Customizing HaloCE Models

This chapter explorers various aspects of using 3D Halo models. Whether you want to build new scenery, a new vehicle, or a completely new map, you will need to learn to create and use 3D models.

In this chapter, you will learn

- How to add scenery to your level
- How to add lights to a model
- How to customize the animation tags for models

Creating New Scenery with the HEK

To give you an example of creating your own scenery for HaloCE, this tutorial will walk you through creating a box and then adding it to the game. It's a simple example, but using these same steps, you can add complex new models to HaloCE.

NOTE

The programs used to create HaloCE models are

- Autodesk 3D Studio Max (3ds) or Autodesk Gmax
- Tool
- Guerilla
- Sapien

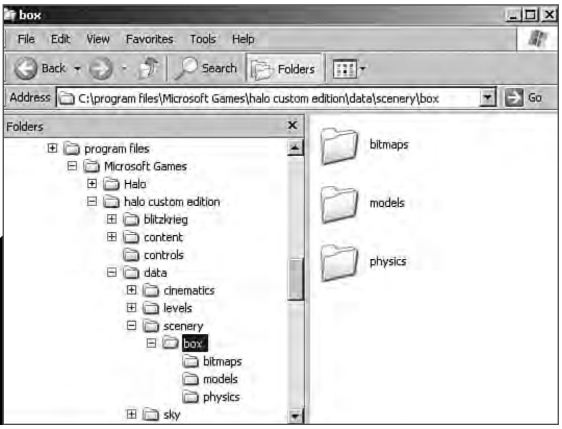
If you want to create your own textures, you also need image-processing software. Refer to Chapter 4, "HaloCE Textures," for more information about creating custom textures.

Creating Directories

Before you begin work in your 3D modeling software, you will want to create directories for your scenery files:

1. Open your HaloCE root directory: `C:\Program Files\Microsoft Games\Halo Custom Edition`. When you are in this directory, open the Data folder. When you have this open, you will see three folders called Cinematics, Levels, and *Sky*.
2. Inside the Data folder, create a new folder called Scenery.
3. Next, create a folder called Box within the new Scenery folder. Box is the main folder for your new scenery object. The path to this folder will be `C:\Program Files\Microsoft Games\Halo Custom Edition\data\scenery\box`.

4. After you have created your new Box folder, open it and create three folders called Bitmaps, Models, and Physics. These folders will contain the bulk of the data for your new model. When you are finished, the directory structure should look like that shown in Figure 3.1.



3.1

The directory structure for your box model.

Modeling in 3D Studio Max

Now that you have created the directories, you can begin to create the actual scenery files.

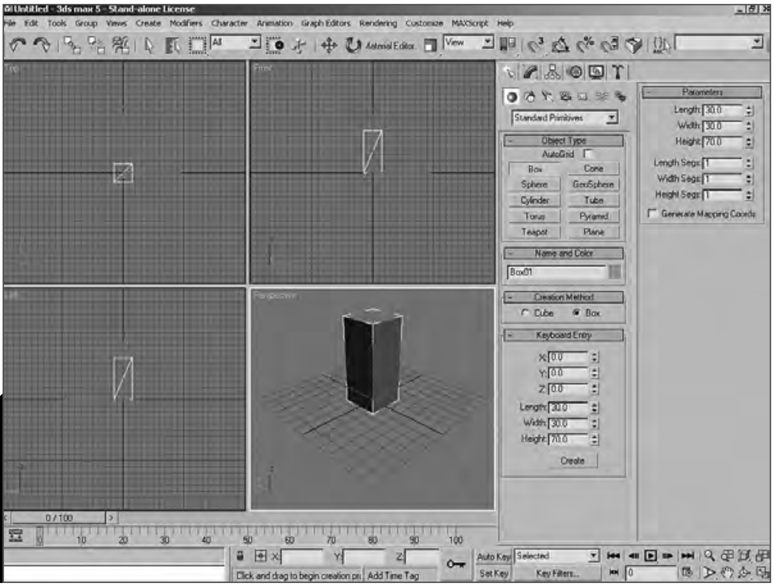
1. Open 3ds and select Create on the toolbar. Select Standard Primitives from the list, and then select Box from the list of primitives. You are going to create two boxes: one for your scenery object and one the size of the Master Chief so that you have a scale reference for your new object.

NOTE

Creating the Master Chief reference object is an optional step. This object gives a visual way of confirming the scale of the objects that you are modeling.

2. Next, create a box in 3ds either by dragging your mouse across any of the four view ports or by using the Keyboard Entry box (see Figure 3.2). In this example, you will use the text entry option to create a box with the following dimensions: width 30, length 30, and height 70 (30 × 30 × 70).

3. Click Create in the Keyboard Entry box, and your new box object will appear in all of the view ports. These are the dimensions of the Master Chief in HaloCE. You can use this box as a means of scaling your scenery.

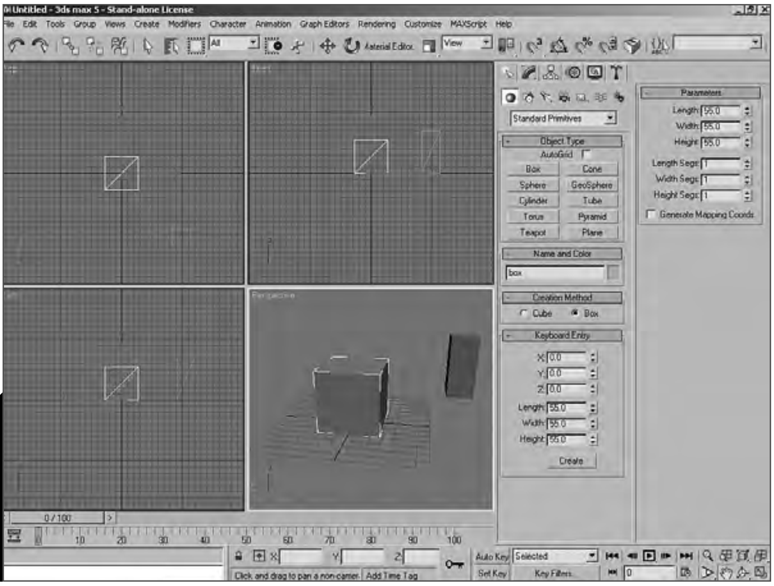

Untitled - 3ds max 5 - Stand-alone License
File Edit Tools Group Views Create Modifiers Character Animation Graph Editors Rendering Customize MAXScript Help
Standard Primitives
Object Type
Box Cone Sphere GeoSphere Cylinder Tube Torus Pyramid Teapot Plane
Name and Color
Creation Method
Keyboard Entry
Create
Parameters
0 / 100
Auto Key Selected
Set Key Key Filters...
Click and drag to begin creation pr Add Time Tag

AutoGrid: unchecked
Name: Box01
Creation Method: Cube ( ) Box (•)
Keyboard Entry: X: 0.0, Y: 0.0, Z: 0.0, Length: 30.0, Width: 30.0, Height: 70.0
Parameters: Length: 30.0, Width: 30.0, Height: 70.0, Length Segs: 1, Width Segs: 1, Height Segs: 1, Generate Mapping Coords: unchecked

3.2

Keyboard entry UI.

4. Next, you will create the actual scenery that will be added to HaloCE. To create your crate, build another box in 3ds with these dimensions: width 55, length 55, and height 55 (55 × 55 × 55). When you have the box created, select it by left-clicking it and then changing the name to `box` (see Figure 3.3). The Name field can be found at the bottom of the main toolbox.


Untitled - 3ds max 5 - Stand-alone License
File Edit Tools Group Views Create Modifiers Character Animation Graph Editors Rendering Customize MAXScript Help
Standard Primitives
Object Type
Box Cone Sphere GeoSphere Cylinder Tube Torus Pyramid Teapot Plane
Name and Color
Creation Method
Keyboard Entry
Create
Parameters
0 / 100
Auto Key Selected
Set Key Key Filters...
Click and drag to pan a non-camer Add Time Tag

AutoGrid: unchecked
Name: box
Creation Method: Cube ( ) Box (•)
Keyboard Entry: X: 0.0, Y: 0.0, Z: 0.0, Length: 55.0, Width: 55.0, Height: 55.0
Parameters: Length: 55.0, Width: 55.0, Height: 55.0, Length Segs: 1, Width Segs: 1, Height Segs: 1, Generate Mapping Coords: unchecked

3.3

The box.

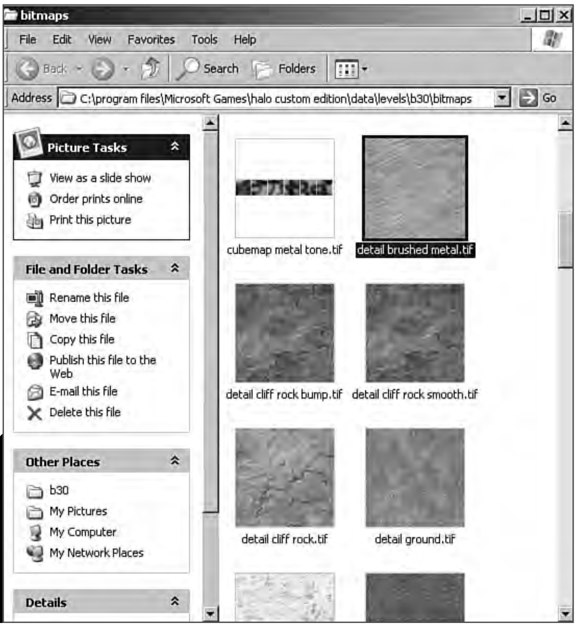
NOTE

If you would like to use the models that came with HaloCE, it is possible to extract them and repurpose them. You can even find the models posted online (for example, http://halomaps.org/index.cfm?pg=3&fid=1313).

Applying Textures Using 3D Studio Max

Your new box won't be very interesting in the game unless you give it a texture:

1. In Chapter 4, you will learn how to create your own HaloCE texture, but for this example, you will use the texture `C:\program files\microsoft games\halo custom edition\data\levels\b30\bitmaps\detail brushed metal.tif` (see Figure 3.4).



3.4

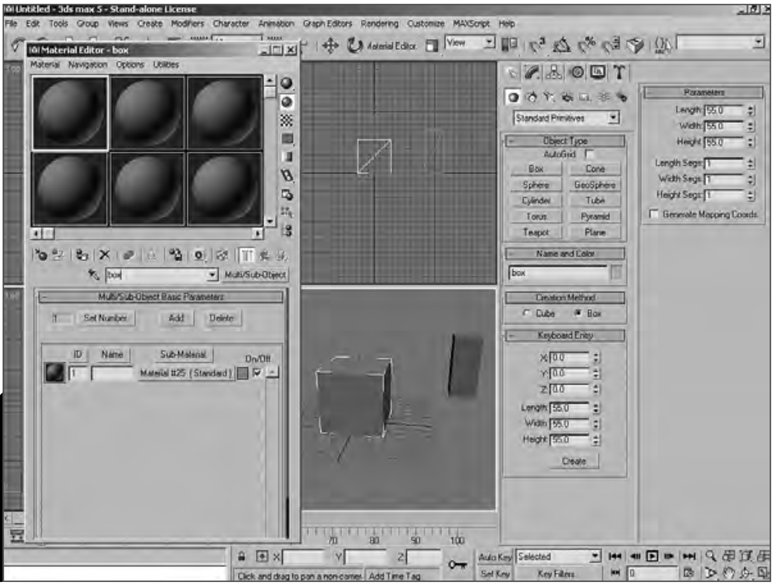
The sample texture.

2. After you have decided which image to use, copy it in your bitmaps folder: `C:\Program Files\Microsoft Games\Halo Custom Edition\data\scenery\box\bitmaps`.

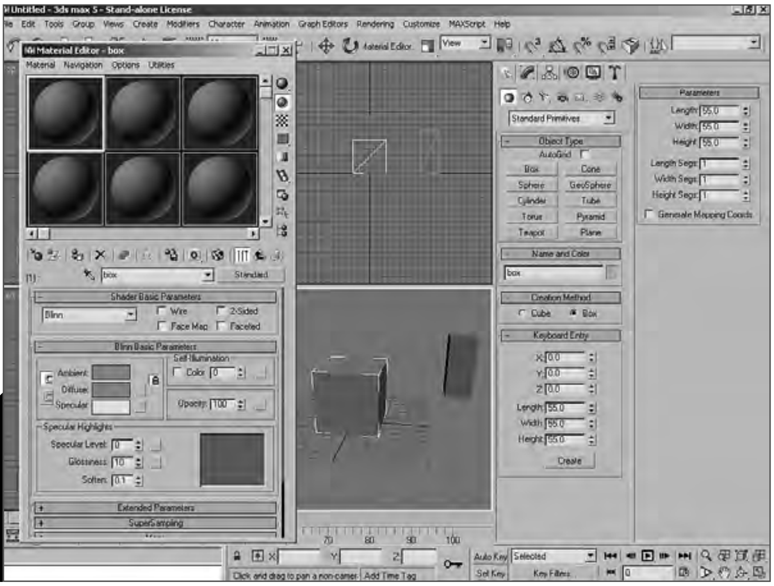
3. After you have copied your texture file into the correct directory, you will need to go back into 3ds and press the M key. This will open the Material Editor dialog window. On the right side of the dialog box, you will see a button labeled Standard. Click this button once and select Multi/Sub-Object from the list of standard materials. After you have done this, you will be prompted to discard old materials or keep old materials as submaterials. Click the Discard Old Materials option.

4. There will now be 10 materials shown in the box. However, you want only one, so click the small button labeled Set Number and change the value to 1. Now that you have only one material, you will want to rename it `box`. To rename it, type `box` into the text field next to the eye-dropper tool (see Figure 3.5). Then click the button under the Sub-Material column and change the name to `box` (see Figure 3.6).

3.5


Untitled - 3ds max 5 - Stand-alone License
File Edit Tools Group Views Create Modifiers Character Animation Graph Editors Rendering Customize MAXScript Help
Material Editor - box
Material Navigation Options Utilities
box
Multi/Sub-Object
Multi/Sub-Object Basic Parameters
Set Number Add Delete
ID Name Sub-Material On/Off
Material #25 (Standard)
Standard Primitives
Object Type
AutoGrid
Box Cone Sphere GeoSphere Cylinder Tube Torus Pyramid Teapot Plane
Name and Color
Creation Method
Cube Box
Keyboard Entry
Create
Parameters
Length Width Height
Length Segs Width Segs Height Segs
Generate Mapping Coords.
Auto Key Selected
Set Key Key Filters
Click and drag to pan a non-camera Add Time Tag

Renaming the material.

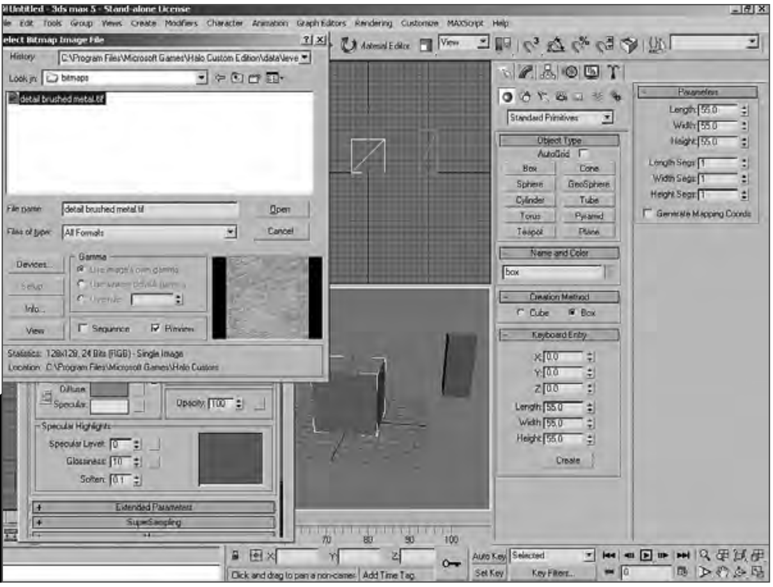


Renaming the submaterial.

5. You need to assign the correct bitmap file to the material. Because you already have the material options open, you will want to click the small box beside Diffuse. This will bring up another list of options. Double-click Bitmap—it's at the top of the list. When you choose the Bitmap option, you will be prompted to select a bitmap file. To find your bitmap file, navigate to the `scenery\box\bitmaps` folder. When you are in the folder, select the file and click OK. You have now assigned your bitmap file as the texture to be used by the material (see Figure 3.7).

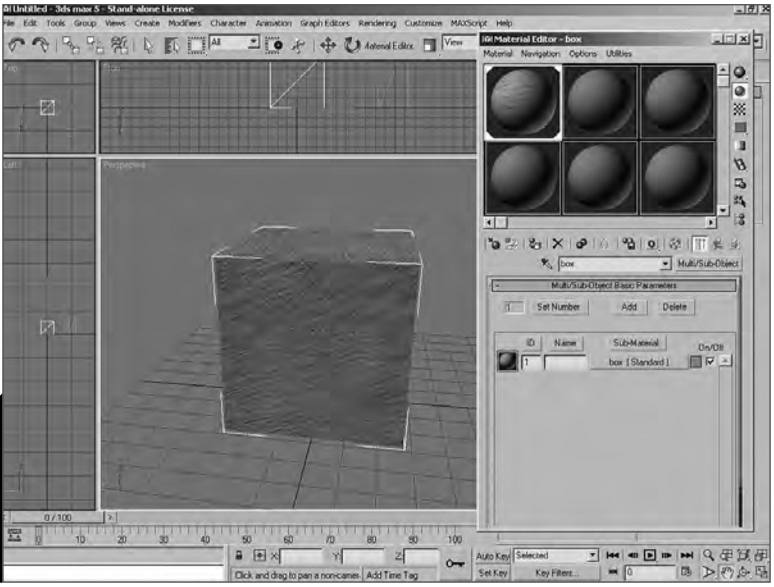
6. Next, you will be going "up to parent" (back a screen in the Material Editor) and then make the material appear in the view ports. To do this, click the up-arrow button and then the checkered box button. In the top of your Materials Editor, you will be able to see your texture wrapped around a ball. Click and drag this onto your box model. After you have applied the texture, your model will look a lot more like a crate (see Figure 3.8).

7. At this point, you will want to save the model to your `scenery\box\models` folder as a `.max` file. Do this by selecting File, Save, from the main 3ds toolbar. It's important to save your model so that you have a backup version. Save it to your Models folder as `box.max`.


Select Bitmap Image File
History: C:\Program Files\Microsoft Games\Halo Custom Edition\data\leve
Look in: bitmaps
detail brushed metal.tif
File name: detail brushed metal.tif
Files of type: All Formats
Open
Cancel
Statistics: 128x128, 24 Bits (RGB) - Single Image
Location: C:\Program Files\Microsoft Games\Halo Custom
Standard Primitives
Object Type
Box
Name and Color
box
Keyboard Entry
Length: 55.0
Width: 55.0
Height: 55.0
Create

3.7

Setting the bitmap to be used by the material.



3.8

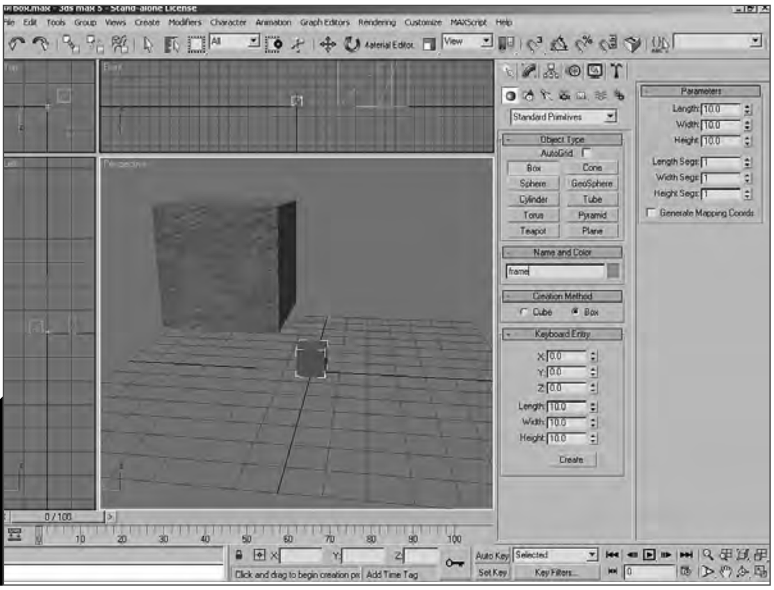
The new crate with the texture applied.

Exporting the Model and Collision Model

The next stage is to export your model out of 3ds so that you can use it with the HEK. To do this, you will use the Blitzkrieg plug-in for 3ds. Follow these steps to properly link and export your objects:

1. You need to export the model as a `.jms` file and save it in your Models folder. However, before you do this step, you will need to create a Frame object and link it to your box model. As explained in Chapter 2, "Creating Your First Halo Level," the Blitzkrieg export plug-in will use the Frame object as the basis for the export process. To make your frame model, create another box that is 10 units wide, 10 units long, and 10 units high (10 × 10 × 10) (see Figure 3.9).

3.9



Name: frame
Length: 10.0
Width: 10.0
Height: 10.0
Length Segs: 1
Width Segs: 1
Height Segs: 1
X: 0.0
Y: 0.0
Z: 0.0

Creating the Frame object.

NOTE

The size of the frame box doesn't matter. The important thing is that the Frame object isn't touching the scenery model.

2. After you have created your Frame object, name it `frame` by selecting it and typing `frame` into its Name text box. To link the Frame object to your scenery box, you will need to select the scenery, click the Link icon on the toolbar, and then click the Select by Name icon. To complete the link, select Frame from the list that appears and click Link.

3. To verify that the box has been linked to the frame, make sure that nothing is currently selected, press the H key and then check the View Subtree check box. If the name `box` is indented—under Frame in the list—you're in good shape. If not, click both objects and unlink them by clicking the Unlink button on the top left, and then relink the models.

4. The box that represented the Master Chief can now be deleted by clicking it and pressing the Delete key on your keyboard.

NOTE

Note: As long as the Master Chief reference model is not linked to the frame, it will not export when exporting to `.jms`. Therefore, deleting it is an optional step.

5. Next, you will make use of the Halo Blitzkrieg export plug-in. To export the model, click File and then Export. Export the file to your models folder, `scenery\box\models`. You should export the model as `.jms` (the Halo Model Exporter file type) and name it `box`.

NOTE

If you receive the error message `There was no geometry to export`, you will need to link Box and Frame again. This error means that the plug-in is not recognizing that any models are linked to the Frame object.

6. After you run Blitzkrieg, you will have exported your model, but you still will need to export the collision model. The collision model determines how objects react to one another in the Halo universe. To export a collision model, rename your box to `collision_model`, then go into the Material Editor by pressing M (if you've customized your toolbar, you can also click the Material Editor button). After you are in the Material Editor, you will rename the main material `collision_model`, and then go to the submaterial and rename it `collision_model`.

TIP

A collision model is not required for all models. Without a collision model, the scenery will still appear, but everything will pass through the object. There may be cases where this behavior is desirable. For example, if you were creating a bush, you might want characters to be able to run right through the object.

In the Sub-Material, you will see the Diffuse section that you used to add the bitmap file. Click the little box beside the Diffuse color well—it will now have the letter M in it. Inside the box, you will see a button labeled Bitmap; click it. Next, select None from the list of options. You have now turned the model into its own collision geometry.

CAUTION

If the collision model is too big or too small, strange behavior will result; you want it to be as close to the actual model as possible. For example, if the collision model is too big, you will find that you run into the object even when you don't appear to be touching it. A box is the simplest shape for collision geometry. Other models, such as people, are much harder to use. For example, in some games you may feel that you have scored a hit, but because the collision model doesn't fit the model's shape, the game engine registers a miss.

7. Finally, export the model to your Physics folder (not your Models folder) and name it `collision_model.jms`. You should also save your collision model into a different `.max` file than `box.max` (for example, `collision_model.max`).

Using Tool

The next step is to use the Tool utility to convert your texture into the appropriate Halo format:

1. The first command to type is `tool bitmaps scenery\box\bitmaps`. When you run this command, you will receive a response similar to the following: `bitmaps created: #512x#512, compressed with color-key transparency, 170K-bytes`. This message shows that your `.tif` image has been converted to a `.bitmap` file. This is the format that the Guerilla program uses when it creates the `.scenery` file.
2. Now that you have the `.bitmap` file, the next step is to create a shader. The shader file determines the material type (for example, dirt, wood, metal) and `.bitmap` file for a model. The various material types have different properties (that is, they react to light differently). To create a shader you need to use Guerilla. Open Guerilla, click File, New, and then select `shader_environment` from the drop-down list. When you click OK, a new `shader_environment` file will be created and you will be able to add your `.bitmap` file to the shader.
3. Now that you have a shader file, you can start to set the properties that will be associated with your crate object. First, click the check box next to Simple Parameterization (beside Flags and under Radiosity Properties). Below the radiosity properties is a section called Physics Properties; click the Material Type tab and change it to Metal (hollow). If you were making a log, you would change it to Wood. You want to set your tag to Metal because your crate is made of metal. This setting will determine how the object reacts to various effects in the Halo universe—for example, what the Halo engine should do when the object is hit or a grenade bounces off of it.

4. Next, you need to add your bitmap to the shader tag. To do this, scroll about a third of the way down the `shader_environment` list until you see a section called Diffuse Properties. Click the Browse button next to Base Map and you will be prompted to pick a bitmap. When the dialog box opens, you should automatically be in the Tags folder under the HaloCE root directory. In the collection of tag folders, you will see one called Scenery. Click this folder and you will be in a directory that contains a number of scenery folders. Find your folder—it will be called Box—and open it. In the Box folder, you will find a folder called Bitmaps; open that directory and you will find the `box texture.bitmap` file that you created with Tool. Click the `.bitmap` file and then click Open. You have now added your bitmap into the shader tag.

5. Your shader tag is now finished, so it's time to save it. In Guerilla, go to File, Save As, and then go into your `tags\scenery\box` folder. In the File view window, right-click (to bring up the context menu), and select New, Folder. Then rename the folder `shaders`, open it, and save your file here. The file will need to be named `box` and it will save as `box.shader_environment`. Make sure the shader was saved in the directory `C:\Program Files\Microsoft Games\Halo Custom Edition\tags\scenery\box\shaders`.

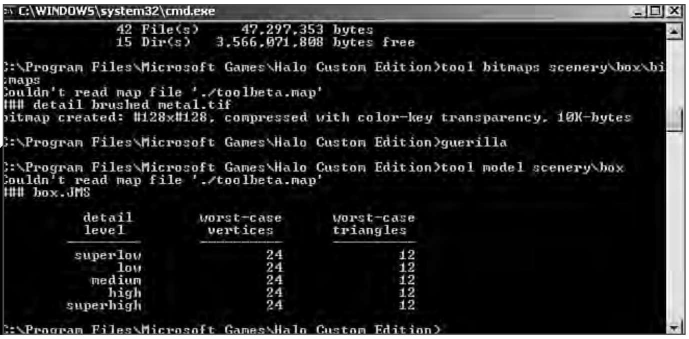
6. The next step is to use Tool to create a `.gbxmodel` file. This file is a tag that holds the scenery models properties (for example, the way it looks and the texture coordinates). The Gbxmodel file is created from the info in the `.jms` files you exported from 3ds. To accomplish this, type `tool model scenery\box`. This command will turn the `.jms` file of the box into a `.gbxmodel` that is used to create the final `.scenery` file. When you run the command, you will see a detailed response from Tool. The detail level, worst-case vertices, and worst-case triangle figures are generated when Tool tests the model with a different level of detail.

NOTE

Different levels of details (LODs) are used to increase performance in video games. When you are far away from an object the game engine will use a lower-detail object and therefore improve rendering performance. As you get closer to the object, the game will use more detailed rendering meshes. Unfortunately, this behavior can cause "popping" issues when the game switches between the model detail levels. But if the object is complex, the levels of detail may be required. One example is a forest of trees. If many foreground trees are visible, the trees farther away do not need to be as detailed because they will be obscured from view.

After you run the commands, your `.jms` model file will have been converted to a `.gbxmodel` file (see Figure 3.10).

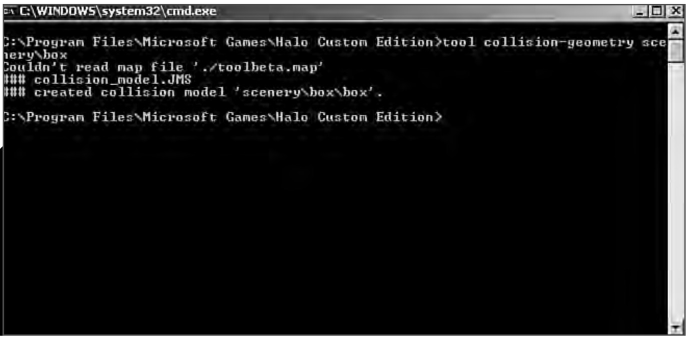
7. Next, you must create the collision model for the scenery object. This command will use the `.jms` file saved in your physics folder. To create the file, type `tool collision-geometry scenery\box` (see Figure 3.11).



3.10

```
C:\WINDOWS\system32\cmd.exe
            42 File(s)     47,297,353 bytes
            15 Dir(s)   3,566,071,808 bytes free

C:\Program Files\Microsoft Games\Halo Custom Edition>tool bitmaps scenery\box\bi
tmaps
Couldn't read map file './toolbeta.map'
### detail brushed metal.tif
bitmap created: #128x#128, compressed with color-key transparency, 10K-bytes

C:\Program Files\Microsoft Games\Halo Custom Edition>guerilla

C:\Program Files\Microsoft Games\Halo Custom Edition>tool model scenery\box
Couldn't read map file './toolbeta.map'
### box.JMS

          detail         worst-case       worst-case
          level          vertices         triangles
         --------       ----------       ----------
         superlow              24               12
              low              24               12
           medium              24               12
             high              24               12
        superhigh              24               12

C:\Program Files\Microsoft Games\Halo Custom Edition>
```

The Tool feedback after creating the .gbxmodel file.



3.11

```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\Microsoft Games\Halo Custom Edition>tool collision-geometry sce
nery\box
Couldn't read map file './toolbeta.map'
### collision_model.JMS
### created collision model 'scenery\box\box'.

C:\Program Files\Microsoft Games\Halo Custom Edition>
```

Tool feedback after creating the collision model.

This response tells you that Tool successfully created the collision model without any errors. When creating more complicated scenery models, the model must not generate errors. If the model contains errors, the collision geometry will not compile properly.

CAUTION

If the geometry of your model is too complex, it may contain holes or degenerate triangles that break the sealed world rules that are required to build a BSP. Generating a rough hull of the object will make the collision data simpler and easier to manage. This is why the collision data and model data is split into separate parts. For example, the collision model for the Warthog is a much simpler version of the Warthog itself. The collision model has a reduced polygon count and it conforms to the sealed object rules. The model itself does not need to conform to these rules.

If you run into problems with errors, there is a workaround that you can try. Create a box that is about the same shape as your scenery, rename it `collision_model`, delete the actual scenery model in 3ds, apply the `collision_model` material to the new box, and then export. The simple box will work as a collision model and it will be easier to work with.

Creating the Object Tag in Guerilla

At this point, you must combine your `.model_collision_geometry` and your `.gxbmodel` tags so that you can create the final `.scenery` file:

1. Open up the Guerilla application. Click File, New, and select Scenery from the drop-down list. The only part of this file that you will be changing is the top section (above Export to Functions). Find the Bounding Radius section and change the value from 0 to 30 world units. The bounding radius determines how far away you can see the object. By setting the value, you're determining how far away the object will be visible. In other words, it won't disappear if you walk a step back from it, but it also won't be using precious memory resources if you're far enough away.

TIP 01011

The bounding radius is a spherical range around an object. When the sphere is in a player's viewing frustum, the object will be drawn.

A size of 30 translates to 3,000 units in 3ds max. This is an extremely high number and it will cause the object to be rendered when it isn't near any players—this can cause a performance reduction.

A good way to gauge this number is to create a sphere at the center point of your scenery object and change the radius until it encompasses your entire scenery object. Then take the radius of the sphere and divide it by 100. The resulting number is the value that you should use in the bounding radius field.

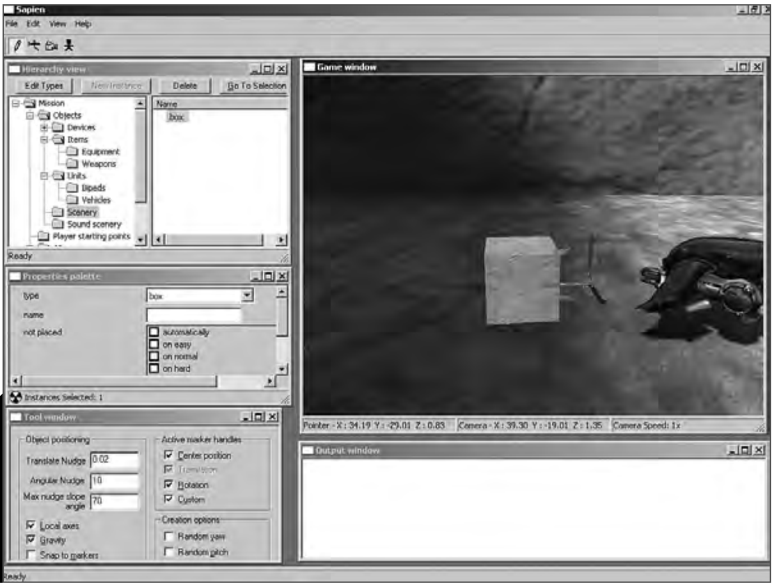
2. At this point, you will see a section labeled Model and a Browse button. Click the Browse button to open your Tags folders. Next, open the Scenery folder and then the Box folder. In this directory, you will see two folders and two files. The folders are bitmaps and shaders; the files are your `.gbxmodel` file and your `.model_collision_geometry` file.

3. Assign the model to the `.scenery` tag. To do this, double-click the box.gbxmodel. In Guerilla, below Model, you will see Animation Graph and Collision Model. Click the Browse button beside Collision Model. When you do this, you will be brought into your tags folder again. Navigate to your Box scenery folder and double-click `box.model_collision_geometry`. This will assign the collision model to your scenery tag.

4. You have now finished your scenery tag. The last step is to save the tag in your `tags\scenery\box` folder. Save the file as `box.scenery`.

You can now use Sapien to add your crate scenery to a map (see Figures 3.12 and 3.13).

3.12



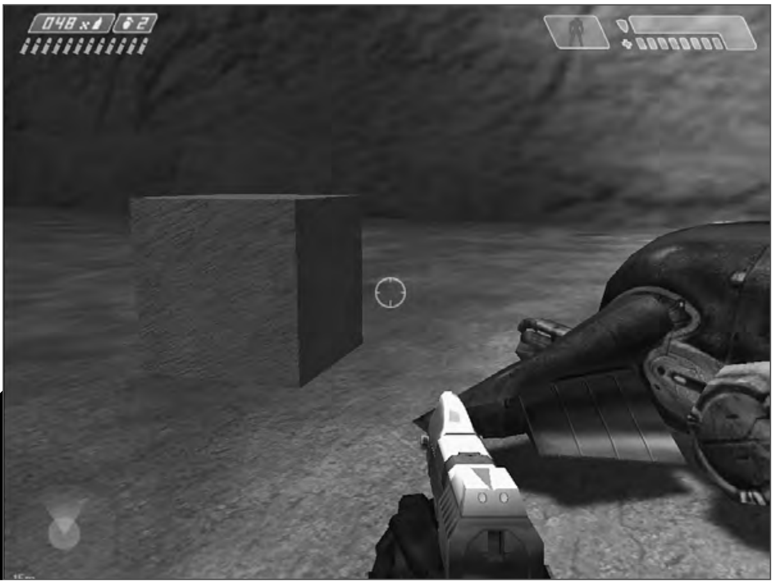
type: box
name:
not placed: automatically, on easy, on normal, on hard
Instances Selected: 1

Translate Nudge: 0.02
Angular Nudge: 10
Max nudge slope angle: 70
[✓] Local axes
[✓] Gravity
[ ] Snap to markers
[✓] Center position
[✓] Translation
[✓] Rotation
[✓] Custom
[ ] Random yaw
[ ] Random pitch

Pointer - X : 34.19 Y : -29.01 Z : 0.83 Camera - X : 39.30 Y : -19.01 Z : 1.35 Camera Speed: 1x

Placing the metal box in a level using Sapien.

3.13

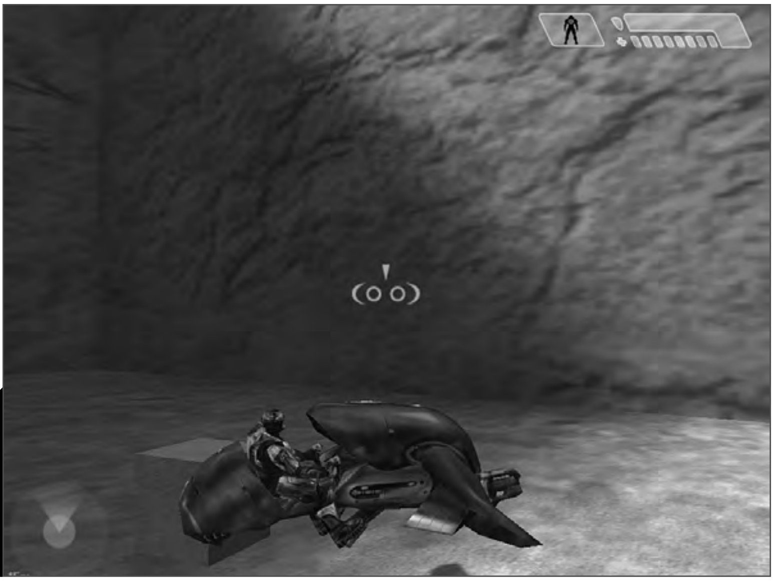


The metal box as it appears in a level.

Adding Lights to a Model

Creating a new box may give you some insight into creating new models for HaloCE, but it doesn't tell you much about the relationship between Halo models and the meta tag system. In this tutorial, you will get a chance to play with the tags at a different level.

This tutorial will walk you through adding new lights to the Covenant Ghost vehicle. Rather than creating a new model, you will alter the properties of the existing Ghost (see Figure 3.14).

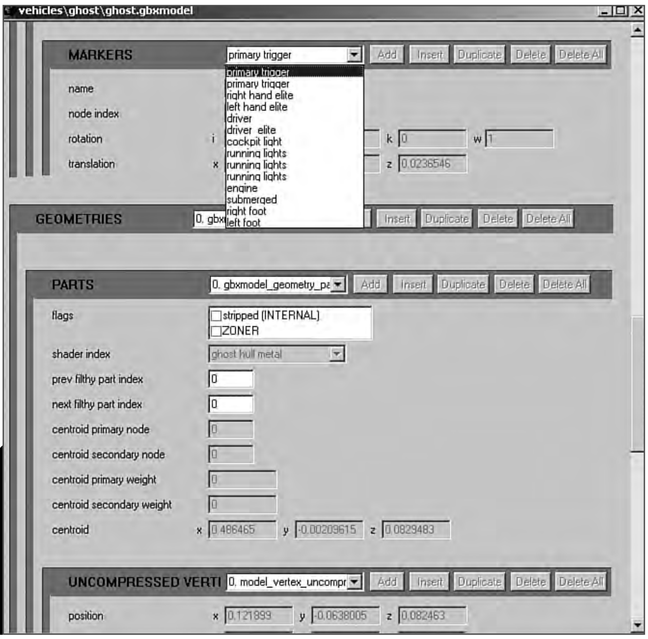


3.14

The Ghost without running lights.

1. The first step is to open Guerilla, which you now know is the HaloCE tag editor. From within Guerilla, open the tag for the object that you want to alter. In this tutorial, you will be adding lights to the Ghost, so you need to open `ghost_mp.vehicle`. The Ghost tag is located at: `C:\program files\microsoft games\halo custom edition\tags\vehicles\ghost`. When you find the tag, click the Open button next to Model. This will open the object's `.gbxmodel` tag.

2. In the `.gbxmodel` tag, scroll down until you see Markers (you may not have to scroll at all). Ignore this category and scroll down until you see Markers again. You should see a pull-down menu with names such as Primary Trigger and Engine. These are special spots on the model that were set when the model was in 3ds. They are called markers and they are used as guides to tell Halo where to put certain things. For example, markers determine the location of particle effects, sounds, and lights (see Figure 3.15).


vehicles\ghost\ghost.gbxmodel

MARKERS: primary trigger
rotation: k 0, w 1
translation: z 0.0236546

GEOMETRIES: 0. gbx

PARTS: 0. gbxmodel_geometry_pa
flags: ☐ stripped (INTERNAL) ☐ ZONER
shader index: ghost hull metal
prev filthy part index: 0
next filthy part index: 0
centroid primary node: 0
centroid secondary node: 0
centroid primary weight: 0
centroid secondary weight: 0
centroid: x 0.486465, y -0.00209615, z 0.0829483

UNCOMPRESSED VERTI: 0. model_vertex_uncompr
position: x 0.121899, y -0.0638005, z 0.082463

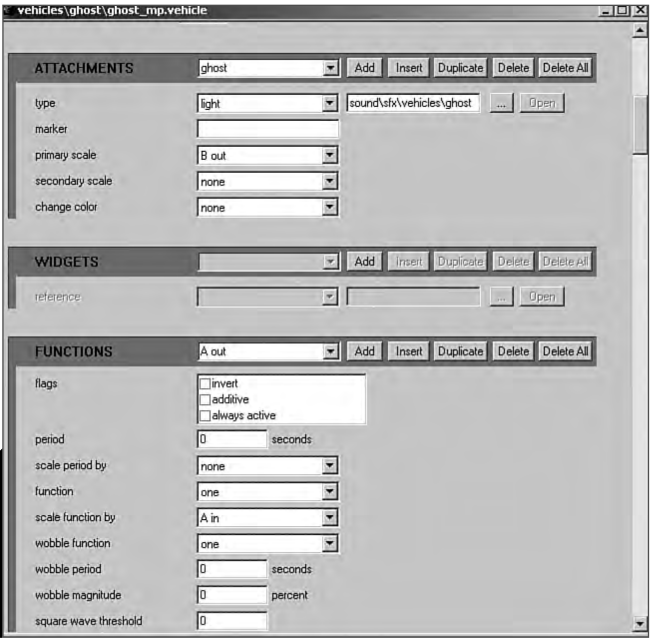
The Markers list.

3. Look for a marker whose name indicates the correct position on the object you are editing. For example, on the Ghost, there are a few markers named Running Lights. These markers are placed on the Ghost's lower portion where, in single-player games, little blue lights are displayed. For some reason (possibly performance), they took out these lights in multiplayer games. However, the markers remain.

CAUTION

It's crucial that you remember the exact name of the marker. You may want to write it down.

4. Go back to the `.vehicle` tag and look for the ATTACHMENTS section. Select Ghost from the drop-down menu and then click the Duplicate button. Then from the drop-down box next to Type select Light (see Figure 3.16).



ATTACHMENTS: ghost
type: light — sound\sfx\vehicles\ghost
marker:
primary scale: B out
secondary scale: none
change color: none

WIDGETS
reference

FUNCTIONS: A out
flags: ☐ invert ☐ additive ☐ always active
period: 0 seconds
scale period by: none
function: one
scale function by: A in
wobble function: one
wobble period: 0 seconds
wobble magnitude: 0 percent
square wave threshold: 0

3.16

Creating a new attachment.

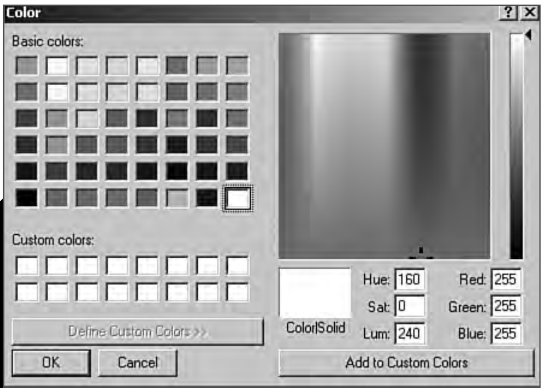
5. Now you should consider how you want your light to appear. Think of something in the game that has light and that has a similar look to the effect that you want. In this example, you'll be using the Banshee's cockpit light. This is a good choice because it appears as a small, shiny light that will fit well on a Ghost. Click the Browse button next to Type and find `tags\vehicles\banshee\banshee cockpit.light`. Do not double-click it yet; instead, right-click it and then click Copy. Next, right-click somewhere else in the window and click Paste. You will notice that a file called Copy of banshee cockpit.light was created. Rename this file to `ghost running light.light`. Double-click this new file.

6. The Browse box should now disappear and you will be back to the `.vehicle` tag.

> NOTE
>
> The text in the drop-down list next to ATTACHMENTS will change from Ghost to Ghost Running Light if you refresh the drop-down list by clicking it.

Click the Open button next to the Browse button. Your newly created `.light` tag should open. Under Color, set the Color Upper Bound to your desired color of the light. By default, it will be a purple color, because this is a color that is on the Banshee (see Figure 3.17).

3.17

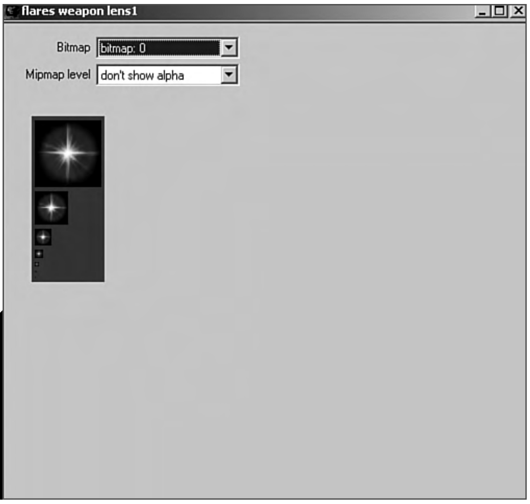


Hue: 160
Sat: 0
Lum: 240
Red: 255
Green: 255
Blue: 255

Editing the light tag.

7. Now create a lens flare effect for your light. The Lens Flare is the bitmap that you see when the light is activated. To assign a bitmap, scroll down to Lens Flare. Click Open, and then click Open (next to Bitmaps) in the dialog box that opens. When it is open, click Show Bitmaps; you will actually see what the light will look like (see Figure 3.18). If this is not what you want your light to look like, close the window and click the Browse button next to Bitmap. You should see a list of several lens flares. You can open each of them until you find one that looks just right to you. For this example, simply use the first one.

8. Now it's time to choose whether you want the light to be dynamic. A *dynamic light* is a light that casts light onto other objects, such as Warthog headlights, the MC's flashlight, and the light you see after a grenade explodes. *Nondynamic lights* are the lights that do not affect anything else. These lights include the Warthog taillights and the landing beacons. Scroll up to the top of your "ghost running light.light" tag. If you want dynamic lights, check the Dynamic flag and the No Specular flag. Underneath that, you will see Shape. In the Radius field, set the amount of world units (how wide of a radius the light will affect). For a small subtle light, 1 or 2 world units will be fine. For something like an explosion that lights a wide radius, you would set this number to 10 or more. If you do not want your lights to be dynamic, leave these fields untouched.



Bitmap: bitmap: 0
Mipmap level: don't show alpha

3.18

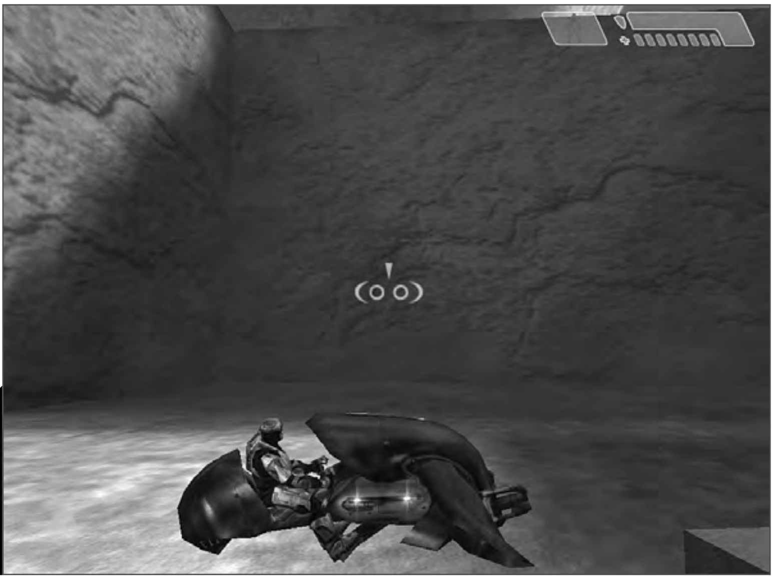
Lens flare bitmap.

9. Next, you must associate the correct marker with an event. First, go back to the `.vehicle` tag. Under Attachments, in the Marker field, put the exact name of the marker that you chose before (running lights). In the Primary Scale field, scroll down to A out, or whatever function you are using in that tag to represent the event in which you want the light to be displayed. In the Ghost's case, A out is the event of activating the vehicle. The light will show only when the vehicle is in use.
10. You are now finished making changes. Go to the File menu and click Save All.
11. Compile your map.

NOTE

Make sure you have at least one Ghost in your map.

12. Enjoy your shiny new ride (see Figure 3.19)!


3.19

The Ghost with new running lights.

`Model_Animations` Tag Reference

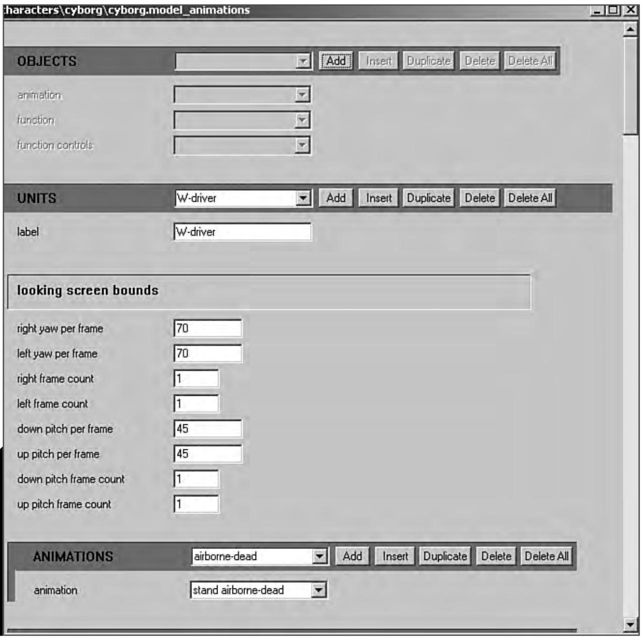
`Model_Animations` is a useful tag; it determines the animated behavior of HaloCE models. The objects that use this tag include bipeds (Elites and the Master Chief), vehicles (the Warthog, Ghost, and Banshee), and devices (the various machines that the Master Chief can interact with). Using this tag, you can create moving doors, elevators, switches, and light fixtures. Another good example is the subtle swaying of pine trees.

TIP

Unfortunately, multiplayer games don't allow you to use switches with elevators, but they function quite well for single-player games. Although you cannot use a flag or device group to automatically turn a `light_fixure` on and off, you can use export functions and a dynamic light to do so—if the scenario does not allow the `light_fixture` to emit radiosity. This can also be accomplished with scripting, if you prefer not to use device groups.

Automatic doors work perfectly in multiplayer games and they can add a cool element to maps. As Halo players know, you can't just shoot through a closed door—you have to walk near it so that it will open.

This reference section will help you learn more about the `Model_Animations` tag. Open any Model_Animations tag in Guerilla and you will see that the tag contains an enormous amount of data (see Figure 3.20). Don't worry if this section seems obscure—it's not meant for pleasure reading. Rather, it is meant as a reference for you to use when experimenting with animations. You may find someday that you need this information for one of your mods.



animation	
function	
function controls	
UNITS	W-driver
label	W-driver

looking screen bounds

right yaw per frame	70
left yaw per frame	70
right frame count	1
left frame count	1
down pitch per frame	45
up pitch per frame	45
down pitch frame count	1
up pitch frame count	1
ANIMATIONS	airborne-dead
animation	stand airborne-dead

3.20

The cyborg.model_animations tag open in Guerilla.

Objects

This section is for linking animations to functions. For example, the Assault Rifle firing animation is linked to a specific user input function. First you select the animation, then you select the function to link it to, and finally you choose how the selected function acts on the animation.

There are three fields in this section:

- Animation—The animation to be played.
- Function—The function that will control when the animation will be played.
- Function Controls—The function can control starting/stopping the animation or the scale of the frame length.

The animation selected must be an overlay animation (saved as `.jmo`) and the function must be set in the parent tag (for example, biped, vehicle, or device). The *function controls* decide how the animation is played. For example, the animation will have its frame decided by the function. When the function is zero, the frame will be one; when the function is one, the frame will be the last frame in the animation. When set on scale, the animation will loop and its play speed will be based on the overlay. When the function is zero, the animation will not play; when the function is one, the animation will play at full speed.

NOTE

The animation will starting playing at frame 1 and then play until the end. If it is an overlay animation, it will never play frame zero.

Units

In the animation tag, there is a large section devoted to units. The most important thing to remember about the unit section is that it maps game events to the appropriate animation. For example, when the Master Chief enters the passenger seat of a vehicle, that event is mapped to the W-Passenger (Warthog Passenger) unit of the `cyborg.model_animations`, and the W-Passenger animation itself is tied to the Enter event.

To link animations with independent models, you need a reference point that is common between the two. Halo does this through the use of seats. The master chief does not simply “pop” into the Warthog vehicle, he climbs in. This is done by animating the Master Chief relative to the Warthog and then storing this animation in the animation tag.

Halo needs to know when to play the animations. In the vehicle tag, there are imaginary seats. These tell Halo that when a Master Chief biped enters the vehicle a certain animation is to be played. This idea is then translated throughout the vehicle\biped system.

Under the UNITS heading, you can select various seats from the drop-down list and you can change the Label—the name of the seat.

The following are the commonly used seats:

- Stand—All the animations associated with a standing unit (including movement animations).
- Crouch—All the animations associated with a unit while it’s crouching.
- Alert (AI only)—The animations for an AI when it’s in a noncombat state. The movement animations are used for when the AI is patrolling.
- Asleep (AI only)—Used when the AI has been told to be asleep (for example, Grunts).
- Flee (AI only)—Used when the AI is scared and running away.
- Flaming (AI only)—Used when the AI has a dangerous object (for example, plasma grenade) attached to it.

- Vehicle seats—This name can be whatever you want it to be, just as long as it matches the name you used in the vehicle tag. Vehicle animations are more restricted than the other seats; while in a vehicle, a unit cannot perform move, dive, melee, and other such animations.

Looking Screen Bound

The up, down, left, and right maximums and minimums for looking while in the seat.

Animations

The nonweapon-specific animations assigned to the seat:

- Airborne-dead—A base animation that loops when the unit is airborne and also dead. This animation is usually the same for all seats and is usually the biped flailing his arms and legs.
- Landing-dead—A base animation played when the airborne (and dead) biped makes contact with the ground. This animation is usually a roll.
- Acc-front-back—An overlay animation used exclusively for vehicle seats or for vehicles themselves. This animation is used for simulating the momentum of sudden starts, stops, and bumps. The animation has three specific frames:
    - 0—default
    - 1—front
    - 2—middle
    - 3—back
- Acc-left-right—This is the same as acc-front-back, but it is used for left and right movement. This animation is also played to make the biped appear to lean when a vehicle is cornering.
- Acc-up-down—This is the same as acc-front-back, but it is used for up and down movement.
- Push—This is unused.
- Twist—This is unused.
- Enter—This is a base animation used when a unit enters a vehicle. Without this animation, a unit cannot enter the vehicle.
- Exit—This is a base animation used in vehicle seats that allows the unit to exit the vehicle. Without this animation, a unit is stuck in the vehicle (unless the vehicle is flipped and the occupants are thrown out).
- Look—The overlay animation associated with the looking bounds (see looking, aiming, and steering animations).

- Talk—A one frame overlay animation used for talking.
  - 0—mouth closed
  - 1—mouth open
- Emotions—This is unused.
- Unused—This is unused (duh)
- User0—This is unused.
- User1—This is unused.
- User2—This is unused.
- User3—This is unused.
- User4—This is unused.
- User5—This is unused.
- User6—This is unused.
- User7—This is unused.
- User8—This is unused.
- Flying front—This is unused in the game. It may be an overlay that you can use in a kind of on\off way.
- Flying back—Same as flying front.
- Flying left—Same as flying front.
- Flying right—Same as flying front.
- Opening—When the vehicle's driver exits the vehicle, this animation plays and then freezes on the last frame.
- Closing—When the vehicle's driver enters the vehicle, this animation plays and then freezes on the last frame.

IK Points

This is used for vehicle seats; it attaches a marker on the biped to a marker on the vehicle. This can cause limbs to bend in awkward ways. The first field is the name of the marker on the biped; the second is the name of the marker to which it is attached on the vehicle:

- Weapons—Weapon categories for animations within the seat. If the weapon type does not exist in the seat, you will not be able to switch to that seat (unless it is a vehicle seat).

- Name—The name of the class of weapons. It is for personal reference only; it does not actually affect anything in the game.
- Grip marker—The marker on the weapon that is attached to the hand marker. If it is left blank, the weapon's origin is used (that is, point 0x 0y 0z).
- Hand marker—The name of the marker to which the weapon is attached, usually "right hand."
- Aiming screen bounds—The up, down, left, and right maximums and minimums for aiming while using the weapon class.
- Animations—The animations specific to this weapon class within the current seat.
- Idle—A base animation that is played when no other animations are being played.
- Gesture—This is unused.
- Turn-left—A turning animation that is used when you aim far enough to the left.
- Turn-right—A turning animation that is used when you aim far enough to the right.
- Dive-front—(AI only) A base animation that is used when there is a hazardous object behind the biped.
- Dive-back—(AI only) Same as dive front, except it moves the biped backward.
- Dive-left—(AI only) Same as dive front, except it moves the biped to the left.
- Dive-right—(AI only) Same as dive front, except it moves the biped to the right.
- Move-front—Played when the biped is moving forward.
- Move-back—Played when the biped is moving backward.
- Move-left—Played when the biped is strafing to the left.
- Move-right—Played when the biped is strafing to the right.
- Slide-front—Sliding animations are played when a vehicle is reacting to gravity. For example, if the vehicle is facing uphill, but is sliding downhill, its "sliding-back" animation will be played. It is most likely a base animation.

NOTE

The difference between a *base animation* and an *overlay* is that an animation defines the base position for the bones/sections of a model, whereas an overlay is a modification to these base animations.

For example, when the Halo 2 Master Chief is standing—and holding his weapon—his standing animation is the base animation. Then a weapon overlay is added to the base standing animation. This overlay brings the Chief's arms to the correct position so that he appears to be holding his weapon. When the Chief fires his weapon, an overlay is also added that shows a slight kickback effect from the weapon.

- Slide-back—Same as slide-front.
- Slide-left—Same as slide-front.
- Slide-right—Same as slide-front.
- Airborne—A base animation that is looped when the biped is in the air.
- Land-soft—A base animation played when the biped performs a soft landing—as defined in the biped tag.
- Land-hard—A base animation played when a biped performs a hard landing—as defined in the biped tag.
- Unused—This is unused.
- Throw grenade—A base animation played when tossing a grenade.
- Disarm—This is unused.
- Drop—This is unused.
- Ready—A base animation that is played when switching to the weapon class.
- Put away—This is unused. When a player is taking a weapon out, the ready animation is played. Put away is assumed to be the opposite—the animation played when putting a weapon away. But it is unused because the Master Chief automatically puts away the old weapon by just making it disappear. He instantly has the new weapon; then the animation just blends from the old weapon's idle stance to the new weapon's idle stance.
- Aim-still—The overlay animation associated with the aiming bounds used (see looking, aiming, and steering animations).
- Aim-move—Overrides aim-still when the biped is moving (see looking, aiming, and steering animations).
- Surprise-front—(AI only) A base animation that is played when an AI is startled by your presence.

- Surprise-back—(AI only) A base animation that is played when an AI is startled by your presence. This usually involves the AI turning around to see you. If the AI does turn in this animation, it must be saved as a turning animation, not a base animation.
- Berserk—(AI only) A base animation played when an AI goes berserk, as defined by the actor variant tag. The AI does not require this animation in order to go berserk.
- Evade left—(AI only) A quick base animation of a hop or roll to the left. This is used by AI to make it harder for the player to hit them.
- Evade right—(AI only) A quick base animation of a hop or roll to the right. This is used by an AI to make it harder for the player to hit them.
- Signal-move—(AI only) A base animation that is used by an AI to signal a maneuver.
- Signal-attack—(AI only) A base animation that is used by an AI to point out a target for an aggressive maneuver.
- Warn—(AI only) A base animation that is used to warn friendly AI of an enemy's presence.
- Stunned front—A base animation that is used instead of move front when you are stunned and trying to move forward.
- Stunned back—Same as stunned front, but used for moving backward.
- Stunned left—Same as stunned front, but used for moving to the left.
- Stunned right—Same as stunned front, but used for moving to the right.
- Melee—(AI only) A base animation of a melee attack. Melee for a player is weapon specific; it is not a base animation.
- Celebrate—(AI only) Used by an AI (in a vehicle seat) when he is excited by something (for example, taking air in a hog).
- Panic—(AI only) Used by an AI (in a vehicle seat) when he is scared by something (for example, taking too much air in a hog).
- Melee-airborne—(AI only) Used by an AI when performing a melee while airborne.
- Flaming—This is obsolete; it was replaced by the flaming seat.
- Resurrect-front—(AI only) A base animation that is used by an AI when reviving after feigning death and falling forward (for example, the Flood combat form).
- Resurrect-back—(AI only) A base animation that is used by an AI when reviving after feigning death and falling backward (for example, the Flood combat form).

- Melee-continuous—(AI only) A base animation that is used by an AI when it is performing a constant melee after attaching to target (for example, the Flood infection form).
- Feeding—(AI only) A base animation that is used while feeding on a victim that has been killed via continuous melee attacks.
- Leap-start—(AI only) A base animation that is used when starting a leaping melee attack.
- Leap-airborne—(AI only) A base animation that is used when airborne in the middle of a leaping melee attack.
- Leap-melee—(AI only) A base animation that is used when performing the melee attack while airborne in a leaping melee attack.
- Zapping—This is unused.
- IK points—This is different than the other IK points. This one is used for weapons and is usually used to help keep the left hand where it needs to be.

Weapons Types

This is where you place the code for the weapon (usually two letters). This can be anything, but it must match the same code put into the weapon tag. Examples: ar = assault rifle, hp = human pistol, pp = plasma pistol, pr = plasma rifle.

Animations

The animations for a specific weapon:

- Reload (1, 2)—An overlay or replacement animation for reloading ammo in your primary (1) or secondary clip (2).
- Chamber (1, 2)—An overlay or replacement animation for chambering ammo. This is used when the Shotgun is reloading.
- Fire (1, 2)—An overlay animation used for firing primary and secondary triggers.
- Charged (1, 2)—An overlay animation that is played when a trigger is charged.
- Melee—A replacement animation that is played when a player performs a melee attack.
- Overheat—An overlay (or possibly even a replacement) animation that is played when a weapon is overheated. An effect is usually employed instead of this animation.

Weapons

To animate weapons, you must add at least one block to this section, even if it isn't a weapon. If you don't, Sapien or HaloCE will crash:

- Idle—Base animation that is used when the weapon is idle.
- Ready—Base animation that is played when a weapon is drawn.
- Put-away—Base animation played when a weapon is put away.
- Reload (1, 2)—A base animation that is played when reloading.
- Chamber (1, 2)—A base animation that is played when chambering (such as Shotgun).
- Charged (1,2)—A base animation that is played when charging.
- Fire (1,2)—A base animation that is played when firing.

Vehicles

The animations used exclusively by vehicles:

- Steering bounds—Much like Looking bound. It is used by a vehicle only when the driver is looking in the right direction.

Animations

The animations in this section are overlays. Overlay animations are a special type of animation that are "morphed" on top of an existing character animation. For example, while the Master Chief is sitting in a braking Warthog, the acc-front-back overlay animation determines how the MC will appear as he shifts forward and backward in the vehicle. While the overlay is playing, one of the base animations will be playing at the same time (passenger firing a weapon, sitting idle, reloading a weapon):

- Steering—The animation attached to steering bounds.
- Roll—Based on vehicle roll.
- Throttle—Based on vehicle throttle.
- Velocity—Based on vehicle velocity.
- Braking—Based on vehicle braking.
- Ground-speed—An overlay animation that plays the last frame (on a loop). The animation speed is determined by the vehicle's ground speed and the wheel circumference field in the vehicle tag. This is how the Warthog's tires spin.

NOTE

Frame 0 is the default frame for all overlay animations because overlay animations are relative to frame 0. If frame 0 has a rotation of 15 degrees and frame 1 has a 30-degree rotation, then frame 2 will have a 40-degree rotation and frame 3 will have a 65-degree rotation. If this animation was applied to ground speed, it would move the node +15 degrees in frame 1, then +25 degrees in frame 2, and +50 degrees in frame 3. Then it would loop back to frame 1 where it would have +15 degree rotation.

- Occupied—Based on whether the vehicle is occupied.
- Unoccupied—The opposite of occupied.

The four animations—roll, throttle, velocity, and braking—are scaled based on the vehicle data in the game. They all contain the following frames:

- Frame 0—Default.
- Frame 1—Zero of the value (for example, zero velocity, zero roll, or zero throttle).
- Frame 2—Half of the value (for example, half of max velocity or half of max roll).
- Frame 3—All of the value (for example, full velocity or full braking).

Suspension Animations

You can use suspension animations to simulate bouncing or other jarring motion in a vehicle. For example, the shock absorbers in the Warthog use suspension animations to simulate realistic movement of the tires. The amount of movement is tied to the vertical impact that the vehicle experiences. If your Warthog performs a short jump, the shocks will react differently than they would if the vehicle were to jump off a cliff.

You can have as many suspension animations as you want, but typically there is one for each contact point with the ground. In the case of the Warthog, there are four suspension animations—one for each tire.

These animations work perpendicular to the ground:

- Mass point index—The index of the mass point that will be affected by the suspension (see your physics file and count down—the first point in the list is zero).
- Animation—The two frame overlay animations are used for this suspension. The name of this animation will become the name in the list of suspension animations. The frames for the overlay are as follows:
    - Frame 0—Default

- Frame 1—Fully extended
- Frame 2—Fully contracted full extension\compression values:

These get added on to the ground depth value in the physics file. They determine the depth of fully extended and full contracted suspension.

Warthog Example

Ground depth = 0.15 (world units)

Full extension ground depth = -0.31445

Full compression ground depth = -0.13452

Ground depth + full extension ground depth = ground actual ground depth when suspension will be fully extended

0.15 + (-0.31445) = -0.16445

Suspension will be fully extended when mass point is 0.16445 world units off the ground

Ground depth + full compression ground depth = ground actual ground depth when suspension will be fully compressed

0.15 + (-0.13452) = 0.01548

Suspension will be fully compressed when mass point is 0.01548 world units in the ground

Devices

The two animations are used by devices. Both are overlays and both run on key frames. Each key frame poses the model in a specific way. The key frames are set at the following:

0-default

1-zero value

(last frame) full value

The animation speed is determined by the device tag:

- Position—Played using the device's position
- Power—Played using the device's power

Unit Damage

This is the place where all death and hurt (ping) animations are placed; it is the most complicated field in animations. If you're working with this field, it's a good idea to use the cyborg's animations as an example.

First-Person Animations

This is where animations involving first-person weapons are placed.

Sound References

An area for you to add sounds so that you can make them play with your animations:

- Sound—The Browse button associated with this field allows the user to select a sound.
- Limp body node radius—The radius around each node that makes collision with the environment. If left at zero, it will use the default 0.04 world units when playing limp body physics after death.
- Flags—Here you can choose to compress all animations and force idle animation compression. You probably don't need to worry about these settings.

NOTE

Flags and limp body node radius are not part of the Sound References block. These flags are child fields of the actual model animation data, not one of the model animation block's data.

Nodes

This is a list set up for a biped to run limp body physics. It is set up upon import and generally should not be messed with. If a value in here is changed incorrectly, the game will crash every time this biped tries to perform limp body physics. If you mess it up and cannot get it back, you can avoid the game crash by unchecking Uses Limp Body Physics in the biped tag (the flag is near the bottom).

Animations

The heart and soul of the `model_animations` tag are the animations themselves:

- Name—The name.
- Type—One of three types: overlay, base, or replacement.
- Frame count—The number of frames in the animation.
- Frame size—The total size of the frames in bytes.
- Frame info type (base type only)—The way the frames are stored.
- None—No frame information.
- dx, dy—The frames have information on X and Y coordinates and thus allow movement in these two directions.

- dx, dy, dyaw—The frames have information on X, Y, and yaw (rotation), thus allowing movement and turning.
- dx, dy, dz, dyaw—The frames allow movement in all directions and the yaw rotation.

Some Notes About Movement

When a biped animation comes to an end, it will reset the biped back to the origin point. However, if it allows movement in the x and y directions, the x and y coordinates will remain the same—even though a new animation is starting. Using this, you can create movement, which is necessary for the move front, left, right, back, dive, evade, and the vehicle enter and exit animations.

Rotation on yaw is important for turning animations so that when the turning animation is done, the character actually inherits the direction of the last frame of the animation (or the current frame if the animation is interrupted early). If turning animations do not have this frame type, the biped will keep looping the turning animation.

The z direction is important if you want a jump in your animation. Without z, the biped will play the animation and it will look normal, but if the animation is interrupted while the biped is in the air (or if it finishes in the air), the biped will instantly snap back to the ground to begin the next animation.

- Node list checksum—A number assigned based on the model's nodes, it doesn't actually do anything; it is just used to make sure a model has the correct animations assigned to it.
- Node count—The number of nodes in the animation.
- Loop frame index—When this animation is told to loop, it will start on this frame.
- Weight—The higher the weight, the greater chance this animation will be chosen (it works only if there are many permutations of the animation).
- Key frame index—The frame in which the bullet is fired or melee damage is dealt.
- Second key frame index—The second frame in which the effect is dealt (this works only for AI melee animations).
- Next animation—The next permutation for the animation.
- Flags
    - Compressed data—Data is compressed.
    - World relative—Unknown.
    - 25Hz (PAL)—Used in the old European Xbox version.
- Sound—Which sound (from sound references) to play with the animation.
- Sound frame index—Which frame the sound begins playing.

- Left foot frame index—Which frame to create the left foot material effect on (movement only).
- Right foot frame index—Which frame to create the right foot material effect on (movement only).

CAUTION

Modifying the bottom four fields will not give you any advantage; it only has the chance of messing up your animations and crashing the game.

- Frame info—The size of the frame info. This is greater than zero only if you have a frame info type. If you change this value, Halo will read too much (or not enough) information, and possibly the wrong information. Therefore, it will mess up your animation and possibly even crash the game.
- Offset to compressed data—The amount of bytes it has to skip to get to the data for this animation. This is used only when you have compressed animation. If you change this, it will point to the wrong data and mess up your animations or crash the game.
- Default data—The size of the information used for the first frame of your animation. If you change this value, Halo will read too much or not enough information, and possibly the wrong information. Therefore it will mess up your animation and possibly crash the game.
- Frame data—The size of the data for all the frames past the first. This may be smaller because Halo needs only the offsets from the first frame and records it only for the nodes that actually move. If you change this value, Halo will read too much or not enough information, and possibly the wrong information. Consequently, it will mess up your animation and possibly crash the game.

NOTE

If you have questions about Guerilla, feel free to contact kornman00 (`kornman00@gmail.com`); he has offered his assistance with any reader inquiries. Please use the subject line “Black Art of Halo Mods question” so that he can identify your message.

Summary

Creating your own models will be necessary if your goal is to create an entirely new map. This chapter touches on a few different aspects of modding existing models or creating your own custom elements. Whether it’s something as simple as a crate or something as complex as a vehicle, the information in this chapter should help you with your modeling needs.