hics & (depth < ≥0000

t = inside 7 1 1 1 0 ht = nt / nc, ddn 0 bs2t = 1.0f - n∺t 0 D, N); ∂)

at a = nt - nc, b = nt - r at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (dd)

= * diffuse = true;

. efl + refr)) && (depth < MAXDEPTI

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed ff; radiance = SampleLight(&rand, I,) water addiance = SampleLight(&rand, I)

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvit at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sec /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 8pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true: $\epsilon(x,x')$

INFOMAGR – Advanced Graphics

Jacco Bikker - November 2021 - February 2022

Lecture 7 - "GPU Ray Tracing (1)"

f(x',x'')dx''

Welcome!



tics & (depth < Modean

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0 Tr) R = (D ⁺ nnt - N - (ddn

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTI

), N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && (dot)

v = true; at brdfPdf

at brdfPdf = EvaluateDiffuse(L, N) * Psurvio at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (PS

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Exam Questions: Sampler
- Introduction
- Survey: GPU Ray Tracing
- Practical Perspective



Exam Questions

tics & (depth < Notion

: = inside ? 1 1 1 1 ht = nt / nc, ddn bs2t = 1.0f - nnt * 1 D, N); D)

at a = nt - nc, b = nt + nc at Tr = 1 - (R0 + (1 - R0 Ir) R = (D = nnt - N - 000

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffu: = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &Light) e.x + radiance.y + radiance.z) > 0) & closed e.x + radiance.y + radiance.z) > 0) & closed

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (c);

andom walk - done properly, closely following Ser /ive)

We use the Surface Area Heuristic to determine a good position for a split plane during BVH construction.

a) One version of the SAH looks as follows:

 $C_{split} = C_T + A_{left}N_{left}C_I + A_{right}N_{right}C_I$ What are C_T and C_I for? How would you modify this formula if your BVH supports spheres and tori?

- b) Explain why we use surface area (rather than e.g. bounding box volume) in the cost function.
- c) The Surface Area Heuristic is a 'greedy' heuristic. What is the meaning of 'greedy' in this context?
- d) What is the algorithmic complexity of the greedy SAH-guided BVH construction algorithm (without binning), and what would be the algorithmic complexity of the non-greedy version?



Exam Questions

nics & (depth < MaxDann

: = inside ? 1 1 1 1 ht = nt / nc, ddn 1 ps2t = 1.0f - nmt 1 2, N); 2)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Ir) R = (D = nnt - N

= * diffuse = true;

efl + refr)) && (depth < MOXDEDII

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &lienco 2.x + radiance.y + radiance.z) > 0) && (closed)

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) ();

andom walk - done properly, closely following Sour /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Behold the Rendering Equation:

$$L_o(x,\omega_o) = L_E(x,\omega_o) + \int_{\Omega} f_r(x,\omega_o,\omega_i) L_i(x,\omega_i) \cos \theta_i \, d\omega_i$$

- a) What does $\cos \theta_i$ do?
- b) Why is the above formulation missing the 'visibility factor'?
- c) Another formulation of the RE is the three-point formulation:

 $L(s \leftarrow x) = L_E(s \leftarrow x) + \int_A f_r(s \leftarrow x \leftarrow x') L(x \leftarrow x') G(x \leftrightarrow x') dA(x')$ What is *A* in this equation? Explain what $G(x \leftrightarrow x')$ does.



tics & (depth < Modean

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0 Tr) R = (D ⁺ nnt - N - (ddn

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTI

), N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && (dot)

v = true; at brdfPdf

at brdfPdf = EvaluateDiffuse(L, N) * Psurvio at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (PS

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Exam Questions: Sampler
- Introduction
- Survey: GPU Ray Tracing
- Practical Perspective



Introduction

at a = nt

refl * E * diffuse;

AXDEPTH)

survive = SurvivalProbability(diff lf; radiance = SampleLight(&rand, I, e.x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse(L, at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely follow /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, & urvive; pdf; n = E * brdf * (dot(N, R) / pdf);sion = true:

Transferring Ray Tracing to the GPU

Platform characteristics:

- Massively parallel
- SIMT
- High bandwidth
- Massive compute potential
- Slow connection to host

Challenges:

 Thread state must be small Efficiency requires coherent control flow





Introduction

sics & (depth < Monosti

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁺ nnt - N - (don)

= * diffuse; = true;

-:fl + refr)) && (depth < MODEPT

D, N); refl * E * diffu: = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly, close
if;
radiance = Si

e.x + radian

w = true; at brdfPdf = at3 factor = at weight = 1 at cosTheta0 E * ((weigh

andom walk -/ive)

at3 brdf = SampleDiffuse(diffuse, N, urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Transferring Ray Tracing to the GPU

Survey

- Understand evolution of graphics hardware
- Understand characteristics of modern GPUs
- Investigate algorithms designed with these characteristics in mind







Microsoft Visual Studio

An unhandled exception of type 'Cloo.InvalidProgramExecutableComputeException' occurred in Cloo.dll

Break

Continue

Additional information: OpenCL error code detected: InvalidProgramExecutable.

Break when this exception type is thrown Break and open Exception Settings tics & (depth < Mot000

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0 Tr) R = (D ⁺ nnt - N - (ddn

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTI

), N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && (dot)

v = true; at brdfPdf

at brdfPdf = EvaluateDiffuse(L, N) * Psurvio at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (PS

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Exam Questions: Sampler
- Introduction
- Survey: GPU Ray Tracing
- Practical Perspective



2002

ics **(depth** < Model

: = inside ? 1 ht = nt / nc, ddm hs2t = 1.0f - nnt 2, N);))

ut a = nt - nc, b = nt nt Tr = 1 - (R0 + (1 - R0 ir) R = (D ⁼ nnt - N = (ddn

= * diffuse; = true;

-:fl + refr)) && (depth < Ⅳ

D, N); refl * E * diffus = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed H; radiance = SampleLight(&rand, I, &L, &L 2.x + radiance.y + radiance.z) <u>0</u> 88

v = true;

at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sa /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, urvive; pdf;

n = E * brdf * (dot(N, R) / pdf); sion = true:

Ray Tracing on Programmable Graphics Hardware*

Graphics hardware in 2002:

- Vertex and fragment shaders only
- Simple instruction sets
- Integer-only (fixed-point) fragment shaders
- Limited number of instructions per program
- Limited number of inputs and outputs
- No loops, no conditional branching

Expectations:

- Floating point fragment shaders
- Improved instruction sets
- Multiple outputs per fragment shader

*: Ray tracing on programmable graphics hardware, Purcell et al., 2002.

No branching



NVidia GeForce 3



ATi Radeon 8500



Ray Tracing on Programmable Graphics Hardware

Challenge: to map ray tracing to *stream processing**.

Stage 1: Produce a stream of primary rays.

containing geometry.

Stage 2: For each ray in the stream, find a voxel

Stage 3: For each voxel in the stream, intersect the

Stage 4: For each intersection point in the stream,

ray with the primitives in the voxel.

apply shading and produce a new ray.

Survey

2002

tics <mark>k (depth</mark> < Mo©orna

z = inside ? 1 1 1 0 ht = nt / nc, ddn os2t = 1.0f - n⊓t n∩n 0, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D - nnt - N - (000

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPIII

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed H; radiance = SampleLight(&rand, I, &L, &L, e.x + radiance.y + radiance.z) > 0) && ()

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvis at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sol. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2*: <u>https://en.wikipedia.org/wiki/Stream_processing</u> urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



2002

at a = nt

), N); refl * E * diffuse;

AXDEPTH)

sion = true:

survive = SurvivalProbability(lf: radiance = SampleLight(&rand, e.x + radiance.y + radiance.z) >

v = true: at brdfPdf = EvaluateDiffuse(| at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely foll /ive)

*: Interactive multi-pass programmable shading, Peercy et al., 2000. at3 brdf = SampleDiffuse(diffuse, N, r1, urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf);

Ray Tracing on Programmable Graphics Hardware

Stream computing without flow control:

Assign a state to each ray.

Traversing;

shading;

done.

3.

4.

state*.

intersecting;



2002

at a = nt

), N);

AXDEPTH)

v = true;

/ive)

lf;

Ray Tracing on Programmable Graphics Hardware

Stream computing without flow control: Render two triangles, shader performs ray tracing efl + refr)) && (depth survive = SurvivalProbabi radiance = SampleLight(& e.x + radiance.y + radiance at brdfPdf = EvaluateDiffu at3 factor = diffuse * INV at weight = Mis2(directPdf at cosThetaOut = dot(N, 1 E * ((weight * cosThetaOut Use stencil to select functionality andom walk - done properl



at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, I urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true:

2002

tics ⊾(depth < PV0000000

: = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt - nn D, N); ?)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N - (300

= * diffuse = true;

efl + refr)) && (depth < MODEDII

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly ff; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && (dot)

w = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf

at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (rac

andom walk - done properly, closely following See /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2*: Accelerated ray tracing system. Fujimoto et al., 1986. pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Ray Tracing on Programmable Graphics Hardware

Acceleration structure (grid) traversal:

- l. setup traversal;
- 2. one step using 3D-DDA*.

Note that *each step* through the grid requires *one pass*.



2002

rics & (depth < ≯Voccorro)

: = inside ? 1 1 1 1 ht = nt / nc, ddn bs2t = 1.0f - nnt = on D, N); 3)

at a = nt - nc, b = nt - n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (dd)

= * diffuse; = true;

. efl + refr)) && (depth < MADEPTI

), N); efl * E * diffu: = true;

MAXDEPTH)

survive = SurvivalProbability efficiency

passes

if; radiance = SampleLight(&rand, I, &L, e.x + radiance.y + radiance.z) > 0) &

w = true; at brdfPdf = EvaluateDiffuse(L, N) = R

at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf

at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Solido /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Ray Tracing on Programmable Graphics Hardware

Results



Here, 'efficiency' is the average ratio of active fragments during each pass.



2002

nics ⊾(depth < Motoonn

z = inside ? 1 1.0 ht = nt / nc, ddm bs2t = 1.0f - nnt " on 0, N); 3)

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0 Tr) R = (D ⁼ nnt - N - (ddn

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPT

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly closed ff; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Psuevon at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (rea

andom walk - done properly, closely following Sec. /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); Sion = true:

Ray Tracing on Programmable Graphics Hardware

Conclusions

- Ray tracing can be done on a GPU
- GPU outperforms CPU by a factor 3x (for triangle intersection only)
- Flow control is needed to make the full ray tracer efficient.



2005

ata = nt -

fl + refr)) && (depth

), N); refl * E * diffuse;

AXDEPTH)

survive = SurvivalProbability(diff lf: radiance = SampleLight(&rand, I, & .x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf) at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely foll /ive)

urvive; pdf;

KD-Tree Acceleration Structures for a GPU Raytracer*

Observations on previous work:

- Grid only: doesn't adapt to local scene complexity
- kD-tree traversal can be done on the GPU, but the stack is a problem.

Goal:

Implement kD-tree traversal without stack.

at3 brdf = SampleDiffuse(diffuse, N, r1,) *: KD-Tree Acceleration Structures for a GPU Raytracer, Foley & Sugerman, 2005

n = E * brdf * (dot(N, R) / pdf);sion = true:

17

Graphics Hardware (2005) M. Meissner, B.- O. Schneider (Editors)

KD-Tree Acceleration Structures for a GPU Raytrace

Tim Foley and Jeremy Sugerman

Stanford University

Abstract

Modern graphics hardware architectures excel at compute-intensive tasks such as ray-triangle intersection, m ing them attractive target platforms for raytracing. To date, most GPU-based raytracers have relied upon unife grid acceleration structures. In contrast, the kd-tree has gained widespread use in CPU-based raytracers and regarded as the best general-purpose acceleration structure. We demonstrate two kd-tree traversal algorithms s able for GPU implementation and integrate them into a streaming raytracer. We show that for scenes with m objects at different scales, our kd-tree algorithms are up to 8 times faster than a uniform grid. In addition, identify load balancing and input data recirculation as two fundamental sources of inefficiency when raytrac on current graphics hardware.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Graphics processors I. [Computer Graphics]: Raytracing

1. Introduction

The computational demands of raytracing have generated interest in using specialized hardware to accelerate raytracing tasks. It has been demonstrated that raytracing can be achieved in real time on custom hardware [Hal01, SWS02, SWW*04], or by using a supercomputer or cluster of computers [PMS*99, WBS03]. Experiments that used programmable graphics for ray-triangle intersection [CHH02, BFH*04] have also demonstrated that GPUs can outperform CPU implementations.

Purcell et al. [PBMH02] show that the entire raytracing process - camera ray generation through shading - can be implemented on a GPU using a stream programming model. Their work has led to several other GPU raytracer implementations [MFM04, Chr05, KL04] and our work is an extension of their approach.

All of these systems used a uniform grid acceleration data structure. Purcell et al. explain that the uniform grid enables constant-time access to the grid cells, takes advantage of coherence using the blocked memory system of the GPU, and allows for easy iterative traversal via 3D line drawing. It is,

Graphics Hardware 2005, 30-31 July 2005, Los Angeles CA © 2005 ACM 1-59593-086-8/05/0007 \$5.00

however, a suboptimal acceleration structure for se nonuniform distributions of geometry.

The relative performance of different acceleration tures has been widely studied. Havran [Hav00] c large number of acceleration structures across a scenes and determines that the kd-tree is the best purpose acceleration structure for CPU raytracers seem natural, therefore, to try to use a kd-tree to GPU raytracing. As we will describe in section the standard algorithm for kd-tree traversal relies ray dynamic stack. Ernst et al. [EVG04] demon this data structure can be built on the GPU, and i a stack-based kd-tree traversal. However, their ap quires storage proportional to the maximum st multiplied by the number of rays, which may limi ber of rays that can be traced in parallel. Also, p the stack requires additional render passes with a operation.

Our work instead presents kd-tree traversal a kd-restart and kd-backtrack that run without a show that these new algorithms maintain the exp formance of kd-tree traversal. We also present a G raytracer that incorporates these algorithms and de that, as on CPUs, they outperform uniform grid ac structures with scenes of sufficient complexity. F

[†] {tfoley, yoe1}@graphics.stanford.edu

2005

ics A **(depth** < PV0000000

: = inside ? 1 1 1 1 ht = nt / nc, ddn us2t = 1.0f - nnt " nn D, N); D)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0) Fr) R = (D ⁼ nnt - N = (ddn

= * diffuse; = true;

. :fl + refr)) && (depth < MOXDERIO

D, N); refl * E * diffuse; = true;

AXDEPTH)

/ive)

sion = true

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L 2.x + radiance.y + radiance.z) > 0)

w = true; at brdfPdf = EvaluateDiffuse(L, at3 factor = diffuse * INVPI;

at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf) * (andom walk - done properly, closely following

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf);

KD-Tree Acceleration Structures for a GPU Raytracer

Recall standard kD-tree traversal:

Setup:

1. tmax, tmin = intersect(ray, root bounds);

Root node:

- 2. Find intersection *t* with split plane
- 3. If tmin $\leq t \leq t$ max:
 - Process near child with segment (tmin, t)
 - Process far child with segment (*t*, tmax)
 - else if t > tmax:
 - Process left child with segment (tmin,tmax)
- 5. else
 - Process right child with segment (tmin,tmax)



2005

tics **k (depth** k ≯VXDEFTH

: = inside ? 1 1.0 ht = nt / nc, ddn os2t = 1.0f - n⊓t = n∩ 0, N); 0)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0) Fr) R = (D ⁼ nnt - N = (ddn

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPIII

D, N); refl * E * diffuse; = true;

AXDEPTH)

sion = true:

survive = SurvivalProbability(diffuse estimation - doing it properly, close If; radiance = SampleLight(&rand, I, &L, 2.x + radiance.y + radiance.z) > 0) 33

v = true; at brdfPdf = EvaluateDiffuse(L, at3 factor = diffuse * INVPI;

at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely following S /ive)

, t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dpdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf);

KD-Tree Acceleration Structures for a GPU Raytracer

Recall standard kD-tree traversal:

Setup:

1. tmax, tmin = intersect(ray, root bounds);

Root node:

- 2. Find intersection *t* with split plane
- 3. If tmin $\leq t \leq t$ max:
 - Push far child
 - Continue with near child

4. else if t >tmax:

Process left child with segment (tmin,tmax)

5. else

Process right child with segment (tmin,tmax)



2005

ics C**depth** < Modelle

: = inside ? 1 1 1 2 ht = nt / nc, ddn = 2 bs2t = 1.0f - ant " ant D, N); >)

at a = nt - nc, b = nt + nc at Tr = 1 - (R0 + (1 - R0) Fr) R = (D = nnt - N = (ddn

= * diffuse; = true;

efl + refr)) && (depth < MAXDEP)

), N); refl * E * diffuse = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, close df; radiance = SampleLight(&rand, I, &L, & 2.x + radiance.y + radiance.z)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf

at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely following Sec /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

KD-Tree Acceleration Structures for a GPU Raytracer

Traversing the tree without a stack:

If we always pick the nearest child, the only value that will change is tmax.

Setup:

- 1. tmax, tmin = intersect(ray, root bounds);
- 2. Always pick the nearest child.
- 3. Once we have processed a leaf, restart with:
 - tmin=tmax
 - tmax= intersect(ray, root bounds)

This algorithm is referred to as *kd-restart*.

Note that the average ray intersects only a small number of leafs. Since restart only happens for each intersected leaf that didn't yield an intersection point, the expected cost is still $O(\log n)$.





2005

rics & (depth < ™ocoorrel

: = inside } 1 ht = nt / nc, ddn bs2t = 1.0f - nmt = nm D, N); ∂)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0) Tr) R = (D = nnt - N - (dd)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEP

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, ff; radiance = SampleLight(&rand, I, &L, &ilent 2.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (1800)

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

KD-Tree Acceleration Structures for a GPU Raytracer

We can reduce the cost of a restart by storing node bounds and a parent pointer with each node.

Instead of restarting at the root, we now restart at the first ancestor that has a non-empty intersection with (tmin,tmax).

This algorithm is referred to as *kd-backtrack*.



2005

ics (depth < MODEFTE

:= inside ? 1 : 1 ... ht = nt / nc, ddn = ... hs2t = 1.0f - nnt = nn p, N); })

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (ddn

* diffuse; = true;

efl + refr)) && (depth < MAXO

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse) estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &I 2.x + radiance.y + radiance.z) > 0) &&

v = true;

at brdfPdf = EvaluateDiffuse(L, N) * Psurvice at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * Psa

andom walk - done properly, closely following Sour /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

KD-Tree Acceleration Structures for a GPU Raytracer

Implementation: each ray is assigned a state:

- . Initialize: finds tmin,tmax for each ray in the input stream
- 2. Down: traverses each ray down by one step
- 3. Leaf: handles ray/leaf intersection for each ray
- 4. Intersect: performs actual ray/triangle intersection
- 5. Continue: decides whether each ray is done or needs to restart / backtrack
- 6. Up: performs one backtrack step for each ray in the input stream.

As before, the state is used to mask rays in the input stream when executing each of the 6 programs.





2005

AXDEPTH)

urvive = SurvivalProbability(diffuse estimation - doing it properly,	brute force
lf; adiance = SampleLight(&rand, I, &L,)	grid
<pre>e.x + radiance.y + radiance.z) > 0) 38 v = toue:</pre>	kd-restart
r= true; t brdfPdf = EvaluateDiffuse(L, N) t3 factor = diffuse * INVPI;	d-backtrack

at weight = Mis2(directPdf, brdfPd

at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

/ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r urvive;

pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

KD-Tree Acceleration Structures for a GPU Raytracer

Results (ms*):

force	23	4620	4770	7350
grid	63	357	8344	2687
start	80	701	968	992
track	84	690	946	857



Advanced Graphics – GPU Ray Tracing (1)

Survey

2007

hics & (depth ≪ Modernal)

: = inside ? 1 1 1 2 ht = nt / nc, ddn = 1 952t = 1.0f - nnt = nn 9, N); >)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (ddn

= * diffuse; = true;

efl + refr)) && (depth <

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed H; radiance = SampleLight(&rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) & closed e.x + radiance.y + radiance

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (add

andom walk - done properly, closely following vive)

; at3 brdf = SampleDiffuse(diffuse, N, r1 urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Interactive k-d tree GPU raytracing* Stackless KD-tree traversal for high performance GPU ray tracing**

Observations on previous work:

- GPU ray tracing performance can't keep up with CPU
- Kd-restart requires substantially more node visits
- Kd-backtrack increases data storage and bandwidth
- Looping and branching wasn't available, but is now.

*: Interactive k-d tree GPU raytracing, Horn et al., 2007

**: Stackless KD-tree traversal for high performance GPU ray tracing, Popov et al., 2007

Buarlord University *			
<text><text><text><text><text><text><text><text><text></text></text></text></text></text></text></text></text></text>	<text><text><text><text></text></text></text></text>		
<text><text><section-header><text></text></section-header></text></text>	<text><text><section-header><text><text><text></text></text></text></section-header></text></text>		
	Anto, Kato, Catoro Rose Depth for goal of the out of 1 of a property		
(i.e. it remains stoticly in the one solution or stoticly in the armsteen), it is and to continue surving the restart imation down. However, the terms of appettus developing any paralle ones to handle experimently to one shift and expr that space	to the vestor with all our optimizations applied (8.3). 4 Implementation		
Individualities, in single-service galaxy and the second pack-down, is simpleforward. The major location of this optimic ation is that it is no larger insertial as soon as inserved momentum the first mode a pre-unit constraint and constraints and the solidary	We implemented the optimizations from Section 3 is a rea- derer for an ATI X100X TX GPU. While the condense can be ready that the index optimized shall be a traditional 320 GPU conduction. One index optimized shall be a traditional 320 GPU conduction.	1.1	

Interactive k-D Tree GPU Ravtracin

2007

ics k (depth ≪ Modestifi

t = inside 7 1 1 1 1 nt = nt / nc, ddn - 1 ns2t = 1.0f - n⊓t - n 2, N); 2)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Γ) R = (D = nnt - N - (00)

= * diffuse; = true;

efl + refr)) && (depth < NOCEPTI

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed H; radiance = SampleLight(&rand, I, &L, aligned e.x + radiance.y + radiance.z) > 0) && closed

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) ();

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Interactive k-d tree GPU raytracing Stackless KD-tree traversal for high performance GPU ray tracing

Ray tracing with a short stack:

By keeping a fixed-size stack we can prevent a restart in almost all cases.





2007

at a = nt

), N); refl * E * diffuse;

AXDEPTH)

urvive; pdf;

survive = SurvivalProbability(radiance = SampleLight(&rand, I,

.x + radiance.y + radiance.z)

v = true: at brdfPdf = EvaluateDiffuse(| at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely foll /ive)

*: Ray tracing with rope trees, Havran et al., 1998 at3 brdf = SampleDiffuse(diffuse, N, r1, n = E * brdf * (dot(N, R) / pdf);sion = true

kD-tree Traversal using Ropes*

"The main goal of any traversal algorithm is the efficient front-to-back enumeration of all leaf nodes pierced by a *ray. From that point of view, any traversal of inner nodes* of the tree (...) can be considered overhead that is only necessary to locate leafs quickly."

Algorithm:

- Traverse to a leaf;
- If no intersection found: 2.
 - Follow rope;
 - Goto 1.



2007

ics k (depth ≪ Modestifi

: = inside ? 1 ht = nt / nc, ddn = 0 ps2t = 1.0f - nmt = n D, N); 2)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPT)

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Pourvis at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Interactive k-d tree GPU raytracing Stackless KD-tree traversal for high performance GPU ray tracing

Ray tracing with flow control:

25x performance of the previous paper1.65x - 2.3x from algorithmic improvements3.75x from hardware advances

 \rightarrow 2.9x from switching from multi-pass to single-pass.



2007

tics & (depth < ≯Vocbartel)

: = inside ? 1 ()) ht = nt / nc, ddn bs2t = 1.0f - n⊓t ~) D, N); ⊅)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁺ nnt - N - (ddm)

= * diffuse; = true;

efl + refr)) && (depth < MODEPID

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &L e.x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse(L, N at3 factor = diffuse * INVPI;

at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

*: Hardware: GeForce 8800 GTX / Opteron @ 2.6 Ghz, performance in fps @ 1024x1024.

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Interactive k-d tree GPU raytracing Stackless KD-tree traversal for high performance GPU ray tracing

Results*:





2007

rics & (depth < ™0055555

: = inside } 1 ht = nt / nc, ddn bs2t = 1.0f - nnt ∩ D, N); ð)

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N ⁻ (ddn

= * diffuse; = true;

. efl + refr)) && (depth < MOODEPI

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L, &liento e.x + radiance.y + radiance.z) > 0) && (dot)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Psucchor at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (Pad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Interactive k-d tree GPU raytracing Stackless KD-tree traversal for high performance GPU ray tracing

Conclusions

- Compared to kd-restart, approx. 1/3rd of the nodes is visited;
- The GPU now outperforms a quad-core CPU;
- NVidia GTX 8800 does 160 GFLOPS; cost per ray is 10.000 cycles...



2007

tics **k (dept**h ≪ Modesti-

: = inside ? 1 1 1 3 ht = nt / nc, ddm ht = 1.0f - nnt - nn 2, N); 3)

at a = nt - nc, b = nt = n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N = (ddm)

= * diffuse; = true;

-:fl + refr)) && (depth < H

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly ff; radiance = SampleLight(&rand, I, &L, &Lie e.x + radiance.y + radiance.z) > 0) & 0)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Psumble at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (();

andom walk - done properly, closely following Sec /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1,) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Realtime Ray Tracing on GPU with BVH-based Packet Traversal*

Observations on previous work:

- kD-trees limit rendering to static scenes
- kD-trees with ropes are inefficient storage wise
- Popov et al.'s tracer achieves only 33% utilization due to register pressure
- Existing GPU ray tracers do not realize GPU potential
- Existing GPU ray tracers suffer from execution divergence.

Solution:

Use BVH instead of kD-tree.



= SampleDiffuse(diffuse, N, r1, r2*: Realtime ray tracing on GPU with BVH-based packet traversal, Günther et al., 2007

2007

hics & (depth < ™ocoorrel

: = inside ? 1 . . . ht = nt / nc, ddn . . bs2t = 1.0f - nnt . . D, N); 3)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - ddm

E * diffuse = true;

. efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse .estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1 urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Realtime Ray Tracing on GPU with BVH-based Packet Traversal

Recall: thread state must be small*.

An important difference between kD-tree packet traversal and BVH packet traversal is that kD-tree traversal requires a stack for the packet plus (tmin, tmax) *per ray*, while the BVH packet only requires a stack.

*: To achieve maximum utilization of a G80 GPU, we need 768 threads per multiprocessor (i.e., 24 warps). Each multiprocessor has 16Kb shared memory and 32Kb register space \rightarrow for 24 warps we have 5 words plus 10 registers per thread available. Beyond that, we are forced to use global memory.



2007

tics k (depth < Modess

: = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt = nn 2, N); 3)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N - (100

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEP

D, N); refl * E * diffus = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L, &L) e.x + radiance.y + radiance.z) > 0) &&

v = true;

at brdfPdf = EvaluateDiffuse(L, N) * Psurvise at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (();

andom walk - done properly, closely following Sour /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



1. A packet consists of 8x4 rays, handled by a single warp

- The packet traverses the BVH using *masked traversal* (where t is used as mask)
- 3. Storage:
 - 1. Per ray: O, D, t (7 floats)
 - 2. Per packet: stack





hics & (depth < ≯voccorre

: = inside ? 1 = 1.0 ht = nt / nc, ddn = 0 os2t = 1.0f - nnt = or 0, N); 2)

at a = nt - nc, b = nt = n at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N = (10)

= * diffuse; = true;

. efl + refr)) && (depth < NACEPI

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly df; radiance = SampleLight(&rand, I, &L, & e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, N) Ps at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sec /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



Observations:

This is hardly a packet traversal scheme; we are essentially traversing 32 independent rays.

However:

the rays in the packet *do* share a single stack.

Question:

will rays ever visit a node they didn't have to visit? (i.e., do they visit a node they would not have visited using a stack per ray?)



2007

at a = nt

AXDEPTH)

survive = SurvivalProbability(dif lf; radiance = SampleLight(&rand, +shadow e.x + radiance.y + radiance.z)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf)

at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely foll /ive)

*: Hardware: GeForce 8800 GTX, rendering at 1024x1024, performance in fps.

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, & urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true:

Realtime Ray Tracing on GPU with BVH-based Packet Traversal

Results*:

primary





hics & (depth < Monocr

: = inside ? | ht = nt / nc, ddn os2t = 1.0f - nnt 0, N); 0)

at a = nt - nc, b = nt + nc at Tr = 1 - (R0 + (1 - R0) Fr) R = (D = nnt - N = (ddn -

= * diffuse; = true;

-:fl + refr)) && (depth < Ⅳ

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(%rand, I, %L 2.x + radiance.y + radiance.z) > 0) %%

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Pa at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Se /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Digest

Challenges in GPU ray tracing:

- Utilizing GPU compute potential (getting it to work \rightarrow beating CPU \rightarrow efficient)
- Mapping an embarrassingly parallel algorithm to a streaming processor
- Tiny per-thread state (balancing utilization / algorithmic efficiency)
- Freedom in the choice of acceleration structure
- Tracing divergent rays





tics & (depth < Mot000

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0 Tr) R = (D ⁺ nnt - N - (ddn

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTI

), N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && (dot)

v = true; at brdfPdf

at brdfPdf = EvaluateDiffuse(L, N) * Psurvio at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (PS

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Exam Questions: Sampler
- Introduction
- Survey: GPU Ray Tracing
- Practical Perspective



sics & (depth < ⊅vocco

= inside ? 1 ht = nt / nc, ddm bs2t = 1.0f - nmt D, N); D)

at a = nt - nc, b = nt + at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (0

= * diffuse = true;

efl + refr)) && (depth < MODEPT

D, N); refl * E * diff = true;

AXDEPTH)

survive = SurvivalProbability(diffus estimation - doing it properly, clust if; radiance = SampleLight(&rand, I, &t, e.x + radiance.y + radiance.z) > 0) &

v = true; at brdfPdf = EvaluateDiffuse(L, N

at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directf

andom walk - done properly, closely † /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, CpOF urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:





2013

tics € (depth < MoxDern

: = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); D)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Tr) R = (D ⁼ nnt - N - (ddn -

= * diffuse; = true;

efl + refr)) && (depth <

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly ff; radiance = SampleLight(&rand, I, &L, e.x + radiance.v + radiance.v) > 0

v = true; at brdfPdf = EvaluateDiffuse

at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf) at cosThataOut = dat(N =));

at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely following /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sign = true;

Pragmatic GPU Ray Tracing*

Context:

- Real-time demo
- 50-100k triangles
- Fully dynamic scene
- Fully dynamic camera (no time to converge)
- Must "look good" (as opposed to "be correct")
- \rightarrow Rasterize primary hit
- → No BVH / kD-tree

→ Use a grid (or better: sparse voxel octree / brickmap).

*: Real-time Ray Tracing Part 2 – Smash / Fairlight, Revision 2013

https://directtovideo.wordpress.com/2013/05/08/real-time-ray-tracing-part-2



2013

ics ▶ (depth < Modern

z = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt 0, N); 8)

at a = nt - nc, b = nt = n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = ddm

= * diffuse; = true;

efl + refr)) && (depth < N

), N); ∵efl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && closed

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Psumble at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (Page 1);

andom walk - done properly, closely following Sour /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Pragmatic GPU Ray Tracing

Grid traversal: 3D-DDA

Brickmap traversal:

- build in linear time
- locate ray origins in constant time
- skip some open space
- little flow divergence in shader
- simple thread state





2013

ics (depth < Moderne

: = inside ? 1 ht = nt / nc, ddn hs2t = 1.0f - nnt = nn), N); >)

t a = nt - nc, b = nt - nc nt Tr = 1 - (R0 + (1 - R0 r) R = (D = nnt - N - (dd)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEP

D, N); refl * E * diffu = true;

(AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly f; radiance = SampleLight(&rand, I, &L, eller e.x + radiance.y + radiance.z) > 0) &

v = true; at brdfPdf = EvaluateDiffuse

at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPd

andom walk - done properly, closely fo

; at3 brdf = SampleDiffuse(diffuse, N, r1, urvive; pdf; n = E * brdf * (dot(N, R) / pdf);

Pragmatic GPU Ray Tracing

Filling the grid: using rasterization hardware.
→ Determine which voxels a triangle overlaps.

Algorithm:

- 1. Determine for which plane (xy, yz, xz) the triangle has the greatest projected area.
- 2. Rasterize to that face; use interpolated x, y and depth to determine voxel coordinate.
- 3. Use conservative rasterization*, **.



https://developer.nvidia.com/content/basics-gpu-voxelization







2013

tics k (depth < Modesta

: = inside ? 1 a 1.3 ht = nt / nc, ddn = 3 bs2t = 1.0f - n⊓t * n D, N); ð)

nt a = nt - nc, b = nt - nc nt Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N - ddn

= * diffuse; = true;

. efl + refr)) && (depth < MaxDErro

), N); refl * E * diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(

at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPd

andom walk - done properly, closely fo /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Pragmatic GPU Ray Tracing

In this case, we are not building a voxel set, but a grid with pointers to the original triangles.

→ Add each triangle to a preallocated list per node.

From grid to brickmap:

- each brick consists of a small grid, e.g. 4x4x4.
- repeat the rasterization process at the higher resolution
- assign each triangle to cells in the fine grid.

Note that voxelization can be part of a rasterization-based rendering pipeline; it can e.g. be fed with triangles of a skinned mesh or even procedurally generated meshes.









2013

ics **b (dept**h < Modelli

: = inside ? 1 1 1 1 ht = nt / nc, ddn - 1 bs2t = 1.0f - nnt - onn D, N); D)

at a = nt - nc, b = nt - r at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ^{*} nnt - N * (dd)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPIII

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (red

andom walk - done properly, closely following Seri /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Pragmatic GPU Ray Tracing

Pragmatic traversal:

- 'Trace' primary ray using rasterization
- Determine secondary ray origin from G-buffer

After this:

 Put a maximum on the number of traversal steps, regardless of bounce depth.







2013

rics ⊾(depth < Not00 = 1

: = inside ? 1 = 1.0 ht = nt / nc, ddn bs2t = 1.0f - nnt = onn D, N); 3)

at a = nt - nc, b = nt + nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (ddn)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPT

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; adiance = SampleLight(&rand, I, &L, e.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psury at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following SOO. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Pragmatic GPU Ray Tracing

Pragmatic diffraction:

Each ray represents 3 'wavelengths', and each results in a different refracted direction. However, only the direction of the first ray is actually used to find the next intersection for the triplet.

EXCEPT: when the rays exit the scene and returns a skybox color; only then the three directions are used to fetch 3 skybox colors which are then blended.



2013

nics **k (depth** < ≫ocos

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Ir) R = (D ⁺ nnt - N

= * diffuse = true;

• efl + refr)) && (depth < MAXDEPII

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly
f;
radiance = SampleLight(&rand, I, &L, &L
2.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following 300 /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Pragmatic GPU Ray Tracing

Pragmatic depth of field:

Since primary rays are rasterized, the camera used is a pinhole camera.

Depth of field with bokeh is simulated using a postprocess.

See for a practical approach:

Bokeh depth of field – going insane! part 1, Bart Wroński, 2014, <u>http://bartwronski.com/2014/04/07/bokeh-depth-of-field-going-insane-part-1</u>





2013

tics k (depth < Modesta

: = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); D)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁺ nnt - N - (dd)

= * diffuse; = true;

. efl + refr)) && (depth < MODEPTH

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &L e.x + radiance.y + radiance.z) > 0) &

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psu at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely followi /ive)

, H33 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Pragmatic GPU Ray Tracing

Limitations:

- Doesn't work well for 'teapot in a stadium'
- Not suitable for very large scenes (area)
- Manual parameter tweaking
- → The method is not good for a general purpose ray tracer, but really clever for a special purpose renderer.

 Performance is very good, although hard to estimate: Demo runs @ 60fps on a high-end GPU; Traces ~1M primary rays; Most rays make several bounces (very divergent!); Guestimate: ~250M rays per second for a fully dynamic scene.









2013

nics & (depth ≪ Moderna

: = inside ? 1 = 1.3 ht = nt / nc, ddn = 3 bs2t = 1.0f - nnt = nnt 2, N); 2)

at a = nt - nc, b = nt = n at Tr = 1 - (R0 + (1 - R0) Γ r) R = (D = nnt - N (ddn)

= * diffuse; = true;

• efl + refr)) && (depth < MAXDED

), N); refl * E = true;

AXDEPTH)

survive = estimati If; radiance

e.x + rad w = true; at brdfPd at3 facto

at weight at cosThe

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely followin /ive)

, H33 brdf = SampleDiffuse(diffuse, N, r1, r2, SR, apdf) urvive; .pdf; n = E * brdf * (dot(N, R) / pdf); Sion = true:

Other Real-time Ray Tracing Demos

For a brief history, see these links:

http://datunnel.blogspot.nl/2009/12/history-of-realtime-raytracing-part-1.html http://datunnel.blogspot.nl/2009/12/history-of-realtime-raytracing-part-2.html http://datunnel.blogspot.nl/2009/12/history-of-realtime-raytracing-part-3.html

Also check here: <u>http://mpierce.pie2k.com/pages/108.php</u>









tics & (depth < Mot000

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0 Tr) R = (D ⁺ nnt - N - (ddn

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTI

), N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && (dot)

v = true; at brdfPdf

at brdfPdf = EvaluateDiffuse(L, N) * Psurvio at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (PS

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Exam Questions: Sampler
- Introduction
- Survey: GPU Ray Tracing
- Practical Perspective



Next Time

sics & (depth < NADD

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (330

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPT

D, N); refl * E * diffu: = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed Hf; radiance = SampleLight(&rand, I, &L, aligned e.x + radiance.y + radiance.z) > 0) && closed e.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (real

andom walk - done properly, closely following SAS. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Coming Soon in Advanced Graphics

GPU Ray Tracing Part 2:

- State of the art BVH traversal by Aila and Laine;
- Wavefront Path Tracing
- Heterogeneous Path Tracing: Brigade.







hics & (depth < Motos

: = inside ? 1 ht = nt / nc, ddn = 0 bs2t = 1.0f - nmt = on D, N); B)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N = (dd)

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) 2:x + radiance.y + radiance.z) > 0

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pourvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (rad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dodf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

INFOMAGR – Advanced Graphics

Jacco Bikker - November 2021 - February 2022

END of "GPU Ray Tracing (1)"

next lecture: "Variance Reduction"

