Chapter 5 SURFACING - Workspace 2

Chapter 5 SURFACING - Workspace

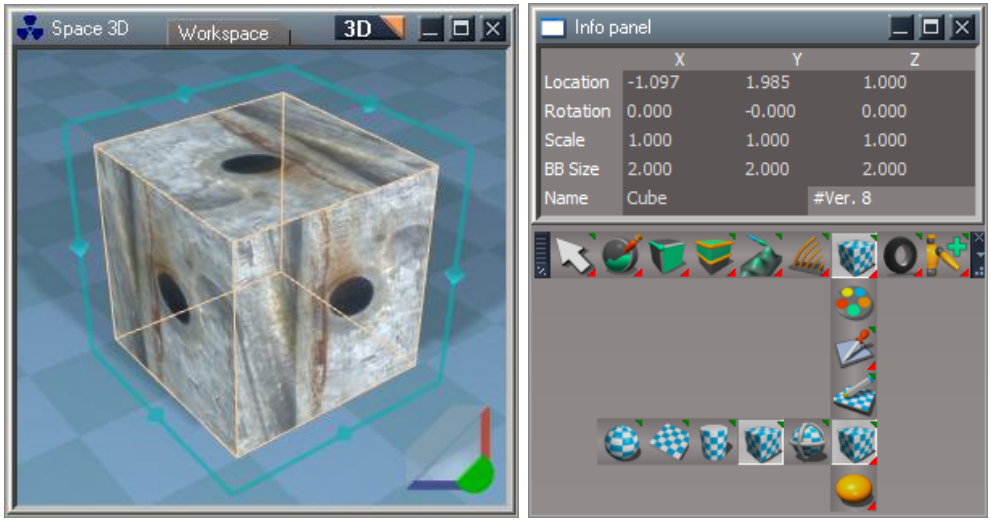
5.1 UV Projections

Although many materials can be recreated using 3D procedural shaders, there are a number of materials that cannot, such as **patterned cloth** and **paint effects**, **transfers**, **decals and logos**, and surface relief materials such as **tiled surfaces**. For these materials, either 2D procedural shaders or image maps must be used, but this creates a problem of its own. When mapping 2D procedural shaders or images to a 3D surface, they become distorted or can "break" or "tear" in areas with complex or sudden changes in curvature.

UV mapping is a general solution to the problems encountered when trying to map 2D data onto a 3D surface.
U and V are virtual co-ordinates with U representing the horizontal component of an image or the x component of a 2D function, and V representing the vertical component of an image or the y component of a 2D function. Although U and V are horizontal and vertical components of a 2D source, when mapped to a 3D object their orientation is arbitrary.

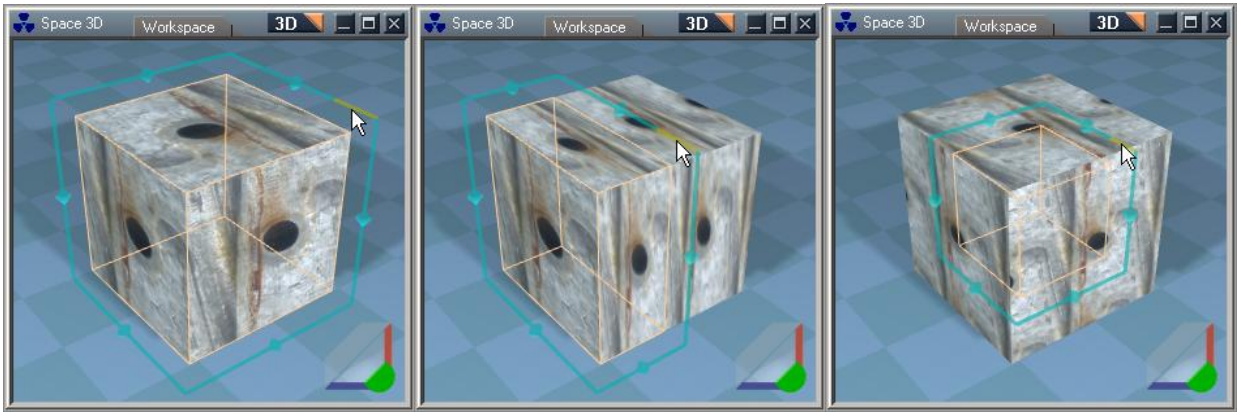
UV mapping takes 3D geometry and assigns 3D points across the surface of the geometry to specific U and V co-ordinates using algorithms which either flatten or unfold the 3D geometry to make a 2D map. When the surface is rendered, the x and y values for a 2D procedural shader or the horizontal and vertical co-ordinates of pixels in an image map can be determined by interpolating the values from the UV map for the corresponding points on the 3D surface.

When a default UV mapping type has been applied to an object a blue control cage shows in the Workspace and this can be used to adjust the rotation and scale and position of the UV mapping projections as a visual aid in positioning textures exactly on faces where they are required. The UV Mapping widget control cage can also be used in conjunction with the object Info panel by using numerical input to adjust the scale, position and rotation of the projection on the objects surface.
As the mouse passes over the various parts of the control cage the activated part will highlight yellow.
A right click in the Workspace window will exit the tool.



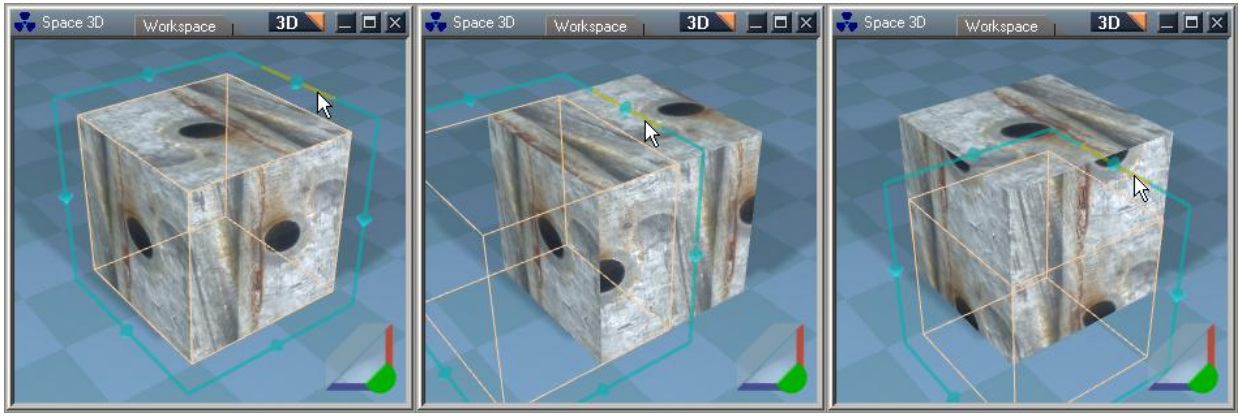
UV control-cage , Info panel , toolbar location of UV projection types

The sections nearest to each of the corners will scale the object in the direction you drag them or scale the object proportionately when you hold down both mouse buttons at the same time and drag.



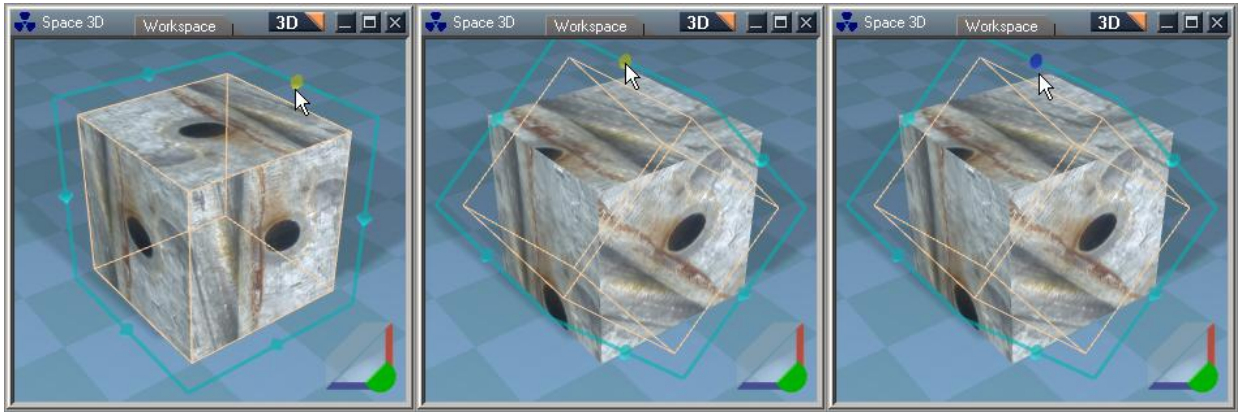
UV control-cage adjustment scale highlighted yellow.
Left mouse drag scales in one direction, left and right mouse held and drag scales proportionally in all direction.

The middle sections will move the object in the direction you drag them. Holding down Left and Right mouse button allows for free movement in all directions.



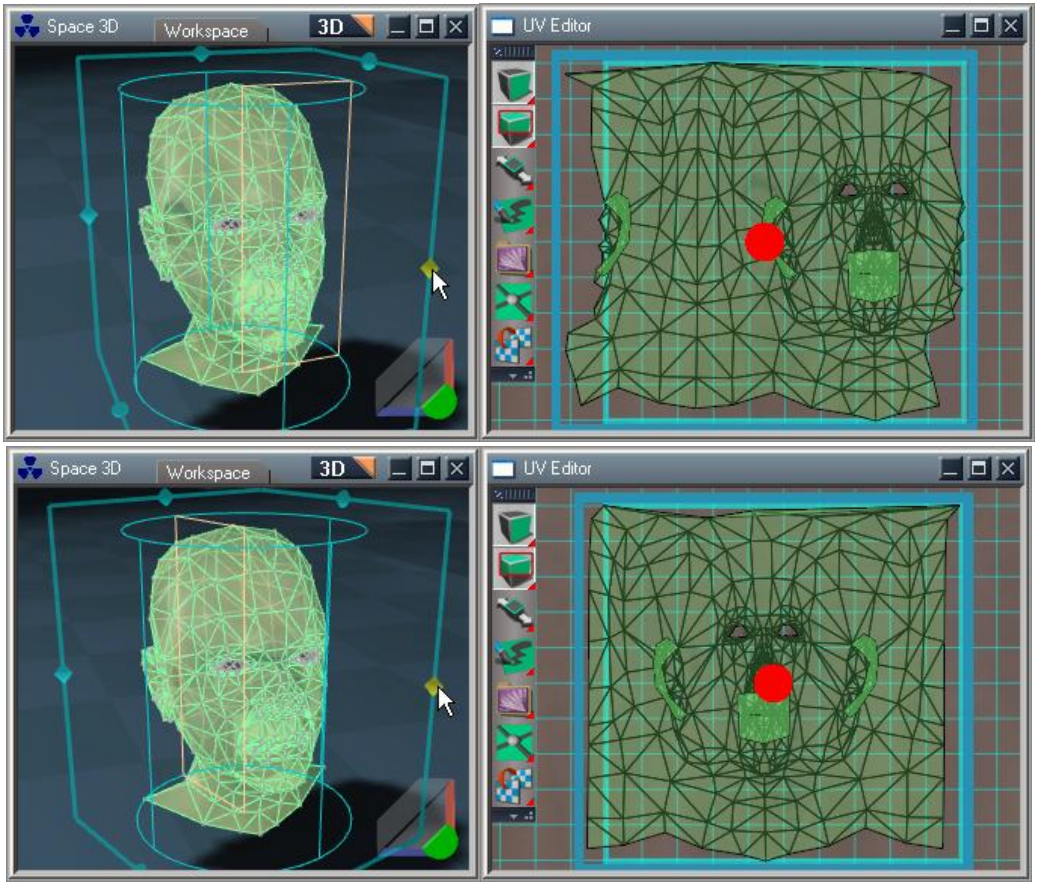
UV control-cage adjustment move highlighted yellow.
Left mouse move the UV projection in one direction . Left and Right mouse button allows for free movement in all directions

The rotation diamonds in the middle of each bar will rotate the object around its various axes.
As you mouse closer to the diamonds you will notice that they darken in color to match the axis they will rotate around. Blue is Y, Green is X, and Red is Z.



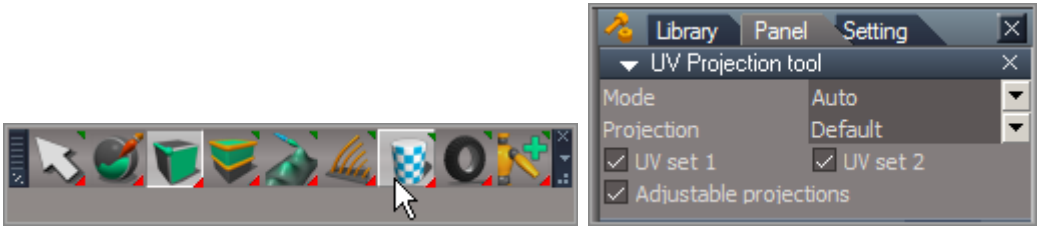
UV control-cage adjustment rotation highlighted yellow.
The rotation diamonds in the middle of each bar will rotate the object around its various axes.

The inside part of the control cage which shows on the object projection shape represents the seams where the texture is split, this provides a visible interactive feedback and can be adjusted to decide where the split should be located. Below shows the differences in the UV editor when a seam has been adjusted by rotation as an example to illustrate the behavior. If the Model is a head as in the example then you could arrange the seam so that it occurred on the back part of the head to make the mapping and texture process easier.



Cylindrical Mapping Seam adjusted by rotation

The Standard UV projections have a panel to adjust the various projection modes and other options .
The UV projection tool panel will show in the stack when a right click is performed over the icon in the toolbar.



right click over the icon in the toolbar UV projection tool panel shows in the stack.

- **Mode** : relates to how the mapping matrix is calculated and whether the tool prefers mesh modifier to allows UV mapping of procedural objects or changes to the input mesh.
- **Auto** : means the UV tool will select and use the modifier for the UV projections if the object is procedural ,

or if the object isn't procedural and just a regular mesh the UV tool will use the mesh for the UV projections.(default Mode is Auto).

- **Mesh** : UV tool will use the mesh for the UV projections (regular mesh).
- **Modifier** : UV tool will use the modifier for the UV projections (procedural objects).

- **Projection** :

- **Default** : Performs mapping for a plane which picks the biggest bbox face as the projection surface, for the other mapping projection types xy is squared and z remains unchanged.
- **Square** : Performs mapping so that the projected pixels have 1:1 aspect ratio. a plane is squared, and a cube is equally sized etc)
- **Tight** : Only the BB Size(bbox) is used, for example if cylindrical mapping is applied to a long box the mapping will be elongated.
- **UV set 1** : When checked uses this working set, unchecked means it doesn't use the set.
- **UV set 2** : When checked uses this working set, unchecked means it doesn't use the set.

- **Adjustable projections** : Checking this uses the state when you previously mapped a surface so you can go back and re-adjust the projection .(default is checked).

Note - For More advanced uses this setting can be also used to transfer projection to another shape (adjustable must enabled) if you map 1st object and then switch to adjust and add other objects to selection, the added objects will inherit the 1st objects mapping matrix.

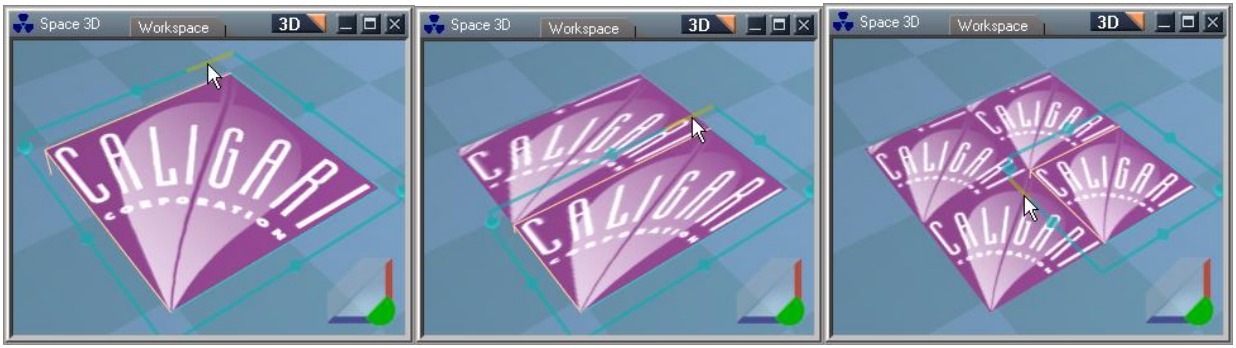
5.1.1 Standard UV Mapping Types

trueSpace offers a number of UV mapping options and also gives control over the repetition of image maps and procedural patterns across an object's UV space. All UV mapping options can be scaled and rotated while the projection tool is active to give the most appropriate fit. A right click in the Workspace window will exit the tool.

 **Planar Projection** 

A UV map is applied that "flattens" the UV space down onto a plane. The plane's orientation can be altered by rotating the object's UV space. The result is that the material texture appears to be projected through the object in a direction perpendicular to the plane.

Scale



UV control-cage adjustment scale highlighted yellow
Middle UV is scaled in one direction. Right UV is scaled in both directions

Move

UV control-cage adjustment move highlighted yellow

Left UV is moved in one direction. Right UV is moved in both directions

Rotate

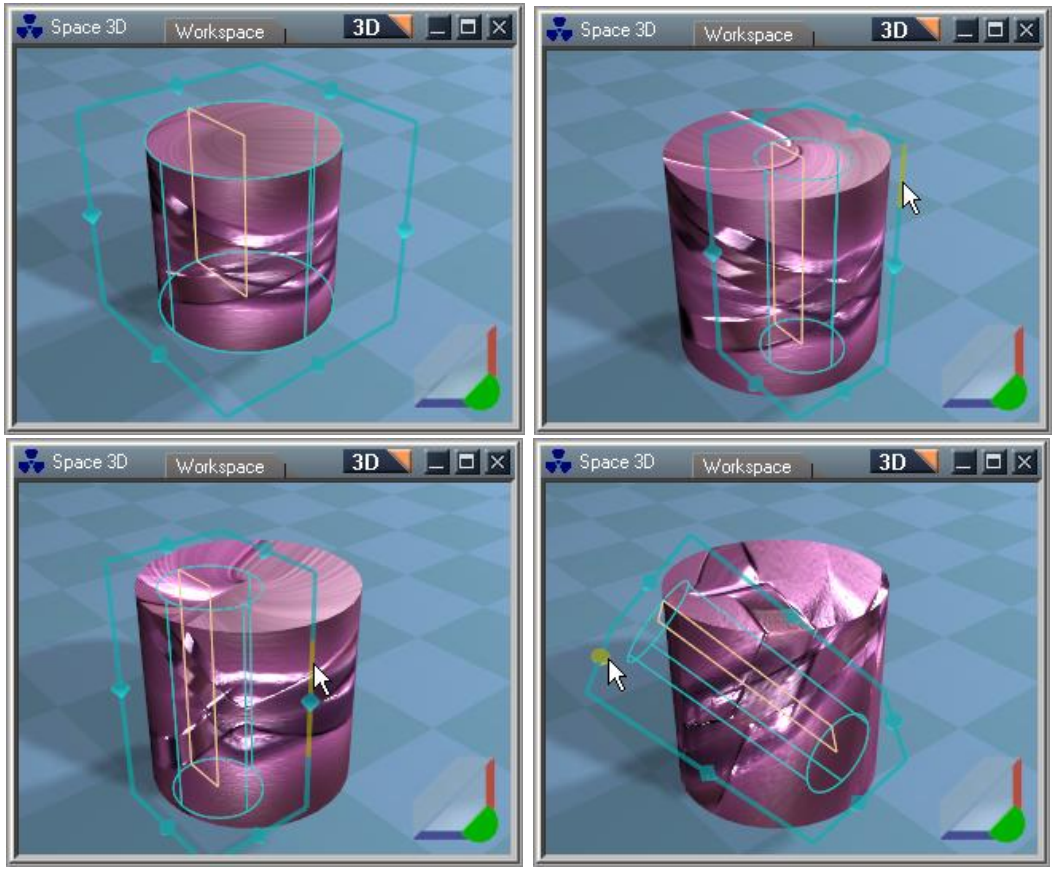
UV control-cage adjustment rotate highlighted yellow

Left UV is rotated in one direction. Right UV is rotated in both directions

Adjusting the UVspace for Planar mapping is straight-forward. The control cage is built from very similar parts. Depending on the axis (Blue [Y] or Green [X] rotation tools), you can move, scale or rotate in any of these axis directions. Planar mapping is 2D mapping, making X and Y axis adjustments very convenient. The image above shows selected highlights for different tools, but each side of the control-cage contains the same tools for move, rotate and scale. Mouse-over the various parts of the control-cage to get a feel for how they react.

Cylindrical Projection

A UV map is applied with a cylindrical volume that surrounds the object. The cylinder's orientation can be altered by rotating the object's UV space. The result is that the material texture appears to be projected inward from the cylindrical space to the center of the object.



UV control-cage adjustment tools being used for scale , move, rotate, respective parts highlighted yellow.

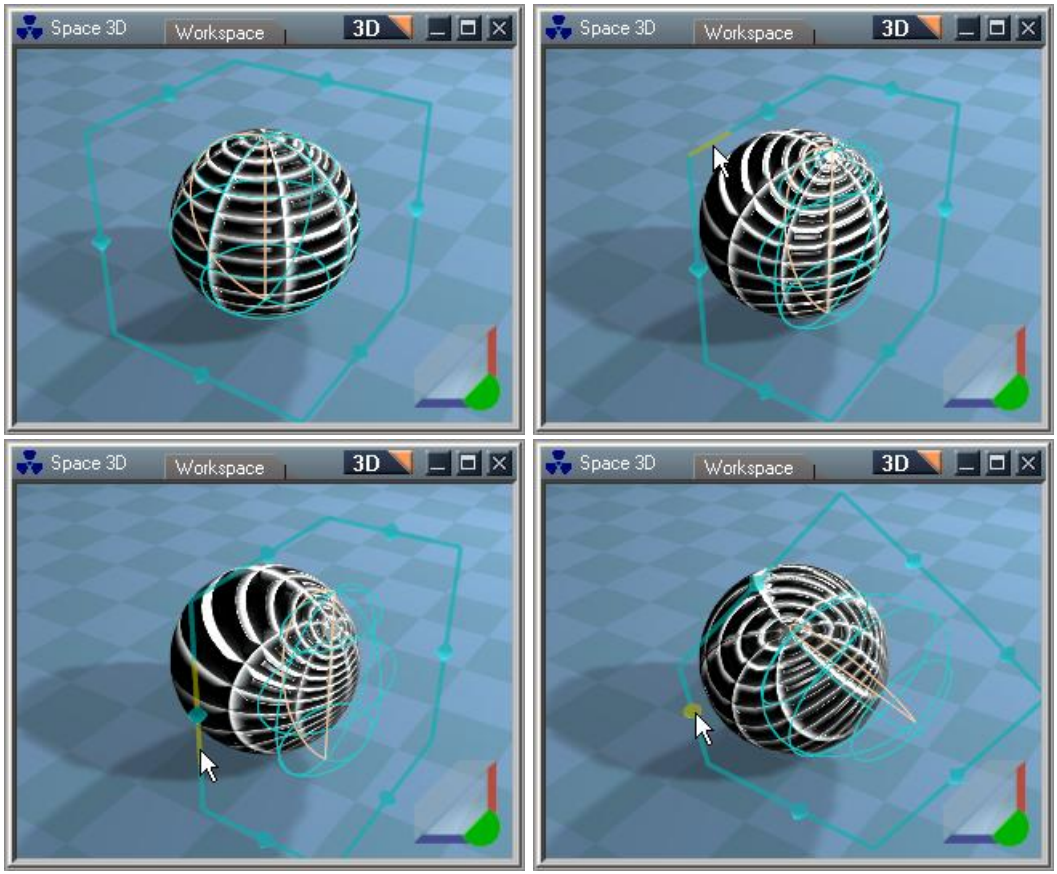
Adjusting the UV space for Cylindrical mapping is straight-forward with a new twist. Because we are now using a 3D projection here, we have access to the Z-axis for adjustment and fine-tuning.

Each side of the control cage is built from very similar parts. Depending on the axis (Red (Z), Blue (Y) or Green (X) rotation tools), you can move, scale or rotate in any of these axis directions. Cylindrical mapping is 3D mapping, making X, Y and Z-axis adjustments very convenient. The image above shows selected highlights for different tools, but each axis/side of the control-cage contains the same tools for move, rotate and scale. Mouse-over the various parts of the control-cage to get a feel for how they react.

The control cage accepts both L-Mouse and R-Mouse presses for the Move and Scale parts of the widget so the results can be different according to which button is pressed on the widget. The Rotation part of the control cage widget only accepts L-Mouse button actions.

Spherical Projection

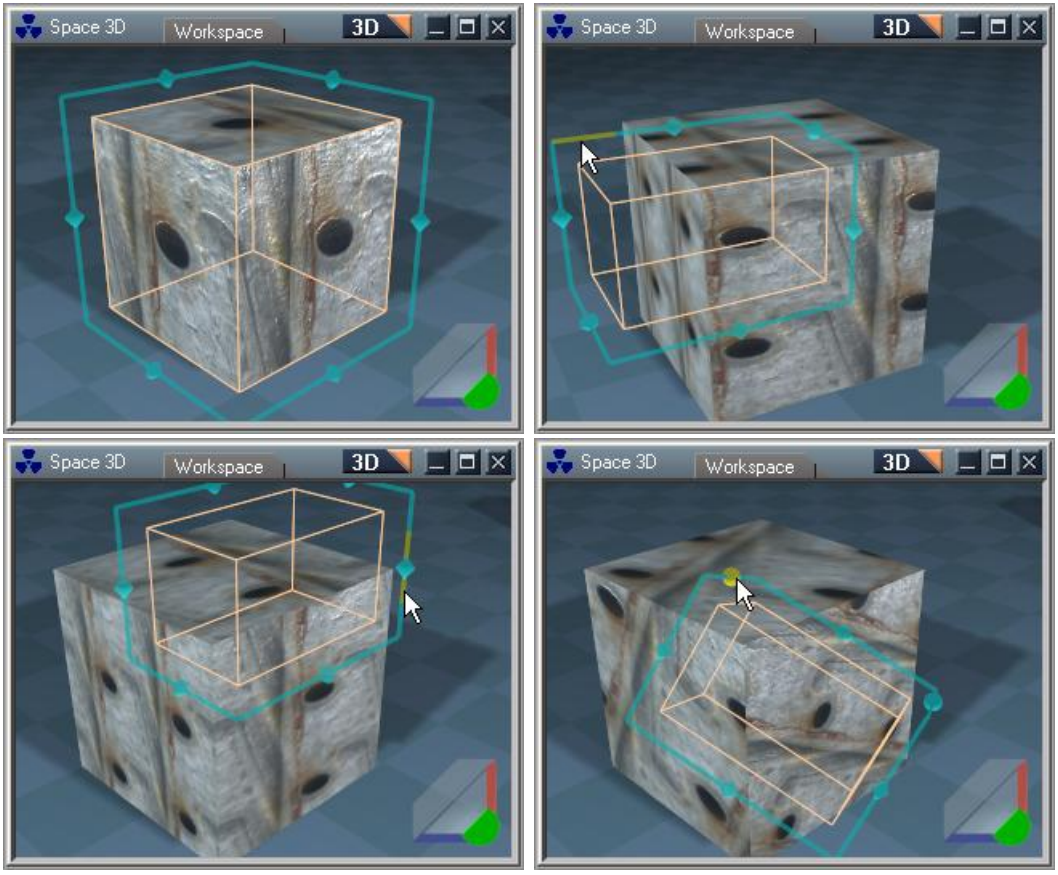
A UV map is applied with a spherical volume that is translated to an evenly spaced 2D grid with divisions representing the latitude and longitude of the sphere. The sphere's orientation can be altered by rotating the object's UV space. The result is that the material texture appears to be projected inward from the spherical space to the center of the object.



UV control-cage adjustment tools being used for scale , move , rotate, respective parts highlighted yellow.

Cubic Projection

A UV map is applied with a cubic volume where each face of the cube is a complete planar projection. Manipulating the control-cage for UV allows you to fine-tune the space to best suit the object.



UV control-cage adjustment tools being used for scale , move , rotate, respective parts highlighted yellow.

Adjusting the UVspace for Cubic mapping is straight-forward. The control cage is build from very similar parts. Depending on the axis (Red, Blue or Green rotation tools), you can move, scale or rotate in any of these axis directions. The image above shows selected highlights for different tools, but each side of the control-cage contains same tools for move, rotate and scale.

Shrink-Wrap

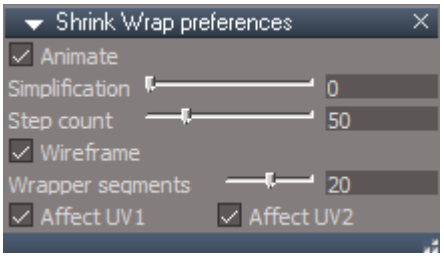
A spherical volume larger than the chosen object is gradually shrunk around the object until its surface roughly conforms to the surface of the chosen object. As the volume is shrunk and distorted a spherical UV mapping is distorted in a similar manner. The result should be a UV projection that gives a better representation of the object's surface than a standard spherical projection.

Cubic UV mapping for object on left, while Shrink-wrap with default settings used on the right hand version.

Second variation of Shrink-wrap UV Projection with settings at step 128 and segments at 40

With Shrink-wrap UV Projection, there is no control-cage to adjust. Adjustments are made via the Shrink-wrap Preferences panel.

Shrink-Wrap Preferences Panel



Shrink Wrap preferences panel

- **Animate** : toggles wrapper object animation and displaying.
- **Simplification:** internal target mesh simplification for faster computation.
- **Step count:** maximum step count.
- **Wireframe :** toggles wireframe display of wrapper object.
- **Wrapper segments :** latitude and longitude segment count of wrapper object initial sphere.
- **Affect UV1 :** toggles UV1 remapping.
- **Affect UV2 :** toggles UV2 remapping.

5.1.2 Default UV Mapping

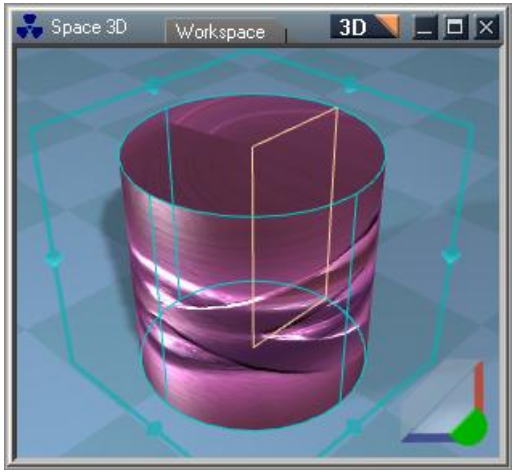
trueSpace assigns UV mappings to objects as they are created, as follows.

- **Spheres** and **Ellipsoid** objects: Spherical Projection.
- **Cylinders:** Cylindrical Projection.
- **Lathe objects:** A modified Cylindrical Projection bent around the object's profile.
- **Cubes** and **Cubic** objects: Cubic Projection.
- **Torus** and **Helix:** Both special cases whereby a Cylindrical Projection is bent to conform to the object's circular shape.
- **Planes:** Planar mapping with the UV space oriented to the plane.
- **Swept objects:** Planar mapping with the UV space oriented to the swept curve for the top and bottom surfaces and a modified cylindrical mapping across the object's length.

5.1.3 Applying UV Mapping

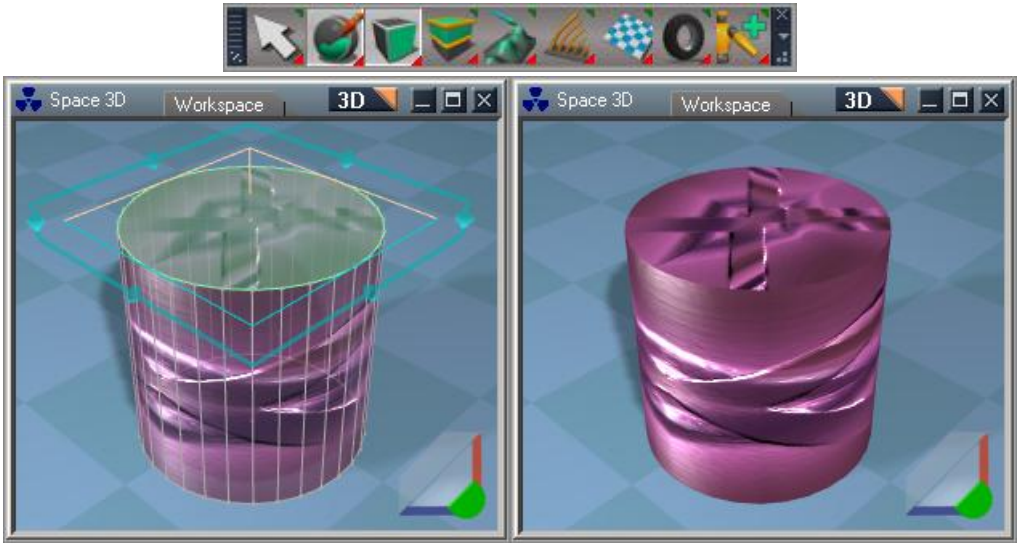
You can apply UV mapping in 1 of 2 ways:

1. To the entire object by selecting the object and then selecting a suitable UV mapping option.



Cylindrical UV mapping applied to entire object

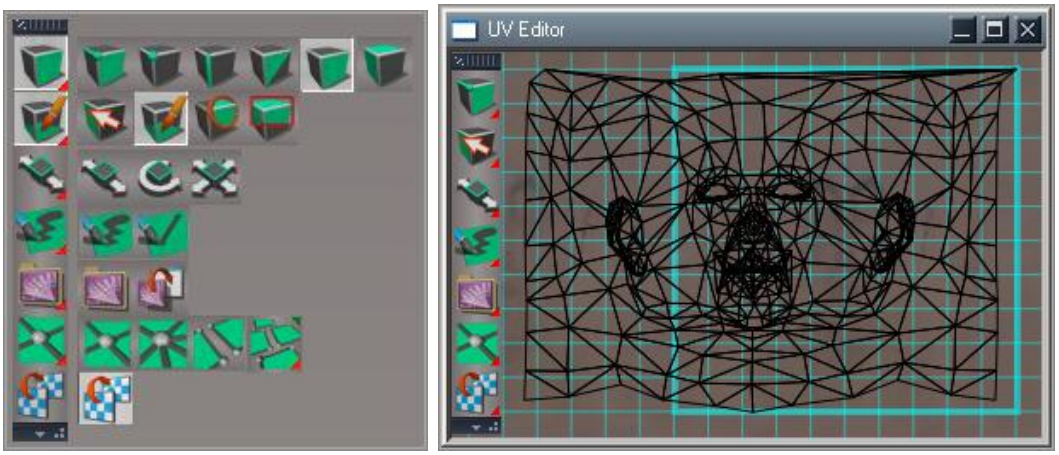
2. By selecting portion of the objects using the polygon and vertex selection tools and applying separate UV mapping options to different areas.



Planar UV mapping applied to the top face selection in Point Edit mode

5.2 UV Editor

 **UV Mapping Editor**



UV Mapping Editor

The **UV Mapping Editor** affords you placement of UV mapping within the texture area using point, edge, triangle and polygon selection and manipulation tools. Selections in either the Workspace while in Point Edit mode, or in the UV Editor itself, are synchronized. What you select in one becomes selected in the other.

The UV Editor allows you to mark textures with a brush or with lines. You can edit textures in a 2D Image Editor of your choice. Save your texture from 2D Editor and it is automatically applied to your model in Workspace. The changed/edited image also shows up in the UV Editor with changes intact. When you combine these capabilities, you are presented with finite control over both texture and UV space.

You have the ability to Weld vertices of UV mapping together, allowing you to form compact UV shapes. Part and parcel with ability to Weld, you also have the ability to Slice and dice your UV mapping in the UV Editor. This allows you to slice and move UV distinct shapes to separate areas of the UV space. One image texture can contain all the required textures for parts of your objects.

The UV Editor is dockable inside the trueSpace interface, or it can float as a separate window. However you decide to place the UV Editor, you will enjoy the ability to edit UV texture and mapping in real-time.

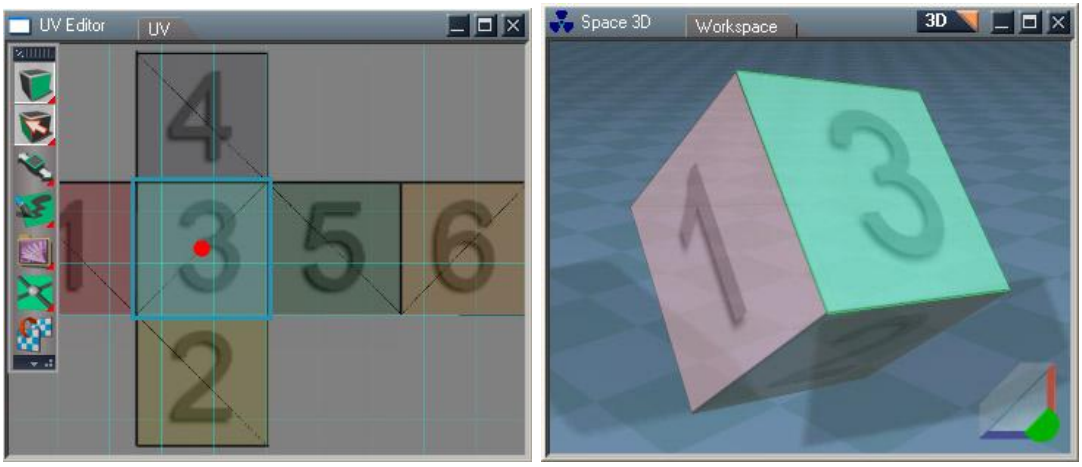
Navigation in the UV editor

The UV editor window can be zoomed and panned.

Drag middle mouse button (mouse wheel) to pan around the UV space.
If you do not have a middle mouse button then ALT- L-mouse hold and drag will pan the UV space.
Rotate the mouse wheel to zoom in and out.
When resizing the UV editor window it will maintain a constant zoom and aspect ratio.

Selection Tools

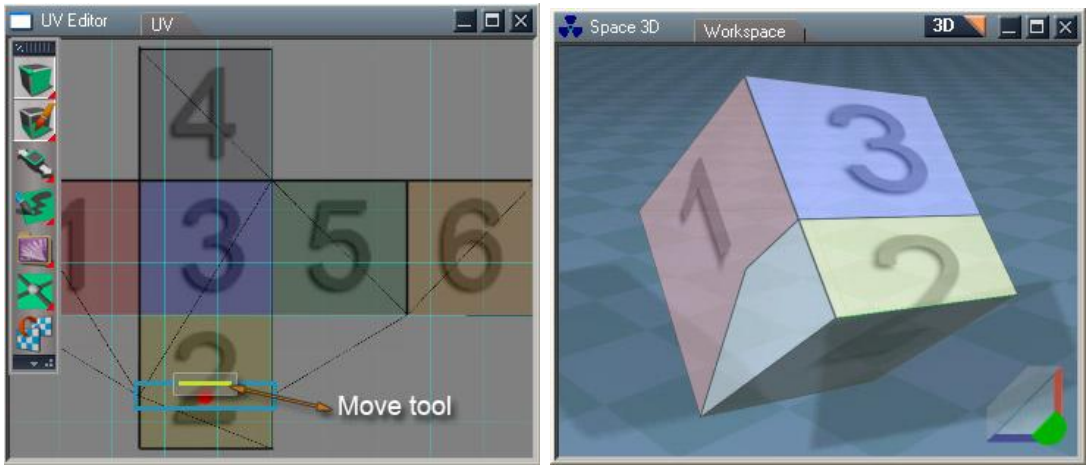
You can select geometry either in the UV Mapping Editor or in the Workspace. In the screenshot below you can see a simple cube with a texture map numbered by face. The face numbered '3' was selected in the Editor using the face select tool. The face could have been just as easily selected in the perspective view window by entering point edit mode and using the face select tool. Note that the face is highlighted both in the Editor (left image) and the Workspace (right image).



Face "3" selected in UV Mapping Editor and in the Workspace view

Manipulation Tools

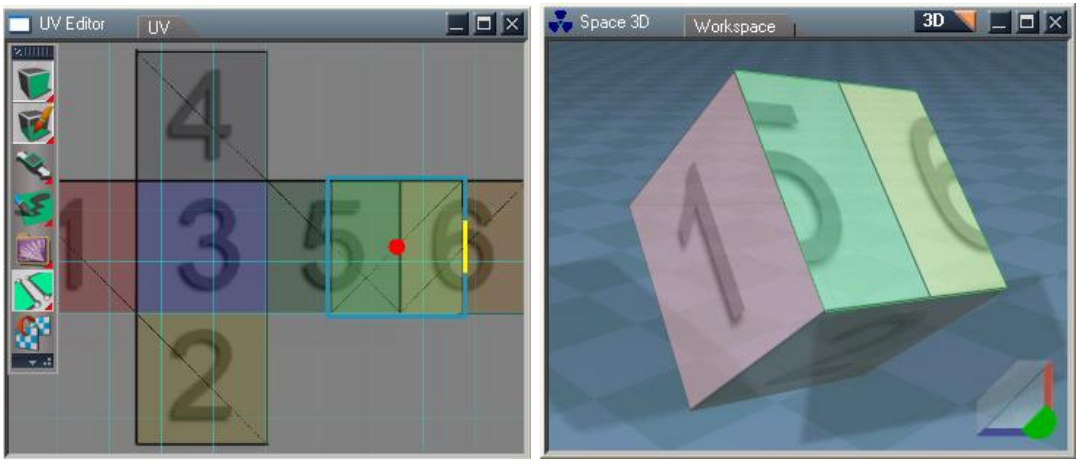
With geometry selected you can easily manipulate UV coordinates by using the move, rotate, and scale tools found in the UV Mapping Editor. You can also weld and Slice UVs to achieve a lot of variety in your mapping technique, depending on your needs. To demonstrate, we have grabbed the bottom edge of the face numbered '3' on our cube from the previous example and dragged it towards the bottom edge of the face numbered '2'. Notice how the texture moves across the faces on the model - with the '3' and '2' faces compressed on one side and how the texture on side '1' and '1' has been shifted and compressed as well.



Moving the texture by moving an edge in the Editor

Though this principle is simple on the face of it, you can achieve some very complex mapping results on more complicated objects. Imagine taking a human head model and mapping it to a flattened, collaged image created from photographs of a person. Clearly there is no way that automatic mapping will ever suffice to properly align the UVs - so you will use this technique to tweak the mapping to achieve a perfect fit.

You can also Slice UVs from their neighbors so that your transformations will only affect the selected geometry. In the example below we have detached face '3' from its neighbors and shifted the UV coordinates of the face to the right a bit. Notice how both the numeral '5' and '6' now appear on face '3'. For precision you can move objects freehand or just grab a selection to slide it orthogonally.

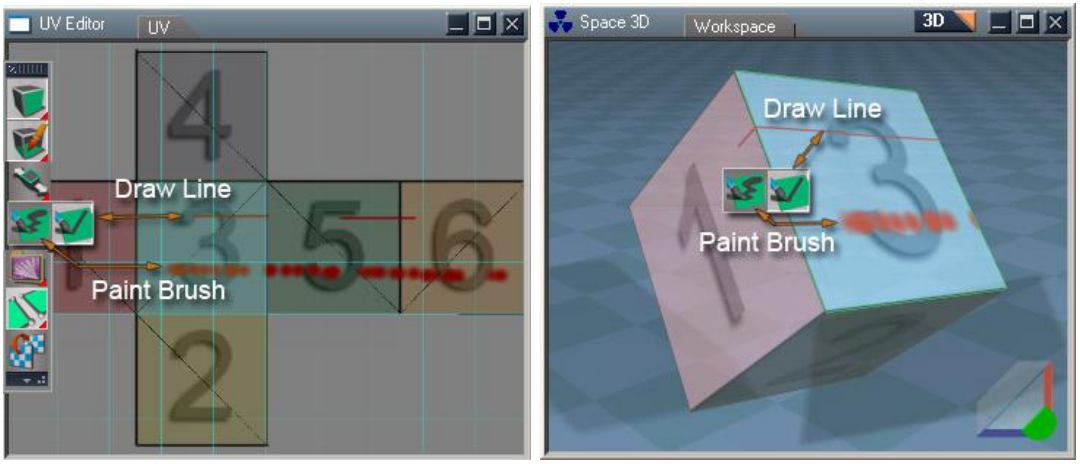


Face is Sliced and moved in UV space

Texture Painting

The paintbrush tools found in the UV Mapping Editor allow you to paint onto textures. Just select **Paint color** from the UV Editor Preference Panel's "Color" aspect and paint right onto your texture map. Below we have painted a couple of stripes onto our example cube to show the tools in action.

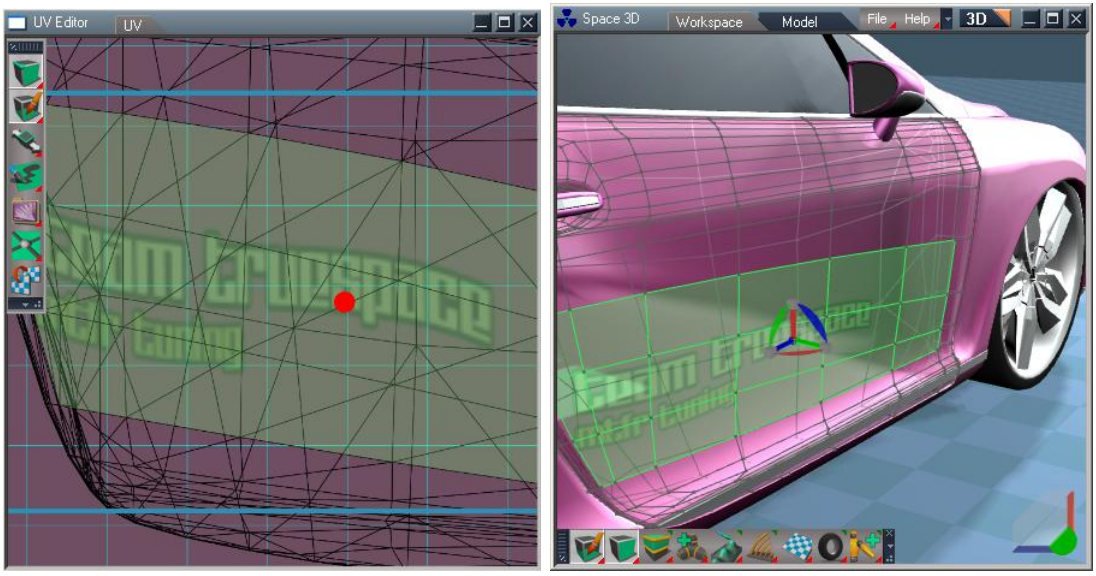
You would use this tool for texture mark-ups, useful for when you edit a texture or create the texture map in an external 2D Image Editor to fine-tune areas where you have used the Draw Line and Paint Brush tools on.
You can Adjust brush color and size in the UV preferences panel.



Some examples of Draw Line and Paint Brush tool use

Exploring UV Mapping

Though you have so far seen only simple examples of using the UV Mapping Editor to lay out UVs, the techniques for using this tool apply to more complicated models as well. On the car model below the faces representing the door are mapped in the UV Editor and shown highlighted in the Workspace.

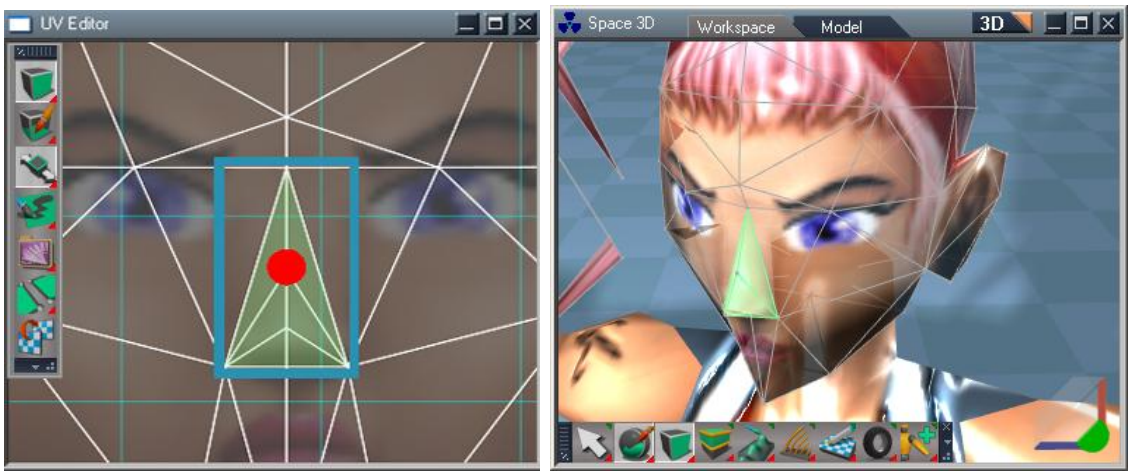


Mapping for a more complex model

The Editor analyzes the selected object (and its sub-components if it is a group) and displays the UV map for each set of coordinates found. You can switch between the textures by clicking on the editor's Preference Panel's 'Previous material' and 'Next material' buttons.

On more complex models you will often need to use the middle-wheel 'zoom' ability to zoom in or out in the UV Editor. You can also adjust the wire colors from the UV Editor Preferences Panel to change how the UVs are displayed in the Editor.

Applying UV maps to organic models such as characters is one of the most difficult tasks for many 3D artists. In the example below, UV Editing of the Tank Girl's face takes place.

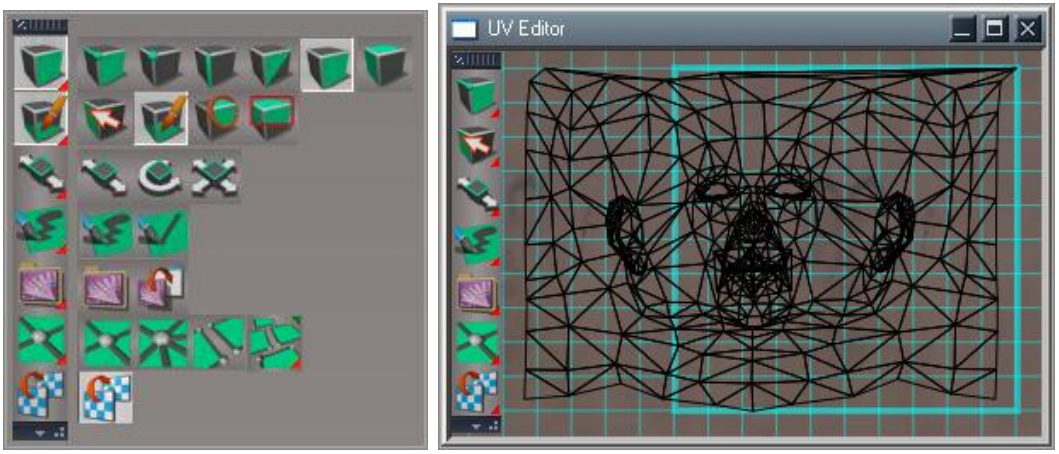


The techniques described apply equally to organic models

Don't let complex geometry stop you from making great maps – select some faces, apply a map, and tweak the UVs. Mapping can be difficult to master but, once you get the hang of it, the satisfaction of turning out a well-mapped model is worth the effort.

5.2.1 UV Mapping Editor Tools

The toolbar attached to the left side of the **UV Mapping Editor** contains tools for manipulating the UV for the selected mesh:



UV Mapping Editor

Select Element tools:

 **Select Context**

 **Select Vertex**

 **Select Edge**

 **Select Triangle Face**

 **Select Face**

 **Select Patch** patches are selected by using the pick and move selection tool  .

These tools are used for selecting by Context, Vertex, Edge, Triangle Face or Face. Left-click the mesh to pick individual vertices, edges, or faces, use CTRL+click to add to the existing selection. Vertices, edges, and faces will be highlighted in light blue (by default) as you move the mouse over them, and green (again by default) once selected.

Selection Tools

 **Select and Move Nearest**

Left-click and drag to select the vertices, edges, or faces under the cursor and move them immediately.

 **Paint Select**

Left-click to paint on vertices, edges or faces you wish to select.

 **Lasso Select**

Left-click and drag to draw a lasso around vertices, edges, or faces to select.

Rectangle Select

Left-click and drag within the UV Mapping Editor window to draw a rectangle, selecting everything within it.

When working with the selection tools, using the Control-Key and Shift-Key allows for selection combining. Use the Control-key while dragging your mouse will merge/combine selected elements. Using the Shift-key while dragging mouse will deselect elements.

Manipulation Tools

Note: For each of these tools, you must have selected part of the mesh before using them. Also, it is important to note that the UV mapping changes, but not the mesh geometry.

Move Selection

Left-click and drag to move a selection.

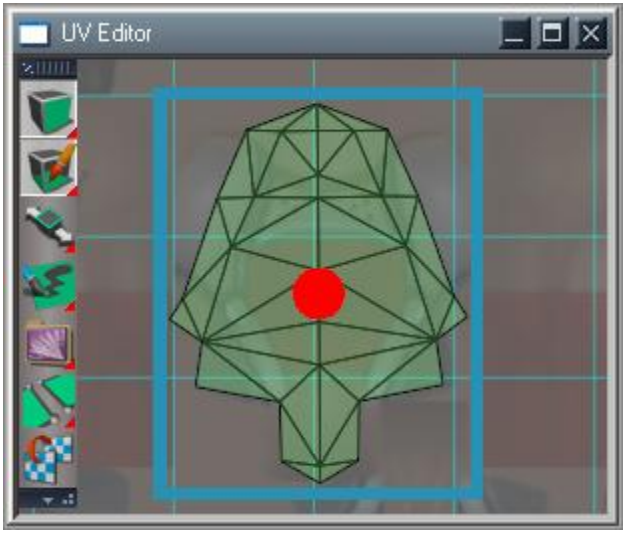
Point Rotate

Left-click and drag to rotate the selected mesh around the center of the selection.

Point Scale

Left-click and drag to scale width or height of the selected mesh.

5.2.2 UV Editor Direct Manipulation Control

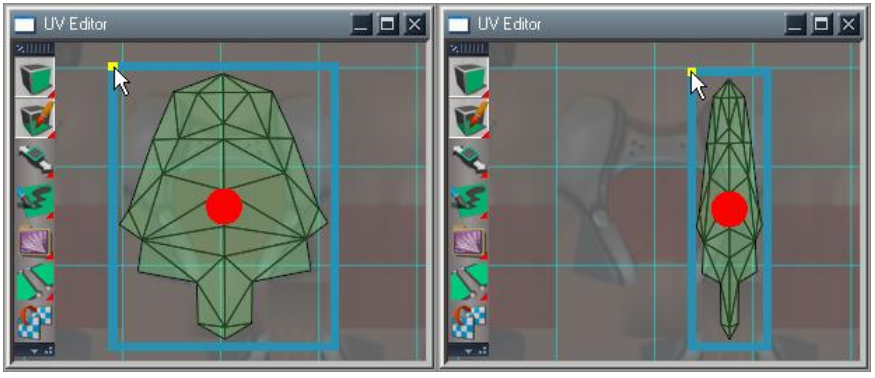


UV Control Cage

The **UV Controller** is a blue control-cage created around the selected mesh elements. It has clickable areas on each side of it which highlight yellow when the mouse is pressed over them for performing Scale, Rotate, Move actions according to which part or mouse button is pressed whilst dragging the area of the controller.

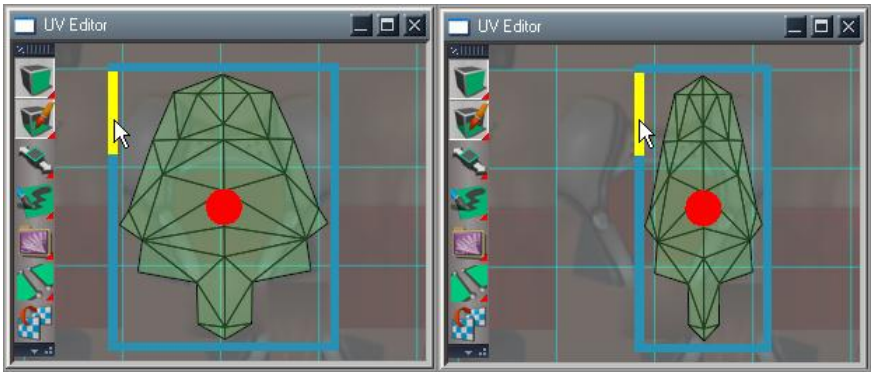
UV Scale

- **Scale:** Left-click and drag any of the corners squares to scale the selection by axis in a non-uniform manner.



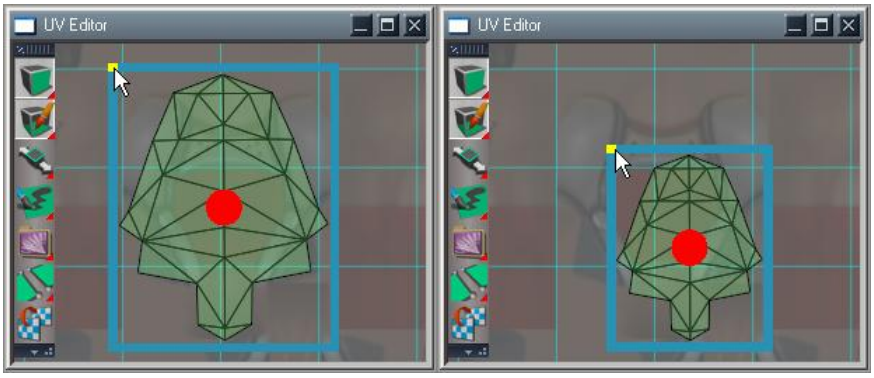
scale the selection by axis using the corners in a non-uniform manner

- **Scale:** Left-click and drag any of the corner bars to scale the selection by axis in a non-uniform manner.



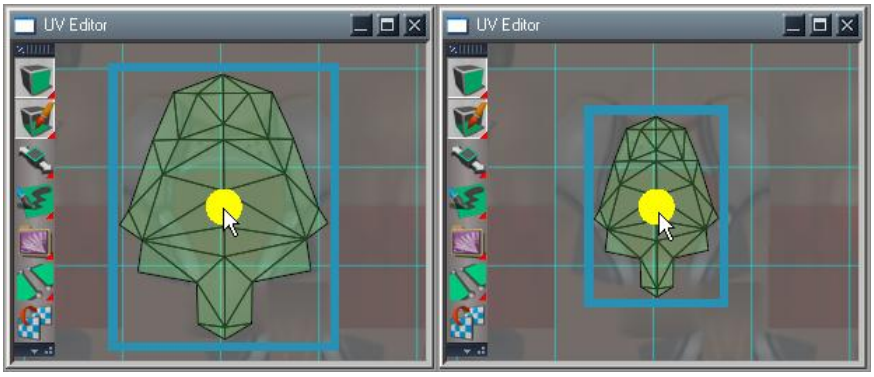
scale the selection by axis using the bars in a non-uniform manner

- **Uniform Scale:** Right-click any of the corners and drag to uniformly scale the selection by axis.



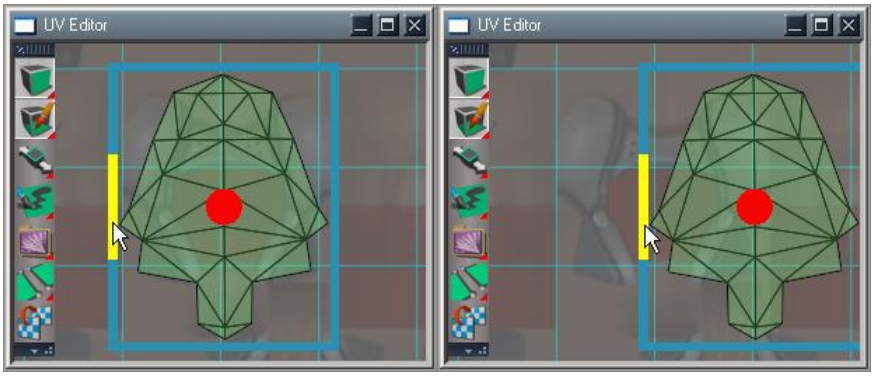
uniformly scale the selection by axis

- **Uniform Scale:** Right-click the red rotate tool and drag to uniformly scale the selection by center.



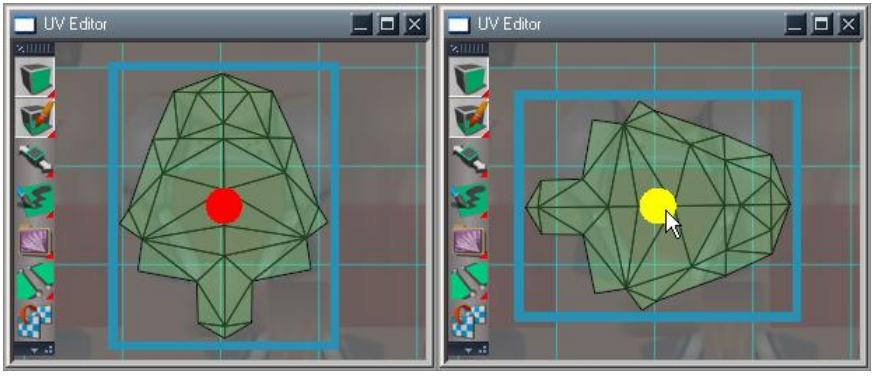
uniformly scale by center

- **Move:** Left-click over the middle of any of the sides and drag to move the selection horizontally or vertically.



move the selection by axis

- **Rotate:** Left-click the red dot in the middle and drag to rotate the selected mesh.

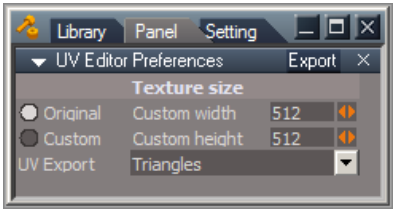


rotate the selection by center

5.2.3 Export Tools

Export Texture to file

Export Texture to file will save the current contents of the editor window (both texture and mesh) into a bitmap file. A **Save** dialog will appear where you can choose the file name. The size of the exported bitmap will match the texture selected in the editor; if there is no texture, it will match the size of the editor window. The settings can be adjusted prior to export by switching from the default tab to the Export tab of the UV Editors preferences.



Export options

Edit Texture in External Editor

Edit Texture in External Editor Texture can be edited in a program which is associated as UV editor. The UV Editor Preferences Panel has the setting to locate your external editor. If no external editor is indicated by you, the default editor for .png files will be used.
Press the tool button and the image is opened in the external editor. When changes are applied to texture and image is saved to disk, changes in texture will reflect in UV editor as well as on the object being mapped. Texture can be edited until the external program is closed, another object is selected, or UV editor is closed.

Texture painting

Paint Brush

This tool allows you to paint onto textures. First you must select an object that has a UV and texture applied. The Paint Brush tool is intended for "marking" your texture to highlight areas for work in the external editor. You could use the Paint Brush to paint the model as final finishing touch if so desired.
The UV Editor Preferences panel has one setting for the brush "color" as well as one setting for Brush size, which will allow you to customize.

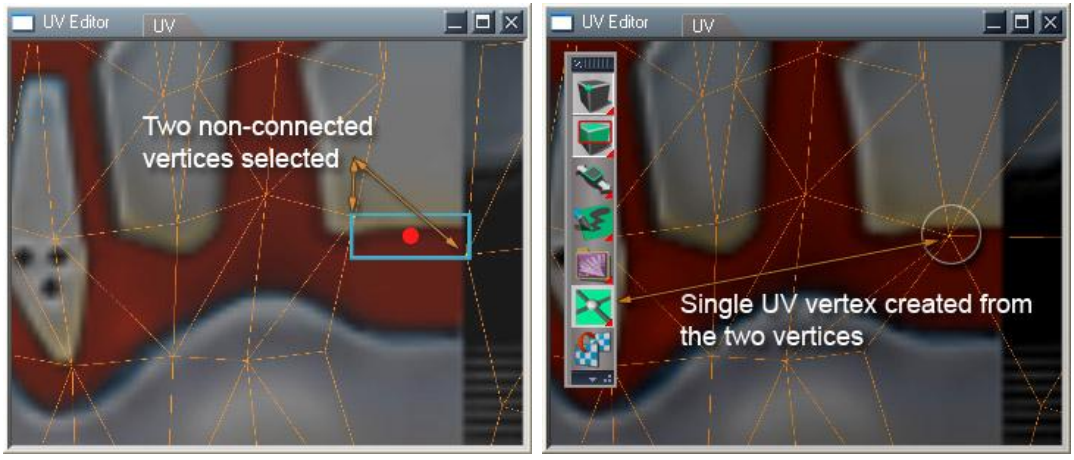
Draw Line

Just as with the Paint Brush tool, the Draw Line tool allows you to draw lines on your mesh. Intended as a method to "mark" your mesh for highlight in external editor. As with Paint Brush, you could use Draw Line tool as finishing touches if desired. The UV Editor's Color/Paint Color setting determines the color of the lines being drawn.

5.2.3 Mesh Tools

Weld UV Vertices tool

You can use this tool to combine selected UV vertices into one vertex, so they are no longer separated.
Further manipulations will affect all adjacent triangles. Note that welding is performed regardless
Of whether selected vertices lie in same location or not. All selected vertices are collapsed into one location, so that every former UV vertex now has same UV assigned. Following figure shows selected vertices before and after using weld tool. Note that also edges and triangles can be selected, selection will be internally converted to vertex once weld tool used.



Welding vertices

Slice selected triangles from rest of UVs

The **Slice** tool (short name) allows you to separate a section of the mesh and manipulate it independently from the rest. By activating this tool, currently selected faces are separated along the edges from the rest of the mesh. Separates selected UV triangles from the rest of UV mapping so that they no longer share vertices. This is useful for relocating certain part of UV coordinates into its own texture location. Following figure shows UV triangles separated using slice tool from the rest of UV mapping, which was otherwise compact.

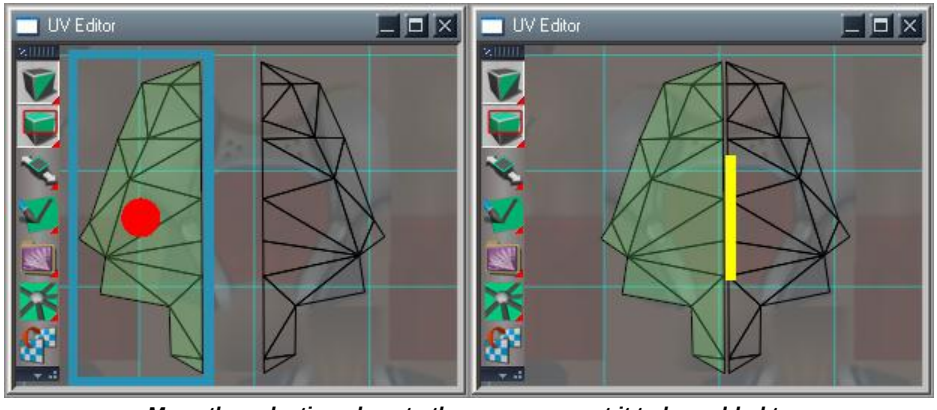
Use the Slice tool to help organize and work with your uv-mesh

Heal selected UVs

Works with triangle selections the tool puts separated triangles close to desired position and welds the UV selection into the main mesh. Right click over the tool shows a panel in the stack to adjust the weld tolerance for welding proximity, larger values will weld to a greater area small values are used for placing objects close together, the default value 0.020 is good for most uses.

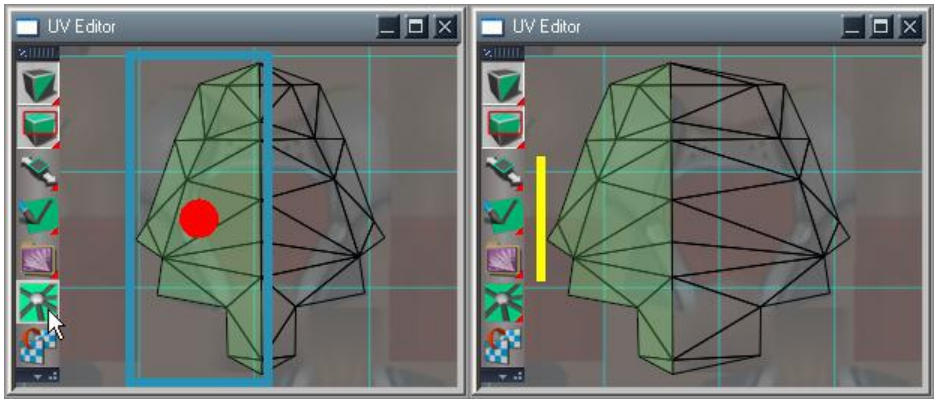
UV Heal Preferences panel

Make your selection and move it to position.



Move the selection close to the area you want it to be welded to

When the tool is used the Selection is welded to rest of mesh. Provides an alternative to using point-by-point welding and can be used instead of the manual point by point select and weld process.



Selection is welded to nearest parts, move just to illustrate that parts are welded together

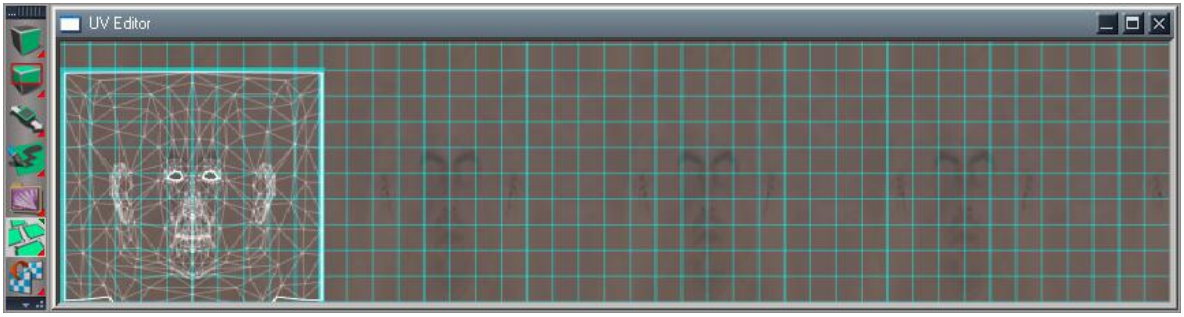
Flatten Selected UV Coordinates

Using this tool decomposes the object or selection to semi-planar segments and projects them into UV space.
For a cube Patches are scaled down to approximately cover the 1x1 square and placed next to each other.
All patches are oriented in such way that they take up minimum space in UV so they may be rotated.

UV Flatten Preferences

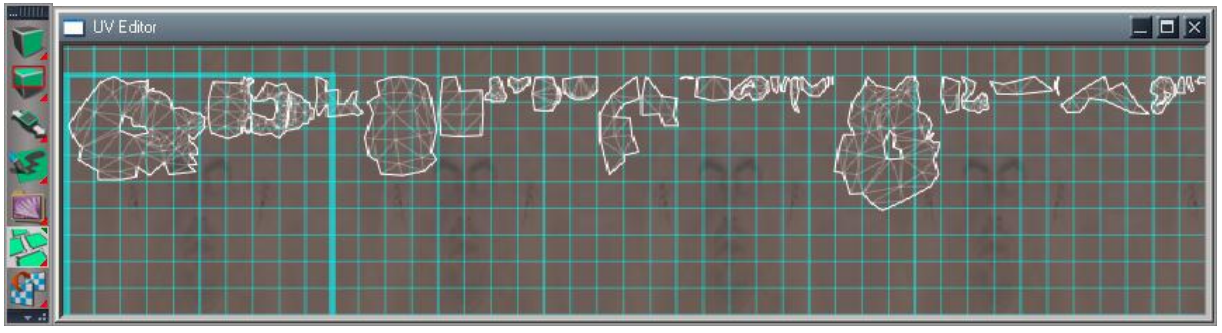
Maximum Angle (deg) : Over which normals of an unwrapped segment can span. For example, 90' will flatten whole half of sphere into one segment if possible. Maximum angle determines how strictly planar the portion of mesh must be when selected as a UV patch. Lower values means more segments with smaller span of normals. Higher values mean less bigger segments projected to plane. Too big angle can introduce distortion. The highest acceptable angle is 90 degrees.

Minimum Segment size : Value taken into consideration when Flattening the UV, higher value = less patches. The Minimum segment size limits the minimal number of faces in a patch to prevent many small patches. When a particular UV patch is too small, additional faces are attached to it, possibly violating the tolerance angle. Usually the distortion created by too big minimal segment value is more acceptable than numerous patches.
Note: on some meshes, it is not possible to keep the number of faces in patch minimal (such as a simple cube).



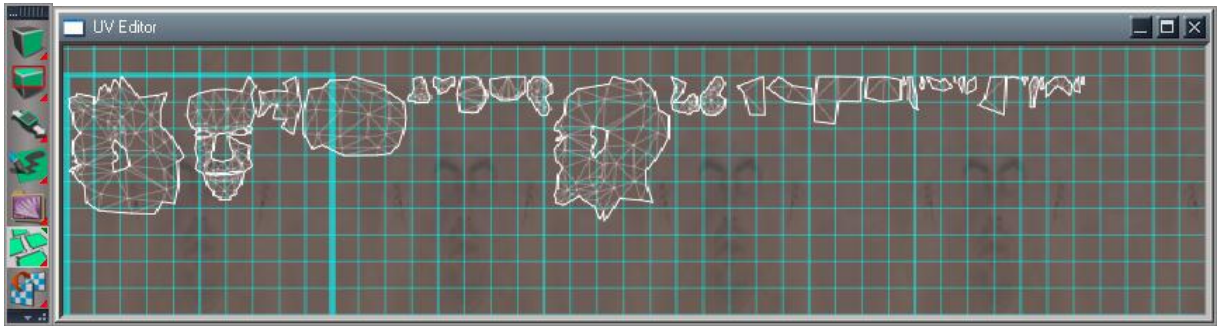
Mesh before flatten is used

Using the default values spreads the selected meshes UV according to the tolerance and segment size.



UV Flatten with default Preferences values

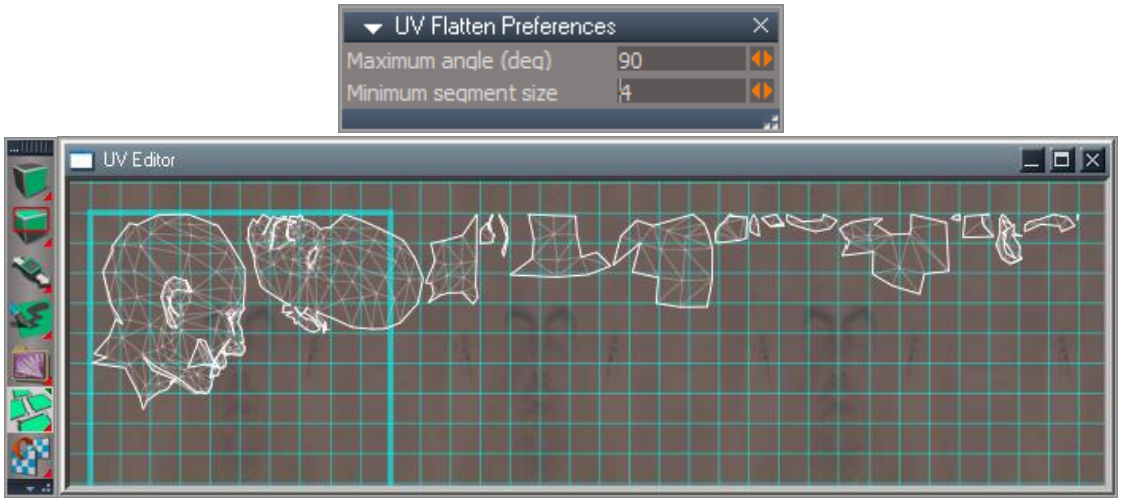
Changing the minimum segment size yields a more compact result.



UV Flatten with Minimum segment size 16

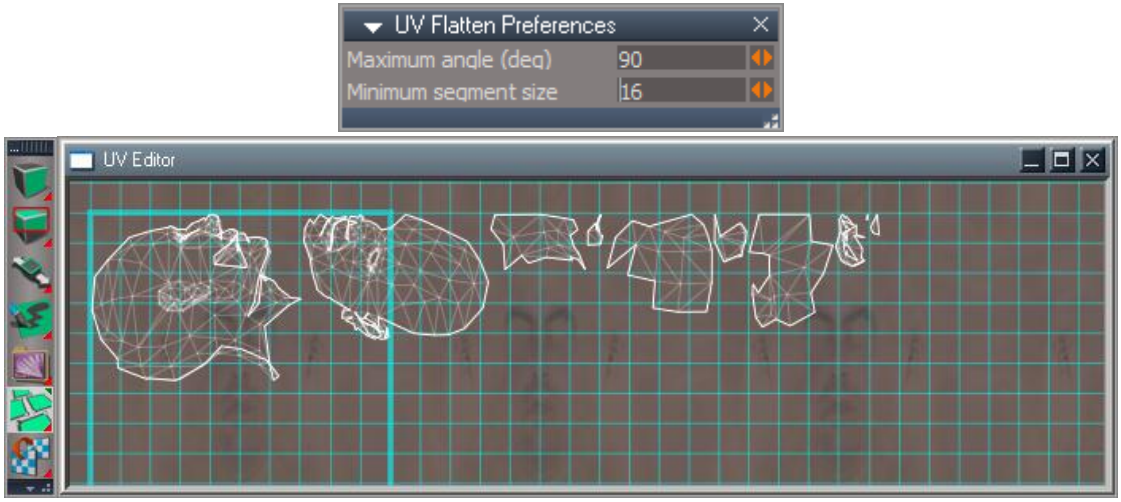
Changing the Maximum angle to 90 but leaving segment size at 4 yields a more compact result but this can be

dependent on the type of mesh being flattened.



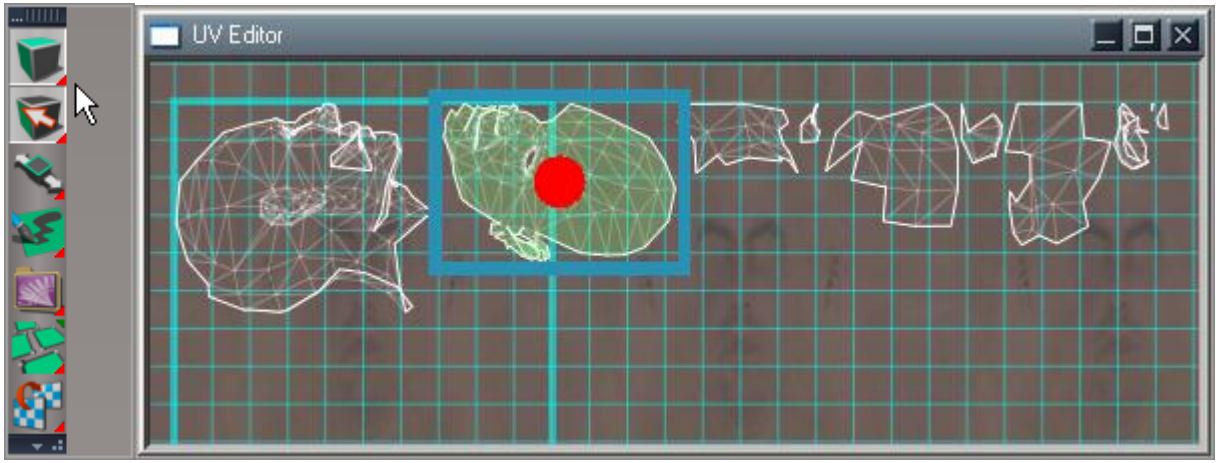
Flatten tool results for Maximum angle at 90 and segment size at 4

Using Maximum angle of 90 and segment size at 16 yields an even more compact result but again this will be dependent on the type of mesh being flattened.



Flatten tool results for Maximum angle at 90 and segment size at 16

After the Flatten tool has been used on a mesh to create the separate patches the individual patches can be manipulated using the patch selection tool  combined with the pick and move selection  .



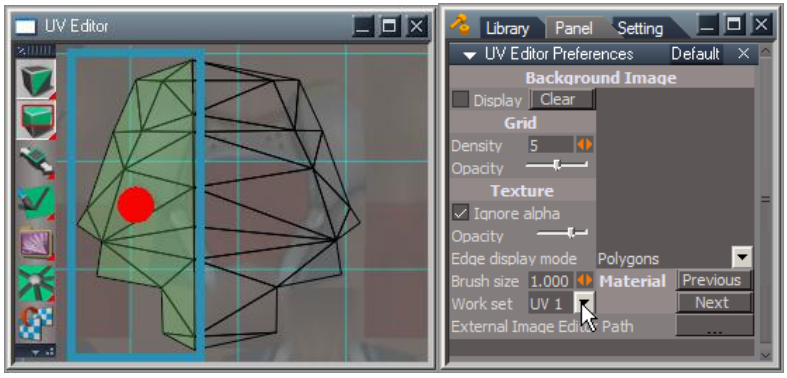
patch selection tool combined with the pick and move selection allows manipulating individual patches

Synchronize UV Coordinates

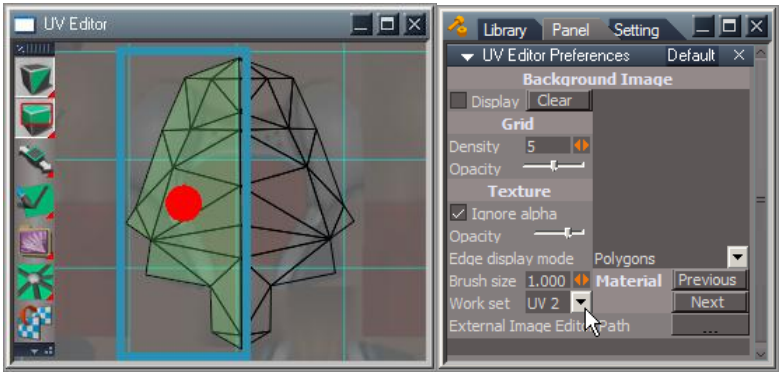
Synchronizes current UV set with the other one.

The two sets of UV coordinates can be used independently from each other but sometimes it is useful to be able to match them so that the sets will match for both the texture and the normal map.

You can see in the images below each UV Work set has different coordinate settings.

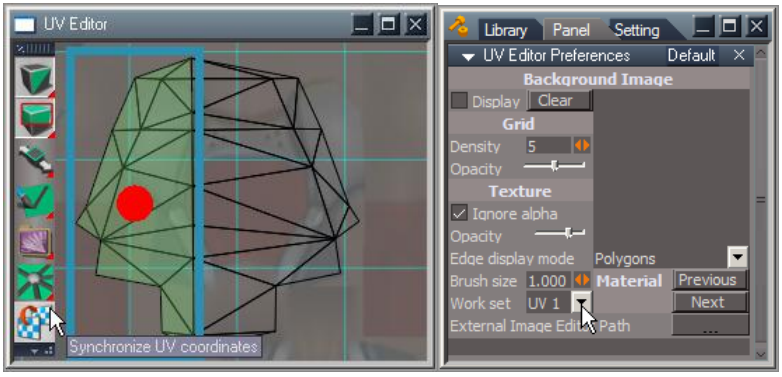


UV Work set 1 coordinates

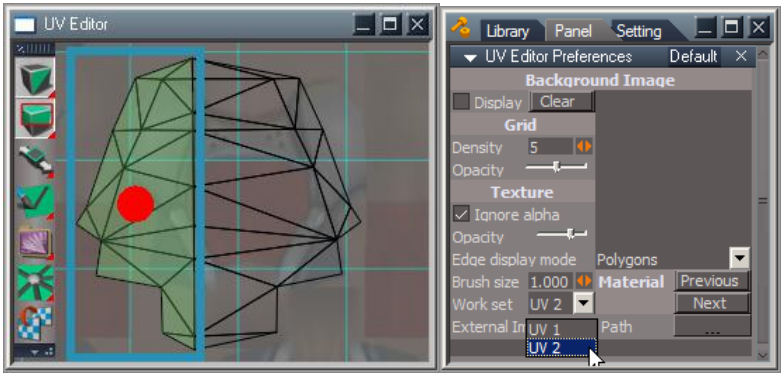


UV Work set 2 coordinates

If you switch back to UV Work set 1 and use the synchronize tool then when you look at UV Work set 2 it will match the UV set 1 coordinates, similarly if you have the UV Work set 2 active and adjust the UV coordinates then use the synchronize tool , then the UV set 1 will match the UV set 2.



Use synchronize on UV Work set 1

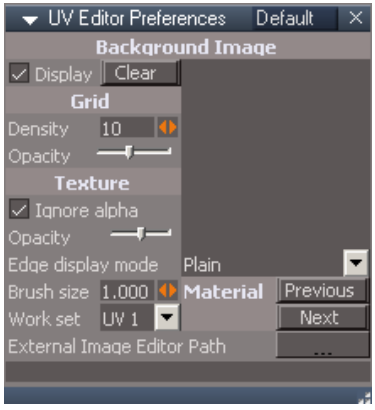


Switching to UV set 2 shows the mesh UV is updated to match UV set 1

5.2.4 Working with multiple materials

If an object contains multiple materials, UV editor displays only triangles to which the appropriate material is mapped. Current material can be switched using “Next material” and “Previous material” in tool view panel. Paint tools work on the diffuse texture assigned for current material. Also, this texture is exported or edited. Use the material editor for changes in material assignments.

5.2.5 UV Mapping Editor Options



Default Aspect

Color Aspect

Export Aspect

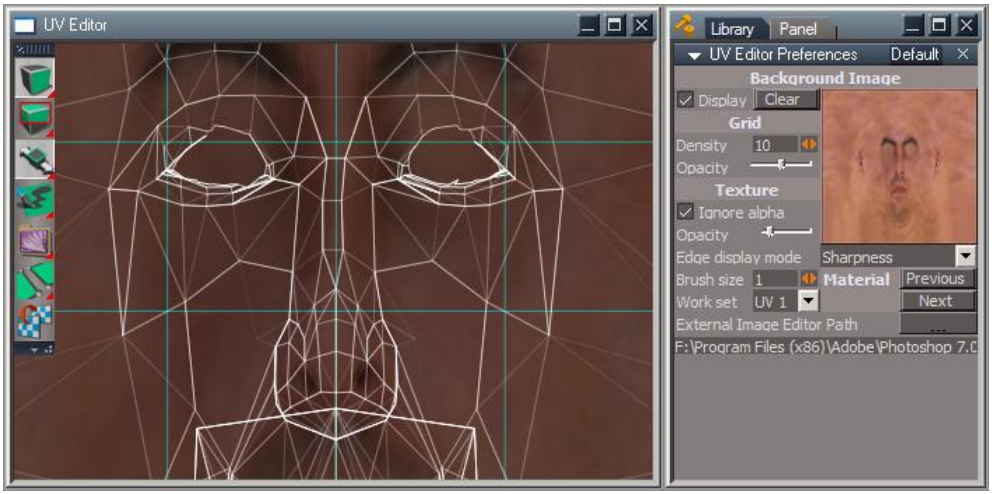
The Preferences panel can be accessed in panel view tab of Stack while UV editor is active by right-clicking on the **UV Mapping Editor** icon.

Background Image
Allows custom background texture being displayed in UV editor as a replacement for blank background. This allows better texture alpha recognition and guiding texture for UV coordinates can be

painted to be used as background.

Clear background

Use this button to wipe the background texture and return to the original background.



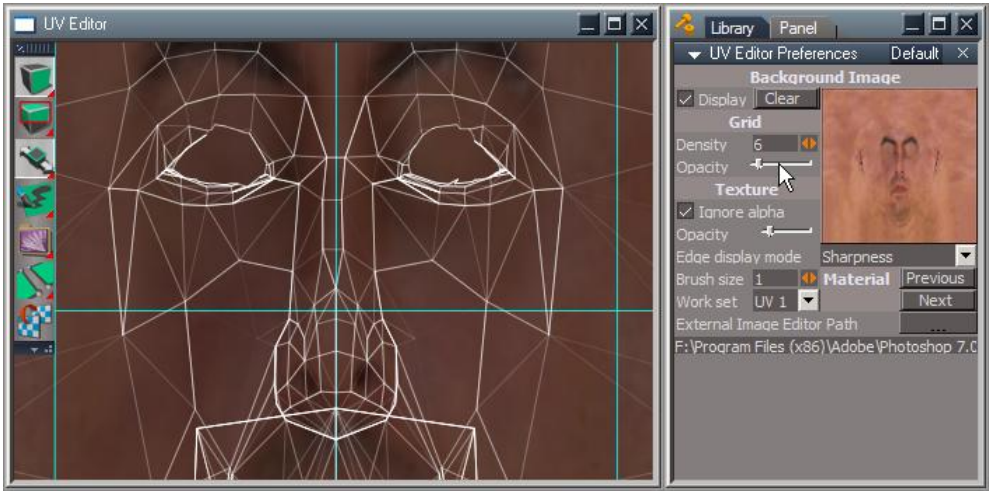
Select a background image from your hard drive to show in the UV Editor

Grid Transparency

Transparency of auxiliary grid displayed in UV editor can be adjusted here.

Grid density

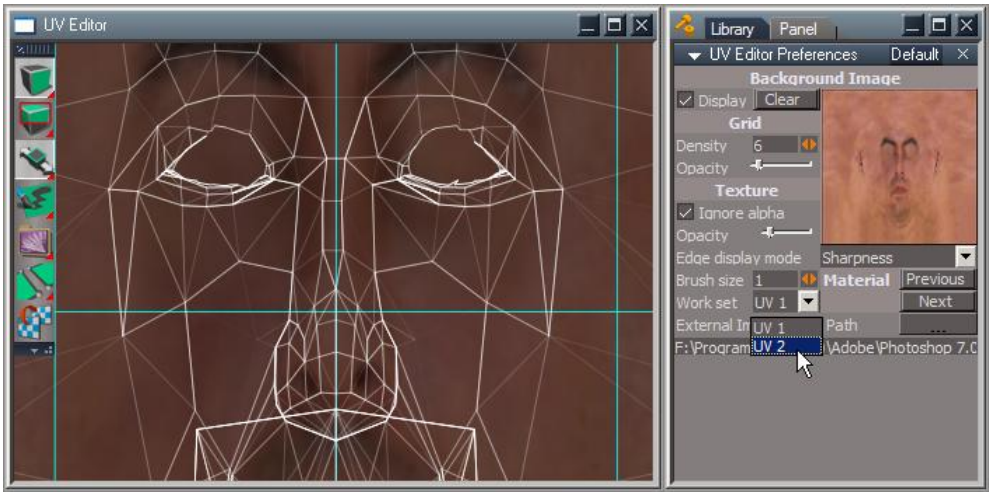
Required density of auxiliary grid can be adjusted. It displays specified number of units per texture dimension in UV editor window.



Grid transparency and density help you navigate your UV Editor

Work set

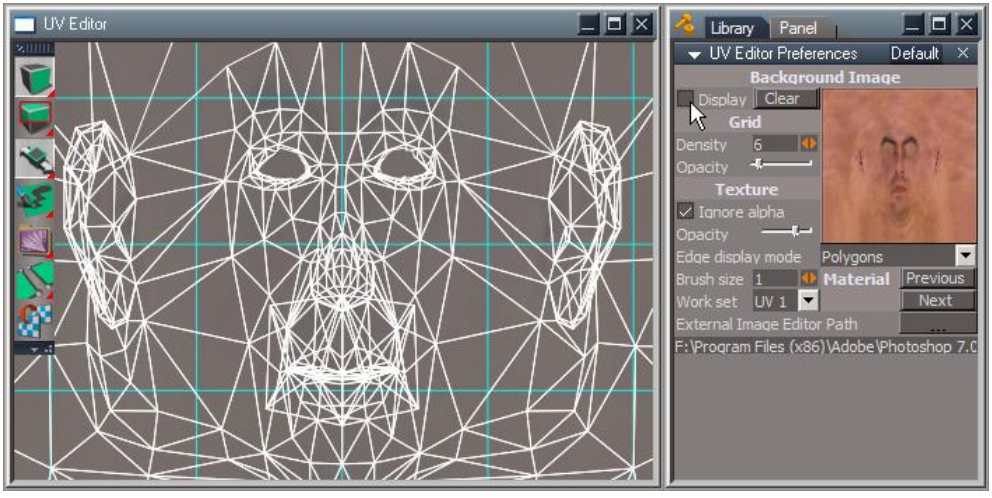
Choose between UV1 and UV2 Work set mappings to be displayed. Note that objects do not necessarily need to have both mappings.



Some objects may contain more than one UV Work set

Display background

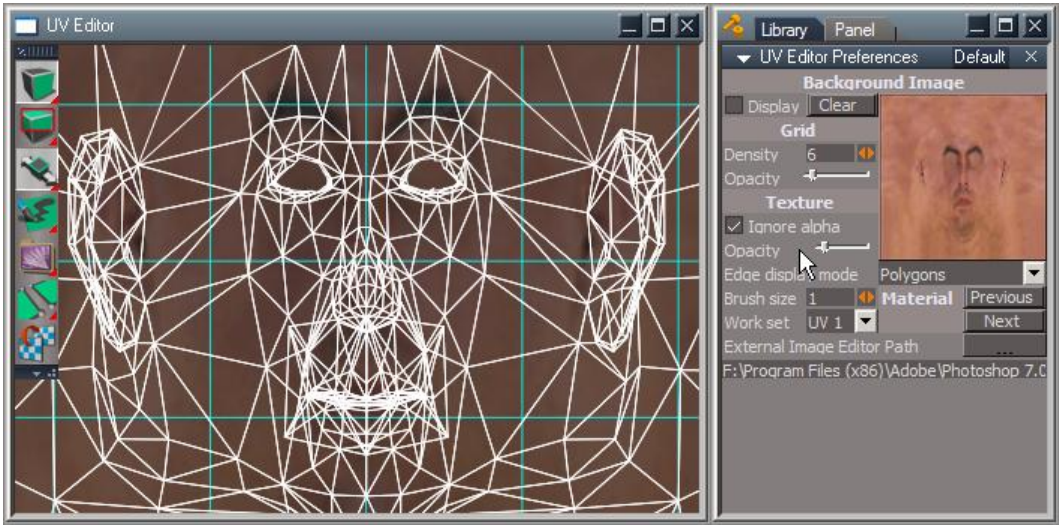
Toggles the display of background texture. Can be used for temporary hiding of background, to avoid reloading it each time it needs to be displayed.



Decide whether the background image should display or not

Texture transparency

Adjust the transparency of diffuse texture displayed in UV editor.



Control the transparency of the UV Texture by using opacity

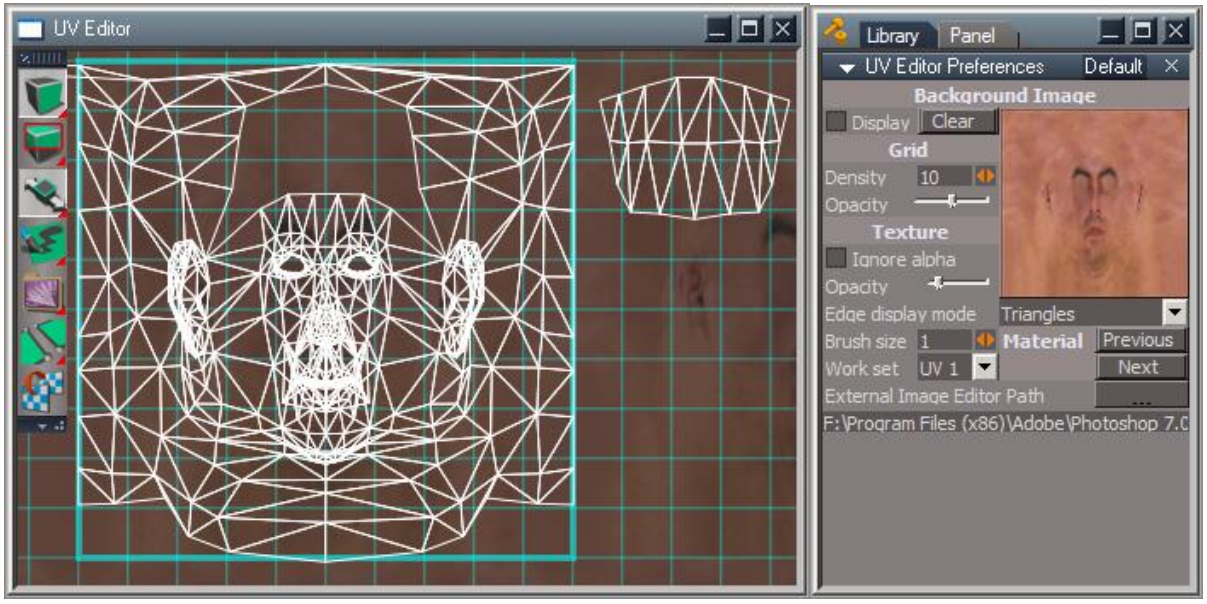
Edge display mode

Adjust the visualization mode for UV mapping displayed in UV editor.

Available modes are:

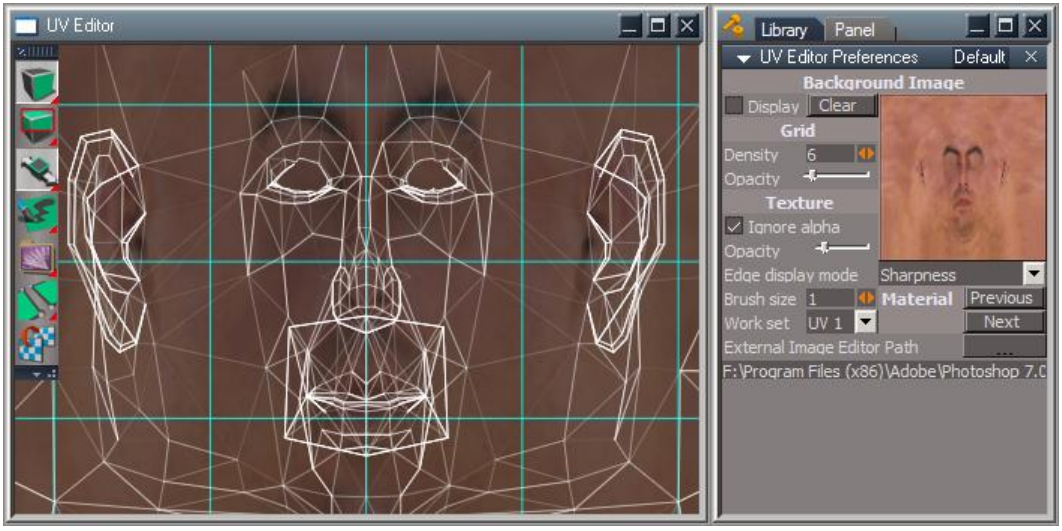
- **Triangles**: Display all triangles and their edges.
- **Sharpness**: Display edges of triangles depending on angle between them. Planar coincident triangles have edges completely transparent, sharp edges are displayed strong. This may be useful when polygons are selected.
- **Polygons**: Display only edges of polygons. Edges between triangles which lie in single plane are not displayed .
- **Patches**: Displays the separated UV parts as patches which can be selected using the patch tool  and  the pick and move selection tool.

Triangles display mode: Every edge of UV mapping is displayed in full strength.



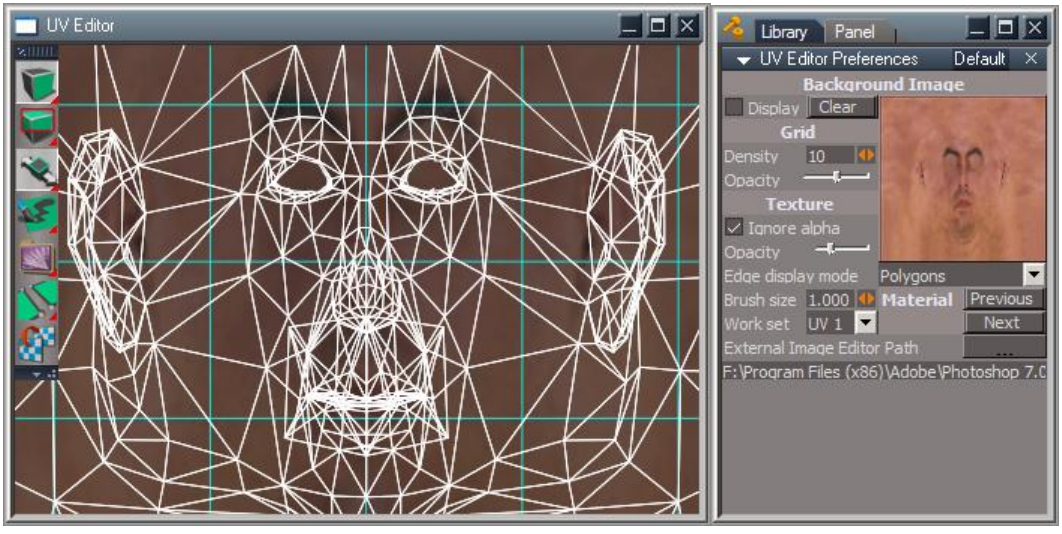
Workflow with Triangles Edge display used

Sharpness display mode: Edges are displayed with strength which corresponds to edge sharpness in geometry.



Workflow with Sharpness Edge display used

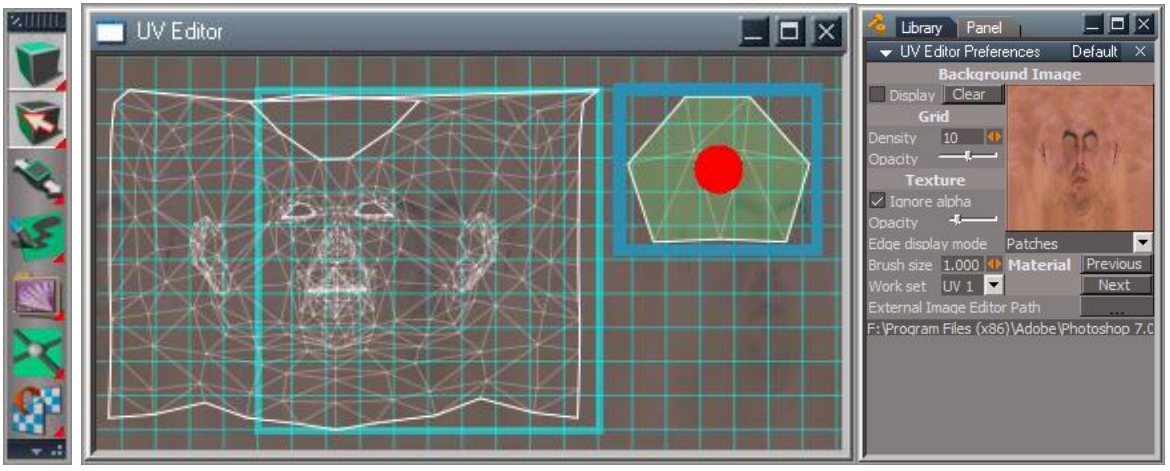
Polygon display mode: Coplanar and almost coplanar triangles are displayed without separating edge.



Workflow with Polygon Edge display used

Patches:

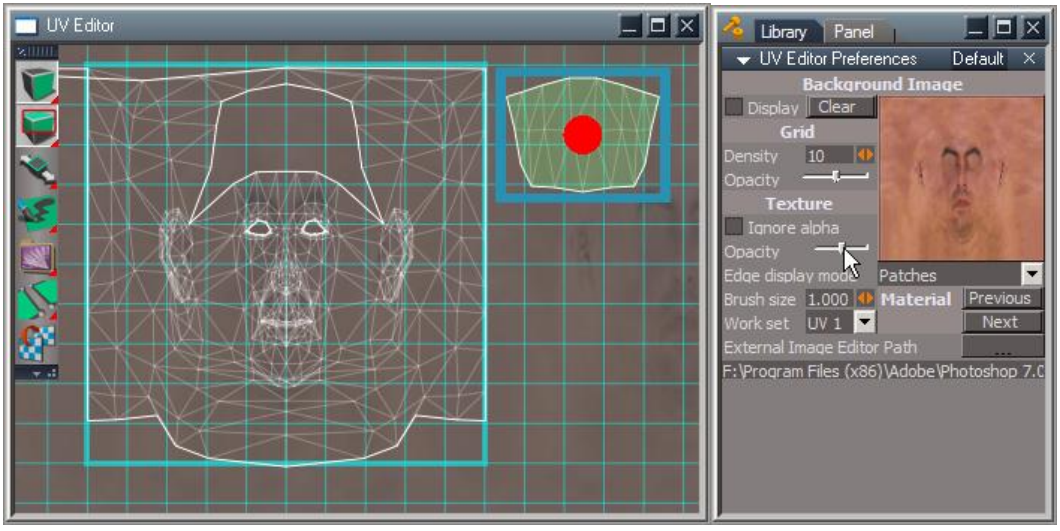
Patch was made by selecting some polygons and applying planar UV then the selection was scaled and moved .



Workflow with Patches display used

Ignore texture alpha

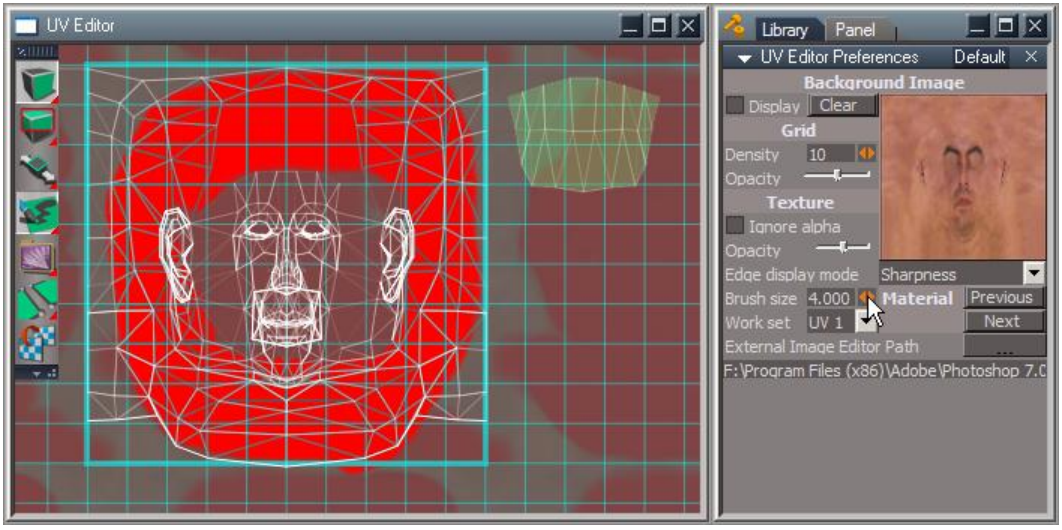
Toggles ignoring of alpha channel of texture and uses Texture transparency instead.



Control UV Texture's Alpha transparency channel if available

Brush size

Adjust the radius of paint brush.



Sizing the Brush in UV Editor

Previous/next material

Use buttons to toggle currently displayed material in UV editor. Diffuse map and assigned triangles are shown with current material.

Image Editor Path

Specify full path and filename of external texture tool which will be then used to edit textures when external editor is used. (e.g. C:\Program Files\Adobe\Photoshop\Photoshop.exe). If nothing is specified here, Windows default program associated as PNG editor is launched.

UV Editor Preferences: Color Aspect:

UV-Color Preferences

Grid color

Adjust color of UV background grid.

Wire color

Adjust color of UV wiring, if not visible properly in-front of the UV-texture.

Selection color
Adjust color of selection displayed in UV editor.

Highlight color
Adjust color of highlighted elements while using paint select tool.

Paint color
Adjust the color used for brush and line paint tools.

UV Editor Preferences: Export Aspect

UV-Export settings

Texture size:

- **Original**: use original texture size when exporting to external editor or saving to file.
- **Custom**: use custom when you wish to save or edit a size other than original.
    - Custom Width: enter value you wish to use here.
    - Custom Height: enter value you wish to use here.
- **UV Export**: Available modes are:
    - **None:** export no UV mapping information.
    - **Plain**: Display all triangles and their edges.
    - **Sharpness**: Display edges of triangles depending on angle between them. Planar coincident triangles have edges completely transparent, sharp edges are displayed strong. This may be useful when polygons are selected.
    - **Faces/Polygons**: Display only edges of polygons. Edges between triangles which lie in single plane are not displayed.

5.3 Workspace Material Editor

There are several different types of material in use within trueSpace. The material Editor has been designed to be flexible enough to deal with the various types of materials it needs to consider.
These are: Lightworks , V-Ray and DX-Materials , this section concentrates mainly on using DX-Materials on both the basic level and also on a more advanced mode as well.

Lightworks Materials

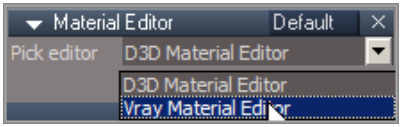
These are the traditional materials you may have already worked with in the Model side, with the four different shaders (Color, Reflection, Transparency, Displacement). These materials are only shown fully when rendered using the Lightworks rendering engine
It should be stressed that the main purpose of the Workspace Material Editor is to edit V-Ray materials and DX Materials, and its recommend to use the Model Material Editor for Lightworks Materials. However Materials you create in the Workspace Material Editor will be translated to their closest Lightworks equivalents when you move back over to Model to render.

- Further reading Ref: Ch 6 Surfacing Model View 6.3 LW Material Editor.

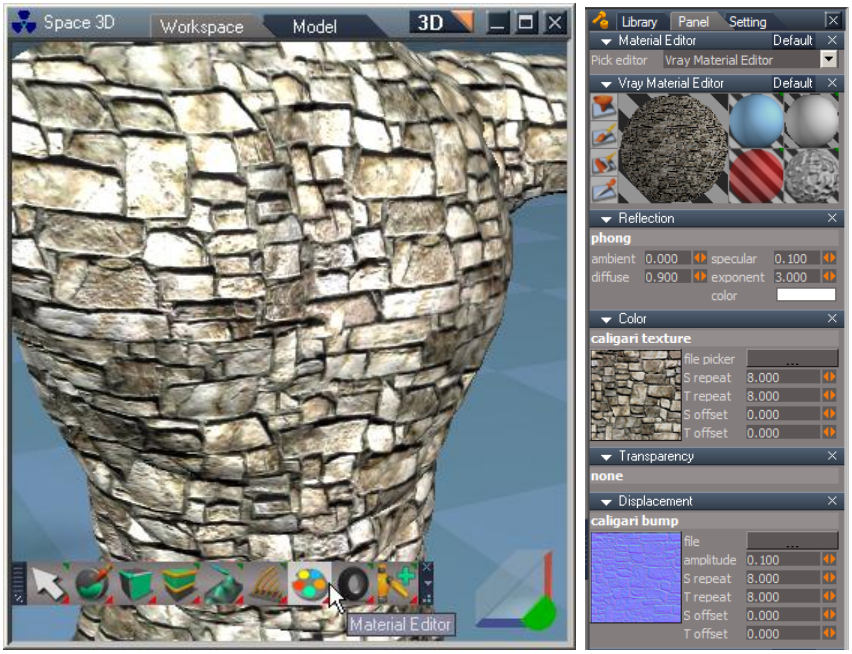
V-Ray Materials

These materials and the editor will only show when the optional V-Ray renderer is installed and loaded – and as the name suggests – are rendered with the V-Ray offline renderer. They offer a similar functionality as the Lightworks materials, but the information is translated to V-Ray data and then stored separately.
While you can work with V-Ray materials in the Model Material Editor the Workspace Material Editor is the recommended option, particularly if you are working with the animation tools. Now all animation, surfacing, and rendering can be performed on the Workspace side, without having to go through the Bridge.
The use and Methods of the Material editor in relation to V-Ray and the types of available V-Ray Materials are covered in the manual which accompanies the optional purchase of the V-Ray Renderer. http://www.caligari.com/products/trueSpace/ts75/Brochure/VRay1p0.asp?Cate=BRendering_vray

If you have V-Ray installed and loaded then the option to use it will show in the settings for the ME.



Material Editor picking options



Material Editor open in V-Ray editing mode.

DX Materials

DX (or DirectX) Materials control how a surface looks within the real-time Workspace view, using custom real-time shaders that can be built in the Link Editor. There does not have to be any real similarity between the DX Material seen in real-time and the results seen in the offline renderers (Lightworks, VirtuaLight, V-Ray, etc.) though you can attempt to create DX Materials that look close to the offline result. Note that you don't *have* to specify a DX Material explicitly – whenever you apply a Lightworks or V-Ray material to a surface, the real-time renderer will attempt to translate basic parameters such as color, texture, shininess etc. and display them in real-time.

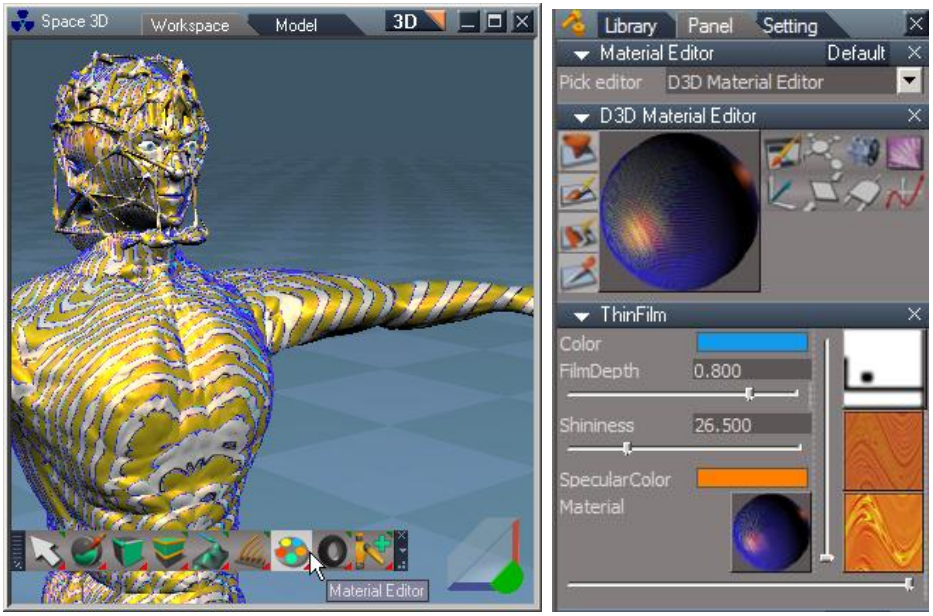
Likewise, if you apply a DX material to a surface and then render with e.g. Lightworks, the basic parameters will transfer over from the real-time material to the output render. Exactly what parameters will be retained varies from shader to shader – for example, if you apply the Gooch DX Material to an object and then render in Lightworks or V-Ray, the object will be a plain color, taken from whatever is specified as the "DiffuseColor" in the Gooch material, the other parameters are not considered by the offline renderer as it has no way of knowing how to deal with them because of the differences in the way the different render engines interpret the available data being fed to them.

The Workspace Material Editor Inspect tool can be used to access the top level basic properties of an object's DX Material or Shader where the exposed attributes such as colors or textures, normal maps, texture repeats, shininess and other properties placed there at the top level by the material or shader designer.

D3D Material Editor

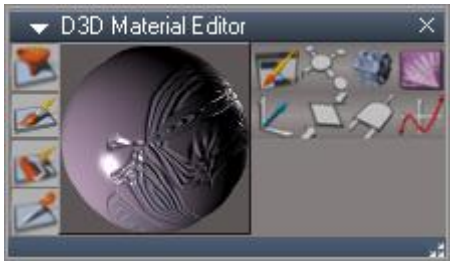
Using the Material Editor provides an easy mechanism for editing top level basic structures of Materials, without needing to use the more advanced editing features that are also accessible from within the Material Editor panel.
The Workspace Material editor can be started by Left mouse-click on any of the material editor tools in the Workspace toolbar, using the tools from here will allow for operations to be performed on selections and applied relating to the currently loaded material in the editor . These same tools are also available in the Material Editor itself as a series of buttons displayed in a column on its left hand side.
There are also several rows of buttons on the right hand side of the Material editor which give easy access to more advanced DX - material editing libraries of objects used to build more complex materials in the Link editor.

Location of Material Editor tools in Workspace toolbar.



Clicking on the Material Editor Icon in Workspace toolbar opens the Material Editor Panel in this case a user created variation of ThinFilm DX-Material was applied to the object and the Material Editor opens in DX editing mode.

Basic Painting Tools:



Material Editor

- **Paint Object:**
  - Assigns material active in Material Editor to all selected objects.

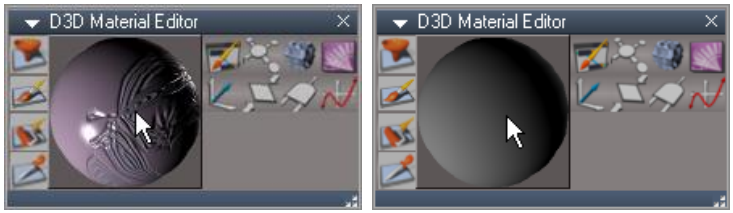
- **Paint Face:**
  - Starts a tool to assign material active in Material Editor to individual faces. Face can be selected either by left-clicking on it or by left-click & drag – in that case all faces which mouse moves over them will be painted by the active Material Editor material. If there is a face selection associated with the object at the time of a mouse click all selected faces will be painted.
  - The tool is active while paint face cursor is active – you can disable it by right-clicking anywhere in the 3D view.
  - Ctrl+click paints only selected objects.
  - Shift+click paints first object.
  - when no shift or ctrl is pressed then the tool first checks for the selected objects and then the unselected, e.g. if you pick the currently selected object it is painted even if it is occluded by unselected objects.

- **Paint Over: Repaint**
  - Starts a tool to repaint a material on object with the material active in Material Editor. The material to replace is selected by left-clicking on any face of the object being painted by a particular material.
  - The tool is active while paint over cursor is active – you can disable it by right-clicking anywhere in the 3D view.
  - Ctrl+click paints only selected objects.
  - Shift+click paints first object.

- when no shift or ctrl is pressed then the tool first checks for the selected objects and then the unselected, e.g. if you pick the currently selected object it is painted even if it is occluded by unselected objects.

- **Inspect:**
  - Starts a tool to acquire material from object and make it the active material in Material Editor. Left-click on any face using a particular material you want to make active in Material Editor.
  - The tool is active while inspect cursor is active – you can disable it by right-clicking anywhere in the 3D view.

- Reset: you can return to the default material easily by a Right-click over the preview area



Right-click over the preview will "Reset" the material editor to the default simple material

Advanced Editing Tools:

The Material Editor provides some buttons to give easy access to some of the most commonly used libraries and components available to design more complex real-time DX Materials and Shaders in the Link Editor.
There are also some additional shader libraries available for use in creating DX materials and these can be opened using the Library browser .

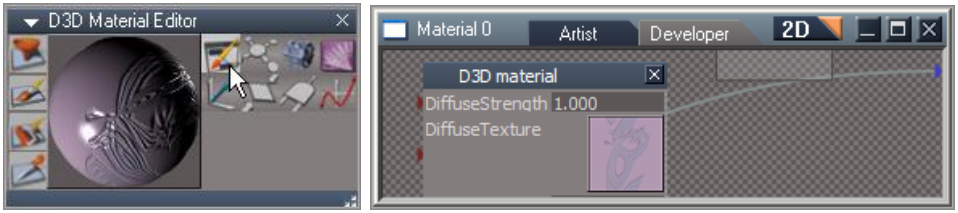
Note:- Both sets of libraries can be opened from the library browser.

- Further Ref: Ch 5 Surfacing Workspace View : 5.4 DX Material Editing

- Edit DX Material in the Link Editor.

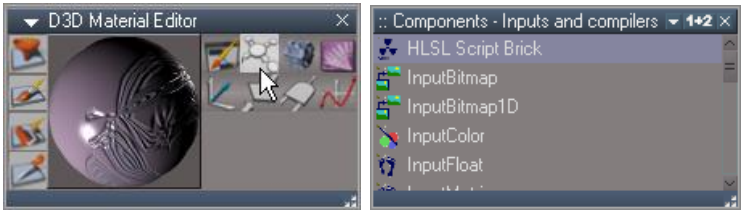
This will open and focus a Link editor on the material for advanced methods of editing.

- Further Ref: Edit DX Material



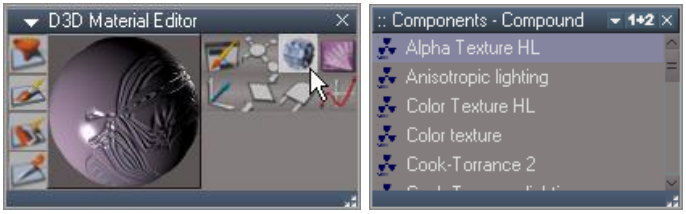
Edit DX Materials in the Link Editor

- DX Components Inputs and Compilers.



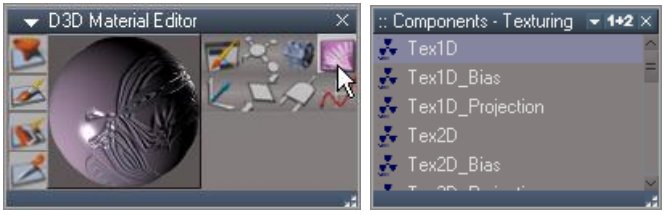
Opens DX Components Inputs and Compilers bricks library

- DX Components Compound.



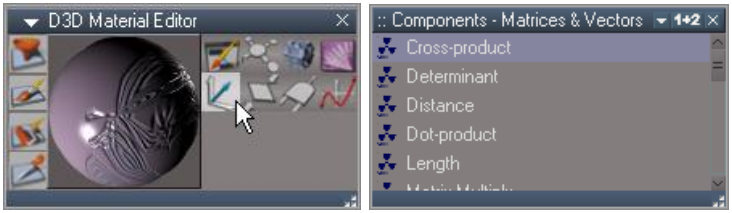
Opens DX Components Compound bricks library

- DX Components Texturing.



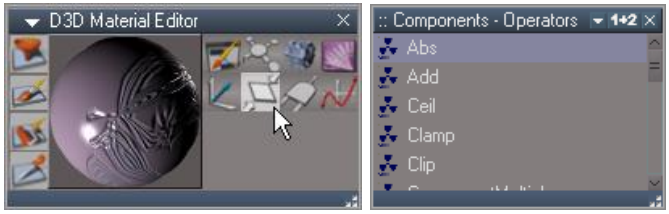
Opens DX Components Texturing bricks library

- DX Components Vectors.



Opens DX Components Vectors bricks library

- DX Components Operators.



Opens DX Components Operators bricks library

- DX Components Logic.

Opens DX Components Logic bricks library

- DX Components Functions.

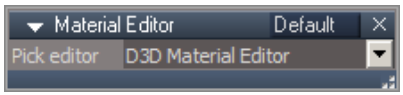
Opens DX Components Functions bricks library

- Further Ref: Edit DX Material

Material Editor Settings:

- Default Aspect:

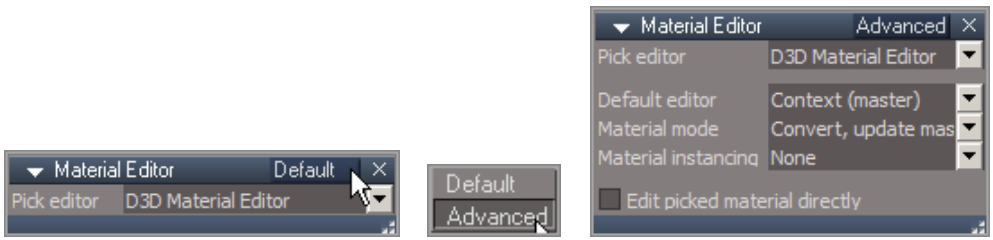
Shows available pick methods , the choices will depend on if there are other offline renderers installed , e.g. V-Ray



Default Aspect

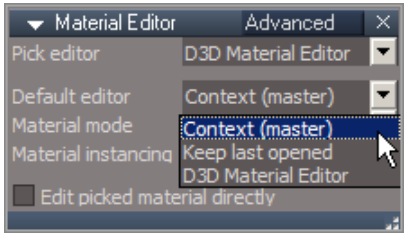
- Advanced Aspect:

You can switch between a basic mode and an advanced mode in the panel, for most purposes the basic mode will be okay and the settings in the advanced tab do not need changing.



Advanced Aspect

- Default editor:



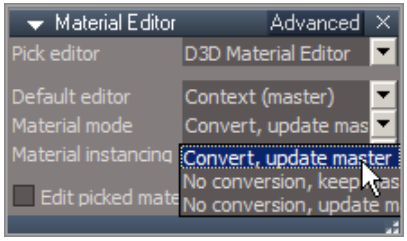
Default editor

- **Context** (**master**) - one material type is marked to be **master**. By default, the last modified material is considered as the **master** material - so if you edit DX material on one object and V-Ray material on another, picking DX material will open DX ME, picking the V-Ray material will open V-Ray ME etc.

- **Keep last opened** - when you pick a new material, material editor is not swapped to editor used to create picked material. For example if you have DX material painted on a cube and V-Ray material editor opened, after picking the cube the DX material it is converted to V-Ray material and you see V-Ray preview. unless you make change to V-Ray material, DX material will be kept untouched. if you modify V-Ray material, then depending on material mode DX will or will not be converted from V-Ray materials.

- **D3D material editor** uses DX material nodes / hlsl bricks+shaders not lw shaders. But you can apply Lightworks/V-Ray shader on D3D material, it will get converted to DX Material but with a limited conversion.

  - Material mode:



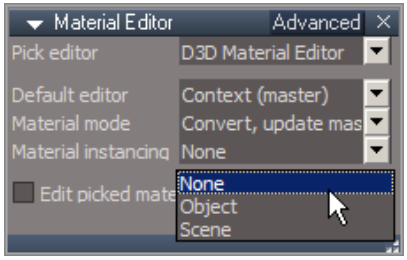
Material mode

- **Convert update master** - default, changing material causes it to become master and other materials are then converted from it when needed.

- **No conversion, keep master** keeps the master material intact. Assume you have set LW materials and want to use LW materials for modeler or 3rd material editor and you want to update DX material. This makes it possible. You make change to material without it becoming template for future changes.

- **No conversion, update master** - assumes that modification makes modified material the best representation of what you want, no conversion keep **master** does not change this information.

  - Material instancing:



Material instancing

- **None** - material change applied to one mesh or surface only. This is a useful mode to use when preparing objects and scene surfaces before using an instance mode to repaint the objects later it helps to separate out the parts into individual color coded areas which you want to share common materials with later and makes it

easy to identify then replace the materials when switching to an instance mode later on in your projects.

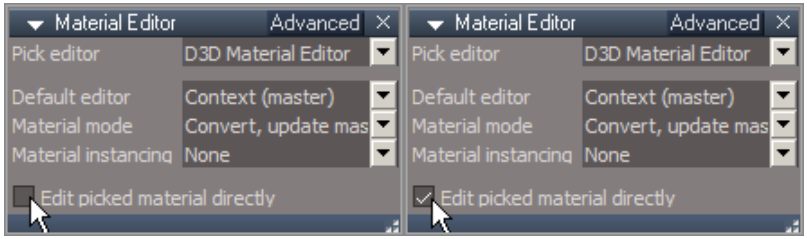
- **Object** - material changes applied to all meshes within the object. Painting In object mode creates a material object in the object root and will share the material for all parts within the object that use it, direct editing of the material will only apply changes the object reference materials and not the scene reference materials.

- **Scene** - material changes applied to all meshes within the scene. Painting in scene mode creates a material object in the scene root. Materials then reference this shared material.

Notes:

- Without instancing each sub-object or object in a scene will have its own individual material object even if they use the same material this can produce an impact on scene size if large textures were used multiple times within an object.
- Repainting in an instance mode will merge equal materials together.
- When repainting: ctrl is used when you want to keep target material instancing mode, for example if you have scene material instancing but your object uses object instancing and you want to do object material instance.
- When repainting: Shift key is used together with shared materials; for example if shift is pressed then repaint tool repaints the shared material (so all objects using it are repainted), otherwise only the objects individual material is updated.

  - Edit picked material directly:



Edit picked material directly

- Unchecked : the material being edited will need to be repainted on an object:
- Checked: the material being edited will show the edits directly on the surface of the object, no need to repaint.

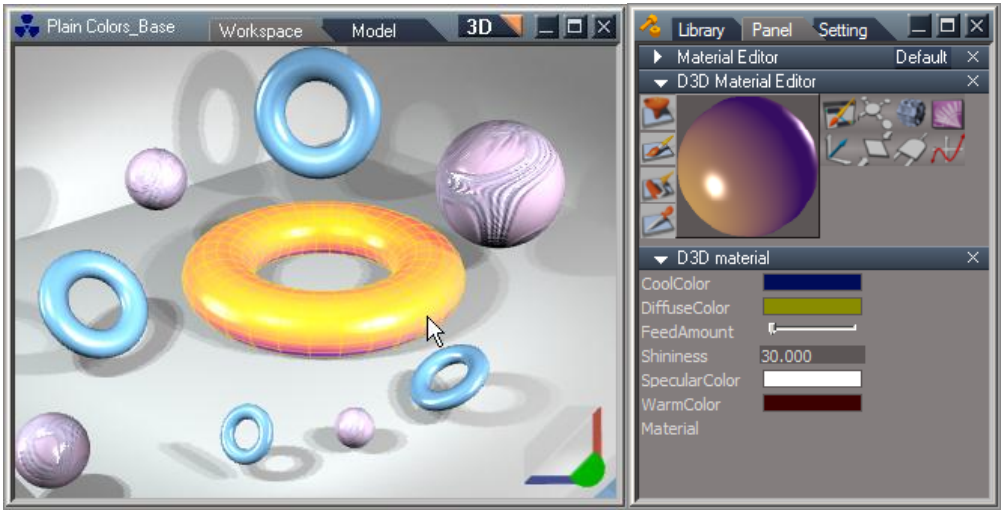
5.3.1 Basics Loading a Material or Shader from a library

Loading and applying materials from the libraries onto objects adopts a similar approach as other library objects and can be done in several ways.
You will probably find your own preferences by trying out each of these ways to load basic materials.

- Method 1: Double click a material or shader item in a library.

You can double - click the left -mouse button on a material or shader and it will be applied to the selected object or to a group of objects if the whole group is selected .
Using this method will also activate the Material Editor and load the active material for editing and repainting .

Double-left mouse click on a material in a library loads a material onto the selected object,



this method also switches the stack and loads the active material in the Material Editor.

- Method 2: R-click Menu in a library to Load an item

R-click over a material or shader and from the menu choose Load item, this will apply the material to the selected active object and also switch the stack to the Material editor with the loaded material , but the material won't be applied to the objects surface until one of the painting tools are used to paint the object with.

Using r-click menu to load from a library activates the Material Editor , material is not applied till painted onto the object.

- Method 3: Drag and Drop from a library

Use left-mouse pressed and Drag and Drop a material or shader from a library onto an object in the Workspace , the object under the cursor will have the material applied when the mouse is released over it. Using this method the Material is still loaded into the Material Editor but library aspect is not switched to panel, so you can continue drag and drop to test different material on the objects without the tabs switching.

Drag and Drop from a library applies the material to the object under the cursor, the Material editor is not activated.

- Method 4: Drag and Drop using windows explorer

Use left-mouse pressed and Drag and Drop a material or shader from a library or folder directly from windows explorer onto an object in the Workspace , the object under the cursor will have the material applied when the mouse is released over it. Using this method the Material is still loaded into the Material Editor but library aspect is not switched to panel, so you can continue drag and drop to test different material on the objects.

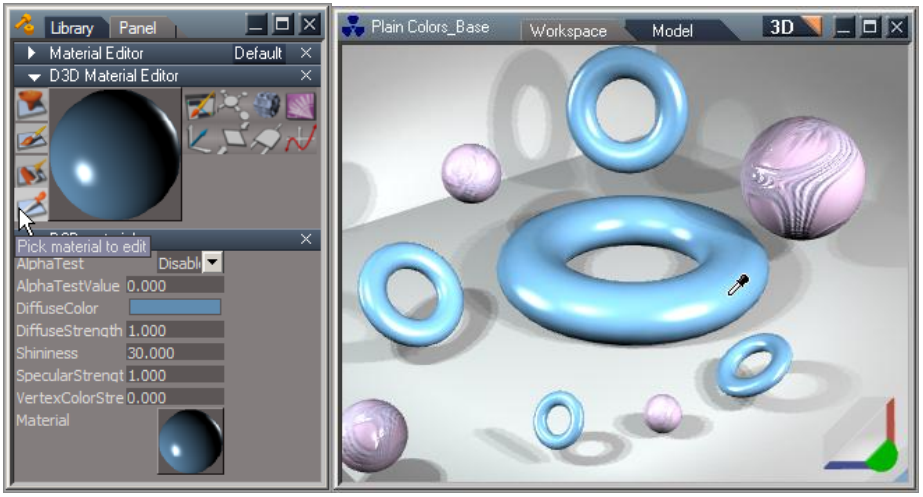
Drag and drop from windows explorer applies the material to the object under the cursor, the Material editor is not activated.

5.3.2 Workflow : Basic Editing

Method 1: Without edit picked material directly.

For Basic editing of DX materials using default settings and keeping edit picked directly unchecked, when you pick a material in the scene it loads it underneath the main D3D material editor.
Here you can change the values for the attributes in the material these are dependent on the type of material currently loaded and after making any changes you can re-apply them to the objects using the painting tools in the Material Editor or the painting tools from the toolbar in the Workspace window.

- Further reference: Included Material Samples description:

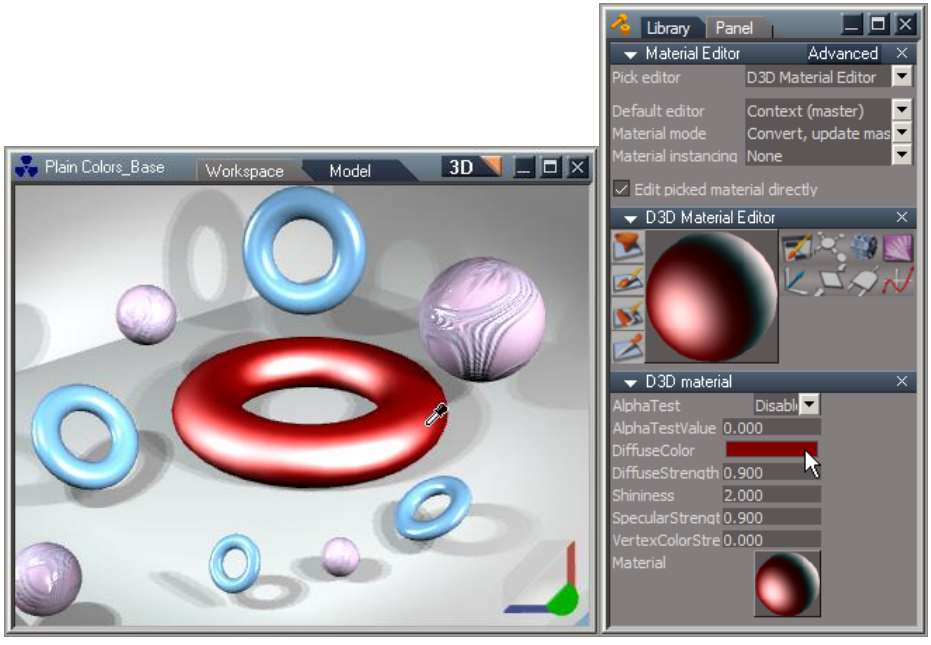


Pick to load material in the editor

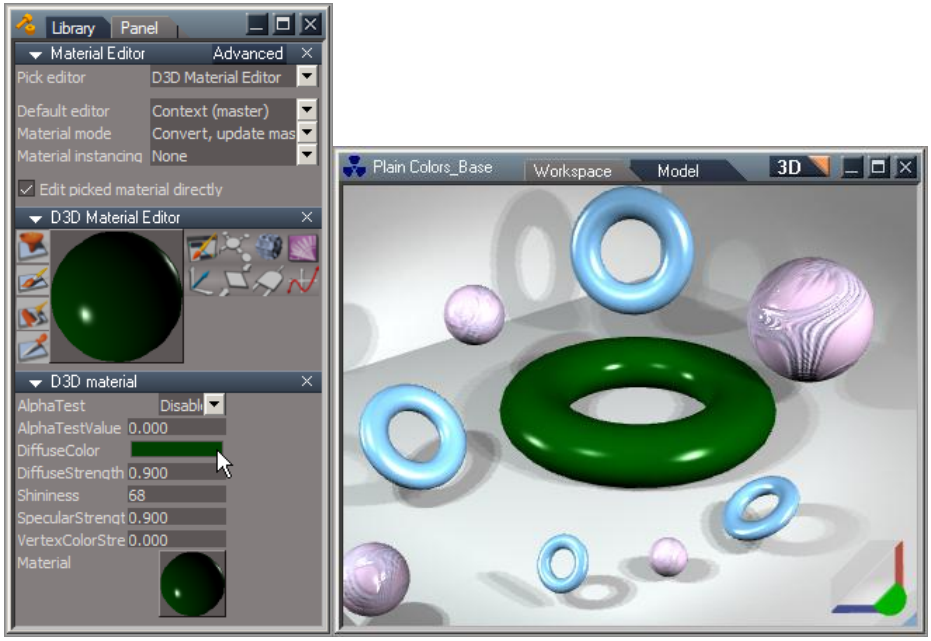
Edit the material and repaint the object

Method 2: With edit picked material directly.

This method offers a more direct surface editing and manipulation approach during basic editing of DX materials. When using default settings and setting edit picked material directly , this time when you pick a material in the scene it still loads the material underneath the main D3D material editor the same as before, but any edits made to the material will be shown on the object surface immediately, without having to use the painting tools to re-apply the changes.



Pick to load material in the editor

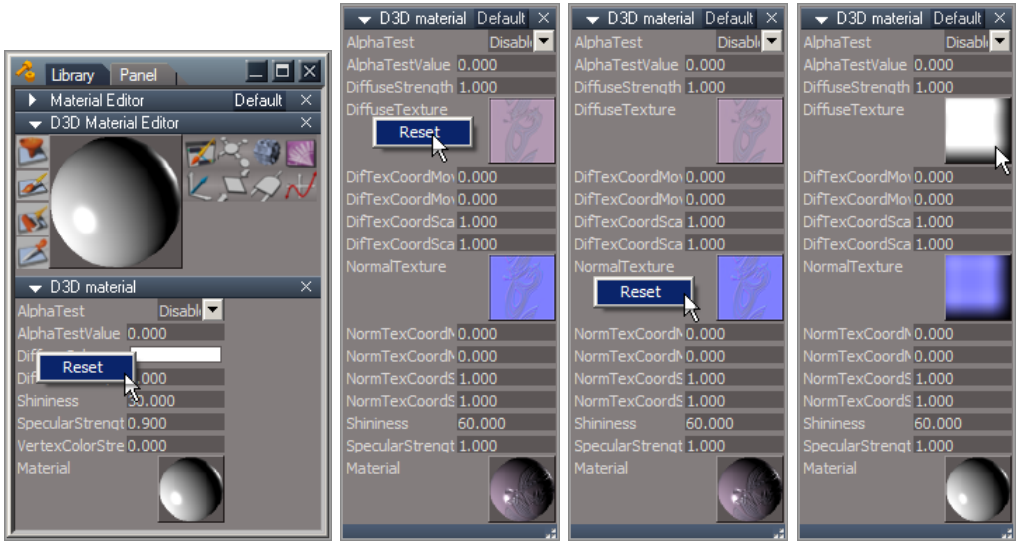


Material changes are applied to the object in the Workspace no need to use the painting tools.

Resetting a material to default values.

When a material is loaded into the editor you can r-click over an attribute and choose "Reset" to reset values to the default properties for that particular attribute. For colors and textures the default setting is white , for Normal textures it will set a flat map with no appearance of surface bumps, numeric values will depend upon the attribute being reset.

Note: this is not the same as using undo on an object, however you can use undo to bring back previous values.
If edit picked direct is used then it will reset the material on the object in the scene as well so you might want to uncheck edit direct before using the reset sometimes.



Resetting a materials attributes

Saving a material to a library:

There are two methods you can use.

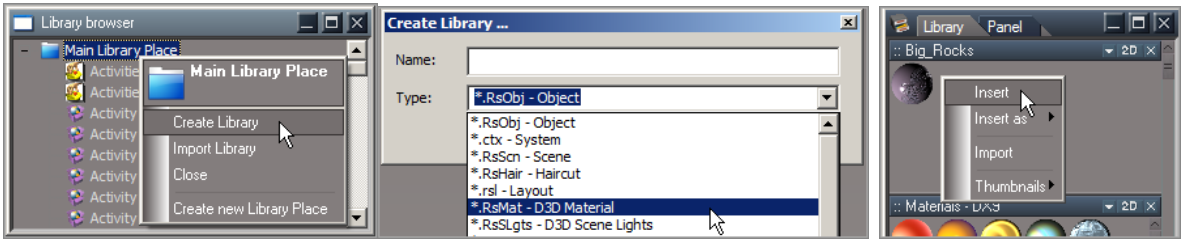
1. Select the object in the Workspace or the LE painted with the material you want to add. right-click in a Material library and select “Insert”
2. Select the object painted with the material you want to add, right-click in an object or other library type and select “Insert as D3D Material (.RsMat)”

You can either use the default libraries or create your own libraries to store the materials in from the library browser window menu.

- Further Ref: **2.3.1 Library Browser :**

Notes and Tips: if the object is part of a group or hierarchy then the individual sub-objects will need to be selected first for the material to be added to a library. You can use the keyboards Arrow keys to Navigate down into the object and then use the right and left keys to select other objects in the same level of the hierarchy, you can use the info panel to keep a check on the object selected or use the desktop preferences highlight selection to make the selection process easier.

"Insert as " a material into library



Create then "Insert " a material into a custom library

Notes and Tips:

Because the Material Editor opens in the stack view, you may find it convenient to open your material libraries in the workspace itself. To do this, open your material library as normal, then drag the window out of the stack into the workspace while holding down the CTRL key. Then you can choose and apply materials while still keeping the Material Editor open.

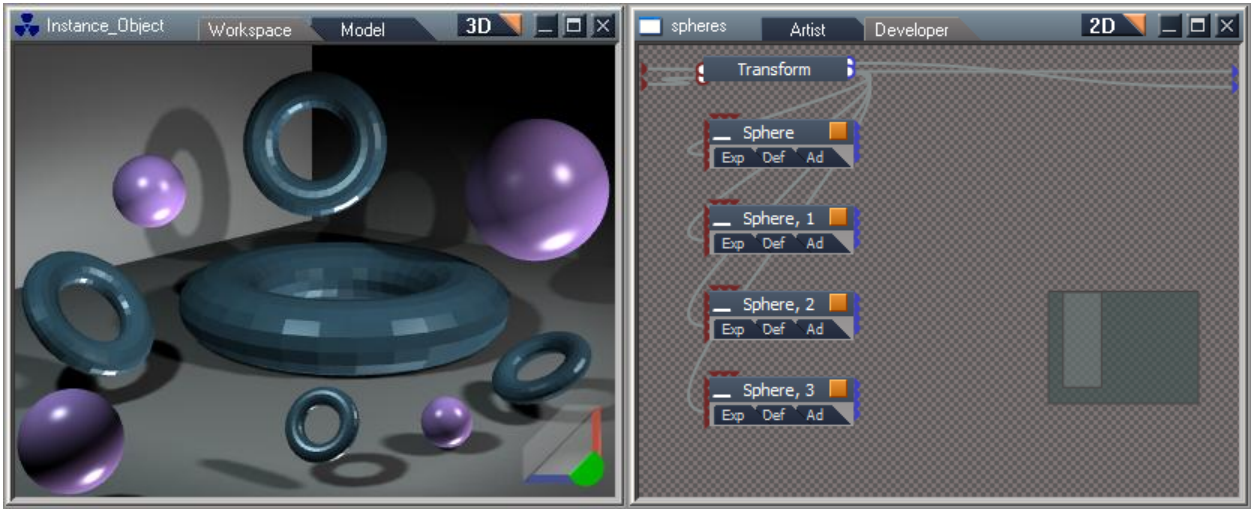
- Further Ref: Ch_2_Window_Docking

5.3.3 Workflow - Material Instancing: Object Mode

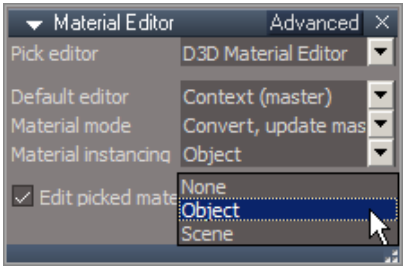
This mode is useful when you have a number of sub-objects contained in a hierarchy or encapsulated group which are using the same material, apart from keeping the size of the files down it also provides a way of quickly changing and

managing the surfaces for all objects that share the same material in an easy way, by allowing editing of a master material that is being referenced for that group or object.

The following image on the right shows what a group looks like when painted normally , each sub object contains its own individually referenced material.
You could paint the whole group with a single material but in normal non instance mode each object would still have its own reference and would need to be edited on a individual level , also there would be no saving in scene or object file sizes.



Structure of a group when painted normally , each sub object contains its own individually referenced material inside it.

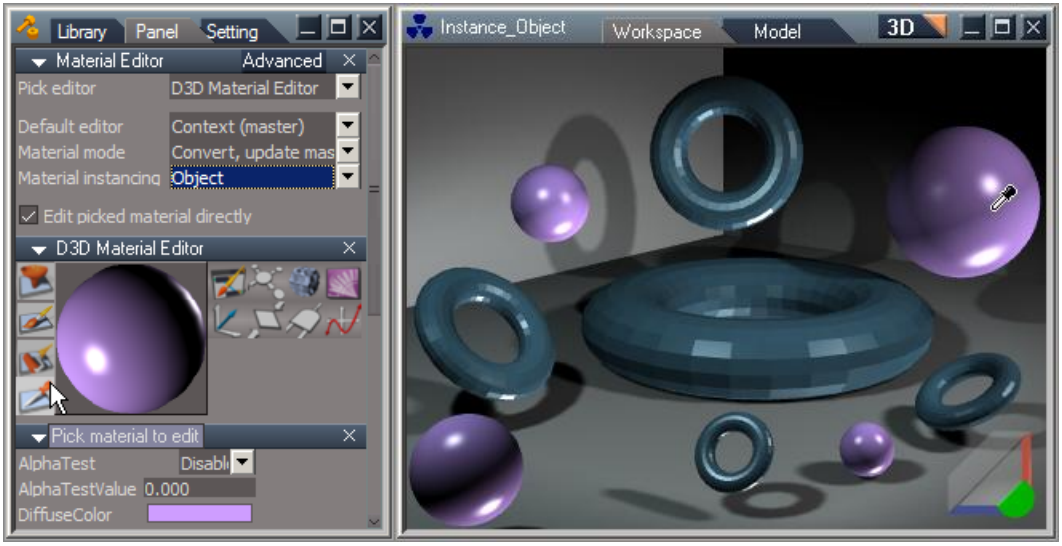


Material Instancing settings object mode.

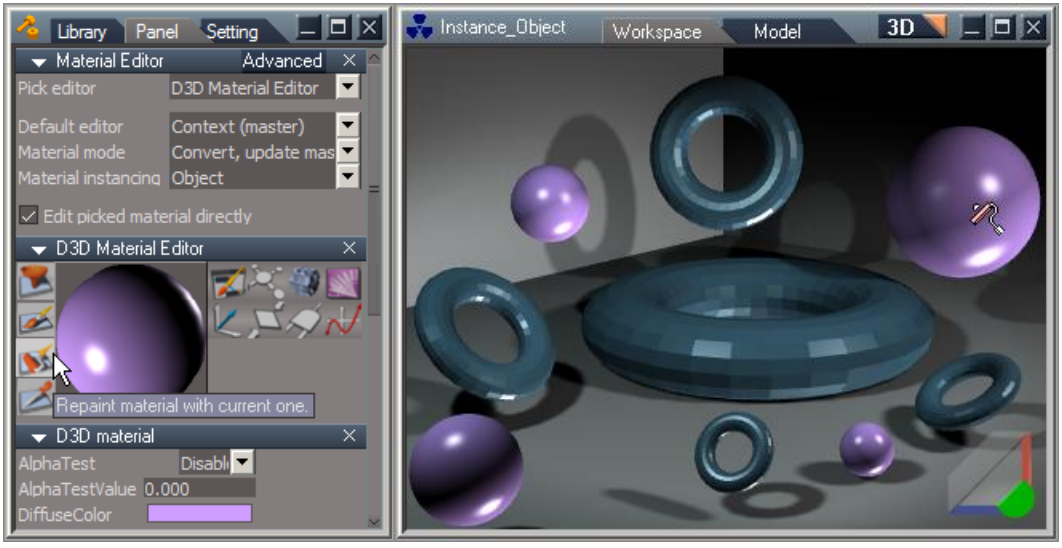
By switching to object instance and then using the repaint tool with a combination of pressing down the shift key and re-painting over each sub-object will create a common shared material instance for all sub-objects in the group that use the same material. This will also create an additional object inside the group which holds the material or materials if multiple materials are applied to the surfaces in the group, these materials can be edited in the Material Editor by picking them and if edit picked directly is checked then changes will be made immediately.

If the Shift key is not used then just the individual object will be repainted.

Note-: If edit picked directly is not checked then changes made to the material will need repainting onto the objects.
Tip-: When your preparing a scene and separating out or marking areas on surfaces to share materials in either object or scene instancing mode it can be useful to uncheck the Edit Picked material directly and make edits to the material colors ,etc, then repaint the objects with the changes.

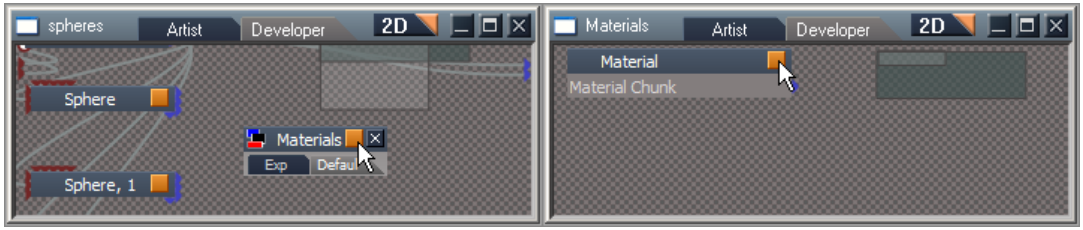


Picking a Material to Edit



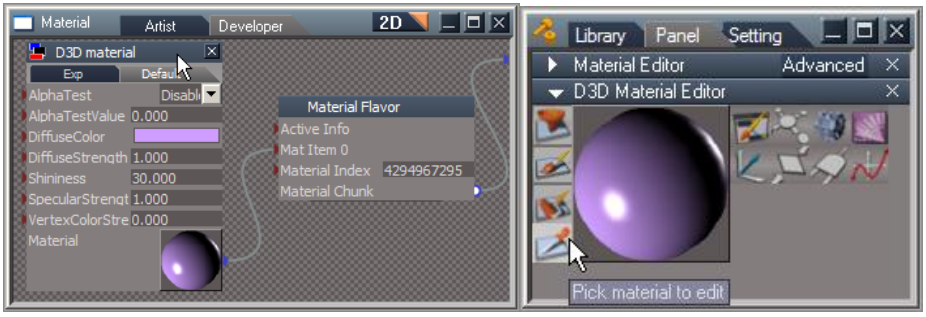
use the repaint tool and hold down the Shift key whilst re-painting will apply the material to all sub-objects that share it

- When repainting: CTRL key can be used when you want to keep target material instancing mode, for example if you have scene material instancing but your object is using object instancing and you want to do (keep) the object material instance.
- When repainting: Shift key can be used together with shared materials; for example if shift is pressed then repaint tool repaints the shared material (so all objects using it are repainted), otherwise only the objects individual material is updated.



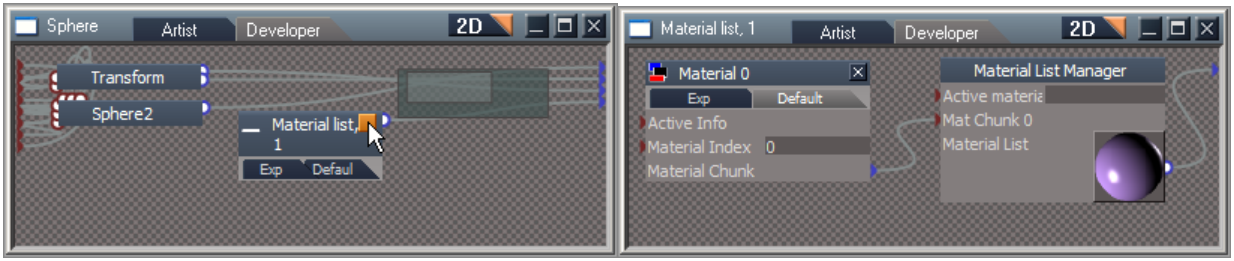
Using instancing mode creates an additional Materials object inside the group which holds the material or materials

If you navigate into the Materials object you will see the actual material or materials that are used by the grouped objects as references to this material.



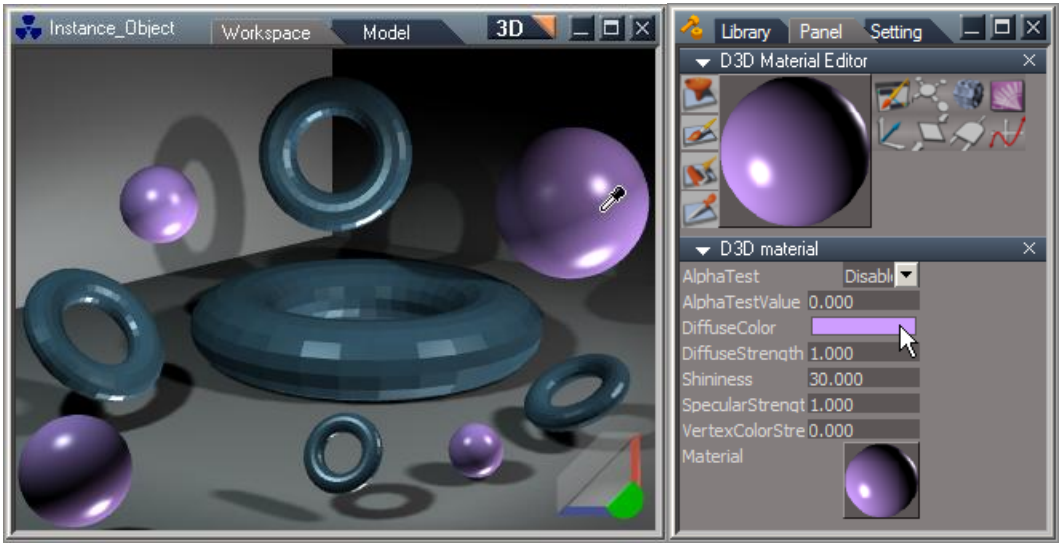
Inside the Material, you can see the actual D3D material being used.

If you look inside the sphere or one of the groups object you can see that now there is only a reference to the material and the object itself has no material inside it.

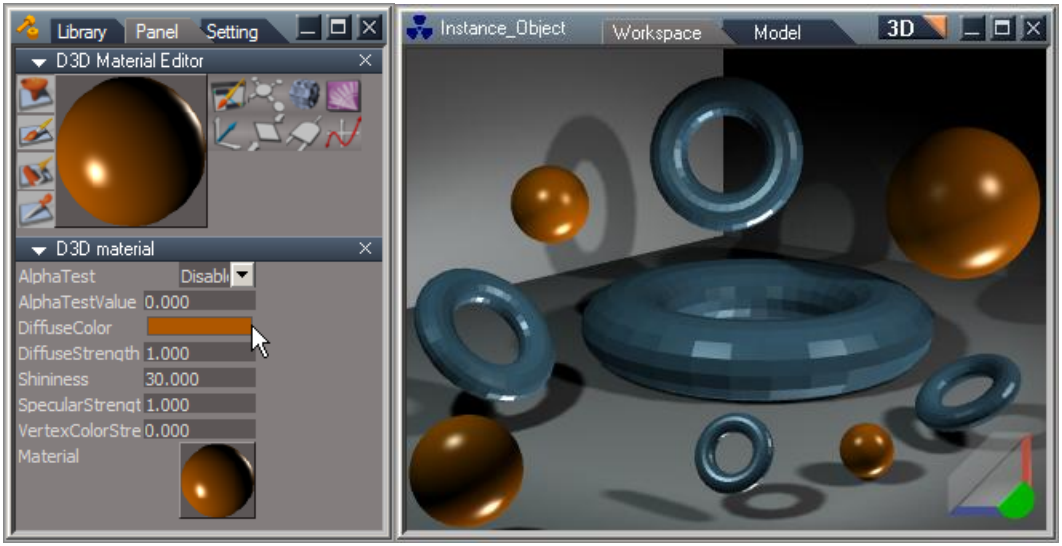


Navigating into the one of the objects only a reference to the material or materials is shown

Now when a material is picked and one of its attributes is changed or the material is replaced the changes are reflected across all the sub-objects that share the referenced material.

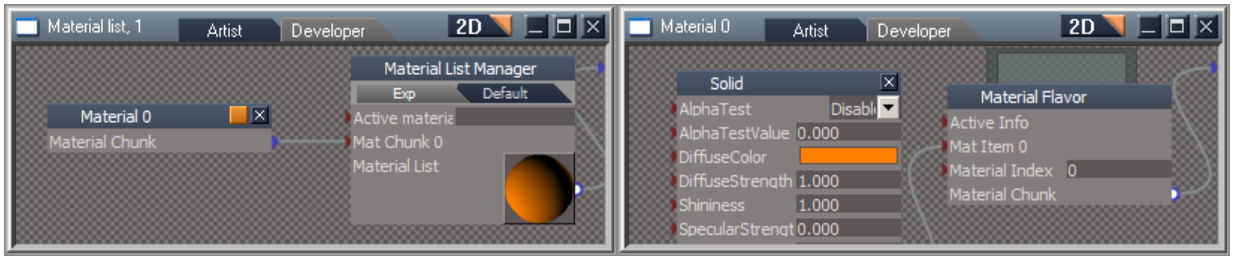


Using object instance when a material is picked and one of its attributes is changed,



the changes are reflected across all the sub-objects or surfaces that share the referenced material.

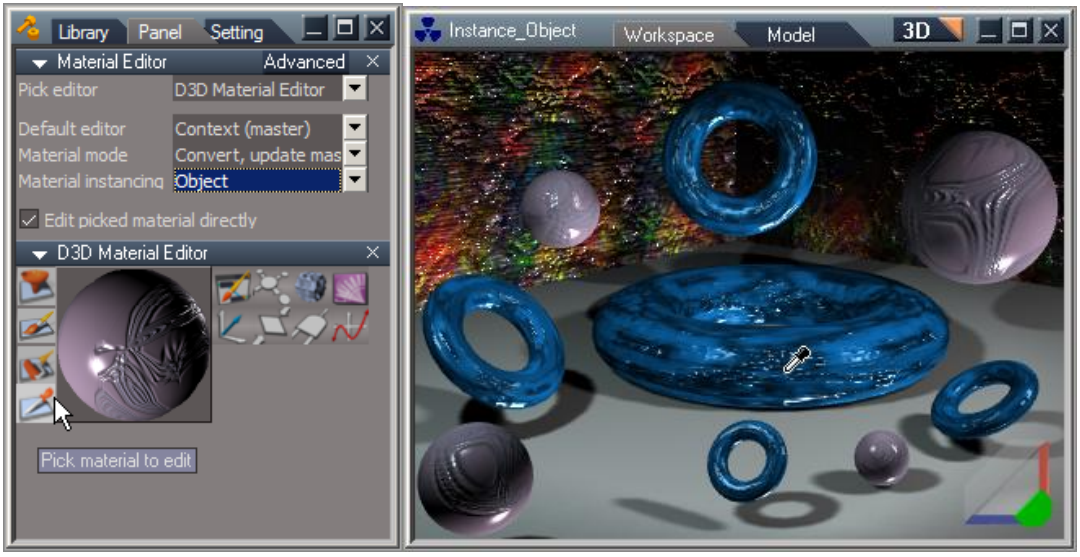
For a comparison between using instancing and none the image below illustrates that without instancing each sub-object will have its own individual material object even if they use the same material and this can produce an impact on scene size , imagine if large textures were used multiple times within an object, the overhead would soon add up.



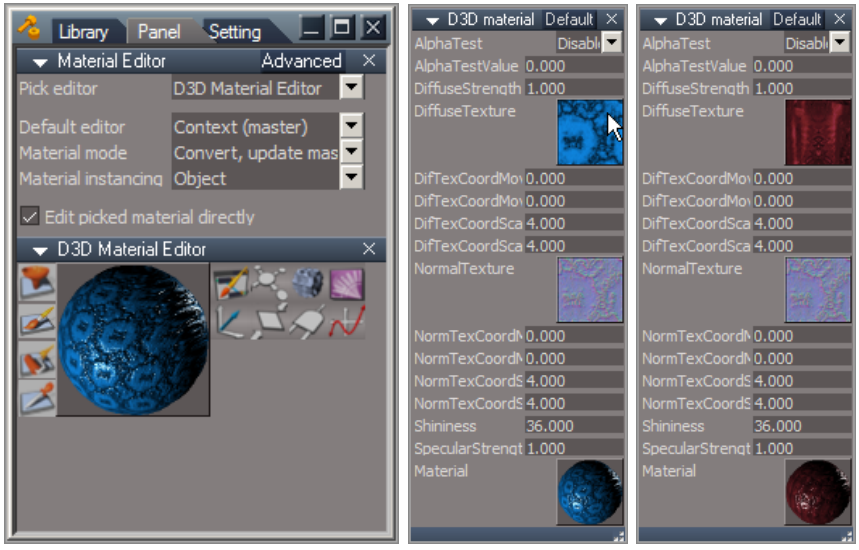
Navigating into the one of the objects painted without instancing the actual material or materials are shown

Example: Object Instancing edit direct

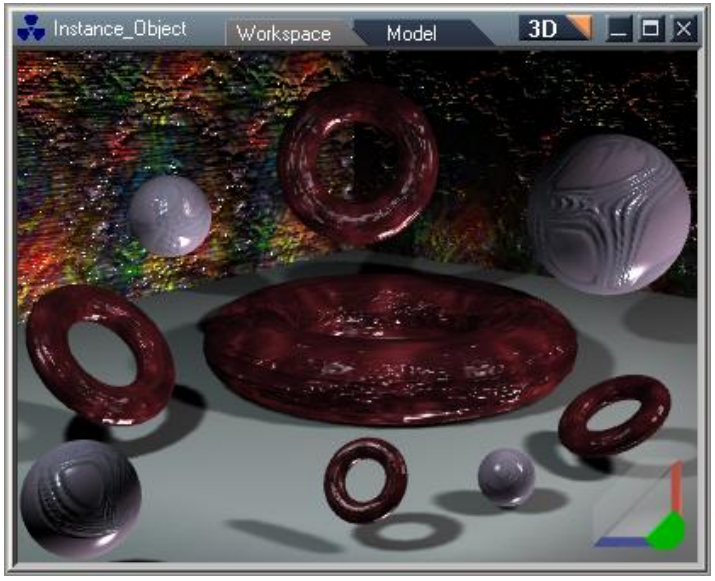
The simple example below contains the same 3 groups of objects as above, sphere , torus, cube , each group has a shared material , the ground which is still part of the cube group has been painted to retain an individual material which isn't shared with the other objects inside the encapsulation.
When using live mode and the material is picked and changed all objects in the group which share the common material painted by object instancing will be updated as well.



Picking a material to change

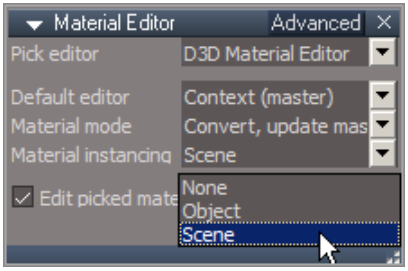


Use ctrl and Left-click on an image input area then browse to a different texture to load.



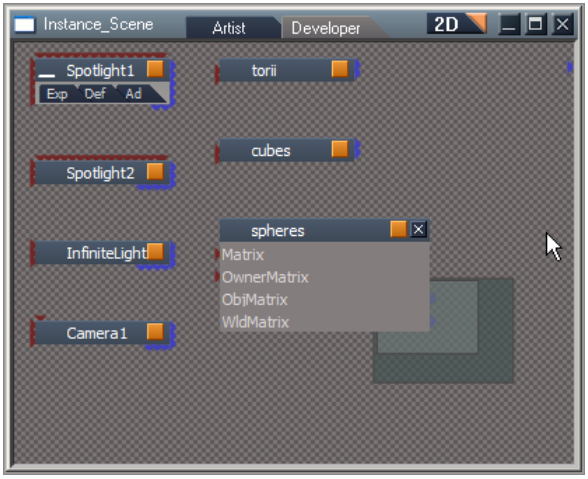
The material is replaced on all objects that share the same material.

5.3.4 Workflow - Material Instancing: Scene Mode



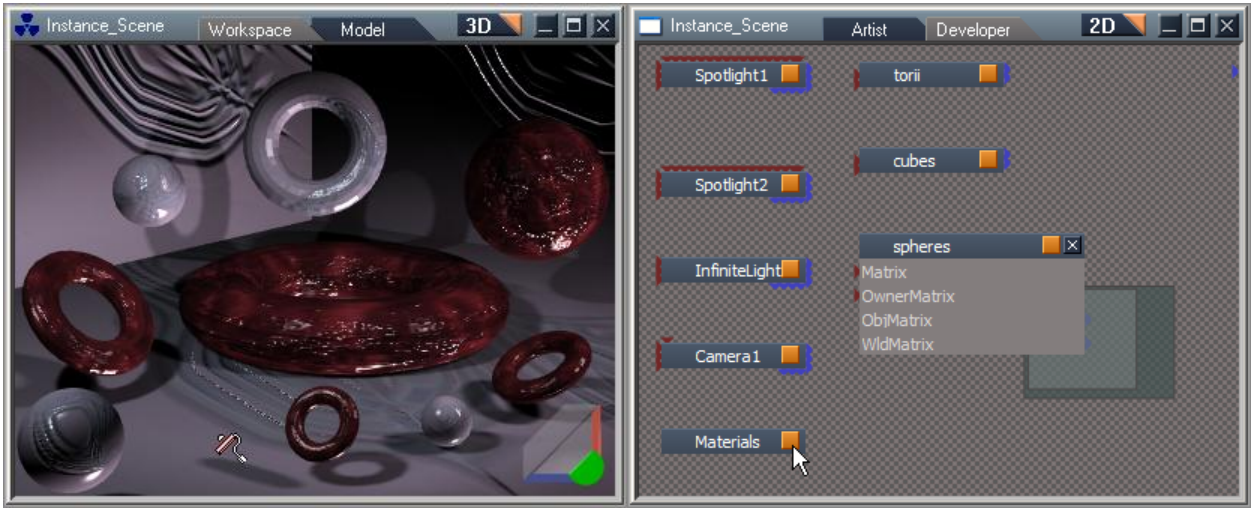
Material Instancing settings Scene mode.

Scene instancing is similar to the object instancing mode in as much as it creates a container in the scenes root (where the other objects in your scene reside, e.g. the lights cameras , meshes) to hold the shared materials that objects painted in the scene can reference.
In normal none instance mode the scene doesn't contain this Materials object.



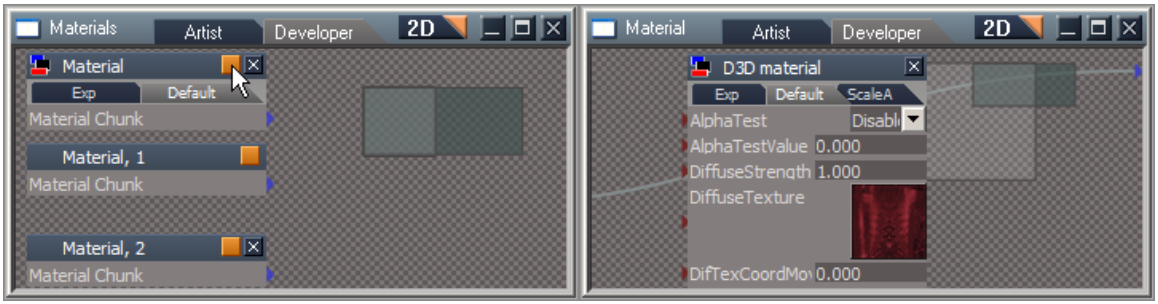
The scene root in non instanced doesn't show a Materials object

After switching to scene instancing and repainting the objects a Materials object is created in the scene root. This object will hold the materials used in the scene which other object will use as a reference. If you look inside the Materials object by clicking its orange square to enter it you can see the individual material objects.



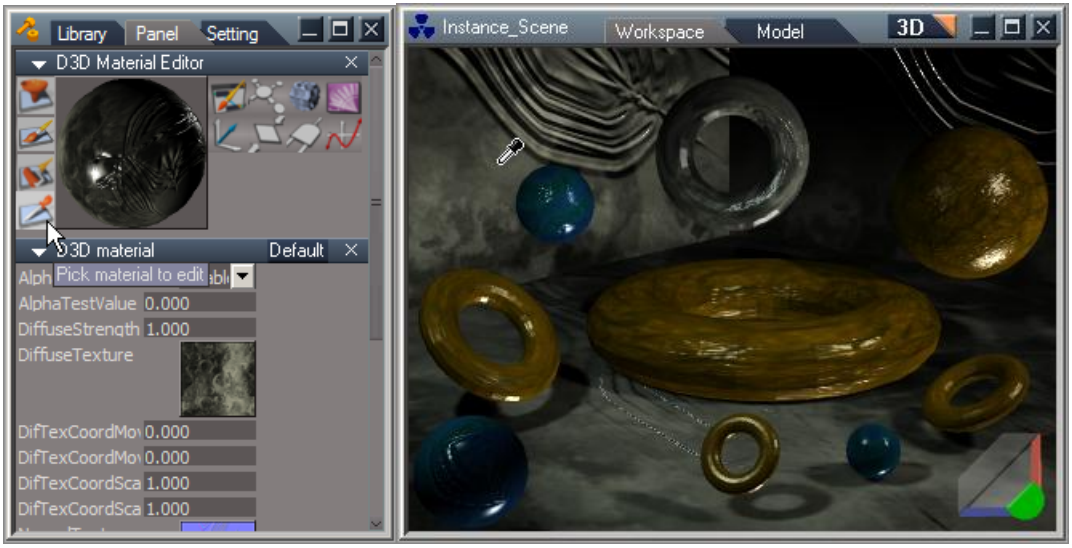
Switching to scene instancing and repainting creates a material object in the scene root

Then If you look inside the individual Material objects you can see the individual materials.



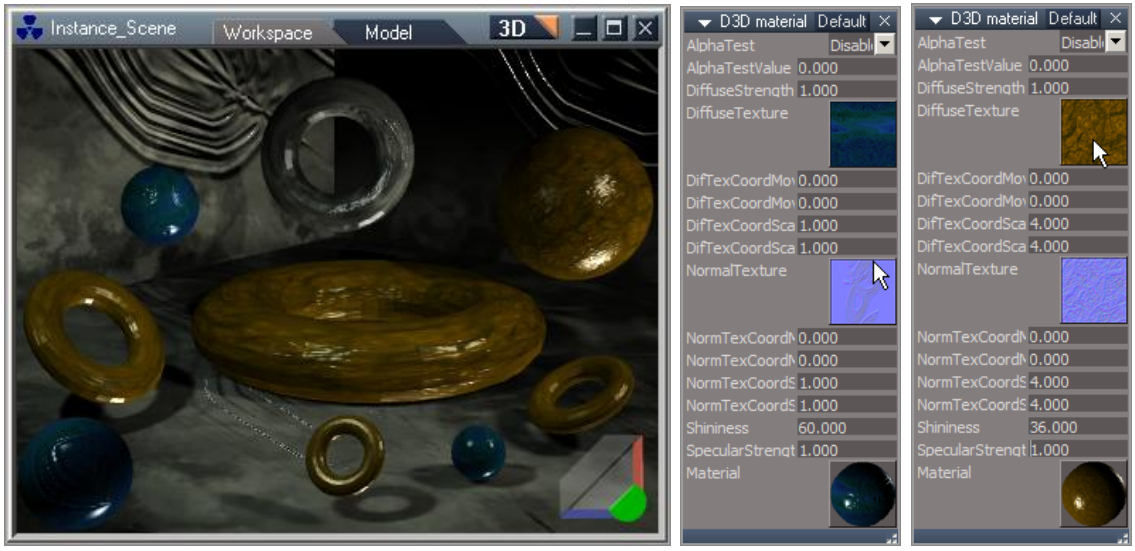
Inside the Materials object are the materials that are used and shared , entering one you can see the individual materials.

Now that objects share and reference materials it makes it very easy to quickly change the materials that are shared between objects in the scene by picking and editing , if you are using edit picked directly all surfaces will be updated as soon as edits are made , if not using edit picked directly re-applying the changes using the SHIFT key with the painting tools will update the shared material , if SHIFT key is not used then just the individual object will be repainted.



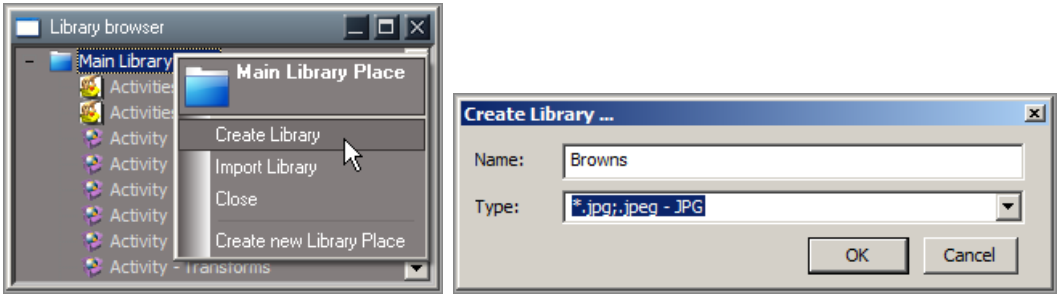
Picking and editing the Material replaces all objects surfaces that share the material

You can change the materials textures by using CTRL and Left-click over the bitmap , this will open a windows dialog where a texture can be browsed to from your local directories and loaded into the material,

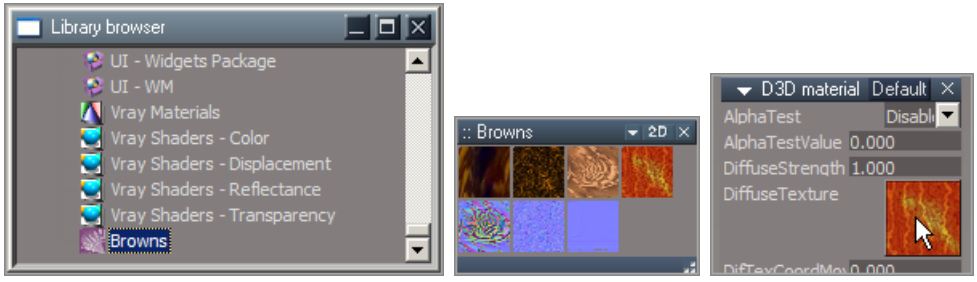


or if you have already created and populated some image libraries then the textures can be dragged and dropped from these onto the input area of the material.

- Reference: Creating Libraries .



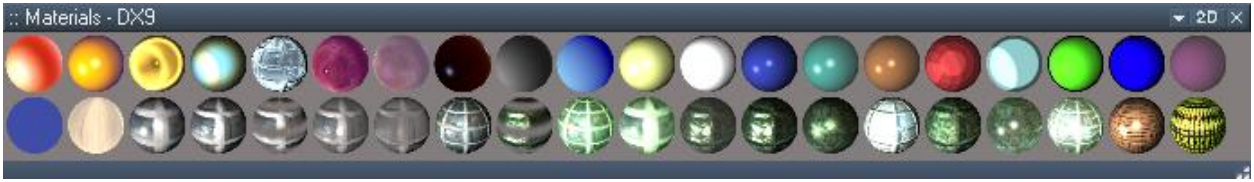
Creating a library



Opening the library and drag and drop onto the image area to replace the texture

5.4 DirectX Material Editing

trueSpace supports the display of DirectX shaders in the Workspace view. For projects such as interactive activities these materials can create **near-photorealistic** imagery in real-time. trueSpace includes a number of these materials in the libraries, but you can also create an unending variety of real-time DirectX shaders by either customizing existing materials via the **Link Editor**, or by writing custom shader scripts.

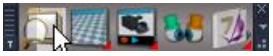


The Materials – DX9 Library

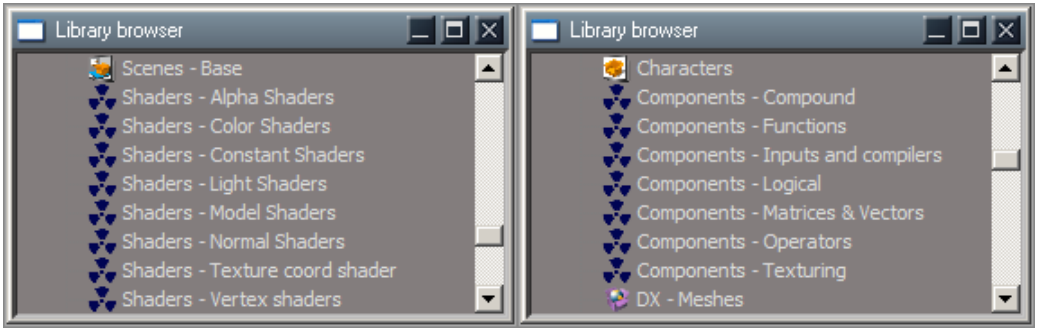
The rest of this section takes you on a tour of basic techniques for creating and editing DirectX materials and lists some of the provided DirectX shaders. These can form an ideal starting point for creating your own materials when creating real-time shaders including scripting and HLSL by using more advanced techniques.

5.4.1 Using the Link Editor as a Material Editor

When you create advanced materials in trueSpace, you will work primarily within the **Link Editor** with a selection of libraries devoted specifically to material creation.



Library Browser Icon

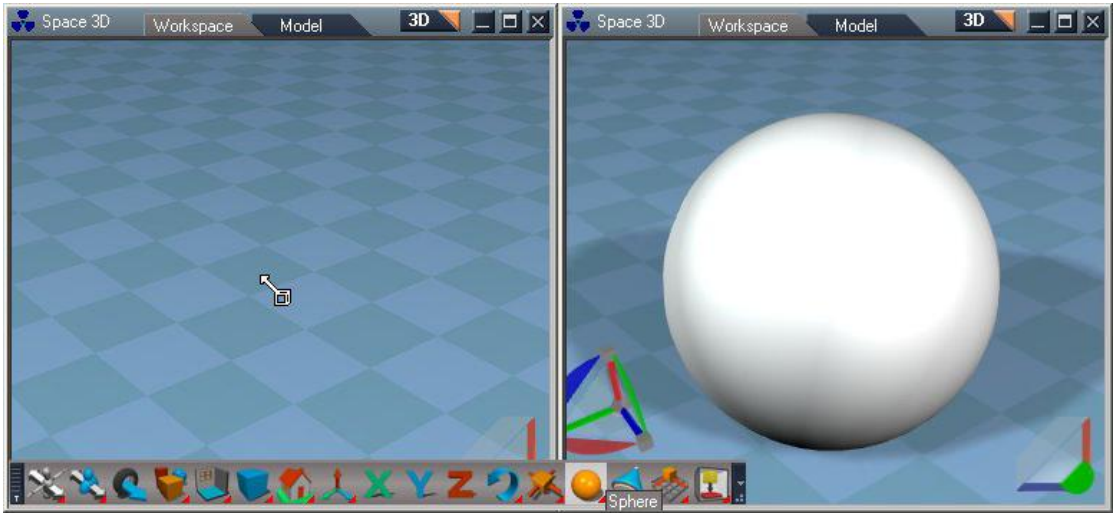


The DX-Shaders and Components – Libraries

A **material object** is visually similar to any other object that you might encounter in the Link Editor view, with the exception that it exports a Material attribute. The Material attribute tells trueSpace how to render objects that are painted with such material.

5.4.2 Examining a DirectX Material

To look at the internal details of a material in the Link Editor, add a Sphere object to the Workspace from the toolbar, by selecting the Sphere from the primitive shapes and clicking once in the Workspace , a right click will exit the tool once you have placed the sphere where you want it.

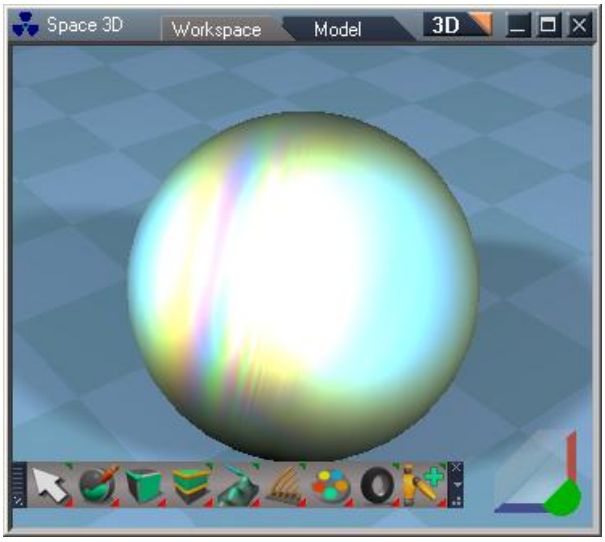


Adding a sphere to the Workspace view

Locate the ThinFilm material in the Materials DX9 library and ether double click the material in the library to apply it to the selected object or drag it onto the sphere in the Workspace view.

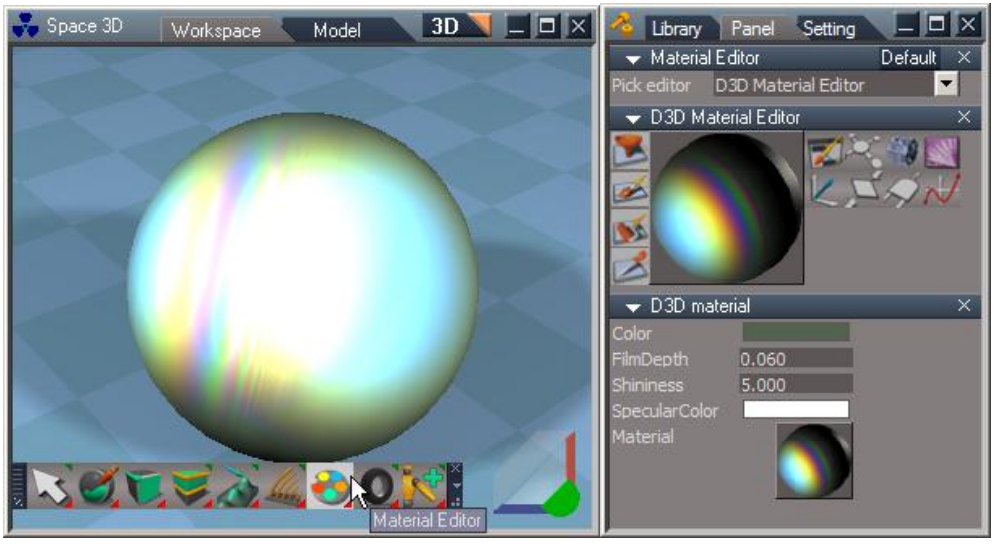
The ThinFilm material

Your sphere should now be colored with a shiny material, as shown in the image below.



The ThinFilm material applied to a sphere in the Workspace view

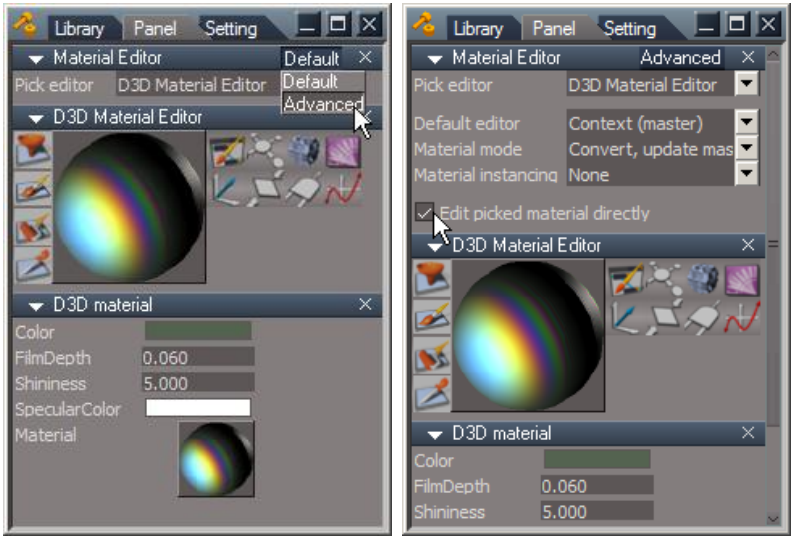
If you double-clicked to use the material then the stack should automatically switch and show the Material Editor. If you dragged and dropped the material then you will need to Activate the Material Editor by either using the icon in the toolbar or by switching the aspect of the stack from Library to Panel and the Material will be shown in the stack.



Activate the Material Editor

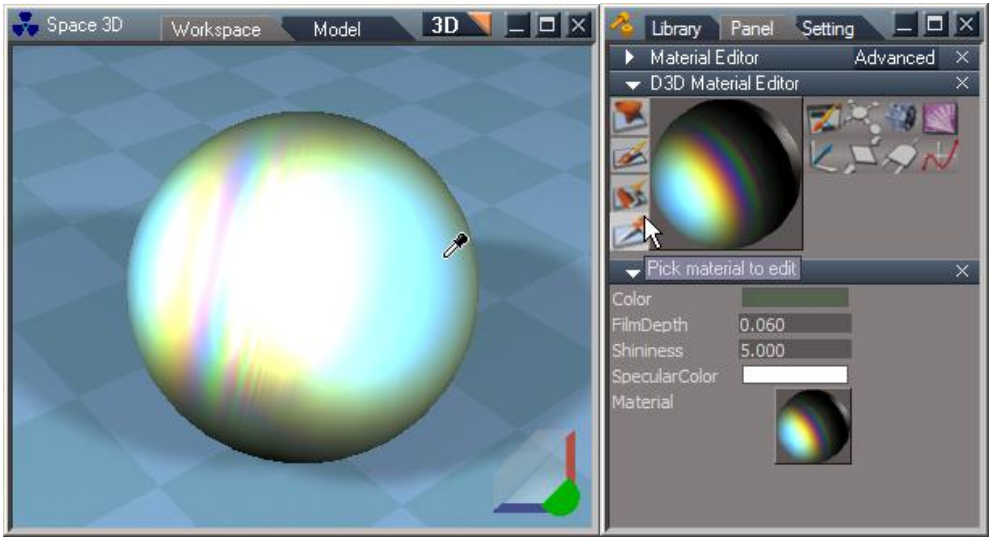
Switch the Material Editor settings to the advanced tab to get access to some additional settings , as we want to activate Edit picked material directly. This means when a material is picked the changes will be reflected directly on the

objects surface in the Workspace when the attributes are adjusted in the Materials panel in the stack, using this setting takes away the need to repaint the object with the Material editors tools after each change is made.



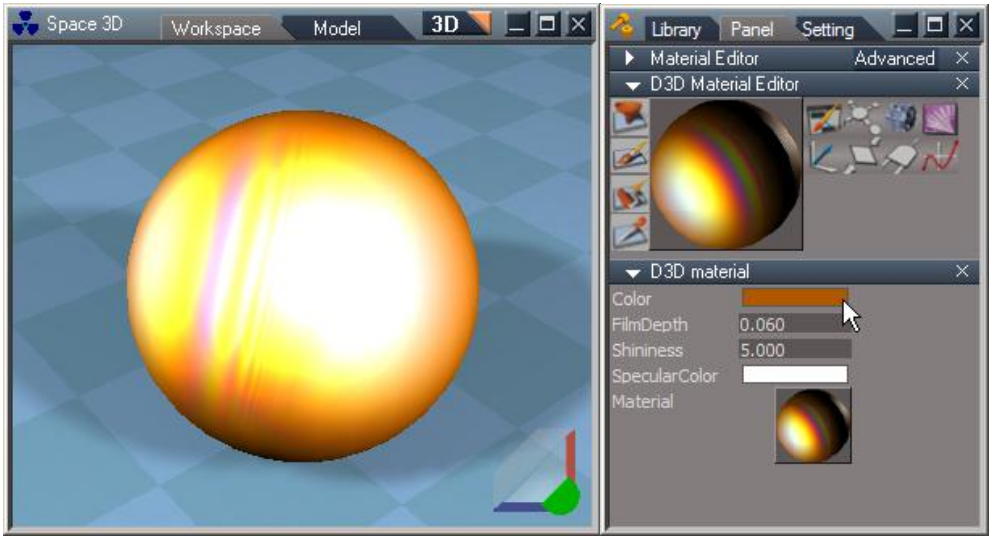
Switch to advanced aspect and check the Edit picked Material directly

Then using the pick tool re-pick the sphere in the workspace to obtain the material applied to it, if you do not re-pick then the material shown in the editor will not relate to the same material that exists on the sphere and adjustments will not be made on the object even though they will show in the Material editors preview.



Use the pick tool on the object.

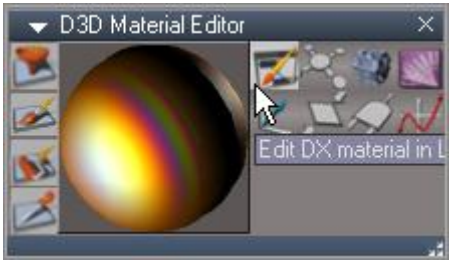
Now when you adjust the colors or other attributes on the material the changes are applied directly on the surface of the object in real-time . Below I simply double-clicked over the Color to open up and changed it to orange.



Changes are applied in real-time directly on objects surfaces in the Workspace as the material is adjusted

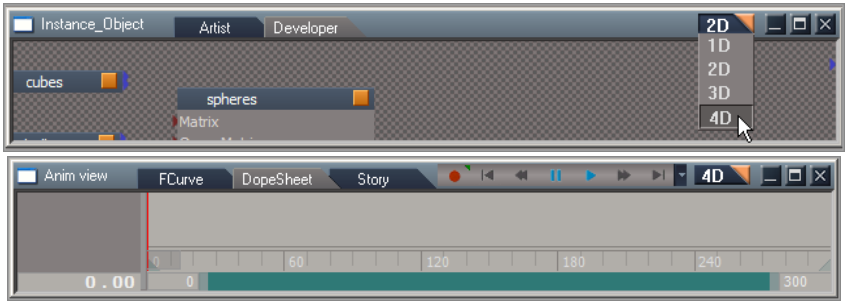
Note- Also using edit picked directly when you want to perform some advanced editing on the Material it will open the Link Editor to edit the actual material on the object and not a version of it that's "held" in the material editor waiting to be painted onto the objects when not in the Edit picked materials mode.

To enter the advanced mode of DX-Material editing click the top left icon in the ME interface doing this will either switch your current Link Editor window focus to the D3D Material or if no Link Editor 2D window is currently being used a new floating Link Editor will open which is focused on the material.



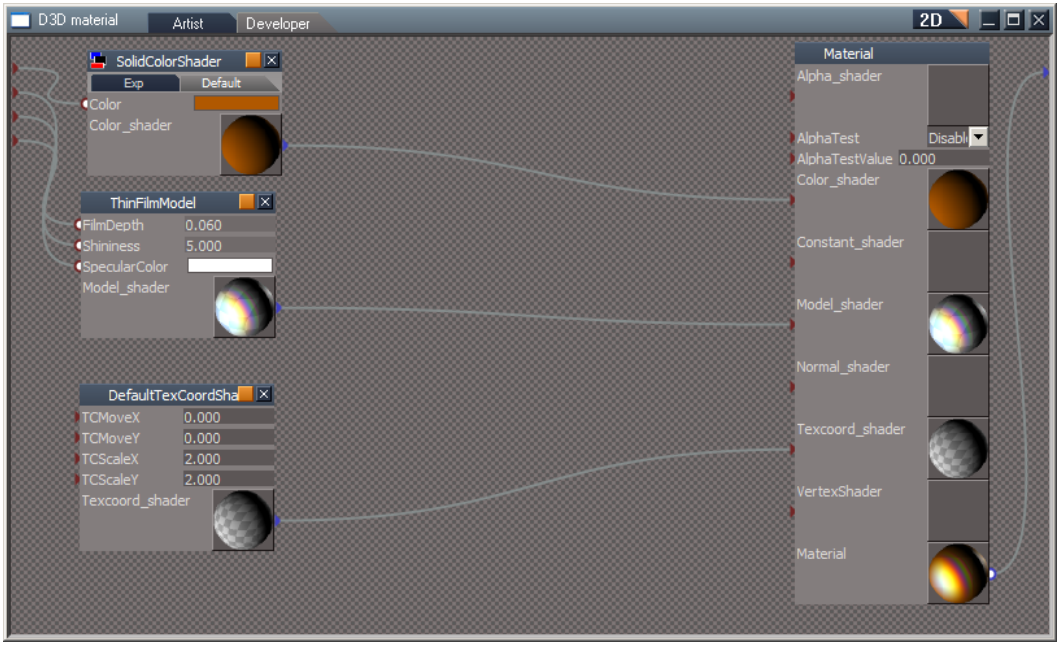
Edit DX-material in the Link Editor

Tip: if you want to maintain the current Link editor focus then switch its aspect to 4D and the Material Editing will take place in a new 2D Link Editor window , you can then switch your 4D window back to 2D and continue using it again afterwards.



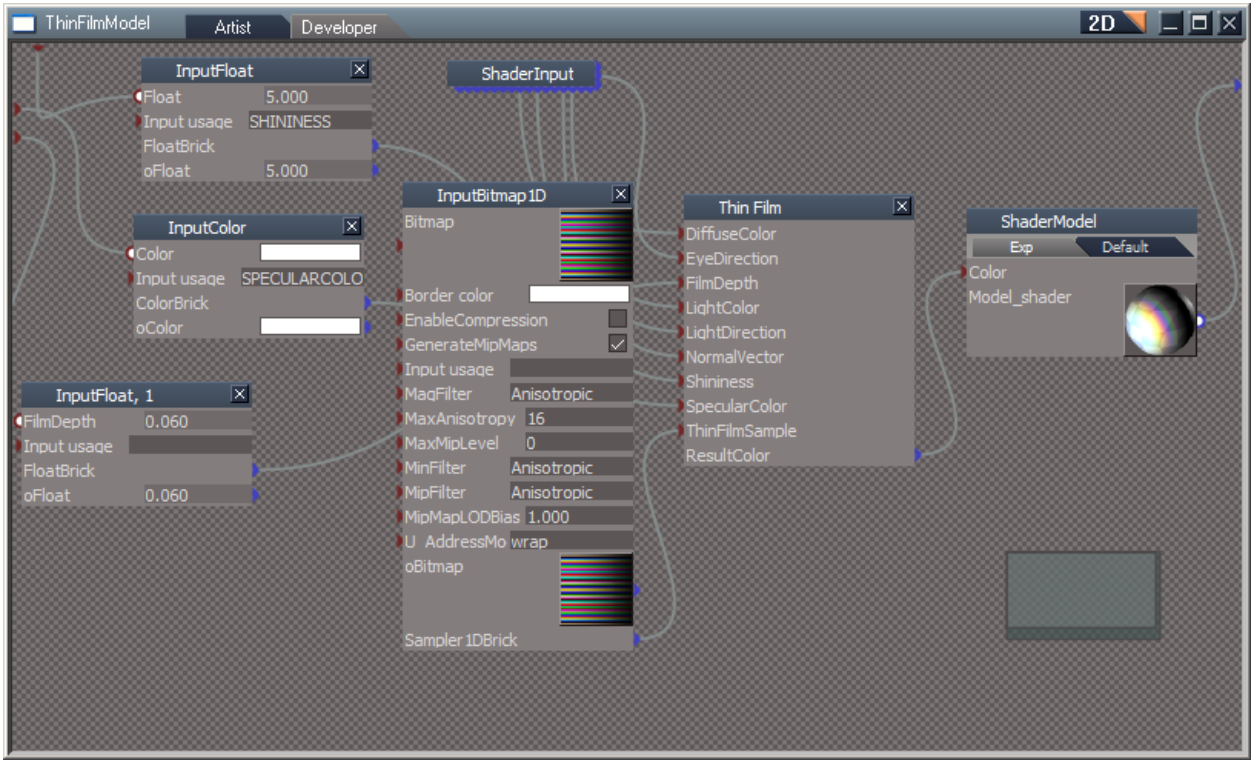
Inside the Link Editor you will see the parts that work together to create the material's effects. This material contains a Material object and several shader components that feed into it: SolidColorShader, DefaultTexCoordShader, and a ThinFilmModel model shader.

Tip: If you don't see the links like in the image below change the tab in the Link Editor window to Developer aspect.



Inside ThinFilm we find the shader components that create layered reflection

In the Link Editor, you can enter any object that displays an orange enter icon in the upper right-hand corner of its panel. For instance, click on the ThinFilmModel shader component to see its contents.
You should see something like the image below.
Tip: the Link Editor title bar name changes to reflect which layer of an object you are currently in.

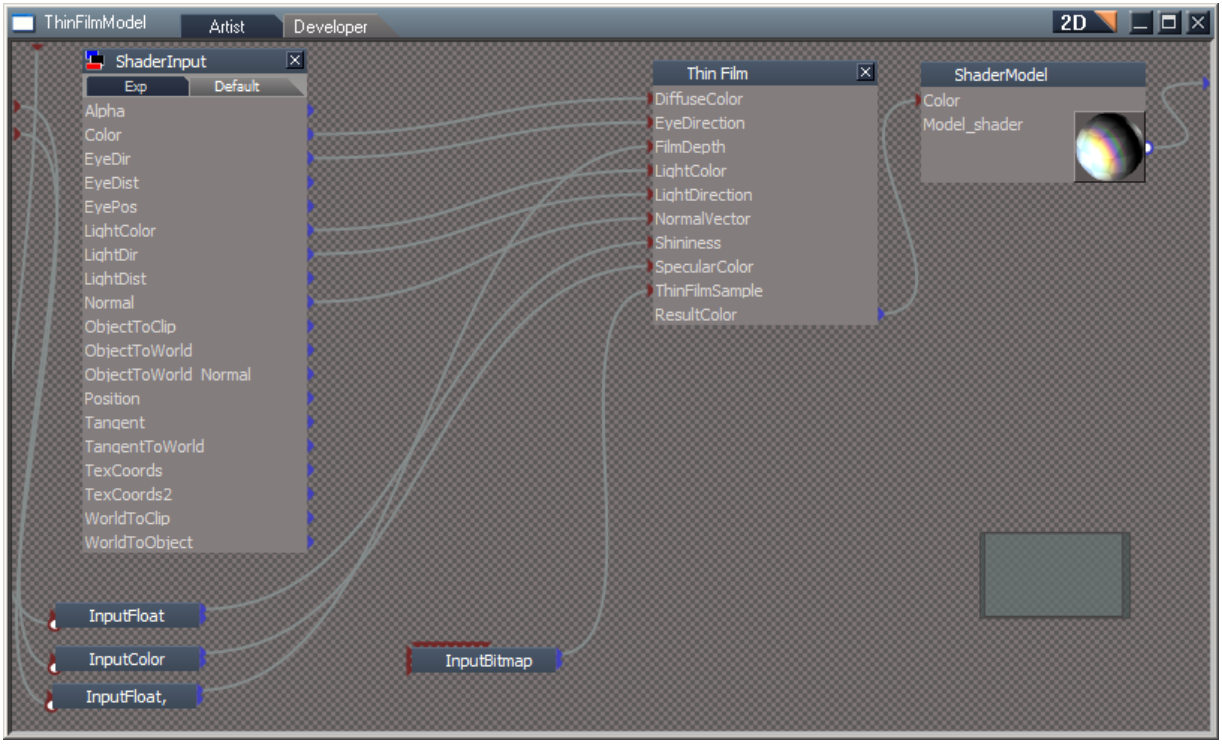


Another level of detail resides inside the ThinFilmModel shader component

The main parts of this shader component are the Thin Film object, which takes lighting and surface information to determine the final output, the ShaderModel object, which exports that result for use with the Material object, and the ShaderInput object, which provides the shader component with data about lighting and other conditions.

The Thin Film object takes in two floating point numbers, provided by the InputFloat objects, to control Shininess and FilmDepth. It also takes a color, provided by InputColor, to use as the specular color of the material. Finally it takes the input of an InputBitmap1D, which is a one-dimensional bitmap with alternating light and dark bands. This bitmap is sampled to provide additional shading information for the material, resulting in the circular "oily puddle" effect you can see on the sphere.

The ThinFilm object also gathers data from the ShaderInput object to make its calculations. Inputs taken from this object are Color, EyeDir (because the effect is view-dependent), LightColor, and LightDir (to determine the contribution from lights in the scene), and Normal (because the effect depends on the surface normal – the shape of the surface – of the model itself).
Below is the same material with the LE rearranged to show the ShaderInput and its connectors.

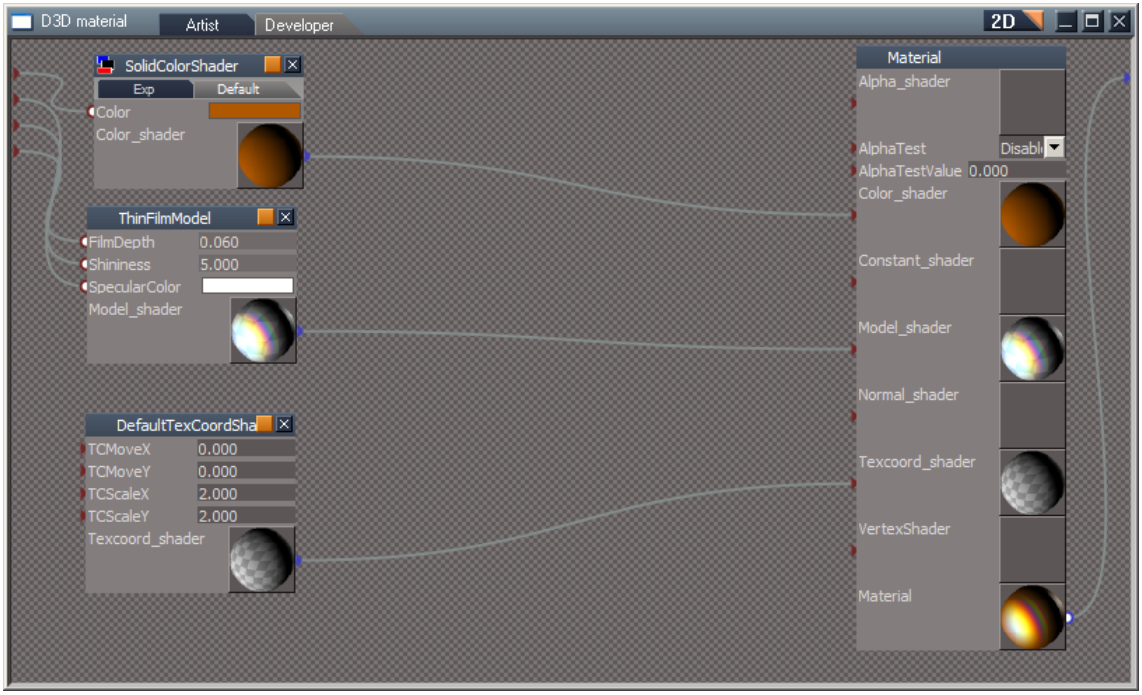


ShaderInput Connections to Thin Film modelshader

Modifying a DirectX Material

You can, of course, modify any of the connections or objects you find in a material using the Link Editor. For instance, to add an alpha component.

Exit the ThinFilmModel shader component one level by clicking the orange triangle near to the 2D in the objects title bar so that you can see the Material object.

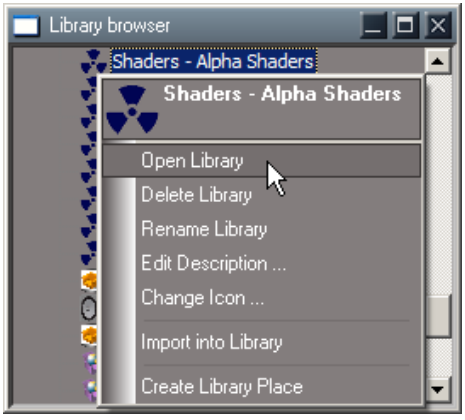


Material Object

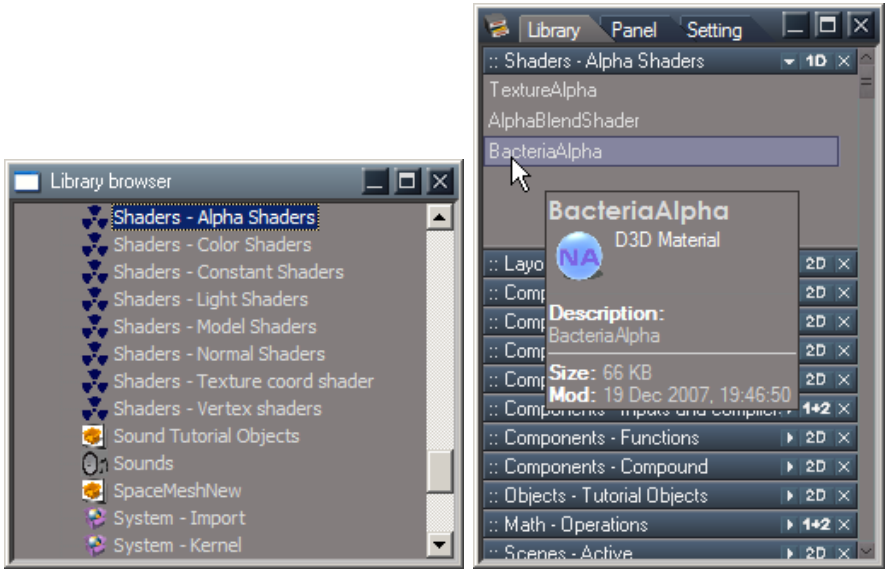
Open the library Browser by pressing the icon in the toolbar.

From the Library browser window open up the Shaders –Alpha Shaders library by either a double click or right click and choose open from the menu, it will open the library in the stack then drag in a BacteriaAlpha shader component from the Shader library into the LE.

Library Browser tool Icon

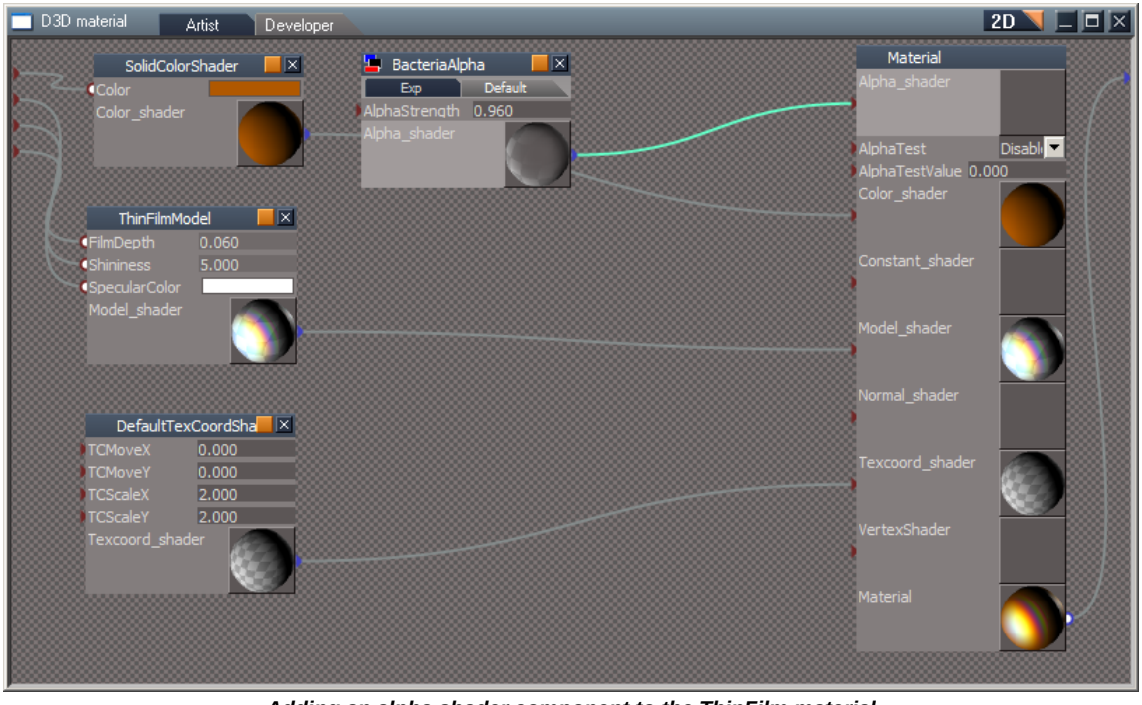


Library Browser menu



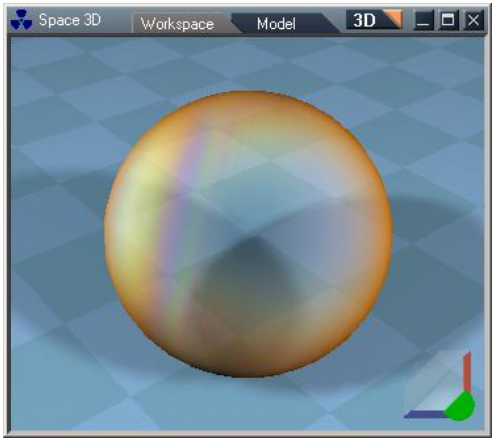
Shaders – Alpha Shaders library and BacteriaAlpha shader component

Connect its Alpha Shader output to the Alpha Shader input connector on the Material object. Try typing in different values for AlphaStrength .



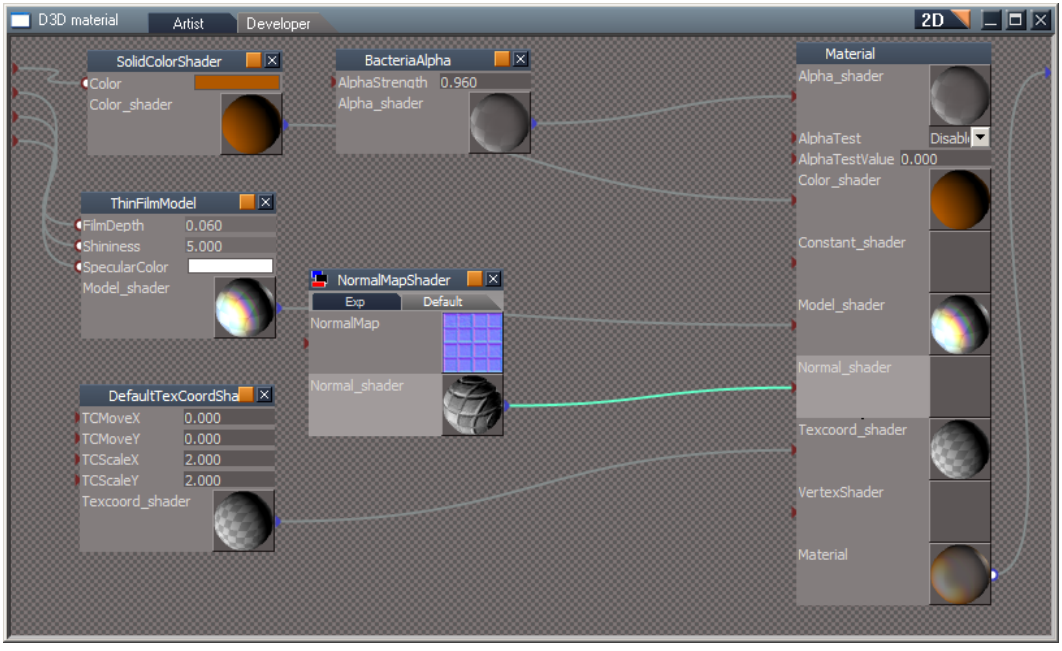
Adding an alpha shader component to the ThinFilm material

When you have made these changes take a look at the Workspace view. Your sphere should look something like the one in the image below. By adding an alpha shader component we have altered the look of the material by introducing transparency.



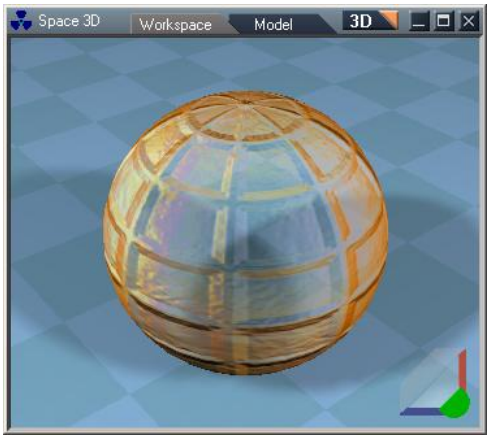
The Workspace view shows the sphere with transparency added

You can use the same procedure to add a "bumpiness" effect to the material. Try dragging in a NormalMapShader component from the Shader library and connect its Normal Shader output to the Normal Shader input connector on the Material object (see below). You can change the bitmap used for the NormalMap if you desire, or just keep the default grid bitmap.



Adding a normal shader component to the ThinFilm material

When you have made these changes take another look at the Workspace view. Your sphere should look like the one in the image below, unless you changed the default normal map. By adding a normal shader component we have introduced bumpiness to the material, giving the sphere an ”embossed” look.



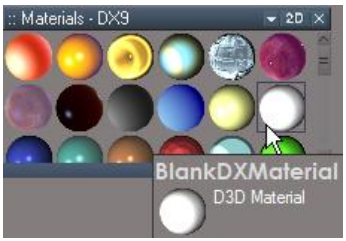
The Workspace view shows the sphere with a normal map added

Often when you are designing a material, you can simply use an existing material from the library and modify it to create the desired effect. You can delete or modify existing shader components, or create new ones using the various components found in the library. As an exercise, before attempting to construct a material from scratch, browse through the Material library and examine how the materials are built and how their various attributes are used to determine the final look of a material.

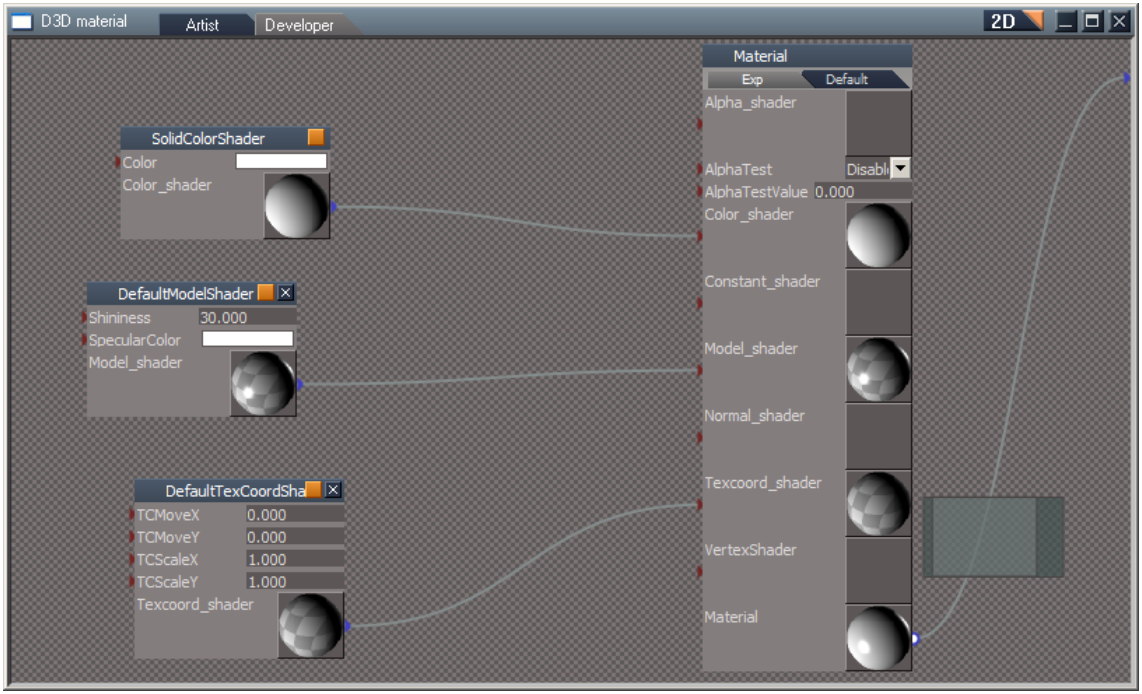
- Go Back to D3D Material editor Start
- Go Back to D3D Advanced start

DirectX Material Creation

When you want to go beyond using and modifying existing materials you will want to try creating your own materials from the components found in the Shaders library. To get started creating your own material, first create a Sphere object from the primitives tool as we did earlier. Then locate the BlankDXMaterial in the Material library and apply it onto the sphere in the 3D Workspace view.

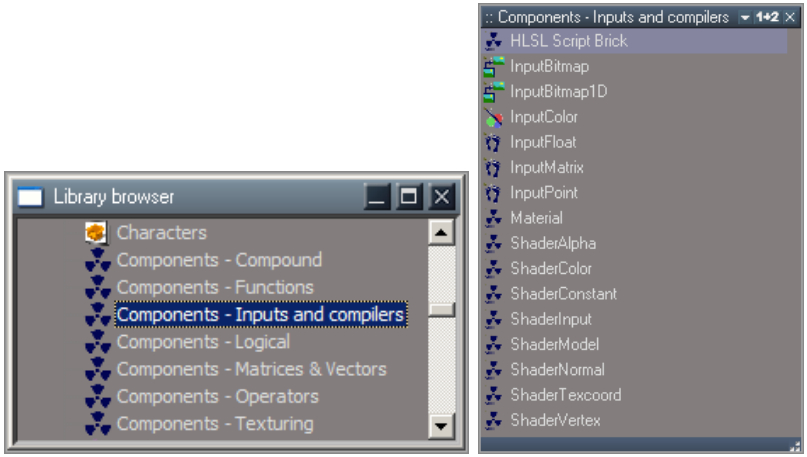


This material template can be used as a base on which to build a new material. If you enter the material you will see that it contains a SolidColorShader with a white color selected, a DefaultModelShader, and a DefaultTexCoordShaders. You can replace these with your own shader components and add an alpha shader, normal shader, and a vertex shader if desired. Simply connect these to the inputs of Material to add them to your material.



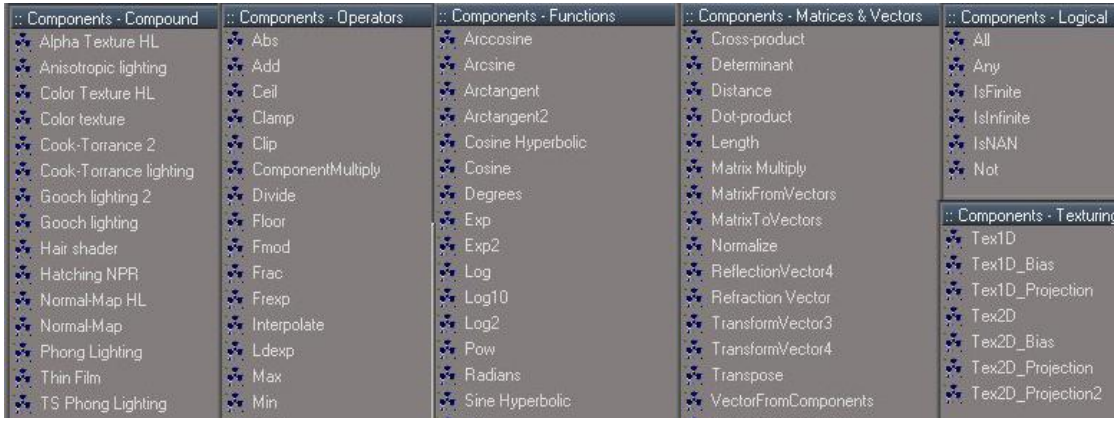
The contents of BlankDXMaterial in the Link Editor

While exploring the creation of your own materials, you might want to examine the following list of commonly-used objects shader objects found in the Components library.



Components – Libraries and Components - Inputs and compilers

- ShaderInput: The shader input object is used to bring data for each pixel from the rendering engine to the shader. Use it to obtain information (such as surface normals) for calculations that will drive the appearance of your material.
  - Detailed Reference: [ShaderInput.]
- Component Input Objects: Several input objects are available to provide data to your shader components. These include InputBitmap and InputBitmap1D, InputFloat, InputColor, InputMatrix, and InputPoint.
- Shader Component Compilers: For each shader component type, there is a special object that compiles input data into output data of the appropriate type for input into the Material object. These objects are ShaderAlpha, ShaderColor, ShaderConstant, ShaderModel, ShaderNormal, ShaderTexCoord, and ShaderVertex.



Other Shader Components Libraries

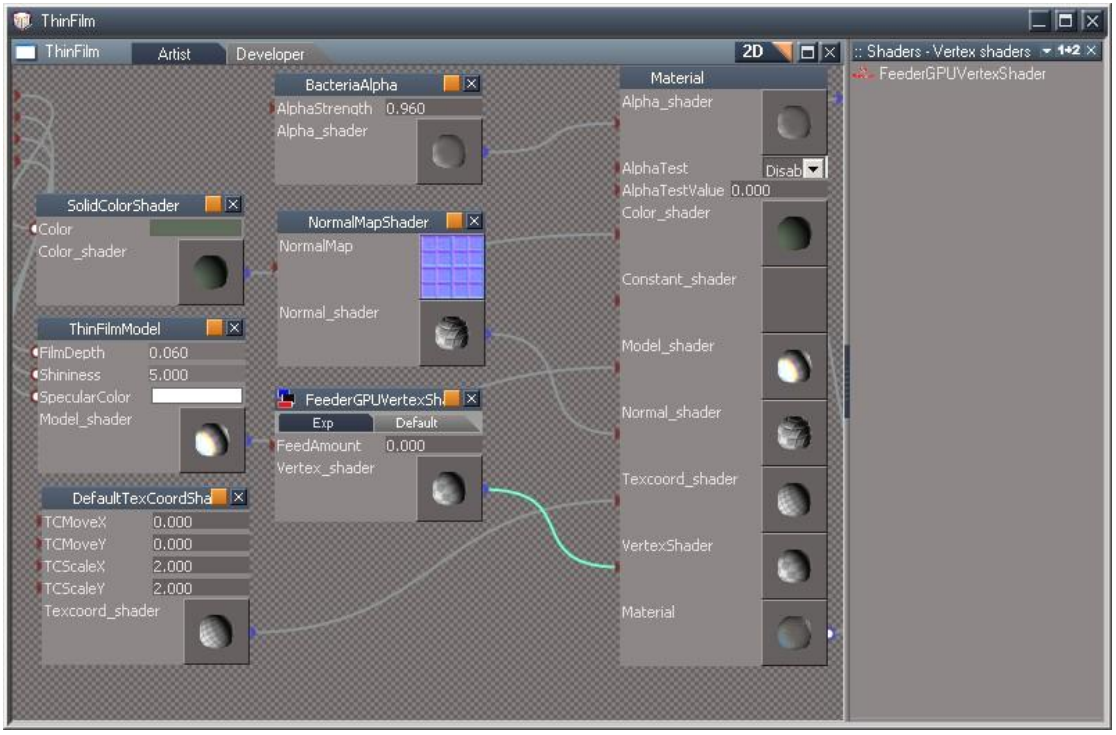
- General Functions and Operators: There are a number of objects for performing calculations within shaders. These include matrix and vector operators like Dot-product and Cross-product, logical operators such as Not and Any, and mathematical functions such as Arcsine and Cosine.
- Other Shader Objects: The Shader library also includes a number of additional components that can be used to construct materials. For instance, the Tex2D object is used to extract a color from a bitmap using a set of 2-dimensional texture coordinates.

You should also bear in mind the purpose of each of the various shader components. These are covered briefly in the following list.

- And are covered in more depth here: Ref: Inbuilt Shader Components descriptions:

- Alpha Shader: This is the shader component where you can specify any information about transparency in your material. You can do this simply using an alpha map, or by using a more complex algorithm of your own devising, such as making the material more transparent as the surface normal faces the view direction.
- Color Shader: This shader component is responsible for the basic surface coloration of your material. As with alpha shaders, you can simply apply a bitmap as a texture or use a solid color. Alternatively, you could try to implement a more complex blending routine that takes into account the surface normal of the underlying geometry (see the terrain tutorial below).
- Constant shader: texture and color data for the surfaces remains static and is not grossly affected by lighting conditions.
- Model Shader: This is where most of the shading work for your material is handled. It defines the interaction of the surface with lights and other factors. You can use one of the model shaders from the library or implement your own model. A computer graphics reference text that covers lighting models might come in handy if you plan to explore creating custom model shaders.
- Normal Shader: This component is used to describe the "smoothness" or "bumpiness" of a surface. You can apply a normal map, a special kind of bitmap that specifies normal direction at a given point on the surface, or implement your own normal modification routine.
- Texcoord Shader: Changes made here affect how textures and other maps are applied to your model. Using just the DefaultTexCoordShader object you can change texture offset and scaling. You can also apply your own mapping models if you want to achieve special effects such as animated textures.
- Vertex Shader: This is a special shader component that alters the apparent position of your model's vertices based on conditions you specify. For instance, you can use a vertex shader to "slide" the vertices in and out along their normal vectors, creating an expanding and contracting surface like a soap bubble. Many unique effects can be achieved with vertex shaders.

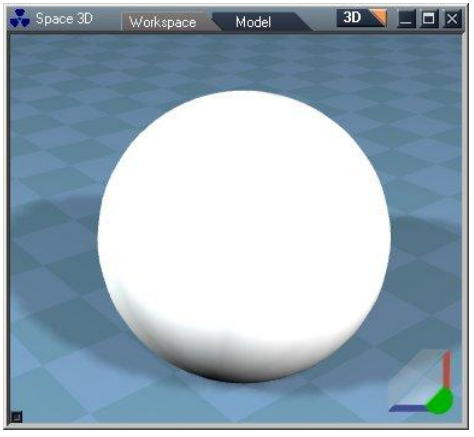
Add the Feeder Vertex Shader to the material and adjust the feed amount to see the effects, one thing that can be done here is the feed amount could be exported out of the material and a timer could be added to control the amount of feed to create an effect of the object swelling or shrinking according to the feed amount being applied and processed by the material compiler.



FeederGPUVertexShader added to the material

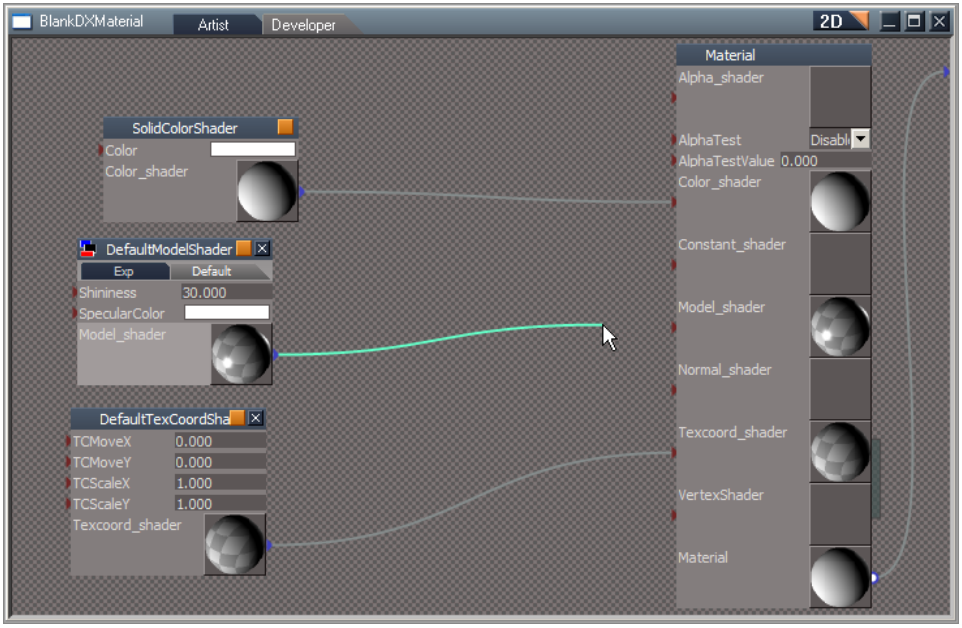
Tutorial: Creating a Simple DirectX Material

In this example we will create a simple DirectX material in the Link Editor using only stock shader components from the Shader library. To begin, create a Sphere in the Workspace view, and then drag the BlankDXMaterial onto the sphere in that view. You should see something like the image below.



The BlankDXMaterial applied to a Sphere in the Workspace view

Enter the Sphere object in the Link Editor ,navigate to, and enter the material ,You will see these components ; SolidColorShader, DefaultModelShader, and DefaultTexCoordShader.



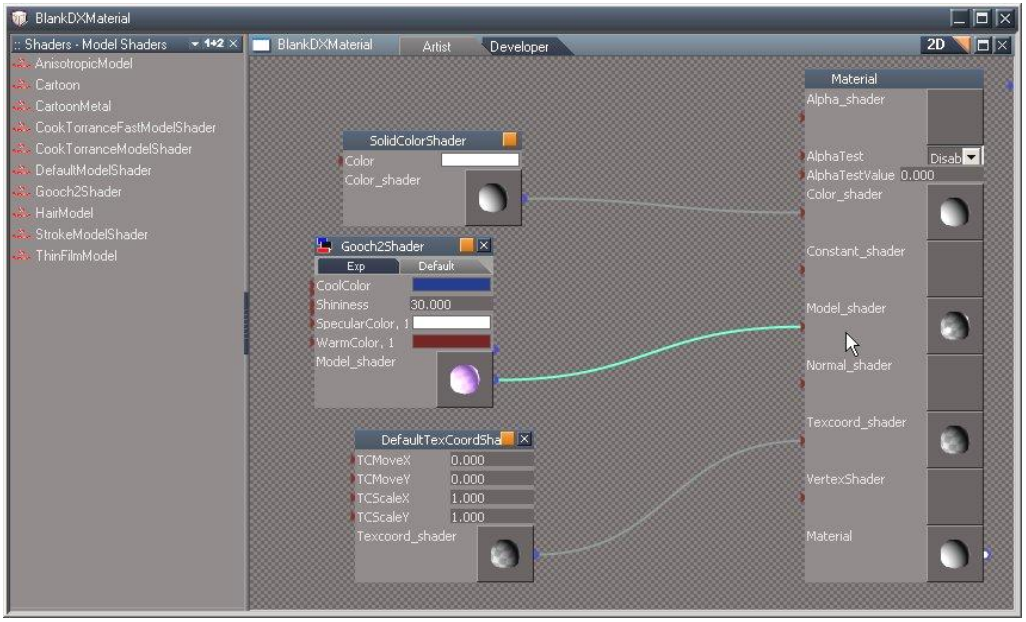
Break the link between the DefaultModelShader and the Material object

First, we will try changing the model shader to see what effects we can achieve quickly. A good way to show what a model shader really does is to take it away and watch the results. Break the link between the DefaultModelShader and the Material object.

The sphere should now appear completely white in the Workspace view, with no shading at all.

The model shader defines how light interacts with the object and material to create the final appearance of your 3D object when rendered. The DefaultModelShader uses a lighting model whose results can range from matte to glossy plastic. Let's try a different model shader.

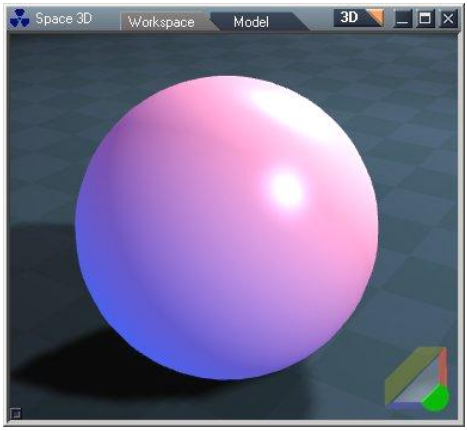
Open the Shader library and find the Model Shaders panel. Drag Gooch2Shader into the Link Editor, and then connect its Model Shader output to the Material object.



Replace the model shader with the Gooch2Shader

If you want to you can delete the infinite lights in the scene for a more dramatic effect Your sphere should now be more colorful .

You can experiment with changing the various colors by double-clicking them. Try changing the CoolColor to a greenish hue and observe how it changes the look of the sphere in the Workspace window.

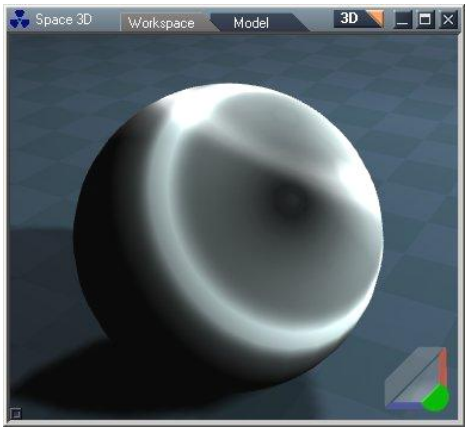


Gooch2Shader model and the infinite lights deleted from the scene

Now delete the Gooch2Shader, drag in the AnisotropicModel object from the library, and connect it to the Material object. Your sphere should change appearance again, as in the image below.

If you rotate the view around the sphere you will see how the materials surface appearance reacts to the lighting

conditions and the eye direction.



The Sphere with the AnisotropicModel shader component added

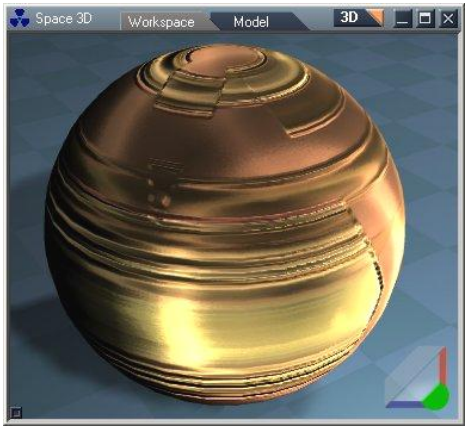
Now for a little color, delete the SolidColorShader and drag in the TextureShader object from the library (Shaders|Color Shaders). Connect its Color Shader output to the Color Shader input on the Material object. Feel free to change the bitmap to something more interesting than the default grid. Here is an image of our sphere, with the default texture in TextureShader applied, and here I added an omni-light to the workspace as well.

The Sphere with the TextureShader component and an omni-light being added to Workspace

Let's add one final component: a **normal shader** component. Find NormalMapShader in the Shader library and drag it into the Link Editor. Connect its Normal Shader output to the Normal Shader input of the Material object. Now the Sphere has a bumpy grid to match its grid texture.

The finished material and its result in the Workspace view

Truthfully, our finished material does not look like much, but this lesson has hopefully shown you the basics of building a material from ”stock” shader components. Try some other library items to see how they affect your final material, and also be sure to enter library components that have an orange enter icon to examine how they are built. Also try making small changes to see how they affect the output. If you break something, just delete the object and get a new one from the library.

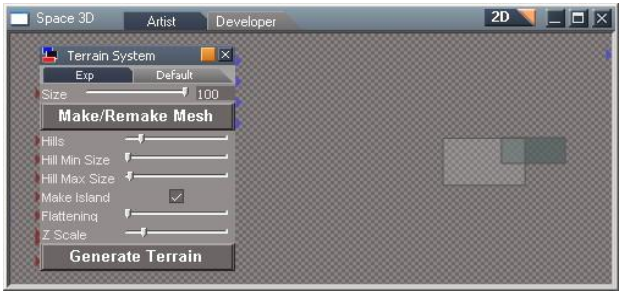


The finished material with some custom textures and its result in the Workspace view

Tutorial: Examining a More Advanced DirectX Material

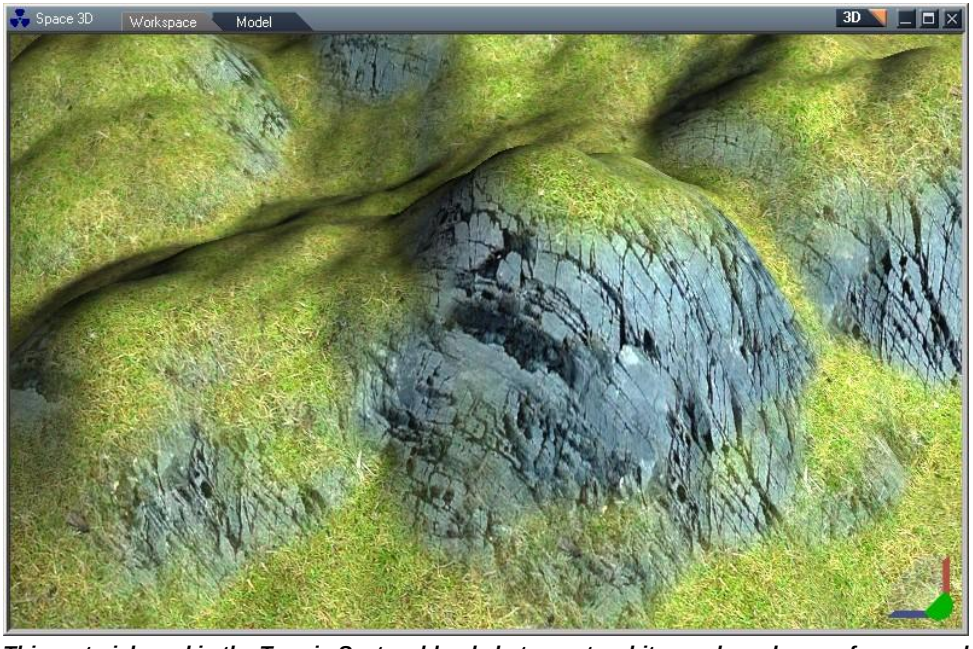
In this tutorial we will go inside a more complex material, in this case a material that blends between two bitmaps based on the direction of the surface normal. First, locate the LandScape generator object (Objects|Script Objects) and drag it into the Link Editor.

Script objects Library



LandScape generator Terrain System object dropped to the LE

You should see a set of rolling hills in the Workspace window. (You may need to zoom out a bit.)

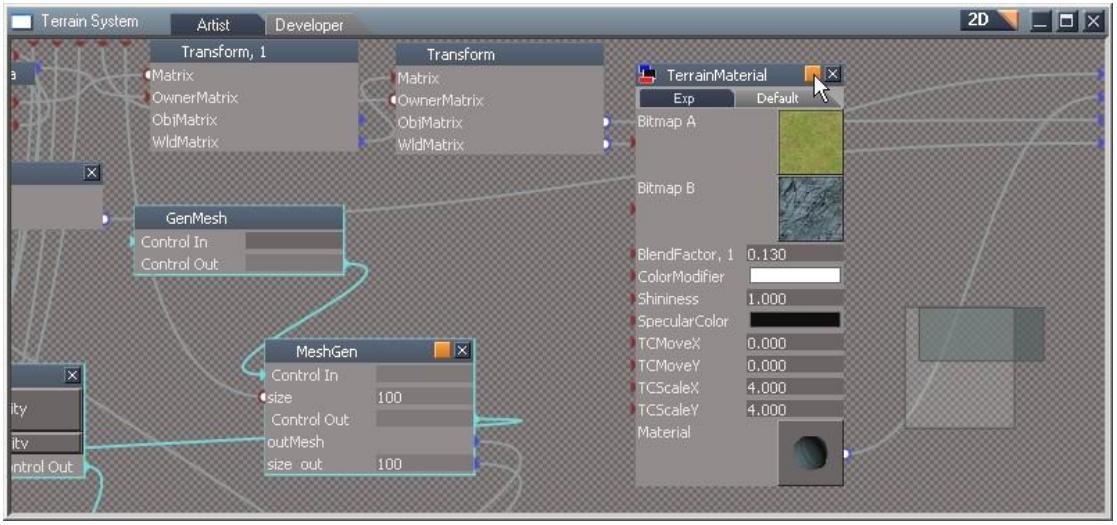


This material used in the Terrain System blends between two bitmaps based on surface normal

LandScape generator Terrain System object

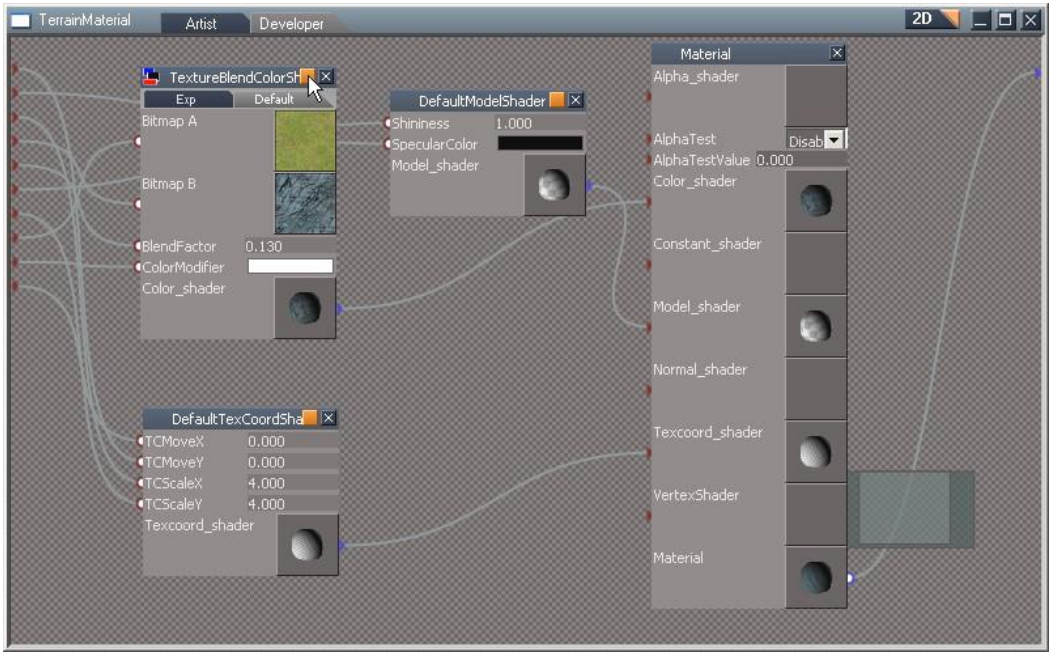
Try changing the Z Scale slider on the front panel of the Terrain System object. You should notice that, as the terrain is scaled in Z, the texture on it changes. As the terrain becomes flatter you see that more parts of it are covered with a grassy texture. As it becomes rougher the vertical areas begin to be covered with a rocky texture. This is the result of a texture blending material within the object.

Now, enter the object in the Link Editor and look around. There are a lot of objects here, but we are primarily interested in the TerrainMaterial object shown in the image below go ahead and enter the TerrainMaterial object.



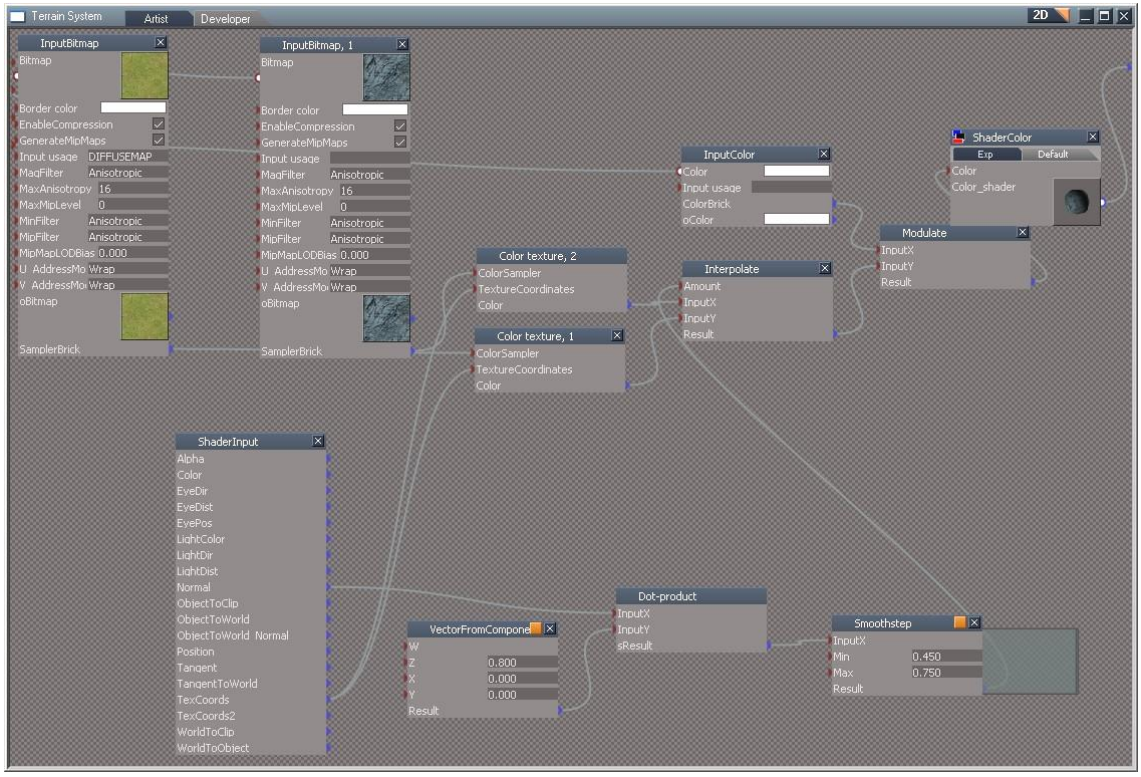
The TerrainMaterial object in the Link Editor

Inside you will find four objects: Material, DefaultTexCoordShader, DefaultModelShader, and TextureBlendColorShader. We have already explored the first three objects in depth in the previous sections, so our main area of focus is the TextureBlendColorShader, where the real work of the material is carried out. You may need to arrange the objects so you can see them all.



Inside the TerrainMaterial object in the Link Editor

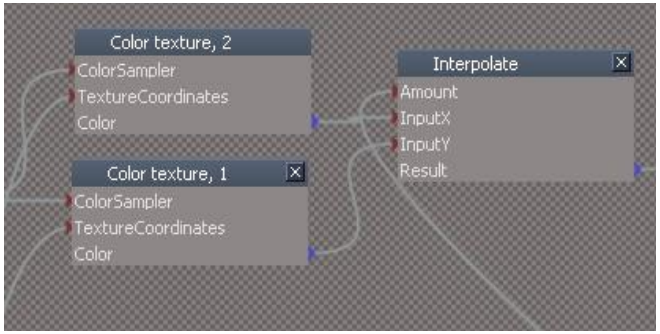
Enter TextureBlendColorShader and take a look at its contents. You should see a pretty complex network of shader objects, all feeding into a ShaderColor object at the right.



TerrainMaterial input bitmaps, samplers, blender, and color shader compiler

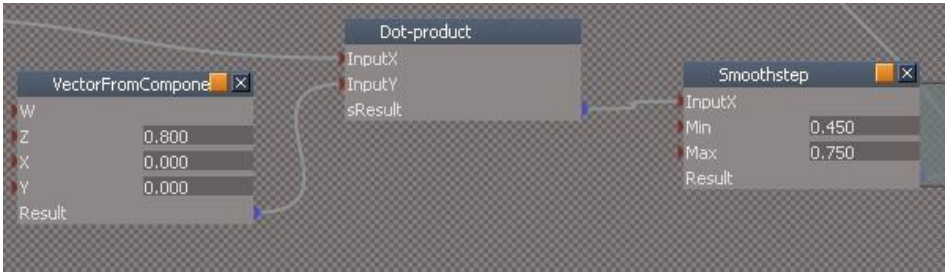
Let's take it from the top and follow the flow of data in this material. First, at the upper left of the construct you should see two InputBitmap objects. These provide the basic color information that will be used to calculate the final color for any given pixel on the material's surface. As each pixel is rendered we will grab a value from these bitmaps using Color Texture objects.
These objects take texture coordinates from the ShaderInput object and then look into the bitmap to find the color at that location.

Now the shader needs to blend between those two colors. We will do this based on surface normal (the vector perpendicular to the surface at the pixel currently being rendered) using an Interpolate object. This object blends between two values based on the input value Amount. You can see that the Color output from the two Color Texture objects feeds into the Interpolate object.



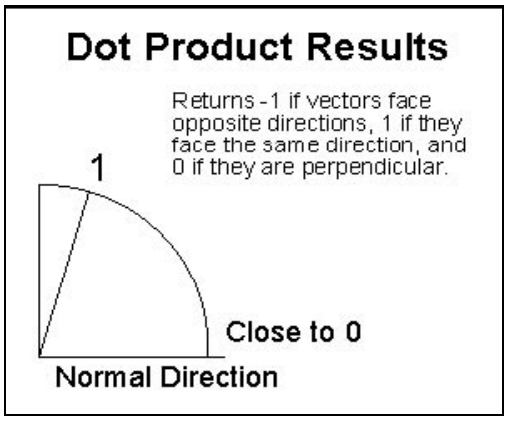
The bitmap samplers feed color values into Interpolate, which blends based on Amount

Amount comes from another structure, a calculation based on the surface normal. This structure performs a dot product operation on the surface normal (obtained from ShaderInput) and a vector (0.0, 0.0, 0.8) created by the InputFloat and VectorFromComponents objects.

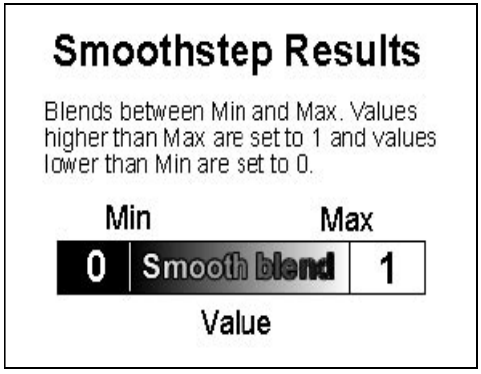


This structure calculates blend Amount based on the surface normal

Dot product returns a number from -1 to 1 by comparing two vectors (directions in 3D space). The result will be 1 if the vectors face in the same direction, 0 if they are perpendicular to each other, and -1 if they face opposite directions. In this case we are comparing the surface normal to a normal that points nearly vertically (0, 0, 0.8). Thus, if the surface normal points straight up (i.e. the terrain is "flat") the value will be close to 1. If the surface normal points horizontally (i.e. the terrain is "steep") the value will approach 0. Incidentally, we know that the value will be between 0 and 1 because the terrain mesh never faces "downward," always "upward."

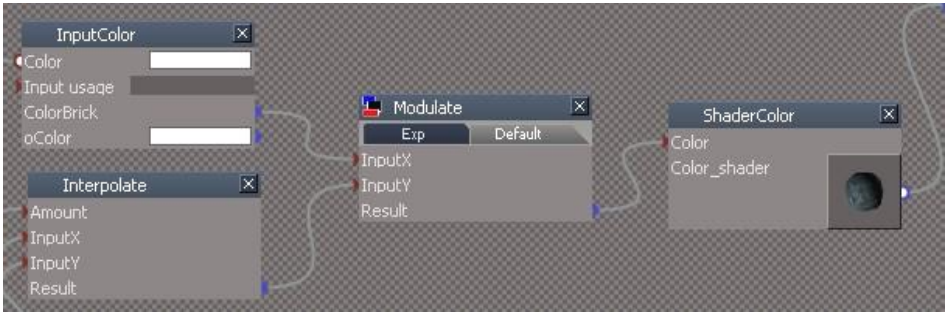


Now that we have this result, we can use it to control Amount on the texture interpolator described earlier. First though, we will perform one more calculation; this is where the Smoothstep and the other two InputFloat objects come into play. Smoothstep returns a Result of 0 if the value of InputX is less than the value of Min, and 1 if it is above Max. For InputX values between Min and Max the object will return a Result that is blended smoothly between the two.



The effect of this is that if the terrain is mainly flat, then return a value of 1. If it is mostly vertical, then return a value of 0. If it is somewhere in between Min and Max, then return a blended value. This result controls the blending between grass and rock texture by the Interpolate object, giving us our nicely blended terrain material.

Finally, the resulting blended color is mixed with white via a Modulate object (essentially a multiplication) and output to the ShaderColor object for use with our Material object.



Final texture blending and mixing with white before outputting the Color Shader

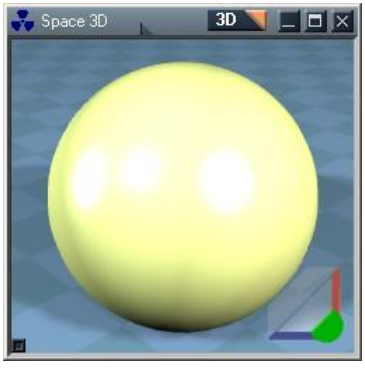
To experiment with this shader component you could try changing the values of the various InputFloat objects to see the effects. For instance, if you change the value of the floating point number feeding into Dot-product, you should see a corresponding change in the material. Try changing the values that feed into the Smoothstep object as well to see how it controls blending. You can also try using different bitmaps for some interesting effects.

As a more advanced exercise, try adding a third bitmap and blending it into the mix. You could try adding a darker green foliage texture that only covers those parts of the terrain that are almost completely flat, to simulate areas where water gathers and plants grow more thickly.

5.4.3 Sample DX9 Materials

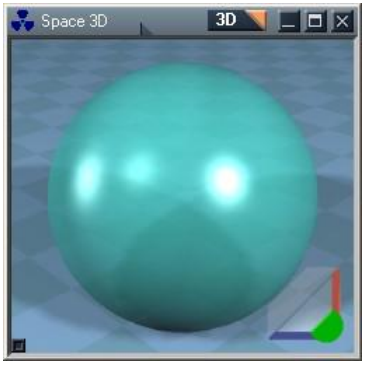
trueSpace includes a selection of DirectX materials in Materials library, from simple to complex, that can be used in your own trueSpace scenes and which may serve as a foundation for your own advanced DirectX materials.

Solid



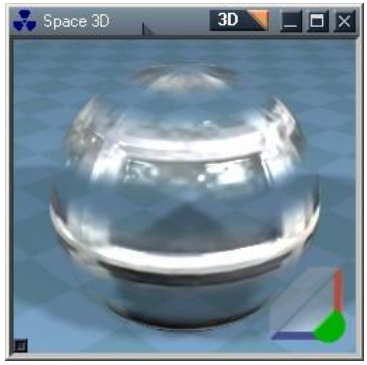
This is the simplest DirectX material, featuring only a few attributes that you can change: DiffuseColor, DiffuseStrength (specifies the amount of diffuse reflection), Shininess and SpecularStrength (modify the specular reflections),VertexColorStrength (affects vertex color if specified in the mesh).

SolidTransparency



This material is similar to Solid, described previously, but adds a Transparency attribute that controls how transparent the material appears.

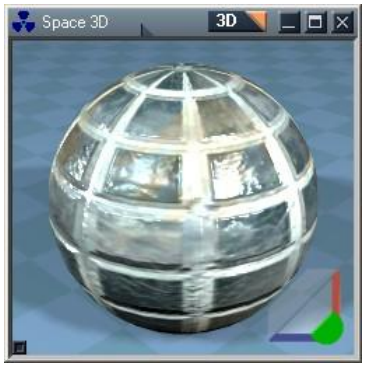
TextureBumpAlpha



This material combines a texture map with a normal map and a transparency value. Under the Simple category of the Materials library there are other materials that offer various combinations of texture, bump, and transparency maps. Their functioning is similar to this material.

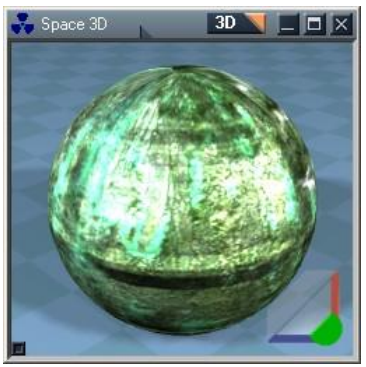
Each map type (diffuse, normal, and alpha) can be offset and scaled by changing the TexCoordMove and TexCoordScale attributes.

LayeredPlastic



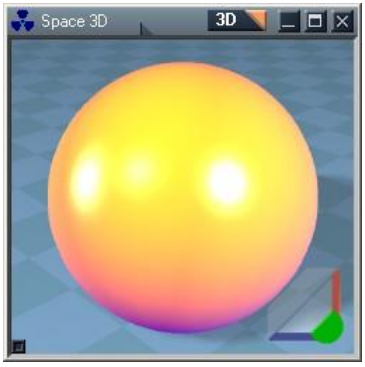
This material blends between two textures based on the BlendFactor attribute.

FastCookTorranceMaterial



This material can be used to simulate extremely shiny surfaces such as metals. The Reflectance attribute specifies how much light penetrates the object, setting the balance between the amount of reflection and refraction. Lower values mean less reflected light. Altering the Roughness attribute changes how much light is reflected towards the camera.

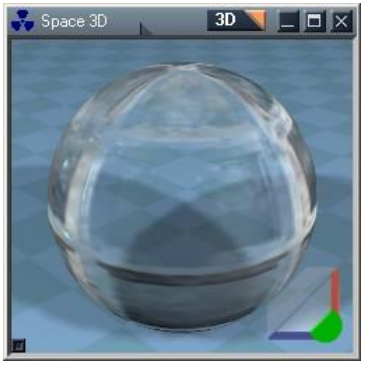
FeederGPU_Gooch



Gooch is a non-photorealistic material that emphasizes the geometric properties of an object by modifying the surface color drastically based on the direction the surface faces.

The *DiffuseColor* attribute specifies the base surface color. *CoolColor* specifies the color seen on the borders of the object. The *WarmColor* color parameter works in a similar fashion. The main difference is that *warm* color affects the surface in areas that have a low angle between the viewer and the surface normal. The Gooch material works best with single light source.
The Feed amount works on altering the meshes vertices to grow and shrink the mesh

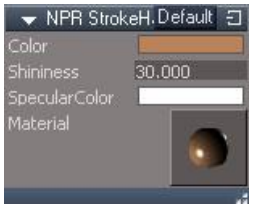
Alpha Bacteria

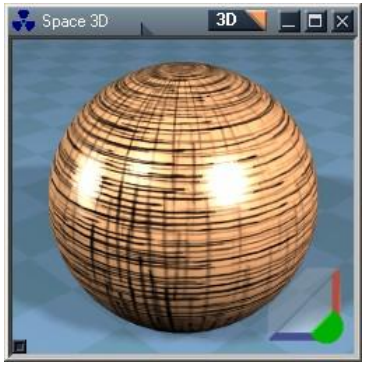


This material is different in that the transparency is computed from the geometry of the object itself. The amount of the transparency depends on the angle of the surface and the direction from the camera to the surface at that point. Shallow angles produce high opacity.

Alpha strength specifies the degree of transparency. Greater values increase the opacity of the material at the boundaries of an object and reduce the opacity “in the middle” of the object.

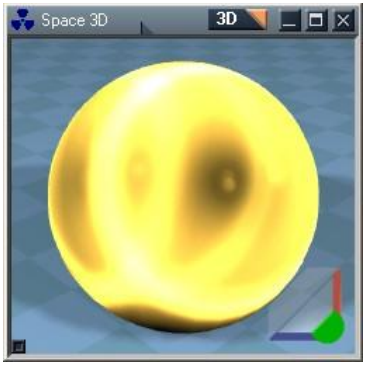
NPR StrokeHatch



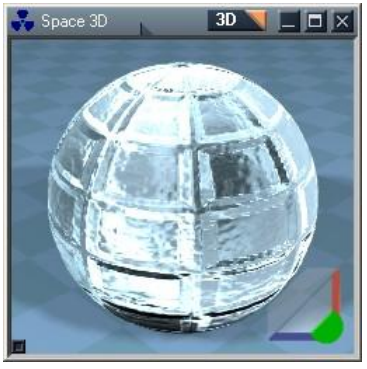


This non-photorealistic material simulates a stroked drawing style. It works best with a single light source.

Anisotropic and AnisotropicMap

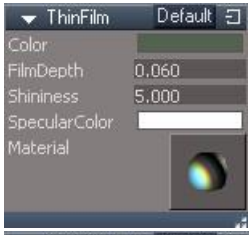


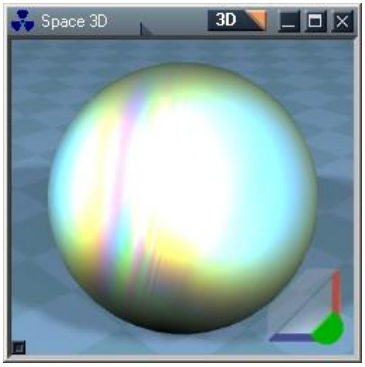
These materials simulate advanced anisotropic lighting.



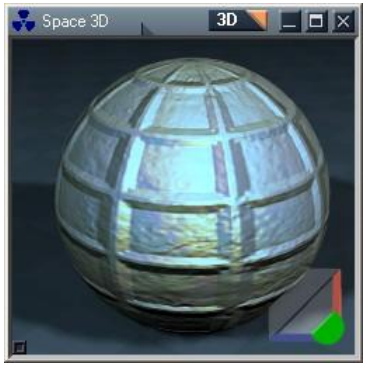
The AnisotropicMap material uses a normal map to simulate bumpiness. Both of these materials work best with a single light source.

ThinFilm and ThinFilmMap





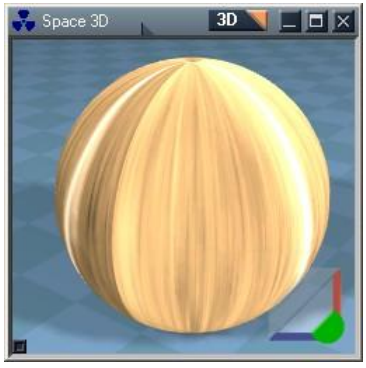
This material simulates a thin, oily film on the surface of an object.



There is also a ThinFilmMap material uses a normal map to simulate bumpiness.

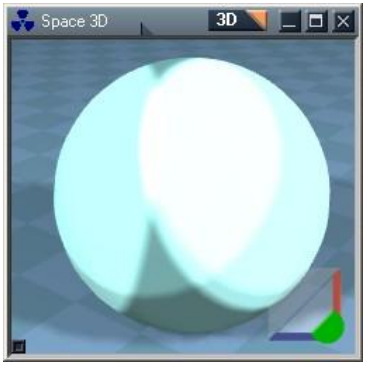
Both materials work best with a single light source.

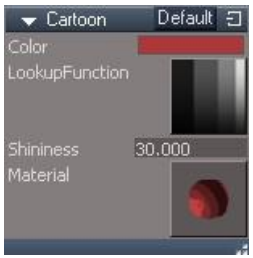
Hair

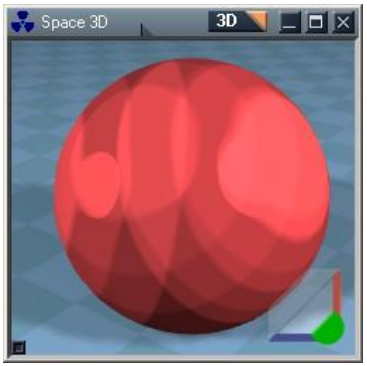


Hair attempts to simply simulate the properties of human hair. The Tint attribute specifies the hair color, while the Bitmap attribute specifies the hair texture. The Shininess attribute alters the glossiness of the hair. Useful values for Shininess are between 0 and 600.

CartoonMetal and Cartoon







These materials simulate cartoon-like rendering. Both materials work best with a single light source. The limited tonal range of these materials is driven by the LookupFunction attribute (a one-dimensional bitmap with stepped greyscale values from light to dark). If you want to alter the appearance of these materials you should replace this bitmap with one of your own .

Constant Shader Materials

ConstantColorMaterial

this material uses ConstantColor shader and completely disables any light contribution. The Final result is a single colored object.

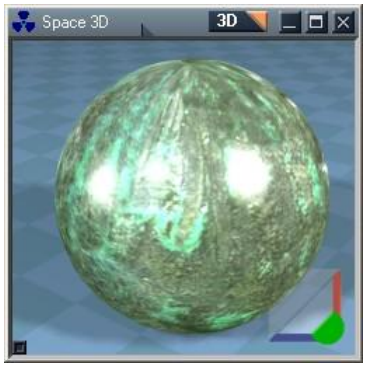
Use this if you do not want the appearance of your material to change when additional lights are placed to the scene.

AmbientPhongMaterial

this material uses ConstantAmbient shader together with color texture and Phong lighting model.

Usage of ConstantAmbient shader allows the object to show color texture (multiplied by ambient color) in dark areas instead of becoming black

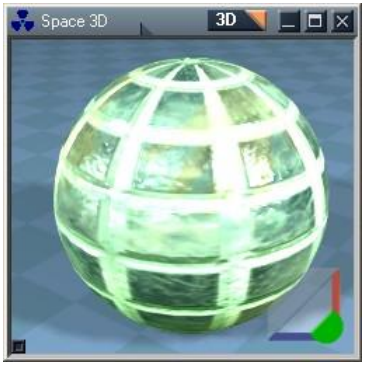
ConstantTexturePhongMaterial



this material uses ConstantTexture shader together with color texture and Phong lighting model.

Usage of ConstantTexture shader allows the object to display separate constant texture rendered in dark areas instead of becoming black.

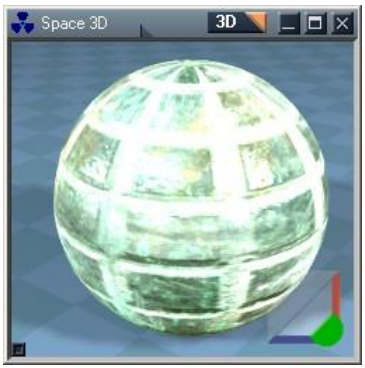
AmbientLayeredPlastic



this material is an extension of the LayeredPlastic material and it has a ConstantAmbient shader.

Usage is shown in the examples which follow later in this section.

ConstantTextureLayeredPlastic

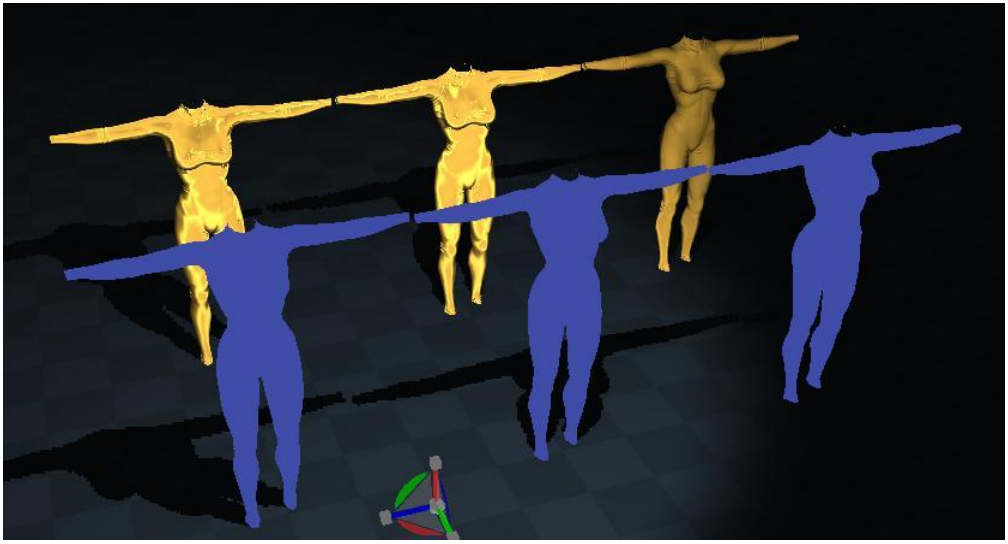


this material is extension of the LayeredPlastic material with the ConstantTexture shader.

Usage is shown in the examples which follow later in this section

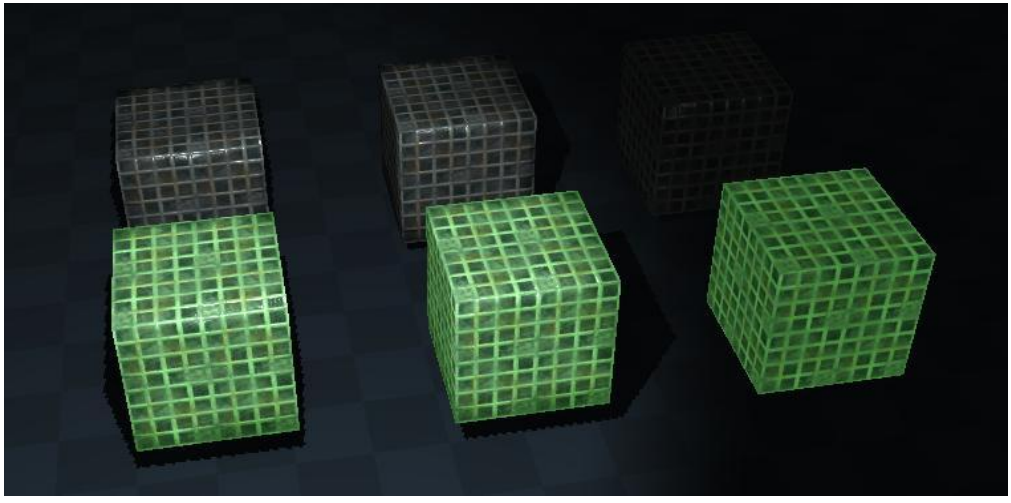
Examples

ConstantColorMaterial applied to the object in the scene.
Moving the object to the dark area does not affect resulting constant color.



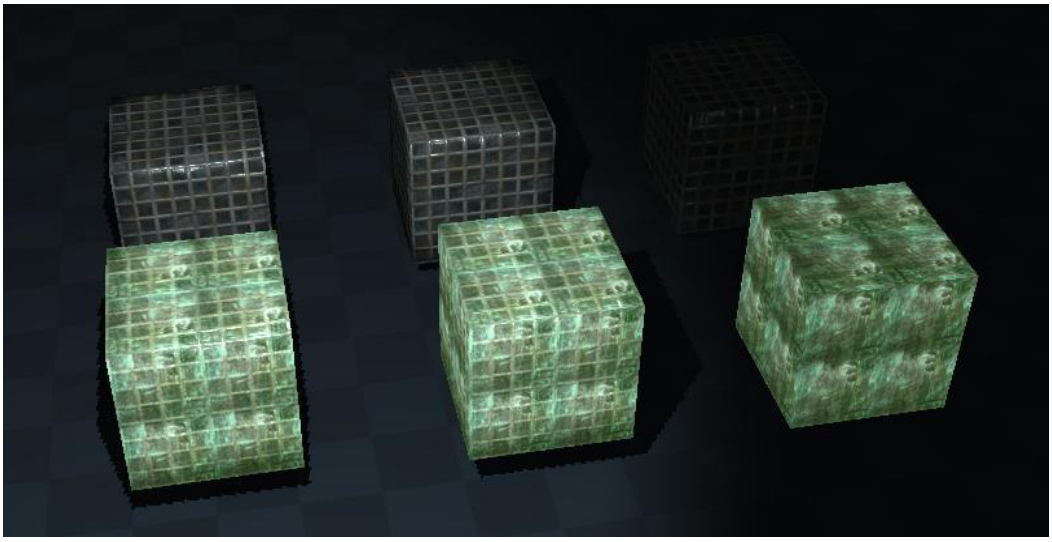
ConstantColorMaterial applied to the object in the scene

Apply AmbientLayeredPlastic to the object in the scene.



AmbientLayeredPlastic applied to the object in the scene

You can see that while the object has correct lighting in the lit area, it shows only diffuse texture as soon as it is in the dark part of the screen. You can notice the loss of lighting to the right while texture is still visible.
Apply ConstantTexturePhongMaterial to the object in the scene..



ConstantTexturePhongMaterial applied to the object in the scene

Object shows regular lighting when affected by light but as soon as it is moved out of the light it shows custom constant texture. Next picture shows this material applied. You can see how constant texture replaces regular texture when there is no lighting on the right side of the image.

Alpha testing

Alpha testing support for materials and shadows. Alpha testing is method of disabling rendering of pixels whose transparency is below a given threshold. An advantage of this method is that it can be used with depth testing and therefore it produces correct sorting.

It allows objects such as vegetation (leafs, grass) to display properly sorted and cast proper shadows even if they are only a single quad with transparent texture. It is also an optimization technique as it allows hardware to skip blending operations on pixels that fail alpha test (and therefore become invisible).

material object

Each material object has attributes for Alpha test and Alpha test value.

Alpha test has three modes – Disabled, Enabled and Enabled as opaque.

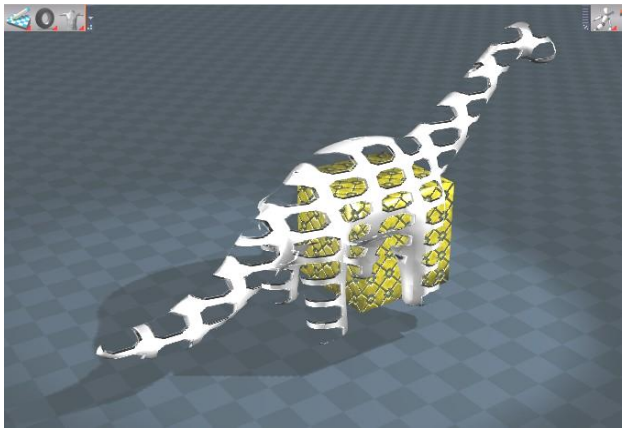
- Mode "Enabled" activates alpha testing for the surface, but it will still be treated as transparent – therefore you get shadows with structure (i.e. holes) and you can increase the speed by avoiding unnecessary blending operations, but you can still have problem with transparency sorting and with multiple lights (see second picture).

- Mode "Enabled as opaque" will work similarly as above, but will treat object as opaque and therefore you get correct lighting and sorting (see third picture).

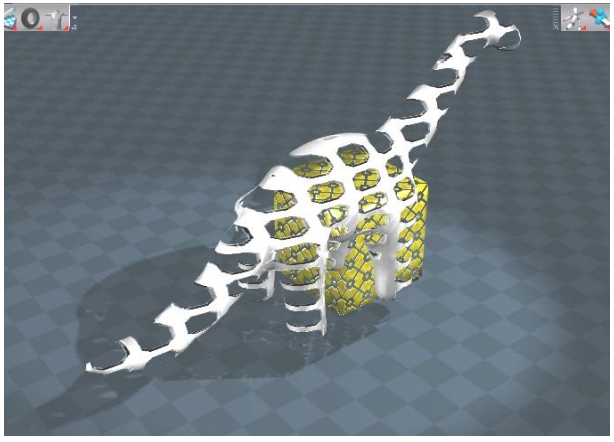
- Disabled mode disables alpha testing for material.

- Alpha test value – specifies threshold value against which alpha of pixel is tested. If pixel's alpha value is lower than threshold it is not displayed. This also means that threshold 0 will display all pixels and threshold 1 will hide entire object (although this is much slower than checking "Visible" flag of render attributes).

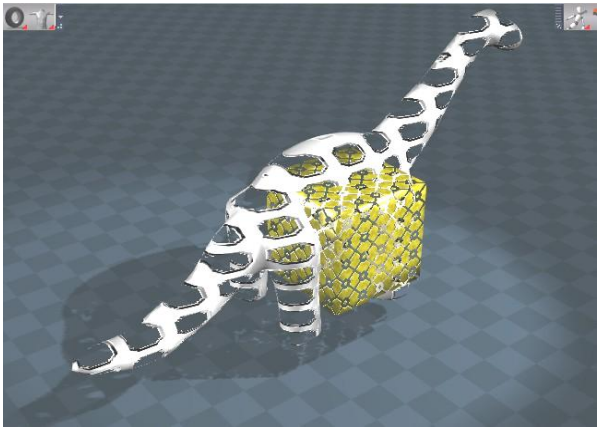
Examples



Two overlapping transparent objects (see incorrect shadows and sorting)



Alpha test enabled – shadows are correct, but the objects still overlap improperly.



Opaque mode alpha test – correct intersection and shadows.

5.4.4 Special Material objects

Render to texture

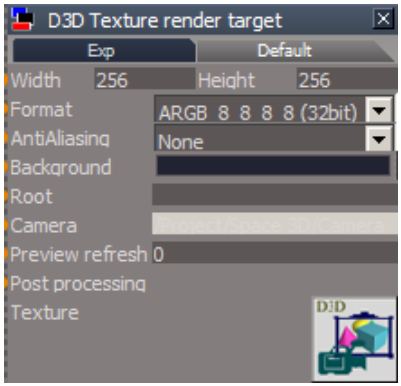
The Render to texture objects will allow you to create various real-time effects by rendering the scene into a texture and then reuse this texture (with real-time update) in a material.

Some effects that could be achieved include simple usage such as a display or beamer simulation and also advanced usages for example flat mirror, reflection or refraction.

Render to texture objects Library

The render-to-texture objects are stored in the “Render to texture” library. These objects are:

Render to texture object

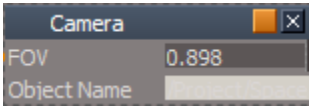


Render to texture object.

This object performs primary render to texture actions. It has following inputs:

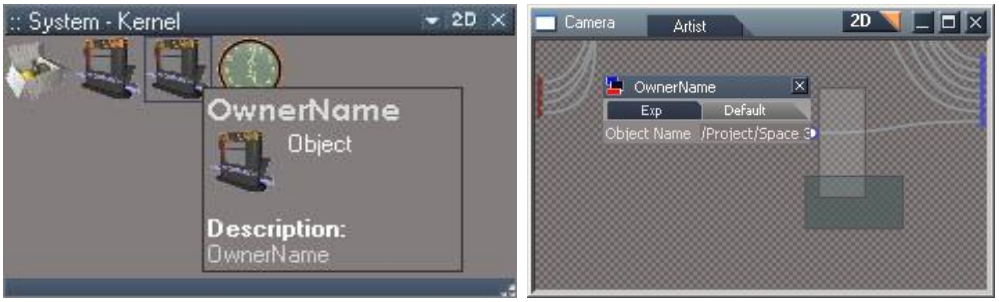
- **Width and height** – dimensions of the final texture.
- **Format** – the final format of texture. List contains all available formats. **Please note that depending on capabilities of your graphics card, some formats might not include blending and filtering operations and this might result in incorrect multipass rendering (i.e. only single light will be visible).**
- **AntiAliasing** – the anti-aliasing quality of rendering into the texture. The list includes maximum quality for all available formats and depending on capabilities of your graphic card and available memory only some of the quality settings will work for specific format.
- **Background** specifies the background color of the texture.
- **Root** – the name of encapsulator which will be used to enumerate objects for rendering (i.e. "/Project/Space 3D"). Empty value will use encapsulator which contains render to texture object.
- **Camera** – the name of used camera (i.e. "/Project/Space 3D/Camera1").
- **Post processing** – the post processing settings.
- **Preview refresh** – refresh of preview bitmap. 0 means no preview, non zero value specifies minimum refresh time in milliseconds (1000 ms = 1s). *Please note that preview refresh is very, very slow operation and you should enable it only in following situations: you require texture to be available for offline rendering (V-Ray) or Modeler or for some other non-D3D usage. Or you require preview feedback in the panel because you cannot use real-time feedback in D3D window.*
- **Texture** – output texture. The output texture can be connected to any attribute accepting a Bitmap object (i.e. materials).

Camera



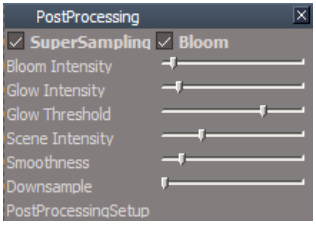
Camera

The Camera in this library is a regular camera object. But It also contains an additional output connector called OwnerName which has been added from the System – Kernel library with the objects name exported so it can be easily connected to the "Camera" attribute of Render to texture object.



Right OwnerName in the System – Kernel Library
Camera OwnerName object with exported connector shown on the Left

Post processing



post processing settings object

This object is a regular post processing settings object (the same as the one used for the workspace windows). You can connect output of this object to "Post processing" connector of render to texture object.

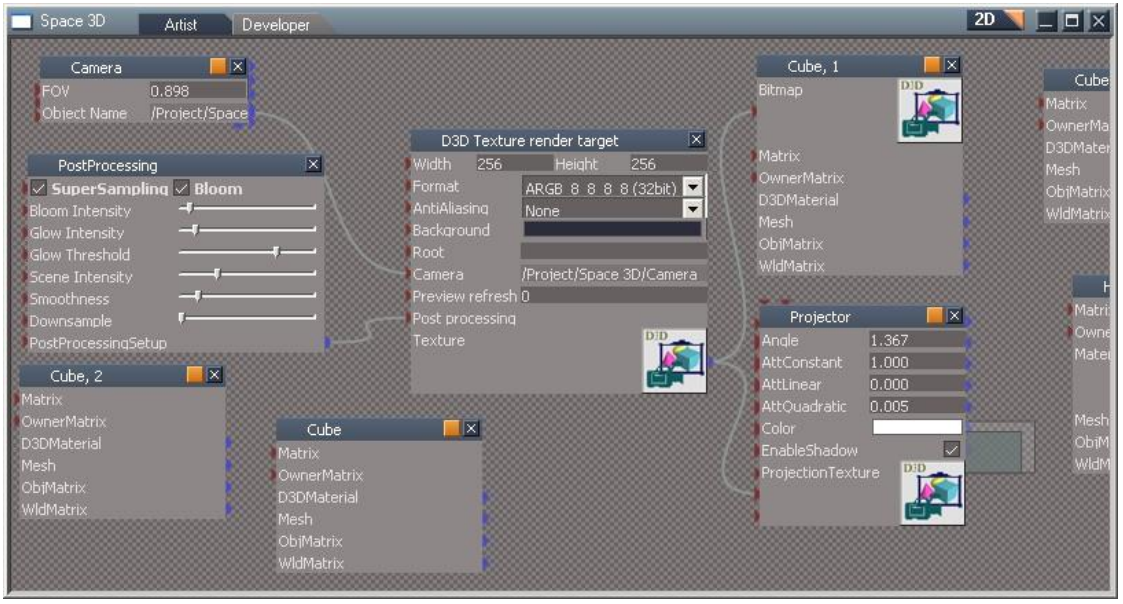
Example

Render to texture library contains sample scene "Render to texture sample". Loading the scene will show two usages of rendering to texture as well as showing the connection setup of the render to texture objects in the LE.

Scene shows the head object captured by camera. The Panel behind the head shows real-time image of the head as captured by the camera. Post processing is enabled with “supersampling” and “bloom”. Next to the panel there is projector (beamer) projecting the texture onto a wall (which casts correct shadows).
If you rotate the head, both panel and beamer update what they show. Also depending on capabilities of your GPU (graphics card) you can notice that because panel is partially visible to camera, it will display an image of itself (as captured by camera).

Render to texture example scene.

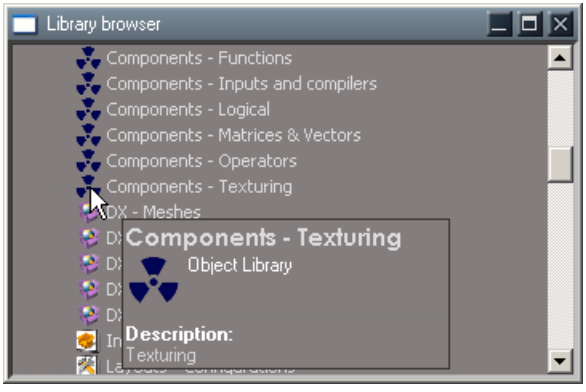
Below is an image of the Link editor which shows how to connect the Camera and Post processing object to the Render to texture object and also how to connect render to texture object to a material or projector light.



The objects linked together in the Link Editor

5.4.5 Tutorial: Constant Alpha Texture

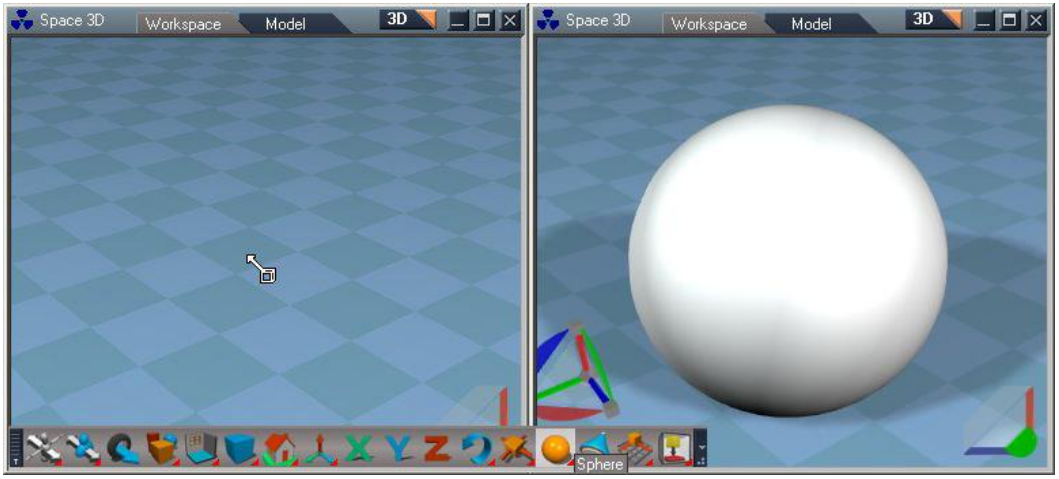
In this basic tutorial you will be creating and adding an alpha texture component to an existing shader.
When you create advanced materials in trueSpace, you will work primarily within the **Link Editor** with a selection of libraries and components devoted specifically to material creation.
A **material object** is visually similar to any other object that you might encounter in the Link Editor view, with the exception that it exports a Material attribute. The Material attribute tells trueSpace how to render objects that are painted with those material components.



The Components – Libraries

Examining a DirectX Material

To look at the internal details of a material, add a Sphere object to the Workspace from the toolbar by selecting the Sphere from the primitive shapes and clicking once in the Workspace. A right click will exit the tool once you have placed the sphere where you want it.

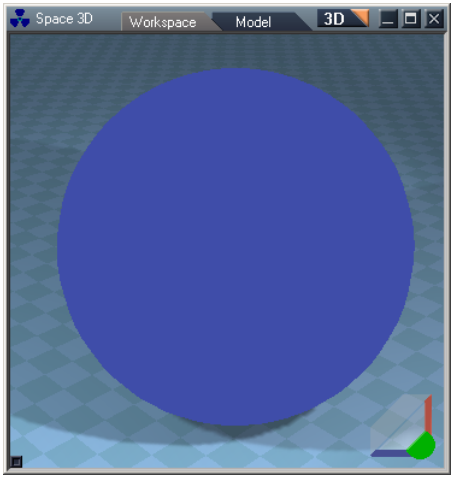


Add a sphere to the Workspace to apply the material to and give some visual reference

Locate the ConstantColorMaterial in the Materials DX-9 library and drag it onto the sphere in the Workspace view. This shader is useful for objects that you do not want to be affected by additional lights.

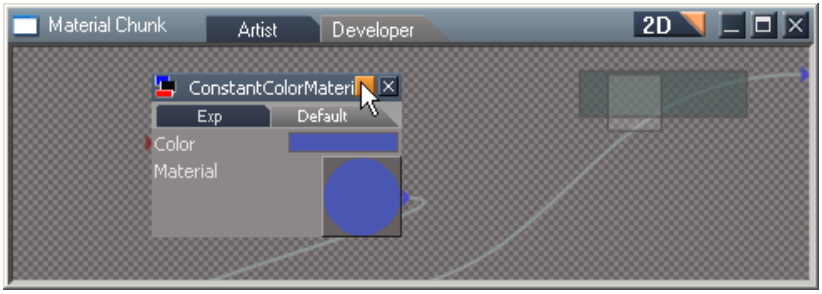
The ConstantColorMaterial

Your sphere should now be colored with a blue material, as shown in the image below.



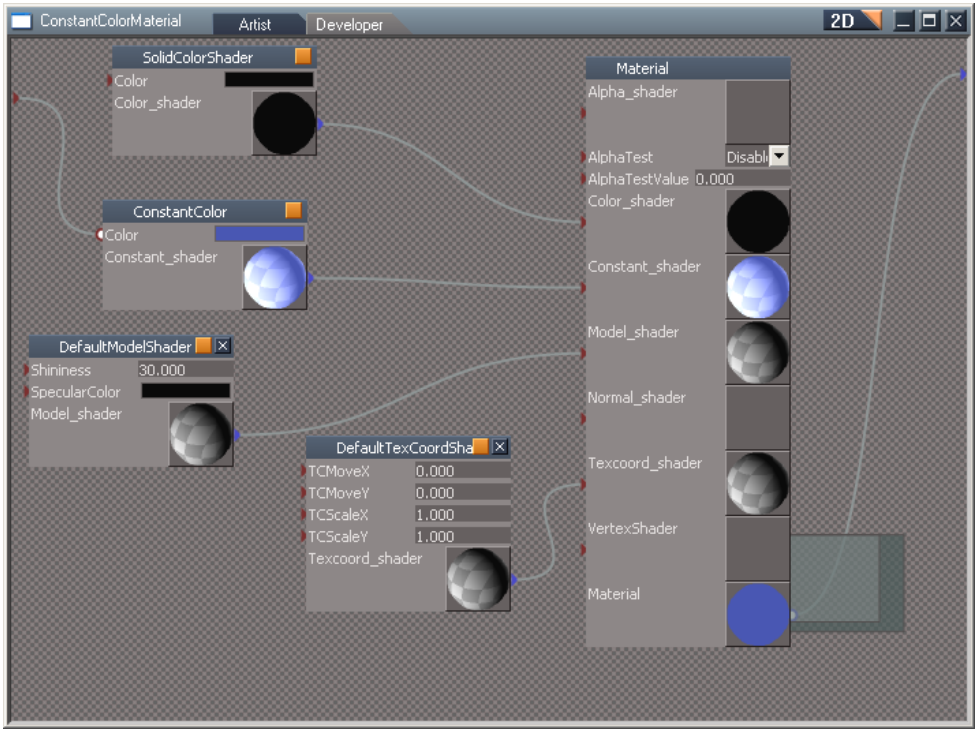
The ConstantColorMaterial material applied to a sphere in the Workspace view

Enter the Sphere object by clicking the orange enter icon, locate and enter the Material List , then locate and enter the Material Chunk, once inside you should see a material object called ConstantColorMaterial, as shown in the image below. This is the material you just applied.
To enter the material click the orange enter icon on the material object's front panel.



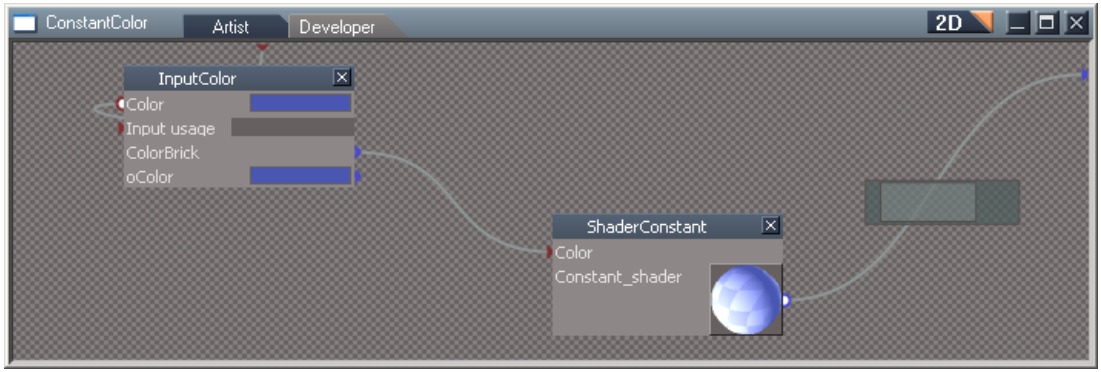
The ConstantColorMaterial object in the Link Editor

Once inside the Material you will see the parts that work together to create the material's effects, as shown in the image below. This material contains a Material object and several shader components that feed into it:
The individual components are the SolidColorShader, ConstantColor, DefaultTexCoordShader, and a DefaultModel model shader.
If you cannot see the links like in the image below then change the tab in the Link Editor window to Developer aspect.



Inside ConstantColorMaterial we find the shader components that make up the shader

In the Link Editor, you can enter any object that displays an orange enter icon in the upper right-hand corner of its panel. For instance, click on the ConstantColor shader component to see its contents.
You should see something like the image below.



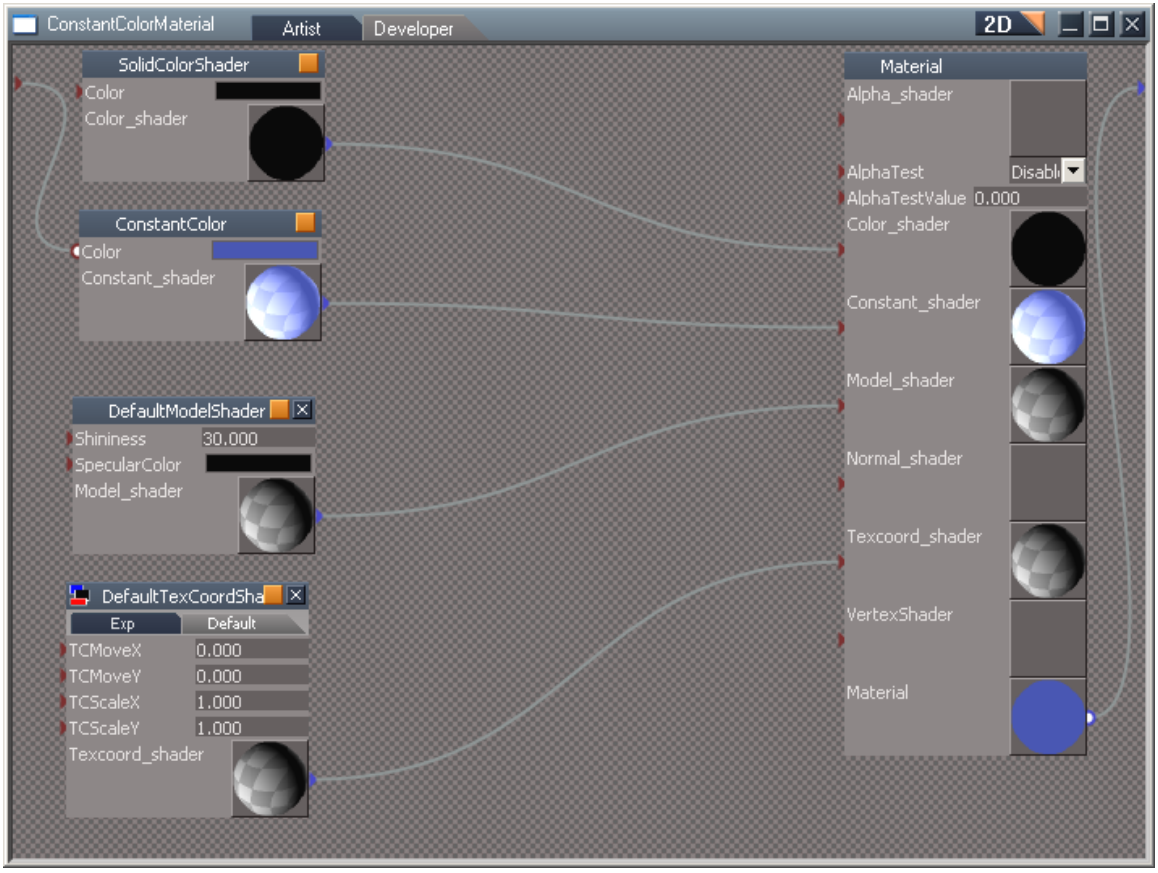
Another level of detail resides inside the ConstantColor shader component

The parts of this shader component are the InputColor object, which takes simple color information to determine the final output, and the ShaderConstant object, which exports that result to the outside for use when its linked to the

Material object .

Modifying a DirectX Material

You can, of course, modify any of the connections or objects you find in a material using the Link Editor. For instance, as we are doing here to create and add an alpha component.
Exit the ConstantColor object one level by clicking the orange triangle near to the 2D in the objects title bar so that you can see the Material object.

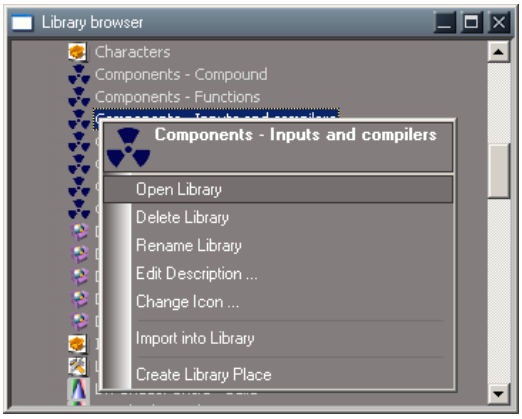


Material Object on the left with material shader components linked to the corresponding inputs.

Open the library Browser by pressing the icon in the toolbar.

Library Browser tool Icon

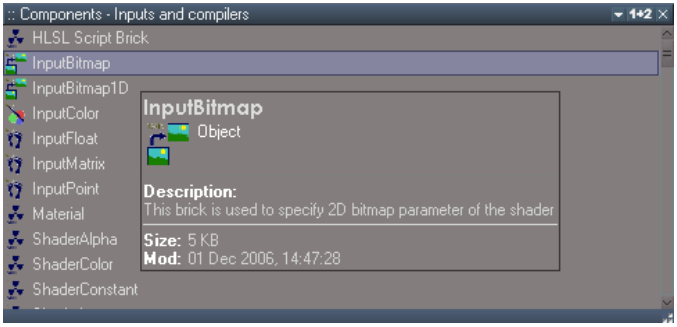
From the Library browser window open up the **Components – Inputs and Compilers** and the **Components – Texturing** libraries by either a double click or right click and choose open from the menu, this will open the libraries in the stack.



R-Click over an item to show the Library Browser menu

Inputs and Compilers , and Texturing – Libraries

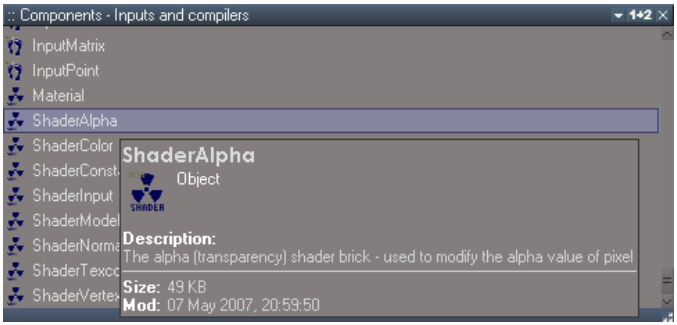
From the Inputs and Compilers library Drag into the LE the following four component objects.
InputBitmap, which allows the shader to use a texture as part of the processing.



InputBitmap

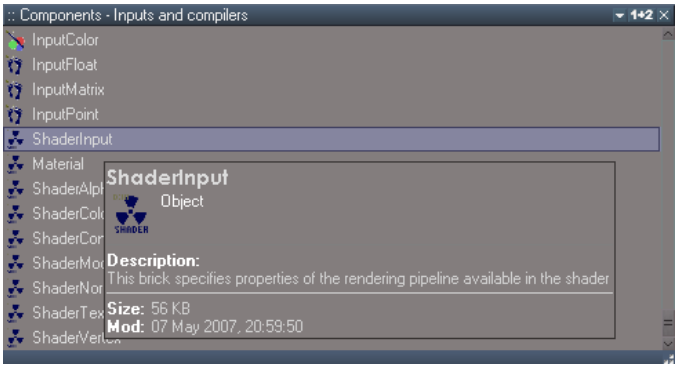
ShaderAlpha , which takes the results of all the processed bricks and prepares them for use with the Alpha part of

the material object.



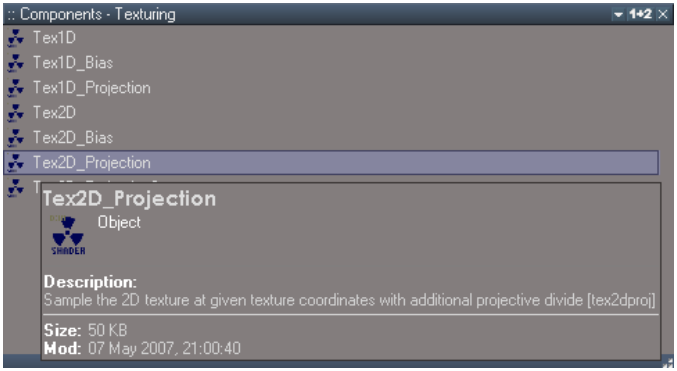
ShaderAlpha

ShaderInput , which allows for other material system properties to be processed and included in the final calculation , in this particular shader we just need to tell it to use the texture coordinates.



ShaderInput

From the Texturing library drag in a **Tex2D_Projection brick** which allows for the texture along with the texture coordinates to be linked into and processed by the material system.

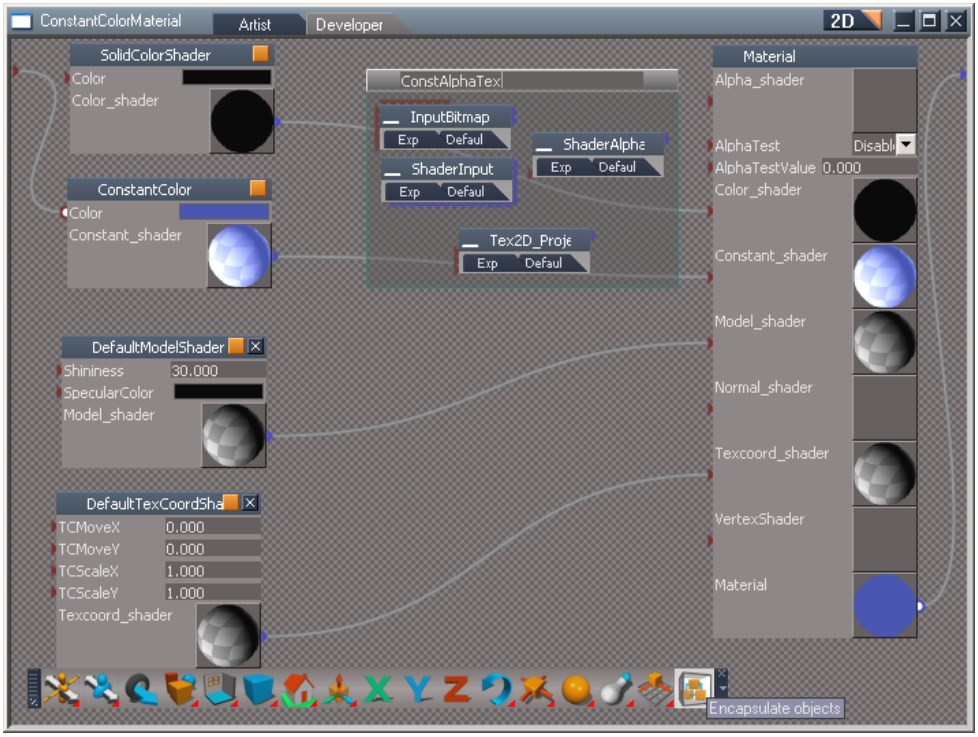


Tex2D_Projection

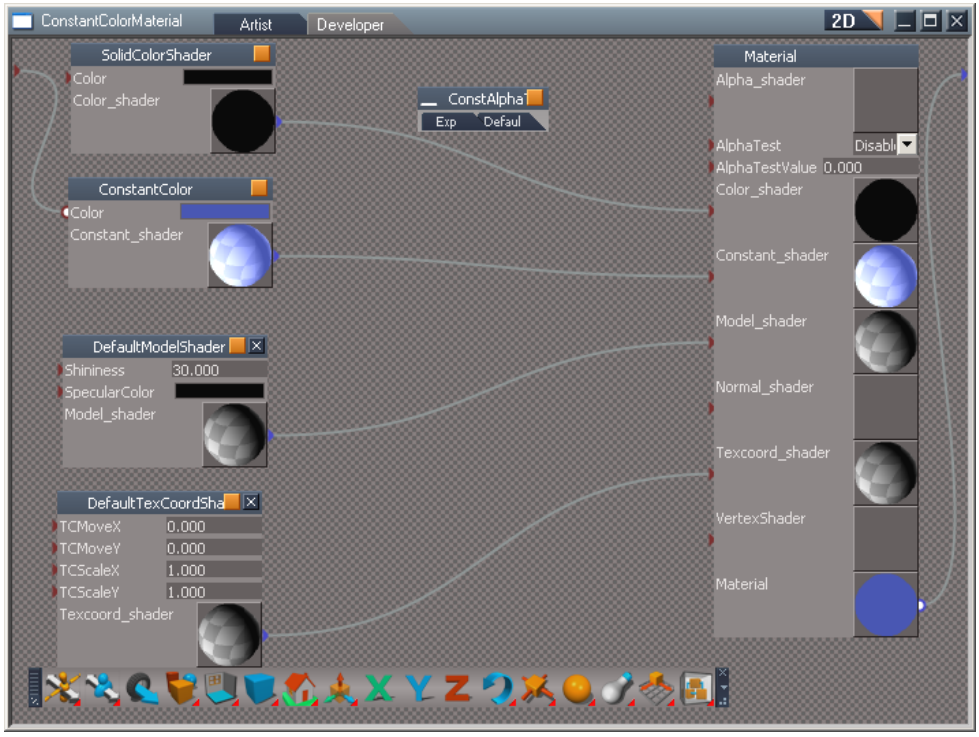
These are the four objects you'll use to create the alpha texture part of the shader.
It doesn't matter too much where you put these in the LE so long as they are near each other as you are going to encapsulate them into one object and then enter it and also arrange the objects inside to work with.

You can encapsulate objects by holding the left mouse down and dragging to draw a box around them in the LE, doing this selects all the objects it passes over and when they are selected click on the encapsulate icon in the main toolbar. There is no need to use encapsulate in 3D as the material does not have a mesh so a regular encapsulate will work ok. When the dialogue appears type in a sensible name which you understand and also describes your object, I shortened mine to ConstAlphaTex. Press enter on your keyboard to accept the name and the selected objects will become a single encapsulated object.

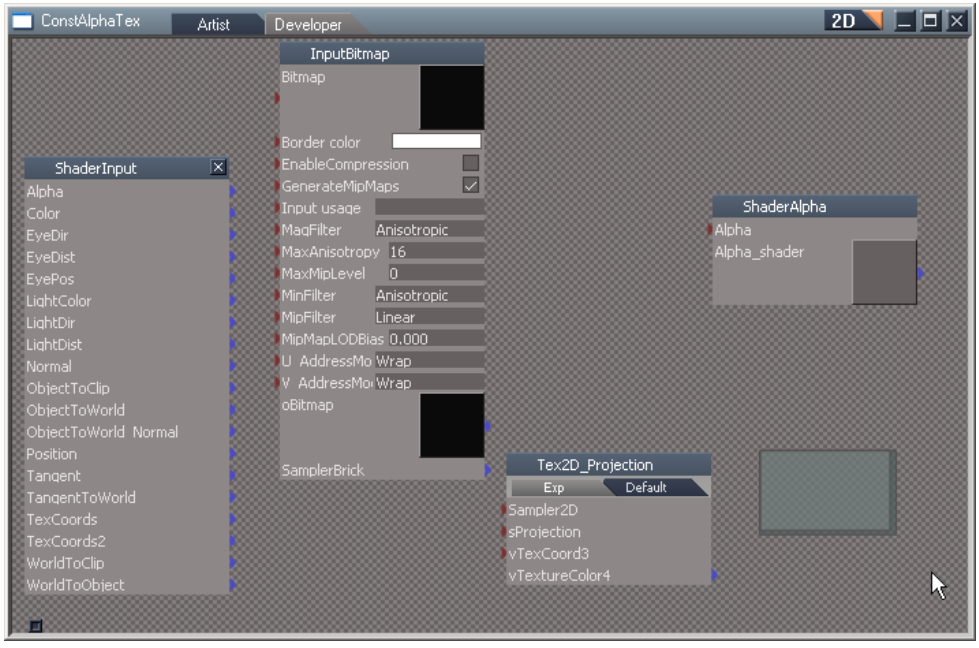
Enter the new object and tidy up the appearance so you can edit and work in a less cluttered way
When you are done you will export only the attributes you need to use or want to allow to be changed to the outside levels of the object.



Rectangle selecting then Encapsulating and Naming the shader components

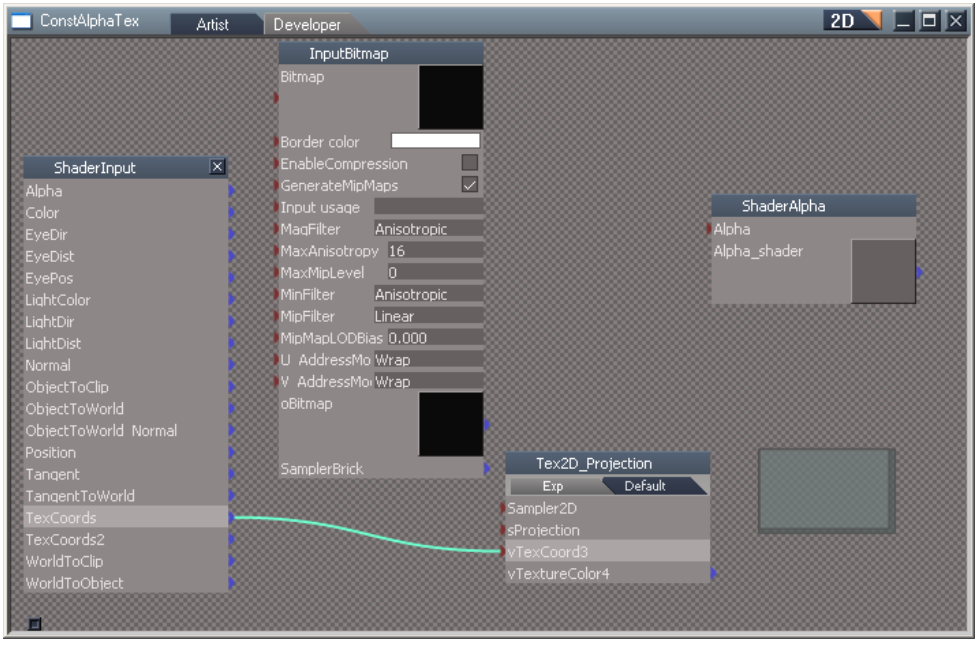


New Encapsulated object ConstAlphaTex



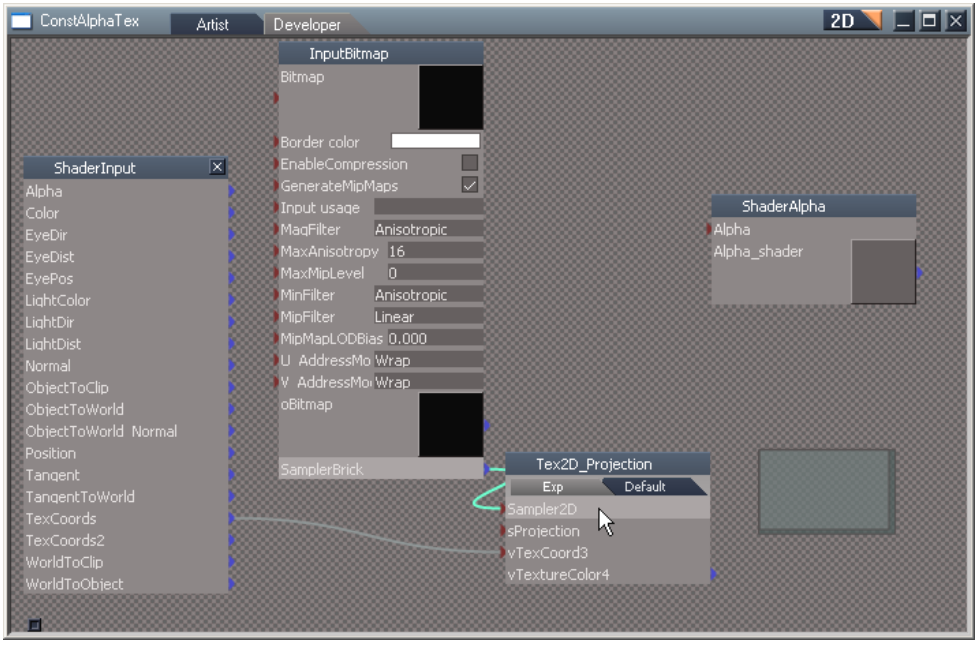
Arrange the objects inside to give a neater appearance

Now you need to drag a few links to connect the objects together to process the information we feed it and this in turn will be exported out and then passed onto the material processing brick
To start drag a link from the ShaderInput bricks Texcoords to the vTexCoord3 in the Tex2D_Projection brick.



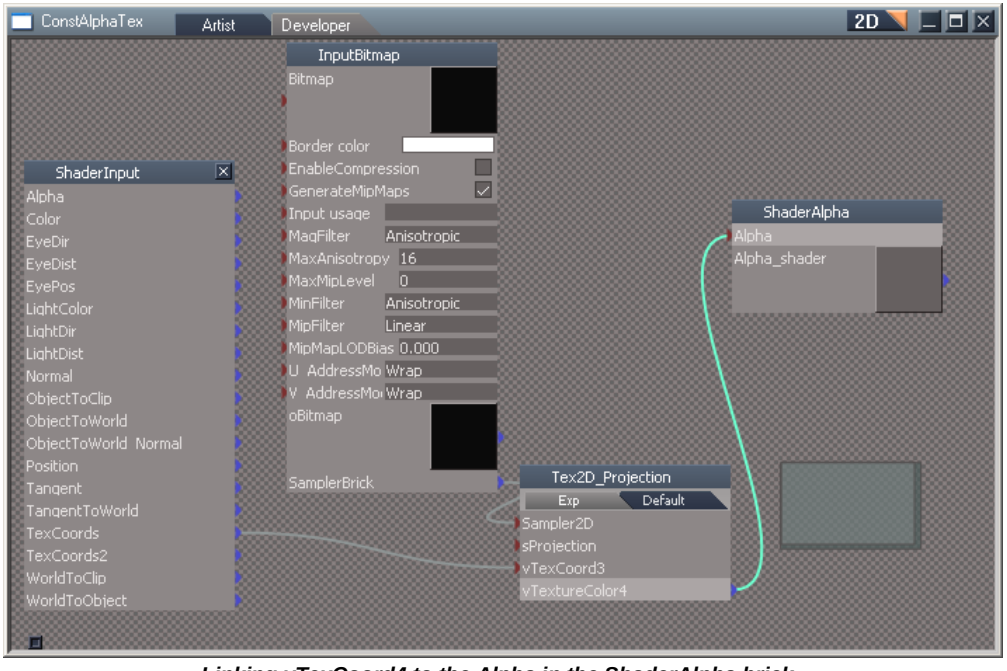
Linking TexCoords to the vTexCoord3 in the Tex2D_Projection brick .

Then drag a link from the SamplerBrick to the Sampler2D in the tex2D_Projection brick.



Linking SamplerBrick to the Sampler2D in the tex2D_Projection brick .

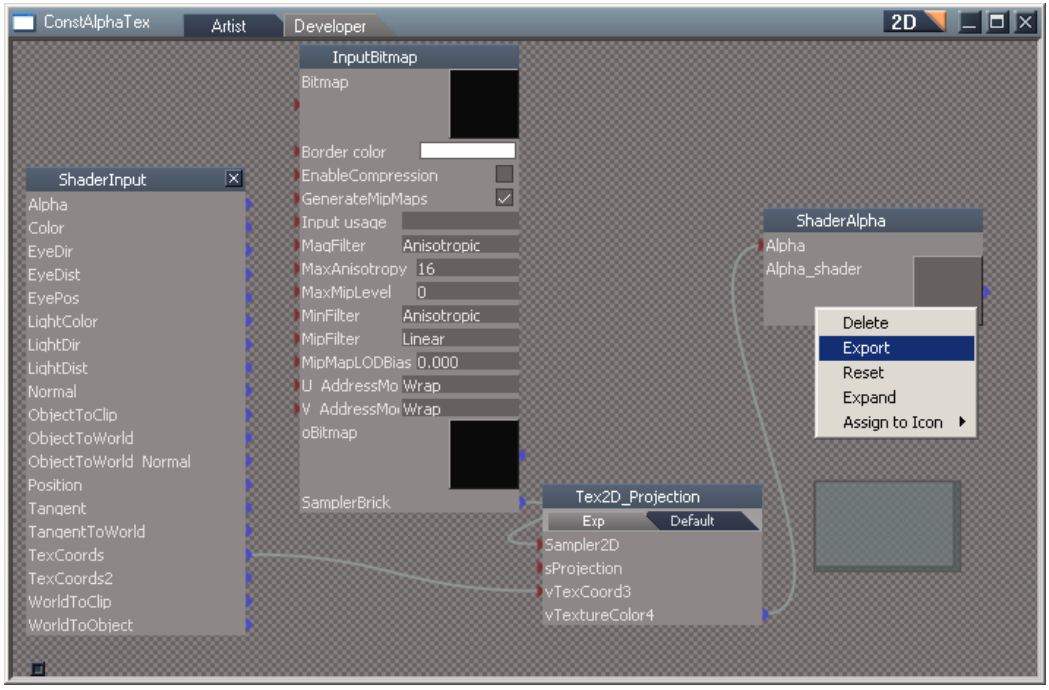
Next drag a link from the vTextureColor4 to the Alpha in the ShaderAlpha brick.



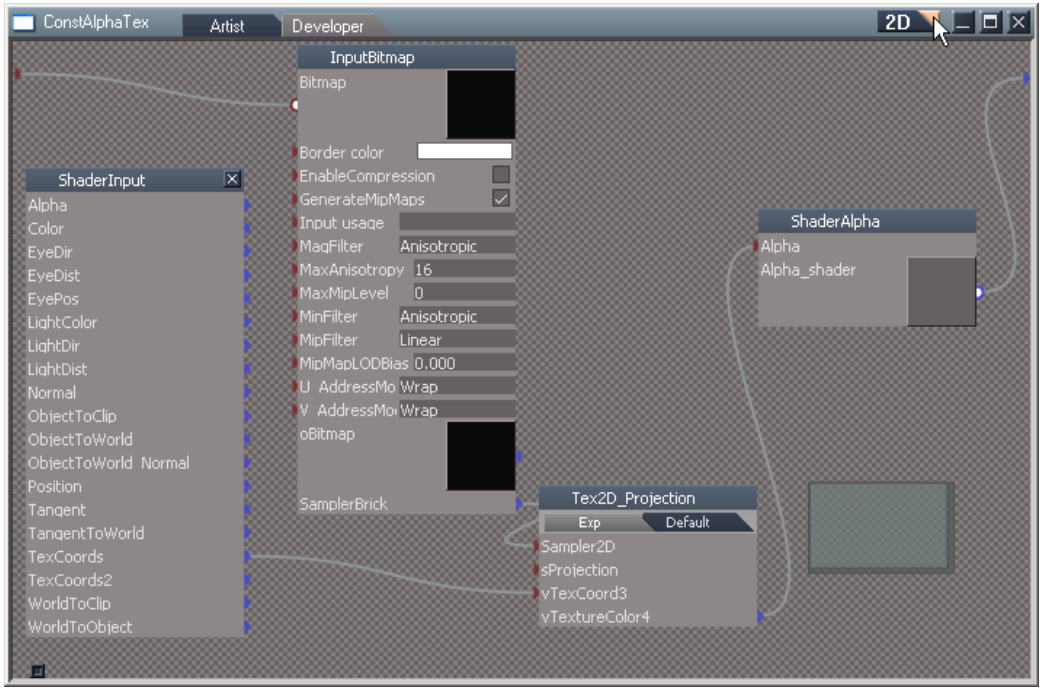
Linking vTexCoord4 to the Alpha in the ShaderAlpha brick .

Export the attributes we need by right-clicking over the Alpha_Shader and choose export from the popup menu, export the Bitmap in the InputBitmap object as well.

You'll notice that two wires are made that connect to the outer wall of our object and they provide a mechanism to send or allow data to be passed and accessed from a higher level of an object, notice as well that the triangles on the objects become closed with a cap and turn white when they have been exported.

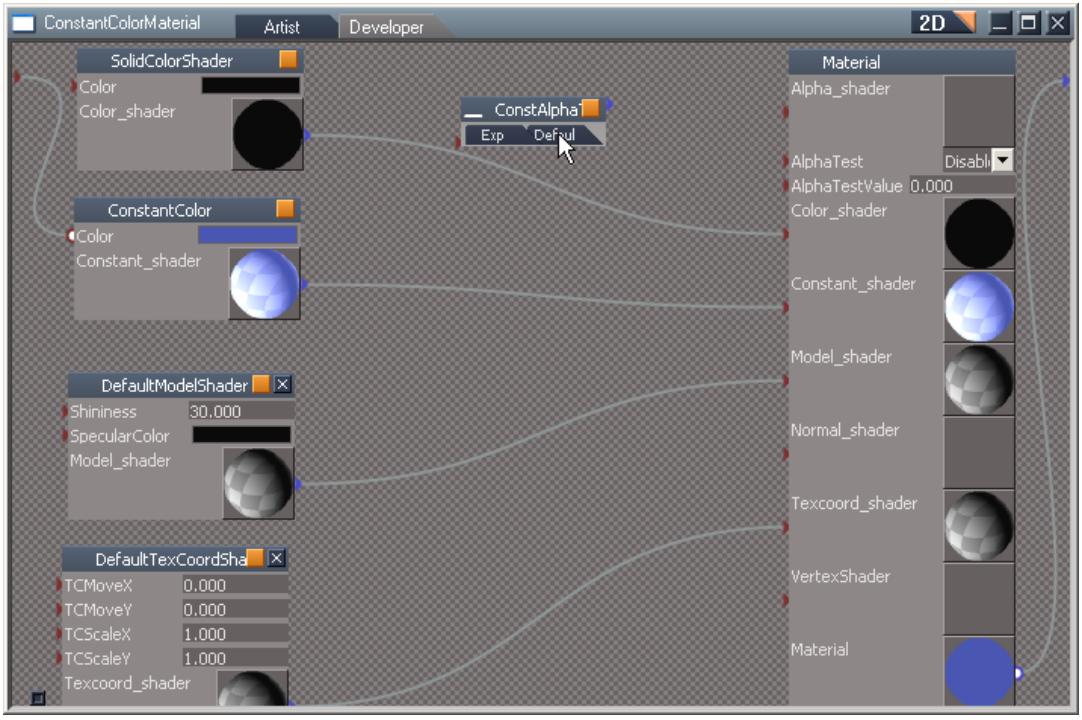


exporting the attribute for the Alpha Shader



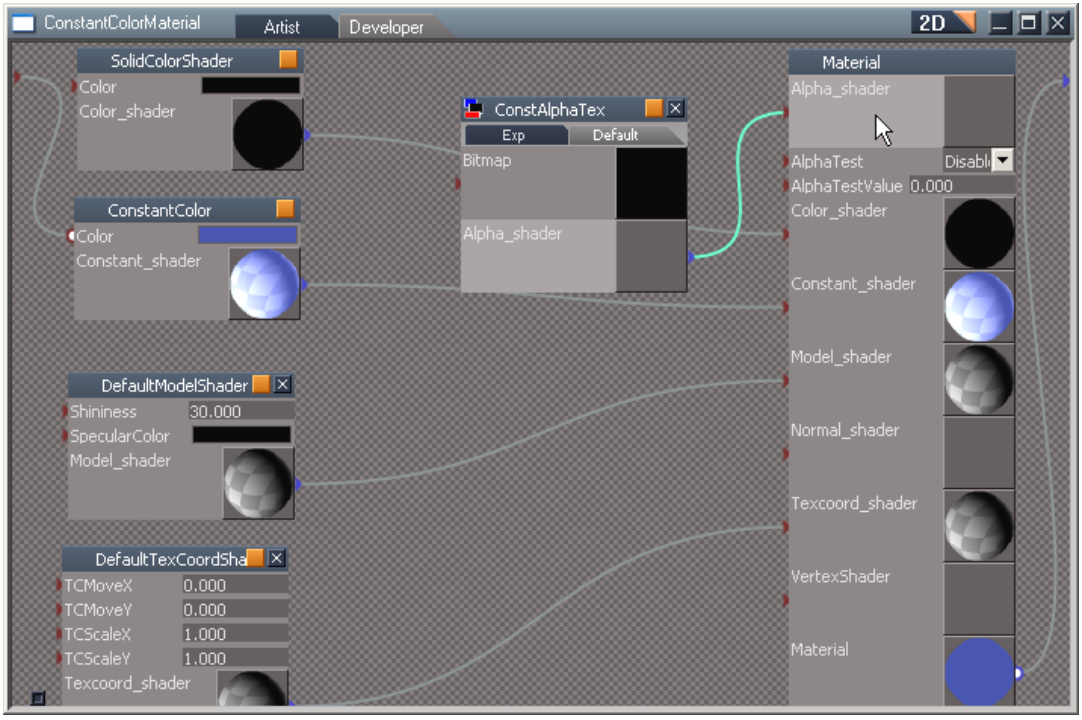
exported attributes show with a link to the outer walls of the object and a white cap on the connectors

Exit the object by using the orange triangle in the LE title bar, and once outside expand the new material component to its default aspect by clicking the tab in its titlebar.



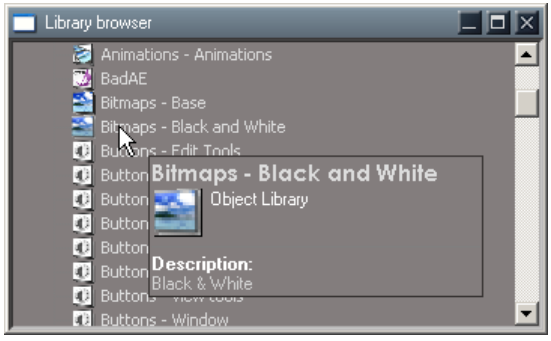
Expand to the default aspect

Drag a link from the ConstAlphaTex Alpha_shader output to the materials Alpha_shader input.



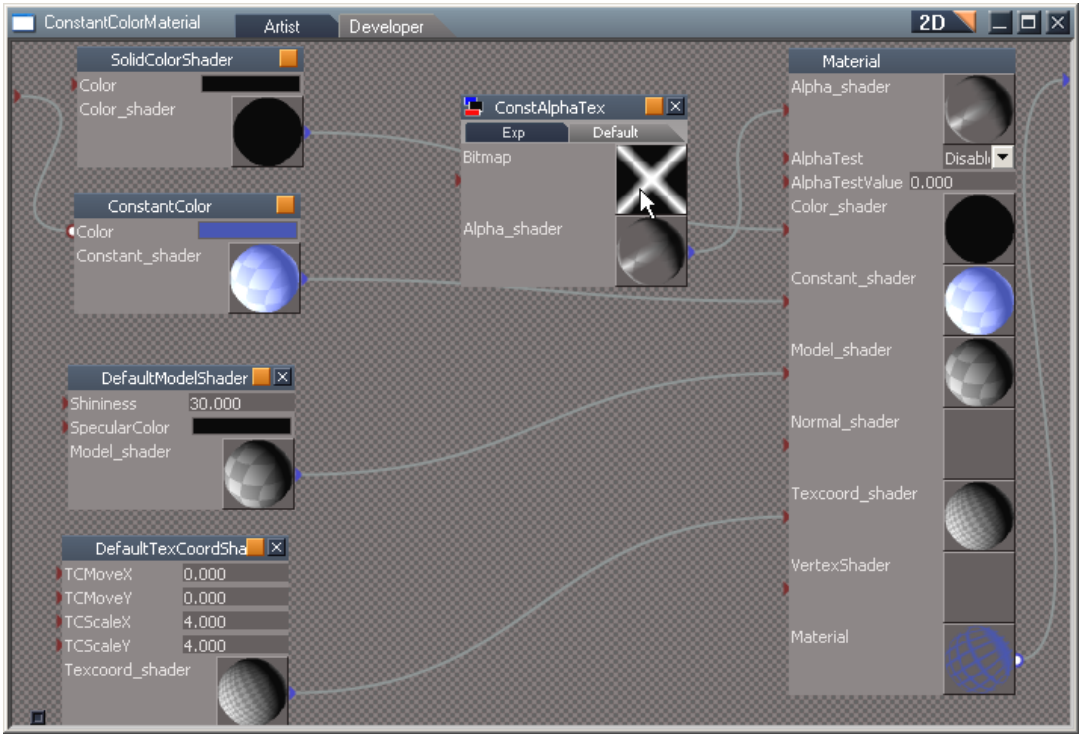
Linking the Alpha_Shader to the Material brick

Load the bitmap library and drag a texture onto the bitmap input and also change the texture scale to 4 in both the x and y



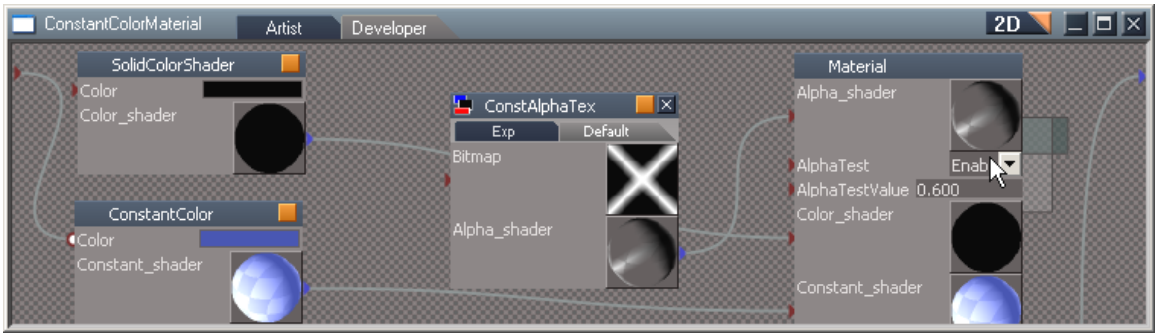
Library Browser

Bitmap library



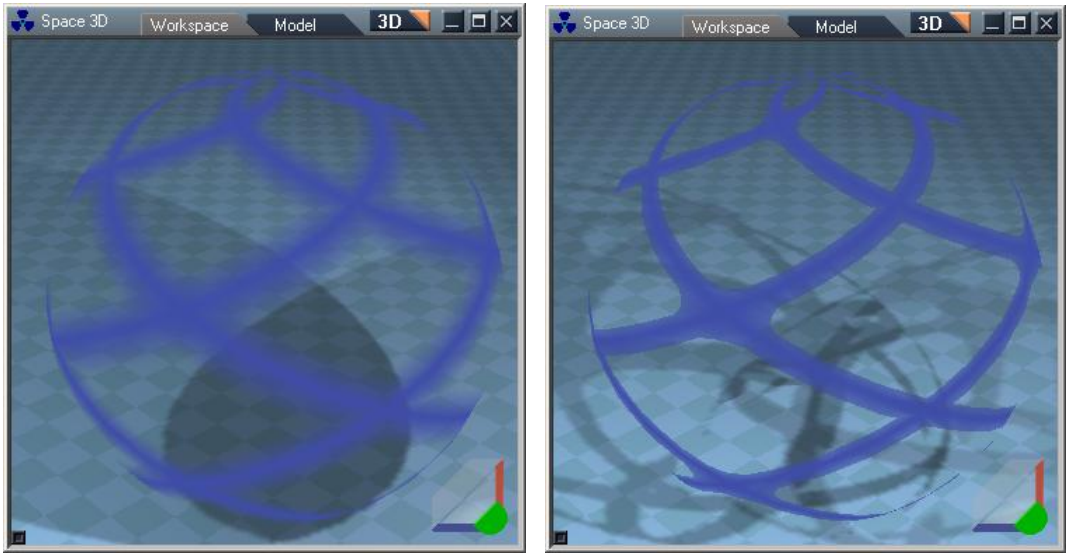
Scale changed to 4 in both the x and y in the DefaultTexCoord object and texture dragged and dropped

By adding an alpha shader component we have altered the look of the material by introducing transparency and altered the texture repeats by changing the scale in the default texcoord component. You can also enable the AlphaTest and put in a value of 0.6 or some other figure to make the shadows being cast take on the appearance of the textures alpha as well , this feature may be dependent on your current hardware and if your graphics card supports the realtime shadows transparencies.



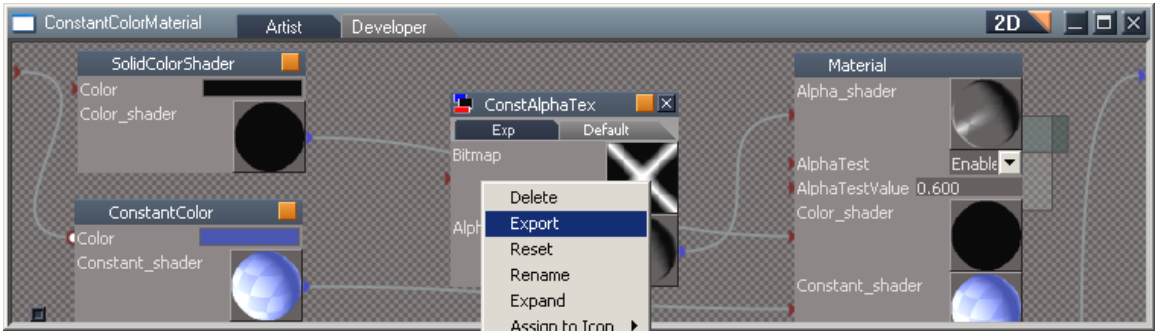
AlphaTest enabled and set to 0.6 value

When you have made these changes take a look at the Workspace view. Your sphere should look something like the ones in the image below.

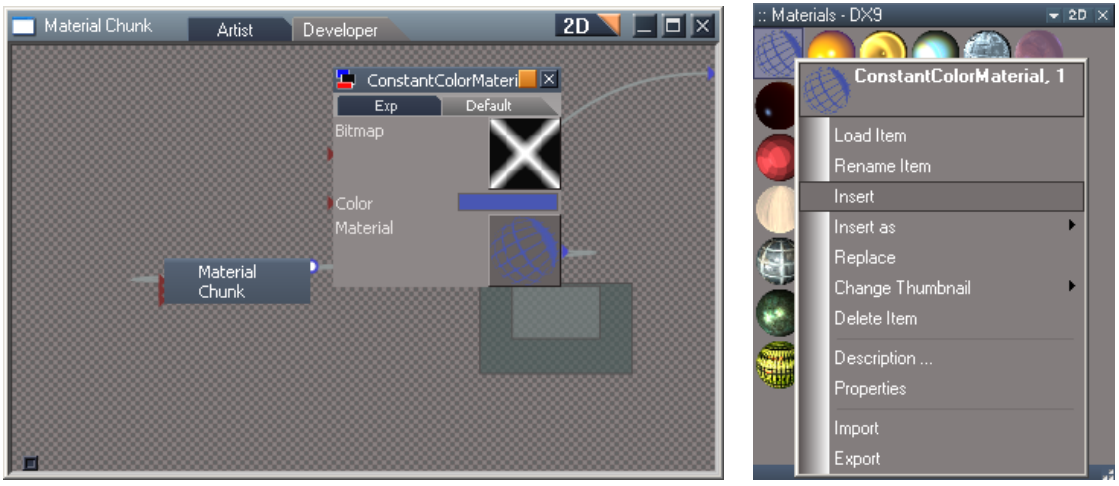


Left shows the sphere with transparency added -- -- Right with AlphaTest enabled for the shadows

Finally you can export the bitmap out further to the top level of the shader and then exit the material and save it to the shader library or to one that you've created specifically for your own materials .



exporting the attribute for the Bitmap one more level



Saving the new shader into a material library

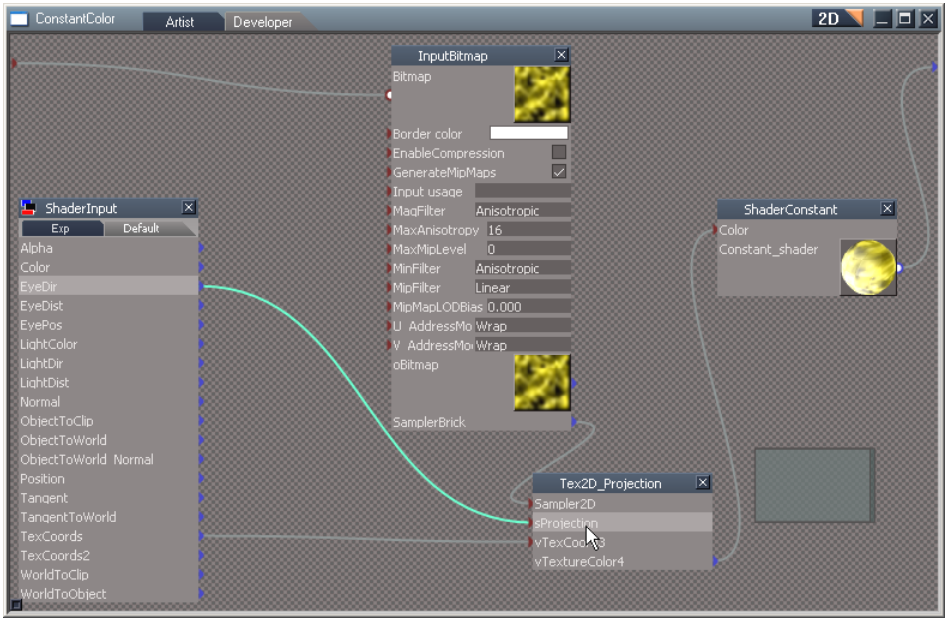
Often when you are designing a material, you can simply use an existing material from the library and modify it to create the desired effect. You can delete or modify existing shader components, or create new ones using the various components found in the libraries.

As an exercise, before attempting to construct a material from scratch, browse through the Material library and examine how the materials are built and how their various attributes are used to determine the final look of a material.

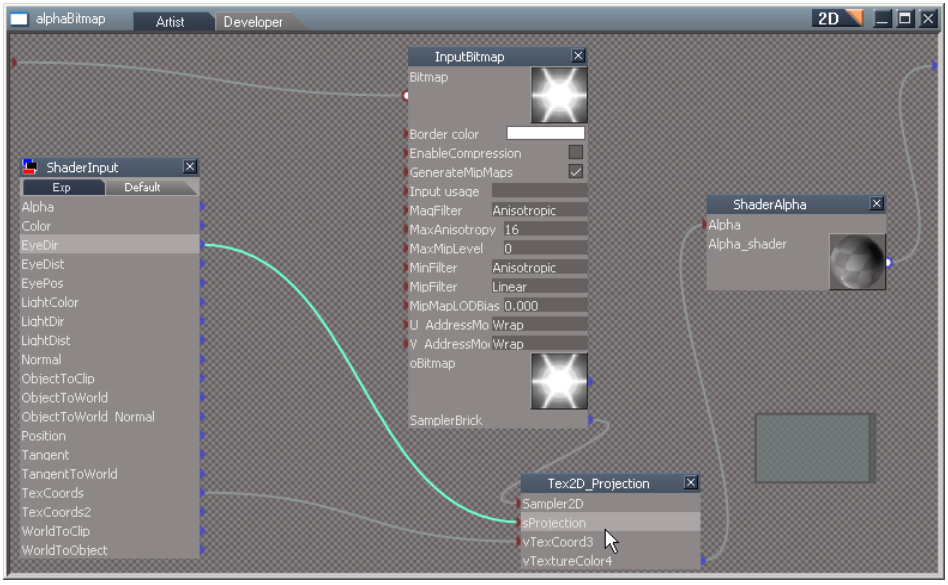
An additional exercise would be to change the color element in the shader so that it can use a texture.
The above principles learned can be used to adapt and change it by adding them into the ConstantColor part of the object.

More things to experiment with is linking the other parts of the ShaderInput :
For example link the EyeDir into the sProjection and see how this affects the object when you move around the sphere in the Workspace view.

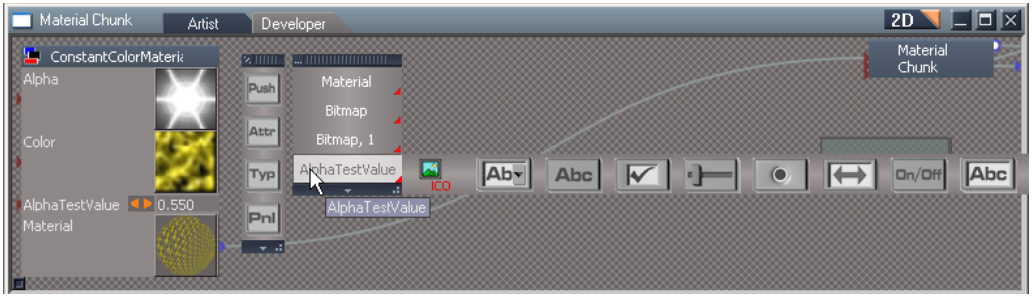
In the examples below I also exported the AlphaTestValue from the material brick and added a scrubber control to the panel. Other sections of the manual cover panel editing in more detail so they won't be covered in too much detail in this section.



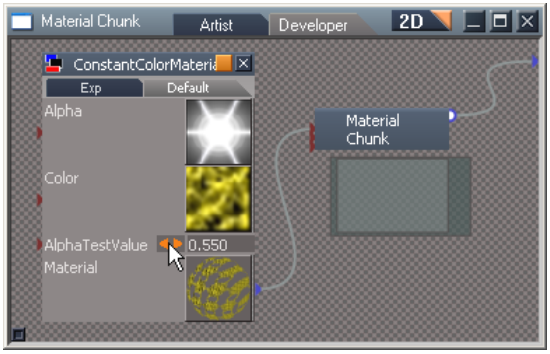
Linking the EyeDir to sProjection for the color



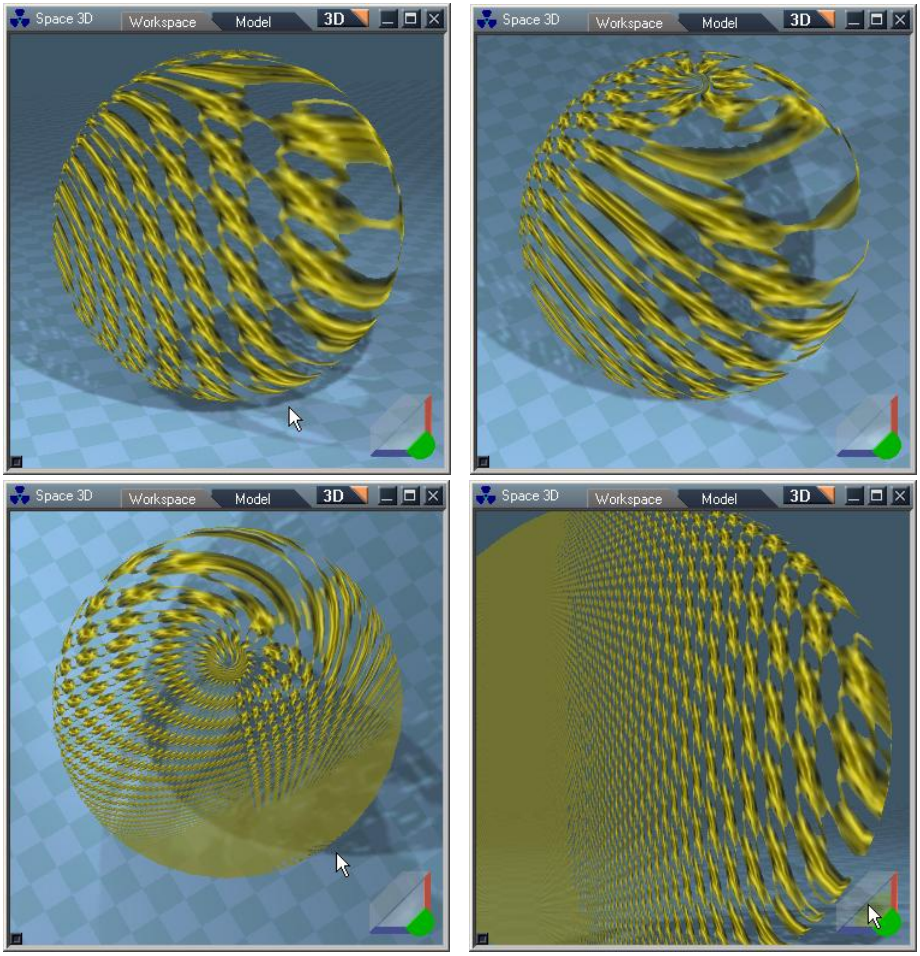
Linking the EyeDir to sProjection for the Alpha



Adding a control for the exported AlphaTestValue



Scrubber control for the exported AlphaTestValue on the top level of the material



Rotating or moving the view point in the Workspace shows different surface effects

- Further ref- Chapter 2 User interface - Panel Editing

5.4.6 Advanced Materials Example: Special Effects tSpecial-E Scene:

- Scene and Special effects material and guide produced and donated by David Bokon and can be found in the resources folder by following this link

5.5 More Advanced Material Editing

For advanced developers, shader scripts offer an even wider range of possibilities than the trueSpace Material Editor. Depending on your needs you can create shader scripts using HLSL for DirectX materials the shader scripts are covered in the sections below.
The method for constructing advanced materials in trueSpace is by combining shader and material component objects in the Link Editor, and you can also use HLSL to create scripted shaders.

You can use the components provided in the various libraries. Often you can simply modify an existing material to achieve the effect you want. Other times you will want to start from scratch, connecting components in the Link Editor to build up your desired effect.
You can also use scripting to create shaders for your objects. Although creating shader scripts is an involved process it is also extremely powerful, and gives complete and explicit control over every aspect of a shader's appearance.

trueSpace DirectX Materials

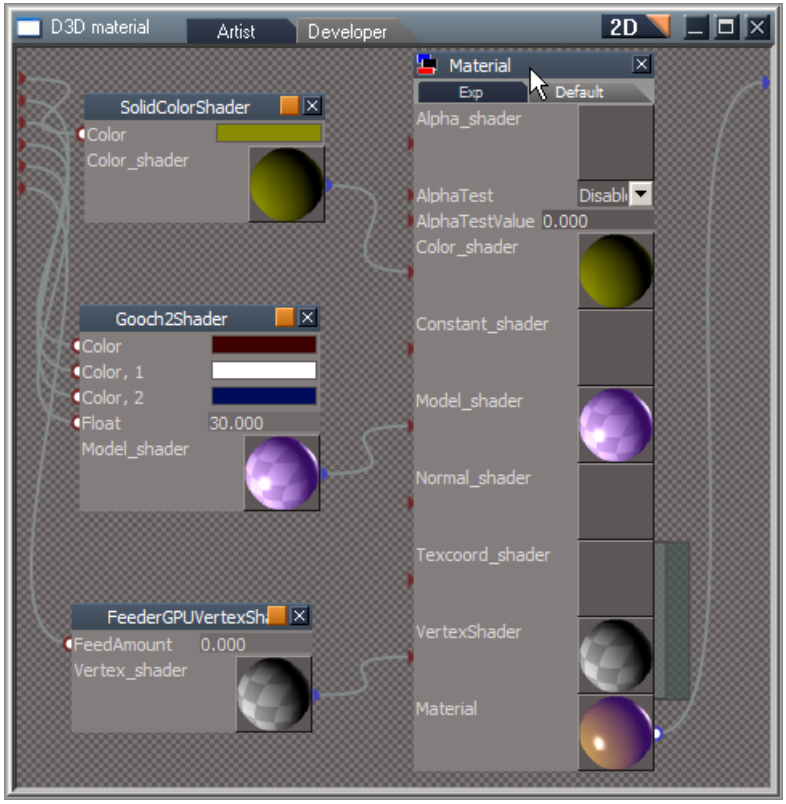
trueSpace offers **DirectX** material types which provide real-time surfacing and effects in the Workspace view, such as per-pixel or per-vertex operations, on your model surfaces and deciding what to use for a given task will depend on your goals.

These materials and their features are discussed in detail in the following sections.

DirectX Materials are real-time materials, meaning that certain components of these materials may be processed on your graphics card on an ongoing basis to create special rendering effects. While a DirectX material can be as simple as a solid color or bitmapped texture it can also include more complex, calculated tasks such as per-pixel coloration or per-vertex distortion.

A DirectX material may have several components.

There are a number of objects here, the most important of which is the Material object. This object essentially serves as a material 'compiler', taking the outputs of all connected shader components and combining them to create the end result. You can see that the Material attribute of this object is exported, giving trueSpace access to the final material for rendering in the Workspace window.



The Material object acts as a material 'compiler'

Shader Components

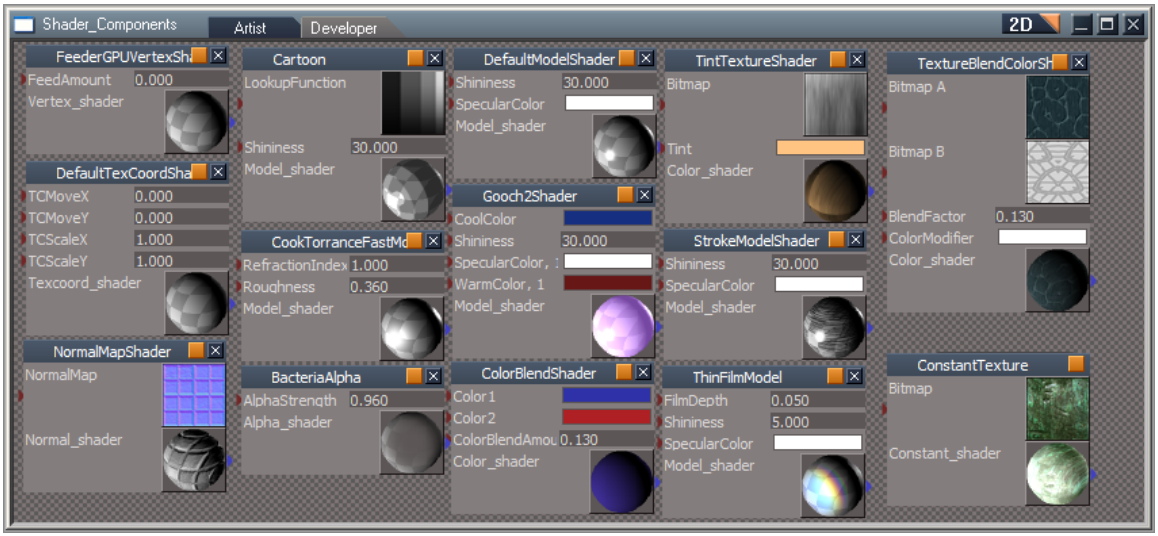
The Material object includes input connectors for seven separate shader components. These are:

- **Alpha Shader -** Information on transparency in the material. This can be defined by using a masking bitmap or through another means.
- **Color Shader -** Coloration data for the material. Like alpha, this information can be provided from a bitmap, by specifying a solid color, or via custom shader components.
- **Constant Shader** - texture and color data for the surfaces remains static and is not grossly affected by lighting conditions.
- **Model Shader -** The overall 'shading model' to be used for the material. The Model Shader specifies how light, shadow, specularity, reflectivity, and other components will interact to create the final, rendered look of the material on the object.
- **Normal Shader -** Specifies how the surface normals of the model will be used to affect the appearance of the rendered image. For instance, a bitmap could be used to alter surface normals to make the surface appear rough or to feature an embossed design.

- **Texcoord Shader** - Determines how the model's texture coordinates will be modified in rendering the final material. They can be stretched, offset, or completely altered by custom routines.
- **Vertex Shader** - DirectX materials can include a special shader that procedurally modifies the vertices of the model itself each frame. You can use this to create special effects such as displacement mapping or expansion and contraction.

There are many available pre-built shader components located in the libraries that appear when you enter a material in the Material Editor - or you can build your own shader components from the various shader objects, also found in these libraries. Advanced material designers can explore creating DirectX shader scripts to extend the possibilities even further.

A collection of shader components is shown in the following image.



Stock shader components for alpha, color, model, texture coordinates, normals, and vertex

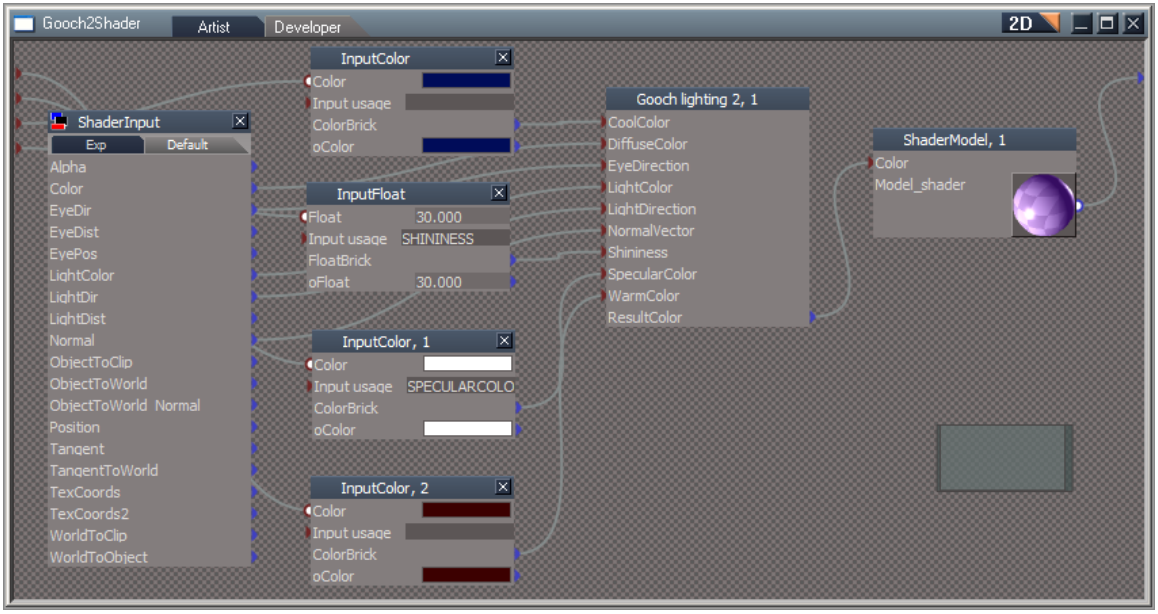
Note that the output for all of the shader components above matches one of the shader inputs on the Material object. So, for instance, to connect the SolidColorShader object to your material, simply join the Color Shader output on that object to the Color Shader input on your Material object. Other shader components can be connected in the same manner.

Custom Shader Components

Often you will want to create an effect that requires a custom shader component. Building a custom shader component in the Material Editor is similar to creating an activity or other object in the Link Editor. First, think about your effect and how you will build it, collect the necessary components from the Material Editor library, then connect them and

test the result by connecting the final output to the appropriate shader component input on the Material object.

One such example is the Gooch2Shader, found in the Materials library under Shaders. Drag this material onto an object in the Workspace view and then enter the object, then enter the Gooch2 object in the Link Editor. This will take you to the Material Editor view. Finally, enter the Gooch2Shader shader component and you should see the collection of objects shown below.



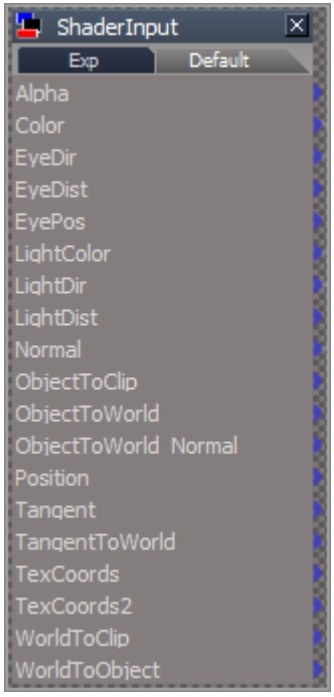
Inside a custom Model Shader we see the components that create the effect

While this looks like a complex assembly bear in mind that the only function of this component is to take into account lighting and surface information to tell trueSpace how to render the material. If you look to the far right of the image above you will see a ShaderModel object, with an exported Model Shader output attribute. This is, like Material, a sort of compiler - taking your finished computed color and converting it to the proper format for use with the Material object.

Gooch Lighting 2 is an object that takes in a number of details about the model, lighting, color, and other surface qualities, and computes a final color for the pixel being rendered based on a Gooch lighting algorithm. Because it is a compiled object we can't see what is going on inside it but information on the Gooch lighting model is available in any good computer graphics reference if you are interested. You can see that the object's input connectors are attached to several input color objects and a number input object.

Shader Component Inputs

Many of Gooch Lighting 2's inputs are connected to an object called ShaderInput, shown by itself below. This object is extremely important because its job is to provide information about the camera, lighting, the model and its surface, and more to your shader components on a per-pixel basic. That is, as your scene is being rendered, this object will constantly update itself - changing its Normal output, for instance, to match the normal vector of the model's surface at the point currently being rendered.



The ShaderInput object provides information to your shader components

As you can see there is a lot of information available via the ShaderInput object - a total of nineteen separate outputs. Some of these, such as Normal and Color, you will use quite frequently to power your custom shaders. Others, such as WorldToClip may be rarely used. A brief description of these outputs follows:

- **Alpha -** The transparency of the current point based on maps and other factors.
- **Color -** The color of the current point based on texture and other color contributors.
- **EyeDir -** Provides a world space vector to the camera from the current point.
- **EyeDist -** The current point's distance from the camera.
- **EyePos -** The location of the camera in world space.
- **LightColor -** The color of the light affecting the current point.

- **LightDir -** The world space vector to the light from the current point.
- **LightDist -** The distance from the light to the current point.
- **Normal -** Provides the normalized world space normal of the current point.
- **ObjectToClip -** Transform from local object space to screen space. Perspective projection - if you want the real screen coordinates then divide by the W component.
- **ObjectToWorld -** Transform from object to world space.
- **ObjectToWorld Normal -** Transform from object to world space - use with normals and tangents.
- **Position -** Provides the position of the current point in world space.
- **Tangent -** The normalized world space tangent of the current point.
- **TangentToWorld -** Transform from tangent to world space.
- **TexCoords -** The current point's texture coordinates in UV space.
- **TexCoords2 -** Coordinates from the second UV set.
- **WorldToClip -** Transform from world to screen space. Perspective projection - if you want the real screen coordinates then divide by the W component.
- **WorldToObject -** Transform from world to object space.

Material Compatibility

In order for your materials to render correctly on a wide range of hardware the trueSpace Direct3D pipeline provides a compatibility rendering mode. In order for materials to render on any generation of hardware you must specify "*usage*" information for some of the input objects (i.e. InputBitmap, InputFloat, …) used in your shader components.

The Direct3D pipeline is then able to use such input objects for rendering the simple Phong material with similar appearance, even if the original complex shader might not be available on your generation of graphics hardware. You can write the usage string into the "Usage" attribute of each shader input object.

Usage string	Description	Applicable input objects
DIFFUSEMAP	Diffuse texture	InputBitmap, InputBitmap1D
DIFFUSECOLOR	Constant diffuse color	InputColor*
DIFFUSEMAPSTRENGTH	Diffuse map strength	InputFloat*
VERTEXCOLORSTRENGTH	Vertex color strength	InputFloat*
NORMALMAP	Normal map	InputBitmap, InputBitmap1D
SPECULARCOLOR	Specular color	InputColor*
SHININESS	Shininess	InputFloat*
SPECULARSTRENGTH	Strength of specular reflections	InputFloat*
DIFFUSESTRENGTH	Strength of diffuse reflections	InputFloat*
C_TCSCALEX	Diffuse texture scale in X direction	InputFloat*

C_TCSCALEY	Diffuse texture scale in Y direction	InputFloat*
C_TCMOVEX	Diffuse texture move in X direction	InputFloat*
C_TCMOVEY	Diffuse texture move in Y direction	InputFloat*
N_TCSCALEX	Normal map scale in X direction	InputFloat*
N_TCSCALEY	Normal map scale in Y direction	InputFloat*
N_TCMOVEX	Normal map move in X direction	InputFloat*
N_TCMOVEY	Normal map move in Y direction	InputFloat*

*Optimal component, but any of these types can be used: InputFloat, InputPoint, InputColor, InputMatrix and the optimal conversion is done internally.

Go Back to DX Material Creation

5.5.1 Writing DirectX Shader Scripts

trueSpace DirectX shader scripts are built using a special Material Editor component called the HLSL Script Brick. By adding this script-based object, defining attributes, and creating an HLSL script to act on those attributes you can create any type of DirectX shader component.

What is HLSL?

Microsoft's HLSL (high-level shader language) is a special scripting language that allows developers to write shader programs that run on compatible graphics cards. HLSL is very similar to the C programming language with some extensions and limitations related specifically to programming graphics hardware.

A full description of HLSL is beyond the scope of this reference but you are encouraged to read the HLSL documentation and reference guide online at the Microsoft Developer Network site:

- The Microsoft DirectX developer site includes tutorials, articles, and an HLSL reference guide. http://msdn.microsoft.com/directx/

- The starting page of the DirectX Graphics documentation guide. The documentation tree also includes an HLSL guide with many shader examples and tutorials. http://msdn.microsoft.com/en-us/library/bb219838(VS.85).aspx

- The HLSL reference guide contains description of the language syntax, supported types, and all available functions. http://msdn.microsoft.com/en-us/library/bb509638(VS.85).aspx

When learning about HLSL it is important to understand the differences between HLSL and standard C. For example, HLSL supports vector operations on structures; it supports access to multiple members of a vector in a single

command and allows ‘component swizzling’ and ‘component masking’ which allows you to modify the order of vector members inside the command.

HLSL in trueSpace

trueSpace makes available a customized subset of HLSL, detailed below, to shader authors.

Comments

HLSL comments are the same as in C. Single line comments are preceded by two forward slashes (//). Block comments begin with a forward slash followed by an asterisk (/*) and end with an asterisk followed by a forward slash (*/).

```
//This is s single line comment

/*
   This is a block comment
*/
```

Data types

trueSpace provides a number of custom HLSL data types including types for vectors and matrices.

RtFloat

RtFloat is a single-component floating point number.

```
RtFloat x;
RtFloat b = 6.0f;
```

RtFloat3

RtFloat3 is a three-component floating point vector structure. The member components of this vector can be accessed using .x, .y, .z or .r, .g, .b suffix.

```
RtFloat3 x;
RtFloat3 b = {1.0f, 2.0f, 3.0f};
RtFloat3 c = RtFloat3(1.0f, 2.0f, 3.0f);
```

RtFloat4

RtFloat4 is a four-component floating point vector structure. The member components of this vector can be accessed

using .x, .y, .z, .w or .r, .g, .b, .a suffix.

```
RtFloat4 x;
RtFloat4 y = {1.0f, 2.0f, 3.0f, 4.0f};
RtFloat4 z = RtFloat3(1.0f, 2.0f, 3.0f, 4.0f);
```

RtFloat3x3 and RtFloat4x4

RtFloat3x3 and RtFloat4x4 are 3x3 and 4x4 component matrix structures, respectively. A matrix contains values organized in rows and columns and provides several member access methods (shown for a 4x4 matrix. The 3x3 matrix uses the top left of the 4x4 matrix).

The zero-based row-column positions are:

```
_m00, _m01, _m02, _m03
_m10, _m11, _m12, _m13
_m20, _m21, _m22, _m23
_m30, _m31, _m32, _m33
```

To access the value of row 0, column 0 use:

```
RtFloat x = Matrix._m00;
```

The one-based row-column positions are:

```
_11, _12, _13, _14
_21, _22, _23, _24
_31, _32, _33, _34
_41, _42, _43, _44
```

To access the value of row 1, column 1, use:

```
RtFloat x = Matrix._11;
```

Using an “array” type access the positions are:

```
[0][0], [0][1], [0][2], [0][3]
[1][0], [1][1], [1][2], [1][3]
[2][0], [2][1], [2][2], [2][3]
[3][0], [3][1], [3][2], [3][3]
```

To access the value at row 0, column 0, use:

```
RtFloat x = Matrix.[0][0];
```

To declare or declare and initialize a matrix, use:

```
RtFloat4x4 Matrix1;
RtFloat3x3 Matrix2 = {  1.0f, 2.0f, 3.0f, // row 1
```

```
                                    4.0f, 5.0f, 6.0f, // row 2
                                    7.0f, 8.0f, 9.0f, // row 3};
```

RtSampler1D and RtSampler2D objects

RtSampler1D is used to load color data from a one-dimensional texture, i.e. only an X coordinate. RtSampler2D can load color data from two-dimensional textures. Please note that samplers cannot be declared inside of an HLSL function; they are used only as input parameters.

Type conversion

To convert between data types you should use one of the methods provided below. These are the same methods that automatically convert between data types when connecting attributes of different types in the Material Editor itself. Please note that sampler objects cannot be converted.

When you need to convert one data type to another, use the following syntax:

```
RsConvert_<DestType>_<SourceType>(<destination_variable>, <source_variable>)
```

The following example converts an RtFloat4 vector to a single RtFloat value. It does this by picking only the .x component of the vector. In this case they .y, .z, and .w components are not converted because RtFloat can only hold a single floating point number.

```
RsConvert_RtFloat_RtFloat4(MyFloat, MyVector);
```

Conversion rules:

- Conversion to RtFloat is done by selecting the .x component of a vector or ._m00 component of a matrix. Conversion from RtFloat to any type components of the target type with that value.
- Conversion from RtFloat3 into RtFloat4 is done by filling the .w component of the target vector with a value of zero. Other components are filled by assignment. When converting from RtFloat4 to RtFloat3 the value of the .w component is lost
- Conversion from vector to matrix is done by replicating the vector into all rows of the matrix.
- Conversion from RtFloat3x3 into RtFloat4x4 is done by adding zero values to the last column and row of the matrix. When converting from RtFloat4x4 to RtFloat3x3 the last column and row values are lost.
- In any other case the conversion is done by filling matching components.

Swizzling and masking, per-component operations

HLSL can perform per-component operations on vector and matrix types. All basic operations (addition, subtraction, etc.) and comparisons are performed on a per-component basis. The following example shows the addition of two

RtFloat3 vectors:

```
RtFloat3 dest, stc0, src1;

dest = src0 + src1;
```

This is the same as writing:

```
dest.x = src0.x + src1.x;

dest.y = src0.y + src1.y;

dest.z = src0.z + src1.z;
```

Also almost all functions that work on scalar values (i.e. sin, cos, max, min, and many other) also accept either vector or matrix operands and perform the operation on all components of the operands.

Another extension of HLSL allows you to specify only a portion of a vector or matrix to take part in the operation. Masking on the destination variable allows you to limit the operation to affect only specific parts of the operand vector or matrix. The following example shows how to write only into the first two components of the vector while the .z component remains unchanged.

```
RtFloat3 dest, stc0, src1;

dest.xy = src0 + src1;
```

This is the same as writing:

```
dest.x = src0.x + src1.x;

dest.y = src0.y + src1.y;

dest.z = dest.z;              //just to show the unchanged component
```

Masking on source registers allows you to limit the operation to using only a part of the vector or matrix.

```
RtFloat3 dest, stc0, src1;

dest = src0 + src1.x;
```

This is the same as writing:

```
dest.x = src0.x + src1.x;

dest.y = src0.y + src1.x;
```

```
dest.z = src0.z + src1.x;
```

Swizzling allows you to change the order of components in the operation.

```
RtFloat3 dest, stc0, src1;

dest = src0.xyz + src1.zyz;
```

This is the same as writing:

```
dest.x = src0.x + src1.z;

dest.y = src0.y + src1.y;

dest.z = src0.z + src1.x;
```

Please refer to the DirectX HLSL reference and programming guide for a full list of allowed swizzle and masking combinations for various shader versions.

Functions

Functions are defined similarly to the C or Java (Jscript language).

```
return_type function_name(access param_type param_name1,
                      access param_type param_name2,
                      …,
                      access param_type param_name3)
{
  //function body
}
```

The return type can be also be void, which specifies that the function does not return a value using the return keyword.

The access specifier tells if the parameter can be read, written, or both:

- in - input parameter can be read within the function and is filled by the caller of the function
- out - output parameter can be written within the function and is read by the caller of the function. Can be used as a return value

- inout – parameter acts as in and out parameter at the same time.

Example:

```
//Function returns maximum of two values and fills Minimum parameter with minimum

//of two values
RtFloat3 MinMax(in RtFloat3 input1, in RtFloat3 input2, out RtFloat3 Minimum)

{

   Minimum = rtx_Min(input1, input2);

   return rtx_Max(input1, input2);

}
```

Note that rtx_Min and rtx_Max are predefined functions that will be described later.

Macros

trueSpace HLSL supports some preprocessor directives similar to the C language. This includes #define, #undef, #if, #elif, #else, #ifdef, #endif, #ifndef. Refer to the HLSL online documentation for a description of these preprocessor directives and available modifications.

Rosetta defines these custom tokens using #define:

- One (and only one) of following tokens specifying available pixel shader version: RSD3D_PROFILE_PS_2_0, RSD3D_PROFILE_PS_2_A, RSD3D_PROFILE_PS_2_B, RSD3D_PROFILE_PS_3_0.
- One (and only one) of following tokens specifying available vertex shader version: RSD3D_PROFILE_VS_2_0, RSD3D_PROFILE_VS_2_A, RSD3D_PROFILE_VS_3_0.
- #define ME_PS_PROFILE available_profile where available_profile might be “ps_2_0”, “ps_2_a”, “ps_2_b” or “ps_3_0”.
- #define ME_VS_PROFILE available_profile where available_profile might be “vs_2_0”, “vs_2_a” or “vs_3_0”.

These predefined macros allow you to modify your shader code according to the available shader version. Macros and preprocessor directives are a very powerful means of controlling the generated shader code. You should never change or undefine any predefined trueSpace macro or use directives that might affect the shader outside of the scope of your function.

Predefined functions

trueSpace HLSL provides a number of predefined functions that you can use in your own HLSL scripts. Most of these functions are also available in the form of shader components in the Material Editor.

The following table contains a complete list of predefined functions along with the syntax for using them. All functions are mappings of the original predefined HLSL functions and the name of the original function is shown in bold in the Description column. Use these original names to find more detailed help in the DirectX HLSL reference guide. All functions have the syntax: returnvalue = function(parameters). Some functions also return an additional value as the last parameter.

Name	Syntax	Description
rtx_Abs	rtx_Abs(a)	Absolute value (per component). **[abs]**
rxt_Acos	rxt_Acos(x)	Returns the arccosine of each component of x. Each component should be in the range [-1, 1]. **[acos]**
rxt_All	rxt_All(x)	Test if all components of x are nonzero. **[all]**
rxt_Any	rxt_Any(x)	Test if any component of x is nonzero. **[any]**
rxt_Asin	rxt_Asin(x)	Returns the arcsine of each component of x. Each component should be in the range [-pi/2, pi/2]. **[asin]**
rxt_Atan	rxt_Atan(x)	Returns the arctangent of x. The return values are in the range [-pi/2, pi/2]. **[atan]**
rxt_Atan2	rxt_Atan2(y, x)	Returns the arctangent of y/x. The signs of y and x are used to determine the quadrant of the return values in the range [-pi, pi]. atan2 is well-defined for every point other than the origin, even if x equals 0 and y does not equal 0. **[atan2]**
rxt_Ceil	rxt_Ceil(x)	Returns the smallest integer which is greater than or equal to x. **[ceil]**
rxt_Clamp	rxt_Clamp(x, min, max)	Clamps x to the range [min, max]. **[clamp]**

rxt_Clip	rxt_Clip(x)	Discards the current pixel, if any component of x is less than zero. This can be used to simulate clip planes, if each component of x represents the distance from a plane. **[clip]**
rxt_Cos	rxt_Cos(x)	Returns the cosine of x. **[cos]**
rxt_Cosh	rxt_Cosh(x)	Returns the hyperbolic cosine of x. **[cosh]**
rxt_Cross	rxt_Cross(a, b)	Returns the cross product of two 3-D vectors a and b. **[cross]**
rxt_Ddx	rxt_Ddx(x)	Returns the partial derivative of x with respect to the screen-space x-coordinate. **[ddx]**
rxt_Ddy	rxt_Ddy(x)	Returns the partial derivative of x with respect to the screen-space y-coordinate. **[ddy]**
rxt_Degrees	rxt_Degrees(x)	Converts x from radians to degrees. **[degrees]**
rxt_Determinant	rxt_Determinant(m)	Returns the determinant of the square matrix m. **[determinant]**
rxt_Distance	rxt_Distance(a, b)	Returns the distance between two points, a and b. **[distance]**
rxt_Dot	rxt_Dot(a, b)	Returns the • product of two vectors, a and b. **[dot]**
rxt_Exp	rxt_Exp(x)	Returns the base-e exponent. **[exp]**
rxt_Exp2	rxt_Exp2(a)	Base 2 Exp (per component). **[exp2]**
rxt_Faceforward	rxt_Faceforward(n, i, ng)	Returns -n * sign(•(i, ng)). **[faceforward]**
rxt_Floor	rxt_Floor(x)	Returns the greatest integer which is less than or equal to x. **[floor]**
rxt_Fmod	rxt_Fmod(a, b)	Returns the floating point remainder f of a / b such that a = i * b + f, where i is an integer, f has the same sign as x, and the absolute value of f is less than the absolute value of b. **[fmod]**

rxt_Frac	rxt_Frac(x)	Returns the fractional part f of x, such that f is a value greater than or equal to 0, and less than 1. **[frac]**
rxt_Frexp	rxt_Frexp(x, out exp)	Returns the mantissa and exponent of x. frexp returns the mantissa, and the exponent is stored in the output parameter exp. If x is 0, the function returns 0 for both the mantissa and the exponent. **[frexp]**
rxt_Fwidth	rxt_Fwidth(x)	Returns abs(ddx(x)) + abs(ddy(x)). **[fwidth]**
rxt_IsFinite	rxt_IsFinite(x)	Returns true if x is finite, false otherwise. **[isfinite]**
rxt_IsInfinite	rxt_IsInfinite(x)	Returns true if x is +INF or -INF, false otherwise. **[isinf]**
rxt_IsNAN	rxt_IsNAN(x)	Returns true if x is NAN or QNAN, false otherwise. **[isnan]**
rxt_Ldexp	rxt_Ldexp(x, exp)	Returns x * 2exp. **[ldexp]**
rxt_Length	rxt_Length(v)	Returns the length of the vector v. **[length]**
rxt_Lerp	rxt_Lerp(a, b, s)	Returns a + s(b - a). This linearly interpolates between a and b, such that the return value is a when s is 0, and b when s is 1. **[lerp]**
rxt_Lit	rxt_Lit(n • l, n • h, m)	Returns a lighting vector (ambient, diffuse, specular, 1): ambient = 1; diffuse = (n • l < 0) ? 0 : n • l; specular = (n • l < 0) || (n • h < 0) ? 0 : (n • h * m); **[lit]**
rxt_Log	rxt_Log(x)	Returns the base-e logarithm of x. If x is negative, the function returns indefinite. If x is 0, the function returns +INF. **[log]**
rxt_Log10	rxt_Log10(x)	Returns the base-10 logarithm of x. If x is negative, the function returns indefinite. If x is 0, the function returns +INF. **[log10]**
rxt_Log2	rxt_Log2(x)	Returns the base-2 logarithm of x. If x is negative, the function returns indefinite. If x is 0, the function returns +INF. **[log2]**
rxt_Max	rxt_Max(a, b)	Selects the greater of a and b. **[max]**

rxt_Min	rxt_Min(a, b)	Selects the lesser of a and b. **[min]**
rxt_Modf	rxt_Modf(x, out ip)	Splits the value x into fractional and integer parts, each of which has the same sign and x. The signed fractional portion of x is returned. The integer portion is stored in the output parameter ip. **[modf]**
rxt_Mul	rxt_Mul(a, b)	Performs matrix multiplication between a and b. If a is a vector, it is treated as a row vector. If b is a vector, it is treated as a column vector. The inner dimension acolumns and brows must be equal. The result has the dimension arows x bcolumns. **[mul]**
rxt_Normalize	rxt_Normalize(v)	Returns the normalized vector v / length(v). If the length of v is 0, the result is indefinite. **[normalize]**
rxt_Pow	rxt_Pow(x, y)	Returns x^y. **[pow]**
rxt_Radians	rxt_Radians(x)	Converts x from degrees to radians. **[radians]**
rxt_Reflect	rxt_Reflect(i, n)	Returns the reflection vector v, given the entering ray direction i, and the surface normal n, such that v = i - 2 * •(i, n) * n. **[reflect]**
rxt_Refract	rxt_Refract(i, n, ?)	Returns the refraction vector v, given the entering ray direction i, the surface normal n, and the relative index of refraction ?. If the angle between i and n is too great for a given ?, refract returns (0,0,0). **[refract]**
rxt_Round	rxt_Round(x)	Rounds x to the nearest integer. **[round]**
rxt_Rsqrt	rxt_Rsqrt(x)	Returns 1/rtx_Sqrt(x). **[rsqrt]**
rxt_Saturate	rxt_Saturate(x)	Clamps x to the range [0, 1]. **[saturate]**
rxt_Sign	rxt_Sign(x)	Computes the sign of x. Returns -1 if x is less than 0, 0 if x equals 0, and 1 if x is greater than zero. **[sign]**
rxt_Sin	rxt_Sin(x)	Returns the sine of x. **[sin]**

rxt_SinCos	rxt_SinCos(x, out s, out c)	Returns the sine and cosine of x. sin(x) is stored in the output parameter s. cos(x) is stored in the output parameter c. **[sincos]**
rxt_Sinh	rxt_Sinh(x)	Returns the hyperbolic sine of x. **[sinh]**
rxt_Smoothstep	rxt_Smoothstep (min, max, x)	Returns 0 if x < min. Returns 1 if x > max. Returns a smooth Hermite interpolation between 0 and 1, if x is in the range [min, max]. **[smoothstep]**
rxt_Sqrt	rxt_Sqrt(a)	Square root (per component). **[sqrt]**
rxt_Step	rxt_Step(a, x)	Returns (x >= a) ? 1 : 0. **[step]**
rxt_Tan	rxt_Tan(x)	Returns the tangent of x. **[tan]**
rxt_Tanh	rxt_Tanh(x)	Returns the hyperbolic tangent of x. **[tanh]**
rxt_Tex1D	rxt_Tex1D(s, t)	1-D texture lookup. s is a sampler or a sampler1D object. t is a scalar. **[tex1D]**
rxt_Tex1D	rxt_Tex1D(s, t, ddx, ddy)	1-D texture lookup, with derivatives. S is a sampler or sampler1D object. t, ddx, and ddy are scalars. **[tex1D]**
rxt_Tex1DProj	rxt_Tex1DProj(s, t)	1-D projective texture lookup. s is a sampler or sampler1D object. t is a 4-D vector. t is divided by its last component before the lookup takes place. **[tex1Dproj]**
rxt_Tex1DBias	rxt_Tex1DBias(s, t)	1-D biased texture lookup. S is a sampler or sampler1D object. t is a 4-D vector. The mip level is biased by t.w before the lookup takes place. **[tex1Dbias]**
rxt_Tex2D	rxt_Tex2D(s, t)	2-D texture lookup. s is a sampler or a sampler2D object. t is a 2-D texture coordinate. **[tex2D]**
rxt_Tex2D	rxt_Tex2D(s, t, ddx, ddy)	2-D texture lookup, with derivatives. s is a sampler or sampler2D object. t, ddx, and ddy are 2-D vectors. **[tex2D]**
rxt_Tex2DProj	rxt_Tex2DProj(s, t)	2-D projective texture lookup. s is a sampler or sampler2D object. t is a 4-D vector. t is divided by its last component before the lookup takes place. **[tex2Dproj]**

rxt_Tex2DBias	rxt_Tex2DBias(s, t)	2-D biased texture lookup. s is a sampler or sampler2D object. t is a 4-D vector. The mip level is biased by t.w before the lookup takes place. **[tex2Dbias]**
rxt_Transpose	rxt_Transpose(m)	Returns the transpose of the matrix m. If the source is dimension mrows x mcolumns, the result is dimension mcolumns x mrows. **[transpose]**
rtx_Modulate	rtx_Modulate(a,b)	Returns per component a * b. **[*]**
rtx_Add	rtx_Add(a,b)	Returns per component a + b. **[+]**
rtx_Sub	rtx_Sub(a,b)	Returns per component a – b. **[-]**
rtx_Divide	rtx_Divide(a,b)	Returns per component a / b. **[/]**
rtx_Modulus	rtx_Modulus(a,b)	Returns per component a modulus b. **[%]**
rtx_Negate	rtx_Negate(a)	Returns per component –a. **[-]**
rtx_Not	rtx_Not(a)	Returns per component logical not a. **[!]**

trueSpace HLSL Shader Components

HLSL shader components are well-defined pieces of shader code that can be combined together in the Material Editor to create a shader. The core of a shader component is a function with a name that is the same as the name of the component. This function has a custom set of input and output parameters. trueSpace represents this function as a visible object in the Material Editor and you can connect the object to other shader components to create your material.

Shader components have several important constraints:

- The shader component function cannot return a value using the *return* keyword – all shader components use functions with a return type of *void*.
- Function parameters can only use *in* or *out* access specifiers. They cannot include *inout* access specifiers.
- Function parameters can be only be one of the previously specified trueSpace HLSL data types (i.e. RtFloat, RtFloat3x3, etc.). Other HLSL datatypes (like float2) can be only specified inside of the function as local variables.
- If you declare helper functions to be used by the main shader component function, you can use any data types for parameters or the return value. The name of these functions must, however, contain the name of the main function as a suffix.
- Violation of these rules will, in most cases, result in uncompilable shader code.

Predefined Shader Components

The following shader components are available in the shader component library.

Components - Compound

Name	Alpha Texture HL
Description	This component returns alpha stored in the red channel of the texture
Attributes	
in RtSampler2D AlphaSampler	Texture that contains alpha values in the red channel
in RtFloat3 TextureCoordinates	Sampling texture coordinates
in RtFloat AlphaTexScaleX	Horizontal scale of the texture coordinates
in RtFloat AlphaTexScaleY	Vertical scale of the texture coordinates
in RtFloat AlphaTexMoveX	Horizontal translate of the texture coordinates
in RtFloat AlphaTexMoveY	Vertical translate of the texture coordinates
out RtFloat Alpha	resulting alpha value

Name	Anisotropic lighting
Description	Simple anisotropic lighting model
Attributes	
in RtFloat4 DiffuseColor	Source diffuse color
in RtFloat3 NormalVector	Current normal vector of the point
in RtFloat3 EyeDirection	Current eye direction
in RtFloat3 LightDirection	Current light direction
in RtFloat4 LightColor	Color of the light
in RtSampler2D AnisotropicMap	Special texture that represents the actual amount of lighting it is stored in the file <installdir>/ts/scripts/d3d/Aniso.png
out RtFloat4 ResultColor	final color of the point

Name	Color texture
Description	This brick returns color stored in a texture
Attributes	
in RtSampler2D ColorSampler	Source 2D texture
in RtFloat3 TextureCoordinates	texture coordinates
out RtFloat4 Color	resulting color

Name	Color Texture HL
Description	This brick returns color stored in a texture with modified texture coordinates

Attributes	
in RtSampler2D ColorSampler	Source 2D texture
in RtFloat3 TextureCoordinates	texture coordinates
in RtFloat ColorTexScaleX	Horizontal scale of the texture coordinates
in RtFloat ColorTexScaleY	Vertical scale of the texture coordinates
in RtFloat ColorTexMoveX	Horizontal translate of the texture coordinates
in RtFloat ColorTexMoveY	Vertical translate of the texture coordinates
out RtFloat4 Color	resulting color value

Name	Cook-Torrance lighting	
Description	Brick computes the Cook-Torrance lighting model which is good for reflective surfaces such as metal. On Pixel Shader 2.0 hardware it falls-back to Phong model with equal the same and diffuse color components which produces similar metallic effect.	
Attributes		
in RtFloat Roughness	The roughness of the surface. Expected values should be in range 0…1 and higher roughness means less shiny surface and less intensive highlights.	
in RtFloat RefractionIndex	This parameter controls the amount of reflected and refracted light. Expected values are in range 0…1. Low values mean smaller reflectivity with more account to real Fresnel term.	
in RtFloat4 DiffuseColor	Diffuse color of the point	
in RtFloat3 NormalVector	Normal vector of the point	
in RtFloat3 EyeDirection	Current eye direction	
in RtFloat3 LightDirection	Current light direction	
in RtFloat4 LightColor	Light color	
out RtFloat4 ResultColor	Final, computed color	

Name	Cook-Torrance 2	
Description	Brick computes the Cook-Torrance lighting model which is good for reflective surfaces such as metal. On Pixel Shader 2.0 hardware it falls-back to Phong model with equal the same and diffuse color components which produces similar metallic effect. The difference from previous brick is that most of the computation is stored in a special function which saves instructions.	
Attributes		
in RtSampler2D Precomputation	Fresnel term and Becman distribution function stored in a HDRI texture <installdir>/ts/scripts/d3d/ RtD3D_BeckmannDistributionFresnel.dds.	

	Cards without support for HDRI textures will not display this material correctly.
in RtFloat Roughness	The roughness of the surface. Expected values should be in range 0…1 and higher roughness means less shiny surface and less intensive highlights.
in RtFloat RefractionIndex	This parameter controls the amount of reflected and refracted light. Expected values are in range 0…1. Low values mean smaller reflectivity with more account to real Fresnel term.
in RtFloat4 DiffuseColor	Diffuse color of the point
in RtFloat3 NormalVector	Normal vector of the point
in RtFloat3 EyeDirection	Current eye direction
in RtFloat3 LightDirection	Current light direction
in RtFloat4 LightColor	Light color
out RtFloat4 ResultColor	Final, computed color

Name	Gooch lighting
Description	The Gooch lighting and shading model was developed to better show geometrical properties of objects. It works best with single light setup.
Attributes	
in RtFloat4 CoolColor	Cool color is used in areas with high angles to the viewer
in RtFloat4 WarmColor	Warm color is used in areas with small angles to the viewer
in RtFloat CoolModifier	This modifier changes the amount of diffuse color in cool areas.
in RtFloat WarmModifier	This modifier changes the amount of diffuse color in warm areas.
in RtFloat4 DiffuseColor	Current diffuse color.
in RtFloat4 SpecularColor	Specular color of the material.
in RtFloat Shininess	Shininess of the material
in RtFloat3 NormalVector	Current normal vector
in RtFloat3 EyeDirection	Current eye direction
in RtFloat3 LightDirection	Current light direction
in RtFloat4 LightColor	Light color
out RtFloat4 ResultColor	Final, computed color

Name	Gooch lighting 2
Description	The Gooch lighting and shading model was developed to better show geometrical properties of objects. This brick provides simpler, better controlled model that works better in multilight environments and shadows.

Attributes	
in RtFloat4 CoolColor	Cool color is used in areas with high angles to the viewer
in RtFloat4 WarmColor	Warm color is used in areas with small angles to the viewer
in RtFloat4 DiffuseColor	Current diffuse color.
in RtFloat4 SpecularColor	Specular color of the material.
in RtFloat Shininess	Shininess of the material
in RtFloat3 NormalVector	Current normal vector
in RtFloat3 EyeDirection	Current eye direction
in RtFloat3 LightDirection	Current light direction
in RtFloat4 LightColor	Light color
out RtFloat4 ResultColor	Final, computed color

Name	Hair Shader
Description	This brick computes Ward's anisotropic lighting model that is used for example for hair simulation.
Attributes	
in RtFloat4 DiffuseColor	Current diffuse color.
in RtFloat4 SpecularColor	Specular color of the material.
in RtFloat Shininess	Shininess of the material
in RtFloat3 TangentVector	Tangent vector used for computing the surface orientation
in RtFloat3 EyeDirection	Current eye direction
in RtFloat3 LightDirection	Current light direction
in RtFloat4 LightColor	Light color
out RtFloat4 ResultColor	Final, computed color

Name	Normal-Map
Description	This brick reads normals stored in a normal map and returns these normals transformed into the world space
Attributes	
in RtSampler2D NormalMap	Normal map texture
in RtFloat3 TextureCoords	Sampling coordinates
in RtFloat3x3 TangentToWorld	Matrix that transforms vectors from tangent space to the world space.
out RtFloat3 NormalVector	Final normal vector

Name	Normal-Map HL
Description	This brick reads normal stored in a normal map and returns these normals transformed into the world space. Additional texture coordinates transformation

	are possible.
Attributes	
in RtSampler2D NormalMap	Normal map texture
in RtFloat3 TextureCoords	Sampling coordinates
in RtFloat3x3 TangentToWorld	Matrix that transforms vectors from tangent space to the world space.
out RtFloat3 NormalVector	Final normal vector
in RtFloat NormalTexScaleX	Horizontal scale of the texture coordinates
in RtFloat NormalTexScaleY	Vertical scale of the texture coordinates
in RtFloat NormalTexMoveX	Horizontal translate of the texture coordinates
in RtFloat NormalTexMoveY	Vertical translate of the texture coordinates

Name	Phong Lighting
Description	The phong lighting model
Attributes	
in RtFloat4 DiffuseColor	Current diffuse color.
in RtFloat4 SpecularColor	Specular color of the material.
in RtFloat Shininess	Shininess of the material
in RtFloat3 NormalVector	Current normal vector
in RtFloat3 EyeDirection	Current eye direction
in RtFloat3 LightDirection	Current light direction
in RtFloat4 LightColor	Light color
out RtFloat4 ResultColor	Final, computed color

Name	TS Phong Lighting
Description	The phong lighting model similar to that used in TS 6.x
Attributes	
in RtFloat DiffuseStrength	Strength of the diffuse component in range 0…1
in RtFloat4 DiffuseColor	Current diffuse color.
in RtFloat SpecularStrength	Strength of the specular component in range 0…1
in RtFloat Shininess	Shininess of the material
in RtFloat3 NormalVector	Current normal vector
in RtFloat3 EyeDirection	Current eye direction
in RtFloat3 LightDirection	Current light direction
in RtFloat4 LightColor	Light color
out RtFloat4 ResultColor	Final, computed color

Name	Thin Film

Description	The phong lighting model with thin oil film effect on the surface.
Attributes	
in RtFloat4 DiffuseColor	Current diffuse color.
in RtFloat4 SpecularColor	Specular color of the material.
in RtFloat Shininess	Shininess of the material
in RtFloat3 NormalVector	Current normal vector
in RtFloat3 EyeDirection	Current eye direction
in RtFloat3 LightDirection	Current light direction
in RtFloat4 LightColor	Light color
in RtSampler1D ThinFilmSample	1D texture with the film coat samples. <installdir>/ts/scripts/d3d/thinfilm.dds.
in RtFloat FilmDepth	Depth of the film layer
out RtFloat4 ResultColor	Final, computed color

Name	Hatching NPR
Description	The hatching non photo realistic rendering which uses a texture for the hatch and stroke simulation and chooses hatch level based on the intensity.
Attributes	
in RtFloat4 Intensity	Intensity of the pixel for example the result of lighting model.
in RtFloat3 Texcoords	Texture coordinates of the object for reading of the texture.
in RtSampler2D HatchingTexture	Texture with strokes of different densities in each channel. <installdir>/ts/scripts/d3d/threshold_hatching2.dds.
out RtFloat4 ResultColor	Final, computed color

Functions
All XML files from <installdir>/ts/scripts/MaterialEditor/Functions folder.

Logical
All XML files from <installdir>/ts/scripts/MaterialEditor/Logical folder.

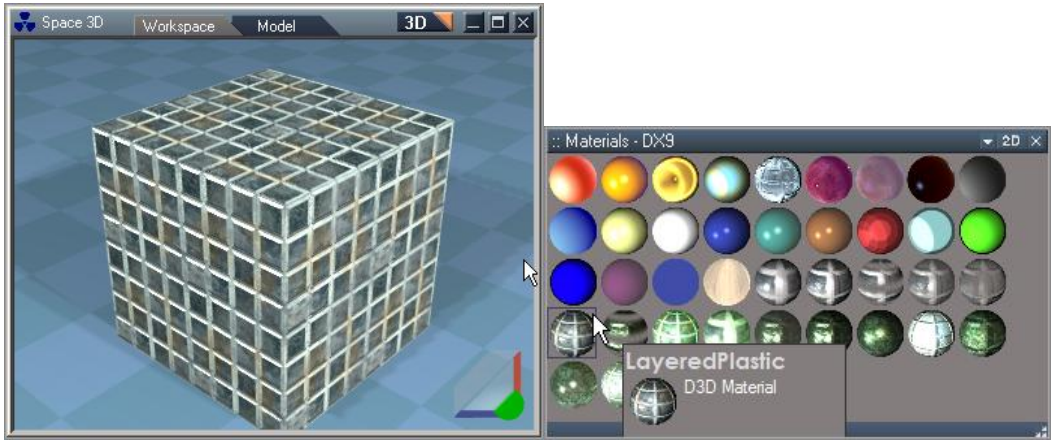
Matrices & Vectors
All XML files from <installdir>/ts/scripts/MaterialEditor/ Matrices & Vectors folder.
Operators
All XML files from <installdir>/ts/scripts/MaterialEditor/ Operators folder.
Texturing
All XML files from <installdir>/ts/scripts/MaterialEditor/ Texturing folder.

Go back DX Material Components Basic.

5.5.2 Tutorial: A Simple Shader Component with HLSL

This tutorial shows you how to create a shader component that uses texture coordinates to create a color shader outputting alternating color squares like a checker board. After you create the HLSL color shader component you will use it to modify an existing material and apply it to an object.

Add a Cube, and apply the LayeredPlastic material to it.

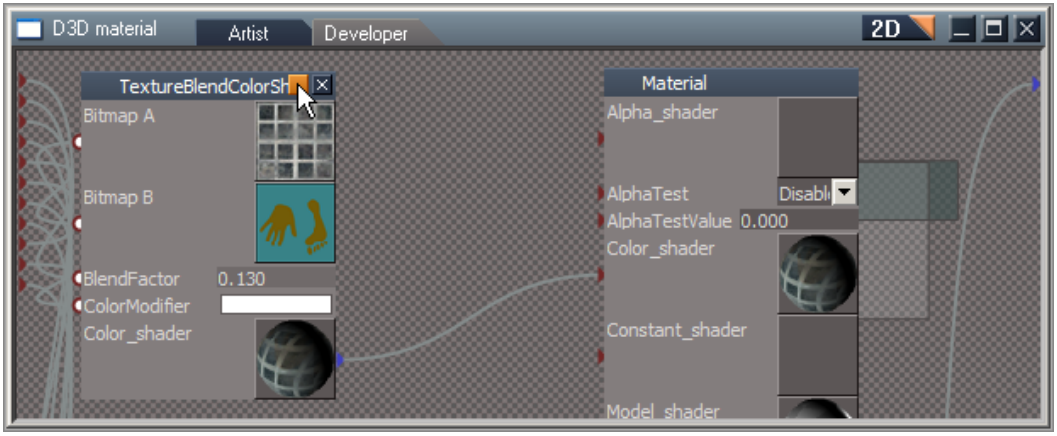


apply the LayeredPlastic material onto the cube

Pick the material then click the button to Edit in the Link Editor

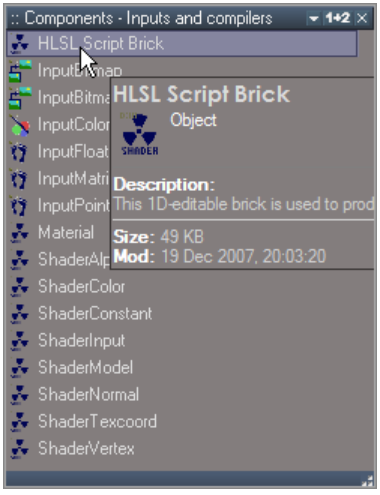
Edit the material in the LE

Enter the TextureBlendColorShader using the orange square in its title-bar.

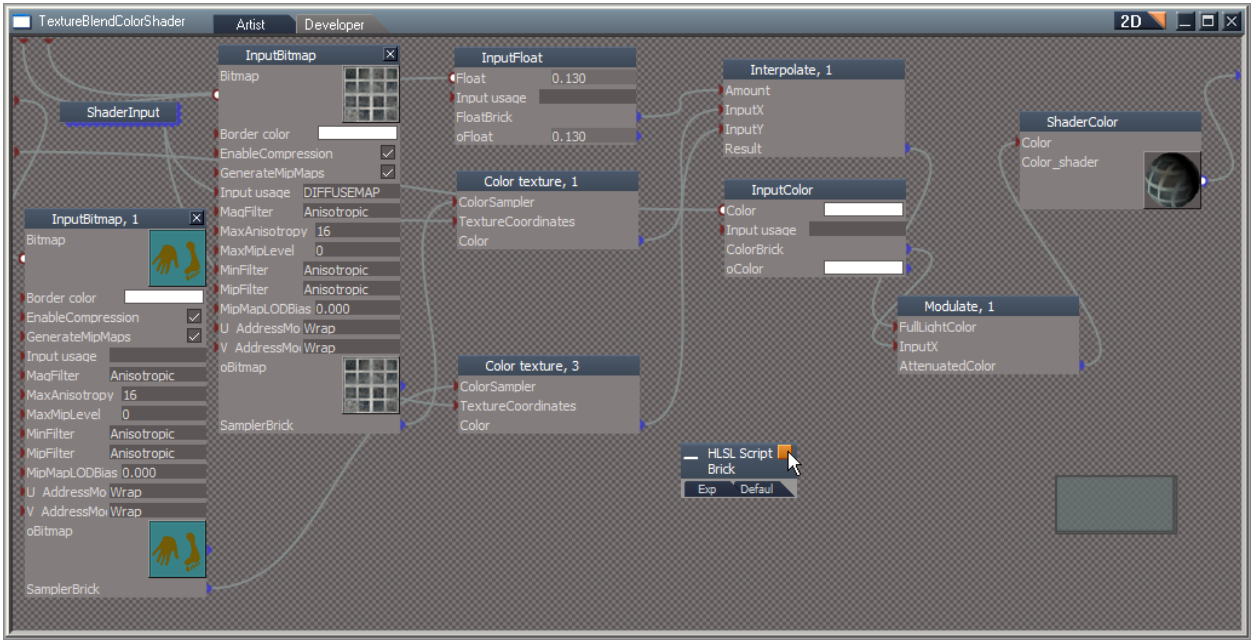


Material in the LE

Now you need to create an HLSL scripting object to hold your new shader script. Open the shader components library using the library browser. Locate the HLSL Script Brick in the Inputs and Compilers category and drag it into the Link Editor as shown below.



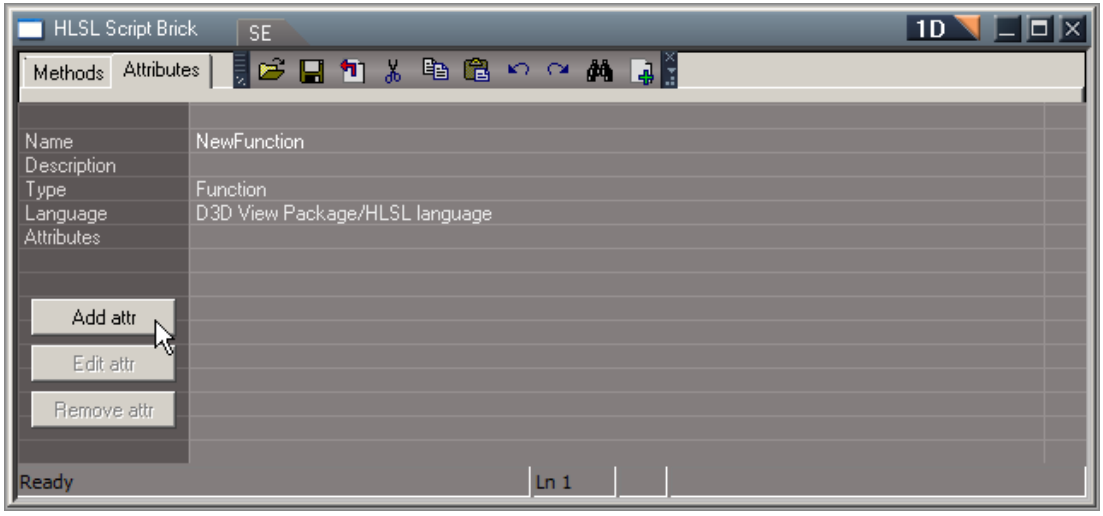
HLSL Script brick



Open the Components Library and Drag and Drop an HLSL Shader Brick object

Then enter the HLSL Script Brick and it will open the script editing interface.

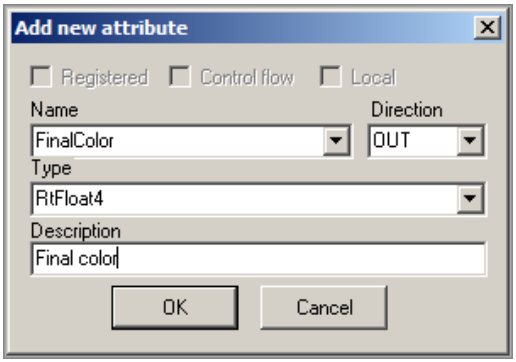
- Reference: Ch2 Script Editor.



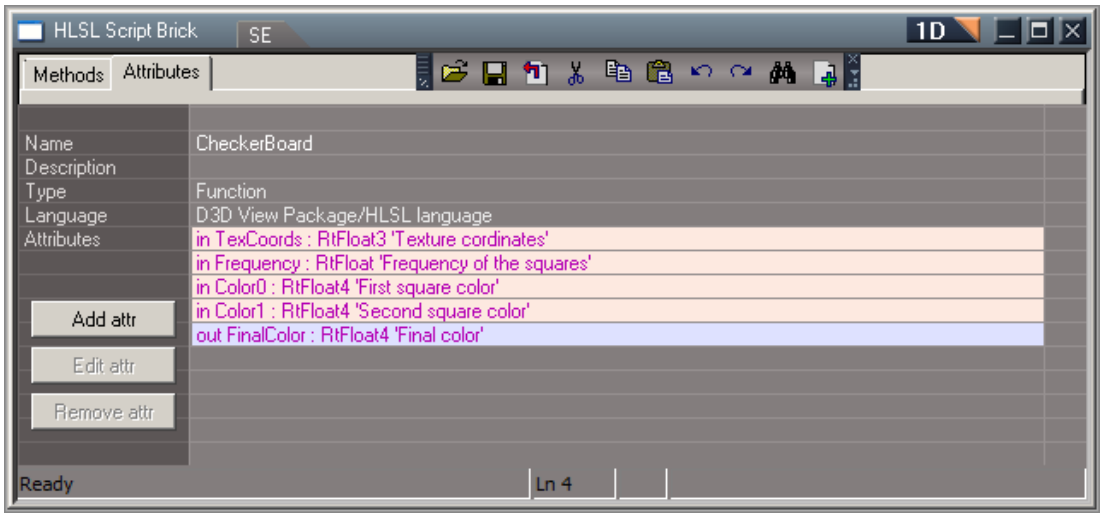
Script editor attributes tab

First you must define the attributes of the shader component by adding the attributes. This component requires access to texture coordinates and needs two color inputs (for the checker board colors) and a number representing the frequency of the squares. It also needs one output color. The following image shows the complete list of attributes and

their types. Rename the object to something descriptive like *CheckerBoard*.

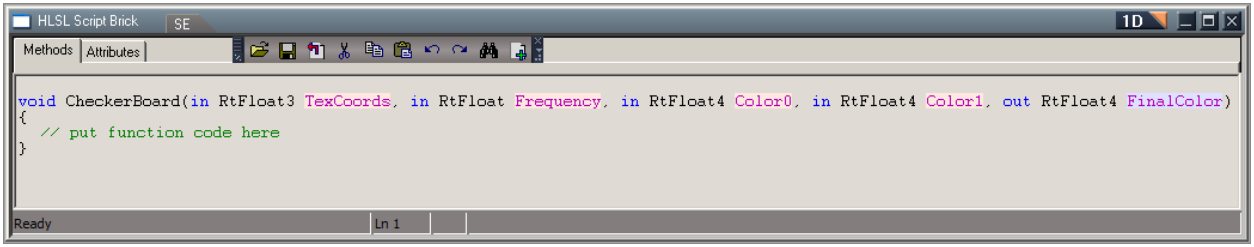


Adding attributes



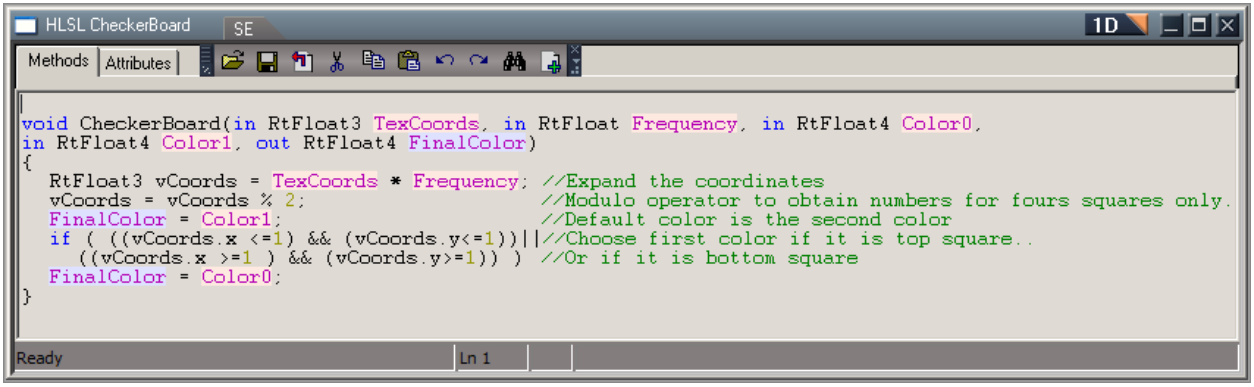
Attributes for the CheckerBoard shader component

Once you've added the all the attributes needed for the shader in the attributes tab switch the editor to the Methods tab:



Methods tab where the functions code needs adding

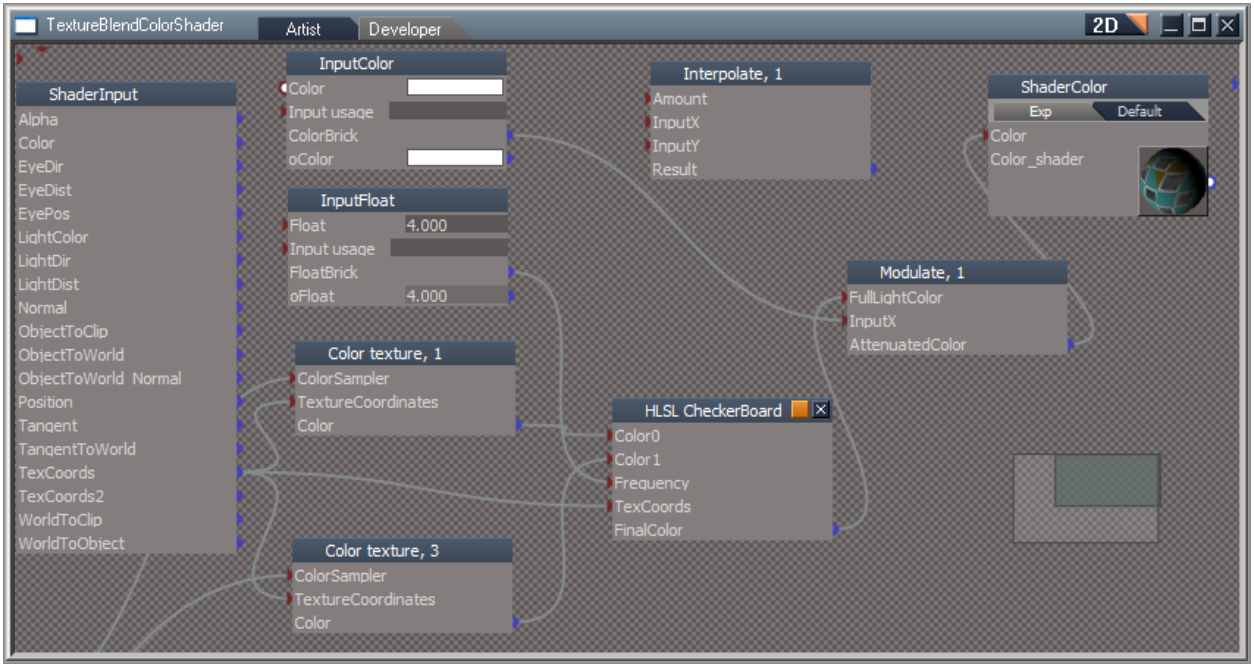
The HLSL code to generate the checkerboard pattern is fairly simple. First it multiplies the texture coordinates by the frequency. This changes the numbers from an initial range of zero to one to a range of zero to the value of frequency. Then it uses the Modulo operator (%) to determine which color should be output for a given texture coordinate. If the -x and -y coordinates are both less than or equal to one or are both greater than one, then Color0 is used, otherwise Color1 is used. The complete script is shown in the following image.



The completed Checkerboard shader component script

After you exit the script editor using the orange triangle in the top right of the title bar you can disconnect the Interpolate object, which currently controls texture blending, and connect your new HLSL script instead. Connect FinalColor to InputY on the modulate object and connect the Color outputs of the two Color Texture objects to the Color0 and Color1 inputs of the script.

Finally, connect the FloatBrick output of the InputFloat object to the Frequency input on the script object.



Connecting the Checkerboard shader component script in the Material Editor

Now, the BlendFactor attribute on the outer LayeredPlastic panel will control the frequency of the checkerboard pattern. If you disconnect the Input float and Change the value to 4 and observe the results -- you should see something similar to the image below.

If you re-link the Input float to its exported connector again and exit the material and edit its top level with Panel editing you can change the values for the slider to allow for higher numbers to be used.

Then when you adjust the slider you should see the checker board script applying changes to your objects surface.

Using the BlendFactor gives different results , if you panel edit the Materials you can set this to higher numbers, ie using 4 as a value gives a dense effect.

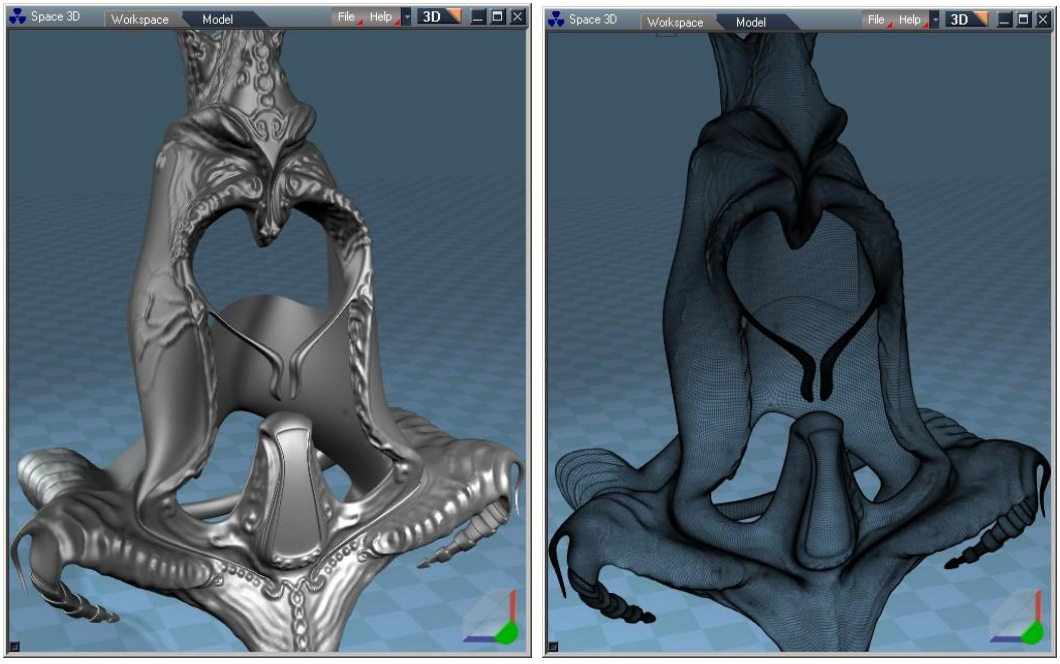
- **Resource- HLSL checkerboard example**
- **Reference Panel Editing**
- External References.
- **http://www.facewound.com/tutorials/shader1/** this is a relatively easy tutorial to follow and adapt for the trueSpace HLSL brick
- **http://www.neatware.com/lbstudio/web/hlsl.html** this has slightly more complicated examples that can be adapted for the trueSpace HLSL brick..

5.6 Normal Mapping

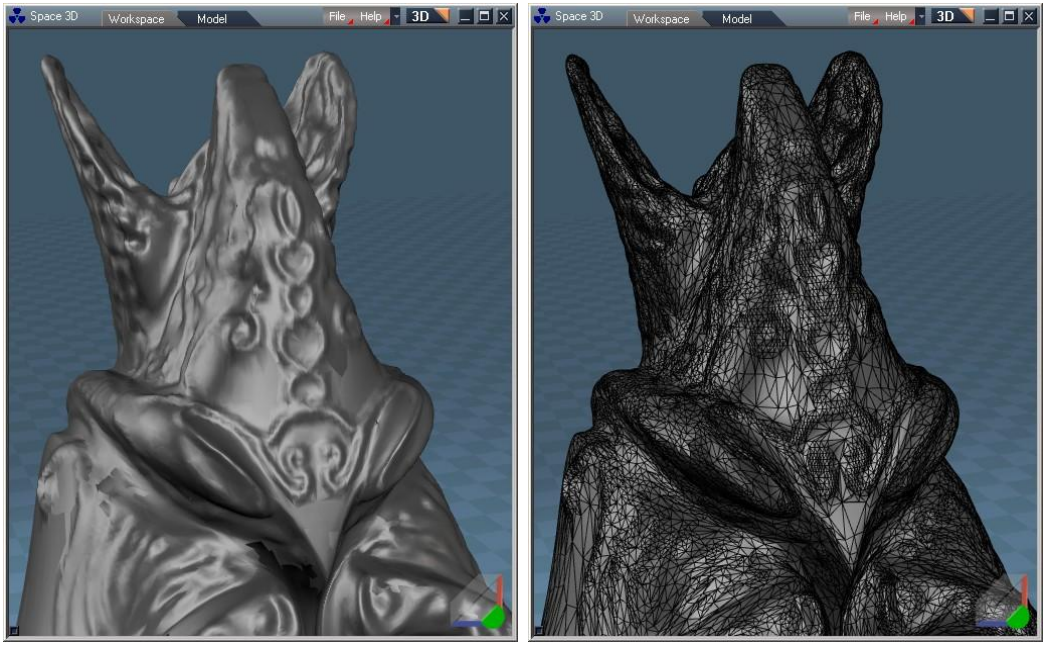
The Workspace view supports the display of normal maps, which function similarly to the bump maps you may already be familiar with in offline render engines. Instead of using a greyscale bitmap with pixels encoding the relative "height" of the surface at a given point, normal maps use a full color image to encode the actual normal of the surface at that point. This results in a much improved method of increasing the apparent geometry of an object without adding more polygons.
trueSpace provides a mesh simplifier and normal map generator tools to help you turn detailed geometric objects into simpler objects with normal maps that take up much less memory and processing power but look nearly identical in the real-time Player view.

Simplify mesh and Normal map tool.

The below images illustrate an example of this by using a mesh supplied by Marcel Barthel. As you can see in the close up view there is still definition of geometry and details in the appearance of the Helmet mesh even though a considerable amount of geometry has been removed by using the Normal map tool.

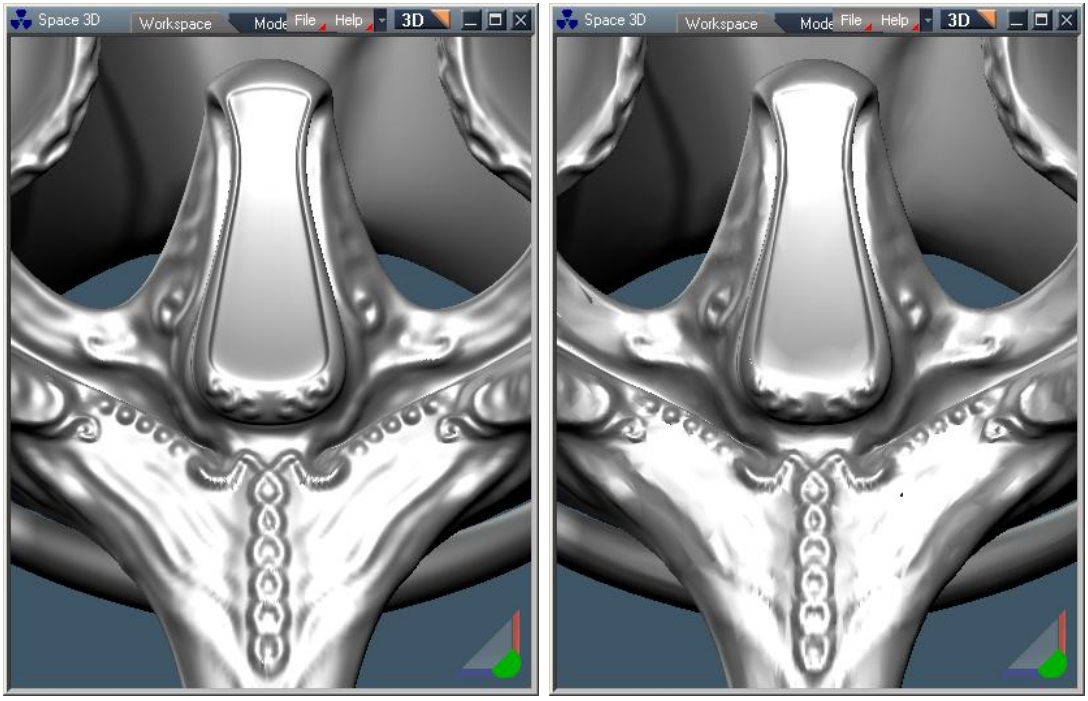


Helmet object by Marcel Barthel original object has 530,449 vertices notice how dense the mesh is when the object is viewed in wireframe in the Workspace window.



Close up view of the Helmet object after simplifying the mesh with the Normal map tool at 10 resulting in 51,213 vertices notice the appearance of the surface is still detailed even though the mesh has been greatly reduced.

Another view of the same object left is the original and the right image is also after using the Normal map tool with a Simplification Level setting of 10.



A different close up view of the Helmet object after using the Simplify mesh and Normal map tool at level 10 original object is on the left 530,449 vertices and on the right 51,213 vertices

5.6.1 Simplifying Meshes for Normal Mapped Objects

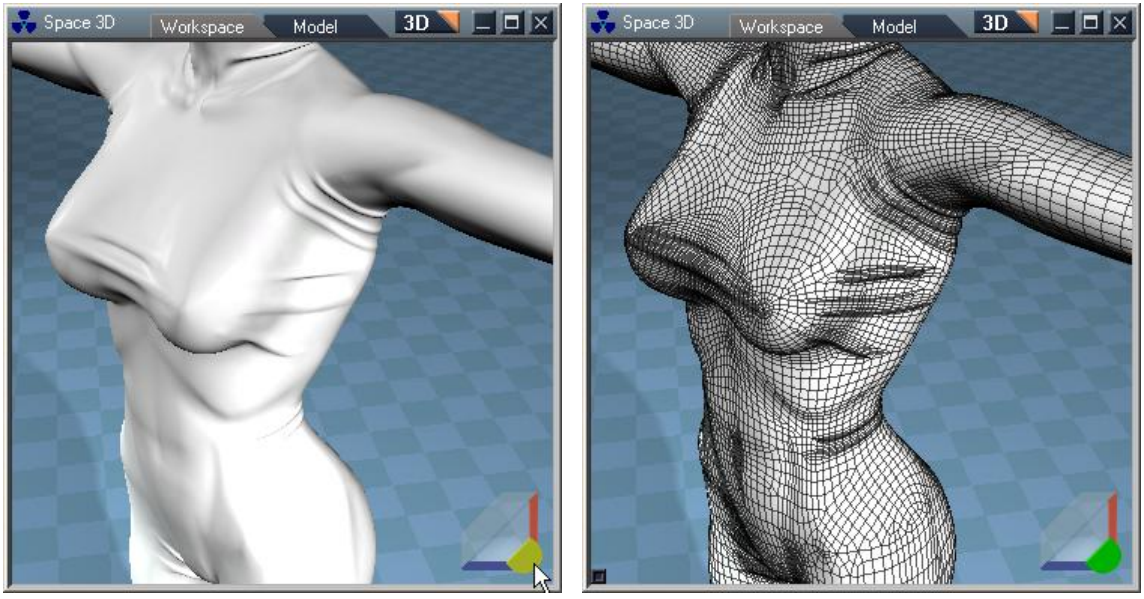
Simplify Mesh

The trueSpace mesh simplifier can be used to generate meshes that approximate high-polygon-count meshes, but with far fewer polygons. The resulting objects are perfect candidates for the application of normal maps (described later in this section). This simplification is achieved by removing vertices, edges, and faces from the high-poly mesh, but doing so in a way that leaves the resulting mesh looking very similar to the original.

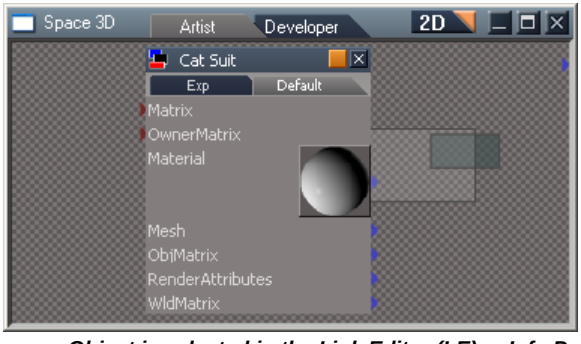
Building a Simplified Mesh

As an example you can drag and drop the Cat Suit from the Characters Library to use but the principle can be applied to any mesh of your own you want to use.

Cat Suit in the Characters Library

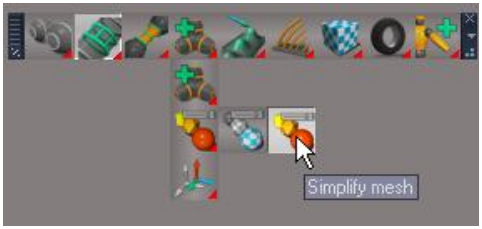


Cat Suit dropped to Workspace. Right image shows wireframe view of the mesh



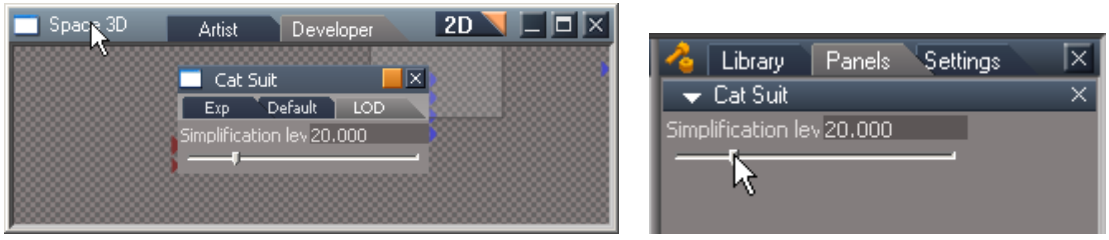
Object is selected in the Link Editor (LE) ***Info Panel shows Original Catsuit model, 31,728 Vertices.***

To create a lower resolution mesh from an existing high-poly mesh, select the object in the Link Editor (LE) or Workspace and click the Simplify Mesh tool.

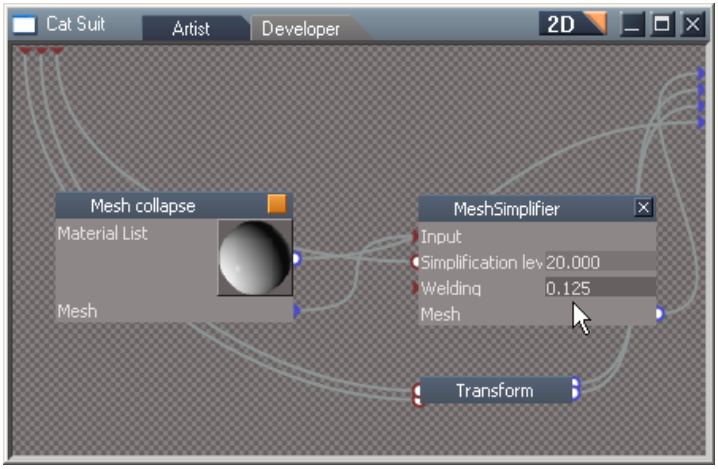


Simplify mesh tool

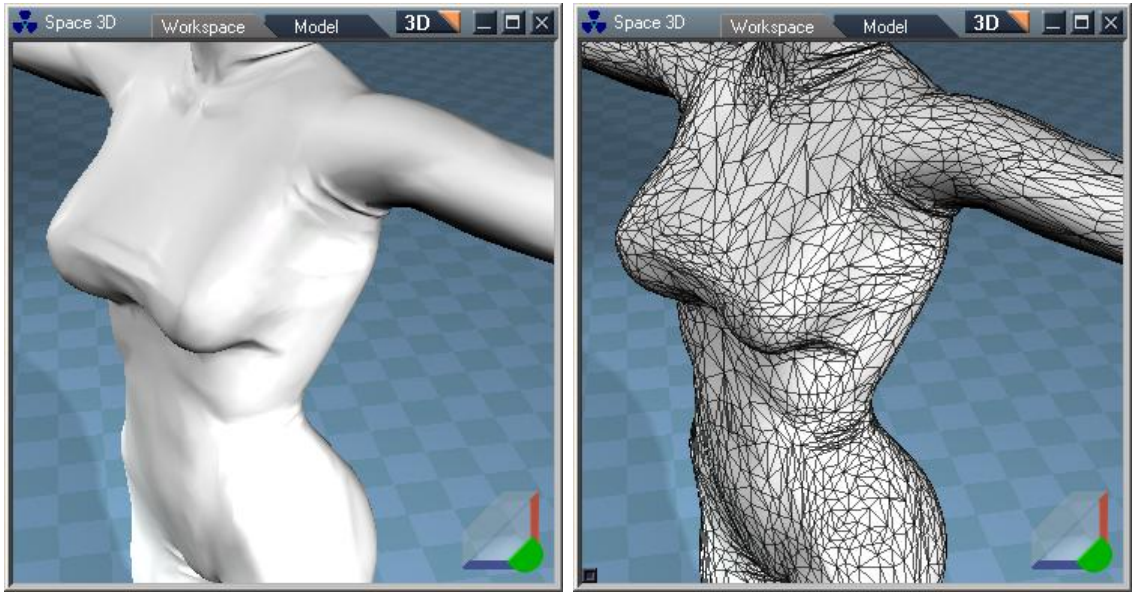
The **Simplify Mesh** tool takes the original mesh and creates a simplified version of it. It creates a panel in the LE and also opens one for the tool in the Panels tab of the stack where you can set the desired simplification level.
You can specify the desired simplification level either by entering a numeric value in the text box or by moving the slider left or right. You can also change settings such as welding distance (for correcting mesh topology errors) by examining the MeshSimplifier object in the newly created simplified mesh. The original mesh remains encapsulated inside of the newly created compound object and can be further modified if desired, with any changes propagating to the simplified mesh.



Left image - the LE Simplify Mesh Panel created for the object. Right - Panel in the stack.

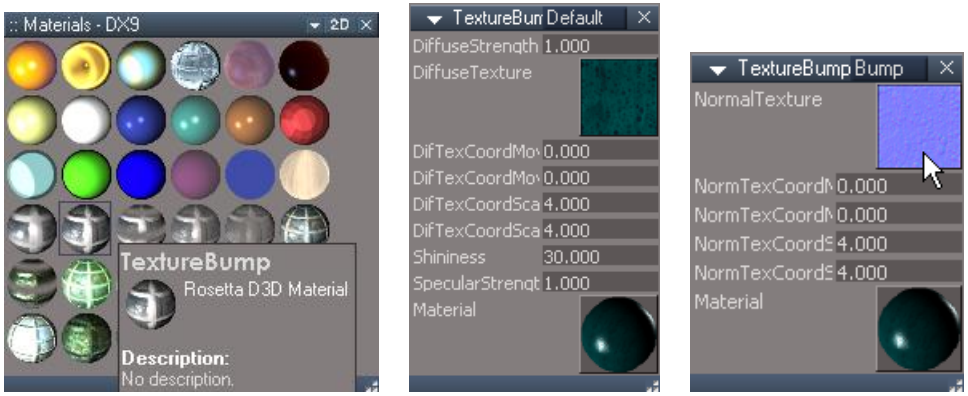


Inside the Cat Suit MeshSimplifier Welding distance attribute value can be changed if there are topology errors.



Simplified Catsuit - right image shows wireframe view of the mesh. Simplified model, 6,278 vertices.

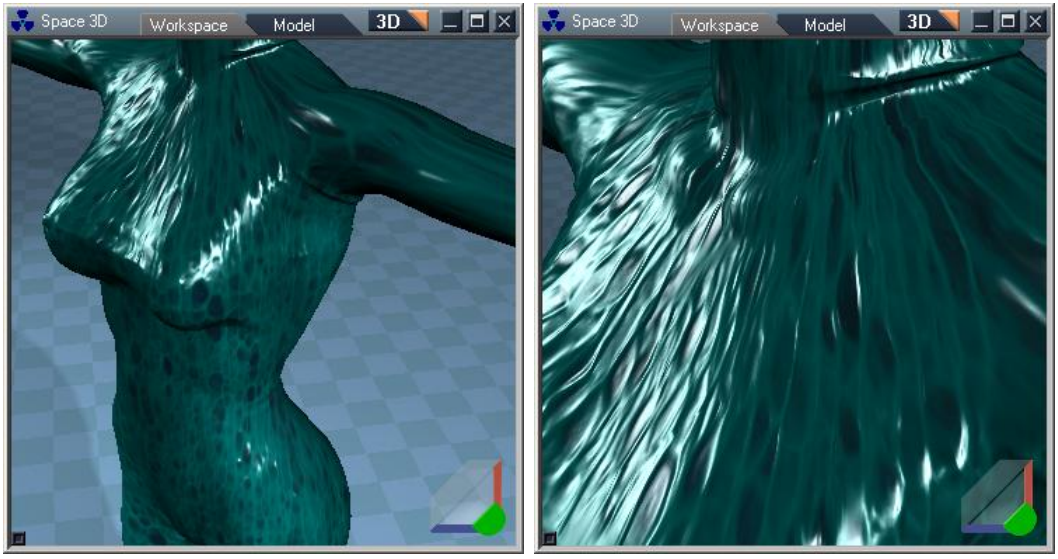
The following images use the simplified mesh with UV projection and the TextureBump material applied.



TextureBump material with its two aspects for the Diffuse and Normal textures

The material uses a diffuse color texture in the default tab and also a normal texture map which was generated in an external application in the bump tab of the material which simulates the appearance of geometry on the surface of the model.

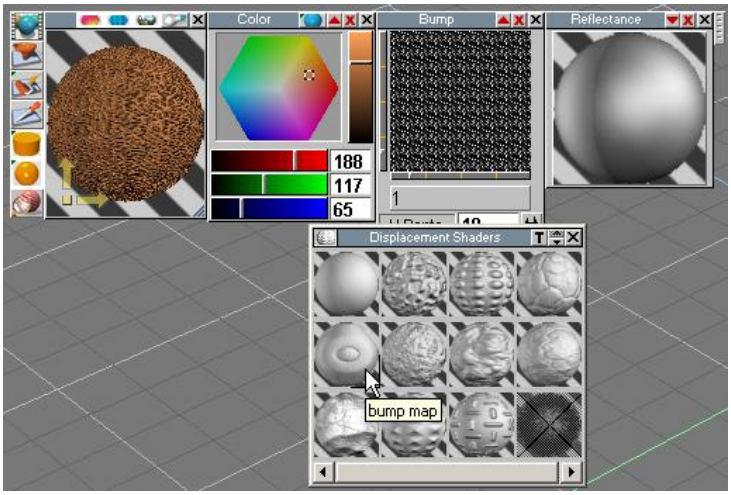
You can see how the lighting reacts to the models surface areas by casting shadows and introducing highlights on the simulated surface.



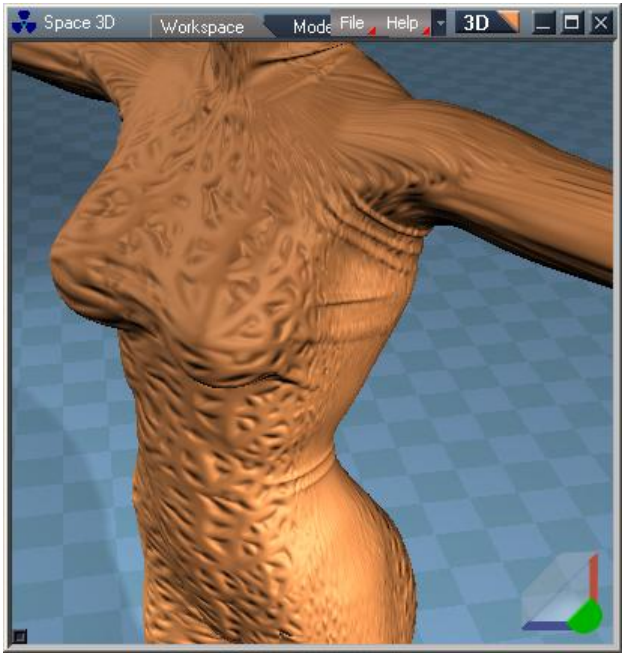
Same model as above but with the texture bump material applied and some texture maps.

If You don't have an application that can create Normal texture maps you can also use the modeler Material editor to load an image file in the bump texture channel and have a normal map texture generated by loading a bump map or other texture in the bump channel and then painting it on the model.

The below example uses just a plain color and one of the bitmaps loaded from the texture directory with a repeat of 10 in the U and V .

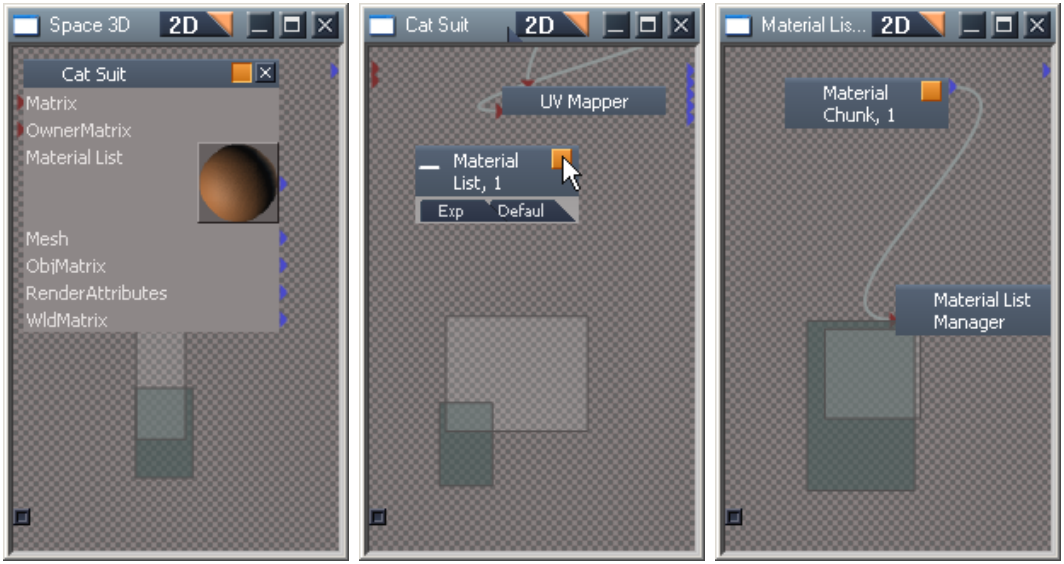


Modeler Material Editor and bump map location



Workspace view with the material applied

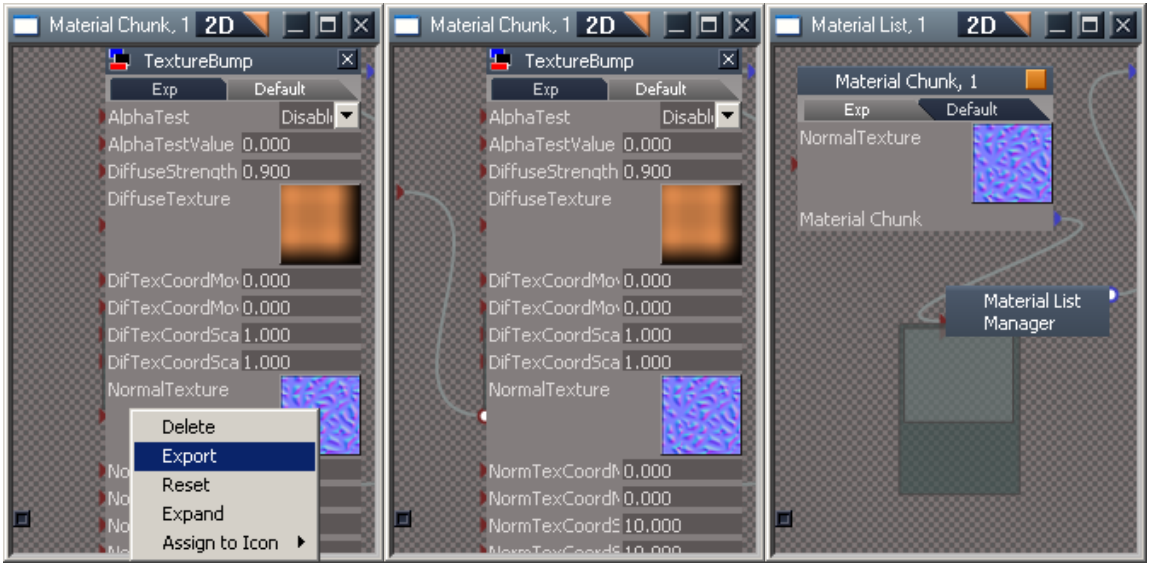
If you want to save the automatically generated normal texture then it can be extracted from the material and saved to a bitmap library by the following methods. Enter the object in the LE using the orange square in its title bar , locate and enter the Material List ,1 next enter the Material Chunk ,1.



Entering the objects layers to locate the material

Inside here you'll find the automatically generated texture map. you may have to expand the TextureBump Material object by clicking on the word default in its title bar.

The next step is to export this out one level so it can be accessed and inserted to a Bitmap Library. To export the bitmap r-click over it and choose export from the menu , if you're in developer mode you'll see a wire go to the outside of the container , then navigate out of the object using the triangle in the LE title bar.



Exporting the materials texture one level, NormalTexture exported, Material Chunk expanded and selected.

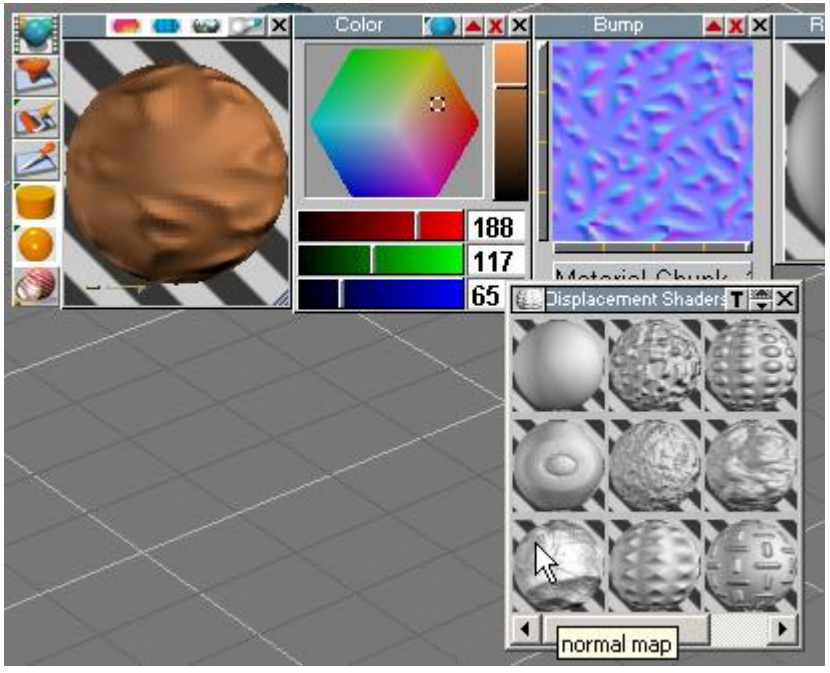
Next making sure the Material Chunk, 1 is selected open a bitmap library from the browser and r-click on an empty space in the library and choose an image format from the menu to insert and save the texture , I chose .dds , but the .bmp format will be suitable as well, you may also want to rename it to something other than the material chunk.

Note- We had to export the bitmap out one level from the material because in the material object only the top level bitmap would be considered when placing it to the bitmap library and the captured texture would be the Diffuse Texture one.

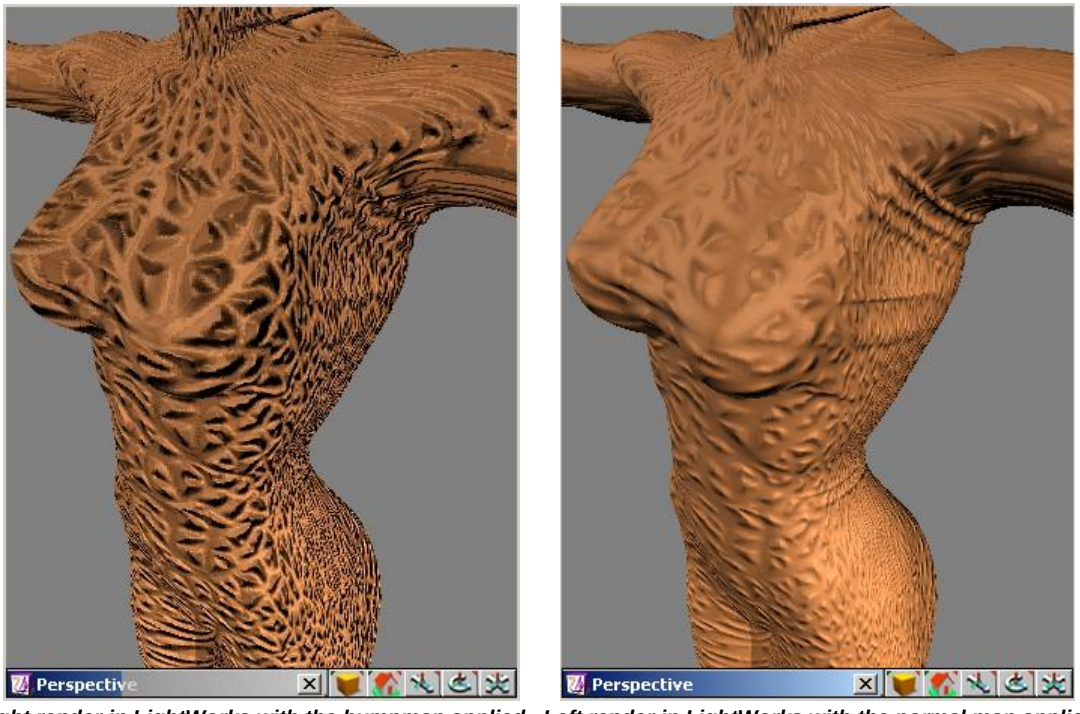
Inserting an objects texture to a bitmap library

Then you can locate and load the normal map texture in the modeler Material editor as a normal map to paint over your object. I set repeats to 10 here again , and when it's used in the offline render engines it gives a better detail than the bumpmap that was currently used .

If you saved your texture to the default libraries then they are located in the trueSpace75\tS\Rs Main Libraries directory .



Loading the normal map texture to the modeler Material Editor



Right render in LightWorks with the bumpmap applied . Left render in LightWorks with the normal map applied.

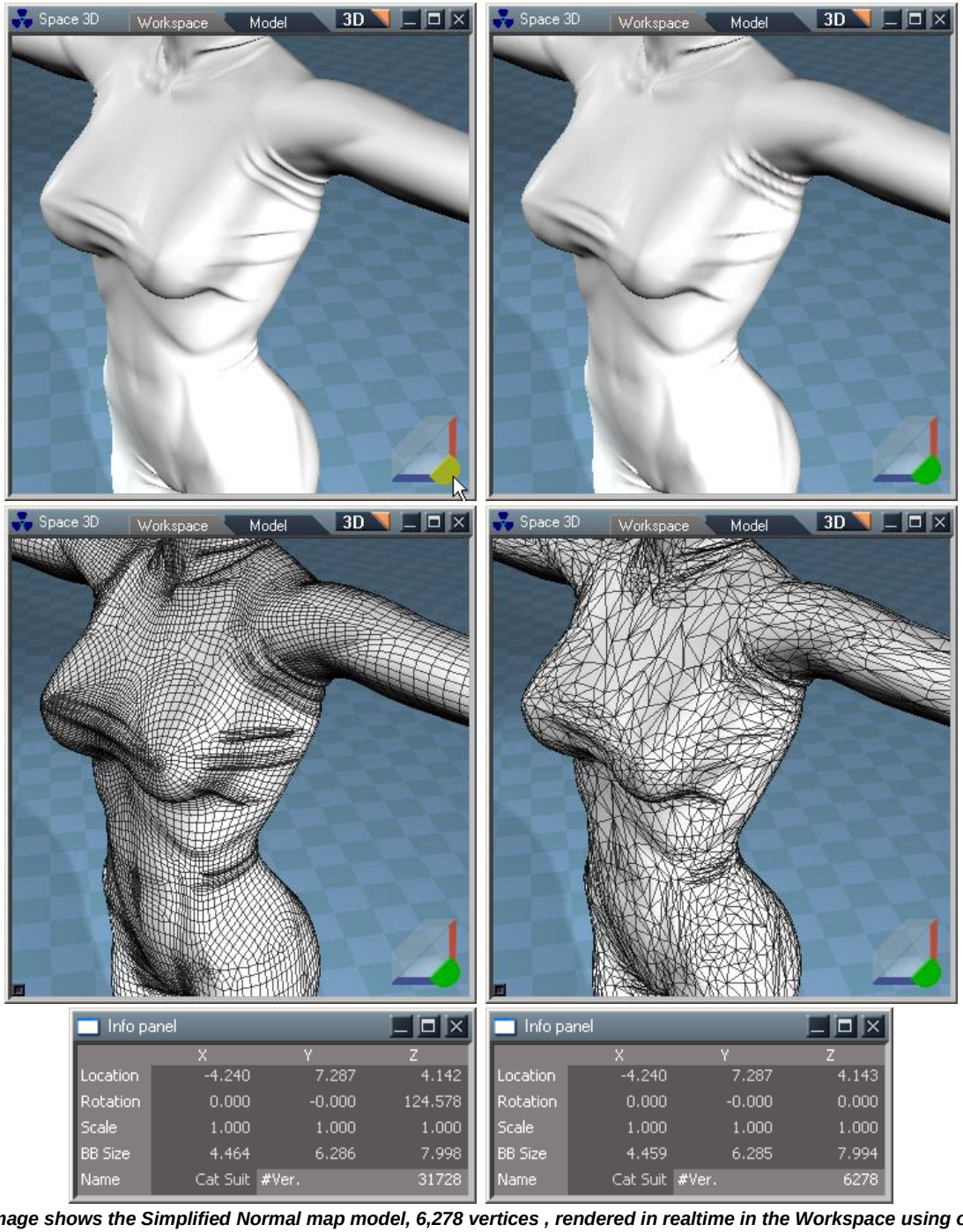
You can also finalize the mesh by using the Flatten History tool but be aware that doing this means that the mesh will no longer behave like a dynamic object , it will become a regular mesh and the panel sliders will no longer be functional.

Flatten History will remove unnecessary sub-objects

5.6.2 Generating Normal Maps

Create Normalmap

It is not possible to render geometry of unlimited detail in real-time. **Normal maps** can be used to add to the apparent detail of a mesh without adding to the actual geometry of that mesh. trueSpace includes a normal map generator to build normal maps from high-poly meshes that can be applied to low-poly meshes to give the appearance or more detail where there is none. The images below shows a high-poly mesh and a simplified mesh with a normal map applied. Unlike the simplify mesh tool used above the Normal Map creates the illusion of the geometry, this is more visible in the mesh by looking at the armpit wrinkles and comparing the difference between the simplified images above and the images below where the normal map tool has been used. Using the Normal Map tool as you can see below, the effect is quite similar when the original and the simplified normal mapped version are side by side.

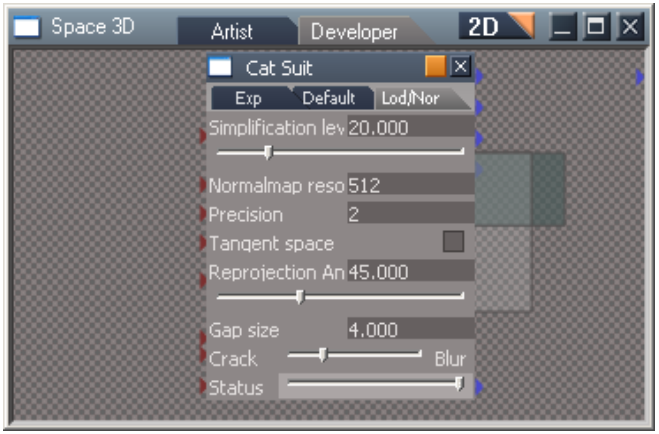


Right image shows the Simplified Normal map model, 6,278 vertices , rendered in realtime in the Workspace using one 512x512 generated normal map. Left image shows the original Cat Suit model, 31,728 Vertices.

Building a Normal Map

You can build a normal map by taking a high-polygon mesh and create both a simplified version of it and a corresponding normal map.
To reduce a high-poly mesh and create a normal-mapped low-poly mesh from it, you must first select the object, and then select the **Normalmap** tool, trueSpace responds by opening the Normal Mapper panel and simplifying the high-poly mesh according to the default settings of the tool.

Normal map tool.



Normal map panel.

Then it generates a corresponding normal map for the newly created low-poly mesh.
The original mesh will be kept inside of the new, normal-mapped object, and any modifications you make to it will be automatically updated on the generated object.LOD-Normal Map Panel Controls
You can change the settings on the LOD-Normal Map panel to affect the appearance of the generated object. Here you can alter the mesh simplification level, change normal map resolution, precision, and the method of normal map generation.

• **Simplification Level** determines how many triangles the generated mesh should have as a percent of those in the input mesh. For example, 20% means that output mesh will have 20% of the

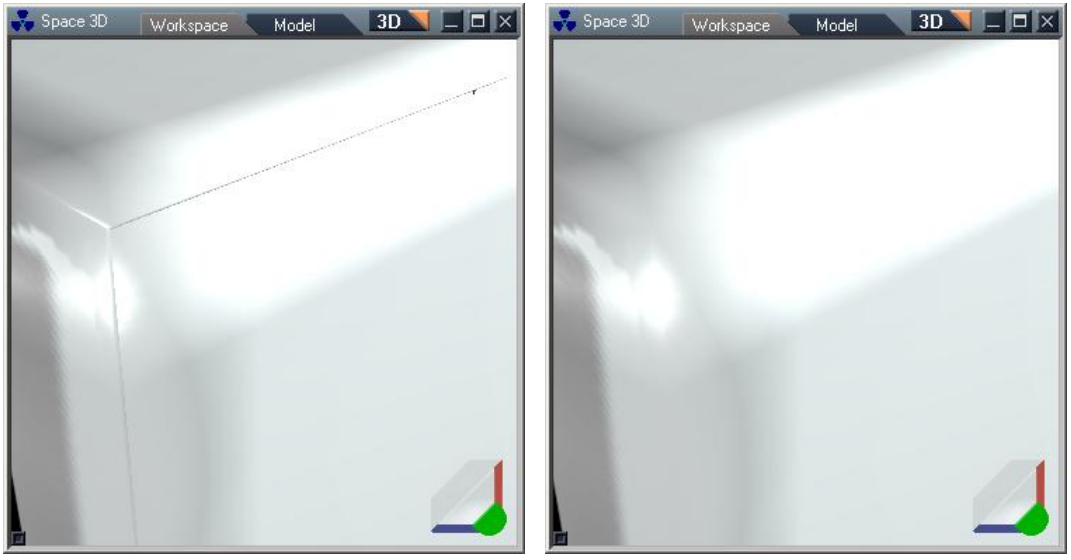
number of triangles found in the original mesh.

• **Normalmap Resolution** sets resolution of the generated normal map. A higher value means better quality, but more memory consumption.

• **Precision** controls number of samples used to create the normal map. A higher value means better quality, but it also significantly increases the time needed to generate the normal map. Lower values may result in dark dots appearing inside of the normal map. For testing, set this value low, then increase it after you are satisfied with the result. A precision level of between 3 and 4 should be sufficient for most final normal maps.

• **Reprojection Angle** controls automatic mesh unwrapping. Higher values will cause smaller texel density for steeper polygons, but will generally create a lower number of patches, and thus will produce better normal map allocation for the rest of the mesh. Smaller values, on the other hand, will lower this reprojection distortion, but more unwrap patches will be generated. Because there is some space needed between patches, this will cause poorer normal map allocation. 45-60 degrees should suffice for most models.

• **Gap size** is the space in texels that should be allocated between normal map patches. Setting this value too low will cause scratches in the rendered image due to MIPMapping. With higher values, smaller parts of the normal map will be allocated for visible pixels, causing the normal map to be blurred.

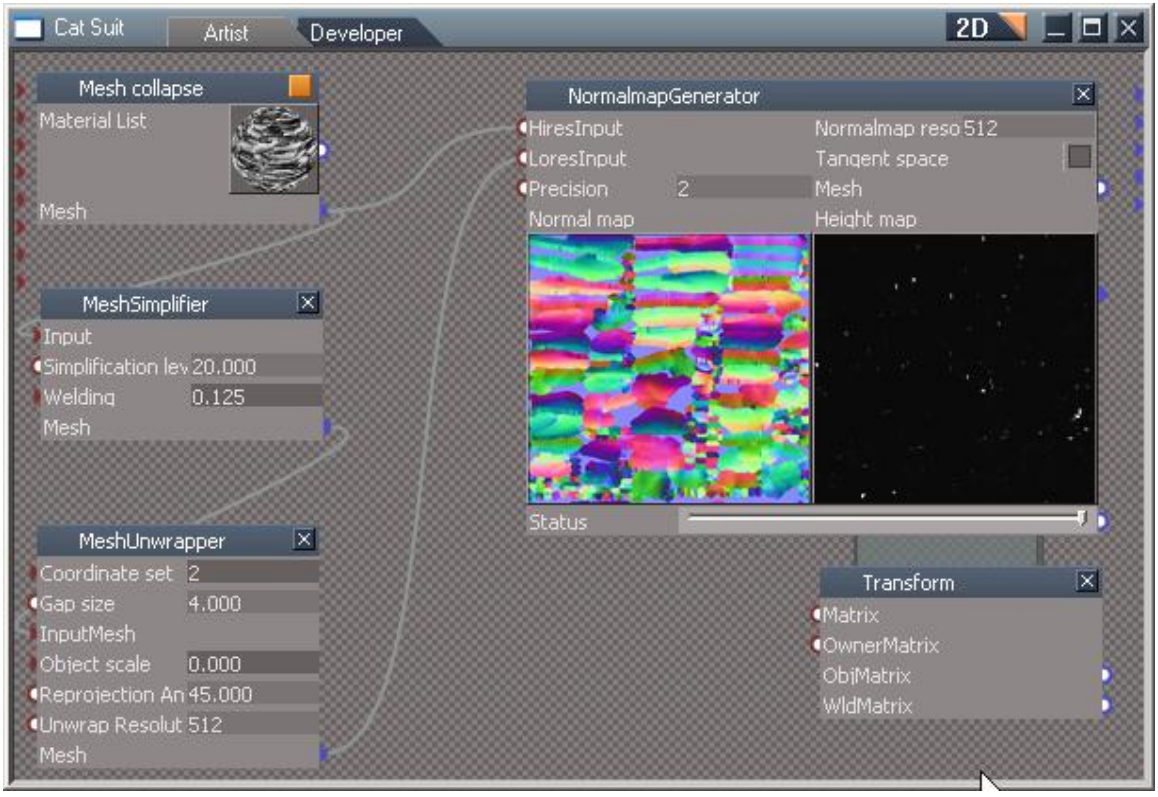


Left Gap size at 0 value shows scratches Right Gap Size at 4

• Checking the **Tangent space** checkbox will result in building a tangent space normal map. Object space normal maps are more precise because there is no need to interpolate tangent space matrices during estimation of normals, but their reuse is somehow limited. On the other hand, tangent space normals can be more reusable, but because tangent space axes are shortened during linear interpolation in graphics hardware, tangent space normal maps cannot compensate for these differences between spherical and linear interpolation, resulting in some artifacts on meshes with high curvature.

Simplified mesh without normal map. Normals in normal map would require being denormalized in order to compensate linear to spherical interpolation error. This causes boundary solver to diverge.

In this case, the correct solution provides only an object space normal map because it is not dependent on normal interpolation, and the boundary solver can compute the correct normal.
Other more advanced settings for tweaking the normal mapper can be modified directly in the Link Editor. These include the welding distance for vertex and edges welding during topology correction (found on the Mesh Simplifier object), coordinate set which is used for texture coordinate unwrapping, and object scale to manually control polygon surface to normal map mapping (both found on the Mesh Unwrapper object).



Advanced settings can be found and changed inside the object in the LE

The Simplify Mesh and Simplify Mesh and Create Normal Map tools are very powerful, particularly as they are dynamic – you can choose to change the level of simplification at any point, and your object will update accordingly.

However, to do this the tools must keep a copy of the high resolution mesh in memory, and while this reduces the demand on the real-time rendering and in moving around in the scene, it still increases the memory resources required overall.

If you are happy with the reduced version of the mesh and do not want to change the level of polygon reduction any further, it is possible to finalize the object with the reduced geometry to give you the benefit of the reduction of demand on the real-time engine, as well as the benefit of lower memory requirements for the object and scene.

You can use the Flatten History Tool to remove unnecessary parts and save the reduced version to the Workspace Libraries

Flatten History will remove unnecessary sub-objects

5.6.3 Limitations

Please note that both the **Mesh Simplifier** and **Normal Map Generator** tools handle DX (Player-originated) materials perfectly, but can simplify complex LW (Modeler-originated) materials for the final object.
E.g., when you apply the mesh simplifier tool to the Bowl object (can be found in Modeler's objects library), the original procedural wood material will be replaced by plain texture.