Chapter 7

Shading in Valve's Source Engine

Jason Mitchell[9], Gary McTaggart[10] and Chris Green[11]

VALVE

7.1 Introduction

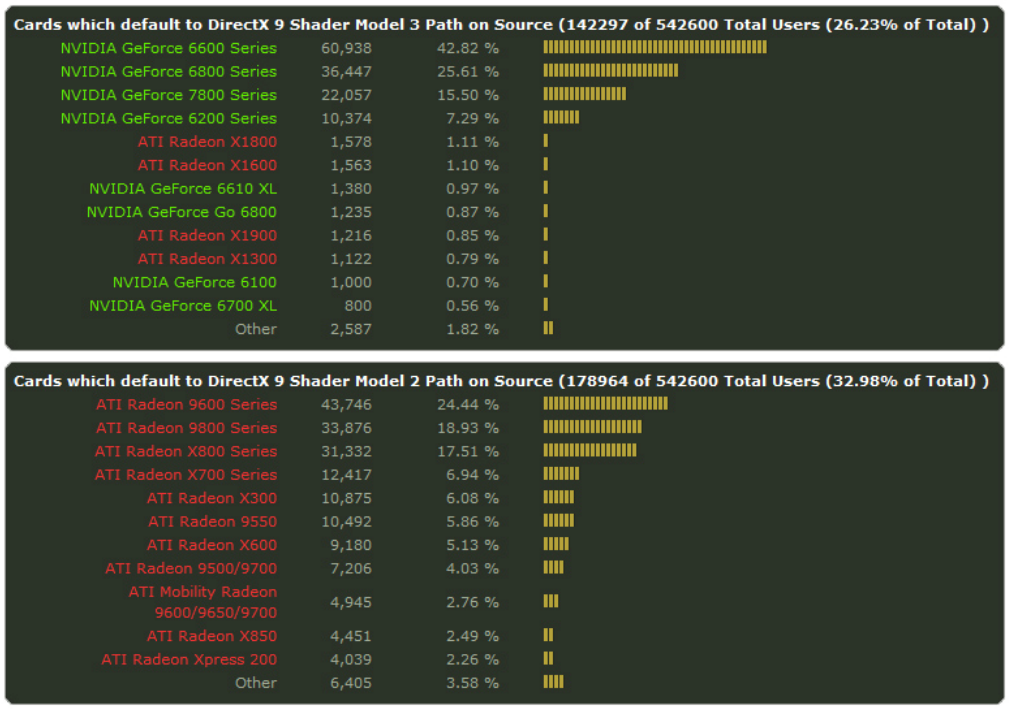
Starting with the release of *Half-Life 2* in November 2004, Valve has been shipping games based upon its Source game engine. Other Valve titles using this engine include *Counter-Strike: Source*, *Lost Coast*, *Day of Defeat: Source* and the recent *Half-Life 2: Episode 1*. At the time that *Half-Life 2* shipped, the key innovation of the Source engine's rendering system was a novel world lighting system called *Radiosity Normal Mapping*. This technique uses a novel basis to economically combine the soft realistic

[9] jasonm@valvesoftware.com
[10] gary@valvesoftware.com
[11] cgreen@valvesoftware.com

lighting of radiosity with the reusable high frequency detail provided by normal mapping. In order for our characters to integrate naturally with our radiosity normal mapped scenes, we used an *irradiance volume* to provide directional ambient illumination in addition to a small number of local lights for our characters. With Valve's recent shift to episodic content development, we have focused on incremental technology updates to the Source engine. For example, in the fall of 2005, we shipped an additional free *Half-Life 2* game level called *Lost Coast* and the multiplayer game *Day of Defeat: Source*. Both of these titles featured real-time High Dynamic Range (HDR) rendering and the latter also showcased the addition of real-time c*olor correction* to the engine. In these course notes, we will describe the unique aspects Valve's shading techniques in detail.

7.2 Engineering / Marketplace Context

Before describing the details of the Source engine's shading techniques, it is important to note that the decision to include particular features must be made in the context of the marketplace. In order to make informed technology tradeoffs, we periodically perform a voluntary survey of approximately one million of our users' PCs via our online distribution system, *Steam*. This survey, which is publicly available at http://www.steampowered.com/status/survey.html, can tell us, for example, how many of our users have video cards capable of running ps_2_b pixel shaders (40%) or how many of our users have 256MB or more video memory (38%). Having an accurate picture of our users' hardware helps us to weigh the engineering cost of adopting a given graphics feature against the benefit to our end users. All of the decisions that drive development in the Source engine are based upon these kinds of measured tradeoffs.



Cards which default to DirectX 9 Shader Model 3 Path on Source (142297 of 542600 Total Users (26.23% of Total))		
NVIDIA GeForce 6600 Series	60,938	42.82 %
NVIDIA GeForce 6800 Series	36,447	25.61 %
NVIDIA GeForce 7800 Series	22,057	15.50 %
NVIDIA GeForce 6200 Series	10,374	7.29 %
ATI Radeon X1800	1,578	1.11 %
ATI Radeon X1600	1,563	1.10 %
NVIDIA GeForce 6610 XL	1,380	0.97 %
NVIDIA GeForce Go 6800	1,235	0.87 %
ATI Radeon X1900	1,216	0.85 %
ATI Radeon X1300	1,122	0.79 %
NVIDIA GeForce 6100	1,000	0.70 %
NVIDIA GeForce 6700 XL	800	0.56 %
Other	2,587	1.82 %

Cards which default to DirectX 9 Shader Model 2 Path on Source (178964 of 542600 Total Users (32.98% of Total))		
ATI Radeon 9600 Series	43,746	24.44 %
ATI Radeon 9800 Series	33,876	18.93 %
ATI Radeon X800 Series	31,332	17.51 %
ATI Radeon X700 Series	12,417	6.94 %
ATI Radeon X300	10,875	6.08 %
ATI Radeon 9550	10,492	5.86 %
ATI Radeon X600	9,180	5.13 %
ATI Radeon 9500/9700	7,206	4.03 %
ATI Mobility Radeon 9600/9650/9700	4,945	2.76 %
ATI Radeon X850	4,451	2.49 %
ATI Radeon Xpress 200	4,039	2.26 %
Other	6,405	3.58 %

Figure 1. *Typical Survey Measurements - sampled from 1 million gamers in Spring 2006*

7.3 World Lighting

For *Half-Life 2*, we leveraged the power of light maps generated by our in-house radiosity solver, *vrad*, while adding high frequency detail using per-pixel normal information. On its own, radiosity has a lot of nice properties for interactive content in that it is possible to pre-generate relatively low resolution light maps which result in convincing global illumination effects at a low run-time cost. A global illumination solution like this requires less micro-management of light sources during content production and often avoids objectionable harsh lighting situations as shown in the images below.

Harsh Direct Lighting Only

Radiosity

Figure 2*. Benefits of Radiosity Lighting*

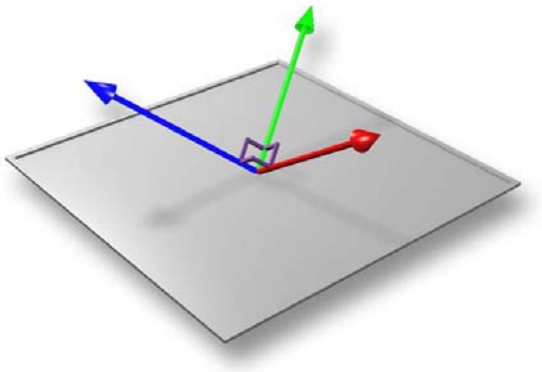
Because of the unpredictable interactive nature of our medium—we don't know where the camera will be, since the player is in control—we can't tune lights shot-by-shot as in film production. Global illumination tends to result in game environments that look pleasing under a variety of viewing conditions. Because of this, heavily favoring the look of global illumination is a common theme in Valve's shading philosophy, as we will discuss later in the section on model lighting via an irradiance volume.

7.3.1 Radiosity Normal Mapping

While radiosity has a lot of strengths, the resulting low-resolution light maps can't capture the high-frequency detail we would like to author into our surfaces. Even at relatively modest luxel density—a luxel is typically 4" by 4" in our game worlds—light maps require a significant amount of video memory in a typical game level since they are unique to a particular polygon or location in the world. While many current games accumulate one local light per rendering pass [Diefenbach97] or perform several local lighting computations in a single pixel shader [Franke06], we seek to effectively perform diffuse bump mapping with respect to an arbitrary number of lights in one pass.

With high frequency surface detail in the form of repeatable/reusable albedo and normal maps that are higher resolution than our 4" by 4" per luxel light maps, we combined the strengths of normal mapping and light mapping in a novel technique we call *Radiosity Normal Mapping*. Radiosity Normal Mapping is the key to the Source engine's world lighting algorithm and overall look [McTaggart04].

The core idea of Radiosity Normal Mapping is the encoding of light maps in a novel basis which allows us to express directionality of incoming radiance, not just the total cosine weighted incident radiance, which is where most light mapping techniques stop. Our offline radiosity solver has been modified to operate on light maps which encode directionality using the basis shown in the figure below.



$$\left\{-\frac{1}{\sqrt{6}},\frac{1}{\sqrt{2}},\frac{1}{\sqrt{3}}\right\}$$

$$\left\{-\frac{1}{\sqrt{6}},-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{3}}\right\}$$

$$\left\{\sqrt{\frac{2}{3}},0,\frac{1}{\sqrt{3}}\right\}$$

Figure 3. *Radiosity Normal Mapping Basis*

That is, at the cost of tripling the memory footprint of our light maps, we have been able to encode directionality in a way that is compatible with conventional Cartesian normal maps stored in surface local coordinates. Because of this encoding, we are able to compute exit radiance as a function of albedo, normal and our directional light maps so that we can diffusely illuminate normal mapped world geometry with respect to an arbitrarily complex lighting environment in a few pixel shader instructions. Additionally, these same normal maps are used in other routines in our pixel shaders, such as those which compute Fresnel terms or access cubic environment maps for specular illumination.

7.3.2 World Specular Lighting

In keeping with our general philosophy of favoring complex global illumination and image based lighting techniques over the accumulation of a few simple local lights, we have chosen to use cubic environment maps for specular world lighting. In order to keep the memory cost of storing a large number of cube maps at a reasonable level, our level designers manually place point entities in their maps which are used as sample points for specular lighting. High dynamic range cube maps are pre-rendered from these positions using the Source engine itself and are stored as part of the static level data. At

run time, world surfaces which are assigned a material requiring a specular term use either the cube map from the nearest sample point or may have one manually assigned in order to address boundary conditions.

7.4 Model Lighting

In order to light our characters and other models that can animate dynamically within our game worlds, we have employed a system that combines spatially varying *directional irradiance samples* from our radiosity solution with local light sources and environment mapping. On graphics hardware that supports 2.0 pixel shaders and lower, we compute diffuse illumination from up to two local light sources, either per pixel or per vertex. On newer graphics chips with longer pixel shaders, we can afford to include specular contributions with Fresnel terms as well as more custom effects.

7.4.1 Ambient Cube

At Valve, we heavily favor the look of global illumination in order to ground characters in our game worlds and make them seem to be truly present in their environment. The most significant component of *Half-Life 2* character and model lighting is the inclusion of a directional ambient term from *directional irradiance samples* precomputed throughout our environments and stored in an irradiance volume as in [Greger98]. While most games today still use a few local lights and a gray ambient term, there have been a few examples of other games including a more sophisticated ambient term. For example, characters in many light mapped games such as *Quake III* include an ambient term looked up from the light map of the surface the character is standing on [Hook99]. Most similarly to the Source engine, the game *Max Payne 2* stored linear 4-term spherical harmonic coefficients, which effectively reduce to a pair of directional lights, in an irradiance volume throughout their game worlds for use in their character lighting model [Lehtinen06].

Due to the particular representation of our irradiance samples (essentially low resolution cubic *irradiance environment maps* [Ramamoorthi01]) and due to the fact that they are computed so that they represent only indirect or bounced light, we call the samples *Ambient Cubes*. The evaluation of the ambient cube lighting term, which can be performed either per pixel or per vertex, is a simple weighted blending of six colors as a function of a world space normal direction as illustrated in the following HLSL routine.

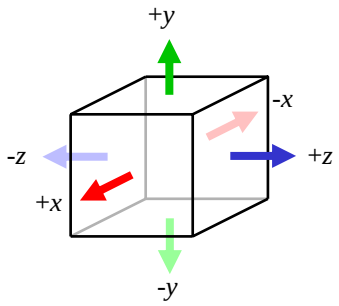
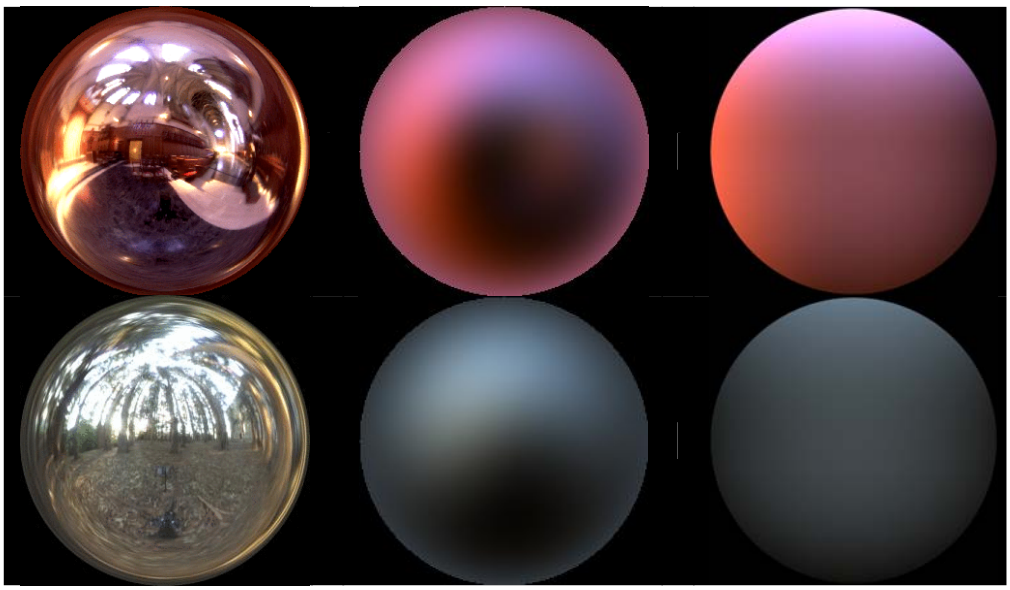


Figure 4. *Ambient cube basis*

```
float3 AmbientLight( const float3 worldNormal )
{
      float3 nSquared = worldNormal * worldNormal;
      int3 isNegative = ( worldNormal < 0.0 );
      float3 linearColor;
      linearColor = nSquared.x * cAmbientCube[isNegative.x]   +
                    nSquared.y * cAmbientCube[isNegative.y+2] +
                    nSquared.z * cAmbientCube[isNegative.z+4];
      return linearColor;
}
```

Listing 1. *Ambient Cube Routine (6 vertex shader instructions or 11 pixel shader instructions)*

The ambient cube is intuitive and easy to evaluate in either a pixel or vertex shader, since the six required colors can be loaded into shader constants. As mentioned above, we can afford to evaluate only two local diffuse lights on our target graphics hardware. In addition to this, any significant lights beyond the most important two are assumed to be non-local and are combined with the ambient cube on the fly for a given model. These six RGB colors are a more concise choice of basis than the first two orders of spherical harmonics (nine RGB colors) and this technique was developed in parallel with much of the recent literature on spherical harmonics such as [Ramamoorthi01]. As shown in Figure 5 below, we could potentially improve the fidelity of our indirect lighting term by replacing this term with spherical harmonics. With the increased size of the ps_3_0 constant register file (up to 224 from 32 on ps_2_*x* models) this may be a good opportunity for improvement in our ambient term going forward.



Original Light Probe	Irradiance environment maps from [Ramamoorthi01]	Irradiance environment maps represented using Valve ambient cube

Figure 5. *Comparison of spherical harmonics with Valve Ambient Cube*

7.4.2 Irradiance Volume

The *directional irradiance samples* or ambient cube sample points are stored in a 3D spatial data structure which spans our game environments. The samples are stored non-hierarchically at a fairly coarse (4' x 4' x 8') granularity since indirect illumination varies slowly through space and we only need a fast lookup, not extreme accuracy. When drawing any model that moves in our environment, such as the character Alyx in the figure below, the nearest sample point to the model's centroid is selected from the irradiance volume without any spatial filtering. Of course, this can result in hard transitions in the directional ambient component of illumination as the character or object moves throughout the environment, but we choose to time-average the ambient and local lighting information to limit the effect of lighting discontinuities, lessening the visual popping in practice. On important hero characters such as Alyx, we also heuristically boost the contribution of the directional ambient term as a function of the direct lights so that we don't end up with the direct illumination swamping the ambient and causing a harsh appearance.

Without boosted ambient term

With boosted ambient term

Figure 6*. Alyx illuminated with two distant direct lights and directional ambient term*

7.4.3 Local lighting

As mentioned above, the ambient cube data precomputed throughout our game worlds incorporates only bounced or indirect lighting. For direct lighting from local lights, we add in modified per-pixel Lambertian and Phong terms. On ps_2_0 video cards (20% of our users) we compute diffuse illumination for up to two local lights. Any nearby local lights beyond the two most significant lights are added into the ambient cube on the fly. Starting with *Half-Life 2: Episode 1*, on video cards which support ps_2_b or ps_3_0 shader models (40% of our users), the Source engine also computes Phong terms for the two most significant local lights.

7.4.4 Half Lambert Diffuse

In order to soften the diffuse contribution from local lights, we scale the dot product from the Lambertian model by ½, add ½ and square the result so that this dot product, the red cosine lobe in Figure 7 at right, which normally lies in the range of -1 to +1, is instead in the range of 0 to 1 and has a pleasing falloff (the blue curve in Figure 7). This technique, which we refer to as *Half Lambert* prevents the back side of an object with respect to a given light source from losing its shape and looking too flat. This is a completely non-physical technique but the perceptual benefit is enormous given the trivial implementation cost. In the original game *Half-Life*, where this diffuse Half Lambert term was the only term in most of the model lighting, it kept the characters in particular from being lit exclusively by a constant ambient term on the back side with respect to a given light source.

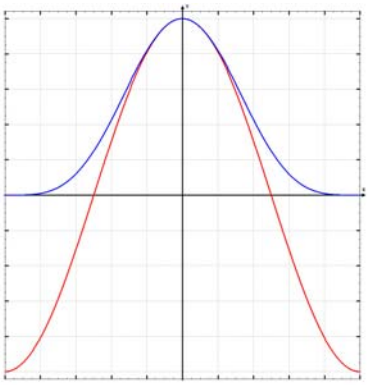
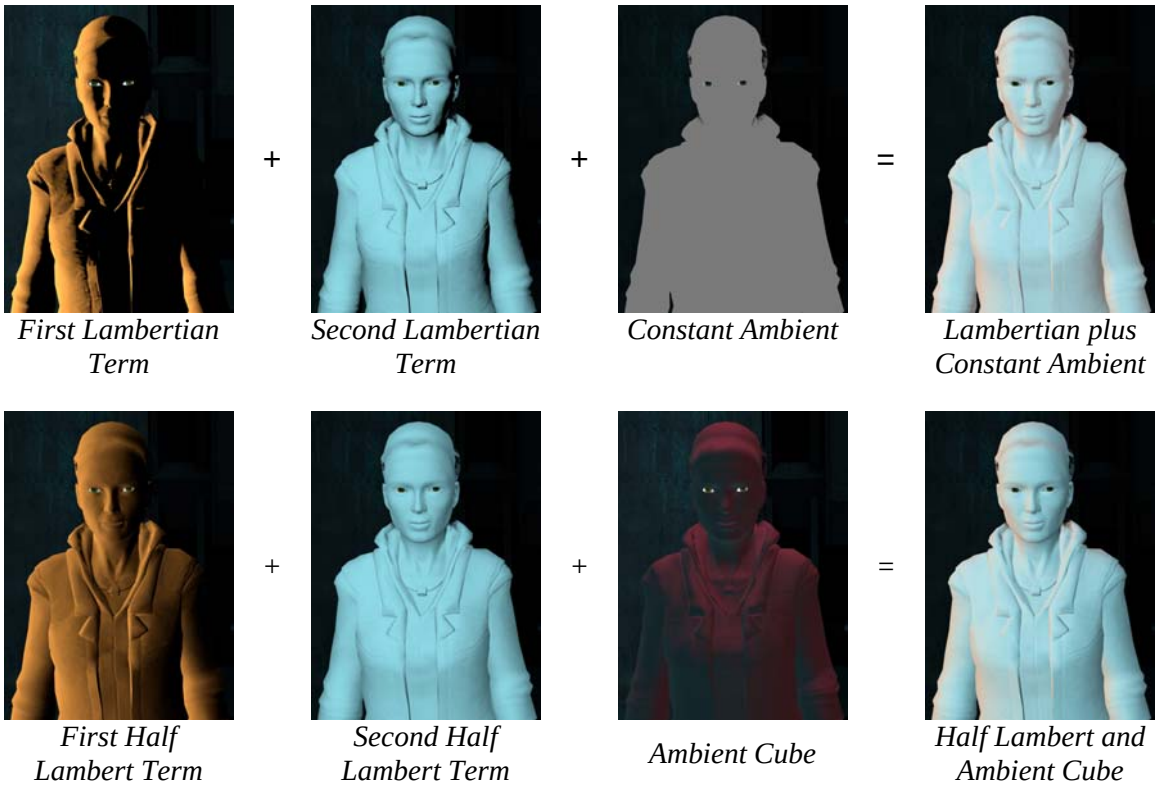


Figure 7*. Cosine lobe (red) Half-Lambert function (blue)*



First Lambertian Term | *Second Lambertian Term* | *Constant Ambient* | *Lambertian plus Constant Ambient*

First Half Lambert Term | *Second Half Lambert Term* | *Ambient Cube* | *Half Lambert and Ambient Cube*

Figure 8. *Comparison of traditional Lambertian and constant ambient terms with Half Lambert and Ambient Cube terms for diffuse lighting*

Diffuse Lighting from Figure 8 Above * *Albedo* = *Diffuse Lighting*

Figure 9. *Diffuse character lit by two Half-Lambert terms and ambient cube*

7.4.5 Local Specular

On pixel shader models ps_2_b and ps_3_0, our shading algorithms are no longer limited by shader length, so an easy way to increase visual quality is the inclusion of specular terms where appropriate. Starting with *Half-Life 2: Episode 1*, we have given our artists the ability to include Phong terms which can be driven by specular mask and exponent textures. As you can see in the images in Figure 11, this term integrates naturally with our existing lighting and provides greater definition, particularly to our heroine, Alyx Vance. In order to keep the specular terms from causing objects or characters from looking too much like plastic, we typically keep the specular masks (Figure 10c) fairly dim and also ensure that the specular exponents are low in order to get broad highlights. We also include a Fresnel term (Figure 10b) in our local specular computations in order to heavily favor rim lighting cases as opposed to specular highlights from lights aligned with the view direction. Under very bright lights, the specular rim highlights are often bright enough to bloom out in a dramatic fashion as a result of the HDR post-processing techniques which we will discuss in the following section.

(a) Two Phong Lights * *(b) Fresnel Term* * *(c) Specular Mask* = *(d) Final Specular*

Figure 10*. Specular lighting in* Half-Life 2: Episode 1

(a) Diffuse from Figure 9 *(b) Specular from Figure 10* *(c) Final Lighting*

Figure 11. *Specular lighting term added for final character lighting equation. In* Half-Life 2: Episode 1*, 60% of our current users will see the image in (a) while the 40% with hardware supporting ps_2_b and higher shaders will see the additional specular lighting in (c).*

7.5 High Dynamic Range Rendering

After shipping Half-Life 2, we implemented High Dynamic Range (HDR) rendering in the Source engine using a novel method which runs on graphics cards which support 2.0 pixel shaders and 16-bit per channel source textures [Green06]. This means that a whopping 60% of our users get to experience our games in HDR today. This technology was used to create the *Lost Coast* demo and has been used in all subsequent Valve games including *Day of Defeat: Source* and the recent episodic release *Half-Life 2: Episode 1*.

Many different methods for performing HDR rendering on modern GPUs were implemented and evaluated before finding the set of tradeoffs most appropriate for rendering our HDR content on current GPUs. Our implementation makes use of primarily 8-bit per channel textures with 16-bit per channel (either floating point or fixed point depending upon hardware support) light maps and cubic environment maps plus a few 16-bit per channel hand-authored textures for the sky, clouds, and sun.

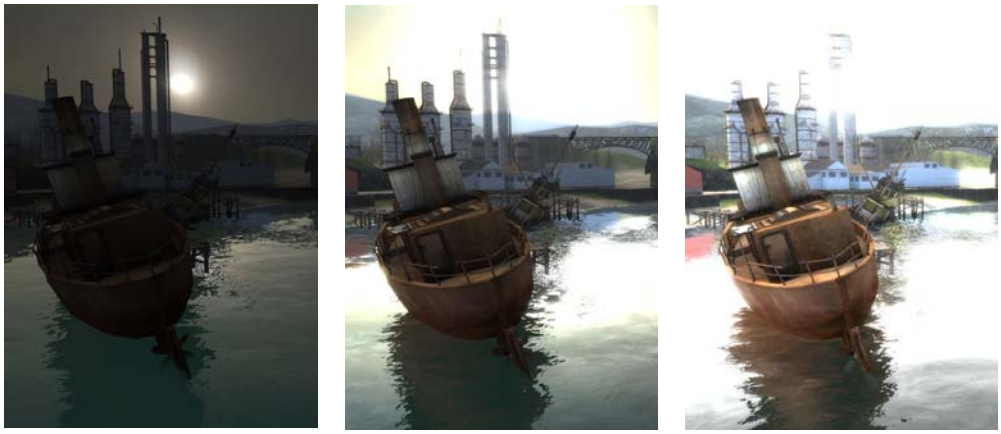


Figure 10*. Scene from Lost Coast at Varying Exposure Levels*

Since all of the shaders in the Source engine's DirectX9 code path are single pass, it is possible to perform HDR lighting and tone mapping operations during normal rendering. This required the addition of a tone mapping subroutine to all pixel shaders but enabled us to render to ordinary 8-bit per channel render targets. By using 8-bit per channel render targets instead of floating point buffers, there is very little performance difference with the low dynamic range path, memory usage is the same and, most importantly, multisample anti-aliasing is supported. This also allows us to avoid an extra copy and conversion of the rendering output to the final display format. When intermediate renderings are used as the inputs for subsequent graphics operations, the dynamic range of these intermediate results is carefully managed so as to still provide high quality outputs from 8-bit data. For instance, renderings done for dynamic reflection in the water have their exposure adjusted by the water's reflectivity coefficient so that proper highlights from the sun are still produced.

When actual full-range HDR rendering outputs are needed (such as when rendering the cube maps used for reflection using the engine), the engine is used to produce a series of renderings at varying exposure levels, which are subsequently combined to produce one floating point HDR output image.

Auto exposure calculations needed for tone mapping are performed in a unique fashion in the Source engine. A pixel shader is applied to the output image to determine which output pixels fall within a specified luminance range and writes a flag into the stencil buffer indicating whether each pixel passed the test. An asynchronous occlusion query is then used to count how many pixels were within the specified range. When the occlusion query's results become available, they are used to update a running luminance histogram. One luminance range test is done per frame, amortizing the cost of generating the histogram over multiple frames. Time averaging is done to smooth out the results of this incremental histogram. This technique has a number of advantages over other methods, including the fact that it computes a full histogram instead of just an average luminance value, and that this histogram is available to the CPU without stalling the graphics pipeline or performing texture reads from GPU memory. In addition to the running histogram, other auto exposure and bloom filter parameters are controllable by level designers and can be authored to vary spatially with our game worlds or in response to game events. In addition to adjusting our tone mapping operations as a function of the running luminance of our scenes, we perform an image-space bloom in

order to further stylize our scenes and increase the perception of brightness for really bright direct or reflected light. In the offline world, such bloom filtering is typically performed in HDR space prior to tone mapping [Debevec00] but we find that tone mapping before blooming is not an objectionable property of our approach, particularly considering the large number of users who get to experience HDR in our games due to this tradeoff.

For those interested in implementation details, the source code for the auto exposure calculation is provided in the Source SDK.

7.6 Color Correction

Another important rendering feature, which we first shipped in the game *Day of Defeat: Source*, is real-time color correction. Human emotional response to color has been exploited in visual art for millennia. In the film world, the process of *color correction* or *color grading* has long been used to intentionally emphasize specific emotions. As it turns out, this technique, which is fundamentally a process of remapping colors in an image, can be very efficiently implemented on modern graphics hardware as a texture-based lookup operation. That is, given an input image, we can take the *r*, *g*, *b* color at each pixel and re-map it to some other color using an arbitrary function expressed as a volume texture. During this process, each pixel is processed independently, with no knowledge of neighboring pixels. Of course, image processing operations such as blurring or sharpening, which *do* employ knowledge of neighboring pixels, can be applied before or after color correction very naturally.

As mentioned in the preceding section on HDR rendering, we are already performing some image processing operations on our rendered scene. Including a color correction step fits very easily into the existing post-processing framework and decouples color correction from the rest of the art pipeline. We have found color correction to be useful for a variety of purposes such as stylization or to enhance a particular emotion in response to gameplay. For example, after a player is killed in our World War II themed game *Day of Defeat: Source*, the player can essentially look over the shoulder of any of his living team-mates for a short period before he is allowed to re-enter play. During this time, we use color correction and a simple overlaid film grain layer to give the rendering the desaturated look of WWII era archival footage.

Of course, color correction in real-time games can go beyond what is possible in film because it is possible to use game state such as player health, proximity to a goal or any other factor as an input to the color correction process. In general, the dynamic nature of games is both a curse and a blessing, as it is more difficult to tweak the look of specific scenes but there is far more potential to exploit games' dynamic nature due to the strong feedback loop with the player. We think of color correction as one of many sideband channels that can be used to communicate information or emotion to a player and we believe we are only just beginning to tap into the potential in this area.

Original Shot *Day-for-Night Color Corrected shot*

Original Shot *Desaturated with Color Correction*

***Figure 11**. Real-Time Color Correction in the Source Engine*

Beyond its use as an emotional indicator, color correction is a powerful tool for art direction. The amount of content required to produce a compelling game environment is rising rapidly with GPU capabilities and customer expectations. At the same time, game development team sizes are growing, making it impractical to make sweeping changes to source artwork during the later stages of production. With the use of color correction, an Art Director is able to apply his vision to the artwork late in the process, rather than filtering ideas through the entire content team. This is not only an enormous effort-multiplier, but it is invariably true that a game development team knows more about the intended player emotional response late in the process. Having the power to make art changes as late as possible is very valuable. If an art team authors to a fairly neutral standard, an art director is able to apply game-specific, level-specific or even dynamic event-specific looks using color correction. Another less obvious benefit which is particularly important to a company like Valve which licenses it engine and actively fosters a thriving *mod* community is the power that color correction provides for mod authors or Source engine licensees to inexpensively distinguish the look of their titles from other Source engine games.

For our games, a level designer or art director authors color correction effects directly within the Source engine. A given color correction function can be built up from a series

of familiar layered operators such as curves, levels and color balance or can be imported from other popular software such as Photoshop, After Effects, Shake etc. Once the author is satisfied with their color correction operator, the effect can be baked into a volume texture that describes the color correction operation and is saved to disk. This color correction operator can then be attached to a node within our game world using our level design tool, *Hammer*. It is possible to attach a color correction function to a variety of nodes such as trigger nodes or distance based nodes, where the effect of the color correction operator decreases with distance from a given point. Essentially any game logic can drive color correction and several color correction operators can be active at once, with smooth blending between them.

7.7 Acknowledgements

Many thanks to James Grieve, Brian Jacobson, Ken Birdwell, Bay Raitt and the rest of the folks at Valve who contributed to this material and the engineering behind it. Thanks also to Paul Debevec for the light probes used in Figure 5.

7.8 Bibliography

DEBEVEC, P. 2000. Personal Communication

DIEFENBACH, P. J. 1997. Multi-pass Pipeline Rendering: Realism For Dynamic Environments. Proceedings, 1997 Symposium on Interactive 3D Graphics.

FRANKE, S. 2006. Personal Communication

GREEN, C., AND MCTAGGART, G. 2006. High Performance HDR Rendering on DX9-Class Hardware. Poster presented at the *ACM Symposium on Interactive 3D Graphics and Games*.

GREGER, G., SHIRLEY, P., HUBBARD, P. M, AND GREENBERG, D. P. 1998. The Irradiance Volume. IEEE Computer Graphics & Applications, 18(2):32-43,

HOOK, B. 1999. The Quake 3 Arena Rendering Architecture. Game Developer's Conference

LEHTINEN, J. 2006. Personal Communication

MCTAGGART, G. 2004. Half-Life 2 Shading, GDC Direct3D Tutorial

RAMAMOORTHI, R. AND HANRAHAN, P. 2001. An Efficient Representation for Irradiance Environment Maps. SIGGRAPH 2001, pp. 497-500.