SIBGRAPI 2011

Beam casting implicit surfaces on the GPU with interval arithmetic

Francisco Ganacim
Luiz Henrique de Figueiredo
Diego Nehab

Maceió – Alagoas - Brazil

VisgrafLab

impa

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico
60 ANOS

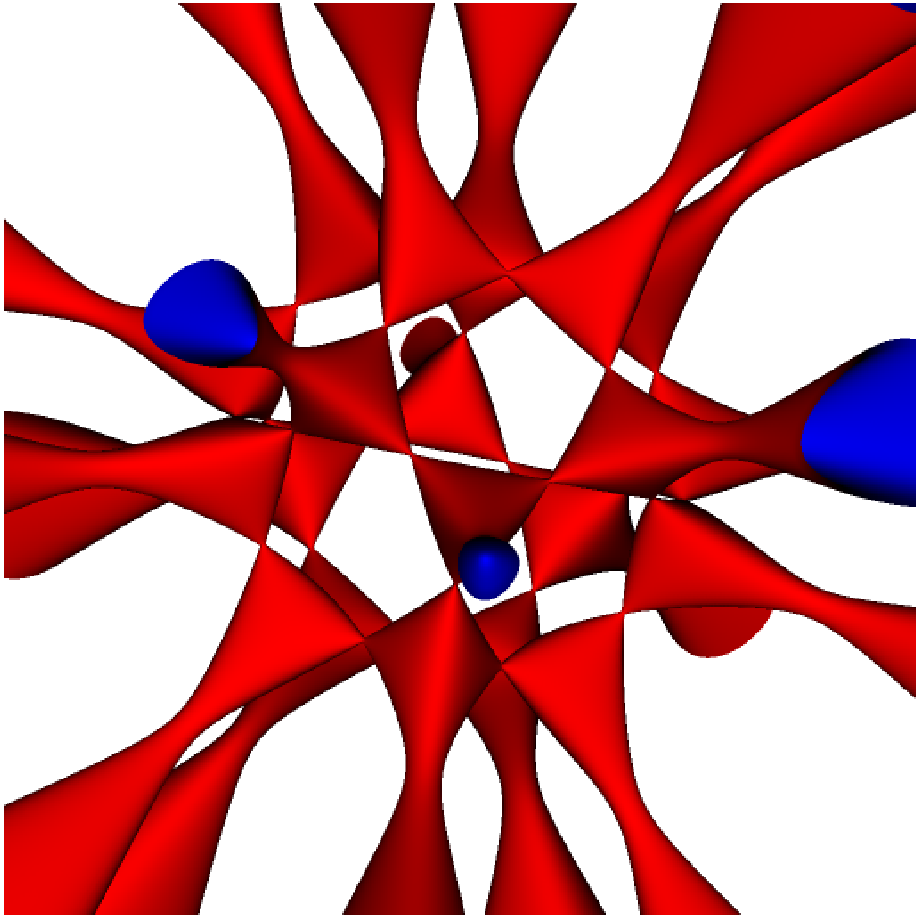
Introduction

Related work

$$f : \mathbf{R}^3 \rightarrow \mathbf{R}$$

$$S = f^{-1}(0)$$

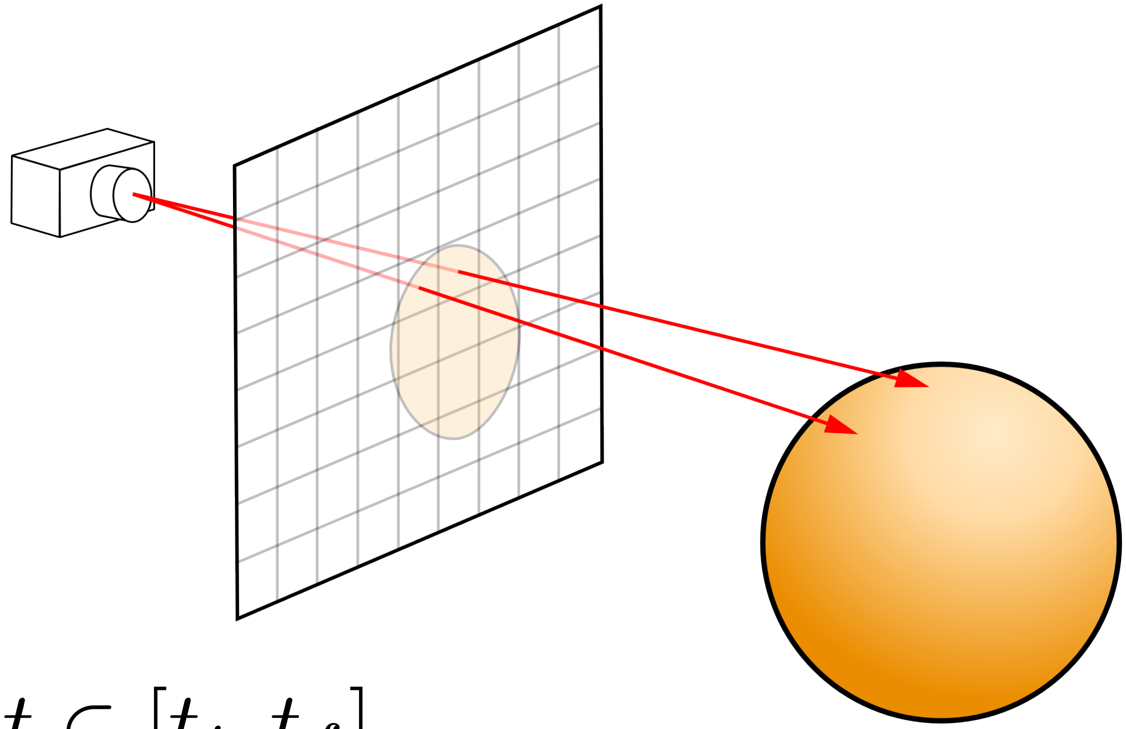
$$n_p = \frac{\nabla f(p)}{\|\nabla f(p)\|}$$



Barth: $4(c^2x^2 - y^2)(c^2y^2 - z^2)(c^2z^2 - x^2) - (1 + 2c)(x^2 + y^2 + z^2 - 1)^2 = 0$

Introduction

Ray casting

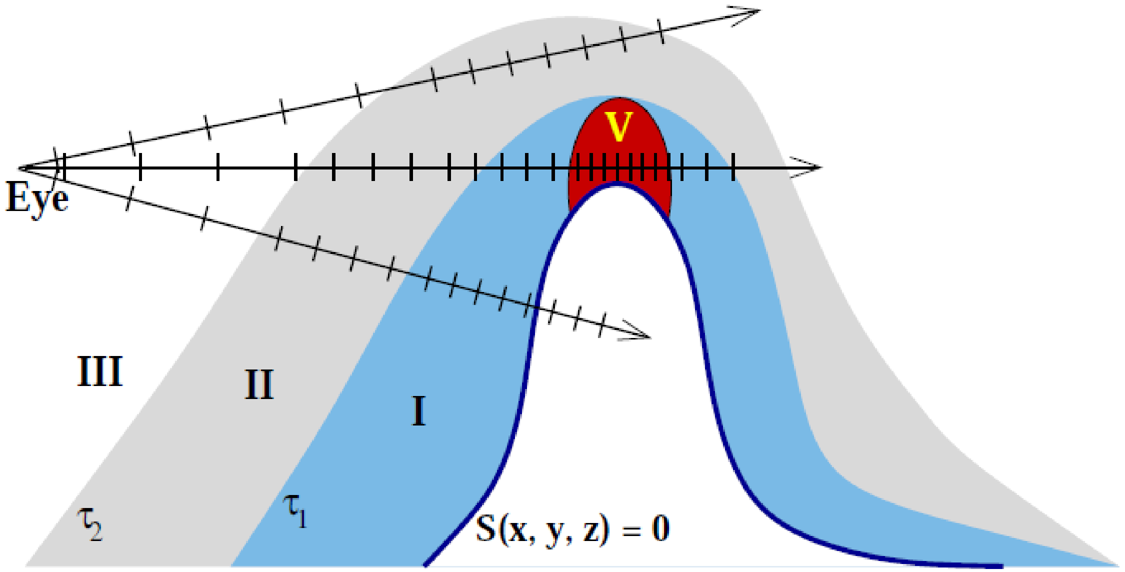


$$r(t) = p + tw, \; t \in [t_i, t_f]$$

$$g(t) := f(r(t)) = 0$$

Introduction

Related work

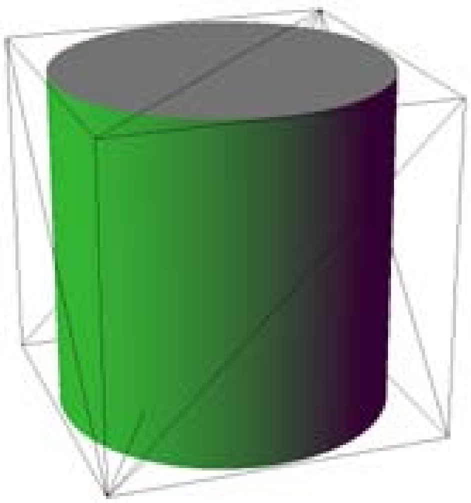


- Root finding by comparing consecutive samples along the ray
- GPU with GLSL

Singh et al. - 2009

Introduction

Related work



- Analytic solution finder for low-degree polynomial surfaces inside tetrahedrons
- GPU with HLSL

Loop & Blinn - 2006

Introduction

Interval arithmetic

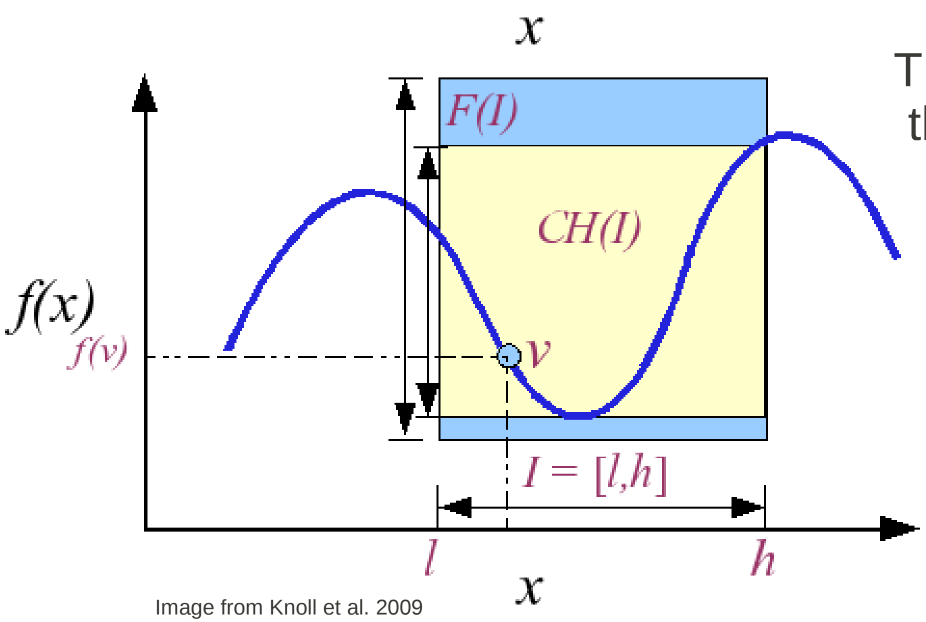


Image from Knoll et al. 2009

The interval extension gives bounds for the range of a function on an interval

$$f(I) \subset F(I)$$

Introduction

Interval arithmetic

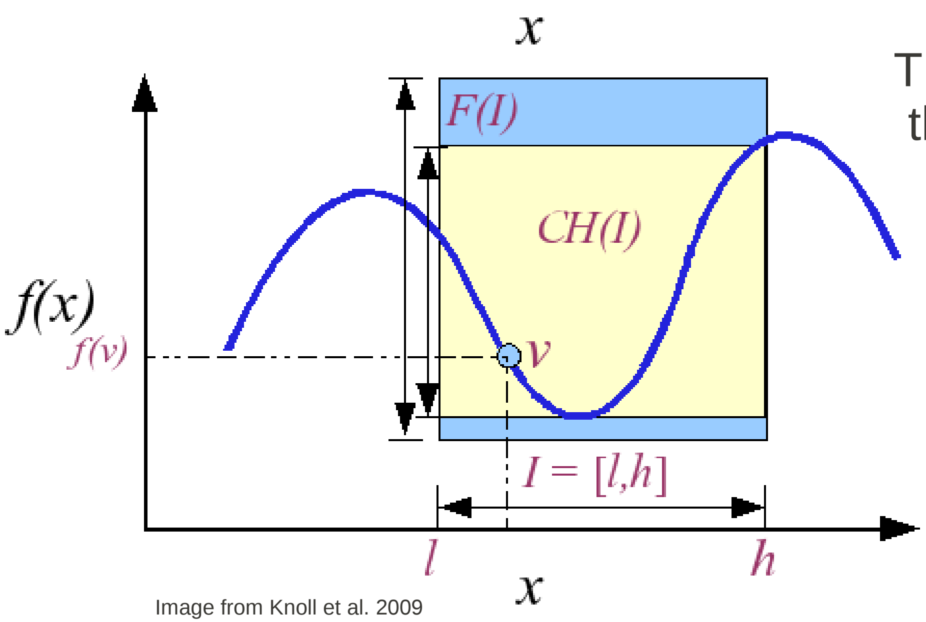


Image from Knoll et al. 2009

The interval extension gives bounds for the range of a function on an interval

$$f(I) \subset F(I)$$

root containment criteria

$$0 \notin F(I) \implies 0 \notin f(I)$$

Introduction

Related work

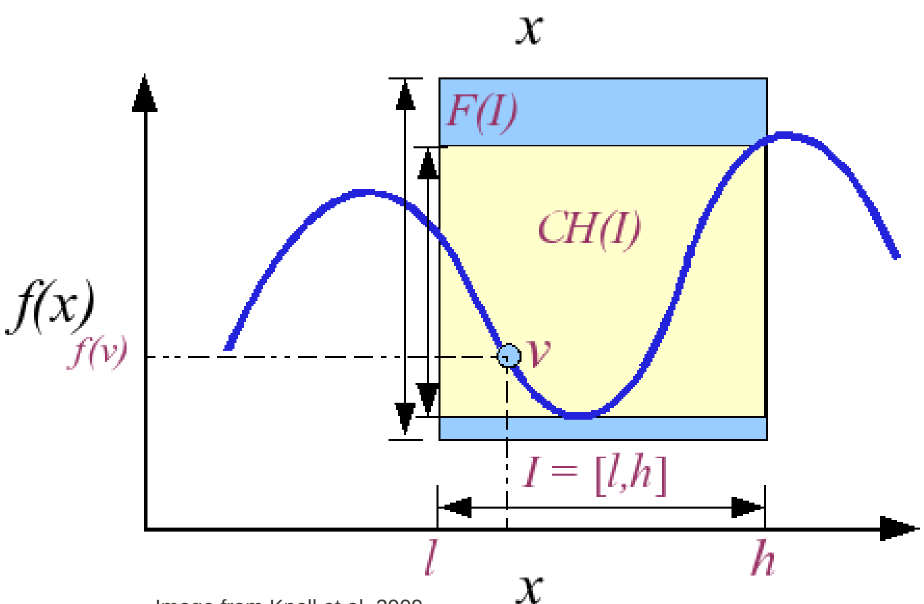
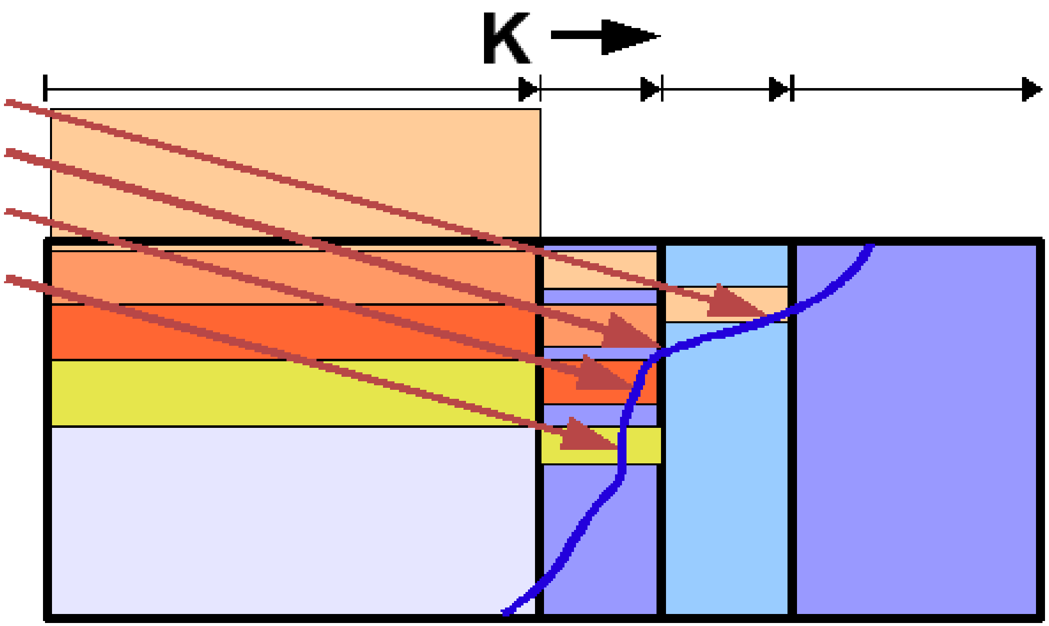


Image from Knoll et al. 2009

- Mitchell - 1990
- Cusatis et al. - 1999

Introduction

Related work

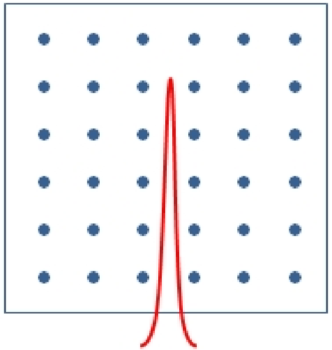


- Real-time ray casting on the CPU using SSE instructions
- Exploits ray coherence

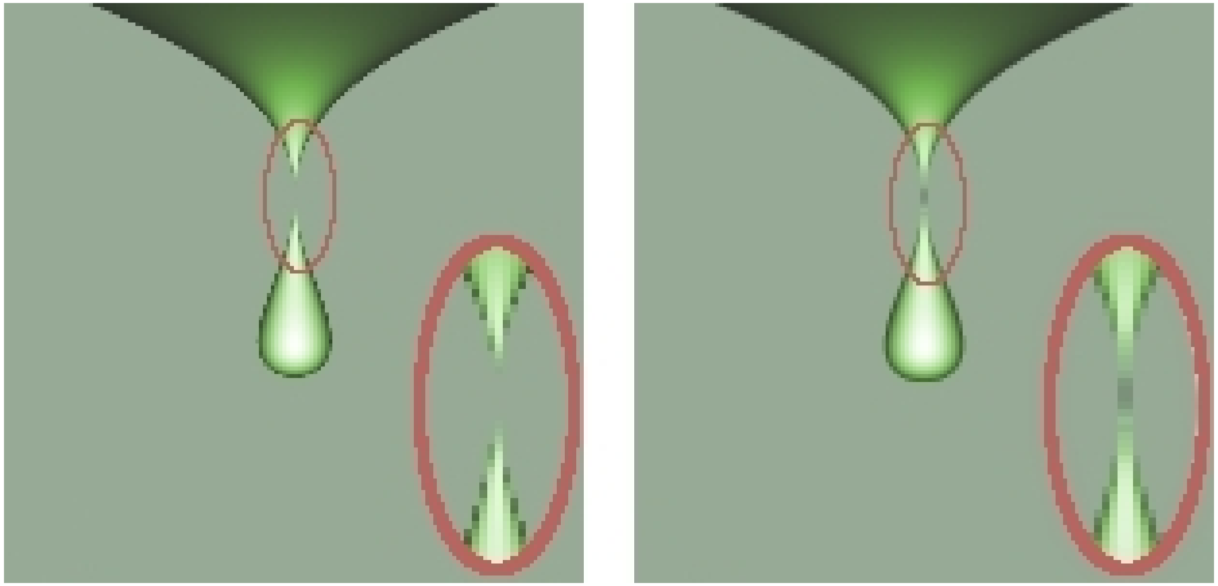
Knoll et al. - 2009

Introduction

Limitations of point sampling



- A low sampling rate is necessary to achieve real-time frame rates
- Thin features may be missed

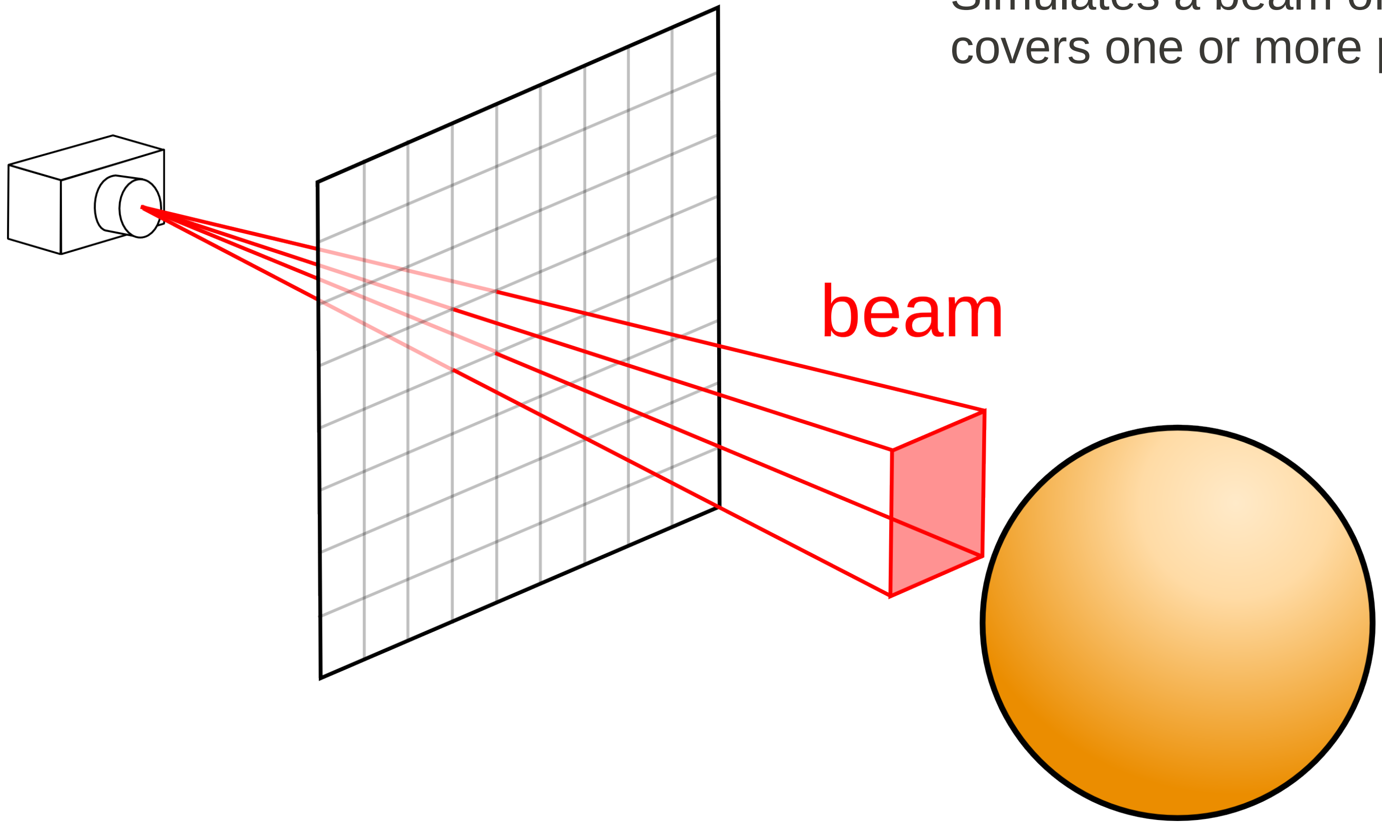


Images from Flórez, 2008

Introduction

Beam casting

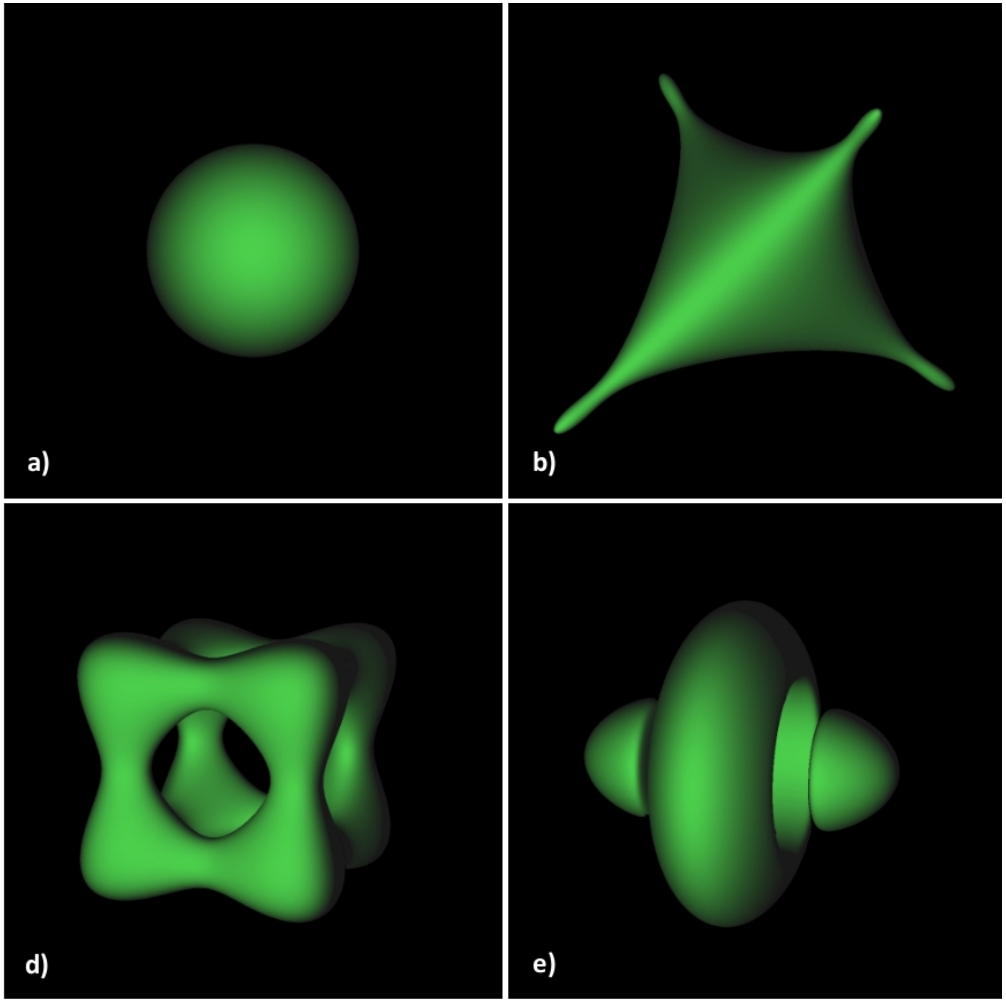
- Simulates a beam of rays that covers one or more pixels



Introduction

Related work

- Beam casting with space subdivision on the CPU



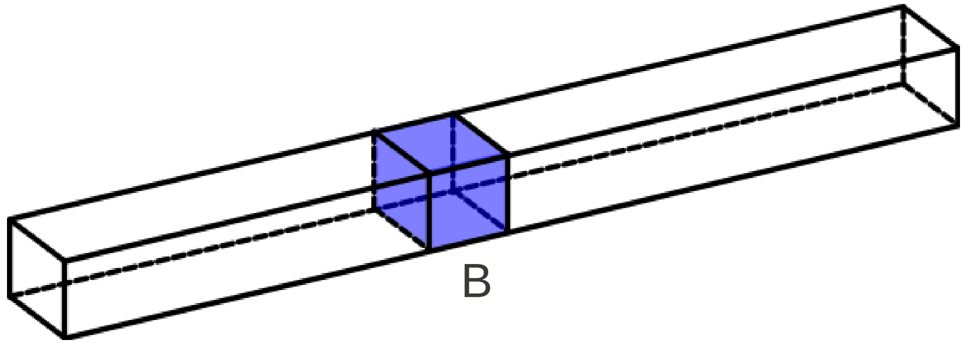
Flórez - 2008
Flórez et al. - 2009

Our method

- Beam casting
- GPU - CUDA
- Space subdivision
- Anti-aliasing

Our method

root containment test



$$f(B) \subset F(B)$$

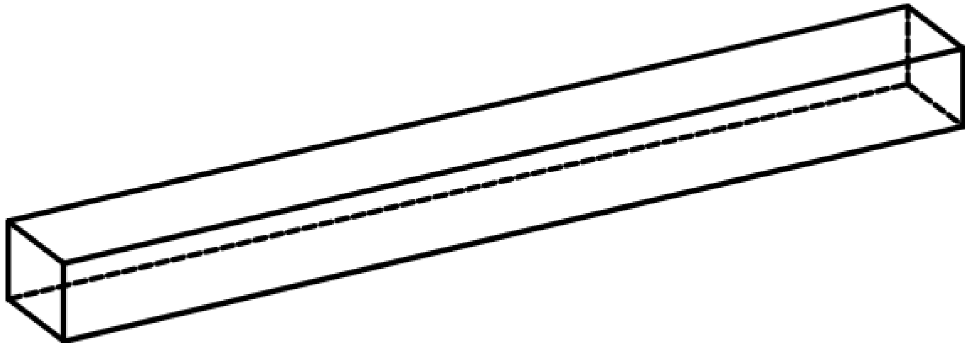
root containment criteria

$$0 \notin F(B) \implies 0 \notin f(B)$$

Our method

Beam bisection

- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam
- Finish when a “small” block is found

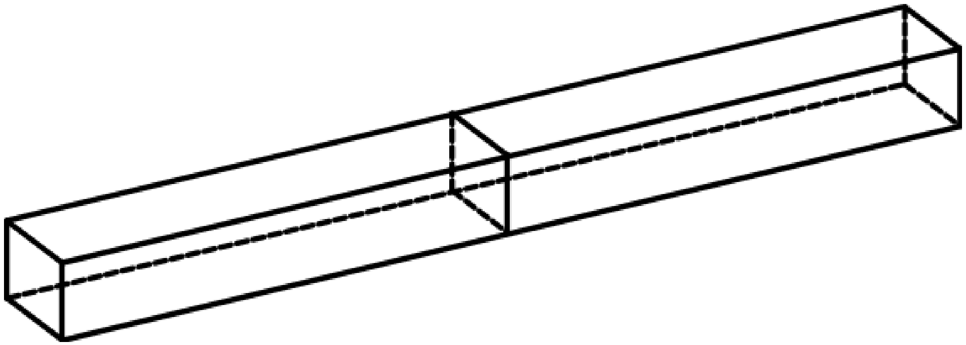


Our method

Beam bisection

- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam

- Finish when a “small” block is found

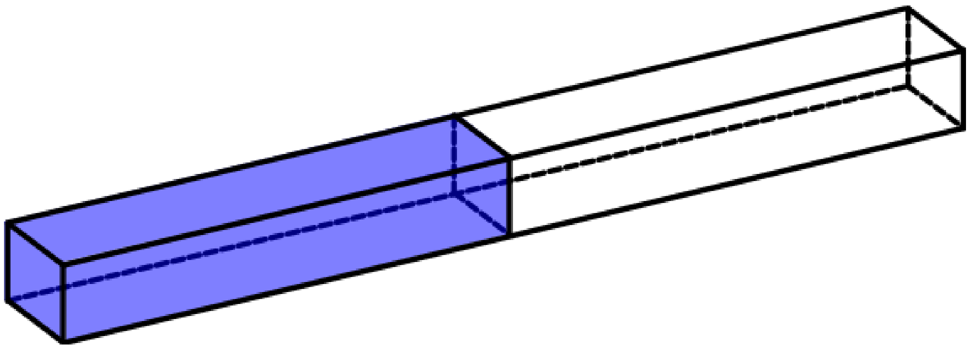


Our method

Beam bisection

- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam

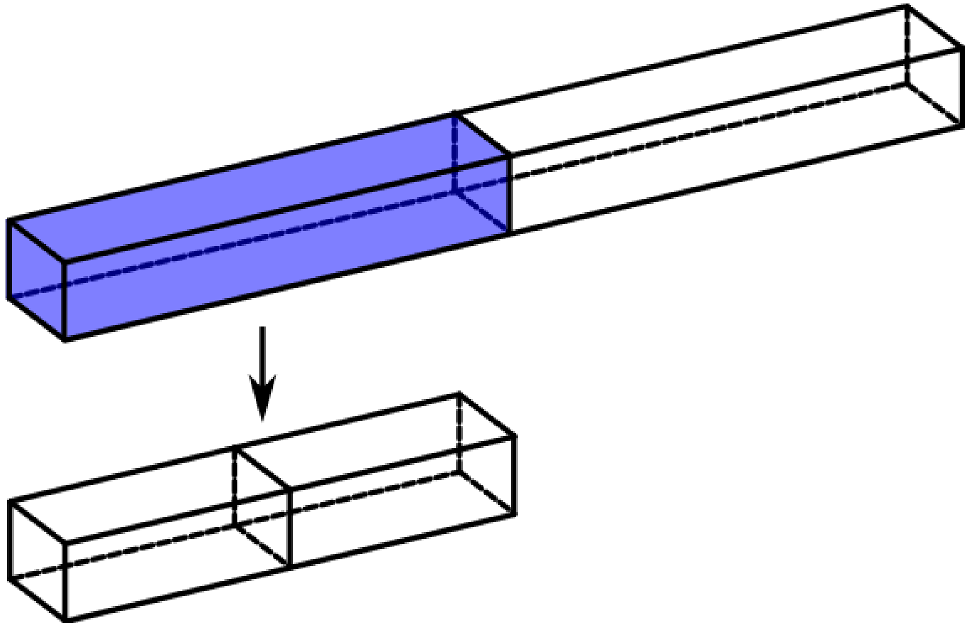
- Finish when a “small” block is found



Our method

Beam bisection

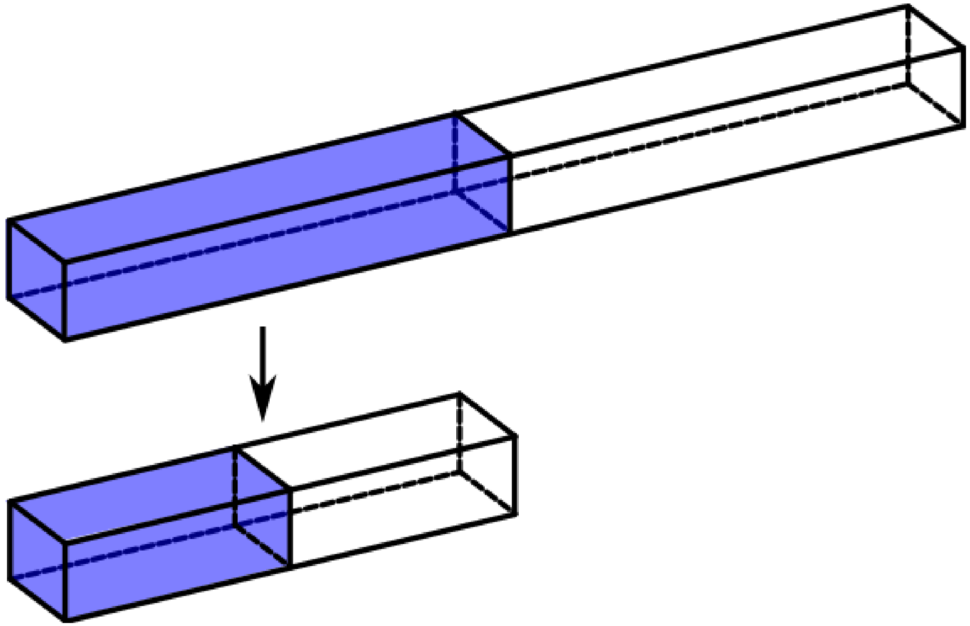
- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam
- Finish when a “small” block is found



Our method

Beam bisection

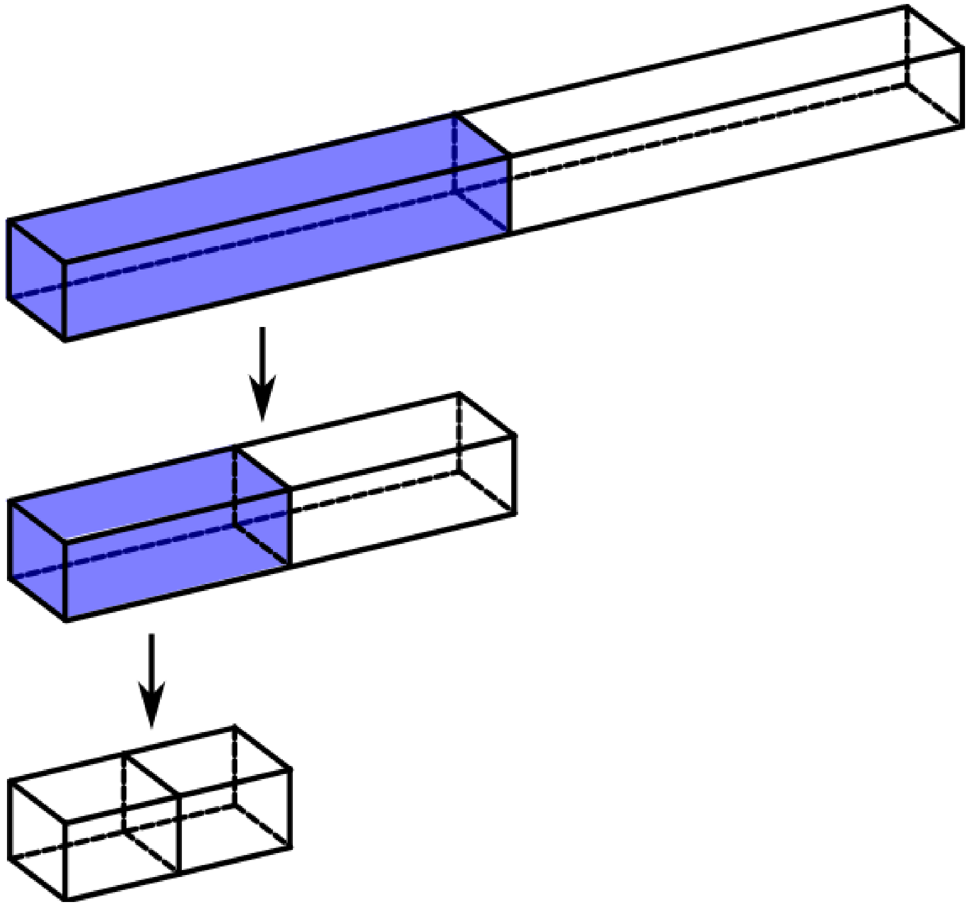
- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam
- Finish when a “small” block is found



Our method

Beam bisection

- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam
- Finish when a “small” block is found

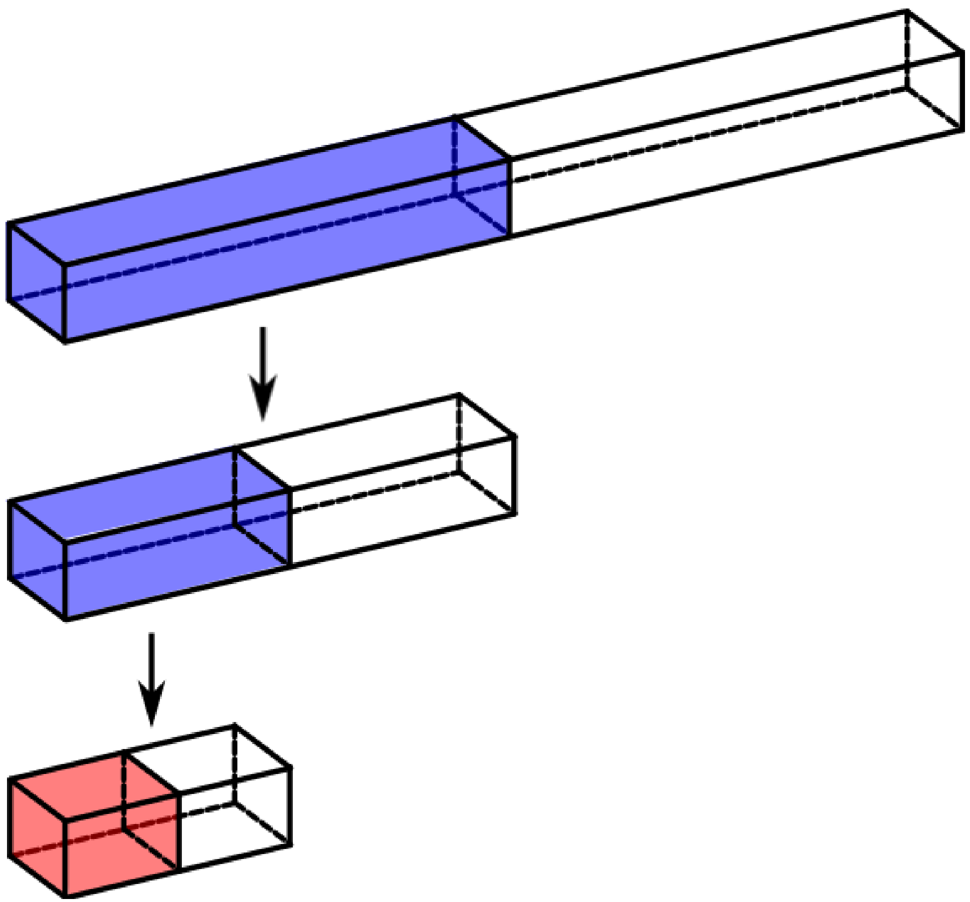


Our method

Beam bisection

- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam

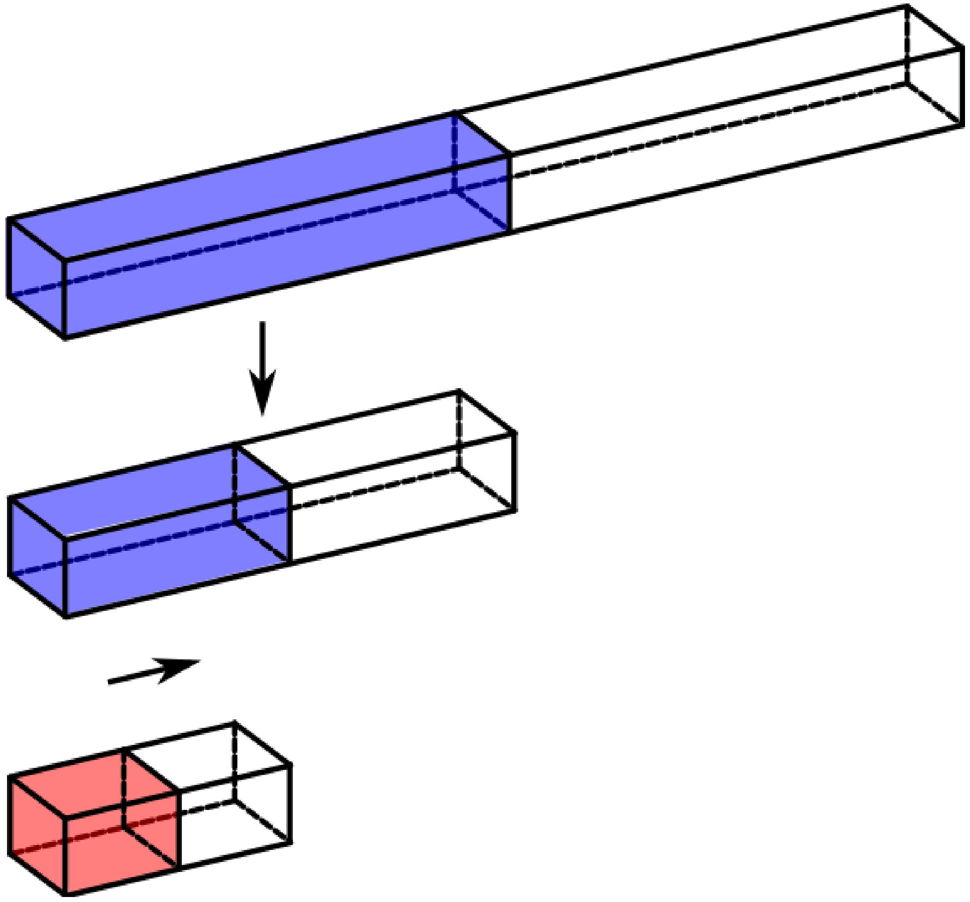
- Finish when a “small” block is found



Our method

Beam bisection

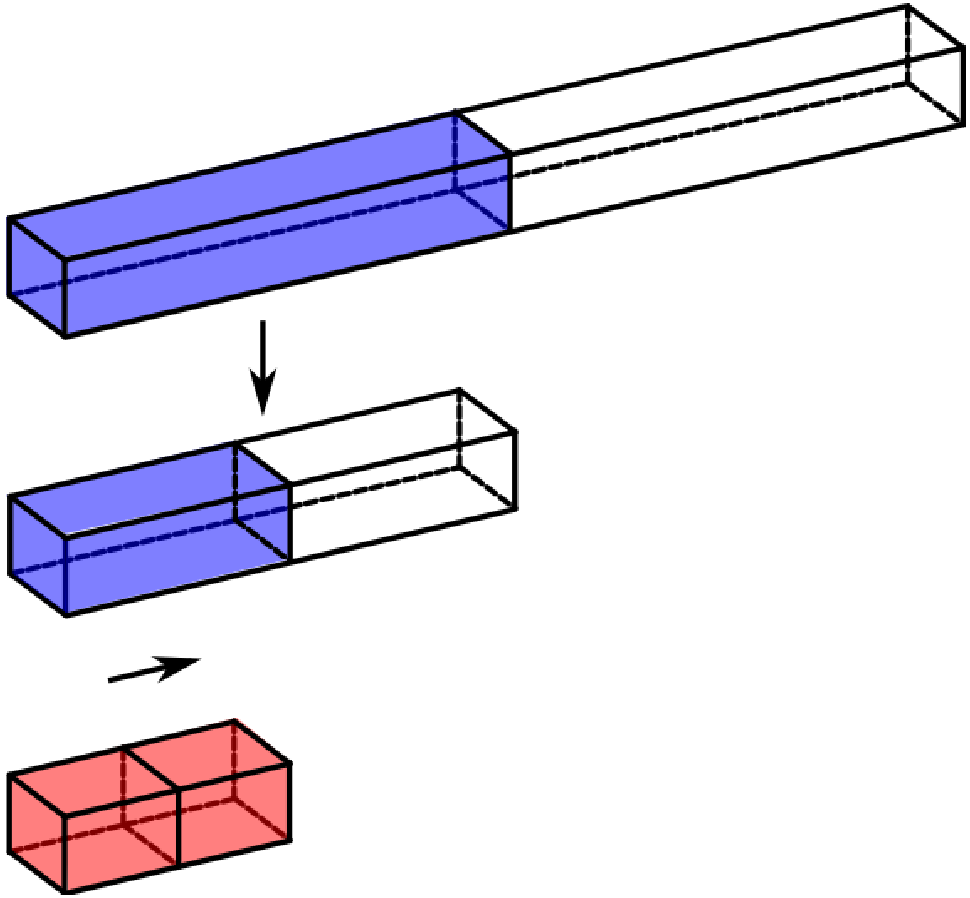
- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam
- Finish when a “small” block is found



Our method

Beam bisection

- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam
- Finish when a “small” block is found

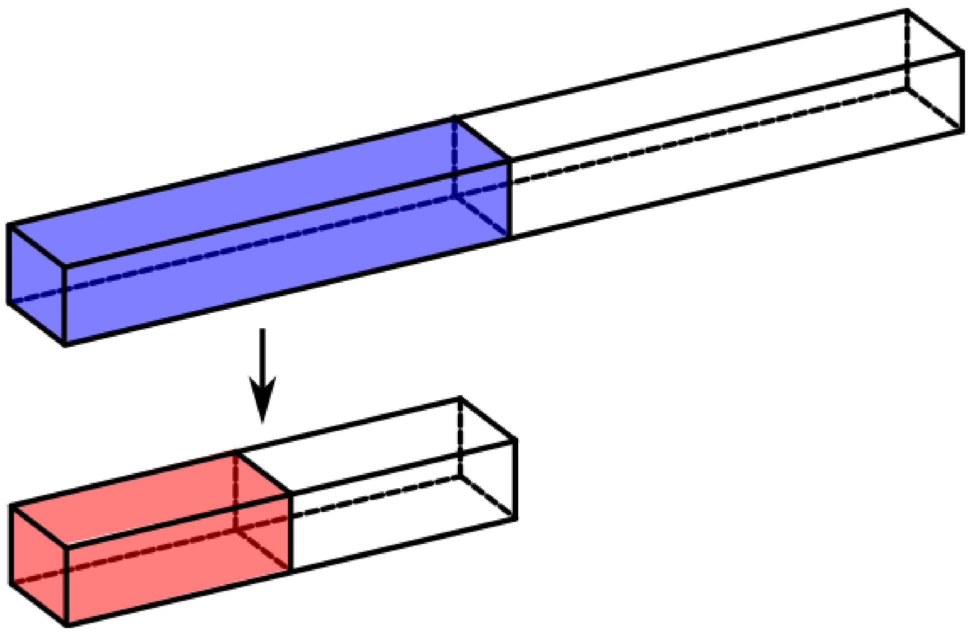


Our method

Beam bisection

- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam

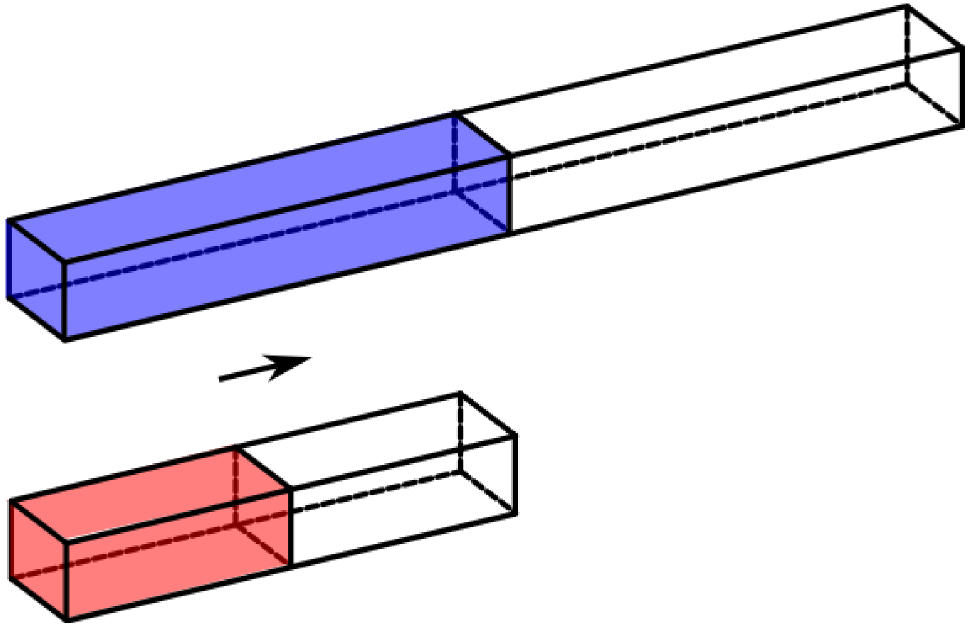
- Finish when a “small” block is found



Our method

Beam bisection

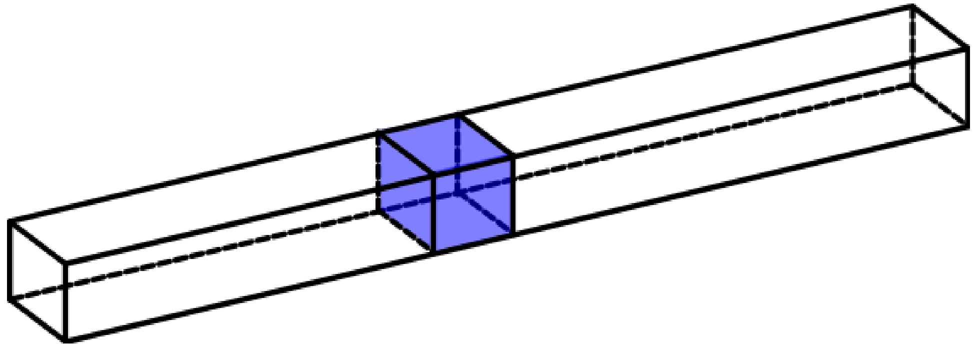
- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam
- Finish when a “small” block is found



Our method

Beam bisection

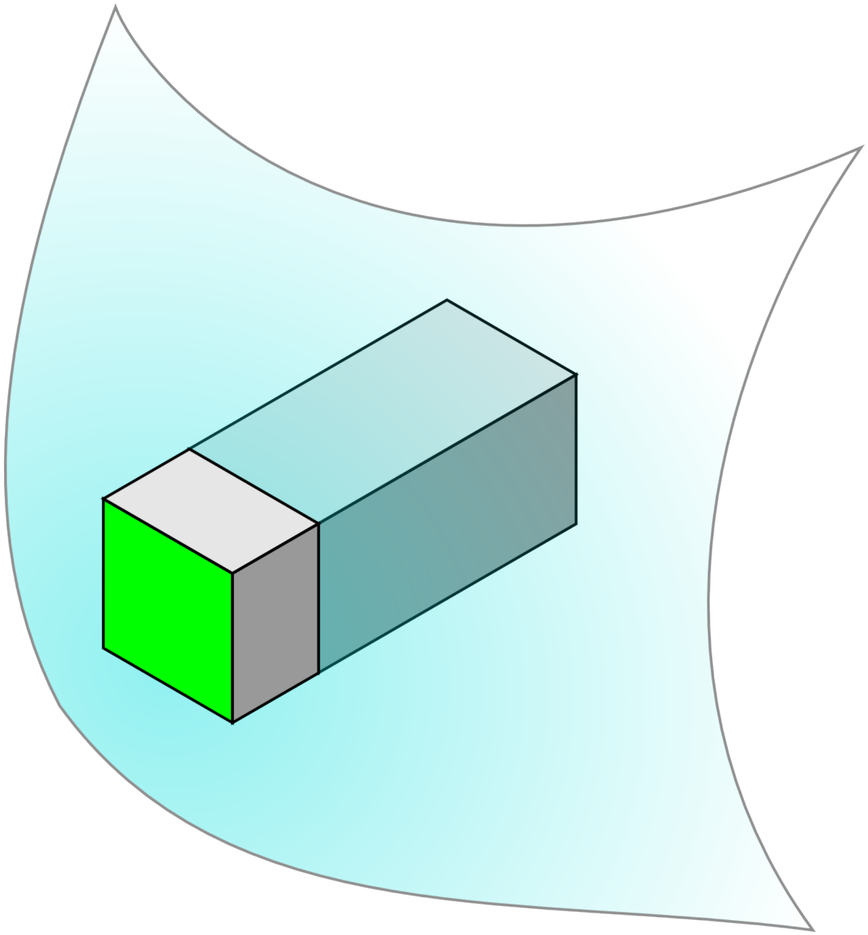
- Depth-first search on a binary tree, where the nodes are generated by bisecting the input beam
- Finish when a "small" block is found



Our method

Bisection precision limit

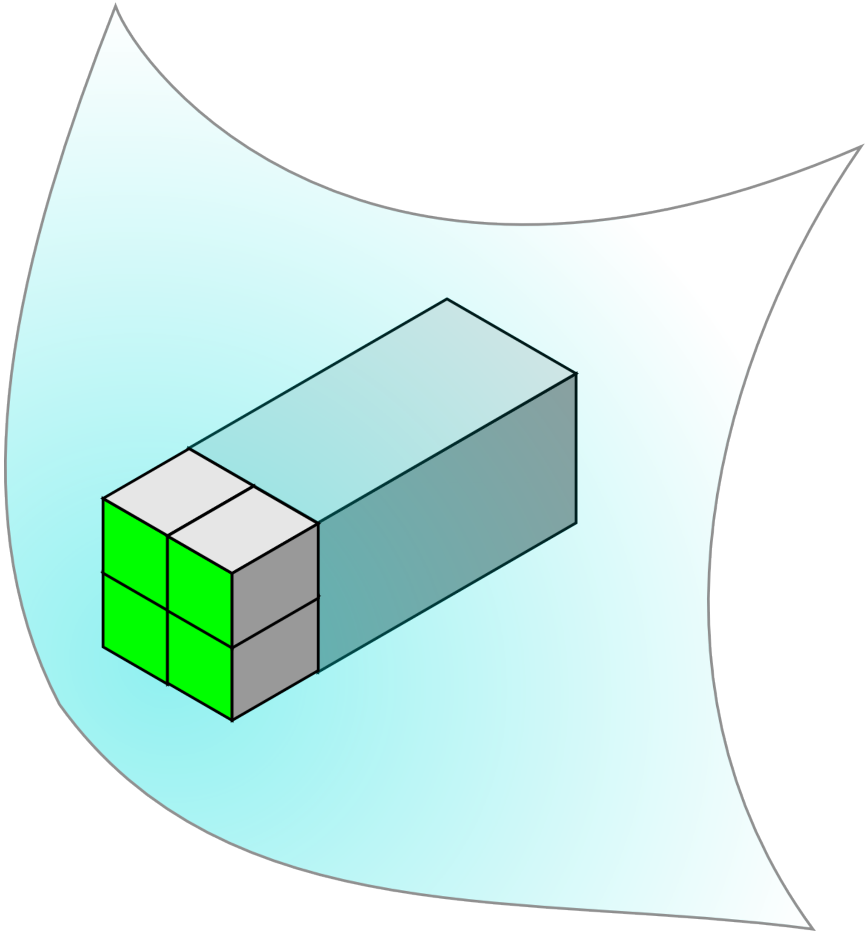
- The geometry of the beam imposes a limit to the precision



Our method

Bisection precision limit

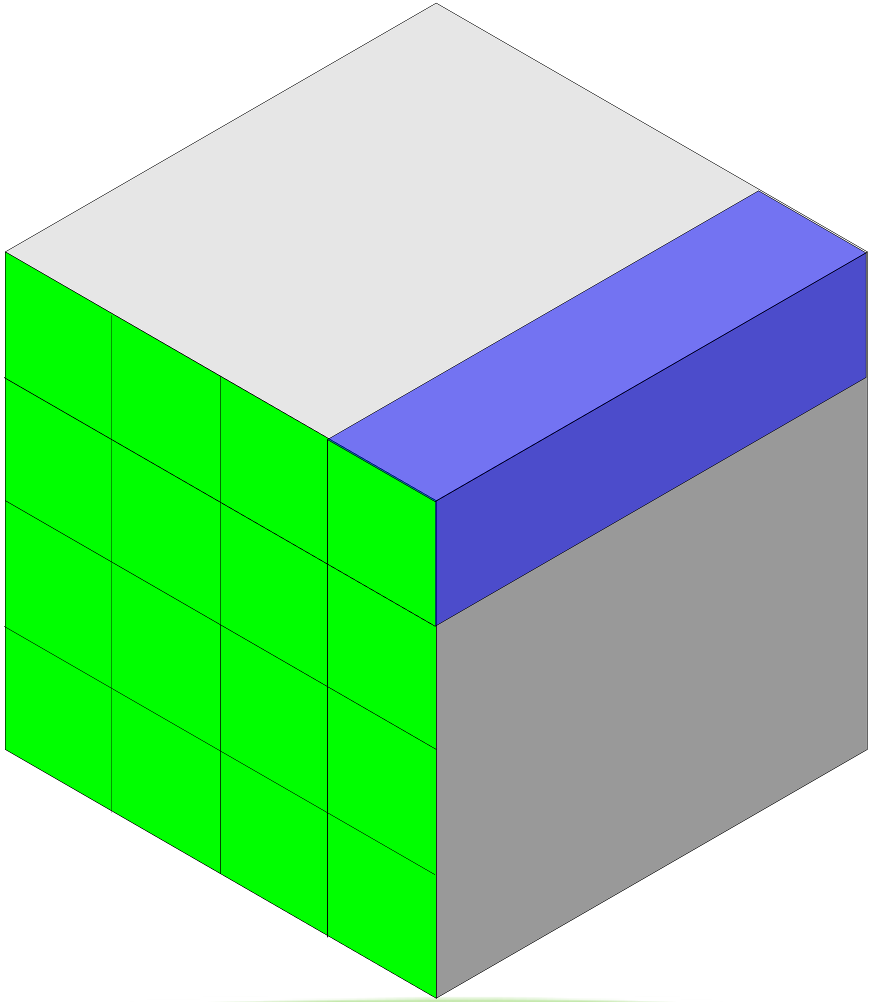
- The geometry of the beam imposes a limit to the precision
- We need to subdivide the beam to achieve more precision



Our method

Spatial subdivision

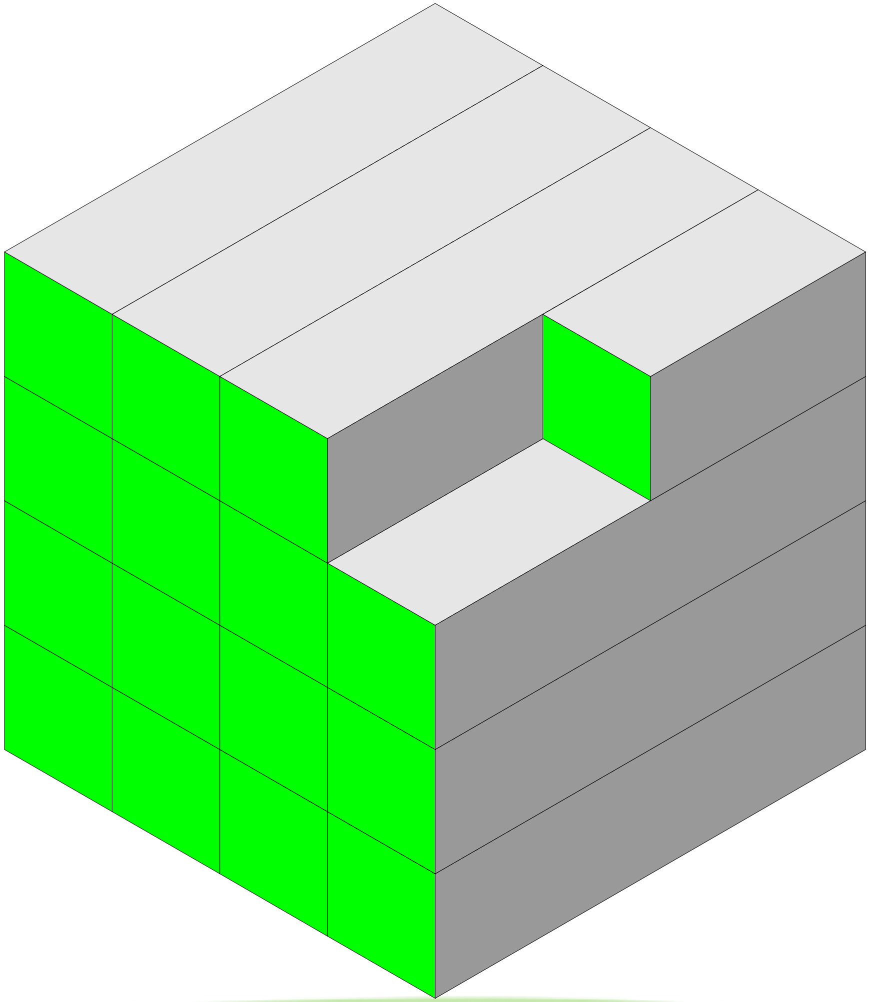
1. Large beams are cast



Our method

Spatial subdivision

1. Large beams are cast

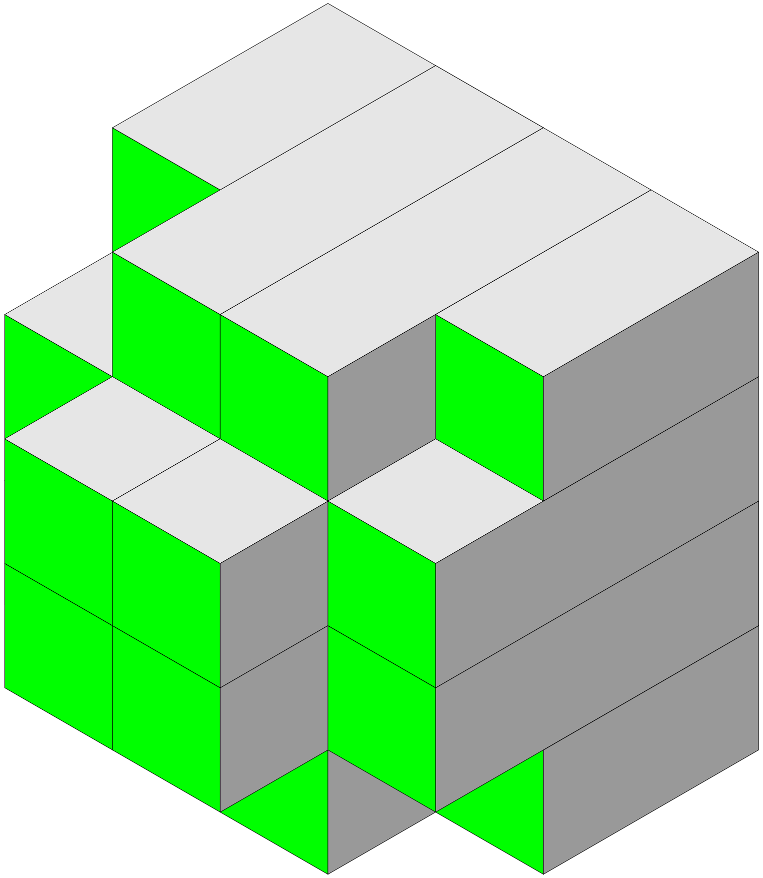


Our method

Spatial subdivision

1. Large beams are cast

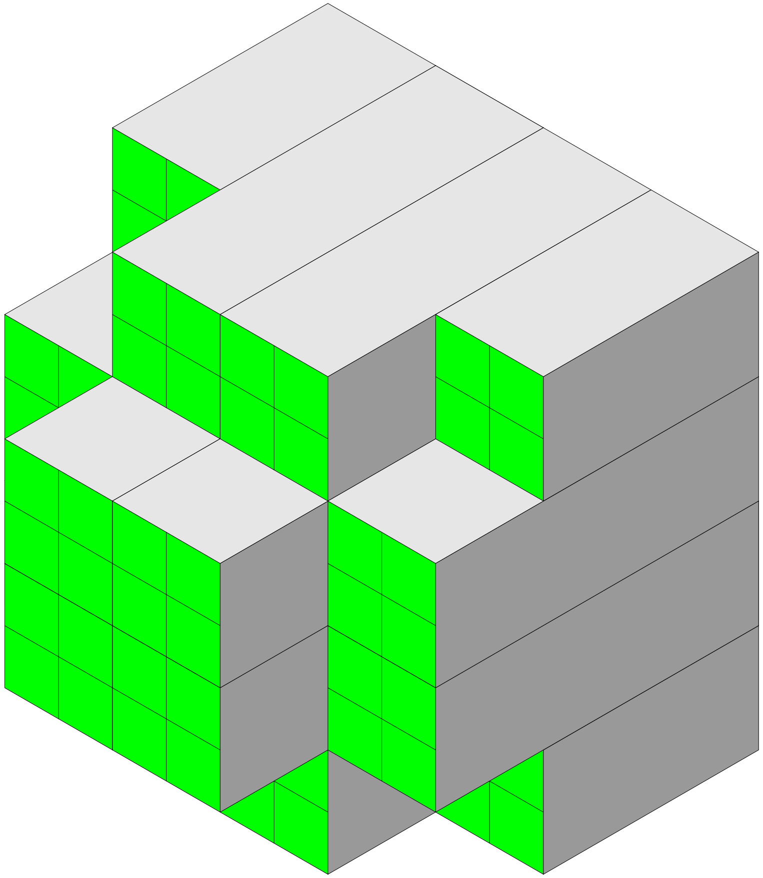
2. We find an approximation of the surface



Our method

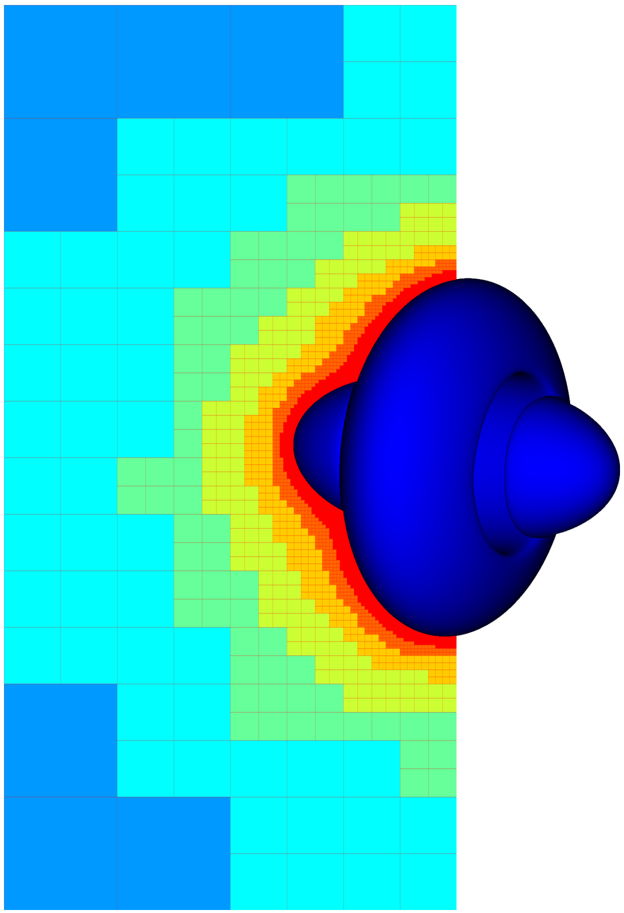
Spatial subdivision

1. Large beams are cast
2. We find an approximation of the surface
3. The remaining beams are subdivided



Our method

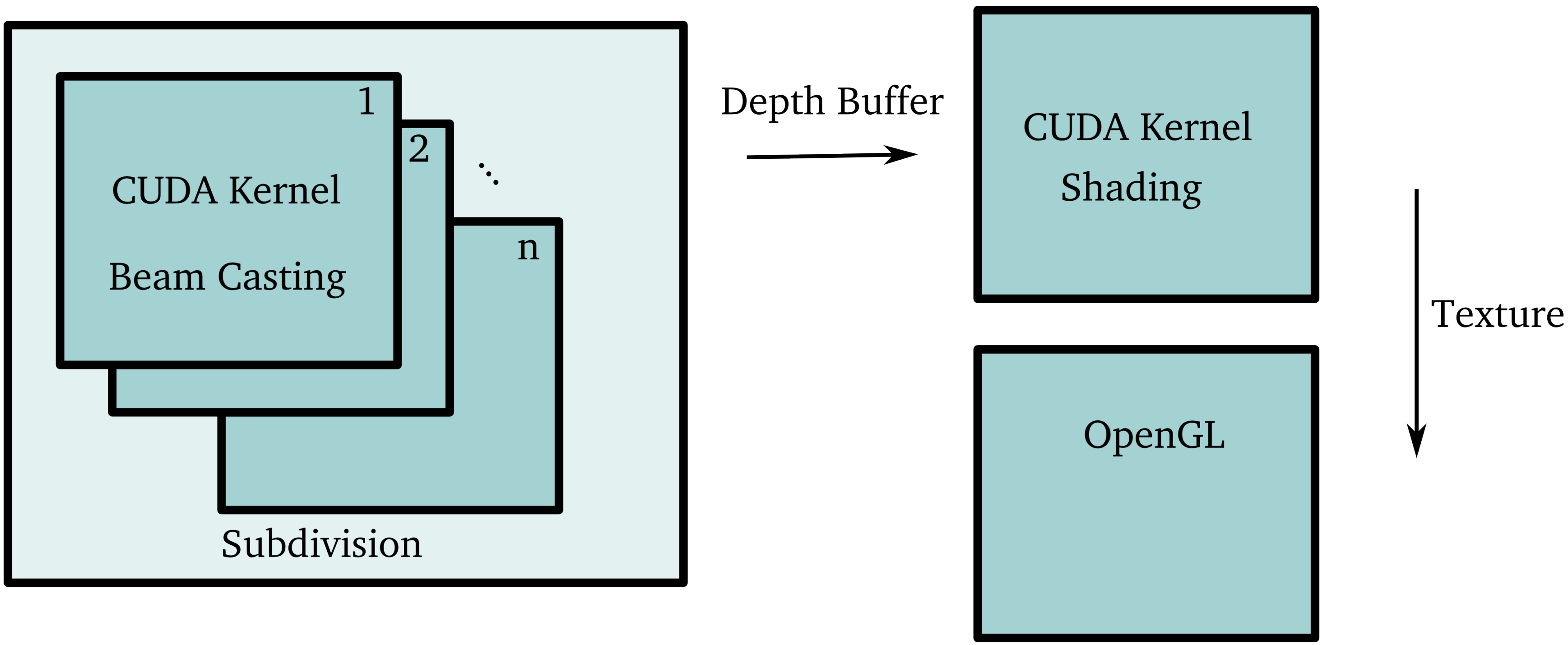
Image space adaptation



- Regions far from the surface are eliminated by large beams
- Regions near the surface require more accuracy and therefore smaller beams

Implementation

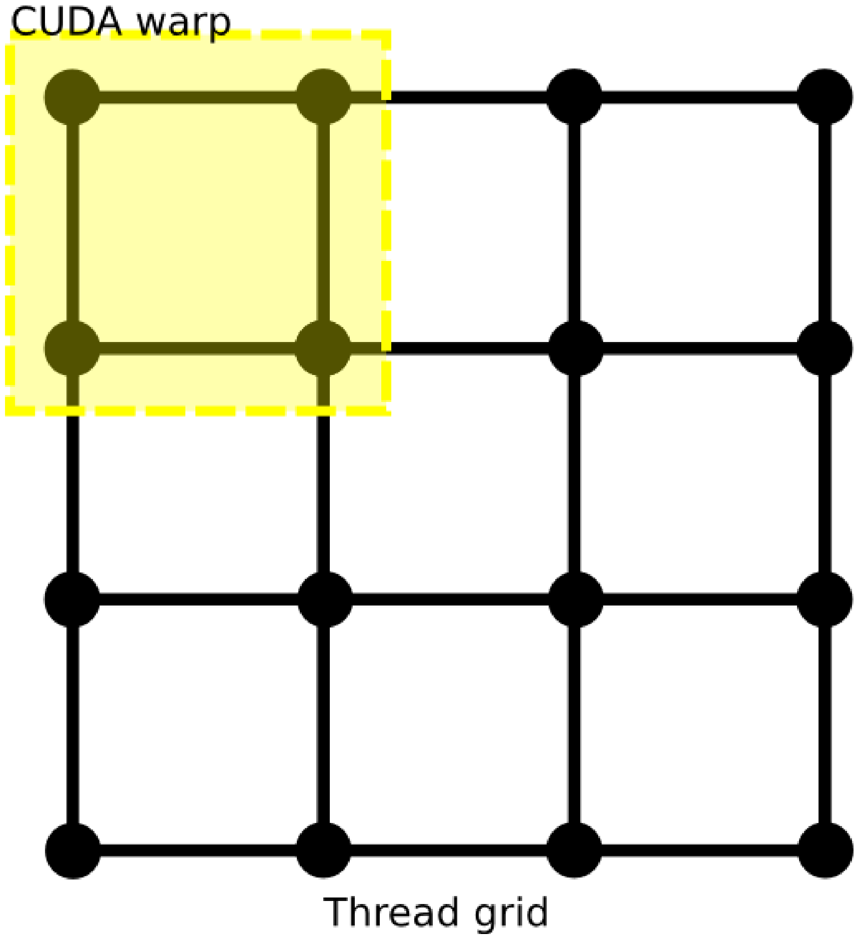
architecture



Implementation

Casting step

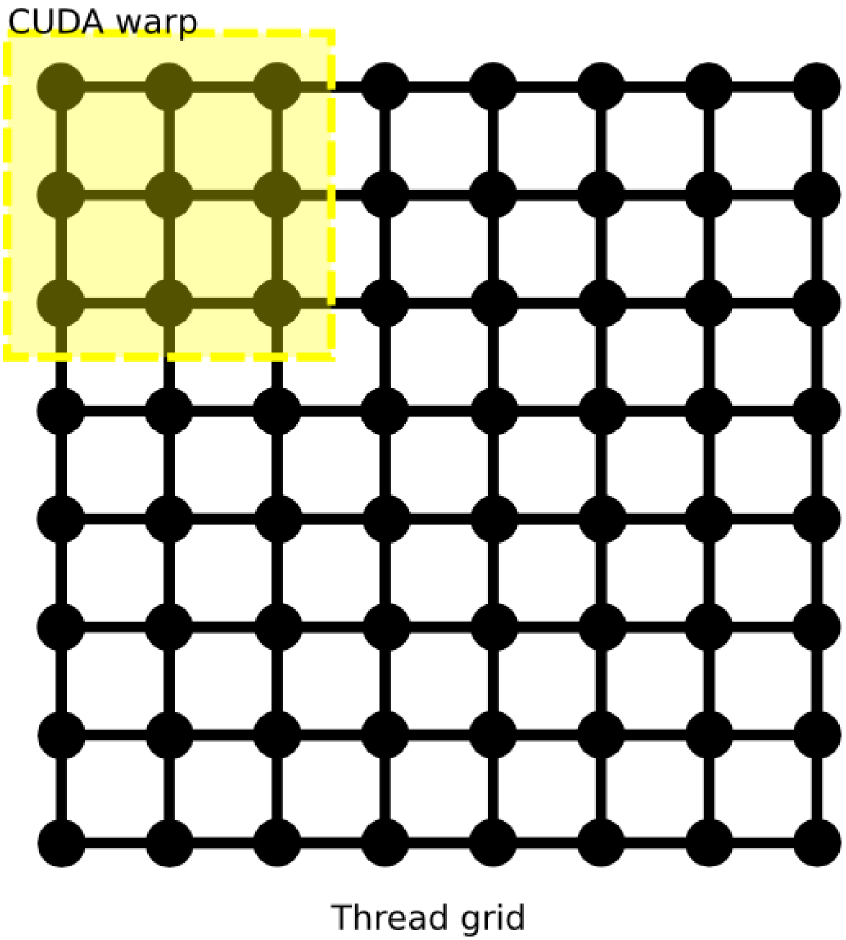
- In each step, the beams are arranged on a grid
- Each beam is processed by one CUDA thread
- The threads are grouped in warps by their geometric proximity to reduce divergence



Implementation

Casting step

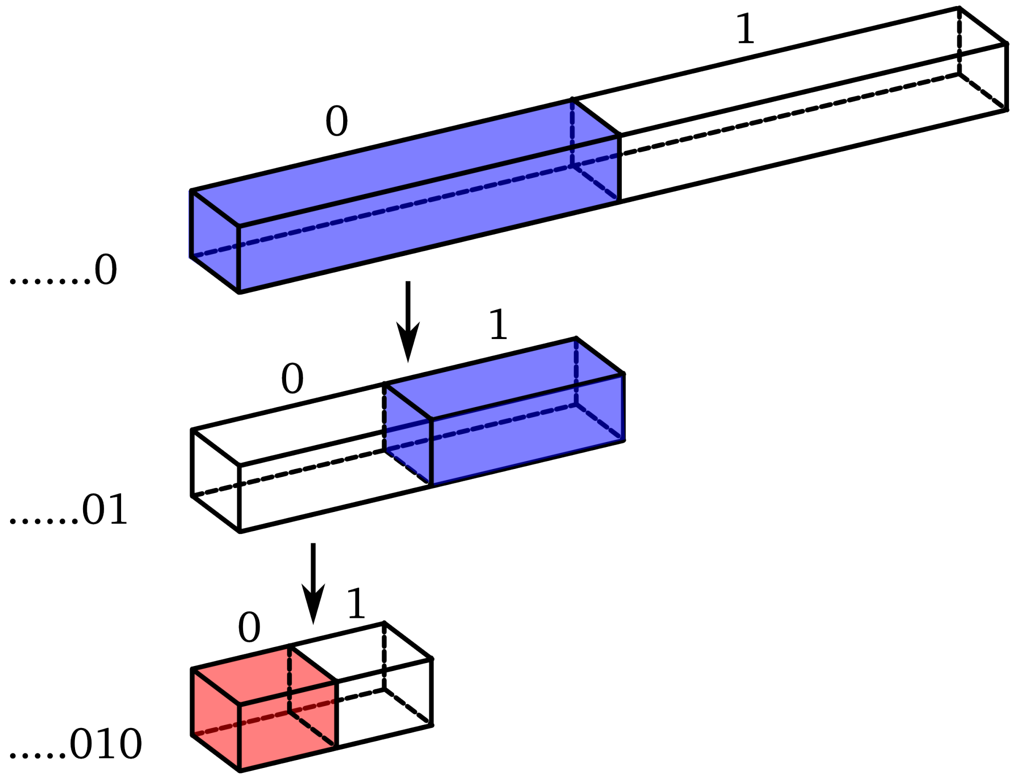
- The result of one step is saved in a memory buffer
- In the next step, the buffer is processed by the new grid



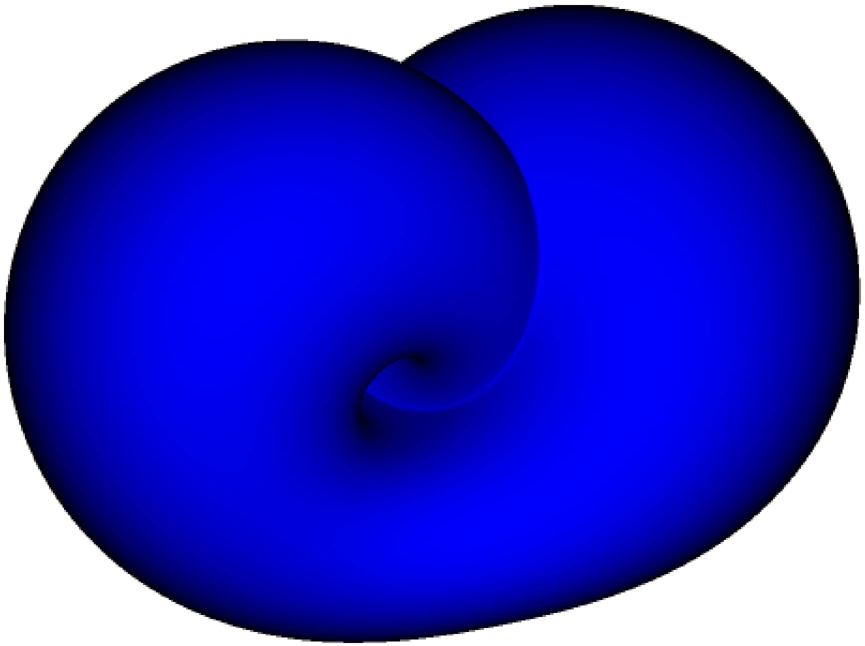
Implementation

Bisection kernel

- To avoid using recursion or an explicit stack, the bisection kernel stores the search status in one integer



Results

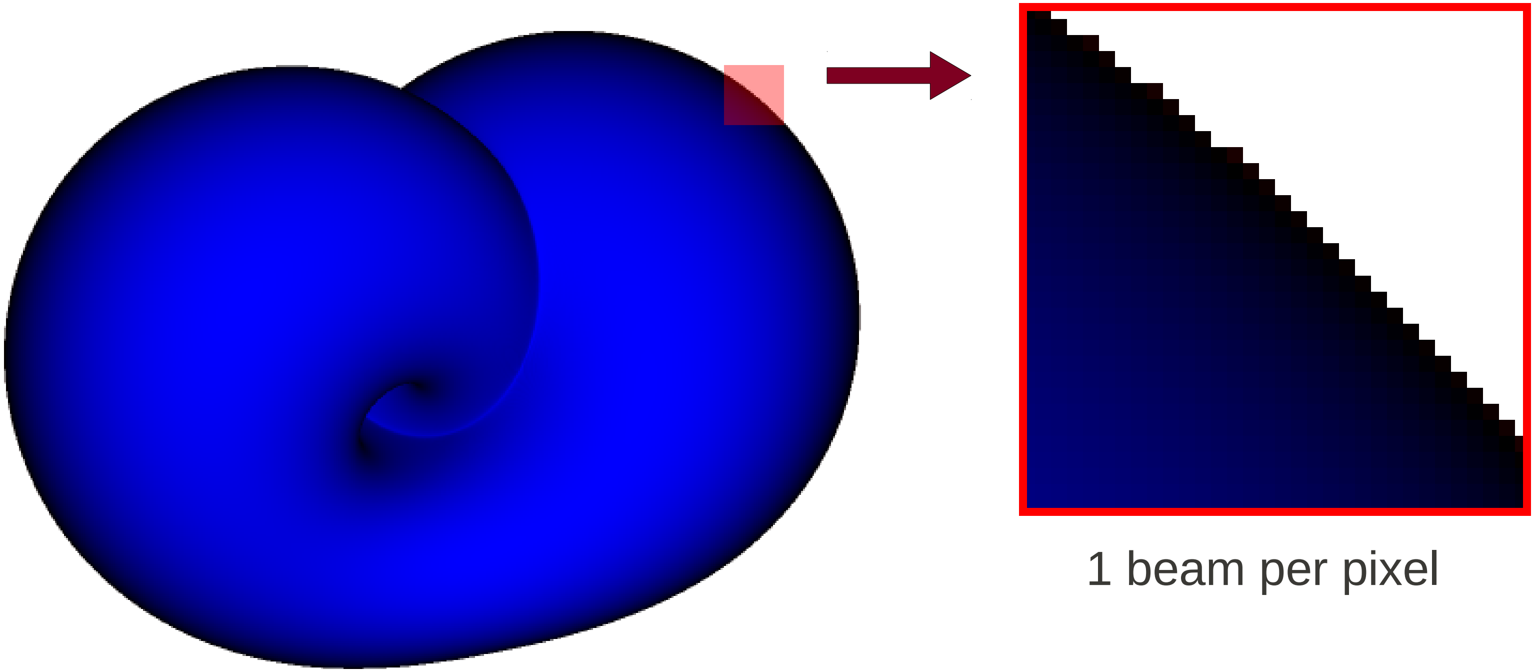


	1 beam per pixel	4 beams per pixel
512^2 pixels	362	133
1024^2 pixels	131	41

performance in *fps*

Klein: $(x^2+y^2+z^2+2y-1)((x^2+y^2+z^2-2y-1)^2-8z^2)+16xz(x^2+y^2+z^2-2y-1)$

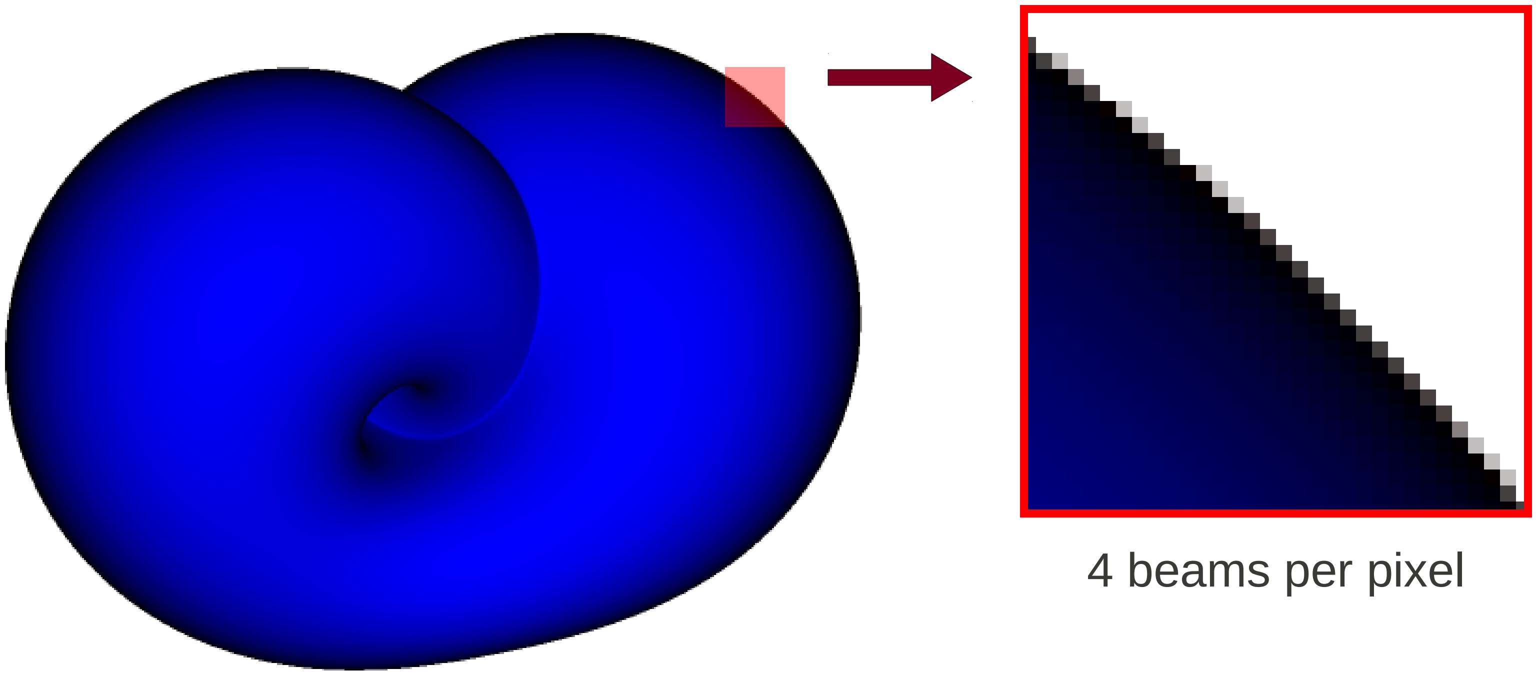
Results



1 beam per pixel

Klein: $(x^2+y^2+z^2+2y-1)((x^2+y^2+z^2-2y-1)^2-8z^2)+16xz(x^2+y^2+z^2-2y-1)$

Results



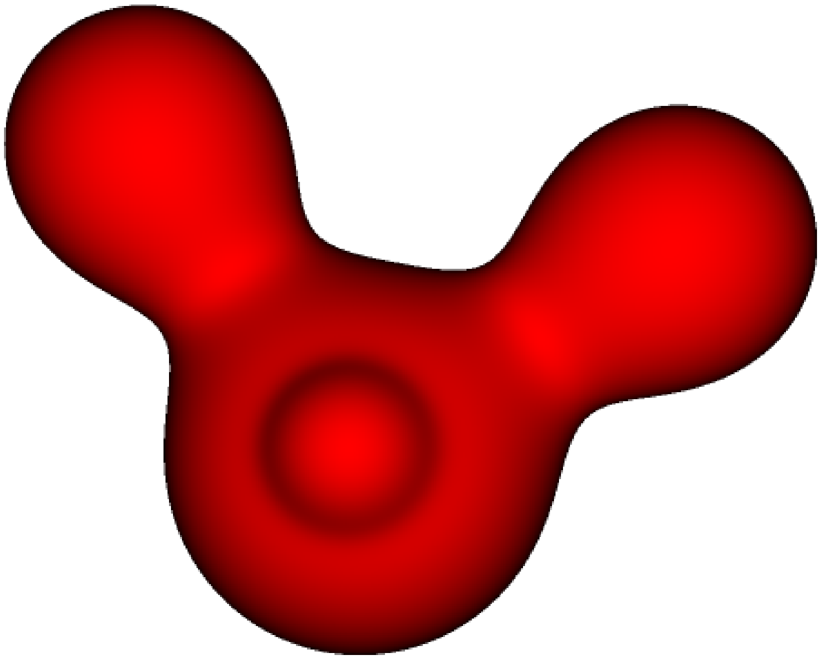
4 beams per pixel

Klein: $(x^2+y^2+z^2+2y-1)((x^2+y^2+z^2-2y-1)^2-8z^2)+16xz(x^2+y^2+z^2-2y-1)$

Results

	1 beam per pixel	4 beams per pixel
512^2 pixels	1322	570
1024^2 pixels	545	182

performance in *fps*



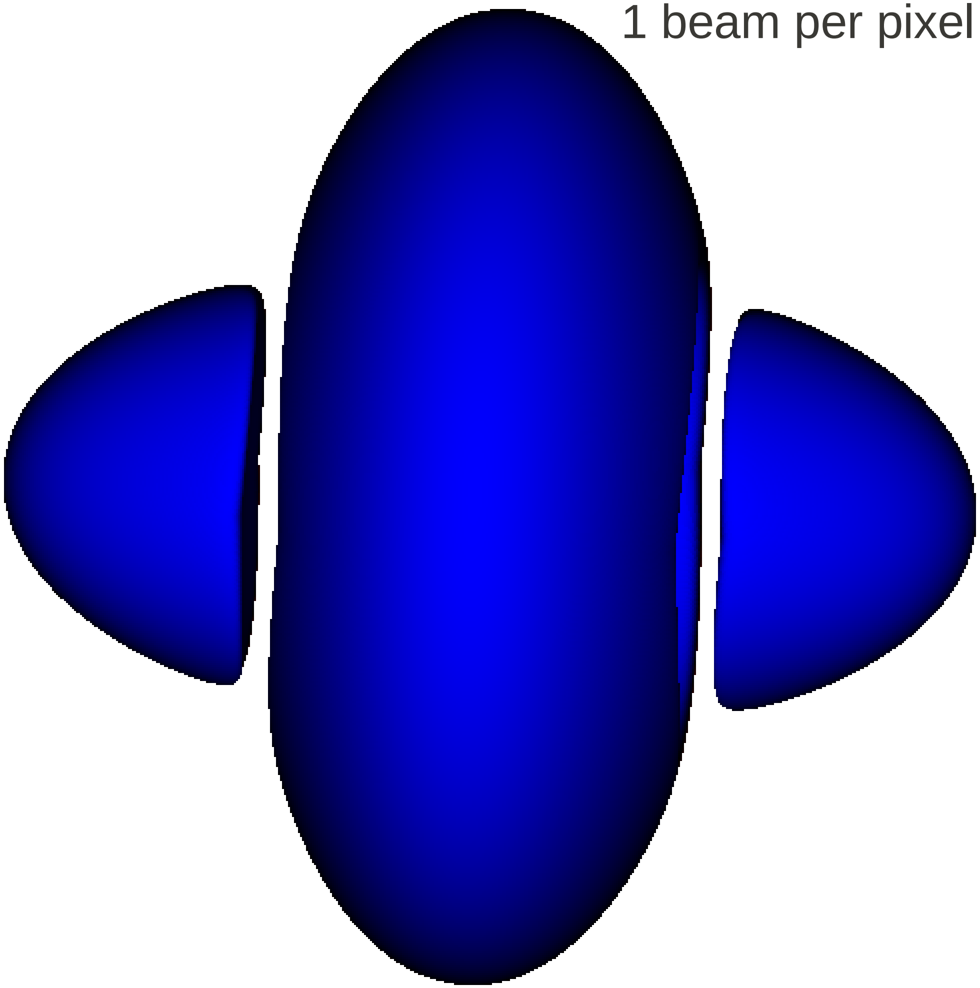
Blob *(non-algebraic)*

Results

	1 beam per pixel	4 beams per pixel
512^2 pixels	240	72
1024^2 pixels	70	26

performance in *fps*

1 beam per pixel



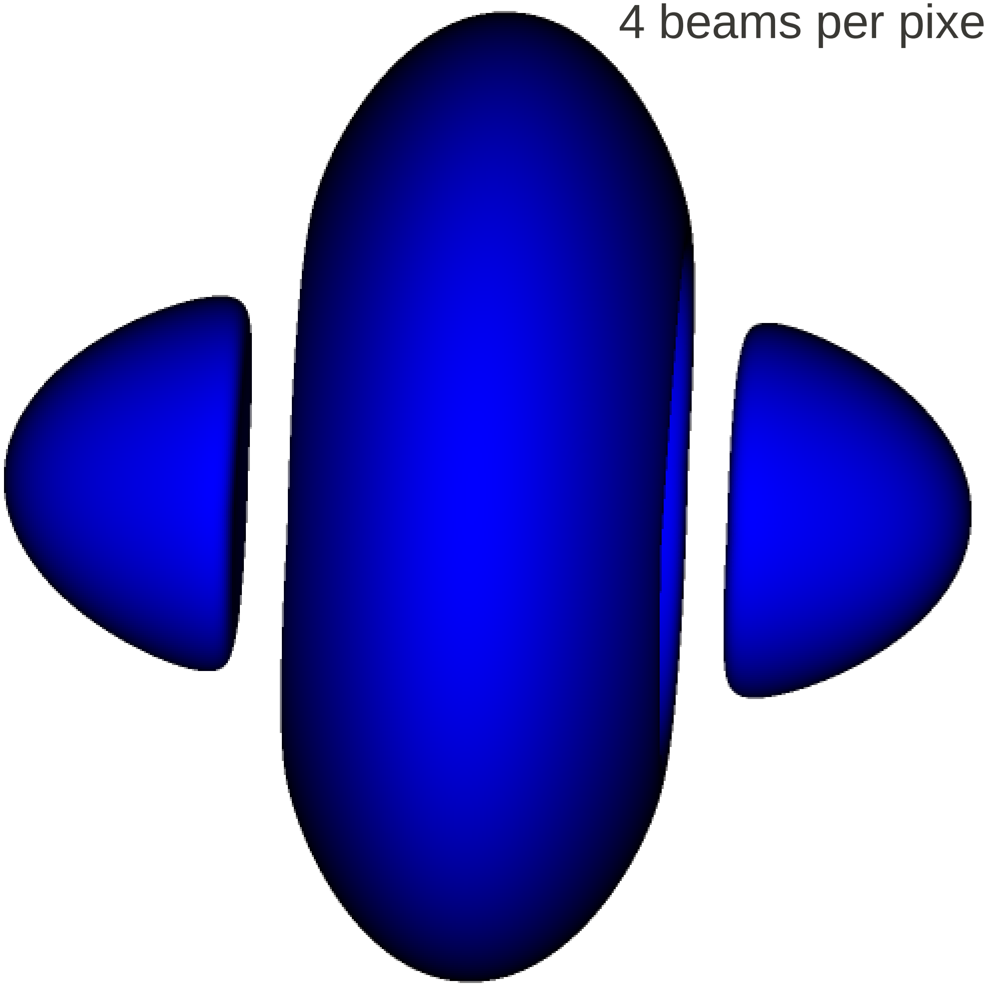
Mitchell: $4(x^4 + (y^2 + z^2)^2 + 17x^2(y^2 + z^2)) - 20(x^2 + y^2 + z^2) + 17$

Results

	1 beam per pixel	4 beams per pixel
512^2 pixels	240	72
1024^2 pixels	70	26

performance in *fps*

4 beams per pixel



Mitchell: $4(x^4 + (y^2 + z^2)^2 + 17x^2(y^2 + z^2)) - 20(x^2 + y^2 + z^2) + 17$

Conclusion

- Simple method
- Easy to implement
- Achieves real-time frame rates
- Anti-aliasing

Future directions

- Affine arithmetic
- Better load balancing
- Investigate the role of thread divergence

Thank you!

SIBGRAPI 2011

Beam casting implicit surfaces on the GPU with interval arithmetic

Francisco Ganacim
Luiz Henrique de Figueiredo
Diego Nehab

Maceió – Alagoas - Brazil

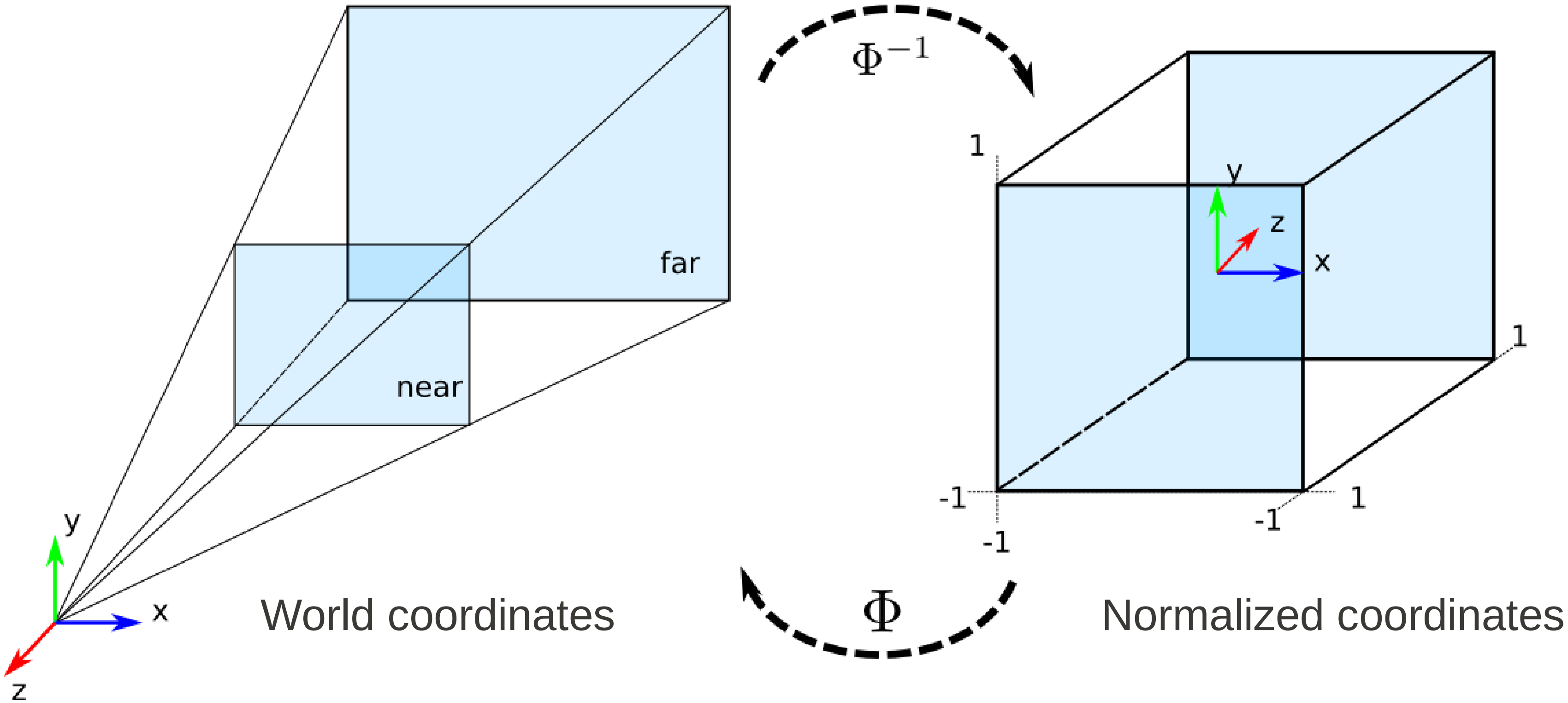
VisgrafLab

impa

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico
60 ANOS

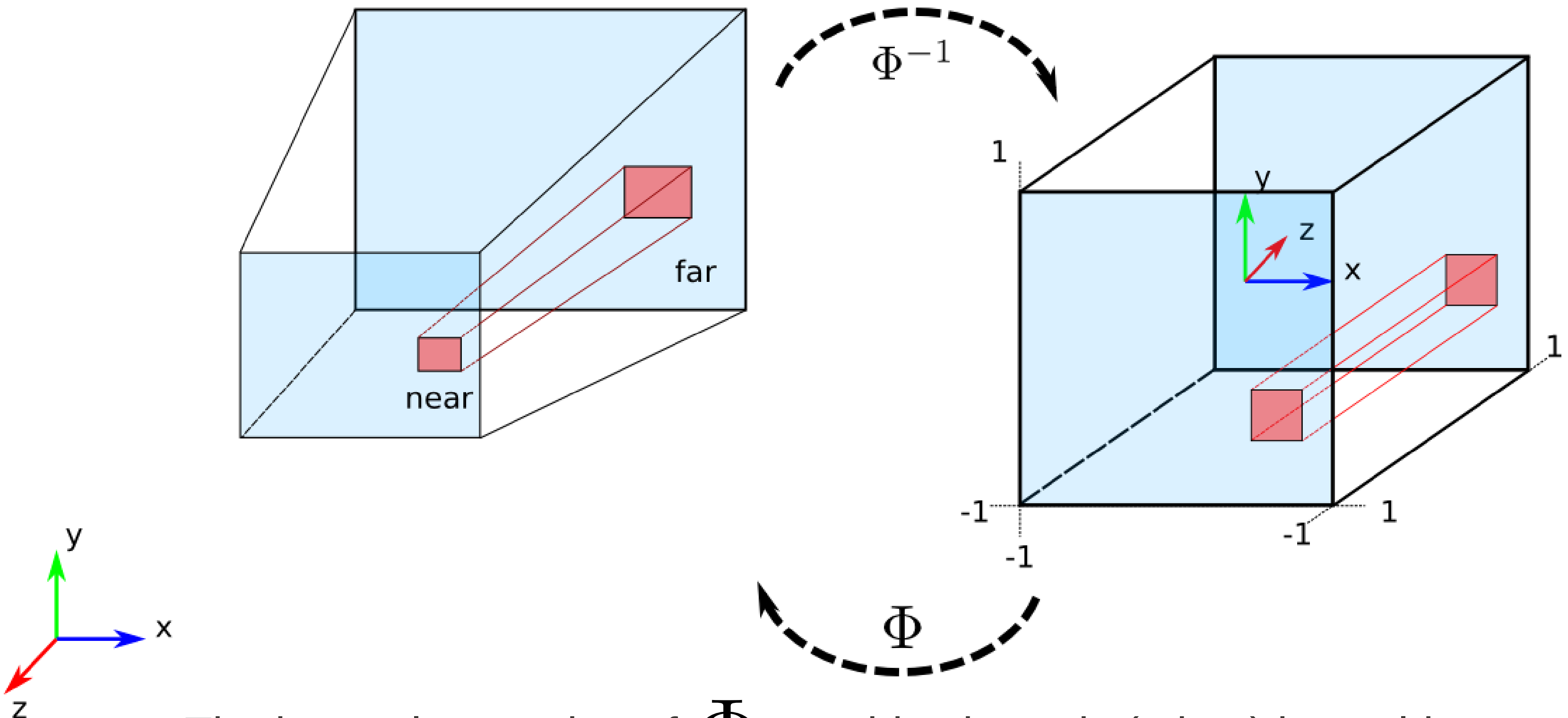
Our method

Projection transformation



Our method

Projection transformation



•The interval extension of Φ provides bounds (a box) in world coordinates, to a given normalized volume