Real-Time Graphics Architecture

Kurt Akeley

Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

Performance Analysis and Characterization

Topics

1. Tracing and quantitative analysis
2. Applications and scenes
3. Triangle size and depth complexity
4. Trends, maxims and pitfalls

Readings

Required

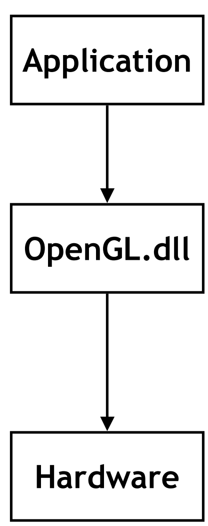
1. J. C. Dunwoody and M. Linton, Tracing interactive 3D graphics programs
2. M. Deering, Data complexity for virtual reality: Where did all the triangle go?

Graphics Performance Analysis

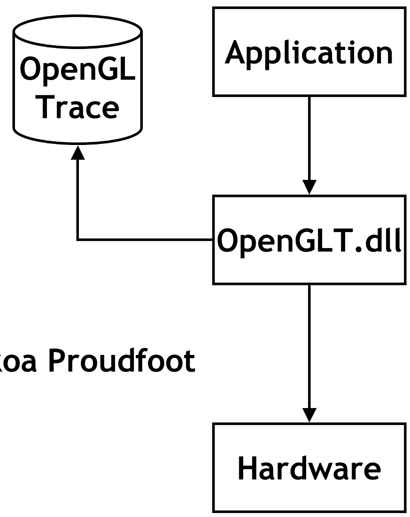
Goals:

1. Characterize application workloads
2. Understand system performance under workloads
3. Simulate new architectures

Tracing



Tracing



glt: Kekoa Proudfoot

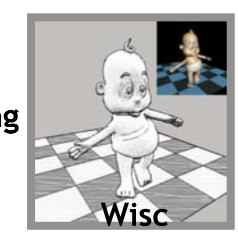
Comments

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

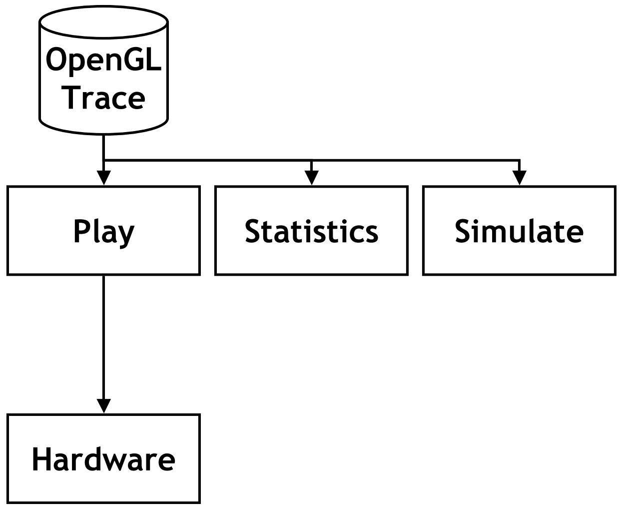
Tricks with Dynamic Libraries

Ability to insert GL filter is very useful

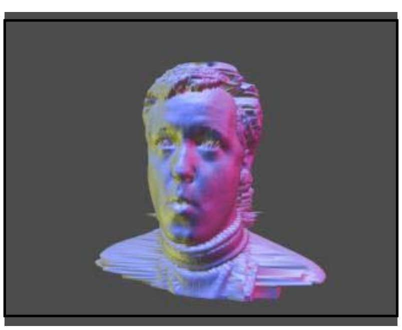
1. Convert to postscript
2. Realistic and non-photorealistic rendering
3. Debugging (application or architect)
4. Network transparent graphics
5. Stereo, rendering to tiled displays, caves, etc.
6. Regression testing
7. Reverse engineering
8. Cheating: player can turn opaque polygons into transparent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



Tracing



Scenes: Head (240 frames)



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

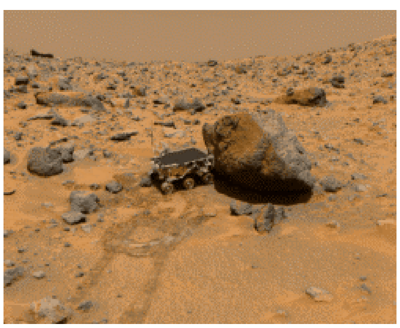
60104	glNormal3fv
60104	glVertex3fv
803	glVertex2fv
722	glLoadMatrixf
720	glMultMatrixf
481	glColor3fv
452	glBegin
452	glEnd
241	glClear
240	gltSwapBuffers
240	glCallList
220	glEndList
220	glNewList
124	glTranslatef
19	glLightfv
9	glEnable
6	glPixelStorei
2	glClearColor
2	glDrawBuffer
2	glLightModelf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

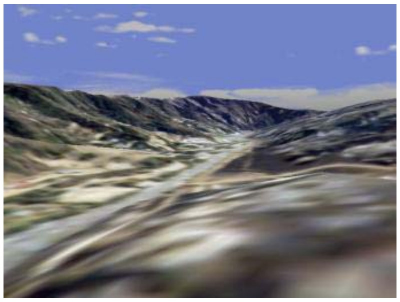
```
109750    glTexCoord2fv
109750    glVertex3fv
54875     glColor4bv
11231     glBindTextureEXT
10975     glBegin
10975     glEnd
1470      glFinish
1468      glLoadIdentity
1468      glMatrixMode
1468      glRotatef
1152      glTexImage2D
768       glTexParameteri
734       gltSwapBuffers
734       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)

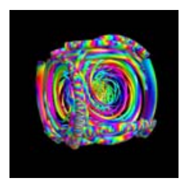
Vertices	3627.0
Triangles (3D)	1801.5
Triangles (2D)	937.4
Fragments	1855471
Image	1024×768

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

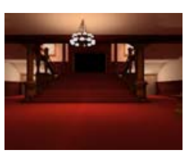
Viewperf OpenGL Benchmark

Alias/Wavefront
Advanced Visualizer
AWadvs-04

DesignReview
DRV-07

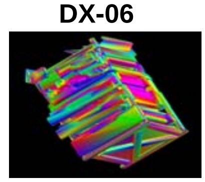


IBM
Data Explorer
DX-06



Lightscape
Light-04

Parametric Technology
ProCDRS-03



Pro/Engineer
medMCAD-01

http://www.specbench.org/gpc/opc.static/opcview.htm

Fragment Formula

Performance: *T a*-pixel triangles per frame

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

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.

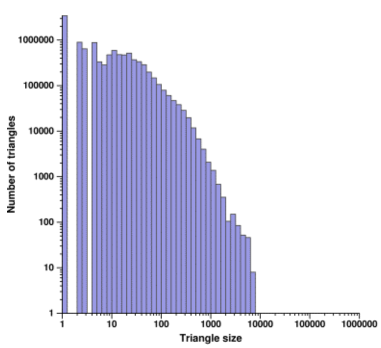
Deering Study

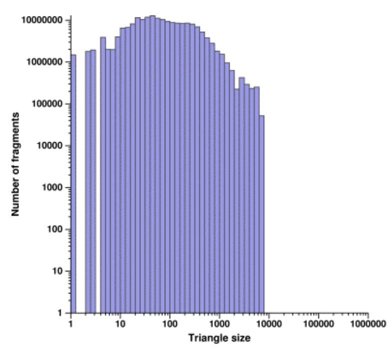
150 optimized triangulations of 3D objects from the Viewpoint, these are created from hand-digitized solid objects, rendered at 700 by 700.

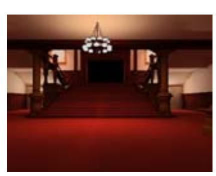
Model	Triangles	F	mean a	median a
85skylark	2116	263933	255	59
R85skylark	2116	304895	305	57
86taurus	2458	278340	230	62
80deloreanM	2770	302871	228	51
83cutlass	3028	245286	156	39
camaro	3640	281127	155	35
…				

Main result: distribution of triangle size in model and screen space is roughly exponential in the direction of small triangles. That is, the median triangle size is smaller than the mean triangle size

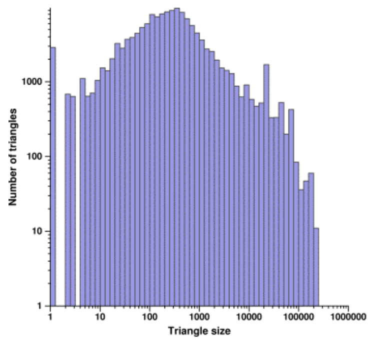
Scene: Light

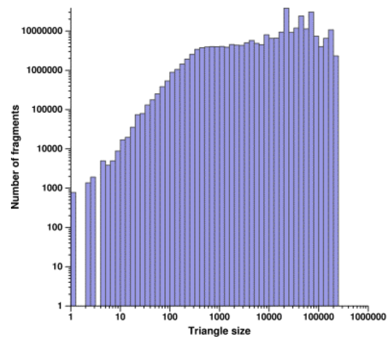






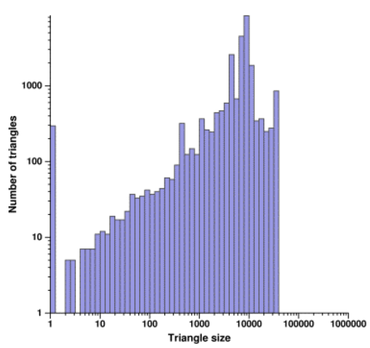
Scene: Flight

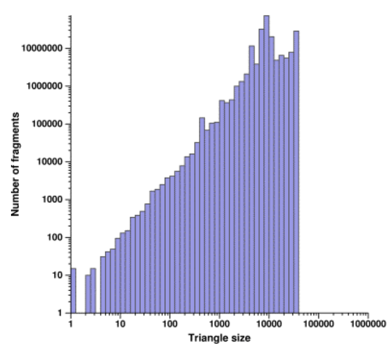




CS448 Lecture 4 Kurt Akeley, Pat Hanrahan, Fall 2001

Scene: QTVR

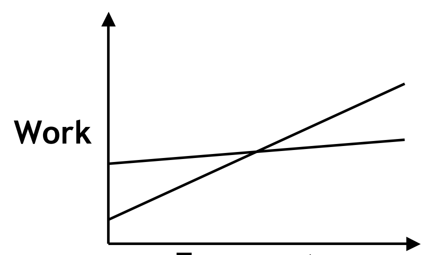




CS448 Lecture 4 Kurt Akeley, Pat Hanrahan, Fall 2001

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



Triangle Size vs. Time (SGI)

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

Peak fill rates

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

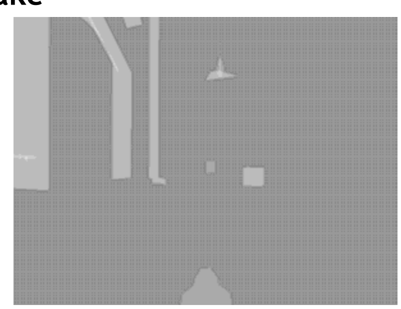
Depth Complexity

Definition:

$$d \equiv \frac{F}{I} \quad \Rightarrow \quad F = dI$$

Quake

Color



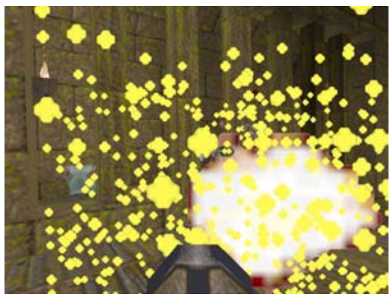
Depth Complexity

Depth Complexity

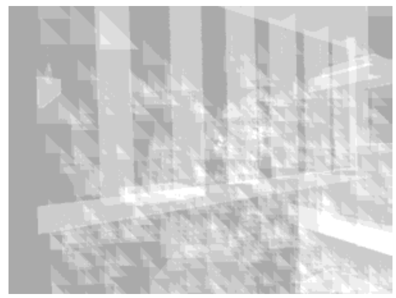
Definition:

$$d \equiv \frac{F}{I} \quad \Rightarrow \quad F = dI$$

Quake



Color



Depth Complexity

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;
}
```

Probability of a write?

1 + 1/2 + 1/3 + 1/4 … 1/n

Knuth: Analysis of Algorithms

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

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

Fill Rates

Need a minimum fill rate (d=1) to be interesting

- For example: VR has high frame rates and hence require high fill rates
- Providing high fill rates has been the major challenge to graphics architects

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

80 Million Triangle Scenes?

Movie quality I = 10 MP (4K by 2.5K)

F = d I = 4 x 10 MP = 40 MF

a = 40 MF / 80 MT = 0.5 F/T (Nyquist limit)

Scaling up to 60 Hz yields 60 I/s * 80 MT/I= 4.8 BT/s

Assumptions:

- Culling limits d to 4
- Level of detail removes really small triangles

Loren Carpenter, Rob Cook, Alvy Ray Smith @ Lucasfilm

Constrained Design Space

$$a\,T = d\,I$$

Parameters

T = Number of triangles

a = Average area of a triangle

F = Number of fragments

I = Image size

d = Depth complexity

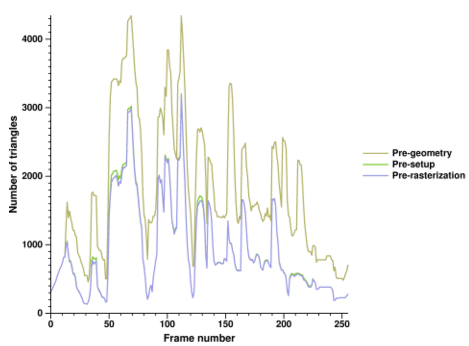
Design Strategies

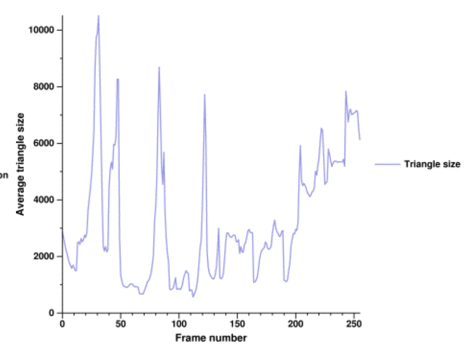
1. Select cost-effective memory technology
   Fixes memory bandwidth and hence fill rate
   Processor capability determines triangle rate
   Triangle area determined

   Lampson and Thacker: GA must fully utilizes bw

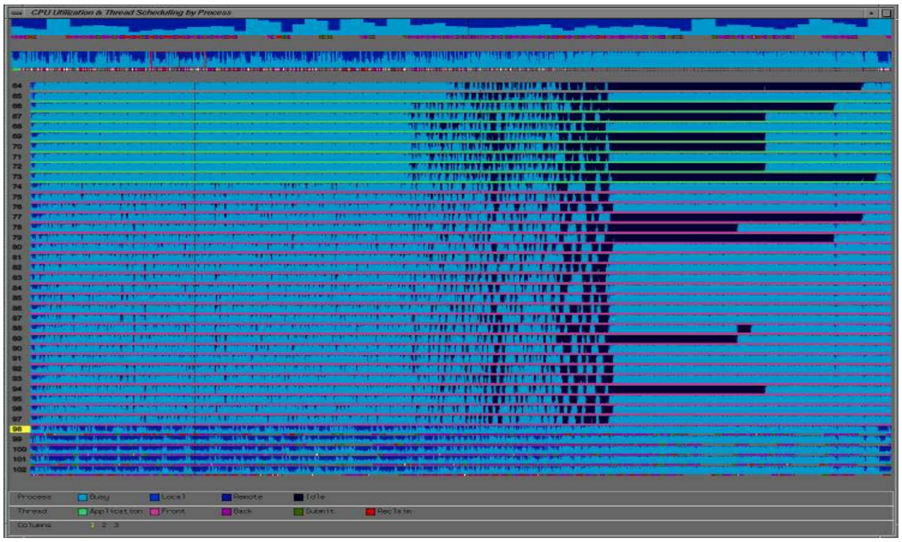
2. Select performance goal
   Target polygon count and average area
   Image size and depth complexity determines fill rate
   Interleave memory to achieve goal

Interframe …





Intraframe …



Performance Limiting Factors

Fill-limited -> Memory limited

Geometry-limited -> Compute- or interface-limited

Application-limited

Most applications are application-limited

- Fundamentally compute-limited

  E.g. marching cubes

- Inefficient use of the graphics system

  E.g. vtk rendering driver

Maxims and Pitfalls

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

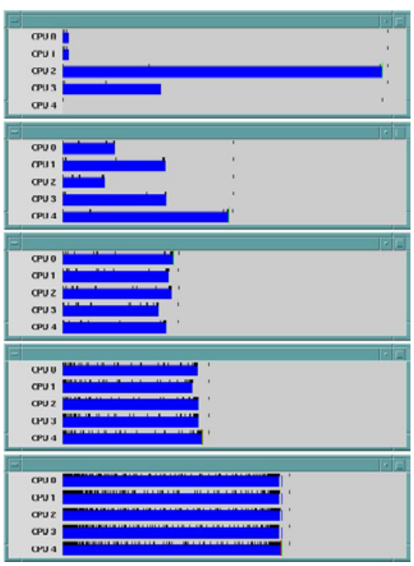
Don't evaluate systems using single-frames; use sequences

Image-Space Work Distribution

Parke - Tiled

Fuchs - Interleaved

Rasterization Cost



512

256

128

64

32

- Large tiles

  Few tasks, greater variation in work

  → bad load balance
- Medium tiles

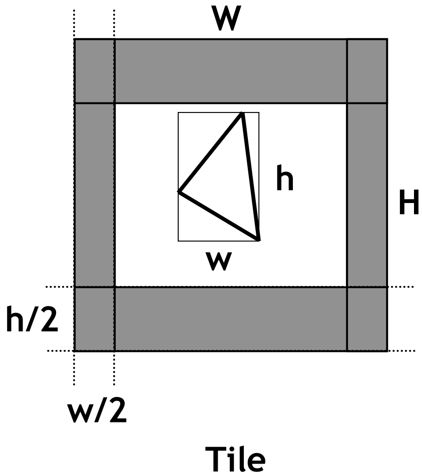
  More tasks, low overlap

  → good load balance
- Small tiles

  High overlap/more redundancy

  → best load balance but redundant work

The Overlap Factor



$$O = \left(\frac{H+h}{H}\right)\left(\frac{W+w}{W}\right)$$

Molnar-Eyles Formula

Derivation! Reference!!

The Overlap Factor