HANDSON @MOZILLA JAPAN

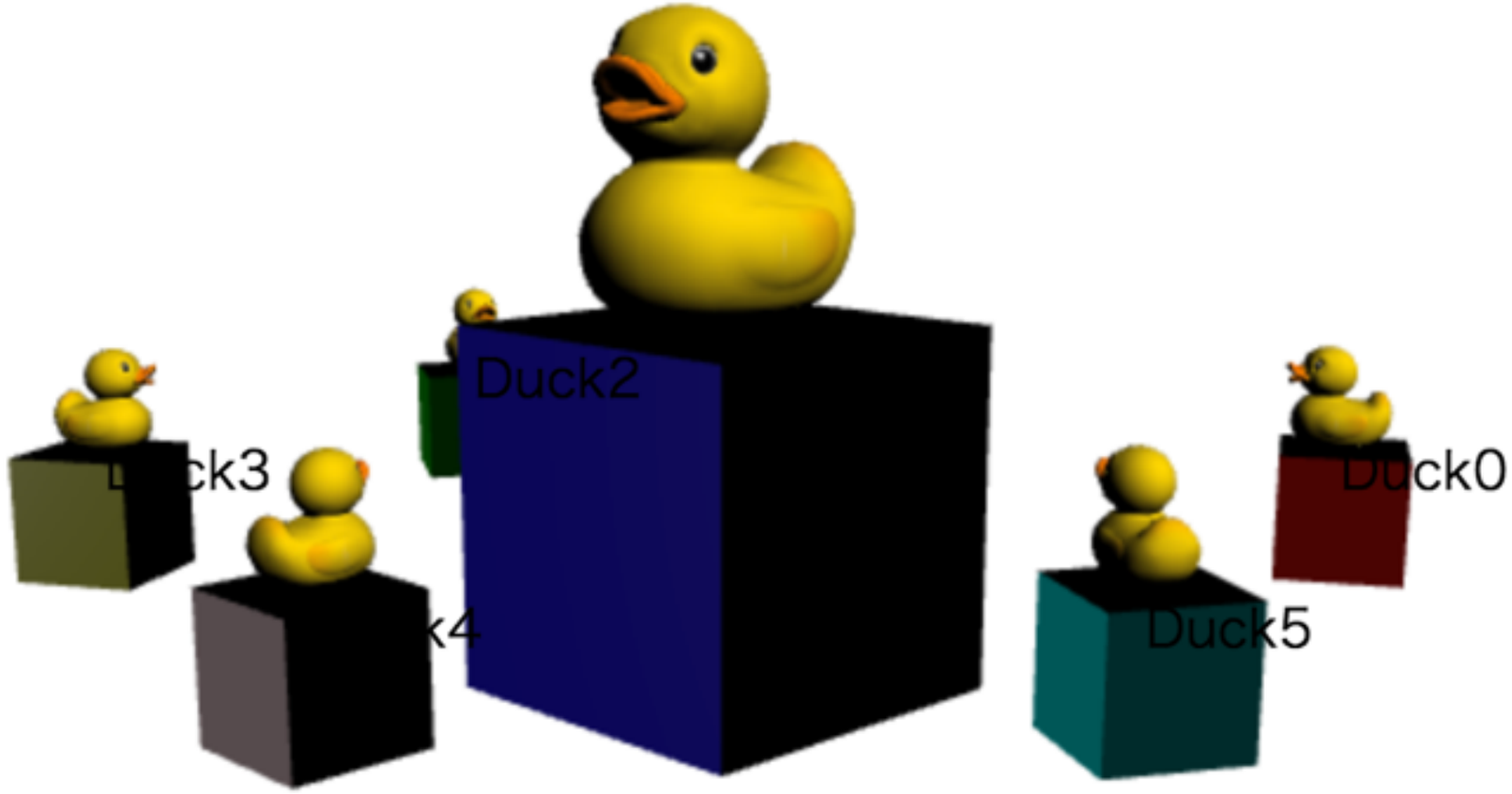
GRIMOIRE.JS SESSION3

自己紹介

- 秋澤 一史
- twitter: @pn1y
- プロジェクトではWeb周りを担当しています。

なにをするか

- 3Dモデルビューワ(.gltf)
- クリックしたら拡大する



なにをするか

- 画像ビューワ
- サムネイルが拡大

3D

- 3Dモデルビューワ(.gltf)
- クリックしたら拡大する

シーンを作る

index.goml

```
<goml height="512" width="512">
  <scene>
    <camera>
      <camera.components>
        <MouseCameraControl></MouseCameraControl>
      </camera.components>
    </camera>
    <light type="directional" rotation="30,30,30"></light>
    <object id="circle"> ...
    </object>
  </scene>
</goml>
```

タグの再確認

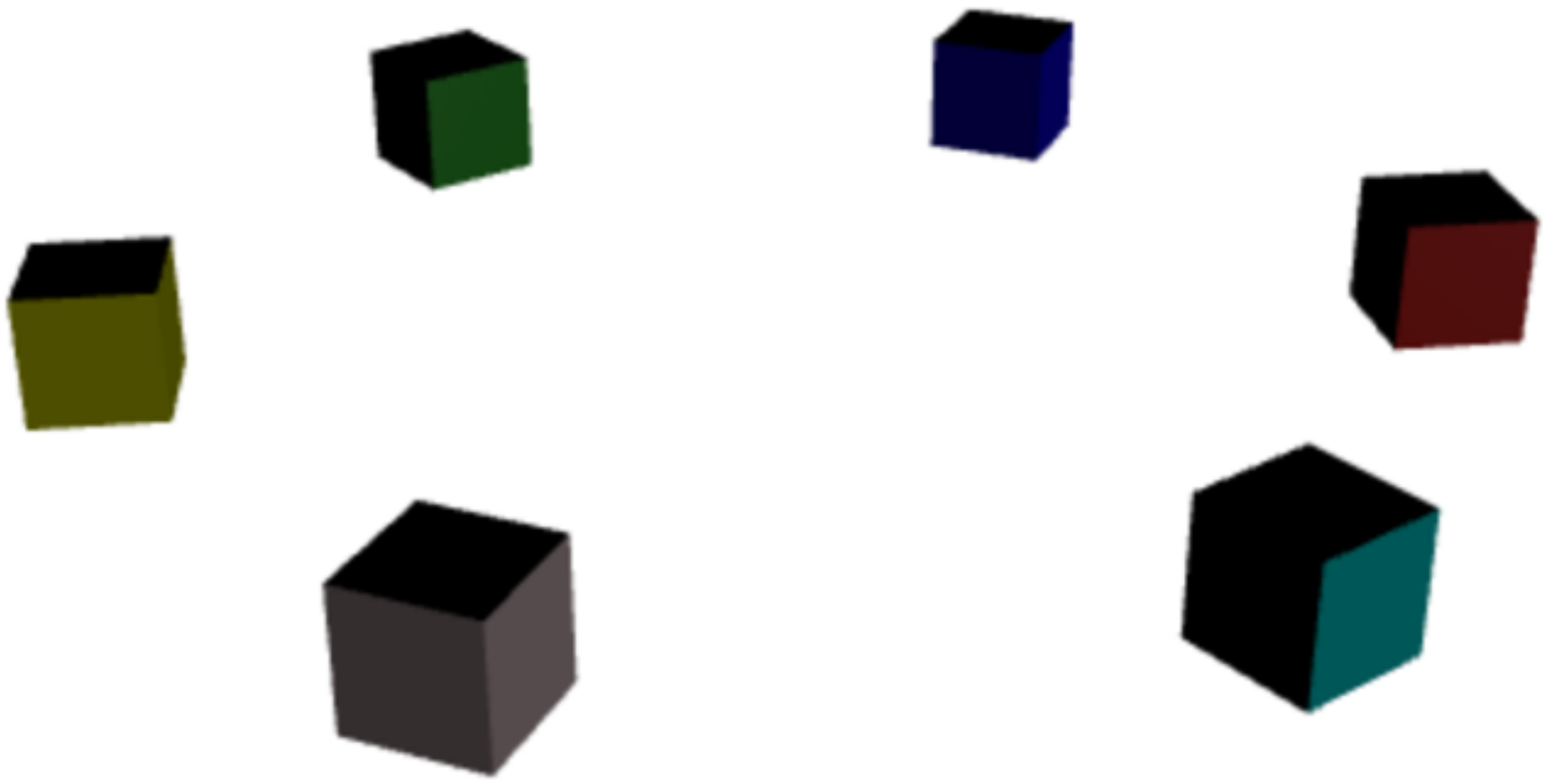
- <camera>
  - カメラの設置
- <MouseCameraControl>
  - マウスでカメラを操作可能にする
- <light type="directional">
  - 並行光源を設置 (rotationで方向を変えられる)

タグの再確認

- <mesh geometry=“cube">
  - メッシュの設置 (動作確認のために一旦)

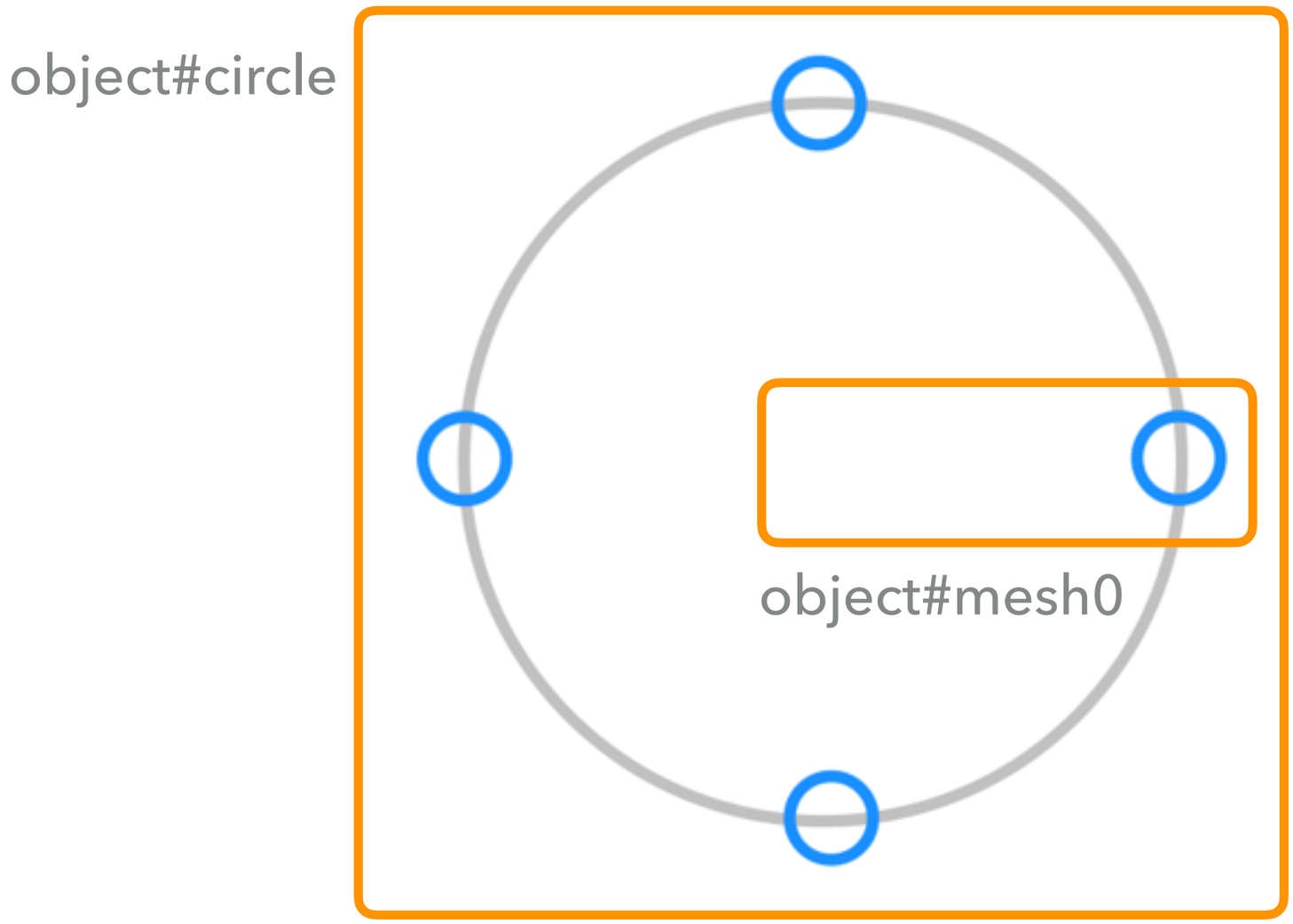
円形状にメッシュを設置する

- 相対座標をうまく使ってmeshの配置をする
  - CSSでのposition: relative; position:absolute;のイメージ



円形状にメッシュを設置する

- <object>
  - 何も描画しない空のオブジェクト(相対座標を作る)



円形状にメッシュを設置する

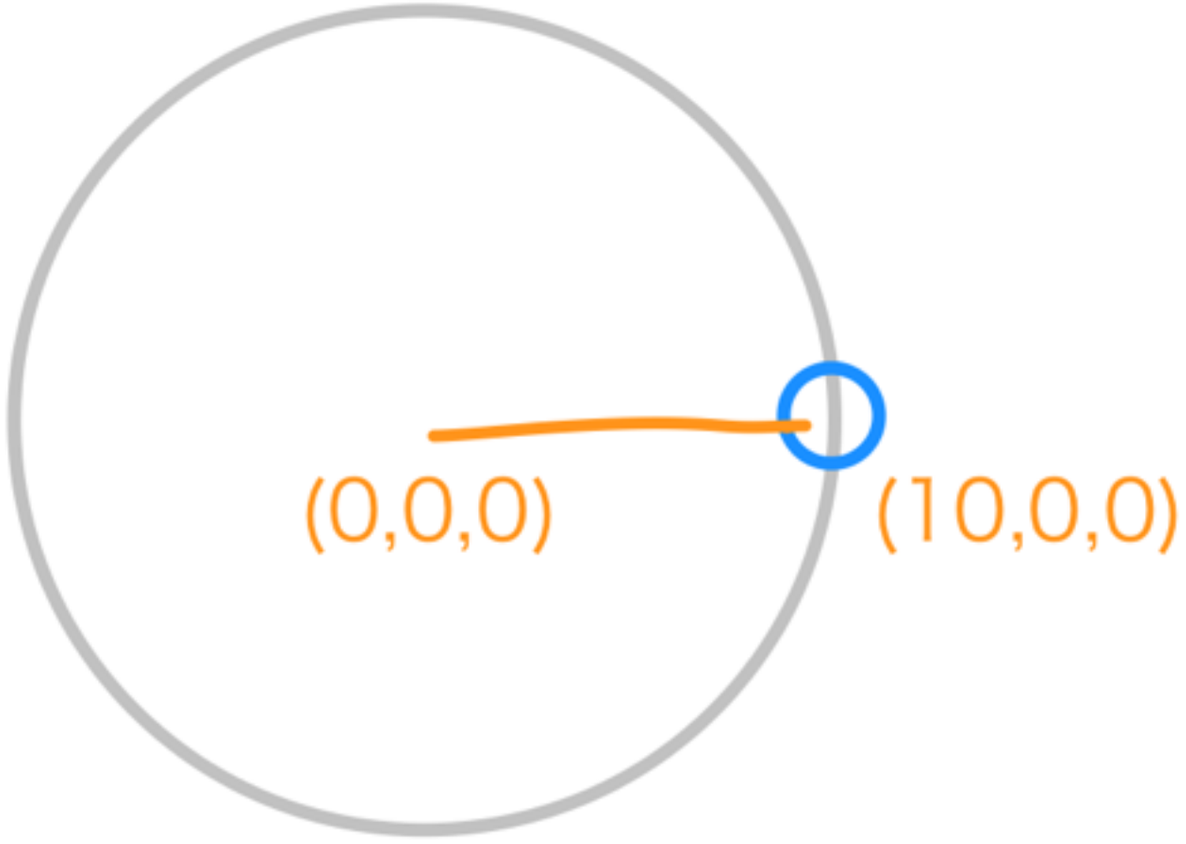
- <object>を複数ネストさせることで座標を切り替えていく

index.goml

```
<object id="circle">
  <object id="mesh0" rotation="y(0)">
    <mesh position="10,0,0" geometry="cube" albedo="red">...
    </mesh>
  </object>
  <object id="mesh1" rotation="y(60)">
    <mesh position="10,0,0" geometry="cube" albedo="blue">...
    </mesh>
  </object>
  <object id="mesh2" rotation="y(120)">
    <mesh position="10,0,0" geometry="cube" albedo="green">...
    </mesh>
```

円形状にメッシュを設置する

- <mesh position=“10,0,0”>　x軸はカメラから見て水平右方向
  - 中心から親の座標に対してx方向へ10

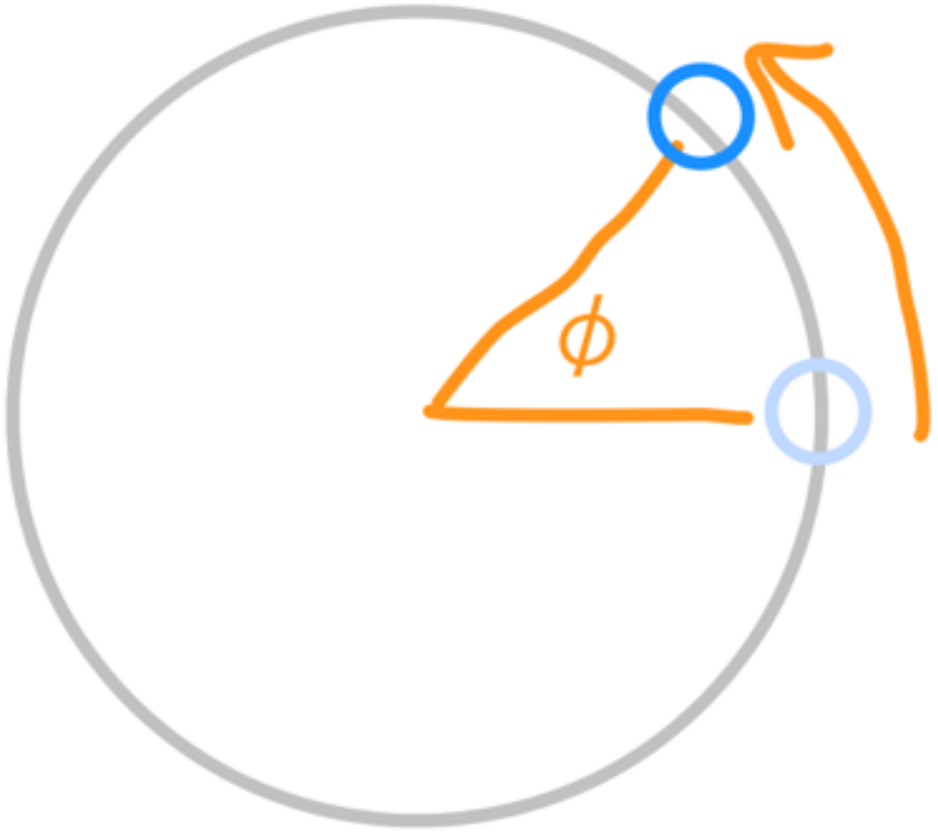


円形状にメッシュを設置する

- <object rotation="y(60)">

xz平面で回転させたいのでy軸中心に
y軸はカメラから見て上方向

  - <mesh>の親の<object>をφ回転させることで円形状に



メッシュを回転させる

- Rotateコンポーネントの利用(セッション2のもの)

index.js

```
const {Vector3} = gr.lib.math;

gr.registerComponent('Rotate', {
  attributes: {
    speed: {
      default: '0,0,0',
      converter: 'Vector3',
    },
  },
  $mount: function() {
    this.phi = Vector3.Zero;
  },
  $update: function() {
    this.phi = this.phi.addWith(this.getAttribute('speed'));
    this.node.setAttribute('rotation', this.phi.X + ',' + this.phi.Y + ',' + this.phi.Z);
  },
});
```

メッシュを回転させる

- registerComponent(‘Rotate’, {...}) で<Rotate>登録

index.js

```
const {Vector3} = gr.lib.math;

gr.registerComponent('Rotate', {
  attributes: {
    speed: {
      default: '0,0,0',
      converter: 'Vector3',
    },
  },
  $mount: function() {
    this.phi = Vector3.Zero;
  },
  $update: function() {
    this.phi = this.phi.addWith(this.getAttribute('speed'));
    this.node.setAttribute('rotation', this.phi.X + ',' + this.phi.Y + ',' + this.phi.Z);
  },
});
```

タグで<Rotate speed=“1,1,1”>のように使えるようにする

初期化時にthis.phiをVector3(0,0,0)

毎フレーム this.phi に speed を足して、rotationを更新する

メッシュを回転させる

- <Rotate>の使い方
  - 毎フレーム増加するeuler角の量
- カメラから見て水平に回転させたい場合(上から見て円を描くように)はy軸中心にφ度ずつという具合

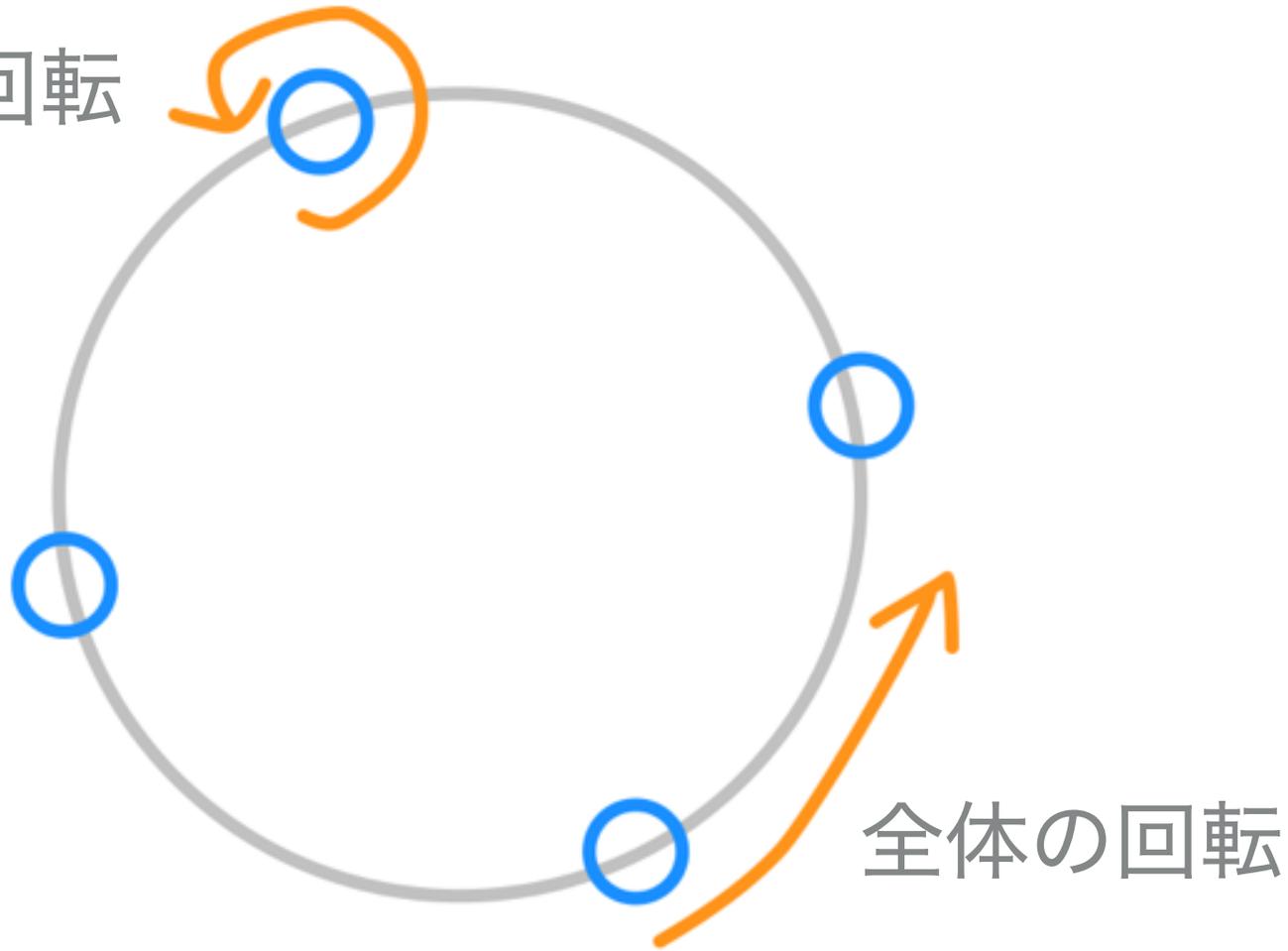
index.goml

```
<Rotate speed="0,0.5,0"></Rotate>
```

この場合はy軸回りに0.5度ずつ毎フレーム回す

メッシュを回転させる

- <Rotate>を使って座標系を回転させる



全体を回転させる

- <object id="circle">は全てのmeshを内包する
  - <object.components>の中に<Rotate>を置いて回転

index.goml

```
<object id="circle">
  <object.components>
    <Rotate speed="0,0.1,0"></Rotate>
  </object.components>
  <object rotation="y(0)"> ...
  </object>
  <object rotation="y(60)"> ...
  </object>
  <object rotation="y(120)"> ...
```

メッシュ単体を回転させる

- <mesh>を回転させればOK
  - いままでと同様に<Rotate>を設置

index.goml

```
<object rotation="y(0)">
  <mesh position="10,0,0" geometry="cube" albedo="red">
    <mesh.components>
      <Rotate speed="0,0.5,0"></Rotate>
    </mesh.components>
  </mesh>
</object>
```

メッシュにHTMLの要素を追従させる

- <HTMLBinder>コンポーネントを利用する
  - htmlQuery属性で指定したクエリをhtmlから取ってきて要素に追従させることができる

index.goml

```
<object rotation="y(0)">
  <mesh position="10,0,0" geometry="cube" albedo="red">
    <mesh.components>
      <Rotate speed="0,0.5,0"></Rotate>
      <HTMLBinder htmlQuery="#bound0"/>
    </mesh.components>
  </mesh>
</object>
```

index.html

```
<div id="bound0">0</div>
```

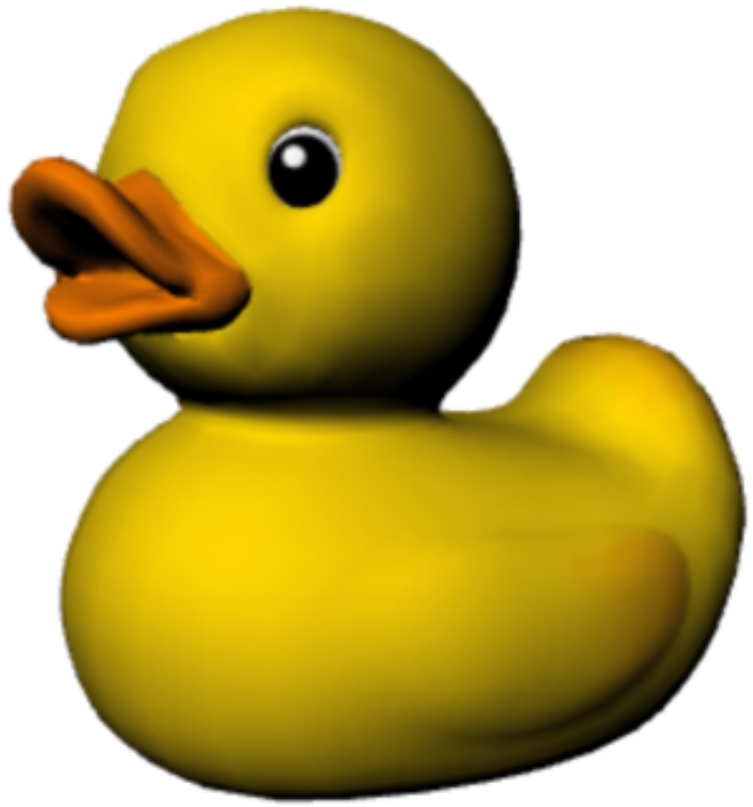
モデルの表示

- grimoire-gltfプラグインを使うことで
  <model>タグが使えるようになる

index.html

```
<script src="https://unpkg.com/grimoirejs-gltf/register/grimoire-gltf.js"></scri
```

- gltfというモデルフォーマットが読み込
- <model src=“path/to/model.gltf” />
  - srcにモデルのパスを指定するだけ



モデルの表示

- <model>タグを追加
  - <mesh>と相対座標に表示したいので、<mesh>の中にいれる。(cubeと一緒に回転する)

index.goml

```
<object rotation="y(0)">
  <mesh position="10,0,0" geometry="cube" albedo="red">
    <mesh.components>
      <Rotate speed="0,0.5,0"></Rotate>
      <HTMLBinder htmlQuery="#bound0"/>
    </mesh.components>
    <model src="Duck/Duck.gltf" position="0,1,0" />
  </mesh>
```

ラベルがクリックされたときにMESHを拡大

- <HTMLBinder>で追従させる要素にいつも通りイベントを追加する

index.js

```
document.addEventListener('DOMContentLoaded', () => {
  Array.from(document.querySelectorAll('.bound')).forEach((elm, i) => {
    elm.addEventListener('click', (e) => {
```

- DOM読み込み後、.boundを取ってきてクリックイベントを追加する

index.html

```
<div class="bound" id="bound0">Duck0</div>
<div class="bound" id="bound1">Duck1</div>
<div class="bound" id="bound2">Duck2</div>
<div class="bound" id="bound3">Duck3</div>
```

ラベルがクリックされたときにMESHを拡大

- クエリを使ってcanvas内のオブジェクトを取得の再確認
  - gr(“#canvas”)(“#mesh0”)
- setAttributeを利用して属性値の変更
  - gr(“#canvas”)(“#mesh0”).setAttribute(“scale”, “2,2,2”);

index.goml

```
<object id="mesh0" rotation="y(0)">
```

index.html

```
<script id="canvas" type="text/goml" src="index.goml"></script>
```

ラベルがクリックされたときにMESHを拡大

- 拡大されているオブジェクトを元に戻す
- 選択されたオブジェクトを拡大する

index.js

```
Array.from(document.querySelectorAll('.bound')).forEach((elm, i) => {
  elm.addEventListener('click', (e) => {
    const cvs = gr('#canvas');
    cvs('.big mesh').setAttribute('position', '10,0,0');
    cvs('.big').setAttribute('scale', '1,1,1');
    cvs('.big').setAttribute('class', '');
    cvs(`#mesh${i} mesh`).setAttribute('position', '0,0,0');
    cvs(`#mesh${i}`).setAttribute('scale', '3,3,3');
    cvs(`#mesh${i}`).setAttribute('class', 'big');
  });
});
```

ラベルがクリックされたときにMESHを拡大

- cvs(`#mesh${i} mesh`).setAttribute('position', '0,0,0');
  - 選択されたオブジェクトを中心に
- cvs(`#mesh${i}`).setAttribute('scale', '3,3,3');
  - x,y,zともに3倍に拡大
- cvs(`#mesh${i}`).setAttribute('class', 'big');
  - 拡大したオブジェクトにclass属性を付与(拡大をリセットするときに利用)

それから

- 今回はデザインは無しで、ロジックのみの説明になりましたが、テクスチャやモデル、CSSなどを弄っていい感じになるともっと良いですね。
- HTMLBinderでリンクや画像をつけたりもできますので、クリックしたらanchorに飛ばしたりするのも面白いかもしれません。
- みなさんもWebGLエンジョイしましょう！