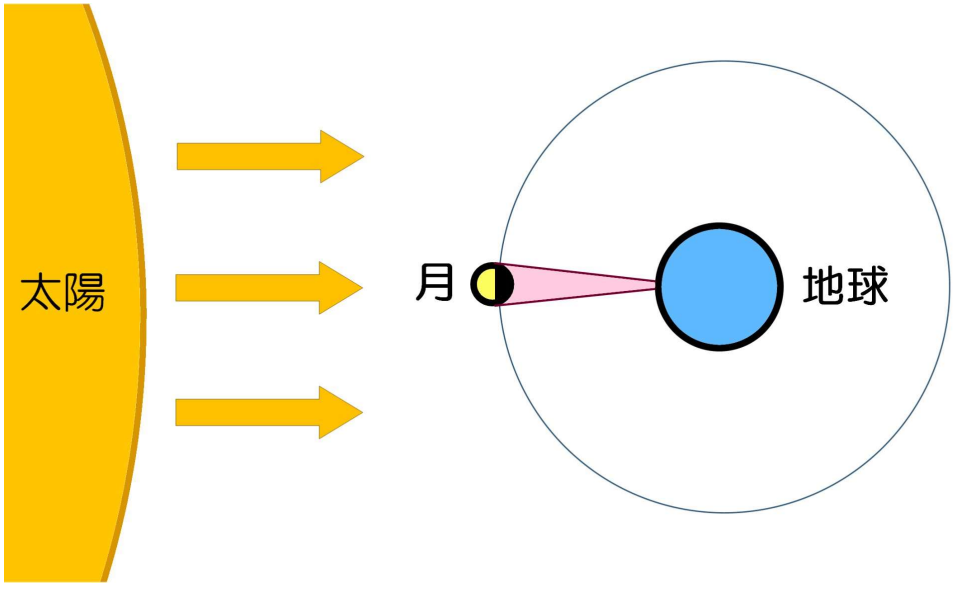
第 6 章

皆既日食を 3DCG で表現する

この章では、Babylon.js で日食の仕組みを表示し教材などに活用できないか検討します。

6.1　はじめに

みなさんは日食を見たことがありますか。学生の頃、理科の教科書で下図のような説明を見たことがないでしょうか。



▲図 6.1　日食の説明図

この図ですと皆既日食は新月の度、見るチャンスがありそうにも見えますが実際はそうではないですよね？ 今回は Babylon.js を利用してこの図を３Ｄで表示し、なぜ毎月皆既日食が見られないのかを解りやすく理解する教材として活用できないか試してみます。

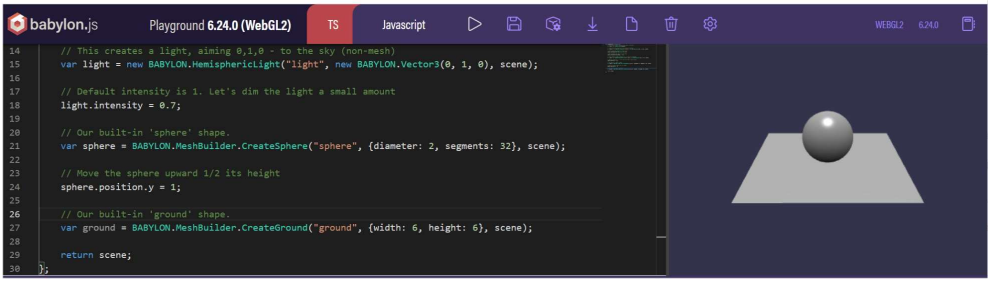
6.2　皆既日食を３Ｄで再現する

6.2.1　太陽と地球を作る

宇宙は広大です。現実の縮尺に合わせて作成すると地球と太陽を並べると地球が小さくなりすぎてしまいますので、最初は見やすいサイズで作ります。

Babylon.js Playground を起動して進めていきます。

初期状態で　下図のように球体と地面が表示されますので、次のように変更します。



▲図 6.2　Playground 初期状態

1. 地面（Ground）は不要なので削除する。
2. CreateSphere で太陽と地球にする球体を２つ作成する。
3. 太陽にする球体は半径を２に大きくする。

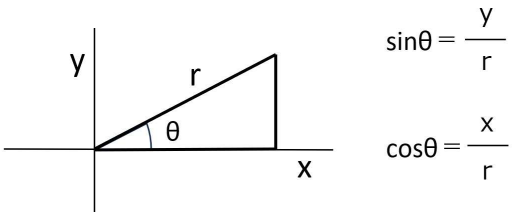
以下が Playground の初期状態から、書き直した部分のソース部分です。（camera と light の定義を除く）

▼　太陽と地球を表示する（JavaScript）

```
1:     var sun = BABYLON.MeshBuilder.
2:         CreateSphere("sun", {diameter: 2, segments: 32}, scene);
3:     var earth = BABYLON.MeshBuilder.
4:         CreateSphere("earth", {diameter: 1, segments: 32}, scene);
5:     earth.position.x =  1;
```

6.2.2　地球の公転を三角関数で実装する

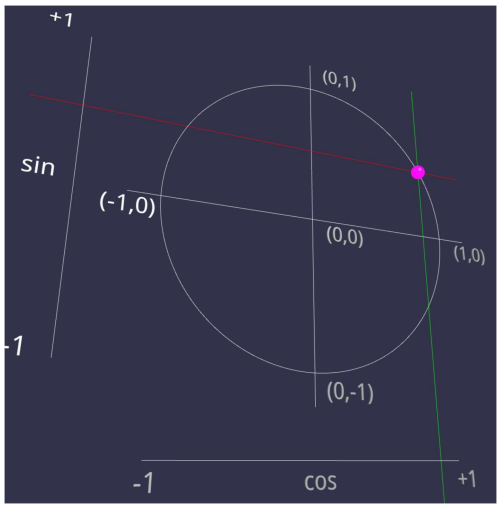
太陽と地球にする球体を２つ用意できましたので、地球の公転をアニメーションさせる処理を作成します。円運動は三角関数で簡単に実装できます。ちょっと三角関数を思い出してみましょう。



▲図 6.3　sin と cos

例えば r ＝ 1 の場合、sin θ ＝ y となり sin は高さを、cos θ ＝ x となり cos は横位置を、簡単に求めることができます。

参考のため、図面を描画する処理を Babylon.js で作成しました。



▲図 6.4　単位円サンプル　https://playground.babylonjs.com/#Y3J7V8

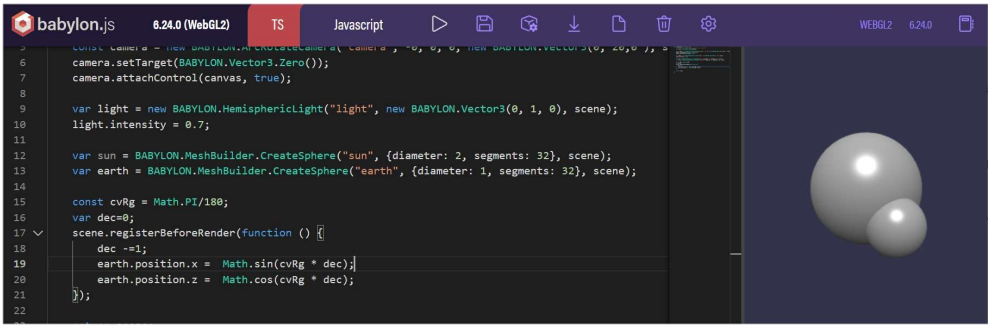
では作成した地球の座標を、sin と cos を使って求めていきましょう。太陽を中心に、地球の X 座標を sin θの値で、Z 座標を cos θの値で設定します。平面上の回転とするので Y 座標は０のままで行います。

修正箇所を以下に記載します。

▼ 地球を公転させる１（JavaScript）

```
1:      var sun = BABYLON.MeshBuilder.
2:          CreateSphere("sun", {diameter: 2, segments: 32}, scene);
3:      var earth = BABYLON.MeshBuilder.
4:          CreateSphere("earth", {diameter: 1, segments: 32}, scene);
5:
6:      const cvRg = Math.PI/180;
7:      var dec=0;
8:      scene.registerBeforeRender(function () {
9:          dec -=1;
10:         earth.position.x =  Math.sin(cvRg * dec);
11:         earth.position.z =  Math.cos(cvRg * dec);
12:     });
```

画面の描画を行う度に、１度（PI ÷１８０）づつ角度を変更して公転を再現しました。



▲図 6.5　円運動サンプル１　https://playground.babylonjs.com/#Y3J7V8#1

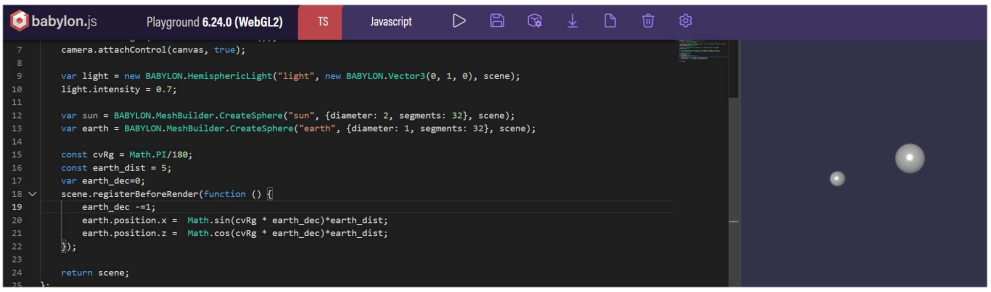
太陽の半径と公転が１の為、めり込んでしまっていますね。これはこれでなんか可愛いですが、x、y の計算結果を５倍して公転半径を広げます。

変更箇所を以下に記載します。

▼ 地球を公転させる２（JavaScript）

```
1:      const cvRg = Math.PI/180;
2:      const earth_dist = 5;
```

```
3:      var earth_dec=0;
4:      scene.registerBeforeRender(function () {
5:          earth_dec -=1;
6:          earth.position.x =  Math.sin(cvRg * earth_dec)*earth_dist;
7:          earth.position.z =  Math.cos(cvRg * earth_dec)*earth_dist;
8:      });
```



▲図 6.6　円運動サンプル 2　　https://playground.babylonjs.com/#Y3J7V8#2

以上のように三角関数で簡単に地球を公転させる事が出来ました。

6.2.3　月の公転を実装する

次は地球を周回する月を追加します。地球の時と同じように月の球体を CreateSphere で追加します。半径は０．３にしました。

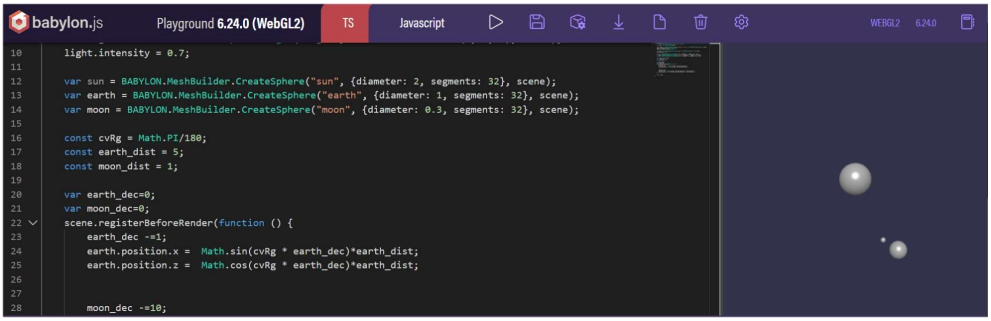
地球の周りを周回させる手順も同じですが、太陽と異なり地球は動いていますので、月の回転の中心が地球と共に動くことを考慮して座標を計算します。

修正箇所を以下に記載します。

▼　地球を公転させる 2（JavaScript）

```
1:      var sun = BABYLON.MeshBuilder.CreateSphere
2:                ("sun", {diameter: 2, segments: 32}, scene);
3:      var earth = BABYLON.MeshBuilder.CreateSphere
4:                ("earth", {diameter: 1, segments: 32}, scene);
5:      var moon = BABYLON.MeshBuilder.CreateSphere
6:                ("moon", {diameter: 0.3, segments: 32}, scene);
7:
8:      const cvRg = Math.PI/180;
9:      const earth_dist = 5;
10:     const moon_dist = 1;
11:
12:     var earth_dec=0;
13:     var moon_dec=0;
```

```
14:     scene.registerBeforeRender(function () {
15:         earth_dec -=1;
16:         earth.position.x =  Math.sin(cvRg * earth_dec)*earth_dist;
17:         earth.position.z =  Math.cos(cvRg * earth_dec)*earth_dist;
18:
19:         moon_dec -=10;
20:         moon.position.x = Math.sin(cvRg * moon_dec)
21:                           * moon_dist + earth.position.x;
22:         moon.position.z = Math.cos(cvRg * moon_dec)
23:                           * moon_dist + earth.position.z;
24:     });
```



▲図 6.7　円運動サンプル３　https://playground.babylonjs.com/#Y3J7V8#3

以上の処理で、地球の周りを公転する月を表現する事が出来ました。

6.2.4　月の影を追加する

日食を表現する為、月の影を追加します。
影は円錐を使って表現します。円錐の定義は以下のようになります。

▼　三角錐の定義

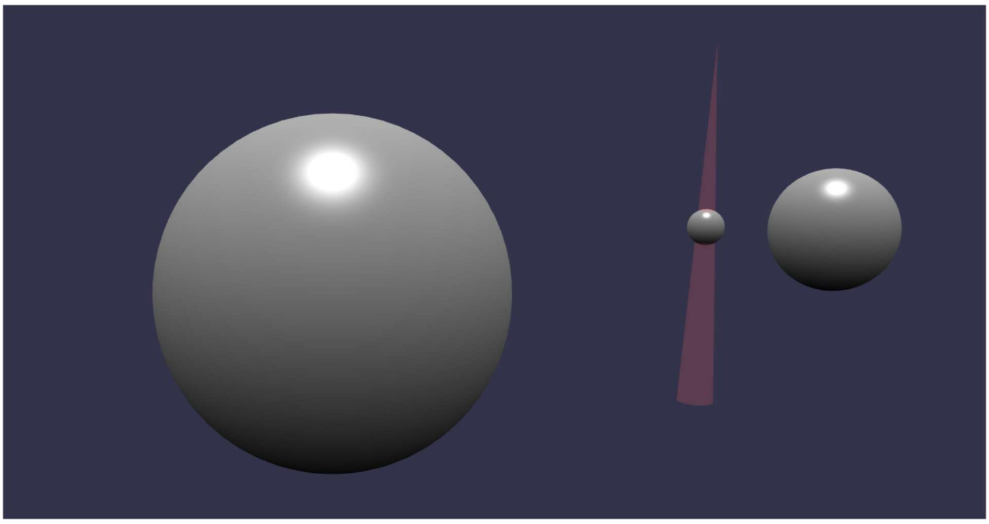
```
1:     BABYLON.MeshBuilder.CreateCylinder("名前", オプション, scene);
```

オプションで円錐の大きさなどを定義します。ひとまずは適当な長さ３を指定し、底面を月と同じサイズになるよう０．３、頂点のサイズは０の影を作成します。オプション部分には、「height:3, diameterTop: 0, diameterBottom:0.3」を記載しました。

他に、月と同じ座標に追従するようにし、影を半透明にします。
修正箇所を以下に記載します。

▼ 月の影を描画する（JavaScript）

```
 1:     //月の影を追加
 2:     var moonShadow = BABYLON.MeshBuilder.CreateCylinder
 3:                      ("moonShadow", {height:3, diameterTop: 0,
 4:                       diameterBottom:0.3}, scene);
 5:     var moonShadowMaterial = new BABYLON.StandardMaterial
 6:                              ("moonShadowMaterial", scene);
 7:     moonShadowMaterial.emissiveColor = new BABYLON.Color3(1, 0, 0);
 8:     moonShadowMaterial.alpha = 0.2;
 9:     moonShadow.material= moonShadowMaterial;
10:     ～中略～
11:
12:     scene.registerBeforeRender(function () {
13:         ～中略～
14:         //影の位置を月に追従させる
15:         moonShadow.position = moon.position;
16:     });
```



▲図 6.8　月の影を追加 1　　https://playground.babylonjs.com/#Y3J7V8#4

御覧の通りここで一つ問題が発生しました。円錐が直立した状態で描画されてしまいます。円錐を回転させ、太陽の反対に伸びるように調整する必要がありますがちょっと面倒ですね。３次元座標上での回転は、クオータニオンを使用すると簡単に求める事が出来ますのでこれを利用します。

修正箇所を以下に記載します。

▼ 月の影を描画する（JavaScript）

```
 1:     scene.registerBeforeRender(function () {
 2:         ～中略～
 3:         //影の位置を月に追従させる
 4:         moonShadow.position = moon.position;
 5:         //太陽と月の位置の差分から、クオータニオンで影の角度を調整
 6:         const direction = sun.position.subtract(moon.position);
 7:         direction.normalize();
 8:         const rot = BABYLON.Quaternion.FromUnitVectorsToRef
 9:   (new BABYLON.Vector3(0,-1,0), direction, new BABYLON.Quaternion());
10:         moonShadow.rotationQuaternion = rot;
11:     });
```

まず、太陽と月のベクトルを sun.position.subtract(moon.position) で求め、normalize で大きさ 1 のベクトルに変換し、次にクオータニオンを使用して回転角度を求めます。クオータニオンの FromUnitVectorsToRef では以下の定義となっているので、第１引数に Ｙ 軸を９０度回転させたベクトル、第二引数に太陽と月のベクトルを指定しています。

修正箇所を以下に記載します。

▼ クオータニオンの定義

```
1: Quaternion.FromUnitVectorsToRef(回転の基準となるベクトルの方向,
2:                                 回転先の方向ベクトル,
3:                                 結果を保存する4元数)
```

以上の処理で、下図のように影の向きが太陽を意識するようになりました。

▲図 6.9　月の影を追加２　　https://playground.babylonjs.com/#Y3J7V8#5

円錐を回転させることで影の向きは太陽と反対方向に延びるようになりましたが円錐と月の中心が同じ場所になってしまっていますね。円錐の位置を半分上にずらしてから回転するようにして月と三角錐の底辺が重なるよう調整し、月から影が伸びているように見えるようにします。

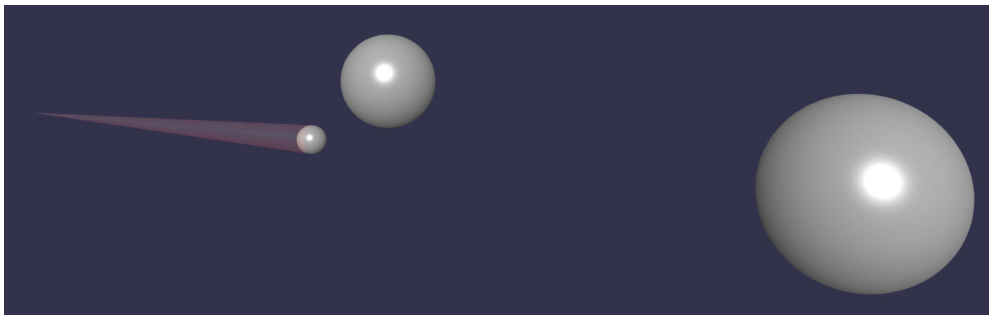
三角錐を作成する際に、オブジェクトの頂点操作を許可するオプション updatable:true を追加します。次に、オブジェクトの頂点データを getVerticesData にて取得します。取得したデータは、１つの頂点につき（X,Y,Z）３つの座標のリストになっているので頂点の数に合わせる為３で除算し、Y 座標（２番目のデータなので [i*3+1]）のみ、影の長さ３の半分の値１．５を加算し、updateVerticesData で影のオブジェクト頂点データを更新します。

修正箇所を以下に記載します。

▼ 月の影を描画する 3（JavaScript）

```
 1:   var moonShadow = BABYLON.MeshBuilder.CreateCylinder("moonShadow",
 2:      {height:3, diameterTop: 0, diameterBottom:0.3, updatable:true},
 3:      scene);
 4:   var moonShadowPos = moonShadow.getVerticesData
 5:                       (BABYLON.VertexBuffer.PositionKind);
 6:   var numberOfVertices =moonShadowPos.length/3;
 7:   for(var i = 0; i<numberOfVertices; i++) {
 8:     moonShadowPos[i*3+1] = moonShadowPos[i*3+1] +1.5;
 9:   }
10:   moonShadow.updateVerticesData(BABYLON.VertexBuffer.PositionKind,
11:                                 moonShadowPos);
```

7
8

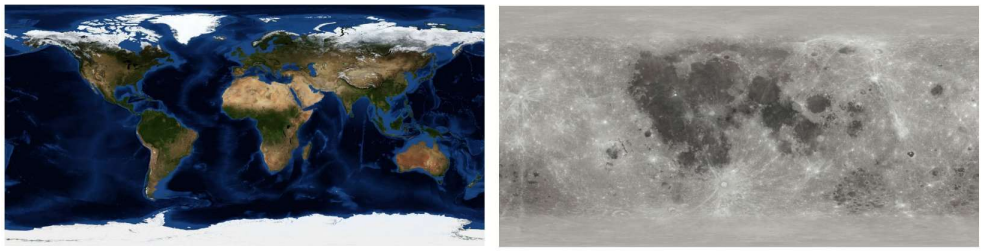


▲図 6.10　月の影を追加 3　　https://playground.babylonjs.com/#Y3J7V8#6

これでようやく、月から影が伸びるような表示を実現することができました。

6.2.5　天体の見た目をそれらしくする

太陽、地球、月と３つの球体を作成しましたが、灰色の球体のままなのは味気ないですね。見た目を少し綺麗にしましょう。地球に地表の画像をテクスチャマップとして球体に張り付けます。地表のテクスチャ用の画像とバンプマップ用の画像は、NASA のライブラリで公開されているものがありますので、そちらを使用させて頂きました。



▲図 6.11　ライブラリで公開されている地球と月の画像

球体にテクスチャーを張るためには以下のようにマテリアルに diffuseTexture で画像を追加します。

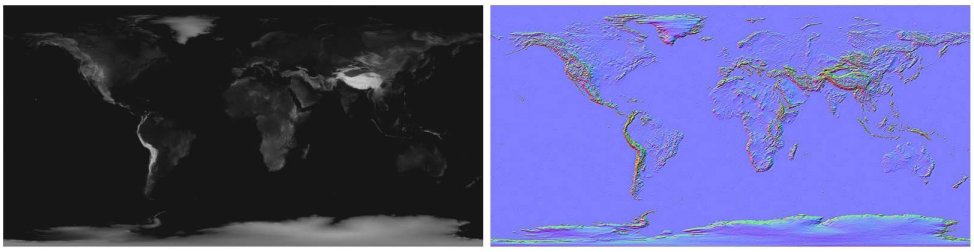
修正箇所を以下に記載します。

▼　地球にテクスチャを追加する

```
1:     var earth_material = new BABYLON.StandardMaterial
2:                              ("earth_material", scene);
3:     earth_material.diffuseTexture = new BABYLON.Texture
4:              ("https://raw.githubusercontent.com/ykoba079/sample1/
5:                master/image/babyBook15/earthTecture.jpg", scene);
6:     earth.material= earth_material;
```

球体の表面に、指定した画像が張り付いて地球らしくなりましたが、ちょっと表面がつるつるすぎて不自然になりますので細かい凹凸をバンプマッピングで追加します。バンプマッピングは反射光の角度を調整して疑似的に凹凸があるように見せかける手法です。

ライブラリには標高を表す白黒画像がありますので、これにソーベルフィルタを用いて下図のような画像に変換します。（ソーベルフィルタは OpenCV などライブラリの他、Unity や画像加工ソフトでも加工が可能です）



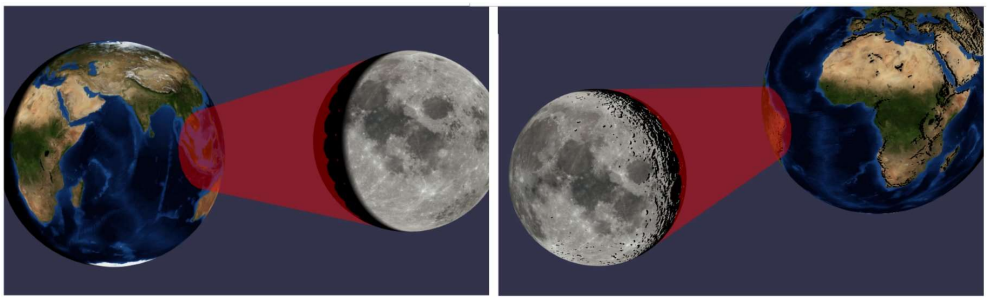
▲図 6.12　ライブラリで公開されている高さ MAP と、ソーベルフィルタで加工したバンプマップ用画像

バンプマッピング用の画像が用意できましたら、下記のように　bumpTexture でマテリアルに追加します。

▼　地球に凹凸の表現を追加する

```
1:   earth_material.bumpTexture = new BABYLON.Texture
2:              ("https://raw.githubusercontent.com/ykoba079/sample1/
3:                master/image/babyBook15/earthBump.jpg", scene);
4:   earth_material.invertNormalMapX = true;
5:   earth_material.invertNormalMapY = true;
6:   earth.material= earth_material;
```

テクスチャーのみのものと、バンプマッピング付のものを比較してみましょう。凸凹が強調されるようになりました。



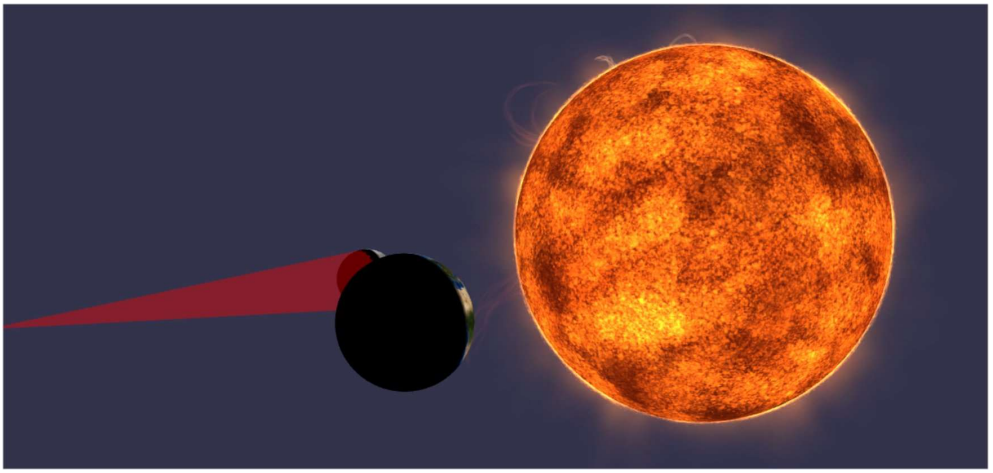
▲図 6.13　テクスチャーのみの表示と、バンプマッピング付きの表示

さて、太陽ですが太陽は個体ではなく核融合をする気体です。地表というものがありません。テクスチャーマッピングなどを用いた手法ですと固い物体のようになってしまいますので、パーティクルシステムで表現します。Babylon.js Playground には太陽の**パーティクルヘルパー**が用意されていますのでこれを使用します。

下記の 3 行で太陽の再現が出来ます。

▼ 太陽のパーティクルヘルパーを使用する

```
1:     BABYLON.ParticleHelper.CreateAsync("sun", scene).then((set) => {
2:         set.start();
3:     });
```



▲図 6.14 太陽のパーティクルヘルパーを追加した表示結果 https://playground.babylonjs.com/#Y3J7V8#7

パーティクルの定義があらかじめ用意されているので、実装が非常に簡単ですね。

ここで一つ注意点があります。描画オプションとして、レイヤーのように描画の並べ替えをコントロールするオプション **renderingGroupId** があります。通常は意識する必要はあまりないのですが、（ドキュメントにも、通常は直接使用しないよう書かれていました）太陽のパーティクルヘルパー定義内でこれを使用しており、その結果太陽の描画が優先され手前に地球が来ても太陽の後ろ側に表示されてしまいました。

パーティクルヘルパー内の定義を見ると、renderingGroupId=3 との記述がありますので、地球、月も同じグループで表示されるように、

▼ 太陽のパーティクルヘルパーと描画レベルを合わせる

```
1:   earth.renderingGroupId=3
2:   moon.renderingGroupId=3
```

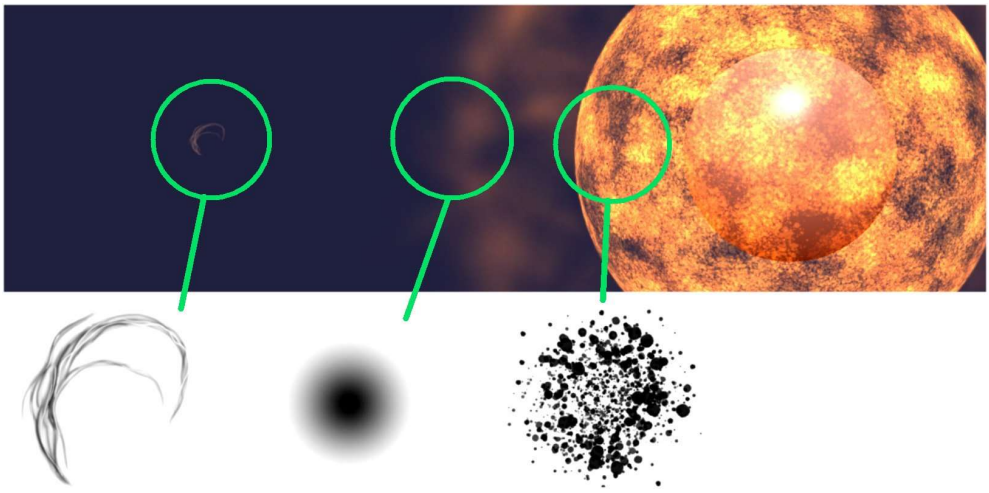
と、それぞれのマテリアルに記述を追加する必要があります。renderingGroupId を合わせる事で、前後関係が正しく表示されるようになります。

6.2.6　みちくさ　太陽のパーティクル

用意されていた太陽のパーティクルヘルパーについて、少し覗いてみましょう。
下記の JSON を使用してパーティクルを作成しているようです。
https://github.com/BabylonJS/Assets/blob/master/particles/systems/sun.json
JSON 内で定義されている要素を分けると、表面のシミ状の模様、ガス状のぼやけた模様、プロミネンス　を、複数作成し、太陽のような外見を実現していました。パーティクルヘルパーを呼び出している箇所で、内部のデータを変更して要素ごとに表示位置を分離してみます。

▼　太陽のパーティクルヘルパーの要素の表示位置をずらす

```
1:     BABYLON.ParticleHelper.CreateAsync("sun", scene).then((set) => {
2:         set.systems[0].particleEmitterType.radius=2;
3:         set.systems[2].particleEmitterType.radius=3;
4:         set.systems[1].particleEmitterType.radius=5;
5:         set.start();
6:     });
```

▲図 6.15　太陽のパーティクルヘルパーのパーツを分割した結果

少数のプロミネンスと、多数のガスと表面用のシミに分割されました。これらを重ねて表示してリアルな太陽を描画していたことが判ります。

有用なヘルパーですので参考にしてみましょう。下記の URL に用意されているパーティクルのリストがあります。

https://doc.babylonjs.com/features/featuresDeepDive/particles/particle_system/particleHelper

6.2.7　皆既日食が毎月発生しない点の表現

ここまでで太陽と地球と月の軌道運動を３Ｄで再現してみました。

教科書によって記載されていたりそうでなかったりするようですが、月の軌道が傾いている為太陽と月が重ならず皆既日食が発生しないと記述がありますので、これも再現します。

月の軌道は ５．１度 傾いているので、傾いた起動で地球を周回するように月の座標を変更するコードを追加します。３次元空間の回転は、行列と sin,cos で行います。

$$(x,y,z)\begin{pmatrix} 1, & 0, & 0 \\ 0, & \cos\theta, & -\sin\theta \\ 0, & \sin\theta, & \cos\theta \end{pmatrix}$$

▲図 6.16　X 軸で XYZ 座標を回転させる式

修正箇所を以下に記載します。

▼　月の軌道を傾ける

```
 1:  moon.position.x = Math.sin(cvRg * moon_dec)
 2:                  * moon_dist + earth.position.x;
 3:  moon.position.z = Math.cos(cvRg * moon_dec)
 4:                  * moon_dist + earth.position.z;
 5: //XとZから、X軸に回転させた座標を指定する
 6:  rotateX(arrMoon[i].position,arrEarth[i].position,【回転させる角度】)
 7:
 8: //X軸に対して指定角度回転させる関数
 9: function rotateX(posMoon ,posEarth ,rg){
10:   //X軸中心に回転させるので、x座標は割愛
11:   var y = posMoon.y - posEarth.y;                  //地球からのy位置
12:   var z = posMoon.z - posEarth.z;                  //地球からのz位置
13:   posMoon.y = y * Math.cos(cvRg*rg) - z * Math.sin(cvRg*rg)+posEarth.y;
14:   posMoon.z = y * Math.sin(cvRg*rg) + z * Math.cos(cvRg*rg)+posEarth.z;
15: }
```

ここまでは見やすくするため、太陽や地球のサイズや距離を無視していましたが、

近すぎて 5 度の傾きが誤差になってしまうので、ここで実際のスケールに合わせます。使用する値を下記に記載しました。

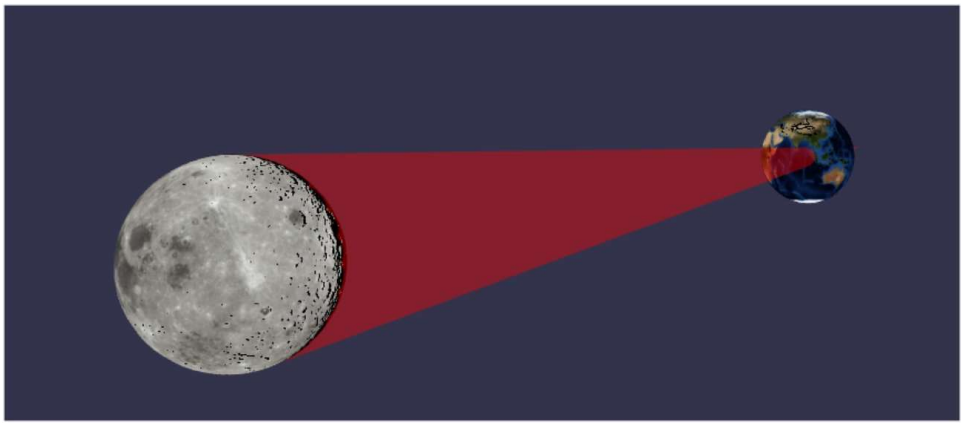
▼表 6.1　太陽と地球と月の大きさ

天体	距離
太陽の大きさ	1,392,700 km
太陽から地球迄の距離	149,600,000 km
地球の大きさ	12,742 km
地球から月までの距離	384,400 km（平均）
月の大きさ	3,475 km

遠くて見にくくなってしまいますが、縮尺の比率を合わせる事で実際と同じように再現が可能です。

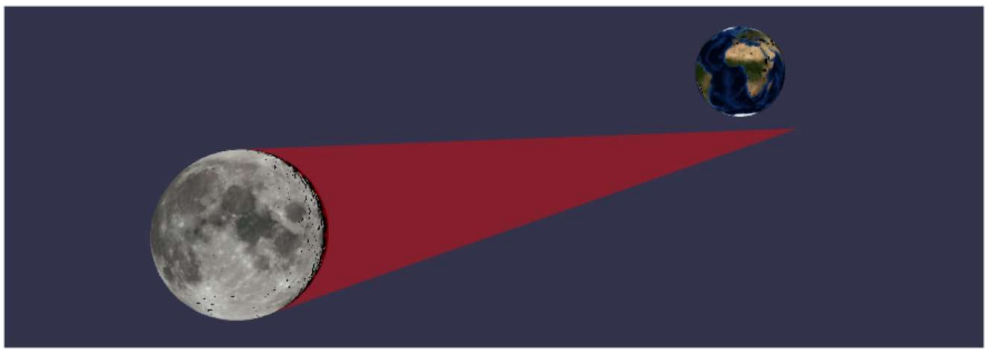
6.2.8　皆既日食と皆既月食は年に 2 回起きている。

実際に動かして確認してみます。初期値 (y 軸を sin にしたため) で月の y 座標がまだ平面上付近の場合は、下図のように影が地球に当たるよう表示されました。



▲図 6.17　皆既日食が発生している状態

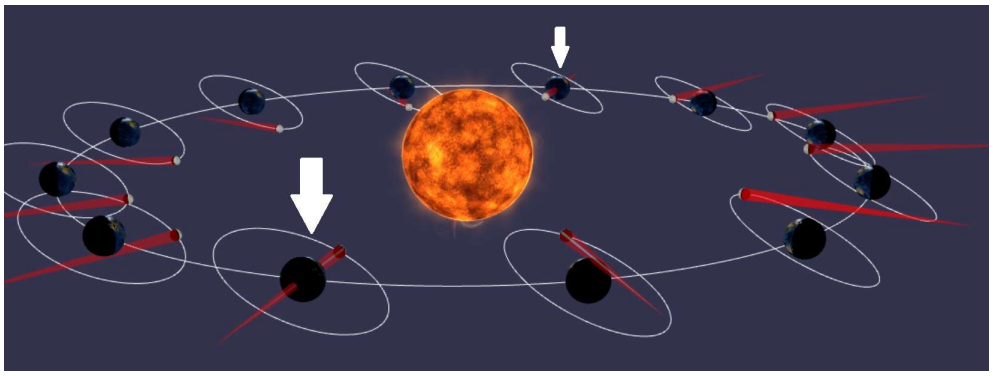
月の軌道が斜めの為、時間を進めると地球の位置が変わっていき、新月のタイミングで下図のように地球からずれた位置に影があるのが判ります。



▲図 6.18　皆既日食にならない状態

ただ、やはりというか実際の大きさに合わせると月と地球すら遠すぎてカメラ操作が非常にシビアになっていまいました。何より宇宙空間は目印が無くて状況の把握が難しいです。

解りやすいよう 1 年間の新月際の月の位置と、その影をデフォルメして表示するバージョンも用意しました。こちらの方が解りやすいですね。月と地球の間隔なども近くしたので、月の傾きを 25 度と強調しています。



▲図 6.19　新月毎の月の影の表示 https://playground.babylonjs.com/#Y3J7V8#8

上の画像で月の軌道を傾けた結果、1 年の間に月の影と地球が重なるチャンスは矢印の 2 回である事が判ると思います。

皆既日食の履歴と予定は天文台などのサイトに載っていますので以下に記載しますが、おおよそ 6 ヵ月周期で皆既日食が発生しています。

▼表 6.2　2020 年以降の皆既日食の発生日

日付	日食の種類	日食が見える地域
2020 年 06 月 21 日	金環日食	アフリカ、アジア、太平洋
2020 年 12 月 15 日	皆既日食	南太平洋、南米、南大西洋
2021 年 06 月 10 日	金環日食	北極付近
2021 年 12 月 04 日	皆既日食	南極付近
2022 年 05 月 01 日	部分日食	南太平洋、南米など
2022 年 10 月 25 日	部分日食	ヨーロッパ、アフリカ北部、中東、インド
2023 年 04 月 20 日	皆既日食	インド洋、アジア、オセアニア
2023 年 10 月 15 日	金環日食	北米、南米

ちなみに、皆既月食も太陽と地球と月が直線状に並んだタイミングで発生するので、皆既日食の前後の満月になったタイミングで発生します。

皆既月食は、地球の影に月全体が隠れる際なので地球の何処から見ても見る事が出来ますが（皆既月食中に夜側に居れば)、皆既日食は、月の影に地球の一部が入っている場合に見る事が出来る現象なので、月の影に入った一部の地域でしか確認できません。

皆既日食自体は半年に一度発生はするものの、観測できる限定された地域にいるかというもう一つのハードルがあるので、実際にはかなりレアな現象になってしまいます。日本の次回の予定は 2030 年 6 月 1 日とのこと。

最近は世界の天文台などで Live 配信もありますが、お住まいの地域で観測できるチャンスがあればぜひ生で体験してみてください。

6.3　おわりに

Babylon.js Playground を使用して子供のころ疑問に感じた皆既日食はなぜ毎月起こらないのか。という点について仕組みを再現してみました。３ＤＣＧを Web ブラウザで簡単に表現できるので、実際に見る事が難しい天体や、生物や分子についての表現が容易になってきています。平面図や写真だけでは解り難いものを説明する資料などに活用が出来るのではないでしょうか。

また、今回は簡単な三角関数も合わせて使用しました。この章の中で数学の汎用性を感じて、プログラムと、数学と、物理など他の科目とを繋ぎ合わせて色んなことが再現できる点に興味を持っていただければ幸いです。