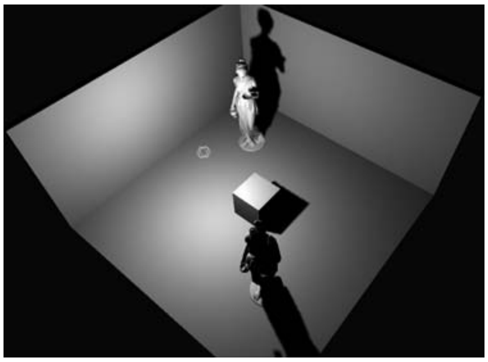
Practical Implementation of Dual Paraboloid Shadow Maps

Brian Osman[*] Vicarious Visions Activision

Mike Bukowski[†] Vicarious Visions Activision

Chris McEvoy[‡] Vicarious Visions Activision



Abstract

In this paper, we present refinements to dual paraboloid shadow mapping algorithm from Brabec, et. al. This work makes the algorithm practical for broader use in video games. We give solutions for the tessellation constraints on shadow casters and receivers present in the original work. We also discuss heuristics for splitting plane placement and pitfalls in filtering.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: dual paraboloid, shadow mapping, video games

1 Introduction

In real time rendering applications, e.g. video games, one of the common algorithms used for rendering shadows is shadow mapping [Williams 1978]. The traditional shadow mapping algorithm consists of two passes. First, the scene is rendered from the light's point of view. The depth values from this render are stored in a texture called a shadow map. This must be done for each light in the scene. In the second pass, the scene is rendered again from the camera's point of view. During this render, each pixel is checked to see if it is obscured, i.e. in shadow. This is done by looking up the depth value for the given pixel in the shadow map generated in the first pass. If the value from the shadow map is less than the actual depth value of the pixel, it is in shadow. For each light, a shadow map comparison must be made.

[*]e-mail: osman@vvisions.com
[†]e-mail: mbukowski@vvisions.com
[‡]e-mail: chris@vvisions.com

Because the scene must be stored in a single texture, the traditional shadow mapping algorithm can only be used for lights with a limited field of view. This makes the traditional approach suitable for spotlights but impossible for lights with a greater field of view. Shadowing hemispherical lights and point lights requires a different approach. Two shadow mapping techniques for such lights are cube shadow maps and dual paraboloid shadow maps.

Cube shadow maps are a simple extension of the traditional shadow mapping technique. The scene is rendered six times from the position of the point light. Each render is then stored as a face in a cube map. The full 360 degree field of view is accurately accounted for. During the lighting pass, cube map lookup functions are used to sample depth values from the cube shadow map. This approach will result in very accurate shadows for point lights, but can result in poor performance due to the number of render passes. Cube shadow mapping performance can be improved by only rendering faces in the shadow map that will actually be needed for lookup [King and Newhall 2005]. Even with performance optimizations, the scene will need to be rendered many times per light.

Dual paraboloid shadow mapping, DPSM, is another extension of the traditional shadow mapping technique [Brabec et al. 2002]. DPSM is based on view-independent environment mapping [Heidrich and Seidel 1998]. When using DPSM, the scene is rendered once for each 180 degree field of view. This means only one shadow render is needed for hemispherical lights. Point lights require a render for each of its hemispheres. When the shadow maps are put back to back, a full 360 degree field of view can be represented. During shadow render each vertex of the object is transformed into paraboloid space and stored in the depth texture as such. The resulting depth texture is a paraboloid representation of the scene. During the scene render, texture coordinates must be generated for the paraboloid shadow map. Once the depth has been obtained from the paraboloid shadow map, the traditional shadow mapping comparisons can be made.

The ability to render a full 360 degree field of view in only two render passes does come with drawbacks. With the traditional DPSM implementation, both shadow casting and shadow receiving geometry needs to be sufficiently tessellated. It will be shown later that a simple modification to the standard implementation can eliminate the need for tessellation on shadow receivers. Tessellation is needed due to the fact that each vertex must be transformed into paraboloid space. If polygons are large with respect to the light, severe distor-

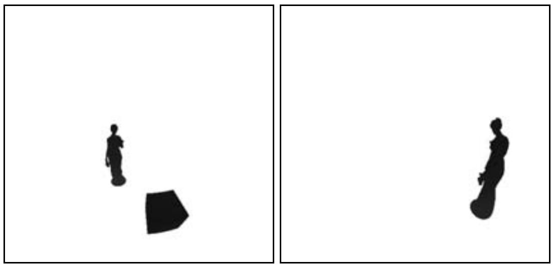


Figure 1: Front and back hemispheres of a dual-paraboloid shadow map.

tion or bending of shadows can occur. This is the result of linearly interpolating non-linear vertex data. If the geometry is sufficiently tessellated, the error due to interpolation is manageable.

On the surface, DPSM may seem infeasible due to the tessellation restriction. In reality DPSM can be a very effective method for implementing real-time shadows. It will be shown that many of the problems associated with DPSM can be mitigated if care is taken when constructing a real-time lighting architecture.

2 Paraboloid Transform in the Pixel Shader

The core problem with DPSM that necessitates tessellated geometry is the non-linear space. Because the GPU performs linear interpolation during rasterization, any values that lie in paraboloid space are not interpolated correctly. When generating the shadow map, there is little that can be done to avoid this, on hardware where the interpolation method is fixed. However, we can solve the problem during the lighting pass.

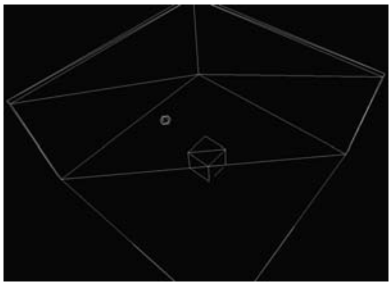


Figure 2: Wireframe image of low polygon test scene.

The standard approach to DPSM suggests always doing the paraboloid transform in the vertex shader. This causes texture coordinates (in paraboloid space) to be interpolated linearly, giving incorrect results, as seen in figure 3. Our approach is to instead send the world-space position of the vertex in question. With this approach, the hardware's linear interpolation gives us the world-space location of the pixel to be shaded, and the interpolation is correct. Then, we transform the position into the light's paraboloid space for each pixel. This gives us correct shadow lookup, even when the shadow receiving geometry has very few polygons. The improvement can be seen in figure 4.

Effectively, by shifting some of the work from the vertex shader to the pixel shader, we have eliminated the tessellation requirement from shadow receivers. Although shadow casters must still be tessellated, we can project shadows on large environmental geometry (floors and walls) without excessive polygon counts. In a constrained environment, this can remove much of the tessellation restriction imposed by traditional DPSM. If many of the shadow casters are smaller or more tessellated, like character models, then high quality shadows can be achieved.

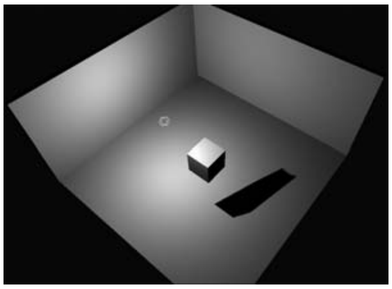


Figure 3: Incorrect shadows from vertex shader transformation.

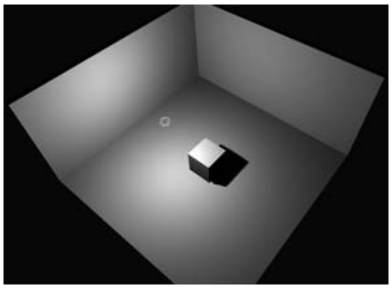


Figure 4: Improved shadows from pixel shader transformation.

3 Hardware Tessellation

We have already demonstrated a simple and reasonable solution to DPSM's tessellation requirements on shadow receivers. In some cases, this will be sufficient to make the technique usable in real-world scenarios. In other cases, though, we will have shadow casters that are not sufficiently tessellated. Figure 5 shows a low polygon caster very close to the light. If we were able to tessellate the caster geometry, these errors could be minimized and/or eliminated. Modern GPUs are beginning to support automatic hardware tessellation of input primitives using a variety of algorithms [Vlachos et al. 2001]. Our target platform was the Xbox 360 game console, which includes this capability. The feature is primarily intended for techniques like displacement mapping, or other vertex shaders that create sub-polygon detail in the final rasterization. It also suits our needs, though, by simply shrinking the final triangles. Most importantly, it enables us to render correct dual-paraboloid shadow maps, even when some of our casters would otherwise lack sufficient tessellation.

Our implementation used a constant tessellation factor during shadow map generation to ensure that triangles were sufficiently small for our typical game scenes and camera angles. A more advanced implementation would precompute the maximum edge length or area of triangles within a single mesh. Using that information, and the position of the mesh relative to the light and splitting plane, the tessellation factor can be adjusted to subdivide geometry only when necessary.

It is important to note that the tessellation requirement is not static. The amount of tessellation a shadow casting object requires is determined by its distance from the light. The actual restriction is on the

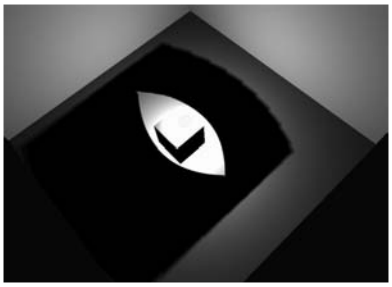


Figure 5: Incorrect shadows from low polygon caster.

size of each rasterized triangle within the shadow map. This knowledge, combined with the hardware tessellation controls of modern GPUs, enables us to automatically tessellate only those objects that require it. As a pre-process, it is easy to determine the area of the largest triangle in each mesh. During shadow render, we can adjust the hardware tessellation parameters to ensure that no rendered triangle exceeds a given size, based on the object's distance to the light. Figure 6 shows a tessellated caster near the light, and the resultant improvement in shadow quality.

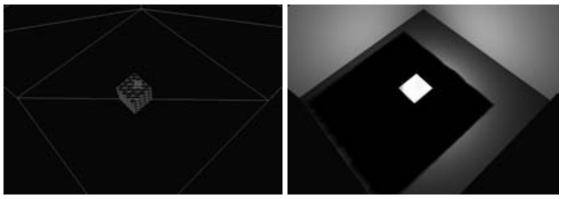


Figure 6: Using a (hardware) tessellated caster creates correct shadows.

4 Splitting Plane Choice

Every problem unique to DPSM is a result of the warped paraboloid space, and its interference with linear operations performed during rasterization and interpolation. Examining a paraboloid shadow map, it is apparent that this causes the most distortion at the perimeter of the valid region, along the plane that divides the two hemispheres of the light. Pixels that fall in the center of either hemisphere suffer little distortion, and generally yield high quality shadows. As an example, the light's splitting plane in figure 5 runs directly through the caster. This contributes to the severe shadow errors. Choosing a splitting plane parallel to the ground would have been a better choice in this scene.

Previous work on DPSM implicitly divided the world along the world-space z-plane. This isn't required and choosing a different splitting plane can help to eliminate artifacts. Modifying the splitting plane is accomplished by simply assigning a full camera matrix to the light (as is done with traditional frustum based spotlight shadow mapping). The rotation of the light's camera also rotates the splitting plane, which will always be the camera plane that passes through the light's position.

Remember that any shadow caster geometry that falls on both sides of the splitting plane will need to be rendered twice, once for each shadow map. Thus, by carefully choosing the splitting plane, we can limit artifacts, and we can also affect performance. Depending on scene construction, various axis-aligned orientations of the splitting plane may interesect more or less geometry. Orienting the plane so that the artifacts fall outside the camera frustum, or in areas where the user is less likely to notice artifacts, can also be very beneficial. Of course, it may not be possible to achieve both goals simultaneously. Nevertheless, careful orientation of the light's hemispheres is a valuable tool when using DPSM.

5 Filtering

Aliasing along the edges of shadows are a common problem in many shadow mapping implementations. A large amount of work has been done exploring how to limit these artifacts [Chan and Durand 2003]. Many techniques involve alternative projections or spaces to use more shadow map texels in areas of interest [Stamminger and Drettakis 2002]. These approaches are unsuitable to DPSM, because of the special (fixed) projection required.

The other primary method of reducing aliasing involves filtering of the shadow map during the lighting pass. Percentage closer filtering, PCF, is often used [Reeves et al. 1987]. PCF is often implemented in hardware during the texel lookup [Brabec and Seidel 2001]. More complex filtering is often performed in the pixel shader, using multiple texture samples and large filter kernels. This type of filtering works to a limited extent with DPSM, but causes nearly as many problems as it solves.

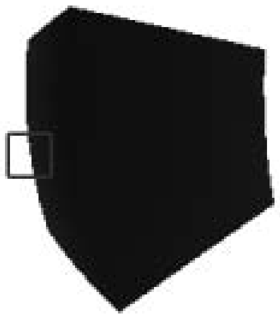


Figure 7: Normal (exaggerated) filter kernel applied to DPSM.

Because normal texture filtering is done in image space, the filter kernel samples a rectangular region of the shadow map (see figure 7). Near the center of the map, this is essentially correct. However, as the pixel being shaded gets closer to the edge of the paraboloid, the filtering becomes less accurate. In the worst case, filtering can cause samples to be read from outside the valid region of the shadow map when the shaded pixel is on or very near the perimeter of valid region. In world space, these samples correspond to the splitting plane that divides the two hemispheres of the light. This plane is often the source of artifacts with DPSM. Unfortunately, using aggressive filtering will often improve the quality of shadows near the center of the map, but degrade the shadows along the spliting plane.

As future work, consider a DPSM-aware filtering method. At the cost of more pixel shader instructions, correct filtering could be performed with a dual-paraboloid shadow map by applying the filter kernel before the conversion to paraboloid space. Each adjusted sample would then be transformed into paraboloid space independently. This would cause the samples generated by the filtering to be correctly warped into paraboloid space, as illustrated in figure 8.

Figure 8: DPSM-aware (exaggerated) filter kernel.

Using this scheme, pixels along the splitting plane would fetch and average texels from both hemispheres of the shadow map.

6 Conclusion

We have presented improvements to the original DPSM algorithm that permit using the technique in a wider variety of settings. We have demonstrated how transferring work to the pixel shader can eliminate the need for tessellation of shadow receivers. Simultaneously, hardware tessellation and clever choice of splitting planes can address the need for tessellation of shadow casters. Most of these techniques are driven by advances in graphics hardware technology since the original DPSM paper was published. We have also suggested a possible further refinement to filtering of dual-paraboloid shadow maps. These improvements make the techique practically usable for real-time applications under some constraints.

We have integrated our implementation of DPSM into an existing game engine on the Microsoft Xbox 360 Game Console. The implementation supports four simultaneous shadow casting point lights. The engine uses the Xbox 360's hardware tesselator to tessellate caster geometry. The splitting plane can be aligned to the camera view to reduce artifacts. Rendering a typical scene with about 50,000 polygons, with game systems active, we observed a solid 30 frames per second.

7 Acknowledgements

We wish to thank Gil Gribb of Raven Software for originally suggesting that we investigate DPSM.

References

BRABEC, S., AND SEIDEL, H.-P. 2001. Hardware-accelerated rendering of antialiased shadows with shadow maps. *Computer Graphics International*, 209.

BRABEC, S., ANNEN, T., AND SEIDEL, H.-P. 2002. Shadow mapping for hemispherical and omnidirectional light sources. In *Computer Graphics International*.

CHAN, E., AND DURAND, F. 2003. Rendering fake soft shadows with smoothies. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 208–218.

HEIDRICH, W., AND SEIDEL, H.-P. 1998. View-independent environment maps. In *HWWS '98: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM Press, New York, NY, USA, 39–ff.

KING, G., AND NEWHALL, W. 2005. Efficient omnidirectional shadow maps. In *ShaderX3: Advanced Rendering with DirectX and OpenGL*, Charles River Media, Hingham, MA, USA, 435–448.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 283–291.

STAMMINGER, M., AND DRETTAKIS, G. 2002. Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002*, ACM Press/ ACM SIGGRAPH, J. Hughes, Ed.

VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. L. 2001. Curved pn triangles. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 159–166.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 270–274.