Subject: I need help from people who know how to model vechicles
Posted by Deactivated on Thu, 15 Apr 2004 05:46:05 GMT
View Forum Message <> Reply to Message

http://renhelp.co.uk/?tut=15
http://renhelp.co.uk/?tut=33

Quote:How To Set Up a Renegade Vehicle

By Greg Hjelstrom

Introduction

Renegade uses a form of rigid body dynamics to simulate a wide variety of vehicles. All of the geometric characteristics of a vehicle are determined directly from a model which is exported from 3ds-max. For example, the number, placement, and behavior of any wheels in the vehicle is

model: front-wheel drive, four-wheel steering, all-wheel drive, tank tracks, etc can all be set up. Parameters such as the mass, engine strength, and suspension stiffness are set in the Renegade level editor.

Balance

The first thing to do is to position your vehicle model such that its center of mass is at the origin in MAX. In the X-Y plane, the CM (center of mass) should usually be placed exactly at the centroid of all of the wheels in the vehicle. If the CM behind this point, the vehicle will lean backwards on its suspension, if it is in front of this point, it will lean forward, etc. In the Z axis, the CM should be placed some distance below the center of the object to help prevent tipping over when turning corners.

Collision Detection (WorldBox)

As is the case with all objects in Renegade, you can use separate meshes for projectile collision detection and physical collision detection. In the case of vehicles you have to use a single OBBox named "WorldBox" for physical collision detection. (Note: physical collision detection is used when the object is moving in the world, projectile collision detection is used when the code is determining whether bullets hit the vehicle). For projectile collision detection, just use some representative meshes from your model. The WorldBox OBBox must be linked to the origin of the vehicle, and be aligned with the world axes. It should also be as small as possible while still containing the wheels and bulk of the body of the vehicle. The physics code in renegade collides with objects in a buffer zone near the surface of the worldbox so it should not completely contain the model. Below is a screenshot of a WorldBox. Note how the WorldBox does not extend all the way to the front, top or back of the vehicle.

initial position (more on this later).

Wheels
Wheels are defined by adding bones with a particular naming convention into your model. All vehicles, including VTOL aircraft, have wheels. At the bare minimum, there must be two bones per wheel; one to define the contact point of the wheel and one to define the center of rotation of the wheel. The code will use the pivot points of the wheel bones to determine the following things: the initial position of the wheel, the radius of the wheel, the axis that the suspension travels along, and the axis that the wheel rotates about. The two basic bones needed by each wheel are the "WheelP" (wheel position) bone, and the "WheelC" (center) bone.
Below is a view of a wheel from the Humm-Vee model (with the WheelP bone selected). Note that the z-axis points along the direction of travel of the suspension and the x-axis points forward. Also, the pivot point of the WheelP bone must be contained inside the world-box. The initial position of all wheels should be at their extreme topmost point; imagine that the vehicle has just fallen off a skyscraper and landed directly on its wheels.
Attached to the wheel position bone, you need a bone which defines the center of rotation of the wheel; the WheelC bone. Below is a view of a wheel with both bones set up. Again, the important axis is the z-axis, it points along the axis of rotation of the wheel; in the "Top" viewport in Max, the z-axis should be pointing down your monitor (for all four wheels).
The graphical representation of the wheel is then attached to the WheelC bone and will rotate and translate as the vehicle is simulated. The simulation pays no attention to the graphical representation of the wheel or even whether it exists or not. The hierarchical linkage for a simple wheel should look like the one below. Note that the WheelC bone is attached to the WheelP bone; not the other way around.
The 'E' at the end of each of the names above is a flag signifies that this particular wheel applies the engine force at its contact point. Through this naming convention you can create four-wheel drive, front-wheel drive, or rear-wheel drive wheels. Other flags that are available include:

wheel steering).

The complete naming convention for a wheel bone is below. The name always starts with the word 'Wheel', followed by a single character and a two digit number (e.g. WheelP00). Following the digits, any of the above flags can be added.

Wheel {P,C,T,F} {00} [S] [I] [E] [L] [R] [F]

Here are some examples of valid wheel bone names:

applied to it

with the vehicle

Advanced Wheel Settings

You may notice that there are two more types of wheel bones that have not been described yet. These bones can be used to create wheels whose suspension moves in a manner more complex than simply translating along the z-axis of the WheelP bone. First I'll show an example of a translational constraint using a WheelT bone. This example is from the front wheel of the Nod Recon Bike:
The presence of the WheelT bone in a wheel hierarchy causes the wheel to translate along the Z-axis of the WheelT bone rather than the contact point bone. This is used for the front tire of the Nod Recon Bike.
The rear wheel of a Recon Bike rotates along an arm. This can be accomplished by using a WheelF bone. Below is a picture of that situation. The WheelF bone will be rotated about its Y-axis to maintain contact between the wheel and the ground (see the picture for how to set it up). This is probably the most complex type of wheel to create... Look at the Nod Recon Bike for reference.

number)

vehicle) and their contact points must be contained inside the world box of the vehicle.

rolling)

down in the Top viewport)

the engine, append an 'S' to the wheels that steer, etc)

Turrets

Vehicles can have turrets which can fire weapons. The turret aiming motion is generated by the game code as long as certain bones are present in the model. There are three types of bones involved in controlling a turret. The 'Turret' bone will be rotated to set the turret's 'heading'. The 'Barrel' bone will be rotated to set the tilt of the weapon. And the 'Muzzle' bone will be used as the location to create projectiles from. Below is a picture of a the turret from the GDI Mammoth Tank: In the above picture, you can see that there are several muzzle bones. There can be up to two MuzzleA bones and up to two MuzzleB bones. The 'A' bones are used for the primary weapon of the vehicle and the 'B' bones are used for the secondary weapon of the vehicle. All of the bones follow the convention that their axes should be aligned with the world axes; Z is up, X points towards the front of the vehicle, Y points to the left of the vehicle. If you create your boxes in the 'Top' viewport, they will be oriented in this fashion automatically. One improvement over the linkage shown above would be to attach the V_Barrels mesh as a child of their Muzzle bones because the game engine automatically applies a recoil to the muzzle bone (attaching them in that way would cause the meshes to recoil when the weapon is fired).

(up to two MuzzleA and two MuzzleB bones)

the game logic controls the turret (i.e. the turret mesh should be attached to the turret bone so that it will rotate properly)

Engine Special Effects
VTOL vehicles can control bones to display engine effects. Here are some examples of engine effects:

(EngineFlame)

(EngineAngle)

Any bone present in the model which starts with the name "EngineFlame" will translate along its Z-axis along with the vehicle's acceleration. Any bone present which starts with the name "EngineAngle" will rotate about its Z-axis. In the screenshot below, the EngineAngle bone for one

of the Orca engines is selected. Below it is the EngineFlame bone and not shown is a skin which has vertices attached to both bones and stretches as the EngineFlame translates.

Helicopter vehicles can use the EngineAngle bone to tilt their rotors as they fly forwards and backwards. In addition, they can contain 'Rotor' bones which will spin about their Z-axis when in flight. The parameters for how much these bones rotate and translate are all controlled through settings in the level editor.

As a vehicle becomes damaged, the game code can un-hide particular bones in your model. This can be used to show damage by attaching emitters to those bones since emitters in the model will start emitting when the bone they are attached to becomes un-hidden. There is support for three stages of damage, activated when the model reaches 25%, 50%, and 75% damage. When the model loses 25% of its health, all bones whose names begin with DAMAGE25 will be un-hidden. Once the object has lost 50% of its health, all bones whose names begin with DAMAGE50 will also become un-hidden. And when it has lost 75%, the DAMAGE75 bones will un-hide.
