
MARCH 5-9
2007
SAN FRANCISCO
MOSCONE CENTER
TAKE CONTROL
www.gdconf.com
CMP

NVIDIA DX10 SDK: SoftShadows

Soft shadows using hierarchical min-max shadow maps

Kirill Dmitriev
Yury Uralsky
NVIDIA

Overview

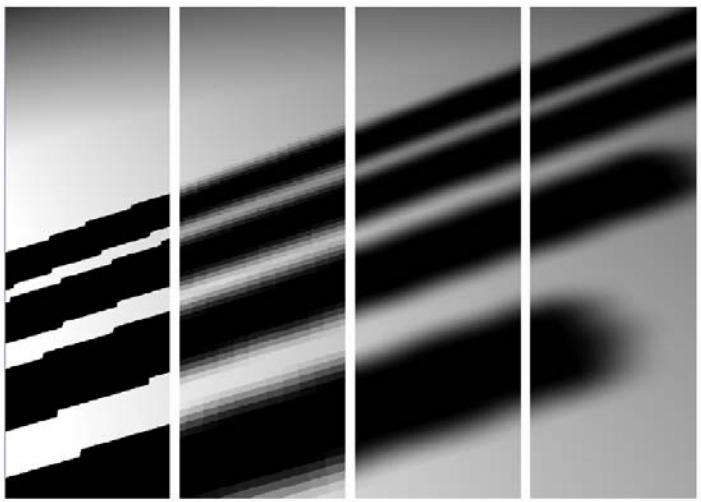
- Traditional algorithms for soft shadows
- Min-max depth mipmap data structure
- Large kernel PCF with min-max mipmap
- Physically plausible soft shadows with min-max mipmap
- Ideas for improving performance

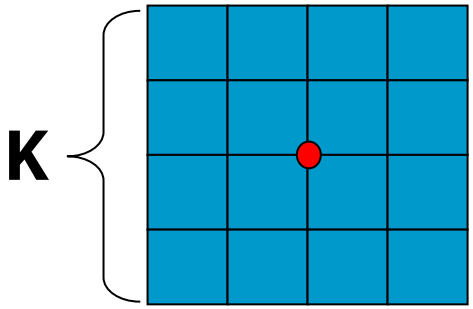
Soft shadows are important

- Important for realism
  - Hard shadows can be cast only by lightsources with zero size, which do not exist in reality
- Hide artifacts that occur due to insufficient shadow map resolution

How do we make soft shadows?

- PCF (Percentage Closer Filtering)
  - Take a number of samples around the shaded fragment. Compute average.
- Kernel size (K) controls “softening”





Percentage closer filtering (PCF)

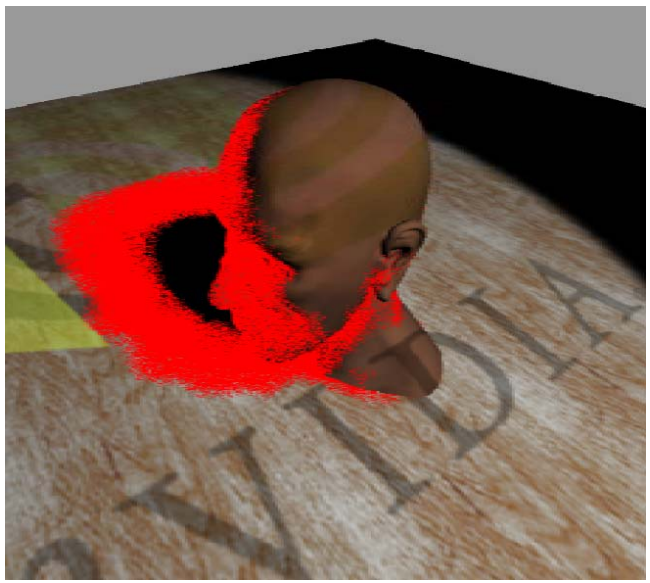
- Important: averaging depth doesn't work, instead:

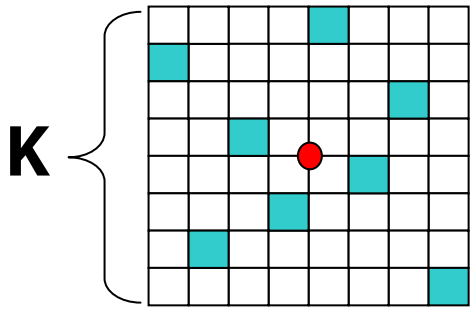
$$S(\mathit{fragment}) = \frac{\sum_{p \in K} \mathit{Depth}(p) < \mathit{Depth}(\mathit{fragment})}{N(K)}$$

- NVIDIA hardware implements it for small K

Jittered PCF

- PCF with large K requires many samples
- Can use jittered sampling
  - Trades banding for noise





Variance shadow maps

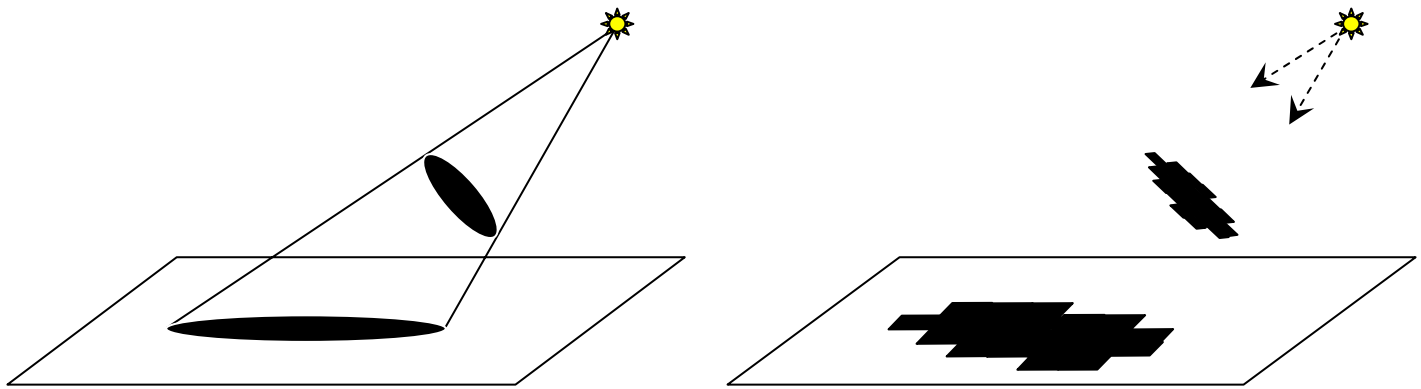
- Store depth AND square of depth
  - Standard deviation of depth can be computed
  - Use Chebyshev's inequality to compute shadowing
- Allow pre-filtering (mipmap)

Problems with current approaches

- Percentage closer filtering
  - Requires a lot of samples for quality
- Variance shadow maps
  - Filter kernel size is fixed

Alternative idea

- Shadowmap can be decomposed into a set of quads floating in 3D space
  - We can compute shadowing from each of the quads individually and then sum up their contributions

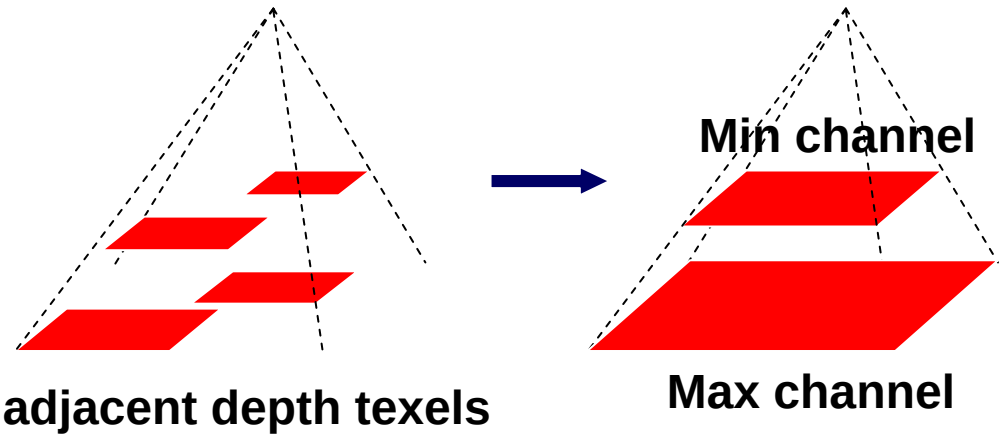


Min-max mipmap shadowmap

- Linear walk over shadowmap texels is wasteful
- Can represent shadowmap data in hierarchical fashion
  - Hierarchical descend allows for efficient pruning of subtrees

Min-max mipmap shadowmap

- Calculate two mipmap pyramids
  - Using **min filter** for construction
  - Using **max filter** for construction
- Can be stored in two-channel texture format



Large kernel PCF with min-max mipmap shadowmap

```
start from the 2x2 miplevel
S = 0; // initialize shadowing
for ( i = 0; ; ) // for each node at current level
{
   if ( ++i == 4 ) {
        { pop mip level and i from stack; continue; }

   if ( DepthMin > FragmentDepth )
        continue; // skip the subtree
   if ( DepthMax <= FragmentDepth || mip == 0 ) {
        S += Overlap(K, CT);
        continue; // skip the subtree
   }
   store i and current mip onto stack;
   go to finer mip;
   i = 0; // start from texel 0 on the new mip level
}
```

Large kernel PCF with min-max mipmap shadowmap

```
start from the 2x2 miplevel
S = 0; // initialize shadowing
for ( i = 0; ; ) // for each node at current level
{
   if ( ++i == 4 ) {
        { pop mip level and i from stack; continue; }

   if ( DepthMin > FragmentDepth )
        continue; // skip the subtree
   if ( DepthMax <= FragmentDepth || mip == 0 ) {
        S += Overlap(K, CT);
        continue; // skip the subtree
   }
   store i and current mip onto stack;
   go to finer mip;
   i = 0; // start from texel 0 on the new mip level
}
```

Large kernel PCF with min-max mipmap shadowmap

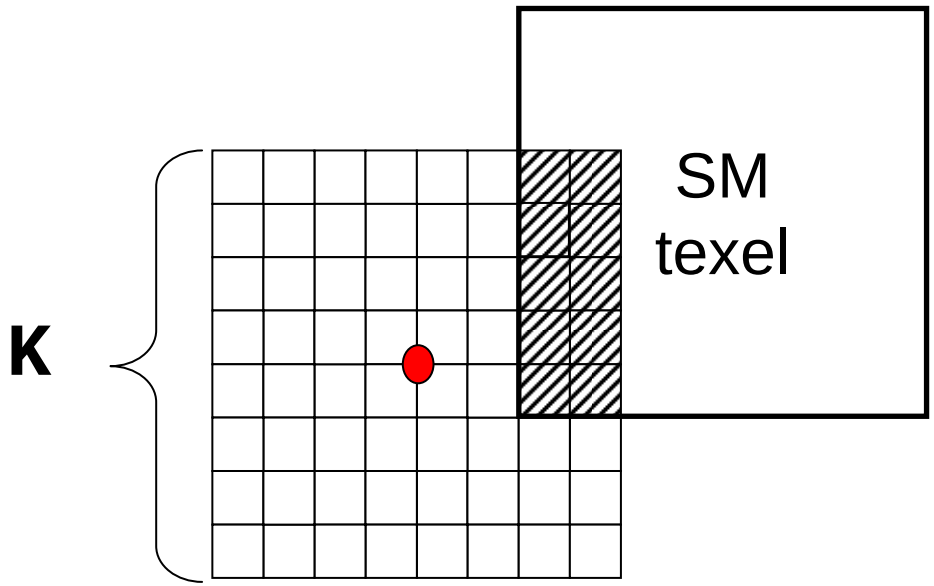
```
start from the 2x2 miplevel
S = 0; // initialize shadowing
for ( i = 0; ; ) // for each node at current level
{
   if ( ++i == 4 ) {
        { pop mip level and i from stack; continue; }

   if ( DepthMin > FragmentDepth )
        continue; // skip the subtree
   if ( DepthMax <= FragmentDepth || mip == 0 ) {
        S += Overlap(K, CT);
        continue; // skip the subtree
   }
   store i and current mip onto stack;
   go to finer mip;
   i = 0; // start from texel 0 on the new mip level
}
```

Large kernel PCF with min-max mipmap shadowmap

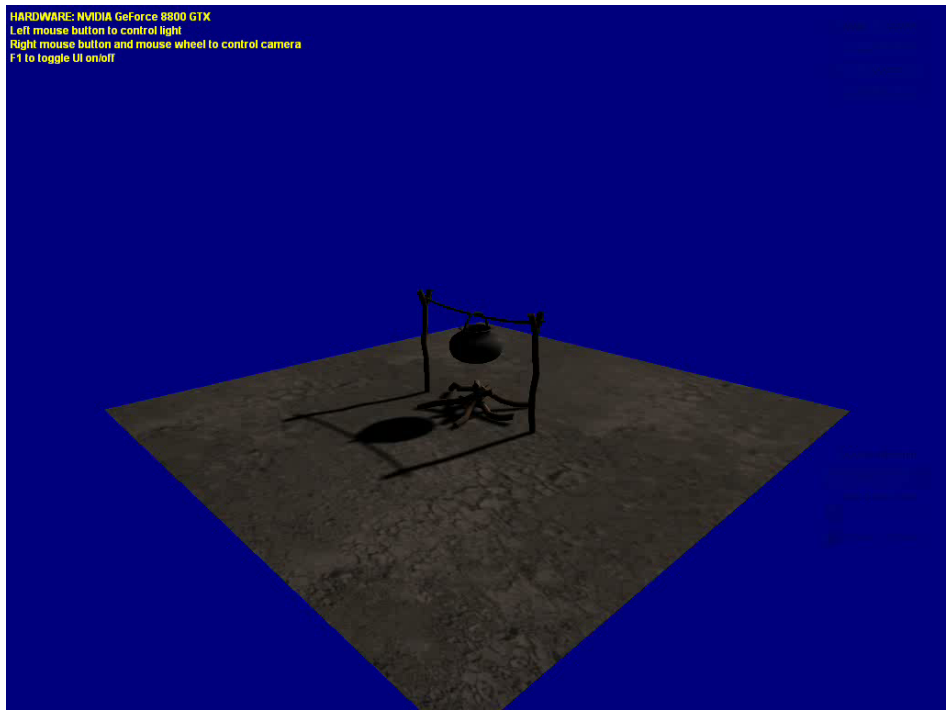
```
start from the 2x2 miplevel
S = 0; // initialize shadowing
for ( i = 0; ; ) // for each node at current level
{
   if ( ++i == 4 ) {
        { pop mip level and i from stack; continue; }

   if ( DepthMin > FragmentDepth )
        continue; // skip the subtree
   if ( DepthMax <= FragmentDepth || mip == 0 ) {
        S += Overlap(K, CT);
        continue; // skip the subtree
   }
   store i and current mip onto stack;
   go to finer mip;
   i = 0; // start from texel 0 on the new mip level
}
```

Overlap() function

- Shadowing is proportional to the amount of overlap between current SM texel and the filter kernel

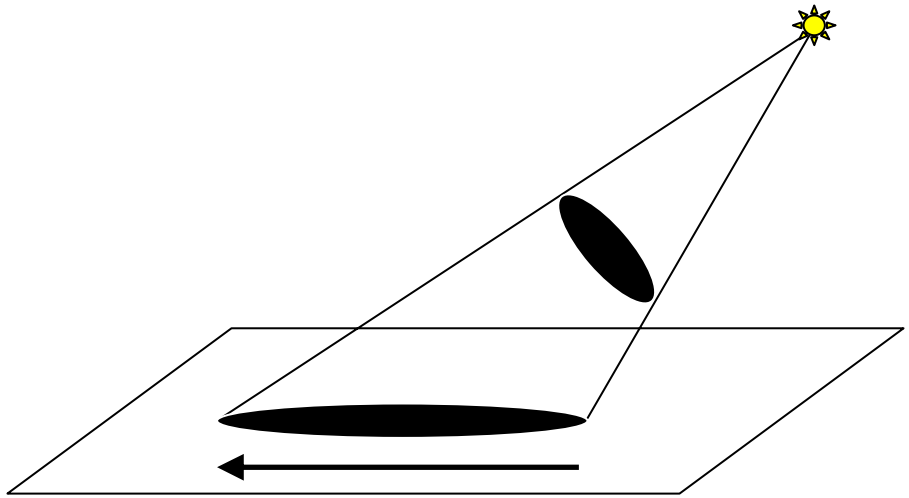


Soft shadows using hierarchical min-max shadowmap



Towards adaptive softening

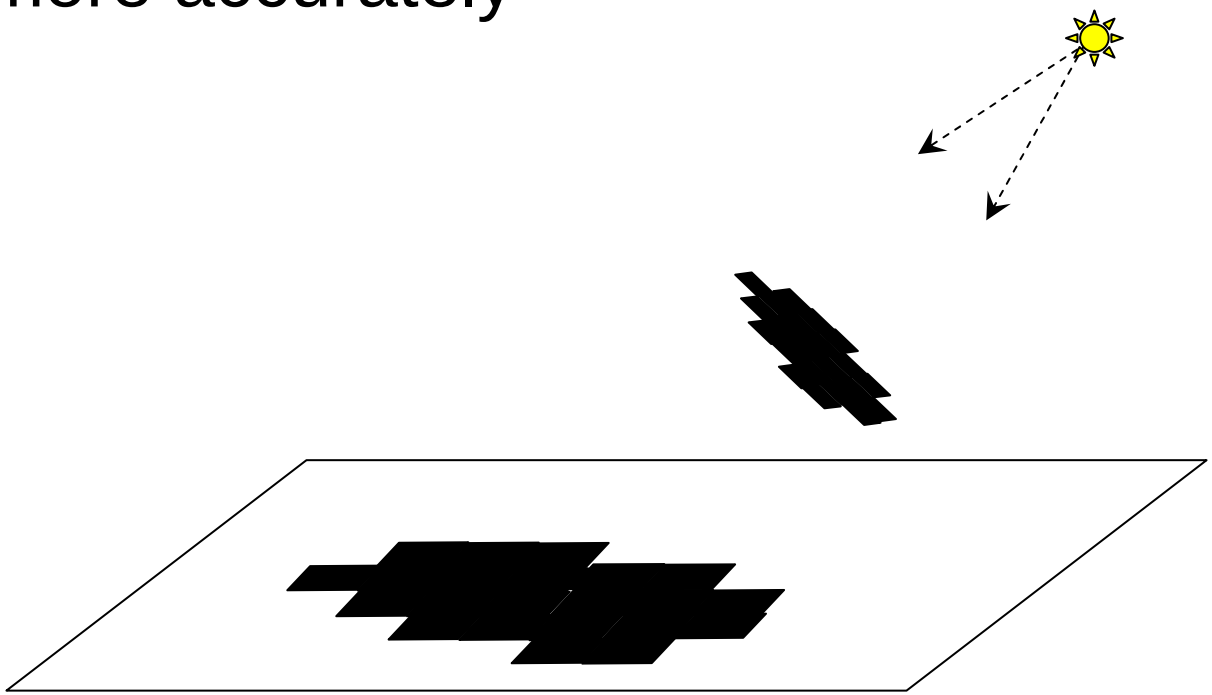
- The “softness” should depend on relative distances b/w current fragment, lightsource and the occluder



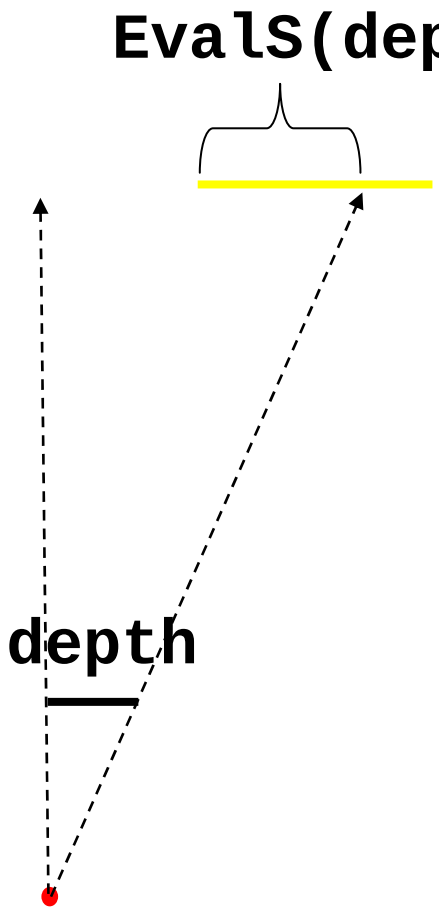
Softening must increase

Physically plausible soft shadows

- Need to compute the overlap function more accurately



Shadowing computation for physically plausible shadows



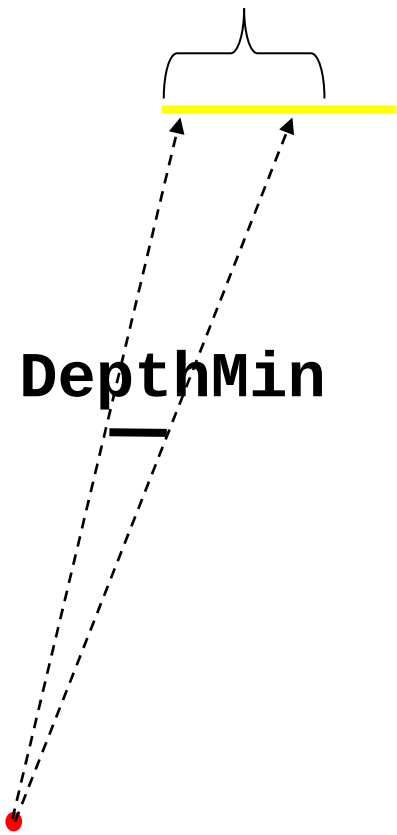
$$S(\mathit{fragment}) = \frac{\sum \mathit{EvalS}(\mathit{depth})}{\mathit{Area}}$$

The above sum goes over all shadowmap texels at level 0.

Shadow pixels are AABBs,

light is AABB: geometry is very simple

Majority of shadow map texels do not contribute




```
Start from coarse miplevel;

if (EvalS(DepthMin) == 0)
{
  prune the subtree;
}
```

Throws away (2^N)^2 pixels
where N is the current mip-level

Physically plausible algorithm

```
begin from the 2x2 mip;
S = 0; // initialize shadowing
for (i = 0; ; ) // for each of node at current level
{
   if ( ++i == 4 )
        { pop mip level and i from stack; continue; }

   if ( EvalS(DepthMin) == 0 )
        continue;
   if (EvalS(DepthMin) == 1 && DepthMax <= FragmentDepth)
        return 0; // fully in shadow
   if ( mip == 0 ) {
        S += EvalS( K, DepthMin );
        continue;
   }
   store i and current mip level onto stack;
   go to finer mip;
   i = 0; // start over from texel 0 on the new mip level
}
```

Physically plausible algorithm

```
begin from the 2x2 mip;
S = 0; // initialize shadowing
for (i = 0; ; ) // for each of node at current level
{
   if ( ++i == 4 )
        { pop mip level and i from stack; continue; }

   if ( EvalS(DepthMin) == 0 )
        continue;
   if (EvalS(DepthMin) == 1 && DepthMax <= FragmentDepth)
        return 0; // fully in shadow
   if ( mip == 0 ) {
        S += EvalS( K, DepthMin );
        continue;
   }
   store i and current mip level onto stack;
   go to finer mip;
   i = 0; // start over from texel 0 on the new mip level
}
```

Physically plausible algorithm

```
begin from the 2x2 mip;
S = 0; // initialize shadowing
for (i = 0; ; ) // for each of node at current level
{
   if ( ++i == 4 )
        { pop mip level and i from stack; continue; }

   if ( EvalS(DepthMin) == 0 )
        continue;
   if (EvalS(DepthMin) == 1 && DepthMax <= FragmentDepth)
        return 0; // fully in shadow
   if ( mip == 0 ) {
        S += EvalS( K, DepthMin );
        continue;
   }
   store i and current mip level onto stack;
   go to finer mip;
   i = 0; // start over from texel 0 on the new mip level
}
```

Physically plausible algorithm

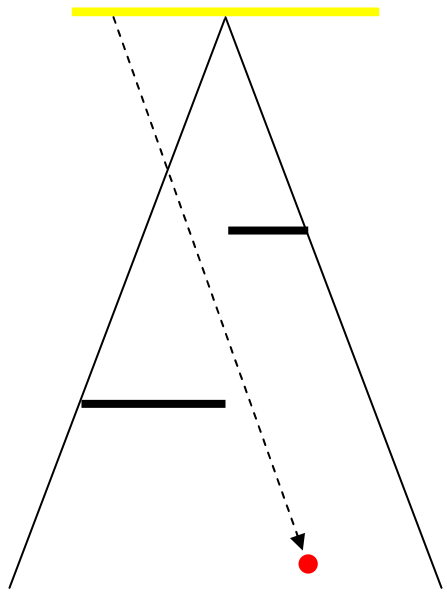
```
begin from the 2x2 mip;
S = 0; // initialize shadowing
for (i = 0; ; ) // for each of node at current level
{
   if ( ++i == 4 )
        { pop mip level and i from stack; continue; }

   if ( EvalS(DepthMin) == 0 )
        continue;
   if (EvalS(DepthMin) == 1 && DepthMax <= FragmentDepth)
        return 0; // fully in shadow
   if ( mip == 0 ) {
        S += EvalS( K, DepthMin );
        continue;
   }
   store i and current mip level onto stack;
   go to finer mip;
   i = 0; // start over from texel 0 on the new mip level
}
```

Physically plausible algorithm

```
begin from the 2x2 mip;
S = 0; // initialize shadowing
for (i = 0; ; ) // for each of node at current level
{
   if ( ++i == 4 )
        { pop mip level and i from stack; continue; }

   if ( EvalS(DepthMin) == 0 )
        continue;
   if (EvalS(DepthMin) == 1 && DepthMax <= FragmentDepth)
        return 0; // fully in shadow
   if ( mip == 0 ) {
        S += EvalS( K, DepthMin );
        continue;
   }
   store i and current mip level onto stack;
   go to finer mip;
   i = 0; // start over from texel 0 on the new mip level
}
```

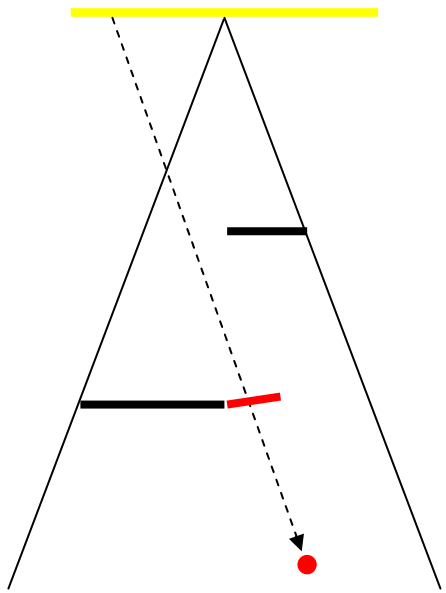
Light leaks

- Since shadow map is constructed for point light, but used for area light, shadow leaks are possible



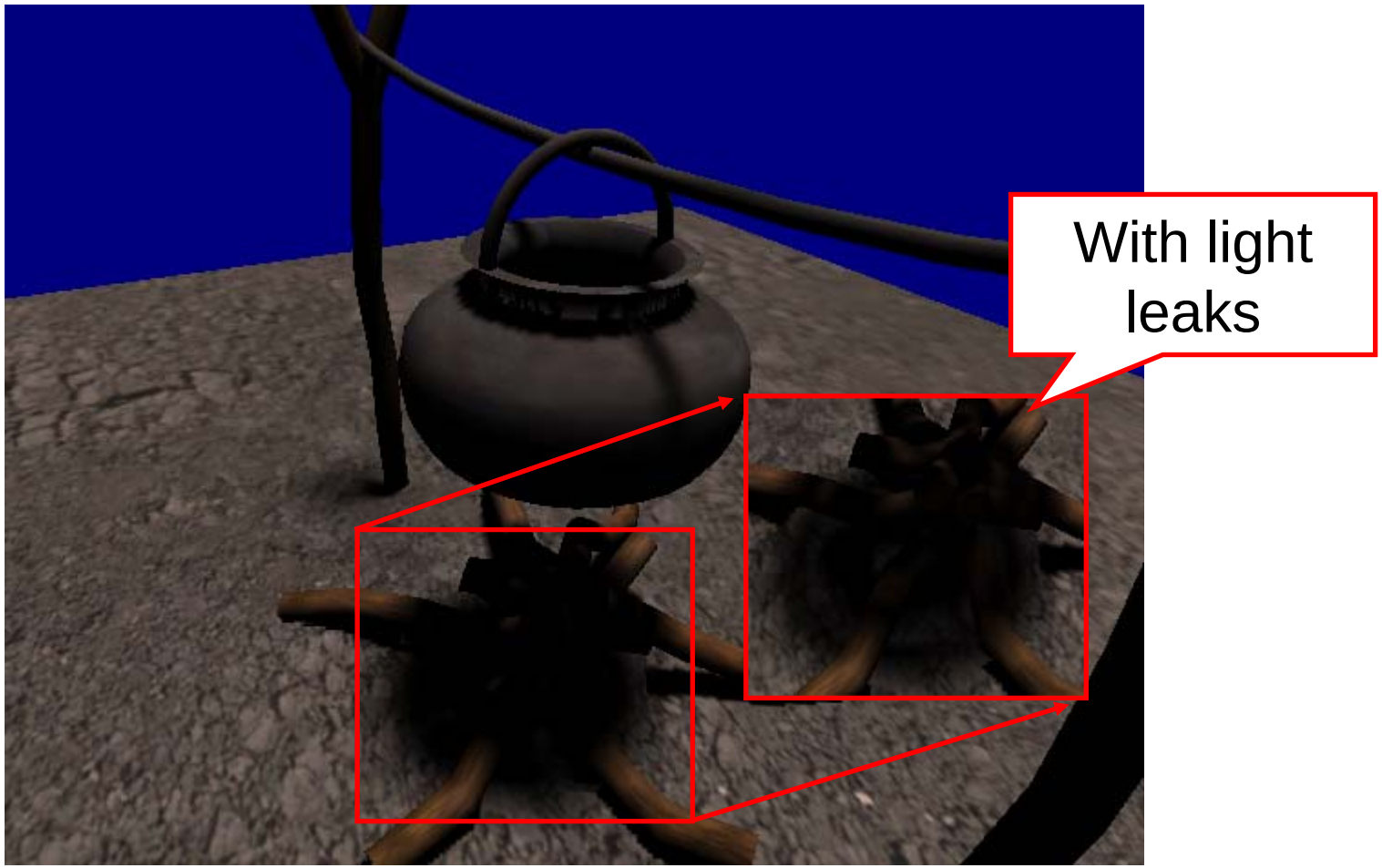
Light leaks example

Removing light leaks

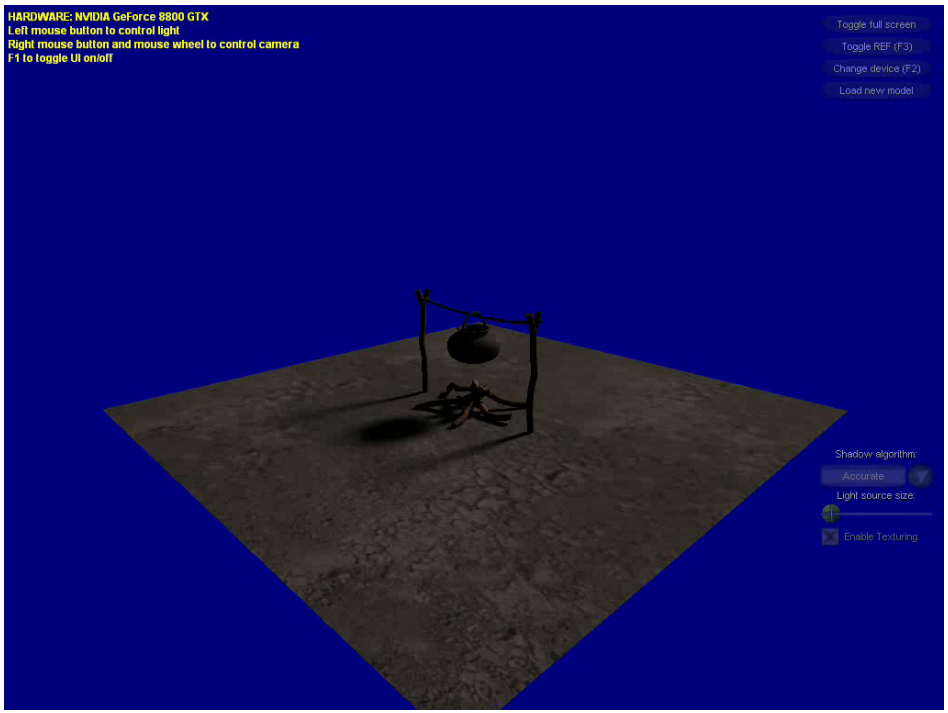


- Remove by artificially extending shadow map texels
  - Texels with larger depth are extended to match borders of neighbors with smaller depth

Removing light leaks example



Physically plausible soft shadows



Efficient stack implementation

- Need to be careful with dynamic indexing
- Need to push/pop only (0 <= i <= 3) and mip-level: 2 + 4 bits

```
uint iHigh, iLow;
Push(uint bits) {
   iHigh = (iHigh >> 6) | (iLow & 0xfc000000);
   iLow = (iLow << 6) | bits;
}
uint Pop() {
   uint bits = iLow & 0x3f;
   iLow = (iHigh & 0xfc000000) | (iLow >> 6);
   iHigh <<= 6;
   return bits;
}
```

Ideas for improving performance

- Can stop traversing the hierarchy at higher levels
  - Can be used to implement shadow LOD
- Compute shadowing on sparse grid (e.g. 4x in each dimension)
  - In-between pixels can be interpolated/computed based on…
  - Need to come up with a good heuristic!

Interpolation heuristic

- Can’t interpolate if highly non-planar surface
- Can’t interpolate if shadowing changes drastically inside the grid cell
- Shadowing may change drastically if there are shadow casters that are quite close

Proposed heuristic

- For every grid node store distance to closest important occluder (one float per node);
- Can interpolate only if sample is inside the box created by 4 cell corners AND distance to closest important occluder is large compared to the box size;

References

[1] See NVIDIA SDK10 “SoftShadows” demo for details

[2] Gael Guennebaud, Loic Barthe and Mathias Paulin. “Real-time Soft Shadow Mapping by Backprojection”. Eurographics Symposium on Rendering 2006, Nicosia, Cyprus.

Questions?

- kdmitriev@nvidia.com
- yuralsky@nvidia.com