MPK Substrings

If any object in the map has any of the substrings below as part of its name, its properties will be altered accordingly.

Renderer
==========
- "portal" = portal (object must be a 2d plane oriented along x, y or z axis of the world)
- "antyp" = antyportal (object must be 3d and convex)
- "trans" = translucent mesh
- "water" = 2d shape that has water shader displayed on it in engine
- "noclip" = object is excluded from Havok physics - player and any physics entity in engine can walk through such object
- "2sided" = two-sided object - if it is a plane it will be visible from both sides
- "atest" = object can have transparent parts based on texture with alpha channel. (This is fast transparency mode in which object can only be completely transparent or completely opaque, depending on black or white pixels in alpha channel of texture)
- "decal" = decal object - a 2d plane with texture that has alpha channel. This plane must be placed on another object's surface). Many decals can be combined into one object.
- "barrier" = invisible barrier for players. They don't stop anything beside players - don't work on monsters, projectiles, ragdolls etc.
- "monster" = Monster Barrier. Barrier that stops only monsters. Doesn't work on anything else (players, projectiles, ragdolls etc.).
- "vollight" = volumetric light
- "volfog" = volumetric fog
- "zone" = zone object (must be convex)

If name contains "zone" it can also contain:

- "death" = death zone, object that kills players and monsters on contact (object must be convex)
- "ladderzone" = ladder zone, within this object player can climb like on a ladder (object must be convex)

Other
==========
- "glass" = Glass object which can be broken by player, monster, weapons and explosions. It must consist of two planes, one for each side.

Physics
==========

- "statdest" = static object that can be destroyed (actually upon activation the statdest object disappears and is substituted with one or many physdest objects). It is activated by explosion.

- "physdest" = object that appears as one or more broken pieces of original statdest object after explosion. As a physics object it will be treated as conxes, even if it isn't.

Physdest objects must include WHOLE shape name of the Statdest object (including _shape that Maya adds) Easiest way is to copy whole name of a statdest, paste it as a name for physdest (Maya will add a digit at the end of the name) and then replace 'stat' with 'phys'.

- "phys" = active mesh, physics object. Inactive at level start, activates on touch (or by weapons and explosions). Upon activation it is treated as a convex shape even if it isn't. For instance you have U shaped object wrapped around a pole, it will be pushed away from the pole upon activation as it will be treated as a full convex shape which it wasn't before activation.

If name contains "phys" it can also contain:

- "autodelete" = physics object that disappears after a few seconds from the moment of activation
- "pinned" = physics object that can be activated only by explosion)

Realtime lights

Rendering lightmaps in Maya or other 3d program takes care of lighting only on the static geometry of your level. You still have to set up lighting that will work on all the items in the map and on players. It goes without saying that good realtime lighting matches that on the lightmaps so that all dynamic items blend in perfectly. This isn't easy because realtime lighting has many limits that 'offline' rendering doesn't, but with some practice you can have great results.
There are three basic types of light sources in the PAIN engine.

- Ambient light
- Point light
- Directional light

Tip: In order to see the visual representation of light direction and ranges in editor click on the Edit Lights icon in the toolbar.

Ambient and Directional light are 'master' lights that are set in level properties, but it would be tough if you were limited only to one ambient light and one directional light in your whole level. If you have a map that is a mix of outdoor and indoor geometry you will find that pretty bright ambient light is needed for sunny outdoor while rather dark ambient is necessary for indoors. In such situations you can use CEnvironment type boxes that have their own ambient and directional light settings. So if most of your level is set outdoors you can set master ambient and directional light values in level properties to reflect high ambience value from the sky and bright directional yellow sunlight, while indoors you place virtual boxes that have their own lighting parameters set. To create such a box press ctrl+o, choose CEnvironment class, type in some name and click OK. You can set whatever parameters you like in the Properties window to the right, but be sure to set Overwrite to “True” if you want the lights to work.

Point lights are simple light sources that emit light in all directions. The easiest way to create point light is to pick it from templates tab of the window on the right. Place any of the lights from \Lights\Points\ in your level and edit its properties - color, intensity, start of falloff and range. Because pointlight shines in all directions you may encounter a problem where you need to simulate a light bulb on the ceiling, but the light from your pointlight will shine through the ceiling and illuminate objects on the floor above. To fix that you can use CEnvironment in the room with your pointlight and set the pointlight as a dependent light in the properties of environment. After that your pointlight will only shine on objects within the linked CEnvironment.

Practical notes:

You should avoid overlaping more than two light sources otherwise you will see light 'popping' in and out on weapon if you move through area that has 3 or more lights. This is the only visual artifact happening if too many lights are used at the same time, but apart from that rendering speed might drop. In most situations you will be fine using one directional light (either from properties of level or from a separate CEnvironment) and one point light. Ambient light is not actually a light source (it simply means 'add a value of x,x,x in RGB to each pixel on screen') so it doesn't fall into the equation, you can set it however you like and still use two other light sources, but you should avoid having directional and two or more intersecting pointlights. It is not a disaster if you have some overlapping lights here and there but try to avoid such situations if possible.

Theoretically you can't place one CEnvironment inside of another CEnvironment, because the engine won't know which one to use, but actually it is possible: CEnvironments work in the order they were created (and saved) - so if you create a small environment, save it and then create a bigger one around it, both will work fine, if you do things in a different order, only the bigger environment will be taken into account.

Total number of light sources and environments in level is theoretically infinite and does not affect rendering speed but you should not go overboard with the lights anyway :) It’s better to have around 100 lights than to have a thousand of them.

You can set any positive number in the intensity setting of each light source but values well above 1 will cause artifacts - for example too intense yellow light might actually become greenish.

There does exist a fourth type of lights in the PAIN engine - spot lights, but they are not quite useful as they only have linear falloff (based on distance from light source) and don’t have any angular falloff (so that light would gradually fade out on an object moving away from the spot light).

Painkiller sky tutorial

Using sky model

The sky model for PainEditor is a set of meshes modeled in Maya and exported as regular .mpk using our exporter. Maya's scene and *.mpk* file of sky model is independent of level map, so you can use one sky model in many levels or easily change the sky in a finished level without modifying the level map. The *.mpk* sky file should be saved in the *Data/Maps* directory, the same where the level maps are located. In PainEditor sky model is imported and set up via *SkyDome* section of level properties (Fig. 1). All sky settings are saved in text *.Clevel* file (*Data/Levels/Nameoflevel directory*), so you can also edit them with any text editor.

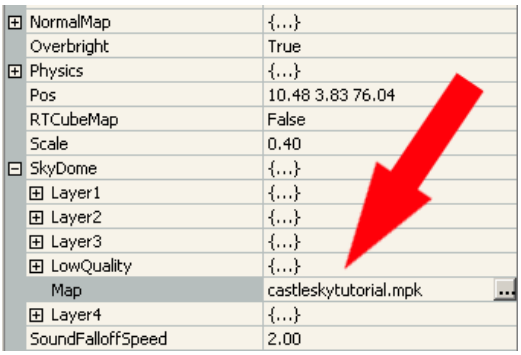


Fig 1. Sky parameters in PainEditor

You can use any sky model from Painkiller levels with your own textures and animations, so let's start by examining the structure of one of them and change its parameters. Run the PainEditor and load *C3L4_Castle* level, to see sky shown on figure 2.

Fig. 2. One of the skies from Painkiller

Sky model can contain up to 4 different meshes called *layers* and represented in PainEditor by *Layer1*, *Layer2*, *Layer3* and *Layer4* fields in *SkyDome* section. On every layer you can mix two different textures with blending mask and use lightmap where it's needed. In the *Castle* level, the sky is constructed with 3

hemispherical layers, as on figure 3. Such model is used in most of Painkiller levels.

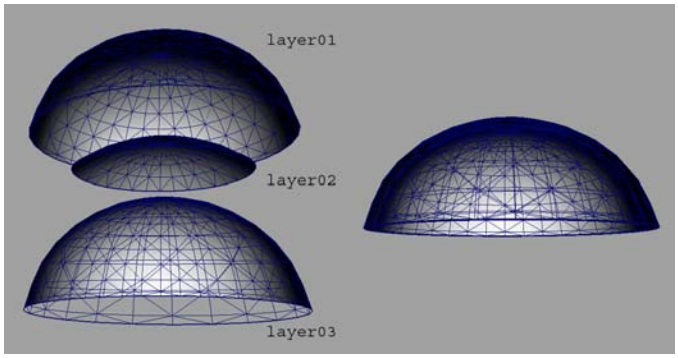


Fig. 3. Three meshes (on the left) are put together to create different layers of sky and clouds (on the right)

Loading textures onto sky layers

On the first, external surface (*layer01*), the main texture is projected and on two internal meshes (*layer02* and *layer03*) are placed moving semi-transparent clouds.
In the simplest case, when you don't want to create sophisticated animation or lighting effects, you'll need only the first layer with basic texture. So, temporarily, we can turn off all other layers and change main texture. Open *Layer2* and *Layer3* rollouts in PainEditor and load *_alpha_zero* texture from *Data/Textures/Skies* to *Tex1* and *Tex2* slots in both rollouts. This is a completely transparent texture (alpha = 0), so those layers become invisible for now. After this soft moving clouds disappear from the sky. Save level properties, using disk icon on main toolbar, so we can always go back to this starting point.
Now open the *Layer1* rollout and load any new texture to *Tex1* slot, replacing existing one (You'll see something like on figure 4).

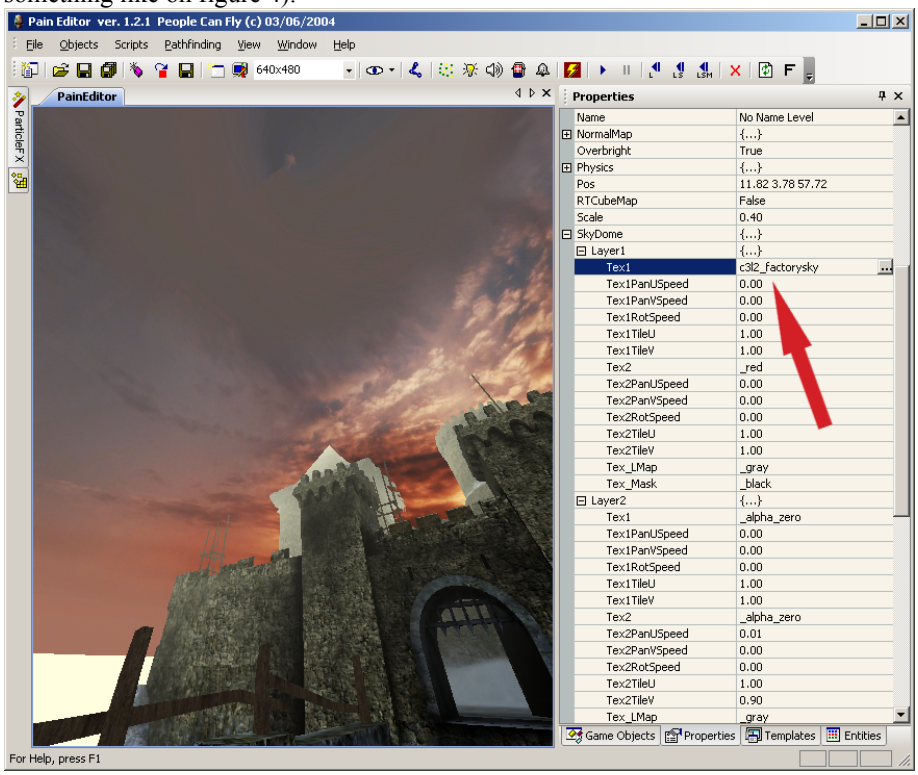


Fig. 4. Replacing the main sky texture on Layer1

Fog color

Using method described above you can load any sky texture into your level and stop thinking about sky, but at least one thing needs further consideration. As you see in left bottom corner of screen on figure 4 (as well as on top of castle towers), the new sky texture doesn't fit to the existing level's fog color. So you have to change setting in *Fog* rollout to give the fog more proper color. After changing fog color to 163,111,92 everything should look much better (Fig. 5).

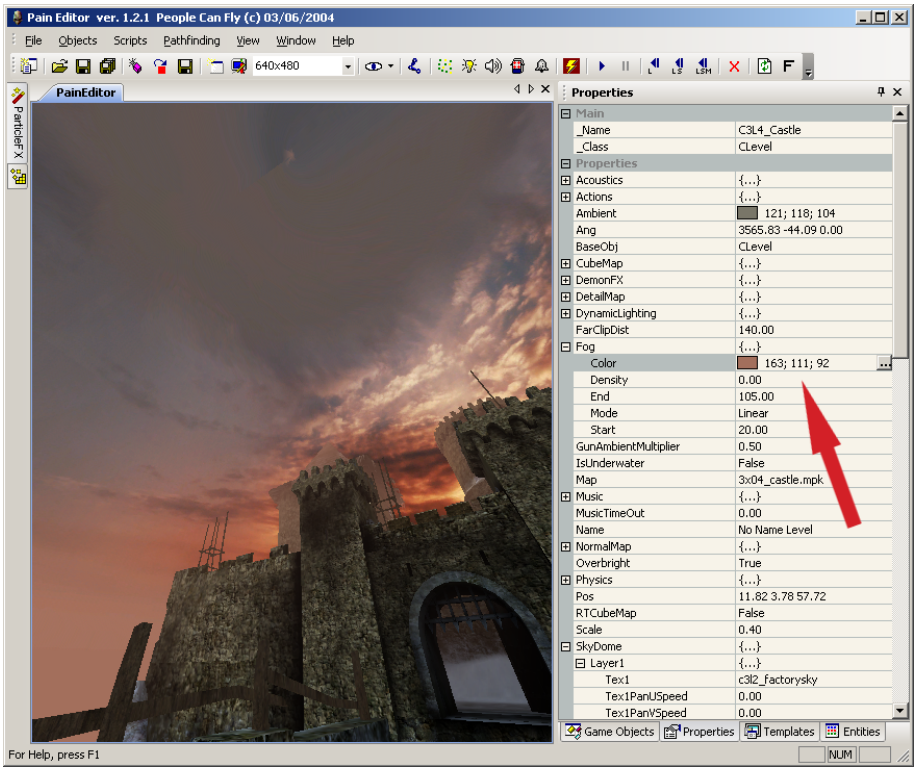


Fig. 5. Fitting the fog color to sky.

Hint: if you need to copy exact fog color from other level to new one, you can read it from .Clevel file (it's written as *o.Fog.Color* parameter, the first 3 numbers in brackets are RGB values to copy).

Additional layers and animations

As we can see on figure 4 and 5, there is a strange distortion of sky texture on top of the screen. It the result of applying cylindrical texture mapping on spherical mesh, so a second layer was added especially to fix this problem. Texture on this layer is blended to transparency and masks distortion on top of first layer. So reload level to earlier saved version and see the distortion of sky above your head. Now load the *c3l4_castleskytop* texture into *Layer2/Tex2* slot (Fig. 6).

Fig. 6. Distortion on sky texture (on the left) is masked under additional layer of clouds (on the right).

The advantage of the structure discussed here is the possibility to animate an additional layer – as you see in editor, the clouds on top of sky are slowly moving in virtual wind direction and blend seamlessly into surrounding environment (instead of cylindrical mapping of texture on *Layer1* we could use planar mapping and avoid distortions, but in that case animation shown here is not possible). You can try different animation effects by changing values of *Tex2* texture's parameters:

1) *Tex2PanUSpeed* – speed in U direction (horizontally in texture coordinates space)
2) *Tex2PanVSpeed* - speed in V direction (vertically in texture coordinates space)
3) *Tex2RotSpeed* – speed of texture rotation
4) *Tex2TileU* – number of texture's repetitions in U direction
5) *Tex2TileV* - number of texture's repetitions in V direction

Texture blending

The effect of transparency in the second layer is accomplished by using a special mask, which creates transition from base cloud texture (*Tex2 = c3l4_castleskytop*) to completely transparent texture (*Tex1 = _alpha_zero*). The mask of transparency is an additional texture, *c3l4_castleskybulbmasker*, which is loaded to *Tex_Mask* slot of *Layer2*. Alpha channel of this texture contains gradient transition from white to black, what results in blending textures from *Tex2* to *Tex1* on *Layer2* surface (for comparison: in *Layer1* we have loaded a completely black mask, which means that only one texture is visible, so it doesn't matter what texture is loaded to second slot, because it's never shown with this mask). In figure 7 the effect of blending is illustrated.

Fig. 7. Blending texture to background with mask

It's important to remember that blending textures with alpha channels is something different than blending them with RGB colors (which is also possible). In this example blending is based on the alpha channel of the mask so RGB part of the mask is completely white. If you load it into picture viewer, you'll see nothing but white screen – to see alpha channel you usually need specialized software, as Photoshop. Also note that mask doesn't move even if all textures are animated – mask is projected from second UV set of mesh, which can't be animated.

The same way as earlier we can set up a third layer, adding more subtle clouds/haze effect. Third layer surrounds the whole sky, so the range of clouds on this layer is much bigger, but finally it also depends on the mask used in *Tex_Mask* slot of *Layer3*. This mask is basically the same as in *Layer2*. As you can see on figure 7, there is a notch in the circular white spot. It's made to make sure that clouds don't reach the sun area, because if the sun was to be blocked out by clouds, all lighting situation in level should naturally change, which is not possible with pre-rendered lightmaps used here. Note, that *c3l4_castleskyclouds* texture used here has it's own alpha channel, so those clouds are much more transparent, giving a very gentle moving haze on the sky.

Lightmaps

On every sky layer you can use lightmap to control lightness of textures, and where it's not needed, neutral gray texture should be placed into *Tex_Lmap* slot. If you see *Layer1* and *Layer2* settings, you'll see such *_gray* texture in their lightmap slots. But on *Layer3* we wanted to use lightmap, because fluffy clouds look much better when they are lighted in surrounding of sun or moon. Lightmap used for this layer is shown on figure 8. Where lightmap is pure grey, lightness of texture doesn't change, and in lighter and darker areas texture varies dependently on lightmap intensity and color. Lightmap, similarly as blending mask, is projected using second UV set, so it doesn't move with animated textures.


Fig. 8. Lightmap used for animated clouds in castle level

Modifying sky model with Maya

You can create your own sky models completely from scratch in Maya, or change example model attached to this tutorial — *CastleSkyTutorial.ma*. Meshes for given sky layer can have any given shape, but you must remember some things:

1) Name of meshes should contain *layer01*, *layer02*, *layer03* or *layer04* string. Only 4 layers are allowed, so any additional or wrong named geometry will be ignored by PainEditor.
2) If given layer is dedicated for transparent textures, its name has to contain *trans* string, so full name of layer is *trans_layer02* or something similar (Fig. 9).
3) If you don't plan to use 4 layers, export only those needed — even if layer has fully transparent textures, it's still rendered as invisible geometry, so it wastes GPU power for unnecessary operations.
4) Too many polygons in meshes results in beefier hardware requirements and slower renderings, but too low-poly meshes causes distortion in textures, especially during animation.
5) Normals of surface need to be reversed to interior of mesh, because during play you are looking on sky from within the model.
6) Sky model is exported from Maya in the same way as level map – select geometry to export, choose *File >> Export Selection*. In Export window choose *Files of type >> MeshPakExport* and navigate to *Data/Maps* folder, saving model with any given name.

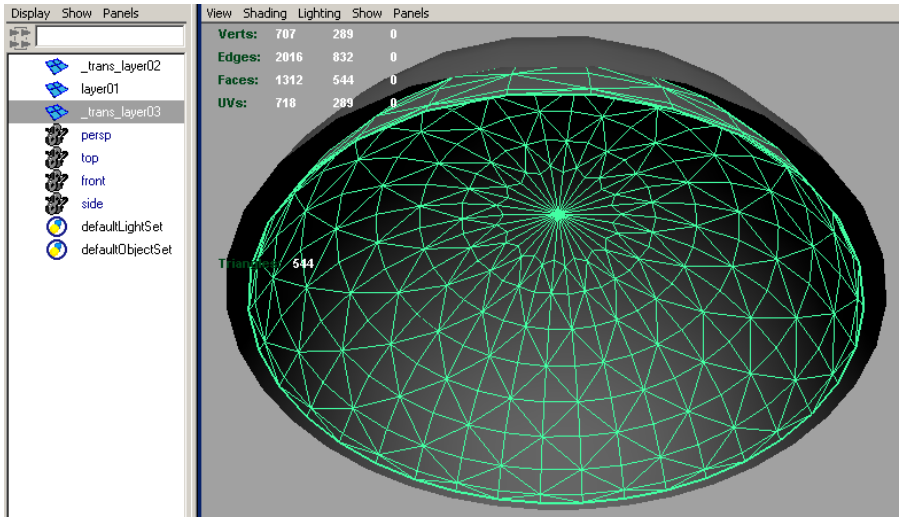


Fig. 9. Example sky model in Maya

Low quality skies

On weaker graphics card, as GeForce 2 or GeForce 4 MX, it is not possible to render all blended textures, masks and lightmaps, so in this case simpler sky model is used. It contains one layer with one texture and its parameters are set up in *LowQuality* rollout of *SkyDome* section. Aside from *Tex* and *Map* fields you have possibility to change *Angle* and *Height* of sky hemisphere over the level.

Sky Tutorial

In the PAIN Engine you can have huge, complex levels that still render very fast. This is because you can tell the engine to render only that which can actually be seen on screen and not the geometry behind the wall or behind a corner. To do that you have to create a set of Zones, Portals and Antiportals in your Maya level geometry. Think of zones as of virtual rooms and think of portals as virtual doors connecting those rooms. When you're standing in one room you can see only as much of another room as the open door lets you. So all you have to do is place in Maya cubes that represent the rooms (they must have 'Zone' string in their name, like FirstZone_shape, or Zone32_shape etc.) and place portals (with 'Portal' string in the name), which can be of any convex shape as long as it lies exactly on a plane where one room ends and another one begins. In some cases Antiportals come in handy - these are convex models that block the view of geometry. They don’t need to be placed between zones (that would be impossible since they are 3d models and not 2d shapes like portals), For example if you have a thick column in the middle of the room you can simply place a cylinder shape inside that column, name it with 'Antyp' string in the name and you're set. Yes, thats 'Antyp' with y, not 'Antip', sorry - that's Polish spelling ;-).

There are a few things to remember. First, you can’t rotate or distort the cubes representing zones. Second, one portal can connect only two zones, but you can have as many portals connecting same two zones as you ant. Third, if some object lies in two zones then it will render all the time, regardless of portals, so it's good to avoid heavy pieces of geometry sharing more than one zone. Fourth, you have to keep in mind that in order to render geometry fast the PAIN Engine considers an object visible only if it's bounding box is visible to the renderer. So even if you can't see some object through the door, you might still be able to see its bounding box, therefore the engine will still render such an object.

It seems then that it might be best to break down all your geometry into small pieces that have small bounding boxes that will work great with the portal system. That is only partially true, because the more objects you have the longer it takes to analyze the whole scene and the longer it takes to render. From a performance point of view, you should find a perfect balance between the number of objects, the number of faces in each objects and their size with respect to visibility. That’s a really tough balance to reach, it will take some practice before you do, but the better you get, the faster your levels will render.

It is our general rule at PCF to keep no more than couple hundred objects in one map, we try to have our objects with no less than 1000 triangles (modern graphics cards render 1k triangle objects almost as fast as 10 triangle objects) and to have a rather small number of zones and portals and definitely to have a small number of antiportals. We have a profiler built in the editor to help you reach a balance. Just hit the ‘H’ key once to display basic information with FPS, and hit the ‘H’ key another time to display detailed information on what percent of processing power is spent on visibility calculation, rendering and other aspects of the game.

Thanks to the profiler you will be able to see if it is rendering that slows down your map or if it is too complex a set of zones, portals and antiportals. Third, if you make some mistakes placing your zones, portals and antiportals you may have parts of geometry disappearing or your FPS may seriously drop so its always good to check the log after loading your map - there is a ton of information in the bottom Output window of PainEd. Load your map, then scroll the Output window up until you see information about zones, portals and antiportals. If something is wrong (for example if you have one portal that connects more than two zones) you will see a warning somewhere in the Output log.

Ed Tutorial

OVERVIEW

Arguably the greatest feature of Painkiller's PAIN Engine is its ability to render levels completely created in professional graphics packages. It allows artists and level designers to completely free their imagination - the tool is no longer a limiting factor. You can build your level using exactly same tools that are used to create CG special effects in Hollywood blockbusters. If you were to create a classic sculpture in any of the existing game engine editors you would quickly realize that the limited tools at your disposal are not enough. Well, the PAIN Engine allows you to create as much as *you* can, not as much as the level editor lets you. Same goes for lighting - you can use unlimited amounts of different light sources in your scene, you can tweak softness of shadows, tweak intensity and color of indirect reflected and refracted light on your lightmaps - whatever you like. And even though you can create very complicated data for your levels, the PAIN Engine will render them very efficiently, not only because it is so robust, but also thanks to the fact that you can manually set up zones, portals and antiportals that tell the engine to render only that part of level which is actually displayed on screen. No other tool can optimize your level better than you can.

Creating levels for Painkiller can be divided into following parts:

- Building level geometry
- Texturing
- Setting up lighting
- Rendering (baking) lightmaps
- Optimizing geometry visibility (zones, portals, antiportals)
- Importing level to PainEd and placing items (spawn points, ammo, teleports etc.)

Painkiller level creation and editing

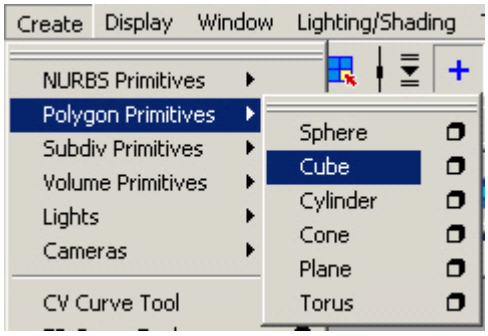
In this tutorial you will learn how to create levels for Painkiller in Maya. I assume basic knowledge of Maya, but nevertheless even simple steps are described in detail so quite possibly even newbies will be able to follow this tutorial without any additional reading. In case there is anything unclear, you can find lots of tutorials on Maya on the Web. During the modeling and texturing phases only standard and easy editing tools are used. Lighting and lightmap baking is a bit more tricky but not difficult once you get the hang of it.
The tutorial is divided into several sections:

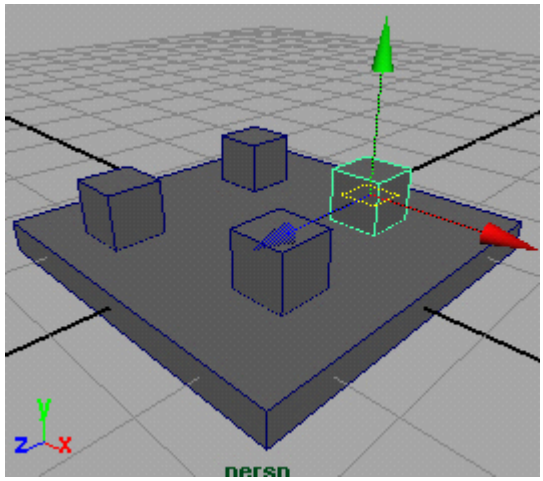
- Building Geometry
- Texturing
- Lighting
- Export
- Level Editing in PainEd
- Appendix 1: Creating Lightmaps in Maya

At the end of each section you will find additional, but often quite important notes.

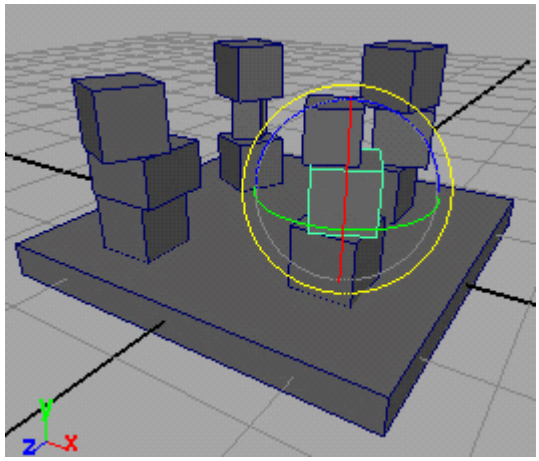
Creating geometry

Let's build a simple scene consisting of few boxes. To create a box we choose from the main menu **[menu|create|Polygon primitives| cube].**





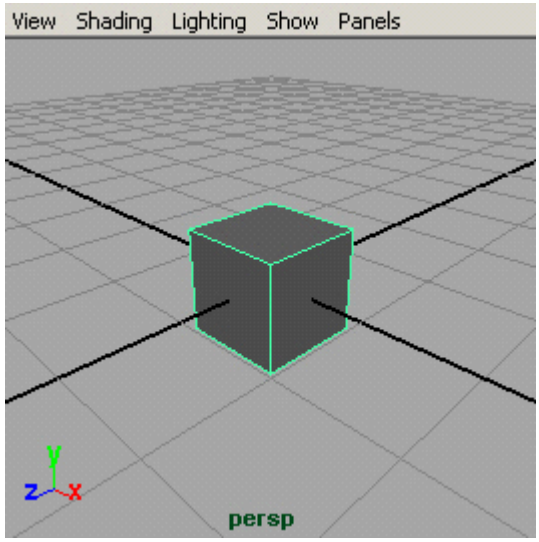
Duplicate even more of the cubes, move them up and rotate a bit - **rotate** (shortcut “e”)



Save the scene as level1.mb

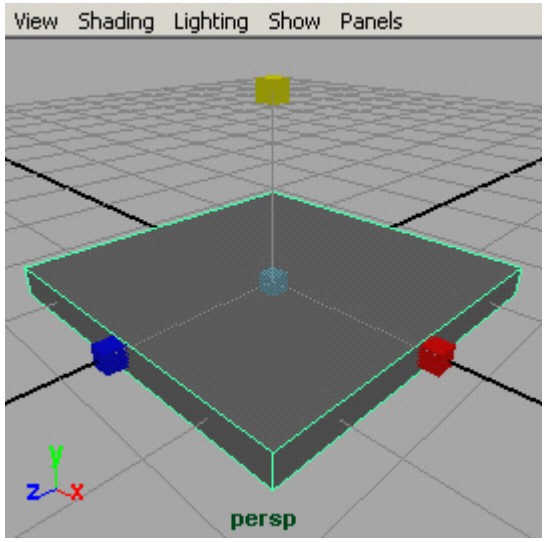
Additional notes

- Only 'mesh' type of geometry can be exported to PainEd
- In Maya each object has two name fields. When you select an object you see it's 'transform' name. PainEd however uses the other name, for example when you create cube, you can see name "pCube1" while the name that PainEd reads is "pCubeShape1" (unless you renamed it of course). You can see the different names in **Attribute Editor** (shortcut “ctrl + a”)
- Mesh name can include additional strings that give those objects special attributes (like transparency, invisible barrier etc.) in PK. More on that in *Export* section.
- Single mesh can have up to 20000 triangles. Whole level



This will be the floor of our scene.

Let’s edit the cube. First we scale it down along the Y axis - **scale** (shortcut “r”) i scale it up along the X and Z.



We need some more geometry in our scene - lets create a new cube. Move it a bit to the corner of the 'floor', duplicate three times - **[menu | duplicate]** and place the new cubes near remaining floor corners - **move** (shortcut “w”).

keep no more than a couple of hundred objects.

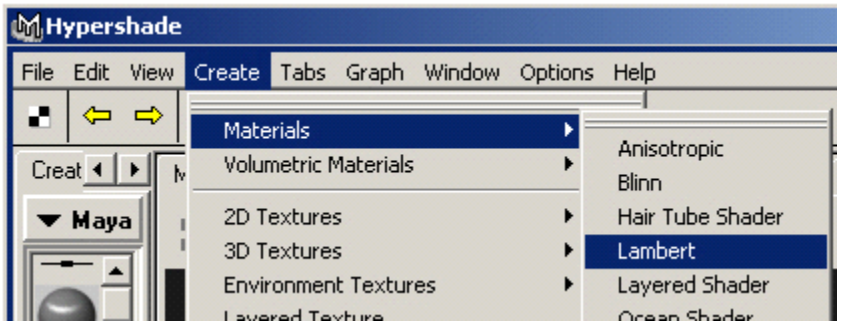
- Remember to avoid long and skinny triangles - they are bad for rendering, bad for physics. Bad.
- Exported geometry is automatically triangulated, but you can also triangulate polygons manually to check if everything is OK - **[menu | polygons | triangulate]**.

Practical notes

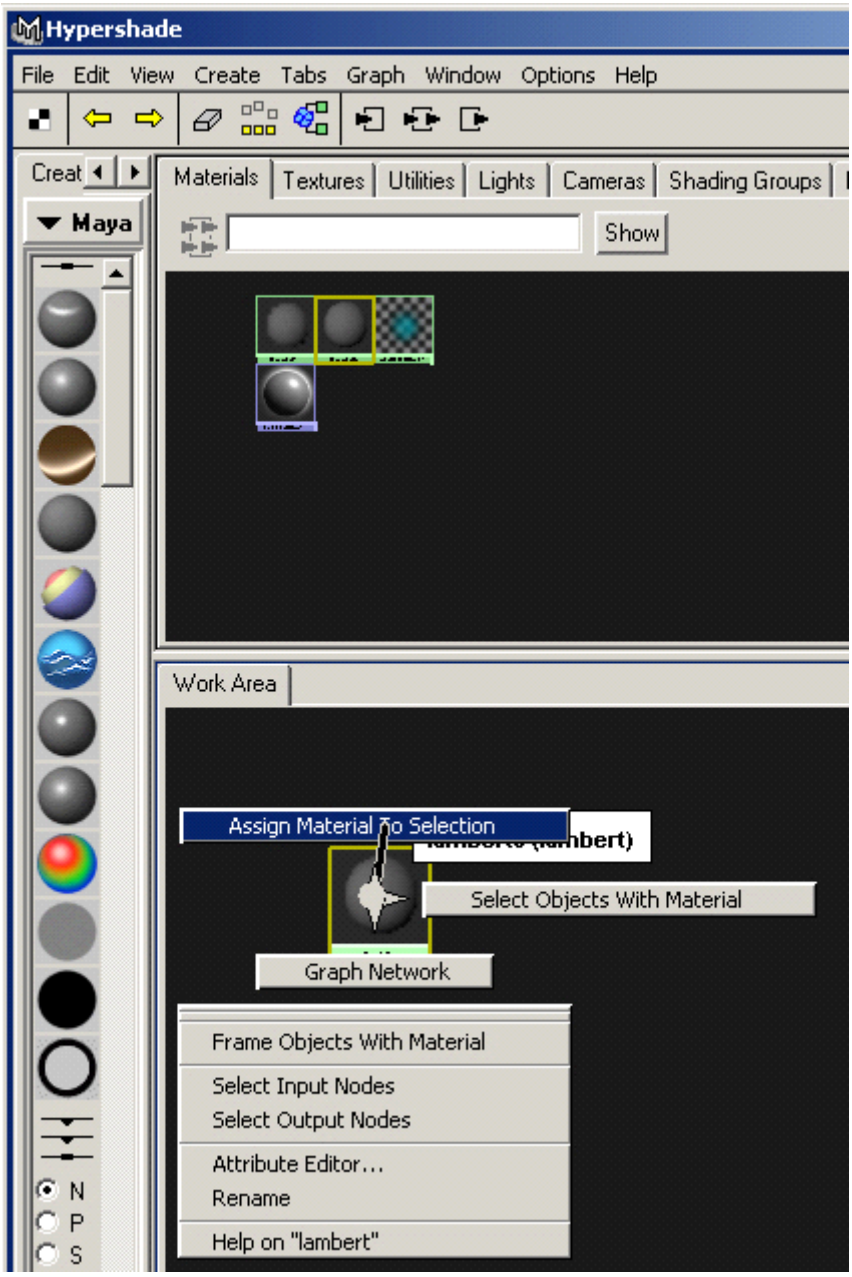
- It makes sense to delete history of your meshes from time to time - **[menu | Edit | delete by Type | history].** It makes the scene file smaller, load times shorter and reduces the risk of errors in geometry.
- Polygons should not overlap or cross each other too much - it will waste lightmap space and increase likelihood of lightmap artifacts.
- Cleaning up geometry - **[menu|edit polygons|clean-up** from time to time can help you maintain clean topology of your geometry.

Texturing

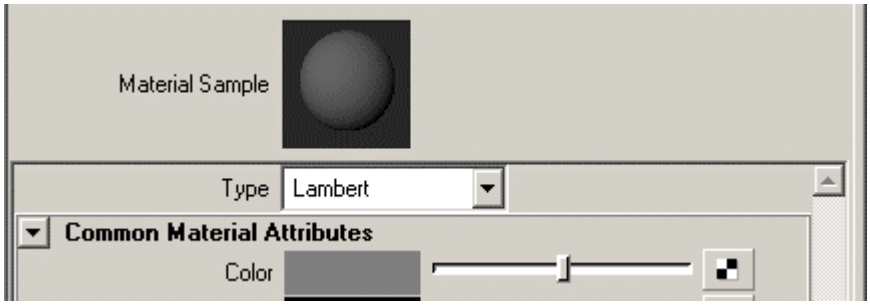
To apply textures to our geometry we use HyperShade editor- **[menu|window|rendering editors | hypershade].**
Let us create a new Lambert material:



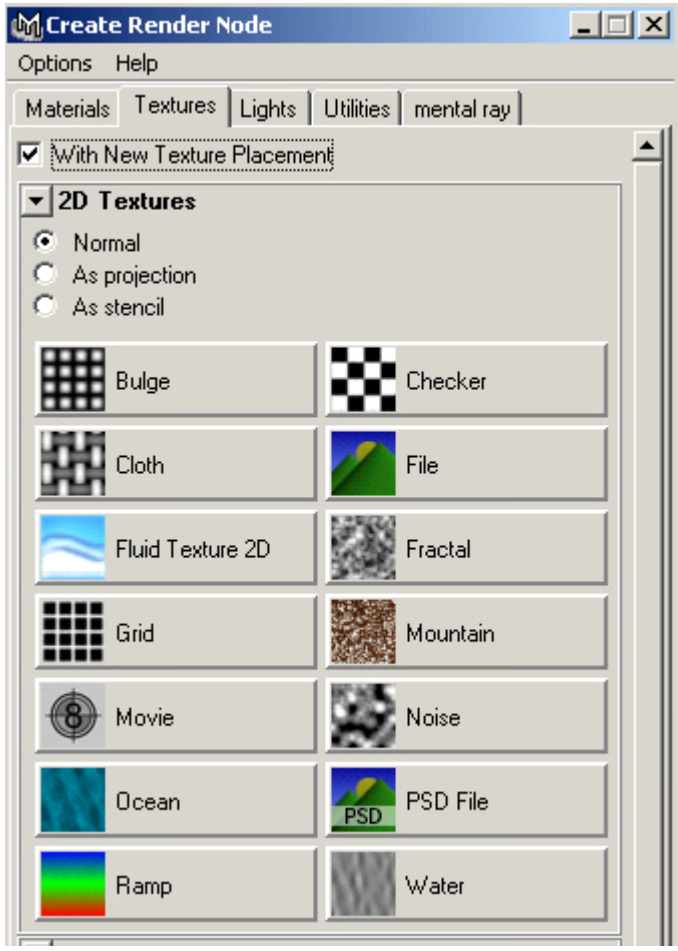
Select the floor object and assign material to it - in the **work area** of **hyperShade** move your mouse over material 'Lamber1', right-click it and choose **"assign material to selection".**



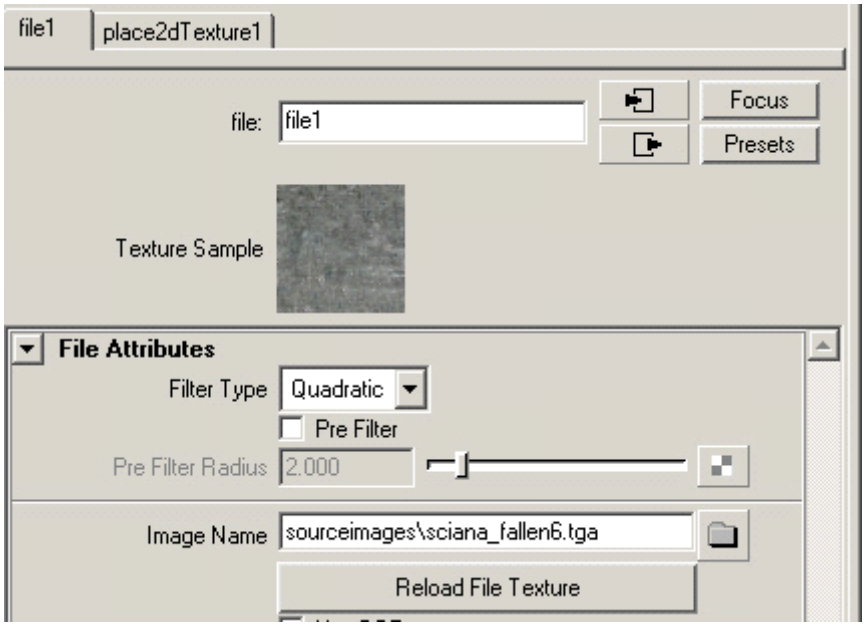
Doubleclick on your material in **hyperShade** and you will see another window - **Attribute Editor(AE)** of that node. Click the small checker button on the 'color' level.



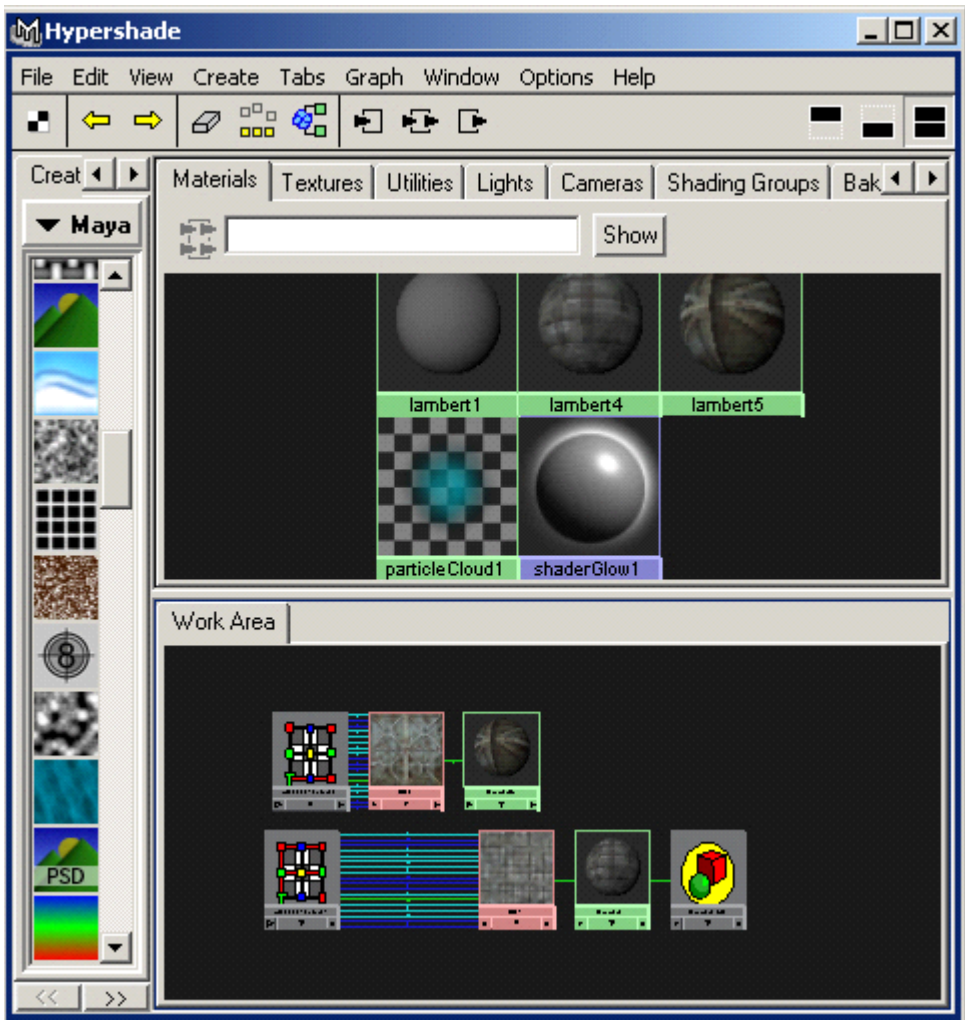
A new **“Create Render Node”** window opens. Click Textures tab then click on the File button in 2D Textures section..



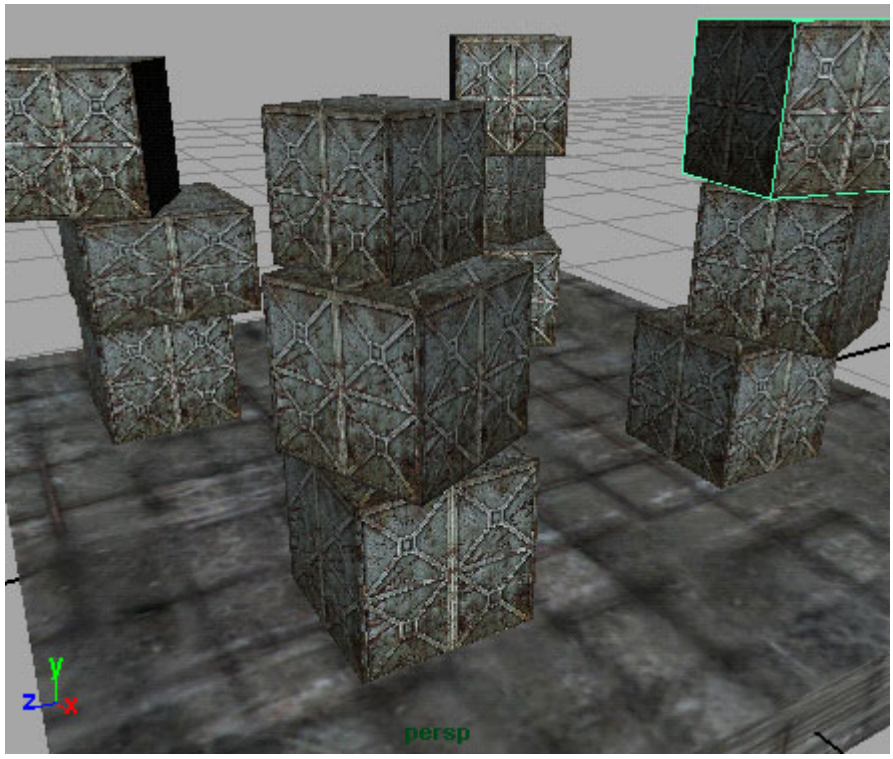
In the File Attributes section you can specify your needed texture file along with its path (you can browse your disk by clicking on the small button to the right).



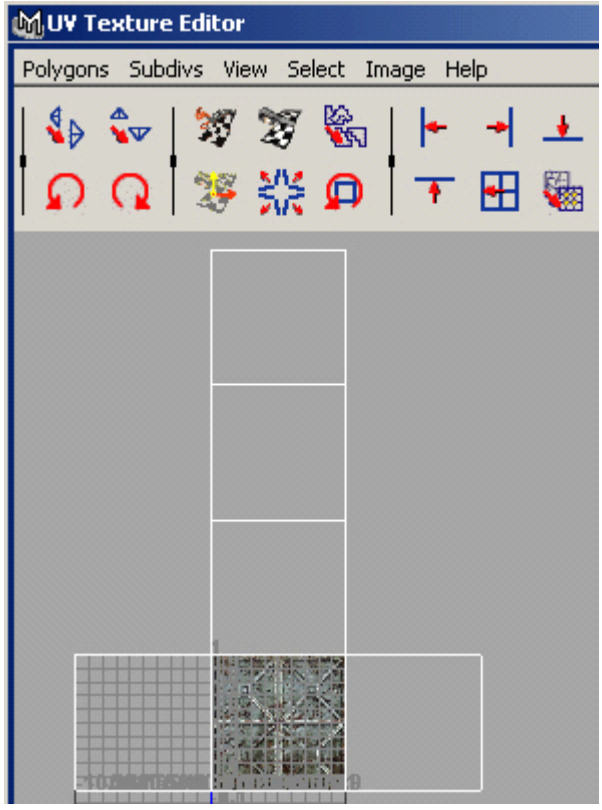
Create another material in the same way as in the steps above, select the remaining cubes in your scenes and apply this material to them. You can always check the graph of your material by clicking the  icon on the toolbar of **Hypershade**. It shows all materials and files assigned to the selected mesh.



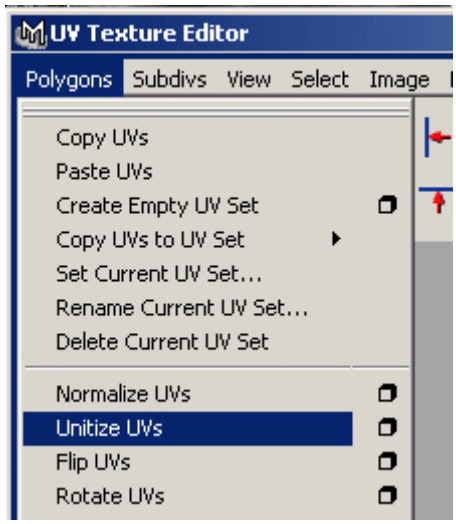
You can see the results of your texturing by switching your active Maya view port to textured mode - just press the shortcut key '6' to do that.



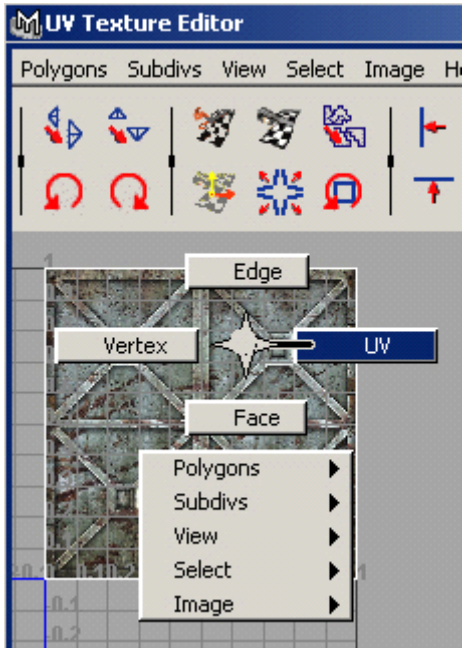
Now adjust mapping (alignment) of your texture. Select the boxes and open **[menu|window|UV Texture Editor].**



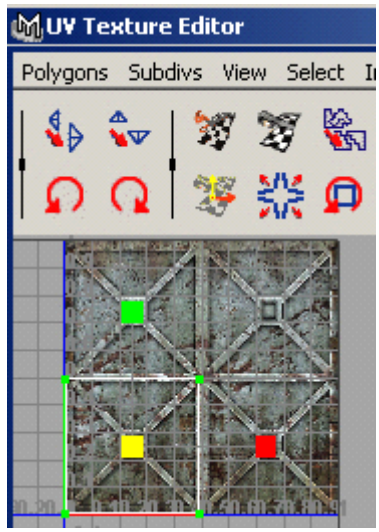
Select **Utilize UVs**



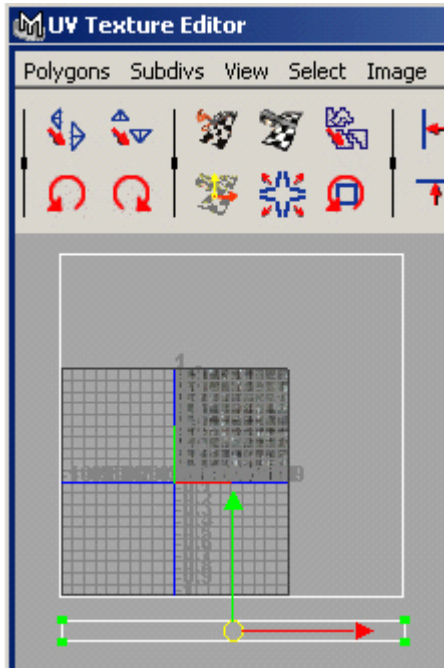
That will normalize mapping coordinates of each cube to a square. You can select UV points (vertices of the shapes that represent sides of your cubes) in **UV Texture Editor** by right-clicking and selecting UV from the pop-up menu:



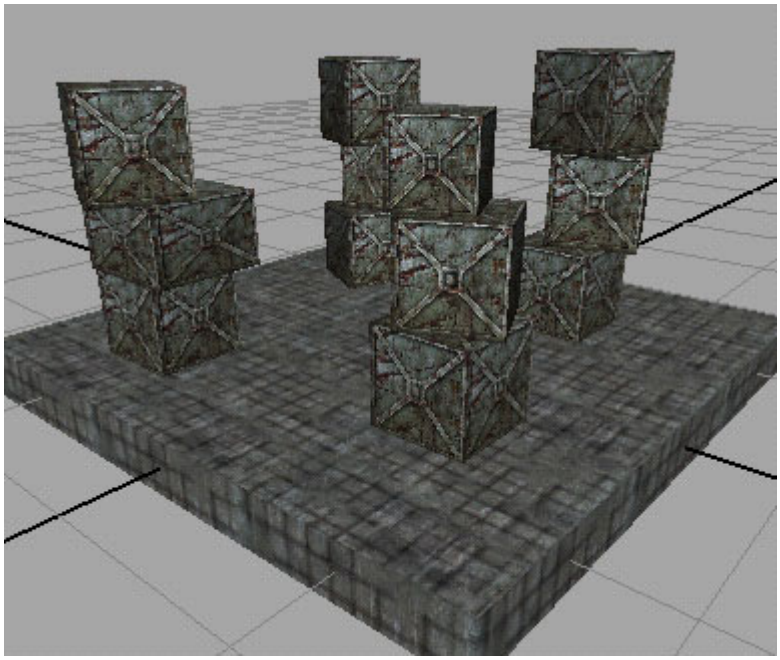
Using the **move** and **scale** tools adjust the UV points.



Adjust floor object mapping as well.



Now it looks like this:



Now we are ready to export our sample scene and load it into PainEd.

Practical notes:

- Remember to add texture to the 'color' attribute of the material, other attributes are not exported to PainEd.
- Transparency can be set for an object by adding “trans” to the object's name. Of course apart from the name change you also need a texture with alpha channel (transparency information). Sorting of visibility in PK is done by objects and not by faces, which means that if you want to have a transparent cube and be able to see one side through another you will need to have each side as a separate object.
- There are many different material types in Maya but you need to stick to Lambert material for PK purposes.
- If you want to tile the texture on your object just do that by scaling the UV coordinates. Other methods of storing texture tiling information (tiling parameters in file node or "place2d") are not supported by PainEd.
- PainEd ignores texture path and texture extension, so there is no problem if you assigned texture “c:\a\sourceimages\delta.bmp” as long as you have a corresponding .dds or .tga texture in PK\data\textures\levels\mylevel\). If there are multiple versions of texture in different formats available, PK uses this order: 1. .dds 2. .tga 3. .bmp

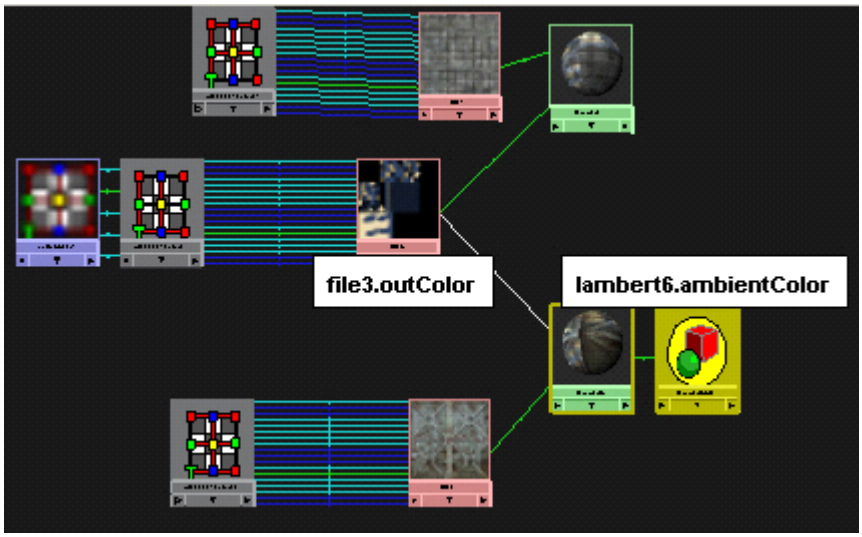
Lightmaps

Lightmaps are used for static lighting in Painkiller. They are bitmaps containing lighting information rendered (or 'baked') in Maya (see Appendix 1: creating lightmaps in Maya).

Practical notes:

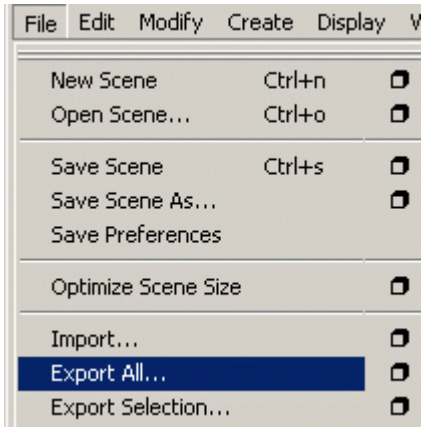
- Lightmaps must be nodes of "file" type linked to "ambientColor" of object material.
- It is best to combine as many objects as possible and render a single large lightmap for all of them together - It renders faster in the engine than many small lightmaps.
- One object can have only one lightmap but many objects can share the same lightmap (after lightmap rendering you can separate previously joined geometry into separate objects).
- Lightmaps are placed on the geometry like any other textures, but instead of the 1st UV channel they use texture coordinates from 2nd UV channel. If an object doesn't have a 2nd UV set defined, then lightmap can't be linked to it and this object is lit in engine by realtime lighting.
- UV coordinates for lightmaps should not overlap, unless of course you wish to have exact same lighting on different parts of geometry.
- UV coordinates of lightmaps must be normalized (meaning they have to be contained in the 0-1 square).
- If a high level of brightness and intensity is desired on lightmaps then 'overbright' option has to be enabled in level settings in the editor. This setting multiplies brightness of lightmaps by two, which means that lights used for rendering lightmaps have to be around two times darker than usual. If normal brightness of lights is used for overbright in engine then color artifacts will occur.

This is an example of two materials with lightmap on a single object.

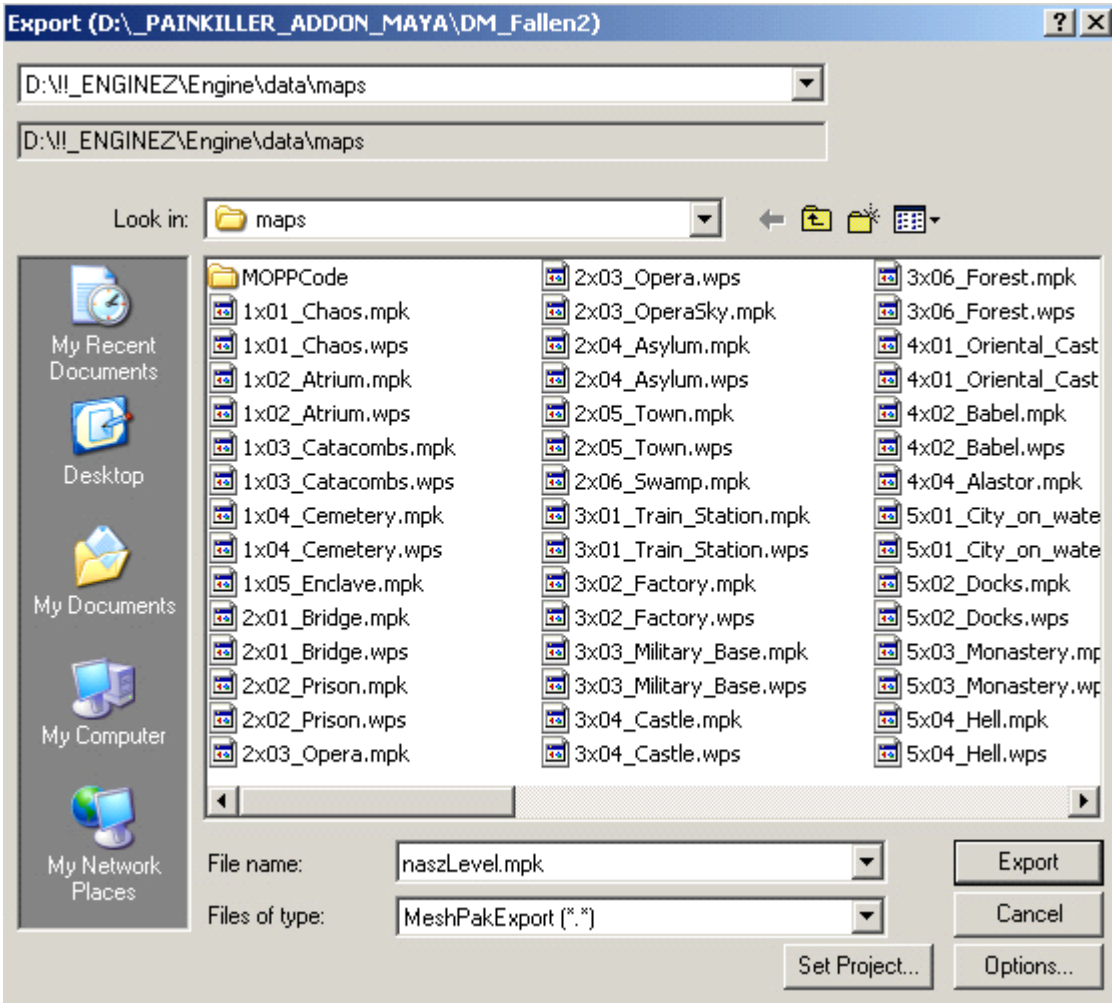


Export

Load **PCFMeshPlug.mll** (it should be placed in \bin\plug-ins\ dir) plugin in:
[menu|window|Settings/Preferences|Plug-in Manager]
You can either "**export all"** geometry or just "**export selected"**.



Export the map into directory "C:\Program Files\Dreamcatcher\Painkiller\data\maps\" (alter the path if you installed PK in different dir).



Textures and lightmaps must be placed in the directory “C:\Program Files\Dreamcatcher\Painkiller\data\textures\levels\mapname”, where 'mapname' is the name of your exported .mpk map.

Practical Notes

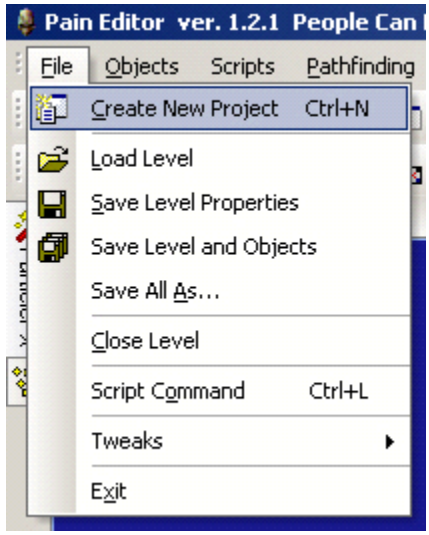
- No matter how small or how huge levels you create in Maya you can always adjust them to the right scale in PainEd using Scale parameter. However it is best to stick to the default 0,3 scale and scale your geometry in Maya to fit editor scale.
- If you fall through the floor or other geometry in PainEd it means that after a geometry change the Havok representation of geometry was not updated. You have to delete the directory with your map's name in \data\maps\MOPPCode\ and reload the level. The engine will recreate the proper physics representation of your new geometry.
- File \MPK Substrings.txt in \Docs dir of the game contains the list of attributes that can be given to objects in your map.

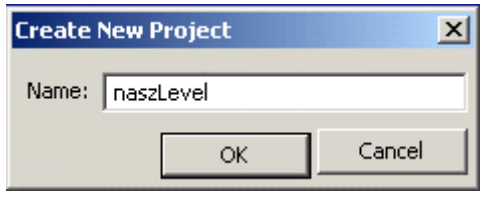
Creating level in PainEd

File Paths:

Editor
"..\Painkiller\Bin\ PainEditor.exe"
Map geometry *.mpk
"..\Engine\Data\maps"
Level settings, items etc.
"..\Engine\Data\levels\levelname\"
Level textures and lightmaps
"..\Engine\Data\textures\Levels\levelname\
"

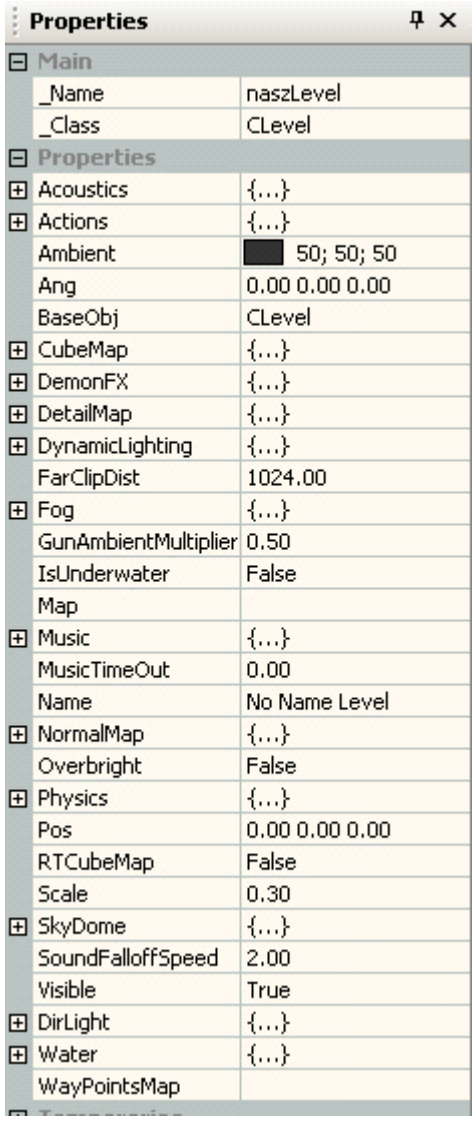
Launch the editor and select "Create New Project".





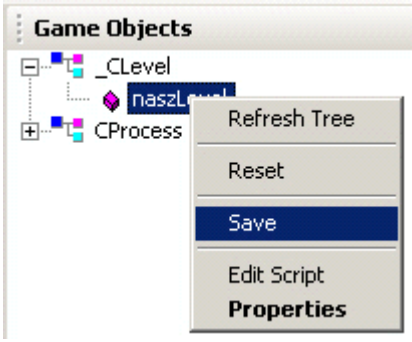
It is best to give your Project level exactly the same name as your .mpk exported map name - it will help you avoid confusion.
In the **Game objects** tab right-click on your level's name and choose **Save.**

Reload your newly created level by clicking  (**“Reload Level Objects, All scripts And Map**”) on the toolbar. After that you can doubleclick on your level's name in **Game objects** tab and **Properties** of your Project level will open**.**



Click on the Map field and choose your .mpk map file. You should see your level

in main window now. Save your level like before.



If you placed all the textures and lightmaps in proper directories you should see your map in its whole glory. You can change the Overbright setting to True (this is the desired setting as it makes lighting appear more intense and vivid. After the change you have to save and reload your level. If the scale of your level is not right you can adjust it in the editor but it will be better to rescale your level in Maya as changing the default scale in the editor might influence physics behavior. You can hit "F" key and start walking around your level.

Here are the basics of working with PainEd:

You can rotate the camera of the viewport with RMB (Right Mouse Button)
You can move the camera in the viewport forward or backwards with LMB (Left Mouse Button)
You can move the camera in the viewport sideways with LMB+RMB buttons
You can select objects in the map by shift-clicking on it

You can move selected object in the map by dragging red, green or blue axis of the object with shift+LMB
You can rotate the selected object along the red, green or blue axis with shift+RMB
You can scale the selected object (like CBox, CEnvironment etc.) with shift+ctrl+LMB. Scale of items has to be changed numerically in the properties of each item.

After you have imported your map geometry into PainEd you need to place some respawn points. Hit Ctrl+O to bring the Create New Object window, select CArea class, type the name for the spawn point and hit OK. Respawn point is created, but you won't see it in the map until you click on Edit Areas icon in the toolbar. The small green square is your respawn point, the red line pointing from it is the direction that player is facing when he respawns. When you select this respawn point (Shift+LMB) you can change the direction of the red line with z and x keys.

After you're done placing and tweaking each respawn point you have to save it. You can either rightclick on the name of the respawn in Game Objects window on the right and choose Save, or if you want to save multiple respawns you can rightclick on the CArea class in same window.
We can now start adding items. Most of the item types are predefined and can be found in Templates tab of window on the right. There are all sorts of ammo boxes, armor, weapons, power-ups, even a jump pad. Many of the objects can have their properties tweaked, for example you can edit the strength of a jump pad. Remember to save each item or all of them together after tweaking.
You can also add teleports to your map. In PainEd Teleport is a virtual box of a CBox class. It means that teleport itself does not have any geometry – it's invisible. Therefore you must prepare some geometry for your teleport in Maya, and then place the actual teleport in the place of geometry, otherwise you won't be able to see a place from which you teleport! Apart from the teleporter itself you also need to define a place where the teleport will send you. This point is of identical class as spawn points (CArea class). After you have created the point of destination you simply provide its name in the properties of teleport and you're done.
These are essential parts of making a MP map. Save all of your work, launch MP and test your map! Keep in mind that unless you FIRST launch MP and THEN do editing you will be in SP physics mode which is significantly different from MP physics (for example rocket jumps in SP are disabled). After you're satisfied with your work you will definitely want to share it with others - easiest way to do it is to create a single .pkm pak containing all the data of your level. Bring up *Choose a level* window (File\Load Level), select your level from the list and click on the Create PKM button. This will generate a .pkm file in \data directory of your game.

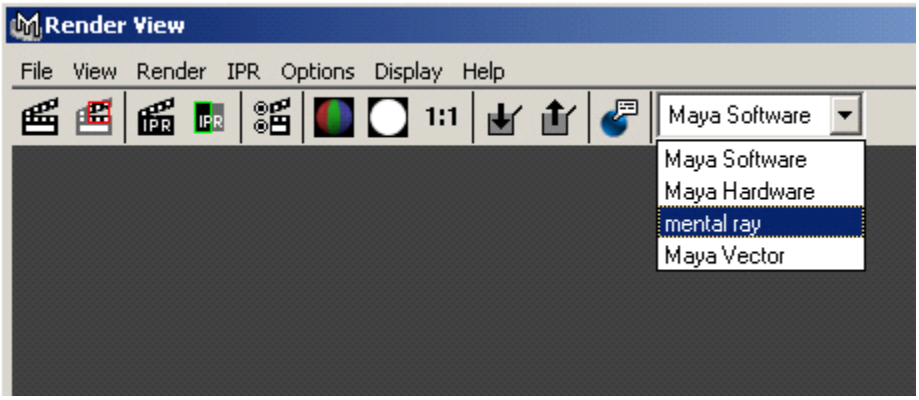
Appendix 1: creating lightmaps in Maya

This tutorial focuses on rendering (or baking) lightmaps using Mental Ray in Maya 6.0, however Mental Ray can be used for rendering lightmaps in Maya 4.5 and later versions.

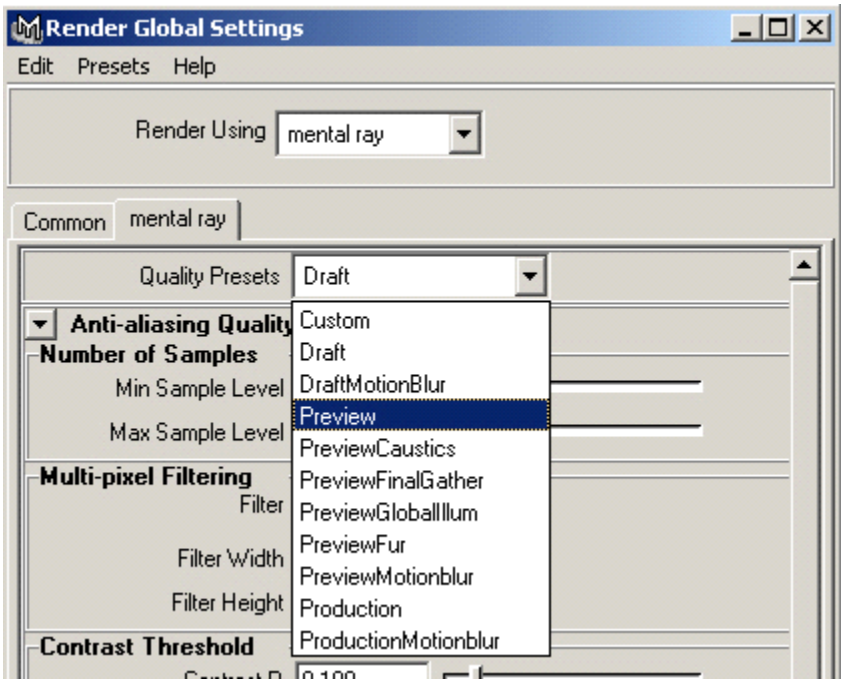
1. Basic steps

Load plug-in Mayatomr.mll in **[menu|window|Settings/Preferences| Plug-in Manager].**

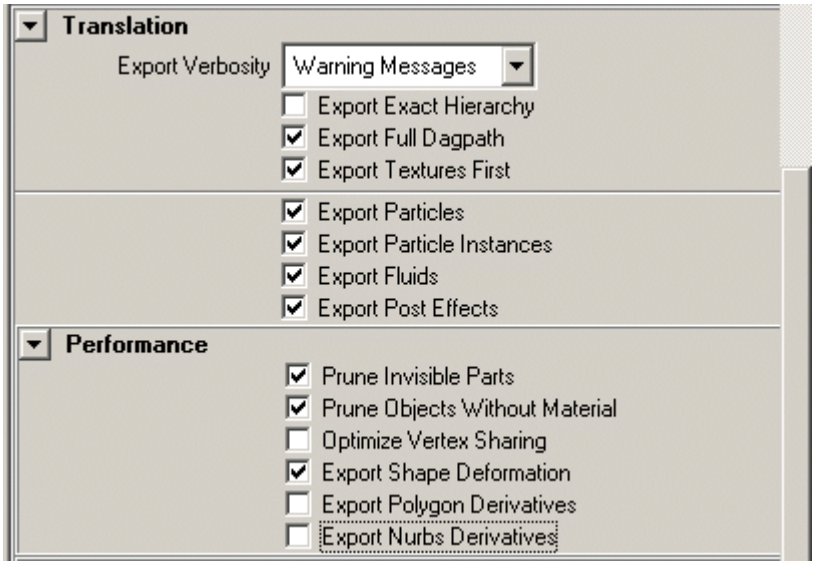
Open window **[menu|window|Rendering Editors| Render View]** and choose Mental Ray renderer.



Open **Render Globals** 
Open **mental ray** tab and choose "Preview" from "Quality Presets".



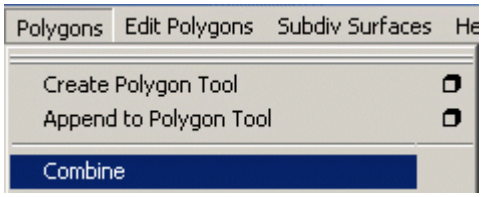
Set all options like in the screenshot below.



2. Combining object into larger entities

We want to have as many objects on a single lightmap as possible. The fewer lightmaps the faster your map will render. It's hard to tell exactly how many objects can share one 1024 by 1024 pixels lightmap, but usually we pack around

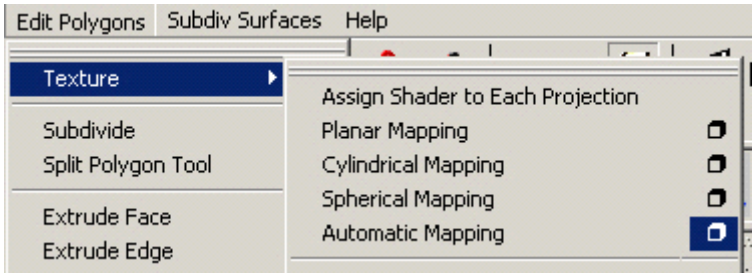
single small objects but keep in mind this is not optimal way. For this tutorial we will divide our scene into two lightmap objects. Choose all 'column' objects and combine them together: **[menu|Polygons|Combine]**.



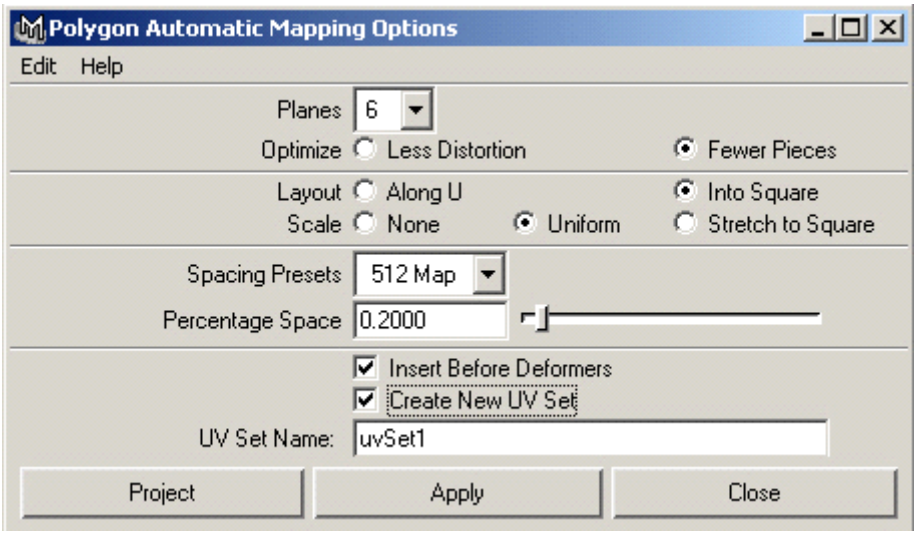
We leave our floor box as a second object with separate lightmap. Now our scene has two objects to bake lightmaps on.

3. Assigning second UV set

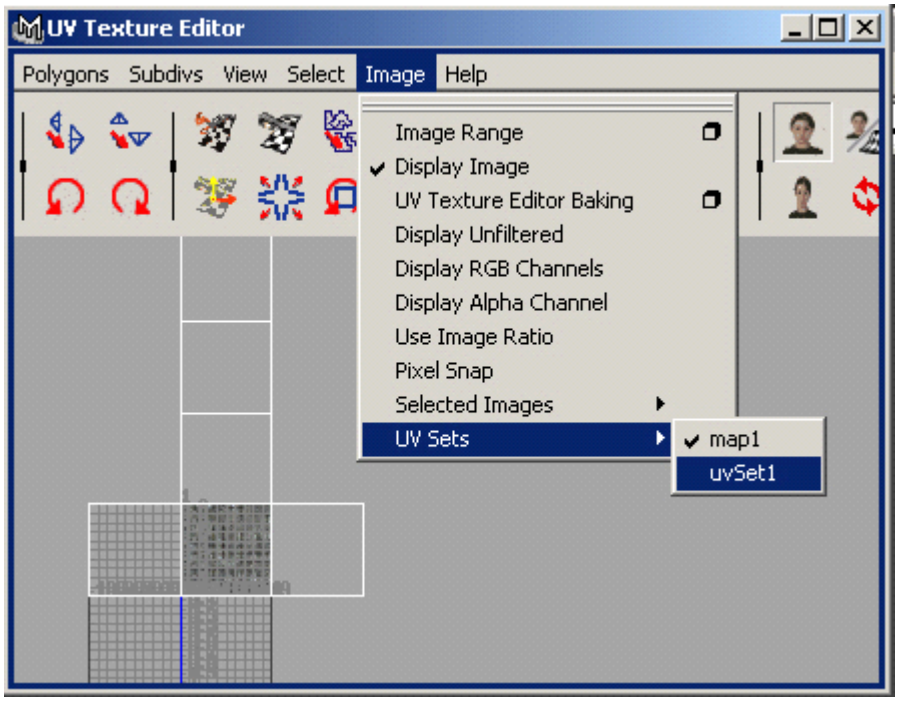
We have to create a second UV set that lightmaps will use. First we have to freeze any transformations of our geometry: **[menu|modify|Freeze Transformations].** Now you can generate mapping coordinates for our lightmaps on second UV set - choose: **automatic mapping tool**.



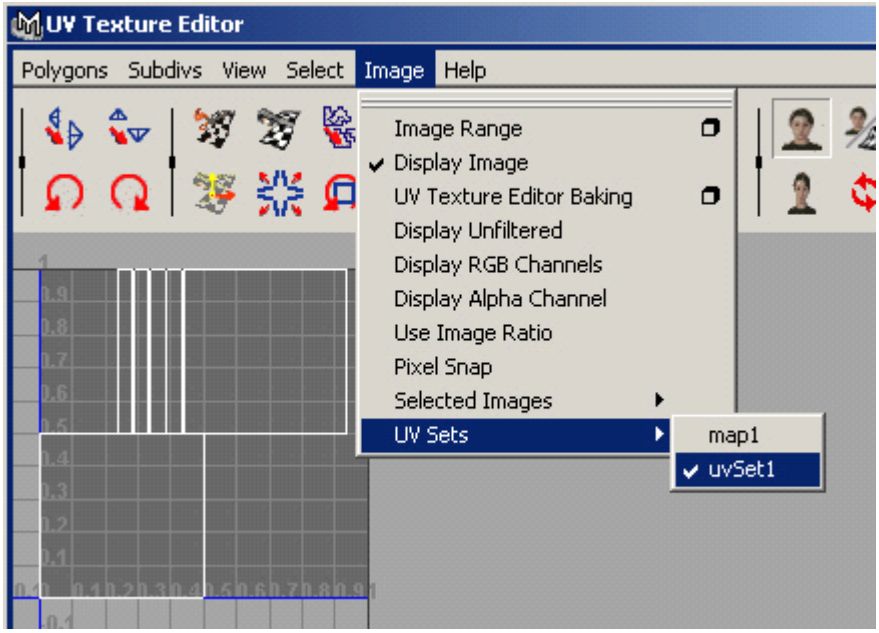
This option generates automatic mapping coordinates for lightmaps. We will layout the UVs for a 512x512 lightmap



After clicking **Project** or **Apply** we can check out the result: Open **[menu|window|UV Texture Editor].**
Choose the new UV set:



Remember, if you want to re-generate automatic mapping (if you want to set different parameters etc.) you have to turn off ***Create New UV set*** otherwise you will end up with a third UV set.

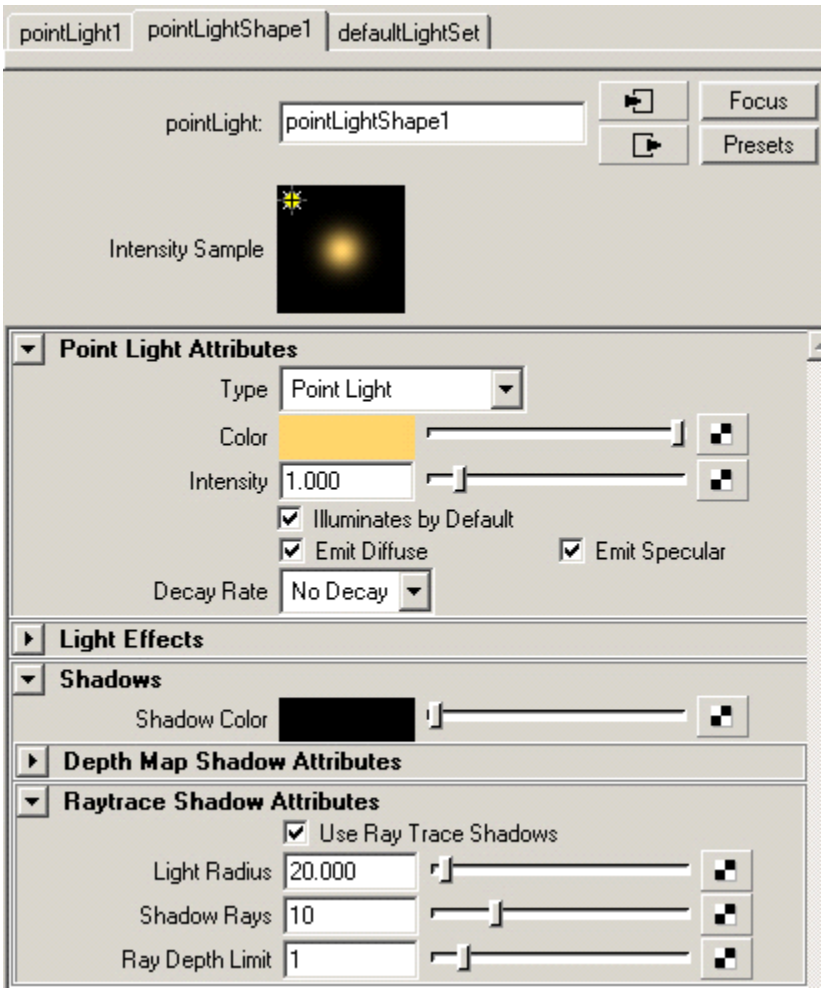


4. Separating materials

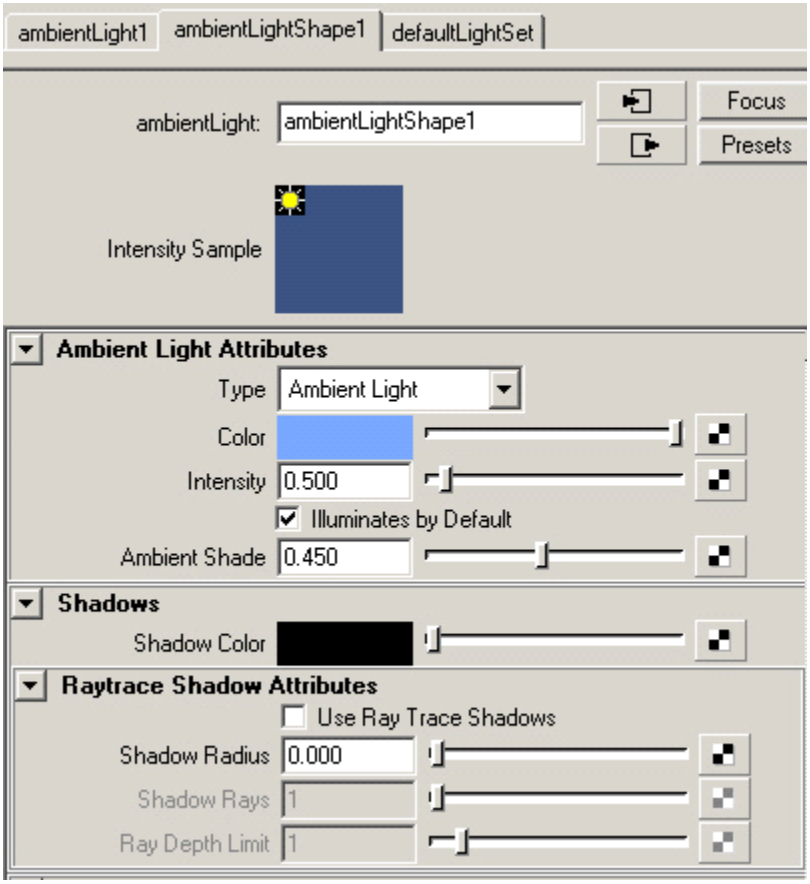
If different lightmap objects share common material (for example if you have two walls with the same brick material and you want each of the walls to have a separate lightmap), you have to copy those materials so that you have two identical but separate materials, otherwise you will have only one lightmap on all the objects with same material. Easiest way to do that is to export selected objects to a separate .mb maya file and then to import it back into the scene. The objects and their materials will still look the same but they will actually carry different material ID than they had before this operation. In our example it is not necessary as two objects in our scene have different materials.

5. Setting up lighting

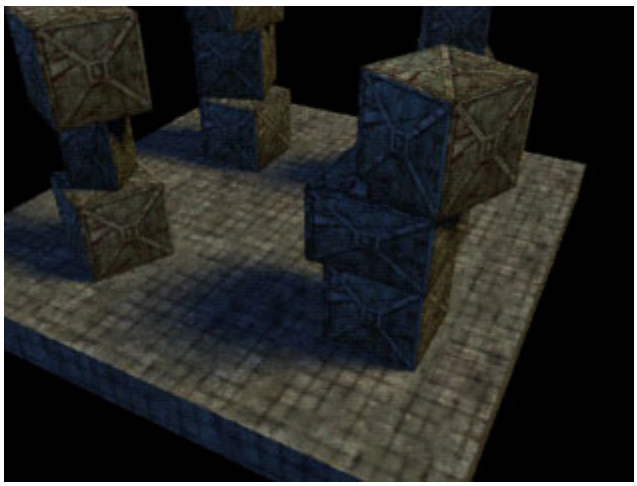
There are many ways to light your scene, we will show you the easiest way using simple point light source casting soft shadows. Add a point light source **Create\Lights\Point Light** with the following parameters:



Create another blue ambient light **Create\Lights\Ambient Light** with the following parameters:

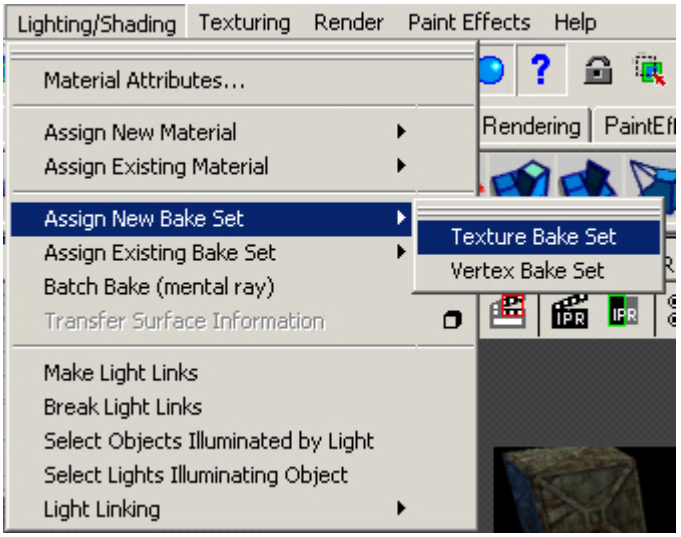


Place the lights reasonably in the scene, click render: **Render View**  and you will end up with something like this.

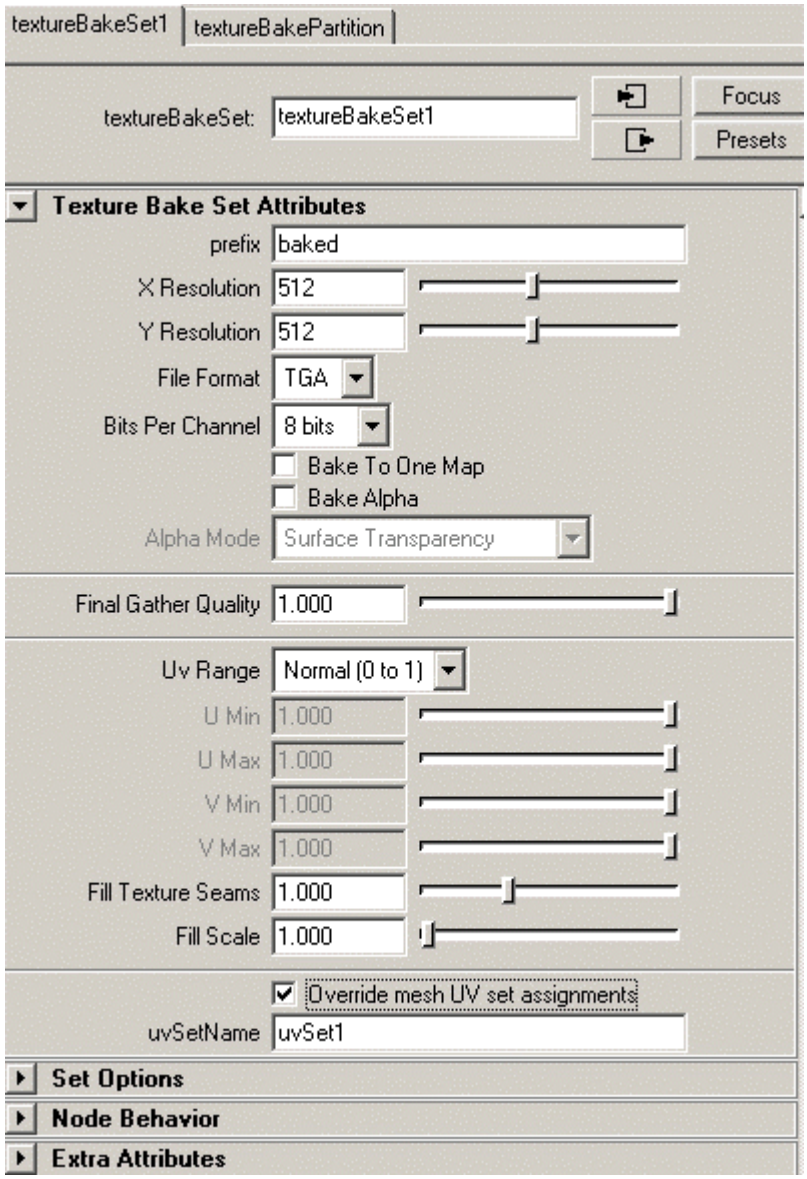


Change Maya module to rendering.

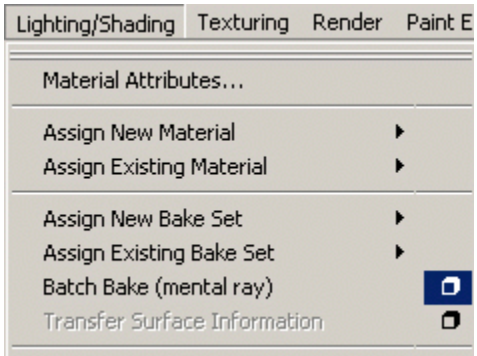
Select both objects and choose **Assign bake set**.



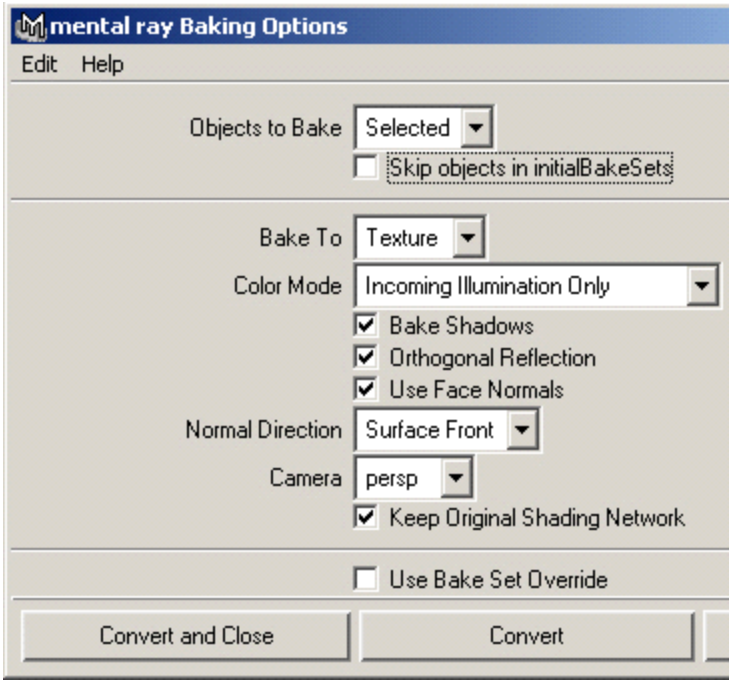
Now you can check all the settings for rendering of the **bake set.**



It is now time to finally generate the lightmaps. Open:

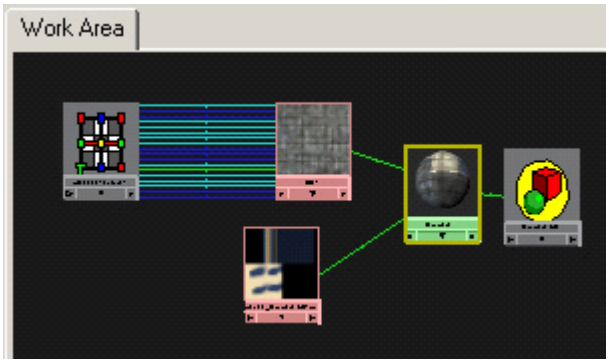


Set the following parameters and hit **Convert**.

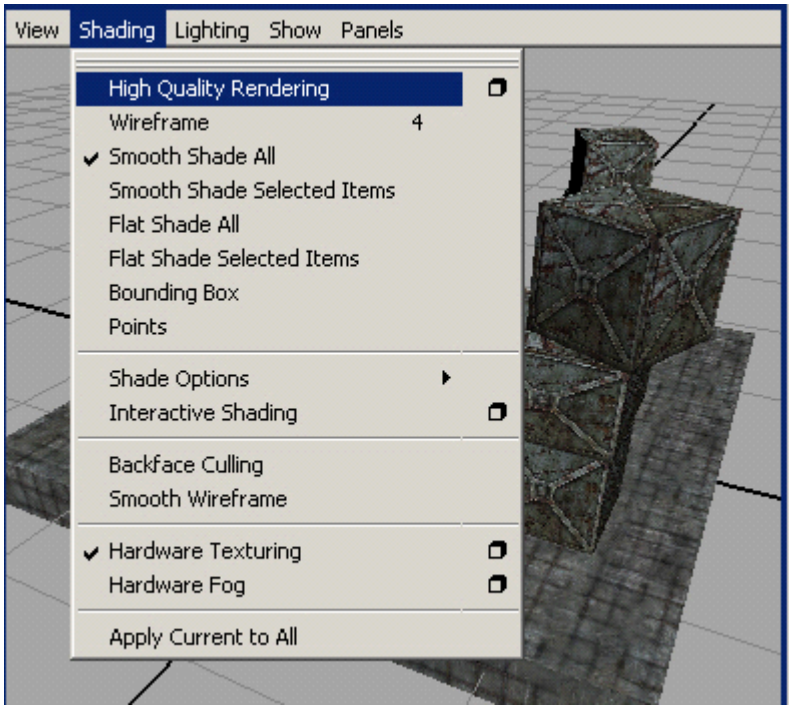


It takes some time for Mental ray to generate the bitmaps with lighting. Keep in mind that depending on lights, settings and geometry that you render lightmaps for it might take hours!

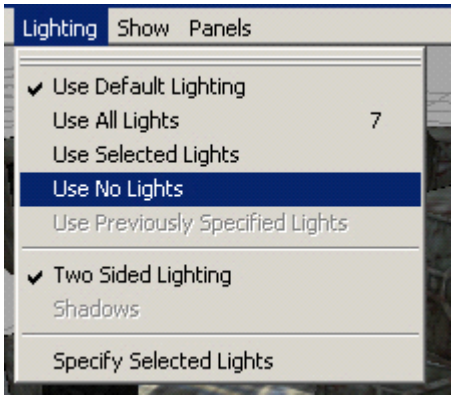
The rendered lightmaps can be found in the subdirectory of your Maya project "..\mentalRay\lightMap". Create a new "file" node, load the rendered bitmap and link it to object material as "**ambient color**" attribute.



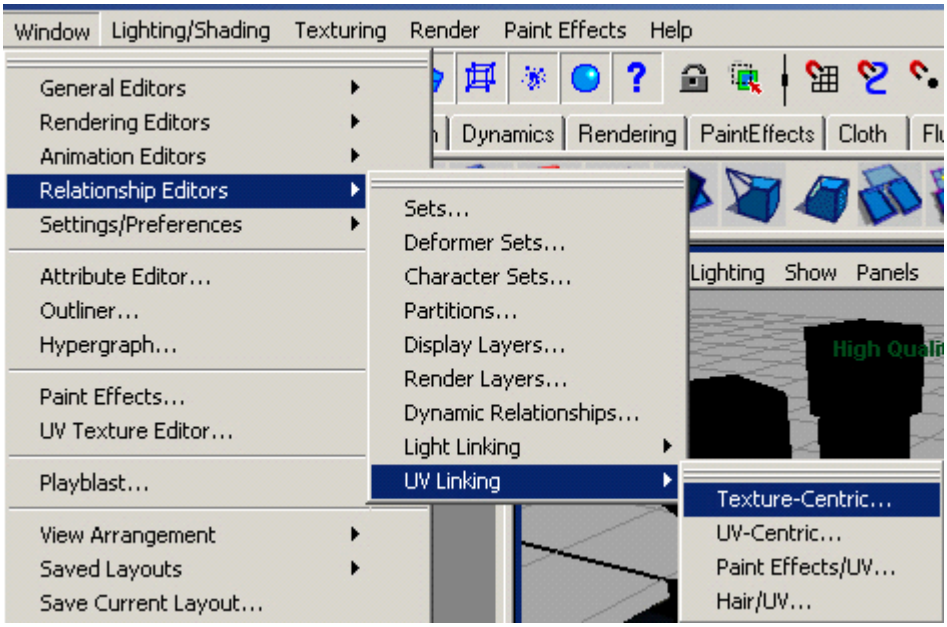
If you want to see lightmaps on your geometry in Maya set the following



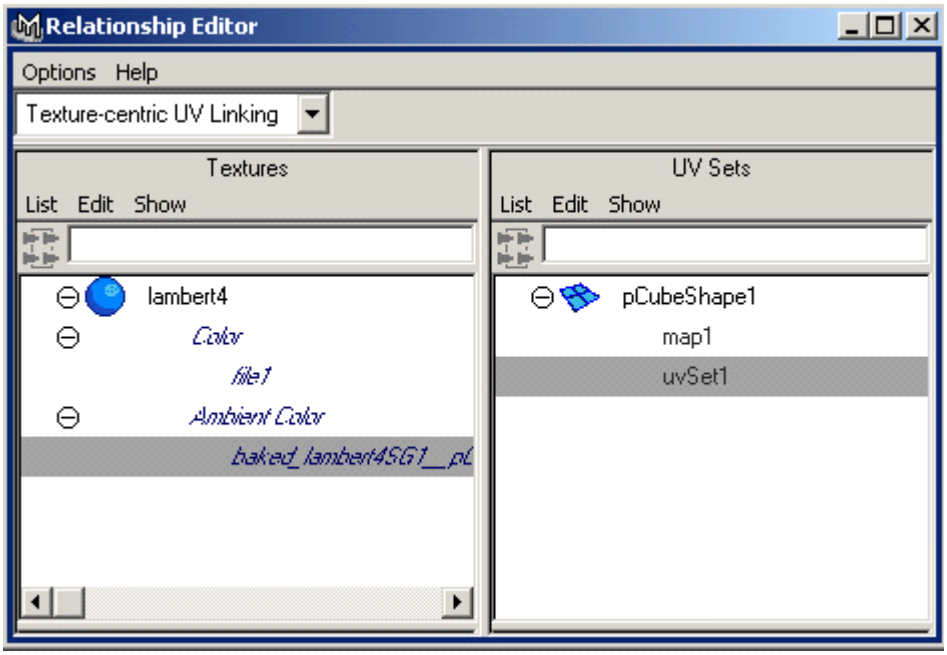
and



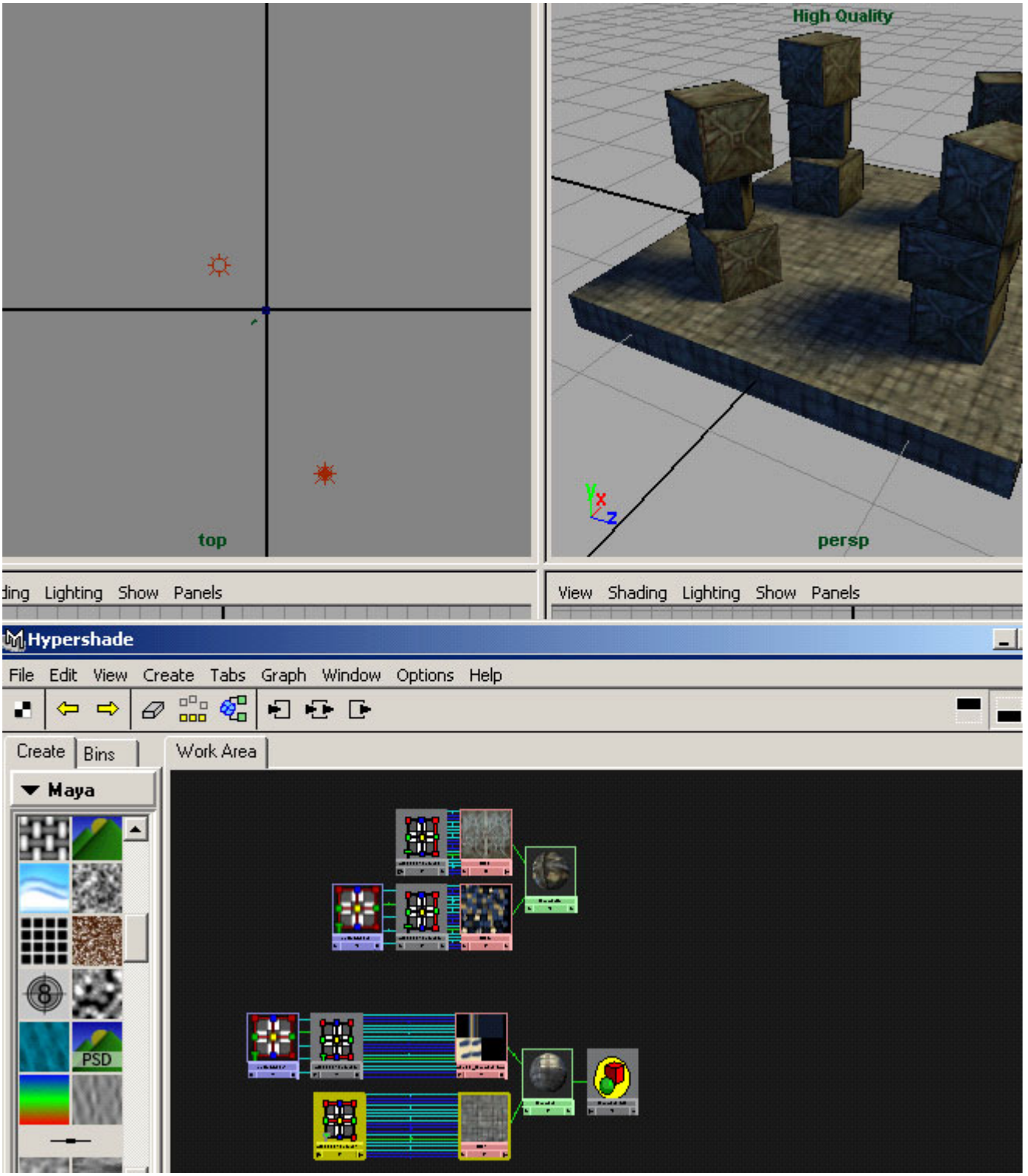
The result is strange, because we need to link the lightmaps to second UV set.



Select the object and in **Relationship Editor** click on the lightmap name in left window and on uvSet1 in the right one.



This is how the final scene and object materials look like:



The scene is ready to be exported to PainEd!