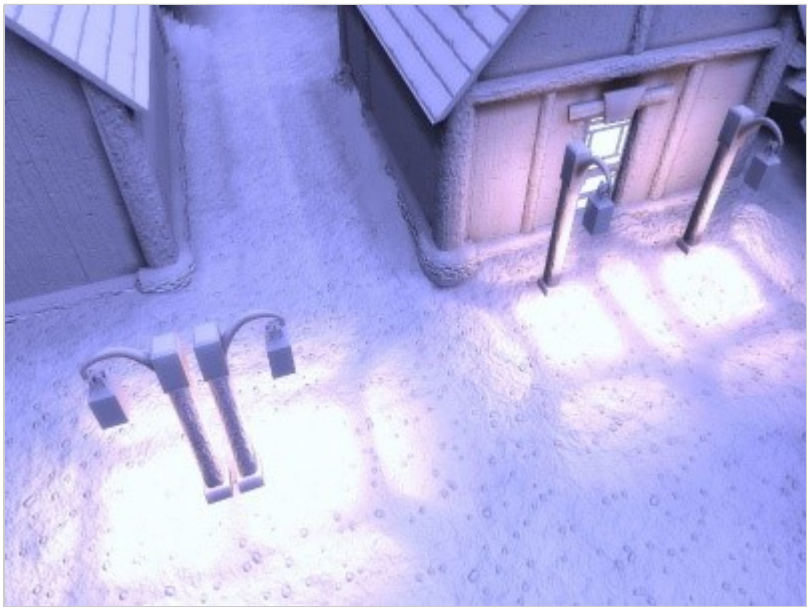
Darkplaces lighting

Lighting is one of the most important and influential elements in level design. It has the power to make or break the visuals, theme and atmosphere. Lighting is just as important as geometry. Without lighting there is no environment but just a group of 3-dimensional objects. Its purpose goes further than just giving the players the ability to see where they are going.

Darkplaces engine combines several lighting techiques in order to get best performance and best looks. Because of that there is lots of options and tricks. Clever usage of them will help designers to make fast-to-render and well themed maps.



Map with only lighting applied

This reference will guide you in the deep dungeon of Darkplaces lighting methods, effects, tricks and quirks.

Important Note

Textures are the base for lighting. Before starting to learn how lighting works, be sure to check out Darkplaces material system as many of it's concepts and features used by lighting too.

Chapters

Introduction

Darkplaces lighting is a quite complex system that have connections with many other engine systems (such as gamecode, clientcode, materials, maps etc.). Each effect (normalmapping, specular, fog, cubemap filter, style etc.) are optional and can be customized on per-object base. This gives an very huge amount of options and rendering tricks.

What is a light

In Darkplaces engine, light is entity that engine or compiler use to illuminate objects around. Light's entity keys are options. There is three type of light sources: **static**, **dynamic** and **fake**.

Static lightsources

Entity which classname begins with "light" is static lightsource. Usually its just classname "light", the

differences between classnames are used to attach some additional functionality (in Quake there was light_torch, light_globe etc., Blood Omnicide dont use this). For map compiler, all "light*" entities are same and using same property set.

Q3map2 generates static lighting data when map is compiled. Depending on map size, it could last from few seconds to a hours of compilation. Generated textures and attributes are very fast to render and dont have any hit from lights count and radius (can have thousands of lights on a map, its just affects the compile time, not gameplay).

Dynamic lightsources

Dynamic lightsources are entities that get illumination generated and rendered by engine. All dynamic lights are omnidirectional, spot lights can be simulated with cubemap filter effects. Dynamic lights can illuminate moving, rotating and animating objects. There cant be many dynamic lights because they are costly to render. Too many dynlights in a scene will cause FPS drop and game slowdown.

Fake lightsources

A special kind of lightsources used for render tricks. They are not real lights, but just make some effects that looks like a light. This is quite grude, but also very fast (since it ignores generic lighting math). Coronas and glares are "fake" lightsources.

Key Concepts

Power has a price

Darkplaces lighting system gives the designer, artist and programmer a great deal of easily accessible power over the appearance and potential special effects. This power comes with price tag attached - more light effects you use, more CPU/GPU time it takes to render. Each light effect makes your videocard to process more shader instructions, some effects are very heavy shader-wise. When working with lighting, be sure to not abuse maps with lots of dynamic lightsources and effects seen at same time as this may overwhelm your videocard and significatly drop FPS. Use the static lighting as much as possible - lightmaps with deluxemaps have the same look as realtime lightsources.

RGB color

This is just like material system's RGB, with several exceptions and additions. 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. Unlike art programs, which shows color values between 0 and 255, Darkplaces have color mapped between 0 - 1 ('1 1 1' is white). Values lesser than 0 means negative color (assigning it to light source make it cast darkness). Values greater than 1 (overbright color) is supported too.

Static vs Dynamic

Darkplaces lighting is not unified. It combines several techniques in order to get best performance and visual quality. There is no need to light up whole level with hi-quality realtime lighting, as it will take too much CPU and GPU time. Small objects should receive a grude and fast lighting, while important objects should

have best quality. To reach this goal, Darkplaces lighting have two parts - static lighting and dynamic lighting.

For static lighting illumination data is generated when map is being compiled. All objects that is not moving, rotating, animating, transforming in any way, by default, should be illuminated with static lighting. Also any light that is not moving, switching, animating, should be static too. The trick is that both of previous rules should be used synchronously. If there is some dynamic object with many static objects, and it cannot be lit well with static lighting, should make all scene illuminated with dynamic light.

Map compiler (for Blood Omnicide it is Q3Map2) is a tool that generates static lighting data. It is a full featured raytracer able to render non-point sources, area sources, spot lights, alpha shadows, can store light vectors in deluxemaps (in order to get normalmapping and specular working for lightmapped surfaces) etc. All this features makes static lighting to look same as dynamic, but render alot faster. There is 3 techniques available:

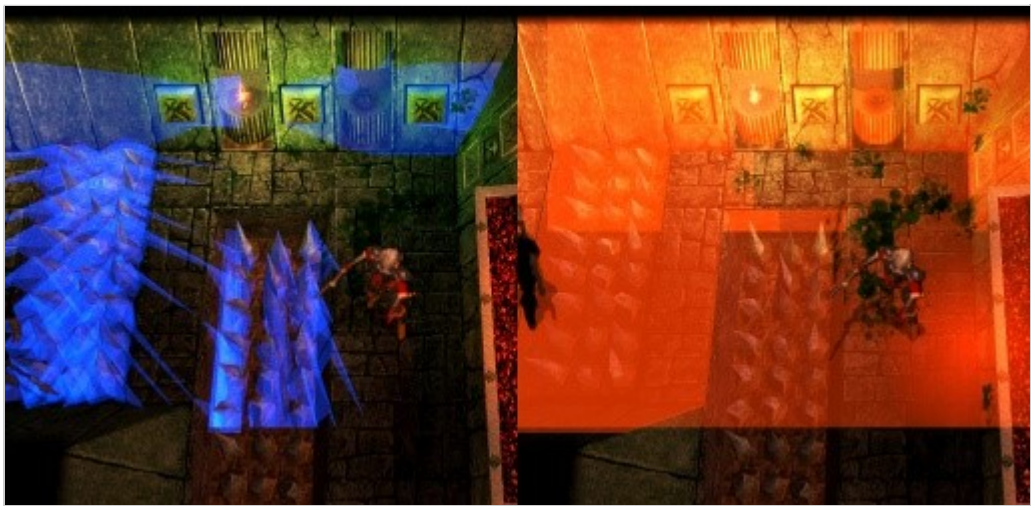
- Vertex lighting - lighting sampled at vertexes, most faster lighting storing only low frequency data (~2 times faster than lightmap)
- Lightmaps - light textures, can store high frequency data and fast to render
- Lightgrid - generated 3d grid storing lighting for moving entities (lighting are sampled at entity position)

Dynamic lighting comes from dynamic lightsources and can illuminate objects that is moving, rotating, animating, transforming in any way. It is very flexible and time consuming as it get computed in realtime, each frame. At other features like simulation of non-point sources, spotlights is less efficient with dynamic lighting because of limited computation time (it should be less than several milliseconds).

Here is the table showing features of different techniques:

Task	Static lighting			Dynamic lighting
	Vertex	Lightmap	Lightgrid	
Features (1 - bad, 2 - average, 3 - good, 4 - very good				
Details	1	3	1	4
Render speed	4	3	3	1
Memory cost	4	1	4	4
Shadows	1	3	1	4
Penumbra (non-point light)	1	4	1	3
Max number of lights	4	4	4	1
Surface-based effects (+ : Supported, - : Not supported)				
Bumpmapping	+	+	+	+
Specular	+	+	+	+
Light filters	-	+	-	+
Non-point lights	+	+	+	-
Light styles	-	-	-	+
Moving lights	-	-	-	+
Used to illuminate				
Map surfaces	+	+	-	+
Static BModels	+	+	+	+
Moving/rotating BModels	-	-	+	+
Animated models	-	-	+	+

Lighting model



Lighting visualized. Blue is shadow volumes, bright orange is zones being processed for dynamic realtime lighting.

Darkplaces uses a variation of Phong shading as a default lighting model. It is based on **per-pixel lighting** - a technique of computing lighting equations for each pixel independently. Renderer contains lots of tricks to allow shading to be fast and flexible. Default model is used on both GL2.0 rendering paths and pre-shader rendering paths (GL1.4), although not all lighting features are possible on pre-shader fixed rendering pipeline.

Lighting are breaked into the passes:

1. Static lighting (lightmap or lightgrid or vertex lighting)
2. Dynamic light 1
3. Dynamic light 2
4. ...
5. Dynamic light X
6. Photon Mapping (optional)
7. Fog (optional)

Data used for each light pass:

- Vertex normal : values between vertices of pixel's triangle are interpolated using Gouraud shading (http://en.wikipedia.org/wiki/Gouraud_shading)
- Color Map : base texture (best way to use diffuse texture)
- Normal Map : texture storing additional surface curvature (optionally, alpha storing height)
- Gloss Map : texture used for specular reflection. Contains gloss color (RGB) and optional exponent multiplier (alpha)
- Glow Map : texture's local luminance texture, RGB additive blended texture that entirely ignores shading
- Light Vector : vector of light direction (either calculated for realtime light or got from deluxemap lightmap component)
- Eye Vector : being computed for each pixel, this vector is used for specular calculation.

This gallery illustrates how lighting features changes the quality:

Attenuation-only 'flat' lighting

+ Shadows

+ Shading

+ Specular

+ Photonmapping

Attenuation

Dynamic lights in Darkplaces engine are omnidirectional, using linear falloff.

Spotlights and projected dynamic lights could be simulated with cubemap filters.

Static lights can have a variety of options for controlling attenuation (this options is only used in Q3map2 LIGHT stage):

- target and radius : simulating spot lights with cone falloff
- _sun flag - infinite distance lights, constant direction
- _deviance and _samples - non-point lights

Shading

Shading is done by using dotproduct of light vector and pixel normal. Some ambient lighting could be added for a more smooth result.

Important: Lightmaps by default dont have shading since they lack light direction data. In order to make lightmaps to use whole shading cycle, map compiler tool should be configured to make deluxemaps (second lightmap set which store per-pixel light directions)

Rough shading algorithm explanation:

```
// Surface's vertex normals are always supplied, texture normalmap is forced to default RGB '0.5 0.5 1' (same as surfac
PixelNormal = Surface.VertexNormal + Texture.NormalMap;
// LightSource are only defined for dynamic lights
// LightmapGlobal.AmbientLight is r_ambient cvar
// Material.AmbientLight is set by dprtlightambient material keyword and only applicable to dynamic lights
AmbientLight = LightSource.AmbientLight + LightmapGlobal.AmbientLight + Material.AmbientLight;
if (LightSource)
```

```
    DiffuseLight = LightSource.DiffuseLight;
else
    DiffuseLight = 1; // Static lighting
Shading = dotproduct(PixelNormal, LightVector) * DiffuseLight + AmbientLight;
```

- **r_shadow_usenormalmap** : Enables use of normalmap texture for lighting. Setting this to to will make only vertex normals to be used.
- **r_shadow_bumpscale_basetexture** : Forces normalmap to be automatically calculated from color texture by using it as bumpmap, value controls magnitude. Requires r_restart.
- **r_shadow_bumpscale_bumpmap** : Normalmap can be supplied as _bump texture, in this case engine converts it to normalmap on load, values is magnitude. Requires r_restart.
- **r_glsl_deluxemapping** : Use full shading on lightmapped surfaces. A value of 2 forces deluxemap shading even if surface have no one.
- **r_ambient** : Amount of ambient light to be added to all surfaces. Makes level brighter.

Specular

Specular (or gloss) is the mirror-like reflection of light from a surface, in which light from a single incoming direction (a ray) is reflected into a single outgoing direction. Many well-known surfaces (metal, plastic, wood) is recognized by specularity they have.

Specular offers two parameters to mess with:

- Specular multiplier: intensity of specular effect, this could be very high or very low. It is multiplier to gloss texture.
- Specular exponent: how 'sharp' gloss is, high values are used to make texture to be plastic-like, while lower ones are suitable for matte surfaces. Basically its just a modifier to gloss texture's alpha channel (which is forced to 1 if not supplied).

Gloss may be forced (see r_shadow_gloss 2 below), in this case texture, if missing its own gloss map, gets a white image for gloss map and parameters from a cvars.

Rough specular algorithm explanation:

```
// Global.GlossIntensity is controlled by r_shadow_glossintensity or r_shadow_glossintensity2 if gloss is forced
// Material.GlossIntensityMod is set by dpglossintensitymod material keyword
SpecularColor = Texture.GlossMap * Global.GlossIntensity * Material.GlossIntensityMod;
// Global.GlossExponent is controlled by r_shadow_glossexponent or r_shadow_glossexponent2 if gloss is forced
// Material.GlossExponentModis set by dpglossexponentmod material keyword
if (Texture.GlossMap.Alpha)
    SpecularExponent = Texture.GlossMap.Alpha * Global.GlossExponent * Material.GlossExponentMod;
else
    SpecularExponent = Global.GlossExponent * Material.GlossExponentMod;
// this is rough specular calculation
// optionally, engine can use real reflection map to get specular normal (see r_shadow_glossexact below)
SpecularNormal = PixelNormal + EyeVector;
Specular = SpecularColor * power(dotproduct(PixelNormal, SpecularNormal), SpecularExponent)
```

Important: Forced gloss (gloss 2) which is used if texture's gloss map is missed, are only used on outside map to simulate wet surfaces effect.

- **r_shadow_glossintensity** : Global intensity for specular calculations, default is 1.
- **r_shadow_gloss2intensity** : Global intensity for specular calculations applied for forced-gloss surfaces, default is 1.
- **r_shadow_glossexponent** : Global gloss exponent used as a base in shader calculations.
- **r_shadow_gloss2exponent** : Same one used for forced-gloss surfaces.

- **r_shadow_glossexact** : Use real reflection math for gloss calculation. This is slower and little more correct.

Shadows

Shadows are most valuable part of realtime lighting. They are increasing scene depth so it looks more realistic.

Important: Shadows are quite complex render task, many lights casting many shadows may decrease rendering speed significantly. Map designer should plan his map with this limitation in mind - there should be no situation of many lights being seen from a certain point, or user will experience a game slowdown.

Darkplaces supports two realtime shadowing techniques: stencil shadow volumes and shadow mapping.

Stencil shadow volumes

Stencil shadow volumes (http://en.wikipedia.org/wiki/Shadow_volume) is a base shadow rendering method in Darkplaces.

This technique is well known for it's shadows not having penumbra. Many other restrictions (high fillrate hit, bad scalability) make this method to be used only on a few games (such as Doom 3).

- **r_shadow_polygonfactor** : how much to enlarge shadow volume polygons when rendering (should be 0!)
- **r_shadow_polygonoffset** : how much to push shadow volumes into the distance when rendering, to reduce chances of zfighting artifacts (should not be less than 0)
- **r_shadow_frontsidecasting** : whether to cast shadows from illuminated triangles (front side of model) or unlit triangles (back side of model)
- **r_showshadows** : Show areas covered by shadow volumes. Useful for finding out why some areas of the map render slowly (bright blue = lots of passes, slow). Only matters if using shadow volumes.

Shadow mapping

In 2010 darkplaces got shadow mapping (http://en.wikipedia.org/wiki/Shadow_mapping) implemented by Eihrul (http://sauerbraten.org/) .

Shadowmapping have a number of advantages over shadow volume rendering and is considered to replace it:

- Penumbra
- Fast to render (especially on complex area maps)
- Takes less CPU time (as construction of shadow volumes is not required)
- Distance-based LOD (far lights rendered with lower shadowmap resolution)

Console variables

- **r_shadow_shadowmapping** : Enables shadow mapping
- **r_shadow_shadowmapping** : Shadowmap bias parameter (this is multiplied by nearclip * 1024 / lodsize
- **r_shadow_shadowmapping_bordersize** : Shadowmap size bias for filtering

- **r_shadow_shadowmapping_depthbits** : Requested minimum shadowmap texture depth bits, could be 16 or 24
- **r_shadow_shadowmapping_filterquality** : Shadowmap filter modes: -1 = auto-select, 0 = no filtering, 1 = bilinear, 2 = bilinear 2x2 blur (fast), 3 = 3x3 blur (moderate), 4 = 4x4 blur (slow)
- **r_shadow_shadowmapping_maxsize** : Shadowmap size limit
- **r_shadow_shadowmapping_minsize** : Shadowmap size limit
- **r_shadow_shadowmapping_nearclip** : Shadowmap near clip in world units. Increasing this will make shadow rendering to be more precise (as more bits goes to middle range), at the cost of the small non-shadowed zone around light
- **r_shadow_shadowmapping_polygonfactor** : Slope-dependent shadow mapping bias
- **r_shadow_shadowmapping_polygonoffset** : Constant shadow mapping bias
- **r_shadow_shadowmapping_precision** : Makes shadowmaps have a maximum resolution of this number of pixels per light source radius unit such that, for example, at precision 0.5 a light with radius 200 will have a maximum resolution of 100 pixels
- **r_shadow_shadowmapping_useshadowsampler** : whether to use sampler2DShadow if available
- **r_shadow_shadowmapping_vsdct** : enables use of virtual shadow depth cube texture
- **r_usedepthtextures** : Use depth texture instead of depth renderbuffer where possible, uses less video memory but may render slower (or faster) depending on hardwaree.

Bug: r_usedepthtextures causing strong precision drop, which leads to splotches around shadow edges

Other light models

Darkplaces includes other lighting models which are used eventually:

Fake light

A developer-only mode which is forced on non-lit maps (ones which was compiled and didnt get LIGHT phase). This is infinite realtime light that is cast from eye position.

Sun light

A similar to fake light technique to render sun lighting on outdoor surfaces, Used for sunset and sun dawn effects on outdoor maps.

Cel shading

Altered shading for default lighting model.

- **r_celshading** : Enable cel shading (alternate diffuse math)

Lightmap

Lightmap with no deluxemap applied (just an old lightmap with no per-pixel lighting effects)

Fullbright

No lighting applied (all textures at their full brightness)

Special effects

This chapter contains reference of various special effects hardcoded in light renderer. This special effects, applied or combined will help to create wide range of things, such as:switchable lights, projective lights, sunlight simulation, flickering lights and more.

Light styles

Stub!

Coronas

Coronas aimed to simulate glow around dynamic lights, which looks good on small and medium size lights. Coronas are customised per-light (size and brightness) an also can be placed without lights.

- **r_coronas** : Brightness of corona effects. 0 disables coronas.
- **r_coronas_occlusionquery** : Fades coronas according to visibility. Bad perormance (synchoronus rendering), even worse on multi-GPU.
- **r_coronas_occlusionquerysize** : Size of lightsource for corona occlusion checksum. Usial value is 0.1

Cubemap filtering

A technique for modeling non-uniform light distribution according to direction, for example a lantern may use a cubemap to describe the light emission pattern of the cage around the lantern (as well as soot buildup discoloring the light in certain areas), often also used for softened grate shadows and light shining through a stained glass window (done crudely by texturing the lighting with a cubemap), another good example would be a discolight.

Any dynamic lightsource can have cubemap attached.

Planar shadows

Simplified global shadows which are cast from entities, not from lights (so planar shadows are just filters, they dont do any lighting math).

Planar shadows are either stencil or shadowmapped if shadowmapping is on.

Tip: Planar shadows are used to simulate sunlight in outdoor locations.

- **r_shadows** : Enable planar shadows cast using lightgrid-stored light direction. When set to 2 always cast the shadows in the certain direction (see below). Shadowmapping only supports r_shadows 2.
- **r_shadows_castfrombmodel** : Enables shadows cast from bmodels.
- **r_shadows_darken** : How much shadowed areas will be darkened.
- **r_shadows_drawafterrtlighting** : Hack to draw fake shadows AFTER realtime lightning is drawn. May be useful for simulating sunlight on large outdoor maps with only one big noshadow rtlight. The price is less realistic appearance of dynamic light shadows.
- **r_shadows_focus** : Offset the shadowed area focus (used for shadowed area bounds).
- **r_shadows_shadowmapscale** : Increases shadowmap quality (multiply global shadowmap precision). Needs shadowmapping ON.

- **r_shadows_throwdirection** : r_shadows 2 throwing direction. Default is '0 0 -1' (down).
- **r_shadows_throwdistance** : How far to cast shadows from models. This sets shadowed area bounds for shadowmapping.

Customized ModelLight

Lightgrid sampling (diffuse, ambient and light vector components) can be overriden by Client-side QuakeC. Allows various lighting effects on models (strobing, using different positions for sampling etc.)

Fog

Fog are global. Optionally, height plane and height texture can be defined which will make fog to fade with height.

Fog have the following options:

- density - how much fog to add per distance, this is virtual value. 1 means very strong fog.
- red - red component of fog color
- green - green component of fog color
- blue - blue component of fog color
- alpha - fog effect opacity (default is 1)
- mindist - distance to start applying fog at
- maxdist - maximal distance apply fog at
- top - origin of height plane at worldspace (game units)
- height - fog height fade in game units
- fadetexture - horizontal texture that resembles fading, left pixel if top bright, right is top dark. Can be used for non-linear fading.

Fog are set by console commands:

- **fog <density> <red> <green> <blue> [<alpha> <mindist> <maxdist> <top> <height>]** : Sets the global fog
- **fog_heighttexture <density> <red> <green> <blue> <alpha> <mindist> <maxdist> <top> <height> <fadetexture>** : Sets the global fog with customized fade texture

---

Light sources

Contents

Static lightsources

Dynamic lightsources

World lights

Worlds lights are loaded from external .rtlights file supplied with a map (foo.bsp will try to load foo.rtlights). If there are not .rtlights file, engine tries to load lights from a map "light" entities, which often is grude and terrible slow (as "lights" are maked

for map compiler LIGHT stage raytracer, not for reatime rendering). So producing a good .rtlights folr is very first step in a process of map creation.

World lights are static, meaning they canot be moved, rotated, altered during a game.

Tip: Blood Omnicide is using world lights to place lights for a Q3map2 LIGHT phase. So unlike Quake, which should have both "light" entities and .rtlights, Blood Omnicide .rtlights containins all map static lights

Each world light can have this parameters:

- origin : position
- angles : rotation
- color : RGB light color
- radius : radius of light in game units
- corona : corona intensity (0 to disable)
- coronasize : size of corona (this is multiplier to light radius)
- style : lights style number (see lightstyles above)
- shadows: whether to cast shadows from light
- cubemap : a path to cubemap
- ambient : ambient light intensity (ignores shading, makes lighting to be more 'flat')
- diffuse : shading light intensity (default is 1)
- specular : specular intensity (default is 1)
- normalmode : if this flag is set, light will be draws if cvar r_shadow_realtime_world is 0
- realtimemode : if this flag is set, light will be draws if cvar r_shadow_realtime_world is 1

Tip: A combination of normalmode = 0, realtimemode = 0 to mark lights which are only appear on lightmap

Tip: Light style 1 will make a dynamic lights that will not appear on lightmap

Tip: Cubemap-filtered lights dont appear on lightmap

Console variables:

- **r_shadow_realtime_world** : Enables rendering of full world lighting (whether loaded from the map, or a .rtlights file, or a .ent file, or a .lights file produced by hlight)
- **r_shadow_realtime_world_lightmaps** : Brightness to render lightmaps when using full world lighting
- **r_shadow_realtime_world_shadows** : Enables rendering of shadows from world lights
- **r_shadow_realtime_world_compile** : Enables compilation of world lights for higher performance rendering (shadow volumes only)
- **r_shadow_realtime_world_compileportalculling** : Enables portal-based culling optimization during compilation (overrides compilesvbsp)
- **r_shadow_realtime_world_compileshadow** : Enables compilation of shadows from world lights for higher performance rendering (shadow volumes only)
- **r_shadow_realtime_world_compilesvbsp** : Enables svbsp optimization during compilation (slower than compileportalculling but more exact)

Tip: In Blood Omnicide, this console varisables are hardcoded since its using a combination of full realtime lightning and lightmaps. Tweak this settings only for debugging purposes or if you want to break something.

Dynamic lights

Dynamic lights are cast from entities. This could be both server-side entities such as rockets or client-side entities such as explosions.

- **r_shadow_realtime_dlight** : Enables rendering of dynamic lights such as explosions and rocket light
- **r_shadow_realtime_dlight_portalculling** : Enables portal optimization on dynamic lights
- **r_shadow_realtime_dlight_shadows** : Enables rendering of shadows from dynamic lights
- **r_shadow_realtime_dlight_svbspculling** : Enables svbsp optimization on dynamic lights

---

Additional options

Darkplaces lighting renderer have a bunch of misc options to tweak.

Optimizations

For best performance, all optimisations should be on.

- **r_shadow_scissor** : Use scissor optimization of light rendering (restricts rendering to the portion of the screen affected by the light)
- **r_shadow_sortsurfaces** : Improve performance by sorting illuminated surfaces by texture
- **r_shadow_usebihculling** : Use Bounding Interval Hierarchy for culling lit surfaces instead of Binary Space Partitioning

Light attenuation

This console variables change attenuation for all dynamic lights.

- **r_shadow_lightattenuationdividebias** : Controls the speed of light fading, 1 is default linear fade.
- **r_shadow_lightattenuationlinearscale** : Linear scale applied during attenuation, larger values will make lesser half-light zones
- **r_shadow_lightintensityscale** : Renders all lights brighter or darker
- **r_shadow_lightradiusscale** : Renders all world lights larger or smaller

---