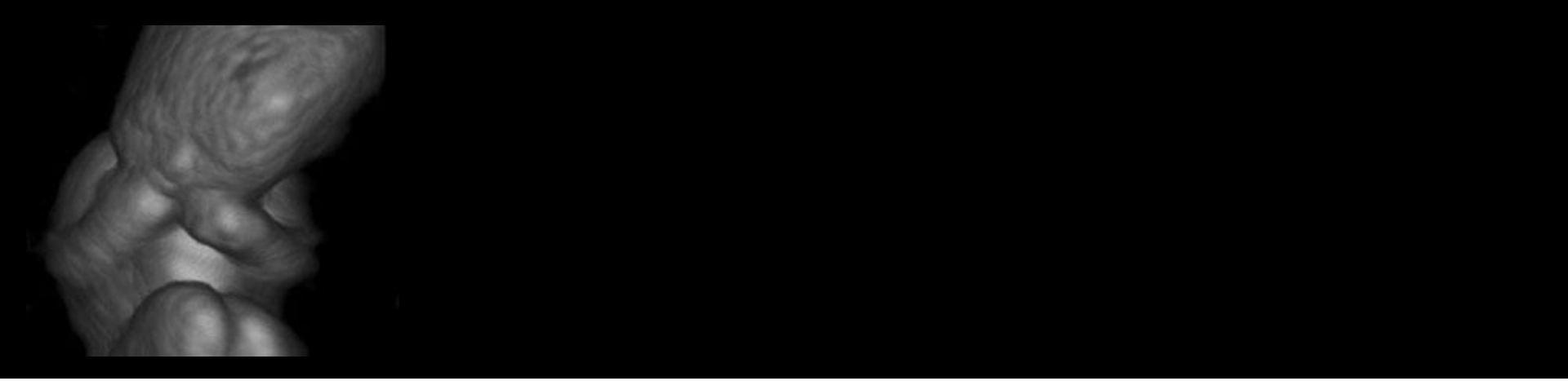
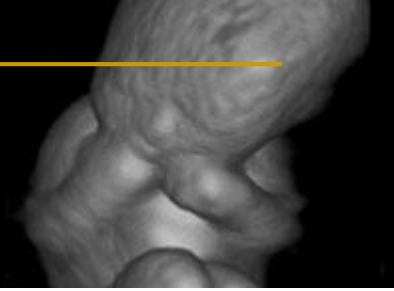


Medical Visualization
- for Ultrasonic Data -

Byeong-Seok Shin

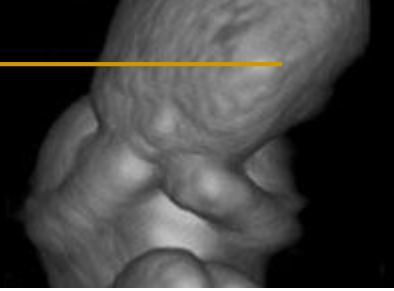
Media Lab. INHA Univ. KOREA

Contents



- Objective
- Issues of Ultrasound Data Visualization
- Image order approach vs. Object order approach
- Main Algorithm
  - Overall pipeline of proposed rendering method
  - Projection of Point Primitives
  - Coordinates conversion & 3D filtering using Shaders
- Experimental Results
- Conclusion

Objective



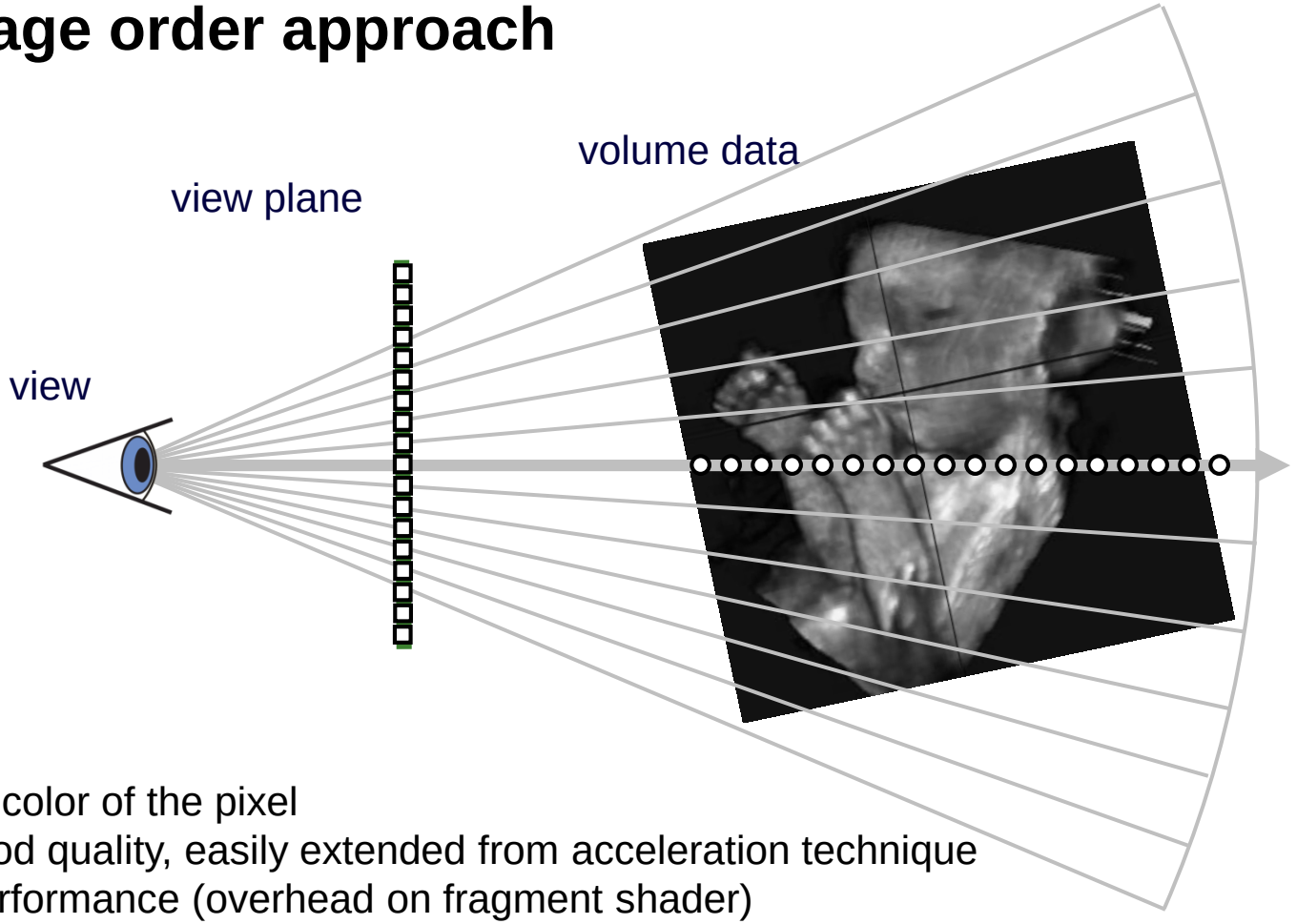
- Provide interactive rendering method of ultrasound data
  - using projection of point primitives on vertex shader
    - Compared with using pixel shader
  - including both coordinates conversion and 3Dfiltering
    - Conical coordinates to Cartesian coordinates
    - Reducing speckle noise with digital filtering operators

Issues of Ultrasound Data Visualization

- Coordinates conversion
  - Ultrasound use conical coordinates rather than Cartesian coordinates
  - We have to convert those coordinates during rendering stage
- Filtering for removing noise and uninterested region
  - We have to reduce uninterested region such as noise and speckles
  - Lots of filtering methods have been proposed
  - Usually time-consuming task
  - We provide real-time 3D filtering using vertex shader

Image vs. Object order approach

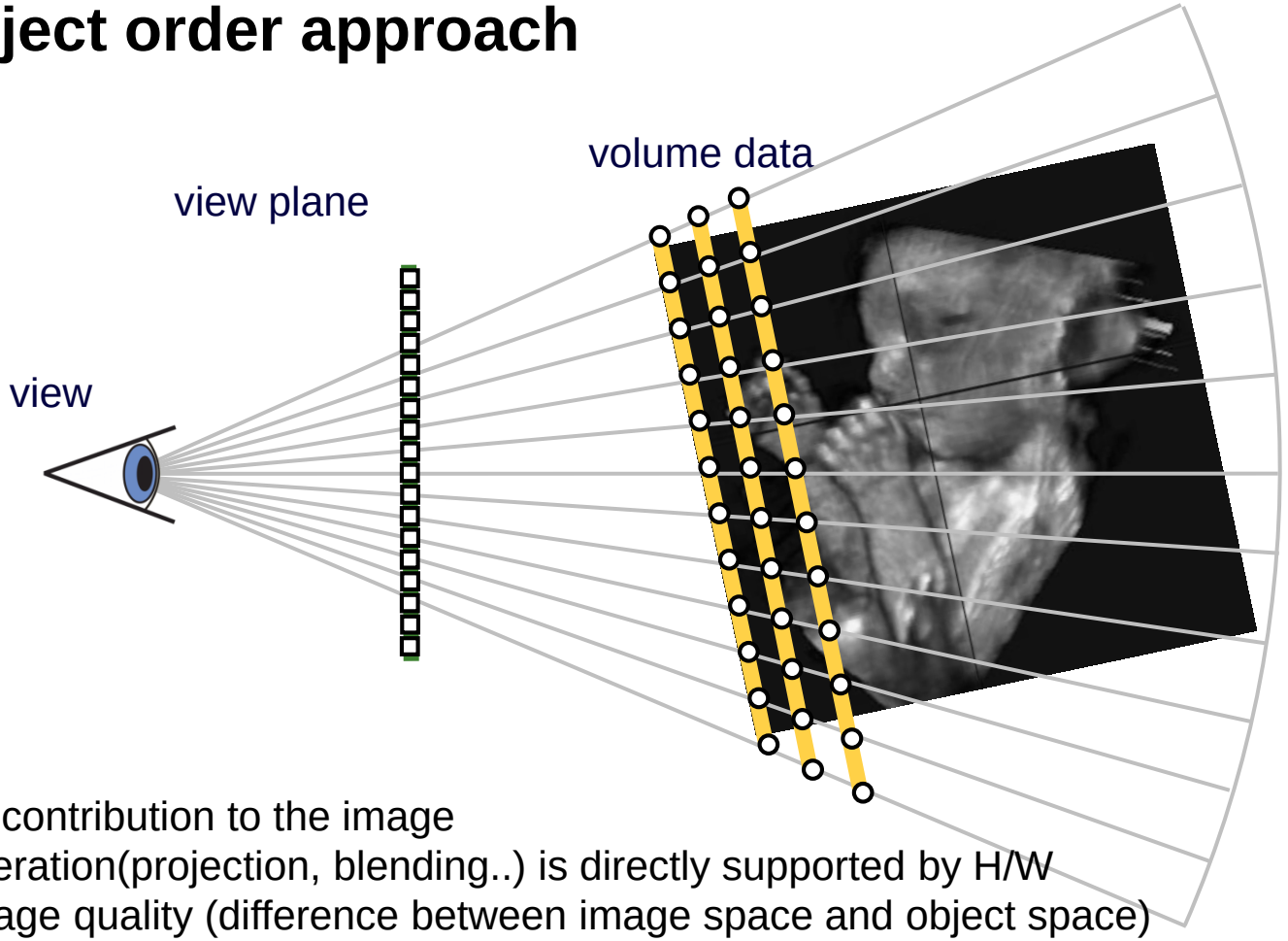
- **Image order approach**



calculate color of the pixel
Pros. Good quality, easily extended from acceleration technique
Cons. Performance (overhead on fragment shader)

Image vs. Object order approach

- **Object order approach**



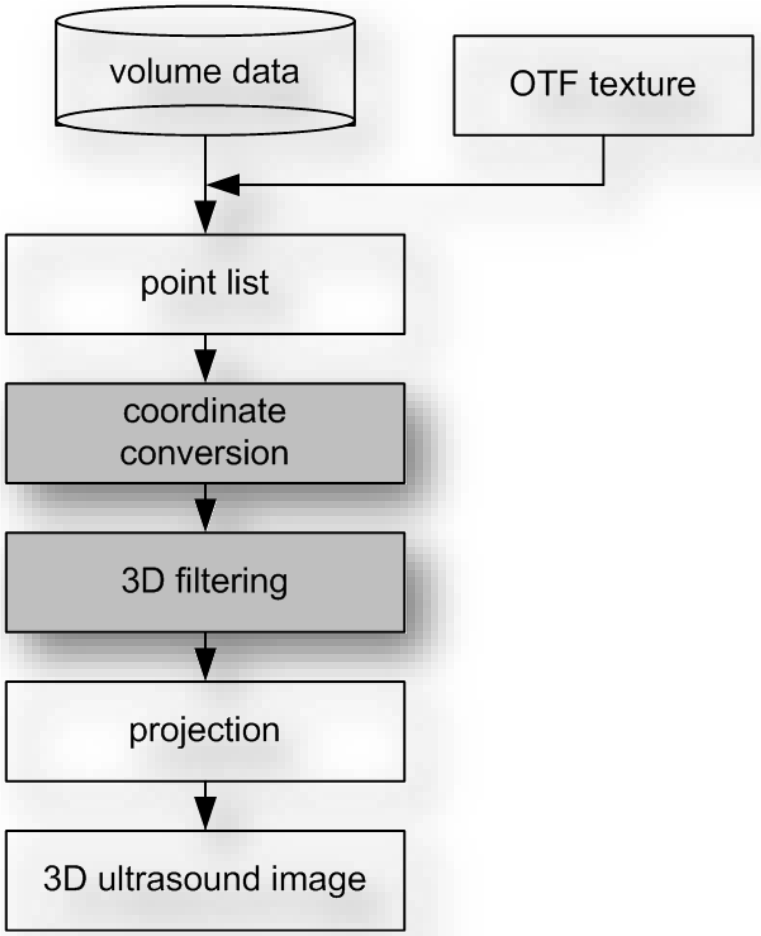
calculate contribution to the image
Pros. Operation(projection, blending..) is directly supported by H/W
Cons. Image quality (difference between image space and object space)

Main Algorithm

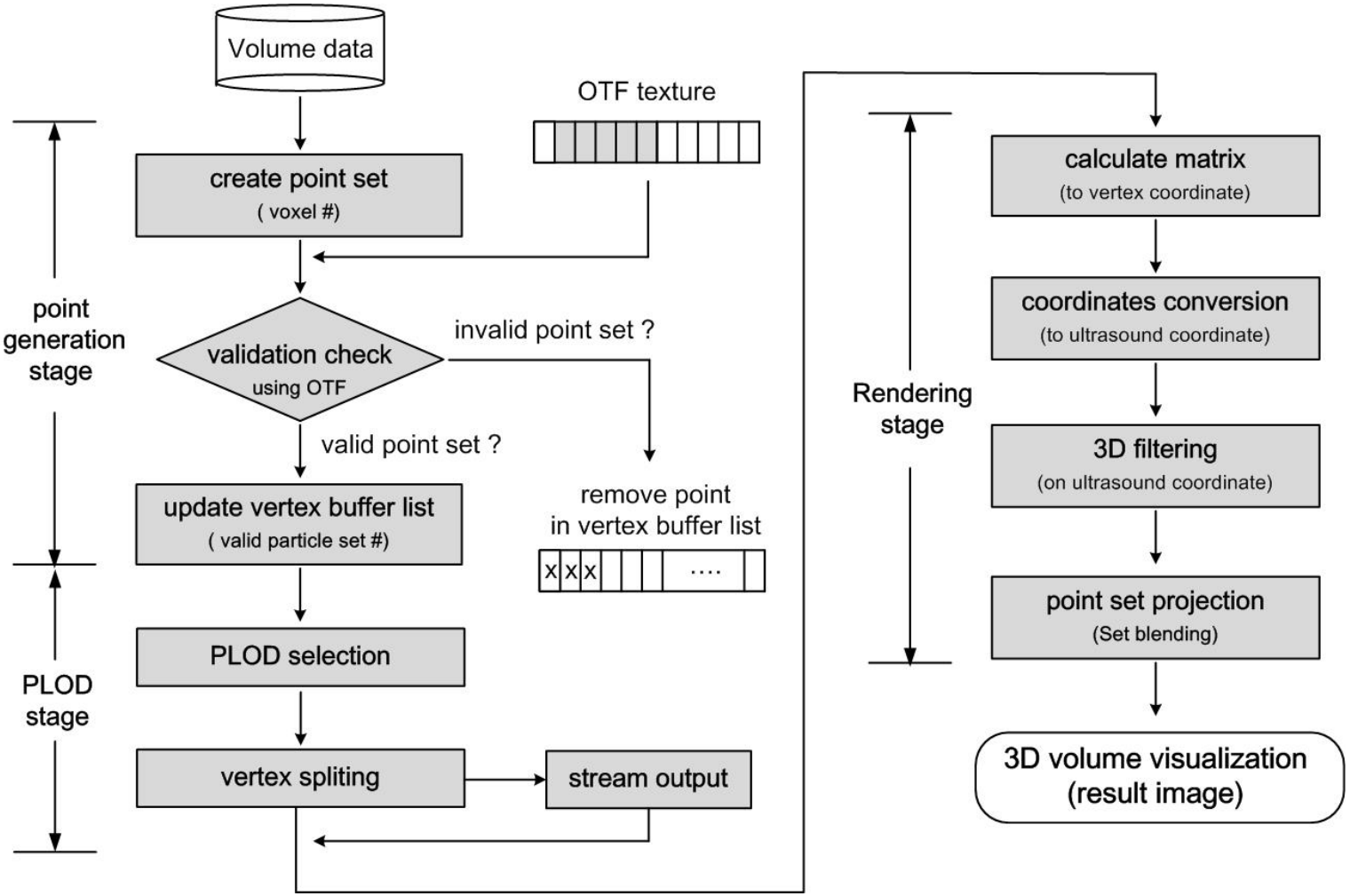
- Ultrasound data visualization
  - Object-order approach
  - 1 thread per vertex

Steps:

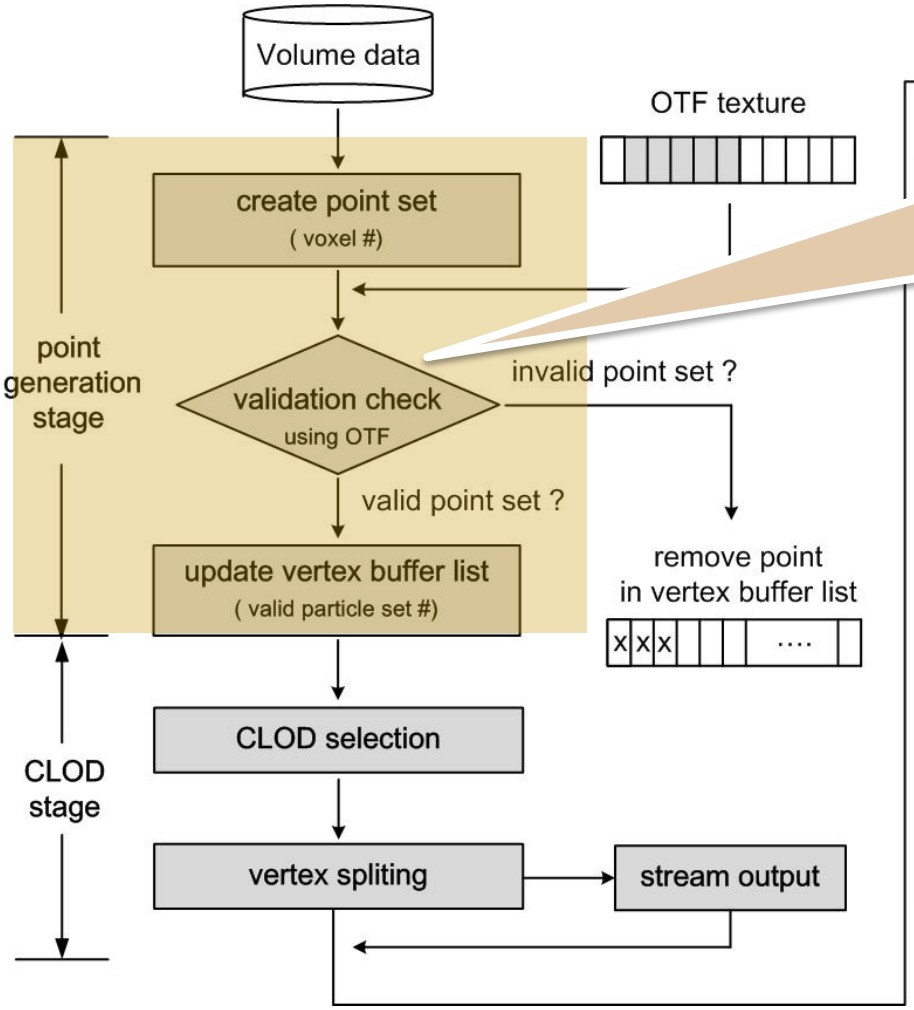
1. Create point list
2. View transform with ultrasound coordinates conversion
3. 3D filtering on point list
4. View plane projection



Overall pipeline

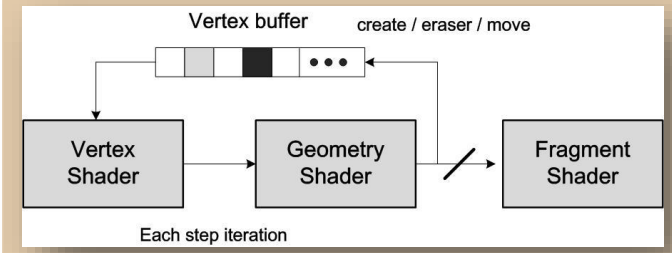


Projection of Point Primitive



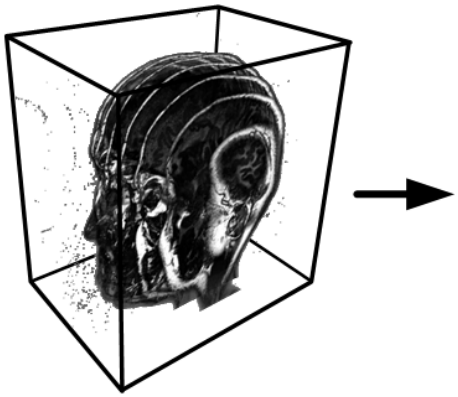
Previous programmable GPU
- performed sequentially
- from vertex to fragment shader

Recent GPU architecture
- possible to perform until middle phase, geometry shader



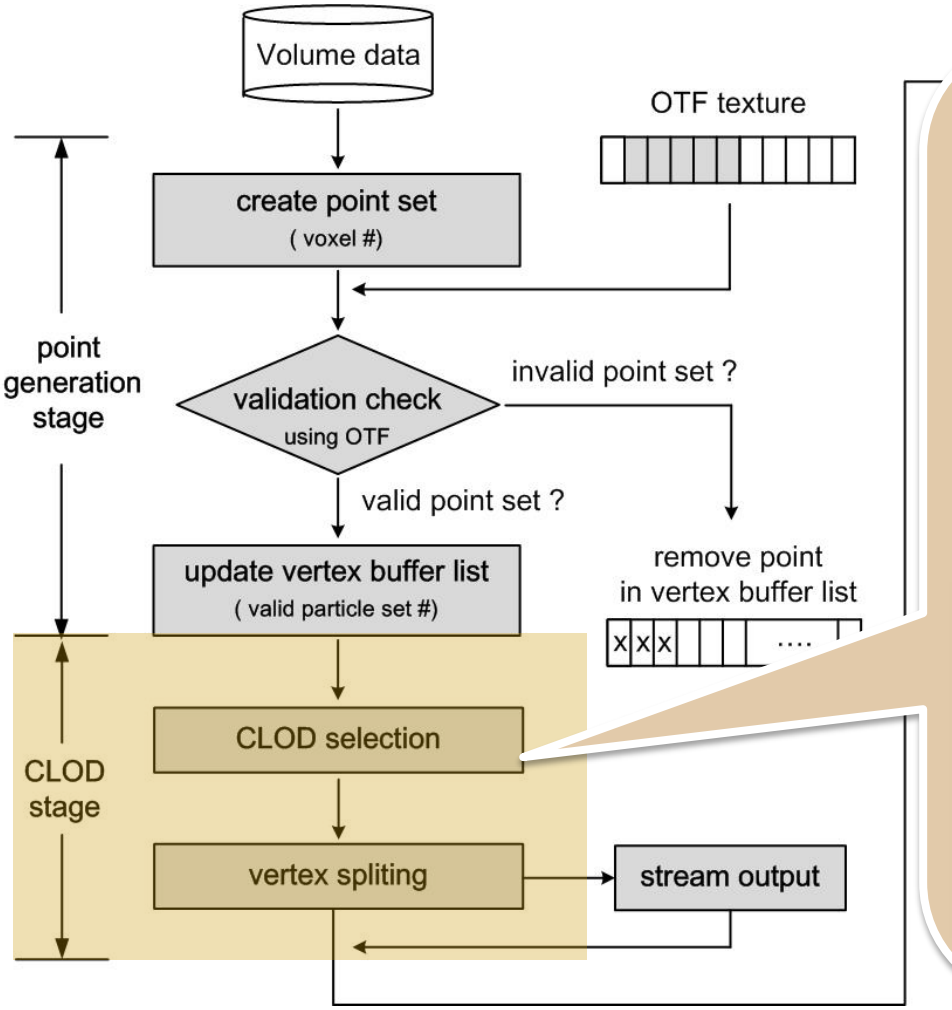
Projection of Point Primitive

- Validation check



Input volume data

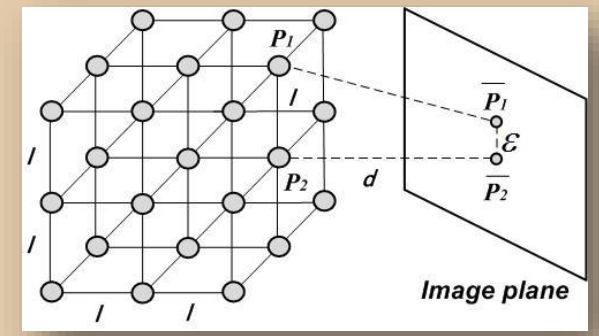
Projection of Point Primitive



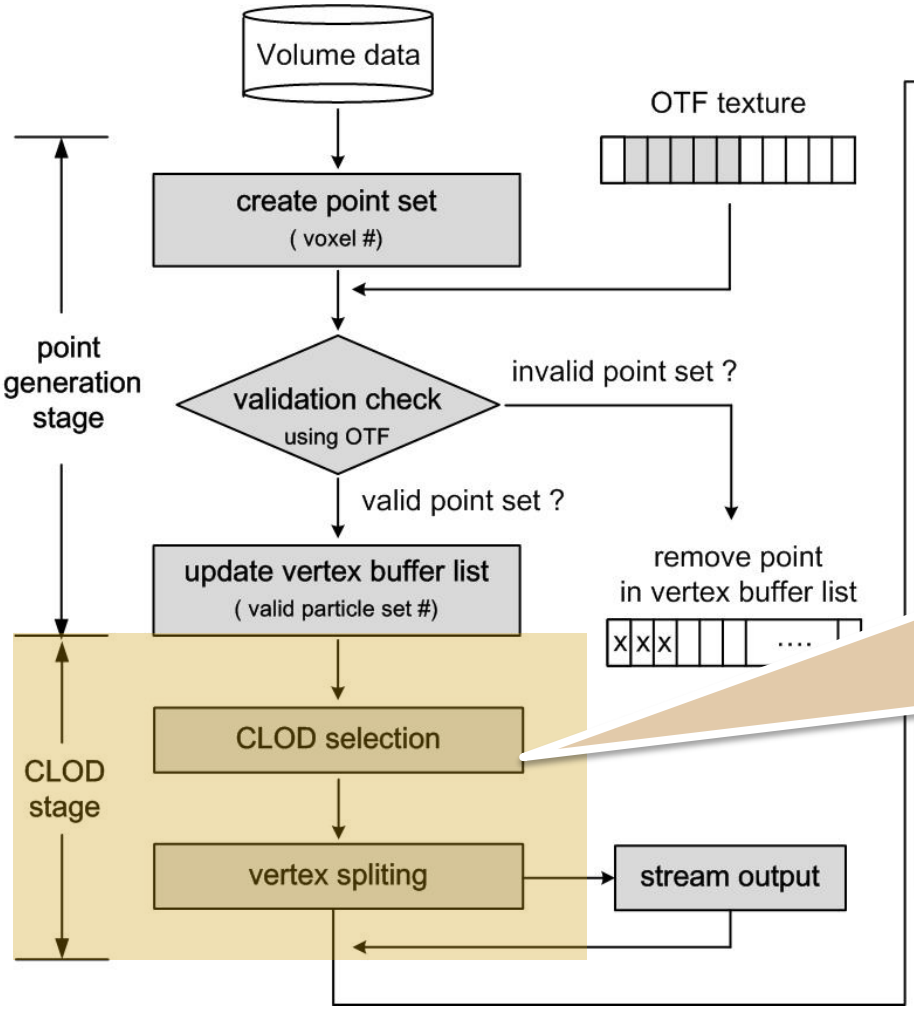
Point CLOD

- reduce hole
- use screen space error (ε), distance of points (l)

$$\varepsilon = l / d^2$$

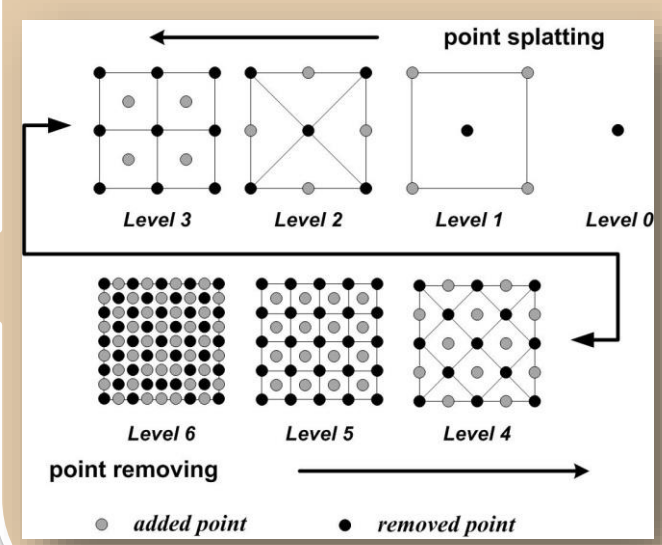


Projection of Point Primitive



Point CLOD

- vertex splitting to transit upper level
- supply additional point set to fill hole

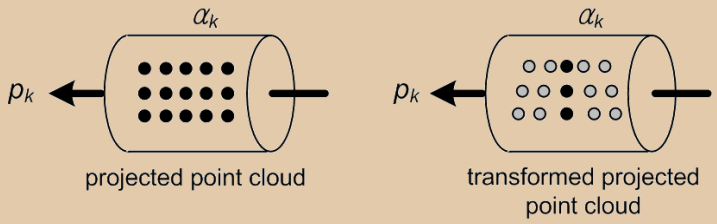


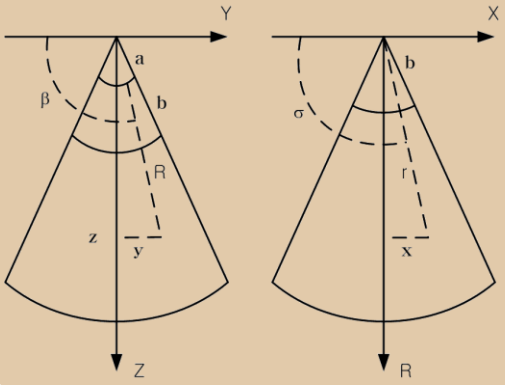
Coordinates conversion

For each fragment,

view-space position(α_k)

- compute by intersecting the viewing ray passing through projection position(P_k)



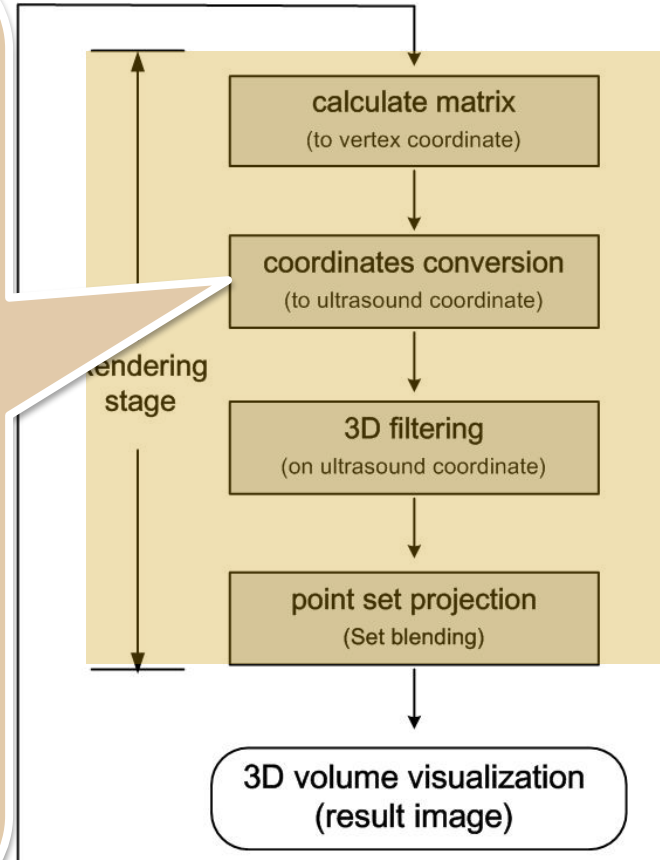


$$R = \sqrt{y^2 + z^2} - a$$

$$\beta = \frac{\pi}{2} + \tan^{-1}\left(\frac{y}{z}\right)$$

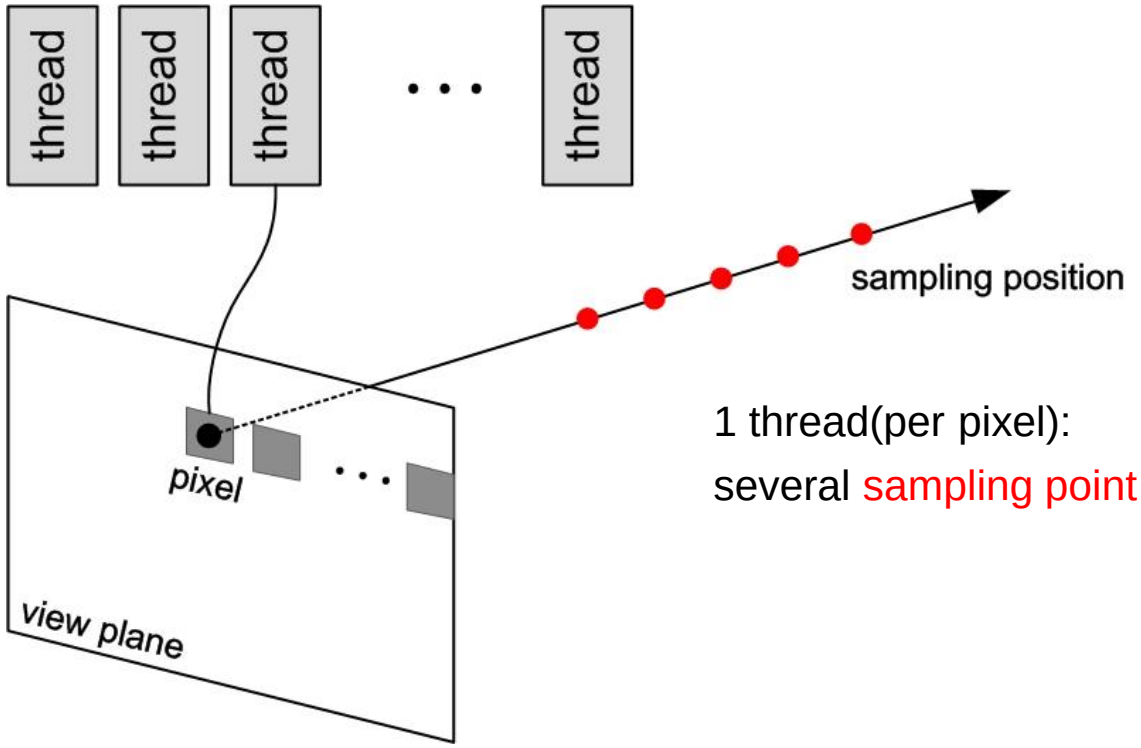
$$\sigma = \frac{\pi}{2} + \tan^{-1}\left(\frac{x}{R}\right)$$

$$r = \sqrt{x^2 + R^2} - b$$



Rendering stage (previous)

FRAGMENT SHADER BASED VOLUME RAY-CASTING



1 thread(per pixel):

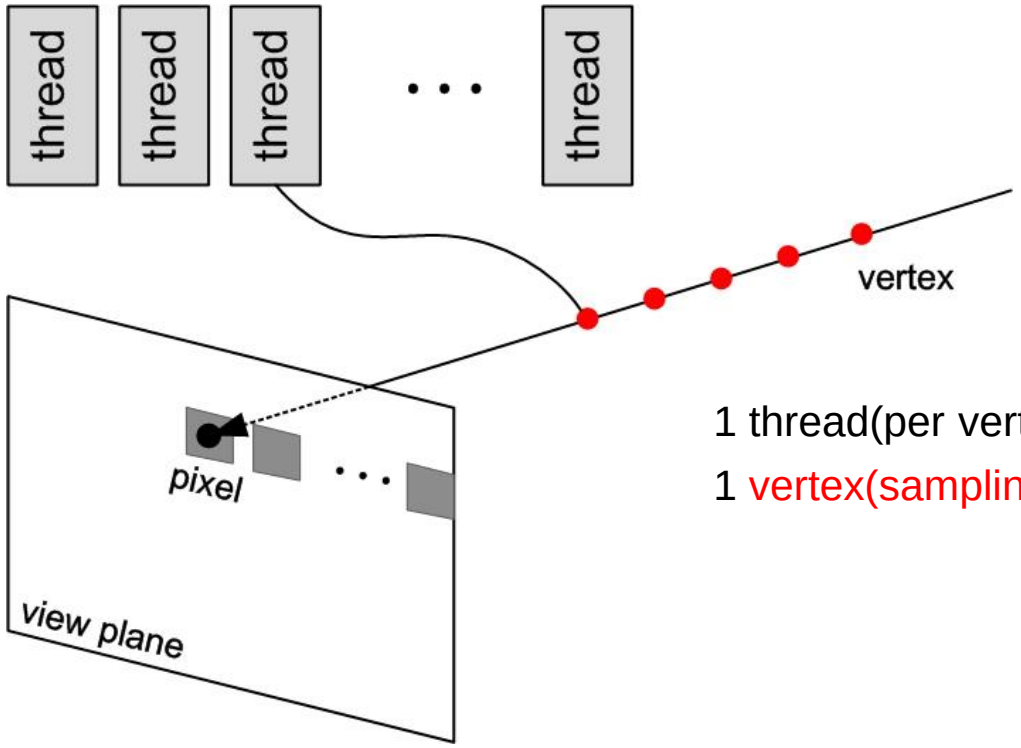
several sampling point x operations

indexing using OTF
Coordinates conversion
shading
color composition

3D filtering (very difficult)

Rendering stage (OURs)

VERTEX SHADER BASED PROJECTION OF POINT PRIMITIVE

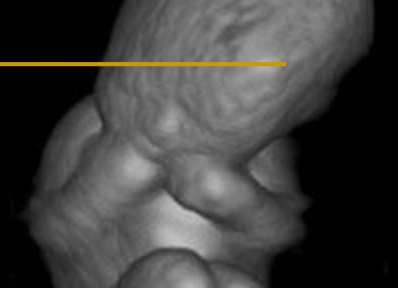


1 thread(per vertex):

1 vertex(sampling point) x operations

indexing using OTF
Coordinates conversion
3D filtering, also
shading
color composition

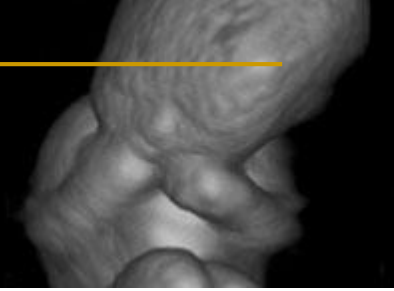
Rendering stage (OURs)



- Contributions
  - balancing between fragment and vertex shader
  - Real-time 3D filtering & coordinates conversion
    - No need pre-processing
  - No needs additional data structure
    - for speed up(empty skipping, early ray termination)
    - we remove non-interest vertex on the vertex creation stage

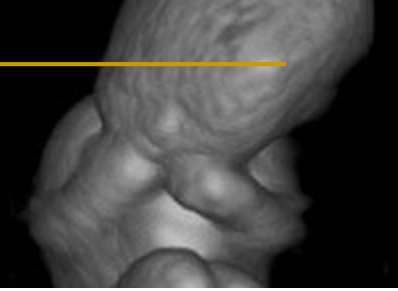
Experimental Results

Environment



- Intel Core2Duo Processor 6400(2.13GHz)
- 4 GB main memory
- NVIDIA GTX260 (512 MB video memory)
- DirectX 10.0
- Shader version 4.0
- HLSL (high level shading language)
- Viewport: 512x512

Experimental Results



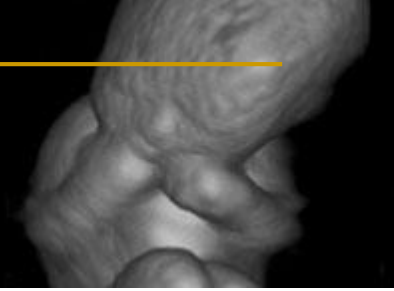
- Ultrasound volume data
  - Phantom, fetus data
    - Clinical data from Medison Co. LTD.
- Performance (rendering speed)

CASE #	GPU-based RC	RC with 3D filtering	Projection of point primitive	PPP with 3D filtering
Phantom 256 x 128 x 110	183 *fps*	6 *fps*	32 *fps*	24 *fps*
Fetus 384 x 208 x 96	165 *fps*	5 *fps*	30 *fps*	17 *fps*

✓ Ray-casting with 3D filtering is time consuming task

✓ Fast 3D filtering is possible in our method

Experimental Results



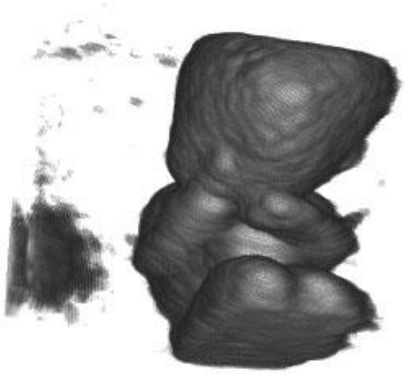
- Performance (filtering speed)

Data	Rendering only		Rendering and filtering (average)		Rendering and filtering (Gaussian)	
	Ray-casting	PPP	3x3x3	5x5x5	3x3x3	5x5x5
Phantom	183 *fps*	34 *fps*	24 *fps*	10 *fps*	24 *fps*	10 *fps*
Fetus	165 *fps*	30 *fps*	17 *fps*	7 *fps*	17 *fps*	7 *fps*

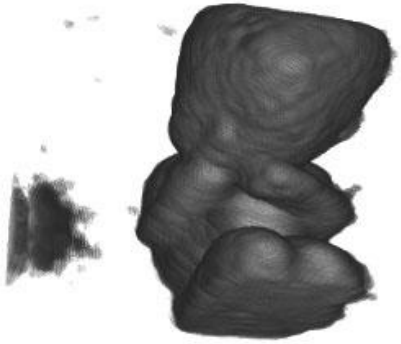
✓ no difference of filtering time between average and Gaussian kernel

Experimental Results

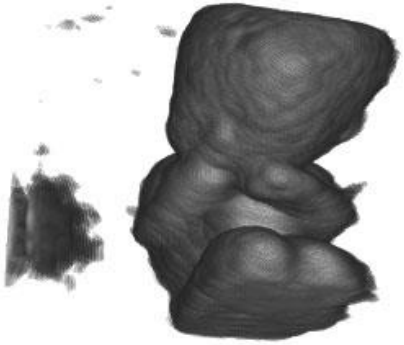
- result images



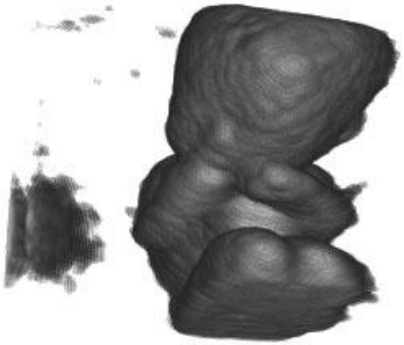
None 3D filtering



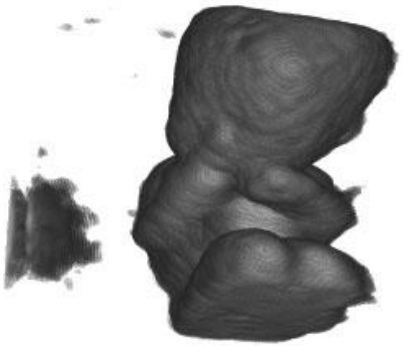
3x3x3 avg. filtering



5x5x5 avg. filtering



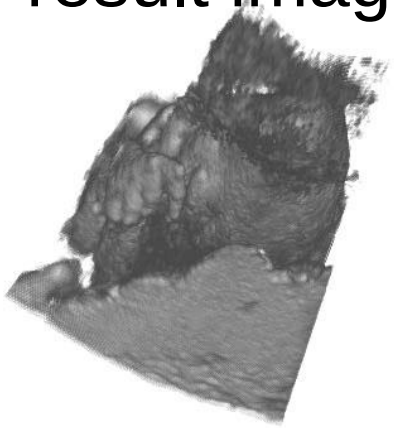
3x3x3 Gau. filtering



5x5x5 Gau. filtering

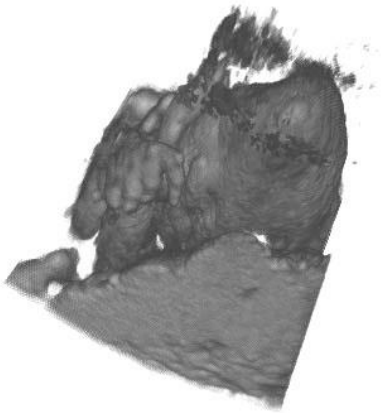
Experimental Results

- result images

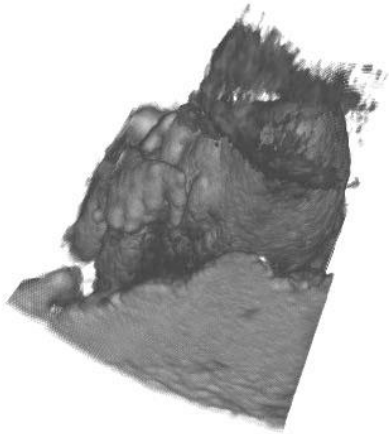


None 3D filtering

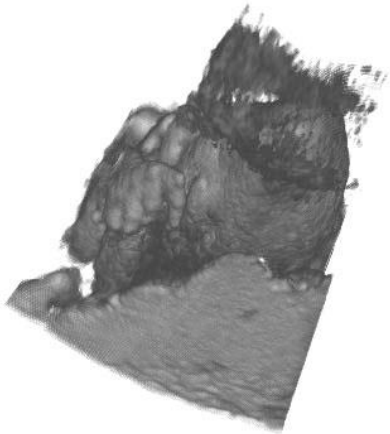
✓ 3x3x3 average filter is better than other filter (speed & quality)



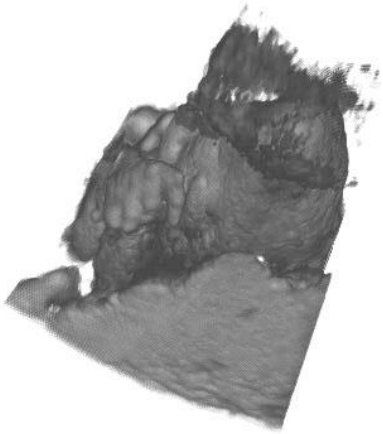
3x3x3 avg. filtering



5x5x5 avg. filtering

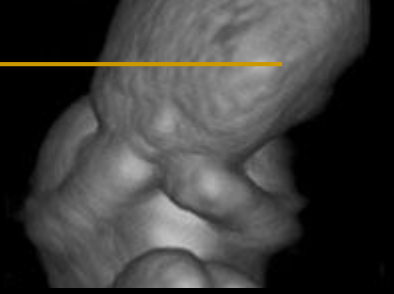


3x3x3 Gau. filtering

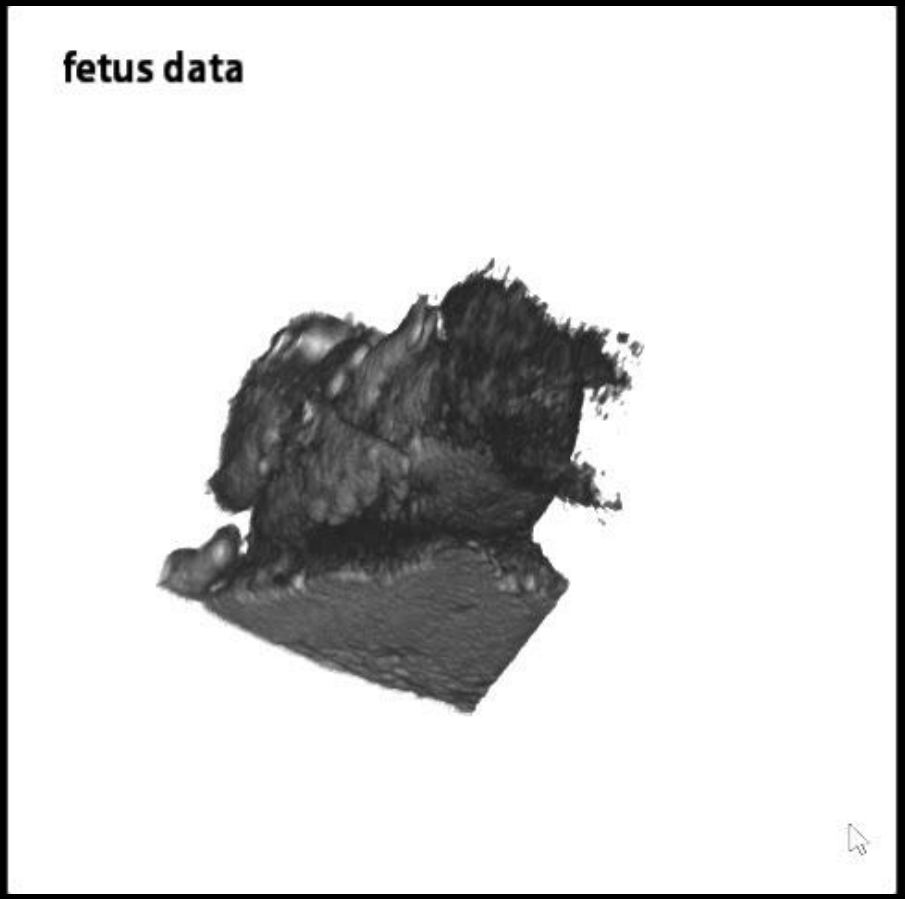


5x5x5 Gau. filtering

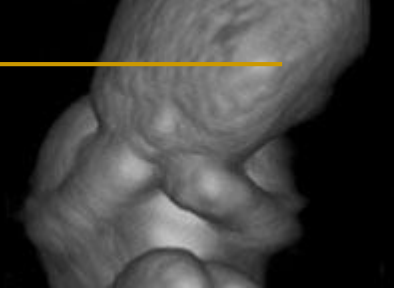
Experimental Results



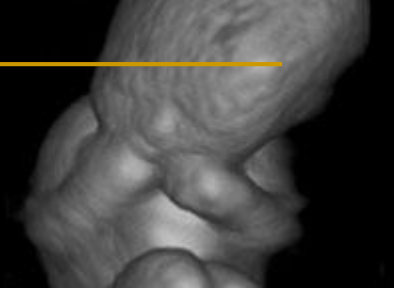
- movie clip



Conclusion



- Previous GPU-based rendering method is performed on fragment shader
  - requires high computational cost
  - hard to performing 3D filtering calculation in real-time
- Ultrasound visualization method
  - convert volume coordinates into ultrasound coordinates
  - remove noise with execution 3D filtering during rendering step



Thanks for your attention