

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Halo Mapping Tutorials Index

Notes:

- *Anyone can Add Tutorials or Edit Current Work by hitting EDIT on any page.*
 - *All changes are logged by IP & This Site is regularly monitored for abuses.*
 - *If you add a page to the Wiki, make sure you add a Link to it from another page!!*
 - *Files & images can be uploaded using ATTACH up to a maximum of 4MB. View the Wiki Homepage for a list of commands*
 - *More Tutorials are to come, but spread the word & encourage people to add to as well as download from this Wiki!!*
-

Halo Editing Kit

- [How do I start mapping for Halo CE?](#)
- [HEK Tutorial](#) Original HTML Document, Modified for Wiki
- [Halo Reticle Tutorial](#) by JackBauer

HEK - Tool.exe

- [Fixing Open Edges & Other Tool Geometry Errors](#) by EViLDaVE
- [Making Custom Sound Tags | Adding Custom Sounds to Your Gun](#) by SuperSunny and UPDATED by CptLag
- [Creating your own Sky Gearbox Models \(WIP\)](#) by EViLDaVE
- [Adding strings with Tool](#) by Steelix B
- [Custom Animations Tutorial - Basic](#) by Ki L La
- [Custom Animations Tutorial - 3p](#) by Ki L La
- [How to create sprite / sequence bitmap tags](#) by Steelix B

HEK - Sapien.exe

- [Wind In Sapien](#) by MasterChief_123
- [Adding AI To MP Maps \(Video\)](#) by Aztec & Some Random Guy
- [Net Game Flags Placement \(inc. Wind for CTF Flags\)](#) by Xzzy
- [“Super Simple Teleporter Tutorial”](#) by ^ d®ûÑk ©øp

HEK - Guerilla.exe

- [Four Seated Warthog](#) by Ang88
- [Making the Custom Sky from the SoT Map](#) by EViLDaVE
- [How to add Water](#) by SuperSunny & Baka?
- [Tutorial on making new skys](#) by mattp001
- [How to add Ghost Plasma Stun Effect](#) by Colin?
- [How to create chain spawning projectiles](#) by Deimos
- [Adding extra seats in vehicles](#) by Super Sunny
- [Basic Destructible Scenery\ Vehicles tutorial](#) by The Missing Link 117?

HEK - Custom Weapons

- [Custom Weapon Tutorial Part1 - Before the HEK](#) by Choel & DepthCharge
- [Custom Weapon Tutorial Part2 - .weapon_is_fun](#) by Choel & DepthCharge
- [Custom Weapon Tutorial Subsection - Light Volumes - DRUGZ ARE FUN](#) by Choel & DepthCharge
- [Simple Weapon Tutorial Step-by-step](#) - By ZiON
- [Custom Weapon Strings](#) by Steelix B

HEK - Custom Vehicles

- [Very Basic tutorial on custom vehicles](#) by Axeman

3D Modeling

- [The Huge-Butt List of 3D Modeling Resources](#)

3DS Max

- [Create an a30 Halo-Style Waterfall](#)
- [Creating Basic Terrain](#) by SomeRhino
- [Creating More Advanced Terrain using Displace Modifier](#) by EViLDaVE
- [3D Modelling Tutorial: the Base from Bloodgulch](#) by EViLDaVE
- [3D Modelling Tutorial: Cel-Shading](#) by SuperSunny and CMDR Reservoir
- [Exporting models using the GBX model extractor](#) by Gav89

Gmax

Skinning & Texturing

- [Skinning Tutorial](#) by [mattp001?](#)
 - [Tutorial on Making Textures Tileable for use as Detail Maps](#) by [EViLDaVE](#)
 - [PSP8 version of Tilable textures](#)
 - [Advanced UVW Mapping using UVWUnwrap \(WIP\)](#) by [EViLDaVE](#)
 - [Creating New Textures for Objects That Already Have a Texture](#) by [HitManBob?](#)
-

Uncategorized

Scripting

- [Adding Custom Scripts](#) by [kornman00?](#)
- [Getting Scripting Documentation](#) by [Pkmnrulz240](#)
- [Playing music with Scripts \(SP\)](#) by [Pkmnrulz240](#)

Other

- [Use -devmode effectively](#) by [Timo12](#)

Please create links to common [Halo](#) Editing utilities or resources, such as [Tool](#) by putting two curly brackets around them & *must start with a capital letter* like this - `{{Tool}}`. This will make it easy for someone else to expand on that utility or Halo Editing resource.

[Halo](#) /

Halo Reticle Tut

- view
- edit
- attach
- print
- history

This tut requires you to know how to compile bitmaps for the HEK, and what an alpha channel is.

Make your reticle with a black background(64 x 64) or any other size that the HEK will take, then make your reticle in white over the black background. Then select the entire bitmap and copy it to the clipboard, then create a new alpha channel and paste that bitmap on the clipboard in. save your bitmap, and compile it, you should then open your compiled .bitmap and change the format type to “32-bit color” and recompile the bitmap. you will have your reticle in high quality. setup your reticle in your weapon_hud_interface, with sequence index 0, and select the flag “Not a sprite”.

Tutorial by [Jack Bauer?](#)

Fixing Open Edge Tool Errors

- view
- edit
- attach
- print
- history

REQUIREMENTS: 3D Max, Tool, A Recent Tool-Compiled JMS file that Has a S*load of Errors**

ACKNOWLEDGEMENTS:

I originally learned this technique from a video tutorial by Kirk4 on [Halomods](#), so great praise & thanks to him for spreading the knowledge 🙏

DESCRIPTION:

Errors that are returned by Tool.exe upon level compilation such as open edges, coplanar surfaces etc... can be located and fixed easily in 3D Max using the following technique. This technique also applies to many of the tool errors listed in the HEK tutorial, including:

1. WARNING found nearly coplanar surfaces (red and green)
2. WARNING: found duplicate triangle building connected geometry.
3. WARNING found # degenerate triangles.
4. WARNING unearthed edge (magenta boxed lines)
5. WARNING: a surface clipped to no leaves (see cyan in error geometry)
6. ERROR found z buffered triangles (red).
7. ERROR edge # is open (red).
8. ERROR couldn't update edge # (red).
9. ERROR couldn't build bsp because of overlapping surfaces (orange)
10. ERROR: portal does not define two closed spaces. (see yellow in error geometry)

STEP BY STEP INSTRUCTIONS:

1.) Geometry created for use in the HEK must adhere to the sealed world rules, described in the HEK Tutorial Document, under [General Overview](#). A sealed world is one where all edges and faces continuous lead on from another, and there are no holes, gaps, or misaligned edges. In addition, a number of other factors will cause Tool errors, such as intersecting faces, two vertices occupying the same space and incorrectly applied materials/[UVW Maps?](#).

2.) Whenever Tool finds an error, (you may get hundreds), it produces an error file. This can be imported into

3DS as follows:

- In 3DS, Select file→import
- Select the file type as VRML
- Navigate to your Halo CE installation directory
- Select the file called mymap.wrl (where mymap is the name of your map)
- Hit OK to import the file

3.) Now, deselect everything in your 3DS environment and press 'h'. Up pops the objects lister. A bunch of objects have been added that highlight the exact areas where there are problems. These are called **VR L01?** or something like that. Select them all.

4.) This geometry is now selected and will appear to be white. In your viewports, it is now possible to locate the errored regions of your geometry by jumping to the location where the a single peice of selected error geomoeetry is. Edges are quite discrete, so a thin white line may well be concealed by your objects. Coplanar surfaces result in red & green triangles being added to your 3DS document. Unearthed edges result in magenta boxed lines being added to your 3DS file and so on (see HEK tutorial, Level Compilation →Troubleshooting, for a full description of error geometries and their manifestations).

5.) Pan & Zoom your Top, Right & Front viewports, and rotate/zoom your perspective viewport until a particular piece of error geometry is in all four view. If it is impossible to see the error geometry (sometimes it is obscured by your object), right click the move gizmo with the error objects selected, and move them 1 point in any direction. This is just to help clarity, don't worry about it if you can see the error geometry.

Example for fixing an Open Edge

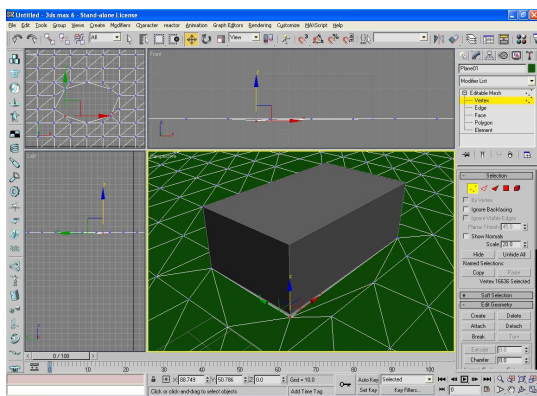
7.) In the perspective view with the errored geometry in view, you have to try to locate that open edge. An open edge as described by the HEK tutorial as follows:... (Find it under level compilation in HEK tut)

There is an open edge in the level.

Typically, there is a "hole" in the level that is preventing the level from satisfying the Sealed World Rules and

therefore the level cannot compile successfully.

Basically, if you have two faces that dont quite meet, a missing face, or you have accidentally moved a vertex so a hole is created, tool will flag an open edge error. The image below shows a rather blatant open edge between a box (simple base :P) and ground (green plane). The vertex that is selected is misaligned to the box, creating a visable hole. This hole must be closed in order for Tool to compile the map successfully.

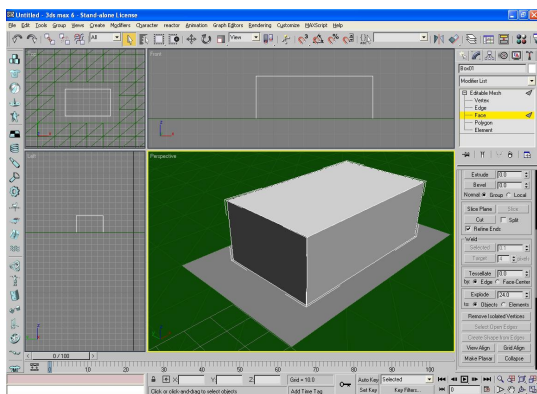


(Click on thumbnail to enlarge)

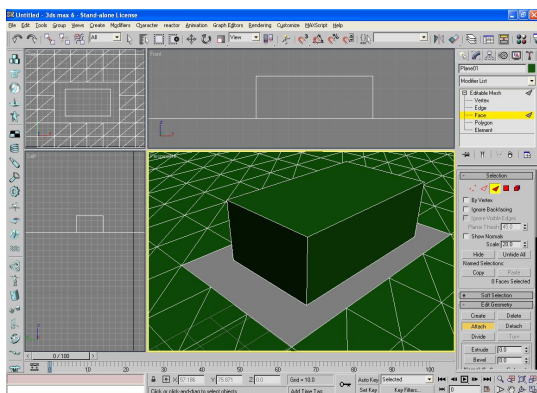
8.) In this simple case, the open edge can be fixed by one of two methods. Firstly, the vertex could simply be moved to align with the appropriate vertex on the base. This alignment must be precise, so rather than moving it by eye, select the vertex on the base you want it to align with and enter the exact x, y & z locations for the vertex on the ground.

Note: Exact transformation can be performed by selecting a vertex (or object) with the move gizmo held down, and then entering the x, y, z values in the boxes at the bottom of the screen in 3DS. If the boxes are grayed out, move the vertex a little, hit undo and then they should be activated.

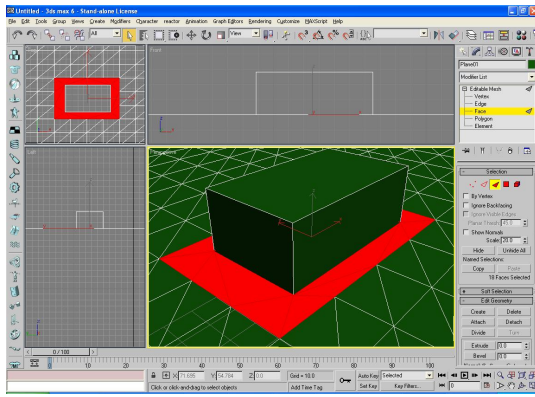
The second, and preferred method for fixing this and more complicated open edge problems is to delete the faces surrounding the error, and reapply them. For instance: select the plane and go into edit mesh mode, & face selection. With ignore backfacing off, in the top or perspective view, delete the faces surrounding the open edge like so:



Now, with the plane selected, and still in edit mesh mode, hit attach in the edit mesh dialog and click on the base. This will attach the base to the ground - it will become part of the ground object, inheriting the ground material:



In the edit mesh dialog, go to face selection mode, and hit create. Now create new faces around the base, so that it is properly attached to the ground.



Note: When creating faces, you click on the vertices you want to become part of the face cyclicly, ie: hit the three points of the triangle either anticlockwise, or clockwise. The direction you turn affects the normals - the direction the new face will point. Clockwise will create a face pointing away from you. Counterclockwise will create one facing towards you. If you mess up the creation, and need to invert a face's direction, hit flip-normals under surface properties with that face selected.

Finally, detach the base from the ground by selecting all the faces that make up the base, and hitting 'detach' in the edit mesh dialog. Remember that to select multiple faces, press control while selecting, or dragging a lasso. To deselect faces, press ALT.

9.) Fix all your open edges one at a time using the method presented above, by jumping to the location, inspecting the geometry and if necessary, deleting and re-creating faces. Remember to re-apply materials, smoothing groups and **UV Wmaps?** after editing faces. Sometimes a hole in the level is simply created because you accidentally moved a point. If this is the case, just move it back, precisely though, using the x, y, z boxes as explained above.

10.) Once you have fixed them all, hit 'h' again, select all the error geometry, delete it, save your 3DS file and re-export. If more errors turn up in tool you can import them as before and look for the solution in the HEK tut.

Application to Fixing Any Other Tool Errors

Each Tool error has its own little personality that must be learned & respected, and for each, the solution is different. In my experience, I have come across open edges, z-buffered triangles, coplanar surfaces, degenerate triangles and I've even had the good old 'couldn't update edge'.

While most of these can be fixed by deleting faces and re-creating them, this is both tedious and sometimes unnecessary. Look to the HEK Tutorial (Level Compilation→Troubleshooting) for a description of what your particular Tool error means. This explanation can sometimes be cryptic, for instance coplanar surfaces is explained by the HEK Tut as where 'two surfaces exist that are coplanar'. Thanks... In fact, all this means, is that Tool got confused because two surfaces that are at an angle appear to be in the same plane, and moving a

vertex that these two triangles share will disrupt the 'plane' that Tool detects, thus avoiding the error.

Dead Giveaways to Look Out For

- *Double Vertices:*

In the region of the error geometry, check there are not two or more vertices occupying the same space. You can check this by selecting a vertex by dragging, going into the edit mesh dialog, and seeing how many vertices are selected. If there is more than 1, in the edit mesh dialog, go to weld. Enter a threshold for welding (this depends on distance between the vertices, if its zero, leave the threshold at 0.1), & hit weld. Multiple pairs of vertices can be welded at once by selecting a whole area of vertices, and setting the threshold low (eg: 0.1) so that only vertices within that threshold distance are welded together.

- *Double Faces:*

Similar to double vertices. Check there are not two faces occupying the same area using the counter in the edit mesh dialog. if there are, delete one, or both, and repair any resulting holes.

- *Intersecting/Overlapping Faces:*

If faces intersect or overlap because they arn't connected to where they should be, they must be deleted and any holes repaired.

STL Check

Finally, selecting all the faces on your problem object and applying an STL Check Modifier (in Modifier list) allows you to check for a number of errors at once, such as open edges, double faces, spikes (overlapping faces) etc. Select the errors you want to check for and hit 'check'. The errors will be highlighted as before. This is often a good check to perform before attempting to compile, although, be aware that open edge errors that STL Check picks up will be fine by Tool, as STL check takes into account the fact that you have different objects. If in doubt, compile it, and import the WRL file.

Have fun fixing your geometry!!!

Keep in mind also that there is an open edge selection tool in 3ds Max. Go into edge selection mode, and below the weld toolbar on the side, there is a button that says select open edges. Click that, then move all the open edges upward so you can see where they are. Of course, you will get the obvious ones where you have attached things to your skybox, but sometimes you can find other errors, things you forgot about etc. Undo the move, then fix the errors.

- view
- edit
- attach
- print
- history

General Overview

The following section gives a general overview for the creation of Multiplayer (MP) Levels for Halo for the PC. This section includes rules, guidelines, components and other useful information related to the creation and editing of Multiplayer environments for Halo and its various game types.

Introduction

Halo content or game assets (levels, vehicles, weapons, characters, etc...) are created using arbitrary polygons. Arbitrary polygons and arbitrary polygon modeling has several advantages including the ability to create complex shapes and surfaces that can be either concave or convex and flexible texture mapping including UVW mapping. The arbitrary polygons are created or modeled using a 3D modeling program such as 3D Studio Max (3ds Max) or equivalent. Halo was created using 3ds Max 3.x on the Xbox and the additional content for the PC version was created using 3ds Max 5.1.

Content Creation Path

The following diagram shows the work flow or content creation path when creating a Halo game environment. The subsequent sections contained under **Multiplayer Level Design** give information and explain each of the major steps (including tutorials) involved in the process of bringing a Halo level from the 3D modeling tool into the Halo game engine.

Level Content Creation Path Flowchart

http://www.drunkdragons.com/wiki/images/Level_content_path_flowchart_final.gif([approve sites](#))

Technical Rules

The following are some basic technical rules that must be followed during the creation and construction of a game level in Halo. These rules must be followed to insure that the level will successfully export and compile in order to run in the game.

Reference Frame

The level **MUST** have a reference frame with valid geometry linked to the reference frame. If the level does not

have a reference frame the Blitzkrieg exporter will not export the .jms file and will give an error dialog “There was no geometry to export”. There must also be valid geometry (See **Sealed World Rules** below) attached or linked to the reference frame or the Blitzkrieg exporter will not export the .jms file and will give an error dialog “There was no geometry to export”.

For more information on the reference frame for the world mesh please see the section **Creation of a Reference Frame** under [Level Creation Part 1](#).

Sealed World Rules

The level **MUST** be sealed. While other models (those that are placed in the level using .scenery, .vehicle, etc... tags) used in Halo for scenery, characters, weapons and vehicles do not have to be sealed and can have open edges, this does not hold true for the world geometry. The polygons used to construct the world must form a contiguous mesh that forms a sealed volume in order for the level BSP (Binary State Partition) that is used for player collision and physics to be created. The sealed volume and no open edges rule applies to objects that are included in the world mesh (for instance a floating box).

As implied in the first paragraph, the sealed world and no open edges rules do not apply to models that are external to the level and are placed or populated in Sapien using their associated tags (.scenery, .vehicle, .weapon, etc...).

For more information on creating the world mesh, creating a sealed world, and open edges please see the various tutorials and information found under the [Level Creation Part 1](#) and [Level Creation Part 2](#) sections.

Multiplayer Level Limitations

Multiplayer levels are very straight forward and simple in terms of what can be placed in the level and what can operate in the level. The multiplayer game and levels do not have the ability to run the complicated scripting systems that are available in the single player game and levels. These systems are simply not included in the current network architecture that Halo uses for multiplayer.

This limits the dynamic objects that can be placed in a multiplayer level since the majority of these kinds of objects (such as doors, light bridges, buttons, etc...) rely on the Halo scripting system to operate. The same holds true for trigger volumes, therefore trigger volumes cannot be used in multiplayer levels.

Certain objects that are simple client side effects can be placed in multiplayer levels. Such objects or effects include particle emitters (such as the water fall mist in Timberland), sounds, and other simple objects. Even the Covenant portable force shields used in multiplayer maps like Danger Canyon and Gephyrophobia are not networked. No information gets sent that tells the game server that “the shield has been disrupted”.

In the case of the Covenant energy shield, the reason it does get turned off on both the client and the server is because the objects that affect the shield (weapons fire and explosions) **DO** have their information sent over the network and when they get updated on the client it affects the objects in the surrounding area. As a result, these objects can get out of synchronization with the server. This is why the use or heavy use of such objects should be avoided.

Continue to the Next Section - [Level Creation Part 1](#)

[Return to Main Page](#)

- view
- edit
- attach
- print
- history

Level Creation - Part 1

The following section provides more detailed information on creating Multiplayer levels and environments and includes tutorials and examples. Topics covered in the tutorials include the creation of a level directory, creation of a reference frame, sealed world rules and explanation, the creation of a simple level, the creation of material and shader groups, the usage and application of materials and shaders, material and shader naming conventions, texture mapping, and miscellaneous and general tips and suggestions.

Getting Started

Before editing or creating content for Halo please make sure to follow the set up and update information found in the section Creating a [Development Environment](#). It is also recommended that the end user read through all the information found in the associated **Reference Sections** under **General**.

A completed version or example version of the level created in the following tutorial sections can be found under the “Halo\data\levels\test\tutorial_examples” directory. The source and related materials have been provided as a reference to aid in the learning process.

Setting Up 3ds Max

Make sure that the Blitzkrieg exporter plug-in for 3ds Max is installed, for more information see the **Halo Editing Kit Installation** topic under the Creating a [Development Environment](#) section. Each individual has their own style and settings that they prefer for 3ds Max the following are some suggestions and guidelines that may help with Halo editing. For additional information, please see the documentation that came with 3ds Max (associated manuals and in program User Reference under Help).

The following are some tips and pointers for setting up 3ds Max for Halo content editing, specifically game environment creation.

NOTE: All the setup suggestions were done using 3ds Max 5.1.

Auto Backup

It is HIGHLY recommended to enable the auto file backup features inherent in 3ds Max. This can save a lot of headaches and wasted time in lost work. To access the auto file backup features do the following:

1. Click on **Customize** and select **Preferences**
2. Click on the **Files** tab
3. On the bottom left click on the Enable check box under Auto Backup
4. Set the number of backup files and time intervals for the files to be backed up.
5. Hit **OK**

It is also highly recommended that work be saved and saved often and that alternative backups be made of all relevant content

Key Configuration

Specific key configurations are not going to be listed here, but suffice it to say, if 3ds Max is being used as a level design tool it would be most beneficial to bind commonly used commands and operations for quick access. Some of the more commonly used commands that may not be as easily accessible with the default key configuration include Move Mode, Rotate, Backface Cull Toggle, Maximize Viewport Toggle, Rotate View Mode, and Zoom Mode.

Configure Modifier Sets

Just as there are several commonly used commands and operations that are used when creating and editing Halo game levels, there are also very commonly used modifiers. These modifiers include Edit Mesh, UVW Map, Unwrap UVW, and STL Check (used for checking if the level is a sealed volume and/or for finding open edges).

Units Setup

1. Click on the menu option **Customize** and select **Units Setup**
2. Select Generic Units at the bottom
3. Hit **OK**

The example levels, content, and other numerical information (measurements for various Halo related statistics such as player dimensions) were done using the Generic Units settings.

View Port Configuration

1. Click on the menu option **Customize** and select **View Port Configuration**
2. Select the **Layout** Tab
3. Select the layout that suits you best, the example levels and tutorials use a 3 window layout with a 3d Perspective View at the top (Smooth and Highlight and Edge Faces turned on), Top Down View on the lower left and a Right\Left\Front view for the lower right window depending on the configuration of the level.
4. Hit **OK**

Creating a Level

The following sections contain various information and examples that form one large tutorial that will go through the creation of a simple box level that will be used to demonstrate the construction of a Halo multiplayer level.

Creation of a Level Directory

The level directory will contain all the raw data or source files used in the creation of the level (.max, .tif, .psd, .jms, etc...). This level directory will have subdirectories. The main directory as well as the subdirectories are specifically used by the tools to compile the exported data formats into something that can be compiled into a .map cache file that is used to run the level in Halo.

The name of the source directory must match the name of the exported .jms file (as will be explained later). The compiled data tags such as the .scenario and .scenario_structure_bsp tags that will exist in the tags directory will also share this name, as well as the .bitmap light map file once it is created by running the radiosity process on the level.

The level directory name **MUST BE UNIQUE**, unless it exists under a totally different directory structure.

As will be seen and demonstrated later, the level can pull shader tag information as well as other tag information from other tags that exist in other game level directories.

The first step is to create a level directory under the level directory structure tree under the data directory in Halo. This level will be called “tutorial”. Follow the steps below:

1. Under the Halo game directory (e.g. c:\Halo) create a directory called “**data**”.
2. Under the new “data” directory create a directory called “**levels**”.
3. Under the “levels” directory create a directory called “**test**”.
4. Finally, under the “test” directory create the level working directory called “**tutorial**”.

The directory structure should now be as follows (if using the above example (e.g. c:\Halo)):

c:\Halo\data\levels\test\tutorial

Now that the level directory has been created, three subdirectories will need to be created that will contain the content source for the level. As mentioned previously, the tools recognize this directory structure during the compilation process of the level and of the .map cache file so it must be properly setup.

Under the “tutorial” level directory add the following directories:

1. **bitmaps** (Using the previous example the directory structure would look like this:
c:\Halo\data\levels\test\tutorial\bitmaps)
2. **models** (Using the previous example the directory structure would look like this:
c:\Halo\data\levels\test\tutorial\models)
3. **scenery** (Using the previous example the directory structure would look like this:

c:\Halo\data\levels\test\tutorial\scenery)

The bitmaps directory is where .tif image files will be placed that can be compiled into .bitmap data tag files. The .tif image files in this directory can also be used for the displaying of the image on a surface in a level in 3ds Max (the .tif gets set as a Diffuse Map in the Sub-Material of a Multi/Sub-Object).

The models directory is where the source files for the level mesh will be saved and where the .jms file used by tool to compile a level BSP data tag will be put.

The scenery directory is where custom scenery model sub-directories and the associated resources will be placed.

This is the level directory structure used by Bungie to develop Halo and is the directory structure that Gearbox adopted in order to keep the source tree consistent. It is also the directory structure that the tutorial and examples will use.

NOTE: As will be seen later, when the data tags are created a directory and associated tags will be created or modified that mirrors the directory structure under data except this directory structure will be found under the “tags” directory under the root directory of Halo. In addition, when a .map cache file is created, the name of the cache file will be that of the main level directory structure used and will be found under the “maps” directory under the main or root Halo game directory.

Creation of a Reference Frame

A Halo level MUST have a reference frame. The reference frame is used as a reference point for the placement of objects in the level. Therefore, once the reference frame is placed and objects are placed in the level using Sapien, do not individually move the reference frame (or its associated polygons or vertices) or the objects will move or shift in the level. The reference frame is very useful during the construction of a Halo level. Any models or geometry that are not linked to the reference frame (the reference frame is considered the “parent”) do not get exported when exporting out the .jms file using the Blitzkrieg plug in. This can make testing and debugging the level very easy. It also means that reference models (such as for players or vehicles) can be placed in the level and they will never get accidentally exported.

It is also important to note that there must also be valid geometry attached or linked to the reference frame or the Blitzkrieg exporter will not export the .jms file and will give an error dialog “There was no geometry to export”.

Additional information on the Reference Frame and other technical rules or guidelines can be found under the **Technical Rules** discussion topic under the [General Overview](#) section under **Multiplayer Level Design**.

A reference frame is very simple to create, the following steps and example images will demonstrate the creation of a reference frame. The reference frame is basically a box. It can be any size (larger is preferred to help with its selection and to make sure it does not get confused with other world geometry) and can even be a sphere, cone, cylinder or pyramid. As long as it is an object that is an enclosed volume.

The reference frame should be created such that it does not interfere with editing and will not obscure parts of the level. For the example, the reference frame will be created such that it exists outside of the “northern” border of the example level.

The following section is a continuation of Level Creation - Part 1 and provides more detailed information on creating Multiplayer levels and environments and includes tutorials and examples. Topics covered in the tutorials and examples include more information on the Sealed World Rules, the additional creation of level geometry including structures, streams, and terrain features, a discussion on water and how it exists and behaves in Halo, additional applications of smoothing groups, the creation and placement of items such as light fixtures and ladders, the placement of visibility portals for rendering, an explanation of player clipping\containment techniques, and miscellaneous and general tips and suggestions.

Getting Started

This section assumes that the end user has successfully completed and understands Level Creation - Part 1 and any additional reference sections. The following tutorials, examples, and images will be using the file “tutorial.max” which was created in the tutorials and examples in Level Creation - Part 1.

Modifying the Level

The following tutorials, examples, and images will help demonstrate additional features for Halo levels through the modification of the level geometry.

Terrain Modification

The following steps and example images will demonstrate the modification of the surfaces that compose the floor of the level to create some simple hills that will act as a border for the level.

Modification of the level floor to create some simple terrain and surrounding hills: 1) Open 3ds Max and load the tutorial.max file from the Halo\data\levels\test\tutorial\models directory and select the level (object named “level”). 2) Select Modify. 3) Under Selection choose Face mode. 4) Select the 2 faces that compose the bottom or floor of the box which is currently the level (its easier to select these faces when	
--	--

Ignore Backfacing is selected under Selection, see the 3ds Max User Reference under Help for more information).

The first image further explains the above procedures.

5) The selected faces are now going to be Tessellated so that there are more polygons or faces to work with in order to create terrain. To subdivide or tessellate the selected faces do the following:

a) In the Weld area under Edit Geometry, enter 0.0 in the Edge tension value field.

b) In the Weld area under Edit Geometry, click the Tessellate button 4 times.

The second image further explains the above procedures and shows the final result of the tessellation procedure.

6) The Tessellation procedure will invalidate the assigned texture coordinates or texture mapping. To help serve as a guideline for the modification of the faces, the polygons or faces will be reassigned texture coordinates.

a) With the faces that compose the floor selected, select UVW Map from the Modifier List pull-down menu.

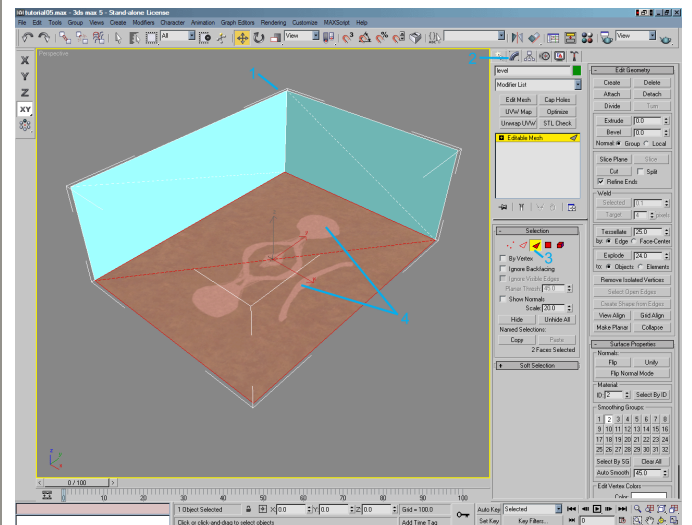
b) Select the Top view.

c) In the Alignment area under Parameters, click on the View Align button.

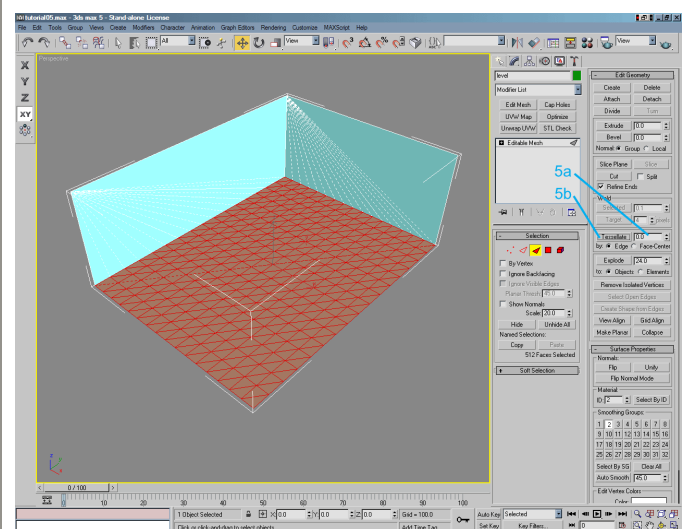
d) In the Alignment area under Parameters, click on the Fit button.

The third image further explains the above procedures.

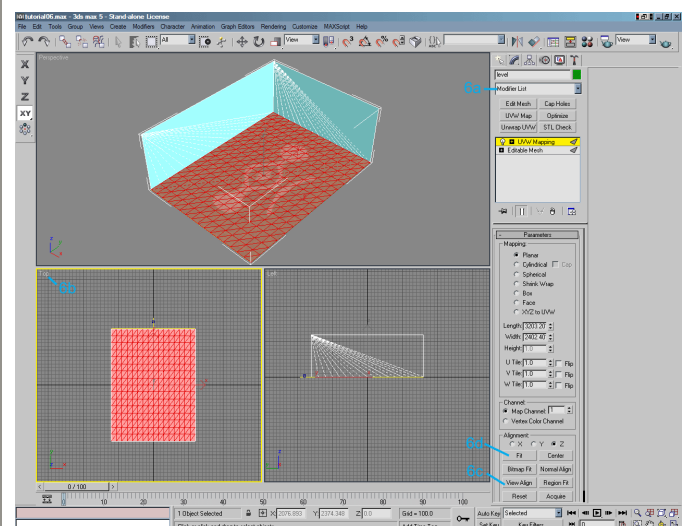
The polygons around the edges of the level (where the terrain meets the sky) are going to be modified such that they create hills around the edge of the tutorial level. There are many methods to achieve this including moving the vertices or moving the faces either individually or in groups. The final result will be that the surround hills will be approximately 128 units high. One such technique for making the hills are listed in the procedures below.



[Click to Open Larger Image in New Window](#)



[Click to Open Larger Image in New Window](#)



[Click to Open Larger Image in New Window](#)

7) To edit the polygons and vertices that compose the terrain, the selected object need to be converted to a mesh again.

a) With the level selected, select Edit Mesh from from the Modifier List pull-down menu.

b) Choose the polygons around the edges of the level such that at the “North” and “South” borders the polygons selected are 1 “row” deep and that the “East” and “West” borders are 2 “rows” deep.

The fourth image further explains the above procedures.

To create the hills and smooth the faces out follow the procedures below:

8) With the border faces still selected, move the faces up 128 units in the Z-axis.

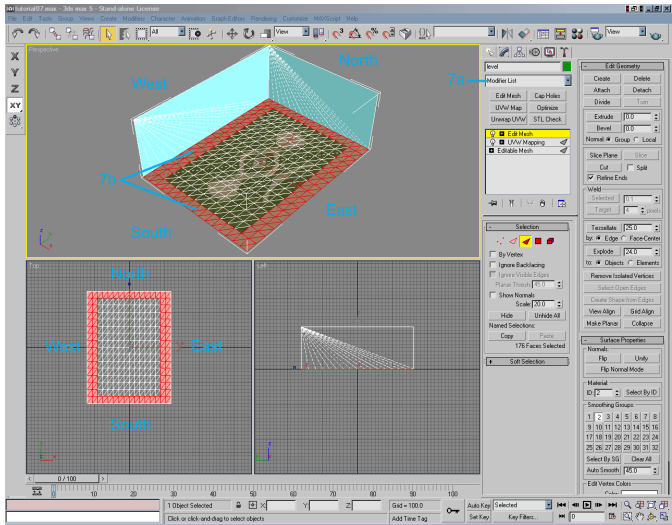
The fifth image further explains the above procedures.

9) Select faces or vertices and move them to give the bordering hills some variation. Edges were turned and vertices were manually tweaked to smooth out the surround hill geometry.

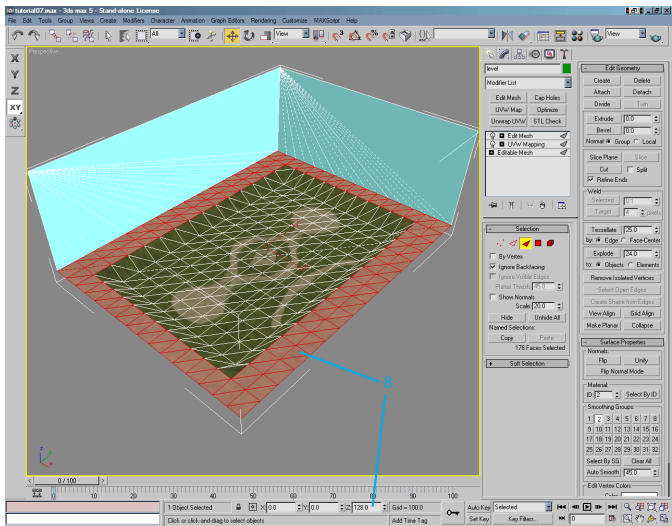
Turning edges is just basically taking two faces (in this case 2 triangles) and reconnecting or turning the edge such that its orientation is changed. The triangle count is preserved as well as the non-turned edges, but the selected edge of the triangle now connects to different vertices than the original configuration. The turn edge functionality (Turn) can be found under Edit Geometry when Edge mode is selected while the object is set as an Editable Mesh (Edit Mesh).

See the User Reference in 3ds Max under Help for more information on moving faces, vertices, and turning edges.

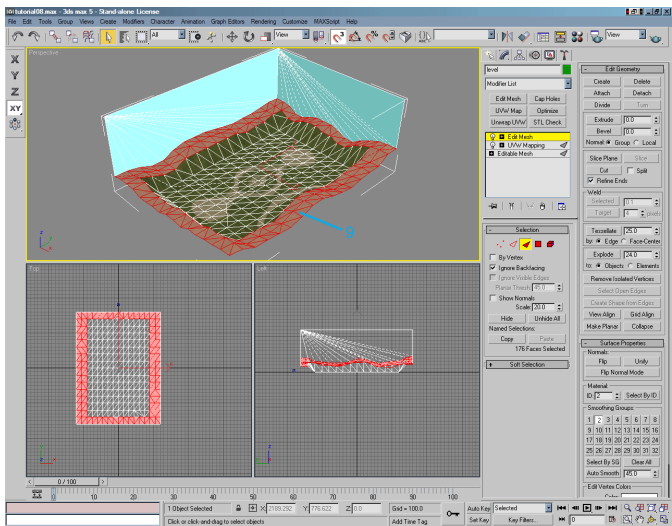
The sixth image or last image further explains the above procedures and shows the final results.



Click to Open Larger Image in New Window



Click to Open Larger Image in New Window



Click to Open Larger Image in New Window

The following steps and example images will demonstrate the modification of the surfaces that compose the floor of the level to create simple outdoor terrain.

Modification of the ground mesh to vary the existing terrain: 1)
To create a small hill in the middle of the level do the following:

a) Select the level (object named “level”).

b) Select Modify.

c) Under Selection choose Face mode.

2) Select the 8 faces that are in the middle of the level.

3) Move these faces up to create a small hill.

The first image further explains the above procedures.

4) To smooth and round the hill do the following:

a) Under Selection select Vertex mode.

b) Select the vertex in the middle of the hill.

c) Move the vertex up in the Z-Axis to help round the hill.

The second image further explains the above procedures.

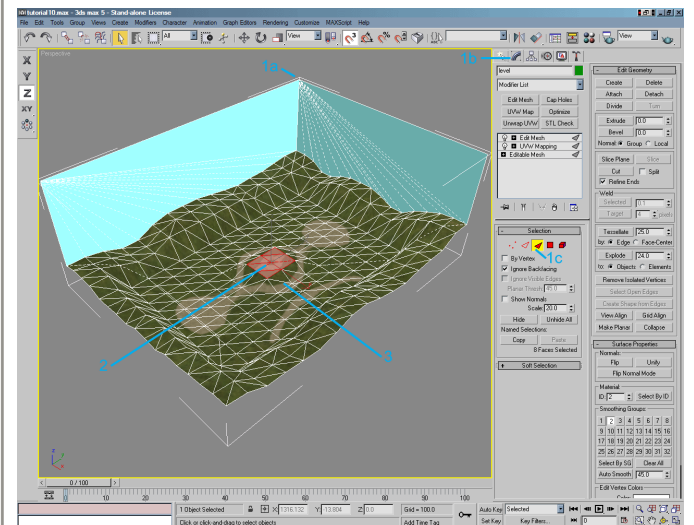
5) Move the faces and vertices and turn the edges to smooth out the hill and to make it follow the dirt paths outlined in the texture.

The third image further explains the above procedures.

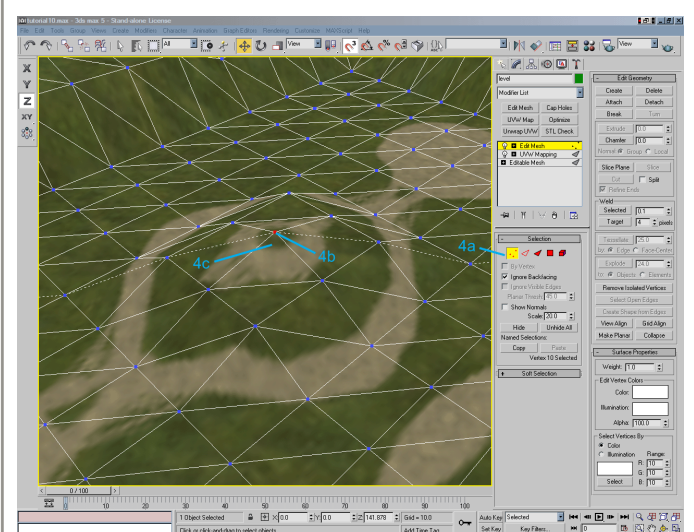
Depending on how the terrain is modified it may be necessary to adjust or reassign the texture coordinates. Follow the previous procedures for UVW Mapping or projection texture mapping the ground.

6) Rough up the surrounding ground by creating a small rolling hill on the “West” border of the level. Do not modify the “East” border or area of the level for right now.

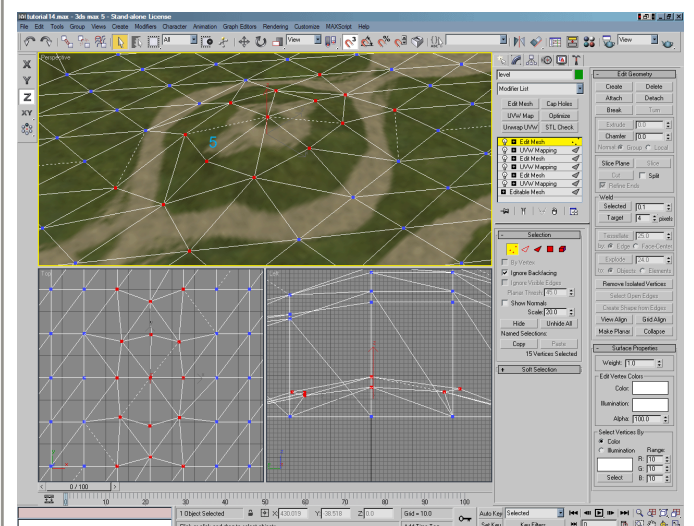
7) Smooth the transition between the ground and bordering hills by moving vertices and faces and turning edges. Again, avoid



Click to Open Larger Image in New Window



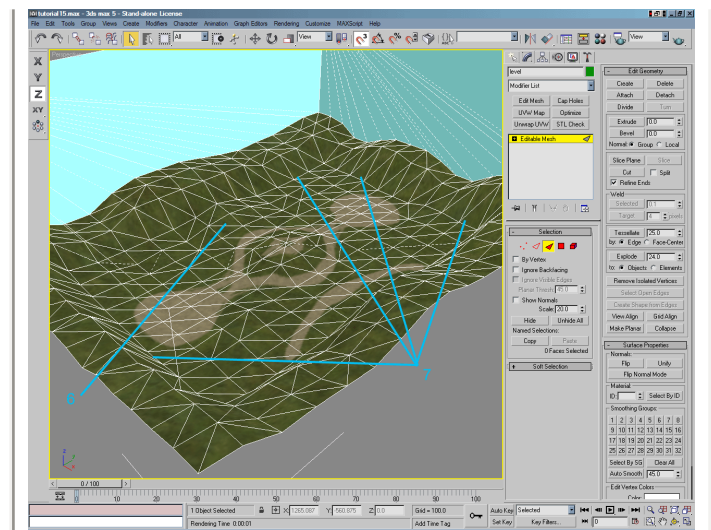
Click to Open Larger Image in New Window



Click to Open Larger Image in New Window

modifying the area on the “East” border of the level.

The fourth image or last image further explains the above procedures and shows the final results.



[Click to Open Larger Image in New Window](#)

The following steps and example images will demonstrate the modification of the terrain to create a stream bed.

Modification of the ground mesh to create a stream bed. 1) To create a stream bed on the “East” border of the level do the following:

a) Select the level (object named “level”).

b) Select Modify.

c) Under Selection choose Face mode.

2) Select the faces on the “East” side of the level such that they encompass the part of the ground texture that represents the stream bed.

The first image further explains the above procedures.

3) To create the stream bed, more polygons and vertices are needed. To get a higher resolution mesh, the selected faces are going to be subdivided or tessellated as was done with the original terrain mesh. Follow the procedures below to tessellate the selected polygons.

a) In the Weld area under Edit Geometry, in the Edge tension value field enter 0.

b) In the Weld area under Edit Geometry, click the Tessellate button once.

The second image further explains the above procedures and shows the final results of the tessellation procedure.

4) Turn the edges that border the outline of the stream on the texture such that the edges follow the stream bed.

5) Create the stream bed by moving the vertices in the middle of the stream down. To do this, follow the procedures below.

a) Under Selection select Vertex mode.

b) Select the vertices that run approximately up the middle of the stream bed outlined in the terrain texture and move the vertices down in the Z-axis.

6) Move the faces and vertices and turn the edges to smooth out the banks that border the stream and to make it follow the stream bed as outlined in the texture.

In addition, the position in the Z-axis of the vertices that run up the middle of the stream bed were varied.

7) Finally, the faces around the stream as well as the vertices and faces that intersect the ground and border hills were modified to smooth out and vary the terrain features.

Depending on how the terrain is modified it may be necessary to adjust or reassign the texture coordinates. Follow the previous procedures for UVW Mapping or projection texture mapping the ground.

The third image or last image further explains the above procedures and shows the final results of the above procedures.

[Attach:hektut11.jpg](#)  

Click to Open Larger Image in New Window [Attach:hektut12.jpg](#)  

Click to Open Larger Image in New Window [Attach:hektut13.jpg](#)  

Click to Open Larger Image in New Window

Creation of a Simple Stream

The following steps and example images will demonstrate the creation of geometry that will be used to add a stream to the existing level.

Creation of the stream geometry: 1) To create the surface of the stream do the following:

- a) Deselect the level (object “level”) and select Create.
- b) Make sure that Standard Primitives is selected in the pull down menu.
- c) Under Object Type, click on the Plane button.
- d) In the Parameters area, set the Length Segs value to 8 and the Width Segs value to 1.

The first image further explains the above procedures.

2) Create the plane such that it roughly covers the area of the stream bed.

3) The stream is going to be made a separate object from the object “level”.

Under Name and Color, name this object “stream”, the object color was also set to blue, this is not necessary but makes it easier to distinguish this object.

The second image further explains the above procedures.

The plane of polygons will now be manipulated so that it fits the stream bed.

4) Follow the procedures below to modify the plane so that it makes a stream.

a) Select Modify.

[Attach:hektut14.jpg](#)  

Click to Open Larger Image in New Window [Attach:hektut15.jpg](#)  

Click to Open Larger Image in New Window [Attach:hektut16.jpg](#)  

Click to Open Larger Image in New Window [Attach:hektut17.jpg](#)  

Click to Open Larger Image in New Window

b) The plane needs to be converted to a mesh so that it can be manipulated. Select Edit Mesh from from the Modifier List pull-down menu.

c) Under Selection select Face mode.

d) Select all the faces for the stream.

e) Move the selected faces down in the Z-axis to form the surface of the water. The plane of polygons should be slightly below the lowest edge or bank of the stream bed.

The third image further explains the above procedures and shows the final result of the performed procedures.

5) Edit the vertices to make the polygons that compose the object “stream” conform to the shape of the stream making sure that all edges extend into or past the geometry that composes the stream bed.

6) The stream is going to be setup such that it “flows” from North to South.

Notice how the edges of the polygons are positioned such that they are either parallel to the flow direction or perpendicular to the flow direction of the stream. This will make the texture mapping or application of texture coordinates for the stream shader (that will be scrolling to simulate the water moving) easier.

The fourth image or final image further explains the above procedures and shows the final result.

The following steps and example images will demonstrate the creation and application of the Multi/Sub-Object material group and Sub-Material needed to complete the stream.

Creation of the stream Material Group\Multi/Sub-Object: 1) In 3ds Max select the stream object. Open the Material Editor.

2) Select an empty Material Group. This Material Group is currently set to a Standard Material.

3) Click on the Standard button to bring up the Material/Map Browser.

4) In the Material/Map Browser click on Multi/Sub-Object.

5) Click on OK.

The first image further explains the above procedures.

6) This Multi/Sub-Object is going to be called “stream”. Type this name into the Multi/Sub-Object description window.

The name does not matter since it is not exported, the Multi/Sub-Objects are being used to organize and apply like groups of materials to objects being created for the level. This will be demonstrated repeatedly in the tutorials and examples.

7) Now apply the Multi/Sub-Object to the stream object. Since the “stream” object is selected, click on Assign Material to Selection. This Multi/Sub-Object has now been assigned to the “stream” object. The Sub-Materials for this Multi/Sub-Object can now be applied to the object using the materials associated ID number.

8) Just for organization, the extra Sub-Materials were deleted by selecting the Sub-Material to delete and clicking on Delete until the desired number of Sub-Materials remained, in this case, 1 Sub-Material.

The single Sub-Material is now going to be named the name of the shader which will be applied to the surfaces and appear in the game.

9) To name or label Sub-Material ID 1 do the following:

a) Click on the Sub-Material button to go from displaying the Multi/Sub-Object Basic Parameters window in the Material Editor to displaying the Shader Basic Parameters, Blinn Basic Parameters, Extended Parameters, [Super Sampling?](#), Maps, Dynamic Properties and Viewport Manager.

The second image further explains the above procedures.

b) Enter the name “example_tutorial_stream” into the name or label field.

10) To display a texture or image for this material when applied to a surface do the following:

a) Under Blinn Basic Parameters, click on the small box to the right of the Diffuse color setting. This will display the Material/Map Browser.

b) Click on Bitmap.

c) Click on OK and the Select Bitmap Image File window will now be displayed.

The third image further explains the above procedures.

11) To select an image to display for the material do the following:

a) In the Select Bitmap Image File go to the bitmaps directory under Halo\data\levels\test\tutorial_examples.

This directory SHOULD exist if the Halo Editing Kit was installed and set up properly following the instructions under Halo Editing Kit Installation in the Creating a Development Environment section under General Reference.

The files in the tutorial_examples level represent the final result of the tutorial and example process and is provided as a reference to aid in the learning process.

b) Under the bitmaps directory there should be several .tif image files. Select the example_tutorial_stream.tif file.

As was mentioned previously, the resources under the tutorial_examples directory represent the end result of all the examples and tutorials contained within the Halo PC Editing Reference.

The Halo End User Editing Kit provides many resources including a wide range of tags as well as image resources (.tif files) that can be

[Attach:hektut18.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut19.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut20.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut21.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut22.jpg](#)  

[Click to Open Larger Image in New Window](#)

used in the creation of Halo multiplayer levels.

The files provided also exist as examples and learning aids for those wishing to create their own custom game resources whose creation and modification may not be covered currently in the Halo PC Editing Reference.

For now, the completed or provided example resources will be used for the following examples for the sake of simplicity.

c) Click on Open.

The fourth image further explains the above procedures.

To return to the previous options in the Material Editor window click on the Go to Parent button.

12) In the Material Editor, enable the display of the material by clicking on the Show Map in Viewport button.

13) The Sub-Materials should be displayed on the Material Sample.

The stream geometry that has been created does not currently follow the Sealed World Rules, it has open edges. This will be solved by adding a Special Shader Symbol to the material name.

14) In the name or label field add a “!” character to the end of the name “example_tutorial_stream”.

The name of the Sub-Material should now be “example_tutorial_stream!”.

The Shader Symbols table located in the General Reference section Materials Overview shows that the “!” symbol tells Tool that when this surface is compiled into the level data tag do NOT associate any collision information with it.

This surface is now RENDER ONLY. Because of this, it can have open edges and now satisfies the Sealed World Rules.

15) To return to the Multi/Sub-Object Basic Parameters click on the Go to Parent button.

16) Close the Material Editor window.

The fifth image or last image further explains the above procedures

With a material having been created and assigned to the faces that compose the stream object, the faces must have mapping coordinates created for them. These mapping coordinates or “texture mapping” will allow the base maps or textures that are in the form of a .bitmap data tag and are referenced in the shader data tag to properly display for the specific faces.

The following procedures and example images will demonstrate using Unwrap UVW to apply texture or mapping coordinates to the faces that compose the stream geometry.

Application of texture coordinates to the faces that compose the stream geometry: 1) Select the stream.

2) Select Modify.

3) Under Selection, select Face mode.

4) Select all the faces that compose the stream geometry.

5) Using the Modifier List pull-down menu select Unwrap UVW.

The first image further explains the above procedures.

Apply the texture coordinates to make the texture for the stream follow the shape of the geometry.

6) Click on the Edit button under Parameters. This will display the Edit UVWs window.

7) In the Edit UVWs window click on the Show Options button at the bottom to show additional properties.

8) In the Edit UVWs window under Bitmap Options enter in the Width field 64 and enter in the Height field 512.

The example_tutorial_stream.tif texture should now display properly.

The second image further explains the above procedures.

[Attach:hektut23.jpg](#) ▲▲

[Click to Open Larger Image in New Window](#)

[Attach:hektut24.jpg](#) ▲▲

[Click to Open Larger Image in New Window](#)

[Attach:hektut25.jpg](#) ▲▲

9) Select Move mode in the Edit UVWs window.

10) Map the stream texture to the faces of the stream geometry by moving the vertices.

11) Once this is done, close the Edit UVWs window.

The third image further explains the above procedures and shows the final result.

12) To make the stream part of the level, it must be linked or attached to the reference frame.

a) Select the stream object.

b) Click on Select and Link.

c) Click on Select by Name, this will display the Select Parent window.

d) In the Select Parent window select “frame”.

e) Click on Link.

The fourth image or last image further explains the above procedures.

[Click to Open Larger Image in New Window](#)

[Attach:hektut26.jpg](#)  

[Click to Open Larger Image in New Window](#)

The stream now has a material and mapping coordinates applied to it that once viewed in game will scroll and give the appearance of running water. For additional water effects the stream bed will have a different material applied to it than the currently applied terrain material. Such additional water effects include: the sound of water splashing when the player runs through the water or stream bed, the water particle effects when the player runs through the stream or stream bed, and the explosion effects and particle effects that occur when such an event occurs in the stream.

It should be noted that the steam in the examples and tutorials is the simplest case of creating or “simulating” water. Before continuing, a short discussion on water and water volumes and how they exist and behave in Halo is in order.

Water and Water Volumes in Halo

Water in Halo is very simple, almost to the point where water does not truly exist (at least in the sense that most people that have edited or created water in other games are used to). This is understandable since there

are no specifically designed underwater missions or environments in Halo. In the case of streams, the water is shallow and the player cannot get under the surfaces of the water and be completely immersed. Couple these facts with the game play mechanic that the player in the role of the Master Chief has a self contained suit of armor and has no fear of drowning, there was no need for complicated underwater player behaviors, physics, and effects.

The majority of water encountered is in the form of simple streams. This kind of “water” has 2 parts: a stream surface with a shader whose settings simulate a water surface (small waves, flowing water, etc...) and the terrain underneath which has specific settings so that when objects interact with the terrain water effects occur (particles, sprites, sounds, etc...) giving the appearance that the stream itself is water. It is this kind of water that is demonstrated in the tutorial below.

There is one instance in the game where a large volume of water exists that the player can interact with or go “completely under water”, this occurs for the island level in single player Halo, Silent Cartographer and in its multiplayer derivative, Death Island.

In this instance, the water is still very simple. Just like the stream example, there are the water surfaces that have the water shader effects and the terrain or surfaces underneath that are setup to simulate water with the effects that occur when objects interact with these surfaces. In the case of the water surfaces, these are defined with a `.shader_transparent_water` where as the streams use `.shader_transparent_chicago` or `shader_transparent_chicago_extended` shader tags. The water is made to have no collision using the “!” Render Only Shader Symbol just as with the stream example below.

The combination of the `.shader_transparent_water` and water fog volumes described below determine additional game behaviors such as the ability of vehicles to travel over the water (Ghosts will eventually stop and sink).

For large volumes of water, a water fog volume is created. This is done by adding the fog plane Shader Symbol “\$” to the material names for the water surfaces, or a similar plane is created with another fog plane defined shader assigned to the surfaces of the plane. The volume is defined to be a water fog volume by enabling the “is water” flag located in the `.fog` tag.

Fog planes or volumes (suchs as those seen throughout the game and in the chasm in the multiplayer level Gephyrophobia) as well as water fog volumes are all created in a similar manner.

Make a plane that covers the entire area and extends past the level borders (this plane does not have to follow the Sealed World Rules), face the normals for the faces of the plane upwards, apply a material with the fog plane symbol or use the `+unused` special material (this material has no collision nor will it draw) with the fog plane symbol (to create the material `+unused$`). In Sapien, create a fog reference (a `.fog` tag that contains fog color and density as well as other properties) for the “fog palette” and then enter the volume and set the `.fog` palette entry to be applied to this volume.

The following procedures and example images will discuss the implementation of materials to achieve water effects. The faces that compose the stream bed will be assigned a material that will give the stream the appearance of being water when the player and other game objects interact with it.

Creation and application of the stream bed material: The stream bed material will be identical to the surrounding terrain material except for the fact that its physics material settings for the shader will be set to “water” instead of “dirt”, this can be observed in the provided example shader files previously mentioned.

This new material will be added as a new Sub-Material to the “tutorial terrain\sky” Multi/Sub-Object.

1) Open the Material Editor and select the “tutorial terrain\sky” Multi/Sub-Object.

2) Add a third Sub-Material by clicking on Add.

3) Set up this new Sub-Material exactly like Sub-Material ID 2 (example_tutorial_ground) except name this Sub-Material “example_tutorial_streambed”

4) Close the Material Editor.

The first image further explains the above procedures.

The new example_tutorial_streambed Sub-Material will now be applied to the stream bed. Follow the procedures below.

5) Select the level.

6) Select the Modify tab.

7) Select Face mode.

8) Select the stream bed faces, this includes all faces under the stream surface as well as those faces that partially intersect the stream surface.

9) Set the faces to Material ID 3. Leave the Smoothing Group still set to 2 since we want the lighting for the terrain and stream bed to be blended together.

The second image or last image further explains the above procedures.

[Attach:hektut27.jpg](#)  

Click to Open Larger Image in New Window

[Attach:hektut28.jpg](#)  

Click to Open Larger Image in New Window

Creation of a Simple Base Structure

The following procedures and examples will show the construction of a simple base structure. Eventually, the structure will be “mirrored” or copied to create a second opposing base structure, this will be covered in subsequent tutorials along with the creation of Multi/Sub-Object material groups and the application of the Sub-Materials and texture coordinates to the faces of the base structure.

The following procedures and example images will demonstrate the creation of a simple base structure.

Creation of the base structure geometry: When creating a level, particularly indoor environments and/or buildings or structures, it is always helpful to have a scale reference. The dimensions of various game elements of Halo, including the player dimensions, are available for reference in the Halo Player Statistics.

Another technique is to use a reference model. The Halo Editing Kit includes a simple reference model for the player (the Master Chief) and a vehicle (the Warthog). The following procedures will demonstrate how to merge or import a reference model for use as scale reference for the creation of the base structure geometry.

- 1) In 3ds Max go to File and select Merge. This will open the Merge File window.
- 2) In the Merge File window go to the master_chief directory under the Halo\data\characters\reference directory structure.
- 3) In this directory there will be a master_chief.max file, select this file.
- 4) Click on Open. The Merge window will now appear.

The first image further explains the above procedures.

- 5) In the Merge window, select the Master Chief object in the list.
- 6) Click on OK.

The Master Chief reference model will now appear in the level and can be moved and rotated as necessary. For this tutorial,

the reference model will be moved to the southern dirt patch of the level where the base structure will be constructed.

Note that the model may appear below the terrain but will be selected, move the model up to make it visible.

The reference model will NEVER be linked to the reference frame and only exists as a reference tool. Since the model is not linked to the frame, it will never be part of the level and will never be exported. Therefore, the model can be kept in the .max file and does not have to be deleted each time the level is exported to a .jms.

The second image further explains the above procedures.

7) To create the core geometry of the base structure follow the procedures below.

a) Select Create.

b) Select Box and create a box that has the following dimensions: Length: 280.0 Width 96.0 Height 96.0 and that has Length Segs: 2 Width Segs: 1 and Height Segs: 1.

c) Position and align the box such that it is roughly in the center of the dirt spot but slightly more towards the center of the level.

The third image further explains the above procedures.

The box will now be manipulated to create a ramp and simple hallway.

8) To make the front ramp to the base do the following:

a) Convert the box to a mesh (Edit Mesh).

b) Make the first segment of the box into a ramp by welding (Vertex Mode and Weld) the top 2 vertices to the bottom 2 vertices.

The fourth image further explains the above procedures and shows the final result of the above procedures.

[Attach:hektut29.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut30.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut31.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut32.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut33.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut34.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut35.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut36.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut37.jpg](#)  

[Click to Open Larger Image in New Window](#)

[Attach:hektut38.jpg](#)  

[Click to Open Larger Image in New Window](#)

9) To create the simple hallway the side faces of the back segment for both sides are deleted as well as the bottom faces for the two segments.

The fifth image further explains the above procedures and shows the final result of the above procedures.

10) Extend the hallway by extruding the side edges (Edge Mode, Select the Edges, and Shift-Drag)

11) The ramp was slightly extended also by just selecting the front vertices and moving them forward.

The sixth image further explains the above procedures and shows the final result of the above procedures.

12) The faces that form the top of the base were hidden to make the construction of the internal hallway and features easier.

Vertices were then created and aligned to form the beginning of the doorways for the hallway. The top vertices were scaled inwards to add a slight taper to the door way once it is completed.

Note that the Master Chief reference model was moved to help determine how big the hallway and entrances\exits should be.

The seventh image further explains the above procedures and shows the final result of the above procedures.

13) Using the newly created and existing vertices, polygons or faces were created to make the sides of the bases with the hallway doorways cutout.

14) The floor and ceiling for the hallway were also created by creating polygons or faces using the newly created vertices.

The normals for the visible collideable faces must point in the proper direction.

The eighth image further explains the above procedures and shows the final result of the above procedures.

15) The faces that compose the hallway ceiling were hidden to

make the viewing and construction of the hallway walls easier. The hallway walls were created by creating polygons or faces using the existing vertices. The ninth image further explains the above procedures and shows the final result of the above procedures. 16) All of the faces were unhidden and the base was named “blue base” by entering this name into the object name field. The tenth image or final image further explains the above procedures and shows the final result.	
--	--

The following procedures and example images will demonstrate the modification of the base structure and the addition of ladders.

Creation of ladder guides and ladder faces: When adding ladders to a level, it is always good to provide some geometry that helps “guide” the player while getting on or off the ladder and that helps keep the player from falling off or shifting on the ladder as the player travels along the ladder face. 1) Two ladder guides or grooves are created on the back side or “South” side of the base structure. 2) The guides or grooves will provide an inset for the ladder faces or polygons to be placed. Since the ladder faces must be offset from the back face of the grooves and cannot be created using the front vertices additional vertices and faces are created midway on the sides of the inset grooves. The reason for this is because if the ladder faces were created using the front vertices multiple edges would be created. Multiple edges are invalid due to the fact that they break the Sealed World Rules. 3) The edges at the start of the ladder guides or grooves are beveled or angled to help guide the	Attach:hektut39.jpg ▲ ▲ Click to Open Larger Image in New Window Attach:hektut40.jpg ▲ ▲ Click to Open Larger Image in New Window
--	--

player onto the ladder as the player approaches the ladder faces from any angle.

The first image further explains the above procedures and shows the final results of the geometry modifications.

4) The faces that will be become the actual ladders are added using the middle vertices set into the grooves created for the ladders.

The second image or last image further explains the above procedures and shows the final results of the above procedures.

The following procedures and example images will demonstrate the modification of the base structure for the inclusion of a teleporter gateway

Creation of a teleporter gateway:

When adding teleporters to a level, it is always good to provide some geometry that helps “guide” the player on entering or exiting a teleporter. This teleporter guide or gateway is also useful in visually indicating to the player the existence of the teleporter as well as the area they must enter in order to trigger the teleporter.

1) A doorway or gateway for the teleporter is created on the back side or “South” side of the base structure.

The first image further explains the above procedures and shows the final results of the geometry modifications.

2) The visible teleporter “field” is added by creating faces that fit the opening but are NOT attached to the base meshes (the vertices and associated edges do not align but the edges are flush with the surrounding surfaces or faces).

The faces are then inset. The faces created form a curved surface for aesthetic purposes.

[Attach:hektut41.jpg](#)  

Click to Open Larger Image in New Window [Attach:hektut42.jpg](#)  

Click to Open Larger Image in New Window

These faces will be made render only later when materials are applied just as the stream faces were and therefore these faces may have open edges since they will be non-solid (no collision). The netgame flags required to make the teleporter work will be placed using Sapien and will be demonstrated in the Population section. The second image or last image further explains the above procedures and shows the final results of the above procedures.	
--	--

The following procedures and example images will demonstrate the addition of lights to the base hallway.

Addition of lights to the base hallway: The tunnel for the base will need some basic lighting. Lighting and radiosity will be covered more in depth in the Radiosity (Lighting) section. 1) Two lights were added to the tunnel by creating 2 faces for each light and then moving these faces slightly off the surface of the tunnel ceiling (this prevents z-buffering render anomalies and Tool compilation errors). Note that these lights currently do not follow the Sealed World Rules since they have open edges. As was demonstrated with the creation of the stream in the level, these faces will be made “render only” and will have no collision once the appropriate material is applied to the faces. These faces will then comply with the Sealed World Rules. In this case, the light faces WILL be visible AND cast light for use in radiosity. As will be covered in the Radiosity (Lighting) section, faces can emit light and not render, or render and not emit light, or both. The Master Chief reference model was moved or hidden in the example image. The image to the right further explains the above procedures and shows the final results for the procedures outlined above.	Click to Open Larger Image in New Window
--	---

Separation of the Level into Separate Mesh Objects

Before creating the Multi/Sub-Object material group for the base structure created above and applying the materials, texture coordinates, and smoothing groups to the base structure faces, the structure must be made part of the level and satisfy the Sealed World Rules. There are several ways to build a level. It is perfectly valid to make the entire level a single mesh that satisfies the Sealed World Rules. It is also perfectly valid to create a single **Multi/Sub-Object** that contains all the Sub-Materials for the level and apply it to this single large mesh.

Since there are so many parts to a level and a level can become quite large and cumbersome to work with, it is very advantageous to separate the level into several objects. These objects can then have their own individual **Multi/Sub-Object** material group. This separation of mesh parts and material groups greatly helps in the organization of the level and greatly decreases the cumbersome nature of dealing with a single large mesh when creating something as complicated as a game environment.

It is the latter approach that will be followed and demonstrated in the proceeding tutorials and examples.

When the level data is exported using Blitzkrieg, only the vertices and subsequent face data (material type\name, smoothing group, texture coordinates, etc...) are exported. Therefore, objects that meshes are separated into only exist in 3ds Max. Because of this, as long as the vertices and related edges match exactly between the mesh objects (satisfying the Sealed World Rules), the exported data will reflect this. The level will export and compile properly.

The following procedures and example images will demonstrate the merging of the base with the level terrain to create polygons to seal the world and to match the relevant vertices and associated edges. The faces that compose the base will then be detached from the level mesh to create a separate object that will have a separate Multi/Sub-Object material group. The base object will then be linked to the reference frame.

The following procedures and example images will demonstrate the merging of the base with the level terrain to satisfy the Sealed World Rules.

Merging the base structure with the level object: 1) Select the level (object named “level”). 2) Under Edit Geometry click on Attach to activate attach mode. 3) Now click on the base object (object named “blue base”). The base has now been merged and is a part of the level geometry. The base may change color or have a texture or	
--	--

textures suddenly applied to it, this does not matter since it will have materials and texture coordinates properly assigned to it later. This is the result of merging the faces of the base (which have a default material id number associated with them) with an object that already has a material assigned to it.

The first image further explains the above procedures and shows the final results of the above procedures.

4) Toggle Attach mode off by clicking on Attach again.

5) Select all the faces or vertices that compose the base structure and then move the base down in the Z-Axis so that it is flush or even with the terrain underneath it.

6) Delete the ground faces that are adjacent or around the base.

The second image further explains the above procedures and shows the final results of the above procedures.

7) Create polygons to fill in the hole around the base and attach the base geometry to the surrounding terrain geometry.

Make sure to fill in the holes and gaps in the ladder guides (the area behind the ladder faces).

8) Assign these newly created faces to Material ID 2 and Smoothing Group 2.

9) Select all the faces that create the terrain and apply a UVW Map (Top down projection texture mapping) to fix the texture coordinates (as demonstrated before).

The third image or last image further explains the above procedures and shows the final results of the above procedures.

[Attach:hektut43.jpg](#)  

Click to Open Larger Image in New Window [Attach:hektut44.jpg](#)  

Click to Open Larger Image in New Window [Attach:hektut45.jpg](#)  

Click to Open Larger Image in New Window

The following procedures and example images will demonstrate the separation of the base into a separate object that will eventually be linked to the reference frame.

Separating the base structure from the level object: 1) Select the level. 2) Enter Face mode and select all the faces that compose the base structure. 3) Under Edit Geometry click on Detach. The Detach dialog window will appear. 4) In the Detach dialog window enter “blue base” in the Detach as: entry field, this will detach the faces to a separate object called “blue base”. 5) Click on OK. The image to the right further explains the above procedures.	Attach:hektut46.jpg ▲▲ Click to Open Larger Image in New Window
---	---

Application of Materials to the Base

Now that the blue base has been separated into a separate object, a separate Multi/Sub-Object material group will be created. The following steps and example images will demonstrate the creation of the Multi/Sub-Object material group and Sub-Materials needed to complete the base structure.

Creation of the base structure Material Group\Multi/Sub-Object: 1) In 3ds Max select the blue base object. Open the Material Editor. 2) Select an empty Material Group. This Material Group is currently set to a Standard Material. 3) Click on the Standard button to bring up the Material/Map Browser. 4) In the Material/Map Browser click on Multi/Sub-Object. 5) Click on OK. The first image further explains the above procedures.	
---	--

6) This Multi\Sub-Object is going to be called “base structures”. Type this name into the Multi\Sub-Object description window.

7) Now apply the Multi\Sub-Object to the blue base object. Since the “blue base” object is selected, click on Assign Material to Selection. This Multi\Sub-Object has now been assigned to the “blue base” object. The Sub-Materials for this Multi\Sub-Object can now be applied to the object using the materials associated ID number.

8) Just for organization, the extra Sub-Materials were deleted by selecting the Sub-Material to delete and clicking on Delete until the desired number of Sub-Materials remained, in this case, 8 Sub-Materials.

9) The Sub-Materials are now going to be named and linked to images (Diffuse maps). Since this has been demonstrated before, the Sub-Material properties will be listed in the table below.

Note: The .tif file reference is the image to be linked to in Diffuse Map settings.

The directory containing these images is the same as the previous examples: Halo\data\levels\test\tutorial_examples\bitmaps

Make sure that in the the Material Editor, to enable the display of the materials by clicking on the Show Map in Viewport button for each Sub-Material.

[Attach:hektut47.jpg](#) ▲▲

Click to Open Larger Image in New Window

[Attach:hektut48.jpg](#) ▲▲

Click to Open Larger Image in New Window

ID	Name	Diffuse Map (.tif file)
1	example_tutorial_metal	example_tutorial_metal.tif
2	example_tutorial_panels	example_tutorial_panels.tif
3	example_tutorial_metal_floor	example_tutorial_metal_floor.tif
4	example_tutorial_plate_floor	example_tutorial_plate_floor.tif
5	example_tutorial_ladder%^	example_tutorial_ladder.tif
6	example_tutorial_lights_blue!	example_tutorial_lights_blue.tif
7	example_tutorial_lights_red!	example_tutorial_lights_red.tif
8	example_tutorial_teleporter!	None (Diffuse color set to green)

Note that the Shader Symbols have been included. The Sub-Material example_tutorial_ladder%^ demonstrates that materials can have multiple Shader Symbols applied. In this case the ladder material has been made viewable from both sides or two-sided using the % symbol and has been made a ladder or climbable by the player using the ^ symbol.

The .tif files listed above were included with the Halo PC End User Editing Kit.

As before, the completed or provided example resources will be used for the following examples for the sake of simplicity.

Once all the Sub-Materials have been created, named, and linked to their reference images, close the Material Editor window.

The second image or last image further explains the above procedures and shows the results in the Material Editor.

Now that the blue base has been separated into a separate object and a separate Multi/Sub-Object material group has been created, the Sub-Materials as well as texture mapping coordinates will be applied.

The following steps and example images will demonstrate application of the Sub-Materials needed to complete the base structure.

Application of Materials and Texture Coordinates to the base and base surfaces: The following table describes the areas of the base and what material is applied to them.

Base Area
Material Applied
ID

All the faces that compose the teleporter gateway at the rear of the base example_tutorial_metal 1 Faces that compose the tunnel ceiling example_tutorial_metal 1 Faces that compose the side walls of the tunnel example_tutorial_metal 1 All the faces that compose the sides of the base (front, right, left and back faces including the ladder guide grooves. example_tutorial_panels 2 Tunnel floor faces example_tutorial_metal_floor 3 Top faces of the base and ramp example_tutorial_plate_floor 4 Ladder faces located at the rear of the base example_tutorial_ladder%^ 5 Base lights located in the tunnel example_tutorial_lights_blue! 6 Not applied to any surfaces for this base example_tutorial_lights_red! 7 Faces for the teleporter shield located at the rear of the base example_tutorial_teleporter! 8

Once the material is applied, the faces have texture coordinates assigned. The texture coordinates or texture mapping is applied to the surfaces using a combination of projection mapping (UVW Map) and UVW Unwrapping and mapping (Unwrap UVW).

The images to the right further explain the above procedures and several images have been provided showing the base from several angles to show the final base material application and texture alignment.

[Attach:hektut47.jpg](#) ▲▲

Click to Open Larger Image in New Window [Attach:hektut48.jpg](#) ▲▲

Click to Open Larger Image in New Window [Attach:hektut49.jpg](#) ▲▲

Click to Open Larger Image in New Window [Attach:hektut50.jpg](#)  

Click to Open Larger Image in New Window

Now that the blue base has had Sub-Materials as well as texture mapping coordinates applied to the surfaces that compose the base, smoothing groups will be applied.

The following steps and example images will demonstrate the application of smoothing groups to the surfaces of the base.

Assigning smoothing groups to the base surfaces: All the faces of the base structure were selected and all the smoothing groups were cleared (Clear All under Smoothing Groups while in Face mode.)

Smoothing groups were then applied to the surfaces of the base. There are many ways to apply Smoothing Groups to all the faces geometry. The Smoothing Groups below were applied manually.

The following table describes the areas of the base and what Smoothing Group is applied to these surfaces.

Base Area\Surfaces
Smoothing Group ID

Faces that compose the front of the teleporter gateway 8 Faces that compose the back of the teleporter gateway 8 Faces that compose the inset sides of the teleporter gateway 9 Faces that compose the tunnel ceiling 6 Faces that compose the inset floor of the teleporter gateway 2 Faces that compose the inset ceiling of the teleporter gateway 11 The external faces on the top of the teleporter gateway 10 The external faces that compose the sides of the teleporter gateway 12 Faces that compose the side walls of the tunnel 7 Faces that compose the sides of the inset grooves for the ladders 4 Faces that compose the sides of the base with the tunnel doorways 4 Faces that compose the sides of the ramps 4 Faces that compose the front faces of the base 5 Faces that compose the walls on the rear of the base 13 Tunnel floor surfaces 3 Top surfaces of base and ramp 3 Ladder surfaces located at the rear of the base None Base lights located in the tunnel None Faces for the teleporter shield located at the rear of the base None

Creating the Opposing Base Structure

Now that the blue base structure has been completed, it can be copied and used to construct a second base structure on the opposite end of the level. The following procedures will demonstrate the structure being “mirrored” or copied and rotated 180 degrees to create a second opposing base structure. Both base structures will then be attached as to the reference frame.

Copying and mirroring the base structure:

- 1) Select the blue base.

Clone the object (Ctrl V). This will bring up the Clone Options window.

2) Make sure that Copy is selected under Object. Under Name enter the name “red base”.

3) Click on OK.

The first image further explains the above procedures.

As mentioned before, constructing the level such that it is centered on the origin can help in the creation of the level, especially when having to duplicate or mirror geometry such as base structures.

In the procedures below, the red base objects Pivot point will be moved to the origin of the level (X: 0.0 Y: 0.0 Z: 0.0) and the newly copied or created “red base” will be rotated about this point.

Using this technique also insures that structures are equally distanced from features in the level, thus making the level more balanced with respect to game play for the player or for teams.

4) With the red base object still selected go to the Hierarchy tab.

5) Under the Adjust Pivot section, click on Affect Pivot Only. The pivot point for the base object will appear.

6) Move the pivot point to the origin of the level (X: 0.0 Y: 0.0 Z: 0.0).

The second image further explains the above procedures.

7) With the red base object still selected, go to the Modify tab.

8) Rotate the base 180 degrees around the Z-Axis. The easiest way to do this is to follow the procedures below:

a) Enter Rotate mode by clicking on the Rotate mode button.

b) Right Click on the Rotate mode button to bring up the Rotate Transform Type-In window.

c) Enter 180 into the Z: field under Absolute: World and hit enter.

The object will be rotated around the pivot point and will now appear on the opposite side of the level the same distance from the origin that the blue base object is.

The third image or last image further explains the above procedures and shows the final results of the base rotation.

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

Now that the red base has been rotated, it must be attached to the level and the holes in the mesh patched by adding polygons and matching vertices and edges to satisfy the Sealed World Rules. The base will then be detached from the level mesh into its own object again and will have some minor material adjustments made. Finally, both bases will then be attached to the reference frame. The following procedures will demonstrate the red base structure being attached to the level mesh, the level mesh and structure being modified to satisfy the Sealed World Rules, the faces that construct the red base being detached into a separate object, the modification of materials to distinguish the base structure, and the linking of both bases to the reference frame.

Completing the red base structure:

- 1) Select the level object.
- 2) Under Edit Geometry, enter Attach mode by clicking on Attach.
- 3) The Attach Options window will appear, select Do Not Modify Mat IDs or Material.

The base will temporarily have the “tutorial terrain\sky” material applied to it since this is the material for the “level” object. As a result, the textures or materials on the faces of the base will appear incorrect.

This is fine since the “Do Not Modify Mat IDs or Material” option was selected and this will keep the material assignments for the base correct. When the faces for the base are selected and detached to form the separate red base object again, the “base structures” Multi/Sub-Object will be applied and the base will have the correct materials again. No additional work will be needed.

The first image further explains the above procedures.

- 4) As demonstrated in the previous tutorial, remove the faces around the base.

The second image further explains the above procedure and shows the final results of the procedure.

- 5) Create polygons to attach the base structure to the terrain mesh. Once completed, fix the texture coordinates for the terrain in the same manner demonstrated when doing this same procedure for the blue base.

The third image further explains the above procedures as well as shows the end result of the above procedures.

- 6) Select all the faces that compose the base structure.

7) Under Edit Geometry, click on Detach. The Detach dialog window will appear.

8) In the Detach dialog window enter “red base” in the Detach as: field.

9) Click on OK.

The fourth image further explains the above procedures.

10) Select the red base object. Open the Material Editor, select the “base structures” Multi/Sub-Object.

11) Click on Assign Material to Selection.

12) Close the Material Editor.

The fifth image further explains the above procedures.

The base should now appear with the correct materials assigned and appearing on the faces of the structure.

13) To help distinguish this as the red base, the blue lights in the base hallway or tunnel will be turned into red lights by following the procedures below:

a) Select the faces that compose the hallway lights.

b) Apply Material ID 7 to these faces.

c) Adjust the texture coordinates (Unwrap UVW).

The sixth image further explains the above procedures as well as shows the final results for the above procedures.

14) Finally, attach the bases to the reference frame by doing the following:

a) Select both the red base and blue base objects.

b) Click on Select and Link.

c) Click on Select by Name. The Select Parent window will appear.

d) Select “frame” from the list.

e) Click on Link.

Both levels will now be a part of the level since they are linked to the frame.

Since the vertices and corresponding edges for each base matches the terrain mesh, the Sealed World Rules are satisfied.

The seventh and final image further explains the above procedures.

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

Player Clipping Techniques

It is sometimes necessary to create additional collision geometry in the level that prohibits the ability of the player to reach certain areas of the level or to help smooth the movement of the player around certain geometry. This technique is sometimes referred to as “player clipping”. Geometry that is not rendered, but only affects the player and vehicles is added to the existing geometry of the level. Restricting access to certain areas of the level such as the tops of canyon walls, roofs of structures, pillars, etc... can be crucial in creating a good game play experience. The player must not be allowed to exploit the game environment and break the game mechanics. A good example are the Capture the Flag (CTF) game modes in Halo, if a player has the flag and can take it to an unexpected area where it is easily defended or they cannot be reached, this breaks the game. Fortunately, Halo does have mechanisms to help when objectives such as the flag or oddball get into odd areas, these items will eventually respawn or reset after a specific amount of time.

Restricting the player is also often used to keep the player from seeing “behind the scenes” of the level by preventing the player from seeing past the edges or over the tops of the core geometry of the level, typically where the level geometry for the terrain mesh intersects with the faces used to render the sky model or sky box.

The following tutorials will discuss the uses of and demonstrate the creation of the player clipping material and the implementation of player containment or player clipping.

Creating the “player clipping” material.

The first step is to create a special material or shader that will not allow the player or vehicles to pass it but still allows objects and projectiles to pass.

1) As demonstrated before, a new Multi/Sub-Object will be created. Open the Material Editor and select an empty Multi/Sub-Object material group. This Material Group is currently set to a Standard Material.

2) Click on the Standard button to bring up the Material/Map Browser.

3) In the Material/Map Browser click on Multi/Sub-Object.

4) Click on OK.

The first image further explains the above procedures.

5) This Multi/Sub-Object is going to be called “player clip”. Type this name into the Multi/Sub-Object description window.

6) Just for organization, the extra Sub-Materials were deleted by selecting the Sub-Material to delete and clicking on Delete until the desired number of Sub-Materials remained, in this case, 1 Sub-Material.

7) The Sub-Material is now going to be named and linked to images. To name or label Sub-Material ID 1 follow the procedures below.

Click on the Sub-Material button to go from displaying the Multi/Sub-Object Basic Parameters window in the Material Editor to displaying the Shader Basic Parameters, Blinn Basic Parameters, Extended Parameters, [Super Sampling?](#), Maps, Dynamic Properties and Viewport Manager.

The second image further explains the above procedures.

8) Enter the name “example_tutorial_playerclip*” into the name or label field.

Notice the “*” Shader Symbol which denotes that this material will be treated by Tool and the game as collideable only, it will not render in the game.

9) To display a texture or image for this material when applied to a surface do the following:

a) Under Blinn Basic Parameters, click on the small box to the right of the Diffuse color setting. This will display the Material/Map Browser.

b) Click on Bitmap.

c) Click on OK and the Select Bitmap Image File window will now be displayed.

The third image further explains the above procedures.

10) To select an image to display for the material do the following:

a) In the Select Bitmap Image File go to the bitmaps directory under Halo\data\levels\test\tutorial_examples.

b) Under the bitmaps directory there should be several .tif image files. Select the example_tutorial_playerclip.tif file.

This image is used as a guide to see the player clipping surfaces in 3ds Max and will never appear in the game. Any image can be used.

As was mentioned previously, the resources under the tutorial_examples directory represent the end result of all the examples and tutorials contained within the Halo PC Editing Reference.

The Halo End User Editing Kit provides many resources including a wide range of tags as well as image resources (.tif files) that can be used in the creation of Halo multiplayer levels.

The files provided also exist as examples and learning aids for those wishing to create their own custom game resources whose creation and modification may not be covered currently in the Halo PC Editing Reference.

For now, the completed or provided example resources will be used for the following examples for the sake of simplicity.

c) Click on Open.

The fourth image further explains the above procedures.

To return to the previous options in the Material Editor window click on the Go to Parent button.

11) In the Material Editor, enable the display of the material by clicking on the Show Map in Viewport button.

12) The Sub-Material should be displayed on the Material Sample.

13) To return to the Multi/Sub-Object Basic Parameters click on the Go to Parent button.

14) Close the Material Editor window.

The fifth image or last image further explains the above procedures.

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

With the “player clipping” material created some simple player clipping geometry will be created. The geometry created to block the player must not have any open edges and must form a sealed volume. The player clipping geometry follows the Sealed World Rules. The player clipping geometry is going to be kept as a separate object so that it can be better organized with its special material and so that it can be hidden or handled more easily in 3ds Max.

The following procedures will demonstrate the implementation of player containment or player clipping.

Creating player clipping or player collision geometry.

To keep players from waiting or “camping” in the recessed portions of the teleporter gateways a “clip” plane is going to be created over the front faces of the teleporter gateways.

- 1) Select the blue base and hide the teleporter shield polygons or faces.
- 2) Create 2 faces that cover the front of the teleporter gate.
- 3) Under Edit Geometry click on Detach. The Detach dialog window appears.
- 4) Enter “player_clip” in the Detach as: field.
- 5) Click on OK.

The first image further explains the above procedures.

6) Apply the player clip material to the object and assign the Sub-Material to the faces.

a) Select the newly created player_clip object and apply the player clip Multi\Sub-Object material group.

b) Select the faces and clear any Smoothing Groups and apply Material ID 1 to the faces.

Perform Steps 1–6 on the red base to create the same player clipping plane for the teleporter gate on the red base.

Both player clipping pieces of geometry should be attached together to form a single “player_clip” object.

The second image further explains the above procedures.

The final set of player containment or player clipping geometry that will be created is a border around the edge of the surrounding hills.

This barrier will prevent the player from getting on top of the hills and seeing past the edge of the world where there is no geometry since only the sky box is drawn past this border.

7) Select the level object.

8) Select the 2 faces that form the top of the sky. Delete these faces.

The third image further explains the above procedures.

9) Select Edge mode.

10) Select all the edges that comprise the front edges of the surrounding hills.

The fourth image further explains the above procedures.

11) While still in Edge mode, extrude these edges up to create new faces by holding down Shift while moving up in the Z-Axis.

The fifth image further explains the above procedures.

12) Make the new vertices and edges even with the sky box vertices and edges. Follow the below procedures:

a) Select Vertex mode and select the top most vertices of the newly created polygons.

b) Click on the Top view. Under Edit Geometry click on View Align, this will make the vertices even in the XY plane.

c) Now move the vertices to be even with the vertices at the top of the sky box.

The sixth image further explains the above procedures and shows the final results of the above operations.

13) Remove any extra faces that may have resulted from extruding the edges. Once this is complete, create polygons (using the existing vertices) to seal the top of the world again so that it satisfies the Sealed World Rules. All of these faces should be made the +sky material by setting their Material ID to 1.

This technique may seem cumbersome or wasteful but for the type of level that the tutorial level represents it is necessary in order to get the clipping planes to work properly, the level to export and compile properly, and to satisfy the Sealed World Rules.

Remember that the number of faces used to construct the sky does not matter since the sky model gets rendered to these faces and the triangle count from the sky model is what will count in this instance.

There are many ways to construct a game environment and its surrounding terrain and sky box surfaces. It is quite common for all the sky box faces to have their top vertices pulled to a single vertex in the center of the level at a very high distance in the Z-Axis. Sometimes, the sky faces are not set back as they are in the tutorial level, instead the sky box faces are used as the boundaries for the level and serve to keep the player from accessing any areas where the player's presence is not desired.

It is recommended that for outdoor levels that have vehicles that there be quite a bit of space left in the Z-Axis. This allows weapons that have a long range and that can fire in an arc (such as the Fuel Rod Gun) to have room to fire without colliding with the ceiling or sky. It is also nice to allow the Halo physics to have room to work so that when vehicles get hit with explosions they have room to do flips and other spectacular vehicular acrobatics.

The seventh image further explains the above procedures and shows the final results of the above operations.

14) Hide the newly created top faces of the level. Now, select all the faces that will compose the player clipping surfaces.

15) Under Edit Geometry, click on Detach. The Detach dialog window will appear.

16) Enter the name "player_clip_01" in the Detach as: field.

17) Click on OK.

The eighth image further explains the above procedures and shows the final results of the above operations.

18) Apply the player clip material to the object and assign the Sub-Material to the faces.

a) Select the newly created player_clip_01 object and apply the player clip Multi\Sub-Object material group.

b) Select the faces and clear any Smoothing Groups and apply Material ID 1 to the faces.

In the example picture the faces also had a simple box map projection applied to them (UVW Map), but this is not necessary, it just makes the faces easier to work with visually.

The player_clip object was then attached to the player_clip_01 object to keep all the player_clip surfaces under one object.

The player_clip object was then linked to the reference frame (frame object).

The ninth image or final image further explains the above procedures and shows the final results of the above operations.

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

Placement of Portals

The final step to completing the core geometry for the level is the addition of surfaces or planes that define portals. These portals are used by the engine for several purposes. The primary function of these portals are as “visibility portals”, portals that the Halo game engine uses to try and cull out objects that cannot be seen to improve performance by not rendering such objects.

The portals in general help break up the level so that it can be handled in parts to help performance in general. Attributes can also be set for portals, such as ambient sounds to play in the portal as well as environmental audio effects. Portals can also be used to define weather volumes and fog volumes.

There are 2 kinds of portals that are defined, “portals” and “exact portals”. The following are descriptions for

each:

1) Exact Portals: These are portals defined by surfaces or planes with the +exactportal material applied to them. Exact portals are created such that the faces (edges and vertices) match the surrounding geometry (edges and vertices). These faces in effect create a “seal”. The volume that is between these surfaces or planes is an exact portal. The direction of the normals of the faces helps determine how the portal is defined.

2) Portals: These are portals defined by surfaces or planes with the +portal material applied to them. These portals are created by faces or planes that are used to divide the game environment. These planes must create a seal with the level, but do not have to match vertices and the associated edges with the surrounding geometry. These planes are much more loosely defined and placed in the level and are used to break or divide the level up into sections. The direction of the normals for the faces helps determine how the portal is defined.

A volume defined as an Exact portal CAN have standard Portal areas defined within it and vice versa as long as the above rules outlined in their definitions are satisfied. However, an open ended or non-enclosed Portal cannot intersect an Exact portal, a “vis error” or rendering anomaly will occur. In other words, be careful to not let a Portal plane improperly divide or bisect an Exact portal volume.

Portals can be used to help portal horizontally (looking left and right in the XY plane) AND vertically (looking up and over objects or geometry that vary in the Z-Axis).

The tutorials below will demonstrate the creation and placement of planes that will define Portal volumes and Exact portal volumes.

The creation of the portal Multi/Sub-Object material group:

The first step is to create the portal materials.

1) As demonstrated before, a new Multi/Sub-Object is created. Open the Material Editor and select an empty Multi/Sub-Object material group. This Material Group is currently set to a Standard Material.

2) Click on the Standard button to bring up the Material/Map Browser.

3) In the Material/Map Browser click on Multi/Sub-Object.

4) Click on OK.

The first image further explains the above procedures.

5) This Multi/Sub-Object is going to be called “portal materials”. Type this name into the Multi/Sub-Object description window.

6) Just for organization, the extra Sub-Materials were deleted by selecting the Sub-Material to delete and

clicking on Delete until the desired number of Sub-Materials remained, in this case, 2 Sub-Materials.

7) Enter the names “+portal” into the name or label field for Sub-Material 1 and “+exactportal” into the name or label field for Sub-Material 2.

These are Special Material names, more information on these materials as well as all Special Materials can be found in the Special Materials section in the Materials Overview reference section.

The Diffuse colors for these materials were set and for each Sub-Material the Show Map in Viewport option was toggled on.

The second image or last image further explains the above procedures.

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

The procedures below will demonstrate the creation and placement of planes that will define Portal volumes and Exact portal volumes.

The creation and placement of portals:

The hallways for the 2 base structures (blue base and red base) will be made Exact portal volumes.

Later, when the level is running in the game, use rasterizer_wireframe 1 to view the level and exit and enter the hallways to see how the level geometry and objects are not drawn from the player view as this portal volume is entered and exited.

1) Select the blue base. In Face mode, create 2 polygons that exactly fit the doorways of the hallways.

2) Make sure the normals for the faces point outwards, this will help define the volume between the faces. the normals can be viewed by clicking on the Show Normals check box under Selection. If the normals are not easily visible, the lengths of the visual representations of the normals can be increased using the Scale value.

3) Detach the faces for the Exact portals by clicking on Detach under Edit Geometry. The Detach dialog window will appear.

4) In the Detach as: field enter the name “base portals”.

5) Click on OK.

Repeat the above procedures 1–5 for the red base and combine the 8 faces that compose the portals into 1

object called “base portals”.

The first image further explains the above procedures.

6) Now select the “base portals” object and apply the “portal materials” Multi/Sub-Object.

Once this is complete, assign the +exactportal material to the faces by selecting all the faces and setting the Material ID to 2 for the faces.

Once this is complete, the “base portals” object should be linked to the “frame”.

Note that the portal materials are at 80% opacity, this is just a setting in the Sub-Material and has no effect in game, it is just used as a visual aid in 3ds Max.

The two hallways have now been defined as Exact Portal volumes.

The second image further explains the above procedures and shows the final results of the listed operations.

The level will now be divided using standard Portals.

7) Deselect all objects. Under Create click on Plane.

8) Enter the following values in the listed fields, Length Segs: 1 and Width Segs: 5

9) In the Right view, create the plane such that the edges extreme edges extend past the extents of the level.

Take this plane and create 3 more planes for a total of 4 planes that run lengthwise “North” to “South”.

The third image further explains the above procedures and shows the final results of the listed operations.

10) Convert all the planes into an Editable Mesh (Edit Mesh) and Attach all the planes into 1 object called “terrain portals”.

11) The planes and faces were all aligned to try and divide the level evenly. Vertices and the associated edges and faces were aligned along major edge boundaries in the level.

The fourth image further explains the above procedures and shows the final results the above procedures.

12) To further subdivide the level into more portals, faces running “West” to “East” were created using the existing vertices and edges for the planes. Only the middle 4 edges and associated vertices were used. In addition, the edges were extended past the level boundaries.

Make sure that the normals are consistent for all the faces, this will determine how the portal volumes are created.

In the example, the faces running “East” to “West” have their normals pointing North, while the faces running “North” to “South” have their normals facing “East”.

13) Select all the faces and Clear All the Smoothing Groups and apply Material ID 1 to all the faces to assign the +portal material.

Once the above procedures are completed, link the “terrain portals” object to the reference frame “frame” object.

The level has now been portalled.

The fifth image or last image further explains the above procedures and shows the final results the above procedures.

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

[Click to Open Larger Image in New Window](#)

Saving the Level

At this point, the level should be saved. To save the level do the following: 1) Go to File.

2) Click on Save As.

3) In the Save File As dialog window go to the Halo\data\levels\test\tutorial\models directory (this directory was created in the above section Creating a Level Directory) and save the file as “tutorial.max” by entering this name in the File name: field.

4) Click on Save.

The level is now saved. This level will be used in the tutorial and example areas of the next sections. It may also be a good idea to make a backup of this file.

Conclusion of Level Creation - Part 2

The level geometry is now “complete” and can be successfully exported and compiled and is ready to go through the steps that will get it running in Halo. As previously mentioned, a completed version or example version of the level created in the tutorial sections can be found under the “Halo\data\levels\test\tutorial_examples” directory. The source and related materials have been provided as a reference to aid in the learning process.

Once the user has successfully completed the tutorials in Level Creation - Part 1 and Level Creation - Part 2 they can proceed to the next section Level Exporting.

Please note that the section Level Exporting and the subsequent sections will assume that the end user has completed all of the examples and tutorials up to and including those in Level Creation - Part 2. The remaining sections in Multiplayer Level Design will use the completed “tutorial.max” file from these sections.

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

- view
- edit
- attach
- print
- history

Super Sunny's Tutorial on how to add custom sounds

Okay, first of all, download [Gold Wave](#) (do a google on it). It's an AWESOME program that allows so many different types of exports and sound editing features. You need this for easier work.

Now, Go find any sound or music you like that is not a midi file. Copy this to your C:/Program Files/Microsoft Games/Halo Custom Edition/data/sound/test/test folder (C:/ is representing your drive, and you must create the test/test folders)...

Now, open the copied file in [Gold Wave](#). Go to file, save as, change the file type to a wav, and change the file type for the wav into a PCM 16 bit stereo file (you can do Mono, but it usually sounds like crap). Save it. Then, in tool.exe, type in:

(tool.exe sounds sound\test\test platform wav)

tool.exe sounds sound\test\test wav 0 ←-- Won't work within Halo of some reason

tool.exe sounds sound\test\test xbox 1

It should say importing "sound name". If not, you messed up somewhere along the directions.

Then, you can take create new .sound_looping files in Geurilla or change existing ones to play your custom sound. In order to confirm your song converted correctly, open Geurilla, go to file, open, find your converted .sound file, open it, scroll all the way done, and select any of the three buttons that say play.

I've already been able to add some of my songs, the Forest Temple Music from Zelda, and more all into Halo. This can be used for custom sound effects also.

How To Make Your Custom Sounds Play When You Fire Your New Gun by: [Cpt Lag](#)

Ok guys, I've been asked numerous times by friends and halo fans alike about how to add custom sounds to your gun. So I'll take you through how to make your gun use the new sound. First of all, you need to make your .sound tag. In order to do this (I'm sure you all have, but just to reiterate {sp?}) check out [Super Sunny's](#) tut above this little update. Now that you've made your .sound tag it's time to begin to add it to your .weapon tag. Now I'm going to assume you've edited an existing weapon tag and resaved it as your own. Open the .weapon of your gun (lets call it the Magnum for easy typing purposes). Scroll down to the very bottom of the

Magnum tag and you should see a block named “Firing Effects”. It’ll look like the picture below:

<http://cptlag.haloimpulse.com/customsnd1.JPG>

Now that your at the Firing Effect block it’s time to create your Magnum firing effect. Choose the open button on the right of the “...” button in the firing effect line (see above picture if you do not have the tag in front of you). Once you have opened the firing effect tag choose File > Save As and browse to the Magnum directory (normally tags\weapons\magnum). Once in there, create a new folder named “effects” so you now have “tags\weapons\magnum\effects”. Save your Firing Effect into there (I suggest keeping the name so you won’t confuse yourself). Alright, now you have a Magnum firing effect tag. In the new tag, scoll down to either the 3rd or 4th block titled “Parts”. It should say “Firing” in the dropdown menu. Now where it says “type” make sure you have “Sound” selected in the dropdown menu. Then choose the “...” button and point to your custom sound (indicated below).

<http://cptlag.haloimpulse.com/customsnd2.JPG>

Now return to your .weapon tag and change where the firing tag points to point to the one you just created. Save all tags and close. Then build ur map cache file and your all set!

EDIT: sry guys but you hafta copy/paste the pix into your browser since idk the Wiki password to allow me to add them.

EDIT: The URL is approved now, np 😊 /Z

Note: You need the Xbox Audio Codec to get this to run. Get it here:

http://wiki.fpsgamers.se/uploads/Halo/xb_adpcm_codec.rar

~[Super Sunny](#) and [Cpt Lag](#)

weapons\h2_magnum\h2_magnum.weapon

projectile

weapons\pistol\bullet

...

Open

misc

ejection port recovery time

0

illumination recovery time

0.1

heat generated per round

0

[0,1]

age generated per round

0

[0,1]

overload time

0

seconds

FIRING EFFECTS

fire bullet

▼

Add

Insert

Duplicate

Delete

Delete All

shot count lower bound

1

shot count upper bound

1

firing effect

effect

▼

weapons\pistol\effects\fire

...

Open

misfire effect

▼

...

Open

empty effect

effect

▼

weapons\pistol\effects\em

...

Open

firing damage

weapons\pistol\trigger

...

Open

misfire damage

...

Open

empty damage

...

Open

weapons\pistol\effects\fire bullet.effect

EVENTS

0. effect_event_block

Add

Insert

Duplicate

Delete

Delete All

skip fraction

0

delay bounds

0

to

0

seconds

duration bounds

0

to

0

seconds

PARTS

fire

Add

Insert

Duplicate

Delete

Delete All

create in

any environment

create in

either mode

location

primary trigger

flags

☐ face down regardless of location

type

sound

sound\sfx\weapons\pistol\

...

Open

velocity bounds

0

to

0

world units per second

velocity cone angle

0

degrees

angular velocity bounds

0

to

0

degrees per second

radius modifier bounds

0

to

0

SCALE MODIFIERS

A scales values

☐ velocity

☐ velocity delta

☐ velocity cone angle

- view
- edit
- attach
- print
- history

:P PAGE UNDER CONSTRUCTION :P

REQUIREMENTS: 3D Max, Photoshop, HEK

ACKNOWLEDGEMENTS:

I originally learned how to edit certain parameters of existing multiplayer skies using Mattyp001's [Custom Sky Tutorial](#) available from [Halomods](#). Using this knowledge I modded an existing Halo CE sky to include custom bitmaps - a full tutorial on this can be found here: [Moonbase Custom Sky](#). This method is inflexible as it requires that you use an existing sky Gearbox Model, so I set about trying to create a gbxmodel from scratch. I got the general idea down, but it suffered some parallax errors. This is documented here: [Nearly Figured Out Custom Skies](#). Finally, Molkien to the rescue, he releases his Cliffhanger sky as an open source download, allowing me to see where I was going wrong - simply that my sky model was too small: [Cliffhanger Open Source Thread](#)

So, basically, acknowledgements to **Mattyp001** and **Molkien** for their part in helping me figure out this tutorial!!

DESCRIPTION:

How to create the Gearbox Models for custom skies. If you dont know what I'm talking about, read my [Custom Sky Tutorial](#) first and you will understand the in's and out's of a Halo CE Sky model. Once you've created your sky Gbx model, you populate it with bitmaps as in the tutorial above.

STEP BY STEP INSTRUCTIONS:

Page will be up soon, watch this space!!

|S_s| Hury up Plz! ...Is anyone even working on it?

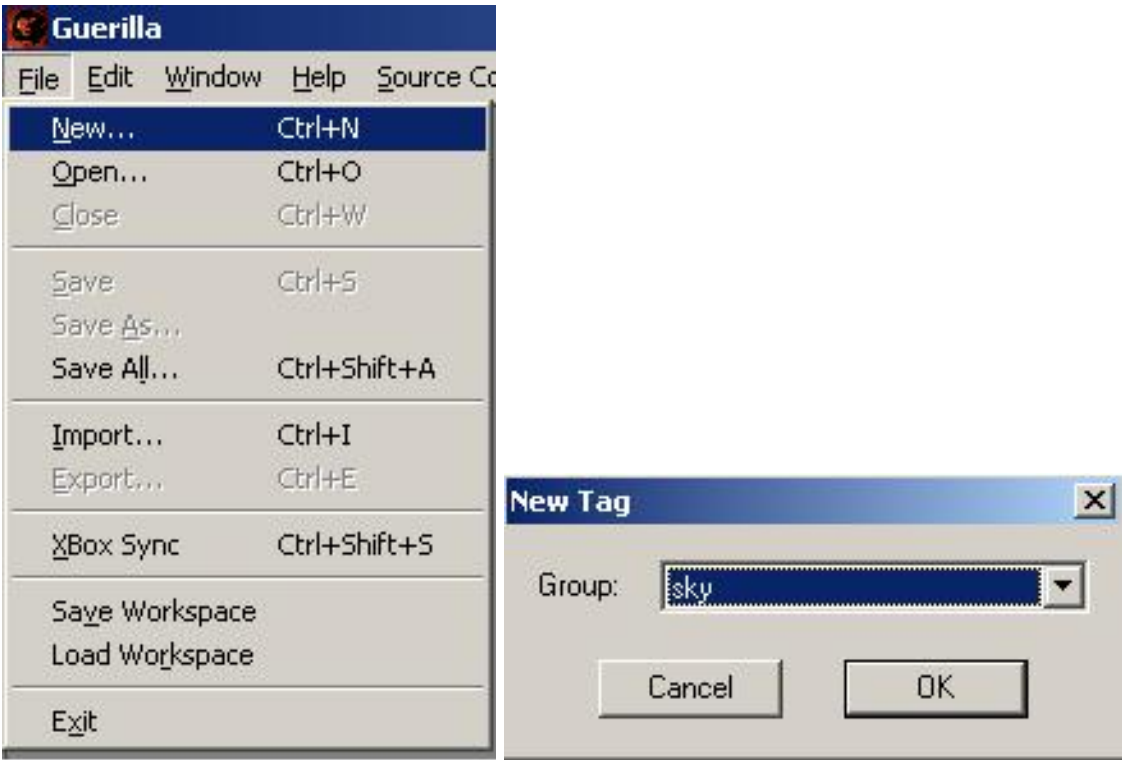
yes plz if your working on this hurry other wise im just gonna wing it and try to make one..

ok er.. I tried and i failed.. its actualy kinda simple to make a basic sky but adding more then one object gets wierd.. stuff appears in wrong order and i cant figure out why u have to give the gbx model regions some how per each object and order them correctly

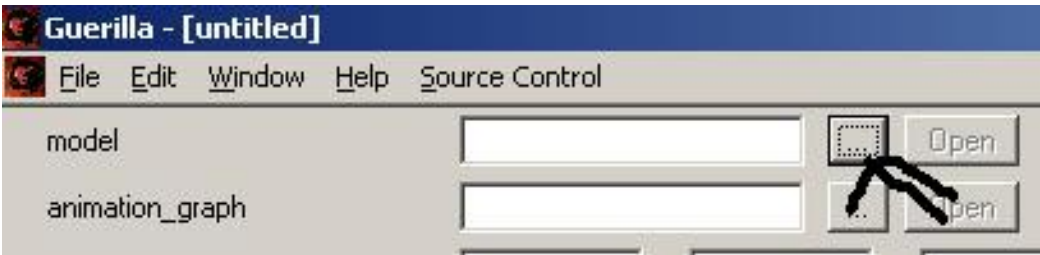
- view
- edit
- attach
- print
- history

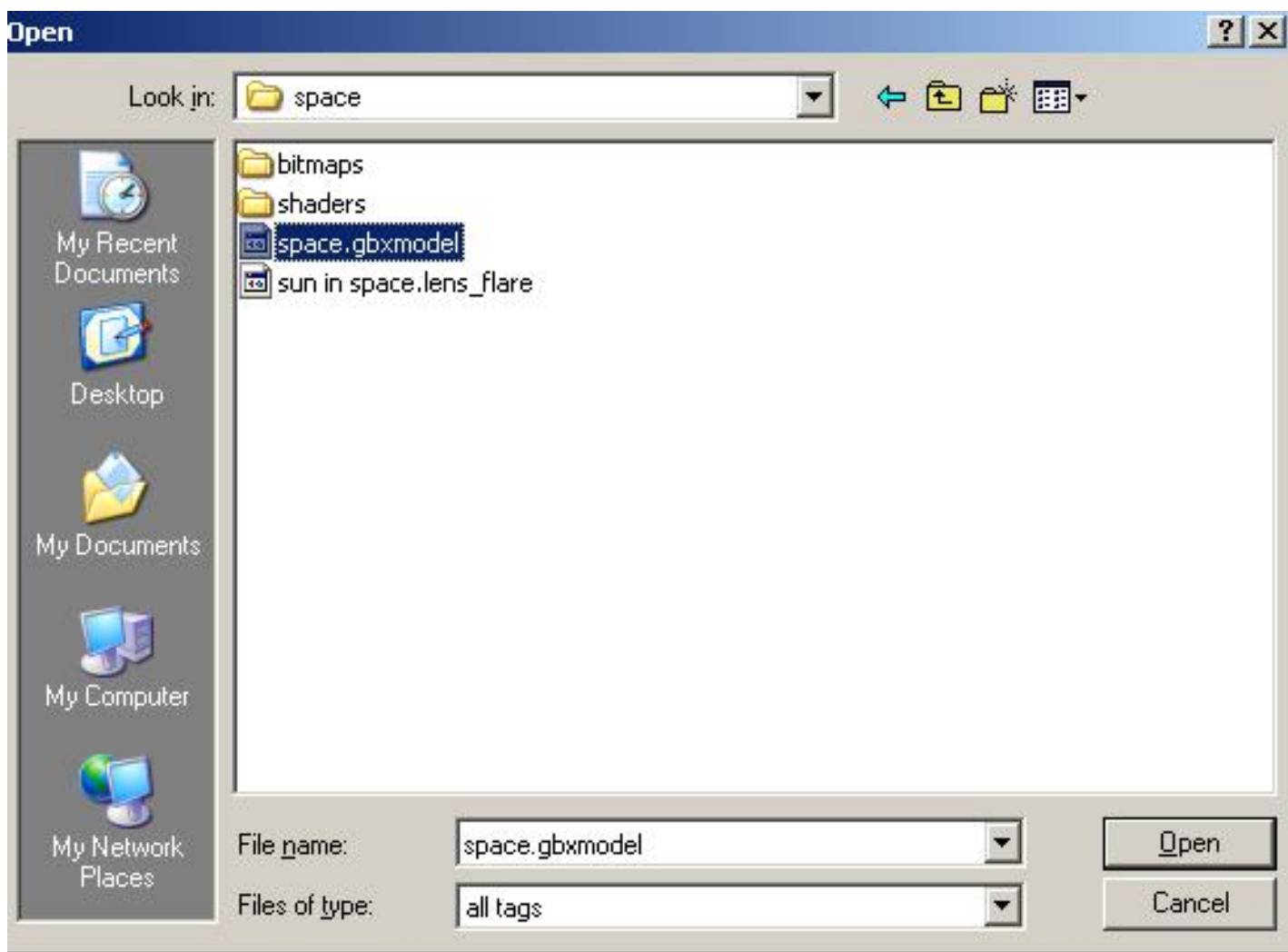
Tutorial on making new skys

1. open up guerilla. 2. go to file>new> and select sky from the list.

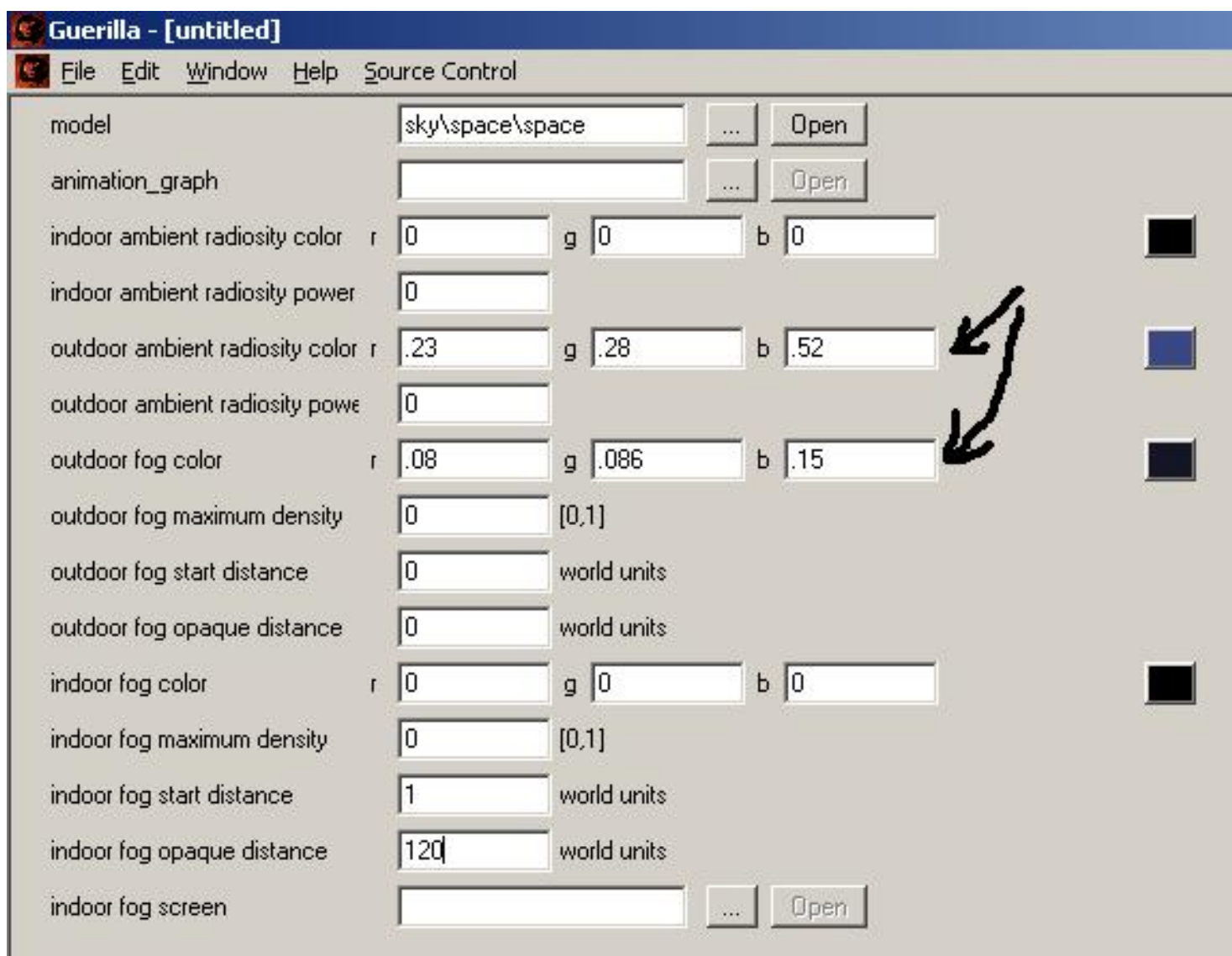


3. now click on the “...” next to model and open up the tags\sky\space\space.gbxmodel



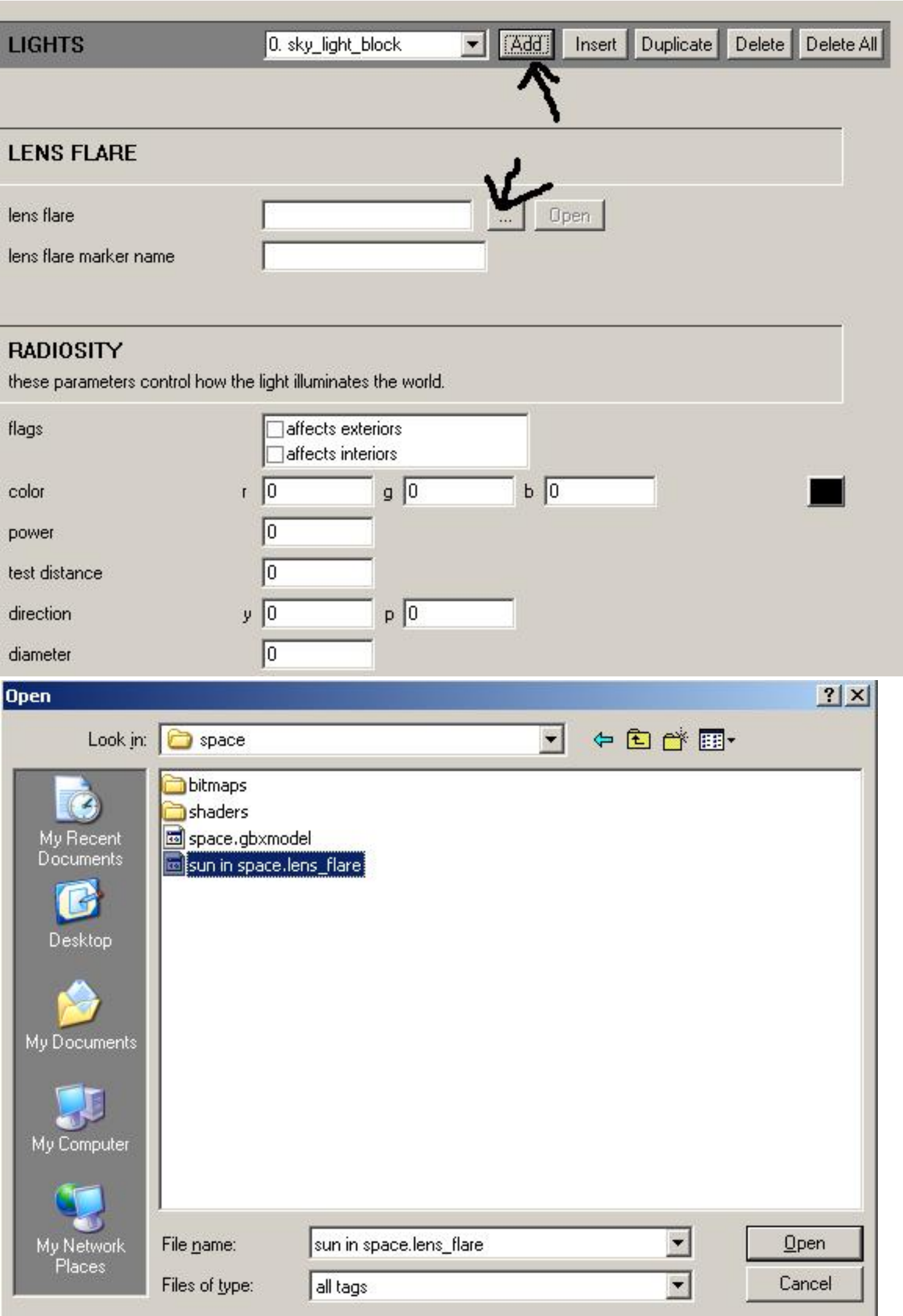


4. now give the outdoor abient radiosity color and outdoor fog color similar values to the ones i have. or just click the color box next to the values on the right and choose your own colors.

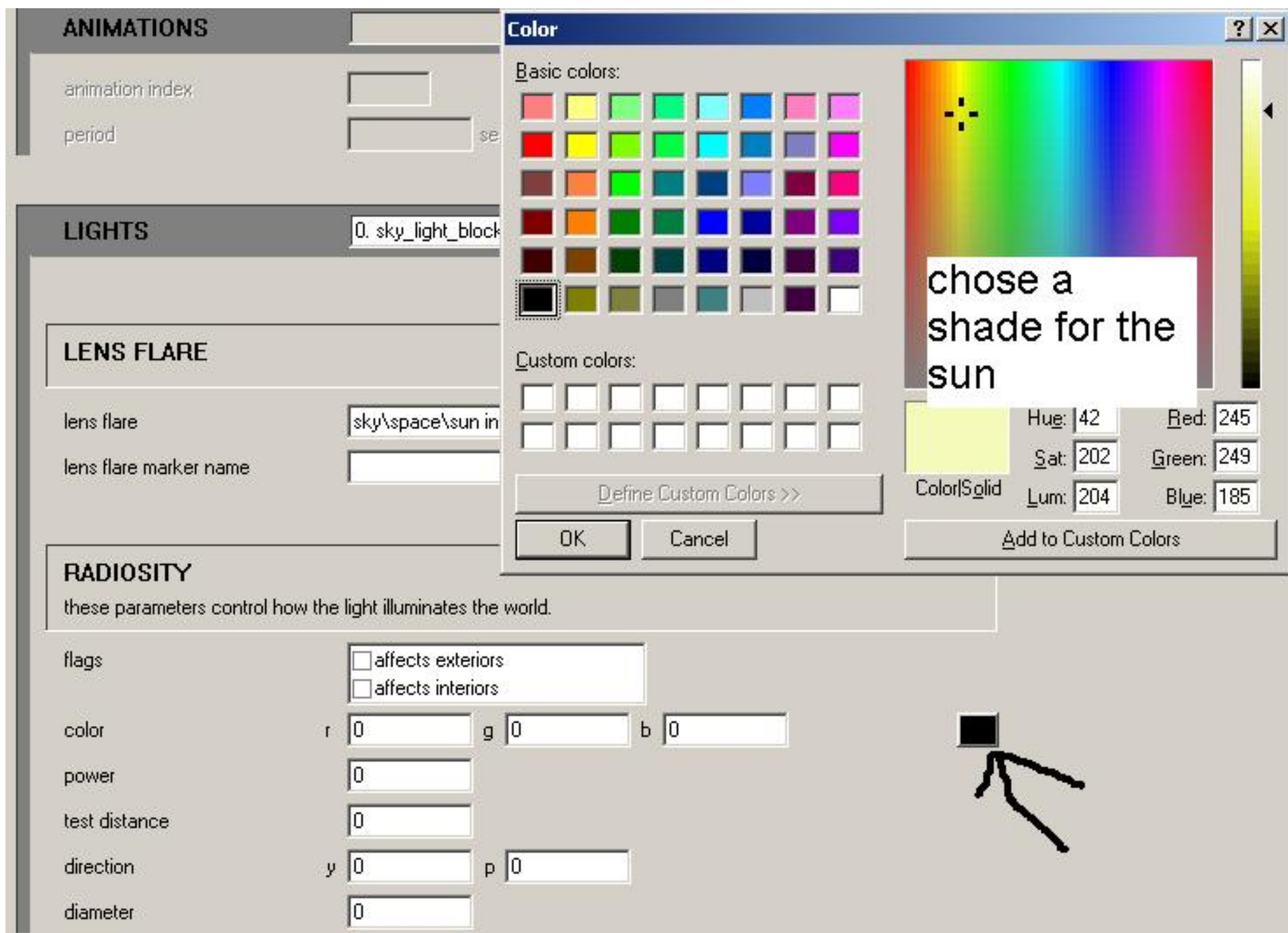


5. now click on the add button for the lights section followed by choosing the sunflare effect in the

tags\sky\space\sun in space.lens_flare



6. now choose the color and set it to a very light yellow like so



7. check the affects exterior button, give it a power of about 1.7, set the test distance to 50, set directions y and p to * -30 and 25* to get the sun over the galaxy, and give it a diameter of about .2. all of this is indicated in the following picture

LIGHTS

0. sky_light_block

Add

Insert

Duplicate

Delete

Delete All

LENS FLARE

lens flare

sky\space\sun in space

...

Open

lens flare marker name

RADIOSITY

these parameters control how the light illuminates the world.

flags

☒ affects exteriors

☐ affects interiors

color

r

0.909804

g

0.917647

b

0.592157

power

1.9

test distance

50

direction

y

-30

p

25

diameter

0.2

8. save the sky

9. now edit your scenario to use this sky and now you have a new sky!

Author: mattp001

Add Strings with Tool

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Strings

Author: [Steelix B](#)

String Lists. No need to hex edit things, heres exactly how make them:

1. Open up [Word Pad](#)
2. Type your first entry for your list
3. Either on the same line as that entry, or on the next line type ‘###END-STRING###’
4. Repeat the last 2 steps for your the rest of the entries for the list
5. You should have something like this:

```
You picked up a flamethrower
###END-STRING###
You picked up %
###END-STRING###
```

6. Save the file as either a unicode or an ansi txt file, save it in a folder in your data folder in the main halo ce directory
7. Depending on the format you saved it as, run one of the following commands in tool:

```
Tool Strings <directory of where you saved your list>
Tool Unicode-Strings <directory>
```

You should now have your string list tags created in the tags folder in your main halo ce directory.

Note that the list must also end on a new line, so that tool doesnt miss off the last string, and also don’t place everything on the same line, otherwise instead of tool removing the new line characters it will remove the first 2 characters of every string after the first. Hope this helps! (And i discovered it myself)

-[Steelix B](#)

To make strings for weapons, read [Here](#)

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Basics of animations

Author: [Ki L La](#)

Edited for grammar and spelling by: An Emu

Ok here some basic guidelines for animating things for halo...

1. All geometry must be linked to a node.(Frame or biped bone)
2. You do not animate the object itself, You simply animate a node which the object is linked to.
3. Keyframe amounts do not matter in base animations aka .jma. In all other animation types, the number of keyframes does matter.
4. Try making all animations end where they started (makes it look smoother ingame)
5. Some notes when exporting:

.jma = base animation

.jmr = replacement animation

.jmo = overlay animations

1. if you are updating an existing animation graph, the file extentions must stay the same or it won't import the animation.
2. When animating an object, the actual object is not needed(but can be used for reference) You ONLY NEED to animate the nodes of the model.

That is all I can think up for now...

Hope this gets you on your way to animating!

- view
- edit
- attach
- print
- history

Custom Animations 3P (Third-Person) View

Author: [Ki L La](#)

You will need Character Studio to do this... OR 3DS Max 7, which has it built in.
Before we get started, I suggest you read the [Basics of Animations](#)

Building Master Chief:

- Open 3DS Max and under the create tab you should see systems, click that, then click on biped.
- Click on the radio button (circle button) that reads “Drag Position”.
- Keep structure source at U/I.
- I don’t think “Body Type” matters but I used skeleton.
- Make sure the arms check box is checked.
- Make sure he has one Neck Link, 2 Spine links 3 Leg links, 0 tail links,0 ponytail1 links,0 ponytail2 links, 0 fingers, 1 finger link, 1 toe and 1 toe link.
- Keep ankle attach at 0.2.
- Make sure it’s hieght is 70(MC is 70 units).
- I used triangle pelvis but I am not sure if this matters.
- Make sure forearm twist is unchecked.
- Default of 2 forearm links at the bottom (don’t think these values are used with out forearm twist).
- Create the biped structure by dragging to his position.
- Make sure his feet are above the grid(press the G key to see it or make it disappear).
- If his feet are not above the grid, his feet ingame will be below the ground.
- Click on bip01(the root object) Select the rotation tool, type in the number “90”.

Now here is the strangest part:

Since Master Chief doesn’t have any toes, and 3DS Max will NOT let you delete the Biped structure toes without deleting the entire Biped you need to take these steps:

- First select both toes and click unlink.
- Then right click and convert to Editable Mesh TWICE. If you do not do this, when you export Blitzkrieg will tell you there are too many actors in this scene.

Now you have finished making a rough structure of Master Chief’s bones.
You may have to scale his shoulders a bit for the right look in-game.

To do this, simply select the Biped and click the motion tab.

Under the heading Biped, click figure mode. Now you can scale his bones and adjust them.

Remember that if you do not use figure mode you cannot scale.

As for animating DO NOT use the already done animations (.bip files) and you cannot use the footsteps. You must set each keyframe by keyframe by rotating or moving.

ANIMATE AWAY!

Note: BACK UP YOUR cyborg.model_animations file BEFORE CONTINUING!

Exporting:

After you are pleased with your animation, you will need to make this folder structure:

data\characters\cyborg\animations

After you've done that go back into 3DS Max and hit File > Export. Make sure the file type is JMA NOT JMS. When saving your JMA file make sure you make it a simple name you can understand easily because you will see these names in Guerilla.

Example: pistol-melee.jma

Export your JMA file to the characters\cyborg\animations folder you just made.

Now open up Tool and type:

```
tool animations characters\cyborg
```

Tool should read something to this effect(plus more deleted animations):

```
WARNING: the animation 'G-driver acc-left-right' is no longer used
```

```
WARNING: the animation 'G-driver acc-front-back' is no longer used
```

```
WARNING: the animation 'CT-driver unarmed idle' is no longer used
```

```
WARNING: the animation 'CT-driver exit' is no longer used
```

```
WARNING: the animation 'B-driver unarmed idle' is no longer used
```

```
WARNING: the animation 'B-driver exit' is no longer used
```

```
WARNING: the animation 'B-driver enter' is no longer used
```

```
some animations were deleted from the graph; do you want to save it anyway? (y/n)
```

When you type Y it should say:

```
model animation compression saved 0 bytes
```

Disclaimer:

Get Kornman's Anim injector if u want to add new animations [http://fuzzy-](http://fuzzy-entertainment.net/seancooper/downloads/HaloAnimationEditor.rar)

[entertainment.net/seancooper/downloads/HaloAnimationEditor.rar](http://fuzzy-entertainment.net/seancooper/downloads/HaloAnimationEditor.rar)([approve sites](#)) For a tutorial on using the

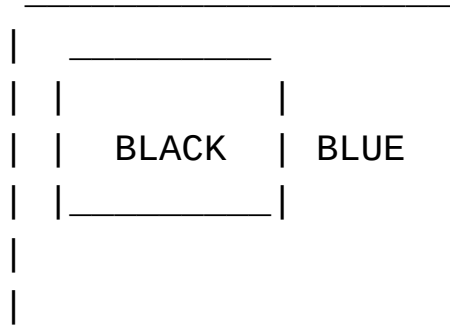
anim injector refer to - <http://www.halo2ce.epgservers.com/tutorial/?index=8>([approve sites](#))

- view
- edit
- attach
- print
- history

Creating Sprites

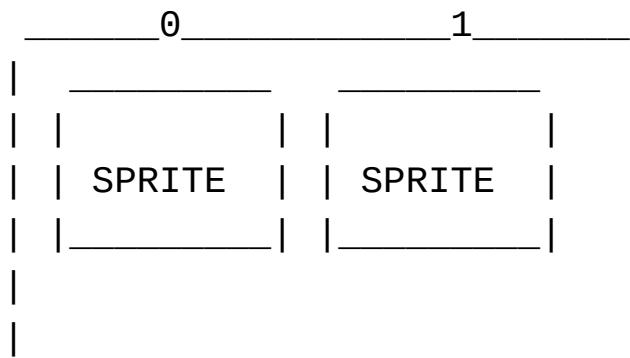
Load up your favourite image editing program.

1. Create a bitmap any size you want (unless it's a replacement bitmap, in which case you need to look at the bitmap tag's page width and height.)
2. Fill the image with the colour blue (RGB 0, 0, 255)
3. Create a black rectangle in the top left corner of the image, and make sure it is at least 1 pixel from the top left of the image. The image should look something like this:



You have now created a single sprite, and a single sequence.

1. To add more sprites to this sequence, add more sprites to the right of this first sprite, for example:



1. To add a new sequence, start a new line of sprites:



			_____			_____	
			_____			_____	
1			SPRITE			SPRITE	
			_____			_____	

(by the way, the numbers on the left represent the sequence index, whilst the numbers on the top represent the sprite index. Ignore these, they are not part of your final image)

Save the image as .tif in a folder under “Halo Custom Edition\Data”

Compiling

There’s nothing to it really, simply compile this bitmap like any other bitmap.

Then open in guerilla, change the type to sprite, and possibly some other values to match up to the type of sprite you are making (for example ui\hud\bitmaps\msg_hud_icons.bitmap - correct me if that filename is wrong).

That’s it, you have a sprite bitmap.

- view
- edit
- attach
- print
- history

How to add wind to a level

This tutorial will explaine how to add wind to a level using sapien.

First of all you must understand how to use all the windows in sapien, (The Game Window, The Hierarchy View, The Tool Window, The Properties Palette) Each of these windows has its use and I will explaine it in this tutorial.

Step 1. Open sapien and navigate to a Scenario file that you want to put wind it. Let it open, it will take a minute. Once it is open arrange your windows how you like them just make sure you can get to all of them.

Step 2. Once sapien opens and you have arraged your windows follow these steps. First go to your Hierarchy view and open the **Mission** folder, then open the **Cluster Properties** folder. Once you have those folders open open the **Palettes** folder under that. Click the **Weather Palette** folder.

Step 3. Once you have navigated down to your **Weather Palette** folder click your Properties Palette to select it. Next to the top box it should say (name). Click that box and name it “wind”. Then scroll down to the bottom of the Properties Palette and near the bottom there should be a box that has a gray background. It should say “wind” click the 3 dots on the right side and navigate to your halo ce root folder and then to levels\test\bloodgulch. Open the multiplayer.wind tag. Then under the wind direction it should have 3 boxes. The first one should have a “i” beside it, the 2nd one should have a “j” next to it also and the 3rd box should have a “k” next to it. In the first box type 3 instead of 0 and leave the other 2 boxes with a 0 in them.

Step 4. Next click the box under the 3 boxes that says “wind Magnitude”. Change the 0 to a 2. 2 is the wind magnitude for all the other halo maps. Leave the “wind scale function box like it is. Once you have a 2 in the “wind magnitude box go up to the Hierarchy View window and click “New Instance” a new instance should appear in the right side of the Hierarchy View that is labled “wind”.

Step 5. Now close the “paletts” folder after adding the wind instance. Click the “Weather” folder under the “cluster properties” folder. Then click the Properties Palette and drop down the box and select “wind” from the list. Then go into your tool window and drop down that drop down box and select “wind” again. After selecting “wind” from the list press space to apply the cluster.

Now you have added wind to a level so that during CTF your flags dont look like poles.



AI Bots in Multiplayer Maps

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Aztec and SRG came up with this really cool video. It explains how to set up encounters/squads/ai in your maps.

Check it out:

(12.5MB, 3.5 minutes)

[Download The Tutorial Video](#)

[Original Thread on Halomods](#)

- view
- edit
- attach
- print
- history

Net Game Flags Placement

ACKNOWLEDGEMENTS: Originally Posted by Xzzy on [Gearbox Software Forums](#). Click [HERE](#) to view the original thread.

DESCRIPTION:

The objects that determine game rules and set objective points in the level are called Netgame Flags. These Netgame Flags are used to define the spot where a flag for CTF is located at, where the Oddballs in Oddball game modes spawn, the Race Points for Race Mode, the Race Vehicle spawn points, and the region or area for the control zone for King of the Hill. In addition, Netgame Flags are used to define the entry and exit points for teleporters.

For more information on the rules for Netgame Flags see the Multiplayer Level Design Technical Rules and Guidelines section under the General Level Design Information section located in the General Reference Sections.

The following section contains tutorials and examples that will demonstrate the placement of the Netgame Flags that are necessary for the proper operation of the level in a multiplayer game.

STEP BY STEP INSTRUCTIONS:

Netgame Flags Placement

Placement of the netgame flags necessary for the Capture the Flag (CTF) game mode in the tutorial level:

- 1) Go to the Hierarchy view window. In the Hierarchy tree expand the “Game data” branch. Next, select the “Netgame flags” branch/
- 2) The netgame flags that will be added are ctf - flag objects.

In the Properties palette make sure the the “type” is set to “ctf - flag”.

In the Game Window, press the Right Mouse Button to place a netgame flag object in the level on top of each base.

Align the netgame flag so that it matches the position of the previously placed “flag_stand.scenery” object.

3) In the Properties palette set the “team index” for each netgame flag. This will determine the team setting for the flag.

The “team index” follows the “team index” conventions covered previously.

The “ctf - flag” netgame flag for the Blue Base should have its “team index” set to “1”.

The “ctf - flag” netgame flag for the Red Base should have its “team index” set to “0”.

The ctf - flag netgame flag represents the location where the flag will appear in the game as well as the area where the opposing teams flag must be taken to in order to score.

Adding Weather Attributes using Net Game Flags

The following section contains tutorials and examples that will demonstrate the application of “Weather” settings to a portal or cluster.

Setting “Weather” attributes in the .scenario_structure_bsp portals or clusters.

For the tutorial level, the weather effect that will be added is wind. Wind affects certain objects, such as the contrail on the sniper rifle bullet as it travels and the waving of the flag used in Capture the Flag.

1) Go to the Hierarchy view window. In the Hierarchy tree expand the “Cluster properties” branch. Next, expand the “Palettes” branch and select the “weather palette” subtree object.

2) Currently, the “weather palette” is empty. To create a “weather palette” follow the procedures below.

a) Click on New Instance button in the Hierarchy view window. A new instance of a weather palette will appear in the right pane of the Hierarchy view window.

b) Select this new instance.

c) This weather palette instance must have a weather (.wind) tag set for it.

Go to the Properties palette and click on the “...” button located in the “wind” section. An Open dialog window will appear.

Most multiplayer levels have their own multiplayer.wind tag file that exists in the root directory for the specific level.

For the tutorial, the multiplayer.wind file that will be used is an example .wind file that is included in the “tutorial_examples” level directory.

3) Go to the tutorial_examples level directory located under the “levels” tags directory.

4) Select the “multiplayer.wind” tag.

5) Click on the Open button.

This tag reference should appear in the “wind” section in the Properties palette window.

6) This weather reference must now be named.

In the “name” field in the Properties palette window enter the name “forest wind”.

The name should update in the right pane of the Hierarchy view window.

The name does not matter, it is just used for reference. The weather can be changed underneath this name. This enables the generic “forest wind” weather instance to be applied to the portals or clusters in the level and then the actual wind tag reference can be changed if needed to adjust or select the appropriate wind and weather effects.

7) The wind direction and magnitude must now be set. Follow the procedures below to enter these settings

a) In the “wind direction” settings enter a “3” in the “i” field. The i, j, and k fields represent a vector.

b) In the “wind magnitude” field enter a value of “2”. This will be the strength of the wind in the direction indicated in the “wind direction” fields.

8) Notice that there is also a “particle system” section in the Properties palette window for the weather palette object.

If rain or some other weather effect was needed in the level, this is the field where a particle system and associated tags would be referenced.

The particle system would be affected by the wind values set.

The first image further explains the above procedures.

The weather will now be applied to all the portals or clusters in the level.

8) Go to the Hierarchy view window. In the Hierarchy tree expand the “Cluster properties” branch. Next, expand the “Palettes” branch and select the “weather” subtree object

The portals or clusters should now appear in the Game Window. The volume for a portal or cluster is outlined. The currently applied weather is color coded for each portal.

9) In the Tool Window, the “Render planes” and “Render non painted clusters” check boxes can be enabled or disabled to help view the portals better.

Move the camera in the Game Window if it is not already clearly in a portal or cluster.

10) The current weather that is applied to the portal that the camera is in is displayed in the “weather” pull down menu located in the Properties palette window. In this case, “NONE” appears since no “weather” have been applied.

To apply “weather” to the portal, use the weather pull down menu to select the “forest wind” weather system. The weather for the portal has now been set.

Move the camera to the next portal and repeat this procedure until all the portals for the level have the “forest wind” sound environment set.

Super Simple Teleporter Tutorial

- view
- edit
- attach
- print
- history

Teleporters are super simple.

Mission>Netgame flags>Properties palette

When placing a netgame flag you change the type to “teleport from” and give it a team index of 0.

Place the next flag where you would like to teleport to and change the type to “teleport to” and leave the index at 0.

Now if you want a 2 way teleporter you will have to assign new placement flags with a new index of 0 & 1 and place the flags in the same spots with reversed types.

```
«»«»«»«»«»
teleport from=index of 0
teleport to=index of 1
-----
teleport to=index of 0
teleport from=index of 1
«»«»«»«»«»
```

You can place as many as you want all over any map as long as "tele-from" and the "tele-to" have the same index number. The 0 & 1 do not represent RED or BLUE such as the CTF flags do. In the case of the teleporters, they can have higher values than 0 or 1.

```
«»«»«»«»«»
example:
tele-from=index of 7
-----
tele-to=index of 7
«»«»«»«»«»
```

from=source
to=target

Face the BLUE arrow on the target towards the wall if you would like to appear in those areas coming out face first.

Hope this helps...

a d®ûÑk ©øp
kidemblazon@yahoo.com

Four Seated Hog

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Ang88's 4 Seated Warthog

-First make a copy of the Warthog folder which is in the tags\vehicles\ folder

-Then rename the folder, and the mp_warthog.vehicle file to something different, this helps when you populate in sapien so that the vehicles will

have different names

-Now, open up Guerilla, and goto File-Open

- from here, open up the new vehicle.vehicle tag

-Scroll about 3/4 of the way down, until you see a section called seats

- this section has all the options for the different seats

-First select the gunner seat in the drop down menu

-Then push delete to delete it, because we won't need it anymore

-Now select the passenger, and push duplicate

-These are the options we want to change for the seat we just created

-Flags: Check Third Person Camera

-Label:scorpionRB

-This tells halo how MC sits in the seat, it is a name for the type of animation

-Marker Name:gunner right foot

-This is the location of the seat, you can find all the markers if you open the warthog's model and goto the markers section

- Camera Marker Name:gunner right foot

- This is the location for rotation of the camera

- Now duplicate this seat

- In the new seat, we just want to change 3 things

- Label:scorpionLB

- Marker Name:gunner left foot

- Camera Marker Name:gunner left foot

- Now save

- Next, we want to get rid of the gun turret

- To do this, we are going to make a shader that is totally clear

- To do this, make a bitmap with a black alpha channel

- Then make a shader_enviroment tag, that has the alpha-tested option check, and the bitmap you created as the bump map

- Now open up the vehicle tag again

- where it says model, click “...” and make sure that the model is the one in the new vehicle folder

- after you picked the right model tag(tags\vehicles\new_warthog\warthog.gbxmodel) push open to see the properties for the model

- Now scroll all the way down to the bottom where it says shaders

- Using the drop down list, pick the shaders that apply to the Chaingun, and change the shader to the clear one you just made

- After doing this, push save

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

REQUIREMENTS: 3D Max, Photoshop, HEK

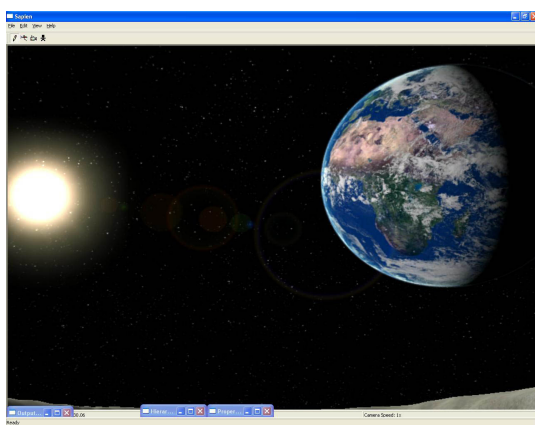
ACKNOWLEDGEMENTS:

I originally learned how to edit certain parameters of existing multiplayer skies using Mattyp001's [Custom Sky Tutorial](#) available from [Halomods](#). Also, the Photo-Realistic Earth model, renderable in 3D Max was created through the use of a 3DS Tutorial from Noir Extreme([approve sites](#)).

DESCRIPTION:

I'm currently working on a [Halo CE](#) map entitled *Sea of Tranquility* - a purpose built training arena on the Moon - that requires a custom sky, altogether different to those provided with the HEK. This tutorial covers step-by-step, the process of re-creating this sky, by modifying a copy of an existing multiplayer sky & inserting custom images where appropriate. The end result is a sky tag that can be used in any level, however the process of sky modification is probably more interesting.

The Finished Sky:



(Click on thumbnail to enlarge)

STEP BY STEP INSTRUCTIONS:

Note: This sky is created by modifying copies of the original Halo skies. This is not very flexible, but it is still possible to get a great result. If you want to create your own Gearbox Model & therefore a truly custom sky, go [HERE](#) for the sky Gearbox Model tutorial.

1.) Open up your Halo CE directory and Navigate to data/sky. Add a folder called moonbase. Within that, add a folder called bitmaps. Your directory structure should then be:

...//Halo Custom Edition/data/sky/moonbase/bitmaps

2.) Now navigate to ...//Halo Custom Edition/tags/sky/. The sky we will be bootlegging for our custom sky is dusk (not space! as space has fairly rubbish uvw mapping when you look straight up & also texture quality is poorer for some reason). Copy the entire folder dusk and paste it, but renaming it after paste from 'copy of dusk' to moonbase. In tags, the directory structure should now look like this:

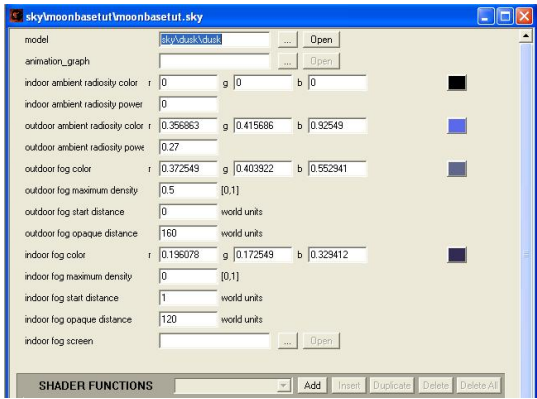
...//Halo Custom Edition/tags/sky/moonbase

3.) Within tags/sky/moonbase, there will be the following files that must be renamed accordingly:

- dusk.gbxmodel (file)
rename to moonbase.gbxmodel
- dusk.sky (file)
rename to moonbase.sky
- sun.lensflare (file)
delete this. Copy in instead ...//Halo CE/tags/sky/space/sun in space.lens_flare
- bitmaps (directory):
this is where your custom bitmaps will go once compiled. Leave this atm.
- shaders (directory):
rename all the shaders to be _moonbase, so basis becomes basis_moonbase.shader_transparent_chicago etc...

The reason for this renaming is just so you don't accidentally modify and mess up your dusk sky. All these files link to each other in Guerilla, so its a good idea to rename them just so you don't link to, and permanently change the wrong ones!!

4.) Open Guerilla, navigate to ...//Halo CE/tags/sky/moonbase & open up moonbase.sky. This is what you should see:

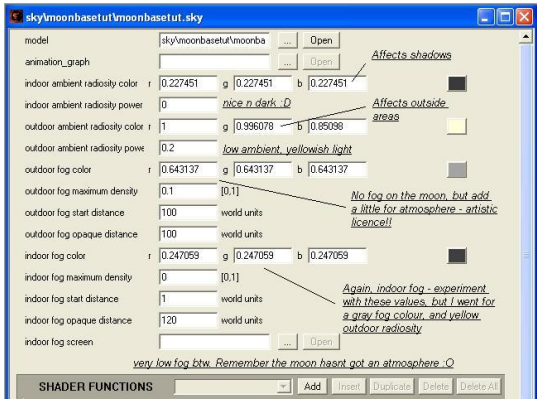


(Click on thumbnail to enlarge)

Notice that it's still pointing to the sky/dusk/dusk.gbxmodel. This is not what we want, so first off, change the model to sky/dusk/moonbase.gbxmodel.

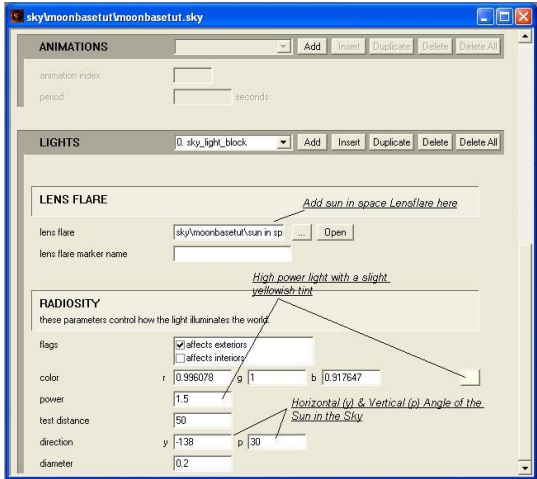
Note: In my screens it says moonbasetut.sky - dont worry about this, I already have a moonbase.sky, so I named mine moonbase tut just so there wouldn't be a conflict. All my files are named with tut, but yours should be named just moonbase.

This is what you should be aiming for:



(Click on thumbnail to enlarge)

While you're there, scroll down and add the sun in space lensflare, and change the colour of the emitted light from the sun, angle and strength of the light:

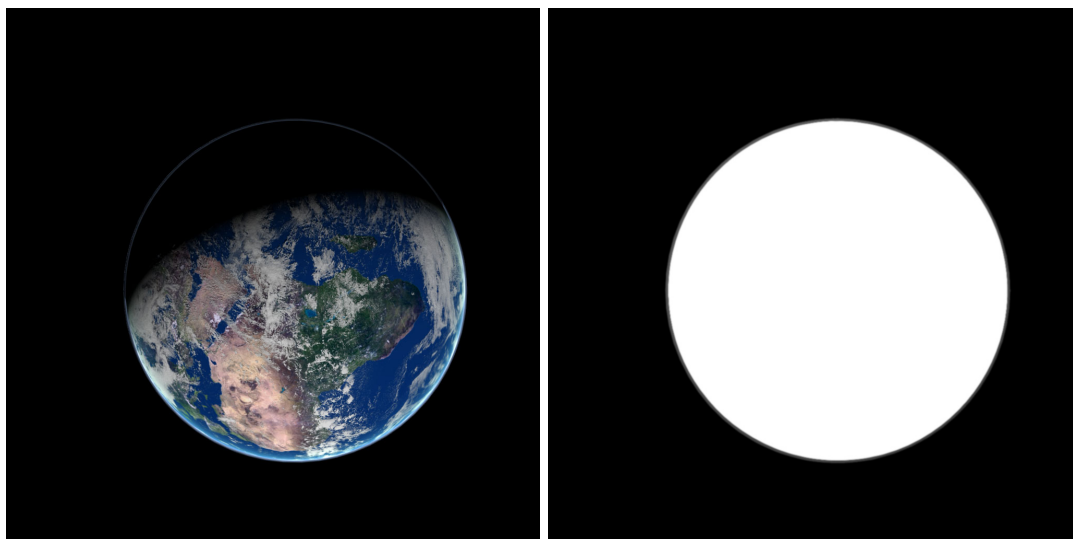


(Click on thumbnail to enlarge)

5.) Hit file→save all to save your sky and open up Photoshop. First, open up a new file, give it a size of 32×32, make it RGB colour and hit ok. Go to the colour palette and select black as your foreground colour. With the image selected, press alt-backspace to fill in the image black. Now, go to the layers window and select the channels tab. We want to create a new channel by hitting the icon in the bottom right of the channels palette. This should create an alpha channel which will be filled in black. If it's not filled in black, fill it in, and save your image in ...//Halo CE/data/sky/bitmaps as nothing.tif (Ensure compression is OFF, and that you saved the alpha channel).

6.) If you want, you can open up Max, and play around with the photoreal earth file available from Noir Extreme([approve sites](#)). I downloaded this file and played around with it a bit until I got a nice shot of the earth on a black background with the light coming from the left side. I chose Europe/Africa as the centre of this shot as its a nicely detailed piece of land with plenty of interesting features to look at. Also, I deleted the 'earth sky at night' texture from the darkened side of the earth, as this gave the impression that the side of the world in shadow was not completely dark.

Alternatively, if you can't be bothered to do all that, download the image I used!!



([Download Earth1024 in .tif Format](#))

Get the tif image into photoshop and check that it has both colour and alpha components like the images above. Alternatively, if you want to generate your own image by rendering the max file, make sure the alpha channel is white for the portions of the earth you want to be visible (opaque) and black for the portions that are transparent. I also used a slight gaussian blur on the alpha channel, to give the effect that the thin atmosphere around the earth was semi-transparent.

Once your tif is complete, you have either downloaded or created your own, place it in ...//Halo CE/data/sky/moonbase/bitmaps

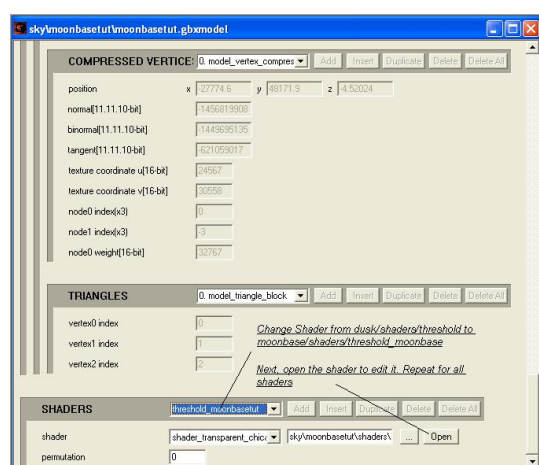
Note: If you're wandering about the wierdly rotated earth in my pics, its because Guerilla is up to its old tricks and the earth appears all wierd and rotated in the final sky. So, I rotated my earth pic anti-clockwise 110 degrees to counteract this effect.

7.) Now its time to compile the bitmaps in Tool. Go to the start menu in the bottom right of Windows, and find 'run'. In run, type 'cmd' and press enter. Now in the command console, type 'cd "c:\program files\microsoft games\halo custom edition"' and hit enter - do not include the single quotes, but use the double quote around the path for your HEK installation. Finally, compile the bitmaps running the command:

tool bitmaps sky/moonbase/bitmaps

You may get a warning for the bitmaps: 'bitmap with greater than 1 bit alpha...' for earth1024.tif and 'compressed as color key transparency' for nothing.tif. The colour key transparency is a bit of a mystery. I only got this error once when compiling bitmaps & to be honest, when I made the sky I got the 'bitmap with greater than 1 bit alpha' error for both. If when you get to the end of the tutorial, there are large blacked out squares in the sky, it will be because of the colour key transparency error. If not, then hey it worked!! Either way, the earth1024 though, must be fixed.

In Guerilla, goto file→open and navigate to ...//tags/sky/moonbase/bitmaps and open earth1024.bitmap. Change the format to 'compressed with explicit alpha' and save & close the bitmap tag.



(Click on thumbnail to enlarge)

Now, recompile in tool using the same command as above, and the bitmap tag should now be saved as compressed with explicit alpha.

8.) Close Photoshop & the command prompt for the moment, and go into Guerilla. If it's not already open, open up moonbase.sky, and scroll back up to the top of that dialog. Now we need to edit the gearbox model slightly, so next to model at the top, hit open.

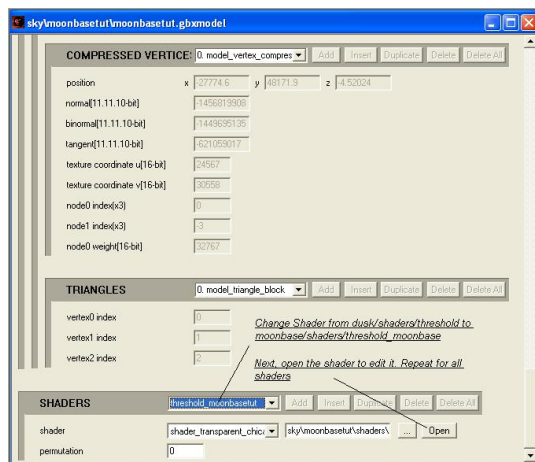
The gearbox model (moonbase.gbxmodel) has been compiled as a model using tool, from a JMS file produced by 3ds max. Essentially, there are a number of 'regions' that have been created in the 3DS file, each with its own shader_transparent_chicago. These shaders, be found at the bottom of the gbxmodel in a drop down box include:

- Threshold - The large gas giant next to the Halo. We will be injecting our earth picture into this shader.

- Basis - A small moon directly overhead in this gbxmodel.
- Ring - The ring of the Halo as it disappears into the distance
- Water - The reflective & animated water leading up to the horizon
- Horizon - err, the horizon
- Sky - The sky, including the blue dusk effect, stars and star twinkle.
- Clouds - The clouds visible on one side of the sky

As this tutorial is essentially about hacking the dusk.sky, we cannot delete or move the locations of these shaders, so what we are going to do is edit the ones we don't want to have an invisible texture, & inject our earth1024.bitmap into the one we do want.

In order to edit a shader, scroll down to the bottom of the Gearbox Model file, select a shader in the drop down box, change the shader to the moonbase version (ie: from sky\dusk\shaders\threshold_dusk.shader_transparent_chicago to sky\moonbase\shaders\threshold_moonbase.shader_transparent_chicago) and then open it. We want to change the shader to the moonbase version so that we don't accidentally & irreversibly change the dusk.sky!!

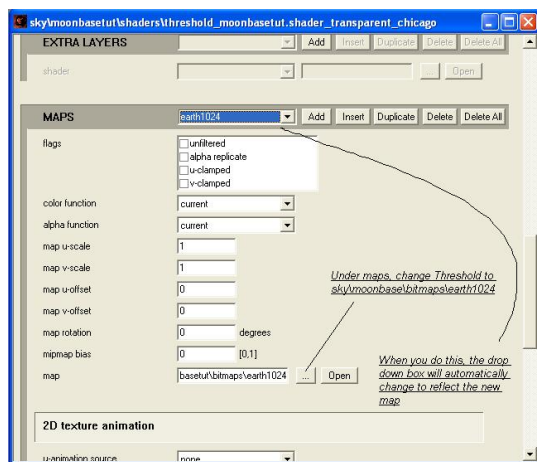


(Click on thumbnail to enlarge)

Repeat this for all shaders - change them over to the moonbase versions and open them. One by one, we then need to edit the shaders we have just opened. The process for this is described below.

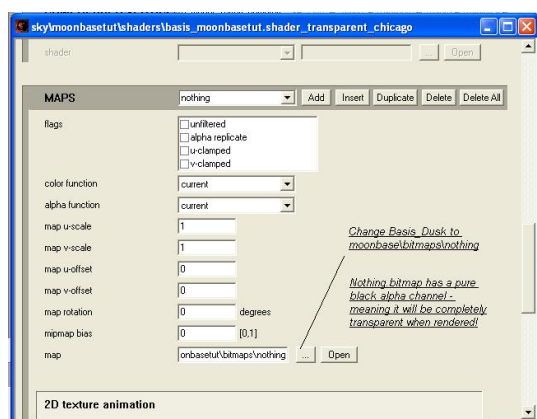
9.) Adjusting the Shader Bitmaps.

In the threshold_moonbase shader, scroll down until you find the section called bitmaps. There should be a drop-down box with 'threshold' written in it. This is the current bitmap in use. We want to inject our picture of the earth here, so at the bottom of the maps section, where the path to the current map is, browse for and open the earth1024 bitmap, in your sky\moonbase\bitmaps folder. When you have finished, close the shader and save it.



(Click on thumbnail to enlarge)

Next, open up the basis shader. This is a small moon in the sky in the dusk model, so we want to edit this out. Import the bitmap 'nothing' - with its pure black alpha channel - into this shader. When the sky renders, the black alpha channel will cause this region to be completely transparent.



(Click on thumbnail to enlarge)

Continue this for all the shaders. The remaining shaders must all be injected with 'nothing.bitmap', but note that some shaders have more than one map. A description of which shaders have which maps are as follows:

Note: You can cycle through maps on a shader by clicking on the drop-down box under the maps section. Make sure you apply 'nothing.bitmap' to all of them!!

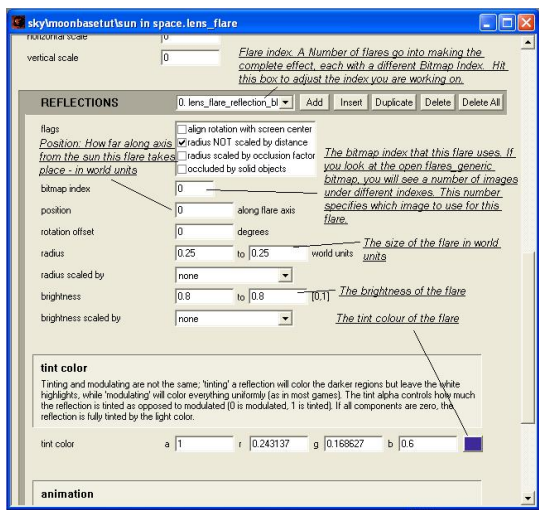
- Threshold shader: Map is threshold_dusk.bitmap. Replace with earth1024.bitmap
- Basis shader: Map is basis_dusk.bitmap. Replace with nothing.bitmap
- Ring shader: Map is ring_dusk.bitmap. Replace with nothing.bitmap
- Water shader: Maps are water_dusk, reflection_water_dusk & waves_bump. Replace all with nothing.bitmap
- Horizon shader: Maps are detail_ground and horizon_land. Replace all with nothing.bitmap
- Sky shader: Maps are stars, stars_twinkle and sky_dusk. Replace ONLY sky_dusk with nothing.bitmap
- Clouds shader: Map is clouds_dusk. Replace with nothing.bitmap.

10.) Adjusting the Lens Flare to get a realistic sun effect.

Close all the files you have open in Guerilla and open up the sky\moonbase\sun_in_space.lens_flare tag. We

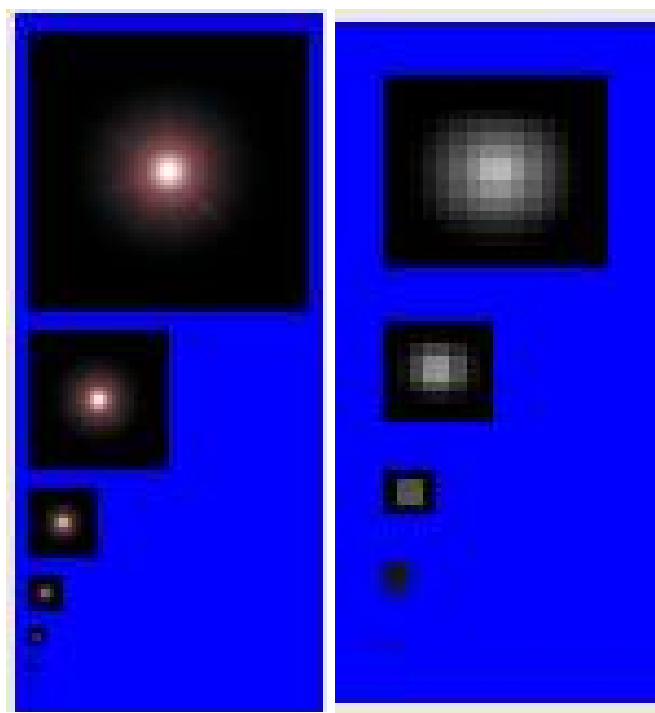
want to make some simple adjustments to this tag to get a more realistic, yellowly/red, burning sun in the sky with a bright corona & lens flare effect.

First, under the bitmaps section, check the box that says ‘sun’. Also, while you’re there, open up the flares_generic bitmap by hitting open just above the sun checkbox. Now, scroll down to the bottom of the lens flare tag where it says ‘reflections’.



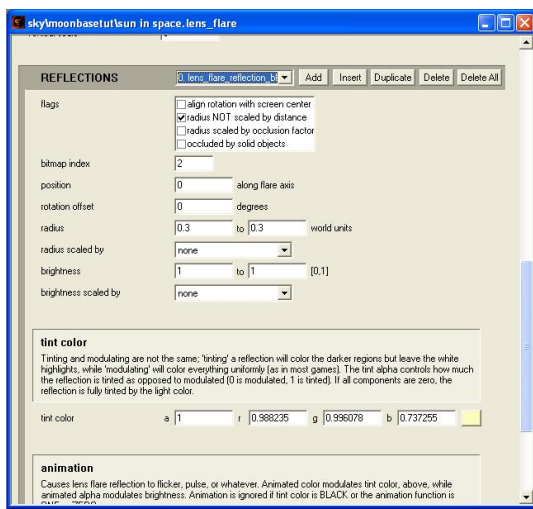
(Click on thumbnail to enlarge)

A whole load of paramters are here that effect the lens flare. First off, the current sun_in_space flare has a sort of blue/purple star shaped tint to the edges. The sun in our solar system is much more yellow and softer at the edges, so we want to change the image index and the tint.



(Image Index 0) (Image Index 2)

For Lens Flare Reflection Block 0, we want to change the image index from 0 to 2. This has the effect of removing the harsh star effect with a purple outline & replacing it with a soft corona that we can colour as we wish. Image indexes can be viewed from within the flares_generic bitmap, but hitting ‘show bitmap’ and changing the index in that dialog. Next, we want to change the radius to 0.3, to make it a little larger, the brightness to 1, and the tint to a soft yellowish colour.



(Click on thumbnail to enlarge)

Repeat this for all lens_flare_reflection_blocks, using the following parameters:

- Block 0:
Bitmap index: 2. Position: 0. Rotation: 0. Radius 0.3/0.3. Brightness: 1. Colour: r=0.988235, g=0.996078, b=0.737255.
- Block 1:
Bitmap index: 1. Position: 0.15. Rotation: 0. Radius 0.025/0.025. Brightness: 0.2. Colour: r=1, g=0.5, b=0.
- Block 2:
Bitmap index: 2. Position: 0.18. Rotation: 0. Radius 0.02/0.02. Brightness: 0.2. Colour: r=0.18, g=0.78, b=0.
- Block 3:
Bitmap index: 0. Position: 0. Rotation: 0. Radius 0.28/0.28. Brightness: 0.9. Colour: r=1, g=0.988235, b=0.72549.
- Block 4:
Bitmap index: 1. Position: 0.26. Rotation: 0. Radius 0.06/0.06. Brightness: 0.25. Colour: r=1, g=0.5, b=0.
- Block 5:
Bitmap index: 4. Position: 0.35. Rotation: 0. Radius 0.09/0.09. Brightness: 0.25. Colour: r=1, g=0.5, b=0.
- Block 5:
Bitmap index: 4. Position: 0.35. Rotation: 0. Radius 0.09/0.09. Brightness: 0.25. Colour: r=1, g=0.5, b=0.
- Block 6:
Bitmap index: 1. Position: 0.37. Rotation: 0. Radius 0.04/0.04. Brightness: 0.35. Colour: r=1, g=0.5, b=0.
- Block 7:
Bitmap index: 1. Position: 0.43. Rotation: 0. Radius 0.03/0.03. Brightness: 0.35. Colour: r=0.3, g=0.8, b=0.25.
- Block 8:
Bitmap index: 2. Position: 0.45. Rotation: 0. Radius 0.02/0.02. Brightness: 0.35. Colour: r=0, g=0.28, b=1.
- Block 9:
Bitmap index: 5. Position: 0.52. Rotation: 0. Radius 0.05/0.05. Brightness: 0.25. Colour: r=1, g=1, b=1.
- Block 10:
Bitmap index: 6. Position: 0.61. Rotation: 0. Radius 0.18/0.18. Brightness: 0.15. Colour: r=0,

g=0, b=0.

- Block 11:

Bitmap index: 4. Position: -0.08. Rotation: 0. Radius 0.05/0.05. Brightness: 0.15. Colour: r=1, g=0.5, b=0.

- Block 11:

Bitmap index: 3. Position: 0.0. Rotation: 0. Radius 0.1/0.1. Brightness: 0.15. Colour: r=0.88, g=0.43, b=0.

Ok, save close and save that file once you're done - You're lens flare is now complete!!

11.) Once you've done that, you have completed your custom sky mod!! Save all your files and to test the sky, open up a scenario tag for a map you are working on, & inject moonbase.sky into the multiplayer skies section!!

Final Stuff

Nice, now that you have learnt how to do your own custom skies, feel free to experiment with adding different bitmaps & such. Note that if you want to change the location and size of images, this is not possible by modding current skies. What you must do is create your own sky Gearbox Model before applying bitmaps in Guerilla. This is possible using 3D Max and a copy of tool - and is described [HERE](#).

E Vi L Da VE :@

How to Add Water

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

How to add water

This is how you create water: Create a plane that is bigger or smaller than the level itself, or wherever you wanna put it. It doesn't follow the Sealed World Rules, so it can extend past the borders of the level. Then, create a new sub material. Name it **water!** (yes, the **!** must be there). Then, apply that texture to your plane. Then, when exporting, it'll automatically take the water shader in your tags directory since your water was called "water", which is what it was called in Death Island, and use it.

If you want your water to have fog underneath it you have to name the material **water!\$** and apply fog to the portal it creates in sapien

For an example bitmap file are that you can attach to water!\$, look at: tags\levels\test\deathisland\distant water.shader_transparent_water

Authors: [Super](#) [Sunny?](#) & Baka

Original Ghost Plasma Sticky Thread

If anyone wants the Ghost’s plasma stickiness effect, they have to add it themselves for their maps. Only Gearbox can readd it to the default maps with a patch, which they probably won’t ever do.

(Why it’s useful.)

It’s simple.

1. Open Guerilla.
2. Open the file \tags\vehicles\ghost\ghost bolt.damage_effect
3. Scroll down to the “Damage” section.
4. Change the following:

Stun - 1
Maximum Stun - 1
Stun Time - 0.15 [seconds]

(Values as shown in the Vehicles\Ghost\Ghost Bolt section of a [Halo PC](#) level viewed with Halo Map Tools.)

5. Save.

That’s it. The next time you compile a level that has a Ghost, that Ghost’s plasma is [Halo PC](#) sticky.

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Chain Spawning Tutorial

This will teach you how to make a projectile spawn another projectile, or other things, after it is destroyed. So if you want a frag grenade to explode, then have 3 armed plasma grenades come out of it, this is the place.

Lets get started.

1. First, go to your “Halo ce\tags\weapons\frag grenade” folder and make a copy and paste it somewhere. It’s a good idea to make a copy of anything you’ll be altering.
2. Now open up Guerilla, and open go to the “weapons\frag grenade\effects” folder, and open “explosion.effect”.

We are going to make a simple chain grenade that spawns 3 plasma grenades.

3. Now that you have explosion.effects open Guerilla, go down to where it reads “PARTS”. This is the 4th section from the top.
4. In this section, click the “Add” button 3 times.
5. Now in the dropdown bar you shuold see “8.effect_part_block”. Select it.
6. Now 3 tabs down is “location” dropdown bar. Click this and select “0. effect_locations_block”.
7. Skip flags and go to the “type” dropdown bar. Select “projectile”. In the tab to the right, hit the “...” button, and go to “weapons\plasma grenade\plasma grenade.projectile”.
8. In the “velocity bounds” section, enter “0.2” in both sections. I believe this is the speed that the projectile travels when spawned. So the higher it is, the further and faster it will go.
9. In the “velocity cone angle” section, enter 25 degrees.
10. In “angular velocity bounds”, put “0.5” in both sections.

11. Repeat this process for the other two “effect_part_block” dropdowns. Just make sure you change the “velocity cone angle”, so the grenades spread out in different directions.

12. Now save the file, and recompile your map.

That’s it, you have just created a frag grenade that will spawn 3 plasma grenades when it explodes. Then the plasma grenades will explode. The possibilities are endless. Try putting other heavy weapon projectiles, or even vehicles and weapons in place of the plasma grenade. You can even spawn a MC decoy, just use the correct “type” tag. “vehicle” for example if you want to spawn a warthog or ghost. Just be careful with the vehicle spawning, to prevent lag, and turn off vehicle respawn in the setup menu, otherwise they will keep piling up at the same spawn points that you set in your map. Also don’t get carried away with the chain spawning, if you were to make an assault rifle fire rockets that spawn 100 extra projectiles, the engine will not be able to handle it and the game will crash. Also, just for fun, edit the plasma grenade to spawn plasma rifle bolts when it explodes, or you can even edit the plasma rifle bolt impact effects files to make it appear to bounce off the walls. Have fun, and soon I will make a quick tutorial on duplicating projectiles. This is so you can have a frag grenade spawn other frag grenades without it going on forever. See you online sometime.

Video demo: <http://files.halomods.com/viewtopic.php?p=97381>([approve sites](#))

Deimos

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

Basic Destructible Scenery\Vehicles tutorial

Author: [The Missing Link 117?](#)

Ok, I figured this out yesterday while making some custom scenery for a map I'm working on. This tutorial only covers the bare basics, because that is all I have been able to figure out yet.

So first off, you follow that guy's (sorry, I think it's mattp001s or something) tutorial on making scenery with collision geometry.

To make the scenery destructibe, you open the materials .model_collision_geometry tag, and edit it, using the SP wraith, ghost, and banshee tags as references.

First off, there are some flags that you can check. Takes shield damage for children and takes body damage for children probably mean when a person is in a vehicle, the vehicle takes the damage, not the driver/passenger. Some of them are self explanatory, only damage while occupied means it can't be destroyed unless someone is in it, and only damaged by explosives means bullets/plasma wouldn't hurt it.

You set the maximum body vitality as the amount of life it has. The body damaged threshold is when the body damaged effect happens. (ex: The banshee starts smoking when it has this much life left).

The body depleted effect is what happens when all of the life has been drained. You can use whatever explosion you wan't here, The wraiths explosion, the plasma explosion, whatever you wan't, depending on the type of scenery\vehicle.

Under shield, I think you add that stuff if you want your vehicle\scenery to have a shield that protects it, similar to the chiefs.

All I did under materials was selected my collision-geometry material, made it thick metal, and set body damage multiplier to 1. Under regions I set the only _unnamed to die when object dies, because it was checked under the wraiths tag. I think that if you had more than 1 part of your vehicle, you could set it so that some parts can be destroyed/smoke before others.

This has just been a very basic tutorial, The next step in this process is figuring out how to make the scenery

object dissapear and/or make a new model appear that is a destroyed version, with a destroyed shader. If anyone else has made any discoveries about this, feel free to add on.

Also, If you are making a vehicle destructible, remember if you wan't it to show up in game you would have to replace it with the normal one in globals, or just swap .vehicle tags.

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Custom Weapon Tutorial By Choel & DepthCharge

Section 1 - Before the HEK

REQUIREMENTS:

3DMax, HEK, a 3DS Model of a Weapon to be Injected into Halo

ACKNOWLEDGEMENTS:

This tutorial has been kindly written & submitted by **Choel & Depthcharge**, after the production of their Unreal Tournament Weapons Mod for Halo. The UT Mod threads can be viewed at the following pages. Check out their work as it's impressive stuff!!

- [UT Mod Thread 1](#) (Gearbox)
 - [UT Mod Thread 2](#) (Gearbox)
 - [Energy Sword FP In Game Pics](#) (Halomods)
 - [UT Mod Progress Update](#)([approve sites](#))
-

DESCRIPTION

A step by step tutorial on how to inject custom weapons into the HEK. Submitted over several parts, this tutorial will cover ..., ... and ... and some other ... too (to be added :P).

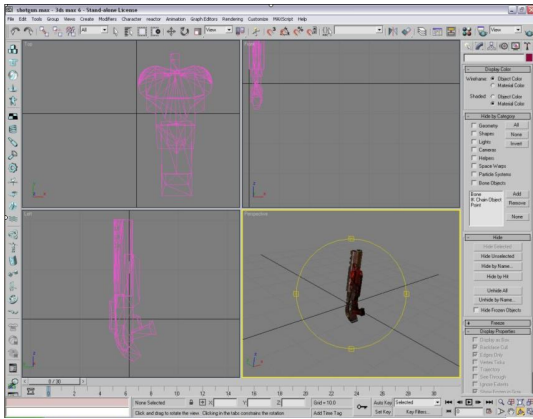
This tutorial assumes you have a model of a weapon in 3DS or [G Max](#) ready to be injected. If you don't have a weapon, try using this freeware model of a Glock-17 from Turbosquid.

- [Turbosquid Freeware Glock \(3DS\)](#)
-

Custom Weapon Tutorial - Part 1–1: Setting up Your Model

‘Note:’ PLEASE MAKE SURE THAT YOU KNOW ALL THE TERMINOLOGY AND HAVE READ THE GEARBOX TUTORIAL ON MAKING MAPS!!

Alright, because I have no idea what your weapons look like, were just going to assume that you've made a basic gun, like this shotgun ...

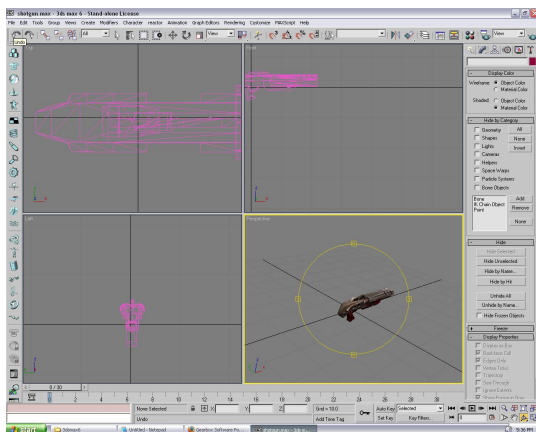


(Click Thumbnail to Enlarge)

Step 1: Rotate your gun so the part where the bullet/projectile is supposed to come out is pointing the direction of “X” on the Compass. For the Shotgun, will need to rotate 90 degrees in Z and 90 degrees in X.

Step 2: We need to scale down (or Up) your model, for reference, here is a picture of my shotgun, in max, and then a pic in sapien.

Note: Use the Master Chief Reference Model as a guide when sizing



(Click Thumbnail to Enlarge)



(Click Thumbnail to Enlarge)

Step 4: Create a Box, thats $64 \times 64 \times 64$, name it frame gun.

Step 5: At the front of your gun, at the end of the barrel, make a shape, any shape, and name it (without quotes) “#primary trigger”

Step 5.5: If you want your weapon to own with animations and stuff use the script that James wrote. It imports the .gbxmodel tag of your choice with nodes, markers, and geometry. The weapon that you are going to base your animations on needs to have these markers. Open up your favorite .gbxmodel tag in guerilla and go file>export and put it somewhere, don't forget where it is. The script by default is in the “Halo Custom Edition” folder so exporting it to there isn't a bad idea. So get the script (<http://files.halomods.com/viewtopic.php?p=102646#102646>[\(approve sites\)](#)) and put in your scripts directory (it might not be a bad idea to read through the readme file it comes with) and then go from the menu at the top of 3dsmax\gmax Maxscript>Run Script>pick the script called gbxmodelimporter. It asks you to open file and then you need to open up the .txt file of a .gbxmodel tag that you had guerilla export. It will import the

stuff I said above. Move the markers to where you want them to go for your weapon. You can use the geometry it imports to sort of figure out where the marker was originally and what it probably does.

Step 6: Select all parts of the model that you want exported, click select and link, then click select by name, and then click frame gun, then Link.

***Note:** IF you think you want to do some more advanced crap, then you should know that you can have more than one frame, but they should all be linked to frame gun. Some other frames that are used are: frame magazine, frame display, frame wing and many others, you can use gurilla to find more.*

Your model is now ready for exporting, move on to part 2 (for TP and FP)...

Custom Weapon Tutorial - Part 1–2, Exporting your model

Ok, you got frame, lets get it in game, and don't fap at my rap.

Step 1. ok, click on File > export.

Step 2. Navigate to your HALO CE directory, and go into data.

Step 3. If you don't have one already, make a weapons directory inside of data

Step 4. in in your /data/weapons/ create a directory for your gun, in my case, its gonna be /q3shotgun/

Step 5. now in your directory for your weapon, /data/weapons/Disco_raper/, or whatever, make ANOTHER NEW FOLDER, call it models.

Step 6. /data/weapons/YOUR_WEAPON_NAME/models/ <— thats where you should, and remember, don't have spaces in your directory name, tool.exe doesn't like it, type in a name for your model, open the file types and find JMS, NOT JMA! Hit enter. You should now have an exported jms file.

Step 7. Reexport the model, into /data/weapons/YOUR_WEAPON_NAME/FP/models/

Step 8. If you have a texture, make sure its dimensions are powers of 2 (IE 2×4, or 16×16, or 1024×256, or 512X16 ..) and move it to /data/weapons/YOUR_WEAPON_NAME/bitmaps/

Now, lets compile the model

Custom Weapon Tutorial - Part 1–3, THE DREADED TOOL.EXE

Too.exe is not a buggy program, if it was, its Error would almost always be “USER ERROR: PLEASE REPLACE USER”. With that said, the tool errors you guys get are because of the CRAP that you shove into it.

Step 1. IF your have [HE Kter](#), or any other tool.exe front end, right click it, and hit delete.

Step 2. Make a new text file, and rename it “MAKE_YOURWEAPON.bat”

Step 3. Right click it, and hit edit.

Step 4. You should see a blank text document...

TYPE IN IT (without the quotes):

```
"Tool.exe model weapons\YOURWEAPON"
```

```
"PAUSE"
```

```
"TOOL.EXE model weapons\YOURWEAPONS\FP"
```

```
"PAUSE"
```

```
"TOOL.EXE bitmaps weapons\yourweapons\bitmaps"
```

```
"PAUSE"
```

Save and Close it

Note: *If your model has no open edges, misfliped normals, or things like that, you should also make a copy of your models folder and name it physics, then at the end of your .bat file, add:*

```
"TOOL.EXE collision-geometry weapons\yourweapons"
```

```
"PAUSE"
```

Step 5: Double click the .bat file you just created It should ask you for a shader type, I recommend #2

Note: [Super Sunny](#) has said that #1 is better because it adds more options, more options ... yes, needed options for a normal model, no.

Step 6: Close the .bat window, and then re-double click it, and hit enter every time it asks you to do so.

You should now have your gbxmodel tags and bitmap tags ... ON TO SECTION 2, YOUNG GRASSHOPPER!

Now read [Custom Weapon Tutorial Part2](#) by [Choel](#) & [DepthCharge](#)

- view
- edit
- attach
- print
- history

Simple Weapon Tutorial - Step by step

Author: ZiON

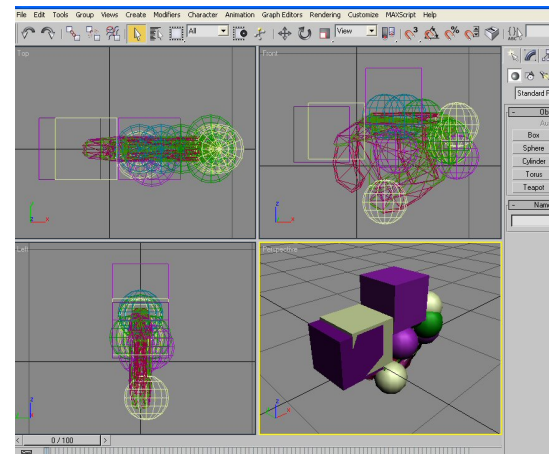
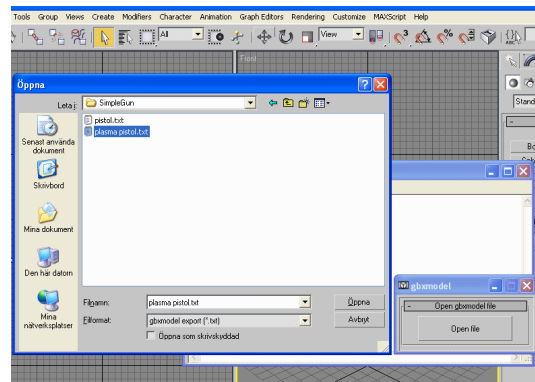
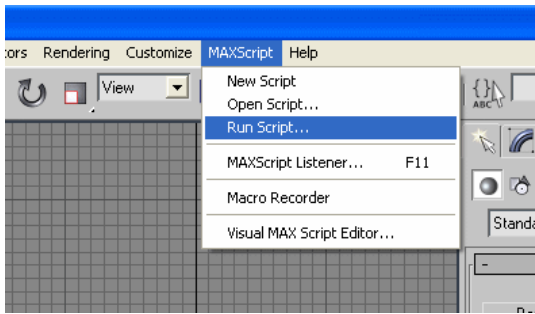
Acknowledgements: James @ www.halomods.com

Ok, so we wanna do a very simple weapon, an additional pistol. This is how:

Since we wanna do a pistol, we are gonna use the pistols animations and such.

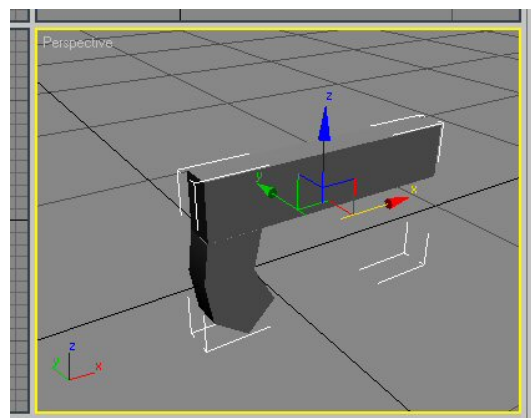
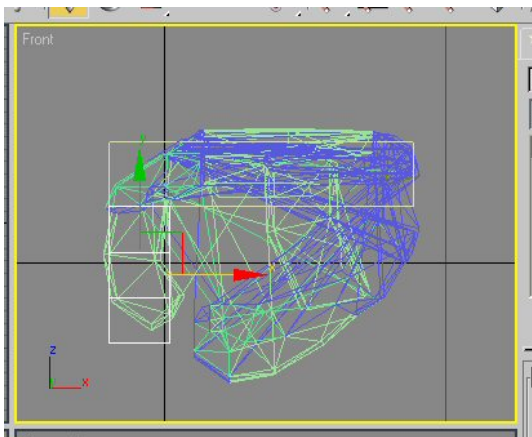
Part 1 - Importing the existing pistol's markers and geometry

1. Create the following directory structure:
 1. \Halo CE\data\weapons\simplegun
 2. \Halo CE\data\weapons\simplegun\bitmaps
 3. \Halo CE\data\weapons\simplegun\physics
 4. \Halo CE\data\weapons\simplegun\shaders
 5. \Halo CE\data\weapons\simplegun\models
 6. \Halo CE\data\weapons\simplegun\fp\models
 7. \Halo CE\data\weapons\simplegun\Max
2. Open Guerilla
3. Open the tag \Halo CE\tags\weapons\plasma pistol\plasma pistol.gbxmodel
4. In Guerilla's menu, select *File - Export* and save the file to \Halo CE\data\Max\plasma pistol.txt
5. Download James model importer, install it using the instructions that comes with the script.
6. In 3D studio Max, choose *MAX Script? - Run script* and select the **gbxmodelimporterX_X.ms'** script.
7. Click Open file, navigate to \Halo CE\data\Weapons\simplegun\Max\ and open the file **plasma pistol.txt**.
8. After some processing, if it all turns out good, you should have the Halo pistol imported properly into 3dsmax. There should be a few boxes and spheres there aswell. Since we aint really gonna use the pistol itself, but only the markers we will keep it all in there and go to the menu and save the file to \Halo CE\Data\Weapons\simplegun\Max\ as **simplegun.max**.



Part2 - Creating our own “gun”

1. We are gonna use the original gun as a rawmodel, but to do that we need to hide all the markers. Click the “*Select by Name*” icon in 3DS Max. Select all names that starts with a # and all that starts with Frame, then click Ok.
2. in any of the viewports, right-click and select *Hide Selection*.
3. Create a box roughly surrounding the upperpart of the plasma pistol. Name it **Upper Part?**.
4. Create another box roughly surrounding the handle plasma pistol. Name it **Lower Part**.
5. Remove the old Plasma gun, all of it, but leave the markers and frame objects intact.
6. Texture the gun as you like and put all bitmaps in the \Halo CE\data\weapons\simplegun\bitmaps folder.
7. Remember to link all parts of your gun to the “**frame gun**” object.



Part3 - Exporting the gun

1. Right-click in any viewport, select *Unhide all*
2. Go to *File - Export*, Select *Halo Model Exporter* as the filetype.
3. Export the model to:
 1. \Halo CE\data\weapons\simplegun\models\
 2. \Halo CE\data\weapons\simplegun\fp\models\
4. Save the gun to **simplegun.max**.
5. Go to the material editor, create a new material which you name “collision-geometry” (without quotes), assign it to the whole gun (all parts).
6. Export the gun to \Halo CE\data\weapons\simplegun\physics\
7. Save the gun to **simplegun-collision.max**.

Part 4 - Tool

Ok, we got this far. It’s this part that might not go as good as it did to this point :P

- 1. Bring up a command prompt.
- 2. Change directory to your halo folder. (cd “\program files\Microsoft Games\Halo CE\”).
- 3. At the command prompt write:
 - 1. **tool model weapons\simplegun**
 - 2. **tool model weapons\simplegun\fp**
 - 3. **tool bitmaps weapons\simplegun\bitmaps**
 - 4. **tool collision-geometry weapons\simplegun**
- 4. If asks for shader type, select 2 and repeat the command once more.
- 5. If something failed, go check that you have linked it to the frame etc. If the Collision-geometry failed (which it most likely did) either go try to fix it in 3DS Max or simple to into max, create one box, surrounding the whole weapon, put the collision-geometry “material” on it and export it to the Physics folder. It’s a gun we are making and not a vehicle =)

Models:

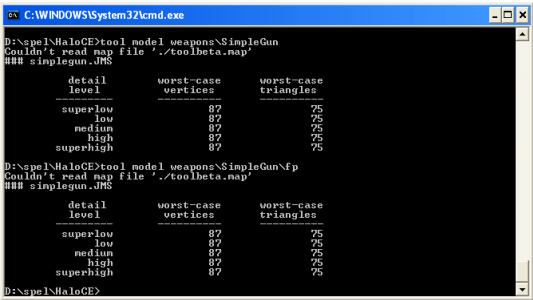
No response at all indicates it couldn’t find the jms file. Check the path and that the file really exist.

simplegun.JMS
Only this and nothing else indicates that the gun wasn’t linked to the “frame gun”.
I.e. no model to convert.

simplegun.JMS

detail level	worst-case vertices	worst-case triangles
-----	-----	-----
super low	87	75
low	87	75
medium	87	75
high	87	75
superhigh	87	75

This is a success.



Part 5 - Guerilla

There.. We’re almost done actually. Just a few things to do in Guerilla. We are gonna be lazy, but remember...

Lazy is good! =)

1. Start Guerilla
2. Open up the shader tags and set the basemap, exactly as in when doing it on a map.
3. Open up the Plasma Pistols weapon tag. (**\Halo CE\tags\weapons\plasma pistol\plasma pistol.weapon**)
4. In the menu, click *File - Save as...*
5. Navigate to **\Halo CE\tags\weapons\simplegun** and save the weapon tag as **simplegun.weapon**, but don't close the tag yet.
6. At the top of the tag in Guerilla, there are a few things we want to change:
 1. *Model* - Change this to **"weapons/simplegun/simplegun"**
 2. *Collision model* - Change this to **"weapons/simplegun/simplegun"**
7. Scroll down to *"Interface"* (a bit lower than middle in the tag).
 1. *First person model* - Change this to **"weapons/simplegun/fp/fp"**
8. Save the tag in the menu - Save
9. In the menu, click *File - New*
10. In the "New tag" window that pops up, select *"Item_collection"*
 1. *Item* - Change to **"Weapon"** and enter **"weapons/simplegun/simplegun"** in the field to the right (or browse to the tag)
11. In the menu, select *File - Save as...*
12. Save it in **"Halo CE\tags\item collections\single weapons"** as **simplegun.item_collection**

That's it!

Just add it in Sapien as all the other guns. Enjoy!



Footnote: This tutorial is more or less the most limited gun you can do. Now it's up to you to experiment with all the endless options in Guerilla. Have fun!

Download the James Model Importer v5.4 here: http://wiki.fpsgamers.se/uploads/Halo/jmi-version5_4.rar

User comments:

- view
- edit
- attach
- print
- history

Weapons Strings

Author: [Steelix B](#)

To create custom strings for your weapon, first of all read my tutorial [Here](#) on creating unicode string lists with tool

Adding your string to the string list

First re-create the hud_icon_messages.unicode_string_list (located inside the tags\ui\hud folder), which includes your custom strings for your weapon at the end of the list. DON'T overwrite old strings, otherwise you could have problems when tool builds your final map.

Also note I say strings. The strings should be *You picked up a <weapon name>*, and something like *You picked up %d rounds for <weapon name>*, replacing *<weapon name>* with the name of your weapon. *%d* is a variable that is replaced in-game to the number of round you picked up.

Changing the string of the weapon you picked up

1. Once your unicode string list has been created, and overwrites hud_icon_messages.unicode_string_list, open Guerilla and open your weapons 'weapon' tag.
2. Scroll down the tag until you find *Predicted Resource*. Below this you will find *message-index*
3. In this section change the value *message index* to the index of the string *You picked up a <weapon name>*. If you don't know the index, open the string list tag into guerilla and have a look at the number on the left of your string. That's the index you use. You don't define the index of the picking up ammo string, this is automatically selected when you define the index for picking up the weapon. However, for this to work properly the format of the string list must be *You picked up a <weapon name>* then straight after, the next string must be *You picked up %d rounds for <weapon name>*.
4. Finally save the weapon tag.

Now when you create your map file your weapon will have your custom string.

-Steelix B

- view
- edit
- attach
- print
- history

Very Basic tutorial on custom vehicles

Right here is how you get a custom vehicle to work in your maps, this is the most basic of ways but can be expanded upon. We will be creating a flying vehicle but you can apply the theory to any vehicle type just as long as it is set up correctly.

STEP 1 : Create your custom Folders

Create some new folders in your haloCE directory, these will be in bold.

go into *C:/program files/microsoft games/halo custom edition/data/* -make these folders→>**Vehicles/ufo**
You will have a directory like this.

C:/program files/microsoft games/halo custom edition/data/Vehicles/ufo

*Within the **ufo** folder create these folders **Models, Physics, Animations and Bitmaps***

When you have added all the folders you can then go into 3dmax or gmax and create your spaceship (or whatever), first thing you will have to do is save the bitmaps you use for your model in the ufo/bitmaps folder, when you have your finished model save it as a 3dmax file in a folder of your choice(just in case you mess it up later).

STEP 2 : How to add markers to your model

Next you must add markers to your vehicle, because we are going to overwrite the banshee we will need to open [Guerilla](#) and then open the **banshee_mp.vehicle** tag, then open the gbx model within the banshee tag(right at the top), within the gbx model scroll down until you see the second markers submenu (about half-way down), click the rolldown menu within the markers (first should be cockpit light) and copy all the names EXACTLY as they are to a notepad or piece of paper.

Note: Or use James model importer, to import it the models and markers into 3ds. /Z

Next go back to your 3dmax model and create a box about 2×2x2 pixels (any size will do but small is better)

and name it **#cockpit light**, you must do this for all of the markers you have written down, when you have made all the markers on the list place them on your image where you think they would be (eg. place the **#cockpit light** within the cockpit and primary trigger is for the gun ect).

When all the markers are in place link your vehicle to the frame (call it frame ufo for ease) and link the markers to your vehicle (this isn't the proper way to do it but we will not have any real animations) then save it as a 3dmax file (save all the time, it will keep you ahead of the game). Use the Exporter plugin to save your model as a .JMS file and save it in the **data/vehicles/ufo/models** as **ufo** .

STEP 3 : Exporting the model

Next you have to convert the model into a gbx model and to do this you will have to open [Tool](#) and type.

```
Tool.exe model vehicles\ufo
```

You may find on the first run of the tool.exe model command that you will be asked for shader types, (these are just your materials from 3dmax so most of them will probably be shader enviornment) just pick shader types for each and run the tool.exe model command again. If all goes smoothly you should have a new folder within your tags/vehicles folder as ufo and within that there should be a gbx model called **ufo**, if all that is correct we will quickly save an anim so your vehicle will work, to do this just go back into 3dmax and export the model to **Animations** as a .JMA file. (I know there is no animation but you will need an animation graph that fits your vehicle)

STEP 4 : Creating and exporting the collision geometry

For the collision geometry we will need to find out what markers you will need (the collision geometry needs different markers to do with physics) first goto [Guerilla](#) and open the Banshee's collision geometry and follow step 2 (but for the collision geometry).go back to 3dmax and re-name all the markers you had for the gbx model in the names of the markers from the banshee collision geometry (you may have to add/delete some markers), again place these where you think they will go. Link these to your model and open the materials menu (M is the shortcut) and change the name of your material group to collision geometry. Dont forget to Save your 3dmax CG model (collision geometry).

Next you must export the model into the **Physics** folder within your **ufo** folder as a JMS file.

STEP 5 : Converting and creating Animation/collision geometry and physics tags

You must now open tool and type in the following commands..

(to create your bitmaps)

```
Tool.exe bitmaps vehicles\ufo\bitmaps
```

(to create your animations)

Tool.exe animations vehicles\ufo

(to create your physics, this is very important)

Tool.exe physics vehicles\ufo

To check whether you have done it correctly you should goto your Tags/vehicles/ufo folder and there should be 4 files and a bitmap folder, if you are missing any of these you may have to do it again.

(Note: now is a good time to set up your shaders)

STEP 6 : Setting up your anim,collision-geometry and physics tags

First you must open the Original **Banshee_mp.vehicle** and save it in your **tags/vehicles/ufo** (leave it named Banshee), next open it up and replace the gbx,animation graph,collision geometry and physics tags with your corresponding ufo tags. Next open the original banshee anim **Banshee.animation_graph** and your custom anim **ufo.animation_graph** then add all the parameters that are in the banshee anim to your ufo anim (leave the actual animations as NONE) and save your anim tag. Next you must set your physics to work properly, to do this you must do pretty much the same as the anims but you must not change any of the vehicles actual physics.

These are what you must change in the first menu.

radius to -1 , moment scale to 1

Leave the matrix and copy all the values from the banshee powered mass points/mass points to your ufo physics tag and assign your markers the same as the banshees (eg change your ufo hullback mass points so it is the same as the banshees), also be aware that the radius of each mass point will become collision geometry so be careful when choosing how big or small they are.

Lastly you should rename your original **Banshee_mp.vehicle** to **Banshee_mp.vehicle.backup** and then save your modified vehicle tag in your new vehicle folder. For example save as, then browse to your vehicles\UFO folder and save as UFO. Then open guerilla, go to the globals folder and open the globals file. Scroll down about 3/4 and you'll see Vehicles. Hit the add tag and then go to your vehicles\UFO folder and select the UFO vehicle. Save it and now you will have your ufo without overwriting any other vehicles.

Then all that needs to be done is place your vehicle into sapien

Hopefully thats it, you can mess about with the settings and physics to get the final result you want but as it is it will fly like a banshee.

Any problems let me know, its pretty much how I did it at first and is a safe bet to get a custom vehicle in for the first time, but also be aware that you may have to set your vehicle bounding radius to its probable size (eg

if your vehicle is about the same size as a hog then just use roughly the same value and add 1 or two), also you may have to offset your vehicle if it is’nt placed properly. This is just one way of doing it and there are probably other ways but this is just the easiest.

Tutorial by

[AoAAxeman?](#)

also known as The Axeman/Axl/Anglegrinder/A BAD MAN 😊

- view
- edit
- attach
- print
- history

The Huge List of 3D Modeling Resources

3DS MAX TUTS

- http://www.discreet.com/education/tutorials_3dsmax.html
- <http://www.3dbuzz.com/> - one of the best!
- <http://www.planetquake.com/polycount/> - great place for job openings, quality user-made models for fps games
- http://sv1.3dbuzz.com/vbforum/unr_main.php

Other Modelling sites

- <http://www.cgchannel.com>
- <http://www.3dspline.com/>
- <http://www.3dkingdom.com/>
- <http://www.http3dluvr.com/>
- <http://www.3dbuzz.com/>
- <http://www.3d-palace.com/>
- <http://www.3dluvr.com/content/>
- <http://www.3dmaxer.dk/>
- <http://www.3dcafe.com/asp/tutorial.asp>
- <http://www.3dkingdom.org/index.php>
- <http://www.maxforums.org/>
- <http://www.3dtotal.com/>
- <http://www.3dlinks.com/index.cfm>
- <http://www.fineart.sk/>
- <http://www.artofillusion.org/index>
- <http://www.raz.ro/tutorials/masina/car1.htm>
- <http://www.chilliweb.co.uk/chilliskinner/>
- <http://www.comet-cartoons.com/toons/>
- <http://www.computerarts.co.uk/>
- <http://www.digitalart.org/>
- <http://www.flay.com/start.cfm>

- <http://planetunreal.com/identitycrisis/tutorials/skinning.shtml>
- <http://www.3dtotal.com/ffa/tutorials/max/joanofarc/joanmenu.asp>
- <http://mr2k.3dvvf.com/>
- <http://www.planetquake.com/polycount/>
- http://www.planetquake.com/polycount/cottages/qbranch/ref_links.shtml
- <http://www3.sk.sympatico.ca/kelbeau/>
- <http://www.turbosquid.com/>
- <http://www.thehobbitguy.com/tutorials/polymodeling/>
- <http://molkien.halocity.org/>
- <http://www.3ducation.com>
- http://www.neilblevins.com/cg_resources/cg_resources.htm
- http://www.neilblevins.com/cg_education/cg_education.htm
- <http://www.virtualpartners.com/3DMAX6freeTutorialLinks.htm>
- <http://www.cgchannel.com>
- <http://www.cgarchitect.com/resources/default.asp>
- <http://www.3dkingdom.org/>
- <http://multimedia.marin.cc.ca.us/index1.htm>
- <http://www.paulneale.com/>
- <http://www.scriptspot.com/start.htm>
- <http://www.footools.com/index.html>
- <http://www.maxplugins.de/max4.php>
- <http://www.3dlinks.com>
- <http://gfxcentral.com/bobo/>
- <http://raph.com/3dartists/tutorials/t-3dsmax.html>
- <http://www.3dcafe.com/asp/freestuff.asp>
- <http://www4.discreet.com/combustion/combustion.php?id=138>
- <http://www4.discreet.com/combustion/combustion.php?id=207>
- <http://www.creativecow.net/articles/combustion.htm>
- http://www.fxguide.com/modules.php?name=fxtips&rop=fxtip_list&catid=8
- <http://www.fxguide.com/forums.html>
- <http://www.creativecow.net/index.php?forumid=14>
- <http://max3d.3dluvr.com>
- <http://3dup.com/>
- <http://www.3drender.com>
- <http://www.vfxpro.com>
- <http://www.cyberedge.com>
- <http://www.3dtotal.com/>
- <http://www.dvgarage.com>
- <http://www.3dluvr.com/pepeland/>
- <http://www.trinity3d.com/>
- <http://www.oken3d.com/index2.html>
- <http://www.righthemisphere.com>
- <http://www.sfsig.org/>
- <http://www.siggraph.org/>
- <http://www.id8media.com/>

- <http://www.ieee.org/portal/index.jsp>
- <http://www.highend3d.com/>
- <http://www.blueroomgallery.org>
- <http://www.ylem.org/>
- <http://www.skunkstudios.com/>
- <http://www.directordev.com/>

Wow, that is one freakin' huge list :P -E Vi L Da VE

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

[Halo](#) /

Create Waterfall

- view
- edit
- attach
- print
- history

Ok, here’s how to create an a30 Halo-style waterfall.

1. Create the mesh for the surface of your waterfall. You can make it more than one poly wide, if you want the surface of the waterfall to curve in plan, but 1 poly wide is fine. Make sure that it has enough vertical divisions to make it’s polys are roughly square.
2. UVW map it planar to the front
3. Use “timberland_waterfall!” as the material name. (make sure that you copy “timberland_waterfall.shader_transparent_chicago_extended” to the shaders folder of your map.
4. UVW unwrap your waterfall mesh, and adjust the mapping so that the polygons at the top of the waterfall taller than the poylgons at the bottom. This makes the scrolling texture accelerate as it moves down the mesh. (This is a bit hard to describe in text)

- That should be it

For more deep and varied waterfalls, copy the mesh inront or behind the original, and move the entire UVW to a different part of the texture - this will give the impression of more ‘layers’ in the waterfall.

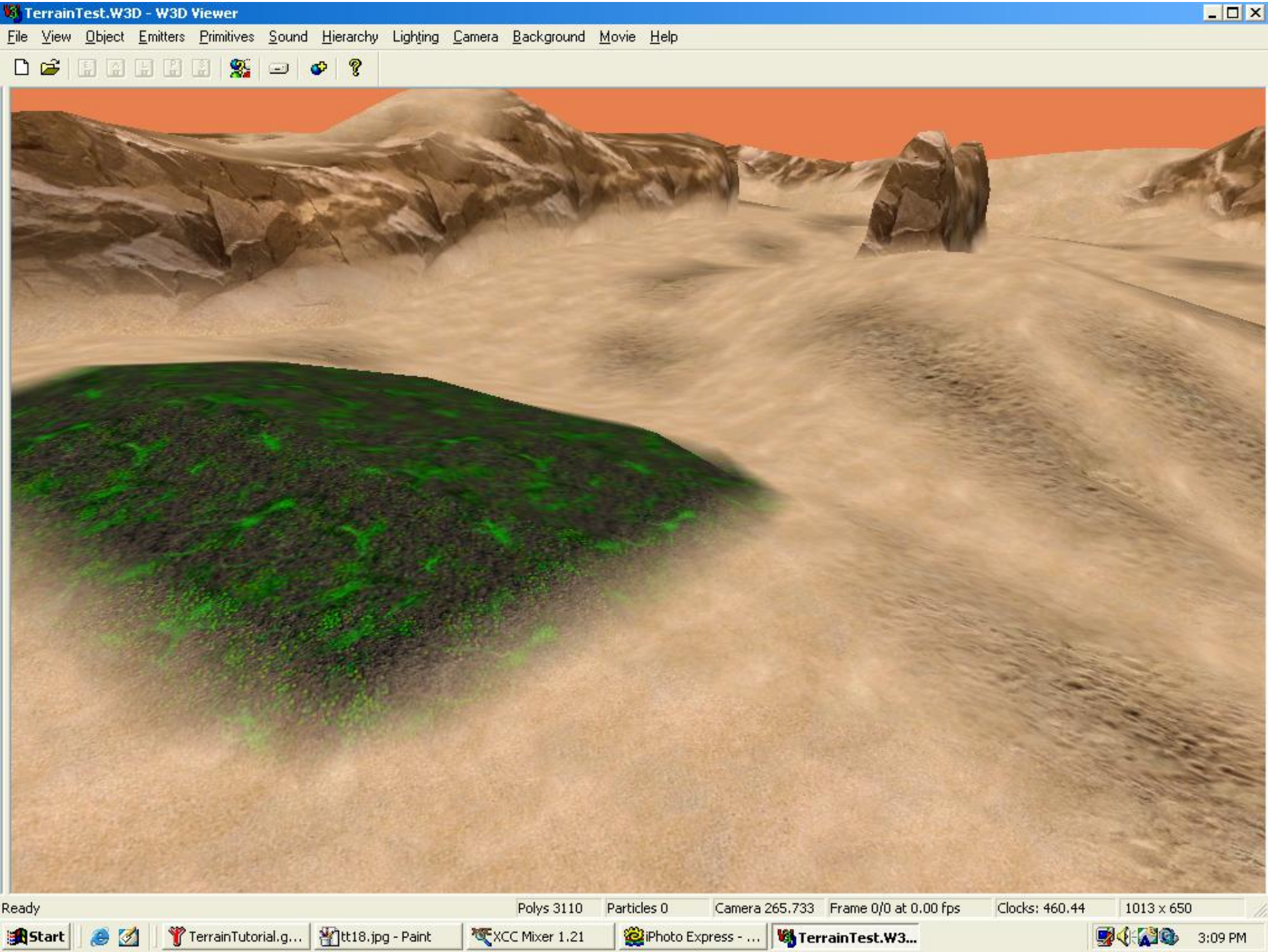
- view
- edit
- attach
- print
- history

Creating Basic Terrain

By [Some Rhino](#)

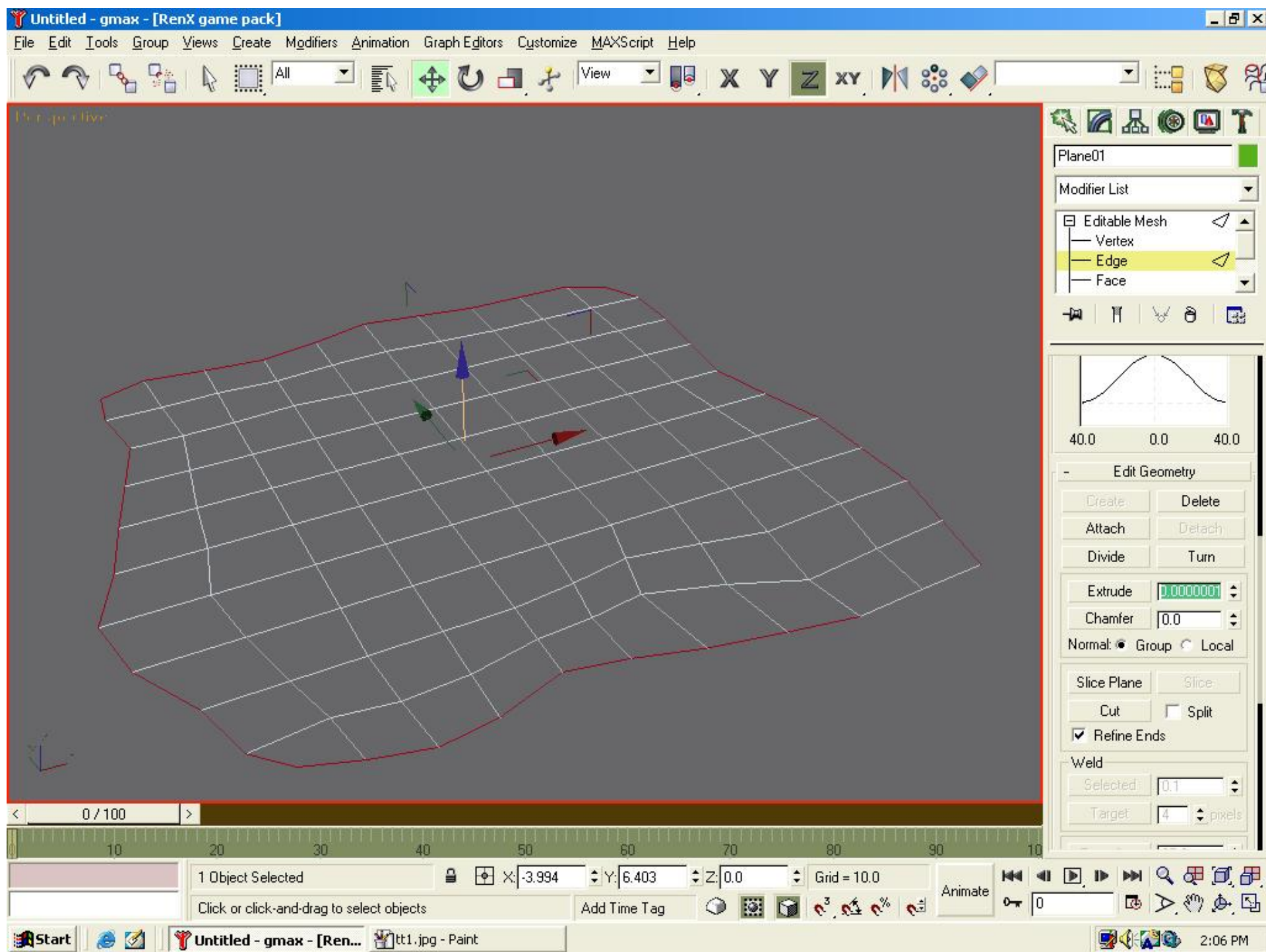
[Some Rhino@renevo.com](#)

Note: This tutorial was originally written for the [W3D](#) engine. I cut out everything that isn't relevant to the Halo engine and have added some that is. Please don't be confused if some of the screenshots don't match up exactly with yours.

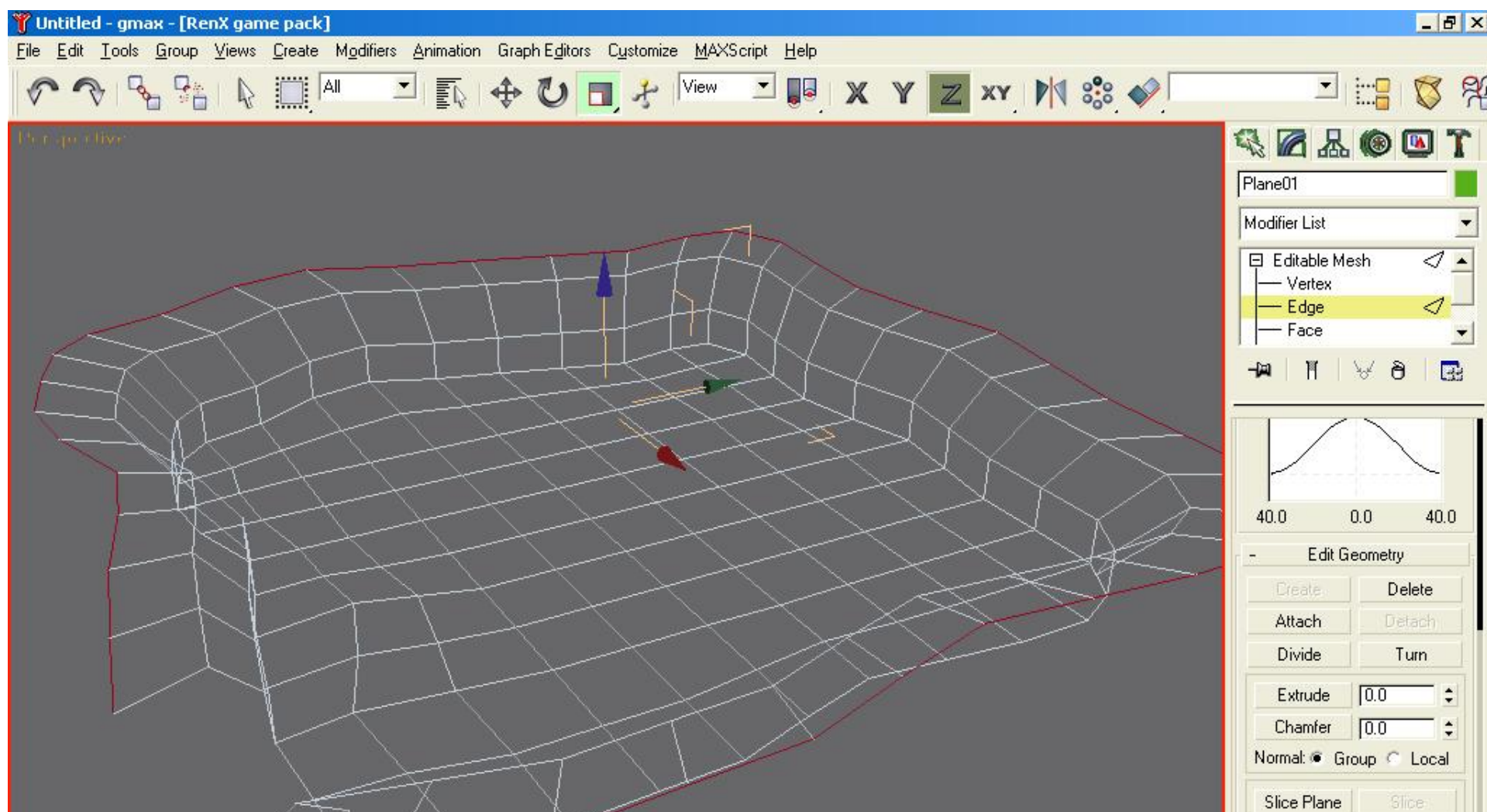


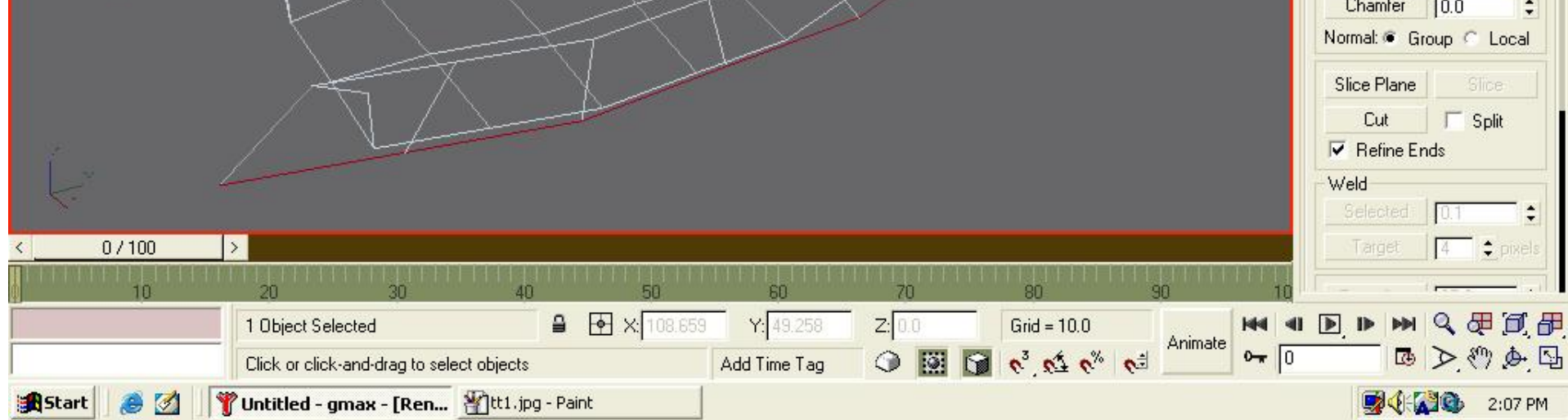
Creating terrain seems like a cumbersome job is you envision aligning planes and boxes only to yield flat ground and blocky mountains, but if you know how to use Max properly, it can be done neatly and in a relatively short amount of time. Note however that this only covers the basics of creating the terrain meshes, nothing advanced. Some more advanced techniques will be available in another upcoming tutorial. I also am assuming you know how to navigate gMax, and how to do apply basic textures. Texturing tutorials are aplenty.

Start off by creating a plane. Convert it to an editable mesh (right click) and drag the edge vertices around using soft selection (found by expanding its tab under Editable Mesh. Round it out a bit to form the base ground. Select the outer edges of the mesh.

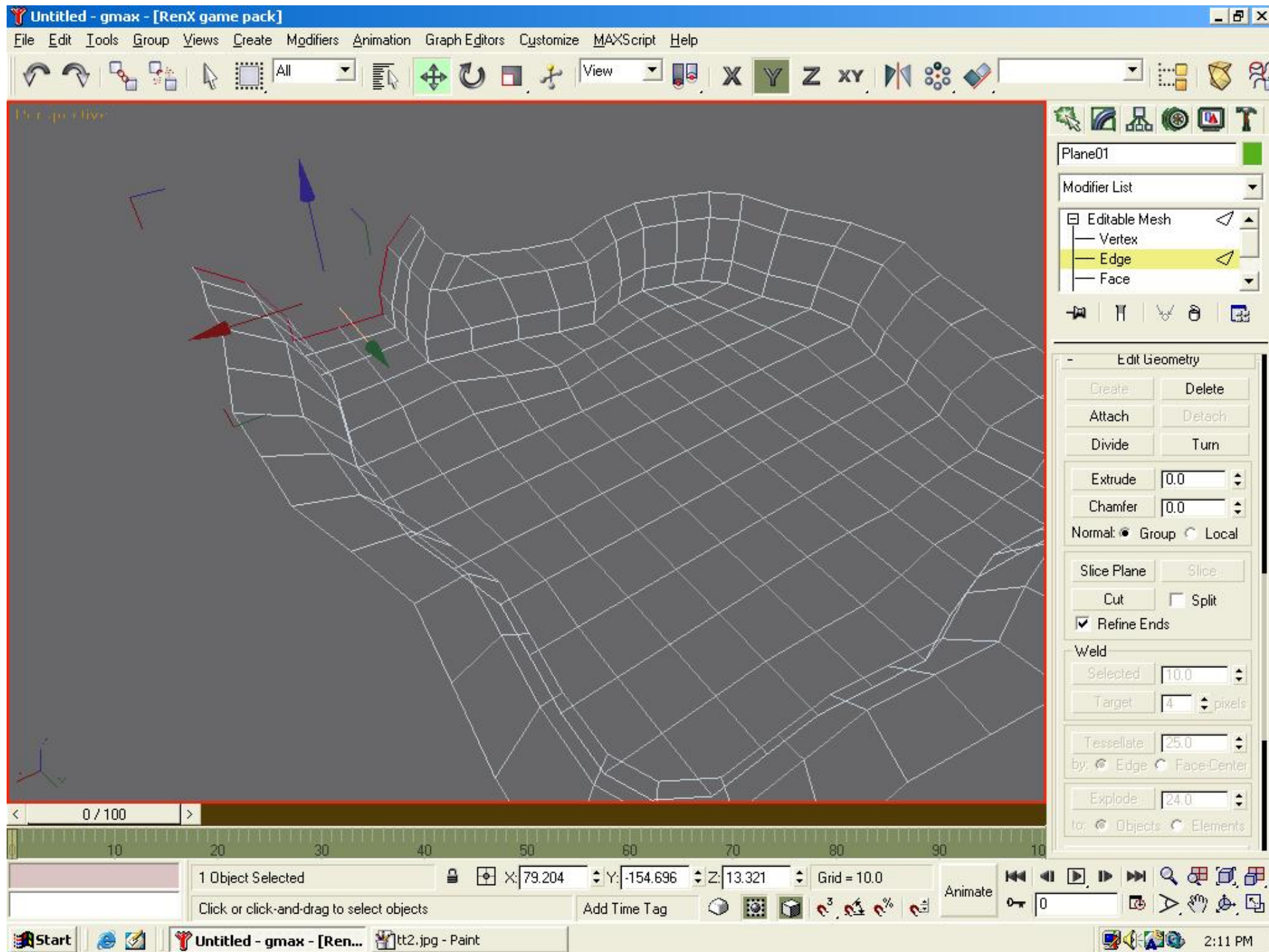


Scroll down to Extrude on the sidebar and set the value to 0.00000001 or similar to keep the extrusion from going anywhere. Hit the Extrude button, and drag the edges vertically. Hit it again, and repeat. Use the scale tool to make the edges curve outward. Do this one or two more times until the edges cannot be seen from the ground below.

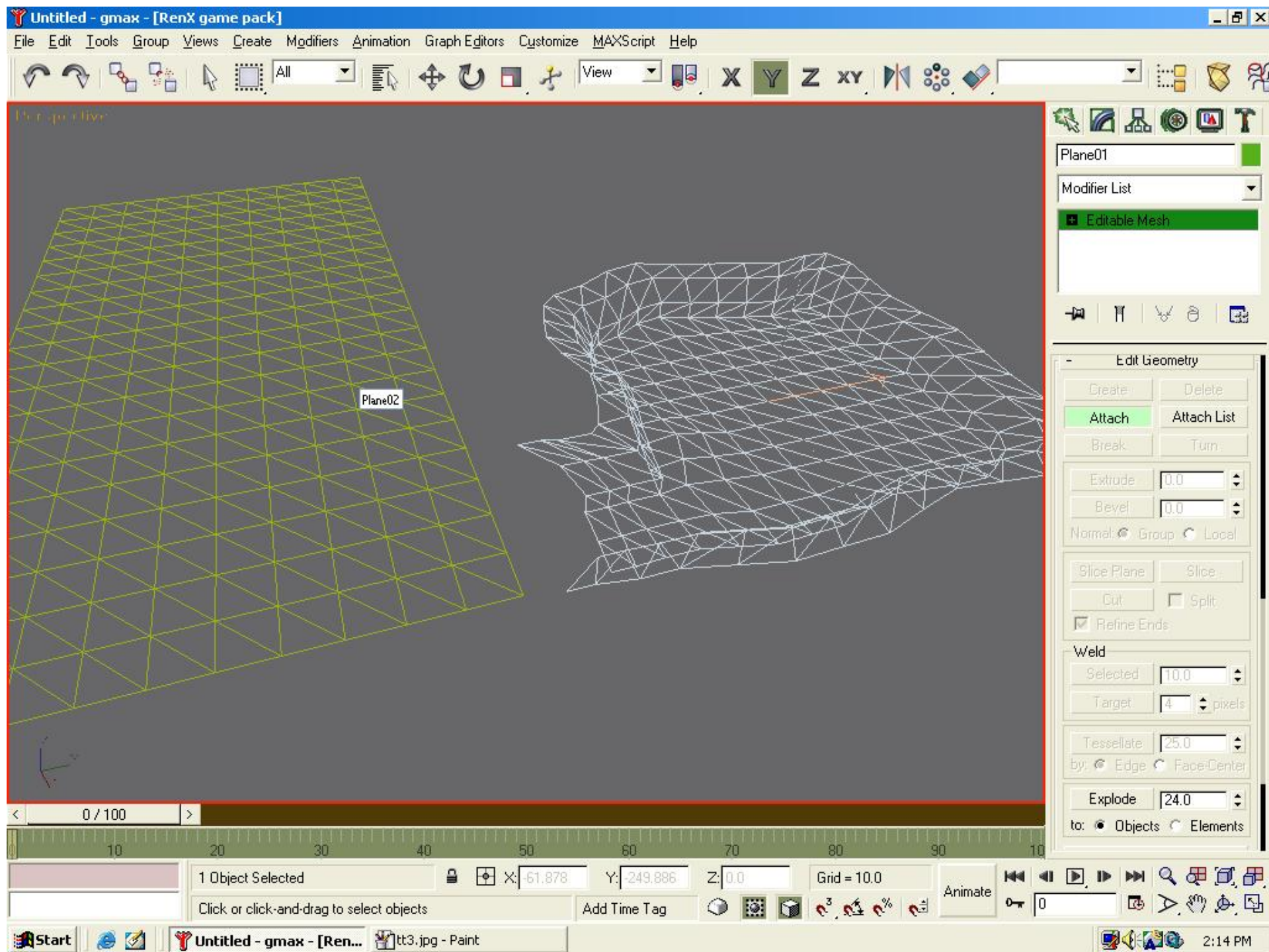




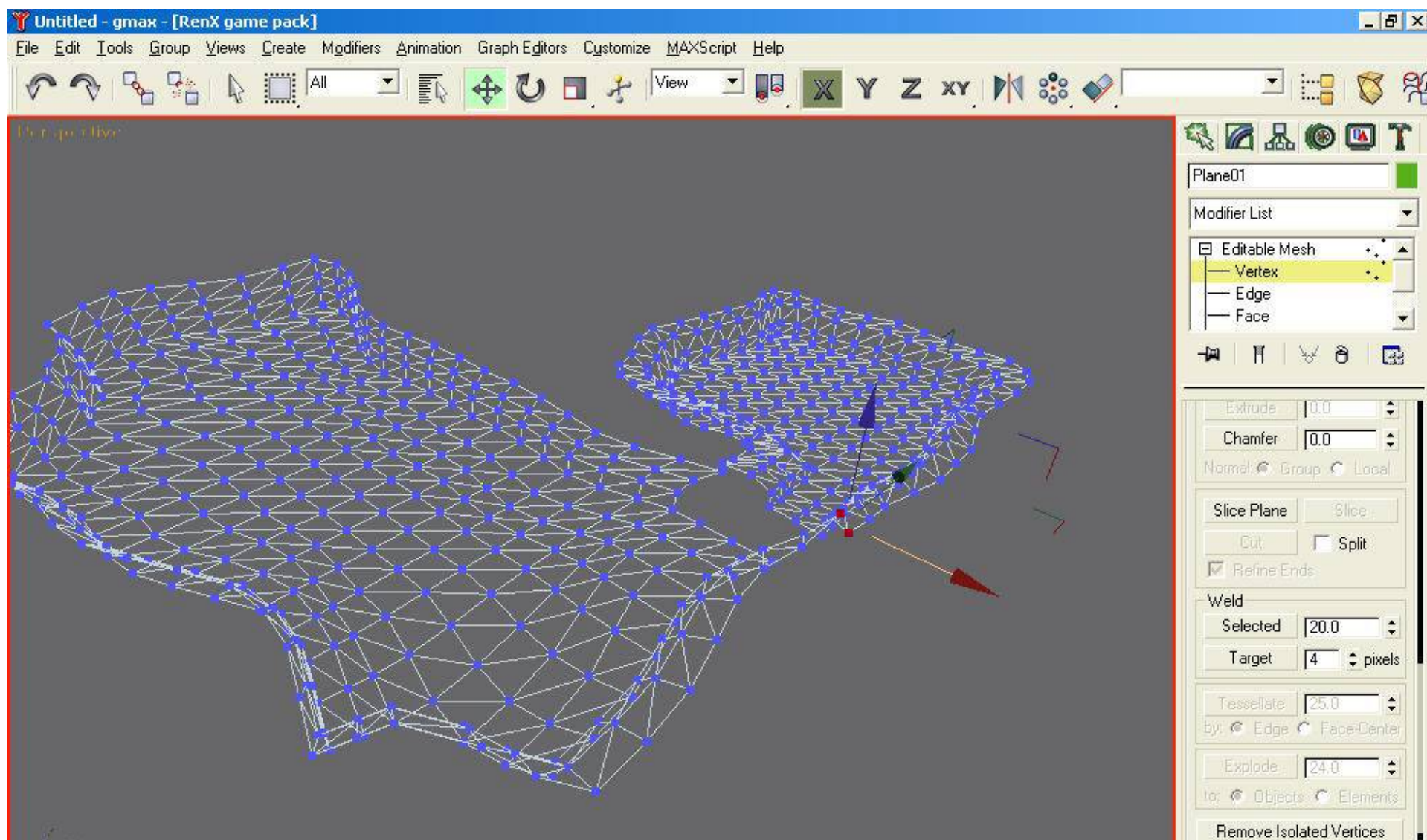
If you don't have one already, create an opening by deleting some polygons (select them and hit delete on your keyboard.) Drag a few of the vertices to make an outer angle. Now select the edges on the opening and extrude some more to create an exit to the area.

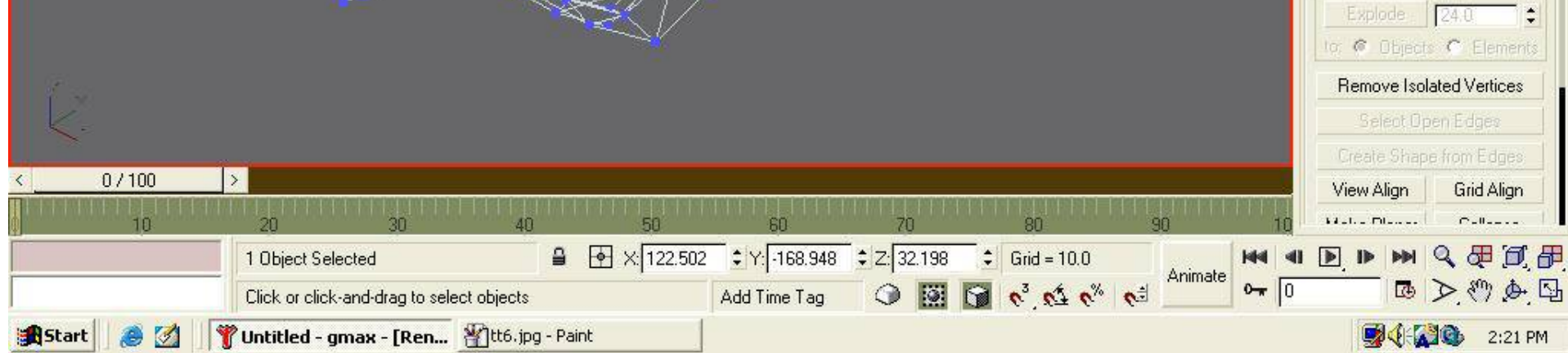


Now create a larger plane, which you will make into the main area of the map. Attach the meshes together using editable mesh.

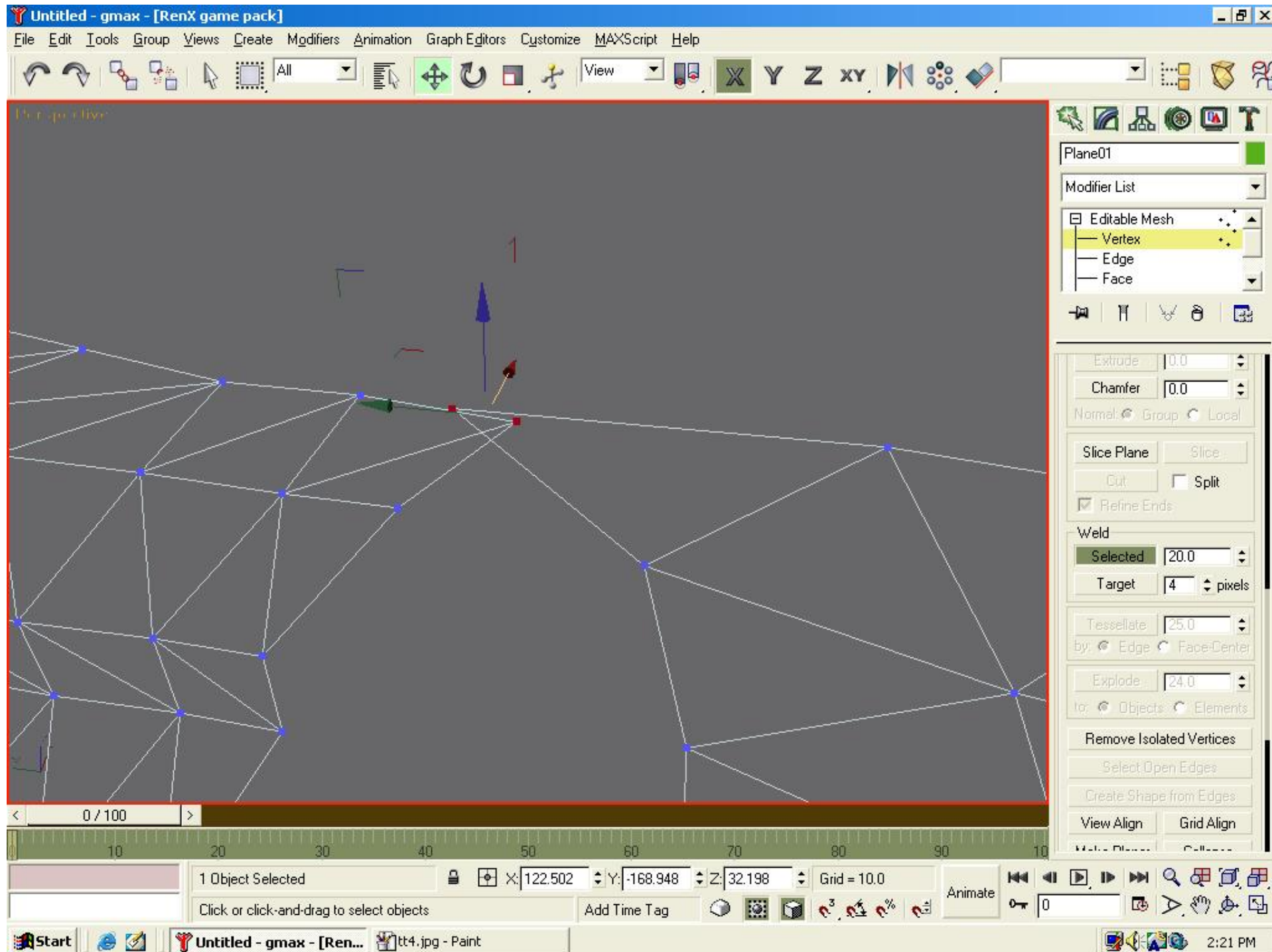


Shape the new plane into more terrain using soft selection on the edge vertices and extruding some more.

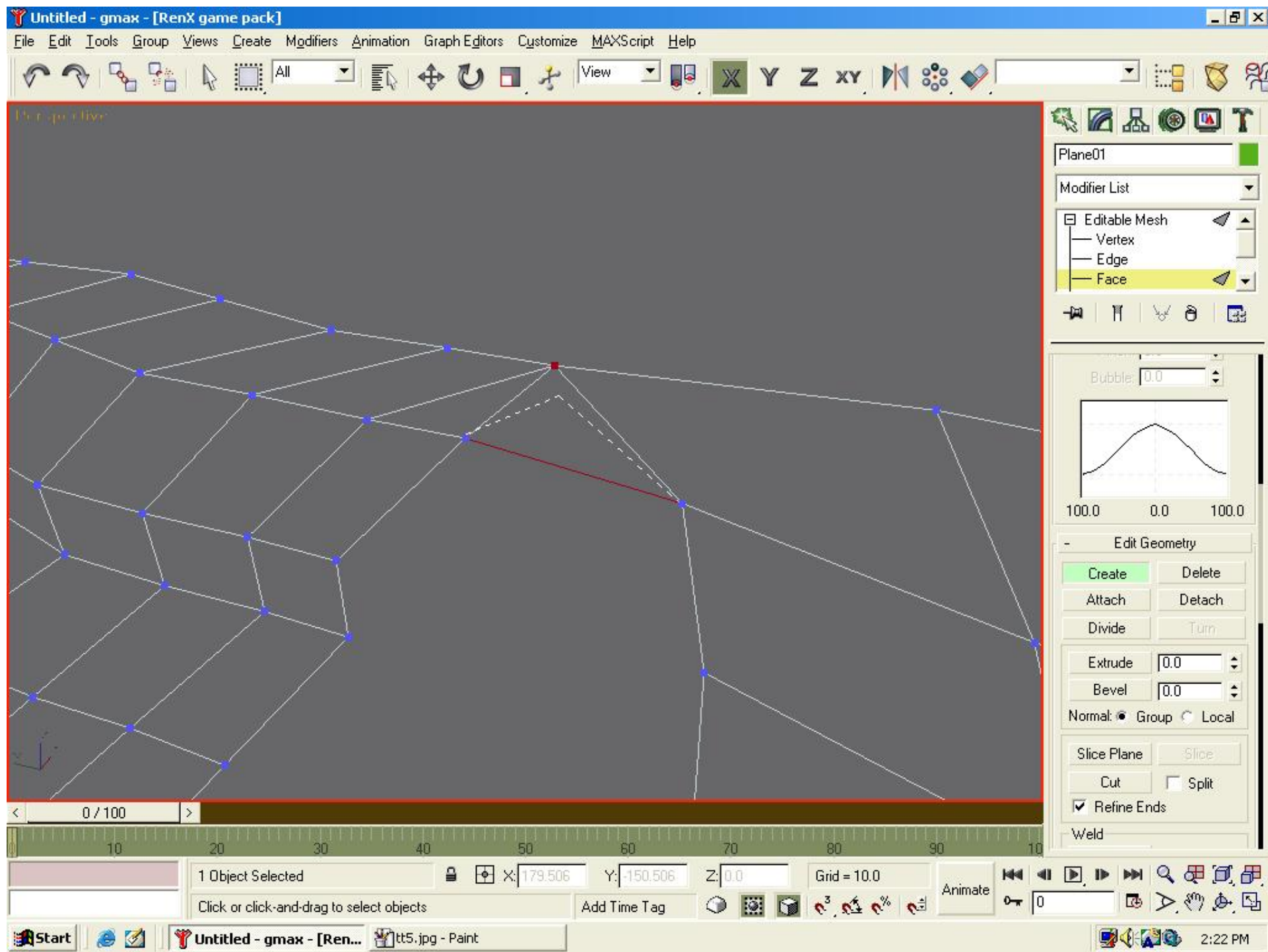




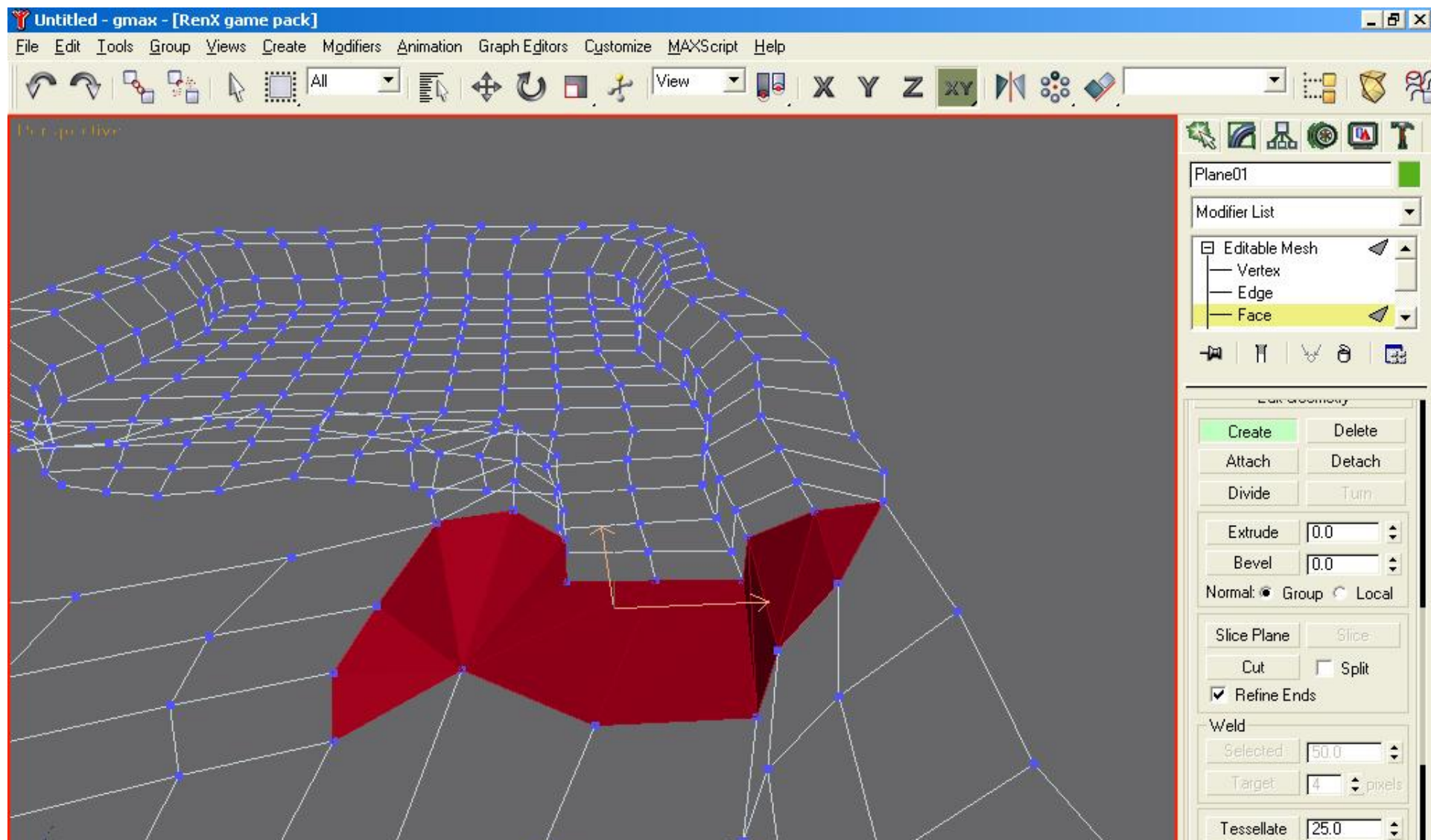
Now it's time to connect the two objects together with polygons. Select vertices that are relatively close to each other and hit the collapse button under editable mesh. The vertices will weld, causing the mesh to become a single element.

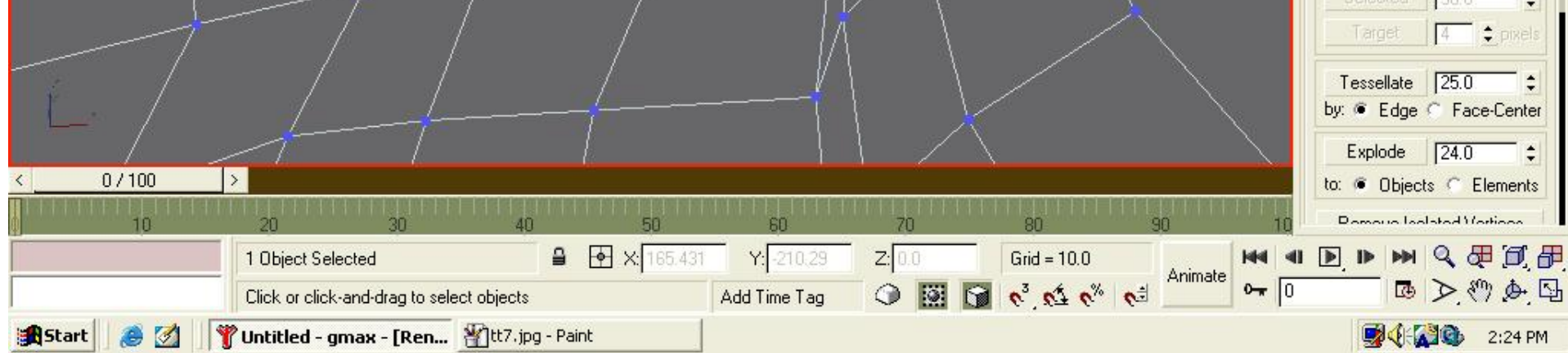


Hit the Face branch on the Editable Mesh tree and hit the Create button. Build face by clicking on three vertices consecutively in Counter-Clockwise order. Hitting them clockwise will build a backface.

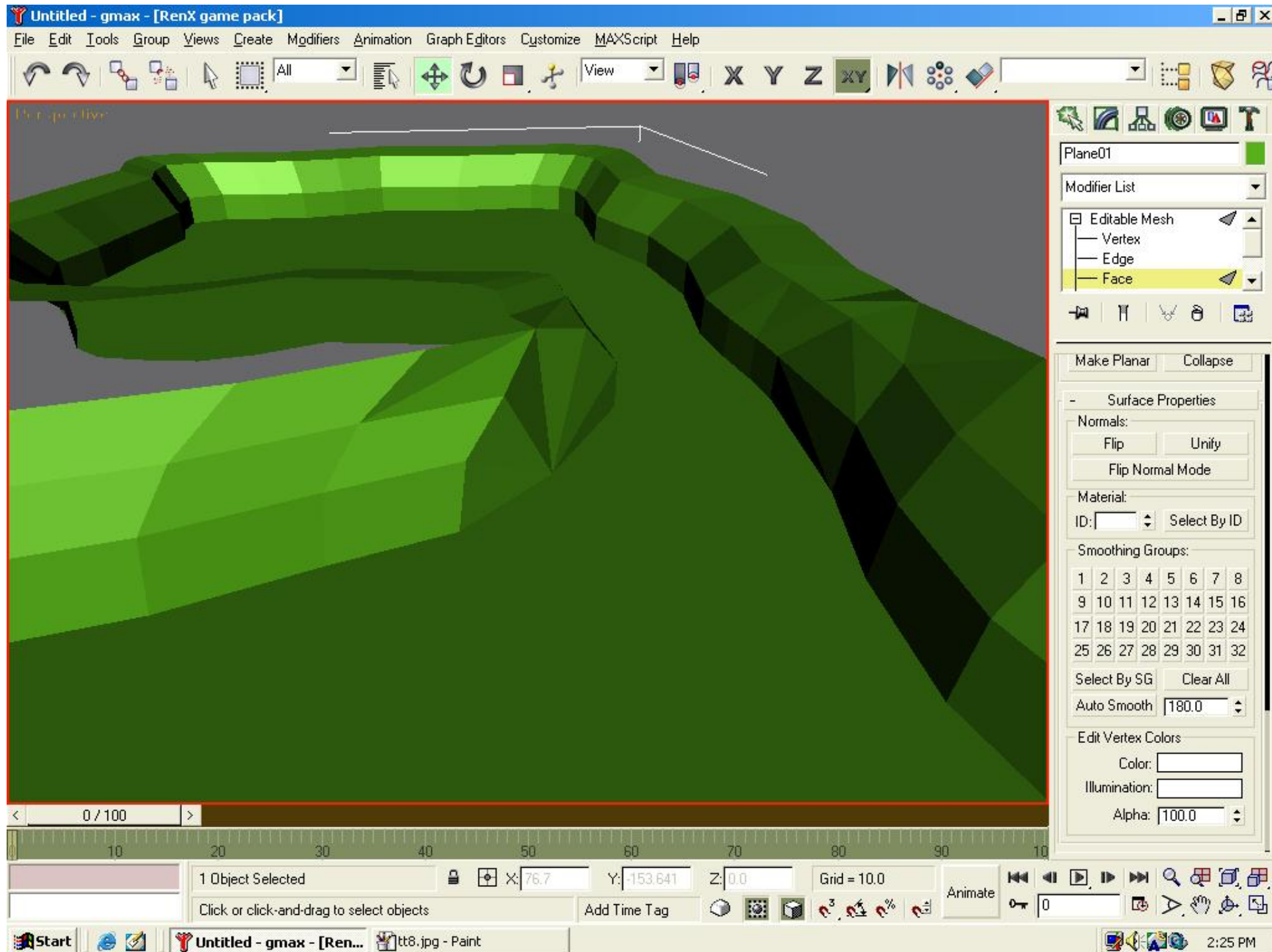


Build more faces to completely connect the objects,

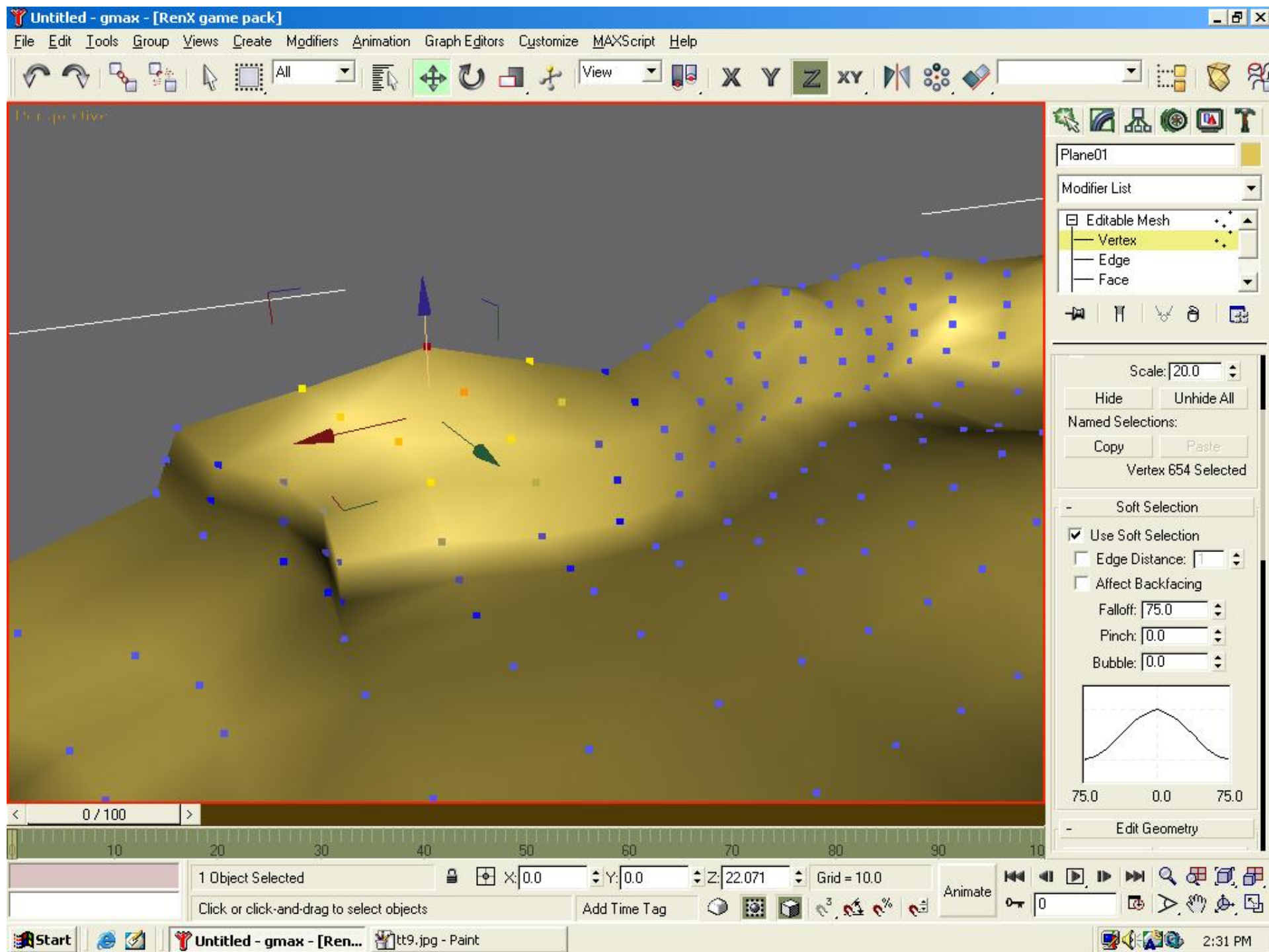




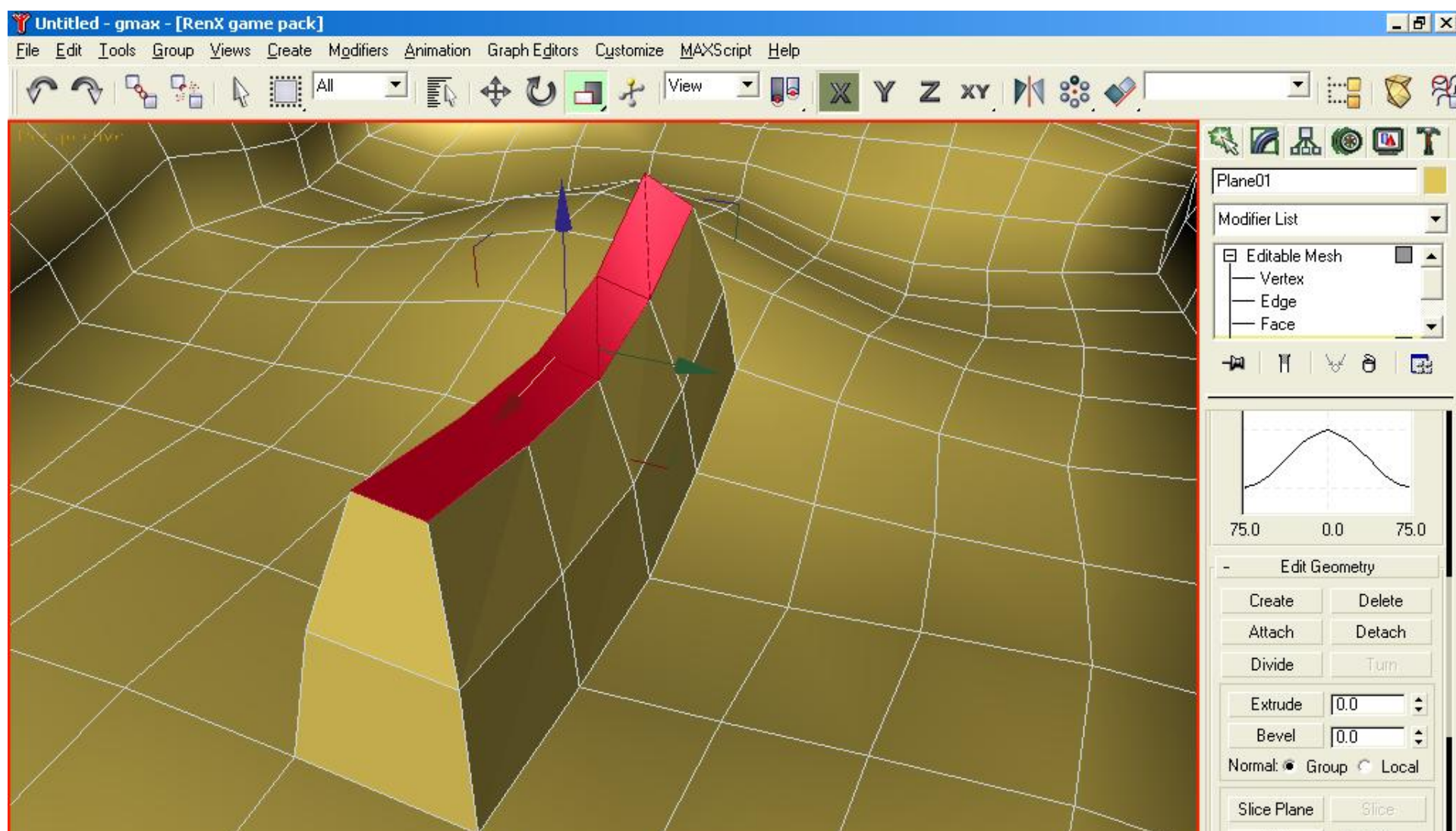
If you switch your view to smooth + highlights, you'll notice that your level will be very faceted and blocky looking. Remedy this by selecting all the faces and scrolling down to the Auto Smooth button. Set the value to 180 and click it.



Now, use soft selection to drag vertices smoothly on your mountains and ground to give the terrain a more realistic shape.

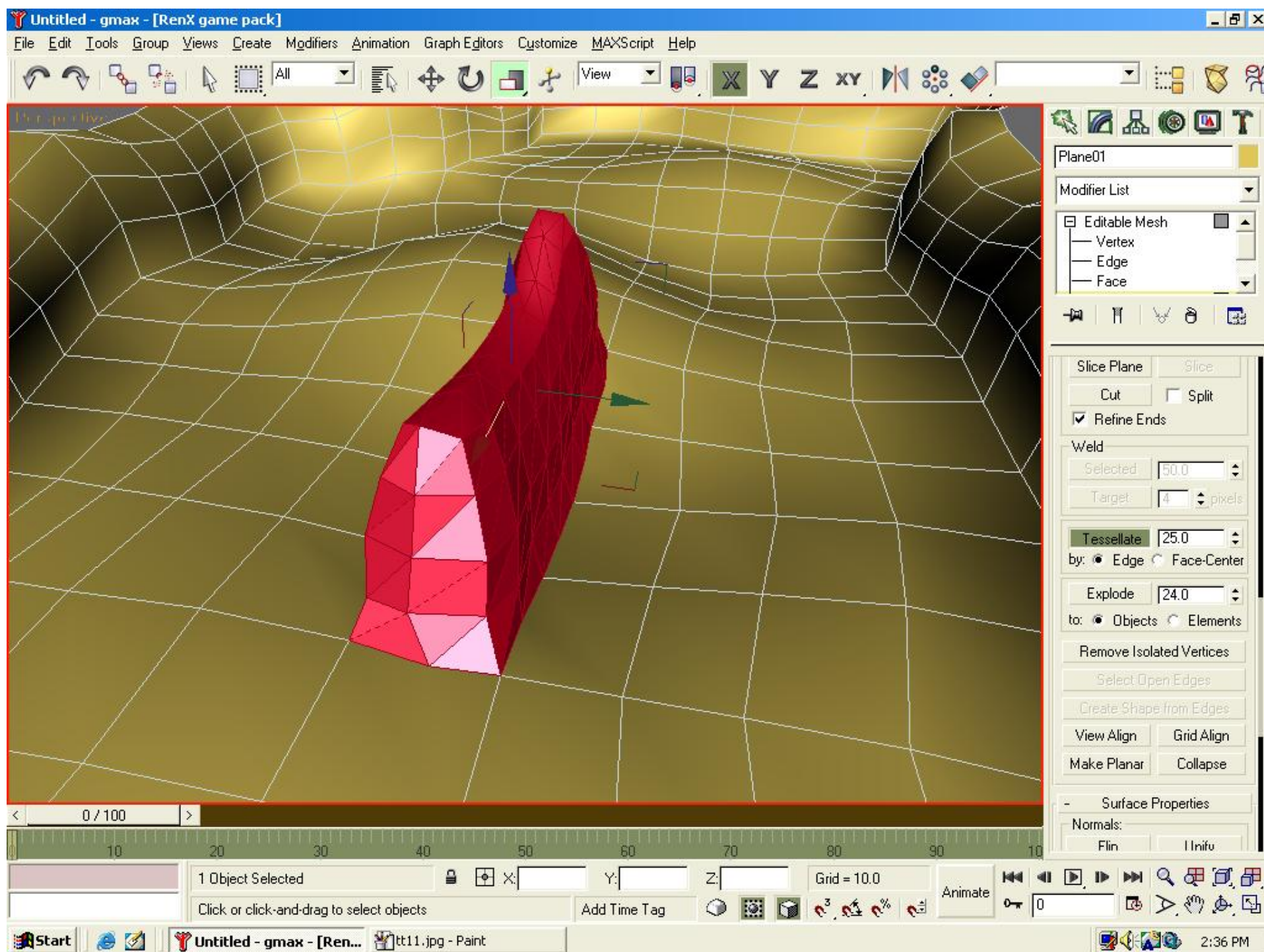


Aircraftkiller taught me an easy technique for creating ridges. Select a few of your ground polygons and extrude them up a couple times scaling them down a bit each time to give the ridge a tapered look.

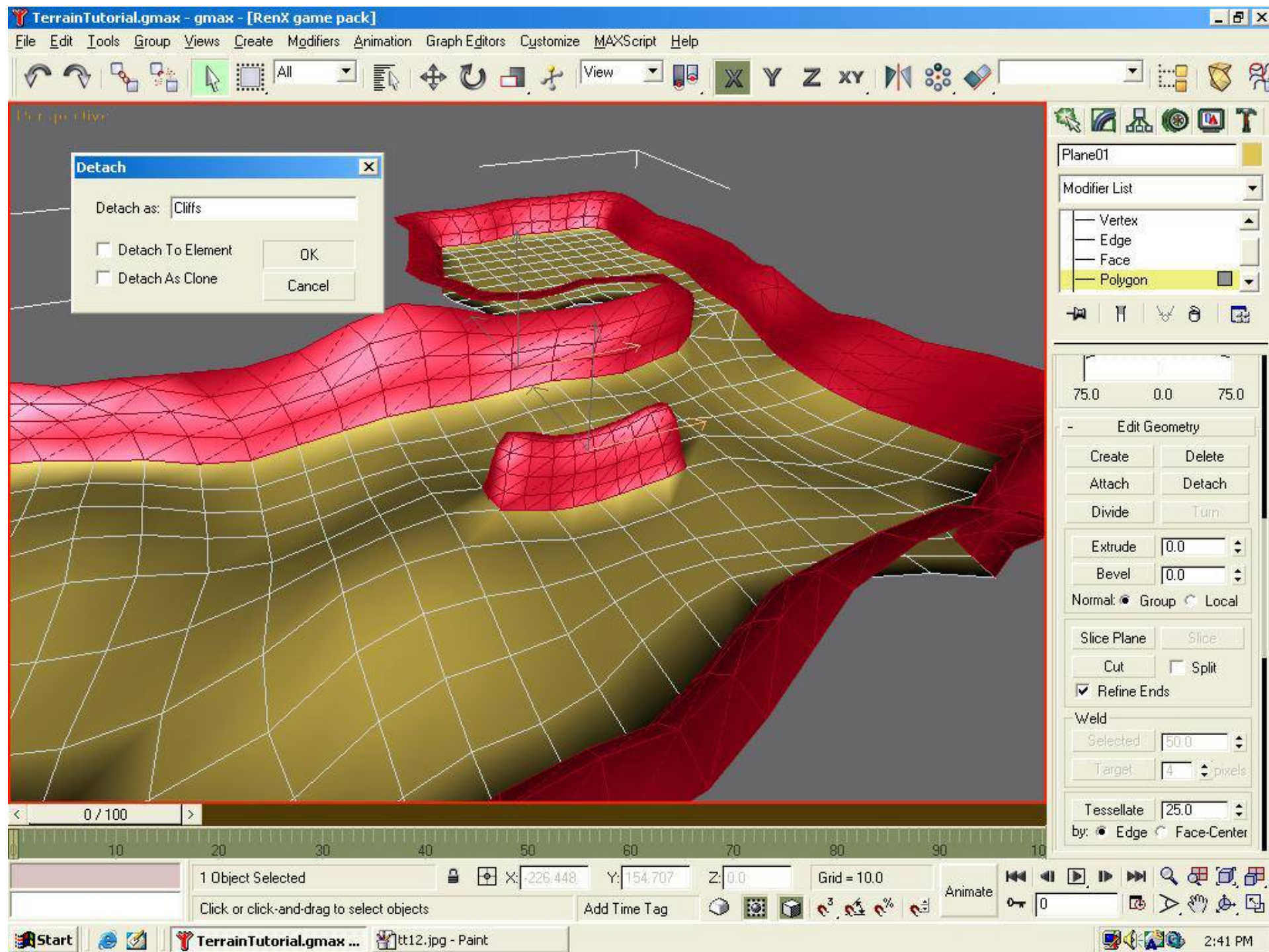




Now, select all the polygons that make up the ridge and find the Tesselate button. Leave it as Edge and 25, and just hit the button. It will smooth out the ridge with more polygons. Again, you need to run the Auto Smooth on your terrain.



Now it's time to divide up your meshes. Select the mountain polygons. I selected mine by going to the front view and selecting the higher polygons. Clean up by holding down the Ctrl key on your keyboard and selecting/deselecting polygons by clicking on them. Hit the detach button on the sidebar, name it, and hit OK.



You should have some nicely modelled terrain when you are done. Your design will probably be more complex than the one used in this tutorial. When you have the terrain shaped as you want, it's time to move onto the next tutorial about setting up the materials and shaders. We can't do this until the SDK comes out, however. Expect the tutorial soon after the Editing Kit comes out.

-Some Rhino

SomeRhino@renevo.com

- view
- edit
- attach
- print
- history

REQUIREMENTS: 3D Max, Photoshop.

ACKNOWLEDGEMENTS: I originally learned this technique from a tutorial by Hailfire on [Halomods](#), so great praise & thanks go to him for sorting out my terrain modelling life, so to speak!!

DESCRIPTION:

Fed up with making terrain by manually shifting vertex after vertex? Using the Displace modifier in 3DMax, you can make highly detailed terrains in a short space of time that look more realistic, more ‘organic’ than the blocky rendition we all got the first time we attempted it in MAX.

Using this technique, you create a heightmap in a program such as Photoshop, then import it to Max to create displaced terrain according to the map. The Heightmap is just a greyscaled bitmap image, where white will eventually translate to high ground and black to low ground.

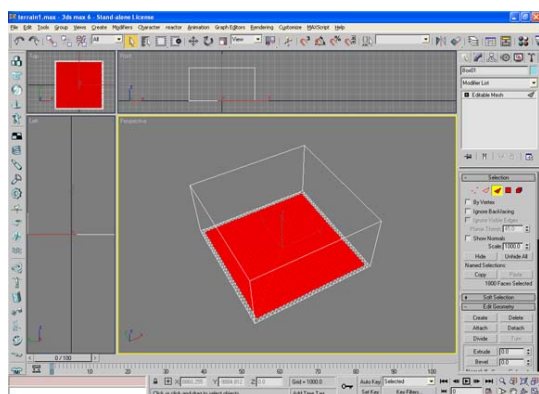
STEP BY STEP INSTRUCTIONS:

1.) Open up MAX. Create a box in the top window, drag it out and give it a length/width of 10,000 each, and a height of 5,000. Right click on the box and select convert to editable mesh. Now, in the edit mesh modifier section, select face selection mode (the triangle - third one along). Make sure ignore backfacing is off, as this option disables the selection of faces that are facing away from the viewer. Drag the selection lasso around the whole box, so all faces are selected. Go down to surface properties in the edit mesh dialog, and hit flip normals. This makes the faces of the box, that were all originally facing outside, to face inside, so that they will be rendered once you are inside the level.

Note: If you want to add a face to a selection, press control while selecting. If you want to remove a face from selection, press alt while selecting. This is really useful if you want to select, say, all faces except the top & sides, just select the entire whole box (face selection mode), then hold alt and then drag the selection lasso over the top & sides.

2.) Deselect all the faces, and go into the top viewport. Make sure ignore backfacing is now ON, and select the two faces that appear to make up the large flat surface of the box. If you did it right, only the bottom two faces should be selected. You can check this by looking in the edit mesh dialog and see the number of faces selected, or by rotating the perspective view. Finally, with the two ground faces selected, in surface properties (edit mesh dialog), go down to tessellate. Enter a value of 0, and hit tessellate 5 times, until the number of faces (written just under the select vertex/edge/triangle/polygon buttons) is 4096. No more than this, or your map will chug along at 1 FPS :S

3.) Now the box is ready for adding the terrain. With the bottom 4096 faces selected, carefully deselect the row of faces at the very edge, by dragging the lasso across the outside row of faces with ALT pressed down. Now, in surface properties, go to material ID, and set a material ID of 2 to these inner faces. This will help with applying the modifier later. This is how it should look so far:



(Click on thumbnail to enlarge)

4.) Open up Photoshop, and create a new canvas with an arbitrary size, like 500×500. Make sure the canvas is filled in white. For this example, we are going to make a crater like environment, so go to the ellipse tool (hold down the mouse button over the rectangle tool, and go down to ellipse), make black your foreground colour, and drag out a large black ellipse on the page. Make it slightly elliptical, rather than plain circular. Once you've done that, go to filters→blur→gaussian blur, enter a value of 5 pixels, and hit ok. It should now look something like this:



5.) Right then, in photoshop, find the smudge tool (hold down mouse over blur tool and select smudge). Select a radius of 25, a strength of 18% and make sure u have a soft edged brush selected. With the smudge tool, run around the edge of the ellipse, making small inward strokes from white to black (dont go back and forth) at irregular intervals (about 1–2 cm spacing on the screen) around the crater. Once you've gone all the way around once, make the radius, 18, and the strength something like 30%. Repeat smudging the outside of the crater from outside to inside at irregular intervals. Finally, select a smaller brush, 12 or so, and a strength of

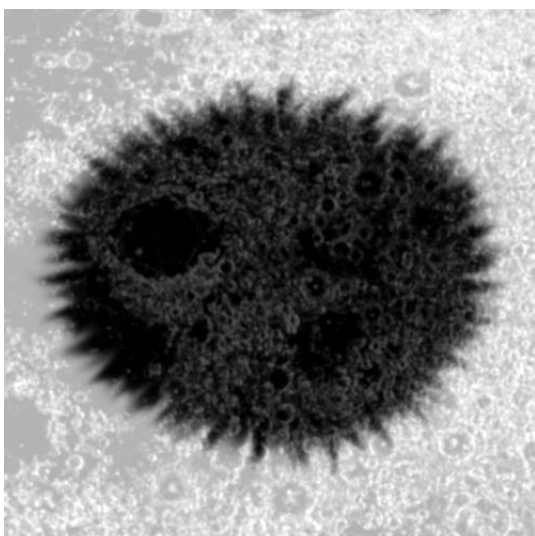
40%, smudge again, but this time make inside to outside strokes. Feel free to experiment with sizes of brush and strengths to get the desired effect. Once you're done, perform a gaussian blur (1 pixel radius) and save. What you are aiming for is something that looks like this:



6.) Now for the actual terrain that will make up your map. Going with the crater theme, I downloaded a topology (bump) map image of the moon, available from this website: [1]([approve sites](#)) (Click number to view). Now, using the select tool, pinch a 500×500 section out of that image. I chose the area roughly to the right of centre, as it has some nice crater-style detail to it. To select an exact sized area, in the toolbar at the top of photoshop, select 'fixed size' under style for the selection tool, and enter the size as 500×500. Now click and drag anywhere in the image to place your selection box.

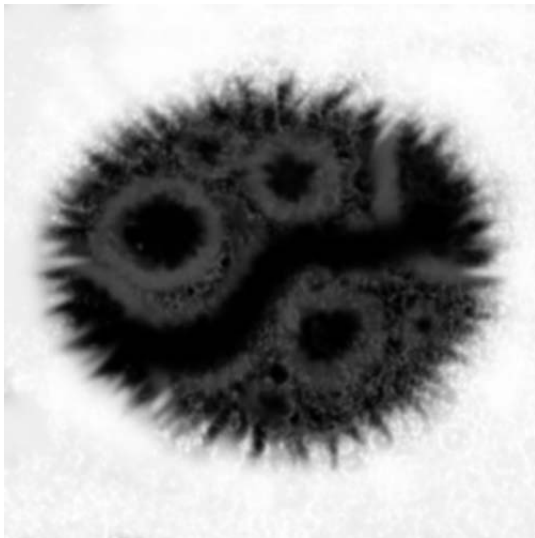
7.) Copy that selected area, go back to your crater image, and select Layer→New Layer. Do this twice, so two new layers are created. In the layer window at the bottom right, click on layer 1, making that one active. Paste your copied moon image into that layer. Now, click on the 'eye' next to layer 1. This defines whether it is visible or not. Hide that layer, and select the background layer with your crater pic on it. Select all, copy it, select Layer 2, and hit paste. You should have copied the background layer into layer2, and have moon data in layer one now. Finally, make layer 2 invisible for the moment (click the eye icon next to the layer) and select the background. With black as your foreground colour, press alt-backspace to fill in the background black.

8.) Now comes the clever bit. In the layer editor window again, make the two layers making up your image visible again by clicking where the eye icon once was. Now, making sure Layer 1 is moon data and Layer 2 crater (you can drag and move layers in the layer editor window), right click the crater layer and go to blending options. Under opacity, hit 70% and click ok ET VOILA!! You have a fairly realistic, craters kind of image that with a bit of luck, should look something like this:



9.) Ok, now select your Layer 1, moon data, and go to image→adjustments→auto contrast. This should lighten up those craters a bit on the moon layer only (as this is the layer selected), making them stand out a bit more. Feel free to take a white paint brush with a diameter of 50, soft brush style, opacity 80% and flow 70% and whiten up the area outside the crater a little (make sure layer 1 - moon - is selected, as the crater outside is already white!!), as this is detail you are not really going to see in-game.

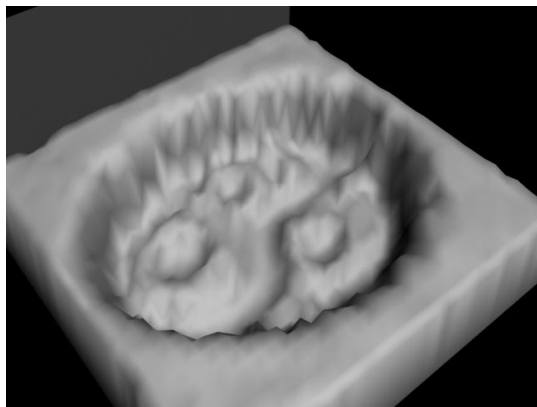
10.) In a similar style to the above point, use a white brush with Radius 20, Opacity 50% and Flow 70% and just draw circles around the edges of any large craters you have on your moon data. Make sure the moon layer (Layer 1) is selected for this, and not the crater layer. Switch the colour to black, keeping the tool constraints the same, and blacken out the inside of the craters. This will make them more flat in the centre with higher ridges around the outside when you import this map into max. Similarly, blacken out a small area at either side of the map for bases, and add any more terrain features you like, such as roads (use black), hills (use white) etc... Once you have added all your features, feel free to use the smudge tool and apply a 1 pixel gaussian blur if you want to make the transistions from higher ground to lower more subtle. With a little luck, you should have something like this:



If so, save your work as a tiff (keep layers, as you can come back to edit it later) and close photoshop.

11.) Go back into MAX now, and select the faces that make up your ground, except the outside row. The simple way of doing this, if you followed the instructions from earlier properly (:P) is to select the box object, go into edit mesh mode (face select), and scroll down to surface properties. Enter 2 in the select by ID box and those faces will be highlighted. If not, no matter, just re-select them using the instructions from above. Now, in the modifier list, with those faces still selected, add a displace modifier. Everything on the modifier stack is effected by whatever is below it. If you go back into edit mesh, select a couple of faces, then click on displace, your displace modifier will now be applied to the selected faces. So, in the displace dialog, select your map that you created and saved in photoshop (go to image→bitmap→ and click on the 'none' to add the bitmap). Once you ahve done that, go to strength at the top of the dialog, and enter 2000. This corresponds to the white height - ie: white in your image will now be 2000 units high, and black 0 units.

12.) BO!! The ground terrain should magically appear. If it didnt, go back and retrace your steps until it does!!! Here is an image of what it should look like. I have added some specular to make the detail of the terrain stand out:



Final Stuff

Nice, now that you have learnt how to do your own custom terrains, you should also know that with the displace modifier in its current state, editing the map in photoshop and reloading the bitmap for the modifier will update the terrain. So, you can get it as right as possible in photoshop, then continue your editing in MAX.

Make sure you use the reference models for the MC to gauge the scale of terrain features. Also, once you are satisfied with your terrain map, if you go into the modifier stack, and add another edit-mesh modifier, you can then edit individual vertices etc... using soft selection amongst other things.

Err, Also,

Once you've completed your terrain and want to add textures for the ground etc, you can use your displace map as a basis for creating base/alpha map for the level, as it is already lined up to the terrain. Make sure you resize the map to 512×512, or another power of two before importing it into the HEK though!!

Good luck making terrains!!!!

E Vi L Da VE :@

- view
- edit
- attach
- print
- history

:P PAGE UNDER CONSTRUCTION :P

REQUIREMENTS: 3D Max, Photoshop, HEK

ACKNOWLEDGEMENTS: None, except a nice book on 3DS Max that has been pretty helpful 🍷

DESCRIPTION:

It seems a lot of people's barriers to map making is the difficulty in using 3DS Max to create level geometry. This is a simple tutorial suitable for beginners, showing some useful techniques that can be used to create bases or other features. The bases from Bloodgulch will be used as an example, as they're quite easy to make, yet some interesting geometry allows for demonstration of efficient modelling techniques in Max.

STEP BY STEP INSTRUCTIONS:

1.) Research. If you want to recreate the base from bloodgulch, we are going to have to research it. In actual fact, it's very important to do research for anything you build. A good idea is to go in-game and take some screenshots of interesting stuff, for reference later on. The single player missions allow for a multitude of 'new' material - I say new because people will be less familiar with sp levels than mp. Do what an artist does - research your environment & draw inspiration from many sources in order to create something original!!

Setting Halo to take screenshots:

- Find your Halo CE shortcut. Right click it and go to properties
- Find where it says 'target' Modify this line: "C:\Program files\microsoft games\halo custom edition". to include -screenshot at the end.
- Result: <"C:\Program files\microsoft games\halo custom edition" -screenshot> without the <>'s but

WITH the space between “ and -screenshot.

- Start Halo CE from that same shortcut. Now, in game, when you press ‘print screen’ (next to F12) it will take a piccy and store it in your Halo CE\screenshots directory. But note you only got 6 piccies before it overwrites them!!

Heres some Screens of da Real base in BG:



Turrets around base



Base from top



Base from side



Inside the base



Inside pt 2

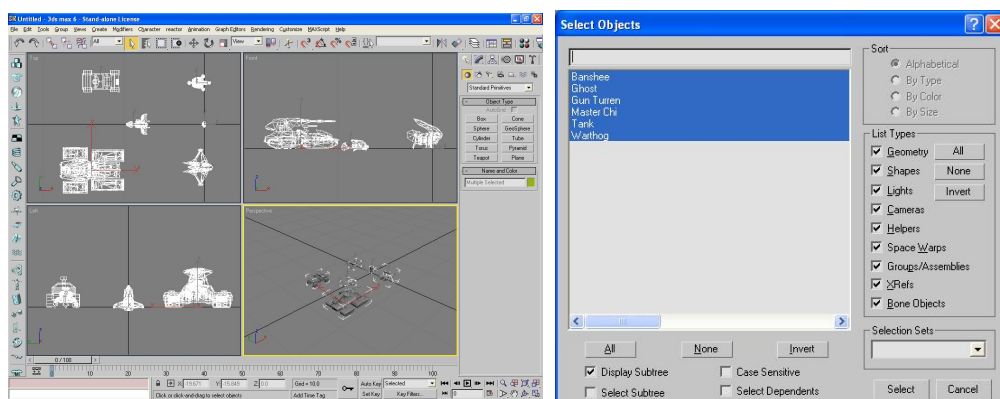


Top again

(Click Thumbnails to Enlarge)

2.) Right then, open up max and let's create this baby. Note that this tutorial is an exercise in basic modelling technique - not in perfectly recreating the base. If I wanted to get a Bloodgulch base into Halo CE, I would a.) Play Bloodgulch or b.) use HMT or Sparkedit to extract the mesh :P

First, we need a reference, If you have the Halo Reference Files, we are going to extract objects from these. If not, you can download the Halo Reference models [HERE](#). Ok, in MAX, go to file→import and select your Halo Reference Models 3ds file. Click ok to merge objects with existing scene. You should now see something like this:



(Click Thumbnails to Enlarge)

Right, now hit 'h' and up pops the objects lister. This is really useful for locating and selecting objects. We don't need the tank, banshee, turret or ghost, so select them (multi-select holding control) and hit ok. Now hit delete to remove all the vehicles you dont need, leaving the Hog and the Cyborg all alone awww :'(

-[X Ref?](#) MC (Pic) -Edit Mesh MC -Create [N Gon?](#) (2xPic) — Sides: 10 — Radius: 400 — Fillet: 0 — Height: 0 - Edit Mesh [N Gon?](#) —Verices Mode - Select All. Should be 11, its 22 (top and bottom). No good. Select all, hit weld. Now its 11 -Out of edit mesh. Posotion ngon so MC is next to it. Back to edit mesh.(Pic) —Face Mode - select and delete all faces in the middle. When prompted 'delete isolated verticles' hit yes. —Vertex Mode - Select all vertices —Click on move button at top. Shift and click on one of the selected vertices. Hit OK for clone to element. —In move rollout, hit the local coords button and enter 125. This should move 10 new verts UP. If it moves them to side, find out which axis is up (ie: x, y, or z) in your perspective view, click undo and then enter 125 in that axes rollout. —With vertices still selected, hit scale button. Scale them down in the TOP viewport until x and y values in the rollout go from 100 to 90. (if you get an oval shape, hit undo and rescale now using both x and y - top viewport :P) (PIC)

—Face Mode —Go to create. Now draw in faces around side (3x pic) ---Note: If you have to delete faces, if prompted 'delete isolated verticles, hit NO!' ---Also if you mess it up, hit undo and redraw. ---Draw in triangles in 3 points. Make sure u go anticlockwise. If you go clockwise, you will get faces with normals facing AWAY.

'Should look like this: show piccy.

Now, you wanna add the interior I guess otherwise doing the tops gonna sux0r

—Out of edit mesh mode.

Creating Ce Shaded Objects

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Open Gmax/Max. Now, this is going to be pretty easy.

Create a sphere, and give it 15 sides. Don't make it smooth, just clear all the smoothening groups. Texture it any way you like. Now, clone the sphere, and move the clone to the side. Flip all the faces around for the clone, and make the clone's texture completely black. Then, place it right over the original model (over as in inside, not on top). Once that is done, select all the faces for the black outline model. Go to the modifiers list, and find "push." Increase the settings a bit, and you should have a fitting outline! Now, to make your texture cel-shaded, open Photoshop or something that can use the Cut-Out feature. In Photoshop, open your texture. Now, go to filters, and find cut-out. Adjust the settings till you get it right.

Or you could just hit "M" (in max 6 only) and go to standard. Select ink'n'paint and it will make a default color for the material. Either select the desired color of the material or select the bitmap option. Apply the texture and boom. It will be Cel Shaded. Hit F9 to see if you like the way it rendered.

Here is a video tut by me ([Super Sunny](#)) <http://www.putfile.com/media.php?n=Vegeta15>

Local mirror: <http://wiki.fpsgamers.se/uploads/Videos/CreatingCeShadedObjects.wmv>

Note: Too good stuff to only have in one place. -Admin

Exporting Models Using the GBX Model Extractor

- view
- edit
- attach
- print
- history

Right like the title says this is a tutorial on how to export the [.GB Xmodel?](#) tags into a usable file type for max. I had major problems with this program at first but I found a way how to get them in max and it worked. So I have, after all this time(i've had a ton of work from college recently) decided to release a tut on how to get the models in max.

Right, now to begin we will get the Plasma rifle into max since other models are missing more polys than this so not to be too complicated.

First actually get the Extractor I can't seem to find a site were you can get it but pm or email me and i'd be more than happy to send you it.

1. Launch the Extractor and click the 3 dotted button this will take you to a dialog box(lol sriry it's what it's technical name is i've been leaning GUI it's a window where you open a file or save)and goto your folder were you store your HEK. Default is in the CE folder and which is probably in programs files. Once in your CE folder goto the tags folder and goto weapons and plasma rifle and if the file type is the same as mine there should be some folders followed by one file the .gbx file. Now while still in the dialog box create a new folder(optional) and call it source model this is where were going to store the model. Once you've done that open the GBX file and choose load.

2. Now you will have two options one Geometry and two LOD both of which I don't understand except geometry has a hiracry that you have to go through to get to the choice of quality and the or LOD lets you choice a radio button (round circle with one choice only) of the quality and lets you select the model on the right of it so select the model from either way (i'd select super high but you can choose what you want. I've never tried the others). Now once you've selected the model choose save (it should be near the bottom of the program window by about)and save it somewere in the folder or the subfolder you've might have created save it as Plasma rifle and choose .obj NOT .3ds. Now you're probably wondering what the save markers/nodes is now I have a theory about it I think it extacts the bones for the model but since I don't know how to rigg i'm leaving it.... for now.

3. Launch Max and goto file, import, and goto the place were you saved the file(make sure the file type is wavefront object .obj or all formats) open it. A box should appear on screen I don't understand most of it but leave it all alone except for one bit, make sure that the vertex scale is 100.0 and click OK; now you should have a plasma rifle model that is exact to scale not too small or too big. There you go you've now got the model in max, this tut can be used for other models just replace the plasma rifle with what ever model you want and please note that some models have multiple sections and are missing more polygons than the plasma rifle so you might have to carry out more work.

(C) 16 November 10:42 GMT Matthew Gavigan(Gav89)

<http://i20.photobucket.com/albums/b209/gav89/Beggruntbeg.jpg>([approve sites](#))

Optional tip: you now have the rifle model but no texture here is a very quick way to textureize(G-ster word lol) it up. Get the halo map tool 3 or 3.5 and extract the bitmaps for the weapon then in max make the material from standard to multi/sub object and add the materials to each section then highlight each bit(from the modify tab, second tab) where each texture goes(i'd do it in by selecting the faces) and scroll to the bottom and make the material ID to what ever ID is the texture.

I'd like to thank firescythe for releasing the rigged Elite and Grunt models keep up the good work mate.

Legal Notice: I have recently found out that since i'm a british citizen that any thing I create is automaticly copyrighted so therefore this means this is copyrighted. Here is my agreement to you the online public, if you wish to publish this on the internet elsewhere or again on this forum please do not claim it as your own give me the credit, the author acknowledge me Gav89 and there will be no problems thnks.

I hope this tutorial help you all and I expect to see more models which are better textureized than that of my own good luck to all.

- view
- edit
- attach
- print
- history

Here is my tutorial skinning [Halo CE](#) models

there are two parts to this: the first is on the plain skins and the second is on cubemaps.

1. okay, since the HEK doesn't provide you with skins of the vehicles and etc. you can do one of two things: use HMT to export(which doesnt work with cubemaps) or you can do it the ghetto way aka my way.
2. open up the .bitmap of the vehicle skin that you want in geurilla.
3. now open up paint.
4. now in geurilla, show the bitmap so you can see it.
5. hit print screen on your keyboard.
6. now in paint paste in the screen shot and move it into the top left corner.
7. make sure that you get the image exact.
8. now reduce the size of the bitmap down to the correct power of two(32×32 , 64×64 , 128×128 , 512×512 , etc.) use the attributes diologe from the image menu to get it right.
9. now save this and import it into your higher end program of choice.
10. do what you want and all that and save it as a .tiff and convert it to a .bitmap like you usually do.

now with the cube maps this is a little bit more difficult.

1. you do the same thing with the opening of the .bitmap to it. you will see 6 little textures.
2. now do the same print screen thing to get that picture into paint.
3. now open up data\levels\ai10\bitmaps\cube map hazard light.tif(i use this because the black colors contrast with the green borders on it) in the higher end program that you use.
4. now one by one select the squares from the paint and put them exactly over the "template" in the same spots. *i strongly suggest using zoom to get this exact*
5. when you are all done with coloring and what not, save this as a seperate map.
6. now when you run it through tool.exe it wont give you errors.
7. now open up the newly created .bitmap in geurilla.
8. make it so that it shows the map, you will notice that all the little pics are seperate from each other and it only shows one at a time.
9. to fix this all you need to do is on . bitmap you have to change it to say cube instead of 2d texture(this is right below the show bitmaps button)
10. now save it.
11. re-run the tool.exe with the bitmap again and now it will be in the cubemap format.

- view
- edit
- attach
- print
- history

REQUIREMENTS:

Photoshop, A poorly tiling texture, Some patience

ACKNOWLEDGEMENTS:

Original sand texture taken from [HERE](#)([approve sites](#)) after a google search using keywords ‘sand AND texture’.

DESCRIPTION:

If you want to use a texture as a detail map, you will probably be aware that images have a course seam at the edges where the tiled map lines up. This brief tutorial discusses how you can make a poorly tiling texture such as a photograph of sand or dirt etc into a tileable one, by editing out the seams in Photoshop. This is just an example, but the technique can be applied to almost any texture. Organic textures such as sand, dirt, grass etc are easier to tile than more regular ones such as brushed steel effect,

STEP BY STEP INSTRUCTIONS:

1.) First up, download the texture below. Save it as sand_original and load it into photoshop. This texture was originally taken from [here](#)([approve sites](#)). All I did was select a 256×256 portion of this texture using the selection box (hint: use the rectangle select tool, and at the top of the page select style→fixed size and enter 256×256 as the desired size). I chose a section that had as few dark regions as possible, as these would make the tile effect more difficult.

D/L Sand_Original (256×256)



(Click on thumbnail to enlarge)

2.) With that texture now in photoshop, apply a high pass filter (Filters menu→Other→Highpass). I used a value of 25 pixels to smooth out any large variations in shade over the whole image. Experiment with different values if you want. A lower value such as 1–10 will only allow very fine details through. A higher value has virtually no effect, other than to eliminate large shading anomalies.

Notice the texture has lost some of its colour. While this cannot be restored, the effect of equalising the content of the image can be achieved using image → adjustments → auto contrast. If the result of the auto contrast is too harsh, undo it and try manually adjusting it using image → adjustments → brightness/contrast.

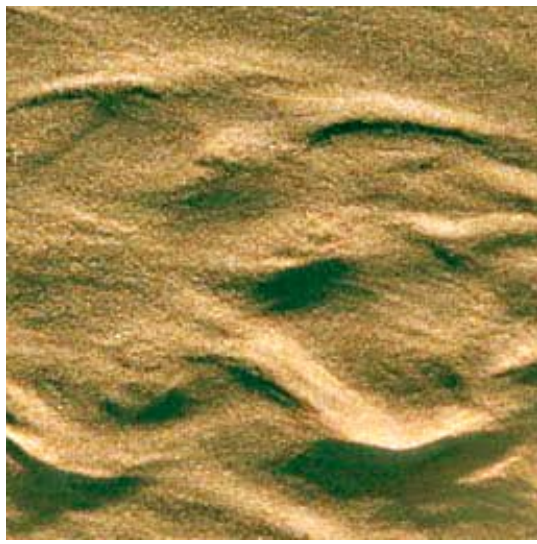
Sand after HPF & Contrast



(Click on thumbnail to enlarge)

To restore the yellowish colour of the sand somewhat, goto image → adjustments → variations and vary the average colour of the texture. I hit the magenta button once, followed by OK to get this result below:

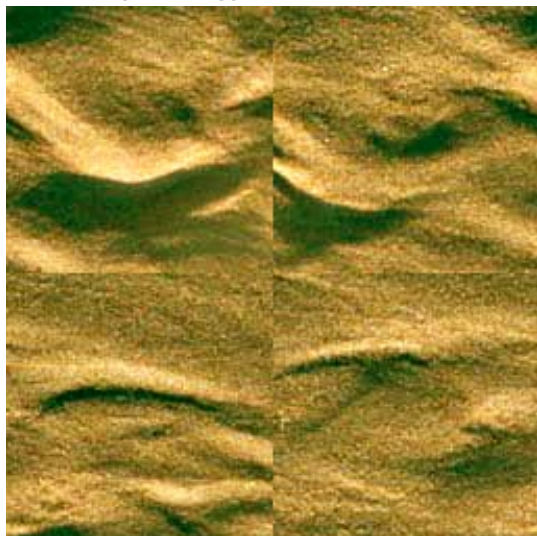
Sand after Variations



(Click on thumbnail to enlarge)

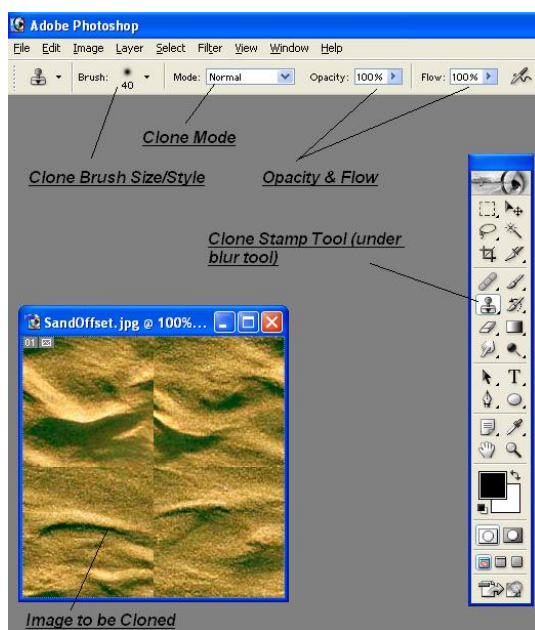
Now the next filter we are going to apply is an offset filter. Go to Filters → Other → Offset and enter 128 & 128 in the horizontal & vertical pixels boxes, and make sure wrap-around is checked. This will have the effect of tiling your texture within the 256×256 image - showing up the harsh seams at the edges. Note that if the size of your image is 512×512, you want to offset at 256×256 and so on. The image should now look something like this:

Sand after Offset Filter



(Click on thumbnail to enlarge)

3.) Ok, the image is now ready to be edited to become seamless. In order to do this, we are going to be using the clone stamp tool. This is available in most packages, not just photoshop, but this example will follow the photoshop layout as that's what I'm used to :P

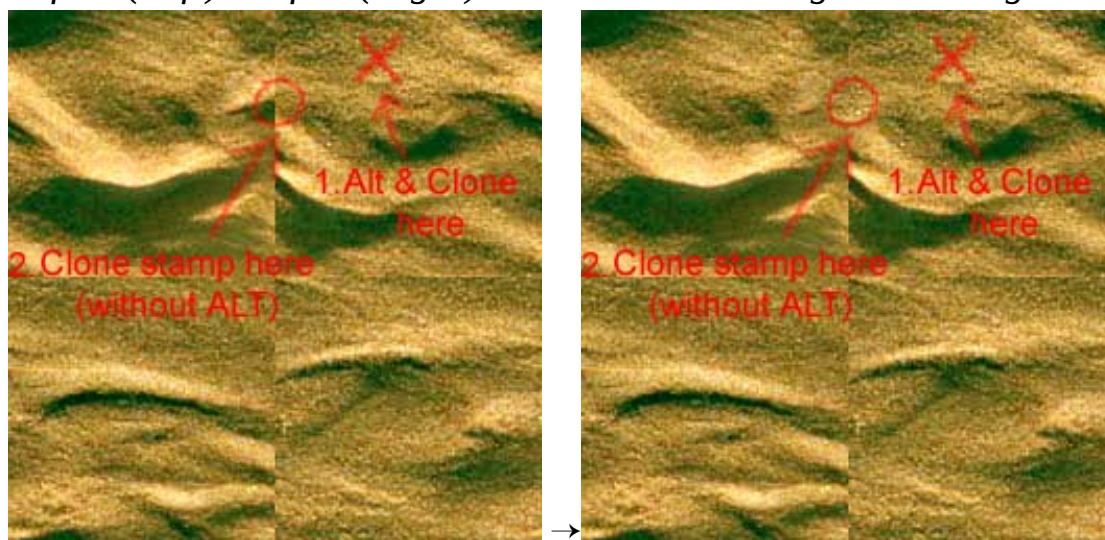


(Click on thumbnail to enlarge)

The parameters of the clone stamp tool that are of interest are the brush size & style, the opacity & the flow (see image above). Essentially, what the clone stamp tool does is allows you to replicate a portion of an image, and clone it elsewhere. By choosing a soft edged brush and carefully selecting source & target sections of the image, its possible to edit almost anything to be seamless!!

To edit this image, first, select the clone tool, choose a brush style with a nice soft edge, and a size of 40 pixels. Drag the opacity down to 50% and leave flow at 100%. Now, if you right-click on the image at any point while holding ALT, that point becomes your source. Once you have defined a source, clicking anywhere else without the alt will make that section the destination. With an opacity of 50% set it may require 3 or 4 clicks to transfer a section of image to another part. If the result looks dodgy, you can always use the multiple undo levels (undo levels are viewable in window → history) and try again. Here's an example:

Before (Left) & After (Right) Clone. Click on images to enlarge

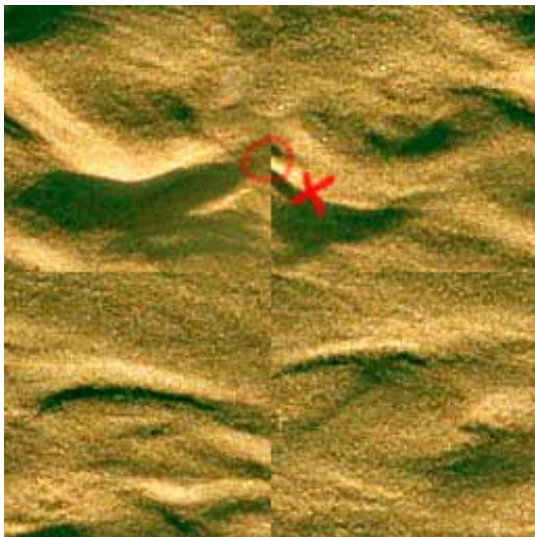


In these images, I alt-cloned under the cross in the first image, and clone-stamped (no alt) under the circle. When clone-stamping, I held the mouse pointer still and right-clicked 4 times to get the desired effect, as the opacity is low. Look carefully at the two images - can you see the difference that the clone stamp has made? It has copied the section from under the cross and pasted it under the circle, eliminating the seam at this precise location.

Note: The clone source (alt & clone) is set as a RATIO from the destination in pixels. If you wish to stamp on another section of image, you must re-position using your source by pressing alt & clone over the desired source section.

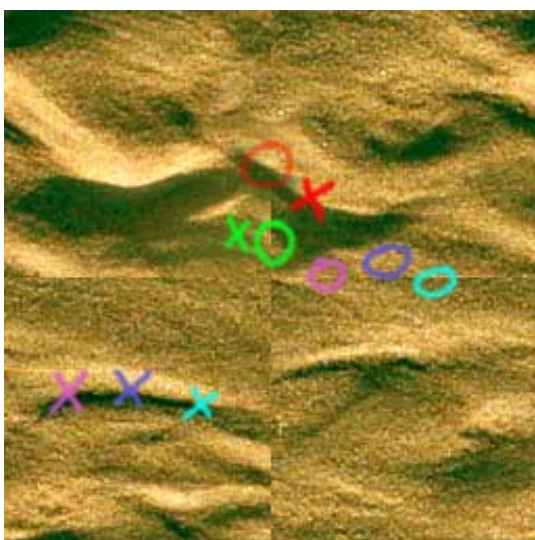
Repeating this clone-stamp action, pasting sections from flat portions of the image over the seams is one method of removing the edges. However, do note that if you merely paste over the seams from flat areas of the image, you will still have a visible tile effect, where the edges of the texture are flatter than others. The trick to this is to carefully pick your source and destination in order to make it look organic - like the original texture. Also, varying brush size & performing multiple clones will allow you paste whole features from one part of the image to another.

4.) Continue your clone stamping until you are happy with the image. It would be too length to show the entire process from start to finish, but try this for starters:



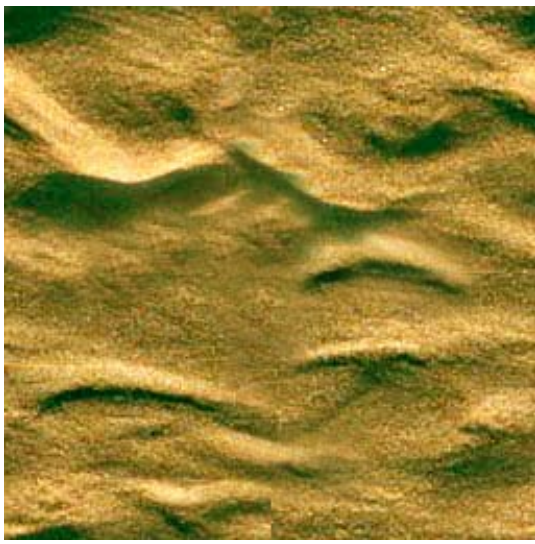
(Click on thumbnail to enlarge)

Above: This nice feature under the X can be copied to the O location to get rid of the seam in that location. Perform a clone operation using the X and O as source and destination respectively.



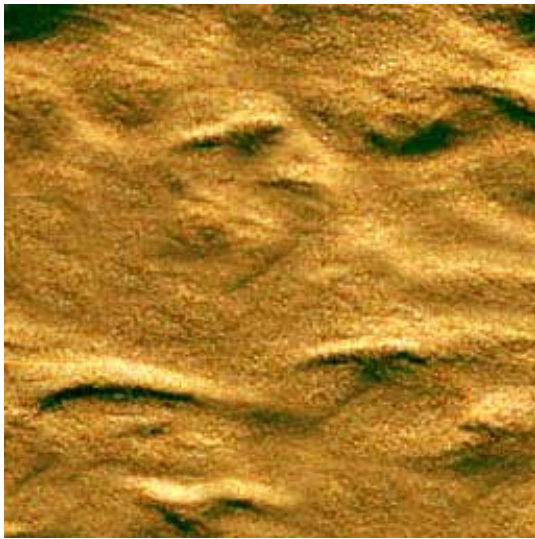
(Click on thumbnail to enlarge)

Above: Repeating this theme, I cloned the following features. X is source (press ALT & Clone) O is destination (Clone). I did these stamp operations in the order Green, Purple, Blue, Cyan.



(Click on thumbnail to enlarge)

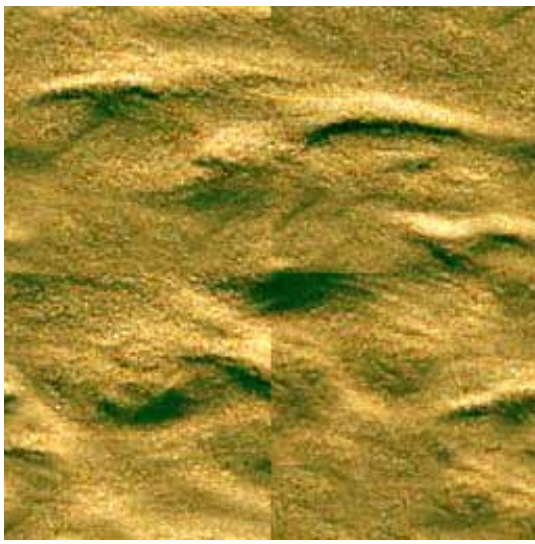
Above: The result of the above operations. Cunningly place the clone source & destination to copy features from one location of the image to another. If you don't like the outcome of a clone op, simply undo it and repeat as necessary. Notice that some operations cause the image to appear blurry. Use a nice featureless section of the image that has sharp image qualities, such as the flat sandy area in the middle-left to clone out these blurry sections.



(Click on thumbnail to enlarge)

Above: Finally, your image should be seamless to the eye. This is an example of my seamless texture, it will most likely differ from yours, but the most important thing is to make it look natural!!

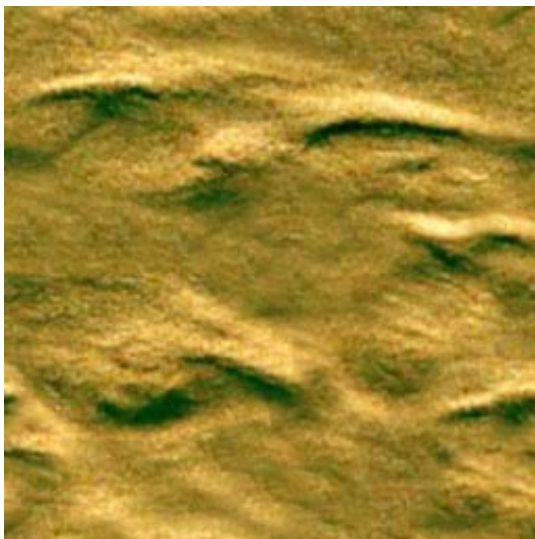
5.) But it's not finished!!!! In removing the seams, you have now introduced additional seams when the tiled texture is de-offset. In order to see these, apply the offset filter once more: Filter → Other → Offset, and select values of 128×128 with wrap around on.



(Click on thumbnail to enlarge)

This time the seams are no where near as severe though. This is just a result of applying a clone stamp near the edge of the original offset image, thereby introducing seams when you revert to the original form. In order to get rid of these slight seams, repeat similar steps to those layed out above, selecting source and destination for the clone stamp, but making sure that the stamp brush DOES NOT go off the edge of the image - keep it within the image boundaries!! Once you've finished your clone stamping, apply a weak gaussian blur (Filters → Blur → Gaussian Blur) with a strength of 0.5 and save.

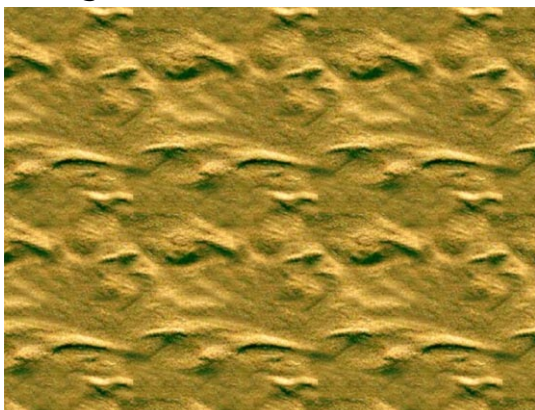
The Final Tileable Texture



(Click on thumbnail to enlarge)

Test your texture by saving it and setting it to the background image in Windows & selecting the option 'Tile' rather than stretch or centre. If it looks something like this, then you've done well!!

Tiling the Sand Texture



(Click on thumbnail to enlarge)

Additonal information

In [PS P8?](#), seamless tiling is very easy. Check this very short tutorial about [Seemless tiling in PSP](#).

Final Stuff:

This tutorial explains the principle of making textures tileable & the best way to learn is to try it out for yourself on images/textures downloadable from the web. The sand texture above tiles pretty well - there are still some seams that need working out, but overall its a fairly decent job.

One point though, the sand texture with it's large amounts of highs and lows will make it inherently difficult to texture map, as from one direction those ridges in the sand will look raised, but if you view the texture from the other direction, the ridges will look like deep furrows in the sand. Also, this map is only 256×256, and with its large variations over the image, even though the seams are not visible, it may well be possible to see the repetition if you were to use this as a detail map.

It depends on what you consider acceptable in your map - if you dont mind this, then by all means, use this type of texture. If it's more of a problem, it may be a better idea to have a flat sand image as the detail map, and apply a greyscale bumpmap to give the appearance of ridges/furrows in the sand. This will mean that the ridges will appear as raised ridges no matter what angle you look at them, although creating a realistic bumpmap is certainly not easy!!

[E Vi L Da VE](#) :@

Addendum for Photoshop 7 users. The healing brush will speed this process up exponentially. It works the same way as the clone stamp tool, but it automatically blurs and does some other stuff all in one thing. Mess around with it, it can save you some time.

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

REQUIREMENTS:

Paint Shop Pro version 8, A poorly tiling texture

ACKNOWLEDGEMENTS:

Goes to [E Vi L Da VE](#), for bringing the subject up and for the text I stole from his tut =)

DESCRIPTION:

If you want to use a texture as a detail map, you will probably be aware that images have a course seam at the edges where the tiled map lines up. This brief tutorial discusses how you can make a poorly tiling texture such as a photograph of sand or dirt etc into a tileable one, by new functions in Paint Shop Pro 8. Organic textures such as sand, dirt, grass etc are easier to tile than more regular ones such as brushed steel effect,

STEP BY STEP INSTRUCTIONS:

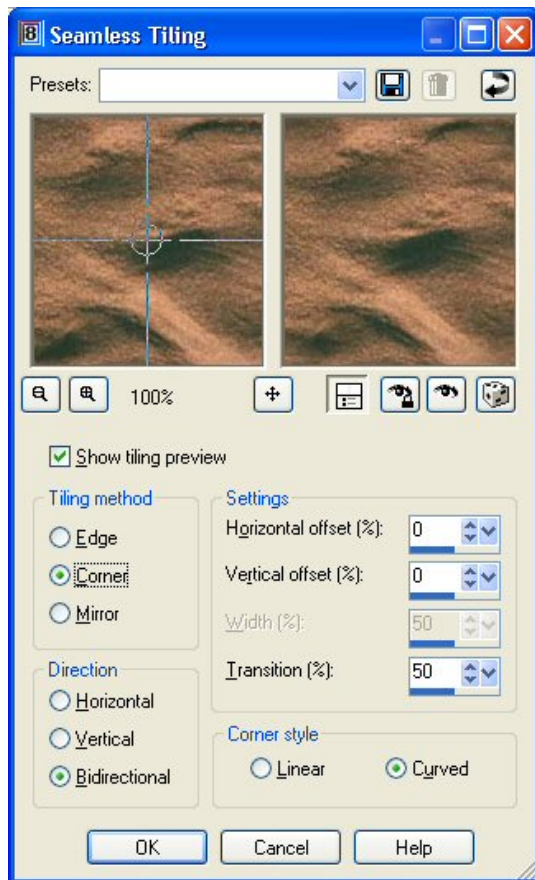
1) Open up the image you want to make tilable. I've used [E Vi L Da VE](#)'s sand texture used in the original tutorial [Tile Texture Tut](#). You might wanna use some of the techniques in that tut to brush the image up somewhat.

D/L Sand_Original (256×256)



(Click on thumbnail to enlarge)

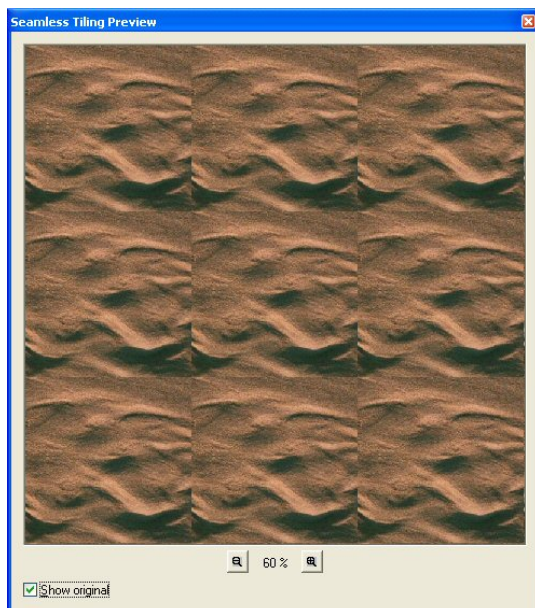
2) Go to *Images - Image Effects - Seamless tiling*



(Click on thumbnail to enlarge)

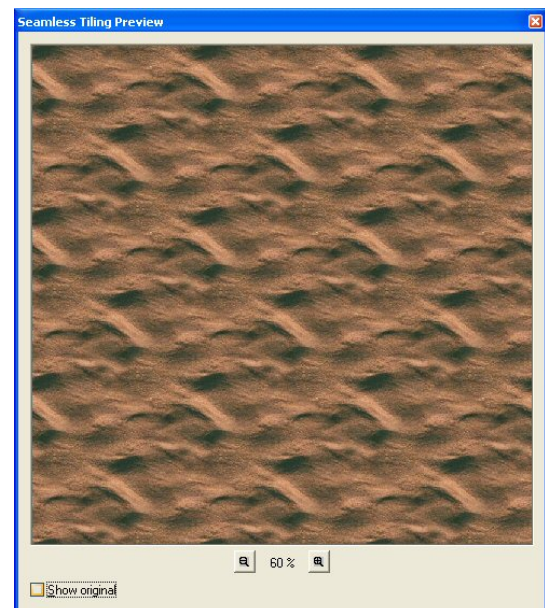
3) In the 'Tiling method' groupbox, select 'Corner' in combination with 'Bi-directional' in the 'Direction' groupbox, since it gives the best result for this image.

Original image



(Click on thumbnail to enlarge)

Seamless tiled



(Click on thumbnail to enlarge)

4) You are done.

- ZiON

I will be buying PS P8? now... - E Vi L Da VE

Powered by PmWiki

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

- view
- edit
- attach
- print
- history

:P PAGE UNDER CONSTRUCTION :P

This tut will cover advanced UVW mapping using the UVW Unwrap function in 3DS Max. Far from the method previously presented where you manually move UVW's to get the desired result, this method uses a cunnnnnnng technique to get that mapping almost perfect - well, more perfect that it was!!

Particularly useful for skinning complex objects that don't fit the basic planar, spherical, cylindrical or box maps!!

Page will be up soon, watch this space!!

Aw Sh** I dont have much time these days. If someone feels like filling this one in, go for it, else I will do it when I get the chance!!!

Edited by Sonic Tag:

Create your mesh, if you are just going to test doing this, then create a box and include some extrudings to make it look as less like a box as possible.

Next, you can then go into the material editor and change the diffuse of a standard material to checkered and aplpy it, this is so that in the perspective you can see what it lookslike with a tessalated texture which can be of some importance, it also shows you if you have any streched areas on your model.

Go to the modifier > UVW MAP, set it first to planar, then go to Modifier > Unwrap UVW, Press Edit, at the top you will see some options, these are very useful to you, especially normal mapping and flatten mapping, unfold mapping is better for the more simple of objects, I use it alot after I have simplified a mesh into alot of smaller meshes within the UVW Wrapping co-ordinates, and then it makes life a little easier. I would reccomend setting up some shortcuts at this point.

Select Normal Mapping, then go down to Box mapping in the selections for that, and apply it. Then you should move, rotate everything apropriatly, keep checking your checker map. Make sure that it is all in the

box.

For the next part what you will want to do is get a tool called Texporter, what this does is take your UVW mapping, and turn it into an image, so you do not have to take a screenshot of the texture coordinates. For this part, you should make it a square, i would reccoment 512 x 512, or 1024×1024 as these are squares and are multiples of 2 which most game engines support.

You can test out this by applying the UVW picture from texporter into your mesh, and taking a render. This should have no red parts, if it does then it means that there are overlappings in the mesh, and also you should see the deteails going along the edge of the polygons.

Now you can simply go over that in your paint program of your choice, I personally prefer Paint Shop Pro, although Photo Shop is better in the perspective of most people.

Most people think that a head is a very complicated shape to unwrap, but it is actually very very simple, all you do is select your head, go to the modify pannel and scroll down on the list to UVW Map, and change it to planar, then in the Unwrap UVW modify tab, just move some vertecies around and that is it, you might not even have to move any vertecies around. Depending on the shape of the head it can get very complicated.

When your shape gets very complicated, what you can start to do is take rows of polygon in the same axis which dont have to much rotation from eachother, maybe even columns of polygons also, and apply a UVW Map in Planar to the correct axis that you have selected, so that the planar will cover it as if it is above that section, then you do this alot and eventually you will have created your unwrap, when you are doing this, make sure that you weld some of the vertecies together in the UVW window, it will make it much easier to apply a paint to.

Automatic UVW mapping method - Click on the “Mapping” menu, then “Flatten Mapping”. Make sure the three checkboxes are checked, and hit “OK”. You’ll see your object broken up into chunks, almost like pieces of a puzzle. If you printed this out and cut out all the pieces, you could actually assemble them into a rough version of your object - they’re not perfect, but they’re pretty close. Although you get alot of odd triangles, in the perspective 3d viewport you should try and select these and put them into a planar map, which is basically a more automated version of selecting the many triangles at a time.

There is also somthing else you can do, basically take parts of your model with having face select on (to do this click the + on the modifier pannel next the Unwrap UVW and then start with the more central face, then basically take a few faces at a time (depends on how flat they are, but if they are not flat then you will want to select them one at a time or row at a time) and basically you planar map that whilst you are in the unwrap UVW so you just have to scroll down the modifier list a little bit, and then simply in max you can stitch it. Here I would defintly reccoment you have a stitch shortcut, because it makes life alot easier and it speeds up

your modeling process by a substantial ammount. and you simply select the few verts where the part that you just planar mapped should attach to and press ctrl + s and it will automatically attach.

There are a few video tutorials I hear for this method of modeling and if you wait then I will make one soon.

Halo /

Creating New Textures

- view
- edit
- attach
- print
- history

This is a tutorial for making a new skin for the warthog. You can apply this tutorial to any object.

This tutorial will work on anything. im just using the warthog as an example because I wanted to put Quaglinoman’s tron warthog into ce

Ok, first of all, you will need HEK and Photoshop or [Paint Shop Pro?](#) with the dds reading plugin Secondly in your data folder, make the following file structure

```
data (should be there already, it’s the folder were starting in)
  vehicles
    warthog
      bitmaps
```

Take the texture in Photoshop or [Paint Shop Pro?](#) and save it as a .tiff in the bitmaps folder (if it is a patriotic warthog texture call it patriothog.tiff etc.) make sure you use LZW compression or else the pictures will have a huge file size

Then open up the command prompt and type in

```
tool bitmaps vehicles\warthog\bitmaps and then hit enter
```

Once that’s done, go into guerilla and open the warthog hull.shader_model. Then scroll down to where it says Maps and under that should be a selection box and it should say base map. To the left there is a little box with three dots, click it. Search for your patriothog.bitmap which you converted from a tiff, double click it once you find it. Now save it.

Open up a .scenario map in Sapien and make a hog (mission>objects>units>vehicles and edit types and find the Warthog.vehicle) and congrats

Halo /

Adding Scripts

- view
- edit
- attach
- print
- history

Adding scripts (Outdated)

Update: This page seems to not be helpful to a few who have tried. Yet it should remain until someone have replaced it with something useful. //admin

Update: I’ve unlocked all of this crap in Guerilla (you can download it from my site, hek.teamhalo.net), so this tutorial really isn’t important reading. Also, these values below are filed in by Sapien so it doesn’t if you make them or not matter.

I have been doing a little Import\Export hacking and got this: If you export your scenerio tag and open it up in a text editor you can see all of its elemets (duh). But I’ve been fooling around with the “script” element (next will be globals). you can replace your current script element which should look like this this Code:

```
scripts      block      0
```

and replace it with this: Code:

```
scripts      block      1
  element 0
    name      string     test
    script type      enum
    return type      enum
    root expression index      long integer      0
  end element 0
```

As you will see, you can’t edit the values in Guerilla still, but its still a major break through. Not sure if you will be able to in Sapien (its script textbox is to in a “readonly” state, but you never know).

This only took me 10 minutes.

Adding Globals

Replace Code:

```
globals      block  0
```

with this

Code:

```
globals      block  1
  element 0
    name      string  test
    type      enum
    initialization expression index      long integer      0
  end element 0
```

A note: You must tab each new thing in the import file. Ex: Code:

```
before: name string test
after:  name      string  test
```

when adding elements and their data you have to make a single space. Ex: Code:

```
before:
ABC block 1
element 0
name string DEF
end element 0
```

```
After:
ABC      block  1
  element 0
    name      string  DEF
```

end element 0

- view
- edit
- attach
- print
- history

The main file for scripting documentation that you will need is `hs_doc.txt`. If you have Sapien, you can create this file with a simple command.

Load up Sapien, and open the command console with the tilde key (~). Type:

“script_doc” (No parenthesis.)

This will create a file in your main Halo: CE folder called `hs_doc.txt`. This contains all the scripting commands you can use in scripting Halo: CE, along with a short description and explanation on how to use them for most of them.

Credit for these findings is due 100% to [kornman00?](#), and perhaps a bit to [CL 9mm man?](#), for leaking some of this information to the public. But mostly to korn, as without his tedious work, scripting Halo: CE would have been impossible for the general public.

Music Scripts

- view
- edit
- attach
- print
- history

Scripting music in Halo: CE maps.

Warning: Scripting syncing requires use of bipeds and vehicles (or other crushing objects.) in order to work over the netcode. These methods will work for single player maps, or multiplayer maps built for just one person. Syncing over Halo’s netcode will most likely not happen unless you work with the bipeds and vehicles. Remember - death’s sync, most else does not.

Warning 2: This method will use AI, and will check for the status of AI. AI ***DOES NOT SYNC*** over the netcode. Don’t try, you will fail. 😊

Tutorial:

This tutorial will assume that you have basic scripting knowledge, basic AI knowledge, can type with a keyboard, and can read. If you do not qualify for any of these needed qualities, please, do not hesitate to smash your face into your keyboard until you can no longer see.

What you’ll need:

1. A compiled .scenario tag that you can edit in Sapien. 2. A .sound tag that you have compiled with tool. This should be the music you want to play in-game. 3. AI actors and actor variants. If you don’t have these, check over at the files.halomods.com forums. Search around there for “Aztec’s Actors” or something like that, and download his actor and actor variant tags.

open the desired .scenario file in Sapien. Place your AI encounters and squads, and give them appropriate names. For this tutorial, I’ll call my encounter “tutleets1”, and the squad name will be “uber.” No reason for the names. :P

Anyways, place the AI, and under the encounters block and the information for it, check the box that says “Not created automatically”, or something to that effect. You can skip the checking of the box, but for this script, we will use commands that place this encounter on the map at a certain time, instead of right away as soon as the game loads. This can come in handy once you work with trigger volumes/loading zones/multiple BSPs, and that sort of thing. Make sure you give your AI spawn points, check the required boxes, and make sure they have respawn enabled, and have their max/min actors set to something other than zero.

Now, load up *gasp* Notepad, and type the following (Note: DO NOT ADD *’s.) :

```
(script startup musicnai
(ai_place tutleets1) *
(sleep_until (= (ai_strength tutleets1/uber) 0)) **
(sound_impulse_predict sound\music\madrigal true) ***
(sound_impulse_start sound\music\madrigal none 1)) ****
```

* This will place the AI that we decided to not create automatically.

** This will pause execution of the rest of the script until the give condition is true, or in other words, until your AI are all dead.

*** This script will bring your sound or music file into the sound cache for your map. This has to be done before actually playing the sound, as far as I know. I'm assuming your file is found at sound\music\madrigal.sound for this. If you were to name your .sound file anything else, change "madrigal" accordingly in the script.

**** This will start playing your music.

I believe that after the last command in the script, you can do a (sound_impulse_predict [sound location] false) command, to remove the song from the cache. But, I'm not sure if you'd have to run a sleep command first, or if you could just run it right after running the sound to begin with. That could stop the sound abruptly, but I'm not sure. Go ahead and experiment with this on your own.

After typing that, save the file as scripts.hsc in your levels\[your level]\scripts directory, or something like that. Go into Sapien, and go to File > Compile scripts. If it gives you any errors... well, try and fix them. I'm not a damn troubleshooting guide. :P

Of course, to play music, you don't need AI or anything. You can use trigger volumes and things like that, but that gets just a tad more complicated. This is just basic scripting using an AI command or two.

Final note: This is a very basic way to play music in Halo: CE. I'm sure there are better, more elaborate ways, but this will get the job done. 😊

Pkmnrulz240/Dark Cloud

- view
- edit
- attach
- print
- history

Sorta tutorial on how to use Devmode effectively

Author: Timo12

Okay since so many people have been asking i decided to post a sorta tutorial thingy. First of all you need to make a shortcut of halo CE on your desktop. done? okay, then right-click and select properties. then in the target box you should see something like this:

“C:\Program Files\Microsoft Games\Halo Custom Edition\haloce.exe”

To be able to kick, you should make it read as follow:

“C:\Program Files\Microsoft Games\Halo Custom Edition\haloce.exe” -console

Then ingame when you hit the ~ key, The “halo cmd” as I like to call it (console) comes up in the corner. Now the quickest way to kick someone if your holding a server is:

1. Hit the ~ key then type in sv_players the players #, amount of tk’s ping etc. will come up.
2. Find the number that is next to the person you want to kick.
3. Finally type in sv_kick # (# is the number of that person)

extra note: or if you want to ban do the same thing except type in sv_ban #. and to find out all the commands hit Tab, but most won’t work unless your holding a dedicated server.

Dev Mode?

To activate devmode is the same to activate the kicking ability. First go into the halo CE shortcut on you desktop, hit properties and if you still have -console in, it will still read like this:

“C:\Program Files\Microsoft Games\Halo Custom Edition\haloce.exe” -console

and for devmode to work it needs to look like this:

“C:\Program Files\Microsoft Games\Halo Custom Edition\haloce.exe” -console -devmode

Okay now that you have activated devmode, when you go to play multiplayer only create game: LAN is avaiable. Then make you own game and again when your ingame hit the ~ key. Now if you hit TAB there is too much for it the screen to pick up and most don’t work. but here are some cheating commands:

- This spawns one of each powerup around the player: **cheat_all_powerups**
- This spawns one of each vehicle around the player: **cheat_all_vehicles**

- This spawns one of each weapon around the player, including the flag and oddball skull: **cheat_all_weapons**
- This will spawn one regular warthog next to the player: **cheat_spawn_warthog**
- This lets you take control of any character you make contact with: **cheat_bump_possession <#>**
- This makes you absolutely invincible, even from falling: **cheat_deathless_player <#>**
- This gives you infinite ammunition for every weapon: **cheat_infinite_ammo <#>**
- Any enemy that looks at the player will automatically die: **cheat_medusa <#>**
- This lets you jump very high. (should be used with deathless player cheat, otherwise you may die while falling): **cheat_super_jump <#>**
- Human weapons and the needler never have to reload, and the plasma weapons can not overheat: **cheat_bottomless_clip <#>**
- Change this from 1–20 to make the game run faster. change it to 0 to make the game pause: **game_speed <#>**
- clears the console screen: **cls**

Now for the camera to work you need to type in **debug_camera_load**. Now the camera will get out of the master chiefs view. To move around you'll need to click-in the scroll mouse/middle button, other wise hit both the left & right buttons at once.

To move around:

NOTE: these are if you have custom controls or not.

W - Forwards.

S - Backwards.

A - strafe left.

D - strafe right.

R - move up

F - move down

G - Spin camera on the X axis???

Scroll mouse up - slows down the camera movement

Scroll mouse down - speeds yo the camera movement

camera_control 0 - takes you back to master chiefs view.

To edit the HUD (heads up display):

hud_show_crosshair 0 - makes crosshair disappear.

hud_show_shield 0 - makes sheild disappear.

hud_show_health 0 - makes sheild disappear.

hud_show_motion_sensor 0 - makes the radar disappear

To make them re-appear instaed of putting a 0 after, put a 1. but if you want to take out everything:

show_hud 0/1 - NOTE: it will say “false” but everything on the HUD should disappear.

Cool things

cinematic_start - makes the screen fade into the widescreen cinematic type thing.

cenematic_stop - makes it fade out into normal

Now, to be able to play halo CE properly again, either quit out of CE or do it the smart way and type in “quit”

into the devmode prompt and it will end halo. Now go back into the halo CE shortcut properties, then target and take off the -devmode.

Hope I helped, please post if i've missed n e thing out

L8erz, Timo12

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Reference Sections

General

- [Technology Overview](#)
- Creating a [Development Environment](#)
- Development [Tools Overview](#)
- General [Level Design Information](#)
- General [Model Information](#)
- [Materials Overview](#)
- Halo [Player Statistics](#)
- [Console Command List](#)
- [Weapons Ammo Items List](#)
- [Vehiclesand Stationary Gun List](#)

Level Design

Multiplayer Level Design

- [General Overview](#)
- [Level Creation Part 1](#)

- [Level Creation Part 2](#)
- [Level Exporting](#)
- [Level Compilation](#)
- [Level Data Manipulation](#)in Guerilla
- [Radio Sity](#) (Lighting)
- [Population Placement](#)of Objects
- [Running a Game Level](#)in Halo

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

Technology Overview

The following section contains a brief description of the engine and related technology used in the Halo for the PC, including hardware system requirements.

Brief Overview

The game technology used by Halo for the PC is a proprietary game engine developed by Bungie Studios and Microsoft with additional code and modifications made by Gearbox Software. The core rendering engine supports 4 rendering paths including fixed function rendering (Direct X? 7.0), Pixel Shaders 1.1 (Direct X? 8.0), Pixel Shaders 1.4 (Direct X? 8.0), and Pixel Shaders 2.0 (Direct X? 9.0) and takes advantage of the latest generation video hardware. The rendering engine supports vertex and pixel shaders in the form of a materials system. The engine uses a bsp-style visibility solution with vertex lighting for models and a simple radiosity solution supporting lightmaps for world geometry.

Minimum System Requirements

The minimum system requirements to run Halo for the PC are as follows: Operating System: Microsoft Windows 98, Microsoft Windows Second Edition, Microsoft Windows Millennium Edition (Me), Microsoft Windows 2000, or Microsoft Windows XP. Computer/Processor: 733 megahertz (MHz) processor. Direct X?: Direct X?® 9.0 or later. (Direct X? 9.0b is installed by Halo.) Memory: 128 megabytes (MB) of RAM. Hard Disk: 1.2 gigabytes (GB) of free hard disk space. Video card: 32 MB or 3D Transform and Lighting capable. CD: 8X. Sound: Sound card, speakers, or headphones with multiplayer play.

- 56.6 Kilobytes per second (K Bps?) modem or 10 MB network adapter. Multiplayer servers require broadband to run a server.
- Additionally, you must have an Internet Service provider (ISP) or a Local Area Network (LAN), if you want to play multi-player games.

Additional System Requirements

The following are the additional system requirements for editing Halo. Please note that the end user should refer to the system requirements for any additional programs (3ds max™, Photoshop®, etc..) that they also intend on using for Halo editing.

Additional system requirements for Halo editing are as follows:

Hard Disk: A minimum of an additional 550 mb of free hard drive space

A 3 button mouse. A third mouse button is necessary to activate the camera controls in the Sapien.

Additional system memory (RAM) and processor speed are highly recommended for editing Halo.

[Return to Main Page](#)

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

- view
- edit
- attach
- print
- history

Creating a Development Environment

This section explains what the editing kit contains, how to install and update Halo PC for editing, and how to install the Halo PC End User Editing Kit as well as how to set up directory structures and prepare Halo for content creation and editing.

*NOTE: Many of the screenshots in this tutorial show a working directory of C:\Halo PC. However, the examples that go along with this tutorial are set up to use the default install path of Halo CE C:\Program Files\Microsoft Games\Halo Custom Edition. It is **‘highly recommended’** that you use this default install path for Halo CE when first getting started.*

Halo Editing Kit Contents

- “Blitzkrieg” - 3DS Max exporter
- “Tool” - Halo file converter
- “Guerilla” - Tag file editor
- “Sapien” - Level populator
- Data Files- Source art
- Tags - Halo game files
- Tutorial - HTML file & Level

Installing and Updating Halo for the PC

To use the Halo Editing Resource and the files included in the Halo Editing Kit you must be running Halo CE, the unsupported version of Halo PC.

Information about Halo CE, the Halo HEK and where to obtain Halo CE can be found at Gearbox Software Halo Page http://www.gearboxsoftware.com/projects_halo.html[\(approve sites\)](#)

Halo Editing Kit Installation

To install the Blitzkrieg Exporter Plug-in for 3ds Max do the following:

1. If you are running 3ds Max, close the program
2. Copy the file Blitzkrieg.dle from the **Halo CE\Blitzkrieg** directory.

3. Place the Blitzkrieg.dle file into the plugins directory under 3ds Studio Max (eg. c:\3dsmax5\plugins).
4. Start 3ds Max.

If the plug in was successfully installed and loaded then you should be able to go to File→Export and see the following export options under **Save as type** in the pull down menu (scroll down the list) “**Blitzkrieg Animation Exporter**” and “**Blitzkrieg Model Exporter**”.

Additional Tools Installation

The Halo Editing Kit requires the installation of some additional tools that are NOT included in the editing kit installation package.

For installation of additional tools or commercial software, such as 3ds Max, follow the installation procedures and instructions included with those programs.

3ds Max 5.1 was used in the tutorial areas of this document. The Blitzkrieg exporter is available for use with 3ds Max 3.x, 4.x and 5.x.

Information on 3ds Max is available here: <http://www.discreet.com/3dsmax/>.

[Return to Main Page](#)

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

Development Tools Overview

This section contains information on the tools and related programs necessary for content creation related to Level Design for Halo PC.

The tools used for creating Halo game assets are a combination of both custom created tools and commercially available tools.

See the section Creating a [Development Environment](#) for information on where these tools and exporters should be placed to allow editing for Halo.

Textures and graphics can be created using any graphics package that can export or save in the .tif graphics format (Photoshop, Paint Shop Pro, etc...).

All the levels as well as models and animation in Halo are created using 3D Studio Max and Character Studio. Plug-ins for exporting from 3D Studio Max to the data formats that get converted to Halo data “tags” are required. Exporter plug-ins exist for 3D Studio R3, 3D Studio Max 4, and 3D Studio Max 5.

The following sections give a brief overview and description of the tools and their function in the development path for Halo.

Blitzkrieg (blitzkrieg.dle)

Blitzkrieg is the exporter plug-in for 3D Studio Max. This plug-in exports the model data (vertices, materials, smoothing group information, animations, etc...) from 3DS Max to the file formats (.jms, .jmp, .jma, .jmm, .jmt) used by tool.exe to compile a data tag that can be used by Halo.

Tool (tool.exe)

Tool is a multipurpose command line program that compiles and converts data that has been exported from other sources such as 3D Studio Max. The data files that are created are referred to as “tags”.

Running “tool.exe” by itself with no command line parameters will display a complete list of the available command line parameters and appropriate variables and variable formats.

Guerilla (guerilla.exe)

Guerilla has a graphical interface that is used to edit the properties and resource references in the Halo data “tags”.

Sapien (sapien.exe)

Sapien is used to edit or modify various aspects of a level that has already been compiled using Tool.

Sapien has a graphical interface that contains a 3D window that renders the game including the level bsp, models, and vehicles as well as playing sounds and running physics.

Sapien is used for many purposes, including the population of characters, weapons, items, scenery objects, and game play and mission objectives

[Return to Main Page](#)

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

General Level Design Information

The following section includes various standards, conventions and guidelines used in Halo Level Design. This section also contains editing information such as polygon counts for level models or meshes, per scene triangle counts and guidelines, and performance guidelines and suggestions.

Style Guidelines

The following are some aesthetic or stylistic suggestions and guidelines when creating new content or levels for Halo multiplayer:

- 1) Halo multiplayer levels tend to be stylistically part of the universe from a design and aesthetics point of view, borrowing geometry or architectural elements from the single player game environments. However, the multiplayer levels tend to be removed from the narrative portion of the Halo universe. This allows a Halo multiplayer level to look and feel like Halo while offering a great degree of freedom.
- 2) Human structures, ships, vehicles, weapons and other related elements are military themed. The human technology style is relatively near future with obvious advanced technology but it is all recognizable and functional without being overly complex or “gadgety”. The color palette tends to follow the aforementioned military theme with darker greens (olive drab), various metallic colors (gun metal, burnished aluminum), dark grays or blacks, as well as various browns. The construction style tends to be angular with 30 degree and 45 degree angles being standard.
- 3) Covenant structures, vehicles, ships, weapons and other related elements are heavily sci-fi themed. The Covenant technology is far more advanced than comparative human technology but not as advanced as the Forerunner technology. The color palette for the Covenant tend to be various metallic reds, blues and purples. These same colors are also used for their energy fields or shields and other luminous elements found on Covenant objects. The construction style tends to be organic utilizing smooth curves and rounded surfaces.
- 4) The Halo structures or Forerunner structures and the technology they contain are far more advanced than either Human or Covenant technology. These structures tend to be massive and distinctly “alien” yet elegant when compared to the Human or Covenant objects and structures. External covenant structures tend to have smooth metallic surfaces with large or bold architectural features. Interior structures tend to be larger (compared to Human-sized rooms or hallways) almost cathedral-like in many instances with some common elements such as pillars, illuminated control panels and wall panels, as well as various alien machinations. The color palette for Forerunner structures tends to stick to metallic colors such silver with blues, purples, and yellows being common for the luminous elements found in the Forerunner structures. The inner surface

of the Halo ring and its terrain and thus associated colors tend to vary. Even though many of the environments do not appear in the game it is assumed that Halo has all the environments found on Earth (except geological formations and features related to geothermal or volcanic activity such as volcanoes or geysers). These environments include rolling plains and grasslands, deserts, oceans, lakes, beaches, islands, forests, snow and ice covered canyons, etc...

Multiplayer Level Design Technical Rules and Guidelines

Here are some technical rules, guidelines and tips that apply to creating Halo multiplayer levels: 1) Halo levels currently HAVE to support ALL 5 game types (Capture the Flag (CTF), Race, Slayer (Deathmatch), King of the Hill (KOTH), Oddball) in order to function properly within the game engine. Missing the netgame flags or necessary elements (the netgame flags for the objectives, player spawns, etc...) for these game modes can cause the game or server to crash or have problems.

Supporting all the game modes in the level also insures consistency with the design of Halo multiplayer and cuts down on the confusion that having game mode specific levels would create.

2) Adequate player spawns. It is suggested that a minimum of 16 player spawns per TEAM (red or blue) for each game mode. Player spawns can actually be set up to be used for more than 1 game type.

A multiplayer level should have 16 player spawns per game type for the level regardless of what the level description says or what the level was designed for. There is a very high chance that a server operator will run the level with 16 players, having adequate spawns will keep the game and player spawning behaving properly.

3) Minimum distance between player spawns. The distance between player spawns should be 20 units apart or greater for non-team game modes and 2.5 units apart or greater for team based game modes. The Halo spawning code is unique. For Capture the Flag, the spawns do have the concept of team or side so players should never spawn in the enemy base or on the enemy side of the level. For other game types, either team or non-team, the spawning code will try and spawn the player near friendly players and avoid spawning the player if an enemy player is within a certain distance of the spawn point. If two or more spawn points are within the minimum distance stated above, then it is possible that a player will not be allowed to spawn at one of the neighboring spawn points if an enemy player spawns or has spawned at another neighboring spawn point (at least until they move out of the minimum safe distance). Understanding how Halo spawns players in the various game modes and team settings is very important in the design of the level and placement of spawns.

4) Physics. Halo has physics, keep this in mind at all times. Physics is fun, but can also be crazy to design for.

5) Collision. Be aware of Halo physics when constructing geometry. Thin geometry that is part of the world mesh is very likely to cause collision problems, typically the player or a vehicle will go straight through it. Larger and thicker geometry is better for collision. For trees and other objects such as rocks, use scenery models when possible and even with these make their manually created collision models simple and robust.

Multiplayer Level Design Suggestions and Guidelines

Here are some basic game design or level design guidelines and tips that directly apply to Halo multiplayer levels:

1) Keep the level simple. A level can have a simple layout yet still offer a great depth of game play for all the Halo game types (Bloodgulch, Timberland and Danger Canyon are good examples of this).

2) Try to support all the Halo game types well. Most Halo multiplayer levels support some game modes extremely well while the other game modes in the level are not as fun or offer as good of a game play experience. This is the most challenging aspect of Halo multiplayer level design. Some level ideas or concepts may not work well with some game types, but its still possible to support the game types and not only make them playable, but enjoyable. Maps that support all the game modes well tend to be the more popular and

3) The more popular Halo levels tend to be symmetric in layout but with balanced asymmetrical features that add variety without giving one side a distinct advantage over the other (in team games). Asymmetrical features can be anything including terrain features, cover such as trees and rocks, even items such as power ups and weapons.

4) Bases or base areas. The more balanced and well rounded levels have 2 distinct team areas or team bases. Bases should have multiple entrances and exits to facilitate movement into and out of the base and to prevent bottlenecks. Keep in mind the fact that vehicles can be driven and\or “tweaked” into just about any area. Vehicles are very effective at blocking off travel routes, design the base area and entrances, exits, and access ramps and areas accordingly.

5) A Halo multiplayer level does not have to have every vehicle in it to be fun, do not force a level to have a certain vehicle in it if its not designed for it. Timberland was designed for ground based combat and vehicles and contains no Banshees, where as Infinity was designed for the Banshee and Banshees work very well in that level.

6) Balancing the level for the player on foot versus the player in a vehicle is crucial. This applies to the travel times and distances for a player on foot versus a player in a vehicle and the combat between a player on foot or a player in a vehicle (give the player on foot terrain features and cover against vehicles).

7) Teleporters are a very good tool, but are very easy to abuse. Remember that the player can take the flag through a teleporter. Keep in mind that teleporters can be made one way. Only players can go through teleporters, no matter how big the size of the teleporter, therefore teleporters are good ways to balance the travel distances and times between players on foot and players in vehicles.

8) Balance the weapons and item placement. Use these items to create or control traffic or game flow in certain areas of the level. More powerful weapons should be towards the middle of the level or require the player to travel a farther distance to get. This also holds true also for items and powerups.

9) Halo has physics! As was mentioned above in the technical section, keep this fact in mind at all times. Take advantage of it (ramps and jumps are fun with vehicles) and try not to abuse it or allow it to be abused\exploited by the player.

10) Make sure to keep the player (and vehicles) out of places where they should not be! Use obstacles, sky, and player clipping to keep players from cheating or exploiting the level.

11) Keep in mind that vehicles (especially Warthogs and Ghosts) can be tweaked or manipulated into areas that they should probably not be. Players will always try to block important level paths or objectives with vehicles. While these can be valid tactics, make sure that such tactics do not “break” the level or limit the game play in the level.

12) Most importantly, be creative!

Polygon Count Guidelines for Construction

The following lists the suggested maximum amount of polygons that should be used to construct a level of the indicated type for Halo.

This value corresponds to the construction limit for the world mesh or model for the level, the basic core geometry for the level.

Additional scenery objects that are placed as models are handled in the following section for Triangle Counts (per scene).

These suggested maximum values are guidelines to insure proper exporting and compilation by the tools as well as reasonable polygon amounts for various other technical aspects of the game.

Multiplayer Levels: 10,000 polygons

Triangle Counts (per scene)

The following lists the suggested maximum amount of triangles that should be allowed to draw per scene.

Multiplayer Levels: 50,000 triangles

The values listed are a general guideline due to the fact that performance is affected by many factors other than triangle counts such as the number and type of materials (shaders) in the scene, sounds, dynamic lights, lens flares, and other game objects.

As can be expected, the fewer triangles being drawn the better. When creating a level, the level designer should always take into the account additional triangle counts added by vehicles, players, weapons, items, and effects and prepare for any worst case scenarios in terms of triangle counts and rendering.

When creating a level, its always a good idea to view a similar level that exists in the original game and see what kind of triangle counts exist per scene at various areas of the level. Checking the triangle counts and other rendering information as well as the frame rate and performance in those areas should give a good

baseline during the development of your own multiplayer environment.

The level designer should also realize that the triangle counts for a scene compared to the polygon counts for a scene when constructed or viewed in 3DSMax may vary greatly depending on the material used and how that material is rendered in the game. Often, the surface has to be rendered multiple times for the applied material and associated rendering effects and properties set for that material or shader.

A simple example, a ladder is constructed with 2 polygons in 3DSMax and a ladder shader is applied to it with the 2-sided property flag checked in the shader due to the fact that the ladder can be viewed from all angles. In game, this 2 polygon ladder will show a triangle count of 4 since BOTH sides are rendered, the back side or back face is not being culled and must be drawn.

Overall Performance Guidelines and Suggestions

As was mentioned above, the performance of a multiplayer level is dependant upon many factors, not just the pure number of triangles being drawn.

Below is a simple list of things to keep in mind when creating a level for Halo:

1) The 50,000 maximum triangle value listed for multiplayer levels is a rough average, keeping the triangle count to around 30,000 triangles per scene or less is a great range to insure smoother performance on a wider variety of machines.

9) Performance is a balancing act. Knowing the general rules and knowing the technology will allow you to exceed some limits or boundaries by decreasing or limiting others. For example, making a scene that is higher in triangle counts than normal but lower in overall material or shaders types that are also of a lower complexity is a common trade off.

2) Avoid using a lot of lens flares in general or lens flares per scene, such as the colored beacons used for marking team areas of the level.

3) Dynamic lights are very, very expensive performance-wise. Limit your use of them or eliminate your use of them all together.

4) The number of different materials or shader types per scene can greatly affect performance. Having a room with all the vehicles, weapons, and items (each has its own unique material or shader for the most part) as well as different materials all over the room can be very costly. Try to limit the unique number of materials per scene.

5) Avoid using or creating a lot of shaders that have a lot of effects or settings or that require multiple rendering passes, this can greatly decrease performance. Shaders with a base map, primary detail map, secondary detail map, micro detail map, bump map, scrolling texture settings, specular properties, and reflection cube map all set decreases rendering performance and uses up more memory.

7) Certain shader effects such as the “dynamic mirror” flag under Reflection Properties for .shader_environments are extremely expensive, do not use these shaders or use these shader properties.

6) Watch the amount of overdraw created by having successive layers of transparent or partially transparent geometry (e.g. several layers of glass, looking through several waterfalls or other similar transparent particle effects)

8) Be careful of the number of ambient sounds placed in the level using .sound_scenery tags.

9) Take into account the triangles rendered in the sky or sky box model you have chosen for the level. Also take into account the model materials and shaders and number of shaders chosen for the sky box model or used in the sky box model. The triangles for the skybox do not get culled or not drawn the same as for world geometry that has a visibility solution in place (portals). If possible, simplify the skybox by making a custom skybox model.

[Return to Main Page](#)

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

General Model Information

This section includes various standards, conventions, and guidelines used in model creation as it relates to Halo Level Design. This section also includes editing information such as polygon counts, guidelines, and methods for trouble shooting and optimizing models.

Style Guidelines

See the **Style Guidelines** section under the General [Level Design Information](#) section.

Polygon Count Guidelines for Construction

The following lists the suggested maximum amount of polygons that should be used to construct a typical model of the type listed. While the Blitzkrieg exporter and data tag compiler Tool can handle much higher polygon counts (since the same tools are used to export level geometry and other higher polygon count models such as character and vehicles), its recommended to keep the values within the range suggested for the listed model type.

Typical objects used for scenery objects include trees, rocks, bushes, and other small details. The value listed below represents the average maximum value for scenery objects used in multiplayer levels.

NOTE: The values listed are for the Super High Model LOD (Level of Detail)

Multiplayer Scenery Objects (Models): 500 polygons

Triangle Counts (in game)

The following lists the suggested maximum amount of triangles that should draw for the listed model. NOTE: The values listed are for the Super High Model LOD (Level of Detail)

Multiplayer Scenery Object: 1,000 triangles

The values listed are a general guideline due to the fact that performance is affected by many factors other than triangle counts such as the number and type of materials (shaders) associated with the model.

As can be expected, the fewer triangles being drawn the better. When creating a model, always take into account its effect on the level and the scene in which it is being placed.

When creating a new model, its always a good idea to view a similar model object that exists in the original game and see what kind of triangle counts exist for that particular model. If creating a new tree model, check the current tree models in the game and see what are their associated triangle counts.

Overall Performance Guidelines and Suggestions

Many of the factors mentioned in the **Overall Performance Guidelines and Suggestions** in General [Level Design Information](#) apply to the performance of a model object in Halo. A model objects effect on performance includes many factors, not just the pure number of triangles being drawn.

Below is a simple list of things to keep in mind when creating models for Halo:

- 1) Be careful of the polygon and resulting triangle counts for the model. The model should not be created without regard for its use and placement in its intended environment. If a level or game environment already has high triangle counts and a model is being placed in the area, the model being created should have a low triangle count and\or the environment it is being used in should have its triangle count lowered.
- 2) Scenery objects are used for many reasons such as decreasing the development or creation time of a level by allowing the population of common objects quickly. The use of scenery objects also reduces memory by using multiple instances of the same model that uses the same material or shader. Memory used by the game is also saved since the objects are not made part of the core level geometry and do not require a light map.
- 3) Always consider using or creating several Levels of Detail (LOD) for the model you are creating, especially if its a very high polygon model that is used frequently in the environment and has many instances per scene. LOD models are manually created lower polygon versions of the higher polygon model seen by the player up close. These LODs are set to replace the higher polygon version at farther distances from the player view. As the distance from the player view increases, lower and lower polygon versions of the model are used. These distances can be manually set in the model tag.
- 4) The number of different materials or shader types per model can greatly affect performance. Try to limit the unique number of materials per model to decrease the number of materials per scene.
- 5) Avoid using or creating a lot of model shaders that have a lot of effects or settings or that require multiple rendering passes, this can greatly decrease performance. Shaders with a base map, multipurpose map, detail map, detail mask, bump map, scrolling texture settings, and reflection cube map all set decreases rendering performance and uses up more memory. Be careful when using the 2-sided model shader flag, this will double the triangle count for the faces on which it is applied. This flag is most commonly used on the leaves for trees and foliage.
- 6) Watch the amount of overdraw created by having successive layers of transparent or partially transparent geometry.

Return to Main Page

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

- view
- edit
- attach
- print
- history

Materials Overview

The following section discusses the materials system used in Halo. The materials or shaders are used to define the properties of the surfaces for which they are applied to as well as the rendering properties for the surface. This section includes material naming conventions and rules, special material names, and special shader symbols or flags.

Overview

Materials define the properties of a surface in Halo. These properties not only include the basic texture image to render on the surface, but includes any special rendering effects as well as other data such as the type of material it is in regards to physics and projectile impact particle effects. Materials are often referred to as shaders. As with most data in Halo, shaders are defined in a data file called a tag. The shader tag contains many properties, flags, and fields for linking to other data tags, such as image data tags known as .bitmap tags. The .bitmap tag contains image data, such as an image and its associated mips, as well as other settings. The most basic shader typically contains just some simple default settings and a reference to a .bitmap for its base map.

There are many different shader types denoted by their file extension. The file extensions are fairly self explanatory, but there are 3 shader types that are commonly used for the creation of level geometry, these are listed and explained in the table below.

.shader_environment
Basic shader applied to the surfaces of a level environment. This shader is used for opaque surfaces. This is the most common type of shader used or applied to surfaces in environments.

This shader contains the basic properties, such as radiosity properties, physics properties, environment shader properties, environment shader type, lens flare, diffuse properties (includes base map, primary and secondary detail maps for texture or material blending, and micro detail map properties), bump properties, texture scrolling animation, self illumination properties, specular properties, and reflection properties (includes cube map properties).

.shader_transparent_chicago
Shader applied to surfaces of a level environment that are to have transparent elements. This shader can be used when only 1 or 2 maps are required to construct or define the shader. Maps refer to a .bitmap tag reference.

This shader contains properties such as radiosity properties, physics properties, chicago shader properties, lens flare, map information (includes the maps to use to create the shader, the color functions, alpha functions, and the map u-scale and v-scale), 2D texture animation, as well as additional flags to define the shader properties.

.shader_transparent_chicago_extended

A more complicated version of the `.shader_transparent_chicago`. This shader is used in place of the `.shader_transparent_chicago` when the shader requires 3 or 4 maps to define it. Maps refer to a `.bitmap` tag reference.

In addition to the properties listed for the `.shader_transparent_chicago`, this shader has two sections that are used to define a 4 Stage Maps for video hardware that supports this kind of rendering and 2 Stage Maps section that is used to define the shader on video hardware that cannot handle 3 or 4 maps in a shader definition.

Material Naming Conventions and Rules

The following are some VERY important rules concerning materials used in Halo. Materials have specific names and are applied to the surfaces (polygons, faces, etc...) of objects. It is these material names that reference a shader (in the form of a shader data tag). The shader is then used by Halo to apply the appropriate properties to the surface for that shader, including the rendering effects that should be applied as defined in the shader tag file.

The material name on the surface corresponds to the name of the shader tag. For example, a material is applied to the surface of a wall (the wall is made up of 2 polygons and the material is applied to these polygons). The name of the material as it appears in 3ds Max is “metal_wall”. A shader will exist for this material, in this case the associated shader data tag would be named “metal_wall.shader_environment”.

The shader data tags referenced for levels exist in subdirectories under the main level directories for that particular level. The subdirectory is called “shaders” (similarly, `.bitmap` files used by the shader files are contained in a directory called “bitmaps”). These shader data tags contained in these shader directories are accessible to any level. Levels can reference their own shaders, shaders from other levels, the shaders can reference `.bitmaps` from other levels, etc...

When the `.map` cache file is created that contains the level assets, the associated resources (such as the shader data tags) are pulled into the `.map` cache file. Because of how the `.map` cache file is created and how the tools work the following rules must be followed:

1) Every material name (shader name) MUST BE UNIQUE. For example, you cannot have two or more shader files named “grass.shader_enviroment” that are used in a level(s) that are placed in multiple level’s “shader” directories.

It is common practice that if custom shaders (new shader or a variation of an existing shader) are need for a specific level, the shader name is prefixed by the name of the level.

Using the example above, if you have a level called “mylevel” and wanted a new shader or to make changes to a shader that already existed called “grass.shader_environment” then you would call the shader “mylevel_grass.shader_environment” and place it in the “shader” sub-directory under the level directory, in

this case the “mylevel” game level directory.

2) Material names (shader names) must be 32 characters or less (excluding the file extension name).

Special Materials

The following table shows the list of special material names in Halo. These material names are hard-coded in the tools and game to be recognized as having specific or unique functionality. These texture names should NOT be used or applied unless it is for the specific function for which they are designed. No data tags (.bitmap, .shader_environment, etc...) should use the name of a Special Material.

+sky

Applied to surfaces to render the skybox or sky model.

+seamsealer

Applied to temporary surfaces or geometry to “seal” the level. Most commonly used to seal holes or other open edged areas of the level during construction and testing. For multiplayer levels, +seamsealer should never be applied to a face in the final build of the level.

+portal

Applied to faces that are used to define general portals used in the visibility solution or rendering occlusion for the level.

+exactportal

Applied to faces that are used to define an exact volume or portal. Such faces typically cover exactly the opening of a doorway, passage, or hallway to define a very distinct volume that can be used to occlude the rendering of other areas of the level.

+weatherpoly

Applied to faces that are used to define a portal or portals to use as a volume for setting weather effects.

+sound

Applied to faces that are used to define volumes for sound.

+unused

Reserved special material that has many uses and can be used in conjunction with the special shader symbols to define its use and behavior.

For example, it can be used with the “\$” Fog Plane Shader Symbol to make +unused\$, which can be applied to faces to construct a fog plane that is used to define a volumetric fog region.

Shader Symbols

The following table shows the list of special material or shader symbols used to assign certain attributes or behaviors to a particular surface in the level. These shader symbols are hard-coded in the tools and game to be recognized as applying their associated attribute when added to the end of a material name.

These shader symbols should NOT be used or added to the name of a material unless it is for the specific function for which it is designed. These symbols should not appear anywhere in the normal name for a material nor should these special symbols appear in the name of a data tag (.bitmap, .shader_environment, etc...).

The shader symbols are taken into account during the compilation of the .jms and affect the surfaces at this time.

For example, if a ladder is created using 2 polygons and has the material "ladder" (without the quotes) assigned to it. This ladder is able to be viewed from both sides. To make the ladder a true ladder with respect to the Halo engine AND to make it able to be viewed or rendered from both sides the "%" would be added as well as the "^". The resulting material name viewed in 3ds Max for the Sub-Material under the Multi/Sub-Object would be "ladder%^" (without the quotes). The data tag name would still be ladder.shader_environment.

%

Two sided property. This flag or shader symbol when applied to a material that is applied to a face or surface renders both sides of the surface instead of just the side that the normal is facing.

#

Transparent property. This flag or shader symbol when applied to a material that is applied to a face or surface allows the surface up to be rendered with transparency.

!

Render only property. This flag or shader symbol when applied to a material that is applied to a face or surface makes the surface only render the face, the face or surface will not have any collision.

*

Large collideable property. This flag or shader symbol when applied to a material that is applied to a face or surface sets the surface up to not be rendered and only have collision.

\$

Fog plane property. This flag or shader symbol when applied to a material that is applied to a face or surface makes the surface not be rendered. The faces acts as a fog plane that can be used to define a volumetric fog region.

^

Ladder property. This flag or shader symbol when applied to a material that is applied to a face or surface sets the surface up to act as a ladder for the player.

The surfaces is climbable.

-

Breakable property. This flag or shader symbol when applied to a material that is applied to a face or surface allows the surface to be able to be destroyed or broken. (e.g. glass)

&

AI deafening property. This property does not apply to multiplayer levels. This flag or shader symbol when applied to a material that is applied to a face or surface allows that surface to inhibit AI functions.

@

Collision only property. This flag or shader symbol when applied to a material that is applied to a face or surface sets the surface up to not be rendered and only have collision.

.

Exact Portal property. This flag or shader symbol when applied to a material that is applied to a face or surface makes the surface able to be used to define an exact portal.

[Return to Main Page](#)

Halo Player Statistics

The following sections include important player values and statistics as well as player dimensions in real world and 3DSMax units for the player dimensions, player actions, as well for some common game objects such as vehicles and doors. All player values and statistics refer to the Master Chief character unless otherwise noted.

Player Screen Field of View (FOV)

The player Field of View (FOV) value is 70 degrees.

Halo to Real World Scale

As a reference, the Master Chief is approximately 7 feet tall.

This makes the unit scale approximately 10 units = 1 foot or about 1 unit = 1.2 inches.

Player Dimensions (Generic 3ds Max Units)

(length(depth) x width x height)
Full Height (Standing): 18.418 x 29.828 x 70.127
Height (Ready Pose): 28.198 x 39.376 x 65.11
Crouched: 43.372 x 30.719 x 42.455

Vehicle Dimensions (Generic 3ds Max Units)

The following are the dimensions for the visible model for the listed vehicles.

(length(depth) x width x height)
Warthog (Both Variants): 191.766 x 134.0 x 104.264
Ghost: 137.967 x 144.484 x 57.926
Banshee (open cockpit): 181.368 x 223.696 x 158.011
Scorpion: 330.661 x 200.49 x 134.61
Shade Plasma Turret: 144.035 x 84.424 x 86.98

Player Speed (world units/second)

Forward Movement Speed (Running): 2.25

Backwards Movement Speed (Running): 2.0

Side Strafe Movement Speed (Running): 2.0

Player Movement Angles (degrees)

The player cannot move up any slope that has a angle value greater than the value listed below.

Maximum Slope Angle: 45

Ground Step Height (Generic 3ds Max Units)

This is the maximum height that a piece of geometry (such as stairs, guide walls, etc...) can be while still allowing the player to do a normal walk or run onto or over it.

NOTE: The values below are for when the player has accelerated to the full velocity for the listed movement type. For geometry such as stairs (vertical faces), the player will always catch or hitch, this is why ramps are used instead of stairs in Halo. There are techniques where visually stairs are represented but for smooth player movement there is invisible collision geometry positioned over the visual geometry that makes the stairs a ramp in terms of player collision and movement (the stairs in the Pillar of Autumn are a good example of this).

Crouched (Walking): 8

Standing (Running): 12

Jump Height (Generic 3ds Max Units)

This is the maximum height that an object can be while still allowing the player to jump on top of it.

NOTE: Other factors that apply additional velocity to the player may affect this value. The approximate values listed assume the player at standard running speed.

Crouched (Walking): 76

Standing (Running): 80

Crouch Jump (Run,Jump,Crouch): 99

Jump Distances (Generic 3ds Max Units)

This is the maximum distance that can be between geometry or objects that are at the same height while still allowing the player to jump from one object to the other.

Crouched (Walking): 136

Standing (Running): 312

Crouch Jump (Run,Jump,Crouch): 330

Ceiling Geometry Height (Generic 3ds Max Units)

The following values are the minimum distance that can be between the ground and the ceiling (interior ceilings, top doorway frames, etc...) while still allowing the player or a vehicle to pass underneath. NOTE: Keep in mind that the 3rd person camera view for vehicles is set at a much higher distance than the actual physical and visual geometry and will clip into the world at the values listed below. These are just the values that will physically prevent the vehicle from passing underneath.

Player

Crouched: 50

Standing: 70

Vehicles

Warthog (Both variants): 75

Ghost: 56*

Banshee: 112*

Scorpion: 118

* These vehicles can be tweaked or forced through the objects depending on the angle and velocity. Keep this in mind when designing the level.

Minimum Path Widths

The following values are the minimum widths that can be between geometry or between objects while still allowing the player or vehicle to pass between them.

Player

Crouched: 40

Standing: 40

Vehicles

Warthog (Both variants): 100*

Ghost: 128*

Banshee: 220*

Scorpion: 230*

* These vehicles can be tweaked or forced through objects or openings depending on the angle and velocity. Vehicles can be manipulated\exploited by forcing them through openings with much smaller values than what are listed above. The values listed are for driving the vehicle through an opening with little hindrance

and\or minimal visual clipping of the model. Keep these factors in mind when designing the level.

Player Falling Damage

The following table shows the resulting values for player falling damage.

Harmful Falling Distance: 3 to 6 (Random Range) units
Damage Lower Bound: 10 damage
Damage Upper Bound: 155 damage

The following table shows the resulting values for automatic death from falling.

Maximum Falling Distance: 16.5 units
Damage Lower Bound: 105 damage
Damage Upper Bound: 105 damage
Return to Main Page

Console Command List

- view
- edit
- attach
- print
- history

Console Command List

The following sections include command line switches and console commands that are useful when editing or creating content for Halo on the PC.

Command Line Switches

The following are valid command lines switches that can be placed after the game executable halo.exe. This list can also be obtained by running Halo with the -? command line swith (e.g. halo.exe -?)

- nosound Disable all sound
- novideo Disable video playback
- nojoystick Disable joysticks and gamepads
- nogamma Disables adjustment of gamma
- useff Forces the game to run as a fixed function card
- use11 Forces the game to run as a shader 1.1 card
- use14 Forces the game to run as a shader 1.4 card
- use20 Forces the game to run as a shader 2.0 card
- safemode Disables as much as possible when running the game

- window Run the game in windowed mode
- width x Forces the game to run at a specified resolution
- vidmode
w,h,r Forces the game to run at the width (w), height(h),
and refresh® specified
- adapter x Forces the game to run fullscreen on a multimon (multiple monitors) adapter

- port x Server port address used when hosting multiplayer games
- cport x Client port address used when joining multiplayer games
- ip x.x.x.x Server IP address used when you have multiple IP addresses
- screenshot Enables the 'Print Screen' key to generate screenshots

NOTE: Having this always enabled can cause a slight decrease in performance

- console Enables the debugging console
- devmode Enables the Halo Developer Mode

-Devmode Console Commands

Adding the -devmode command line switch will allow you to use many debug and cheat tools that will help you test your level. Please note that this mode is not meant to be used during real games and thus the ability to join games has been removed when in this mode.

NOTE 1: You MUST use the -devmode switch in conjunction with the -console switch for this mode to work correctly.

The following are valid console commands that can be entered into the Halo console when in developer mode. (-devmode), To bring down the console press the ~ key, you must have the command line switch -console enabled for halo.exe, Sapien has the console enabled by default. See the above section Command Line Switches for additional help and information.

There are many Halo console commands, this is not a complete list of the available console commands. The following console commands are common console commands that are helpful when editing Halo content.

NOTE 2: These commands are for development use only and thus may not provide a 100% stable game environment. Meaning - though useful, some commands may cause the game to crash.

NOTE 3: For commonly used commands, it is suggested that these commands be placed in an init.txt file. This file is a simple text file that is put in the root Halo game directory (the same directory that contains the halo.exe file) that contains the console commands and parameters exactly as they appear when typed into the console. When Halo is run, this file is read and the commands are run automatically.

Command or Variable	Default	Value	Valid Values	Description
---------------------	---------	-------	--------------	-------------

cheat_all_powerups	N/A	N/A		This console command will spawn all the powerups near the player.
--------------------	-----	-----	--	---

cheat_all_vehicles	N/A	N/A		This console command will spawn a standard warthog near the player.
--------------------	-----	-----	--	---

cheat_all_weapons	N/A	N/A		This console command will spawn all the weapons around the player.
-------------------	-----	-----	--	--

cheat_bottomless_clip	0	0-1		Enabling this command will give the player a bottomless clip which means that the weapon will never have to be reloaded and will never experience overheating.
-----------------------	---	-----	--	--

cheat_bump_possession	0	0-1		Enabling this command means that every time the player bumps a character they will take control of that particular character.
-----------------------	---	-----	--	---

cheat_deathless_player	0	0-1		Enabling this command makes the player invulnerable to any damage, including the maximum falling damage height restrictions.
------------------------	---	-----	--	--

cheat_infinite_ammo	0	0-1		Enabling this command will give the player unlimited ammunition.
---------------------	---	-----	--	--

cheat_medusa	0	0-1		When this command is enabled, any enemy unit that sees the player will be instantly killed.
--------------	---	-----	--	---

cheat_spawn_warthog	N/A	N/A		Typing this console command in will spawn a standard warthog near the player.
---------------------	-----	-----	--	---

cheat_super_jump	0	0-1		Enabling this command allows the player to jump very high, which is useful for maneuvering around a level during its construction.
------------------	---	-----	--	--

cheat_teleport_to_camera	N/A	N/A		This command will teleport the player to the current camera position.
--------------------------	-----	-----	--	---

cls	N/A	N/A		Clears the console screen.
-----	-----	-----	--	----------------------------

debug_objects	0	0-1		Enabling this command will give visual representations of the game objects or models bounding radii and collision models.
---------------	---	-----	--	---

debug_portals 0 0–1 Enabling this command will draw or outline the portal definitions in the game.

debug_sound 0 0–1 Enabling this command will give visual representations of the sounds being played and their audible radii along with labels indicating the specific sound being played to the player.

error_suppress_all 0 0–1 The suppression of error messages is by default on, disabling the error suppression (setting the value to 0) will show errors being encountered by the game and is useful in debugging content and game problems.

game_speed <#> 1 0–20 This command changes the entire speed of the game. Its very useful for pausing the game or travelling around a environment quickly during level creation or debugging.

NOTE: The value entered can go higher than 20 but higher values tend to be uncontrollable or problematic

hud_blink_health 0 0–1 Blinks the health indicator red.

hud_blink_shield 0 0–1 Blinks the border of the shield indicator.

hud_blink_motion_sensor 0 0–1 Blinks the entire motion sensor in green and red.

hud_show_health 1 0–1 Shows/hides the health indicator.

hud_show_shield 1 0–1 Shows/hides the shield indicator. Does not remove your shield.

hud_show_motion_sensor 1 0–1 Shows/hides the motion sensor.

hud_show_crosshair 1 0–1 Shows/hides the crosshair.

map_name <map name> N/A N/A Use this command to manually load a specific level.

net_graph_enabled 0 0–1 Displays a text graphic of the network activity.

players_unzoom_all N/A N/A Unzooms your weapon, if it is zoomed.

radiosity_quality 0 0–1 Sets the quality of the radiosity to be performed. See the section Radiosity (Lighting) for more details and information.

radiosity_start N/A N/A Starts radiosity lighting on the level using the current radiosity settings. See the section Radiosity (Lighting) for more details and information.

radiosity_save N/A N/A Saves the current radiosity lighting to a .bitmap lightmap file for the selected level (.scenario_structure_bsp). See the section Radiosity (Lighting) for more details and information.

rasterizer_environment_

diffuse_textures 1 0–1 This toggles the drawing of the diffuse textures for the world on or off, effectively displaying only the lightmap data for the level geometry.

rasterizer_fog_atmosphere 1 0–1 This command toggles atmospheric fog in the level on or off if the level has atmospheric fog set for the level in the tags.

rasterizer_fog_plane 1 0–1 This command toggles the fog plane in the level on or off if the level has a fog plane set for the level in the tags.

rasterizer_wireframe 0 0–1 This draws the current world scene including the world model (bsp) and all models such weapons, vehicles, characters and effects in wireframe mode, including non-visible geometry that has collision. This command is extremely useful in determining the placement of visibility portals, please see the section Level Creation for more details and information.

rasterizer_fps 0 0–1 This is the equivalent to rasterizer_stats 5.

rasterizer_stats 0 0–5

Each setting for rasterizer_stats displays the following:

0 - This disables the display of the rasterizer stats.

1 - Unsupported

2 - This is the most useful output option for rasterizer_stats. This setting displays a frame rate counter as well as categorized

vertice and triangle counts for the scene being rendered. Additional information such as the number of dynamic lights,

lens flares, decals and shadows are also displayed.

3 - Unsupported, used for Xbox version of Halo.

4 - Displays information on memory usage for the game.

5 - Simple frame rate counter and information such as maximum, minimum and average frame rates for the scene.

NOTE: Hitting the CTRL-F12 key combination will bring up a frame rate counter and graph.

rider_ejection 0 0–1 Stops the player from being ejected of the vehicle when it flips.

quit N\A N\A Quits the game

[Return to Main Page](#)

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print](#) [history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

[Halo](#) / Weapons Ammo Items List

- view
- edit
- attach
- print
- history

Weapon\Ammo\Items List

The following section is a quick reference list of all weapons, associated ammo, and items by game name and tag name\location.

Single Weapons

The following are the list of available weapons for placement in multiplayer Halo PC levels. Halo weapons are placed as item_collections with an item collection being either a single weapon or a weapon group.

The following list is for the placement of ONLY the specified weapon.

The following item collection tag references are the tag subdirectory under the Halo main directory (i.e. C:\Halo\tags\item collections\single weapons\assault rifle.item_collection).

Weapon Name Description Item Collection Tag
Human Weapons

- M A5B Assault Rifle Assault Rifle item collections\single weapons\assault rifle.item_collection
- M6D Assault Pistol Pistol item collections\single weapons\pistol.item_collection
- M90 Shotgun Shotgun item collections\single weapons\shotgun.item_collection
- S2 AM Sniper Rifle Sniper Rifle item collections\single weapons\sniper rifle.item_collection
- M199 SSM Rocket Launcher Rocket Launcher item collections\single weapons\rocket launcher.item_collection
- M7057 Defoliant Projector Flamethrower item collections\single weapons\flame thrower.item_collection
- M9 HE-DP Frag Grenade Frag Grenade item collections\single weapons\frag grenades.item_collection

Covenant Weapons

- Plasma Pistol Plasma Pistol item collections\single weapons\plasma pistol.item_collection
- Plasma Rifle Plasma Rifle item collections\single weapons\plasma pistol.item_collection
- Needler Needler item collections\single weapons\needler\needler.item_collection
- Fuel Rod Gun Fuel Rod Gun item collections\single weapons\plasma_cannon.item_collection
- Plasma Grenade Plasma Grenade item collections\single weapons\plasma grenades.item_collection

Weapon Groups

The following are the list of available weapons for placement in multiplayer Halo PC levels. Halo weapons are

placed as item_collections with an item collection being either a single weapon or a weapon group.

The following list is for the placement of weapon groups in which a list of weapons are specified and spawn or alternate their spawning in the game.

The following item collection tag references are the tag subdirectory under the Halo main directory (i.e. C:\Halo\tags\item collections\single weapons\assault rifle.item_collection).

Weapon Group Name Group Contains Item Collection Tag

Group All Weapons Assault Rifle, item collections\weapon groups\group all weapons.item_collection

Pistol,

Shotgun,

Sniper Rifle,

Rocket Launcher,

Flamethrower,

Plasma Pistol,

Plasma Rifle,

Needler,

Plasma Cannon (Fuel Rod Gun)

Group Heavy Weapons Flamethrower, item collections\weapon groups\group heavy weapons.item_collection

Rocket Launcher,

Plasma Cannon (Fuel Rod Gun)

Group Pistols Pistol, item collections\weapon groups\group pistols.item_collection

Plasma Pistol,

Needler

Group Rifles Assault Rifle, item collections\weapon groups\group rifles.item_collection

Plasma Rifle,

Shotgun

Group Special Weapons Flamethrower, item collections\weapon groups\group special weapons.item_collection

Rocket Launcher,

Sniper Rifle,

Plasma Cannon (Fuel Rod Gun)

Items

The following are the list of available items for placement in multiplayer Halo PC levels. Halo items are placed as item_collections with an item collection being either a single item or a group of items. Items refer to objects such as powerups

The following list is for the placement of ONLY the specified item.

The following item collection tag references are the tag subdirectory under the Halo main directory (i.e. C:\Halo\tags\item collections\powerups\powerup health pack.item_collection).

Weapon Group Name Description Item Collection Tag

Powerup Health Pack Health Pack item collections\item collections\powerup health pack.item_collection
Powerup Invisibility Active Camouflage item collections\item collections\powerup invisibility.item_collection
Powerup Super Shield Overshield item collections\item collections\powerup super shield.item_collection

Item Groups

The following are the list of available items for placement in multiplayer Halo PC levels. Halo items are placed as item_collections with an item collection being either a single item or a group of items. Items refer to objects such as powerups

The following list is for the placement of item groups in which a list of items are specified and spawn or alternate their spawning in the game.

The following item collection tag references are the tag subdirectory under the Halo main directory (i.e. C:\Halo\tags\item collections\powerups\powerup health pack.item_collection).

Weapon Group Name Group Contains Item Collection Tag

Group All Powerups Active Camouflage, item collections\item collections\group all powerups.item_collection
Overshield,
Health Pack

Group Shield and Active Camouflage, item collections\item collections\shield-invisibility.item_collection
Invisibility Overshield
Return to Main Page

- view
- edit
- attach
- print
- history

Level Exporting

The following section provides information and a tutorial on the export process from 3ds Max to the .jms format which is used to compile a Halo game level.

Exporting The Level

The following procedures and example images demonstrate using the Blitzkrieg plug-in in 3ds Max to export the .jms file for the level. The .jms file is used by Tool to compile the level data into the .scenario and .scenario_structure_bsp data tags.

It is these data tags will be manipulated to create a working multiplayer level that will then be compiled into a .map cache file and run in Halo.

Exporting the level using Blitzkrieg:

In 3ds Max open the tutorial.max file that was created in the previous tutorials.

1) To export the level using the Blitzkrieg exporter plug-in follow the procedures below:

a) Click on File and select Export. The Select File to Export window will appear.

b) Select the same directory in which the tutorial.max file exists (Halo Custom Edition\data\levels\test\tutorial\models), this is the directory created in the section Creation of a Level Directory under the Creating a Level section in Level Creation - Part 1.

c) In the Save as type: pull down select Blitzkrieg Model Exporter (*.JMS,*.JMP).

d) In the File name: field enter the name tutorial.jms.

e) Click on Save.

The level should have exported successfully.

The first image further explains the above procedures.

[Click to Open Larger Image in New Window](#)

Troubleshooting the Export Process

The following table outlines some common problems or errors that may occur when exporting the .jms for the level and provides suggestions or solutions that should help in the successful export of the level using the Blitzkrieg exporter plug-in.

Problem or Error Suggestions or Solution

When exporting using Blitzkrieg an error dialog appears that says “There was no geometry to export”. Make sure that the level has a reference frame. When exporting using Blitzkrieg an error dialog appears that says “There was no geometry to export”. If the level has a reference frame, make sure that there is valid geometry present and that the geometry is attached to the reference frame. When exporting using Blitzkrieg the .jms file does not appear to have exported and been saved. Make sure that the .jms file is being saved to the correct directory. When exporting with Blitzkrieg using an existing .jms file name, the .jms file does not appear to have saved or updated the existing file. Make sure that the existing .jms file is not set to Read-only. When exporting using Blitzkrieg 3ds Max crashes.

This is usually related to some strange geometry construction error or illegal material assignment.

Try the following steps below to track down the error.

- 1) Systematically unlink objects from the reference frame and try to export until the level exports successfully. This can help determine which object and related geometry are causing export problems.
- 2) If an object has a Multi/Sub-Object applied to it, make sure that it is applied to the entire object.
- 3) Make sure that all faces have a Multi/Sub-Object Sub-Material or Standard material applied to them.
- 4) Make sure that the mesh geometry does not have isolated (unused) vertices.
- 5) Make sure that there are no double faces or polygons (faces or polygons that occupy the exact same space and share the same vertices).
- 6) Check for double edges or illegal faces or polygons.

Conclusion to Level Exporting

The level geometry has now been successfully exported and is ready to be compiled in preparation for it to go through the processes that will get it running in Halo.

As previously mentioned, a completed version or example version of the level created in the tutorial sections can be found under the “Halo\data\levels\test\tutorial_examples” directory. The source and related materials have been provided as a reference to aid in the learning process. The successfully exported tutorial.jms file for the example level has also been made available.

Once the user has successfully completed the export process they can proceed to the next section Level Compilation.

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

Halo /

Manipulationin Guerilla

- view
- edit
- attach
- print
- history

Level Data Manipulation in Guerilla

The following section provides a tutorial on using Guerilla and information on this tool and what it is used for in relation to Multiplayer Levels.

Introduction to Guerilla

Guerilla is a tool that has a graphical interface that is used to edit the properties and tag resource references in the Halo data “tags”. Every data tag in the game can be opened and viewed in Guerilla. Not all the data and attributes are allowed to be edited (some are locked to prevent accidental modification that would result in errors).

Guerilla is a very straightforward program. Tags are opened in individual windows that can be arranged or resized as needed.

To find out more information about a particular field or setting, hover the mouse cursor over the field. A description label will appear giving some insight into what the value, reference field, or attribute does. Description or help labels do not exist for all settings, fields, and attributes.

Tags can be loaded normally by going to File → Open or tags can be “dragged and dropped” onto Guerilla to open them.

Setting Level Attributes Using Guerilla

Now that the level has been successfully compiled and the tutorial.scenario and tutorial.scenario_structure_bsp files have been created, these data tags can be opened in Guerilla and have certain attributes set that will affect the properties and behavior of the game level.

The majority of work done to the tutorial.scenario and tutorial.scenario_structure_bsp data tags are done using Sapien, however there are certain attributes that cannot be accessed using Sapien and instead must be modified using Guerilla.

The following tutorials and examples will demonstrate the uses for Guerilla in relation to the .scenario and .scenario_structure_bsp data tags. Guerilla will be used to set the sky model to use for the level, the relative

North direction, and the type of level (single player or multiplayer).

The setting of certain attributes for the .scenario tag using Guerilla:

Go to the main Halo directory. In the root directory is where all the Halo tools exist, there should be a program file called guerilla.exe.

1) Run Guerilla and open the tutorial.scenario tag.

A window displaying all the tag information for the tutorial.scenario tag should now appear.

2) Since the tutorial level is an outdoor level, a sky model must be referenced in order for a sky to render properly in the game. In addition, the .sky tag contains information that will be used by radiosity to light the level. To reference a sky model follow the procedures below:

a) At the top of the tag information there is a section called “Skies”.

b) Click the Add button. A sky reference has now been added but the specific reference must be defined.

c) In the sky field reference a file by clicking on the “...” button. A file Open dialog window will appear.

d) The majority of skies that can be used are found under the main tags directory under the “sky” subdirectory.

Go to this directory. There should be several subdirectories, several .skies have been included in the Halo Editing Kit.

For the tutorial level, the sky_timberland.sky will be used. Enter the sky_timberland subdirectory.

e) Select the data tag sky_timberland.sky.

f) Hit Open.

The tag sky\sky_timberland\sky_timberland should now appear in the “sky” field.

To view the sky_timberland.sky tag and its associated values click on the Open button located to the right of the “sky” field.

If the level requires different settings than what exist in the .sky tag do NOT modify the existing .sky tag if it is used by other levels or those levels will be affected as well. The proper procedure is to create a new custom .sky tag.

The first image further explains the above procedures and shows the final results of the above actions.

3) The level must be defined as a multiplayer level. To set the level up as a multiplayer level go to the “type” pulldown menu under the Skies section and select “multiplayer”.

Setting the level as multiplayer is important since this tells Tool and the game what resources should be included during the .map cache file creation process.

4) Next, the relative North for the level will be set. This setting affects the direction that the compass on the Assault Rifle points to.

Under the section Child Scenarios, enter the angle value 90 in the “local north” field. The tutorial level is rotated 90 degrees, thus putting the intended North end of the level at this same angle.

The second image further explains the above procedures and shows the final results of the above actions.

5) The default weapons and equipment for the player will now be defined. The default or starting weapons can be defined specifically for this level for each different game type.

a) Find the section “Starting Equipment”. This section is located about halfway down the tag data.

b) Three different profiles will be added. Click on the Add button 3 times.

c) Using the pull down menu select the “0. scenario_starting_equipment” settings.

Set the following settings for Capture the Flag (CTF).

d) For the “type 0” field use the pull down to select “ctf”. This setting tells the game that when CTF game variants are run to use the item collections defined below.

For CTF, the default weapons will be the Pistol, Assault Rifle and Frag grenades. The player will spawn with these weapons every time the game first starts or when they are eliminated and respawn.

e) In the “item collection 1” field hit the “...” button. An Open file window will now appear.

f) Go to the “item collections” tag directory and then choose the “single weapons” tags subdirectory.

g) Select the “assault rifle.item_collection” tag file.

h) Click on Open.

i) In the “item collection 2” field follow the same procedures but select and add the “pistol.item_collection”.

The player WILL spawn with standard Frag grenades in this game mode, so the grenade check boxes will not be touched (these are located under the “flags” section at the top of the “Starting Equipment” section).

Item collections (.item_collection data tags) contain single weapons and items or combinations of weapons and items that can be referenced in order to place weapons and items (such as powerups) in a level.

The third image further explains the above procedures and shows the final results of the above actions.

6) The next settings for default weapons will be made for ALL game types or game variants except those for CTF and Race modes.

Games that run these game modes will have the player spawn with just the Pistol and Frag grenades.

a) Using the pull down menu select the “1. scenario_starting_equipment” settings.

b) For the “type 0” field use the pull down menu to select “all except race & ctf”. This setting tells the game to use the item collection settings below for all game types other than CTF and Race modes. The Slayer, Oddball and King of the Hill modes will use the item collection settings defined below.

c) In the “item collection 1” field hit the “...” button. An Open file window will now appear.

d) Go to the “item collections” tag directory and then choose the “single weapons” tags subdirectory.

e) Select the “pistol.item_collection” file.

f) Click on Open.

The player WILL spawn with standard Frag grenades in these game modes, so the grenade check boxes will not be touched (these are located under the “flags” section at the top of the “Starting Equipment” section).

The fourth image further explains the above procedures and shows the final results of the above actions.

7) The next settings for default weapons will be made for game types that are based on Race mode.

Games that run the Race game modes will have the player spawn with just the Plasma Pistol and NO grenades of any type.

a) Using the pull down menu select the “2. scenario_starting_equipment” settings.

b) For the “type 0” field use the pull down to select “race”. This setting tells the game to use the item collection settings below for all game types that use Race mode.

- c) In the “item collection 1” field hit the “...” button. An Open file window will now appear.
- d) Go to the “item collections” tag directory and then choose the “single weapons” tags directory.
- e) Select the “plasma pistol.item_collection” file.
- f) Click on Open.
- g) The player will spawn with NO grenades in this game mode.

Under the “flags” section at the top of the “Starting Equipment” section check the “no grenades” option box.

Once the above procedures have been completed, save the tutorial.scenario tag. This can be done by going to File → Save.

The fifth image or last image further explains the above procedures and shows the final results of the above actions.

As objects are added in Sapien, additional data will be added to the .scenario tag. This data can be viewed using Guerilla. Using Guerilla in this manner can help track down errors or view resources used in the level.

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

For the tutorial level, no modifications to the tutorial.scenario_structure_bsp using Guerilla are necessary. However, there are some fields that are commonly used that will be quickly discussed in the section below.

Taking a look at the .scenario_structure_bsp data tag.

- 1) In Guerilla and open the tutorial.scenario_structure_bsp tag.

A window displaying all the tag information for the tutorial.scenario_structure_bsp tag should now appear.

2) At the top of the file is an entry for “lightmaps”. Since radiosity has not been run on the level, this field will be blank. When radiosity for the level is run, the resulting lightmap .bitmap for the level will automatically be entered in this field.

3) Below the “lightmaps” field are two fields relating to vehicles.

The “vehicle floor” value limits the distance in the negative Z-Axis that a vehicle can travel. When the vehicle hits this distance, it will stop or collide with this invisible plane. The player and other objects can still pass through this plane.

The “vehicle ceiling” value limits the distance in the positive Z-Axis that a vehicle can travel. When the vehicle hits this distance, it will stop or collide with this invisible plane. The player and other objects can still pass through this plane.

Both of these fields are very useful in constraining vehicles and preventing them from breaking the game play of a level. The “vehicle ceiling” value is especially useful in containing flying vehicles such as the Banshee. This is how the Banshee is contained in Z-Axis in multiplayer levels, such as Death Island and Gehphyrophobia.

To constrain vehicles in the horizontal or XY plane, player or vehicle clipping techniques such as those covered in the tutorial sections should be used. Note that player and vehicle clipping techniques can also be used to constrain the travel of vehicles in the Z-Axis also.

Since the tutorial level is so small and no flying vehicles will be included in it, these values are being left at their default values.

4) Below the vehicle ceiling and floor fields are a set of values used to control various ambient lighting effects and attributes in the level.

The image to the right further explains the above information.

In addition, the data in the scenario_structure_bsp can be viewed using Guerilla. Using Guerilla in this manner can help track down errors or view resources used in the level.

[Click to Open Larger Image in New Window](#)

Conclusion to Level Manipulation in Guerilla

The level data tags have been manipulated such that they are now ready to be lit using radiosity and loaded into Sapien.

Completed versions or example versions of the level tags that are created in the tutorial sections can be found under the “Halo\tags\levels\test\tutorial_examples” directory. The compiled and modified tags have been provided as a reference to aid in the learning process.

Once the user has successfully completed the above tutorials and examples they can proceed to the next section Radiosity (Lighting).

*

Continue to the Next Section - Radiosity (Lighting)

*

Return to Main Page

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

Radiosity (Lighting)

The following section gives a quick overview of the radiosity lighting solution used in Halo. Topics and tutorials include running the radiosity process on the level as well as information on the settings and options for radiosity, the creation of a lightmap for the level (.bitmap) and troubleshooting common problems encountered with lighting in Halo.

Radiosity Overview

Halo has both static and dynamic lighting. The static lighting is created by running a radiosity process on the level geometry that then saves the results to a light map file that is just a special kind of .bitmap data tag.

The light used by the radiosity process can come from many sources. The sky box or .sky data tag has numerous lighting settings and attributes that can be set. Materials have radiosity properties that can be set to emit light that can be used in the radiosity process. These materials can be applied to surfaces in the level environment. Models can be setup to emit light using .light data tags as well.

The effect of the radiosity process can be controlled for almost every face or surface in the game. In the material or shader settings there is a value called “detail level” which exists under the “radiosity properties” area when the tag file is viewed in Guerilla. This detail level setting affects the amount of tessellation or how much the surface is subdivided for the lightmap. The higher the setting, the higher the resolution of the lightmap on the surface and the longer the radiosity process takes. In addition, higher settings increase the amount of lightmap data (the .bitmap gets much larger) and thus affects the amount of memory used.

On a related note, just as there is static and dynamic lighting, there are also static and dynamic shadows. Dynamic shadows are cast from objects such as vehicles and characters. Static shadows of course are a result of the radiosity process. Static models such as those used for .scenery objects can be setup to cast shadows or block light during the radiosity process.

The radiosity process can be run using Tool or in Sapien. Sapien has a developer console which can execute many of the commands that can be run in the Halo game console.

The following examples and tutorials will discuss the radiosity command options and demonstrate running the radiosity process on the tutorial level using Sapien.

Running Radiosity on a Level

The following examples and tutorials will discuss the radiosity command options and demonstrate running the radiosity process and saving the lightmap data to a .bitmap on the tutorial level using Sapien.

Lighting the tutorial level (running radiosity on the tutorial level):

Go to the main Halo directory. In the root directory is where all the Halo tools exist, there should be a program file called sapien.exe.

1) Run Sapien. When Sapien loads, an Open dialog window will appear. Use this dialog to load the tutorial.scenario tag.

Sapien will now load. Sapien runs many of the same systems as the game so give it time to load, especially for larger levels.

The first image further explains the above procedures.

Maximize or make Sapien go fullscreen. Roughly organize the windows so that there is easy access to the Game Window. The Game Window is the 3D window which renders many of the Halo game objects exactly as they will appear in game. The 3D window runs the same rendering engine as Halo itself.

Sapien will be covered in more detail in the section Population. For now, all that matters is the large 3D window.

2) Use the mouse pointer to click on the Game Window. In the game window, bring up the developer console by hitting the ~ key.

When the game console is visible, the game and input controls for the camera are paused.

There are 3 console commands that will be used to run and complete the radiosity process.

radiosity_quality <0 or 1> - Sets the quality of the radiosity to be performed.

A setting of 0 runs what is called a quick or “fast radiosity”. The number of “light bounces” or “radiosity bounces” that are run is fewer and the process ignores light occlusion or blocking caused by models. As a result the radiosity process does not take as long and is less accurate and the resulting lightmap .bitmap is smaller.

A setting of 1 runs what is called a “full radiosity”. This is the process that should be run for the final or “release” version of the level. A full radiosity runs the radiosity process at its full settings. As a result the radiosity process takes much longer and is much more accurate. The resulting lightmap .bitmap is much larger.

`radiosity_start` - Starts the radiosity lighting process on the level using the current radiosity settings.

`radiosity_save` - Saves the current radiosity lighting to a .bitmap lightmap file for the selected level (.scenario_structure_bsp). This command can be used at anytime after the `radiosity_start` command is used. It will stop or interrupt the process and save the current progress to the lightmap .bitmap file. As a result, any radiosity process (even a fast or quick radiosity process) that takes a long time can be cut short, saving time. If this command is used upon the completion of the radiosity process, it behaves as normal and saves the lightmap .bitmap file out.

For more information on the console commands that will be used as well as console commands in general, see the Console Commands table in the Console Command List section under General Reference.

Since the tutorial level is small and radiosity will not take a long time to run, a “full radiosity” will be used.

Type the following into the console and hit Enter:

```
radiosity_quality 1
```

Note that at anytime the values for the radiosity settings can be checked by typing in the console command without any setting or value and hitting Enter. This will return the current value or setting.

The second image further explains the above procedures.

3) Now type the following into the console and hit Enter:

```
radiosity_start
```

When this command is executed 3 numbers should appear in the Game Window. These numbers represent the RGB values for the lighting. These numbers will countdown as the process continues, when all 3 numbers reach 0.0, the radiosity process is complete.

The third image further explains the above procedures.

4) When the 3 displayed values reach 0.0 type the following in the console and hit Enter.

```
radiosity_save
```

The fourth image further explains the above procedures.

The level should now be visible. If it is not, move the camera around in the Game Window until the level is visible. The camera can be moved and rotated by holding down the Middle Mouse button and using the following keys to move around the level:

The mouse controls the pitch (up and down) and yaw (left and right) movement of the camera.

Shift - Cycles through the camera speed, making it move faster.

W - Moves the camera forward

A - Moves the camera backwards

S - Pans the camera left

D - Pans the camera right

R - Moves the camera along the Z-Axis in the positive direction (up).

F - Moves the camera along the Z-Axis in the negative direction (down).

The fifth image or last image further explains the above procedures and shows the results of the radiosity process. The level should now be fully visible.

[Click to Open Larger Image in New Window](#)

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Click to Open Larger Image in New Window

Troubleshooting Errors in Radiosity

The following table outlines some common problems or errors that may occur as a result of or in relation to the radiosity process and offers suggestions and solutions to these problems.

Problem or Error Suggestions or Solution

When the level is viewed in Sapien or in Halo it is completely dark.

Make sure that the level has lights and that radiosity has successfully completed and created a lightmap .bitmap.

Remember that to save the radiosity solution the lightmap must be manually saved using the radiosity_save command. The lighting in the level is very dark or darker than expected.

There are several problems that can cause this, try the following solutions:

If the radiosity process was stopped prematurely, try running the radiosity process longer.

Run a full radiosity (radiosity_quality 1). A full radiosity bounces the light more, therefore the level is lit better.

Check the light value settings and attributes in the appropriate tag files. The lighting on certain surfaces is very low resolution (its “blocky” or “splotchy”)

There are several problems that can cause this, try the following solutions.

Use Guerilla to check the “detail level” setting in the “radiosity properties” area in the shader tag. If the value is set to low or “turd” try setting it to medium or high to see if this smooths the lighting on the surface.

Check the Smoothing Group settings in 3ds Max for the faces or surfaces.

Try increasing the amount of light that is falling on the surface. Either increase the light values or add additional lighting that further illuminates the problematic surfaces. There are black lines or patches on surfaces in the level.

There are several problems that can cause this, try the following solutions.

Check the Smoothing Group settings in 3ds Max for the faces or surfaces.

Make sure that there are no duplicate Smoothing Groups being applied to multiple faces or objects.

Make sure that the Smoothing Groups are not applied to faces that are separated or exist far apart from each other.

Try increasing the amount of light that is falling on the surface. Either increase the light values for the surrounding lights or add additional lighting that further illuminates the problematic surfaces.

Make sure there are no errors or warnings being reported by Tool during the compilation process. Some of these errors can cause lighting anomalies.

Increase the amount of faces, subdivide the surfaces. Increasing the density of the mesh may help smooth the lighting, this is especially true of very large faces or polygons.

Modify or adjust the geometry to see if it fixes the errors. The surfaces are extremely bright or fullbright. The surface appears to be completely saturated with light.

There are several problems that can cause this, try the following solutions:

Check the light value settings and attributes in the appropriate tag files. Make sure these values are not set too high.

Check the self-illumination properties and settings in the appropriate tag files. If self-illumination was not intended, set the associated values to 0.

Conclusion to Radiosity

The level has now had the radiosity process successfully completed. The level is now ready for object placement and editing using Sapien

Completed versions or example versions of the level tags that are created in the tutorial sections can be found under the “Halo\tags\levels\test\tutorial_examples” directory. The compiled tags have been provided as a reference to aid in the learning process.

Once the user has successfully completed the procedures above to successfully light the level they can proceed to the next section Population.

*

Continue to the Next Section - Population

*

Return to Main Page

Powered by [PmWiki](#)

[view](#) [edit](#) [attach](#) [print history](#) | [Halo](#) | [Recent Changes](#) | [Search Wiki](#)

Placement of Objects

- [view](#)
- [edit](#)
- [attach](#)
- [print](#)
- [history](#)

Population

The following section covers the population or placement of objects in the level using Sapien. Tutorials and information cover an overview of the Sapien tool, player spawn point placement and information, scenery object implementation and placement, vehicle and weapon implementation and placement, netgame flags information and placement, and the manipulation and setting of BSP portal data including fog, sounds and weather effects.

Sapien Overview

Sapien is used to place objects in the level. The placement of objects in a level is sometimes referred to as “population”. The objects that are placed in the level include objects for aesthetics and game play (scenery objects such as trees, rocks, bushes, etc..) as well as objects that are required for the level to run and operate properly in the game (netgame flags, netgame equipment, spawn points, etc..).

Sapien has a graphical interface that contains a 3D window (the Game Window) that renders the game including the level bsp, models, and vehicles. The Game Window also runs other game systems as well and plays sounds and runs physics.

Sapien does not have any form of “undo” functionality. Therefore, it is HIGHLY recommended that the .scenario tag be saved and saved often. Saving off sequential versions of the file is also HIGHLY recommended (i.e. tutorial_01.scenario, tutorial_02.scenario, etc...)

Halo has a lot of objects that can be used in a level. These objects all exist as a data tag. To help in the organization of objects or tags for placement in the game level using Sapien, a subset of objects or tags is created for certain object types or groups in the hierarchy. This specific set of tags is called a “palette”. Objects or tags are then selected from the palette for placement in the level. The palette can have items added or removed at any time. Removal of an item reference from the palette does NOT remove the item marker from the level, instead the item is set to NONE so that the item will no longer appear in the level. This empty item marker should be removed from the level.

The images and discussions below will explain the basics of the Sapien Graphical User Interface.

Overview of the Sapien Graphical User Interface:

Go to the main Halo directory. In the root directory is where all the Halo tools exist, there should be a

program file called sapien.exe.

Run Sapien. When Sapien loads, an Open dialog window will appear. Use this dialog to load the tutorial.scenario tag.

Sapien will now load. Sapien runs many of the same systems as the game so give it time to load, especially for larger levels.

The tutorial level should now appear in the Game Window.

Click to Open Larger Image in New Window

The following descriptions will point out a particular section or

Running a Game Level in Halo

The following section provides information and a tutorial on preparing a level to run in a Halo multiplayer game. The information and tutorial covers the creation of a .map cache file that contains the level and all tag resources needed to run the level in the game.

Cache File Creation

A .map cache file contains all the data tag resources need to run the level. Because of this, the level is completely self contained and can be distributed very easily. All that needs to be distributed is the .map cache file and other users have everything needed to play the level.

The following procedures and example images demonstrate the creation of a .map cache file using Tool.

Creating the tutorial.map cache file using Tool:

In Windows, open a Command Prompt. The Command Prompt can be found by going to Start → Accessories → Programs → Command Prompt

In the Command Prompt go to the main Halo directory. In the root directory is where all the Halo tools exist, there should be a program file called tool.exe.

Typing just tool.exe at the Command Prompt and hitting Enter will show all the available command line parameters as well as the usage and syntax options.

The command line that will be used in the following example is “build-cache-file” and it has the following syntax:

build-cache-file <scenario-name>

The syntax parts are described below:

scenario-name - The scenario name includes the name of the directory where the level directory structure is

located, as well as the name of the .scenario tag file. The directory structure that was set up for Halo editing is very specific. The full directory structure name is not needed because of this. The file extension name is also NOT necessary.

1) At the Command Prompt type the following and hit Enter:

```
tool.exe build-cache-file levels\test\tutorial\tutorial
```

There should be quite a bit of output indicating the processes being performed during the creation of the .map cache file.

The output will also indicated any errors that occur. These errors typically take the form of missing data tag files. Depending on the severity of the error, the .map cache file may not get created.

The tutorial.map cache file should have been successfully created.

The first image further explains the above procedures and shows the output of Tool.exe in the command window.

The tutorial.map cache file should now appear in the Halo\maps directory. All cache files for the game are located in this directory.

[Click to Open Larger Image in New Window](#)

Running the Level in a Halo Multiplayer Game

Halo has 2 different modes that it can run in, one mode for playing the game normally in both single player and multiplayer, and one mode that is a development mode which is extremely useful when creating and debugging a multiplayer level. The 2 different modes are described in more detail below.

The first mode is the Normal Mode or default mode that is run when just Halo.exe is executed. This mode has limited console commands available as well as fully functional multiplayer. This is a “safe” or “secure” mode which is why developer or debug functionality is limited. This helps prevent cheating or other exploits. For more information on the console and -console command line parameter refer to the Console Commands section located in the Console Command List section under General Reference category.

The second mode is a Developer Mode. Developer Mode is enabled by adding the Command Line Switch “-devmode” in the command line after halo.exe (i.e. halo.exe -devmode). The Halo developer mode enables a wide variety of console commands or cheats that are extremely useful when creating new Halo content. The developer mode also makes several changes to how Multiplayer games work. When in Developer Mode, only a LAN server can be created and the maximum number of players allowed is 1 (just the end user). This allows the level to be tested to see if it operates properly and allows the developer console commands to be used to help debug the level. Also, when in Developer Mode, the end user cannot connect to any servers, either LAN or Internet servers. In order to properly test the level with other players, just run Halo normally. For more

information on the console and console commands refer to the Command Line Switches section located in the Console Command List section under General Reference category.

The following procedures and example images demonstrate accessing end user created multiplayer levels in Halo.

Accessing an end user created multiplayer level in Halo.

The following procedures apply to loading Halo in either Normal Mode or Developer Mode.

The procedures below will demonstrate how to load the tutorial level that was created in the tutorial sections for Multiplayer Level Design under Level Design.

- 1) Load Halo.
- 2) Select Multiplayer from the main menu.
- 3) In the Multiplayer options screen, choose LAN under Create Game.
- 4) In the Select Map screen, scroll down until the level is found. In this case find the “tutorial” level and click on it.
- 5) Select OK.
- 6) Now select any game type in the Game Type screen and select OK.
- 7) Set the server options in the Server Setup and select Start Game.
- 8) The tutorial level should now load.

The image to the right shows the Select Map screen and the tutorial level being displayed.

User created levels do not have a map description and image that display for the level when it is selected.

Click to Open Larger Image in New Window

* Return to Main Page