

Q3Map2 Shader Manual

0 Preface

- Credits
- Contact

1 Introduction

- What is a Shader?
- Shader Name & File Conventions
- :q3map Suffix
- Keyword Types
- Documenting With Comments
- Key Concepts

2 General Shader Keywords

- skyParms
- cull
- deformVertexes
- fogParms
- noPicMip
- noMipMaps
- polygonOffset
- portal
- sort

3 Q3Map2 Specific Shader Keywords

- q3map_alphaMod
- q3map_backShader
- q3map_backSplash
- q3map_baseShader
- q3map_bounce
- q3map_bounceScale
- q3map_clipModel
- q3map_cloneShader
- q3map_extraShader
- q3map_fadeAlpha
- q3map_fogDir
- q3map_forceMeta
- q3map_forceSunlight
- q3map_fur
- q3map_globalTexture
- q3map_indexed
- q3map_invert
- q3map_lightImage
- q3map_lightmapAxis
- q3map_lightmapBrightness
- q3map_lightmapFilterRadius
- q3map_lightmapGamma
- q3map_lightmapMergable
- q3map_lightmapSampleOffset
- q3map_lightmapSampleSize
- q3map_lightmapSize
- q3map_lightRGB
- q3map_lightStyle
- q3map_lightSubdivide
- q3map_noClip
- q3map_noFast
- q3map_noFog
- q3map_nonPlanar
- q3map_normalImage
- q3map_noTJunc
- q3map_noVertexShadows

q3map_offset
q3map_patchShadows
q3map_replicate
q3map_shadeAngle
q3map_skylight
q3map_splotchFix
q3map_styleMarker
q3map_styleMarker2
q3map_sun
q3map_sunExt
q3map_sunlight
q3map_surfacelight
q3map_surfaceModel
q3map_tcGen
q3map_tcMod
q3map_terrain
q3map_tessSize
q3map_textureSize
q3map_traceLight
q3map_vertexScale
q3map_vertexShadows
q3map_vlight

4 Q3Map2 Specific Surface Parameter Shader Keywords

surfaceparm alphashadow
surfaceparm antiportal
surfaceparm areaportal
surfaceparm botclip
surfaceparm clusterportal
surfaceparm detail
surfaceparm donotenter
surfaceparm dust
surfaceparm flesh
surfaceparm fog
surfaceparm hint
surfaceparm ladder
surfaceparm lava
surfaceparm lightfilter
surfaceparm lightgrid
surfaceparm metalsteps
surfaceparm monsterclip
surfaceparm nodamage
surfaceparm nodlight
surfaceparm nodraw
surfaceparm nodrop
surfaceparm noimpact
surfaceparm nomarks
surfaceparm nolightmap
surfaceparm nosteps
surfaceparm nonsolid
surfaceparm origin
surfaceparm playerclip
surfaceparm pointlight
surfaceparm skip
surfaceparm sky
surfaceparm slick
surfaceparm slime
surfaceparm structural
surfaceparm trans
surfaceparm water

5 Editor Specific Shader Keywords

	qer_editorimage
	qer_nocarve
	qer_trans
6 Stage Specific Shader Keywords	
	Texture Map Specification
	blendFunc
	rgbGen
	alphaGen
	tcGen
	tcMod
	depthFunc
	depthWrite
	detail
	alphaFunc
7 Quake 3 Engine Game Specific Shader Keywords	
	Return to Castle Wolfenstein
	Return to Castle Wolfenstein: Enemy Territory
	Raven
	Ritual
8 Shader Effects Creation Tips	
	Notes on Alpha Channels
	Troubleshooting Shaders
	Creating New Textures
Appendix A: Triggerable Shader Entities	

Appendix B: Terrain Entities (new improvements, lightmapped terrain, alphaMod dotproduct, etc)

Appendix C: Foghull

Appendix D: Fur

Appendix E: Celshading

Appendix F: Bumpmapping

Appendix G: Lightstyles

Appendix H: Lightmapped Mapmodels

Appendix I: Light Emitting Shaders

The Q3Map2 Edition of the Shader Manual

Q3Map2 is an updated version of the original Q3Map program used to compile .map files created in a level editor such as *Radiant* into .bsp files used by *Quake III* engine games. Since its introduction, ydnar has added so many new features and improved efficiency to the degree where Q3Map2 has become the standard compiling program for the level editing community as well as for many commercial games.

While still accurate, the original QeRadiant Shader Manual written by the staff at id Software did not contain shader keywords pertaining to the new Q3Map2, hence the need for this updated version of the manual including recent developments to the compiler.

This manual is based on the original manual with some minor reformatting, edits and error corrections, but the main difference is the inclusion of Q3Map2 specific shader keywords. Additionally, this version also includes an expanded appendix with information on selected Q3Map2 features as well as a new chapter for shader keywords used specifically with third party Quake III engine games.

Since Q3Map2 is being updated with additional features on a regular basis, this manual will also be updated to reflect any new shader keywords being introduced so check back for updates.

-Obsidian



Credits

Q3Map2 Shader Manual

Written by Obsidian and ydnar

The Q3Map2 Shader Manual is based on the *QeRadiant Shader Manual - Revision #12*. Other portions were adapted from the *Q3Map2 Handbook (Beta Edition)*, the *Q3Map2 Readme* document, and from various discussions on the forums/IRC channels. All works were consulted with permission from the authors.

QeRadiant Shader Manual, Revision #12

- Written by Paul Jaquays and Brian Hook.
- Additional material by John Carmack, Christian Antkow, Kevin Cloud and Adrian Carmack.
- Converted into a web friendly version by John Hutton.
- Appendix A by TTimo.

Q3Map2 Handbook (Beta Edition)

- Written by Jetscreamer, Shadowspawn and ydnar.

Q3Map2 Readme

- Written by ydnar

Special Thanks:

- Paul Jaquays for his support, feedback and for writing the original QeRadiant Shader Manual.
- ydnar for Q3Map2, thereby making level editing 2x more fun at 1/10 the compile time. Thanks for his technical expertise and help in writing this manual.

This is not an id Software product. Do not contact them for support.



Contact

Shader Manual Development

Feel free to e-mail me if you have any suggestions, comments or error corrections.

obsidianlordz@hotmail.com

(Soon to be obsidian@quake3world.com)

Q3Map2 Shader Manual at Shaderlab:

http://shaderlab.com/q3map2/shader_manual/

Forums

[Quake 3 World Level Editing and Modeling Forum](#)

[Splash Damage Q3Map2 Support Forum](#)

1. Introduction

The graphic engine for *Quake III Arena* has taken a step forward by putting much more direct control over the surface qualities of textures into the hands of designers and artists. In writing this manual, we have tried to define the concepts and tools that are used to modify textures in a way that, it is hoped, will be graspable by users who already have basic knowledge of computer graphics but are not necessarily computer programmers. It is not a tutorial, nor is it intended to be one.

The Manual for the Q3Radiant editor program contains a section called *Creating New Assets* that has the necessary information for setting up the files to create your own custom *Quake III Arena* shaders. You should follow a similar convention when creating your custom shaders. It is reproduced here for your convenience:

Quote: from Q3Radiant Editor Manual

Creating New Assets

If you are familiar with the required tools, creating new assets for use in Quake III Arena is not particularly difficult. As a rule, you should create new directories for each map with names different from the names used by id. If you are making a map that will be called "H4x0r_D00M", every directory containing new assets for that map should be titled H4x0r_D00M. This is to try and avoid asset directories overwriting each other as the editor and the game load in assets.

It is recommended that you study the scripts in this document and in the individual shader scripts. Pay careful attention to syntax and punctuation. This is where you are most likely to make mistakes.



What is a Shader?

Shaders are short text scripts that define the properties of a surface as it appears and functions in a game world (or compatible editing tool). By convention, the documents that contain these scripts usually has the same name as the texture set which contains the textures being modified (e.g; base, hell, castle, etc.). Several specific script documents have also been created to handle special cases, like liquids, sky and special effects. For Quake III Arena, shader scripts are located in [Quake3]/baseq3/scripts.

A Quake III Arena shader file consists of a series of surface attribute and rendering instructions formatted within braces ("{" and "}"). Below you can see a simple example of syntax and format for a single process, including the Q3Map2 keywords and "Surface Parameters", which follow the first bracket and a single bracketed "stage":

Script: Syntax and Format

textures/liquids/lava-example	//Shader Name
{	
deformVertexes wave sin 0 3 0 0.1	//General Shader Keywords
q3map_tessSize 64	//Q3Map2 Specific Shader Keywords
surfaceparm lava	//Q3Map2 Specific Surface Parameter Shader Keywords
qer_editorimage textures/common/lava.tga	//Editor Specific Shader Keywords
{	
map textures/common/lava.tga	//Stage Specific Shader Keywords
}	
}	

Shaders need to be referenced by the map editor, compiler (Q3Map2) and game engine - each of which uses a different part of the shader. In the above example shader, the different sections are listed on the right.



Shader Name & File Conventions

The first line is the shader name. It is used by the map editor, compiler, game engine, and models to reference the script and art files associated with the shader. Shader names can be up to 63 characters long. The names are often a mirror of a pathname to the texture image without the file extension (.tga or .jpg) or base dir (/quake3/baseq3 in our case), but they do not need to be.

Shaders that are only going to be referenced by the game code, not modeling tools, often are just a single word, like "projectionShadow" or "viewBlood".

Shaders that are used on character or other polygon mesh models need to mirror the art files, which allows the modelers to build with normal textures, then have the special effects show up when the model is loaded into the game.

Shaders that are placed on surfaces in the map editor commonly mirror an image file, but the "qer_editorimage" shader parameter can force the editor to use an arbitrary image for display.

Shader path names have a case sensitivity issue - on Windows, they aren't case sensitive, but on Unix they are. Try to always use lowercase for filenames, and always use forward slashes "/" for directory separators.

Following the shader name, is an opening bracket "{" designating the beginning of the shader contents, and a closing bracket "}" at the end of the shader. Notice that the contents of the above example shader are indented for better legibility.



:q3map Suffix

The *:q3map* suffix can be added to the end of the shader name of "compile-time" shaders. This means that shaders with names marked with *:q3map* will be used by the compiler only and will be ignored by the game engine.



Keyword Types

Only the shader name is case sensitive (use all lowercase). Shader keywords (aka. directives) are not, but by convention, you should use mixedCase (ie. *q3map_alphaMod*). Some of the shader keywords are order dependant, so it's good practice to keep all keyword of a similar type grouped together in the following order, from top-down:

General Shader Keywords

A shader may contain general shader keywords, which affects the shader's appearance globally (affects the entire shader), and changes the physical attributes of the surface that uses the shader. These attributes can affect the player. These parameters are executed at runtime, that is, in order to see the changes one does not need to re-compile the map.

Q3Map2 Specific Shader Keywords

The general shader keywords may be followed by Q3Map2 specific keywords. These keywords are used by the compiler to change the physical nature of the shader's surface and are also global.

Q3Map2 Specific Surface Parameter Shader Keywords

These are actually part of the Q3Map2 specific shader keywords that are flags telling the compiler about the physical nature of the shader's surface. These are also global.

Editor Specific Shader Keywords

Editor specific shader keywords change the appearance of the shader in the map editor only. It has no effect on the shader in game.

Stage Specific Shader Keywords

The shader may contain one or more "stages". Each stage consists of an opening and closing bracket along with the contents of the stage (indented). The stage will contain a texture map specification which specifies the source image, and various other stages specific shader keywords that modifies the appearance of the source image. Stage specific keywords are processed by the renderer. They are appearance changes only and have no effect on game play or game mechanics. Changes to any of these attributes will take effect as soon as the game goes to another level or vid_restarts (type command "/vid_restart" in the game console).



Documenting With Comments

It is often useful to document parts of shaders, especially for testing purposes or when working with a development team of other designers. Comments added to shaders are ignored completely, making it useful for adding notes, or for temporarily removing keywords for testing.

Shaders use the same double forward slashes "//" convention common with many programming languages for commenting. All text after the "//" on the same line are ignored from the shader. See various shaders for examples.



Key Concepts

Ideally, a designer or artist who is manipulating textures with shader files has a basic understanding of wave forms and knows about mixing colored light (high school physics sort of stuff). If not, there are some concepts you need to have a grasp on to make shaders work for you.

Surface Effects vs. Content Effects vs. Deformation Effects

Shaders not only modify the visible aspect of textures on a geometry brush, curve, patch or mesh model, but they can also have an effect on both the content, "shape," and apparent movement of those things. A surface effect does nothing to modify the shape or content of the brush. Surface effects include glows, transparencies and rgb (red, green, blue) value changes. Content shaders affect the way the brush operates in the game world. Examples include water, fog, nonsolid, and structural. Deformation effects change the actual shape of the affected brush or curve, and may make it appear to move.

Power Has a Price

The shader script gives the designer, artist and programmer a great deal of easily accessible power over the appearance of and potential special effects that may be applied to surfaces in the game world. But it is power that comes with a price tag attached, and the cost is measured in performance speed. Each shader phase that affects the appearance of a texture causes the *Quake III* engine to make another processing pass and redraw the world. Think of it as if you were adding all the shader-affected triangles to the total r_speed count for each stage in the shader script. A shader-manipulated texture that is seen through another shader-manipulated texture (e.g. a light in fog) has the effect of *adding* the total number of passes together for the affected triangles. A light that required two passes seen through a fog that requires one pass will be treated as having to redraw that part of the world three times.

RGB Color

RGB means "Red, Green, Blue". Mixing red, green and blue light in differing intensities creates the colors in computers and television monitors. This is called *additive color* (as opposed to the mixing of pigments in paint or colored ink in the printing process, which is *subtractive color*). In *Quake III Arena* and most higher-end computer art programs (and the color selector in Windows), the intensities of the individual Red, Green and Blue components are expressed as number values. When mixed together on a screen, number values of equal intensity in each component color create a completely neutral (gray) color. The lower the number value (towards 0), the darker the shade. The higher the value, the lighter the shade or the more saturated the color until it reaches a maximum value of 255 (in the art programs). All colors possible on the computer can be expressed as a formula of three numbers. The value for complete black is 0 0 0. The value for complete white is 255 255 255. However, the *Quake III Arena* graphics engine requires that the color range be "normalized" into a range between 0.0 and 1.0.

Normalization: a Scale of 0 to 1

The mathematics in *Quake III Arena* use a scale of 0.0 to 1.0 instead of 0 to 255. Most computer art programs that can express RGB values as numbers use the 0 to 255 scale. To convert numbers, divide each of the art program's values for the component colors by 255. The resulting three values are your *Quake III Arena* formula for that color component. The same holds true for texture coordinates.

Texture Sizes

TGA texture files are measured in pixels (picture elements). Textures are measured in powers of 2, with 16 x16 pixels being the smallest (typically) texture in use. Most will be larger. Textures need not be square, so long as both dimensions are powers of 2. Examples include: 32x256, 16x32, 128x16.

Color Math

In *Quake III Arena*, colors are changed by mathematical equations worked on the textures by way of the scripts or "programlets" in the shader file. An equation that adds to, or multiplies the number values in a texture causes it to become darker. Equations that subtract from or modulate number values in a texture cause it to become lighter. Either equation can change the hue and saturation of a color.

Measurements

The measurements used in the shaders are in either game units, color units, or texture units.

Game unit: A game unit is used by deformations to specify sizes relative to the world. Game units are the same scale we have had since way back in the *Wolfenstein* days - 8 units equals one foot. The default texture scale used by the Radiant map editor results in two texels for each game unit, but that can be freely changed.

Color units: Colors scale the values generated by the texture units to produce lighting effects. A value of 0.0 will be completely black, and a value of 1.0 will leave the texture unchanged. Colors are sometimes specified with a single value to be used across all red, green, and blue channels, or sometimes as separate values for each channel.

Texture units: This is the normalized (see above) dimensions of the original texture image (or a previously modified texture at a given stage in the shader pipeline). A full texture, regardless of its original size in texels, has a normalized measurement of 1.0 x 1.0. For normal repeating textures, it is possible to have value greater than 1.0 or less than 0.0, resulting in repeating of the texture. The coordinates are usually assigned by the level editor or modeling tools, but you still need to be aware of this for scrolling or turbulent movement of the texture at runtime.

Waveform Functions

Many of the shader functions use waveforms to modulate measurements over time. Where appropriate, additional information is provided with wave modulated keyword functions to describe the effect of a particular waveform on that process. Currently there are five waveforms in use in Q3A shaders:

Sin: Sin stands for sine wave, a regular smoothly flowing wave function ranging from -1 to 1.

Triangle: Triangle is a wave with a sharp ascent and a sharp decay, ranging from 0 to 1. It will make choppy looking wave forms.

Square: A squarewave simply switches from -1 to 1 with no in-between.

Sawtooth: In the sawtooth wave, the ascent is like a triangle wave from 0 to 1, but the decay cuts off sharply back to 0.

Inversesawtooth: This is the reverse of the sawtooth... instant ascent to the peak value (1), then a triangle wave descent to the valley value (0). The phase on this goes from 1.0 to 0.0 instead of 0.0 to 1.0. This wave is particularly useful for additive cross-fades.

Waveforms all have the following properties:

base: Where the wave form begins. Amplitude is measured from this base value.

amplitude: This is the height of the wave created, measured from the base. You will probably need to test and tweak this value to get it correct for each new shader stage. The greater the amplitude, the higher the wave peaks and the deeper the valleys.

phase: This is a normalized value between 0.0 and 1.0. Changing phase to a non-zero value affects the point on the wave at which the wave form initially begins to be plotted. Example: In Sin or Triangle wave, a phase of 0.25 means it begins one fourth (25%) of the way along the curve, or more simply put, it begins at the peak of the wave. A phase of 0.5 would begin at the point the wave re-crosses the base line. A phase of 0.75 would be at the lowest point of the valley. If only one wave form is being used in a shader, a phase shift will probably not be noticed and phase should have a value of zero (0). However, including two or more stages of the same process in a single shader, but with the phases shifted can be used to create interesting visual effects. Example: using rgbGen in two stages with different colors and a 0.5 difference in phase would cause the manipulated texture to modulate between two distinct colors. Phase changes can also be used when you have two uses of the same effect near each other, and you don't want them to be synchronized. You would write a separate shader for each, changing only the phase value.

freq: Frequency. This value is expressed as repetitions or cycles of the wave per second. A value of 1 would cycle once per second. A value of 10 would cycle 10 times per second. A value of 0.1 would cycle once every 10 seconds.

2. General Shader Keywords

IMPORTANT NOTES: Once again, be aware that some of the shader commands may be order dependent, so it's good practice to place all global shader commands (keywords defined in this section) at the very beginning of the shader and to place shader stages at the end (see various examples).

These Keywords are global to a shader and affect all stages. They are also ignored by Q3Map2.



skyParms *farbox cloudheight nearbox*

Specifies how to use the surface as a sky, including an optional far box (stars, moon, etc), optional cloud layers with any shader attributes, and an optional near box (mountains in front of the clouds, etc).

farbox : Specifies a set of files to use as an environment box behind all cloudlayers. Specify "-" for no farbox, or a file base name. A base name of "env/test" would look for files "env/test_rt.tga", "env/test_lf.tga", "env/test_ft.tga", "env/test_bk.tga", "env/test_up.tga", "env/test_dn.tga" to use as the right / left / front / back / up / down sides.

cloudheight : Controls apparent curvature of the cloud layers - lower numbers mean more curvature (and thus more distortion at the horizons). Higher height values create "flatter" skies with less horizon distortion. Think of height as the radius of a sphere on which the clouds are mapped. Good ranges are 64 to 256. The default value is 128.

nearbox : Specified as farbox, to be alpha blended ontop of the clouds. This has not been tested in a long time, so it probably doesn't actually work. Set to "-" to ignore.

Design Notes:

- If you are making a map where the sky is seen by looking up most of the time, use a lower cloudheight value. Under those circumstances the tighter curve looks more dynamic. If you are making a map where the sky is seen by looking out windows most of the time or has a map area that is open to the sky on one or more sides, use a higher height to make the clouds seem more natural.
- It is possible to create a sky with up to 8 cloudlayers, but that also means 8 processing passes and a potentially large processing hit.
- Be aware that the skybox does not wrap around the entire world. The "floor" or bottom face of the skybox is not drawn by the game. If a player in the game can see that face, they will see the "hall of mirrors" effect.
- There's a bug in *Quake 3* (but fixed in *Enemy Territory*) that causes a shader vertex overflow if more than two cloud layers are used in maps with a lot of visible sky. To compensate, either reduce the amount of visible sky or limit the shader to two cloud layers.

Q3Map2 sky shaders work differently from the original and contain a number of improvements in terms of efficiency and visually. The example given below is an original *Quake III Arena* sky shader. While it is still operational, it is a little outdated and is being kept here for legacy purposes only. It is recommended that you take advantage of the new features of Q3Map2 skies by consulting [Appendix I: Light Emitting Shaders > Skies](#).

Script: Original Quake III Arena sky shader

```
textures/skies/xtoxicsky_dm9
{
    qer_editorimage textures/skies/toxicssky.tga
    surfaceparm noimpact
    surfaceparm nolightmap
    q3map_globaltexture
    q3map_lightsubdivide 256
    q3map_surfacelight 400
    surfaceparm sky
    q3map_sun 1 1 0.5 150 30 60
    skyparms full 512 -
    {
        map textures/skies/inteldimclouds.tga
        tcMod scale 3 2
        tcMod scroll 0.1 0.1
    }
    {
        map textures/skies/intelredclouds.tga
        blendFunc add
        tcMod scale 3 3
        tcMod scroll 0.05 0.05
    }
}
```



cull side

Every surface of a polygon has two sides, a front and a back. Typically, we only see the front or "out" side. For example, a solid block you only show the front side. In many applications we see both. For example, in water, you can see both front and a back. The same is true for things like grates and screens.

To "cull" means to remove. The value parameter determines the type of face culling to apply. The default value is cull *back* if this keyword is not specified. However for items that should be inverted then the value *front* should be used. To disable culling, the value *disable* or *none* should be used. Only one cull instruction can be set for the shader.

front : The front or "outside" of the polygon is not drawn in the world. It is used if the keyword "*cull*" appears in the content instructions without a *side* value.

back : Cull back removes the back or "inside" of a polygon from being drawn in the world.

disable or **none** : Neither side of the polygon is removed. Both sides are drawn in the game. Very useful for making panels or barriers that have no depth, such as grates, screens, metal wire fences and so on and for liquid volumes that the player can see from within. Also used for energy fields, sprites, and weapon effects (e.g. plasma).

Design Notes

For things like grates and screens, put the texture with the cull none property on one face only. On the other faces, use a non-drawing texture.



deformVertexes *type* ---

This function performs a general deformation on the surface's vertexes, changing the actual shape of the surface before drawing the shader passes. You can stack multiple deformVertexes commands to modify positions in more complex ways, making an object move in two dimensions, for instance. There are 6 possible values for the *type* parameter, each of which will be described in more detail: *wave*, *normal*, *bulge*, *move*, *autosprite*, *autosprite2*. Depending on which of the 6 *type* parameters are used, different additional parameters will need to be used, including the generalized waveform functions (see [Chapter 1: Key Concepts](#))

Design Notes:

The div and amplitude parameters, when used in conjunction with liquid volumes like water should take into consideration how much the water will be moving. A large ocean area would have massive swells (big div values) that rose and fell dramatically (big amplitude values). While a small, quiet pool may move very little.

deformVertexes *wave div func base amplitude phase freq*

Designed for water surfaces, modifying the values differently at each point. The *div* parameter is used to control the wave "spread" - a value equal to the [q3map tessSize](#) of the surface is a good default value. It accepts the standard wave functions *sin*, *triangle*, *square*, *sawtooth* or *inversesawtooth*.

deformVertexes *normal div func base amplitude freq*

This deformation affects the normals of a vertex without actually moving it, which will effect later shader options like lighting and especially environment mapping. If the shader stages don't use normals in any of their calculations, there will be no visible effect. The *div* parameter is used to control the wave "spread" - a value equal to the [q3map tessSize](#) of the surface is a good default value. Good values for amplitude ranges from 0.1 to 0.5 while values of 1.0 to 4.0 are good for frequency.

Design Notes:

Putting values of 0.1 to 0.5 in Amplitude and 1.0 to 4.0 in the Frequency can produce some satisfying results. Some things that have been done with it: A small fluttering bat, falling leaves, rain, flags.

deformVertexes *bulge bulgeS bulgeT bulgeSpeed*

This forces a bulge to move along the given s and t directions. Designed for use on curved pipes. The *bulgeS* and *bulgeT* parameters is the amount of bulge displacement measured in game units. *bulgeSpeed* is the number of seconds it takes for the bulge to complete a single cycle.

deformVertexes *move x y z func base amplitude phase freq*

The *move* parameter is used to make a brush, curve patch or model appear to move together as a unit. The x y z values are the distance and direction in game units the object appears to move relative to it's point of origin in the map. The *func base amplitude phase freq* values are the same as found in other waveform manipulations.

The product of the function modifies the values x, y, and z. Therefore, if you have an amplitude of 5 and an x value of 2, the object will travel 10 units from its point of origin along the x axis. This results in a total of 20 units of motion along the x axis, since the amplitude is the variation both above and below the base.

It must be noted that an object made with this shader does not actually change position, it only appears to.

Design Notes:

If an object is made up of surfaces with different shaders, all must have matching *deformVertexes move* values or the object will appear to tear itself apart.

deformVertexes *autosprite*

This function can be used to make any given triangle quad (pair of triangles that form a square rectangle) automatically behave like a sprite without having to make it a separate entity. This means that the "sprite" on which the texture is placed will rotate to always appear at right angles to the player's view as a sprite would. Any four-sided brush side, flat patch, or pair of triangles in a model can have the *autosprite* effect on it. The brush face containing a texture with this shader keyword must be square.

deformVertexes *autosprite2*

Is a slightly modified "sprite" that only rotates around the middle of its longest axis. This allows you to make a pillar of fire that you can walk around, or an energy beam stretched across the room.



fogParms (*r g b*) *opacity*

Note: You must also specify "[*surfaceparm fog*](#)" to cause Q3Map2 to identify the surfaces inside the volume. Fogparms only describes how to render the fog on the surfaces.

r g b : These are normalized values. A good computer art program should give you the RGB values for a color. To obtain the values that define fog color for Quake III Arena, divide the desired color's red, green and blue values by 255 to obtain three normalized numbers within the 0.0 to 1.0 range.

opacity : This is the distance, in game units, until the fog becomes totally opaque, as measured from the point of view of the observer. By making the height of the fog brush shorter than the distance to opaque, the apparent density of the fog can be reduced (because it never reaches the depth at which full opacity occurs).

- The fog volume can only have one surface visible (from outside the fog).
- Fog must be made of one brush. It cannot be made of adjacent brushes.
- Fog brushes must be axial. This means that only square or rectangular brushes may contain fog, and those must have their edges drawn along the axes of the map grid (all 90 degree angles).

Design Notes:

- If a water texture contains a fog parameter, it must be treated as if it were a fog texture when in use.
- If a room is to be filled completely with a fog volume, it can only be entered through one surface (and still have the fog function correctly).
- Additional shader passes may be placed on a fog brush, as with other brushes.



noPicMip

This causes the texture to ignore user-set values for the *r_picmip* cvar command. The image will always be high resolution. Example: Used to keep images and text in the heads up display from blurring when user optimizes the game graphics.



noMipMaps

This implies *noPicMip*, but also prevents the generation of any lower resolution mipmaps for use by the 3D card. This will cause the texture to alias when it gets smaller, but there are some cases where you would rather have this than a blurry image. Sometimes thin slivers of triangles force things to very low mipmap levels, which leave a few constant pixels on otherwise scrolling special effects.



polygonOffset

Surfaces rendered with the *polygonOffset* keyword are rendered slightly off the polygon's surface. This is typically used for wall markings and "decals." The distance between the offset and the polygon is fixed. It is not a variable in Quake III Arena.

Design Notes:

- Use this for wall or floor markings, particularly for direction arrows for team games. Texture the brush with the decal shader on one face and the other faces with a nodraw shader. Then place the brush flush with the surface of the wall or floor.
- When using a *_decal* entity for texture projection, *polygonOffset* must be used to prevent Z-fighting. If you experience problems with depth sorting, try using [sort 6](#).



portal

Specifies that this texture is the surface for a portal or mirror. In the game map, a portal entity must be placed directly in front of the texture (within 64 game units). All this does is set "*sort portal*", so it isn't needed if you specify that explicitly.



sort value

Use this keyword to fine-tune the depth sorting of shaders as they are compared against other shaders in the game world. The basic concept is that if there is a question or a problem with shaders drawing in the wrong order against each other, this allows the designer to create a hierarchy of which shader draws in what order.

The default behavior is to put all blended shaders in sort "additive" and all other shaders in sort "opaque", so you only need to specify this when you are trying to work around a sorting problem with multiple transparent surfaces in a scene.

The value here can be either a numerical value or one of the keywords in the following list (listed in order of ascending priority):

portal (1): This surface is a portal, it draws over every other shader seen inside the portal, but before anything in the main view.

Sky (2): Typically, the sky is the farthest surface in the game world. Drawing this after other opaque surfaces can be an optimization on some cards. This currently has the wrong value for this purpose, so it doesn't do much of anything.

Opaque (3): This surface is opaque (rarely needed since this is the default with no *blendfunc*)

Banner (6): Transparent, but very close to walls.

Underwater (8): Draw behind normal transparent surfaces.

Additive (9): Normal transparent surface (default for shaders with *blendfunc*'s)

Nearest (16): This shader should always sort closest to the viewer, e.g. muzzle flashes and blend blobs.

3. Q3Map2 Specific Shader Keywords

These keywords change the physical nature of the textures and the brushes that are marked with them. Changing any of these values will require the map to be re-compiled. These are global and affect the entire shader.



q3map_alphaMod func ---

This is used for special blending effects on shaders by altering the amount of blending falloff depending on specific surface properties such as the surface's normal axis or the vertexes contained within its volume. *alphaMod* operations are applied to an object's vertexes so the *rgbGen vertex* directive is required for each affected stage.

q3map_alphaMod dotproduct (X Y Z)

It is used to blend textures using *alphaFunc* or *blendFunc* in the shader's second pass, with the falloff depending on the surface's normal axis. This is achieved by doing a vector dot product of the specified normalized vector value (X Y Z) and the vertex normal

which yields the amount of blending. The dot product operation multiplies each element of one vector against the corresponding elements of a second vector, then adds them. Examples:

```
( 0 0 1 ) dp ( 0 0 1 ) = 0 * 0 + 0 * 0 + 1 * 1 = 1
( 0 0 1 ) dp ( 0 0 0.5 ) = 0 * 0 + 0 * 0 + 1 * 0.5 = 0.5
( 0.5 0.5 1 ) dp ( 0 0.5 0.5 ) = 0.5 * 0 + 0.5 * 0.5 + 1 * 0.5 = 0.75
```

q3map_alphaMod dotproduct2 (X Y Z)

This works in a similar way to *dotproduct* except it exaggerates the differences in vertex normals by squaring the final dot product value. With the same values as the above example, *dotproduct2* would give the following:

```
[ ( 0 0 1 ) dp ( 0 0 1 ) ] ^2 = ( 0 * 0 + 0 * 0 + 1 * 1 ) ^2 = 1
[ ( 0 0 1 ) dp ( 0 0 0.5 ) ] ^2 = ( 0 * 0 + 0 * 0 + 1 * 0.5 ) ^2 = 0.25
[ ( 0.5 0.5 1 ) dp ( 0 0.5 0.5 ) ] ^2 = ( 0.5 * 0 + 0.5 * 0.5 + 1 * 0.5 ) ^2 = 0.5625
```

Script: *q3map_dotproduct on terrain*

```
textures/shaderlab_vector_1/rock_1_z_lodterrain
{
    //Used for radiosity lighting
    q3map_lightImage textures/shaderlab_vector_1/rock_1.tga

    q3map_nonplanar
    q3map_shadeAngle 179
    q3map_tcGen ivector ( 512 0 0 ) ( 0 512 0 )
    q3map_tcMod rotate 33
    q3map_lightmapAxis z

    // this means dot product squared, for faster falloff between vertical and horizontal planes
    q3map_alphaMod dotproduct2 ( 0 0 0.95 )

    surfaceparm nonsolid
    surfaceparm pointlight

    {
        map textures/shaderlab_vector_1/rock_1.tga
        rgbGen vertex
    }
    {
        map textures/slterra/sand_1.tga
        blendFunc GL_SRC_ALPHA GL_ONE_MINUS_SRC_ALPHA
        rgbGen vertex
    }
}
```

Design Notes:

Typical examples of use include snow covering the top faces of objects, or terrain with grass growing on horizontal planes blending into rocky cliffs on near vertical surfaces. It is an excellent way of automatically creating realistic alpha-blended terrain without the complicated steps in setting up an alpha map.

q3map_alphaMod scale N.N

Used in conjunction with *q3map_alphaMod volume*. Scales the vertex alpha by N.N.

q3map_alphaMod set N.N

Used in conjunction with *q3map_alphaMod volume*. Sets the vertex alpha (regardless of any previous alpha values) to N.N * 255.

q3map_alphaMod volume

This was created as a way to explicitly set the amount of vertex alpha-blending by altering the alpha values of vertexes contained within a brush volume marked with this shader directive. Applies all other q3map_alphaMod directives to each vertex inside a brush textured with this shader, allowing large faded scrolling fire shaders, waterfalls, marquees, explicit dotProduct terrain blending control, etc.

Design Notes:

This is usually used in special alphaMod volume "common" shaders for use within the editor only. A brush textured with the alphaMod volume shader is used to overlap the vertexes of another brush using an alpha-blended shader, altering the vertex alpha values. Worldspawn alphaMod volume brushes will affect all surfaces. You can func_group an alphaMod volume brush to affect only that entity.



q3map_backShader *shadername*

This allows a brush to use a different shader when you are inside it looking out. By way of example, this would allow a water brush (or other) surfaces to have a different [sort](#) order or appearance when seen from the inside. [q3map_backShader](#) only works on brush faces. For this reason, it is deprecated in favor of using [q3map_cloneShader](#) where the target shader contains [q3map_invert](#).



q3map_backSplash *percentage distance*

A surface light is lit by itself, often causing areas of higher light intensity than other areas. [q3map_backSplash](#) moves the light source away from the surface of the shader, allowing it to create smoother lighting over the face. By default, all shaders are assigned backsplash values, 0.05 for the percentage, 23 units for the distance.

percentage : Specifies the intensity percentage of the light generated by [q3map_surfacelight](#) to be redirected back at the surface. Use a value of 0 or a negative value to disable back splash lights.

distance : Distance of the back splash lights from the surface.



q3map_baseShader *shadername*

Allows shaders to be subclassed (Q3Map2 relevant portions only, such as surfaceparms, lighting, texture projection, etc). Subclassed shaders can reference the base shader by referring to the base shader's name. In order for [q3map_baseShader](#) to work correctly, the base shader must be specified before any shaders that subclass it. Some *EasyGen* terrain templates incorrectly specified the base shader after the terrain shaders that depended on it, resulting in some interesting errors.



q3map_bounce *N.N*

Deprecated! Use [q3map_bounceScale](#) instead.



q3map_bounceScale *N.N*

Use a number between 0 and 1.0 (or higher), to scale the amount of light reflected in radiosity passes. You can oversaturate it by using a number higher than 1.0, but this can lead to excessive compile times. Using 90 would probably make things positively glacial. 1.0 is a default, fudged number that looked OK with the maps that were tested. Tweaking it to 1.5 or 2.0 won't hurt anything, per se, but it does give you finer control over how each shader re-emits light. The poorly worded *q3map_bounce* has been renamed to *q3map_bounceScale*. While its use has been deprecated, *q3map_bounce* still works.



q3map_clipModel

Automatically clips misc_model entities for player and weapon collision. This should only be used on large models such as terrain (not small decorative models - manually clip those). The shader's surfaceparms are inherited by the magic clip brush, so if you have *surfaceparm nonsolid* in your model's shader that uses *q3map_clipModel*, then the brush will also be non-solid. This can also be set on a per model basis with spawnflags 2.



q3map_cloneShader *shadername*

A shader with this keyword will inherit the target shader's properties and appearance. Be careful, this can lead to an infinite loop if a cloning shader references another cloning shader or itself.



q3map_extraShader

Does not exist!



q3map_fadeAlpha *N*

Does not exist!



q3map_fogDir *angle*

Specifies the direction a fog shader fades from transparent to opaque.



q3map_forceMeta

Forces model (MD3, ASE, etc.) surfaces to be broken down into their component triangles like brush faces and passed through the meta code on a per shader basis. This is required for lightmapped models. Setting spawnflags 4 on a misc_model will set *q3map_forceMeta* on all its surfaces.



q3map_forceSunlight

Obsolete!

By default, no sunlight is cast on vertex-lit .md3 models or vertex-lit terrain. Using this option, sunlight (overbright bits created by the q3map_sun option) will be cast on these surfaces. *q3map_forceSunlight* is now obsolete since suns are now first class light sources.



q3map_fur layers offset fade

This is used for generating fur over a surface. This is typically used in conjunction with [q3map_cloneShader](#) in the surface (parent) shader and references the fur shader as the clone. A possible application of this is to create grass on alphablended terrain. Keep in mind that the use of a fur shader may cause a large hit to performance but when used sparingly, it can produce some interesting effects. (See [Appendix D: Fur](#))

layers : This specifies the number of desired replicated fur layers generated.

offset : The distance (in game units) between subsequent layers.

fade : A normalized value indicating the fade falloff between subsequent layers.



q3map_globalTexture

Use this shader in the global keyword commands whenever the *tcMod scale* function is used in one of the later render stages. Many problems with getting shader effects to work across multiple adjacent brushes are a result of the way Q3Map2 optimizes texture precision. This option resolves that, but at the expense of some precision of the textures when they are far away from the origin of the map.



q3map_indexed

This is used for explicit terrain-style indexed mapping. It instructs Q3Map2 to look at the func_group terrain entity's `_indexmap` key for an image to pull index values from, and then to construct a shader name with the root based on the `_shader` key's value.



q3map_invert

Inverts a surface normal. Works on brush faces, models and patches. Used in celshading to achieve the inverted backfacing hull.



q3map_lightImage texturename

By default, surface lights use the average color of the source image to generate the color of the light. *q3map_lightImage* specifies an alternate image to be used for light color emission, radiosity color emission, light filtering and alpha shadows. You can even use a

light image with a different alpha channel for blurrier alpha shadows. The light color is averaged from the referenced texture. The texture must be the same size as the base image map. `q3map_lightImage` should appear before [qer_editorImage](#).

The reason `q3map_lightImage` is specified for the light in the example below, is because the blend map is predominantly yellow, but the base image is not. The designer wanted the color of the light to be sampled from the blend map instead of the base image.

Script: Taking light from another source image

```
textures/eerie/ironcrosslt2_10000
{
    q3map_lightImage textures/gothic_light/ironcrosslt2.blend.tga
    // this TGA is the source for the color of the blended light

    qer_editorImage textures/gothic_light/ironcrosslt2.tga
    //editor TGA (used because the shader is used with several different light values)

    q3map_surfacelight 10000
    //emitted light value of 10,000

    {
        map $lightmap
        //source texture is affected by the lightmap
        rgbGen identity
        // this command handles the overbright bits created by "sunlight" in the game
    }
    {
        map textures/gothic_light/ironcrosslt2.tga
        blendFunc filter
        rgbGen identity
    }
    {
        maptextures/gothic_light/ironcrosslt2.blend.tga
        blendFunc add
    }
}
```



q3map_lightmapAxis axis

Takes a single argument: either x, y or z. The keyword [q3map_terrain](#) has an implicit (read default) `q3map_lightmapAxis` defined as z. This keyword is not recommended for things like caves or arches which have undersides.



q3map_lightmapBrightness N.N

Lightmap brightness scaling. A value of 2.0 will be twice as bright (linearly) and a value of 0.5 will be half as bright.



q3map_lightmapFilterRadius self other

This is usually used on light emitting shaders to approximate finer subdivided lighting. It adds a gaussian blur effect to the lightmaps of either the shader itself, or the surfaces affected by the shader, or both. The values for *self* and *other* are measured in world units of filtering (blurring) of lightmap data cast by any lightsources. The *self* parameter can be set for surfacelights for finer subdivided lighting, but should be set to 0 for sky shaders since they don't have lightmaps. The *other* parameter should be set just high

enough to eliminate the "stadium shadow" effect sometimes produced by [q3map_skylight](#) or to smooth out the lighting on surfacelights. If using a value higher than 4 for the *iterations* parameter on [q3map_skylight](#), you don't need *q3map_lightmapFilterRadius* as much, but at the expense of higher compile times. *q3map_lightmapFilterRadius* should be placed before any light related shader directives that you want it to affect. (see [Appendix I: Light Emitting Shaders](#))



q3map_lightmapGamma *N.N*

Deprecated! Use [q3map_lightmapBrightness](#) instead.



q3map_lightmapMergable

Allows terrain to be mapped onto a single lightmap page for seamless terrain shadows. It will specify that the shaders using it can merge nonplanars together onto a single lightmap, so you can have a single 512x512 lightmap across a terrain entity.



q3map_lightmapSampleOffset *distance*

Takes a single parameter, defaulting to 1.0, which specifies how many units off a surface should Q3Map2 sample lighting from. Use larger values (2.0-8.0) if you're getting ugly splotches on lightmapped terrain.



q3map_lightmapSampleSize *N*

Surfaces using a shader with this option will have the pixel size of the lightmaps set to (NxN). This option can be used to produce high-resolution shadows on certain surfaces. In addition, it can be used to reduce the size of lightmap data, where high-resolution shadows are not required. The default *Quake III* lightmap sample size is 16.



q3map_lightmapSize *width height*

Specifies the size of the lightmap texture that surface lightmaps get packed on to. Used mainly in *Enemy Territory* for terrain lightmaps (512x512) in concert with the Q3Map2 switch -lightmapsize.



q3map_lightRGB *red green blue*

This forces a specified color of light to be emitted from a surface or sky light, rather than sampling colors from a lightimage, editor image or the texture map. Three normalized color values of light are required for the *red green blue* parameters. This does not affect bounced light in radiosity or lightfilter.



q3map_lightStyle *N*

Used to set light styles on surface lights. Equivalent to "style" "N" on a light entity. Sets the emitted light's style. Useful primarily in *Raven* games (JKII, SOF2, JA).



q3map_lightSubdivide *N*

Used on surface lights (see [q3map_surfacelight](#)). Chops up the surface into smaller polygons for more uniform lighting. It defaults to 120 game units, but can be made larger or smaller as needed (for light surfaces at the bottom of cracks, for example). This can be a dominant factor in processing time for Q3Map2 lighting. Can have an increasingly "darker" effect when used with -fast. Compensate by raising the surface light value. For sky shaders, use [q3map_skylight](#) instead (see [Appendix I: Light Emitting Shaders](#)).



q3map_noClip

You might have noticed that terrain has been made to not clip or t-junction anymore. It was causing too many issues, so a new parameter was added: *q3map_noClip*. Normally, Q3Map2 clips all faces to the BSP, and then takes the minimum polygon that encompasses all visible fragments. *q3map_noClip* forces Q3Map2 to use the original brush faces (This is implicit for autosprite(2) surfaces). Therefore, if you map tidy, you could theoretically use *q3map_noClip* on all your shaders. *q3map_noClip* and *q3map_noTJunc*, when used in combination, will preserve mesh geometry exactly as you make it.



q3map_noFast

When used on surfaces that emit light, this will disable -fast optimizations. This is useful for large areas of dim sky, where you want the dim light to reach all surfaces. This shader keyword prevents fast from affecting dim sky surfaces. It is necessary, if you can't do a workaround with brighter skies or by using a larger *q3map_lightSubdivide* value.



q3map_noFog

Volumes marked with a shader containing this keyword will not be affected by fog.



q3map_nonPlanar

Instructs Q3Map2 to merge any adjacent triangles into a non-planar triangle soup.



q3map_normalImage *texturename*

Allow the use of a normal (height) map to simulate textured bumpmapping. This isn't real bumpmapping per se, but generates a static lightmap image that reflects the normal map and light source.



q3map_noTJunc

Read as "no T-Junc". With this option, surfaces modified by a shader are not used for T-junction fixing. *q3map_noClip* and *q3map_noTJunc*, used in combination will preserve mesh geometry exactly as you make it.



q3map_noVertexShadows

Obsolete!

Vertex lighting code was rewritten a couple of times, rendering this keyword irrelevant. Shaders that are used on misc_models or terrain were to use *q3map_noVertexShadows* to disable shadows being cast on the vertex lit surfaces. Casting shadows at small, misc_model objects often makes sense. However, having shadows on large, vertex lit terrain surfaces often looks bad. Shadows are not cast on forced_verts_lit surfaces by default (shaders with pointlight).



q3map_offset *N.N*

Offsets a surface along the vertex normals N.N units. Used in celshading.



q3map_patchShadows

Obsolete!

When this option is used in conjunction with the original lighting algorithm (-light), surfaces with textures modified with this option will show shadows that are cast by curve patches. Curve patches do not cast shadows by default.



q3map_replicate

Does not exist!



q3map_shadeAngle *angle*

Specifies the breaking angle for phong shading. This allows for smooth shadows between brush faces like patches. The *angle* parameter is the angle between adjacent faces at which smoothing will start to occur. Typical values are usually in the 120-179 range.



q3map_skylight *amount iterations*

This replaces *q3map_surfacelight* and *q3map_lightSubdivide* on sky surfaces for much faster and more uniform sky illumination. *Amount* is a brightness value, similar to what you would use in *q3map_sun*. Good values are between 50 and 200. *Iterations* is an exponential factor. 3 is the best value that balances speed and quality. Values of 4 and 5 are higher quality at the expense of higher compile time. Values below 3 are not too useful



q3map_splotchFix

This is used on lightmapped model shaders if splotched lighting artifacts appear. Any shadows at the ambient/dark level will be flooded from neighbouring luxels. This gets rid of shadow acne, but a surface must be more or less uniformly lit or this looks ugly. Try using [q3map_lightmapSampleOffset](#) first before using this as a last resort.



q3map_styleMarker

For use on shaders that accompany style lights. For any shaders that may be hit by a styled light, add *q3map_styleMarker* after the lightmap stage and before the texture stages so Q3Map2 can properly create the fake lightmap stages (see [Appendix G: Lightstyles](#) for details).



q3map_styleMarker2

Similar to [q3map_styleMarker](#) except it is used on masked textures where a depthFunc equal is required.



q3map_sun *red green blue intensity degrees elevation*

This keyword in a sky shader will create the illusion of light cast into a map by a single, infinitely distance parallel light source (sun, moon, hellish fire, etc.). This is only processed during the lighting phase of Q3Map2. While still perfectly usable, *q3map_sun* is now depreciated in favour for *q3map_sunExt* (see below).

red green blue : Color is described by three normalized RGB values. Color will be normalized to a 0.0 to 1.0 range, so it doesn't matter what range you use.

intensity : The brightness of the generated light. A value of 100 is a fairly bright sun. The intensity of the light falls off with angle but not distance.

degrees : The angle relative to the directions of the map file. A setting of 0 degrees equals east. 90 is north, 180 is west and 270 is south. In the original version of Q3Map, non-axial values had a tendency to produce jagged shadows. With Q3Map2, this problem is avoided with new options like lightmap filtering, raytracing and penumbra effects.

elevation : The distance, measured in degrees from the horizon (z value of zero in the map file). An elevation of 0 is sunrise/sunset. An elevation of 90 is noon.

Design Notes:

Sky shaders should probably still have a *q3map_surfacelight* or preferred *q3map_skylight* value. The "sun" gives a strong directional light, but doesn't necessarily give the fill light needed to soften and illuminate shadows. Skies with clouds should probably have a weaker *q3map_sun* value and a higher *q3map_surfacelight* or *q3map_skylight* value. Heavy clouds diffuse light and weaken shadows. The opposite is true of a cloudless or nearly cloudless sky. In such cases, the "sun" or "moon" will cast stronger shadows that have a greater degree of contrast. This is also why *q3map_sunExt* is preferred. It gives the designer greater control over shadow contrast with a penumbra effect.

Design Trick:

Not certain what color formula you want to use for the sun's light? Try this. Create a light entity. Use the Radiant editor's color selection tools to pick a color. The light's *_color* key's value will be the normalized RGB formula. Copy it from the value line in the editor (CTRL+c) and paste it into your shader.



q3map_sunExt *red green blue intensity degrees elevation deviance samples*

Works like *q3map_sun* with the addition of two new parameters to create "light jittering" for penumbra (half-shadow) effects. This gives you much more realistic shadows from the sun, especially when trying to simulate a cloudy day or a wide sun. The penumbra effect can also be applied to entity lights (point, spot or sun) with the *_deviance N* (distance in world units for point/spot lights and degrees for suns) and *_samples N* (number of samples) key/value pairs.

deviance : The number of degrees for the half-shadow. General values up to 2 or 3 are acceptable. The real sun has a solid angle of about half a degree.

samples : The number of random jitters distributed over the solid arc (~16).



q3map_sunlight

Does not exist!



q3map_surfacelight *value*

The texture gives off light equal to the *value* set for it. The relative surface area of the texture in the world affects the actual amount of light that appears to be radiated. To give off what appears to be the same amount of light, a smaller texture must be significantly brighter than a larger texture. Unless the *q3map_lightImage* keyword is used to select a different source for the texture's light color information, the color of the light will be the averaged color of the texture. For sky shaders, use *q3map_skylight* instead for faster and more uniform sky illumination.



q3map_surfaceModel *modelpath density odds minscale maxscale minangle maxangle oriented*

A surface with *q3map_surfaceModel* in its shader will randomly place a specified model across it's face. This is designed to place grass or tree models over terrain.

modelpath : The path to the model file (any supported format).

density : The density of the models, in game units.

odds : The odds of the model appearing (normalized?).

minscale : The minimum scale of the model from its original size of 1.0.

maxscale : The maximum scale of the model from its original size of 1.0.

minangle : The model's minimum angle of rotation.

maxangle : The model's maximum angle of rotation.

oriented : This is a flag, either 0 or 1, and sets whether the model gets fitted to the orientation of the surface.



q3map_tcGen *func ---*

This currently supports two functions, *vector* and *ivector*. Both functions are used for texture projection and do the exact same thing. The only difference is in the math, *ivector* was designed to be more intuitive.

q3map_tcGen vector sVector tVector

Projects a texture *Ns* units by *Nt* units along a chosen axis. *q3map_tcGen vector* (*1/256 0 0*) (*0 1/256 0*) will project a texture every 256 units in x, and every 256 units in y, along the z-axis.

q3map_tcGen ivector 1.0/sVector 1.0/tVector

Projects a texture *Ns* units by *Nt* units along a chosen axis. *q3map_tcGen ivector* (*256 0 0*) (*0 256 0*) will project a texture every 256 units in x, and every 256 units in y, along the z-axis. *Ivector* means inverse vector, and this means you won't have to do the divide with a calculator. Inverse = 1.0/n, unless the value is 0, then the matrix value is set to 0. A bit of dodgy math, but it works.



q3map_tcMod *func ---*

This works in a similar manner to the stage specific *tcMod* keyword (see Chapter 6 [tcMod](#)), except in the compiler, so that modified texture coordinates are "baked" into the surface. This lets you set up less obvious texture tiling on natural and organic surfaces (especially terrain).

q3map_tcMod rotate degrees

Rotates the texture (around origin, not center) a specified number of degrees

q3map_tcMod *scale s-scale t-scale*

Scales S (x) and T (y) texture co-ordinates. *scale 2 2* would halve the size of the texture (doubling the texture co-ordinates).

q3map_tcMod *translate or move or shift s-offset t-offset*

Shifts texture co-ordinates by S, T amount. *translate 0.5 0* would shift it one-half in S, and none in T.



q3map_terrain

Terrain shaders (typically textures/common/terrain and terrain2) must have the *q3map_terrain* keyword. Terrain is handled completely differently from previous versions. Q3Map2 no longer looks for the word terrain in the shader name to determine whether or not its an indexed shader. It looks for *q3map_indexed*, or *q3map_terrain*, which then sets off a bunch of stuff shoehorned into it, like: the lightmap axis, texture projection, etc.

By default, *q3map_terrain* sets the following:

```
q3map_tcGen ivector ( 32 0 0 ) ( 0 32 0 )
q3map_lightmapAxis z
q3map_nonplanar
q3map_shadeAngle 180 (maybe 175?)
q3map_indexed
```



q3map_tessSize *amount*

This controls the tessellation size (how finely a surface is chopped up in to triangles), in game units, of the surface. This is only applicable to solid brushes, not curves, and is generally only used on surfaces that are flagged with the *deformVertexes* keyword. Abuse of this can create a huge number of triangles. This happens during Q3Map2 processing, so maps must be reprocessed for changes to take effect. The poorly named *tessSize* keyword still works but has been deprecated in favour of *q3map_tessSize* for the sake of consistency.

Design Notes:

It can also be used on tessellating surfaces to make sure that tessellations are large and thus, less costly in terms of triangles created.



q3map_textureSize *X Y*

Deprecated. Useful when you don't specify an editor or light image for a shader. Recent builds of Q3Map2 will find some referenced image in a shader and use that as a fallback. Historically, this was used for surface splitting for RTCW for PS2, to accomodate the hardware's limited texture range precision.



q3map_traceLight

Obsolete!

Surfaces using a shader with this option will always be lit with the original light algorithm. Patches will not cast shadows on this surface, unless the shader option *q3map_patchShadows* is also used.



q3map_vertexScale *scale*

The light value, at the vertices of a surface using a shader with this option, is multiplied by the scale value. This is a way to lighten, or darken, a *vertex_lit* surface, in comparison to other *lightmap_lit* surfaces around it.



q3map_vertexShadows

Obsolete! (See [q3map_noVertexShadows](#))

By default, no shadows are cast on *vertex_lit* surfaces (see *surfaceparm pointlight*). In addition, when running *Quake III Arena* in vertex light, no shadows are cast upon any surface at all, since shadows are part of the lightmap. When using this shader keyword, shadows will be cast on surfaces that are vertex lit. However, sharp shadow edges won't be seen on the surfaces, since light values are only calculated at the vertices.



q3map_vlight

Obsolete!

4. Q3Map2 Specific Surface Parameter Shader Keywords

All *surfaceparm* keywords are preceded by the word *surfaceparm* as follows: *surfaceparm fog* or *surfaceparm noimpact*.

These keywords change the physical nature of the textures and the brushes that are marked with them. Changing any of these values will require the map to be re-compiled. These are global and affect the entire shader.

It should be noted that some of these surface parameters will change both the surface as well as the content of an object (*surfaceparm water*, for example). Shaders containing content altering surface parameters should usually be used on all sides of the object.

Many of these keywords are only used in "common" shaders (*baseq3/scripts/common.shader*), which are editor specific shaders used by the level designer. Such keywords are usually not used for the design of custom assets.

Originally, *surfaceparm*'s were part of the previous chapter since they are actually Q3Map2 specific keywords. I've moved them into their own chapter for the sake of being easier to reference, since both this and the previous chapter were getting too long.



surfaceparm alphashadow

This keyword applied to a texture on a brush, patch or model will cause the lighting phase of the Q3Map2 process to use the texture's alpha channel as a mask for casting static shadows in the game world.

Design Notes:

Alphashadow does not work well with fine line detail on a texture. Fine lines may not cast acceptable shadows. It appears to work best with well-defined silhouettes and wider lines within the texture. Most of our tattered banners use this to cast tattered shadows. With Q3Map2, it is possible to increase the resolution of the lightmap receiving the shadows with a slight the cost of memory. This can be achieved with the *q3map_lightmapSampleSize* keyword on the shadow receiving shader or by creating a func_group of the shadow receiving brushes and adding the *_lightmapScale* key with a floating-point value for the scale of the lightmap.



surfaceparm antiportal

Works like *hint* brushes in that it creates BSP nodes, but unlike hint, it blocks vis by not creating a portal at the split. This is designed to be used with large terrain maps to block visibility without having to resort to tricks like sky or caulk brushes penetrating the terrain and throwing ugly shadows.

<http://shaderlab.com/mpcenter/q3map/antiportal.jpg>

Players in part A of the map will not be able to see into part B and vice-versa. You can walk through the *antiportal* just fine. Note that this also blocks light. There are two caveats: They are opaque to light, and if aligned to another BSP cut (such as blocksize or the origin) they will not function correctly. Don't align it with anything else (like blocksize or another brush face) and it'll block vis. This keyword is found in "common/antiportal" so you shouldn't need to specify this. "common/antiportal" was added by ydnar, so if you're missing this shader, it is included with the latest version of [Q3Map2](#).



surfaceparm areaportal

A brush marked with this keyword functions as an *areaportal*, a break in the BSP tree. It is typically placed on a very thin brush placed inside a door entity (but is not a part of that entity). The intent is to block the game from processing surface triangles located behind it when the door is closed. It is also used by the BSPC (bot area file creation compiler) in the same manner as a *clusterportal*. The brush must touch all the structural brushes surrounding the *areaportal*. This keyword is found in "common/areaportal" so you shouldn't need to specify this.



surfaceparm botclip

Blocks bot movement only. Other game world entities and human players can pass through a brush marked *botclip*. The intended use for this is to block the bot but not other players or projectiles. This keyword is found in "common/botclip" so you shouldn't need to specify this.

Design Notes:

Careful use of *botclip* in a map can greatly reduce the complexity of the .aas bot navigation file, resulting in "smarter", more efficient bots.

**surfaceparm clusterportal**

A brush marked with this keyword function creates a subdivision of the area file (.aas) used by the bots for navigation. It is typically placed in locations that are natural breaks in a map, such as entrances to halls, doors, tunnels, etc. The intent is keep the bot from having to process the entire map at once. As with the the *areaportal* parameter, the affected brush must touch all the structural brushes surrounding the *clusterportal*. This keyword is found in "common/clusterportal" so you shouldn't need to specify this.

**surfaceparm detail**

This surface attribute causes a brush to be ignored by the Q3Map2 process for generating possible break-points in the BSP tree. Generally speaking, detail brushes are usually set in the editor, so you shouldn't need to specify this.

**surfaceparm donotenter**

Read as "*do not enter*". Like *clusterportal*, this is a bot-only property. A brush marked with *donotenter* will not affect non-bot players, but bots will not enter it. It should be used only when bots appear to have difficulty navigating around some map features. This does not physically stop the bot from entering a region (as with *botclip*). Bots will not enter the area on their own but may, for example, be blasted into the region with a rocket launcher. This keyword is found in "common/donotenter" so you shouldn't need to specify this.

Design Notes: *donotenter* can be (sparingly) used in space maps around the void or around lava/slime in certain places where bots have an overwhelming tendency to commit suicide (lemming style).

**surfaceparm dust**

If a player lands on a surface that uses a shader with this parameter, a puff of dust will appear at the player's feet. Note that the worldspawn entity must contain the "enabledust" key with a set value of "1".

**surfaceparm flesh**

This will cue different sounds (in a similar manner to metalsteps) and cause blood to appear instead of bullet impact flashes. Actually, the code for this was never fully implemented by id Software. It remains half-finished so unfortunately it doesn't work.



surfaceparm fog

fog defines the brush as being a "fog" brush. This is a Q3Map2 function that chops and identifies all geometry inside the brush. The general shader keyword *fogparms* must also be specified to tell how to draw the fog.



surfaceparm hint

When Q3Map2 calculates the vis data, it tries to place portals in places in the map in an attempt to limit the potential viewable set (PVS). Brushes marked by a *hint* shader are used to manually place portals to force a break in the PVS. This keyword is found in "common/hint" so you shouldn't need to specify this.



surfaceparm ladder

Supposedly used to allow the player to climb vertically. This is not functional in *Quake III Arena*.



surfaceparm lava

Assigns to the texture the game properties set for lava. This affects both the surface and the content of a brush.



surfaceparm lightfilter

Causes the Q3Map2 light stage to use the texture's RGB and alpha channels to generate colored alpha shadows in the lightmap. For example, this can be used to create the colored light effect cast by stained glass windows. This can be used with *surfaceparm alphashadow*.



surfaceparm lightgrid

The min/max bounds of brushes with this shader in a map will define the bounds of the map's lightgrid (model lighting). Make it as small as possible around player space to minimize bsp size and compile time. This keyword is found in "common/lightgrid" so you shouldn't need to specify this. "common/lightgrid" was added by ydnar, so if you're missing this shader, it is included with the latest version of [Q3Map2](#).



surfaceparm metalsteps

The player sounds as if he is walking on clanging metal steps or gratings. Other than specifying *flesh* (doesn't work), *metalsteps*, *nosteps*, or default (i.e. specify nothing) it is currently not possible for a designer to create or assign a specific sound routine to a texture. Note: If no sound is set for a texture, then the default footsteps sound routines are heard.



surfaceparm monsterclip

Blocks monster movement. Not functional in *Quake III Arena*.



surfaceparm nodamage

The player takes no fall damage if he lands onto a texture with this surfaceparm. This keyword is found in "common/cushion" but you may want to specify this on certain shaders (jump pads, for example).



surfaceparm nodlight

Read as "*No Dee-Light*". A texture containing this parameter will not be affected or lit by dynamic lights, such as weapon effects. An example in *Quake III Arena* would be solid lava.



surfaceparm nodraw

A texture marked with nodraw will not visually appear in the game world. Most often used for triggers, clip brushes, origin brushes, shaders with *cull none* or *cull disable* and so on. This keyword is found in "common/nodraw" so you shouldn't need to specify this.



surfaceparm nodrop

When a player dies inside a volume (brush) marked *nodrop*, no weapon is dropped. The intend use is for "Pits of Death." Have a kill trigger inside a *nodrop* volume, and when the players die here, they won't drop their weapons. The intent is to prevent unnecessary polygon pileups on the floors of pits. This keyword is found in "common/nodrop" but you may want to specify this on certain shaders (fog volumes in pits, for example).



surfaceparm noimpact

World entities will not impact on this texture. No explosions occur when projectiles strike this surface and no marks will be left on it. Sky textures are usually marked with this texture so those projectiles will not hit the sky and leave marks.



surfaceparm nomarks

Projectiles will explode upon contact with this surface, but will not leave marks. Blood will also not mark this surface. This is useful to keep lights from being temporarily obscured by battle damage.

Design Notes:

Use this on any surface with a `deformVertexes` keyword. Otherwise, the marks will appear on the unmodified surface location of the texture with the surface wriggles and squirms through the marks.

**surfaceparm nolightmap**

This texture does not have a lightmap phase. It is not affected by the ambient lighting of the world around it. It does not require the addition of an `rgbGen` identity keyword in that stage.

**surfaceparm nosteps**

The player makes no sound when walking on this texture.

**surfaceparm nonsolid**

This attribute indicates a brush, which does not block the movement of entities in the game world. It applied to triggers, hint brushes and similar brushes. This affects the content of a brush.

**surfaceparm origin**

Used on the "origin" texture. Rotating entities need to contain an origin brush in their construction. The brush must be rectangular (or square). The origin point is the exact center of the origin brush. This keyword is found in "common/origin" so you shouldn't need to specify this.

**surfaceparm playerclip**

Blocks player movement through a *nonsolid* texture. Other game world entities can pass through a brush marked `playerclip`. The intended use for this is to block the player but not block projectiles like rockets. This keyword is found in "common/clip" so you shouldn't need to specify this.

Design Notes:

playerclip is often useful for "smoothing" out the geometry of the map, preventing the player from snagging on objects. It is also used in open sky areas of maps, preventing the player from flying too high and seeing the "Hall of Mirrors" effect at the bottom of the cloud layer.



surfaceparm pointlight

Sample lighting at vertices??? I'm not sure what this does at this point.



surfaceparm skip

Works just like *Quake II* skip texture. Use on sides of hint and antiportal brushes where you don't want BSP splits. This keyword is found in "common/skip" so you shouldn't need to specify this. "common/skip" was added by ydnar, so if you're missing this shader, it is included with the latest version of [Q3Map2](#).



surfaceparm sky

This flags the compiler, telling it that this surface should be rendered as sky.



surfaceparm slick

This surfaceparm included in a texture should give it significantly reduced friction. This keyword is found in "common/slick" but you may want to specify this on certain shaders (ice, for example).



surfaceparm slime

Assigns to the texture the game properties for slime. This affects both the surface and the content of a brush.



surfaceparm structural

This surface attribute causes a brush to be seen by the Q3Map2 process as a possible break-point in a BSP tree. It is used as a part of the shader for the "*hint*" texture. Generally speaking, any opaque texture not marked as "*detail*" is, by default, *structural*, so you shouldn't need to specify this.



surfaceparm trans

Tells Q3Map2 that pre-computed visibility should not be blocked by this surface. Generally, any shaders that have blendfunc's should be marked as surfaceparm trans.



surfaceparm water

Assigns to the texture the game properties for water. This affects both the surface and the content of a brush.

5. Editor Specific Shader Keywords

These instructions only affect the texture when it is seen in the Radiant editor. They should be grouped with the surface parameters but ahead of them in sequence.



qer_editorImage *textureName*

This keyword creates a shader name in memory, but in the editor, it displays the TGA art image specified in qer_editorImage (in the example below this is textures/eerie/lavahell.tga).

The editor maps a texture using the size attributes of the TGA file used for the editor image. When that editor image represents a shader, any texture used in any of the shader stages will be scaled up or down to the dimensions of the editor image. If a 128x128 pixel image is used to represent the shader in the editor, then a 256x256 image used in a later stage will be shrunk to fit. A 64x64 image would be stretched to fit. Be sure to check this on bouncy, acceleration, and power-up pads placed on surfaces other than 256 x 256. Use *tcMod scale* to change the size of the stretched texture. Remember that *tcMod scale* 0.5 0.5 will double your image, while *tcMod scale* 2 2 will halve it.

Design Notes:

The base_light and gothic_light shaders contain numerous uses of this. It can be very useful for making different light styles (mostly to change the light brightness) without having to create a new piece of TGA art for each new shader.

Script: Setting an editorImage

```
textures/liquids/lavahell2                                //path and name of new texture
{
    qer_editorImage textures/eerie/lavahell.tga           //based on this
    qer_nocarve                                           //cannot be cut by CSG subtract
    surfaceparm noimpact                                  //projectiles do not hit it
    surfaceparm lava                                       //has the game properties of lava
    surfaceparm nolightmap                                 //environment lighting does not affect
    q3map_surfacelight 3000                               //light is emitted
    tessSize 256                                           //relatively large triangles
    cull disable                                           //no sides are removed
    deformVertexes wave 100 sin 5 5 .5 0.02
    fogparms 0.8519142 0.309723 0.0 128 128
    {
        maptextures/eerie/lavahell.tga                   //base texture artwork
        tcMod turb .25 0.2 1 0.02                        //texture is subjected to turbulence
        tcMod scroll 0.1 0.1                             //the turbulence is scrolled
    }
}
```



qer_nocarve

A brush marked with this instruction will not be affected by CSG subtract functions. It is especially useful for water and fog textures.



qer_trans value

This parameter defines the percentage of transparency that a brush will have when seen in the editor (no effect on game rendering at all). It can have a positive value between 0 and 1. The higher the value, the less transparent the texture. Example: `qer_trans 0.2` means the brush is 20% opaque and nearly invisible.

Design Notes:

If you use `qer_trans` on a shader whose `qer_editorImage` has an alpha channel, the transparent areas of the `editorImage` will also be transparent in the editor. To keep the solid areas of the `editorImage` opaque, use a `qer_trans` value of "0.999". Useful for grates, windows, fences, etc.

6. Stage Specific Shader Keywords

Stage specifications only affect rendering. Changing any keywords or values within a stage will usually take effect as soon as a `vid_restart` is executed. Q3Map2 ignores stage specific keywords entirely.

A stage can specify a texture map, a color function, an alpha function, a texture coordinate function, a blend function, and a few other rasterization options.



Texture Map Specification

map texturename

Specifies the source texture map (a 24 or 32-bit TGA file) used for this stage. The texture may or may not contain alpha channel information. The special keywords `$lightmap` and `$whiteimage` may be substituted in lieu of an actual texture map name. In those cases, the texture named in the first line of the shader becomes the texture that supplies the light mapping data for the process. The texture name should always end with the ".tga" suffix regardless of whether the source texture map is actually a .tga file or .jpg.

\$lightmap

This is the overall lightmap for the game world. It is calculated during the Q3Map2 process. It is the initial color data found in the framebuffer. Note: due to the use of overbright bits in light calculation, the keyword `rgbGen identity` must accompany all `$lightmap` instructions.

\$whiteimage

This is used for specular lighting on MD3 models. This is a white image generated internally by the game. This image can be used in lieu of `$lightmap` or an actual texture map if, for example, you wish for the vertex colors to come through unaltered.

clampMap texturename

Dictates that this stage should clamp texture coordinates instead of wrapping them. During a stretch function, the area, which the texture must cover during a wave cycle,

enlarges and decreases. Instead of repeating a texture multiple times during enlargement (or seeing only a portion of the texture during shrinking) the texture dimensions increase or contract accordingly. This is only relevant when using something like *deformTexCoordParms* to stretch/compress texture coordinates for a specific special effect. Remember that the *Quake III Arena* engine normalizes all texture coordinates (regardless of actual texture size) into a scale of 0.0 to 1.0.

Proper Alignment: When using *clampTexCoords* make sure the texture is properly aligned on the brush. The *clampTexCoords* function keeps the image from tiling. However, the editor doesn't represent this properly and shows a tiled image. Therefore, what appears to be the correct position may be offset. This is very apparent on anything with a *tcMod rotate* and *clampTexCoords* function.

//obsidian: Figure 2a & 2b??? Where???

//obsidian: clampTexCoords???

Avoiding Distortion: When seen at a given distance (which can vary, depending on hardware and the size of the texture), the compression phase of a stretch function will cause a "cross"-like visual artifact to form on the modified texture due to the way that textures are reduced. This occurs because the texture undergoing modification lacks sufficient "empty space" around the displayed (non-black) part of the texture (see figure 2a). To compensate for this, make the non-zero portion of the texture substantially smaller (50% of maximum stretched size -- see figure 2b) than the dimensions of the texture. Then, write a scaling function (*tcMod scale*) into the appropriate shader phase, to enlarge the image to the desired proportion.

The shaders for the bouncy pads (in the *sfx.shader* file) show the stretch function in use, including the scaling of the stretched texture:

Script: Using clampMap to control a stretching texture

```
textures/sfx/metalbridge06_bounce
{
    //q3map_surfacelight 2000
    surfaceparm nodamage
    q3map_lightimage textures/sfx/jumppadsmall.tga
    q3map_surfacelight 400
    {
        map textures/sfx/metalbridge06_bounce.tga
        rgbGen identity
    }
    {
        map $lightmap
        rgbGen identity
        blendfunc gl_dst_color gl_zero
    }
    {
        map textures/sfx/bouncepad01b_layer1.tga
        blendfunc gl_one gl_one
        rgbGen wave sin .5 .5 0 1.5
    }
    {
        clampmap textures/sfx/jumppadsmall.tga
        blendfunc gl_one gl_one
        tcMod stretch sin 1.2 .8 0 1.5
        rgbGen wave square .5 .5 .25 1.5
    }
    // END
}
```

animMap frequency texture1... texture8

The surfaces in the game can be animated by displaying a sequence of 1 to 8 frames (separate texture maps). These animations are affected by other keyword effects in the same and later shader stages.

frequency : The number of times that the animation cycle will repeat within a one second time period. The larger the value, the more repeats within a second. Animations that should last for more than a second need to be expressed as decimal values.

texture1... texture8 : the texture path/texture name for each animation frame must be explicitly listed. Up to eight frames (eight separate .tga files) can be used to make an animated sequence. Each frame is displayed for an equal subdivision of the frequency value.

Example:

animMap 0.25 textures/sfx/b_flame1.tga textures/sfx/b_flame2.tga textures/sfx/b_flame3.tga textures/sfx/b_flame4.tga would be a 4 frame animated sequence, calling each frame in sequence over a cycle length of 4 seconds. Each frame would be displayed for 1 second before the next one is displayed. The cycle repeats after the last frame in sequence shown.

Design Notes:

To make a texture image appear for an unequal (longer) amount of time (compared to other frames), repeat that frame more than once in the sequence.

Script: Specifying an animMap

```
textures/sfx/flameanim_blue
{
    // *****
    // *          Blue Flame          *
    // *      July 20, 1999 Surface Light 1800      *
    // *      Please Comment Changes              *
    // *****
    qer_editorimage textures/sfx/b_flame7.tga
    q3map_lightimage textures/sfx/b_flame7.tga
    surfaceparm trans
    surfaceparm nomarks
    surfaceparm nolightmap
    cull none
    q3map_surfacelight 1800
    // texture changed to blue flame.... PAJ
    {
        animMap 10 textures/sfx/b_flame1.tga textures/sfx/b_flame2.tga
            textures/sfx/b_flame3.tga textures/sfx/b_flame4.tga
            textures/sfx/b_flame5.tga textures/sfx/b_flame6.tga
            textures/sfx/b_flame7.tga textures/sfx/b_flame8.tga
        blendFunc GL_ONE GL_ONE
        rgbGen wave inverseSawtooth 0 1 0 10
    }
    {
        animMap 10 textures/sfx/b_flame2.tga textures/sfx/b_flame3.tga
            textures/sfx/b_flame4.tga textures/sfx/b_flame5.tga
            textures/sfx/b_flame6.tga textures/sfx/b_flame7.tga
            textures/sfx/b_flame8.tga textures/sfx/b_flame1.tga
        blendFunc GL_ONE GL_ONE
        rgbGen wave sawtooth 0 1 0 10
    }
    {
        map textures/sfx/b_flameball.tga
        blendFunc GL_ONE GL_ONE
        rgbGen wave sin .6 .2 0 .6
    }
}
```



blendFunc func

Blend functions are the keyword commands that tell the *Quake III Arena* graphic engine's renderer how graphic layers are to be mixed together.

Simplified Blend Functions

The most common blend functions are set up here as simple commands, and should be used unless you really know what you are doing.

add : Shorthand command for *blendFunc gl_one gl_one*. Effects like fire and energy are additive.

filter : Shorthand command that can be substituted for either *blendFunc gl_dst_color gl_zero* or *blendFunc gl_zero gl_src_color*. A filter will always result in darker pixels than what is behind it, but it can also remove color selectively. Lightmaps are filters.

blend : Shorthand command for *blendFunc gl_src_alpha gl_one_minus_src_alpha*. This is conventional transparency, where part of the background is mixed with part of the texture.

Explicit Blend Functions

Getting a handle on this concept is absolutely key to understanding all shader manipulation of graphics.

blendFunc or "Blend Function" is the equation at the core of processing shader graphics. The formula reads as follows:

$$[\text{source} * (\text{srcBlend})] + [\text{destination} * (\text{dstBlend})]$$

Source is usually the RGB color data in a texture TGA file (remember it's all numbers) modified by any *rgbGen* and *alphaGen*. In the shader, the source is generally identified by command [map](#), followed by the name of the image.

Destination is the color data currently existing in the frame buffer.

Rather than think of the entire texture as a whole, it maybe easier to think of the number values that correspond to a single pixel, because that is essentially what the computer is processing... one pixel of the bitmap at a time.

The process for calculating the final look of a texture in place in the game world begins with the precalculated lightmap for the area where the texture will be located. This data is in the frame buffer. That is to say, it is the initial data in the *destination*. In an unmanipulated texture (i.e. one without a special shader script), color information from the texture is combined with the lightmap. In a shader-modified texture, the *lightmap* stage must be present for the lightmap to be included in the calculation of the final texture appearance.

Each pass or "stage" of blending is combined (in a cumulative manner) with the color data passed onto it by the previous stage. How that data combines together depends on the values chosen for the *source blends* and *destination blends* at each stage. Remember it's numbers that are being mathematically combined together that are ultimately interpreted as colors.

A general rule is that any *source blend* other than *GL_ONE* (or *GL_SRC_ALPHA* where the alpha channel is entirely white) will cause the *source* to become darker.

Source Blend *srcBlend*

The following values are valid for the *source blend* part of the equation.

GL_ONE This is the value 1. When multiplied by the *source*, the value stays the same. The value of the color information does not change.

GL_ZERO This is the value 0. When multiplied by the *source*, all RGB data in the *source* becomes zero (essentially black).

GL_DST_COLOR This is the value of color data currently in the *destination* (frame buffer). The value of that information depends on the information supplied by previous stages.

GL_ONE_MINUS_DST_COLOR This is nearly the same as *GL_DST_COLOR* except that the value for each component color is inverted by subtracting it from one. (i.e. $R = 1.0 - \text{DST.R}$, $G = 1.0 - \text{DST.G}$, $B = 1.0 - \text{DST.B}$, etc.)

GL_SRC_ALPHA The TGA file being used for the *source* data must have an alpha channel in addition to its RGB channels (for a total of four channels). The alpha channel is an 8-bit black and white only channel. An entirely white alpha channel will not darken the *source*.

GL_ONE_MINUS_SRC_ALPHA This is the same as *GL_SRC_ALPHA* except that the value in the alpha channel is inverted by subtracting it from one. (i.e. $A = 1.0 - \text{SRC.A}$)

Destination Blend *dstBlend*

The following values are valid for the *destination blend* part of the equation.

GL_ONE This is the value 1. When multiplied by the *destination*, the value stays the same the value of the color information does not change.

GL_ZERO This is the value 0. When multiplied by the *destination*, all RGB data in the *destination* becomes zero (essentially black).

GL_SRC_COLOR This is the value of color data currently in the *source* (which is the texture being manipulated here).

GL_ONE_MINUS_SRC_COLOR This is the value of color data currently in *source*, but subtracted from one (i.e. inverted).

GL_SRC_ALPHA The TGA file being used for the *source* data must have an alpha channel in addition to its RGB channels (four a total of four channels). The alpha channel is an 8-bit black and white only channel. An entirely white alpha channel will not darken the *source*.

GL_ONE_MINUS_SRC_ALPHA This is the same as *GL_SRC_ALPHA* except that the value in the alpha channel is inverted by subtracting it from one. (i.e. $A = 1.0 - \text{SRC.A}$).

Doing the Math: The Final Result

The product of the *source* side of the equation is added to the product of the *destination* side of the equation. The sum is then placed into the frame buffer to become the destination information for the next stage. Ultimately, the equation creates a modified color value that is used by other functions to define what happens in the texture when it is displayed in the game world.

Default Blend Function

If no *blendFunc* is specified then no blending will take place. A warning is generated if any stage after the first stage does not have a *blendFunc* specified.

Technical Information/Limitations Regarding Blend Modes:

The Riva 128 graphics card supports ONLY the following blendmodes:

GL_ONE, GL_ONE
GL_DST_COLOR, GL_ZERO
GL_ZERO, GL_SRC_COLOR
GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA
GL_ONE_MINUS_SRC_ALPHA, GL_SRC_ALPHA

Cards running in 16 bit color cannot use any GL_DST_ALPHA blends.



rgbGen *func*

There are two color sources for any given shader, the texture file and the vertex colors. Output at any given time will be equal to TEXTURE multiplied by VERTEXCOLOR. Most of the time VERTEXCOLOR will default to white (which is a normalized value of 1.0), so output will be TEXTURE (this usually lands in the *Source* side of the shader equation). Sometimes you do the opposite and use TEXTURE = WHITE, but this is only commonly used when doing specular lighting on entities (i.e. shaders that level designers will probably never create).

The most common reason to use rgbGen is to pulsate something. This means that the VERTEXCOLOR will oscillate between two values, and that value will be multiplied (darkening) the texture.

If no rgbGen is specified, either "identityLighting" or "identity" will be selected, depending on which blend modes are used.

Valid parameters are *wave*, *identity*, *identityLighting*, *entity*, *oneMinusEntity*, *fromVertex*, and *lightingDiffuse*.

rgbGen *identityLighting*

Colors will be (1.0, 1.0, 1.0) if running without overbright bits (NT, Linux, windowed modes), or (0.5, 0.5, 0.5) if running with overbright. Overbright allows a greater color range at the expense of a loss of precision. Additive and blended stages will get this by default.

rgbGen *identity*

Colors are assumed to be all white (1.0, 1.0, 1.0). All filters stages (lightmaps, etc) will get this by default.

rgbGen *wave func base amplitude phase freq*

Colors are generated using the specified waveform. An affected texture will become darker and lighter, but will not change hue. Hue stays constant. Note that the RGB values for color will not go below 0 (black) or above 1 (white). Valid waveforms are sin, triangle, square, sawtooth and inversesawtooth (see [1.4.8 Waveform Functions](#)).

rgbGen *entity*

Colors are grabbed from the entity's modulate field. This is used for things like explosions.

Design Notes:

This keyword would probably not be used by a level designer.

rgbGen *oneMinusEntity*

Colors are grabbed from 1.0 minus the entity's modulate field.

Design Note:

This keyword would probably not be used by a level designer.

rgbGen *vertex*

Colors are filled in directly by the data from the map or model files.

Design Note:

rgbGen vertex should be used when you want the RGB values to be computed for a static model (i.e. mapobject) in the world using precomputed static lighting from Q3BSP. This would be used on things like the gargoyles, the portal frame, skulls, and other decorative models put into the *Quake III Arena* world.

rgbGen *oneMinusVertex*

As *rgbGen vertex*, but inverted.

Design Note:

This keyword would probably not be used by a level designer.

rgbGen *lightingDiffuse*

Colors are computed using a standard diffuse lighting equation. It uses the vertex normals to illuminate the object correctly.

Design Notes:

rgbGen lightingDiffuse is used when you want the RGB values to be computed for a dynamic model (i.e. non-map object) in the world using regular in-game lighting. For example, you would specify on shaders for items, characters, weapons, etc.

**alphaGen *func***

The alpha channel can be specified like the RGB channels. If not specified, it defaults to 1.0. Valid **func** values are *lightingSpecular*, *wave*, *entity*, *oneMinusEntity*, *vertex*, *oneMinusVertex*, *portal*.

//Obsidian: Describe each in more detail... like rgbGen above.

alphaGen *portal*

This rendering stage keyword is used in conjunction with the surface parameter keyword *portal*. The function accomplishes the "fade" that causes the scene in the portal to fade from view. Specifically, it means "Generate alpha values based on the distance from the viewer to the portal." Use *alphaGen portal* on the last rendering pass.

//Obsidian: needs formatting???



tcGen *coordinateSource*

Specifies how texture coordinates are generated and where they come from. Valid functions are *base*, *lightmap*, *environment* and *vector*.

base : Base texture coordinates from the original art.

lightmap : Lightmap texture coordinates.

environment : Make this object environment mapped.

vector : Projects a texture from a specified direction.

tcGen vector (sx sy sz) (tx ty tz)

New texture coordinate generation by world projection. This allows you to project a texture onto a surface in a fixed way, regardless of its orientation.

S coordinates correspond to the "x" coordinates on the texture itself.

T coordinates correspond to the "y" coordinates on the texture itself.

The measurements are in game units.

Example: tcGen vector (0.01 0 0) (0 0.01 0)

This would project a texture with a repeat every 100 units across the X/Y plane. The value comes from dividing the game units from 1, in this case, 1/100.

//Obsidian: is there a tcGen ivector?



tcMod func ---

Specifies how texture coordinates are modified after they are generated. The valid functions for tcMod are *rotate*, *scale*, *scroll*, *stretch*, *transform* and *turb*. *Transform* is a function generally reserved for use by programmers who suggest that designers leave it alone. When using multiple *tcMod* functions during a stage, place the *scroll* command last in order, because it performs a mod operation to save precision, and that can disturb other operations. Texture coordinates are modified in the order in which *tcMods* are specified. In otherwords, if you see:

```
tcMod scale 0.5 0.5
```

```
tcMod scroll 1 1
```

then the texture coordinates will be scaled *then* scrolled.

tcMod rotate degrees/sec

This keyword causes the texture coordinates to rotate. The value is expressed in degrees rotated each second. A positive value means clockwise rotation. A negative value means counterclockwise rotation. For example "tcMod rotate 5" would rotate texture coordinates 5 degrees each second in a clockwise direction. The texture rotates around the center point of the texture map, so you are rotating a texture with a single repetition, be careful to center it on the brush (unless off-center rotation is desired).

tcMod scale sScale tScale

Resizes (enlarges or shrinks) the texture coordinates by multiplying them against the given factors of *sScale* and *tScale*. The values "s" and "t" conform to the "x" and "y" values (respectively) as they are found in the original texture TGA. The values for *sScale* and *tScale* are NOT normalized. This means that a value greater than 1.0 will increase the size of the texture. A positive value less than one will reduce the texture to a fraction

of its size and cause it to repeat within the same area as the original texture (Note: see clampTexCoords for ways to control this).

Example: *tcMod scale 0.5 2* would cause the texture to repeat twice along its width, but expand to twice its height (in which case half of the texture would be seen in the same area as the original)

tcMod scroll sSpeed tSpeed

Scrolls the texture coordinates with the given speeds. The values "s" and "t" conform to the "x" and "y" values (respectively) as they are found in the original texture TGA. The scroll speed is measured in "textures" per second. A "texture" is the dimension of the texture being modified and includes any previous shader modifications to the original TGA). A negative s value would scroll the texture to the left. A negative t value would scroll the texture down.

Example: *tcMod scroll 0.5 -0.5* moves the texture down and right (relative to the TGA files original coordinates) at the rate of a half texture each second of travel.

This should be the LAST tcMod in a stage. Otherwise there maybe popping or snapping visual effects in some shaders.

tcMod stretch func base amplitude phase frequency

Stretches the texture coordinates with the given function. Stretching is defined as stretching the texture coordinate away from the center of the polygon and then compressing it towards the center of the polygon. (see [Chapter 1: Key Concepts](#) for waveform parameter definitions).

tcMod transform m00 m01 m10 m11 t0 t1

Transforms each texture coordinate as follows:

$$S' = s * m00 + t * m10 + t0$$

$$T' = s * m01 + t * m11 + t1$$

This is for use by programmers.

tcMod turb base amplitude phase freq

Applies turbulence to the texture coordinate. Turbulence is a back and forth churning and swirling effect on the texture.

base : Currently undefined.

amplitude : This is essentially the intensity of the disturbance or twisting and squiggling of the texture.

phase : See the explanation for phase under the deformVertexes keyword.

freq : Frequency. This value is expressed as repetitions or cycles of the wave per second. A value of one would cycle once per second. A value of 10 would cycle 10 times per second. A value of 0.1 would cycle once every 10 seconds.



depthFunc func

This controls the depth comparison function used while rendering. The default is "lequal" (Less than or equal to) where any surface that is at the same depth or closer of an existing surface is drawn. This is used for textures with transparency or translucency. Under some circumstances you may wish to use "equal", e.g. for light-mapped grates that are alpha tested (it is also used for mirrors).



depthWrite

By default, writes to the depth buffer when depthFunc passes will happen for opaque surfaces and not for translucent surfaces. Blended surfaces can have the depth writes forced with this function.



detail

This feature was not used in Quake III Arena maps, but should still function. Designates this stage as a detail texture stage, which means that if the `c_var`, `r_detailtextures`, is set to 0 then this stage will be ignored (detail will not be displayed). This keyword, by itself, does not affect rendering at all. If you do put in a detail texture, it has to conform to very specific rules. Specifically, the `blendFunc`:

`blendFunc GL_DST_COLOR GL_SRC_COLOR`

This is also the simple blend function: *blendFunc filter*

And the average intensity of the detail texture itself must be around 127.

Detail is used to blend fine pixel detail back into a base texture whose scale has been increased significantly. When detail is written into a set of stage instructions, it allows the stage to be disabled by the `c_var` console command setting "`r_detailtextures 0`".

A texture whose scale has been increased beyond a 1:1 ratio tends not to have very high frequency content. In other words, one texel can cover a lot of real estate. Frequency is also known as "detail." Lack of detail can appear acceptable if the player never has the opportunity to see the texture at close range. But seen close up, such textures look glaringly wrong within the sharp detail of the Quake III Arena environment. A detail texture solves this problem by taking a noisy "detail" pattern (a tiling texture that appears to have a great deal of surface roughness) and applying it to the base texture at a very densely packed scale (that is, reduced from its normal size). This is done programmatically in the shader, and does not require modification of the base texture. Note that if the detail texture is the same size and scale as the base texture that you may as well just add the detail directly to the base texture. The theory is that the detail texture's scale will be so high compared to the base texture (e.g.; 9 detail texels fitting into 1 base texel) that it is literally impossible to fit that detail into the base texture directly.

For this to work, the rules are as follows:

1. the lightmap must be rendered first. This is because the subsequent detail texture will be modifying the lightmap in the framebuffer directly.
2. The base texture must be rendered next.
3. The detail texture must be rendered last since it modifies the lightmap in the framebuffer. There is a bug in *Quake III* that disables all stages in a shader after a "detail" stage if `r_detailTextures` is set to 0.
4. The detail texture MUST have a mean intensity around 127-129. If it does not then it will modify the displayed texture's perceived brightness in the world.
5. The detail shader stage MUST have the "detail" keyword or it will not be disabled if the user uses the "`r_detailtextures 0`" setting.
6. The detail stage MUST use "`blendFunc GL_DST_COLOR GL_SRC_COLOR`". Any other BlendFunc will cause mismatches in brightness between detail and non-detail views.

7. The detail stage should scale its textures by some amount (usually between 3 and 12) using "tcMod scale" to control density. This roughly corresponds to coarseness. A very large number, such as 12, will give very fine detail, however that detail will disappear VERY quickly as the viewer moves away from the wall since it will be MIP mapped away. A very small number, e.g. 3, gives diminishing returns since not enough is brought in when the user gets very close. I'm currently using values between 6 and 9.5. You should use non-integral numbers as much as possible to avoid seeing repeating patterns.
8. Detail textures add one pass of overdraw, so there is a definite performance hit.
9. Detail textures can be shared, so designers may wish to define only a very small handful of detail textures for common surfaces such as rocks, etc.

An example (non-existent) detail shader is as follows:

Script: Detail shaders

```
textures/bwhptest/foo
{
    q3map_globalTexture //may be required when using tcMod scale in later stages

    // draw the lightmap first
    {
        map $lightmap
        rgbGen identity
    }

    // draw the base texture
    {
        map textures/castle/blocks11b.tga
        blendFunc filter
    }

    // highly compressed detail texture
    {
        map textures/details/detail01.tga
        blendFunc GL_DST_COLOR GL_SRC_COLOR //MUST BE USED
        detail //allows detail shaders to be disabled
        tcMod scale 9.1 9.2
    }
}
```



alphaFunc func

Determines the alpha test function used when rendering this map. Valid values are GT0, LT128, and GE128. These correspond to "GREATER THAN 0", "LESS THAN 128", and "GREATER THAN OR EQUAL TO 128". This function is used when determining if a pixel should be written to the framebuffer. For example, if GT0 is specified, the only the portions of the texture map with corresponding alpha values greater than zero will be written to the framebuffer. By default alpha testing is disabled.

Both alpha testing and normal alpha blending can be used to get textures that have see-through parts. The difference is that alphaFunc is an all-or-nothing test, while blending smoothly blends between opaque and translucent at pixel edges. Alpha test can also be used with *depthWrite*, allowing other effects to be conditionally layered on top of just the opaque pixels by setting *depthFunc* to equal.

7. Quake 3 Engine Game Specific Shader Keywords

The Quake 3 Engine has been licensed to several different companies for the development of third party games. This chapter is a sub-manual documenting the Quake 3 Engine game specific shader keywords developed for third party games like Return to Castle Wolfenstein: Enemy Territory, Jedi Knights II, Soldier of Fortune II, Jedi Academy, Star Trek: Elite Force, as examples.

Note: These are just keywords that have been submitted by people on the forums/irc channels. I have no idea about the accuracy of this chapter. Please use with caution. Please e-mail me (Obsidian) for any additions or error corrections to this list.

All default Quake III Arena shader keywords **should** also work for the below list of games.



Return to Castle Wolfenstein

Q3Map Specific Surface Parameter Shader Keywords

surfaceparm grasssteps
surfaceparm gravelsteps
surfaceparm metalsteps
surfaceparm roofsteps
surfaceparm snowsteps
surfaceparm woodsteps
surfaceparm ladder
surfaceparm carpetsteps (ET???)
surfaceparm glass
surfaceparm playerslick
surfaceparm entityMergable???
surfaceparm monsterSlick
surfaceparm nofog

Are these surfaceparm's?

ai_nopass
ai_nopasslarge
ai_nosight



Return to Castle Wolfenstein: Enemy Territory

Note: unless otherwise noted, all RTCW keywords also work in ET.

Q3Map Specific Shader Keywords

q3map_foliage

Q3Map Specific Surface Parameter Shader Keywords

surfaceparm landmines



Raven Software

Jedi Knights II, Jedi Academy, Soldier of Fortune II

Q3Map Specific Shader Keywords

q3map_flare [shadername]
q3map_flareShader
q3map_material
q3map_onlyVertexLighting



Ritual Entertainment

Q3Map Specific Shader Keywords

8. Shader Effects Creation Tips

This chapter covers some miscellaneous shader and texture creating tips. Note: I've temporarily removed these for now, since I'm not sure how useful they are. A lot of this was covered or should be covered in the Radiant Manual. I'll come back to this chapter when I have time.

Obsidian's Notes:

Stage transparency using blendFunc blend vs. shader transparency.

Texture Creation

If you are familiar with the required tools, creating new assets for use in Quake III Arena is not particularly difficult. As a general rule, you should create new directories for each map with names different from the names used by id. If you are making a map that will be called "H4x0r_D00M", every directory containing new assets for that map should be titled H4x0r_D00M. This is to try and avoid asset directories overwriting each other as the editor and the gameload in assets.

Tools Needed

Any combination of graphic programs and plug-ins that can output a 24 bit MS windows compatible Targa (.tga) or JPEG (.jpg) graphic file. If you plan to make textures that will have an alpha channel component (a 4th 8-bit greyscale channel that is used by the shaders to further manipulate the art), you must have a program that can create 32-bit art with that fourth channel.

Adobe Photoshop has the ability to easily create alpha channels. PaintShop Pro from Corel (v5.0+) can also make an alpha channel by creating a mask and naming it "alpha". Free and open source GIMP also works quite well.

Generally speaking, regardless of the program used, we found it best to do most of the art manipulation of the alpha channel in a separate layer or file and then paste it into the alpha channel before saving.

Setting up Files

The editor and the game program look for assets to be located along the paths set up in your project file. Start by creating a directory for you new textures by creating file folders to make a directory path as follows:

```
quake3\baseq3\textures\[mymapname]
```

The installation of Q3Radiant will create a text document called "shaderlist.txt" in the following path:

```
quake3\baseq3\scripts\shaderlist.txt
```

Q3Radiant will use the contents of this script to grab your new textures for inclusion in the game. The contents of shaderlist.txt document will contain a listing of all the shader documents that were used by id Software to create Quake III Arena.

Since you will obviously want to create your own shaders, you need to put them in separate folders and create a new shader script for them.

If you plan to work on several maps at once and want to distinguish between textures used in each map, simply add additional map names here. For map and mod makers, we **STRONGLY** recommend that any new shader scripts created use the name of the map or mod in the shader file name. We know we can't avoid every incident of files overwriting each other, but we certainly can advise you how to try.

Now, in the scripts directory that you just created, create another text file and call it:

```
[mymapname].shader
```

This file will contain the shader scripts you write to modify a particular texture.

Rules and Guidelines

Follow these rules when creating textures for the Quake III Arena engine:

- Save your textures into your new [map name] directories.
- Don't use the same names that id used for textures. It will cause problems.
- For best quality, save textures without an alpha channel as 24 bit TARGA files. Using JPEG files can save memory space, but at the risk of losing detail and depth in the texture. JPEG files cannot be used for textures requiring an alpha channel.
- Textures containing an alpha channel must be saved as 32 bit TARGA files.
- If a new texture requires no further manipulation, it does not need a shader script.
- Size textures in powers of 2. Example: 8x8, 16x16, 32x32, 64x64 pixels and so on.
- Textures don't need to be square. A 32x256 pixel texture is perfectly acceptable.

The following are some things the id designers learned about textures.

- Create textures in "suites" built around one or two large textures with a number of much smaller supporting detail or accent textures.

- Very large textures are possible, but some video cards compress textures larger than 256x256 pixels.
- Textures are grouped alphabetically by name in the texture display window, so you may want to give suites of textures similar names.
- Use the shader function `qe3_editorimage` to conserve memory when making multiple versions of a single texture (as in the case of a glowing texture with several light values).
- Unless you are creating special effects or textures designed to draw the player's eye to a specific spot, muted, middle value colors work best with the game engine.
- Extremely busy (a lot of fussy detail) textures can break up or form visually unpleasant patterns when seen at distances.

Making the .pk3 File

When you go to distribute your creation to the gaming world, you need to put your newly created map, textures, bot area files, and shader documents into an archive format called a "pk3" file. You do not need to include the `shaderlist.txt` file, since that is only used by the editor. You will need to keep the paths to the various assets the same. So your paths should be something like this:

```
Textures: baseq3/textures/[mymapnamefolder]
Bsp & aas: baseq3/maps/mymapname.bsp, mymapname.aas
Shader scripts: baseq3/scripts/mymapname.shader
```

You need to use a "zip" archiving program ([7-zip](#) for example) to make the pk3 file. Make a zip archive called `mymapname.zip`. Zip all the required assets into a zip archive file (Quake III Arena DOES support compressed pk3 files). Rename the zip archive to `mymapname.pk3`. Put it where the Quake III Arena community can find it.

Alpha Channels

To use some blend modes of `alphaFunc`, you must add an alpha channel to your texture files. Photoshop can do this. Paintshop Pro has the ability to make an alpha channel but cannot work directly in to it. In Photoshop you want to set the type to Mask. Black has a value of 255. White has a value of 0. The darkness of a pixel's alpha value determines the transparency of the corresponding RGB value in the game world. Darker = more transparent.

Care must be taken when reworking textures with alpha channels. Textures without alpha channels are saved as 24 bit images while textures with alpha channels are saved as 32 bit. If you save them out as 24 bit, the alpha channel is erased. Note: Adobe Photoshop will prompt you to save as 32, 24 or 16 bit. Choose wisely. If you save a texture as 32 bit and you don't actually have anything in the alpha channel, Quake III Arena may still be forced to use a lower quality texture format (when in 16 bit rendering) than if you had saved it as 24 bit.

To create a texture that has "open" areas, make those areas black in the alpha channel and make white the areas that are to be opaque. Using gray shades can create varying degrees of opacity/transparency.

```

// Opaque texture with see-through holes knocked in in.
textures/base_floor/pjgratel
{
    surfaceparm metalsteps
    cull none

    // A GRATE OR GRILL THAT CAN BE SEEN FROM BOTH SIDES
    {
        map textures/base_floor/pjgratel.tga
        blendFunc GL_SRC_ALPHA GL_ONE_MINUS_SRC_ALPHA
        alphaFunc GT0
        depthWrite
        rgbGen identity
    }
    {
        map $lightmap
        rgbGen identity
        blendFunc GL_DST_COLOR GL_ZERO
        depthFunc equal
    }
}

```

The alpha channel can also be used to merge a texture (including one that contains black) into another image so that the merged art appears to be an opaque decal on a solid surface (unaffected by the surface it appears to sit on), without actually using an alpha function. The following is a very simple example:

Start with a TGA file image. In this case, a pentagram on a plain white field (figure 1A). The color of the field surrounding the image to be merged is not relevant to this process (although having a hard-edged break between the image to be isolated and the field makes the mask making process easier). Make an alpha channel. The area of the image to be merged with another image is masked off in white. The area to be masked out (notused) is pure black (figure 1B). The image to be merged into is greenfloor.tga (figure 1C).

Make a qer_editorimage of greenfloor.tga. This is placed in the frame buffer as the map image for the texture. By using GL_SRC_ALPHA as the source part of the blend equation, the shader adds in only the non-black parts of the pentagram. Using GL_ONE_MINUS_SRC_ALPHA, the shader inverts the pentagram's alpha channel and adds in only the non-black parts of the green floor.

In a like manner, the alpha channel can be used to blend the textures more evenly. A simple experiment involves using a linear gradient in the alpha channel (white to black) and merging two textures so they appear to cross fade into each other.

A more complicated experiment would be to take the pentagram in the first example and give it an aliased edge so that the pentagram appeared to fade or blend into the floor.

Advanced Decal Tricks 101

Multiplicative Decals

Instead of doing blended or additive decals, sometimes the best effect is actually a multiply:

```
blendFunc GL_DST_COLOR GL_ZERO
```

The source texture is white with yellow/blue (or whatever) arrow on it. Combined with `polygonOffset` and perhaps a sort key, this can be a great way to get a decal effect that works okay with bullet marks and player shadows.

Inverse Multiplicative Shadows

While the above trick works well 95% of the time, sometimes you need to use fog. In order for a multiplicative decal to face out properly in fog, it has to be inverted, and using an inverse `blendFunc`:

```
blendFunc GL_ZERO GL_ONE_MINUS_SRC_COLOR
```

The source image will be negative of the above image, (white = black, blue = yellow, etc). This is the trick that the player shadow mark shader use.



```
textures/obsidian-blastburn_decals/p-geisha
{
    noPicMip
    //draws polygons of this shader just above coplanar surface
    polygonOffset
    //prevents bounce from affecting this shader
    q3map_bounceScale 0
    surfaceparm detail
    surfaceparm nomarks
    surfaceparm nonsolid
    {
        map textures/obsidian-blastburn_decals/p-geisha.tga
```

```

        //inverse multiplicative blend (TGA channels inverted)
        blendFunc GL_ZERO GL_ONE_MINUS_SRC_COLOR
    }
}

```

Using _decal Entities

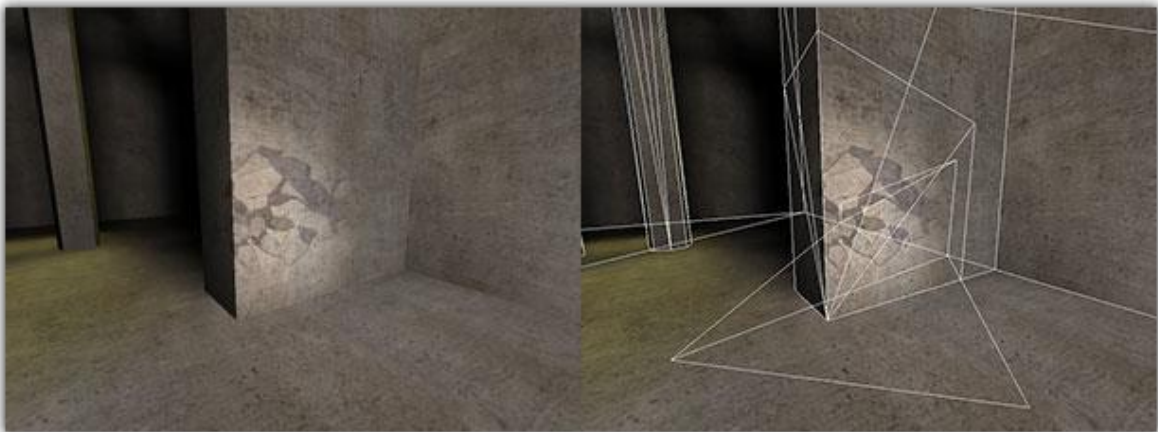
Sometimes laying out perfect patch meshes aligned with geometry is too much of a pain in the ass. Enter _decal entities. Add the following to your entities.def (Enemy Territory mappers should already have it):

```

/*QUAKED _decal (0 1.0 0) ?
----- KEYS -----
"target" : the name of the entity targetted at for projection
----- SPAWNFLAGS -----
(none)
----- NOTES -----
Compiler-only entity that specifies a decal to be projected. Should contain
1 or more patch meshes (curves) and target an info_null entity. The
distance between the center of the _decal entity and the target is the axis
and distance of projection.
*/

```

Make a simple patch mesh (it can be bent, but the quads in Radiant must be rectangular), select it and use the right-click context menu to turn it into a _decal entity. Target an info_null below the _decal entity and compile. Everything between the decal patch and the info_null will have a decal projected onto it.



Make sure you compile with a recent (post-2.5) version of Q3Map2 with _decal support.

It will project onto terrain, brushes, models, patches, whatever. To suppress decals on particular shaders, use surfaceparm nomarks.

Happy decaling!

-ydnar

Appendix A – Triggerable Shader Entities

By TTimo, 31.08.01

The targetShaderName and targetShaderNewName keys can be used with any entity that supports the target key (the entity instance does not actually have to use the target key for these new keys to work). If both are defined, then when the entity decides to activate its targets, all shaders/textures in the map that were originally the same name as the targetShaderName value, will be changed to the targetShaderNewName value.

For example this would make it look like the red light shader is "turning on":

```
"targetShaderName" "textures/proto2/redlight_off"  
"targetShaderNewName" "textures/proto2/redlight_on"
```

And this would turn it back off:

```
"targetShaderName" "textures/proto2/redlight_off"  
"targetShaderNewName" "textures/proto2/redlight_off"
```

Note that the ORIGINAL shader name is used in both instances, not whatever it happens to be currently. Also, of course, this will happen globally. If the mapper wanted to affect only a certain set of red lights, he/she would need to make a unique shader name to be used with that set.

The code that supports these keys is in G_UseTargets in g_utils.c

Appendix B – Terrain

Q3Map2 terrain has changed radically since Quake III: Team Arena first hit the shelves. Alphamapped terrain shaders are difficult to set up, limiting and lacks the ability for the mapper to fine tune the effect. The new terrain improvements in Q3Map2 have removed these difficulties, allowing the mapper to fully exploit terrain shaders for breathtaking outdoor scenes. This section will assume that you are already familiar with the method of setting up a standard alphamapped terrain as described in the Terrain Construction for Quake 3 Engine Games manual by Paul Jaquays, and will focus on the new enhanced features of Q3Map2 terrain.



General Changes

Q3Map2 terrain has changed dramatically in terms of not only visually, but also in terms of how it is processed by the compiler. To ensure compatibility, there needs to be a few changes. Foremost, make sure that all terrain shader files are listed in the shaderlist.txt file.

Before using Q3Map2 alphamapped terrain, it must be noted that terrain is now handled completely different than before. Q3Map2 no longer looks for the word terrain in the shader name to determine whether or not it is an indexed shader. Instead it looks for q3map_indexed or q3map_terrain. For this reason, the alphamap terrain shaders have been slightly modified. If using Q3Map2 alphamaps, make sure that your "textures/common/terrain" and "textures/common/terrain2" shaders have been updated to the following:

Script: "common" terrain shaders

textures/common/terrain

```
{
    q3map_terrain
    surfaceparm nodraw
    surfaceparm nolightmap
    surfaceparm nomarks
}
```

textures/common/terrain2

```
{
    q3map_terrain
    qer_editorimage textures/common/terrain.tga
    surfaceparm dust
    surfaceparm nodraw
    surfaceparm nomarks
    surfaceparm nolightmap
}
```

Base Shader

To simplify the terrain shader writing process, a base shader can be used as a template with subclassed shaders referencing it. This makes use of the `q3map_baseShader` directive. The base shader can consist of all `q3map_*` directives and must be processed before all other terrain shaders referencing it. The use of a base shader is recommended for all terrain shaders.

tcMod Functions

`q3map_tcMod` can be used on terrain shaders to minimize any obvious texture tiling, particularly `q3map_tcMod rotate`. In some situations `q3map_tcMod scale` or the stage driven `tcMod scale` may be used to scale the textures - though typically `tcGen` texture projection functions are usually the preferred method.

tcGen Functions

In most situations, terrain textures will need to be projected onto the terrain mesh. This is achieved by using either `q3map_tcGen` function or the stage specific `tcGen`. Any `tcGen` functions will overwrite any `tcMod scale` functions.



Lightmapped Terrain

The foremost improvement to Q3Map2 terrain is the ability to use lightmaps as a lighting system rather than basic vertex lighting. This allows terrain to not only cast detailed shadows onto itself, but also allows other map geometry to cast shadows onto the terrain as well. The additional rendering pass of the lightmap creates a slight performance hit as a price for the improved visuals. Several new `q3map_*` directives are typically used to tweak the terrain lightmap:

q3map_nonPlanar

Required on all terrain shaders, q3map_nonPlanar allows shadows to be cast across non-planar edges. This fixes a problem where lightmapped terrain would not light properly across uneven surfaces.

q3map_lightmapAxis

Optionally used, q3map_lightmapAxis can be used to specify the axis in which the lightmap is projected from. This is typically used on normal triangle-quad souped terrain and is set implicitly to "q3map_lightmapAxis z" with q3map_terrain. It is not recommended for terrain with caves or undersides.

q3map_shadeAngle

Most terrain artists will choose to enable triangle edge shadow smoothing to reduce the appearance of sharp shadow edges across the terrain mesh. q3map_shadeAngle specifies the triangle edge angle at which the light will be diffused. For best results, it is recommended that you start with low values and tweak the angle parameter in small increments until a satisfactory result is produced. Overly high values will wash out shadow details.

q3map_lightmapMergable

Setting q3map_lightmapMergable merges all terrain into one seamless lightmap, reducing the appearance of artifacts spanning across separate lightmap images.

q3map_lightmapSampleSize

Lightmapped terrain can be very memory intensive as Q3Map2 has to typically generate a large amount of lightmap data. q3map_lightmapSampleSize can be used to reduce the amount of lightmap memory used by limiting the resolution of the lightmap images. Similarly, it can also be used to increase the lightmap resolution at the cost of memory.

q3map_lightmapSampleOffset

If experiencing lightmap splotches over the surface of terrain lightmaps, q3map_lightmapSampleOffset can be used to fix this. For best results, start with small numbers (default 1.0) and slowly increase this value until the splotches disappear. Overly high values will cause unsatisfactory results.



AlphaMod Dotproduct Terrain

AlphaMod dotproduct terrain is a revolutionary way in which Q3Map2 terrain is blended. It removes the need for an alphamap and metashader and automatically determines the blending of terrain depending on the vertex normals of the terrain mesh. Equally important, this new system removes the need of a triangle-quad grid, allowing the designer to create a terrain mesh using any triangle shape or size.



AlphaMod Volume Terrain

Following the automatic generation of dotproduct blending, alphaMod volume terrain blending was developed to give direct control of the alpha blending back to the designer. Using this system the designer can modify the exact location and amount of blending to occur, directly from within the editor.

Appendix C – Foghull

Most recent Quake III engine games are programmed with a feature known as distance clipping used to clip (cull) map geometry beyond a user set distance. At the time of Quake III Arena's release, distance clipping was not an available feature. The Q3Map2 foghull feature was designed to simulate true distance clipping for Quake III Arena and Team Arena games.



What is a Foghull?

Farplane distance clipping is a feature used to cull (remove) the drawing of polygons beyond a certain distance from the player in an attempt to improve performance. It is typically used on large, open terrain maps with little vis-blocking structures. Through the use of distance clipping a maximum vis distance is set, which provides the culling of rendered polygons. Fog is used to hide the effect of polygons appearing and disappearing by obscuring the maximum distance that the player can see.

Since distance clipping is not a feature natively built into Quake III Arena, using distance culling would result in a hall of mirrors (HOM) effect where the culled geometry begins, since nothing is being drawn in the frame buffer. To compensate for this, the foghull feature uses a series of six skybox images that are drawn in place of the absent culled geometry, thus preventing the HOM effect.



Skybox Images

The skybox images used with the foghull feature should never be actually seen since the idea is to use the fog to obscure the maximum distance that the player can see. It only exists to prevent the HOM effect. To pull this off in a convincing manner, the skybox images should be six identical 8x8 pixel (to save on texture memory) textures each filled with a flat color matching the exact color of the fog. The six skybox images must be named in accordance to the [skyParms farbox](#) convention, using the `_ft`, `_rt`, `_bk`, `_lf`, `_up`, `_dn` suffixes.



Foghull Shaders

Two shaders are required when using the foghull feature, a fog volume shader and a skybox shader, both of which are simple, standard shaders.

The color of the fog used must match the color used in the skybox images. Any decent image editing software will give you the three RGB color values of your skybox images, which must be normalized by dividing by 255. As with any other fog shader, the [fogParms](#) and [surfaceparm fog](#) keywords must be present.

Script: Fog Volume Shader

```
textures/env/fog1024
{
    fogParms ( 0.8 0.8 0.8 ) 1024      //Normalized RGB, distance to opaque

    surfaceparm fog                    //Must be used
    surfaceparm nolightmap
    surfaceparm nonsolid
    surfaceparm trans

    qer_editorImage textures/sfx/fog_grey.tga
    qer_trans 0.4
}
```

The [skyParms](#) and [surfaceparm sky](#) keywords must be used to create the skybox "hull" of the map. The [skyParms farbox](#) value must point to the base name of the skybox images (sans suffix). Optional surface emitted sun lighting can be added using [q3map skylight](#) with [q3map sun](#) or [q3map sunExt](#) (see [Appendix I: Light Emitting Shaders](#)).

Script: Skybox Shader

```
textures/skies/foghullsky
{
    skyParms textures/skies/foghullsky 0 -      //farbox cloudheight nearbox

    surfaceparm nolightmap
    surfaceparm nonsolid
    surfaceparm sky                            //Must be used
    surfaceparm trans
}
```



Entity Key Value Pairs

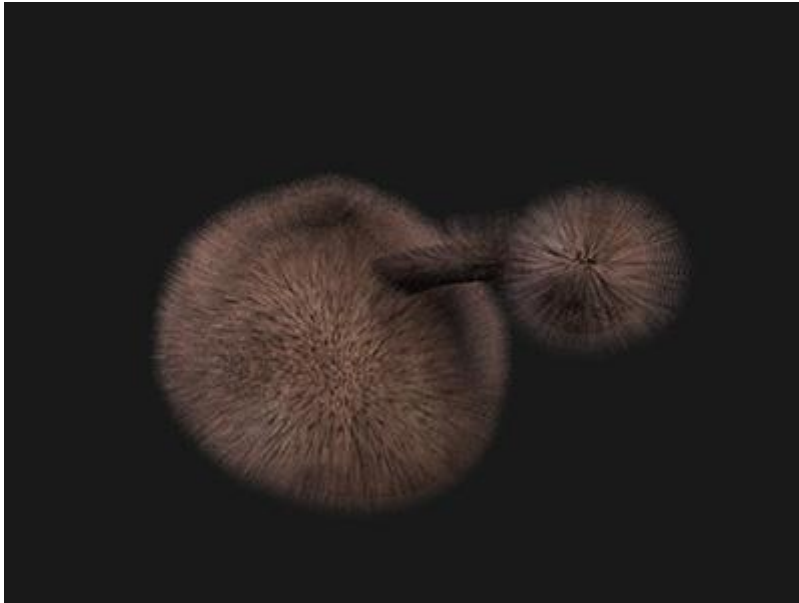
To activate the foghull feature, the `_foghull` and `_farplanedist` worldspawn entity key/value pairs must be set in the Entity Inspector (in Radiant, select any non-entity brush and press "N").

The `_foghull` key's value works similar to that of terrain entities. It must point to the name of the skybox shader, minus the standard "textures/" prefix. For the skybox shader example above with the shader name "textures/skies/foghullsky", you would use a `_foghull` value of "skies/foghullsky".

The `_farplanedist` value is simply the distance at which polygons will begin to get culled. One important note is that the `_farplanedist` value must be greater than the distance to opaque value set in the fog shader. In the above fog shader, the distance to opaque value was set to 1024 game units. The `_farplanedist` value must therefore be greater than 1024, otherwise the HOM effect will appear.

Appendix D – Fur

"Fur" shaders allow a surface to replicate itself or another shader above its surface in multiple layers. Care should be taken when using fur shaders, they can quickly cause drops in performance.



q3map_cloneShader

[q3map_cloneShader](#) allows the base shader to inherit the target shader's properties and appearance. Beware not to reference another cloning shader or itself as this can lead to an infinite loop. For fur, the base shader is the texture applied to the polygon surface.

```
// base texture
textures/fur/pink_base
{
    // points to the fur shader (see below)
    q3map_cloneshader textures/fur/pink_fur
    {
        map $lightmap
    }
    {
        map textures/fur/pink_base.tga
        blendFunc GL_DST_COLOR GL_ZERO
    }
}
```

q3map_fur

[q3map_fur](#) generates additional surfaces above the base shader. The `q3map_fur` directive takes a few values: layers, offset and fade. Layers controls the number of surfaces generated (start with low values, high values will very quickly cause a reduction in performance). Offset controls the distance between layers. Fade controls how much each additional layer fades in addition to its previous layer. The fur shader shouldn't be applied to surface geometry as it is implicitly generated above the base shader.

```
// fur texture
textures/fur/pink_fur
{
    q3map_lightimage textures/fur/pink_fur.q3map.tga

    q3map_notjunc
    q3map_nonplanar
    q3map_bounce 0.0
    q3map_shadeangle 120

    // format: q3map_fur
    q3map_fur 8 1.25 0.1

    surfaceparm trans
    surfaceparm pointlight
    surfaceparm alphashadow
    surfaceparm nonsolid
    surfaceparm noimpact

    nomipmaps
    {
        map textures/fur/pink_fur.tga
        //alphaFunc GE128
        blendFunc GL_SRC_ALPHA GL_ONE_MINUS_SRC_ALPHA
        rgbGen vertex
    }
}
```

Appendix E – Cel Shading

Cel shading is the technique used to make your maps render like cartoons, with black (or other color) solid outlines tracing hard edges.

In order for celshading to work properly, you must use -meta when compiling the BSP. The funny little nonplanar/shadeangle/invert/offset nonsense in cel.shader is what makes it work properly. You'll know when it's working when it starts taking about 10-20x as long to make the BSP.

```
// ink shader for maps
// to use, add "cel" to shaderlist.txt
// add a "_celshader" key to worldspawn entity with a value of "cel/ink"

textures/cel/ink
{
    qer_editorimage gfx/colors/black.tga

    q3map_notjunc
    q3map_nonplanar
    q3map_bounce 0.0
    q3map_shadeangle 120
    q3map_texturesize 1 1
    q3map_invert                //inverts drawing surface for backfacing
hull
    q3map_offset -2.0           //offsets surface by the specified distance

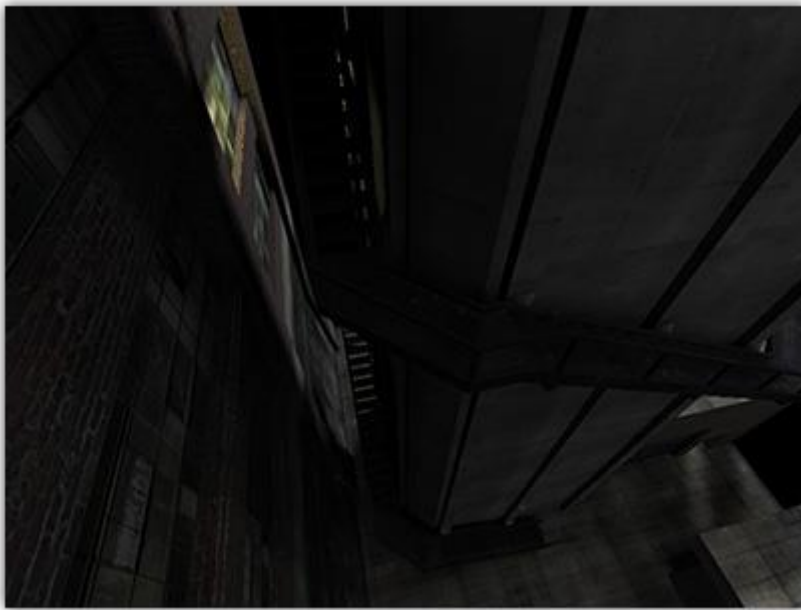
    surfaceparm nolightmap
    surfaceparm trans
    surfaceparm nonsolid
    surfaceparm nomarks

    sort 16

    {
        map gfx/colors/black.tga
        rgbGen identity
    }
}
```

Appendix G – Lightstyles

Q3Map2 light styles are a way to simulate flickering/blinking dynamic lights by modulating values between up to 3 different dynamic lightmap styles per surface. This feature was added in Q3Map2 to support Quake 3 and RTCW. SOF2/JK2 already have native support for light styles. Light styles will only affect lightmapped objects, it has no effect on vertex lit objects and the light grid.



Worldspawn Keys

To create some flickering lights, we need some waveform functions. These are set on the worldspawn entity in Radiant's Entity Inspector using two new available key/value pairs: `_styleNrgbGen` and `_styleNalphaGen` keys, where "N" is the style index number, an integer between 1 and 31. Both keys will take [standard waveform functions](#) as values. As an example:

```
_style1alphaGen    wave sin .5 .3 .25 1.5
_style1rgbGen      wave noise 0.5 1 0 5.37
_style2alphaGen    wave sin .8 .3 .25 1.5
_style2rgbGen      wave square -.3 1.3 0 5.3
classname          worldspawn
```

Lights

Next, we need to associate your lights with the style index numbers that were set in the worldspawn. You can add light styles to either light entities or light emitting shaders.

Light Entities

With a light entity selected, open up the Entity Inspector and add a "style" key. Use a value between 1 and 31 matching the style index number previously set in the worldspawn.

Light Emitting Shaders

You can also use `q3map_lightStyle N`, where "N" is a value between 1 and 31 matching the style index number set in the worldspawn, on light-emitting shaders to have them emit styled light.

```
textures/slstyle/light
{
    q3map_surfacelight 3700
    q3map_lightStyle 1    // sets style index #1
    {
        map $lightmap
        rgbGen identity
    }
    q3map_styleMarker    // note: after the $lightmap stage
    {
        map textures/slstyle/light.tga
        blendFunc GL_DST_COLOR GL_ZERO
        rgbGen identity
    }
    {
        map textures/slstyle/light.blend.tga
        blendfunc GL_ONE GL_ONE
    }
}
```

Lightmapped Surfaces

For any shaders that may be hit by a styled light, you'll need to add `q3map_styleMarker` after the lightmap stages and before the texture stages so Q3Map2 can properly create the fake lightmap stages. For masked textures where a `depthFunc equal` is required, add `q3map_styleMarker2`.

Shaders with lightmaps after texture passes will look odd. This may change in the future. Try to rearrange your shaders, if possible, to have lightmaps first.

```
textures/slstyle/plywood2sided
{
    cull none
    qer_editorImage textures/slstyle/plywood-2-tone.tga
    {
        map $lightmap
        rgbGen identity
    }
    q3map_styleMarker    // note: after the $lightmap stage
    {
        map textures/slstyle/plywood-2-tone.tga
        blendFunc GL_DST_COLOR GL_ZERO
        rgbGen identity
    }
}
```

Compiling

Compile your map with Q3Map 2.5.5-test-6 or later. Be sure to use the -nocollapse switch in the -light phase. This is important, because styled lights generate shaders, and this minimizes the number of unique shaders.

In game, you might get a warning message in the console, "WARNING: reused image *lightmap4 with mixed glWrapClampMode parm", which you can safely ignore.

References

- [Q3Map 2.5.5-test-6 \(lightstyles\)](#), ydnar 2003
- [slstyle demo map \(mirror\)](#), ydnar 2003.

Appendix I – Light Emitting Shaders

Q3Map2 surface light and sky shaders are quite different than the original *Quake III* shaders. As new lighting algorithms were introduced, new shader keywords were created to accompany or replace the original keywords. This section will illustrate the differences between these shaders.



Surface Lights

nothing here yet

Mention:

*q3map_lightRGB red green blue



Skies

Originally, sky shaders were just very large surface lights that casted parallel directional lighting. With Q3Map2, there are a few differences that sets sky shaders apart from surface lights. First of all, we'll take a look at how the pre-Q3Map2 shaders were set up:

Script: Pre-Q3Map2 sky shaders

```
textures/shadermanual/sky
{
    skyparms textures/shaderlab_terrain/env/sky 1024 -           //farbox cloudheight nearbox

    q3map_lightImage textures/shaderlab_terrain/sky_clouds.tga

    q3map_sun 1 1 1 140 -35 25      //red green blue intensity degrees elevation
    q3map_lightSubdivide 256        //sets a pointlight every 256 game units
    q3map_surfaceLight 200          //emits 200 units of light

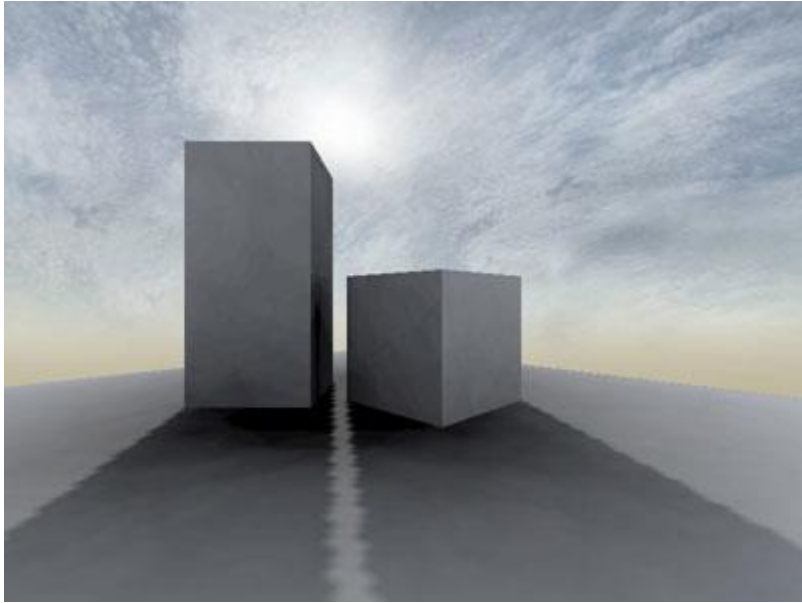
    surfaceparm sky                //flags compiler that this is sky
    surfaceparm noimpact
    surfaceparm nolightmap
    surfaceparm nodlight

    nopicmip
    nomipmaps

    qer_editorimage textures/shaderlab_terrain/sky_clouds.tga

    {
        map textures/shaderlab_terrain/sky_clouds.tga
        tcMod scale 3 3
        //tcMod scroll 0.005 -0.0125
        rgbGen identityLighting
    }
    {
        map textures/shaderlab_terrain/sky_arc_masked.tga
        blendFunc GL_ONE_MINUS_SRC_ALPHA GL_SRC_ALPHA
        tcMod transform 0.25 0 0 0.25 0.1075 0.1075
        rgbGen identityLighting
    }
}
```

Keep in mind that this is a generalized shader, and that there can be a lot of different variations to yield different effects. Take a look at some of the original *Quake III Arena* shaders for more examples. In this screenshot (compiled with LIGHT -fast, viewed with /r_lightmap 1), the effect isn't bad, but the shadows are a bit jagged.



Q3Map2 sky shaders improves on the way lightmaps are calculated, improving both quality and compiler performance. This is essentially the same shader with some small changes:

Script: Using q3map_skyLight

```
textures/shadermanual/sky
{
    skyparms textures/shaderlab_terrain/env/sky 1024 -

    q3map_lightImage textures/shaderlab_terrain/sky_clouds.tga

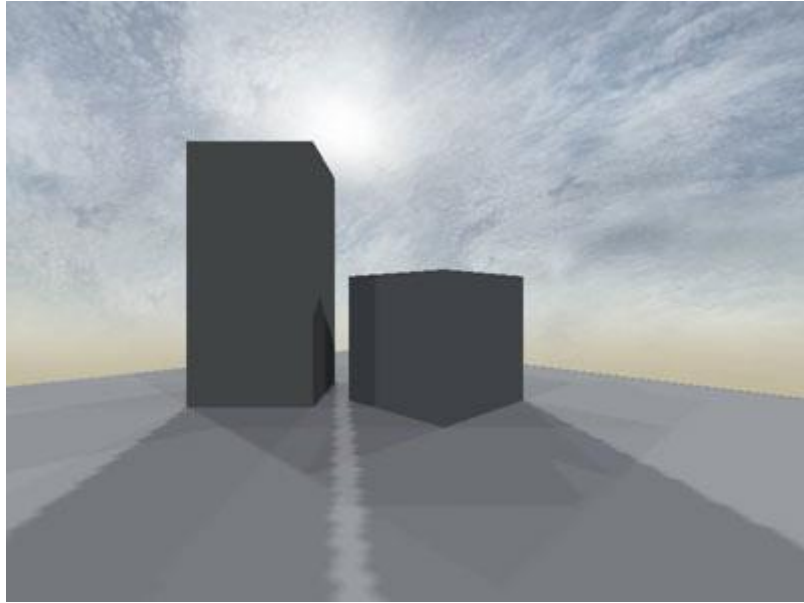
    q3map_sun 1 1 1 140 -35 25
    q3map_skylight 100 3 //amount iterations

    surfaceparm sky
    surfaceparm noimpact
    surfaceparm nolightmap
    surfaceparm nodlight

    nopicmip
    nomipmaps

    qer_editorimage textures/shaderlab_terrain/sky_clouds.tga
    {
        map textures/shaderlab_terrain/sky_clouds.tga
        tcMod scale 3 3
        //tcMod scroll 0.005 -0.0125
        rgbGen identityLighting
    }
    {
        map textures/shaderlab_terrain/sky_arc_masked.tga
        blendFunc GL_ONE_MINUS_SRC_ALPHA GL_SRC_ALPHA
        tcMod transform 0.25 0 0 0.25 0.1075 0.1075
        rgbGen identityLighting
    }
}
```

What we've done here is replace *q3map_lightSubdivide* and *q3map_surfacelight* with *q3map_skylight* which yields more uniform shadows at a fraction of the compile time. However, this also generates the "stadium light" effect - producing some unwanted shadows. We'll fix this later.



To solve the problem with jagged shadow edges, we can smooth out the shadows by blurring the lightmap. Depending on the type of lighting that you want to achieve for the sun (a cloudy day, for example), you can create a penumbra (half-shadow) effect using *q3map_sunExt*. This simulates the way sunlight bounces in certain conditions, creating a slight "jittering" effect. This is the same shader again with *q3map_sunExt*.

Script: Using *q3map_sunExt*

```
textures/shadermanual/sky
{
    skyparms textures/shaderlab_terrain/env/sky 1024 -

    q3map_lightImage textures/shaderlab_terrain/sky_clouds.tga

    q3map_sunExt 1 1 1 140 -35 25 3 16 //adds deviance and samples
    q3map_skylight 100 3

    surfaceparm sky
    surfaceparm noimpact
    surfaceparm nolightmap
    surfaceparm nodlight

    nopicmip
    nomipmaps

    qer_editorimage textures/shaderlab_terrain/sky_clouds.tga

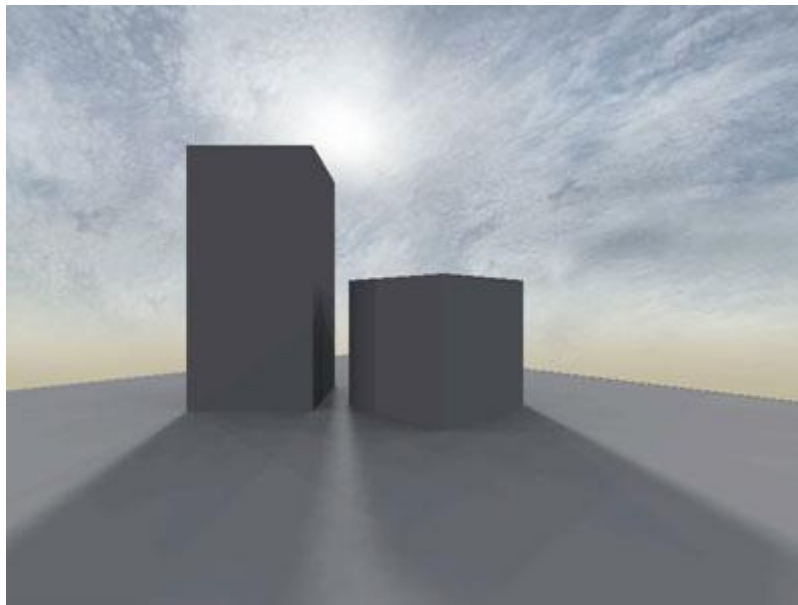
    {
        map textures/shaderlab_terrain/sky_clouds.tga
        tcMod scale 3 3
        //tcMod scroll 0.005 -0.0125
        rgbGen identityLighting
    }
    {
        map textures/shaderlab_terrain/sky_arc_masked.tga
        blendFunc GL_ONE_MINUS_SRC_ALPHA GL_SRC_ALPHA
        tcMod transform 0.25 0 0 0.25 0.1075 0.1075
        rgbGen identityLighting
    }
}
```

In the following screenshot, you can see that the jagged shadow edges are gone.

As mentioned above, you may be faced with problems involving the "stadium lights" effect when using *q3map_skyLight*. We can eliminate this problem by using higher values for the *q3map_sunExt* samples and *q3map_skyLight* iterations parameter, but at the cost of a higher compile time. For example, *q3map_sunExt 1 1 1 140 -35 25 3 32* and *q3map_skyLight 100 6*.

Note:

Since the time that these screenshots were taken, the skylight subdivision code has been greatly improved (Q3Map2 2.5.14) for far more uniform lighting and faster compiles, so using higher iteration values can result in better quality, reducing the "stadium light" effect and without necessarily increasing compile times.



A faster approximate alternative of getting rid of the "stadium lights" effect is to use *q3map_lightmapFilterRadius*.

Script: Using q3map_lightmapFilterRadius

```
textures/shadermanual/sky
{
    skyparms textures/shaderlab_terrain/env/sky 1024 -

    q3map_lightImage textures/shaderlab_terrain/sky_clouds.tga

    q3map_sunExt 1 1 1 140 -35 25 3 16
    q3map_lightmapFilterRadius 0 8 //self other
    q3map_skyLight 100 3

    surfaceparm sky
    surfaceparm noimpact
    surfaceparm nolightmap
    surfaceparm nodlight

    nopicmip
    nomipmaps

    qer_editorimage textures/shaderlab_terrain/sky_clouds.tga

    {
        map textures/shaderlab_terrain/sky_clouds.tga
        tcMod scale 3 3
    }
}
```

```

        //tcMod scroll 0.005 -0.0125
        rgbGen identityLighting
    }
    {
        map textures/shaderlab_terrain/sky_arc_masked.tga
        blendFunc GL_ONE_MINUS_SRC_ALPHA GL_SRC_ALPHA
        tcMod transform 0.25 0 0 0.25 0.1075 0.1075
        rgbGen identityLighting
    }
}

```

The *self* and other parameters are the amount of filtering applied on the lightmap in world units. The *self* value is always set to "0" on sky shaders since skies don't have lightmaps. The *q3map_lightmapFilterRadius* directive should always be placed before any light-related directives that you want it to affect. In our case, we placed it after *q3map_sunExt* and before *q3map_skyLight* so that it filters the stadium lights, but won't blur the sun shadows which are already jittered. This produces very similar results without the long compile times.

screenshot here

For additional information, see this thread:

[Shader Lighting Experiment](#)



Lighting Effects

Here are some additional features that you can use to create special lighting effects:

- *multiple suns by adding more than one *q3map_sun* or *q3map_sunExt* to shader
- *stuff about compiler switches
- *-skyfix
- *_skybox entity does not work with *surfaceLight*, must use *skyLight*