**Getting Started - Chapter 3 - A Walk Around The Village**

**@@启程-第三章-绕着村庄走 @@**

**A Walk Around The Village**

**@@绕着村庄走@@**

There is a useful property of a mesh, *movePOV* which allows us to move a mesh relative to its point of view. Generally a newly created mesh will be considered as facing the negative z direction and this is the direction of its point of view. To move a mesh forward 6 units in the direction of its point of view you use

@@网格有一个很有用的属性——“*movePOV* ”，它允许我们相对于一个网格的观察角度（面向的方向）移动网格。一般地，一个新建的网格被认为是面向z轴反方向的，并且这个方向就是它的“观察角度”。你可以使用如下方法，沿着一个网格的观察角度移动6单位@@

mesh.movePOV(0, 0, 6)

The parameters are, in order, distance to move to the right, up and forward, generally these are the negative x axis, the positive y axis and the negative z axis in the mesh's local space.

@@这些参数按顺序分别是，向右、向上、向前的移动距离，通常它们是网格局部坐标系中x轴负方向、y轴正方向、z轴负方向。@@

（译者注：使用这个方法就不需要再手动将模型的运动向量转为全局坐标系中的向量，以下是手动转换的方法：BABYLON.Vector3.TransformNormalToRef(vector1, mesh.worldMatrixFromCache, vector1);）

In Babylon.js you can write code that will be executed before the rendering of the next frame using

@@在Babylon.js中你可以使用以下方法书写将要在下一帧渲染前执行的代码：@@

scene.onBeforeRenderObservable.add(() => {

//code to execute 要执行的代码

});

In this way properties of objects can be changed render frame by render frame.

@@使用这样的方法，将可以一帧接一帧地改变对象的属性。@@

Let us take the simple case of a sphere moving around the edges of a triangle. We want the sphere to appear to slide along one side, turn to slide along the next and then turn and slide along the last side and then repeat.

@@让我们考虑一个球体绕一个三角形的边缘运动的简单情况。我们希望球体看起来沿着轨迹的一侧滑动，在到达下一条轨迹时转变方向沿着下一条轨迹滑动，并不断重复。@@

This is also an opportunity to introduce two types of mesh you can create, a sphere and a series of lines. Take a sphere sliding around an isosceles right angled triangle as an example.

@@这也是一个介绍两种你可以建立的网格的机会——球体以及线段组（译者注：严格来讲“线”和“点”并不属于“网格”，它们与网格对象的属性差别较大，在OpenGL底层的绘制方法也不同）。这里以一个绕等腰直角三角形滑动的球体为例：@@

const sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 0.25});

//end points for the line sequence in an array 将线段序列的端点保存在一个数组中

//y component can be non zero y值可以不为零

const points = [];

points.push(new BABYLON.Vector3(2, 0, 2));

points.push(new BABYLON.Vector3(2, 0, -2));

points.push(new BABYLON.Vector3(-2, 0, -2));

points.push(points[0]); //close the triangle; 闭合三角形，这里可以重用向量对象！

BABYLON.MeshBuilder.CreateLines("triangle", {points: points})

Also you can see an other method, *rotate*, of rotating. This method rotates the mesh about the given axis by the given angle in radians. It adds to the current rotation.

@@你还能看到另一个方法“旋转”。这个方法绕给定的轴按以弧度为单位的角度旋转网格。这种姿态改变将被添加到网格当前的姿态属性中。@@

mesh.rotate(axis, angle, BABYLON.Space.LOCAL);

To produce the animation before each render frame the sphere will move a distance of 0.05. When the distance it has travelled is greater than 4 the sphere will make a turn, greater than 8 it will turn again and when greater than the perimeter it will reset and start again.

@@要产生动画，球体需要在每一个渲染帧之前移动0.05单位的距离。当球体的移动距离超过4时，球体将进行转向，当大于8时它将再次转向，当大于三角形的周长时球体将复位，然后再次开始运动。@@

We set up a track array of objects with the properties turn and distance. After travelling the given total distance the sphere will rotate by the given turn value.

@@我们建立了一个轨道数组，其中的每个元素对象都是转弯属性和移动距离。当移动了数组元素给定的合计距离后这个球体将按照给定的转弯数据进行转弯。@@

const slide = function (turn, dist) { //after covering dist apply turn 当到达指定距离后进行转向

this.turn = turn;

this.dist = dist;

}

const track = [];

track.push(new slide(Math.PI / 2, 4)); //first side length 4 第一条边长为4

track.push(new slide(3 \* Math.PI / 4, 8)); //at finish of second side distance covered is 4 + 4 完成第二条边时移动距离为4+4

track.push(new slide(3 \* Math.PI / 4, 8 + 4 \* Math.sqrt(2))); //all three sides cover the distance 4 + 4 + 4 \* sqrt(2) 走完所有的三条边经过的距离为4+4+4\*根号二

Whenever the required distance is reached a turn is made and the array index pointer, p, is increased by 1. The modulo operator *%* is used to reset the pointer to zero at the end of the array.

@@当达到指定距离时执行一次转弯，同时数组的索引游标p增加1.这里的取值操作符“%”，用来在数组的末尾将索引重置为0。@@

if (distance > track[p].dist) {

sphere.rotate(BABYLON.Axis.Y, track[p].turn, BABYLON.Space.LOCAL);

p +=1;//p=p+1

p %= track.length;//p=p% track.length

}

To prevent floating point errors accumulating, whenever the index pointer is reset to 0 the position and rotation of the sphere is also reset

@@为了防止浮点数计算错误积累，当索引值重置为0时，球体的位置和姿态也将重置。@@

if (p === 0) {

distance = 0;

sphere.position = new BABYLON.Vector3(2, 0, 2); //reset to initial conditions 重置为初始状态

sphere.rotation = BABYLON.Vector3.Zero();//prevents error accumulation 防止浮点数计算错误积累

}

Animating To a Path

路径动画

[https://playground.babylonjs.com/#N9IZ8M#1](https://playground.babylonjs.com/#1)

const createScene = function () {

    const scene = new BABYLON.Scene(engine);

    const camera = new BABYLON.ArcRotateCamera("camera", -Math.PI / 1.5, Math.PI / 5, 15, new BABYLON.Vector3(0, 0, 0));

    camera.attachControl(canvas, true);

    const light = new BABYLON.HemisphericLight("light", new BABYLON.Vector3(1, 1, 0));

    //create a sphere

    const sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 0.25});

    sphere.position = new BABYLON.Vector3(2, 0, 2);

    //draw lines to form a triangle

    const points = [];

    points.push(new BABYLON.Vector3(2, 0, 2));

    points.push(new BABYLON.Vector3(2, 0, -2));

    points.push(new BABYLON.Vector3(-2, 0, -2));

    points.push(points[0]); //close the triangle;

    BABYLON.MeshBuilder.CreateLines("triangle", {points: points})

    const slide = function (turn, dist) { //after covering dist apply turn

        this.turn = turn;

        this.dist = dist;

    }

    const track = [];

    track.push(new slide(Math.PI / 2, 4));

    track.push(new slide(3 \* Math.PI / 4, 8));

    track.push(new slide(3 \* Math.PI / 4, 8 + 4 \* Math.sqrt(2)));

    let distance = 0;

    let step = 0.05;

    let p = 0;

    scene.onBeforeRenderObservable.add(() => {

        sphere.movePOV(0, 0, step);

        distance += step;

        if (distance > track[p].dist) {

            sphere.rotate(BABYLON.Axis.Y, track[p].turn, BABYLON.Space.LOCAL);

            p +=1;

            p %= track.length;

            if (p === 0) {

                distance = 0;

                sphere.position = new BABYLON.Vector3(2, 0, 2); //reset to initial conditions

                sphere.rotation = BABYLON.Vector3.Zero();//prevents error accumulation

            }

        }

    });

    return scene;

};

A little trickier and using a bit of trial and error for the turns and distance we can achieve a more complicated walk for the character around the village. One reason for using degrees and converting them to radians for the *rotate* method is that it is easier to adjust by adding one or two degrees.

@@使用一点小技巧再加上对转向和距离的反复调试，我们就可以为绕着村庄行走的角色实现一个更加复杂的行动路径。之所以使用角度单位，然后再把角度转换为旋转方法所需的弧度单位，是因为使用角度单位时更容易对旋转角度进行微调。@@

Since the character, dude, imported from the *.babylon* file has had its rotation set using a *rotationQuaternion* rather than rotation we use the rotate method to reset the characters orientation.

@@因为从.babylon模型文件中导入的角色——老哥，已经具有以“旋转四元数”方式保存的姿态信息，而不是以姿态属性保存，我们要使用旋转方法重置角色的朝向（然而代码中用的是克隆的初始旋转四元数属性）@@

dude.position = new BABYLON.Vector3(-6, 0, 0);

dude.rotate(BABYLON.Axis.Y, BABYLON.Tools.ToRadians(-95), BABYLON.Space.LOCAL);

const startRotation = dude.rotationQuaternion.clone(); //use clone so that variables are independent not linked copies 使用克隆方法，这样startRotation变量将会是与网格无关的自由变量，否则旋转时startRotation对象也会改变，就无法重置为最初值了。

if (p === 0) {

distance = 0;

dude.position = new BABYLON.Vector3(-6, 0, 0);

dude.rotationQuaternion = startRotation.clone();

}

Character Walking Through Town

角色走过小镇

https://playground.babylonjs.com/#KBS9I5#81

We now have two things moving around the village a car and a character. How can we avoid them colliding?

@@现在我们有了两个绕村庄移动的东西——一辆小车和一个角色。我们该如何避免它们相撞呢？@@

const createScene = function () {

    const scene = new BABYLON.Scene(engine);

    const camera = new BABYLON.ArcRotateCamera("camera", -Math.PI / 1.5, Math.PI / 2.2, 15, new BABYLON.Vector3(0, 0, 0));

    camera.attachControl(canvas, true);

    const light = new BABYLON.HemisphericLight("light", new BABYLON.Vector3(1, 1, 0));

    BABYLON.SceneLoader.ImportMeshAsync("", "https://assets.babylonjs.com/meshes/", "village.glb");

    const walk = function (turn, dist) {

        this.turn = turn;

        this.dist = dist;

    }

    const track = [];

    track.push(new walk(86, 7));

    track.push(new walk(-85, 14.8));

    track.push(new walk(-93, 16.5));

    track.push(new walk(48, 25.5));

    track.push(new walk(-112, 30.5));

    track.push(new walk(-72, 33.2));

    track.push(new walk(42, 37.5));

    track.push(new walk(-98, 45.2));

    track.push(new walk(0, 47))

    // Dude

    BABYLON.SceneLoader.ImportMeshAsync("him", "/scenes/Dude/", "Dude.babylon", scene).then((result) => {

        var dude = result.meshes[0];

        dude.scaling = new BABYLON.Vector3(0.008, 0.008, 0.008);

        dude.position = new BABYLON.Vector3(-6, 0, 0);

        dude.rotate(BABYLON.Axis.Y, BABYLON.Tools.ToRadians(-95), BABYLON.Space.LOCAL);

        const startRotation = dude.rotationQuaternion.clone();

        scene.beginAnimation(result.skeletons[0], 0, 100, true, 1.0);

        let distance = 0;

        let step = 0.015;

        let p = 0;

        scene.onBeforeRenderObservable.add(() => {

            dude.movePOV(0, 0, step);

            distance += step;

            if (distance > track[p].dist) {

                dude.rotate(BABYLON.Axis.Y, BABYLON.Tools.ToRadians(track[p].turn), BABYLON.Space.LOCAL);

                p +=1;

                p %= track.length;

                if (p === 0) {

                    distance = 0;

                    dude.position = new BABYLON.Vector3(-6, 0, 0);

                    dude.rotationQuaternion = startRotation.clone();

                }

            }

        })

    });

    return scene;

};