Performance Analysis and Characterization

Real-Time Graphics Architecture

Kurt Akeley
Pat Hanrahan

http://graphics.stanford.edu/courses/cs448-07-spring/

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

Inspired by John Hennessy and David Patterson, Quantitative Computer Architecture
Tracing

Application

OpenGL.dll

Hardware

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

glt: Kekoa Proudfoot
Tricks with Dynamic Libraries

Ability to insert GL filter is very useful
1. Convert to postscript, create animation
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

HijackGL [Mohr/Gleicher]

http://www.cs.wisc.edu/graphics/Gallery/HijackGL/
Visualization of Game Play

C. Neiderauer, M. Houston, M. Agrawala, G. Humphreys, Non-invasive visualization of dynamic architectural environments, 2003

Tracing

OpenGL Trace

Play  Statistics  Simulate

Hardware

CS448 Lecture 3  Kurt Akeley, Pat Hanrahan, Spring 2007
**Scenes: Head (240 frames)**

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

**Scenes: Light (101 frames)**

- Vertices: 1800116
- Triangles (3D): 900058
- Triangles (2D): 106503
- Fragments: 1818726
- Image: 1024×768
Scenes: QTVR (734 frames)

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

Scenes: Town (1338 frames)

| Vertices | 4326.8 |
| Triangles (3D) | 2535.3 |
| Triangles (2D) | 939.0 |
| Fragments | 1353892 |
| Image | 1024×768 |
**Scenes: Flight (123 frames)**

![Image of Flight scene]

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

**Scenes: Quake (330 frames)**

![Image of Quake scene]

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

CS448 Lecture 3

Kurt Akeley, Pat Hanrahan, Spring 2007
SPECviewperf9 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.spec.org/gpc/opc.static/vp9results.html

Balancing the Graphics Pipeline

- Triangle rate
- Fragment rate
- Impedance match/mismatch
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

Key Parameter: Triangle Area

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 rate.

Implications:

- Triangles with average number of pixels greater than \( a \) typically will render at a rate less than \( T \), because the triangles are fill-limited.
- Triangles smaller than \( a \) pixels will render at a rate no faster than \( T \), as such triangles are geometry-limited.

Characteristic of pipelined architectures
Deering Study

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Triangles</th>
<th>F</th>
<th>mean a</th>
<th>median a</th>
</tr>
</thead>
<tbody>
<tr>
<td>85skylark</td>
<td>2116</td>
<td>263933</td>
<td>255</td>
<td>59</td>
</tr>
<tr>
<td>R85skylark</td>
<td>2116</td>
<td>304895</td>
<td>305</td>
<td>57</td>
</tr>
<tr>
<td>86taurus</td>
<td>2458</td>
<td>278340</td>
<td>230</td>
<td>62</td>
</tr>
<tr>
<td>80deloreanM</td>
<td>2770</td>
<td>302871</td>
<td>228</td>
<td>51</td>
</tr>
<tr>
<td>83cutlass</td>
<td>3028</td>
<td>245286</td>
<td>156</td>
<td>39</td>
</tr>
<tr>
<td>camaro</td>
<td>3640</td>
<td>281127</td>
<td>155</td>
<td>35</td>
</tr>
</tbody>
</table>

Primary result: distribution of triangle size in model and screen space is skewed towards small triangles. That is, the median triangle size is smaller than the mean triangle size.

Scene: Light
Scene: Flight

Scene: QTVR
Deering Study

Secondary result: most fragments from large triangles.

Triangle Area 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)

<table>
<thead>
<tr>
<th>Year</th>
<th>Product</th>
<th>F</th>
<th>T</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>1984</td>
<td>Iris 2000</td>
<td>46M</td>
<td>10K</td>
<td>4600</td>
</tr>
<tr>
<td>1988</td>
<td>GTX</td>
<td>80M</td>
<td>135K</td>
<td>592</td>
</tr>
<tr>
<td>1992</td>
<td>RE</td>
<td>380M</td>
<td>2M</td>
<td>190</td>
</tr>
<tr>
<td>1996</td>
<td>IR</td>
<td>1000M</td>
<td>12M</td>
<td>83</td>
</tr>
</tbody>
</table>

**Peak fill rates**

## Triangle Size vs. Time (NVIDIA)

<table>
<thead>
<tr>
<th>Season</th>
<th>Product</th>
<th>F</th>
<th>T</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>2H97</td>
<td>Riva 128</td>
<td>20M</td>
<td>3M</td>
<td>6.67</td>
</tr>
<tr>
<td>1H98</td>
<td>Riva ZX</td>
<td>31M</td>
<td>3M</td>
<td>10.33</td>
</tr>
<tr>
<td>2H98</td>
<td>Riva TNT</td>
<td>50M</td>
<td>6M</td>
<td>8.33</td>
</tr>
<tr>
<td>1H99</td>
<td>TNT2</td>
<td>75M</td>
<td>9M</td>
<td>8.33</td>
</tr>
<tr>
<td>2H99</td>
<td>GeForce</td>
<td>120M</td>
<td>15M</td>
<td>8.00</td>
</tr>
<tr>
<td>1H00</td>
<td>GeForce2</td>
<td>200M</td>
<td>25M</td>
<td>8.00</td>
</tr>
<tr>
<td>2H00</td>
<td>NV16</td>
<td>250M</td>
<td>31M</td>
<td>8.06</td>
</tr>
<tr>
<td>1H01</td>
<td>NV20</td>
<td>500M</td>
<td>30M</td>
<td>16.67</td>
</tr>
<tr>
<td>2H01</td>
<td>NV25</td>
<td>1200M</td>
<td>75M</td>
<td>16.00</td>
</tr>
<tr>
<td>2H02</td>
<td>NV30</td>
<td>2000M</td>
<td>187.5M</td>
<td>10.67</td>
</tr>
<tr>
<td>2H04</td>
<td>NV40</td>
<td>6400M</td>
<td>600M</td>
<td>10.67</td>
</tr>
<tr>
<td>1H05</td>
<td>G70</td>
<td>6880M</td>
<td>860M</td>
<td>8.00</td>
</tr>
<tr>
<td>1H06</td>
<td>G71</td>
<td>10400M</td>
<td>1300M</td>
<td>8.00</td>
</tr>
<tr>
<td>2H06</td>
<td>G80</td>
<td>36800M</td>
<td>575M?</td>
<td>????</td>
</tr>
</tbody>
</table>
**Depth Complexity**

**Definition:**

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

Quake

Color

Depth Complexity

CS448 Lecture 3

Kurt Akeley, Pat Hanrahan, Spring 2007
Depth Complexity

Definition:

\[ d \equiv \frac{F}{I} \Rightarrow F = dI \]

Quake

Color

Depth Complexity

Z-buffer Reads and Writes

\[
\text{If}(\text{fragment.z} < z[\text{fragment.x}][\text{fragment.y}])\
\quad c[\text{fragment.x}][\text{fragment.y}] = \text{blend(fragment)};\
\quad z[\text{fragment.x}][\text{fragment.y}] = \text{fragment.z};
\]

Probability of a write?

\[ 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \ldots \frac{1}{n} \]

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

Creating content is expensive
Movie set analogy (don’t build parts of the environment that can’t be seen)
Well-written apps only draw what is needed
Culling and level-of-detail strategies
Adds significant complexity to the application
Complexity of a Realistic Picture

“80 million triangles are required for realism”
Loren Carpenter, Rob Cook, Alvy Ray Smith @ Lucasfilm

Pixels 3000 x 1667 (5 MP)
Depth complexity 4
Size of polygon ¼ pixel (4 per pixel)
Micropolygons 80,000,000
@60 Hz 4,800,000,000

Constrained Design Space

\[ F = aT = dI \]

Parameters

\( T \) = Number of triangles
\( a \) = Average area of a triangle
\( F \) = Number of fragments
\( I \) = Image size
\( d \) = Depth complexity
Interframe ...

Intraframe ...
Trends, Maxims and Pitfalls

Make sure last year's benchmarks run well

New features may run slowly (at first)

Revolutionary architectures are hard to introduce. They are fast at new stuff and slow at old stuff

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

High fill-rate is the most challenging for now. But will probably eventually level off

Difficult to balance the pipeline; one stage becomes rate-limiting (vertex, geometry, setup, fragment,...)

Trend is towards non-pipelined architectures (unified architectures presents additional challenges)