CGT 581I - Parallel Graphics and Simulation

Performance Analysis

Bedrich Benes, Ph.D.
Professor
Department of Computer Graphics
Purdue University

Performance Analysis

- We want high speed of displaying [fps]
  complex scenes [Δps]
- Higher speed achieved by searching and destroying bottlenecks
- Performance depends on
  - Scene complexity
  - Hardware you use
  - The way you use it (programming)

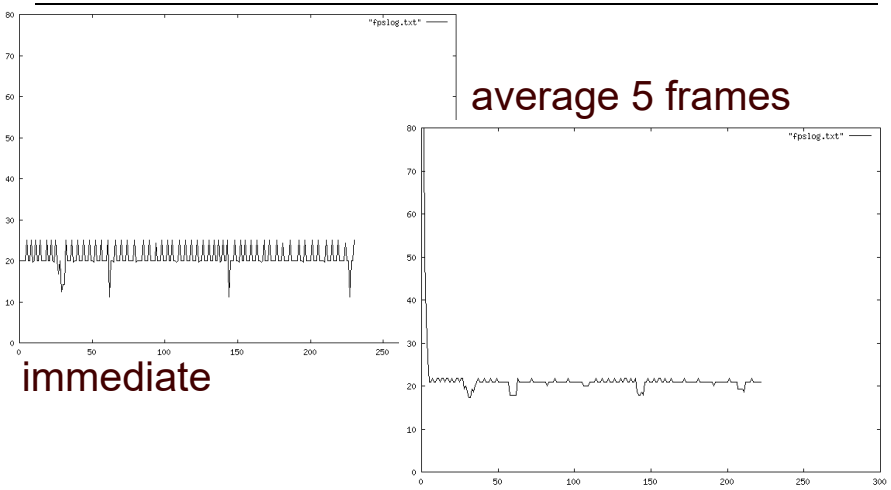
Performance Measurement

- The trirate [Δps] as described by the HW providers may not be accurate (artificial tests, well fed HW, nice scenes, small depth complexity, etc.)
- Measure it yourself
- Two options 1) write an app
  2) use existing

Performance Measurement

- Do it yourself
  - immediate frame rate
    is the plot of the real FPS
  - running average (floating average)
    average of a few (2-10) last frames
    smoother, eliminates small peaks
    nice final look of the graph

Performance Measurement



Performance Measurement

The time spent by **one frame**

```
#include <time.h>
clock_t tin, tout; //time in [ms]
//since the first call
void Display(void){
 tin=clock();
 RenderAll();
 tout=clock();
 timeSpent=tout-tin;
 fps=1000/timeSpent;
}
```


Performance Measurement

The **average** time spent by 5 frames

```
 clock_t t1,t2,t3,t4,t5;
 tin=clock();
 RenderAll();
 tout=clock();
 t2=t1;t3=t2;t4=t3;t5=t4;
 t5=tout-tin;
 sum=t1+t2+t3+t4+t5;
 fps=1000.f/5.f/sum;
}
```


Performance Measurement

An elegant way to do it…

```
#ifdef FPS //FPS logging
#ifdef LOG //writing into a file
   #define LOGFILENAME "fpslog.txt"
   long frame;
   FILE *log;
#endif
#endif
```

Somewhere in the main()…

```
#ifdef LOG
   log=fopen(LOGFILENAME,"wt");
#endif
```

Performance Measurement

Inside the Display() function

```
#ifdef FPS
   clock_t tin,timeSpent;

#ifdef AVGFPS
   static long t1,t2,t3,t4,t5;
#endif
  tin=clock();
#endif
RenderAll();
```


Performance Measurement

```
#ifdef FPS
   timeSpent=clock()-tin;
#endif
#ifdef FPS
#ifdef AVGFPS
  t1=t2;t2=t3;t3=t4;t4=t5;t5=timeSpent;
  timeSpent=(t1+t2+t3+t4+t5)/5.f;
#endif
#ifdef LOG
  frame++;fprintf(log,"%i
   %f\n",frame,1000.f/timeSpent);
#else
  printf("%f ",1000.f/timeSpent);
#endif
#endif
```


Performance Measurement

Using it:

`#define FPS` ~ measures FPS
`#define LOG` ~ saves it into file, otherwise screen
`#define AVGFPS` ~ gets average fps

gnuplot

There are many ways
how the results can be displayed

The good one is using *gnuplot*

GNU, freeware,
LINUX, WIN, MAC OS
User-friendly

http://www.gnuplot.info/
news://comp.graphics.apps.gnuplot

gnuplot

Gnuplot dialog

>set xrange [0:1000] //last frame
>set yrange [0:60] //min max fps
>set data style lines //the way it is shown
>plot “fpslog.txt”

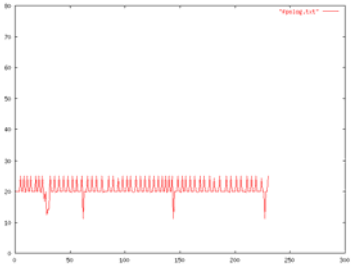
Content of the fpslog.txt:

1 20.000000
2 20.000000
3 20.000000

gnuplot

Writing into a file

> set output “log.gif”//filename
> set terminal gif //the output format
> plot “fpslog.txt” //the file is created…



Performance Measurement

- Measure it yourself

- Two options 1) write an app
  2) use existing

GLtrace

- replaces opengl32.dll
- Put it into the directory of the application
- Create gltrace.ini
  and define what should be traced
- Run the application
- An external text log file is created
- A histogram of OpenGL calls

GLtrace

- GLTrace
- Result can look like this:

```
Call Counts:
   glBitmap                    173
   glClear                     19
   glColor3f                   19
   glEnable                    304
   glEvalMesh2                 2432
…
```


FRAPS

- www.fraps.com
- An external logger
- Displays FPS in the application
- Captures animations, benchmarks
- Free or full version

FRAPS

Results:

1) [fps] is on the screen
2) Log file:

```
2017-01-27 00:25:13 - application
Frames: 56
Time: 2724ms
Avg: 20.558
Min: 20
Max: 21
```


Searching for bottleneck

- A bottleneck is
  the slowest part of your application
- after you detect and eliminate it
  a *new one* will appear
- bottleneck shifts
- How can we find it?

CG Bottleneck

Fill rate [fps]
- # of Δ/s
- texturing

Geometry stage
- composition of Δ, tessellation
- typical for scientific apps. that generate large data

Application
- AI, physics, scene generation, etc.

HW characterization

Fragment formula

$F = a\,T$ *[fragments per second],*

where: T – # of Δ/s

a – average triangle area

Area fill limited

the average area of $\Delta > a$

(T goes down)

Geometry limited

the average area of $\Delta < a$

(T goes down)

HW characterization

Depth complexity (overdraw)

$d = F/I,$

where:

d is the depth complexity [-]

F is the number of fragments

I *is the* number of image pixels

d: because all F cannot all fit on I, some of them are *overdrawn*

$d = 4$ is usually the worst case…

HW characterization

Depth complexity (overdraw)

How can we measure it?

```
glStencilOp(GL_KEEP, GL_INCR, GL_INCR)
```

Z-buffer test:

```
if (frag.z<z[i][j]) {          //read
    color[i][j]=frag.color;    //modify
    z[i][j]=frag.z;            //write
}
```

Read – Modify – Write

is slower than *read only*

HW characterization

- *Level of detail* should remove small Δs
  they slow down significantly
  because of the geometry limit
  (it makes no sense to calculate triangle
  that is really small…)

- *Culling* should eliminate $d \leq 4$
  multiple Z-buffer hits slow down…

Searching for bottleneck

- How can we find it?

- Use a performance measuring [fps]
- Decompose the scene
- Decompose the program

Searching for bottleneck

Step 1) Avoid ALL graphics

Pass NULL to all CG calls.

This gives pure application overhead.

The best case for the pure application
(infinitely fast graphics)

Searching for bottleneck

Step 2) Forced Serialization

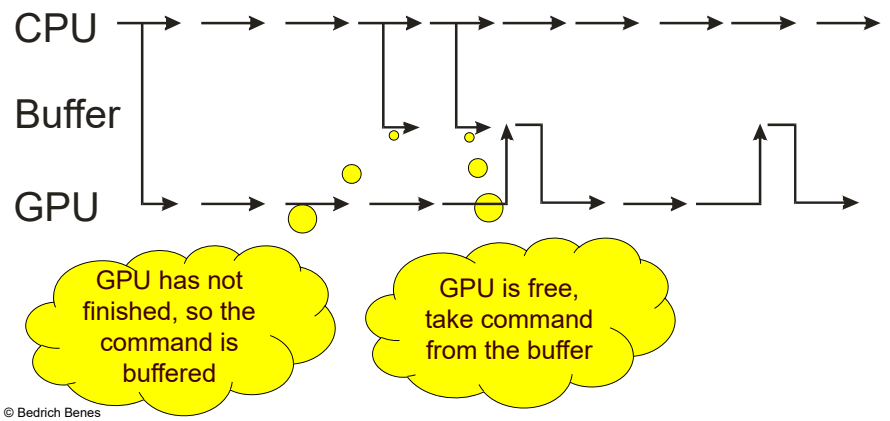
Make it serial
Pass `glFinish()` frequently
The worst case on your HW

Normally:
Driver buffers CPU commands for GPU,

This approach stops it,
so they work as serial units

Searching for bottleneck

Step 2) Normally



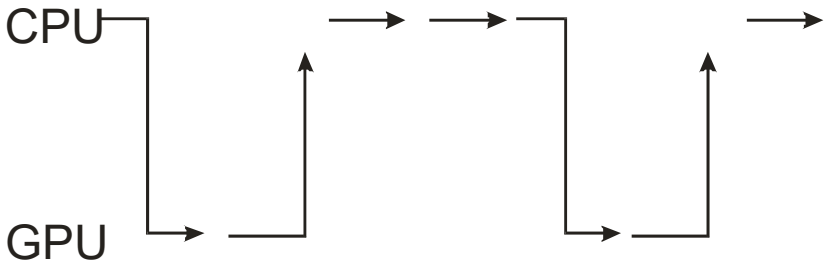
Searching for bottleneck

`glFinish()` in fact means,

Wait till the command buffer is empty and
The GPU has nothing to do

Searching for bottleneck

Step 2) Forced serialization



Searching for bottleneck

Searching for bottleneck is iterative:

1. Take the application and
   - While not sufficiently small
     - Divide the application into parts
     - Find the slowest one
   - Improve the bottleneck
2. If it is not enough go to 1)

Readings

- www.gnuplot.info
- www.fraps.com