High-Quality Unstructured Volume Rendering on the PC Platform

Stefan Guthe, Wolfgang Strasser
WSI/GRIS University of Tuebingen

Stefan Röttger, Andreas Schieber,
Thomas Ertl
IfI/VIS University of Stuttgart

Hardware Workshop 2002

Overview

Introduction

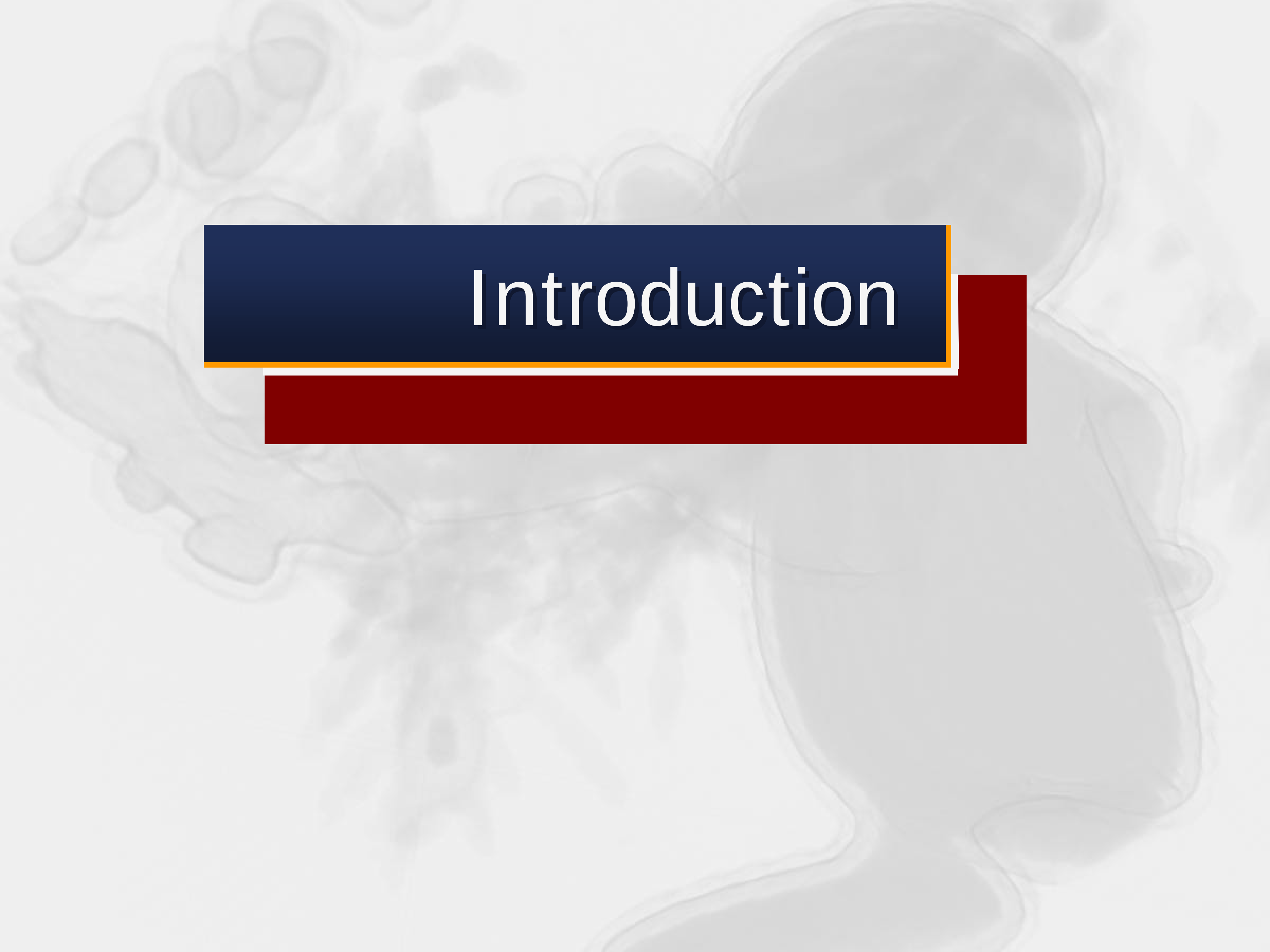
- Motivation
- Cell Projection

High Resolution Ray Integral

- Opacity Reconstruction
- Chromaticity Reconstruction

Hardware Accelerated Pre-Integration

Results & Conclusion



Introduction

Motivation

Tetrahedral Meshes

- Common for numerical simulations
- Adaptive resolution
- Straight forward multiresolution algorithms

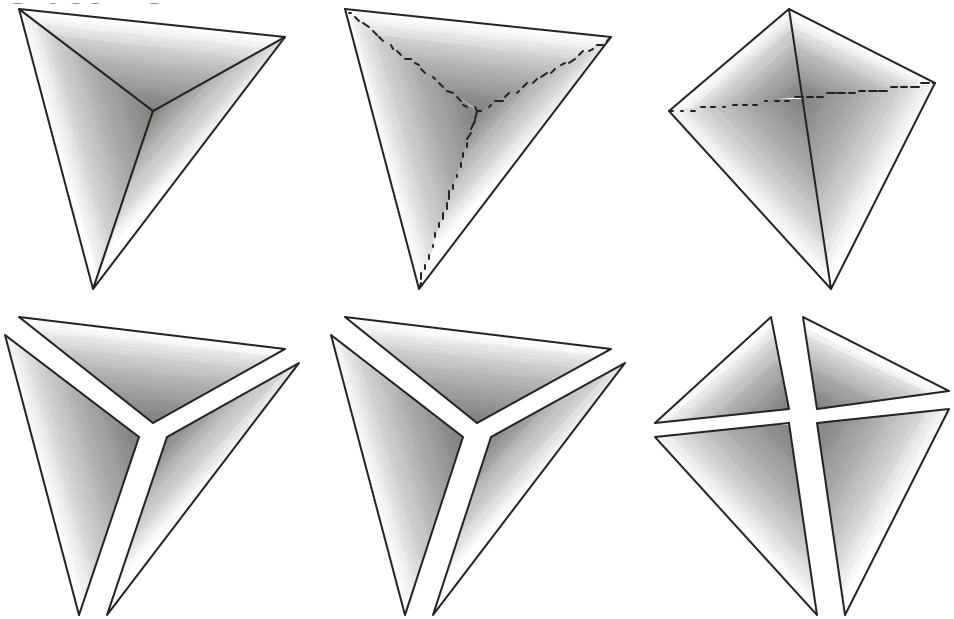
General purpose hardware

- Widely available
- Fast polygonal rendering
- Flexible fragment shading for recent generations
- Fast development of future generations
- Cheap compared to special purpose hardware

Cell Projection

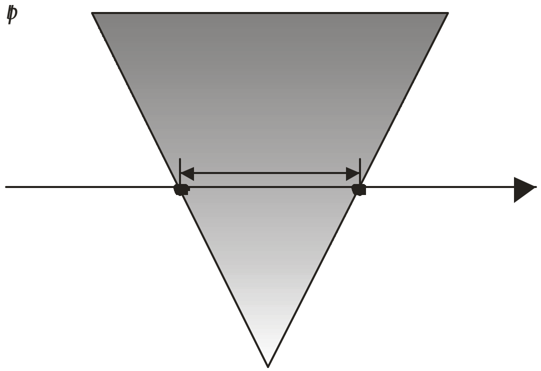
Projected Tetrahedra (PT) Algorithm

- Shirley and Tuchman ’90
- Classify tetrahedra based on profile of projection
- Split tetrahedra into 3 or 4 triangles



Cell Projection

Projected Tetrahedra (PT) Algorithm



- Render projected profiles
  - Chromaticity vector

    $$\mathbf{k} = \mathbf{k}(f(x,y,z))$$

  - Scalar optical density

    $$\mathbf{r} = \mathbf{r}(f(x,y,z))$$

  - Resulting ray integral

$$S_l(x) = S_f + \frac{x}{l}(S_b - S_f)$$

$$C(S_f, S_b, l) = \int_0^l e^{-\int_0^t r(S_l(u))\mathrm{d}u}\, \mathbf{k}(S_l(t))\, \mathbf{r}(S_l(t))\mathrm{d}t$$

$$\mathbf{a}(S_f, S_b, l) = 1 - e^{-\int_0^l r(S_l(t))\mathrm{d}t}$$

Ray Integral

Opacity Reconstruction

Approximation of opacity

- Corresponding portion of the ray integral

  $$a(S_f, S_b, l) = 1 - e^{-\int_0^l r(S_l(t))\,\mathrm{d}t}$$

- Original approximation
  - Calculate correct values for vertices
  - Interpolate linearly between vertices
- Improvement by Stein et al. ’94
  - Calculate average extinction coefficient r
  - Use texture map for exponential lookup

    $$a(lr) = 1 - e^{-lr}$$

  - Linear opacity or piecewise linear (HIAC ‘98)

Opacity Reconstruction

Approximation of opacity

- Corresponding portion of the ray integral

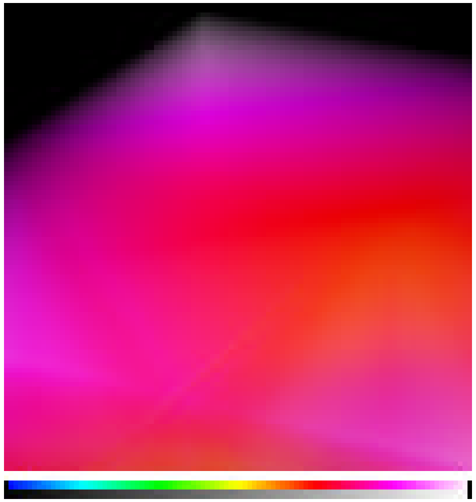
  $$a\left(S_f, S_b, l\right) = 1 - e^{-\int_0^l r(S_l(t))\mathrm{d}t}$$

- Further improvements
  - 2D texture map for lookup of average extinction
  - 1D dependent texture lookup

    $$r\left(S_f, S_b\right) = \int_0^1 r\left(S_l(t)\right)\mathrm{d}t$$

    $$a(lr) = 1 - e^{-lr}$$

  - No restriction to linear opacity



Opacity Reconstruction

Approximation of opacity (GeForce 4)

- Texture setup

unit	coordinates	RGB	A
0	S_f, S_b	chrom. (RGA)	$\int_0^1 r(S_l(t))\mathrm{d}t$
1	$0,0,l$		$1-e^{-lr}$

- Pixel shader

```
ps.1.3
def     c0, 1, 1, 0, 0
tex     t0                   // load chromaticity and density
texdp3  t1, t0               // dependent lookup
lrp     r0.rgb, c0, t0, t0.a // extract chromaticity...
+mov    r0.a, t1.a           // and alpha for final color
```

Opacity Reconstruction

Approximation of opacity (Radeon 8500)

- Texture setup

unit	coordinates	RGB	A
0	S_f, S_b	chromaticity	$\int_0^1 r(S_l(t))\mathrm{d}t$
1	$0,0,l$		$1-e^{-lr}$

- Pixel shader

```
ps.1.4
texld   r0, t0                 // load chromaticity and density
texcrd  r1, t1                 // pass l into register
mul     r1, r0.a, r1.b         // multiply density and l
phase
texpass r0, r0                 // transfer chromaticity
texld   r1, r1                 // dependent lookup
mov     r0.a, r1.a             // correct alpha for final color
```

Chromaticity Reconstruction

Approximation of chromaticity

- Corresponding portion of the ray integral

$$C(S_f, S_b, l) = \int_0^l e^{-\int_0^t r(S_l(u))\mathrm{d}u} \boldsymbol{k}(S_l(t))\, \boldsymbol{r}(S_l(t))\mathrm{d}t$$

- Original approximation
  - Calculate correct values for vertices
  - Interpolate linearly between vertices
- Improvement in HIAC ‘98
  - Calculate values for slices through tetrahedra
  - Texture lookup instead of linear interpolation
  - Support of piecewise linear transfer functions

Chromaticity Reconstruction

Approximation of chromaticity

- Corresponding portion of the ray integral

$$C(S_f, S_b, l) = \int_0^l e^{-\int_0^t r(S_l(u))\,\mathrm{d}u}\,\boldsymbol{k}(S_l(t))\,\boldsymbol{r}(S_l(t))\,\mathrm{d}t$$

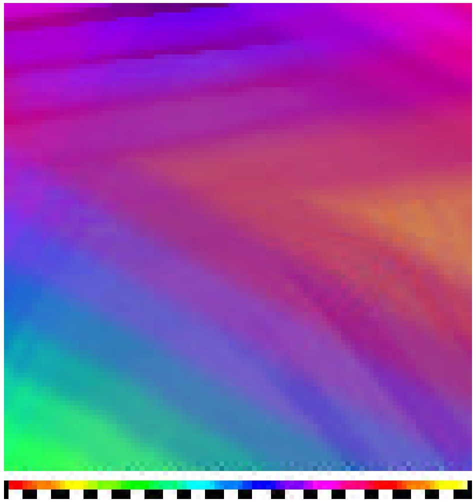
- Improvement by Roettger et al. ‘00
  - 3D texture for chromaticity and opacity
  - Slow update of transfer function
  - High memory requirements of 3D textures
  - Accurate only for small tetrahedra due to limited resolution of pre-integration table

Chromaticity Reconstruction

Approximation of chromaticity

- Corresponding portion of the ray integral

$$C(S_f, S_b, l) = \int_0^l e^{-\int_0^t r(S_l(u))\mathrm{d}u} \boldsymbol{k}(S_l(t))\, \boldsymbol{r}(S_l(t))\mathrm{d}t$$



- Different approach
  - Higher order polynomials in *l*
  - Number of triangles equal to PT
  - Only 4 slices for cubic polynomials
    - –Higher resolution table ⇒ high image quality
    - –Faster update of transfer function

$$C(S_f, S_b, l) \approx \left(1 - e^{-\int_0^l r(S_l(t))\mathrm{d}t}\right) \sum_{i=0}^{n} l^i C_i(S_f, S_b)$$

Opacity Reconstruction

Approximation of chromaticity (GeForce 4)

- Texture setup (B and A swapped for unit 0)

unit	coordinates	RGB	A
0	S_f, S_b	$C_2(S_f, S_b)$	$\int_0^1 r(S_l(t))\mathrm{d}t$
1	S_f, S_b	$C_1(S_f, S_b)$	-
2	S_f, S_b	$C_0(S_f, S_b)$	-
3	$0,0,l$	-	$1-e^{-lr}$

- Additionally store *l* in primary color alpha
- Distribution of duplicate values via vertex shader

Chromaticity Reconstruction

Approximation of chromaticity (GeForce 4)

- Pixel shader

```
ps.1.3
def     c0, 1, 1, 0, 0
tex     t0                   // load chromaticity and density
tex     t1
tex     t2
texdp3  t3, t0               // dependent lookup
lrp     r0.rgb, c0, t0, t0.a // extract chromaticity
mad     r0.rgb, v0.a, r0, t1 // calculate polynomial...
mad     r0.rgb, v0.a, r0, t2
+mov    r0.a, t1.a           // and get alpha for final color
```

Opacity Reconstruction

Approximation of chromaticity (Radeon 8500)

- Texture setup

unit	coordinates	RGB	A
0	S_f, S_b	$C_4(S_f, S_b)$	$\int_0^1 \boldsymbol{r}(S_l(t))\mathrm{d}t$
1	S_f, S_b	$C_3(S_f, S_b)$	-
2	S_f, S_b	$C_2(S_f, S_b)$	-
3	S_f, S_b	$C_1(S_f, S_b)$	-
4	S_f, S_b	$C_0(S_f, S_b)$	-
5	$0,0,l$	-	$1-e^{-lr}$

- Additionally store *l* in primary color alpha

Chromaticity Reconstruction

Approximation of chromaticity (Radeon 8500)

- Pixel shader

```
ps.1.4
texld   r0, t0               // load chromaticity and density
texcrd  r5, t5               // pass l into register
mul     r5, r0.a, r5.b       // multiply density and l
phase
texld   r1, t1               // load other coefficients
texld   r2, t2
texld   r3, t3
texld   r4, t4
texld   r5, r5               // dependent lookup
mad     r0.rgb, v0.a, r0, r1 // calculate polynomial...
mad     r0.rgb, v0.a, r0, r2
mad     r0.rgb, v0.a, r0, r3
mad     r0.rgb, v0.a, r0, r4
+mov    r0.a, r1.a           // and get alpha for final color
```

Chromaticity Reconstruction

Problems of this approach

- Limited precision of textures could be a problem
$\Rightarrow$ normalize coefficients
- Additional vertex shader needed
- Optimal approximation requires a least square fit for chromaticity (infeasible)
- Part of the three-dimensional pre-integration table needs to be computed
- Interactive change of classification no longer possible with software-only calculation of approximation textures

HW Acceleration

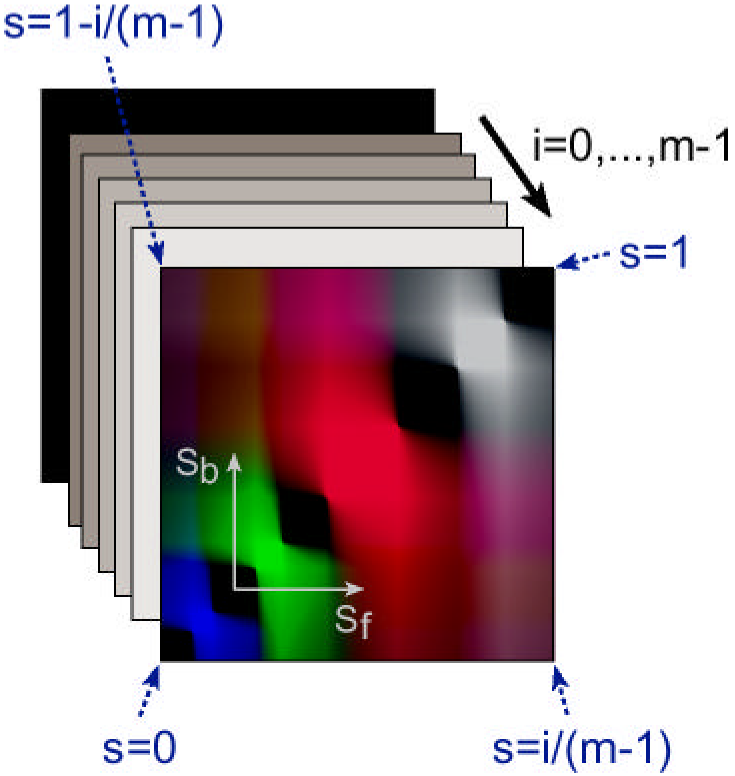
HW Accelerated Pre-Int.

Hardware accelerated pre-integration

- Use blending capabilities of graphics card
- Construct pre-integration table slice by slice (*l* constant)

Problems

- High error with few blending operations
- High error with lots of blending operations
  - Slow, due to large amount of frame buffer writes
  - Accuracy of 8 bits too low
  - No problem with new floating point hardware



HW Accelerated Pre-Int.

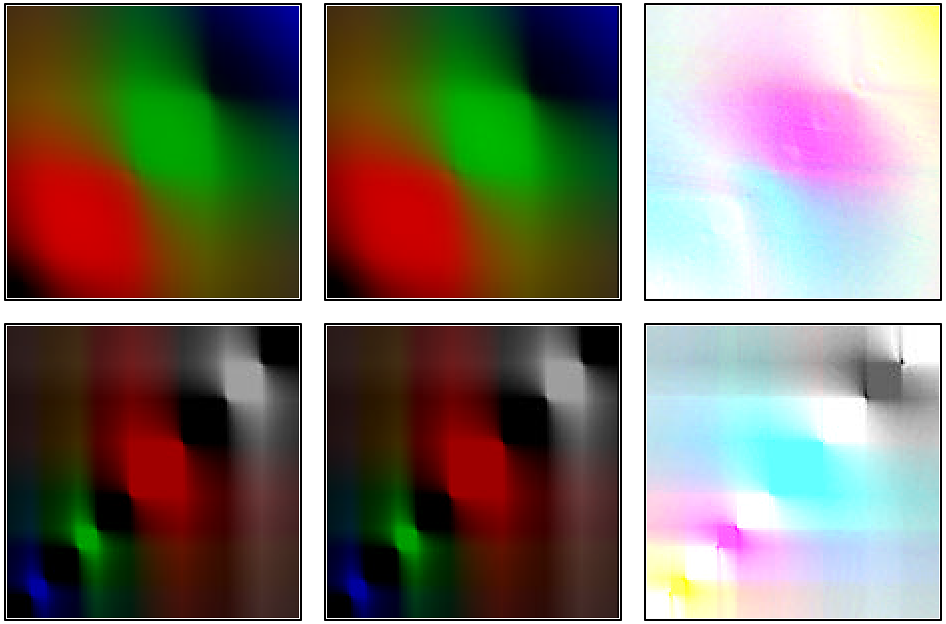
High accuracy pre-integration

- Use high internal precision of pixel shader
  - Create pre-integration table using 12-bit values
- Perform multiple blending operations at once
  - 4 blending operations in one step ⇒ speedup of approximately 2
- Store high precision values in two 8-bit values
  - Loose some instructions to combining and splitting high precision values
  - No alpha blending ⇒ ping-pong rendering
  - Separate passes for R,G and B

HW Accelerated Pre-Int.

Comparison of software and hardware pre-integration

- Speedup of about 700%
- Relatively low error



hardware software difference×8

HW Accelerated Pre-Int.

HW accelerated pre-integration (Radeon 8500)

- Pixel shader combine

```
def      c0, 0.0019608, 0, 0, 0          // 1/256
mad      r0, r0.ggaa, c0.r, r0.rrbb      // combine values
```

- Use R and B for calculations
  - Multiply result by 8 during last blending operation ⇒ faster split
- Pixel shader split

```
add_x8   r0.ga, r0_x2.rrbb, r0_x2.rrbb  // get low bits
mov_d8   r0.rb, r0.rrbb                 // get high bits
```

HW Accelerated Pre-Int.

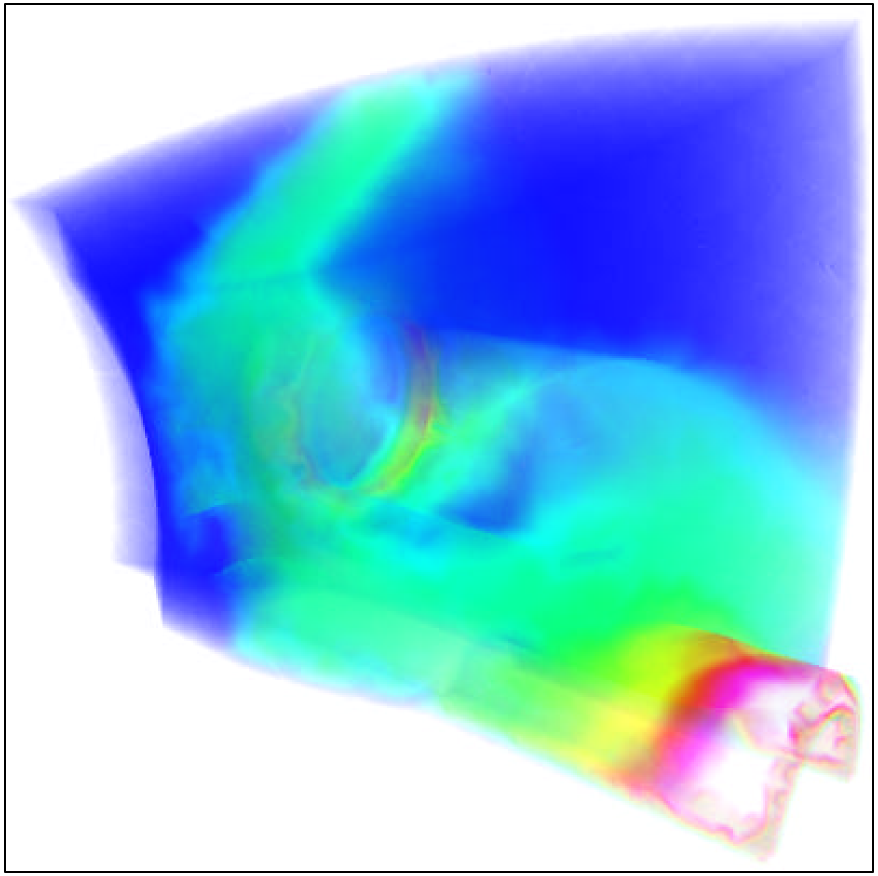
HW accelerated pre-integration (Radeon 8500)

```
ps.1.4
def     c0, 0.0019608, 0, 0, 0          // 1/256
texld   r0, t0                          // previous data
texld   r1, t1                          // 4 samples
...     ...
texld   r4, t4
mad     r0, r0.ggaa, c0.r, r0.rrbb      // combine values
mad     r1, r1.ggaa, c0.r, r1.rrbb
mad     r2, r2.ggaa, c0.r, r2.rrbb
mad     r3, r3.ggaa, c0.r, r3.rrbb
mad     r4, r4.ggaa, c0.r, r4.rrbb
phase
mad     r1.rb, r0, 1-r1.b, r1           // perform blending
mad     r2.rb, r1, 1-r2.b, r2
mad     r3.rb, r2, 1-r3.b, r3
mad_x8  r4.rb, r3, 1-r4.b, r4
add_x8  r0.ga, r4_x2.rrbb, r4_x2.rrbb  // get low bits
mov_d8  r0.rb, r4.rrbb                  // get high bits
```

Results

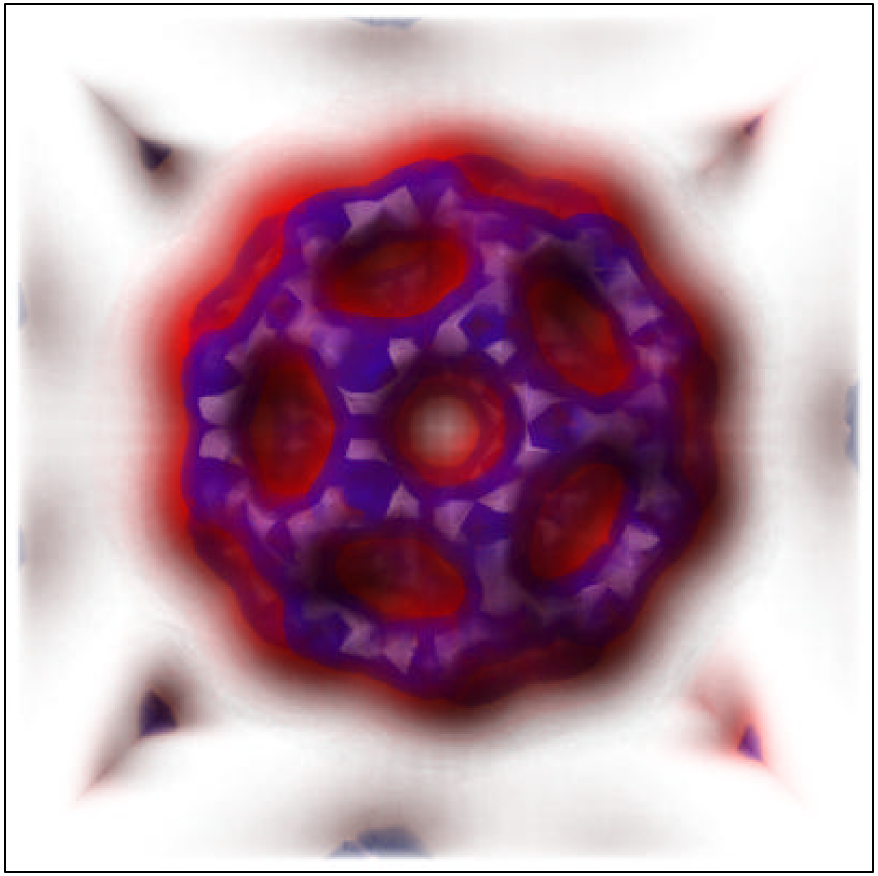
Results

Prototype (12,936 tetras)



1280×960 at 89.45 fps

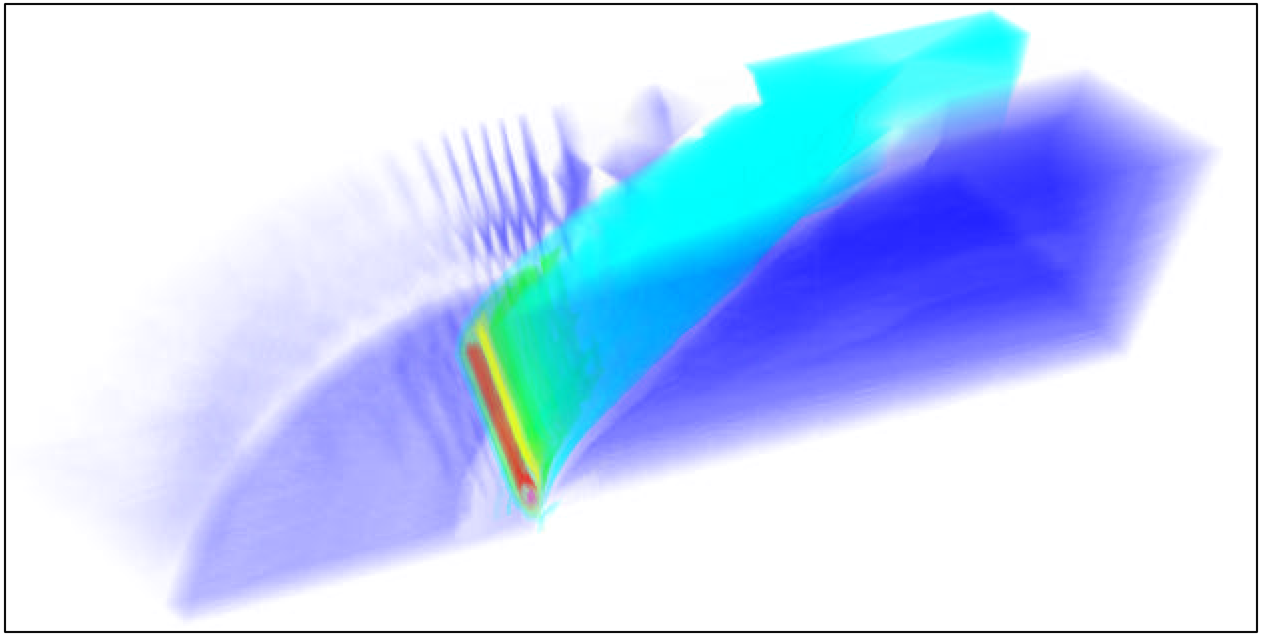
Buckyball (176,856 tetras)



1280×960 at 2.46 fps

Results

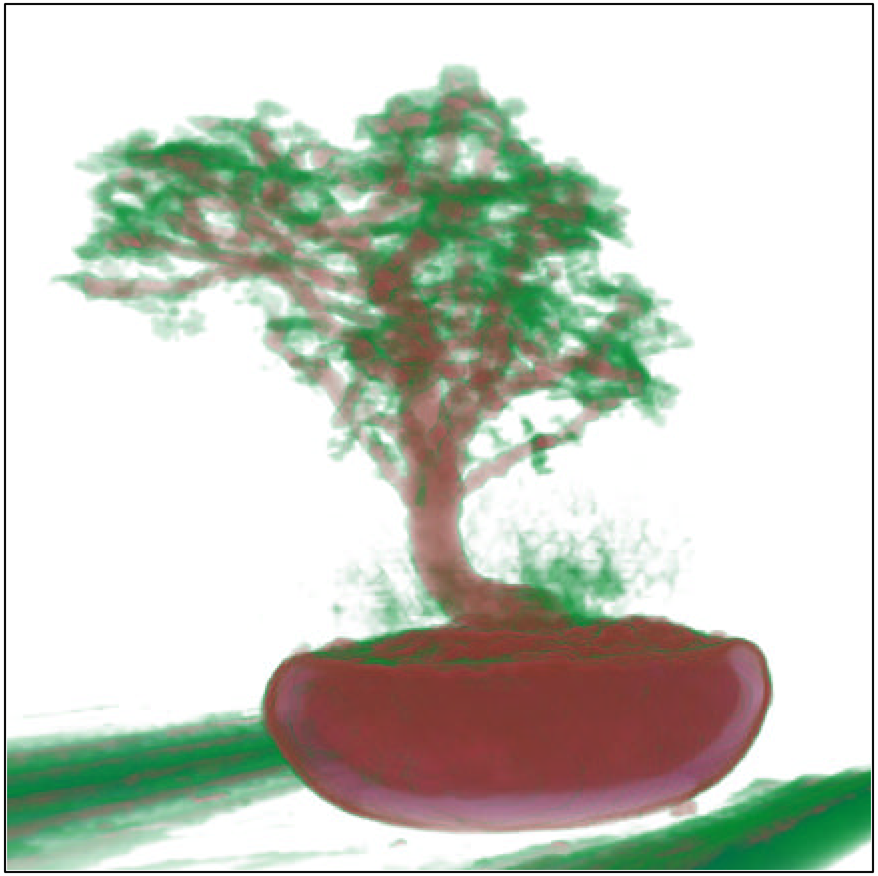
Blunt fin (187,395 tetras)



1280×960 at 3.18 fps

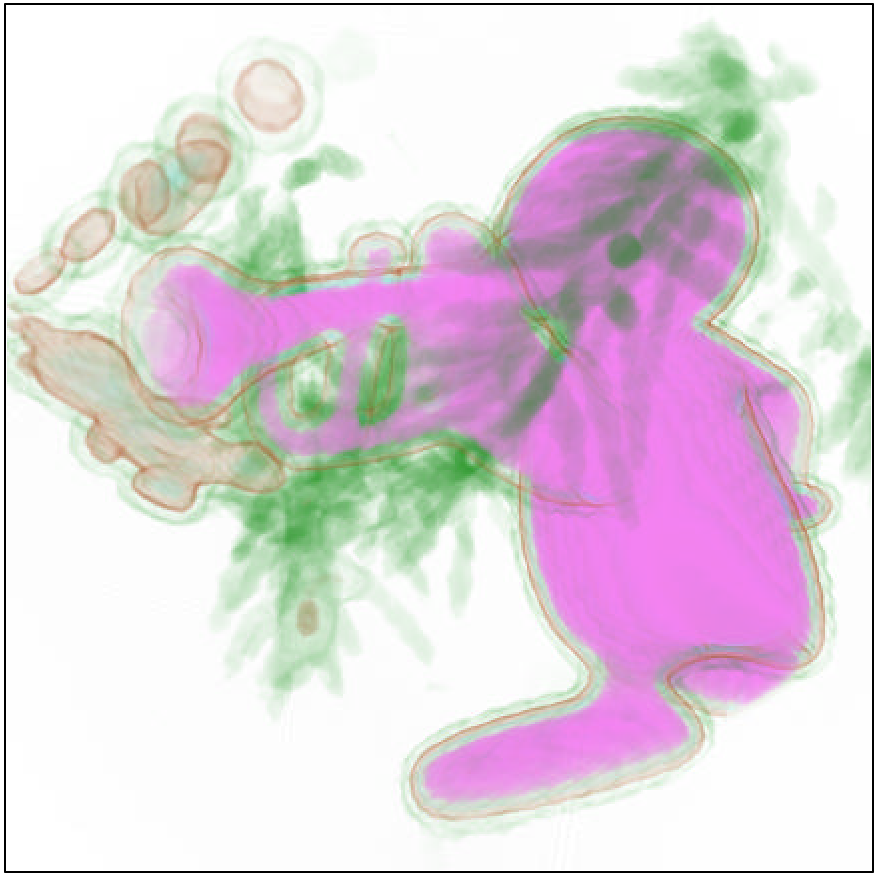
Results

Bonsai (538,937 tetras)



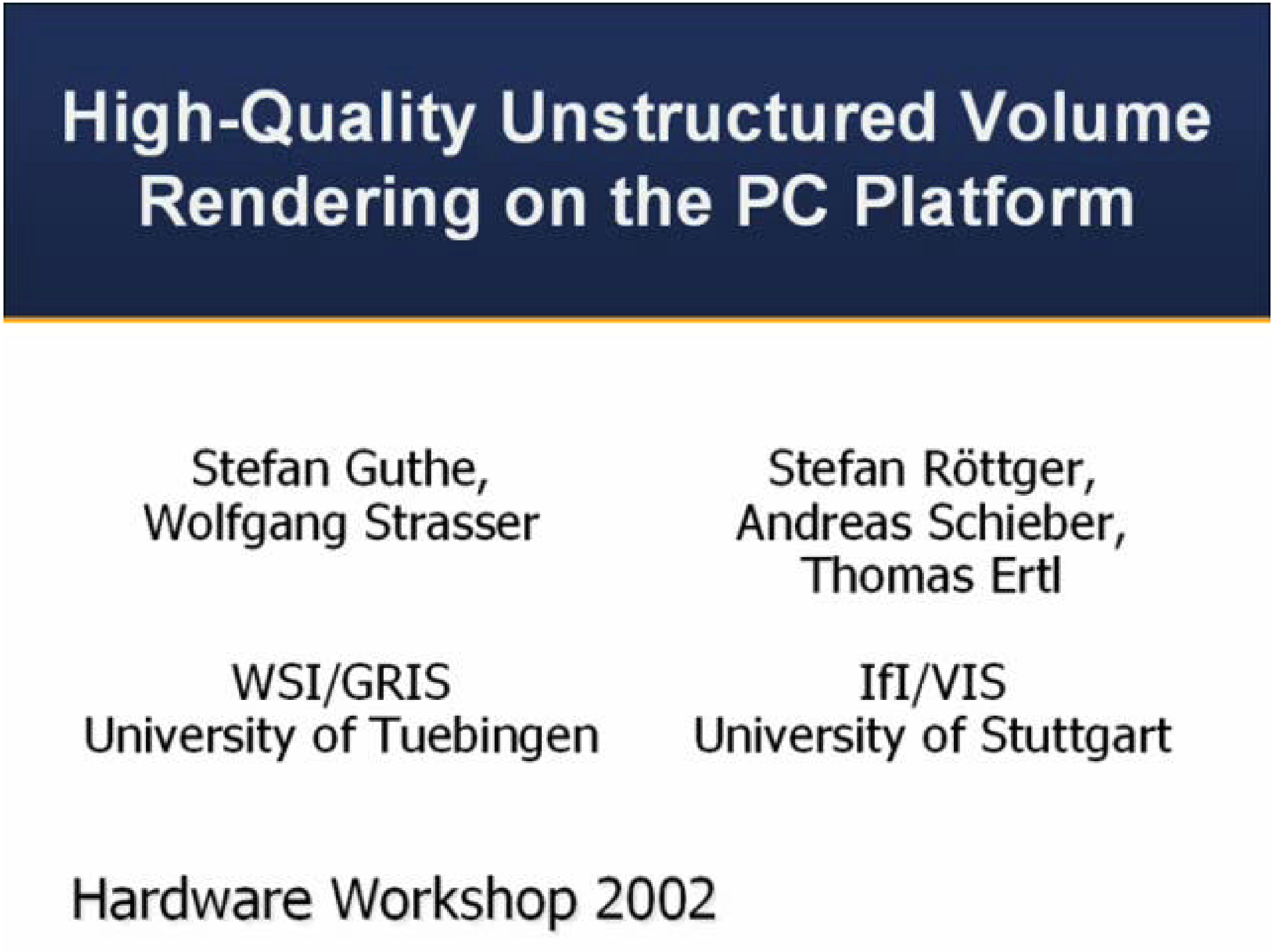
1280×960 at 1.20 fps

Trumpet (1,567,755 tetras)



1280×960 at 0.48 fps

Video



High-Quality Unstructured Volume Rendering on the PC Platform

Stefan Guthe,
Wolfgang Strasser

WSI/GRIS
University of Tuebingen

Stefan Röttger,
Andreas Schieber,
Thomas Ertl

IfI/VIS
University of Stuttgart

Hardware Workshop 2002

Conclusion

Algorithm overview

- Dependent texture for opacity
- Polynomial approximation of chromaticity
  - High resolution pre-integration table
  - High quality rendering
  - 3 slices for quadratic approximation
  - 5 slices for fourth order approximation
  - Fast update whenever transfer function changes
- Number of triangles equal to original PT algorithm
  - Fast rendering

Conclusion

Migration to new graphics hardware

- HW pre-integration
  - Floating point improves accuracy
  - More blending steps at once ⇒ even more performance gain
- Image quality will be improved by floating point frame buffer precision
- No dependent texture lookup due to exp-function in pixel shader

Questions?