Volume Rendering from Difficult Data Formats

Klaus Engel
Siemens Corporate Research
Princeton

Markus Hadwiger
VR VIS Research Center
Vienna, Austria

Joe M. Kniss
Scientific Computing and
Imaging Insitute,
University of Utah

Aaron E. Lefohn
Institute for Data Analysis
and Visualization
University of California, Davis

Christof Rezk Salama
Computer Graphics and
Multimedia Group
University of Siegen, Germany

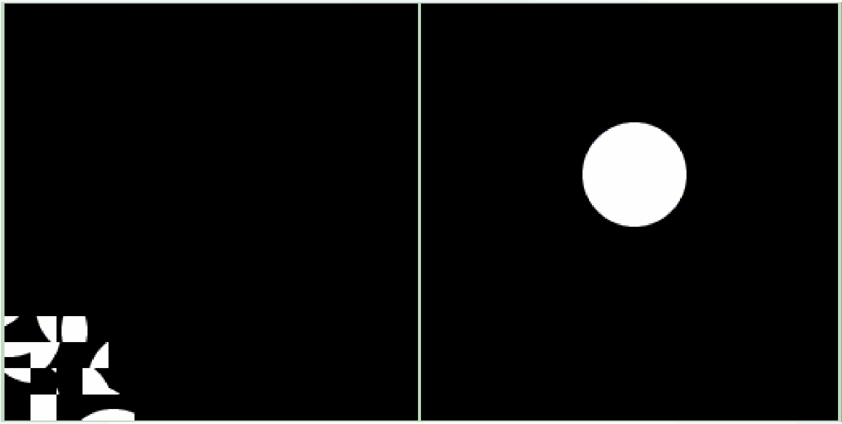
Daniel Weiskopf
Visualization and Interactive
Systems Group,
University of Stuttgart, Germany

Overview

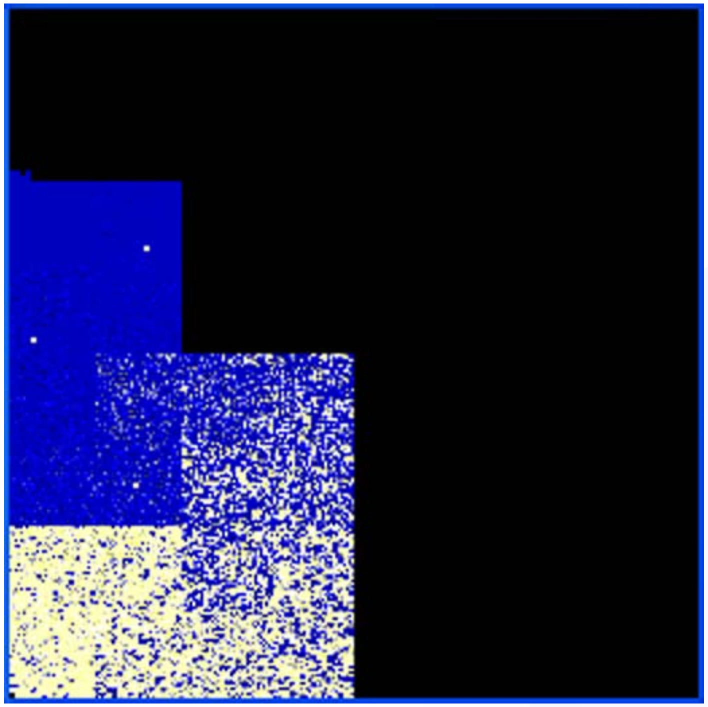
- What is a Difficult Data Format?
- Why Difficult Data Formats?
- The "Difficult Data" Rendering Pipeline
  - Reconstruction of volume data
  - Filtering
  - Transfer function and lighting
- Conclusions

Motivation

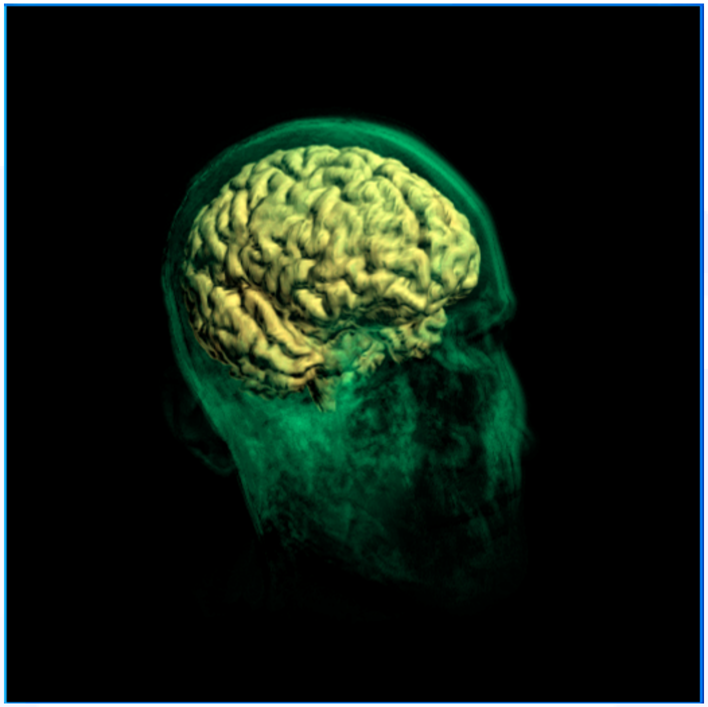
- What is a *Difficult Data Format*?
  - A data format that differs from the 3D domain on which the volume rendering algorithm is defined
  - Examples
    - Compressed data
    - Sparse data
    - Set of 2D slices



Difficult-Data Volume Rendering



Compressed Data Format

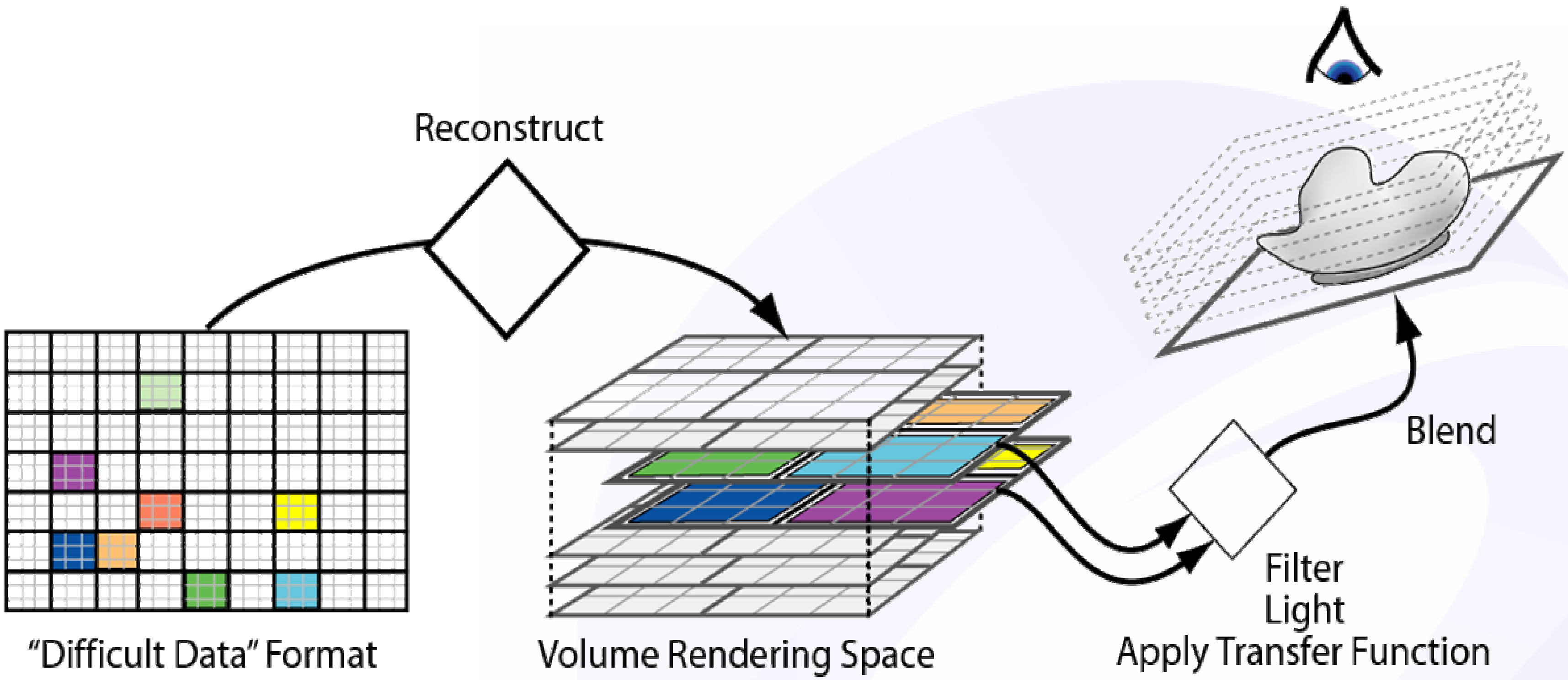


Interactive VolumeRendering

Motivation

- Why *Difficult Data Formats?*
  - Data compression
    - On-the-fly decompression of large data sets
  - Acceleration of volume rendering
    - Store only the “important“ data in a packed format
  - GPU-generated data (GPGPU computation)
    - Set of 2D dynamic textures (pbuffers)

Rendering Pipeline Overview

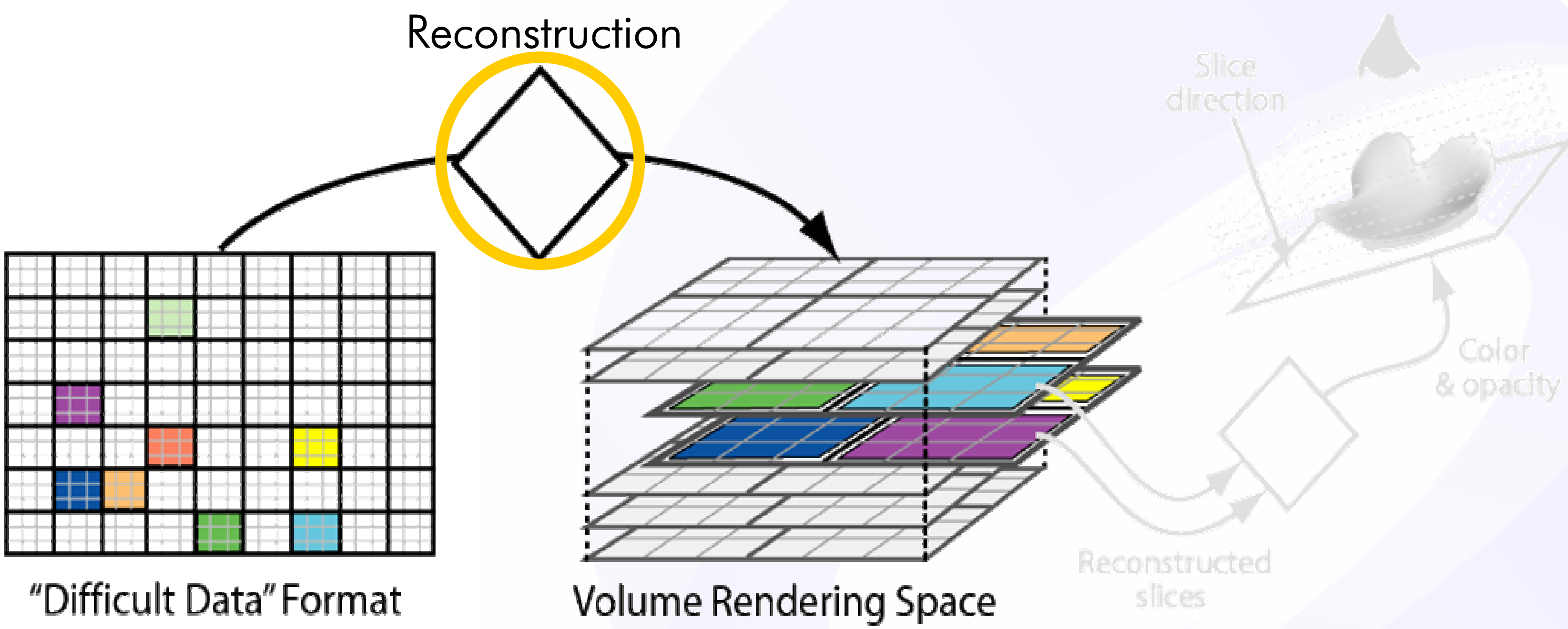


Overview

- What is a Difficult Data Format?
- Why Difficult Data Formats?
- The “Difficult Data“ Rendering Pipeline
  - Reconstruction of volume data
  - Filtering
  - Transfer function and lighting
- Conclusions

Rendering: Reconstruction

- Idea:
  - Map data from “difficult-data space“ to volume rendering space



Rendering: Reconstruction

- Three Basic Types of Mappings
  - Analytic
    - Mapping is procedurally computed on-the-fly
      - Ex: Heirarchical bricking

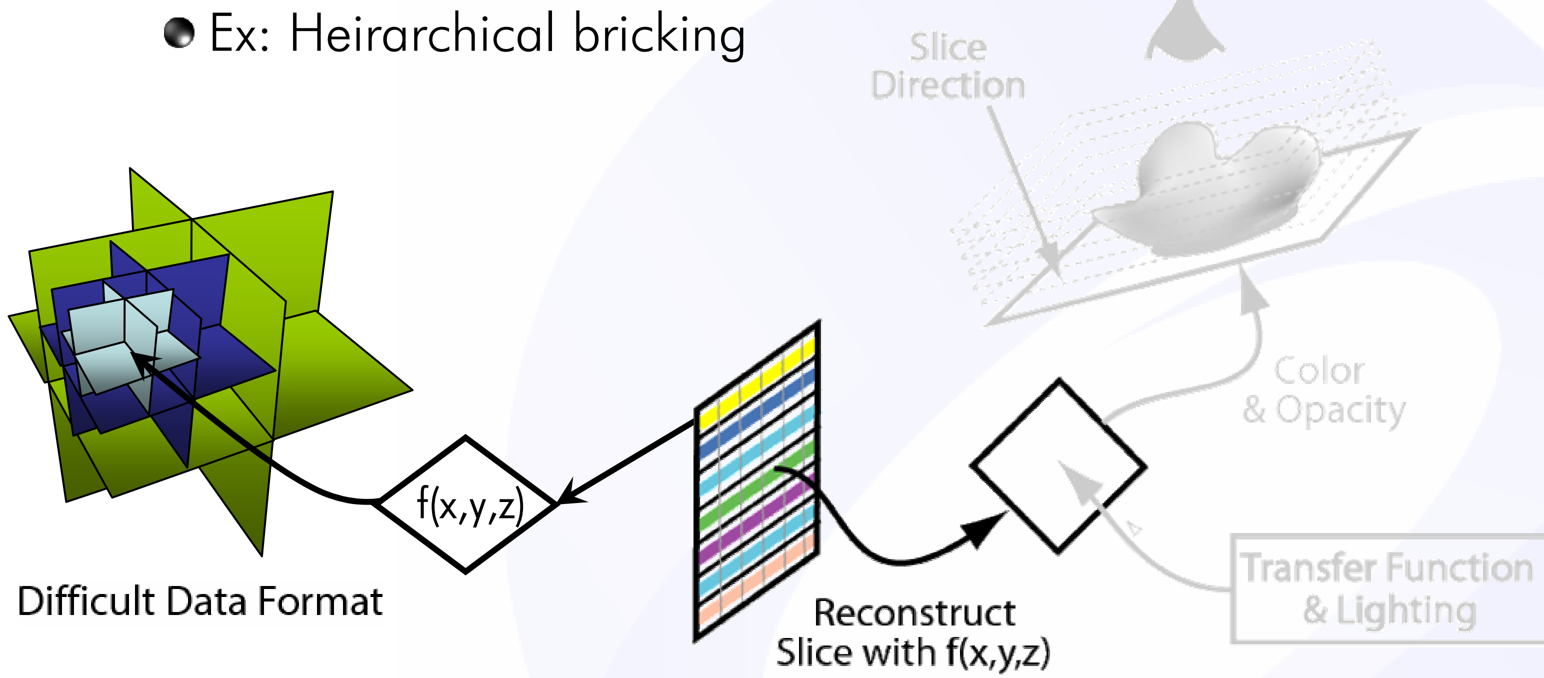
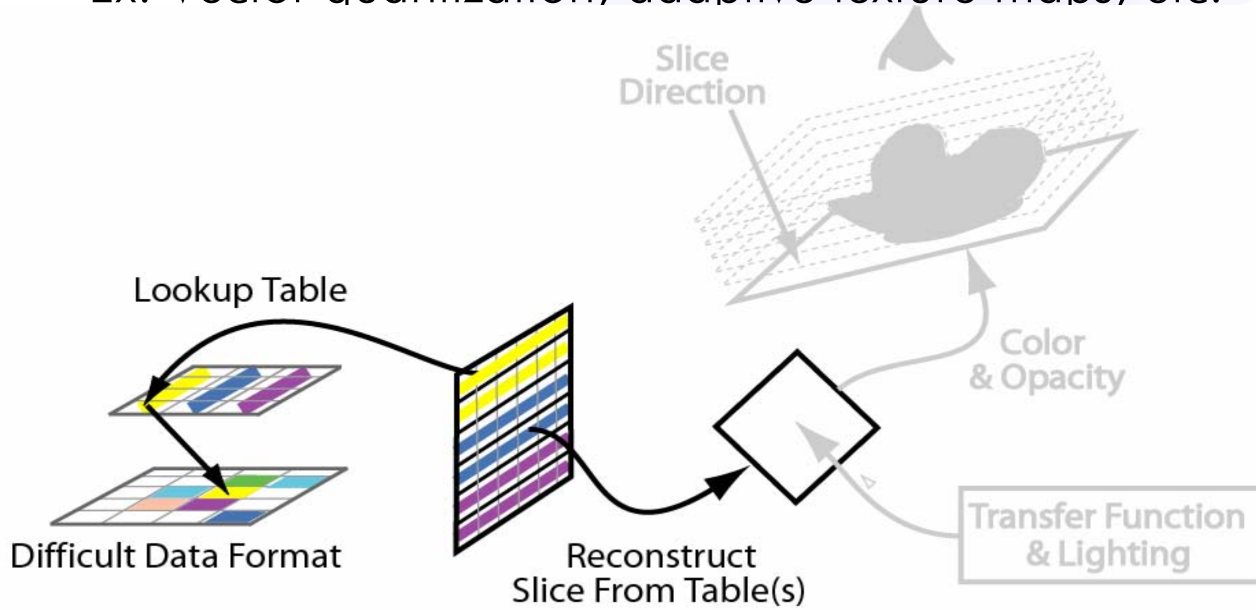


Figure Courtesy of Klaus Engel

Aaron E. Lefohn
Institute for Data Analysis and Visualization, UC Davis, USA

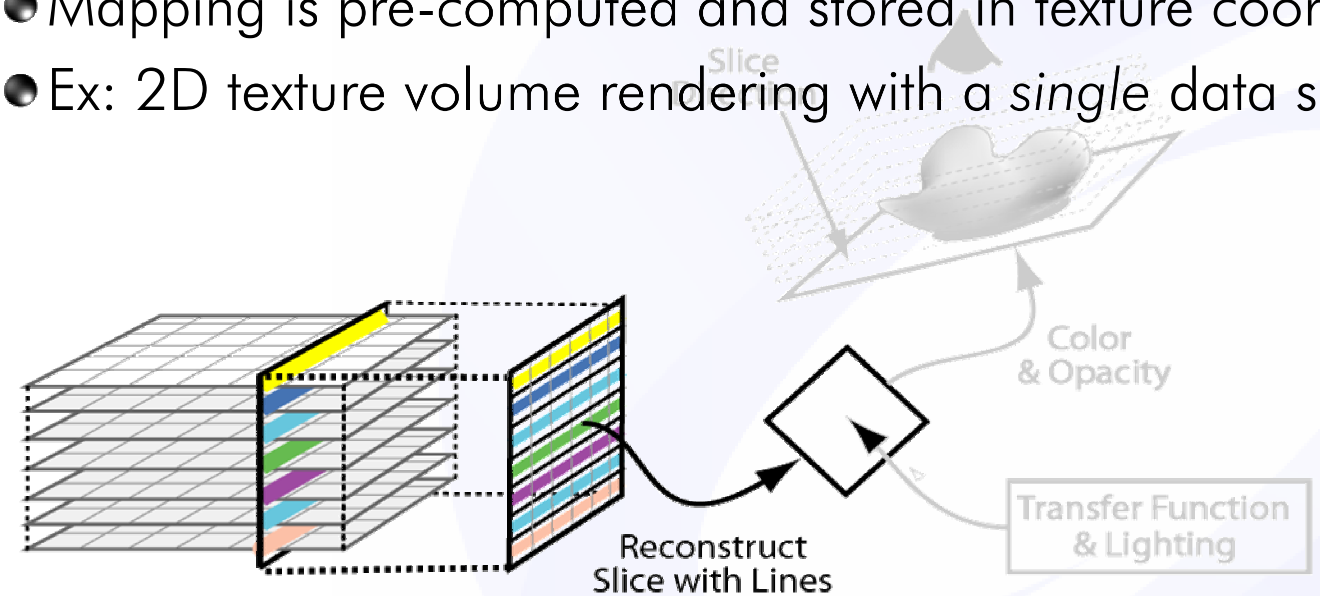
Rendering: Reconstruction

- Three Basic Types of Mappings
  - Analytic
  - Table-Based
    - Mapping is stored in GPU-based table(s)
      - Ex: Vector quantization, adaptive texture maps, etc.



Rendering: Reconstruction

- Three Basic Types of Mappings
  - Analytic
  - Table-Based
  - Geometry-Based
    - Mapping is pre-computed and stored in texture coordinates
    - Ex: 2D texture volume rendering with a *single* data set



Reconstruction Conclusion

- Data reconstruction is the first step in all difficult-data volume rendering algorithms
  - Mapping from rendering-space to data-space
  - Analytic, table-based, or geometry-based

Overview

- What is a Difficult Data Format?
- Why Difficult Data Formats?
- The “Difficult Data“ Rendering Pipeline
  - Reconstruction of volume data
  - Filtering
  - Transfer function and lighting
- Conclusions

Rendering: Filtering

- Construct Continuous Signal from Discrete Data

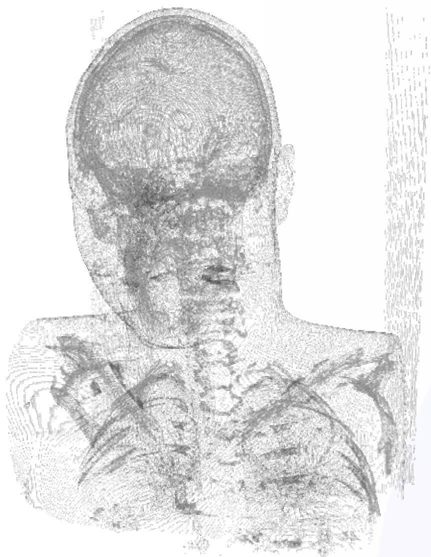
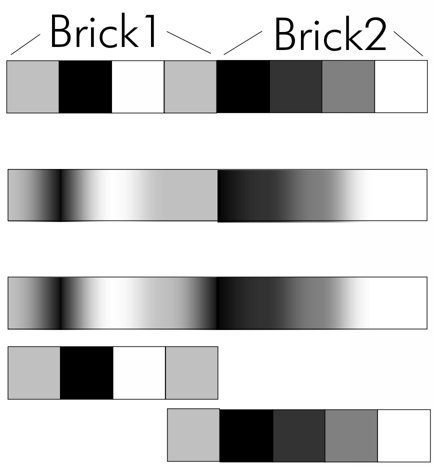


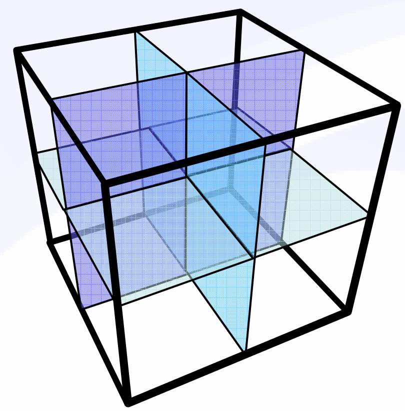
Figure Courtesy of Klaus Engel

- Achilles' heal of difficult-data volume rendering
  - Native GPU filtering cannot be used in many cases
  - Many difficult-data techniques provide no filtering

Rendering: Filtering

- In Which Space Should We Filter?
  - Difficult-data domain
    - Data format must contain contiguous regions of the original volume (e.g. sub-volume bricks, etc.)



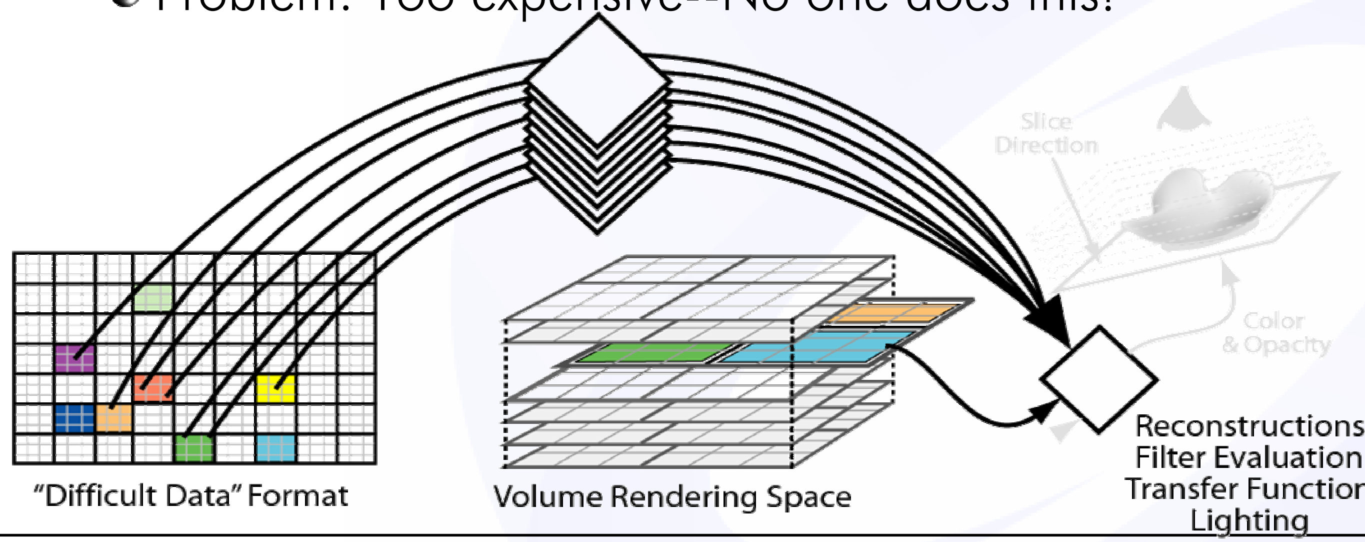


    - Over-represent data by width of filter kernel
    - Use native GPU trilinear filtering

Figure Courtesy of Klaus Engel

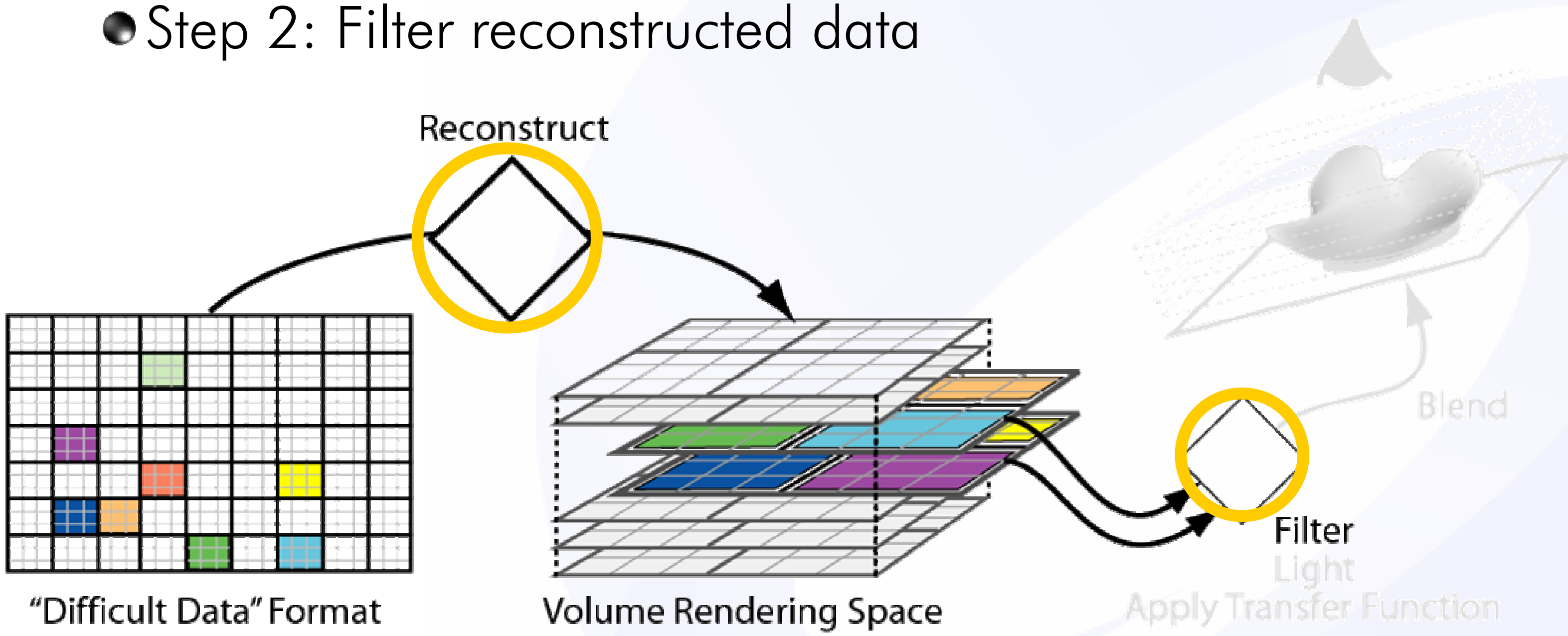
Rendering: Filtering

- In Which Space Should We Filter?
  - Difficult data domain
  - Reconstructed rendering domain
    - Required if data is not “over-represented sub-volumes“
    - Must reconstruct a data point for each filter sample
      - Problem: Too expensive--No one does this!



Solution: Deferred Filtering

- Idea
  - Similar to deferred shading and separable convolution
  - Separate reconstruction and filtering steps
    - Step 1: Reconstruct original discrete data
    - Step 2: Filter reconstructed data

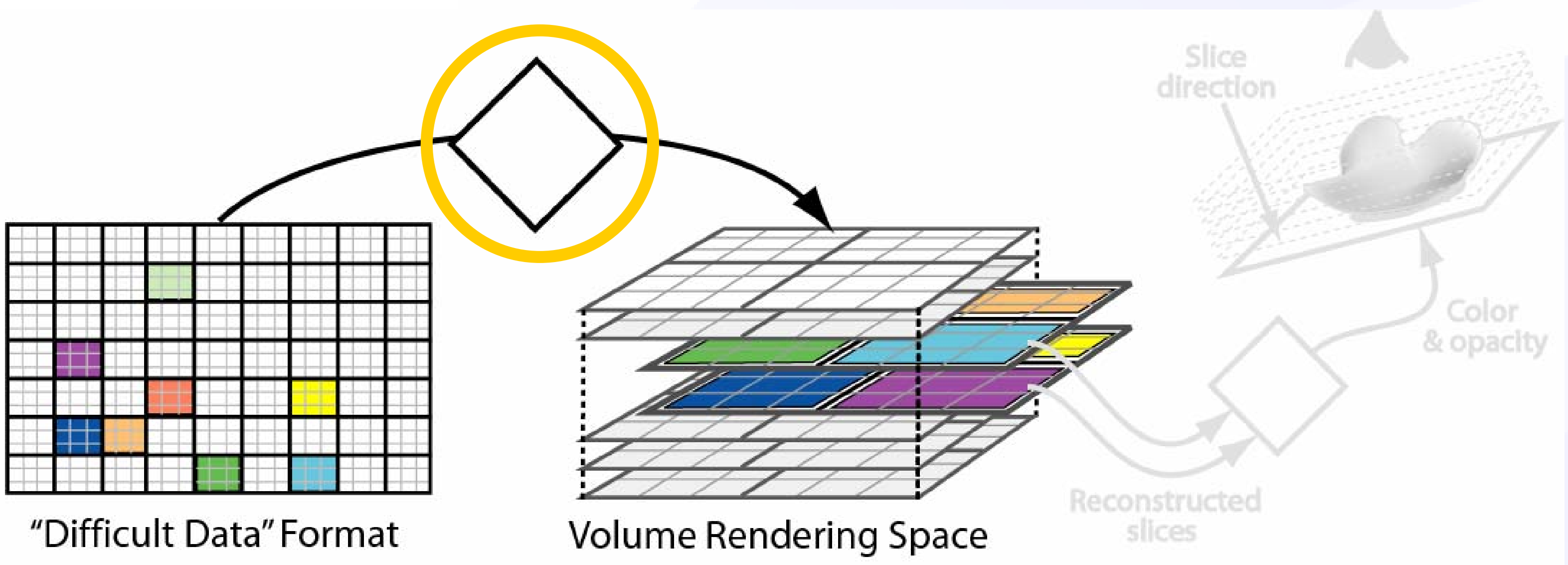


Deferred Filtering: Why?

- Fast
  - Avoid redudant data reconstruction computations
  - Use native GPU filtering
- Modular
  - Works with any reconstruction algorithm

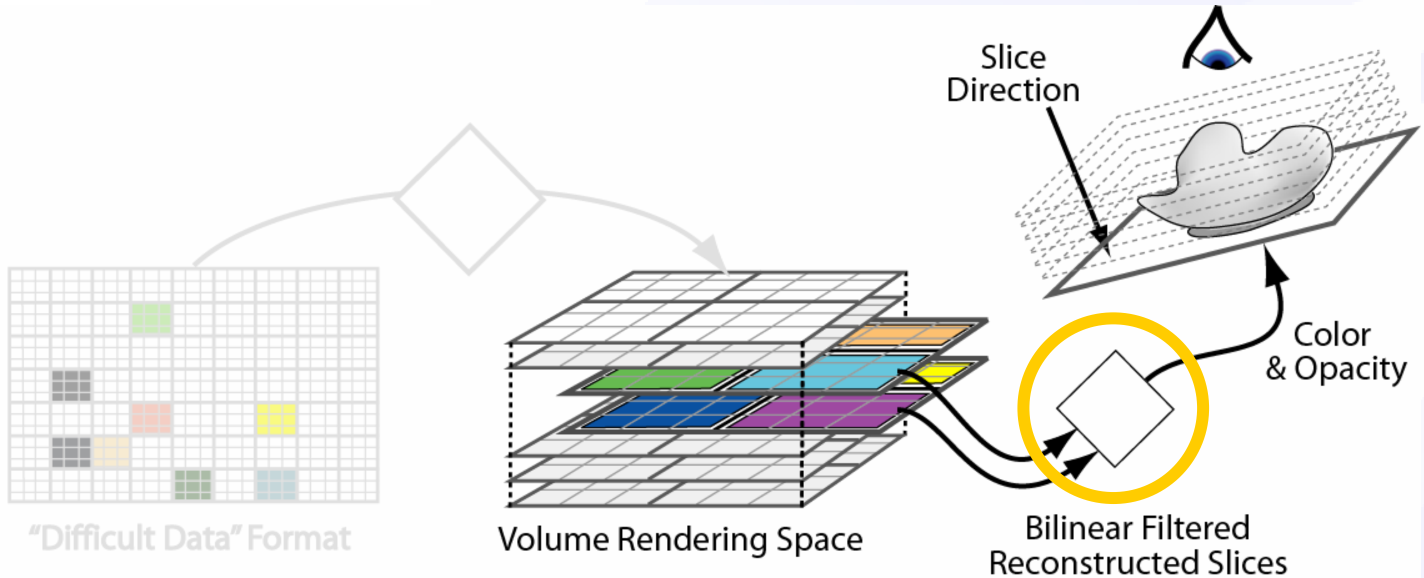
Deferred Filtering

- Algorithm
  - Pass 1:
    - Reconstruct data for two adjacent slices
    - Save results to textures/pbuffers



Deferred Filtering

- Implementation
  - Pass 2:
    - Read data from saved slices
      - Both samples are bilinearly filtered by GPU
    - Compute z-interpolation in fragment program



Deferred Filtering: Cost Analysis

- Let's Run Some Numbers…
  - Nr = Number instructions to reconstruct one value 4
  - Nf = Number of samples in filter 8
  - Ni = Number of instructions for trilinear interp 14
  - Nd= Number of elements in data space 256^3
  - Nv = Number of elements in rendering space 512^3

- Non-Deferred Filtering
  (Nr * Nf + Ni) * Nv = 6.2 Billion

  Deferred Filtering
  (Nr * Nd) + Nv = 0.6 Billion

- Deferred filtering gives nearly ~10x fewer instructions!
- Real-world example shows ~7x speedup (Nathan Fout)

Deferred Filtering Conclusions

- Pros
  - Enables filtering with any data reconstruction algorithm
  - Avoids redundant data reconstruction (fast)
  - Leverages GPU's native bilinear filtering (fast)

- Cons
  - Partitioning shader at reconstruction/filter call
  - Overhead of extra render pass (??)

- Reference
  - "A Streaming Narrow-Band Algorithm: Interactive Computation and Visualization of Level Sets," Lefohn, Kniss, Hansen, Whitaker, TVCG July/August 2004

Filtering Conclusions

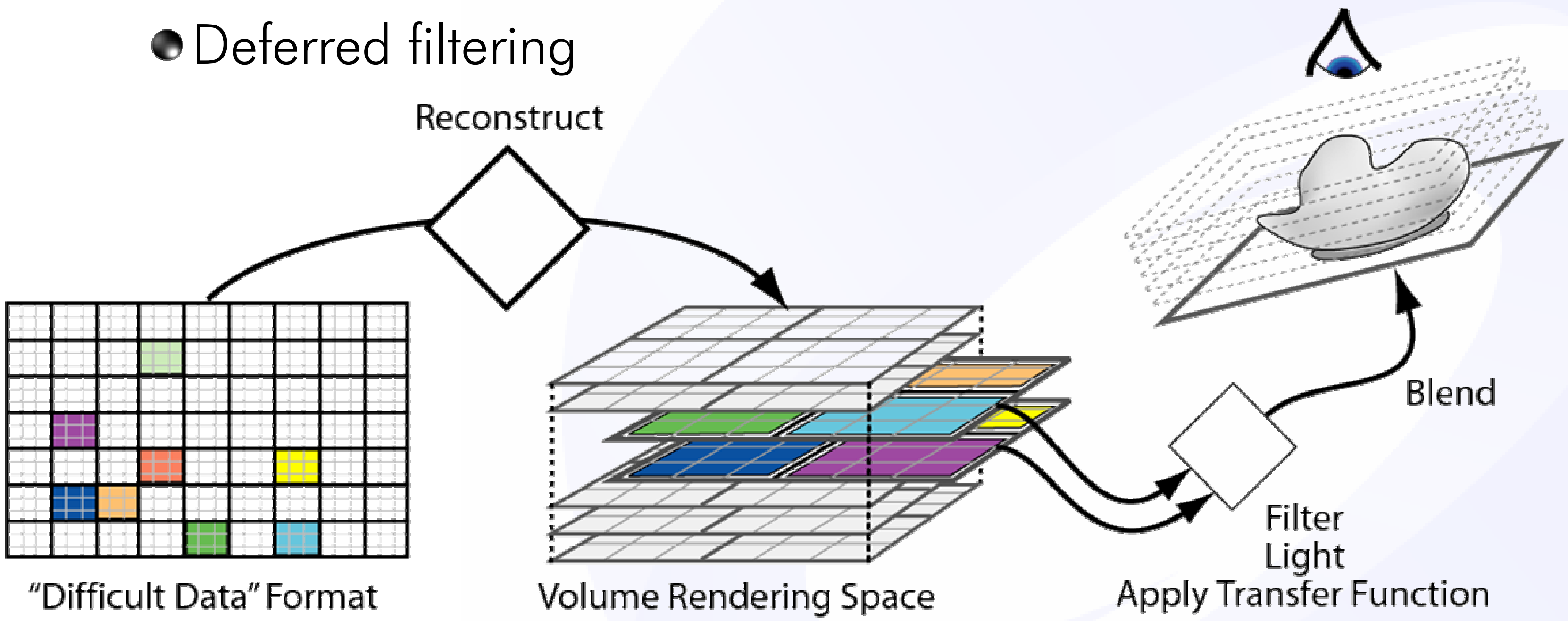
- If Data Stored as "Over-Represented Sub-Volumes"
  - Reconstruct and filter data simultaneously
    - GPU's native trilinear filtering
    - Other filtering methods discussed in this course
- Else
  - Use deferred filtering
    - Pass 1: Reconstruct data for two adjacent slices
    - Pass 2: Filtering
      - Trilinear
        - GPU's native bilinear filtering with in-shader z-interpolation
      - Other filtering methods discussed in this course

Overview

- What is a Difficult Data Format?
- Why Difficult Data Formats?
- The “Difficult Data“ Rendering Pipeline
  - Reconstruction of volume data
  - Filtering
  - Transfer function and lighting
- Conclusions

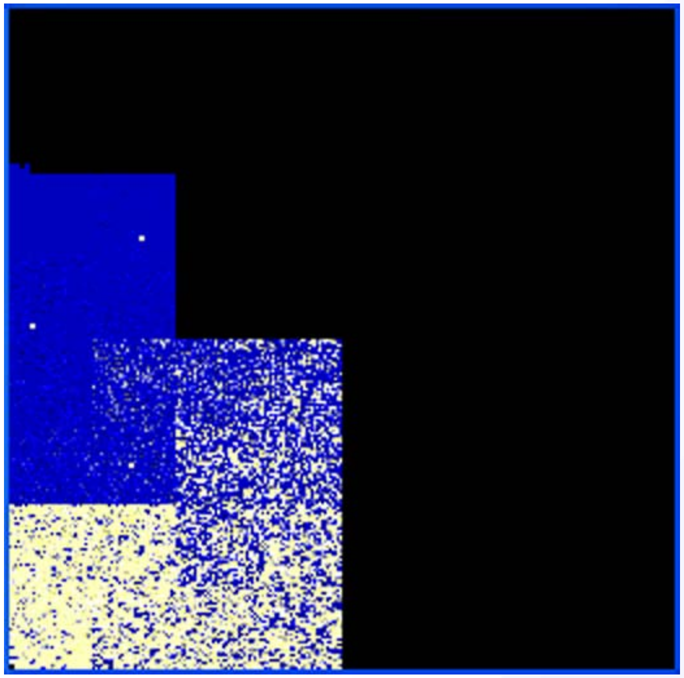
Transfer Function and Lighting

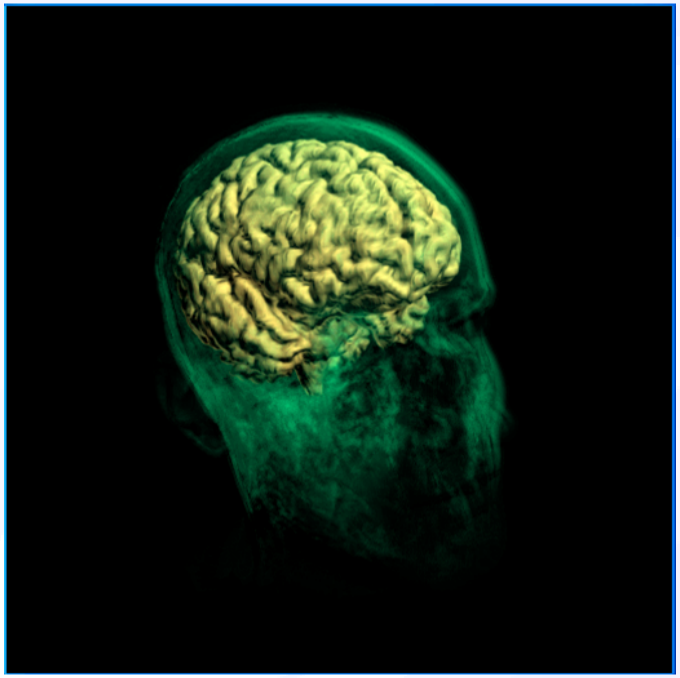
- Nothing Changes!
  - Exactly the same techniques discussed throughout course
  - Difficult data formats simply add two steps to the pipeline
    - Reconstruction
    - Deferred filtering



Rendering

- Putting it all together
  - Reconstruction
  - Filtering
  - Transfer function and lighting





Conclusions

- Volume Rendering from Difficult Data Formats
  - Adds two steps to volume rendering pipeline
    - Reconstruction
    - Deferred filtering
  - Enables
    - Acceleration strategies
    - Data compression: Rendering large data sets
    - Interactive rendering of 3D GPGPU computations
  - Increasingly important as more powerful GPUs permit more complex on-the-fly reconstruction schemes

Acknowledgements

- Joe Kniss, Ph.D. student, SCI Institute, Univ. of Utah
- The other Real-time Volume Graphics course presenters
- Nathan Fout, Ph.D. student, Univ. Of California Davis

- Ross Whitaker, M.S. advisor, SCI Institute, Univ. of Utah
- John Owens, Ph.D. advisor, Univ. of California Davis
- Evan Hart, Mark Segal, Jason Mitchell at ATI Technologies, Inc.

- National Science Foundation Graduate Fellowship
- Office of Naval Research grant #N000140110033
- National Science Foundation grant #ACI008915 and #CCR0092065
- Pixar Animation Studios, summer internships
- ALIENWARE Interchangeable mobile GPUs

Questions?

- Thank you!

- For more information
  - Google “Lefohn GPU“
  - http://graphics.cs.ucdavis.edu/~lefohn/