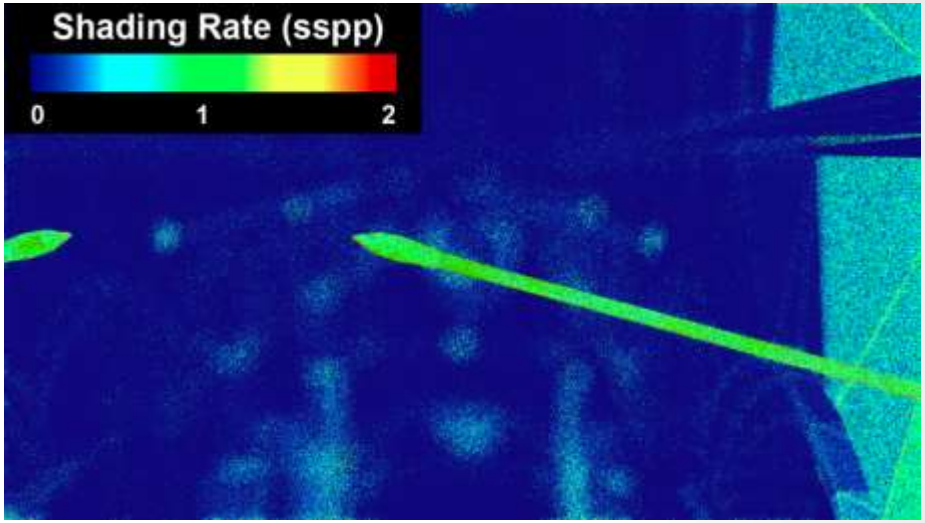
Decoupled Deferred Shading for Hardware Rasterization



Miklas & Yassen

Content

- Introduction
- The Compact Geometry Buffer
- The Algorithm
- Evaluation
- Conclusion

Introduction

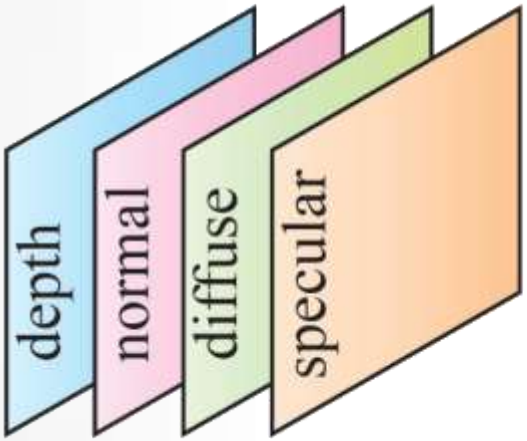
- High Quality Cinenatic Effects
  - Distributed Ray tracing
  - Stohastic Rasterization
- Antialiasing
  - MSAA

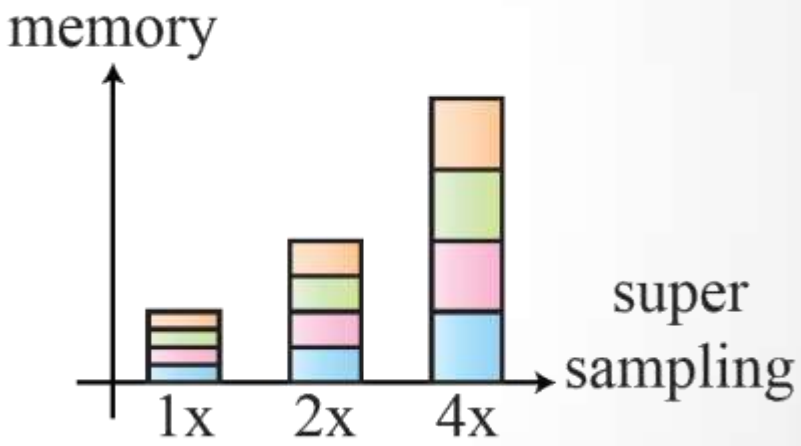
Introduction

- Deferred Shading
- The problem with multisampling
- Decoupling
  - Visibility Samples
  - Shading Samples

Geometry Buffer

- Naively couples visibility and shading

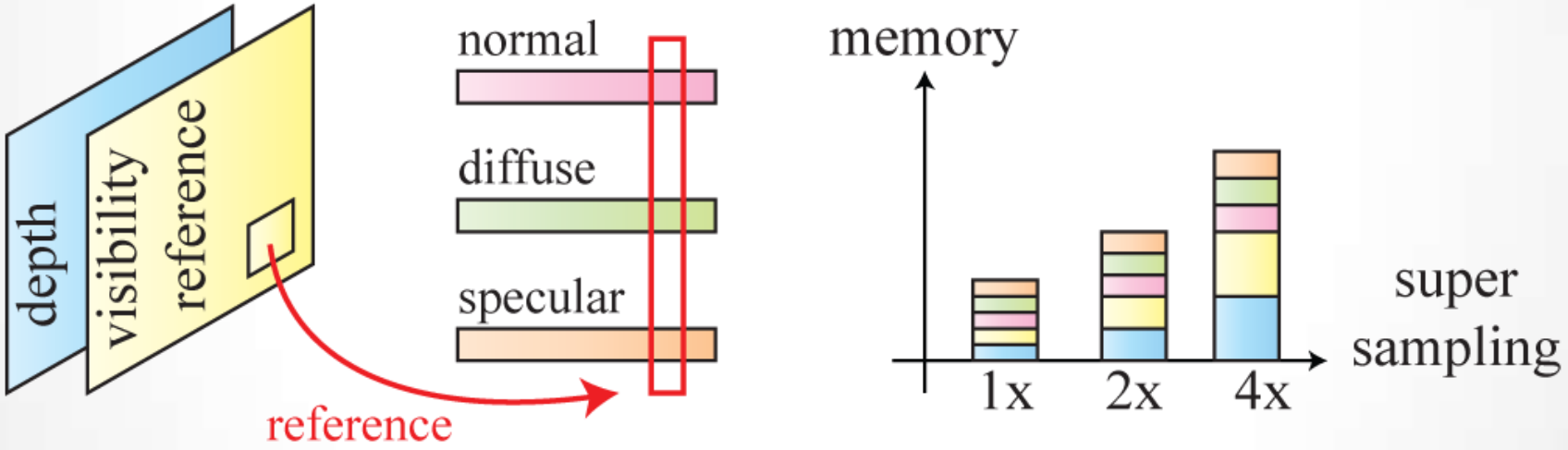




- Memory consumption grows exponentially

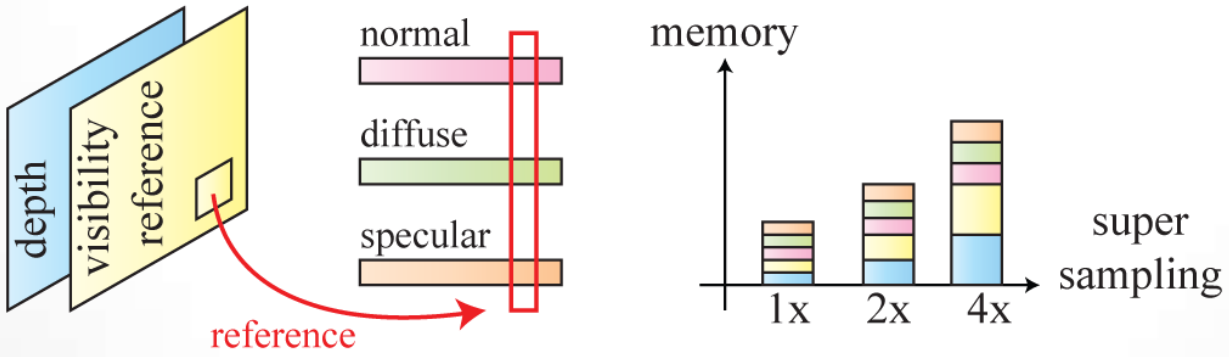
Compact Geometry Buffer

- The solution: Separate shading and visibility



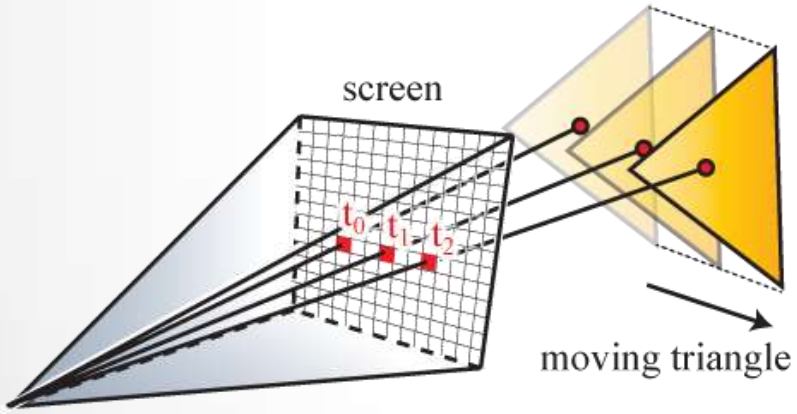
The CG Buffer

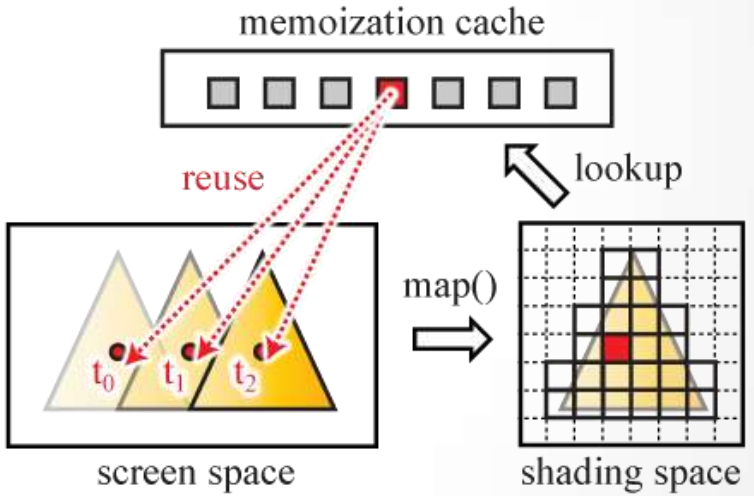
- Compact Geometry Buffer
  - Avoids storing redundant shading data
  - Less dependent on sampling density



CG Buffer

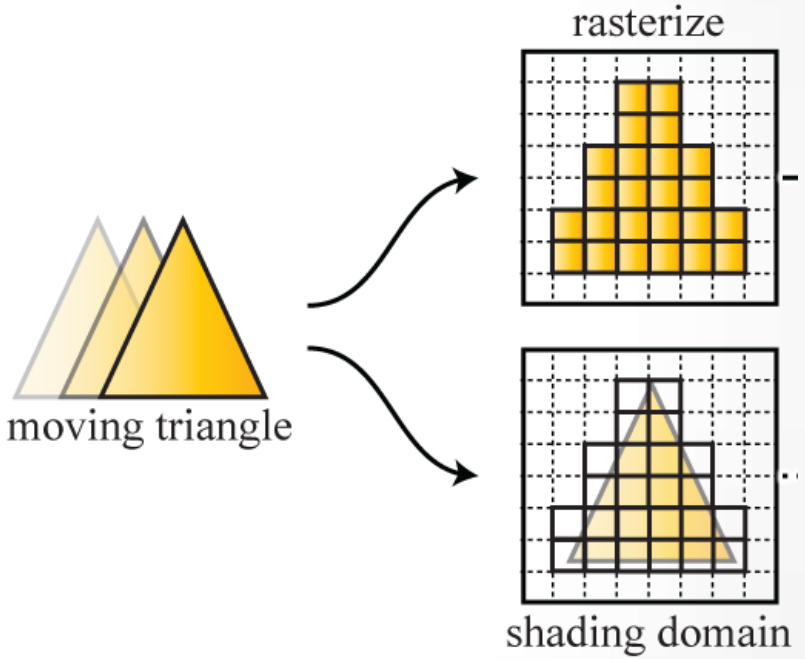
- What the CG Buffer Does:





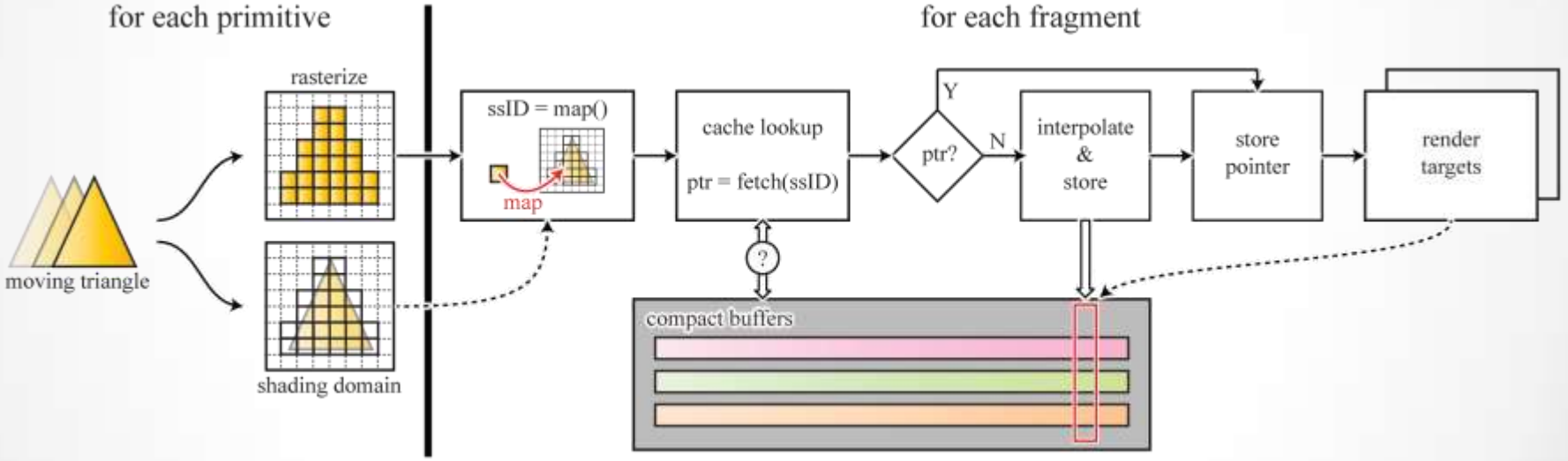
The CG Buffer

- Decoupled Shading
  - Map Visibility samples to a "Shading Domain"
  - Many-to-one mapping



CG Buffer

- How does the CG buffer work?



How does the CG Buffer work, cont.?

- Assign Unique Id's to Shading Samples(ssIDs)
  - ssID is unique across the image
  - Map ssIDs to each fragment
  - Cache shading data

Algorithm

- Global cache generation
  - Sampling and shading done in one pass
  - Uses standart fragment shaders
- Local cache
  - Multi Pass Approach
  - Uses GPU Compute Kernels

Global Cache Method

- Replaces G-Buffer generation

1. Determine ssID range for each primitive
   - In Geometry Shader
2. Map each fragment to an ssID
   - Depending on the domain

Global Cache Method cont.

- Check for the address in the cache
  - If it's there return it in the visibility reference
- What happens when ssID is not in cache?
  - Acquire lock for the ssID - Address pair
  - Generate buffer data in render targets
  - Save address in the cache and return it

Global Cache

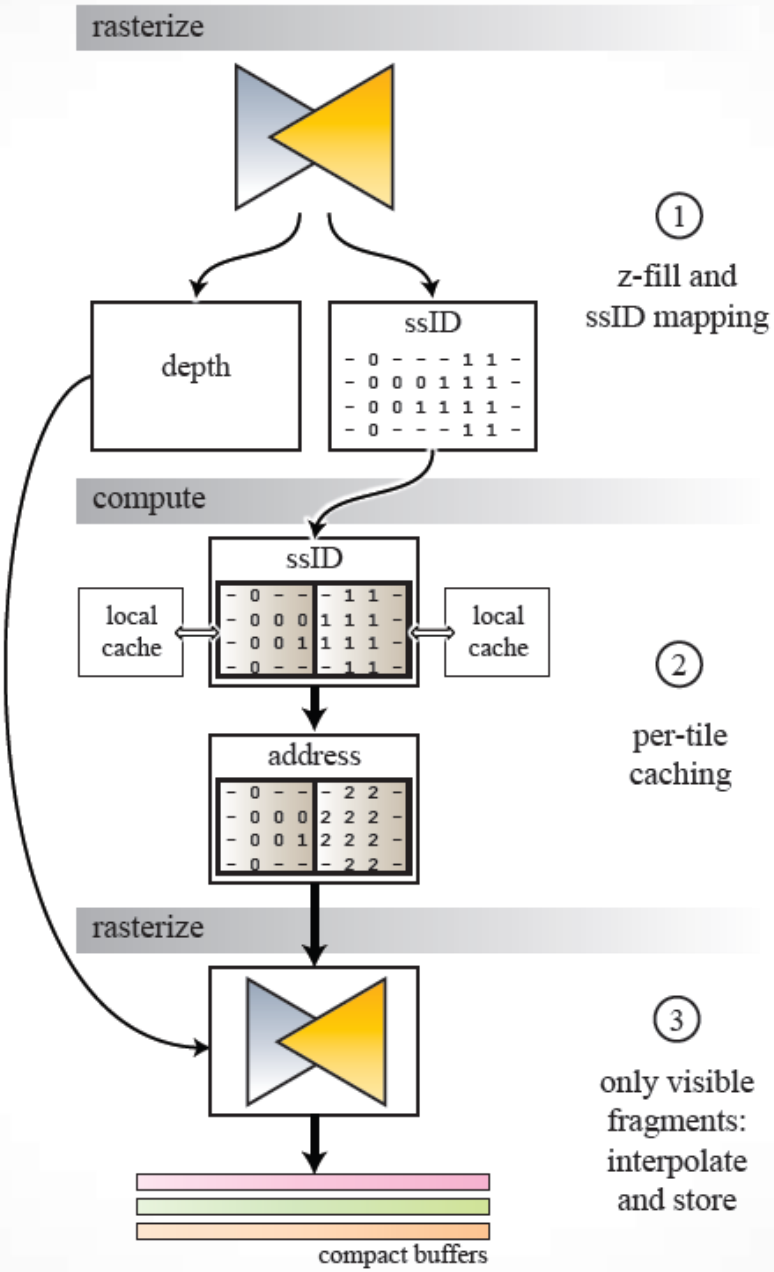
- Use cache to store recent ssIDs
  - Hit rate vs. Overhead
- Cache is divided into buckets
  - Buckets are mapped to ssIDs through a hash function
- Each bucket has an array of pairs
  - Each pair is ssID-Address

Global Cache - Problems

- Race conditions
- Compaction
  - Overdraw generates unreferenced data in the CG Buffer
  - Fill z-buffer in a pre pass, or
  - Clean up buffers after sampling

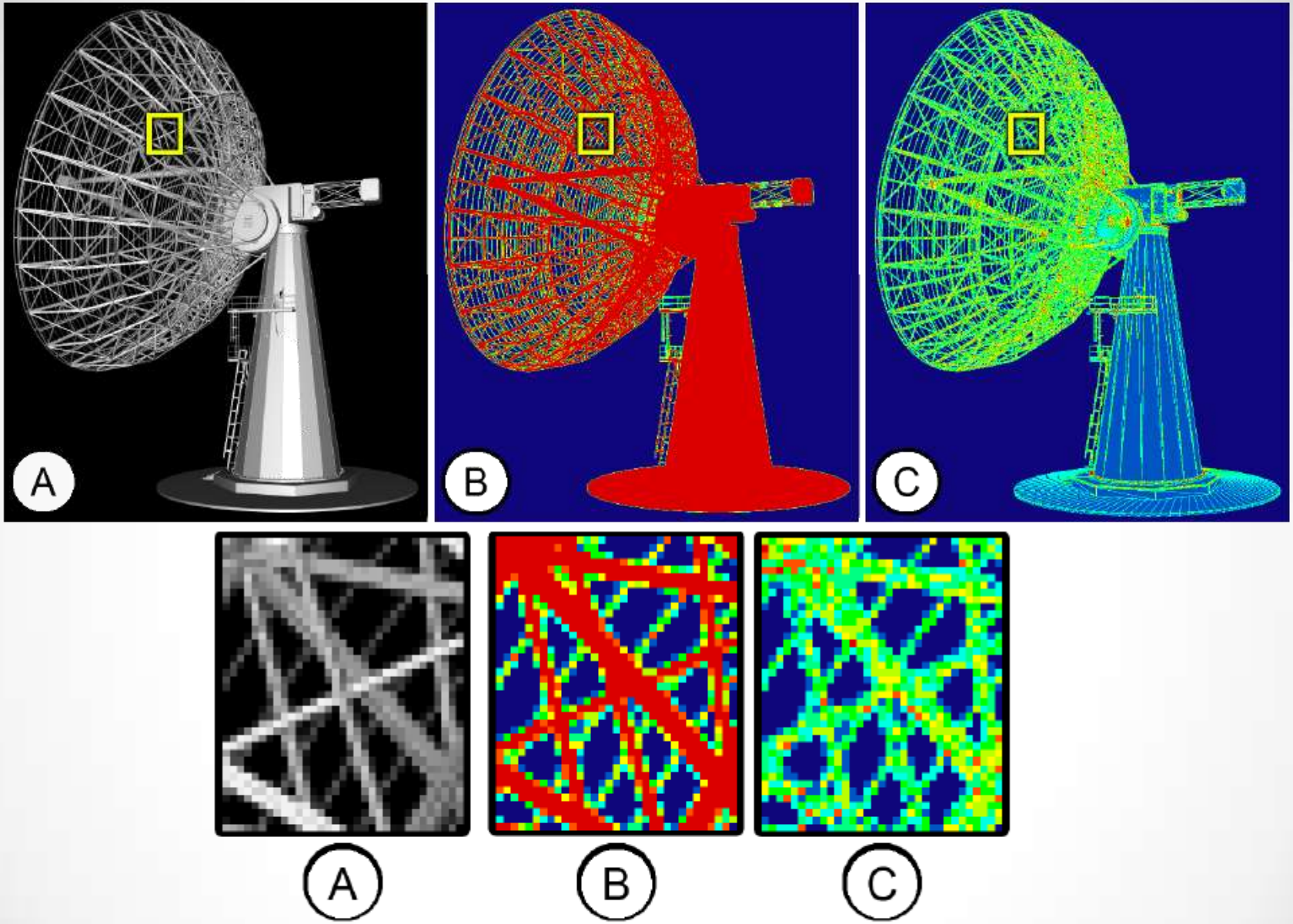
Per-Tile Shading Cache

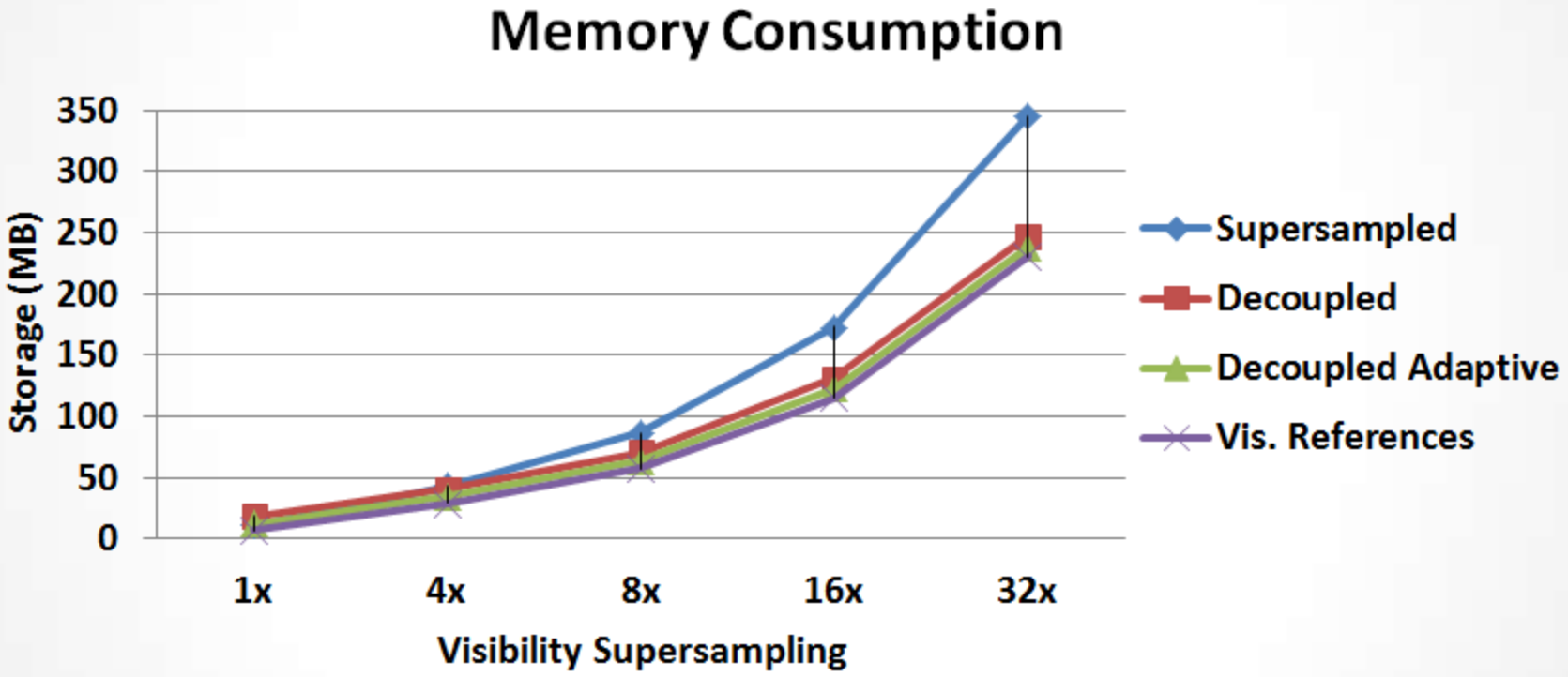
- **Global** shading cache
  - Becomes major bandwidth bottleneck
  - Most shading samples are visible in small areas, not globally.


rasterize
depth
ssID
- 0 - - - 1 1 -
- 0 0 0 1 1 1 -
- 0 0 1 1 1 1 -
- 0 - - - 1 1 -
1
z-fill and
ssID mapping
compute
ssID
local
cache
local
cache
2
per-tile
caching
address
- 0 - - - 2 2 -
- 0 0 0 2 2 2 -
- 0 0 1 2 2 2 -
- 0 - - - 2 2 -
rasterize
3
only visible
fragments:
interpolate
and store
compact buffers

Results

+ Real time
  + Adaptive shading rate
+ Cheap
  + Motion blur
  + FOV
  + AA (≥ 4x)
+ Low bandwidth (AA (≥ 4x)


A
B
C
A
B
C
Shading Rate (sspp)
0
4
8


Memory Consumption
Storage (MB)
350
300
250
200
150
100
50
0
1x
4x
8x
16x
32x
Visibility Supersampling
Supersampled
Decoupled
Decoupled Adaptive
Vis. References

Results

- Optimization (local cache) is **slower**
- If no AA and shading rate = 1, **double** memory usage (than deferred shading)

Results

- Not physically correct
- Possible artifacts on fast-moving objects

Decoupled Deferred Shading for Hardware Rasterization

Gabor Liktor and Carsten Dachsbacher

Computer Graphics Group
Karlsruhe Institute of Technology

Questions?