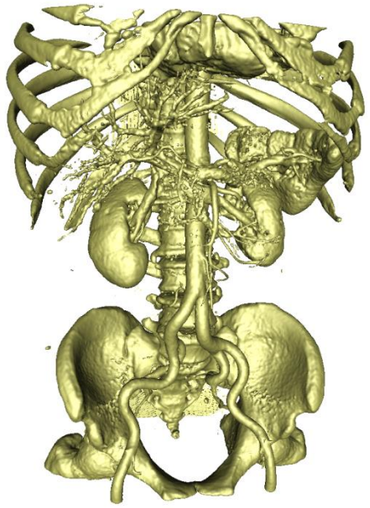
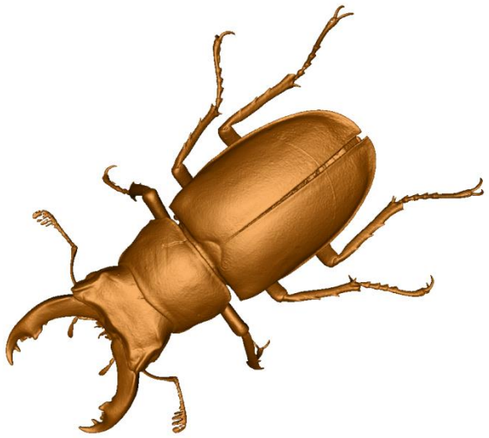
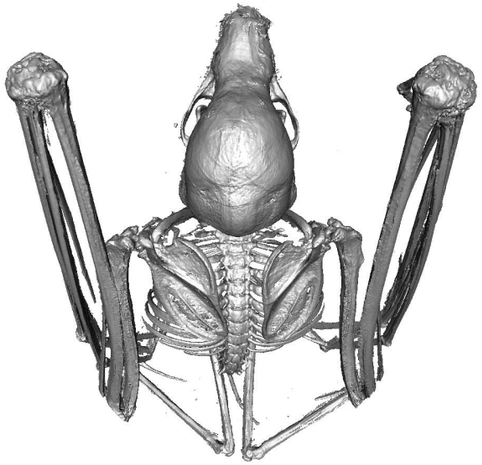
Computation-to-Core Mapping Strategies for Iso-Surface Volume Rendering on GPUs

Junpeng Wang Fei Yang Yong Cao

junpeng@vt.edu feiy@nvidia.com yongcao@vt.edu







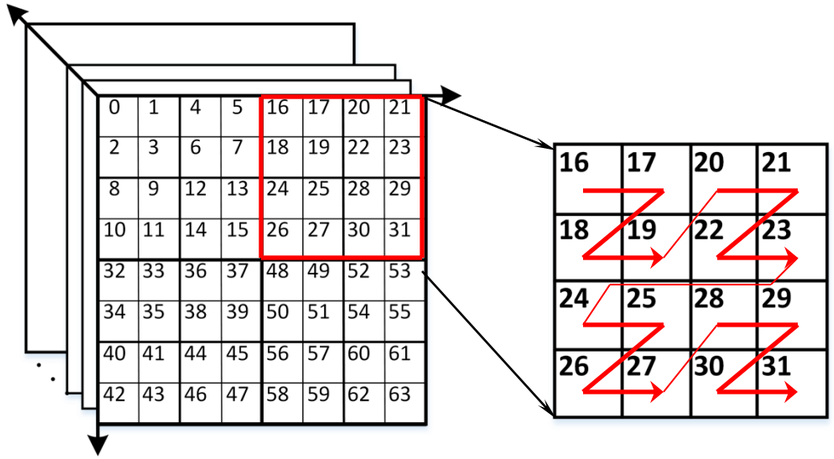
VirginiaTech Invent the Future

NVIDIA

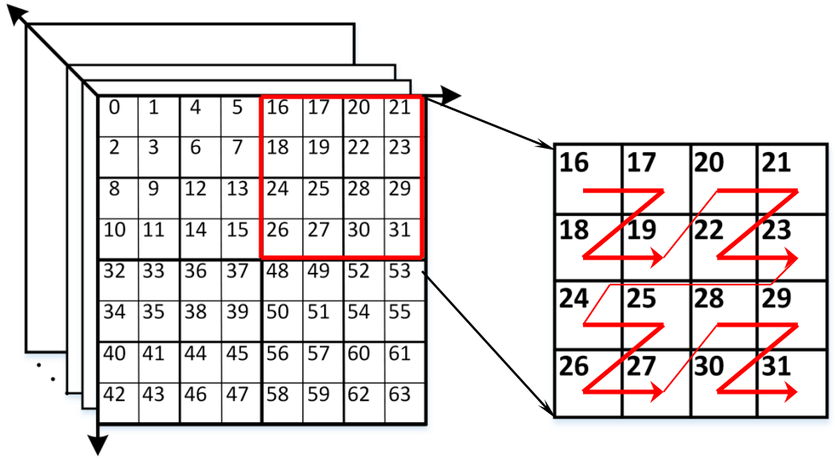
Overview

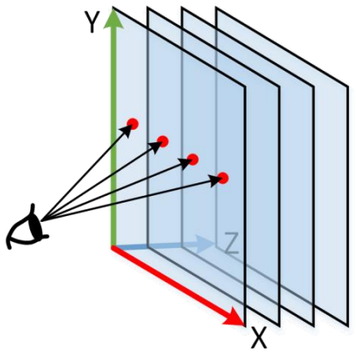
- Motivation
  - Texture Cache
  - Load Balancing
- Contribution
  - Warp Marching
  - Analysis on Load Imbalance
- Results
  - Cache Performance
  - Load Balancing
- Conclusion

Motivation


0 1 4 5 16 17 20 21
2 3 6 7 18 19 22 23
8 9 12 13 24 25 28 29
10 11 14 15 26 27 30 31
32 33 36 37 48 49 52 53
34 35 38 39 50 51 54 55
40 41 44 45 56 57 60 61
42 43 46 47 58 59 62 63
16 17 20 21
18 19 22 23
24 25 28 29
26 27 30 31

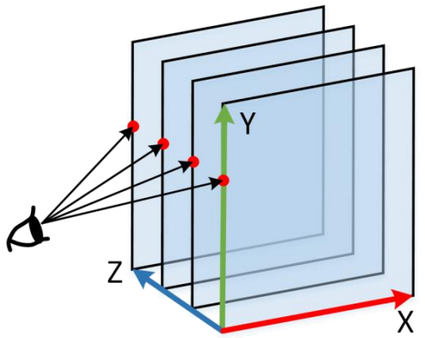
Motivation





facing XY

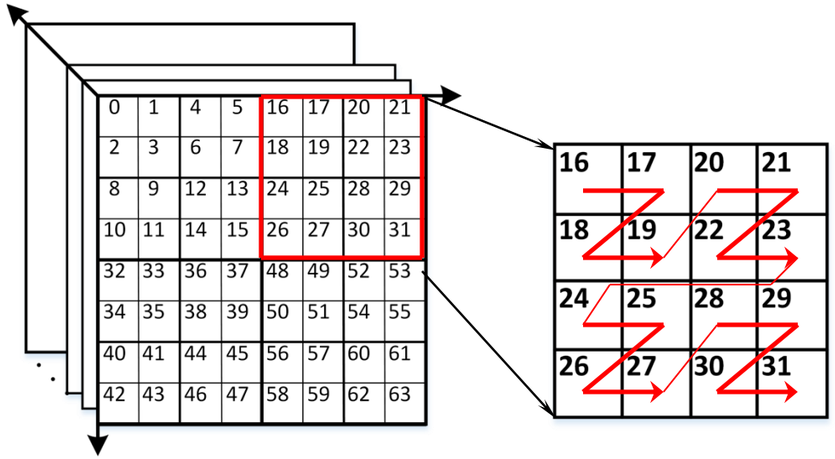
Perpendicular



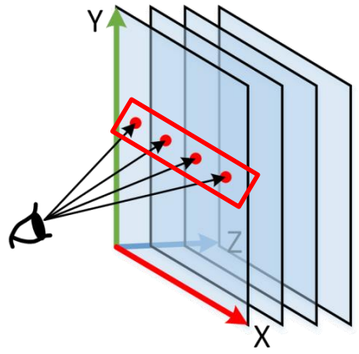
facing ZY

Parallel

Motivation

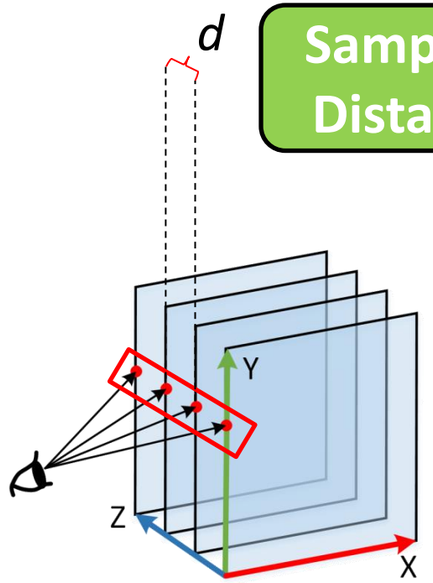


Viewing Direction



facing XY

Perpendicular

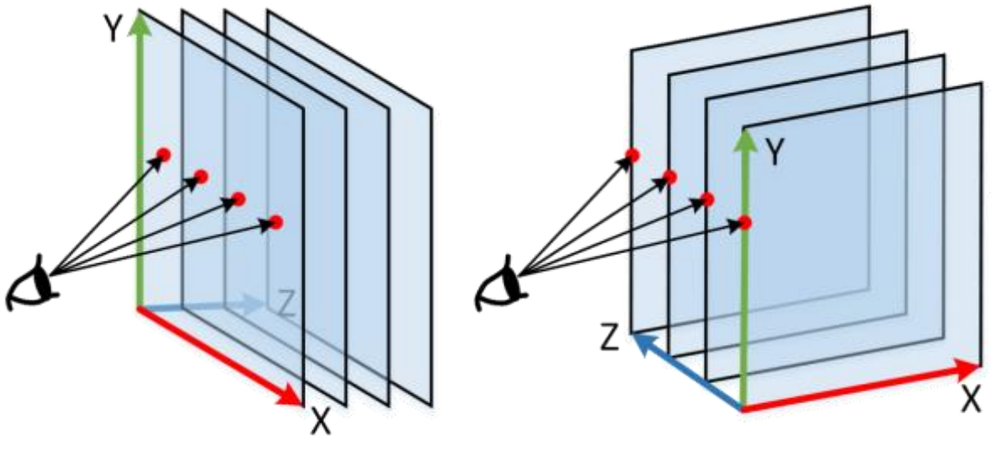


facing ZY

Parallel

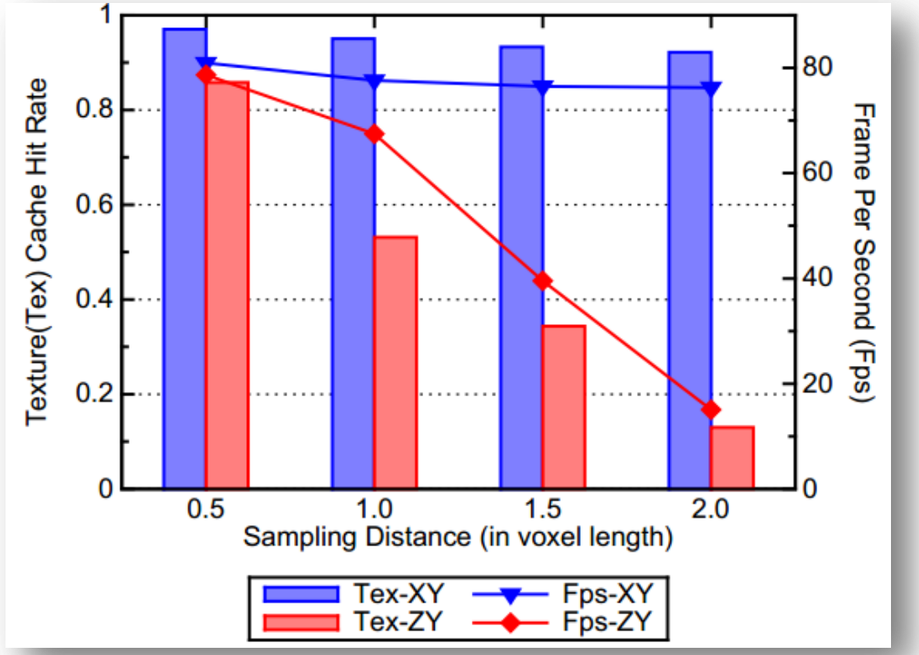
Sampling Distance

Motivation



Perpendicular

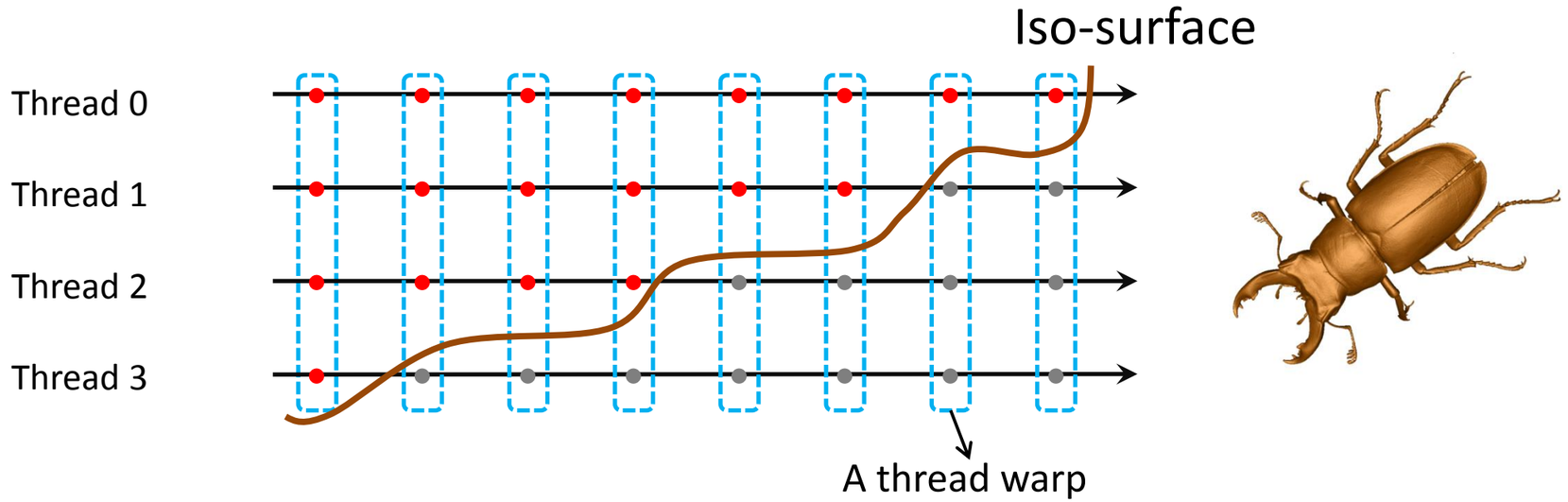
Parallel



GPU: GTX GeForce Titan
Volume size: 1024x1024x1024 x 8bit
Rendered image size: 512x512

Motivation

Ray-casting



Load imbalance among threads of a warp

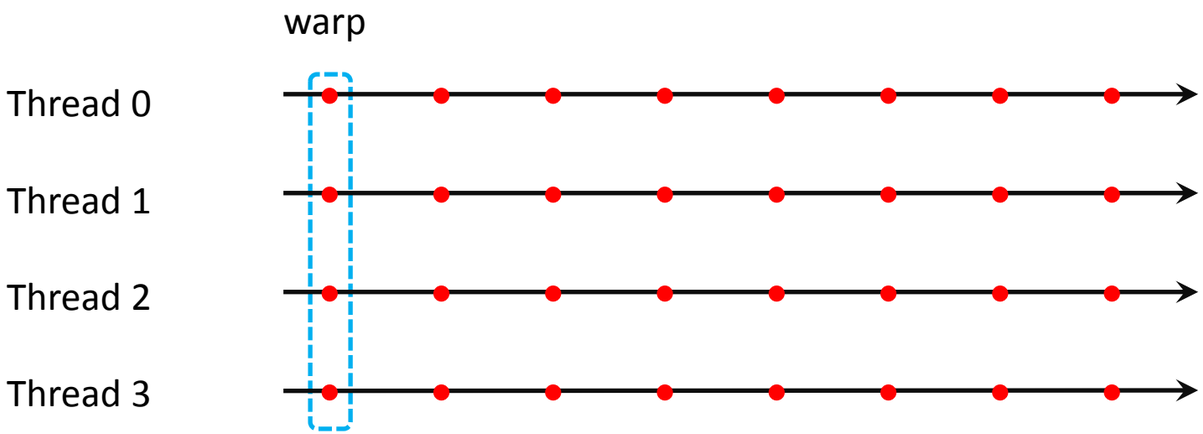
Contribution

1. Improve the texture cache performance by minimizing the memory stride inside a **warp** of GPU threads.

2. Address load balancing issues in both thread-level and SM-level

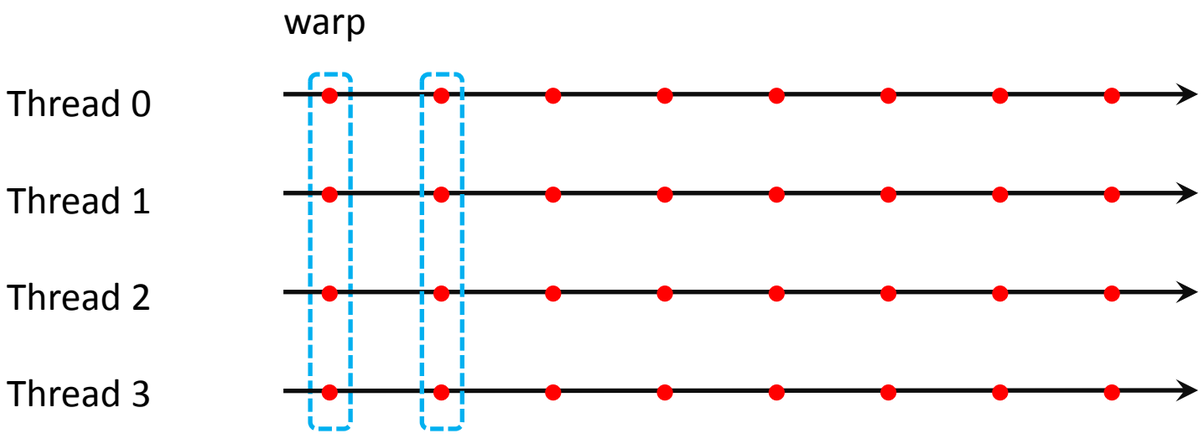
Warp Marching

Contribution



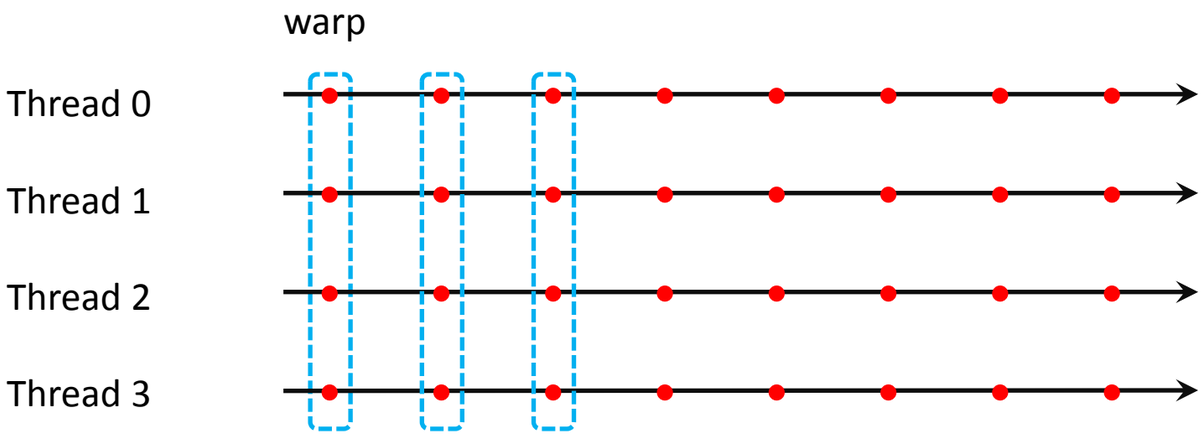
Map one thread to one ray

Contribution



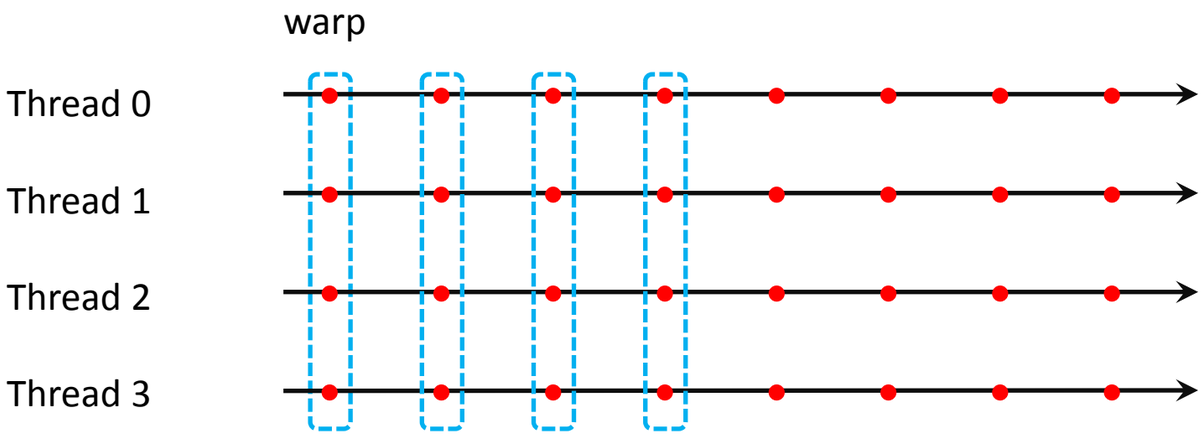
Map one thread to one ray

Contribution



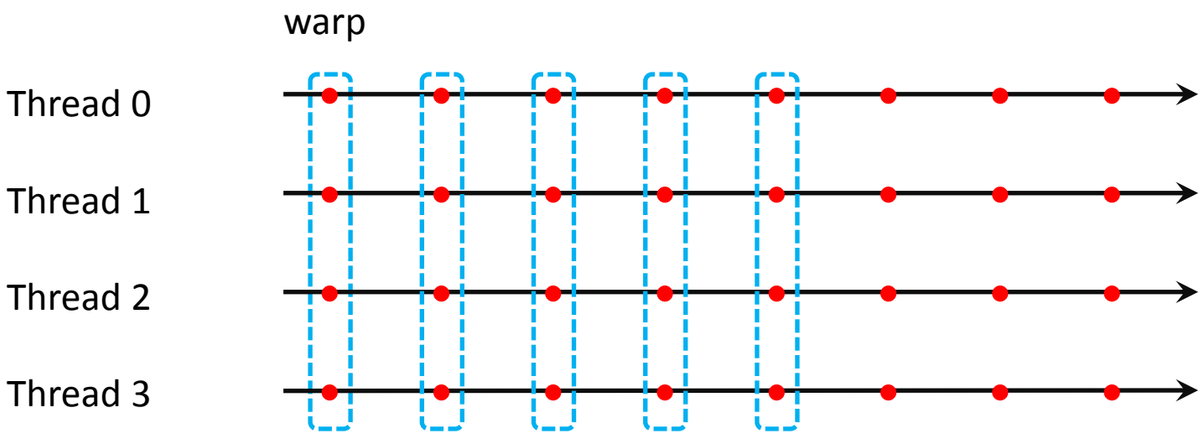
Map one thread to one ray

Contribution



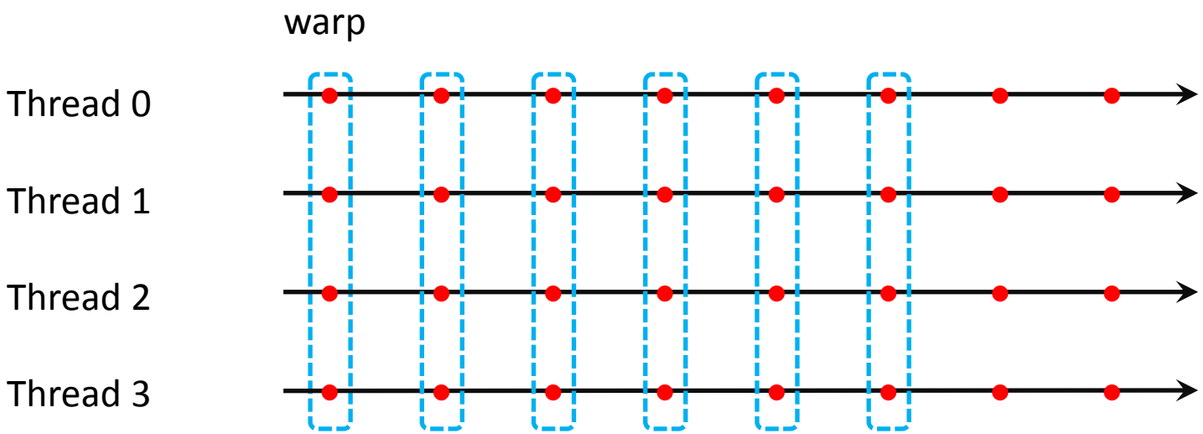
Map one thread to one ray

Contribution



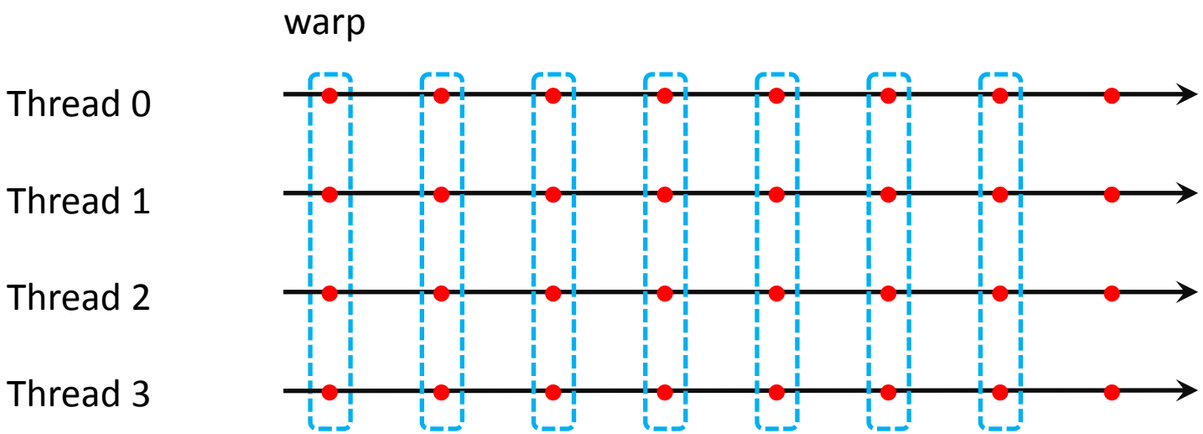
Map one thread to one ray

Contribution



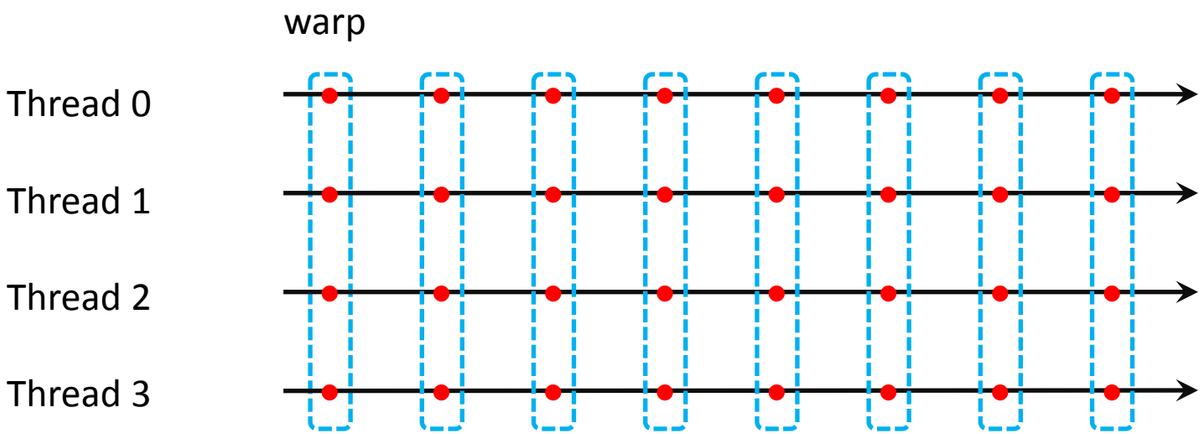
Map one thread to one ray

Contribution



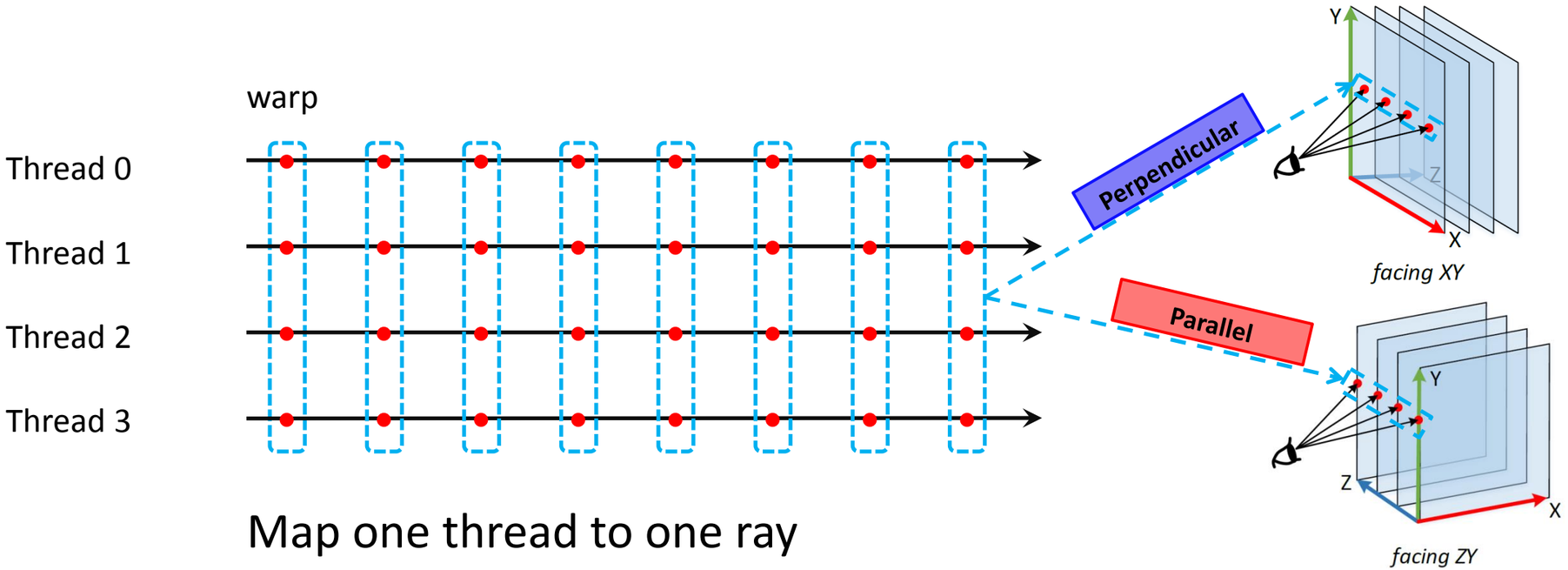
Map one thread to one ray

Contribution



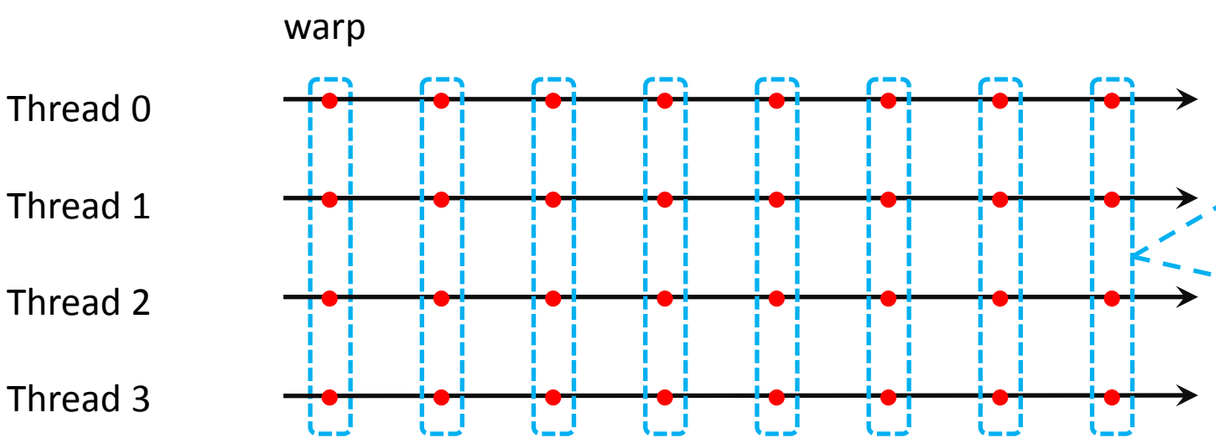
Map one thread to one ray

Contribution

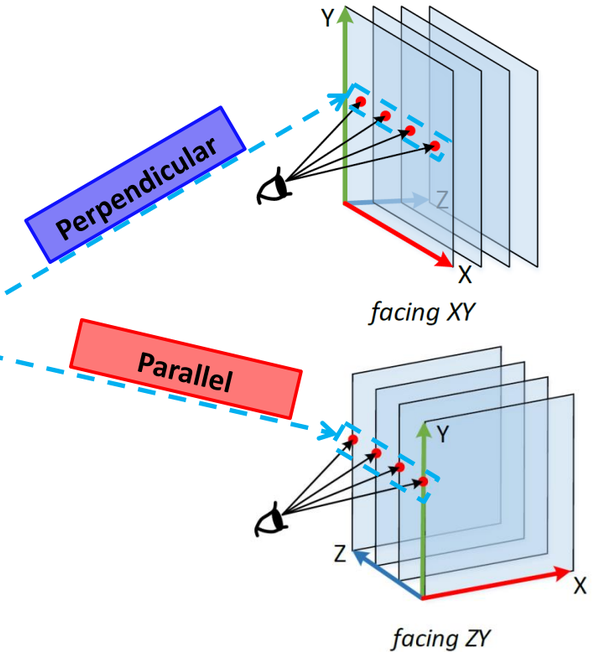


Map one thread to one ray

Contribution

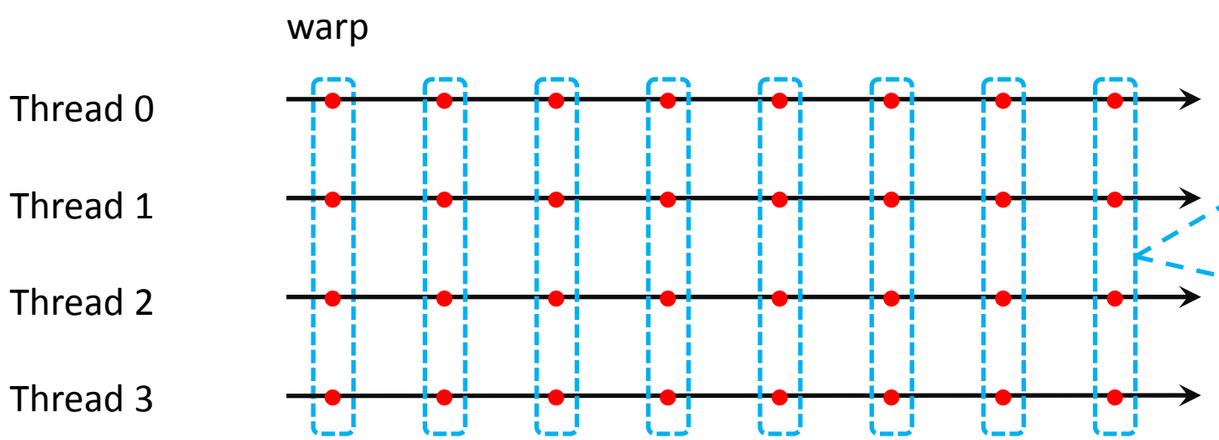


Map one thread to one ray

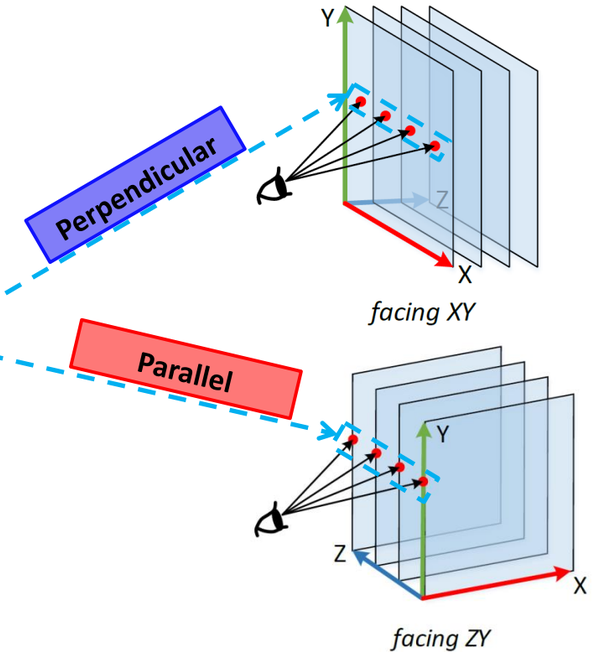


The Standard

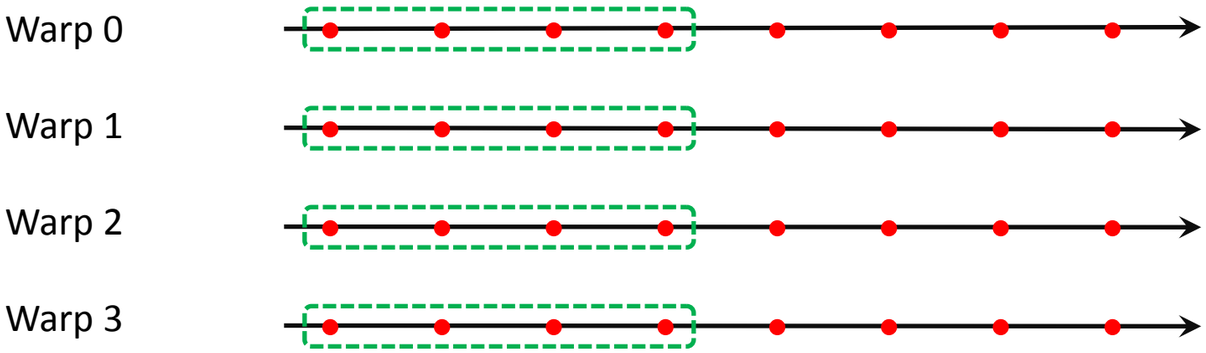
Contribution



Map one thread to one ray

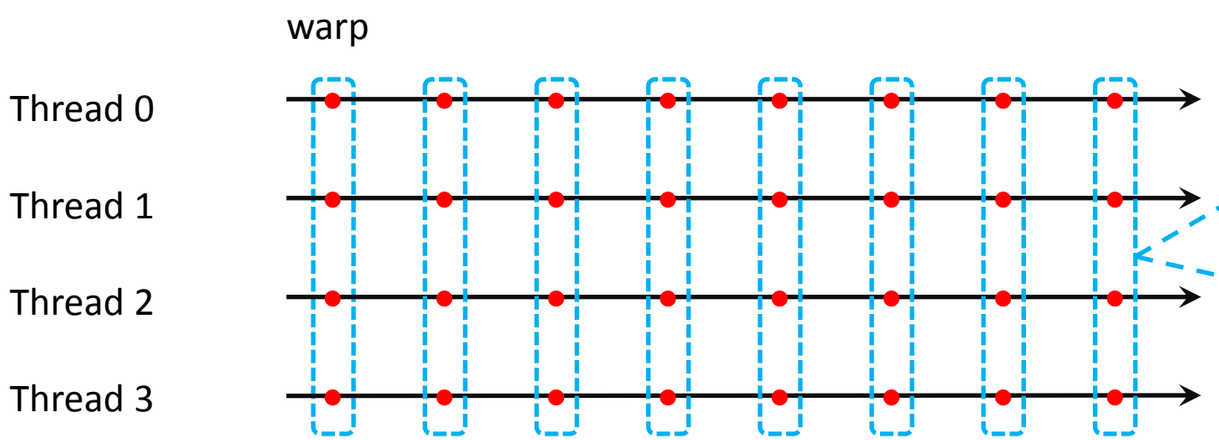


The Standard

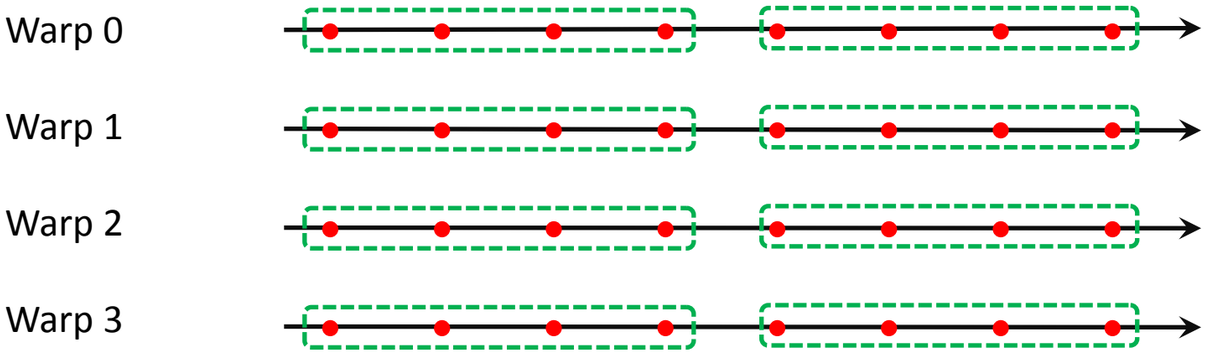


Map one warp of threads to one ray

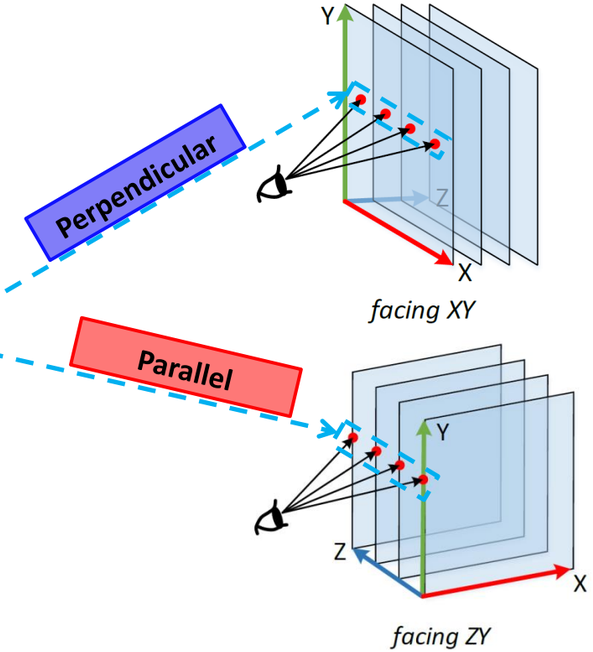
Contribution



Map one thread to one ray

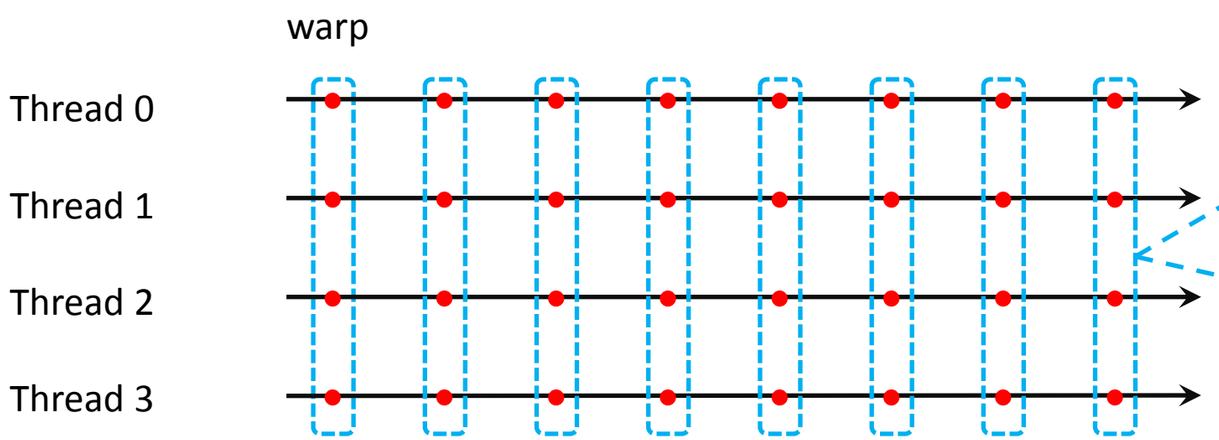


Map one warp of threads to one ray

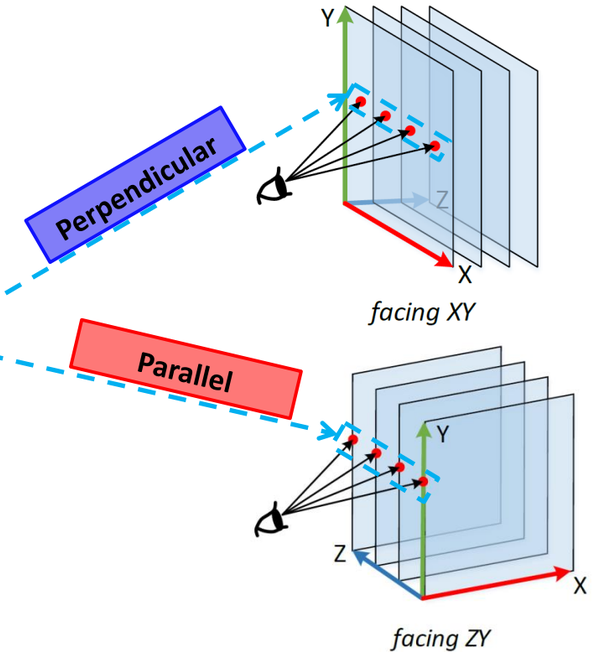


The Standard

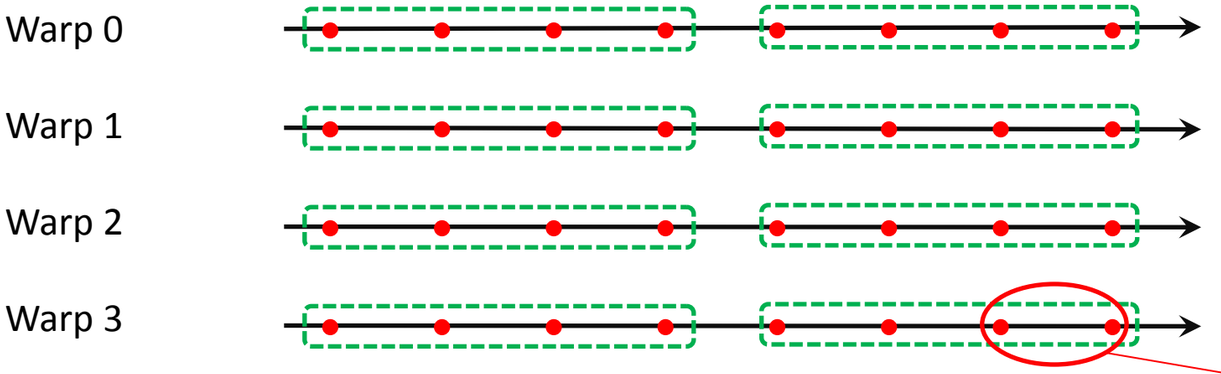
Contribution



Map one thread to one ray



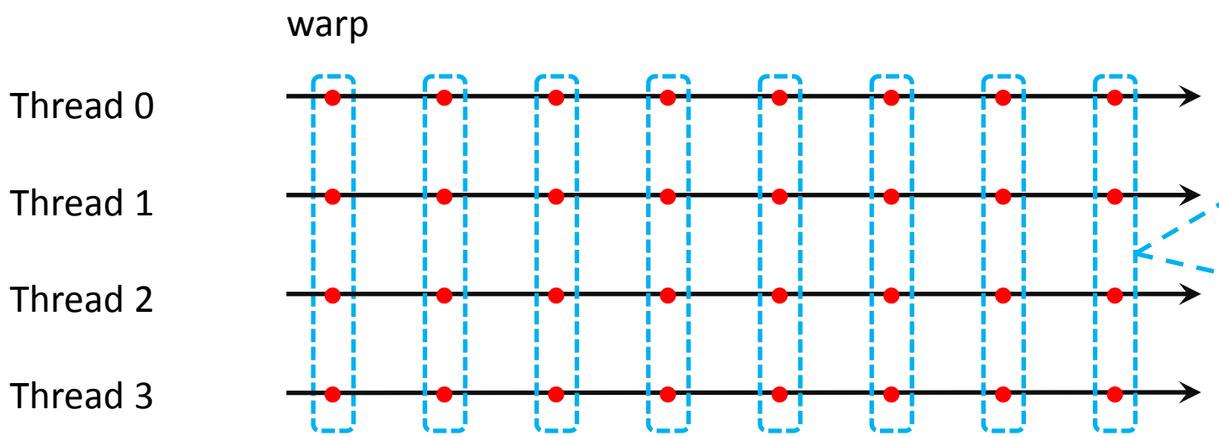
The Standard



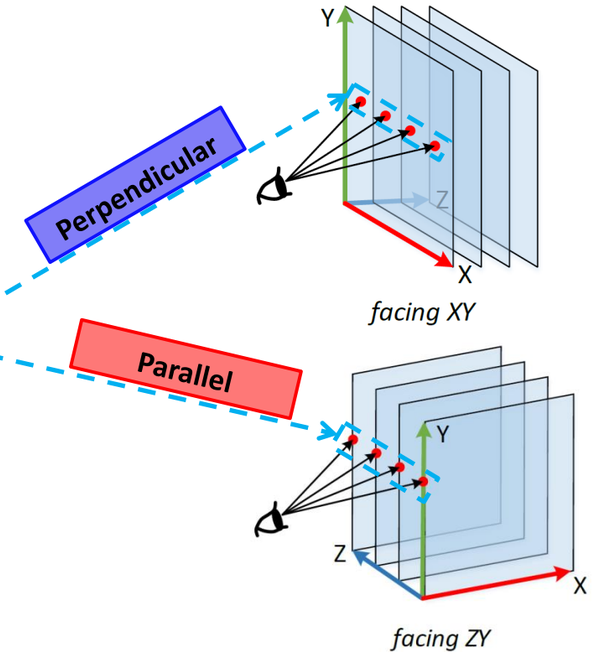
Map one warp of threads to one ray

The distance is easy to control in Warp Marching

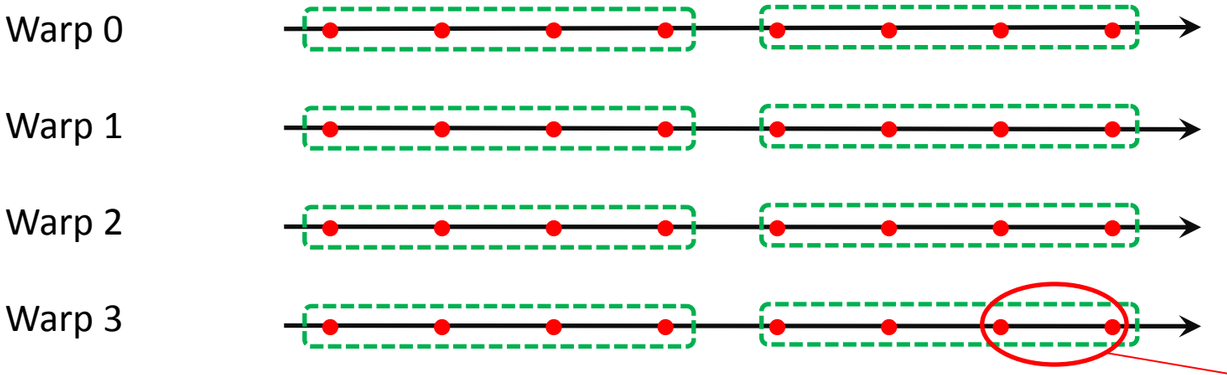
Contribution



Map one thread to one ray



The Standard

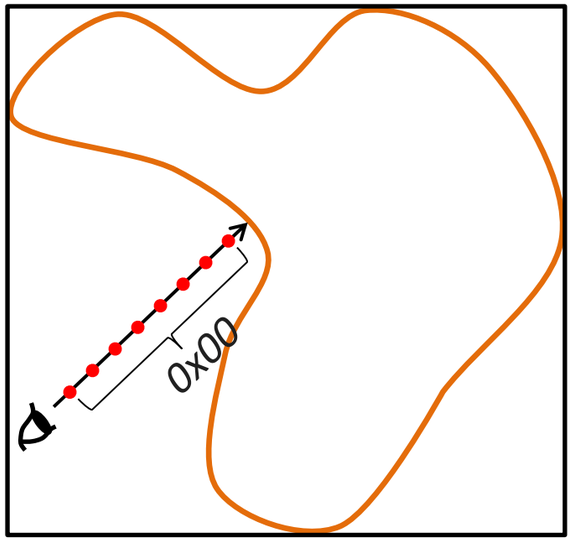


Map one warp of threads to one ray

The distance is easy to control in Warp Marching

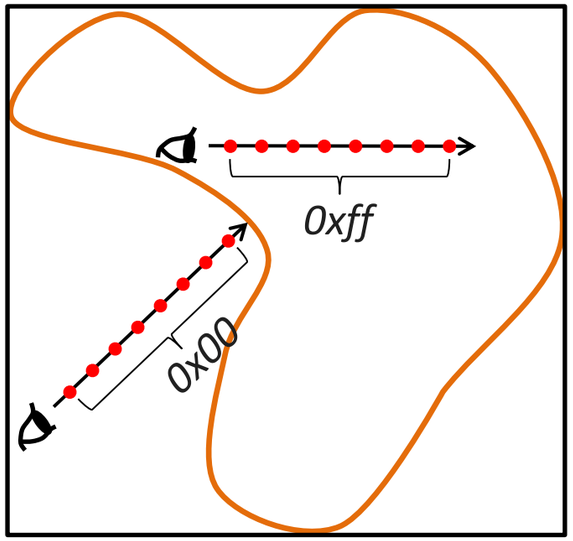
Warp Marching

Contribution



(1) Outside iso-surface, __ballot(warpActive) = *0x00*

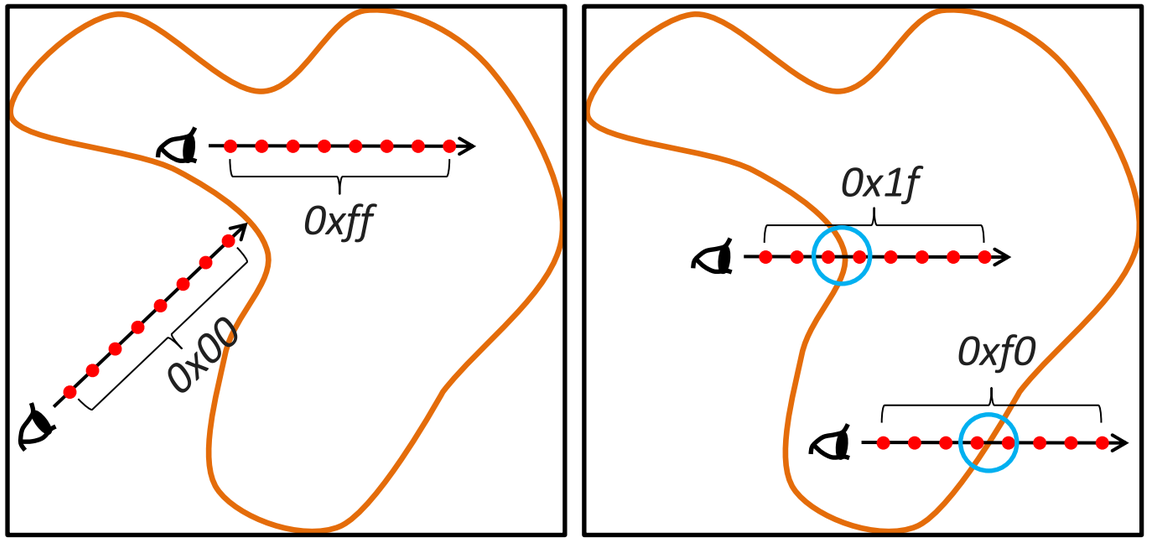
Contribution



(1) Outside iso-surface, __ballot(warpActive) = *0x00*

(2) Inside iso-surface, __ballot(warpActive) = *0xff*

Contribution

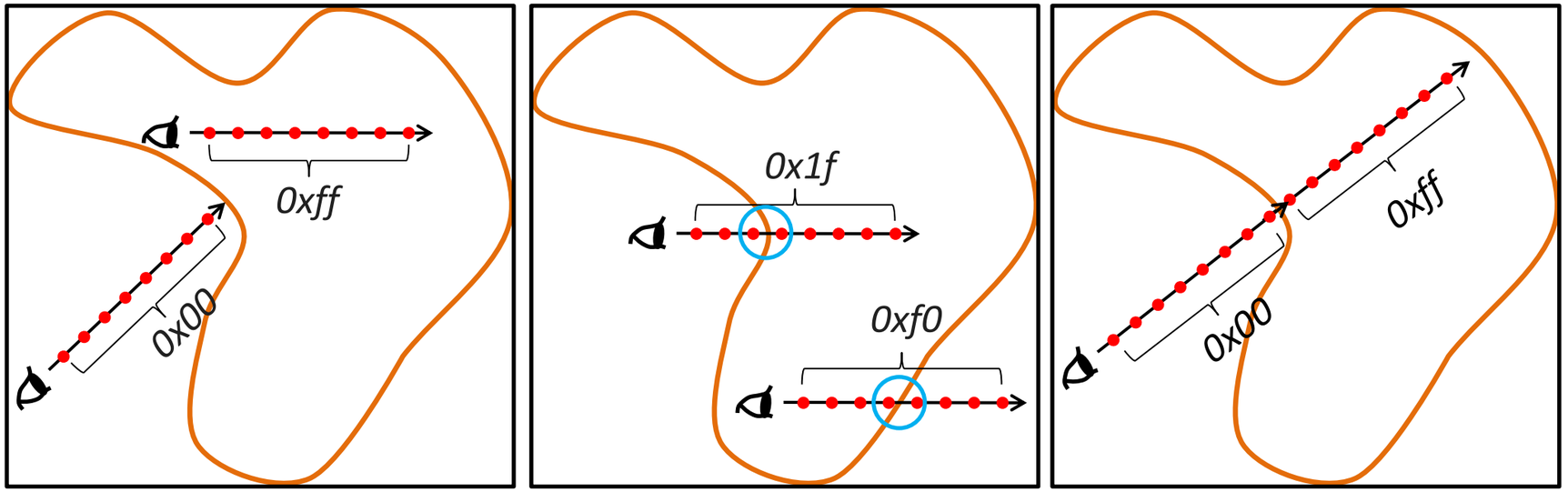


(1) Outside iso-surface, __ballot(warpActive) = *0x00*

(2) Inside iso-surface, __ballot(warpActive) = *0xff*

(3) Across iso-surface, __ballot(warpActive) = (*0x00*, *0xff*)

Contribution

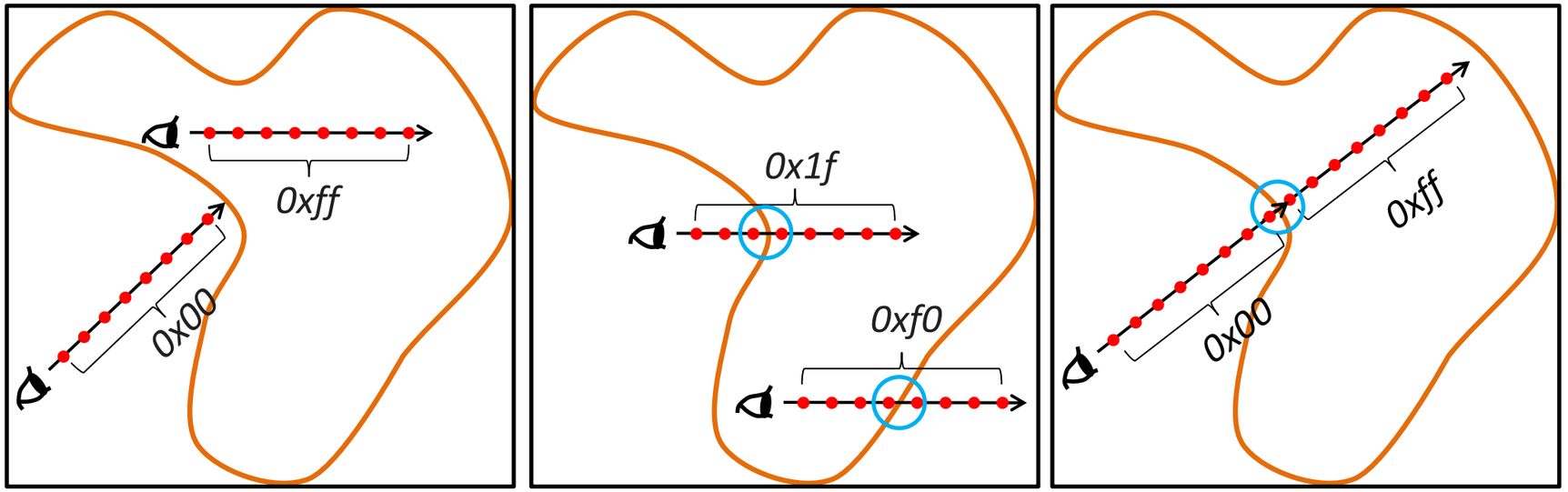


(1) Outside iso-surface, __ballot(warpActive) = *0x00*

(2) Inside iso-surface, __ballot(warpActive) = *0xff*

(3) Across iso-surface, __ballot(warpActive) = (*0x00*, *0xff*)

Contribution

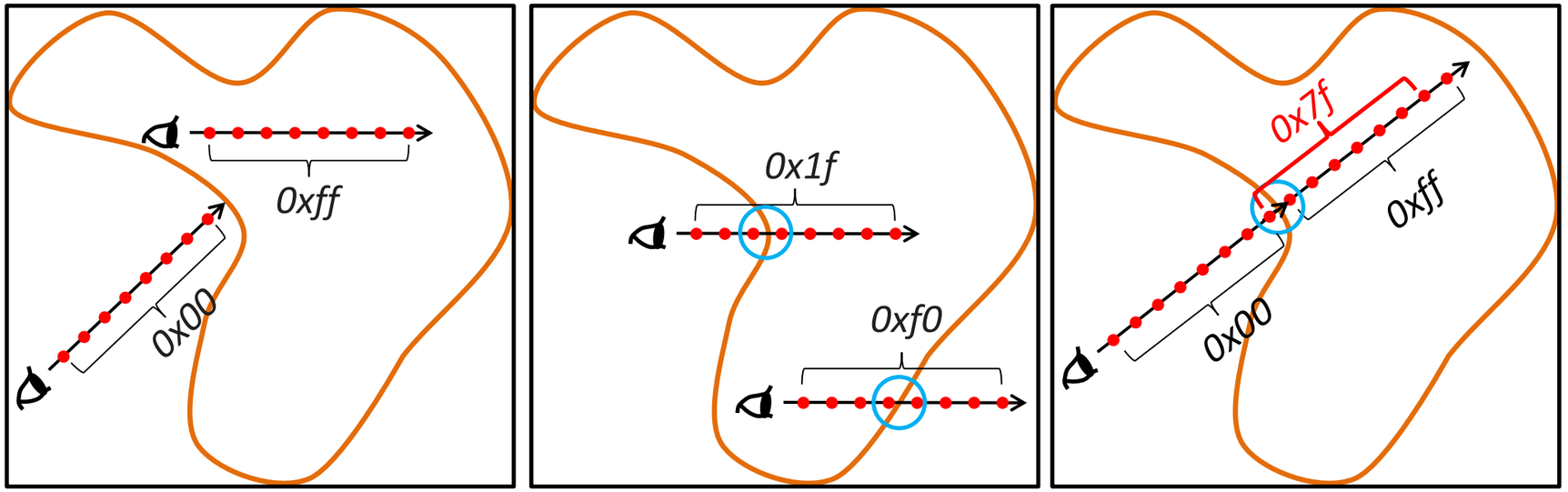


(1) Outside iso-surface, __ballot(warpActive) = *0x00*

(2) Inside iso-surface, __ballot(warpActive) = *0xff*

(3) Across iso-surface, __ballot(warpActive) = (*0x00*, *0xff*)

Contribution

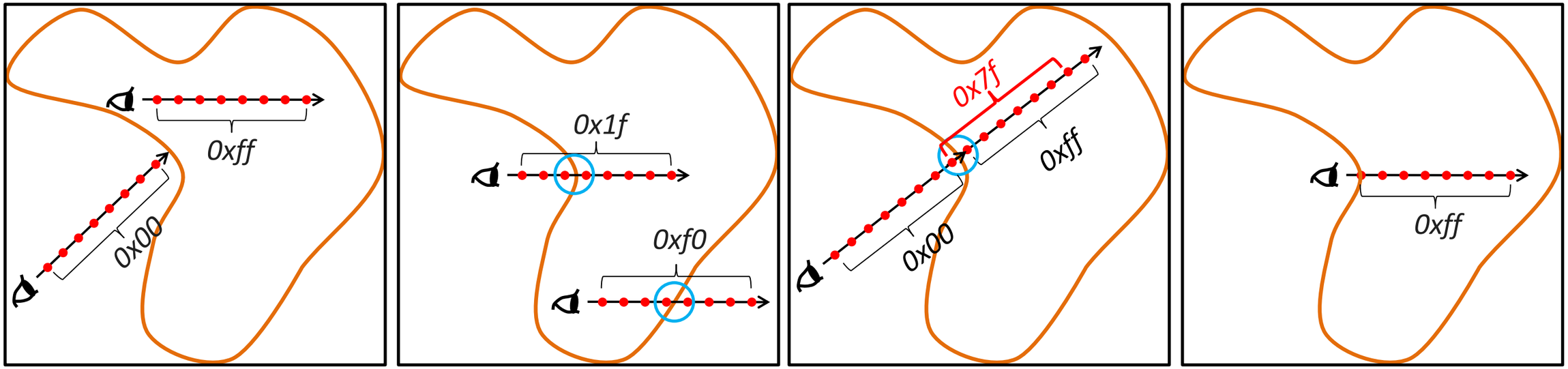


(1) Outside iso-surface, __ballot(warpActive) = *0x00*

(2) Inside iso-surface, __ballot(warpActive) = *0xff*

(3) Across iso-surface, __ballot(warpActive) = (*0x00*, *0xff*)

Contribution

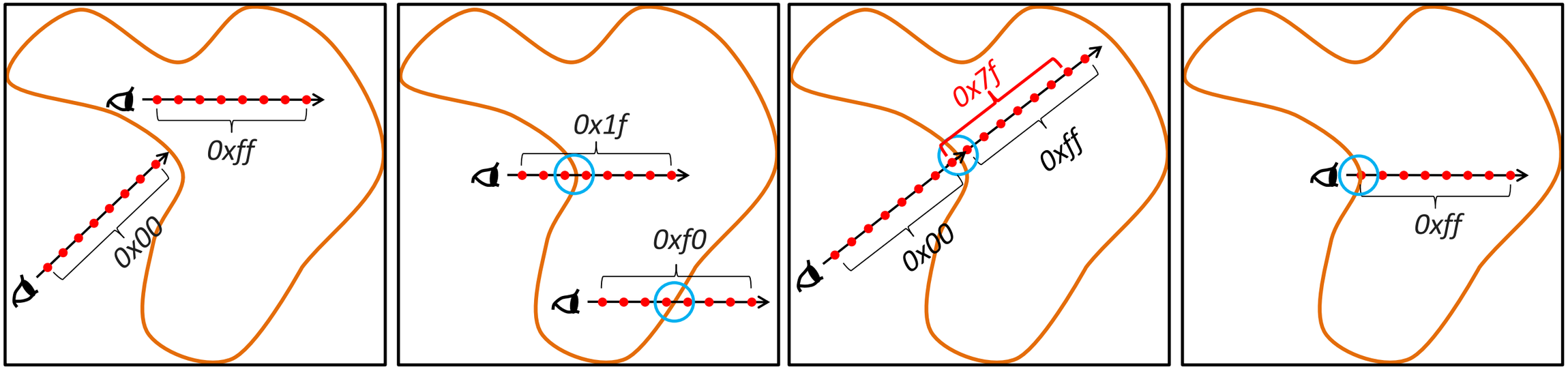


(1) Outside iso-surface, __ballot(warpActive) = *0x00*

(2) Inside iso-surface, __ballot(warpActive) = *0xff*

(3) Across iso-surface, __ballot(warpActive) = (*0x00*, *0xff*)

Contribution



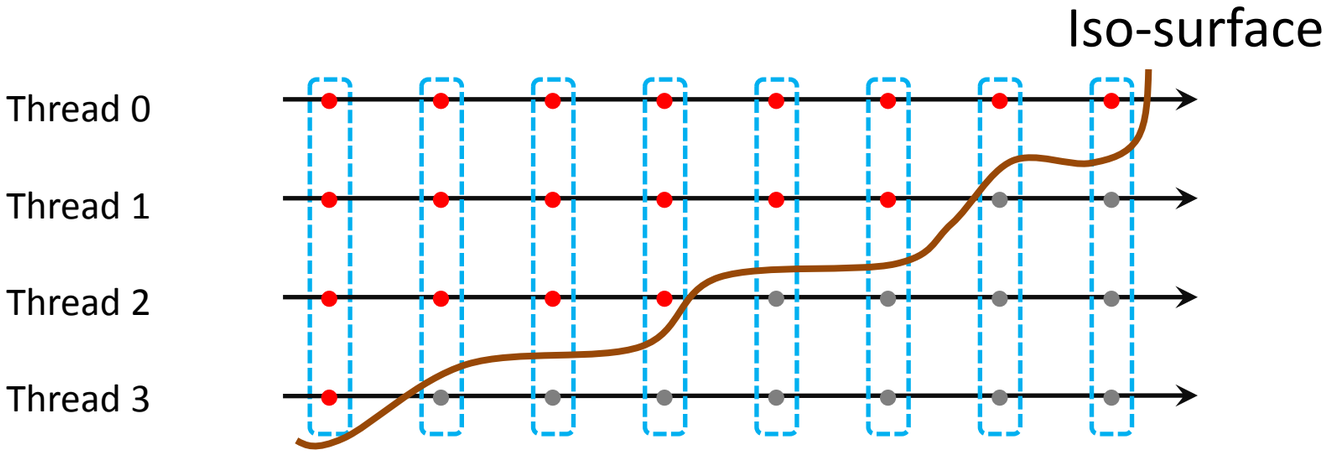
(1) Outside iso-surface, __ballot(warpActive) = *0x00*

(2) Inside iso-surface, __ballot(warpActive) = *0xff*

(3) Across iso-surface, __ballot(warpActive) = (*0x00*, *0xff*)

Load Imbalance (Thread-Level)

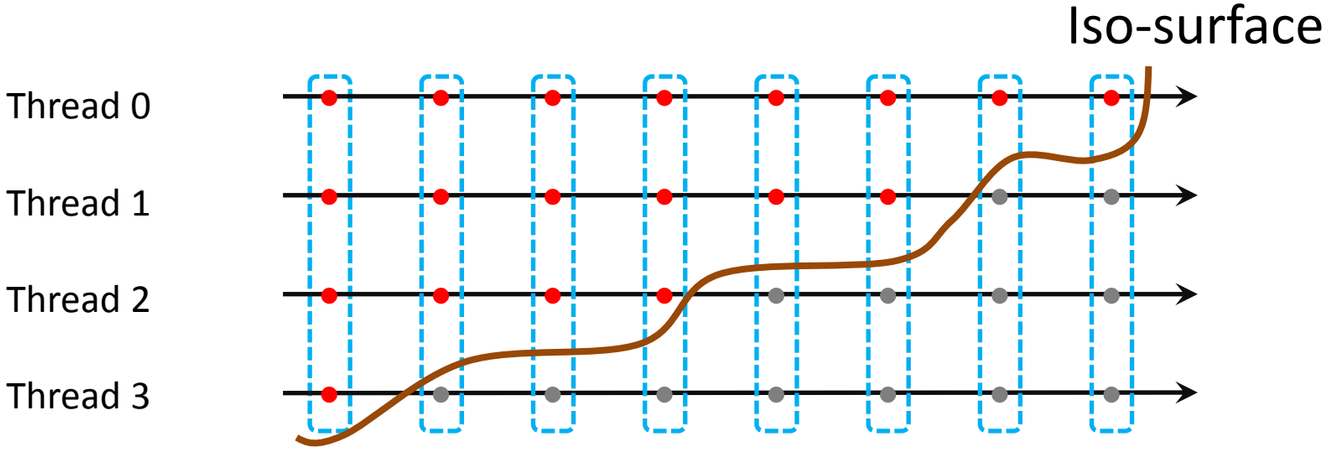
The Standard



Load imbalance happens many times in a warp

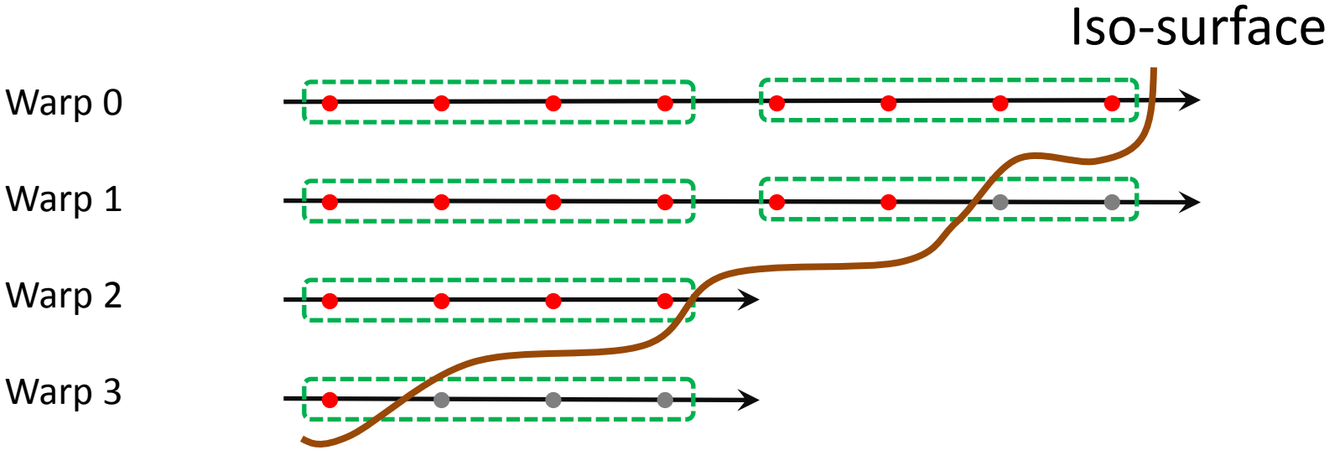
Load Imbalance (Thread-Level)

The Standard



Load imbalance happens many times in a warp

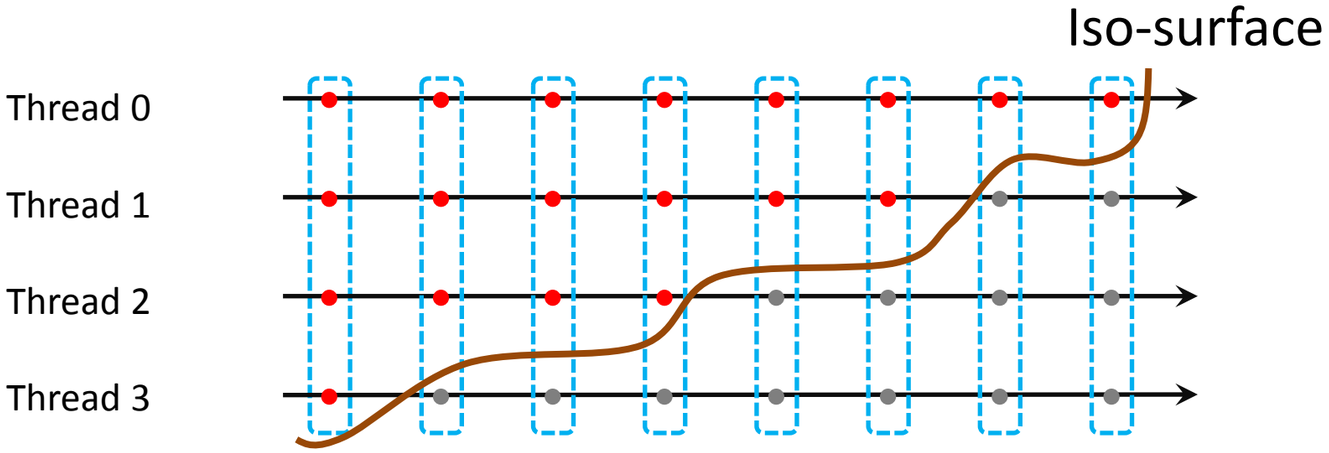
Warp Marching



Load imbalance happens at most once per warp

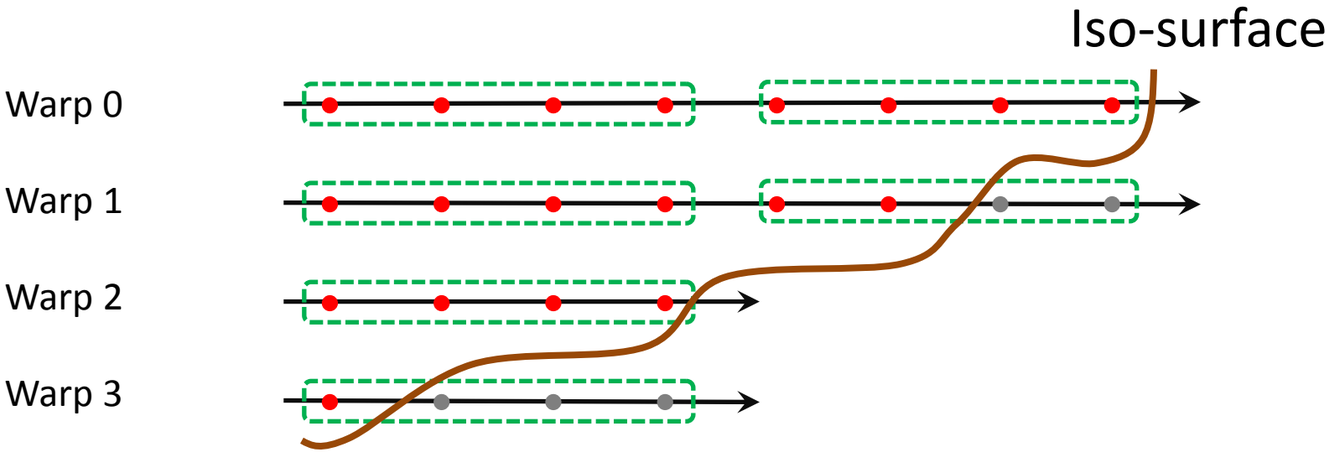
Load Imbalance (Thread-Level)

The Standard



Load imbalance happens many times in a warp

Warp Marching



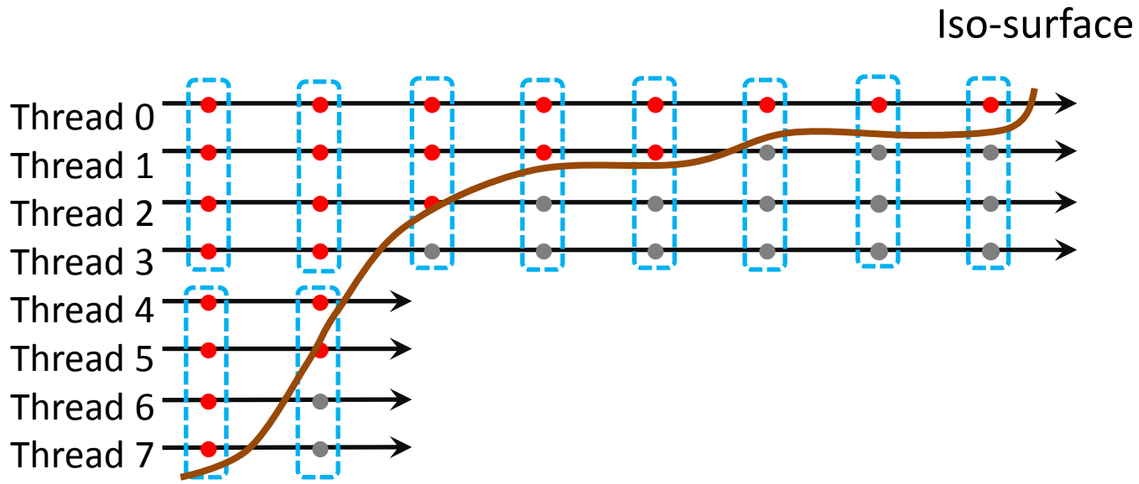
Load imbalance happens at most once per warp

$$\textit{Warp Execution Efficiency} = \frac{\textit{(Thread Instructions Executed)}}{\textit{(Instructions Executed) X (Warp Size)}}$$

(a.k.a. Control Flow Efficiency)

Load Imbalance (SM-Level)

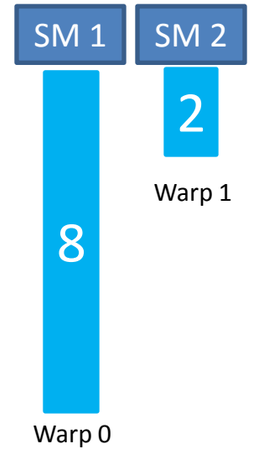
The Standard



A warp covers 4 rays, it has to be active as long as even a single ray is marching.

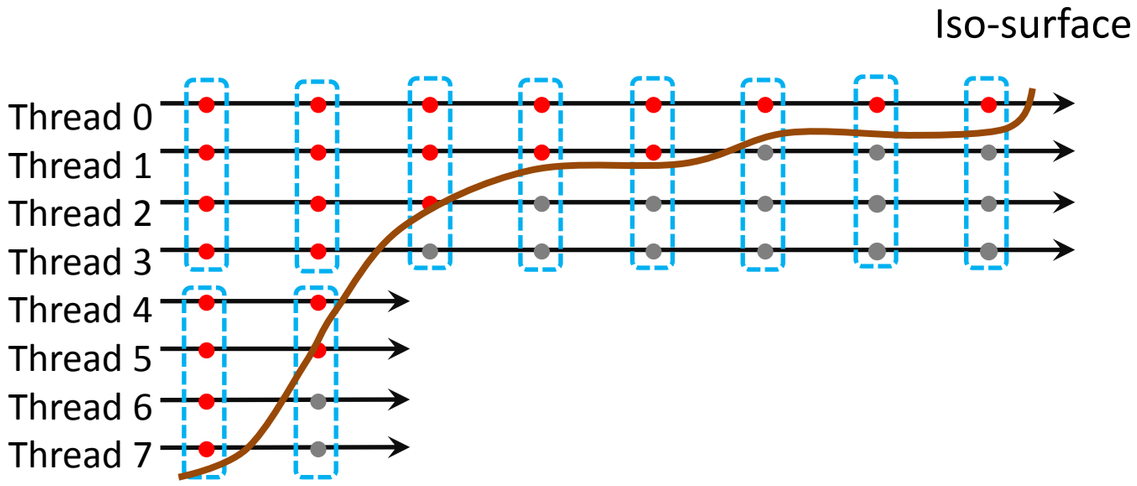
*Lengthy warp
*Large workload variance

8 cycles in total, 50% hardware is idle in 6 cycles



Load Imbalance (SM-Level)

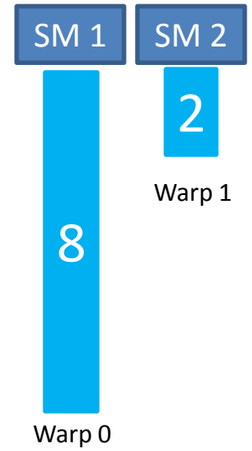
The Standard



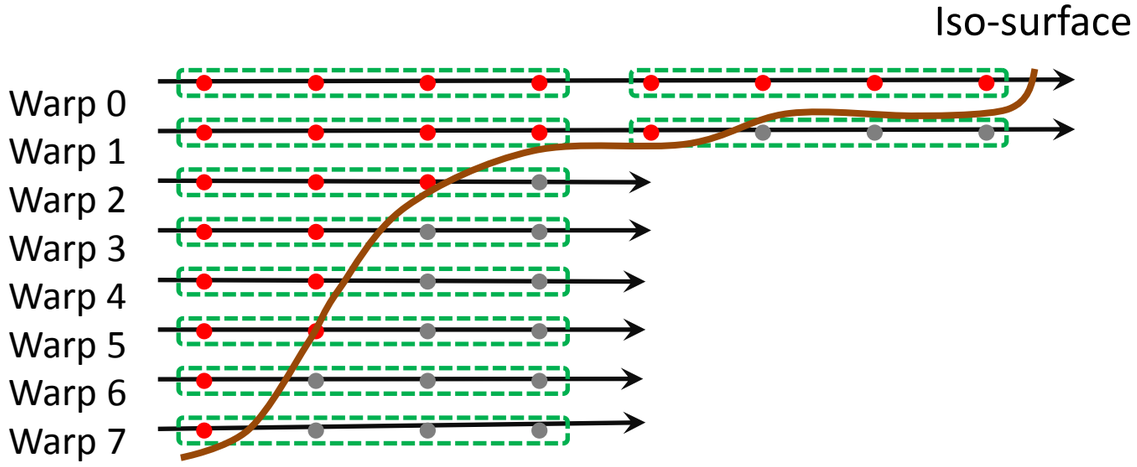
A warp covers 4 rays, it has to be active as long as even a single ray is marching.

*Lengthy warp
*Large workload variance

8 cycles in total, 50% hardware is idle in 6 cycles



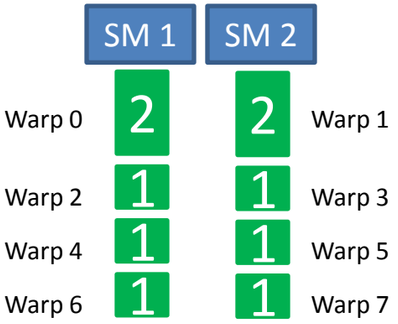
Warp Marching



A warp covers 1 rays, it quits as long as the ray is terminated

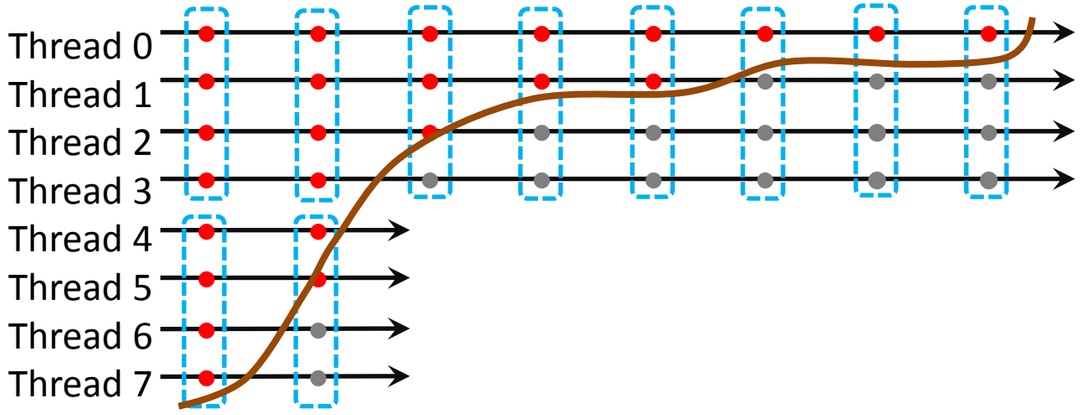
*More warps
*Small workload variance

5 cycles in total, 100% hardware utilization



Load Imbalance (SM-Level)

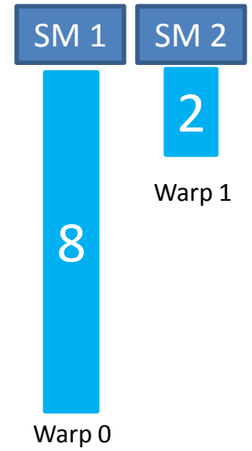
The Standard



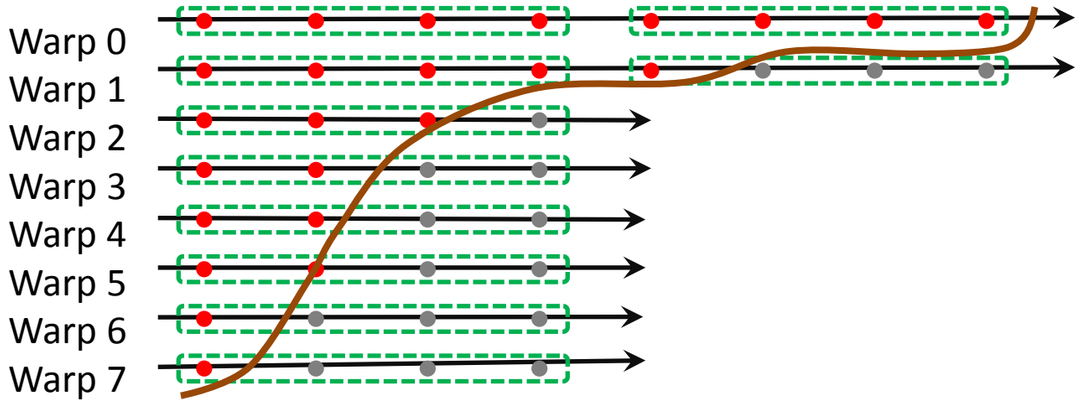
A warp covers 4 rays, it has to be active as long as even a single ray is marching.

*Lengthy warp
*Large workload variance

8 cycles in total, 50% hardware is idle in 6 cycles



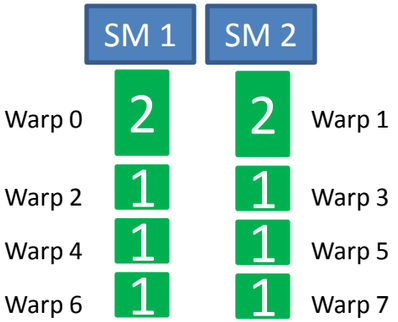
Warp Marching



A warp covers 1 rays, it quits as long as the ray is terminated

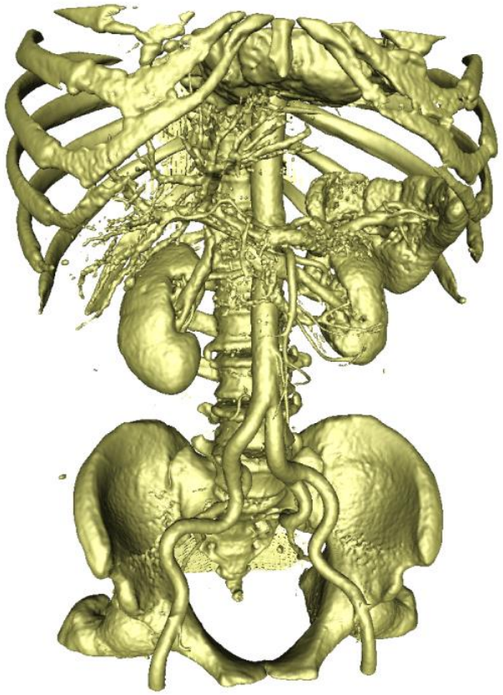
*More warps
*Small workload variance

5 cycles in total, 100% hardware utilization



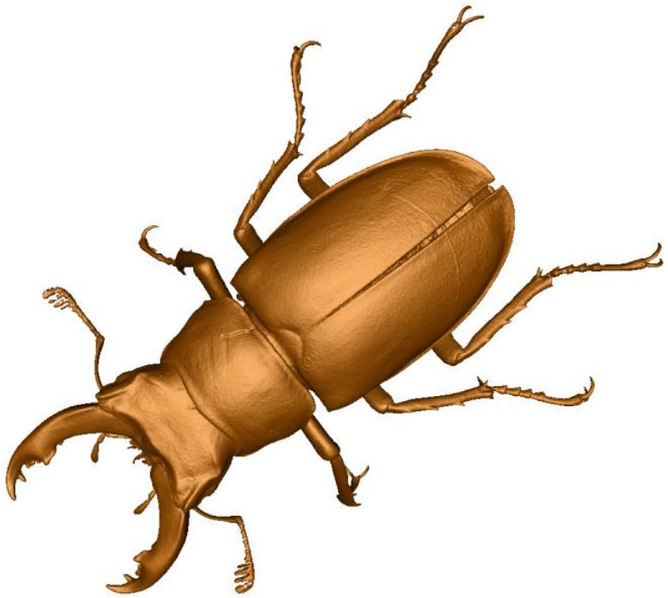
Stdev(Active Cycles of 14 SMs)

Results



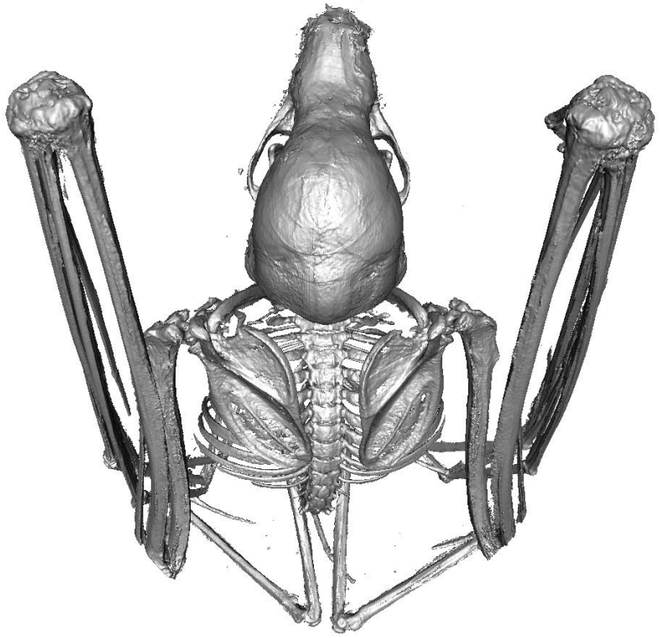
Abdominal

340MB 16-bit
512x512x681



Stag Beetle

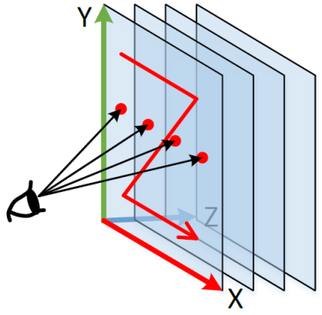
652MB, 16-bit
832x832x494

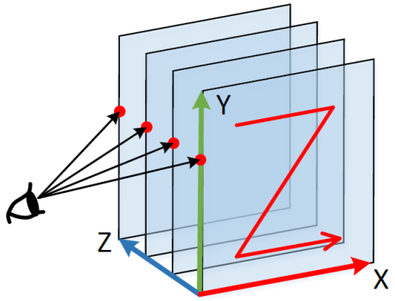


Bat

1.2 GB, 8-bit
906x911x1466

Results

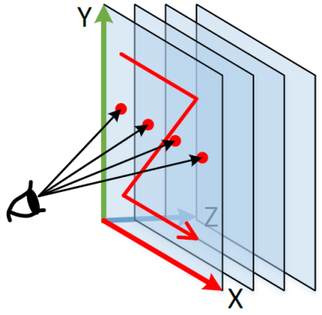


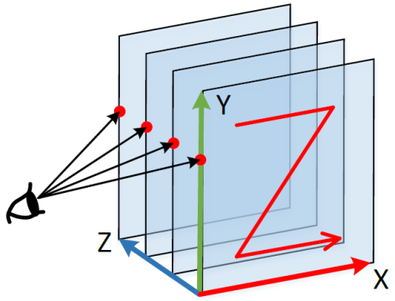


		Facing the *XY* Plane			Facing the *ZY* Plane		
	Data Set	Abdominal	Stag Beetle	Bat	Abdominal	Stag Beetle	Bat
	Dimension	512×512	832×832	906×911	681×512	494×832	1466×911
	Projected Resolution	780×780	871×871	772×776	872×738	603×1014	1023×652
	Ratio	0.66×0.66	0.96×0.96	1.17×1.17	0.78×0.69	0.82×0.82	1.43×1.40
Texture Cache Hit Rate	*Standard*	91.01%	86.64%	87.98%	56.25%	51.62%	29.12%
	Warp Marching	55.44%	66.67%	63.59%	82.87%	88.38%	87.80%
	Speedup	0.61	0.77	0.72	1.47	1.71	3.02
Warp Execution Efficiency	*Standard*	90.09%	94.62%	84.88%	90.42%	91.62%	90.89%
	Warp Marching	92.46%	95.50%	97.09%	92.01%	96.36%	96.75%
	Speedup	1.03	1.01	1.14	1.02	1.05	1.06
Standard Deviation of Active Cycles	*Standard*	5352.90	11024.53	12470.70	5931.84	7965.91	25816.52
	Warp Marching	150.30	103.26	157.53	137.19	143.15	177.05
	Speedup	35.61	106.76	79.16	39.30	55.65	145.81
Frame Per Second	*Standard*	135.52	71.71	48.28	49.81	15.24	3.50
	Warp Marching	68.63	47.67	34.44	106.28	62.24	46.01
	Speedup	0.51	0.66	0.71	2.13	4.08	13.15

Table 1: Comparison of the Warp Marching and the Standard algorithm. Results are collected in perspective projection with a 45° field of view.

Results

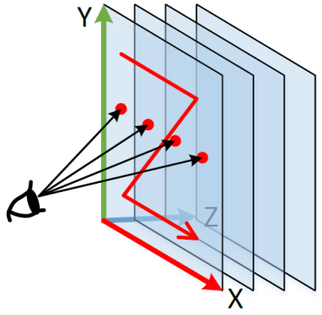


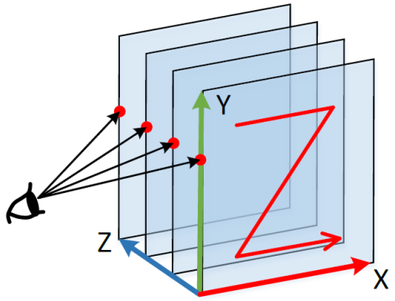


		Facing the *XY* Plane			Facing the *ZY* Plane		
	Data Set	Abdominal	Stag Beetle	Bat	Abdominal	Stag Beetle	Bat
	Dimension	512×512	832×832	906×911	681×512	494×832	1466×911
	Projected Resolution	780×780	871×871	772×776	872×738	603×1014	1023×652
	Ratio	0.66×0.66	0.96×0.96	1.17×1.17	0.78×0.69	0.82×0.82	1.43×1.40
Texture Cache Hit Rate	*Standard*	91.01%	86.64%	87.98%	56.25%	51.62%	29.12%
	Warp Marching	55.44%	66.67%	63.59%	82.87%	88.38%	87.80%
	Speedup	0.61	0.77	0.72	1.47	1.71	3.02
Warp Execution Efficiency	*Standard*	90.09%	94.62%	84.88%	90.42%	91.62%	90.89%
	Warp Marching	92.46%	95.50%	97.09%	92.01%	96.36%	96.75%
	Speedup	1.03	1.01	1.14	1.02	1.05	1.06
Standard Deviation of Active Cycles	*Standard*	5352.90	11024.53	12470.70	5931.84	7965.91	25816.52
	Warp Marching	150.30	103.26	157.53	137.19	143.15	177.05
	Speedup	35.61	106.76	79.16	39.30	55.65	145.81
Frame Per Second	*Standard*	135.52	71.71	48.28	49.81	15.24	3.50
	Warp Marching	68.63	47.67	34.44	106.28	62.24	46.01
	Speedup	0.51	0.66	0.71	2.13	4.08	13.15

Table 1: Comparison of the Warp Marching and the Standard algorithm. Results are collected in perspective projection with a 45° field of view.

Results

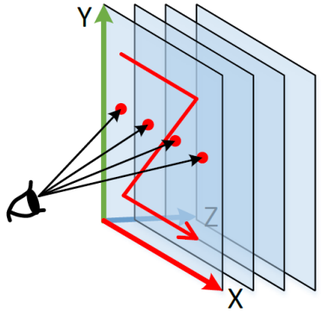


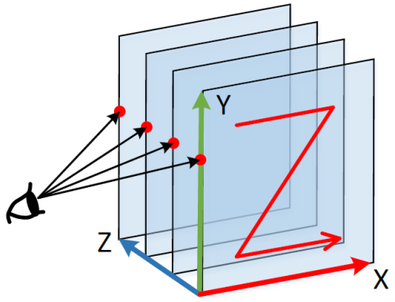


		Facing the *XY* Plane			Facing the *ZY* Plane		
	Data Set	Abdominal	Stag Beetle	Bat	Abdominal	Stag Beetle	Bat
	Dimension	512×512	832×832	906×911	681×512	494×832	1466×911
	Projected Resolution	780×780	871×871	772×776	872×738	603×1014	1023×652
	Ratio	0.66×0.66	0.96×0.96	1.17×1.17	0.78×0.69	0.82×0.82	1.43×1.40
Texture Cache Hit Rate	*Standard*	91.01%	86.64%	87.98%	56.25%	51.62%	29.12%
	Warp Marching	55.44%	66.67%	63.59%	82.87%	88.38%	87.80%
	Speedup	0.61	0.77	0.72	1.47	1.71	3.02
Warp Execution Efficiency	*Standard*	90.09%	94.62%	84.88%	90.42%	91.62%	90.89%
	Warp Marching	92.46%	95.50%	97.09%	92.01%	96.36%	96.75%
	Speedup	1.03	1.01	1.14	1.02	1.05	1.06
Standard Deviation of Active Cycles	*Standard*	5352.90	11024.53	12470.70	5931.84	7965.91	25816.52
	Warp Marching	150.30	103.26	157.53	137.19	143.15	177.05
	Speedup	35.61	106.76	79.16	39.30	55.65	145.81
Frame Per Second	*Standard*	135.52	71.71	48.28	49.81	15.24	3.50
	Warp Marching	68.63	47.67	34.44	106.28	62.24	46.01
	Speedup	0.51	0.66	0.71	2.13	4.08	13.15

Table 1: Comparison of the Warp Marching and the Standard algorithm. Results are collected in perspective projection with a 45° field of view.

Results





		Facing the *XY* Plane			Facing the *ZY* Plane		
	Data Set	Abdominal	Stag Beetle	Bat	Abdominal	Stag Beetle	Bat
	Dimension	512×512	832×832	906×911	681×512	494×832	1466×911
	Projected Resolution	780×780	871×871	772×776	872×738	603×1014	1023×652
	Ratio	0.66×0.66	0.96×0.96	1.17×1.17	0.78×0.69	0.82×0.82	1.43×1.40
Texture Cache Hit Rate	*Standard*	91.01%	86.64%	87.98%	56.25%	51.62%	29.12%
	Warp Marching	55.44%	66.67%	63.59%	82.87%	88.38%	87.80%
	Speedup	0.61	0.77	0.72	1.47	1.71	3.02
Warp Execution Efficiency	*Standard*	90.09%	94.62%	84.88%	90.42%	91.62%	90.89%
	Warp Marching	92.46%	95.50%	97.09%	92.01%	96.36%	96.75%
	Speedup	1.03	1.01	1.14	1.02	1.05	1.06
Standard Deviation of Active Cycles	*Standard*	5352.90	11024.53	12470.70	5931.84	7965.91	25816.52
	Warp Marching	150.30	103.26	157.53	137.19	143.15	177.05
	Speedup	35.61	106.76	79.16	39.30	55.65	145.81
Frame Per Second	*Standard*	135.52	71.71	48.28	49.81	15.24	3.50
	Warp Marching	68.63	47.67	34.44	106.28	62.24	46.01
	Speedup	0.51	0.66	0.71	2.13	4.08	13.15

Table 1: Comparison of the Warp Marching and the Standard algorithm. Results are collected in perspective projection with a 45° field of view.

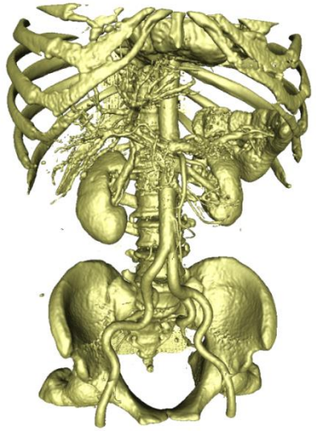
Conclusion

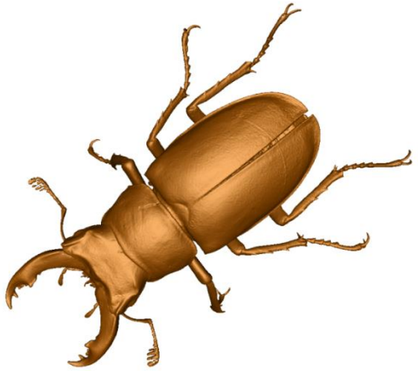
- Cache Performance: two computation-to-core mapping strategies show complementary patterns in different viewing directions.

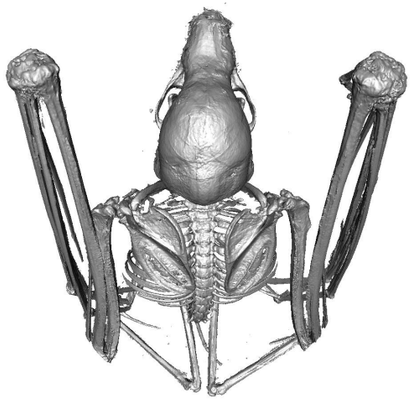
- Load Balancing: Warp Marching always has a smaller workload variance (among threads/warps)

Computation-to-Core Mapping Strategies for Iso-Surface Volume Rendering on GPUs

Thank you







VirginiaTech
Invent the Future

NVIDIA

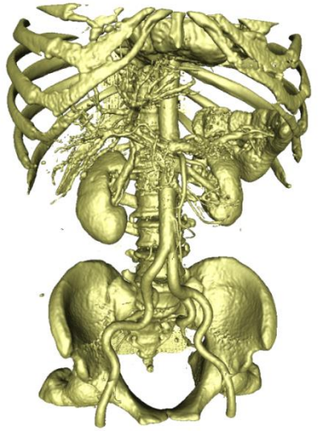
Questions?

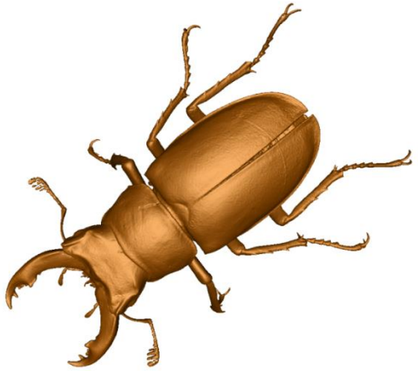
Computation-to-Core Mapping Strategies for Iso-Surface Volume Rendering on GPUs

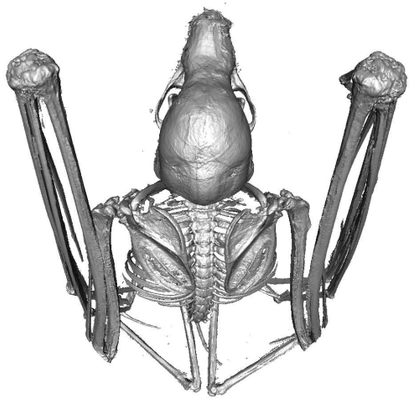
Junpeng Wang
junpeng@vt.edu

Fei Yang
feiy@nvidia.com

Yong Cao
yongcao@vt.edu







VirginiaTech
Invent the Future

NVIDIA