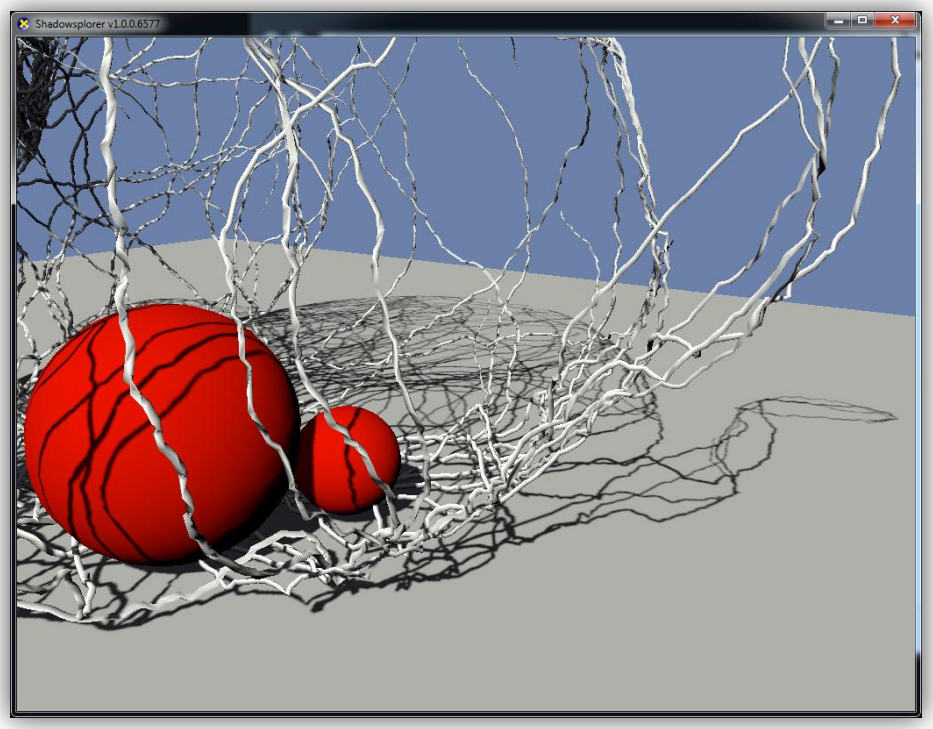
Comparing Shadow Mapping Techniques with Shadow Explorer



Introduction

Most modern games use shadows to some degree. The challenge is to know which algorithm to use and what the tradeoffs are of different techniques—what combination of quality and performance is best suited for the game. This sample, Shadow Explorer, lets the user compare and contrast four different algorithms, adjust various parameters for each one, and observe the effects in real time. The shadow mapping algorithms presented are: simple, percentage closer filtered (PCF), variance (VSM), and exponential variance (EVSM).

Sample Usage

At the highest level, this sample allows the user to compare and contrast the quality and performance characteristics of the different shadowing algorithms. There are two scenes that each algorithm can be applied to: a city and a teapot. The city scene represents a "typical" setting that might be encountered in a game, consisting of a variety of object sizes, dimensions, and other characteristics. The teapot is a "worst case" scenario consisting primarily of thin primitives, casting shadows not only on themselves but also on curved surfaces and a flat plane.

Sample Architecture

Shadow Explorer is implemented as a Microsoft DirectX* application based on the DXUT framework. All four shadow mapping algorithms basically follow the same code path, except that different shaders are used and that VSM and EVSM run a filter over the shadow map before the scene is rendered. Listing 1 shows the basic flow of the source code for creating and using the shadow map. Every frame, the shadow map is created in `RenderShadowMap()` which builds a standard cascaded shadow map by running the *SceneZ* shader. As mentioned, for VSM and EVSM the shadow map is then filtered using the *FilterV* and *FilterH* shaders. After the shadow map is created, the scene is rendered with the shader *SceneMain* which, depending on the algorithm selected, will call a different version of `IsNotInShadow()`. This is done by simply including different .fxh files and recompiling the shaders when a new algorithm is selected. For example, if PCF is selected then `IsNotInShadow()` will call the version in filter_PCF.fxh.

```
static void RenderShadowMap(...)
{
   // For each cascade
   for( int i = 0; i < iLayers; ++i )
   {
      // Set the Z only pass
      SetRenderTargets( 0, NULL, g_d3d.pShadowMapDSV[i] );
      g_d3d.pTechSceneZ->GetPassByIndex( 0 )->Apply( 0,
         pd3dImmediateContext );
      // Render the appropriate scene
      g_pSelectedMesh->Render( pd3dImmediateContext );
   }

   // VSM and EVSM techniques require additional filters
   if(Filter_Type_VSM || Filter_Type_EVSM)
   {
      // Use the shadow map just generated in the filters
      g_d3d.pVarShadowTex->SetResource( g_d3d.pShadowMapSRV );

      // Run the vertical filter
      SetRenderTargets( 1, &pShadowMapRTV_vsm[0].p, NULL );
      g_d3d.pTechFilter_V->GetPassByIndex( 0 )->Apply(...);
      pd3dImmediateContext->Draw(...);

      // Use the result of the vertical filter
      g_d3d.pVarShadowTex->SetResource( g_d3d.pShadowMapSRV_vsm[0] );

      // Run the horizontal filter
      SetRenderTargets( 1, &pShadowMapRTV_vsm[1].p, NULL );
      g_d3d.pTechFilter_H->GetPassByIndex( 0 )->Apply(...);
      pd3dImmediateContext->Draw(...);
   }
}

static void RenderScene(...)
{
   RenderShadowMap(...)

   if(Filter_Type_VSM || Filter_Type_EVSM)
      // Use the filtered shadow map
      g_d3d.pVarShadowTex->SetResource( g_d3d.pShadowMapSRV_vsm[1] );
   else
      g_d3d.pVarShadowTex->SetResource( g_d3d.pShadowMapSRV );
   g_pSelectedMesh->Render(...);
}
```

Listing 1 – Basic flow of application code in Shadow Explorer

The simplest technique presented is that of a basic, cascaded shadow map. This is obviously going to be the fastest method as it does the least amount of work, but leaves a lot to be desired in terms of quality. The next technique is PCF, which works off the same shadow map but samples the texture multiple times to alleviate aliasing problems and soften the shadow edges. Better results can be achieved by

taking more samples, but with the expected tradeoff in performance. By using a non-uniformly distributed sample distribution, fewer samples need to be taken to achieve good results. Shadow Explorer uses a pre-calculated Halton sequence to calculate "random" offsets instead of a grid based offset. Even using the non-uniform method, PCF still requires a lot of sample look ups, making the algorithm fairly time consuming on the GPU.

The next technique, VSM, takes a different approach by storing both the depth and depth squared values into the shadow map. In Shadow Explorer, the shadow map is rendered normally and the depth squared value is added during a separate pass immediately after the z-pass is done. Additionally, this second pass runs a box filter over the data to soften the edges of the shadows. The main drawback to the VSM algorithm is the light bleeding effect when multiple occluders overlap each other and the ratio of their distances from the shadow receiver is high. This can be seen in Figure 1, where the shadow of the tall building in the background is outlined in light where it overlaps the shadows of the two smaller buildings in the foreground.

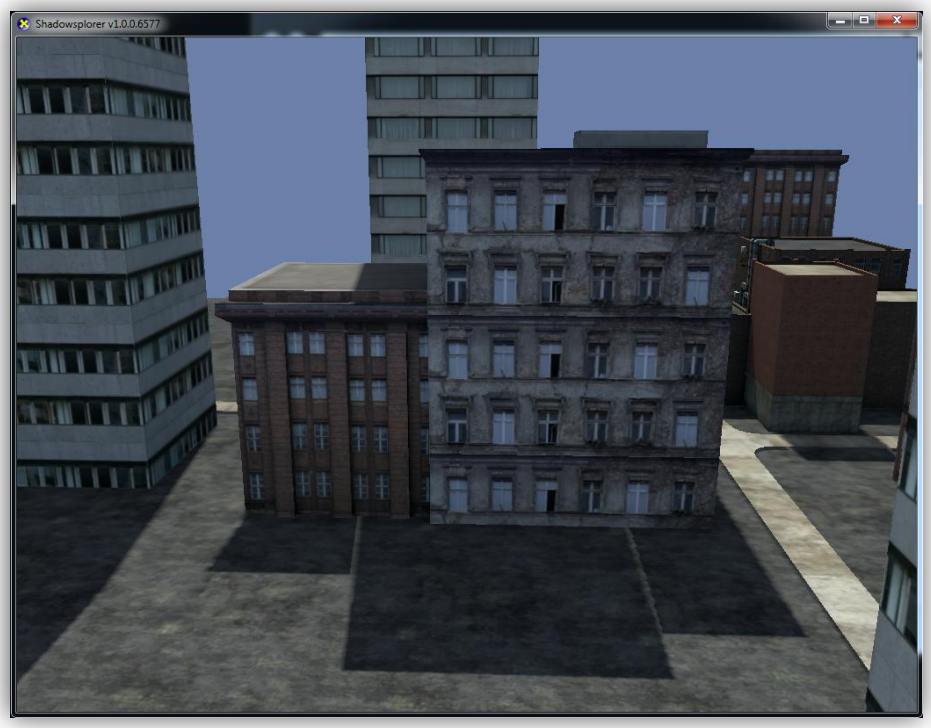


Figure 1 - VSM results in light bleeding artifacts

Depending on the scene, light bleeding may not be noticeable enough to be a problem. If it is, EVSM can be used to eliminate the problem, as can be seen in Figure 2. The main difference with VSM is that EVSM "warps" the depth value when storing and reading from the shadow map. This has the effect of bringing the relative distances of the occluders to the receiver closer together in order to minimize or eliminate light bleeding. Shadow Explorer does this in the shader function `WarpDepth()`.

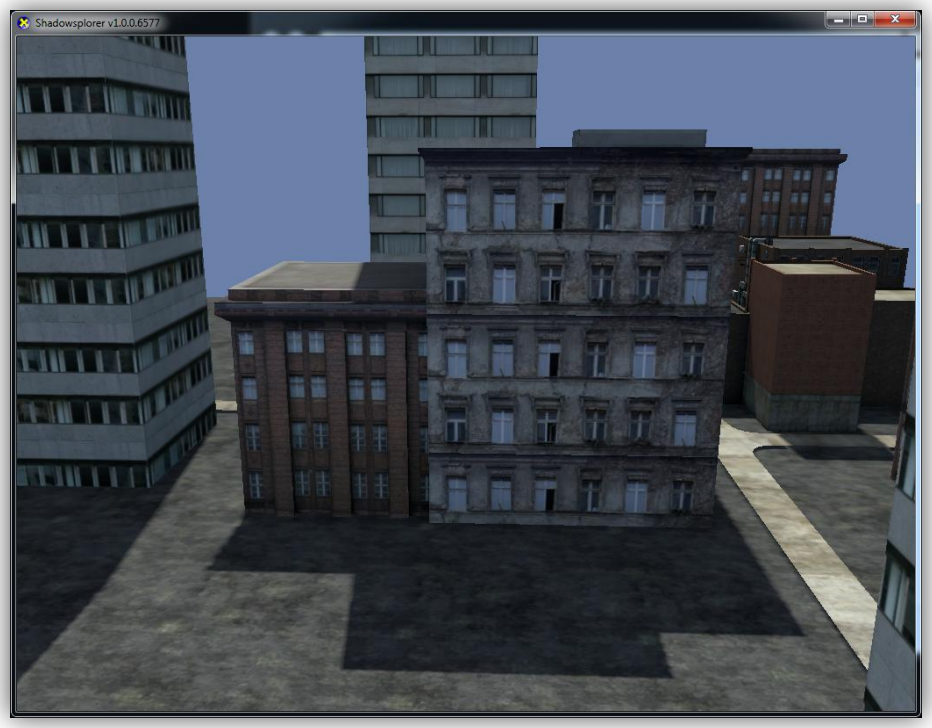


Figure 2 - EVSM fixes light bleeding

Conclusion

Shadow Explorer implements four shadow mapping techniques with runtime modification of various parameters so the user can compare both the performance and the quality of the methods. For each technique, care was taken to ensure optimal performance of the algorithms. Some of the optimizations done include using half floats instead of floats, unrolling loops in PCF, running the vertical filter before the horizontal filter, selecting the cascade Z interval instead of fitting to best cascade, and calculating shadows only for triangles oriented towards the light. Additional resources, including both binary and source code versions, are available for download at http://software.intel.com/en-us/articles/shadowexplorer.

Some areas for further development would be the addition of spot and point lights to demonstrate the flexibility of the shadow techniques. Animated objects could also be added to create an environment more representative of a game. Also, different algorithms could be used to more accurately place the split planes.

Controls

Shadow Explorer allows the user to change a variety of parameters to modify the shadow algorithms and change the shadow technique being used. Here is a list of the controls along with a brief description:

1. **Toggle full screen** - Turns full screen on/off
2. **Change device** - Change the d3d device
3. **Scene drop down list** - Change the scene being viewed
4. **Align light to camera** – Rotates the light along the light direction for optimal usage of shadow map space.
5. **Algorithm drop down list** - Change the filtering method being used
6. **Shadow Map BPP** - How many bits are used for each pixel in the shadow map
7. **SM resolution** - resolution of the shadow map
8. **Filter size** – The size of the filter used in PCF, VSM, and EVSM
9. **Cascade layers** - How many cascades are used
10. **Cascade Factor** – Adjusts how the split space is partitioned by split planes
11. **Aperture** – Spatial screen space filter aperture for PCF
12. **Visualize Cascades** - Visualize the different cascade levels
13. **Visualize Light Space** - Displays the scene from the light source's point of view.
14. **Use Texture Array** – Use texture array for cascades. Each cascade layer receives one texture, otherwise a texture atlas is used.
15. **Z interval selection** – How to determine which cascade a particular pixel is in. If enabled, then only the view space z distance is used. Otherwise, the world position of the pixel is tested against each cascade.
16. **Deduce Z range** – Calculates z-range from the current view. Otherwise uses the scene bounding box as the upper bound.
17. **Deduce Res** – Resolution of the render target used to calculate z range.
18. **Downscale Factor** - Every subsequent GPU deduce pass uses a render target texture of smaller size. This option determines the relation between texture sizes of subsequent passes.
19. **Downscale Limit** - Maximum texture size when Z-range deduction is performed on GPU. At some point it should be more efficient to pull the whole resource to the CPU and finish deduction there, saving several draw calls.

About the Authors

Alexey Rukhlinskiy is a Graphics Software Engineer with Intel's Advanced Visual Computing Division, and has been involved in professional graphics software development for over 10 years. Alexey received his MS degree in Computer Science from Novosibirsk State University in 2002.

Quentin Froemke has been plunking away at a keyboard for over 10 years and is currently ensconced in Intel's Visual Computing Software Division where he helps optimize and improve PC games.

References

1. Williams, L. 1978. Casting curved shadows on curved surfaces. In Proc. SIGGRAPH, vol. 12, 270–274. http://portal.acm.org/citation.cfm?id=807402
2. Donnelly, W. and Lauritzen, A. Variance shadow maps. In SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games. 2006. pp. 161–165. New York, NY, USA: ACM Press. http://www.punkuser.net/vsm/vsm_paper.pdf

3. Lauritzen, Andrew and McCool, Michael. Layered variance shadow maps. Proceedings of graphics interface 2008, May 28–30, 2008, Windsor, Ontario, Canada. http://portal.acm.org/citation.cfm?id=1375714.1375739&coll=GUIDE&dl=GUIDE
4. Isidoro, J. R. Shadow Mapping: GPU-based Tips and Techniques. Conference Session. GDC 2006. March 2006, San Jose, CA. http://developer.amd.com/media/gpu_assets/Isidoro-ShadowMapping.pdf
5. The Halton Sequence. http://orion.math.iastate.edu/reu/2001/voronoi/halton_sequence.html
6. Engel, Wolfgang F. Section 4. Cascaded Shadow Maps. *ShaderX*^{5} , Advanced Rendering Techniques, Wolfgang F. Engel, Ed. Charles River Media, Boston, Massachusetts. 2006. pp. 197–206.