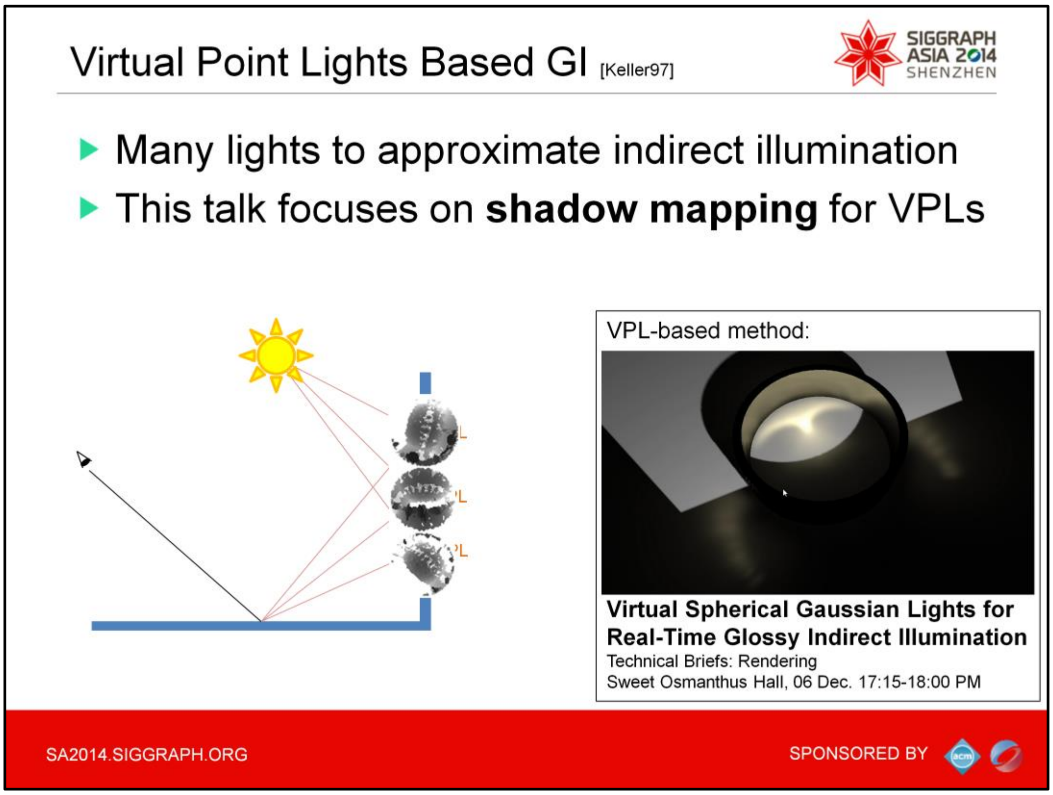
Point Cloud Culling for Imperfect Shadow Maps

Yusuke Tokuyoshi
Square Enix Co., Ltd.

SIGGRAPH ASIA 2014 SHENZHEN

SA2014.SIGGRAPH.ORG

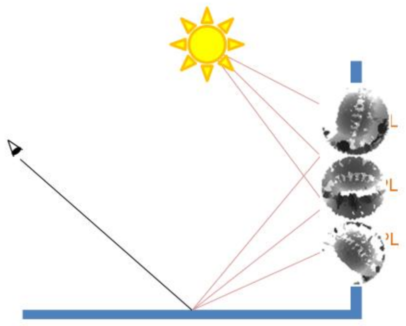
SPONSORED BY



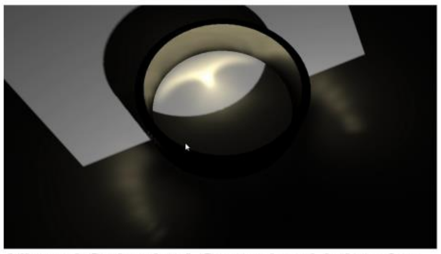
Virtual Point Lights Based GI [Keller97]

SIGGRAPH ASIA 2014 SHENZHEN

- Many lights to approximate indirect illumination
- This talk focuses on **shadow mapping** for VPLs



VPL-based method:

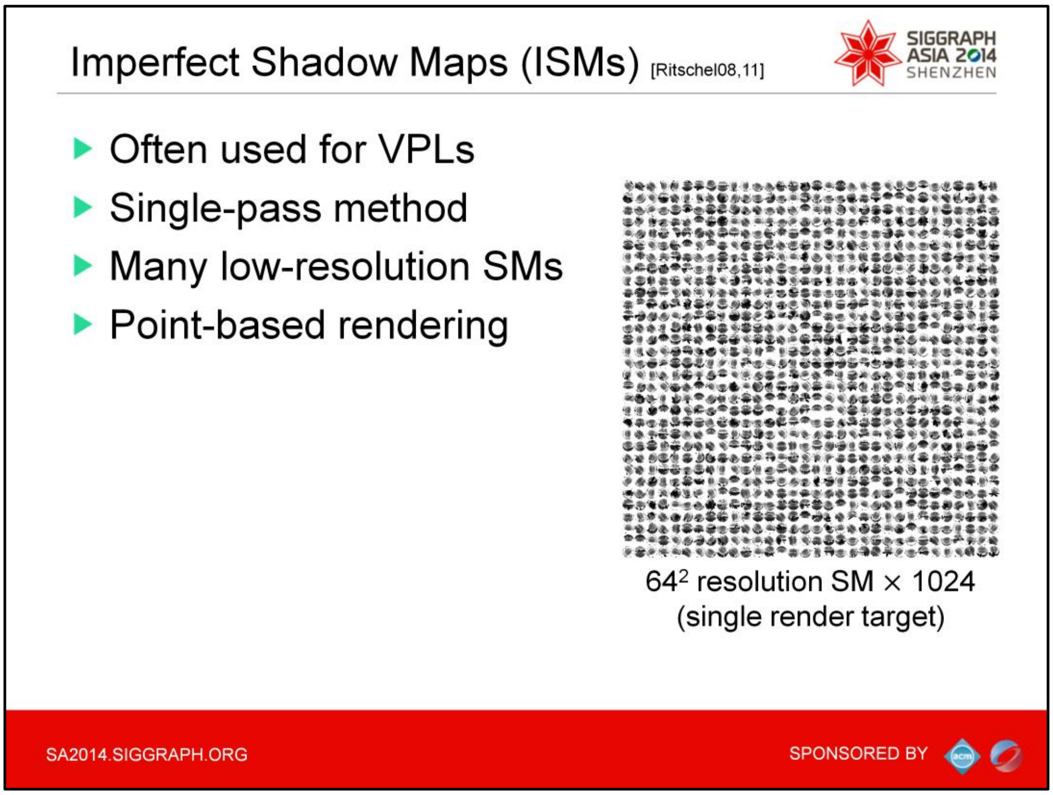


Virtual Spherical Gaussian Lights for Real-Time Glossy Indirect Illumination
Technical Briefs: Rendering
Sweet Osmanthus Hall, 06 Dec. 17:15-18:00 PM

SA2014.SIGGRAPH.ORG SPONSORED BY

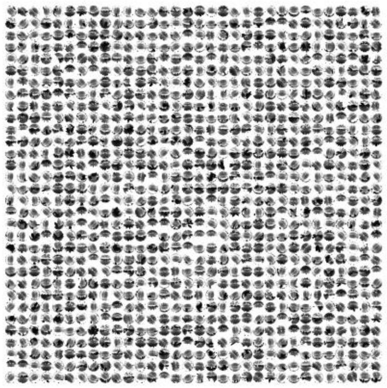
Virtual point lights are many light sources used for approximating indirect illumination.
For interactive or real-time applications, a lot of VPL-based methods have been developed.
We’ll also present a new technique entitled “Virtual Spherical Gaussian Lights for Real-Time Glossy Indirect Illumination” in the Technical Brief session.
For the detail of this method, please come to the technical brief session.

In this course, we’d like to focus on shadow mapping for each VPL.
This shadow mapping can be a bottleneck for VPL-based real-time global illumination.



Imperfect Shadow Maps (ISMs) [Ritschel08,11]

- Often used for VPLs
- Single-pass method
- Many low-resolution SMs
- Point-based rendering

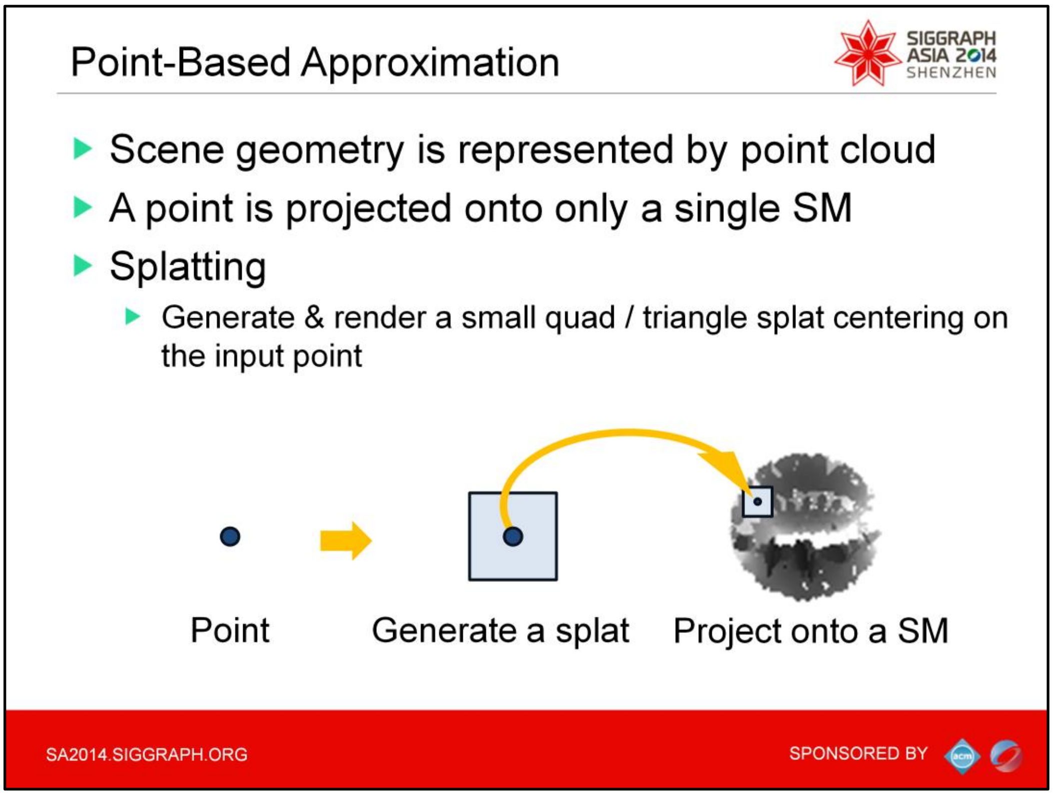


64^2 resolution SM × 1024
(single render target)

SA2014.SIGGRAPH.ORG

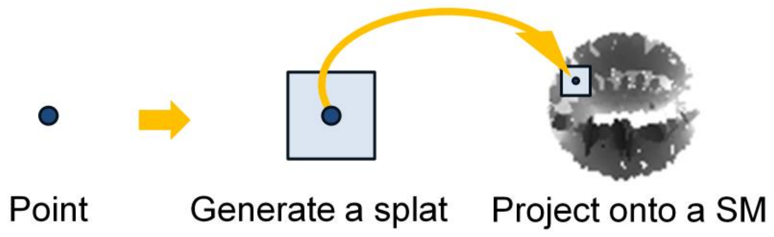
SPONSORED BY

Imperfect shadow maps are often used for VPLs.
This is a single-pass method to render an arbitrary number of shadow maps which are tiled in a single render target.
This shadow mapping is done by point-based rendering.



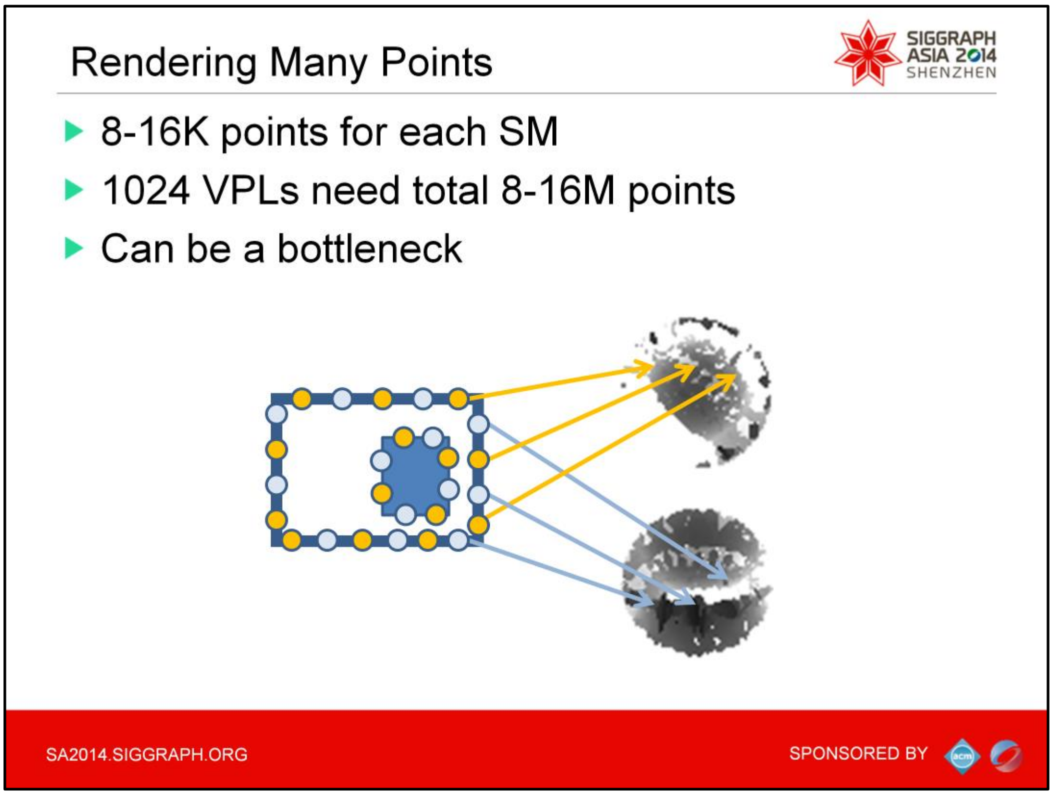
Point-Based Approximation

- Scene geometry is represented by point cloud
- A point is projected onto only a single SM
- Splatting
  - Generate & render a small quad / triangle splat centering on the input point



SA2014.SIGGRAPH.ORG SPONSORED BY

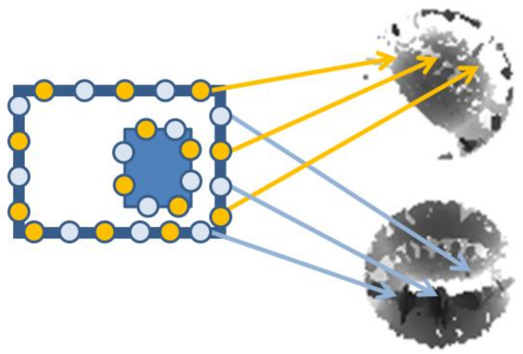
To generate imperfect shadow maps, the scene geometry is first approximated by a point cloud.
Instead of polygons, we render this point cloud.
Similar to particle rendering, the points are rendered using splatting.
This is done by generating a small quad or triangle splat centered on each input point, and then this splat is projected onto a single shadow map.
This splatting can be done with a single pass.
The computation time depends on the total number of points.



Rendering Many Points

SIGGRAPH ASIA 2014 SHENZHEN

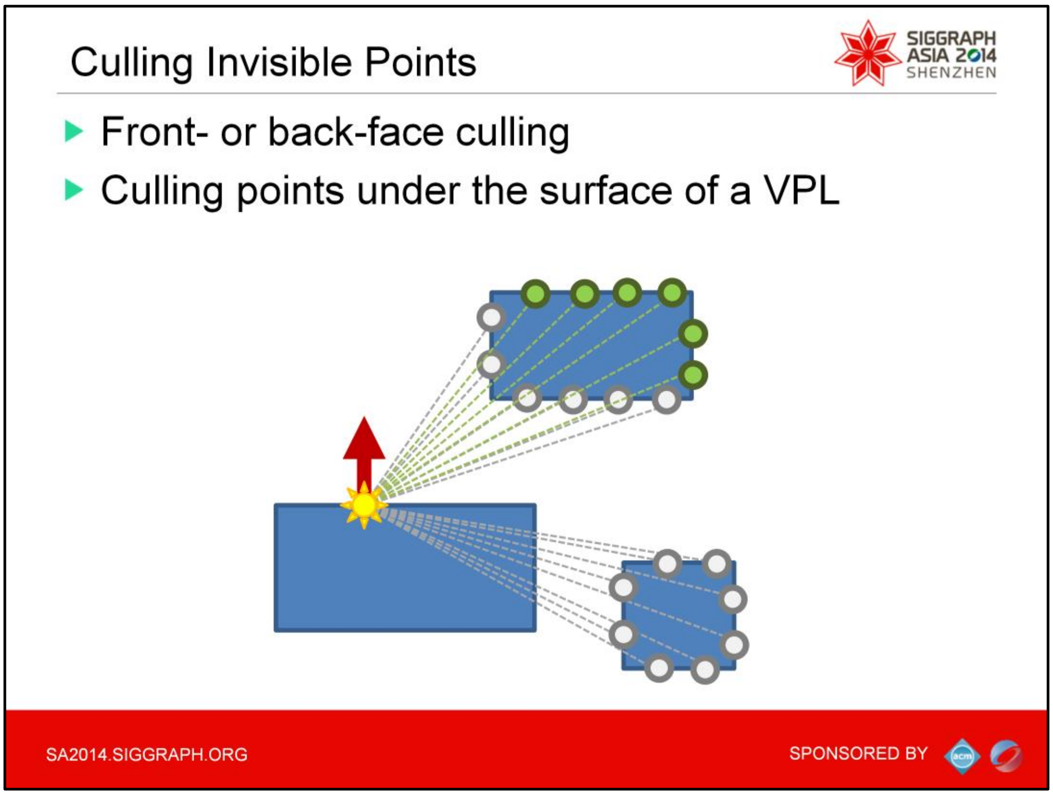
- 8-16K points for each SM
- 1024 VPLs need total 8-16M points
- Can be a bottleneck



SA2014.SIGGRAPH.ORG

SPONSORED BY

In order to generate imperfect shadow maps for many lights, we have to render many points.
Generally, 8 to 16 K points should be used for each VPL.
For 1 K VPLs, this means a total of 8 to 16 M points.
Splatting these many points can be a bottleneck for global illumination rendering.



Culling Invisible Points

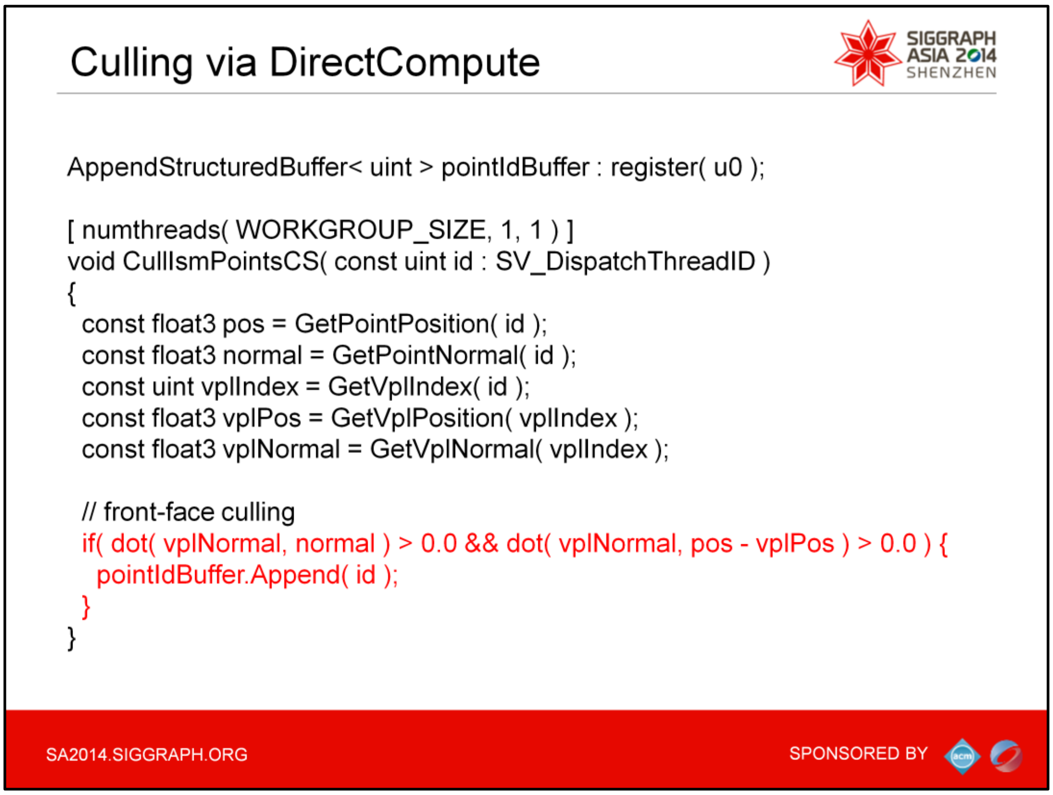
- Front- or back-face culling
- Culling points under the surface of a VPL

SA2014.SIGGRAPH.ORG

SPONSORED BY

Therefore, culling invisible points before the rendering is very effective.
In this figure, only green points are visible to a VPL.
Gray points are invisible.

All points are generated on surfaces, and each point can have a surface normal.
Thus, similar to conventional shadow mapping, we can use front- or back-face culling for each point.
In addition, points under the surface of a VPL are also invisible.
These points should be culled before the splatting.
This culling can be implemented using a compute shader using an AppendStructuredBuffer.



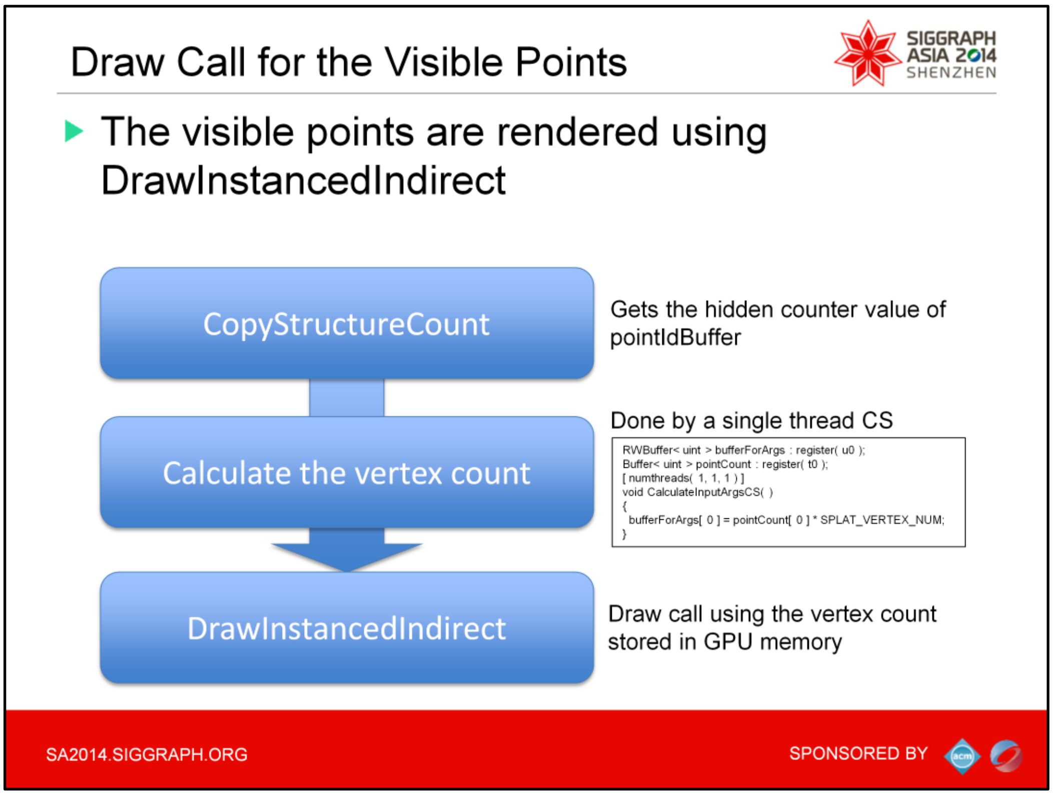
Culling via DirectCompute

```
AppendStructuredBuffer< uint > pointIdBuffer : register( u0 );

[ numthreads( WORKGROUP_SIZE, 1, 1 ) ]
void CullIsmPointsCS( const uint id : SV_DispatchThreadID )
{
  const float3 pos = GetPointPosition( id );
  const float3 normal = GetPointNormal( id );
  const uint vplIndex = GetVplIndex( id );
  const float3 vplPos = GetVplPosition( vplIndex );
  const float3 vplNormal = GetVplNormal( vplIndex );

  // front-face culling
  if( dot( vplNormal, normal ) > 0.0 && dot( vplNormal, pos - vplPos ) > 0.0 ) {
    pointIdBuffer.Append( id );
  }
}
```

SA2014.SIGGRAPH.ORG

SPONSORED BY

This is the compute shader.
As you can see from the code, it is quite simple.
Please note the red lines.
If a point is visible, it is appended into the pointIdBuffer which is an AppendStructuredBuffer.
By using this compute shader, points are culled in parallel on the GPU.



Draw Call for the Visible Points

SIGGRAPH ASIA 2014 SHENZHEN

- The visible points are rendered using DrawInstancedIndirect

CopyStructureCount
Calculate the vertex count
DrawInstancedIndirect

Gets the hidden counter value of pointIdBuffer

Done by a single thread CS

```
RWBuffer< uint > bufferForArgs : register( u0 );
Buffer< uint > pointCount : register( t0 );
[ numthreads( 1, 1, 1 ) ]
void CalculateInputArgsCS( )
{
 bufferForArgs[ 0 ] = pointCount[ 0 ] * SPLAT_VERTEX_NUM;
}
```

Draw call using the vertex count stored in GPU memory

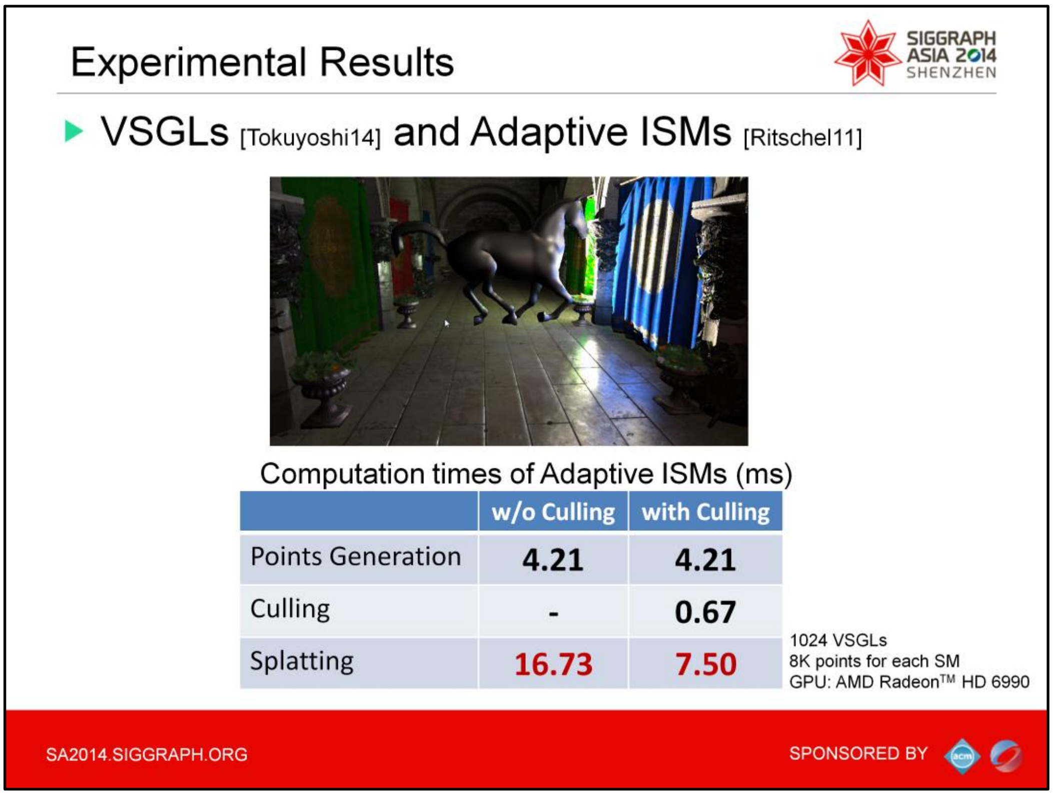
SA2014.SIGGRAPH.ORG

SPONSORED BY

After the culling, only the visible points stored in the pointIdBuffer are rendered.
This is done by DrawInstancedIndrect for triangle splats.

In order to obtain the vertex count for the input argument of this draw call, CopyStructureCount and a compute shader are used.
CopyStructureCount is used for getting the hidden counter value of the pointIdBuffer which is an AppendStructuredBuffer.
After that, the total vertex count of all the splats are calculated using a single thread compute shader.
For triangle splats, this is calculated by multiplying the counter value by three.

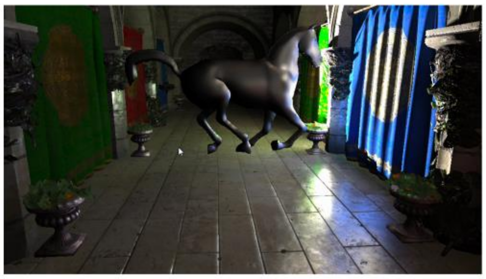
Finally, DrawInstancedIndirect is called using the input arguments stored in the device memory.



Experimental Results

SIGGRAPH ASIA 2014 SHENZHEN

- VSGLs [Tokuyoshi14] and Adaptive ISMs [Ritschel11]



Computation times of Adaptive ISMs (ms)

	w/o Culling	with Culling
Points Generation	**4.21**	**4.21**
Culling	**-**	**0.67**
Splatting	**16.73**	**7.50**

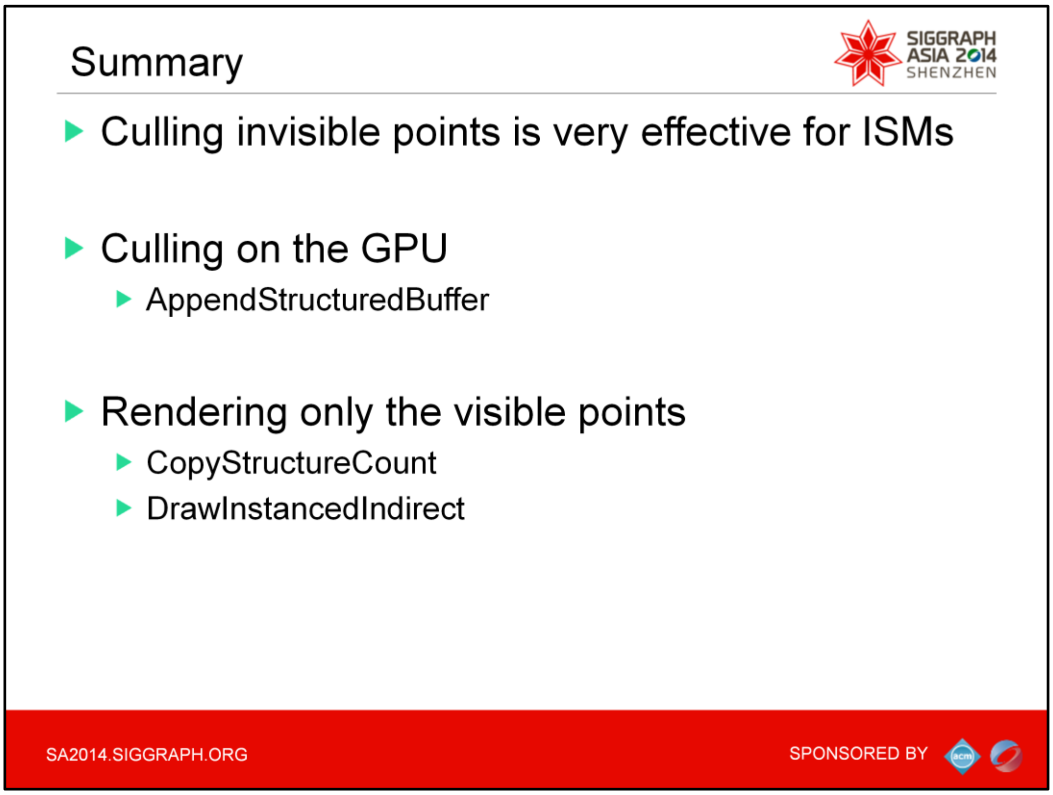
1024 VSGLs
8K points for each SM
GPU: AMD Radeon^{TM} HD 6990

SA2014.SIGGRAPH.ORG

SPONSORED BY

So, let me show you an experimental result of the point cloud culling.
In this experiment, 1 K virtual spherical Gaussian lights and adaptive imperfect shadow maps are employed.
For each shadow map, we use 8 K points.

The computation time of adaptive points generation for this scene is about 4 milliseconds.
The point splatting time without culling is about 17 milliseconds.
On the other hand, the point splatting time with culling is 7.5 milliseconds.
The overhead of the culling is only 0.67 milliseconds.
So, the splatting pass can be twice as fast by using point cloud culling.


Summary

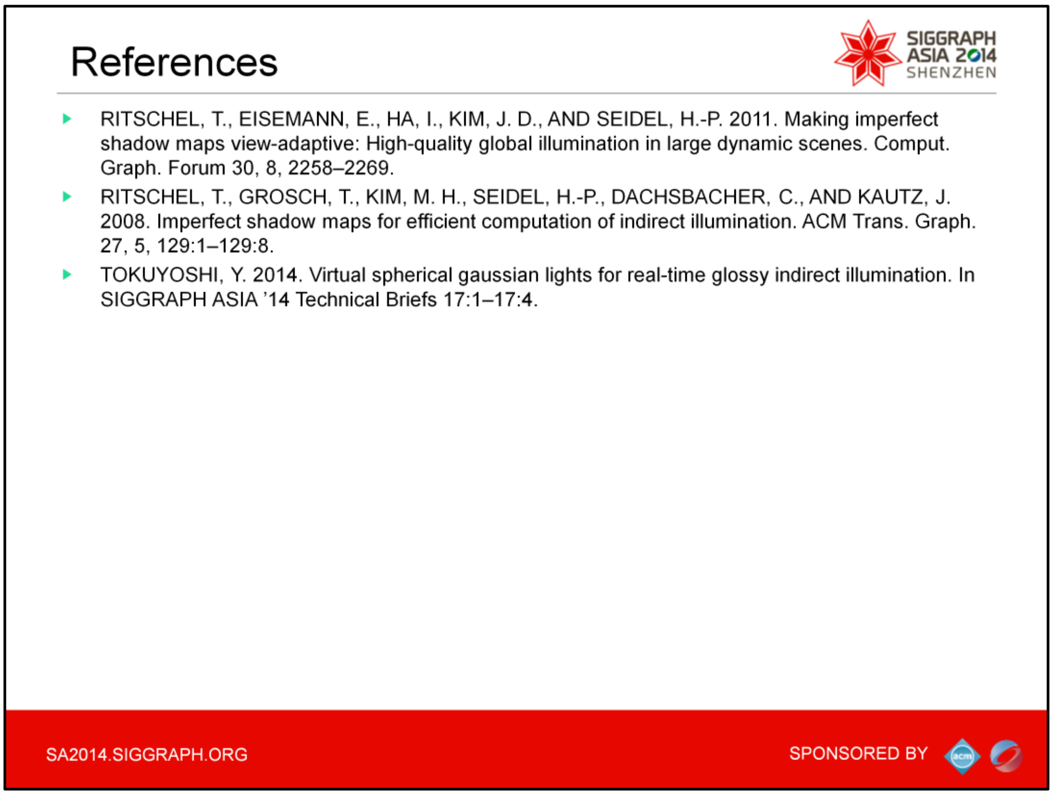
SIGGRAPH ASIA 2014 SHENZHEN

- Culling invisible points is very effective for ISMs
- Culling on the GPU
  - AppendStructuredBuffer
- Rendering only the visible points
  - CopyStructureCount
  - DrawInstancedIndirect

SA2014.SIGGRAPH.ORG

SPONSORED BY

To conclude, culling invisible points for imperfect shadow maps is very effective. This culling can be performed on the GPU by using an AppendStructuredBuffer. Then, we render only the visible points for imperfect shadow maps using CopyStructureCount and DrawInstancedIndirect.



References

- RITSCHEL, T., EISEMANN, E., HA, I., KIM, J. D., AND SEIDEL, H.-P. 2011. Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. Comput. Graph. Forum 30, 8, 2258–2269.
- RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. ACM Trans. Graph. 27, 5, 129:1–129:8.
- TOKUYOSHI, Y. 2014. Virtual spherical gaussian lights for real-time glossy indirect illumination. In SIGGRAPH ASIA '14 Technical Briefs 17:1–17:4.

SA2014.SIGGRAPH.ORG SPONSORED BY