

VIS for dummies, and why use detailed brushes?

This is a short tutorial for us “dummies”; it will help you get a grasp of what VIS is (does), and why you should use detailed brushes. To explain this I’m **not** going to use all the right terms like “view portals”, “leaf nodes” etc so the advanced mapper might think this is a stupid tutorial (but I don’t care).

Why compiling the map!!

To turn the editor map file from editor code into a useable game map, it must be processed or “compiled”. It creates the “BSP” file.

Compiling for Mohaa is divided into three parts, BSP, VIS and LIGHT. The first, the bsp process establishes the data organization of the map. The second part, the *vis* process, builds the walls and *internal spaces*. The third process lights the map, calculating brightness and shadow and the color of the light that falls on every map surface. (The first part creates the BSP file, the other parts add their info to the BSP file, it’s only the BSP file you will need when you play your map in Mohaa).

Now we focus on vis and “internal spaces”

When you have made a map in the editor you have created a “space” for the player. The player will move around in this “space” when you play the game. The thing most people think of, as the “space for the player” is everything inside the skybox. Picture 1 shows, with one side of the skybox removed, that space.

NO PICTURE AVAILABLE

When compiling the above map the VIS process would basically see this as one “space” and noting more, compile would be fast.

Lets add some brushes to this boring map!!

This map is kind of boring so lets add something for the player to look at. First you should know that when you add a brush to a map it is a **structural** brush unless you tell it to be otherwise. Don't think more about that now! Just remember that it's structural.

This is the picture with the brush added (now the player really has something funny to look at). The brush has six sides but the one in the top touches the sky, and the one in the bottom touches the floor. Basically we got four new sides in the map.

NO PICTURE AVAILABLE

When compiling this map, things start to happen! Structural brushes make a split in the view or to use the same term as earlier, it causes a split in the "space". And it's not the brush itself but each side of the structural brush that causes a split. In the above you could say that you get four splits in the player space, (or in other terms the player gets four spaces to walk around in). The next picture shows how the compiler will see the map as four spaces. The top of the skybox is removed and to show how the compiler sees the map I have used some see through textures to illustrate the splits, those textures have nothing to do with this map.

NO PICTURE AVAILABLE

This is what the compiler sees when compiling. As a player you will not see these four "spaces" you will only see one "space" with a brick wall in the middle. The structural brush has added to your maps VIS data size and compile will take longer.

Why will compile take longer???

It will take longer since now the VIS compile part needs to calculate which "spaces" that can be seen from another "space". You might have seen this when compiling, it goes something like this "XXX mightsee: YYY cansee: ZZZ (XXXXX chains)". Look at the modified picture below.

NO PICTURE AVAILABLE

When the player (marked with a ring) is inside the turquoise space he can actually see into the red, yellow and purple spaces. Hmm **I should not** use the term “player can see” because the compiler doesn’t care what you as a player can see or not. The compiler determines and calculates what the “turquoise space can see”, and a space can see any other space unless there is a structural brush that block the view. From the arrows you can see that our structural brush isn’t blocking the view from one space to another. Compile could be something like this;

Turquoise *cansee* Purple, **Turquoise** *cansee* Red, **Turquoise** *cansee* Yellow

Red *cansee* Purple, **Red** *cansee* Turquoise, **Red** *cansee* Yellow

(Note: this is very simplified, not totally accurate but you can think in these basic terms)

Well I don’t need to write them all down, if you don’t grasp it try and read again.

Did you notice that I mentioned, “unless there is a structural brush that block the view”. I have made another example picture with some more structural brushes added, remember that they go from the floor all the way up and touches the skybox roof. I also want to say that it’s not sure that the compiler splits the “space” in the same way as I have drawn it, but my pictures still applies to how it basically works.

As you can see in the next picture the Turquoise space can’t see the Red and Purple space. The structural brush is blocking the view. The VIS data is higher in this map since we got one more structural brush and the outcome is five spaces instead of four. This might not increase compile time that much. In the first example there are 4 spaces that could see 3 spaces each. 4×3 is 12 calculations. In this other map there are basically the following;

Turquoise can see 2 spaces, Green can see 3 spaces, Yellow can see 4 spaces, red can see 3 spaces and purple can see 2 spaces = sum 14 calculations. (Try and understand by looking at the picture).

NO PICTURE AVAILABLE

Well what's the use of this??? Increase in FPS my man!! When you play the first map with four spaces, it won't matter where you are in that map as a player, the whole map will be rendered since it has calculated that from one space every other space can be seen. In the second map it has calculated that if the player is in the Turquoise space, the "Turquoise space" cannot see the red and purple space and therefore they will not be rendered. As soon as the player moves in to the green space the red space will be rendered too. Now you might get a clue to how things work.

Use your imagination; imagine that you have 10.000 "spaces" in your huge map and a lousy graphics card. What do you think the FPS will be in a map where every "space" can see 9.999 other spaces and therefore they will all be rendered! And what will compile time be? **Well placed structural brushes** that blocks views will make more spaces in your map. That will increase compile time; on the other hand it will also improve FPS. The increase in compile time might not be that high since "chains" in the mightsee, cansee process will be shorter. The chain part in the mightsee, cansee process will be longer if a huge amount spaces can be seen from each other.

Short summary before we go on!

Remember that all brushes you draw in a map are structural brushes (unless you tell them to be different). Structural brush sides create a split in the "space". VIS calculates spaces and what each space can see; it doesn't care what you actually can see as a player inside the game.

All structural brushes add to the VIS data size, more structural brushes gives generally longer compile time. If the map is well planed so that you get more spaces but have spaces that can't see every other space you will gain FPS in you map.

Now me move on with some more pictures and soon we will start with detailed brushes

By know you know that structural brush sides create splits in the view, in the next picture you see a basic room that any mapper could have created, but every mapper doesn't think about the consequence of their mapping. In the picture you see a stair and every part of it is drawn with brushes.

NO PICTURE AVAILABLE

First imagine the room to be completely empty, then you could assume that the compiler will see this room as one "space". Look at the picture again and try counting the number of structural brush sides. You remember the picture where we put the brick wall in the middle? That space was divided into four spaces. How many spaces do you think this room will be divided into? I can't give you the exact answer because as I said before it's really difficult to say exactly how the splits are done. Structural brush sides will split in all directions, diagonally and vertically. The only way I can show you the difference is by showing you some facts from the compile process. **Room 1** is a map that looks like the first picture in this document, **Room 2** is a map that looks like the second picture in this document and finally **room 3** is the map with a stair in it.

Compiled map	Room 1	Room 2	Room 3
Portalclusters	1	4	102
Numportals	0	4	274
Numfaces	6	20	534
Visdatasize	16	40	1640
Rows in mightsee, cansee process	0	8	(Lost count) +500
Average clusters visible	1	4	80

As I said earlier in the beginning, the experienced mapper might not totally agree with my explanations since they are very simplified. I'll bring in some new words and if you don't get anything of this, then keep on reading about detail brushes after the "advanced box".

THE ADVANCED BOX

What does the BSP stage do?

The player-navigable space inside the World is split into convex volumes bounded by planes. These convex volumes are called **Leaf Nodes**. The Leaf Nodes are stored in a binary tree called a BSP Tree.

How and why is the BSP created?

A map is a 3-dimensional volume of space extending +/- 4096 units from the origin in X Y and Z, containing smaller solid convex volumes bounded by planes (***structural brushes***). The goal is to use the fewest splits possible to split the space of the map into convex volumes, each of which do not contain any *structural* brushes.

A simple cube room is a convex volume bounded by six planes. Convex volumes are important for visibility, because it is simple and fast to determine whether two convex volumes can see one another. These convex volumes are stored as Leaf Nodes in a binary tree, making it easy to determine which Leaf Node the viewpoint or objects (entities, tris, patches) are in. This is called a Binary Space Partitioning Tree, or BSP Tree.

The Leaf Nodes are bounded either by other nodes or by *structural* brush faces. **A Portal is created for every face of a Leaf Node that is bounded by another Leaf Node. Each Portal is like a window from one node looking outwards into another.** The Portal information is not needed in the BSP, but it is essential to VIS, so it is stored in the portal file "mapname.prt".

VIS!

VIS is short for Visibility. The relevant part of this process is the creation of the PVS Table for the Portals in the map. As the player's viewpoint moves around a FULLY vis'ed bsp different areas become visible or hidden, depending on which area the viewpoint is in.

How is the PVS created?

Every Portal in the portal file is checked against every other Portal for visibility. Portal 1 is visible to Portal 2 if a straight line of sight can be drawn between any part of Portal 1 and Portal 2 without passing through a solid *structural* brush. Every Portal then gets a list of the other Portals that can be seen from it. This information is the **Potentially Visible Set** and is stored in the PVS Table.

Compiling with fast VIS?

The fast option does not create a PVS, **leaving every Portal visible to every other Portal.**

How do Portals affect visibility?

Every Leaf Node has one or more Portals (unless there is only one node in the BSP tree). **If a Portal belonging to Node 1 can see a Portal belonging to Node 2, Node 1 is visible to Node 2. When the player Viewpoint is anywhere in Node 1, every object in Node 2 is drawn. When the player Viewpoint is in Leaf Node x, every object in every other Leaf Node visible from x will be drawn.** A Brush Face is drawn when any part of the face is touching a visible Leaf Node. A patch is drawn when any of its control points are touching a visible Leaf Node. An entity is drawn when any part of its bounding box is touching a visible Leaf Node.

Summary!

Used effectively, solid *structural* brushes block visibility between the contents of Leaf Nodes. Curves, *detail* brushes and Entities **do not block visibility**. Visibility between two areas will only be blocked when both areas are completely hidden from each other. Generally, the more Leaf Nodes visible from the Leaf containing the Viewpoint, the more objects will be drawn. The aim when optimizing for visibility, is to have the smallest number of other Leaf Nodes visible from any one Leaf Node.

More of the map will be drawn if the BSP has a small number of large leaf Nodes, or **an**

inefficient arrangement of Leaf Nodes. (“Bold”; The problem with many custom made maps)

You can reduce the number of Leaf Nodes created by reducing the number of unique *structural* Planes in the map, by making all unnecessary *structural* Face-Planes into **detail Brushes**. (This will be the topic after this “advanced box”)

You can **control** visibility by placing Hints to create the Leaf Nodes and their Portals exactly where you want them to be, splitting the space into more Leaf Nodes only in areas where they are needed. (Hint brushes are not included in this document)

The time -vis takes is roughly proportional to the number of Portals in the map. The number of portals is displayed when you start VIS, as "numportals xxxx". The visdatasize is the size the PVS table takes up in the mapname.bsp file - this is limited to around 2MB.

A lot of Leaf Nodes means a lot of Portals. A lot of Portals means a long vis time.

The number of Leaf Nodes depends entirely on the complexity of the BSP.

Solution: detail brushes - Making a brush *detail* will stop it from affecting the BSP Tree, reducing the number of Leaf Nodes formed.

I’ll try and tie this together with my simplified explanation!

Try and see the colored boxes that I drew as created leaf nodes (I called them spaces), a portal is created where a leaf node touches another leaf node, a portal was a window to other leaf nodes. Try and see the side of a colored box as a portal, but only the side that touches a side of other colored boxes.

In that map there was four leaf nodes (colored boxes, called spaces), each leaf node touches other leaf nodes on two sides. $4 \times 2 = 8$ portals. Look at the result from compiling.

NO PICTURE AVAILABLE

As I said earlier outside the advanced box, all four “spaces” could be seen from each space since our structural brush didn’t block any views. You can see that each cluster as four visible clusters (itself + three others). The average clusters visible thing speaks for itself.

Now we stop and move outside this box, to join the ones who skipped this!

What is a detail brush and what can it do for us?

A detail brush is something that will look like the plain structural brush inside the game but it will not cause a split in the “space” when compiling. I’ll start this section with a picture that shows a basic room with some brush made stuff. This is a cellar from a map that I’m working on right now. The screenshots are from two parts of the room and it shows some of the things I’ve added to the room to make it look good in game. All things are made out of brushes and you can see a few of them in the following two pictures.

NO PICTURES AVAILABLE

Remember earlier the picture of a map with a brush made stair in it. As said each side of these brushes will cause a split in the “player space”, it would cut up this room in hundreds of small spaces. **I solve this problem by using detail brushes!** Detail brushes will be ignored when creating BSP and the splits in the “space”. I will only keep the floors, the walls and the ceiling as structural brushes; those brushes will help my frame rate later in the game since they actually might block the view to other parts of the map. The boxes, paintings etc will never block any view to other parts of the map.

How do I make a brush detail?

First select the brush in mohradian, then you got a few options; if you right click with your mouse on the 2D grid you will get a menu, select the option “make detail”, you can also use the menu select -> make detail or the short cut command ctrl+shift+d. If you have made this correct the brush will be outlined with green color in the 2D grid instead of the original black color (that’s when the brush is unselected).

Using filter in Mohradian

When you have a lot of brushes and other stuff in your map it often gets kind of messy in the 2D grid. Mohradian has several filter options that allows you to filter out different things. This is very helpful when mapping. I will mention the one that filters out all detail brushes.

By pressing ctrl+d all detail brushes will be filtered out. Look at the following two pictures, they are exactly the same as the two previous pictures but all detail brushes are filtered out. This is how you room should be when you compile it. (Remember that we are talking about brushes, you might have models or other entities in your room but they are not mentioned here because they don’t affect the VIS)

NO PICTURE AVAILABLE

This thing with detail brushes does not only apply to things inside buildings, you should keep the outside of buildings, ground etc clean. For example if you have made a sign outside with a warning for mines, then the brushes it is made of should be “detail” because these brushes will never block any views. That was just an example, I hope you get the picture how the map should look when detailed brushes are filtered out.

Here are some compiling facts from the pictures above. **Structural** are the results when compiling and every brush in the room were structural, **Detail** are the results when compiling and every brush inside the room were detail

Compiled map	Structural	Detail
Portalclusters	153	23
Numportals	457	42
Numfaces	522	117
Visdatasize	3680	192

As you can see it's a large difference between these maps, and remember this was just one single room, just imagine how it would have been in a large map. (The reason that the “detail” room had as many portalclusters as 23 was that it was not a square or rectangular room, but you didn't see that in the pictures)

A final word

By using detail brushes on things I have reduced VIS a lot, you could say that I have removed “useless” VIS. Remember that VIS is not a bad thing, it's just a matter of having the right (effective) VIS in your map. With the right VIS the engine won't render all parts of the map and that will speed up your map (if you don't understand why then go back and read from the beginning again).

This was a final word but not my last....this tutorial will be continued with some other things, like the use of area portals. Hopefully you, “the dummie” like me (just kidding) have gained a better understanding of what VIS is or does. The difficult part is the exactly understand how the splits in the “space” are done and how all the new “spaces” look (using the terms I started with in this tutorial).

Also remember that **only** structural brushes block view, not models, detail brushes, lod terrain entities like doors, etc. The compiler will just see through these things. This is important to know if you start building you map and plan on having an effective VIS.

To be continued.....

//Tutorial written by **TheStorm**, The Modding Theater, thestorm@modtheater.com

//Tutorial based on knowledge gained at www.modtheater.com, and mainly from Balr14 (thanks)

// Advanced box based on [Spog's technical](#)