# Ray Tracing Animations Using 4D Kd-Trees

Jens Olsson

Examensarbete 2007-02-19

# Outline

- Introduction
  - Ray-Tracing
  - Motion Blur
  - Acceleration Structures
- Project
  - Motivation
  - Theory
  - Implementation
- Results
  - Data
  - Conclusions
  - Future Work

# Outline

- Introduction
  - Ray-Tracing
  - Motion Blur
  - Acceleration Structures
- Project
  - Motivation
  - Theory
  - Implementation
- Results
  - Data
  - Conclusions
  - Future Work

# Outline

- Introduction
  - Ray-Tracing
  - Motion Blur
  - Acceleration Structures
- Project
  - Motivation
  - Theory
  - Implementation
- Results
  - Data
  - Conclusions
  - Future Work

# Outline

- Introduction
  - Ray-Tracing
  - Motion Blur
  - Acceleration Structures
- Project
  - Motivation
  - Theory
  - Implementation
- Results
  - Data
  - Conclusions
  - Future Work

# Ray-Tracing

- Technique used to generate convincing imagery
- Mimics a camera, but reversed light flow
- Millions of rays are traced through the scene
- Very general, extends naturally from simple to complex models

# Ray-Tracing

- Technique used to generate convincing imagery
- Mimics a camera, but reversed light flow
- Millions of rays are traced through the scene
- Very general, extends naturally from simple to complex models

# Ray-Tracing

- Technique used to generate convincing imagery
- Mimics a camera, but reversed light flow
- Millions of rays are traced through the scene
- Very general, extends naturally from simple to complex models

# Ray-Tracing

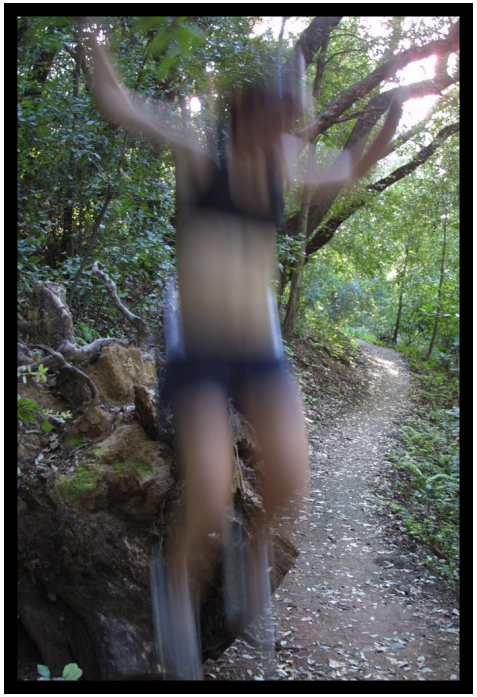
- Technique used to generate convincing imagery
- Mimics a camera, but reversed light flow
- Millions of rays are traced through the scene
- Very general, extends naturally from simple to complex models

# Ray-Tracing

- Technique used to generate convincing imagery
- Mimics a camera, but reversed light flow
- Millions of rays are traced through the scene
- Very general, extends naturally from simple to complex models

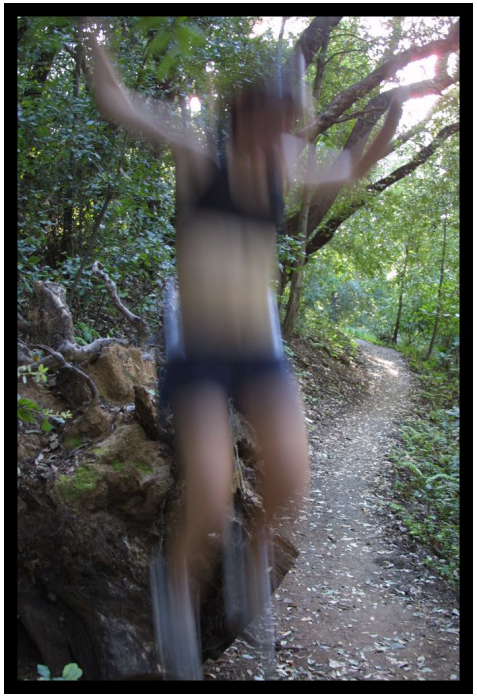
# Motion Blur

- Camera film is exposed for a short moment
- Moving objects exposure are spread out over the film
- The objects appear as smeared in the accumulated image



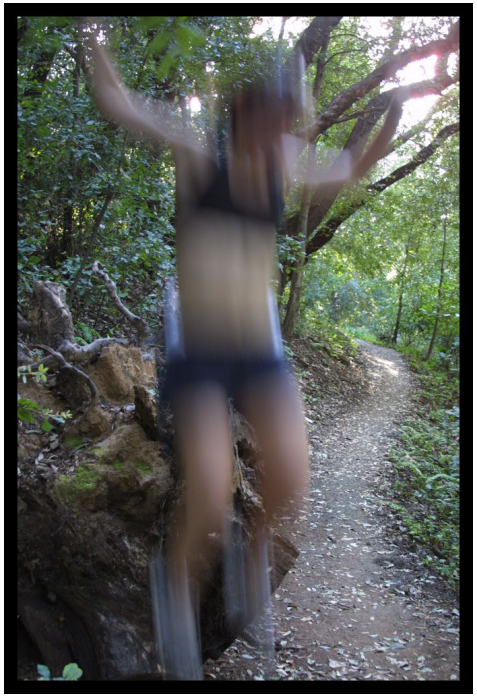
# Motion Blur

- Camera film is exposed for a short moment
- Moving objects exposure are spread out over the film
- The objects appear as smeared in the accumulated image



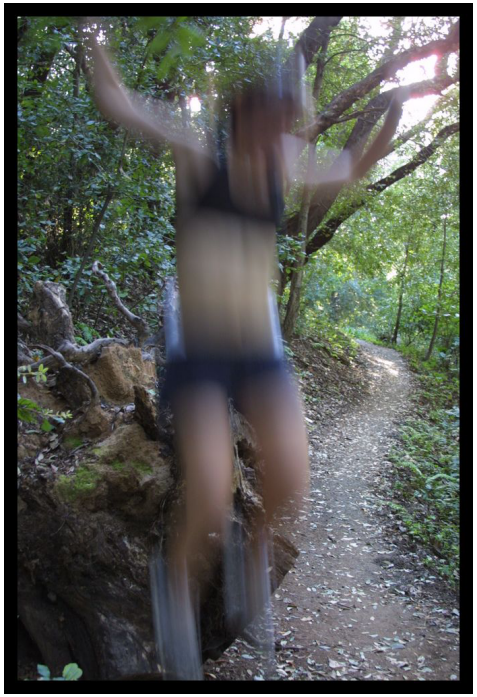
# Motion Blur

- Camera film is exposed for a short moment
- Moving objects exposure are spread out over the film
- The objects appear as smeared in the accumulated image



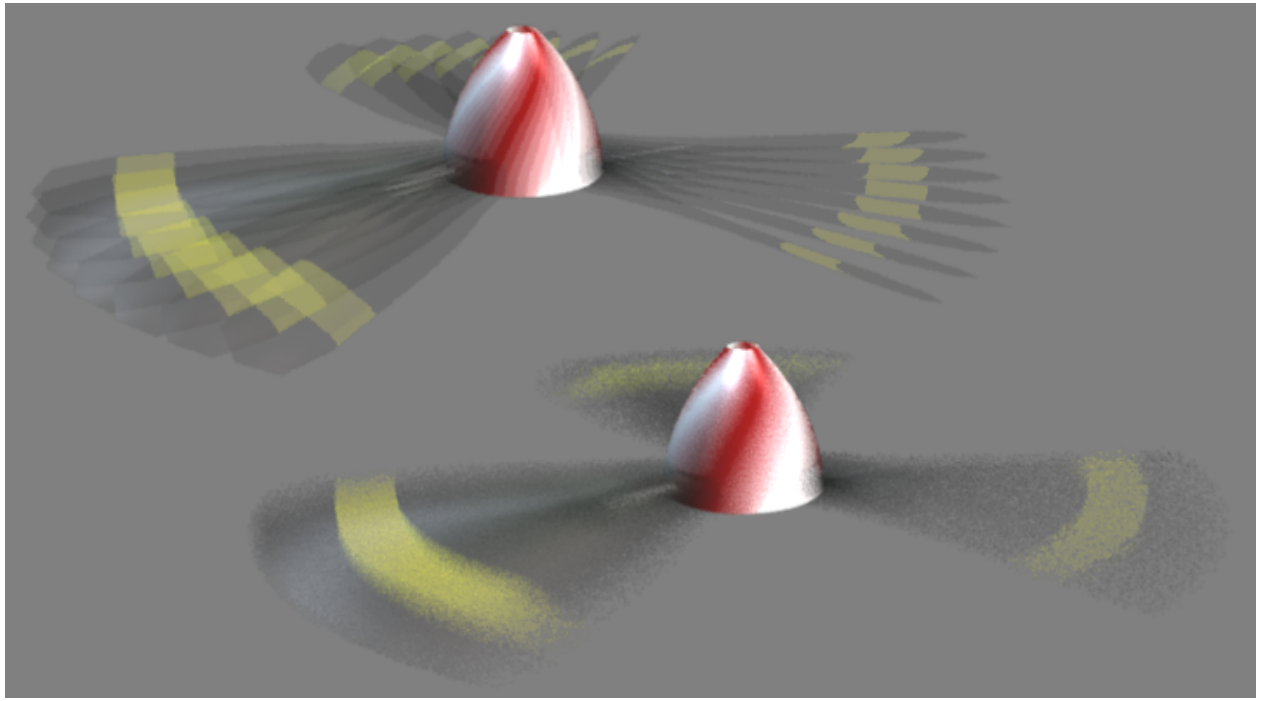
# Motion Blur

- Camera film is exposed for a short moment
- Moving objects exposure are spread out over the film
- The objects appear as smeared in the accumulated image



# Motion Blur contd.

- Simulated by stochastic sampling, accumulation buffers or blur post-process



# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Acceleration Structures

- Ray tracing mainly performs visibility queries
- A strategy is needed to make visibility queries fast
- All strategies pre-sort scene primitives in some way
- Grids , Bounding Volume Hierarchies , BSP Trees , Ray Classification etc etc etc

# Kd-Trees

- Simple data structure in the BSP tree family
- Scene volume is recursively subdivided by arbitrarily positioned axis-aligned splitting planes
- Much research has already been done, regarded as one of the most efficient acceleration structures
- Existing algorithms for construction as well as traversal

# Kd-Trees

- Simple data structure in the BSP tree family
- Scene volume is recursively subdivided by arbitrarily positioned axis-aligned splitting planes
- Much research has already been done, regarded as one of the most efficient acceleration structures
- Existing algorithms for construction as well as traversal

# Kd-Trees

- Simple data structure in the BSP tree family
- Scene volume is recursively subdivided by arbitrarily positioned axis-aligned splitting planes
- Much research has already been done, regarded as one of the most efficient acceleration structures
- Existing algorithms for construction as well as traversal

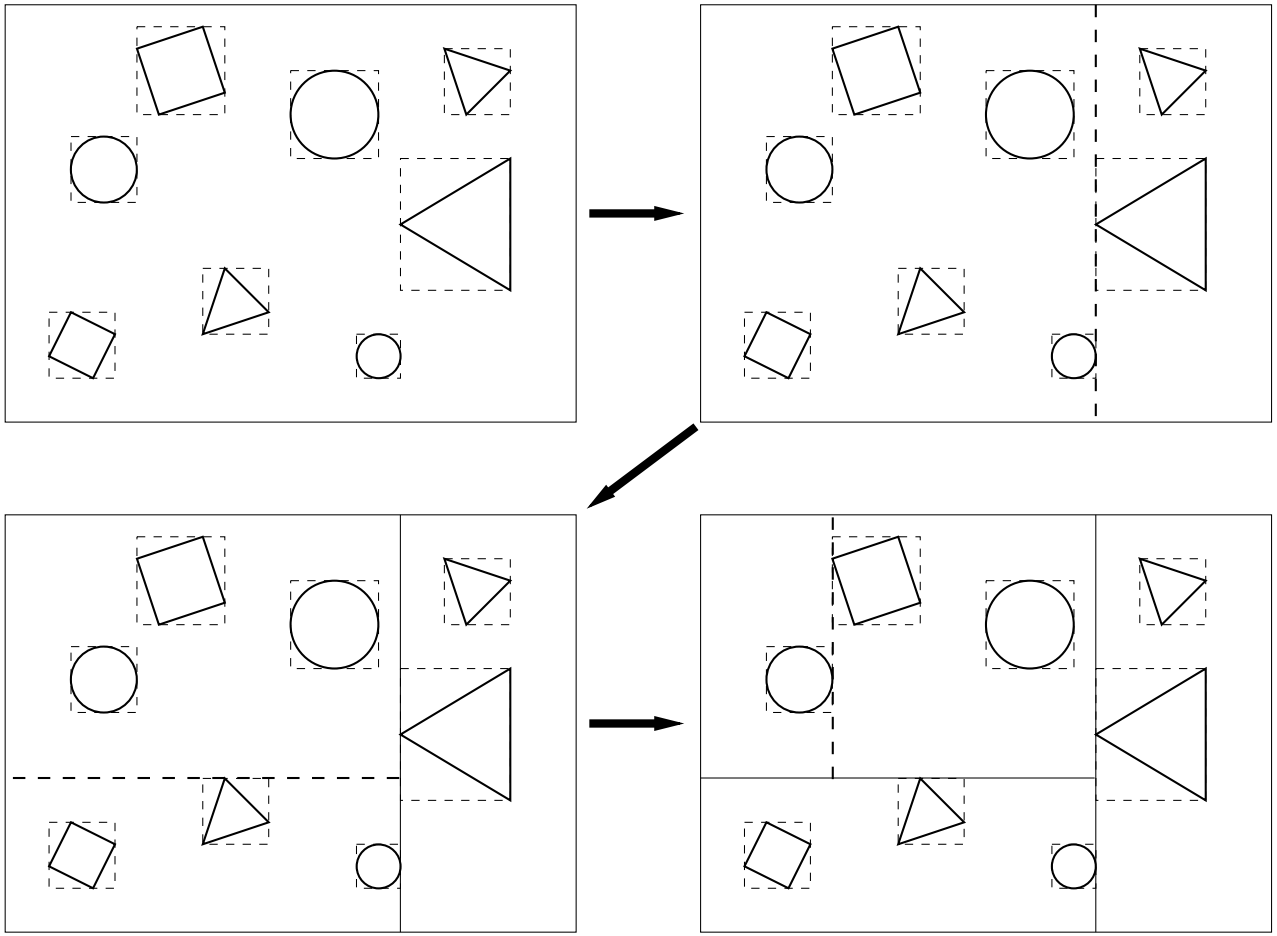
# Kd-Trees

- Simple data structure in the BSP tree family
- Scene volume is recursively subdivided by arbitrarily positioned axis-aligned splitting planes
- Much research has already been done, regarded as one of the most efficient acceleration structures
- Existing algorithms for construction as well as traversal

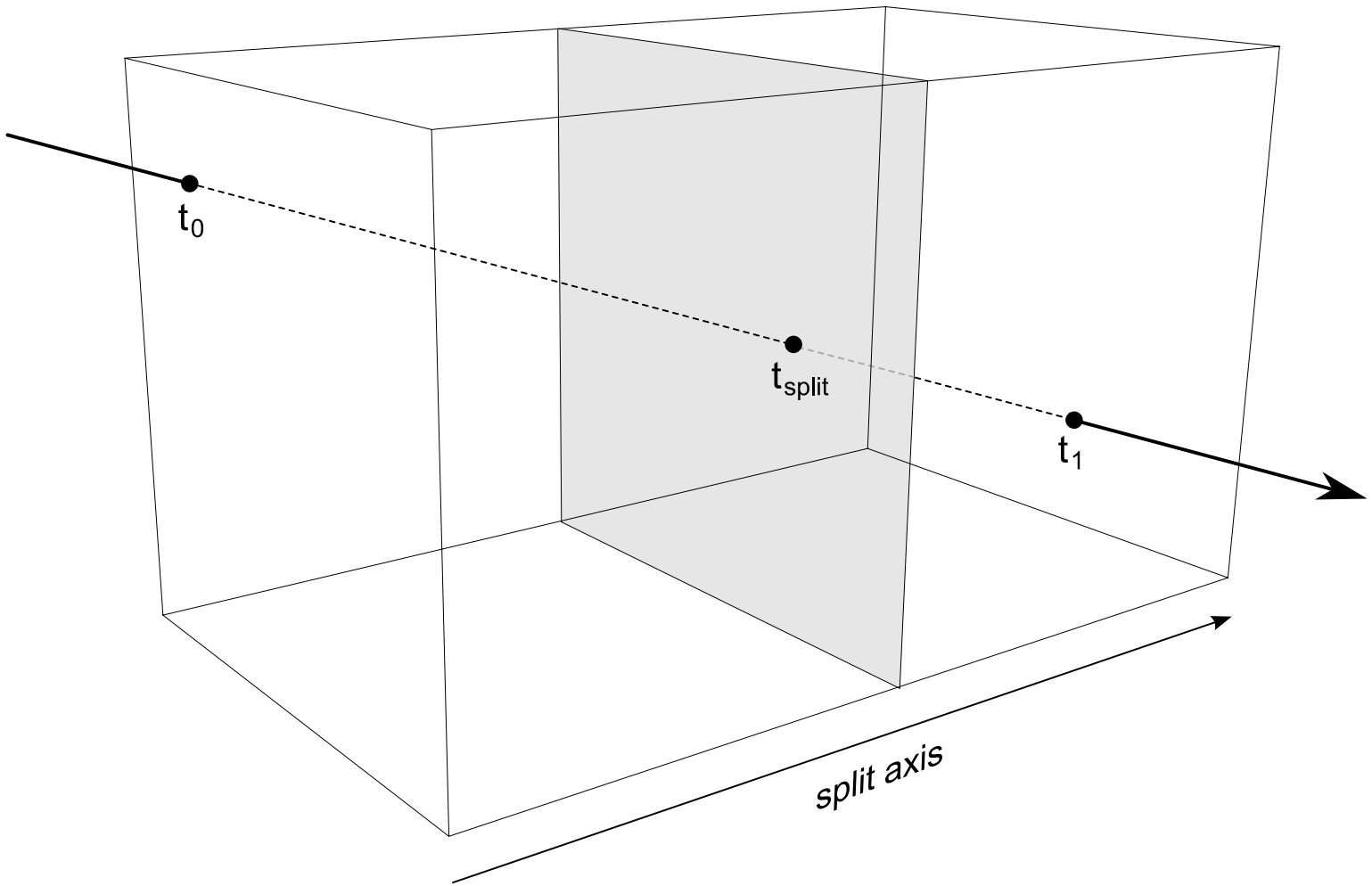
# Kd-Trees

- Simple data structure in the BSP tree family
- Scene volume is recursively subdivided by arbitrarily positioned axis-aligned splitting planes
- Much research has already been done, regarded as one of the most efficient acceleration structures
- Existing algorithms for construction as well as traversal

# Kd-Trees contd.



# Kd-Trees contd.



# Motivation

- Easy to formulate time-dependent ray tracing
- Attractive properties for rendering sequences of images
- Practical problems when actually implementing it
- Efficient techniques lays the foundation for more advanced future work (ie. global illumination)

# Motivation

- Easy to formulate time-dependent ray tracing
- Attractive properties for rendering sequences of images
- Practical problems when actually implementing it
- Efficient techniques lays the foundation for more advanced future work (ie. global illumination)

# Motivation

- Easy to formulate time-dependent ray tracing
- Attractive properties for rendering sequences of images
- Practical problems when actually implementing it
- Efficient techniques lays the foundation for more advanced future work (ie. global illumination)

# Motivation

- Easy to formulate time-dependent ray tracing
- Attractive properties for rendering sequences of images
- Practical problems when actually implementing it
- Efficient techniques lays the foundation for more advanced future work (ie. global illumination)

# Motivation

- Easy to formulate time-dependent ray tracing
- Attractive properties for rendering sequences of images
- Practical problems when actually implementing it
- Efficient techniques lays the foundation for more advanced future work (ie. global illumination)

# Theory

We wish to adapt kd-trees to store time-dependent data

- We need to subdivide primitives temporally
- We need a new heuristic for constructing the tree

# Theory

We wish to adapt kd-trees to store time-dependent data

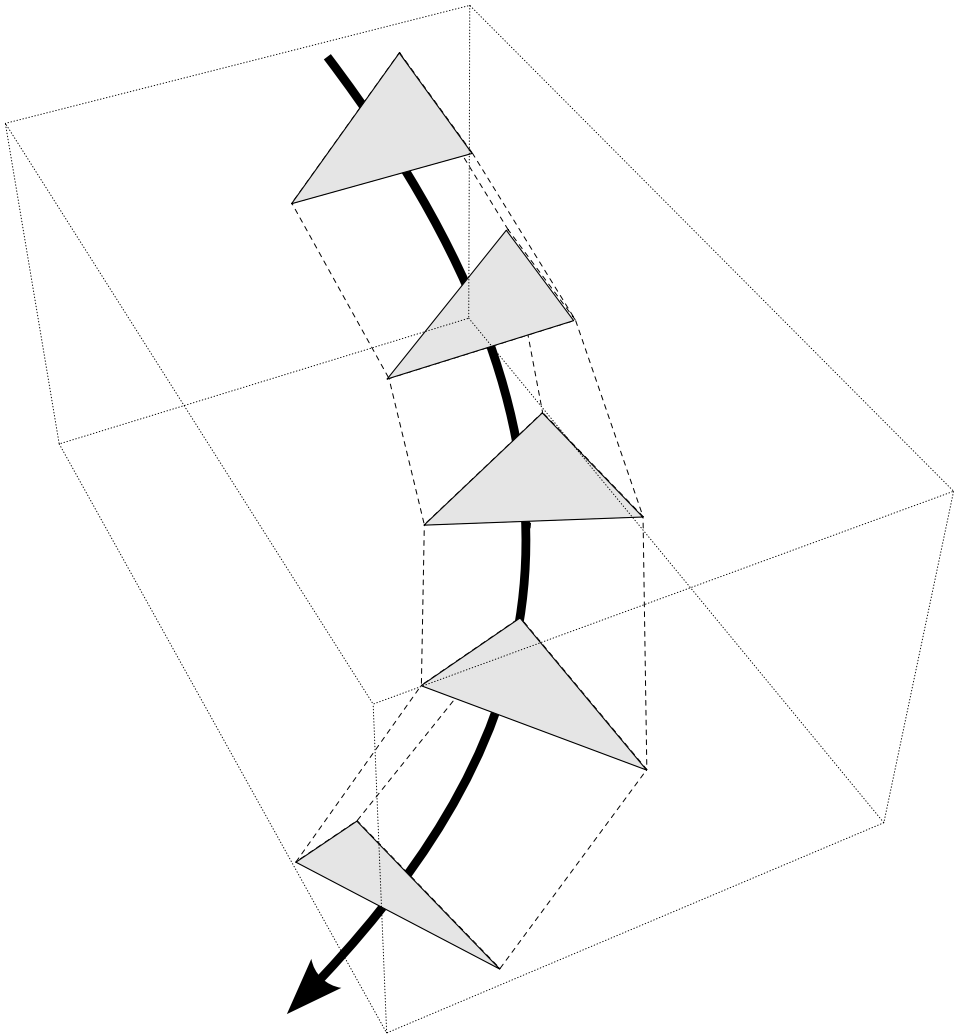
- We need to subdivide primitives temporally
- We need a new heuristic for constructing the tree

# Theory

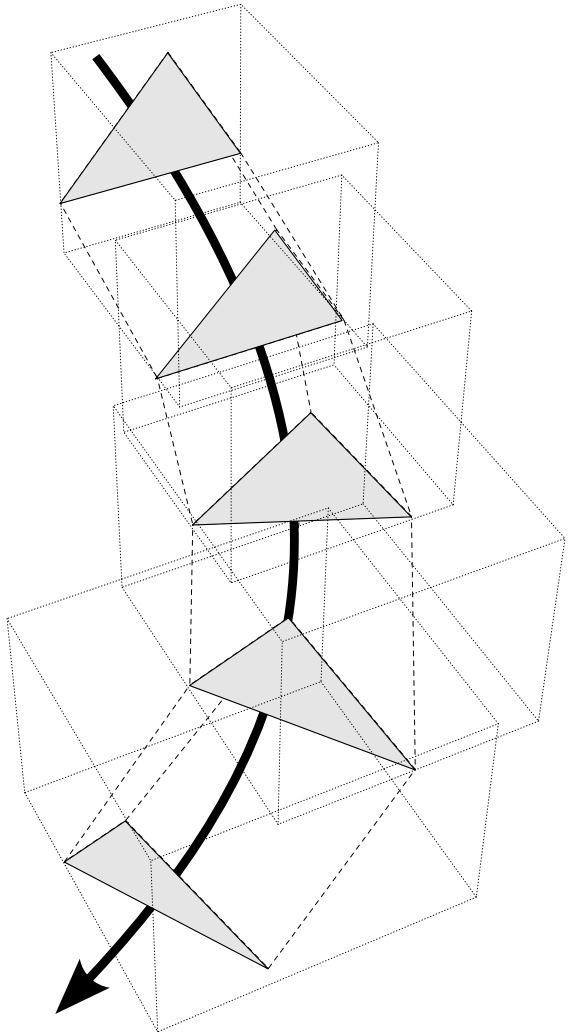
We wish to adapt kd-trees to store time-dependent data

- We need to subdivide primitives temporally
- We need a new heuristic for constructing the tree

# Theory contd.



# Theory contd.



## Theory contd.

Temporal segmentation rule

find the largest $t_b > t_a$ such that

$$\mathbf{S}\big(\mathbf{B}\big(\cup_i \mathbf{B}(P_{t_i})\big)\big) \leq \kappa\ \mathbf{S}(\mathbf{B}(P_{t_a}))\ ,\ t_i \in [t_a, t_b]$$

## Theory contd.

- Nodes are split using a heuristic
- Most popular heuristic for kd-trees are Surface Area Heuristics (SAH)
- Makes a local estimate of the future costs of traversal and intersection tests of the resulting subnodes

## Theory contd.

- Nodes are split using a heuristic
- Most popular heuristic for kd-trees are Surface Area Heuristics (SAH)
- Makes a local estimate of the future costs of traversal and intersection tests of the resulting subnodes

## Theory contd.

- Nodes are split using a heuristic
- Most popular heuristic for kd-trees are Surface Area Heuristics (SAH)
- Makes a local estimate of the future costs of traversal and intersection tests of the resulting subnodes

## Theory contd.

The SAH cost function

$$C = c_t + (1 - b_e)(p_A N_A c_i + p_B N_B c_i)$$

We reinterpreted these for the temporal case

## Theory contd.

The SAH cost function

$$C = c_t + (1 - b_e)(p_A N_A c_i + p_B N_B c_i)$$

We reinterpreted these for the temporal case

# Theory contd.

- Spatial case

$$p_A = p(\mathcal{B}_A|\mathcal{B}) = \frac{S_{\mathcal{B}_A}}{S_{\mathcal{B}}} \quad p_B = p(\mathcal{B}_B|\mathcal{B}) = \frac{S_{\mathcal{B}_B}}{S_{\mathcal{B}}}$$

- Temporal case

$$p_A = p(\mathcal{B}_A|\mathcal{B}) = \frac{|t_{\mathcal{B}_A}|}{|t_{\mathcal{B}}|} \quad p_B = p(\mathcal{B}_B|\mathcal{B}) = \frac{|t_{\mathcal{B}_B}|}{|t_{\mathcal{B}}|}$$

# Theory contd.

- Spatial case

$$p_A = p(\mathcal{B}_A|\mathcal{B}) = \frac{S_{\mathcal{B}_A}}{S_{\mathcal{B}}} \quad p_B = p(\mathcal{B}_B|\mathcal{B}) = \frac{S_{\mathcal{B}_B}}{S_{\mathcal{B}}}$$

- Temporal case

$$p_A = p(\mathcal{B}_A|\mathcal{B}) = \frac{|t_{\mathcal{B}_A}|}{|t_{\mathcal{B}}|} \quad p_B = p(\mathcal{B}_B|\mathcal{B}) = \frac{|t_{\mathcal{B}_B}|}{|t_{\mathcal{B}}|}$$

# Implementation

- Developed a simple ray tracer
- Written in C++
- Plug-in inside off-the-shelf 3D software

# Implementation

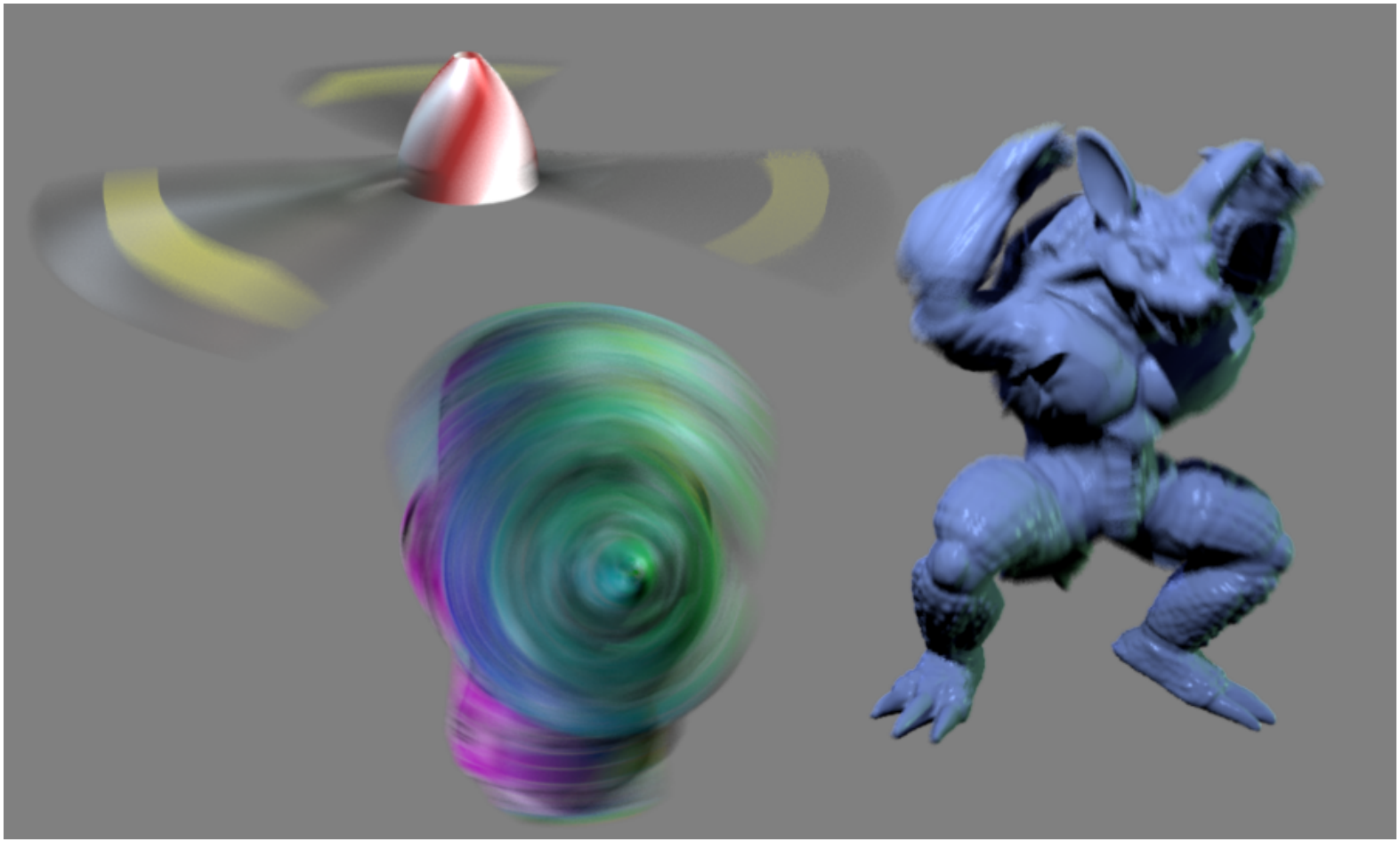
- Developed a simple ray tracer
- Written in C++
- Plug-in inside off-the-shelf 3D software

# Implementation

- Developed a simple ray tracer
- Written in C++
- Plug-in inside off-the-shelf 3D software

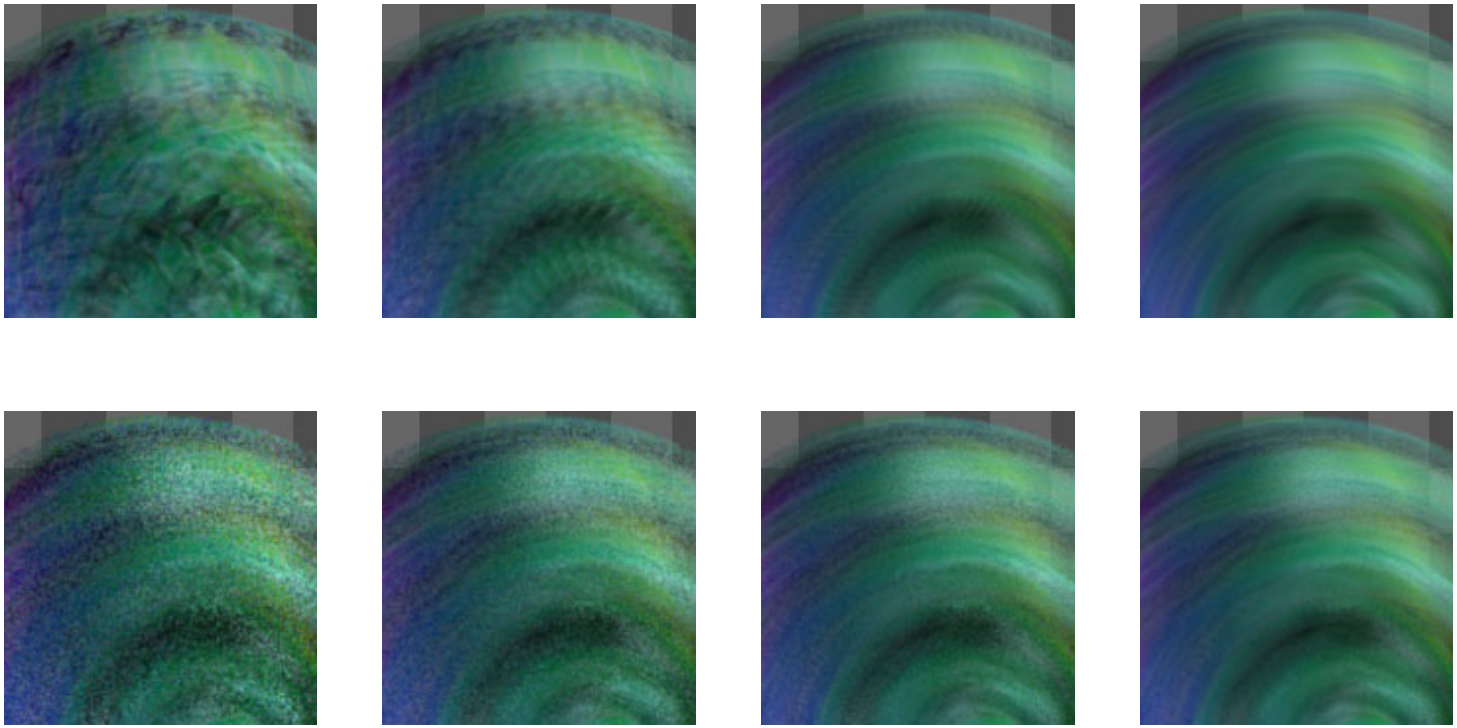
# Data

The 3 test scenes



# Data contd.

## Head Test



# Data contd.

Head Test Results

| | Head test common data | | |
|---|---|---|---|
| method | accelerator | tree memory size | primitives memory size |
| accumulation | 3D kd-tree | 0.2 MB | 0.9 MB |
| stochastic | 4D kd-tree | 18.4 MB | 12.2 MB |

# Data contd.

Head test results

| accumulation | 8 subframes | 16 subframes | 32 subframes | 64 subframes |
|---|---|---|---|---|
| time (min) | 1.29 | 2.63 | 4.99 | 8.87 |
| total rays | 1961956 | 3909952 | 7806048 | 15599764 |

| stochastic | 4 samples | 8 samples | 16 samples | 32 samples |
|---|---|---|---|---|
| time (min) | 1.25 | 1.95 | 3.37 | 6.22 |
| total rays | 933204 | 1886628 | 3822216 | 7687188 |

# Data contd.

Head test rendering time breakdown (seconds)

| accumulation | 8 subframes | 16 subframes | 32 subframes | 64 subframes |
|---|---|---|---|---|
| update data | 4.3 | 8.3 | 16.3 | 26.1 |
| build tree | 2.6 | 5.5 | 10.3 | 17.6 |
| raytrace | 70.0 | 143.5 | 271.6 | 487.6 |

| stochastic | 4 samples | 8 samples | 16 samples | 32 samples |
|---|---|---|---|---|
| update data | 1.5 | 1.5 | 1.5 | 1.5 |
| build tree | 26.6 | 26.4 | 26.0 | 26.0 |
| raytrace | 45.8 | 87.9 | 173.7 | 344.8 |

# Data contd.

Stochastic rendering time comparison (minutes)

| kd-tree | segmented | 4 samples | 8 samples | 16 samples | 32 samples |
|---|---|---|---|---|---|
| 3D | no | 16.7 | 36.3 | 65.4 | 130.7 |
| 3D | yes | 2.3 | 3.5 | 6.1 | 11.1 |
| 4D | yes | 1.2 | 1.9 | 3.4 | 6.2 |

Stochastic rendering time breakdown (seconds)

| kd-tree | segmented | update data | build tree |
|---|---|---|---|
| 3D | no | 1.2 | 0.5 |
| 3D | yes | 1.5 | 51.9 |
| 4D | yes | 1.5 | 26.0 |

# Conclusions

- Possible to store animation in a 4D kd-tree
- Better performance compared to storing the animation in a 3D kd-tree
- Memory requirements were higher than expected
- Interesting applications for 4D kd-trees

# Conclusions

- Possible to store animation in a 4D kd-tree
- Better performance compared to storing the animation in a 3D kd-tree
- Memory requirements were higher than expected
- Interesting applications for 4D kd-trees

# Conclusions

- Possible to store animation in a 4D kd-tree
- Better performance compared to storing the animation in a 3D kd-tree
- Memory requirements were higher than expected
- Interesting applications for 4D kd-trees

# Conclusions

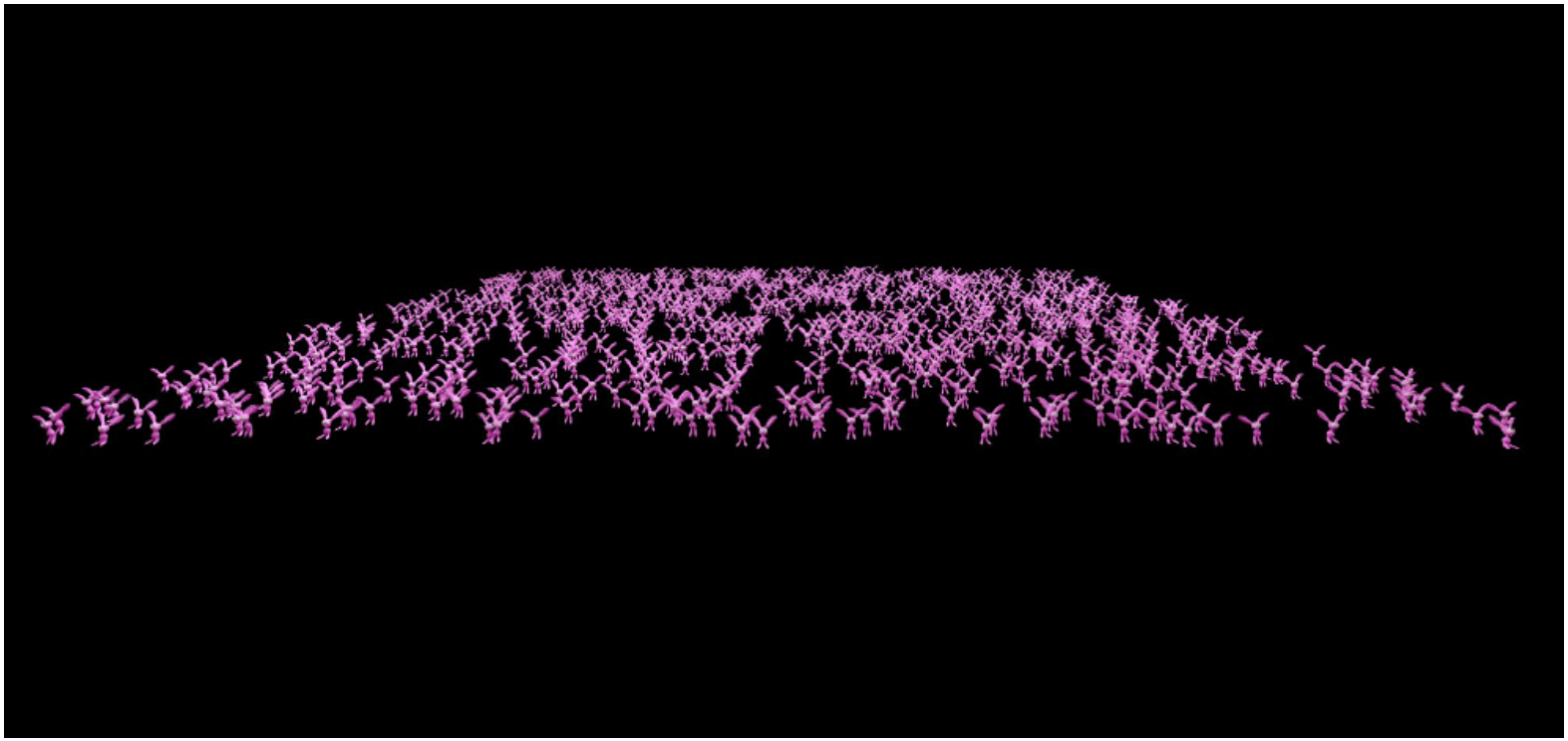
- Possible to store animation in a 4D kd-tree
- Better performance compared to storing the animation in a 3D kd-tree
- Memory requirements were higher than expected
- Interesting applications for 4D kd-trees

# Conclusions

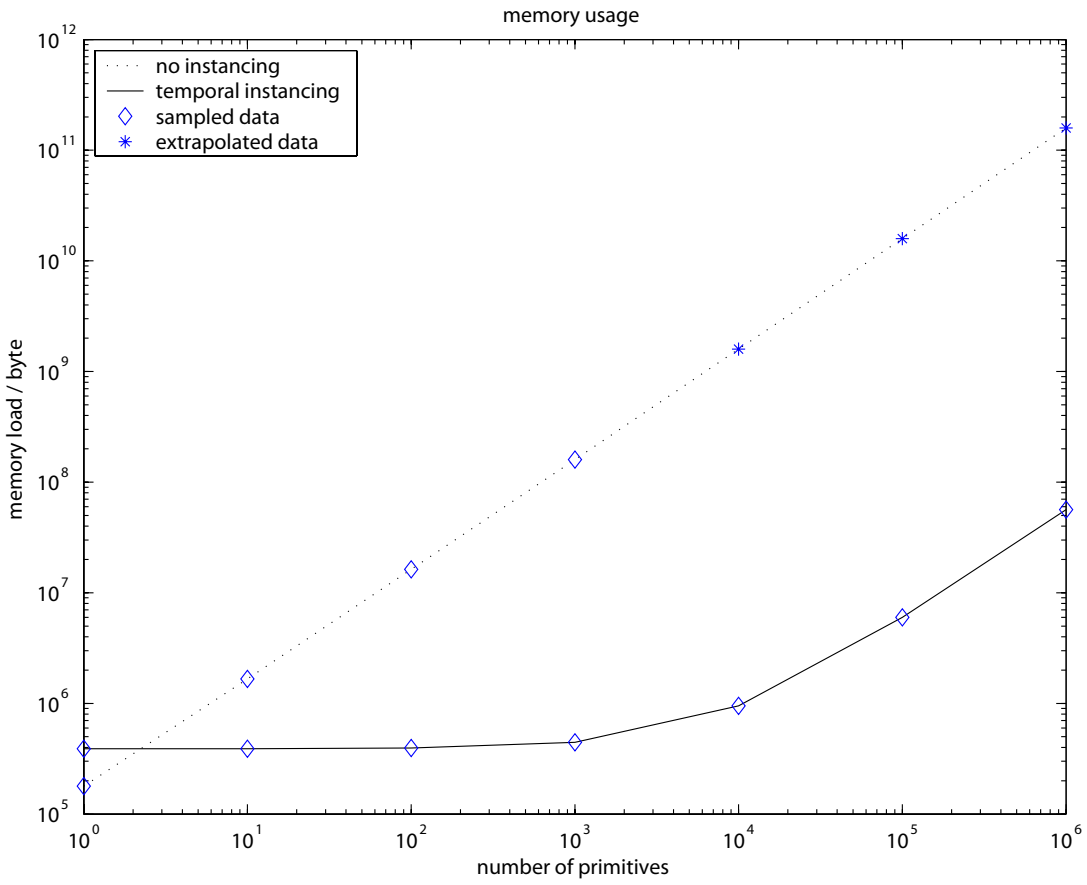
- Possible to store animation in a 4D kd-tree
- Better performance compared to storing the animation in a 3D kd-tree
- Memory requirements were higher than expected
- Interesting applications for 4D kd-trees

# Applications - Temporally instanced primitives

# Applications - Temporally instanced primitives



# Applications - Temporally instanced primitives



# Future Work

- Triangle interpolation needs to be way faster; is it possible to cpu- and cache-optimize the triangle interpolation in an efficient way?
- Costs have to be calculated at least twice for every node; Any way of avoiding this without explicitly relating temporal and spatial axes?
- Global Illumination for animations; How can we take advantage of a time-dependent structure?
- Investigate applications further; Temporal instances could be used for crowd simulations

# Future Work

- Triangle interpolation needs to be way faster; is it possible to cpu- and cache-optimize the triangle interpolation in an efficient way?
- Costs have to be calculated at least twice for every node; Any way of avoiding this without explicitly relating temporal and spatial axes?
- Global Illumination for animations; How can we take advantage of a time-dependent structure?
- Investigate applications further; Temporal instances could be used for crowd simulations

## Future Work

- Triangle interpolation needs to be way faster; is it possible to cpu- and cache-optimize the triangle interpolation in an efficient way?
- Costs have to be calculated at least twice for every node; Any way of avoiding this without explicitly relating temporal and spatial axes?
- Global Illumination for animations; How can we take advantage of a time-dependent structure?
- Investigate applications further; Temporal instances could be used for crowd simulations

# Future Work

- Triangle interpolation needs to be way faster; is it possible to cpu- and cache-optimize the triangle interpolation in an efficient way?
- Costs have to be calculated at least twice for every node; Any way of avoiding this without explicitly relating temporal and spatial axes?
- Global Illumination for animations; How can we take advantage of a time-dependent structure?
- Investigate applications further; Temporal instances could be used for crowd simulations

## Future Work

- Triangle interpolation needs to be way faster; is it possible to cpu- and cache-optimize the triangle interpolation in an efficient way?
- Costs have to be calculated at least twice for every node; Any way of avoiding this without explicitly relating temporal and spatial axes?
- Global Illumination for animations; How can we take advantage of a time-dependent structure?
- Investigate applications further; Temporal instances could be used for crowd simulations