Sample Distribution Shadow Maps

Andrew Lauritzen, Marco Salvi, Aaron Lefohn
Intel Corporation

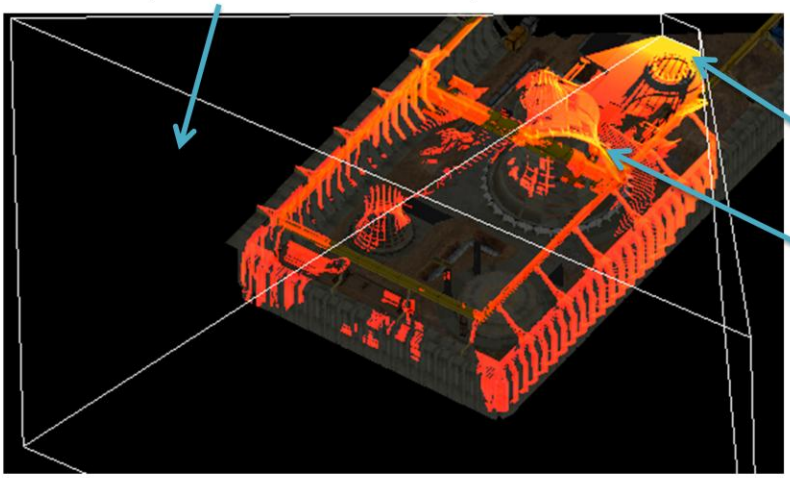
Shadowing Problem

Here's the general problem we want to solve efficiently:

Given a light and a set of pixels in view space, resolve occlusion between each pixel and the light.

Naïve Shadow Mapping

Wasted space that is never sampled



To visualize this problem, consider the shadow rays from the lights point of view.
The camera frustum is visualized in white.
The coloured pixels represent samples that the camera can see and thus for which we need shadow data.
The yellower the pixel, the denser the samples.

As this image demonstrates, the naive shadow mapping projection is pretty inefficient.

It wastes resolution where there are no objects in light space.

It produces perspective aliasing near the camera (lots of samples there means we need lots of resolution).

And finally you get projective aliasing on objects nearly parallel to the shadow rays.

Related Work

- Frustum partitioning needed
  - Z-partitioning is simple with good results
    - [Lloyd et al. 2006]
    - Also called “cascaded shadow maps” [Engel 2006]
    - Forms the basis for this work
  - Warping can be used, but partitioning is required
  - Addresses perspective aliasing

There are a number of ways to address these issues.

Z-partitioning (also called cascaded shadow maps) is the most common and produces good results.

Warping methods like perspective shadow maps can also be used, but some partitioning is always required.

However these methods only explicitly addresses perspective aliasing.

Related Work

- Projective aliasing is hard
  - Irregular and unbounded
  - Good partitioning helps but doesn't solve this aliasing
  - Difficult to address with fixed performance
- Suffer in performance but maintain quality:
  - Resolution-Matched SMs [Lefohn et al. 2007]
  - Irregular Z-Buffer [Johnson et al. 2005]

Projective aliasing is simply a hard problem as it produces unbounded and highly irregular sample distributions.

Thus it's difficult to address while maintaining a given performance and memory budget.

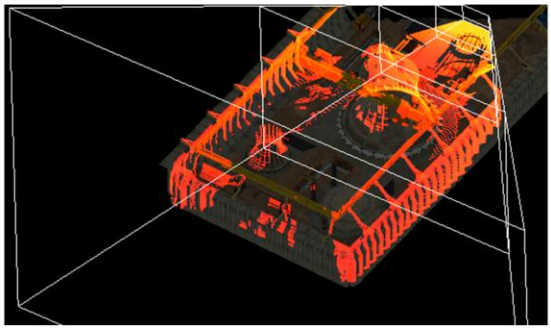
Our goal here is to maintain a fixed performance and memory budget and get as good quality as possible within that budget.

Conversely, techniques like resolution-matched shadow maps and the irregular Z-buffer are able to guarantee a quality level, but drop in performance for complex cases like heavy projective aliasing.

Thus they are typically unsuitable for games which have a fixed frame budget.

Z-Partitioning

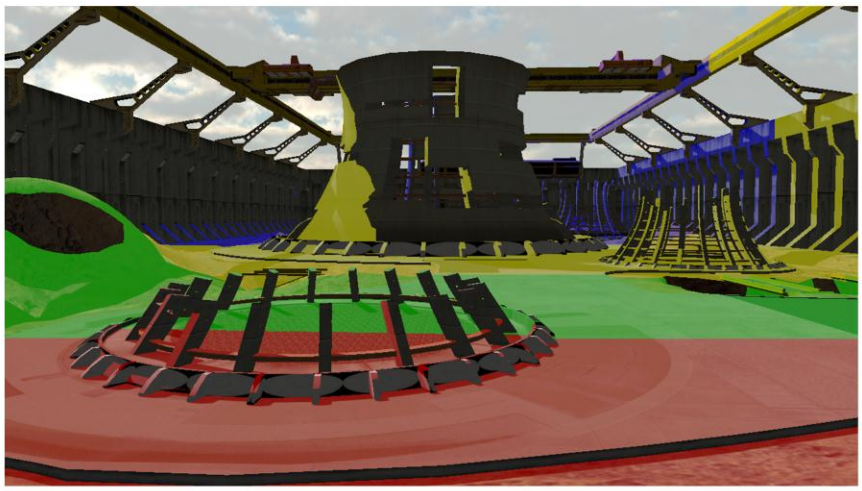
- Split camera frustum in Z
- Use different shadow maps for each partition



Our work is based on Z-partitioning.

The idea of Z-partitioning is to split the frustum into multiple segments and use a different shadow map for each.

Z-Partitioning



This image shows a visualization of Z-partitioning.

The coloured regions indicate which shadow map (four in this case) is being used for each part of the scene.

As you can see, the shadow maps partition the camera's Z axis.

Z-Partitioning

- Fixed performance and memory cost
  - Choose number of partitions and resolution of each
- Orthogonal to most other algorithms
  - Each partition is just like a normal shadow map!
- Open questions:
  - Where to split the frustum (in Z)?
  - Where to put the shadow maps in light space?

Z-partitioning is actually a really good solution with a lot of desirable properties such as predictable performance, memory costs and simplicity.

It is also orthogonal to many other shadow map algorithms such as warping and filtering, making it easy to fit into established pipelines.

However it does leave open a few questions.

In particular, how do you partition the Z range for optimal shadow map usage?

After that partitioning is done, where do you place the shadow maps in light space to avoid waste and ensure the best resolution possible?

Where to Partition Z?

- Logarithmic is best [Lloyd et al. 2006]
  - But only if the entire Z range is covered!
  - Needs tight near/far planes
- Parallel-Split Shadow Maps [Zhang et al. 2006]
  - Mix of logarithmic and uniform
  - Requires user to tune a parameter
    - Optimal value related to tight near plane…

Brandon Lloyd showed that a logarithmic partitioning produced the minimal maximum error over a Z range.

He noted though that it's very important to have tight bounds on the near plane in particular, or else the majority of partitions and resolution will be wasted in empty space near the camera.

To address this problem in a practical way, parallel-split shadow maps mix a logarithmic and uniform distribution with a tuneable parameter.

However the optimal value of this parameter is related to the optimal near plane of the scene and so it's not possible to set it in a globally-optimal way.

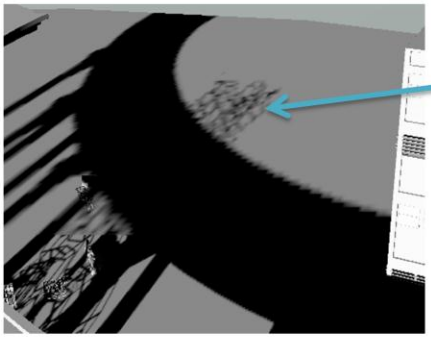
Where to Partition Z?

- In practice, artists tune for specific views
  - Tedious
  - Not robust to scene/camera changes
  - Ultimately suboptimal for arbitrary views

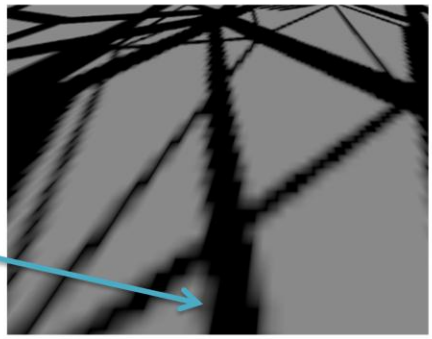
In practice, artists manually place partitions for specific views and scenes, but this is time consuming and ultimately sub-optimal.

This is the largest complaint about Z-partitioning that we hear from game developers.

Static Partitions (PSSM)



Too little resolution far



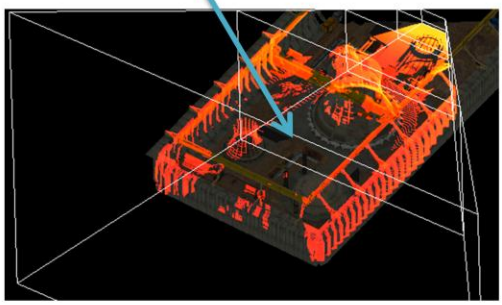
Too little resolution close

Here's an example of PSSMs with two different views of the same scene.

No static partitioning can handle both of these cases well.

Where to Place Shadow Maps?

- AABB of frustum segment?
  - Does not exploit vast regions of light space that are occluded or empty



A further problem is where to put the shadow maps in light space once we have the frustum partitions.

Almost everyone uses a simple axis-aligned bounding box of the frustum corners, but this is often suboptimal since it doesn't consider empty or occluded space.

Sample Distribution Shadow Maps

- Analyze the shadow sample distribution
  - Find tight Z min/max
  - Partition logarithmically based on tight Z bounds
  - Adapts to view and geometry with no need for tuning
- Compute tight light-space bounds
  - Tight axis-aligned bound box per partition
  - Greatly increases useful shadow resolution

Sample distribution shadow maps address both of these issues by analyzing the actual distribution of light space samples required to shadow the current view.

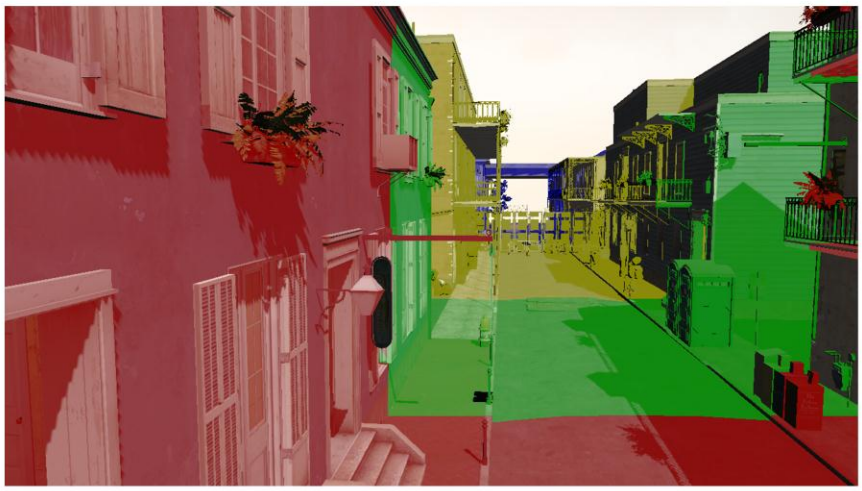
To ensure minimal wasted shadow map space, we compute a tight light-space bounding box of the required shadow map samples for each partition.

This greatly increases the useful amount of shadow map space by taking advantage of occluded or empty regions in light-space.

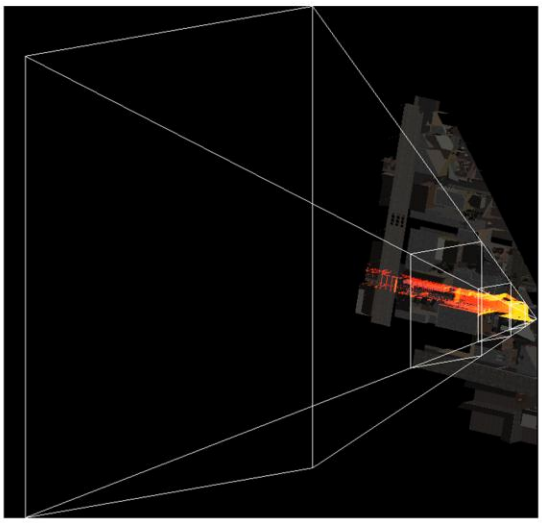
Example: PSSM

Significant aliasing problems both near the camera and far away.
The PSSM tuneable parameter has been tweaked for this view to provide a reasonable solution.
Significantly worse results can occur in practice!

Example: PSSM Partitions

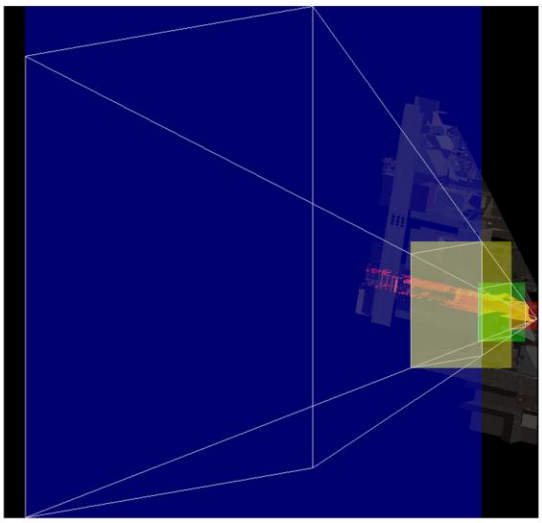


Example: PSSM Light Space



This is what the camera frustum looks like from light space.

Example: PSSM Partitions



A simple frustum partition bounding box results in lots of wasted space.

Large parts of the frustum are empty or occluded (this is typical).

Example: SDSM

Comparatively, SDSMs achieve sub-pixel resolution almost everywhere.

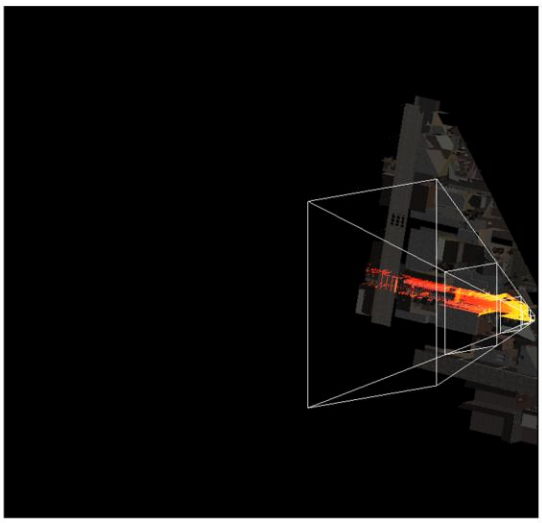
Note that filtering and shadow MSAA have been disabled to better display the raw resolution.

Example: SDSM Partitions

Partitioning is similar to the hand-tuned solution but fully automatic and adapts to the view.

Tight bounds achieve significantly better shadow resolution throughout.

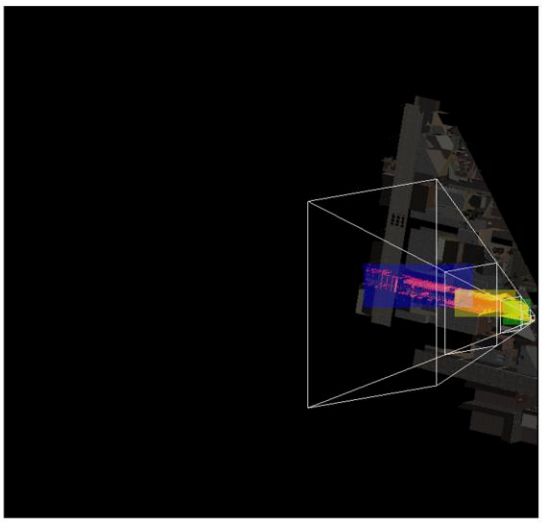
Example: SDSM Light Space



Near/far has been tightened to cover only visible samples.

Note that near being tightened is actually the most important.

Example: SDSM Partitions



Tight light space bounds take advantage of camera space occlusion and produce excellent bounds for each partition.

Partitioning Variants

- K-means clustering
  - Place partitions where there are lots of samples in Z
  - Good results for average error but has glass jaws
- Adaptive logarithmic
  - Like basic logarithmic but avoid gaps in Z
  - Very situational… usually not worth the effort
- These schemes require a depth histogram

There are many options for deriving a Z partitioning from the sample distribution data beyond the simple logarithmic scheme.

We explored a number of more complex algorithms including K-means clustering and adaptive gap-skipping algorithms.

Unfortunately these algorithms tend to be fairly situational and occasionally introduce glass jaws where particularly poor solutions are selected.

These fancier schemes also involve generating a full depth histogram while the simple logarithmic scheme only requires a min/max Z reduction.

Implementation

- Two different implementations:
  - Simple “reduce” implementation for logarithmic
    - Could be implemented in pixel shaders on DX9/10 hardware
  - General depth histogram implementation
    - Shared memory atomics make this feasible
    - Too slow and data-dependent on pre-DX11 hardware

We implemented these algorithms in DirectX 11 using compute shaders and they run entirely on the GPU.

There are two paths: the first uses a simple reduction which is all that the logarithmic scheme requires.

This path can be implemented in pixel shaders and is suitable for previous generation hardware.

The second path uses a generalize histogram and can support all of the variants.

This path uses shared memory atomics to generate the histogram efficiently.

Without this feature, generating this histogram is generally too slow to be practical.

Performance Overhead

- Overhead of SDSM analysis at 1080p:
  - Reduce path:
    - ATI 5870: 1.1ms
    - NVIDIA 480: 0.9ms
  - Histogram path:
    - ATI 5870: 1.4ms
    - NVIDIA 480: 7.2ms
- Logarithmic with reduce path is most practical

We measured the cost of the SDSM analysis step compared to using static partitions.

At 1080p the overhead of the reduce path is around 1ms for these GPUs.

The histogram path is practical on the ATI card.

On the NVIDIA card is impractically slow, perhaps due to the heavy use of shared memory atomics.

Thus together with it being the most robust, we recommend using the logarithmic scheme with the reduce path.

Performance (Full Frame Time)

- High quality setup:
  - Complex scene (Left 4 Dead 2), 1080p
  - 4 partitions with exponential variance shadow map filtering
    - Each 1024x1024 with 4x shadow AA
    - Mipmapped (full chain) with 16x anisotropic filtering
- ATI 5870:
  - PSSM: 14.9 ms
  - SDSM: 13.0 ms
- NVIDIA 480:
  - PSSM: 13.9 ms
  - SDSM: 12.3 ms

We now measure full frame times when using the technique.

We're using a very high quality shadow map setup that produces extremely nice results.

Shadow map MSAA is effectively jittered super-sampling since we're using only depth (which is super-sampled).

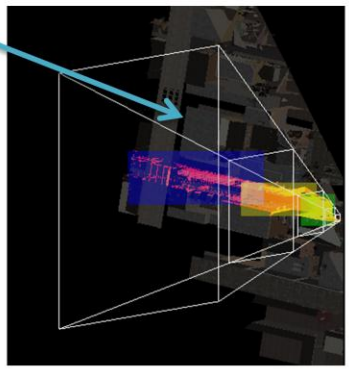
Note that because we're using a deferred renderer we only need the storage footprint for a single shadow map partition.

Even with these high quality settings we achieve good frame rates.

You'll also notice that SDSMs actually come out faster here than PSSMs in addition to being much higher quality...

Higher Quality AND Faster??

- SDSM produces tighter light space frusta
  - Less geometry rendered into the shadow maps



The reason is because of the tighter light-space frusta that we derive which enables better frustum culling.

It's effectively free occlusion culling for the shadow maps.

This is a typical result for SDSM in complex scenes.

Frustum Culling Caveat

- Partition bounds data generated on the GPU
  - Not available to the CPU for frustum culling!
- Stall and read it back (it's very small)
  - Awful, but this is what we do now and it is fairly fast…
- Future: do frustum culling on the GPU
  - Not a great mechanism right now…
  - Maybe if the GPU could submit work to itself? ☺

A potential caveat about using the SDSM results for better frustum culling is that the data is generated on the graphics card and thus is not readily available to the CPU for culling.

One option is simply to stall and read back the partition results.

While this is typically a bad thing to do, it's what we do for now and it's less slow than you might imagine.

Frustum culling can actually be computed very efficiently on the GPU right now, but there's currently no good mechanism for the GPU to drive rendering.

In the long run we really need better hardware and programming models that allow the GPU to submit work to itself instead of being spoon-fed by the CPU.

Temporal Coherence

- Shifting resolution can lead to temporal aliasing
- Quantize partition boundaries in light space?
  - Directional lights only
  - Cannot move or resize partitioning at all
  - Problems with some camera transformations
  - Too restrictive and sub-optimal

Another potential concern of any view-adaptive technique is temporal aliasing.

The typical solution to this for cascaded shadow maps is to quantize the partition boundaries in light space so that you only move them by texel-sized chunks.

Unfortunately this places far too many restrictions on the partitions, lights and camera and results in bad shadow solutions.

Temporal Coherence

- Quantize partitions to power-of-2 sizes?
  - Works, but harsh... wastes a lot of resolution
- Aim for sub-pixel shadow resolution
  - Needs sufficient partition resolutions (~ screen res.)
  - Use good filtering and shadow map anti-aliasing!

A slightly better option is to quantize the partitions to power-of-2 sizes so that you always “double” or “halve” resolution, which avoids sub-pixel shifts.

This works, but it’s too extreme.

Far too much shadow resolution is wasted.

The best option we’ve found is just to aim for sub-pixel shadow resolution.

If you can achieve that, the temporal shifts are not visible.

This requires sufficient shadow partition resolutions and good filter, but it is quite achievable on modern cards.

Future Work

- Better partitioning schemes?
  - We investigated quite a few, but there could be fancier algorithms that work well in practice
- Hybrid algorithms to address projective aliasing
  - Use more expensive algorithm where error is high

Our hope is that this idea spawns future work in the space.

There are many directions to go in terms of finding better partitioning schemes.

We investigated some of the more obvious ones but there may be better options just waiting to be discovered.

To address projective aliasing efficiently it will be necessary to look at hybrid algorithms.

Since we know the sample distribution we know the places that exhibit high error and could potentially apply a more expensive algorithm to those regions.

More investigation is needed to see if there is a practical and artifact-free way to do this.

References

- Wolfgang Engel. Cascaded Shadow Maps. In *ShaderX*[5]. Charles River Media, 2006.
- Gregory S. Johnson, Juhyun Lee, Christopher A. Burns, and William R. Mark. The Irregular Z-Buffer. In *ACM Transactions on Graphics 2005*.
- Aaron Lefohn, Shubhabrata Sengupta, and John D. Owens. Resolution-Matched Shadow Maps. In *ACM Transactions on Graphics 2007*.
- Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and Partitioning for Low Error Shadow Maps. In *Proc. Eurographics Symposium on Rendering 2006*.
- Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-Split Shadow Maps for Large-Scale Virtual Environments. In *Virtual Reality Continuum And Its Applications 2006*.
- Fan Zhang, Hanqiu Sun, and Oskari Nyman. Parallel-Split Shadow Maps on Programmable GPUs. In *GPU Gems 3*. Addison-Wesley 2008.

Acknowledgements

- Jason Mitchell and Wade Schin from Valve
- Natasha Tatarchuk and Hao Chen from Bungie
- Johan Andersson from DICE
- Square Enix
- Matt Pharr, Kiril Vidimce, Craig Kolb and the rest of the Advanced Rendering Technology Team at Intel
- Nico Galoppo, Greg Johnson, Doug McNabb, Anupreet Kalra and Mike Burrows from Intel

Questions?

- Full source and demo available at:
  - http://visual-computing.intel-research.net/art/publications/sdsm/