Procedural Mapping Tools Manual

2022 06

Abstract

ProceduralMappingTools is a Houdini node library that implements a toolset for Source Engine 1, Unreal Engine 1 and IdTech4 level design (mapping).

Contents

1 Introduction to ProceduralMappingTools

1.1 Introduction

ProceduralMappingTools implements a Houdini pipeline aimed at the creation of large scale levels for Source Engine 1, Unreal Engine 1, and IdTech4. The node library is focused on environmental modeling, and is not designed for other functions such as level scripting.

It includes:

- Parsers used to interpret files of the target engine, including Source1(.fgd, .vmt), Unreal1(.uc), and IdTech4(.def, .mtr, .sndshd).
- A set of nodes for environmental modeling, used to specify geometry, materials and UVs for brushes, Source1 displacement maps, Unreal1 BSP Terrain and IdTech4 Patches.
- A set of nodes for entity placement(props, sounds, sprites, and lights).
- Level and static model exporters for Source1(.vmf, .smd/.qc), Unreal1(.t3d, .obj), and IdTech4(.map, .ase).

PMT is designed to run on Windows 10 with Houdini 19.0. The setup procedure involves extracting and converting assets from the target engine. This is done with third party tools and Python scripts.

In order to fully utilize the features of this toolset, the user should be familiar with VEX, Python and computational geometry.

1.2 An example node network

In general node networks will be shaped like a tree, with many branches that eventually converge into a root (the export node). The general flow of nodes is as follows:

1. Create geometry; use pmt_material_assign to set material and pmt_bsp_group to mark as BSP geometry.
2. Merge all geometry and run pmt_bsp_partition to prepare for export.
3. Run a coordinate conversion node (pmt_vmf_coord_houdini_to_vmf) to convert from Houdini to the target engine's coordinate system. In Houdini, Y+ is up by default while Z+ is up in Source, UE1, and IdTech4. PMT nodes generally assume Y+ is up.
4. Run UV nodes (pmt_material_uv_init, pmt_material_uv_align) to set BSP geometry UVs. Node that pmt_material_uv_align should only be run after converting coordinate systems.

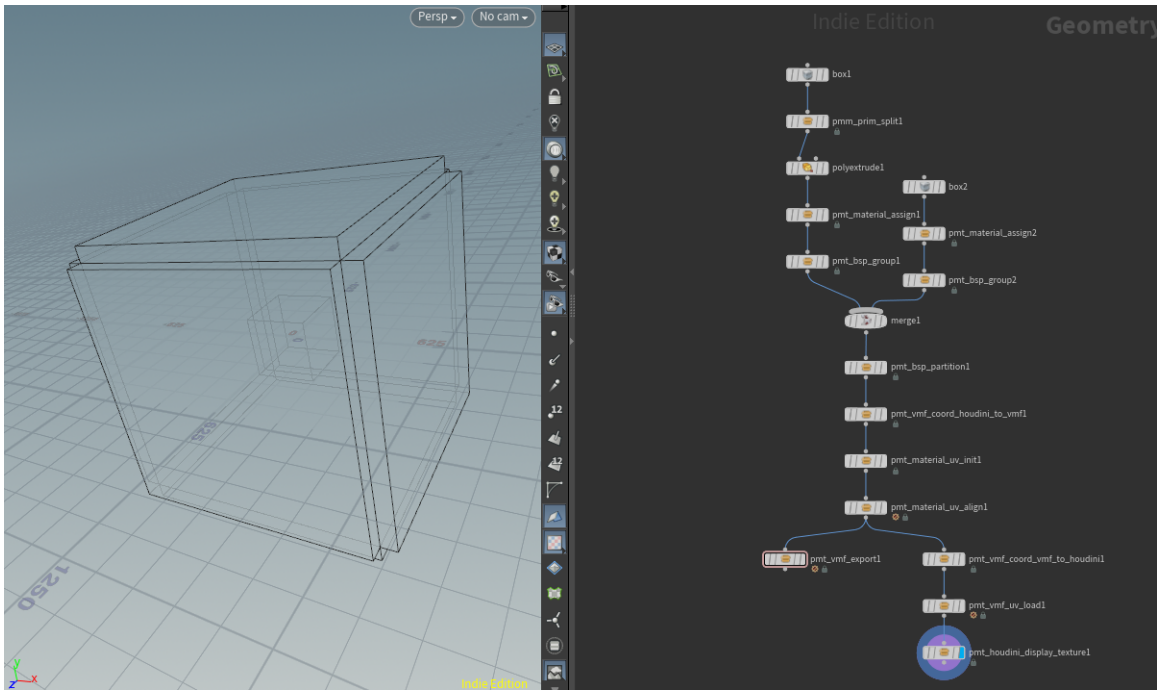


Figure 1: Example network.

5. Run an export node (pmt_vmf_export) node to write the level file to disk.

1.3 File structure

PMT should be installed to C:/pmt, with the node libraries(.hda/.hdalc) and Python panels stored at C:/pmt/hda. To install PMT to another directory, it is recommended to create symlinks using the 'mklink' command.

In C:/pmt:

- bin - contains third party tools used by PMT.
- classdefs - contains class definitions; .fgd for Source 1, .uc for Unreal Engine 1, and .def for IdTech4.
- exports - the default directory to save level files and static meshes to; .vmf for Source 1, .t3d for Unreal Engine 1, and .map for IdTech4.
- hda - contains the node/digial asset libraries.
- materials - contains .vmt (Source) and .mtr (IdTech4) material definition files; UE1 does not have a material system, and only uses diffuse textures.

- materialsets - contains lists of materials(.matlist.txt), each file is a list of materials with one material per line.
- models - contains .obj meshes that have been converted from the target engine's native model format.
- scripts - contains Python scripts which are loaded into the PMT nodes. Also contains Python scripts for setup.
- sounds - contains .wav (all engines) and .ogg (IdTech4) files.
- textures - contains .png (Source and IdTech4) and .bmp (UE1) diffuse textures converted from the target engine's native texture format. Specular, normal, and other maps are unused by PMT.

In C:/pmt/hda:

- ProceduralMappingTools.hdalc - contains nodes which are exclusively useful for BSP mapping, as well as nodes used to interface with the target engine(pmt, pme, pmi nodes).
- ProceduralMappingTools_modeler.hdalc - contains nodes which may have uses outside of BSP mapping (pma, pmg, pmm nodes).
- ProceduralMappingTools_globals.hdalc - is used to contain the PMT globals node(pmt__global_config), which contains the database for textures, materials, models, sounds, and class definitions. Since it is very slow to load(and save), it is put in a separate file.
- ProceduralMappingTools_gpl.hdalc - an optional file that contains the node (pmi_compute_map_UV_axes_GPL), which is only needed for UV mapping with IdTech4. As implied by the name, the node in this library is GPL licensed.

1.4 Node overview

All nodes are in the pmt:: namespace. Note that 'higher-level' nodes, starting with pma_ or pmg_ should be regarded as examples or templates that can be modified.

1.4.1 pmt

Nodes beginning wiht pmt_ handle core functionality such as:

- Export (and limited import)
- Nodes to mark BSP geometry and special geo (such as displacement maps, bspterrain and patches)

- Material and UV assignment
- Coordinate system conversions

1.4.2 pma 'asset'

'high level' modeling nodes that generally take a guide geometry to produce a complex output. An example would be a node that takes a floorplan and outputs a building.

1.4.3 pmg 'generator'

'mid level' modeling nodes that generally take a guide geometry to produce a intermediate output. An example would be a node that takes a square and outputs a window.

1.4.4 pmm 'modeler'

'lower level' modeling nodes, which focus on directly manipulating vertices and prims.

1.4.5 pme 'entity'

Nodes for configuring point entities, including models, lights, sprites and other entities.

1.4.6 pmi 'internal'

Internal nodes; these should generally not be used.

2 Setup procedure

Setup involves extracting assets and converting into a format that Houdini can read using Python scripts and third-party tools. For textures .png (Source 1 / IdTech4) and .bmp (Unreal Engine 1) are used while meshes need to be converted into .obj.

All third-party dependencies are stored in /pmt/bin. Python setup scripts (.py) and batch scripts(.cmd) are stored in /pmt/scripts/setup/.

Once assets have been setup, PMT can be used by adding all .hda(hdalc) in /pmt/hda to a Houdini project.

2.1 Shared Third-party tools/dependencies

- Assimp - for converting model data into .obj
  https://github.com/assimp/assimp/releases/tag/v4.1.0
  https://github.com/assimp/assimp/releases/download/v4.1.0/assimp-sdk-4.1.0-setup.exe
- ImageMagick - for image analysis and comparison; also for converting IdTech4 textures(.dds) files to .png
  http://www.imagemagick.org
- Python 3.7 - for asset conversion and setup scripts
  https://www.python.org

2.2 Source Engine 1

For Source 1, the relative paths of textures, materials, meshes, and sounds must be maintained. The paths of .fgd files do not matter.

2.2.1 Third-party tools/dependencies

- Crowbar - for unpacking .vpk files and decompiling .mdl files.
  https://steamcommunity.com/groups/CrowbarTool
  https://github.com/ZeqMacaw/Crowbar/releases
- vtfedit - for converting .vtf textures to .png.
  https://web.archive.org/web/20170913055549/http://nemesis.thewavelength.net/index.php?c=238#

For each 'source project' folder, assets must be extracted and merged in the order that they are loaded. For example, with a default SourceSDK2013 installation, it is necessary to extract and merge in the order project_folder →ep2 →episodic →hl2.

- Using Crowbar, extract .vtf, .vmt, .mdl, .wav and /scripts folder from all .vpk(s)
- Extract materials .vmt and copy into /pmt/materials/vmf/.
- Extract and convert textures .vtf to .png, using vtfedit, and copy into /pmt/textures/vmf/.
- Extract and convert models .mdl to .smd to .obj
  - Using Crowbar, extract .mdl files, then decompile into .qc and .smd files.
  - Use vmf_rename_qc-smd.py script to rename .smd files (.mdl.ref.smd for visual meshes and .mdl.phy.smd for collision meshes).
  - Copy meshes into /pmt/models/vmf/.
  - Run vmf_assimp_smd_to_obj.py to convert from .smd to .obj.
- Copy .fgd definitions into /pmt/classdefs/vmf/.
- Extract .wav sound and scripts and copy into /pmt/sounds/vmf/.
  - .wav files should be placed in /pmt/sounds/vmf/sounds.
  - sound scripts should be placed in /pmt/sounds/vmf/scripts. This includes all .txt files in /scripts (the source engine project directory, not PMT directory) beginning with game_sounds_, level_sounds_, npc_sounds_ and soundscapes.
- Run material conversion script vmf_mapspecific_prop_material_convert.py for map-specific props. Copy /pmt/materials/vmf/materials/_msp to the source project directory if MSPs are used.

Example paths:

- If vmf_materials_path is c:/pmt/materials/vmf/materials, tools/toolsskybox should be at c:/pmt/materials/vmf/materials/tools/toolsskybox.vmt
- If vmf_textures_path is c:/pmt/textures/vmf/materials, tools/toolsskybox should be at c:/pmt/textures/vmf/materials/tools/toolsskybox.png
- If vmf_models_path is c:/pmt/models/vmf/, error.mdl should be at c:/pmt/models/vmf/models/error.mdl.ref.smd.obj
- If vmf_sounds_path is c:/pmt/sounds/vmf/, soundscapes.txt should be at c:/pmt/sounds/vmf/scripts/soundscapes.txt, and ui/buttonrollover.wav should be at c:/pmt/sounds/vmf/sound/ui/buttonrollover.wav

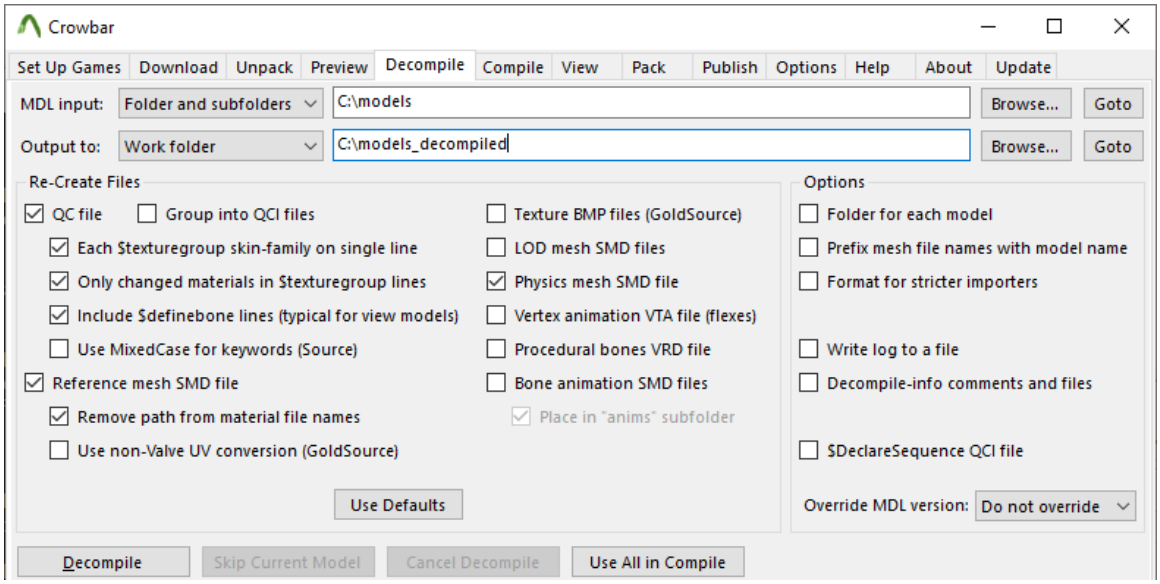


Figure 2: Settings for decompiling .mdl with Crowbar.

2.3 Unreal Engine 1

For UE1, the relative paths of textures, meshes, and sounds must be maintained. The paths of .uc files do not matter. These instructions are for Unreal 227i(UnrealEd 2.1) or UT469b.

Running 'Batchexport' and 'Export all scripts' commands from UnrealEd will export the data to the root UE1 directory (containing /Maps, /System, and /Textures) into a folder with the same name as the package. For example, if the UnrealEd.exe path is C:/Unreal/System/UnrealEd.exe and 'Batchexport Vertex Meshes from Package(.3d)' is run on the 'UnrealI' package, then the meshes will be exported into C:/Unreal/UnrealI.

Textures, sounds, and classes can be exported by running a batch script or manually. Batch export for meshes is not supported, so it must be done manually.

1. Batchscript method: Copy ue1_ucc_batchexport.cmd from /pmt/scripts/setup to the UE1 root directory (containing /Maps, /System, and /Textures) and run it. Following the above example, the path of the batch script should be C:/Unreal/ue1_ucc_batchexport.cmd.
2. Manual method, using UnrealEd:
    - Extract textures .bmp: for each .utx, in 'Textures' window, run 'Batchexport to BMP from package' and copy into /pmt/textures/t3d/.
    - Extract class definitions .uc: in 'Actor classes' window, run 'Export all scripts' and copy into /pmt/classdefs/t3d/.
    - Extract .wav: for each .uax, in 'Sounds' window, run 'Batchexport to WAV from package' and copy into /pmt/sounds/t3d/.

Next, extract and convert meshes:

- For each package in 'Meshes' window, run 'Batchexport Vertex Meshes from Package(.3d)'.
- Copy the result into /pmt/models/t3d/.
- Run t3d_assimp_3d_to_obj.py to convert from .3d to .obj.

Example paths:

- If t3d_textures_path is c:/pmt/textures/t3d/,
  detail.marble(Detail.utx) should be at
  c:/pmt/textures/t3d/Detail/Textures/Marble.bmp
- If t3d_models_path is c:/pmt/models/t3d/,
  UnrealShare.BarrelM(UnrealShare.u) should be at
  c:/pmt/models/t3d/UnrealShare/models/BarrelM_d.3d.obj

- If t3d_sounds_path is c:/pmt/sounds/t3d/, Activates.Beeps.mactiv10(Activates.uax) should be at c:/pmt/sounds/t3d/Activates/Sounds/Beeps.mactiv10.wav or c:/pmt/sounds/t3d/Activates/Sounds/Beeps/mactiv10.wav (if manually extracted)

2.4 IdTech 4

For IdTech4, the relative paths of textures, meshes, and sounds must be maintained. The paths of material(.mtr), soundshader(.sndshd), and class definition(.def) files do not matter.

Copy all .pk4 files and extract them into a single directory(.pk4 is the same format as .zip). From the directory with extracted files:

- Copy materials .mtr files from /materials into /pmt/materials/map/.
- Copy all folders with images(such as .dds, .tga, .jpg, or .png) into /pmt/textures/map/. At a minimum the /textures folder should be copied.
- Convert all textures in /pmt/textures/map/, including .dds, .tga, and .jpg, into .png; run imagemagick_mogrify_tga-jpg-dds_to_png.cmd.
- If there is a /dds folder at /pmt/textures/map/dds, it should be extracted into /pmt/textures/map/. For example, if there is a folder /pmt/textures/map/dds/models/, then it should be merged into /pmt/textures/map/models/.
- Move extracted /models folder, containing .ase, .lwo, and .md5mesh models, to /pmt/models/map/models.
- Run map_assimp_lwo_ase_md5mesh_to_obj.py to convert to .obj. Note that .md5mesh to .obj conversion is not working, and is currently unsupported.
- Copy class definitions .def in /def into /pmt/classdefs/map/.
- Move extracted /sound folder, containing .wav and .ogg sounds and .sndshd soundshaders, to /pmt/sounds/map/sound.
- To preview .ogg sounds in the editor, install opencodecs_0.85.17777.exe from
  https://xiph.org/dshow/downloads/
  (we use Qt.QMediaPlayer, which uses directshow on Windows to play sounds)

Example paths:

- If map_textures_path is c:/pmt/textures/map/,
  textures/common/caulk should be at
  c:/pmt/textures/map/textures/common/caulk.png
- If map_models_path is c:/pmt/models/map/,
  the folder 'models/mapobjects' should be at
  c:/pmt/models/map/models/mapobjects
- If map_sounds_path is c:/pmt/sounds/map/,
  the folder 'sound/ambient' should be at
  c:/pmt/sounds/map/sound/ambient/

2.5 Additional Setup Steps (all engines)

- (This step is only necessary in order to use pmt_material_shuffle 9.8.) Run image_size.py and image_similar.py scripts in /pmt/scripts/analyze to perform analysis on textures for search features. For each .png in /textures, this generates an .analyze.txt file containing the results. The image_similar.py scripts may take several hours to run.

3 Working with BSP Geometry

Although it is possible to create complex geometry in Houdini, that does not mean that the result will work in the target engine. It is recommended to keep brushes as simple as possible to avoid leaks and other BSP compile errors.

It is a good idea to generally avoid using booleans and convex decomposition (except for 2d meshes), as these methods often produce invalid brushes. Even when working with 2d meshes care should be taken, as booleans and convex decomposition tend to produce colinear points and zero-length edges.

When starting a level, the first action should be to clearly define the map area and to create a set of brushes that seals the map from the void. At the scale we are working at, it is extremely difficult to locate leaks, so the sealing brushes should be considered permanent and should never be changed. An alternate approach is to define a floorplan that encompasses the entire map and then to use an algorithm to generate sealing brushes.

3.1 Scale

PMT is designed to run at the scale of the target engines. This means that 64 units is slightly wider than the width of a single door and 128 units is approximately 1 story. Float tolerances are setup around a minimum scale of 1 unit, so using smaller scales can cause PMT nodes to fail.

Note also that Houdini is setup around a scale of 1 unit == 1 meter, meaning that defaults are often too low and higher values should be used. In some cases, such as the 'remesh' node, the values are very low and can cause Houdini to hang. This can be avoided by creating the node first and increasing the scale before connecting it.

3.2 Brushes

A brush is a convex polytope that can be specified in 2 ways:

- As the result of a series of plane(halfspace) intersections(used in .vmf and .map formats).
- As a set of polygons composed of vertices(used in .t3d format).

Brush geometry requirements:

- Each brush must be convex. An intuitive definition of convex is that a line can be drawn between any 2 points in the brush without exiting the mesh.
- Each brush must be sealed. It must have no holes and should have a clearly defined inside and outside.

- The faces of each brush should have no interior edges.
- The prims of each brush should face outwards; the normals of each face should be pointing away from the center of the brush.
- The points of each prim should lie in the same plane.
- The points of each brush should be shared by prims of the same brush.

In addition, each face of the brush has a material and UV:

- In Source and UE1, this includes a u-axis, v-axis, scale and offset.
- In IdTech4, this includes a rotation, scale, and offset.

When extruding a 2d polygon into a 3d brush, keep in mind that:

- Colinear points will create faces with interior edges, resulting in invalid brushes.
- Each resulting brush must be convex. The 'divide' node and pmm_prim_split can be used to split a 2d polygon into separate convex pieces.

3.3 Engine Limitations

Source Engine 1 limitations:

- All points must be in +/-16384 units (32768 units total).
- Displacement maps must be created from quads.

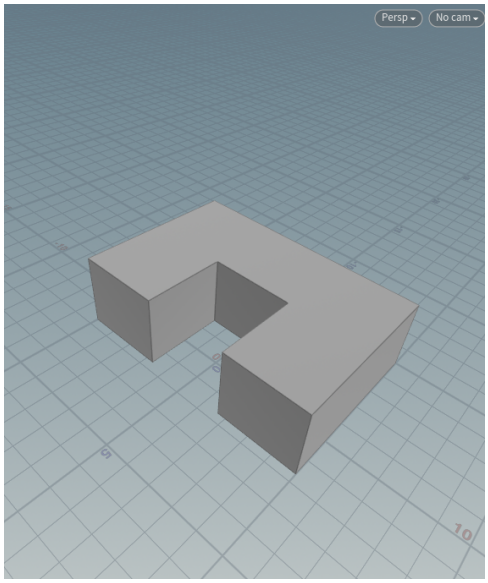
Unreal Engine 1 limitations:

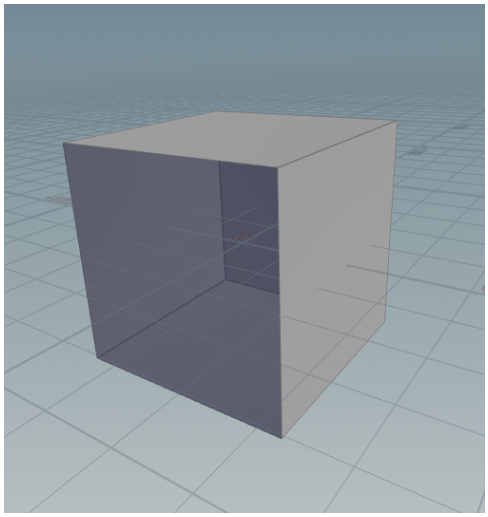
- All points must be in +/-32768 units (65536 units total).
- BSP terrain brushes must be all triangles; non-trianglar meshes can be imported by UnrealEd, but frequently result in BSP holes.

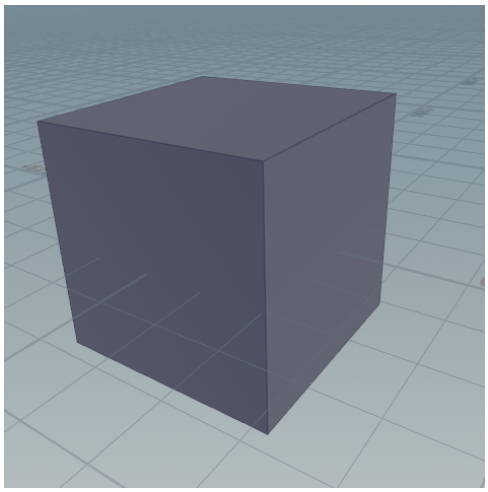
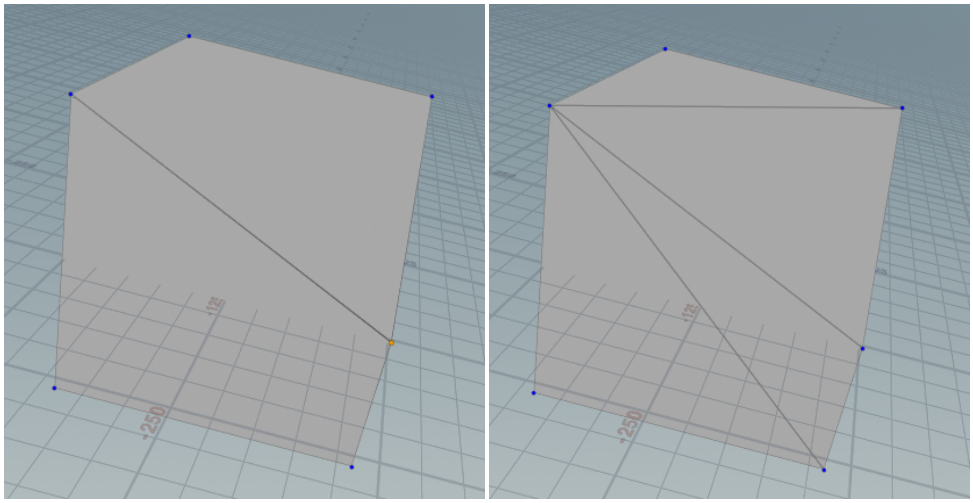
IdTech 4 limitations:

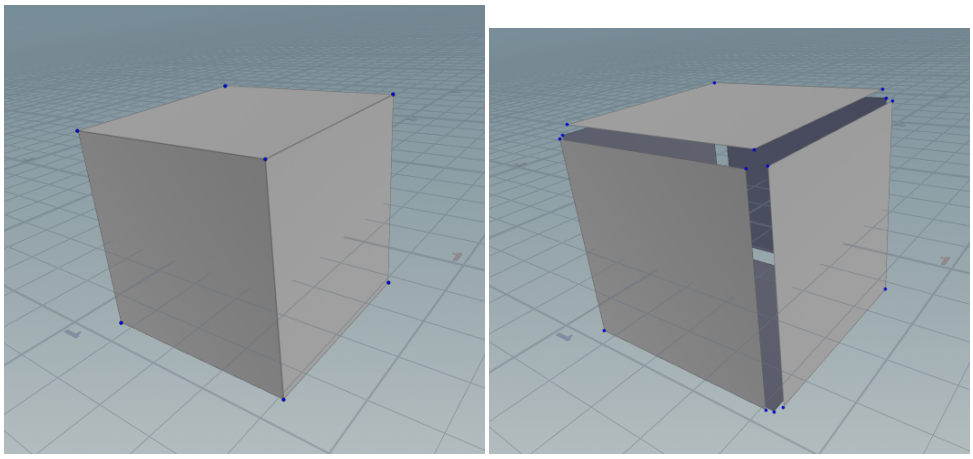
- All points should be in +/-32768 units. Although the actual limit is +/-65536 units (131072 units total), brushes may fail to appear at points far from the origin.

3.4 Examples of invalid geometry

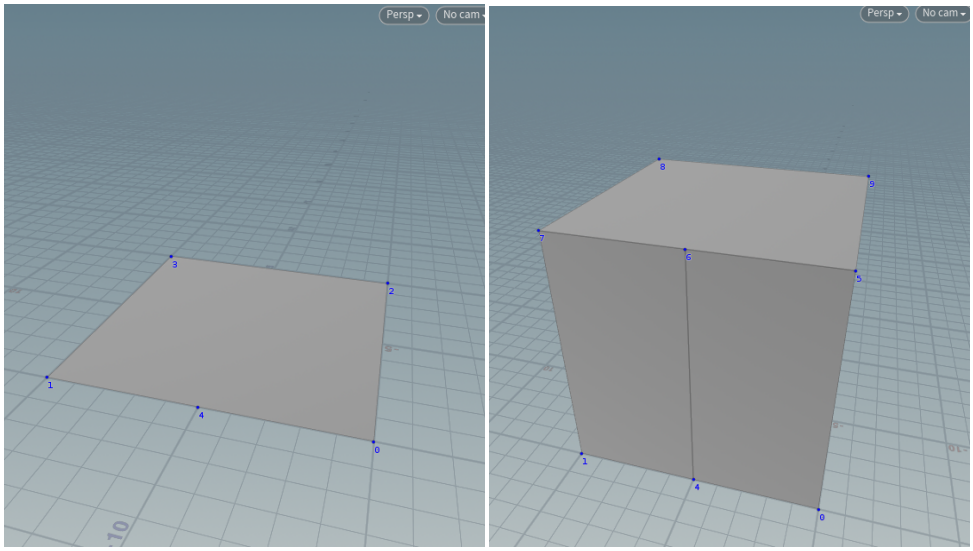
- 

- This brush is invalid as it is not convex.
- 
- This brush is invalid as it is not sealed.

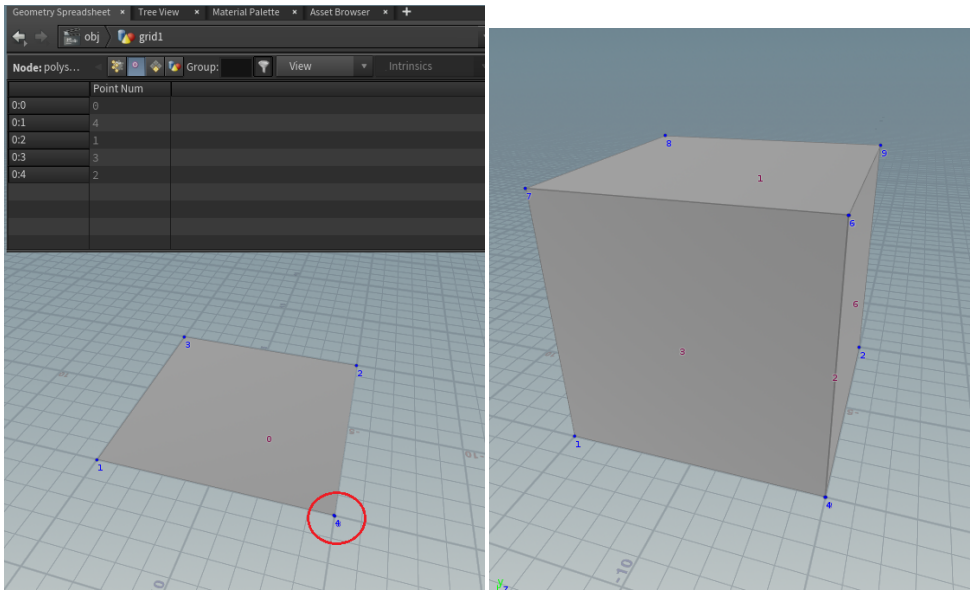
- 
- The normals of each prim on this brush is facing inwards. As a result, there is no volume that is inside all prims(halfspaces).
- 
- The points of the top face of this brush are not coplanar. When loaded by the target engine, the shape of this brush might not be the same.

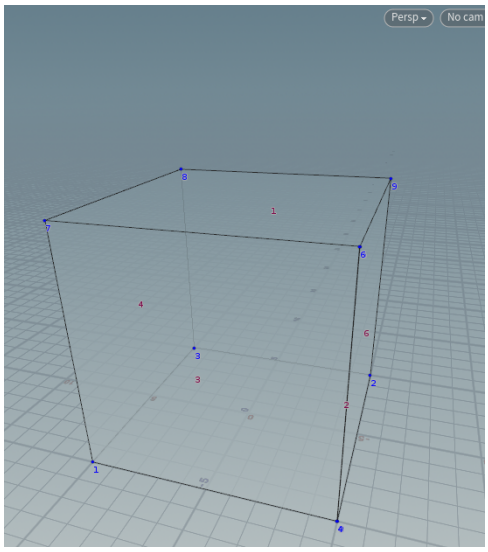
- 

- The left brush does not appear to have any issues, but does not share vertices between prims. When exploded with the 'exploded view' node, the problem becomes clear.

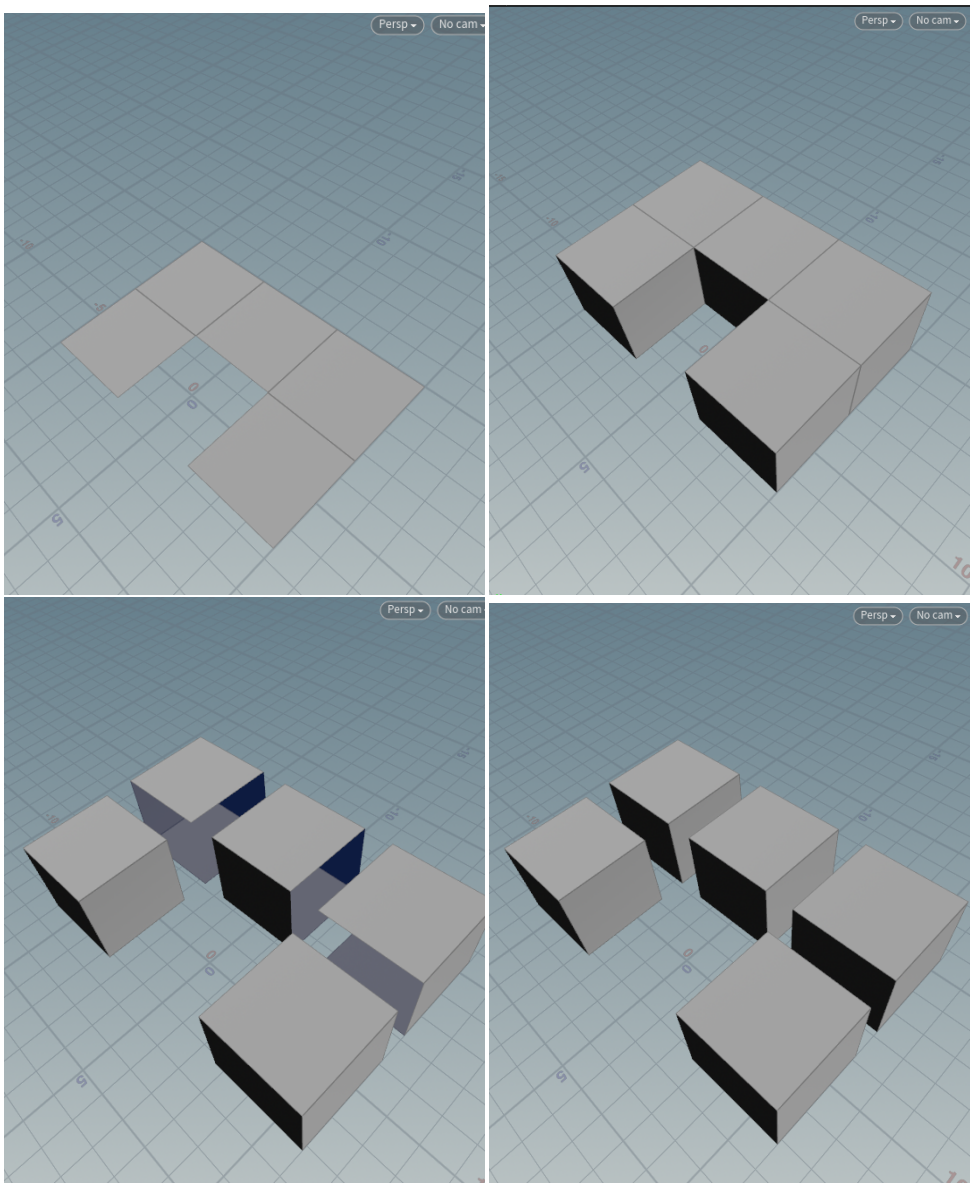
- 

- A 2d polygon with a colinear point(4) is extruded. This results in an invalid brush; brush faces should not have interior edges.

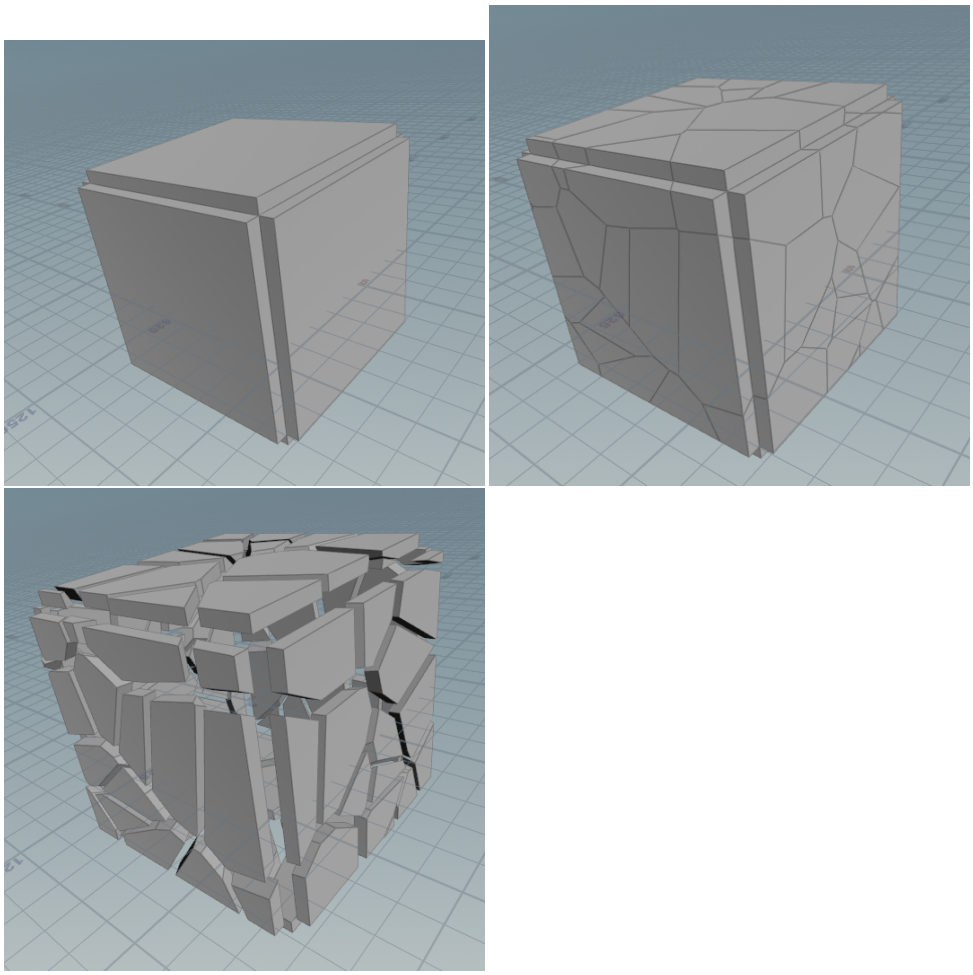
- 

- 

- The presence of colinear points is not always obvious. The 2d polygon above seems to have 4 points at first glance but it actually has 5. When extruded, this results in an invalid brush; note the presence of an extra prim(2).

- 

- The above 2d polygon was extruded using the 'PolyExtrude' node. At first glance(top-right) it appears to be correct, but each of the 5 brushes is not closed(bottom-left). This is since the PolyExtrude node 'Divide into' parm was set to 'Connected Components'. To get the correct result(bottom-right), PolyExtrude should be set to 'Individual Elements' or the prims should be split(pmm_prim_split) first.

- 

- In this example, 6 brushes are split into pieces using the 'Voronoi Fracture' node. Strictly speaking there should be no issues with this as each piece is closed and convex. However, the complexity of the geometry means that not all brushes might appear when loaded in the level editor, and there will likely be issues when compiling. Further, this set of brushes is likely to leak if used as sealing geometry.

3.4.1 Debugging

When dealing with invalid brushes and BSP compile errors, it might be useful to note:

- pmm_null_switch is useful for debugging using process of elimination.
- The 'exploded view' node can be used to inspect individual brushes.
- Each engine's level editor has its own data sanitization routines, so simply loading a level in Hammer, UnrealEd, or DarkRadiant/GtkRadiant and saving it can fix invalid geometry and stop leaks. However, if the geometry is sufficiently degenerate this may remove sealing brushes resulting in leaks.

4 Feature Notes

4.1 Map-specific Props

As implied by the name, a 'map-specific prop' (MSP) is a static mesh that is intended to be used on a single map. This can be useful in cases where a map exceeds the brush, plane, or vertex limits. Another use case is for handling complex geometry that cannot be expressed with brushes. It is recommended to use MSPs whenever possible as they are not as limited as brushes and the geometry does not need to be as robust.

4.1.1 Source

In Source 1, it corresponds to a prop_static. The Python script /pmt/scripts/setup/vmf_mapspecific_prop_material_convert.py must be run before MSPs can be used, as .vmt files that are used for brushes are not compatible with models in Source. In general, the script duplicates materials using the 'LightmappedGeneric' shader, which can only be used by brush geometry, to 'VertexLitGeneric', which is usable by props. Additionally, the resulting materials in /pmt/materials/vmf/_msp must be copied to the target project directory.

4.1.2 Unreal Engine 1

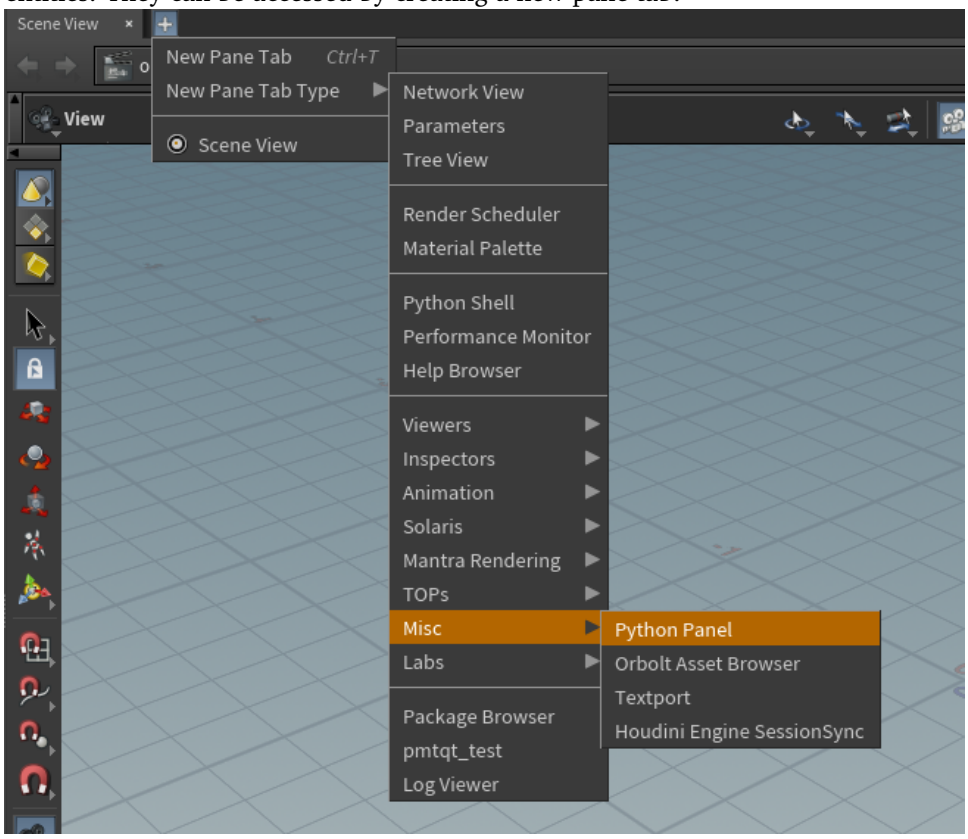
In UE1, it uses static meshes, which are a Unreal 227i+ specific feature. The MSPs are exported as .obj files which can be imported with UnrealEd.

4.1.3 IdTech 4

In IdTech4, a MSP is a func_static .ase that is exported to /models/msp.

4.2 Python Panels

PMT includes a number of Python panels for selecting materials and configuring entities. They can be accessed by creating a new pane tab:



4.3 pmt_qt_entity_kv_editor

The Entity-Keyvalue(kv) editor is used to adjust entity keyvalues.

4.4 pmt_qt_material_selector

The material selector is used to select materials for pmt_material_assign.

4.5 pmt_qt_materialsets_editor

The materialsets editor is used to select groups of materials 'materialsets' that can be batch assigned to geometry using pmt_materialsets_assign.

4.6 Materialsets

Materialsets are used to batch assign materials to prims. A materialset comprises a list of materials that are divided by 2 levels. The first level is the 'style' and the second is the 'category'. Each style contains a number of categories, which each contain a list of materials.

For instance:

- Category 'A'
  - Style '0'
    - brick_a
    - brick_b
    - brick_b2
  - Style '1'
    - wood
    - wood2
  - Style '2'
    - ...
  - ...
- Category 'B'
  - Style '0'
    - brick0
    - brick0a
    - brick3
  - ...

5 Attributes and Groups

Attributes used to store entity and brush data can be manipulated with attribute wrangle nodes. In PMT only point and prim attribs are used to store data. That is, all modeling occurs at the geometry level in a SOP(surface operator) node.

In the following, $ENGINE refers to 'vmf'(Source), 't3d'(UE1), or 'map'(IdTech4). Additionally, VEX notation is used to indicate attrib type. (i@ is int, f@ is float, u@ is float2, v@ is float3, s@ is string, d@ is dict; see VEX documentation for other types.)

5.1 Entity point and prim attribs

- s@pmt_$ENGINE_entity_class - indicates the class of the brush or point entity. It is ignored if the value is empty or set to 'pmt_none'. Used by both brush and point entities.
- d@pmt_$ENGINE_entity_keyvalues - stores the key-value pairs of the entity. Each key and value is a string. Used by both brush and point entities.
- v@pmt_$ENGINE_rotation_euler - stores the rotation about each axis(x, y, z) in degrees. Point entities only.
- i@pmt_has_euler_rotation - set to 1 to indicate that a entity has a rotation, 0 otherwise. Point entities only.

5.2 Brush and brush entity prim attribs

- i@pmt_bsp_island - indicates the brush that prim belongs to. This is computed by pmt_bsp_partition.
- i@pmt_entity_island - indicates the brush entity that prim belongs to. A value is set even for non-entity brushes. This is computed by pmt_bsp_partition.

5.3 Brush UV prim attribs

- s@pmt_$ENGINE_material - stores the material of the face
- u@pmt_$ENGINE_texture_size - size in pixels of the diffuse texture.
- s@pmt_$ENGINE_texture_path - the current filesystem path of the diffuse texture.
- v@pmt_vmf_uv_u_axis - normalized vector indicating the direction of the UV u axis.
- v@pmt_vmf_uv_v_axis - normalized vector indicating the direction of the UV v axis.

- u@pmt_vmf_uv_scale - scale of the texture; multiplying this value by 2.0 doubles the texture size while multiplying by 0.5 halves.
- u@pmt_vmf_uv_offset - texture shift in pixels.
- v@pmt_t3d_uv_u_axis - same as Source 1(vmf) attrib; converted to UE1 values on export.
- v@pmt_t3d_uv_v_axis - same as Source 1(vmf) attrib; converted to UE1 values on export.
- u@pmt_t3d_uv_scale - same as Source 1(vmf) attrib; converted to UE1 values on export.
- u@pmt_t3d_uv_offset - same as Source 1(vmf) attrib; converted to UE1 values on export.
- u@pmt_map_uv_scale - same as Source 1(vmf) attrib; converted to IdTech4 values on export.
- u@pmt_map_uv_offset - same as Source 1(vmf) attrib; converted to IdTech4 values on export.
- f@pmt_map_uv_rotation_degrees

5.4 Vertex attribs

- v@uv - UV coordinates for map-specific props and IdTech4 patches.

6 Per Node Documentation (pmt_vmf_)

6.1 pmt_vmf_uv_load

Computes uv coordinates per vertex for visualization; this is not needed to export the level. The equation in 'calculating texture coordinates' from [1] is used to compute the UVs.

6.2 pmt_vmf_export

Exports a .vmf file.

6.3 pmt_vmf_import

(work in progress) Imports a .vmf file.

6.4 pmt_vmf_coord_vmf_to_houdini

Converts coordinate system from Source 1(vmf) to Houdini.

6.5 pmt_vmf_coord_houdini_to_vmf

Converts coordinate system from Houdini to Source 1(vmf).

6.6 pmt_vmf_msp_create

Creates a 'map-specific' prop.

6.7 pmt_vmf_msp_export

Exports 'map-specific' props .smd/.qc files that can be compiled with studiomdl.exe.

6.8 pmt_vmf_dispmap_create

Creates a displacement map from a quad mesh. Outputs a low resolution and high resolution mesh; only the vertices of the high resolution mesh can be moved. Both outputs are passed into pmt_vmf_dispmap_transfer_hi_lo to create the displacement map geo that is actually exported.

6.9 pmt_vmf_dispmap_transfer_hi_lo

Combines a low-resolution and high-resolution mesh created by pmt_vmf_dispmap_create to create a 'brush' that can be exported.

7 Per Node Documentation (pmt_t3d_)

7.1 pmt_t3d_uv_load

Computes uv coordinates per vertex for visualization; this is not needed to export the level.

7.2 pmt_t3d_export

Exports a .t3d file.

7.3 pmt_t3d_import

(work in progress) imports a .t3d file.

7.4 pmt_t3d_coord_t3d_to_houdini

converts coordinate system from Unreal Engine 1(t3d) to Houdini.

7.5 pmt_t3d_coord_houdini_to_t3d

converts coordinate system from Houdini to Unreal Engine 1(t3d).

7.6 pmt_t3d_msp_create

Creates 'map-specific props', which are StaticMeshActor(s).

7.7 pmt_t3d_msp_export

Exports 'map-specific props' so they can be loaded by UnrealEd.exe.

7.8 pmt_t3d_polyflags

Used to configure polyflags, which are a per-face option of UE1 brushes.

8 Per Node Documentation (pmt_map_)

8.1 pmt_map_uv_load

Computes uv coordinates per vertex for visualization; this is not needed to export the level.

8.2 pmt_map_patchdef_reverse_winding

Reverses a patch, so that it faces the other way.

8.3 pmt_map_patchdef_polyquad_to_bezier

Converts a mesh created with pmt_map_patchdef_polyquad_create into an actual bezier surface.

8.4 pmt_map_patchdef_polyquad_create

Subdivides a quad mesh to create patches.

8.5 pmt_map_patchdef_group

Marks a mesh as a patch so it can be exported.

8.6 pmt_map_export

Exports a .map file.

8.7 pmt_map_import

(work in progress) imports a .map file.

8.8 pmt_map_coord_map_to_houdini

converts coordinate system from IdTech 4(map) to Houdini.

8.9 pmt_map_coord_houdini_to_map

converts coordinate system from Houdini to IdTech 4(map).

9 Per Node Documentation (pmt_)

9.1 pmt_model_delete_visual_points

Deletes visual meshes of entities, typically loaded by pme_$ENGINE_model nodes.

9.2 pmt_bsp_shell

For Source1 and Idtech4, wraps the input geometry with a box of 6 brushes that seals the level. For Unreal Engine 1, creates a single subtractive box brush that encompasses the input mesh.

9.3 pmt_bsp_partition

Using the connectivity between prims, assigns sets of prims to brushes and brush entities. This must be run before the export step.

9.4 pmt_bsp_group

Marks a mesh as part of a brush or brush entity.

All engines:

- bsp: the brush is a regular additive brush used for computing visibility.
- bspdetail: the brush is a 'detail' brush; this is a func_detail in Source, a semi-solid brush in UE1, and a func_static in IdTech4.
- bspentity: indicates that the brush is part of a brush entity.

Unreal Engine 1 only:

- bspnonsolid: the brush is non-solid, meaning that actors can pass through it.
- bspsubtract: by default subtractive brushes are ordered first.
- bspterrain: terrain brushes are closed triangle meshes; unlike other brushes, they do not have to be convex.
- bspterraindetail and bspterrainsubtract: terrain variants of detail and subtractive brush types.

9.5 pmt_bsp_entity_link

Connects brushes to indicate that they are part of the same entity. For Source 1 and IdTech4 only.

9.6 pmt_material_assign

Assigns a material to prims. The path of the material is stored in the prim attrim s@pmt_$ENGINE_material.

9.7 pmt_material_assign_with_search

Uses a search query to assign a material to prims. This is a highly experimental node and only supports 2 queries:

- path: Searches for a keyword in the path of the material.
- name: Searches for a keyword in the name of the material.

9.8 pmt_material_shuffle

Switchs materials on a prim using absolute similarity which is computed by comparing images with ImageMagick(the comparison is done in an offline step by running scripts in /pmt/scripts/analyze). 'absolute similarity' is simply the fraction of pixels that match between 2 images. For each material, its diffuse map is compared with other materials in the same folder. If the proportion of pixels of the other material that match exceeds a user-specified threshold from 0 to 1 then it is considered for shuffling to.

9.9 pmt_materialset_init

Assigns a materialset to a set of prims. A materialset is a list of materials that can be used to batch texture a set of brushes. For instance, a materialset might contain a list of brick wall materials that are later assigned to a set of buildings, so that each building gets a different material.

9.10 pmt_materialset_assign

Assigns materials to prims using materialsets.

9.11 pmt_material_uv_transform

Scales, translates, or rotates per-prim UV coordinates.

9.12 pmt_material_uv_policy

Sets UV parameters that can later be computed using pmt_material_uv_align.

9.13 pmt_material_uv_init

Initializes per-prim UV attribs for brushes.

9.14 pmt_material_uv_align

Computes UV coordinates for brushes. The UVs are computed in brush format, which handles UVs per prim instead of per vertex. As a result, the UVs computed with this node are not immediately visible. pmt_vmf_uv_load, pmt_t3d_uv_load, or pmt_map_uv_load can be used to visualize the UVs.

- realign_axis: if not 'no_realign', aligns the specified axis using 'realign_axis_policy'.
- realign_axis_policy: if not 'no_realign', specifies how to align the UV axes per prim.
  - btPlaneSpace: uses prim's normal to compute U and V axes.
  - longest_edge: realign_axis will match the longest_edge of the prim.
  - most_vertical_y: realign_axis will match the edge with the greatest y magnitude.
  - increasing_point_attrib: realign_axis will match the edge where the attrib in 'realign_axis_attrib' is increasing.
- realign_axis_attrib: only used if 'realign_axis_policy' is set to 'increasing_point_attrib'. It is the name of the point attrib
- align_u, align_v: if not 'no_align', computes a UV offset.
- scale_u, scale_v: if checked, scales the UVs to 'target_scale'. to check. The 'merge_adjacent_faces' option will consider multiple prims as a single face, so that they all share the same UV parms.

9.15 pmt_houdini_display_texture

Loads diffuse textures from materials for visualization in Houdini.

9.16 pmt_validate_poly2d

Checks for zero-area prims, which will cause the BSP compilers to fail if exported.

9.17 pmt_validate_brushes

Checks for zero-area or zero-volume brushes.

10 Per Node Documentation (pmm)

10.1 pmm_sidefx_gamedev_dissolve_flat_edges

From the SideFX gamedev toolset (now SideFX Labs). Removes interior edges based on the angle between prims.

10.2 pmm_sidefx_gamedev_axis_align

From the SideFX gamedev toolset (now SideFX Labs). Aligns a set of geometry with an axis. Since the coordinates of each engine are limited in range, it is recommended to run this node on all geometry before the export step.

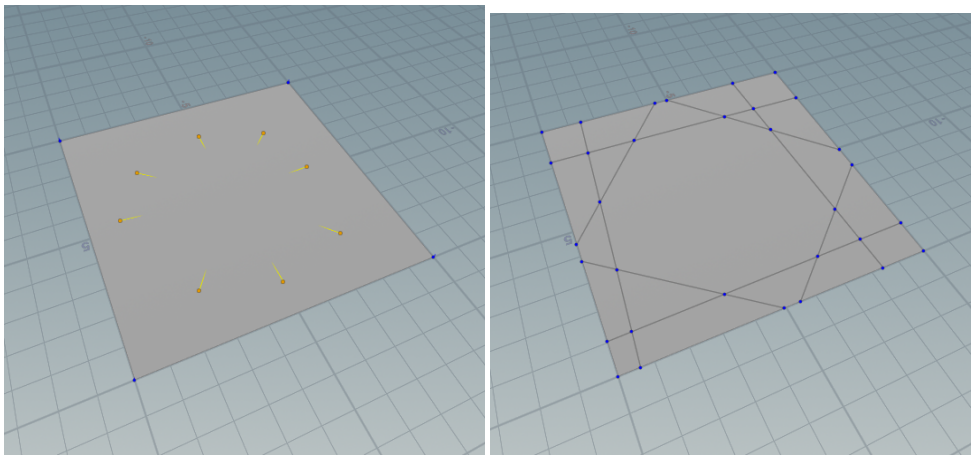
10.3 pmm_project_prim_vertices_coplanar

Attempts to move the vertices of a prim so that they all lie on the same plane.

10.4 pmm_null_switch

A convienence node that simply blocks anything that is passed into it. This can be used to exclude geometry when debugging or for performance reasons.

10.5 pmm_multiknife



Similar to the 'knife' node, but it takes multiple points instead of using parms. In the left image above, input0 is the square, input1 are the yellow points. The result is shown on the right.

- Input0) is the mesh to split.
- Input1) are knife points; each point has a position, v@P, and normal, v@N, and represents a cutting plane.

10.6 pmm_bsp_knife

Spits a convex brush with a plane, similar to the 'knife' node.

10.7 pmm_bsp_carve

Performs a 'carve' operation between 2 brushes.

10.8 pmm_bsp_multiknife

Clips a convex brush with multiple planes. See also pmm_multiknife.

10.9 pmm_prim_split

Splits all prims on the input mesh, creating separate copies of edges and vertices/points for each prim.

10.10 pmm_quad_coordinate_frame

Computes a local (per-prim) coordinate frame.

10.11 pmm_quad_create_point

Creates a single point using the coordinate frame established by pmm_quad_coordinate_frame.

10.12 pmm_multiknife_along_axis

Splits a mesh along a single axis.

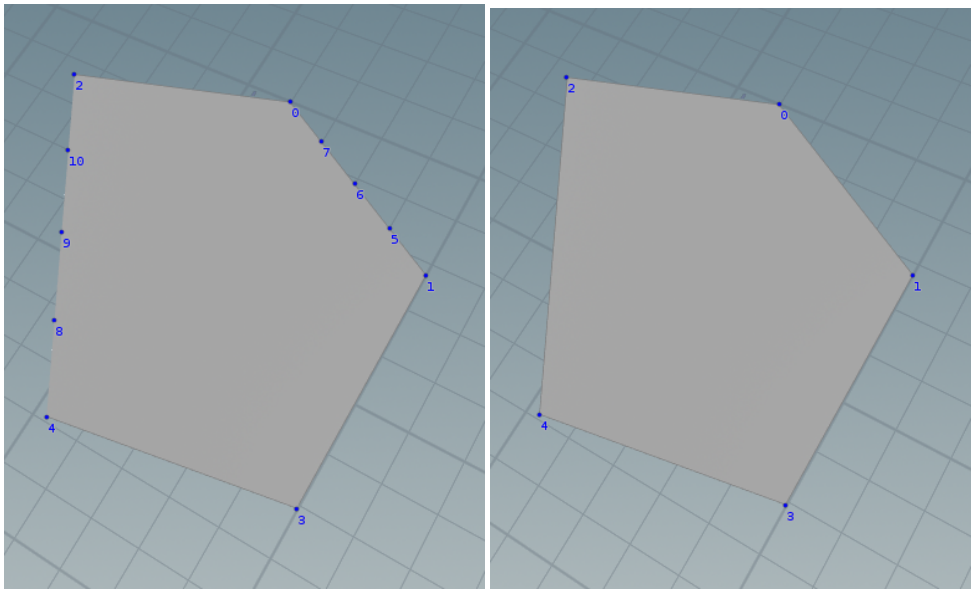
10.13 pmm_measure_along_axis

For each prim, measures the length of the prim along that axis and computes an index based on the position of the prim's centroid.

10.14 pmm_prim_group_interior_exterior_edges

Assigns edges of the input mesh to 'interior_edges' or 'exterior_edges' edge groups. An edge is interior if it is shared by 2 or more prims and exterior if connected to a single prim.

10.15 pmm_2d_dissolve_colinear_points



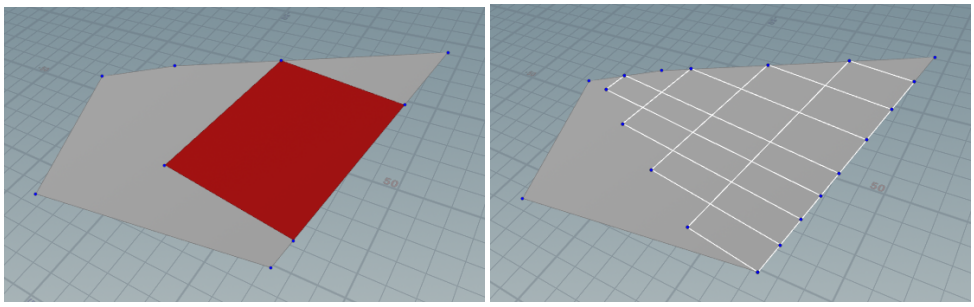
Removes colinear points of 2d polygons, which can cause BSP compilers to fail if extruded.

- colinear_threshold: A point is colinear if the 2 edges it is connected to point in the same direction. This is tested using the dot product of the edge directions. The threshold is the minimum dot product to be considered colinear, so lower values mean that larger angles between edges are considered colinear.
- keep_overlapping_points_on_other_prims: if true, points which overlap between 2 prims are not deleted.
- overlapping_fuse_radius: the distance at which 2 points overlap.
- delete_colinear_points: if false, adds points to a group instead of deleting.
- colinear_group_name: the group to add colinear points to, if 'delete_colinear_points' is false.

10.16 pmm_2d_convex_dissolve

Removes merges prims among a set of 2d polygons by removing interior edges, if the result is convex.

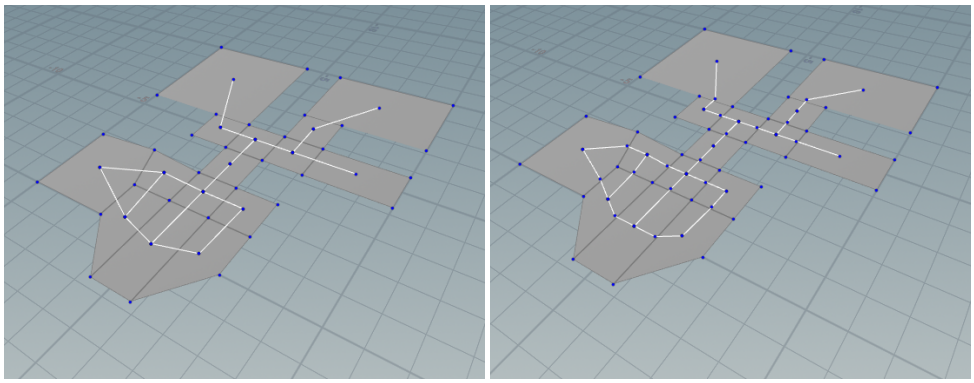
10.17 pmm_2d_convex_to_quad



Left shows an input mesh in grey, result in red. Right shows the candidate quads generated from the longest edge.

Takes a convex 2d polygon as input and outputs a quad that fits within the convex. It does so by colliding a series of quads moved inwards from the mesh's longest edge, and selecting the one with the largest area.

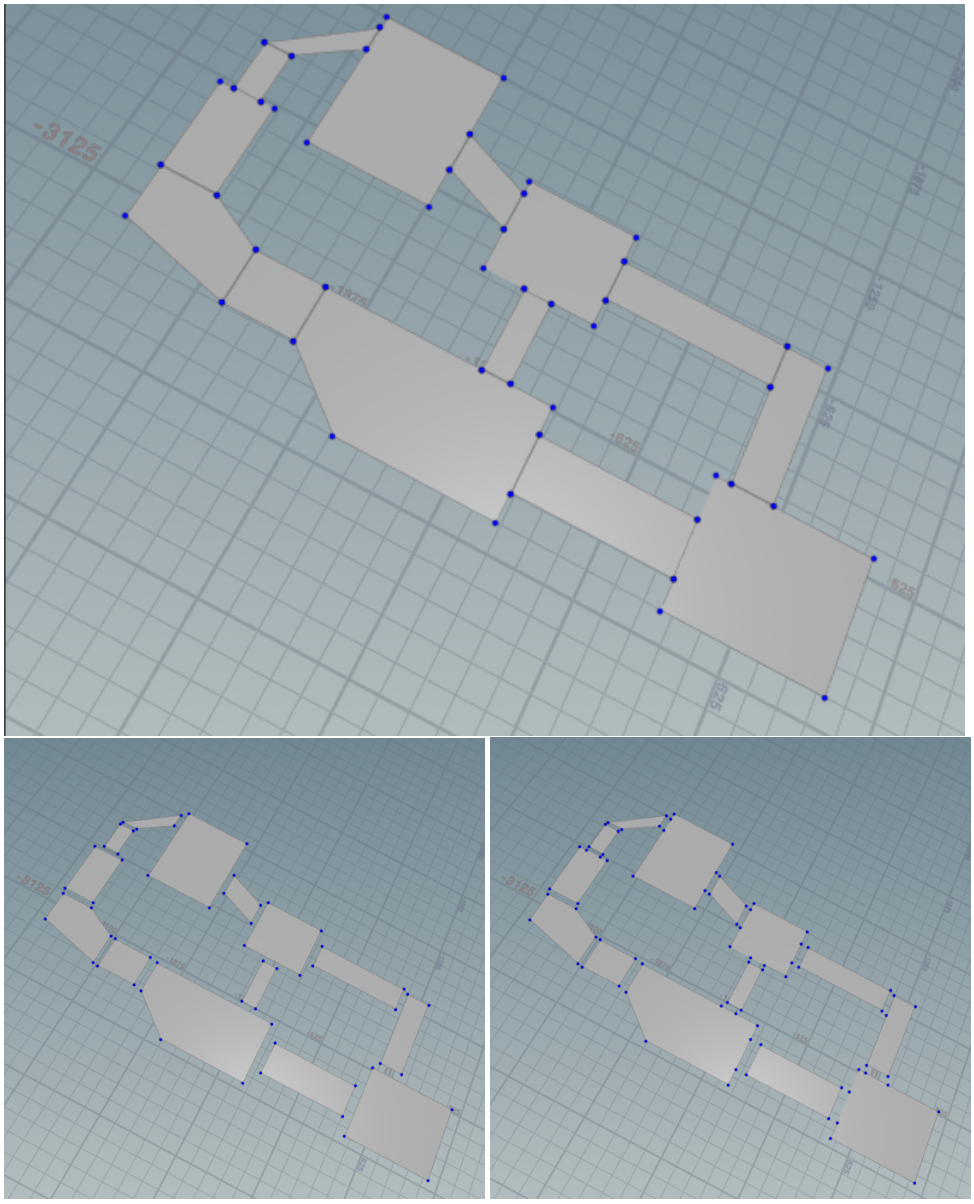
10.18 pmm_2d_dual_graph



In the images above, the input mesh in grey is converted to the graph in white. On the left 'add_edge_center_points' is disabled; on the right it is enabled.

The node converts a 2d floorplan mesh into its 'dual graph', which can be used for pathfinding queries(see the Houdini node 'find shortest path'). Each prim is converted to a point at its center, and each edge shared between 2 prims are converted into a line.

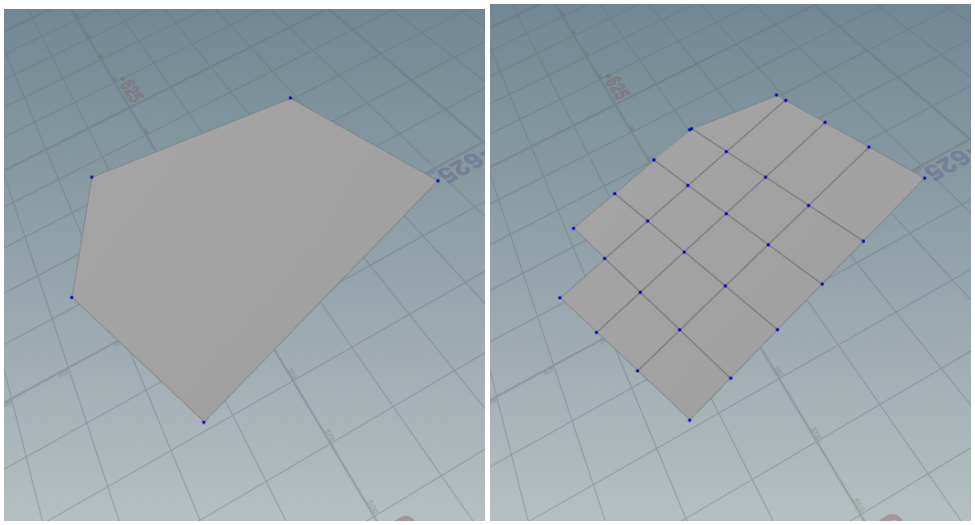
10.19 pmm_2d_split_overlapping_edges



Top shows an example input. Bottom shows an exploded view; left is before splitting and right is after splitting.

Given a set of 2d prims, splits edges by adding points on edges that overlap.
This is used to ensure valid edges on floorplans. For example, pmm_prim_group_interior_exterior_edges will not correctly detect interior edges unless shared edges also share points.

10.20 pmm_2d_obb_rasterize



Slices a 2d mesh along planes that are aligned with its OBB (oriented bounding-box).

10.21 pmm_2d_dissolve_interior_edges

Removes interior edges of a flat closed 2d mesh; the input should have no holes.

10.22 pmm_2d_obb

Computes the OBB (oriented bounding-box) of a 2d polygon.

10.23 pmm_2d_convexify

Performs a partial convex decomposition. Compares the ratio between the area of a non-convex 2d polygon and the area of its convex hull and splits the mesh if the area ratio is below a threshold.

10.24 pmm_uv_unwrap

Performs a world-aligned uv unwrap, computing UV coordinates for each vertex. This is not used for brushes, but patches and map-specific props.

10.25 pmm_raycast_along_mesh

Moves a point with direction along the surface of a mesh.

10.26 pmm_convertline_with_edge_groups

Similar to the 'convert line' node, except it also converts edge groups into prim groups.

11 Per Node Documentation (pmg)

11.1 pmg_stair_maker

A simple stair generator.

11.2 pmg_stair_from_curve

Generates a stair from a guide curve.

11.3 pmg_road_from_curve_on_heightfield

Generates a road from a set of road curves, a cross section, and a heightfield.

11.4 pmg_model_select

11.5 pmg_levelpath_linear_partition

Deprecated. Partitions a curve into 'per-level' and 'shared' regions.

11.6 pmg_levelpath_linear_create_junctions

Deprecated. Takes the output of pmg_levelpath_linear_partition to generate level transition regions.

11.7 pmg_levelpath_linear_blockout_voronoi_bspshell

Deprecated. Takes the output of pmg_levelpath_linear_partition to generate sealing BSP geometry.

11.8 pmg_levelpath_graph_generate

Creates a 2d floorplan for multilevel mapping.

11.9 pmg_levelpath_graph_partition

Partitions a 2d floorplan of multiple levels into separate levels with overlap regions.

11.10 pmg_jetty_from_walls

Generates jetty(s) from a set of 2d quad polygons.

11.11 pmg_floor_plan_generator3_

11.12 pmg_floor_plan_generator2_

11.13 pmg_floorplan_generator4_

Attempts to generate a floorplan by copying shapes to the edges of a base floorplan.

11.14 pmg_blockout_voronoi_bspshell

Deprecated. Generates sealing BSP geometry from a curve.

11.15 pmg_blockout_curve_to_bspshell

Generates sealing BSP geometry from a curve.

11.16 pmg_select_random

Selects a random input.

11.17 pmg_select_match

Selects an input using a query.

11.18 pmg_quad_to_door

Creates a simple door from a quad(a polygon with 4 vertices).

11.19 pmg_quad_to_window1_cover

Creates a simple window cover from a quad.

11.20 pmg_quad_to_window1_

Creates a simple window from a quad.

11.21 pmg_floorplan_from_lot

Generates a 2d building floorplan from a 2d polygon, by rasterizing the polygon to create tiles and clustering the tiles.

11.22 pmg_fp3_stair

Generates stair quads from a flat 3d floorplan mesh.

12 Per Node Documentation (pmg_placer_)

Placer nodes are used to place meshes onto a typically 2d floorplan.

12.1 pmg_placer_aabb

Places a mesh at a point on the AABB(axis-aligned bounding-box) on another mesh.

12.2 pmg_placer_stack

Recursively stacks meshes along the Y-axis.

12.3 pmg_placer_pathfind

Generates paths between random points on a 2d floorplan mesh.

12.4 pmg_placer_plane_project

Fits meshes onto a convex 2d floorplan by projecting the meshes along planes formed by the edges of the floorplan.

12.5 pmg_placer_poly2d_corner

Places meshes on the corners of a 2d polygon.

12.6 pmg_placer_poly2d_edge

Places a single mesh along the edge of a 2d polygon.

12.7 pmg_placer_poly2d_surround

Places meshes along all edges of a 2d polygon, surrounding it.

12.8 pmg_placer_poly2d_fence

Takes a 2d floorplan mesh as input and places 'fence' meshes around its perimeter.

12.9 pmg_placer_poly2d_circular

Copies meshes onto a 2d floorplan, spacing the meshes apart based on the 2d radius of each mesh.

12.10 pmg_placer_poly2d_validate_points

Checks whether meshes would fit on a 2d floorplan if the meshes were copied to the points.

12.11 pmg_placer_poly2d_prune

Removes meshes that do not fit on a 2d floorplan.

12.12 pmg_placer_poly2d_flowfield

Copies meshes to points so that they are aligned with a set of guide curves.

13 Per Node Documentation (pma)

13.1 pma_building1_from_floorplan_

Takes a 2d floorplan mesh and generates a building.

14 Per Node Documentation (pme)

14.1 Model/Model entity loaders

pme_$ENGINE_model nodes are used to place models or entities with models.

- set_class_to(vmf)/set_actor_to(t3d)/set_entity_to(map): a dropdown menu that is used to set the class of the entity; this does not actually store the class, which is set in the 'read only' section.
- convert_to_houdini_coords: converts the mesh from $ENGINE to Houdini coordinate system
- pack_visual_mesh: runs the Houdini 'Pack' node on the mesh. This improves performance in the viewport but makes the mesh uneditable. This option must be disabled in order to access the mesh. For example, the option should be disabled if there is a desk mesh and we want to delete all prims but those on the top(facing upwards) to place items on it.

14.1.1 pme_vmf_model

Loads a Source1 model, typically a prop_static, prop_dynamic or prop_physics, so that it can be placed in the level.

14.1.2 pme_map_model

Loads a IdTech4 model entity so that it can be placed in the level.

14.1.3 pme_t3d_model

Loads a UE1 model entity (an actor) so that it can be placed in the level.

14.2 pme_transform_rotate

Rotates an entity, adjusting its attrib v@pmt_$ENGINE_rotation_euler.

14.3 pme_transform_rotation_from_N

Assigns v@pmt_$ENGINE_rotation_euler from a v@N attrib per point.

14.4 pme_transform_rotation_from_quaternion

Assigns v@pmt_$ENGINE_rotation_euler from a p@orient attrib per point.

14.5 pme_io_connect_entities

Connects entities using the input-output system of each engine.

14.6 pme_io_resolve_connections

Assigns names to entities and saves the actual key-value pairs used for entity-entity io.

14.7 pme_vectorkey_create

A helper node that creates 'vectorkey' points, which are used to set vector-type key-value pairs.

14.8 pme_vectorkey_save

Takes 'vectorkey' points and saves them to the key-value pairs.

14.9 pme_t3d_mover_create

Creates a mover brush entity.

14.10 pme_light

A convienence node that creates a light.

14.11 pme_create_fakeprim

For each point, creates a 'fake' prim so that prim attribs can be associated with it. This is mainly used for multi-level export by assigning the prim attrib i@pmt_levels_cluster.

14.12 pme_levelprops

Used to configure global per-level properties.

14.13 pme_startpoint

A convienence node used to quickly setup a player start point.

15 Per Node Documentation (pmi)

15.1 pmi_convert_texture_paths

Converts between relative and filesystem texture paths.

15.2 pmi_entity_init

A base node for entities; provides a set of dictionary parms that can be edited with the entity-kv editor.

15.3 pmi_entity_link

Connects a visual mesh with a point entity. Used by pme_vmf_model, pme_t3d_model, pme_map_model.

15.4 pmi_link_prims

Creates link prims, which are 2 point open(line) prims used to connect a set of points or prims.

15.5 pmi_load_texture_paths

Loads filesystem texture paths for computing UVs or visualization.

15.6 pmi_load_texture_size

Loads the (width, height) dimensions of textures so that UV offsets can be computed.

15.7 pmi_globalconfig_vex

Copies PMT databases for materials, textures, and sounds as a set of detail attribs so that they can be accessed from VEX.

15.8 pmi_globalconfig_to_vex

Wrapper for pmi_globalconfig_vex.

15.9 pmi_set_coord_system

Sets a detail attrib to indicate the current coordinate system. This does not convert between coordinate systems, but is rather used for error reporting.

16 Known issues

Since the source code for Source 1 and Unreal 1 have not been released, support for these engines is implemented using publicly and legally available documentation and best effort guesses. This means that there will inevitably be a gap between PMT and the actual engine.

16.1 Source Engine 1

- The selection of vertices per plane of each brush is not well documented, so there will be differences between the geometry in PMT and Source1, especially for complex brushes.
- If a map has too many prop_static entities, collision can be disabled and it can cause crashes. This issue might not apply to later versions of Source.
- Degenerate displacement maps can cause vrad.exe to fail, meaning that the map will not be lit.

16.2 Unreal Engine 1

- UVs for Unreal Engine 1 are mostly working but not fully correct. In particular, UV offsets for BSP terrain are not correctly computed in PMT so there will be a discrepancy between the preview in Houdini and the exported result.

16.3 IdTech 4

- The class hierarchy is encoded in C++, which makes it difficult to parse entity key-value pairs.
- Many complex patches with self-overlapping surfaces may cause dmap to freeze or run slowly.

16.4 Houdini

- copytopoints: When using 'Transform Using Point Orientations' with a line prim, the meshes to copy might be rotated even if there is no N, orient, rot, etc. attribute.
- copytopoints: If both inputs have a point attrib, the point attrib from "Target Points to Copy to" is preferred. This means that if the target points have an attrib such as pmt_vmf_entity_class that is empty, then the resulting points will all have pmt_vmf_entity_class == "". If the mesh that is copied is a light mesh, for example, then this would discard it.

- Although Houdini can load .lwo files (glightwave), it does not always work. The Houdini 19.0 docs note that .lw files 'from version 3.5 and earlier' are supported; it is possible that IdTech4 uses a newer version.

17 Additional Notes

17.1 Coordinate conversion

Although it is possible to configure Houdini to use Z-up instead of Y-up, we use Y-up and convert the coordinates of geometry before export. This is since:

- Houdini uses Y-up by default, and nodes/hdas assume this. For example, the 'copy to points' node can be combined with points having a normal(v@N) attrib to assign a yaw to each copied mesh. Another example is that the gamedev building generator does not work with Z-up.
- The handedness of each coordinate system varies from engine to engine, so some type of coordinate conversion is necessary regardless. Additionally, the direction of rotation about each axis differs by engine.

17.2 Houdini file formats support

- It might be possible to load Source (.mdl), UE1 (.3d) and IdTech4(.lwo, .ase, .md5mesh) meshes in Houdini by developing converters to Houdini's native format (.geo/.bgeo) and adding entries in GEOio. However, this would take considerable development time and is not a priority.
- Although Houdini is able to load IdTech4 image formats(.jpg, .tga, .dds) on Windows, and Source .vtf might be supported by developing converters and adding entries in FBio(or FBformats?), the version of Qt integrated with Houdini does not support .dds/.vtf and it is unclear whether it is possible to add support. The material selector and materialsets selector depends on Qt, so as a result the conversion of image formats is necessary.

17.3 Various notes

- PMT stores all paths as lowercase and parses all files as lowercase and is case-insensitive as a result.
- Houdini 'ordered menu' parms for which the list of options are runtime generated should never be used to store values. This is since the list of options can change, erasing the current selection. Instead, create a separate parm and use the 'ordered menu' parm to set it.

References

[1] Stefan Hajnoczi. .map files file format description, algorithms, and code. https://github.com/stefanha/map-files, 2001. Accessed: 25 March 2022.