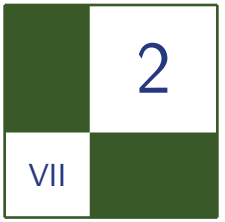

Hybrid Min/Max Plane-Based Shadow Maps

Holger Gruen

2.1 Overview

This chapter presents how to derive a secondary texture from a normal depth-only shadow map. This secondary texture can be used to heavily speed up expensive shadow filtering with big filter footprints. It stores hybrid data in the form of either a plane equation or min/max depth for a two-dimensional block of pixels in the original shadow map. The technique is specifically suited to speeding up shadow filtering in the context of big filter footprint and forward rendering, e.g., when the shadow filtering cost increases with the depth complexity of the scene.

2.2 Introduction

Hierarchical min/max shadow maps [Dmitriev at al. 07] were introduced in order to quickly (hierarchically) reject or accept sub-blocks in a shadow filter footprint that are in full light or in full shadow. For these sub-blocks no additional expensive texture lookup or filtering operations are necessary and this helps to greatly increase the speed of filtering.

Walking the min/max shadow hierarchy does add to the texturing cost. Ideally the texture operations count for quick rejections should be as low as possible. Also min/max shadow maps tend to not always quickly reject pixel blocks for flat features like floor polygons. This is especially true if one pixel in the min/max shadow map maps to a quadrangle of on-screen pixels. To be quickly rejected, one of these on-screen pixels has to either be in front of the minimum depth or behind the maximum depth. Without a big depth bias and all its associated problems

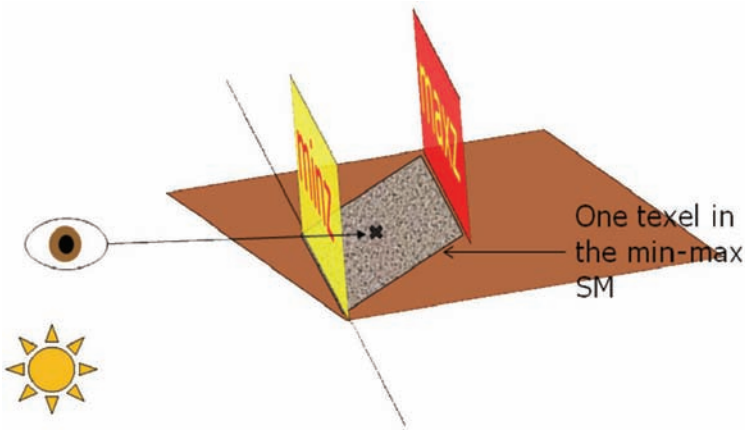

Figure 2.1. One min/max shadow map pixel (noisy quadrangle) can map to many on-screen pixels.

a pixel can usually only be rejected after a deep descent down the min/max hierarchy. This situation is depicted in Figure 2.1.

In order to remedy these shortcomings, this chapter first proposes flattening the min/max texture hierarchy from [Dmitriev at al. 07] to just one min/max texture T that contains the min/max depth data for a block of texels of the original shadow map. This block should be big enough to ideally allow for min/max rejections for all texels of the original shadow map that fall into the block with only one texture lookup.

To also get around losing the quick rejections for planar features, T is used to either store min/max depth data or a plane equation. The plane equation is stored if the block of shadow map pixels that is considered for constructing one min/max pixel lies within a plane. The plane equation allows to decide if a pixel is in front or behind a plane and does not suffer from the need for a high depth bias. Because T stores min/max depth or a plane equation it can be called a *hybrid min/max plane shadow map* (HPSM). A simple form of HPSM has been introduced in [Lobanchikov et al. 09].

The remainder of the text will discuss how to construct an HPSM, how to use the HPSM to quickly reject expensive filter operations, and other uses for HPSMs.

2.3 Construction of an HPSM

If one wants to construct an HPSM from a shadow map, the first thing to decide is what dimension to chose for the HPSM. A typical choice could be to make it 1/4 of the width and 1/4 of the height of the original shadow map.

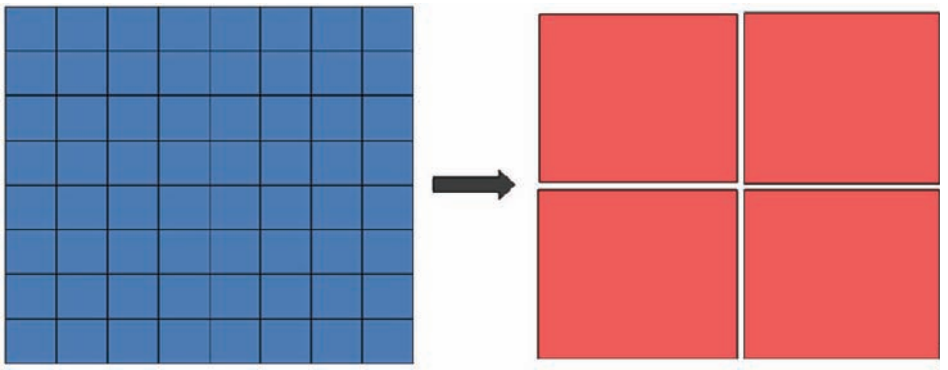

Figure 2.2. A shadow map (left) is converted into an HPSM that is 1/16 the size of the shadow map.

This means that a naive algorithm to construct the HPSM is to just collapse 4×4 texels of the shadow map into one pixel of the HPSM (see Figure 2.2).

Unfortunately, since the filter footprint of a shadow filter can also touch neighboring pixels this may not be enough to construct the most efficient HPSM for quickly rejecting pixels as fully lit or fully shadowed. Figure 2.3 shows that with a naive construction method several texture fetches are necessary to get the data for all HPSM texels that touch the shadow map filter.

The target is to get down to only one texture fetch for quick rejections. One solution to reach that target is to not only look at a 4×4 block of texels but extend this block on all sides (top, bottom, left and right) by half of the size of the shadow filtering kernel as depicted in Figure 2.4.

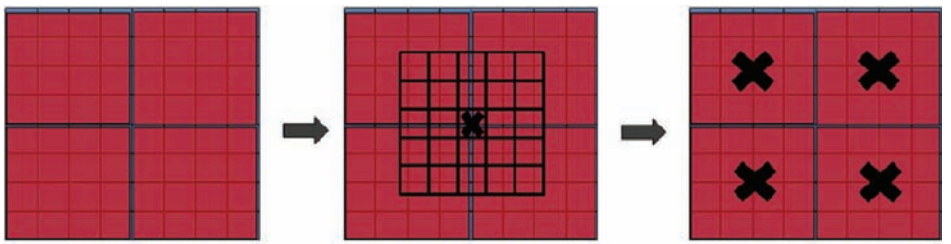

Figure 2.3. A shadow map (left) is converted into an HPSM. The filter footprint of the shadow filter touches more than just one HPSM texel (middle). Instead it can touch four or more neighboring HPSM texels (right).

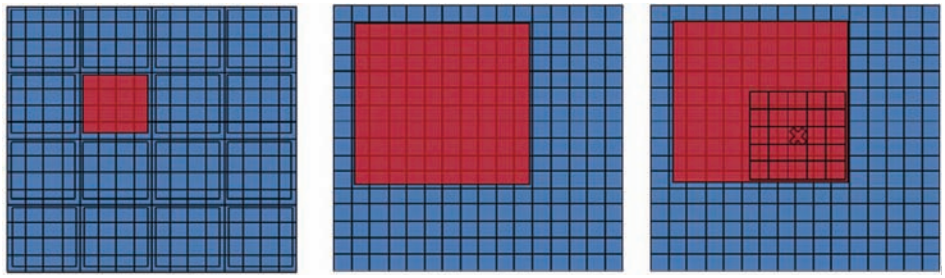

Figure 2.4. One of the HPSM pixels covering 4×4 shadow pixels is highlighted (left). Extending the box from 4×4 to 10×10 (middle) makes sure that the shadow filter footprint always stays inside the support of the HPSM pixel (right).

It is obvious that one needs to consider quite a lot of shadow map texels to create an HPSM that can reject pixels with only one texture instruction. This can make HPSM construction expensive if only Shader Model 4.0 instructions are available. Direct3D 10.1 and Direct3D 11 support the `Gather()` instruction, which can speed up HPSM instruction enormously since one texture instruction can get four depth values from the shadow map. Further, using Compute Shaders under Direct3D 11 allows facilitating the thread shared memory that significantly speeds up HPSM construction for overlapping construction kernels.

Computing min/max depth of a box of pixels is trivial. How about detecting a plane equation? One way to do this is to convert each shadow map depth value of the construction texel block back to linear a three-dimensional space, e.g., light view camera space:

$$\begin{aligned} Q = \frac{Z_f}{Z_f - Z_n}, w = \cot\left(\frac{\text{fov}_w}{2}\right), h = \cot\left(\frac{\text{fov}_h}{2}\right). \\ Z_{\text{cam}} = \frac{-Q \cdot Z_n}{Z_{\text{sm}} - Q}. \end{aligned} \tag{2.1}$$

Equation (2.1) shows how to convert from shadow map depth (Z_{sm}) back to linear light space depth $Z_{cam}$for a perspective light view used to draw the shadow map. Listing 2.1 now presents a shader function that converts a depth value from the shadow map back to a three-dimensional light space point.

Given Listing 2.1, the pixel shader code for creating an HPSM is presented in Listing 2.2. Please note that the length of the three-dimensional vector stored in the *yzw* part of the HPSM encodes if an HPSM pixels stores a plane or just min/max depth.

```
// Convert from camera space depth to light space 3d.
// f2ShadowMapCoord is the shadow map texture coordinate for
// fDepth.
float3 GetCameraXYZFromSMDepth(float fDepth,
                               float2 f2ShadowMapCoord )
{
  float3 f3CameraPos;

  // Compute camera Z: see Equation 2.1.
  f3CameraPos.z = -g_fQTimesZNear / ( fDepth - g_fQ );

  // Convert screen coords to projection space XY.
  f3CameraPos.xy = (f2ShadowMapCoord * g_f2ShadowMapSize ) -
                       float2( 1.0f, 1.0f );

  // Compute camera X.
  f3CameraPos.x = g_fTanH * f3CameraPos.x * f3CameraPos.z;

  // Compute camera Y.
  f3CameraPos.y = - g_fTanV * f3CameraPos.y * f3CameraPos.z;

  return f3CameraPos;
}
```

Listing 2.1. A function to convert from shadow map depth to a linear camera space three-dimensional point.

```
float4 main(float4 pos2d : SV_POSITION ) : SV_Target
{
  float2 tc     = pos2d.xy / (g_f2ShadowMapSize / 4 );
  float4 f4MinD = ( 10000.0).xxxx;
  float4 f4MaxD = (-10000.0).xxxx;
  float  fPlane = 1.0f;

  // Call function to gather four depth values from
  // the shadow map: for a Shader Model > 4.0
  // Gather() can be used; otherwise four point samples
  // need to be used.
  float4 f4D  = gather_depth( tc, int2( 0, 0 ) );
  float3 f3P0 = GetCameraXYZFromSMDepth(f4D.x, tc +
                float2( 0,1 ) * 1.0f / g_f2ShadowMapSize );
  float3 f3P1 = GetCameraXYZFromSMDepth(f4D.y, tc +
                float2( 1,1 ) * 1.0f / g_f2ShadowMapSize );
  float3 f3P2 = GetCameraXYZFromSMDepth(f4D.y, tc +
                float2( 1,0 ) * 1.0f / g_f2ShadowMapSize );

  float3 f3N0 = normalize( fP0 - fP1 );
  float3 f3N1 = normalize( fP0 - fP2 );

  // Construct plane normal at central point.
```

```
  float3 f3N = cross( fN0, fN1 );

  for( int row  = -SHADOW_FILTER_WIDTH/2;
           row  < 4 + SHADOW_FILTER_WIDTH/2;
           row += 2 )
  {
    for( int col  = -SHADOW_FILTER_WIDTH/2;
             col  < 4 + SHADOW_FILTER_WIDTH/2;
             col += 2 )
    {
      // Gather four depth values from the shadow map.
      float4 f4D = gather_depth( tc, int2( row, col ) );

      // Min/max construction
      f4MinD = min( f4D, f4MinD );
      f4MaxD = max( f4D, f4MinD );

      // Look at each cam space point.
      float3 f3P = GetCameraXYZFromSMDepth(f4D.x, tc +
                   ( float2( 0,1 ) * float2 ( row, col ) ) *
                   1.0f / g_f2ShadowMapSize );

      // EPS is the maximum allowed distance from the plane
      // defined by f3P and f3N.
  fPlane *= abs( dot( f3P - f3P0, f3N ) ) < EPS ?
                1.0f : 0.0f;

    }
  }

  // If this is a plane
  if( fPlane != 0.0f )
  {
    // res.x = distance of plane from origin.
    // res.yzw is normalized normal of plane.
    return float4( length( f3P0 ), f3N );
  }
  else
  {
    // Make sure that length(yzw) is bigger than 1
    return float4( min( min( f4MinD.x, f4MinD.y ),
                        min( f4MinD.z, f4MinD.w ) ),
                   max( max( f4MaxD.x, f4MaxD.y ),
                        max( f4MaxD.z, f4MaxD.w ) ),
                   100.0f, 100.0f );
  }
}
```

Listing 2.2. A pixel shader that constructs an HPSM from a normal shadow map.

2.4 Using an HPSM

Having constructed an HPSM, using it is straightforward and demonstrated by the shader snippet in Listing 2.3.

```
// LSP is the light space position of the current pixel;
// tc.xy is the shadow map texture coordinate for the current
// pixel; tc.z is light space depth of the pixel.
// It is assumed that any necessary depth bias (e.g., to
// deal with the EPS for plane construction) has already been
// add to tc.z.
float shadow(float3 LSP, float3 tc, inout float fLight )
{
  float4 f4HPMS        = g_txHPSM.SampleLevel( s_point, tc.xy,
                                               0 );
  float  fLenSqrNormal = dot( f4HPMS.yzw, f4HPMS.yzw );
  float  fReject = false;

  // Min/max
  if( fLenSqrNormal > 1.1 )
  {
     float fMin = f4HPMS.x;
     float fMax = f4HPMS.y;

     if( tc.z < fMin )
        fLight = 1.0f;
     else if( tc.z > fMax )
        fLight = 0.0f;
     else // call expensive filter
        fLight = filter_shadow( tc );
  }
  else // Plane
  {
     float3 f3P  = f4HPMS.x * f4HPMS.yzw;
     float fDist = dot( f3P - LSP, f4HPMS.yzw );

     if( fDist <= -EPS )
        fLight = 0.0f;
     else // Full light
        fLight = 1.0f;
  }
}
```

Listing 2.3. A pixel shader that uses the HPSM to reject pixels that are in full light or in full shadow.

2.5 Other Uses for the HPSM

As pointed out in [Lobanchikov et al. 09], HPSM can be used to accelerate all sort of shadow map queries. [Lobanchikov et al. 09] uses a simple form HPSMs to accelerate not only normal shadow filtering but also the rendering of sun shafts. Basically, the shader in question integrates light along a ray from the scene towards the eye. For each point on the ray four PCF shadow map samples are necessary to generate smooth looking sun shafts. Using an HPSM to quickly reject points on a sun shaft ray generates a speedup of $\sim$12% at a resolution of 1600×1200 on an AMD HD4870 GPU versus doing all four PCF samples for every point on the ray.

Bibliography

[Dmitriev et al. 07] Kirill Dmitriev and Yury Uralsky. "Soft Shadows Using Hierarchical Min-Max Shadow Maps." Presented at Game Developers Conference, San Francisco, March 5–9, 2007. Available at http://developer.download.nvidia.com/presentations/2007/gdc/SoftShadows.pdf

[Lobanchikov et al. 09] I. Lobanchikov and H. Gruen, "Stalker: Clear Sky—A Showcase for Direct3D 10.0/1." Presented at Game Developers Conference, San Francisco, March 23–27, 2009. Available at http://www.gdconf.com/conference/Tutorial.