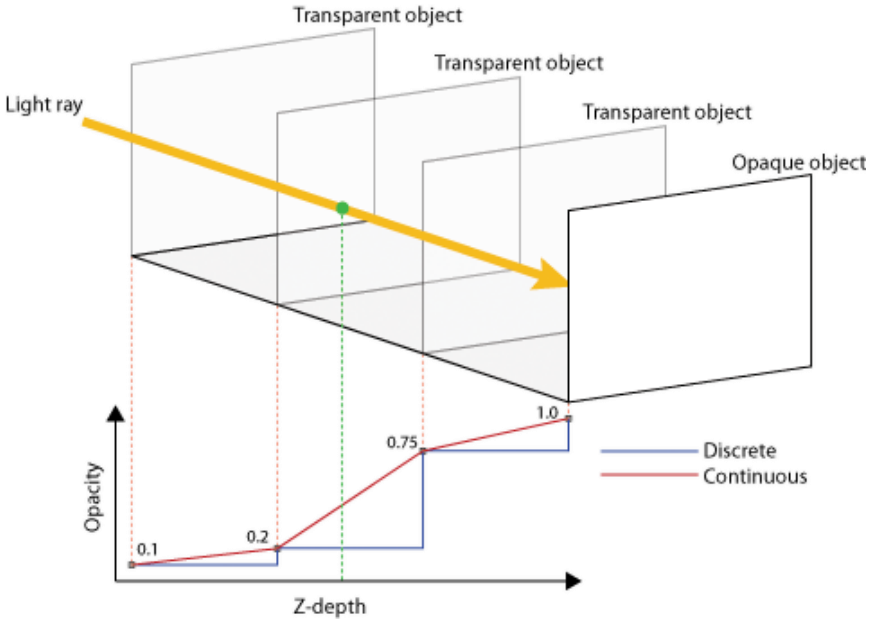
Deep Shadow Mapping

Sandeep Thippeswamy thippesw@usc.edu, Sohil Himanish himanish@usc.edu, Ansh Bahri abahri@usc.edu, Anuja Chandorkar achandor@usc.edu

Abstract:

Implementing Deep Shadow Mapping, a technique for producing high quality shadows. Traditional shadow maps store a single depth at each pixel, whereas deep shadow mapping uses a representation of fractional visibility through the pixel at all possible depths.

Introduction:

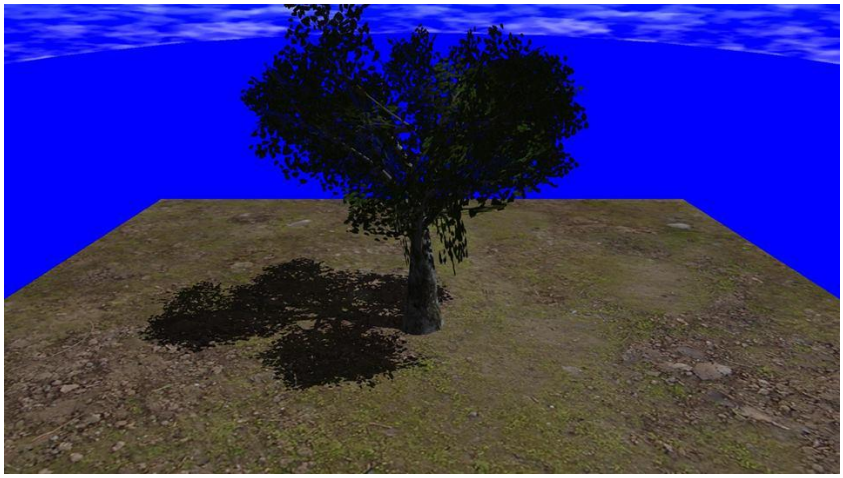


If there were no transparent objects in the above diagram, we could use a traditional shadow mapping method. But due to the presence of transparent objects, the shadows formed are inaccurate. Hence transparency of the objects has to be taken into account for accurate rendering of shadows.

As shown in the above diagram, the ray of light passing through the objects considers the alpha component for determining the transparency.
Since the 1st transparent object is 10% opaque, it allows 90% of light intensity to pass through. Similarly for the other objects, the light intensity varies at each depth. Therefore, each object is lit by a different intensity of light, having a different level of brightness, giving the effect of self shadowing. Of course, if the intensity of light is 0 on any surface, the surface will be black (assuming there is no ambient light).

We use DirectX11 as the basis of our renderer which we extend to implement deep shadow mapping. We start creating the test scene by taking quads as branches. Then we add a leaf texture on the quads. Now we alpha blend this quad with a transparent shader and get a composite image which exhibits partial transparency. Then we use a shader to create shadow maps for this image. This gives us the required result for rendering shadows for volumetric transparent objects. The shadows obtained are deep shadow maps as we take quads of leaves which are at varying depths from the light source. Hence we can see self shadowing between the branches, i.e. the shadow of one branch falls on another branch.

Pixel color = (Alpha * light * color) at a depth z. We keep accumulating these pixel values till we find an alpha=1.0.



Rather than storing a single depth at each pixel, a deep shadow map stores a fractional visibility function (or simply visibility function) that records the approximate amount of light that passes through the pixel and penetrates to each depth. The visibility function takes into account not only the opacity of the surfaces and volume elements encountered, but also their coverage of the pixel's filter region.

Comparison with traditional shadows:

Deep shadow mapping supports prefiltering and has faster lookups compared to other traditional shadow mapping techniques. This method is much smaller in comparison to an equivalent high resolution depth map (dependent upon compression). Fortunately, at any sampling rate there is an error tolerance that allows significant compression without compromising quality. Shadows of detailed geometry have an expected error of about O ($N^{-½}$). This error is a measure of the noise inherent in the sampled visibility function. Actually implemented with a tolerance of 0.25* ($N^{-½}$) – half of the maximum expected noise magnitude. TSM uses O (N) storage, where DSM uses O ($N^{-½}$), approaches O ($N^{-¼}$) when functions are piecewise linear.

Significantly it becomes more expensive to compute than a regular shadow map at the same pixel resolution. Even bias artifacts are possible, due to constant z depths. Deep shadow maps are exacerbated by encouragement of large filter widths. This might be useful, as it provides an extra degree of freedom. Shadow resolution should be chosen according to minimum filter width desired (shadow detail) and number of samples per pixel should be determined by maximum acceptable noise.

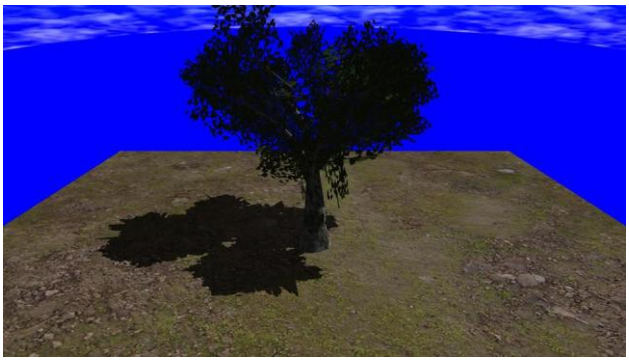
Advantages over simple shadow mapping:

- Deep shadow mapping supports semitransparent surfaces and volumetric primitives such as smoke.
- For high-quality shadows, they are smaller than equivalent shadow maps by an order of magnitude and are significantly faster to access.
- Unlike ordinary shadow maps, they support mip-mapping. This can dramatically reduce look up costs when objects are viewed over a wide range of scales.
- They can efficiently support high quality motion blurred shadows.

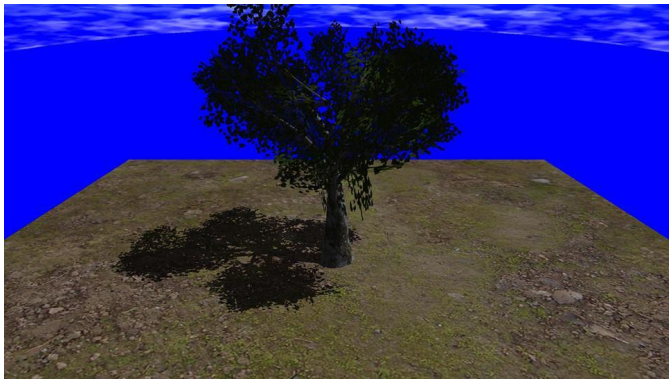
Results:

In our project, we have shown the results of the deep shadow maps using different levels of alpha of the transparent shader for the varying levels of transparency for the leaves of the tree.

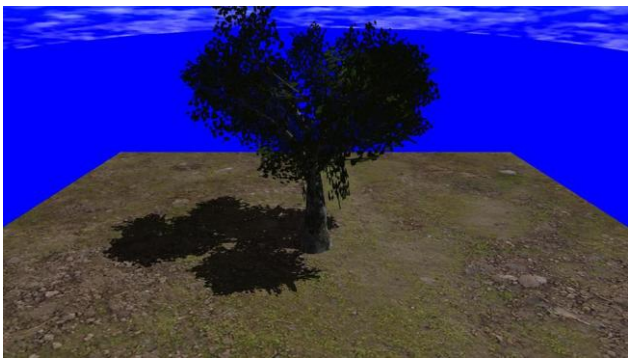
With the traditional shadow mapping technique, we get the following shadow map. As seen, it is blocky in nature, and fails to exhibit the fine detailed shadow which takes into considerations the leaves.

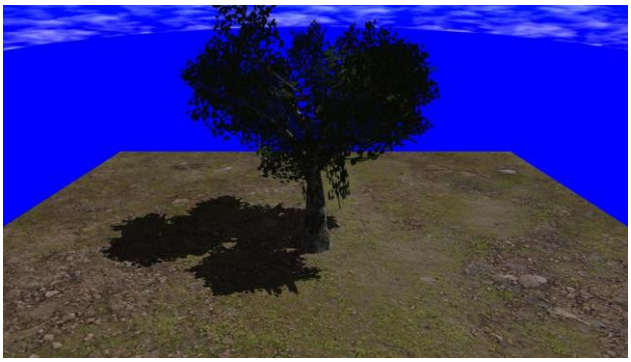


We were able to generate the following shadow map which takes into account leaves at every layer and creates a shadow map which is realistic and accurate.



Following are some of the images which were obtained by tinkering with the renderer.





Challenges faced:

One of the main challenges we faced was choosing a renderer. We felt that things would be much easier on a more complete rendering platform. We tried different approaches such as creating a plugin in MAYA, Opengl, and Directx11.
However, we did not know much about the Maya plugin creation whereas among the DirectX and OpenGl platforms which are similar, we decided to use DirectX since we were a bit familiar with it. Another challenge was learning the advanced functionality we needed in DirectX such as HLSL, Vertex Shaders , Lights etc. Everything that we have learnt in graphics, we found an api which did something similar, so our learning was accelerated. We read up on transparency, and came across Alpha blending. Since we did have an idea about how alpha blending worked, we went through a few tutorials to get better acquainted with it. We implemented our visibility function using the alpha channel. The final difficulty which we

came across was actually finding input data online. Since our input required complex geometry, it was hard to find. We went through a lot of tutorials and material before settling on this one. We have another input in the works which is Clouds, but we were not able to finish the pixel shaders for these in the given time.

Conclusion:

[1] implements deep shadow mapping using a much more complex and computationally intensive approach. We used a simplistic approach which follows very closely to the underlying concept of deep shadow maps. And even though we might compromise on accuracy, quality of the shadow maps a bit, we are still able to reproduce very close deep shadow maps for complex transparent objects.

References:

[1] [Lokovic 2000] Tom Lokovic and Eric Veach. “Deep Shadow Maps.” SIGGRAPH 2000

[2] [ZC03] ZHANG C., CRAWFIS R.: Shadows and soft shadows with participating media using splatting. IEEE Transactions on Visualization and Computer Graphics 9, 2 (2003), 139–149.

[3] Tom Lokovic and Eric Veach. 2000. "Deep Shadow Maps." In Proceedings SIGGRAPH 2000. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. 385-392

[4] BRABEC, S., AND SEIDEL, H.-P. 2002. Single Sample Soft Shadows Using Depth Maps. In Proc. Graphics Interface, 219–228.

[5] http://www.rastertek.com/ DirectX 11 tutorials