Example MudRunner Assets

Revision 1.0 (2019-02-28)

The assets described by this document are intended to complement the Big Book of MudRunner Map Making by providing example custom features.

This document provides some additional information about the assets, including technical problems and solutions.

Source files are provided where appropriate for 3-D models (in Blender format) and textures (in Gimp format).

Table of Contents

Introduction

Two groups of assets are included:

The Rancho Montaño assets were developed for my Rancho Montaño MudRunner map MudRunner map. As such, these assets have real textures, coherent features, and are tuned for real play. The Blender and Gimp source files are separated out into their own directories so that you can easily copy the desired asset files or directories directly into your mod.

The BBMMM assets were created to explore game support for various asset features and XML properties. As such, these assets have no real coherence, and in many cases the XML files reflect whatever experiment I ran most recently on each asset. The Blender and Gimp source files are directly in the meshes and textures directories to allow fast experimentation, with the assumption that the experimental assets will never be part of a published mod.

Within Blender, many of the cdt meshes are set with a Maximum Draw Type of “Wire” so that they don’t visually intrude on the model. On the other hand, some cdt meshes are given the X-Ray property to make them visible even where they are embedded inside a visual mesh.

Gimp files are provided where I have them, but in many cases the Gimp files provide nothing that isn’t already in the raw texture file. In some cases, however, the Gimp files include multiple image layers, which allows them to be more easily understood and modified.

In many cases I describe how certain feats are performed in Blender or Gimp. However, these descriptions are not generally detailed enough for you to replicate the feat without additional knowledge. Use your favorite search engine to find detailed instructions about how to use the specific tool features that I mention.

Exporting Models from Blender

All of my Blender models are created with “up” in the +Z direction. As such, they need special handling to export properly to MudRunner, which expects “up” to be +Y. I use my own DirectX exporter for Blender, as described in the Big Book of MudRunner Map Making.

Most models are exported with the default settings (e.g. Propagate: “Auto”).

Rancho Montaño Assets

You might find it useful to play the Rancho Montaño map (available on Steam) to become familiar with how these assets are used and how the dynamic models move.

Material

Asphalt

MudRunner doesn’t come with an asphalt material. So I took its concrete texture and adjusted the levels in Gimp to make it look more like asphalt.

Dirt

When MudRunner mangles the junction or sharp bend of a dirt road, I use a dirt terrain material to fill in the gaps. However, the color of MudRunner's built-in usa_dirt material doesn't quite match the color of its usa_dirt_road overlay. So I copied the usa_dirt texture and massage its color balance to get the color I wanted.

Soil

Initially I only modified MudRunner's usa_soil to remove the grass brush, since I wanted a different grass, and I prefer to distribute my grass manually anyway. (With manual distribution, I can remove grass that would otherwise poke through models. Grass looks especially terrible when it pokes through a bridge deck.)

Then I noticed that since MudRunner doesn't draw grass at long distances, the color of the soil at a distance was distinctly different from the mix of soil and grass closer to the camera. So I applied a low-opacity green fill to the texture to make it look better. I copied the original alpha channel to a separate layer mask first so that it wouldn't be altered by the fill.

Concrete Lattice

_rm_concrete_lattice is intended to emulate concrete paving stones. These are sometimes used in a real life driveway to allow grass to grow around them while providing support for a vehicle. In the Rancho Montaño map, they provide a temporary ground support for trucks while the highway bridge is being worked on.

Of course, the material itself does not prevent a truck from sinking into mud. It simply provides the player with a visual indication that there's no mud in that area.

I probably could have done something similar with an overlay, but it would have been a nightmare to disguise its junction with another road overlay. Using a material nicely solved that problem while also allowing the material to be placed with an arbitrary outline.

The concrete lattice material takes advantage of how MudRunner mixes materials with alpha channels. When a terrain tile with two materials has a region of mixed material strength, it mixes the materials in part based on their relative alpha values. The concrete lattice has a mix of fully opaque and fully transparent alpha values so that it is laid down correctly with any moderate material intensity.

The material texture is simply MudRunner's concrete material with its alpha channel replaced. I created the alpha channel by making a single X shape with appropriate dimensions, tiling it across the texture, and then using Gimp's Filter → Noise → Spread tool to add some graininess to the alpha edges.

Distributions

The brushes.xml file contains my custom distribution brushes. This is actually my archive file that preserves my changes in case the master brushes.xml file gets trashed in an update. The brushes in the custom file must be copied into the Media/prebuild/common/brushes.xml file in the MudRunner Editor's application directory in order to be used.

Custom Density

Many of the brushes simply alter the default distribution density. I could instead reduce a plant's density by drawing its density with a less intense brush, or I could increase a plant's density by distributing it multiple times in the same area. But it's just a lot easier to make a custom brush to do what I want.

The following custom brushes primarily change the density of distribution (custom vs. original). Other small tweaks are also mentioned below.

- _SparseStumps (16) vs. Stumps (32)
- _SparseRocks (8) vs. SmallRocks (700) [also removes LinkedGrassBrush="DebrisStones"]
- _RiverTwigs (120) vs. SmallTwigs (320)
- _ConiferTwigs (20) vs. SmallTwigs (320)
- _ConiferBushes (64) vs. BushesBig (64) [removes LinkedGrassBrush="GroundLeaves"]
- _Cones (25) vs. Strobiles (200)
- _SmallRipRap (4096) vs. DebrisStones (1200)
- _DenseGrass (1000) vs. GrassLong (700)

Sparse Pines

The _SparsePines brush is copied from the PinesSmall brush, but allows the small pines to be much larger.

Riprap

The _RipRap brush (which I capitalized incorrectly) is intended to create a jumble of overlapping rocks. It is similar to the SmallRocks brush, but has **much** greater density, the rocks have more variation in size (accomplished using the ScaleRandom property instead of MinSize and MaxSize), and the rocks are allowed up to 90° of divergence from vertical. The _RipRap brush uses LinkedGrassBrush="_SmallRipRap" to mix in additional small stones.

The high density of riprap of course makes it unsuitable for large areas. The Rancho Montaño map uses it only around the concrete pipe culverts to prevent water erosion.

Conifers

The _Conifers brush was originally intended to include everything I wanted in a conifer forest. As such, it contains both regular and small pines. It also includes LinkedGrassBrush="_Cones".

I then realized that I wanted to specify a different number of PlantsPerBlock for various plants in the mix. So I created a separate _ConifersAlt brush to hold the lying firs and dry pines. The dry pines are allowed significant Divergence to make them more interesting.

My conifer forest also includes the _ConiferBushes, _SparseRocks, and _ConiferTwigs brushes mentioned above.

River Trees

The _RiverTrees brush was a significant undertaking to try to emulate the distribution of trees around a river bank. It is designed around the assumption that the distribution is drawn with small scales low in the river valley and large scales high on the river banks, with low to moderate randomization of scales (not the full range) throughout the distribution.

As such, the _RiverTrees brush makes significant use of the MinMapScale and MaxMapScale properties. Large trees and small dry trees are only allowed at large scales. Small leafy trees and bushes are allowed only at small scales.

I also assume that the distribution is drawn with decreasing density in the lowest (and most often flooded) areas as well as the highest (and driest) areas.

The _RiverTrees brush is paired with the _RiverTwigs brush to distribute small twigs throughout the range of river trees, but again less dense in the lowest and highest areas.

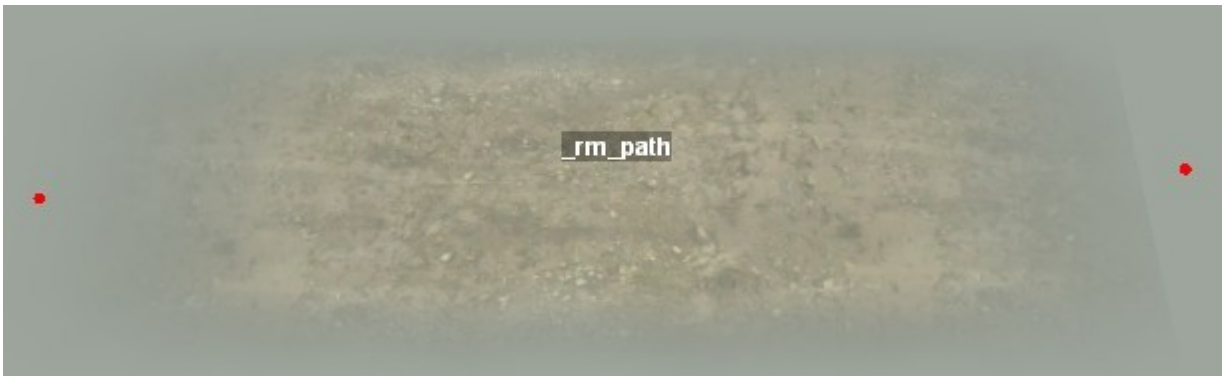
Roads

Double Track

_rm_double_track is simply the built-in usa_dirt_road texture with the grass_road alpha channel. I copied the alpha channel by transferring it to a layer mask, then copied and pasted the layer mask.

BTW, you can add grass to the center of the path by creating a separate grass distribution with Ignore Overlays: True. The distribution resolution isn't really fine enough to restrict the grass exactly to the center of the path, but it's close enough to look sufficiently like an ill-maintained road.

Path



The built-in dirt_path looks too developed to me. _rm_path is simply a copy of dirt_path with the entire alpha channel set to 75% opacity. This allows the underlying terrain material (_rm_soil) to shine through.

Invisible Roads

I created four invisible road overlays:

- _rm_invisible_asphalt
- _rm_invisible_dirt
- _rm_invisible_path
- _rm_invisible_bridge

These road overlays use Layer=”-1”, so they are almost always discarded in favor of a higher-priority road overlay. The primary purpose is simply to have the editor draw these “roads” on the navigation map.

I create most of my road junctions with the lower-priority road ending prior the edge of the terrain block so that there isn’t an abrupt edge to the road. I then fill in the gap with an appropriate material. However, this means that there is then a significant gap in the road on the navigation map. Using an invisible road to fill in this missing gap works great. I also rearranged the order of road overlays in the map’s XML file so the roads would draw onto the navigation map in the desired order. It’s a huge pain in the butt to get the invisible roads positioned to overlap just the right amount, especially with angled road junctions, so I’m not sure that I’ll ever do it again.

A secondary benefit is that the invisible road can continue the HeightmapOffset of the road across the filler material. I made sure to set ApplyOffset to “true” in the editor for invisible roads that continued the road in the correct direction and to “false” for invisible roads that were angled for specialty situations.

Texture

Road overlays don’t have a DiffuseMultiplier property that could make them a uniform color with alpha equal to zero. Instead, all invisible road overlays share an 8×8 fully transparent texture.

Static Models

Stone Wall

The stone wall models are designed to supplement the stone wall overlay, described later.

The stone wall models come in three pieces:

- _rm_stone_pillar
- _rm_wall_rancho
- _rm_wall_montano

I don’t remember why I separated the pillar out into its own model. It could also have been attached to both ends of each wall model.

Both walls use the same mesh, _rm_stone_wall.x. A separate XML file for each tells one to use the “Rancho” texture and the other to use the “Montaño” texture. These “sign” textures float just above the surface of the wall (as they would be for a mounted sign) and use AlphaKill=”true”.

Blender

Each Blender model was built from a cube, stretched to the correct shape. I then applied the scale to the mesh, but this could also have been done during export.

As an optimization, I cut the bottom face off all of the models (which is below ground level), and I cut off the end faces of the wall (which should abut a pillar).

Each cube originally had two materials, one for the concrete top, and one for the stone sides. However, since the stone wall overlay is limited to a single material, I updated the models to match so that they could share the same texture. The UV map for the top of the wall uses coordinates 11x the width of the texture to force it to be tiled across the top of the wall. The sign is a separate mesh with its own texture.

Textures

The stone texture is derived from the one here. Although I liked the top of the supplied wall, it didn’t seem practical to to try to morph the flat texture into a 3-D wall, so I just cut it off. I then replicated the texture to make it wider and used Gimp’s “smudge” tool to blend the pieces together reasonably smoothly. (See if you can

find the join!) I also used Gimp's "cage" tool to warp the wall a bit so that the top and bottom of the texture were perfectly straight.

The concrete texture is from here, simply cut into a square shape.

The stone and concrete textures are combined into a single texture file, with the stone texture stretched to fit the remaining space. Since the texture was originally created for the conforming overlay, it can be found in overlays/_rm_stone_wall_overlay__d.png.

The sign textures were drawn with Gimp's text tool. The font is Great Vibes. The two textures are in models/_rm_rancho_sign__d_a.png and models/_rm_montano_sign__d_a.png.

Speed bump

Blender

The speed bump is a sphere with the bottom cut off, and with the right side extruded and stretched far away from the let side.

I copied the visual mesh to the cdt mesh, then reduced its polygon count by manually removing vertexes & edges.

With just a diffuse texture, the speed bump looks a bit angular depending on how the light hits it. I addressed this by creating a normal map for it, thus smoothing the model from a lighting perspective. To do this, I copied the visual mesh to a separate Blender layer and applied a "subdivide" modifier to the copy. I then baked a normal map to reflect the difference in normals between the high-poly mesh and the regular mesh.

The high-poly mesh shouldn't be exported. Therefore, I "Export Selected Objects Only" with the "cdt" frame, "visual" frame, and their parent "speedbump" frame selected, but not with the "visual_hp" frame selected.

Texture

The speed bump is placed on dirt and on the _rm_asphalt material, and I want it to not have mismatching asphalt, so the speed bump texture is simply a piece of the _rm_asphalt texture. I then painted semi-transparent yellow bars across it. They are on a separate layer, so the base asphalt texture can easily be swapped for something different.

The various texture warps and cuts at the ends of the speed bump gave me trouble, so I just made them all yellow. I realized later that Blender provides a way to paint directly on the 3-D model, and it will update the UV texture appropriately. I'm not sure if that would have allowed me to create better-looking ends to the speed bump.

Concrete Pipe

Blender

The model is a cylinder, stretched, morphed, extruded, and hammered. (Hammered is not a technical term.)

I made multiple attempts to create this with a normal map similar to the speed bump. However, I eventually realized that I want the concrete texture to wrap seamlessly around the pipe, but the normal map needs to have gaps between each polygon so that the normals from one polygon to leak onto the adjacent polygon. (This probably probably afflicts the speed bump, but I never noticed it as much there.)

I finally rebuilt the concrete pipe simply using more polygons.

When the pipe model is stuck partially into the ground to emulate a culvert, then the player can potentially look into the pipe and see the terrain blocking it. I added a concrete plug to the pipe to try to hide this, although it's probably not very successful. Something that looks more like the darkness of shadow would probably be better. I didn't try too hard, though, because my pipes are relatively small diameter, and it's hard for the player to get a good angle to look into them on the Rancho Montaño map.

Texture

The concrete texture was simply copied from MudRunner's concrete material texture. (Although now that I look at it again, the two textures don't really match. I think MudRunner may have updated its texture since I made my copy.)

MudRunner's material has a alpha channel that isn't useful for the model. Of course, the alpha channel is ignored in MudRunner without AlphaKill="true", but it makes it a pain to work with the texture in Blender. If you simply delete the alpha channel in Gimp, it fills the transparent sections with the background color, which is not correct. Instead, I transferred the alpha channel to a layer mask, then deleted the layer mask.

Dynamic Models

Barrel

MudRunner's barrels are static models. But I wanted to put my barrels out where they could easily get hit by a truck, and it's very unnatural if hitting a barrel (even a filled one) causes huge damage and brings the truck instantly to a halt. So I made a dynamic barrel using MudRunner's barrel texture.

Although the barrel is mobile, I elected to not put DynamicModel="true" in its class XML file. Because most trucks simply roll over the barrel pretty quickly, it doesn't tend to move very far. With DynamicModel="false", the barrel's position gets reset when the player moves far enough away from it.

Blender

The barrel is a simple cylinder. I sized it according to the proper dimensions of a 55-gallon drum, not the wee little barrels that MudRunner gives you at Scale=1.

After having the same trouble with the normal map as I had with the concrete pipe, I tried a more sophisticated solution for the barrel. I stretched the barrel texture at 6 seams, and then arranged the UV map to have a small gap between polygons at each seam. This arrangement of the UV map means that normals don't leak from one polygon to the next, so the barrel looks a lot better.

The normal map was baked from a high-poly mesh named "visual_hp". As with the speed bump, this mesh must be excluded when exporting the model.

Texture

MudRunner includes the barrel texture in its “gas_station_stuff_01” omnibus texture. I copied the barrel pieces out into their own small texture for faster loading. I also warped the barrel side texture as described above.

Barricade

Before I created this, I had to first research what it’s called. Apparently it’s a “Type I Barricade”. (A type II barricade would have two safety reflectors on each side instead of just one.)

The barricade is hinged at the top so that the two sides can be pushed together, while a constraint prevents them from being pulled apart.

Although the barricade is mobile, I elected to not put DynamicModel=”true” in its class XML file. Because most trucks simply flatten the barricade, it doesn’t tend to move very far. With DynamicModel=”false”, the barricade’s position gets reset when the player moves far enough away from it.

Blender

The barricade is mostly made of cubes stretched into legs and boards. The origin for each body is placed at the top so that a hinge can join them.

I cheaped out on the chains between the two sides. These are deformable meshes that simply shrink as the two sides of the barricade collapse together.

All hierarchical transforms must be pushed through to the barricade's deformable mesh, but the location and rotation must remain in the hierarchy for use by the hinge constraint between the two sides. My exporter automatically detects and handles this case with Propagate: "Auto".

Texture

Plenty of barricades of this type are made from plastic, and plastic can be modeled simply as a diffuse color without any texture, so that was easy. I also made the chains a sightly darker gray, again without any texture.

I copied the texture for the reflector from MudRunner's "models_road_signs_02" omnibus texture. Interestingly, there are no MudRunner models that actually use this portion of the texture.

I tried everything I could think of to make the texture actually reflective at night (brighter in the truck's lights than surrounding material), but I couldn't find any combination of properties that accomplish this. Oh well.

This texture is the first one described here that I put in my own omnibus "_rm_road_signs" texture. I was getting little hiccups while driving around my map. I still don't know what was causing it, but combining many of my small texture files into a single texture file with power-of-2 dimensions may have helped.

Signs

With the experience I'd gotten from the other models, I whipped these signs together pretty quickly, and I feel like they really help sell the "California" aspect of the map.

The graffiti on one of the speed limit signs is pretty obvious. You have to look closer at the other speed limit sign to see the bullet holes.

BTW, in keeping with the fictional aspect of Rancho Montaño, G49 is not a real county route in California. The G series of routes is appropriate for Santa Clara County, which is the notional home of Rancho Montaño. "49" alludes to California's gold rush of 1849, the San Francisco 49ers football team (which is now based in Santa Clara), and state route 49 (which is a wonderful road in the foothills of the Sierras).

All of the signs are breakable models. Their physics are copied pretty much directly from MudRunner's built-in signs, although I raised their centers of gravity to behave more naturally when they're broken. (The MudRunner signs have a concrete plug that gets ripped out of the ground, and perhaps that's why they made their center of gravity so low. But I never see that plug when I barrel into a sign, so it's really weird to see the sign get plastered high on my grill and eventually get sucked under the truck when I expect it to go bouncing over the top.)

Blender

Each sign has a folded sheet of metal for the post (rather than a round pole). Each sign is a pair of planes pointing in opposite directions. (They're not a single TwoSided polygon because I need a different texture for each side.) I didn't bother modeling an edge to each sign.

I included a cdt mesh for the post, the main sign, and the "truck" sign if present. I didn't bother with a cdt mesh for the smaller "G49" sign.

Textures

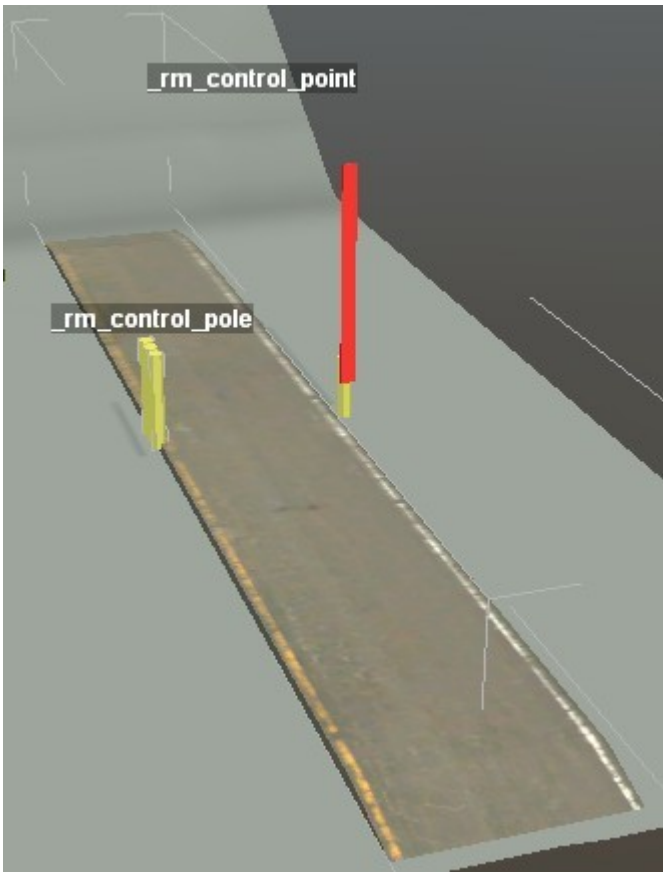
I build all of the textures into a single texture file for efficiency. The original textures are in gimp/models/road_signs, and the final omnibus texture is "_rm_road_signs__d_a".

I copied the texture for the rear side of the signs from MudRunner's "road_signs_02" omnibus texture. For the rectangular signs, I cut the corners off with AlphaKill="true". The various signs have different aspect ratios, but it's close enough to look fine. I made a separate copy for the G49 sign with its own AlphaKill pattern.

The standard highway signs are available in various non-copyrighted formats directly from the government.

The G49 sign was pasted together from multiple county route signs which were kindly put into the public domain for use on Wikipedia.

Control Point



The control point is a (perhaps overly ambitious) attempt to prevent trucks from passing through while allowing lighter passenger vehicles. When a heavy truck drives onto the model, its weight pulls the red gate down to bar the path.

This model is definitely best understood by giving it a try. It is on the main highway of the Rancho Montaño map, not too far from the starting truck's location. Or create your own sample map to try it out on.

The full structure is comprised of one _rm_control_pole (which is actually 3 poles) and two copies of _rm_control_point. All three models share the same origin so that they can be placed exactly together in the MudRunner Editor. One of the _rm_control_point models is then rotated 180° to cover the opposite lane of the road.

_rm_control_point has four movable bodies:

The "plate" is the 2-D plane with the road texture on it (excluding the ramps at each end). It uses a Prismatic constraint to move only up and down, and it has a huge AngularFriction value to prevent trucks from tilting the plate against its constraints. The plate has a fairly large mass, so that it can win physics fights with the other movable bodies, buts its GravityFactor is set to 0 for simplicity. The plate's constraint uses a Force motor to

return to its base position with a constant force that is stronger than the weight of a passenger vehicle but weaker than the weight of a truck. Thus, a truck pushes the plate down, and a passenger vehicle does not.

The “gate” is the red bar that swings down in the truck’s path. It uses a Hinge constraint for its swing action. It is not directly attached to the plate.

Attached to the plate is a “pivot”, which is an invisible body with Collisions=”None”. The pivot has a Hinge constraint to its parent plate, and it has a Ragdoll cross-constraint to the gate. The hinge constraint ensures that the pivot moves up and down with the plate while allowing it some angular freedom. The ragdoll constraint ensures that the pivot also stays a constant distance from a slightly offset point on the gate, which pulls the gate down when the pivot moves down (and pushes the gate up when the pivot moves up). I would have preferred another hinge constraint here, but I needed a working PivotOffset in order to pull on the side of the gate and not its origin.

The final body is an “extension”, which is another 2-D plane attached with a Prismatic constraint to the “plate” body. The only purpose of the extension is to make sure the model’s physics remain enabled until the gate can return to its fully upright position. Therefore, the extension is extremely light and has a very loose constraint. When a truck drives over it, the extension simply gets flattened against the ground and applies no force to the parent plate. But because that constraint is there, the plate’s physics are enabled as long as the truck is touching the extension.

The truck can’t push the plate below ground level, so the plate must start slightly above ground level. A static ramp at each end of the plate allows a truck to climb up to the plate level without a damaging bump. Note that the plate’s cdt mesh will pass through the ramps’ cdt meshes. Fortunately, that is possible because MudRunner does not check for collisions among bodies within a model.

The tuning of the physics is a hard problem because a very limited movement of the plate must quickly yank the gate through a large movement. The gate must have a reasonable mass to prevent its constraints from being torn apart by a truck, but that mass also endows the gate with a large moment of inertia. Theoretically I could zero out the moment of inertia by offsetting the gate’s center of mass directly at its pivot, but I had limited success with that. Either I don’t fully understand the details of MudRunner’s center of mass calculation, or perhaps MudRunner refuses to reduce the moment of inertia too far.

Obviously, the physics work out better with a heavier truck than with one that is not much heavier than the plate’s Force motor. Then there is the problem with trucks such as the B-66, which is only 2000 kg, compared to the Hummer H1 which is 3500 kg. I “solved” that problem by not including a B-66 on the Rancho Montaño map. Some trucks also have an issue where the weight of a single axle is not enough to activate the plate. That’s why I had to stretch it to be long enough to reach the rear axles of the long American cabs. As it is, if a truck barrels quickly up to the control point, the gate tends to swing down when the truck is partway through the gate. The gate ends up smashing violently into the windshield and doing a ton of damage to the truck. But I choose to believe that that’s a fair punishment to the player. She should pay more attention to those “Truck Detour” signs.

Once a truck has left the plate, the force under the plate must also be strong enough to push the gate back up reasonably quickly.

Having the _rm_control_pole as a separate model is helpful because when a truck pushes on the gate arm, the gate gets pushed into the pole. Since the pole is a separate model, the two models collide, and the gate can't be pushed very far against its constraint. The flat side of the pole also prevents the gate from twisting under pressure.

_rm_control_pole is three poles. The primary reason for this is to keep a truck from approaching the gate at a strong angle and thus getting a nose under it before the plate activates. If a truck ever gets a nose under the gate, the lever action of the gate can easily lift the plate even with the full weight of the truck on it.

Even after all the tuning and technical measures taken, it is certainly possible for a motivated player to get a truck through the gate. But on the Rancho Montaño map, there's very little to gain by doing so, so whatever.

Blender

After all that was said above, there's not much more to say about the Blender models.

The plate's visual mesh includes side panels so that it doesn't visually look like a 2-D plane. These visual polygons pass into the ground when the plate is pushed down.

While the visual meshes for the ramps come to the full height of the plate, the cdt meshes for the ramps are slightly shorter. This ensures that the trucks wheels make solid contact with the plate as soon as they reach the top of the ramp. Otherwise we'd have to wait for gravity to pull the wheels down from their rising arc off the ramp before the plate and gate would activate.

The cdt meshes of _rm_control_pole's poles do not extend all the way to the ground because doing so would intersect _rm_control_point's plate and extension. In any case, the player will never be able to notice the cdt gap, and the visual mesh does reach all the way to the ground.

Texture

Rancho Montaño places the control point overlapping a usa_asphalt_road_double_2 road overlay and extending onto a bridge_road_02 model, which also uses the usa_asphalt_road_double_2 texture. Therefore, that is the texture I used for the plate and ramps of the control point. I cut the texture in half (since each control point covers only one lane), and I replaced the transparent edges of the road with more asphalt texture to give myself more room for the shoulders of the plate and ramps. (And without those transparent edges, the texture can be optimized without the alpha channel.)

The UV map for the plate uses 1.5 tiles of the road texture to cover the plate with the proper aspect ratio. The UV maps for the ramps were carefully adjusted to seamlessly continue the same texture. Unfortunately, they

don’t quite line up with the bridge since the MudRunner folks weren’t as careful when attaching the texture to the bridge. But it’s not bad.

The gate and the various poles are simple untextured colors.

Both the pivot and the extension bodies have a visual mesh because MudRunner can’t generate a body without it. However, the color of these meshes is set to fully transparent, so they can’t be seen. You can change the color if you want a bit more visual evidence of the control point’s mechanisms.

Grass

Poppies

You’ve already seen this model in the BBMMM.

There is a California Poppy model here, but it is way too many polygons to use as a MudRunner grass. So I took the textures from that model and remixed them to allow a simple pair of polygons for each flower's leaves and a cup of 4 triangles for the petals. All polygons are TwoSided="true", which means that the bottom of the petals looks the same as the top (including the various pollenization parts in the center). Fortunately, the player can't get out of her truck to look under the flowers close up, so that's OK.

The result still has many more polygons that most MudRunner grasses, so I tried to distribute it over wide areas only with low density, or with high density in only small areas. This is sufficient to give the grassy hills a California flavor without being oversaturated with flowers.

Plants

Hedge

I've wanted a hedge for MudRunner for a while. I'm not super satisfied with this one, but it's OK.

The cdt mesh is smaller than the visual mesh and is embedded inside the hedge. This means that a truck can stick a nose into the foliage before starting to push on the plant. I experimented with IsCapsuleCDT, which does something similar, but never got a satisfactory result. (I'm not particularly satisfied with how MudRunner's built-in bushes work, either, so maybe I'm just picky.)

The top of the hedge tilts on a Ragdoll constraint with its pivot offset to 3 meters below ground. Thus, pushing the hedge horizontally a large amount causes the top to tilt only a small amount. This approximates my experience with hedges; they tend to keep their local shape and orientation when being pushed around rather than being uniformly distorted. See also my discussion of vertex weighting below.

The ragdoll constraint allows zero twist. A long hedge such as this would normally be comprised of several intergrown bushes, each with its own trunk, so twisting the hedge monolithically would look really weird. Of course, a small amount of twist against the constraint is still possible, but I gave the hedge enough mass so that the constraint is fairly strongly enforced.

The ragdoll constraint has a very strong spring on it. A steady push from a jeep does almost nothing. A jeep requires a running start to shift the hedge noticeably, or a big truck such as a Maz can give the hedge a strong push.

The hedge is tuned to break into chunks only if it gets a hit by a heavy truck with a running start. It seems to be particularly vulnerable to being clipped at a corner, especially by a truck's trailer. To reflect the large size of the hedge, it breaks into 6 bush chunks.

The hedge is a long rectangle, but MudRunner places plant landmarks with a random orientation, so that rectangular shape can't be transferred on the map. Instead I use the standard circular tree landmark, colored and sized appropriately for the hedge. Since I always place the hedges in a repeating line on the map, the regular row of landmarks does a decent job of demarking the hedge locations.

The hedge includes a winch socket mounted low and centered. Multiple winch sockets would have been nice, but MudRunner doesn't support that.

Blender

As is required for plants, there is only a single "plant" mesh. It has two materials: one for the sides and one for the top and bottom. (Although with the hedge attached to ground, the bottom is probably never visible.)

The top and bottom are indented slightly so that the alpha cutouts of the side textures don't allow you to look inside the hedge.

As is required for a multi-body plant, the vertexes of the single plant mesh are weighted to the different bodies. The bottom of the hedge is fully weighted to the static "root" body so that it doesn't move. The vertexes near the top of the hedge are fully weighted to the "hedge" body, so they follow the position and rotation of the body as it is pushed around.

I subdivided the sides into multiple strips so that I could customize the weights of the vertexes between the top and bottom of the hedge. I had in mind a bit of an S curve to the hedge as it is pushed: the bottom portion remains vertical, the middle is most strongly angled where the truck is pushing on it, and the top returns closer to vertical. The vertex weights reflect this. In particular, the topmost vertexes actually have less weight than ones a little bit lower down in order to slightly reduce the position and rotation influence of the hedge body.

Textures

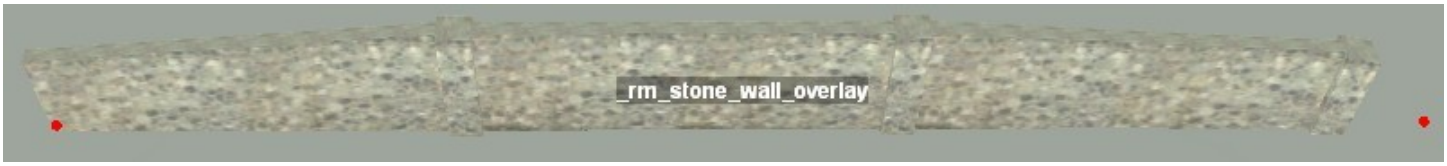
I no longer remember where I got the textures from. Google image search, probably. I passed the top texture through a context-sensitive tiling tool to make it wide enough for the edge. For the side texture, I meticulously cut out all of the ground and deep shadow at the bottom and the sky and thin leaves at the top.

Although the hedge looks fine in Blender, it looks way too bright in MudRunner. So I used the DiffuseMultiplier property to greatly decrease its brightness.

The default clipping behavior allows the camera to look inside the hedge when it gets close, and that looks really weird. The hedge isn't tall enough to pose a real problem to the player's view, so I just set IsNearClip="false".

Conforming Overlays

Stone Wall



The stone wall overlay combines one _rm_stone_pillar model and one _rm_stone_wall model (without the sign texture) into a single conforming overlay.

The overlay isn't great at sharp corners, since that tends to distort the pillars. The Rancho Montaño map has one sharp corner which it accomplishes by ending the overlay at a pillar model, and then using an individual wall model to turn the corner.

MudRunner decreases the overlay's height at the ends. However, the height change is minor compared to the height of the wall, and it can be disguised by having the overlay meet a pillar model which is slightly embedded in the ground so that they have the same height.

Bumpy terrain tends to distort the vertexes at the top of the wall, which is especially noticeable at the tops of the pillars. I used the Flatten brush on the terrain height to smooth the terrain around the base of each pillar and thus flatten the tops of the pillars. This also puts the two sides of the wall at roughly equal heights, so the only tilt along the wall section is lengthwise (to match up with the height of the next pillar).

Grass and plant distributions have an unfortunate tendency to be relocated to the top of the wall. They also fly over the top of the wall where the ground height doesn't follow the same path as the interpolated wall segment height. For this reason, I cleared all grass and plant distributions from my map below the entire length of the overlay.

Blender

The 3-D mode for the stone wall overlay is simply a combination of the stone pillar and stone wall models (removing the floating plane for the sign texture). MudRunner correctly categorizes the combined cdt mesh for the pillar and wall as “mmop”, so nothing special needed to be done there.

As in the related models, the stone wall overlay extends below ground level. When the overlay is stretched over a local low spot on the ground, these lower areas of the model become visible. In a few cases I raised the ground height on the map to fill gaps below the wall. (And in other cases I lowered the ground height to prevent it from covering too much of the wall.)

Texture

A conforming model does not support multiple materials. Therefore, the stone and concrete textures are combined into a single texture file. The description of the related static models above includes a more detailed description of the texture.

Horse Fence

The horse fence is a high visibility fence designed to enclose horses. I arranged its dimensions to match the horse-safety guidelines I found on the internet. In particular, since the fence runs along a highway, it is on the tall side to prevent any possibility of being jumped by a horse.

I found different answers for which side of the fence the posts should go on, so I erred on the side of horse safety and put the posts on the outside of the enclosure where a horse can’t run into it.

As with the stone wall overlay, the horse fence overlay has trouble looking good on sharp corners. I tried a few arrangements of the end post and went with the one that looked best, but I also didn’t put any sharp corners on the Rancho Montaño map.

As with all conforming overlays, the ends of the overlay droop. If I had a horse stable model, I could hide the fence ends in that. Instead, I chose to hide the drooping ends behind a boulder.

The horse fence isn’t super complex, but it has enough polygons that it seemed worthwhile to reduce their count when viewed at a distance. So I created a “far” mesh with just four TwoSided planes as the boards, and an X of TwoSided planes as the post. (I could have also created a single TwoSided plane for all four boards, using AlphaKill to cut gaps between them, but I didn’t feel that ambitious.)

The cdt mesh is a single plane for all of the boards and another single plane for the post. A truck can intrude slightly into the post before encountering this plane, but not enough to be noticeable. The post plane also sticks into the ground, while the boards plane is raised above the ground. This creates enough difference that MudRunner correctly interprets the two planes as an “mmop” shape and not a “box” shape.

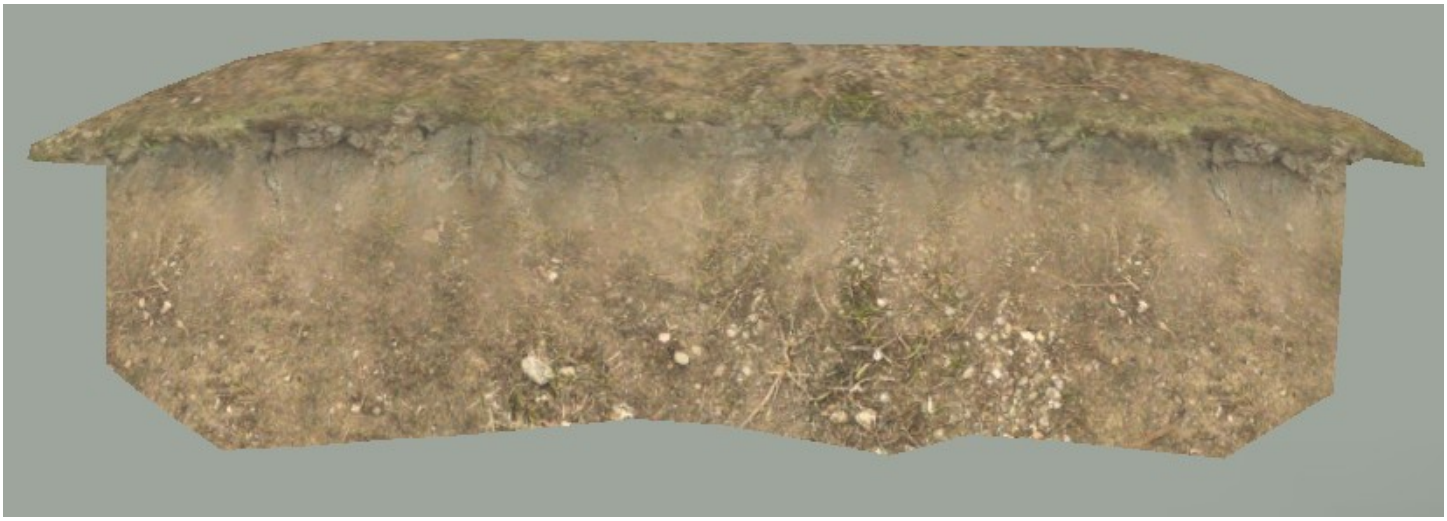
Texture

The board texture was copied from one of the built-in MudRunner textures, and I applied a semi-transparent white color to it to emulate white paint. I also moved the board texture to the omnibus “_rm_road_signs” texture so that it doesn’t have to be loaded separately.

The MudRunner editor shows some orange leaking onto the fence at moderate distances. Some leakage between texture pixels can be expected when the texture is down-sampled for the mip maps, but I wouldn’t expect leakage from the top of the texture to the bottom of the next tile. I went ahead and adjusted the UV map to not use the bottom 4 pixels of the board texture, and it the orange is still leaking, so I don’t know what’s up with that. Fortunately I don’t notice it when playing the game, though.

I had some trouble with the fence appearing visibly narrower at far distances. I eventually tracked down the problem in the far mesh, but in the meantime I’d changed the far mesh to use an untextured gray color. This looks fine and possibly could perform better, so I left it that way.

Cliff



I didn’t like all of that super green grass at the top of MudRunner’s built-in cliffs, so I made my own texture without it. Unfortunately, there is no way to apply my own texture to a built-in model, so I had to make my own model, too.

I tried to copy the approximate shape of the built-in cliff model as well as I could. But while I was at it, I reduced the variation at the bottom of the cliff because it tends to look weird when placed close to a road.

Blender

The overlay model simply has the “overlay” mesh and the “cdt” mesh. I created the cdt mesh by applying copying the overlay mesh and using “Limited Dissolve” to reduce its polygon count. More could be done, but I don’t have enough cliff in the Rancho Montaño map to bother even thinking about it.

Texture

The texture is built from two built-in textures: the usa_cliff_soil overlay texture and the usa_soil material texture. I used a layer mask to blend the usa_soil texture over the grassy part of the usa_cliff_soil so that the cliff would have soil at both top and bottom.

I had a bit of trouble with the lighting under the lip of the cliff when it is applied to a steep surface, possibly because the model is stretched, but the polygon normals aren’t adjusted to match. I added BacksideLighting=”0.7”, and now it looks OK (in the one place I used it).

Navigation Map

I wanted to add some text to the navigation map. You can find the Gimp file with my modifications in “gimp/map/level_rancho_montano.xcf”.

In the Gimp file, I have loaded the auto-generated map from the levels directory, transferred the alpha channel to a layer mask, and disabled the layer mask to make the map fully visible. I added the text (with partial transparency), and flipped it vertically to match MudRunner’s goofy coordinate system.

I initially wanted to make the text red, but MudRunner renders the navigation map without color until it is explored. Driving a vehicle across the bridge would make part of the text red while leaving the rest dark gray, which just looked goofy. So I just made the text black so that it would stand out either way.

The export process is made complicated by Gimp's limitations and by MudRunner's expectations for the alpha channel. When exporting to DDS, Gimp only exports the currently selected layer. But merging layers does a poor job when the base layer is fully transparent. So here is the export process I use:

Select the (disabled) layer mask and press ctrl-C to copy it.

Image → Merge Visible Layers (ctrl-M).

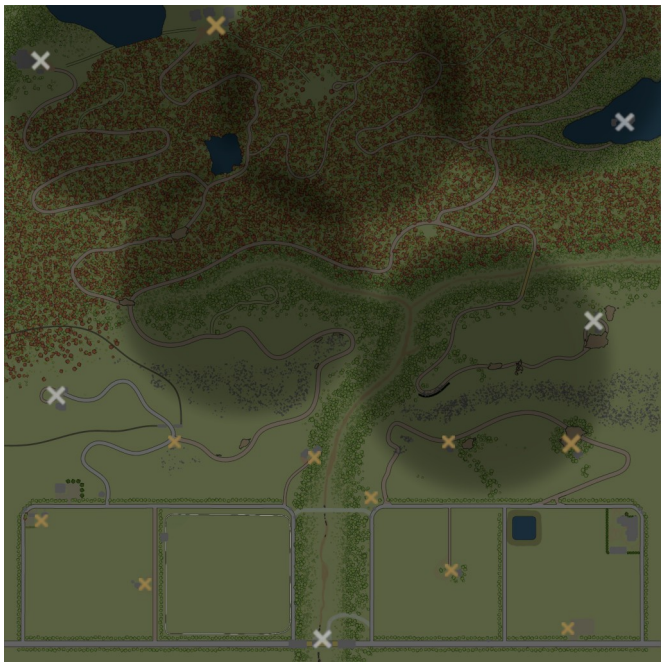
Layer → Mask → Add Layer Mask.

Press ctrl-V to restore the old layer mask.

Layer → Anchor Layer (ctrl-H).

File → Export As (ctrl-shift-E). Save back to to the original DDS file with the following options:
- Compression: BC2 / DXT3.
- Mipmaps: Generate mipmaps.

Steam Workshop Preview



I wasn’t satisfied with the preview images autogenerated by the MudRunner Editor. You can find the Gimp file with my modifications in “gimp/map/preview big.xcf”. I chose to generate the preview without my edits to the navigation map (above) since the text is illegible at preview size.

The highest resolution preview image is in preview_wide.png, but it includes blue bars on each side to fit the standard screenshot area in the Steam Workshop. preview.png is the same as preview_wide.png, but scaled down to fit within a 512×512 square. But this scaling seems dumb since it brings the blue bars on each side with it, and then has to add blue bars to the top and bottom to make it square again.

I started my modified image by cropping the interesting 512×512 section from preview_wide.png.

I added diffuse black circles to represent the areas occluded by watchpoints. I initially wanted to make them much more opaque to prevent players from “cheating” by peeking at the preview map. However, this caused the Rancho Montaño preview to show only the “boring” flat areas with most of the interesting tree-covered areas obscured. So I relented and decreased the opacity of the black circles. To avoid hiding the log stations and lumber mill objectives, I copied their X markers to a separate layer that I placed above the layer with the watchpoint circles.

For some reason the autogenerated preview doesn’t show scavenge kiosks. I copied one of the log station X markers to its own layer and reduced it in size a bit. I then replicated that layer 8 times and positioned each layer appropriately for the location of a scavenge kiosk. I copied the scavenge kiosk coordinates to their own file (“scavenge.pos”) and used an awk script (“scavenge.script”) to convert them to image coordinates. This script hard codes the following values:

- The map size is 1024×1024.
- The preview map image size is 508×508, offset by 2 pixels from the top left.
- The X layer for the scavenge kiosk is 12×12.

Then I experimented a bit and realized that the Steam Workshop can render a larger screenshot perfectly well when the user clicks on it, so a large preview would show off my map better. So I copied level_rancho_montano.dds as the new base layer (with its alpha channel removed) and scaled up the remaining layers to match it.

Finally, I added the tilted Rancho Montaño text over the image.

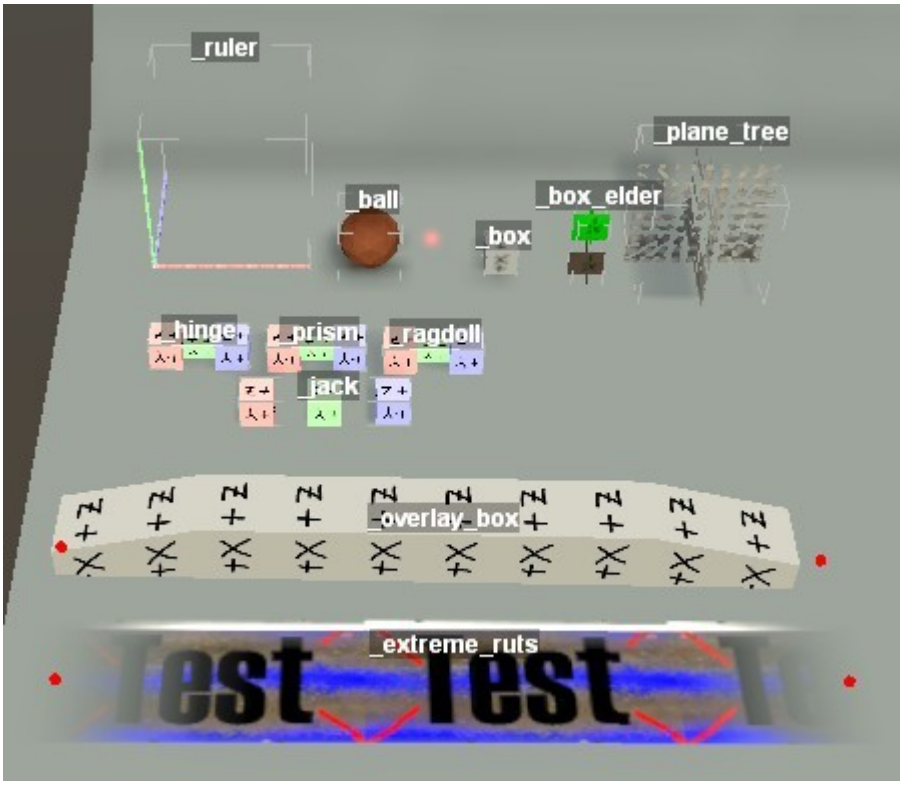
To export to preview.png, I make the Rancho Montaño text visible and scale the image down to 512×512. (A larger image appears to make the publish process silently fail.)

To export to preview_wide.png, I make the Rancho Montaño text invisible, but export the image at full size.

BBMMM Assets

The BBMMM assets were created to explore game support for various asset features and XML properties. As such, these assets have no real coherence, and in many cases the XML files reflect whatever experiment I ran most recently on each asset. Their names often fail to reflect their current purpose, and the texture filenames don’t always follow the MudRunner guidelines.

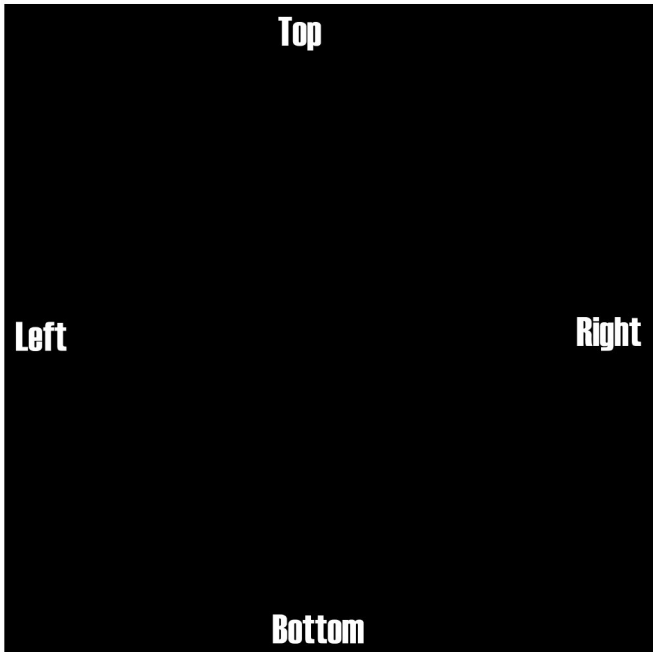
Caveat emptor.



River Water

_water currently looks a lot like the built-in “blue” water. But it’s there and ready to be modified.

Material



The name of the material is dirt2, and its texture tblr__a_a.tga. The texture doesn’t actually contain any transparency, although it really should for use as a material.

In its current incarnation, dirt2 is useful for checking position and orientation of the material texture.

There is no dirt1.

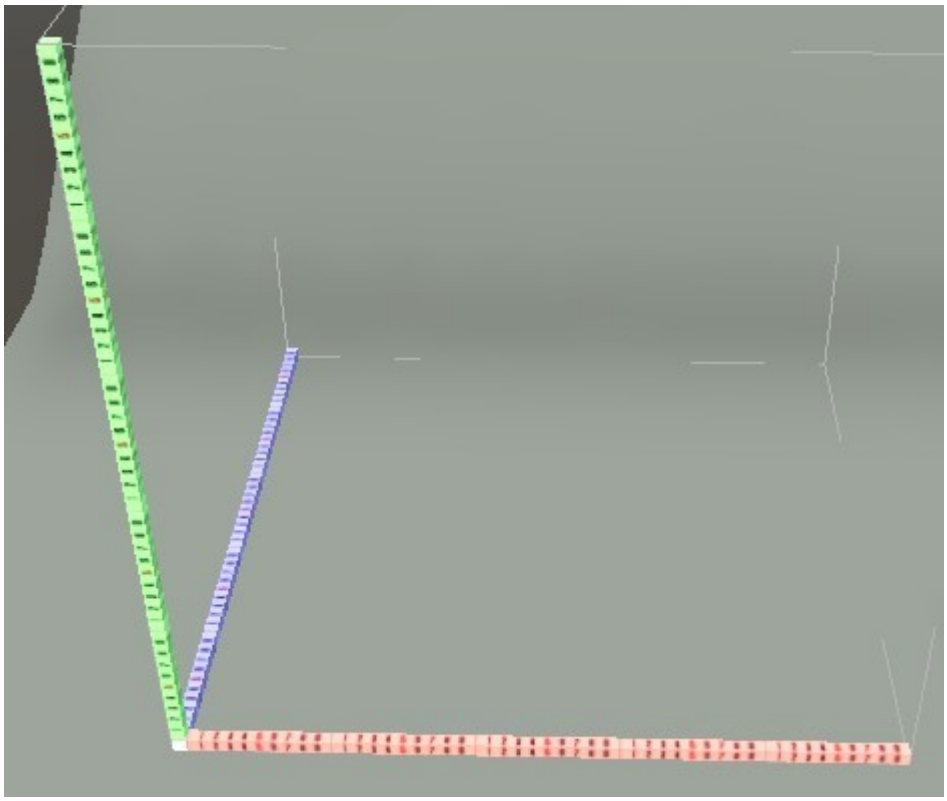
Road

_extreme_ruts has taken many testing roles, and that’s reflected in its current chaotic texture, overlays/test__d_a.tga. It also has a specular texture, overlays/test__s_d_a.tga.

_extreme_ruts currently has no ruts at all, but it sure will if you uncomment its HeightmapOffset=”common__s_d_a.tga” line. That file can be found in prebuild/common.

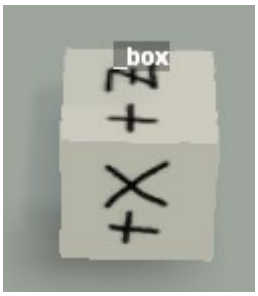
Models

Ruler



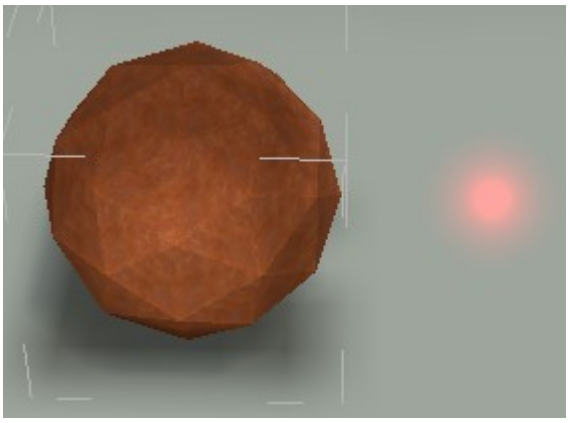
This isn’t built for experimentation. It’s built to *support* experimentation. Want to check how long a shadow is? How deep underwater you can see? How wide a truck is? A good ruler can do all of those things and more! Each axis is 5 meters long, numbered every 10 cm, with a tick mark every centimeter.

Box



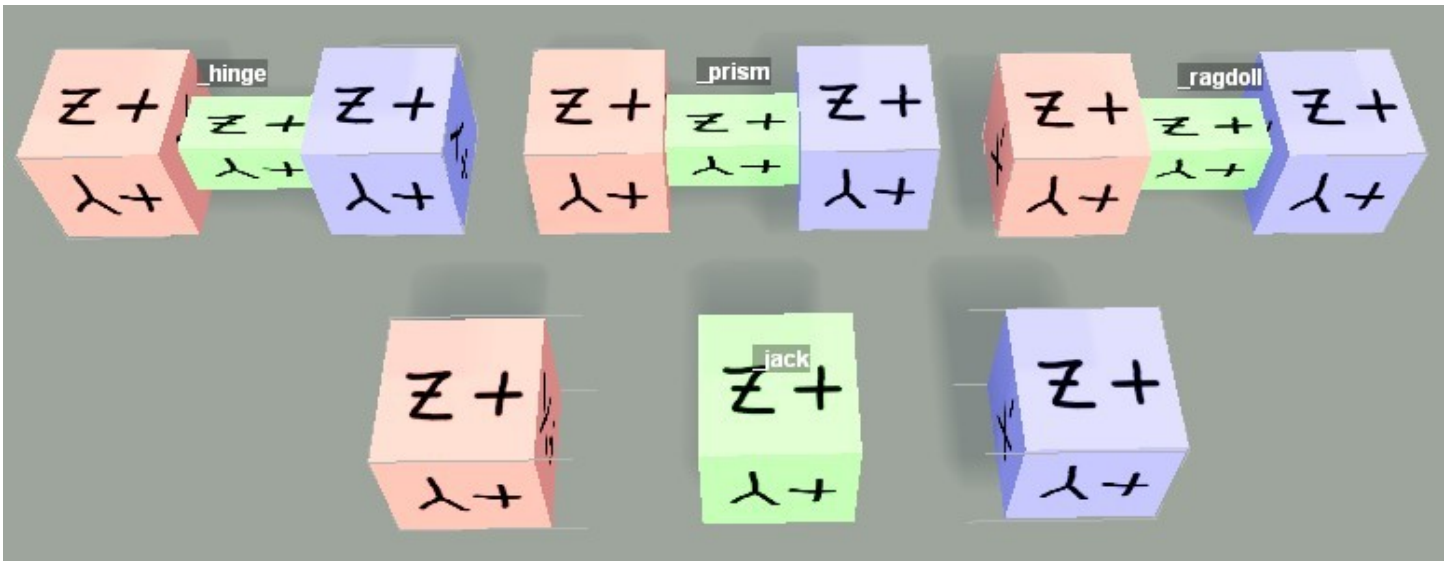
It’s a box. It’s named _box. Its model is named _box. Its texture is named _cube__d_a.png. Because consistency is hard. Its texture also secretly has a semi-transparent section that you can reveal with AlphaKill=”true”.

Ball



It’s _ball. Set up the right constraints and you can play soccer with it. Although it’ll look weird because it currently has a flare and a light attached to it. Its texture also secretly has a transparent spider that you can reveal with AlphaKill=”true”.

Constrained Cubes

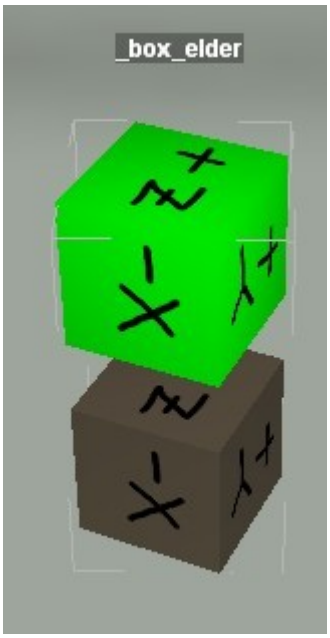


_hinge, _prism, and _ragdoll are set up to test the Hinge, Prismatic, and Ragdoll constraints, respectively. They all share the same _cubes2_deform model which includes two bodies (“red” and “blue”) and a stretchy connector (“green”). If you want to simplify, you can instead use the _cubes2 model without the green connector.

_jack has three bodies, so you can check the interaction among constraints in a model. It’s currently set up with a weird cross constraint, so you’ll need to do some editing to get it to do anything useful.

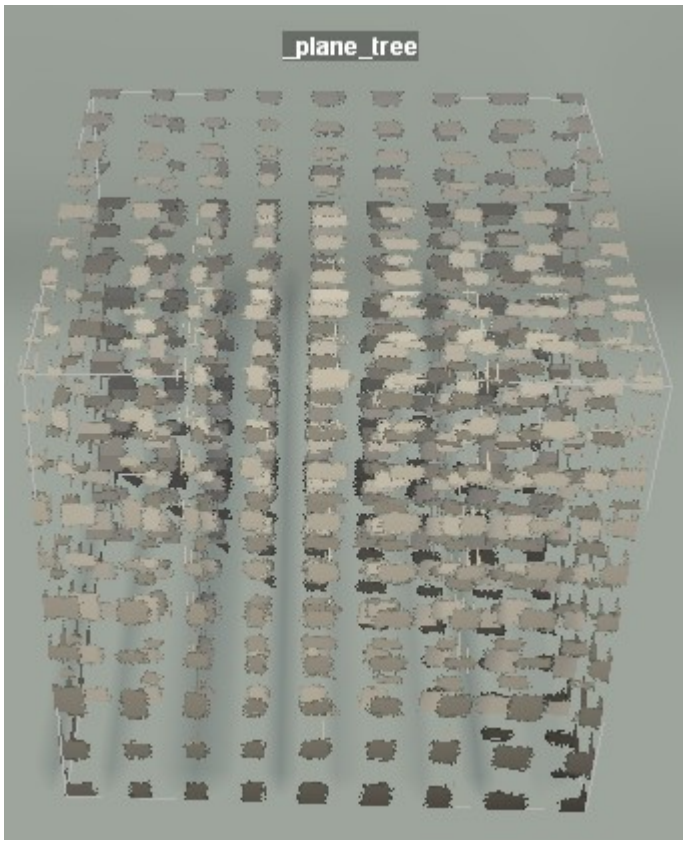
Plants

Box Elder



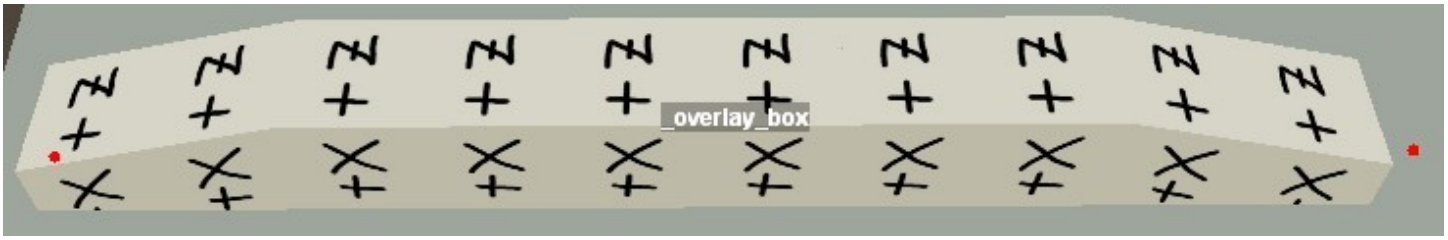
Another pair of cubes, but _box_elder has its own model since a plant can only have a single visual mesh. This plant is useful for checking root and foliage constraints.

Plane Tree



A semi-transparent grid of textures that I created to test volumetric self occlusion. I never did figure it out.

Conforming Overlay



It’s a box. It’s a conforming overlay. It’s _overlay_box.

As mentioned in the BBMMM, MudRunner can’t handle a box or convex cdt shape for a conforming overlay, so I put a kink in the cdt to make it an mmop.

If you change the OverlayType of _overlay_box, you can make it a wire overlay instead.