Simpler Soft Shadow Mapping

Lee Salzman

September 20, 2007

Lightmaps, as do other precomputed lighting methods, provide an efficient and pleasing solution for lighting and shadowing of relatively static scenes. Most of the work of lighting computations can be offloaded to a precomputation step and only leave simple texture lookups to be performed at the time of rendering. As one makes the technological leap to dynamic lighting and shadowing, one is confronted by methods, such as stencil shadow volumes and shadow mapping, which at first appear simple, yet become increasingly complex and expensive as one tries to recover the quality of appearance provided by simple lightmapping. One can either use these methods uniformly throughout a scene to give the lighting a uniform appearance, but at much greater expense than precomputed lighting, or integrate precomputed lighting with a dynamic shadowing method with disparate appearance – such as hard-edged or blocky, aliased dynamic shadows over soft, static lighting. Ideally, one needs dynamic shadowing tools that provide a pleasing level of consistency when integrated with precomputed lighting without the leap in cost of applying dynamic lighting and shadowing to everything in the scene.

Background and Motivation: Variance Shadow Mapping

Variance shadow mapping [1] is a recent shadow mapping variant for efficiently approximating the soft-edged shadows caused by area lighting, helping reduce the visual quality gap with precomputed lighting. It overcomes some of the efficiency concerns of previous techniques by allowing the shadow map to be directly linearly filtered, as by a separable blur or the 3D hardware's provided bilinear and trilinear filtering, as opposed to requiring specialized filtering methods at the time of shadow map lookup. It accomplishes this by storing in a shadow map, for each depth value d of a pixel as viewed from a light source, both d and d^2. Let $E(d)$ and $E(d^2)$ correspond to each of these values after some arbitrary linear filtering applied to the shadow map, respectively. It then interprets $E(d)$ as the mean u and $E(d^2)-E(d)^2$ as the variance v. Finally, given the depth z of a point to be shadowed, if z is greater than (farther from the light) than the mean occluder depth u, then it is determined to be in shadow and then approximates a shadowing term $v/[v + (z-u)^2]$, where lower values imply more shadow and higher values more light. Otherwise, If z is less than or equal to (closer to the light) the mean u, then the pixel is assumed not occluded and to be fully lit.

Suppose one wishes to integrate this method with precomputed lighting, ideally by only rendering dynamic elements of the scene into the variance shadow map. Initially, the shadow map is cleared with depth values representing the far plane of the light source. Assuming the depth values are normalized, this value is simply 1. Then only dynamic elements of the scene are rendered into the shadow map with normalized depth values in the range of ***[0,1]***. Finally, suppose a linear filter is applied to two texels straddling the edge of a dynamic scene elements shadow in this sparse shadow map. One texel lies outside the shadow and has a depth value of ***1***, and the other lies inside with an intermediate depth value ***s***. If ***s*** is small relative to the depth range, then after linear filtering between the two texels, the resulting mean ***u*** term readily skews towards the far plane depth value of ***1***. Thus, any depth ***z*** for a point to be shadowed very close to the occluder depth ***s*** will tend to be rejected as lit by testing against this skewed mean ***u***. One can work around this problem by rendering the entire scene into the variance shadow map but at added cost.

Even ignoring this difficulty, variance shadow maps store a squared term d^2 which effectively doubles the precision requirements of the shadow map. This, for most practical purposes, compels one to use 16 or 32 bit-per-component textures to represent the shadow map, which do not necessarily have guaranteed or even efficient bilinear and trilinear filtering on available consumer 3D hardware.

Another Approach

Consider the supposed use case again. One wants to be able to linearly filter the depth values of dynamic scene elements and the far plane together to get a resulting shadowing term that can be applied to static elements of the scene and mask off their precomputed lighting.

Taking inspiration from variance shadow maps, assume the desired mask is a blurred (by a separate filter) gray-scale image of the shadows of the dynamic parts of the scene that need to be projected on the static parts of the scene. Suppose this image is augmented, before blurring it, with depth values for each occluding texel as in a standard shadow map. If only the shadow mask is blurred, then the blurred mask will extend outside where the depth values were rendered and cut off much of the faux-penumbra of the shadow. If the depth values are also blurred, the depth values will extend as far as the faux-penumbra, but using these depth values as-is for the shadowing test will result in the previous problem observed with variance shadow mapping. A method of recovering the original, unfiltered depth value from the filtered depth value is needed.

Consider the same two texels on a shadow boundary as in the above example, such that a linear filter ***p*** has been applied to the occluder's depth value ***s*** and the far plane depth ***1***. Then the resulting depth value ***d*** retrieved from the shadow map should satisfy the following: $d = p*s + (1-p)*1 = p*(s-1) + 1$. Now, suppose the shadow map is augmented with a shadow mask (a grayscale image of the shadow), that contains the value ***1*** where there is shadow and ***0*** where there is none. The resulting mask value ***m*** retrieved from the shadow map should also satisfy the following: $m = p*1 + (1-p)*0 = p$. Thus this shadow mask will, within the limits of the shadow map precision, both serve as the shadowing to be applied and give the filter ***p*** that was applied to this shadow map texel. Since ***p*** is known, one may directly solve for the actual depth of the occluder ***s***: $s = (d-1)/m + 1$. Finally, once you have the occluder depth ***s***, you test the depth ***z*** of the point to be shadow against ***s***, and if it is farther, then use the mask ***m*** as the intensity of shadowing to apply.

Implementation

The steps to apply this representation become:

1) clear the shadow map so all depth values are at the far plane ***1***, and all mask values are ***0***
2) for each occluder render depth ***d*** as in a normal shadow map, but also a mask value of ***1***
3) apply a desired linear filter as with variance shadow mapping
4) when shading, recover the unfiltered depth ***s = (d-1)/m + 1*** and test against it. If shadowed, use the filtered mask ***m*** directly as the shadow intensity.

Advantages

The advantages of this representation are that there are no squared terms as in variance shadow mapping, so the precision demands are much less. An 8-bit-per-component shadow map texture, which can be bilinearly and trilinearly filtered on most consumer 3D hardware, is feasible provided the depth range of the light is not extreme. Larger depth values biases may also suffice to accommodate reduced precision usages, as for the intended use one is merely trying to cast shadows from objects onto a scene, rather than compute fine-grained self-shadowing within any individual object.

Disadvantages

The representation is only suitable for casting a soft-edged shadow of the fused silhouette of some shadow casters onto a scene. While it can be used for self-shadowing as in a normal shadow map, it would fail to provide soft shadowing for that purpose.

In the case occluders at different depths are filtered together, the reconstructed occluder depth will end up some weighted average of them, which can cause shadowing artifacts in some cases. In the case that there is a shadow receiver between two shadow casters along a single light ray, the reconstructed occluder depth value may be pushed behind the shadow receiver such that that receiver fails to shadow. The representation is most suitable for scenes where there is not a complex intermingling of objects and scene or where such artifacts would be transient and not often noticed.

Integrating with Single-pass Lightmaps

If one uses a single-pass lightmapping system that combines the contribution of all lights in a scene into a single lightmap and uses a single shadow map to mask off this lightmap and create faux-shadows, the resulting faux-shadows may “bleed” through surfaces. This is, however, not a weakness of the above representation, but of applying a shadow map to a single-pass combined lightmap.

Suppose a character model is standing above a bridge, with a single sunlight casting down upon him. The shadow map is likewise oriented to cast faux-shadows downward and will determine that all points below the character are in shadow. Assuming the sunlight is the only light source, the lightmaps would already encode the fact that the area under the bridge is in shadow, so that masking off any light underneath would only shadow an area that is already in shadow. Thus, in the case of a single light per lightmap, masking off the light it provides should not cause artifacts. Suppose, however, that there is a torch underneath the bridge. The character's faux-shadow will also mask off this torch's light, even though the character is not visible in the are underneath the bridge. Similar artifacts might occur if indirect lighting were to brighten the area in which the character's shadow fell. Ideally, the character should only cast a faux-shadow on the surface immediately behind him from the light's point of view.

This clamping of the shadow can be accomplished by adding an additional value to the shadow map that sets a far bound on the shadowing region, in addition to the near bound already set by the occluder depth stored in the shadow map. A method to generate this far bound, inspired by depth peeling, proceeds as follows:

1) clear a third component of the shadow map storing the far bound to the far plane ***1*** when clearing the other two components
2) render occluder depths and mask values into the shadow map, leaving the far bound untouched
3) invert the z-buffer test so that only surfaces behind the occluders will render into the shadow map, and disable z-buffer writes so that all such surfaces behind the occluders will be rendered
4) using min-max blending functionality, such as the GL_EXT_blend_minmax extension in OpenGL, set the blend function to accumulate the minimum value in this third far bound component, either by utilizing a color mask if available so that only the third component is written to, or by setting the other values of components to be blended to some suitable value such as ***1*** so that they remain unaltered
5) render the shadow receivers - preferably the back faces so as to minimize depth biasing problems
6) when shading, reconstruct the far bound in parallel with the occluder depth (taking advantage of the vector nature of instructions on GPUs), using the same method described above, and verify that the pixel to be shadowed also lies within this far bound to determine if it is actually shadowed

This method, while requiring the shadow receivers to be rendered into the shadow map, is still never-the-less likely more efficient than rendering the entire scene into an entire shadow map, as due to the inverted z-buffer test and the fact that unshadowed texels of the shadow map have their z-buffer values initialized to a suitable far plane, most of the shadow map will be effectively stenciled off by the 3D hardware, rejecting all writes to the shadow map except to specifically those areas where there are shadow casters.

Also, since renderable two-component formats are, at the time of this writing, relatively rare, the space overhead of this extra component is amortized – the extra component would otherwise go unused.

Efficient Blurring

Even using a separate blur filter, the number of texture lookups per shadow map texel to achieve a desirable blur can become expensive on older 3D hardware. Taking advantage of the fact that the shadow casters occupy the shadow map only sparsely, one can limit the blur to take place only within specific portions of the shadow map containing the shadow casters where it is actually needed.

One simple way to achieve this is to subdivide the shadow map into a grid, rendering the bounding square of the shadow casters into the grid. The grid can then be grouped into maximally sized quadrilateral regions using a simple run-length step upon non-empty grid squares. Blurring is then performed only within these quadrilaterals. If size of the grid is adequately chosen, each row of the grid can be efficiently represented by a bit vector, reducing the cost of managing the grid. This same grid can also be used to test if a shadow receiver is within the area of influence of any shadow caster, by checking its bounding square in the shadow map against this grid, also making efficient usage of the bit vector representation of grid rows.

Possible Extensions

If one considers that mask values for shadows may be rendered as anywhere in the range of ***[0,1]***, rather than just ***0*** to indicate a background and ***1*** to represent an occluder, one may directly render the transparency of an object into a third shadow contribution component, which gets used as the final shadowing contribution instead of the mask value, allowing a transparent occluder to only contribute partial shadowing.

References

[1] William Donnelly and Andrew Lauritzen, "*Variance Shadow Maps*," In *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games 2006*.