SIGGRAPH2006

The Parthenon Demo

Preprocessing and Real-Time Rendering Techniques for Large Datasets

Pedro Sander

ATI Research

Introduction

- At Siggraph 2004 Debevec et al presented "The Parthenon".
  - Structures laser-scanned and photographed
  - Captured HDR Lighting
- Our goal was to make a real-time version of this demo using these datasets.

The Challenge

- These sizes of the datasets are humongous!
  - 15 million triangles of geometry.
    - Simplified from original raw 90 million triangle model.
  - 2.1GB of HDR sky imagery.
  - 300MB (@350 512x512 textures) of texture data.
- This talk focuses on techniques for compressing, managing, and rendering these datasets in real-time on our next generation graphics cards.

Overview

- Progressive Buffers
- Video skybox
- Lighting and rendering

Overview

SIGGRAPH2006

- **Progressive Buffers**
- Video skybox
- Lighting and rendering

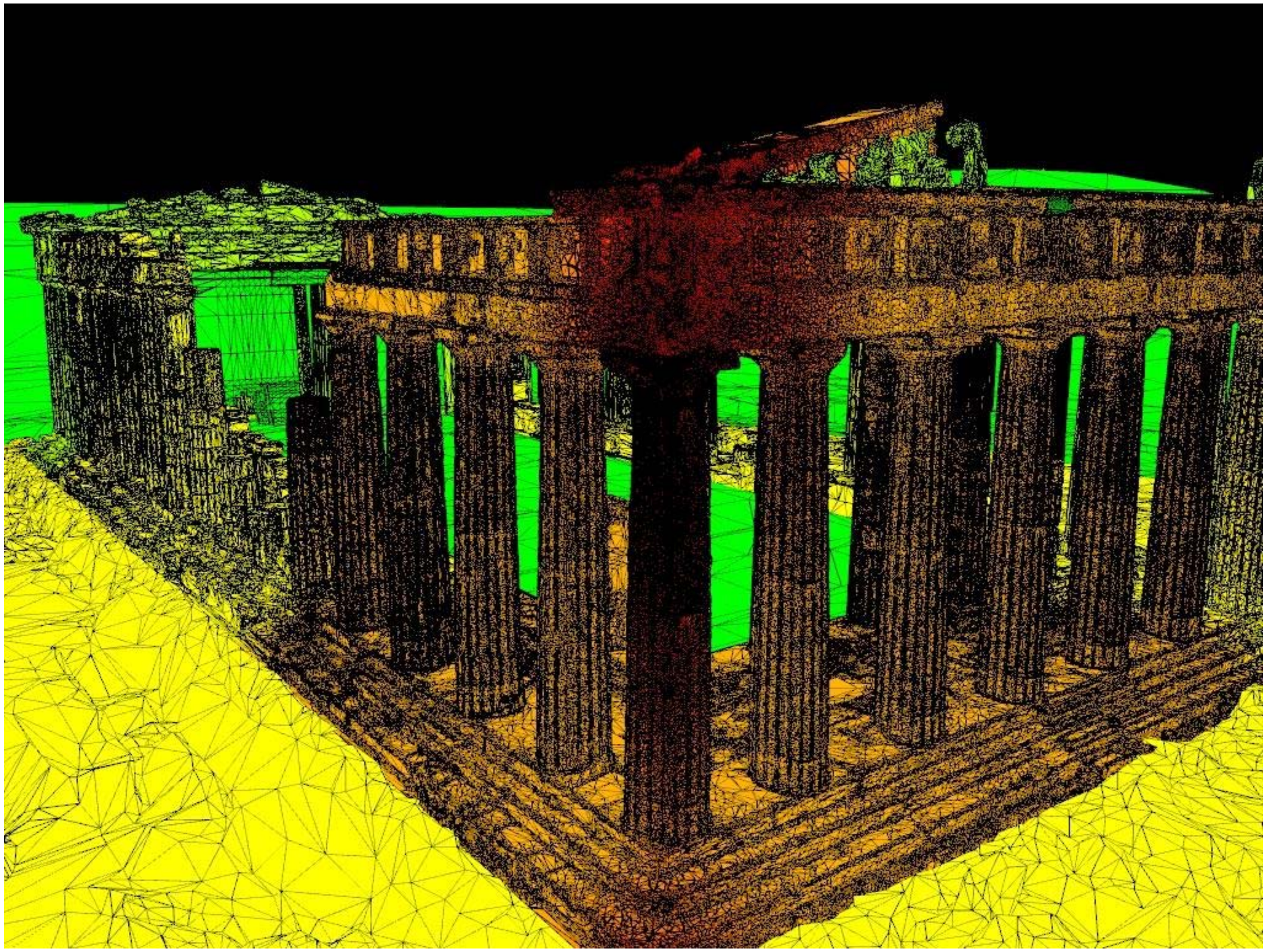
Overview

- A data structure and system for rendering of a large polygonal model:
  - Out-of-core
  - Texture/normal-mapping support
  - Smooth transitions between levels of detail *(no popping)*

Progressive Buffer Example

Progressive Buffer Example

SIGGRAPH2006



- Example: Five levels of detail color coded from Red (highest res) to green (lowest res)

Talk outline

SIGGRAPH2006

- Previous work
- The progressive buffer
  - Geometry LOD
  - Texture LOD
  - Coarse buffer hierarchy
- Automatic LOD control
- Memory management
- Results
- Future directions

Previous work

- View-dependent rendering (early works)
- [e.g., Xia and Varshney 1996, Hoppe 1997, Luebke and Ericson 1997, …]
  - Mostly per triangle operations
- Out-of-core view-dependent rendering [e.g., El-Sana and Chiang 2000, Vadrahan and Manocha 2002, Lindstrom 2003, Cignoni et al 2004, Yoon et al 2004, …]
  - Multiple static buffers
  - More efficient on current GPUs

Previous work

- Geomorphing static buffers
  [Gain 03]
- Per-vertex geomorphing
  [Grabner 01]
- Our method:
  - Geomorphs on GPU
  - Texture mapping
  - Hierarchy of clusters to reduce draw calls
- More similar to independent work of Borgeat 05

Continuous LOD control

SIGGRAPH2006

- Texture-mapping
  Allows for lower geometric level of detail without loss in quality (e.g., flat regions can be textured).
- Geomorphing
  A lower number of rendered triangles causes undesired popping when changing level of detail. Geomorphing provides a smoother transition.
- Summary:
  - Complex models
  - Wide range of graphics hardware
  - No need for tiny pixel-sized triangles

The progressive buffer (PB)

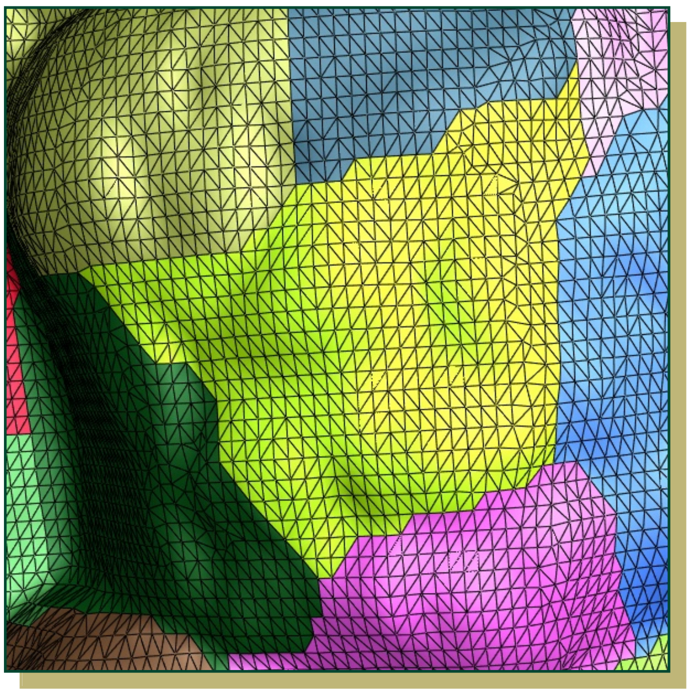
SIGGRAPH2006

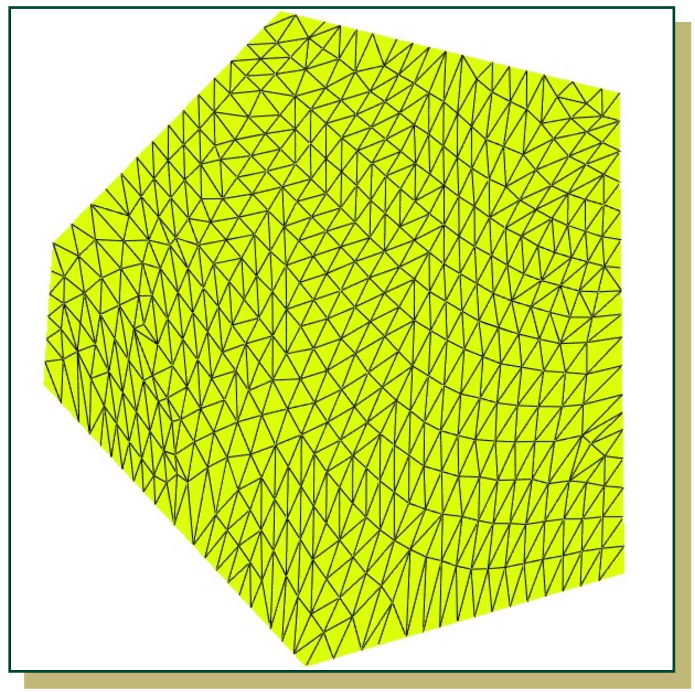
Preprocess (mostly based on previous methods):

- Split model into clusters
- Parametrize clusters and sample textures
- Create multiple (e.g., five) static vertex/index buffers for different LODs, each having ¼ of the vertices of its parent
  - We achieved this by simplifying each chart at time from one LOD down to the next, also simplifying the boundary vertices to its neighbor
  - Simplify respecting boundary constraints and preventing texture flips
[Cohen 98, Sander 01]
- Perform vertex cache optimization for each of these buffers [DX9; Hoppe 99]

Texture parametrization

- Goal: Penalizes undersampling
  - L^2 geometric stretch of Sander et al. [2001]
  - Hierarchical algorithm to generate texture coordinates

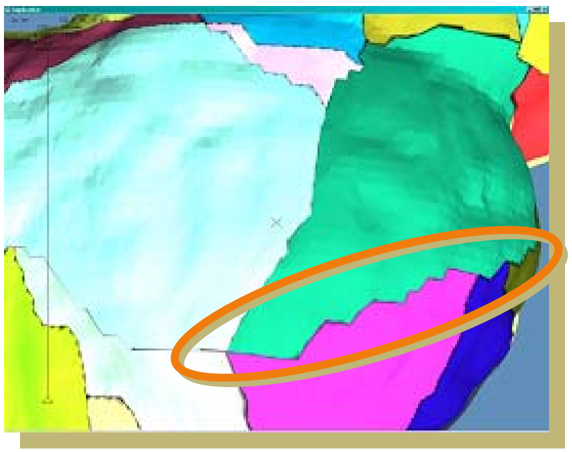


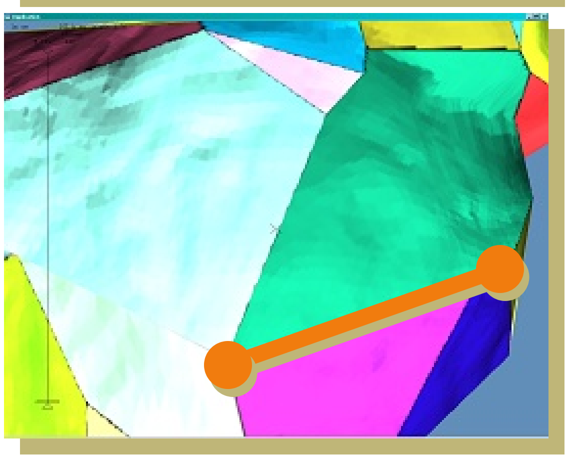


Straight texture boundaries

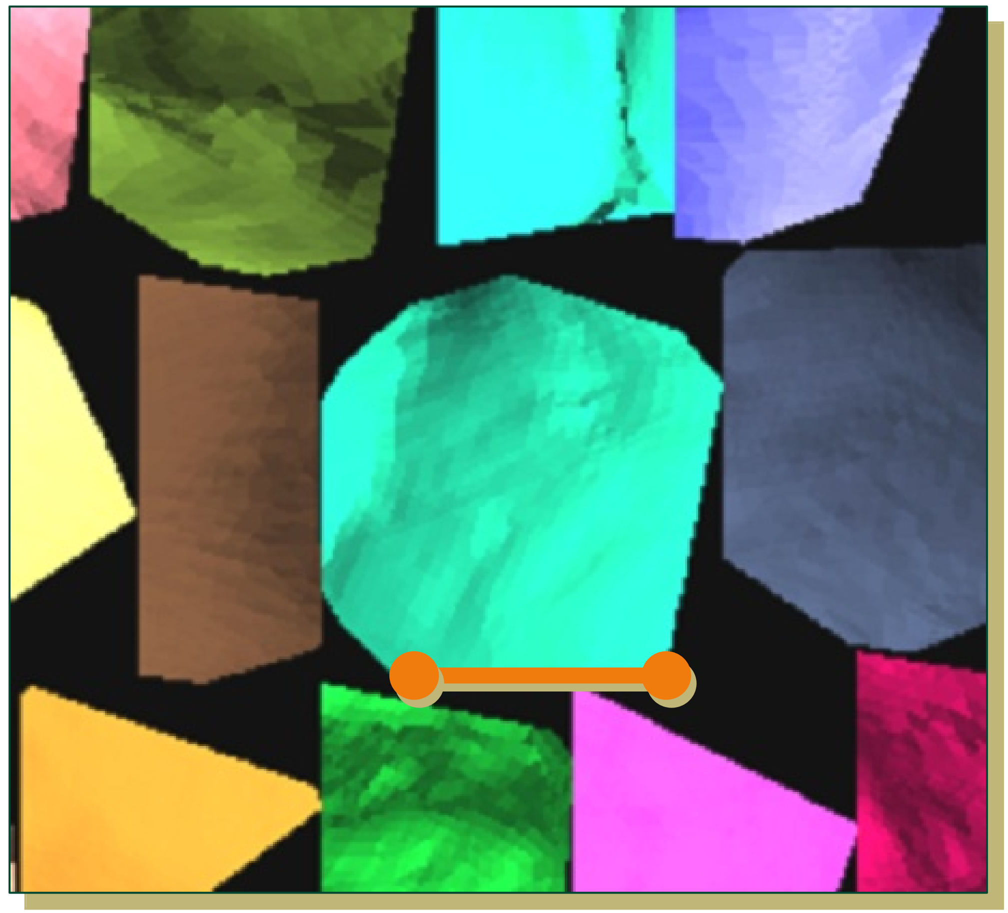
SIGGRAPH2006

fine mesh



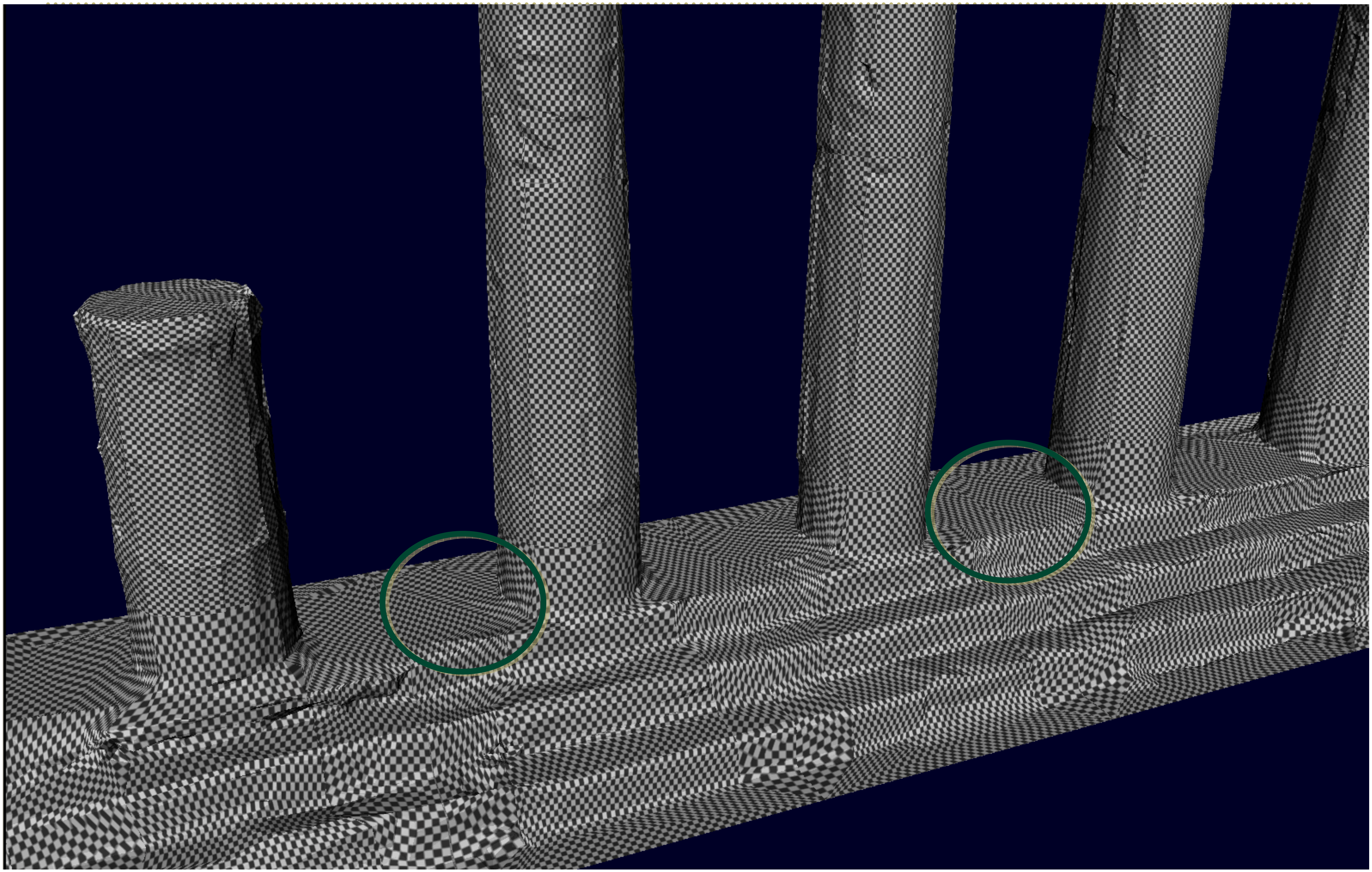


coarse mesh



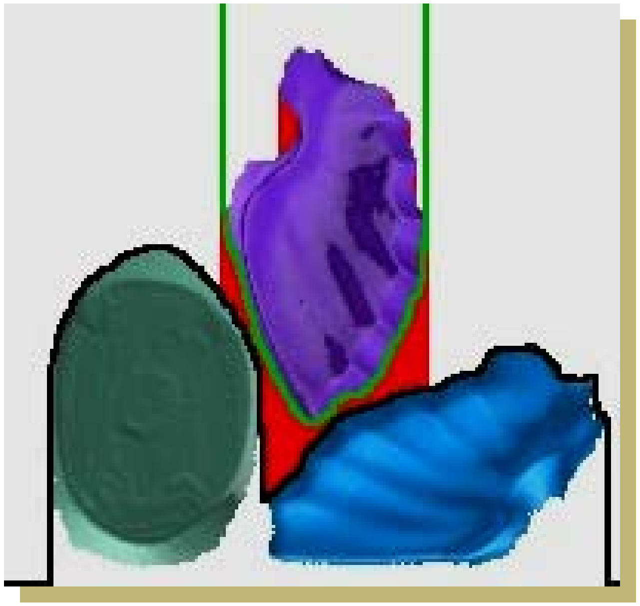
texture map

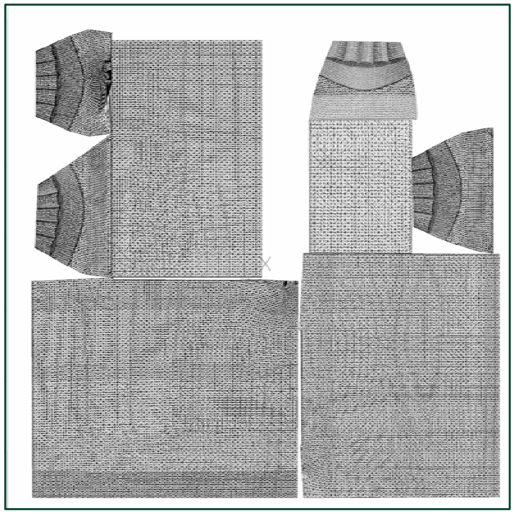
Straight boundary distortion



Texture packing

- Tetris packing [Levy 02]
  - Goal: minimize wasted space (red)
  - Place a chart at a time (from largest to smallest)
  - Pick best position and rotation (minimize wasted space)
  - Repeat above for multiple square dimensions
    - pick best



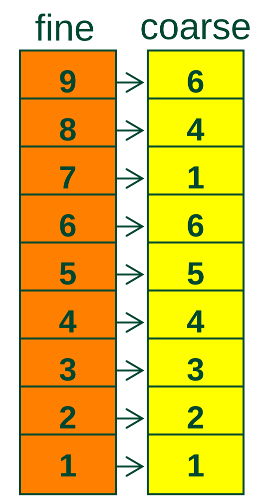


The progressive buffer (PB)

SIGGRAPH2006

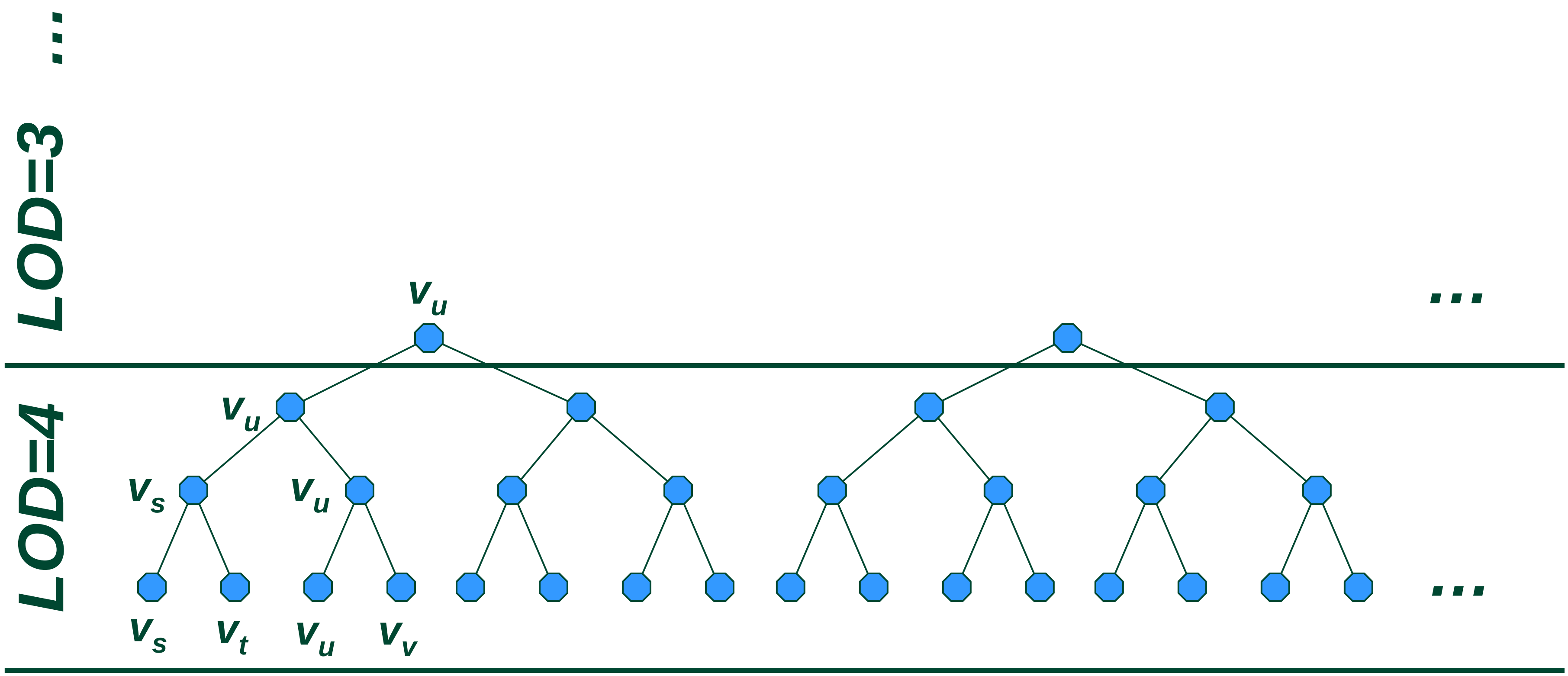
Static buffers:

- Each static buffer will contain an index buffer and two vertex buffers:
  - Fine vertex buffer
    Representing the vertices in the current LOD
  - Coarse vertex buffer
    Vertex-aligned with the fine buffer such that each vertex corresponds to the "parent" vertex of the fine buffer in the next coarser LOD
    (Note: requires vertex duplication)



PB_i

The progressive buffer (PB)



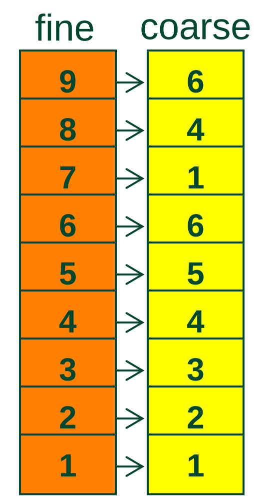
Vertex parents for LOD=4: $v_s, v_t, v_v \rightarrow v_u$

The progressive buffer (PB)

SIGGRAPH2006

Runtime:

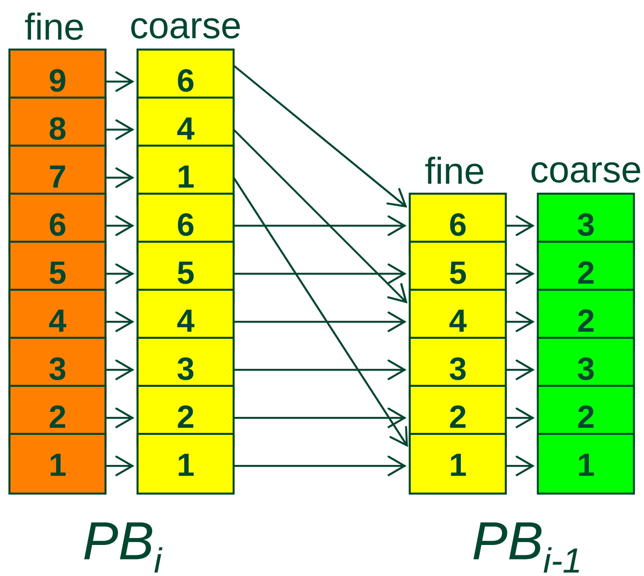
- A static buffer is streamed to vertex shader
  *(LOD determined based on **cluster's center** distance to camera)*
- Vertex shader smoothly blends position, normal and UVs.
  *(blending weight based on **vertex** distance to camera)*



PB_i

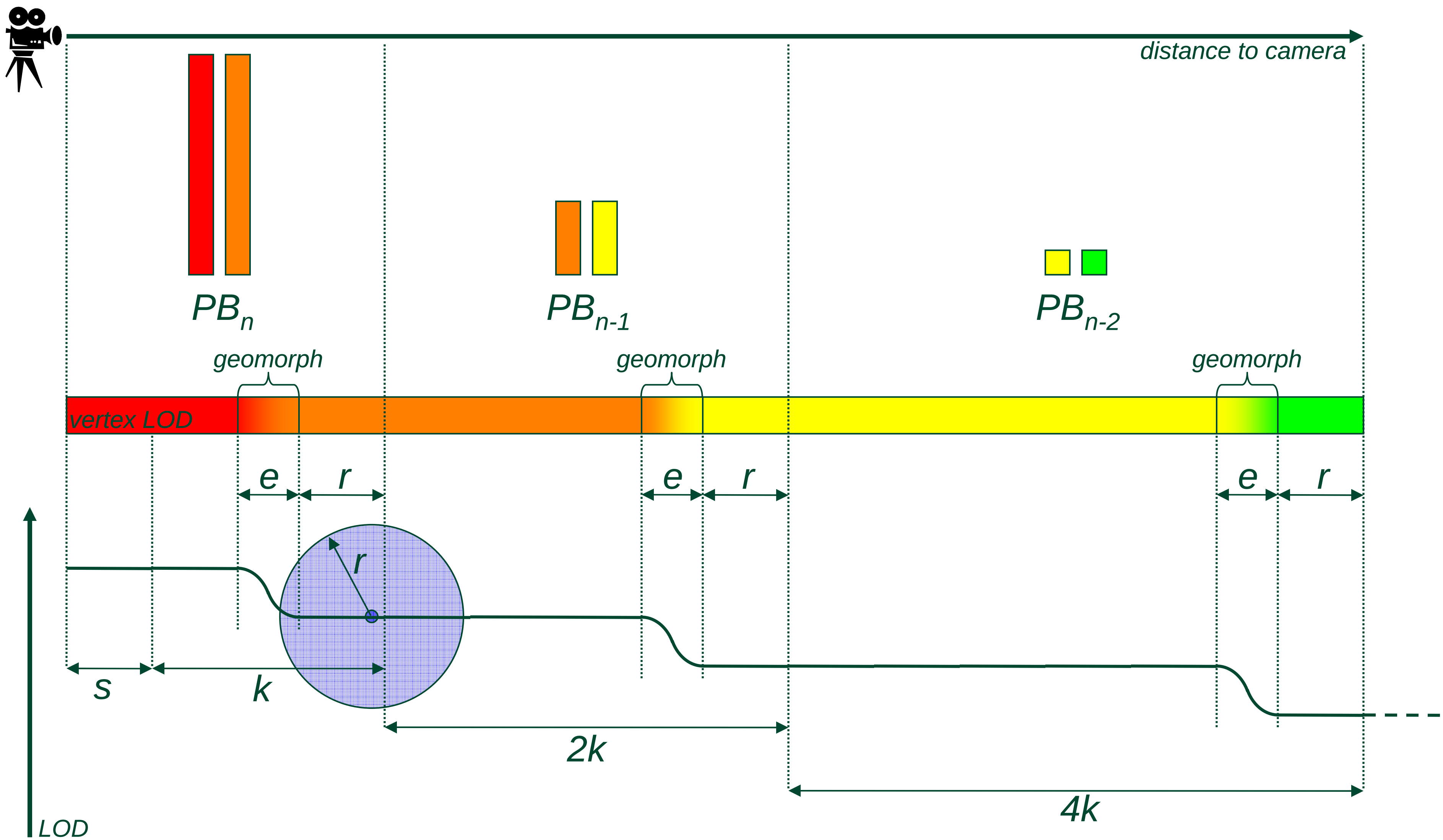
Buffer geomorphing

- Decrease level of detail:
  - Geomorph
    PB_i orange → yellow
  - Switch buffer
    $PB_i \rightarrow PB_{i-1}$
  - Geomorph
    PB_{i-1} yellow → green
  - …



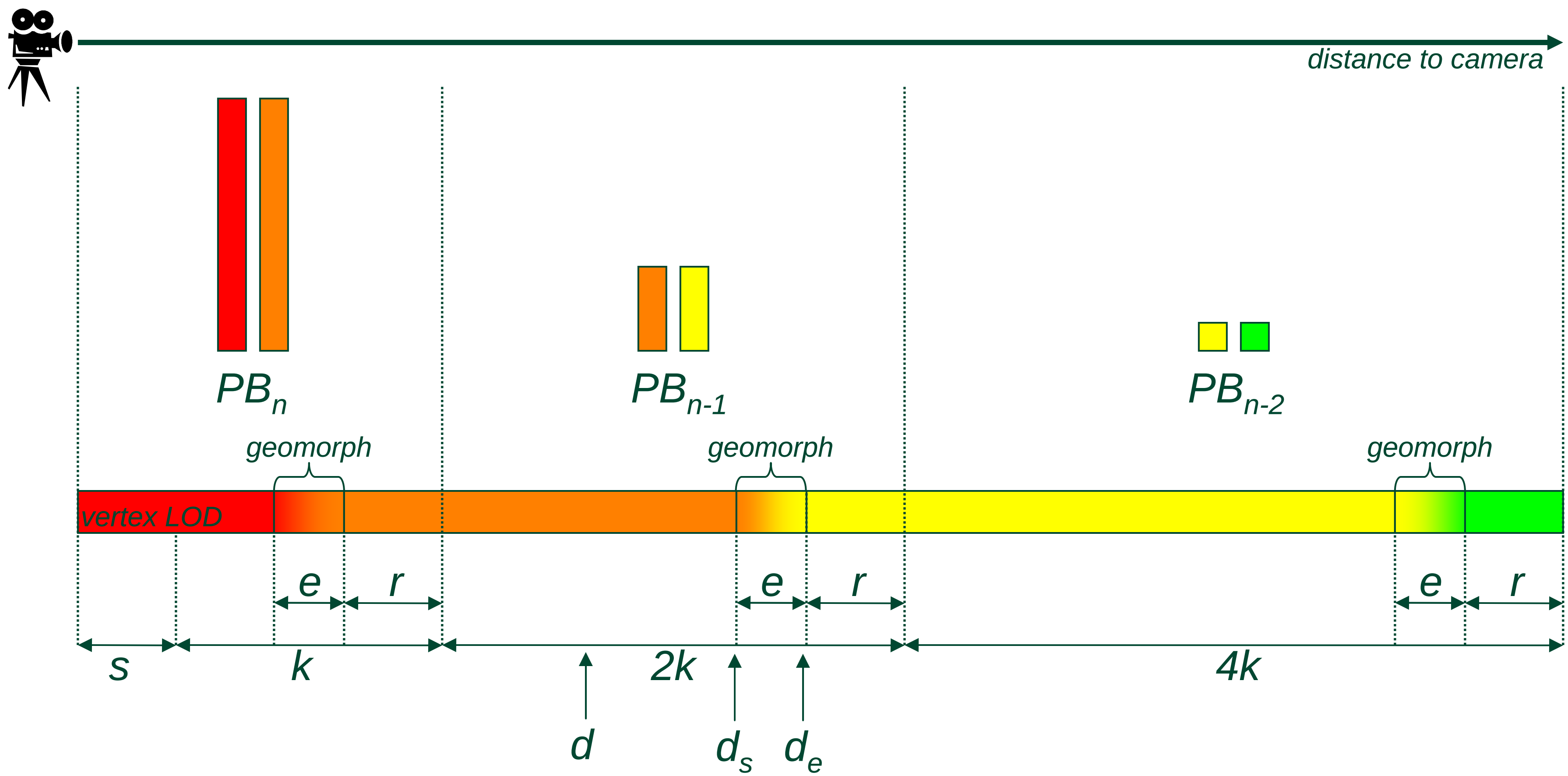
- Increase level of detail by reversing the order of operations.

How it works



distance to camera
PBn
PBn-1
PBn-2
geomorph
geomorph
geomorph
vertex LOD
e
r
e
r
e
r
r
s
k
2k
4k
LOD

LOD bands and weights



$$i = \mathrm{floor}\left(\log_2\left(\frac{d-s}{k}+1\right)\right) \quad d_e = (2^{i+1}-1)k + s - r$$

$$d_s = d_e - e$$

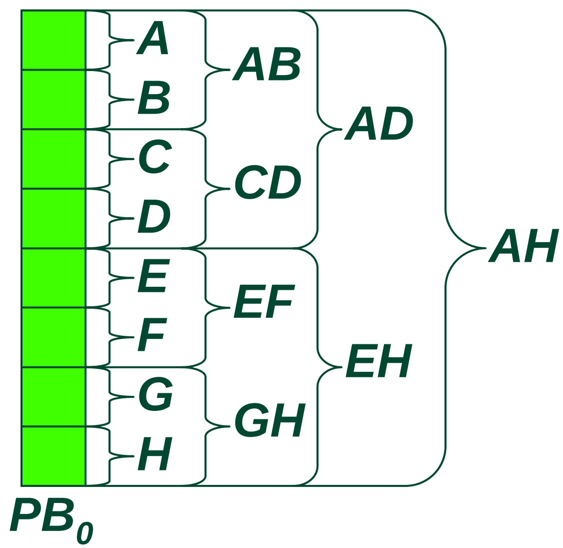
Texture LOD

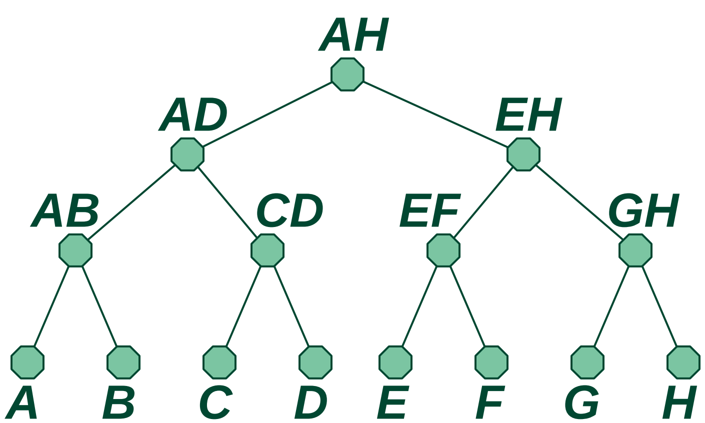
SIGGRAPH2006

- Analogous to vertex LOD
- Each LOD also has texture
- Each coarser LOD has ¼ of the # of vertices and ¼ of the # of texels of the previous LOD
- Essentially, we drop the highest mip level when coarsening, and add a mip level when refining
- Textures are blended just like vertices:
  - Vertex geomorph weight passed down to pixel shader
  - Pixel shader performs two fetches (one per LOD)
  - Pixel shader blends resulting colors according to the interpolated weight

Coarse buffer hierarchy (CBH)

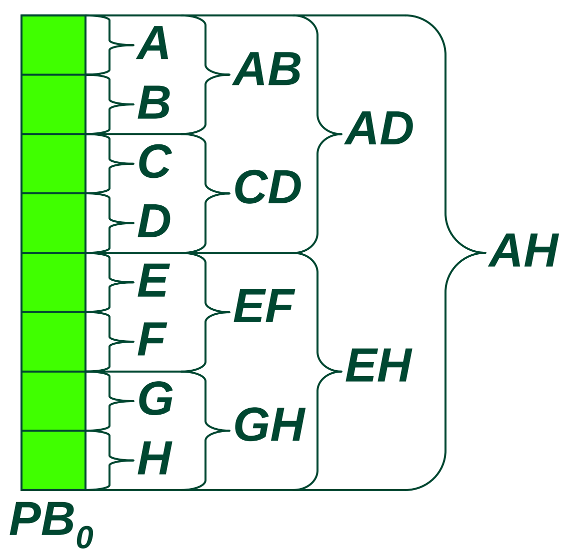
- Store coarse LOD of all clusters in a single vertex/index/texture buffer in video memory
- Group draw calls when adjacent clusters are far from camera

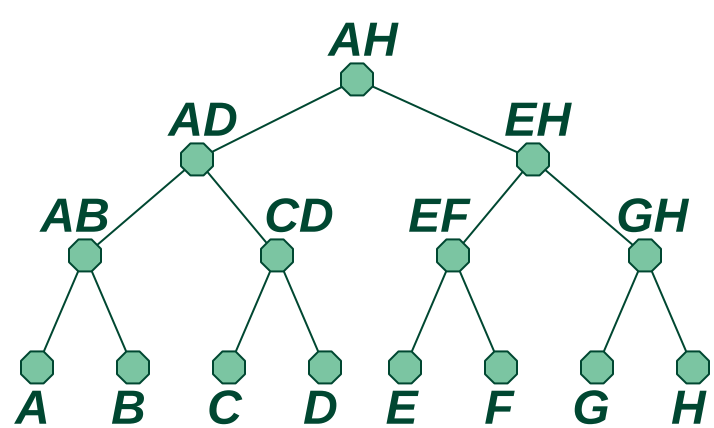




Coarse buffer hierarchy (CBH)

- Binary tree constructed using a bottom-up greedy merge algorithm
- Priority metric is the radius of bounding sphere of potential merged cluster





CBH textures

- Textures of voxels at coarsest LOD are grouped:

- Always stored in video memory
- Texture coordinates in the CBH buffer adjusted.
- No visible popping when switching from coarse static buffer to CBH buffer

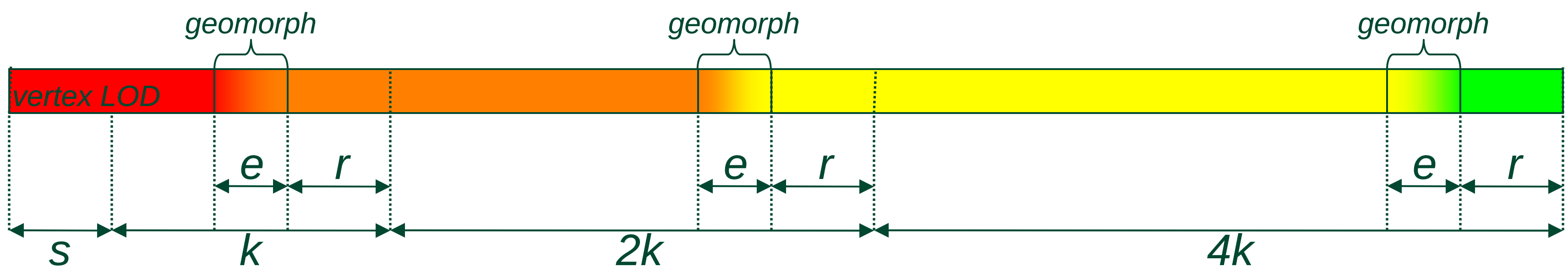
Limitations of data structure

SIGGRAPH2006

- Vertex buffer size is doubled
  (but only small subset of data resides in video memory)
- Clusters should be about the same size
  (a large cluster would limit minimum LOD band size)
- Larger number of draw calls than purely hierarchical algorithms
  (cannot switch textures within same draw call;
  coarse level hierarchy partly addresses this)
- Texture stretching due to straight boundaries

Automatic LOD control

- Bounds:
  - System memory
  - Video memory
  - Framerate (less stable)
  - Maximum band size
- Values of *k* and *s* slowly adjusted accordingly to remain within the above bounds



Memory management

SIGGRAPH2006

- Separate thread loads data, and based on distance to viewer sets priorities as follows:

Priority	System memory	Video memory*	Sample bounds
3 (active)	Yes	Yes	100MB
2 (almost active)	Yes	Yes	20MB
1 (needed soon)	Yes	No	50MB
0 (not needed)	No	No	Full dataset

*Priority (with LRU as tie-breaker) used for determining what is loaded on video memory

Memory management

SIGGRAPH2006

- We compute continuous LOD of each buffer.
- Taking the integer part, we get the static buffer, and assign it priority 3:

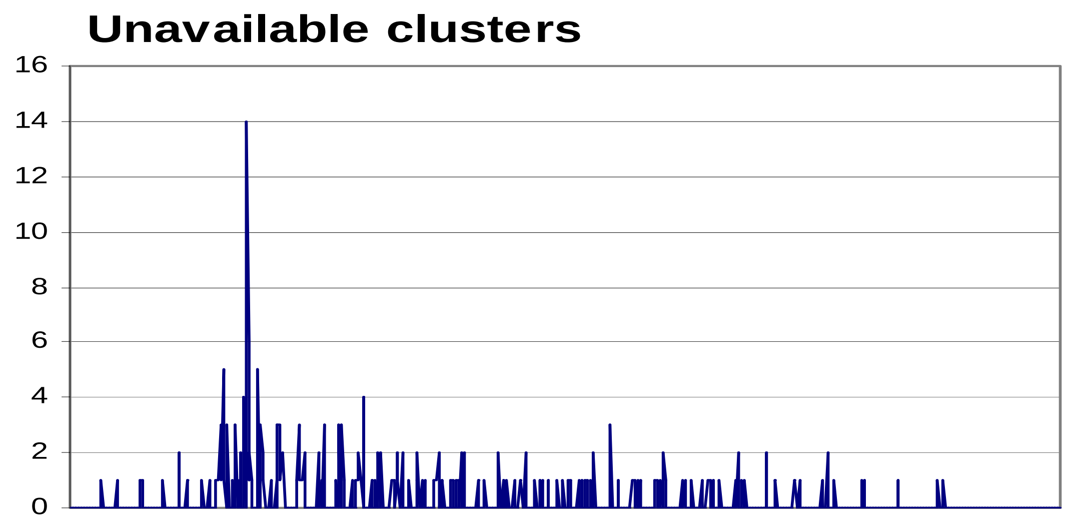
$$i = \text{floor}\left(\log_2\left(\frac{d-s}{k}+1\right)\right)$$

- If the continuous LOD is within a specified threshold of another static buffer's LOD, we set that buffer's priority accordingly:

Threshold	**Priority**	**Target**	**Example**
e_{video}	2	Video memory	0.75
e_{system}	1	System memory	1.00

Prefetching results

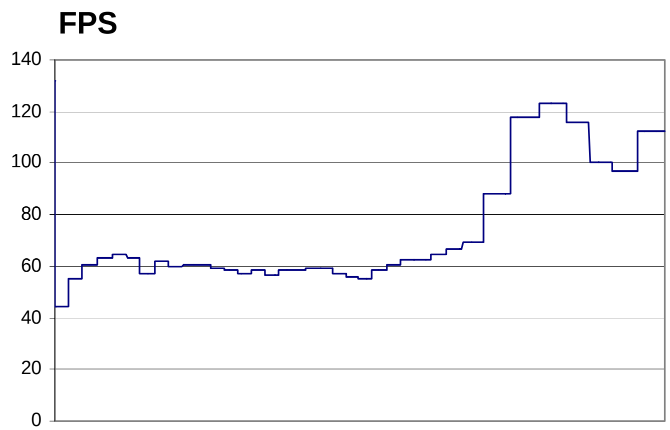
- By prefetching and keeping approximately 20% of additional data than that being rendered, we ensure we have the appropriate cluster LODs required for rendering
- Without prefetching, several buffers may become unavailable:

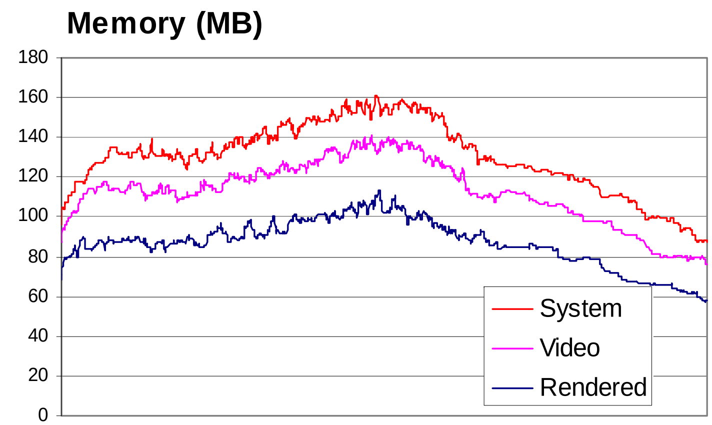


- May vary dramatically based on hard drive seek times, background tasks, other CPU usage.

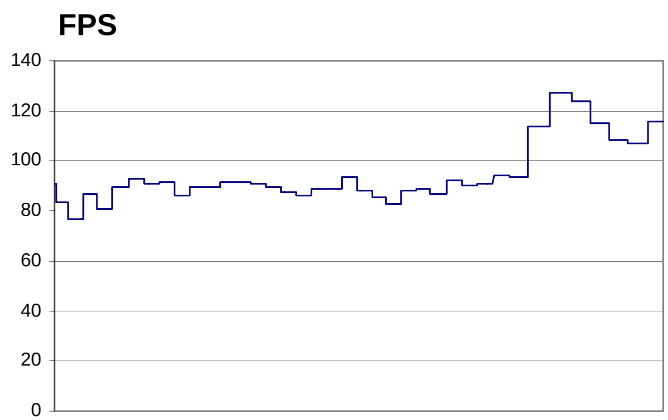
Statistics

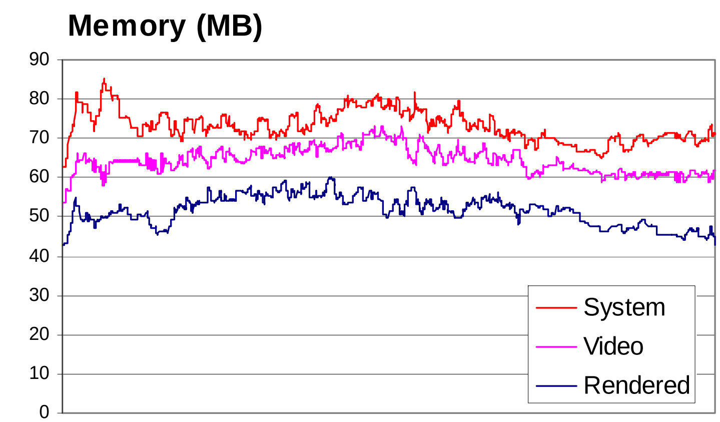
Fixed LOD





Variable LOD



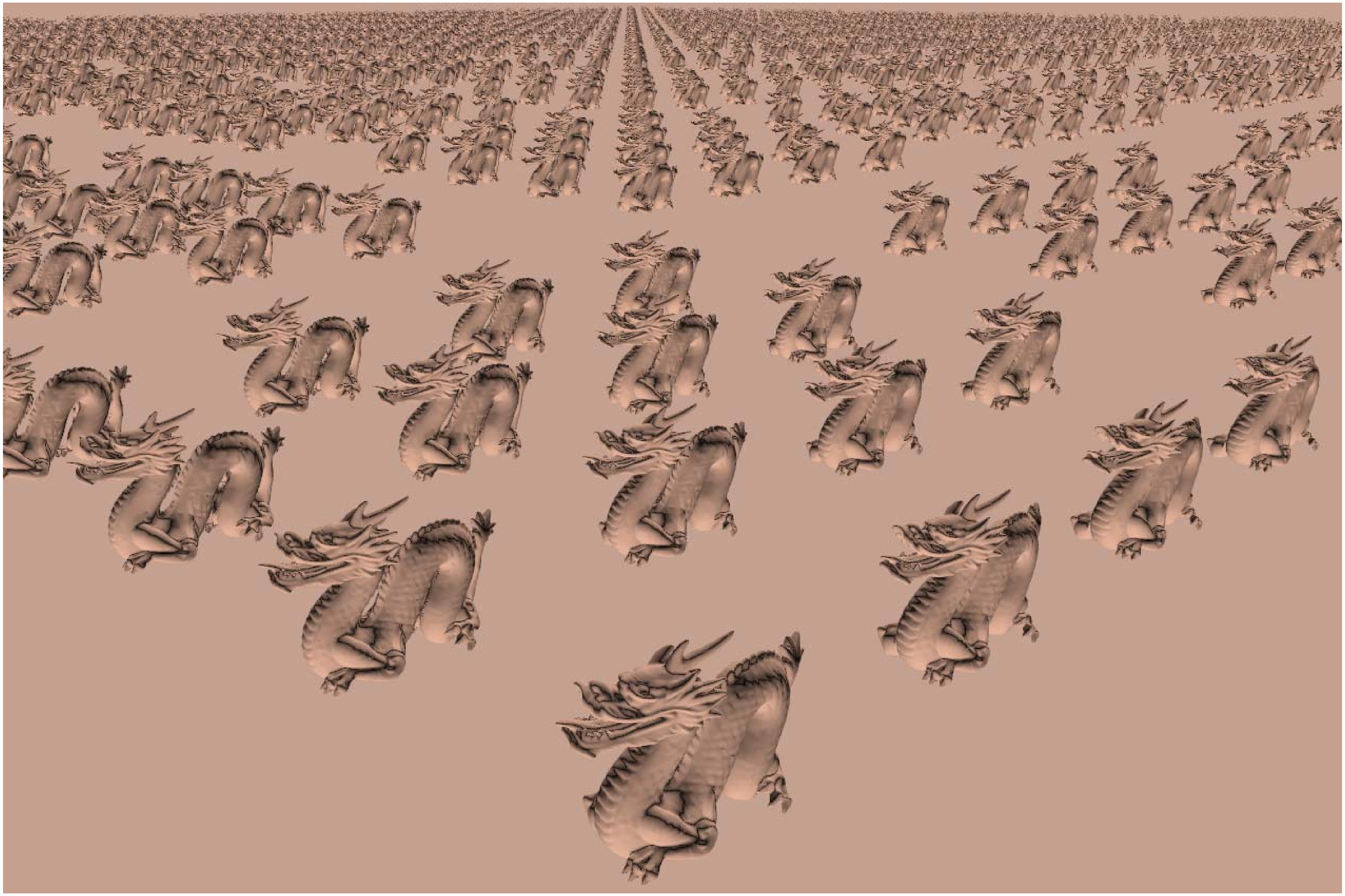


Shadow map example

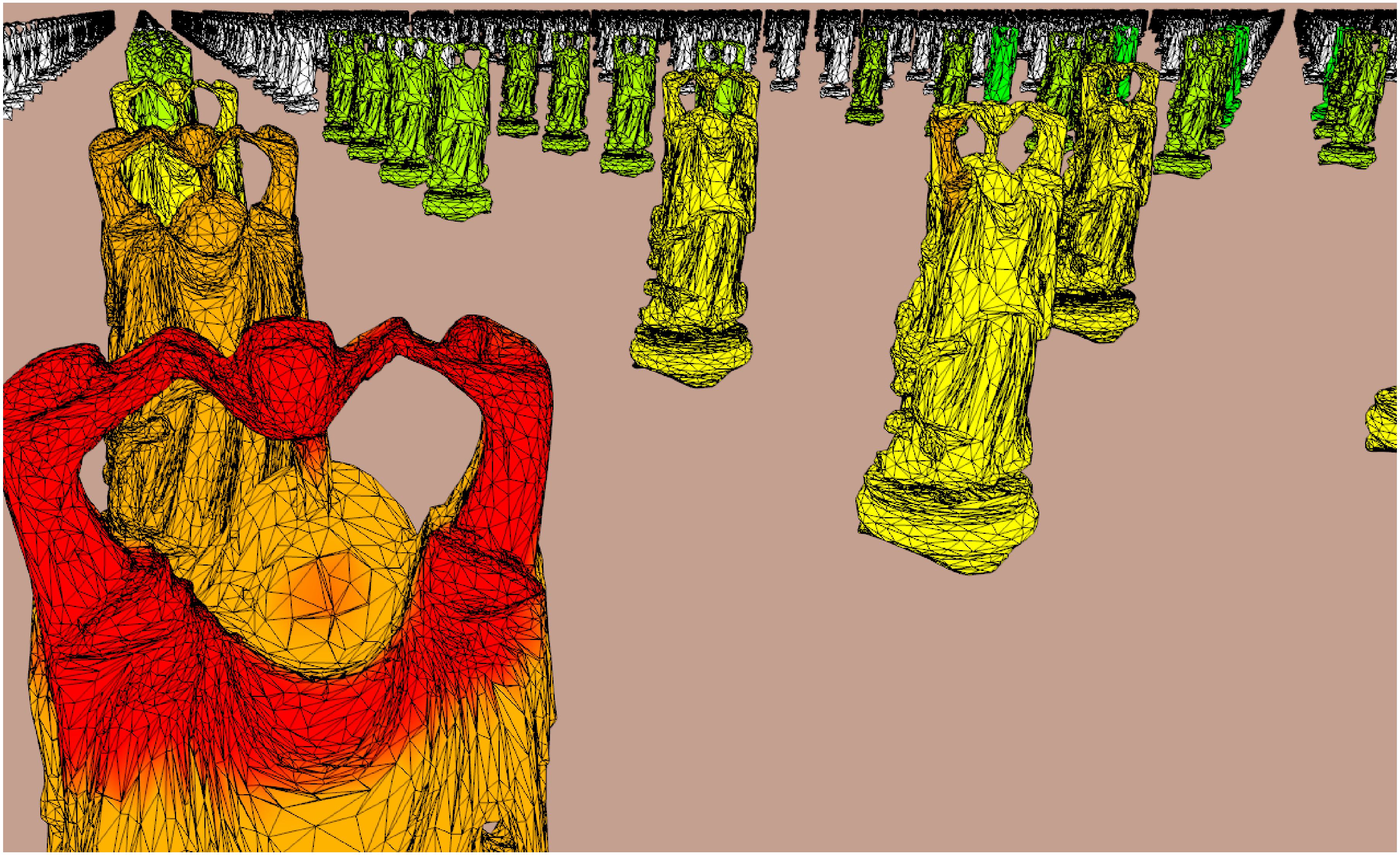
- Uses CBH to do one draw call to render shadow map

Instancing example

1600 dragons, 240M polygons



Instancing example



Possible improvements

Take advantage of new hardware features:

- With performant vertex fetch, can consider fetching all coarse vertex data (20 bytes) from textures to avoid buffer duplication
- Instead of blending between two textures, one of which simply contains an extra mip level, we can:
  - query mip level for highest LOD
  - adjust it based on blending weights with lowest LOD
  - perform a single fetch

Future work

- Hierarchical CBH textures
  (requires multiple texture coordinates)
- Animated geometry
  (need to preprocess conservative bounding spheres)
- Tiled geometry
  (need to simplify respecting boundary constraints)

Summary

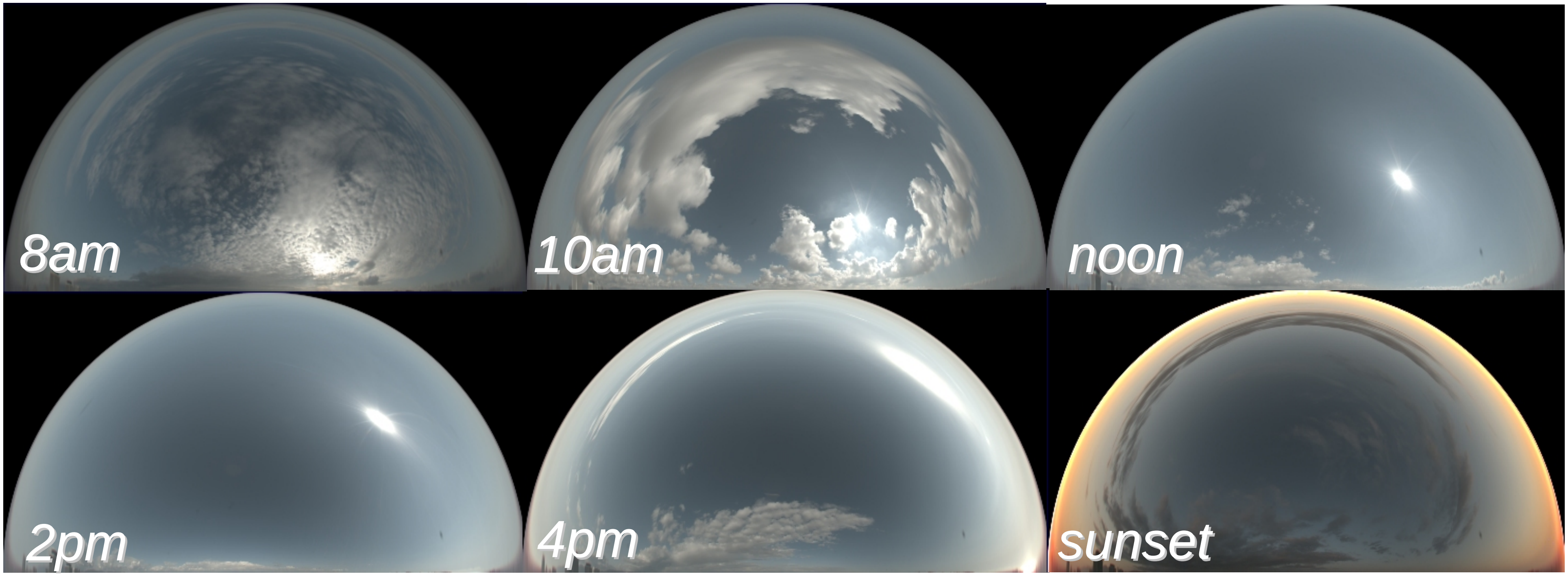
- Presented new LOD algorithm for rendering large datasets
- Features include:
  - Out-of-core rendering with prefetching
  - Texture-mapping
  - Geomorphing
  - Uses “GPU-friendly” irregular meshes
  - Only requires based shader programmability

Up Next: Video Skybox

- Progressive Buffers
- **Video skybox**
- Lighting and Rendering

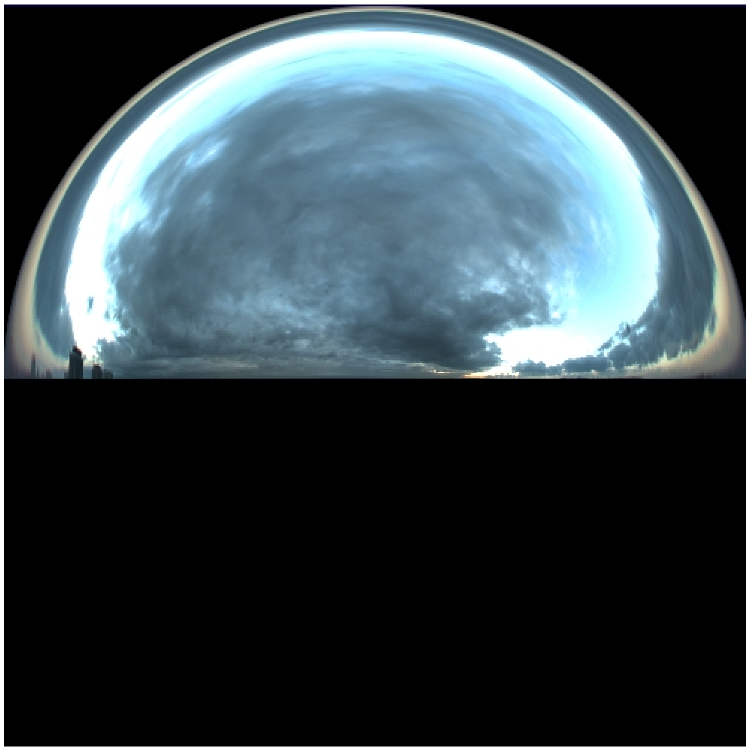
Input Skybox Imagery

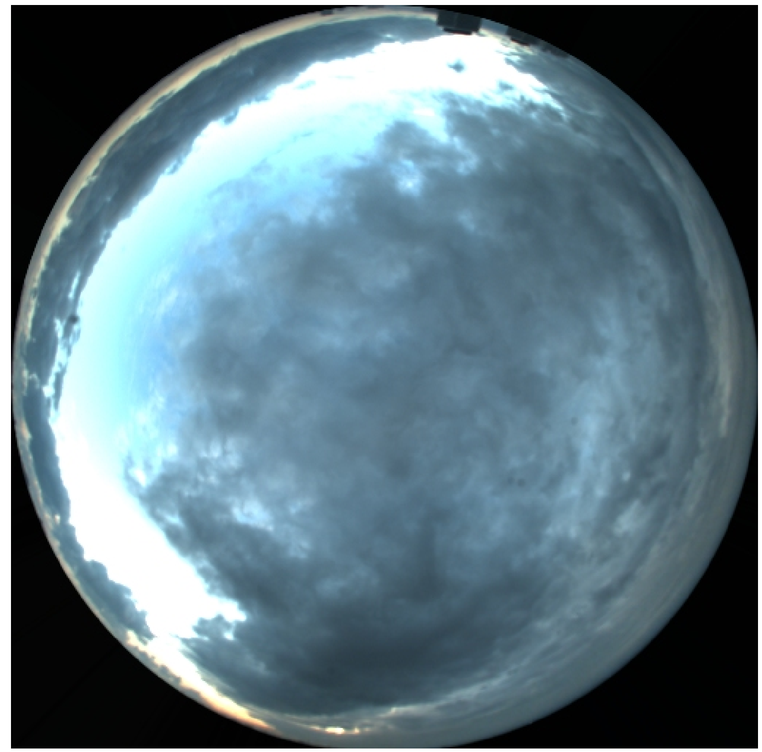
SIGGRAPH2006



- Input: 2.1GB compressed HDR imagery, captured at one minute intervals over the course of the day.
  - (670 frames of data: each frame 1024x1024 anglemap)
- Goal: we would like to compress this data in a compact and performant manner for playback and HW rendering.

Preprocessing: Resampling

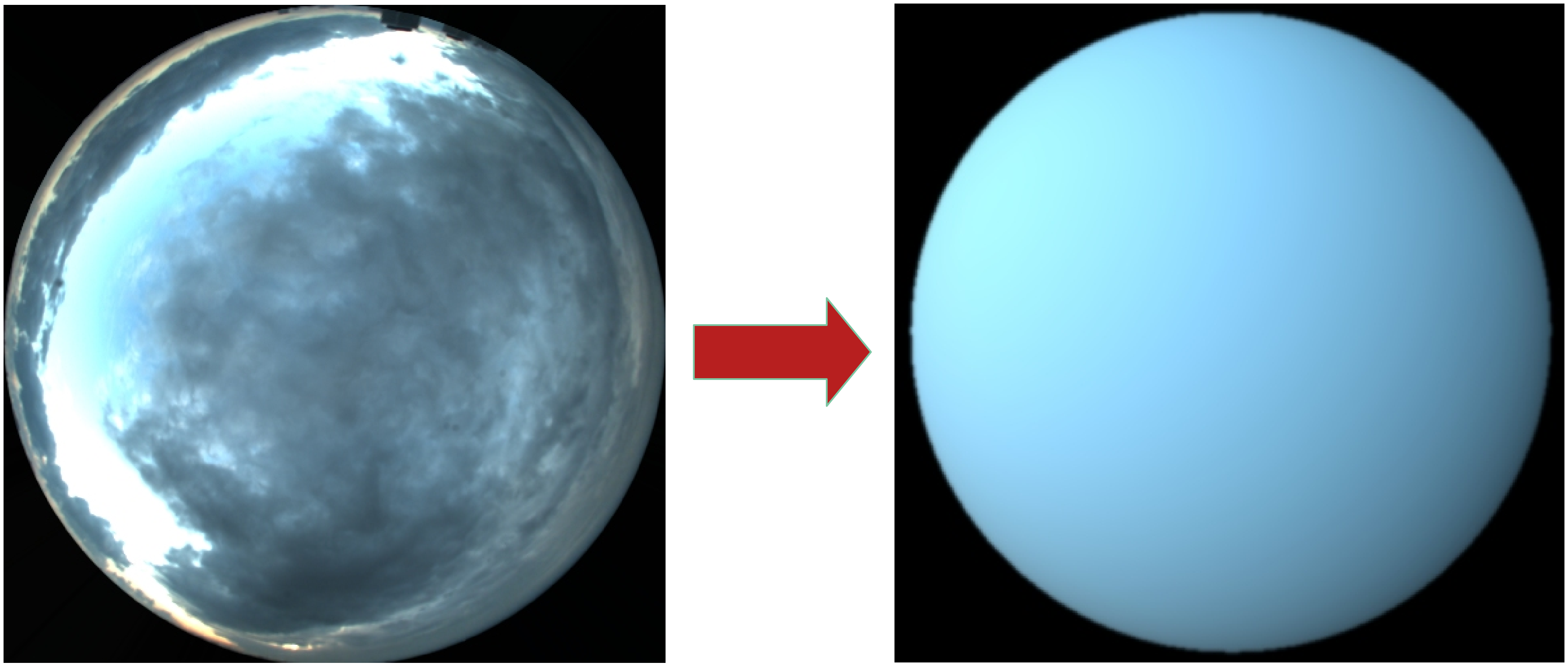




- HDR skybox re-sampled from angular mapping to a paraboloid environment map.
  - Simplifies per-pixel math for rendering the sky box only **tex2dproj** is required.

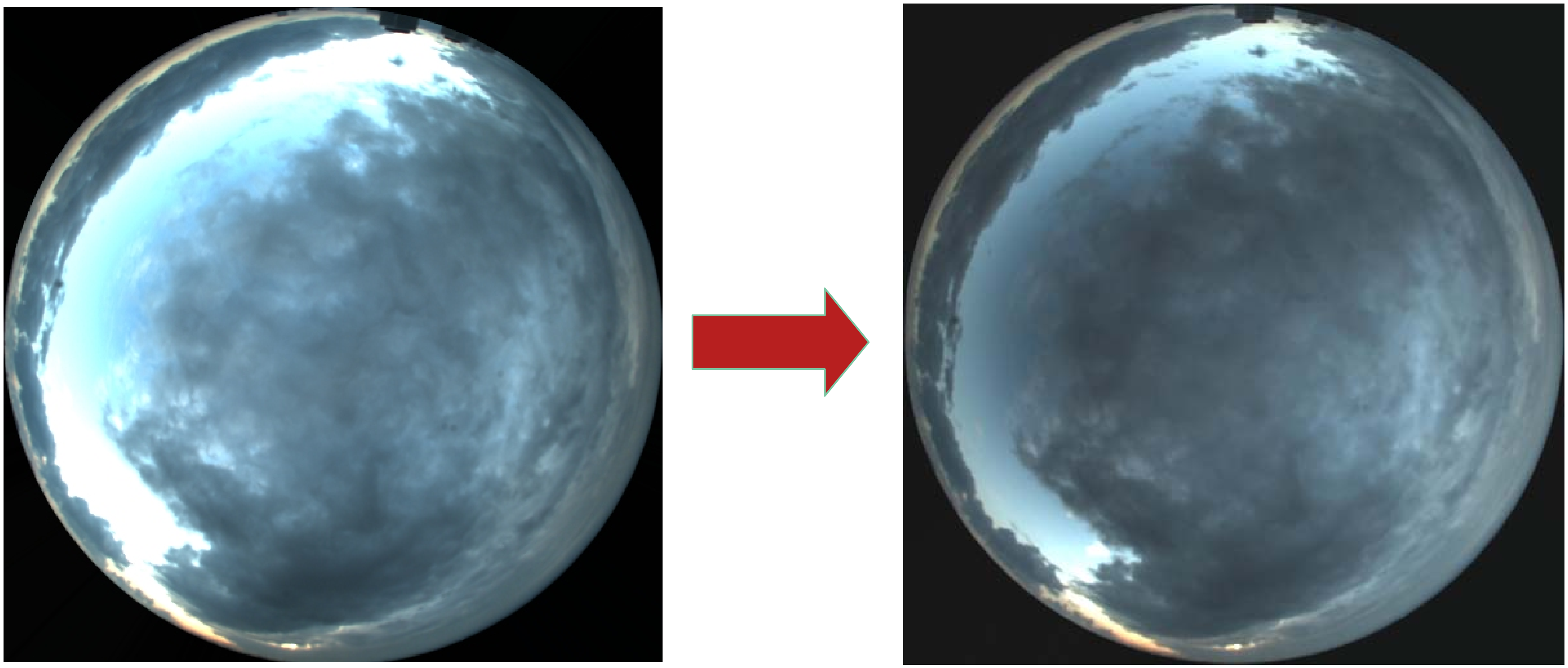
Preprocessing: SH Estimation

SIGGRAPH2006



- HDR lighting information is derived and recorded for each frame using 3rd order spherical harmonics (SH).
- This lighting information is used to provide diffuse lighting information to render the scene.

Preprocessing: Range Reduction

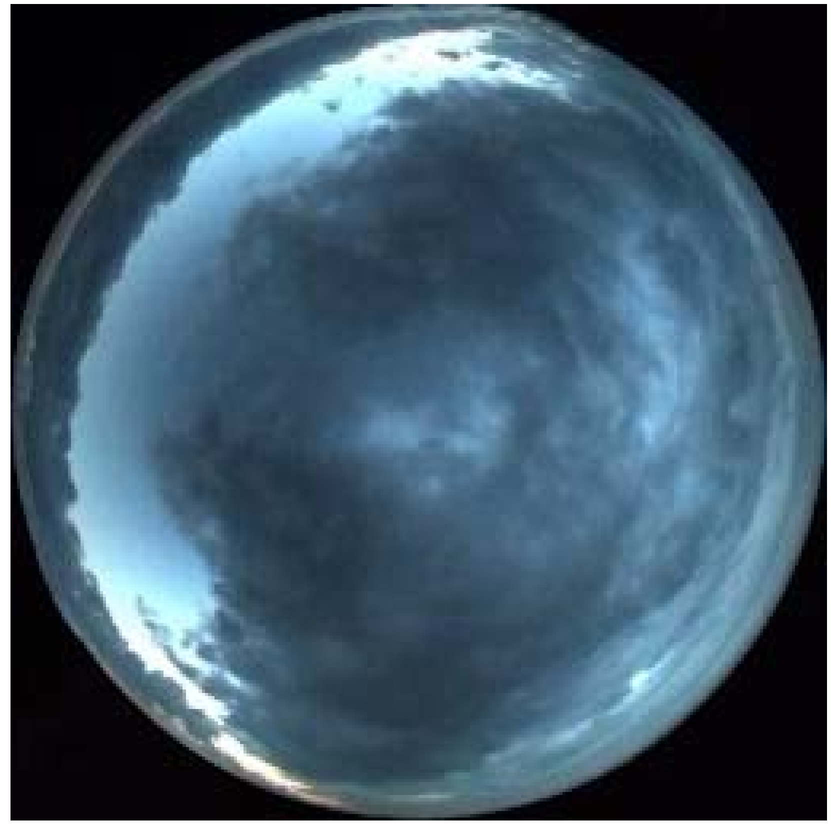


- Each frame of video is divided through by the SH representation in order to reduce the range of each frame to a 24-bit RGB image.

Encoding the Video Sequence

SIGGRAPH2006

- The reduced range video frames can be compressed using a standard video codec.
  - We found that DivX and XVid worked well.
  - 1024x1024, 670 frame sequence reduced from 2.1GB to 38.1MB
  - 72k for SH coefficients for all frames. (27 floats per frame)
  - 1MB optic flow data……



Preprocessing: Optic Flow

- Making a little video go a long way
  - 670 frames at 30fps is only @ 23 seconds of video.
  - Playback at a lower frame rate is choppy even with lerp between frames.
- Estimate optic flow between frames to estimate cloud motion.
- At runtime optic flow is used to warp one frame into the next.
  - Gives additional in-between frames, can use lower FPS video.
- Optic flow is stored as a 16x16x1024 R16G16 volume texture
  - Texture filtering for smooth interpolation between flow fields.

Video Playback

SIGGRAPH2006

- At runtime video is decoded and used as a texture.
  - Video is decompressed on the fly in its own thread.
- Optic flow based warping between frames in pixel shader.
  - Current frame is warped toward next frame.
  - Next frame is inverse warped back towards current frame.
  - Results are lerped together. (similar to morphing)
  - Flow is selectively applied (in area around sun, the flow is attenuated)
- Summary
  - Fast (two lookups, and blend)
  - Compact (DivX compression)

Up Next: Lighting and Rendering

SIGGRAPH2006

- Progressive Buffers
- Video skybox
- **Lighting and Rendering**

Ambient Lighting from Sky

- Per-vertex bent normal used to lookup into SH representation.
  - Cartesian SH evaluation used, 12 instructions for 3rd order.
- Ambient occlusion texture (half resolution) used to attenuate ambient lighting.

Direct Lighting from the Sun

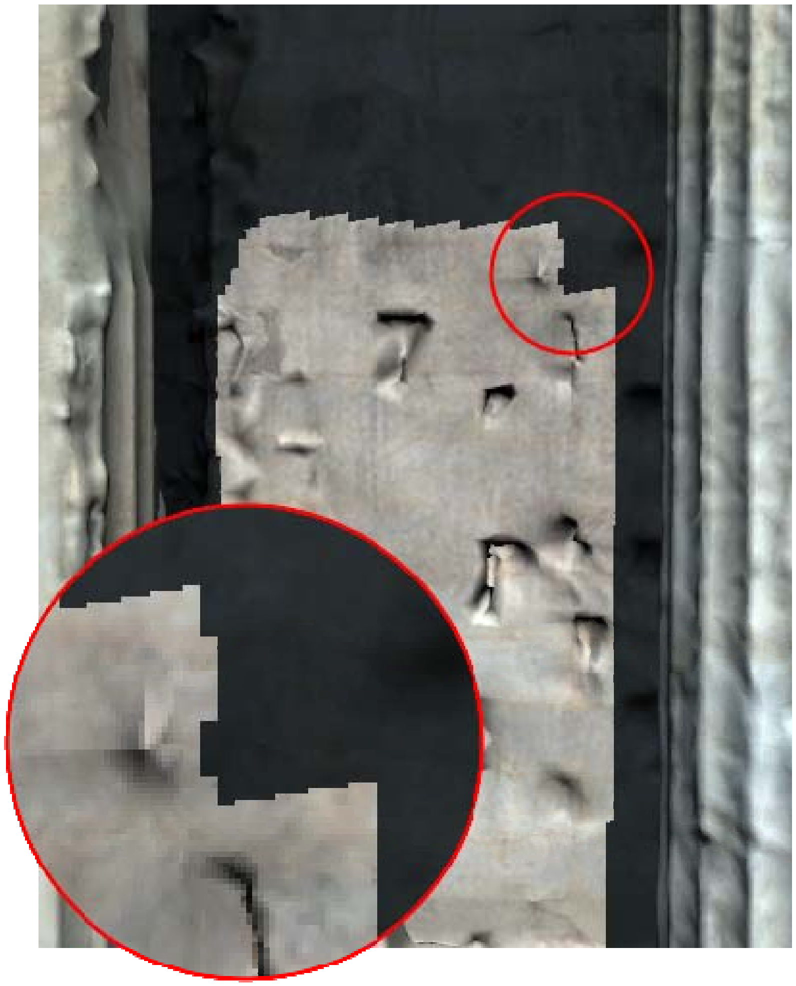
- Per frame sun color, intensity and position extracted from skybox.
- Bump mapping was only needed as detail texture.

Shadow mapping

SIGGRAPH2006

- Uses CBH for lowest level of detail to render shadow map using a single draw call.
- Multi-tap PCF w/ random rotation
- Selective post process blurring of shadow edges
  - Computation culling with early-z

Shadow mapping PCF



1 tap

4 tap PCF

Shadow mapping PCF

16 tap PCF

4 tap PCF

Reusing shadow map tests

SIGGRAPH2006

- Store shadow in alpha
- Read previously combined results from alpha (using projection matrix of previous frame)
- Recursively combine new and old results
- Store new shadow opacity value on alpha
- Display

Shadow mapping comparison

16 tap PCF

4 tap amortized

Shadow mapping comparison

4 tap amortized

4 tap PCF

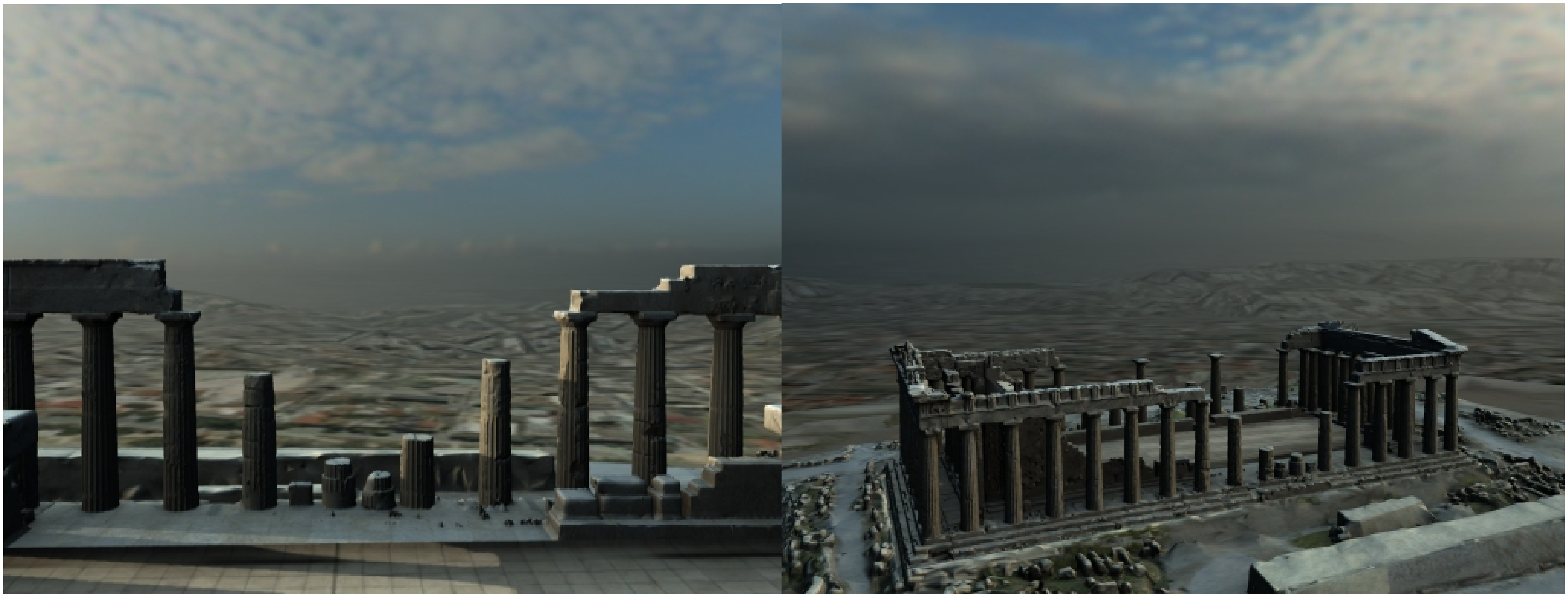
Amortized computation

- For additional details on this amortized computation, see:

- Sketch: Cache Flow session
  The Real-Time Reprojection Cache
  Thursday, 3 August
  3:45-5:00 (last talk)

Haze & Lighting for Far Geometry

SIGGRAPH2006



- Haze in distance also uses color from SH representation of sky.
  - Better match with sky color than single color haze.
  - Terrain blended using distance, sky blended using horizon.
- Sky texture used to project cloud shadows onto far geometry.

Fading in the sculptures

- Cross fade between statues in museum and on pediment.
  - Statue geometry is rendered only once with interpolation between the two different lighting conditions inside the shader.

Occlusion Query Geometry Culling

SIGGRAPH2006

- Each cluster drawn is occlusion query tested to see how many pixels get drawn for the current frame.
  - If any pixels get drawn, the voxel is flagged to be drawn next frame.
  - If no pixels get drawn, the next frame an inexpensive ‘probing’ quad is rendered with color and Z writes disabled instead of the voxel.

Overview

- Progressive Buffers
- Video skybox
- Lighting and rendering

Acknowledgements

- John Isidoro, Jason Mitchell, and Josh Barczak for participating in the development and implementation of the parthenon demo.
- Eli Turner for his work on the datasets and all the additional original artwork that went into this demo.
- Paul Debevec and Andrew Jones for providing the Parthenon dataset and HDR skybox imagery

- For more information on Progressive Buffers see:

[Sander05] P. V. Sander and J. L. Mitchell, “Progressive Buffers: View Dependent Geometry and Texture LOD”, *Symposium on Geometry Processing 2005*

Thank you!

- Thank you for attending!