Chapter 5

Production-Ready Global Illumination

Hayden Landis,
Industrial Light + Magic

5.1 Introduction

Global illumination can provide great visual benefits, but we're not always willing to pay the price. In production we often have constraints of time and resources that can make traditional global illumination approaches unrealistic. "Reflection Occlusion" and "Ambient Environments" have provided us with several reasonably efficient and flexible methods for achieving a similar look but with a minimum of expense.

This chapter will cover how Industrial Light and Magic (ILM) uses Reflection Occlusion and Ambient Environment techniques to integrate aspects of global illumination into our standard RenderMan pipeline. Both techniques use a ray-traced occlusion pass that is independent of the final lighting. The information they contain is passed on to our RenderMan shaders where they are used in the final lighting calculations. This allows lighting and materials too be altered and re-rendered without having to recalculate the occlusion passes themselves.

We will show that when used together these two techniques have given us an integrated solution for realistically lighting scenes. They allow us to decrease setup time, lighting complexity, and computational expense while at the same time increasing the overall visual impact of our images.

5.2 Environment Maps

Reflection Occlusion and Ambient Environments are most effective when used with an accurate environment map. We will also explain how standard light sources benefit from using these techniques, but environment maps still remain the easiest and most accurate way of lighting with either technique. So let's take a moment to discuss how we go about creating these environments.

Whenever possible we have traditionally shot a reflective "chrome" sphere and a diffuse "gray" sphere on location as lighting reference (see Figure 5.1). They provide a way of calibrating the CG lighting back at ILM with the lighting environment that existed on location.

Figure 5.1: Chrome sphere, gray sphere, and Randy Jonsson on location in Hawaii.

While the gray sphere is most often used simply as visual reference, the chrome sphere can be applied directly to our CG scene as a representation of the surrounding environment. As shown in Figure5.2, we take the image of the chrome sphere and unwrap it into a spherical environment map. This allows us to access the maps with a standard RenderMan environment call. At times unwanted elements, like the camera crew for example, can be found hanging out in the reflection. These are easily removed with a little paint work (see Figure 5.2, right).

Figure 5.2: Unwrapped chrome sphere environment texture and final painted version.

More than just a reference for reflections, the chrome sphere and it's resulting environment map give us a reasonably complete representation of incoming light on the location it was shot. If no chrome sphere exists then it is up to the artist to construct their own environment maps from the background plate or other photographed reference. It is also possible to use running footage in environment maps to give you interactive lighting based on events taking place in the shot.

HDR Images

I'm sure someone out there is asking: "What about High Dynamic Range Images?" There is no reason we can't use HDR images with either of these techniques. However, in practice we are usually lucky to get just a single chrome sphere image from location let alone a series of calibrated exposures. I'm sure that sometime in the future we will start to use HDR images more often but they are not always necessary. We have found that the single chrome sphere image can work just fine in many cases.

There is the issue of how to represent very bright areas and highlights in these environments. Once you adjust reflection levels properly for an environment you often notice that you've lost the brightest highlights and reflective areas of the environment. They become dull and washed out. We have come up with a useful trick, shown in Listing 5.1, that allows us to retain these intense reflection highlights. This works by taking the brightest parts of the image and expanding them to a set intensity. While not as necessary for Ambient Environments, this technique comes in very handy for reflection environments.

Listing 5.1 Example of expanding select ranges of an environment map.

```
float expandDynRange = 1.00; /* Expand specified range of the map to this max value.*/
float dynRangeStartLum = 0.50; /* Starting luminance for expansion to begin. */
float dynRangeExponent = 2.00; /* Exponent for falloff */
color Cenv = color environment (envMap, R, "blur", envBlur, "filter", "gaussian");
if (expandDynRange > 1) {
    /* Luminance calculation, 0.3*Red + 0.59*Green + 0.11*Blue.*/
    float lum = 0.3*comp(Cenv,0)+0.59*comp(Cenv,1)+0.11*comp(Cenv,2);
    if (lum > dynRangeStartLum) {
        /* remap lum values 0 - 1*/
        lum = (lum - dynRangeStartLum)/(1.0 - dynRangeStartLum);
        float dynMix = pow(lum,dynRangeExponent);
        Cenv = mix(Cenv, Cenv*expandDynRange, dynMix);
    }
}
```

5.3 Reflection Occlusion

First developed during *Speed II* and enlisted into full time service on *Star Wars: Episode 1*, Reflection Occlusion has become an important tool for creating realistic looking reflections.

When you use an all encompassing reflection environment you have the problem of occluding inappropriate reflections. Single channel Reflection Occlusion maps, like those shown in Figure 5.3, allow us to attenuate reflections in areas that are either self occluding or blocked by other objects in the scene. As illustrated in Figure 5.4, our surface shaders read these occlusion maps and then attenuate the environment to provide us with more realistic reflections.

Figure 5.3: Example of Reflection Occlusion passes: B25 and Spinosaurus.

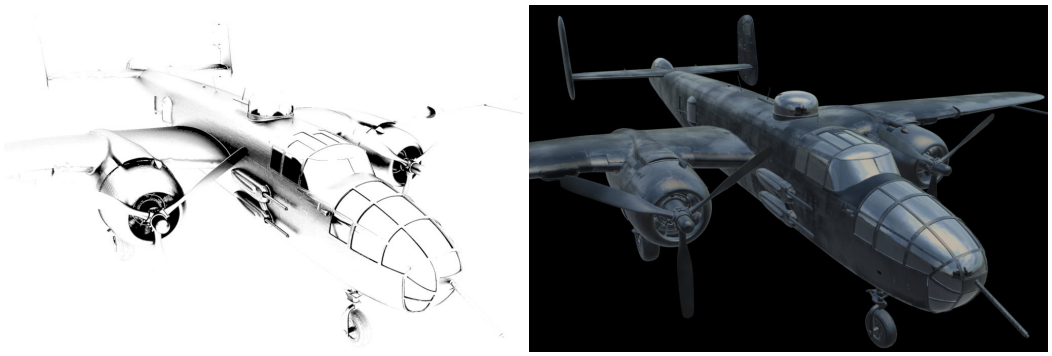

Figure 5.4: B25 rendered with reflections (left), and with the addition of reflection occlusion (right). ©Lucas Digital Ltd. LLC. All rights reserved.

Reflection Blur

Reflection blur is another important component of Reflection Occlusion and is used to help simulate various surface textures in the model. It is achieved by jittering the secondary rays around the main reflection vector. Mirror surfaces get little or no blur while matte surfaces receive a more diffused occlusion. As shown in Figure 5.5, glass surfaces, receive an almost mirror reflection while a rubber surface has a very diffused occlusion. For a proper occlusion, transparent surfaces should be made visible to primary rays but not secondary rays.

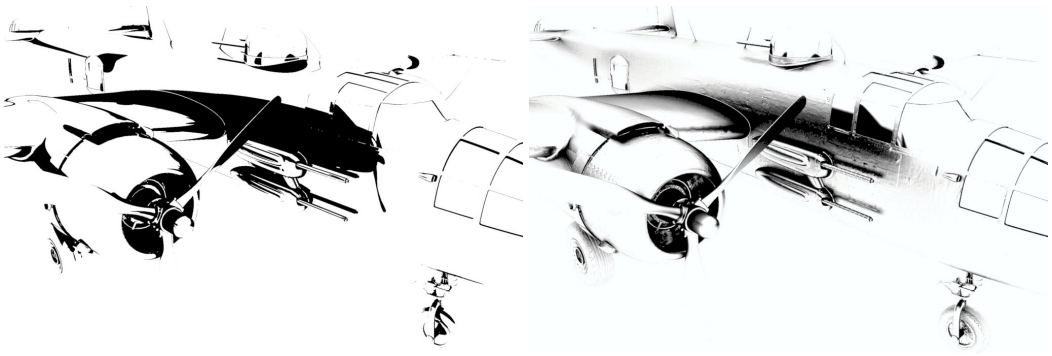

Figure 5.5: The image to the left shows a perfectly mirrored reflection occlusion. The image on the right shows differing blur amounts for reflective glass areas, diffuse paint, and more diffuse rubber tires. ©Lucas Digital Ltd. LLC. All rights reserved.

Reflection occlusion gives us some of the advantages of doing a full ray-traced reflection pass without all of the expense. As long as our animation doesn't change we can keep reusing the same occlusion pass for subsequent iterations of the final render. It allows us the convenience of using standard RenderMan environments and reflections but gives them the illusion of a more complex ray-traced scene. For reflective objects this solution allows us to bypass the expense and hassle of a full ray-traced render unless it's absolutely necessary. An example reflection occlusion shader is shown in Listing5.2.

Listing 5.2 `reflOccl.sl`: Example of an Entropy shader that produces a reflection occlusion image.

```
#include "entropy.h"

surface reflOccl_srf (float rflBlurPercent = 0.00;
                      float rflBlurSamples = 4.00;)
{
    Oi = Os;
    if (raylevel() > 0) {
        Ci = Oi;
    } else {
        float rflBlur = rflBlurPercent/100;
        normal NN = normalize(N);
        vector IN = normalize(I);
        vector R = reflect(IN,NN);
        float occ = environment ("reflection", R, "blur", rflBlur,
                                 "samples", rflBlurSamples);
        Ci = (1-occ)*Oi;
    }
}
```

5.4 Ambient Environments

Lighting with fill lights involves guesswork and can take up a great deal of time if done correctly. Ambient Environments is a technique that was developed to try and free us from the necessity of wrangling lots of fill lights.

There are two components to an Ambient Environment. “Ambient Environment Lights” provide illumination while “Ambient Occlusion” provides shadowing and important directional information for looking up into the Ambient Environment map. Similar to Reflection Occlusion, Ambient Environments also use a pre-rendered occlusion map accessed at render time to give our scene realistic shadowing. We can conveniently use the same environment map for both our ambient environment and our reflection environment.

Traditionally ambient lights have never been too popular in production because they simply add a single overall color, a less than spectacular lighting effect. By naming this technique “Ambient Environments,” we hope to help restore the good name of the much maligned ambient light.

Developed initially during *Pearl Harbor* this technique quickly spread to other shows and has since become an important lighting tool for most productions at ILM.

5.4.1 Ambient environments defined

ambient (’am-b{e-}-*nt)
Etymology: L i[ambient-], i[ambiens], prp of i[ambire] to go around, fr. i[ambi-] + i[ire] to go – more at ISSUE aj, surrounding on all sides: ENCOMPASSING

An “ambient environment” represents the diffuse fill light that surrounds an object. It’s not intended to represent direct light sources. These are left to standard RenderMan lights. The Ambient Environment’s job is to give us indirect “bounce” or “fill” light from the environment. Rather than setting up multiple fill lights and guessing their color, intensity and direction, an ambient environment light provides all of this relatively free. It’s also necessary to provide shadowing from the surrounding lighting environment. Points not fully exposed to the environment need to be attenuated properly. This process is known as “Ambient Occlusion.” An example of the Ambient Environment process is shown in Figure 5.6.

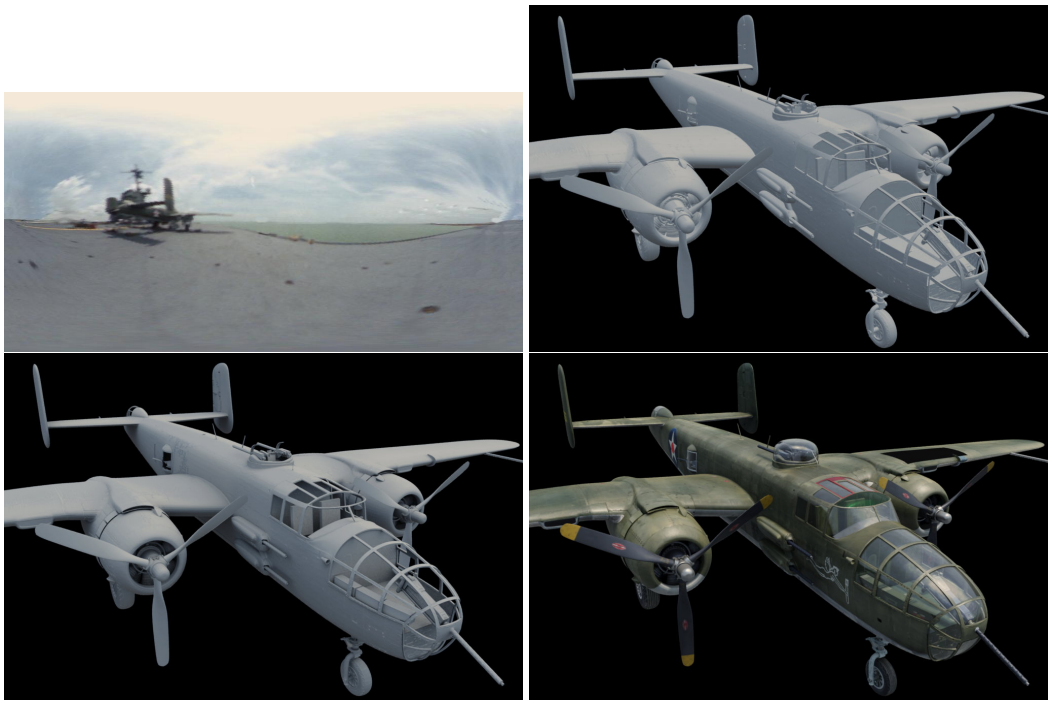

Figure 5.6: Simple example of Ambient Environment process. Top: environment map and plastic B25 illuminated with only an Ambient Environment Light. Bottom left: Ambient Environment Light with occlusion and ”bent normals”. Bottom right: The B25’s final beauty render.

5.4.2 Why ambient environments?

There are several advantages of using this method over traditional fill lighting techniques.

- Using a chrome sphere gives you a more accurate representation of the environment than placing fill lights by hand. There is little guess work involved and you get the exact environment as reflected in the chrome sphere.
- The light completely surrounds an object. No dark patches or areas of missing illumination.
- It is very efficient to set up and adjust - one light, one map.
- Fast! One ambient environment light replaces 3 or more fill lights. There are no multiple shadow passes to render, only a single Ambient Occlusion pass is required. You save the time it takes to compute the additional lights and shadow passes.
- View independent. If the environment’s orientation changes there is no need to re-render the occlusion pass. If “baked” occlusion maps exist for a model, no shadow or occlusion renders are necessary except for your key light and any other direct ”hard shadowed” light sources. Baked maps also free you from any dependence on camera or object orientation.

5.4.3 Ambient environment lights

An Ambient Environment Light is simply a modified environment reflection. Rather than using the reflection vector

```
R = reflect(IN,NN);
```

that we normaly use with an environment map, an Ambient Environment uses the surface normal

```
R = normalize( faceforward(NN,I) );
```

which is the direction of the greatest diffuse contribution, to gather illumination from the environment. This is illustrated by the top two images of Figure 5.7.

Since the RenderMan environment call conveniently blurs across texture seams, we can apply a large map blur value (25%-30%) rather than sampling the environment multiple times as you might do with a ray-traced approach. The blurred lookup into the environment map represents the diffuse contribution of the environment for any point on the surface (see bottom image, Figure 5.7). This has the speed advantage of sampling our environment only once rather than multiple times.

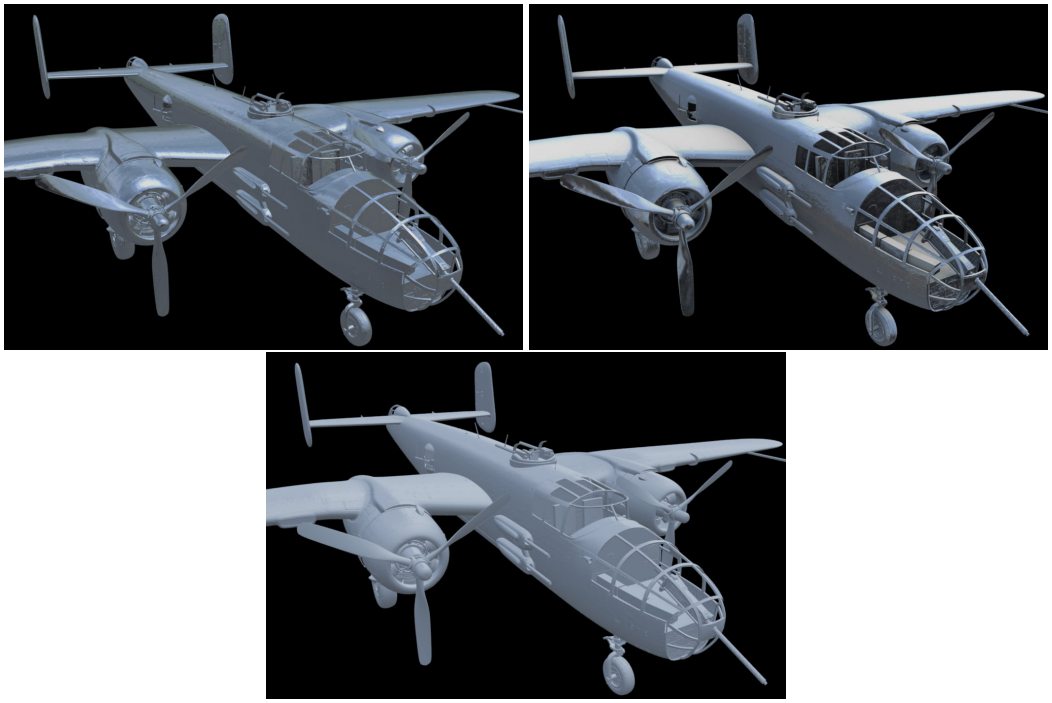

Figure 5.7: Top Left: Regular reflection environment lookup. Top right: Environment lookup using surface normal. Bottom: Environment lookup using surface normal with 30% blur.

5.4.4 Ambient occlusion

Ambient occlusion is a crucial element in creating a realistic ambient environment. It provides the soft shadowing that we have come to expect from global illumination and other more complex indirect lighting techniques. Surfaces not fully exposed to the environment need to be attenuated properly so that they do not receive the full contribution of the ambient environment light. This is one of the main attractions of using the Ambient Environment technique. In Figure 5.8 you can begin to see some of the subtle visual cues that will eventually help convince us that the lighting is real.

Figure 5.8: Example Ambient Occlusion images. B25 bomber, Spinosaurus and Tyrannosaurus. ©Lucas Digital Ltd. LLC. All rights reserved.

In order to get this effect, it is necessary to have an ambient occlusion render or "baked" ambient occlusion maps that represent this attenuation of outside light. Ambient occlusion is achieved through the following process: For every surface point, rays are cast in a hemisphere around the surface normal. The final occlusion amount is dependent on the number of rays that hit other surfaces or objects in the scene.

Figure 5.9: Simple illustration of surface sending out rays, some of which are blocked. Perhaps from the B25 fuselage under the wing. Showing blocked rays from wing and engine nacelle. ©Lucas Digital Ltd. LLC. All rights reserved.

Since the strongest diffuse contribution comes from the general direction of the surface normal, the result is weighted to favor samples that are cast in that direction. If there is an object directly parallel to the surface it will be occluded more than if the same object were placed to the side. Transparent or glass materials should be excluded from the Ambient Occlusion render. If you have opacity maps you want to make sure that your ambient occlusion shader takes this into account. This pass can be rendered each frame for objects with internal animation. For solid objects with few moving parts it can be rendered once and baked into texture maps. Baking the occlusion maps gives you a huge advantage since they only need to be rendered once per object. This works because unlike Reflection Occlusion, Ambient Occlusion is not dependent on orientation of the object or environment. You can share the same maps amongst multiple instances of an object and in any scene.

Bent normals

Another important component of Ambient Occlusion is the addition of an "average light direction vector." This represents the average direction of the available light arriving at any point on the surface. The unoccluded rays from the initial occlusion calculation are averaged together to find the difference between this "average light direction vector" and the original surface normal. This offset is stored in the G, B and A channels of the Ambient Occlusion map (see Figure 5.10, right). This vector is used to redirect the lookup into the ambient environment so that the color is gathered from the appropriate direction. The surface normal originally used to lookup into the environment texture will now be bent at render time to point in this new direction (see Figure 5.11). We use the term "bent normals" to refer to this effect since it is difficult to say "average light direction vector" ten times fast.

These two components of the Ambient Occlusion render combine to give us realistic shadowing as well as an accurate lookup into the Ambient Environment texture.

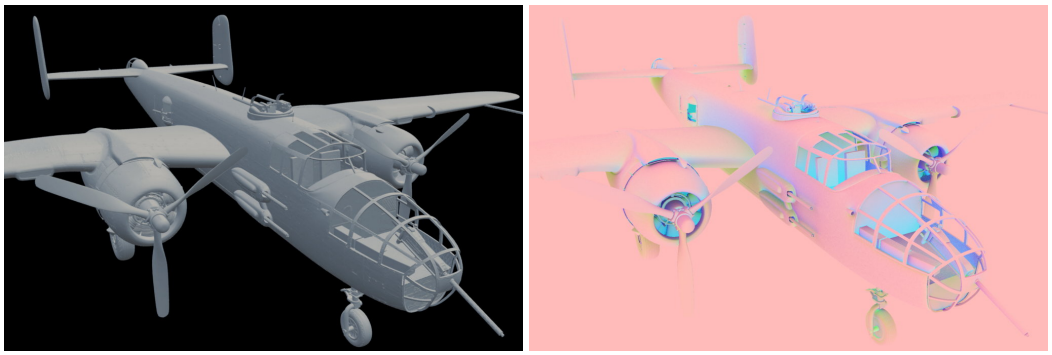

Figure 5.10: Raw materials: Ambient environment light render and an Ambient Occlusion render, representing the stored occlusion and bent normal data. ©Lucas Digital Ltd. LLC. All rights reserved.

Figure 5.11: Final product: Plastic render of the B25 with an occluded ambient environment light. The image on the right shows the final step of integrating the “bent normals.” ©Lucas Digital Ltd. LLC. All rights reserved.

Listing 5.3 is an example of an Ambient Occlusion shader. Listing shad:hayden:bendnorms contains a pseudocode example of how we convert the bent normals stored in the Ambient Occlusion maps back into a normal that the surface shader can use.

5.4.5 Other Ambient Environment light types

We can use the occlusion and directional information contained in the Ambient Occlusion map and apply it to other light sources as well. If you take a standard point or spot light, pass it the ”bent normal” rather than the original surface normal and then attenuate it with the occlusion channel, you will get a nice soft fill light source with no additional shadowing necessary (see Figure 5.12). This makes it fairly cheap to add lights to an object already using Ambient Occlusion. These additional lights allow you to add or subtract light from the base ambient environment.

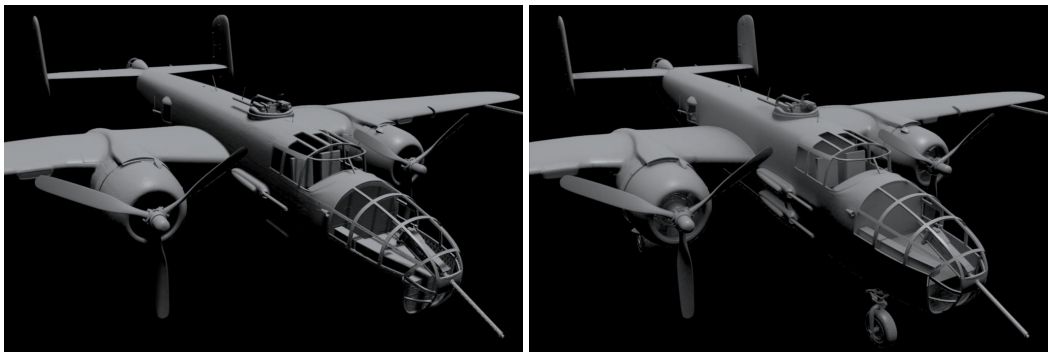

Figure 5.12: Example of a spot light using standard shadows and a spot light using occlusion and bent normals for shadowing.

5.4.6 Other uses for Ambient Occlusion

We've found several other useful applications for ambient occlusion. One of these is for creating contact shadows (see Figure 5.13). Hiding shadow casting objects from primary rays but not secondary rays allows us to create contact shadows for objects that can then be applied in the render or composite.

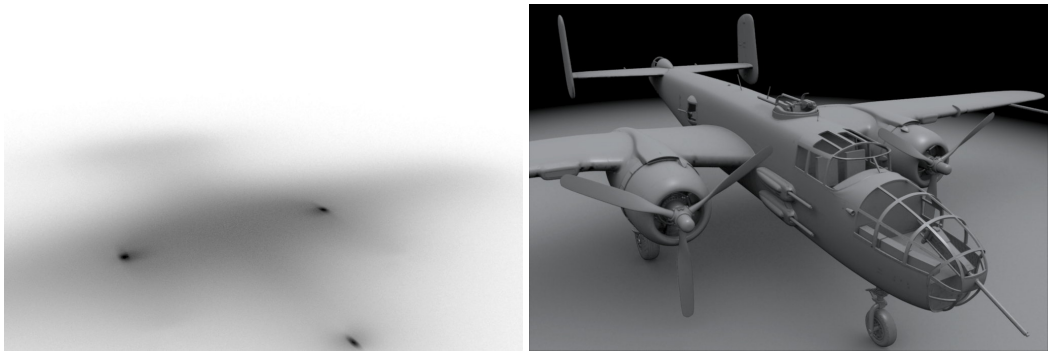

Figure 5.13: Example of Ambient Occlusion contact shadow.

5.5 Application

Combining Reflection Occlusion and Ambient Environments has allowed us to realistically light complex scenes with a minimum of effort. The lighting of many final scenes has been accomplished by using only three lights: Key, reflection, and Ambient Environment lights.

Figure 5.14 shows an example from *Pearl Harbor*. This shot required us to place 14 computer generated B25 bombers next to four real B25 bombers on the deck of an aircraft carrier in Texas. Then we had to place this landlocked carrier it in the middle of the Pacific Ocean.

Figure 5.14: From *Pearl Harbor*, two frames from the establishing shot of Doolittle's raid. ©Lucas Digital Ltd. LLC. All rights reserved.

One key to the success of this shot was the development of materials that were created in a calibrated lighting environment. This environment was built using the gray and chrome sphere references as well as photos of the real B25 bombers. If this is done correctly you can drop the model and its' materials into any other environment and soon have it looking right at home.

Armed with an environment map, reflection occlusion pass, baked ambient occlusion maps and a good set of materials, it took only part of a day to tweak the final lighting for this shot. Only three lights: Key, reflection environment, and ambient environment were used. Not every shot goes this smoothly but it is a testament to the ease of using this simple but effective lighting setup.

On *Jurassic Park III* we had the task of creating realistic dinosaurs and used Ambient Environments to create several interesting effects in addition to their regular lighting tasks. By adding running footage of flames to an environment, a realistic and interactive lighting effect was created for the Spinosaurus in this shot (see Figure 5.16).

5.6 Conclusion

I am sure that at some point we will be ray tracing complex scenes on our palm pilots. Until then we'll continue to create efficient cheats and tricks to get the visual advantages of global illumination

Listing 5.3 `occlusion.sl`: Entropy Ambient Occlusion example shader.

```
#define BIG 1e20

color vector2color(vector v) {
    return ((color v) + 1) / 2;
}


surface occlusion (float samples = 16;
                   float doBendNormal = 0;)
{
    normal NN = normalize(N);
    vector up = vector(0,1,0);
    float sum = 0;
    float i;
    vector dirsum = 0;
    vector side = up ^ NN;
    for (i = 0; i<samples; i = i+1) {
        float phi = random()*2*PI;
        float theta = acos(sqrt(random()));
        vector dir = rotate(NN,theta,point(0),side);
        dir = rotate(dir,phi,point(0),NN);
        point Ph;
        normal Nh;
        if (rayhittest(P,dir,Ph,Nh) > BIG) {
            sum = sum + 1;
            dirsum = dirsum + dir;
        }
    }
    sum /= samples;
    dirsum = normalize(dirsum);
    if (doBendNormal != 0) {
        vector bend = dirsum - NN;
        Ci = vector2color(bend);
    } else {
        Ci = sum;
    }
}
```

Figure 5.15: Frames of the B25 in its look development environment composited over reference photos of the real B25.

Figure 5.16: We used running footage from this background in our ambient environment map to create interactive fire light on the Spinosaurus. ©Lucas Digital Ltd. LLC. All rights reserved.

without the time and expense.

We continue to expand on the basic concepts of Ambient Environments. Some colleagues at ILM have already been busy adding features that allow for self illumination and other more advanced lighting effects. At some point I am sure this will eventually migrate to full blown global illumination.

For some shots it is true that you can get away with only the minimum of these lighting tools alone. However, in production we unfortunately know that "real" is never quite good enough. For other shots these techniques are not the "end all, be all" but a solid foundation on which to build your final lighting. We hope you find these techniques useful and as much fun to work with as we have had in developing them.

Acknowledgements

I would like to thank the following people:

Dan Goldman and Christophe Hery for their assistance in preparing this chapter; Thanks to Dan as well for the Entropy shader examples; Dawn Yamada and the ILM PR department; Simon Cheung for the great B25 model and Tony Sommers for his wonderfull paint work.

Reflection Occlusion has a long list of contributors. It was originally developed by Stefen Fangmeier on *Speed II*. Reflection blur implemented by Barry Armour. Further work by Dan Goldman on *Star Wars: Episode I* provided speed and quality optimizations. Other contributors include: Ken McGaugh and Will Anielewicz.

Ambient Environments would never have gotten off the ground without the hard work and inspiration of Ken McGaugh and Hilmar Koch. Thanks guys! Further refinements have been contributed by Kevin Sprout, Dan Goldman, and Doug Sutton.

Thanks to everyone at ILM for your hard work and dedication. Without your imagery I'd be using tea pots and mandrills. Not that there is anything wrong with that...

Listing 5.4 This is an example of how we convert the information stored in the Ambient Occlusion images back into occlusion and ”bent normal” data that can be used by the surface shader.

```
uniform float ambOcclDensity; // Amount of the original occlusion you want
                              // to mix in (usually 1)
vector IN = normalize(I);
normal NN = normalize(N);
normal bentNormal = normalize( ntransform("object", faceforward(NN,IN)) );

color ambBendNormColor = 0;
Pndc = transform("NDC",P);
ambS = xcomp(Pndc);
ambT = ycomp(Pndc);

ambOccAmt = float texture(ambOcclMap[0], ambS, ambT);
ambOccAmt = mix(1, ambOccAmt, ambOcclDensity);

ambBendNormColor = color texture(ambOcclMap[1], ambS, ambT);
/*This line actually reads the "bent normal" channels 1-3 of the map,
  not just channel 1 as it might appear. */

normal addNormal = -1 + 2 * (normal ambBendNormColor);
bentNormal = normalize(bentNormal + addNormal);

R = vtransform("object",environmentSpace,bentNormal);
R = normalize(R);
Cenv = color environment (envMap, R, "blur", blurAmount, "filter", "gaussian");


/* Note: We have come across a problem with the way RenderMan blurs
 * across texture seams.  Sometimes this shows up as a pixel wide line
 * that appears in the results of your environment() calls. If you do run
 * into this problem the following is a workaround.
 */
R = vtransform("object",environmentSpace,R);
vector R1 = R + Du(R)*du;
vector R2 = R + Dv(R)*dv;
vector R3 = R1 + R2 - R;
color Cenv = color environment (envMap, R,R1,R2,R3, "blur", blurAmount,
                                "filter", "gaussian");
```