Ray Tracing: Strengths and Opportunities

Eric Haines
Autodesk Inc.

Pretty soon, computers will be fast. – Billy Zelsnack

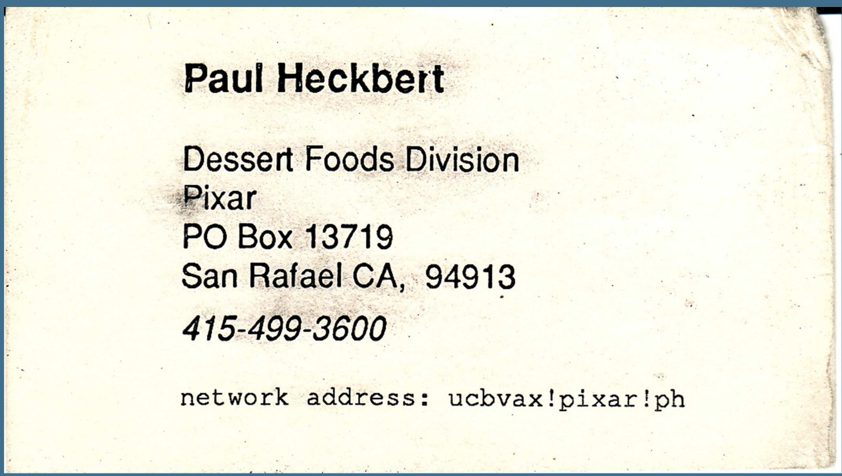
However, it still takes up to twenty seconds for me to find what’s in a directory when I double-click it.

Why the Beveled Top?

Why is the diameter a bit smaller?

Why Ray Tracing is Great

- Size:



Why Ray Tracing is Great

- Size: kernel idea is truly small

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>=1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){printf("%d %d\n",32,32);while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*pixar!ph*/
```

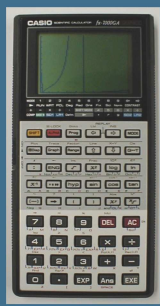
From 1987, seminal paper being *Ray Tracing Jell-O Brand Gelatin*

Why Ray Tracing is Great

- Size: kernel idea is truly small

Tube
by
Baze

252 byte program (and 4 byte signature)



422 byte program for a Casio FX7000Ga, Stéphane Gourichon, 1991

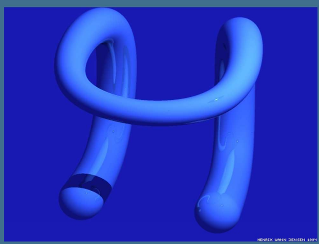
http://amphi-gouri.org/cv/2001/

Why Ray Tracing is Great

- Shapes: intersectable == renderable

Turner Whitted

William Hollingworth



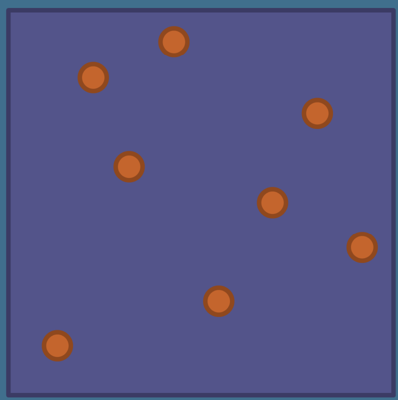
Henrik Wann Jensen

Ken Musgrave

Upper left: Whitted, Upper Right: William Hollingworth http://web2.iadfw.net/will/gallery.html, Lower Left: Henrik Wann Jensen http://graphics.ucsd.edu/~henrik/images/, Lower Right: Ken Musgrave kenmusgrave.com

Why Ray Tracing is Great

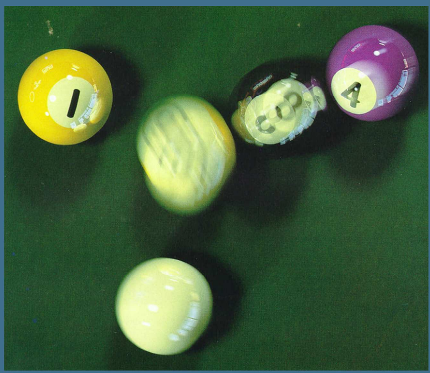
- Sampling: nonuniform, adaptive



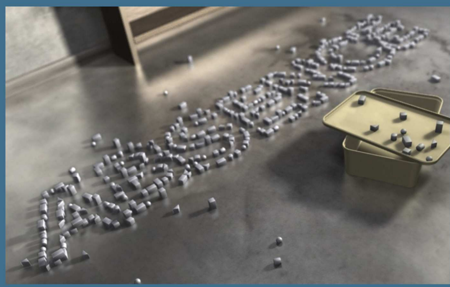
Apply the samples where you think you need them. So many criteria to choose from (color differences, shadow edges, texture frequencies, change in Phong lobe, etc.)

Why Ray Tracing is Great

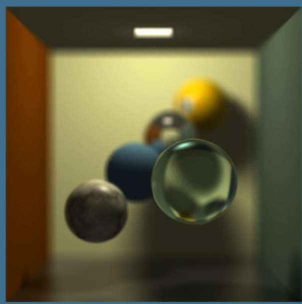
- Stochastic Effects



by Tom Porter based on research by
Rob Cook, Copyright 1984 Pixar



Matt Roberts

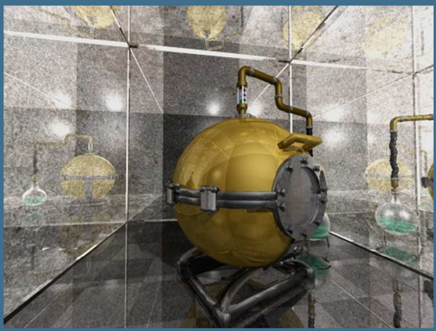


Jason Waltman

1984 by Tom Porter, Pixar, Blocks by Matt Roberts http://www.irtc.org/stills/2006-10-31.html, DOF by Jason Waltman http://www.jasonwaltman.com/graphics/rt-dof.html

Why Ray Tracing is Great

- Reflections, Refractions



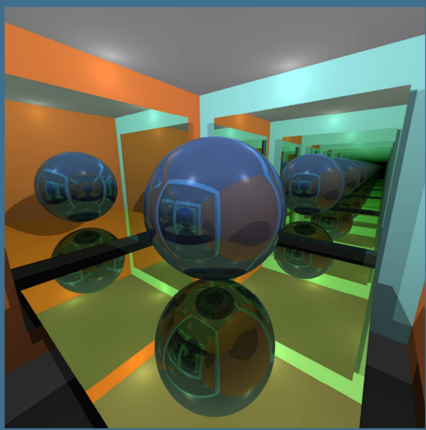
Användare:Mewlek, wikimedia

Gilles Tran, wikimedia

Left http://commons.wikimedia.org/wiki/Image:Raytracing-Boiler.jpg. Right by Gilles Tran, http://commons.wikimedia.org/wiki/Image:Glasses_800_edit.png, both from the useful page http://commons.wikimedia.org/wiki/Category:3D_computer_graphics

… is Great.

Which is ray traced, which is rasterized?



Timothy Cooper

Kasper Høy Nielsen

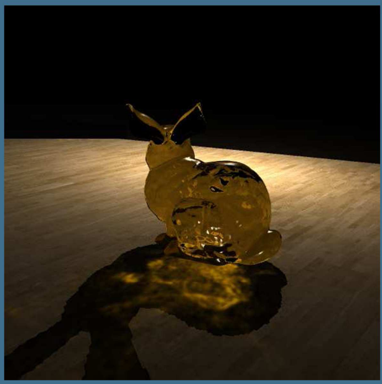
The one on the left is ray traced, you can see the barbershop mirror effect to infinity. On the right recursive environment mapping and “reflect and clip through a plane” is used for this rasterized image.

And So Is Rasterization

- Reflections, Refractions, even Caustics

From *Crysis*, by Crytek

Musawir Ali, Univ. of Central Florida

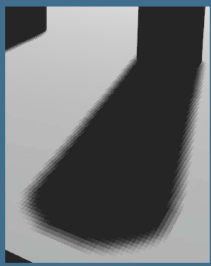


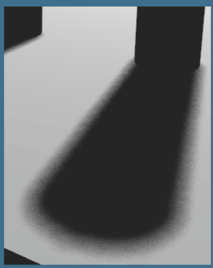
http://www.graphicshardware.org/program.html, http://www.cs.ucf.edu/~mali/

And So Is Rasterization

- Stochastic Effects

Microsoft SDK



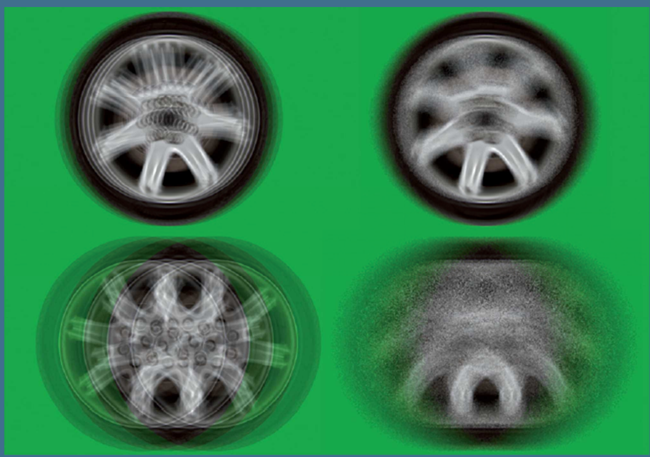


John Isidoro, ATI/AMD

NVIDIA "Toys" demo

And So Is Rasterization

- Sampling: nonuniform, adaptive

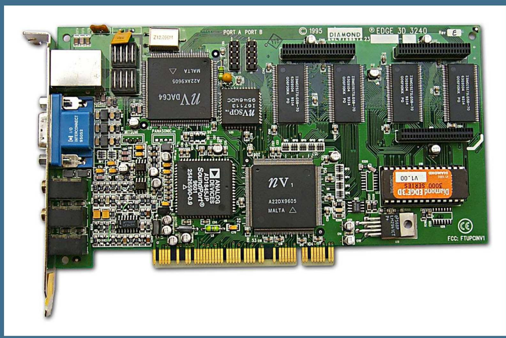


Akenine-Moller et al., GH 2007

Akenine-M¨oller, Tomas, Jacob Munkberg, and Jon Hasselgren, “Stochastic Rasterization using Time-Continuous Triangles,” *Graphics Hardware, pp. 7–16, August* 2007.

And So Is Rasterization

- Shapes:

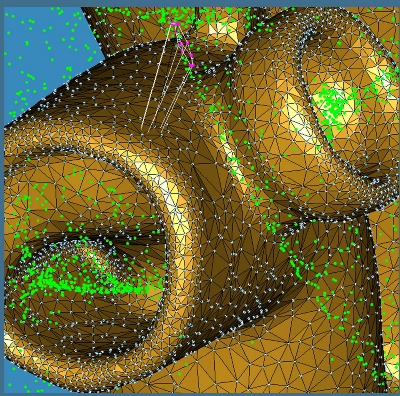


NVIDIA's first graphics card, the NV1 (circa 1995), supported ellipsoids. This design decision helped almost kill the company.

HP hardware in the late 1980's supported spline surfaces with trimming curves. Beyond cracking, you also had to avoid getting too close to a spline: data explosion, took forever to draw. I'll be interested to see how DirectX 11's tessellation engine is used to avoid the "I'm standing on the earth, looking at the horizon" problem.

Other examples: N-patches (PN triangles) never caught on.

And So Is Rasterization

- Shapes: 

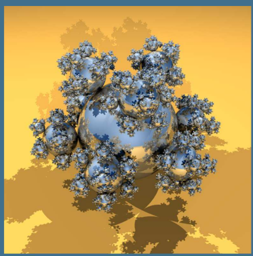
Well, everything is a point, line, or triangle, but that’s mostly true for RTRT now, too.

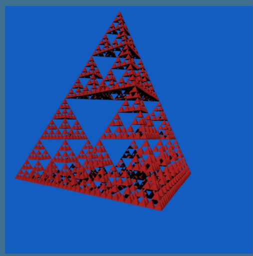
Michal Varnuška; http://herakles.zcu.cz/~miva/index.php?prom=projects&lang=en

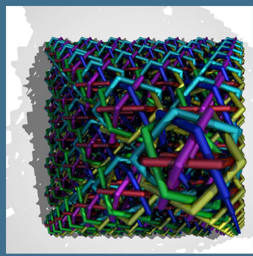
By the way, how does a ray tracer trace constant-width edges on a surface? Not obvious to me.

An Aside: SPD

- Non-triangles are mostly not worth using.







SPD: good for checking that your ray tracer is working. "Real" data is usually better for testing.

There is an option in SPD that does tessellate everything into polygons, by the way… It's the databases themselves that are sketchy: we just don't use that many quadrics in "real" scenes.

And So Is Rasterization

- Size: perhaps it cannot fit on a business card, but it can work on a cell phone or iPod.

OpenGL ES demo of Siege,
by developer Denied Reality

"… the brute-force approach … is ridiculously expensive." - Sutherland, Sproull, and Schumacker, *A Characterization of Ten Hidden-Surface Algorithms,* 1974

About the z-buffer algorithm. Brute force beats elegance on transistor count. Uniform data structures much cheaper to put into silicon.
128 MB in 1971 would have cost $50,688 in 2008 dollars.
This same quote is often applied to ray tracing nowadays, as far as performance goes.

Where Rasterization Is

From Battlefield: Bad Company, EA Digital Illusions CE AB

Very little reflection here, and not needed. Maybe 10% of objects in a normal view of the world are reflective.

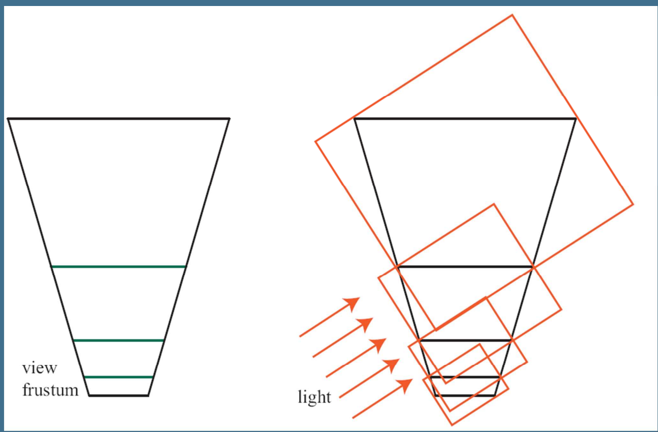
Rasterization Shadows

Fan Zhang, The Chinese University of Hong Kong

Zhang, Fan, Hanqiu Sun, and Oskari Nyman, “Parallel-Split Shadow Maps on Programmable GPUs,” in Hubert Nguyen, ed., *GPU Gems 3*, Addison-Wesley, pp. 203–237, 2007.

Rasterization Shadows

- Cascaded Shadow Maps



Research in rasterization field is getting quite good.

Rasterization Depth of Field

Circle of confusion map

Resulting image

EA Digital Illusions CE AB

So, what are the ray tracing buffers you use? None? Not surprising, if you're not on a GPU.

Opportunity: Buffers

- Hard shadows fail to please. Soft shadows are expensive with stochastic ray tracing. Buffers?
- For example: “conservative rasterization” actually makes it easy to learn exact coverage of a grid by a triangle.
  - One use: implement light buffers for RT shadows.
- Rotated-buffer per pixel for soft reflections is an old idea; idea is used for soft PCF shadows.
- GPGPU is all about buffers and streaming.

Hasselgren, J., T. Akenine-Möller, and L. Ohlsson, "Conservative Rasterization," in Matt Pharr, ed., *GPU Gems 2*, Addison-Wesley, pp. 677-690, 2005.
http://developer.nvidia.com/object/gpu_gems_2_home.html

First mention I know of rotated-grid idea for reflections is John Wallace’s thesis (Cornell Univ.).

Rasterization Is Just That Simple…

- Shadows? Cascading shadow maps, plus enhancements for objects that span the transition between two maps, plus separate buffer for animated objects, plus…
- Transparency? Sort objects, but that's error-prone and expensive. Alpha to coverage is good for cutouts but not much else. Depth peeling is too slow. Now there's stencil routing, but only on DirectX 10, and uses lots of memory, and no AA.
- Depth of Field/Motion Blur? Don't get me started…

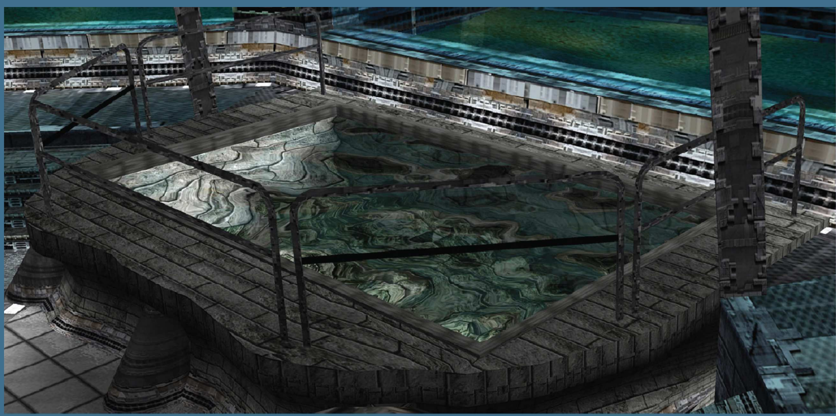
Strength of Ray Tracing: Simplicity

Ray tracing is generally easier to program and to think about.

- Ray casting and ray spawning can do it all.
- Core optimization pays off everywhere.
- Maps well to the real world.
- Easy to explain to artists (which is where the bulk of the money goes for game development).

Rasterization: Ray/Simple Object Intersection

- Done in shader

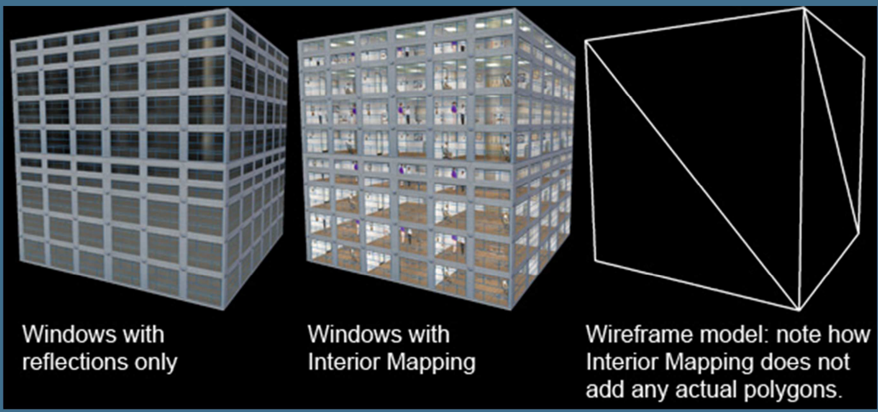


2001, Alex Vlachos, ATI Technologies Inc.

Vlachos, Alex, “Approximating Fish Tank Refractions,” in Mark DeLoura, ed.,*Game Programming Gems 2,* Charles River Media, pp. 402–405, 2001.

Rasterization:
Ray/Object Intersection

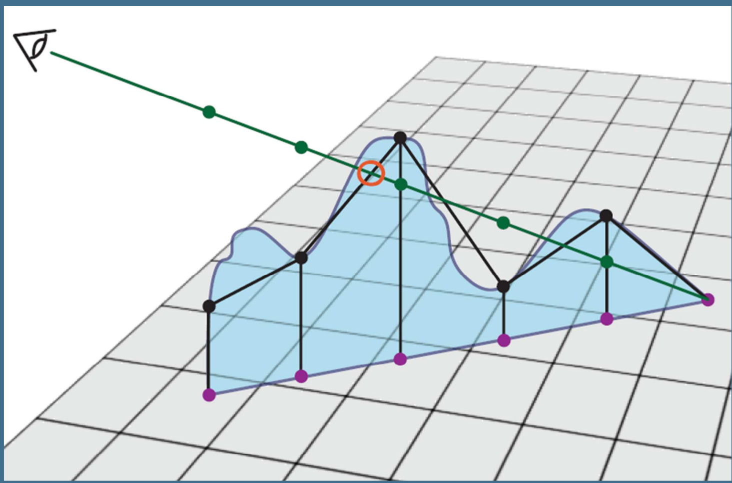
- Intersect sets of planes in shader itself



2007, Joost 'Oogst' van Dongen

Ray/More Complex Object Intersection

- Relief (or Parallax Occlusion) Mapping:



How it works.

Ray/More Complex Object Intersection

Crytek

This one amazed me when I confirmed with Martin Mittring that it was from a rasterizer. Astounding what relief mapping + shadows can do!

Ray/Even More Complex Object Intersection

GPU sampled rays, Natalya Tatarchuk, AMD, Inc.

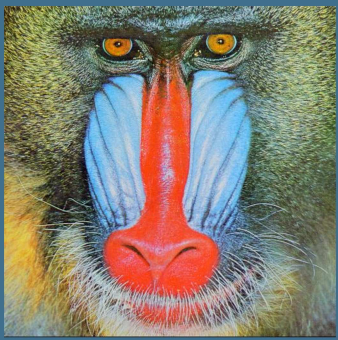
Is this rendered with rasterization or ray tracing? (And does it matter?)

Using Kr¨uger, Jens, and R¨udiger Westermann, “Acceleration Techniques for GPU-based Volume Rendering,” *IEEE Visualization 2003,* pp. 287–292, 2003. Image from Tatarchuk, Natalya, and Jeremy Shopf, “Real-Time Medical Visualization with FireGL,” *SIGGRAPH 2007,* AMD Technical Talk, August 2007.

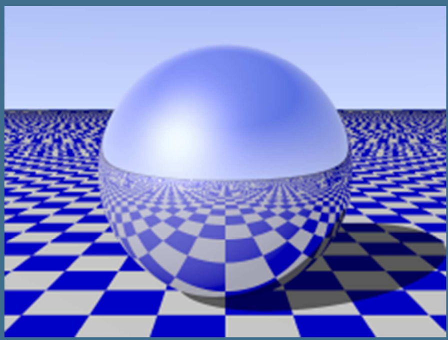
GPUs are the only type of parallel processor that has ever seen widespread success... because developers generally don’t know they are parallel! – Matt Pharr

Interactive Rendering

- We’ve had interactive ray tracing since (at least) 1987: AT&T Pixel Machine, and the Connection Machine (16k processors).



+



The Demo Scene’s done RTRT for awhile, too. “Heaven seven” by exceed is a lovely example.

Where Interactive Ray Tracing Is

Quake 4 by Daniel Pohl, Intel ray tracing group, id Software, Splash Damage

Fall 2007:
90 FPS on an 8-core system, HD resolution

Point of comparison:
Fall 2005:
115 FPS on an NVIDIA 7800 GTX , 2.8 GHz AMD Athlon 64

Not bad: 2 years behind, 25% slower.

http://www.idfun.de/temp/q4rt/, Benchmark:
http://www.pcper.com/article.php?aid=506&type=expert

http://www.tomshardware.com/reviews/vga-charts-viii,1184-11.html

Display Trends

- Rising resolution
- Higher sampling rates
- Larger filtering kernels

HIPerSpace,
UC San Diego,
287 million pixels

You can always use up processing with higher res.
Rasterization uses MSAA, CSAA, mipmaps, more.

Scenes are more complex

- Rate is much faster than display increase.

25 million quads, interactive.

Autodesk Mudbox

4-25 million quads or so… subdivision surface ray tracing might be helpful here.
Rasterization breaks down beyond the 2x2 size. But why would we want much smaller?
LOD should get used at this resolution.
http://area.autodesk.com/mudbox_preview

Strength of RT: Scene Complexity

- Favors ray tracing, in that the efficiency structure gives effectively O(log n) search (but not build).
  - Please watch the "rasterization is O(n) vs. ray tracing is O(log n)" argument. For rasterization:
    - Hierarchical frustum culling is a given.
    - Level of detail is vital, for pipeline and for limiting memory use; could be useful for ray tracing shadows and reflections, etc. "Space is Speed."
    - Occlusion culling is getting better (still not great). GPU hierarchical occlusion culling is built-in (HyperZ).
      - This truly is a place where O(n-ish) vs. O(log n) applies

Sorting from front to back helps GPU rasterization perform optimally.
After frustum and LOD, you still have to rendering everything visible, as occlusion culling is tough to use well.
Triangles smaller than 2x2 waste pixel shader processing ("fake" pixels rendered for derivatives on surface)

"Cache is king" and "Space is Speed" are from http://www.gotw.ca/publications/concurrency-ddj.htm, as well as "Andy giveth, Bill taketh away"

Interactive Rendering

- What is the most important effect in interactive rendering of any sort?

Interactivity: 6+ FPS frame rate.

Everything else is icing.

… so stop comparing with film rendering

Film rates can vary from 20 minutes a frame to 20 hours a frame (glory shots), so don't bother using it as a yardstick (though we still do).

Challenge: Constant Cost

- You have 33 ms for 30 FPS (or 16.7 ms for 60 FPS). No going over this amount of time, ever.
- Zoom in on a refractive object with ray tracing and the ray tree explodes, killing frame rate.
  - Reflection maps and similar are constant-cost.
  - Shadow volumes aren’t, so are dying out.
  - With GPUs having more complex shaders and algorithms, this problem is not just RT’s anymore.

The constant cost question comes from Johan Andersson, DICE.
http://www.graphicshardware.org/program.html

Strength of Ray Tracing: no API

- Rasterization performance is about minimizing state changes and avoiding small batches.
- CPU ray tracing works on most any computer, no chip or driver dependencies.
- Of course, we do need an API (mostly).
  - But as a productivity aid, not as a limiter.

Example: *(from Humus-3D)* This demo uses D3D10, but it could have benefited from D3D10.1 in at least two ways. First of all, multisampled depth buffers can't be used for texturing in D3D10, so this demo uses a separate render target for this purpose. Also, now the depth bits of the depth-stencil buffer is entirely unused, which is wasteful. Secondly, and probably more important, is that multisampled buffers can't be CopyResource'd in D3D10. Currently a significant chunk of the frame time is consumed just initializing the stencil buffer. A better way to handle this would be to initially set up a stencil clear-buffer just once, and then clear the active stencil buffer by copying that stencil clear-buffer into it. A copy is likely a good…

Special-purpose processors always choke off real algorithmic creativity - Jim Blinn

I don’t entirely buy this (an FPU used to be a special purpose processor), and GPUs are becoming more general, but an interesting quote to discuss. The point, to me, is that creativity should be unfettered by the hardware. The hardware is there for only one reason: speed. Otherwise, it gets in the way. However, given “frame rate is key” is the major rule, then there’s *lots* of creativity in achieving speed. Even in batch rendering, speed is vital.

Modern Processor Trends

- Moore's Law: ~1.6x transistors every year (10x every 5 years).
  - DRAM capacity similar 1.6x from 1980-1992, slowed to 1.4x 1996-2002.
- DRAM bandwidth is improving about 1.25x, 25%, a year (10x every 10 years), and latency only **5%** (10x every 48 years).
  - Bandwidth improves by at least the square of the improvement in latency [Patterson2004].

The Three Walls

- Instruction Level Parallelism (ILP): branch prediction, out of order processing, and other control improvements are mostly mined out.
- Memory: load and store is slow.
- Power: the whole reason we have multicore.
  - GHz peaked in 2005 at around 3.8 GHz.
  - Diminishing returns: increasing power does not linearly increase processing speed. 1.6x speed costs ~2-2.5x power and ~2-3x die area.

Analysis from the Berkeley report cited at end.

An old game I loaded on a new computer gave a warning that my machine wasn’t fast enough, because my MHz of the quad core had dropped.

Spending Transistors

- CPUs spend them mostly on control logic (ILP) and on memory.
- GPUs (used to) spend them on algorithm logic.
- Now the two are heading towards each other, in some ways:
  - CPU: for example, SSE through SSE5, 128 bit registers, going to data path of 256 bits with AVX in 2010.
  - GPU: Unified shaders with large pools of registers, less fixed-function stages, multiple paths out of GPU.

AVX info is on Wikipedia.

Ray Tracing: Massively Parallel

- If each computer renders one pixel…

North Korea’s Arirang festival, 100,000 people train for a year.
http://www.everyoneforever.com/content/2002-04-30/arirang_festival/

Turner Whitted mentioned the idea of a grid of computers, each with a red, green, and blue light bulb over it.

Memory & Latency

- “Cache is King”
- Missing the L2 cache and going to main memory is death, 10-50 slower. Why secondary rays usually stink.
- CPUs focus on very fast caches,
  GPUs try to hide latency via many threads.

Opportunity: Latency Hiding

- for i = 1 to 100:
  a[i] = b[i] * c + d[i]

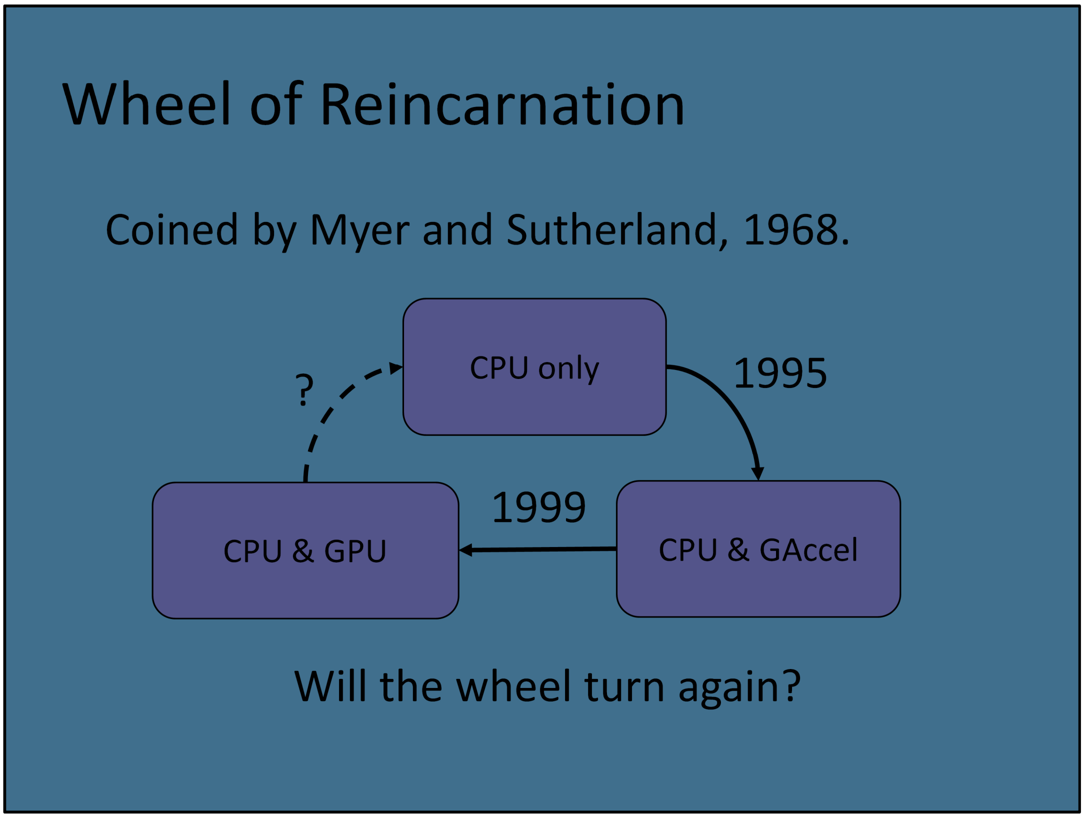
Instead, to hide memory access:

- for i = 1 to 100: t[i] = b[i]
  for i = 1 to 100: t[i] *= c
  for i = 1 to 100: t[i] += d[i]
  for i = 1 to 100: a[i] = t[i]

Idea from presentation at http://c0de517e.blogspot.com/2008/07/gpu-versus-cpu.html – clever way to talk about it.

GPUs: Upstream over Time

Display	The dark ages (early-mid 1990's), when there were only frame buffers for normal PC's.
Rasterization	Some accelerators were no more than a simple chip that sped up linear interpolation along a single span, so increasing fill rate.
Triangle Setup	Once even high-end systems supported just triangle setup and fill. CPU sent triangle with color and depth per vertex and it's rendered.
Projection & Clipping	This is where pipelines start for PC commodity graphics, 1995-1998. Seminal event is 3dfx's Voodoo in October 1996.
Transform & Lighting	This part of the pipeline reaches the consumer level with the introduction of the NVIDIA GeForce256, Fall 1999.
Geometry Shader, Instancing, Stream Out, Tessellation, …	More and more moves to the GPU – what is the best division of labor? Should it even be a pipeline, or something more general?



ATI Fusion: one effort replaces one or two of four cores with a graphics core instead. For laptops, more about power than anything.

If we stop having

The Future: The Easy Predictions

- Parallelism is the future. Design must change.
  - Intel: in 2006 gave plan of 80 cores by 2011
  - Berkeley: could have thousand cores on a chip
- Tradeoff: large, fast core vs. many slower cores.
  - All tasks need to run reasonably, serial and parallel
  - Implies a hybrid: some fast cores, many small ones
    - The “HPU”: what is our goal? Solve for “H”
- GPU ray tracing, while “transitory”, is vital for exploration of these new ecosystems
    - Using CTM, CUDA, Larrabee – all to the good!

Autotuning is very interesting for parallelism. One example: Utah’s work in tuning an algorithm by autogenerating variants.

One processor for processing, one for spyware, checking for iTunes updates, etc.

We have the CPU, skipped the DPU, EPU (FPU was used and folded back in) and jumped to the GPU, so next is the HPU. Well, there’s also the RPU.

There is an old joke that goes, "Ray tracing is the technology of the future, and it always will be!"
– David Kirk

http://www.pcper.com/article.php?aid=530

Rasterization/Ray Tracing Hybrids

Rinspeed's sQuba car

Here's a hybrid car, but not like a Prius. Any guesses as to what sort of hybrid? Concept car introduced in 2008. $1.4 million. I love that it's a convertible. More photos at http://www.carzi.com/2008/02/25/squba-underwater-car-by-rinspeed-photos-video/. That said, 4 mph as a boat, 2 mph underwater (75 mph on land).

Rasterization/Ray Tracing Hybrids

Rinspeed's sQuba car

My concern is that ray tracing as an optional frill doesn't seem likely to survive.

Another point here: ray tracing can't expect but a few transistors to be spent on it.

Other Futures

- Cloud computing. For example, Microsoft adds 10k cores a month to their cloud.
  - US broadband is slower than many countries.
  - Main problem for interactivity is lag, though. Role-Playing Games, yes; First-Person Shooters, harder.
- Mobile computing. Fewer pixels = less computation. Many small cores = low power. But, tile-based hardware is well-established and a pretty good fit.
- So, what are we forgetting?

We always forget something, when predicting. In older science fiction stories people will be running around with ray guns, but still be using old-fashioned radio sets.

Jet packs, underwater cities,…

- Between 2007 to 2024: *A Bug's Life* in real-time – Möller & Haines, 1999
- 2030: mind uploading – Kurzweil, 2005
- 2040: a thousand rays per pixel – Wallace & Haines, 1990
- 2045: The Singularity – Kurzweil, 2005
- 2070: All major minerals and energy resources exhausted – Club of Rome, 1970

What resolution? How much antialiasing? With motion blur and depth of field? At what quality? There are a few gigabytes of textures in A Bug's Life in a typical scene, so we're not there yet.

Why the Beveled Top?

The top is thicker, more expensive aluminum

Cheaper by a fraction of a cent: 1/10th of a cent * 1 billion = $1 million

It all comes down to economics: if it's more cost effective and more money can be made by doing it, it'll (eventually) get done.

The display is the computer.

– Jen-Hsun Huang, CEO of NVIDIA

Strength of Ray Tracing: It's Right

- Monte Carlo ray tracing ultimately gives the right answer. It's the "ground truth" algorithm. [Well, ignoring polarization, diffraction, etc.]
- We can (and must) simplify any number of elements – BRDFs, light transport paths - for the sake of FPS. We simplify less each year.
- Long and short, the basic idea of ray tracing will be around a very long time.

The Dangers of Ray Tracing

The Dangers of Ray Tracing

I3D Call For Participation

- Boston, February 27-March 1, 2009
- Papers deadline: October 24, 2008
- Posters/Demos deadline: December 19, 2008

http://www.i3dsymposium.org

Ray tracing at SIGGRAPH 2008:

http://realtimerendering.com/blog

leftovers

Further Reading (Hardware)

- “Streaming Architectures and Technology Trends,” John Owens, *GPU Gems 2*
- “The Landscape of Parallel Computing Research: A View from Berkeley”
- “Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism”
- *Graphics Hardware* presentations