Darkplaces material system

Darkplaces material system is created to put more direct control over the surface qualities of textures into the hands of designers and artists. This reference contains all information needed to work with Darkplaces materials.
Since Darkplaces material system is inspired by the Quake 3 Shader System, but adapted and extended to Darkplaces way-of-things-done.

Introduction

Contents

What is a Material?

Materials 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 have the same name as the texture set which contains the textures being modified (e.g; textures_inn, models_mapobjects_crypt, etc,). Several specific script documents have also been created to handle special cases, like liquids, environments and special effects.

For Darkplaces, material scripts are located in 'Path_To_The_Game/gamedir/scripts'.

A Darkplaces material 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":

```
// material definition
textures/crypt/stone {  // keywords
        qer_editorImage radiant/textures/crypt/stone
        surfaceparm stone
        dpglossexponentmod 0.5
        // base stage
        {
                map textures/crypt/stone // engine will search fo textures/mymaterial.tga, textures/mymaterial.jpeg
        }

  // lighting stage
        {
                map $lightmap
        }
}
```

Material names & Conventions

The first line is the material name. Material 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 (/Blood Omnicide/kain in our case), but they do not need to be.

Materials that are only going to be referenced by the game code, not modeling tools, often are just a single world, like "collision" or "dpbihclip".

Materials 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.

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

Material 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.

Material scripts Formatting

In order to get materials to be easily readable and editable it is recommended to format all material scripts like this template:

```
<materialname>
{
	<Editor Specific Keywords>
	<Q3Map2 Specific Keywords>
	<General Keywords>
	{
		<Stage Keywords>
	}
}
```

Inspired by Quake 3 but different

Quake 3 shader scripts was an inspiration for darkplaces material system and visual look of material scripts is very very close to quake 3. So if you are familiar with Id Tech 3 shaders system, you will be right home with material script syntax. Therefore Darkplaces material system was written independently and there is some serious differences:

Realtime lighting

Any shader keywords that are not compatible with realtime lighting were removed. Other ones have sligtly changed behavior.

Aiming to realtime lighting changes the way stages work. There are no render passes (so multilayered material can only be created by duplicating real surfaces). Darkplaces material render pipeline can have 1 to 3 fixed usage stages:

- No draw (0 stages)
- Unlit texture (1 stage)
- Lit texture (2 stages)
- Lit texture with terrain blending (3 stages)

All other Id Tech 3 stage configurations such as skies are not supported.

To simulate multilayer materials, should make several copies of surface with different material attached (for map surfaces can use Q3map2's cloneShader keyword)

New keywords

Darkplaces introduces many of its own keywords that do not exist in Id Tech 3.

Q3 Texture = Darkplaces skinframe

Id tech 3 only needs color info for texture. Darkplaces engine using normalmap, specularity map, glow texture and other textures that store additional info. All that texture files together (color + normalmap + glossmap + glow etc.) forms a skinframe.

Different speed costs

Darkplaces's engine rendering pipeline is very different from Id Tech 3. By using pixel and vertex shaders, it introduces new speed cost parameters that was not exist in Idtech 3:

- shader instructions - amount of vertex and pixel shader instructions generated for particular material (depends of effects used)
- texture lookups - amount of texture lookups in pixel shader, additional textures in a skinframe, parallax mapping increase amount of texture lookups
- use of VBO - non-animated models render faster with VBO, some shader keywords (such as tcMod turbulent, deformVertexes) disables VBO
- use of Lighting - realtime light is slower on blended (add more draws of surface, one per light) or offset mapped surfaces (offset mapping is processed again for each light)

Material is 'Shader'

Since Darkplaces uses tools with Id Tech 3 roots, materials are often spelled as 'shaders' there. To not be confused, this terms are identical in q3 tools:

- Shader = Material
- Shader name = Surface shader = Material name = Surface material
- Shader script = Material script
- Shader keywords = Material keywords
- etc.

Material Types

The keywords that affect materials are divided into three classes. The first class of keywords are global parameters. Some global parameters ("surfaceparms" And all "q3map_" keywords) are processed by Q3MAP2 and change physical attributes of the surface that uses the material. These attributes can affect the player. To see changes in these parameters one must re-compile the map.

Second class are keywords parsed by the engine server part (surfaceparms, some other keywords). They change physical properties of the surface and can affect gameplay.

The remaining global keywords, and all Stage Specific Keywords are processed by the renderer. They are appearance changes only and have no effect on gameplay 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).

Material keywords are not case sensitive.

Important: Some of the material commands may be order dependent, so it's good practice to place all global material commands (keywords defined in this section) at the very beginning of the material and to place material stages at the end.

Key Concepts

Contents

Ideally, a designer or artist who is manipulating textures with material 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 materials work for you.

Materials not only modify the visible aspect of textures on a geometry brush, curve 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 materials affect the way the brush operates in the game world. Examples include water, nonsolid, and detail. Deformation effects change the actual shape of the affected brush or curve, and may make it appear to move.

Power has a Price

The material 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. Many OpenGL 2.0 effects attached with material keywords (such as water shader, refraction cubemap, offset mapping) makes renderers do more calculations,

which take more CPU/GPU time and make games slower. Water shader surface will draw a world 2 times to get reflective and refractive image, refraction cubemap will add one more texture sampler, parallax mapping will make more texture lookups (from 3 for traditional offset mapping to 14-15 for relief mapping). Blended surfaces will be drawn twice if lit by one or more realtime lights (this means their triangle count will have double effect on r_speeds).

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 Darkplaces engine 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 numbblack is 0 0 0. The value for ers. The value for complete white is 255 255 255. However, the Darkplaces graphics engine requires that the color range be "normalized" into a range between 0.0 and 1.0.
Tip: often you can see the RGBA abbreviation which stands for 'RGB + alpha', this is RGB with one more channel - transparency channel. 32-bit images are RGBA, while 24-bit is RGB.

Normalization: a scale of 0 to 1

The mathematics in Darkplaces engine 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 Darkplaces formula for that color component. The same holds true for texture coordinates.

Texture Sizes

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.

Measurements

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

Game unit

A game unit is used by deformations to specify sizes relative to the world. In Blood Omnicide 16 units equals one foot, 48 units equals to one meter. The default texture scale used by the NetRadiant map editor results in two texels for each game unit, but that can be freely changed and perturbed with material scripts.

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 coordinates

This is the normalized (see above) dimensions of the original texture image. 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 values 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

Some of the material rendering 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 material scripts:

- **sin**: 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 choppy looking waveforms.
- **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 waves, 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 baseline. A phase of 0.75 would be at the lowest point of the valley. If only one wave form is being used in material, 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 material, but with the phases shifted can be used to create interesting visual effects. 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 material 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.

General Keywords

surfaceparm

This is, by the fact, the most widely used and most tricky material keyword.

All surfaceparm keywords are preceded by the word *surfaceparm* as follows: *surfaceparm nonsolid* or *surfaceparm trans*. Each surfaceparm has a corresponding q3map_*surfaceparmname* keyword (**q3map_nonsolid**, **q3map_trans** etc.) which is only acquired by Q3map2. Surfaceparm keywords change the physical nature of the materials and the brushes. Changing any of these values will likely require the map to be re-compiled. These are global and affect the entire shader.

This document skips all Q3map2 surfaceparms that is not used by Blood Omnicide.

General surfaceparms

surfaceparm alphashadow

This keyword applied to a side 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.
Used by Q3map2 light phase.

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. It is possible to increase the resolution of the lightmap receiving the shadows with a slight 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 **_ls** key with a floating-point value for the scale of the lightmap.

surfaceparm detail

This surface attribute causes a brush to be ignored by the Q3Map2 process for generating possible break-points in the BSP tree. Its functionality is the same as marking brush as detail in level editor.
Used by Q3map2 BSP phase

surfaceparm lava

Assign to the material the game properties set for lava. This affects the contents of the brush.
Used by engine (BIH collision) and q3map2 (bsp phase).

This keyword is used on common/ilava material.

surfaceparm lightfilter

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.
Used by Q3map2 light phase.

surfaceparm nodraw

Prevents Q3map2 from generating drawsurfaces for this material.
Used by Q3map2 BSP phase.

surfaceparm nomarks

Prevent engine from spawning decals for this surface.
Used by engine renderer.

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 surface will have no lightmap generated. This will save some memory and make map compilation faster.
Used by Q3map2 light phase.

This keyword has no effect at vertex-based lighting (lighting calculated for vertices and being saved to their rgba attributes). Surface with no lightmap will be rendered with vertex-based lighting. To disable vertex lighting calculation, use the **q3map_noVertexLight** keyword.

surfaceparm nonsolid

Surface does not block the movement of entities in the game world. This affects the content of a brush. This still makes q3map2 generate collision surfaces.
Used by engine (BIH collision) and q3map2 (bsp phase).

surfaceparm slime

Assign to the material the game properties set for slime. In Blood Omnicide, slime is a swamp.
Used by engine (BIH collision) and q3map2 (bsp phase).

This keyword is used on common/islime material.

surfaceparm trans

Tell Q3Map2 that pre-computed visibility should not be blocked by this surface.

Used by q3map2 BSP phase.

Any materials that have blendfunc should be marked as surfaceparm trans (anything behind them will not get culled).

surfaceparm water

Assigns to the material the game properties set for water. This affects both the surface and the content of a brush.
Used by engine (BIH collision) and q3map2 (bsp phase).

This keyword is used on **common/iwater** material.

System surfaceparms

This surfaceparms is only used in common materials. You don't need to specify them on your new materials.

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.
Used by Q3map2 BSP phase. This keyword is used on **common/hint** material.

surfaceparm monsterclip

Surface with this flag will block NPC movement. Should be used with **surfaceparm nonsolid**.
Used by engine (collision). This keyword is used on **common/monsterclip** material. You dont need to specify it on your new materials.

surfaceparm lightgrid

The min/max bounds of brushes with this material in a map will define the bounds of the map's Lightgrid.
Used by Q3map2 light phase. This keyword is used on **common/lightgrid** material. You dont need to specify it on your new materials.

surfaceparm playerclip

Surface with this flag will block player movement. Should be used with **surfaceparm nonsolid**.
Used by engine (collision). This keyword is used on **common/playerclip** material.

surfaceparm origin

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.
Used by q3map2 BSP phase. This keyword is used on **common/origin** material

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 common/hint material. Generally speaking, any opaque material not marked as "detail" is, by default, structural, so you shouldn't need to specify this.
Used by q3map2 BSP phase.

Custom surfaceparms

These surface parameters are defined by custinfoparms.txt and used to assign a game-related properties to material, such as hit effect and step sound.

Blood Omnicide have this game-physics materials:

- grass
- ice
- dirt
- wood
- oldwood
- ground
- sand
- dullmetal
- brookwater
- glass
- fur
- marble
- hay
- slate
- flesh
- metal
- oldmetal
- stone
- oldstone
- plan

cull none

Every surface of a polygon has two sides, a front and a back. Typically, we only see the front side. For example, a solid block you only show the front side. In some situations we see both (grates, screens etc.).

To "cull" means to remove. This keyword allows to disable side culling, so both sides of the surfaces would be visible.
Tip: Realtime lighting and lightmaps don't work well with two sided surfaces, causing the back side to be lit the same as the front. To avoid that, you should create two one-sided surfaces in game.

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

noPicMip

This causes the texture to ignore **r_picmip** cvar. The image will always be high resolution. Apply to small low-resolution textures that will be over blurred with r_picmip. Not used in Blood Omnicide.

Bug: Darkplaces can crash if the texture has a DDS file with no mipmap layers generated and allows for mipmaps in material. Use this keyword to suppress that.

noMipMaps

Implies noPicMip. Also prevents the generation of any lower resolution mipmaps for use by the 3D card. This saves 25% of memory used to store this texture (both RAM and VRAM).

deformVertexes

This function performs a general deformation on the surface's vertices, changing the actual shape of the surface before drawing the material. You can stack multiple deformVertexes commands to modify positions in more complex ways, making an object move in two dimensions, for instance.

deformVertexes wave <func> <base> <amplitude> <phase> <freq>

Designed for water surfaces, modifying the values differently at each point. It accepts the standard waveform functions.
The "div" parameter is used to control the wave "spread" - a value equal to the tesselation distance (see q3map_tessSize, a subdivision size, in game units, used for the material when seen in the game world).

Typical usage: deformVertexes wave 128 sin 0 0.3 10 0.5

deformVertexes normal <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 material options like lighting (especially specular) and environment mapping (see stage specific keywords for tcGen environment). If the materialdon't use normals in any of their calculations, there will be no visible effect.
IMPORTANT NOTE: In Darkplaces, offset mapping uses vertex normals for the calculations. deformVertexes normal most likely will mess it up.
Design Notes: Putting values of 0.1 t o 0.5 in Amplitude and 1.0 to 4.0 in the Frequency can produce some satisfying results.

Typical usage:

deformVertexes normal 2 0.5

Bug: On a lightmapped surface, deformVertexes normal has no effect if tangentspace deluxemapping is used (since tangentspace deluxemapping do not use vertex normals). Use modelspace deluxemapping.

deformVertexes bulge <bulgeWidth> <bulgeHeight> <bulgeSpeed>

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

Typical usage:

deformVertexes bulge 0.1 0.6 0.18

deformVertexes roundwave <radius> <wavesize> <radiusexponent> <x> <y> <z> {wavefunc}

Makes round waves coming from the center point. Designed to simulate fountains and pools. "Radius" is max radius of roundwave. "Wavesize" is length of wave at the start. "Exponent" is a power function, it makes wave longer with the distance from center point. X,y,z are effect offsets from entity center (or from world origin for worldspawn surfaces).
"Wavefunc" is standart wavefunction used to create waves - 5 parms (<wavefunc> <base> <amplitude> <phase> <freq>).
example: deformVertexes roundwave 180 18 3 0 0 0 sin 0 1.00 0.0 1.2

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

This keyword is used to make a brush, curve patch or 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 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.

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

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

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.

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 choppy looking waveforms. 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 inverse sawtooth 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 Waveforms 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.

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

Engine renderer features

These keywords control various renderer features.

dpOffsetMapping <type> <scale> [<biastype> <biasvalue>]

todo

dpSelfShadowing <scale> <offsetscale> [<offsetbiastype> <offsetbiasvalue>]

todo

dpNoFog

Disables fog rendering for a surface.

dpPolygonOffset <factor> <offset>

Allows arbitrary offsets of a surface's depth during rendering. Material with this keyword is able to stack over the rest of geometry with no z-fighting. Useful for decals.

Both parameters are optional, if not used, a default factor of 0 and offset -2 is used.
Tip: mod_q3shader_default_polygonfactor and mod_q3shader_default_polygonoffset control default values.

dpTransparentSort <sort>

Rendering of blended materials requires software sorting to keep the depth order of geometry. This keyword changes default distance sorting to allow for special effects.

- **distance** : default distance-based sorting
- **hud** : this surface will always be ontop of all another blended materials
- **sky** : opposite to hud, this surface will always ve vehing all other blender materials.

Per-Pixel Lightning

These keywords control various parameters used by per-pixel lighting of a surface.

dpShadow

Forces this material to be shadow caster even if it's blended or not drawn (blended surfaces by default are not casting shadows).
Tip: This keyword is useful in creating **shadow meshes, a meshes** that are invisible (and having lower polycont) but used for shadow casting.

dpNoShadow

Discards material from shadow casting.

dpNoRtlight

Disables any realtime lighting on a surface.

dpGlossIntensityMod <value>

Multiplies gloss intensity of this material. Default value is 1; 0.5 will halve gloss effect intensity, 2.0 will double it etc. This are used ontop on base gloss intensity set by a texture's _gloss color.

dpGlossExponentMod <value>

Scales gloss exponent for a surface. Gloss exponent controls how sharp the gloss effect is. Low values will make the surface matte, higher values will make it look like it is covered by a plastic film. Just like intensity mod, this is a multiplier to a value in texture's _gloss alpha (if there is no alpha - base gloss exponent is forced to be 1).

Typical values are 0.25 (wood), 2.0 (glass).

dpRtlightAmbient <value>

Smooths out shading of a surface by adding an amount of ambient light. This is useful on very grude and lowpoly surfaces (such as grass) because strong shading will show artefacts caused by low detalisation of a mesh.

Reflection and refraction

This material keywords allows to add refraction and reflection effects to a surface. All reflective and refractive effects are part of the water shader, so they are only visible when **r_water** is 1.

dpReflect <distortion> <r> <g> <b> <a>

Make this material reflective. The reflection is alpha blended on the texture with the given alpha, and modulated by the given color. Distort is used in conjunction with the normalmap to simulate a nonplanar water surface.
Important: This is a 'hard' shader that requires much computation time. Abuse of this will slow down rendering.

dpRefract <distortion> <r> <g> <b>

Works pretty much the same as reflection and can be combined with it (though it's not good). The refraction replaces the transparency of the texture. Distort is used in conjunction with the normalmap to simulate a non planar water surface.

dpWater <reflectmin> <reflectmax> <refractdistort> <reflectdistort> <refractR> <refractG> <refractB> <reflectR> <reflectG> <reflectB> <alpha>

Add a water shader to a material. This combines the effects of dpreflect and dprefract to simulate a water surface. However, the refraction and the reflection are mixed using a Fresnel equation that makes the amount of reflection slide from reflectmin when looking parallel to the water to reflectmax when looking directly into the water. The result of this reflection/refraction mix is then layered BELOW the texture of the shader, so basically, it "fills up" the alpha values of the water. The alpha value is a multiplicator for the alpha value on the texture (set this to a small value like 0.1) to emphasize the reflection and make the water transparent; but if r_water is 0, alpha isn't used, so the water can be very visible then too.

Typical usage:

dpwater 0.4 1.0 0.3 1.0 0.28 0.21 0.18 0.80 0.76 0.65 0.0

dpWaterScroll <speedScale> <globalScale>

Animate surface normalmap in water shader by duplicating it, scrolling, offsetting and multiplying the layers.
Tip: Globalscale affects speed.

dpCamera

Flags this surface as a camera surface. Used to make portal effects like ones seen in Pray. Not used in Blood Omnicide.

dpReflectCube <cubemap>

Sets static cubemap texture for reflections.
Important: A _reflect skinframe texture sets local intensity of reflections and should be presented in order to work.
Tip: Cubemapped reflections a lot faster than realtime ones.

Typical usage:

dpreflectcube textures/environment/reflect1

Shader Kill

These keywords are used to hide this material (as if it has no stages) under certain circumstances. Only affects visual appearance of material, not collisions or contents.

dpShaderKillIfCvar <cvar> <operator> <value>

Kills this material if cvar matches a value.

```
// kill this shader if r_foo is 42
dpshaderkillifcvar r_foo == 42
```

Operators are:

- **==** - equal
- **!=** - not equal
- **>** - greater
- **>=** - greater or equal
- **<** - lesser
- **<=** - lesser or equal

dpNoShaderKillIfCvar <cvar> <operator> <value>

Prevents material from being killed. Have a priority over dpshaderkillifcvar.

dpShaderKillIfCvarZero <cvar>

Kills material is supplied cvar's float value is 0.

dpNoShaderKillIfCvarZero <cvar>

Opposite to above.

Physics

These keywords control parameters for game physics.

dpMeshCollisions

Use surface's draw geometry for collision detection instead of generated BSP geometry (some models have no one). This works only if BIH is enabled.

dpNoBih

Disable Bounding Interval Hierarchy tree construction for a surface. Used an optimisation on surfaces which never need game physics (such as grass).

Special Keywords

Q3Map2 Specific Keywords

All of q3map2 specific keywords are processed during map compile. If these keywords change, recompiling the map is required for the changes to take effect.

General use keywords

q3map_globalTexture
This keyword disables texture coordinates optimization for texture projection on brushes (optimization tries to keep texture coordinates closer to 0-1 range), making them to be related to the world center, not brush. This is useful when applying **tcMod scale** to several adjastent brushes, which can be wrong if not using this parameter.

q3map_tessSize <amount>
Controls the tessellation size (how finely a surface is chopped up into 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.
Important: Abuse of this can create a huge number of triangles.

q3map_invert
Inverts a surface normal. Works on brush faces, models and patches.

q3map_offset <N.N>
Offsets a surface along the vertex normals N.N game units.

q3map_cloneShader <materialName>
A shader with this keyword will inherit the target shader's properties and appearance.

Important: Be careful, this can lead to an infinite loop if a cloning shader references another cloning shader or itself.

q3map_backShader <materialName>
This allows a brush to use a different shader on the back side of the surface. By way of example, this would allow a water brush (or other) surfaces to have a different appearance when seen from the inside.

Back shader is a better alternative to **cull none** because it is consistent with lighting. Drawbacks are that it produces more geometry data (larger loading times, more memory required), which

may be abused if the surface is used too many times.
Tip: Back shader is identical to q3map_cloneShader (with cloned shader having q3map_invert)

q3map_tcGen ivector (<Sx> <Sy> <Sz>) (<Tx> <Ty> <Tz>)
Generate new texture coordinates for a surface. Projects a texture S units by T units along a chosen axis. q3map_tcGen vector (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.

q3map_tcMod <function>
This works in a similar manner to the stage-specific tcMod keyword, 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).

- **rotate <degrees>** - Rotates the texture (around origin, not center) a specified number of degrees.
- **scale <S> <T>** - Scales S (x) and T (y) texture coordinates. Scale 2 2 would halve the size of the texture (doubling the texture co-ordinates).
- **translate <S> <T>** - Shifts texture coordinates by S, T amount. Translate 0.5 0 would shift it one-half in S, and none in T.

q3map_noClip
Normally, Q3Map2 clips all faces to the BSP, and then takes the minimum polygon that encompasses all visible fragments. This keyword forces Q3Map2 to use the original brush faces.

q3map_noTJunc
Read as "no T-Junc". With this option, surfaces with this material are not used for T-junction fixing.
Tip: q3map_noClip and q3map_noTJunc, used in combination, will preserve mesh geometry exactly as you make it.

Per-vertex RGBA tweaks

These keywords are used to tweak generation of per-vertex RGBA components which is normally used for vertex lighting and terrain blending/vertex controlled transparency.

q3map_colorMod <function>
Used to generate or modify RGB components on all surface vertices. Functions are:

- **set (<r> <g> <b>)** - replace vertex RGB with supplied values.
- **scale (<r> <g> <b>)** - multiplies RGB with supplied values.
- **dotproduct (<X> <Y> <Z>)** - multiplies color by dotproduct of vertex normal and given vector.
- **dotproduct2 (<X> <Y> <Z>)** - works in a similar way to dotproduct except it exaggerates the differences in vertex normals by squaring the final dot product value.
- **volume** - a special keyword that lets a brush affect all vertices inside it (see colormod brushes).

q3map_alphaMod <function>
Works in a similar way as q3map_colorMod but affects alpha components. Functions are:

- **set <value>)** - replace vertex alpha.
- **scale <value>)** - multiply vertex alpha.
- **dotproduct (<X> <Y> <Z>)** - same thing as for q3map_colorMod.
- **random <A.A> <B.B>** - sets the random alpha to vertices in A-B range (this will keep junctions - vertices at the same origin will receive the same random value).
- **randomjitter <A.A> <B.B>** - works like random but value gets added, not replaced.
- **randomscale <A.A> <B.B>** - works like random but value is a multiplier.
- **volume** - a special keyword that lets a brush affect all vertices inside it (see colormod brushes).
- **wateralpha <N.N>** - this sets alpha only on surfaces with surfaceparm water, lava or slime. Should be used in conjunction with **q3map_alphaMod volume**.
- **waterspring <N.N>** - obsolete, dont use.
- **watermove <A.A> <F.F>** - obsolete, dont use.
- **waterwarp <N.N>** - obsolete, dont use.

q3map_alphaLayers <backgroundIndex> <foregroundIndex>
This keyword is used to assist terrain blending. Q3map2 tries to remove a seams on the connected terrain blend materials; this is done by forcing both material (eg. forcing per-vertex alpha component) to show the same texture at connection point. In order to do that, q3map2 needs to know what texture is used in the background and foreground layer of a material. Supplied indexes give that info. All textures that are used for terrain blending should be indexed (e.g. terrain/grass is 1, terrain/ground is 2, terrain/rock is 3). For grass-ground blend - q3map_alphaLayer 1 2, for ground-rock blend - q3map_alphaLayers 2 3 etc.

q3map_noVertexLight

Material with this keyword will skip vertex light calculations entirely. This keyword should be in material if vertex RGBA data is used for some effect.

q3map_vertexPointSample

Vertex lighting of a surface will be calculated using lightgrid's *LightContributionToPoint* function (instead of default *LightContributionToSample* which works for lightmaps).

Area lights

q3map_sunExt <r> <g> <b> <intensity> <degrees> <elevation> <deviance> <samples>

This keyword in a sky shader will create the illusion of light cast into a map by a single, infinitely distant parallel light source (sun, moon, hellish fire, etc.). This is only processed during the lighting phase. Parameters are:

- **r g b** - Color of light. 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.
- **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.
- **deviance** - The number of degrees for penumbra (half-shadow). General values up to 2 or 3 are acceptable. The real sun has a solid angle of about half a degree. This gives you much more realistic shadows from the sun.
- **samples** - The number of random jitters distributed over the solid arc (16 is a good value).

Sky shaders should probably still have a q3map_skyLight value. The sun gives a strong directional light, but doesn't necessarily give the fill light needed to soften and illuminate shadows.

q3map_skyLight <amount> <iterations>

Make a surface to cast light. Amount is a brightness value, similar to what you would use in **q3map_sunExt**. Good values are between 50 and 200. Iteration 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_lightImage <texturePath>

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_skyLight 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.

q3map_lightRGB <r> <g> <b>

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.

Tip: This does not affect bounced light in radiosity or lightfilter

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 splotches on lightmapped terrain.

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 light sources. The self parameter can be set for skyLights 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. 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.

Tip: q3map_lightmapFilterRadius should be placed before any light related shader directives that you want it to affect

Lighting tweaks

q3map_nonplanar
Instructs Q3Map2 to merge any adjacent triangles that don't lie in the same plane into a non-planar triangle soup. This allows shadows to be cast correctly across non-planar edges. It is typically used on lightmapped terrain shaders. This improves lightmapping of curved brush faces covered by this material.

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_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 material basis. This is required for lightmapped models.**Tip:** forceMeta is also applied by **Lightmapped** flag on misc_models

q3map_shadeAngle <angle>
Specifies the breaking angle for smooth 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.
Tip: This keyword only affects light calculations, not real surface normals

q3map_lightmapSampleSize <N>
Surfaces using a material with this keyword 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 lightmap sample size is 16 (can be overriden using -samplesize key of q3map2 light phase).

q3map_bounceScale <value>
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. Default bounce scale is 0.25.

q3map_normalImage <texturePath>
Specifies normalmap image to be used for calculating lightmap intensities. This will, effectively, bake normal map-caused shading into lightmap.

Editor Specific Keywords

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

qer_editorImage <textureName>

This keyword makes the level editor to display different textures in the 3D viewport.
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.
Later, Q3map2 will use this texture to generate texture coordinates on brushes.
Conventions: All textures supplied for qer_editorImage should be stored in radiant/ folder (like **radiant/textures/town/wall1_1**, **radiant/models/mapobjects/crypt/statue1** etc.)
Design Notes: It is useful to always specify qer_editorImage on materials that could be used on brushes. This will separate material's real texture from the editor texture and give the ability to replace game real texture with upscaled/downscaled one without re-editing a map.

qer_trans <N.N>

This keyword defines the percentage of transparency that a surface 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 surface is 20% opaque and nearly invisible.

qer_alphaFunc <func> <N.N>

If material has alphaFunc, this keyword will make it to be rendered with alpha test in level editor too.
Functions are:

- **greater** - greater than
- **gequal** - greater or equal
- **lesser** - lesser than
- **lequal** - lesser or equal
- **equal** - equal

Stages

Texture Map

Specifies the base texture map (a 24 or 32-bit TGA or JPEG file) used for this material.

map <textureName>

The texture may or may not contain alpha channel information. Texture name can have no extension as the engine will automatically add it. For texture/blah, darkplaces will search for this textures:

- dds/texture/blah.dds
- texture/blah.tga
- texture/blah.png
- texture/blah.jpeg

clampMap <textureName>

Specifies the source texture map (see map). Clamping keeps texture from tiling (texture coordinates are clamped to 0-1).
Design notes: level editor doesnt show clampMap properly, moreover brushes use texture projection which is converted to real texture coordinates when map is compiled. To bypass that limitation use the "Fit" instrument in the surface/patch inspector as it always makes 0-1-compliant texture projection.

animMap <frequency> <texture1> ... <texture64>

The surfaces in the game can be animated by displaying a sequence of 1 to 64 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 it repeats within a second. Animations that should last for more than a second need to be expressed as decimal values.
- *texture1...texture64*: the texture path/texture name for each animation frame must be explicitly listed. Up to 64 frames (64 separate texture files) can be used to make an animated sequence. Each frame is displayed for an equal subdivision of the frequency value.

If you want to clamp texture coordinates such as in **clampMap**, put **animClampMap** instead of **animMap**.

Example:
// 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.
animMap 0.25 **models/fx/flame1 models/fx/flame2 models/fx/flame3 models/fx/flame4**

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.

Blend Function

Blend functions are the keyword commands that tell the engine's renderer how material will be mixed with objects that are behind it.

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

blendFunc blend

Traditional alpha blending (shorthand command for *blendFunc GL_SRC_ALPHA GL_ONE_MINUS_SRC_ALPHA*).

blendFunc add

Additive blending (shorthand command for *blendFunc GL_ONE GL_ONE*).

blendFunc addalpha

Additive blending with alpha (shorthand command for *blendFunc GL_ONE GL_ONE_MINUS_SRC_ALPHA*).

blendFunc filter

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

blendFunc <source> <destination>

Alternative, GL-friendly blending mode definition. Source is a pixel of texture. Destination is a pixel in the framebuffer. Possible codes for source and destination:

- GL_ONE
- GL_ZERO
- GL_SRC_COLOR
- GL_SRC_ALPHA
- GL_DST_COLOR
- GL_DST_ALPHA
- GL_ONE_MINUS_SRC_COLOR
- GL_ONE_MINUS_SRC_ALPHA
- GL_ONE_MINUS_DST_COLOR
- GL_ONE_MINUS_DST_ALPHA

alphaGen vertex

Enables vertex-controlled transparency. Requires per-vertex RGBA attributes which are supported on submodels.

ASE format used by misc_model (and converted to map geometry during compile) supports per-vertex transparency.

Other model formats such as MD3, DPM, IQM do not support per-vertex RGBA attributes, hence cannot have per-vertex transparency.

tcGen environment

Generate spherical environment-mapped texture coordinates. Used for fake gloss effect.

tcMod <function>

Specifies how texture coordinates are modified after they are generated. The valid functions for tcMod are *rotate*, *scale*, *scroll*, *stretch*, *transform*, *turb* and *page*. Up to 4 tcMod's are allowed. 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.

Example:

```
// texture coordinates will be scaled then scrolled.
tcMod scale 0.5 0.5
tcMod scroll 1 1
```

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. 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.
Example:
// repeat twice along texture 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 scale 0.5 2

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 file. The scroll speed is measured in "textures" per second. A "texture" is the dimension of the texture being modified and includes any previous material modifications to the original texture file). A negative s value would scroll the texture to the left. A negative t value would scroll the texture down.
Example:
// Moves the texture down and right (relative to the TGA files original coordinates) at the rate of a half texture each second of travel.
tcMod scroll 0.5 -0.5

IMPORTANT NOTE: This should be the last tcMod in a stage. Otherwise there may be a popping or snapping visual effect in some materials.

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 Key Concepts for waveform parameter definitions).
Example:
// stretch coordinates using sin wave with 1.5 seconds cycle time
tcMod stretch sin 1.2 0.8 0 1.5

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. S stands for side coordinate (width), T stands for top (height).

Example:

```
// Transform that does nothing
tcMod transform 1 0 0 1 0 1
// Shift texture up by 1/128 of it's height
tcMod transform 1 0 0 1 0 0.0078125
```

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* : 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.

IMPORTANT NOTE: This should be the first tcMod in a stage because it is only software tcMod (all other ones transform texture matrix).

tcMod page <width> <height> <delay>

Texture animation using texture coordinates shifting. Require all animations to be stored in single texture in the form of tiles.
The texture is shifted by 1/<width> every <delay> seconds, and by 1/<height> every <delay>*<width> seconds.
Example:

```
// To use that animation, make a texture with the frames aligned in a grid like this:
//   1  2  3  4
//   5  6  7  8
// Then align it in on the model/brush/curve so only one of the animation frames can be seen on the surface.
// Engine will then display the frames in order and the cycle will repeat every 0.8 seconds.
tcMod page 4 2 0.1
```

Lightmap Stage

This stage tells the renderer to show material with static lighting (lightmap, vertex lighting or lightgrid) applied. It has no parameters and must always be the last stage.

Example:

```
models/mapobjects/cave/crystal01
{
   dpglossintensitymod 0.5
   dpglossexponentmod 0.2
   dpmeshcollisions
   dpreflectcube textures/envmaps/crystal01_
   dpnoshadow
   {
      map models/mapobjects/cave/crystal01
   }
   // lightmap stage
   {
      map $lightmap
   }
}
```

Terrain Blending Stage

For terrain blending to be used on material, an additional stage that represents a second texture (foreground layer) for terrain blending should be added.

Example:

```
textures/terraintest
{
   qer_editorimage radiant/textures/terraintest

   surfaceparm stone
   surfaceparm detail
   surfaceparm trans
   // base stage - background texture
   {
      map textures/terrain/grass
   }
   // terrain blending stage - foreground texture
   {
      map textures/terrain/rock
      blendFunc blend
      alphaFunc GE128 // optional parm and enables additional alpha-splatting effect to be used in blending algorithm
      alphaGen vertex
   }
   // lightmap stage
   {
      map $lightmap
   }
}
```

Author: VorteX (https://omnicide.razorwind.ru/wiki/)