An introduction to optimization

Before we start on how to best optimize your map, you may want to fully know what you can expect from optimization. It's no use reading through this entire tutorial to find out at the end your problem can't be solved using optimization. So, here goes.

Optimization allows you to simplify your map. Done right, this can save on compile times (the time it takes to build your maps) and/or ingame speed (known as fps, frames per second). Done wrong, optimization can actually worsen the way your map plays, or increase compile times. You have to remember optimization doesn't do wonders. Build your maps with the tips in this guide to make sure you don't end up with a map that runs horrible nor can hardly be improved because of wrong planning. Optimization takes time, you shouldn't expect to get an amazing ability to make every map run at 100fps on a geforce 2mx directly after reading through this guide. Be patient, experient yourself, and you should atleast know how to go into the right drection. Some people will never get the hang of optimizing.

If optimization is an University-grade study, then mapping and scripting entities is kindergarten.

Common terms you may not know but will encounter:

- World brush: a brush that isn't tied to any entity
- Skybox: the collection of "tools/skybox" textured brushes that cover the top of your map, giving the appearance of a sky
- 3d-skybox: see above, only instead of an appearing sky projects another part of the map
- Framerate, or FPS (frames per second): how fast your map runs, the higher the smoother, the better!
- Compiling: To turn an editable map into a playable map
- vbsp: The first compiling program, takes care of general BSP structure, and portalling
- vvis: The second compiling program, takes care of visibility, can take hours on an unoptimised map, luckily can also take seconds if optimised right
- vrad: The third and last compiling program, takes care of (the static) lighting your map
- bsp: Binary space partitioning, basically a format to write and use maps (originally ported from Quake)

I will often refer to NODRAW-brushes, CLIP-brushes, HINT-brushes. By them I mean brushes with their corresponding "tools/tools*" texture.

If you find terms or words you do not know, look them up using www.google.com. This is, after all, a university grade guide :P
If you think some important terms are missing then please contact me about it.

All of Valves example maps can be found here:

Your steam drive:\your steam directory\your steam username\sourcesdk_content\hl2\mapsrc

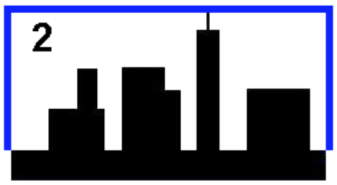
My example map can be found here

Some general notices

First things first, preventing is easier than fixing up later. Here are some tips:

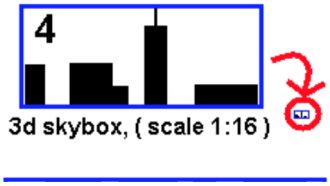
Tip 1: Don't enclose your map in one huge box. Take a look at the image below. This is the skyline of an example map. Black are brushes, blue is the skybox.

1.) This map is pretty optimized. By making sure there is as little "wasted" space as possible you make sure the compile process doesn't waste time and effort on area's the player never gets to. With "wasted" space I mean space which is on the inside of your map, but the player never gets to. Large (high) skyboxes are not good, as the player never gets to them but the compile programs actually do work just in case the player does get to them. What a waste of time! You are far better at deciding where the player can or can't get to.



2.) This map is quite acceptable, though you can understand how situation 1 is much better, since it has the least amount of "wasted" space in it. This situation has one advantage though: Because there is skybox above the lower buildings, the compile tools will understand the player can see over the smaller buildings to see the big one. In situation 1, a player standing to the left of the leftmost building wouldn't be able to see the big building even if that building was twice as high.

3.) This situation is bad. DON'T DO THIS! There is tons of wasted space, and the green area below our level is completely useless. It may even cause the compile programs to think the player can see underneath the level, which causes the entire level to be drawn when that is not needed. Also, the red sides of the brushes are rendered, whereas in the other situations they would have been deleted because they were considered outside the level.



4.) This is the most ideal situation I can think of. The 3d-skybox allows the player to see all the buildings they can, while still making sure the playable area is as small as posslbe, thus giving us the least amount of wasted space. Remember that geometry (=brushes) in the 3d-skybox is much easier to render than geometry in the main level, this is because a 3d-skybox is scaled down to 1/16th of the normal size. This is why this is the best situation. If you have Counterstrike:source, check out de_aztec: You will be able to see how low the actual skybox is: all the roofs you see are actually part of the 3d-skybox. For a tutorial about 3d-skyboxes, check this out. Besides optimizing, 3d-skyboxes are also usefull for making the level appear much bigger than it actually is. If you have a skybox, also make a 3d-skybox, even if it's just for looks. Don't make the 3d-skybox too detailed though, as it gets rendered behind everything.

Don't try to blindly decrease the size of your skybox, players don't like to bump into the walls of your skybox. Even though it looks like the level is much bigger, no player can get through the skybox walls. So make sure the skybox is big enough to accomodate jumping player, thrown grenades, flinging physics objects and bodies flying through the air due to big explosions. Nothing is sadder than a flying body trying to get into orbit but getting stopped by the skybox walls!

Fix leaks

Fixing leaks increases compile times, because the biggest part of the compile process doesn't happen when leaks are present. So don't think leaks are a good thing because of that! Fixing leaks WILL increase map-speed. A tutorial about leaks, what they are and how to fix them, can be found here. If your map has leaks, the main optimizing process, vvis, will not work at all. Vvis is the process that calculates which parts of the level are visible from where, thus allowing the game to know which parts of the level don't need to be rendered at any time. Thus leaks cause more of the map to be rendered than needed, and that sucks.

Don't make brushes overlap

Overlapping brushes are a sign of sloppy work. Your level looks much cleaner without them, and you don't need to worry about compile tools getting a headache figuring out how these brushes should have been made. Ocassionally these brushes are visible ingame, resulting in wierd effects. Unless that is your goal, don't make brushes overlap. There are exceptions offcourse: triggers and other volumetric entities consisting of brushes can overlap each other no problem. As long as the overlapping brushes themselves are not visible, nothing is wrong. However, it is still a good thing to reduce this as much as possible, so you can work easier. Noone can work on a messy desk, nor can anyone with a messy map. Learn to do this and you'll thank yourself later.

Don't make displacements overlap

Displacements aren't cut to size like world-brushes, they are rendered as you make them. If you make a displacement under a house where noone can see it, it still gets rendered. ENTIRELY! Even when the visible part is 1 unit big!
Big displacements that go under walls and other big objects may be rendered more often than you desire!

Don't overdo dynamic lights

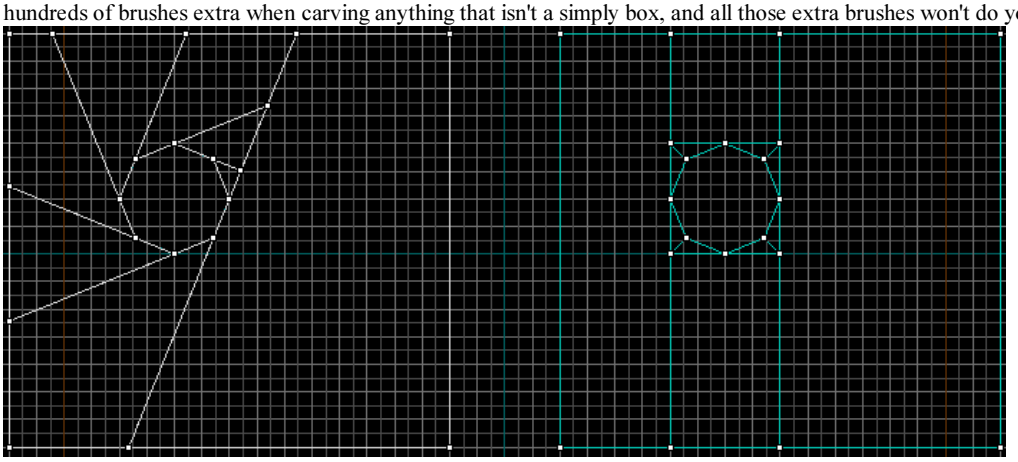
Dynamic lights are lights whose effects are calculated ingame. In HL2 and HL1 all lighting effects are calculated during the compiling process of the map, as opposed to during playing the map. This enables us to use precious CPU and GPU power for other things, like physics and AI, because the game doesn't have to calculate how dark a wall has to be while playing the map. However, sometimes we do want dynamic light effects, like swinging lights. For these situations the light_dynamic exists. This special light isn't pre-calculated, so it requires a lot more cpu/gpu than other kinds of lights. Therefore, use these sparingly. Also note that if you have lights with custom appearances (eg fading or flickering), you will also have a dynamic component: for these lights, only the 100% dark and the 100% lit state will be calculated. The rest of the states (eg 50% lit) will be calculated by the difference between the two pre-calculated states. Though not as hard to render as real dynamic lights, they are still harder to render than normal static lights. Games like FEAR or DOOM III use allmost solely dynamic lights, this is because their engine is much better at handling dynamic lights than source is. This is also why their shadows are so much better than those of source. However they do come at a high price, which is visible when comparing system requirements between source and Doom's engine, for instance.

Make brushes standard sizes

With standard meaning x^2, eg 8,16,32,64,96,128,256,512 units. The more you do that, the more your brushes fit the grid, the less leaks you will get and the less messy your map will look in the editor. It's also better when fitting textures, a texture looks hundreds of times better on a scale of 0,25 than on a scale of 0,26 (though the difference seems minimal check it out ingame and be amazed of the difference in sharpness). I suggest you map on a big grid (eg 16 units) and make it smaller when adding details, and larger again for walls etc. (use [and] to quickly alter the size of the grid).

Dont carve

Yes, I know. Carving brushes is very easy, and gives you results very fast. But if you can, DO IT YOURSELF. Hammer has the tendency to create hundreds of brushes extra when carving anything that isn't a simply box, and all those extra brushes won't do your map any good.

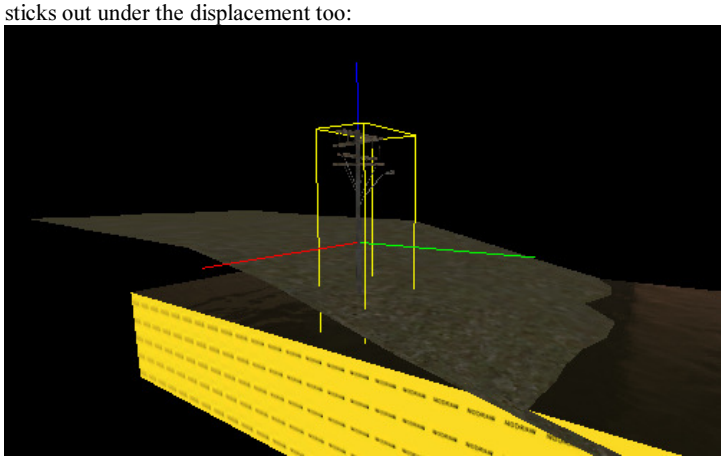


See the difference between Hammer 's carving (left) and mine (right) ? Not only does my version look better, it's also easier to scale, and gives you a less cluttered view, less chance of microleaks between those brushes. The advantages are limitless, so, once again: Don't carve. In the sole occasion where carving is ok, it's faster to do it yourself anyways. Please note that the number of brushes between the left and right situation doesn't always matter: If brush-sides match up nicely the compiler will try to merge the sides. Brush-sides that partly overlap, or have even the smallest of space between them, will not be merged though.

Don't overdo water

Water is very nice in source. But, because of that, also very hard to render. So when placing water, make it as small as possible. In multiplayer maps this means water is usually a no-go, especially in high-action area's, but it's okay in moderation. If you really want water, but find it causes to much lag (makes your level run too slow) there are a few things you can do:

- Make the water cheap. Cheap water looks less good, but is easier to render, thus causes less lag. You can make water cheap by either:
  - putting a water-texture on the water brush with "cheap" in the texture name
  - use a water_lod_control to determine the distance at which the water will transition from expensive to cheap water. This allows you to have good-looking (but expensive) water when the player is near, and slowly fade the water to it's cheaper version as the player get further away.
- Reduce the size of your water brushes. Never let them intersect with world brushes, as the water will still be rendered but not seen, and also be cautious when using displacements, as it's very easy to put the water under the displacement where noone can see it anyways...
- Make sure as few entities as possible touch the water. Any model that touches the water (but isn't completey in it) will be rendered twice: Once normally, and once with special water effects. This doubles the load each model gives. Be even more cautious when models appear to not touch the water, but their bouding box still does because eg you let a part of the water-brush stick out under a displacement and the model sticks out under the displacement too:



Avoid open area's

If your entire level can be seen from a certain point there is not a lot of optimization you can do. Make sure there are actually brushes available which you can use to hide parts of your level behind so they don't need to be rendered. Try to block the player's view (that of vvis to be exact) with buildings or walls to make sure you can actually hide parts where the player isn't looking. (with or without the use of entities and hints) Also, adding lots of corners in straight hallways may enable you to optimize your map better, but it comes at a cost: Will your map still be any fun? Will it still look great? (did it ever?).

Plan ahead

It's always a good idea to start with a layout. With a layout you can determine where the high-action area's are. High-action area's shouldn't be too high in detail (low framerates show best in action moments, one hardly notices low framerates when quietly walking through a hallway), low-action area's should be mainly there to show off your mapping skills.
So make low-action area's high in detail (as high as you can), but keep more room for high-action area's where players meet enemies or physics stuff (like explosions).
You should playtest a lot to find out where these area's are, and offcourse, where you can spare more details.

The NODRAW-texture

This magical texture removes the brush-side which you stick in on. This can be very useful. You apply it as any other texture. You can find it searching for "nodraw" in the texture browser. When you use it on a face, the engine doesn't have to draw that face at all, meaning less load and faster maps. The effect for one face is minimal, but using it massively is a different matter.

NODRAW-brushes can seal the map, that is why you should use them behind entities and displacements to seal the map (you shouldn't use other textures on these brushes as they will be rendered even though you can't see them through the displacements/entities). The nodrawwed brushes are just as solid as with standard textures. So when do you use this magical texture? As much as possible. All brush-sides the player will never see, should have the NODRAW texture. Whenever you make a brush ask yourself: will the player see all brush-sides of this brush? If not, these never seen brush-sides may need to be nodrawwed. I say 'may', because certain non-visible brush-sides will be deleted in the compile process:

- All brush-sides on the outside of the map (given there is no leak)
- All brush-sides completely covered by other world-brush-sides (eg the brush-sides between adjacent walls)

This means that a lot of brush-sides will be deleted anyways, so it's useless to give them the nodraw texture. However, please mind the following

- When a brush-side is completely covered by a prop that doesn't move, nodraw it. The compile process doesn't delete brushes covered by props. Examples are the bits of walls covered by window-props, or walls covered by big combine props.
- The same thing applies to displacements: brush-sides covered by displacements are not deleted in the compile process, so they must be nodrawwed manually.

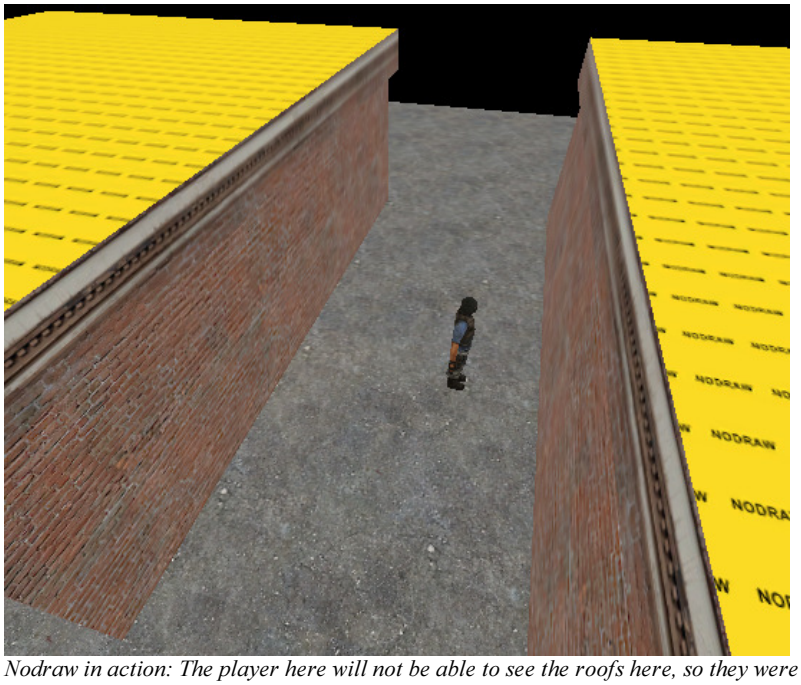
Entities are considered to be movable at all times, so whatever brush is behind them gets rendered normally. If a brush-side is covered by a non-moving entity, including props, displacements, func_detail and normal entities, be sure to nodraw it. Don't nodraw brush-sides behind movable entities, as the void will be visible when the entity does move.

*The void can have two appearances: Looking at it can either give a Hall Of Mirrors (HOM) effect or show you the skybox instead: "the **hall of mirrors (HOM) effect** is an effect in which a computer program attempts to draw an image of what is technically "nothing," and as a result of which, repeats whatever image is near to the null item, causing a shimmer or an endless repetition of an image, similar to the appearance of two mirrors reflecting each other, hence the name." - Wikipedia about the Hall of mirrors effect*

Besides that, don't forget the entities themselves! Especially entities with a lot of detail (like func_details) have lots of brush-sides you can't see but are still rendered. Nodraw them!

The compile process doesn't determine where the player can or cannot get. If the player never gets on the flat roof of a house, or the other side of a wall, nodraw them. Also don't forget the small brushes! If you have a frame around a door that is sticking out of the wall, you may have a tiny bit of brush at the topside the player never sees.

Sometimes, especially if you are new to this practice, it can be quite hard to determine which brushes to nodraw. Usually you forget to do a lot of them, resulting in lower framerates than necessairy. It is a good practice to map with the nodraw-texture as default texture, and only texture the brush-sides with visible textures which you can see when the 3d-camera in the editor is at a place the player can get to. That way it isn't very likely you nodraw to little, but offcourse the problem of forgotten brush-sides allows players to see the void in some cases. Use whatever technique you desire, but most of all playtest! By playtesting you know exactly where the player can or cannot get to, and thus what they can or cannot see, and consequently which brush-sides should and shouldn't be nodrawwed.



Nodraw in action: The player here will not be able to see the roofs here, so they were nodrawwed.

One thing to remember for Counterstrike is that players in spectator-mode can see every part of the level. If you want them to get a nice view of a building, don't nodraw the roofs, as nodrawwed roofs will cause spectators to not see these roofs. So much for a nice view...

Oh, and lastly, i'd like to point out that cutting up brushes to nodraw parts of them usually isn't a good idea. Unless the remaining brush-sides are much smaller/less than the before state, don't bother doing this. Also, some other textures that seem to do the same, "tools/toolsskip" and "tools/toolsinvisible", don't really do the same: the SKIP-texture should only be used on HINT-brushes, and invisible just makes the brush-side invisible, it still gets rendered (I know it sounds odd the game can render something invisibile, but that's the way it is).

Func_detail

The brush-work in your level may get very complicated. For every brush that isn't an entity, the compile process will try to determine what the player can or cannot see when he is looking at this brush. This is okay for big walls, but you certainly don't need to have this done for every brush. Imagine you have a room with 300 small pebbles* (all brushes) spread accross the floor, four straight walls a floor and a roof. If you had to calculate what brushes can or can't be seen from any point in this room, you can be quick: all brushes can be seen at any time. The small pebbles won't cover any whole brushes, so we dont have to bother figuring out whether or not certain brushes don't need to be rendered because they are covered with the pebbles. Go AI. Unfortunately, the compile programs don't have AI and will try to calculate what you can or cannot see using every single pebble in the room. Yes, that is way over 300 calculations where none are required! What a waste of time! What can we do against that? There must be a way to tell the compile programs what brushes to ignore?

** It is never a good idea to make 300 brush-based pebbles, but this is used as an example. Please do not try to make a room with 300 brush-based pebbles, unless you have a death wish.*

Like I said, entities aren't used in the visibility determining process. I will explain why and how in the visleafs chapter. Putting 1 and 1 together we can conclude that by turning all our pebbles into entities, we can tell the compile program to ignore the pebbles! But this is a waste of memory: all options of those entities (inputs, outputs, movements etc) will be loaded into memory. Say we tie our pebbles to a func_door, then the pebbles will use the memory of a func_door. All pebbles will be able to move and act like a func_door. Since the pebbles will never be used as a func_door, this is a waste of memory. So, in essence, we need an entity that is ignored in the visibility determining process, but also doesn't have any special functions so it doesn't use extra memory ingame. This entity listens to the name of func_detail.

A func_detail can be made like any other brush-based entity, simply select a number of brushes and press "tie to entity". Func_detailed brushes act just like world brushes: They are solid, cast shadows etc. There is only a minor bug, which causes light to bounce around a func_detail because the

func_detail doesn't break up the world brushes around it (this effect is called lightbleeding, as the light is 'bleeding' around the brushes. Suffice to say that if you didn't understand that, don't worry about it. You will only see it when you make a large brush (eg wall or floor) func_detail which blocks light/casts shadows.

Lightbleeding is caused by func_details not breaking up world brushes around them. Because of the way the lighting is calculated, light is able to cross the light-blocking func_detail by travelling along the lightmaps of the world brushes around it. If a func_detail is between a very dark and a very light room, then the func_detail on the side of the dark room will appear dark (as it should) but with a light edge around it due to the lightbleeding. You can stop this by increasing the thickness of the func_detail, or by turning it back to a normal world brush.

It's handy to use the func_detail as default brush-based entity in the editor's options. It doesn't matter if you tie 1000 brushes to 1 func_detail or 1000 func_details, but usually it's easiest to group certain brushes into a single func_detail (eg group all brushes making up a window into one func_detail). It only makes editing easier, it doesn't matter ingame. You will probably do this automatically once you use get to use func_detail extensively. Because you will be using it a LOT. Or atleast you should be. Learn all about the main uses of this magical entity below.

Firstly, you must know that for determining what parts of your level can be seen from where, the compile programs (vbsp.exe) divide your level into areas, called 'visleafs'. Then vvis.exe will figure out whether area 'a' can see area 'b', or 'c', or 'd' etc. Consequently, each area doubles the work that has to be done! This is why reducing the amount of visleafs greatly reduces the time it takes to compile your map.

You must also know that these visleafs have the same rules for their shapes as normal brushes. That is, they can't be concave. Dividing a level into visleafs is similar to carving two brushes. What we are actually doing is carving the empty space in your level with ALL the world-brushes you made! Remember what I said about carving?

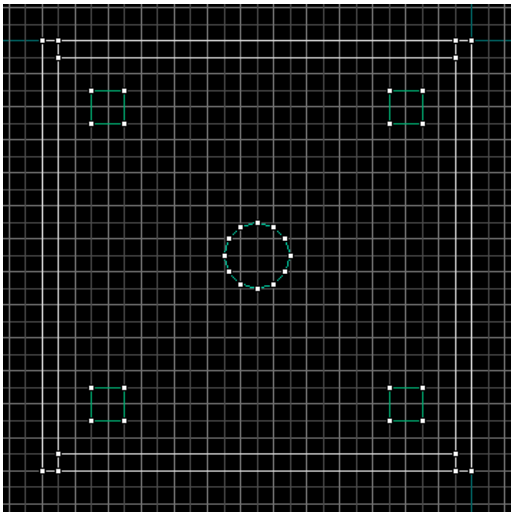
Don't carve. In the sole occasion where carving is ok, it's faster to do it yourself anyways.

And here we go having our level carved all over during the compile process! It's a VERY fast process, even with thousands of brushes it only takes a few seconds, but it does generate tons of visleafs. Ouch. So, what do we do? Simple, we make sure our level is the sole occasion where carving is ok! As I said, carving is usually okay of your brushes are square. So we must make sure our level is as square as possible. Everything in our level that isn't square, must become func_detail, or any other entity if you require their possibilities. Displacements are counted as entities already, so they dont need to be func_detail (in fact, doing so will result in an error). Actually, anything that we don't expect to hide big parts of the level behind them should become func_detail.

I will now discuss some images and try to explain to you what to func_detail and what not to.

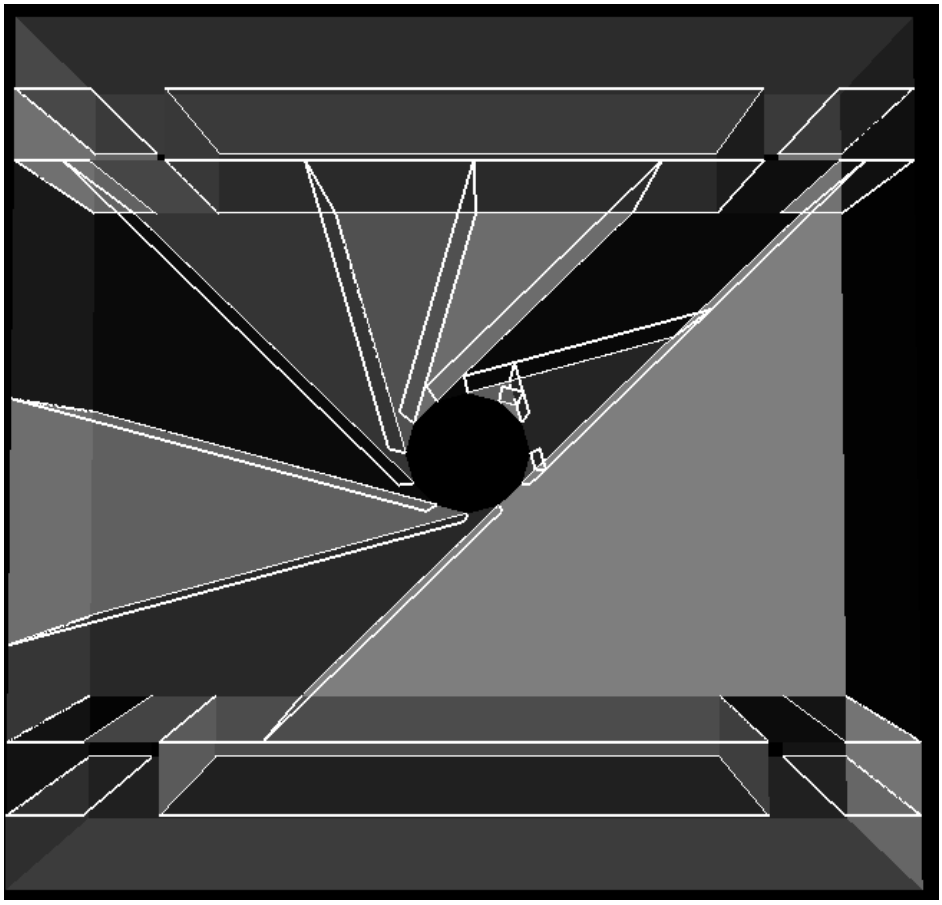
1) pillars

Lets examine this overly simple room with four square pillars and one round one in the center.

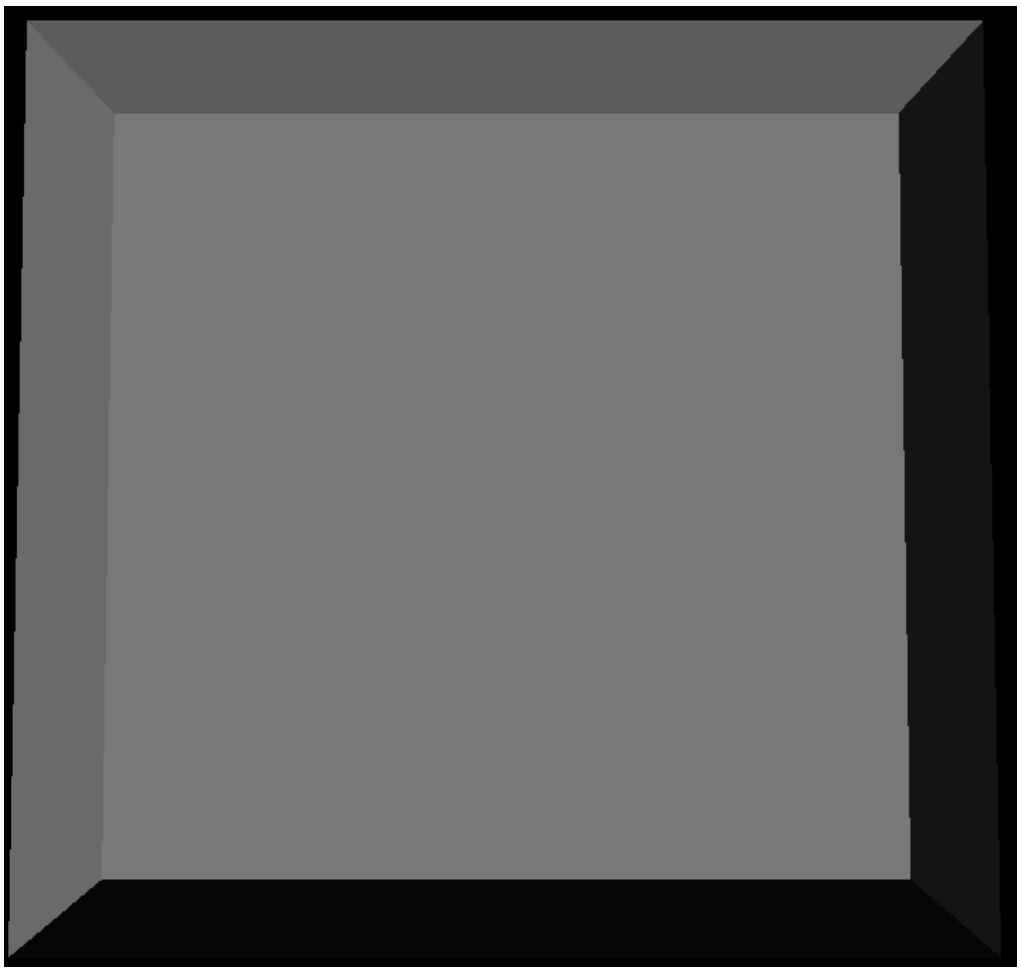


Note that the brushes in the corners overlap: If these brushes are world brushes that doesn't matter, since vbsp will cut the overlapping parts away.

And now compare it to this image (which is a visual representation of how the visleafs are shaped in the level, topside)



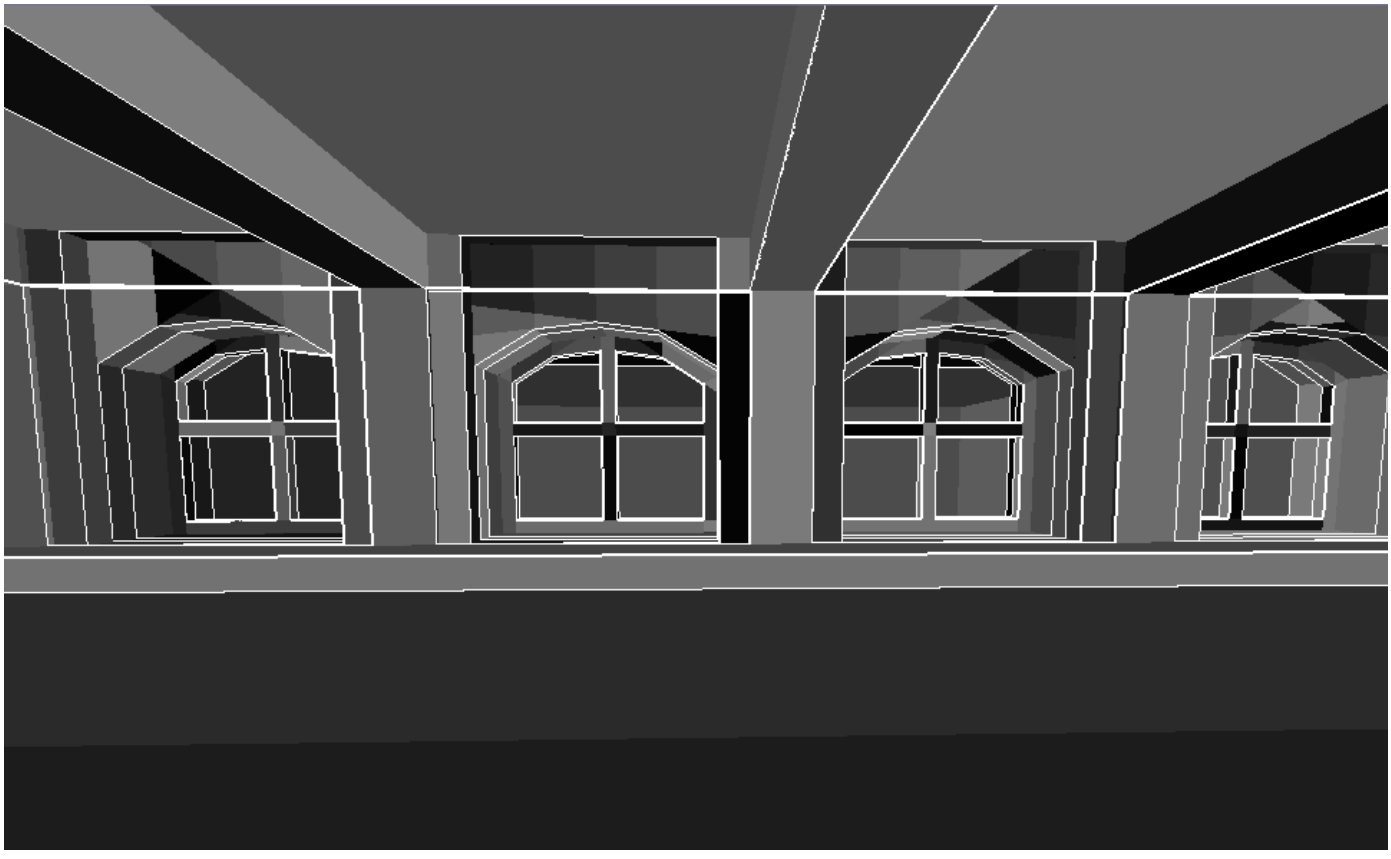
You probably don't even recognise the room, wtf happened? The compile tools tried to carve the pillars into the visleafs, resulting in this mess. There are over 20 visleafs in this simple room! However, since none of the pillars is big enough to actually hide huge parts of this level, we don't even need all those visleafs in the first place! Lets say we func_detail all pillars here, then we would get this image:



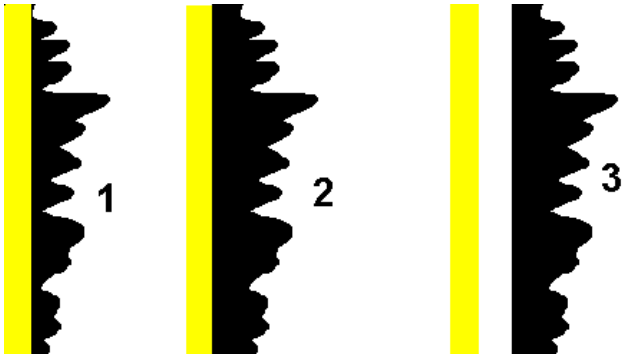
ONE visleaf! What a difference. It allmost makes me cry.

2) A detailed wall

This next wall is pretty detailed, I can't even imagine how horrible this would look in visleafs



Now those are a lot of visleafs. Lets see. This is a wall (no shit Sherlock). Generally, a wall shouldn't become a func_detail because it blocks visibility. But in our case we can safely turn this into a func_detail, since the four windows allow the player to see through the wall. This wall hardly blocks visibility (no more than the pillars in the last example) and so should be turned into a func_detail so the map compiles faster. By doing so we saved roughly 50 visleafs. But what if there weren't any windows? The wall would still cut this corner into tons of visleafs, and we can't turn the wall into a func_detail since it would make the player render the other side of the wall while the player couldn't see that side at all. The solution? A hidden wall! We can still turn this entire wall into a func_detail, but then add a much simpler (and straight) wall inside the func_detail that does the visibility blocking! Because this wall is a single brush, there wouldn't be any more leafs than needed and we still have 100% visibility blocking. Offcourse we can do the same with the outer walls of our level. If we func_detail these walls we would get a leak, but by adding a hidden wall inside these func_details we would stop the leak. Just like we do when placing a hidden wall (or floor) under a displacement. Always make the hidden wall nodraw, and always place the hidden wall inside, or directly behind the func_detail. If you would leave a gap between the func_detail and the hidden wall you would get unused space, which the player can't get to, but it still gets rendered!



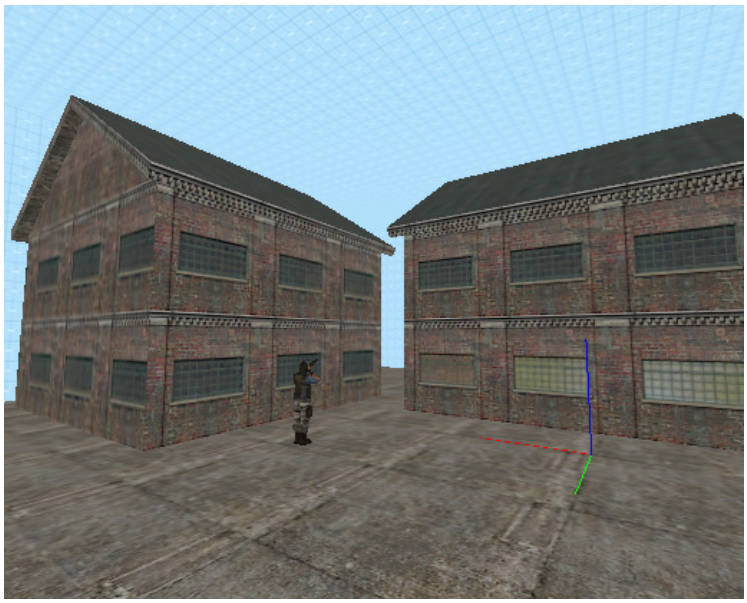
In situation 1 the NODRAW brush (yellow) is inside the detail entity (black). If you were standing at 1, you would see a very detailed wall, but there wouldn't be any extra visleafs because the nodraw brush would be used for making them. In situation 2 the nodraw-brush is directly behind the detail-brush. Though not as optimal as 1, it is still acceptable. In situation 3 however, there is a gap between the nodraw-brush and the detail entity. A player standing at 3 would not only get the nice side of the detail entity rendered, he would also get the backside of the detail brush rendered, not to mention the fact that the other two walls at the side and the floor and ceiling are bigger than they need to be. Therefore, 3 is the worst situation.

It usually is a good idea to place a clip-brush around detailed objects to make sure players don't get stuck in them. Clip-brushes (brushes with the any of the clip-textures, search for 'clip' in the texture browser) block players from going to certain places, for instance tiny nooks and corners in/around detailed objects.

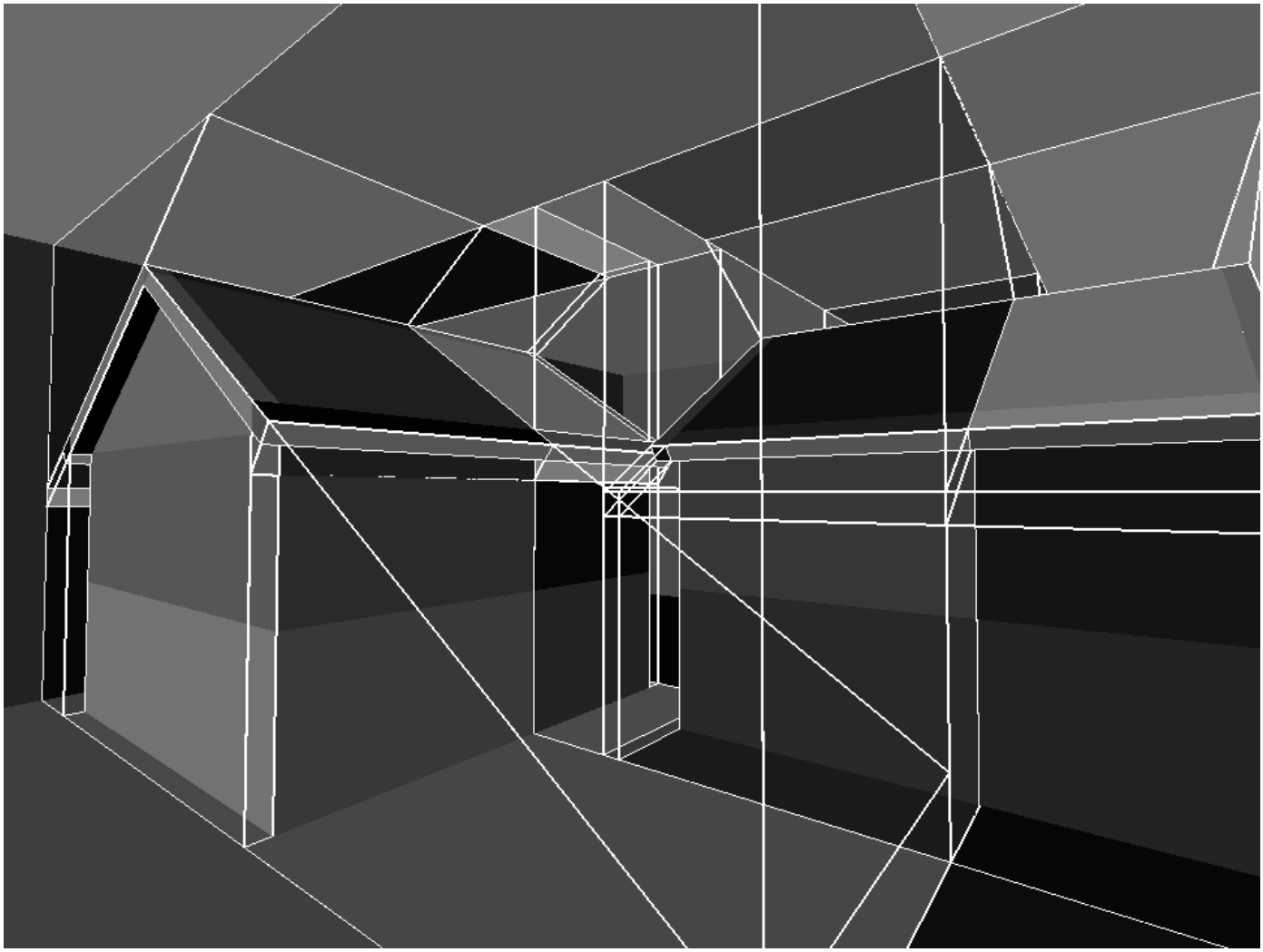
The 'hidden wall' technique is also good when making rounded walls, or roofs. Talking about roofs...

3) roofs

Sometimes you have brushes sticking out, like roofs, which cause extra visleafs. Usually, this is not needed. Take the following example:



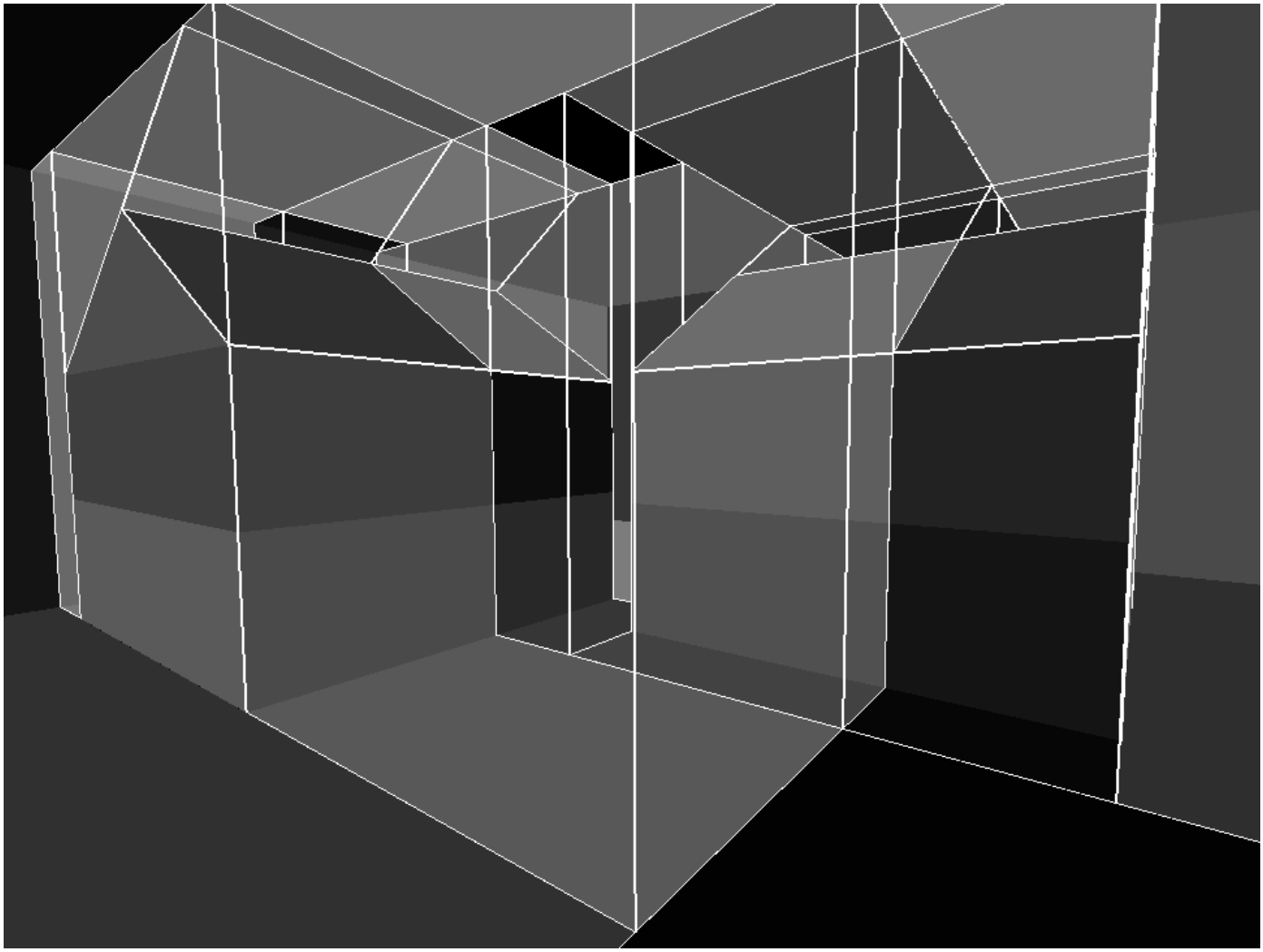
These are two big buildings, with slanted roofs, which stick a bit out. This may seem like a little thing, but lets see what these little things can cause:

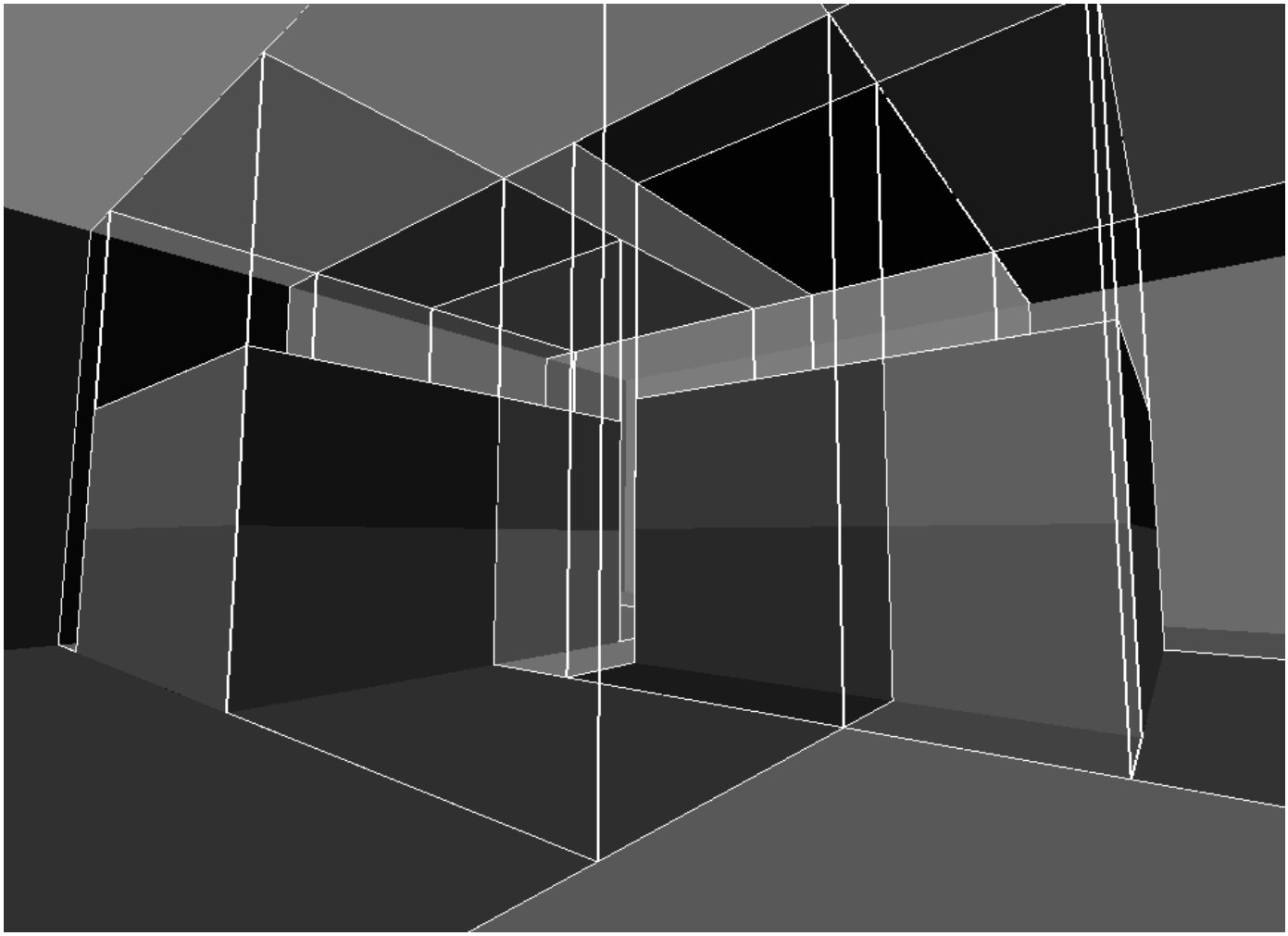


OH, SHIT. This crap is horrible. I'm not even gonna count the number of visleafs these two buildings have caused with their fancy roofs. We could make both buildings func_detail entirely, but that would mean the player can see the high poly objects behind the buildings.

Note that some visleafs in this example are cut because of other reasons, see the chapter about visleafs for that.

Offcourse if we can't make the buildings func_detail entirely, we must make them func_detail partly. Bear in mind that "the squarer the better" and use that to know how far to go. Compare the next two images, one where only the roof ir func_detailed, and the next where the entire 2nd floor is func_detailed. I'm sure you can figure out how the second is the best option.



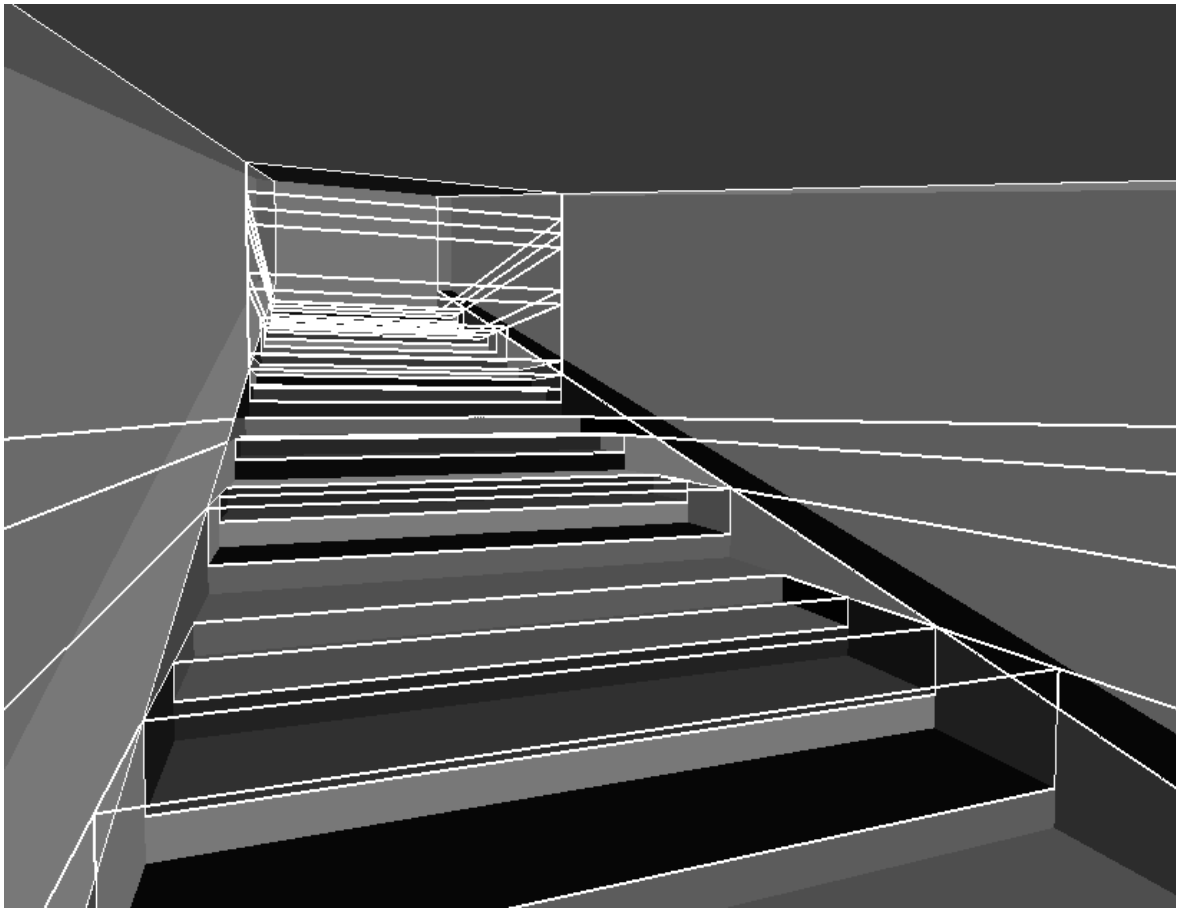


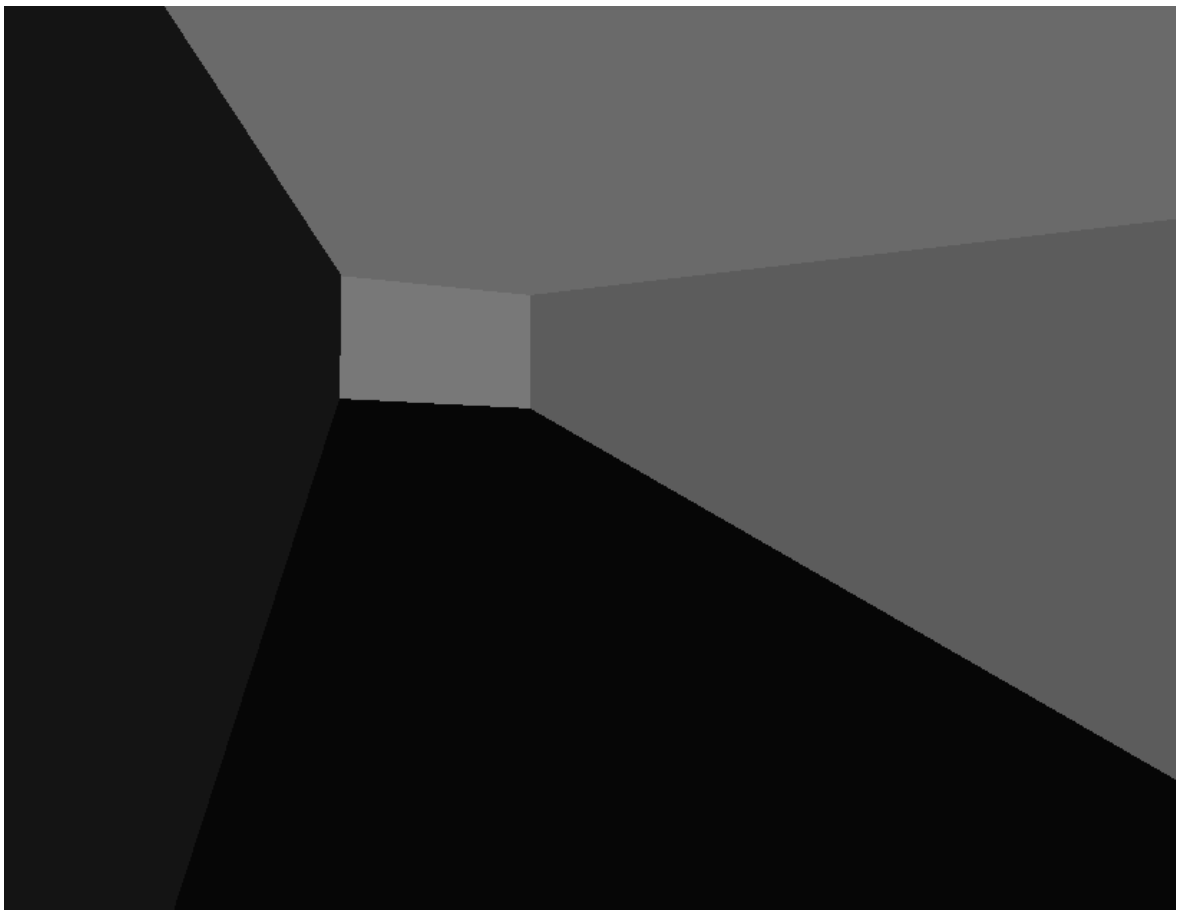
Never forget that these examples don't reflect YOUR map, they are just examples. If a player can go into these houses, there are windows in them, or other buildings where a player can climb upwards, all these rules can still be used to reduce the number of visleafs, but that may not be good for the framerate in your map. You will read more about this in the next two chapters.

If your buildings have things like balconies and/or drains or other brush-based objects hanging on the walls or roof, be sure to func_detail them if they aren't already an entity. Things like these rarely block visibility, and only cause your map to compile longer!

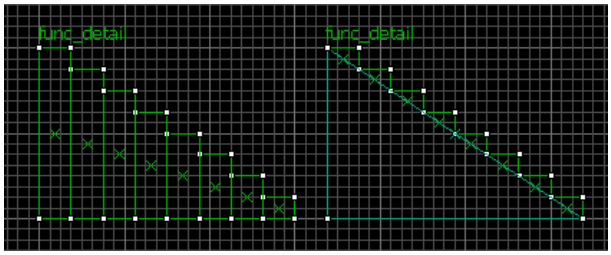
4) stairs

Stairs. Yes. Stairs. They allow you to climb to the next level in both real-life as the amount of visleafs your map has. If you don't make all those steps entities, you will get a visleaf for every step. Please look at the following pictures to understand that. I don't think you will need comments to understand them. The first picture is an editor shot of the stairs, then a picture with all the steps and sides not func_detailed and lastly one where the stairs and sides are func_detailed:





Sideways, this is the right situation of the picture below:



The big triangular brush is a world brush, the small ones are the steps. I could offcourse used the left situation here, but since the walls and ceiling are already going upwards, we might as wel have the fake floor go upwards aswell.

So, in summary:

A (group of) brush(es) should be func_detailed when:

a)If the player stands on one side of the brush(es), he sees a lot more or less than when he is standing on the other side looking in the same direction. (like the pillars)

b)The brush(es) is/are detailed or small (like the details on the wall, or the stair-steps)

c)The brush(es) is/are not square or straight (like all cylinders or arches, or the middle pillar in the pillar example)

d)The brush(es) are at an angle (like the roofs)

e)The brush(es) jut out into a room while no visblocking is required

You should add a hidden nodraw brush inside this func_detail if you still need the visibility blocking properties of the (group of) brushes you just func_detailed. You will learn more about when you need these visibility blocking properties in the next two chapters.

If you ever want to know how your level is divided into visleafs (this is very handy to find out whether or not you have func_detailed enough, and/or you missed some stuff, use Glview. Read more about this program and how to use it here

If you are new to func_detailing, use Glview a lot to find out how your level is divided into visleafs. You won't even believe how much visleafs a single brush can make. And you will never learn either, unless you use Glview (or the ingame alternative) to experiment

The example map made by Valve dealing with func_detail can be found in "sourcesdk_content\hl2\mapsrc\sdk_func_detail.vmf"

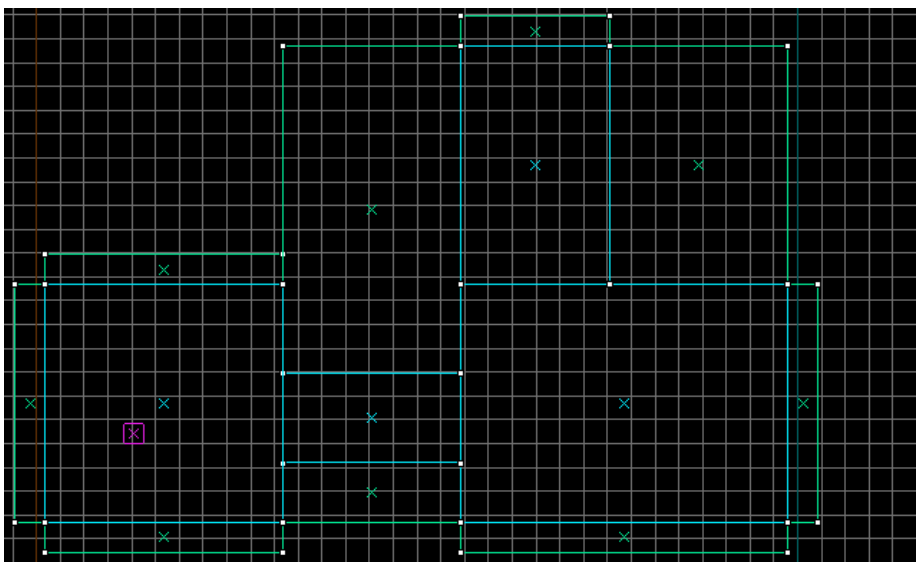
Visleafs

This chapter is all about visleafs, so sit back, relax and enjoy yourself :P

As written earlier, the hl2 map compile-programs divide the map into area's called visleafs to determine what area's of the map can be seen from where, and thus what brushes and entities need to be drawn when a player is at a certain place. The maths is simple: if a line can be drawn from the players view to a certain brush, the brush needs to be drawn. Using real-time calculations would be very heavy, as their would be 1000's of these brushes to test for visibility, and thus does the opposite of what we want (a fast map). Instead, Valve uses a system to pre-calculate visibility. The vvis-process, the second program that compiles the hl2 maps, does nothing more than trying to draw lines to see what parts of the map can be seen from where. You can imagine that because a player's view can be litterally everywhere, this would seem like a near-endless process. To get around that, we will divide the level into parts, and then check what part can see what other parts. If we then know in what part of our map the player currently is, we will be able to find out what parts of the map he certainly cannot see.

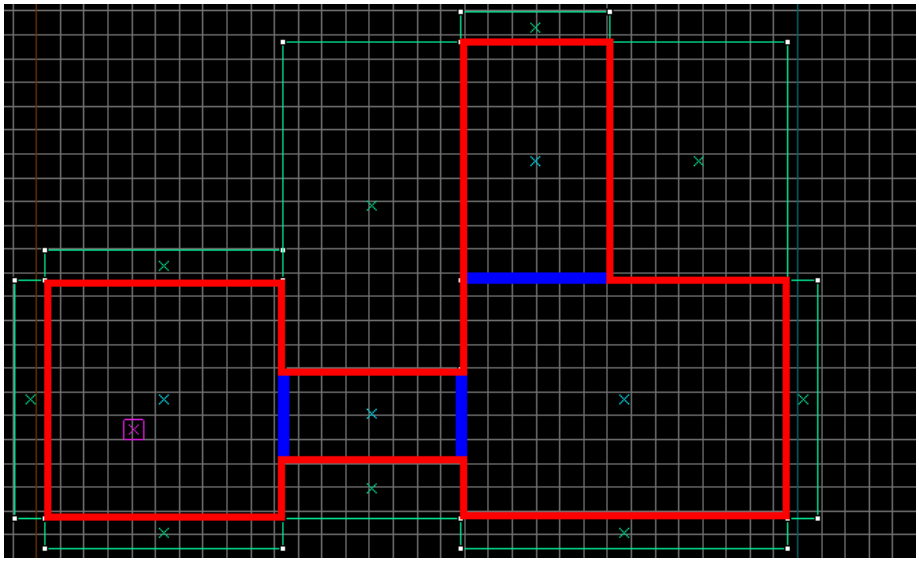
If this still sounds strange to you, here's a step-by-step explanation:

Our test level:



Step 1. Divide the level into area's

To divide the level into chunks, we simply ignore all entities and displacements, and start carving the playing space into convex area's:



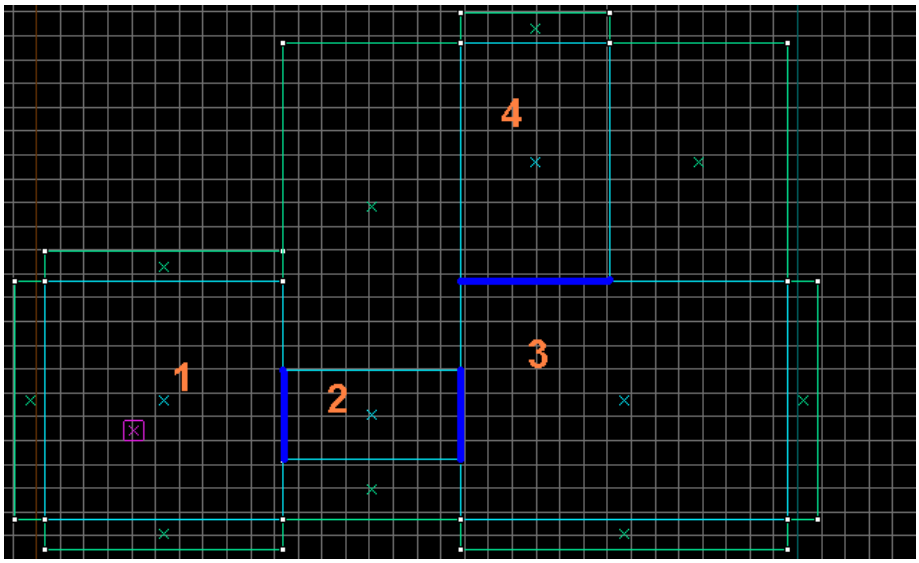
This step is done by vbsp, the first of the compile programs. The process is called "portalling", as the planes where two visleafs touch is called a 'portal'. The result is outputted to a portalfile ([mapname].prt), which is picked up by vvis, the second program of the compile process. In red marked are the visleafs, blue lines denote portals. As you may notice during compiling, portalling is very fast, usually no more than a few seconds for a huge level.

The reason why entities and displacements are skipped is that entities are considered to be dynamic (able to move) and we simply can't make dynamic visleafs if we want to be able to precalculate their visibility. Displacements would create thousands of visleafs on their own, which is why they are ignored aswell (They were originally also meant to be able to move).

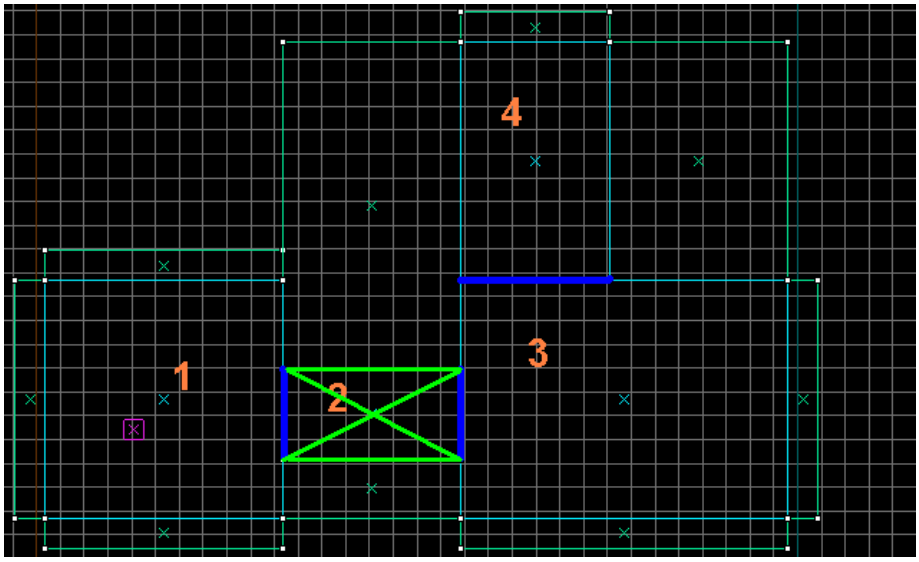
Step 2. Making the visibility table

Vvis will now draw straight lines from each corner of each portal to all other portals in the map. If all lines between two portals cross brush-geometry, the portals are considered to be invisible to each other. If one of the lines can be drawn without crossing brushes, the portals can see each other. If that is the case, both visleafs (one on each side of the portal) are able to see the other pair of visleafs (one on each side of the other portal).

We take the portals:

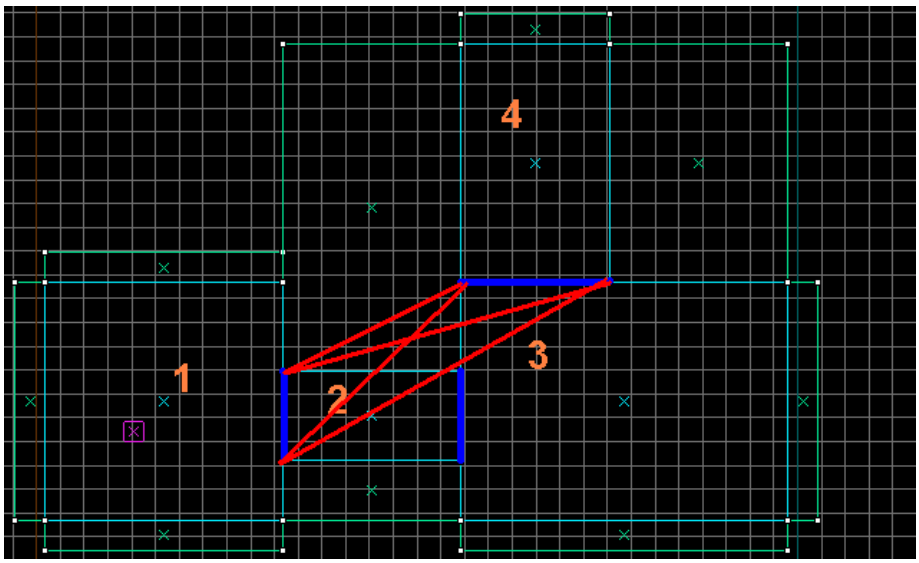


To see if visleaf 1 can see visleaf 2 we don't have to do anything, as both share a portal (they touch each other directly).
To see if visleaf 1 can see visleaf 3 we will try to connect one of visleaf 1's portals to one of visleaf 3's portals:



As you can see, this is possible, thus visleaf 1 can see visleaf 3.

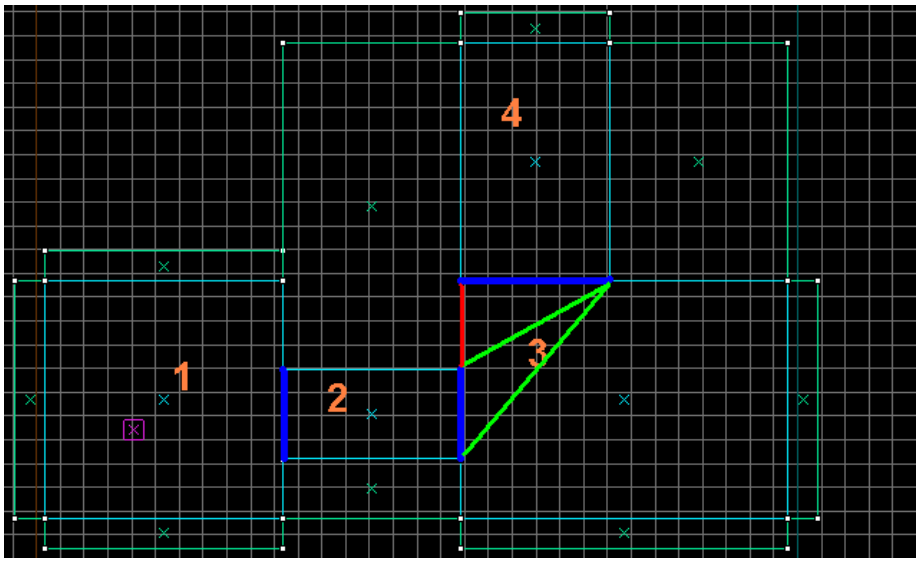
To see if visleaf 1 can see visleaf 4 we do the same again:



As you can see this isn't possible, all four lines we can make between these portals goes through brush-geometry. Visleaf 1 cannot see visleaf 4.

In short, visleaf 1 can see visleaf 2 and 3, but not 4.

Visleaf 2 can see visleafs 1 and 3 (because they directly touch each other), but what about visleaf 4? We know the portal between 1 and 2 cannot see the portal between 3 and 4, but can the portal between 2 and 3 see the portal between 3 and 4? The answer?



Yes! Visleaf 2 can see all other visleafs in this level!

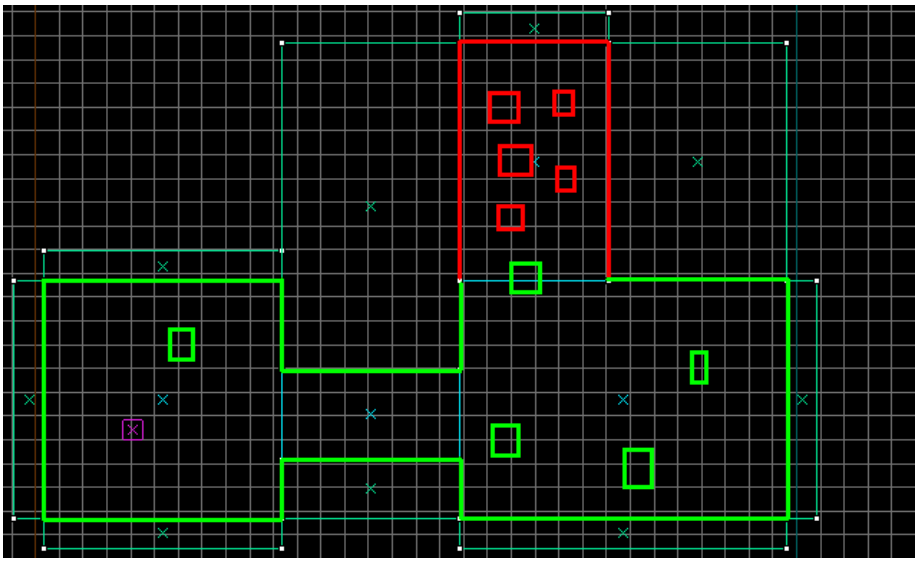
Our visibility table now looks like this:

- 1 -> 2 visible
- 1 -> 3 visible
- 1 -> 4 not visible
- 2 -> 3 visible
- 2 -> 4 visible
- 3 -> 4 visible

Because visibility is reciprocal, we also know that 4 cannot see 1 etc.
The amount of calculations needed depends not on the amount of visleafs, but on the amount of portals. as you can see, three portals take 3 calculations. With each portal the amount of calculations is increased by the total amount of already existing portals. 1001 portals thus take 1000 calculations more than 1000 portals!

Step 3: using the visibility table

Now that we know what visleafs can see which ones, we can make out what to draw when a player is at a certain visleaf. This is done ingame, realtime. For instance, if the player is in visleaf 1, we don't have to draw visleaf 4:



(green is drawn, red is hidden, hollow boxes are entities).

Note: Brushes only exist in the editor. Ingame, only the brush-sides that make up the brushes exist, but they don't belong to each other anymore. This is why in the above example brush-sides can be hidden while other brush-sides of the same brush are visible.

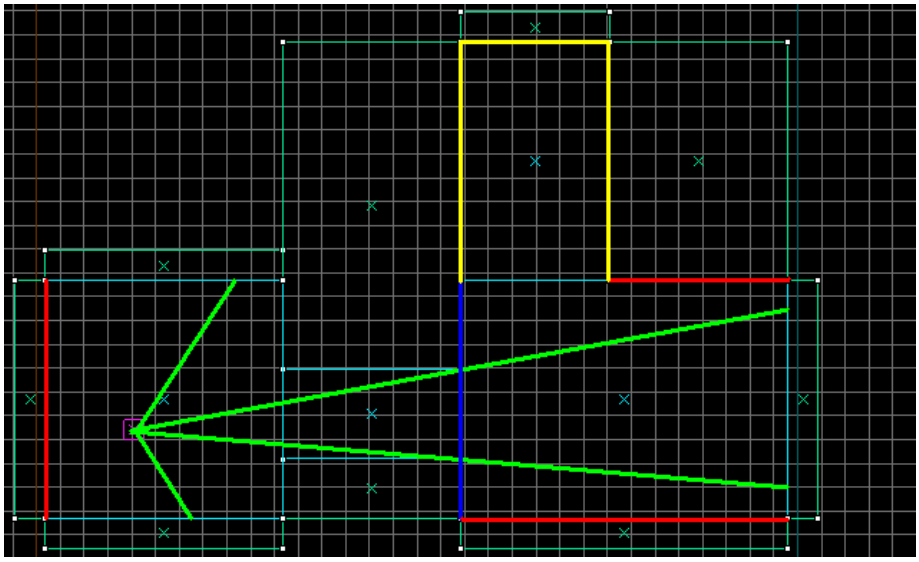
I hope you can imagine why reducing visleafs can increase compile times (as we need to connect less portals), but can also decrease the framerate of our level. Can you imagine what would happen if the brush north of visleaf 2 was an entity (pretending this wont cause a leak)? In that case the entire level would be drawn, even the parts that are red now, even though the player wouldn't be able to see through the wall... This is a serious waste of power you can use for extra frames per second, or more details in your level! Less visleafs mean a slower map because there is less to hide. Big visleafs are harder to hide than small ones, just as an elephant is easier to spot in the woods than an ant. You some how have to make a compromise: long compile times and a faster map, or shorter compile times and a more lagging map? This isnt very easy, especially since extra visleafs dont HAVE to increase map speed, so you may be compiling useless data, which will in turn make your map slower.

You have to determine when visleafs are uselessly small, and when they are uselessly big. This is not easy. In the next chapter i will tell you how to change the shapes of visleafs, and then a few hints on when and where to do so.

Discussion:

Some things you don't really need to know, but can be really usefull:

- What is rendered also depends on the viewcone of the player. If the player in our level would look to the west, nothing would be rendered behind him (half of visleafs 1, and all of visleafs 2,3 and 4 wouldn't be rendered
- The mapsystem Valve uses (BSP, Binary space partition), also hides brush-sides that aren't directed to the player. For instance, in the example level below the blue faces will never be drawn if the player is east of them:



The red brush-sides aren't rendered because they aren't in the player's viewcone (=green, looking eastwards), yellow isn't drawn because that visleaf is considered to be hidden from the player, and blue isn't rendered because the player is looking at those brush-sides 'from behind' (I'm sure you can imagine the 'front' of these faces being the side facing the west).

Useless info:

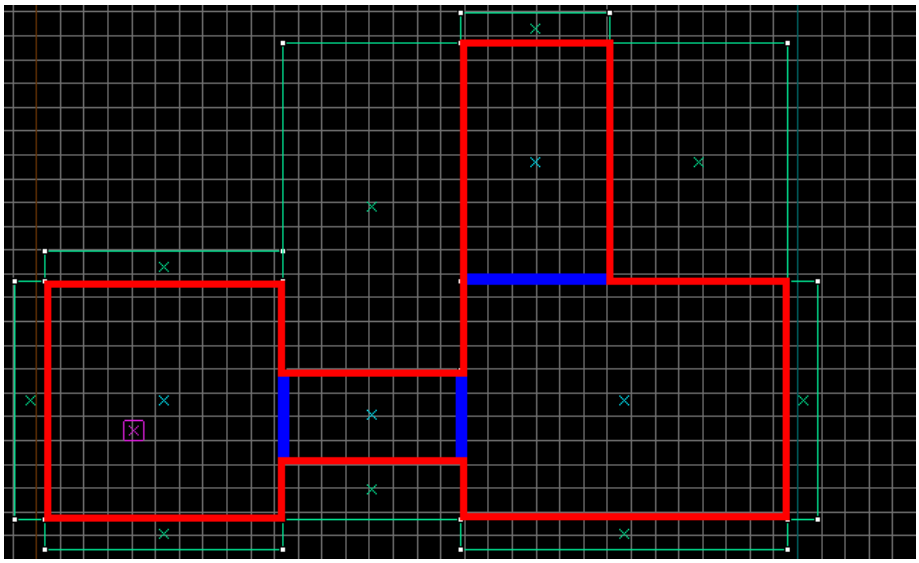
For determining visibility, the vvis process actually uses a third portal (this portal being a portal that could occlude both checked portals, and is acually the portal between a leaf in the playing space and a leaf with CONTENTS_SOLID. What you have read untill now is just a simplification, it gets much more complicated. Because this is just interesting, but not necessairy to know, I will just give you a link to learn more about this concept:

http://qxx.planetquake.gamespy.com/bsp/

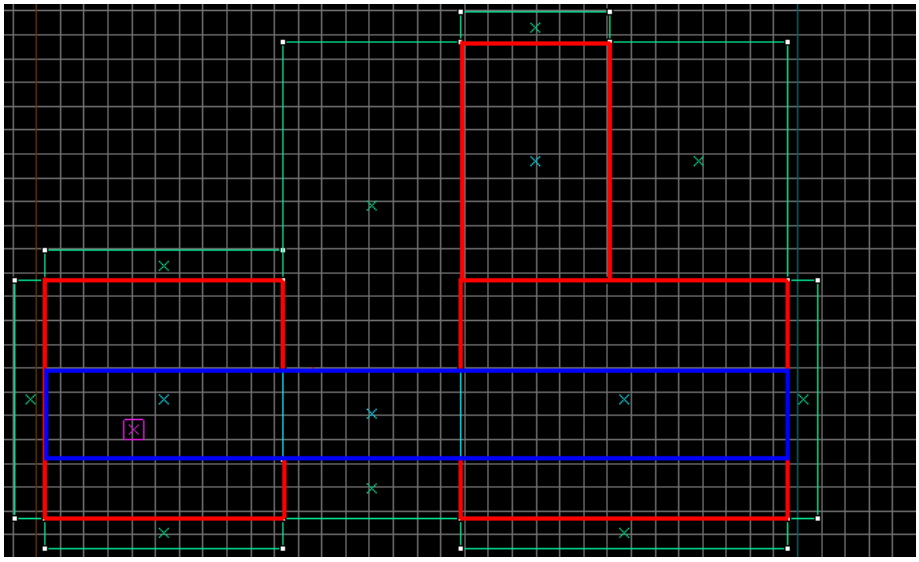
As you can see this links to a site for Quake, which is the mother of BSP (Valve copied the BSP system from quake), and therefore partly of the hl2 engine. Like I already said you shouldn't worry about this technique, the simplified version I explained more than enables you in your optimization process.

Hints

The way your level is cut up into visleafs affects the speed of the map a lot. Let's look back at the example from the previous chapter.



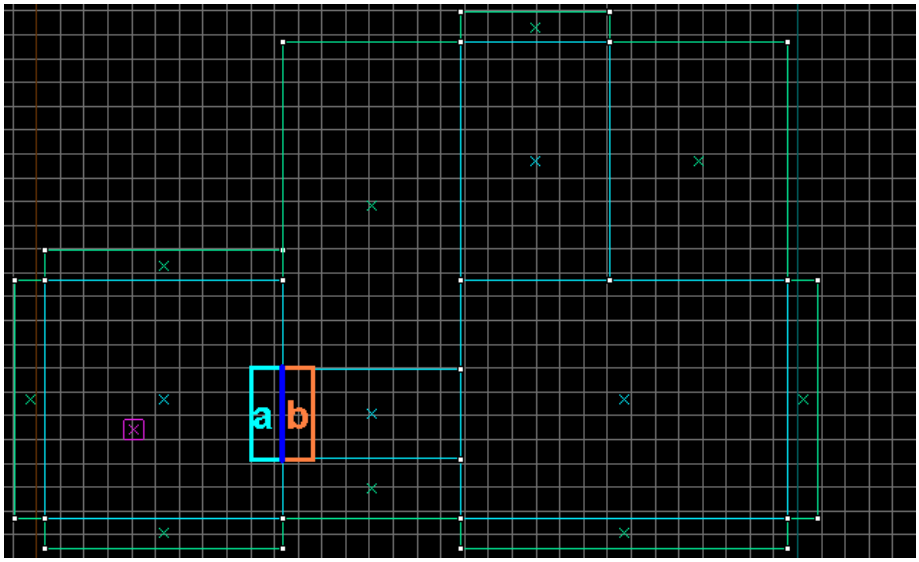
What if the level was cut up not like above, but like below?



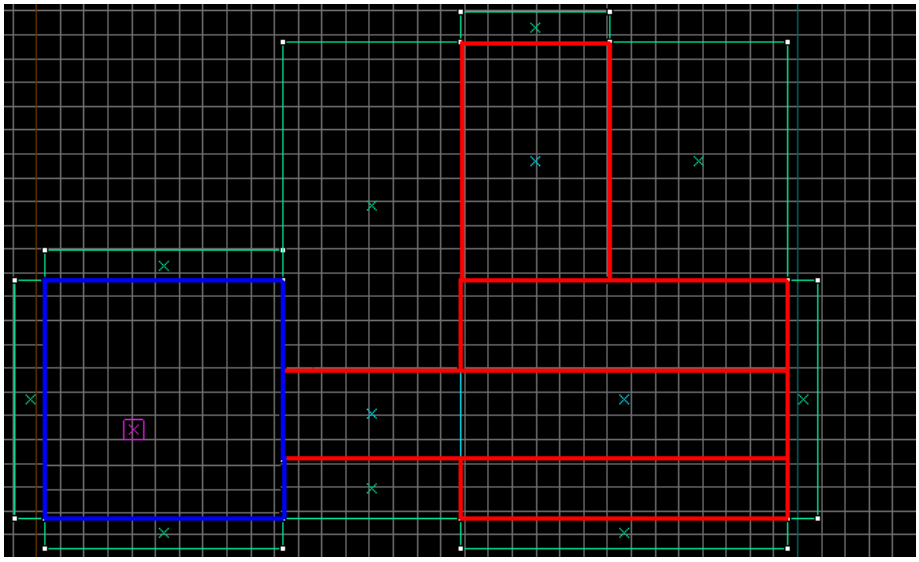
In this case, the visleaf the player is in (blue) can see all other leafs in the level. Thus, the entire level will be drawn when the player is standing here, as opposed to the situation above where the top visleaf would be hidden. This shows how the shape of visleafs can affect the framerate of your map, even when the brushwork in the map is completely the same!

There must be a way to control how your level is divided into visleafs, after all that's the purpose of this chapter. Hinting is a way to tell the compile-programs where YOU think THEY should cut visleafs. Offcourse they still cut up visleafs where they think it's needed, but they will always cut up visleafs where you say they should (they are very obedient dogs, arf). Making Hints is quite easy. All you have to know is that visleafs are cut whereever the "tools/toolshint"-texture is placed. This brush MUST have the "tools/toolsskip"-texture on all sides which aren't used for hinting. Therefore, hint-brushes can only have two textures: the hint and the skip texture. No more.

In our case we can either make brush A or B:



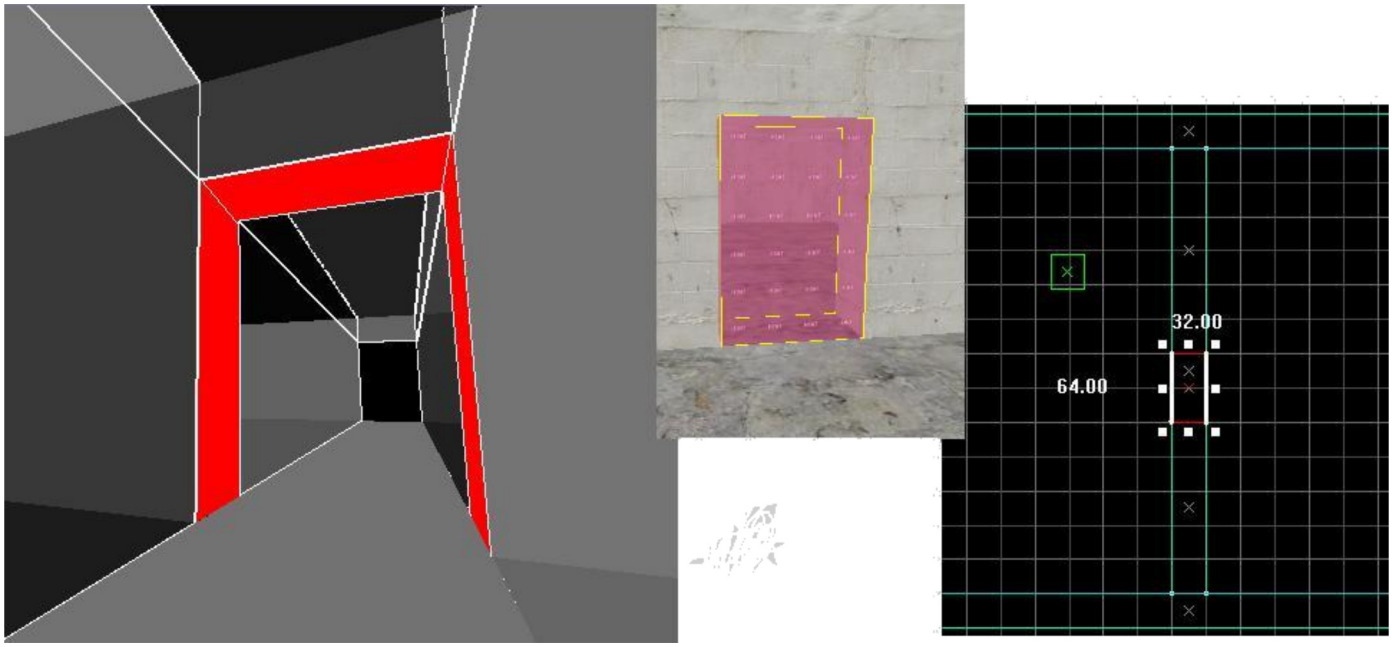
Where lightblue and brown are the brush-sides with the tools/toolsskip texture, and the blue line is the brush-side with the "tools/toolshint"-texture. Whether you make brush A or B, or a brush one hundred times as big doesn't matter, as long as the side with the "tools/toolshint"-texture is at the blue line we get our effect:



Not only does the visleaf span two rooms, we also optimised the map by making less visleafs! Because the compile programs found that the left room was already square, they did not see a reason to cut it up into multiple visleafs. Creating a hint-brush on the entrace of the other room would decrease the map by another two visleafs! Not much, but if you do this everywhere in your map, you are bound to get a faster compiling map. You can also take a look at the example map made by Valve in: "sourcesdk_content\hl2\mapsrc\sdk_hints.vmf" for more information on how to make these hint-brushes.

Hint brushes can be used to make sure certain visleafs don't see each other, but also to reduce visleafs

Most of the time vbsp (the program that cuts up the map into visleafs) makes the visleafs in a logical way, but not always. Doorways and windows are bound to cut up the room into visleafs:



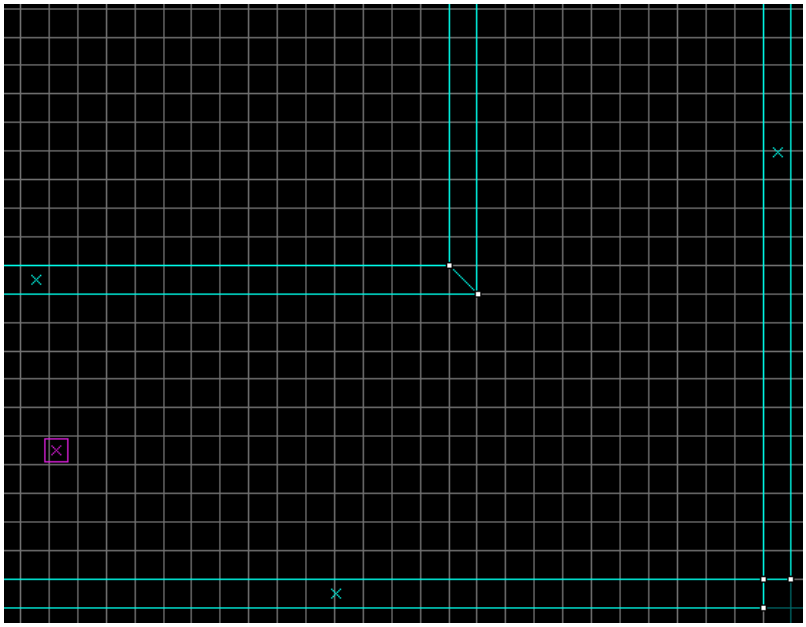
Sometimes this WILL improve mapspeed, eg when there is some heavy brushwork on the other side of the wall and the player isn't looking straight at the doorway, but in most cases it is just useless. So it's best to fit a hint-brush inside the doorway, OR make the entire wall (if possible) an entity. Offcourse you must remember when placing hints to make sure you don't create more visleafs than you need. Make sure your hints are as flush with the walls as possible, fit tightly (NO GAPS) and as straight as possible. If I ever catch you making a hint-brush out of a cylinder I swear I an going to kill you.

Ok, you now know what hints do, and you should know what to do with them: making sure certain visleafs don't see each other. But you don't know how? Hinting isn't the easiest thing in the world, hinting requires every braincell in your body (including the ones in your arse, haha). I will show you some examples on how and where to use hinting, but don't expect to become an expert right away. Hinting will mostly be trial-and-error, especially in the beginning. Play the map, find an area that isn't running smoothly, find out what causes it, place a hint-brush to stop the cause, compile again, play again, repeat. Finding out what the cause of the lag will be discussed in the checking progress chapter.

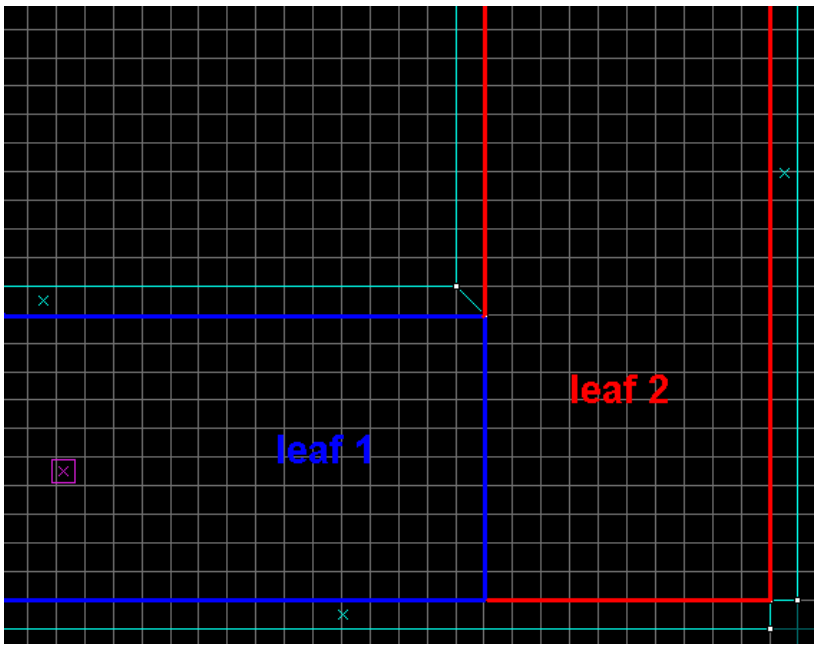
The essence of hinting is to make sure visleafs can't see around corners, over or under walls or buildings, or through openings in walls (eg doorways, windows).

1) not seeing around the corner

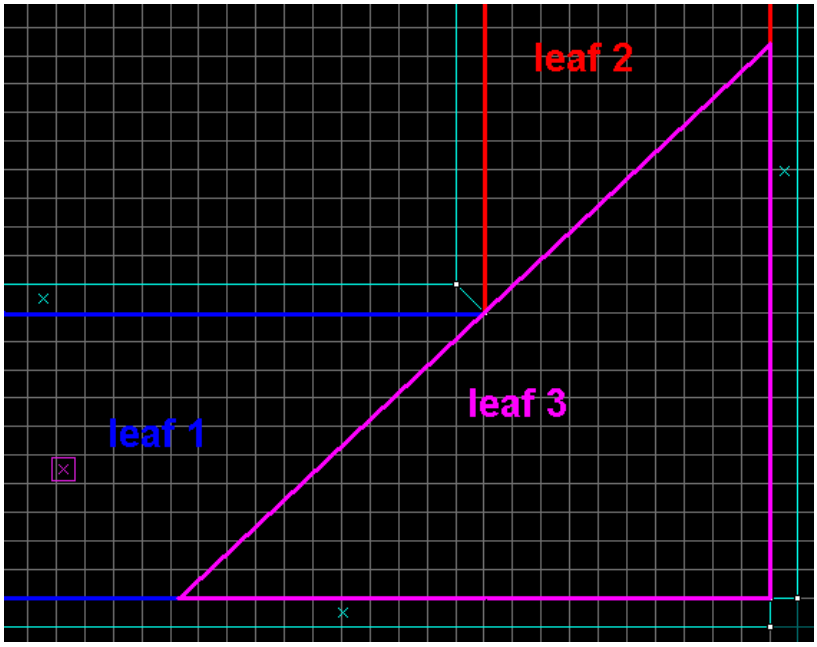
When dealing with corners, you have to imagine this example:



In the current location, the player will not be able to see whatever is around the corner. However, it will still get rendered, as this image will show you:

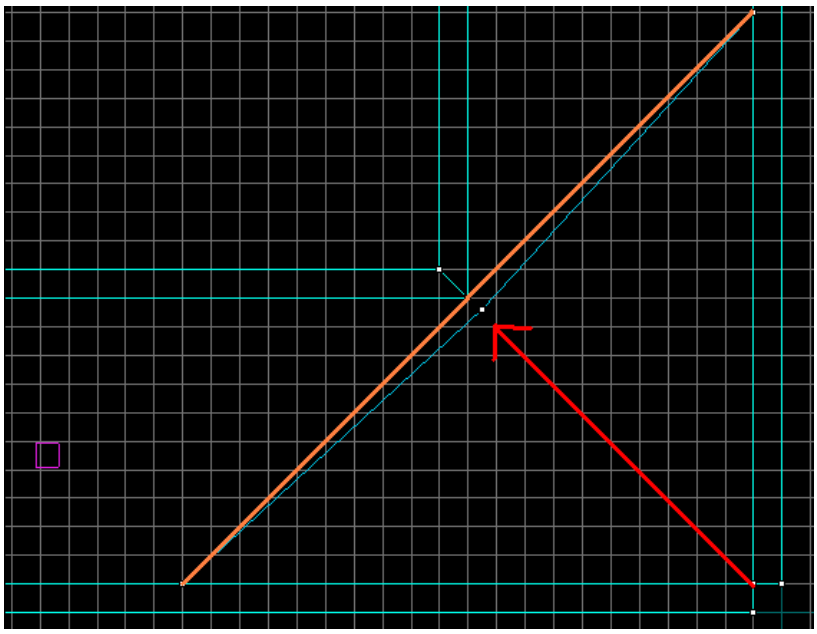


Because the visleaf the player is in (no 1) is able to see visleaf 2 (they touch each other) everything in the hallway around the corner will be rendered. To stop this, we need to make sure no straight line can be drawn from visleaf 1 to visleaf 2



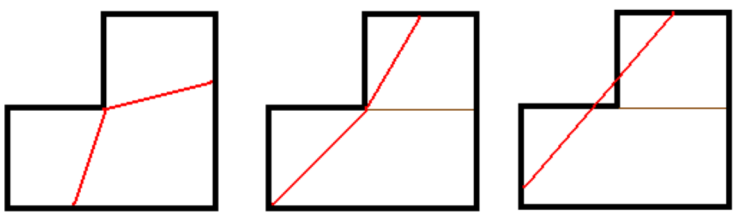
I made a diagonal hint-brush in the corner, touching the inner corner of this hallway. By doing so I did have to make an extra visleaf, but this one is for the better: when the player is in visleaf 1, only visleaf 1 and 3 will be drawn. Visleaf 2 will be invisible to visleaf 1, thus it won't be rendered at all. The player will be able to see everything when he is in visleaf 3 though, so no visleafs will be hidden in that case (in this example).

The angle of the diagonal HINT-brush doesn't matter. As long as it is one brush, both outer visleafs can't see each other. The shape of the hint-brush is equal to visleaf 3 (with only the diagonal side with the HINT-texture), but offcourse its perfectly legal to drag the bottom-right corner more to the middle so you can navigate easier through your level in the editor:



You should be able to know why:

- **the hint brush has to touch the inner corner:** if it didn't, visleaf 1 and 3 would touch each other and the effect we wanted lost.
- **the hint brush is one brush:** I could use two or more brushes, but then I would need to worry about the angle of both brushes: If their angle is less than 180 degrees, visleafs 1 and 3 would see each other again
- **the hint-brush doesn't cross the corner:** If it did, we would only reduce the effect slightly, but also create more visleafs. It's also a lot less neat.

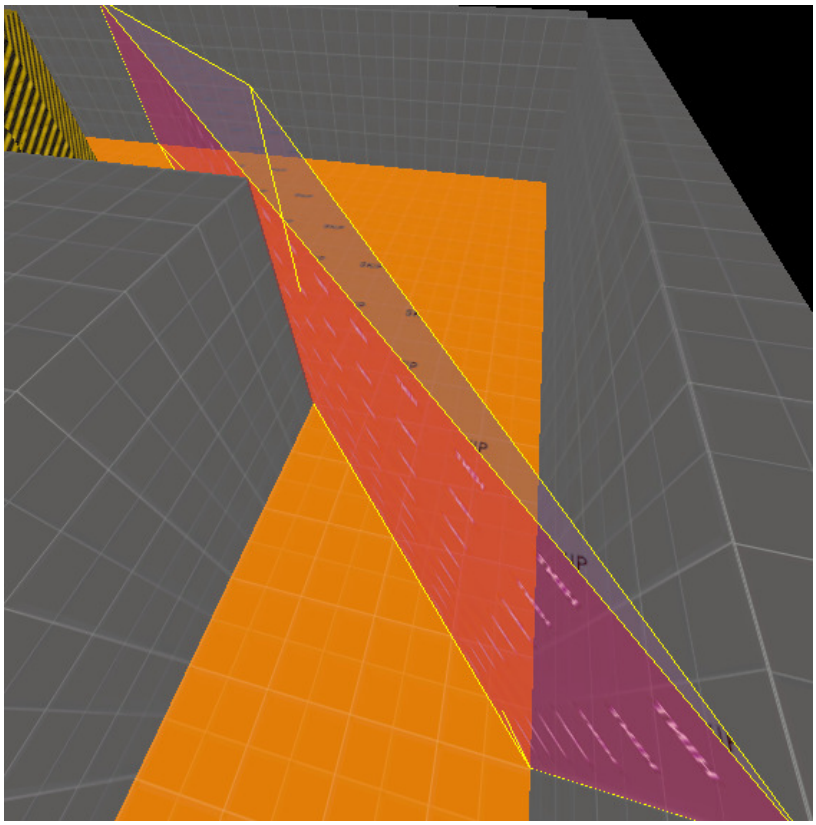


Corner < 180 degrees
Visleafs 1 and 3 see each

Corner > 180 degrees
Visleafs 1 and 3 can't see

Hint brush goes through the corner

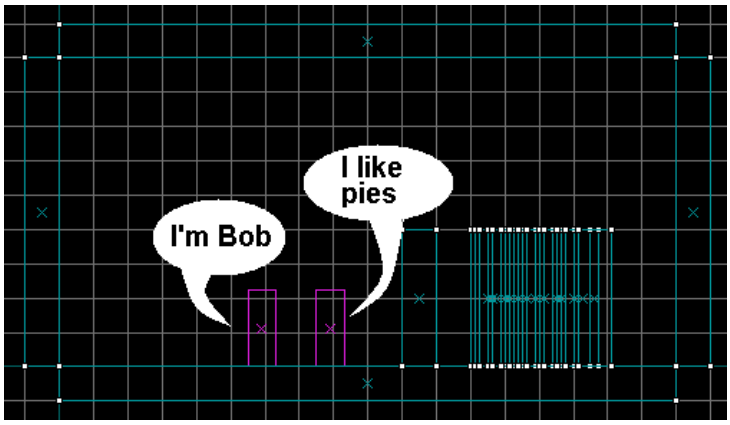
other, a straight line can be drawn between them	each other, no straight line can be drawn between them, but the effect is less than above because visleaf 3 is bigger (and split)	This causes the same as the situation in the middle, only much worse!



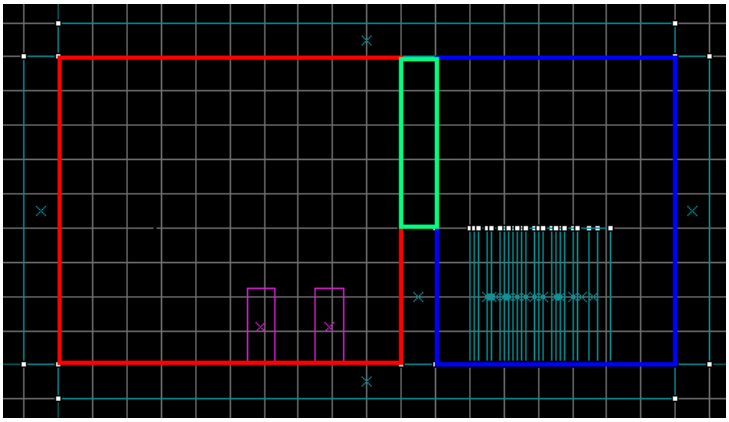
A HINT-brush in Hammer

2) Not seeing over or under walls/buildings/other objects

Normally players can't see through walls, so you also don't want the game to render what is on the other side of a wall. Usually this isn't done anyway, but occasionally it might. If that is the case, remember Weebl and Bob!

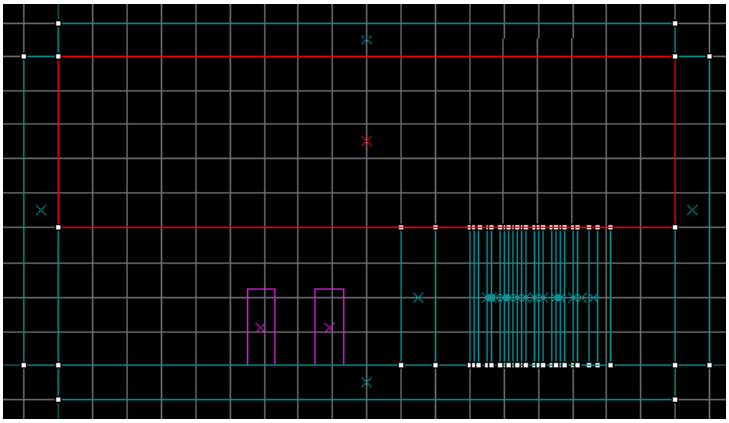


Weebl and Bob are two eggs (Their site) but also players in this mini-level. This mini-level features walls, a ceiling, a floor and ugly pipes. Luckily the mapper put a wall between Weeble, Bob and the pipes so Weeble and Bob don't have to see the pipes. Even though that's nice and all, Weeble and Bob still have those ugly pipes rendered. The reason? Vbsp did a pis-poor job in dividing this level into visleafs:

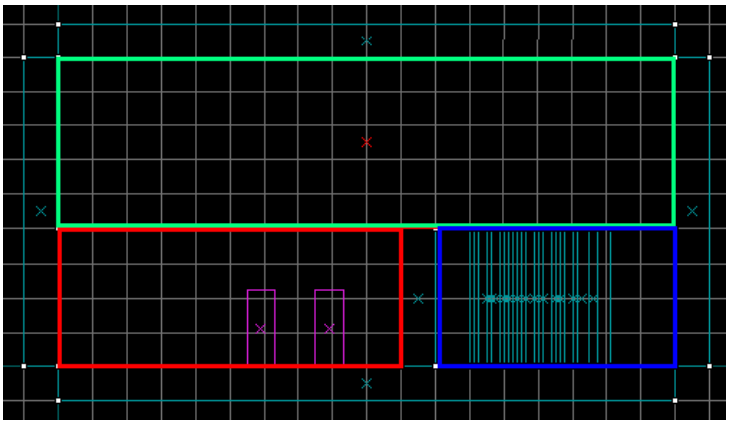


See? The visleaf Weeble and Bob are in (red) can see the visleaf with the pipes (blue). That's therrible? What will we do?

The solution is to place a horizontal hint-brush at the top of the wall, like so:



The selected brush is a hint-brush, with the bottom brush-side covered in HINT-texture. Because of this hint-brush, the new visleafs would look like this:

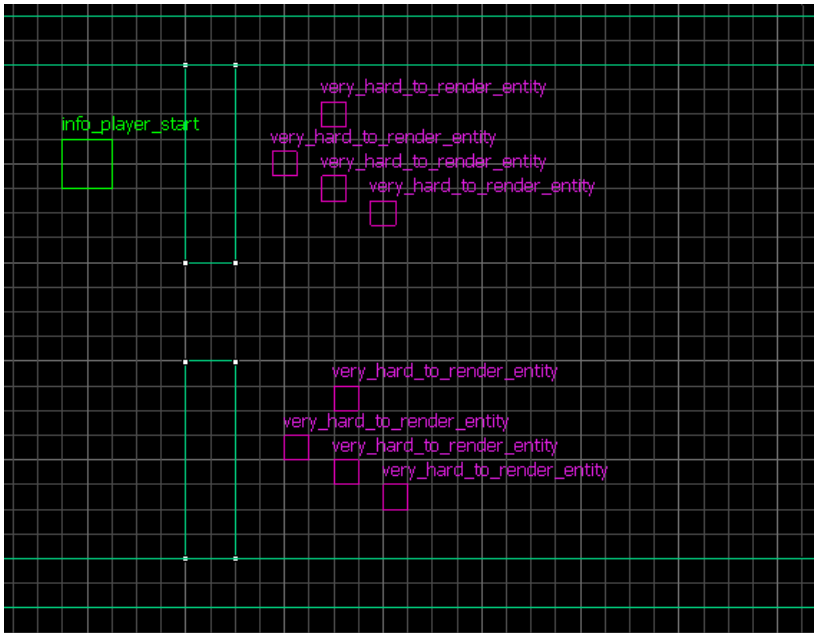


In this case, Weeble and Bob's visleaf can't see the visleaf with the pipes, so no pipes are drawn! Weeble and Bob are happy now. They wont be happy if they jumped or climb too high though: If they would do that, they would get into the green visleaf, which can see the blue visleaf with the pipes. So we must make sure they can't climb or jump that high, OR make the wall higher, or simply don't care if Weeble or Bob can see the pipes.

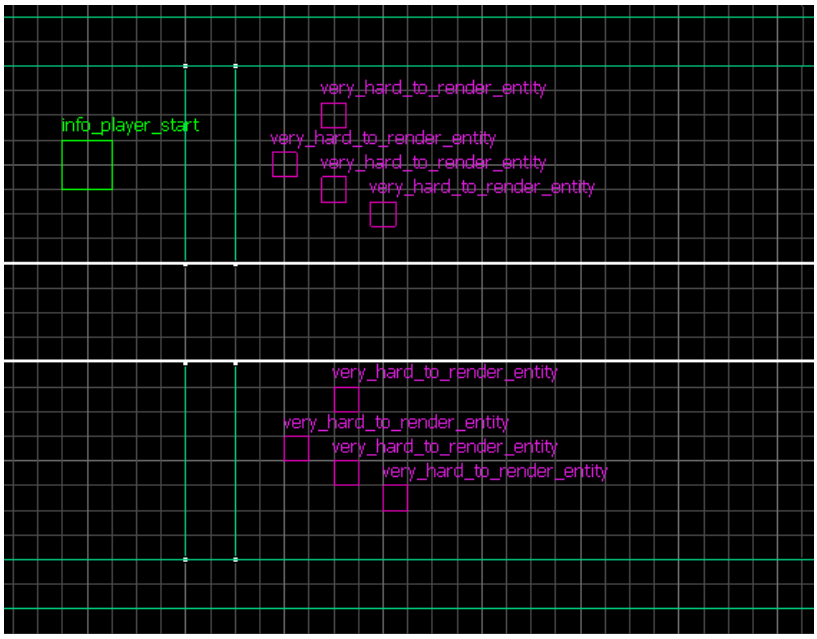
The same thing applies to houses in a village for instance. If your map has buildings which don't reach to the top of the skybox, add a horizontal HINT-brush at the top of the houses (try to find the lowest possible height, where no player will be able to get up to) to make sure nothing is rendered at the other side of the houses because the visleaf the player was in could see over the houses. Alternatively, make the skybox lower and put the top parts of the buildings in your town in a 3d skybox.

3) Not seeing through openings in a wall

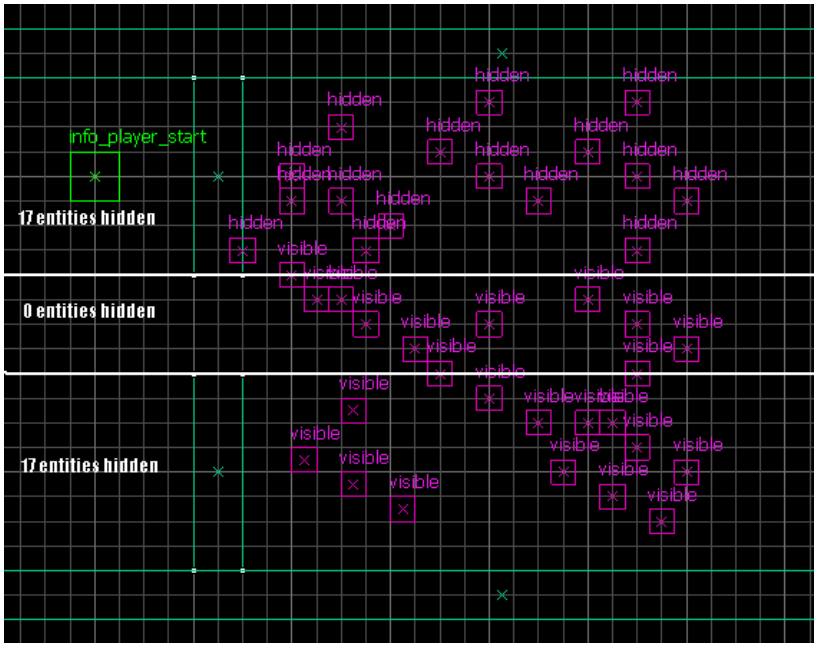
Offcourse you want the player to be able to look through holes in a wall, but what if the player isn't standing right in front of the doorway or window?



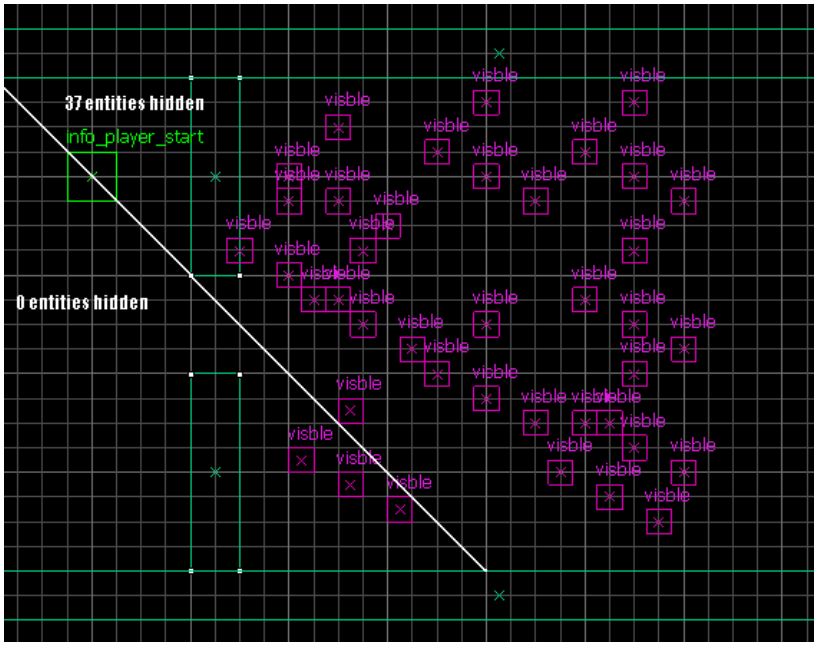
If the player (the green box) would stand here he would only see the bottom four very_hard_to_render_entity's, but all eight of them would be rendered! That's eight very_hard_to_render_entity's! we could solve this situation similar to Weeble and Bob's wall:



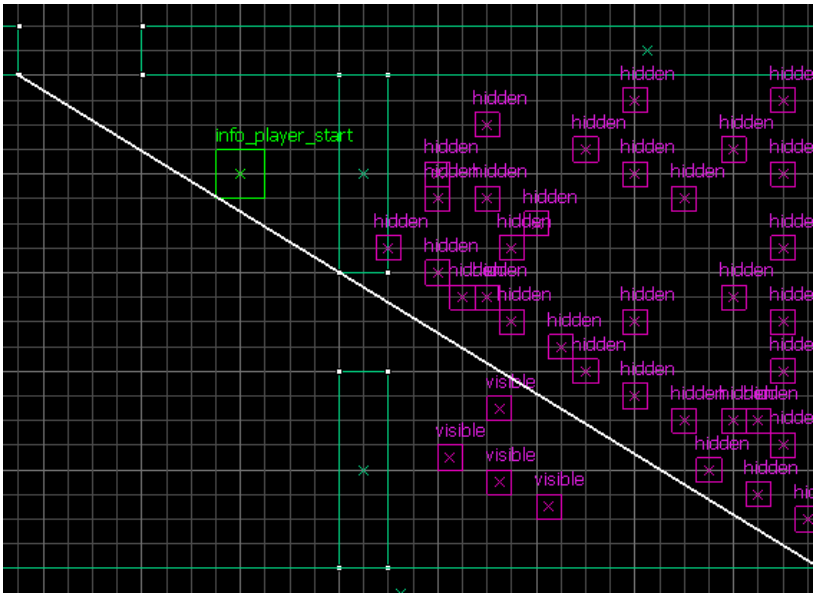
Which would be ideal if these eight entities is all that is hard to render. But imagine this room being racked to the roof with these entities:



Not very ideal: still 23 entities will be drawn, about 1/3rd of the total. Isn't there an alternative? Offcourse there is! Don't you see this doorway is nothing more than the corner from the beginning?



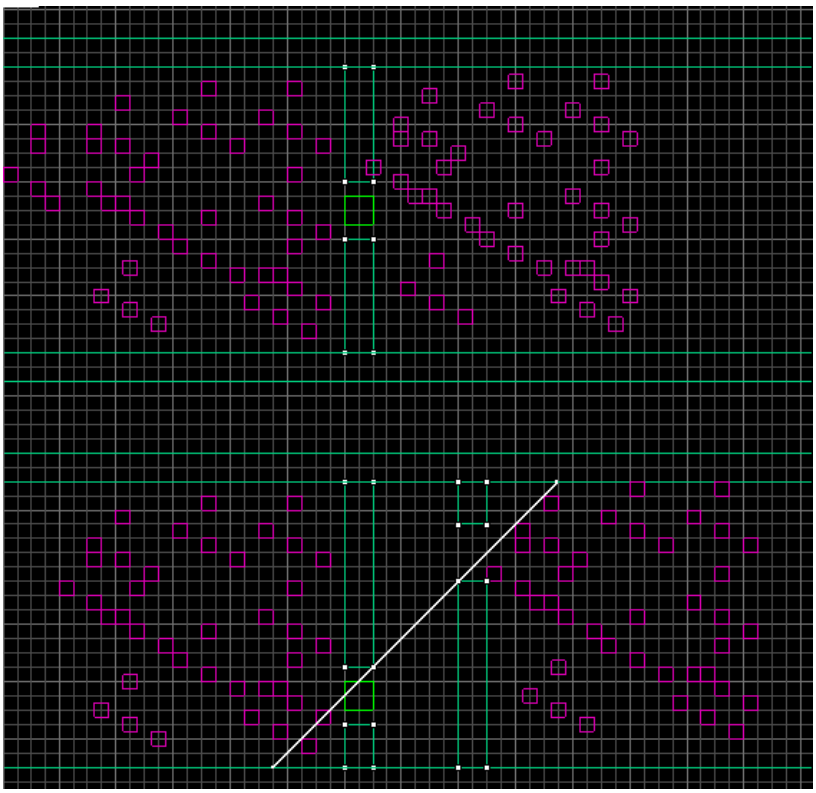
This seems far better, but think about this: If the player is just a teensy weensy more to the top, only three entities will be visible. However, if the player would move a teensy weensy more to the bottom, all entities would be rendered! What we need to do here is choose an angle where it is affordable to have more entities drawn. The more horizontal this hint-brush is, the less will be hidden from the player, but the bigger the space is where the player must be for this hiding to happen. The more the hint-brush is vertical, the more entities are hidden, but the smaller the area is where that will happen. Confused? You should (NOT) be. Look at the two pictures above and you will see what I mean. The hint-brush in the bottom situation is near-useless, as the entities will only be hidden in a very small corner. However, if we could make the hint brush reach past an entrance of somekind, we would be hiding these entities for a larger area, as can be seen below:



Now for everyone being in the same visleaf as the player is here, or on the other side of the entrance at the top left, only four entities would be rendered. If this hint-brush would have been like it was in the previous situation, any player on the other side of this entrance would likely see a lot more entities rendered.

Hinting requires you to think miles ahead. Don't just try to hide one room from another. Try to hide them the most optimally. Try to hide roomS from other roomS. Think about what area's are the most intense (most players, most action, most monsters, most brushwork, most entities, HDR intensive, etc) and try to optimize those parts the most. For those area's, it might pay off to hide every small nook you can possibly hide. For other area's this isn't needed at all.

Carefull planning helps you a lot in optimizing as well. If you have a very calm/easy to render area between two very intensive area's you can make sure the player never has to draw both intensive area's at the same time. Compare the next two situations, the first is completely unoptimizationable (nice word for Scrabble), the second will actually be very easy to optimize:



As you can see, with the white line being a HINT-brush, there is no place in this bottom level where more than 40 entities are visible. When the player is in the left room, or in the middle hallway below the hint brush, only the left entities will be drawn. If the player is in the right room, or above the hint brush in the middle hallway, only the right entities will be drawn. Compare that to the top situation, and you will know which situation is best fps-wise. Offcourse, you must remember: Does this new hallway fit the theme? If it doesn't, be creative :P

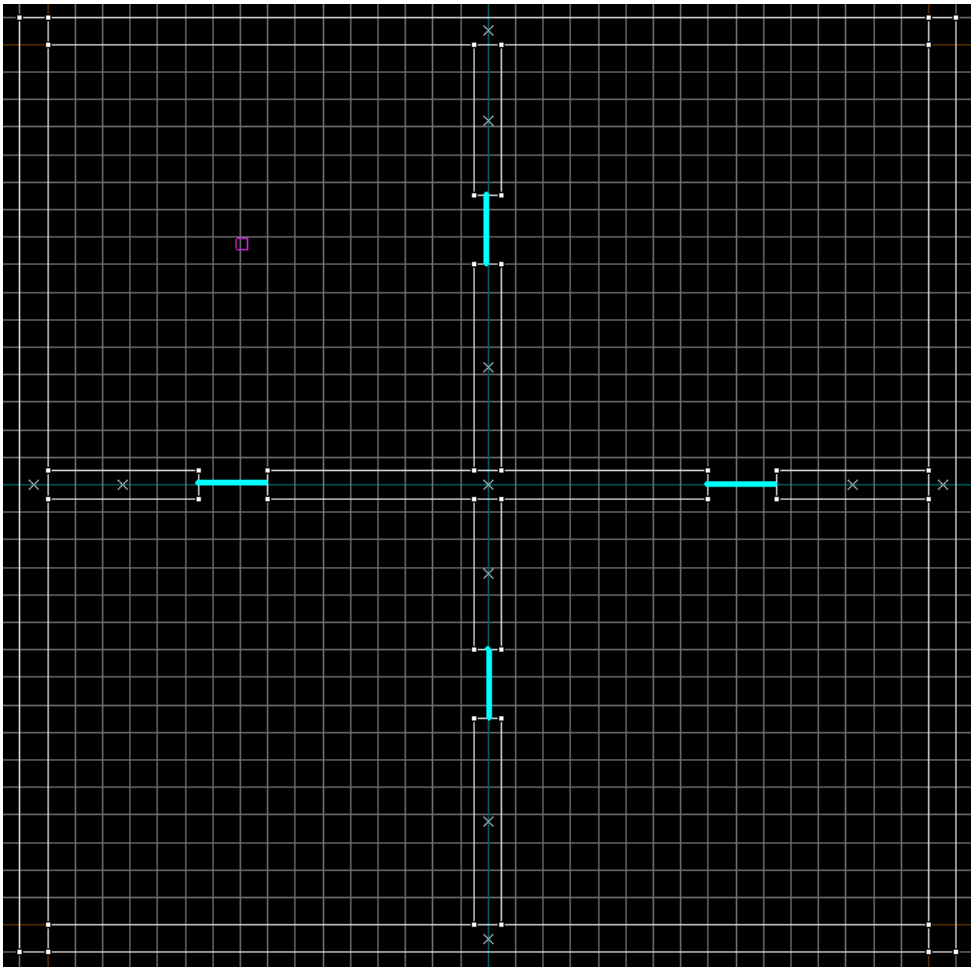
I could make up examples all year long, but I got better things to do. Remember the lessons you have learned so far, remember the ones you are still going to get, and experiment with them. You know how it works, so get that useless grey matter in your head working! A usefull tip is to open up paint (or any other paint program) and start drawing lines on an overview screenshot from Hammer and Glview. Find out how your visleafs are made, and for all i care you start drawing lines in paint to see why that one visleafs draws that other visleaf when you dont want it to, and how you can fix it. Like all images above, they can help you determining where your HINT-brushes go best. Also remember the 3rd dimension! All these images are just 2d, but along with the extra dimension comes a complete new level of mathematics. I hope you are up for the task, I am not going to do it for

you. Never. Ever. Now shoo to the next chapter.

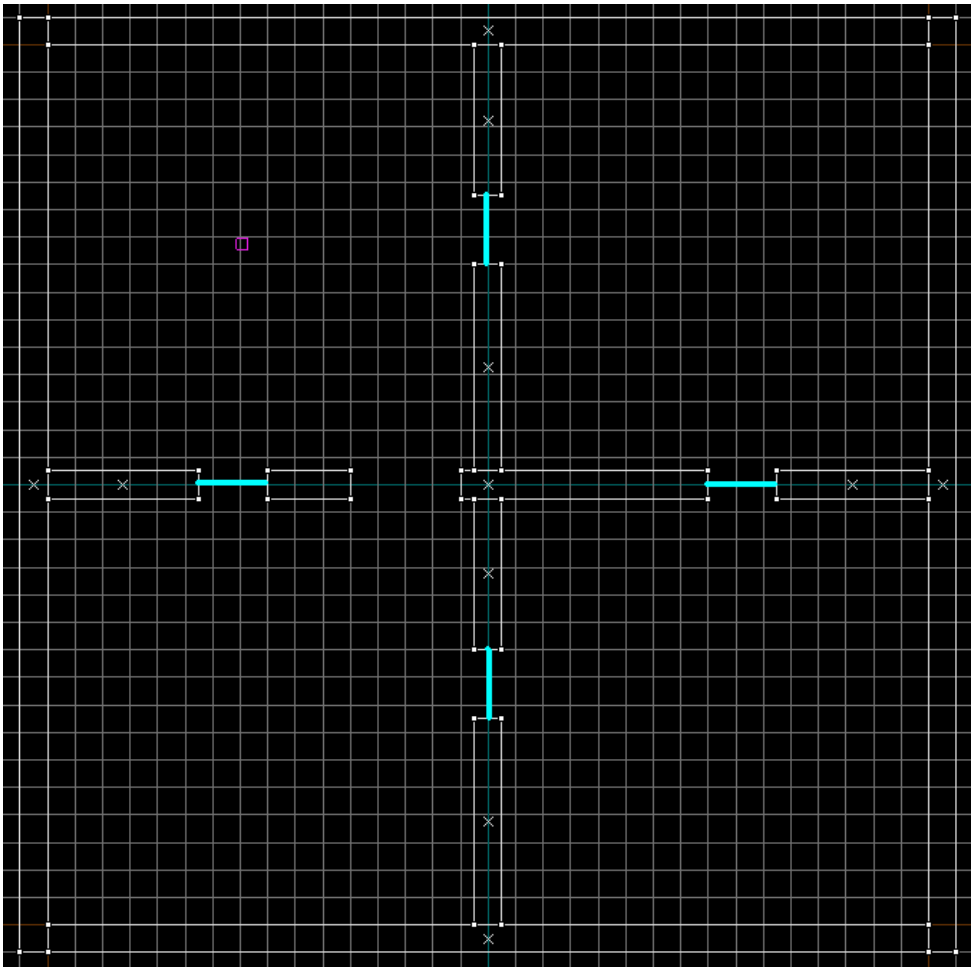
Func_areaportal(window)s

Sometimes you have visleafs that need to be visible at one time, but hidden at another time. Conditional visibility if you like. The way we do that in HL2 is by using areaportals. Remember what a portal is, and how they are used for visibility? The areaportal is a conditional portal. In it's closed state it acts like a solid wall, meaning it blocks visibility, in it's opened state it acts like any other portal (allowing visleafs to see through them). Areaportals are defined by giving a brush the "tools/toolsareaportal"-texture (all sides of the areaportal brush need to have this texture) and making the brush a func_areaportal or func_areaportalwindow (the difference between these two will be explained later).

However, we have a problem. Areaportals can open or close ingame, and our visibility table (the table of data containing which visleaf can see which other visleafs) is static. This clashes somewhere, doesn't it? Nope, it doesn't. Because of one thing: AREAportals. Areaportals divide the level into areas, mini-levels if you like. Each area is a complete mini-level indeed, as each has it's own entities, it's own visleafs, etc. They can even LEAK seperately!



The four (blue) areaportals divide this level into four area's. Topleft, topright, bottomleft, bottomright. Each of these area's must be sealed like a normal level. They can't even leak to each other:



See how the top-left area leaks into the bottom-left area, actually creating a single area? The left areaportal says these two area's must be seperated, but there is only one area (I failed to seperate both area's with world brushes and areaportals). This situation is considered as a leak. The pointfile will show a leak from one side of the areaportal to the other. This also explains why areaportals need to be fitted exactly. Any space around them results in a leak, just like above. For instance if you are trying to stop the inside of a house to be rendered don't just put an areaportal in the front door, but also put ones in the backdoor, all the windows and maybe even the chimney. It's very easy to forget an entrance, so be carefull.

If a line can be made from one side of an areaportal to the other side without crossing world-geometry or other areaportals, this areaportal has leaked!

This shows that you can't just place areaportals wherever you like: Each areaportal must seal an area completely. Areaportals are only allowed to touch two area's, no less (this will cause a leak as previously explained) but also no more (you can't have areaportals cross other areaportals or area-dividing world geometry). Since areaportals may span through brushwork, it's also possible to use one areaportal for two doors by stretching the areaportal brush across both doors, if they are next to each other.

If **ALL** areaportals between the player and a certain area are closed, this area doesn't exist for the renderer. In other words: You must not be able to walk to this area without crossing world geometry or closed areaportals. In the first example that means:

- If only North is open, only the top area's are considered to be drawn
- If East is open, only the right area's are are considered to be drawn
- If North and East are open, only the bottom-left area isn't considered to be drawn
- If North and East and South are open, all area's are considered to be drawn

Offcourse the normal visleaf visibility rules apply to whatever is considered to be drawn. Areaportals only affect visibility, whatever entity or player is bouncing around in these other area's happily keeps bouncing. Areaportals are NOT solid, you can walk, shoot, bounce or fly straight through them. They won't block light either. Areaportals cause a slight slowdown ingame, but usually this slowdown is much lower than the slowdown caused by any geometry they hide. Still, it's always good to check if the effects of an areaportal are good for the framerate or not. Areaportals are also said to cull visleafs (in their open state), but whether or not this effect is solely caused by the areaportals cutting up any visleaf that touches them is unknown to me.

In multiplayer games, areaportals are considered seperately for every player. That is, if player 1 cannot see area A because of a closed areaportal, then that doesn't mean that player 2 cannot see area A too; Player 2's computer will also determine what area's he can or cannot see, and they don't have to be the same as that of other players. So you don't have to worry about players seeing the rooms around them disappear because an areaportal has closed.

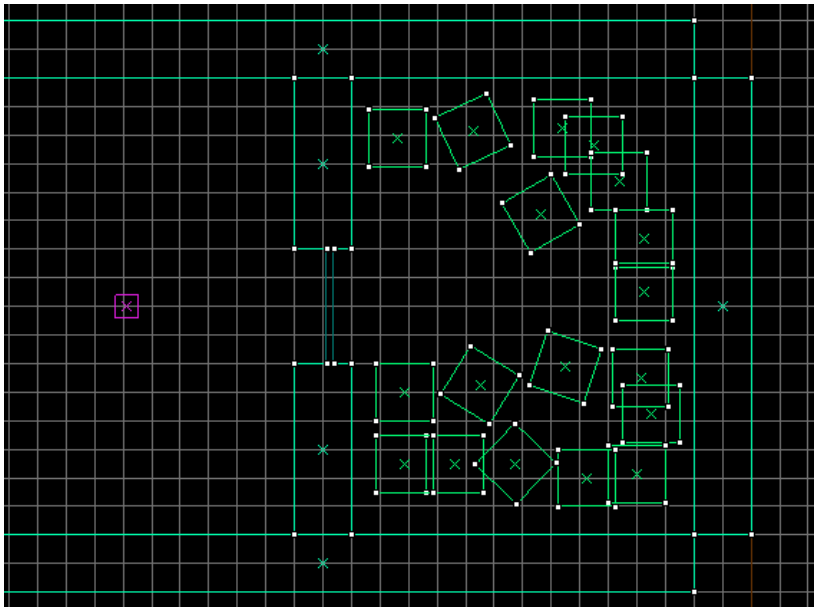
When looking straight at an closed areaportal a player will see either HOM or the skybox, just like with NODRAW-brushes. So unless you want that, you need to cover up your closed areaportals. For instance by a door!

As said earlier, there are two types of areaportals, the normal func_areaportal and the func_areaportalwindow

func_areaportal

This areaportal can recieve open and close inputs, so you can use triggers (eg trigger_multple's, doors) to open and close them! They can also be linked to a door so they open when the door is open, and close when the door is closed. The effect is the same for all players in a level though, if the areaportal is open then it's open for all players, and if it's closed it's closed for all players.

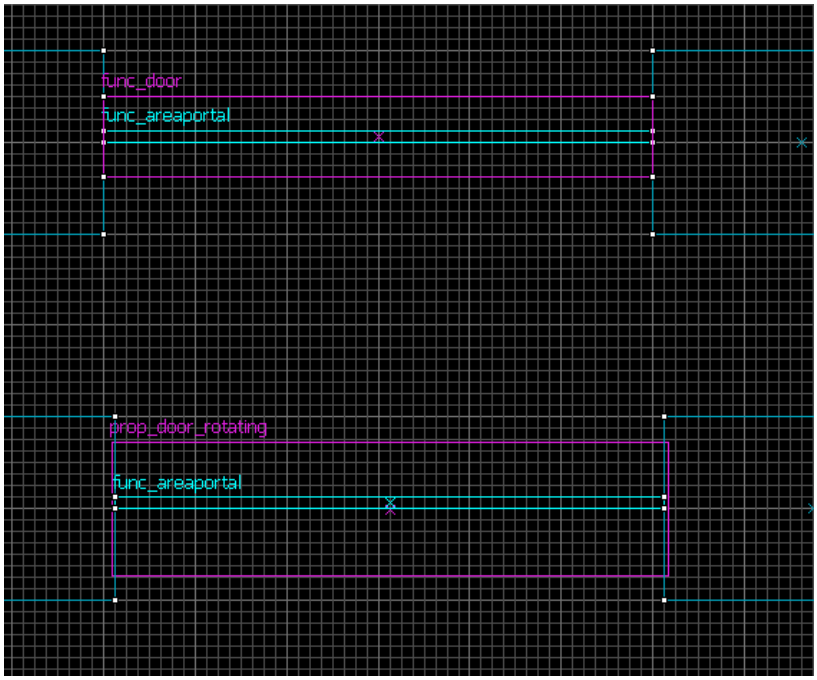
Imagine a room behind a solid door: It's stupid to render this room when the door is closed, but even more stupid to not render it when the door is open.



Imagine this room. The player (purple) wouldn't be able to see anything in the room, because the door is closed. However still everything in this room is rendered. If we put an areaportal in the doorway, and link it to the door that gives access to this room, then we can make sure this room is only rendered when the door is open or when the player is INSIDE this room make sure the hallway is only rendered if the door is open.

In this example, when the areaportal is closed, any player outside this room won't have anything inside the room rendered, and any player inside the room wont have anything rendered outside the room. How cool is that?

To make sure the door itself is rendered when the areaportal is closed (and to hide that ugly HOM or skybox appearing instead) we need to make sure the areaportal is COMPLETELY INSIDE the door. This is easiest when the areaportal is as thin as possible, 1 unit:



With both doors the door is still visibile when the areaportal is closed. For models only the bounding box needs to be on both sides of the areaportal, for brushes atleast a part of that brush (and only the part that needs to be rendered).

To link the areaportal with the door either:

- Enter the name of the door in the "linked door" property of the func_areaportal
- Use the OnOpen and OnFullyClosed outputs of the door to trigger a Open or Close command with the func_areaportal

The Open and Close inputs can also be used to control the areaportal using triggers. This can be usefull for instance when you want to link the areaportal with a func_breakable (OnBreak->Open), or Close or Open an areaportal if a player has reached a certain part of your level.

It's sometimes a good idea to have areaportal-linked doors autoclose, so you have the optimal effect of your areaportals. Though it looks stupid, it's great for those players that can't close the doors behind their backs. Some players just don't have any manners.

Func_areaportalwindow

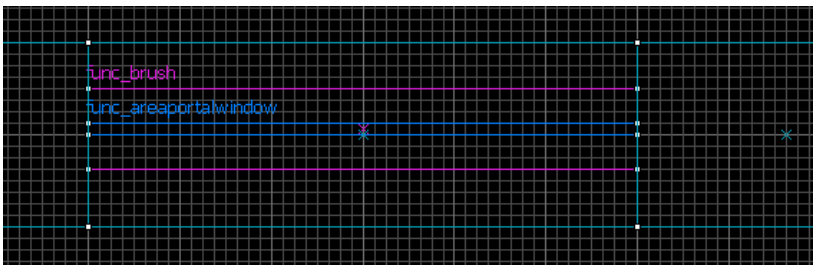
The func_areaportalwindow differs from the func_areaportal in two ways:

- It has no in- or outputs
- It opens or closes depending on what distance the player is at
- It is clientside, meaning that it can be open for player 1, while being closed for player 2.

Wow, what was that second one? Opens or closes depending on the distance? Yes. Aren't you excited now? You should be. These areaportals have two properties, the "Fade start distance" and the "Fade end distance". The idea is that when you are far from a window you can't really see through it, so the window brush is visible only. But, as you come closer, the window brush becomes more and more translucent untill it is completely see-through. This areaportal is closed when the window-brush is visible, and as soon as the window-brush becomes more translucent (as the player moves from out of the "Fade end distance" to the "Fade start distance") the areaportalwindow opens and the 'outside' is rendered.

Take this example. When the player is near, the window is translucent and the room with the drums is visible. As the player moves further away from the window, the window becomes less see-through. When the player is outside the "Fade end distance" the window isn't see-through at all, the areaportal has closed and the drums and their room isn't rendered.

Setting up an func_areaportalwindow isn't as easy as setting up a normal func_areaportal.



You (again) need an areaportal, the func_areaportalwindow and a brush to cover it up (in this case a func_brush with a window-like texture). Setting this up is more diffucult, luckily the func_areaportalwindow does everything we need:

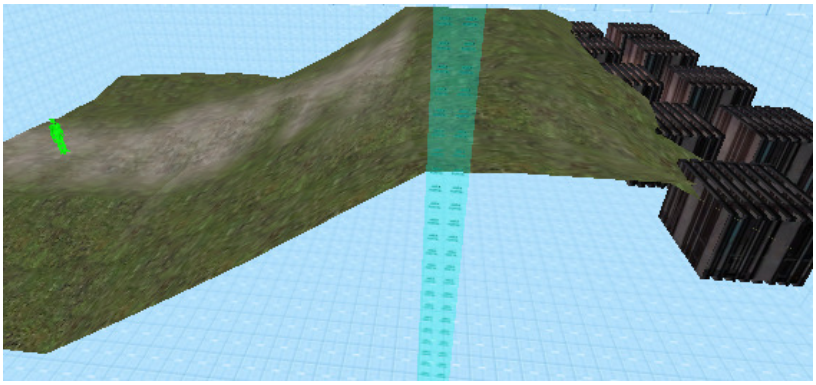
- Fade start Distance: Where the window-brush should start fading (default is 128 units)
- Fade end Distance: Where the window-brush should be completely opaque and the func_areaportalwindow should close (default is 512 units)
- Rendered window: The brush that is going to be faded in our out (the glass of the window). In this case the func_brush

There are more options, but I'll leave you to experiment with them yourself.

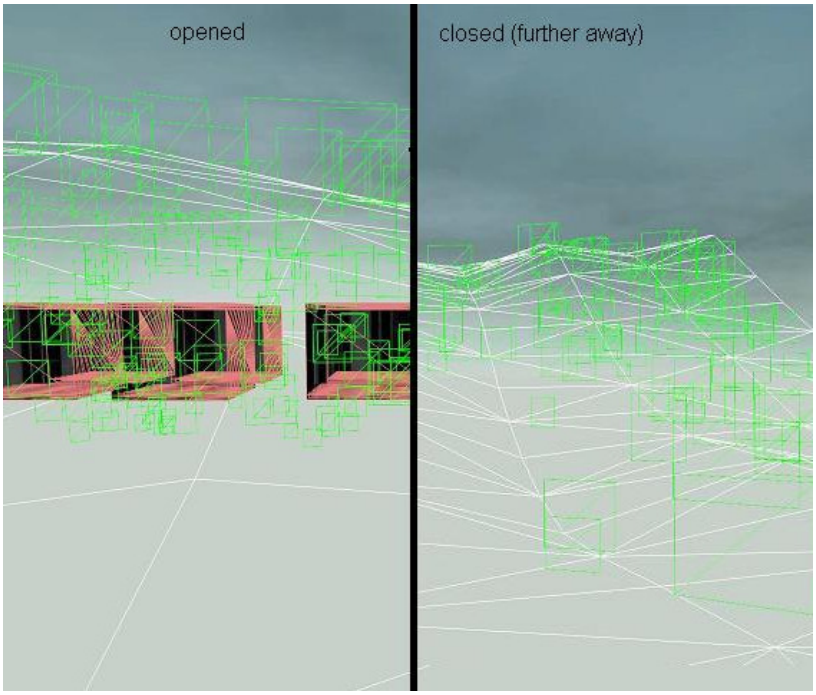
Note that the func_areaportalwindow only has one usefull input, the Kill command. If the window is destroyed (eg broken) you should also Kill the areaportalwindow as you can't fade a brush that doesn't exist... Failing to do so will make your map look stupid.

Areaportalwindows work awesomely with env_lightglow: You can simulate a window which looks really bright from far (use the env_lightglow for the blinding effect, and use a brush with a white texture to fade in and out with the areaportalwindow), and normally seen through from close. An EXCELLENT way to optimize maps, as long as the effect fits your theme. Make sure you match the fading distances between the two entities for optimal effect.

Areaportalwindows can also be used without a fading window, which is great for hiding area's the player can't see if he is far away from the areaportal. A good example is when you have a hill made out of a displacement and don't want the player to see the other side. The nice thing about areaportals is, is that when you have a skybox in your map, closed areaportals also show skybox. So if we would close this areaportal (purple) the player will see skybox! Which he would also see without the areaportal. You can also throw grenades through the closed areaportal, as if it wasn't there.



View from the player ingame: (wireframe, shows outline of everything that is being drawn)



Notice how you see (should see) exactly the same if we didn't have wireframe on, only to the right the boxes are not rendered because the areaportal was closed. For the rest, it works just like the previous window, just make sure the window brush is non-solid if you plan on it being passable.

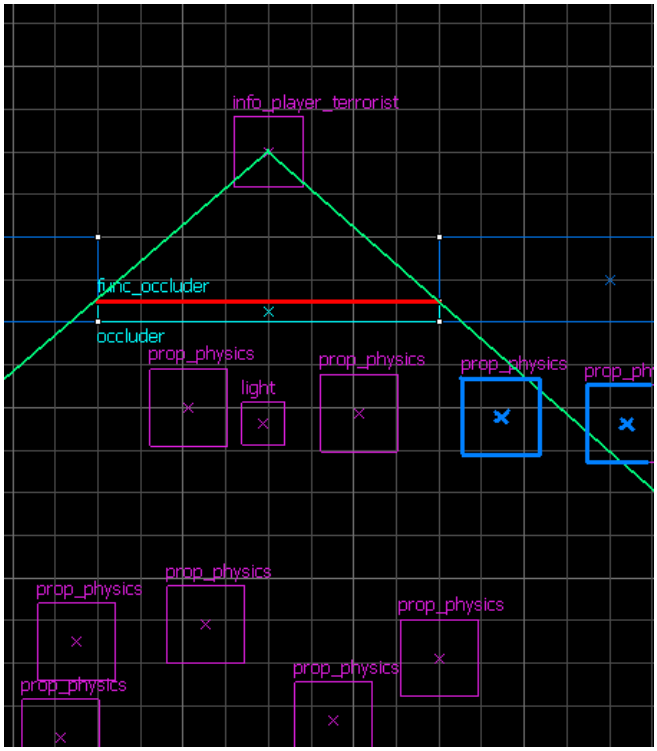
The only problem with this method is that you need to make an area of "the part behind the hill", so that may be a problem for complicated outdoors. Still, it's probably better than nothing.

Again, for areaportals and -windows, experiment a bit. Build yourself an example level, place a few of these entities in them and find out what is rendered (see this chapter for how to do that). Use entity inputs (type "ent_fire [name of areaportal] [open or close]" in the console to open or close an areaportal in your level. Experimenting is the greatest way to learn.

Func_occluders

The func_occluder hides entities from you, simple as that. It can be turned on and off by triggering it to do so.

You make one by making a brush with the NODRAW-texture, and giving the occluding sides the "tools/toolsoccluder"-texture. ONLY THE OCCLUDING SIDES SHOULD GET THIS TEXTURE.



The occluding side is marked RED. All other sides of this func_occluder should be NODRAW. In the picture, purple entities are hidden, blue ones are still drawn. The occluding side is the brush-side which the player has to look through in order to occlude the entities behind that face. So if the player was south of this occluder and you wanted to occlude some entities north of this occluder, the south brush-side of this func_occluder becomes the occluding side. Confusing? Nah. Experiment with the example map and you will know what I mean.

Awesome? Not really. Some bad point:

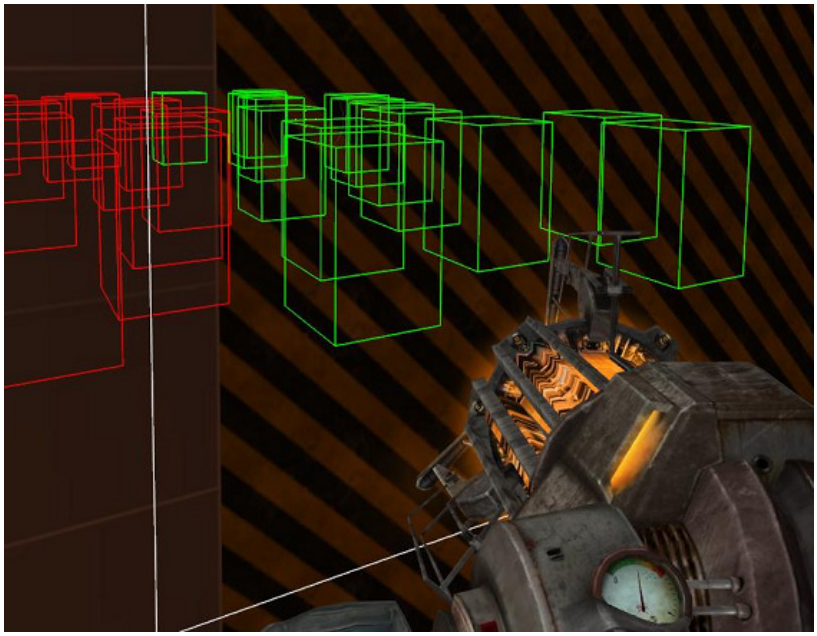
- Func_occluders work real-time. That is, while playing your game, the engine has to decide which entities are, or aren't blocked by any func_occluder. This requires some calculating power, therefore you have to make them worthwhile: a func_occluder has to hide at least a few (more or less) complicated models to be effective. Failing to do so will lower FPS, because the cost of the func_occluder becomes higher than its profit. To check, run your level and use "r_occlusion 1" and "r_occlusion 0" in the console to see where you get the best performance. If "r_occlusion 0" gives you higher FPS, ditch your occluder, otherwise keep it. Simple as that.
- The occlusion for entities is calculated per side. The engine has to decide for each brush-side with the OCCLUDER-texture if it does or doesn't occlude a certain entity. That is why you should only give the occluding sides the 'tools/toolsoccluder' texture and the rest the nodraw-texture. That is, only the front of the occluder should get the occluder texture. Even the small sides of the brush could bring down the FPS dramatically, even though they aren't needed in the occluding process.
- Func_occluders only block entities, not brushes. If you want to block brushes as wel, use an areaportal instead. Also, shadows (even of blocked models) are still drawn.

Using r_visocclusion (again 1 is on, 0 is off) you can see your occluders in action:

red means entity is drawn
green means entity is blocked (not drawn)
white lines denote active func_occluders
entities that aren't affected by occluders have no border

func_occluders don't need vis or rad to be run, and work on maps with leaks too (though not really usefull, it does imply you can test them while having leaks). They can cause errors when touching a func_areaportal or when being in a brush that touches more than one area ("Warning: func_occluder straddles multiple area's"). An area is defined as a part of your map sealed of by areaportals and world brushes.

An example of usage is (like in my example map) a func_breakable, which blocks vision of models, but once its broken it deactivates the occluder so the player can see all the models like he should:



Sometimes it's better to use an areaportal for things like this, but they are not always possible. Occluders can easily be used inside entities which cover up a lot more entities. Or inside hills, where they could cover up entities on the other side of the hill like in the areaportal example.

My recommendation is to use this entity as a last resort, you can hardly use it to such an extend they are worthwhile. Unless you have a space with a lot of entities. Always try hints and areaportals first. And if you can't get enough effect of them, try again. And again. And again. Only after you've tried for more than two months should you start considering occluders.

The example map by Valve about occluders can be found here: "sourcesdk_content\hl2\mapsrc\sdk_occluders.vmf"

Checking your progress: Know your enemy

Half-life 2 has some very usefull commands to show how well your map runs, what it's chocking on and what is rendered but cannot be seen.

Most of these commands require the HL2 developer console. So learn to use it!

It's the same place as where you entered the codes for your godmode-cheats you had to use to finish HL2.

Showing visleafs

If you want to know how your level is divided into visleafs you should compile your map in gl mode, and then use a program called "Glview.exe" to view this gl-file.

In expert compile mode (I hope you know how that works) make a new configuration, called Glview (this allows you to use it for all your maps), and create the following commands:

```
Executable      parameters
$bsp_exe        -Glview $path\$file
Glview*         -portal $path\$file.gl

*the command for Glview isn't known by Hammer , so youll have to find it yourself: press executable, and browse to
Glview.exe (in the sourcesdk\bin directory)
```

make sure both commands are set to run, and press GO! You can use your mouse and the WASD-keys to move around your level when in Glview. They 1 and 2 keys toggle world-brushes and portals. Glview won't work if there is a leak in your level, or any other error that may fail vbsp.

Note that Glview has a bug where it can't open files located in a directory with a dot in it. For instance, trying to open "C:\steam \your.username@hotmail.com\mapsrc\testmap.prt" will make Glview look for "C:\steam\your.prt". It's very sad Valve made a stupid mistake like that, but it's just the way it is. Either use the copy command in Hammer to move the files Glview needs to a path without a dot in it, or download my version of Glview (at your own risk, offcourse) which doesn't have this annoying bug.

Ingame you can also use the console command

mat_leafvis 1

to show the visleafs in your level as you walk through them.

Showing framerate

Enter

cl_showfps #

in the console, where # is:

0 - Off

1 - Displays a simple meter

2 - Displays a more advanced meter

Showing what is rendered

Enter

mat_wireframe 1

to show everything the engine draws, including models, but only their outer lines (you can toggle it off with "mat_wireframe 0") Therefore, you can see through walls to see EXACLY everything the engine draws. If you were Weeble or Bob, you wouldn't see the pipes without mat_wireframe on. Mat_wireframe is usefull to check if and how your hints, occluders and areaportals are working.

Showing what is chocking your framerate

Enter

+showbudget

in the console to get a screen with graphs for various types of systems. For a good list of what each means, check out the HL2 wiki about showbudget.

You can toggle it off with "-showbudget". Quite logical, no?

You can also use one of these commands to toggle various effects, and using an fps-meter to determine the impact of that system

- **mat_showwatertextures 1**: Shows which entities are drawn with water-effects because they are (partially) inside water.
- **r_occlusion 0/1:** Toggles func_occluders on and off, to see their effect.
- **mat_bumpmap 0:** Turns off bumpmapping, so you can see what kind of impact it has on your map ("mat_bumpmap 1" to put it back on)
- **mat_specular 0:** Turns off reflections, so you can see what kind of impact they have on your map ("mat_specular 1" to put it back on)

If all else fails

A few usefull things to consider using when your map is still lagging to much is the use of the following things

env_fog_controller The env_fog_controller is an entity that makes fog, but its most appreciated function is the fact you can set a maximum visible distace. In other words, everything more that so many units from the player will not be drawn. The fog can be used to mask this clipping, so players will think you made a nice map with fog for a nice effect, but actually you were trying to hide your bad optimizing skills...
Just place the env_fog_controller anywhere in your map, and enter the "fog start" and "fog end" values. Everything further away than the "fog end" value from the player will not be drawn.
For optimal effect, make sure the primairy fog color is whitish (its always like that) and that the primairy color is more blended with your skycolor. For instance, if your sky is yellow, make the secondairy fog color yellow-whitish, and if your skycolor is blue, make the secondairy fog color blue-whitish.

You can offcourse google some pics of fog to find out what color to make it. Fog in England is always grey, evening fog darkblueish, etc. You can even use yellow for as if it was dust in a de_dust map. Offcourse if your map isn't based on our earthly athmosphere, make up your own colors :) . Experiment to find the values that suit your map best. You may need to find a 'foggy' skybox to suit your map best. Sometimes fog just doesn't fit the map though, can't help that.

Also, when you are using a 3d skybox, make sure you set the same fog properties in the sky_camera entity, or youll see strange effects with near buildings being fogged and far away buildings being completely visible.

fade-properties When you have helpers turned on in Hammer (the diamond shaped button on the top toggles them) you'll notice entities get circles. You can use these circles to denote at which distance your model entities should fade out (inner circle) and dissappear totally (outer circle). These circles correspond with the "start fade distance" and "end face ditance" properties of the model-based entities.
Few words of warnings:

1) If you set the fade-distances too low, the object wouldn't be visible when you should see them. This also happens when you accidentally select the circles instead of something else and make then really small. Because of that its best to hide helpers as long as you don't use them.
To reset the distances, go to the properties of the entity and make sure "start fade dist" equals -1, "end fade dist" equals 0 and "fade scale" equals 1.

2) static entities can be invisible when the player is out of its area (and out of the room) but when dealing with moving entities remember they may be brought to open area's where they can't be visible while they should.

3) You only gain performance outside the "end fade dist", fading entities cost just as much as (even slightly more than) normal ones.

4) On machines running DirectX level 7, the props will fade earlier than the values set in the entity to further improve rendering speed. This can be controlled on a per-MOD basis by creating a "dxsupport.cfg" file for your mod and specifying the values for the console variables "cl_detaildist" and "cl_detailfade" for the various dx support levels (taken literally from www.Valve-erc.com).

Fading objects is fun, but it may look rubbish. Since you only gain performance one the objects are completely invisible, there really isn't much you can gain.

Func_lod A func_lod is a brush-based entity that brings fade-distances to brushes. Just like the fade-distances does on models, this entity allows you to make brushes that disappears after a set distance. Great for outdoor optimization of details.

This is a real entity, so it won't seal leaks, and it costs more to render than func_detail.

Lightmaps To make vrad.exe take less time, you can increase the lightmap size. vrad.exe divides each face it has to light in squares, and then calculates for each square how much lighting it needs. Afterwards, all squares are faded into each other to create a nice fluent effect.
Reducing the size of these squares increases vrad.exe-times and qualitiy of lighting, increasing the size does the opposite.

You can specify the size of the lightmaps on a face by using the texture application tool. To the right of "texture shift" you'll see a box where you can input your custom lightmap scale. Default is 16 (units per luxel). Increasing this number will decrease the time vrad.exe takes, but also decrease lighting quality.

Because larger lightmaps mean less quality, only lower them on faces that are equally lit and have no shadows falling upon them, or faces players can hardly see (for instance behind a model or far away faces). The editor has a special view for lightmaps, in the 3d view click "camera" and select "lightmaps"

Also see the example map from Valve about lightmaps: "sourcesdk_content\hl2\mapsrc\sdk_lightmaps.vmf"

HDR Because HDR is only available for the somewhat more modern cards, there is no reason to not use it for optimization purposes, as all these cards should be fast enough to play your map with decent speeds anway. Just so you know.

Physics Having a lot of physics happening at the same time can really bring down any computer to it's knees. Make sure this never/rarely happens! For multiplayer games use prop_physics_multiplayer instead of prop_physics: you will get crappy/buggy physics, but atleast you got some more fps. And that is what it's about if you need these things. For the rest just use your mind and don't place 1000's of objects in a high-action-area. Be wise. For Counterstrike mappers: keep those physics objects away from bombsites if the explosion lags like usual. Only one thing sucks more than a smoothly running map with a laggy 'explosion de finale'.