Cafu Engine Documentation

Carsten Fuchs and others

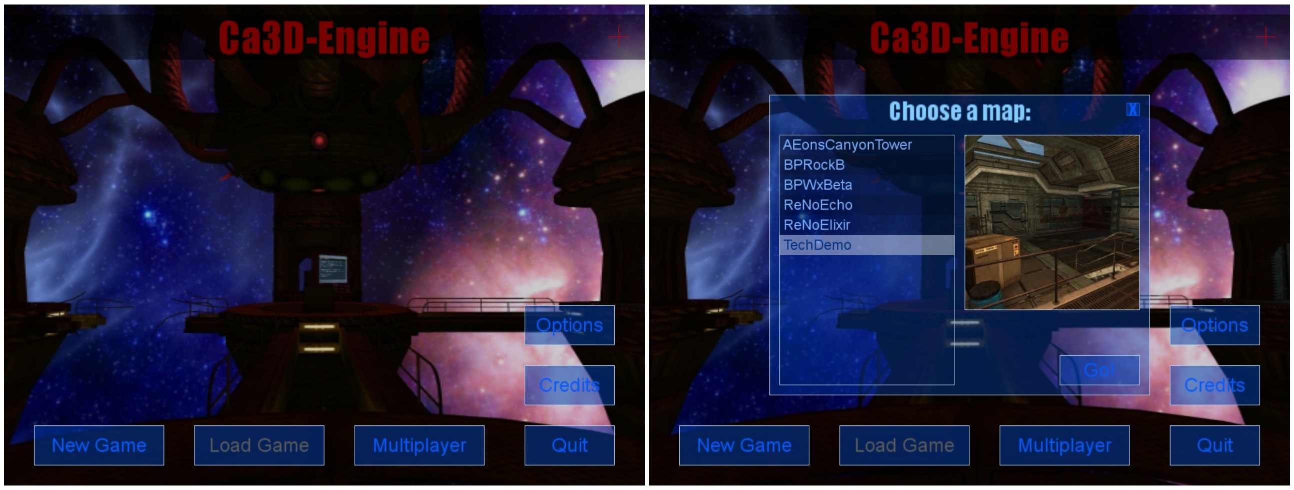
Feb 11, 2020

User Documentation

CHAPTER 1

Quick Start

The User Documentation describes the *Cafu Engine Demo* packages that are released periodically to demonstrate the features of the Cafu Engine.



This section is intended for advanced users who are experienced with downloading and installing software. If you are in a hurry and don't want to study the entire document, but still want to make sure that you don't miss any of the important Cafu features, you will find everything here to get you started:

- Installation is reduced to unzipping the downloaded file.
- Double-clicking on `Cafu.exe` starts the program.
- In the program main menu click on New Game, then select a map in the following screen and click Go! to start the game.
- The `ESC` key opens the main menu again, where you can change game options or quit the game. The `F1` key opens the in-game console.
- If you have questions, please consult the rest of this documentation first. If you can't find your answer here, please join and post a message at the Cafu forums – we're happy to help!

CHAPTER 2

Installation

2.1 Minimum System Requirements

Cafu has been designed to run on a very broad range of hardware. Therefore, meeting or failing a certain set of minimum system requirements is not necessarily a question of getting all-or-nothing. Instead, the details of a systems hardware usually just scale the performance and/or the quality that the Cafu engine can achieve on that particular system – it even has automatic fall-back capabilities built-in in order to handle old and very old hardware well.

Your computer system should meet the following *minimum requirements* for Cafu to run (failing these requirements probably means that Cafu does not run at all):

- Intel i386 compatible CPU with 2 GHz
- 1 GB RAM (2 GB with Windows Vista or 7)
- 3D graphics board with OpenGL support and 256 MB graphics RAM
- Windows (2000, XP, Vista, or 7) or Linux operating system
- TCP/IP (network drivers)
- On Windows: DirectX 7.0 or higher

Systems that do not meet these requirements may still work fine with Cafu, but both the main processor and/or the graphics processor RAM may easily be exceeded by loading larger maps with many textures, resulting in very bad performance. You can use the Cafu Options dialog then to reduce the texture quality and/or load smaller maps.

In addition to the above, here is a list of *recommended system features* that allow Cafu to run reasonably well (in all cases, more is better):

- Intel i386 compatible CPU with 3 GHz
- 2 GB CPU RAM
- contemporary 3D graphics board with OpenGL support, programmable GPU and 256 MB graphics (GPU) RAM

In order to use the advanced networking features of Cafu, the network environment of your system must be appropriately configured.

In other words, if some of the modern commercial 3D computer games run on your system, then this is a good indication that Cafu will work, too.

We are developing Cafu under Windows and Ubuntu Linux, and test it on as many platforms as we can get access to. We have only little experience with other Linux distributions, but have received only positive reports up to now, regarding compatibility with all supported operating systems.

2.2 Installing Cafu

In order to install Cafu, simply unzip the packed file that you downloaded from the website. Please make sure that the directory structure is preserved. This is the common extraction process for nearly all contemporary decompression software. However, for some older programs you need to explicitly specify a command line option or check a checkbox. A new directory like `Cafu-9.06` will be created and all files will be extracted into this directory.

2.3 Upgrading

In order to upgrade the engine from an older, existing installation, please delete all the old files and directories and then re-install the new version as stated above. We are sorry for any inconvenience caused by this, but this is the best way to avoid many possible upgrade and version conflict problems.

2.4 De-Installation

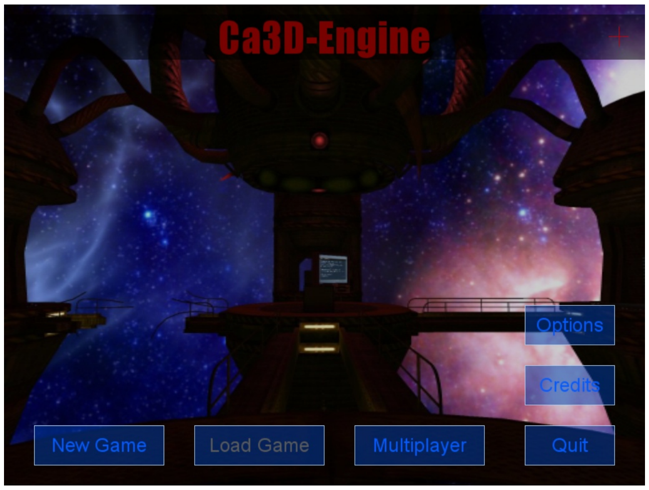
Because the installation makes no changes to your system, it is sufficient to just delete the `Cafu-x.y` folder for a completely clean de-installation.

CHAPTER 3

Running Cafu

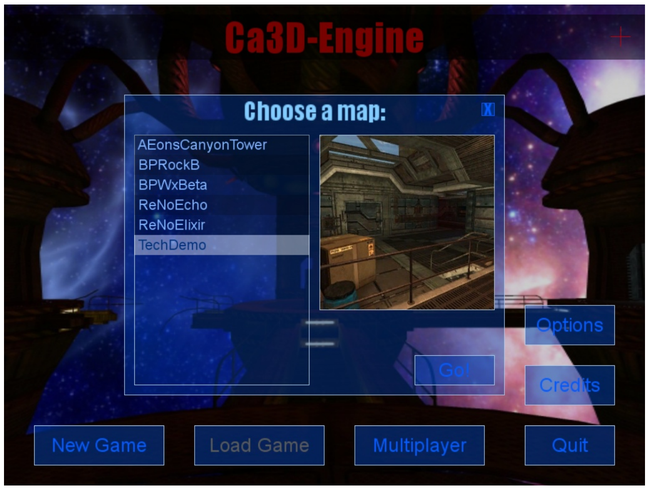
In order to start Cafu, simply double-click on the `Cafu.exe` executable file. This file is in the `Cafu-x.y` folder that was created during installation.

3.1 The Main Menu



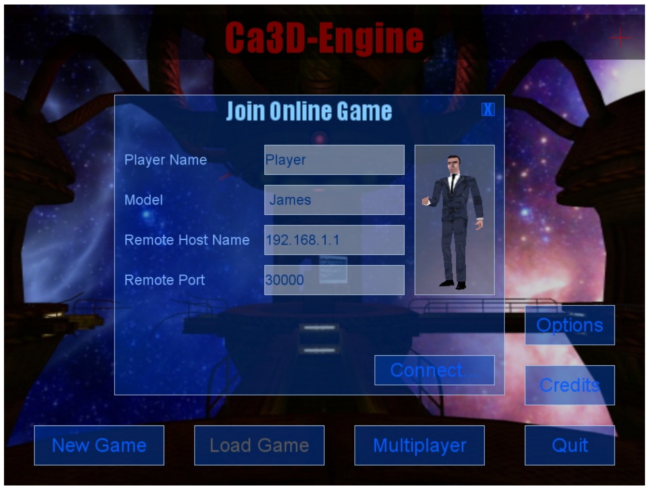
After starting, the *Main Menu* appears as shown in the figure. From here you can configure Cafu's graphic, audio and control options, start new games, join multiplayer games and even host your own multiplayer server. *To start playing immediately just click* New Game*, select a map to play and click* Go! *to play this map on your local machine.* For hosting multiplayer games or joining them please read the following sections.

3.1.1 New Game



This dialog allows you to quickly start a new game. You can choose a map from the list on the left side of the screen. If available a preview screenshot of the map is shown on the right side. To start the game you just have to click on Go! and the the engine will automatically host a local game server with the chosen map and connect you as a client to it.

3.1.2 Join Online Game



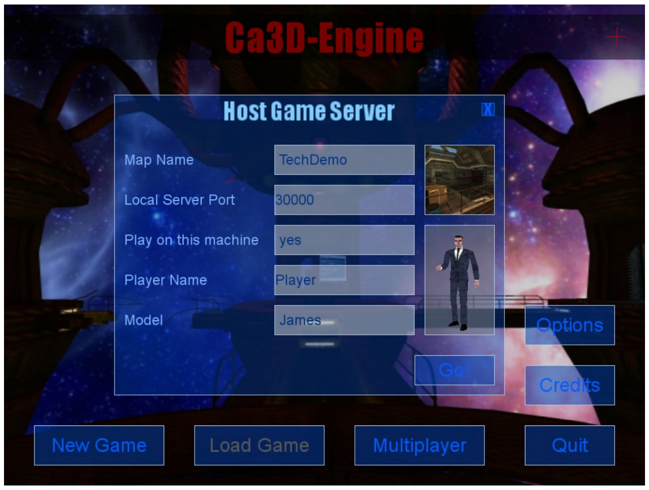
Player Name: Player
Model: James
Remote Host Name: 192.168.1.1
Remote Port: 30000

This dialog lets you join a multiplayer game using the given parameters:

- Player name Your player name. Other players see this name when you talk to them.
- Model Your appearance in the game world. Clicking on this control cycles through all available player models (the currently chosen model is shown in the preview windows to the right)
- Remote Host Name The name or the IP of the server that you wish to connect to.
- Remote Port The port number at which the server is listening on the remote host.

Clicking Connect will attempt to connect you to the chosen server and the game will start if connecting was successful.

3.1.3 Host Game Server



Map Name: TechDemo
Local Server Port: 30000
Play on this machine: yes
Player Name: Player
Model: James

This dialog lets you start and configure a multiplayer server.

- Map Name The map hosted on the server. Clicking on this box cycles through all available maps (a preview of the currently chosen map is shown on the right).
- Local Server Port The port number at which your server will listening for incoming connections.
- Play on this machine This box lets you choose whether you want to play on the server yourself or just start a stand alone server.

The following two options are only relevant if you have chosen Yes in Play on this machine.

- Player Name Your player name. Other players see this name when you talk to them.
- Model Your appearance in the game world. Clicking on this control cycles through all available player models (the currently chosen model is shown in the preview windows to the right).

Clicking Go! will start the game server and optionally connect you to it.

3.2 Keyboard Layout

In order to control the player and operate the game, you can use the keys as shown in the table below. The demonstrated layout reflects both *keyboard-only* player control layout and the popular *mouse+keyboard* player control layout. The keyboard layout will be made user configurable in future releases.

Action	Key	Alternative
Move forward	(Arrow up)	`W`
Move back	(Arrow down)	`S`
Turn left	← (Arrow left)	(Mouse left)
Turn right	→ (Arrow right)	(Mouse right)
Strafe left	`A`	`,`
Strafe right	`D`	`.`
Jump	`SPACE`	
Walk	`R_SHIFT`	`L_SHIFT`
Run (even faster)	`R_CTRL`	
Look up	`PAGE UP`	(Mouse up)
Look down	`PAGE DOWN`	(Mouse down)
Look banked CW	`HOME`	
Look banked CCW	`INSERT`	
Look straight ahead	`END`	
Talk / Chat	`T`	
Toggle Console	`F1`	
Screen-shot	`F5`	
Quit program	`ESC`	

Keyboard layout for player control and game operations.

3.3 The Command Console

In order to activate or deactivate the in-game console, press the `F1` key.

The large upper area of the window shows the console output of any component or subsystem of Cafu. The "Close" button in the lower right corner closes the console window again, bringing you back to the game. Other options to close the console window include pressing the `F1` or `ESC` key. The "Clear" button clears the contents of the console output window.

The text input field in the lower left is used for entering console commands. Note that the `TAB` key command-completion feature is available to facilitate text entry. The up and down arrow keys navigate through the command history for repeating previously entered commands.

Technically, the console interpreter is actually an instance of a Lua program in interactive mode. The Cafu engine binds its console variables and functions to that program so that they can be accessed like any other native Lua value. You may want to refer to the Lua documentation to learn more about the language Lua and its possibilities. The fact that you can write entire Lua programs at the Cafu in-game console provides great customizability and flexibility for both game developers and users.

Nonetheless, the Cafu console can also be used without any prior Lua knowledge, or any programming knowledge at all. Here are some getting-started examples that demonstrate the basic usage:

Entering the command

```
help()
```

prints out a short help text with instructions on how to obtain more help and the list of all available commands. Note that `help()` is actually a function call and therefore the brackets are mandatory.

```
list()
```

lists all available Cafu console functions, console variables, and global Lua values. As such, the `list()` function is useful to look-up the available commands. Note that the `TAB` key for command-completion serves a similar purpose, but the `list()` function provides more information about the values and the output is formatted.

```
help("quit")
list("cl")
```

The `help(“quit”)` function call provides help specific for the `quit` console variable, including a description of its purpose. This works analogously with any other console command, too. The `list(“cl”)` call works like the normal `list()`, but restricts the output to values that begin with the string `“cl”`.

```
screenSuffix="png"
```

sets the value of the console variable `screenSuffix` to `png`. Use this when you want to save your screenshots in `png` image file format instead of the default `jpg`. Enter `help(“screenSuffix”)` for more information about the `screenSuffix` console variable.

```
rcon("changeLevel('filename')")
```

`rcon` is the abbreviation for “remote console”. It means that the rest of the line is not processed by the local client, but rather sent to the remote server console. Therefore, this line instructs the server to initiate a level (world) change.

Pressing the `F1` key deactivates the console and brings you back to the game.

3.3.1 The config.lua File

When `Cafu.exe` starts up, it automatically processes the `config.lua` file. The `config.lua` file is located in the Cafu base directory. It is run in the context of the in-game console and can therefore contain any statement that you could also enter interactively as described above. Because the `config.lua` file is processed early during engine initialization, it is the ideal place to set default values and to keep any console statements that you find yourself entering over and over again. You can also define new functions or convenient abbreviations, and in fact, a complete, full-featured Lua program can be written into this file, either by the user, the game developer, or both. Just have a look into the `config.lua` file in the Cafu base directory to see some examples.

CHAPTER 4

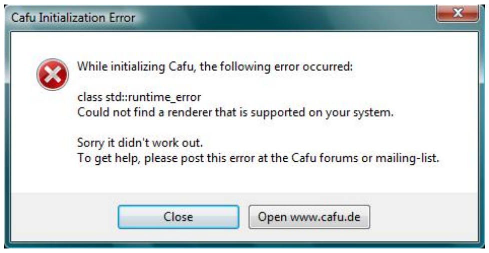
User FAQ

4.1 Why does my Personal Firewall report Cafu.exe?

When you start Cafu, it internally starts a client and a server, then connects to itself when a new map is loaded. This is done even for pure single-player games, and thus your Windows or Personal Firewall reports that `Cafu.exe` is establishing a network connection.

Therefore, please set your firewall to not block Cafu from network access. If you do not plan to host a Cafu game server or join a Cafu internet game, there is of course no harm in blocking access from and to the internet for `Cafu.exe`, but for single-player games, basic access to the local network is required.

4.2 Cafu reports an "Initialization Error", what can I do?



Cafu Initialization Error

While initializing Cafu, the following error occurred:

class std::runtime_error
Could not find a renderer that is supported on your system.

Sorry it didn't work out.
To get help, please post this error at the Cafu forums or mailing-list.

If you see an initialization error similar to the one in the image to the right, the most frequent cause is that the drivers for your sound or graphics hardware are outdated or incomplete (e.g. missing an OpenGL implementation). Please install or update the drivers for your hardware first, then try again.

If that doesn't help, run `Cafu.exe` from the command prompt with the `-log console.txt` option. For example:

```
> Cafu.exe -log console.txt
```

The contents of the newly generated file `console.txt` might give you a clue about why the initialization error occurred. If it doesn't, please post both the error message and the contents of the `console.txt` file at our forum or mailing-lists.

Note that you can run

```
> Cafu.exe --help
```

to see a list of all available command line options. They might be useful to work around the problem, and if you post at the forums, we might ask you to try some of them to further narrow the problem down.

4.3 Why do you use OpenGL? I think that DirectX is better.

Here are some features of OpenGL that made us choose it as the main rendering API:

- very well documented,
- portable and available on many platforms,
- simple to use,
- independent industry standard,
- powerful extension mechanism that makes it possible to support the latest features of new 3D graphics boards immediately.

You may also find this blog message interesting.

4.4 Why is the image quality low? How can I improve it?

In normal operation, the Cafu engine provides state-of-the-art image quality that is achieved by employing the latest methods in rendering technology. However, if the image quality is lower than you expected or not as good as shown in the reference screenshots at the Cafu website, these are the common causes for reduced image quality:

- If your graphics board is of an older make and model, it might not have a programmable GPU. Programmable GPUs however are required for rendering the sophisticated image effects of the Cafu engine. As a consequence, if Cafu has detected that no programmable GPU is present on your system, it automatically falls back into the best compatible mode that still works on your system. Normally you should also see an info message about this situation, but it is easy to miss the message. Unfortunately, the fall-back mode cannot provide the image quality that programmable GPUs can provide, as then the lighting and shadows have to be computed per-vertex rather than per-pixel, or their effects even have to be disabled entirely. In this case, you will see no dynamic lighting and no shadows at all. If this situation applies to your system, nothing can be done about it but getting a more recent graphics card.
- The texture detail in the initial options dialog is only set to "medium" quality by default. We have chosen that setting in order to not disappoint the FPS freaks among you, because higher quality normally also reduces the FPS. You can set the texture detail to the "high" setting in order to get higher texture resolution and thus higher image quality. Please note that the "high" quality setting can sometimes considerably lower the frame-rate, depending on the details of your graphics board. You may also want to read the other FAQ about low FPS in this regard.
- The default display resolution of Cafu is 1024*768. That setting has been proven a good compromise between image quality and frame-rate, but you might increase it within the console in order to increase the image quality. 2048*1536 looks really cool, but then the FPS is also lower.
- Some graphics boards with early generations of programmable GPUs are inherently limited in the image quality that they can provide (at a reasonable performance). For example, GPUs that are based on the NV2X series of NVidia GPUs do show specular highlights that look "blocky". Nothing can be done about that (except getting a more recent 3D board with at least NV3X or ATI Radeon 9600 or higher support).

4.5 Why is the frame-rate (FPS) low? How can I improve it?

During the past few years, many people have become used to use 3D engines that employ relatively old rendering technologies on relatively new graphics accelerator boards. This combination, observed in games like Counter-Strike, Quake3 and similar, achieved impressive frame-rates (often a hundred FPS and more) and fluid display updates.

The difference with Cafu is that Cafu employs dynamic, per-pixel lighting and shadows. The underlying rendering techniques are much more sophisticated than previous rendering methods, and relatively new. Despite a thoroughly optimized implementation, these techniques inherently cause lower frame-rates than older the techniques. In fact, if you compare Cafu to comparable 3D engines that are in the same category (only very few are, like Doom3), Cafu beats surprisingly well. ;)

There are many factors that have an effect on rendering performance. You can tweak some of them in order to manipulate the frame-rate, but the actual change in frame-rate much depends on your computer system and its components. Some of the suggestions below may even have no observable effect on your system at all. This is because every system has a different bottle-neck, and only finding and removing the bottle-neck yields the greatest effect.

Here is a list of options and suggestions that might help with the performance. You may simply try them out, and see what works best for your system. However, please note that most of these options *will buy you faster rendering at the cost of decreased image quality*.

Vertical synchronization. People frequently report that their frame-rate seems to be capped to 60 or 30 (or another multiple of 15) FPS. This is usually because *vertical synchronization* is enabled for your graphics board. Please see the Wikipedia article at Vertical_synchronization for technical details on the matter. Find the related setting in the control panel of your graphics driver to disable the vertical synchronization. Your graphics driver and board will then render the frames without the delay inherent to vertical synchronization, which in turn increases the FPS.

Video memory and texture size. If your graphics board has too little video memory to store all the textures simultaneously that Cafu needs for rendering a single frame, the FPS usually suffers significantly. In this case you can set a lower texture quality in the initial Options dialog. This effectively scales all textures down to not exceed a maximum size, and therefore improves the chances that they all fit into video RAM simultaneously. Another option is to add physical memory to your graphics board. 128 MB normally work well, 256 MB are even better and can store the textures even in maximum resolution.

Display resolution. The default display resolution of Cafu is 1024*768. That setting has been proven a good compromise between image quality and frame-rate, but you might decrease it within the console in order to increase the frame-rate. Reducing to 800*600 or even to 640*480 helps a lot.

Pixel fill-rate. A common reason for low FPS is that dynamic lighting and shadows consume a *very high pixel fill-rate*. Besides reducing the display resolution as mentioned above, an effective way to improve FPS in this case is to turn dynamic lighting and shadow effects entirely off. One way to achieve this is to set the `cl_maxLights` console variable to `0`. This can be done in the game console or by putting the statement .. code:: lua

  cl_maxLights=0;

into your `config.lua` file.

When Cafu is run from the command-line, it is also possible to specify a renderer DLL that makes use of less graphics features. For example, you might try the OpenGL 1.2 renderer in order to increase the FPS:

```
C:\Cafu-9.06> Cafu.exe -clRenderer ..
↪\Libs\build\win32\vc8\release\MaterialSystem\RendererOpenGL12.dll
```

CHAPTER 5

Developer FAQ

5.1 Is Cafu right for our game of genre "…"?

Variants of this question include:

- Can we use Cafu to make a MMORPG?
- Is Cafu fit for our 3D strategy game?
- Will we able to implement our car racing game with Cafu?

While we would love to provide you with a clear and authoritative "yes" or "no" answer to these questions, that is easier said than done:

When you ask one of these questions, you usually have a clear mental image already on how your game should look like – an image that we don't have, because often we know way too little about your project: your plans, your team, your skills (modelling, mapping, scripting, programming, …), your deadlines, etc. Not knowing what you *really* want and if you're ready to – for example – writing material shaders or making changes to the C++ source code is what makes this question difficult to answer.

However, there is an alternative that is much better than any statement we could make: Familiarize yourself with Cafu, try it yourself, and evaluate if it is right for your needs.

For example, download and run the latest demo. Explore the TechDemo map. Then try making a small map of your own, or edit one of the included maps in the editor. Browse the scripts and source code. If there are problems or questions, ask.

Not only will you become familiar with the Cafu details, you will also be able to form a very competent opinion about Cafu's suitability for your project.

5.2 Where are the Visual Studio project files?

In the development of Cafu, an important consideration is the build process that turns the source code into executable programs. This build process is a challenge in its own right, because it must work

- on all supported platforms (currently Windows and Linux, soon MacOS, too),
- with several compilers (and several versions of each),
- in several variants (32- and 64-bit; debug, profile and release),
- possibly with multiple combinations of the above, simultaneously on the same computer,
- with all external libraries,
- in changing environments (e.g. variable number of game libraries),
- flexibly, robustly and easily configurable for our fellow developers and users.

Achieving these goals with Visual Studio solution files on Windows (and possibly Makefiles on Linux) is a maintenance nightmare that borders on the impossible. For these reasons, we have chosen to use SCons as the build system for the Cafu Engine. SCons meets the above requirements, and it can still be used automatically and conveniently from most text editors.

Alas, we realize that having true Visual Studio project and solution files (and similar native files for other IDEs) would still be nice to have. We therefore started looking into these programs:

- Premake
- Bakefile
- CMake

These all look very promising, but each has problems of its own and until today nobody ever finished a complete, working solution. If you would like to help improving the situation, it would be very welcome.

5.3 What is the maximum number of network players?

There is no hard limit built into Cafu, but the general rule is that in the worst case, updates of everyone must be distributed over the network to everyone else. That means that the overall network load (on the server) roughly climbs with the number of players squared.

Fortunately, the Cafu network code is able to cut the network load significantly in many cases:

- Each network message is very small from the beginning, and additionally compressed.
- The Cafu server knows "who can (potentially) see who", and optimizes all cases where updates would be sent that cannot be observed by the receiving client.

With these features, Cafu can handle a large number of players in a single level all at the same time. We have demonstrated that connecting 8 clients over the internet is possible without problems (and with some bandwidth left) – and that was several years ago over a slow ISDN (64 kbit/s) Internet connection.

5.4 How can I clone the Cafu source code repository?

You can clone the Cafu source code repository like this:

```
> git clone --recursive https://bitbucket.org/cafu/cafu.git Cafu
```

Also see *Getting Started with the Cafu Source Code* for more details and https://bitbucket.org/cafu/cafu, where you can browse the repository online, create forks, submit pull requests, and find additional help texts.

5.5 How do I start a new game (MOD) project?

This is now explained in its own section in the documentation: See *Starting your own Game* for details.

5.6 How do I dynamically reload the map script in-game?

While you're developing a script for one of your maps, it can be a very helpful shortcut to reload the map script while the game is running, without interrupting it. This avoids the normally required cycle of leaving and re-starting the map.

As a preparatory step, add a function like this to your map script:

```
-- This function reloads and runs this script again.
-- It is useful for working with and testing the script "online",
-- i.e. while the game is running and without reloading the map!
function reloadScript()
    -- Adjust the path in the next line as required!
    dofile("Games/DeathMatch/Worlds/BPWxBeta.lua");
    Console.Print("Reloaded the map script.\n");
end
```

Then, at the *Cafu in-game console*, enter

```
runMapCmd('reloadScript()')
```

to dynamically reload the script.

In theory, you could combine these two steps into one, using something like `runMapCmd('dofile(“...”)')`, but that is even harder to type than the version above, and less flexible. (We realize that the entire `runMapCmd(...)` business is not optimal. Suggestions for making this more convenient are welcome.)

Note that if you use this technique, it can be helpful (but is not required) to understand how it works: The map script “lives” inside a Lua state that is initialized together with the map. (The in-game console has an entirely separate Lua state, which is why `runMapCmd(...)` is needed.) The above commands essentially run `dofile(...)` in the context of the maps Lua state, which in turn re-runs the map script.

5.7 Where can I learn more about 3D programming or game engines?

There are many available resources regarding 3D engines and related their subjects, both online and offline in the form of traditional books and papers. The following list is neither complete nor up-to-date, but we hope that it serves as a starting point for your own investigations into the matter:

- NeHe OpenGL Tutorials Excellent OpenGL tutorials for beginners to the advanced.
- http://www.opengl.org The official OpenGL website, and the first place to search for information about OpenGL.
- “Visibility Computations in Densely Occluded Polyhedral Environments” (1992) Seth Teller's dissertation about Potentially Visibility Sets (PVS): a common data structure in modern 3D engines that speeds up rendering.
- BSP-FAQs Introductory information about BSP trees.
- “Computer Graphics, Principles and Practice” by Foley, vanDam, Feiner & Hughes. Some say this is the bible of computer graphics – and it's true. It covers nearly everything there is to know. An excellent reference, but have a look into the table of contents before buying it.

- “Radiosity and Realistic Image Synthesis”, Michael F. Cohen and John R. Wallace. Morgan Kaufmann Publishers, Inc., San Francisco, California.
- “Radiosity. A programmers perspective.”, Ian Ashdown. John Wiley & Sons, Inc., New York, N.Y.
- “Zen of Graphics Programming, 2nd edition” by Michael Abrash. Most of this book is about low level DOS graphics programming (outdated now, but good nonetheless). The final chapters are a collection of Abrash’s articles that appeared in the “Dr. Dobbs SourceBooks” series. They contain information about BSP trees, span sorting, etc.
- C++ FAQs by Cline, Lomow, and Girou; Addison-Wesley. Available online and as a book.
- “Effective C++, Third Edition” by Scott Meyers.
- The C++ Programming Language by Bjarne Stroustrup.
- “Programming in Lua, Second Edition” by Roberto Ierusalimschy. The first edition of this book is also available online.

CHAPTER 6

Installation, Initial Configuration and De-Installation

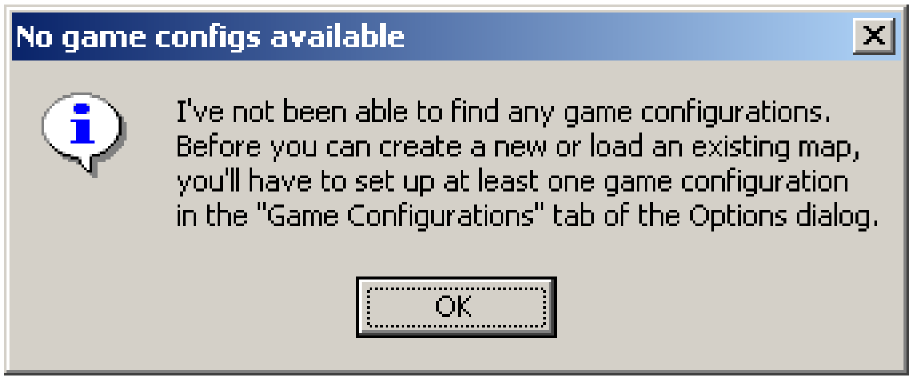
6.1 Installation

The CaWE editor is included with the regular Cafu releases, and as such requires no explicit installation: After unzipping the packed file, simply change into the `Cafu-x.y` directory. Under Windows, then double-click on `RunCaWE.bat`. Under Linux, click `CaWE-run` in the `Projects/Cafu/` sub-directory.

6.2 First-Time Configuration

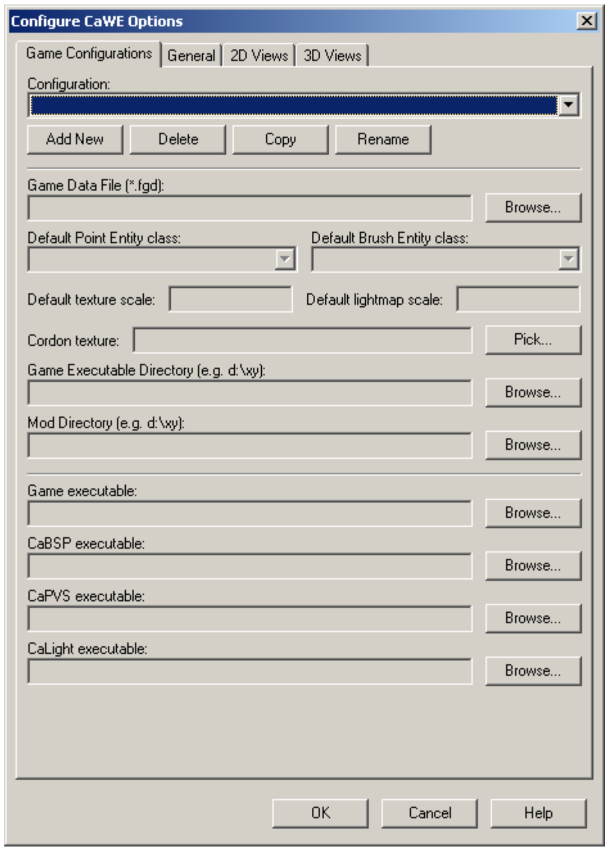
This section is relevant to CaWE releases before February 2008 (version 8.02) only. If you have version 8.02 or newer, the initial setup is fully automatic and you can skip this section safely. You may still check out the *Configure CaWE Options* dialog anytime later.

CaWE requires some basic configuration for normal use, and thus asks you for providing at least one game (or "MOD") configuration when it is first run:

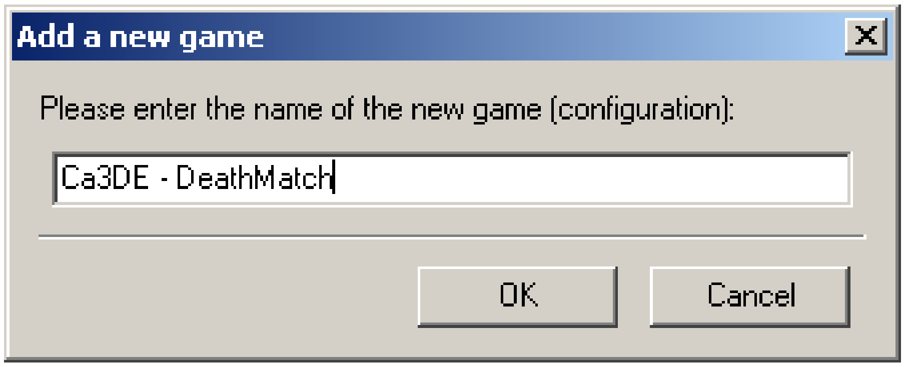
1. 
No game configs available

I've not been able to find any game configurations.
Before you can create a new or load an existing map,
you'll have to set up at least one game configuration
in the "Game Configurations" tab of the Options dialog.

After clicking on **OK**, CaWE will automatically show you the related configuration dialog.

2. 

Here, click on **Add New** for adding a new game configuration. You will then be asked for the desired name of the new config.

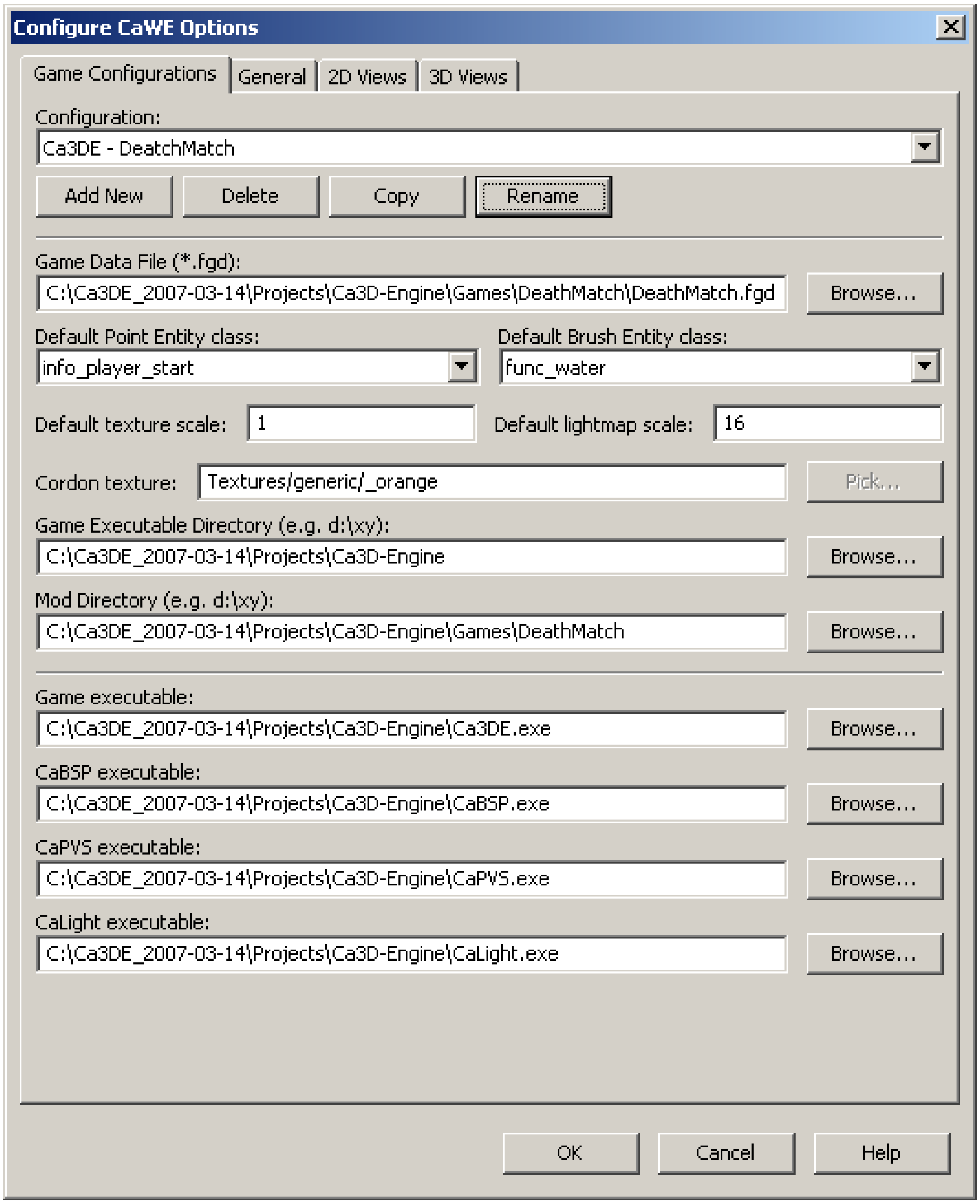
3. 

Please enter the name of the new game (configuration):

Ca3DE - DeathMatch

You can enter anything here, i.e. normally something that reminds you what MOD this config is related to, like "Cafu DeathMatch".

4. Next you have to fill-in where the Game Data File is located. This is a file with `.fgd` suffix. One such file is already provided with the Cafu package: the `DeathMatch.fgd` file in the `Projects/Cafu/Games/DeathMatch` subdirectory. Click on the related **Browse…** button and locate the file.

5. 

Configuration: Ca3DE - DeatchMatch

Game Data File (*.fgd): C:\Ca3DE_2007-03-14\Projects\Ca3D-Engine\Games\DeathMatch\DeathMatch.fgd

Default Point Entity class: info_player_start

Default Brush Entity class: func_water

Default texture scale: 1

Default lightmap scale: 16

Cordon texture: Textures/generic/_orange

Game Executable Directory (e.g. d:\xy): C:\Ca3DE_2007-03-14\Projects\Ca3D-Engine

Mod Directory (e.g. d:\xy): C:\Ca3DE_2007-03-14\Projects\Ca3D-Engine\Games\DeathMatch

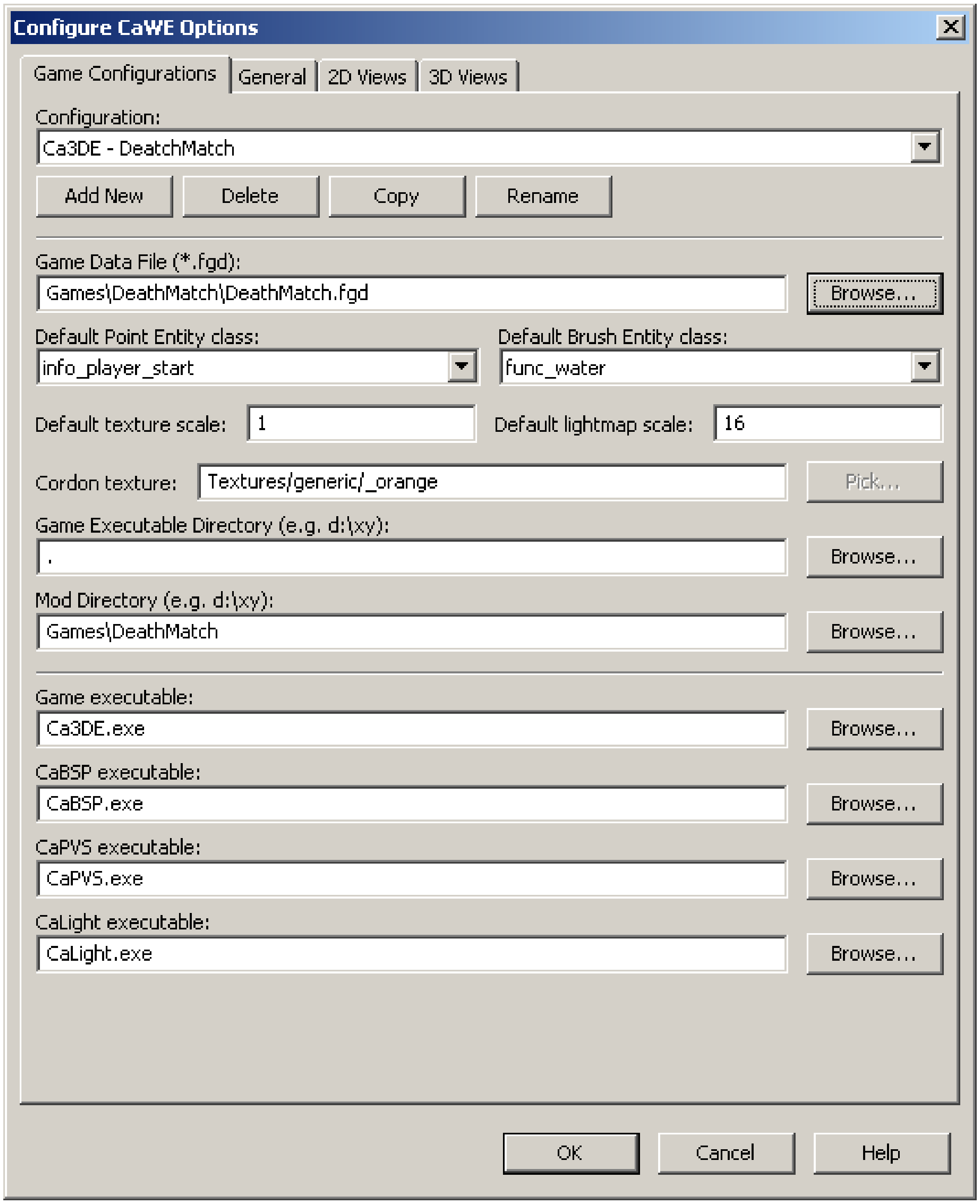
Game executable: C:\Ca3DE_2007-03-14\Projects\Ca3D-Engine\Ca3DE.exe

CaBSP executable: C:\Ca3DE_2007-03-14\Projects\Ca3D-Engine\CaBSP.exe

CaPVS executable: C:\Ca3DE_2007-03-14\Projects\Ca3D-Engine\CaPVS.exe

CaLight executable: C:\Ca3DE_2007-03-14\Projects\Ca3D-Engine\CaLight.exe

Finally, fill-in the rest of the dialog appropriately such that it eventually looks like this. The paths will of course be different for you, depending on where you installed the Cafu package and its version. See the next step however for an alternative setup.

6. 

Configure CaWE Options

Configuration: Ca3DE - DeatchMatch

Game Data File (*.fgd): Games\DeathMatch\DeathMatch.fgd

Default Point Entity class: info_player_start

Default Brush Entity class: func_water

Default texture scale: 1

Default lightmap scale: 16

Cordon texture: Textures/generic/_orange

Game Executable Directory (e.g. d:\xy): .

Mod Directory (e.g. d:\xy): Games\DeathMatch

Game executable: Ca3DE.exe

CaBSP executable: CaBSP.exe

CaPVS executable: CaPVS.exe

CaLight executable: CaLight.exe

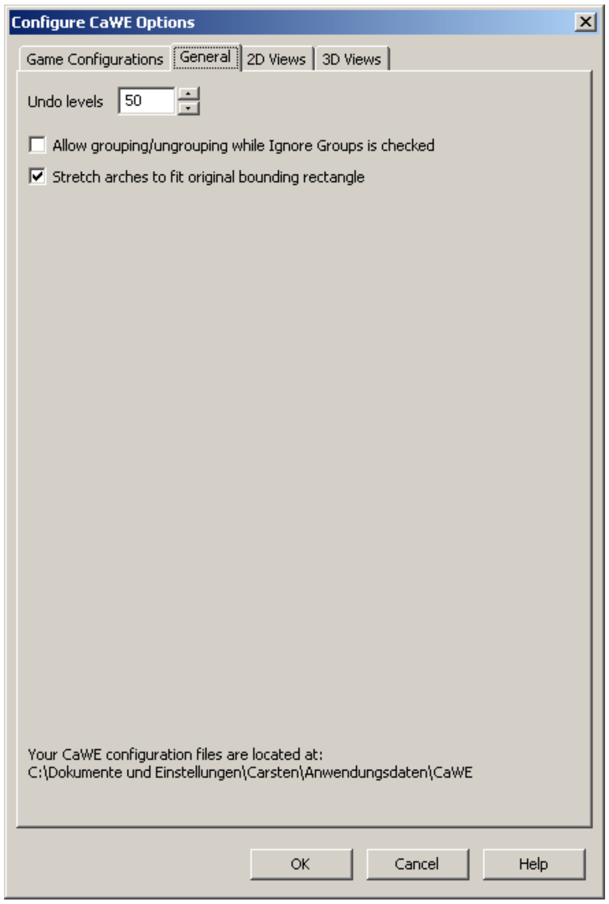
Alternatively to the previous step, you can also fill-in relative paths as shown in this image. This is the preferred method, because it works better with future versions of CaWE, makes upgrading to a newer version of Cafu and CaWE easier, solves some problems with map compilation some people were reporting, and will become the default with future releases of CaWE! Please note that under Linux, you have to use forward-slashes as the path separator (e.g. `Games/DeathMatch`) and executable files must be referred to like e.g. `./CaLight`.

All settings in this dialog can be revised *later at any time* via the **File** → **Configure CaWE...** menu. Adding new or additional game configurations at that time may however require to restart CaWE for the settings to take effect.

6.3 De-Installation

CaWE, as all other Cafu related software, has been designed to interfere with your computer and operating system as little as possible. In general, it is enough to delete the directory that the packed file was unzipped to during installation, and all traces of CaWE are gone.

There is one exception though: When run, CaWE stores your configuration settings in a newly created directory. This directory is located in the "home" directory of the current user. The exact place on your harddisk depends on your OS and even on its version, but CaWE states the full path to it both in the **Configuration** dialog (**General** tab), and in the **About** dialog box that is accessible via the **Help** → **About** menu.

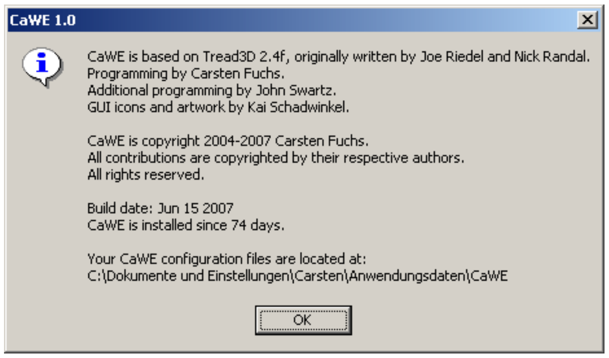


Undo levels 50

☐ Allow grouping/ungrouping while Ignore Groups is checked

☑ Stretch arches to fit original bounding rectangle

Your CaWE configuration files are located at:
C:\Dokumente und Einstellungen\Carsten\Anwendungsdaten\CaWE



CaWE is based on Tread3D 2.4f, originally written by Joe Riedel and Nick Randal.
Programming by Carsten Fuchs.
Additional programming by John Swartz.
GUI icons and artwork by Kai Schadwinkel.

CaWE is copyright 2004-2007 Carsten Fuchs.
All contributions are copyrighted by their respective authors.
All rights reserved.

Build date: Jun 15 2007
CaWE is installed since 74 days.

Your CaWE configuration files are located at:
C:\Dokumente und Einstellungen\Carsten\Anwendungsdaten\CaWE

CHAPTER 7

Getting Started

7.1 Introduction to Editing

7.1.1 Brushes

Do you know how a Lego model or a real house is built from bricks? Invididividual, small bricks are combined to form a complex architectural structure. CaWE works principally in the same way, and thus if you can visualize how individual bricks form a model or a big building, you will find it easy to grasp the concepts behind CaWE.

In CaWE, the bricks are called *brushes*, and they come in several shapes: blocks, wedges, cylinders, pyramids etc. Moreover, you can modify the default brushes in many ways. They can be scaled, sheared, mirrored, cut in pieces, carved and so on. You can create an arbitrary big number of these brushes in any shape you like, and use and combine them for building the matter that forms your world: walls, floors, roofs, furniture, rocks, … . Brushes eventually get *materials* assigned that represent the surface properties of that brush, as for example rock, glass, concrete, sand or water. The creation of brushes is detailed at The New Brush Tool.

7.1.2 Bezier Patches and Terrains

Modelling really complex surfaces can be difficult with brushes alone, especially if the surfaces should be curved, organic, or very big. Cafu and CaWE therefore provide two additional basic elements that complement brushes: *Bezier patches* are curved surfaces that you can imagine like bent or stamped metal plates. They can be used to model pipes, curves, smooth arches, and many other organic objects. The creation of bezier patches is detailed at The New Bezier Patch Tool. *Terrains* are similar, but as their name suggests, aim at surfaces that are much bigger and more irregular. They're also treated specially by the engine and have very high performance. The creation of terrains is detailed at The New Terrain Tool.

7.1.3 Entities

Entities define and represent all other interesting things in a world, and they come in two flavours:

Point-based entities exist at a specific coordinate in space and can be thought of as a reference point. Although these entities may refer to objects that actually have a volume, for example a monster, they are only treated as points in CaWE, and the Cafu engine provides the volume later in the game. An example is a monster entity: In CaWE, it just represents the point where the monster will be, in Cafu, it will acutally be there.

Brush-based entities are related to at least one brush or bezier patch and thus have a volume. Examples for brush based entities include doors and platforms, bodies of water, area triggers, etc.

The creation of entities is described in the *New Entity tool* documentation. A list with a description of all available entity types can be found in the *Entity Guide*.

7.1.4 Static Detail Models

While *static detail models* are really nothing more than (point-based) entities of type `static_detail_model`, they provide another powerful means to augment the detail in your world in a way that is not easy or even impossible to achieve with brushes, bezier patches and terrains alone. Such models can be imported from your favourite 3D modelling program, they can be arbitrarily complex, and they can for example be under the control of a level-of-detail system. Examples include models for furniture, statues, household or laboratory inventory, vehicles, etc.

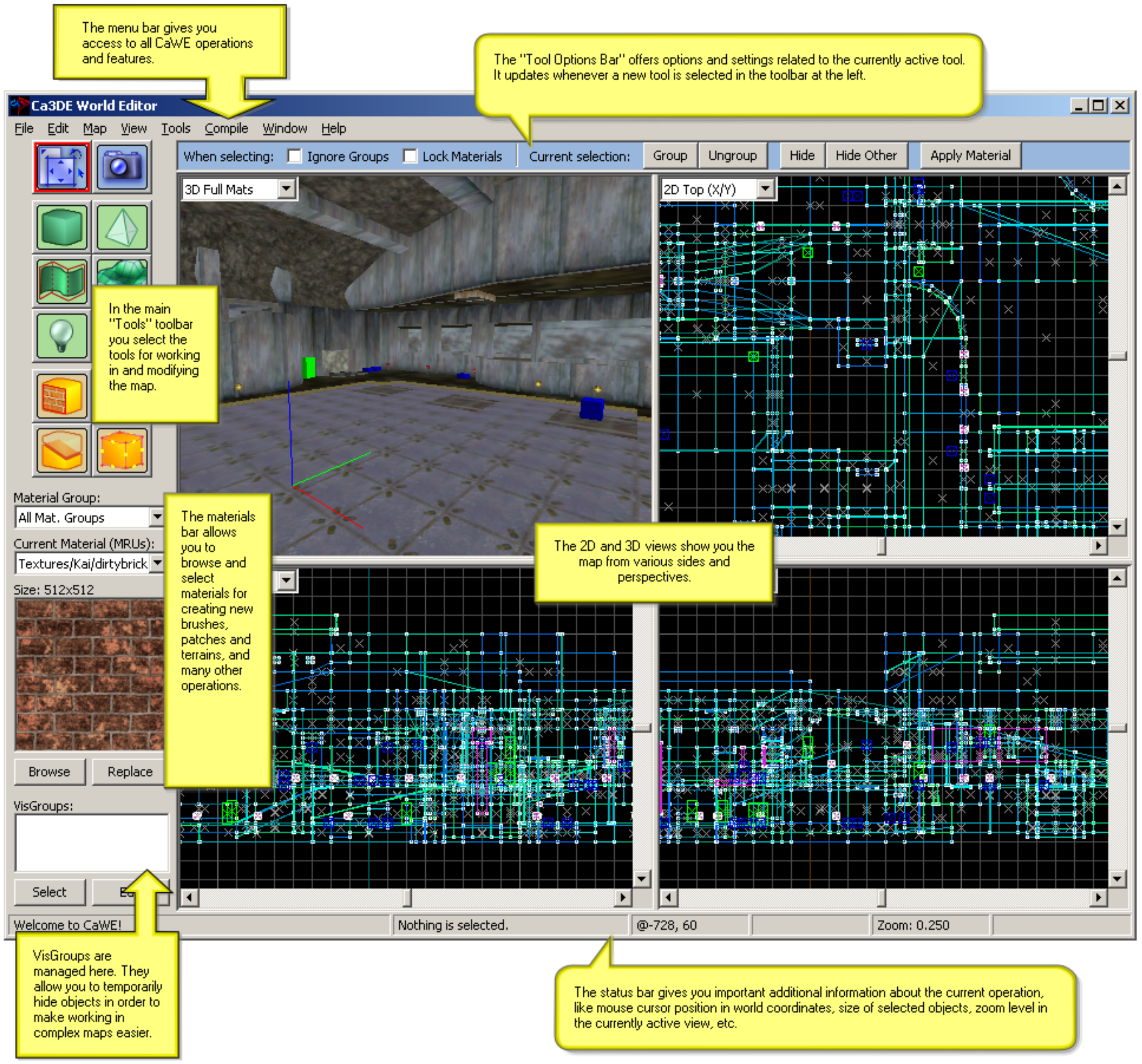
7.1.5 Putting it All Together

By combining these simple components, you can create stunningly virtual worlds whose number of variants is virtually unlimited. Starting with a rough outline of brushes and possibly terrains, you can fill in architectural detail with other brushes, patches and entities. Eventually you will populate your world with more entities that bring your creation to life.

The final step in making a world is compiling it so that it can be run with the Cafu engine. This is a quite complex process that precomputes visibility, lighting and many other details. However, CaWE has a great built-in facility to make the compilation process as easy as possible for you. The article *Compiling Maps for Cafu* has more information about how CaWE assists in compiling maps.

7.2 The Main Window User Interface

Whenever you work with a map in CaWE, the screen presents a number of toolsbars and controls. Here is an overview of the most important elements of the user interface.


The menu bar gives you access to all CaWE operations and features.
The "Tool Options Bar" offers options and settings related to the currently active tool. It updates whenever a new tool is selected in the toolbar at the left.
Ca3DE World Editor
File Edit Map View Tools Compile Window Help
When selecting: Ignore Groups Lock Materials Current selection: Group Ungroup Hide Hide Other Apply Material
3D Full Mats
2D Top (X/Y)
In the main "Tools" toolbar you select the tools for working in and modifying the map.
Material Group: All Mat. Groups
Current Material (MRUs): Textures/Kai/dirtybrick
Size: 512x512
The materials bar allows you to browse and select materials for creating new brushes, patches and terrains, and many other operations.
The 2D and 3D views show you the map from various sides and perspectives.
Browse Replace
VisGroups:
Select
Welcome to CaWE!
Nothing is selected.
@-728, 60
Zoom: 0.250
VisGroups are managed here. They allow you to temporarily hide objects in order to make working in complex maps easier.
The status bar gives you important additional information about the current operation, like mouse cursor position in world coordinates, size of selected objects, zoom level in the currently active view, etc.

Each toolbar and interface element outlined above has its own detailed section within this manual. Refer to the table of contents in order to learn more about each in greater depth.

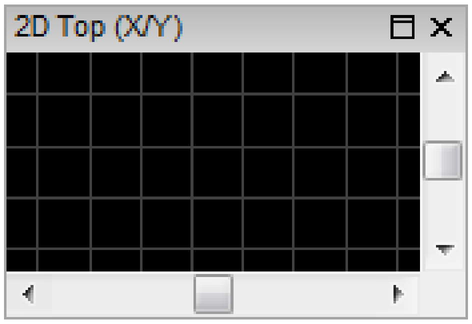
7.3 2D and 3D Views

The 2D and 3D views are the center of the activities when you work with a map. They have a number of features that make working with them very comfortable and straightforward.

In order to familiarize yourself with the material in this section quickly, we recommended that you try it out and reproduce it with one of the example maps that come with Cafu. Use the **File** → **Open...** menu for loading a map from the `DeathMatch/Maps` directory, e.g. `Kidney.cmap`.

By default, the *main window* is divided into three views that show your map from different sides and perspectives. In the center is the 3D view, which shows the map as a perspective rendering. The two 2D views show the map from the top and from the side like an architects plan.

7.3.1 The view windows



Each view is a window whose title bar indicates the type and render mode of the view. You can maximize, minimize or close each view window with the buttons in the upper right corner. In order to create a new 2D or 3D view, choose **View** → **New 2D view** or **View** → **New 3D view** from the main menu.

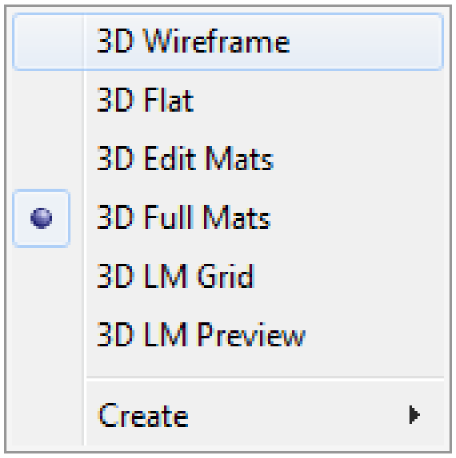
By clicking and dragging a windows title bar, the views can be docked, undocked and arranged as you like. Press `Ctrl` while dragging a window in order to keep it floating (prevent it from docking to one of the highlighted dock positions).

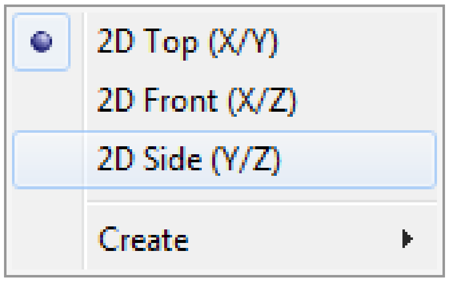
7.3.2 The active view

A view is *activated* when you move the mouse pointer into its window.

Keyboard input, menu commands and status bar information all refer to the most recently activated view. For example, if you want to navigate the views with the keyboard as presented below, just move the mouse pointer into the desired view window, and all keyboard input will be directed to it.

7.3.3 Changing the view mode

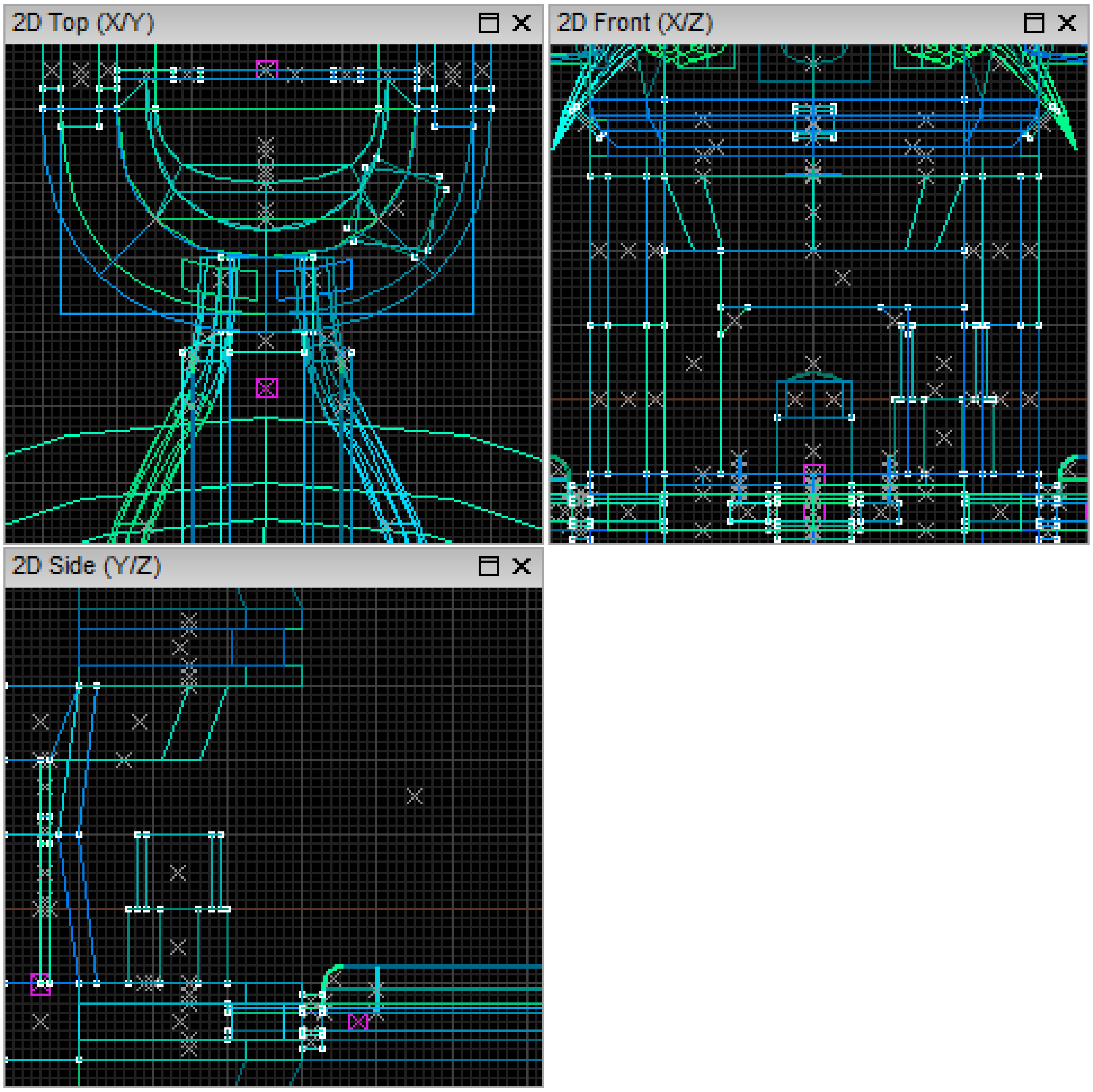




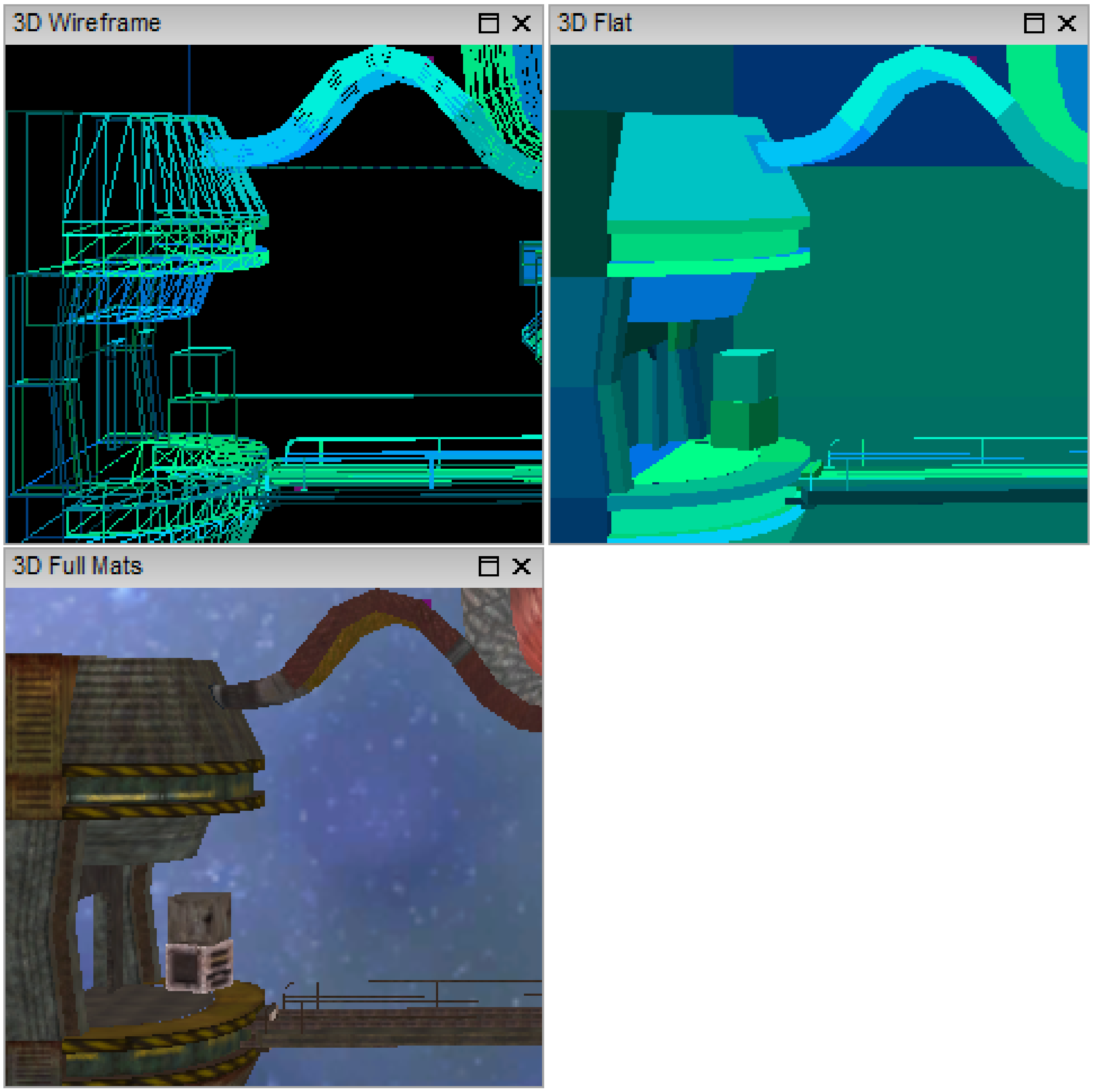
In order to change the mode of a 2D or 3D view, use:

- the `Tab` key,
- the `Shift+Tab` key,
- or the context menu (right mouse button click).

The 2D views show the map from the top, the front or the side:



The 3D views show the map in different render modes, for example “wire-frame”, “flat colored”, “edit materials” or “full materials”:



Note that render mode “3D Edit Mats” often looks much like “3D Full Mats”, but whereas the latter shows the materials in their “natural” appearance (as in the game engine, e.g. translucent, distorted, invisible, black, etc.), “3D Edit Mats” shows plainly textured surfaces instead. This can be very helpful for seeing the materials properly for editing purposes.

7.3.4 Navigating the 2D views

Use the following keyboard and mouse input in order to navigate the 2D views:

Keyboard Input	Action
Arrow keys ←↑↓→	Scroll the map into the direction of the arrow.
`Space`	Pan the view: while the key is being held down, scroll the map by moving the mouse.
+ or −	Zoom in or out (on mouse pointer).
`1`, `2`, `3`, . . . , `0`	Zoom to a preset level.
`Ctrl+E`	Center all 2D views onto the currently selected object(s).
`Z`	Toggle the pan feature: Pressing `Z` once is like holding down `RMB` or `Space` all the time. Pressing `Z` anew turns panning off again.
`Tab`	Cycle through top, front or side view modes as described *above* (`Shift+Tab` for opposite order).
Mouse Input	Action
`RMB` (click)	Open the context menu (or apply a tool-specific function).
`RMB` (drag)	Pan the view: while the button is being held down, scroll the map by moving the mouse.
Wheel	Zoom in or out (on mouse pointer).

In the status bar at the bottom of the screen you will see both the current position of the mouse pointer in world coordinates as well as the current zoom level of the active view. The currently active tool may provide additional information in the status bar as well.

7.3.5 Navigating the 3D views

Each 3D view has an associated camera that can be manipulated in order to show the map from arbitrary points, directions and perspectives. Use the following keyboard and mouse input in order to navigate (the cameras of) the 3D views:

Keyboard Input Action

Arrow keys ←↑↓→

Rotate the camera (look around).

`W`, `A`, `S`, `D`

Move the camera left/right and forwards/backwards (along its depth axis).

`Space`

An alternative to the `RMB` that works exactly alike, including all combinations of `Shift` and `Ctrl`.

`Z`

Toggle camera control: Pressing `Z` once is like holding down `RMB` or `Space` all the time. Pressing `Z` anew turns camera control off again.

`Tab`

Cycle through the render modes as described *above* (`Shift+Tab` for opposite order).

`1`, `2`

Advanced: Move the far clip plane closer to or farther from the camera.

Mouse Input

Action

`RMB` (click)

Open the context menu (or apply a tool-specific function).

`RMB` (drag)

Rotate: Move the mouse to rotate the camera (look around).

`RMB+Shift`

Pan: Move the mouse to pan the camera left/right and up/down.

`RMB+Ctrl`

Fly: Move the mouse to pan the camera left/right and move it forwards/backwards.

`RMB+Ctrl+Shift`

Walk: Move the mouse to rotate and move the camera in the XY-plane.

`MMB` (drag)

Orbit around the point under the mouse pointer (horizontally and vertically).

`MMB+Shift`

Orbit, and move the camera closer to or farther from the object under the mouse pointer.

`MMB+Ctrl`

Same as `MMB+Shift`.

Wheel

“Zoom” (move the camera closer to or farther from the object under the mouse pointer).
Combine with `Ctrl` *or* `Shift` for slower (but more precise) movement.
Combine with `Ctrl` *and* `Shift` for even slower (but even more precise) movement.

Notes and references:

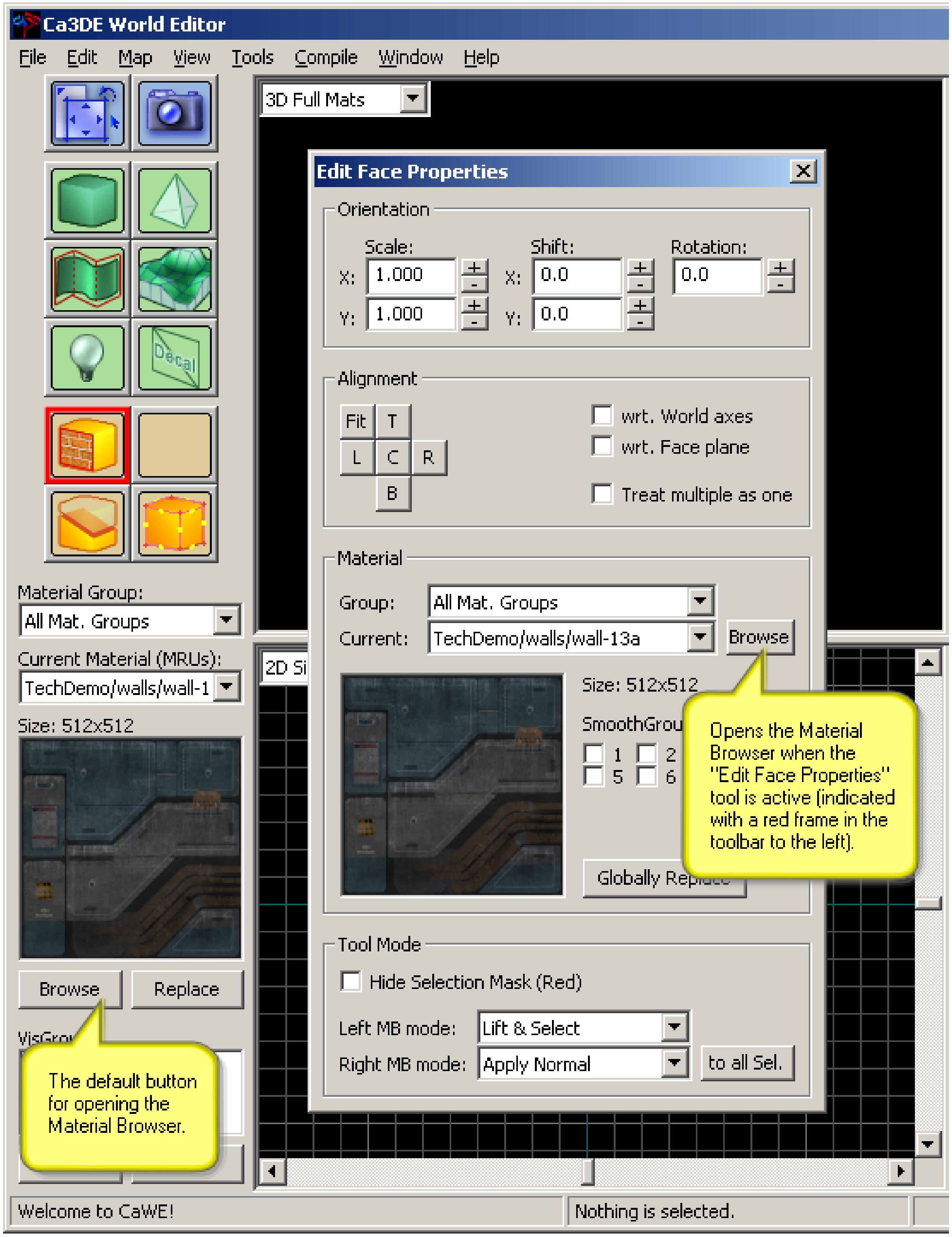
- Except for opening the context menu, which requires a RMB *click*, the mouse buttons RMB and MMB must be pressed *and held* while the mouse is moved to control the camera.
- You can use the *Camera Tool* for more exotic (and less frequently needed) camera control features, such as creating additional cameras or setting the camera origin and orientation in a 2D view.
- The *Configure CaWE* dialog has options that affect the 3D views and their cameras, e.g. for setting the movement speed or reversing the mouse Y-axis.

7.3.6 Video

This video demonstrates the most important concepts for navigating the 2D and 3D views.

You can also download the high-quality edition: CaWE_Navigating_the_Views.mp4

7.4 The Material Browser

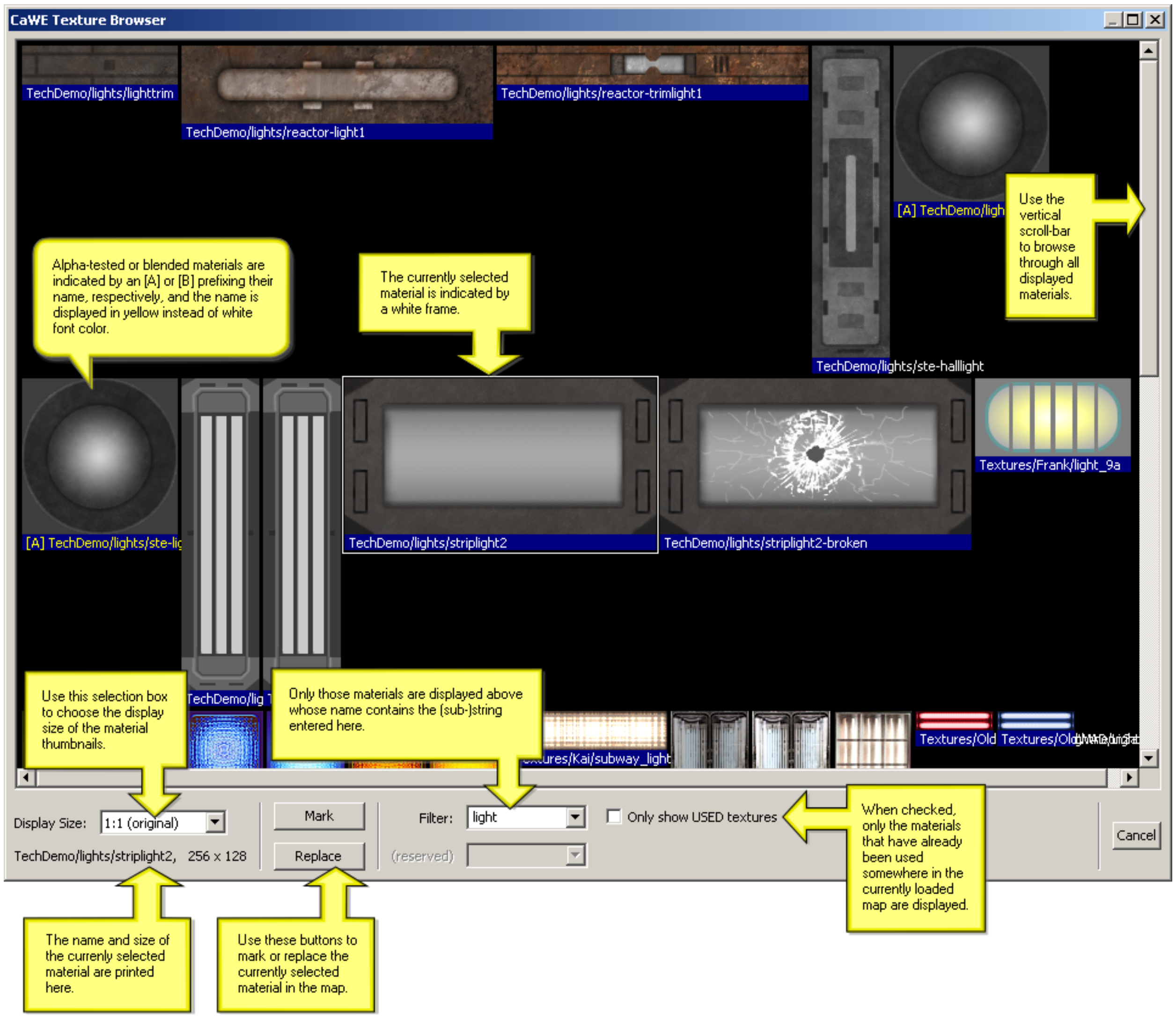


One of the most important dialogs when working with CaWE is the *Material Browser*. Materials are used by many tools, e.g. when new brushes, bezier patches or terrains are created or when the surface properties of such an element are edited. The Material Browser displays all available materials as thumbnails and allows you to select one for subsequent operations with the tools.

The image to the right shows the essential buttons for opening the Material Browser. In most cases and with most tools, you'll use the default button in the left Material Toolbar for opening the Material Browser, but the "Edit Face Properties" tool has an own dialog that has its own button.

Please note that when you press the `Browse` button *for the first time during a CaWE session*, CaWE has to cache-in the available materials all at once. Therefore it may take a short while until all materials are loaded. This will only happen once during a CaWE session, and pressing the `Browse` button another time will open the Material Browser instantly.

The most essential details of the Material Browser dialog are presented in the image below.


CaWE Texture Browser
TechDemo/lights/lighttrim
TechDemo/lights/reactor-light1
TechDemo/lights/reactor-trimlight1
[A] TechDemo/ligh
TechDemo/lights/ste-halllight
[A] TechDemo/lights/ste-lig
TechDemo/lights/striplight2
TechDemo/lights/striplight2-broken
Textures/Frank/light_9a
TechDemo/lig
Textures/Kai/subway_light
Textures/Old
Textures/Old
Display Size:
1:1 (original)
Mark
Replace
Filter:
light
(reserved)
Only show USED textures
Cancel
TechDemo/lights/striplight2, 256 x 128

Use the vertical scroll-bar to browse through all displayed materials.

Alpha-tested or blended materials are indicated by an [A] or [B] prefixing their name, respectively, and the name is displayed in yellow instead of white font color.

The currently selected material is indicated by a white frame.

Use this selection box to choose the display size of the material thumbnails.

Only those materials are displayed above whose name contains the (sub-)string entered here.

When checked, only the materials that have already been used somewhere in the currently loaded map are displayed.

The name and size of the currently selected material are printed here.

Use these buttons to mark or replace the currently selected material in the map.

7.5 Your First Map

The flash tutorial below introduces you to the basic features and concepts of CaWE and demonstrates how you can create your first map. It starts with opening a new map file in the Cafu World Editor CaWE, then walks you step-by-step through the construction process. At the end, the map is compiled and run in the Cafu engine.

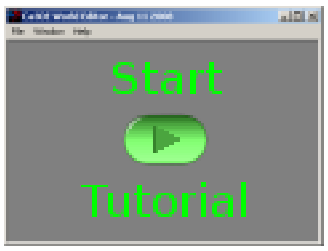
The tutorial assumes that you have already installed CaWE and completed the initial one-time configuration as described in section *Installation, Initial Configuration and De-Installation*.

Although the tutorial does not require any previous knowledge, it is helpful to read the preceding "Getting started" chapters either before starting or while working through the tutorial.

7.5.1 Tutorial Contents Overview

1. Verification of essential settings in the *CaWE Options* dialog.
2. Opening of a new map file in which a single room will be built.
3. Creation of the first brush for the floor.
   - Use of the *Material Browser* to select an appropriate material.
   - Use of the *New Brush* tool to create a brush outline.
   - Dragging of the outline and updating of the 3D camera position.
4. Creation of the second brush for the left wall analogous to the first.
   - Zooming in the 2D views and more advanced manipulation of the brush outline.
5. Creation of the third brush for the right wall by employing the clone-dragging technique of the *Selection* tool.
6. Creation of the fourth brush for the back wall by using the "Copy" and "Paste" items of the *Edit* menu.
   - Includes the transformation of a selected object to (re-)shape it as desired.
7. Creation of the fifth brush for the front wall as a copy of the back wall by use of the related keyboard shortcuts.
8. The ceiling brush (sixth brush) is created by clone-dragging it from the floor brush.
9. Application of a sky material to the ceiling brush as a post-creation step.
10. Demonstration of the difference between the *render modes* "3D Full Mats" and "3D Edit Mats".
11. Placement of an `info_player_start` entity near the center of the room, using the *New Entity* tool.
12. The map is *compiled and run* in the Cafu engine.

7.5.2 The Flash Tutorial

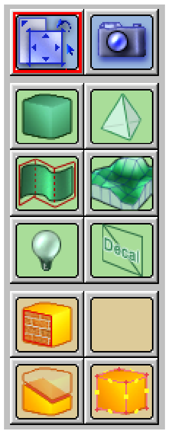


Click here to start the flash tutorial.

CHAPTER 8

Map Editing Tools

8.1 The Selection Tool



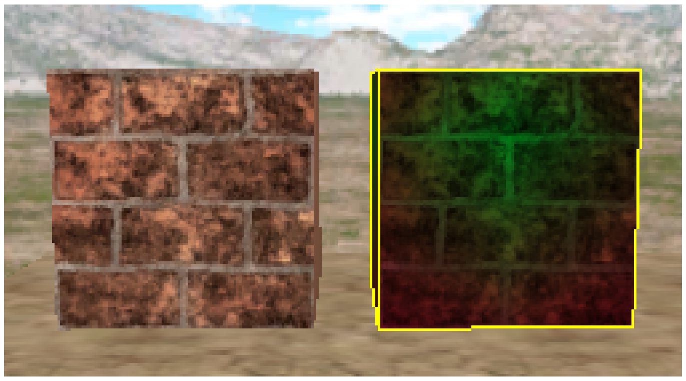
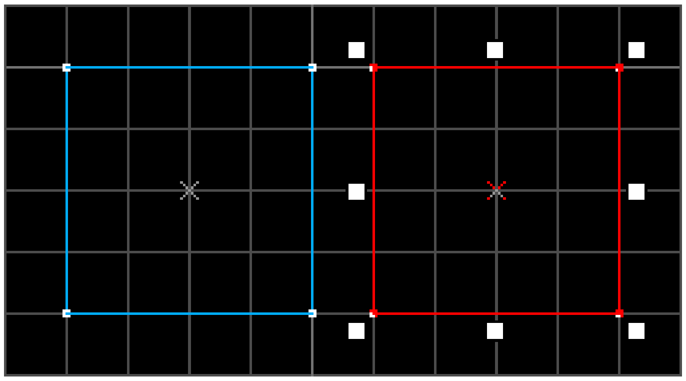
The Selection tool allows you to select objects like brushes, entire entities or bezier patches in your map and move, scale, rotate or shear them. Apart from these functions directly activated by the Selection tool, CaWE provides a lot more functions that are related to selected objects. Since these functions are only usable with one or more objects, the Selection tool is a very important instrument to create maps.

You can activate the Selection tool by clicking on the related icon marked in the left screen or by pressing the `SHIFT+S` keyboard shortcut.

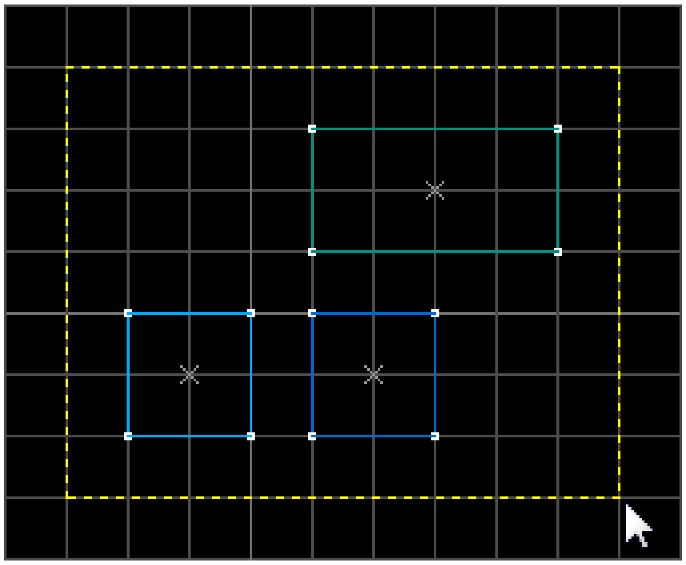
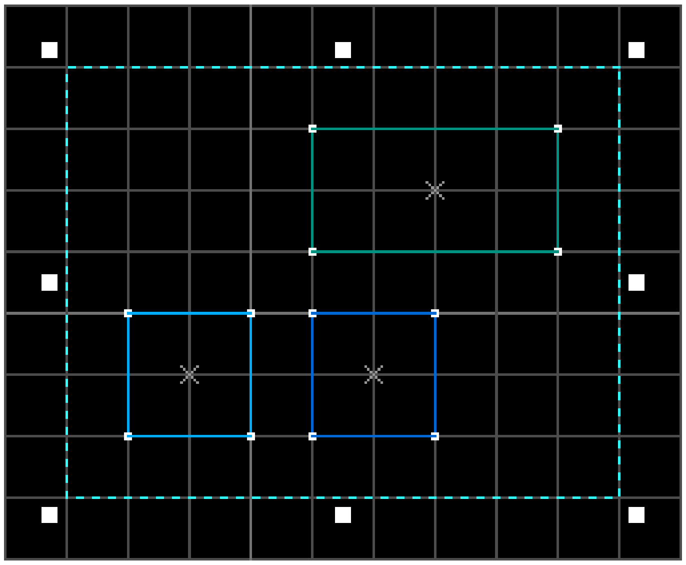
8.1.1 Selecting objects

To select an object, click the object once. In the 3D view, click on any surface that is part of the object to select it. In the 2D view click either on an edge that is part of the object or on its center marker.

In the 3D view, the surface of the object will be changed in color and its edges are colored yellow to mark it as selected. In the 2D view, the edges of the selected object are colored red.

To select one or more objects you can also drag a selection box in a 2D view window. Releasing the mouse button opens the selection box, but does not select the objects yet. You can move and scale this selection box until it covers all objects you want selected. To eventually select the objects, press the ENTER key. This way all objects that are covered whole or partially by this selection box are selected. Note that you can also scale the selection box in another 2D view to create a spatial selection box and narrow down the selected objects.

You can change the behavior of a selection box in the *Configure CaWE Options* dialog at the **2D Views** tab. **Automatic infinite selection in 2D windows (no ENTER)** enables you to drag a selection box and instantly select all objects by releasing the mouse button instead of pressing the ENTER key. This way however you loose the ability to open a spatial selection box. If **Selection box selects by center handle only** is activated, only objects whose center handle is covered by a selection box count as selected objects.

Another way to select more than one object is to select objects while pressing the CTRL key. This way you can choose as many objects as you want and perform an action on all these objects simultaneously.

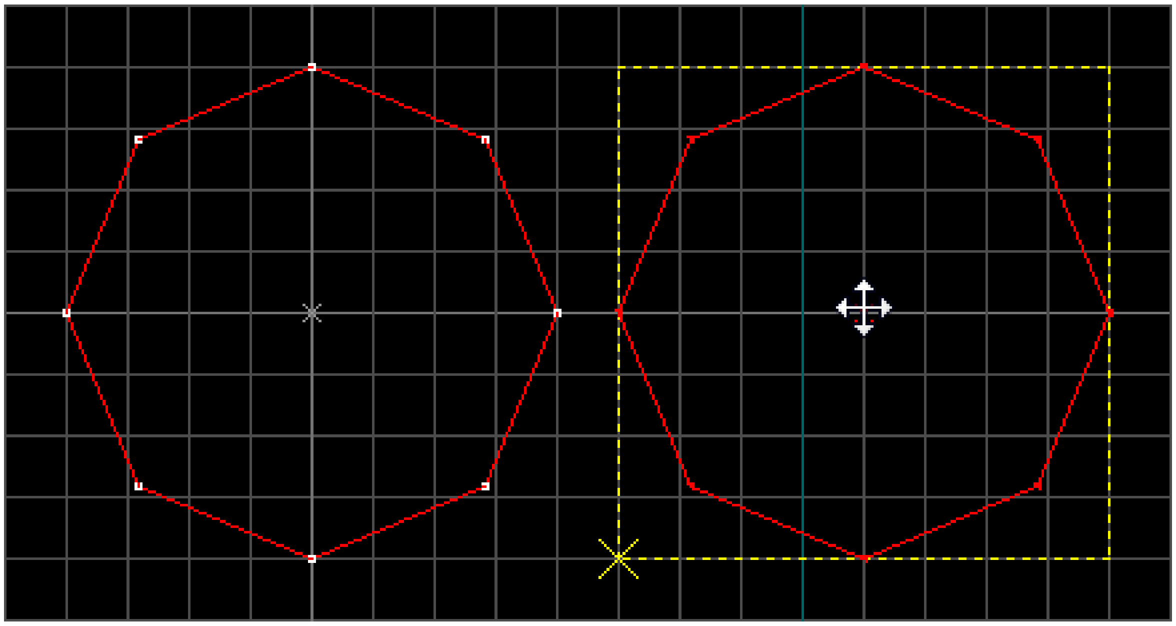
You can select all objects in a map by clicking the **Select All** option from the *edit menu*. To deselect all objects, use the **Select None** option from the *edit menu*.

You can also deselect all selected objects by pressing the ESC key, which clears the selection.

8.1.2 Working with selected objects

Now that one ore more objects have been selected you can perform actions on them.

Moving and Cloning objects



By clicking inside the selection box of an object, you can drag it in a 2D view and move its position to a new location. If grid snapping is activated, you can ignore it while moving the object by pressing the `ALT`-key. This also works with the transformations described below.

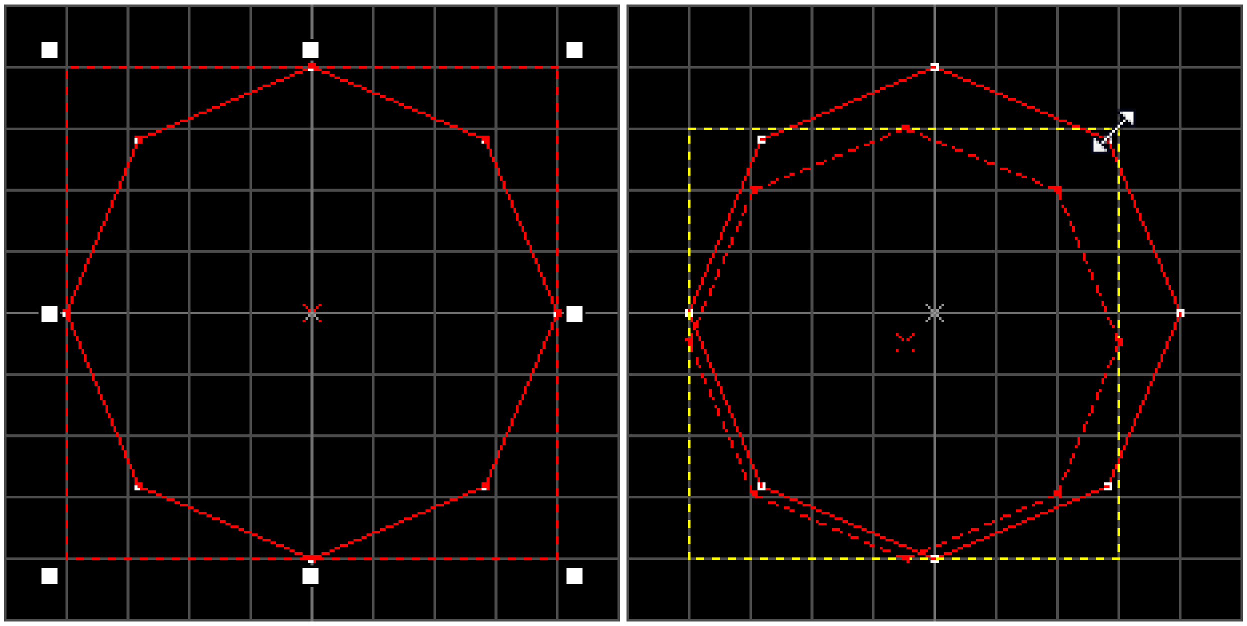
You can also move objects with the "arrow" keys, if you select the **Arrow keys nudge selected object/vertex** option in the *Configure CaWE Options* dialog under **2D Views**.

An easy way to duplicate objects is using the **Clone** function while moving an object. Simply move your object as described above, but before releasing the left mouse button to move the object, press and hold the `SHIFT` key. In this way a clone of the object is created at its new location. Note that cloning an object while moving it with the "arrow" keys is not supported.

Tranformation modes

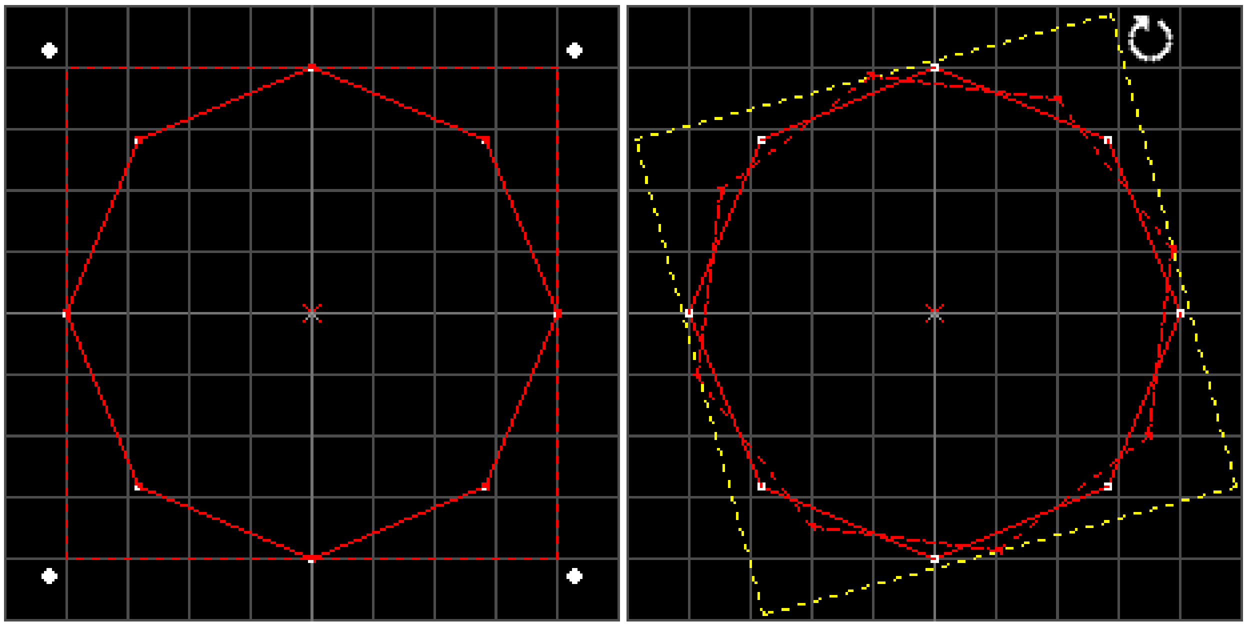
To resize, rotate or shear an object you have to select the appropriate transformation mode. You can switch between the 3 modes by clicking inside the selection box of the object you want to transform. Different transformation modes are marked by handles at the edges and vertices of the selection box.

Resizing an object



A dashed rectangular box is drawn around a selected object by the editor (this box is only visible if the object itself isn't rectangular). The resize mode is marked by white squares at the edges and vertices of this box. By moving the mouse over one of these squares and dragging this square by holding down the left mouse button you can resize the object in the direction of the square you clicked on.

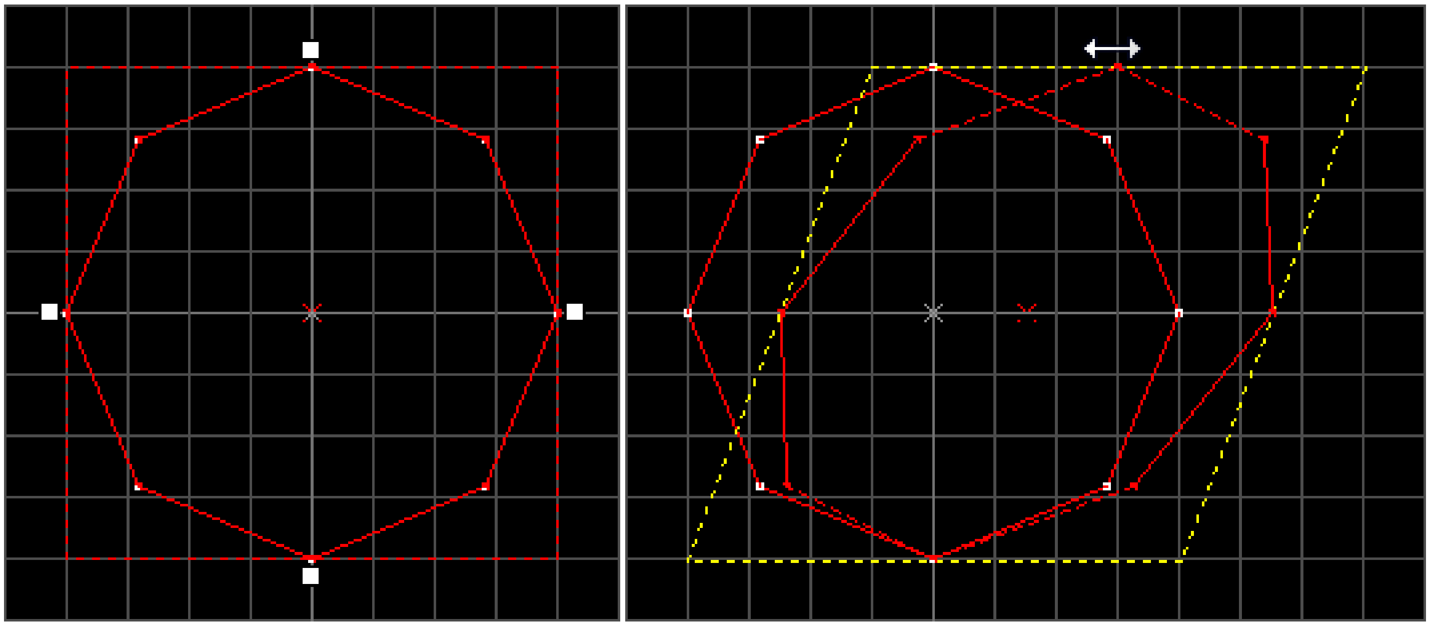
Rotating an object



The rotate mode is marked by white dots at the vertices of the box. Moving the mouse over one of these dots and dragging this dot by holding down the left mouse button rotates the object smoothly. It is also possible to rotate the object in multiples of 15 degrees by pressing and holding down the `SHIFT`-key while rotating an object.

You can change the default behavior to rotating in 15 degree steps if you select the **Default to 15 degree rotations** option in the *Configure CaWE Options* dialog under **2D Views**. If this option is select, pressing `SHIFT` will allow you to rotate the object smoothly.

Shearing an object



White squares at the edges of the selection box mark the shearing mode. By left clicking on a square and moving the mouse while holding the left mouse button down this edge can be moved in two directions along its own axis and in this way the object is deformed.

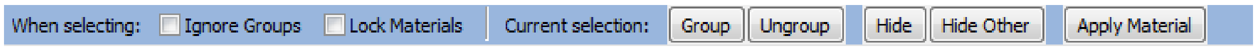
Deleting an object

To delete a selected object just press the `DEL`-key on your keyboard or choose **Delete** from the *Edit menu*.

More

There are a lot more things you can do with selected objects. These additional functions are explained in the documentations of the *Edit menu*, *Map menu*, *View menu* and *Tools menu*.

8.1.3 The Tool Options Bar



Once the Selection tool is activated, its option bar is visible above the view windows. This bar contains selection parameters and buttons for further functions related to selections.

Ignore Groups This option specifies if grouped objects are selected as a group (selecting one object selects all objects) or if groups are ignored and objects are selected normally even if they are part of an object group.

Lock Materials This option determines if the Material of an object is locked to the object itself and therefore doesn't change, even tough the object is moved. If this option is not selected the Material is locked onto the point of origin of the world coordinate system and therefore the part of the texture visible on the object may change when the object is moved.

Group This button puts two ore more previously selected objects into a group. Grouped objects can be selected all at once by clicking on one object of the group.

Ungroup This button reverts grouped objects back to single objects. This function only works if a group of objects has previously been selected.

Hide Hides all selected objects from the view windows. The objects are not deleted, but simply not displayed, which is particularly useful when working with big maps that contain a lot of objects.

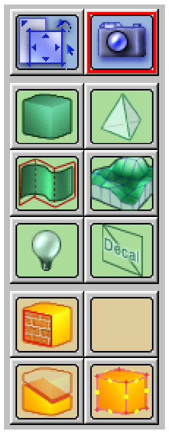
Hide Other Hides all objects except the selected objects.

Apply Material Applies the material that is currently selected in the *materials bar* to the selected object(s).

8.1.4 Selection tool keyboard shortcuts

- `SHIFT+S`:
  - Activates the Selection tool.
- `ENTER`:
  - If a box selection is opened → selects all objects that are part of the box selection .
- `ESC`:
  - If a box selection is opened → closes the box selection.
  - If an object is currently resized, rotated, sheared or moved → move object back into original position and form.
  - If objects are selected → deselect objects.
- `DEL`:
  - Deletes the selected object(s) from map.
- `CTRL`:
  - If in selection-mode → add new selected objects to selection list instead of removing old selection and only selecting clicked object.
- `ALT`:
  - While moving or cloning an object → ignore grid snapping and move object smoothly
  - While tranforming an object → ignore grid snapping and transform object smoothly
- `SHIFT`:
  - If Rotating an object → rotate in multiples of 15 degrees.
  - If moving an object → object is cloned first then moved, so original object remains in place.
- `Page up`:
- `Page down`:
  - If last select click covered more than one object → cycle trough objects.

8.2 The Camera Tool



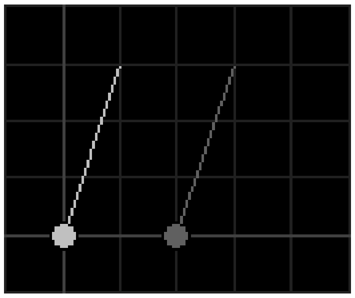
In section *2D and 3D Views* we have explained how 2D and 3D views work and how you use them to explore and navigate the map. We have also mentioned (albeit briefly) that each 3D view has a "camera" assigned: A camera specifies a viewer *origin* (sometimes also called "eye position") and an *orientation* ("direction"). When you use the mouse or keyboard to look, move, pan, zoom or orbit in a 3D view, you actually manipulate the related camera position and orientation, and the 3D view in turn uses the updated camera details to update the scene rendering.

We believe that the mouse and keyboard camera controls of section *2D and 3D Views* are so powerful and easy to use that you will use them virtually all the time.

The purpose of the Camera tool is to add the less frequently needed extras: You can create new cameras (and assign them in turns to 3D views), and edit their origins and orientations in the 2D views.

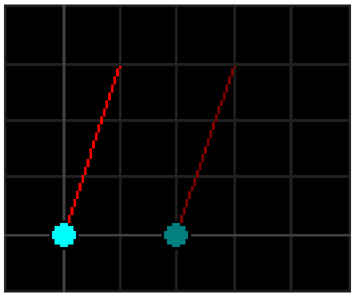
To activate the Camera tool, click on the related icon in the tool bar or press `Shift+C`.

8.2.1 Camera display in 2D views



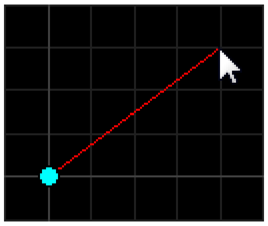
When a tool other than the Camera tool is active, cameras are shown with a gray dot at their origin and a gray line that indicates their viewing direction (orientation).

The most recently used (or changed) camera is highlighted in a brighter shade of gray: For example, *activating a 3D view* highlights the related camera.


When the Camera tool is active, the cameras are displayed in colors so that they are easier to spot in the map.

As before, the most recently used (or changed) camera is highlighted in brighter colors.

8.2.2 Creating and deleting cameras


To create a new camera, press and hold the `Shift` key, then use the left mouse button to drag a line in one of the 2D views. The starting point of the line becomes the camera origin, the line itself defines the orientation.

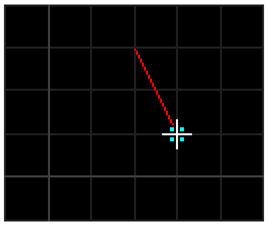
Having multiple cameras distributed at key locations in the map can facilitate map navigation a lot, as a 3D view can cycle through all available cameras as described *below*.

At this time, cameras that are not assigned to a 3D view are not saved with the map; they are lost when the map is closed. You can manually delete an active camera via the **Edit** → **Delete** menu item. The last camera cannot be deleted, as there is at least one 3D view that it is assigned to.

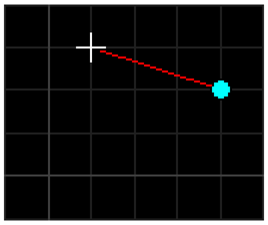
8.2.3 Manipulating the origin and orientation

The most powerful and most convenient methods to manipulate a camera are the controls described in section *Navigating the 3D views* (in addition, the controls described there do not require the Camera tool to be active: they work always).

Sometimes however, it is worthwhile to set the origin and/or the orientation in a 2D view:


Click and drag the dot in order to move the camera origin.

- Holding the `Alt` key while dragging toggles grid snapping.
- Holding the `Ctrl` key moves the line as well (and thus the camera as a whole).


Click and drag the end of the line in order to change the camera orientation. (The *length* of the line does not matter. You can for example drag the end of the line onto the object of interest, to center it precisely in the related 3D view.)

- Holding the `Alt` key while dragging toggles grid snapping.
- Holding the `Ctrl` key moves the dot as well (and thus the camera as a whole).

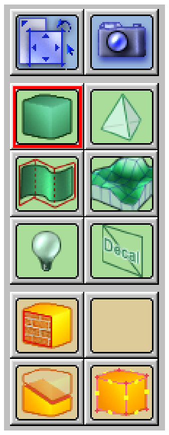
8.2.4 Switching cameras

In order to assign a 3D view (the *active* 3D view) another camera, use the `Page Up` and `Page Down` keys to cycle through all cameras in the map.

8.2.5 Camera tool keyboard shortcuts

Key	Action
`Page Up`	Assign the next camera to the currently active 3D view.
`Page Down`	Assign the previous camera to the currently active 3D view.
`ESC`	Quit the camera tool and switch to the *Selection* tool.
`Shift`	LMB-clicking and dragging in a 2D view creates new camera.
`Ctrl`	Move the camera as a whole when dragging one of its handles.
`Alt`	Temporarily toggle grid snapping while dragging a camera handle.

8.3 The New Brush Tool



The New Brush tool is, as its name says, used to create new brushes.

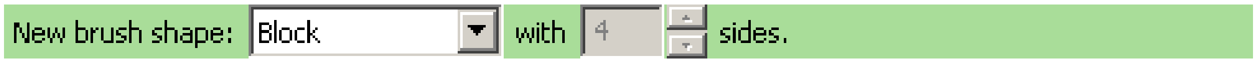
Brushes are primitive objects with a simple structure and come in different shapes, like a block, a cylinder, etc..

Most parts of a map in CaWE are build from brushes, whereas more complex structures are built as a combination of brushes.

The Brush Tool can be activate by clicking on its related icon or by pressing `SHIFT+B`.

8.3.1 The Tool Options Bar

As the New Brush tool is activated, its options bar is visible above the view windows.

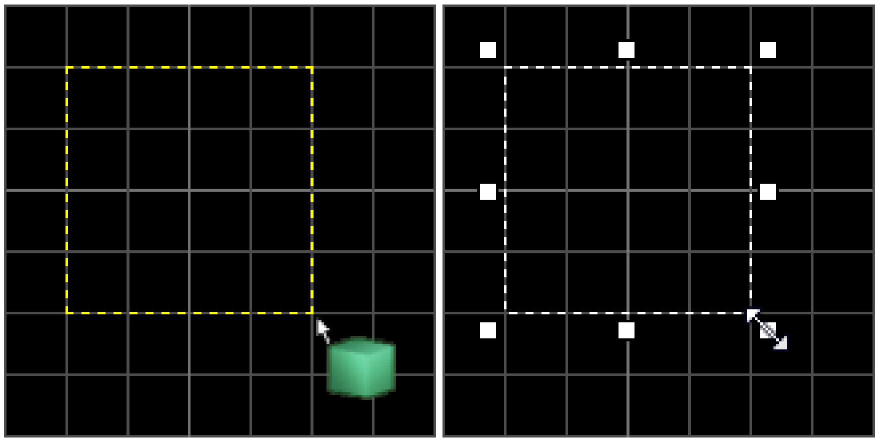


New brush shape: Block with 4 sides.

Here you can select the shape of the brush you want to create and the number of sides the brush should have. For some shapes the number of sides is fixed (as for example the block shape) and number of sides setting is grayed out.

8.3.2 Creating a new brush

To create a new brush you have to drag a box in a 2D view, that represents the height, width and depth of the brush. Since the brush is not created after dragging the box you are still able to move and resize it.

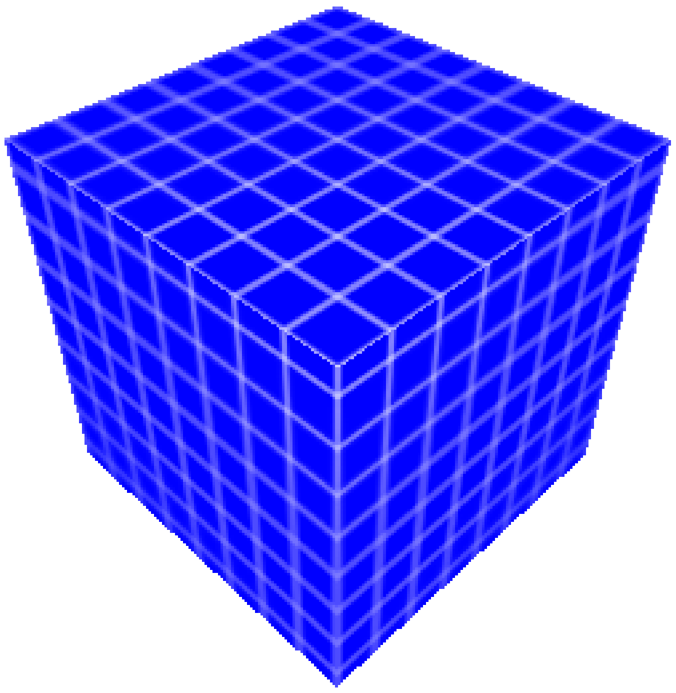


Note that holding down the `ALT` key while creating or resizing the box, you can temporarily ignore the editors snap to grid setting.

To create the new brush you have to press `ENTER`. The material that is currently selected in the *materials bar* is automatically applied to the new brush.

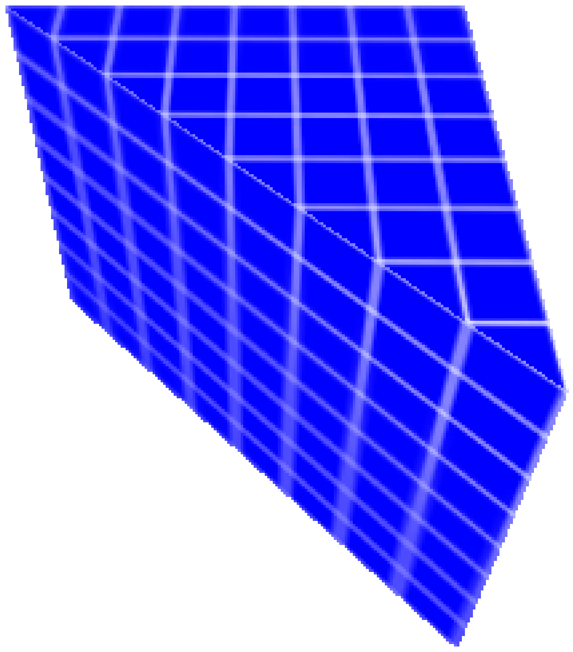
8.3.3 Different Shapes

Block



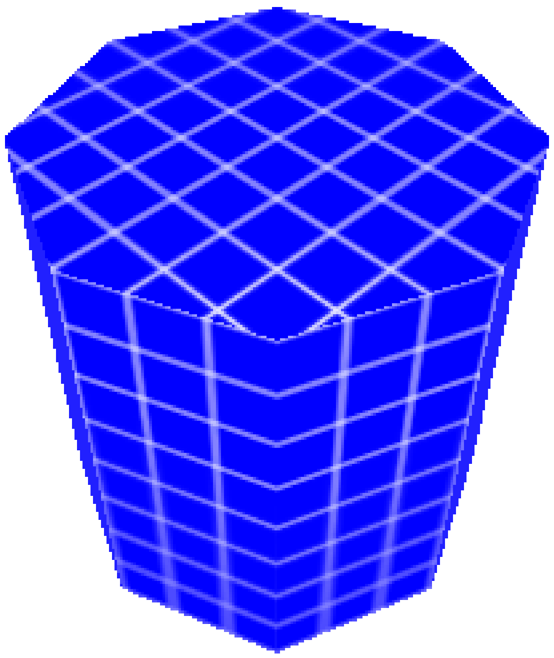
A block is a simple rectangular box with 4 sides.

Wedge



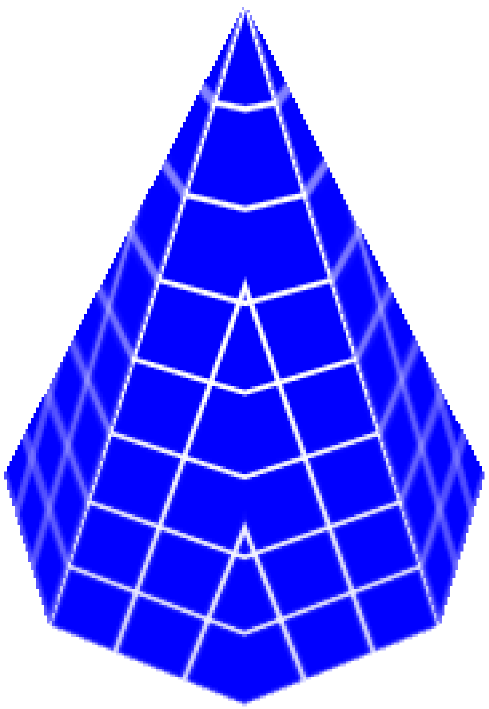
A wedge is a triangular box with 3 sides.

Cylinder



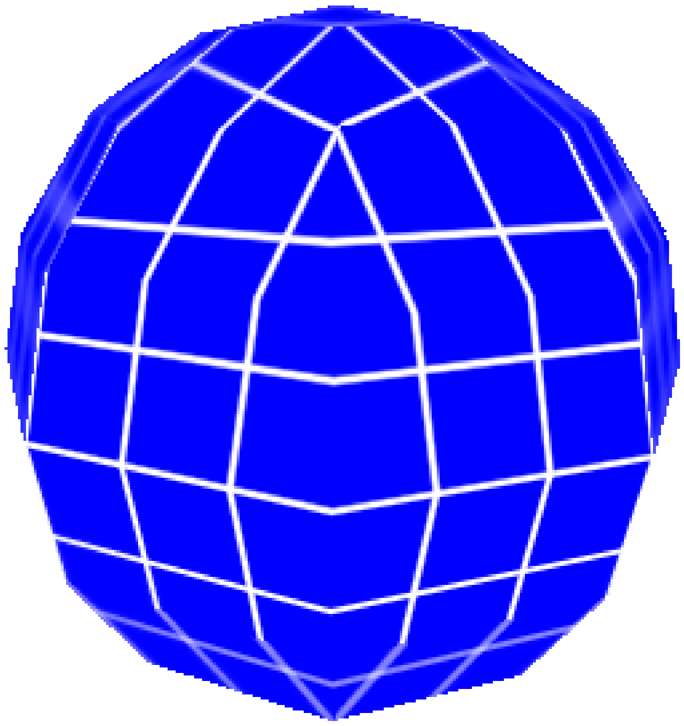
A cylindrical brush with variable sides from 3 to 32.

Pyramid



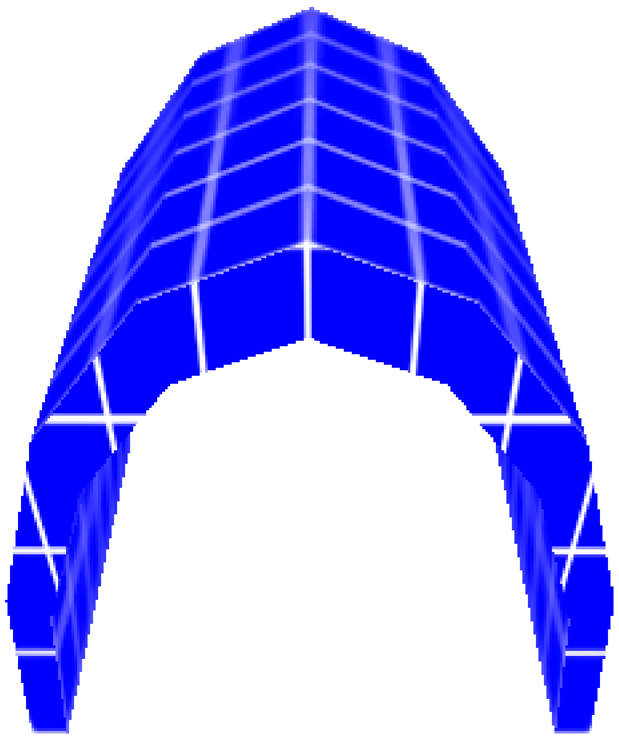
A pyramid is shaped like a cylinder, but the top vertices are all in one point. It can have 3 to 32 sides.

Sphere

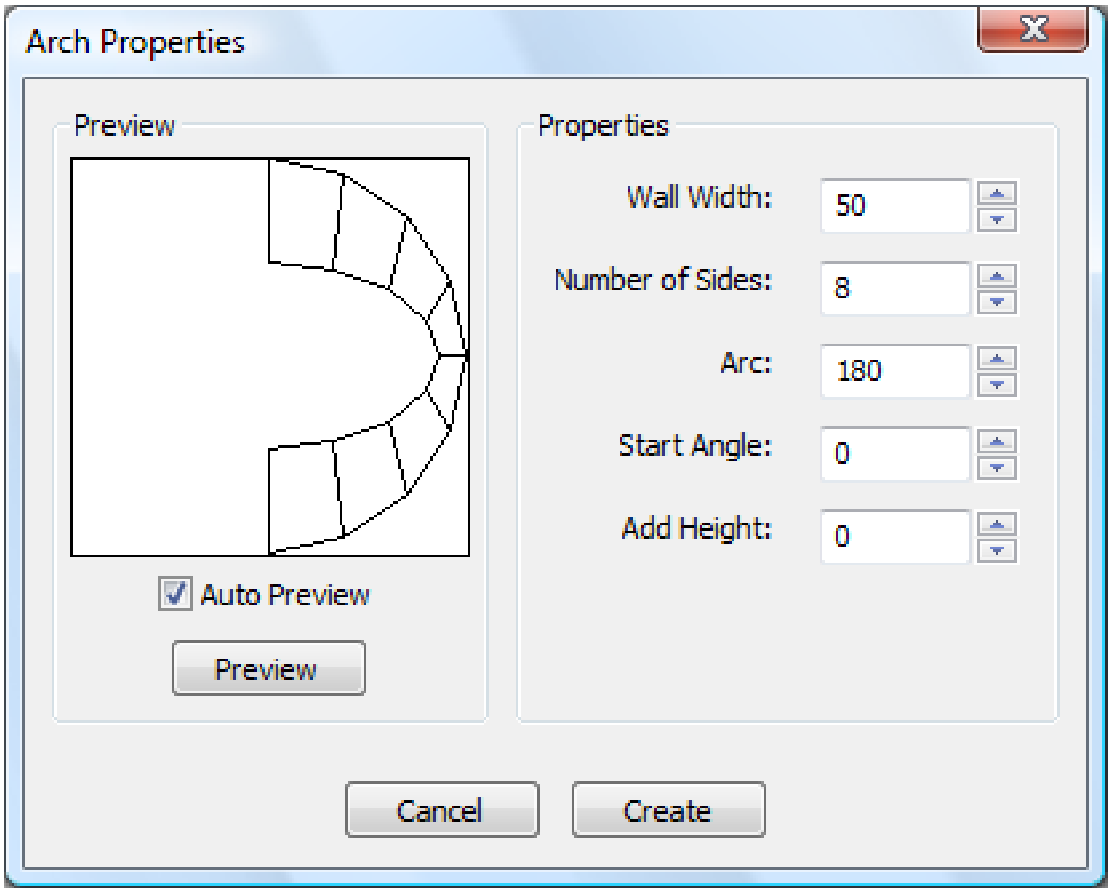


A spherical brush with 3 to 32 sides.

Arch



An arch with variable values such as wall width, side number and arch span. These values can be configured in a special arch diaglog.



Properties

Wall Width: 50

Number of Sides: 8

Arc: 180

Start Angle: 0

Add Height: 0

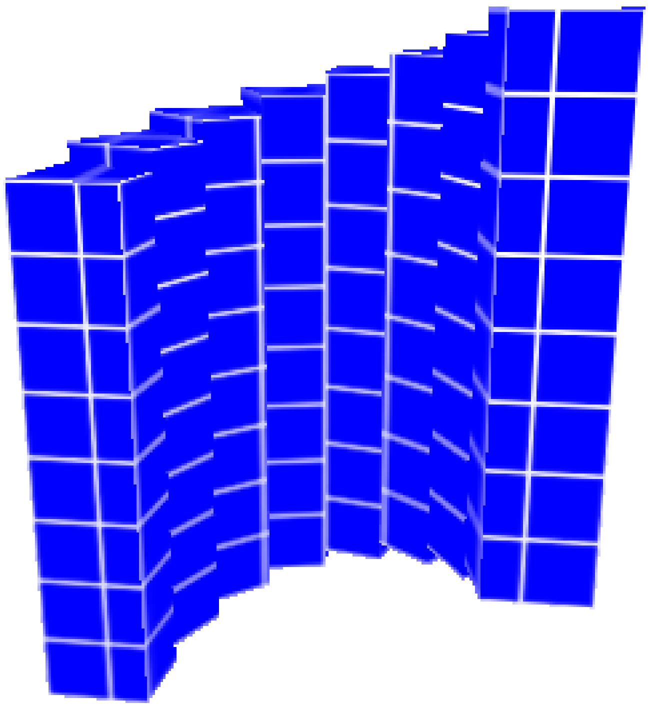
Wall Width Thickness of the archs wall.

Number of Sides Number of pieces the arch consists of.

Arc Span of the arch in degree.

Start Angle Start angle offset.

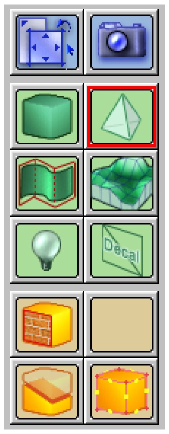
Add Height Height difference between 2 arch pieces (see screen below).



8.3.4 New Brush tool keyboard shortcuts

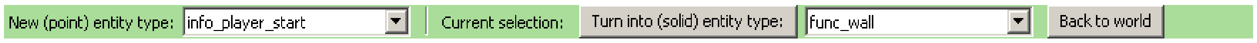
- `SHIFT+B`:
  - Activates the New Brush tool.
- `ENTER`:
  - If a brush box is opened → create the brush (only works in a 2D view).
- `ALT`:
  - If creating a brush box or moving/resizing it → temporarily deactivate snap to grid
- `ESC`:
  - Back to Selection tool

8.4 The New Entity Tool


The New Entity tool allows you to add both new point-based and new brush-based entities to your worlds. Please refer to the *Introduction to Editing* for a quick overview on point-based vs. brush-based entities. The *Entity Guide* lists and describes all entity types that are available with Cafu.

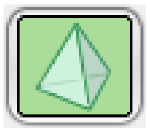
The New Entity tool is activated by either clicking on its related button on the tools toolbar, or by pressing the `Shift+E` keyboard shortcut.

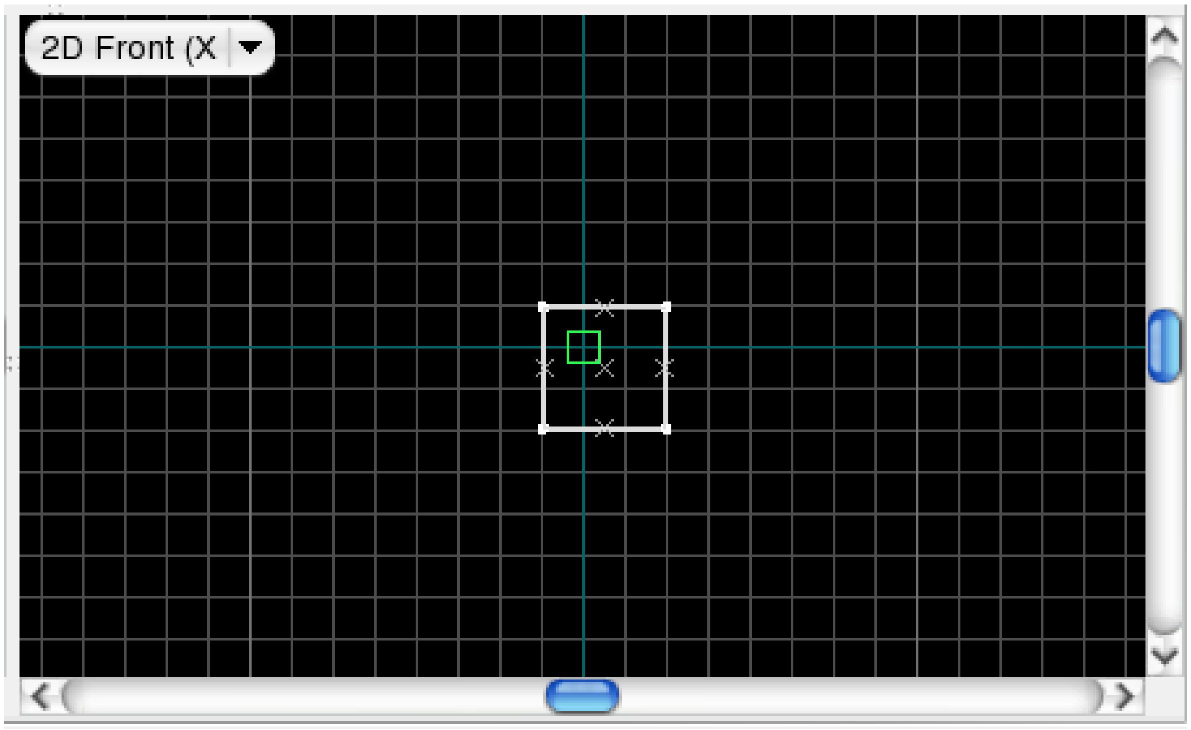
When you activate the tool, the tool options bar shows controls for creating new entities:



The left half of this bar is relevant for placing point-based entities, i.e. at `New (point) entity type`: you choose the type of the new entity that is to be created. The right half of the bar allows you to turn the currently selected brushes into brush-based entites and back. That is, the `Turn into (solid) entity type`: button turns world brushes into a brush-based entity of the type that has been chosen from the list to the right of the button, whereas the `Back to world` button reverses the process by turning the currently selected brush-based entity into regular world brushes again.

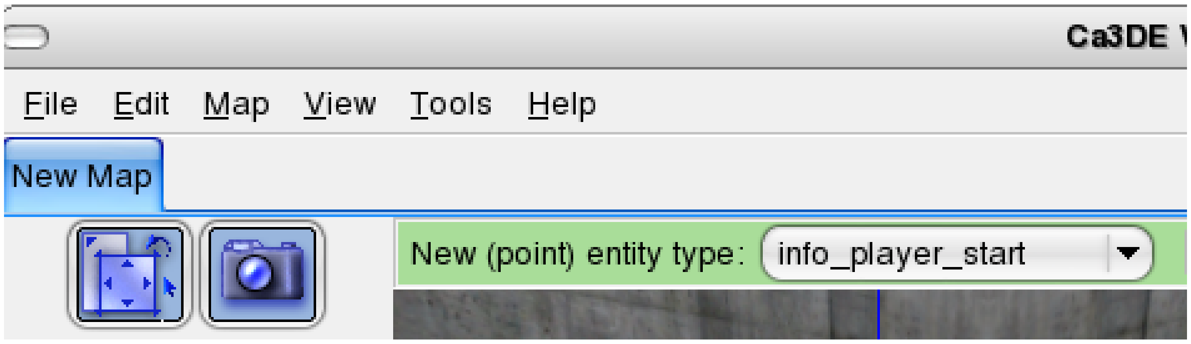
8.4.1 Placing Point Entities

1.  Select the Entity tool.

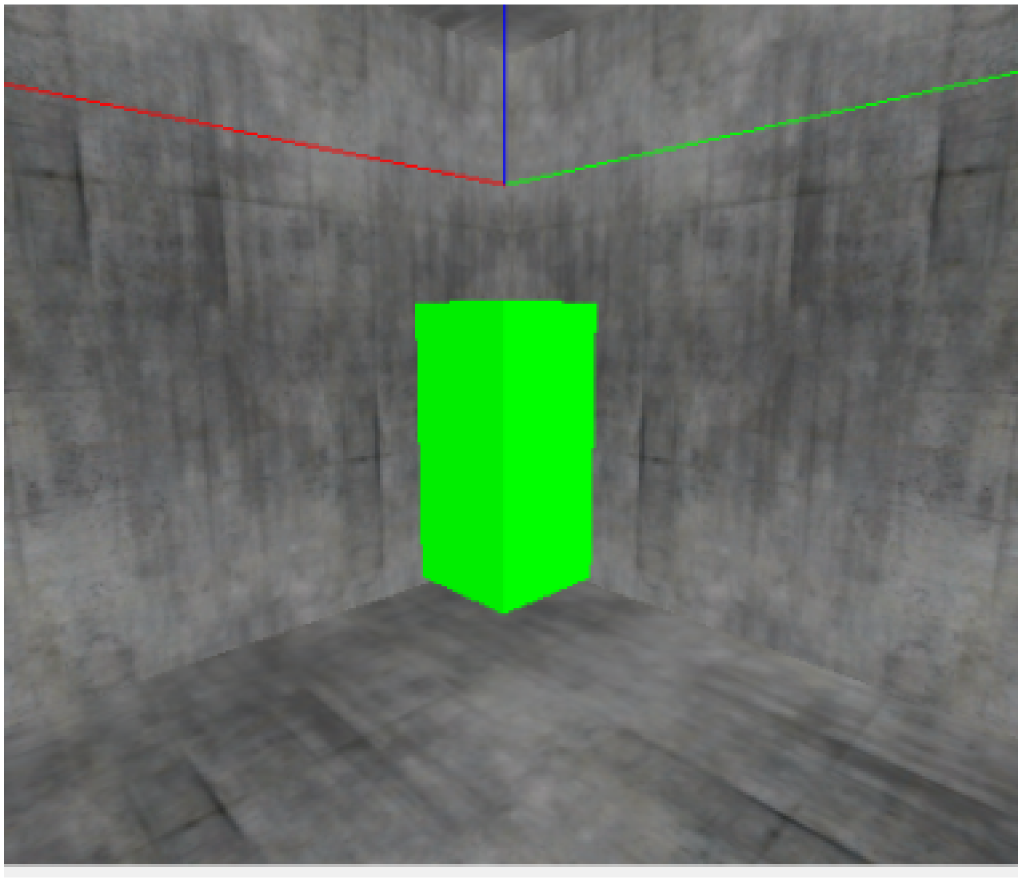
2. 

In one of the 2D windows, click where you want to place your entity. A green box will appear where you clicked. This is where the entity will be inserted.

3. Drag the green box to where you want the entity in the other views.

4. 

Select the type of entity you want to insert from the "New (point) Entity" dropdown box. I'm inserting an info_player_start.

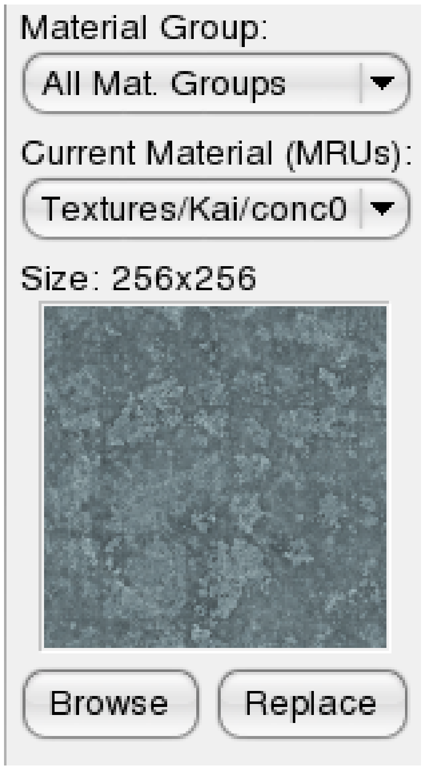


5. Put your mouse cursor over one of the 2D views, and press enter. The entity should appear. You should be able to see it in both the 2D and 3D views. *Note that other entities are different colours, shapes and sizes.*

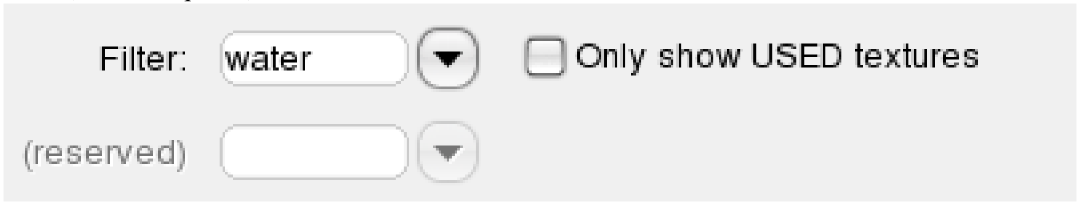
8.4.2 Placing Brush-Based Entities

In this example, I will be adding water to a very simple map. The process is very similar when inserting other brush entities.

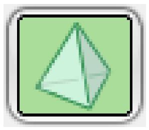
1. Select the *New Brush* tool.
2. In a 2D view, drag out a box the size you want your water to be. Adjust the size in the other 2D views.
3. *(This step is not necessary for most other entities)* Click the "Browse" button in the texture settings (under the tool icons). When the viewer pops up, change the filter to "wa-

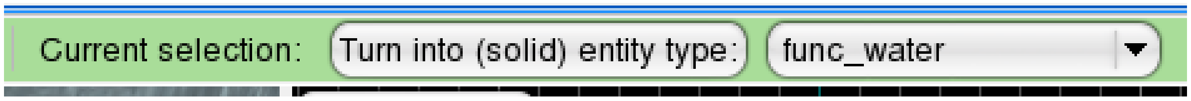


ter" (without quotes). Select one of the materials. →



4. With your mouse cursor over one of the 2D views, press Enter. The brush should be selected if not, select it with the selection tool.

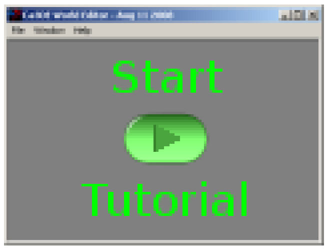
5.  Click the New Entity tool.

6. 

On the far right side, select "func_water" from the drop down box. Click "Turn into (solid) entity type".

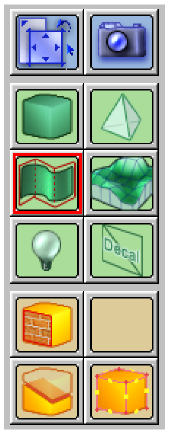
Now compile your map, and you should see water!

8.4.3 See Also



Flash Tutorial – A flash tutorial that demonstrates how entities of type `static_detail_model` are created and how their visual and collision models are properly set. The second part of the tutorial examines a collision model by example and summarizes how new collision models are created.

8.5 The New Bezier Patch Tool

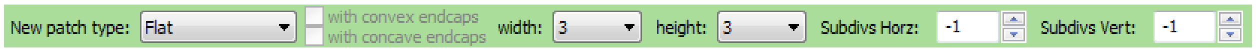


The New Bezier Patch tool is an easy to use tool, if you want to create curved surfaces such as pipes or uneven floors.

A Bezier Patch is in its pure form a plane with a grid of vertices (with variable grid density), that can be moved to change the form of the Bezier Patch.

To activate the tool, use the related icon from the tool bar or press `SHIFT+P`.

8.5.1 The Tool Options Bar



New patch type: Flat | with convex endcaps | with concave endcaps | width: 3 | height: 3 | Subdivs Horz: -1 | Subdivs Vert: -1

The Tool Options Bar lets you choose from different source forms, that a created Bezier Patch will adopt at the beginning. Possible forms are:

Flat A simple flat plane.

Cylinder A Bezier Patch bent into a cylindric form without bottom or top.

Open Box A rectangular box without bottom or top.

Half Cylinder A Bezier Patch bent into a half cylinder.

Quarter Cylinder A Bezier Patch bent into a quarter cylinder.

Edge Pipe Basic form to easily create edges of pipes.

Cone A Bezier Patch bent into a simple cone.

Sphere A Bezier Patch bent into a spherical form.

Convex Endcap Creates a quarter cylinder convex endcap.

Concave Endcap Creates a quarter cylinder concave endcap.

The Checkboxes **with convex endcaps** and **with concave endcaps** determine whether a newly created bezier patch should have fitting endcaps at creation.

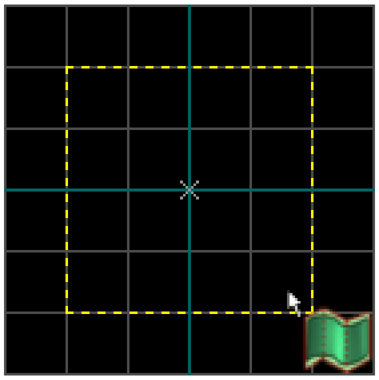
The fields **width** and **height** define the number of vertices on the two axis of the Bezier Patch. Higher values mean a higher density of vertices.

Note that these options are not available for all basic bezier patch forms.

The **Subdivs Horz** and **Subdivs Vert** controls adjust the subdivisions of a bezier patch in x or y direction. A higher value means a more seamless bezier patch surface. You can also use these two controls to adjust the subdivisions of a currently selected bezier patch.

Values of -1 mean, that the number of subdivisions is chosen automatically.

8.5.2 Creating a Bezier Patch

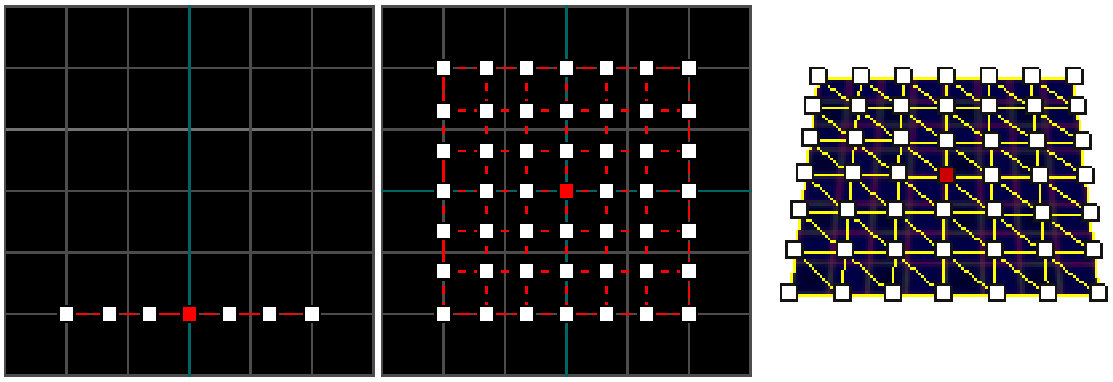


To create a new patch you have to drag a box in a 2D view that defines the size of the Bezier Patch (note that the box has to be spatial even if you want to create a **Flat** Bezier Patch). After the box is created you can then resize it to your liking. Finally press `ENTER` to create the Bezier Patch in its chosen source form.

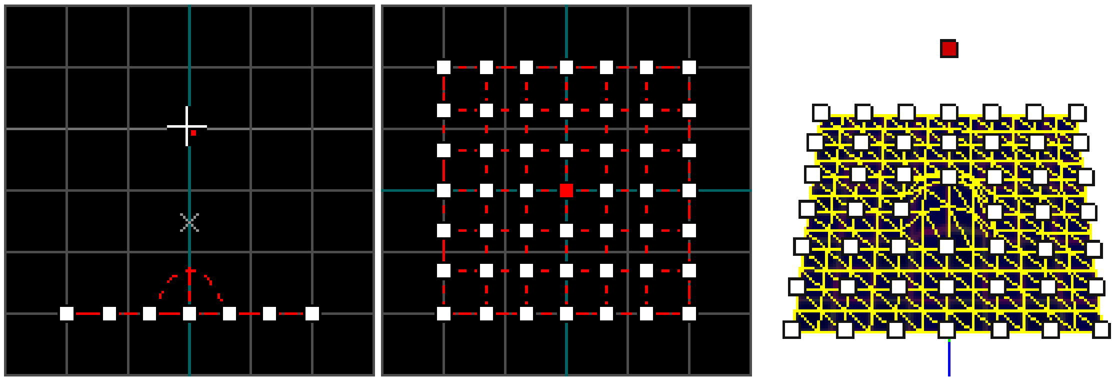
8.5.3 Forming a Bezier Patch

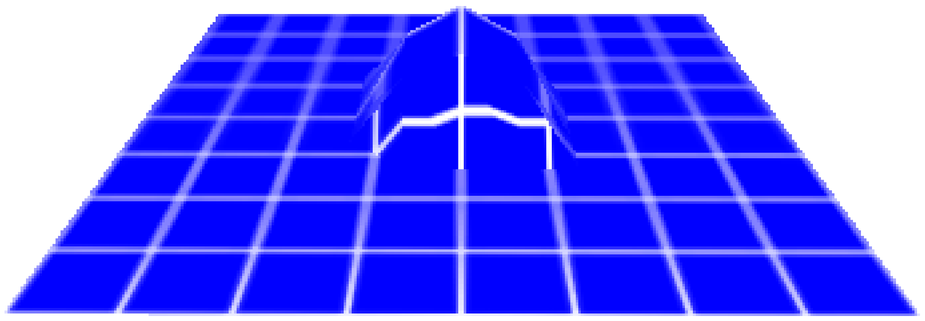
To form a Bezier Patch you have to activate *The Selection Tool* and select the patch you want to form.

Forming the patch is done by activating *The Edit Vertices (Morph) Tool*. Now you can see all control vertices of the Bezier Patch in the 2D views as well as in the 3D view. You can drag control vertices in both views to form the patch in the same way you drag vertices of brushes (see *here*).



In this example the center vertice is selected and moved up in a side view.





Altough only the center vertice has been moved, the Bezier Patch has changed its form creating a smooth bump instead of a peak.

8.5.4 Applying materials to Bezier Patches

Materials are applied onto bezier patch surfaces using *The Edit Surface Properties Tool*.

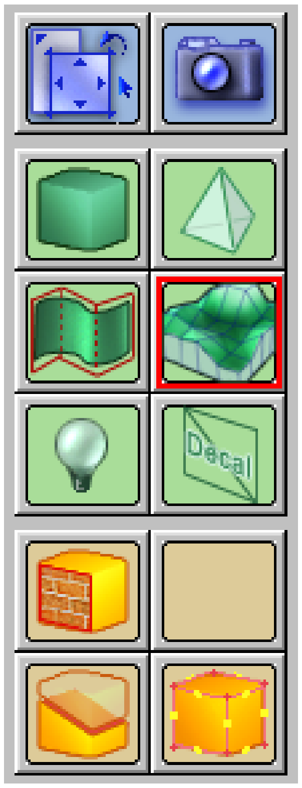
8.5.5 New Bezier Patch tool keyboard shortcuts

- `ESC`
  - Back to Selection tool.
- `ENTER`

   - If Bezier Box is opened → create Bezier Patch.

- ALT
   - Temporarily ignore grid snap while creating a Bezier box

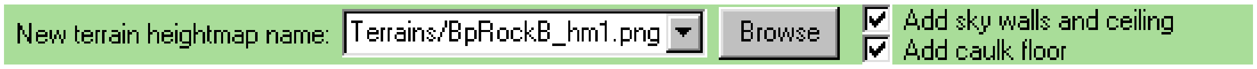
8.6 The New Terrain Tool



The "New Terrain" tool helps you to create new terrains. These terrains can be very large, have very good rendering performance, and are specially treated by the Cafu engine.

The "New Terrain" tool is activated by either clicking on its related button on the Tools toolbar, or by pressing the Shift+T keyboard shortcut.

When you activate the tool, the tool options bar shows controls for creating new terrains:



New terrain heightmap name: Terrains/BpRockB_hm1.png

☑ Add sky walls and ceiling
☑ Add caulk floor

The New terrain heightmap name determines the file that is used as the heightmap image that defines the shape of the new terrain. Use the combobox to select a previously used heightmap file, or use the Browse button to select a new file from disk. Note that the supported file formats include the most common image file formats (jpg, png, tga and bmp) as well as *Terragen* (ter) and *portable graymap* (pgm) files. Also note that heightmaps *need* to be grayscale and need to be power of two + 1. For example, a 512×512 heightmap needs to be 513×513 to work correctly in Cafu. A 1024×1024 needs to be 1025×1025. And so on. The section *Creating Height-Maps for your Terrains* explains how you can create your own height-maps for use here.

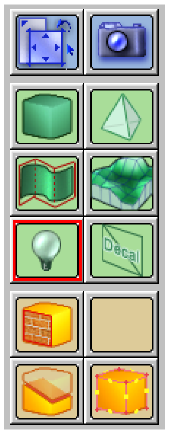
The two checkboxes to the right control whether sky walls, a sky ceiling and a floor are created automatically with the terrain. Although terrains are often used for large outdoor areas, they must still exist "inside a room". That is, terrains must be bounded by brushes that form four walls, a ceiling and a floor. With the two checkboxes, you can control whether CaWE should automatically create such bounding elements for you together with the new terrain (recommended!). You can later always edit or delete the so created walls (they are regular brushes), e.g. in order to make a doorway to some indoor or underground area of your map.

Finally, you have to pick a material for the new terrains surface, which you can conveniently do as usual with the Materials toolbar.

Note that all these attributes determine how the new terrain will be initially created. Of course, you can also review and change all these aspects for existing terrains later.

With everything being setup, you create the new terrain in the same manner as you create new brushes: Draw a box in the 2D views. The box determines the bounds of the new terrain, both laterally as well as the height of the lowest and highest points. When you're satisfied with the position and size of the box, press `RETURN` or right-click in one of the views to confirm the terrain creation.

8.7 The New Light Tool



The New Light tool is used to place light entities and is actually a subpart of the *New Entity tool*. The difference is, that the New Light tool can only place light entities, whereas the *New Entity tool* can place all entities.

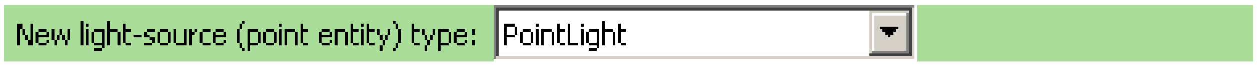
Lights are used to illuminate a map and come in two forms:

Static lights: These lights are precalculated by the editor when compiling a map. Lightmaps are built to define which parts of the level textures are illuminated and shadowed.

Dynamic lights: These lights are placed in the editor but their illumination and shadow casting is calculated dynamically by the engine during runtime. This makes them perfect to calculate light and shadow with movable objects, which is impossible with static lights. It is also possible to move the light source itself, which makes this a very flexible light source, but has also very high processing costs.

You can activate the New Light tool using the related icon in the tool bar or by pressing `SHIFT+L`.

8.7.1 The Tool Options Bar



New light-source (point entity) type: PointLight

In the options bar of the New Light tool you can choose what light you want to create.

There are 2 options available:

PointLight A static light source from which lightmaps are calculated.

PointLightSource A dynamic light source which is handled by the engine at runtime.

8.7.2 Creating a light

To create a light source, you can click on a location in the 3D view where the light source is instantly created.

If you click in the 2D views a point is created, which can be properly positioned before creating the light source by pressing the `ENTER` key.

8.7.3 Configuring a light

To configure the properties of a light source, you have so select the light with the *Selection tool* and choose **Object Properties** from the **Edit** menu or press `ALT+ENTER`.

The properties for the **PointLight** include:

Opening Angle Opening angle of this light source

Intensity red/green/blue Intensity of the different light colors

The properties for the **PointLightSource** include:

Dyn. light radius Radius of light dissemination

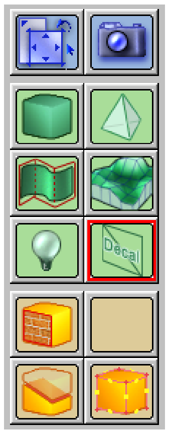
Dyn. light diffuse color Color of diffuse light

Dyn. light specular color Color of specular light

8.7.4 New Light tool keyboard shortcuts

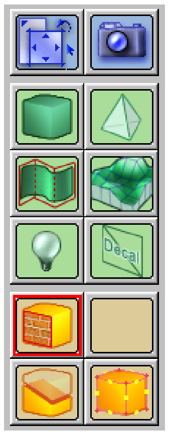
- `ENTER`
  - create light source after placing it in a 2D view
- `ESC`
  - switch back to the Selection tool
- `ALT+ENTER`
  - if light source selected → open its properties dialog

8.8 The New Decal Tool



The New Decal tool is not yet implemented.

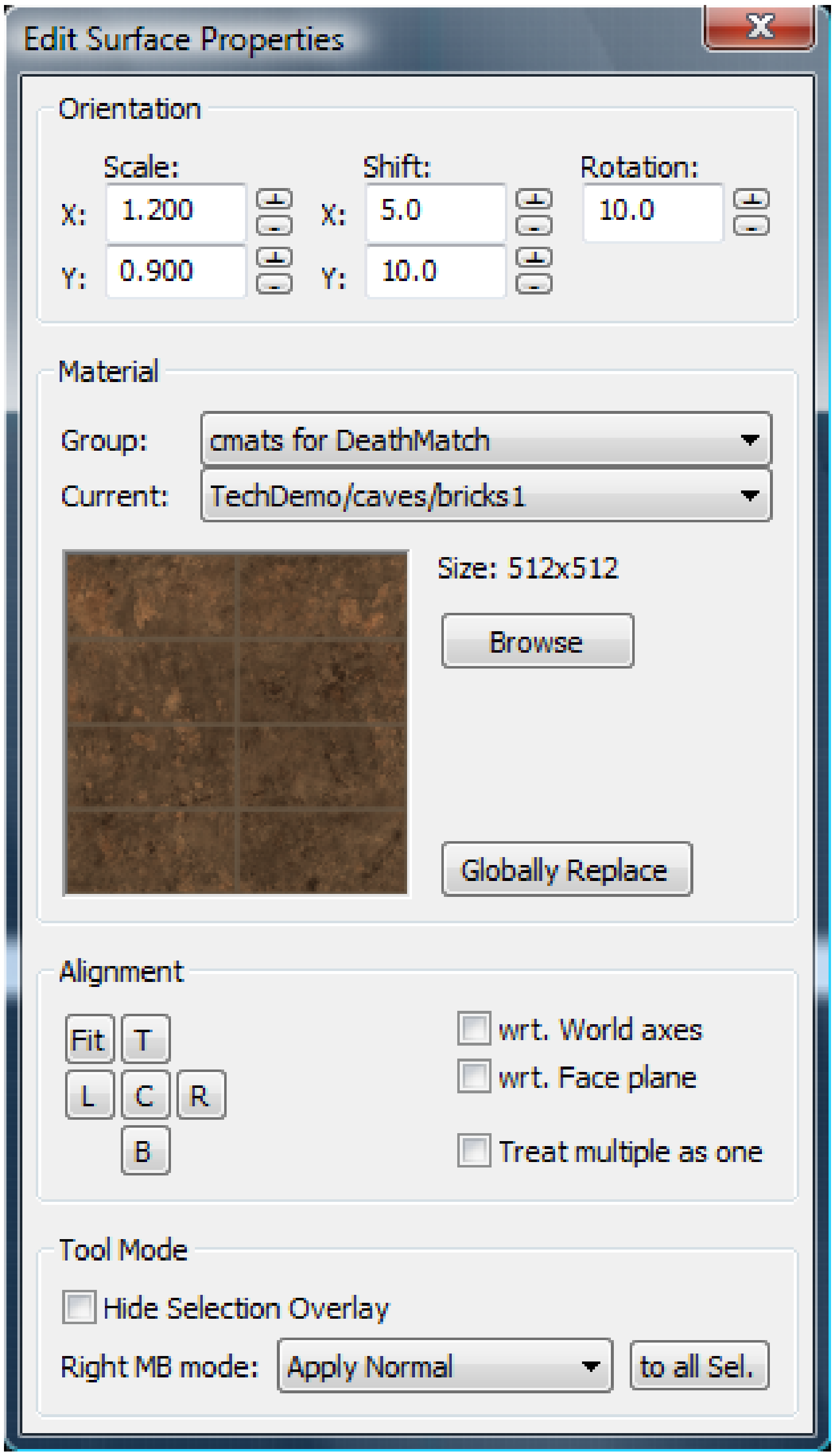
8.9 The Edit Surface Properties Tool



The Edit Surface Properties tool gives you control over the *surfaces* of *Bezier patches* and *brush faces*. You can use it to apply and change materials and modify their scale, shift and rotation on the surface. Sophisticated controls are available to determine these and other attributes of a surface.

The Edit Surface Properties tool can be activated by clicking on the related icon in the Tools toolbar or by pressing the `SHIFT+A` shortcut key.

8.9.1 Left-Clicks: Selecting and Picking



Scale: X: 1.200 Y: 0.900
Shift: X: 5.0 Y: 10.0
Rotation: 10.0

Group: cmats for DeathMatch
Current: TechDemo/caves/bricks1
Size: 512x512

☐ wrt. World axes
☐ wrt. Face plane
☐ Treat multiple as one

☐ Hide Selection Overlay
Right MB mode: Apply Normal

A normal click with the left mouse button on a surface (Bezier patch or brush face) in the 3D view accomplishes two tasks:

- it *selects* the surface (you can see how it becomes highlighted by a reddish overlay), and
- it *picks* the properties of the surface and updates the values in the related dialog accordingly (the **Orientation**

and **Material** sections are updated).

As usual, you can hold the `CTRL` key to select more than one surface at a time. Please note that `CTRL`-clicked surfaces are selected, but not picked. That is, only the first selected surface (left mouse button click without `CTRL`) implies the pick operation.

In fact, selection and picking are actions that are independent of each other:

Both: A normal left mouse button click performs both.

Selection only: Holding the `CTRL` key while clicking toggles the selection, but doesn't pick.

Picking only: You can also pick but not select by holding the `ALT` key during the mouse click, which activates the eyedropper mode. The properties of such clicked surfaces will appear in the dialog, but their selection status won't change.

Finally, you can also hold the `SHIFT` key during a left mouse button click. The `SHIFT` key does not add anything special when the click is on a Bezier patch, but for brush faces, it causes *all* faces of the affected brush to become selected. (If `CTRL` is not held at the same time, the clicked brush face will also be picked for updating the dialog values.) You can combine the `SHIFT` with the `CTRL` key in order to toggle the selection status of all faces of the brush.

Brushes that were selected when the Edit Surfaces Tool was activated are automatically changed into face selections.

You can hide the reddish overlay of selected surfaces by checking the **Hide Selection Overlay** checkbox in the **Tool Mode** section of the dialog. This improves the visual perceptibility of the surfaces in some cases, which in turn can be helpful when using this tool.

8.9.2 Orientation and Material

Once you have selected one or more surfaces, you can modify their orientation and material attributes.

If you have multiple surfaces selected at the same time, the new value of the changed attribute is applied to all of them, while all other attributes are left alone. This allows you for example to replace the material or to set a common scale on a number of surfaces all at the same time, while the individual shift, rotation and other surface aspects are left alone.

8.9.3 Right-Click Application

The orientation and material attributes in the dialog can also be applied to surfaces by right-clicking on them. The right-clicked surface needs not be selected for this to work, and all attributes (orientation and material) are applied at the same time. This feature makes the application of the attributes in the dialog to a number of surfaces quick and convenient.

With right-click application, the way in which the attributes are applied to the surfaces can be modified at the **Right MB mode** choice in the **Tool Mode** section of the dialog. The following options are available:

Apply Normal Applies the material and the orientation attributes in the dialog "normally" to the surface.

Apply Material only Applies the material, but not the orientation attributes.

Apply View Aligned This mode works like a slide-projector: The material is projected onto the clicked surface, where the base of the cameras view pyramid defines the plane of projection. (Contrary to a real slide-projector though, parallel rather than perspective projection is employed. Variants of parallel projection are also employed by the other apply modes and are more appropriate for the task at hand.) The orientation attributes are also taken into account. This is a great mode for texturing a rocky wall or any other irregular shape (that consists of multiple adjacent surfaces) seamlessly.

Apply Edge Aligned This mode makes sure that when you click on a surface *next* to the previously picked one (i.e. left-clicked, possibly with `ALT`), the material is seamlessly aligned across the common edge. This is a worthwhile feature especially when the two surfaces are not coplanar. (It even works when the clicked surface is not immediately adjacent to the other.) **(FIXME!)** What exactly are the restrictions when Bezier patches are used?

Apply Projective Like Apply Normal, but this mode also applies the *texture planes U- and V-vectors* of the picked surface to the clicked surface. This mode is the generalization of both the Apply Normal as well as the Apply View Aligned modes, and as the underlying technique is a mathematical projection, it is called Apply Projective. Refer to the advanced section below for more technical information. This is the mode of choice whenever you want to texture a Bezier patch exactly like an adjacent brush face. For example, if you have a wall or a floor that is made of both brush faces and Bezier patches, and you want to apply the material on the brush faces seamlessly to the Bezier patches, just pick up the surface attributes from a brush face, and Apply Projective to the Bezier patches.

When `SHIFT` is held during a right-click on a brush surface, the application is performed on all surfaces of the brush.

The button **to all Sel.** is equivalent to (but more convenient than) right-clicking on all currently selected surfaces with the current **Right MB mode** setting.

8.9.4 Alignment

The materials of selected *brush faces* (not Bezier patches) can be automatically aligned at the top, bottom, left or right edge of the face by the related button in the **Alignment** section of the dialog. The material can also be centered or made fit on the brush face. In all cases, the effect is achieved by auto-adjusting the Shift and/or Scale attributes of the selected brush faces appropriately.

(**(FIXME!)** Re-pick the first selected surface in order to update the dialog attributes!)

The **Treat multiple as one** checkbox determines whether multiple selected surfaces are treated, with regards to their spatial extends, as multiple individuals or as a single merged one when one of the Alignment buttons is pressed.

Fit on Bezier patches

For Bezier patches, the (highly useful!) **Fit** button works slightly differently than for brush faces: Although in both cases the material will be made “fit” to the underlying surface, Bezier patches are then in a different mode regarding texture-coordinate generation than brush faces.

This is because for brush faces, you could have achieved the same result by manipulating the Scale and Shift manually. With Bezier patches, which can be curved to any shape (e.g. cylindrical or spherical), the same mathematics cannot achieve the same effect, and thus using the **Fit** button on Bezier patches puts them into Fit mode that yields the desired result.

The essence of Fit mode is that the Scale attributes now indicate the number of repetitions of the material along the surface, rather than the number of texels per world unit as usual.

Because of this difference, while applying the surface properties of such Bezier patches to other Bezier patches is straightforward and intuitive, applying the same attributes to brush faces is sometimes not possible without distortion. Such cases are easily fixed though by clicking and editing the affected brush surface.

The line “Mode: Fit” near the Orientation controls indicates that the current surface values were taken from a Bezier patch that was in Fit mode.

You can “revert” the mode of such a Bezier patch surface back to normal by picking the surface attributes of a brush face and applying them to the Bezier patch in **Apply Projective** mouse-button mode.

Plane Indicators

The **wrt. World axes** checkbox indicates whether the texture plane of the surface happens to be parallel to one of the three major planes through the world axes.

The **wrt. Face plane** checkbox indicates whether the texture plane of the surface is parallel to the spatial plane of the surface. This is never true for Bezier patches (they in general have no inherent spatial plane), and normally always true for brush faces; however you can use the **Apply Projective** right-click apply mode in order to force any texture plane onto any surface.

8.9.5 How do I . . .

. . . texture brush faces?

Individual brush faces are easily textured by left-clicking them to pick their attributes into the dialog. Then you use the controls in the dialog to directly modify the orientation, alignment, etc. The changes become immediately visible on the selected face in the 3D view. You may check the **Hide Selection Overlay** checkbox in order to hide the selection mask.

For texturing a larger area of your map where many brush faces are to be textured in a similar style, you should first texture one of the surfaces as described above, then use the *Right-Click Application* feature to apply the texture to all desired surfaces in the 3D view quickly. Make sure that you understand the apply modes of the right mouse button in order to get the best out of them.

Repeat this with other textures of your choice, and fine-tune them individually in the end.

. . . texture Bezier patches?

This is as easy as texturing brush faces, but sometimes Bezier patches are textured for a specific purpose:

Bezier patches that are a part of a floor or ceiling plane are often to be textured so that the transition to the adjacent brush is seamless. This is easily achieved by picking the texture from the nearby brush, select **Apply Projective** as the **Right MB mode**, then right-click apply the material to the Bezier patch.

Bezier patches that represent bulges, dents or small terrains that are expected to blend seamlessly into neighboring brushes are also well approach by the projective or view aligned apply modes.

Pipes, pillars, wall arches and similar objects are best textures by using the **Fit** alignment button. The material then covers them naturally, and you can conveniently set the number of texture repetitions by the Scale values in the dialog.

. . . deal with the "Picking [. . .] is not possible" message?

This message can occur when picking (left-clicking) Bezier patches that were created with previous versions of CaWE, or Bezier patches that were imported from other games and file formats. Such Bezier patches got custom texture-coordinates assigned and are in "Custom" tex-coord generation mode; you may refer to section *Advanced Considerations* below for more details.

Nothing is inherently wrong with Bezier patches in Custom mode, you just cannot pick their surface properties and apply them to anything else. As soon as you assign surface properties from another brush face or Bezier patch (that is not in Custom mode) though, the Custom mode is overwritten with the newly applied properties, and the Bezier patch acts normally then.

. . . deal with “Mode: Fit” orientation attributes?

Bezier patches that are newly created or whose material has been applied using the **Fit** button are in the “Fit” texture-coordinate generation mode, whose essence is that the Scale attributes now indicate the number of repetitions of the material along the surface, rather than the number of texels per world unit as usual.

Refer to section *Fit on Bezier patches* above and *Advanced Considerations* below for more details.

8.9.6 Advanced Considerations

The Edit Surface Properties tool has been written in order to make the texturing of brush faces and Bezier patches simple and consistent. However, brushes and patches are inherently different, and therefore you can do things with the one that you cannot do with the other, and vice versa. This is true for geometric modelling but also has consequences for texturing them with this tool. Moreover, the tool offers several advanced ways to apply textures to the different kinds of surfaces, and you might be interested to learn more on how they work.

This section first describes the texture-coodinate generation modes that the tool internally uses, then explains the details of the plane projection mode.

The Tex-Coord Generation Modes

Each surface (brush face or Bezier patch) is internally assigned a texture-coordinate generation mode that determines how the texture-coordinates at the vertices of the surface are computed:

Plane Projection is the most frequently used and most important mode. It is *always* used on brush faces (which can never use another mode), but it can also be used on bezier patches. The Plane Projection mode is explained in greater detail below.

- The **Fit** mode is special to Bezier patches, and can only be used on them. Whenever you press the **Fit** button in the dialog, the tex-coord gen. mode of the selected bezier patch is changed to this mode. It is typically used on pipes, columns and similarly shaped objects, making sure that the material naturally fits into the surface of the object. The number of repetitions can be given by the “Scale” parameter in the dialog, which therefore has a different meaning from the Plane Projection mode. (When the **Fit** button is pressed for brush faces, a visually similar operation is performed in Plane Projection mode.)

Custom is reserved for future use and applies whenever the user has specified custom UV texture-coordinates for the vertices of a surface that cannot be achieved with the other modes. It is also used on Bezier patches that were created when the CaWE map file format did not support the tex-coord generation modes. This is true for older CaWE maps and for maps that are imported from other games or file formats.

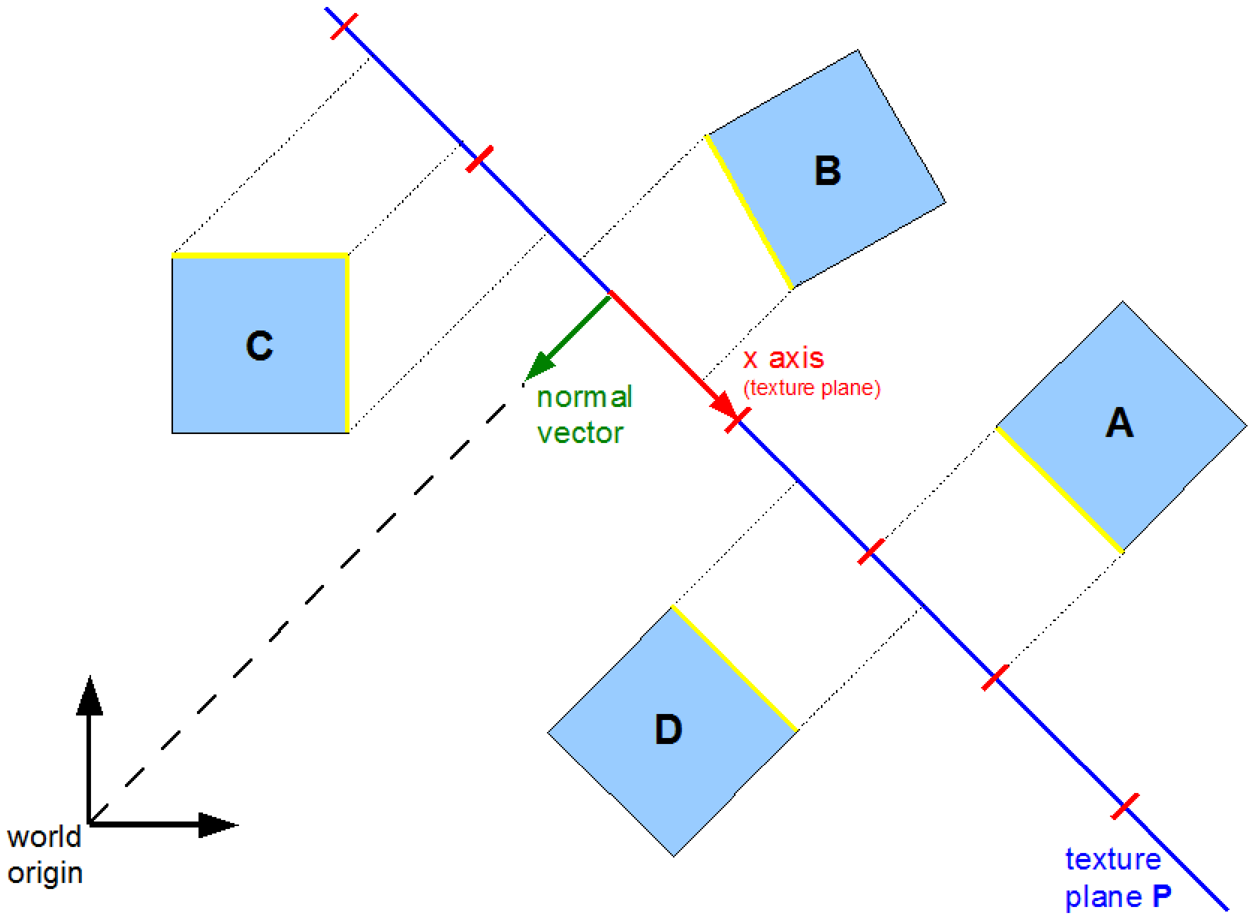
When you pick the attributes of a surface into the dialog, it also remembers the tex-coord generation mode of that surface. Picking is only possible when the surface is in Plane Projective or Fit mode, never in Custom mode. Applying previously picked attributes to another surface (either a brush face or Bezier patch) is always possible, although applying “Fit” mode attributes to brush faces (which can only be in Plane Projective mode) can yield unexpected results. **(FIXME!)**

Picking and therefore applying attributes that are associated with Custom mode to any surface is not possible at all.

The Plane Projection Mode

In Plane Projection mode, the texture-coordinates at the vertices of the desired surface are determined by an orthogonal projection of a texture plane P onto the vertices. The parameters of the plane are determined by the Orientation controls in the dialog.

The following figure presents the setup in a top-down view:



The texture plane P is indicated by the dark blue color, its normal vector in dark green. The x-axis span vector is indicated in red, as well as integral multiples of its length in both directions. The y-axis span vector is not included in the image, as it points up or down, into or out of the figure.

A texture plane is obtained by picking a surface in Plane Projective mode. For example, texture plane P in the image was obtained by left-clicking on the highlighted face of brush A.

Note that the x-axis represents one width of the material that is to be applied to the surfaces: If you change the Scale values in the dialog, the length of the x-axis (and y-axis) changes, too, implying that more or fewer repetitions of the material span a given distance in the world. Changing the Shift values in the dialog moves the x-axis arrow and its multiples along the plane. If rotation was applied, the x-axis would rotate “into” our “out of” the image, while the y-axis became visible.

In the dialog, the **wrt. World axes** and **wrt. Face plane** checkboxes indicate whether the dialogs last picked plane happens to be parallel to the major world axes and/or parallel to the physical surface of the picked-from face.

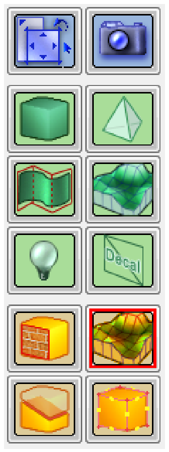
The dotted lines indicate how the material would be applied from the texture-plane to the objects surfaces. At object A there is a natural fit, because integral multiples of the material match the extends of the object. Object D is also natural case, but the material would appear shifted on it’s surface. You would have to modify the Shift attributes in order to align it with the extends of the surface.

Note that objects B and C are special cases: When you normally apply the attributes of plane P to their surfaces, P is moved and rotated first so that it gets parallel to the desired surface. This is what the **Apply Normal** mode of the right mouse button does.

The **Apply Projective** mode however does not include the additional move and rotate step, and applies texture-coordinates *directly as indicated in the figure*. This is sometimes a very useful feature in order to achieve certain effects, for example with texturing Bezier patches in some cases.

The **Apply View Aligned** mode works similarly, except for the fact that plane P is always forced to be parallel to the base of the view pyramid of the currently active 3D camera.

8.10 The Edit Terrain Tool



The Edit Terrain Tool allows users to easily manipulate the surface structure of a terrain.

The surface structure of a terrain is stored in a height map containing elevation values that build up the terrain. Each elevation value lies within a certain value range whereas higher values mean a higher terrain elevation at this position (please refer to the Heightmap Wikipedia article for more information on height maps).

To visualize this elevation values 3-dimensionally the engine create a smooth terrain surface model from the height map.

This tool lets the user modify the elevation values of a terrain by presenting them as a 2-dimensional height map that can be edited similar to a paint program as well as a 3-dimensional surface structure.

You can activate the Edit Terrain tool by clicking on the related icon marked in the left screen or by pressing the `SHIFT+F` keyboard shortcut.

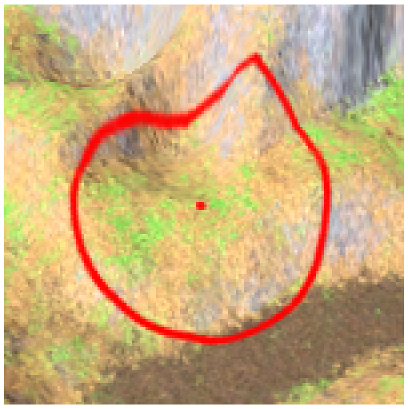
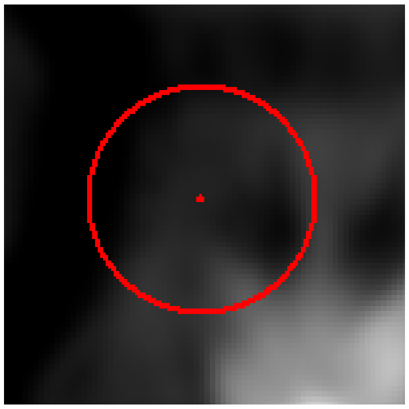
To create an editable terrain please refer to *The New Terrain Tool*.

8.10.1 General

The tool is mainly composed of three parts:

- The 2-dimensional terrain height map representation in the top-down 2D view.
- The 3-dimensional terrain surface representation in the 3D view.
- The terrain editor dialog.

The 2D and 3D view let you select and modify terrains with your mouse and therefore incorporate the tools main functionality. The dialog lets you change tool parameters and select different terrain editor sub tools (informations on tool parameters and sub tools are explained in detail below).

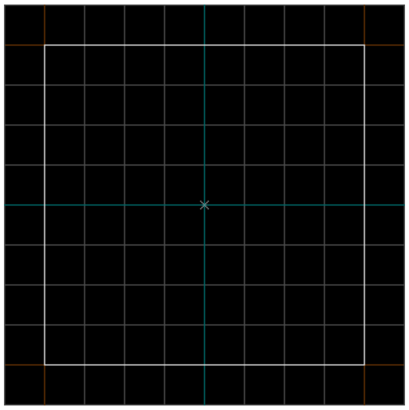


The terrain editor always works within a adjustable circular area that is visualized in the top down 2D view as well as in the 3D view as seen in the images on the right.

To modify a terrain you first have to select one by clicking on it. Then you can see the red circle at the position of your mouse cursor (if it is inside the bounds of the selected terrain) showing the area of the terrain that will be affected by the currently active sub tool.

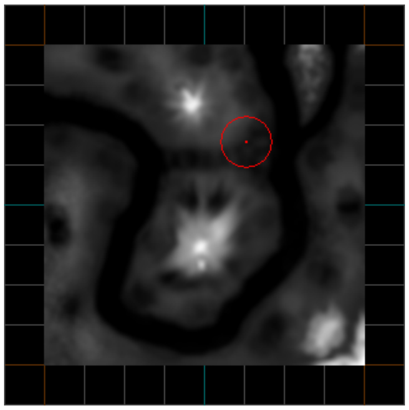
By left clicking you can modify the terrain in the area of effect. You can even hold down the left mouse button and move the mouse to easily modify the terrain along your mouse movements.

This tool handling is the same in the 2D and 3D views and allows to easily switch between 2D and 3D view editing according to the user preference.

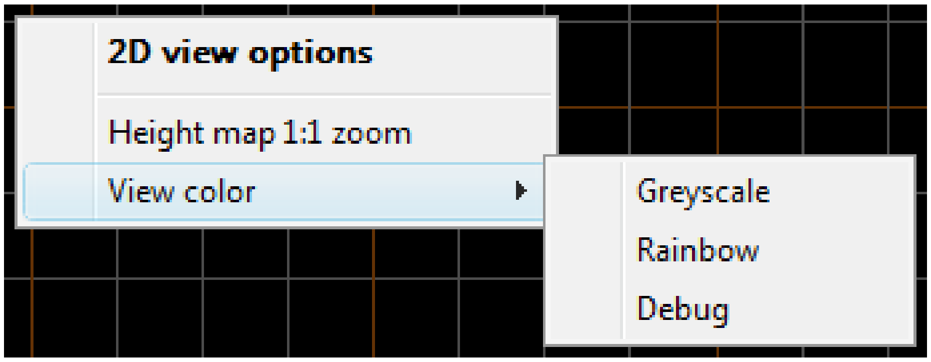
8.10.2 The (top down) 2D View


When a terrain isn't selected by the terrain editor it is rendered like every other brush so just its center point and bounding box is visible (in the same way as it is rendered if the terrain editor tool is inactive).

To select a terrain for editing you have to click either on the edges of its bounding box or on its center point in the same way as with *The Selection Tool*.

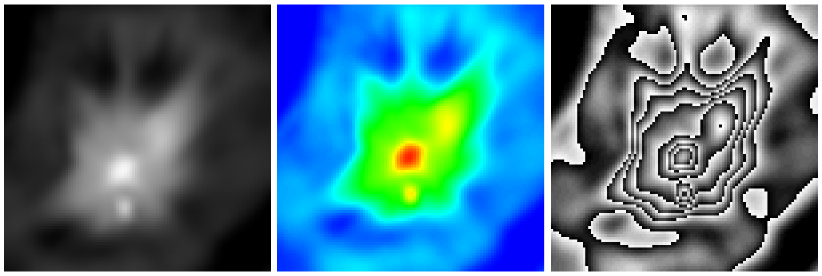

When a terrain is selected its height map is rendered in a gray scale mode in the 2D view. Looking at the image to the left you can see that higher elevations are rendered brighter whereas lower elevations are rendered darker. This lets the user easily assess and modify a terrains surface structure even from a non 3-dimensional view.

The tool cursor (the red circle) shows the current tool position and affected area and will always be set to the mouse position in the 2D view (as long as the mouse is positioned inside the bounding box of the terrain).



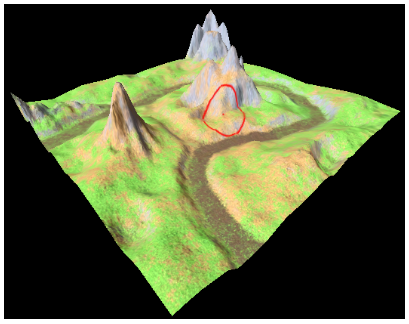
The 2D view has a context menu when the terrain editor tool is active. It lets you adjust the appearance of the terrain in the 2D view.

- Height map 1:1 zoom: Zooms the 2D view so each elevation value of the terrains height data will be rendered as one pixel in the 2D view.
- View color



- Greyscale: Renders the height map in a greyscale gradient from black to white where black represents the lowest elevation and white the highest elevation.
- Rainbow: Renders the height map in a rainbow gradient from blue to red where blue represents the lowest elevation and red the highest elevation.
- Debug: Renders the height map in repeating greyscale gradients from black to white. This mode is useful to make detailed terrain modifications where the spectrum of the other color gradients would be to small to see the differences.

8.10.3 The 3D View


A terrains surface structure is rendered in the 3D view surface regardless if the tool is active or inactive or if the terrain is selected by the terrain editor or not.

The only difference is that you will see the circular tool cursor on the terrains surface if the terrain is selected and the tool is active.

To select a terrain in the 3D view you just click on it in the same way as with *The Selection Tool*.

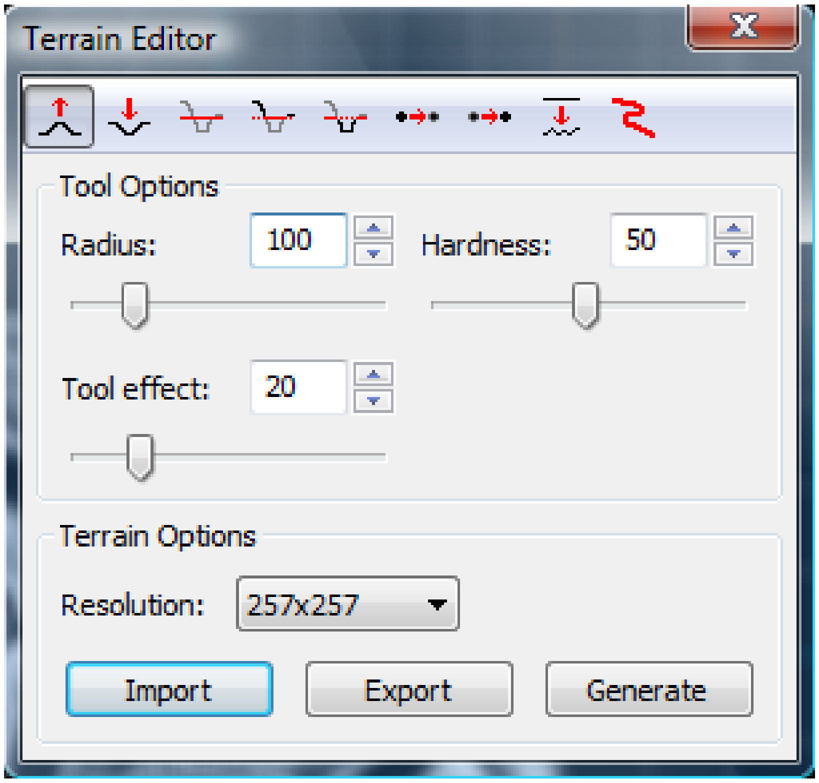
The tool position is placed where the mouse cursor hits the terrain, so in contrary to the 2D view where you work from a top down view onto the terrain, you can modify the terrain 3-dimensionally and easily work on the slope of a mountain for example.

An exception of this rule takes effect when the tool is actively used (e.g. the user flattens a part of the terrain). To avoid "jumping" of the tool cursor it is no longer placed where the mouse cursor hits the terrain but instead on a even plane. This makes using the tool in the 3D view more intuitive.

Since the terrain editor tool visualizes changes to the terrains height data in real time, the surface of the terrain must also be updated each time the user modifies the terrain.

Because updating the terrain surface is a costly operation, a faster operation is used to update the terrain surface while actively working with a tool (e.g. raising parts of a terrain with the left mouse button down). In consequence the terrain surface will be updated with smaller detail in real time and the final high quality terrain surface is calculated when the user stops using an editing tool (releases the left mouse button).

8.10.4 The Terrain Editor Dialog



Where the views allow you to modify the terrains height data at the tool position, the terrain editor dialog allows you to select different sub tools of the terrain editor and to adjust parameters for this tools. Furthermore you can change properties of the whole terrain with it.

At the top of the dialog you can see the toolbar with buttons for each sub tools (these sub tools are explained in the next chapter).

Following are 3 tool option sliders that let you adjust tool functionality:

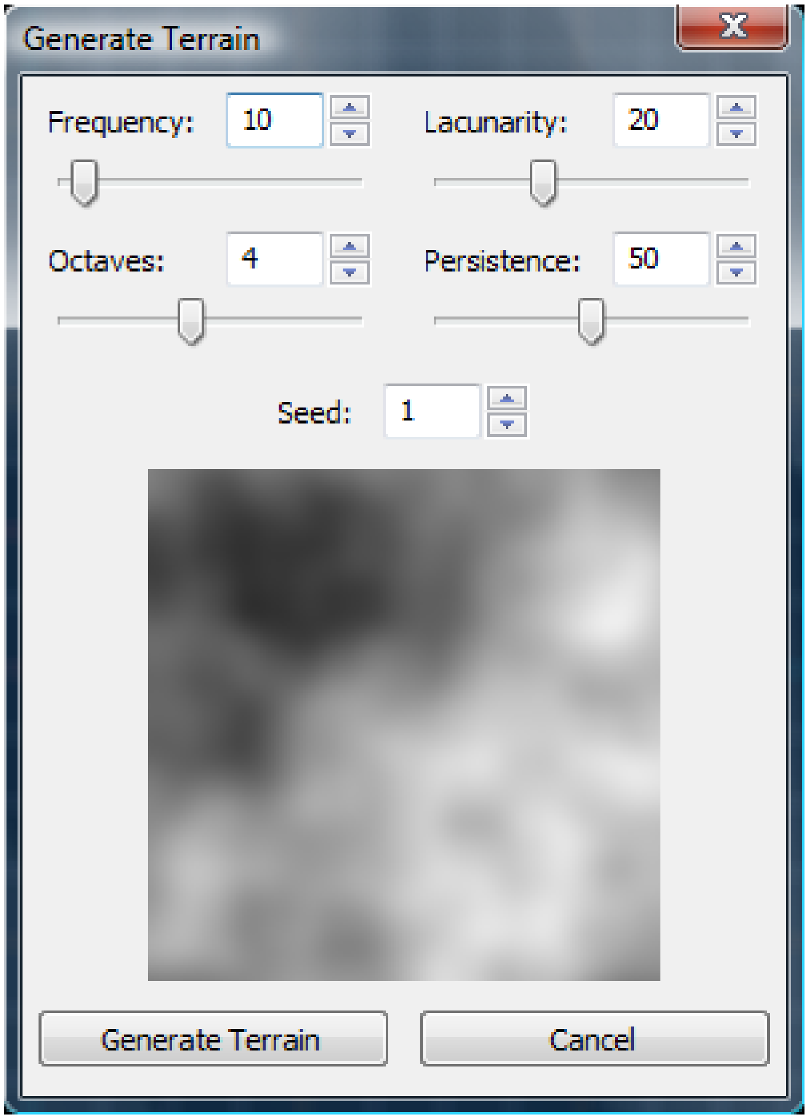
- Radius: The radius determines the size of the circular area of effect at the current tool position.
- Hardness: The hardness describes if the changes made in the circular area of effect should smoothly blend into the surrounding or have a hard edge at the border of the area of effect.
- Tool effect: The tool effect has different meanings for different sub tools, but can usually be pictured as the intensity with which the tool is used.

At the bottom of the dialog you can find options that affect the whole terrain.

- Resolution: The resolution of the terrain height data (elevation values). This has nothing to do with terrains size. The size is determined by the terrains bounding box (which can be scaled like every other brush with *The Selection Tool*). The resolution determines the number of height values that build up the terrains surface. A higher value usually means that a terrain surface is smoother and more detailed.
- Import: You can import the terrains height data from various file formats including greyscale images in BMP, JPG, PNG and PGM format as well as Terragen files.

- Export: Exporting the terrain into a greyscale image (same formats as stated above) or Terragen file is possible as well.

Terrain Generation



If you have created a new empty terrain and you haven't got any height data at hand to create its surface, it is often desirable to automatically generate a basic terrain.

The terrain generation dialog allows for this. By specifying a set of input parameters it lets you create different base terrains and shows the result in real time in a preview window.

- Frequency: A high frequency results in a bumpy surface with many hills and valleys whereas a low frequency will create fewer broader hills and valleys.
- Lacunarity: A multiplier that determines how quickly the frequency increases for each successive octave in the Perlin-noise function. The frequency of each successive octave is equal to the product of the previous octave's frequency and the lacunarity value.
- Octaves: A higher value adds structural detail to the terrain to the terrain so its surface becomes more natural.
- Persistence: A high value increases the surface contrast which results in steeper slopes between hills and valleys.

- Seed: Determines the random arrangement of hills and valleys for a terrain. You can play with this value to get different results that have all the same characteristics defined by the other parameters above. Note that the resulting terrain from a seed will always be the same (modified by the other parameters of course), so you can always recreate the same terrain again by using the same seed.

You can also find very good and more technical descriptions of these parameters in the libnoise glossary.

8.10.5 The Terrain Editor Sub Tools

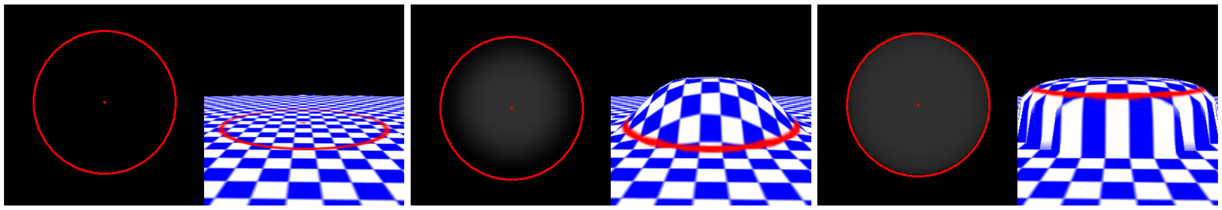
Since modifying terrains isn't a matter that can be handled by a single tool, the terrain editor consists of several sub tools that modify the terrain in their own way.

Tip: While working on your terrain you can undo/redo each step you made by either using *The Edit Menu* or the undo/redo shortcuts `CTRL-Y` and `CTRL-Z`. This allows you to easily try something with the terrain and undo it if the results are not what you expected.

The Raise Tool

This tool allows you to raise the terrain inside the tools area of effect. A high hardness will result in hard edges along the tool circles border and the tool effect determines the amount of world units by which the terrain is raised.

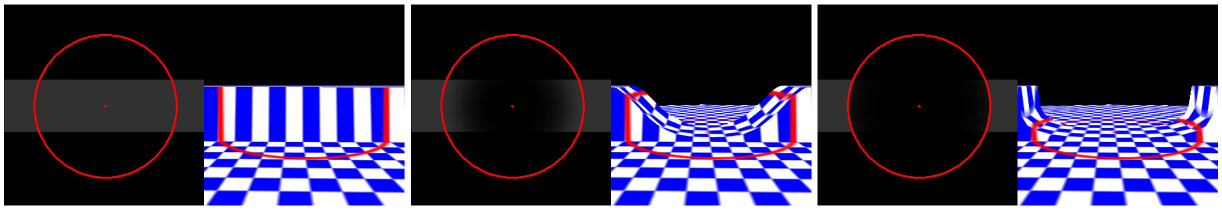
The following images show the usage of the raise tool with a low and high hardness:



The Lower Tool

This tools lowers the terrain inside the tool circle. A high hardness results in hard edges along the tool border and the tool effect determines how much the terrain is lowered.

Note that you can easily switch between the raise and lower tool using the middle mouse button. The following images show the usage of the lower tool with a low and high hardness:



The Flatten Tool

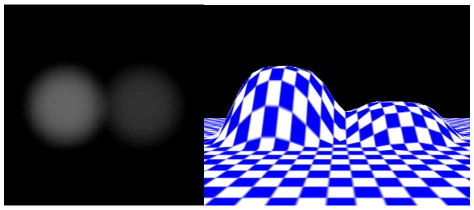
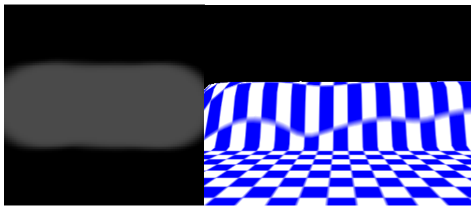
The flatten tool like its name says flattens the terrain in its area of effect. The tool effect has no meaning for it and is therefore not accessible. The hardness works as with every other tool.

The special thing about the flatten tool (and also the following two tools) is that it needs a reference height value. This is the height everything is flattened to.

To get a reference value you have to pick up one from the existing terrain elevations. This is either done by activating the Eyedropper mode by holding down the `ALT` key and then clicking on the position from which you want to pick up the height value (this works regardless of the activated sub tool).

Another way to pick up a height value is to use the middle mouse button (this only works for the flatten, fill and ablate sub tool).

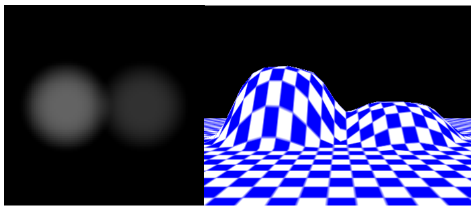
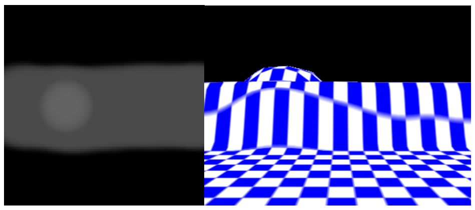
The following screenshots show usage of the flatten tool with a reference height between the two hills visible in the screens. As you can see everything is flattened to the same height.

The Fill Tool

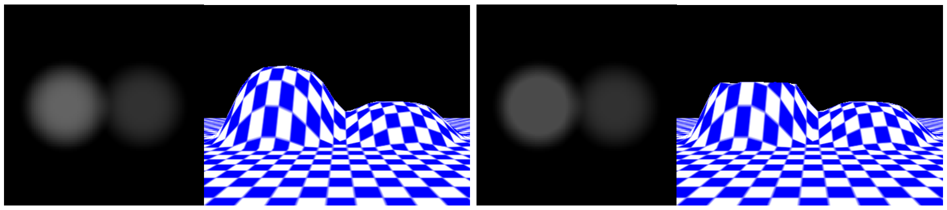
Works in the same way as the flatten tool with the exception that only those parts of the terrain are flattened that lie below the reference height value. This allows you to easily fill up valleys while leaving mountains as they are.

Using the fill tool with the same reference height as in the example above results in the flattening of all parts of the terrain except the top of the first hill.

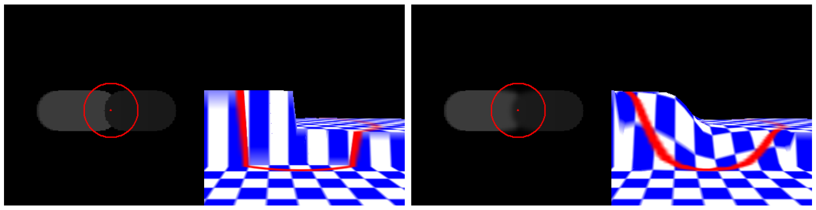
The Ablate Tool

Works in the same way as the flatten tool with the exception that only those parts of the terrain are flattened that lie above the reference height value. This allows you to easily ablate mountains while leaving valleys as they are.
As shown below the ablate tool only cuts the top of the first hill and leaves everything else as it was.



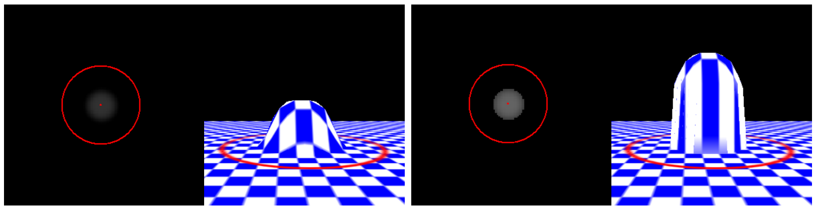
The Blur Tool

The blur tool will soften the terrain by reducing the elevation difference between the height values in the area of effect. The result is a smoother less edged terrain surface.
The tool effect determines how much the terrain surface is blurred/smoothened.



The Sharpen Tool

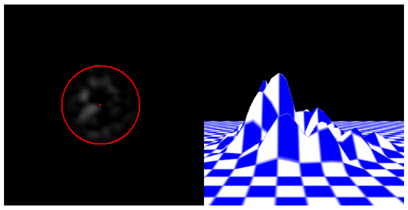
This is the contrary to the blur tool. It sharpens the edges of a terrain surface by increasing the elevation difference between height values and thus creating a more rocky/edgier terrain surface.
Again the tool effect determines how intensive the sharpening is.



The Noise Tool

The noise tool raises and lowers parts of the terrain in the area of effect at random. This makes the terrain in this area bumpier and adds structure to a flat surface.

The tool effect determines the maximal height difference between the current elevation and the randomly chosen one. A high tool effect makes the terrain more bumpy a low tool effect less.



The Road Tool

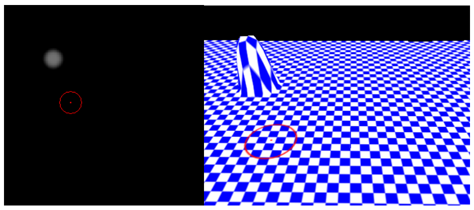
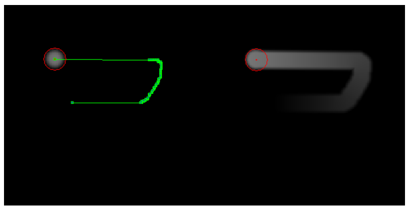
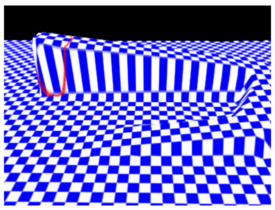
Unless you want to leave your terrains completely natural realistic roads are an important part of them. A road is an even plane that runs along a specific path through the terrain and might also have a slope between its starting and end point.

The road tool lets you specify the path that builds up the road and automatically creates a slope between the elevation at the roads beginning and end.

To create the roads path you have to click onto different points inside your terrain that make up the final road path. You can also hold down your left mouse button and continuously create road points while moving the mouse to create smooth curves.

To finally create the road, you have to press the `RETURN` key and the road is constructed. You can also undo road points by clicking the middle mouse button or pressing the `BACK` key.

The hardness determines if the road has a sharp edge to the surrounding terrain or smoothly blends into the terrain.

8.10.6 Creating textures for a terrain

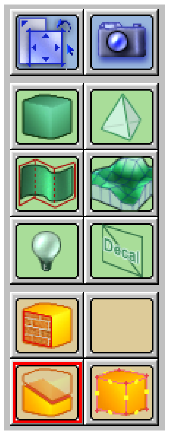
At this point you can't create terrain textures using CaWE and have therefore to rely on other external tools. Assigning a texture is done using *The Edit Surface Properties Tool*.

To create a texture you have to export your terrains height data and import it in an external program that allows you to create a surface texture for your terrain.
Possible solutions are Terragen and World Machine.

8.10.7 Keyboard shortcuts

- `SHIFT+F`:
  - Activates the Terrain Editor tool.
- `ALT`:
  - Activates the Eyedropper mode that lets the user pick up a reference height value.
- `ENTER`:
  - If the Road sub tool is active a road is constructed from the available road reference points.
- `BACK`:
  - Removes the last set road reference point.

8.11 The Clipping Tool

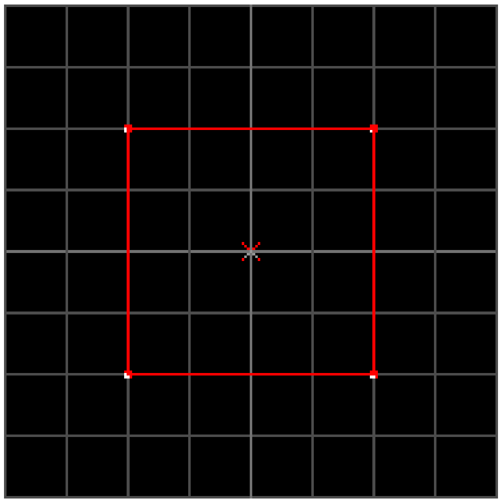


The clipping tool allows you to cut brushes into 2 pieces along a defined clipping plane. You can keep either one of the pieces or both after cutting them.

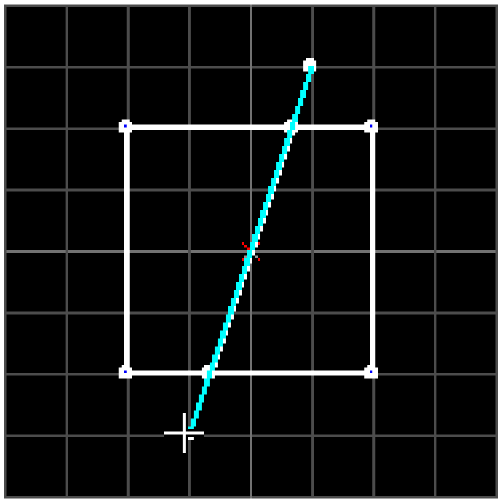
Although you could use *The Edit Vertices (Morph) Tool* to shape a brush, it is sometimes easier to cut off pieces from a larger brush to get the fitting result.

To activate the Clipping tool, click on the related icon in the tool bar or use `SHIFT+X`.

8.11.1 Using the Clipping tool

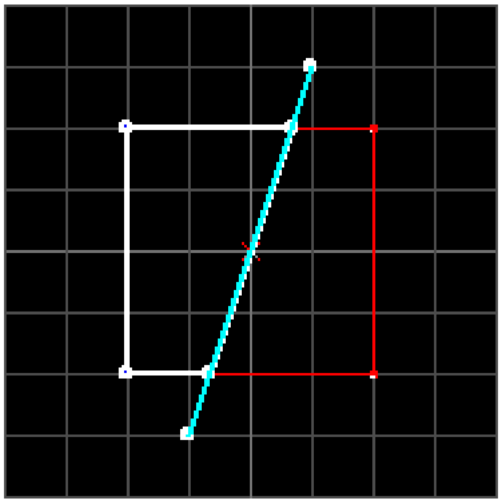
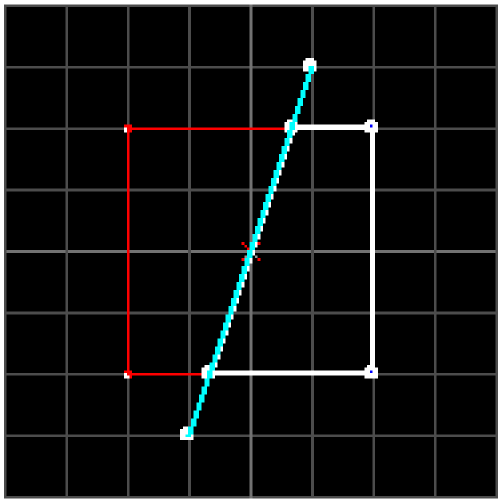


First you need to have an object already selected by *The Selection Tool*.

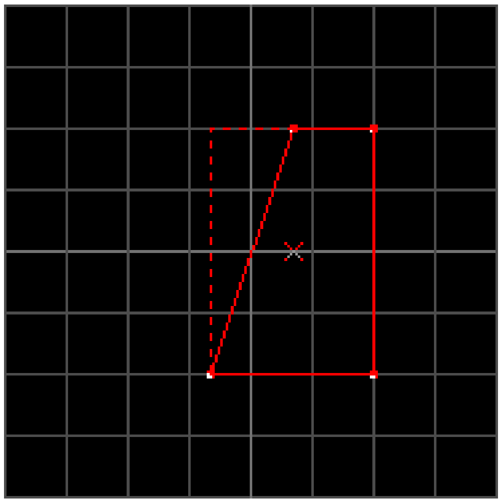


Now you can drag a line between to points in a 2D view to define a clipping plane. The cyan colored line is your clipping plane and shows where the brush is cut. Thick white lines show the parts of the brush that will be kept after the cut. If the whole brush is painted in white lines, the brush will be cut, but both pieces will be kept.

Note that you can't temporarily deactivate the grid snap setting by pressing `ALT` here, so you have to deactivate it permanently by pressing `SHIFT+W` if you want to make finer cuts. You can activate it again by pressing `SHIFT+W` after your finished.

You can switch between the pieces to keep by clicking on the Clip tool icon in the tool bar or by pressing `SHIFT+X`.

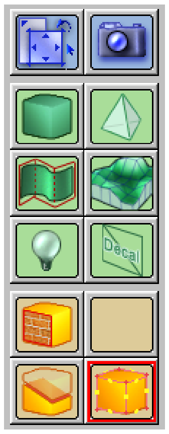


To do the cut you have to press `ENTER` and only the chosen pieces remain.

8.11.2 Clipping tool keyboard shortcuts

- `SHIFT+X`
  - Activate Clipping tool.
  - If activated → cycle trough clipping modes.
- `ESC`
  - Switch back to Selection tool.

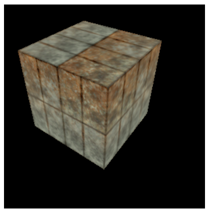
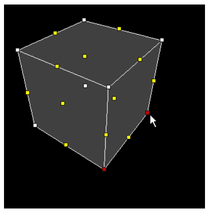
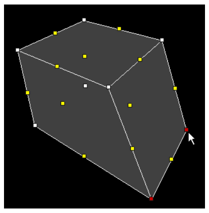
8.12 The Edit Vertices (Morph) Tool


The "Edit Vertices" tool (also called the "Morph" tool) allows you to edit the individual vertices of a brush and the control vertices of a bezier patch. (Please refer to *The New Brush Tool* and *The New Bezier Patch Tool* in order to learn more about how brushes and bezier patches are created.)

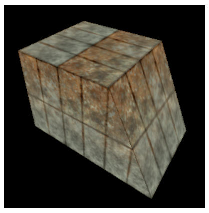
The vertices are the corner points of brushes. Being able to spatially manipulate them indiviually allows you to shape brushes in arbitrary ways that are difficult if not impossible to achieve with the other tools. Also the shape of bezier patches is controlled by special vertices that are called the *control vertices* of the patch. Modifying these control vertices indirectly modifies the shape of the bezier patch, so that they can be brought into every possible form.

8.12.1 Overview

Here is an overview of how the Edit Vertices tool is used:
(Click each of the images below to see them in their larger, natural size.)

1.  Select the object (brush or bezier patch) whose vertices you want to edit with the *Selection tool* first. You can also select multiple objects for vertex editing at the same time, but it's not recommended.
2.  Activate the Edit Vertices (Morph) tool by clicking on the related button on the Tools toolbar or by pressing the `SHIFT+V` shortcut key.
3.  Use the mouse to select or drag the vertex or edge handles. The vertex handles are shown as white squares, the edge handles as yellow squares. When you left-click to select individual handles, note

that you can use the `CTRL` key in order to "toggle" the selection. (This works very much as with files in the Windows Explorer.) Selection and dragging works both in the 2D as well as the 3D views.

4.  When done, just leave the Morph tool e.g. by switching back to the *Selection tool*. CaWE will then render the resulting brush as usual.

8.12.2 The Convex-Hull Principle

For the purpose of editing brushes, the CaWE Edit Vertices tool is implemented according to the "convex-hull principle". Roughly said, a convex-hull is the smallest space that contains a set of vertices. For a more thorough but mathematical definition please see the Wikipedia article here.

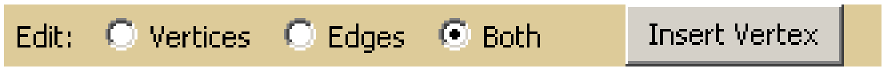
However, most CaWE users are already familiar with convex-hulls from another aspect of mapping, because they occur in a broader context: brushes! Each and every (valid) brush that you create with the *New Brush tool* is a convex hull of its vertices, and it stays that way no matter how you scale, rotate, shear, clip or morph it later.

Therefore, the morph tool operates by switching back and forth between the normal representation of a brush as you are used to, and it's convex hull representation. When you activate the Morph tool, the selected brush is first converted into its convex hull representation. Then, when you drag its vertices, you are actually dragging the vertices of the convex hull, and when you're done, the convex hull is just converted back into the normal brush representation. This works so well because brushes are per definition just another representation of the vertices of a convex hull – switching back and forth is mathematically a straightforward operation.

As a consequence, while it is impossible to use any of the other CaWE tools to turn a valid brush into an "invalid" brush (whatever that is), this statement also holds for the CaWE Morph tool! That is, it is quasi *impossible* to turn a valid brush into an invalid brush by dragging a vertex or edge handle*1)*. This property makes CaWE different from map editors of other engines (e.g. the Valve Hammer Editor 4.0), where it is *really easy* to accidentally turn a valid into an invalid brush (which means lost work) with the Morph tool.

In summary, the convex-hull principle that is implemented in the CaWE Morph tool is a very powerful concept that makes vertex editing very simple and intuitive, and at the same time *guarantees* that you cannot accidentally make an invalid brush. Just try the CaWE Morph tool out for a while – and you won't go back to other map editors!

8.12.3 The Tool Options Bar



When you activate the tool, the tool options bar shows controls that are specific to the Morph tool:

At the `Edit` radio buttons you can choose whether you would like to edit only the vertices (represented by white handles), only the edges (yellow handles), or both. Usually it's the easiest to leave the setting at both, but if you have a preference you can set it here. Note that only brushes have edges, the concept does not apply to bezier patches.

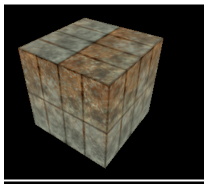
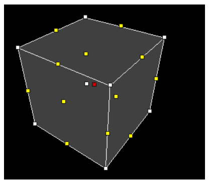
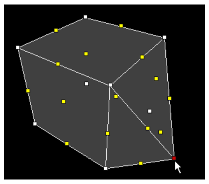
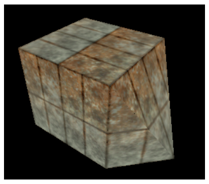
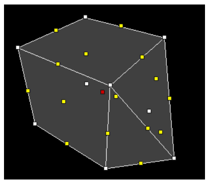
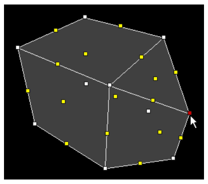
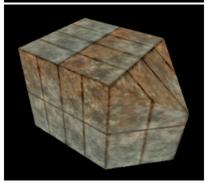
The `Insert Vertices` button inserts a new vertex into the center of a selected brush. It does only work if you have exactly one brush selected. That is, if you have instead a bezier patch patch selected or more than one brush or even a mix of multiple objects, it will just do nothing except for presenting you a message box that explains the exactly-one-brush requirement.

The inserted vertex will initially be located in the center of the brush. Thus, according to the convex-hull principle, it will be "loose" and initially *not* affect the shape of the brush. However, when you start to drag this vertex so that it becomes relevant for the convex hull, CaWE will automatically create the new shape of the brush according to the convex-hull principle.

The next section has a series of images that demonstrate how you use the `Insert Vertices` button at the example of a cube that gets two vertices added to form a more complex brush.

8.12.4 Face Splitting and Removal

Adding new faces to or removing existing faces from brushes naturally follows from the convex-hull principle. Please refer to the following image series in order to learn how new faces are added to a simple brush by first clicking the `Insert Vertex` button and then dragging the newly inserted vertex. In the second and fifth image, the new vertices that were inserted by a click on the `Insert Vertex` button have been selected (red squares) for better visual accentuation. The respectively next images (the third and sixth) show the result of these vertices being dragged into new positions.
(Click each of the images to see them in their larger, natural size.)

Faces are removed again my dragging one or more of their related vertices back “into” (somewhere near the center) of the brush or just onto other vertices. According to the convex hull principle, they then cannot contribute to the convex hull and thus the brush any more – the face has been removed. (Just read the above image sequence in reverse to see how this works.)

Note that for brushes, the Morph tool automatically eliminates duplicate vertices. That is, if you drag one vertex exactly onto another so that you cannot tell them apart any more, CaWE will just remove the duplicate vertex. This is also why pressing the `Insert Vertices` button does apparently nothing when it is pressed several times in a row: It just creates vertices all in the same place (the exact center of the brush), which are immediately collected again because they are duplicates of each other.

For bezier patches, being able to have multiple individual vertices all in the same place is sometimes a crucial feature, so the removal of duplicates does not apply.

8.12.5 Shaping Bezier Patches

When you use the Morph tool to edit the control vertices of bezier patches, the concepts of edges, convex hulls, vertex insertions and duplicate removals do not apply, but otherwise the tool behaves naturally and analogously to editing the corner vertices of brushes.

Selecting vertices individually

With bezier patches, it frequently occurs that multiple vertices are in the exact same position, overlaying each other. This is often intentionally and required to achieve certain spatial shapes. When you click on such vertices in the 2D views, they all get selected at the same time. This is usually what is wanted, but if instead you want to selected the vertices separately, just clear the selection and then click on the vertex in the *3D view*. Contrary to the 2D views, selecting vertices in the 3D view only ever selects *one vertex at a time*. Although the situation looks visually the same as before in the 2D views, you can now use the mouse to drag the single vertex in the 2D or 3D views without affecting the others in the same place.

1)

To be honest, there *is* a way to create invalid brushes with the CaWE Morph tool, namely when you create a zero-volume brush by dragging too many vertices "onto" each other (into the same spatial position). CaWE will then reconstruct the original brush when you try to exit the Morph tool in that situation.

CHAPTER 9

Selected Topics

9.1 Groups

Groups are a great aid for organizing your map and for making many mapping tasks easier. Map elements that have been put into a group can be treated as a whole, and thus they can be edited, transformed and modified as if they were a single object. Groups can also hide their elements from the views and lock them in order to prevent them from being edited.

In the latest versions of CaWE, groups are also persistent: Groups are saved in and loaded from Cafu map files so that they are available from one session to the next. Moreover, any changes to groups and their properties are recorded in the edit history and can be undone via the **Edit** → **Undo** menu.

In summary, groups act like layers in image processing programs. They are helpful and easy to use. This section explains the details.

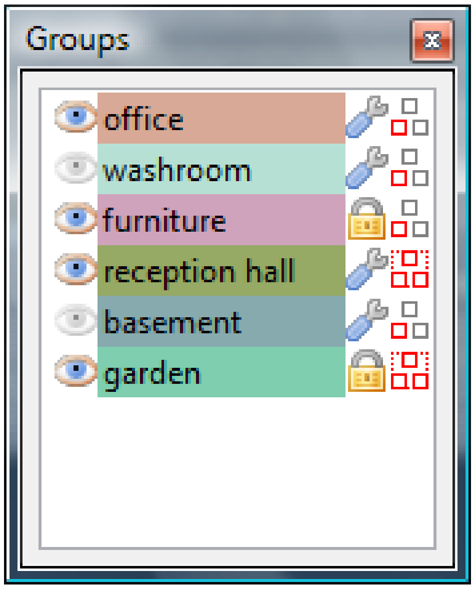
9.1.1 Creating Groups

In order to create a group, use the *Selection tool* in order to select the map elements that you want to become the members of the new group. Then use one of the following methods in order to put the selected elements into a new group:

- Use the **Tools** → **Group (Ctrl+G)** menu.
- Click one of the buttons **Group**, **Hide**, or **Hide Other** in the *Selection tool* toolbar.

Each of these menu items or buttons creates a new group and puts the selected map elements into it. The **Hide Other** button is an exception, as it will group everything else. The two **Hide** buttons additionally mark the newly created group as hidden, so that the selected map objects will disappear from the views.

9.1.2 The Groups Panel



The Groups panel shows a list of all groups in the map. If the Groups panel is currently not visible, use the **View** → **Panels** → **Groups** menu to show it.

Each group is shown in a line with information about the groups settings: its visible/hidden status, its name, its color, its edit/locked status, and whether the group members are selected individually or as a whole. Each of the icons can be clicked in order to toggle its status.

Group Visibility

The members of a group can be hidden from the views:

the group is visible.

the group is hidden.

This is very useful for getting things "out of the way" when you're editing a portion of the map that is usually occluded by other map objects. It's also a nice trick to enhance the performance on older computer systems: When the rendering is too slow, just hide the parts of the map that you're currently not editing, and the rendering of the remaining objects of interest will be much smoother.

Group Name and Color

The initial name of a group is automatically assigned when the group is first created, and can be changed at any time: Press **F2**, single-click a selected group, or use the context menu (described below) in order to assign a new name, e.g. one that describes the content or purpose of the group.

When you double-click the group name, the members of the group are selected in the map.

The groups color is indicated by the background color of the group name. When activated in the *CaWE Options* dialog, the member elements of a group are rendered in the groups color, so that they are easy to see in the map views.

Locking Groups

The members of a group can be prevented from being clicked in the map views:

the members of the group can be selected.

the group is locked.

This icon toggles whether groups can be selected with the *Selection tool*. When the icon shows the lock, the members of the groups cannot be selected in the 2D or 3D views, and thus they cannot become the subject of inadvertent edits, transformations, or other modifications.

Group Selection

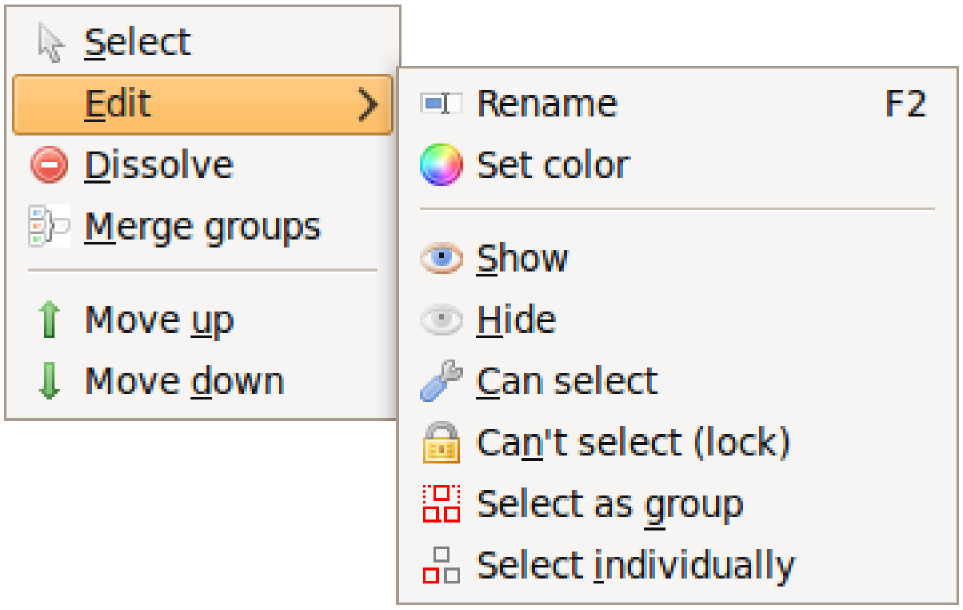
The grouping nature of a group can be turned on and off:

the members of the group are selected individually.

the members of the group are selected as a whole.

This icon toggles the actual grouping behavior of the group. Usually, you will want to have this set to “select as a whole”, so that a mouse click on a part of a group selects the whole group. However, it can be preferable to have this set to “select individually”, for example when the primary purpose of the group is to temporarily hide or lock its elements.

9.1.3 Editing groups



Groups can be edited via the Groups panels context menu: Right-click into the Group panel in order to open its context menu:

Select Selects all members of the group in the map, just as if they had been selected with the *Selection tool*. This is the same as a double-click on the group name.

Edit Using the submenu, you can change the properties of one or more groups that are selected in the Groups panel.

Dissolve This menu item ungroups the members of the group and then deletes the group. Note that it does not delete the map elements itself, they’re just ungrouped.

Merge groups When two or more groups are selected, their members can be merged into a single group.

Move up/down Use these menu items to change the order of the groups in the list.

It is important to note that *multiple* groups can be edited via the context menu: Just select one or more groups, and the context menu operation will affect them all at once.

Any changes to groups and their properties are recorded in the edit history and can be undone via the **Edit** → **Undo** menu.

9.2 Dealing with Leaks

This section explains what a leak is, why leaks are important, how they are found and fixed, and how leaks are prevented.

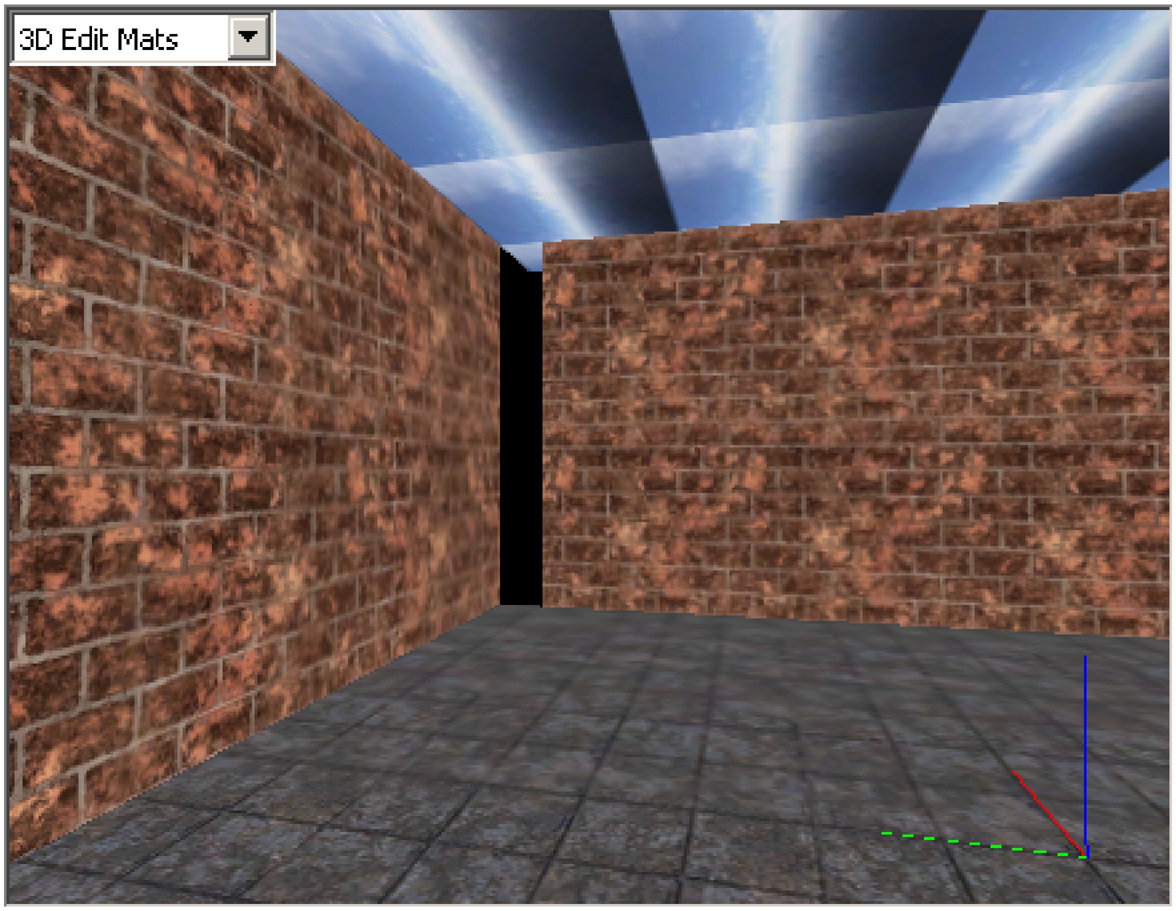
9.2.1 What is a Leak?

Maps created for the Cafu engine are preprocessed by several compile tools (e.g. CaBSP, CaPVS, CaLight) for optimal performance. These compile tools assume that any map has an inside (where the player and monsters are), and an outside that nobody except for the mapper will ever see. The inside must be *sealed "air-tight" or "water-tight"* from the outside.

That is, if you imagine you flood-filled the map from the inside with water or pressurized air, at no point must the water or air be able to escape to the outside. Therefore, each place where that would be possible is called a *leak*.

"A leak is a hole in the world, where the inside of it is exposed to the (unwanted) outside region." – CaBSP error message when a leak was found.

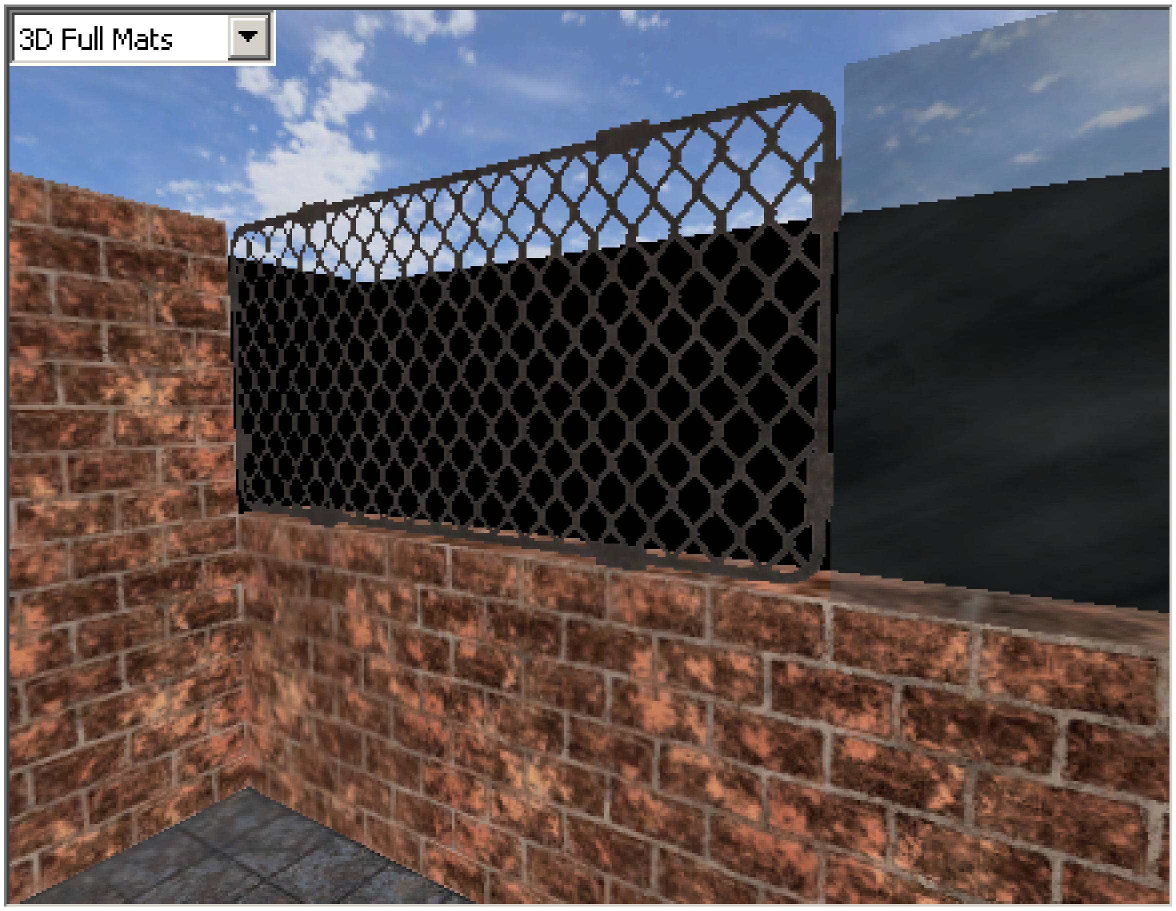
9.2.2 Examples for Leaks



Gaps in the geometry

The example to the right shows a leak that is caused by an obvious gap in the geometry that leads to the void (the outer region). This is the classical example and most common type of leaks, and such leaks are usually easy to find and fix.

However, note that gaps like these can be very small, and often they're somewhere inside complex geometry. Don't hesitate to zoom close whenever the situation is not as obvious as in this example.



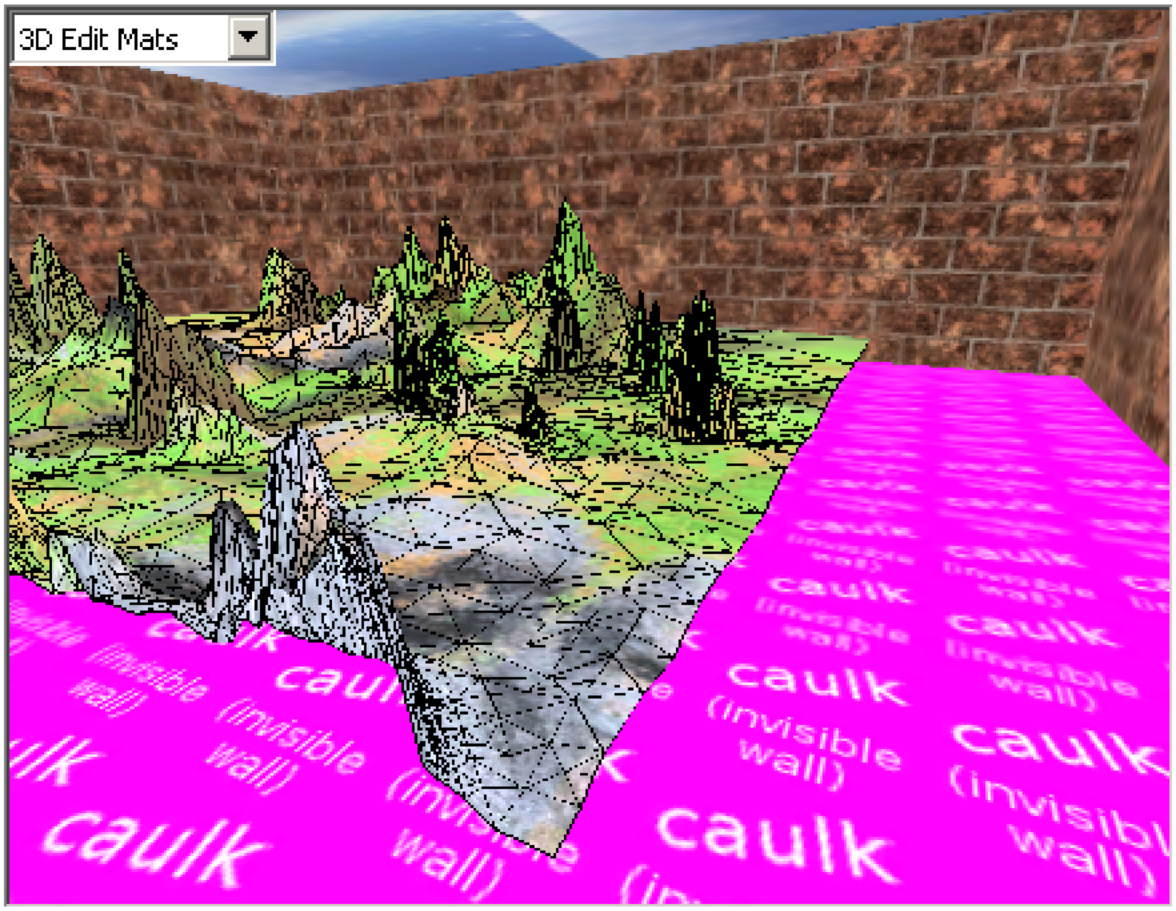
Inappropriate materials on outside walls

Materials that are see-through (like the grate in the center of the image) or translucent (like the glass pane to the right) cannot be used on outside walls as is this example, because such materials obviously don’t seal the map.

To fix the problem, you either have to replace such materials with other materials that are solid, or make another room that has solid walls on the other side of (“behind”) these surfaces.

Note that contrary to the materials shown in this example, *skydome* materials **do** seal the map.

(?) How you do know which materials seal the map and which don’t?
→ Only those materials that have the `bspPortals` *clip flag* set in their material definition seal the map, all others don’t. Future versions of CaWE will directly indicate the status of this flag in the Material Browser, but currently you have to look-up the definition of a material in its `.cmat` file to find out the status of this flag. Good news though is that most materials behave totally naturally:
(!) *As a rule of thumb, if you can see through, then CaBSP can see through, too, and thus such materials cannot be used on outside walls.*



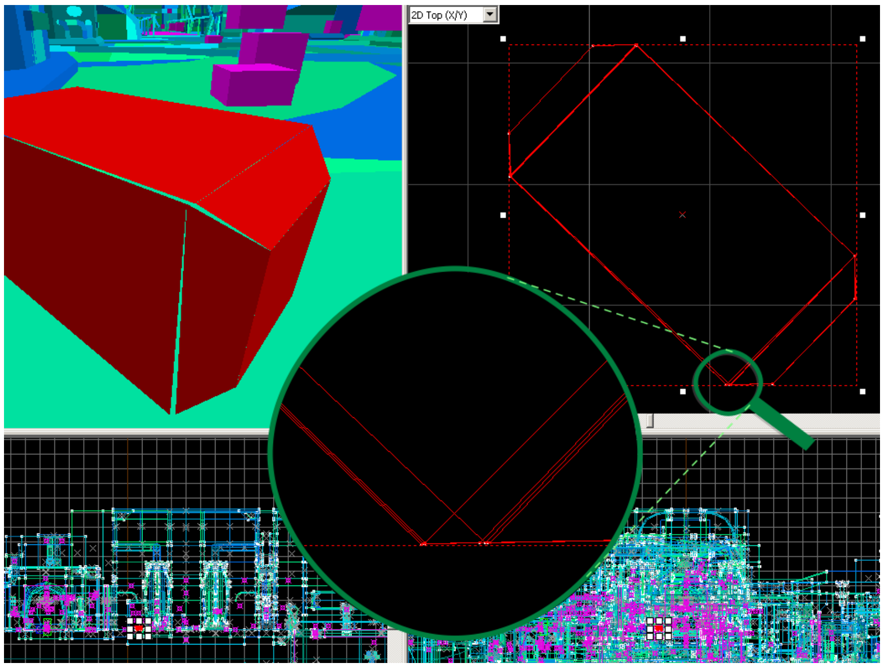
Terrains, Bezier Patches and Entities

Terrain and Bezier Patch primites *never* seal the map, even if their material is solid. Therefore, if you dragged the terrain in this example to cover the entire floor, you still cannot expect it to prevent the map from leaking.

The same is true for *brush entities*, i.e. entities that are made from brushes. For example, if you make a `func_wall` or `func_door` entity and place it so that it appears to contribute to the outside wall, CaBSP will still report a leak, because only the global "world" brushes are taken into account for the sealing hull.

As shown in the image, the proper solution is to use the `Textures/meta/caulk` and other materials like walls and skies to make a "containing room" around such primitives and entities. The `Textures/meta/caulk` material is especially useful in such cases because it seals the map but doesn't render (is invisible) in the engine later.

The *New Terrain tool* assists you with constructing new terrains that have a proper containing room right from the start, and you may want to have a look at the `TechDemo.cmap` for a properly sealed terrain example.



Leaks due to bad brushes

Sometimes leaks can also be caused by bad brushes like the one shown in this example (click image to enlarge). Such leaks are the most problematic: They are very hard to see visually, and they can get CaBSP thoroughly confused*1)*. Even worse, usually not much helps but deleting the offending brush, and make it anew with the *New Brush* and other tools.

On the other hand, the occurrence of such bad brushes is rare. They sometimes exist as the result of importing a map from another game or editor, but are hard to create accidently in CaWE. As mentioned before, the best solution is to re-make the offending brushes, or put them into a separate entity, because only the world brushes contribute to the sealing hull, whereas the entities are treated independently.

9.2.3 Effects of a Leak

If your map has a leak, the CaBSP compiler will automatically detect it and consequently abort the compilation with an error message like this (if there is more than one leak in the map, only the first leak will be reported):

```
*** Fill Inside ***                                    0: 0: 1

### LEAK DETECTED! ###

A leak is a hole in the world, where the inside of it is exposed to the
(unwanted) outside region. Thus, a leak pointfile has been generated.
Load this file into the world editor CaWE, and find the beginning of the line.
Hint: The beginning is always near one of the "info_player_start" entities.
Then find and fix the leak by tracing the line until you reach the outside.
(The line always takes the shortest path, so this should be easy.)

Notes:
- Leaks can be *very* small. Use a close-up view, if necessary.
```

(continues on next page)

(continued from previous page)

```
- Use the grid. The grid is useful to fix leaks + to avoid them from the start.
- Make sure that *all* "info_player_start" entities are inside the world!
- Be aware that both the clip hull and the draw hull must be sealed.
- Please refer to the documentation for additional information.

Pointfile written to Games\DeathMatch\Maps\LeakDemo.pts

FATAL ERROR: Stopped by leak.
Program aborted.
```

That means that with a leak in the map, you cannot run the map at all. As CaBSP doesn't write any output file in this case (except for the above mentioned pointfile), neither the subsequent compilers (CaPVS and CaLight), nor the Cafu engine itself can run the map.

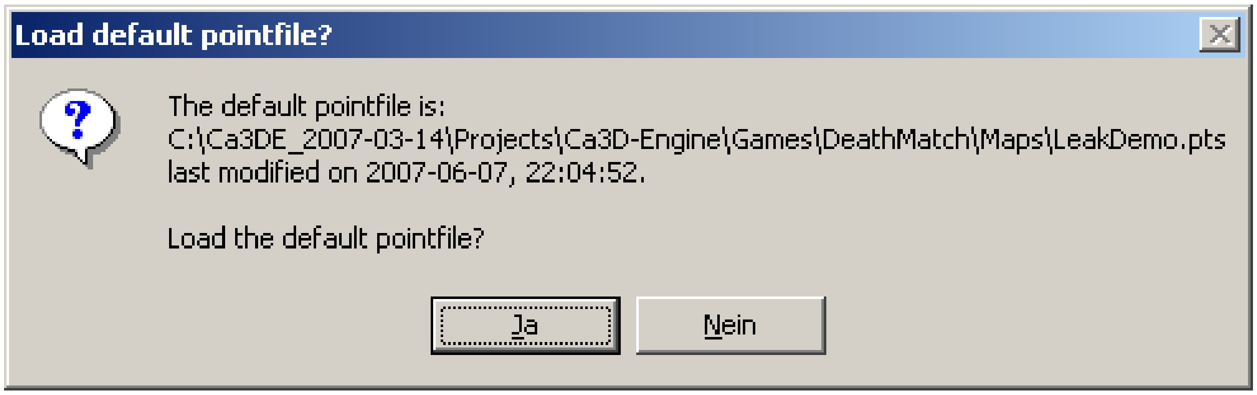
Future versions of CaBSP may reduce the severity level of the occurrence of a leak from an error to a mere warning. This will allow you to run your map with Cafu even if it has leaks, but nonetheless will leaks remain a problem that should be fixed. Leaks always indicate that the map could not be optimally processed, and thus imply higher polygon counts and lower performance.

The next section will tell you how to find the leaks reported by CaBSP in the map.

9.2.4 Locating Leaks in the Map

Finding leaks is very easy: When CaBSP reports that it found a leak (see example error message and description above), it also creates a corresponding *pointfile*. A pointfile is an auxiliary file that contains the description of a trail from one of the `info_player_start` entities to the outside. The leak is always located somewhere along that path.

For loading the pointfile into CaWE, simply open the relevant map in CaWE (**File** → **Open…**, if you haven't already), then select the **Map** → **Load Pointfile** menu item. CaWE assumes that you'll want to load the recently created pointfile for the currently open map, and thus asks for confirmation, e.g. like this:



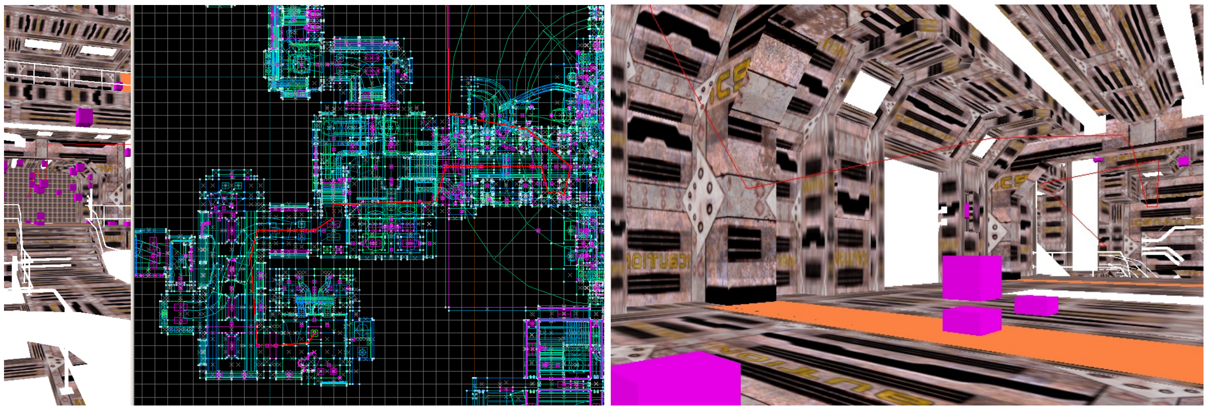
Load default pointfile?

The default pointfile is:
C:\Ca3DE_2007-03-14\Projects\Ca3D-Engine\Games\DeathMatch\Maps\LeakDemo.pts
last modified on 2007-06-07, 22:04:52.

Load the default pointfile?

Click "Yes" ("Ja") to open the suggested default pointfile, which is normally the desired action. Clicking "No" ("Nein") will open a file selection dialog where you can choose a different file or cancel.

Here are example screenshots of a pointfile that has been loaded into an (unfinished) map:


(Click on the images to enlarge them.)

Use the 2D views and especially the 3D camera view (see *2D and 3D Views* for details) to follow the trail from the starting point near one of the `info_player_start` entities to the place where the line leaves the inside of the map and escapes to the outside. Fix the so found leak however seems appropriate.

You may notice that loading the pointfile can decrease the rendering performance in CaWE, especially if the trail is very long. Therefore, once you've found the leak, you can unload the pointfile again by selecting the **Map** → **Unload Pointfile** menu item.

Finally, restart the compilation (CaBSP) to check if the map is now completely sealed or if there are more leaks.

9.2.5 Preventing Leaks

The best way to deal with leaks is to prevent them right from the start. Here is a list of suggestions and considerations in this regard:

1. Use the grid (**Map** → **Show grid**) and turn on grid snapping (**Map** → **Snap to grid**). The grid and grid snapping are your most powerful allies when it comes to see and avoid leaks while you're constructing your map.
2. Run the **Map** → **Check for Problems** menu item occassionally and before compiling your map.
3. Avoid all tools that potentially introduce rounding errors and thus misaligned brushes. As a gross guideline, all tools that operate on the grid and that are supposed to *produce results on the grid* are usually safe, such as resizing rectangular blocks, shearing, vertex manipulation, mirroring, clipping and carving. The following operations are sometimes less safe: Arbitrary (any angle) rotation and clipping and carving when the result has vertices that are off-grid. For less-safe operations it is often better to try to mimic the same effect with the safer operations. For example, when you want to carve an arched door into a wall, cutting and placing the wall brushes manually is often better than employing the (more convenient) **Tools** → **Carve** tool.

Note that leaks are **not (!)** prevented or fixed by putting the *entire map* into a big box, even if the strategy that has been suggested above for working properly with terrains might make you think so. Although putting everything into a big box will make CaBSP compile the map successfully and thus seemingly fix the leak, the problems associated with a leak (e.g. high polygon count and bad performance) will persist. This is not a solution.

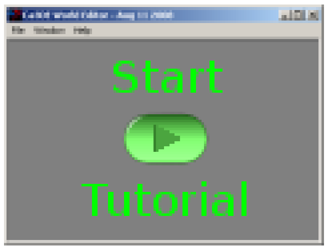
9.2.6 Conclusion

Being able to load the CaBSP-generated pointfiles directly into CaWE makes finding and fixing leaks relatively easy. However, one of the most important aspects about fixing leaks is to prevent them right from the start.

Working carefully and making sure that brushes are properly aligned (snapped) to the grid can help significantly with avoiding leaks even before they occur. The cleaner and more organized you build your geometry, the easier will it be for you to not build them accidently into your map in the first place, and when they still occur, the easier they are to locate and fix.

In general, it's also a good idea to occasionally test-compile your map while it is only partially complete and still a work-in-progress. Besides that this will help you to fine-tune your geometry, it will also point out leaks one at a time, which makes finding and fixing a lot simpler and faster than later when your map is large and fully detailed.

9.2.7 See Also



Flash Tutorial – A short flash tutorial that presents the essentials about dealing with leaks.

1)

If you ever experience such a case and cannot get it fixed by remaking the offending brushes, let us know. Please include the related map as an attachment.

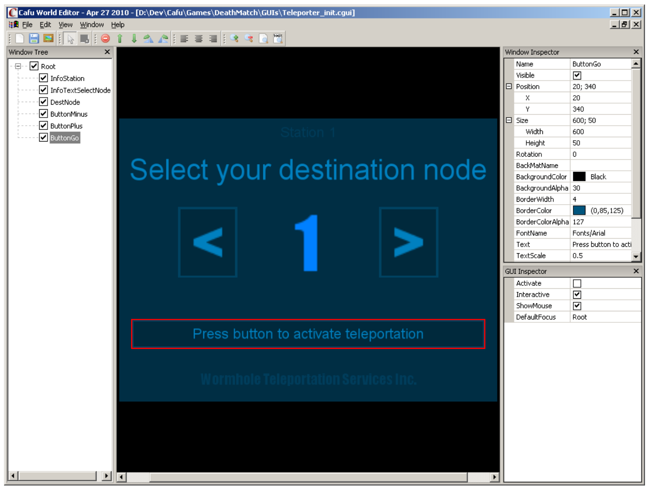
9.3 Creating Teleporter Stations

This tutorial shows how a network of teleporter stations can be added to a game map: A set of teleporter stations is distributed throughout the map. Each station has a graphical user interface that the player can use to select the station that he wants to teleport to, and to activate the teleportation.

The tutorial involves the placement of new entities into the map, GUI scripting and map scripting. As such, it is also a gentle introduction into Cafu game scripting.

All tasks are demonstrated using the sample map **BPWxBeta** that is included with Cafu.

9.3.1 Creating the teleporter controls GUI


Cafu World Editor - Apr 27 2010 - [D:\Dev\Cafu\Games\DeathMatch\GUIs\Teleporter_init.cgui]
File Edit View Window Help
Window Tree
Root
InfoStation
InfoTextSelectNode
DestNode
ButtonMinus
ButtonPlus
ButtonGo
Station 1
Select your destination node
<
1
>
Press button to activate teleportation
Wormhole Teleportation Services Inc.
Window Inspector
GUI Inspector

Name	ButtonGo
Visible	✓
Position	20; 340
X	20
Y	340
Size	600; 50
Width	600
Height	50
Rotation	0
BackMatName	
BackgroundColor	■ Black
BackgroundAlpha	30
BorderWidth	4
BorderColor	■ (0,85,125)
BorderColorAlpha	127
FontName	Fonts/Arial
Text	Press button to acti
TextScale	0.5

Activate	☐
Interactive	✓
ShowMouse	✓
DefaultFocus	Root

Each teleporter station is supposed to have a GUI that the player can use to control which other station he wants to teleport to, and to activate the teleportation.

We use the GUI editor component of CaWE in order to create the new GUI, an example is shown in the image.

The next section explains how the new GUI is scripted to properly react to player input: The < and > buttons should increment or decrement the number of the station that the player is being teleported to, and the big button below should actually trigger the teleportation.

9.3.2 Writing custom code for the teleporter controls GUI

When the GUI was saved in the GUI editor, it automatically wrote two files:

- `Teleporter_init.cgui` and
- `Teleporter_main.cgui`

The first file contains the actual data that the GUI editor loads and saves, the second file is for your hand-made customizations. In short, note that the first file is overwritten whenever the GUI editor saves a GUI file, whereas the second file is only created once as an empty stub file. The GUI editor touches the second file never again, so that you can use it for hand-written customizations.

We now edit file `Teleporter_main.cgui` and enter the following script code:

```
dofile("Games/DeathMatch/GUIs/Teleporter_init.cgui");


function ButtonMinus:OnMouseEnter()
    self:set("borderColor", 1.0, 0.0, 0.0, 1.0);
    self:interpolate("textScale", 2.2, 2.4, 500);
end

function ButtonMinus:OnMouseLeave()
    self:set("borderColor", 0, 0.333333, 0.490196, 0.5);
    self:interpolate("textScale", 2.4, 2.2, 500);
end

function ButtonMinus:OnMouseButtonUp()
```

(continues on next page)

(continued from previous page)

```
    local NodeNr=tonumber(DestNode:get("text"));

    if (NodeNr>1) then
        DestNode:set("text", NodeNr-1);
    end

    return true;
end


function ButtonPlus:OnMouseEnter()
    self:set("borderColor", 1.0, 0.0, 0.0, 1.0);
    self:interpolate("textScale", 2.2, 2.4, 500);
end

function ButtonPlus:OnMouseLeave()
    self:set("borderColor", 0, 0.333333, 0.490196, 0.5);
    self:interpolate("textScale", 2.4, 2.2, 500);
end

function ButtonPlus:OnMouseButtonUp()
    local NodeNr=tonumber(DestNode:get("text"));

    if (NodeNr<MAX_NODES) then
        DestNode:set("text", NodeNr+1);
    end

    return true;
end


function ButtonGo:OnMouseEnter()
    self:set("borderColor", 1.0, 0.0, 0.0, 1.0);
    self:interpolate("textScale", 0.5, 0.52, 500);
end

function ButtonGo:OnMouseLeave()
    self:set("borderColor", 0, 0.333333, 0.490196, 0.498039);
    self:interpolate("textScale", 0.52, 0.5, 500);
end

function ButtonGo:OnMouseButtonUp()
    local origNr=OUR_NODE_NR;
    local destNr=DestNode:get("text");

    game.runMapCmd("teleport(" .. origNr .. ", " .. destNr .. ");");
    return true;
end


-- This function is called as soon as the entity related to this GUI has been␣
↪initialized.
-- Note that our entities must have names like "teleporter_2_of_5" for this to work.
function OnEntityInit()
    -- Figure out the total size of the teleportation network (number of nodes)
    -- that this station is in, and which number this node/station has.
    OUR_NODE_NR, MAX_NODES=string.match(gui:getEntityName(), "(%d+)_of_(%d+)");
```

(continues on next page)

(continued from previous page)

```
    OUR_NODE_NR=tonumber(OUR_NODE_NR) or 1;
    MAX_NODES  =tonumber(MAX_NODES) or 3;

    InfoStation:set("text", "Station " .. OUR_NODE_NR);
    DestNode:set("text", (OUR_NODE_NR % MAX_NODES)+1);
end
```

The first line of the script loads and runs the previously mentioned first file, that initializes the basics of the GUI. As such, the first line is crucial because it is responsibe for "connecting" both files.

After the GUI script is fully loaded, the Cafu Engine initializes the GUI by

- first calling all `OnInit()` methods of all windows (they are defined in `Teleporter_init.cgui`),
- then calling all `OnInit2()` methods of each window (of which we have none at all, but if we had, they were in `Teleporter_main.cgui`),
- and finally calling function `OnEntityInit()` for 3D in-game GUIs such as ours.

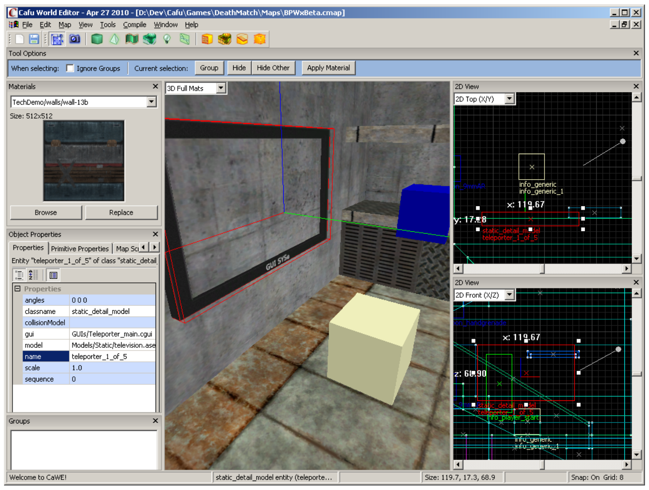
The `OnEntityInit()` method is very important as is determines the number of this teleporter station (or "node") and the total number of nodes, and saves both values in global variables for future use. It also updates the "Station XY" text of the `InfoStation` window that indicates to the player at which station he is currently standing, and sets a reasonable default for the destination station that we're teleporting to.

All other methods are event handlers that the Cafu engine calls whenever the related event occurs. The `OnMouseEnter()` and `OnMouseLeave()` methods are only used for adding some visual eye candy, and so we won't further discuss them.

The most interesting method is `ButtonGo:OnMouseButtonUp()`, that is called when the player presses the button that is supposed to activate the teleporter. However, note that this GUI script is like implementing the control software that runs on the teleporter station, just like a desktop program that would run on the station if the teleporter was real. As such, the GUI cannot implement the teleportation itself, but it must ask the map script to do it.

The GUI can use the `runMapCmd()` function of the `game` object in order to ask the game script to do something. We will later write a game script function `teleport(origNr, destNr)` that does precisely that: Teleport a player from station `origNr` to station `destNr`, and thus when the player presses the button, we have this function called here.

9.3.3 Placing teleporter stations in the game map



Cafu World Editor - Apr 27 2010 - [D:\Dev\Cafu\Games\DeathMatch\Maps\BPWxBeta.cmap]

TechDemo/walls/wall-13b

Size: 512x512

Entity "teleporter_1_of_5" of class "static_detail

Properties	
angles	0 0 0
classname	static_detail_model
collisionModel	
gui	GUIs/Teleporter_main.cgui
model	Models/Static/television.ase
name	teleporter_1_of_5
scale	1.0
sequence	0

Welcome to CaWE! | static_detail_model entity (teleporte... | Size: 119.7, 17.3, 68.9 | Snap: On Grid: 8

The next task is quite simple: We place the new teleporters into the map. To do so, use the *New Entity* tool in order to create two new entities whereever you want to have a teleporter station:

- One `static_detail_model` that shows the big screen with the controls GUI, and
- one `info_generic` entity that indicates the source and destination point of the teleportation.

Note that you *must* name the `static_detail_model` with a name of the form `somename_X_of_Y`, where X is the number of this station and Y is the total number of the stations. This is because our GUI script above has been written to figure out these numbers from the name of the entity.

9.3.4 Implementing teleport() in the map script

The final task is to write the above mentioned `teleport()` function in the map script: For the **BPWxBeta** example map, we create or open `Games/DeathMatch/Worlds/BPWxBeta.lua`, and enter the following code:

```
-- It's more readable to call wait(x) rather than coroutine.yield(x).
wait=coroutine.yield;

-- This function reloads and runs this script again.
-- Useful for working with and testing the script "online",
-- i.e. while the engine is running and without reloading the map!
function reloadscript()
    dofile("Games/DeathMatch/Worlds/BPWxBeta.lua");
end


-- This function is called by the teleporter GUI script when the user wants to␣
↪teleport.
-- origNr and destNr are numbers like 1 or 5 that indicate from and to which node in␣
↪the
-- teleportation network the teleportation should occur.
--
-- TODO/FIXME: It would be nice if we could learn from an additional parameter which
--    (player) entity was actually operating the teleporter controls (i.e. who pressed
--    the "Go" button on the related GUI).
function teleport(origNr, destNr)
    -- Prevent re-entrancy in the case that this method is called very quickly in␣
↪succession.
```

(continues on next page)

(continued from previous page)

```
    -- This can happen with the current EntHumanPlayerT code, which is sort of a bug.
    if (isTeleporting) then return true end

    local ox, oy, oz=_G["info_generic_" .. origNr]:GetOrigin();
    local dx, dy, dz=_G["info_generic_" .. destNr]:GetOrigin();

    Console.Print("Teleporting from node " .. origNr .. " to node " .. destNr .. ",
↪");
    Console.Print("which is at coordinate (" .. dx .. ", " .. dy .. ", " .. dz .. ").
↪\n");

    -- Teleport all entities with name "Player*" to the destination node.
    for PlayerNr=1, 99 do
        local PlayerEnt=_G["Player" .. PlayerNr];

        if not PlayerEnt then break end

        local px, py, pz=PlayerEnt:GetOrigin();

        local ax=px-ox;
        local ay=py-oy;
        local az=pz-oz;

        if (pz>oz and math.sqrt(ax*ax + ay*ay + az*az) < 2000.0) then
            -- VARIANT 1:
            -- When teleporting PlayerEnt to (dx+ax, dy+ay, dz+az), it is safe to␣
↪teleport
            -- multiple players all at the same time (continuing the loop), as their
            -- relative positioning doesn't change.
            -- However, depending on the map geometry, teleporting to (dx+ax, dy+ay,␣
↪dz+az)
            -- could teleport the player into solid matter, as it requires that at␣
↪the dest
            -- there is as much free space (in the same absolute direction) as at the␣
↪origin.
            -- PlayerEnt:SetOrigin(dx+ax, dy+ay, dz+az);

            -- VARIANT 2:
            -- When teleporting PlayerEnt to (dx, dy, dz+az), the results are much␣
↪easier
            -- to foresee, but this requires teleporting only one player at a time.
            PlayerEnt:SetOrigin(dx, dy, dz+az);
            break;

            -- TODO/FIXME: We should check that the destination space is actually free
            --   from other players and entities before we relocate PlayerEnt there!
        end
    end

    isTeleporting=true;
    wait(0.5);
    isTeleporting=false;
end
```

9.4 About Lights and Shadows

(FIXME!) This page is still under construction **(!)**

One of the key questions that mappers frequently raise is how to define and place light sources. While placing light sources is generally easy, this section explains the types of available lighting techniques in Cafu and the options to create light sources for each.

The Cafu engine implements two different types of lighting techniques:

- Radiosity-based lighting and
- dynamic lighting.

The two lighting techniques are quite distinct from each other, and each comes with a set of feature of its own. Nonetheless, both techniques peacefully coexist in the Cafu engine and mappers frequently employ both in a single map.

9.4.1 Radiosity-based Lighting

One of the most fundamental lighting techniques in computer graphics is the Radiosity method. Radiosity methods compute lighting based on a physical model, and it is the nature of the underlying mathematics that characterizes the features of the technique.

The big downside of Radiosity is that it requires a preprocessing step. This preprocessing step can be computationally very expensive and thus take a long time to complete. It is implemented in a special tool called "CaLight", which is one of the map compilers that are covered in sections *Compiling Maps for Cafu* and *Compiling Maps at the Command-Line*.

The fact that Radiosity is slow to precompute also means that it cannot work with dynamic objects: Radiosity takes walls and any other static object into account, but anything that moves will not influence the Radiosity-based lighting result.

Good news about Radiosity-based lighting is that once the preprocessing step it complete, it is *fast*. Independently of the number of light sources or complexity of the scene, Radiosity-based lighting provides very good performance at run-time during the game. It is therefore a natural choice for large-scale base lighting of any level.

Even better, Radiosity lighting results looks stunningly good. No matter what hardware vendors tell you about their latest products, no matter what game developers tell you about dynamic lighting (as I do, too; see below), none of these techniques can beat the realism of lighting generated by Radiosity methods. Based on an accurate physical model, the natural propagation and appearance of lights and shadows is unique to Radiosity methods.

The Radiosity method also handles *area light sources* without special measures, but as a natural part of the algorithm. Area light sources are an important key feature, because they are the main contributors to a realistic lights and shadows distribution with diffuse reflections and soft shadows. This is contrary to dynamic lighting, where lights are only points in space, soft shadows are always the result of some kind of trick, diffuse reflections are not possible at all, and a physical model is never involved.

For these reasons, Radiosity was the first lighting technique implemented in Cafu.

Here is an overview of the available Radiosity light sources and how they are created in a map:

Area Light Sources

Any primitive (brush or Bezier patch) surface that you place in a Cafu map can be turned into an *area light source*. The key point here is that a surfaces characteristic of being an area light source is *not* **(!)** directly set in the Cafu World Editor CaWE. Rather, being an area light source is a *material* property. That means that a surface becomes an area light source when it has a material applied to it which in turn is defined to emit Radiosity light.

- In order to place an area light source, use the *Edit Surfaces* tool to apply an appropriately defined material to the desired brush or Bezier patch surface. You can learn which materials cast Radiosity area light by clicking on them once in the *Material Browser*. The bar at the bottom will show “Radiant Exitance” information on the selected material, if any. Tip: Filter for materials with the string “light” in their name. This heightens the chance to find a relevant material quickly.
- In order to change the definition of a new or existing material to emit light, make sure that the `meta_radiantExitance` keyword is employed. Please refer to the Material Systems *Keyword Reference* for more details. Note that the effect of changing the definition of an *existing* material is *global*. That is, if the material is used elsewhere (in another map), also the other map will be affected when CaLight is run next.

Point light sources

You can also place *point light sources* to participate in the Radiosity lighting process. This is easily achieved with the *New Light tool*, whose documentation explains the placement and parameter setup in detail.

Sky Light

Adding sun- or moonlight to a map works analogously to adding an area light source: Just apply a “sky” material to any surface, and if the material is defined to emit skylight, everything else will happen automatically:

- The *Material Browser* browser informs you whenever a material with “Irradiance” values is selected.
- The material definition can be changed according to the Material Systems *Keyword Reference*, see the `meta_sunlight` keyword.
- You may refer to the *Your First Map* tutorial to see an example. It’s actually that easy!

9.4.2 Dynamic Lighting

Dynamic lighting has become popular with the advent of hardware-acceleration for 3D graphics in the recent years. It is the preferred and most frequently employed lighting technique in most recent commercial graphics software.

The essential characteristic of dynamic lighting is that its computational steps occur *per-pixel* and *per-frame*. *Per-pixel* means that the lighting computations occur on the basis of the output pixels of the rendered primitives (triangles). *Per-frame* refers to the fact that all lighting computations occur only at runtime, and in fact, they are performed for each rendered frame anew. This allows both light sources and lit objects to move arbitrarily through the scene, with the lights and shadows being updated correctly and instantaneously.

These properties have many implications for the practical use of dynamic lighting: Dynamic lighting is fast, flexible, dynamic, instantaneous, yields good results and is well supported and accelerated in hardware.

Even though dynamic lighting looks not quite as real as the Radiosity method, this is often not a problem, because lighting that looks dramatically good is often preferred over the “boring natural”. Moreover, dynamic lighting avoids the computational complexity that comes with the Radiosity method. Therefore, dynamic lighting is an important component in Cafu.

Point light sources

Projective lights

Stencil shadows

Shadow-Map shadows

9.4.3 Spherical Harmonics Lighting

Lighting with Spherical Harmonics (“SHL”) is an area of active research in computer science. Its goal is to combine the merits of Radiosity-based lighting with light sources that can move, especially when the light sources are very far away from the scene, like for example the sun or the moon.

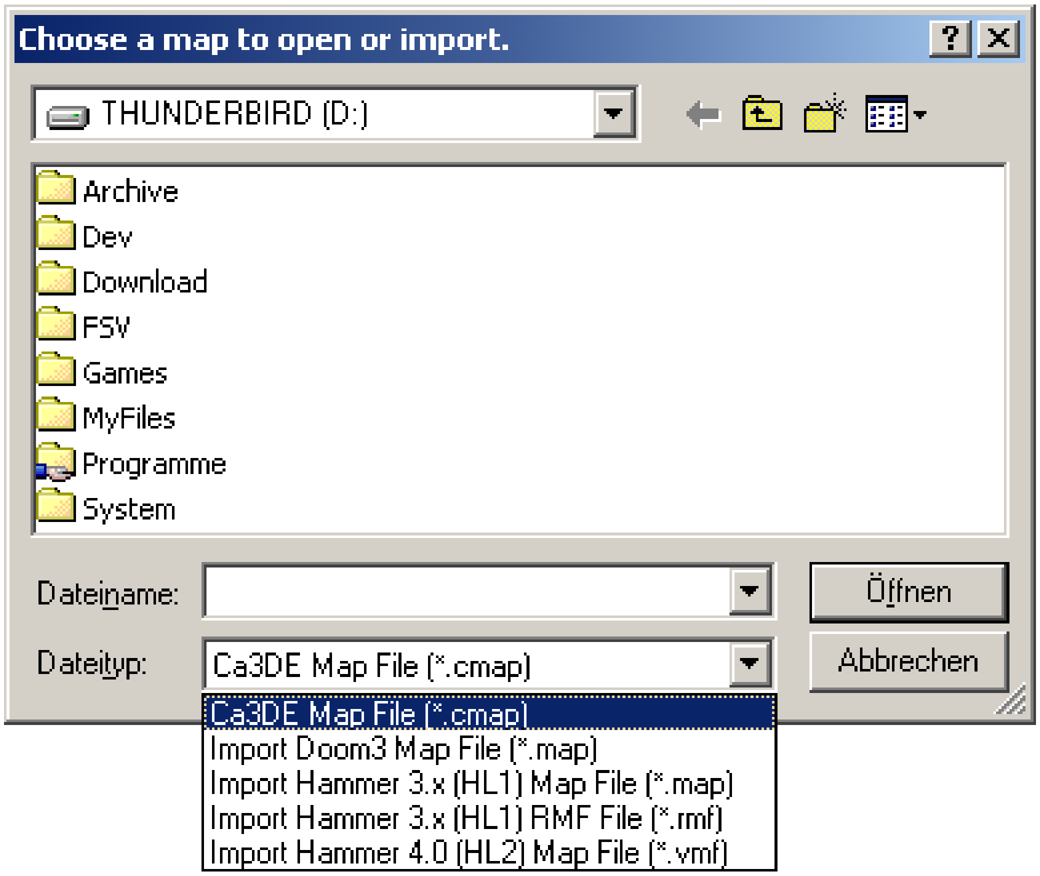
SHL has been implemented in the scope of my diploma thesis in the Cafu engine, but is currently not supported by the public releases. You may refer to the thesis linked below for more information.

9.4.4 See Also

- Methods for Real-Time Lighting – Diploma (Masters) Thesis by Carsten Fuchs.

9.5 Importing Maps from other Games/Editors

If you have already made maps with other editors for games like Half-Life 1, Half-Life 2, Quake 3 or Doom 3, you possibly wish to use them with CaWE and Cafu as well. For this purpose, several importers are provided that can load such maps directly into CaWE.


Choose a map to open or import.
THUNDERBIRD (D:)
Archive
Dev
Download
FSV
Games
MyFiles
Programme
System
Dateiname:
Dateityp:
Ca3DE Map File (*.cmap)
Öffnen
Abbrechen
Ca3DE Map File (*.cmap)
Import Doom3 Map File (*.map)
Import Hammer 3.x (HL1) Map File (*.map)
Import Hammer 3.x (HL1) RMF File (*.rmf)
Import Hammer 4.0 (HL2) Map File (*.vmf)

In order to import a foreign map, just activate the **File** → **Open…** menu item or press **CTRL+O** in order to open the file open dialog.

The drop-down box near the bottom presents a list of all file types that CaWE can directly load or import. Just choose the desired file type and proceed to open a map file of that type as usual.

After the import, there are normally two additional problems that you may encounter: material and entity mismatches.

Material mismatches are caused by the fact that the material systems of all game engines are different from each other. While we can create new materials in Cafu under the same names as those that occur in the imported map, converting the used textures and other material details is generally not feasible. Some importers therefore substitute all material occurrences in the imported map file with native Cafu materials that are valid, but have different names.

In all cases, you'll have to re-texture the imported map manually with native CaWE materials. This is recommended and frequently desired for best results anyway. Employing the powerful *Edit Surfaces* tool and the *Replace Materials* dialog gets the job quickly and conveniently done. Making use of the "Only show USED materials" checkbox in the *Material Browser* is also helpful.

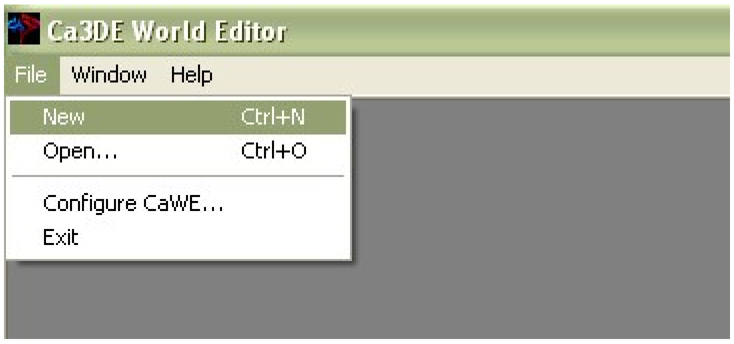
The following list summarizes what you have to know and consider about *entity mismatches*:

- Because the existing entities in your world were defined and made for another game, they will mismatch with the Cafu MOD entity definitions! Note that there are two kinds of mismatch:
  1. Some entities exist both in the other game as well as Cafu (for example translucent brushes, ladders, . . .), but their names and/or their properties differ.
  2. Other entities only exist in the other game and Cafu has no equivalent entity at all, or vice versa! For example, Cafu DeathMatch has different weapons, but unfortunately lacks of many of the advanced entities of other games. I am very sorry, but at present you have to find some kind of work-around in such cases. However, I am doing my best to improve the situation in the future.
- Here are some suggestions to fix the entity mismatchings:
  1. It's probably the safest (easiest, but most expensive) to simply delete *all* existing entities from your world, and only keep the geometry. Then, recreate them from the Cafu (DeathMatch) entities.
  2. In many cases it might be preferable to simply open the **Properties** dialog for each entity, and to reset their properties there. Just make sure that you actually check out each individual property tag.
  3. If you are very experienced, you might solve many mismatchings directly by loading the `map` file into a text editor.
- Although Cafu ignores unknown entities (e.g. entities left from other games), and fills-in default values for unknown properties, it is still best to do the conversion carefully in order to avoid unexpected results. For example, be specially careful with `func_wall` entities: you'll have to reset their *rendermode* properties (translucent, blue becomes invisible, . . .) *explicitly*!

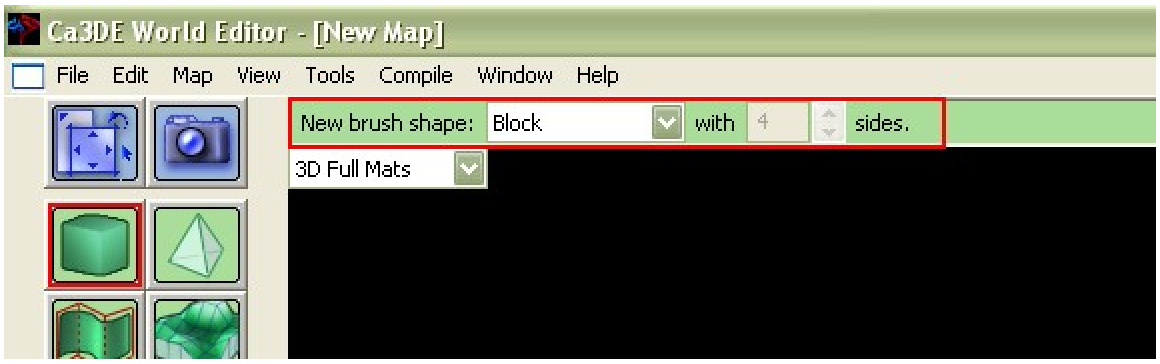
The goal of this tutorial is to show some methods creating a sky or an evironment for your scene.

9.6 Method One: Creating a skybox

First, create a new empty map by clicking "File", then "New" in the menubar. Alternatively, you can open an existing map, to which you want to add a skybox.



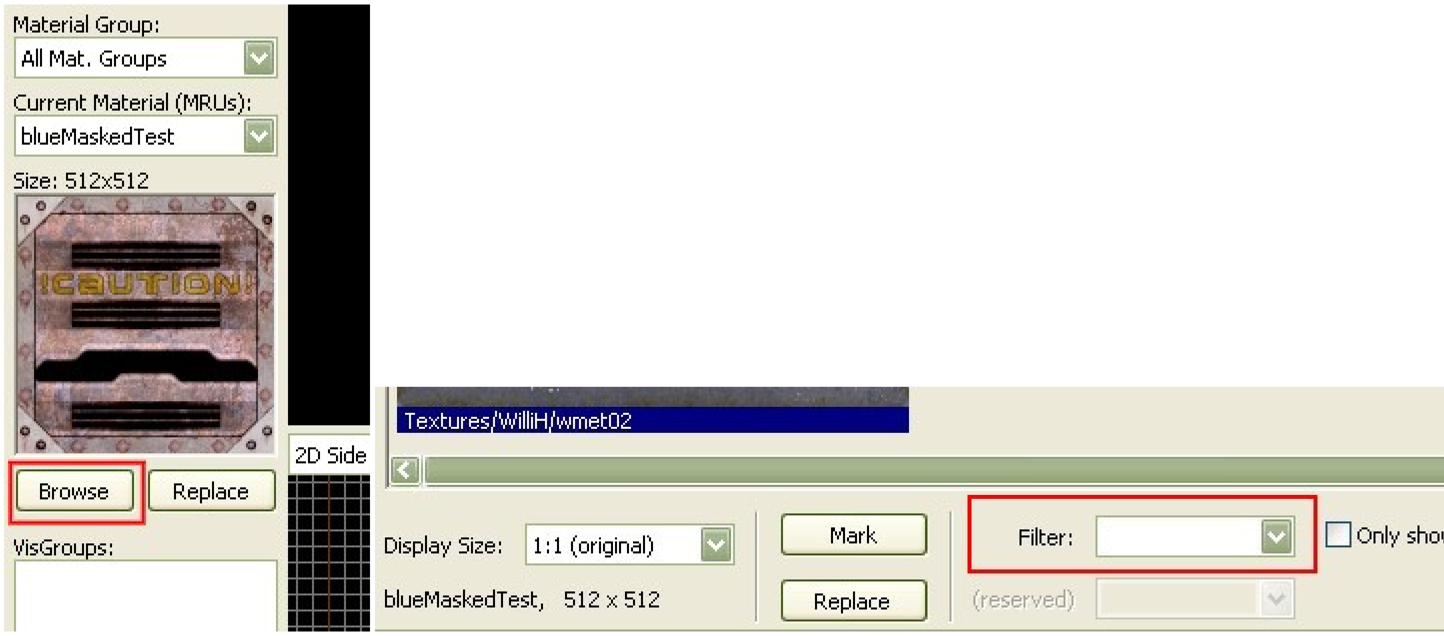
Create a block. For this, click the “New Brush” Icon in the left toolbar. In the appearing green bar at the top of the main-window check if “block” is selected as “New brush shape”. In the orthogonal windows (those with the front view, left view, down view), draw a cube. Make it’s dimensions bigger than your scene shall be, so that it surrounds the entire scene. Also, be sure that the size of height, width and depht is the same. Press enter. Don’t worry, if the size is not really exact now, you can simply correct it later.


Ca3DE World Editor - [New Map]
File Edit Map View Tools Compile Window Help
3D Full Mats

New brush shape: Block with 4 sides.

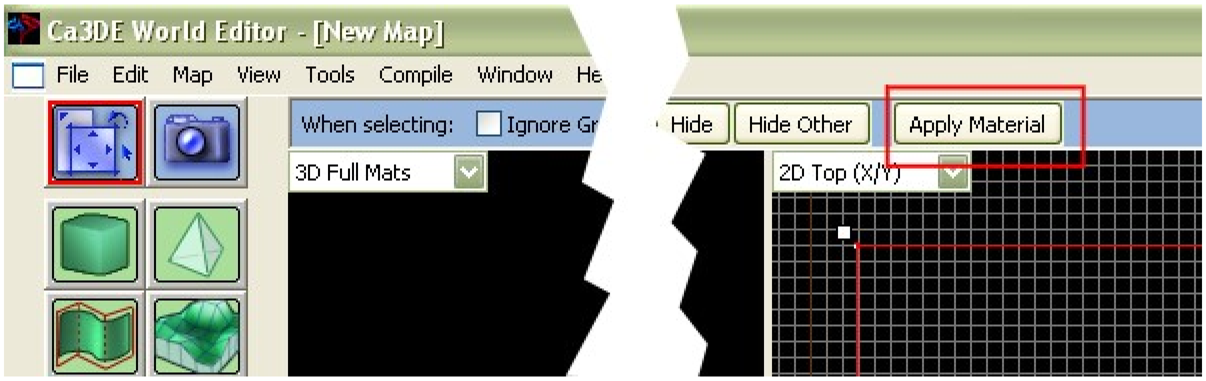
Next, choose your texture for your sky-map. You can select a texture from the skydome-folder in the textures-directory coming with your Cafu SDK, or you can make your own skybox, textures. How this can be done, I will explain later in another tutorial.

To select your sky-texture, press the “Browse”-Button, which is located under the Texture-Preview-Window. A new window opens, showing all texures the CaWe can find in it’s pathes. In the filter input box, at the bottom of this window, type sky, so that only the textures related to this word are shown.

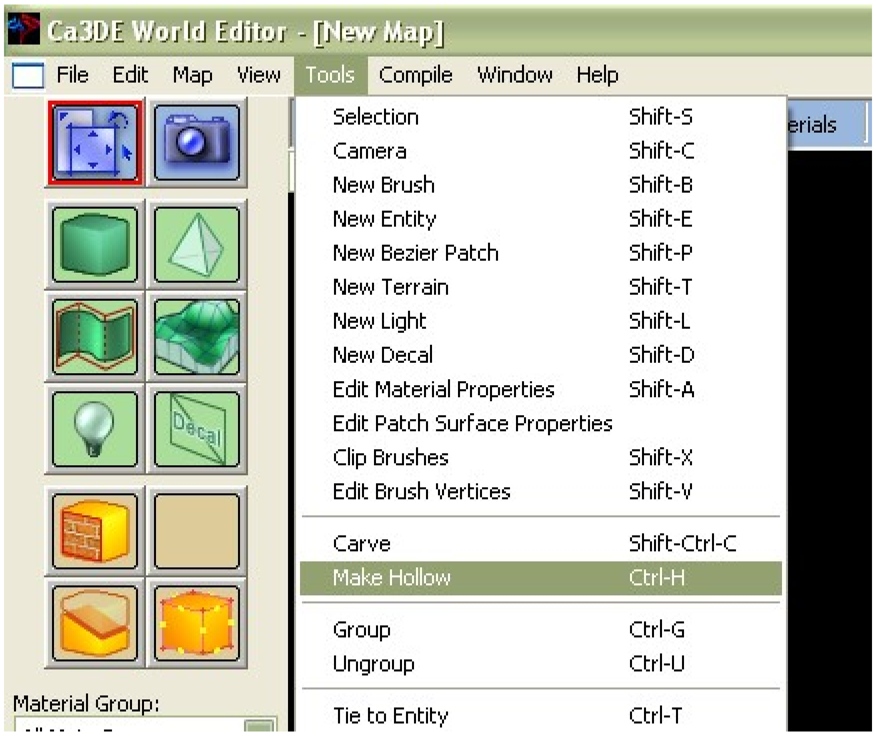


Material Group: All Mat. Groups
Current Material (MRUs): blueMaskedTest
Size: 512x512
Textures/WilliH/wmet02
Display Size: 1:1 (original)
blueMaskedTest, 512 x 512
Filter:
(reserved)

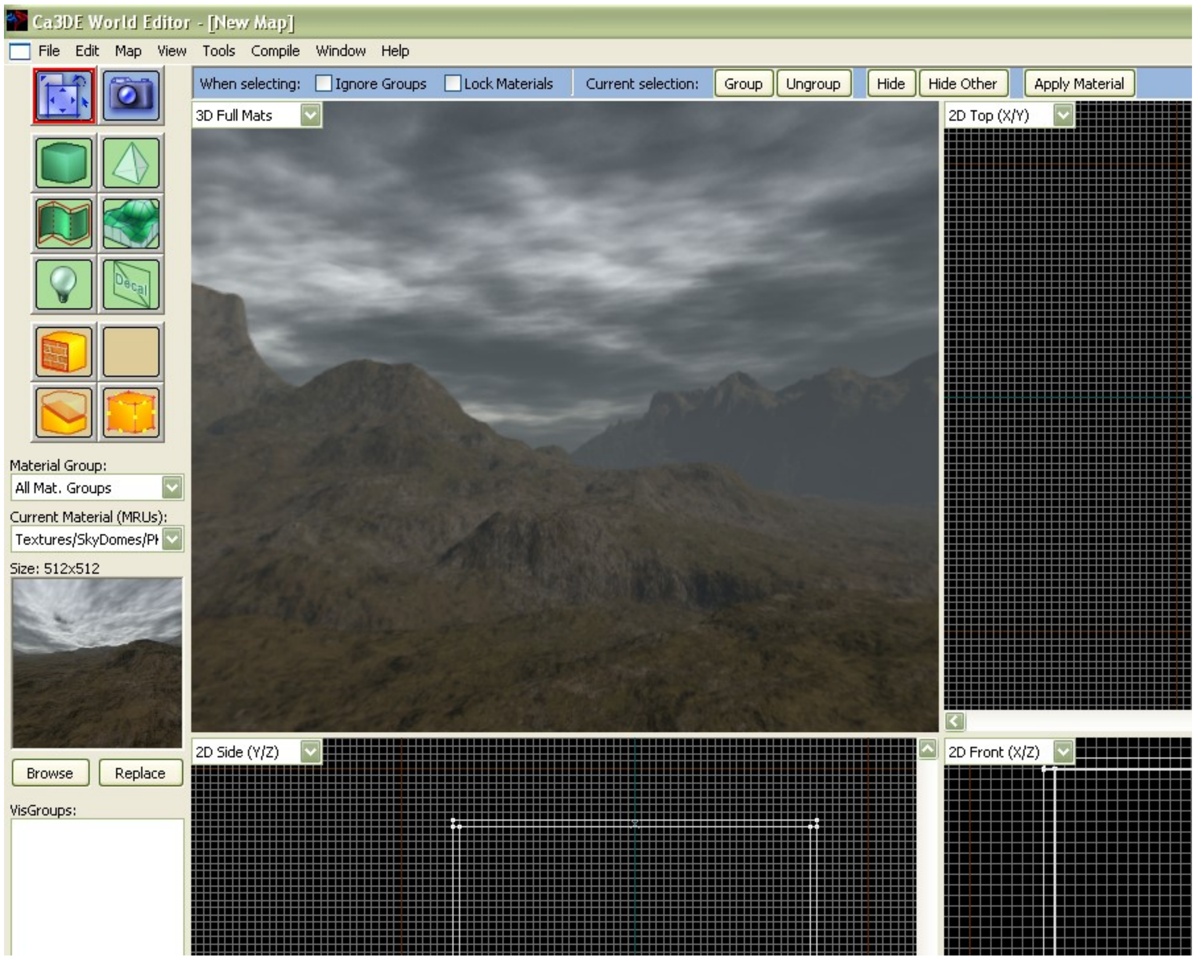
For example, select the “PK_Autumn”-Texture, by double-clicking it. The window will close, and the selected texture will be shown in the preview window. Now, select your cube with the selection tool. For this, click the upper left blue icon in the left toolbar, then click on one face of your cube. The whole cube will be drawn in red color. Alternatively, you can draw a select box around your cube, and it will be highlighted in red too. If you have a map loaded before, be sure that nothing els is selected. In the blue toolbar at the top of the main-window, click “Apply Material”. With your mouse pointer in one of the orthogonal windows, press enter. The sky-texture will be projected on your cube immediately in a perfect way. Don’t worry, if your 3D-window will stayed black. Probably your camera is inside your cube, and you are not able to see it right now. This will change during the simple next, last step.


Ca3DE World Editor - [New Map]
File Edit Map View Tools Compile Window He
When selecting: Ignore Gr
Hide
Hide Other
Apply Material
3D Full Mats
2D Top (X/Y)

Again, select your Box with the select tool. Then, in the menu-bar, first click "Tools", then "Make hollow".



In the opening window, confirm the entry of 32 units, press "Enter", and you are done. Your skybox will be shown also in the 3D-Window, and you are ready to compile your map.


Ca3DE World Editor - [New Map]
File Edit Map View Tools Compile Window Help
When selecting: Ignore Groups Lock Materials
Current selection: Group Ungroup Hide Hide Other Apply Material
3D Full Mats
2D Top (X/Y)
2D Side (Y/Z)
2D Front (X/Z)
Material Group:
All Mat. Groups
Current Material (MRUs):
Textures/SkyDomes/Pl
Size: 512x512
Browse
Replace
VisGroups:

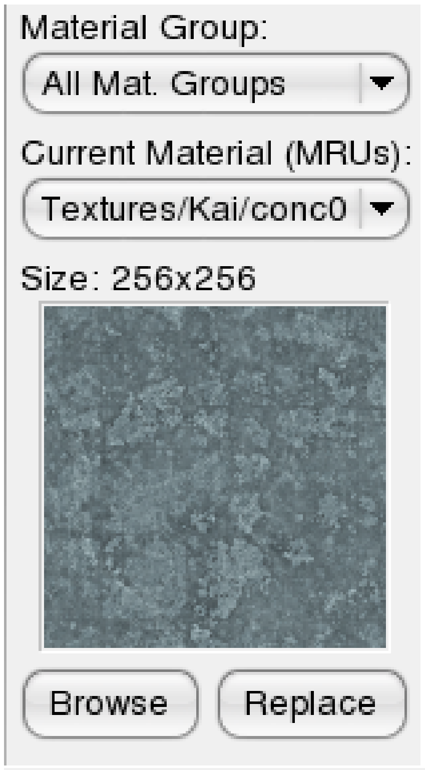
9.7 Placing Water

This tutorial was taken from the *New Entity* page.

1. Select the "*New Brush*" tool.

2. In a 2D view, drag out a box the size you want your water to be. Adjust the size in the other 2D views.

3. *(This step is not necessary for most other entities)* Click the "Browse" button in the texture settings (under the tool icons). When the viewer pops up, change the filter to "water" (without quotes). Select one of the materials.

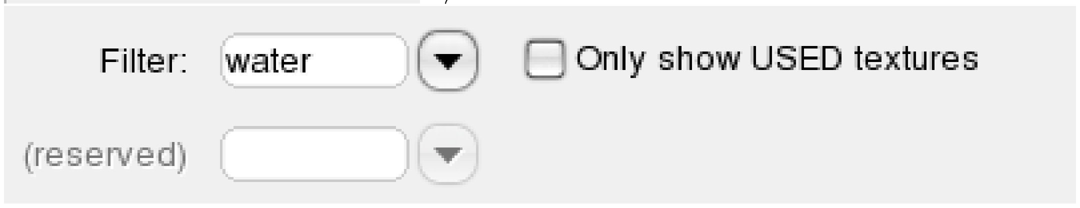


Material Group: All Mat. Groups

Current Material (MRUs): Textures/Kai/conc0

Size: 256x256

→



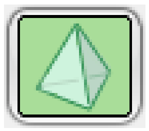
Filter: water

☐ Only show USED textures

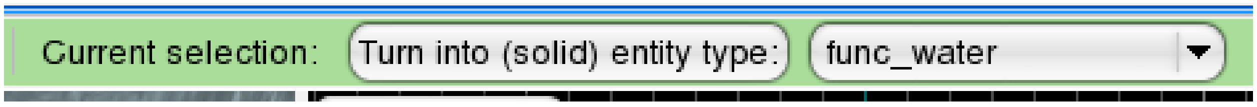
(reserved)

4. With your mouse cursor over one of the 2D views, press Enter. The brush should be selected if not, select it with the selection tool.

5. Click the [[mapping:cawe:editingtools:newentit|New Entity] tool



6. On the far right side, select "func_water" from the drop down box. Click "Turn into (solid) entity type".



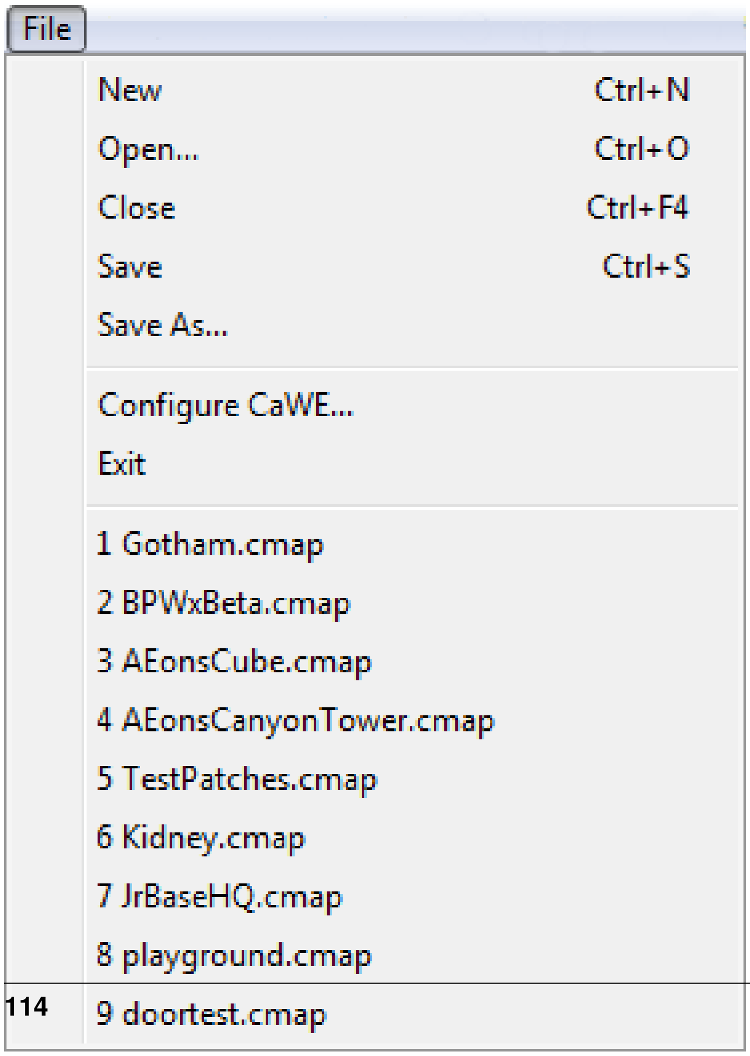
Current selection: Turn into (solid) entity type: func_water

Now compile your map, and you should see water!

CHAPTER 10

Menu Reference

10.1 The File Menu



10.1.1 New

Creates a new map.

10.1.2 Open…

Opens a .cmat file or imports another map file format. Importable file formats are:

.map	Doom 3 map format or Hammer 3.x Halflife 1 map format
.rmf	Hammer 3.x Halflife 1 map format

10.1.3 Close

Closes the currently opened map.

10.1.4 Save

Saves the map as the current filename or open **Save as…** dialog if map has no filename yet.

10.1.5 Save as…

Opens a dialog to save the map as a specific filename.

10.1.6 Configure CaWE…

Opens the *Configure CaWE Options* dialog.

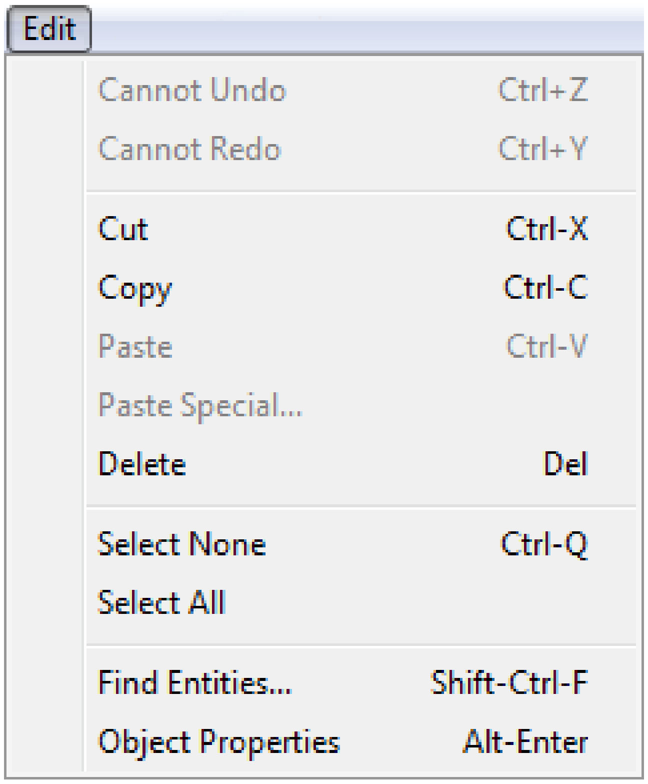
10.1.7 Exit

Exits CaWE.

10.1.8 Most recently used maps

Allows to open the 9 recently edited maps directly.

10.2 The Edit Menu



10.2.1 Undo

Revokes the last action performed by the editor.

10.2.2 Redo

Restores the last revoked action.

10.2.3 Cut

Cuts the currently selected objects from the map and stores a copy of the selection in the clipboard (requires *The Selection Tool*).

10.2.4 Copy

Stores a copy of the currently selected objects in the clipboard (requires *The Selection Tool*).

10.2.5 Paste

Pastes the content from the clipboard into the map.

10.2.6 Paste Special...

Opens the *Paste Special* dialog.

10.2.7 Delete

Deletes the currently selected objects from the map (requires *The Selection Tool*).

10.2.8 Select None

Cancels the current selection.

10.2.9 Select All

Selects all objects on the map.

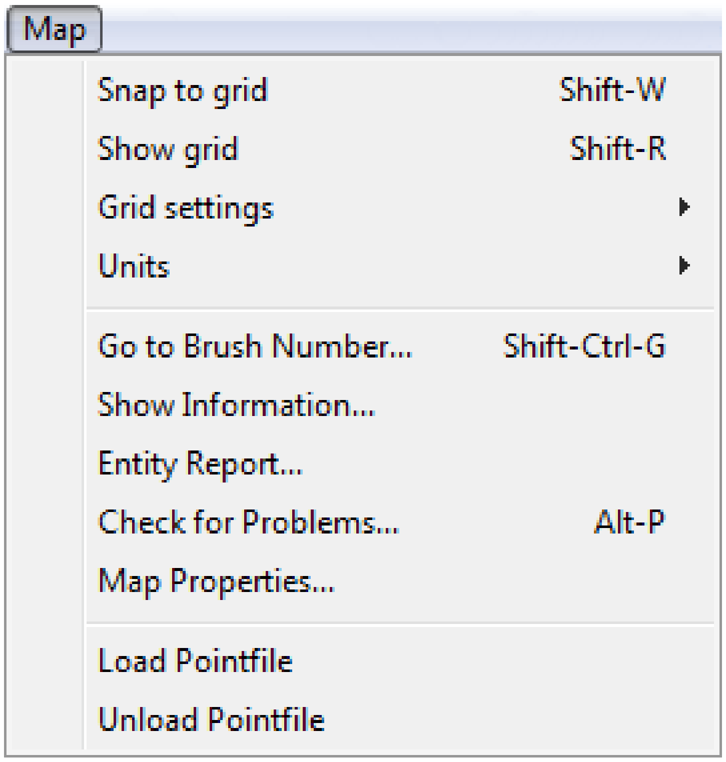
10.2.10 Find Entities...

Opens the *Find Entities* dialog.

10.2.11 Object Properties

Opens the *Object Properties* dialog for the selected object (requires *The Selection Tool*).

10.3 The Map Menu



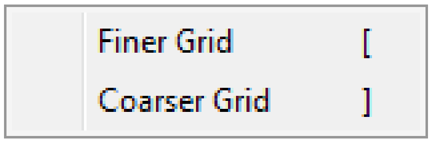
10.3.1 Snap to grid

Switches grid snapping on and off.

10.3.2 Show grid

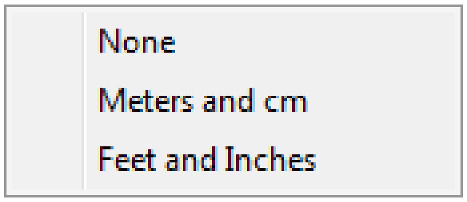
Turns grid rendering on and off.

10.3.3 Grid settings



Adjusts the grid density. **Finer Grid** increases and **Coarser Grid** decreases the grid density.

10.3.4 Units

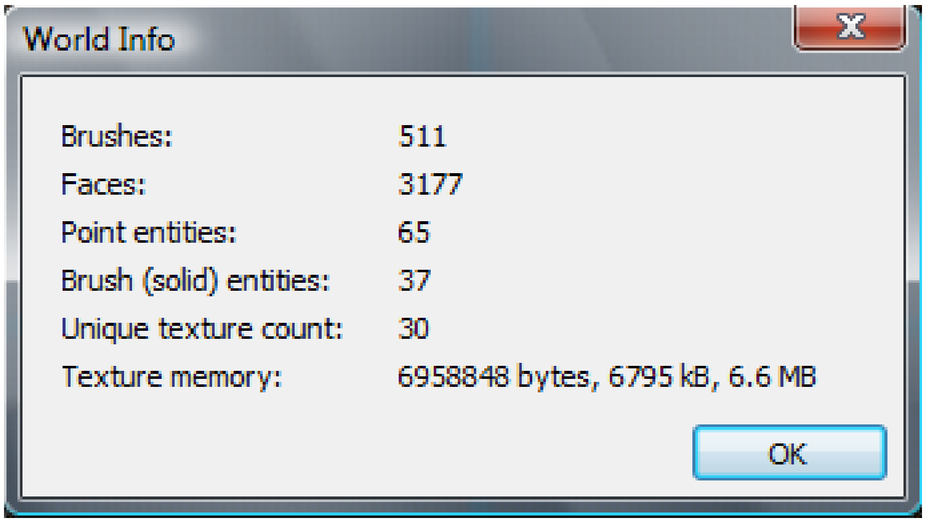


(FIXME!) not functional

10.3.5 Go to Brush Number...

Opens the *Goto Brush/Entity* dialog.

10.3.6 Show Information


World Info

Brushes:	511
Faces:	3177
Point entities:	65
Brush (solid) entities:	37
Unique texture count:	30
Texture memory:	6958848 bytes, 6795 kB, 6.6 MB

Opens the World Info dialog:

Brushes Number of brushes in this map.

Faces Number of faces in this map.

Point entities Number of point entities in this map.

Brush (solid) entities Number of entities related to a brush.

Unique texture count Number of different textures.

Texture memory Amount of memory used by the maps textures.

10.3.7 Entity Report

Opens the *Entity Report* dialog.

10.3.8 Check for Problems...

Checks for problems in the map architecture and opens the *Map Error Report* dialog if problems were found.

10.3.9 Map Properties

Selects the world object (and therefore recursively all map objects) and opens the *Inspector* dialog to show the properties of the map world entity.

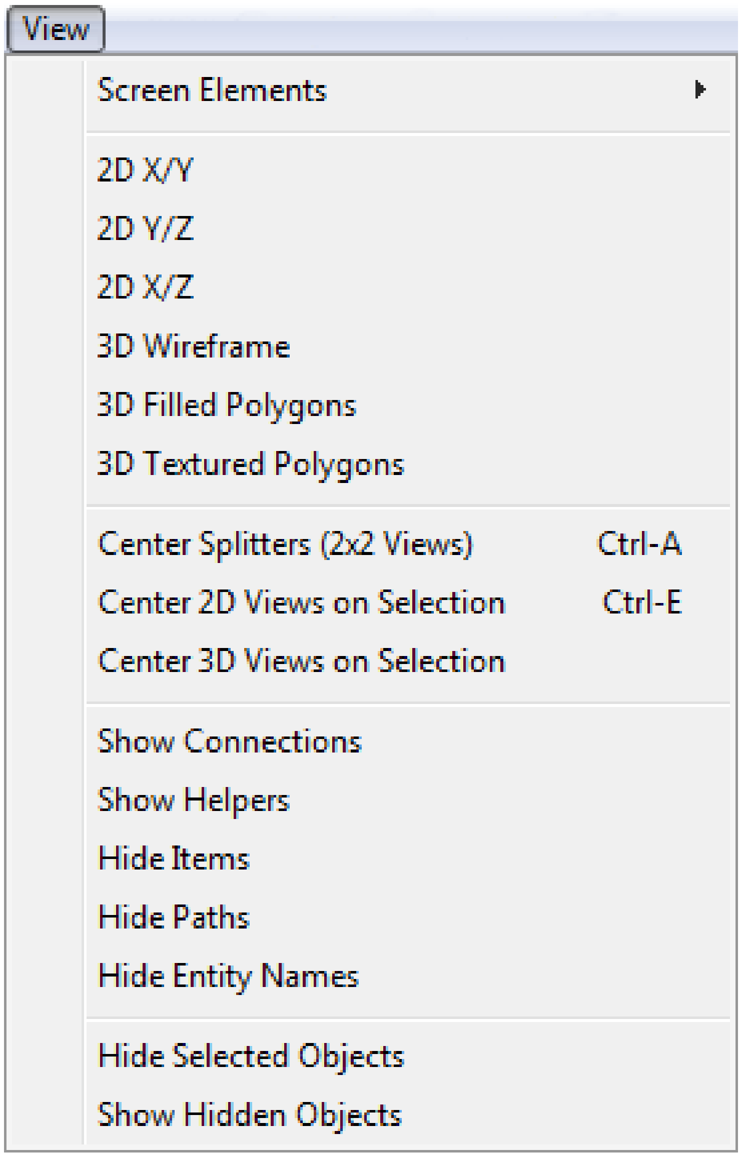
10.3.10 Load Pointfile

Loads a pointfile into the map (see *Dealing with Leaks* for more details).

10.3.11 Unload Pointfile

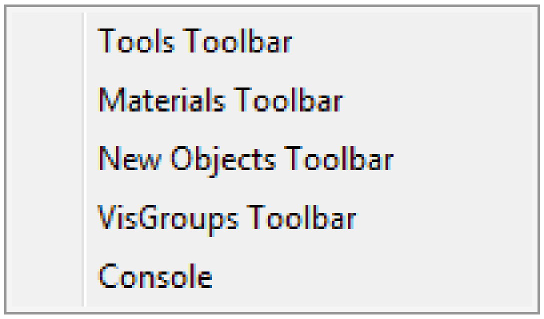
Removes a previously loaded pointfile.

10.4 The View Menu



The View menu contains items to control the display of screen elements of the currently active map window.

10.4.1 Screen Elements



Toggles the visibility of a specific screen element (see right screen). Currently this works only for **Console**.

10.4.2 2D X/Y

This menu item will be revised and documented later.

10.4.3 2D Y/Z

This menu item will be revised and documented later.

10.4.4 2D X/Z

This menu item will be revised and documented later.

10.4.5 3D Wireframe

This menu item will be revised and documented later.

10.4.6 3D Filled Polygons

This menu item will be revised and documented later.

10.4.7 3D Textured Polygons

This menu item will be revised and documented later.

10.4.8 Center Splitters (2x2 Views)

If the splitters that separate the 3D view and 2D views have been shifted, this item moves them back to their initial position.

10.4.9 Center 2D Views on Selection

Centers all 2D views onto the selected object.

10.4.10 Center 3D Views on Selection

Centers the 3D view onto the selected object.

10.4.11 Show Connections

This menu item will be revised and documented later.

10.4.12 Show Helpers

This menu item will be revised and documented later.

10.4.13 Hide Items

Toggles point based entities visibility.

10.4.14 Hide Paths

This menu item will be revised and documented later.

10.4.15 Hide Entity Names

Toggles entity names visibility in 2D views.

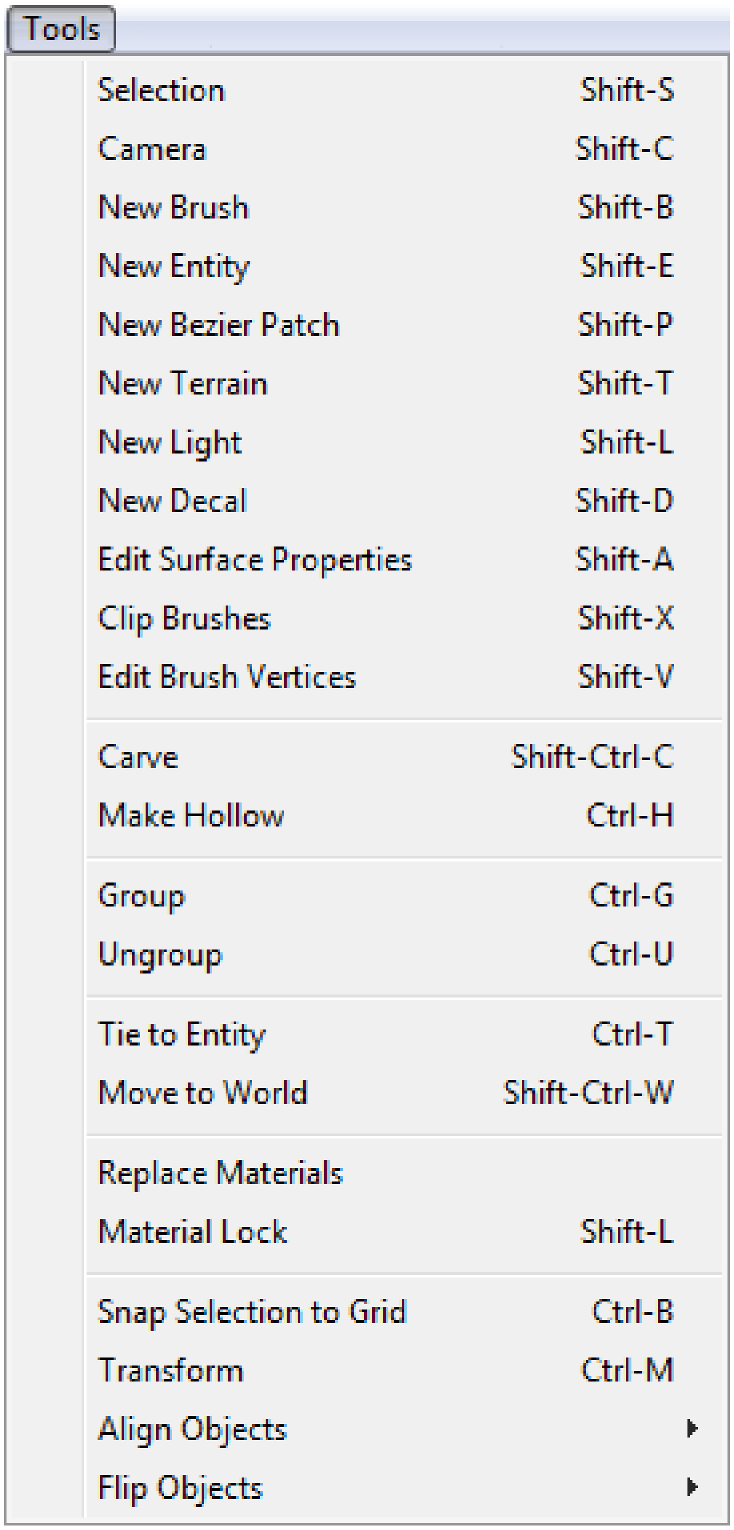
10.4.16 Hide Selected Objects

Hides the currently selected objects.

10.4.17 Show Hidden Objects

Shows all currently hidden objects.

10.5 The Tools Menu



Contains items that are related to CaWE tools as well as object alteration.

10.5.1 CaWE Tools

Opens the chosen tool in the same way as when clicked on the related icon in the tool bar.

10.5.2 Carve

Subtracts the space occupied by the selected brush from all other brushes with whom the selected brush intersects.

10.5.3 Make Hollow

Turns all faces of the selected brush into a solid wall with a thickness, that is specified in a popup dialog.

10.5.4 Group

Puts all selected objects into a group, so they can be selected together by clicking at one group member.

10.5.5 Ungroup

Cancels the group status of all selected groups.

10.5.6 Tie to Entity

Transforms the selected brushes to an solid entity.

10.5.7 Move to World

Transforms a solid entity back to a normal brush.

10.5.8 Replace Materials

Opens the *Replace Materials* dialog.

10.5.9 Material Lock

If activated materials are locked and no longer automatically shifted if you move, resize, rotate or shear a brush and its faces.

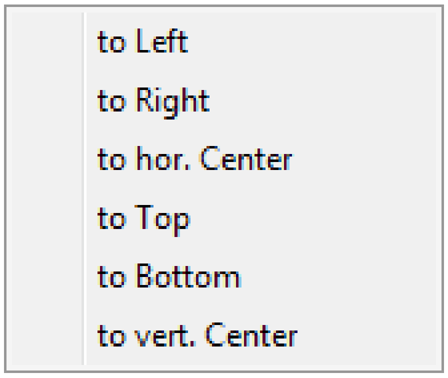
10.5.10 Snap Selection to Grid

Snaps the point of origin of the selection onto the nearest grid point.

10.5.11 Transform

Opens the *Transform Type-In* dialog.

10.5.12 Align Objects



Only works for more than one selected object

to Left Moves all objects to the most left edge of the selection.

to Right Moves all objects to the most right edge of the selection.

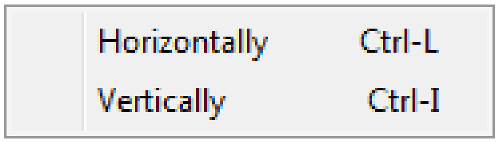
to hor. Center Moves all objects to the horizontal center of the selection.

to Top Moves all objects to the top edge of the selection.

to Bottom Moves all objects to the bottom edge of the selection.

to vert. Center Moves all objects to the vertical center of the selection.

10.5.13 Flip Objects



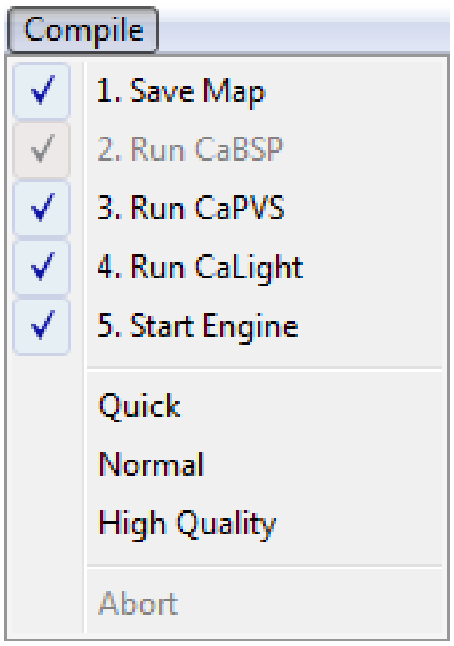
Only works for more than one selected object

Horizontally Flips the objects along a horizontal line trough the center of the selection.

Vertically Flips the objects along a vertical line trough the center of the selection.

10.6 The Compile Menu

More details about map compiling can be found *here*.



10.6.1 Compile Options

1. **Save Map**: Saves the map before compiling.
2. **Run CaBSP**: Runs the CaBSP compiler when compiling (this can't be deactivated since it is the essential part of compiling a map).
3. **Run CaPVS**: Runs the CaPVS (Potentially Visibility Set) compiler when compiling a map.
4. **Run CaLight**: Precomputes static lighting when compiling a map.
5. **Start Engine**: Starts the engine when the map is compiled.

10.6.2 Quick

Compiles the map in low quality.

10.6.3 Normal

Compiles the map in medium quality.

10.6.4 High Quality

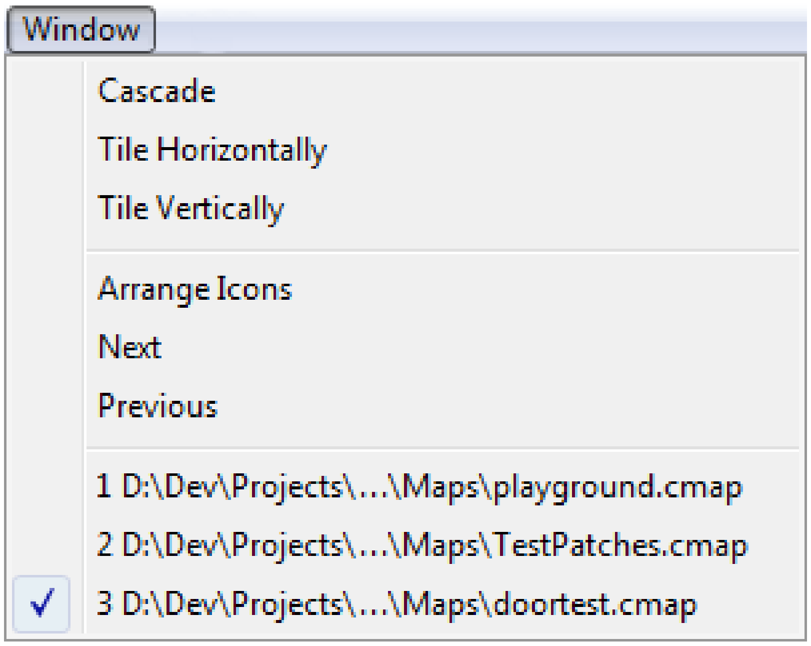
Compiles the map in high quality.

10.6.5 Abort

Aborts the currently running compiling process.

10.7 The Window Menu

This menu is only relevant if you have more than one map opened in CaWE. It contains items to control the display of the map windows.


Window
Cascade
Tile Horizontally
Tile Vertically
Arrange Icons
Next
Previous
1 D:\Dev\Projects\...\Maps\playground.cmap
2 D:\Dev\Projects\...\Maps\TestPatches.cmap
✓ 3 D:\Dev\Projects\...\Maps\doortest.cmap

10.7.1 Cascade

Cascades all currently open map windows.

10.7.2 Tile Horizontally

Displays all currently open map windows and tiles them horizontally.

10.7.3 Tile Vertically

Displays all currently open map windows and tiles them vertically.

10.7.4 Arrange Icons

Arranges the icons off all minimized map windows in the lower left corner of the program window.

10.7.5 Next

Displays the next map window.

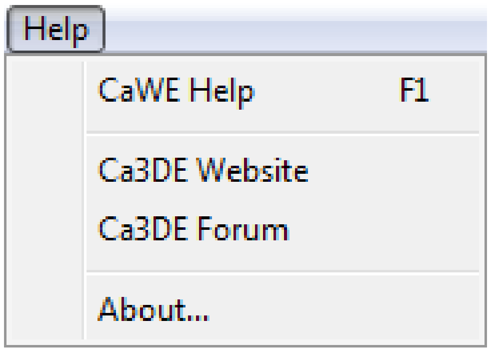
10.7.6 Previous

Displays the previous map window.

10.7.7 Currently opened maps

Shows all currently opened maps and marks the displayed map. If you click on the name of a map it is displayed in the window.

10.8 The Help Menu



10.8.1 CaWE Help

Opens this Wiki in the default browser.

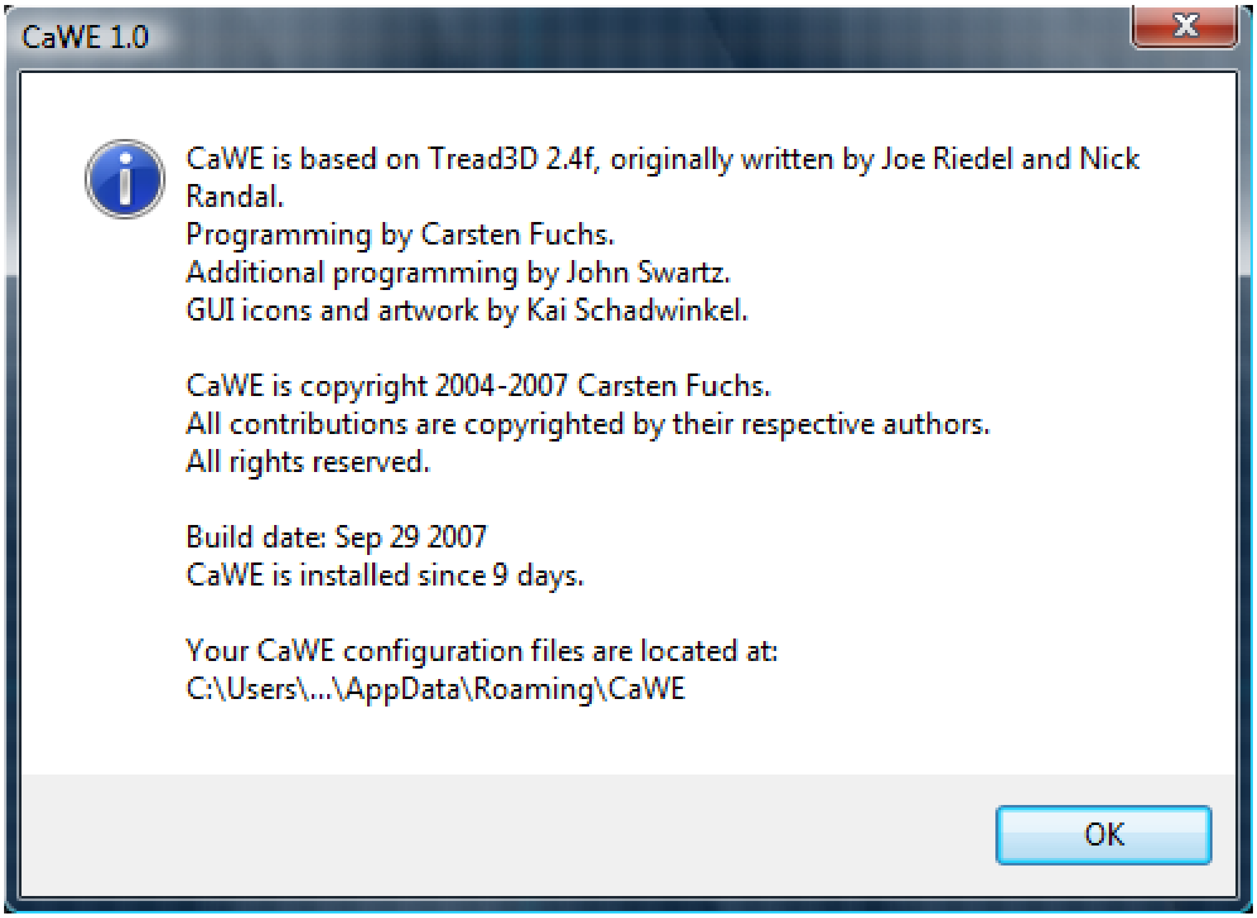
10.8.2 CaWE Website

Opens the Cafu Website in the default browser.

10.8.3 CaWE Forum

Opens the Cafu Forum in the default browser.

10.8.4 About...



CaWE is based on Tread3D 2.4f, originally written by Joe Riedel and Nick Randal.
Programming by Carsten Fuchs.
Additional programming by John Swartz.
GUI icons and artwork by Kai Schadwinkel.

CaWE is copyright 2004-2007 Carsten Fuchs.
All contributions are copyrighted by their respective authors.
All rights reserved.

Build date: Sep 29 2007
CaWE is installed since 9 days.

Your CaWE configuration files are located at:
C:\Users\...\AppData\Roaming\CaWE

Shows informations about CaWE in a popup window.

CHAPTER 11

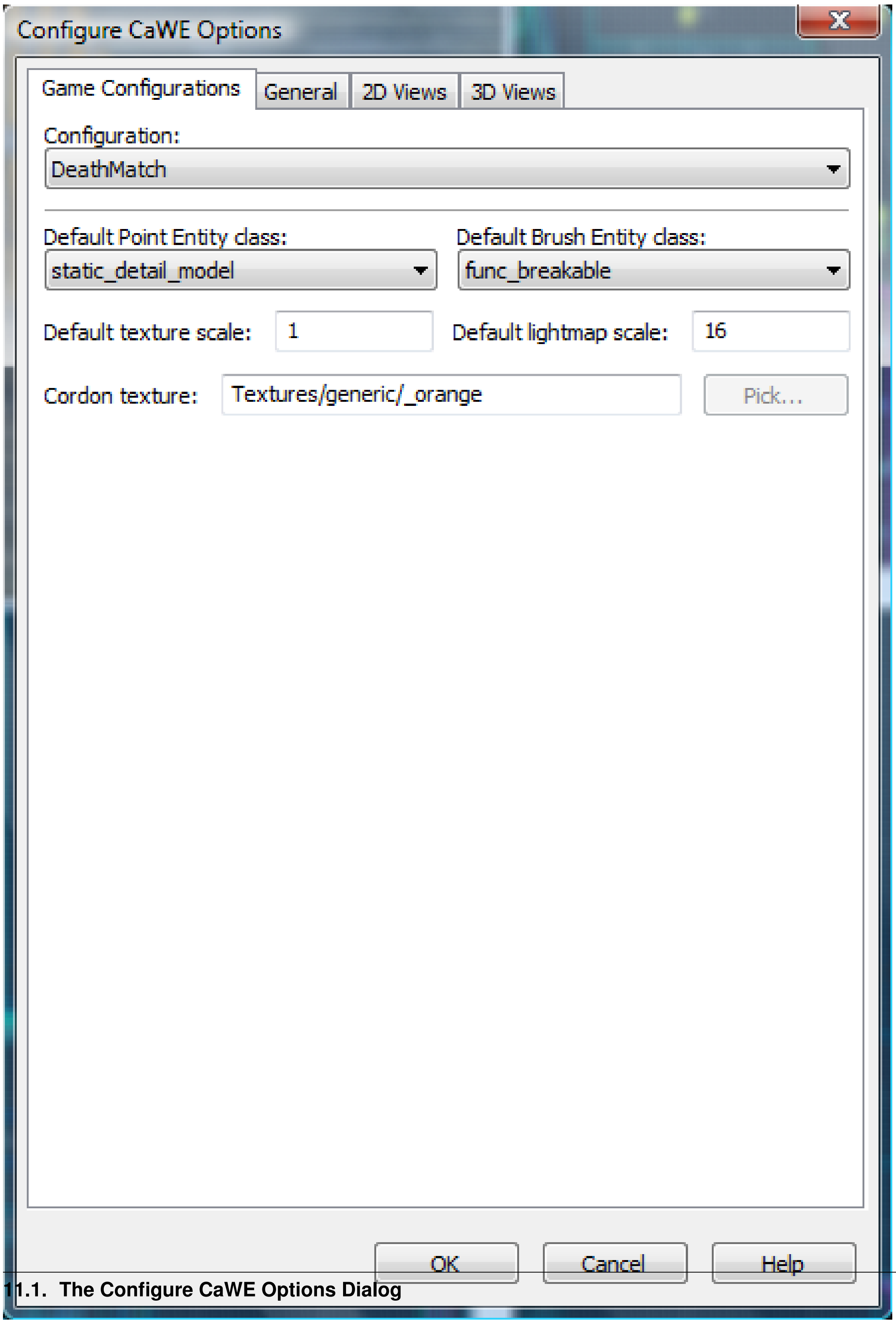
Dialog Reference

11.1 The Configure CaWE Options Dialog

This dialog is opened by selecting **Configure CaWE…** from *The File Menu*.

It is used to configure the general appearance and handling of CaWE.

11.1.1 Game Configurations



Configuration: DeathMatch

Default Point Entity class: static_detail_model

Default Brush Entity class: func_breakable

Default texture scale: 1

Default lightmap scale: 16

Cordon texture: Textures/generic/_orange

This tab contains configuration options for game configurations. Each map has to use a game configuration in order to work.

Game configurations are automatically created from the subdirectories of /Games that have a game data file (.fgd) with the same name as the directory in it. For example /Games/Deathmatch/Deathmatch.fgd is a valid game configuration with the name Deathmatch.

In this tab you can edit properties for game configurations that will be used when editing a map that uses this game configuration.

Game Configuration Selection Selects the game configuration to adjust its properties.

Default Point Entity class Sets the Point Entity class that is selected by default when a new map is opened/created.

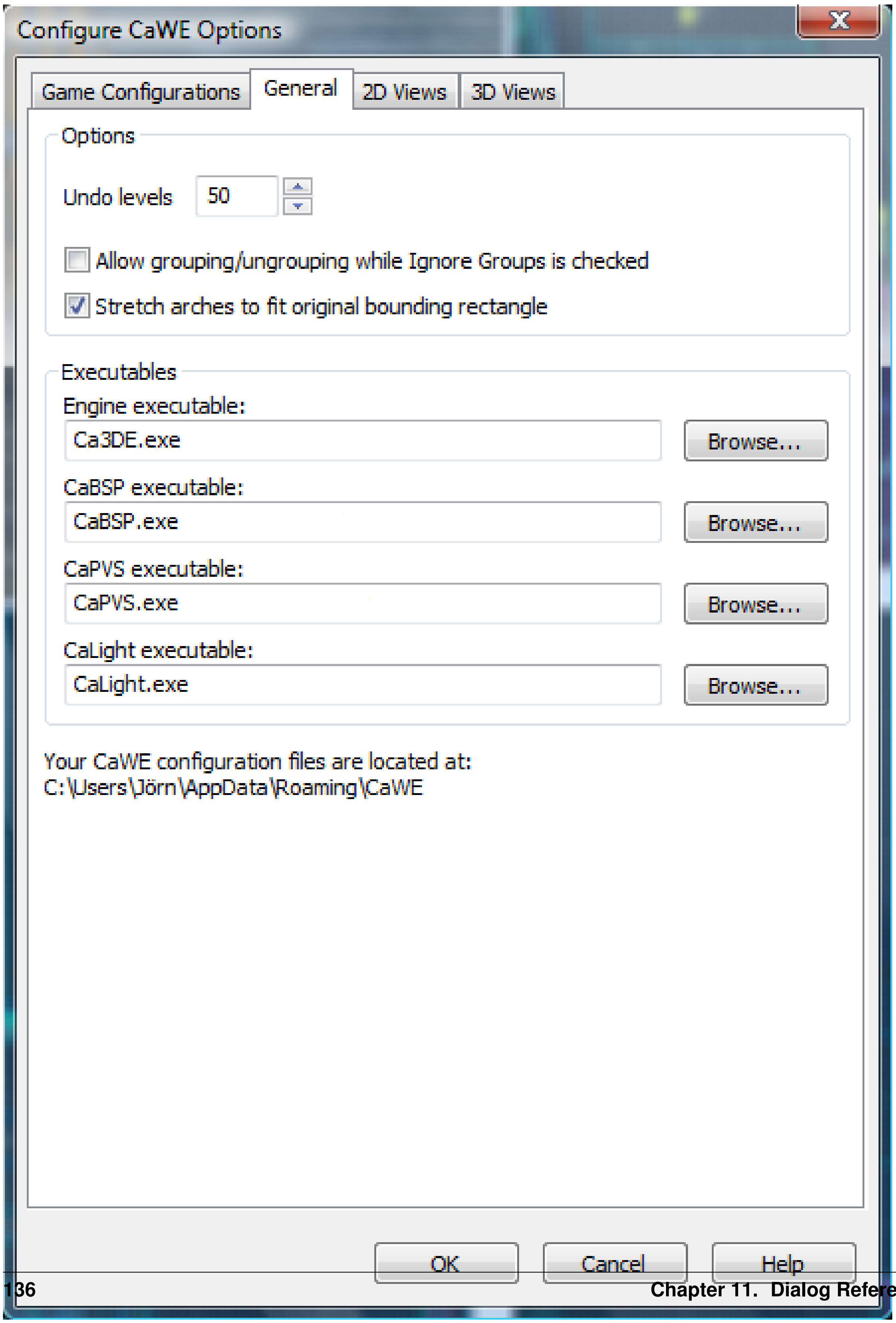
Default Brush Entity class Sets the Brush Entity class that is selected by default when a new map is opened/created.

Default texture scale Sets the default texture scale for this profile.

Default lightmap scale Sets the default lightmap scale for this profile.

Cordon texture This option is not implemented at this time.

11.1.2 General



Undo levels 50

[ ] Allow grouping/ungrouping while Ignore Groups is checked

[✓] Stretch arches to fit original bounding rectangle

Engine executable: Ca3DE.exe

CaBSP executable: CaBSP.exe

CaPVS executable: CaPVS.exe

CaLight executable: CaLight.exe

Your CaWE configuration files are located at:
C:\Users\Jörn\AppData\Roaming\CaWE

This tab contains general configuration options for the editor to control its behavior. The paths to the executables in this tab are very important so CaWE can find them to compile and run edited maps.

Undo Levels Sets the maximum number of actions performed by the editor that can be revoked.

Allow grouping/ungrouping while Ignore Groups is checked Toggles if you can group or ungroup selected objects even if the **Ignore Groups** Option from the *Selection Tool* options bar is checked.

Stretch arches to fit original bounding rectangle This option stretches a arch to fit the bounding rectangle when created. If this options is not selected the size of the arch will be determined by the size of the bounding rectangle as well as the parameters set in the arch creation dialog (see *The New Brush Tool*).

Executables

The path of executables used to compile and run maps can be configured here in case the user changed their paths or wants to use different ones.

Engine executable Sets the path to the Cafu executable.

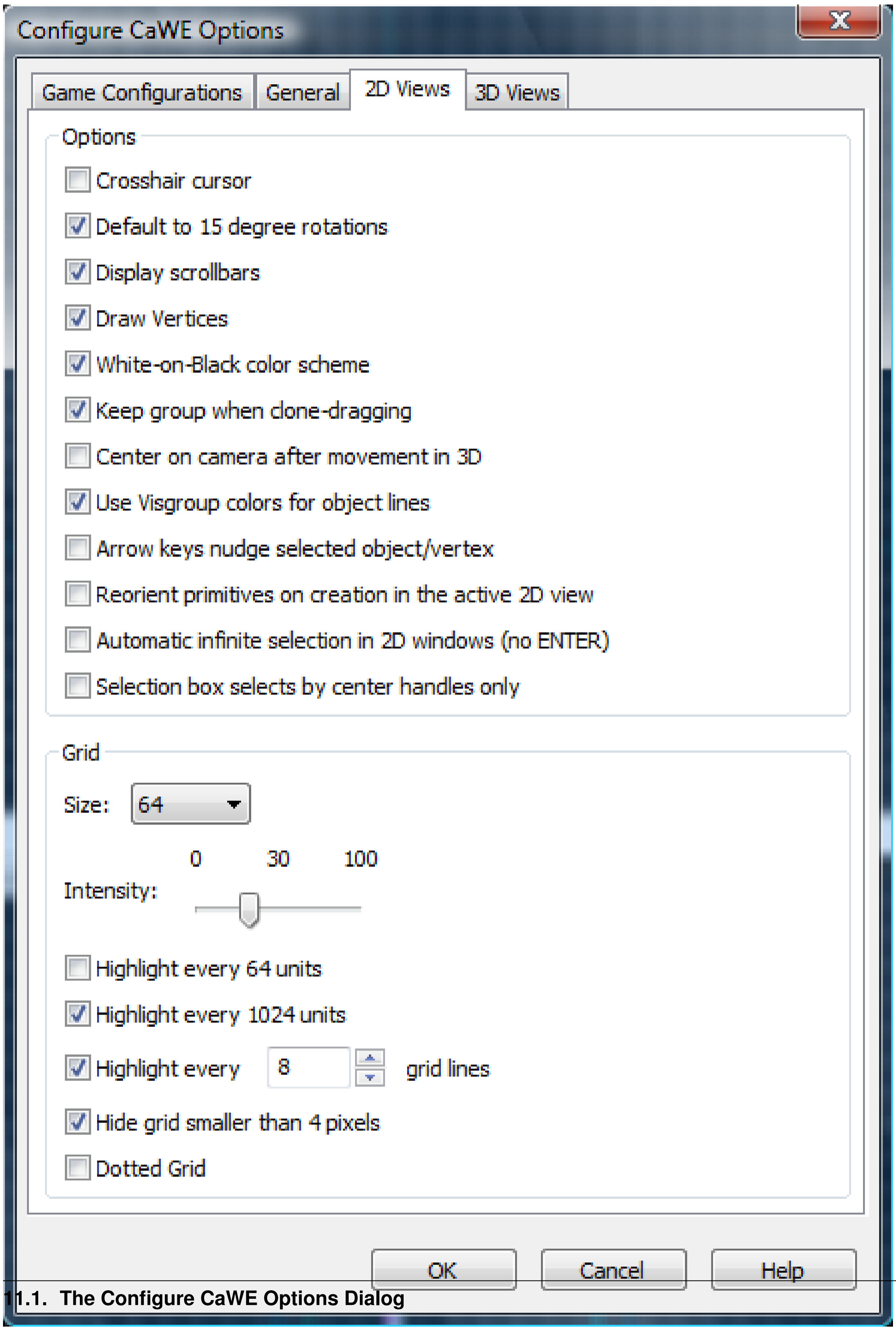
CaBSP executable Sets the path to the CaBSP executable.

CaPVS executable Sets the path to the CaPVS executable.

CaLight executable Sets the path to the CaLight executable.

At the bottom of this tab, the path to your configuration data is shown.

11.1.3 2D Views



Options

- [ ] Crosshair cursor
- [✓] Default to 15 degree rotations
- [✓] Display scrollbars
- [✓] Draw Vertices
- [✓] White-on-Black color scheme
- [✓] Keep group when clone-dragging
- [ ] Center on camera after movement in 3D
- [✓] Use Visgroup colors for object lines
- [ ] Arrow keys nudge selected object/vertex
- [ ] Reorient primitives on creation in the active 2D view
- [ ] Automatic infinite selection in 2D windows (no ENTER)
- [ ] Selection box selects by center handles only

Grid

Size: 64

Intensity: 0 30 100

- [ ] Highlight every 64 units
- [✓] Highlight every 1024 units
- [✓] Highlight every 8 grid lines
- [✓] Hide grid smaller than 4 pixels
- [ ] Dotted Grid

This tab holds configuration options that define the appearance of the editors 2D views.

Options

Crosshair cursor This option is not implemented at this time.

Default to 15 degree rotations If rotating an object this sets the rotation steps to 15 degree instead of smooth rotating. Note that you can still rotate objects smoothly by pressing `SHIFT` while rotating.

Display scrollbars Toggles the display of scrollbars in the 2D views.

Draw vertices Toggles display of objects vertices in a 2D view.

White-on-Black color scheme Switches between white grid on black background and black grid on white background modes.

Keep group when done dragging When cloning objects from a VisGroup, this option adds the cloned objects to the same VisGroup after dragging them.

Center on camera after movement in 3D This option is not implemented at this time.

Use Visgroup colors for object lines Toggles if the line color defined in the Edit VisGroups dialog is used to display the objects that belong to this group in a 2D view.

Arrow keys nudge selected object/vertice If activated the arrow keys can be used to move objects or vertices.

Reorient primitives on creation in the active 2D view When creating a brush in a 2D view, this brushes top side is per default oriented in the direction of the top down 2D view. If this option is activated the top side of the brush will be oriented in the direction of the 2D view in which the brush has been created.

Automatic infinite selection in 2D windows (no ENTER) If dragging a selection box, this option determines if the objects in the selection box are selected instantly after the mouse button is released.

Selection box selects by center handles only Determines if the center handle of objects has to lie within a selection box for the object to be selected. Otherwise any object that lies partially in the selection box is selected.

Grid

Size Defines the default grid size.

Intensity Adjusts the grids color intensity.

Highlight every 64 units If checked, a grid line is highlighted every 64 units.

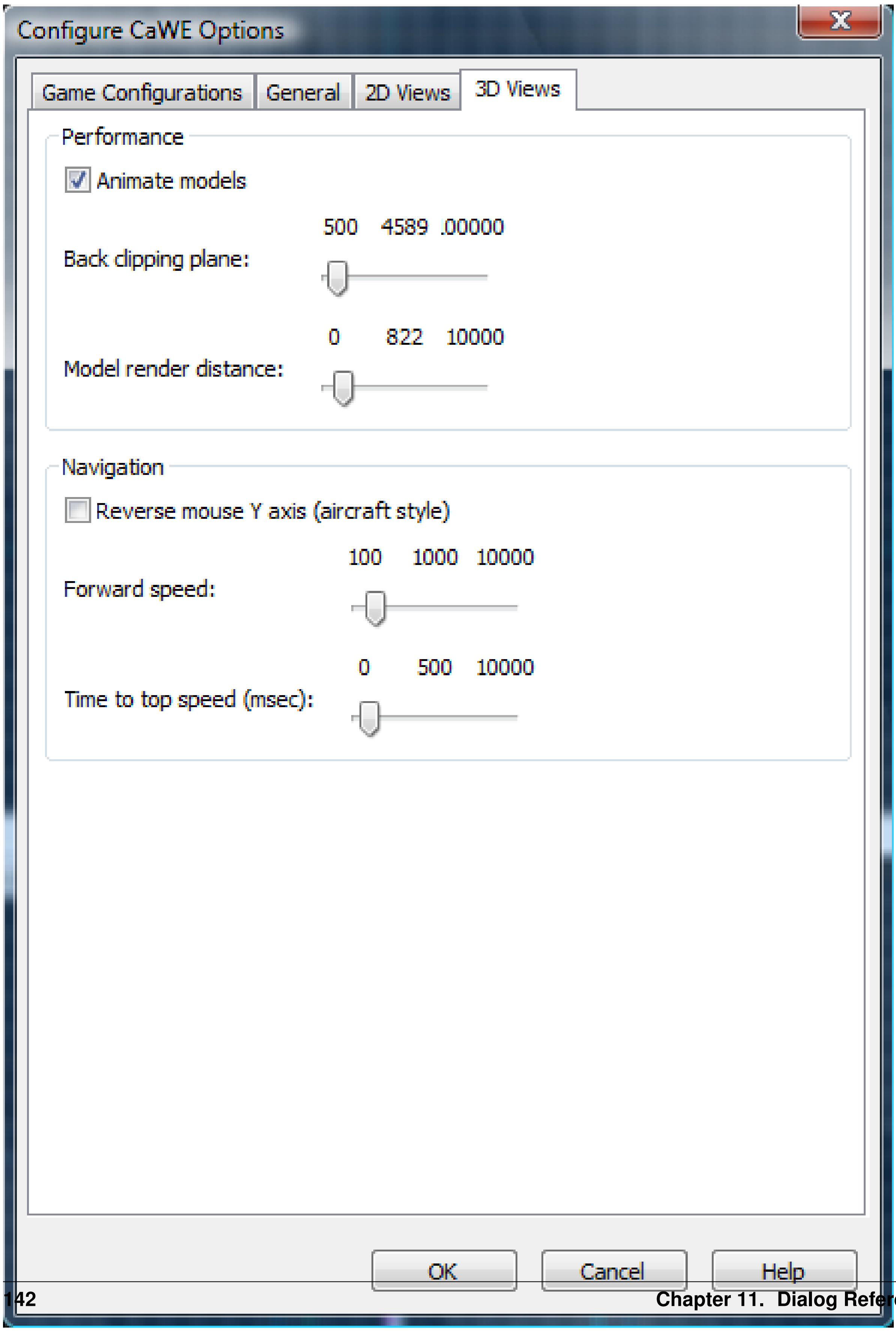
Highlight every 1024 units If checked, a grid line is highlighted every 1024 units.

Highlight every *X* grid lines If a grid line is the *X*. grid line in a row, it is highlighted.

Hide grid smaller than 4 pixel If the distance between two grid lines gets smaller than 4 pixels in a 2D view, the grid lines are no longer displayed.

Dotted Grid Toggles between a grid with solid lines or dots.

11.1.4 3D Views



Configure CaWE Options

Game Configurations | General | 2D Views | 3D Views

Performance

[✓] Animate models

Back clipping plane: 500 4589 .00000

Model render distance: 0 822 10000

Navigation

[ ] Reverse mouse Y axis (aircraft style)

Forward speed: 100 1000 10000

Time to top speed (msec): 0 500 10000

This tab holds configuration options that define the appearance of the editors 3D view.

Performance

Animate models With this option activated, models are animated in the editor.

Back clipping plane Sets the back clipping plane in the editors 3D view. With high values even geometry far away from the camera is rendered. This option has a huge impact on the performance of the editor, especially in big maps.

Model render distance Determines the distance at which models are rendered as bounding boxes instead of the real model meshes.

Navigation

Reverse mouse Y axis (aircraft style) Toggles reversion of the Y axis of the mouse, when changing the cameras view direction in a 3D view with the mouse.

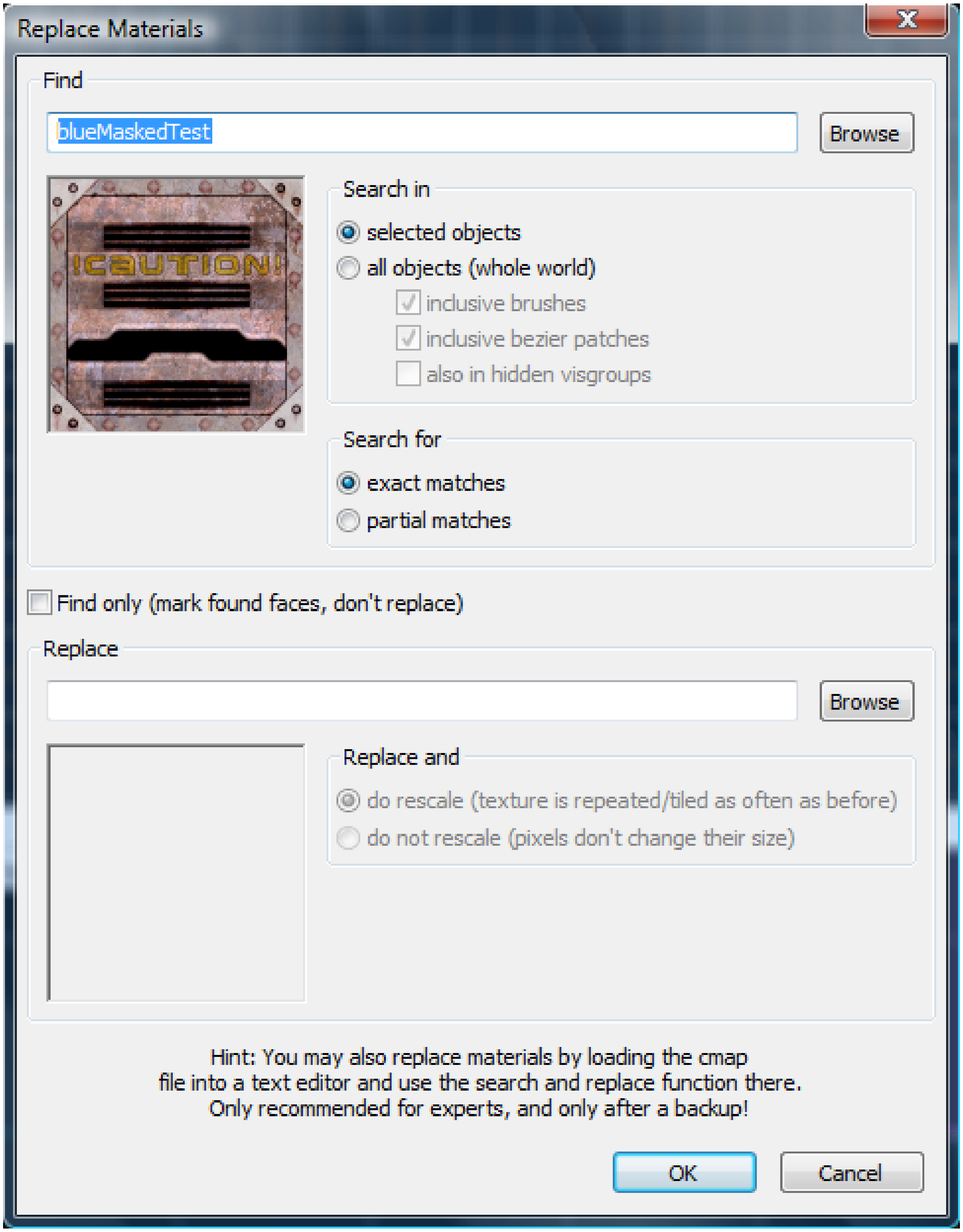
Forward speed Sets the maximum speed when moving forward in a 3D view.

Time to top speed (msec) Adjusts the time until a movement in a 3D view speeds up to its maximum speed.

11.2 The Replace Materials Dialog

This dialog can be opened by clicking on the related menu item in *The Tools Menu* or by clicking on the icon in the materials bar (see *The Main Window User Interface*).

With this dialog you can find a specific material and its related faces and replace this material by another one.


Replace Materials

Find

blueMaskedTest

Browse

Search in

[✓ selected objects]

all objects (whole world)

[✓] inclusive brushes

[✓] inclusive bezier patches

[ ] also in hidden visgroups

Search for

[✓ exact matches]

partial matches

[ ] Find only (mark found faces, don't replace)

Replace

Browse

Replace and

[✓ do rescale (texture is repeated/tiled as often as before)]

do not rescale (pixels don't change their size)

Hint: You may also replace materials by loading the cmap
file into a text editor and use the search and replace function there.
Only recommended for experts, and only after a backup!

OK

Cancel

11.2.1 Find

Material field Name of the material to search for. **Browse** opens *The Material Browser*.

11.2.2 Search in

selected objects Searches only in the currently selected objects for faces with the material to find.

all objects (whole world) Searches in all map objects.

> **inclusive brushes** Includes brushes into the search.
>
> **inclusive bezier patches** Includes bezier patches into the search.
>
> **also in hidden visgroups** Includes invisible objects into the search.

11.2.3 Search for

exact matches Only materials that match the exact material name given in the material field are found.

partial matches Materials whose names match the searched material name partially are also found (e.g. search name: wall; partially matches with wall_01, mywall, etc.).

Find only Found faces with the specified material are selected but the material is not replaced.

11.2.4 Replace

Material field Name of the material to replace with. **Browse** opens *The Material Browser*.

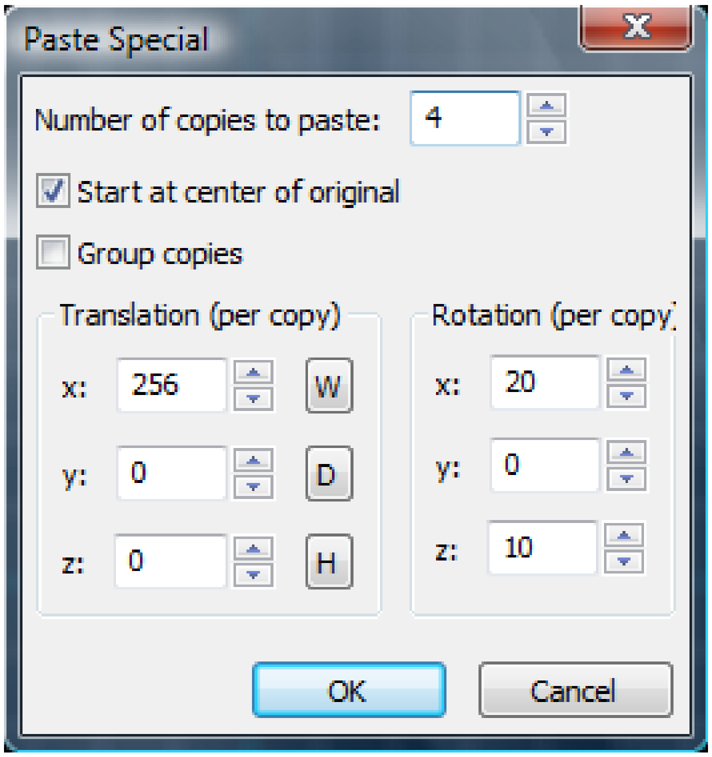
Replace and rescale The replacement material is rescaled in the same way as the old material.

Replace and not rescale The replacement material is applied without rescaling.

11.3 The Paste Special Dialog

This dialog is opened by clicking on **Paste Special...** in *The Edit Menu*. This option is only available if an object has been copied to the clipboard before.

Paste Special allows you to paste more than one copy of a selection into the map. You can for example paste a row of the same object into the map, whereat the object is pasted multiple times and each copy is shifted in position to form a row with the other objects.



Number of copies to paste The number of copies from the objects in the clipboard that will be pasted.

Start at center of original If this is activated, pasting starts at the center of the original object/s. Otherwise it starts at the center of the last two selected 2D views.

Group copies Objects are combined into a group after pasting them.

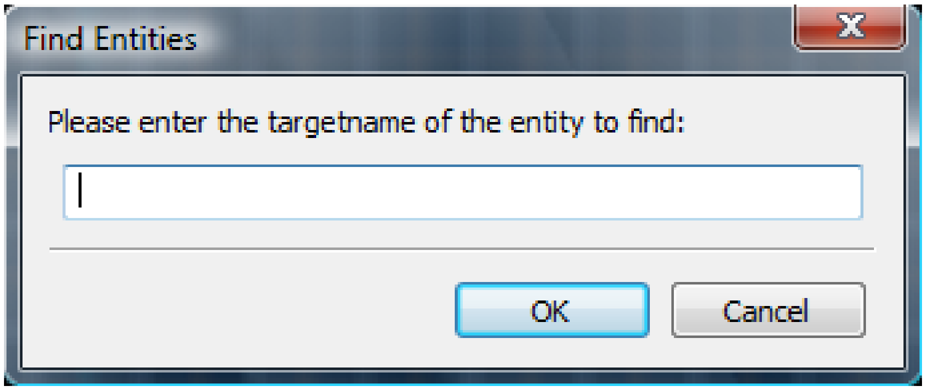
Translation (per copy) Sets the offset in x/y/z direction of each pasted object. This offset is accumulative, so the offset of an object is its own offset and the sum of the offsets off all previous objects.

Rotation (per copy) Sets the rotation offset of each pasted object. This offset is accumulative, so the offset of an object is its own offset and the sum of the offsets off all previous objects.

11.4 The Find Entities Dialog

This dialog can be activated using the related item from *The Edit Menu*.

It is used to find entities by a given name.



Enter the name of the entity

you want to find in the name field.

By pressing **OK** a search is started and the found entities will be selected on the map.

11.5 The Object Properties Dialog

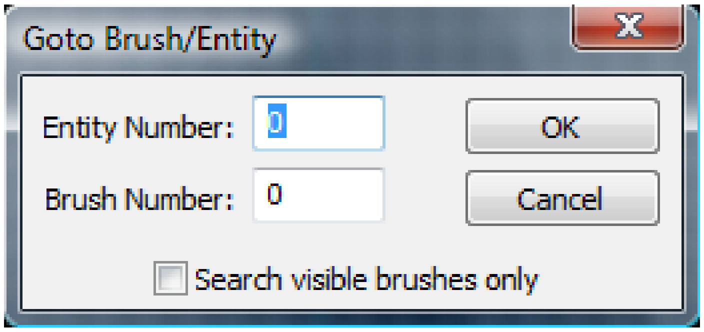
This dialog is obsolete since CaWE release 8.02 (February 2008). If you employ CaWE version 8.02 or newer, please refer to the new *Entity Inspector* dialog instead. The new *Entity Inspector* dialog subsumes all functionality of the Object Properties dialog.

This dialog will be revised and documented later.

11.6 The Goto Brush/Entity Dialog

This dialog can be accessed by clicking on the related item in *The Map Menu*.

If you got an invalid brush or entity on your map, the compiler will tell you the number of the according brush/entity, so you can select it with this tool and it will be marked and centered in the views.



Entity Number: 0

Brush Number: 0

Search visible brushes only

Entity Number Enter the number of the brush you want to find here.

Brush Number Enter the number of the entity you want to find here.

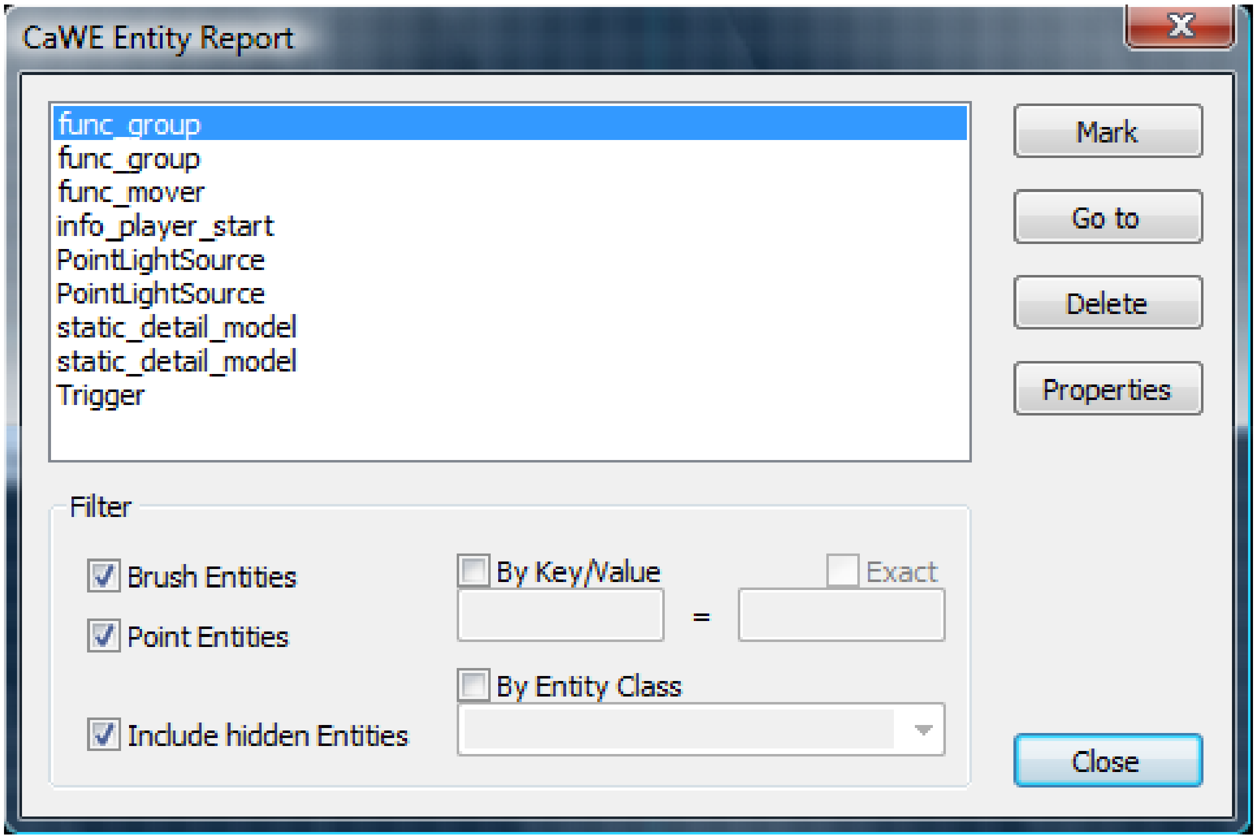
Search visible brushes only Toggles if only visible brushes/entities are considered in the search.

11.7 The Entity Report Dialog

This dialog is obsolete since CaWE release 8.02 (February 2008). If you employ CaWE version 8.02 or newer, please refer to the new *Entity Inspector* dialog instead. The new *Entity Inspector* dialog subsumes all functionality of this Entity Report.

This dialog is accessed by clicking on the related item in the *The Map Menu*.

It shows a list of all entities in this map, that is filterable with different filter options.



Entity list A list of all entities in this map, filtered by the given filter options below.

Mark Marks the currently selected entity in the list on the map.

Go to Centers the views onto the selected entity from the list.

Delete Deletes the selected entity from the list and the map.

Properties Opens the *Properties Dialog* for this entity.

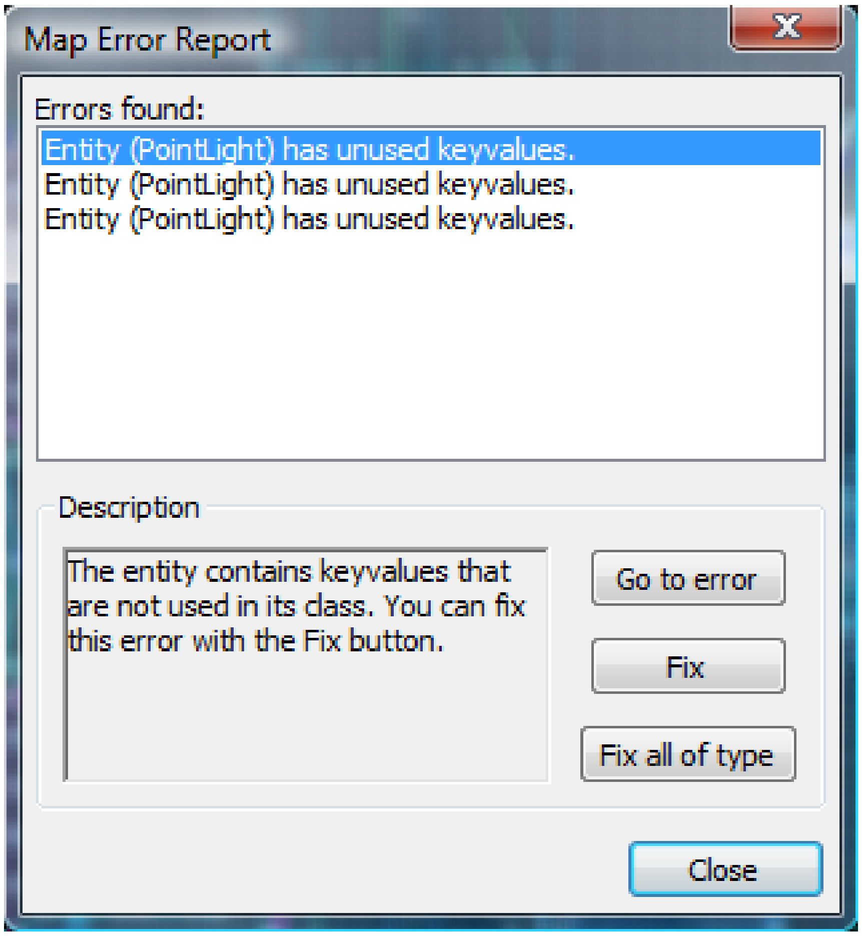
Filter Filter settings.

> **Brush Entities** Show brush entities in list.
>
> **Point Entities** Show point entities in list.
>
> **Include hidden Entities** Show hidden entities in list.
>
> **By Key/Value** Filter the list by a specific key/value pattern in the entity properties.
>
> **Exact** Toggles if the key/value pattern has to be exact the same or if a partial match counts.
>
> **By Entity Class** Filters the list by entities from a specific entity class.

11.8 The Map Error Report Dialog

This dialog is opened by clicking on its related item in *The Map Menu*.

It checks the map for errors and shows them in an error report list.



Entity (PointLight) has unused keyvalues.
Entity (PointLight) has unused keyvalues.
Entity (PointLight) has unused keyvalues.

The entity contains keyvalues that are not used in its class. You can fix this error with the Fix button.

Errors found The list of errors found in the map.

Description Detailed description of the currently selected error.

Go to error Centers the selected error in the views.

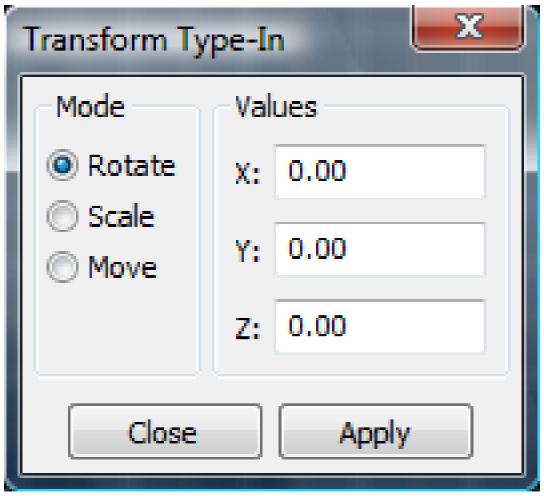
Fix Fixes the selected error.

Fix all of type Fixes all errors of the same type as the selected error.

11.9 The Transform Type-In Dialog

This dialog is activated by clicking on the **Transform** item in *The Tools Menu*.

It allows you to translate (rotate, scale and move) and object by entering exact translation values, instead of translating the object by mouse.



11.9.1 Mode

The transformation mode is set here:

Rotate Object is rotated by the transformation values.

Scale Object is scaled by the transformation values.

Move Object is moved by the transformation values.

11.9.2 Values

X Transformation in x direction.

Y Transformation in y direction.

Z Transformation in z direction.

11.10 The Surface Properties Dialog

This dialog represents the settings of the *Edit Surface Properties Tool*. Please refer to the *Edit Surface Properties Tool* documentation for more information.

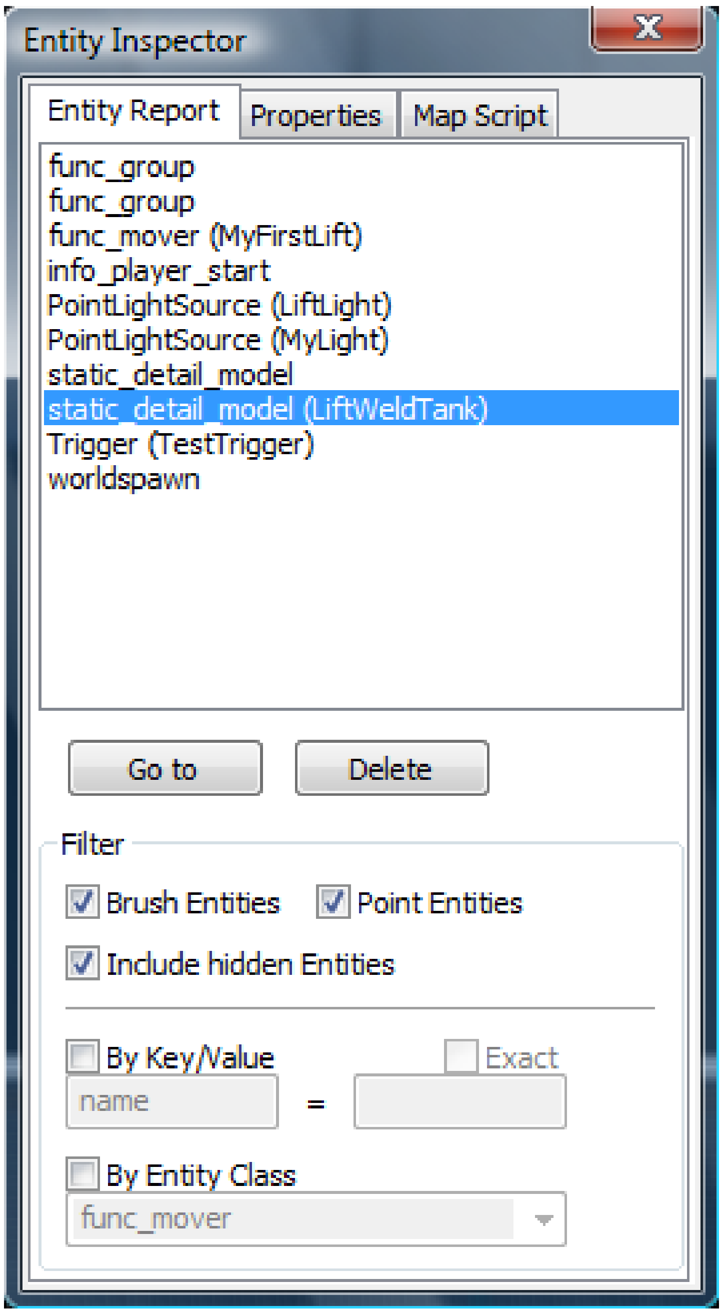
11.11 The Entity Inspector Dialog

This dialog presents detailed information about your map and its entities. It shows you the hierarchically structured list of entities that your map is composed of, allows you to inspect the properties of each entity, and provides a built-in source code editor for the map script.

The dialog is opened by selecting the **Entity Inspector** item from the *Edit* menu.

The following text describes the three "tabs" that implement the main features of the dialog.

11.11.1 Entity Report



func_group
func_group
func_mover (MyFirstLift)
info_player_start
PointLightSource (LiftLight)
PointLightSource (MyLight)
static_detail_model
static_detail_model (LiftWeldTank)
Trigger (TestTrigger)
worldspawn

The Entity Report keeps a list of all entities that exist in the map. It is used to easily manage, find and select individual entities, and thus provides a convenient alternative to locating entities in your map manually.

The entities are listed by the name of their entity class and their instance name, if one has given, e.g. "`static_detail_model (TreasureChest)`". Clicking on a list entry selects the related entity in the 2D and 3D views of the map as well. You can select multiple entities at the same time by holding the `CTRL` or `SHIFT` key while clicking on an entry: this works just like in Windows Explorer. The presented list is sorted in alphabetical order and can be filtered by several options as described below.

Go to Centers the 2D views on the selected entities.

Delete Deletes the selected entities (both from the list and from the map).

Filter The controls in the filter box determine which entities are included in the list above.

Brush Entities When checked, the brush-based entities are included in the list.

Point Entities When checked, the point-based entities are included in the list.

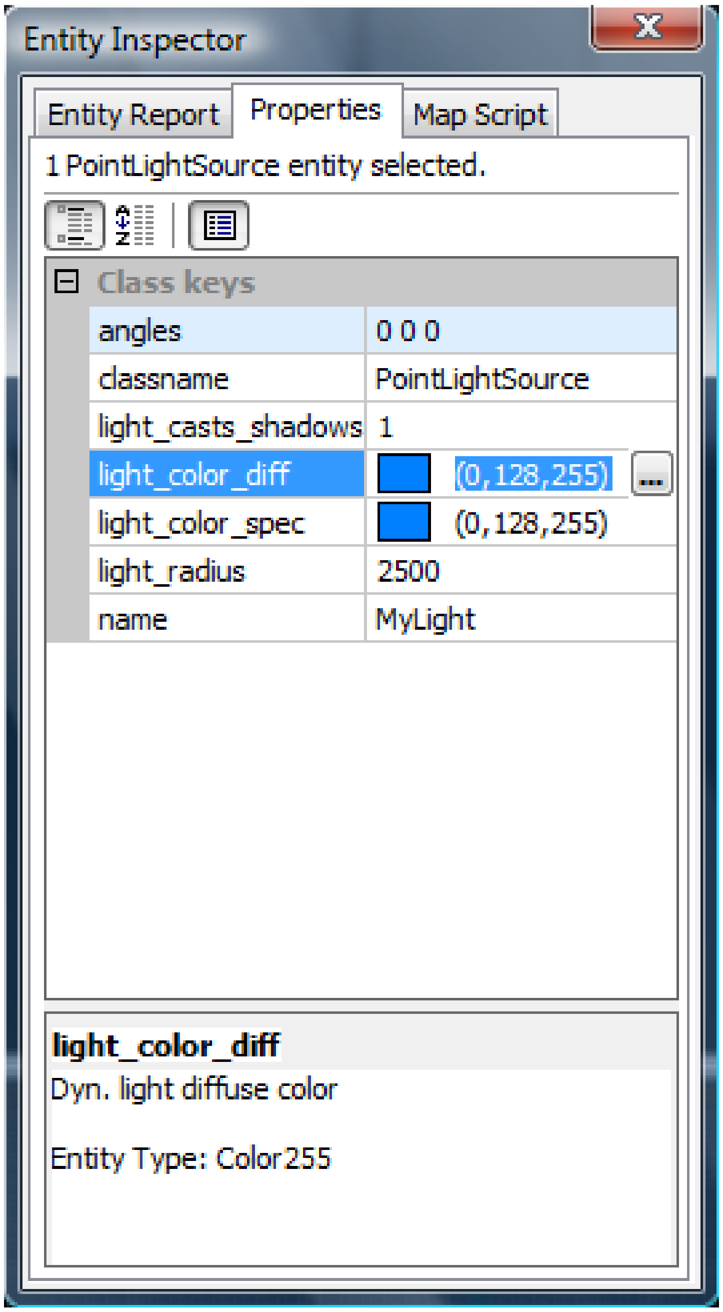
Include hidden Entities When checked, entities are included even though they are normally hidden (as part of a VisGroup) in the 2D and 3D views.

By Key/Value You can reduce the list of entities to those whose properties match the given key/value pattern.

Exact Toggles if the key/value pattern has to be an exact match, or if partial match counts as well.

By Entity Class If checked, only entities of the specified entity class are shown.

11.11.2 Properties



1 PointLightSource entity selected.

Class keys

angles	0 0 0
classname	PointLightSource
light_casts_shadows	1
light_color_diff	🟦 (0,128,255)
light_color_spec	🟦 (0,128,255)
light_radius	2500
name	MyLight

light_color_diff
Dyn. light diffuse color

Entity Type: Color255

The Properties tab is one of the most important dialogs in CaWE, because it allows you to inspect, set and modify all of the advanced attributes of the currently selected entity.

The properties are presented in tabular form as pairs of *keys* (on the left) and their related *values* (on the right): The *set of keys* that are available for an entity is defined by the entity's class. It can therefore not be changed directly within CaWE. Adding, removing or renaming keys is only possible via the game's `EntityClassDefs.lua` script, which contains the definitions of all entity classes for a game. Updating this script possibly affects all maps of a game, and is consequently normally never done for an individual map, but in the early planning stages of a MOD.

Changing the *values* however is the daily bread and butter of a mapper. Therefore, doing so is not only easy, but it has also been made as convenient as possible. In most cases, you just click into the desired value field and enter a new value directly. For each variable (that is, a key/value pair), also the variables *type* is known to CaWE, which is uses to assist you entering correct values:

- keys that only accept integer values do not accept commas, periods or letters,
- keys that accept boolean values or a choice from an enumeration act accordingly,
- keys that expect colors will provide you a color chooser and an automatic preview,
- keys whose values are multi-line strings come with an appropriate editor,
- keys that expect file names (e.g. for sounds, models, etc.) will provide you with a three-dots "…" browse button (which is *very helpful* (**!**)),
- and so on.

We're currently working on improving this feature further, e.g. with "Play" buttons for listening to sound files, a preview facility also for model files, etc.

Note that at the top of the dialog, there is a summary about the number of selected entities and their classes. At the bottom of the dialog, a short help text for the currently selected key is provided. Help is also available in form of tool-tips for the key the mouse pointer is currently over.

Classname

Changing the class of an entity is a special action, because it corresponds to deleting the entity and re-instantiating it under a different class but with the same properties. Nonetheless, you can use the special `classname` property to change an entity's class like any other property: You're provided with a choice box that contains all possible new classes for that entity, and just making the selection reincarnates the entity as an object of the new class.

Please note that currently there is a restriction that brush-based (solid) entities can only be converted into entity's of classes that are brush-based, too. Similarly, entities that are point-based can only be converted into entity's that are point-based, too.

Undefined Properties

Sometimes entities come with key/value pairs that have no corresponding definition in their entity class. Such key/value pairs can occur occasionally e.g. when an entity was converted to a different class (see below), an entity class got a variable definition removed after a map has already been made with it, or when an entity has been imported from an entirely different game. The entity might then bring in a key named e.g. `light_pulse_frq` that is utterly unknown in the context of the current games entity definitions.

Another but similar case occurs when the entity's class name is unknown among all entity classes of the current game. This typically happens when you try to import for example an entity of class "stuffed_animal" of a children's game into a game about aliens or robots, or in any other case when the current game just knows nothing about the imported entity class.

In these cases, there is no corresponding definition for such keys in the game definition. CaWE therefore collects such occurrences separately and keeps them under the special category "Undefined Properties".

Because undefined properties are not dealt with by the game in any way, you're free to change and modify them at will. You can even right-click them to open a context menu that allows you to rename them, to add new ones or to delete them.

In most practical cases, you'll want to get rid of undefined keys though, which can be achieved by renaming them to something meaningful, by deleting them, by changing the entire entity class to something else, or a combination thereof.

Background Colors

The property grid employs several background colors in order to indicate certain conditions of the values:

- White background indicates that this is just a normal value.
- Light blue background also indicates a normal value that also happens to be the entity class's default value for this variable.
- Light orange background is used whenever multiple entities are selected that have *different* values for the *same* key. You can still enter a new value for such keys: The key of each selected entity will then be set to the same new value and the background color will change back to white accordingly.
- Light red background is currently used at only one occasion: When you set a `model` key and have not yet specified a value for a `collisionModel`, the `collisionModel` variable is temporarily highlighted with this color in order to remind you to also set a collision model.

Working with multiple entities

The Properties tab works also very well when *more* than a single entity is selected, or in fact, any number of them is selected at the same time. In order to present you the properties of multiple entities, CaWE "overlays" them for you, so that even great amounts of simultaneously selected entities remain easy to understand and manage.

In order to achieve this, CaWE makes sure that properties that are candidates for layering, but otherwise incompatible with each other for any reason (e.g. because the property is known only to some but not all of the entities, or defined multiply with different types), are treated specially as "mixed" properties. A property that is marked as "mixed" has attributes in one entity that are non-existent or different in another, which generally makes them incompatible to each other.

Such mixed properties are therefore listed in another separate category, and they are protected from editing ("read-only"). This makes sure that you can see such affected properties, but not inadvertently get nonsensical values written into lots of entities. Note that this is not a restriction, because it's just another way of saying: *"Stop! If you assign a value to this property, there is at least one entity affected for which doing this makes no sense at all."*

If you still want to edit a property that is listed as "mixed", there is an easy way to overcome the problem: Just select fewer entities until the entity that was the reason for the property being listed as "mixed" is gone. When no conflicts remain, the property gets re-listed among the normal properties, and you can edit it freely.

11.11.3 Map Script

In the Map Script tab, you can view and edit the source code of the maps current script.

(FIXME!) This feature is not yet implemented – I'm very sorry. Both the functionality and the documentation will be provided later. However, please note that the game code and the Cafu engine already fully implement and support map scripting! Use any text editor of your choice in order to program your map scripts independently and outside of CaWE. In fact, all the example scripts that come with the Cafu releases have been written that way.

CHAPTER 12

Entity Guide

An entity is an object that has characteristics which are different to the normal world brushes. There are two kinds of entities in Cafu. These include brush-based entities and point-based entities.

Point-based entities exist at a specific coordinate in space and be can be thought of as a reference point. Although these entities may refer to objects that actually have a volume, for example a monster, they are only treated as points in CaWE, and the Cafu engine provides the volume later in the game. An example is a monster entity: In CaWE, it just represents the point where the monster will be, in Cafu, it will actually be there.

Brush-based entities are related to at least one brush or bezier patch and thus have a volume. Examples for brush based entities include doors and platforms, bodies of water, area triggers, etc.

12.1 Brush-Based Entities

Solid entities are world brushes that have extra characteristics. Note: This list is not fully complete.

12.1.1 func_illusionary

This is a func_wall with no clipping. The player can walk through a func_illusionary, making it useful for secret paths.

12.1.2 func_water

This is the entity for water.

12.1.3 func_ladder

The player can climb up a func_ladder. Please note that this entity is invisible, so you need to place a textured brush **behind** it.

12.1.4 func_breakable

This is a breakable object. You can set the material as glass, wood, metal, flesh, cinder block, ceiling tile, computer, unbreakable glass or rocks. You can also set the strength of the object.

12.1.5 func_particle_generator

This entity emits particles. It could be used for fire, water etc.

12.1.6 func_pushable

This is an object that the player can push around the map.

12.1.7 func_terrain

This is the entity used to add a terrain to a world. You must specify a heightmap for the terrain, and a detail texture. See *Creating heightmaps for your terrains* and the NewMaterials document for more info about terrains.

12.2 Point Entities

12.2.1 info_player_start

The info_player_start is the spawn point for the character(s). You must have at least one in any map you create, and you must ensure that **every** info_player_start is inside the map.

12.2.2 PointLight

This is a light to complement the texture area lights. You used to have to put one of these in front of every surface that emitted light, but now you should not need to use this entity much, as any texture that looks like a light will emit light in the game.

12.2.3 static_detail_model

Note: This entity is specific to the Deathmatch mod.

This is a model that is included in a map, usually just for visual effect. You can specify an external model, how big it should be (scale), and if applicable, which animated sequence it should be performing. This entity can be used for anything that would be difficult to model out of brushes in CaWE.

Please note that this entity doesn't clip, so you need to put a clip brush around the model if you don't want the player to be able to walk through it.

12.2.4 Weapons, Ammo and items

Note: The following entities are specific to the Deathmatch mod.
These entities are the weapons, ammo and items in the deathmatch mod. These all start with weapon_, ammo_, or item_. Some of the operational weapons included in the deathmatch mod are:

- weapon_shotgun (Shotgun)
- weapon_357handgun (357 Handgun)
- weapon_9mmAR (Assult Rifle)
- weapon_crossbow (Crossbow)
- weapon_rpg (RPG Launcher)
- weapon_handgrenade (Grenade)

Some other weapons are included, but most of them don't work.

12.2.5 Monsters

Note: The following entities are specific to the Deathmatch mod.
These are basically NPCs (Non-Player Characters). They can be animals, allies or enemies. The deathmatch mod currently contains three types of monsters:

- monster_butterfly - A group of three butterflies that fly around in circles.
- monster_companybot - The default Trinity bot that follows you around.
- monster_eagle - An eagle which flies overhead. Requires a lot of space.

12.2.6 Tree

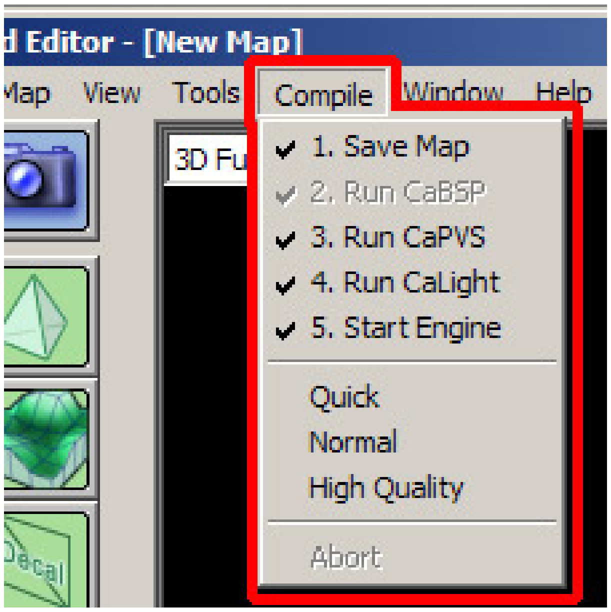
(FIXME!) I don't know what this is…

CHAPTER 13

Compiling Maps for Cafu

When you have finished editing a map in CaWE, you'll want to run it with Cafu. The engine however requires that the map is in a precomputed form, and this section describes how you turn your map into a fully precomputed world file. The process of turning your **cmap map file** into a for the engine usable **cw world file** is called *compiling*.

As to not force you to run the compile procedure for all your cmap files *by hand*, the CaWE mapping editor provides a "Compile" menu with which you can easily and very conveniently accomplish this task. In this section, you'll learn how to use it.



You will find the "Compile" menu in the top menu bar of CaWE. It's items can roughly be separated into three distinct groups: The upper group tells CaWE *what* to do, the middle group says *how* to do it and the lower group is for *stopping* the compile process early. The function of all items is explained in greater detail below:

13.1 Step 1: The compile steps

The upper group of menu items tells CaWE which steps of the compile process should be run. You can turn the individual parts on or off by checking or unchecking the mark in the appropriate line.

1. **Save Map:** Saves the map. Checking this does exactly the same as a click on the "File → Save" menu item before compiling the map. Having this checked is practical and recommended when you work on a map and want to test it "live" in the Cafu engine every now and then, because it makes sure that the compilers pick up the latest version of your map with the most recent changes.
2. **Run CaBSP:** This is the crucial step that runs the CaBSP compiler on your map. CaBSP takes your map file and creates a world file for use by the Cafu engine from it (the map file is not modified). As all other compilation steps build on this one and the generated cw file, you cannot turn it off to omit CaBSP.
3. **Run CaPVS:** Runs the CaPVS compiler on the intermediate result obtained from CaBSP. CaPVS precomputes the PVS (Potentially Visibility Set) for the world file. This can be a lengthy process on its own, and you may wish to skip it when you're just testing your map. However, it also has a potential for improving the rendering performance of your map significantly, so you should never leave this out for release maps.
4. **Run CaLight:** Runs the CaLight compiler on your map, which precomputes the static lighting of your maps by using physically accurate Radiosity methods. CaLight too can take very much time to complete, and should therefore never be run without a preceding CaPVS pass. However, for testing you may want to skip CaLight as well as CaPVS, because your map will be full-bright when CaLight is not run, which may be sufficient for checking e.g. the geometry or entity placement during map development.
5. **Start Engine:** If this entry is checked, Cafu will automatically be launched with your map when the compiling process is finished.

13.2 Step 2: Starting the compile process

After you have selected which steps should be included in the compile process, you can start it with varying quality settings by clicking on one of the three items of the middle group ("Quick", "Normal", or "High Quality"). These settings mostly affect the quality of the lighting generated by CaLight, and therefore have a direct impact on its run-time.

After you started the compile process, a console window opens, showing compile details and progress information. Note that you can simply close the console window at any time and continue to work on your map while the compilation process keeps running in the background. Just use the "View → Screen Elements → Console" menu to toggle the console window on and off. The three menu items to start a compile will be grayed-out while another compilation is running, so that starting two compile processes simultaneously is not possible.

However, you can stop the entire currently running compilation process prematurely by clicking on the "Compile → Abort" menu item.

If you activated the "Start Engine" item, Cafu will be started when the compile process is complete and you'll just have to click on "OK" in the Cafu dialog to start and play your map!

13.3 Tips, Tricks and Trouble-shooting

Using the "Compile" menu requires that you have setup the "Configuration" dialog of CaWE completely and properly. If a compiler cannot be run correctly, please check if you got all the executable file names right in this dialog. Please refer to *Installation, Initial Configuration and De-Installation* for more details.

If your map turns out to be pitch-black when run in the engine after the last compilation step, a likely cause is that you kept the CaLight compiler enabled, but did not place any radiosity light source (i.e. no light or sky material). For checking this quickly, just disable CaLight in the "Compile" menu, then start the compilation again and review the result in the Cafu engine. Thereafter, you may want to place some light sources into your map by applying materials that cast radiosity light, and re-enable CaLight again in order to compute a proper, realistic radiosity solution.

For quick tests during map development, you can save a lot of compile time by turning off the CaPVS and CaLight compile steps. You may get very low frame-rates and only full-bright lighting, but that's usually fine for checking the map geometry. When you're done, or about to release your map for others, you definitively should compile it once with CaPVS and CaLight enabled, in order to obtain the maximum speed and quality.

Very advanced users can also compile their maps manually at the command-line. This offers the possibility for further customisations of the compile process. Compiling maps at the command-line is explained *here*.

CHAPTER 14

Compiling Maps at the Command-Line

When you have finished editing a map in CaWE, you'll want to run it with Cafu. The engine however requires that the map is in a precomputed form, and this section describes how you turn your map into a fully precomputed world file.

In order to use CaBSP, CaPVS and CaLight "by hand" (rather than *from within CaWE*), you should be familiar with the command-line interface of Windows or Linux, because these programs just cannot be started by a simple double-click on them.

Under Windows, you can open the command-line window by opening "Programs → Accessories → Command Prompt" (in German: "Programme → Zubehör → Eingabeaufforderung") from the Start menu. Initially, the window shows the directory you are currently in. You can browse into the parent directory by typing `cd ..`, as for example in

```
C:\Windows> cd ..
C:\>
```

and you can enter a directory by typing

```
C:\> cd directoryname
C:\directoryname>
```

(Replace "directoryname" with the name of the directory that you want to open.) Other useful commands are `dir` that shows all files that are in the directory you are currently in and `e:` (for example) to change the current hard drive.

In order to run the programs with the commands presented below, you'll first have to browse into the directory they are in (which is the `Cafu-9.06` directory).

14.1 Compiling step 1: CaBSP

This program takes a cmap map file and creates a cw (Cafu world) file from it. Example:

```
C:\Cafu-9.06> CaBSP Games/DeathMatch/Maps/MyMap.cmap Games/DeathMatch/Worlds/MyMap.cw
```

(!) WARNING: The specified destination file (`Games/DeathMatch/Worlds/MyMap.cw` in the above example) gets overwritten without prior notice! This is also true for mis-spelt file names. For example, if you type `Games/DeathMatch/Maps/MyMap.cmap` instead of `Games/DeathMatch/Worlds/MyMap.cw` – catastrophe! As with most other computer software, in order to prevent a disaster, please *backup your files***before***you begin*!

For quick tests during world development, you might want to skip the next two compiling steps (CaPVS and CaLight). You can run the file obtained from the CaBSP tool directly in the Cafu engine. It will be unlit and without PVS information (which results in a lower frame rate), but is usually sufficient for early tests.

14.2 Compiling step 2: CaPVS

This program creates the *Potentially Visibility Set* for your worlds. Example:

```
C:\Cafu-9.06> CaPVS Games/DeathMatch/Worlds/MyMap.cw
```

14.2.1 The CaPVS compile switches

Even though the latest versions of CaPVS have become fast enough to deal with even the most complex worlds within reasonable time bounds, sometimes it may still be preferable to obtain faster results (at the cost of accuracy).

Before you read on, though, my advice is to simply skip the rest of this section, and continue with the next. Why? Well, the compile switches introduced below were made at a time when CaPVS was inherently slower. Since then, it has undergone many algorithmic improvements, and the switches were only left for safety and as a last resort. I hardly ever use them myself (although their implementation is interesting), and nowadays never use them for release compiles (the worst world I have takes 27 hours on a 866 MHz Pentium III this way). During world development, you should skip CaPVS entirely rather than trying to tune it for faster compilation. Also keep in mind that the proper use of `func_detail_object` entities can have a big impact on performance in general, and on CaPVS compile time in particular.

Anyway, the latest version of this tool has two new command line switches that were made to lower its compile times by effectively trading quality for speed. Both switches control the creation of *SuperLeaves*. To understand what SuperLeaves are, you must first have a rough idea what *leaves* are: Leaves are spatial convex cells into which CaBSP subdivides your world and in which the action of the game takes place later on. Simplified spoken, you can imagine that each room of your world corresponds to a leaf. CaPVS then determines the PVS by calculating if any unobstructed lines of sight exist between any pair of two leaves. It is not only the sheer number of leaves that makes the computation so slow, but also their spatial relations among each other. In some cases, there are unnecessarily too many leaves for the purposes of CaPVS, and it is preferable to merge some of them. Thus, a *SuperLeaf* is a set or a group of simple leaves. One could say that SuperLeaves subdivide the world in cells as leaves do, but the subdivision is "coarser" than with the simple leaves. When you run CaPVS on a world, it tells you how many SuperLeaves it constructed from the simple leaves. By default (without command line switches), the SuperLeaves are identical to the simple leaves, as is their number.

Now about the actual switches: Running CaPVS without any parameters (not even a world name) will print out a self-explaining usage information. I'll not repeat it here, but rather give you some tips on the usage of the two mentioned switches. Note that you always trade quality for speed, but the compromise is usually a very good one (quality remains high even for quite fast compiles).

In general, try to reduce the compile time with care. If the situation was actually hopeless before, it does not make sense to use the switches to bring the compile time down to a few minutes or seconds. (That would be more like not running CaPVS at all.) Rather, a good idea would be to send me an email with a brief description of the problem, along with the world you are trying to compile (I'll only believe that you created a world that's too hard for CaPVS after I've seen it myself).

Then, consider two strategies: You can either choose to work "top-down" by choosing the parameters such that the number of SuperLeaves is gradually reduced in small steps. The quality remains high, but the downside is that you possibly have to wait another long time until you know if the reduction was enough.

Thus, I'd recommend a "bottom-up" approach: Initially reduce the number of SuperLeaves a lot, start with a quite small number. If CaPVS then takes only a few minutes, but you want it rather to spend some hours, adjust the parameter values carefully so that the number of SuperLeaves increases slowly and gradually. You'll probably have to re-try and experiment a few times until you get a feel of it.

Finally, you should prefer the `-minAreaSL` switch over the `-maxRecDepthSL` switch, because the former is more sensitive to the actual world geometry. Do not worry about high numbers on `-minAreaSL`. When I last used it, I usually had values ranging from 1,000,000 to 5,000,000.

14.3 Compiling step 3: CaLight

This program calculates the lighting for a map. This program can take very long too, but you have better control over it by specifying command line switches. Examples:

```
C:\Cafu-9.06> CaLight
C:\Cafu-9.06> CaLight Games/DeathMatch/Worlds/MyMap.cw
C:\Cafu-9.06> CaLight Games/DeathMatch/Worlds/MyMap.cw -BlockSize 8 -StopUE 2
C:\Cafu-9.06> CaLight Games/DeathMatch/Worlds/MyMap.cw -fast
```

The first line prints out detailed usage information for the CaLight tool, the second line runs default lighting, which is recommended for most cases. The third and fourth lines are for very fast lighting, which is useful for quick tests during map development.

Please note that it is almost never reasonable to set BlockSize below 3 and StopUE below 0.1 for highest quality lighting, even though that is possible (some of the worlds of the Cafu demo releases were compiled with a StopUE of 0.05).

During the second phase of the computations ("bounce lighting"), note that you can interrupt the program by pressing the `SPACE` button. That's sometimes useful for quick tests during map development.

Finally, a word about memory consumption: CaLight requires *plenty* of it! I would recommend to run CaLight only on computers with at least 1 GB of RAM, it may or may not run with less. If *permanent swapping* (extensive disk activity) occurs during the bounce lighting phase of CaLight, better abort it! Under such circumstances, it will proceed *very* slow anyway, and the lengthy swap activity is certainly not healthy for your hard-disk. **(!)**

14.4 Done: Running the world with Cafu

If not already there, copy your world into the `Games/DeathMatch/Worlds` directory if you made the world for the DeathMatch MOD, or into the `Games/OtherMOD/Worlds` directory of the MOD you made the world for. Then simply run `Cafu.exe` as stated in the user manual of the current demo release. Your world will be listed among the other worlds in the dialogs world list ("server options").

CHAPTER 15

Creating Height-Maps for your Terrains

The shapes of the terrains that you can create with *The New Terrain Tool* are defined by *height-maps*. A height-map is usually a square gray-scale image whose black pixels represent the lowest heights (0%) and whose fully white pixels describe the highest heights (100%). The gray values between them represent intermediate height values – the brighter a pixel, the higher is the resulting terrain at that spot.

CaWE and Cafu support height-maps in all the image file formats that are also supported for textures (`jpg`, `png`, `tga` and `bmp`), as well as Terragen (`ter`) and portable graymap (`pgm`) files. The image file formats usually have a precision of 8 BPP (bits-per-pixel), while the `ter` and `pgm` formats have a much higher precision of 16 BPP, and are therefore the preferred choice for height-maps.

Height-maps must also meet the additional requirement of having side lengths of (2:sup:*n*+1) * (2:sup:*n*+1) pixels, where n is an integer between 1 and 15. Examples of typical height-map dimensions are 129*129, 257*257, 513*513 and 1025*1025.

15.1 Programs for Creating and Editing Height-Maps

Creating and editing high-quality height-maps can be a complicated and demanding process that typically requires the assistance of specialized software. I've intensively tested many software packages for this purpose, many of which just don't meet the requirements. Typical problems include limited or broken importers and exporters (e.g. only to and from proprietary file formats), limits in the supported high-map dimensions (e.g. only $2^n * 2^n$, not $2^n+1 * 2^n+1$), usability, stability (e.g. frequent crashes) and the inability to switch off the "relative height editing mode", which seemingly allows users to dig deeper than 0% or raise terrain higher than 100% by adjusting the relative height of all *other* terrain.

The following lists mentions several programs that I consider useful or relevant for creating and editing height-maps. None of them is perfect or can do everything that is needed all in one program, so you might be also interested in reading the [**(FIXME!)** Link to tutorial about how the TechDemo terrain was made.] tutorial.

15.1.1 CaWE

The obvious and best choice for editing the height-maps of terrain entities is CaWE.

It is possible to edit and modify height-maps directly in the 3D view of CaWE, in a "live" manner that naturally and immediately shows the interactions with other elements of the map, as for example brush-based buildings and walls, other entities, etc. For more information about the terrain tool please check *here*

Note that although CaWE would be the ideal place for height-map *editing and modification*, it is not necessarily the best place for height-map *creation*. Although that would be and will be possible, too, other programs below that are specialized on height-map creation may do a much better job in creating the initial height-map, especially if very large, natural-looking shapes are desired. In such cases, you would use another program to create the initial height-map, then add it to your map in "rough" form using *The New Terrain Tool*, and then continue to edit the fine details in CaWE, for example the alignment with buildings, other brushes, etc. You may want to have a look at the [**(FIXME!)** Link to tutorial about how the TechDemo terrain was made.] tutorial in this regard.

15.1.2 Regular 2D Applications

Although not meant to visualize the actual 3d surface itself, nearly all 2d applications are able to provide a decent way of creating a highmap. However most of these methods are more likely to create a basic uneven surface that can then be altered inside the Cafu Editor for more precise interaction with bsp or mesh based geometry.

While most terrain specific tools offer much more natural control over the final result (like eroding based on water and wind) almost every 2d application can provide hightmaps.

Noise filter (like perlin noise) are very powerful and can create a wide variety of surface distortions that are great as a base. There are many different applications out there commercial and free, yet we would like to prefer free ones.

Gimp (Freeware)

Obviously the most common freeware 2d application, that mimics the commercial *Adobe Photoshop* in many ways and beyond. It matured over the years into a reliable 2d package and should be considered as a professional tool. By using filter and custom brushes it is a very good way to start a decent highmap

Gimp is loacted here: http://www.gimp.org/

Create basic landscape http://springrts.com/wiki/Height_Map_Tutorial

Flash based (Freeware)

Despite the negative critiqe of the last months, Flash is still a powerhorse. There are Flash apps amazingly versatile for editing, and they are always available as long as there is a web connection

Sumopaint offers a lot of tools and filter to create highmaps http://www.sumopaint.com/app/
Pixlr offers a similiar experience like sumo http://pixlr.com/
SplashUp another flash based 2d app http://www.splashup.com/

15.1.3 L3DT (Free/Commercial)

Under developement for many years, **L**arge **3D T**errain is an app made for game devs. The basic version is free but there is also a Pro version that costs about 34$ for an indie license.
http://www.bundysoft.com/L3DT/

15.1.4 Geo Control 2 (Demo/Commercial)

Quite new and very powerful in all its features, Geo Control offers a 31 day demo with all features or can be purchased for 129€.
It offers many realistic features like erosion, isoline control for artifical leveling and a nice gradient based texturing system. There is also a version 1 available for a lower price. http://www.geocontrol2.com

15.1.5 World-Machine (Free Edition/Commercial)

The free edition is limited to 513*513 pixel but offers all tools. The next basic version costs 89$, can go beyond 8k resolution for maps, which also authorizes for comercial use.
This program has become my favourite for height-map *creation*, see the manufacturers website at http://www.world-machine.com/ for more details. I also used it to create the height-map for the Cafu TechDemo with it. While it does not allow to "paint-edit" very fine details in a height-map, it is very good at composing logically structured terrains in a repeatable manner. The features of all logical devices that are used to build a height-map can be edited independent of each other, allowing for very flexible, reproducible results.

15.1.6 Leveller (Commercial)

Looks like it offers quite a few different tools to work with, however the price of 199 dollar is way too much and there is not much indication about when the last update actually took place. http://www.daylongraphics.com/products/leveller/

15.1.7 Terragen (Free Edition/Commercial)

This is *the* software for terrain rendering, but due to it's "relative height editing mode" I'd not recommend it for height-map creation or editing. Terragen *is* very good though at generating a base (ground) texture from a height-map! http://www.planetside.co.uk/

15.1.8 Earth Sculptor (Commercial/outdated)

A small app, that works fine and allows for decent manipulation. Unfortunatly it's development stopped since 2008, but still its worth noticing it http://www.earthsculptor.com/index.htm

15.1.9 Other 3D Engines

There are a few games with a great support for terrain editing, such as Crysis or Unreal3. Epic offers the Unreal Developer Kit as a free alternative, and the terrain editor is capable of exporting. This may sound strange but still offers another option which is also completly free.

15.1.10 Other Programs

If you have experience with other programs, please add them here.

There are several other good programs, but I don't use them often. Maybe one of them has released a new version in the meanwhile that addresses some of the problems of older versions, so it might be worthwhile to have a look at them! For example, for basic terrain creation, programs like TerraBrush, Terraformer 1.8b or TerraMaker already do a good job.

15.2 Summary

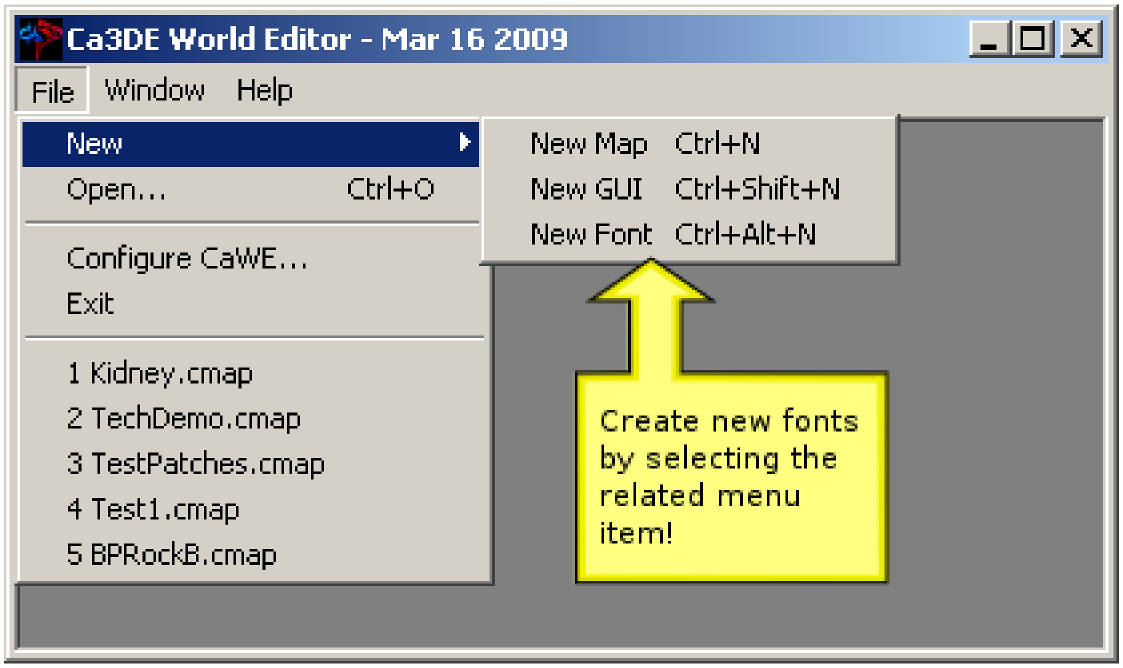
Use any of the above mentioned programs for creating the initial height-map. However CaWE will allow you to further edit these maps conveniently in the editor. If you must fine-tune them before that, patience and creativity is required, although the above programs will help with that, too.

Once your height-map is complete enough for use, you'll also want to have a base texture for it that represents the color of the ground and is used in the material definition with which your terrain is rendered. While the texturing of the terrain might be possible in future versions directly in CaWE, too, I currently just import my height-maps into Terragen for this purpose, make sure the dimensions (lateral size and altitudes) are correct, setup the Terragen materials, set the sun appropriately (full-bright and no shadows) and turn off all atmospheric effects, set an orthogonal camera view and then use the Terragen rendering result as the base texture.

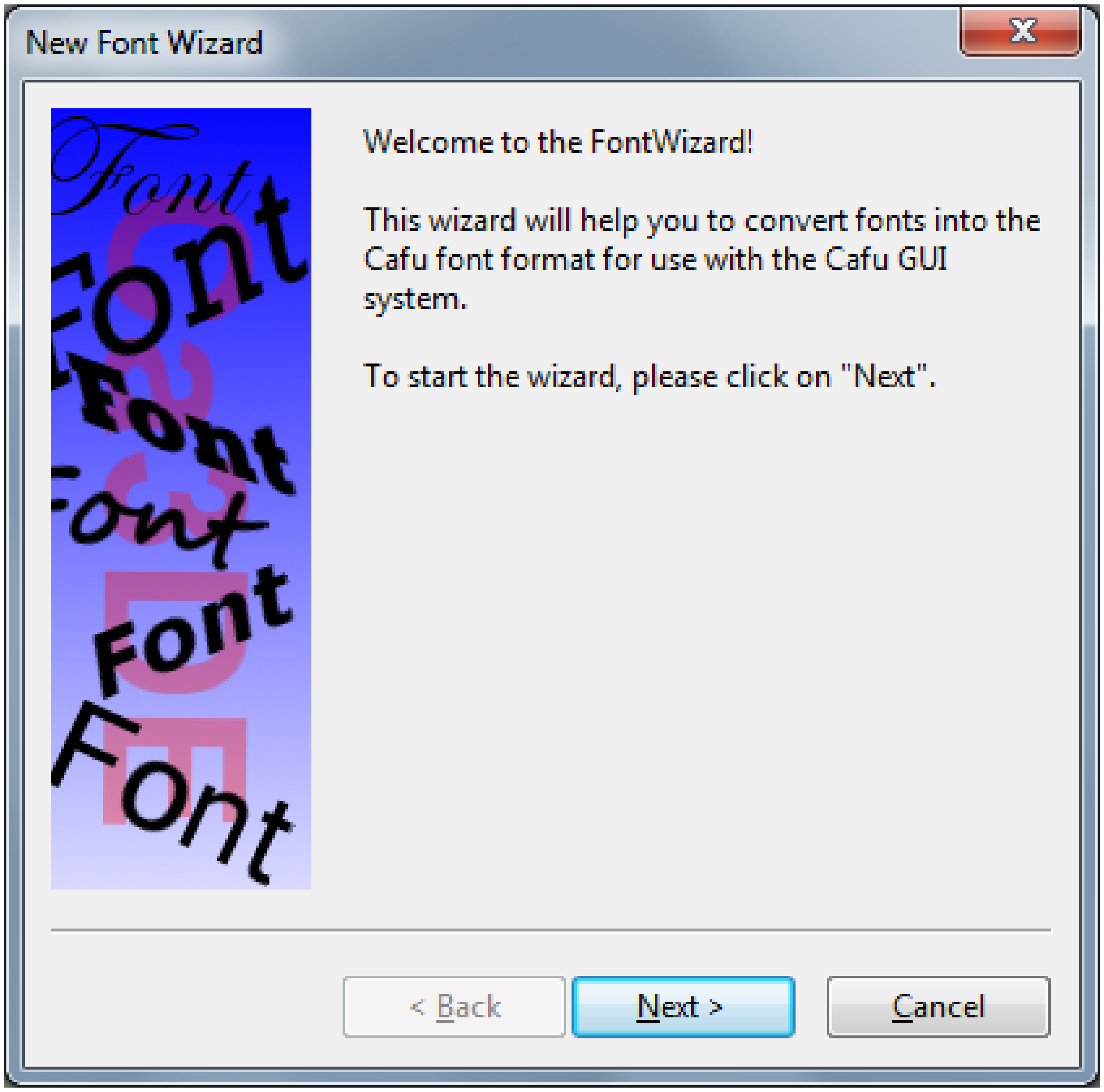
CHAPTER 16

The Font Wizard: Creating new fonts

Preparing new fonts for rendering text within a GUI is very easy. The key idea is to convert a font file (e.g. one of those installed with your OS, or downloaded from the Internet) to a set of graphics files that the Cafu GuiSys can use:



Start CaWE and select the related menu item to open the Font Wizard.

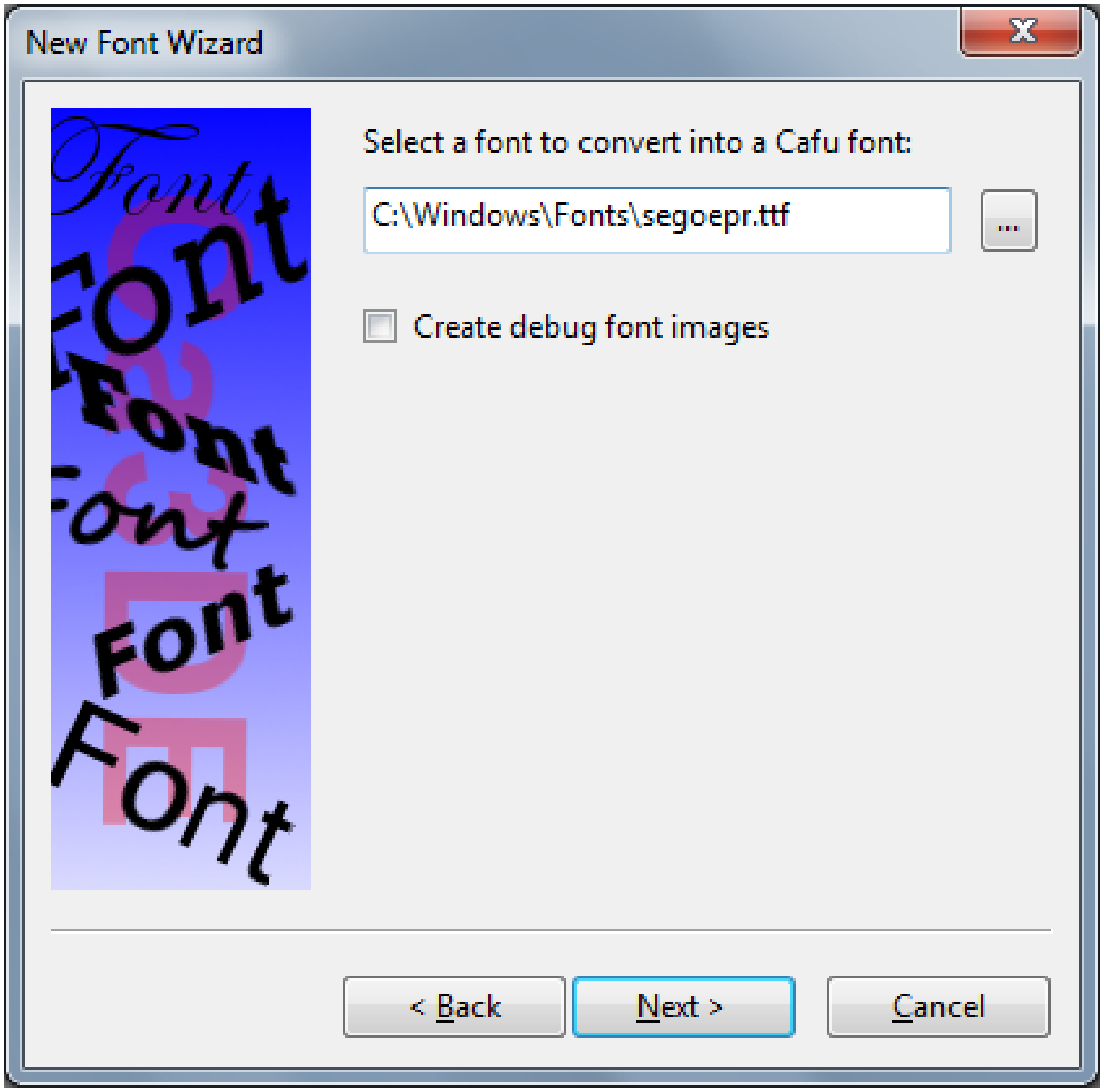


Welcome to the FontWizard!

This wizard will help you to convert fonts into the Cafu font format for use with the Cafu GUI system.

To start the wizard, please click on "Next".

Step 1: The welcome page.

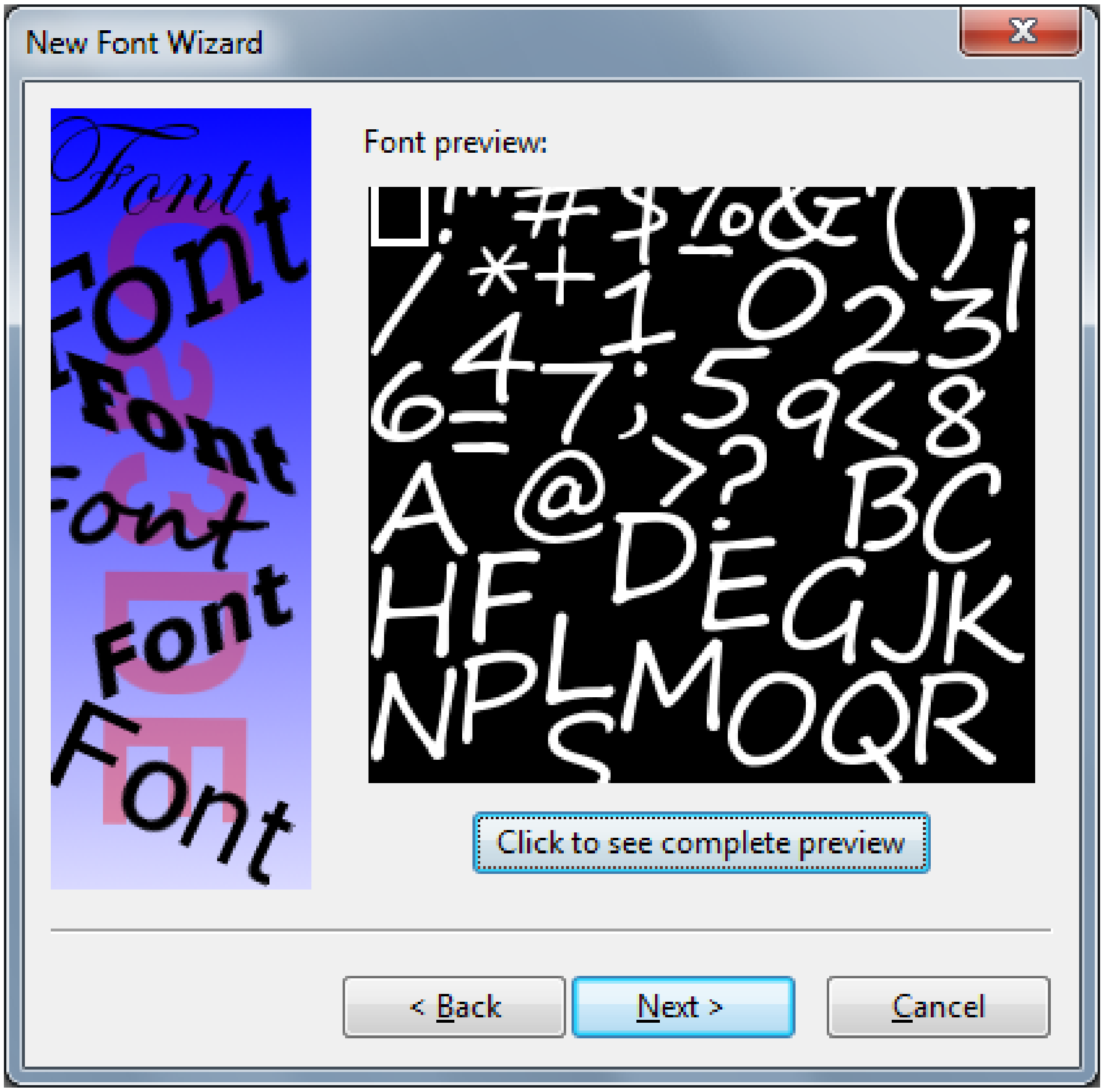

New Font Wizard
Select a font to convert into a Cafu font:
C:\Windows\Fonts\segoepr.ttf
...
Create debug font images
< Back
Next >
Cancel

Step 2: Click the "..." browse button to select the desired font, or enter the filename directly into the related field.

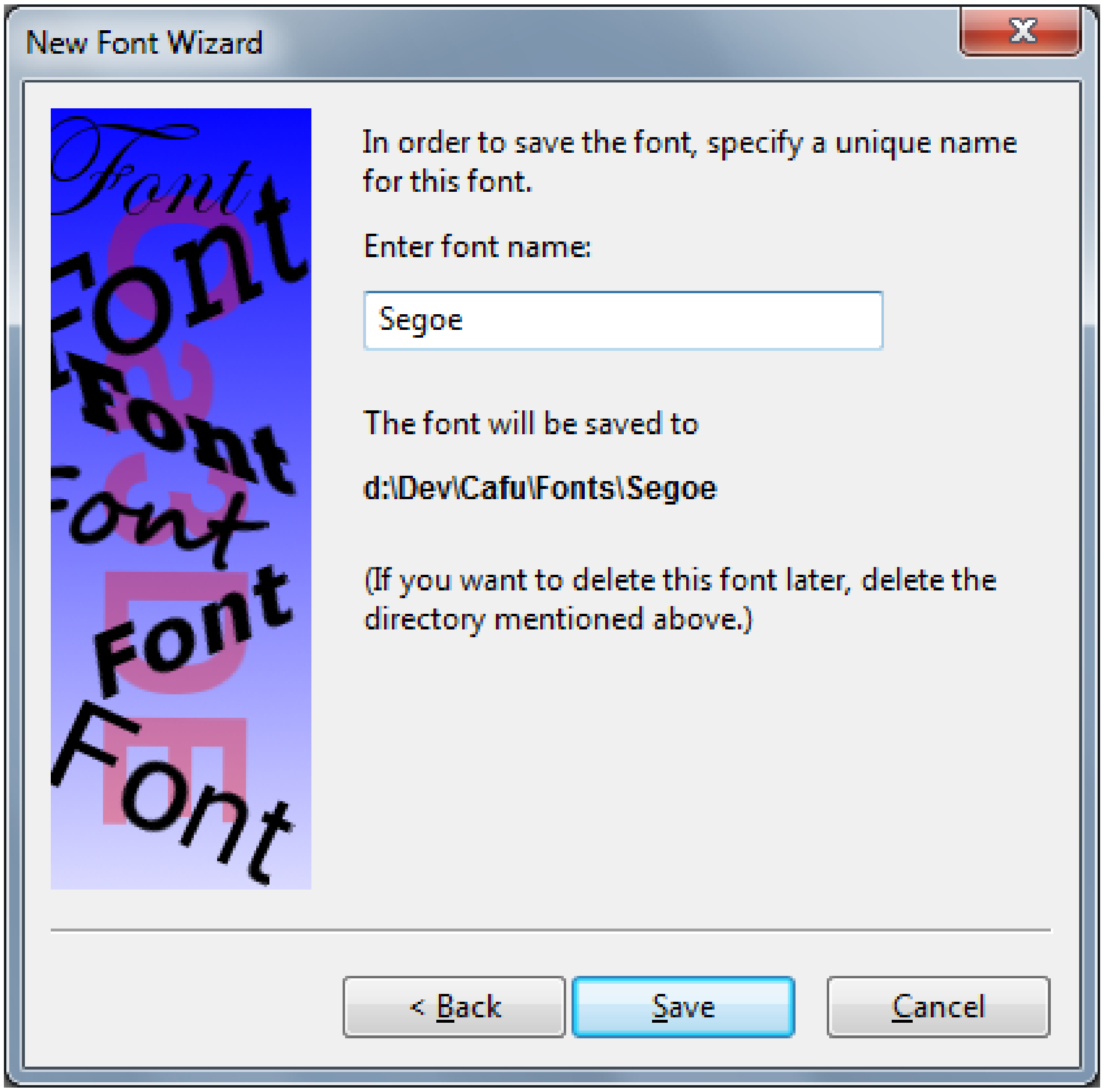
Microsoft Windows users, please note that for some versions of Windows, most notably **Windows 7**, Microsoft has chosen to special-case the related "File Open" dialog so that the system folder `C:\Windows\Fonts` cannot be browsed. (The problem is neither related to access rights nor is it specific to CaWE.)

In order to overcome the problem, i.e. if you want to use a system font from `C:\Windows\Fonts` with the Font Wizard, we recommend to either

- type the path and file name of the desired font manually into the input field, or
- at the command prompt copy the desired font files from `C:\Windows\Fonts` into any other directory, then use the "..." browse button to select the related file.



Step 3: Check the preview of the generated font texture. Optionally click the button to see a preview of all generated texture images.



In order to save the font, specify a unique name for this font.

Enter font name:

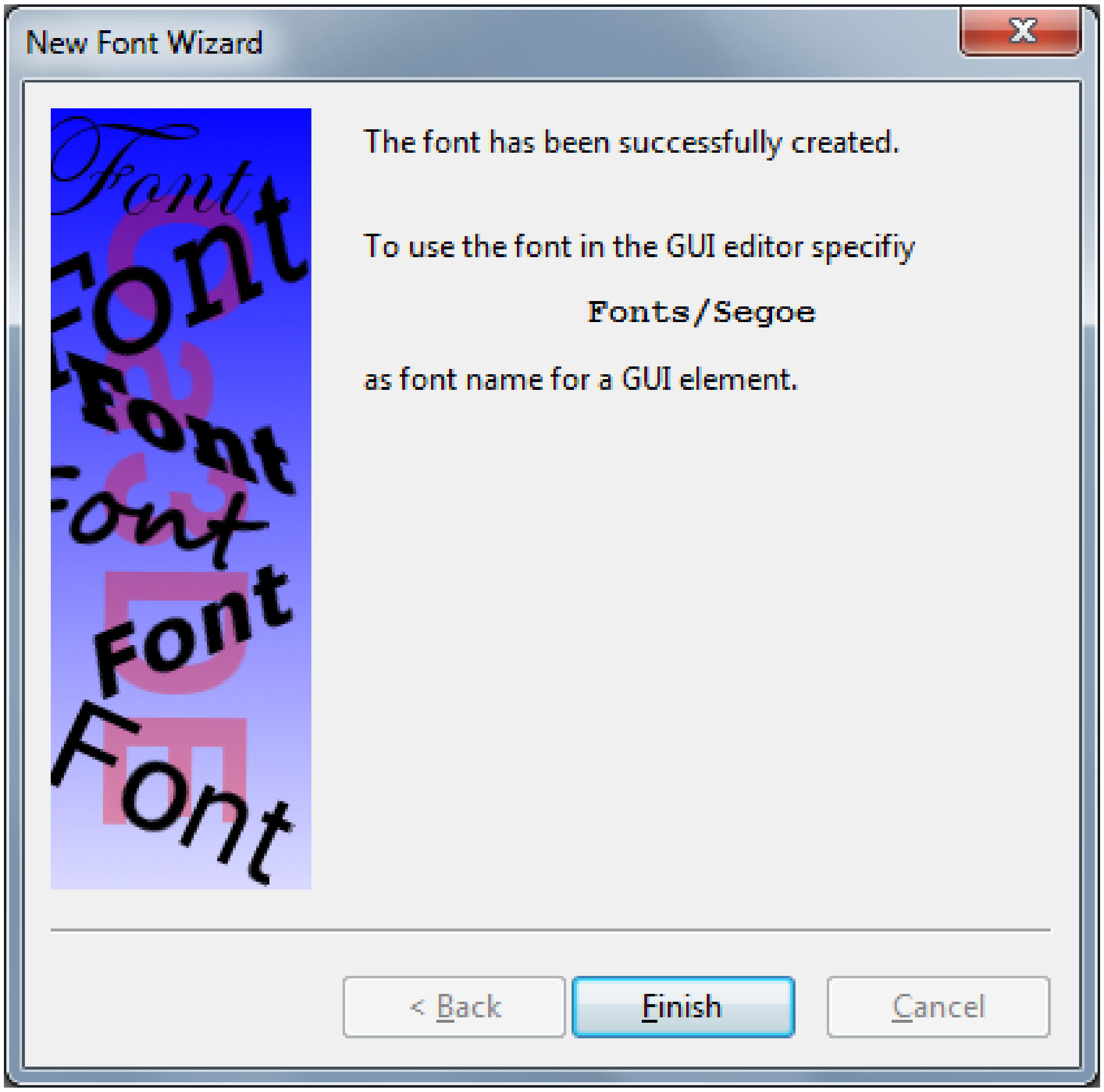
Segoe

The font will be saved to

d:\Dev\Cafu\Fonts\Segoe

(If you want to delete this font later, delete the directory mentioned above.)

Step 4: Enter the name under which you would like to save the font. A new directory is created with the given name and all files related to the font are saved into this directory.


New Font Wizard

The font has been successfully created.

To use the font in the GUI editor specifiy

Fonts/Segoe

as font name for a GUI element.

Step 5: Use the indicated name in the GUI Editor or a GUI script to activate the new font (see below for details).

16.1 Creating fonts at the command prompt

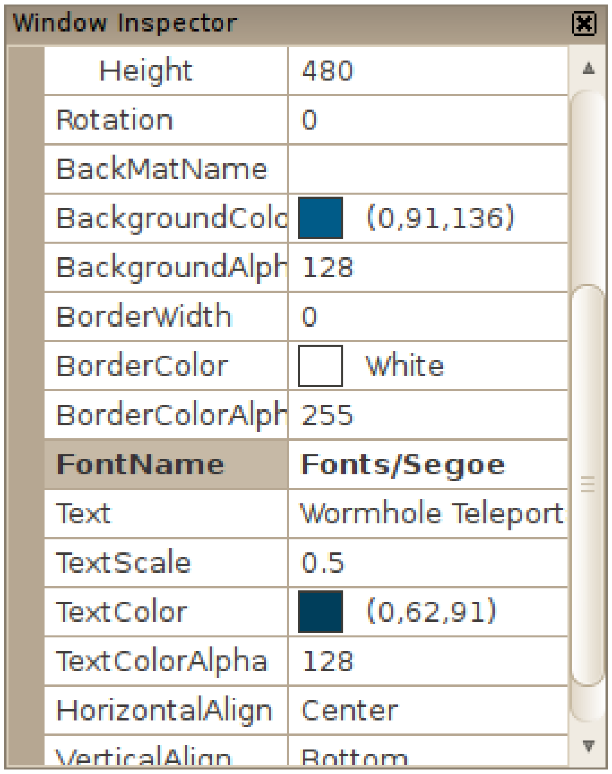
As an alternative to the above, fonts can also be created with the `MakeFont` program at the command prompt. Here is the sequence of commands that creates the same font as above:

```
d:\Dev\Cafu> cd Fonts
d:\Dev\Cafu\Fonts> mkdir Segoe
d:\Dev\Cafu\Fonts> cd Segoe
d:\Dev\Cafu\Fonts\Segoe>..\..\MakeFont.exe C:\Windows\Fonts\segoe.ttf -m=Segoe
The Cafu Font Maker, version July 01 2009.

Portions of this software are copyright (c) 2006 The FreeType Project
(www.freetype.org). All rights reserved.
```

`MakeFont` takes two parameters: the name of the font file whose font you want to use with Cafu GUIs, and the base name of the font materials. The base name is specified as `-m=<base_name>`. Although `MakeFont` will also work if you omit the `-m=...` part, your fonts will not work properly without it, so please make sure to include it. (The Cafu MatSys needs this option in order to be able to render the fonts properly.)

16.2 Using the newly created font



Height	480
Rotation	0
BackMatName	
BackgroundColo	🟦 (0,91,136)
BackgroundAlph	128
BorderWidth	0
BorderColor	⬜ White
BorderColorAlph	255
FontName	**Fonts/Segoe**
Text	Wormhole Teleport
TextScale	0.5
TextColor	🟦 (0,62,91)
TextColorAlpha	128
HorizontalAlign	Center
VerticalAlign	Bottom

Your font is now ready to be used in a GUI. Its name always begins with `Fonts/`, followed by the previously chosen font name. In our example above, the font name is `Fonts/Segoe`.

In the GUI Editor, enter the font name into the appropriate field of the window that should use the font.

Alternatively, if you edit the GUI script manually, just add

```
self:set("font", "Fonts/Segoe");
```

to the appropriate `OnInit()` function to use the new font.

CHAPTER 17

GUI Files Explained

Each GUI comprises several files on disk, such as the GUI scripts, material definition files, and texture images.

GUI files can be created with the GUI Editor, written by hand, or a combination of those:

- The GUI Editor allows users to define and design the contents and layout of the windows that compose a GUI. However, the work in the GUI Editor is focused mainly on static aspects like window position and size, texts, colors, borders, etc.
- Everything "dynamic" and all sorts of effects can, by their nature, not be added by means of the GUI Editor, but must be written as custom, hand-crafted script code that augments the script that is auto-generated by the GUI Editor each time the "static" part of the GUI is saved.

One key question addressed in this section is how we can combine hand-crafted script code and editor-edited script code, while keeping both separated so that they don't overwrite each other.

In general, when you save a GUI in the GUI Editor, the editor creates or re-creates some of the files it is composed of, updates others, and leaves alone the rest. The result is normally exactly what you want and expect, but sometimes you may wish to or must hand-tune the details (such as the GUI scripts or material definitions) without and independently of the GUI Editor. In such cases, it is very helpful to understand the files that belong to a GUI.

This section explains the files that together form a GUI, how they relate to each other, and how hand-crafted and editor-created scripts can peacefully coexist.

17.1 One directory per GUI

Although not a technical requirement and not enforced by the Cafu GUI code (and in fact, at the time of this writing, not yet implemented by the example GUIs that ship with Cafu), it is highly recommended to save each GUI in a directory of its own. This

- explicitly groups all files that logically form and belong to the GUI, and
- makes packaging a complete GUI in a `my_GUI.zip` archive possible, so that the GUI can easily and safely be distributed, shipped and handled.

The name of the directory should match the file name of the GUI. That is, if your GUI's file names are `CallLift_init.cgui` and `CallLift_main.cgui`, it should be stored in a directory with the same name `CallLift/` (or in a zip archive with the same base name `CallLift.zip`).

Note that when you are saving a new GUI that does not yet have a separate directory, you can use the "New Folder" button (or right-click context menu) of the "Save" dialog to create such new directories as required.

17.2 cgui GUI definition files

The `cgui` files are the core files of the GUI: They contain the definitions for the positions, sizes, colors, texts, effects, animations, hierarchy and other properties of the windows that form the GUI.

17.2.1 cgui files are Lua scripts

The Cafu Engine and the GUI Editor load `cgui` files as Lua scripts, and as such they can be inspected or edited in a text editor.

The GUI Editor is usually used to create and edit the static aspects of GUI windows, automatically generating the related script code when saving the file. Dynamic aspects like animations or other kinds of effects typically require adding custom script code, so editing `cgui` files (usually the `_main.cgui` file) is something that you'll likely want to do often.

See the Lua Scripting Overview and the GUI Scripting Reference Documentation for more details.

17.2.2 Augmenting GUI Editor scripts with custom script code

For one GUI there is usually a *pair* of `cgui` files, one suffixed `_init.cgui` and one suffixed `_main.cgui`. For example:

```
d:\Dev\Cafu\Games\DeathMatch\GUIs> dir Teleporter\*.cgui
Teleporter_init.cgui
Teleporter_main.cgui
```

The `_main.cgui` file is for your hand-written GUI script code, if any, and is never touched or overwritten by the GUI Editor (with one exception, see below).

The GUI Editor also writes a secondary `cgui` file whose name ends with `_init.cgui`. This file is written anew each time the GUI is saved, and contains GUI window definitions whose script code was not hand-crafted, but who were created or edited in the GUI Editor.

The two `cgui` files are linked as follows: When the Cafu code loads a GUI, it opens the `_main.cgui` file (`Teleporter_main.cgui`). This file contains a statement like

```
-- Include the GUI Editor generated file.
dofile("Games/DeathMatch/GUIs/Teleporter/Teleporter_init.cgui");

-- Add your hand-written custom code below this line.
-- ...
```

in order to include and process the secondary `_init.cgui` along with the main file.

The only exception when the GUI Editor touches the main `Teleporter_main.cmat` file is when the file does not yet exist, or doesn't contain the `dofile()` reference to the init file. In this case, the `_main.cgui` would not be loaded at all, and thus the GUI Editor inserts the `dofile()` line into the `_main.cgui` file.

In summary, the goal of keeping two separate `cgui` files that are linked as described above is to keep your hand-crafted GUI script code and the GUI Editor edited window definitions cleanly separated, without any danger of one overwriting the other:

- When you edit your GUI in the GUI Editor, you only load and save file `Teleporter_init.cgui`.
- All hand-written code enters file `Teleporter_main.cgui` instead.
- The connection between the two files is made by the `dofile()` statement.

17.3 cmat material definition files

The `cmat` files contain the material definitions for the graphical elements of this GUI.

At the time of this writing, the materials for GUIs are still defined in the "global" material scripts for the MOD, but for the future we intend to have separate material scripts for each GUI that work analogous to cmat material definition files for models.

17.4 Texture images

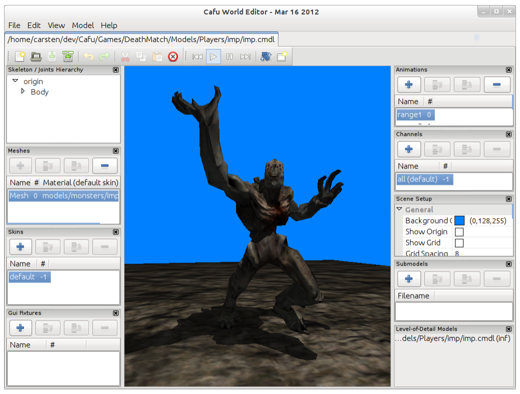
The texture images are referenced from the material definition scripts. See the documentation about the Cafu Material System for more details.

CHAPTER 18

The Model Editor: Introduction

The Cafu Model Editor answers the question:

18.1 *"How do I get my models into Cafu?"*


Cafu World Editor - Mar 16 2012
File Edit View Model Help
/home/carsten/dev/Cafu/Games/DeathMatch/Models/Players/imp/imp.cmdl
Skeleton / Joints Hierarchy
origin
Body
Meshes
Name # Material (default skin)
Mesh 0 models/monsters/imp
Skins
Name #
default -1
Gui Fixtures
Name #
Animations
Name #
range1 0
Channels
Name #
all (default) -1
Scene Setup
General
Background C (0,128,255)
Show Origin
Show Grid
Grid Spacing 8
Submodels
Filename
Level-of-Detail Models
...dels/Players/imp/imp.cmdl (inf)
Thus, its purpose is to:

- load and import model files,
- support many file formats (see below for a list),
- let you inspect the loaded model both graphically and structurally,
- enable you to tweak and adapt the model (e.g. import additional animations, remove unused skeleton nodes, etc.)
- set or adjust all Cafu-specific settings (such as animation properties, GUI panels, collision details, etc.),
- assign and edit render materials,
- save the resulting model in Cafu's own file format.

Summarized, the Cafu Model Editor makes your models ready for use in the Map Editor and the Cafu Engine.

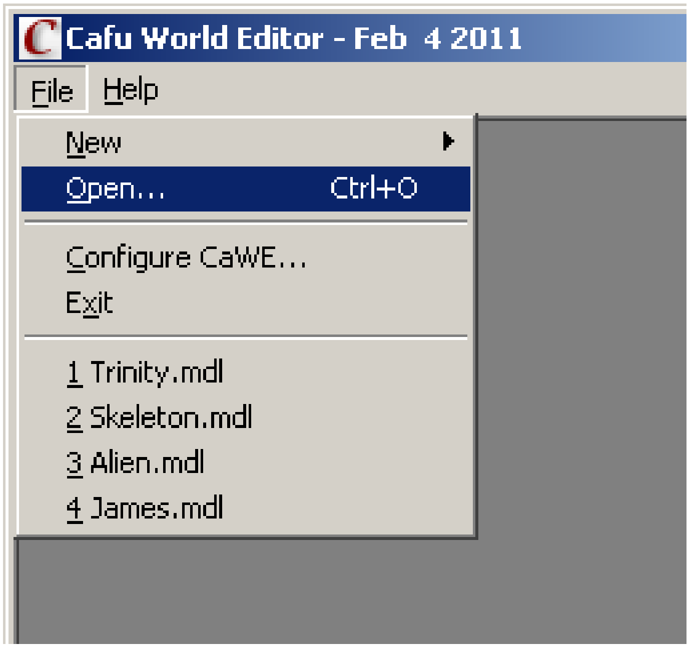
18.1.1 What the Model Editor is not

The Cafu Model Editor is *not*, despite its name, a general-purpose editor that lets you shape meshes or create a model from scratch: There are many commercial and free programs for this purpose already that we don't intend to rival. In fact, model artists use one or more of these programs early in the work pipeline to create and shape the model, then use the Cafu Model Editor as described above.

Also, with the Model Editor we don't attempt to provide a graphical user interface that saves you from editing script code: *Many models files are scripts*, and accessing some features is only possible by editing the script code in these files. Our goal is to employ the Model Editor for everything that is hard or impossible to accomplish with model scripts or otherwise. Tasks that are best solved in script code are left in script code, for which we provide thorough documentation in order to make them easy to understand and master.

With these key ideas, we can achieve the most effective solutions for all tasks about models.

18.1.2 Starting the Model Editor



Like the other asset editors, the Model Editor is a part of CaWE. You start it via menu items

- File → Open… (Ctrl+O) *or*
- File → New → New Model (Ctrl+Shift+N)

With both menu items, you're next presented the **File Open** dialog to open a previously created model.

Check out the *Main Window* and the *How-Tos* for the next steps.

18.1.3 Supported File Formats

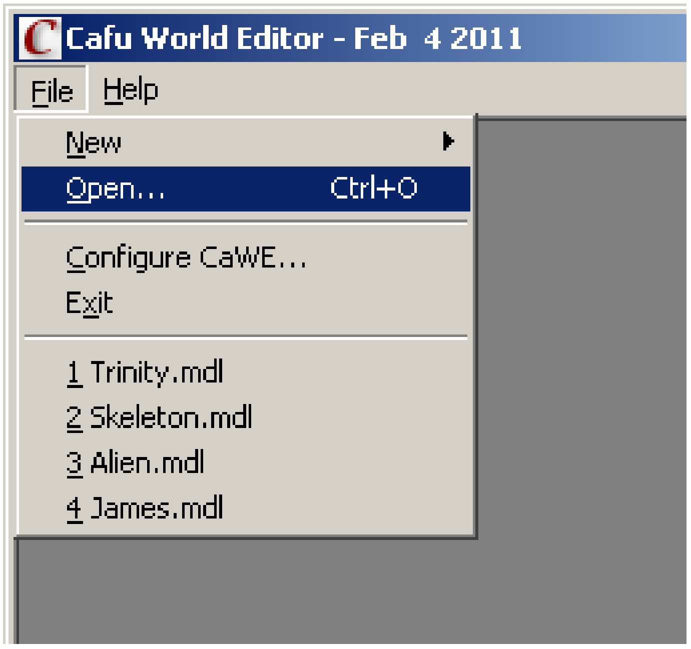
At this time, the following file formats are supported:

Extension	Description
`.cmdl`	Cafu Engine native model file format
`.3ds`	3D Studio 3DS
`.ase`	Ascii Scene Export, 3D Studio Max ASE
`.dxf`	Autodesk AutoCAD DXF
`.dae`	Collada DAE
`.fbx`	Autodesk FBX
`.lwo`	Lightwave Object LWO
`.md5`	Doom3 / Quake4 MD5
`.mdl`	Half-Life1 MDL
`.obj`	Alias OBJ

CHAPTER 19

The Main Window

19.1 Starting the Model Editor



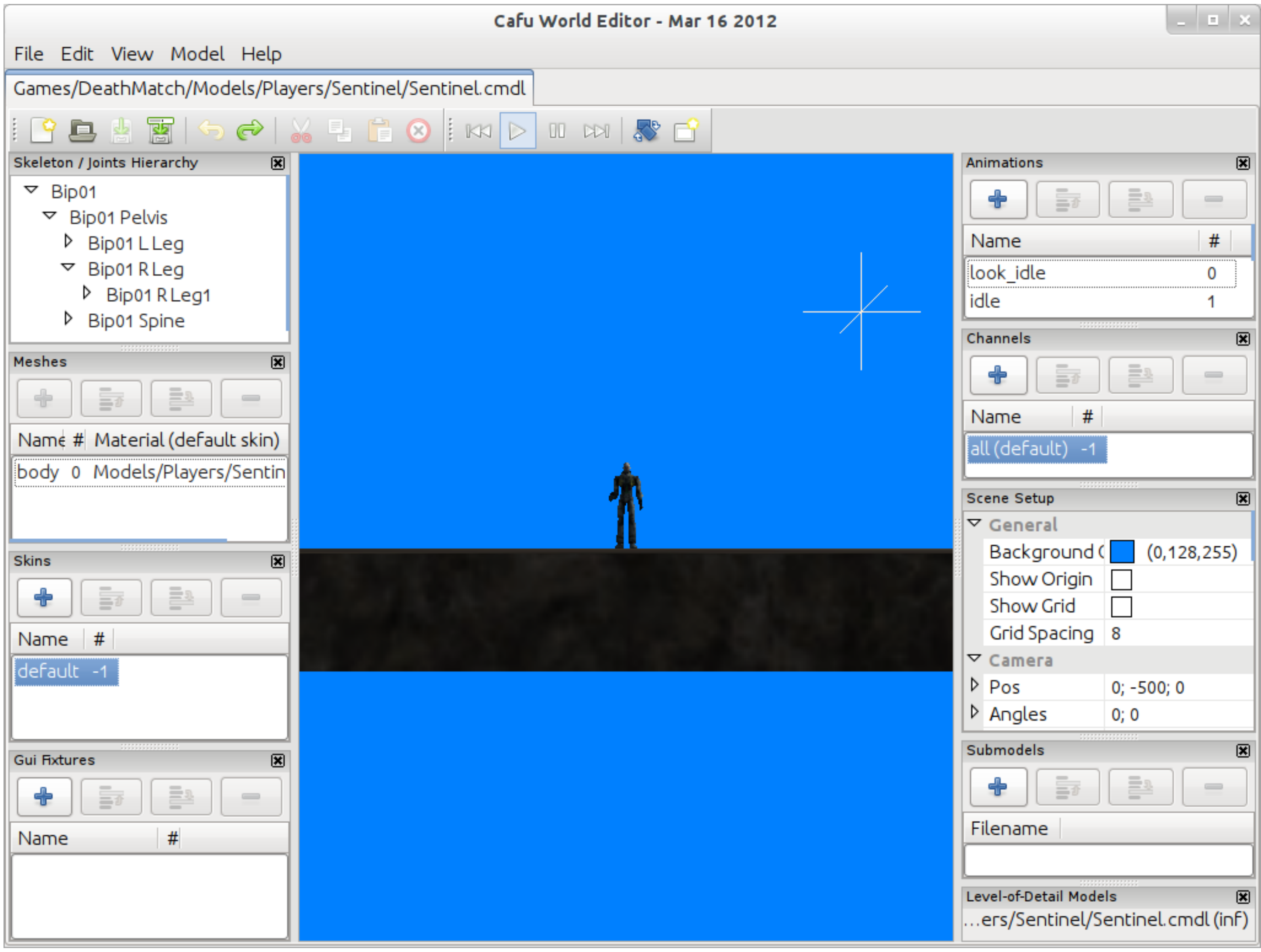
Like the other asset editors, the Model Editor is a part of CaWE. You start it via menu items

- File → Open. . . (Ctrl+O) *or*
- File → New → New Model (Ctrl+Shift+N)

With both menu items, you're next presented the **File Open** dialog to open a previously created model.

Note that while the Model Editor can load or import models from *many different file formats*, it can *save* models only in the `cmdl` file format that is specific to the Cafu Engine: Unlike any other file format, `cmdl` models files reflect all the model features that are implemented in the Cafu Engine.

19.2 Main Window Elements



Games/DeathMatch/Models/Players/Sentinel/Sentinel.cmdl

Bip01
Bip01 Pelvis
Bip01 L Leg
Bip01 R Leg
Bip01 R Leg1
Bip01 Spine

Name	#	Material (default skin)
body	0	Models/Players/Sentin

Name	#
default	-1

Name	#

Name	#
look_idle	0
idle	1

Name	#
all (default)	-1

General	
Background (	🟦 (0,128,255)
Show Origin	☐
Show Grid	☐
Grid Spacing	8
Camera	
Pos	0; -500; 0
Angles	0; 0

Filename

...ers/Sentinel/Sentinel.cmdl (inf)

19.2.1 3D View

The 3D view shows the currently loaded model. The mouse and keyboard navigation works exactly like in the 3D views of the Map Editor: See

- *Navigating the 3D views* and
- the *related video*

for details.

Tip: Especially the `MMB` can be helpful for effective navigation in the 3D view.

19.2.2 Detail Panes

Model details, scene settings and some dialogs of the Model Editor are presented in *panes*: Small windows that can be docked to the borders of the parent frame, or be "floating" freely on the desktop.

If you grab a pane by its title bar and drag it over the screen, a translucent highlight will indicate where the pane will dock when the left mouse button is released. Press and hold the `CTRL` key while dragging the pane in order to prevent it from docking at all (it will remain floating instead).

If you close a pane, use the appropriate item in the **View** menu to show it again at any time.

Model Element Panes

Most of the panes present model elements of a specific type:

- the skeleton (also called the joints hierarchy),
- meshes,
- skins,
- GUI fixtures,
- animations,
- channels,
- submodels, and
- level-of-detail models.

Note that for most model elements, there are in fact *two* panes: One that shows the list of all elements in the model, and one that shows the details of the currently selected element.

For example, the "Skeleton" pane lists all joints in the model, hierarchically arranged. Double-click any of them to bring up the "Joint Inspector" that shows the properties of the currently selected joint.

On the following pages in this manual there is one chapter for each element type. Each chapter explains the model element type and its related "list" and "inspector" panes.

19.2.3 The Menu

The program menu provides access to all program features.

Refer to chapter *Menu and Toolbar Reference* for a detailed description of each menu item.

19.2.4 The Toolbar

The toolbar provides quick access to the most frequently used menu items.

Move the mouse over any toolbar button to see a tooltip that describes the purpose of the button.

CHAPTER 20

How-Tos

20.1 How do I . . .

20.1.1 . . . get my model into Cafu?

See this video for details.

Additional notes:

- When watching the video, change the playback quality to “720p HD” if possible.
- There is no audio track in this video.
- Use the *New Entity* tool for placing the newly created `.cmdl` model into a map. Watch the Placing a Model tutorial for related information.
- If you want to make adjustments later, both the `.cmdl` model as well as the material definitions can be edited as shown above also *after* the model has been placed into a map or world file. The changes will take effect when the program is next restarted.
- In the video, why did we *close* the model window before editing the material definitions? When you save a model file, the Model Editor *overwrites* the `_editor.cmat` file as explained at *Model Files Explained*. In fact, we should never have hand-edited the `_editor.cmat` file, but do all edits in `.cmat` (without the `_editor` part) instead, but we thought it’s too difficult to explain that in the video. Additionally, the Model Editor had to re-read the new material definitions anyway, so even if our edits were made in the right file, we had to close and reopen the model.

20.1.2 . . . scale my model?

There are two main methods for scaling a model:

- via the Skeleton / the Joint Inspector
- via the Transform dialog

The *Joint Inspector* can transform individual joints of the model, but only for the *bind pose*. It is mostly useful for static, non-animated models, but even then only if there is a special requirement that the Transform dialog cannot handle.

The *Transform dialog* is much easier to use, as it transforms the entire model (the whole skeleton): the transformation is applied to all currently selected animation sequences. If no animation sequence is selected at all, the "bind pose" is transformed.

20.1.3 ... import animations from other models?

In some cases, importing animations is very easy: Use the "**+**" button in the *Animations* pane as demonstrated in the above video.

However, this works only if the animations are stored in separate files that are compatible with the already loaded mesh. This is specifically the case with `md5anims` for their related `md5mesh` file.

We will extend this functionality in the future. Until then and for all other cases, see the next How-To about generically sharing and re-using elements.

20.1.4 ... share and re-use animations, skins etc. from other models?

Creating variants of a model by sharing or re-using elements of another model is a straightforward idea, but generally not easy to implement: For example, it's easy to share meshes and animations when the skeleton is an exact match, but much harder to do otherwise.

The solution is to manually edit the `cmdl` model file, which is a Lua script as documented at *Model Files Explained*.

For example, copying and pasting the definition of an animation from one player model into another (that has a compatible, matching skeleton), tends to work very well.

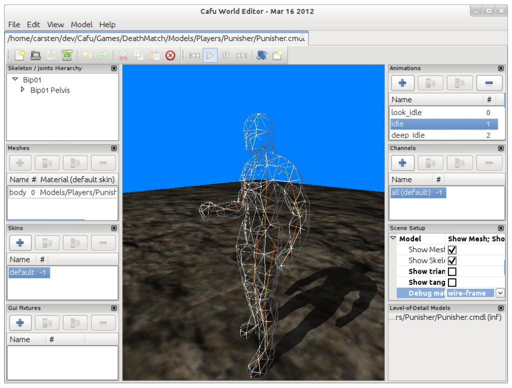
It is also possible to use Lua's `dofile()` statement to include models into each other, which we intent to augment in the future to load globals from one model file into a table of another, as to keep the global namespace of the current model clean. All variants of such file inclusion however suffer the problem that it creates a dependency of one file on another. This may be exactly what you want, or maybe not – where in the latter case falling back to copy and paste is probably the better solution.

CHAPTER 21

Model Elements

21.1 Skeleton

The skeleton forms the basis of a model. Technically, it is built from a hierarchy of “coordinate systems”: Starting at the root, each coordinate system is defined in the coordinate system of its parent. Model artists use these coordinate systems to fix the points (weights and vertices) that eventually form the meshes of the model.



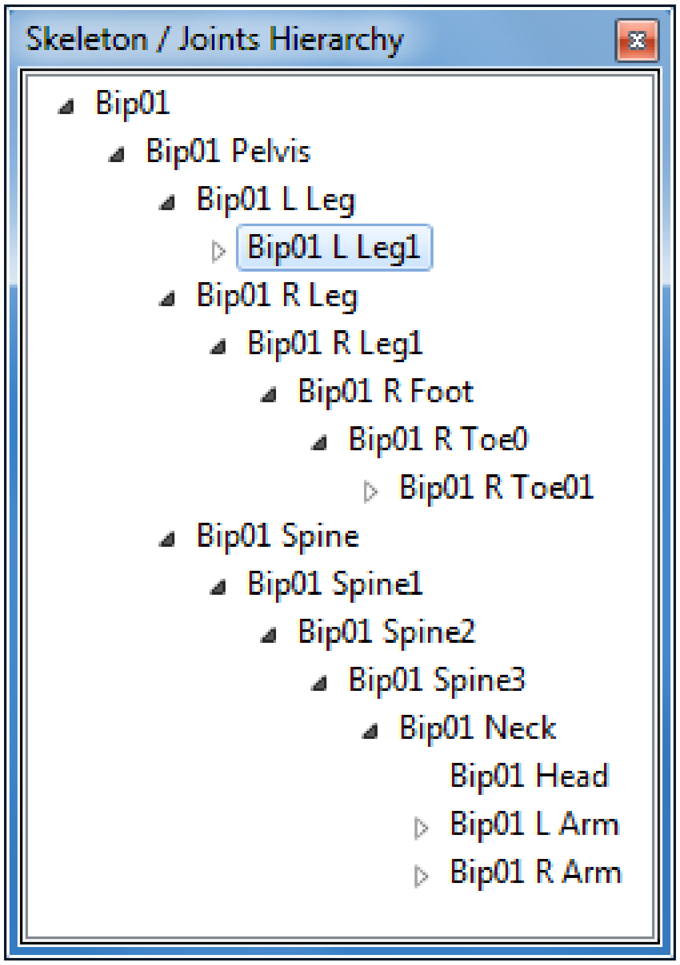
Cafu World Editor - Mar 16 2012
File Edit View Model Help
/home/carsten/dev/Cafu/Games/DeathMatch/Models/Players/Punisher/Punisher.cmdl
Skeleton / Joints Hierarchy
Bip01
Bip01 Pelvis
Meshes
Name # Material (default skin)
body 0 Models/Players/Punish
Skins
Name #
default -1
Gui Fixtures
Name #
Animations
Name #
look_idle 0
idle 1
deep_idle 2
Channels
Name #
all (default) -1
Scene Setup
Model Show Mesh; Sho
Show Mesh ✓
Show Skele ✓
Show trian
Show tang
Debug mat wire-frame
Level-of-Detail Models
...rs/Punisher/Punisher.cmdl (inf)

If we compare the skeleton of a model to the skeleton of the human body, the “coordinate systems” closely correspond to “joints”: the coordinate origin is in the center of the joint, and the coordinate axes correspond to the bones that extend from the joint. In this analogy, a “bone” is the line segment between a coordinate origin and the coordinate origin of its parent system.

You can inspect the skeleton in 3D view in the *Scene Setup*:

- set **Show Mesh** to “no”
- set **Show Skeleton** to “yes”

21.1.1 The joints hierarchy

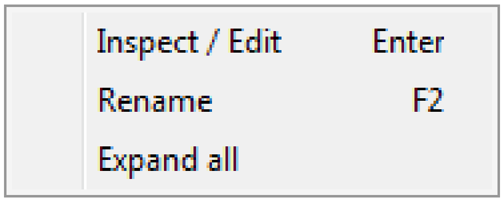


The **Skeleton / Joints Hierarchy** pane shows all the joints in the skeleton, hierarchically arranged. Items in the hierarchy tree can be expanded or collapsed with the nearby symbols.

A single click on a joint selects it, a double click opens the **Joint Inspector** pane as well.

Pressing the **F2** key or a single-click on an already selected joint allows you to rename the joint in place.

Context menu



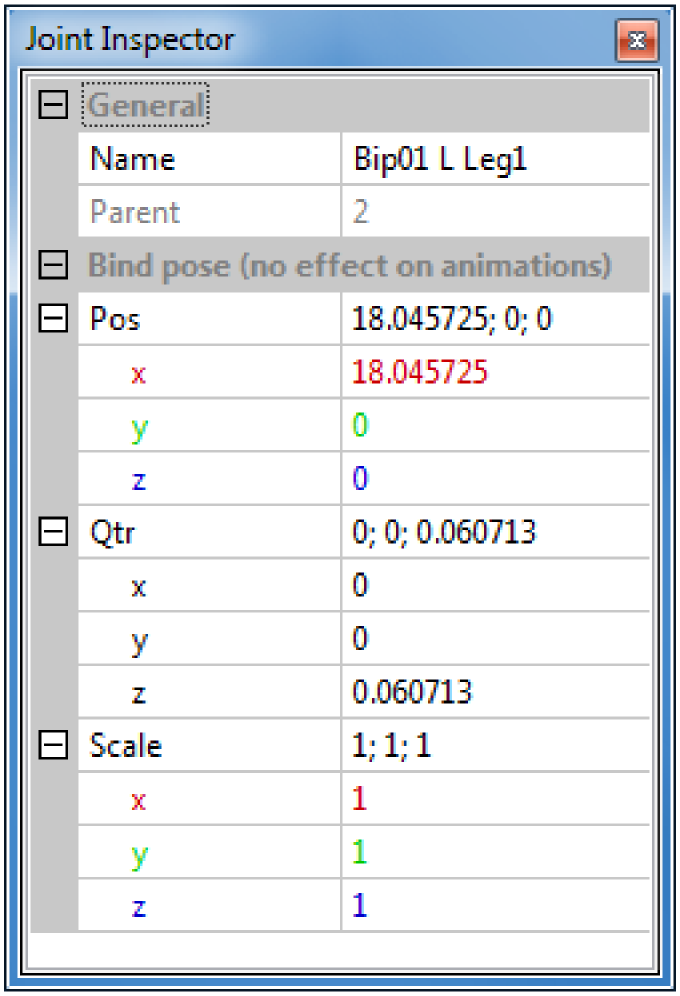
An RMB click in the **Skeleton / Joints Hierarchy** pane opens the context menu:

Inspect/Edit opens the **Joint Inspector** pane,

Rename allows to rename the joint,

Expand all expands all items in the joints hierarchy tree.

21.1.2 The joint inspector



Joint Inspector

General	
Name	Bip01 L Leg1
Parent	2
Bind pose (no effect on animations)	
Pos	18.045725; 0; 0
x	18.045725
y	0
z	0
Qtr	0; 0; 0.060713
x	0
y	0
z	0.060713
Scale	1; 1; 1
x	1
y	1
z	1

The **Joint Inspector** pane shows the details of the currently selected joint.

General

The General section shows the name of the joint (it's another option to rename it), and the index number of the parent joint.

The parent index number is a technical detail that cannot be changed. You need it only if you *edit the model file* in a text editor or work with the Cafu model C++ data structures.

Bind pose

This section shows the position, orientation and scale of the joint (i.e., the coordinate system).

Note that the details presented here only affect the *bind pose* of the model. The *bind pose* is the default pose of the model: it is used whenever no animation sequences are loaded or none are actively used.

That is, the **Joint Inspector** is great for manipulating *the individual bones of static models*.

It is however useless for animated models, because technically, each animation sequence brings its own set of (animated) joints that cannot be manipulated individually. If you want to do that, refer to *Transforms: translate, rotate and scale* instead.

The currently selected joint of the bind pose is described by these parameters:

Pos defines the position of the coordinate origin relative to the parent system.

Qtr defines the first three values of the quaternion that describes the orientation of the coordinate system. You should *not* attempt to manipulate these values if you don't know what quaternions are and why it only shows three instead of four values.

Scale expresses the scale (or relative length) of the three coordinate axes.

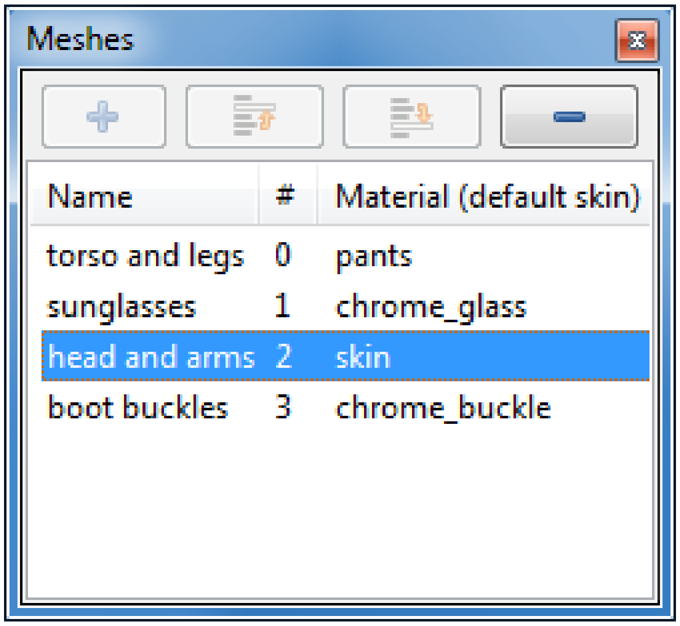
21.2 Meshes

The meshes define the shape of the model.

A model can have several meshes. Each mesh is made of triangles, whose vertices are composed of one or more weights, that in turn are attached to the *Skeleton* of the model.

Despite its name, it's not the purpose of the Model Editor to modify the positions of weights and vertices, or to provide means to generally shape the meshes. Please see the *The Model Editor: Introduction* for more details. Instead, the Model Editor focuses on setting details that are specific to the Cafu Engine.

21.2.1 The meshes list



Name	#	Material (default skin)
torso and legs	0	pants
sunglasses	1	chrome_glass
head and arms	2	skin
boot buckles	3	chrome_buckle

The **Meshes** pane lists all meshes in the model.

For each mesh,

- its name,
- the mesh number, and
- the mesh material (in the currently selected *skin*)

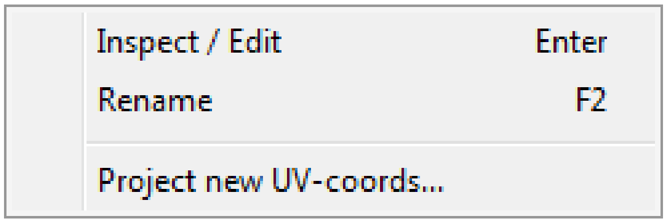
is shown.

A single click on a mesh selects it, a double click opens the **Mesh Inspector** pane as well.

Pressing the **F2** key or a single-click on an already selected mesh allows you to rename the mesh in place.

The "**-**" button at the top of the list deletes the currently selected meshes.

Context menu

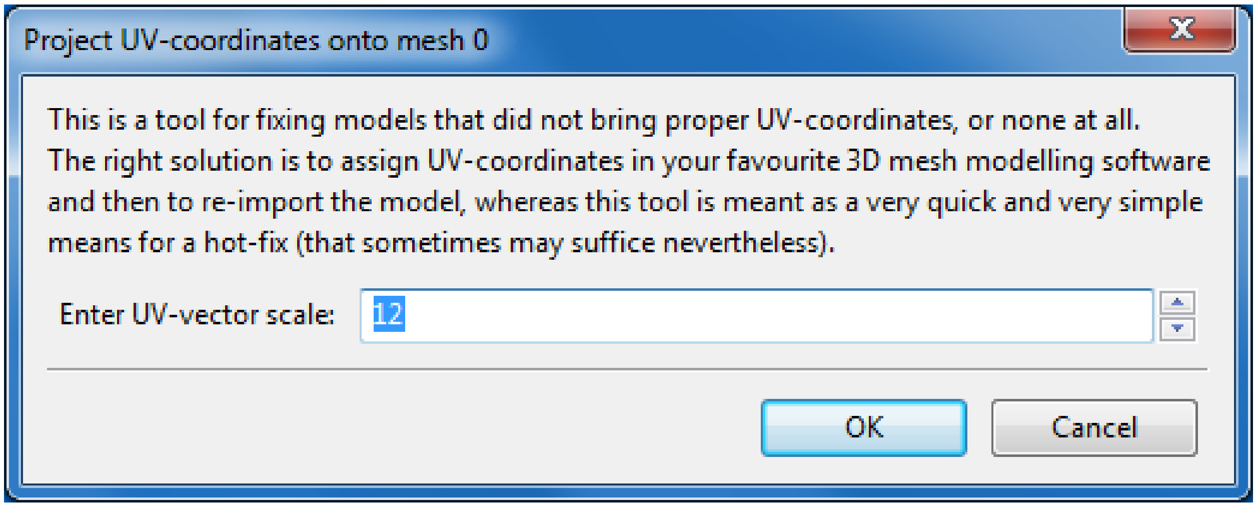


An RMB click in the **Meshes** pane opens the context menu:

Inspect/Edit opens the **Mesh Inspector** pane.

Rename allows to rename the mesh.

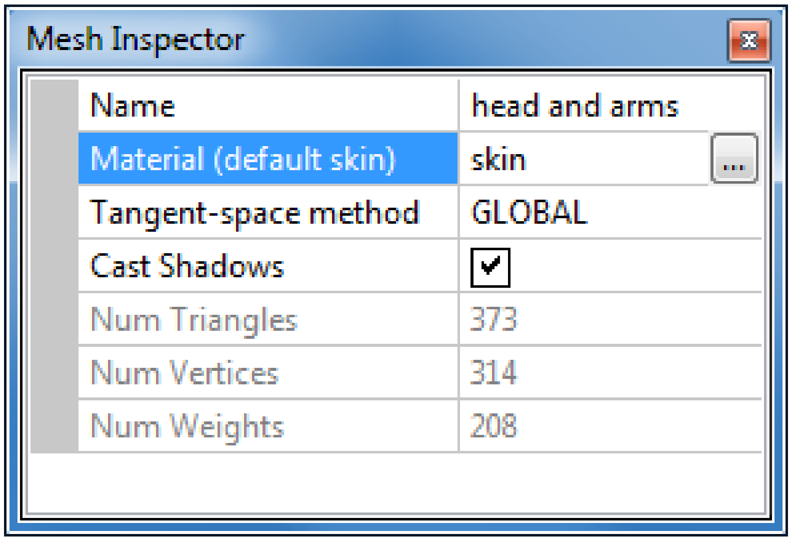
Project new UV-coords… is a tool for hot-fixing models that did not bring proper UV-coordinates, or none at all:


Project UV-coordinates onto mesh 0

This is a tool for fixing models that did not bring proper UV-coordinates, or none at all.
The right solution is to assign UV-coordinates in your favourite 3D mesh modelling software and then to re-import the model, whereas this tool is meant as a very quick and very simple means for a hot-fix (that sometimes may suffice nevertheless).

Enter UV-vector scale: 12

21.2.2 The mesh inspector



Name	head and arms
Material (default skin)	skin
Tangent-space method	GLOBAL
Cast Shadows	✓
Num Triangles	373
Num Vertices	314
Num Weights	208

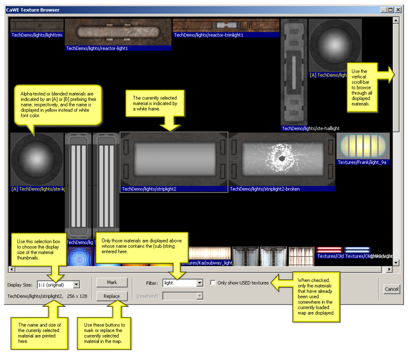
The **Mesh Inspector** pane shows the details of the currently selected mesh.

Name

Shows the name of the currently selected mesh. The name can be edited in order to rename the mesh.

Material

The material that is used for rendering the mesh is set here: Press the "…" button in order to open the **Material Browser**.


CaWE Texture Browser
TechDemo/lights/lighttrim
TechDemo/lights/reactor-light1
TechDemo/lights/reactor-trimlight1
[A] TechDemo/ligh
TechDemo/lights/ste-halllight
[A] TechDemo/lights/ste-lig
TechDemo/lights/striplight2
TechDemo/lights/striplight2-broken
Textures/Frank/light_9a
TechDemo/lig
xtures/Kai/subway_light
Textures/Old
Textures/Old
Display Size:
1:1 (original)
Mark
Replace
Filter:
light
(reserved)
Only show USED textures
Cancel
TechDemo/lights/striplight2, 256 x 128

Alpha-tested or blended materials are indicated by an [A] or [B] prefixing their name, respectively, and the name is displayed in yellow instead of white font color.

The currently selected material is indicated by a white frame.

Use the vertical scroll-bar to browse through all displayed materials.

Use this selection box to choose the display size of the material thumbnails.

Only those materials are displayed above whose name contains the (sub-)string entered here.

When checked, only the materials that have already been used somewhere in the currently loaded map are displayed.

The name and size of the currenly selected material are printed here.

Use these buttons to mark or replace the currently selected material in the map.

The Material Browser is the same as the one used in the Map Editor, but it only shows the materials that are specific to the currently loaded model. It also has some basic capabilities to *edit* materials and to set their properties, but is currently not very advanced.

At this time, we recommend that you use the Material Browser for assigning materials to meshes, but in order to create and edit material definitions, best use a programmers editor for editing the *related cmat material files* directly: Material scripting is documented in chapter The Cafu Material System.

(It is our goal to turn the Material Browser into a full-featured Material Editor in the near future, but we currently work on improving other parts of the Cafu code. If you want to lend a helping hand, please let us know!)

Tangent-space method

The tangent-space method determines the algorithm that is used for computing the normal, tangent and bi-tangent vectors at the vertices of the meshes.

Ideally, the algorithm that is set here should match the algorithm that was used by the external program that was used for creating the *normal map* for this mesh. This is important so that the Cafu model code can accurately reproduce the subtle details involved in generating the axes of tangent-space, as only then the lighting results will exactly match those in the original program. Unfortunately, the involved algorithms are not standardized at this time (and often not even documented!), so that we can only try a best guess.

Refer to CafuModelT::MeshT::TangentSpaceMethodT for the currently available tangent-space methods. More tangent-space methods will be added in the future.

If in doubt, there is a very simple receipt for choosing the right method: simply pick the one that looks best.

Cast shadows

This setting determines whether the mesh casts shadows when lit by a dynamic light source.

This can be useful for performance tuning, e.g. in order to not have shadows casts for very small meshes that implement model details, or generally for models that are very far away, or never seen in a context where shadows play a role.

Statistics

The last three numbers show the number of triangles, vertices and weights in the mesh. They cannot be changed.

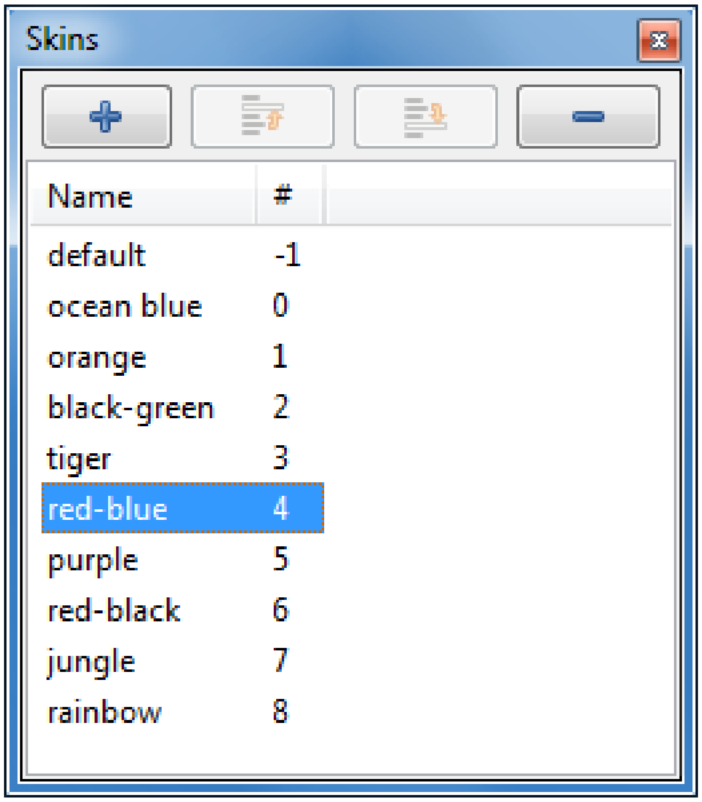
21.3 Skins

Skins are used to apply alternative sets of materials to a model.

Typical examples are the butterfly model, where a single mesh is used with multiple skins to implement several species of butterflies; or a monster that exists in tame, mean and hellborn variants.

Skins are defined and edited in the Model Editor as described below. The *use* of skins is up to the game's script or C++ code.

21.3.1 The skins list



Name	#
default	-1
ocean blue	0
orange	1
black-green	2
tiger	3
red-blue	4
purple	5
red-black	6
jungle	7
rainbow	8

The **Skins** pane lists all skins in the model.

For each skin,

- its name and
- the skin number

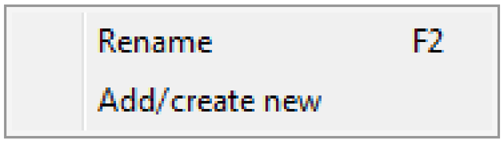
is shown.

A single click on a skin selects it, a double click opens the **Skin Inspector** pane as well.

Pressing the **F2** key or a single-click on an already selected skin allows you to rename the skin in place.

The "**+**" button creates a new skin and adds it to the list. The "**-**" button deletes the currently selected skins.

Context menu

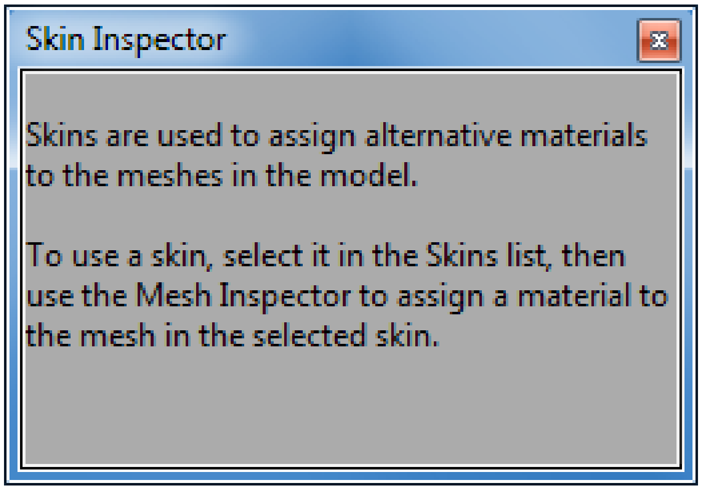


An RMB click in the **Skins** pane opens the context menu:

Rename allows to rename the skin.

Add/create new , like the "**+**" button, creates a new skin and adds it to the list.

21.3.2 The skin inspector


Skin Inspector

Skins are used to assign alternative materials to the meshes in the model.

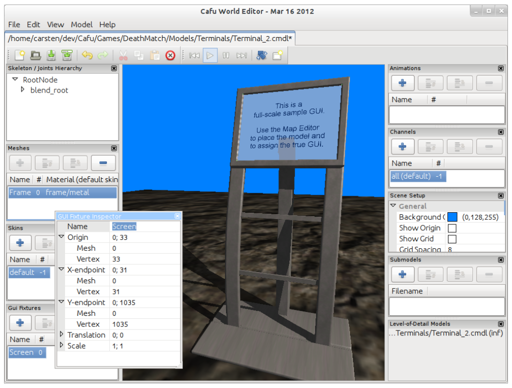
To use a skin, select it in the Skins list, then use the Mesh Inspector to assign a material to the mesh in the selected skin.

Skins don't have explicit properties of their own, thus the **Skin Inspector** pane is just a help dialog that explains the situation:

To use a skin, select it in the **Skins** list (shown above), then use the *Mesh Inspector* to assign a material to the mesh in the selected skin.

21.4 GUI Fixtures

GUI fixtures are used to attach GUIs to models.


Cafu World Editor - Mar 16 2012
File Edit View Model Help
/home/carsten/dev/Cafu/Games/DeathMatch/Models/Terminals/Terminal_2.cmdl*
Skeleton / Joints Hierarchy
RootNode
blend_root
Meshes
Skins
Gui Fixtures
Animations
Channels
Scene Setup
Submodels
Level-of-Detail Models
This is a full-scale sample GUI.
Use the Map Editor to place the model and to assign the true GUI.

Meshes: Name | # | Material (default skin — Frame 0 frame/metal

Skins: Name | # — default -1

Gui Fixtures: Name | # — Screen 0

Channels: Name | # — all (default) -1

GUI Fixture Inspector

Name	Screen
Origin	0; 33
Mesh	0
Vertex	33
X-endpoint	0; 31
Mesh	0
Vertex	31
Y-endpoint	0; 1035
Mesh	0
Vertex	1035
Translation	0; 0
Scale	1; 1

Scene Setup — General: Background C (0,128,255); Show Origin ☐; Show Grid ☐; Grid Spacing 8

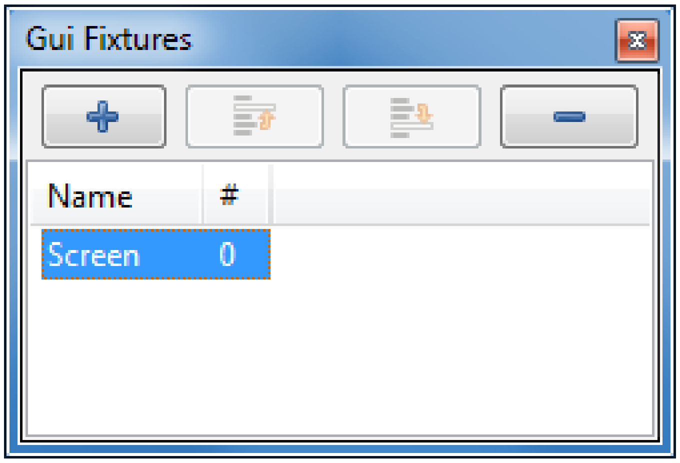
Submodels: Filename

Level-of-Detail Models: ...Terminals/Terminal_2.cmdl (inf)

GUIs are virtual computer desktops that the player can interact with to unlock doors, call lifts, obtain information, and much more.

They are created in the Cafu Engine Gui Editor and use scripts for their implementation and for handling events when interacting with the player.

21.4.1 The GUI fixtures list



The **GUI Fixtures** pane lists all GUI fixtures in the model.

For each GUI fixture,

- its name and
- the GUI fixture number

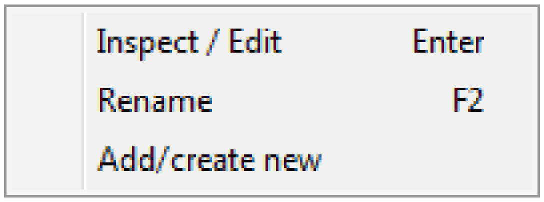
is shown.

A single click on a GUI fixture selects it, a double click opens the **GUI Fixture Inspector** pane as well.

Pressing the **F2** key or a single-click on an already selected GUI fixture allows you to rename the GUI fixture in place.

The “**+**” button creates a new GUI fixture and adds it to the list. The “**-**” button deletes the currently selected GUI fixtures.

Context menu



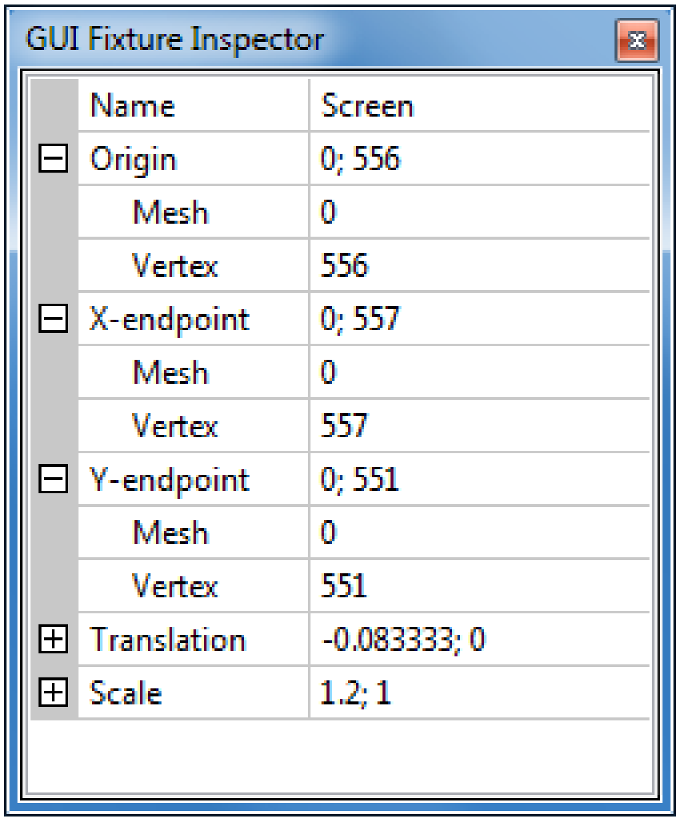
An RMB click in the **GUI Fixtures** pane opens the context menu:

Inspect/Edit opens the **Gui Fixture Inspector** pane.

Rename allows to rename the GUI fixture.

Add/create new , like the “**+**” button, creates a new GUI fixture and adds it to the list.

21.4.2 The GUI fixture inspector



GUI Fixture Inspector

Name	Screen
⊟ Origin	0; 556
Mesh	0
Vertex	556
⊟ X-endpoint	0; 557
Mesh	0
Vertex	557
⊟ Y-endpoint	0; 551
Mesh	0
Vertex	551
⊞ Translation	-0.083333; 0
⊞ Scale	1.2; 1

The **GUI Fixture Inspector** pane shows the details of the currently selected GUI fixture.

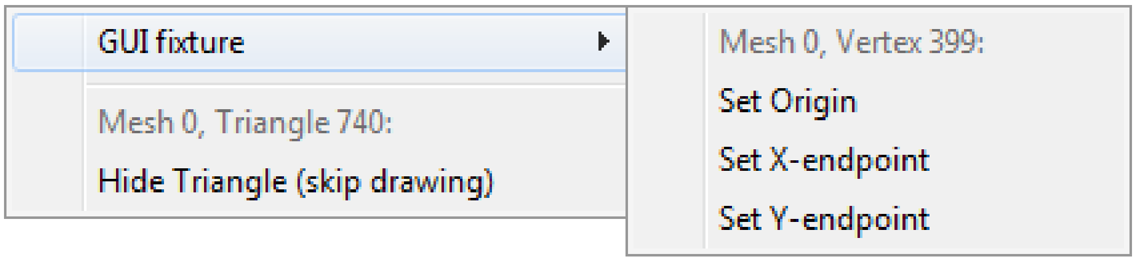
Name

Shows the name of the currently selected GUI fixture. The name can be edited in order to rename the GUI fixture.

Origin and endpoints

The origin and the endpoints of the x- and y-axes determine the position and orientation of the GUI.

You can enter and edit the mesh and vertex numbers manually, but it is much easier and faster to right-click directly on the model in the 3D view:



Selecting one of the three **GUI fixture** menu items will fill in the related numbers in the **GUI Fixture Inspector** automatically.

Translation and scale

Normally, the GUI rectangle is exactly aligned to the origin and the endpoints of the axes. Using the translation and scale, you can move the GUI rectangle from its original position and adjust its overall size:

- The **Scale** values set the relative lengths of the axes.
- The **Translation** values move the origin in multiples of the scaled axes.

21.4.3 Example video

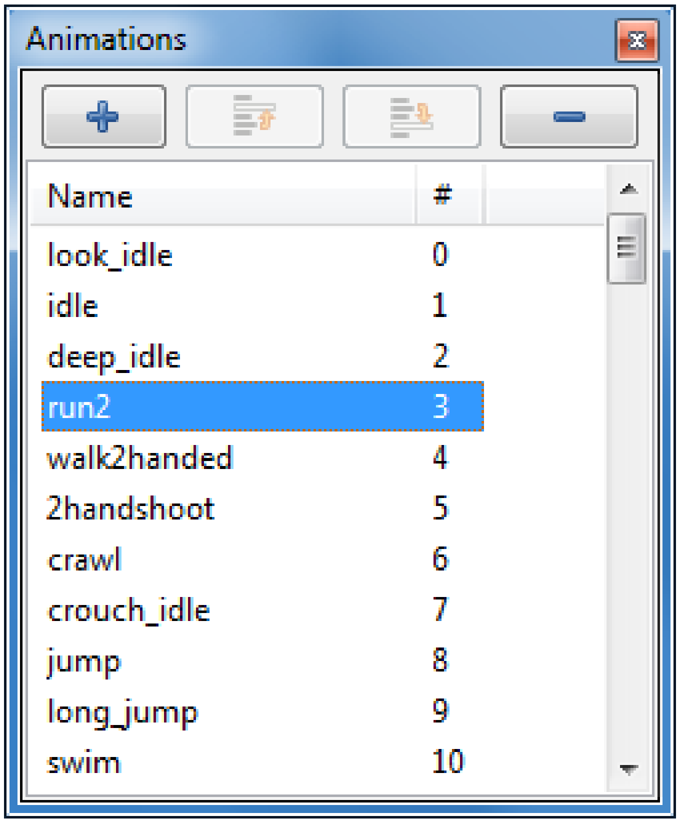
This video shows a GUI that is attached to an animated model and uses custom translation and scale settings.

21.5 Animations

For many models, animations are the most noticeable and most important feature: Animation sequences define how a model moves, behaves or transforms over a period of time.

Animation sequences, like the model itself, are typically created by the model artist before the model is imported into the Model Editor.

21.5.1 The animations list



Name	#
look_idle	0
idle	1
deep_idle	2
run2	3
walk2handed	4
2handshoot	5
crawl	6
crouch_idle	7
jump	8
long_jump	9
swim	10

The **Animations** pane lists all animation sequences in the model.

For each animation sequence,

- its name and
- the sequence number

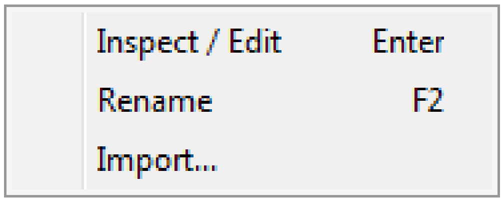
is shown.

A single click on an animation sequence selects it, a double click opens the **Animation Inspector** pane as well.

Pressing the **F2** key or a single-click on an already selected animation sequence allows you to rename the sequence in place.

The "**+**" button imports a new animation sequence (e.g. from a `.md5anim` file) and adds it to the list. The "**-**" button deletes the currently selected animation sequences.

Context menu



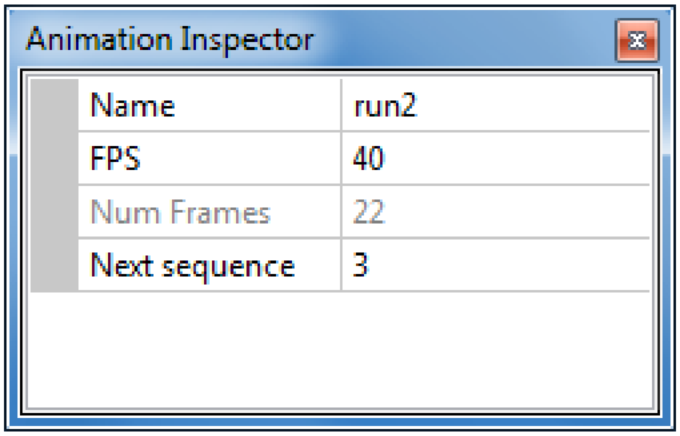
An RMB click in the **Animations** pane opens the context menu:

Inspect/Edit opens the **Animation Inspector** pane.

Rename allows to rename the animation sequence.

Import... , like the "+" button, imports a new animation sequence (e.g. from a `.md5anim` file) and adds it to the list.

21.5.2 The animation inspector



Name	run2
FPS	40
Num Frames	22
Next sequence	3

The **Animation Inspector** pane shows the details of the currently selected animation sequence.

Name

Shows the name of the currently selected animation sequence. The name can be edited in order to rename the sequence.

FPS

The speed in frames-per-second with which this animation sequence is played.

It's normally set by the model artists and needs only be changed e.g. for special effects such as slow motion, or if the right number was not correctly imported from the original model file, etc.

Num Frames

The number of key frames that the sequence consists of. The key frames are created by the model artist and cannot be changed here.

Next sequence

This is the number of the animation sequence to play after the current sequence.

This value is set to -1 to indicate "no" next sequence, which is used for animations that should just stop when they've reached their last frame. For example, a sequence of a player model for dropping to the ground after it was mortally wounded typically has a next sequence value of -1, indicating that this is a "non-looping" sequence.

Setting the value to the same number as the number of this sequence means "play this sequence again when its end has been reached". It's another way of saying that this is a looping sequence, such as walking or running.

Setting the value to a number other than -1 ("non-looping") or the own sequence number ("looping") can be useful to indicate logically consecutive sequences. For example, if a player model has a sequence for holstering its weapon, the "next sequence" value might be the number of an idle sequence – whatever the player does after he has the hands free.

In summary, note that the "next sequence" setting is mostly used to indicate looping vs. non-looping sequences, and that often the game script or C++ code will override whatever has been entered here: The model artists and the game programmers will have to communicate regarding the exact purpose of the animation sequences in order to make best use of this feature.

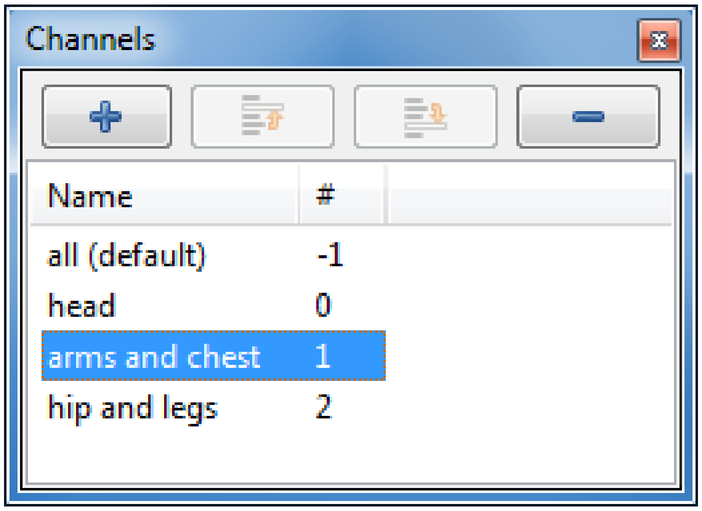
21.6 Channels

Channels are used for grouping joints. They allow animations to play only on a subset of the model joints, so that multiple animations can play on different parts of the model at the same time.

For example, you can play a walking animation on the legs, an animation for swinging the arms on the upper body, and an animation for moving the eyes on the head.

Animation channels are defined and inspected in the Model Editor as described below. The *use* of channels is up to the game's script or C++ code.

21.6.1 The channels list



Name	#
all (default)	-1
head	0
arms and chest	1
hip and legs	2

The **Channels** pane lists all animation channels in the model.

For each channel,

- its name and
- the channel number

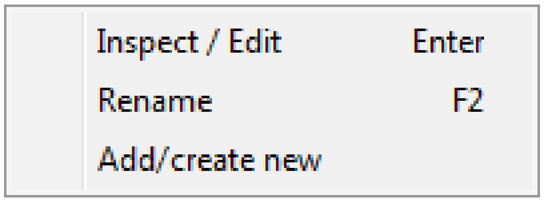
is shown.

A single click on a channel selects it, a double click opens the **Channel Inspector** pane as well.

Pressing the **F2** key or a single-click on an already selected channel allows you to rename the channel in place.

The “**+**” button creates a new channel and adds it to the list. The “**-**” button deletes the currently selected channels.

Context menu



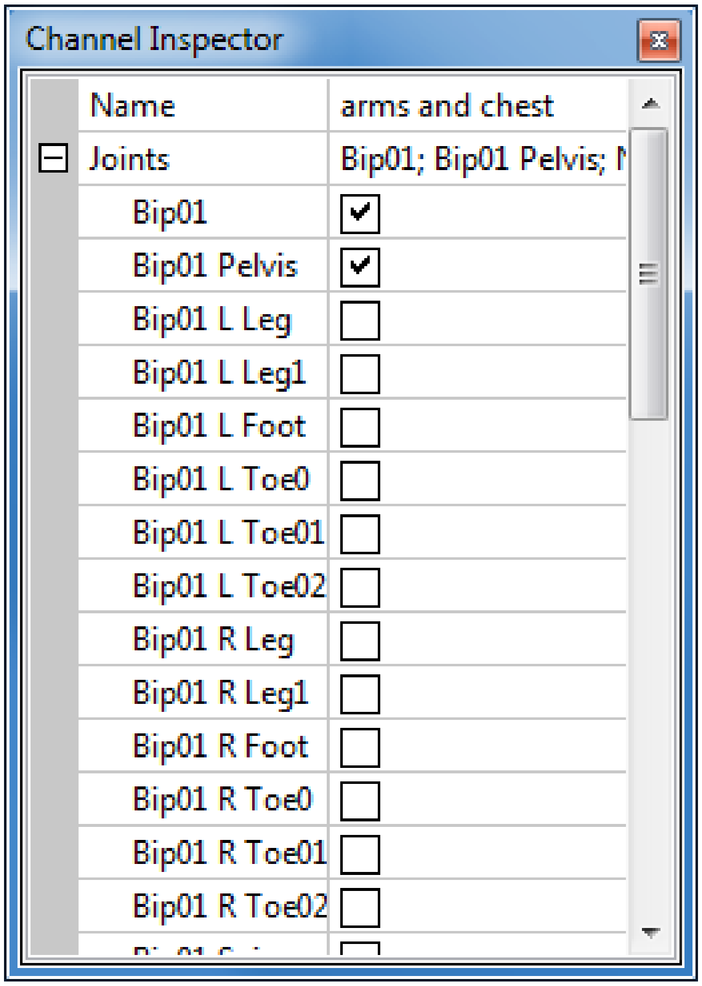
An RMB click in the **Channels** pane opens the context menu:

Inspect/Edit opens the **Channel Inspector** pane.

Rename allows to rename the animation channel.

Add/create new , like the “**+**” button, creates a new channel and adds it to the list.

21.6.2 The channel inspector


Channel Inspector

Name	arms and chest
⊟ Joints	Bip01; Bip01 Pelvis; N
Bip01	☑
Bip01 Pelvis	☑
Bip01 L Leg	☐
Bip01 L Leg1	☐
Bip01 L Foot	☐
Bip01 L Toe0	☐
Bip01 L Toe01	☐
Bip01 L Toe02	☐
Bip01 R Leg	☐
Bip01 R Leg1	☐
Bip01 R Foot	☐
Bip01 R Toe0	☐
Bip01 R Toe01	☐
Bip01 R Toe02	☐

The **Channel Inspector** pane shows the details of the currently selected channel.

Name

Shows the name of the currently selected animation channel. The name can be edited in order to rename the channel.

Joints

Select for each individual joint whether it will be a member of the channel.

When the game code plays an animation on this channel, the animation will only affect joints that are a member of the channel.

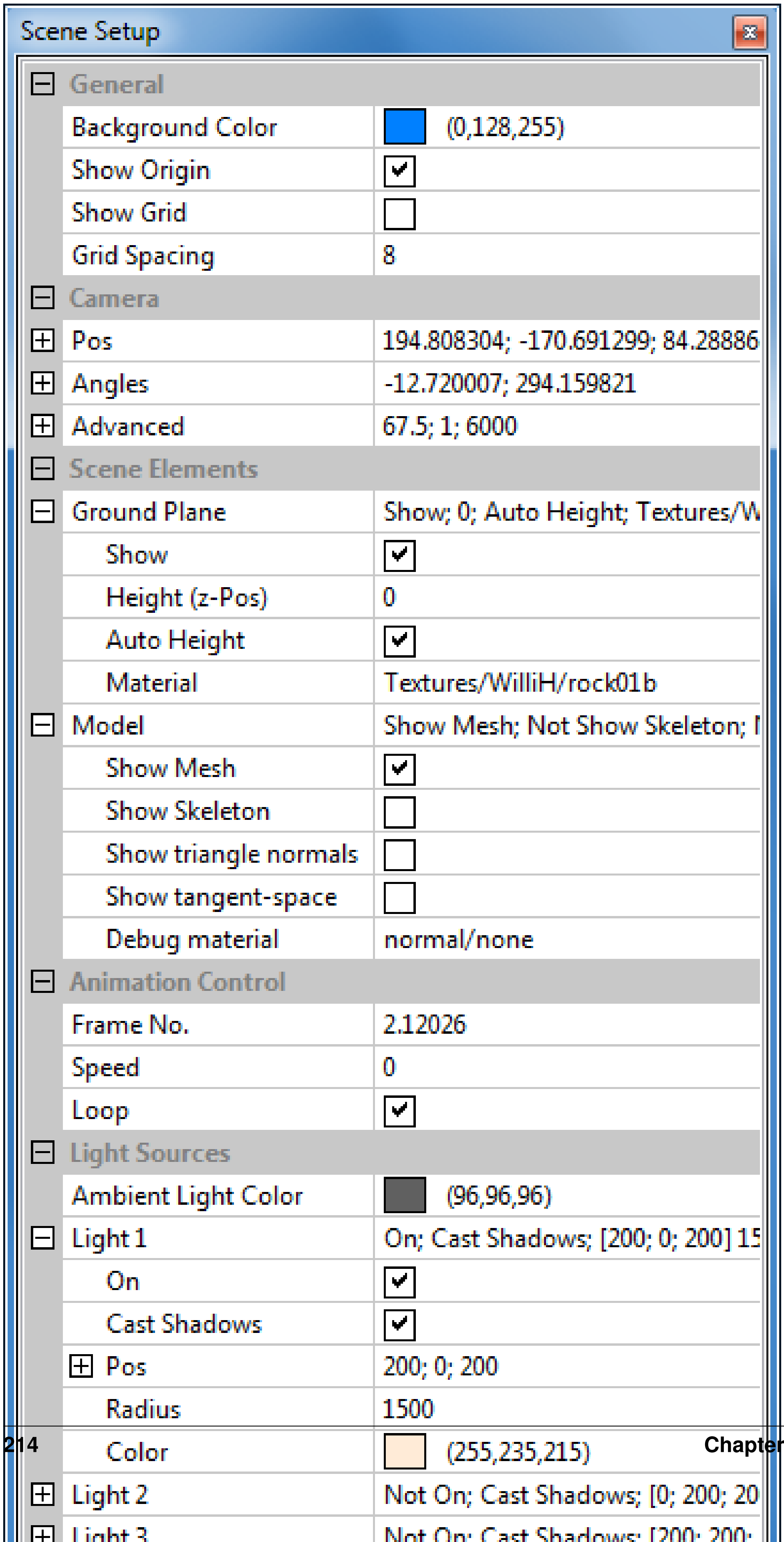
CHAPTER 22

Program Dialogs

22.1 Scene Setup

One of the Model Editor's main fields of application is to let you visually inspect the model as rendered by the Cafu Engine and as it will appear in the game.

For this purpose, the **Scene Setup** dialog lets you change the details on how the Model Editor renders the model in the 3D view. Refer to section *The Main Window* for more information about the 3D view itself.



Scene Setup

General	
Background Color	🟦 (0,128,255)
Show Origin	✓
Show Grid	☐
Grid Spacing	8
Camera	
Pos	194.808304; -170.691299; 84.28886
Angles	-12.720007; 294.159821
Advanced	67.5; 1; 6000
Scene Elements	
Ground Plane	Show; 0; Auto Height; Textures/W
Show	✓
Height (z-Pos)	0
Auto Height	✓
Material	Textures/WilliH/rock01b
Model	Show Mesh; Not Show Skeleton; I
Show Mesh	✓
Show Skeleton	☐
Show triangle normals	☐
Show tangent-space	☐
Debug material	normal/none
Animation Control	
Frame No.	2.12026
Speed	0
Loop	✓
Light Sources	
Ambient Light Color	⬛ (96,96,96)
Light 1	On; Cast Shadows; [200; 0; 200] 15
On	✓
Cast Shadows	✓
Pos	200; 0; 200
Radius	1500
Color	(255,235,215)
Light 2	Not On; Cast Shadows; [0; 200; 20
Light 3	Not On; Cast Shadows; [200; 200;

22.1.1 General

Background Color sets the "sky" or background color. Future versions of the Model Editor will allow to load cube maps for proper sky environment rendering as well.

Show Origin determines whether the principle axes of model space, emanating from the origin, are shown.

Show Grid , when selected, renders a spatial coordinate grid, centered at the origin.

Grid Spacing sets the spacing between grid lines.

22.1.2 Camera

Camera attributes are rarely entered manually: Normally, you navigate the 3D view as described at section *The Main Window*, and the camera details update automatically.

- **Pos** shows the x-, y- and z-coordinates of the camera position.
- **Angles** shows the orientation of the camera as angles for **Pitch** and **Yaw**.
- The **Advanced** settings control the shape of the camera's view frustum: **vertical FOV** is the vertical field-of-view, and **near/far plane dist** is the distance of the near/far clipping plane relative to the cameras position.

22.1.3 Scene Elements

Ground Plane

Show determines if the ground plane is shown at all.

Height (z-Pos) sets the z-position of the ground surface.

Auto Height , if set, automatically adjusts the z-position of the ground surface to the dimensions of the model.

Material sets the material that is used to render the ground plane.

Model

Show Meshes determines whether the meshes of the model are rendered.

Show Skeleton , if activated, renders the joints and the "bones" of the model.

Show triangle normals , if activated, renders the normal vector of each triangle. The color of the normal vector indicates the Polarity of the triangle.

Show tangent-space , if activated, renders the axes of tangent space at each vertex of each mesh.

Debug material allows to override the mesh materials of the currently selected skin with "debug" materials that facilitate the inspection of the model: **plain (white)** is useful for inspecting the shades resulting from lighting, **wire-frame** shows the individual triangles that the meshes are composed of.

22.1.4 Animation Control

Frame No. is the frame number in the currently playing sequence. It advances automatically if an animation is playing, but can be set manually as well.

Speed is the relative speed with which the animation is currently playing. It's automatically set to 0 or 1 if you press the Play or Pause buttons in the toolbar.

Loop , if set, forces the Model Editor to play the current animation in an infinite loop.

22.1.5 Light Sources

Ambient Light Color is the color of the ambient light. Ambient light is also there when all other light sources have been turned off.

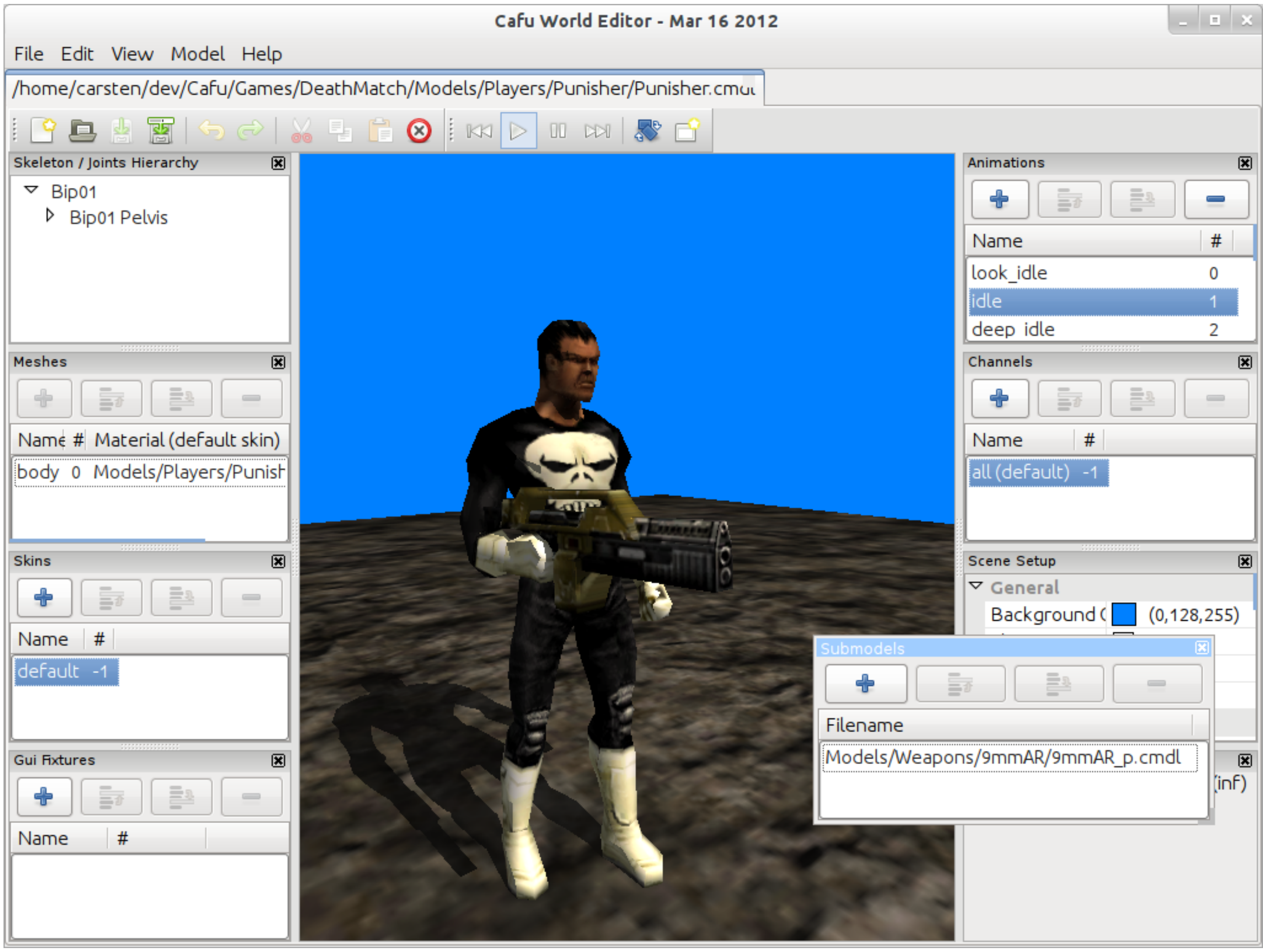
Light 1, 2, 3 are dynamic light sources as they can also be set in the Map Editor and occur in the game:

- **On** determines whether the light source is on or off.
- **Cast Shadows** sets if the light source casts shadows at all.
- **Pos** the position of the light source.
- **Radius** the radius of the light source.
- **Color** the color of the light source.

22.2 Submodels

Submodels are models whose skeleton is (partially) aligned to the skeleton of their parent model.

The most prominent examples for submodels are weapon models that are combined with player models: Each player model can be combined with any weapon model, and the animation sequences of either combine naturally with each other:


Cafu World Editor - Mar 16 2012
File Edit View Model Help
/home/carsten/dev/Cafu/Games/DeathMatch/Models/Players/Punisher/Punisher.cmdl
Skeleton / Joints Hierarchy
Bip01
Bip01 Pelvis
Meshes
Skins
Gui Fixtures
Animations
Channels
Scene Setup
General
Background (0,128,255)
Submodels

Name	#	Material (default skin)
body	0	Models/Players/Punish

Name	#
default	-1

Name	#
look_idle	0
idle	1
deep_idle	2

Name	#
all (default)	-1

Filename
Models/Weapons/9mmAR/9mmAR_p.cmdl

It's typically the job of the game code to load and combine parent models with their submodels, but the Model Editor implements this feature as well so that you can inspect loaded submodels easily:

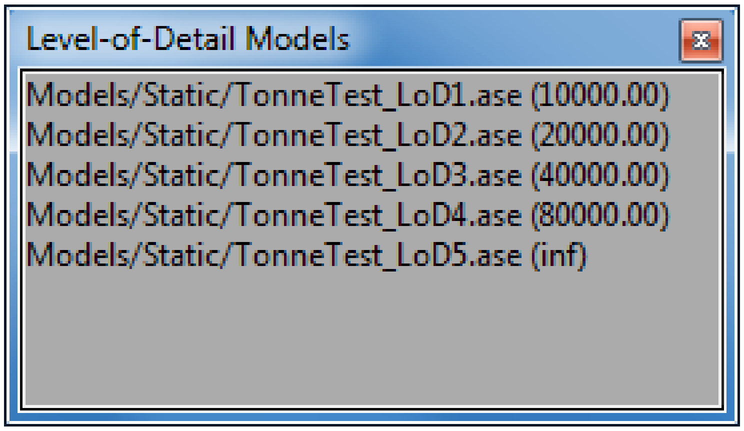
Press the "**+**" button in the **Submodels** dialog in order to load a new submodel.

In theory, you can load more than one submodel per parent model, although that only makes sense if you load e.g. different weapon models for the left and right hands of the player, or child monsters that are carried in different locations in the parent monster, etc.

The "**-**" button unloads the selected submodels again.

22.3 Level-of-Detail Models

Level-of-detail models are a kind of "meta" models: They consist of two or more "normal" models, but only one of the referenced models is shown at any time, depending on the distance to the viewer.

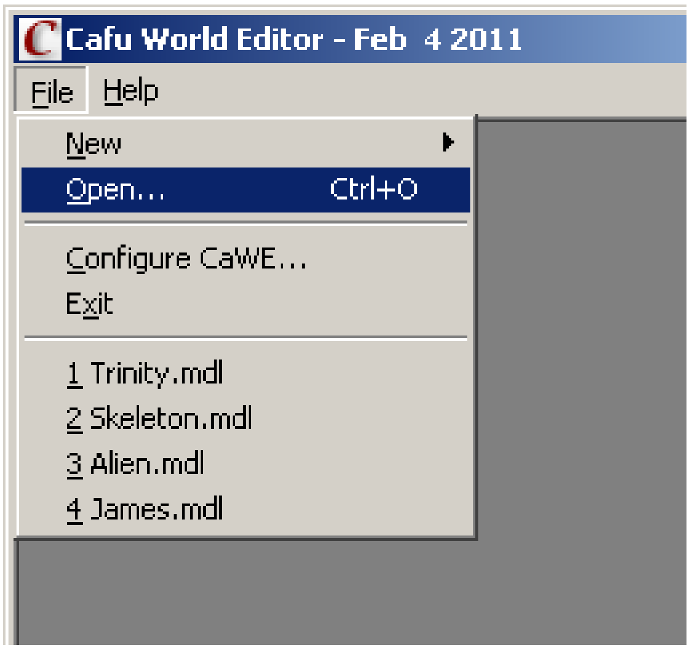

Level-of-Detail Models

Models/Static/TonneTest_LoD1.ase (10000.00)
Models/Static/TonneTest_LoD2.ase (20000.00)
Models/Static/TonneTest_LoD3.ase (40000.00)
Models/Static/TonneTest_LoD4.ase (80000.00)
Models/Static/TonneTest_LoD5.ase (inf)

The key idea is to render a detailed model only when the player (the camera) is close to it. When the camera gets farther away, a less detailed model is rendered. And when the camera gets even farther away, an even less detailed model is used.

This technique, while not without its downsides, is a common approach for reducing the computational effort required for rendering a model relative to the amount of detail that the player can perceive of distant objects.

22.3.1 Loading LoD models



Level-of-detail models have a file suffix of `.dlod`, and you open them in the Model Editor like `.cmdl` models or any other model files.

The **Level-of-Detail Models** dialog will show the list of “real” models that the LoD model is referring to, and up to which viewer distance each model is used.

22.3.2 Inspecting and saving LoD models

In the list and inspector panes, the Model Editor will only show the elements of the *first* (closest, most detailed) referenced model, even if the camera is moved far enough away so that in the 3D view one of the lower detailed models is rendered.

If you attempt to save a `.dlod` model, the saved file will be the first referenced model as well. That is, it will be treated like an individual, independent model, not as the whole set of models in the `.dlod` file.

As a result, you should load LoD models only for inspecting them in the 3D view. In order to edit the component models the `.dlod` file is referring to, you should load them directly by their true name, each one separately from the other.

22.3.3 Creating LoD models

`.dlod` files cannot be created in the Model Editor. They are simple text files that you have to create yourself, using a programmers text editor.

Here is an example, taken from `Games/DeathMatch/Models/Static/TonneTest.dlod`:

```
TonneTest_LoD1.ase  10000
TonneTest_LoD2.ase  20000
TonneTest_LoD3.ase  40000
TonneTest_LoD4.ase  80000
TonneTest_LoD5.ase
```

Each line lists a concrete model and the distance up to which the model is rendered.

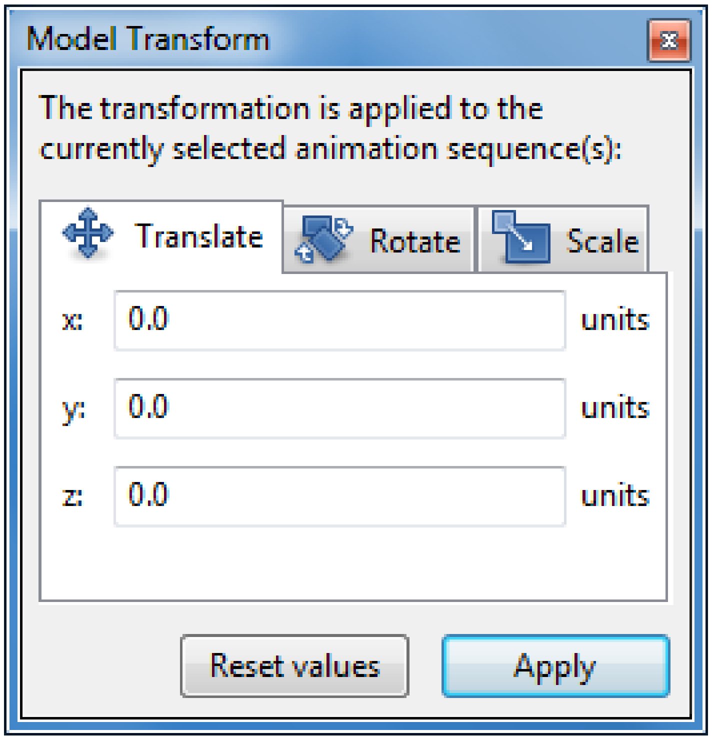
- The distance for the last, farthest model is optional. If given, it is ignored: the last model is used up to infinity.
- The file names are relative to the path of the parent (`.dlod`) file. Valid examples include `a/x.cmdl` and `..\y.ase`, whereas `c:\z.mdl` is invalid.
- The numbers are given in Cafu world units (which are currently *not* the same as CaWE world units: they are only 1/25.4 of the Cafu units!). In Cafu worlds, each unit is 1 millimeter. 1000 units are one meter or 3.2808 feet.
- In many ways, the first model in the list is "responsible" for the collective: The Cafu Engine gets the bounding boxes, number of animation sequences etc. all from the first model. Ideally though, all models should have the same BB, same (logical) animation sequences, etc.

22.3.4 An example

The image below shows three models, a simple barrel. The three models are independent of each other, and each is stored in a separate model file. They all implement the same logical object though, each at a different level-of-detail:

Combine models like these in `.dlod` files as shown above – that's all! Your LoD model is now immediately available for use in the Map Editor and the Cafu Engine.

22.4 Transforms: translate, rotate and scale



The transformation is applied to the currently selected animation sequence(s):

x: 0.0 units

y: 0.0 units

z: 0.0 units

The **Model Transform** dialog is used to translate, rotate or scale the entire model.

It is opened

- via menu item **Model** → **Transform… (Ctrl+T)**,
- or the "Transform" button in the *application toolbar*.

Contrary to the *Joint Inspector*, the **Model Transform** dialog can *not* be used to transform individual joints in a skeleton, a feature that is only useful for static, non-animated models.

The virtue of the **Model Transform** dialog is that it affects the entire model (the whole skeleton), and that the transformation is applied to all currently selected animation sequences. If no animation sequence is selected at all, the default pose (also called the "bind pose") is transformed.

22.4.1 Purpose

The **Model Transform** dialog's main area of application is to adjust newly imported models to the Cafu coordinate system:

An authoring program might have exported a model in a scale or default orientation that does not agree well with the Cafu coordinate system, or just is different from other models that you already have.

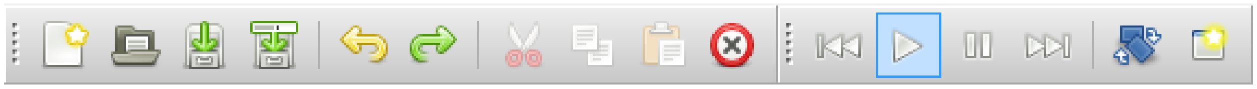
Use the **Model Transform** dialog to fix such and similar cases easily.

CHAPTER 23

Menu and Toolbar Reference

This section lists the menu and toolbar items that are available in the Model Editor.

23.1 The Toolbar



New and **Open**: In the Model Editor, as new models cannot be created from scratch, these two buttons serve the same purpose: They open the dialog for loading or importing model files.

Save Saves the model under the current name.

Save As... Saves the model under a new name.

Undo Un-does the previously made change to the model.

Redo Re-does a previously undone change.

Cut , **Copy** and **Paste**: These three are not yet implemented in the Model Editor.

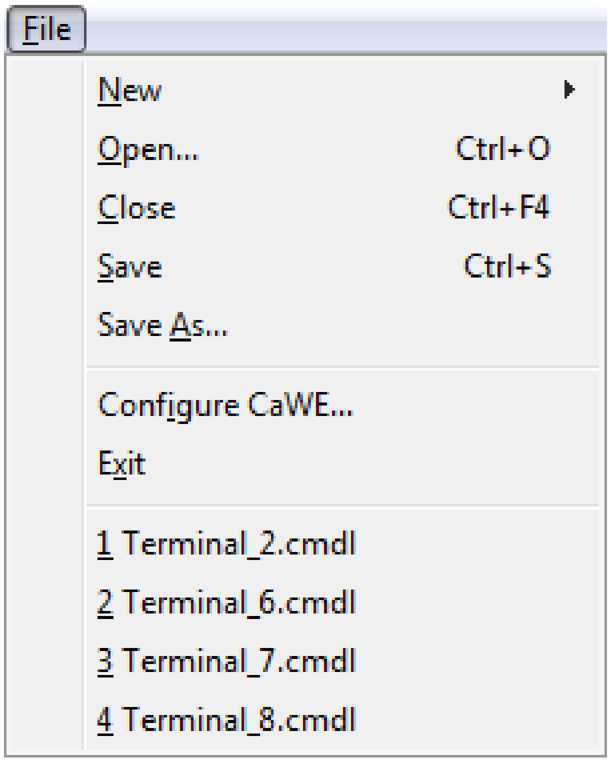
Delete If the currently selected model element can be deleted, this button deletes it.

Animation playback play or pause the currently selected animation sequence, or switch to the previous or next available sequence in the animations list.

Transform model Opens the *Transform Model* dialog.

Add GUI fixture Adds a new *GUI fixture* to the model.

23.2 The File menu



New Opens the dialog for creating a new map or new GUI, or for importing a model.

Open. . . Opens the dialog for loading or importing a map, a GUI or a model.

Close Closes the currently active file.

Save Saves the file under the same name.

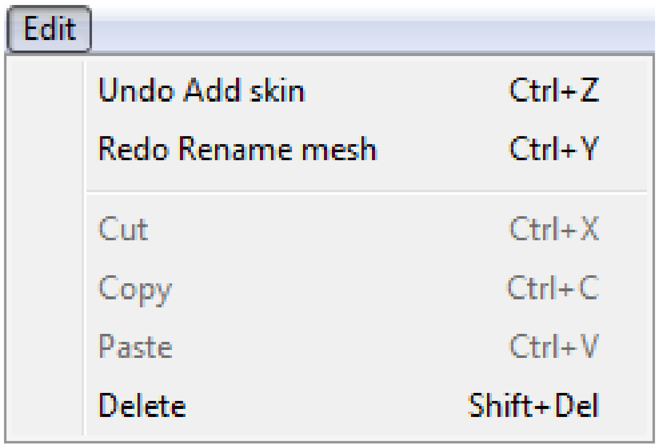
Save As. . . Saves the file under a new name.

Configure CaWE. . . Opens the *Configure CaWE* dialog.

Exit Checks if all modified files have been saved and quits the application.

Recently opened files This part of the menu lists the most recently opened files. Select one to open it.

23.3 The Edit menu



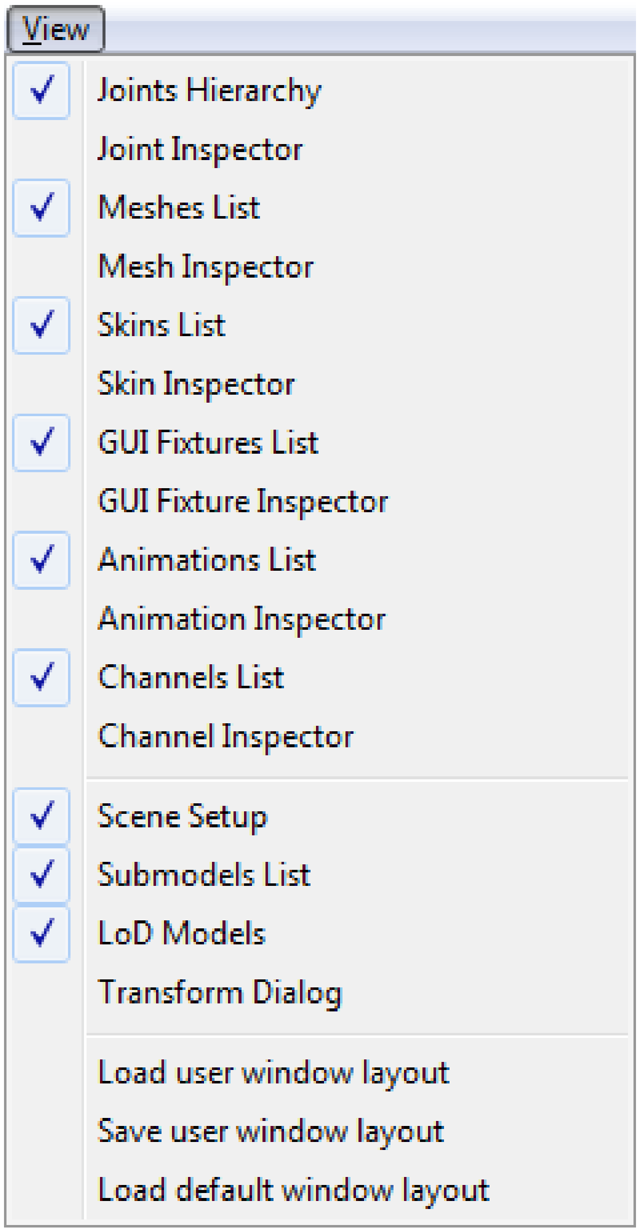
Undo Un-does the previously made change to the model.

Redo Re-does a previously undone change.

Cut , **Copy** and **Paste**: These three are not yet implemented in the Model Editor.

Delete If the currently selected model element can be deleted, this button deletes it.

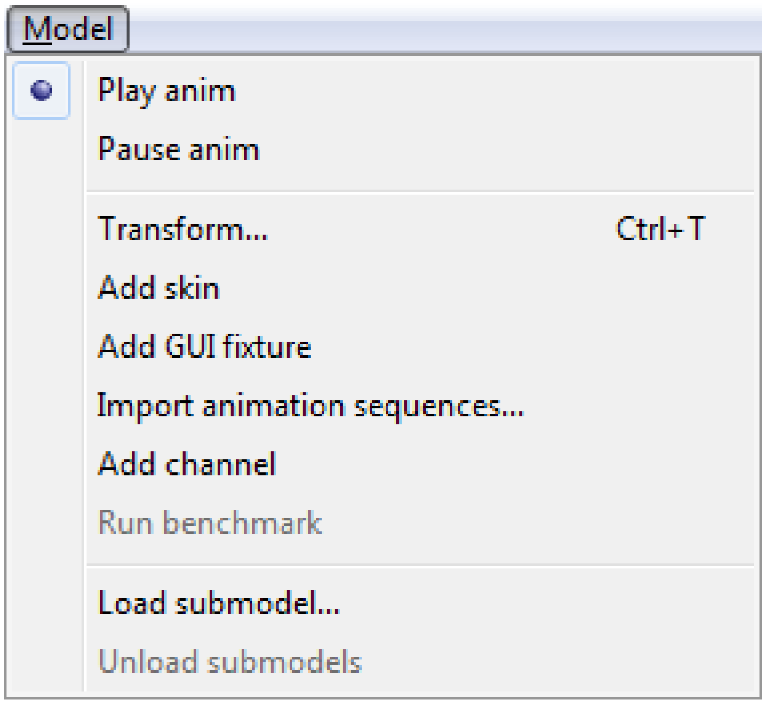
23.4 The View menu



The **View** menu items open and close the dialogs and panes of the Model Editor: Check an item to show the related pane, and un-check the item to hide it again.

This is especially useful if you've accidentally closed a pane that you want to bring up again.

23.5 The Model menu



Play anim and **Pause anim**: Like the related toolbar buttons, play and pause the currently selected animation sequence.

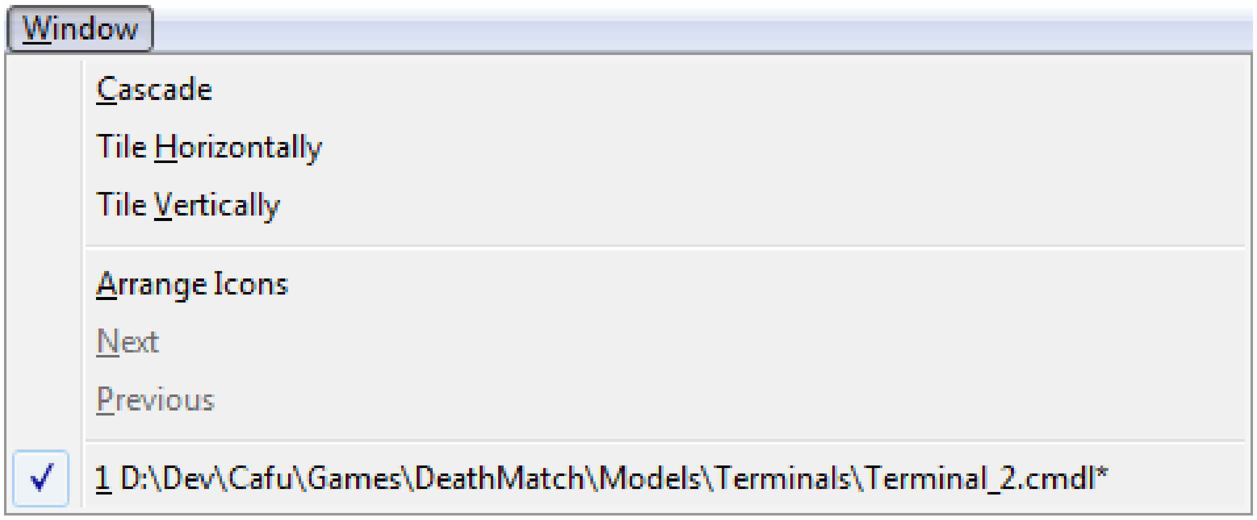
Transform... Opens the *Model Transform* dialog.

Add... or **Import...**: These items are the menu equivalents to the "**+**" buttons in the model element lists: The add or import a new model element of the specified type.

Run benchmark This is not yet implemented.

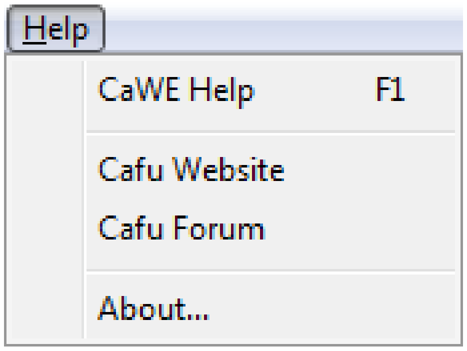
Load submodel... and **Unload submodel**: Loads and unloads submodels, just like the "**+**" and "**-**" buttons of the *Submodels* dialog.

23.6 The Window menu



As in all of the Map, GUI and Model Editors, the **Window** menu can be used to arrange and change to the currently opened windows.

23.7 The Help menu



As in all of the Map, GUI and Model Editors, the **Help** menu opens the web browser with related websites for help and support, and shows the CaWE **About** dialog.

CHAPTER 24

Model Files Explained

Each model comprises several files on disk, such as the model file, material definition files, and texture images.

When you save a model in the Model Editor, the editor creates or re-creates some of these files, updates others, and leaves alone the rest. The result is normally exactly what you want and expect, but sometimes you may wish to hand-tune some details (such as the material definitions) without and independently of the Model Editor. In such cases, it is very helpful to understand the files that belong to a model.

This section explains the files that together form a model and how they relate to each other.

24.1 One directory per model

Although not a technical requirement and not enforced by the Cafu model code, it is highly recommended to save each model in a directory of its own. This

- explicitly groups all files that logically form and belong to the model, and
- makes packaging a complete model in a `my_model.zip` archive possible, so that the model can easily and safely be distributed, shipped and handled.

The name of the directory should match the file name of the model. That is, if your model's name is `Trinity.cmdl`, it should be stored in a directory with the same name `Trinity/` (or in a zip archive with the same base name `Trinity.zip`).

Note that when you are saving a new model that does not yet have a separate directory, you can use the "New Folder" button (or right-click context menu) of the "Save" dialog to create such new directories as required.

Example:

```
d:\Dev\Cafu\Games\DeathMatch\Models\Players> dir Trinity
ChromeBuckle_diff.png
ChromeGlass_diff.png
Pants_diff.png
Skin_diff.png
Trinity.cmat
```

(continues on next page)

(continued from previous page)

```
Trinity.cmdl
Trinity.cmdl_bak
Trinity_editor.cmat
```

24.2 cmdl model files

The `cmdl` file is the main file of the model: It contains the definitions for the skeleton, the meshes and the animation sequences of the model.

The Cafu Engine and the Model Editor load `cmdl` files as Lua scripts, and as such they can be inspected or edited in a text editor if desired.

Besides the main `cmdl` file, the Model Editor also creates a `cmdl_bak` backup file that contains the contents of the `cmdl` file before it was last saved. (If you ship your model as a `zip` archive, the `cmdl_bak` file is typically redundant and can be omitted from the archive for space efficiency.)

Example:

```
d:\Dev\Cafu\Games\DeathMatch\Models\Players> dir Trinity\*.cmdl*
Trinity.cmdl
Trinity.cmdl_bak
```

24.3 cmat material definition files

The `cmat` files contain the material definitions for the meshes of this model.

The main `cmat` file must have the same base name as the model ("Trinity"), so that the full name is for example `Trinity.cmat`. This file is for your hand-written material script code, if any, and is never touched or overwritten by the Model Editor (with one exception, see below).

The Model Editor also writes a secondary `cmat` file whose name ends with `_editor.cmat`. This file is written anew each time the model is saved, and contains material definitions whose script code was not hand-crafted, but who were created or edited in the Model Editor.

Example:

```
d:\Dev\Cafu\Games\DeathMatch\Models\Players> dir Trinity\*.cmat
Trinity.cmat
Trinity_editor.cmat
```

The two `cmat` files are linked as follows: When the Cafu model code loads the materials of a model, it opens the main `cmat` file (`Trinity.cmat`). This file contains a statement like

```
dofile("Trinity_editor.cmat");
```

in order to include and process the secondary `_editor.cmat` along with the main file.

The only exception when the Model Editor touches the main `Trinity.cmat` file is when the file does not yet exist, or doesn't contain the `dofile()` reference to the editor file. In this case, the `_editor.cmat` would not be loaded at all, and thus the Model Editor inserts the `dofile()` line into the main `cmat` file.

In summary, the goal of keeping two separate `cmat` files that are linked as described above is to keep your hand-crafted material script code and the Model Editor edited material definitions cleanly separated, without any danger of one overwriting the other.

24.4 Texture images

The texture images are referenced from the material definition scripts. See the documentation about the Cafu Material System for more details.

Example:

```
d:\Dev\Cafu\Games\DeathMatch\Models\Players> dir Trinity\*.png Trinity\*.jpg
ChromeBuckle_diff.png
ChromeGlass_diff.png
Pants_diff.png
Skin_diff.png
```

CHAPTER 25

Dependencies among Models

Although it might not be obvious at a first glance, there are several dependencies among certain models. It is worthwhile to have knowledge about these issues before you start making own models, because it helps with resource planning and prevents expensive problems later.

We'll describe several typical kinds of dependencies, considering the example of the Cafu DeathMatch MOD: In the Cafu DeathMatch MOD, mutual dependencies affect the human player models and their weapons.

First, lets deal with all "other" models, i.e. those that are neither player nor weapon models: Usually, these other models are not closely related to each other, each of them has a separate piece of game code associated that handles it, and thus they do not suffer from any inherent dependency problems.

25.1 Human player models

Human player models are special, because they are usually intended to be 100% equivalent to each other. That is, if somebody makes a new human player model and offers it for download, you expect it to work exactly like the ones that you already know. In order to achieve this kind of equivalence, two assumptions must hold: The skeleton of the new models must basically match the skeleton of the old models, and the animation sequence numbers must refer to reasonably identical animations.

The animation sequences must match, because the game code has built-in knowledge that, for example, sequence number 3 refers to an "idle (waiting)" animation, and that sequence number 27 refers to an "aiming with a shotgun" animation. It is entirely up to you to animate your model to look at his wrist watch while aiming with the shotgun, or to pick his nose, but it *is* important that sequence number 27 corresponds to some "aiming with a shotgun" animation – because the same is true for all other models, and the game code relies on it.

The skeletons must also match, for similar reasons: At least the basic hierarchical structure (starting from the pelvis to the most important bones) must be identical, as well as the *names*(!) of the corresponding bones. However, you are free to add bones to the skeleton as you like, change their sizes or lengths, change their positions relative to each other, and do many other interesting things. You may even omit bones if you want to create a one-armed, one-legged hero. What works and what works not is easiest determined by trying it out, but please do also refer to the next part about weapons.

25.2 Weapons

Weapons do usually come as a set of three models: “world” models, “player” models, and “view” models.

World models are the models that lie around in the world, before someone picked them up. They are usually not animated (or only have a single animation sequence), are independent from anything else, and are therefore in the same category as the “all others” models, so that we need not be further concerned about them.

Player models are the weapons that you see in the hands of *other* players who have picked up and are using that weapon. For the following discussion, we’ll refer to the “player” weapon model as the “`_p`” model, and to the character model of a human player as the “body” model.

First, if you consider the skeleton of a `_p` model in the Model Editor, you will find that it resembles a partial body model (the bones from the pelvis to the shooting arm are there!), before it diverges into additional bones for the actual weapon.

Here is the crucial point: In order for the engine to compute the proper position of the `_p` model relative to the body model, it (partially) has to match the skeletons of both models! That is, it first computes the skeleton of the body model (depending on its current animation pose). Then it considers the skeleton of the `_p` model, starting at its root, and tries to match it bone-by-bone to the previously computed body skeleton. If a match was found, the engine simply takes the information from the body model bone also for the `_p` model bone. Only when the matching breaks for the first time, the engine resumes normal bone computing also for the `_p` model. (This way you can for example see a face-hugger in the hands of *another* player that is wagging it’s tail.) Matches are made by comparing the *names* of the concerned bones.

As a consequence, if you want to make additional body models for the DeathMatch MOD, and additional weapons, and you want to be able to combine each body model with each weapon, *then you have to make sure that they***all***have a corresponding skeletal structure and bone names!*

View models are the models that you see in 1st persons view after you have picked up a weapon yourself. They are also independent from anything else, but the game has usually special code for handling them. Thus, you can well make a *replacement* weapon for e.g. the shotgun, matching the animation sequences of the existing “view” weapon model according to similar rules as indicated for making replacement human player models. You can also introduce *entirely new* weapon models, but be prepared that is requires to augment the game’s C++ or script code as well.

25.3 Applicability to your own game

If you create an own game, things may or may not be different, of course. However, keep in mind that if you want to achieve a high degree of flexibility and ease of maintenance, you’ll sooner or later probably experience the same rules and dependencies as described here. They are the – relatively cheap – price for the ability to combine every human player model with every weapon model.

CHAPTER 26

Supported file formats

The Cafu engine currently supports 4 different image file formats: **.jpg, .png, .tga and .bmp**. This documentation shows you the advantages/disadvantages of those file formats.

.png file format: The .png file format is the optimal file format for Cafu! It can feature transparent parts that allows them to be used for textures that have invisible parts (like windows). Advantage: The transparent parts can be created very easily with image editing software. There is NO loss of quality at all with this compression - optimal for high detailed textures. Disadvantage: They are bigger in the size than jpg on the hard drive.

.jpg file format: The .jpg image format is very small in size on the hard drive because of it's compression. jpg-files can be used for diffuse maps - but with care - dont compress them too much! The disadvantages are that they can't handle transparent parts, which makes them unusable for some tasks and the compression leads to a loss of quality. Don't use it for high frequency textures!

.tga file format: The .tga file format is like the .png format - the only difference is the size - tga files are bigger in size on the hard drive than png!

.bmp file format: This file format is very large in size and can't feature transparent parts. It's advantage is it's quality - there is no loss of informations when saving into this format!

Summary: Use png files whenever it's possible. The alternative for textures without alpha channles is bmp, for those with it's tga. Try to avoid jpg for high frequency textures and normal-maps!

CHAPTER 27

Texture types

Most surface materials that occur in Cafu are defined by the mathematical composition of several texture images, where each texture image describes another aspect of the material. The most common texture image types are explained here.

In almost all cases, you start making a new material with the **diffuse-map**. Diffuse-maps are almost always created manually in an image processing program, e.g. hand-painted or delivered from a photograph, similar to the way “old-style” textures were made in the past. The diffuse-map shows material color when the surface is diffusely lit. Thus, you should not draw any hard shadows into the diffuse-map - they are automatically created by the engine later from the information that comes from the normal-map and the light sources. You may however, in some cases, draw some soft shadows into the diffuse map, a sort of “reachability factor” (“How hard is it for the light to reach a certain spot on the texture?”). This implies that a diffuse-map that shows for example corrugated metal or a rough rock surface could be an image that only has a single shade of grey! Also note that the final material when rendered by the engine tends to look best when you use pixel values of medium brightness. Moreover, high-contrast and high-frequency components should be used with care in diffuse-maps, as such components often interfere with normal-maps later, compromizing the effect of dynamic lightning. Also the specular highlights might look strange with such diffuse-maps.

Specular-maps (sometimes also called gloss-maps) define the shininess of the material. They are conveniently created together with or derived from their diffuse-map. Bright values mean that the material is very shiny, dark values mean that the material is mat. Note that specular-maps are not limited to gray-scaled images: Their tone (color) modulates with the color of the light source. Specular-maps often have the strongest impact on dynamic lightning. Note that for many materials that only have diffuse light reflection characteristics (e.g. sandstone), specular-maps can often be omitted entirely.

Luminance-maps define the light that a texture emits. As with specular-maps, they are easiest created together with their diffuse-map, and often very simple in nature. The light of luminance-maps is local to the texture, and does not cast on any other surfaces or objects. Typical occurances for luminance-maps are with LED panels or computer-screens, but frequently they are not present at all, because most materials do not actively emit light themselves.

Height-maps (also called bump-maps) are gray-scale images that define the height of a surface: dark is low and white is high. They often only serve as an intermediate product for creating normal-maps. In fact, Cafu converts all height-maps to normal-maps internally before use. Some people convert the diffuse-maps to gray-scale images in order to obtain height-maps, but this does almost always yield in bad quality - this is just a lazy trick that one should never use. Instead, height-maps should be painted or created from scratch. This is almost always a very difficult task though, and works best with either natural or organic materials or high-frequency components like scratches, dents, and so on. Another method is to obtain height-maps from the depth buffer information of some rendered geometry. In this case, however, I'd recommend to skip height-maps entirely, and render normal-maps directly.

Normal-maps are the most important component in dynamic lightning. They contain information about the shape of the surface by color-encoding the surfaces normal vectors. They are normally never hand-made, but either derived from height-maps or created from true 3D geometry. For example, plug-ins for Photoshop as well as for TheGimp exist for converting a height-map into a normal-map. For normal-maps that represent technical features like the example panel above, the best results are achieved by creating them from 3D geometry though, which for example is possible with 3D Studio Max. Please note that combining diffuse-maps and normal-maps that both have high-frequency components (and the diffuse-map possibly highly contrasting colors) tends to compromize the effect of dynamic lightning.

CHAPTER 28

Skybox creation (Environmental Map, Cubemap)

A **skybox** is a special set of textures that is used to display the sky. Cafu uses skyboxes to create skies. A skybox consists of **6** textures. Try to imagine a map that is surrounded by a box. The walls are covered with the sky textures. Sky + box = Skybox ;)

This documentation focuses at first on the integration process into Cafu and displays later, how to create a skybox in general!

Here is an example of what those 6 sky textures can look like. If you have some knowledge in geometry, you'll notice that the textures are in this example put next to each other in form of an unfolded cube to demonstrate how this skybox system works.

(Click on the picture to enlarge it)

28.1 How to integrate a skybox into Cafu

The textures can have the dimensions *256×256*, *512×512* and *1024×1024* (other dimensions don't work or don't make sense). It is important that they are seamless when being put together, otherwhise you have viual glitches ingame.

If you have a close look at the picture, you can see that it includes the names of the textures. As you can see, their names begin with sky1 but end with keys like _py. The endings tell the engine which texture demonstrates which site, sky1 is just the name of the sky used in this tutorial. Like this, it is important to name your textures right. Here is a small overview about which ending stands for which direction:

(FIXME!) This passage needs to be filled with content

- _nz :
- _nx :
- _pz :
- _px :
- _py : The upper side
- _ny : The lower side

Once named your sky textures right, you can place them in the skybox directory that can be find in the textures directory.

Then, you have to create a shader file for the sky. If you don't know what a shader is, please have a look at this desciption.

In the case you named your sky sky1, the shader file would look similar to this:

```
Textures/SkyDomes/sky1
{
AmbientShader A_SkyDome
LightShader   none      // == noDynLight keyword
// The '#' in the next line is auto-replaced with the relevant suffixes (_px, _ny, ...
↪).
cubeMap Textures/SkyDomes/sky1#.jpg, wrapS clampToEdge, wrapT clampToEdge
ambientMask d           // Don't write into the z-Buffer, so that entities (like␣
↪missiles) outside of the map can still be drawn.
noShadows               // This material does not cast dynamic shadows.
meta_noLightMap         // Don't create or keep lightmaps for this material, don't␣
↪participate in Radiosity computations.
meta_sunlight           // This keyword states that this material casts sunlight.
    ( 2  4  6)          // The irradiance of the sunlight in Watt/m^2 that comes (or␣
↪shines) through this material. Values (100  90  80) might work, too.
    (-2 -5 -9)          // The direction of the incoming sunlight rays. The z-
↪component should be negative. These values match the actual position of the sun in␣
↪the cube-maps.
}
```

Basically, **line 1, 7, 13 and 14** are important.

- In line 1, you have the textures path for CaWE, basiacally you just have to change sky1 into your sky name.
- In line 7, you have to change the path so the sky name (in this case sky1) is right and the texture format (in this case .jpg).
- In line 13, the shader defines the colour of the sunlight (there's a description for this values later in the tutorial).
- In line 14, the shader defines where the sun is in the sky (important for light calculation in maps, a descriptions can be found later in this tutorial).

When finished, save your cmat shader file and you can find your sky in CaWE. Note: the first line of the shader (*Textures/SkyDomes/sky1*) is also the name of the sky in CaWE.

That's it!

28.1.1 The irradiance of the sunlight

(FIXME!) This passage needs to be filled with content

28.1.2 The direction of the sunlight

(FIXME!) This passage needs to be filled with content

28.2 How to create a skybox with Terragen

Terragen is a terrain generator that can also create skyboxes. The skyboxes included in Cafu were created with Terragen. There are various resources in the internet that cover skybox creation with Terragen. Here are some recommended links:

- Official Terragen website
- Creating Environment Maps with Terragen, tutorial on gamedesign.net
- Static Skies Tutorial with Terragen, tutorial on 3DNA

28.3 How to create a skybox with SkyGen

SkyGen is a simple web based (WebGL) skybox generator.

- SkyGen

28.4 How to create a skybox out of photos

(FIXME!) This passage needs to be filled with content

28.5 How to create a skybox out of panoramic images

(FIXME!) This passage needs to be filled with content

CHAPTER 29

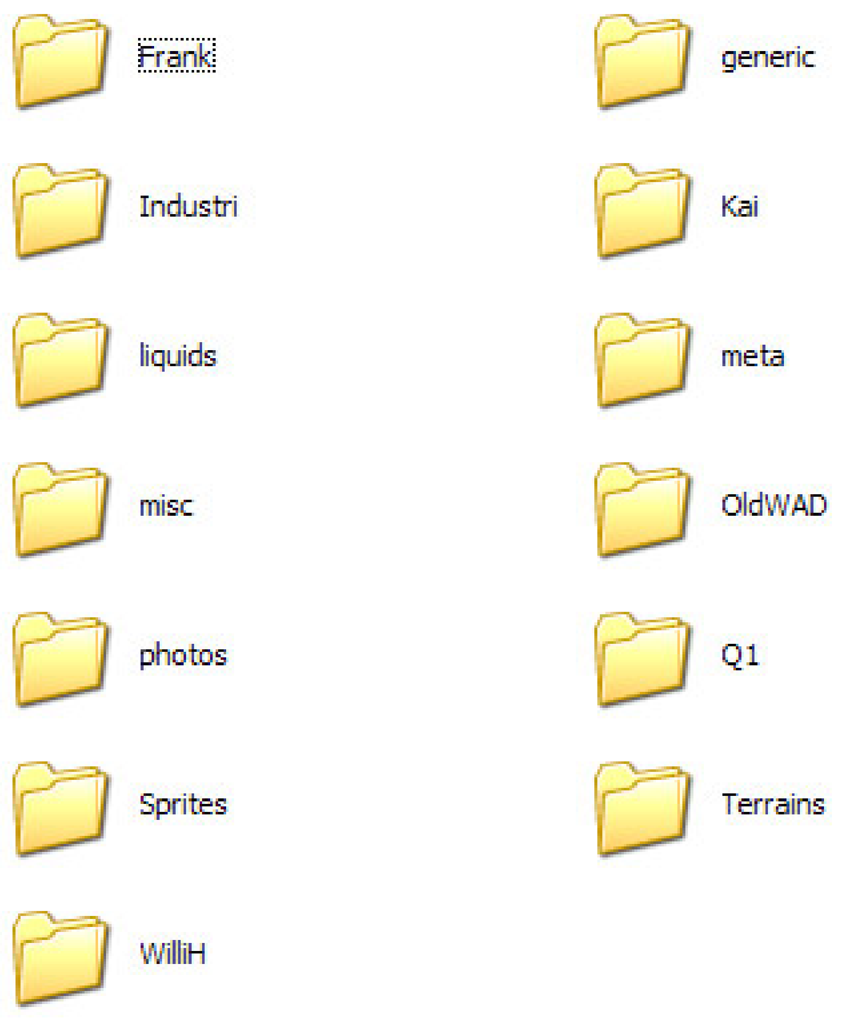
Using own textures

This tutorial has 3 part. The first part covers textures for maps, the second part covers skins for models and the third part skybox textures.

29.1 Part 1: Map textures

Create your texture. Then save it into one of the supported file formats (those are: .jpg, .png, .tga, .bmp).

Go into your Cafu directory and from there into the textures directory: **Games** → **DeathMatch** → **Textures**. There it should somehow look like this:



Now create a new directory, I call mine “tutorial”. Go into it and place your texture there. I call my texture “texture1”, it consists of a diffuse map, a normal map and a specular map. I add endings to the different textures to exactly know which texture is which type. My diffuse maps have a _diff tag in their name, normal maps get _norm, specular maps get _spec and luminance maps _lum (not necessary to give them those tags though).

OK, that was the first half, what we now have to to is to write a so called “shader” file, a text file including names and paths of textures that gives the informations about the textures, e.g. if they have normal maps and so on or if they have transparent parts.

Let’s go back to the DeathMatch directory. There you can find the “Materials” directory in which all shader files are placed. They can be opened with simple text editors like notepad. If you open for example the “generic.cmat” file, you can see that it contains informations about various textures. The system works like this:

```
Textures/directory_name(in which the textures are palced)/texture name
{
    type_of_texture Textures/directory_name/texture_patch
}
```

The best way to describe those shaders is to actually have a look at them. Here is the shader for my texture:

```
Textures/tutorial/texture1
{
```

(continues on next page)

(continued from previous page)

```
    diffusemap Textures/tutorial/texture1_diff.jpg
    normalmap Textures/tutorial/texture1_norm.jpg
    specularmap Textures/tutorial/texture1_spec.jpg
    lightmap $lightmap
}
```

If you have a luminance map you would have to write "lumamap" (without quotes).

It's easy, isn't it? And that's already it! You can save it as yourshadername.cmat or within another shader and it will work. Start your mapping editor and you will be able to use the texture!

Download: Tutorial example files (.zip file) **Download:** Tutorial example files (.tar.gz file)

29.2 Part 2: Model textures

Using own skins (model texures) is as easy as using map textures. There are only slight differences.

Instead of being placed in a directory in the textures directory, you have to place them in one of the directories in the Models (like the models themselves) directory. Let's say we place them into the Static directory that is placed in the Models directory. Our shader-file has to be placed in the Models directory that can be found in the Materials directory. It would look like this:

```
Models/Static/mymodelname/a_name_for_my_skin
{
    diffusemap Models/Static/a_name_of_my_skin_diff.png
    normalmap Models/Static/a_name_of_my_skin_norm.png
    specularmap Models/Static/a_name_of_my_skin_spec.png

    red ambientLightRed
    green ambientLightGreen
    blue ambientLightBlue
}
```

There are only small differences: 1. you have to add a model name too and not only a texture name ("mymodelname/a_name_for_my_skin") - note that you can choose whatever you want, those names dont have to meet the model name/texture name. 2. in the lower part you can see that you have to add those red, green and blue informations

Well that's it! If there are questions, feel free to post them in the forums!

29.3 Part 3: Skybox textures

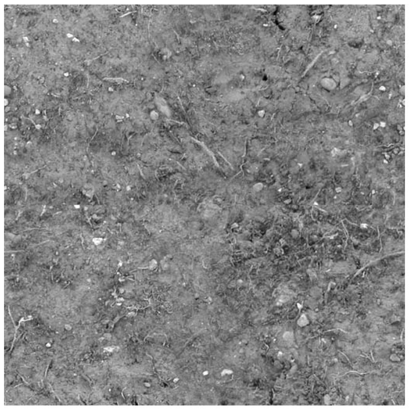
You can find detailed tutorial concerning the integration of new skyboxes here:

- *How to integrate a skybox into Cafu*

CHAPTER 30

Making “perfect” Detail-Maps

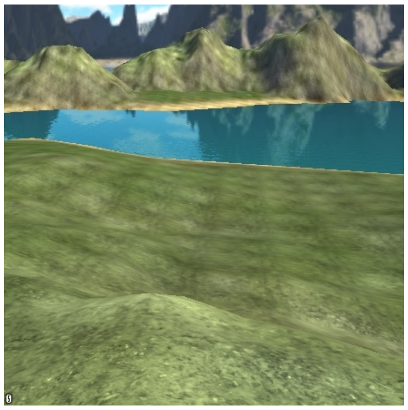
30.1 The Problem


Detail-maps, in order to work well, must meet two requirements:

1. Their average value must be 128 (50%) so that they don’t change the total brightness of the texture they’ll be applied to, and
2. they must not show tile patterns when they are minified (e.g. when they’re viewed from a great distance).

Getting 1. right is the task of the artist, and is easily done in any paint program that supports contrast and brightness adjustments and histogram views.

Getting 2. right is more difficult if not impossible for the artist. For example, I started with the original detail map shown at the right (click to enlarge). Observe that this detail-map has a big dark spot roughly in the mid of the right half.


This screenshot shows the effect of applying the detail-map to a large terrain. Note the well-visible pattern both on the near ground as well as on the far rocks. This strong pattern is caused by the non-balancedness of the detail texture.

30.2 The Idea

If we had a low-frequency graymap E that just represents the "error" in the above detail-map, we could subtract that error-map from the detail map D to just remove the error, and then add 128 (50%) to move the remaining high-frequency parts around 0 back to the average of 128. That is, the perfectly fixed detail map is found by computing D - E + 128.

30.3 The Solution

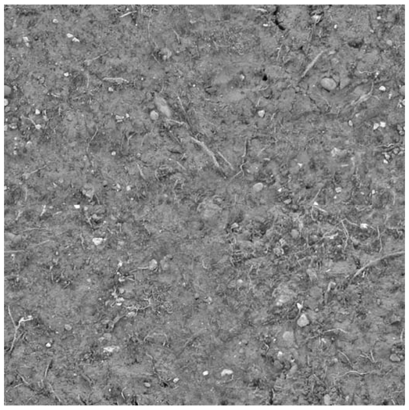
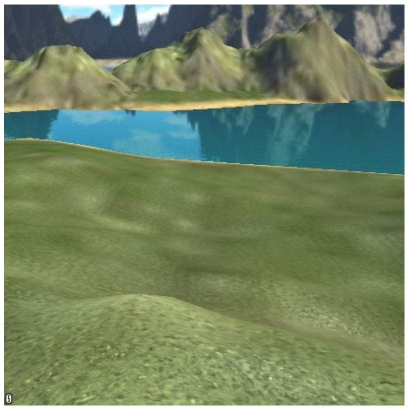
First, observe that E is easily found by blurring a copy of D! Any paint program can easily do that! I used The Gimp to apply a Gaussian blur filter to the detail map, creating test images with filter widths of 5, 20, 40 and 60 pixels respectively. The left image below shows my so obtained "E", which is the original detail-map blurred with a 60 pixels width Gauss filter!

The next problem is that computing the above D - E + 128 has a tendency to produce intermediate values that are less than 0 or greater than 255. Therefore, this operation can **not** be done with layer techniques in PhotoShop, PaintShop, Gimp, etc.!

In order to get correct results easily, I just wrote a small C++ program that reads E and D from two images files on disk, computes D - E + 128, and writes the result back into a third file. Please send me an email if you're interested in the exe file for Windows!

And yes, that's all! I applied my program to the above detail-map D and it's error-map E (left image below). The center image below shows the result. *Notice how beautifully the dark spot is gone!* In the right image you see a screenshot of the same scene as above, now with the new detail-map being applied. Note that the pattern is (almost) entirely gone!

Also please note that now you see a new "pattern" at the mountains – this is the result of a not-so-good base texture that is showing up now that the broken pattern of the detail-map has been fixed and does not any longer distract from it. This matter is not related to detail-map issues, and is just fixed by using a better base texture.

30.4 Discussion and Results

One open question is how much D should be blurred in order to obtain E. This is easily answered: As much as possible! In the above example, I used a blur filter of 60 pixels width, and the pattern as (almost) completely gone. Using smaller filter sizes reduces the rests even further. For example, I also made tests with blur filters of 5, 20 and 40 pixels width. The smaller the filter kernel width was, the more homogeneous became the end result. (Blurring not at all obviously yields the 128 planar detail map (because then E=D, and D-D+128 = 128) and has no visual impact at all!)

Therefore, the recommended strategy is to start with a really big amount of blurring, and only reduce it if that amount was not sufficient to remove the most significant non-balancedness of the original detail-map.

30.5 High-Pass Filters in PhotoShop

After I had completed this article, Kai sent me a link to this Gamasutra site: http://www.gamasutra.com/features/20010523/hajba_01.htm It deals with the same problem and is a lot more detailed than this text – definitively a recommended read! As the solution, High-Pass Filters are presented which achieve exactly the same results and which are – contrary to my above statements – well built into PhotoShop and possibly other paint programs, too. So if you're a dedicated user of your favourite paint program, you'll probably want to give the built-in high-pass filter a try before trying my home-grown solution.

CHAPTER 31

What is it? An Introduction

The Cafu Material System is the central rendering subsytem of the Cafu engine and its graphical tools like CaWE, the model viewer, the terrain viewer, and so on. It takes polygonal meshes of geometry and renders them on the screen. While doing so, it is responsible for the *materials* that the mesh surfaces show. Said differently, it defines how the rendered polygons *look*.

The Material System is *not* involved with or responsible for *what* gets rendered, *when* it is rendered, how the meshes are spatially formed or organized, etc.

31.1 Purpose, Goals and Features

The introduction of the Cafu Material System solved incredibly many problems and comes with equally many advantages. Some of them are hard to explain and/or hard to grasp, but here is a (partial) list anyway:

- Unique rendering across all programs: The Cafu engine, CaWE, the model viewer, the terrain viewer, the material viewer, ... all use the same rendering code and technology. I was finding myself writing, debugging and synchronizing the same code over and over again, for each mentioned program and for each supported 3D API, multiplying the expenses until chaos. This was one of the main reasons to start the MatSys. The savings are enormous, and I can be sure that models in the model viewer are rendered 100% identical as in CaWE or the engine.
- Unique rendering across all types of geometry: Worse than the above, I found myself implementing the same technology even several times in the same program: once for the models, once for the world polygons, once for the terrains, etc. Sharing global resources like textures or GPU programs was impossible, and the Cafu engine happended to render the world polygons with different technology than the models, the terrrains with a third, and so on. Debugging was a nightmare. That's all over now, everything is now handled by the MatSys, and the savings and the leap in improved code-design were *huge*.
- Can easily add support for new platforms, operating system, APIs. Without any danger to break existing code, and even without having to touch existing code at all, new renderers and new shaders can be introduced: If hot-plugging a new graphics board into your computer was possible, the Cafu engine could handle that situation *while running*. You can even supply and use a *completely new renderer* without stopping or recompiling the executables. Just have the engine reload the renderer, and the new technology will be immediately used.

- Users and artists get a much greater and more flexible control over the rendering, as all materials can be edited independently from the worlds or models that they are applied to.

31.2 Renderers

The MatSys is shipped in several separate modules. Each module offers the same features, but they are different in that the implementation of each is based on a different underlying technology. These modules are called the **Renderers** of the Cafu MatSys. For example, a MatSys renderer exists that is based on OpenGL 1.2, another is based on the latest programmable GPU features and others are somewhere in the middle. In fact, Cafu ships with several of them, and it uses the MatSys by actually employing one of the renderers.

Technically, renderers are dynamically loaded libraries (`.dll` files on Windows systems), and you can find them in the `Renderers/` subdirectory of the demo or SDK. When the Cafu engine or another program starts, it first scans the list of all available renderers, automatically determines the one that is the best (or most appropriate) for your system, and then loads it (this specific instance of the Material System) for use.

This way it does not matter wether your computer is very old or the latest leading-edge system, wether it uses OpenGL, Direct3D or software-only rendering or wether its OS is Windows or Linux: You always get the best possible graphical output – this is one of the goals the MatSys has been designed for!

31.3 Materials

Materials define what surfaces look like and what features a surface has, and if you are interested in Cafu editing, you'll most likely get in touch with them.

For example, mappers assign materials to world geometry in order to give all objects the look and feel that they want. Modellers often create new materials for the creatures that they create, and then assign them to the polygonal meshes of their model.

When it comes to rendering, the Cafu engine takes both the meshes and their assigned materials, and hands them to the Material System. The MatSys then does it's best to render the mesh with that material as accurately as possible on the available technology.

Each material is referred to by its name, and is defined in a *material definition script* file. These files have the suffix `.cmat` and can contain the definition of one or more materials. They are simple ASCII text files, and you can find many examples in the `Games/DeathMatch/Materials/` subdirectory of the demo or SDK.

Here is an example of a material definition from `Games/DeathMatch/Materials/Kai.cmat`:

```
Textures/Kai/3r_metpan01
{
    diffusemap  Textures/Kai/3r_metpan01_diff.png
    normalmap   Textures/Kai/3r_metpan01_norm.png
    specularmap Textures/Kai/3r_metpan01_spec.png
    lightmap    $lightmap
}
```

Using materials is very easy. For example, when you're mapping with CaWE, you apply materials to world brushes in the same way as well-known textures used to be applied in other or older map editors, too. Please see the CaWE User Guide in this Wiki for more information.

Creating materials often requires two steps: You first have to create or acquire the texture maps that are used or required by the material. These are typically png, jpg or tga images that define the diffuse-, normal-, specular-, and other component maps that will be referred to by the material. Then you have to write a material definition into a `.cmat` script file like in the example above. Very advanced materials may also involve a third step, namely in the case that they require their own unique shader (see below). The documentation in this Wiki has detailed documentation about each of these steps.

31.4 Shaders

The meaning of the word **Shaders** in the context of the MatSys is a bit different than its meaning in other contexts (the word “shader” has many meanings, after all). Shaders are objects of C++ code, and each renderer has several of them built-in (in some cases up to a few dozens). Most people will therefore never be faced with one of them.

However, shaders are the final instance in rendering, they define exactly how the looks of a certain material is achieved. Therefore, when the MatSys is given a certain material, it consults its library of built-in shaders and selects the very shader that is best suitable for rendering that material. The selection is automatic in most cases, but can also be manually overridden by the material.

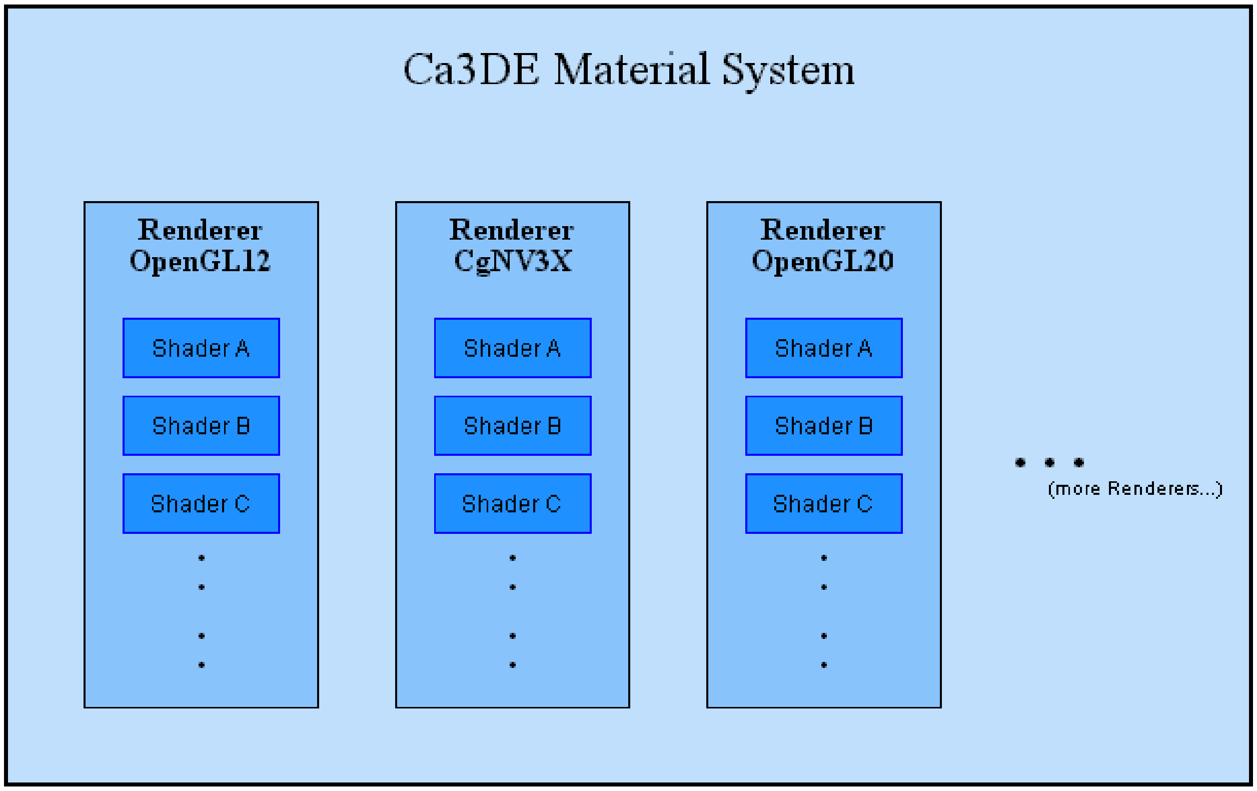
Even the case that there is a non-optimal assignment may sometimes occur, e.g. when on the underlying hardware no shader can be implemented that renders the desired material perfectly. In this case, a sub-optimal (but still the best) shader is selected. This mechanism provides a fall-back solution for arbitrarily old hardware, meeting another important design goal of the MatSys.

Shaders therefore provide the key to be able to always support new hardware: If new 3D hardware features become available, we can quickly write a new shader for it, put it into the appropriate renderer, and then materials can use it, either from auto-selection or by explicit statement. This ability is a great feature that makes it easy to scale the MatSys for future hardware, limited (e.g. embedded) hardware, old hardware, etc.

Future SDKs will come with the ability to plug-in custom shaders, and the Wiki documentation about shaders will updated until then.

31.5 Summary

Technically and internally, the MatSys is organized as follows:



The real shaders actually have different names than in this sketch, but shaders with the same name (e.g. `Shader B`) all (try to) render a given material in the same way – as much as that is possible on the underlying rendering technology.

The shaders of each renderer may ship as built-in shaders or can be provided as custom plug-in shaders by MOD authors.

CHAPTER 32

The Materialviewer

The Cafu SDK comes with a Materialviewer program, that can load and render a material in the same way the engine would.

The program is therefore a very useful utility to test your materials and see how they will look in the engine.

32.1 The command-line

The Materialviewer is a command-line driven program. If you run it without any parameters, it prints a short help message:

```
C:\Cafu-9.06> MaterialViewer.exe

Cafu Material Viewer (Jul 04 2009)

Warning: Games/DeathMatch/Textures/TechDemo.zip: No such file or directory
Warning: Games/DeathMatch/Textures/SkyDomes.zip: No such file or directory
Please use the -m option in order to specify the desired material!

OPTIONS:

-m=MyMaterial specifies the name of the desired material, which must be
   defined in one of the material script files (see below).

-bd=base/dir specifies the base directory from which I look both for material
   scripts and the materials associated textures.
   The default (if you don't use -bd) is Games/DeathMatch

-ms=MyMatScript.cmat specifies the material script that contains a definition
   of "MyMaterial" (see above). You may use -ms several times, making me look
   into each specified script for a definition of the material.
   If you do not use -ms at all, I'll look into ALL material scripts that I can
   find in Games/DeathMatch/Materials, so you probably don't need it as well.
```

(continues on next page)

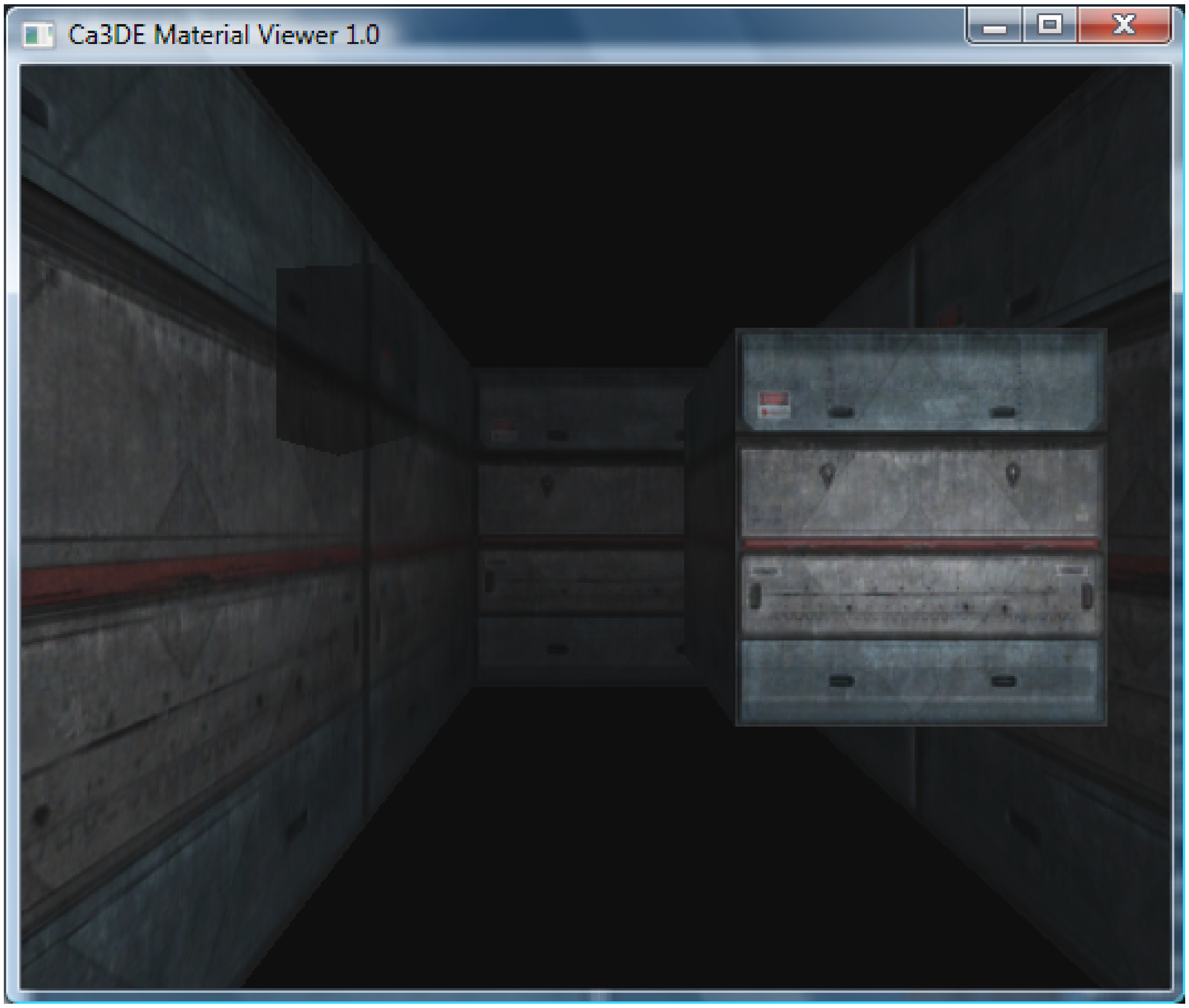
(continued from previous page)

```
-r=RendererXY overrides the automatic selection of the "best" renderer,
   and loads the renderer with base name RendererXY instead.
   Only provide the base name (e.g. RendererOpenGL12), no path and no suffix.
```

In most cases, just pass the name of the desired material as shown in *The Material Browser* in order to run the Materialviewer. Example:

```
C:\Cafu-9.06> MaterialViewer.exe -m=TechDemo/walls/wall-13f
```

32.2 The Materialviewer Window



The Materialviewer window contains a cube whose faces are applied with the selected material. The cube is surrounded by 4 walls, that have also the material applied. The bottom and top of this “room” are colored black.

You can turn on up to 4 dynamic light sources, that move around the scene and illuminate the cube from different angles. Light sources come in 4 colors (white, red, green, blue) and can all be activated separately using the keys `1-4`.

32.3 Keyboard and Mouse Controls

Use the following keys to control the Modelviewer:

Key	Action
`ESC`	Quits the program, same as `ALT+F4`.
`1`	Toggles if white light is visible.
`2`	Toggles if red light is visible.
`3`	Toggles if green light is visible.
`4`	Toggles if blue light is visible.

The following keys can be held down to achieve an effect:

Key	Action
`W` or `Arrow key up`	Moves the camera forward.
`S` or `Arrow key down`	Moves the camera backwards.
`A` or `Arrow key left`	Strafes the camera to the left.
`D` or `Arrow key right`	Strafes the camera to the right.

Further on you can change camera direction along the x axis by holding down the left mouse button and moving the mouse left or right.

CHAPTER 33

The Materials (cmat files) Manual

Cafu materials are defined in *material script files*, which in turn are simple ASCII text files with file name suffix `.cmat`. This page describes how such "cmat scripts" can be written, their syntax and semantics.

33.1 Overview

First of all, here is a simple example for a material definition script. It was taken from the `Cafu-9.06/Games/DeathMatch/Materials/Kai.cmat` file, which also contains several other, very similar material definitions:

```
Textures/Kai/3r_metpan01        // Material definitions start with the material name.
{
    diffusemap  Textures/Kai/3r_metpan01_diff.png     // This line says which texture␣
↪is used as diffuse-map.
    normalmap   Textures/Kai/3r_metpan01_norm.png
    specularmap Textures/Kai/3r_metpan01_spec.png
    lightmap    $lightmap
}
```

Before we dig into the details about keywords and structure of such material definitions, here are some general properties of cmat files:

- cmat files are simple ASCII text files, containing material definition scripts. Their file name ends with `.cmat`.
- All statements in such files are *case-sensitive*. That means that `text` and `Text` are *not* the same. This is also true for filenames, like the `Textures/Kai/3r_metpan01_diff.png` filename above, because some operating systems like Linux have a case-sensitive file-system.
- C++ style comments are allowed in material scripts: `// This is a comment.`
- The cmat files for MOD "MyMOD" are all stored in the `Cafu-9.06/Games/MyMOD/Materials/` directory and its subdirectories. This is necessary because the Cafu engine automatically scans this directory for material scripts whenever MOD "MyMOD" is run.
- Tokens in cmat files are separated by white-space and these individual characters: `( { [ ] } ) ,`

- Quoted tokens are recognized. That is, everything between two `"..."` is considered as one statement, even if white-space or one of the above characters is inside it. That means, if for example you *really* want to have a material name like `my(new and cool)material`, then you have to enclose it in quotation marks like this: `"my(new and cool)material"` in order to account for both the white-space and the brackets. Using quotation marks is not recommended, though! They're mostly useful if somebody created textures with weird file names like for example `"{_SomeFile.bmp"`. Write `MyNewAndCoolMaterial` or `My/New/And/Cool/Material` or something similar for your material names instead.

Material definitions always start with the **material name**. In the example above, that's `Textures/Kai/3r_metpan01`. You can name materials almost anything you like. If you want to use white-space, commas or brackets in their name, you'll have to put the name into quotation marks as mentioned above. However, it is important that the name is *unique*. If the same material name appears again in any other cmat file of the same MOD, the engine will use only the first occurance, so the chances are 50:50 that your material wins over the other. Most materials that come with Cafu have a filename that roughly resembles the name of its texture image files. That is often a helpful hint for conveniently working with the material, but by no means a requirement.

The **body** of the material definition is enclosed in a pair of `{ ... }`. In many cases, it will only contain a few texture map specification statements like in the example above.

33.2 Texture Map Specifications

The simplest form of a material definition is to only specify the texture map images that compose the material. The previous example, repeated here, was an example for such a simple definition:

```
Textures/Kai/3r_metpan01        // Material definitions start with the material name.
{
    diffusemap  Textures/Kai/3r_metpan01_diff.png     // This line says which texture␣
↪is used as diffuse-map.
    normalmap   Textures/Kai/3r_metpan01_norm.png
    specularmap Textures/Kai/3r_metpan01_spec.png
    lightmap    $lightmap
}
```

A texture map specification starts with a keyword (e.g. `diffusemap`) and is followed by a "map composition".

Map compositions are normally just the path plus file name of a texture image relative to the MOD directory, like for example `Textures/Kai/3r_metpan01_diff.png`. However, map compositions can also be more complex constructs as explained in the next section *Map Compositions*.

The initial keyword (`diffusemap` etc.) defines how the texture image is used in dynamic lighting computations during rendering. This is very similar to Doom3 materials, and both Cafu and Doom3 implement in this regard a form of the *Phong lighting model*.

Here comes a list of all available keywords for texture map specifications, along with a short description of their default meaning. (The default is dynamic Phong lighting, which however can be overridden, as explained at *Shader Specifications*.)

`diffusemap` The texture map image that defines the diffuse color (or diffuse reflectivity) of the material. The alpha channel of the diffuse-map specifies the translucency of the material.

`normalmap` The texture map image that specifies the color-encoded normal-map of the materials surface. The normal vectors must be specified in tangent space, range compressed (-1 to 1 maps to 0 to 1), and the y-axis points top-down. Note that heightmaps can be converted into normal-maps as described in section *Map Compositions* below.

`specularmap` The image for the materials specular highlights.

lumamap This texture map image defines the luminance for this material. Note that the light emittance that is defined here is local only (as is a general property of the Phong lighting model). It is not cast onto other surfaces.

lightmap This materials lightmap image. This image is normally computed and provided by the Cafu engine or the program (e.g. the material viewer or CaWE) with which you use the material. Although you can provide texture map file names as with the above keywords, that rarely ever makes sense. Normally, simply specify the special lightmap `$lightmap` in combination with the `lightmap` keyword in order to use whatever lightmap the Cafu engine provides for this material. This is demonstrated in the preceding example. **(!)** If you don't use this keyword at all, or specify something different than `$lightmap`, the compile tools will not cover surfaces that use this material with a lightmap, and these surfaces will neither receive nor reflect Radiosity light. Note that for many effect materials, this is desired and very useful behaviour, as for example with decals, skies, water surfaces etc.

shlmap The image that contains color-encoded coefficients for *Spherical Harmonic Lighting* for this material. As for lightmaps, you may specify arbitrary texture map file names with this keyword, but normally just use `$shlmap` in order to let the engine supply the proper SHL map.

cubeMap A six-sided cube-map image that is used e.g. for environmental mapping. Normally, cube-map specifications are per default *ignored*, because they are not regularly used with Phong lighting. Instead, they are specially activated as described in section *Shader Specifications*. As a single cube-map is actually composed of six individual images, a special convention for its file name is employed: A `'#'` character in the file name is automatically replaced with the six possible cube-map suffixes _px, _nx, _py, _ny, _pz and _nz. For example, consider this material definition from the `Cafu-9.06/Games/DeathMatch/Materials/SkyDomes.cmat` file:

```
Textures/SkyDomes/PK_BrightDay2
{
    AmbientShader A_SkyDome
    LightShader   none      // == noDynLight

    // The '#' in the next line is auto-replaced with the relevant suffixes (_px,␣
↪_ny, ...).
    cubeMap Textures/SkyDomes/PK_BrightDay2#.png, wrapS clampToEdge, wrapT␣
↪clampToEdge

    // ...
}
```

This example has keywords and elements that will be explained further below, but for now observe that the file name that is assigned to the cube-map is `Textures/SkyDomes/PK_BrightDay2#.png`. When the Material System loads the six individual images from disk, it will therefore load them from the files `Textures/SkyDomes/PK_BrightDay2_px.png`, `Textures/SkyDomes/PK_BrightDay2_nx.png`, `Textures/SkyDomes/PK_BrightDay2_py.png`, and so on.

cubeMap2 This is like the above `cubeMap` keyword. It allows you to specify a second cube-map for special-purpose shaders that require two cube-maps. This keyword too is normally ignored unless a shader is specified that makes use of it. See *Shader Specifications* for more details.

You can specify arbitrary combinations of these keywords in one material, as only the `diffusemap` keyword is mandatory. However, if you use the same keyword more than once, only the last occurrence is considered. The order of the keywords occurrences is not relevant.

33.2.1 Map Compositions

Texture map image specifications with the above keywords can not only be simple file names, but also be more powerful **Map Compositions**. A map composition is a description of how a *single* texture map image is composited from several source images on disk. Here is an example for a simple material whose normal-map is defined by a complex map composition:

```
Textures/Kai/barrel_rst
{
    diffusemap Textures/Kai/barrel_rst_diff.png
    normalmap  combineNMs(MyNm1.png, hm2nm(add(MyHm2.jpg, MyHm3.tga)))
    lightmap   $lightmap
}
```

(This example is overly complex for demonstration purposes, and not really meaningful. Real-life examples are normally much simpler.)

The expressions that are valid to define a map composition are defined as follows. Please note that the *arbitrary nesting* of expressions is expressly permitted, yielding great freedom for artists.

`filename` This is the most simple expression: a path plus a filename, as e.g. `Textures/Kai/barrel_rst_diff.png` in the above example for the `diffusemap`. The path is relative to the directory of the MOD for which this material script was written. Supported file extensions include png, tga, jpg and bmp.

`add(e1, e2)` This expression adds the colors of `e1` and `e2`, where `e1` and `e2` can be arbitrary sub-expressions. The resulting RGBA values are clamped to 1.0.

`mul(e1, e2)` This expression multiplies/modulates/filters the colors of `e1` and `e2`.

`combineNMs(e1, e2)` Treats the colors of `e1` and `e2` as color-compressed normal vectors, and combines or “adds” them in a mathematically correct fashion. (This it *not* the same as the `add(...)` operation.)

`hm2nm(e1, scale)` Assumes that `e1` is a gray-scale heightmap and converts it into a normal-map. The relative height of the heightmap is scaled by factor `scale` in order to weaken or pronounce the resulting effect. Values between 1.0 and 10.0 are normal use, but numbers greater than 10.0, less than 1.0, or even negative numbers are allowed, too.

`flipNMyAxis(e1)` Considers the colors of `e1` as color compressed normal-vectors, and flips their y-component. This is useful for normal-maps that have their y-component pointing into the wrong direction. Such normal-maps occurred in the early days of dynamic lighting or were created for other programs than Cafu. This function is for fixing such cases, and should rarely be needed.

`renormalize(e1)` Considers the colors of `e1` as color compressed normal-vectors, and renormalizes them (scales them to unit length). This is mostly useful for testing and debugging.

`blue2alpha(e1)` This function is for use with old diffuse-maps. It replaces the alpha channel of `e1` with value 0.0 (transparent) if the RGB color at this pixel is pure blue (0.0, 0.0, 1.0), and 1.0 (opaque) otherwise. Moreover, pure blue pixels are replaced with the average pixel color of the non-blue pixels in order to account for texture filtering.

`(automatic scaling)` Whenever you employ one of the above expressions to combine the results of two expressions `e1` and `e2` that have different lateral dimensions, `e2` is automatically scaled to match the size of `e1`.

You can apply map composition expressions to *all* above mentioned texture map specification keywords, i.e. they work with `diffusemap`, `normalmap`, `specularmap`, `cubemap`, etc.

Technically, a map composition is completed before the Cafu engine or the graphics board see them. In other words, the engine or the 3D hardware never see the individual images, only the composite result. *Everything that is done by these composition steps could also be pre-worked by the artist in an image processing software. There would be***no***difference for the engine, the hardware, or in the resource (memory) consumption.* Note that this feature has nothing to do with dynamic lighting or how a texture map image is combined with images of other texture map specification keywords!

Finally, you can specify several comma-separated options for the map composition:

`minFilter` This controls MipMap usage and the filter that is used for texture minification. While the default setting usually looks best and also yields the best performance on modern graphics hardware, sometimes it is desireable to turn filtering off and accept some aliasing, as for example for font textures. The `minFilter` keyword must be followed by one of the filter methods

- `nearest` or `point`
- `linear` or `bilinear`
- `nearest_mipmap_nearest`
- `nearest_mipmap_linear`
- `linear_mipmap_nearest`
- `linear_mipmap_linear` or `trilinear` (This is the default.)

`magFilter` This controls the filter that is used for texture magnification and must be followed by one of

`nearest` or **‘‘point‘‘** (There is almost never a reason to use this, except for very rare and special purposes, like some kinds of debugging.)

`linear` or **‘‘bilinear‘‘** (This is the default and gives best results.)

`wrapS` This controls horizontal texture coordinate wrapping and must be followed by one of

`repeat` for repeating the texture in horizontal direction. This is the default.

`clamp` for clamping the texture in horizontal direction, taking the border color into account. As the Cafu MatSys never uses or sets the border color, using `clamp` is rarely ever useful.

`clampToEdge` for clamping the texture in horizontal direction to its edge color. Often useful with cube-maps or terrain base images.

`wrapT` This controls vertical texture coordinate wrapping and must be followed by one of

`repeat` for repeating the texture in vertical direction. This is the default.

`clamp` for clamping the texture in vertical direction, taking the border color into account. As the Cafu MatSys never uses or sets the border color, using `clamp` is rarely ever useful.

`clampToEdge` for clamping the texture in vertical direction to its edge color. Often useful with cube-maps or terrain base images.

`noScaleDown` specifies that the texture image is never scaled down, not even if the user selects a medium or low texture detail setting for tuning the graphics performance. Useful for fonts, HUDs, lightmaps (implicitly), some model textures (see example below), and everything else that must not get mixed up or blurred by image filtering. Also used e.g. for the Cafu splash screen logo, which would get blurred otherwise (look into `Games/DeathMatch/Materials/Splash.cmat` if you want to toy around with it a little).

`noCompression` exempts this texture image from being stored in a compressed format in video memory, even if the user generally enabled texture compression for tuning the graphics performance. In the Cafu engine, texture compression is by default enabled for all texture images except normal-maps. Although Cafu automatically selects and employs the latest and highest quality compression method that the graphics driver offers (this even works when the Cafu executable is *older* than the driver!), sometimes the compression process comes with some loss of image detail or introduces small artifacts. `noCompression` can then be used to ensure no compression for a particular texture.

`useCompression` is more or less the opposite of `noCompression`: it turns compression back on. Normally there is no reason to ever use this keyword. It exits for symmetry to `noCompression` and because `noCompression` is the default for normal-maps: if you want to have compression enabled for a particular normal-map, specifying `useCompression` will turn it on. However, please note that compression artifacts in normal-maps tend to disturb the lighting computations so much that the generated output images drop to questionable quality. Also note that `useCompression` is "weak": If the user generally disables all compression, it will have no effect.

The meaning of the `minFilter`, `magFilter`, `wrapS` and `wrapT` options is analogous to their respective meanings in the OpenGL and DirectX APIs. The OpenGL Programming Guide (the "Red Book") about OpenGL version 1.2 and higher has a good explanation about these options. Although the text is specific to OpenGL, the same concepts apply to the above mentioned options. The "Red Book" for version 1.1 does not address the `clampToEdge` option, but its text is available online at http://www.rush3d.com/reference/opengl-redbook-1.1/chapter09.html.

The options `noScaleDown` and `minFilter bilinear` are often combined, because both scaling down textures for better graphics performance as well as using `trilinear` filtering for rendering have a tendency to mix the colors of neighboring pixels. In some cases such as font textures, even the `bilinear` filtering is too much mix-up, requiring us to combine `noScaleDown` with `minFilter nearest`.

Options Example 1

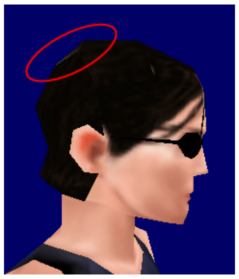
Here is an example from `Games/DeathMatch/Materials/Fonts.cmat` that demonstrates how the options are used:

```
Fonts/Arial
{
    diffusemap ../../Fonts/Arial.png, minFilter nearest, magFilter nearest,␣
↪noScaleDown
    // ...
}
```

Options Example 2

Another example for the `noScaleDown` and `minFilter` keywords.

This is a typical skin texture that a modeller has produced for application to one of his model meshes.


A straightforward material definition would look like this:

```
Models/Players/Trinity/trinityskin3
{
    diffusemap Models/Players/Trinity_Skin_diff.png

    red   ambientLightRed
    green ambientLightGreen
    blue  ambientLightBlue
}
```

The image to the left shows the result of the this material definition being applied to a model mesh. Notice the small glitch in the image, which is a result of mipmaps being applied to the above shown texture: Mip-mapping mixes black pixels of the hair with adjacent, bright pixels of the skin, yielding the intermediate colors that are marked in the result image to the left. Such glitches are even more disturbing and better visible with animated models, e.g. when the head of the model slightly turns.

Note that these kinds of artifacts are *no* bugs, they are a normal result from mipmap filtering.

With the `noScaleDown` and `minFilter bilinear` keywords applied in the material script, the glitch disappears as shown here:

```
Models/Players/Trinity/trinityskin3
{
    diffusemap Models/Players/Trinity_Skin_diff.png, minFilter bilinear, noScaleDown

    red   ambientLightRed
    green ambientLightGreen
    blue  ambientLightBlue
}
```

33.3 Shader Specifications

The examples of material definitions that we have seen so far were only composed of texture map specifications. In the *MatSys Introduction* we have learned that internally, "shaders" take materials to finally implement how they are rendered and how they look. In cases like our example materials above, the MatSys selects shaders from a built-in library automatically, picking the one that best fits the material.

But what if you want to change the way how your materials are rendered, either just in the details or radically from ground up? What if you don't want regular Phong lighting? Or you want an entirely different method to render diffuse-maps or a different way to combine them with normal-maps? What if the built-in shaders don't cover a way of rendering that you prefer? It also happens that the built-in shaders do not take cube-maps into account at all, so what if you want to have a material with cube-map environment reflections?

The answer to these questions is two-fold: You have to override the automatic shader selection by assigning a shader manually. But you also have to "know" that shader: You have to know its name, or you cannot assign it to your material. You have to know wether it needs a diffuse-map or a cube-map or both, or any other combination of texture map images, so that you can specify them in your material definition. Sometimes you have to know even more details about it.

The good news is that the MatSys renderers have more shaders built-in than just those that are considered during the automatical selection. They are documented below, inclusive sample material definitions, and you can use them to override the auto-selection. Alternatively, if you know or are a programmer, you can also make your own shaders. By making own shaders you gain the greatest flexibility that you can get, because only shaders eventually define how materials look, and shaders can do everything that the underlying hardware can do!

There is another issue that you should know about shaders: Each material gets not only one shader assigned, but *two*. This is because the Cafu Material System renders materials in two steps: First, the *ambient* part of a material is rendered, that is, everything that is rendered even if no light source is present. The *ambient shader* of a material is responsible for how this part is rendered. Then, the parts of the material looks that are contributed by each light source are rendered separately, as for example diffuse reflections, specular highlights and other effetcs. The per-light-source contributions are controlled by the *light shader* of a material.

You set the ambient and light shader of a material (and thereby override the automatic selection) with the following keywords in the material definitions body:

`AmbientShader MyAmbientShader` assigns the shader with name `MyAmbientShader` as the ambient shader to this material.

`LightShader MyLightShader` assigns the shader with name `MyLightShader` as the per-light-source shader to this material.

Here is an example for how these statements could be used in a material definition:

```
Textures/Kai/barrel_rst
{
    AmbientShader myCarMetallicBlue_ambient
    LightShader   myCarMetallicBlue_light
```

(continues on next page)

(continued from previous page)

```
    diffusemap    Textures/Kai/barrel_rst_diff.png
    normalmap     flipNMyAxis(Textures/Kai/barrel_rst_norm.png)
    cubeMap       Textures/SkyDomes/ReflectiveCubeMap#.jpg
}
```

33.3.1 Built-in special-purpose Shaders

Currently, all MatSys renderers come with several special-purpose shaders built-in (special-purpose means that they are never taken into account for automatic assignment). In most cases, they are actually pairs of shaders, one shader for the ambient and one shader for the per-light contribution of the same effect, but you will find that the exception to this rule is more often true than not. More built-in shaders are currently in preparation, for example for special effects like cube-map environment reflections, realistic water surfaces, etc. Note that any programmer can also write his own shaders, allowing him to implement *any* rendering effect that he wants! This makes the MatSys highly flexible, extensible and future-proof, and was one of its primary design goals.

The following special-purpose shaders are currently built into all renderers of the MatSys. Please note these shaders and their related examples are pretty advanced, and you might want to skip them until later. It may be convenient to first finish reading the rest of this documentation, which will help to fully understand the special-purpose shader examples.

The SkyDome Shader

In order to create surfaces that are invisible like fully transparent glass and instead only show the far sky dome beyond them, the ambient shader **‘‘A_SkyDome‘‘** exists. It basically only requires a cube-map for the sky to be specified. In fact, it ignores the specification of any other (i.e. diffuse-, normal-, etc.) maps. However, in most cases you'll want to specify additional keywords that further describe the properties of the sky surfaces. Here is an example:

```
Textures/SkyDomes/PK_BrightDay2
{
    // Activate the A_SkyDome shader as the ambient shader
    AmbientShader A_SkyDome

    // Have no dynamic light affect surfaces with this material.
    LightShader   none
    noDynLight

    // The '#' in the next line is auto-replaced with the relevant suffixes (_px, _ny,
↪ ...).
    cubeMap Textures/SkyDomes/PK_BrightDay2#.png, wrapS clampToEdge, wrapT clampToEdge


    // Don't write into the z-Buffer, so that entities (like missiles) outside of the␣
↪map can still be drawn.
    ambientMask d

    // This material does not cast dynamic shadows.
    noShadows

    // Don't create or keep lightmaps for this material, don't participate in␣
↪Radiosity computations.
    meta_noLightMap

```

(continues on next page)

(continued from previous page)

```
    // This keyword states that this material casts sunlight.
    meta_sunlight
        // The irradiance of the sunlight in Watt/m^2 that comes (or shines) through␣
↪this material.
        // Values like (100  90  80) might work, too.
        (220  180  100)
        // The direction of the incoming sunlight rays. The z-component should be␣
↪negative.
        // (These values match the actual position of the sun in the cube-maps.)
        (-17 -699 -715)
}
```

The **‘‘AmbientShader A_SkyDome‘‘** line activates the ambient sky dome shader. Note that the far away sky dome is *not* affected by light of any dynamic light source, and therefore we assign the **‘‘none‘‘** shader as the per-lightsource shader in order to make sure that no dynamic light is applied to sky surfaces. The **‘‘noDynLight‘‘** keyword does essentially the same as **‘‘LightShader none‘‘** and will soon be obsoleted. Until then, please use it together with each occurrence of **‘‘LightShader none‘‘**.

The **‘‘cubeMap . . . ‘‘** statement specifies the cube-map that is to be used for this sky. If you also specified other texture maps like diffuse-maps or specular-maps, they would simply be ignored, as the **‘‘A_SkyDome‘‘** shader has no use for them.

The remaining keywords further specify important properties of this material. Please refer to section *Keyword Reference* for a detailed description. Short explanations of their meanings are given in the comments in the above example.

The Terrain Shader

The terrain shader exists in order to render the Cafu outdoor terrains, which work a bit different than regular, Phong-lit surfaces. You activate the ambient terrain shader by writing

```
AmbientShader A_Terrain
```

in the materials body. Normally, you would now to expect to also assign a terrain-specific shader for the per-lightsource contribution to the terrain, as in

```
LightShader L_Terrain
```

However, I have not yet written the **‘‘L_Terrain‘‘** shader, and so we have to turn off dynamic lighting for terrains:

```
LightShader none
noDynLight
```

(?) Why do terrains not account for light by dynamic light sources? Well, there are several reasons:

1. Terrain is typically employed by mappers in outdoor areas that are in bright sunlight. The effect of dynamic light sources would barely be visible, if at all (but cost *a lot* of performance instead).
2. Dynamic light sources are normally very “small” when being compared to the extends of terrains, and mappers tend to place them inside buildings rather than in the open area. This further limits the per-lightsource contribution on terrain surfaces.
3. I simply have not yet had the time to write the **‘‘L_Terrain‘‘** shader.

(!) But if for example somebody wanted to model an indoor cave with the Cafu terrain technique, having a shader that accounts for the light of dynamic light sources even on terrains would make a lot of sense. Good news is that, as indicated above, it is *easy* to write such a shader – I’ll probably do so as soon as I need one.

Next, the **‘‘A_Terrain‘‘** (and the future **‘‘L_Terrain‘‘**) shader requires a diffuse-map to be specified. This diffuse-map will be scaled to match the physical size of the terrain, that is, it will cover the terrain completely. In order to get the edges of the terrain properly textured, it makes sense to also specify **‘‘clampToEdge‘‘** border wrapping for the diffuse texture:

```
diffusemap Textures/Terrains/BPRockB_tx1.png, wrapS clampToEdge, wrapT clampToEdge
lightmap $lightmap
```

As the Cafu map compile tools will also generate a lightmap for the terrain in order to e.g. take sunlight or other radiosity light sources into account, we specify the **‘‘lightmap‘‘** keyword with the engine-supplied lightmap. Note that while the lighting of dynamic light-sources is *not* taken into account, the light of radiosity light sources *is*!

Next, the **‘‘A_Terrain‘‘** shader employs the texture-map that is specified with the **‘‘lumamap‘‘** keyword as a *detail-map* for the terrain. This is a good example for how the specification of a custom shader (here **‘‘A_Terrain‘‘**) can entirely alter the meaning of a material keyword. We will see below how the coarseness (the repetition-count) of the detail-map is set.

```
lumamap    Textures/Terrains/CommonDetail2.png      // "A_Terrain" takes the Luma-map␣
↪as Detail-map (optional).
```

The detail-map is optional, and may be omitted. However, terrains look a lot better with them, and so their use is recommended.

(?) You may be wondering wether multiple detail-maps can be handled, as for example in FarCry, where they have a detail-map for beach sand, one for rocks, one for grass and one for pavement.
(!) The answer is: The **‘‘A_Terrain‘‘** shader can indeed *not* handle such detail maps. But as before, it would actually be easy to write such a shader as soon as one is needed!

Finally, you need this block of statements in the body of your material definition:

```
shaderParamExpr fParam4    // The first eight shader parameters are taken from␣
↪fParam4 to fParam11
shaderParamExpr fParam5    // and specify the coefficients of two planes for␣
↪automatic tex-coord generation.
shaderParamExpr fParam6
shaderParamExpr fParam7
shaderParamExpr fParam8
shaderParamExpr fParam9
shaderParamExpr fParam10
shaderParamExpr fParam11
shaderParamExpr 21.3       // Scale / repetitions of the Detail-map.
```

The only thing that you may change here is the number of repetitions of the detail-map, 21.3 in the above example. The lines above that are required in order to get the terrain properly rendered and cannot reasonbly be altered.

For more information on what all the **‘‘shaderParamExpr fParam*‘‘** lines do, please refer to section **(FIXME!)** (TODO).

Here is a complete example for a terrain shader:

```
Terrains/BPRockB_tx1
{
    AmbientShader A_Terrain
    LightShader   none
    noDynLight
```

(continues on next page)

(continued from previous page)

```
    diffusemap Textures/Terrains/BPRockB_tx1.png, wrapS clampToEdge, wrapT clampToEdge
    lightmap   $lightmap
    lumamap    Textures/Terrains/CommonDetail2.png

    shaderParamExpr fParam4     // The first eight shader parameters are taken from␣
↪fParam4 to fParam11
    shaderParamExpr fParam5     // and specify the coefficients of two planes for␣
↪automatic tex-coord generation.
    shaderParamExpr fParam6
    shaderParamExpr fParam7
    shaderParamExpr fParam8
    shaderParamExpr fParam9
    shaderParamExpr fParam10
    shaderParamExpr fParam11
    shaderParamExpr 21.3        // Scale / Repetitions of the Detail-map.

    twoSided                    // "twoSided" is required for the SOAR terrain␣
↪algorithm.
}
```

The WaterCubeReflect Shader

In order to turn a polygon into a translucent water surface with moving waves and reflected environment with Fresnel effect, employ the **‘‘A_WaterCubeReflect‘‘** shader: **(FIXME!)** (This section is not complete!)

The “none” Shaders

You can use the special **‘‘none‘‘** shader both as an ambient or a per-lightsource shader in order to have no shader for ambient or per-lightsource contributions at all.

As you have seen in the examples above, this makes sense in several situations. Especially materials that are not affected by local, dynamic light sources often have the **‘‘LightShader none‘‘** statement, as for example for sky dome. Note that currently, you still have to combine any **‘‘LightShader none‘‘** statement with the **‘‘noDynLight‘‘** statement in order to take proper effect. The **‘‘noDynLight‘‘** will however become obsolete in future releases of the Cafu Material System.

Using **‘‘AmbientShader none‘‘** is much less frequently useful, and almost only ever employed for “invisible” materials. Note that **‘‘AmbientShader none‘‘** also implies **‘‘LightShader none‘‘**. Also, in order to take proper effect, the **‘‘noDraw‘‘** keyword is required with each occurrence of **‘‘AmbientShader none‘‘**, but that requirement will be removed and **‘‘noDraw‘‘** be obsoleted in future versions of the MatSys.

33.4 Expressions and Tables

Some material keywords take not only plain numbers as their parameters, but entire mathematical expressions that are evaluated whenever the material is employed for rendering. Expressions in turn may refer to look-up tables, which are a very simple but flexible means to approximate arbitrary mathemathical functions.

This section explains the details of expressions and tables.

33.4.1 Expressions

Some of the above keywords cannot only take numbers as their arguments, but entire mathematical **expressions**. An expression is a combination of numbers, mathematical operations, symbols, and table look-ups. Here is an example for such an expression:

```
(1 + sinTable[ time*0.25 + 3 ]) / 2
```

Unfortunately, there is also bad news: The current parser of the Cafu MatSys is not yet powerful enough to parse expressions that use operator infix notation like the one above. Instead of `+`, `-`, `*`, `...` we therefore have to use explicit prefix notation, which is much easier to parse. The prefix notation is explained below. Here is the above example in prefix notation that we have to use until I can improve the parser to support infix notation:

```
div(add(1, sinTable[ add(mul(time, 0.25), 3) ]), 2)
```

Expressions are defined recursively as follows:

`$num` A number.

`$var` A variable. Please see below for a list of all valid variables.

`$table[ $expr ]` A table look-up. The value of the table at position `$expr` is returned. The details of the look-up depend on the tables definition. The default tables `sinTable` and `cosTable` are always predefined. Custom tables can be defined as described in subsection *Tables*. Note that table look-ups are normalized. That is, all table elements are accessed by indices between 0.0 and 1.0, that is, the fractional part of `$expr`.

`add($expr1, $expr2)` Adds the results of `$expr1` and `$expr2`.

`sub($expr1, $expr2)` Subtracts the results of `$expr1` and `$expr2`.

`mul($expr1, $expr2)` Multiplies the results of `$expr1` and `$expr2`.

`div($expr1, $expr2)` Divides the results of `$expr1` and `$expr2` if `$expr2` is not 0. Otherwise, this evaluates to 0.

The following variables are defined:

`time` The current system time in seconds, starting from zero.

`ambientLightRed`, `ambientLightGreen`, `ambientLightBlue` The Cafu engine provides the color of the ambient light *that is contributed by radiosity light sources* in these variables. Therefore, almost all material definitions that are employed with *entities* (e.g. player models, weapon and item models, etc.) normally use these variables so that the entity is colored according to the ambient radiosity light. Please see the `.cmat` files in `Games/DeathMatch/Materials/Models/` for many examples.

Tables and table look-ups are described in subsection *Tables*.

33.4.2 Tables

A special expression is a table look-up of the form `$myTable[$indexExpr]`, where `$myTable` is the name of a table that must have been defined before use, and `$indexExpr` is another expression that determines where in `$myTable` the look-up occurs. The scope of `$myTable` begins at its definition and ends at the end of the material script.

Here is a simple but complete example with a table of four values:

```
// This line defines the table "myTestTable":
table myTestTable { { 0.2, 1.4, 0.6, 1 } }

```

(continues on next page)

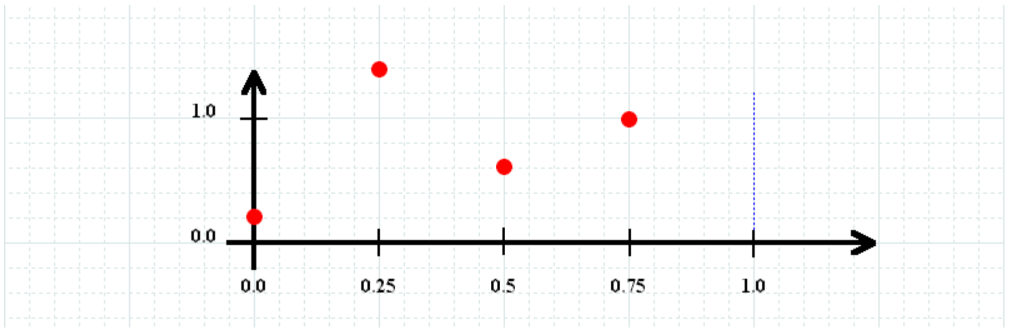
(continued from previous page)

```
TestMaterialForTableLookup
{
    diffusemap  someDiffuseMap.png
    // ...

    rgb div(myTestTable[mul(time, 0.5)], 1.2)   // equiv. to:   rgb␣
↪myTestTable[time*0.5]/1.2
}
```

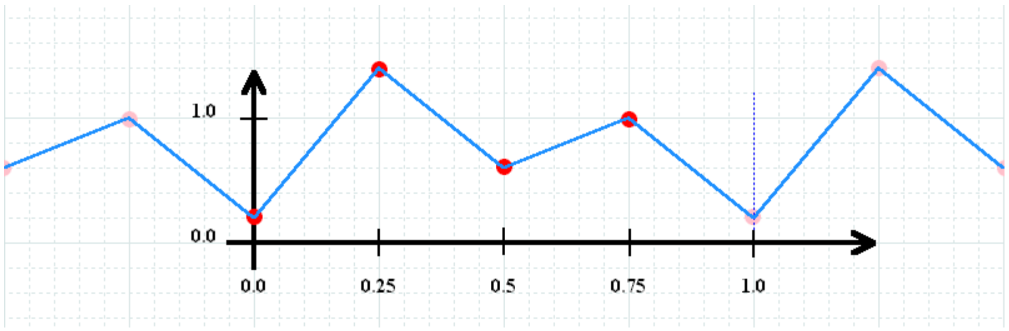
As you can see, table definitions start with the keyword `table`, followed by the name of the table. Then come two { brackets, the table data values separated by commas (arbitrarily many), and then the closing two } brackets.

Note that table data values are always mapped into the range 0 to 1. Therefore, the graphical representation of `myTestTable` looks like this:



`myTestTable[0]` yields 0.2, `myTestTable[0.25]` yields 1.4, and so on…

You may wonder what you get for `myTestTable[x]` if x is smaller than 0 or greater than 0.75, or what happens if x is "between" two table values. Per default, table values are infinitely repeated outside of the range 0 to 1, and linearly interpolated between two adjacent values. Therefore, `myTestTable` from the above example in fact represents the following function, graphically shown in light blue color:



Observe how the values repeat with each integral number. That is, the table in range 0 to 1 is repeated between 1 to 2, between 2 to 3, 3 to 4, and so on. It is also repeated into the negative range, that is from -1 to 0, from -2 to -1, -3 to -2, and so on.

Also observe how intermediate values are linearly interpolated. For example, the value at `myTestTable[0.375]` (in the mid between 0.25 and 0.5) yields 1.0 as the result.

For example, if you access `myTestTable` via the MatSys variable `time`, as in `myTestTable[time]`, then it always takes exactly one second to traverse the entire table. If you want to change the speed with which the table is traversed, then you'll have to multiply the index variable `time` with an appropriate scale factor. For example, `myTestTable[div(time, 3)]` will take three seconds to walk the entire table once.

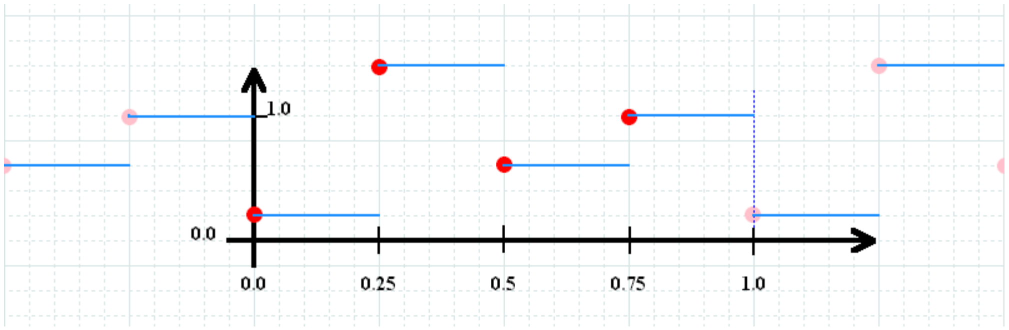
snap and clamp

For special-purpose tables, you can insert the keywords `snap` and/or `clamp` between the two opening `{` brackets of the table definition.

"snap" turns the linear interpolation off, and instead repeats the previous value until the next table value. Thus, if we changed the definition of our `myTestTable` above to

```
table myTestTable { snap { 0.2, 1.4, 0.6, 1 } }
```

then our graphical representation of the table becomes:

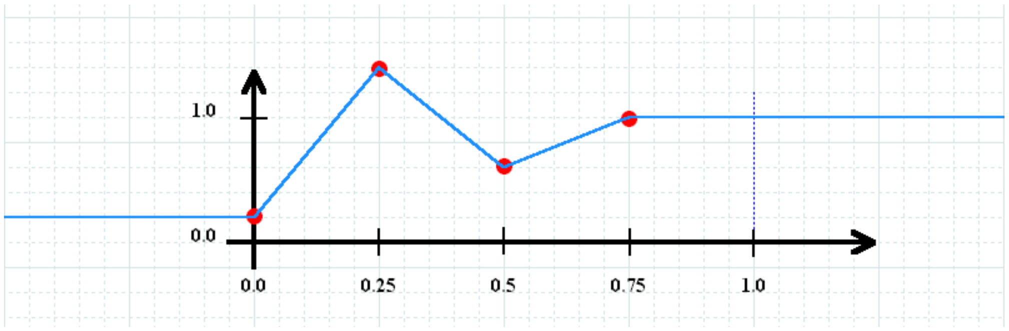


`snap` turns the interpolation off, and just repeats one value until the next.

Snapping is useful whenever you want to have a table to encode functions that have "hard" rather than "smooth" transitions. For example, in order to have LEDs flicker the SOS morse code, you'd use this table:

```
table sosTable { snap { 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
↪ 0, 0 } }
```

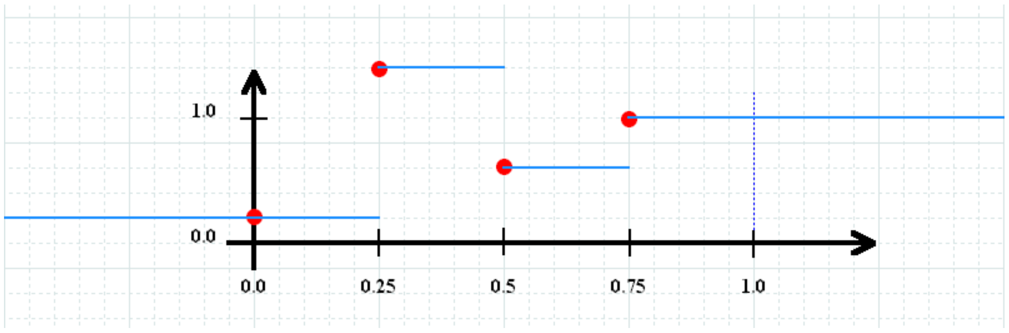
"clamp" turns off the repetition of the table values outside of the range 0 to 1. That is, adding `clamp` to our definition of `myTestTable` yields:



```
table myTestTable { clamp { 0.2, 1.4, 0.6, 1 } }
```

With `clamp`, the first table value is returned for `myTestTable[x]` where x is less than 0, and the last table value is returned whenever x is greater than 1 (in fact, greater than 1-1/TableSize).

Finally, you can also combine `snap` and `clamp`:



```
table myTestTable { snap clamp { 0.2, 1.4, 0.6, 1 } }
```

Predefined Tables

Normally, tables must be defined before their first use, but there are also tables that are inherently defined by the MatSys and can always be used without prior definition:

`sinTable` yields the sinus of its argument. Note that the entire 360° (2pi) arc is compressed into the range 0 to 1, not 0 to 2pi.

`cosTable` yields the cosinus of its argument. Note that the entire 360° (2pi) arc is compressed into the range 0 to 1, not 0 to 2pi.

`sinTable01` like `sinTable`, but the values are not returned in range -1 to 1, but "compressed" to 0 to 1. That is, `sinTable01[x]` is equivalent to `div(add(sinTable[x], 1), 2)`.

`cosTable01` like `cosTable`, but the values are not returned in range -1 to 1, but "compressed" to 0 to 1. That is, `cosTable01[x]` is equivalent to `div(add(cosTable[x], 1), 2)`.

33.5 Keyword Reference

This section lists all keywords that are allowed in the body of a materials definition.

Some of the keywords influence the automatic selection of the shader for the material, others just give more control over the shader that is eventually used for the material, for making better use of the features of that shader, increasing its usefulness.

Note that the built-in shaders of all MatSys renderers were written to comply to the documentation of the keywords below, or rather, the documentation was written to reflect the implementation of the built-in shaders. If you employ a custom shader (e.g. a self-written one) instead, there is no guarantee that this shader interprets and implements the keywords in an identical manner, because the shader is flexible and free to do anything that its author made it to. This is where you need a documentation that is specific to that shader, see *Built-in_special-purpose_Shaders* for the built-in ones.

In general, most keywords should occur only once per material definition. If such a keyword is stated several times, only its last occurrence is taken into account, overriding its previous statements. The order in which such keywords are specified in the body of the material definition is irrelevant, you may state the keywords in any order.

However, some keywords are allowed to occur several times, for example for enumerating textures or numbers. For such keywords, the order in which they appear *is* important. Keywords with this feature are explicitly mentioned below.

(FIXME!) We should get rid of this exception by changing the syntax to e.g. **‘‘shaderParamExpr ($expr1, $expr2, . . . , $exprN)‘‘**.

33.5.1 Shader Specifications

`AmbientShader $name`, `LightShader $name`

These keywords are used to override the automatic shader selection. They are explained in detail in section *Shader Specifications*.

The precise meaning of all other keywords is “vague”, that is, they depends on the implementation by the selected shader **(!)**

33.5.2 Texture Map Specifications

`diffusemap $mc`, `normalmap $mc`, `specularmap $mc`, `lumamap $mc`, `lightmap $mc`, `shlmap $mc`, `cubeMap $mc`, `cubeMap2 $mc`, `shaderParamMapC $mc`

With these keywords you specify texture map compositions **‘‘$mc‘‘** for use with this material. The keywords and texture map compositions are explained in detail in section *Texture Map Specifications*.

The **‘‘shaderParamMapC‘‘** keyword is special, as it can occur arbitrarily often! You can use it to pass an arbitrary number of texture-maps to the shader. Materials that employ a custom shader that for example combines several diffuse-maps and several normal-maps might use this keyword in order to list all the textures that the shader employs.

33.5.3 Custom Data

`shaderParamExpr $expr`, `shaderParamMapC $mc`

These two expressions are used to pass additional information to custom shaders. They can occur arbitrarily often in a material definition, so their order is important.

(FIXME!): This should be **‘‘shaderParamExpr ($expr1, . . .)‘‘** etc. in order to get rid of the order requirement.

The meaning of the expressions and map compositions that you pass to the shader with these keywords entirely depends on how the shader interprets them! The *Terrain* shader is a good example that employs these keywords.

(FIXME!): Provide more details here.

(FIXME!): The terrain shader should employ **‘‘shaderParamMapC‘‘** rather than **‘‘lumamap‘‘** for its detail-map. . .

33.5.4 Rendering Options (Universal)

The following material parameters are *global rendering parameters*. They affect the entire material, but only how it is rendered, that is, its “looks”. These parameters are never looked at or taken into account by the map compile tools.

`noDraw` If specified, this material does not render at all, it becomes invisible. This is probably mostly useful for debugging. (For mappers: `noDraw` does *not* cause CaBSP to remove any faces from the world. If you want faces to be actually removed, you should rather use the `meta_XXX_TODO` keyword in order to make CaBSP remove them right from the start.)

noShadows This material never casts shadows. Note that shadow-casting can also be controlled on a per-model and per-lightsource basis.

twoSided Normally, the back or “inside” faces of materials are not rendered. This renders both sides of the material. Useful for “indefinitely” thin objects like metal grates and fences, spider webs, ... Note that two-sided faces are currently not properly lit by dynamic light when seen from the back side.

depthOffset $num This sets the polygonal depth offset. Z-buffer values of meshes that are rendered with this material are slightly offset as defined by $num, which should normally be a negative number. Materials that employ `depthOffset -1.0` are for example used internally by CaWE, in order to render the highlighting of selected objects.

polygonMode $pm This sets the polygon mode of a mesh. CaWE employs this keyword for rendering the objects in the 3D wire-frame mode. `$pm` must be one of

- `filled`
- `wireframe`
- `points`

33.5.5 Rendering Options (Ambient only)

The following parameters do only affect the ambient contribution of a material (as implemented by the Material Systems ambient default shaders). The map compile tools never look at these parameters. Some of these parameters take expressions as their arguments, which are denoted by `$expr`. Expressions can be simple numbers or more complex constructs, they are discussed in greater detail in subsection Expressions.

alphaTest $expr Activates the alpha test with the result of `$expr` as the reference value. The alpha test passes if and only if the alpha value of the ambient contribution (which normally comes from the diffuse-map) is greater than the reference value. Note that the reference value can be specified as an expression, that is, it can be *varying over time*. This can produce interesting effects (i.e. materials that appear to grow or shrink) if the diffuse-map comes with an appropriate alpha channel. Negative values turn the alpha test off. The test is off by default.

blendFunc $src_factor $dst_factor This parameter determines the blend function for the ambient contribution of the material. `$src_factor` and `$dst_factor` must be one of `zero`, `one`, `dst_color`, `src_color`, `one_minus_dst_color`, `one_minus_src_color`, `dst_alpha`, `src_alpha`, `one_minus_dst_alpha`, or `one_minus_src_alpha`. Note that not all combinations make sense. Using `blendFunc` will be documented in greater detail in future releases of this text. Per default, blending is turned off.

red $expr, green $expr, blue $expr, alpha $expr, rgb $expr, rgba $expr These parameters all define expressions for (channels of) the color with which the ambient contribution is modulated. I have defined the default ambient shaders such that for materials that have luma-maps, only the luma-map is modulated. Materials that have no luma-map get the entire ambient contribution modulated. This allows to create effects such as panels that have flickering LEDs, glowing lights etc. Note that you can specify different expressions for different color channels. That is, if you have a luma-map for a computer panel that has both red and green LEDs, you can for example have the red LEDs morse SOS, while the green LEDs change gradually by a sinus function. The default expression for all color channels is 1.0 (identity).

ambientMask $turnoff This specifies into which framebuffer channel should *not* be written when rendering the ambient pass. `$turnoff` must be a combination of the characters `r`, `g`, `b`, `a` and `d`, where `r` refers to the red color channel, `g`, `b` and `a` refer to the green, blue and alpha color channels respectively, and `d` refers to the depth buffer value. For example, `ambientMask d` is used with many particle (sprites) material definitions, in order to avoid that particles change the contents of the depth buffer, in order to meet the fact that particles are often rendered in an unsorted, additive manner. `ambientMask d` is also used for sky-dome material defs, so that we are able to render e.g. far away missiles or other objects in the sky that would otherwise be rejected by the depth test against the (nearer) sky dome polygons. Note that employing `ambientMask d`, that is, *not* writing into the depth buffer during the ambient pass, also makes dynamic lighting of such affected meshes impossible. Therefore, `ambientMask d` should always be combined with `LightShader none` and `noDynLight`. **(FIXME!)**: The MatSys should issue a warning if that is not the case. Another example for employing this keyword is `ambientMask gb`, which makes sure that nothing gets rendered into the green and blue color channels, leaving only red (and alpha and depth). This yields an effect similar to Doom3, when you see only red while the player has a foreboding vision.

`useMeshColors` Additionally to the above color definitions, this will modulate the materials color with the colors that are specified with the vertices of the mesh that is to be rendered. This option does currently only work in a very limited set of shaders, and is mostly useful internally in CaWE, the Cafu World Editor, e.g. for rendering wire-frame stuff. You will rarely ever need this in a real-world material script.

33.5.6 Rendering Options (Light only)

`noDynLight` If specified, this material does not receive any light from dynamic light sources, only the ambient contribution is rendered. (Technically, the material does not even get a per-light-source shader assigned.) Useful e.g. for sky domes, additive effects like particles, translucent surfaces like water and glass etc.

`lightMask $turnoff` This is very similar to `ambientMask`, but it only affects the per-lightsource passes. This keyword is currently not used in any Cafu material.

33.5.7 Other Keywords

There are also keywords that define other aspects of the material that are not directly related to their rendering.

`clip $c1, $c2, ...` Defines for whom or for which purposes the material is considered solid in clipping (collision detection) computations. This is mostly interesting in game related settings (e.g. with player or monster clip brushes), but also programs like CaBSP and CaLight perform clip and ray trace tests that in turn are affected by this keyword. `$c1, $c2, ...` must be a comma-separated list of at least one of the flags below. The default setting (i.e. if you do not use the `clip` keyword at all), is `clip all`. When you use `clip`, the value is first cleared (reset to `nothing`) and then additively rebuilt from the parameter list. Here is an explanation of the individual flags:

- `nothing` means that a surface with this material clips against nothing, it is non-solid.
- `players` means that the surface is solid to players.
- `monsters` means that the surface is solid to monsters (AI entities).
- `moveables` means that the surface is solid to moveable entities.
- `ik` means that the surface is solid to IK.
- `projectiles` means that the surface is solid to the projectiles of weapons (e.g. water surfaces, but not sky).
- `sight` means that the surface blocks line-of-sight tests (used e.g. for AI).
- `bspPortals` means that the surface clips portals, that is, it is solid to the flood-fill operation of CaBSP. **(!)** Note that each Cafu map must *entirely be sealed* by materials with the `bspPortals` property, or else CaBSP will not be able to flood-fill it properly and report that the map has a leak. Developers, note that this means that maps can be made where objects can fall out of the map (e.g. through the floor). Considering the clip-world only, this can be desirable e.g. for rockets that are fired into the sky, even though Cafu has to treat them specially for drawing (render objects in non-PVS “outer” leaves). CaBSP issues a warning if it ever finds a material that has `bspPortals`, but not both `players` and `monsters` set, in order to make you aware that in such places, players or monsters could fall out of the map.

- `radiance` means that the surface blocks the transfer of light energy in the Radiosity computations of CaLight (most walls do, but for example glass and decals usually don't).
- `all` means that the surface is solid to all of the above (but not the `trigger` flag below).
- `trigger` means that the surface is part of a trigger volume. Entities that have trigger volumes can detect when something enters their volume and then take special actions. For example, a player that walks into a trigger volume can cause a script function to be run that in turn opens a door and spawns some monsters. The `trigger` flag is normally not combined with any of the other flags, but for special cases it is still possible to write e.g. `clip monsters, trigger` to define a material that is solid only to monsters and a trigger to everything else (e.g. players).

`surfaceType $st` Defines the type of the surface. `$st` must be one of `none`, `stone`, `metal`, `sand`, `wood`, `liquid`, `glass`, or `plastic`. The meaning of the surface types is solely defined by the game code, it will usually use them to play footstep sounds, ricochet effects, etc.

33.5.8 Meta Keywords

Finally, here are the meta-parameters that are taken into account by the Cafu map compile tools. These parameters are not directly related to the rendering of the material.

`meta_radiantExitance $r $g $b $scale` Radiant exitance RGB values plus intensity (scale). Used whenever a material should also be a source of radiosity light, as computed by CaLight. The radiant exitance is defined by `$r`, `$g` and `$b`, scaled by `$scale`.

`meta_radiantExitance_byImage` Radiant Exitance RGB values from image file. Used by CaLight, plus Radiant Exitance intensity (scale) for the RGB values from image file. Used by CaLight, but not yet implemented.

`meta_sunlight ($r $g $b) ($x $y $z)` This keyword states that the materials casts (radiosity) sunlight. The first three numbers define the irradiance of the sunlight in Watt/m^2 for the red, green and blue wavelengths. As the light actually comes from a very far away lightsource (the sun, moon, etc.), it is not cast by the materials itself, but rather *through* them, like sunlight shining through a window. The second triple of numbers specify the directional vector of the incoming light rays. The `$z` number should always be negative, so that the light appears to come from above. In the file `Games/DeathMatch/Materials/SkyDomes.cmat` you can see exemplary use of this keyword.

`meta_alphaModulatesRadiosityLight` Makes CaLight handle the `diffusemap` alpha channel and the $alpha and $rgba keywords properly. For fences, grates, glass, water, etc. Not yet implemented.

CHAPTER 34

Getting Started with the Cafu Source Code

This text explains how you get and compile the Cafu source code.

Most steps that are explained below are only necessary once, when you get and compile the Cafu source code for the first time.

34.1 Getting the Source Code

This section explains how you obtain the Cafu source code by checking it out from its Git repository.

Git is a version control system that allows the Cafu developers to work on their copy of the source code independently of each other, and to distribute the latest version of the source code to other developers at any time. Moreover, many developers prefer to manage their project independently in their own version control system, but also wish to integrate the latest changes (e.g. bug-fixes and new features) from the official Cafu repository into theirs from time to time: Git makes the synchronization of the projects code with the original Cafu source code as convenient and automatic as possible. If you don't know Git yet, have a look at the Git website and the Git documentation. The "Pro Git" book is very well written and available for free in several languages.

For Windows and many other systems, you can get Git from the Git downloads page. Popular Git clients with a graphical user interface are available at the Git downloads page as well.

With most Linux distributions, installing Git via the systems package manager is usually preferred. For example, under Ubuntu:

```
> sudo apt-get install git gitk
```

Then check out the source code at the command-line under Windows or Linux with this command:

```
> git clone --recursive https://bitbucket.org/cafu/cafu.git Cafu
```

Note the `--recursive` option in the command above: it makes sure that submodules are automatically checked out as well. As we keep the texture images for the DeathMatch example game in a submodule, it is convenient to have them checked out along with the main repository. Be prepared though that this adds approximately 180 MiB to the download volume.

Our Git project page is https://bitbucket.org/cafu/cafu, where you can browse the repository online, create forks, post pull requests, and find additional help about Git.

34.2 Python and SCons

Cafu uses SCons as its software build system on all supported platforms. SCons is a modern replacement for `make` and `Makefiles`. SCons requires the Python scripting language to be installed, so that you have to get and install both Python and SCons on your computer.

34.2.1 Windows

Under Windows,

- first get and install Python **2.7** (*not* one of the newer, but backwards-incompatible 3.x editions!) from http://www.python.org/download/,
- then get and install SCons **2.3** (or any later version) from http://www.scons.org/download.php.

If in doubt, pick the Windows installers for both Python 2.7 and SCons 2.3: they are easy to use, lightweight, and automatically setup the proper environment variables.

Important notes:

- Pick the **32-bit** edition of Python even on 64-bit systems! (Unfortunately, SCons does not yet work with the 64-bit builds of Python.) That is, python-2.7.12.msi is the right file for all Windows systems.
- On Windows 7, 8 and 10, run the SCons installer via right-click, then select **"Run as administrator"**.

That is normally all, but you may wish to check if the Python installer added Python's `Scripts` directory to the `PATH` environment variable of your system. For example, if Python was installed in `C:\Python27`, then `PATH` should contain both `C:\Python27` and `C:\Python27\Scripts`. Otherwise, you have to add the proper directories manually: See http://support.microsoft.com/kb/310519 and http://vlaurie.com/computers2/Articles/environment.htm for additional information.

In order to verify that everything is working correctly, open a new command prompt and enter `scons -v`:

```
> scons -v
SCons by Steven Knight et al.:
        script: v2.3.0, 2013/03/03 09:48:35, by garyo on reepicheep
        engine: v2.3.0, 2013/03/03 09:48:35, by garyo on reepicheep
        engine path: ['C:\\Python27\\Scripts\\..\\Lib\\site-packages\\scons-2.3.
↪0\\SCons']
Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012,␣
↪2013 The SCons Foundation
```

The report of the SCons version indicates that both Python and SCons are ready for use. Please make sure that you use SCons version **2.3 or newer**, older versions don't work!

34.2.2 Linux

Under Linux, just use the systems package manager in order to install SCons. The package manager will automatically install SCons and all software that SCons depends on, such as Python. For example, under Ubuntu:

```
> sudo apt-get install scons
```

34.3 Linux Packages

Additional software packages must be installed on a Linux system before the Cafu source code can be compiled.

On a fresh, newly installed Debian or Ubuntu system, you'll need the following packages. They can be installed at the command prompt with the `apt-get install` command, or via the easy to use "Synaptic Package Manager". Similar packages and package managers also exist on RedHat Linux, SuSE, etc., where you can proceed analogously.

- A graphics driver with 3D hardware acceleration (the right driver is usually auto-detected and installed by the "Driver Manager").
- **build-essential** – The compiler and basic tools required to compile C and C++ programs.
- **libgtk2.0-dev** – Developer files for GTK 2.0, needed for building wxGTK.
- **libgl1-mesa-dev** and **libglu1-mesa-dev** – OpenGL developer files, needed for building wxGTK and the Cafu rendering subsystem.
- **libxxf86vm-dev** – An interface to the XFree86-VidModeExtension extension.
- **libasound2-dev** – ALSA developer files, needed for building OpenAL-Soft.

Make sure that when you're done, all packages from the list and their dependencies are installed on your Linux system.

34.4 Compiler Setup

Cafu can be compiled as 32-bit and 64-bit software on all platforms. The following compilers are supported:

Windows	Remarks
Visual C++ 2015	You can use the free Express Editions with Cafu. They are available at http://www.microsoft.com/express/vc/ and automatically install the related Microsoft Windows SDK.
Visual C++ 2013	
Visual C++ 2012	
Visual C++ 2010	
Linux	Remarks
GCC 4.4+	The GNU Compiler Collection with its C++ front-end, version 4.4 or any newer version.

The Cafu-specific compiler setup is fully automatic: No action is required in this step, and you can directly proceed to the next.

Optionally, if you want to change the default settings now or later, here is an overview of how it works:

1. When SCons is run in the next step, it first determines if file `Cafu/CompilerSetup.py` already exists. If it doesn't (e.g. when you do this for the first time), it automatically creates the file as a copy of `Cafu/CompilerSetup.py.tmpl`.
2. Alternatively, you can also copy `Cafu/CompilerSetup.py.tmpl` to `Cafu/CompilerSetup.py` manually. This is only required once, and only if you want to edit `Cafu/CompilerSetup.py` before SCons is run for the first time.

3. You can edit `Cafu/CompilerSetup.py` in order to set the compiler and tools that are used to build Cafu, and to set the target architecture (such as `x86` or `x86_64`) that it is build for. Each setting is well documented, so you should have no problems to make the desired changes. The default settings automatically determine the latest installed compiler and the current architecture, so in most cases the file works out-of-the-box and you have to change nothing at all. Also if you have doubts about the right settings, just continue and use the file as-is.

If you do this for the first time, we recommend that you don't bother with `Cafu/CompilerSetup.py` at all. Just let the automatism determine the settings and come back later if desired or required.

34.4.1 64-bit Windows with Visual C++ 2010 or 2012 Express Editions

On 64-bit Windows systems, SCons tries to find a 64-bit compiler, but Visual C++ 2010 Express Edition comes with a 32-bit compiler only. To tell SCons to use the 32-bit compiler instead, please edit file `Cafu/CompilerSetup.py` as described here.

Even though the Visual C++ 2012 Express Edition comes with a 64-bit compiler, at this time the same edits must be done for it as well.

34.5 Compiling the Source Code

You are now ready compile the Cafu source code. At the command prompt, change into the top-level `Cafu` directory (where also the files `CompilerSetup.py(.tmpl)` and `SConstruct` are), then start the compilation with:

```
> scons
```

When you do this for the first time, be prepared that it will take a while: Everything is being built from scratch, once for the debug and once for the release build. When you repeat the same command in the future, e.g. after source code changes or updates, only the minimum set of files will be rebuilt and the entire process will complete *much* faster.

Under Linux, if you have a multi-core system, you can speed this up by having several build jobs run in parallel:

```
> scons -j N
```

where N is the maximum number of jobs to start simultaneously. For example, try `-j 4` on a quad-core machine.

34.6 Adding Binary Assets

You may use the time while the build process is running to download some supplemental files. These files are not strictly required, but will help you with trying out the newly compiled programs as described below.

We recommend that you download and extract the following files:

Download	Extract into
Textures.zip	`Cafu/Games/DeathMatch/Textures`
Worlds.zip	`Cafu/Games/DeathMatch/Worlds`

Note that although these files are zip archives, they are for use on all platforms (Windows and Linux). With these files, you can immediately run the Cafu engine and see some example worlds when the compilation has finished.

34.7 Running the new Programs

When the SCons script has finished, all Cafu programs have been built in several variants:

- The `debug` variant contains additional program information and no code optimizations, and is often used during development and for finding and fixing bugs.
- The `release` variant has program optimizations enabled in order to maximize the execution speed and contains no debug information. This variant is usually shipped to end users.
- The `profile` variant is not built by default. It is similar to the `release` variant, but also has debug information in order to make it suitable for use with a performance profiler.

As all variants can be built for each compiler on each platform, the build process organizes the binaries in a directory hierarchy of the form

```
build/$platform/$compiler/$variant
```

That is, if your compiler is Visual C++ 2015 (abbreviated as `vc14`) and your platform is Windows (abbreviated as `win32`), and you want to run the release variant of the new programs, then the related binaries are relative to the path

```
build/win32/vc14/release/
```

Another important consideration is that all programs assume the top-level `Cafu` directory as their working directory.

Combining these facts provides you with several options on how you can run the new programs:

1. Copy the programs of the platform, compiler and build variant of your choice from `Cafu/build/$platform/$compiler/$variant/...` directly into the `Cafu` directory. This is what we do for the prepackaged binary releases, as end-users can then simply and directly double-click each executable in the Windows Explorer.
2. Create a Windows link (`.lnk`) or batch (`.bat`) file in the proper working directory that calls the desired program in its original location.
3. For developers, the most flexible and recommended approach is to run the program directly from the command prompt: First change the current working directory to `Cafu`, then type the relative path to the desired program (TAB completion – also on Windows – proves to be a great help here!). For example, running the debug version of CaWE under Windows looks like this (note that the example also shows the current working directory at the left, the actual command follows the > character):

   ```
   D:\Cafu> build\win32\vc14\debug\CaWE\CaWE.exe
   ```

   Running the release version of Cafu under Linux looks like this:

   ```
   ~/Cafu> ./build/linux2/g++/release/Ca3DE/Cafu
   ```

If you have downloaded the binary assets as instructed above, you can now run your self-compiled Cafu engine e.g. at `build\win32\vc14\release\Ca3DE\Cafu.exe` and try one of the included demo levels!

34.8 What next?

Congratulations! When you get here, you have managed to successfully compile the Cafu source code and run the resulting binaries. You may now wish to familiarize yourself more completely with the world of Cafu.

One way to do this is to approach Cafu from a map designers perspective, for example by exploring the Cafu World Editor CaWE. CaWE is not only a map editor, it also contains the GUI editor, a font conversion tool, a materials viewer and a model editor. CaWE can also act as a bridge towards scripting: map scripting, GUI scripting, materials scripting and vegetation scripting are powerful features in Cafu, and it doesn't stop there!

Programmers should definitively have a look at the other sections in this chapter (At the Core: The Cafu Source Code), such as *Starting your own Game*. If instead you prefer to dig into the source code right away – just pick your favorite piece of code and start reading and hacking.

Consider *Using the Autodesk FBX SDK* in order to add support for several additional file formats in the Model Editor.

Or you just browse the files and folders in the `Cafu` directory, and start with whatever you feel most attracted to.

In any case, if you have questions or comments, or if you need help, post a message at the support forums at any time!

CHAPTER 35

How to Submit Patches

Cafu is to a community project and we need and very much appreciate your help. Your contributions and especially patches are very important for Cafu. Patches help us to add new features, improve code quality and fix bugs, so we are happy and grateful if you contribute them.

35.1 Changing Cafu

Please read the *Coding Conventions* and try to conform to them. In particular, please respect the indentation rules (4 spaces, no TABs) – patches are really difficult to read otherwise.

35.1.1 Provide documentation

Bug fixes and elegant program solutions often involve complex code that can be difficult to understand without documentation, and an undocumented new feature is hardly useful to anyone but its author.

Therefore, please provide any required documentation such as source code comments (preferably in familiar and simple Doxygen format), high-level overviews, diagrams, etc. Without documentation, another developer would have to write it, and the patch cannot be applied until he has time to do it.

35.1.2 Make atomic patches

A patch should be self-contained – *one patch for one thing*.

Do not combine multiple new features in a single patch: A patch that adds bitmaps to menu items and fixes a bug in the network code is a bad patch. It should be splitted into two patches.

On the other hand, do not split a single code change into multiple patches: Two patches, one of them being "implementation of new member-functions", the other "changes in class documentation to accommodate new members" are two bad patches. They are related to one, logically indivisible thing, so they should be in one common patch.

35.2 Final Considerations

1. On Dive-In, we need "How to contribute" / "Fork me on Bitbucket".
2. Why Your Project Doesn't Need a Contributor Licensing Agreement, http://ebb.org/bkuhn/blog/2014/06/09/do-not-need-cla.html
3. https://kernel.org/doc/html/latest/process/submitting-patches.html

CHAPTER 36

Coding Conventions

This chapter outlines the basic styles and patterns that are used throughout the Cafu source code.

These guidelines intend to achieve uniform code that is easy to read and maintain by many persons, and should be applied to all newly written code. Old code that does not comply to these rules can still be found in some places, and should be updated to conform to these guidelines on convenient occasions.

Of course, the guidelines described herein are not meant to be exhaustive, and breaking one of these rules in well justified cases (usually to increase readability) is an option.

36.1 Control Structures

With respect to brackets positioning, indentation and whitespace, control structures are laid out like this:

```
if (...)
{
    ...;
}
else if (...)
{
    ...;
}
else
{
    ...;
}

for (...; ...; ...)
{
    ...;
}

while (...)
{
```

(continues on next page)

(continued from previous page)

```
    ...;
}

do
{
    ...;
}
while (...);

switch (...)
{
    case 1: ...; break;
    case 2: ...; break;
    case 3: ...; break;

    case 4:
        ...;
        break;

    case 5:
    {
        // Extra parentheses are needed for declaring local variables here.
        ...;
        break;
    }

    default:
        ...;
        break;
}

// "if" variants that are also OK (similar applies to "for" and "while").
if (...)            // Omit unnecessary brackets when there is only a single␣
↪statement.
    ...;

if (...) ...;       // Optionally omit the line break.

if (...) ...;       // Special vertical alignment.
    else ...;

     if (...) ...;  // Another special vertical alignment (rare).
else if (...) ...;
else          ...;
```

- Indentation is **always 4 spaces** (never a TAB (**!**)).
- Braces are always vertically aligned and each is on a line of its own.
- Note the space in `if (`, `for (`, `while (`, etc.
- Note the space after the semicolons in the `for` statement.
- There is one empty line after the closing brace of a multi-line control structure.
- Omit unnecessary braces (but beware of cascaded `if` statements!).

36.2 Classes

Class declarations follow this pattern:

```
class MyClassT : public Base
{
    public:

    MyClassT(float x_=0);

    float GetX() const { return x; }


    private:

    float x;
};
```

- Note the `T` suffix (short for “Type”) at the end of the class name, indicating a custom type.
- Classes that are used as interfaces (ABCs (abstract base classes), pure-virtual classes) are suffixed by a capital `I` (short for “Interface”) instead.
- Definition of methods in-line is OK if the method is short (as `GetX()` above). Most methods should instead be defined as described below.

36.3 Methods and Functions

The definition of methods and functions follows this pattern:

```
MyClassT::MyClassT(float x_) : Base(...), x(x_)
{
    ...;
}


// Alternative constructor definition (preferred!).
MyClassT::MyClassT(float x_)
    : Base(...),
      x(x_),        // Note that indentation is now really 6 spaces.
      y(0)          // This is because vertical alignment takes precedence in this␣
↪case.
{
    ...;
}


float MyClassT::GetX() const
{
    return x;
}
```

- Note the **two** blank lines between each function definition.

36.4 Whitespace

As a general rule, use whitespace as you'd use it in written English prose.

That is, do *not* use whitespace

- with unary operators (e.g. `-`, `!`, `~`, `++`, `-` etc.),
- with binary arithmetic operators (e.g. `+`, `*`, etc.),
- with comparison operators (e.g. `==`, `>`, `>=`, etc.),
- with other operators like `=` (assignment), `::` (scope resolution), `->` (member by pointer), etc.

Not using whitespace around the above mentioned operators (all but unary) works best when your code editor supports syntax highlighting that renders the operators in a style or color that is different from that of the operands. If your code editor doesn't support highlighting of operators and/or using a space before and after the operator improves the readability of the code, using such whitespace is fine, too.

Use whitespace

- with logical and bitwise operators (e.g. `&&`, `||`, `&`, `|`, `^` etc.),
- with the ternary conditional operator (`cond ? a : b`),
- after commas (e.g. in lists) and semicolons (e.g. in for-loops),
- after keywords (e.g. `if`, `for`, `while` etc.).

(!) Whitespace must only ever consist of SPACE characters, never of TABs. Setup your text editor to automatically replace one press of the TAB key with the insertion of four SPACE characters. Avoid whitespace at the end of lines, setup your text editor to remove it automatically. Files must end with exactly *one* newline character.

36.5 Indentation

For the indentation of blocks of code, we use the "Allman" or "university" style, where the opening and closing brackets are vertically aligned and each bracket is on a separate line as shown in all examples above and below. The enclosed block of code is always indented by four spaces (*never* by a TAB character!).

Example:

```
bool IsValidTime(unsigned long Hours, unsigned long Minutes, unsigned long Seconds)
{
    if (Hours<24 && Minutes<60 && Seconds<60)
    {
        // Yes, this is a valid time.
        return true;
    }
    else
    {
        // No, this is invalid.
        return false;
    }
}
```

Of course, in real code this trivial example would be expressed as

```
bool IsValidTime(unsigned long Hours, unsigned long Minutes, unsigned long Seconds)
{
    return Hours<24 && Minutes<60 && Seconds<60;
}
```

which is both much shorter and more readable.

36.6 Vertical Alignment

Make use of vertical alignment. Obviously,

```
std::string Search     []={ "mo",     "tu",      "we" };
std::string Replacement[]={ "Monday", "Tuesday", "Wednesday" };
```

is more expressive and suggestive than

```
std::string Search[]={ "mo", "tu", "we" };
std::string Replacement[]={ "Monday", "Tuesday", "Wednesday" };
```

36.7 Parentheses

Use parentheses whenever necessary to clarify the operator precedence. Even if a set of parentheses is redundant with respect to the definition of the C++ language, use them e.g. when whitespace alone is insufficient or you had to lookup the proper precedence of not-so-frequently used operators in literature.

Both of these examples are fine:

```
bool IsValidTime(unsigned long Hours, unsigned long Minutes, unsigned long Seconds)
{
    // Okay: Whitespace (and programmer knowledge) alone make operator precedence␣
↪clear.
    // This is the preferred variant.
    return Hours<24 && Minutes<60 && Seconds<60;
}

bool IsValidTime2(unsigned long Hours, unsigned long Minutes, unsigned long Seconds)
{
    // Also okay: (redundant) parentheses make operator precedence explicitly clear.
    return (Hours<24) && (Minutes<60) && (Seconds<60);
}
```

Avoid "null" parentheses, though. For example

```
return (Hours<24 && Minutes<60 && Seconds<60);
```

in the above examples should rarely be written.

36.8 Comments

All comments are to be written in the English language as error-free as possible, with the proper or at least best possible spelling, grammar and punctuation. Prefer C++ comment styles over C comments:

```
int yes=1;     // Use // to initiate most comments.
int no =0;     /* Do NOT write C-sytle one-line comments like this. */
```

We use Doxygen for writing code documentation. Doxygen comments are to be written "inline", that is, into the header files with the declarations. Within Doxygen comments, we use the JavaDoc style with the `JAVADOC_AUTOBRIEF` option set to `YES`. Prefer /// for multi-line comments above the declaration and ///< for short single-line comments trailing the declaration.

36.9 Conclusions

The above presented coding conventions and guidelines are to achieve a uniform and readable code style. Exceptions should be kept to a minimum, but are allowed whenever they increase the readability.

36.10 References

These references give general information about coding conventions, listed here to help improving this section. The coding conventions that they describe may differ from the conventions established here, and are not meant to apply to Cafu.

- http://de.wikipedia.org/wiki/Quelltextformatierung
- http://de.wikipedia.org/wiki/Programmierstil
- http://de.wikipedia.org/wiki/Einr%C3%BCckungsstil
- http://en.wikipedia.org/wiki/Code_convention

CHAPTER 37

Selected Topics

37.1 Game Development Overview

The Cafu Engine can be used both for developing single- or multiplayer games that ship as individual products, as well as for products that support multiple, independent games in a single program that the user can select from at runtime.

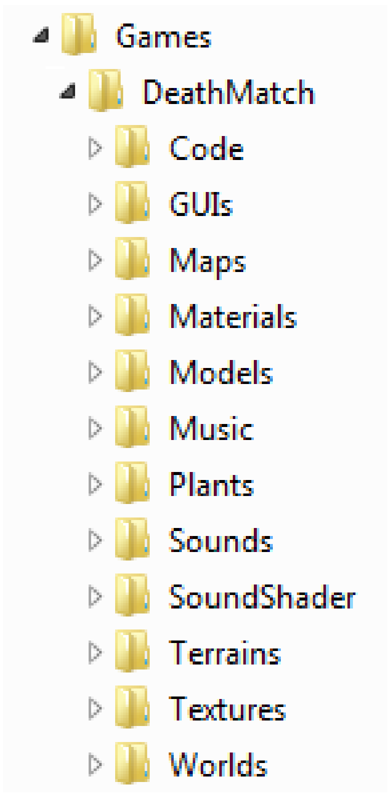
For example, after starting the program, the user might choose from a list of online game servers (each possibly running not only different game maps, but entirely different games), or a list of available single-player games, and join one seamlessly without restarting the program.

The fact the multiple games can be present at the same time makes it worthwhile to keep all related files and program code well organized, so that games can be developed and shipped to end users modularly and independently of each other and of the Cafu core engine.

37.1.1 One directory per game

In order to achieve these goals, all files, data and program code for a game are stored in one common directory, which in turn must be a subdirectory of `Games/`.

For an example, see the `Games/DeathMatch/` directory of the DeathMatch example game that ships with Cafu:



(Also see *Starting your own Game* for help about creating a new game directory.)

Although you're largely free to organize the contents of the game directory as desired, per convention some of the resource files must be kept under fixed names or in given directories, so that the Cafu Engine can find and load them when it is newly initializing the game.

The following sub-sections list and explain most of these "fixed" files and directories.

GUIs/

The graphical user interfaces that are used in a game are kept in `GUIs/`.

GUIs are used both

- in "2D", e.g. for the main menu, the console window, the "HUD" (head-up display) in the game, etc., as well as
- in "3D", as parts of the game worlds, that the player can use e.g. to call lifts, to unlock doors, to activate teleporters, or to control any other game script action.

The GUIs kept in `GUIs/` are usually "2D" GUIs, and "3D" GUIs that are universally used in multiple maps. "3D" GUIs that are specific to a single map can either be stored in `GUIs/` as well, or along with the map in the same directory as the map.

At this time, the following GUIs are automatically loaded by the Cafu Engine:

- `GUIs/ChatInput.cgui` – implements the text input when the player presses `T` to talk in the game.
- `GUIs/Console.cgui` – implements the in-game command console, opened with `F1`.
- `GUIs/MainMenu/MainMenu_main.cgui` – implements the main menu that is shown after program start.

In addition, the DeathMatch game code in `DeathMatch/Code/HumanPlayer.cpp` loads

- `GUIs/HUD_main.cgui` – implements the players head-up display (cross-hair, ammo count, etc.).

You can replace all these GUIs by providing your own editions in their place, or just copy them from the DeathMatch example game and modify them as you like.

Maps/

Contains the map files as saved by the CaWE Map Editor.

You don't actually have to save your map files in this directory, but we may merge `Maps/` and `Worlds/` sometime in the future, and thus `Maps/` is still the recommended place to save `.cmap` files.

Materials/

All material definitions for the game maps are kept in `Materials/`. Materials are often reused in several maps of a game, and thus `Materials/` is where all map materials are centrally stored.

The CaWE Map Editor loads the material definitions from this folder in order to provide the map designer with the choices in the Material Browser dialog. The Cafu Engine loads the material definitions from `Materials/` as well in order to render the materials in the game.

Music/NextTitle.txt

Whenever the player enters a new map, the client calls console function `StartLevelIntroMusic()`, that in turn is defined in file `config.lua` in the Cafu top level directory and that uses file `Music/NextTitle.txt` in order to play the next title from the list of files in `Music/`.

Textures/

The texture images that the materials in `Materials/` are referring to are stored here.

Worlds/

Maps from `Maps/` must be "compiled" in order to be used with the Cafu Engine. Compiling turns a map file (suffix `.cmap`) into a world file (suffix `.cw`) and is documented at *Compiling Maps for Cafu*.

The world files generated by compiling are stored in `Worlds/`, where the main menu GUI (`MainMenu_main.cgui` in the DeathMatch example game) looks for `.cw` files that the player can choose to enter. It also looks for a matching screenshot image to show with the maps list.

The `Worlds/` directory is also the right place for map scripts:

- A map scripts must be placed into the `Worlds/` directory "next" to the compiled world file. That is, the script must have the same base name as the world, but suffix `.lua` instead of `.cw`.
- The map script is *not* compiled along with the `.cw` file: You can change it at any time, and the changes become effective as soon as the world is loaded/restarted the next time. In fact, it is even possible to *update a map script while the game is running*, that is, *without interrupting it*.

37.1.2 Exploring the code

The class hierarchy is shown in `Ca3DE/ClassDiagram.dia`

37.2 Starting your own Game

After you took *the first steps with the Cafu source code*, you're ready to use it to develop your own game or application.

This section lists the most commonly used methods for starting new projects and explains the required steps to give you a good start.

37.2.1 Cloning DeathMatch

The "normal" and recommended way to start a new game is to create a copy of the example game DeathMatch under a different name. This is useful because it immediately provides you with the proper directory structure, working code, and example files.

Creating the clone

Creating the copy is straightforward:

1. In `Games`, duplicate the directory `DeathMatch` and name the resulting copy as desired, e.g. `MyGame`.
2. Delete the directory `build` in `Games/MyGame/Code`.

The second step is necessary because previous builds of DeathMatch may have created files whose copy can confuse the linker when it is creating the program library for `MyGame` for the first time.

To implement these steps, you can use any method you like, for example a file manager like Windows Explorer, or the command-line. This example shows how you can do it under Linux:

```
$ cp -r Games/DeathMatch/ Games/MyGame/
$ rm -rf Games/MyGame/Code/build
```

Using your new game

Technically, the above steps are all that is required to create the basis for your new game:

- When you re-run SCons as documented at *Getting Started with the Cafu Source Code*, the code of your new game will automatically be picked up, compiled and linked.
- When you start CaWE, it will automatically recognize the new game and show it in the "Configure", "New Map", "New GUI", ... etc. dialogs. If "Start Engine" is checked in the "Compile" menu, it will also start the Cafu Engine with the new game.
- If you run the Cafu Engine from the command prompt, make sure to add parameter

  ```
  -svGame MyGame
  ```

  to the command-line (see below how to get rid of it again).

Tips and Tricks

- If you want to run your game by default (without `-svGame`), see the section about the variable `GameLibs` in file `CompilerSetup.py`: .. code:: python

  # When you're developing your own game, you might want to keep game DeathMatch # for reference. Cafu should run your game by default (first in list), whereas # DeathMatch would be readily available via the -svGame command-line option: GameLibs = ["MyGame", "DeathMatch"]

Note that you have to re-run SCons in order for changes to take effect.

- In `MyGame`, some paths in the code and scripts still point to `Games/DeathMatch/...`. For the beginning, that's ok – your game will find some of its resources in game DeathMatch, and you can update these occurrences step by step at any later time (or even intentionally keep things like this).
- Check out *Game Development Overview* for details about the files and subdirectories in the new game directory.

37.2.2 Alternatives

Here is a list of alternative courses of action that you may want to consider in order to start a new game:

Modifying DeathMatch

Working directly with the DeathMatch game "in situ" may seem like a totally silly idea to you, but in some cases it can be a worthwhile consideration nevertheless:

If you want to quickly try something out (e.g. a new entity class), or if you work on something that is not directly or entirely a completely new game (for example, any changes to the core engine), the game DeathMatch might be a welcome testbed.

Also, if you wish to communicate and collaborate with people who are not directly involved with your game code, the DeathMatch code can be a good common ground.

Renaming DeathMatch

Renaming DeathMatch, instead of properly copying it, would principally work, if it was not for these factors:

- It does not agree well with version control systems (Git or Subversion). If you follow our recommendation to checkout the Cafu source code from our Git or Subversion repository, so that you can keep track of our latest developments while developing your own game in parallel, then renaming the DeathMatch directory is (indeed not impossible, but) somewhat incompatible with the way how these repositories work.
- Some file paths in code and scripts still point to `Games/DeathMatch/...`. If you rename DeathMatch, all these paths are broken, and your game will not work without fixing them first. If you instead copy DeathMatch, your game will find some of its resources in game DeathMatch, and you can update these occurrences step by step at any later time (or even intentionally keep things like this).

Overall, we recommend *not* to rename DeathMatch. Better clone it as explained above.

Starting from scratch

Creating a new, empty directory in `Games/` in order to start from scratch, then fill it with contents as required, is certainly possible.

Doing this can be a very good choice if your main goal is to learn and understand the technical details from the ground up, or if you only want to have original content in your game right from the start.

In fact, we're considering complementing the DeathMatch example game with another "Minimal" example game that comes closer to these goals.

If you start with an empty directory,

- your progress will be slower,
- despite the support forums are always available, you should be technically versed enough to cope with most problems and errors,

- be prepared to copy pieces of DeathMatch example code anyway.

Using VSWM

The VSWM MOD is old, obsolete, dysfunctional, and a candidate for removal. Not a good basis for a new game. Don't use it.

37.3 Loading game worlds

While we have introduced the GUI and entity component systems at several occasions:

1. todo

this document provides an overview of how the Cafu programs actually instantiate a game world that is composed of game entities.

All Cafu programs can load game worlds, i.e. the game server, client, the world editor CaWE and the compile tools. We will use the server's code for reference, but as the code is similar in all programs, we will cover the specifics of other programs as well.

Also use the class diagram in `Ca3DE/ClassDiagram.dia` besides this text for reference.

The server loads a game world by creating a `CaServerWorldT` instance which derives from `Ca3DEWorldT`.

The `Ca3DEWorldT`'s `const WorldT* m_World` member contains the data from the BSP, PVS and Radiosity compile process: for each static entity, it has the BSP tree, a collision model, lightmaps, etc. (**(FIXME!)** not only static, but in fact for *all* entities in the cmap?) This data is later correlated to (and becomes a component of) the main entity instances `cf::GameSys::EntityT`.

The `Ca3DEWorldT` keeps an instance of a `cf::UniScriptStateT`. This script state is a Lua instance to which later our objects are bound so that we then can write code such as:

```
local new_ent = world:new("EntityT", "left_wing")
Door:AddChild(new_ent)
new_ent:GetTransform():set("Origin", 480, -352, 104)
```

Note that *both* game entities *and* GUI windows, our two sibling component systems, can be instantiated in the script state of a `Ca3DEWorldT`.

37.4 Using the Autodesk FBX SDK

When you build Cafu from source, support for several file formats is provided by Cafu's own source code so that the Model Editor can import static and animated models in the most important file formats.

Optionally, it is possible to have the build scripts account for and use the Autodesk FBX SDK in order enable the Model Editor to import model files in these additional file formats as well:

- Autodesk FBX (`.fbx`), version 6.0 to 7.2
- Autodesk AutoCAD DXF (`.dxf`), version 13 and earlier
- Collada DAE (`.dae`), version 1.5 and earlier
- 3D Studio 3DS (`.3ds`), all versions
- Alias OBJ (`.obj`), all versions

The FBX SDK is a separate download from Autodesk that we cannot distribute together with Cafu: The combined file size for all platforms is *massive*, Autodesk prefers that software vendors don't redistribute their SDK and in fact requires prior written permission, and we're happy to maintain a certain degree of independence, for example when we want to use Cafu with compilers or on platforms where the FBX SDK is not available.

Therefore, we made the use of the Autodesk FBX SDK very easy, but entirely optional. If you don't use it, all that you're missing is that the Model Editor cannot load the above mentioned file formats. (When we make official binary releases of Cafu, we always ship with FBX support enabled.)

Follow these steps in order to activate support for FBX:

1. Download the Autodesk FBX SDK from http://www.autodesk.com/fbx
   - The download requires a one-time registration, but it's free.
   - You just need the FBX SDK, *not* the FBX Extensions SDK.
   - At this time, we use version 2017.1 of the FBX SDK, but any later version should work as well.

2. Run the installer to extract the SDK contents into a convenient location. Their installer is exceptionally nice, it neither modifies the Start Menu nor the Windows Registry, but just extracts the files into a directory.
    - Assuming that your top level Cafu source code directory is `Cafu/`, you can directly install the FBX SDK into `Cafu/ExtLibs/fbx/`. Make sure that the installer doesn't automatically append another directory for the SDK version, and if it asks if you would like to keep a copy of the old FBX installation, answer "No" (otherwise it first renames `Cafu/ExtLibs/fbx/` to `Cafu/ExtLibs/fbx.old/`, creating an (easily fixed) problem if you use Git).
    - Alternatively, just install the FBX SDK into a convenient location first (e.g. into a temporary directory or the default directory suggested by the installer), then copy or move the contents into `Cafu/ExtLibs/fbx/`. (Under Linux, we recommend to *not* install into the suggested `/usr` directory. Better install into a temporary directory in your home folder, no root privileges required.)
3. As a result from the previous step, make sure that you now have directories `examples`, `include` and `lib` (and possibly others) immediately below `Cafu/ExtLibs/fbx/`.
4. (Re-)Run SCons as described at *Getting Started with the Cafu Source Code* in order to automatically recompile with FBX support enabled.