

Quake III Arena

Shader Manual

Revision #12

By Paul Jaquays and Brian Hook

(with additional material by John Carmack, Christian Antkow, Kevin Cloud, & Adrian Carmack)

1 Preface: Making Your Own Shaders

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. 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.

2 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 was it intended to be one.

2.1 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 Q3MAP keywords or “Surface Parameters”, which follow the first bracket and a single bracketed “stage”:

```
textures/liquids/lava
{
    deformVertexes wave sin 0 3 0 0.1
    tessSize 64
    {
        map textures/common/lava.tga
    }
}
```

2.2 Shader Name & File Conventions

The first line is the shader name. Shader names can be up to 63 characters long. The names are often a mirror of a pathname to a .tga file without the extension or basedir (/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 world, like “projectionShadow” or “viewBlood”.

Shaders that are used on characters or other polygon models need to mirror a .tga file, 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 a .tga file, but the “qer_editorimage” shader parameter can force the editor to use an arbitrary image for display.

Shader pathnames 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.

2.3 Shader Types

The keywords that affect shaders are divided into two classes. The first class of keywords are global parameters. Some global parameters (“**surfaceparms.**” And all “**q3map_**” keywords) are processed by Q3MAP, and change physical attributes of the surface that uses the shader. These attributes can affect the player. To see changes in these parameters one must re-bsp the map.

The remaining global keywords, and all 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).

Shader keywords are *not* case sensitive.

IMPORTANT NOTE: 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).

2.4 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.

2.4.1 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.

2.4.2 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 Q3:A 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.

2.4.3 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.

2.4.4 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.

2.4.5 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.

2.4.6 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.

2.4.7 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 Q3Radiant 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.

2.4.8 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 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 a choppy looking wave forms.

Square: A square wave 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.

3 General Shader Keywords

IMPORTANT NOTE: 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 Q3MAP.

3.1 *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 cloud layers. 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 on top 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 cloud layers, 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.

Example: Sky script

```
textures/skies/xtoxicssky_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 scroll 0.1 0.1
        tcMod scale 3 2
    }
    {
        map textures/skies/intelredclouds.tga
        blendFunc add
        tcMod scroll 0.05 0.05
        tcMod scale 3 3
    }
}
```

3.2 *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 front* if this keyword is not specified. However for items that should be inverted then the value *back* should be used. To disable culling, the value *disable* or *none* should be used. Only one cull instruction can be set for the shader.

3.2.1 *cull front*

The front or “outside” of the polygon is not drawn in the world. This is the default value. It is used if the keyword “**cull**” appears in the content instructions without a *<side>* value or if the keyword *cull* does not appear at all in the shader.

3.2.2 *cull back*

Cull back removes the back or “inside” of a polygon from being drawn in the world.

3.2.3 *cull disable, cull 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.

3.3 *deformVertexes*

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.

3.3.1 *deformVertexes wave* <div> <func> <base> <amplitude> <phase> <freq>

Designed for water surfaces, modifying the values differently at each point. It accepts the standard wave functions of the type **sin**, **triangle**, **square**, **sawtooth** or **inversesawtooth**. The "div" parameter is used to control the wave "spread" – a value equal to the tessSize of the surface is a good default value (tessSize is subdivision size, in game units, used for the shader when seen in the game world) .

3.3.2 *deformVertexes normal* <div> <func> <base> <amplitude ~0.1~0.5> <frequency ~1.0~4.0>

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.

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.

3.3.3 *deformVertexes bulge* <bulgeWidth> <bulgeHeight> <bulgeSpeed>

This forces a bulge to move along the given s and t directions. Designed for use on curved pipes.

Specific parameter definitions for deform keywords:

<div> This is roughly defined as the size of the waves that occur. It is measured in game units. Smaller values create a greater density of smaller wave forms occurring in a given area. Larger values create a lesser density of waves, or otherwise put, the appearance of larger waves. To look correct this value should closely correspond to the value (in pixels) set for tessSize (tessellation size) of the texture. A value of 100.0 is a good default value (which means your tessSize should be close to that for things to look "wavelike").

<func> This is the type of wave form being created. Sin stands for sine wave, a regular smoothly flowing wave. Triangle is a wave with a sharp ascent and a sharp decay. It will make a choppy looking wave forms. A square wave is simply on or off for the period of the frequency with no in between. The sawtooth wave has the ascent of a triangle wave, but has the decay cut off sharply like a square wave. An inversesawtooth wave reverses this.

<base> This is the distance, in game units that the apparent surface of the texture is displaced from the actual surface of the brush as placed in the editor. A positive value appears above the brush surface. A negative value appears below the brush surface. An example of this is the Quad effect, which essentially is a shell with a positive base value to stand it away from the model surface and a 0 (zero) value for amplitude.

<amplitude> The distance that the deformation moves away from the base value. See Wave Forms in the introduction for a description of amplitude.

<phase> See Wave Forms in the introduction for a description of phase)

<frequency> See Wave Forms in the introduction for a description of frequency)

Design Note: 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 have massive swells (big div values) that rose and fell dramatically (big amplitude values). While a small, quiet pool may move very little.

3.3.4 **deformVertexes move** <x> <y> <z> <func> <base> <amplitude> <phase> <freq>

This keyword is used to make a brush, curve patch or md3 model appear to move together as a unit. The <x> <y> and <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> and <freq> values are the same as found in other wave form 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 Note: 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.

3.3.5 **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 an .md3 model can have the **autosprite** effect on it. **The brush face containing a texture with this shader keyword must be square.**

Design Note: This is best used on objects that would appear the same regardless of viewing angle. An example might be a glowing light flare.

3.3.6 **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.

3.4 **fogparms** <red value> <green value> <blue value> <distance to Opaque>

Note: you must also specify "surfaceparm fog" to cause q3map to identify the surfaces inside the volume. Fogparms only describes how to render the fog on the surfaces.

<red value> <green value> <blue value> 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.

<distance to opaque> 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.

3.5 *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.

3.6 *nomipmap:*

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.

3.7 *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.

3.8 *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.

3.9 *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 blendfuncs)

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

4 Q3MAP 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.

4.1 *tessSize* <amount>

For consistency's sake, this really should have been called `q3map_tessSize`. But it wasn't. The `tessSize` shader 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 `q3map` processing, so maps must be reprocessed for changes to take effect.

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

4.2 *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 (see *sort* above) or appearance when seen from the inside.

4.3 *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 `q3map` 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.

4.4 *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 light source (sun, moon, hellish fire, etc.). This is only processed during the lighting phase of `q3map`.

<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> is 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> is the angle relative to the directions on the map file. A setting of 0 degrees equals east. 90 is north, 180 is west and 270 is south.

<elevation> is 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 NOTE: Sky shaders should probably still have a `q3map_surfacelight` 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` 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.

Design Trick: Not certain what color formula you want to use for the sun's light? Try this. Create a light entity. Use the Q3Radiant 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.

4.5 `q3map_surfacelight` <light value>

The texture gives off light equal to the <light 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 `qer_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.

4.6 `q3map_lightimage` <texturepath/texturename>

The keyword `q3map_lightimage` generates lighting from the average color of the TGA image specified by the `q3map_lightimage`.

The keyword sequence for generating light on a `q3map_surfacelight` should be ordered as follows:

- 1) `q3map_lightimage` ; (the texture providing the light and the color of the light)
- 2) `qer_editorimage` ; (the editor-only image used to select the source map for the texture)
- 3) the average color of the light emitted from the shader is calculated from the `lightimage`.)

The reason `q3map_lightimage` is specified for the light in the example below, is because the blend map is predominantly yellow, and the designer wanted more yellow light to be emitted from the light.

Example: Taking light from another source texture

```
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
    //base 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
    }
    {
        map textures/gothic_light/ironcrosslt2.blend.tga
        blendFunc add
    }
}
```

4.7 *q3map_lightsubdivide* <value>

This allows the user to define how large, or small to make the subdivisions (triangles) in a textured surface, particularly aimed at light-emitting textures like skies. It defaults to 120 game units, but can be made larger (256 or 512) for sky boxes or smaller for light surfaces at the bottoms of cracks. This can be a dominant factor in processing time for q3map lighting.

4.8 *surfaceparm* <parm>

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

4.8.1 *alphashadow*

This keyword applied to a texture on a brush, patch or model will cause the lighting phase of the q3map process to use the texture's alpha channel as a mask for casting static shadows in the game world.

Design Note: 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.

4.8.2 *areaportal*

A brush marked with this keyword functions as an area portal, a break in the Q3MAP 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 cluster portal. The brush must touch all the structural brushes surrounding the **areaportal**.

4.8.3 *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 **areaportal**.

4.8.4 *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.

4.8.5 *flesh*

This will cue different sounds (in a similar manner to metalsteps) and cause blood to appear instead of bullet impact flashes.

4.8.6 *fog*

Fog defines the brush as being a "fog" brush. This is a Q3MAP 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.

4.8.7 *lava*

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

4.8.8 metalsteps

The player sounds as if he is walking on clanging metal steps or gratings. Other than specifying **flesh**, **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.

4.8.9 nodamage

The player takes no damage if he falls onto a texture with this surfaceparm

4.8.10 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.

4.8.11 nodraw

A texture marked with nodraw will not visually appear in the game world. Most often used for triggers, clip brushes, origin brushes, and so on.

4.8.12 nodrop

When a player dies inside a volume (brush) marked nodrop, no weapon is dropped. The intended 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.

4.8.13 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.

4.8.14 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 Note: 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.

4.8.15 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.

4.8.16 nosteps

The player makes no sound when walking on this texture.

4.8.17 nonsolid

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

4.8.18 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.

4.8.19 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.

4.8.20 slick

This surfaceparm included in a texture should give it significantly reduced friction.

4.8.21 slime

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

4.8.22 structural

This surface attribute causes a brush to be seen by the Q3MAP 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.

4.8.23 trans

Tells q3map that pre-computed visibility should not be blocked by this surface. Generally, any shaders that have blendfuncs should be marked as surfaceparm trans.

4.8.24 water

Assigns to the texture the game properties for water.

5 Editor specific shader instructions

These instructions only affect the texture when it is seen in the Q3Radiant editor. They should be grouped with the surface parameters but ahead of them in sequence.

5.1 *qer_editorimage* < texture path/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 brightnesses) without having to create a new piece of TGA art for each new shader.

```
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
    {
        map textures/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
    }
}
```

5.2 *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.

5.3 *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.

6 Stage Specific 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. Q3MAP 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.

6.1 Texture map specification

6.1.1 `map <texturepath/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.

\$lightmap

This is the overall lightmap for the game world. It is calculated during the Q3MAP 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.

6.1.2 `Clampmap <texturepath>`

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` be 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.

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 (`tcScale`) 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:

Example: Using clampTexCoords 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
}
```

6.1.3 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 texturepath/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 animMap 10 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 seconds before the next one is displayed. The cycle repeats after the last frame in sequence is 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.

```
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
    }
}
```

6.2 Blend Functions

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

6.2.1 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.

6.2.1.1 *blendfunc add*

This is a shorthand command for **blendfunc gl_one gl_one**. Effects like fire and energy are additive.

6.2.1.2 *blendfunc filter*

This is a 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.

6.2.1.3 *blendfunc blend*

Shorthand 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.

6.2.2 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:

$$[\text{Source} * \langle \text{srcBlend} \rangle] + [\text{Destination} * \langle \text{dstBlend} \rangle]$$

Source is usually the RGB color data in a texture TGA file (remember it’s all numbers) modified by any rgbgen and alphasen. 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 may be 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 bit map 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.

6.2.3 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.; $A = 1.0 - \text{SRC.A}$)

6.2.4 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.

6.2.5 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.

6.2.6 Technical Information/Limitations Regarding Blend Modes:

The Riva 128 graphics card supports ONLY the following blend modes:

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.

6.3 *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 <func> parameters are **wave**, **identity**, **identityLighting**, **entity**, **oneMinusEntity**, **fromVertex**, and **lightingDiffuse**.

6.3.1 *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.

6.3.2 *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.

6.3.3 *rgbGen wave* <func> <base> <amp> <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.

<func> **Wave** forms and their effects:

Sin: color flows smoothly through changes.

Triangle: color changes at a constant rate and spends no appreciable time at peaks and valleys.

Square: color alternates instantly between its peak and valley values.

Sawtooth: With a positive frequency value, the color changes at a constant rate to the peak then instantly drops to its valley value.

Inversesawtooth: An inverse sawtooth wave will reverse this, making the ascent immediate (like a square wave) and the decay fall off like a triangle wave.

<base> Baseline value. The initial RGB formula of a color (normalized).

<amp> Amplitude. This is the degree of change from the baseline value. In some cases you will want values outside the 0.0 to 1.0 range, but it will induce clamping (holding at the maximum or minimum value for a time period) instead of continuous change.

<phase> See the explanation for phase under the waveforms heading of Key Concepts.

<freq> Frequency. This is a value (NOT normalized) that indicates peaks per second.

6.3.4 *RgbGen entity*

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

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

6.3.5 *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.

6.3.6 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 MD3s put into the Quake III Arena world.

6.3.7 rgbGen oneMinusVertex

As rgbGen vertex, but inverted.

Design Note: This keyword would probably not be used by a level designer

6.3.8 rgbGen lightingDiffuse

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

Design Note: - 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.

6.4 AlphaGen <func>

The alpha channel can be specified like the rgb channels. If not specified, it defaults to 1.0.

6.4.1 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.

6.5 tcGen <coordinate source>

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

<base> = base texture coordinates from the original art.

<lightmap> = lightmap texture coordinates

<environment> = Make this object environment mapped.

6.5.1 tcGen vector (<sx> <sy> <sz>) (<tx> <ty> <tz>)

New texcoord 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.

6.6 **tcMod <func> <...>**

Specifies how texture coordinates are modified after they are generated. The valid functions for tcMod are **rotate**, **scale**, **scroll**, **stretch** and **transform**. **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 other words, if you see:

tcMod scale 0.5 0.5

tcMod scroll 1 1

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

6.6.1 **tcMod rotate <degrees per per second>**

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).

6.6.2 **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)

6.6.3 **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 may be popping or snapping visual effects in some shaders.

6.6.4 **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

<base>: A base value of one is the original dimension of the texture when it reaches the stretch stage. Inserting other values positive or negative in this variable will produce unknown effects.

<amplitude>: This is the measurement of distance the texture will stretch from the base size. It is measured, like scroll, in textures. A value of 1 here will double the size of the texture at its peak.

<phase>: See the explanation for phase under the deform vertexes keyword.

<frequency>: this is wave peaks per second.

Wave Functions <func>

Sin wave: the texture expands smoothly to its peak dimension and then shrinks smoothly to its valley dimension in a flowing manner.

Triangle wave: The textures stretch at a constant rate and spend no appreciable time at the peak or valley points.

Square wave: The texture is shown at its peak for the duration of the frequency and then at its valley for the duration of the frequency.

Sawtooth: the texture stretches like a triangle wave until it reaches a peak, then instantly drops to the valley, as in a square wave.

Inversesawtooth: this is the reverse of the sawtooth wave.

6.6.5 tcMod <transform> <m00> <m01> <m10> <m11> <t0> <t1>

Transforms each texture coordinate as follows:

$$S' = s * m00 + t * m10 + t0$$
$$T' = t * m01 + s * m11 + t1$$

This is for use by programmers.

6.6.6 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.

The parameters for this shader are defined as follows:

<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 **deform vertexes** 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.

6.7 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 lightmapped grates that are alpha tested (it is also used for mirrors).

6.8 *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.

6.9 *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:

- A.) the lightmap must be rendered first. This is because the subsequent detail texture will be modifying the lightmap in the framebuffer directly;
- B.) the detail texture must be rendered next since it modifies the lightmap in the framebuffer
- C.) the base texture must be rendered last ;
- D.) 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
- E.) the detail shader stage MUST have the "**detail**" keyword or it will not be disabled if the user uses the "`r_detailtextures 0`" setting ;
- F.) 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.;
- G.) the detail stage should scale its textures by some amount (usually between 3 and 12) using "`tcMod`" 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.

H.) detail textures add one pass of overdraw, so there is a definite performance hit .

I.) 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:

Example: Texture with Detail

```
textures/bwhtest/foo
{
    // draw the lightmap first
    {
        map $lightmap
        rgbGen identity
    }
    // modify the lightmap in the framebuffer by
    // a highly compressed detail texture
    {
        map textures/details/detail01.tga
        blendFunc GL_DST_COLOR GL_SRC_COLOR
        // YOU MUST USE THIS!!

        detail
        // for the detail to be disabled, this must be present

        tcMod scale 9.1 9.2
    }
    // now slap on the base texture
    {
        map textures/castle/blocks11b.tga
        blendFunc filter
    }
}
```

6.10 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 Notes on 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 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.

Example: An opaque texture with see-through holes knocked in it.

```
textures/base_floor/pjgrate1
{
    surfaceparm metalsteps
    cull none

    // A GRATE OR GRILL THAT CAN BE SEEN FROM BOTH SIDES
    {
        map textures/base_floor/pjgrate1.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:

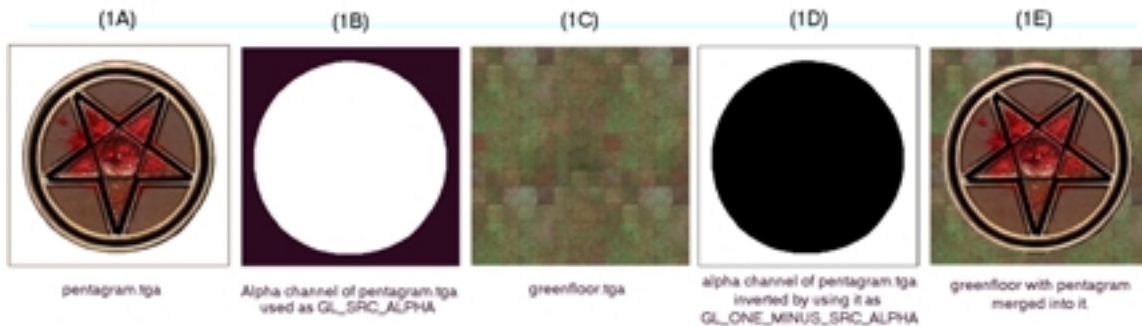


Figure 1

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 (not used) 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_MINUS_ONE_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.

8 Troubleshooting Shaders

If a shader is not working, look first for syntax errors.

Are the brackets correctly set?

Do you have too many parameter values on a line?

Are you using a word in a parameter that wants a numerical value?

Are you using a numerical value in a parameter that wants a word?

Are the path names to your textures correct?

Are your texture names correct? There is a chance that the texture name is too long or too complex. Try renaming a texture with a shorter, simpler name.

9 Creating New Textures

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 game load in assets.

9.1 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. Paint Shop Pro from JASC (v5.0+) can also make an alpha channel by creating a mask and naming it "alpha".

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.

9.2 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.

9.3 Rules and Guidelines

9.3.1 Rules

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.

9.3.2 Guidelines

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.

9.4 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 an archiving program call Winzip to make the pk3 file. Get Winzip from <http://www.winzip.com/winzip/winzip.htm>

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.