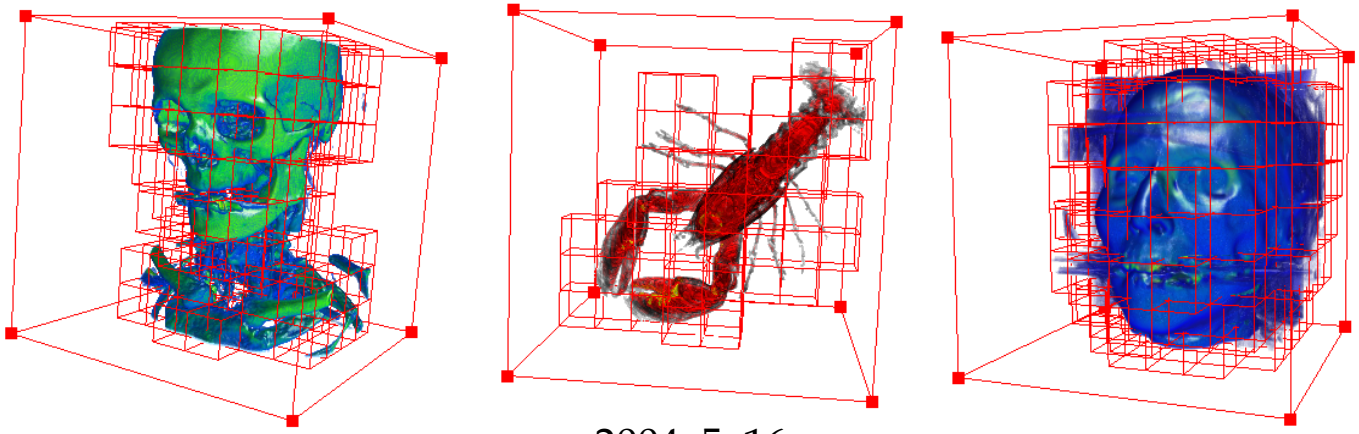
A Bandwidth Reduction Scheme for 3D Texture-based Volume Rendering on Commodity Graphics Hardware



2004. 5. 16

Won-Jong Lee, Woo-Chan Park, Jung-Woo Kim,
Tack-Don Han, Sung-Bong Yang, and Francis Neelamkavil

Media System Lab. Yonsei University, Seoul, Korea
email : airtight@kurene.yonsei.ac.kr
www : http://airtight.yonsei.ac.kr

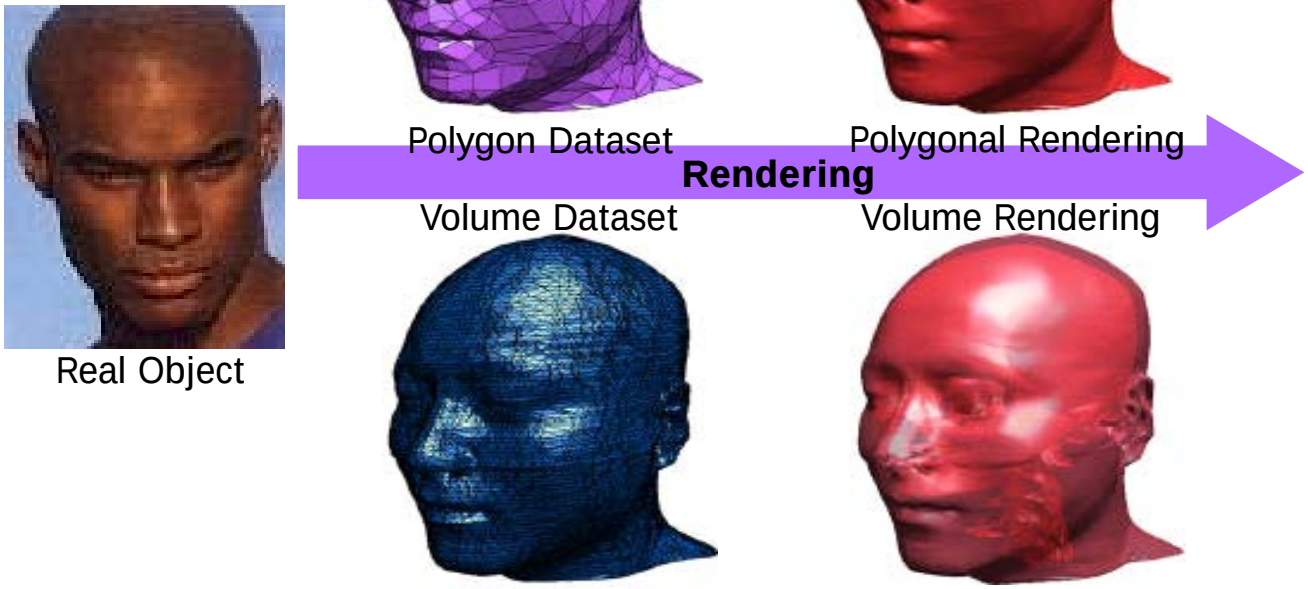
Contents

- Introduction
  - Why Volume Rendering?
  - Applications
  - 3D Texture Mapping and Its Performance Limitation
- Bandwidth-Effective Sub-Volume Rendering
  - Preprocessing Step
  - Sub-Volume Rendering and Blending
- Simulation Results
- Conclusion & Future Work

Introduction

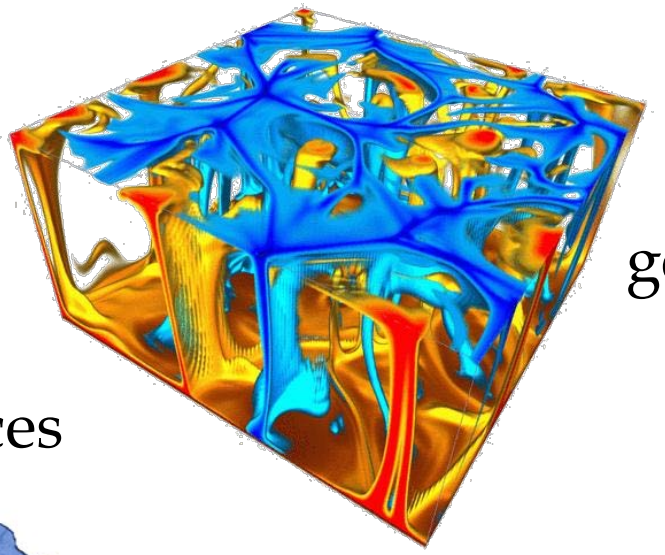
Volume Graphics

- Why Volume Rendering?
  - Overcome limitation of Polygon-Based Rendering
  - Volume rendering can generate photorealistic and non-photorealistic scenes



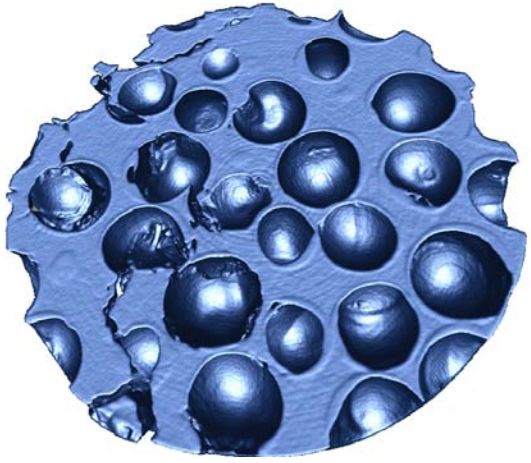
Applications

- Scientific Visualization



geology

material sciences



medical visualization

Applications

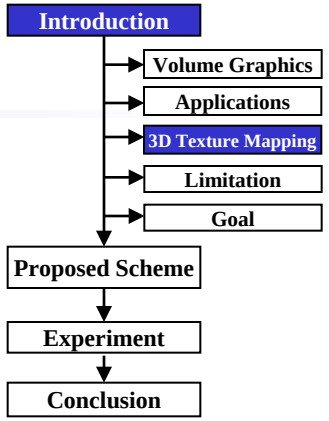
- Entertainments

atmospheric effects

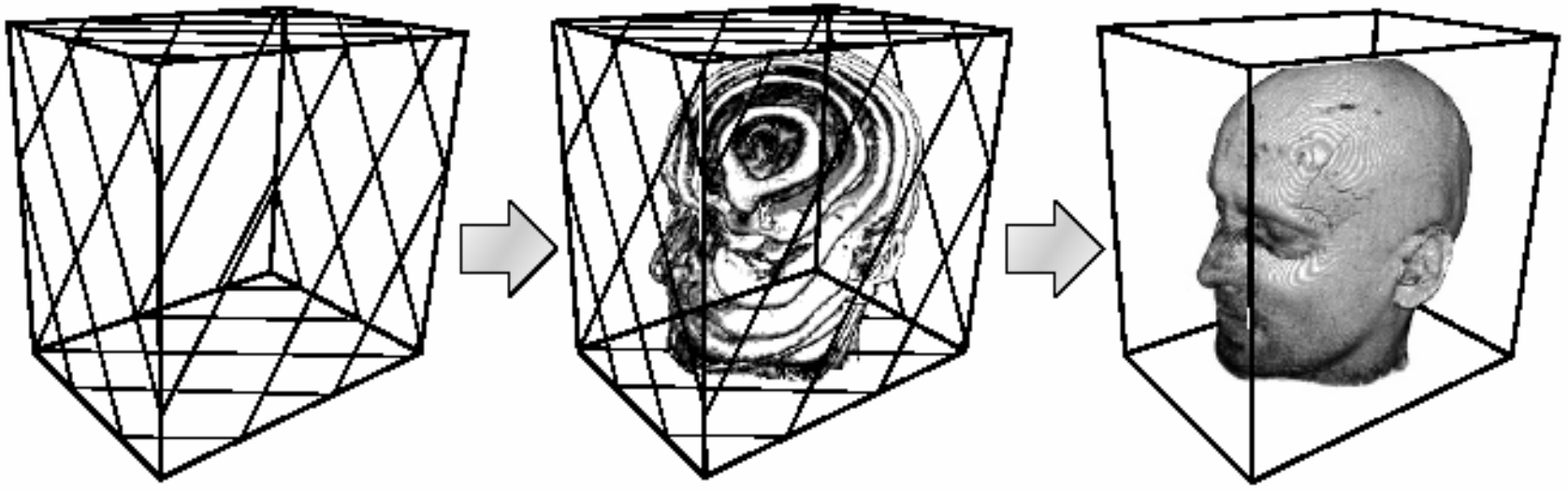
explosions, FX

volumetric surfaces

Volume Rendering with 3D Texture Mapping



- 3D Texture Mapping
  - DVR (Direct Volume Rendering) method
  - Supported by recent GPU (GeforceFX, RADEON 9800) and standard graphics API (OpenGL, Direct3D)
  - Slicing → Mapping (Sampling) → Blending

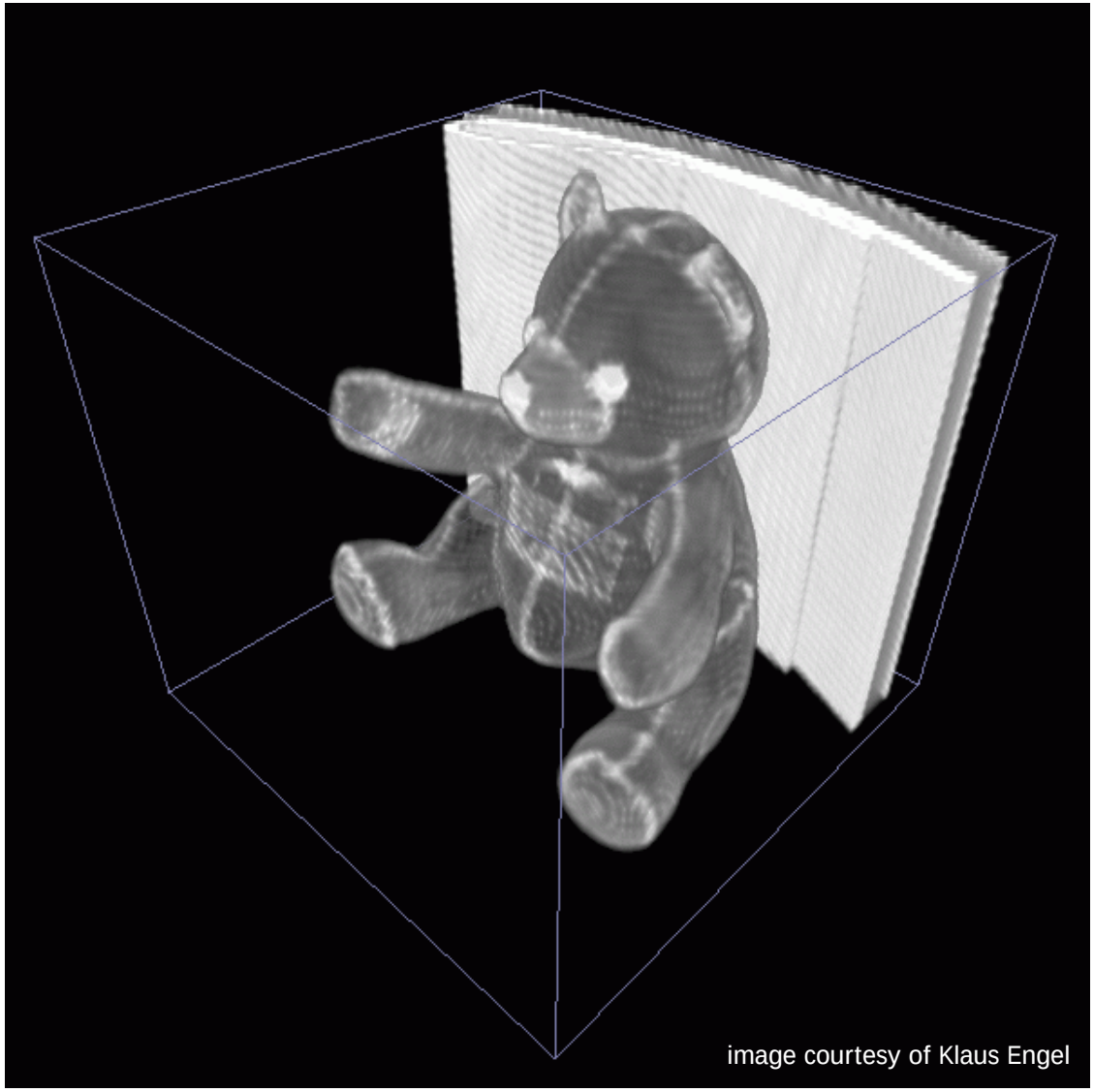


Dynamic Slicing

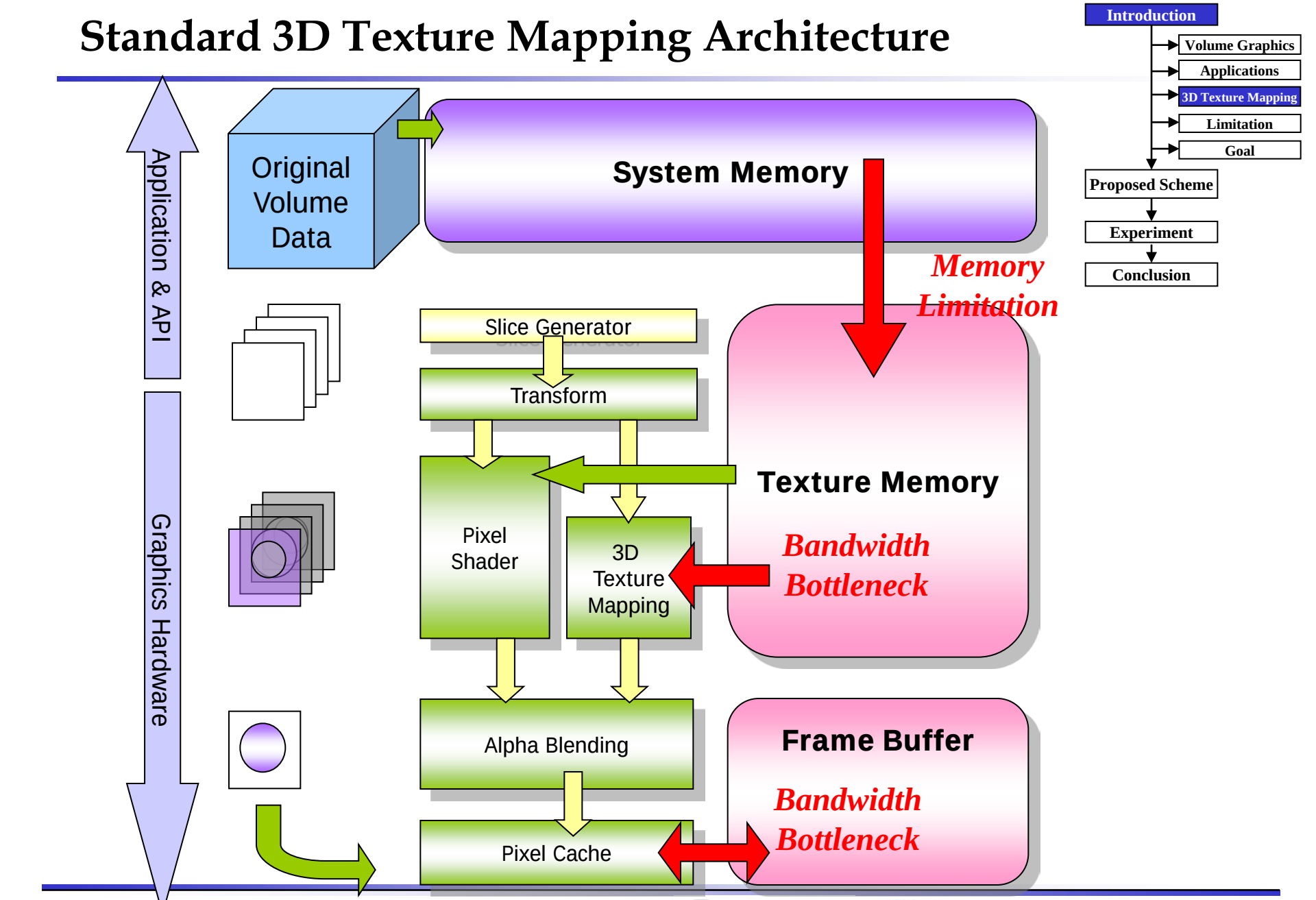
3D Texture Sampling and Mapping

Blending

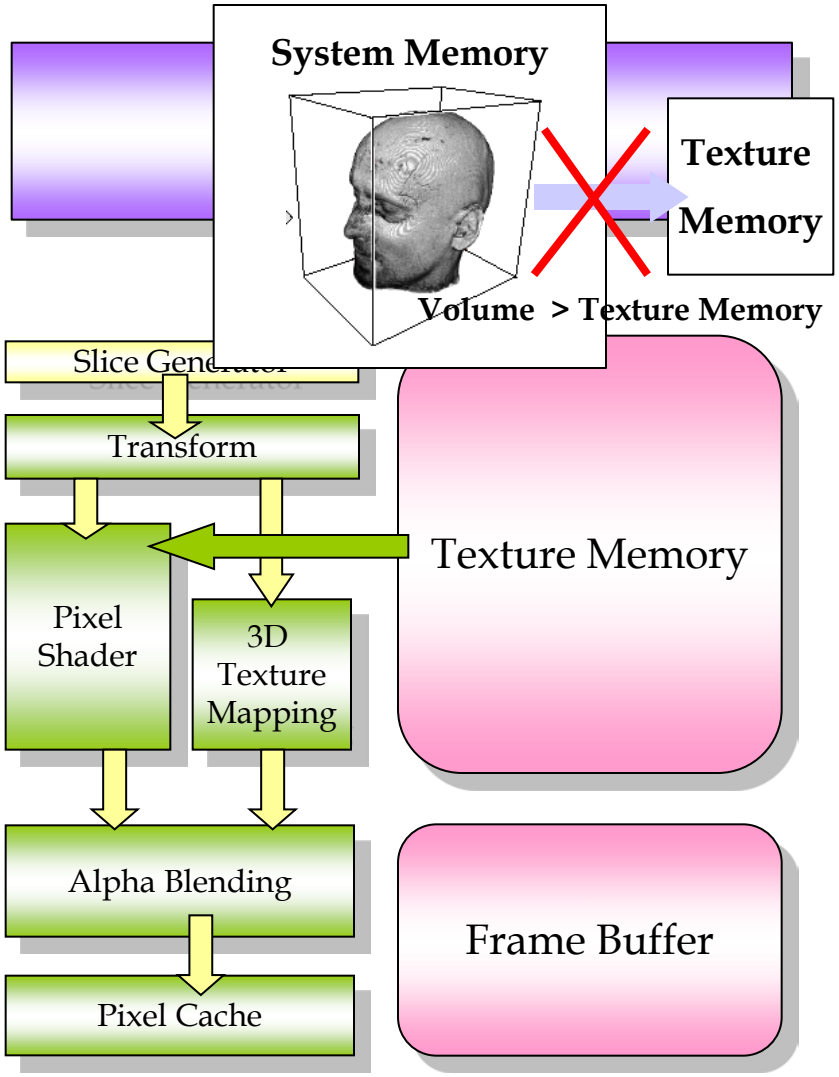
Volume Rendering with 3D Texture Mapping



Standard 3D Texture Mapping Architecture



Performance Limitation of 3D Texture Mapping

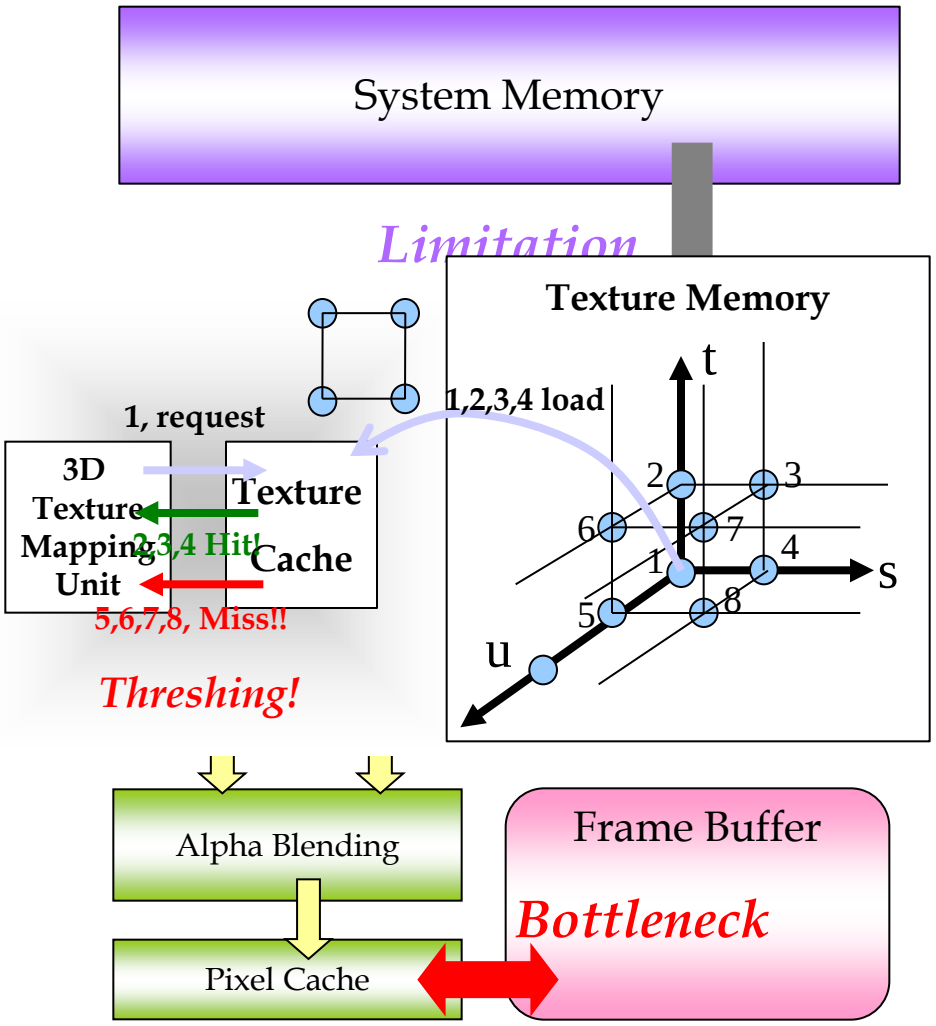


- **The size of volume data sets can be processed is still limited**
  - usually, 64-256MB graphic memory, middle sized(512^3) 8|16bit volume data, 256MB|512MB
- Bottleneck in graphic memory bus
  - lots of texture & pixel traffic (interpolation, α-blending)
  - Limited memory bandwidth: local 20GB/s, AGP8x:2GB/s
- Brute-forced approach
  - classical optimization technique cannot be utilized

  (empty-space skipping, early-ray termination)

=> Dividing a volume into a few sub-volumes

Performance Limitation of 3D Texture Mapping

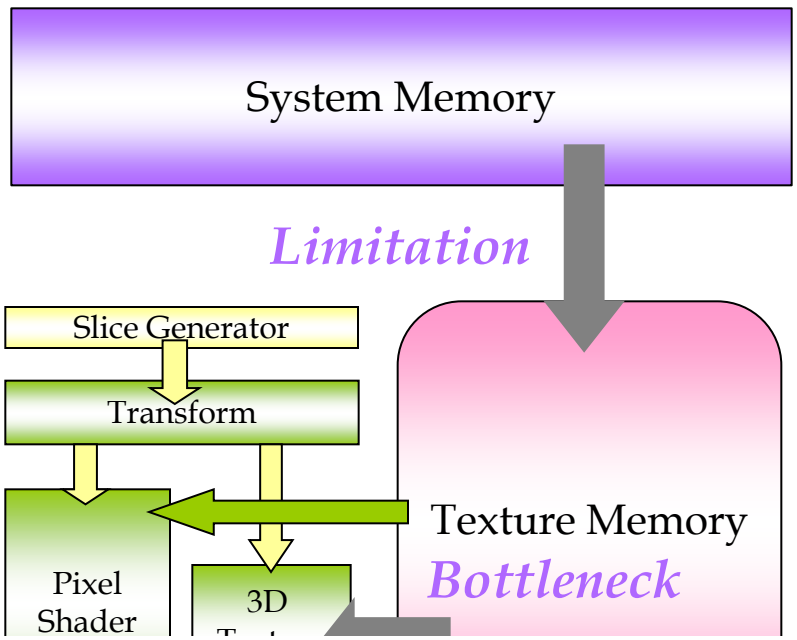


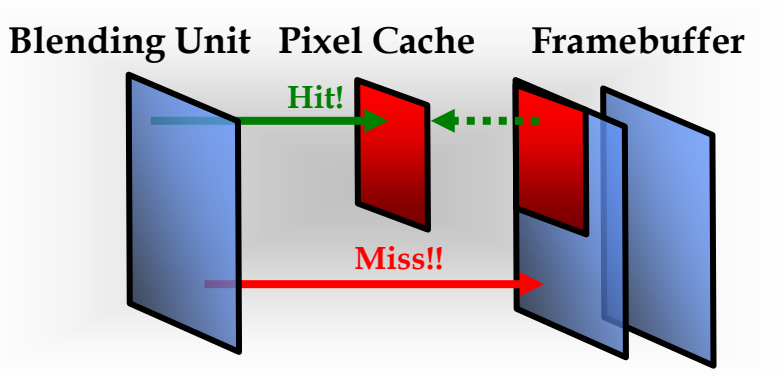
- The size of volume data sets can be processed is still limited
  - usually, 64-256MB graphic memory, middle sized(512^3) 8|16bit volume data, 256MB|512MB
- **Bottleneck in graphic memory bus**
  - lots of texture & pixel traffic (interpolation, α-blending)
  - Limited memory bandwidth: local 20GB/s, AGP8x:2GB/s
- Brute-forced approach
  - classical optimization technique cannot be utilized

  (empty-space skipping, early-ray termination)

=> Dividing a volume into a few sub-volumes

Performance Limitation of 3D Texture Mapping



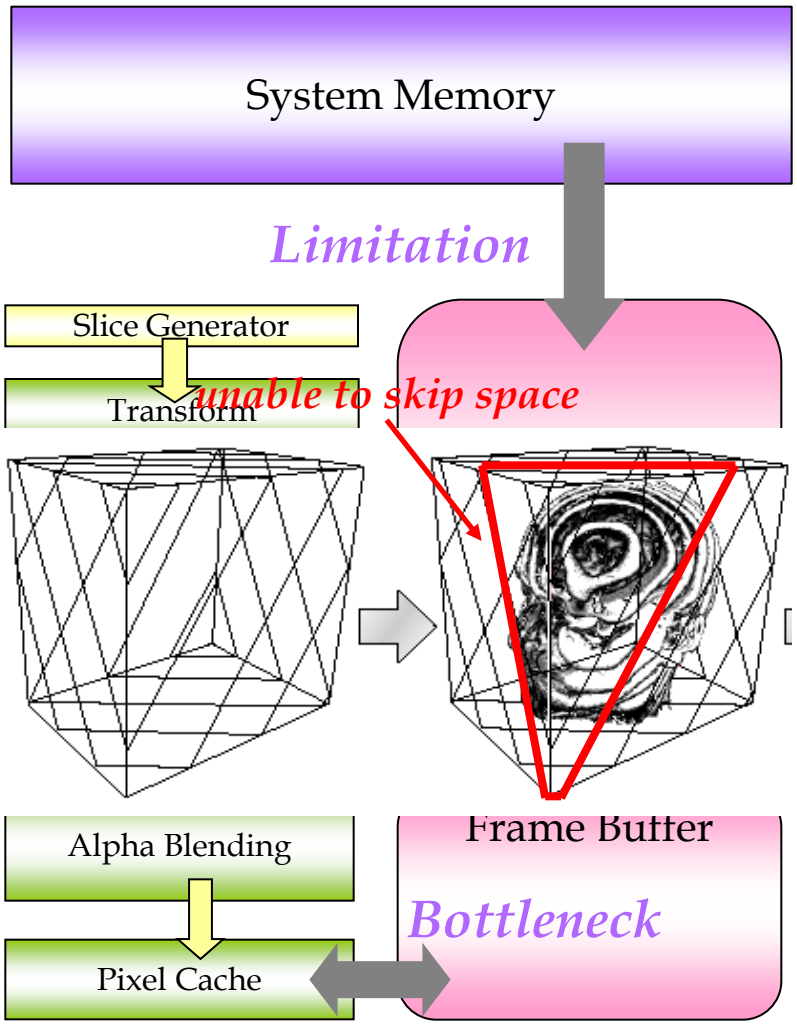


- The size of volume data sets can be processed is still limited
  - usually, 64-256MB graphic memory, middle sized(512^3) 8|16bit volume data, 256MB|512MB
- **Bottleneck in graphic memory bus**
  - lots of texture & pixel traffic (interpolation, α-blending)
  - Limited memory bandwidth: local 20GB/s, AGP8x:2GB/s
- Brute-forced approach
  - classical optimization technique cannot be utilized

  (empty-space skipping, early-ray termination)

=> Dividing a volume into a few sub-volumes

Performance Limitation of 3D Texture Mapping

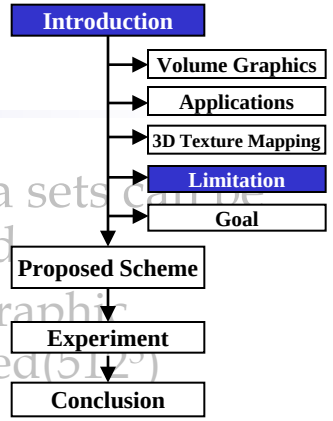


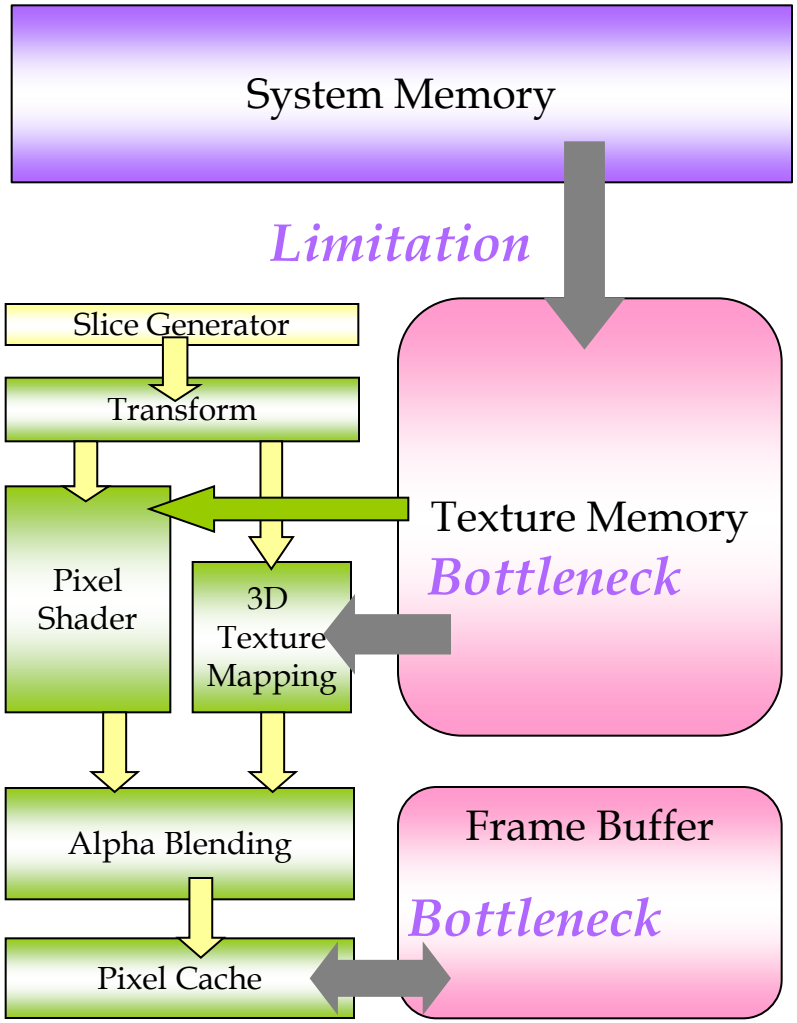
- The size of volume data sets can be processed is still limited
  - usually, 64-256MB graphic memory, middle sized(512^3) 8|16bit volume data, 256MB|512MB
- Bottleneck in graphic memory bus
  - lots of texture & pixel traffic (interpolation, α-blending)
  - Limited memory bandwidth: local 20GB/s, AGP8x:2GB/s
- **Brute-forced approach**
  - classical optimization technique cannot be utilized

  (empty-space skipping, early-ray termination)

=> Dividing a volume into a few sub-volumes

Performance Limitation of 3D Texture Mapping



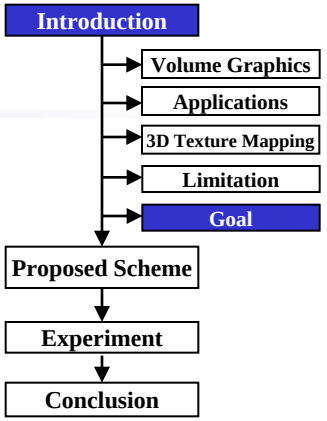


- The size of volume data sets can be processed is still limited
  - usually, 64-256MB graphic memory, middle sized(512^3) 8|16bit volume data, 256MB|512MB
- Bottleneck in graphic memory bus
  - lots of texture & pixel traffic (interpolation, α-blending)
  - Limited memory bandwidth: local 20GB/s, AGP8x:2GB/s
- Brute-forced approach
  - classical optimization technique cannot be utilized

  (empty-space skipping, early-ray termination)

=> Dividing a volume into a few sub-volumes

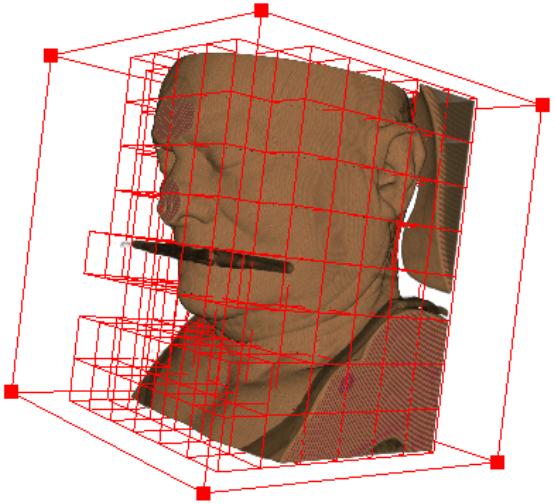
The Goal of This Research



- **Resolve the memory bandwidth problem in 3D TM**
  - Utilize sub-division of original volume
  - Increase texture and pixel cache efficiency

- **Achieve the additional performance gain**
  - Enabling to employ ESS (Empty Space Skipping)

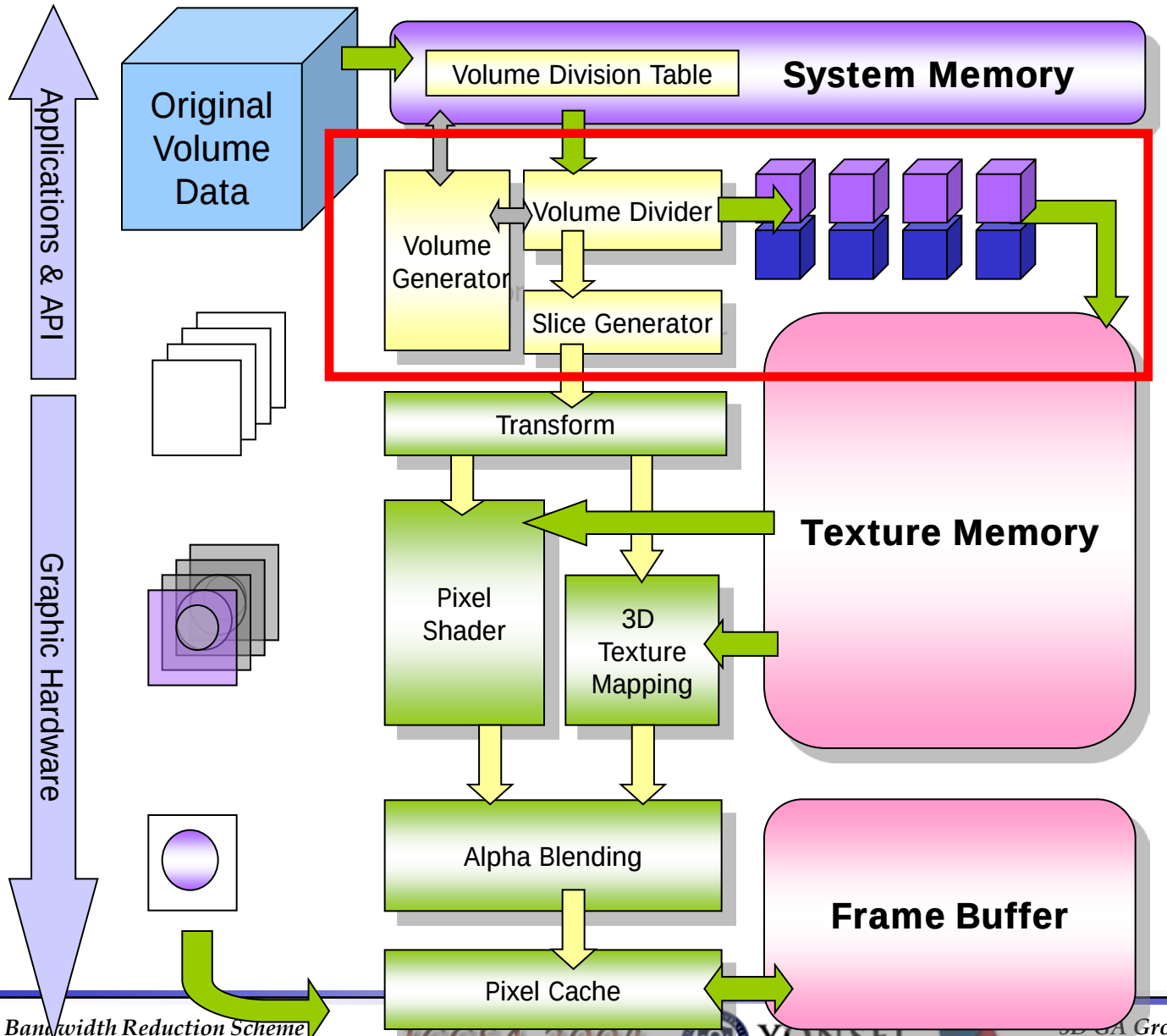
- **Render the volume which is not fit into texture memory**
  - Sub-volume ordered rendering



Proposed Scheme :

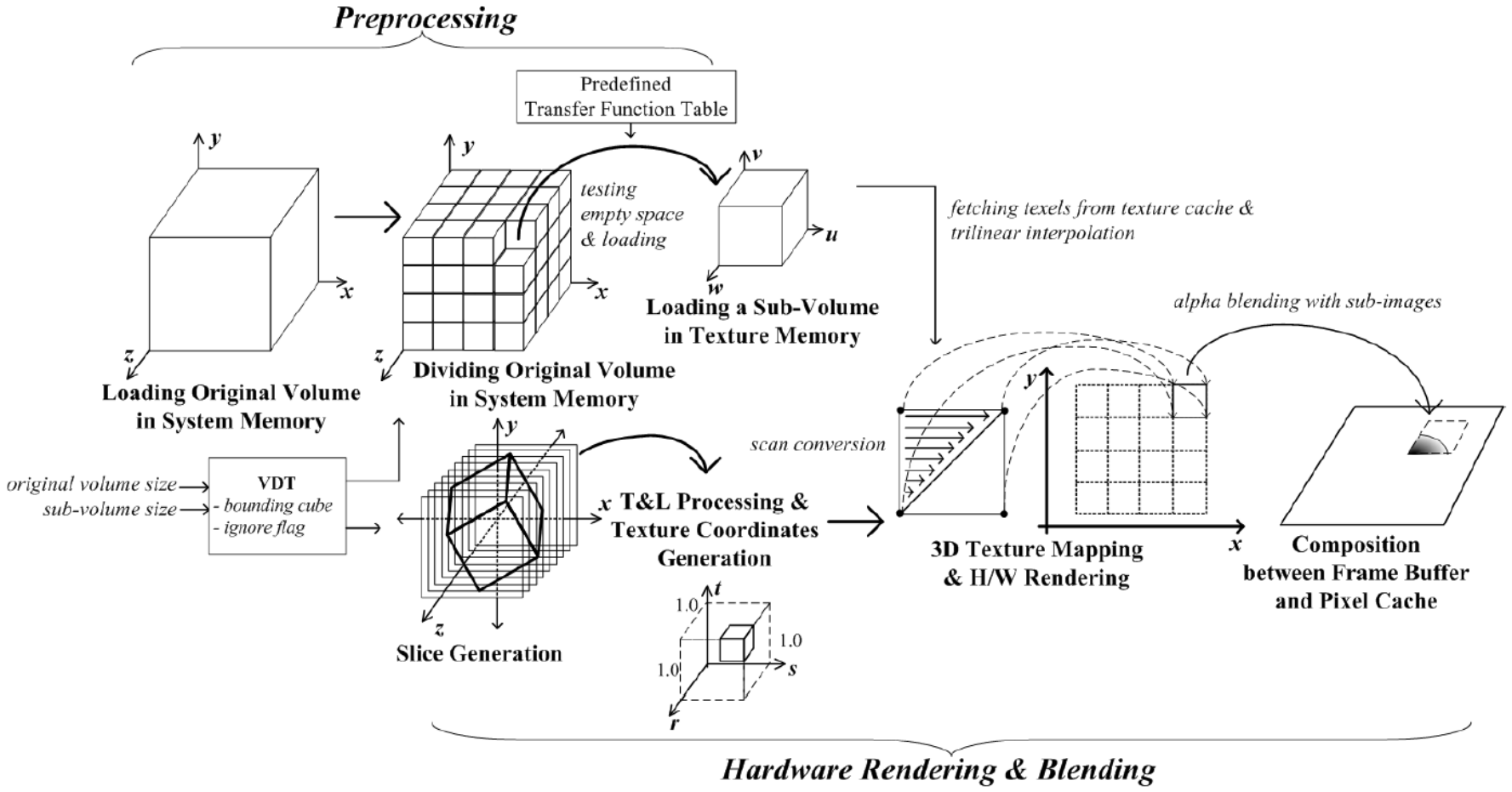
Bandwidth-Effective Sub-Volume Rendering

Bandwidth-Effective Sub-Volume Rendering



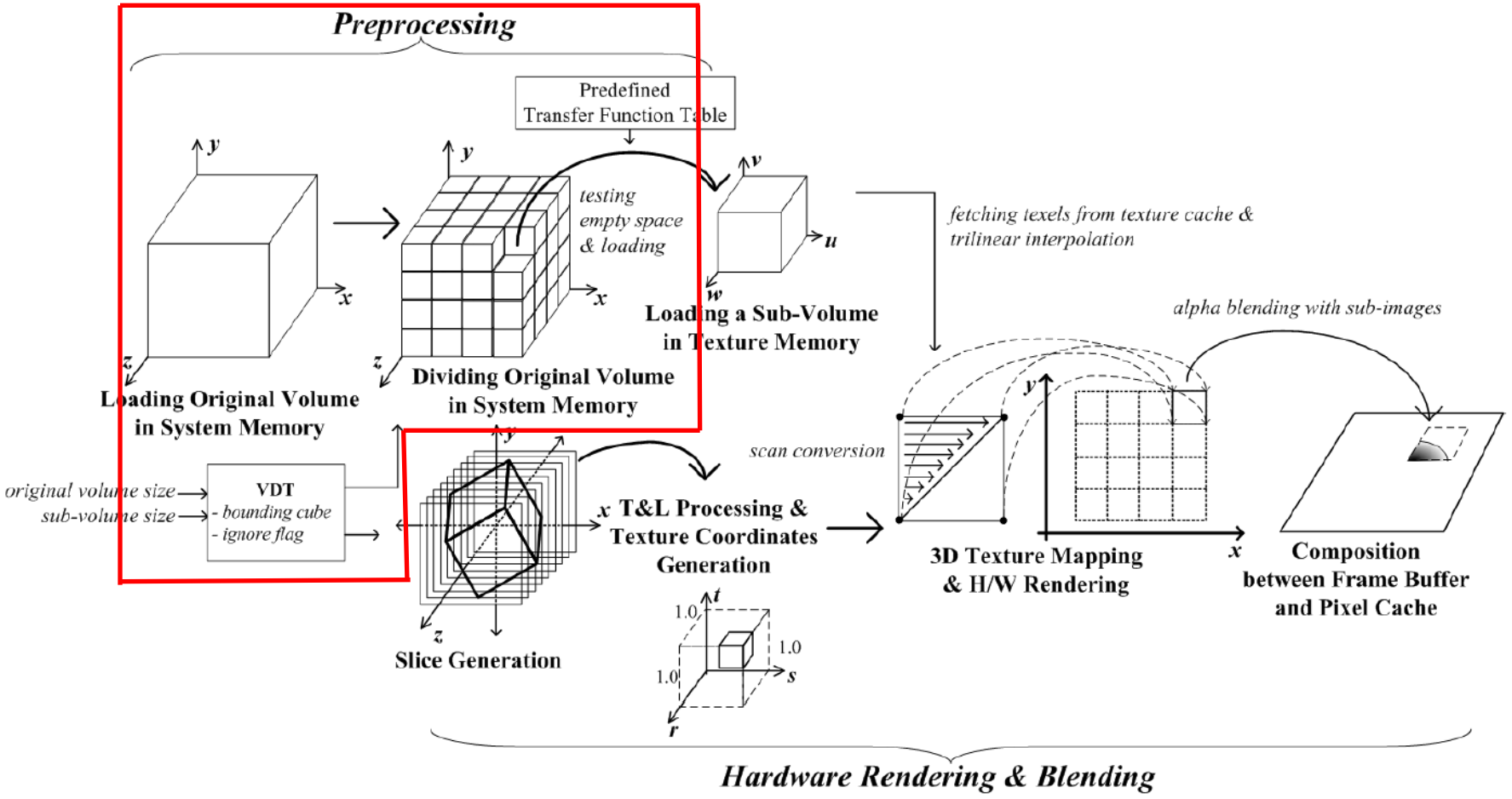
Bandwidth-Effective Sub-Volume Rendering

- Processing Flow
  - Pre-processing → Rendering & Mapping → Blending



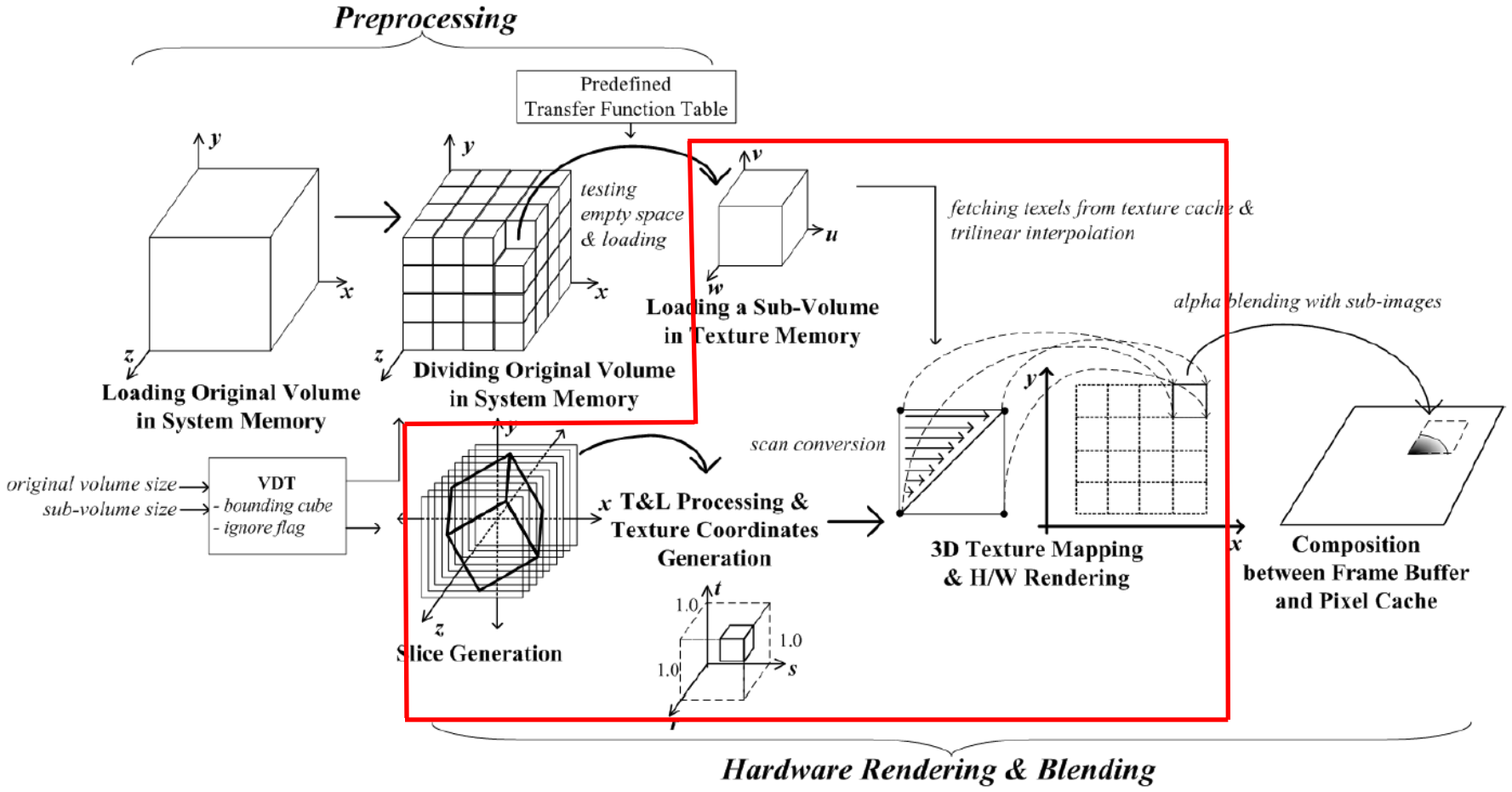
Bandwidth-Effective Sub-Volume Rendering

- Processing Flow
  - **Pre-processing** → Rendering & Mapping → Blending



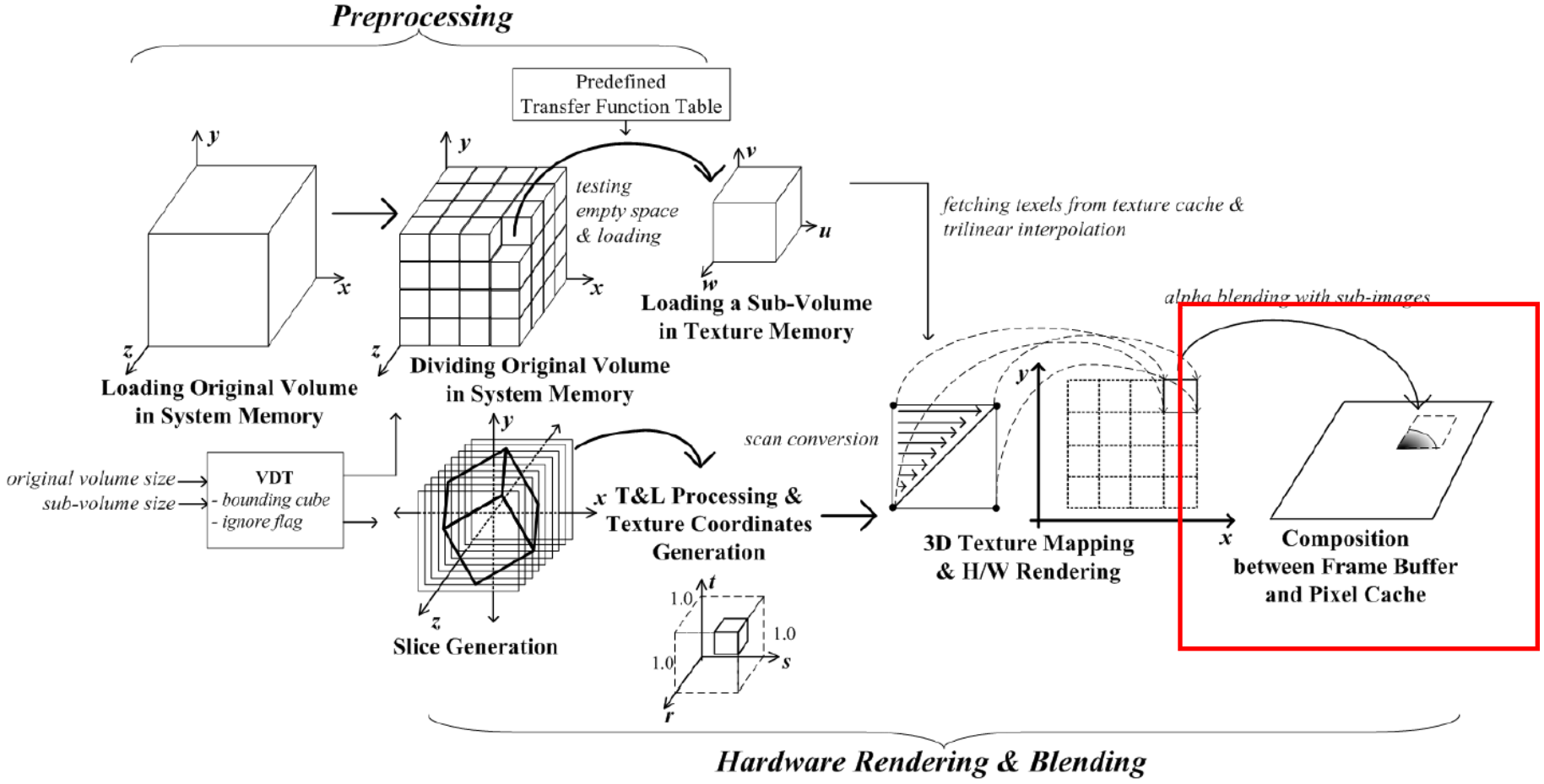
Bandwidth-Effective Sub-Volume Rendering

- Processing Flow
  - Pre-processing → **Rendering & Mapping** → Blending

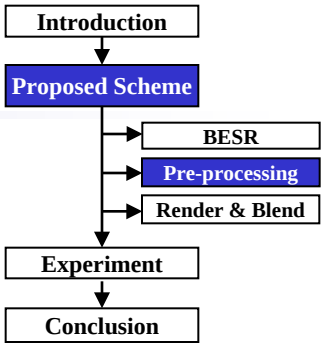


Bandwidth-Effective Sub-Volume Rendering

- Processing Flow
  - Pre-processing → Rendering & Mapping → **Blending**

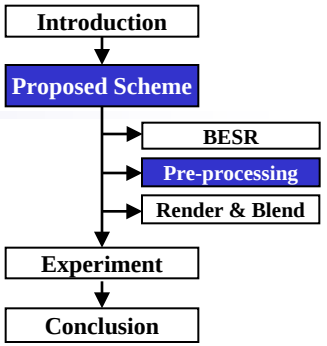


Bandwidth-Effective Sub-Volume Rendering



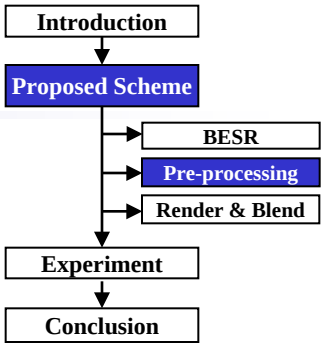
- **Pre-processing for sub-volume setup**
  1. Loading original volume data to system memory
  2. Choosing the size of the sub-volume
  3. Dividing original volume into sub-volumes (VDT creation)
  4. Creation of transfer function table for classification
  5. Checking if current sub-volume is empty or not (VDT update)

Bandwidth-Effective Sub-Volume Rendering



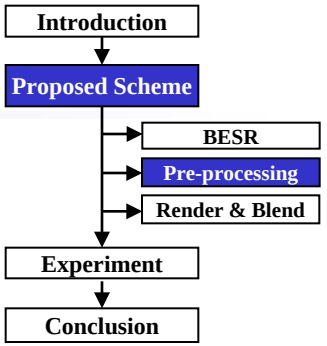
- **Pre-processing for sub-volume setup**
  1. Loading original volume data to system memory
  2. Choosing the size of the sub-volume
  3. Dividing original volume into sub-volumes (VDT creation)
  4. Creation of transfer function table for classification
  5. Checking if current sub-volume is empty or not (VDT update)

Bandwidth-Effective Sub-Volume Rendering



- **Pre-processing for sub-volume setup**
  1. Loading original volume data to system memory
  2. Choosing the size of the sub-volume
  3. Dividing original volume into sub-volumes (VDT creation)
  4. Creation of transfer function table for classification
  5. Checking if current sub-volume is empty or not (VDT update)

Bandwidth-Effective Sub-Volume Rendering



- **Step1 : Pre-processing for sub-volume setup**

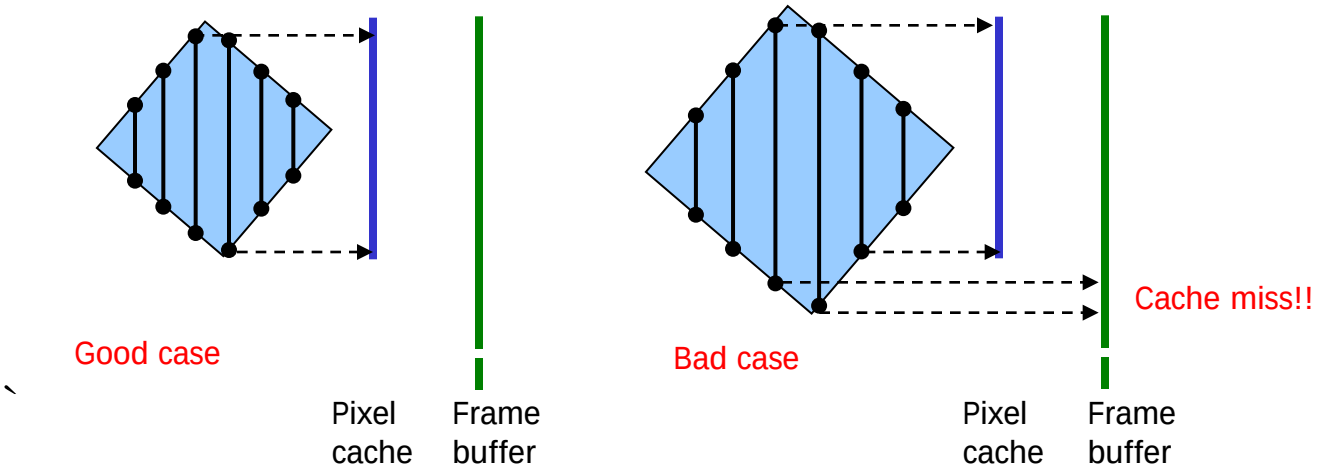
1. Loading original volume data to system memory
2. Choosing the size of the sub-volume
   - How to choose? - Slices must be fit into the pixel cache!!
3. Dividing original volume into sub-volumes (VDT creation)
4. Creation of transfer function table for classification
5. Checking if current sub-volume is empty or not (VDT update)

Bandwidth-Effective Sub-Volume Rendering

- **Step1 : Pre-processing for sub-volume setup**

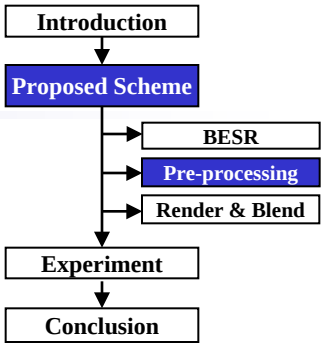
1. Loading original volume data to system memory
2. Choosing the size of the sub-volume

- How to choose? - Slices must be fit into the pixel cache!!



3. Dividing original volume into sub-volumes (VDT creation)
4. Creation of transfer function table for classification
5. Checking if current sub-volume is empty or not (VDT update)
6. If (the size of volume data < the size of texture memory)
   Loading full original volume data to GPU's texture memory

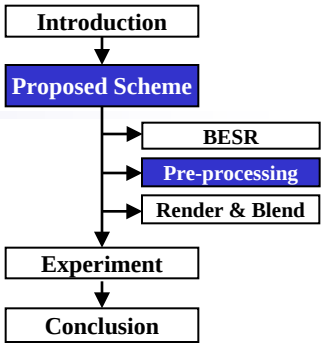
Bandwidth-Effective Sub-Volume Rendering



- **Step1 : Pre-processing for sub-volume setup**

1. Loading original volume data to system memory
2. Choosing the size of the sub-volume
3. Dividing original volume into sub-volumes (VDT creation)
4. Creation of transfer function table for classification
5. Checking if current sub-volume is empty or not (VDT update)
6. If (the size of volume data < the size of texture memory)
   Loading full original volume data to GPU's texture memory

Bandwidth-Effective Sub-Volume Rendering



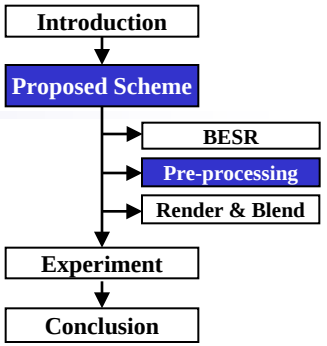
- **Step1 : Pre-processing for sub-volume setup**

1. Loading original volume data to system memory
2. Choosing the size of the sub-volume
3. Dividing original volume into sub-volumes (VDT creation)

   – Defined new data structure –***Volume Division Table*** (VDT)
     - MIN/MAX vertex & texture coordinates of each sub-volume
     - Empty space boolean tag for each sub-volume

4. Creation of transfer function table for classification
5. Checking if current sub-volume is empty or not (VDT update)
6. If (the size of volume data < the size of texture memory)
   Loading full original volume data to GPU's texture memory

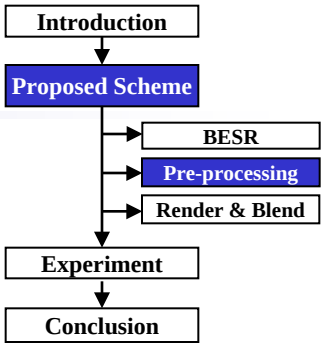
Bandwidth-Effective Sub-Volume Rendering



- **Step1 : Pre-processing for sub-volume setup**

1. Loading original volume data to system memory
2. Choosing the size of the sub-volume
3. Dividing original volume into sub-volumes (VDT creation)
4. Creation of transfer function table for classification
5. Checking if current sub-volume is empty or not (VDT update)
6. If (the size of volume data < the size of texture memory)
   Loading full original volume data to GPU's texture memory

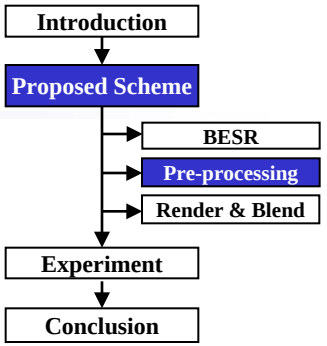
Bandwidth-Effective Sub-Volume Rendering



- **Step1 : Pre-processing for sub-volume setup**

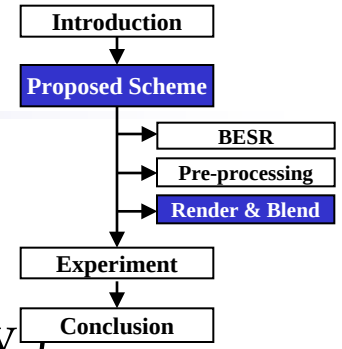
1. Loading original volume data to system memory
2. Choosing the size of the sub-volume
3. Dividing original volume into sub-volumes (VDT creation)
4. Creation of transfer function table for classification
5. Checking if current sub-volume is empty or not (VDT update)
6. If (the size of volume data < the size of texture memory)
   Loading full original volume data to GPU's texture memory

Bandwidth-Effective Sub-Volume Rendering



- **Step1 : Pre-processing for sub-volume setup**
  1. Loading original volume data to system memory
  2. Choosing the size of the sub-volume
  3. Dividing original volume into sub-volumes (VDT creation)
  4. Creation of transfer function table for classification
  5. Checking if current sub-volume is empty or not (VDT update)
  6. If (the size of volume data < the size of texture memory)
     Loading full original volume data to GPU's texture memory

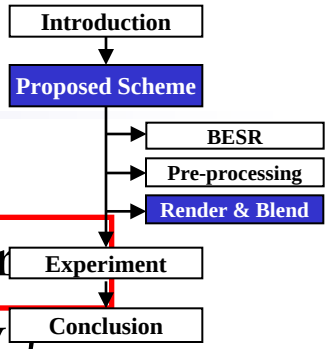
Bandwidth-Effective Sub-Volume Rendering



- **Step2 : Sub-Volume Rendering**
  1. Setup for sub-volume rendering order
  2. If (the size of volume data > the size of texture memory)
     Loading current sub-volume data to GPU's texture memory
  3. Compute the vertex and texture coordinates of slices
  4. 3D texture mapping of non-empty sub-volume only

- **Step3 : Blending**
  - Blend each texture mapped slices

Bandwidth-Effective Sub-Volume Rendering



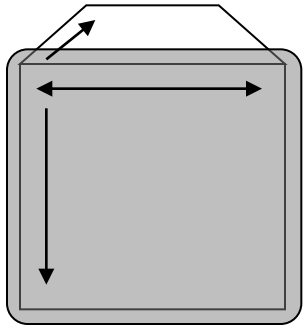
- **Step2 : Sub-Volume Rendering**
  1. Setup for sub-volume rendering order – Visibility Order
  2. If (the size of volume data > the size of texture memory)
     Loading current sub-volume data to GPU's texture memory
  3. Compute the vertex and texture coordinates of slices
  4. 3D texture mapping of non-empty sub-volume only

- **Step3 : Blending**
  - Blend each texture mapped slices

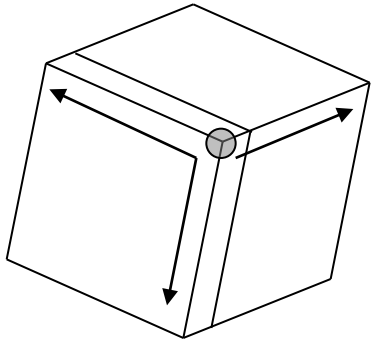
Bandwidth-Effective Sub-Volume Rendering

- **Step2 : Sub-Volume Rendering**

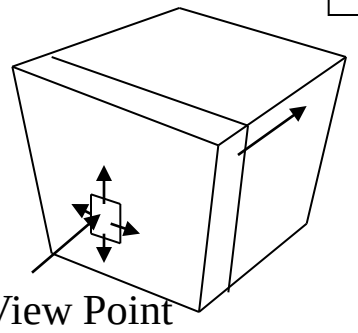
1. Setup for sub-volume rendering order – Visibility Order



Case1: Face on Plane



Case2: Face on Point or Edge



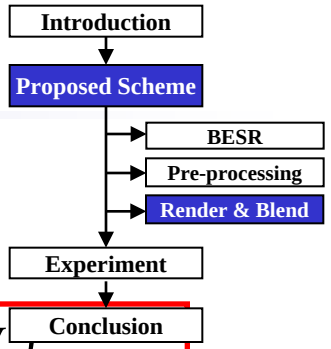
Case3: Perspective View

Ref) The RaceII Engine, SIGRAPH/Eurographics Graphics Hardware 2000

2. If (the size of volume data > the size of texture memory)
   Loading current sub-volume data to GPU's texture memory
3. Compute the vertex and texture coordinates of slices
4. 3D texture mapping of non-empty sub-volume only

- **Step3 : Blending**
  - Blend each texture mapped slices

Bandwidth-Effective Sub-Volume Rendering

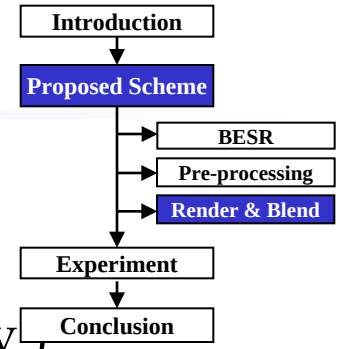


- **Step2 : Sub-Volume Rendering**

  1. Setup for sub-volume rendering order
  2. If (the size of volume data > the size of texture memory)
     Loading current sub-volume data to GPU's texture memory
  3. Compute the vertex and texture coordinates of slices
  4. 3D texture mapping of non-empty sub-volume only

- **Step3 : Blending**
  - Blend each texture mapped slices

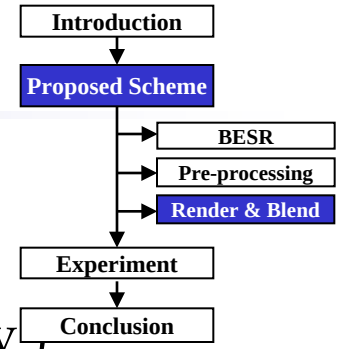
Bandwidth-Effective Sub-Volume Rendering



- **Step2 : Sub-Volume Rendering**
  1. Setup for sub-volume rendering order
  2. If (the size of volume data > the size of texture memory)
     Loading current sub-volume data to GPU's texture memory
  3. Compute the vertex and texture coordinates of slices
  4. 3D texture mapping of non-empty sub-volume only

- **Step3 : Blending**
  - Blend each texture mapped slices

Bandwidth-Effective Sub-Volume Rendering



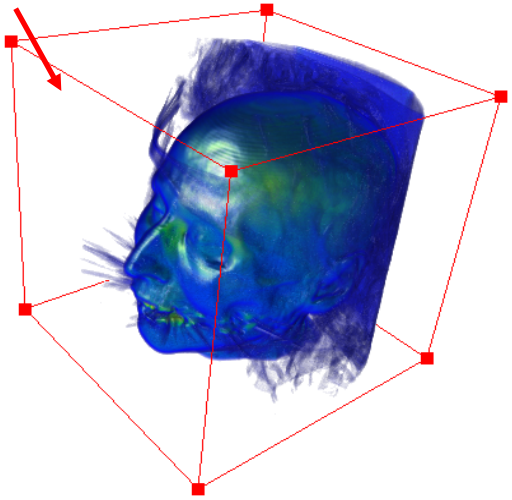
- **Step2 : Sub-Volume Rendering**
  1. Setup for sub-volume rendering order
  2. If (the size of volume data > the size of texture memory)
     Loading current sub-volume data to GPU's texture memory
  3. Compute the vertex and texture coordinates of slices
  4. 3D texture mapping of non-empty sub-volume only

- **Step3 : Blending**
  - Blend each texture mapped slices

Bandwidth-Effective Sub-Volume Rendering

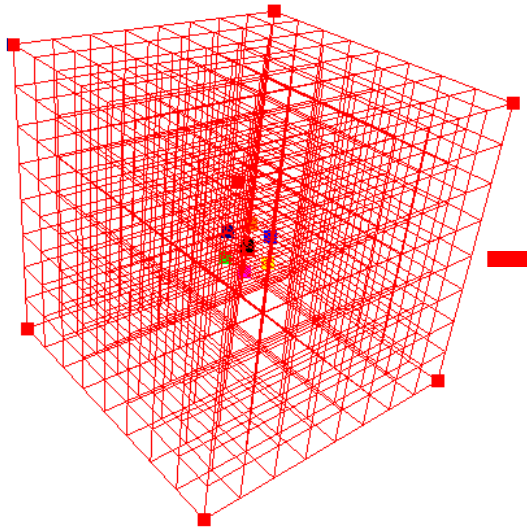
- **Step2 : Sub-Volume Rendering**
  1. Setup for sub-volume rendering order
  2. If (the size of volume data > the size of texture memory)
     Loading current sub-volume data to GPU's texture memory
  3. Compute the vertex and texture coordinates of slices
  4. 3D texture mapping of non-empty sub-volume only

unable to skip empty space

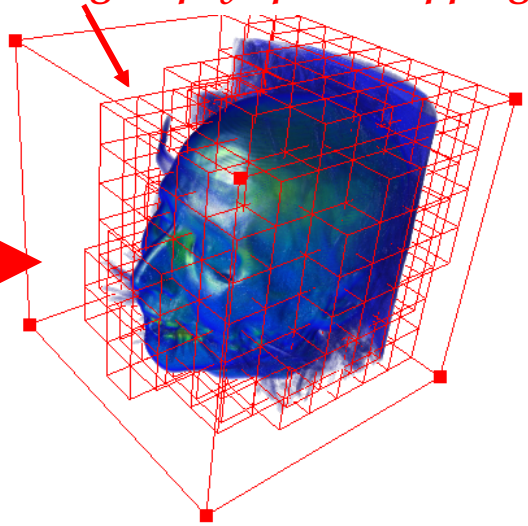


standard method

Sub-division

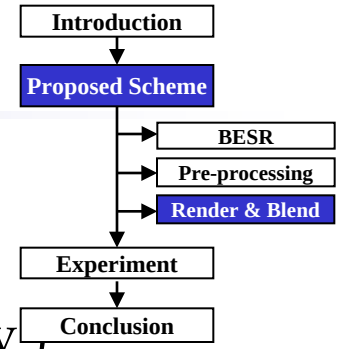


enabling empty space skipping



proposed method

Bandwidth-Effective Sub-Volume Rendering

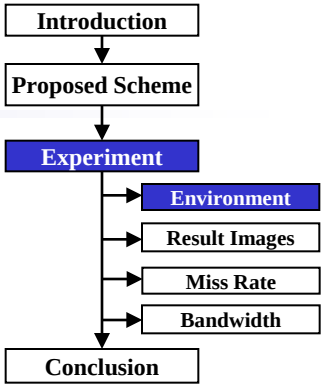


- **Step2 : Sub-Volume Rendering**
  1. Setup for sub-volume rendering order
  2. If (the size of volume data > the size of texture memory)
     Loading current sub-volume data to GPU's texture memory
  3. Compute the vertex and texture coordinates of slices
  4. 3D texture mapping of non-empty sub-volume only

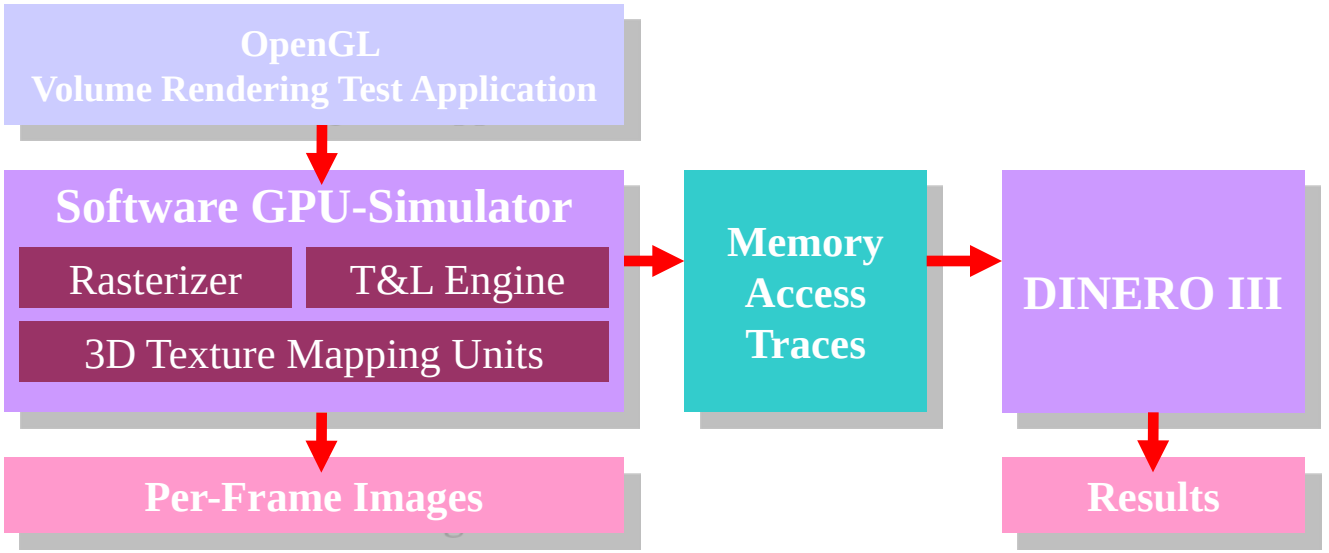
- **Step3 : Blending**
  - Blend each texture mapped slices

Experimental Results

Simulation

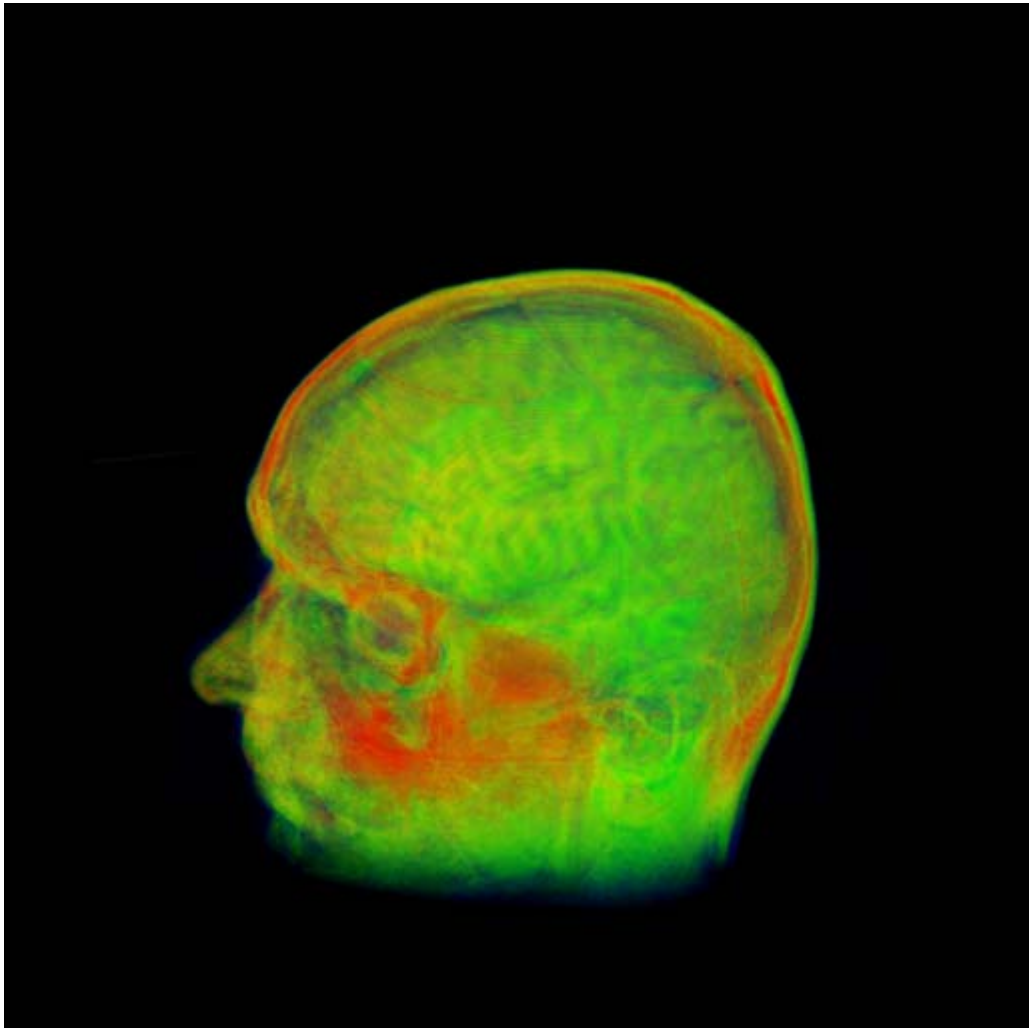


- Environments
  - Trace-driven C++ V/R simulator (T&L engine, Raterization & 3D TM unit, blending unit, framebuffer, texture memory)
  - DINERO III for cache simulation
  - Cache configuration : direct mapped, 32B-block, 16KB~128KB-size
  - Benchmarks : HeadMR, Foot, Skull (512^3-resampled)
  - # of slices : 100, # of rendered frames : 10
  - Empty space skipping is used



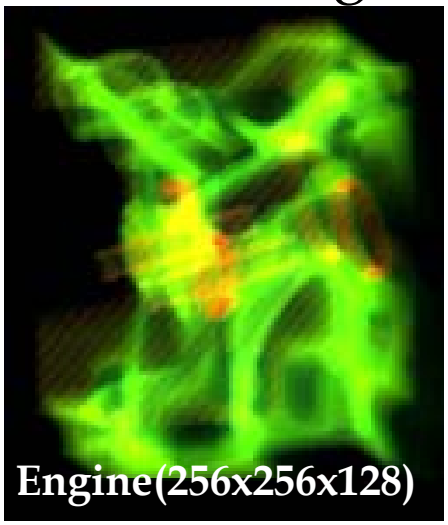
Result Images

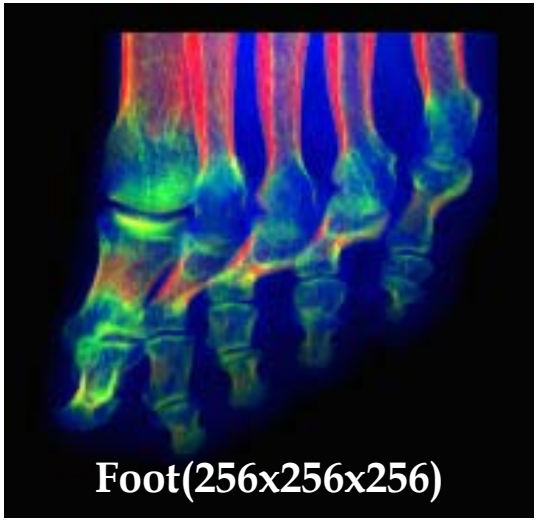
- Sub-volume image composition procedure

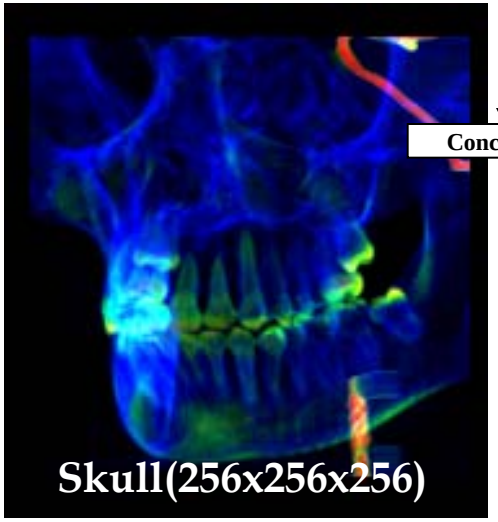


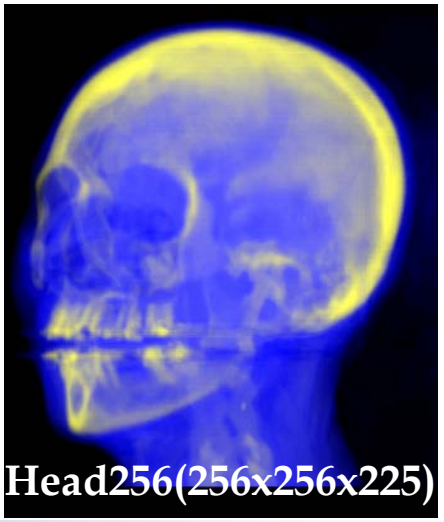
Result Images

- Other Images



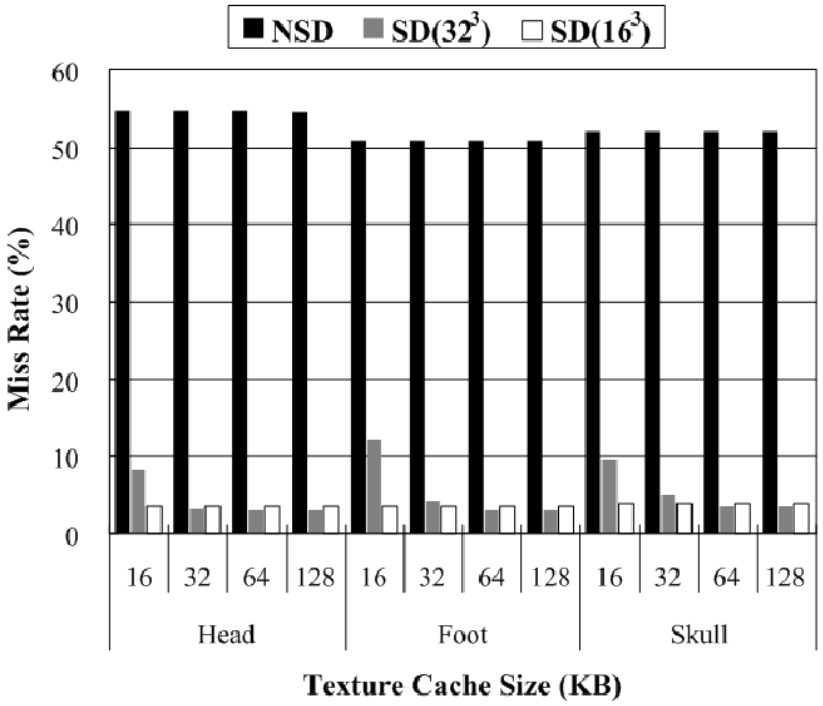


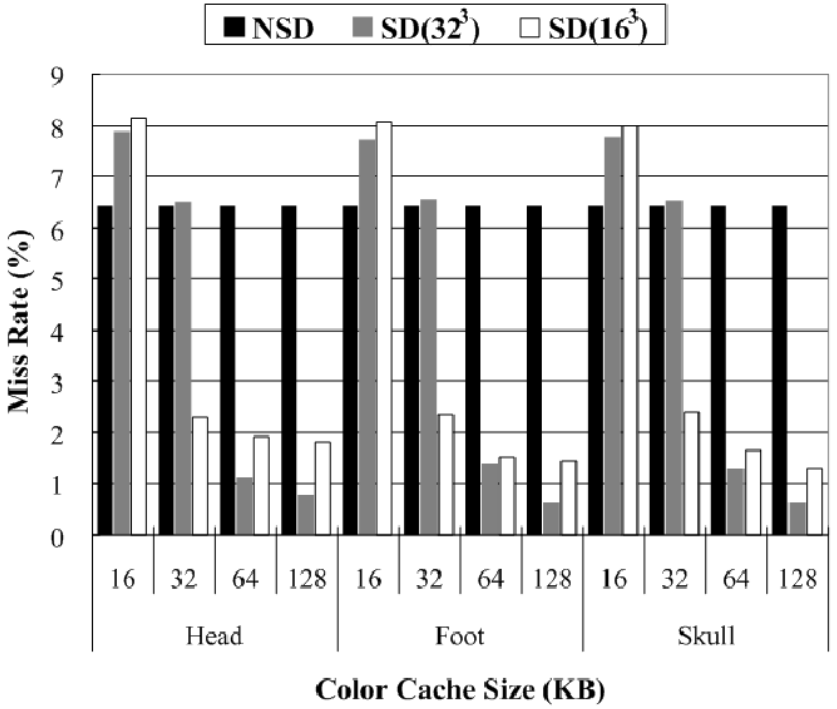




Simulation Results

- Comparison of cache (texture,pixel) miss rate





Simulation Results

- Comparison of required memory bandwidth

Dataset	Rendering Method	Sub-Volumes	Empty-Space Skipped Sub-Volumes	Vertices	Texture Cache Size(KB)	Read Data from Texture Memory by Cache Miss(Bytes)	Pixel Cache Size(KB)	Read/Write Data from/to Framebuffer by Cache Miss(Bytes)	**Total Bandwidth needed for 30f/s(GB/s)**
HeadMR (Fig.4-a)	NSD	-	-	4,000	16	15,025,144,064	16	1,926,144,000	**47.37**
					32	15,024,721,472	32		**47.37**
					64	15,023,112,832	64		**47.36**
					128	15,016,700,416	128		**47.34**
	SD(32)	4,096	2,042	572,120	16	1,764,230,976	16	2,927,880,896	**13.16**
					32	680,485,568	32	2,418,177,920	**8.71**
					64	630,633,824	64	417,452,352	**2.98**
					128	630,477,568	128	289,171,136	**2.62**
	SD(16)	32,768	18,894	2,219,840	16	642,788,320	16	2,572,624,960	**9.19**
					32		32	726,369,856	**4.03**
					64		64	610,183,616	**3.71**
					128		128	575,279,040	**3.61**
Foot (Fig.4-b)	NSD	-	-	4,000	16	13,478,043,488	16	1,926,144,000	**43.04**
					32	13,477,901,216	32		**43.04**
					64	13,477,758,976	64		**43.04**
					128	13,477,473,600	128		**43.04**
	SD(32)	4,096	1,859	626,360	16	2,772,522,176	16	3,456,836,352	**17.47**
					32	969,392,416	32	2,935,609,984	**10.98**
					64	684,198,304	64	616,759,424	**3.7**
					128	684,198,304	128	215,540,160	**2.58**
	SD(16)	32,768	17,735	2,405,280	16	880,501,952	16	4,000,987,136	**13.86**
					32		32	1,155,852,416	**5.91**
					64		64	748,096,256	**4.78**
					128		128	710,950,016	**4.67**
Skull (Fig.4-c)	NSD	-	-	4,000	16	14,249,069,696	16	1,926,144,000	**45.2**
					32	14,249,069,696	32		**45.2**
					64	14,249,069,696	64		**45.2**
					128	14,249,069,696	128		**45.2**
	SD(32)	4,096	2,935	325,080	16	2,281,828,608	16	1,745,188,160	**11.29**
					32	578,412,608	32	1,465,006,080	**5.74**
					64	426,176,352	64	291,141,632	**2.04**
					128	426,176,352	128	143,732,992	**1.63**
	SD(16)	32,768	26,585	989,280	16	321,912,992	16	1,193,654,720	**4.33**
					32		32	358,563,008	**2.0**
					64		64	247,057,408	**1.68**
					128		128	193,965,568	**1.54**

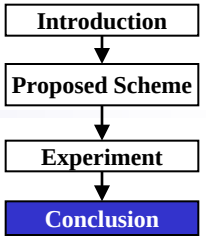
NSD - Non-SubVolumed

SD(32) - 32^3-SubVolumed

SD(16) - 16^3-SubVolumed

Conclusion & Schedule

Conclusion & Future Work



- Bandwidth-Effective Sub-Volume Rendering
  - Maximize the temporal & spacial locality of memory access
  - Enable the empty space skipping

  ⇒2~30x bandwidth reduction
  - Could render volume data which is not fit into the texture memory
- Future Work
  - Octree-based sub-volume rendering on GPU