Soft shadows

Steve Marschner
Cornell University
CS 569 Spring 2008, 21 February

Soft shadows are what we normally see in the real world. If you are near a bare halogen bulb, a stage spotlight, or other artificial point source you may see hard-edged shadows, but even the sun (or the moon, which has the same apparent size) is large enough that the shadows it casts are noticeably blurry. The perfect, hard-edged shadows that are the easiest to render in graphics systems using point sources look artificial and are often not what you really want.

If you think about it, soft shadows contain a lot less information than hard ones: they are blurry and indistinct, and details about the shape of the shadow caster are lost. And yet they have traditionally been much more expensive to compute than hard shadows. One thread of research over the last 7 or 8 years has been to take advantage of the smooth, low-frequency nature of many illumination effects to make them efficient to compute. We'll only manage to look at the most basic instances in this class.

There are *many* methods in the research literature for rendering soft shadows with all manner of variants on hard-shadow methods. Very few are general and robust. I'm going to cover some that are reasonably principled and/or have seen broad adoption.

Accumulating shadow textures

One simple approach to soft shadows is to approximate the area light with constellation of point lights scattered uniformly over the area light's surface. This could be done randomly or regularly, though if you do it randomly you probably want to choose one random set of points and keep it fixed. The result of this amounts to approximating a soft shadow as the sum of a finite number of hard shadows.

If we're rendering shadows into a shadow texture for a planar surface, using the methods from the previous lecture (i.e. loading a projection that maps primitives from 3D space onto the texture space of the receiving surface, clearing the texture to white, and then drawing the whole scene in black), one can add up the effects of a constellation of point sources using an *accumulation buffer*. An accumulation buffer is a frame buffer (usually with a bit higher precision than a normal color buffer) into which you render many images with the pixel blending mode set to "add." In this way one can compute the sum of a whole bunch of images.

So the rendering process looks like this:

1. Set up two buffers: a shadow texture and an accumulation buffer of the same resolution.
2. Clear the accumulation buffer to zero.
3. For each light source:
    a. Load up the projection matrix that projects from the source into texture space.

b. Clear the shadow texture to white and draw the whole scene in black.

c. Add the shadow buffer into the accumulation buffer.

4. Divide the accumulation buffer by the number of lights.

The accumulation buffer then holds the soft shadow texture, which you can use to render the receiving surface.

This approach, in the limit of many light sources, produces very accurate soft shadows. The trouble with it is that it's quite expensive. The fact that we're adding up a finite number of hard shadows is obvious unless we use enough light sources that the hard shadow edges are close enough together, and have low enough contrast, that we don't notice them any more. This requires a lot of lights, especially for large sources. Heckbert & Herf's results use 30 to 50 points, which costs 30 to 50 times as much as a simple hard shadow computed using the same method.

As presented by H&H, this method only works for planar receivers, because it doesn't have any way to keep track of what is in front of what other than the clipping plane that prevents things behind the receiver from shadowing it. How could we upgrade it to work with shadow maps? (We can't add up the shadow maps!)

This method also only works for diffuse surfaces, where we can compute the total illumination first, without regard to where it came from, and then compute the shading afterwards. How could we upgrade it to work with arbitrary surface shading?

(The answers I have in mind lead toward the idea that the only really general version of this is to render a bunch of complete images, using a shadow pass and a shading pass for each one, and to accumulate those finished images. This is exactly the same as just putting a large number of point lights in the scene.)

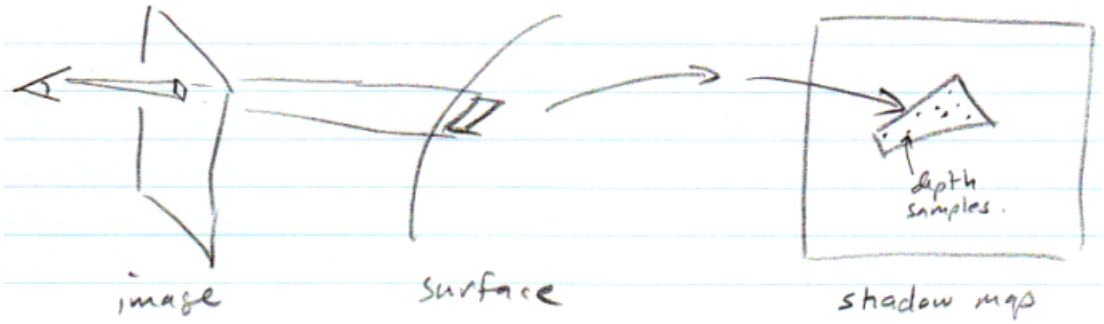
Percentage closer filtering

One of the nice features of shadow textures is that you can use image operations on them. You can make a cheap (if not too accurate) approximation of a soft shadow by blurring the shadow texture. You can also filter them and mip-map them like any other texture (at least to the extent that their effect on the shading is linear, which it normally is). For a projective shadow texture that's being used for arbitrary surfaces, we've seen that the resulting hard-to-control changes in magnification can lead to severe aliasing artifacts that are large and obvious in the rendered image. Being able to do a nice job of filtering is very useful here: it can change the aliasing into blurring, which usually looks like a soft shadow and doesn't bother anyone.

However, filtering a shadow map lookup is not so easy, because the answer is not linear with respect to the numbers stored in the map. What will happen if you sample a shadow map with normal texture filtering turned on?

To get the right answer for a filtered shadow map lookup, you have to do the lookups first, then filter. That is, rather than taking an average depth value and comparing it with your depth, you have to take a number of depth values and find what fraction (percentage) of them are closer to the light than you. This operation is known as "percentage closer filtering."

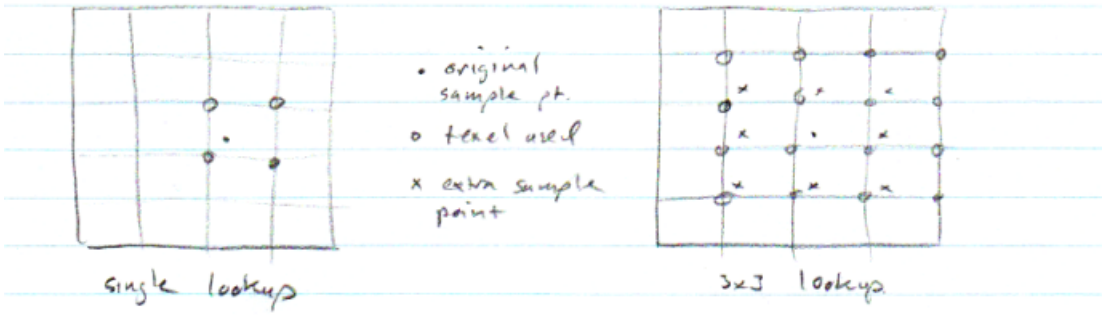
It was originally proposed (by Reeves et al 87) in the context of the Renderman system, a software graphics pipeline. Their approach is to map the footprint of the area of surface being shaded (in our context, the pixel footprint) into the shadow map, then take several random samples in that footprint:



In practice they actually approximate the projected footprint by its bounding box (à la mipmaps) because in shadows blurring is well tolerated. If the bounding box is large you may need a fair number of samples to get a good estimate. Of course, to prevent aliasing in the magnification case, you don't let the sampled area get smaller than a texel or so in the shadow map.

You can simulate soft shadows just by artificially enlarging the footprint area.

For interactive applications this random sampling approach is generally too involved. Instead, we just use the four nearest texels as the sample points, using bilinear interpolation to average them. In fact, modern hardware already does this automatically for shadow map lookups. Still, bilinear interpolation is not nearly smooth enough to filter sharp shadow boundaries, so one can average together a number of shadow samples over a fixed area in the map to provide the desired amount of blurring.



In the slides you can see the result of doing this with 16 samples.

Environment illumination

For illumination from an environment map, you can think of the map as a giant extended light source and render accumulated shadow textures (or accumulate many two-pass shadow map renderings) for many light sources sampled all over the map. This is a reasonable approach that can produce very good results, but it requires a lot of light sources.

How do we generate lights to approximate an environment map? Clearly we want a set of directional (infinitely distant) lights. It was easy to generate some lights on a polygonal source by positioning them randomly and giving them the same intensity.

One approach to the env. map problem would be to split each face of a cubemap, say, into a number of rectangles, and generate a source in each rectangle with the appropriate brightness to represent the total light coming from that section of the map. (Note that to be really correct you have to watch out for the varying solid angles of the different areas.) This will work all right, but because it can produce lights with very different intensity, you may be able to see hard shadow edges cast by the brightest lights even if you are using lots of sources.

A better way to do this is to split the environment up into a number of patches, each of which has roughly the same total brightness, and make one light for each. This will cluster the lights around bright areas of the map, which is what you want.

Key to doing this well is to use a method that produces a set of samples of roughly similar intensity, with more of them in brighter areas of the map. This is known as "importance sampling" and is a little tricky to do for an image, but there are several methods available (which I won't get into). Note that this problem is closely related to halftoning—generating a pattern of dots that, when printed by an inkjet printer, for example, will average out to the right image.

Ambient occlusion

The most extreme case of a large emitter is light that comes uniformly from the whole environment. Under this kind of lighting, the direction of the surface normal does not affect the shading at all, so the traditional shading cues are missing. However, there is still shading, and it provides useful cues to surface shape. The shading results from different amounts of light being able to reach different parts of the surface. As usual, think of an ant standing on the surface; if it is in a convex area, it's illuminated by the entire sky, but if it's at the bottom of a depression, the surface blocks part of the illumination. The idea of "ambient occlusion" is to store a texture with this information in it: for a given surface point the texture value tells you what fraction of the sky is visible.

If you're looking at a diffuse surface in a perfectly uniform environment, then the ambient occlusion map tells you what you need to know. But your surface might not be diffuse, and the only place you're likely to encounter a perfectly uniform radiance field (outside the laboratory) is in the interior of a cloud. So we still need to account for surface properties and varying illumination.

A popular approach is to use an irradiance environment map (see the reflections lecture). This map will tell you what the unoccluded diffuse illumination would be for a point with a given surface normal. You can then attenuate that by the occlusion value, resulting in shading that depends both on the environment illumination and on local shadowing. Don't forget that this is *not* the right answer, though—it assumes that occlusion and illumination are uncorrelated.

A clever hack to improve this a little bit is *bent normals*. The idea there is to keep track of the average unoccluded direction, and then look up your diffuse shading as if that was the surface normal. That way you'll get shading on the right side of surface relief.

How to compute occlusion maps? Use ray tracing, or lots of shadow maps.

Note that disregarding indirect illumination will cause ambient occlusion to underestimate the illumination in the dark areas (thereby overestimating the contrast) for light-colored surfaces. Actually getting this right is an involved computation, but this could be helped by assuming that there's some constant illumination coming from the visible surface.