**Unity3D教程：车辆性能算法**

Posted on 2013年06月05日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 123 次

首先要了解真实车辆的原理：车辆分前轮驱动，后轮驱动和四驱动。动力由引擎提供，反应的力到轮胎上，因此产生转数，即RPM。引擎的功率可以由RPM得到公式为 : RPM = 引擎功率×60/2×pi ， 这些都是模拟，只为了更好的为下面的动作服务。还有大众关心的“漂移” ，所谓 漂移就是，在后驱车辆中，前轮方向旋转大角度，地面给于一定向心力，同时后轮又给予更多动力，导致“漂移”动作。

首先，车辆不可能整个套一个外壳，原因是在接触地面时，对车辆所使的力不可能达到你预期的目标，引起，必须在车辆轮胎以上做外壳碰撞，轮胎以下就需要有力来支持它始终保持不掉下来。

Unity3D中有个WheelCollider ， 它是专门模拟轮胎支持力和轮胎转数，以及轮胎横向力，前进力，以及悬架避震系统。这个东西非常方便，只要你把这个东西套在4个轮胎上，调试下他的forwardFriction 和 sidewaysFriction达到你想要的效果，然后对驱动轮的motorTorque进行赋值，你的车辆就能动了。

记得你需要无数次调试 前进摩擦力和横向摩擦力 。 至于悬架系统在你需要时也可以改变其值。还有，这两个摩擦力，是一个由低到高，再由高到稳定的一条曲线。

这个WheelCollider非常好用，曾一度沉迷于此。但后来发现，他有很多不合理的地方。想要得到最好的效果，还是抛弃了他。为了支持车辆的碰撞外壳不接触地面，必须写一个悬架动态支持力，在4个轮胎位置，支持整辆车悬浮于地面之上。

关于这个悬架动态支持力：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | **void** SuspensionHoldForce() |
| 02 | { |
| 03 | **float** fullCompressionSpringForce = **this**.rigidbody.mass \* 0.25f \* 2.0f \* -Physics.gravity.y; |
| 04 | **this**.OnGround = **true**; |
| 05 |  |
| 06 | **foreach**( GameObject item **in** FwheelModels ) |
| 07 | { |
| 08 | RaycastHit hit; |
| 09 | **bool** onGround = Physics.Raycast( item.transform.parent.position , -item.transform.parent.InverseTransformDirection(Vector3.up), **out** hit, **this**.suspensionTravel + **this**.radius); |
| 10 |  |
| 11 | **if** (onGround && hit.collider.isTrigger) |
| 12 | { |
| 13 | onGround = **false**; |
| 14 | **float** dist = **this**.suspensionTravel + **this**.radius; |
| 15 | RaycastHit[] hits = Physics.RaycastAll( item.transform.parent.position , -item.transform.parent.InverseTransformDirection(Vector3.up) , **this**.suspensionTravel + **this**.radius ); |
| 16 | **foreach**(RaycastHit test **in** hits) |
| 17 | { |
| 18 | **if** (!test.collider.isTrigger && test.distance <= dist) |
| 19 | { |
| 20 | hit = test; |
| 21 | onGround = **true**; |
| 22 | dist = test.distance; |
| 23 | } |
| 24 | } |
| 25 | } |
| 26 |  |
| 27 | **if**( onGround ) |
| 28 | { |
| 29 | Vector3 wheelVelo = **this**.rigidbody.GetPointVelocity (item.transform.parent.position); |
| 30 | Vector3 localVelo = transform.InverseTransformDirection (wheelVelo); |
| 31 | Vector3 groundNormal = transform.InverseTransformDirection (hit.normal); |
| 32 | **float** damperForce = Vector3.Dot(localVelo, groundNormal) \* 5000f; |
| 33 | **float** compression = 1.0f - ((hit.distance - radius) / suspensionTravel); |
| 34 | Vector3 springForce = ( fullCompressionSpringForce\*compression - damperForce ) \* item.transform.parent.InverseTransformDirection(Vector3.up); |
| 35 |  |
| 36 | springForce.z = springForce.x = 0f; |
| 37 |  |
| 38 | **this**.rigidbody.AddForceAtPosition( springForce , item.transform.parent.position ); |
| 39 |  |
| 40 | } |
| 41 | **else** |
| 42 | { |
| 43 | **this**.OnGround = **false**; |
| 44 | } |
| 45 | } |
| 46 |  |
| 47 | **foreach**( GameObject item **in** BwheelModels ) |
| 48 | { |
| 49 | RaycastHit hit; |
| 50 | **bool** onGround = Physics.Raycast( item.transform.parent.position, -item.transform.parent.InverseTransformDirection(Vector3.up), **out** hit, **this**.suspensionTravel + **this**.radius); |
| 51 |  |
| 52 | **if** (onGround && hit.collider.isTrigger) |
| 53 | { |
| 54 | onGround = **false**; |
| 55 | **float** dist = **this**.suspensionTravel + **this**.radius; |
| 56 | RaycastHit[] hits = Physics.RaycastAll( item.transform.parent.position, -item.transform.parent.InverseTransformDirection(Vector3.up) , **this**.suspensionTravel + **this**.radius ); |
| 57 | **foreach**(RaycastHit test **in** hits) |
| 58 | { |
| 59 | **if** (!test.collider.isTrigger && test.distance <= dist) |
| 60 | { |
| 61 | hit = test; |
| 62 | onGround = **true**; |
| 63 | dist = test.distance; |
| 64 | } |
| 65 | } |
| 66 | } |
| 67 |  |
| 68 | **if**( onGround ) |
| 69 | { |
| 70 | Vector3 wheelVelo = **this**.rigidbody.GetPointVelocity (item.transform.parent.position); |
| 71 | Vector3 localVelo = transform.InverseTransformDirection (wheelVelo); |
| 72 | Vector3 groundNormal = transform.InverseTransformDirection (hit.normal); |
| 73 | **float** damperForce = Vector3.Dot(localVelo, groundNormal) \* 5000f; |
| 74 | **float** compression = 1.0f - ( ( hit.distance - radius ) / suspensionTravel ); |
| 75 | Vector3 springForce = ( fullCompressionSpringForce\*compression - damperForce ) \* item.transform.parent.InverseTransformDirection(Vector3.up); |
| 76 | springForce.z = springForce.x = 0f; |
| 77 | **this**.rigidbody.AddForceAtPosition( springForce , item.transform.parent.position ); |
| 78 | } |
| 79 | **else** |
| 80 | { |
| 81 | **this**.OnGround = **false**; |
| 82 | } |
| 83 | } |
| 84 | } |

那么在完成悬架支撑后，就该设计车辆动力了。

这里也有2种方法：一个方向是真实车辆行驶轨迹，另一个是模拟型车辆轨迹。

前者的方法是 ， 将动力点放在车辆驱动轮上，例如后轮。用rigidbody的AddForceAtPosition可以做到，前轮只需要提供横向力就可以实现转弯的轨迹。但别看说说这么容易，这里面还涉及非常多的数值和曲线问题。在提供车辆动力时，你需要一条曲线，以致车辆不会匀加速，因为这样很不真实，还有在前轮横向力中，你必需是条由0到最高点，然后下降到平衡点的曲线。这样你的赛车才显得更真实。这些都需要用到几个数学知识。Unity3D教程手册

后者，是用算法来模拟的一种车辆轨迹。这个算法所有作用力作用在车辆的中心点。

转弯轨迹，我是用转弯半径来表示，使得车辆在转弯时有相当的真实性，必须改变车辆转弯速度。当然，用到了些数学知识。代码如下：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | #region 计算转弯角度 |
| 02 | **void** Steering( **bool** canSteer , Vector3 relativeVelocity ) |
| 03 | { |
| 04 | **if**( canSteer && **this**.OnGround ) |
| 05 | { |
| 06 | **if**( **this**.shiftthrottle == 1 ) |
| 07 | { |
| 08 | **this**.transform.RotateAround( **this**.transform.TransformPoint( ( **this**.FwheelModels[0].transform.localPosition + **this**.FwheelModels[1].transform.localPosition) \* 0.5f ) , **this**.transform.up , **this**.rigidbody.velocity.magnitude \*2f\* **this**.steeringInput \* Time.deltaTime \* 2f ); |
| 09 | *//~ this.rigidbody.AddForceAtPosition( this.FwheelModels[0].transform.TransformDirection(Vector3.right\*this.steeringInput) \* 3f \* this.rigidbody.mass, this.FwheelModels[0].transform.position);* |
| 10 | *//~ this.rigidbody.AddForceAtPosition( this.FwheelModels[1].transform.TransformDirection(Vector3.right\*this.steeringInput) \* 3f \* this.rigidbody.mass, this.FwheelModels[1].transform.position);* |
| 11 | **return** ; |
| 12 | } |
| 13 |  |
| 14 | **if**( **this**.throttle \* **this**.transform.InverseTransformDirection(**this**.rigidbody.velocity).z < 0 ) |
| 15 | **return** ; |
| 16 |  |
| 17 | **float** turnRadius = 3.0f / Mathf.Sin( (90f - **this**.steering) \* Mathf.Deg2Rad ); |
| 18 | **float** minMaxTurn = EvaluateSpeedToTurn(**this**.rigidbody.velocity.magnitude); |
| 19 | **float** turnSpeed = Mathf.Clamp(relativeVelocity.z / turnRadius, -minMaxTurn / 10, minMaxTurn / 10); |
| 20 | **this**.transform.RotateAround( **this**.transform.position + **this**.transform.right \* turnRadius \* **this**.steeringInput , transform.up , turnSpeed \* Mathf.Rad2Deg \* Time.deltaTime \* **this**.steeringInput ); |
| 21 |  |
| 22 | *//~ Vector3 debugStartPoint = transform.position + transform.right \* turnRadius \* this.steeringInput;* |
| 23 | *//~ Vector3 debugEndPoint = debugStartPoint + Vector3.up \* 5f;* |
| 24 |  |
| 25 | *//~ Debug.DrawLine(debugStartPoint, debugEndPoint, Color.red);* |
| 26 | } |
| 27 | } |
| 28 |  |
| 29 | **float** EvaluateSpeedToTurn( **float** speed ) |
| 30 | { |
| 31 | **if**(speed > **this**.topSpeed / 2) |
| 32 | **return** minimumTurn; |
| 33 | **float** speedIndex = 1 - ( speed / ( **this**.topSpeed / 2 ) ); |
| 34 | **return** minimumTurn + speedIndex \* (maximumTurn - minimumTurn); |
| 35 | } |
| 36 | #endregion |

这个模拟车辆轨迹，不能达到漂移的性能，但我加了一个滑动比例计算的算法，用车辆横向移动速度，和前进速度，的比例来确定，该车辆是否处于漂移状态，如处于，则启动漂移滑动程序。当然，我的赛车是很自然的，不做做。至于轮胎痕迹，就是判断是否触底后，在该点生成轮胎痕迹gameobject，如此而已。Unity3D教程手册

最后，再介绍下，所有车辆都需要模拟的，行驶时，轮胎随速度旋转这个关系到车辆看起来真实性的东西。其实非常简单。代码如下：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | #region 轮胎滚动与旋转模拟 |
| 02 | **void** WheelRoll() |
| 03 | { |
| 04 | **float** averageAngularVelo = ( **this**.rigidbody.GetPointVelocity(**this**.BwheelModels[0].transform.parent.position).magnitude + **this**.rigidbody.GetPointVelocity(**this**.BwheelModels[0].transform.parent.position).magnitude )/2f; |
| 05 | **float** engineAngularVelo = averageAngularVelo \* 3f; |
| 06 |  |
| 07 | **float** rpm = engineAngularVelo \* (60.0f/(2\*Mathf.PI)) \* (**this**.transform.InverseTransformDirection(**this**.rigidbody.velocity).z > 0f ? 1f : -1f ); |
| 08 |  |
| 09 | *//~ Debug.Log(this.transform.InverseTransformDirection(this.rigidbody.velocity).z);* |
| 10 |  |
| 11 | FwheelModels[0].transform.rotation = FwheelModels[0].transform.parent.rotation \* Quaternion.Euler (RotationValue, **this**.steering , 0);*//旋转* |
| 12 | FwheelModels[1].transform.rotation = FwheelModels[1].transform.parent.rotation \* Quaternion.Euler (RotationValue, **this**.steering , 0);*//旋转* |
| 13 |  |
| 14 | BwheelModels[0].transform.rotation = BwheelModels[0].transform.parent.rotation \* Quaternion.Euler (RotationValue, 0, 0);*//旋转* |
| 15 | BwheelModels[1].transform.rotation = BwheelModels[1].transform.parent.rotation \* Quaternion.Euler (RotationValue, 0, 0);*//旋转* |
| 16 |  |
| 17 | RotationValue += rpm \* ( 360f/60f ) \* Time.deltaTime; |
| 18 | } |
| 19 | #endregion |