Lecture 4:

Graphics Performance and Characterization

EEC 277, Graphics Architecture
John Owens, UC Davis, Winter 2017

Assignment 2 Part 1 (everyone)

Explore the crossover point between geometry and rasterization.

- **Smooth shaded triangles**
- **Textured triangles (how about multitexture?)**
- **Lit triangles (complexity of lighting?)**
- **Textured, lit triangles**

Good: Have wesbench running by next Tuesday

Better: Have this test done by next Thursday

Outline

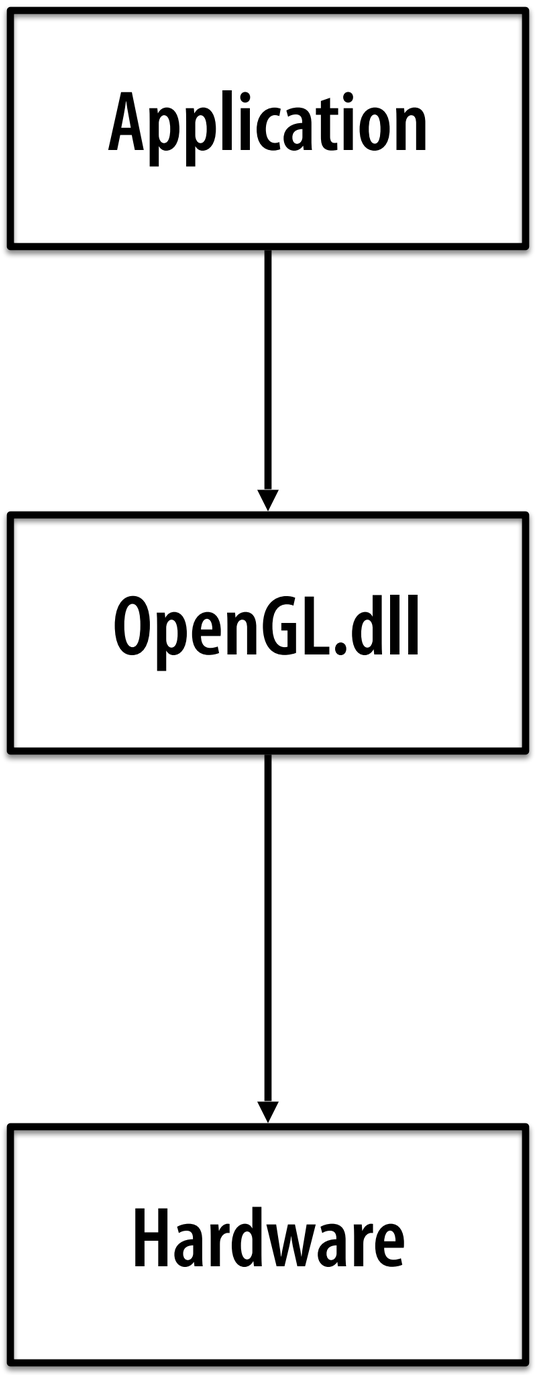
- **Tracing and quantitative analysis**
- **Applications and scenes**
- **Triangle size and depth complexity**
- **Trends, maxims and pitfalls**

Graphics Performance Analysis

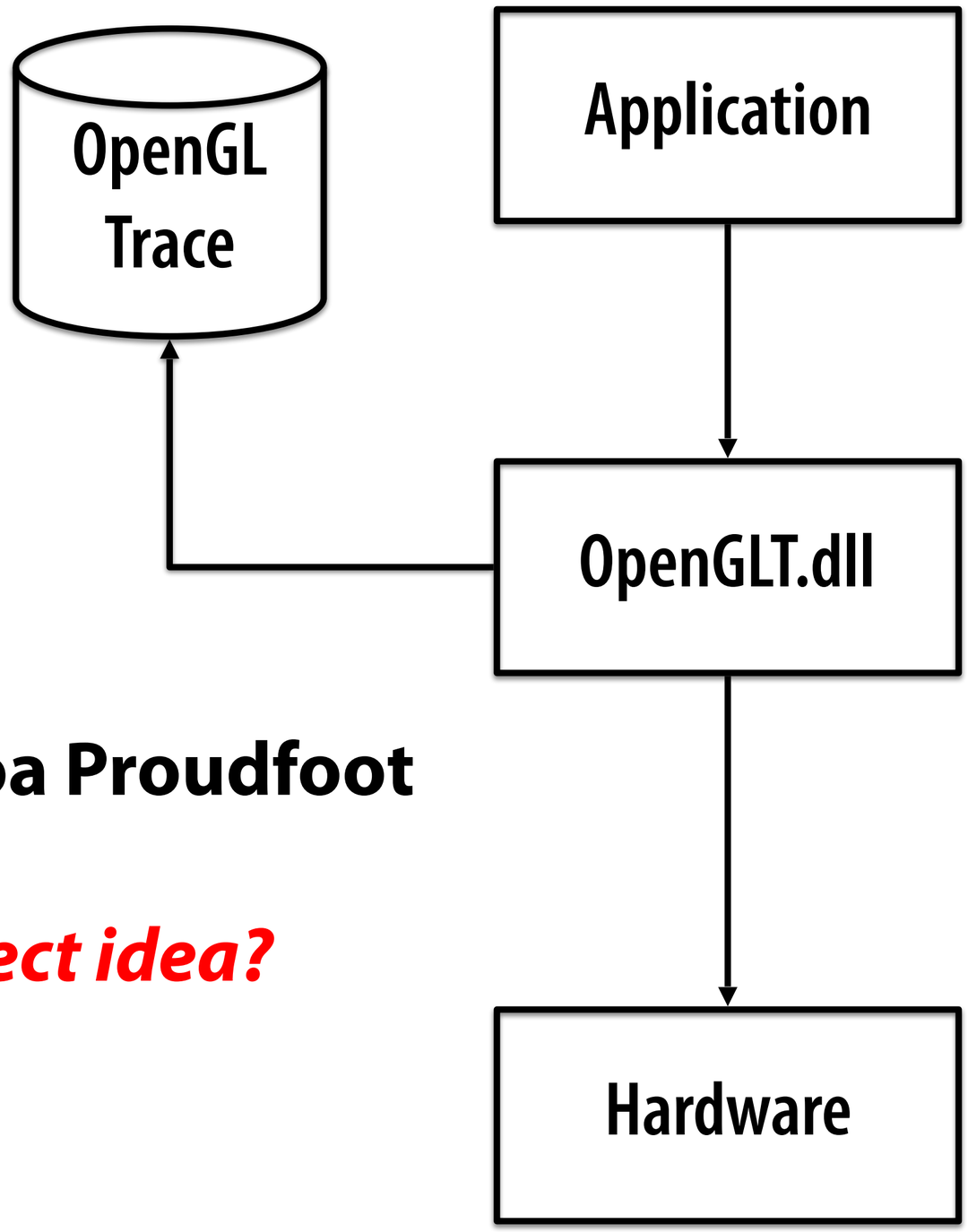
Goals:

- Characterize application workloads
- Understand system performance under workloads
- Simulate new architectures

Tracing



Tracing



glt: Kekoa Proudfoot

Project idea?

Comments

1. **Enabled by simple function pointer interface (jump tables)**
2. **Must not perturb interaction**

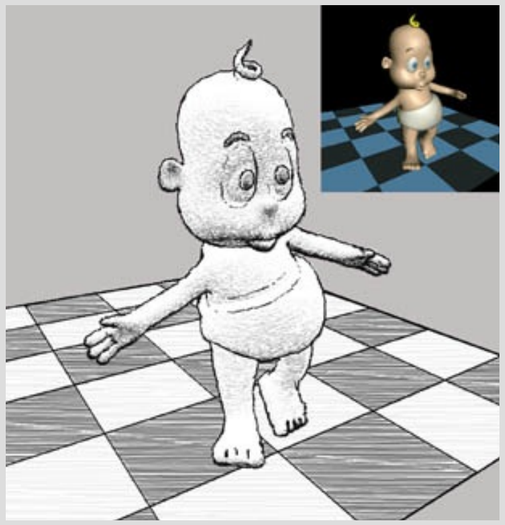
Could replace OpenGL calls with your own calls. Why useful?

https://graphics.stanford.edu/software/glt/

Tricks with Dynamic Libraries

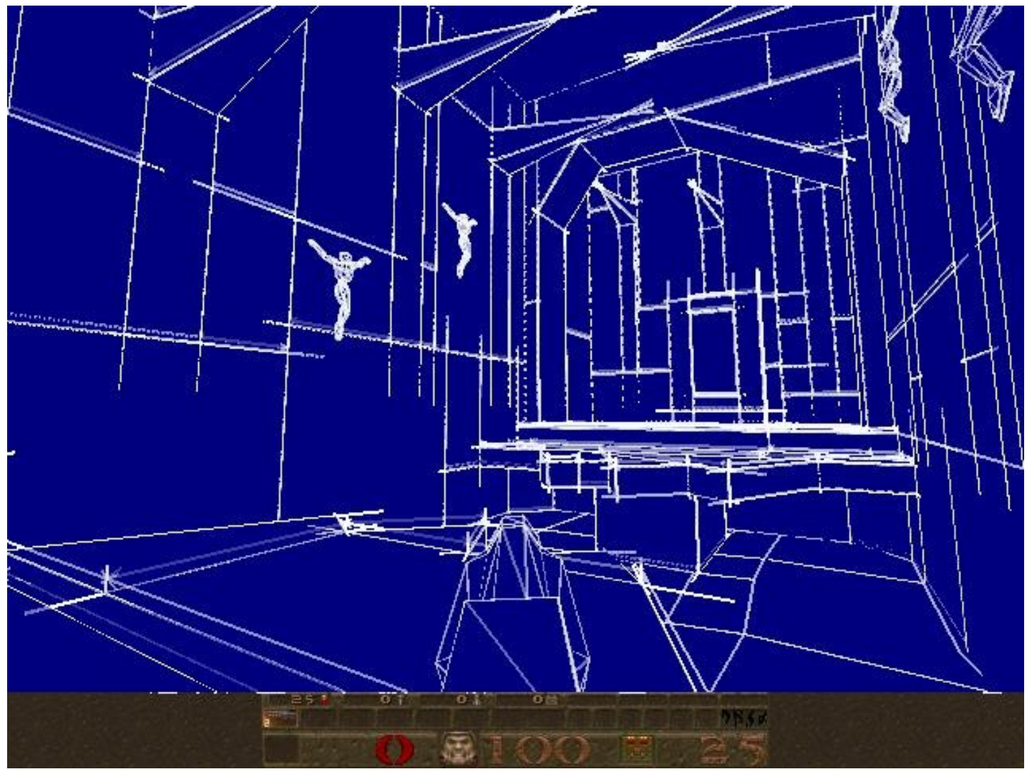
- Ability to insert GL filter is very useful

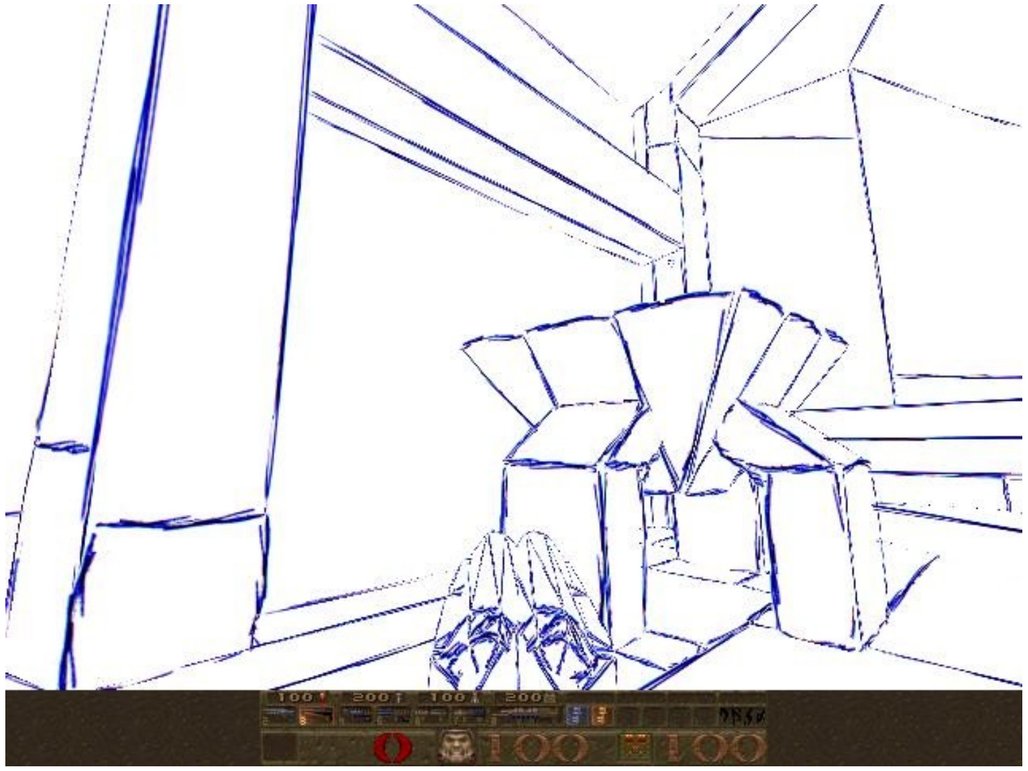
1. Convert to postscript
2. Realistic and non-photorealistic rendering
3. Debugging (application or architect)
4. Network transparent graphics
5. Stereo, rendering to tiled displays, caves, etc.
6. Regression testing
7. Reverse engineering
8. Cheating: player can turn opaque polygons into transparent polygons
9. Stealing models: capture scene geometry



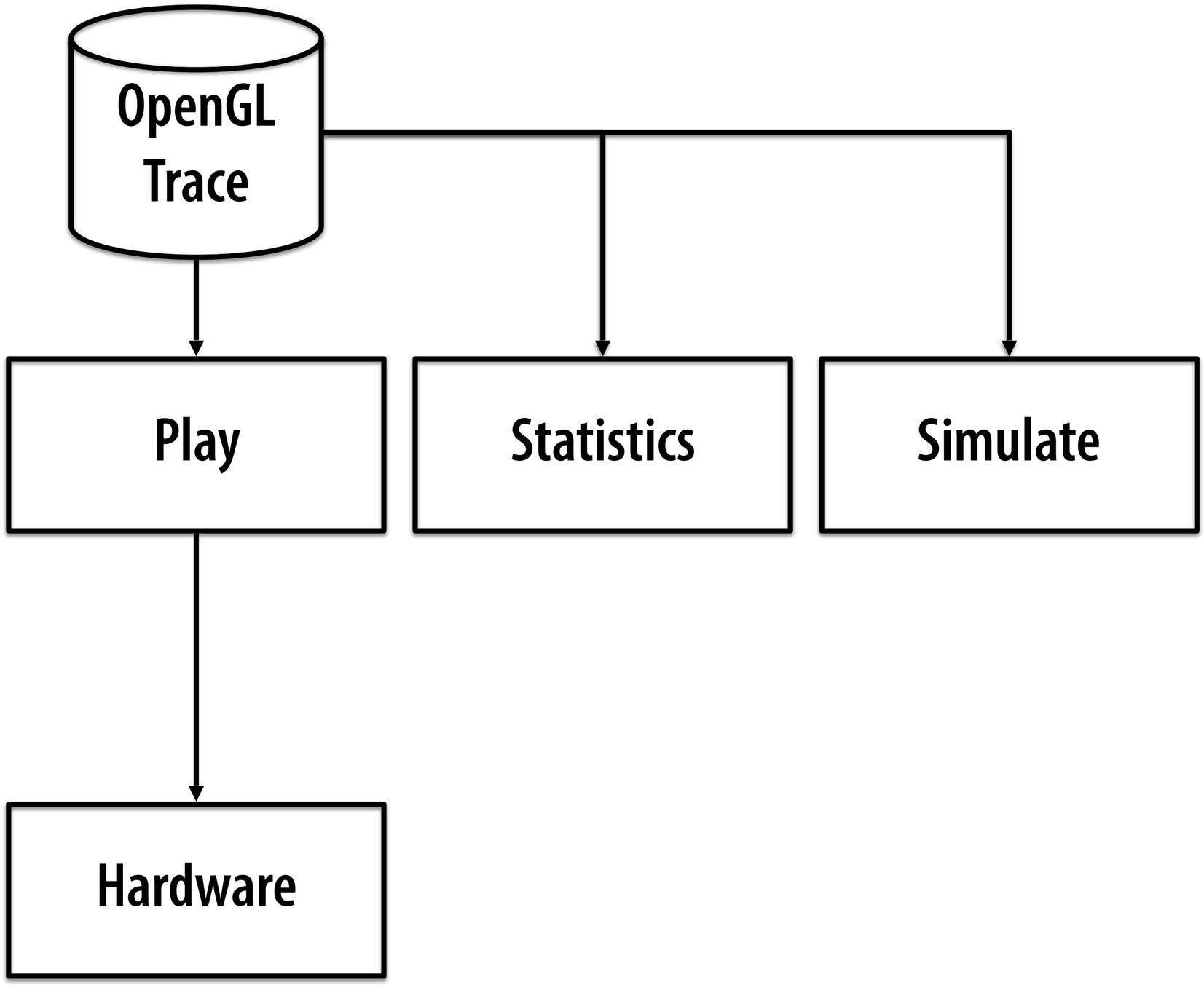
Mohr/Gleicher

HijackGL (Mohr/Gleicher, Wisconsin)

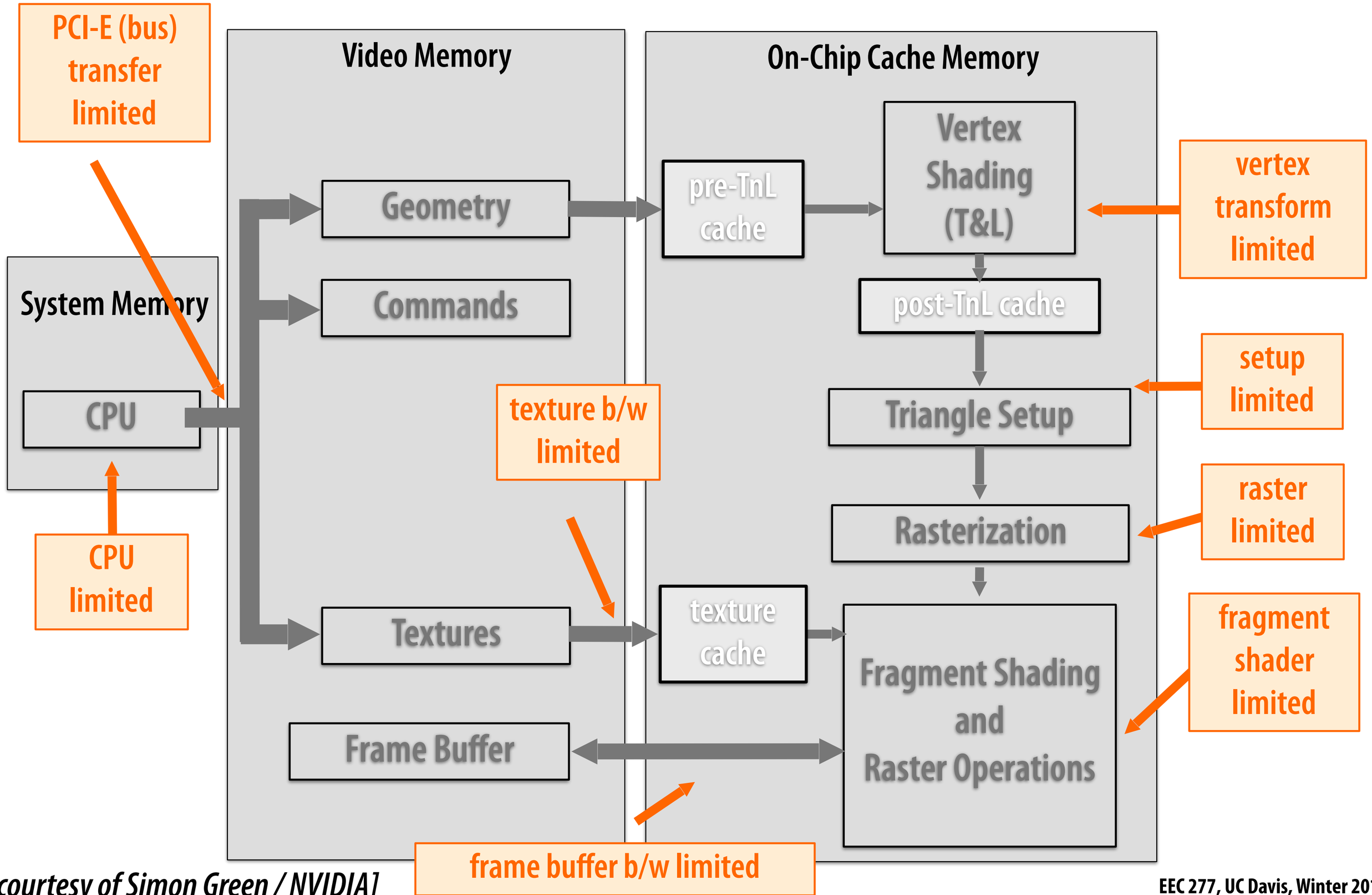




Tracing

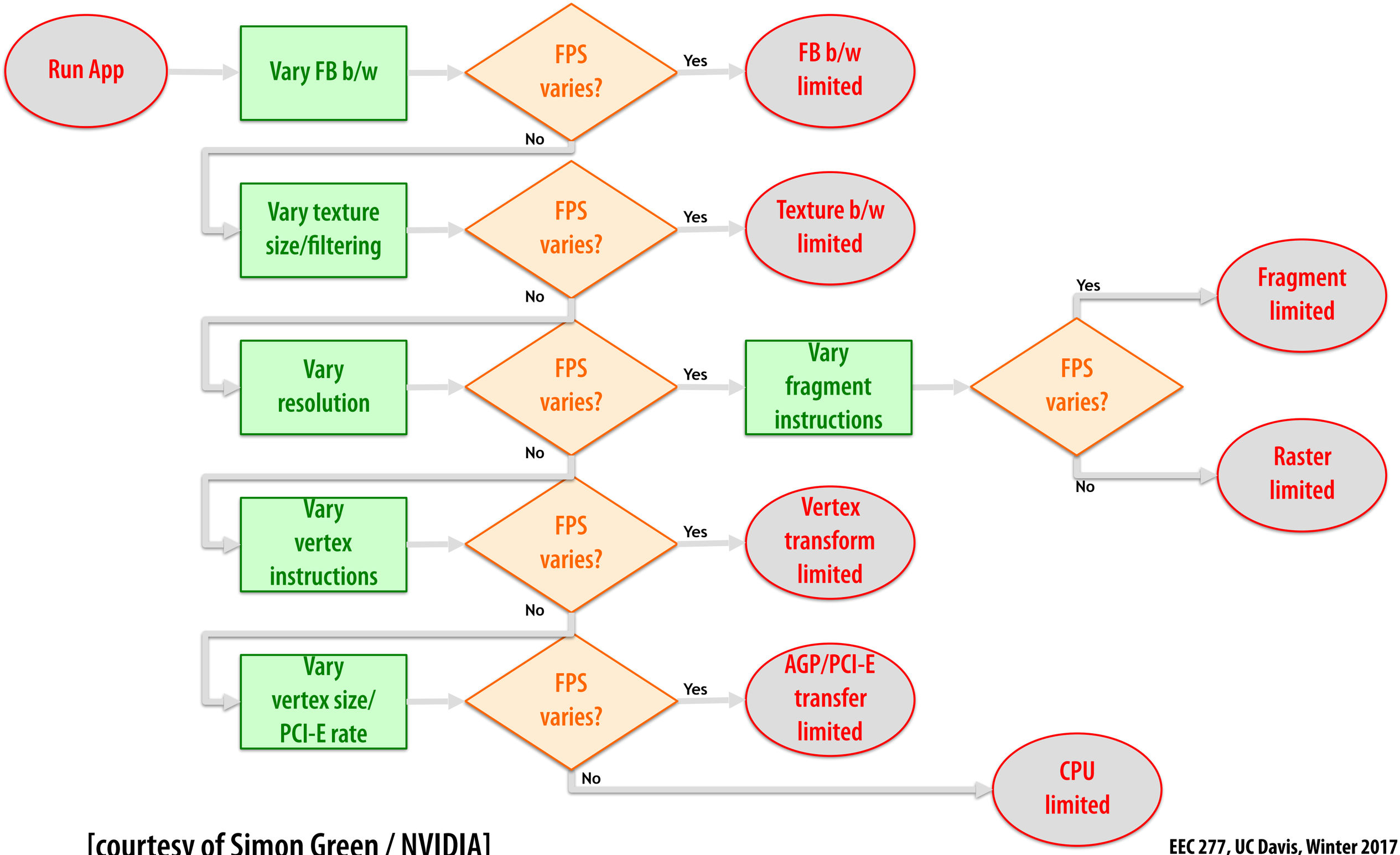


Potential Bottlenecks



[courtesy of Simon Green / NVIDIA]

Bottleneck Identification

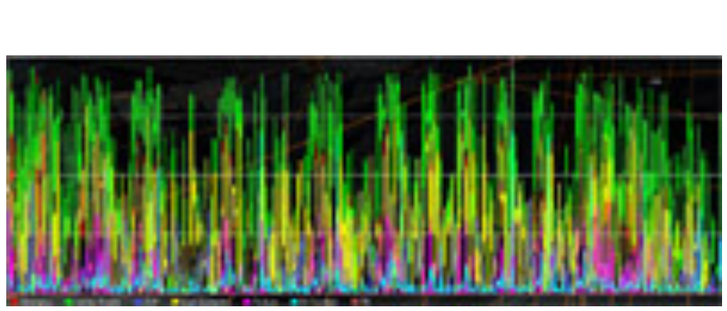


[courtesy of Simon Green / NVIDIA]

NVPerfHUD (NVIDIA)

Overlay that shows vital various statistics as the application runs

GPU Unit Utilization



Draw Call Utilization

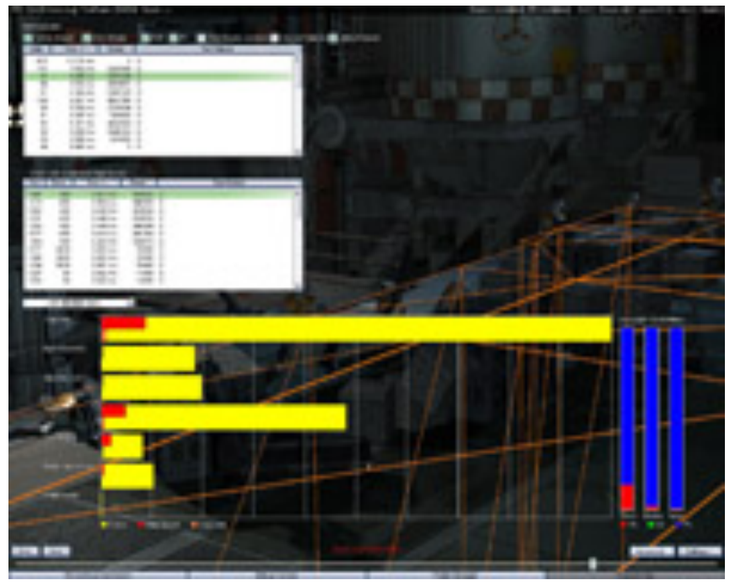
State buckets:

Vertex Shader | Pixel Shader | ROP

Calls	Time	Pixels	
852	13.735 ms	0	0:
147	7.552 ms	5325168	0:
47	4.088 ms	5894584	0:
69	2.742 ms	2605672	0:
91	2.364 ms	1247520	0:
120	2.261 ms	9607368	0:
80	1.500 ms	2278104	0:
31	1.247 ms	929520	0:
15	1.207 ms	4464568	0:
10	1.008 ms	1848440	0:
26	0.999 ms	834656	0:
46	0.947 ms	0	0:

Draw calls in selected State Bucket:

Ord	Prims	Time	Pixels	
1497	450	0.513 ms	839832	0:
1115	450	0.504 ms	840920	0:
1553	450	0.499 ms	839832	0:
1231	450	0.498 ms	839832	0:
1329	450	0.496 ms	840984	0:



Outline

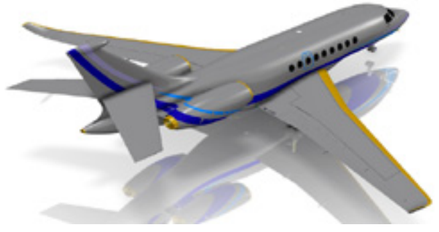
- Tracing and quantitative analysis
- Applications and scenes
- Triangle size and depth complexity
- Trends, maxims and pitfalls

Benchmarks

What is a benchmark? Why do we use them?

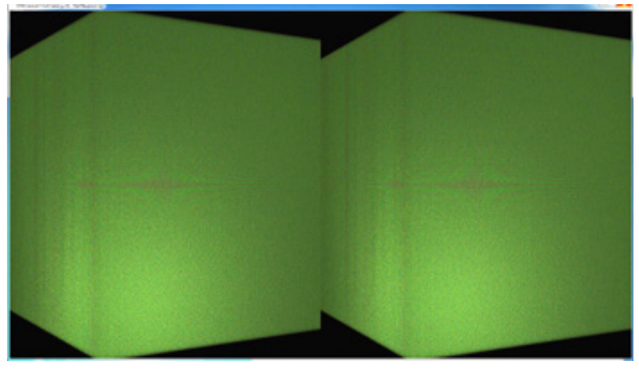
What makes a good benchmark?

SPECViewperf OpenGL Benchmark

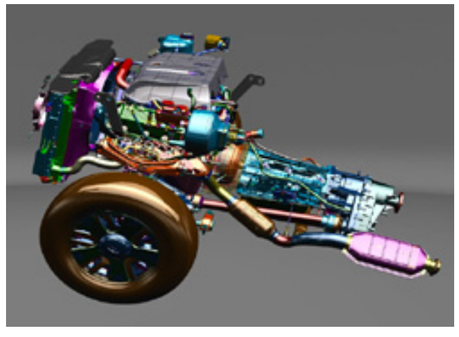


CATIA (catia-04)

Creo (creo-01)

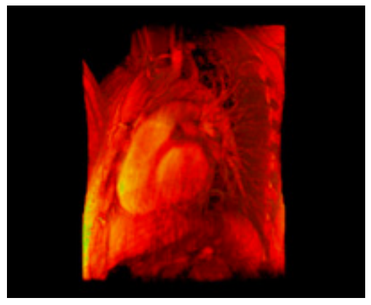


Energy (energy-01)



Siemens NX (snx-02)

Maya (maya-04)



Medical (medical-01)

Showcase (showcase-01)

Solidworks (sw-03)

SPECViewperf Factors

- It uses datasets that are designed for and used by real applications.
- It uses rendering parameters and models selected by independent software vendors (ISVs) and graphics users.
- It produces numbers based on frames per second, a measurement with which users can readily identify.
- It provides one number for each rendering path using one data set.

SPECViewperf non-factors

- Effects caused by switching primitives
- Input effects on the event loop
- User interface rendering and management
- Complex motion of multiple models
- Effects of CPU load on the graphics subsystem
- Color index visual performance
- Multi-context, multi-window effects

Benchmarks taken seriously!

http://www6.tomshardware.com/column/20030213/index.html

Columns

3D Mark 2003: The Gamers' Benchmark (?)

3D Mark 2003 - The Gamers' Benchmark (?)

Article Info

3D Mark 2003: The Gamers' Benchmark (?)

Created:
February 13, 2003

By:
Lars Weinand

Category:
Columns

Summary:
The new 3D Mark 2003 is finaly available but are we happy with it? Here are some thoughts.

The new 3D Mark 2003 is here! After the initial hype, however, users and hardware testers quickly sobered up. The CPU performance doesn't appear to play a very big role for the final result. Only the graphics card is ultimately responsible for the 3D Mark score.

And here's where the real discussion begins. Even NVIDIA is clearly distancing itself from the new release and believes that this benchmark test has no relation to reality. For one thing, the choice of game tests, beginning with the rarely played flight scenes test (Wings of Fury), is nothing close to actual practice. It goes further with the tests GT2 and GT3. The 3D engine that makes up the foundation of the tests is the same, and both tests use pixel shaders based on version v1.4. Here is an excerpt from NVIDIA's statement:

> *These tests use ps1.4 for all the pixel shaders in the scenes. Fallback versions of the pixel shaders are provided in ps1.1 for hardware that doesn't support ps1.4. Conspicuously absent from these scenes, however, is any ps1.3 pixel shaders. Current DirectX 8.0 (DX8) games, such as Tiger Woods and Unreal Tournament 2003, all use ps1.1 and ps1.3 pixel shaders. Few, if any, are using ps1.4.*

Benchmarks taken seriously!

Intel graphics drivers employ questionable 3DMark Vantage optimizations

3DMurk all over again?

by Geoff Gasior — 11:02 AM on October 12, 2009

IN THE EARLY DAYS of GPUs, application-specification performance optimizations in graphics drivers were viewed by many as cheating. Accusations were hurled with regularity, and in some cases, there was real cheating going on. Some optimizations surreptitiously degraded image quality in order to boost performance, which obviously isn't kosher. Optimizations that don't affect an application's image quality are harder to condemn, though, especially if you're talking about games. If a driver can offer users smoother gameplay without any ill effects, why shouldn't it be allowed?

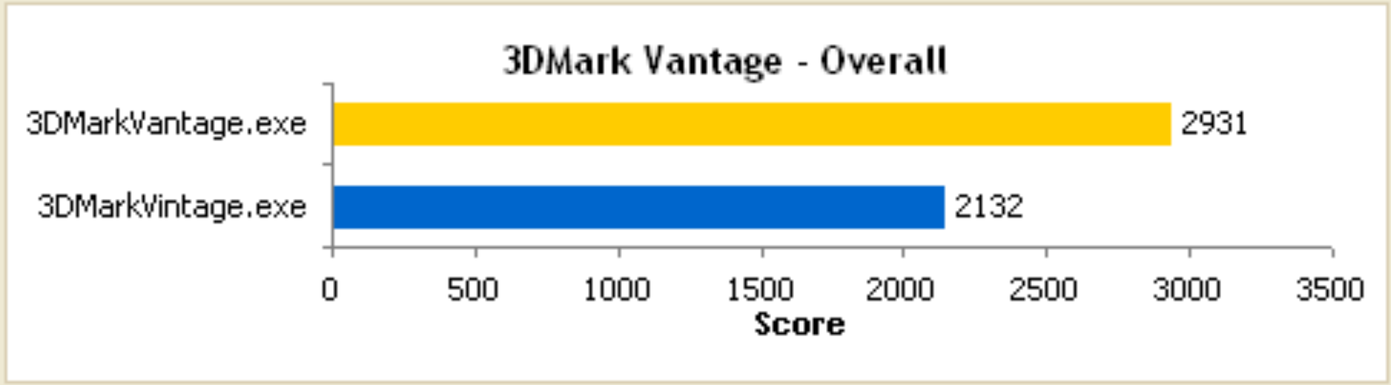
The situation gets more complicated when one considers optimizations that specifically target benchmarks. Synthetic tests don't have user experiences to improve, just arbitrary scores to inflate. Yet the higher scores achieved through benchmark-specific optimizations could influence a PC maker's choice of graphics solution or help determine the pricing of a graphics card.

Futuremark's popular 3DMark benchmark has been the target of several questionable optimizations over the years. Given that history, it's not surprising that the company has strict guidelines for the graphics drivers it approves for use with 3DMark Vantage. These guidelines, which can be viewed here (PDF), explicitly forbid optimizations that specifically target the 3DMark Vantage executable. Here's an excerpt:

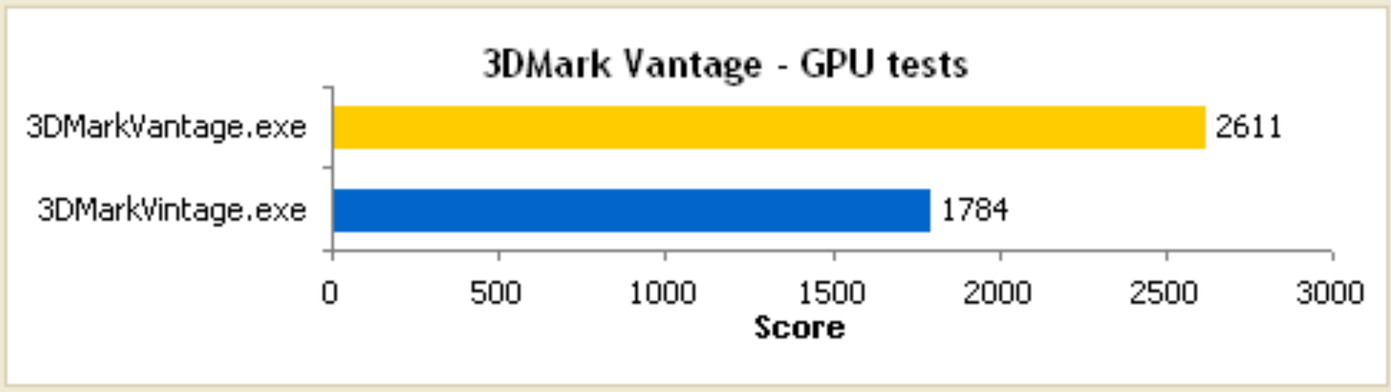
> With the exception of configuring the correct rendering mode on multi-GPU systems, it is prohibited for the driver to detect the launch of 3DMark Vantage executable and to alter, replace or override any quality parameters or parts of the benchmark workload based on the detection. Optimizations in the driver that utilize empirical data of 3DMark Vantage workloads are prohibited.

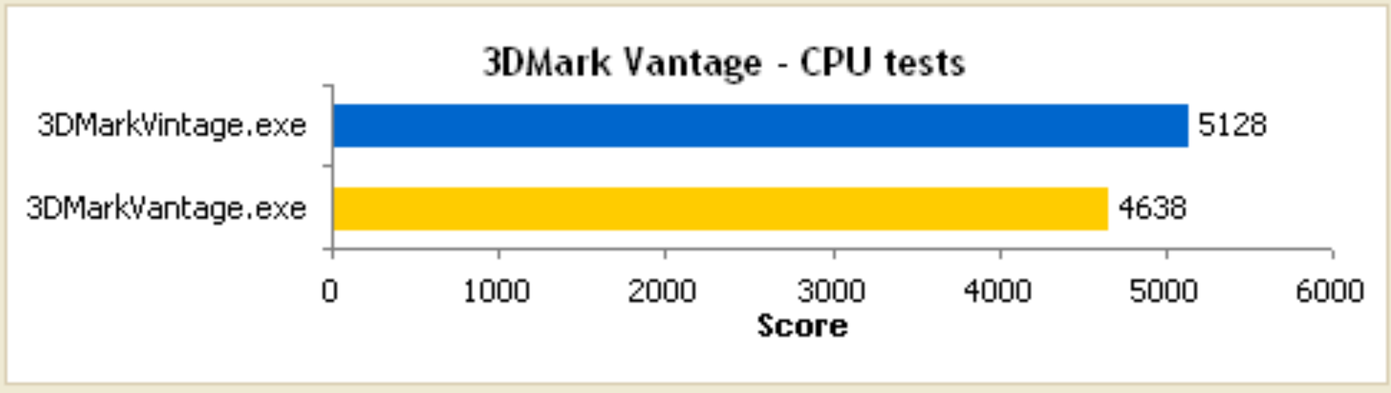
Benchmarks taken seriously!

We first ran the benchmark normally. Then, we renamed the 3DMark executable from "3DMarkVantage.exe" to "3DMarkVintage.exe". And—wouldn't you know it?—there was a substantial performance difference between the two.



Our system's overall score climbs by 37% when the graphics driver knows it's running Vantage. That's not all. Check out the CPU and GPU components of the overall score:





Benchmarks taken seriously!

http://www.anandtech.com/show/7384/state-of-cheating-in-android-benchmarks

They're (Almost) All Dirty: The State of Cheating in Android Benchmarks

by Anand Lal Shimpi & Brian Klug on October 2, 2013 12:30 PM EST

Thanks to AndreiF7's excellent work on discovering it, we kicked off our investigations into Samsung's CPU/GPU optimizations around the international Galaxy S 4 in July and came away with a couple of conclusions:

1) On the Exynos 5410, Samsung was detecting the presence of certain benchmarks and raising thermal limits (and thus max GPU frequency) in order to gain an edge on those benchmarks, and

2) On both Snapdragon 600 and Exynos 5410 SGS4 platforms, Samsung was detecting the presence of certain benchmarks and automatically driving CPU voltage/frequency to their highest state right away. Also on Snapdragon platforms, all cores are plugged in immediately upon benchmark detect.

The table below is a subset of devices we've tested, the silicon inside, and whether or not they do a benchmark detect and respond with a max CPU frequency (and all cores plugged in) right away:

I Can't Believe I Have to Make This Table								
Device	SoC	Cheats In						
		3DM	AnTuTu	AndEBench	Basemark	Geekbench	GFXB	Vellamo

What's Interesting to Measure?

Primitive type (tris, quads, meshes, etc.)

Depth complexity (F = dI)

Immediate mode vs. retained mode

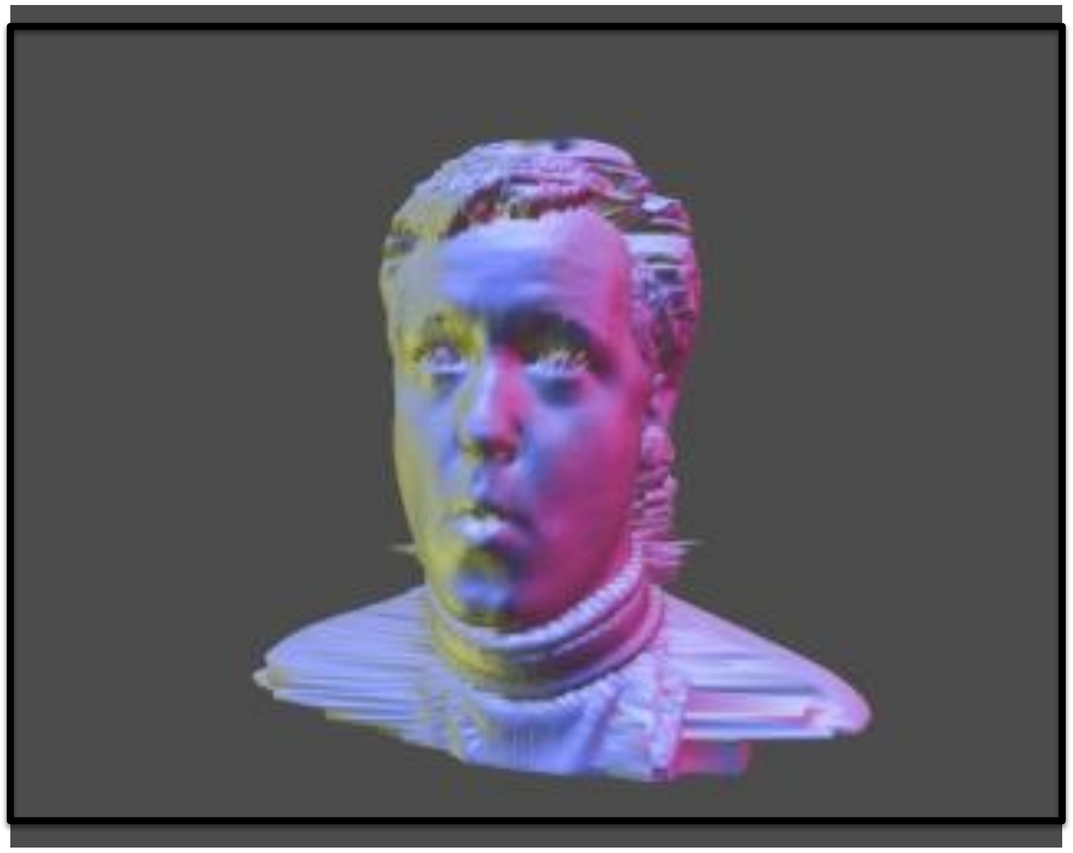
Triangle size

Triangle size variance

Cull/clip efficiency

Shading strategy (color? lights? texture?)

Scenes: Head (240 frames)



Vertices	60104
Triangles (3D)	59592
Triangles (2D)	24884
Fragments	263369
Image	1024×768

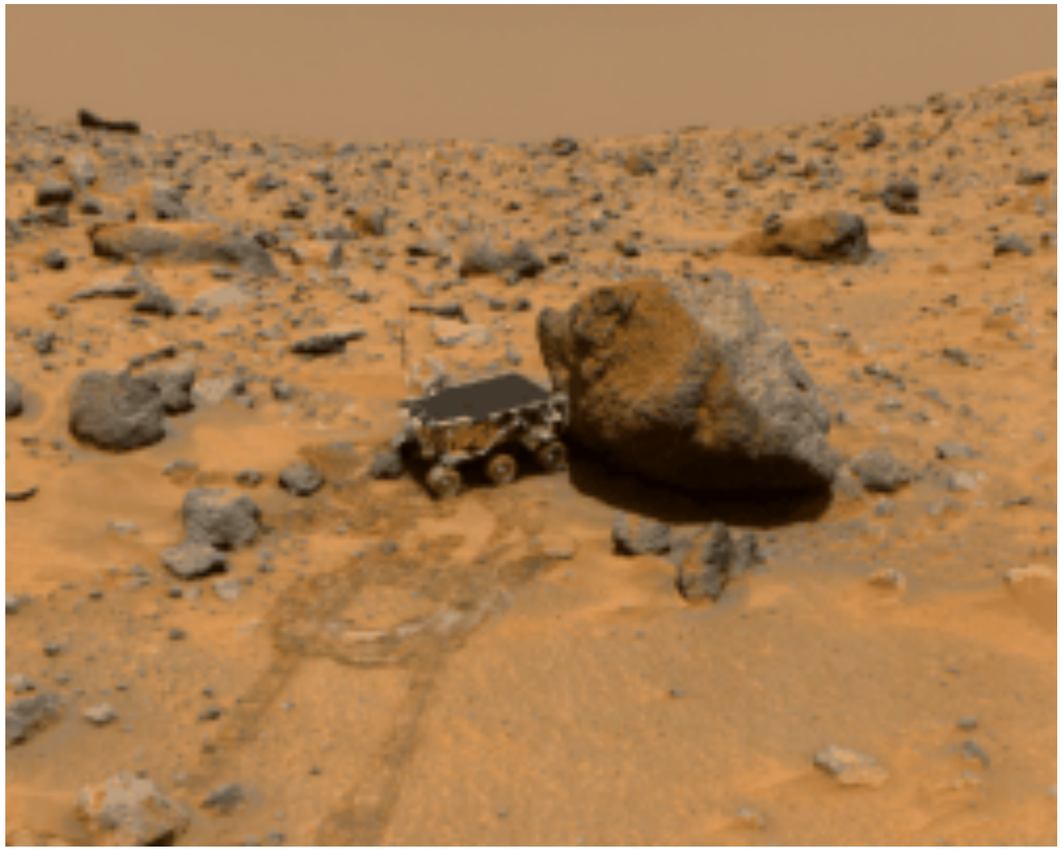
```
60104       glNormal3fv
60104       glVertex3fv
803         glVertex2fv
722         glLoadMatrixf
720         glMultMatrixf
481         glColor3fv
452         glBegin
452         glEnd
241         glClear
240         gltSwapBuffers
240         glCallList
220         glEndList
220         glNewList
124         glTranslatef
19          glLightfv
9           glEnable
6           glPixelStorei
2           glClearColor
2           glDrawBuffer
2           glLightModelf
2           glMaterialfv
2           glMatrixMode
2           glViewport
1           gltPad
```

Scenes: Light (101 frames)

Vertices	1800116
Triangles (3D)	900058
Triangles (2D)	106503
Fragments	1818726
Image	1024×768

```
1800117     glColor3fv
1800116     glVertex3fv
15001       glBegin
15001       glEnd
400         glRotatef
202         glPopMatrix
202         glPushMatrix
101         gltSwapBuffers
101         glCallList
101         glClear
101         glLoadMatrixf
100         glTranslatef
5           glFinish
4           glLoadIdentity
3           glMatrixMode
2           glEnable
2           glPolygonMode
1           gltPad
1           gltCreateContext
1           gltMakeCurrent
1           glClearColor
1           glClearIndex
1           glColorMask
1           glDeleteLists
```

Scenes: QTVR (734 frames)



Vertices	145.8
Triangles (3D)	116.6
Triangles (2D)	94.2
Fragments	786431
Image	1024×768

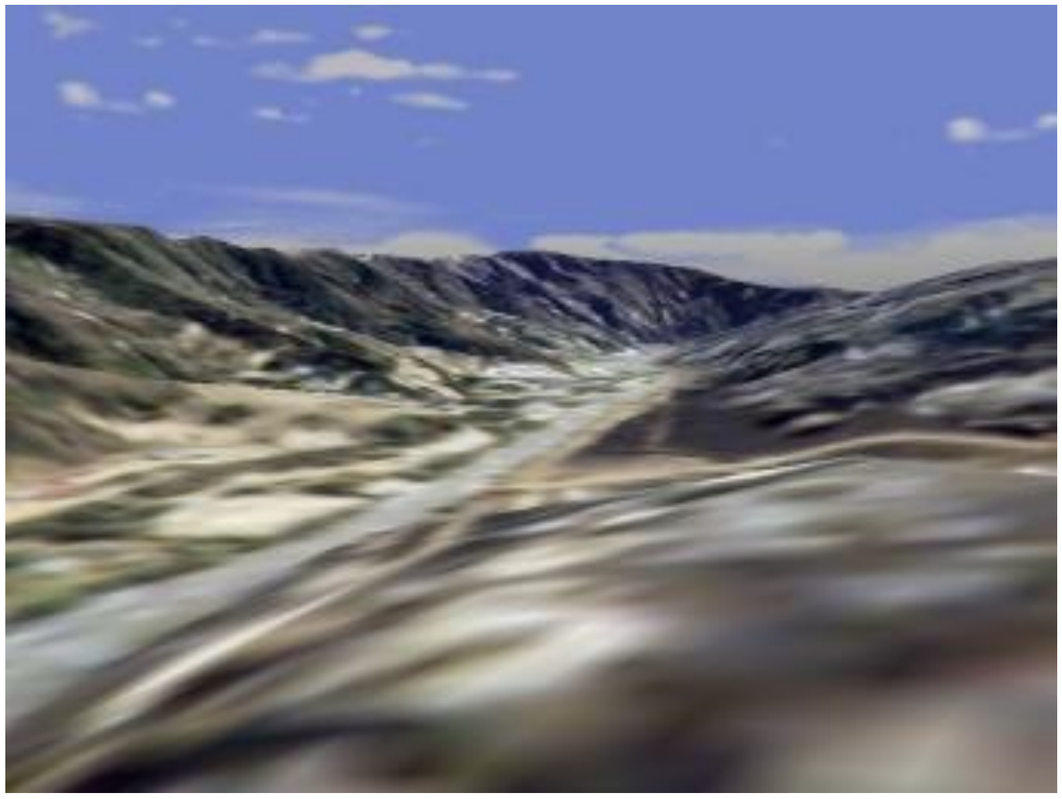
```
109750      glTexCoord2fv
109750      glVertex3fv
54875       glColor4bv
11231       glBindTextureEXT
10975       glBegin
10975       glEnd
1470        glFinish
1468        glLoadIdentity
1468        glMatrixMode
1468        glRotatef
1152        glTexImage2D
768         glTexParameteri
734         gltSwapBuffers
734         glMultMatrixd
734         glPopMatrix
734         glPrioritizeTexturesEXT
734         glPushMatrix
733         gltPad
6           glPixelStorei
2           glViewport
1           gltCreateContext
1           gltMakeCurrent
1           glDepthFunc
1           glDisable
```

Scenes: Town (1338 frames)

Vertices	*4326.8*
Triangles (3D)	*2535.3*
Triangles (2D)	*939.0*
Fragments	*1353892*
Image	*1024×768*

```
738541      glVertex3fv
728673      glTexCoord2fv
224682      glColor4fv
206474      glNormal3fv
201074      glCallList
180574      glBegin
180574      glEnd
168356      glBindTextureEXT
22659       glEnable
21150       glMaterialfv
20557       glDisable
9622        glShadeModel
5706        glPopMatrix
5706        glPushMatrix
4216        glBlendFunc
3478        glMatrixMode
3164        glLoadIdentity
3010        glDepthMask
2546        glAlphaFunc
2546        glMultMatrixf
2105        glTexEnvf
1992        gltPad
1676        glEndList
1676        glNewList
```

Scenes: Flight (123 frames)



Vertices	*3932.8*
Triangles (3D)	*2843.3*
Triangles (2D)	*553.0*
Fragments	*1004604*
Image	*1024×768*

```
588499      glVertex3fv
565531      glNormal3fv
487229      glTexCoord2fv
94632       glBegin
94632       glEnd
26178       glColor4fv
1985        glEnable
1971        glDisable
1368        glPopMatrix
1368        glPushMatrix
1338        glBindTextureEXT
1331        glMultMatrixf
1264        glMaterialfv
945         glMaterialf
646         glShadeModel
370         glMatrixMode
321         glCullFace
315         glAlphaFunc
307         glLoadIdentity
256         glTexParameteri
252         glVertex2fv
218         glTexImage2D
184         glViewport
174         gltPad
```

Scenes: Quake (330 frames)

Vertices	*3627.0*
Triangles (3D)	*1801.5*
Triangles (2D)	*937.4*
Fragments	*1855471*
Image	*1024×768*

```
869677      glTexCoord2f
531658      glVertex3fv
528161      glVertex3f
351303      glColor3f
221434      glTexCoord2fv
182997      glBegin
182997      glEnd
88470       glColor3ubv
31292       glVertex2f
10657       glRotatef
9843        glBindTextureEXT
5160        glTranslatef
4285        glTexEnvf
4188        glDisable
3643        glShadeModel
3532        glEnable
3037        glBlendFunc
3036        glDepthMask
3019        glPopMatrix
3019        glPushMatrix
1821        glScalef
1606        glTexImage2D
1518        glColor3ub
1300        glLoadIdentity
```

Outline

- Tracing and quantitative analysis
- Applications and scenes
- Triangle size and depth complexity
- Trends, maxims and pitfalls

Fragment Formula

Performance: T a-pixel triangles per frame

$$a \equiv \frac{F}{T} \Rightarrow F = aT$$

Parameters

T = Number of triangles

a = Average area of a triangle

F = Number of fragments

Per-frame and per-second related by fps

Triangle Area Implications

The average triangle area a represents a balance point between the floating point computation needed to process a triangle independent of pixel area, and the framebuffer fill capacity

Implications:

Triangles with average number of pixels greater than a typically will render at a rate less than T, because the triangles are fill-dominated.

Triangles smaller than a pixels will render at a rate no faster than T, as such triangles are geometry-limited.

Triangle Size vs. Time (SGI)

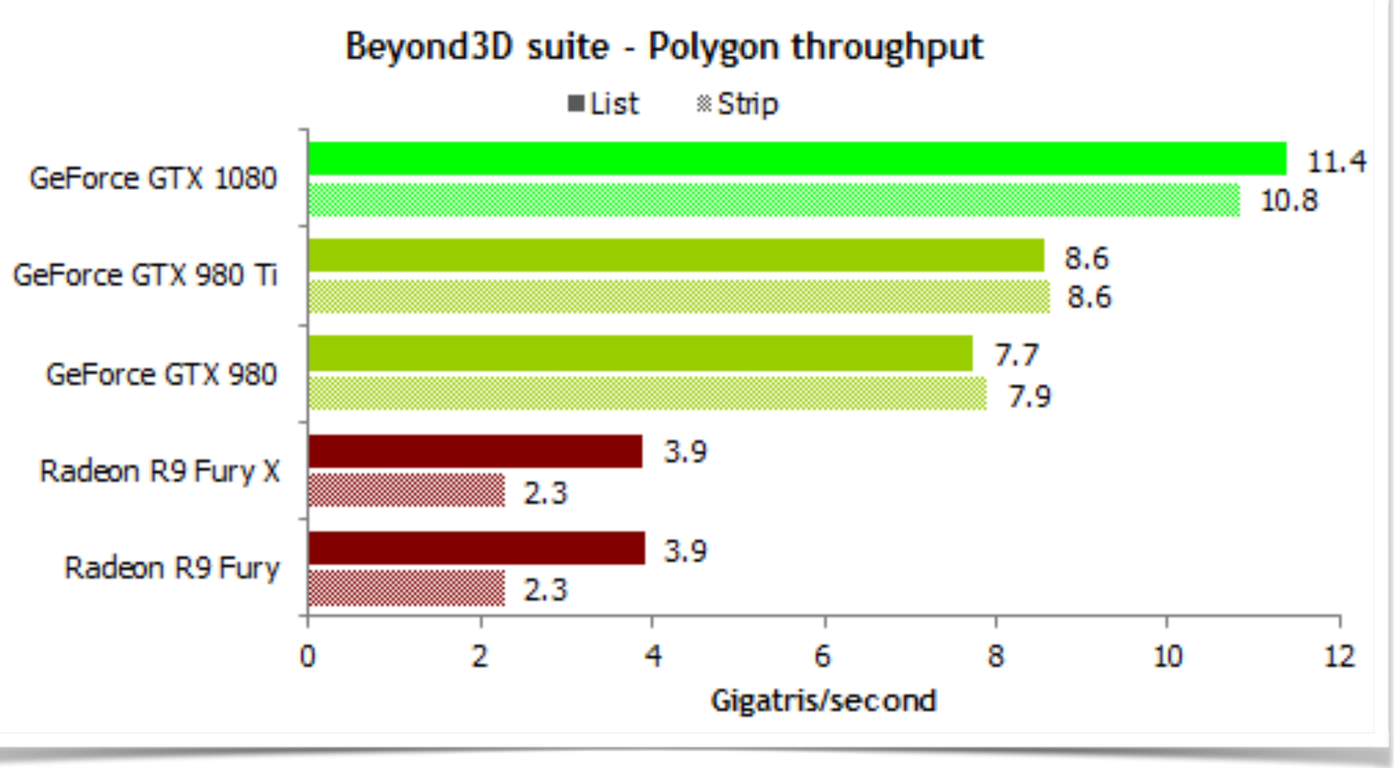
- Peak fill rates

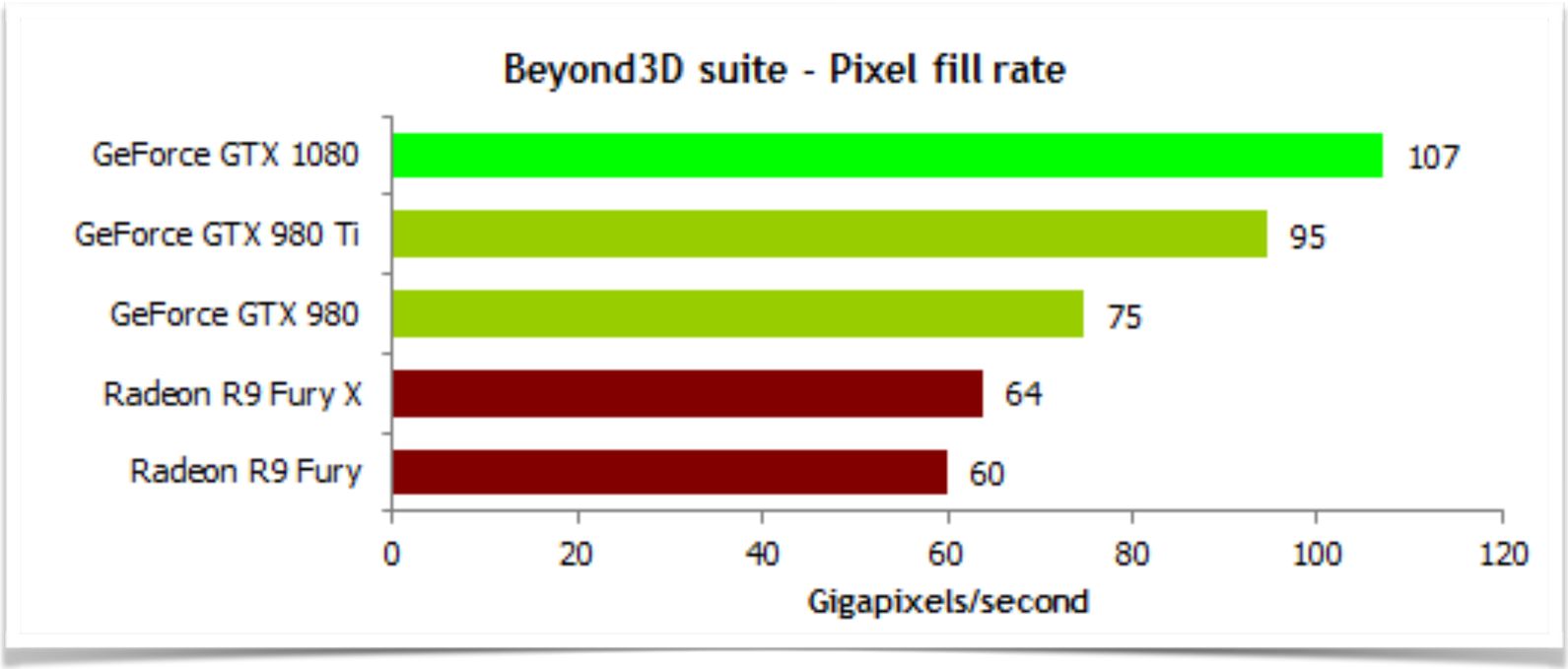
Year	*Product*	*F*	*T*	*a*
1984	*Iris 2000*	*46M*	*10K*	*4600*
1988	*GTX*	*80M*	*135K*	*592*
1992	*RE*	*380M*	*2M*	*190*
1996	*IR*	*1000M*	*12M*	*83*

Triangle Size vs. Time (NVIDIA)

Season	Product	F	T	a
2H97	Riva 128	20M	3M	6.67
1H98	Riva ZX	31M	3M	10.33
2H98	Riva TNT	50M	6M	8.33
1H99	TNT2	75M	9M	8.33
2H99	GeForce	120M	15M	8.00
1H00	GeForce2	200M	25M	8.00
2H00	NV16	250M	31M	8.06
1H01	NV20	500M	30M	16.67
2H01	NV25	1200M	75M	16.00
2H02	NV30	2000M	187.5M	10.67
2H04	NV40	6400M	600M	10.67
1H05	G70	6880M	860M	8.00
2H06	G71	10400M	1300M	8.00

More recent grain-of-salt numbers

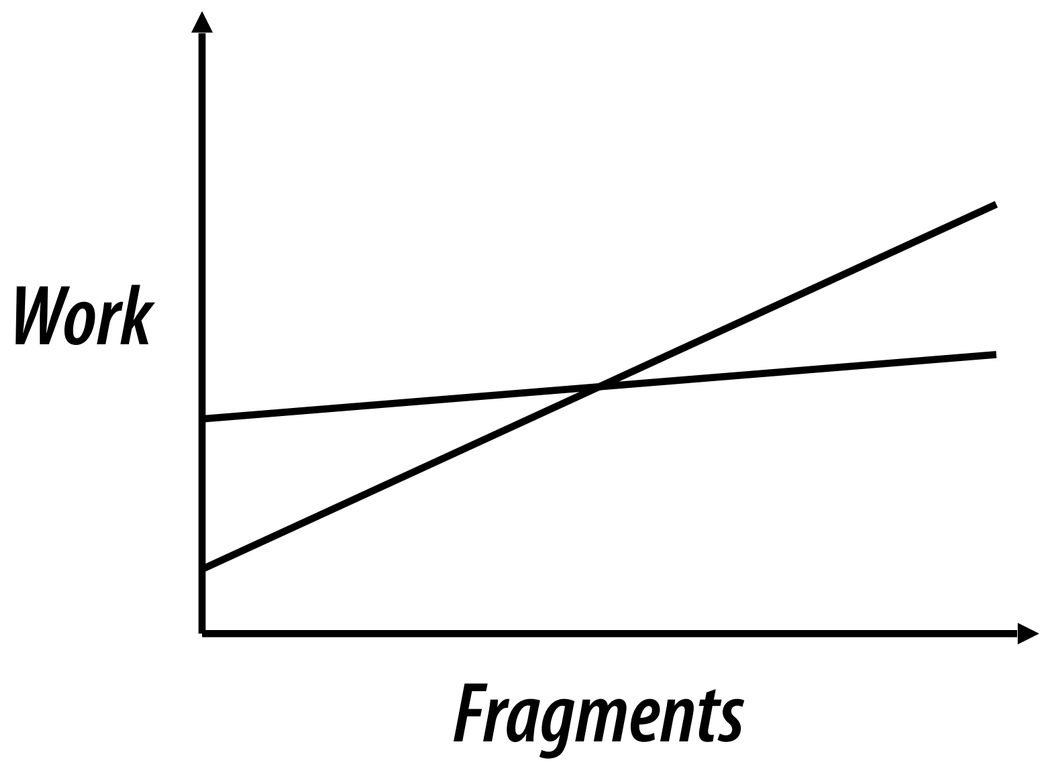




http://techreport.com/review/30281/nvidia-geforce-gtx-1080-graphics-card-reviewed

Triangle Area Histogram Implications

- Motivate two types of rasterization
- Large triangles = amortize the cost of setup
  - Maximum per-triangle; minimum per-fragment
- Small triangles
  - Minimize the cost of producing a few pixels



Z-buffer Reads and Writes

```
if(fragment.z < z[fragment.x][fragment.y]){
  c[fragment.x][fragment.y]=blend(fragment);
  z[fragment.x][fragment.y]=fragment.z;
}
```

With *n* fragments, how many writes?

1 + 1/2 + 1/3 + 1/4 … 1/*n*

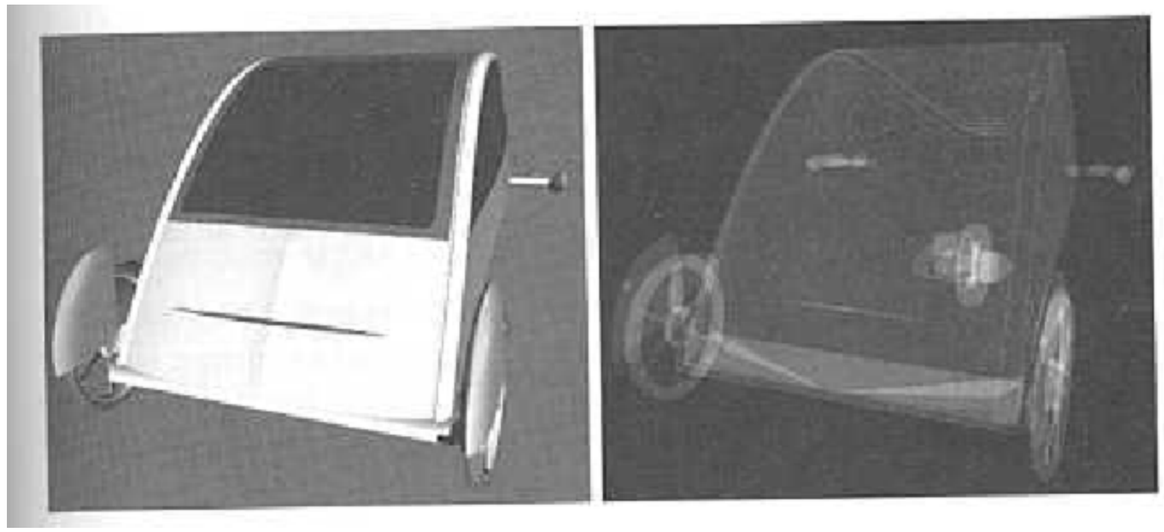
Knuth: Analysis of Algorithms

***H*(*n*) : Harmonic numbers; asymptotically ~ log(*n*)**

(lim *n*->inf *H*(*n*) = ln(*n*) + gamma)

Best case: 1; Worst case: *n*; Random case for d=4 is ~2

Visualizing Depth Complexity



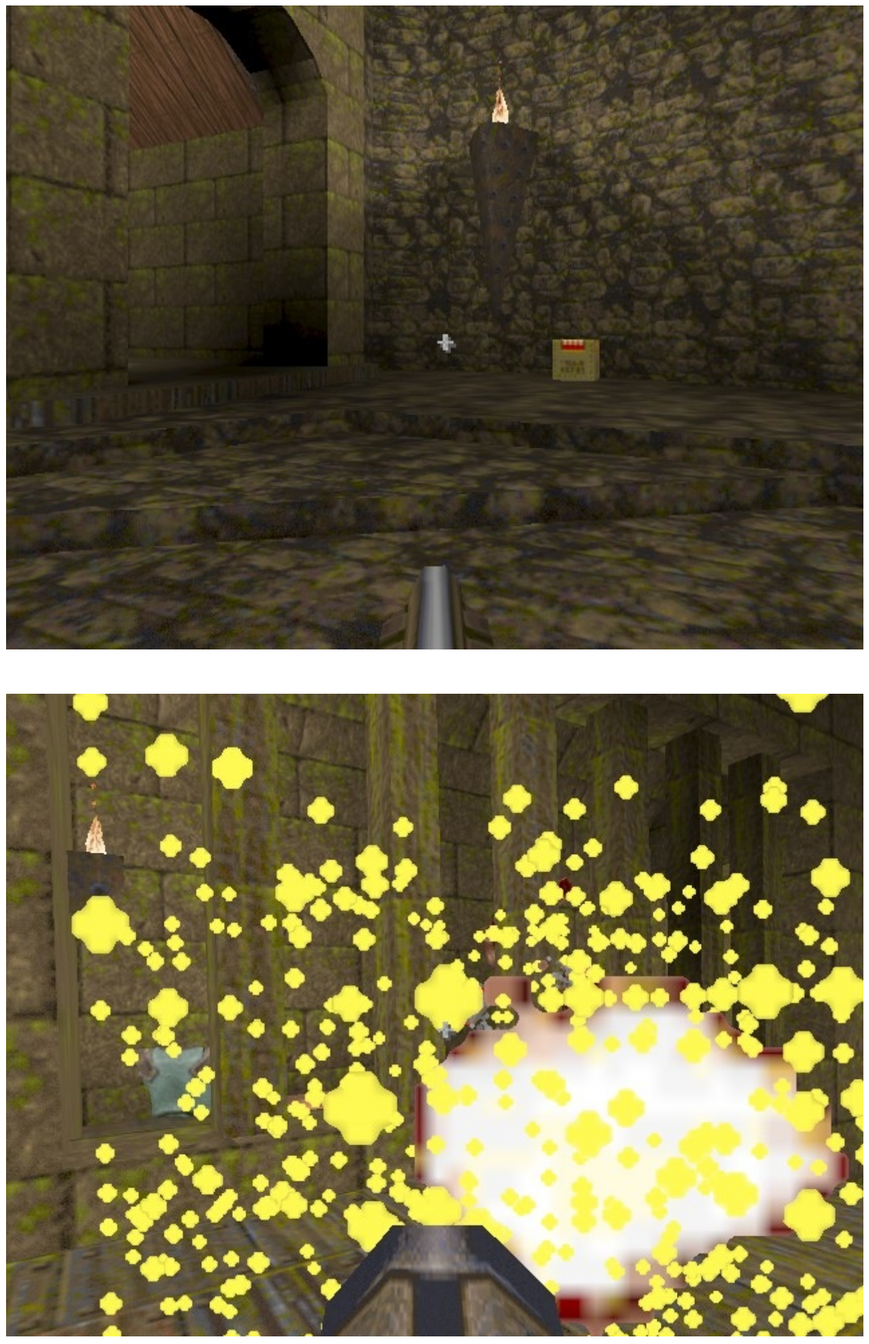
Use OpenGL!

1. Disable z-buffer
2. glBlendFunc(GL_ONE, GL_ONE)

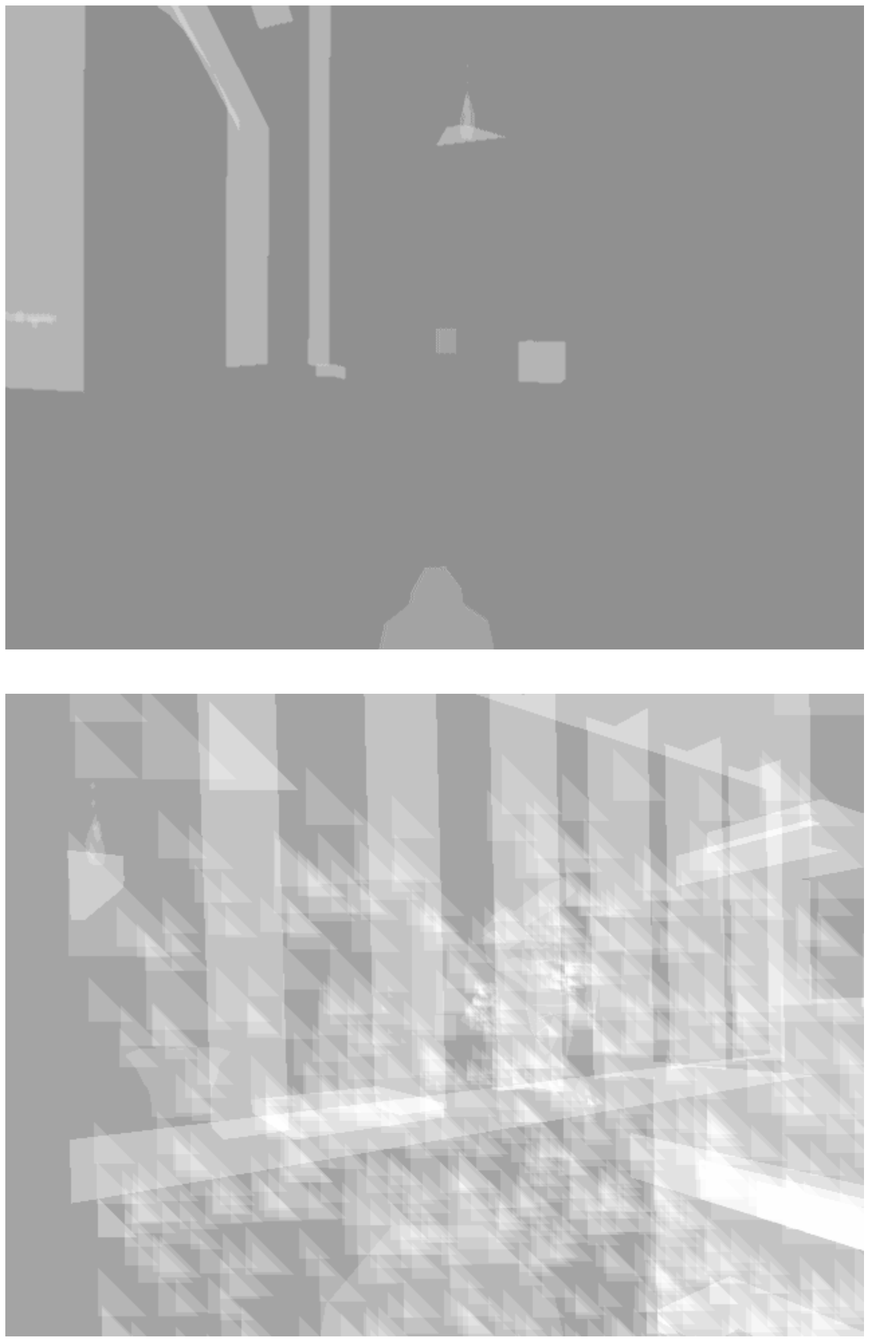
   $c = r_{src} * src\_func + r_{dest} * dest\_func$
3. Clear image to black
4. Render all objects with color (0, 0, 1/256)

How about how many *passed* the depth test?

Depth Complexity (Quake)



Color



Depth Complexity

Depth Complexity is Bounded

- High-quality rendering
  - Movie set analogy (don't build parts of the environment that can't be seen)
- Well-written apps have low depth complexity
  - Culling and level-of-detail strategies (Performer)

Adds significant complexity to the application

Outline

- Tracing and quantitative analysis
- Applications and scenes
- Triangle size and depth complexity
- Trends, maxims and pitfalls

Maxims and Pitfalls (Pat Hanrahan)

Don’t design for last year’s scenes

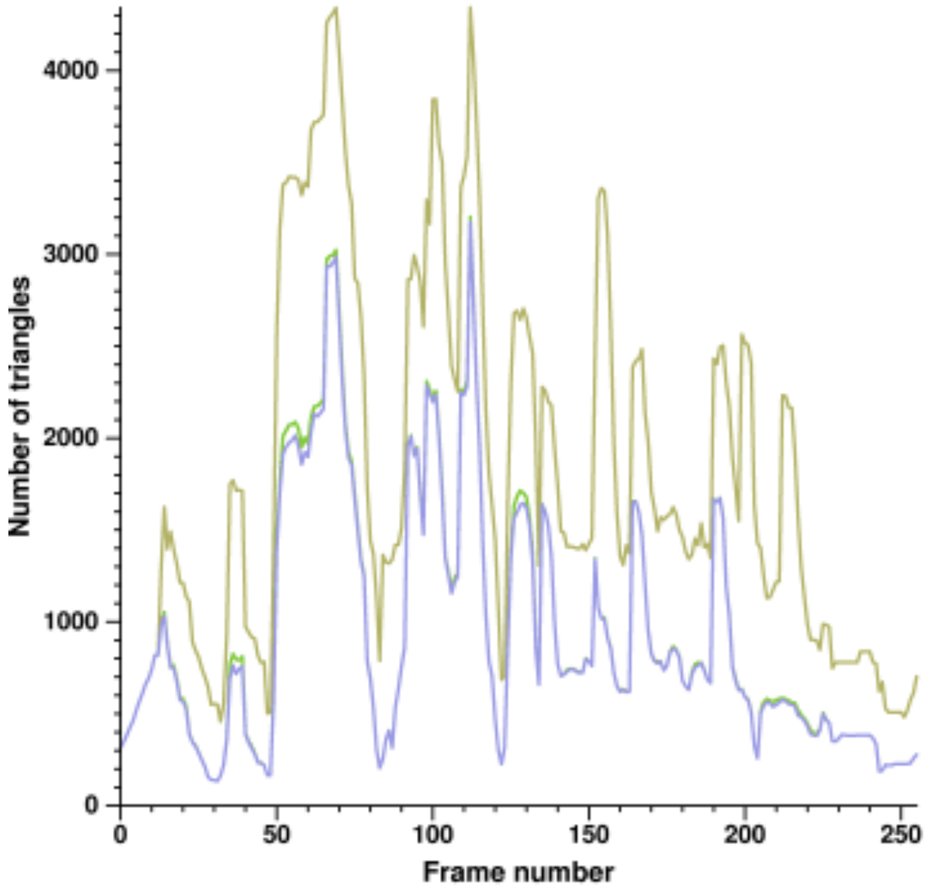
Old benchmarks may not use new features; this presents a challenge since new systems may not necessarily accelerate old applications

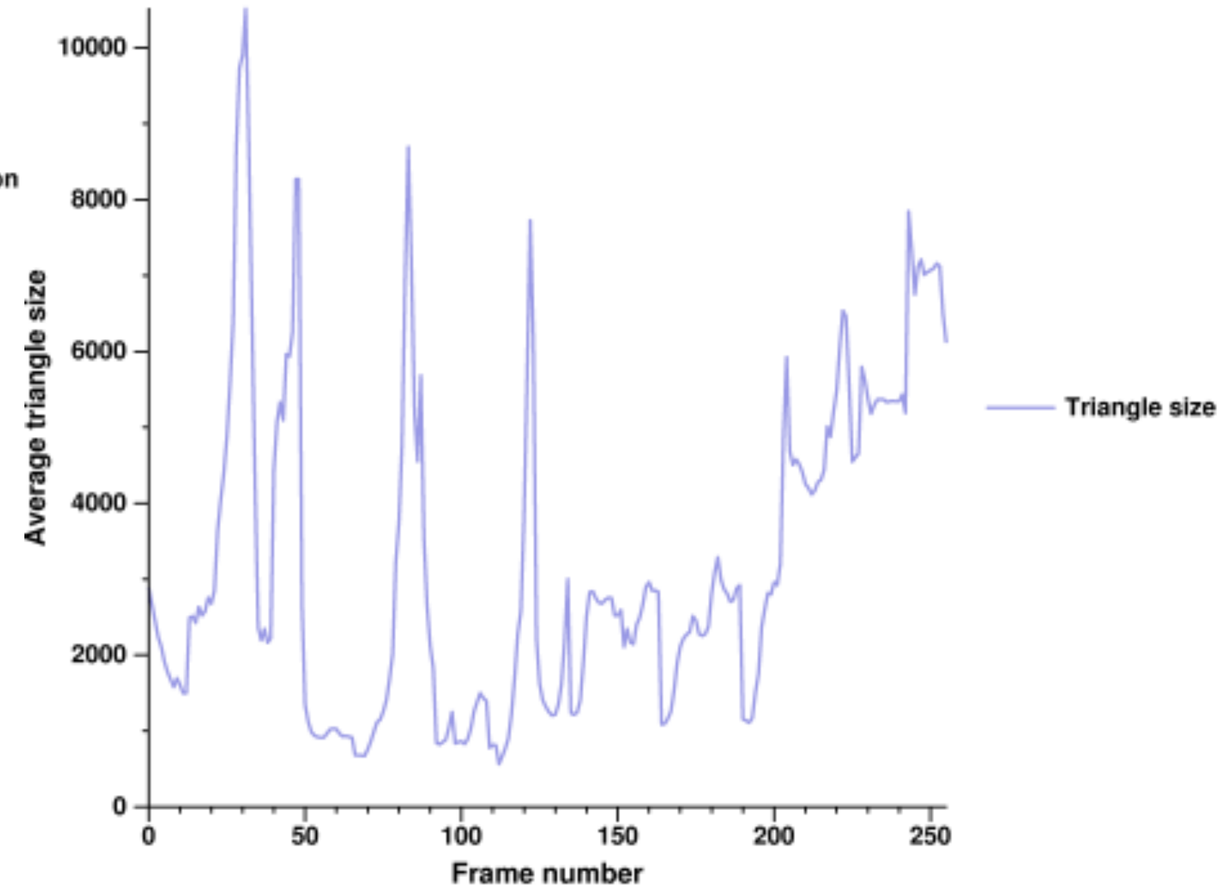
Biggest challenge is balancing the system

Very difficult to simultaneously achieve both peak fill and geometry rates

Don’t evaluate systems using single-frames; use sequences

Interframe Variability





Fill Rates

- Need a minimum fill rate (d=1) to be interesting
  - For example: VR has high frame rates and hence requires high fill rates
  - Providing high fill rates has been the major challenge to graphics architects

Next lecture

Programmability (T Jan 24)