# Getting Started Mapping

## From Quake Wiki

Revision as of 12:46, 9 September 2022 by 79.224.223.58 (talk)
(diff) ← Older revision | Latest revision (diff) | Newer revision → (diff)

This article is meant as a basic introduction to how a Quake level is constructed in the modern Quake editing software TrenchBroom. The goal is to teach a new level designer what goes into making a map, the terms used, what compiling is, and how to compile and play a created map. If you have ever created levels for the other Quake games, or for the Half-life/Source engine games, the basic terms of this will be familiar as they evolved from Quake.

## Contents

## What's a map!?

A Quake level is created in level editing software specifically designed for the task. There are several editors available, and all of them create .map files. These .map files are used by Quake compilers to generate the final .bsp file levels that can be loaded by the Quake engine. Think of the .map file as a blueprint, and it contains instructions of how to make all the solid geometry of a level, and where all the lights and monsters and pickups should go. There are 2 major objects used to make these instructions: Brushes and Entities.

### Brush: Your basic building block!

Our basic building block for constructing a level is the brush. All of our level's solid geometry will be constructed from brushes. But what do we mean by 'brushes'? The semi-technical definition in Quake level creation is that they are convex (http://en.wikipedia.org/wiki/Convex) polyhedrons (http://en.wikipedia.org/wiki/Polyhedron). The less technical definition is that they are 3D objects made of faces which cannot 'see' each other. Most commonly, you will use cubes (http://en.wikipedia.org/wiki/Cube) or cuboids (http://en.wikipedia.org/wiki/Cuboid), but know that any convex polyhedron is allowed.

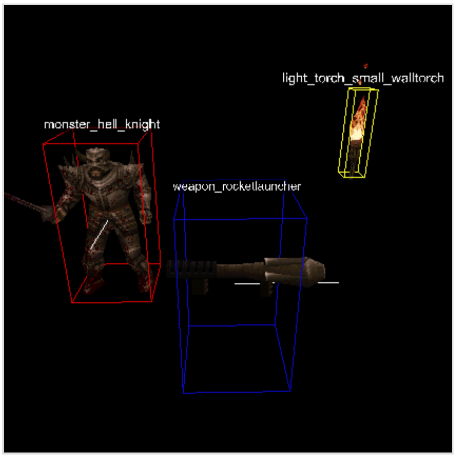
It's ok if your head is spinning from reading all that, even if you don't fully understand that definition, using brushes in Quake editing software is easy, and the hard work of mathematically defining them is done for you, hidden away in the background. All you need to do is plop them down, and arrange them to make the walls and floors of your level!

### Entities: The life of the party

So we have brushes to define our world's geometry... but it wouldn't be a Quake level without weapons and monsters! And entities are just that, they are any of the functional objects defined in the game code for you to place into your level.

There are 2 types of entities: Brush entities and Point entities. Brush entities are things like doors, platforms, and trigger volumes; they are any functional object which need brush geometry tied to them to do their job. Point entities are things like weapons, monsters, and lights; they are all objects which are just simply dropped into place (at a point) in the world.

Entities of both types have various properties that can be edited by the designer to modify specified effects on the entity. Each property is a combination of a 'key', which is the name of the properties, and it's 'value'. As an example, light entities have a key called 'light', and its value is set to whatever brightness you want the light to be.



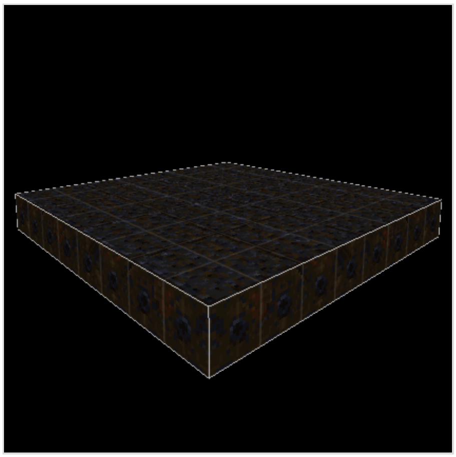
A few entities

## Textures Wads and Compilers

To round out our definitions, let me also talk about some external things we will need along with our .map file to create our final .bsp level.

Each face of a brush is allowed 1 texture, which can be rotated, scaled, and translated. But where do we get our textures? They are stored in .wad files, which are a collection of textures to be used in levels. If you are familiar with older Doom/Doom2 level design, these are NOT the same as their WAD files (http://doomwiki.org/wiki/WAD), despite having the same name.

We also need compilers, which take our raw .map file and turn it into a .bsp file which Quake can load. There are 3 compilers which are used: QBSP, which turns the .map into a .bsp. Light, which calculates all the lighting information in the .bsp using our .map's instructions. And Vis, which calculates visibility in the level to optimize Quake's rendering.


A brush

# Tools of the trade

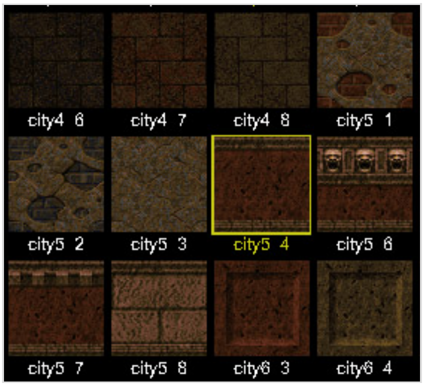
Now that we have definitions out of the way, let's get all the tools we will need. Trenchbroom you will need to install, the others are loose files which you will need to put somewhere accessible. I personally have a folder on my hard drive E:\q1maps\ where I keep all my .wads, compiling tools, .maps, etc.

**Editing Software**: This tutorial will use Trenchbroom. [Homepage (https://trenchbroom.github.io/)]

**A texture .wad**: We will use Q.wad, which are all the textures from Quake. [Download (http://www.quaddicted.com/files/wads/q.zip)]

**Map compiler tools**: This tutorial will use Tyrann's TyrUtils. [Homepage (http://disenchant.net/utils/)]

**Necros' CompilingGUI** *(optional)*: This is a handy front end for the compilers so you do not have to use command prompt or batch files to run the compilers. [Homepage (http://shoresofnis.wordpress.com/2010/03/30/ne_q1spcompilinggui/)]



Some textures

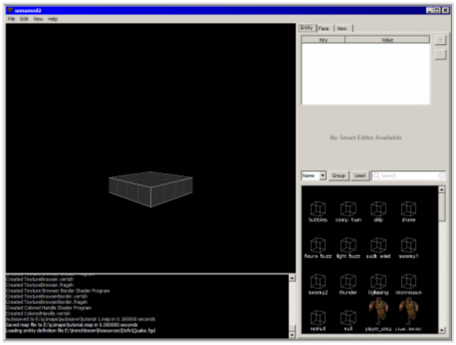
# Putting it together

Now that we have definitions out of the way, let's open up Trenchbroom and actually build a simple level.

## Some setup

When you first open Trenchbroom, you should have something much like the image on the right, but before we dive into some editing, there's a couple of things to set up. First, you likely won't see the little Quakeguy models, or any of the entity models, in the lower right section of your Trenchbroom in the Entity Browser. To fix this, go to **View>Preferences...** and set your Quake Path to where your Quake is installed (ex: C:\Quake\). Once you've set this, Trenchbroom will remember it for all future maps you make. Other settings here include OpenGL display settings, and Mouse sensitivity and axis inverting if you wish to modify these.

Next, we need to let Trenchbroom know which textures we want to use for this map. Go to **Edit>Map Properties...** and in the lower half of this dialog, you'll see the empty list of Texture Wads. Click on the **+** button, and browse to the **Q.wad** you downloaded earlier. You will be asked how you would like to store the path, leave the default 'Absolute' selected, and click Ok. Close out of the Map Properties dialog, and click on the **Face tab** in the upper right of Trenchbroom. You should now see all the Quake textures loaded in the lower right browser. Every time you create a new map, you will need to follow these steps to choose which texture wad you'd like to use.


Trenchbroom

## Painting the town

Now we're ready to begin our map. Our brush is currently a sad, untextured grey. Let's fix that by putting a texture on it. **Select the brush** in the 3D view by **Left Clicking** on it, it will shade red when selected, and show some 'laser lines' extending from it's bounds. Now, let's put a texture on it. We're going to make this a floor, so find a good floor texture in the browser and **Left Click** on it to **apply** it. I'm using *city4_2*.

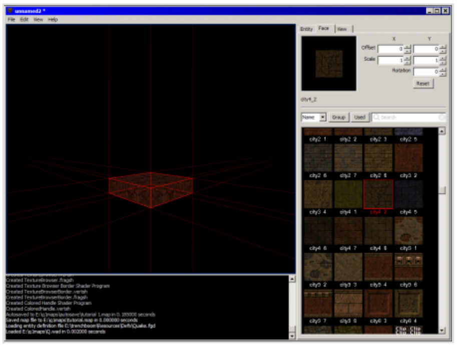
You can **select a single face** of a brush by **Shift-Left Clicking** a face, and **select multiple faces** by **Ctrl-Shift-Left Clicking**. In the upper right of Trenchbroom, we can also choose to offset, scale, or rotate this texture on our brush. Textures dimensions Quake must be powers of 2 (http://en.wikipedia.org/wiki/Power_of_two), often 64x64 or 128x128, so you will generally shift them by 1, 2, 4, 8, 16, 32, or 64 units. Scale is a multiplier, so to make textures 'smaller' you will use numbers smaller than 1, such as 0.5 for half-scale. Scale can also be use to mirror a texture, so a scale of -1 will flip it. Rotation is in degrees, 0-360, and negative values are allowed.

## Making some room

Our floor is a little small, let's make it a bit larger, and also familiarize ourselves with the 3D view a bit. To **look around** in the 3D view, **Right Click and Hold** in the 3D view while dragging your mouse. To **pan** left/right up/down, **Middle Click and Hold** in the 3D view. You can also **orbit** the camera by pressing **Alt-Right Click and**

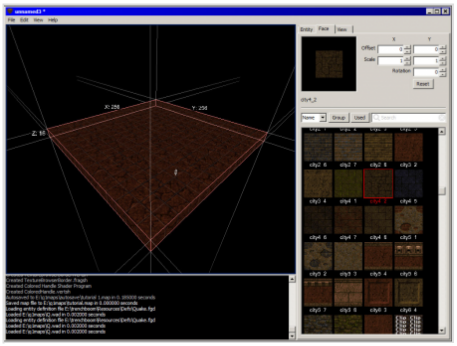
**Hold**.

Now, let's resize our floor brush. To do this, the brush must be selected, so reselect it if it is not or you only have some of it's faces selected. Now, **Hold down Shift** to enter face dragging mode and display the brush's dimensions. You will notice as you mouse over different parts of your brush, their edges will turn white, while continuing to **Hold down Shift**, you may **Left Click and Drag** a white face. Let's make our floor 256x256 wide and 16 units tall. You will notice that when resizing, you are constrained to 16 unit increments. You can change the **Grid Size** with **Ctrl--** and **Ctrl-+**, which will decrease or increase the grid by a power of 2, though it is recommended to stick to 16 units for this tutorial.


Applying Textures

## A new brush

Until now, we've only worked with the brush that Trenchbroom created for us, but now we want to make a wall, so we need a new brush. First, let's **Deselect** our floor by either **clicking in the black void**, or by pressing **Ctrl-Shift-A**. Now, to make a new brush, **Left Click and Drag** in the 3D view. While continuing to **Hold Left Click**, you effect the newly created brush's height by **Scrolling the MouseWheel**. Don't be concerned if the brush is not perfectly sized or positioned, you can always edit it. To **Move a Brush**, all you need to do is **Left Click and Drag** a selected brush. Let's position our wall at one of the edges of our floor, and size it 256x16x128 as shown in the picture to the right. Make sure the edges are perfectly aligned as we put brushes together. Also, give it a nice wall texture, I am using *city2_8*.

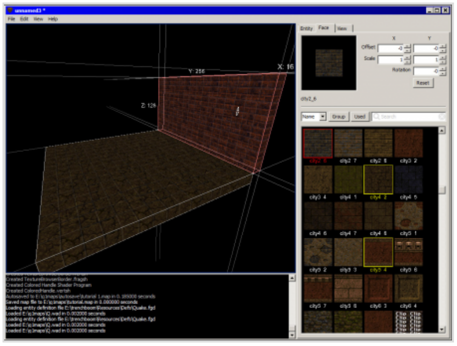

Resizing brushes

## Even more brushes

Let's keep going with constructing our room. To make another wall, let's **Duplicate** the wall we already have by selecting it, and pressing **Ctrl-D**. Now position it on the opposite edge of our floor. Let's also make a ceiling, so select the floor, and duplicate it as well. Now, we need our brush to **Move Vertically**, and for that, we **Hold down Alt** whilst moving our brush, notice your cursor changes to 2 arrows pointing up and down. Let's give this a nice ceiling texture, I am using *city5_3*.

Although not pictured to the right, let's close up our room now. **Select Multiple Brushes** by **Ctrl-Left Clicking** them, and select both of our walls. Duplicate them, and let's rotate them. You can quickly **Rotate 90° Horizontally** by pressing **Alt-Left Arrow** and **Alt-Right Arrow**. You can also **Rotate 90° Vertically** by pressing **Alt-Up Arrow** and **Alt-Down Arrow**. You can also **Flip Horizontally** with **Ctrl-F** and **Flip Vertically** with **Ctrl-Alt-F**.

Again, make sure all of our brushes have their edges aligned so our room is sealed. For our simple room this is not the end of the world if they are not, but it is good habit to get started with and will be required on larger, more complex maps to ensure they do not leak. Leaks occur when your brushes don't form a complete seal around the playable world, and prevent the compiling process Vis from running.


Creating new brushes

## Add some functionality

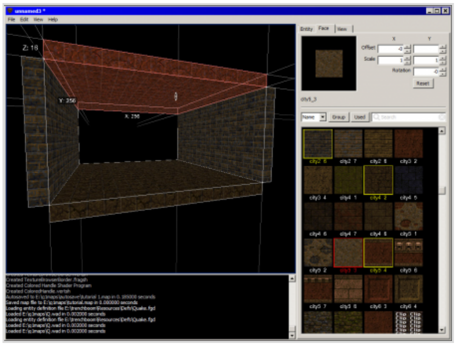
Now that we have a closed room, let's do some work so we can use it. The first thing to do is make a place for the player to start the level at. Go to the Entity tab in Trenchbroom, and scroll down a little in the Entity Browser on the bottom right of the editor. Find the model of Quakeguy with the words **info_player_start** under it. **Left Click and Drag** it into your 3D view to place it. info_player_start is an entity which does not need any more set up than that to work!

You do not always have to use the entity browser to make entities. You could also have done this by **Right Clicking** in the 3D view, and navigating the menu **Create Point Entity>Info>Player_start**.

## Lights and Properties

Technically we could compile and try our map now, but let's add some more stuff to our map first. How about a light? Either drag the entity **Light** from the entity browser, or in the **Right Click Menu** select **Create Point Entity>Light>Light**. Move this somewhere in the center of our room. Now, we do actually need to modify this entity for it to work right. With the light still selected and the Entity tab open, notice in the upper right corner of Trenchbroom our entity's keys and values. Trenchbroom, as of this writing, only puts some very basic keys and values into entities, so for now you will want to refer to the Entity Guide for all the keys and values you may need on your entities.

We want to set the brightness of our light. To do this, we need a **New Key**. Click on the **+ button** next to our list of keys, and rename our newly created key "light" and set it's value to "200".


Duplication

## Triggers!

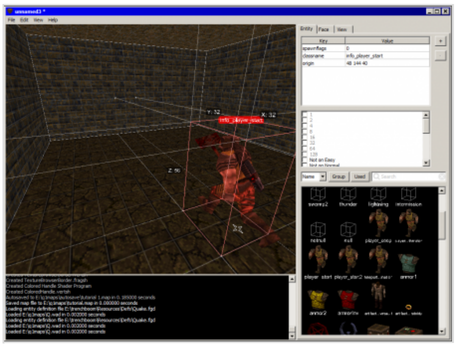
Let's also make sure everyone knows how great Quake is by telling them. To do this, we're going to make a brush entity that when the player walks into it, it displays a message. First, we need to make a brush as normal. This is going to be a trigger, a type of brush entity that is invisible and nonsolid, so it doesn't matter what texture is on our brush, but general convention is to use the texture *trigger*. Now, **Right Click** with our brush still selected, and go to **Create Brush Entity>Trigger>Multiple**. You should now see the text 'trigger_multiple' over our brush. Add some keys to this entity, "Message" with the value "Quake is Great!", and "wait" with the value "5". Message is the text we will display to the player, wait is how long the game should wait between triggering again so we aren't spamming our message constantly.

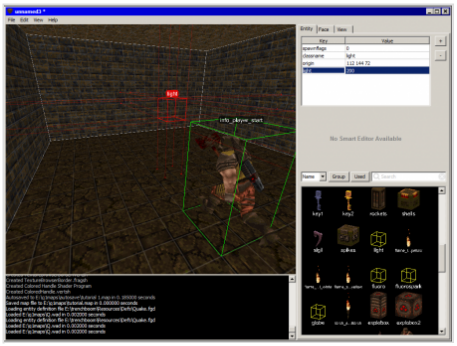
# Time to compile!

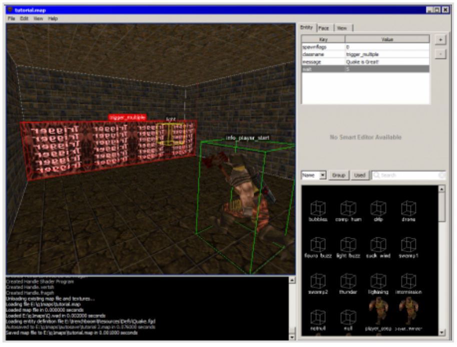
Let's save our map now with **File>Save** or **Ctrl-S**. Place it somewhere smart, we'll be using the .map we've created. The various Quake compilers are all command line interfaces (http://en.wikipedia.org/wiki/Command-line_interface), but let's use a nice front end to make it easy for us.

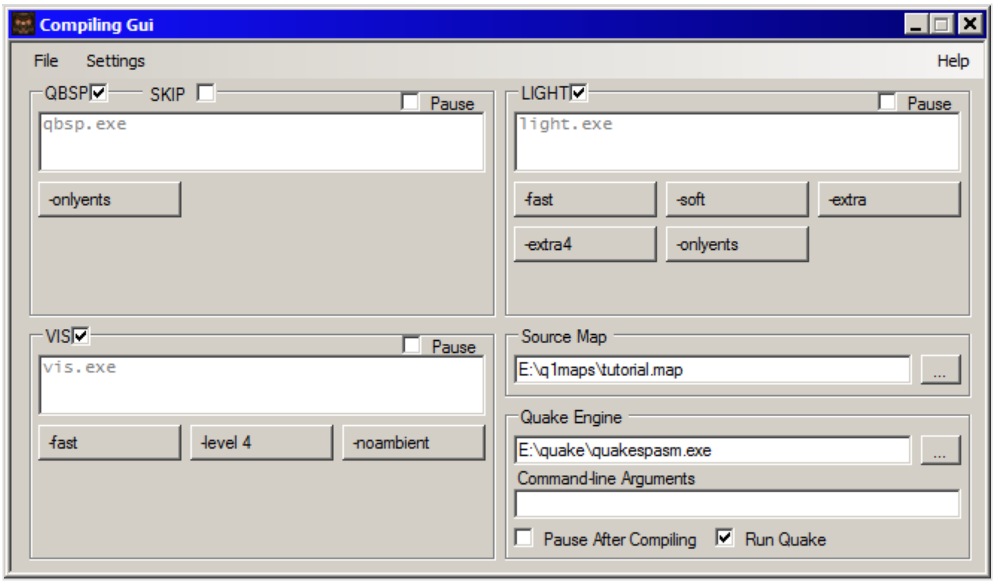
## Setting up the GUI

Now it's time to use Necros' CompilingGUI, as pictured below. Let's make sure this are set up properly before we begin.


Our first entity


Setting Properties


Making a Trigger



Compiling Gui

File Settings Help

QBSP ✓ SKIP ☐ ☐ Pause
qbsp.exe
-onlyents

LIGHT ✓ ☐ Pause
light.exe
-fast -soft -extra
-extra4 -onlyents

VIS ✓ ☐ Pause
vis.exe
-fast -level 4 -noambient

Source Map
E:\q1maps\tutorial.map

Quake Engine
E:\quake\quakespasm.exe

Command-line Arguments

☐ Pause After Compiling ✓ Run Quake

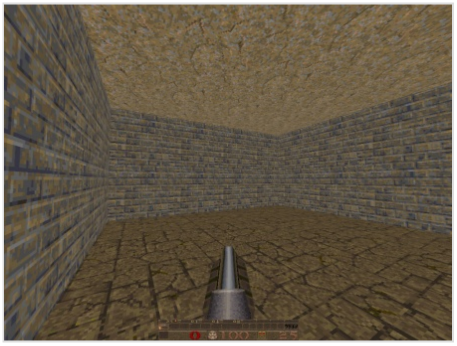
In **Settings>Folder Setup**, we need to point the program at the location of our tools folder, which is the directory on your computer where you places the TyrUtils compilers you downloaded earlier. Ensure that the .exe file names of each of the compilers is correct (do not worry about skip.exe, we will not be using it). For now, we do not need to set any Command-line Arguments.

Working Folder should be set to where you save your .map files. Output Folder needs to be set to Quake's **\id1\maps\** directory where it searches for levels, such as **C:\Quake\id1\maps\**. You may need to create this directory if you do not already have it. When mapping for mods, such as Quoth, this directory will be **\quoth\maps\** instead. Exit Preferences once this is all complete.

In the main dialog of the GUI, we need to set a couple of more things. First is the **Source Map**, which is the .map file we just created. We also need to set the **Quake Engine** we are using. This will be an executable inside your Quake directory, such as "GLQuake.exe" or "Quakespasm.exe" or "Darkplaces.exe". If you require any Command-line Arguments when running Quake, or if you are mapping for a mod (ex: "-hipnotic -game quoth" when making a map for the Quoth mod), place them in the line provided.

## QBSP

Now that things are set up, let's compile and try out our map! We're going to compile our map step by step to see the different parts in action, but you could run them all at once if you wanted as they will run in order. In the main dialog of the GUI, check **only** the box next to the word "QBSP", and the box in the lower right corner which says "Run Quake", and uncheck all other boxes. Now let's compile, go to **File>Compile** or press **Ctrl-C**. A command-prompt dialog box will appear on screen for a second or so, and then Quake should launch and you'll be playing your map!


Our map in game!

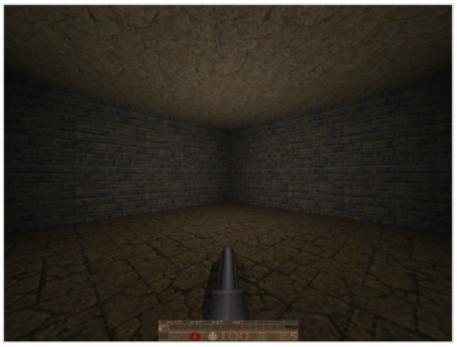
If you've been paying attention, you shouldn't be surprised to see that your map is a fully bright box with no shadowing. This is because we have only run QBSP on our .map. QBSP turns all our brushes into polygons which are nicely organized for Quake into a format called .bsp, in a process called Binary Space Partitioning (http://en.wikipedia.org/wiki/Binary_space_partitioning).

QBSP is also responsible for taking your textures out of your .wad and compiling them into the .bsp. This means you do not need to supply a .wad alongside your level when sending it to other people, and that a tool such as TexMex is able to open and extract textures from a .bsp file!

See QBSP for more information and common command-line arguments.

## Light

Once you're done examining our full-bright room, exit Quake and let's get some light working. Back in our compiling GUI, uncheck the box next to the word "QBSP", and check the box next to the word "LIGHT". Compile again, and once back in Quake, our map is now nicely shadowed!

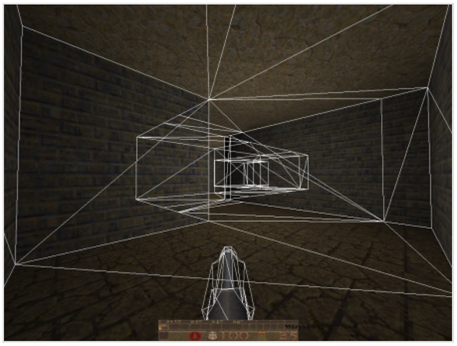

Light and Shadow

General lighting in Quake is not dynamic, but is calculated by our light compiler into what are known as "lightmaps", which are stored in our .bsp file. By default, all lights have a linear falloff, but more modern light compilers (such as the light compiler provided in TyrUtils) have functionality to support other falloffs. Light, especially with some of it's command-line arguments for improving details, on more complex maps can often take awhile to complete, so often when testing small changes to your map like a couple of new monsters, you may not bother running it. This may be obvious, but if you have no light entities in your map and run light, your map will be fully dark.

See Light (map compiling) for more information and common command-line arguments.

## Vis

You can now compile with Vis if you'd like. Unfortunately, our little single room map is a bit too simple to actually show what it does. Vis is a process which computes visibility of areas, and creates a table known as the Potentially Visible Set (http://en.wikipedia.org/wiki/Potentially_visible_set) (often shortened to PVS), which Quake uses to determine areas which the player cannot possibly see, and thus, does not need to render. Note that brush entities (or any entities), such as doors or triggers, do not block Vis, nor do special brushes like clips or liquids. Only solid brush geometry.

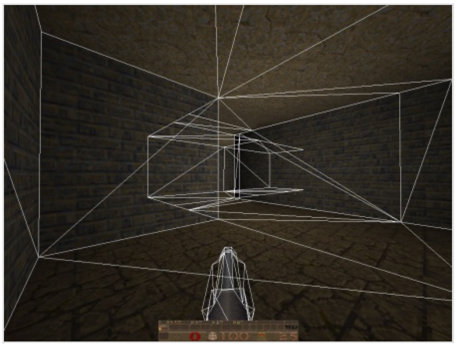

No PVS

I have constructed a simple snaking hallway map to demonstrate. In the pictures to the right, I have enabled "r_showtris 1" in Quakespasm, which outlines the polygons rendered by Quake. Notice that in the picture with No PVS, there are polygons in the distance around corners which are being rendered, which in the PVS picture, these polygons are no longer rendered. Even in this simple map it is not a huge amount of savings, but as your levels get larger and more complex, this will be a large optimization.

Vis requires that a .bsp be fully sealed to run. This means that all of your entities must be surrounded by brushes, and cannot trace a line into the void. QBSP will warn you when a .bsp leaks like this, and generate a pointfile which can be loaded in either Quake or Trenchbroom to help you find where you need to seal your level.

It should be noted that in general, Vis on a more complex map will be the compiling tool which takes the longest. Vis times are effected by the size of rooms, and the complexity of brushes. Large, open areas and very complex geometry can lead to massive compile times (days or weeks!) even on modern CPUs and are part of the reason why things like this are rare in Quake. When you are just starting out, it is advised to build areas the size and detail seen in stock Quake until you have a feel for how Vis times are effected.

See Vis for more information and common command-line arguments.


PVS