



Análise de Desempenho

*Pipeline Gráfico e
Profiling CG*



O Pipeline Gráfico

- Análise Conceptual do Pipeline Gráfico:
 - Estudo do processo de produção de imagem...

... do ponto de vista das funcionalidades do pipeline gráfico.



Arquitectura

- O *Pipeline Gráfico* pode ser conceptualmente dividido em três fases:
 - Aplicação
 - Geometria
 - Rasterizer
- Cada fase realiza um determinado conjunto de funções.
- A relação entre as funções e os estágios do *pipeline* são dependentes da implementação deste.



Fase 1: Aplicação

- Parte essencialmente realizada em software.
- No final desta fase, os dados são passados para a fase da geometria.

Tarefas Gráficas:

- Detecção de Visibilidade
- Detecção de Oclusões
- Detecção de Colisões
- Manipulação de Vértices
- Animação
- etc...

Tarefas Não Gráficas:

- Processamento de teclado e rato
- Inteligência Artificial
- Gestão de recursos de rede
- etc...



Fase 2: Geometria

- Funções:
 - conversão espaço mundo \rightarrow espaço câmara
 - iluminação (por vértice)
 - projecção: espaço câmara \rightarrow espaço clip
 - clipping
 - conversão coord. normalizadas \rightarrow espaço ecrã



Fase 3: Rasterizer

- Fase responsável pela criação final da imagem.
- Necessita de preencher os polígonos atribuindo uma cor a cada *pixel*/dependendo de alguns factores como:
 - Visibilidade: Z-buffer
 - Texturas
 - Blending
 - Nevoeiro
 - ...



Análise de Desempenho

- Optimizar tudo!
- Má estratégia :(
- Garante-se um ganho em desempenho mas...
- sem aproveitar as capacidades do sistema.



Análise de Desempenho

- Objectivo: Equilibrar a carga das diversas fases

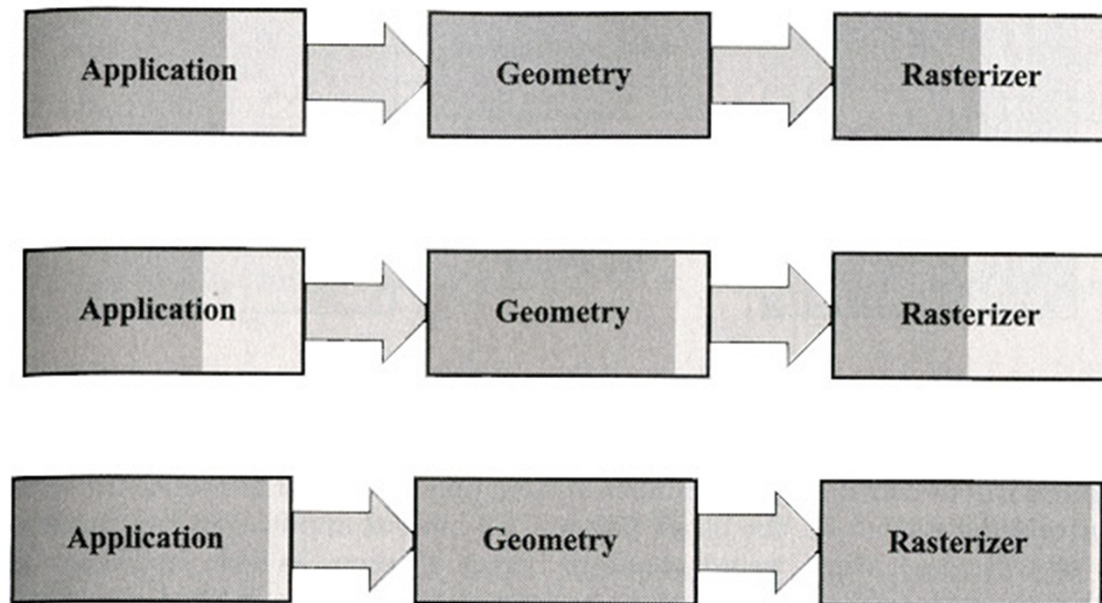


figura do livro "Real Time Rendering"



Análise de Desempenho

- Estratégia - Iterativamente fazer:
 - Passo 1: Identificar estrangulamentos
 - Passo 2: Eliminar o estrangulamento



Análise de Desempenho

- Alguns pontos a ter em consideração:
 - Memória gráfica é suficiente para geometria e texturas?
 - Por frame é desenhado só o que é realmente visível?
 - A aplicação está a tirar partido do CPU e do GPU?



Análise de Desempenho

- Fase de raster:

Dimensão da janela

Utilização de texturas

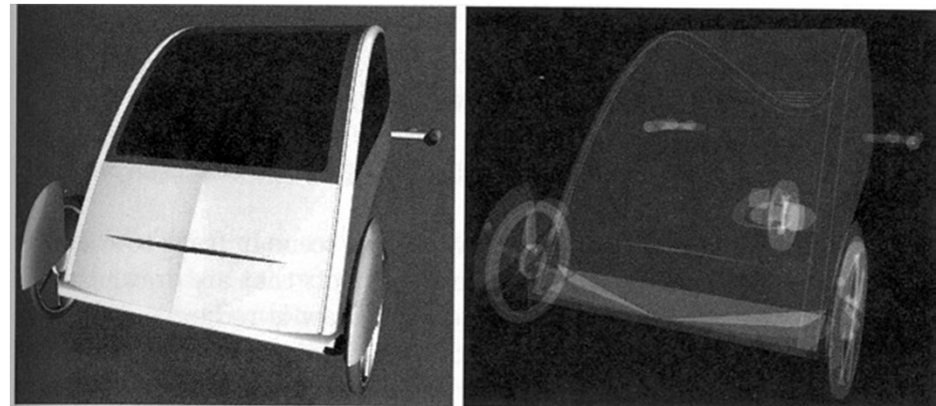


figura do livro "Real Time Rendering"

Mapas de complexidade:

- Indicam-nos o número de vezes que cada *pixel* é testado em relação ao Z-buffer.



Análise de Desempenho

- Considerando que os polígonos são submetidos de forma aleatória, ...
- ... se um *pixel* tem uma complexidade n ...
- ... então o número de vezes que o *pixel* é efectivamente desenhado é em média :
 - $H(n) = 1 + 1/2 + 1/3 + \dots + 1/n$
- Exemplo:
 - $n = 4 \rightarrow H(n) = 2.08$
 - $11 \rightarrow H(n) = 3.02$
 - $12367 \rightarrow H(n) = 10$



Análise de Desempenho

- Gerar mapas de complexidade na prática:
 - Desactivar o Z-buffer, iluminação e texturas
 - Atribuir a cor (1,1,1) a cada vértice // inteiros
 - `glBlendFunc(GL_ONE, GL_ONE)`
- Testar o número de vezes que o *pixel* é desenhado
 - Equivalente ao processo anterior mas com o Z-buffer activado



Análise de Desempenho

- Os mapas de complexidade não representam a complexidade da cena.
 - Não fornecem informação sobre a carga imposta na fase da geometria
 - Fornecem no entanto uma forma de analisar a carga na fase de raster.
 - Note-se, no entanto, que a carga do rasterizer é também dependente da utilização de texturas, blending, etc...



Optimização

- *Display lists*

- permitem armazenar uma sequência de comandos de forma otimizada para o hardware gráfico;
- eliminam o overhead da invocação de funções;
- permitem o armazenamento em memória de mais rápido acesso por parte do hardware gráfico.



Optimização

- Utilizar *strips* e *fans*
 - menos vértices a enviar para a fase da geometria
 - evita processamento repetido de vértices
 - menos memória consumida na placa gráfica



Optimização

- *Vertex Buffers*

- reduz invocações de funções à API;
- permite evitar o reprocessamento de vértices;
- elimina número excessivo de invocações para submeter vértices
- permite o armazenamento na memória do hardware gráfico.



Optimização

- Precisão

- Se possível utilizar dados mais económicos para minimizar transferência e armazenamento de dados

- Texturas

- Tentar garantir que não se ultrapassa os limites de memória da placa gráfica (juntamente com a geometria).



Optimização

- Tópicos avançados:
 - Detecção de Visibilidade
 - Detecção de Oclusões
 - Níveis de Detalhe
 - Impostores



Profiling

- Objectivos:
 - Obter um quadro claro da distribuição temporal das tarefas desempenhadas.
- Questões:
 - Quais as funções onde se gasta mais tempo?
 - Quais as funções que são invocadas mais vezes?
 - Quais os picos de congestionamento?



Profiling

- Visual Studio - Function

Func Time	%	Func+Child Time	%	Hit Count	Function
19828.853	84.1	19828.853	84.1	179716	_glutSolidSphere@16 (glut32.dll)
1521.176	6.5	1521.176	6.5	1	_glutCreateWindowWithExit@8 (glut32.dll)
891.893	3.8	891.893	3.8	44929	_glutSolidCone@24 (glut32.dll)
703.682	3.0	21424.428	90.9	44929	drawSnowMan(void) (glutsnowman.obj)
214.521	0.9	21955.474	93.2	1248	renderScene(void) (glutsnowman.obj)
134.222	0.6	134.222	0.6	12481	GetExactTime(void) (custom_time.obj)
79.197	0.3	79.197	0.3	1248	_glutSwapBuffers@0 (glut32.dll)
75.232	0.3	22040.395	93.5	1	_glutMainLoop@0 (glut32.dll)
47.964	0.2	47.964	0.2	28	glutSetWindowTitle@4 (glut32.dll)
13.768	0.1	13.768	0.1	1248	MarkTimeThisTick(void) (custom_time.obj)

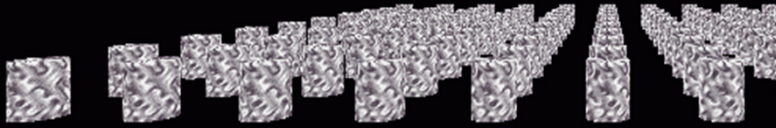


Profiling

Profiler para CG

- Vantagens:
 - Conceito de bloco (pode coincidir com uma frame)
 - display em tempo real na aplicação
- Quatro funções são suficientes:
 - Inicialização
 - Inicio de Bloco
 - Fim de Bloco
 - Relatório

Name	#c	ms	wt
Render	1	33.807	0.000
Draw	1	33.257	0.000
mediate	100	33.203	0.006
Stats Dump	1	1.166	0.000
dummy	0	0.000	0.000
Swap	0	0.097	0.000





Profiling

Utilização:

```
void renderScene(void) {  
  
    {  
        PROFILE("Render");  
        ... //preparação para rendering  
  
        {  
            PROFILE("Draw");  
            drawObjects();  
        }  
  
        {  
            PROFILE("Stats Dump");  
            ... // apresentação das estatísticas  
        }  
  
        { PROFILE("dummy");  
          ... // tarefas não gráficas  
        }  
  
        {  
            PROFILE("Swap");  
            glutSwapBuffers();  
        }  
    }  
}
```



Profiling

- Análise:

Identificador do bloco		Tempo cumulativo gasto por frame			Tempo gasto no profiling	
Nome		Número de invocações por frame	#c	ms	wt	
Render		1	40.569	0.000		
Draw		1	33.383	0.000		
immediate		100	33.330	0.008		
Stats Dump		1	1.175	0.000		
dummy		1	5.823	0.000		
Swap		1	0.102	0.000		



Profiling

- Análise:

modo imediato

Name	#c	ms	wt

Render	1	40.569	0.000
Draw	1	33.383	0.000
imediato	100	33.330	0.008
Stats Dump	1	1.175	0.000
dummy	1	5.823	0.000
Swap	1	0.102	0.000

c/ vertex buffers

Name	#c	ms	wt

Render	1	15.204	0.000
Draw	1	0.060	0.000
VB	100	0.033	0.004
Stats Dump	1	1.158	0.000
dummy	1	5.704	0.000
Swap	1	8.227	0.000



Profiling

- Análise:
 - Impacto de tarefas não gráficas no modo imediato

Name	#c	ms	wt	Name	#c	ms	wt
-----				-----			
Render	1	40.569	0.000	Render	1	34.605	0.000
Draw	1	33.383	0.000	Draw	1	33.218	0.000
immediate	100	33.330	0.008	immediate	100	33.165	0.007
Stats Dump	1	1.175	0.000	Stats Dump	1	1.197	0.000
dummy	1	5.823	0.000	dummy	1	0.000	0.000
Swap	1	0.102	0.000	Swap	1	0.112	0.000



Profiling

- Análise:
 - Impacto de tarefas não gráficas com vertex buffers

Name	#c	ms	wt	Name	#c	ms	wt
-----				-----			
Render	1	9.526	0.000	Render	1	9.491	0.000
Draw	1	0.056	0.000	Draw	1	0.058	0.000
VB	100	0.033	0.004	VB	100	0.034	0.004
Stats Dump	1	1.331	0.000	Stats Dump	1	1.341	0.000
dummy	1	5.797	0.000	dummy	1	0.000	0.000
Swap	1	2.296	0.000	Swap	1	8.050	0.000



Profiling

- Conclusão:
 - A utilização de uma ferramenta de profiling permite determinar a influência de cada tarefa...
 - ... e identificar estrangulamentos potencialmente difíceis de detectar



Referências

- *Game Programming Gems, Vol I*
- *Real Time Rendering, Moller and Haines*