38      plane.rotateX(-Math.PI/2,0,0);
39
40      scene.add(plane);
41
42      //光源設定
43
44      // 平行光源
45      const directionalLight = new THREE.DirectionalLight(0xffffff,1);
46      directionalLight.position.set(1, 1, 1);
47
48      // シーンに追加
49      scene.add(directionalLight);
50
51      // 初回実行
52      renderer.render(scene, camera);
    }
```

3.5. 座標系と投影法

3.5.1. ワールド座標系とスクリーン座標系

コンピュータ内部の 3 次元座標系をワールド座標系，ディスプレイ上の 2 次元平面の座標系をスクリーン座標系と呼ぶ（**図 3**）.

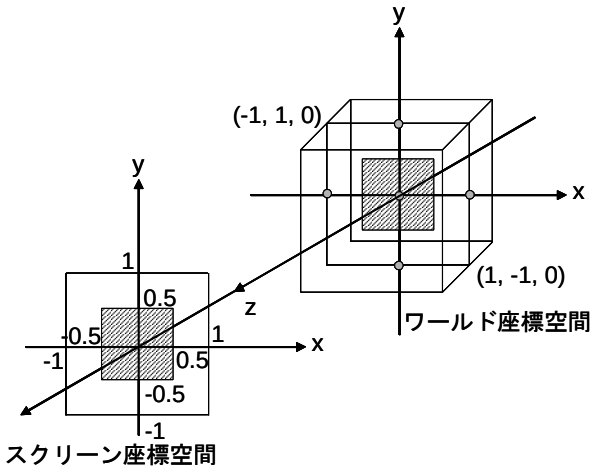

図 3 ワールド座標系とスクリーン座標系

●投影法

ワールド座標系の 3 次元物体をスクリーン座標系に映すことを投影と呼ぶ. 投影法は多数あるが CG では**図 4** に示す平行投影と透視投影が主に用いられる.

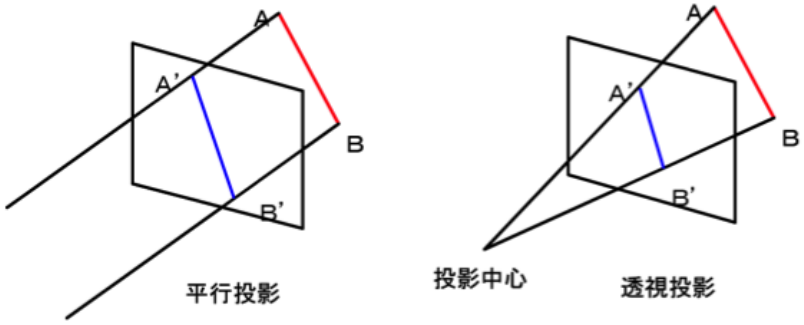

図 4　平行投影と透視投影

平行投影では，投影線が平行であるために，投影面を垂直に取ると物体の寸法を正確に表すことができる．しかし，平行投影では遠くの物体も近くの物体も同じ寸法に表示されてしまう．

透視投影では，投影点が視点（投影中心）に収束するように投影される．我々の視覚系と同様に遠近感が得られるため，現実に近い画像生成が可能である．

●投影法とビューボリューム

一般に CG では，変換の対象となる 3 次元空間を限定することによって，計算の効率化を図る．まず，投影面にウィンドウを設定し，視点から見える視野の範囲を限定する．このウィンドウ外の物体の投影を除く作業をクリッピングという．このような作業を経て，投影によりウィンドウに表示される領域をビューボリュームと呼ぶ．平行投影と透視投影におけるビューボリュームを**図 5**，**図 6** に示す．Three.js では，以下の関数を用いて投影の方法と座標系を指定する．ビューボリュームの外に描かれた物体は表示されないので注意すること.

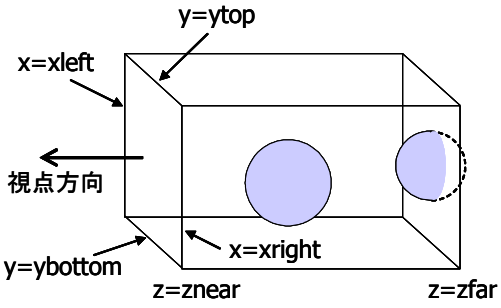

図 5　平行投影のビューボリューム

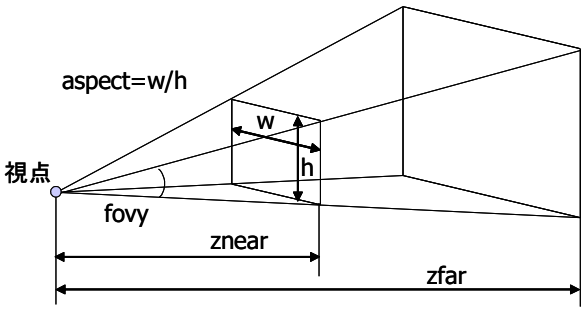

図 6　透視投影のビューボリューム

(1) 平行投影

```
const camera = new THREE.OrthographicCamera(left, right, top, bottom, near, far);
```

引数は，ビューボリュームの左右，上下，前後の各面を表す.

(2) 透視投影

```
const camera = new THREE.PerspectiveCamera(fovy, aspect, near, far);
```

fovy は，ビューボリュームの上下の開き角（°），aspect は断面の幅／高さの比，near と far は，視点から頂面までと底面までの距離である．

(3) 視点位置

```
camera.position.set(ex, ey, ez);
```

camera オブジェクトの position プロパティに数値を代入する．3 つの引数 ex，ey，ez を視点の位置として設定することができる．

(4) 視線方向

```
camera.lookAt(new THREE.Vector3(cx, cy, cz));
```

camera オブジェクトの lookAt()メソッドを使って視線方向を指定する．lookAt()メソッドはどの位置からでも指定した座標に強制的に向かせることが可能である．3 つの引数 cx，cy，cz を目標の位置として設定することができる．

課題 5　sample4.js は，sample3.js を基に，ビューボリュームが-30<x<30，-30<y<30，-100<z<100 の平行投影となるように設定したものである．sample4.js を実行しなさい．また，sample4.js を以下のような**透視投影**となるように変更したプログラム kadai5.js を作成せよ．

・ビューボリューム：上下の開き角 fovy=45.0，aspect=1.0，near=1.0，far=1000

・視点の位置：(0, 20, -40) から (0.0, 0.0, 0.0) の方向を見る

```
1  /* sample4.js */
2  window.addEventListener("DOMContentLoaded", init);
3
4  function init() {
5      const width = 500;
6      const height = 500;
7
8      // レンダラーを作成
9      const renderer = new THREE.WebGLRenderer({
10         canvas: document.querySelector("#myCanvas")
11     });
12     renderer.setSize(width, height); /* ウィンドウサイズの設定 */
13     renderer.setClearColor(0x000000); /* 背景色の設定 */
14
15     // シーンを作成
16     const scene = new THREE.Scene();
17
18     // カメラを作成
```

```
19      const camera = new THREE.OrthographicCamera(-30, 30, 30, -30, -100, 100);
20      camera.position.set(0, 20, -40);
21      camera.lookAt(new THREE.Vector3(0,0,0));

～    /*■sample3.js と同じ■*/

53  }
```

3.5.2. 座標変換（回転・移動）

幾何変換とは，ワールド座標空間において，物体の位置や姿勢を変えたり，拡大・縮小の変形を加えたりする操作を指している．Three.js（および多くの CG ライブラリ）では，関数を利用して，各オブジェクト座標系の変換が実行される．

(1) 平行移動

```
オブジェクト名.translateA(t);
```

A は移動軸(X or Y or Z)，t は各軸方向への移動量.

> XYZ まとめて行う関数は
> .translateOnAxis(new THREE.Vector3(tx,ty,tz),l);
> tx，ty，tz は移動方向を示すベクトル．l は移動距離.

(2) 回転

```
オブジェクト名.rotateA(r);
```

A は回転軸(X or Y or Z)，r は回転角（radian）.

> XYZ まとめて行う関数は
> .setRotationFromAxisAngle(new THREE.Vector3(x,y,z), θ);
> θは回転角度 (radian), rx，ry，rzは回転軸のベクトル.

(3) 拡大・縮小

```
オブジェクト名.scale.set(sx, sy, sz);
```

sx，sy，sz は各軸方向への拡大量．マイナスの値で座標系が反転する．また，set を x や y や z に置き換えることで指定した軸のみ拡大・縮小させることが可能となる.

> .translateA(t) や .rotateA(r) のような相対値で移動させる関数とは違い，
> .scale.set(sx,sy,sz) は絶対値を指定する関数であることに注意.
> 座標や回転角も同じように.position.set(x,y,z), .rotation.set(x,y,z) で絶対値を指定できる.

課題 6　kadai5.js を基にオブジェクトを y 軸周りに 70°回転させてから x 軸方向に 10 平行移動するように変更した kadai6.js を作成しなさい．変換の順序に気を付けること.

```
kadai5.js を改変すること
```

3.5.3. アニメーション

アニメーションは細かく数値を変更しながら描画を繰り返すことで再現される．アニメーションのための処理を行う関数（今回は animate() ）を用意し，requestAnimationFrame() を用いて繰り返しその関数を呼ぶ．また，アニメーション処理の最後には必ず再描画（Three.js では render() ）を行う．

課題 7　sample5.js は sample4.js に回転するアニメーションを追加したものである．このプログラムを実行しなさい．また，これを参考に回転しながら y 軸上方向に平行移動するようにプログラムを変更し，kadai7.js を作成しなさい．

```
1  /* sample5.js */
2  window.addEventListener("DOMContentLoaded", init);
3
4  function init() {
5      const width = 500;
6      const height = 500;

～     /*■kadai6.js と同じ■*/

18     // カメラを作成
19     const camera = new THREE.PerspectiveCamera(30, width / height, 1.0, 1500);
20     camera.position.set(0, 20, -40);
21     camera.lookAt(new THREE.Vector3(0,0,0));

～     /*■kadai6.js と同じ■*/

35     // アニメーション処理
36     function animate() {
37       let requestId = requestAnimationFrame(animate);
38       box.rotation.y += 0.1;
39       render();
40     }
41   /*■kadai6.js と同じ■*/
42
43     // 初回実行
44     let render = function () { renderer.render(scene, camera); };
       render();
～
       // アニメーション開始
54     animate();
55 }
```

3.5.4. インタラクション（キーボード，マウス）

Three.js はイベント駆動型のプログラムであり，キーボードからの入力や，マウスによる入力など，入力されたイベントに応じてあらかじめ登録されたコールバック関数が実行される．

●キーボード入力のプログラム

キーボードからの入力を利用するためには，まず document.addEventListener(“キーボードイベント名”，コールバック関数名)を実行し，キーボード機能を呼び出すコールバック関数を指定する必要がある．こうすることで，キーボードからの入力があった場合に，押されたキーの種類が取得されコールバック関数に渡される．

課題 8　sample6.js は，キーボードで“ q ”と打つと，ウィンドウが閉じるように sample4.js のプログラムを変更したものである．このプログラムを実行しなさい．次に，プログラムを実行すると，(1) キーボードで“r”と打つたびに，物体が y 軸中心に 30 度ずつ回転し，(2)“c”と打つたびに色が白→赤→緑→青→白に変わる，ようにプログラムを変更して kadai8.js を作成しなさい．

- keyCode は“r”が [82],“c”が [67]（いずれも **onkeydown イベント時の keyCode**）
- 物体の色を変えるには，オブジェクト名.material.color.set(カラー); を使用すればよい．
- 白は (0xffffff), 赤は (0xff0000), 緑は (0x00ff00), 青は (0x0000ff) である．

```
1  /* sample6.js */
2  window.addEventListener("DOMContentLoaded", init);
3
4  function init() {
5      const width = 500;
6      const height = 500;

～     /*■kadai6.js と同じ■*/

18     // カメラを作成
19     const camera = new THREE.PerspectiveCamera(30, width / height, 1.0, 1500);
20     camera.position.set(0, 20, -40);
21     camera.lookAt(new THREE.Vector3(0,0,0));

～     /*■kadai6.js と同じ■*/

31     document.addEventListener("keydown", onDocumentKeyDown, false);
32     function onDocumentKeyDown(event_k) {
33         let keyCode = event_k.which;
34         // q: ページを閉じる
35         if (keyCode == 81) {
36             window.close();
37         }
```

```
38
39          render();
40      }

～    /*■kadai6.js と同じ■*/

49      // 初回実行
50      let render = function () { renderer.render(scene, camera); };
51      render();
52  }
```

●マウス入力のプログラム

マウス操作に関するイベント処理は，ボタンのオン・オフ（クリック）とマウスの移動（ドラッグ）の 2 種類の設定が可能であるが，ここでは，クリックについてのみ説明する．マウスボタンのクリックによる入力を利用するためには，document.addEventListener(“マウスイベント名”，コールバック関数名) を実行し，マウス機能を呼び出すコールバック関数を指定する必要がある．

課題 9　sample7.js は，ウィンドウ内をマウスで左クリックすると，物体が y 軸中心に一回転するように kadai8.js を変更したものである．このプログラムを実行しなさい．次に，ウィンドウ内をマウスで**左クリックすると回転が始まり，右クリックすると回転が止まる**ようにプログラムを変更した kadai9.js を作成しなさい．(右クリック機能がない場合は，クリックすると回転が始まり，もう一度クリックすると回転が止まるようにプログラムしなさい)

```
1   /* sample7.js */
2   window.addEventListener("DOMContentLoaded", init);
3
4   function init() {
5       const width = 500;
6       const height = 500;
7       let rot = 30;
8       let count_color = 1;
9       let flag_rotation = 0;
10      let count_rotation = 0;

～    /*■kadai8.js と同じ■*/

69      document.addEventListener('mousedown', onDocumentMouseDown, false);
70      function onDocumentMouseDown(event_m) {
71          switch (event.button) {
72              // 左クリック
```

```
            case 0:
                flag_rotation = 1;
                animate();
                break;
            // ホイール
            case 1: break;
            // 右クリック
            case 2: break;
        }
    }

    function animate() {
        if (count_rotation >= 2 * Math.PI) {
            count_rotation = 0;
            flag_rotation = 0;
        }
        else if (flag_rotation == 1) {
            let requestId = requestAnimationFrame(animate);
            box.rotation.y += 0.1;
            count_rotation += 0.1;
            render();
        }
    }

～  /*■kadai8.js 同じ■*/

}
```

4. ロボットを作る課題

Three.js では複数のオブジェクトをグループ化することができる．グループ化を行うと，そのグループ全体を 1 つのオブジェクトとして扱うことができる．これは複数のオブジェクトを同時に移動させる場合や，オブジェクトをいくつかのまとまりに分けたい場合に役立つ．グループ化を利用する際は以下のように定義する．

```
const group = new THREE.Group();
group.add(オブジェクト 1，オブジェクト 2，...);
```

課題 10　sample8.js はロボットの頭を表示するプログラムである．このプログラムを改変してロボットの頭に耳を追加し，パーツをすべてグループ化したプログラム kadai10.js を作成しなさい．耳に使用するジオメトリは任意のものを使用してよい．また，顔をより複雑に改変しても構わない．

```
/* sample8.js */
window.addEventListener("DOMContentLoaded", init);

function init() {
    const width = 500;
    const height = 500;

    // レンダラーを作成
    const renderer = new THREE.WebGLRenderer({
        canvas: document.querySelector("#myCanvas")
    });
    renderer.setSize(width, height); /* ウィンドウサイズの設定 */
    renderer.setClearColor(0x000000); /* 背景色の設定 */

    // シーンを作成
    const scene = new THREE.Scene();

    // カメラを作成
    const camera = new THREE.PerspectiveCamera(45, width / height);
    camera.position.set(0, 0, -70);
    camera.lookAt(new THREE.Vector3(0,0,0));

    const bodyMat = new THREE.MeshStandardMaterial({
        color: 0xaaaaaa
    });
    const highlight = new THREE.MeshStandardMaterial({
        color: 0x4444ff
    });


    const head = new THREE.Mesh(new THREE.BoxGeometry(20,16,16),bodyMat);
    scene.add(head);

    const eye1 = new THREE.Mesh(new THREE.SphereGeometry(1.5,16,12),highlight);
    eye1.position.set(5,3,-8);
    scene.add(eye1);

    const eye2 = new THREE.Mesh(new THREE.SphereGeometry(1.5,16,12),highlight);
    eye2.position.set(-5,3,-8);
```

```
40      scene.add(eye2);
41
42      const mouse= new THREE.Mesh(new THREE.CylinderGeometry(2,2,1,3),highlight);
43      mouse.position.set(0,-3,-8);
44      mouse.rotation.set(Math.PI/2,Math.PI,0);
45      scene.add(mouse);
46
47
48      //光源設定
49
50      // 平行光源
51      const directionalLight = new THREE.DirectionalLight(0xffffff,1);
52      directionalLight.position.set(0, 0, 1);
53      // シーンに追加
54      scene.add(directionalLight);
55
56      const ambientLight = new THREE.AmbientLight(0xffffff,0.8);
57      scene.add(ambientLight);
58
59      // 初回実行
60      let render = function () { renderer.render(scene, camera); };
61      render();

    }
```

課題 11　頭（課題 10 で作成したものでよい），胴体，腕 2 本，脚 2 本からなるロボットを作り，いずれかパーツに対して色を設定せよ．各パーツをどのジオメトリで表現するかは各自の選択に任せる．手足を複数のジオメトリの組み合わせで表現してもよい．上記機能を持つプログラム kadai11.js を作成せよ．手足は各 2 本以上でもよい．

課題 12　キーボードから“h”を入力すると，ロボットの頭が y 軸を中心に 1 回転するプログラム kadai12.js を作成せよ．

課題 13　キーボードから“j”を入力すると，ロボットがジャンプするようなプログラム kadai13.js を作成せよ．ジャンプ動作をどのようにモデル化したかについて説明し，その妥当性について考察せよ．

課題 14　マウスを操作してロボットを歩かせる，あるいは移動させるプログラム kadai14.js を作成せよ．マウスを使ったロボット移動操作機能をどのような意図に基づいて，どのようにモデル化し実現したか，どのような操作インタフェースとしたか，および，それらの根拠について説明せよ．「特に理由なし」などは無記入とみなす．また，結果の妥当性について考察せよ．

課題 15　1 つ以上の新規パーツを任意に追加するとともに，1 つ以上の新規対話操作機能をロボットに付加する．対話操作機能は新規パーツに適用してもよいし，ロボット本体に適用してもよい．上記機能を持つプログラム kadai15.js を作成せよ．機能内容と操作インタフェースについて詳細に説明し，その目的や意図を述べよ．「特に理由なし」などは無記入とみなす．また，結果の妥当性について考察せよ．なお，例えば「目玉が飛び出す」あるいは「バラバラになる」のような創意工夫のないものは，概ね低い評価とせざるを得ない．本課題は，実世界情報実験２「CG」の集大成であるので，労力を惜しまないこと．

5. 参考資料

[1] https://threejs.org/

[2] https://threejs.org/docs/#manual/en/introduction/Browser-support

[3] https://caniuse.com/webgl

[4] https://ics.media/entry/14771/

[5] https://ics.media/tutorial-three/

[6] https://ja.wikipedia.org/wiki/Three.js

[7] https://ja.wikipedia.org/wiki/WebGL