GPU-Assisted High Quality Particle Rendering

Deukhyun Cha[†], Sungjin Son[‡], and Insung Ihm[†]

[†]Department of Computer Science and Engineering, Sogang University, Korea

[‡]System Architecture Lab., Samsung Electronics, Korea

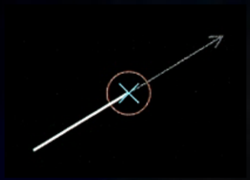
Contents

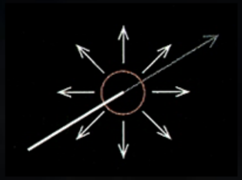
- Backgrounds
- Our contributions
- Previous work
- GPU-assisted rendering pipeline
- Adaptive density volume generation
- Accurate volume photon tracing
- Efficient Radiance Estimation with Illumination Cache
- Ray marching with Perlin noise
- Experimental results
- Conclusion and future work

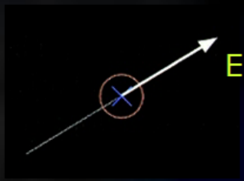
Backgrounds

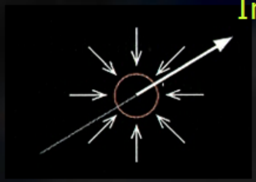
- Numerical simulation of complicated light transport phenomena

$$
\underbrace{L(x,\vec{w})}_{\text{Outgoing radiance}} = \underbrace{L(x+s\vec{w},\vec{w})}_{\text{Incoming radiance}}\, \underbrace{e^{-\int_0^s \sigma_t(x+s'\vec{w})ds'}}_{\text{Extinction due to absorption and out-scattering}}
+ \underbrace{\int_0^s L_e(x+s'\vec{w})\, e^{-\int_0^{s'} \sigma_t(x+t'\vec{w})dt}\, ds'}_{\text{Emission}}
+ \underbrace{\int_0^s \left\{ e^{-\int_0^{s'} \sigma_t(x+t'\vec{w})dt}\, \sigma_s(x+s'\vec{w}) \int_{\Omega_{4\pi}} p(x+s'\vec{w},\vec{w}',\vec{w}) L(x+s'\vec{w},\vec{w}')\, d\vec{w}' \right\} ds'}_{\text{In-scattering}}
$$



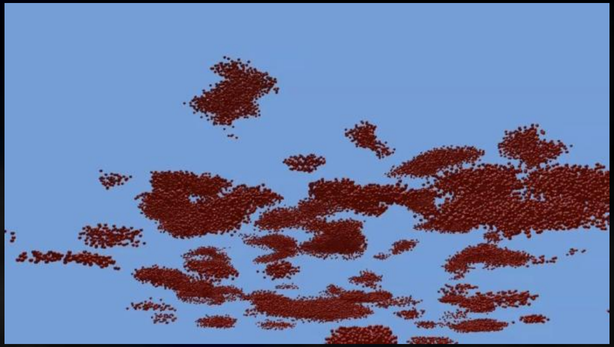


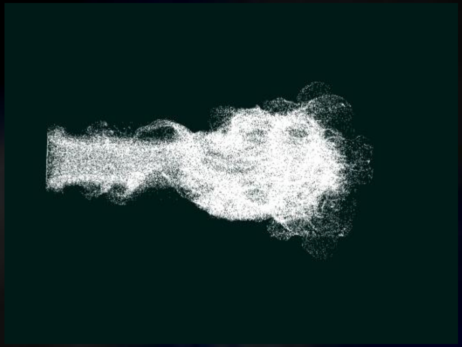




For high quality volume rendering,
the volume rendering equation must be solved as accurately as possible.

- Visualization of dynamic participating media in particle form


Modeled by animators


Physically simulated

The generation of physically accurate rendering images from large particle datasets by solving the volume rendering equation often leads to a substantial expense in computation.

Our Contributions

- Present GPU-assisted computation techniques designed for high quality particle volume rendering.

1. Present a three-pass, adaptive density estimation method for particle datasets.

2. Propose to use the mathematically correct distance generation method for volume photon tracing in nonhomogeneous participating media.

3. Exploit an illumination cache scheme for efficient estimation of in-scattered radiance.

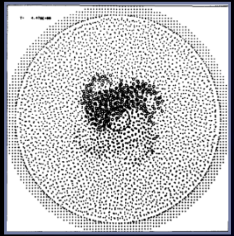
4. Apply Perlin noise in the ray marching stage for modeling fuzzy appearance of participating media.

Previous Work

- Particle based representation of participating media

W. REEVES,1983

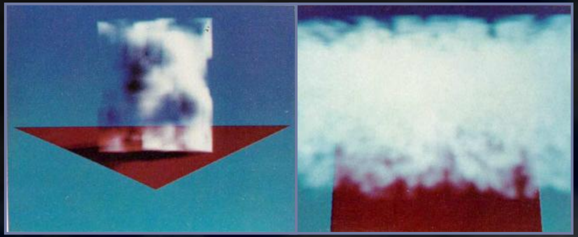
A. KAPLER, 2003



J. MONAGHAN, 1988

- Realistic rendering of participating media

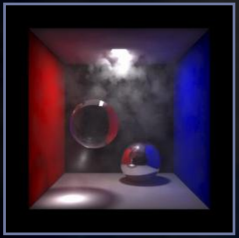
J. BLINN,1982



J. KAJIYA and B. V. HERZEN,1984

"A survey on participating media rendering techniques"

E. CEREZO, et al., 2005



H. W. JENSEN,1998

W. JAROZ, et al., 2008

F. QIU, et al., 2007

- Real-time rendering of participating media

Y. DOBASHI, et al., 2002

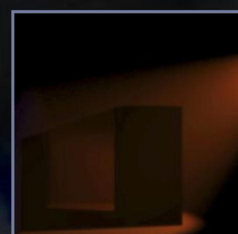
J. KNISS, et al., 2003

K. RILEY, et al., 2004

B. SUN, et al., 2005

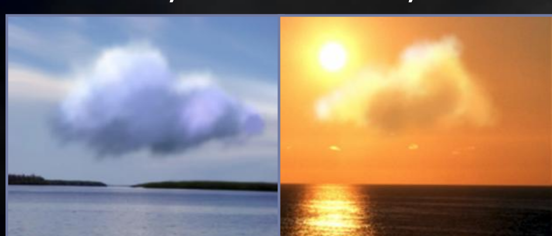
- Pre-computation of light transport information

M. HERRIS, and A. LASTRA, 2001

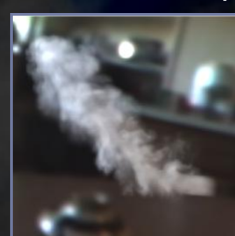


S. PREMOŽE, et al., 2002

K. HEZEMAN, et al., 2004

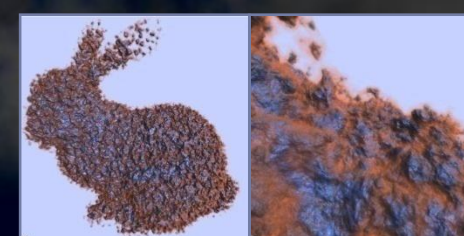


L. SZIRMAY-KALOS, et al., 2005



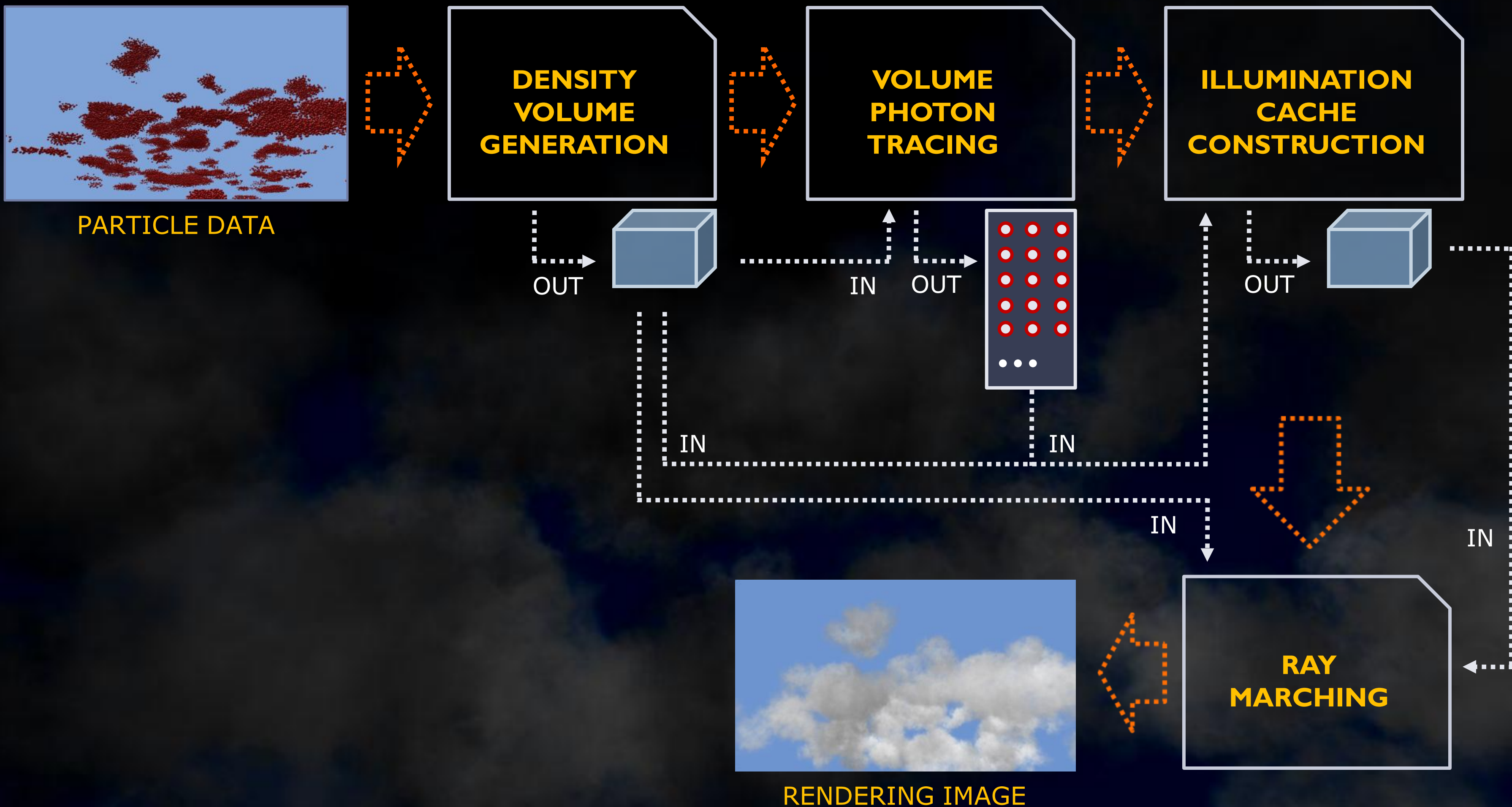
K. ZHO, et al., 2008

A. BOUTHORS, et al., 2008



C. CRASSIN, et al., 2009

Our GPU-Assisted Rendering Pipeline



Adaptive Density Volume Generation

- Problem: Build volumetric density field from input particles.
- Reconstruction of density at grid point p_{ijk} from particle distribution using kernel function (SPH)

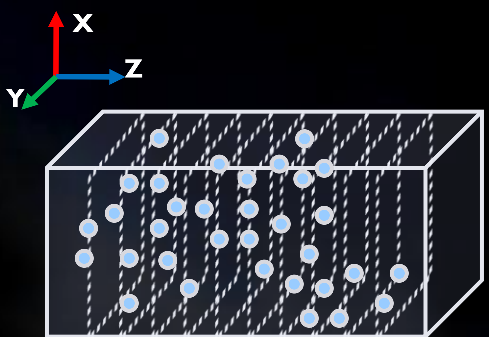
$$\rho(p_{ijk}) = \sum_q m_q W(|p_{ijk} - p_q|, h_{ijk})$$

$W(x, h)$: a smoothing kernel with smoothing length h

- GPU-assisted density estimation
  - Estimation with uniform smoothing length
  - Estimation with adaptive smoothing length
    - Local accuracy depends on the number of neighboring particles involved.

Density Estimation: Non-adaptive

- Use uniform smoothing length.

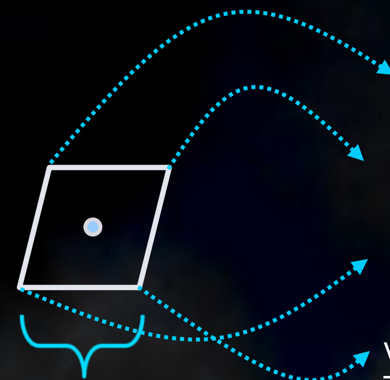


INPUT PARTICLES

➢ DENSITY VOLUME GRID (NX, NY, NZ)

KERNEL DIAMETER

➢ PARTICLE BIN (PARTICLE INDEX LIST)



KERNEL DIAMETER

➢ QUAD FOR PARTICLE KERNEL

r : KERNEL RADIUS

V[i] ← $\{P_x - r, P_y - r, P_z\}$;
T[i] ← $\{P_x, P_y\}$

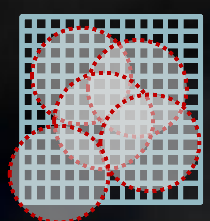
V[i+4] ← $\{P_x - r, P_y + r, P_z\}$;
T[i+2] ← $\{P_x, P_y\}$

V[i+8] ← $\{P_x + r, P_y + r, P_z\}$;
T[i+4] ← $\{P_x, P_y\}$

V[i+12] ← $\{P_x + r, P_y - r, P_z\}$;
T[i+6] ← $\{P_x, P_y\}$

```
glEnableClientState (GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, V);

glEnableClientState (GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer(2, GL_FLOAT, 0, T);
```



➢ PROJECTED KERNEL

```
for( z=0; z< NZ; z++) {
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, frameID);
  glFramebufferTexture3DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_3D, densityTex, 0, z);

  glDrawElements(GL_QUADS, nQuad, GL_UNSIGNED_INT, PAR_BIN_INDEX[z]);
}
```

➢ DRAW QUADS IN CURRENT BIN

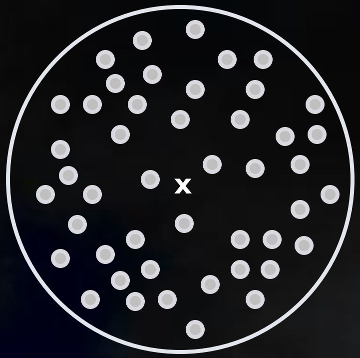
20th eurographics
SYMPOSIUM ON RENDERING
UNIVERSITAT DE GIRONA

Density Estimation: Adaptive

- Use a three-step predictor-corrector scheme for automatically choosing adaptive smoothing length.
  - Pass 1: prediction stage
  - Pass 2: correction stage
  - Pass 3: density computation stage

Pass 1: Prediction Stage

- Given a number N^*, predict a smoothing length $h^{(1)}_{ijk}$ that, hopefully, contains N^* particles in the supporting domain.



1. Quad drawing with $\boxed{h^{(0)}}$

$N^{(0)}_{ijk}$

$$\boxed{h^{(1)}_{ijk} = \min\{h^{(0)}(\frac{N^*}{N^{(0)}_{ijk}})^{\frac{1}{3}}, h^{(0)}\}}$$

Prediction

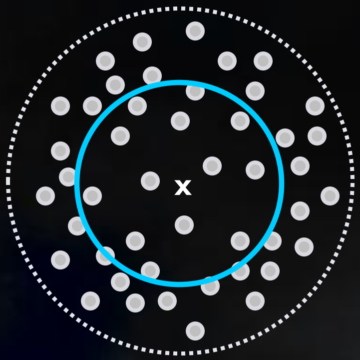
$$\bar{\rho}_{ijk} = \frac{mN^{(0)}_{ijk}}{\frac{4}{3}\pi(\kappa h^{(0)})^3} = \frac{mN^*}{\frac{4}{3}\pi(\kappa h^{(1)}_{ijk})^3}$$

Pass 2: Correction Stage

- Count the number of particles $N^{(1)}_{ijk}$ found with $h^{(1)}_{ijk}$, and compute a new length h^*_{ijk} corrected according to the particle distribution.
- Case 1: underestimation ($N^{(1)}_{ijk} < N^*$)

Underestimation

$N^* > N^{(1)}_{ijk}$



1. Quad drawing with $h^{(0)}$

$N^{(0)}_{ijk}$

$$h^{(1)}_{ijk} = \min\{h^{(0)}\left(\frac{N^*}{N^{(0)}_{ijk}}\right)^{\frac{1}{3}}, h^{(0)}\}$$

Prediction

2. Quad drawing with $h^{(1)}_{ijk}$

$N^{(1)}_{ijk}$

$$h^*_{ijk} = \left(\frac{(N^* - N^{(1)}_{ijk})((h^{(0)})^3 - (h^{(1)}_{ijk})^3)}{N^{(0)}_{ijk} - N^{(1)}_{ijk}} + (h^{(1)}_{ijk})^3\right)^{\frac{1}{3}}$$

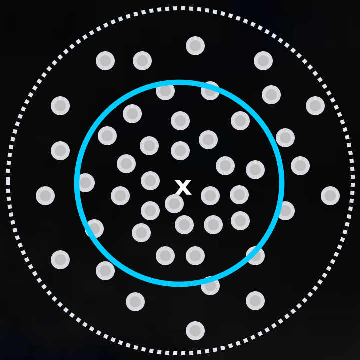
Correction

$$\bar{\rho}'_{ijk} = \frac{m(N^{(0)}_{ijk} - N^{(1)}_{ijk})}{\frac{4}{3}\pi(\kappa h^{(0)})^3 - \frac{4}{3}\pi(\kappa h^{(1)}_{ijk})^3} = \frac{m(N^* - N^{(1)}_{ijk})}{\frac{4}{3}\pi(\kappa h^*_{ijk})^3 - \frac{4}{3}\pi(\kappa h^{(1)}_{ijk})^3}$$

Case 2: overestimation ($N^{(1)}_{ijk} > N^*$)

Overestimation

$N^* < N^{(1)}_{ijk}$



1. Quad drawing with $h^{(0)}$ → $N^{(0)}_{ijk}$

$$h^{(1)}_{ijk} = \min\{h^{(0)}(\frac{N^*}{N^{(0)}_{ijk}})^{\frac{1}{3}}, h^{(0)}\}$$

Prediction

2. Quad drawing with $h^{(1)}_{ijk}$ → $N^{(1)}_{ijk}$

$$h^*_{ijk} = \alpha h^{(1)}(\frac{N^*}{N^{(1)}_{ijk}})^{\frac{1}{3}}$$

Correction

$$\bar{\rho}_{ijk} = \frac{mN^{(1)}_{ijk}}{\frac{4}{3}\pi(\kappa h^{(1)})^3} = \frac{mN^*}{\frac{4}{3}\pi(\kappa h^*_{ijk})^3}$$

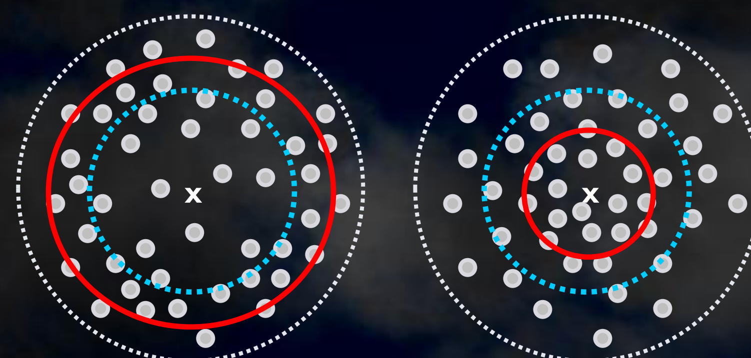
$$\alpha = (1 - s\frac{N^{(1)}_{ijk}}{N^{(0)}_{ijk}})$$

s : user defined scale factor

Pass 3: Density Computation Stage

- Estimate volume density using adaptively chosen smoothing distance h^*_{ijk}.

1. Quad drawing with $h^{(0)}$ → $N^{(0)}_{ijk}$
2. Quad drawing with $\boxed{h^{(1)}_{ijk}}$ → $N^{(1)}_{ijk}$
3. Quad drawing with $\boxed{h^{*}_{ijk}}$ → Density estimation

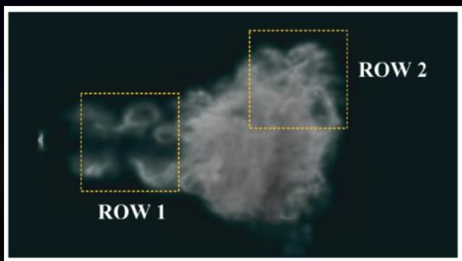


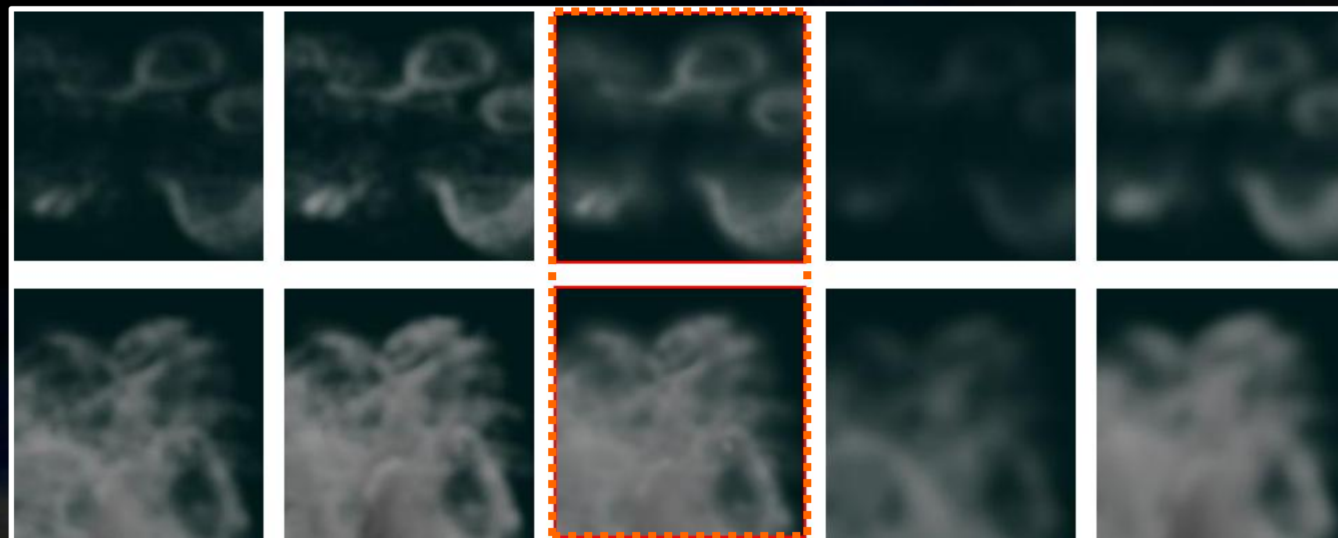
Underestimated case   Overestimated case

20th eurographics
SYMPOSIUM ON RENDERING
UNIVERSITAT DE GIRONA

Results: Uniform vs. Adaptive Smoothing Lengths

Rendering images





Uniform smoothing length

$h^{(0)} = 0.48, m = 0.004, 0.008$

Adaptive smoothing length

$h^{(0)} = 0.96, m = 0.006$

Uniform smoothing length

$h^{(0)} = 0.96, m = 0.001, 0.004$

- ✓ Bumpy and noisy
- ✓ Excessive blurring
- ✓ Smooth out noise in low density region.
- ✓ Preserve details in high density region.

Rendering animations

Uniform smoothing length

$h^{(0)} = 0.32$

Adaptive smoothing length

$h^{(0)} = 1.44$

Uniform smoothing length

$h^{(0)} = 1.44$

- ✓ Bumpy and noisy
- ✓ Smooth out noise in low density region.
- ✓ Preserve details in high density region.
- ✓ Excessive blurring

Run times

- GPU: NVIDIA GeForce GTX 280
- Particles: 326,752
- Include the CPU time for building the uniform grid and slice bins

(unit: sec.)

Resolution	$h^{(0)}$	UG	USL	ASL2	ASL3
$129\times103\times107$	0.96	1.38	0.53	0.92	1.20
$257\times206\times213$	0.96	5.48	1.84	3.55	4.66
$257\times206\times213$	1.44	10.02	4.53	7.47	9.98

UG : uniform grid, similar to [T. PURCELL et al., 2003]

USL : uniform smoothing length

ASL2 : two-step adaptive smoothing length without correction step

ASL3 : three-step adaptive smoothing length with correction step

Accuracy of smoothing length prediction

	$h^{(0)}$	N^*	N_{avg}^{used}	N_{stdev}^{used}	$\frac{N_{stdev}^{used}}{N_{avg}^{used}}$
USL	0.24	-	11.08	15.42	1.39
	0.48	-	74.37	55.50	0.74
	0.96	-	447.49	250.21	0.55
	1.44	-	1282.80	667.91	0.52
ASL2	0.96	32	61.32	30.65	0.49
		64	98.14	41.47	0.42
	1.44	32	80.21	41.25	0.51
		64	136.66	59.76	0.43
ASL3	0.96	32	33.05	10.14	0.30
		64	62.21	12.22	0.19
	1.44	32	34.65	8.51	0.24
		64	67.00	13.03	0.19

USL : uniform smoothing length

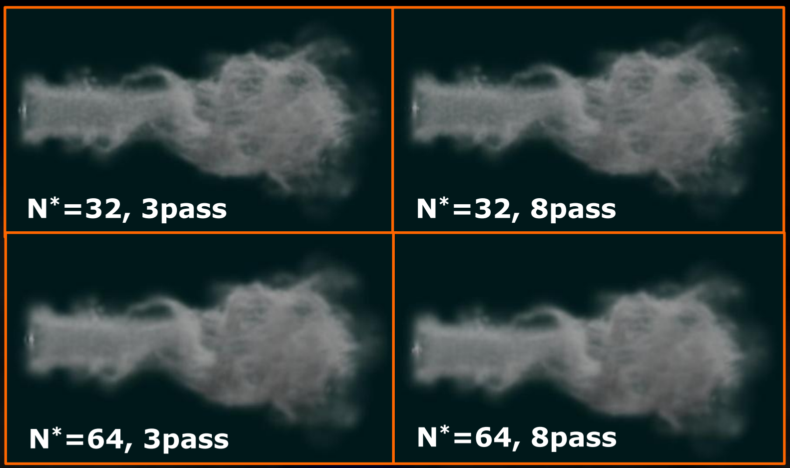
ASL2 : adaptive smoothing length without correction step (2 steps)

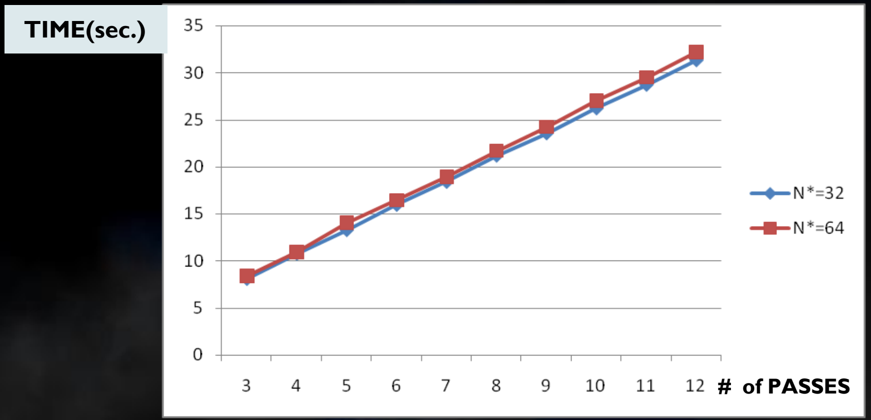
ASL3 : adaptive smoothing length with correction step (3 steps)

N_{avg}^{used} : average number of particles used for density estimation

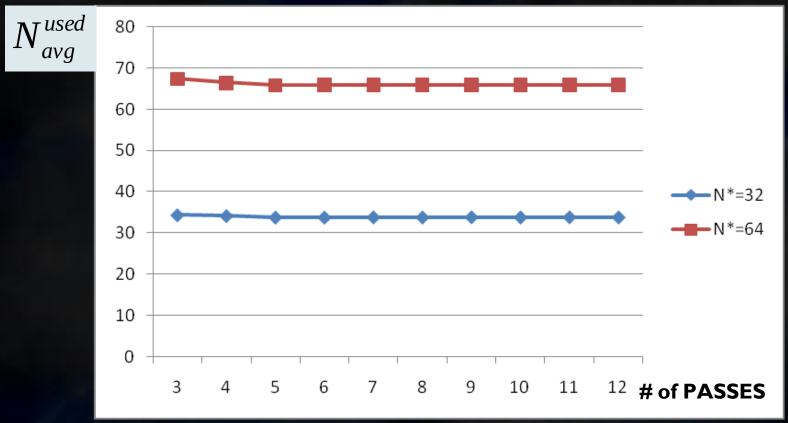
N_{stdev}^{used} : standard deviation of particles used for density estimation

More correction passes needed?

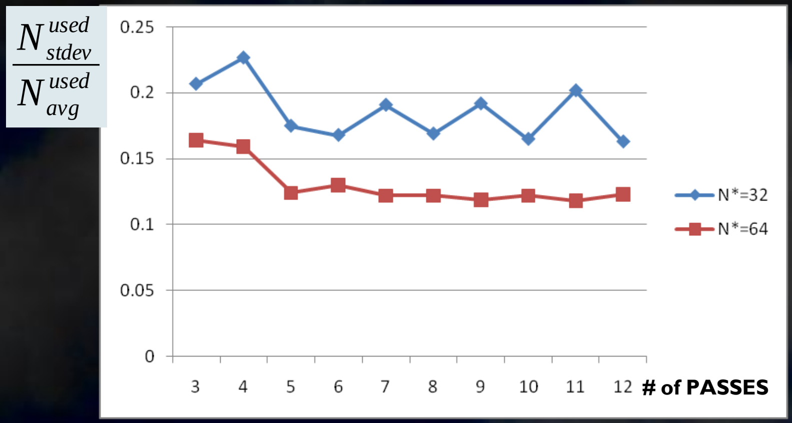




Timing statistics



# of average particles



Standard deviation ratio

Accurate Volume Photon Tracing

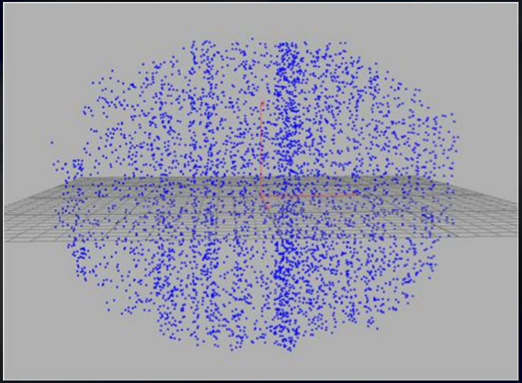
- Problem: Stochastically generate distance d_{next} to the next interaction point for effective volume photon tracing.
- Still often used method 1

$$d_{next} = -\frac{\log \xi}{\sigma_t}$$

uniform random variable $\xi \in (0,1)$

  - Does not reflect varying extinction coefficients in nonhomogeneous media, leading to erroneous simulation of light transport phenomena.

Emitted photons: 4,403,200
Stored photons: 117,757
GPU tracing time: 3.934 sec.



Photons traced

Rendering result

Still often used method 2

$$d_{next} = \min\{-\frac{\log \xi}{\sigma_t}, d_{\max}\}$$

Bound d_{next} by a maximum distance d_{max}.

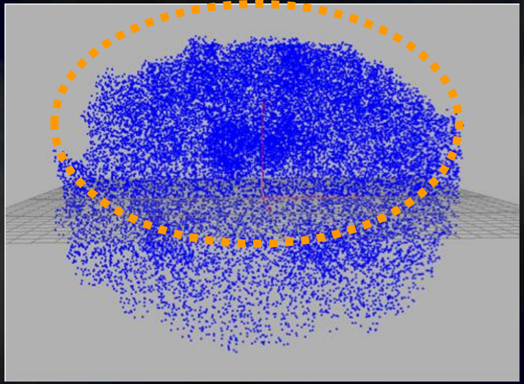
$$d_{next} = -\frac{\log \xi}{\alpha \sigma_t}$$

Scale up the extinction coefficient.

- May avoid an excessive overestimation, but still does not reflect the distribution of extinction coefficients in nonhomogeneous media.

Scale factor $\alpha = 50$

Emitted photons: 409,600
Stored photons: 416,219
GPU tracing time: 1.359 sec.



Photons traced

Rendering result

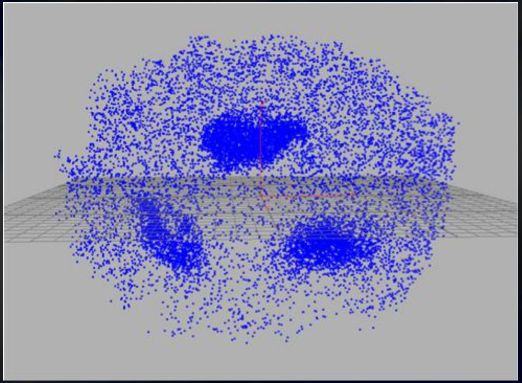
Mathematically correct method

- Find d_{next} that satisfies the following integral equation:

$$\int_0^{d_{next}} \sigma_t(s)ds = -\ln(1-\xi)$$

- Can be solved numerically by incrementally evaluating the integral until the sum exceeds the right term value.
- The current GPU allows to solve the equation in practical time!

Emitted photons: 614,400
Stored photons: 191,636
GPU tracing time: 1.038 sec.



Photons traced

Rendering result

Results: Incorrect vs. Correct Distance Generation

Still often used method 1 | Still often used method 2(scale factor 50) | Mathematically correct method

Emitted photons: 4,403,200
Stored photons: 117,757
Tracing time on GPU: 3.934 sec.

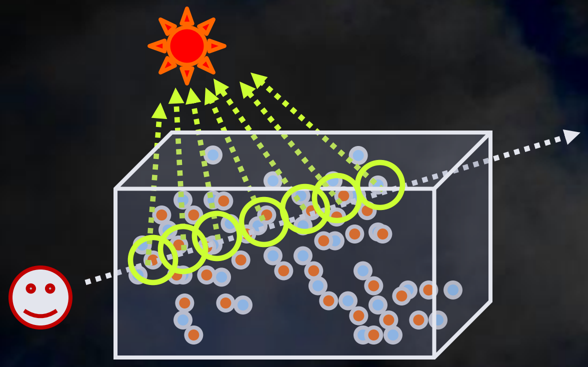
Emitted photons: 409,600
Stored photons: 416,219
Tracing time on GPU: 1.359 sec.

Emitted photons: 614,400
Stored photons: 191,636
Tracing time on GPU: 1.038 sec.

Efficient Radiance Estimation with Illumination Cache

- Problem: Estimate in-scattered radiance at each ray sample point during the ray marching stage.
  - Most of the rendering time is spent in computing incoming radiance due to single and multiple scattering on the fly during ray marching.
  - A radiance caching scheme was presented for volume rendering by Jarosz et al. (2008), but is not well-suited for GPU implementation.

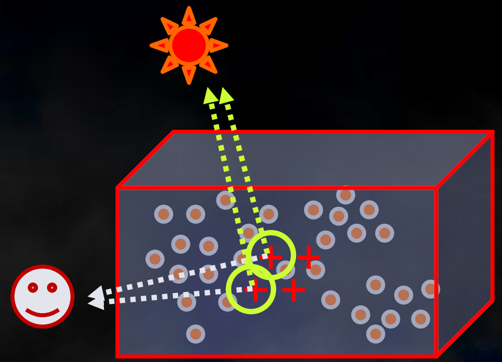
- Brute force ray marching without caching



✓ Density volume generation
✓ Volume photon map generation
✓ Ray marching
✓ Single scattering
✓ Multiple scattering (volume photon gathering)

Efficient ray marching with illumination cache

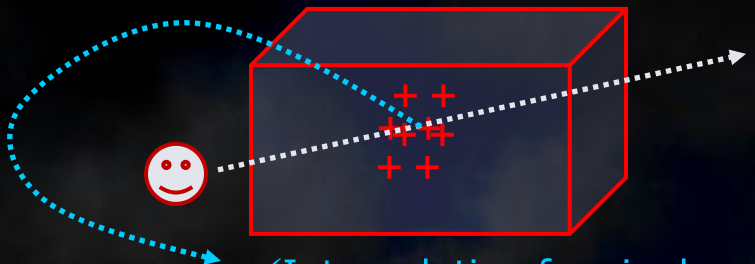
① Illumination cache generation



✓ Illumination cache grid

- ✓ Single-scattered radiance at grid points
- ✓ Multiple-scattered radiance at grid points
  (use quad-drawing based volume photon gathering)

② Ray marching using illumination cache



✓Interpolation for single-
& multiple-scattered radiance

20th eurographics
SYMPOSIUM ON RENDERING
UNIVERSITAT DE GIRONA

Results: With vs. Without Illumination Cache

- Run times: *generation of illumination cache*
  - CPU: 3.16GHz Intel Core 2 Duo, GPU: NVIDIA GeForce GTX 280
  - Particles: 500,000(Cloud1), 50,000(Cloud2), 326,752(Smoke)
  - The tested CPU version also implements our illumination cache technique.

Cloud 1

Cloud 2

Smoke

	Resolution	CPU	GPU	
Cloud1	189×23×200	4.5	0.012	375x
	283×35×300	16.4	0.032	512x
Cloud2	200×107×200	38.6	0.119	324x
	299×160×300	165.5	0.502	329x
Smoke	200×161×167	55.3	0.136	406x
	300×242×250	257.3	0.630	408x

Accumulation of single-scattered radiance

	Resolution	CPU	GPU	
Cloud1	189×23×200 (2.8)	19.2 (165,087)	0.98 (195,060)	19x
	283×35×300 (2.8)	62.2	1.84	33x
	283×35×300 (5.6)	422.6	9.90	42x
Cloud2	200×107×200 (2.6)	121.3 (190.067)	3.01 (231,505)	40x
	299×160×300 (2.6)	394.1	9.25	42x
	283×35×300 (5.2)	2988.5	53.09	56x
Smoke	200×161×167 (0.8)	17.5 (130.560)	0.94 (141,828)	18x
	300×242×250 (0.8)	56.1	2.08	26x
	300×242×250 (1.6)	417.0	6.85	60x

Accumulation of multiple-scattered radiance

- Run times: *ray marching with illumination cache*

Cloud 1

Cloud 2

Smoke

	Resolution	CPU	GPU	
Cloud1	1,024×576 (1)	296.1 (380.1)	0.14 (4.66)	2115x
	2,048×1,152 (1)	498.3 (586.9)	0.83 (5.89)	600x
	2,048×1,152 (4)	1,962.6 (2,043.7)	2.72 (8.01)	721x
Cloud2	1,024×576 (1)	356.6 (863.3)	0.21 (12.01)	1698x
	2,048×1,152 (1)	1,409.2 (1,985.4)	1.36 (14.06)	1036x
	2,048×1,152 (4)	7,605.6 (8,001.8)	4.47 (16.84)	1701x
Smoke	1,024×768 (1)	290.0 (506.9)	0.19 (5.59)	1526x
	2,048×1,536 (1)	1,118.0 (1,455.9)	1.77 (7.56)	631x
	2,048×1,536 (4)	6,410.5 (6,663.6)	4.38 (10.19)	1463x

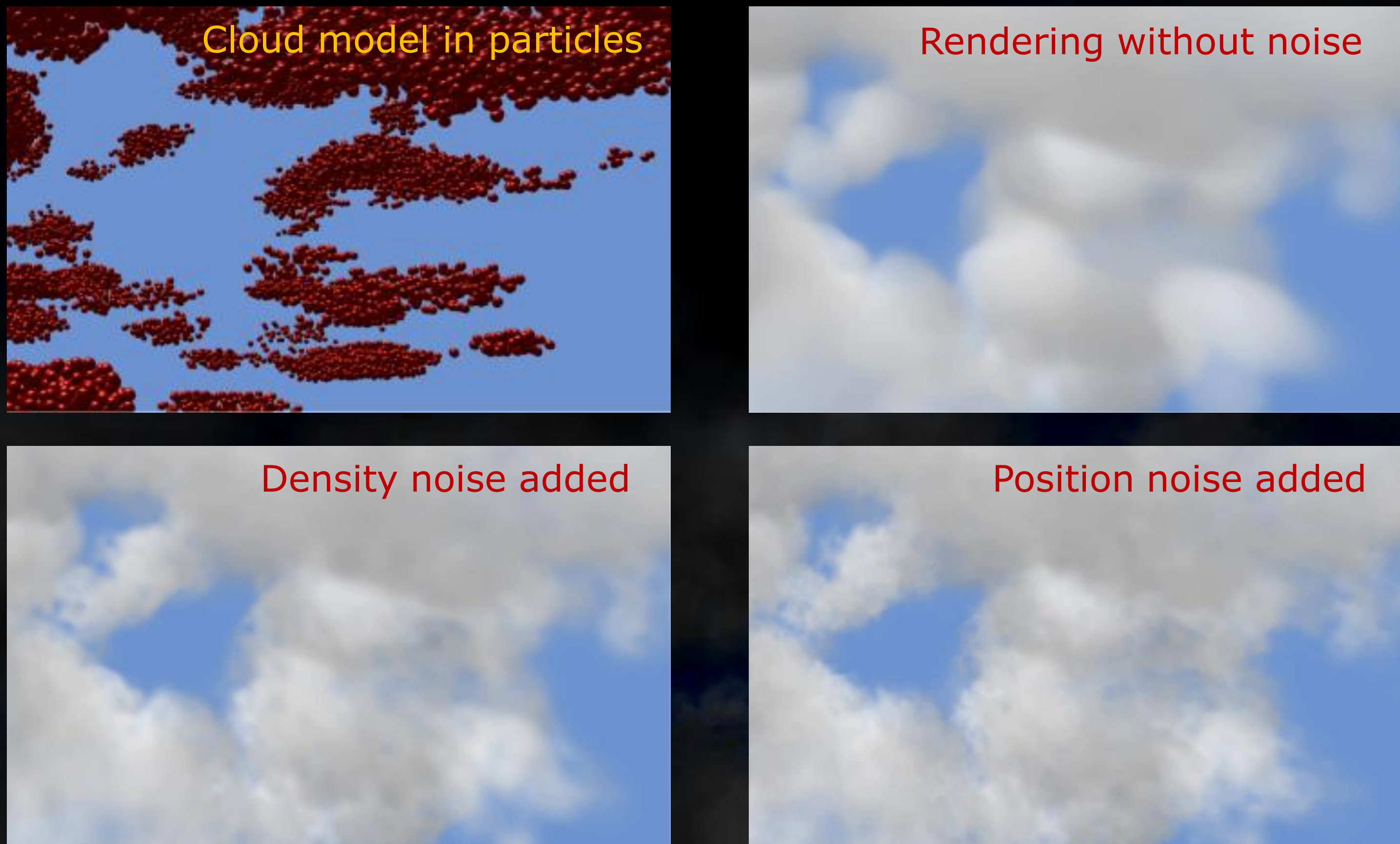
- Note: The rendering times by a CPU version without an illumination cache were prohibitive!

Ray Marching with Perlin Noise

- Problem: Exploit procedural noise for producing natural fuzzy effects during ray marching.
- Particle data that animators routinely generate often do not contain much details.
- Slightly dispersing ray sample points with Perlin noise is a good way of producing fuzzy effects.

- Our strategy
  - Density noise: apply scalar noise to disperse estimated densities after density estimation.
  - Position noise: apply vector noise to disperse ray sample points during ray marching.

Results: Before and After Adding Noises

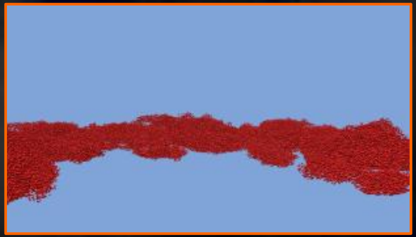


- The extra cost is very cheap as pre-computed noises are stored in four-component 3D texture for a fast noise application.

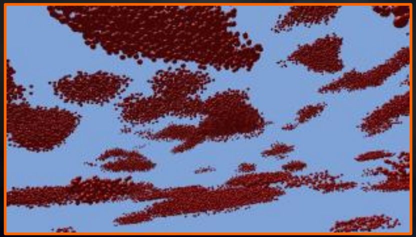
Experimental Results

- Experiment environment
  - CPU: 3.16GHz Intel Core 2 Duo, GPU: NVIDIA GeForce GTX 280
  - APIs: OpenGL and Cg
  - Image and grid resolutions

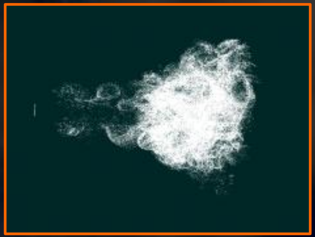
	Cloud1	Cloud2	Smoke
Image size	1024×576	1024×576	1024×768
	2048×1152	2048×1152	2048×1536
Particles	500,000	50,000	326,752
Grid size	$339 \times 42 \times 361$	$299 \times 160 \times 301$	$280 \times 255 \times 233$



Cloud 1

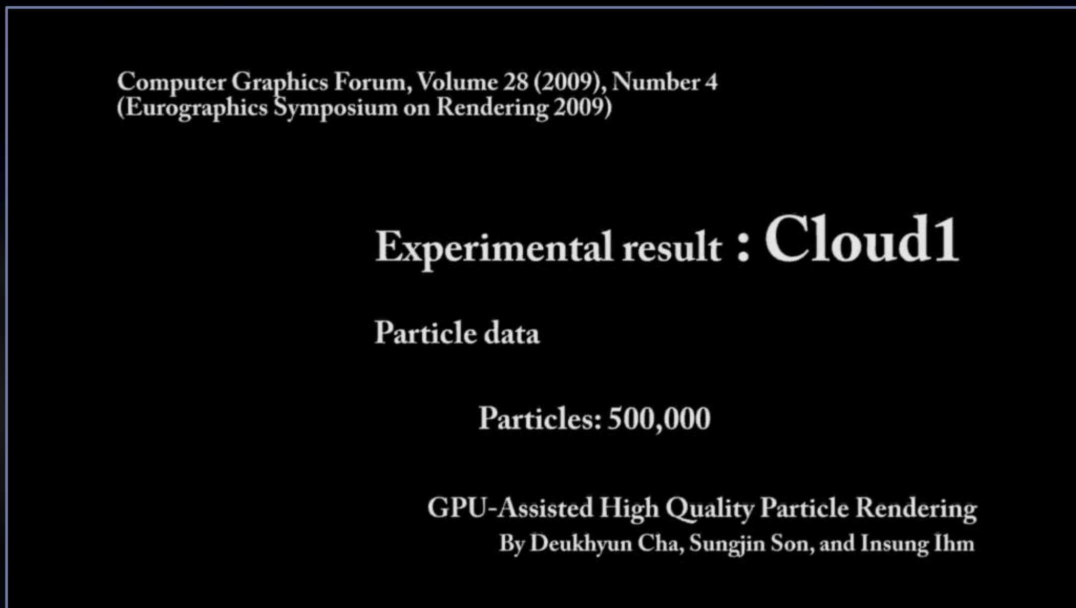


Cloud 2



Smoke

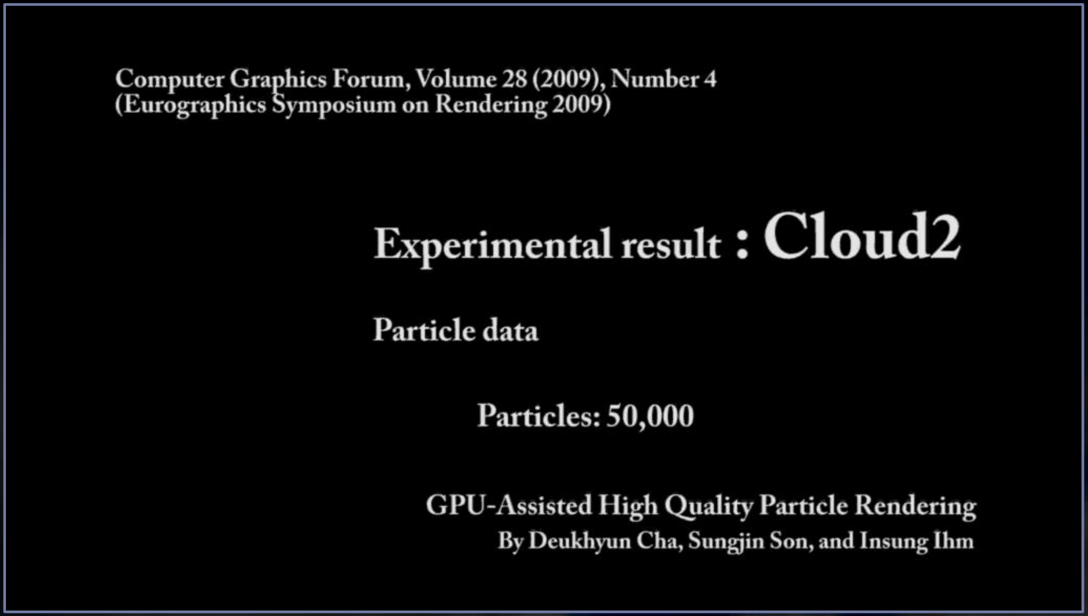
Rendering results



Average rendering time: 5.04(s)
Image resolution: 1024x576

Used rendering features:
✓ Adaptive density estimation, accurate photon tracing, Illumination cache, position noise in ray marching

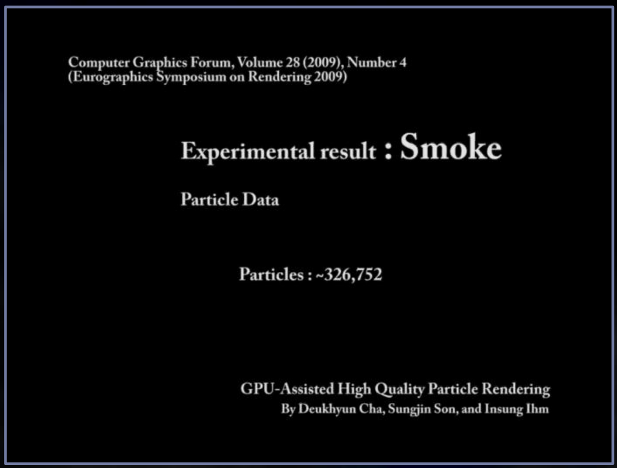
Rendering results



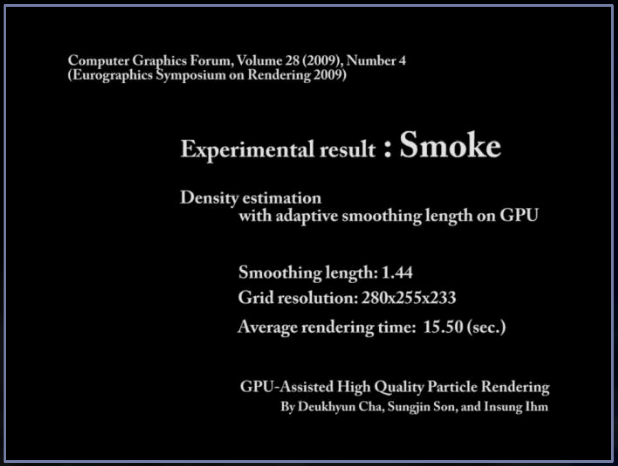
Average rendering time: 10.27(s)
Image resolution: 1024x576

Used rendering features:
✓ Accurate photon tracing, Illumination cache, position noise in ray marching

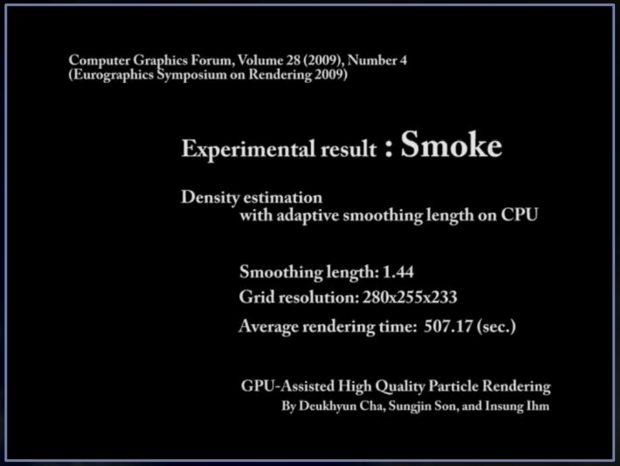
Rendering results



PARTICLE DATA



GPU RENDERING



CPU RENDERING

Physically simulated particle data

Maximum # of particles: 326,752

Average rendering time: 15.50(s)
Image resolution: 1024x768

Used rendering features:
✓ Adaptive density estimation
✓ Accurate photon tracing
✓ Illumination cache

Average rendering time: 507.17(s)
Image resolution: 1024x768

Used rendering features:
✓ Adaptive density estimation(kd-tree)
✓ Accurate photon tracing
✓ Illumination cache

Overall timing performance

		DVG	VPT	ICC	RM	
Cloud1 $(h^{(0)} = 1.07)$	CPU	15.37	1.40	65.81	43.26	164.58
	GPU	1.90	0.79	2.22	0.55	2.16
Cloud2 $(h^{(0)} = 1.6)$	CPU	13.97	2.05	307.69	168.21	585.52
	GPU	0.55	0.74	8.29	0.66	2.54
Smoke $(h^{(0)} = 1.44)$	CPU	64.17	16.59	175.76	191.77	517.39
	GPU	10.56	1.39	3.90	0.74	2.94

DVG : density volume generation
VPT : volume photon tracing
ICC : illumination cache construction
RM : ray marching [1K | 2K]

Cloud 1

Cloud 2

Smoke

- Timing performance at multiple volume resolutions

	DVG	VPT	ICC	RM
$100 \times 54 \times 101$ (a)	0.11	0.54	0.69	0.54
$150 \times 80 \times 151$	0.15	0.51	1.21	0.54
$200 \times 107 \times 201$ (b)	0.23	0.49	2.60	0.58
$249 \times 133 \times 251$	0.35	0.53	4.67	0.62
$299 \times 160 \times 301$ (c)	0.55	0.74	8.29	0.66

DVG : density volume generation
VPT : volume photon tracing
ICC : illumination cache construction
RM : ray marching

(a)

(b)

(c)

Conclusion and Future Work

- Presented a GPU-assisted particle rendering schemes.
  1. Presented a three-pass, adaptive density estimation method for particle datasets.
  2. Proposed to use the mathematically correct distance generation method for volume photon tracing in nonhomogeneous participating media.
  3. Exploited an illumination cache scheme for efficient estimation of in-scattered radiance.
  4. Applied Perlin noise in ray marching stage for modeling fuzzy appearance of participating media.

- Currently, we are extending our renderer to include light emission phenomena of hot gaseous fluids like fire, flame, and explosion.

Thank you!

- http://grmanet.sogang.ac.kr

Results: Before and After Adding Noises

- Additional results

Ray marching without noise

Image resolution: 2048x1152
Density grid: 128x68x129
Ray marching time: 2.0639(s)

Ray marching with noise

Image resolution: 2048x1152
Density grid: 128x68x129
Ray marching time: 2.0675(s)

Entire image