Workshop

Optimizing CPU Performance: Image Creation in Computer Graphics

Universidade do Minho
22nd January 2004

Carlos Silva & Manuel Carvalho

Conteúdos

- PIRT (Parallel Inteligent Ray Tracer)
- Técnicas de optimização (exemplos)
- Aplicação prática das técnicas no PIRT
- Visualização e comparação de resultados
- Conclusões
- Debate

Programa analisado: PIRT 1.0

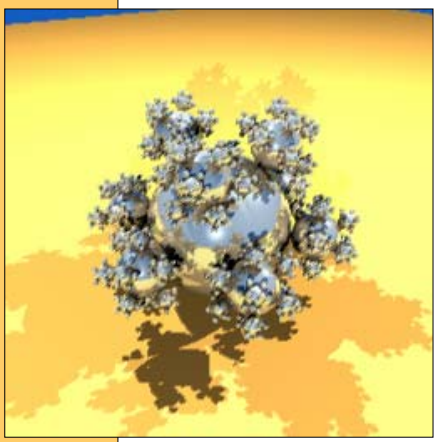
(Parallel Inteligent Ray Tracer)

- O programa lê de um ficheiro de texto onde está descrito um cenário com objectos e fontes de luz
- Traça raios de luz sobre esses objectos
- Verifica onde o raio de luz intercepta o objecto
- Verifica possíveis reflexões do raio por parte do objecto interceptado
- Constrói um ficheiro de imagem do tipo TARGA

Exemplo: Pirt balls3.nff

```
b 0.078 0.361 0.753
v
from 2.1 1.3 1.7
at 0 0 0
up 0 0 1
angle 45
hither 0.01
resolution 256 256
l 4 3 2
l 1 -4 4
l -3 1 5
f 1 0.75 0.33 0.8 0 100000 0 1
p 4
12 12 -0.5
-12 12 -0.5
-12 -12 -0.5
12 -12 -0.5
f 1 0.9 0.7 0.5 0.5 45.2776 0 1
s 0 0 0 0.5
s 0.272166 0.272166 0.544331 0.166667
s 0.420314 0.420314 0.618405 0.0555556
. . .
s 0.490576 -0.542955 -0.222222 0.0185185
s 0.419026 -0.523783 -0.222222 0.0185185
```

PIRT



Programa analisado: PIRT 1.0

(Parallel Inteligent Ray Tracer)

- Código em C standard (não usa objectos)
- Desenvolvido em Visual C++ (6.0)
- Orientado para PCs mono-processador
- PIRT 2.0
  - Corre em ambiente paralelo
  - Tese de doutoramento do *chairman (prof. Luis Paulo Santos)*

Técnicas de optimização

1. Editor/Compilador
2. Acelerar cálculos matemáticos
3. Substituição de cálculos
4. Acelerar ciclos
5. Remover cálculos de ciclos
6. Expressões de cálculos repetidos
7. Inclusão de assembly
8. Inline

1. Editor/Compilador

- Não há alteração de código
- Conhecimento das opções de compilação
  - Velocidade *vs* tamanho do código
- Conhecimento das arquitecturas dos computadores
  - Diferentes processadores
  - Diferentes Sistemas Operativos

2. Acelerar cálculos matemáticos

- Redução de operações:
  - `a*b + a*c = a*(b+c);` → menos 1 multiplicação
  - `b/a + c/a = (1/a)*(b+c);` → as multiplicações são mais rápidas.
  - `(b+c)/a;` → Uma variação ainda mais rápida
- metodologia *lazy evaluation*
  - `((a || b) && c) = (c && (a || b));`

3. Substituição de cálculos

- Exemplos:
  - Raízes quadradas inteiras de inteiros

Ex: raiz quadrada de 9:
9 – 1 = 8
8 – 3 = 5
5 – 5 = 0

  - Multiplicação e divisões por potências de 2

x << y é o mesmo que $x * 2^y$
x >> y é o mesmo que $x / 2^y$
(shift de y posições de x)

4. Acelerar ciclos

Exemplo:

```
for( int i = 0; i < 3; i++ ) array[i] = i;
```

é logicamente igual a

```
array[0] = 0; array[1] = 1, array[2] = 2;
```


5. Retirar cálculos de ciclos

Exemplo:

```
for( int i = 0; i < numPixels; i++ ){
   float brighten_value = view_direction *
            light_brightness*( 1 / view_distance );
   back_surface_bits[i] *= brighten_value;
}
```

***brighten_value* é sempre igual**

```
float brighten_value = view_direction *
            light_brightness*( 1 / view_distance );
for( int i = 0; i < numPixels; i++ ){
  back_surface_bits[i] *= brighten_value;
}
```


6. Reduzir repetição de expressões de cálculo

Exemplo:

```
if ((dataStructPointer->ExpensiveFunctionCall()) < 10)
{
   /* código */
}
else if ((dataStructPointer->ExpensiveFunctionCal()) > 30)
{
   /* código */
}
```

Este pode ser reescrito da seguinte forma:

```
int temp = dataStructPointer->ExpensiveFunctionCall() ;
if(temp < 10)
{
     /* código */
}
else if(temp > 30)
{
     /* código */
}
```

7. Inclusão de assembly

```
#include "stdafx.h"

#define myfabs(x) fabs(x)

double myfabs(double x)
{
  _asm
  {
      FLD [x];     /* Carrega x para ST0*/
      FABS;        /* Calcula o valor abs de ST0 e
  armazena resultado em ST0*/
      FSTP [x];    /* Armazena ST0 em x e faz pop da
  stack*/
  }
  return x;
}
```


8. Inline

```
extern double soma (double a, double b)
{
  return (a + b);
}

extern void xpto (double *v)
{
  double n;
  . . .

  n = soma (v[0], v[1]);
  . . .
}
```

n = v[0] + v[1]);

Configurações do compilador

- Menu Project → Settings → C/C++ → Category
- General
  - **Debug Info = none**
  - Preprocessor definitions = WIN32, _DEBUG, _CONSOLE, _MBCS
- C++ Language
  - Representation method = Best-Case Always
  - Enable exception handling = OK
- Optimizations
  - **Nenhuma activa**
- Todas as outras opções foram mantidas.

Profile (funções que gastam mais tempo)

Func Time	%	Func+Child Time	%	Hit Count	Function
19.457,150	**29,8**	20.937,040	32,1	**22.366.995**	AlignedBoxInt()
16.188,580	**24,8**	65.191,340	100	**1**	_main ()
8.370,702	**12,8**	34.577,580	53	**7.016.899**	PartIsItVisible()
4.274,973	***6,6***	*4.274,973*	*6,6*	***11.859.558***	*GetObject()*
4.044,491	***6,2***	*8.680,304*	*13,3*	***11.405.261***	*SphereInt()*
3.715,801	**5,7**	3.715,801	5,7	**22.844.193**	Norma()
2.721,261	**4,2**	4.668,234	7,2	**11.474.987**	Normalize()
2.496,392	***3,8***	*13.094,940*	*20,1*	***2.181.601***	*PartWhichObject()*
2.424,928	***3,7***	*11.943,870*	*18,3*	***11.466.112***	*Intersect()*
TOTAL:	**97,6**				

Optimização 1: AlignedBoxInt()
(Pointer Dereferencing)

```
extern int AlignedBoxInt (Point *B1, Point *B2, Ray *r, double *tn)
{
   double tnx, tny, tnz, tfx, tfy, tfz;
   double tnear, tfar;
   myBool inside=0;
   /* if the point is inside the box then it must be intersected */

   if ((r->S.X-B1->X >= -FZERO) && (r->S.X-B2->X<=FZERO) &&
        (r->S.Y-B1->Y >= -FZERO) && (r->S.Y-B2->Y<=FZERO) &&
        (r->S.Z-B1->Z >= -FZERO) && (r->S.Z-B2->Z<=FZERO))
        inside = 1;
   if (fabs(r->Dir.X) < FZERO)  /* ray in the x direction */
   {
   }
   el
   {
        tfx = (B1->X - r->S.X)/r->Dir.X;
   }
. . .
```

Cálculos com estruturas de apontadores:

RS->X ⇔ REG ← RS+X (uma soma)

r->S.X ⇔ REG ← r+S+X (duas somas)

Optimização 1: AlignedBoxInt()
(Pointer Dereferencing)

```
extern int AlignedBoxInt (Point *B1, Point *B2, Ray *r, double *tn)
{
   double tnx, tny, tnz, tfx, tfy, tfz;
   double tnear, tfar;

   double RSX = r->S.X  , RSY = r->S.Y   ,RSZ = r->S.Z;
   double B1X = B1->X   , B1Y = B1->Y    ,B1Z = B1->Z;
   double B2X = B2->X   , B2Y = B2->Y    ,B2Z = B2->Z;
   double RDX = r->Dir.X, RDY = r->Dir.Y ,RDZ = r->Dir.Z;
   myBool inside=0;
   /* if the point is inside the box then it must be intersected */

   if ((RSX-B1X >= -F
        (RSY-B1Y >= -
        (RSZ-B1Z >= -
        inside = 1;
. . .
   else if (RDX < 0.0
   {
        tnx = (B2X -
        tfx = (B1X -
   }
. . .
```

Optimização:

RS->X ⇔ REG ← RS+X
r->S.X ⇔ REG ← r+S+X

double RSX = r->S.X;
double B1X = B1->X;

Optimização 2: PartIsItVisible()
(Pointer Dereferencing)

- Idêntica à Optimização 1
- …

Optimização 3: Normalize()
(Inline)

```
extern double Norma (Vector *v)
{
  return(sqrt(v->X*v->X + v->Y*v->Y + v->Z*v->Z));
}
/* This function normalizes a vector */
extern void Normalize (Vector *v)
{
  double n;
  n = Norma (v);
  if (fabs(n) > FZERO) {
       v->X /= n;
       v->Y /= n;
       v->Z /= n;
  }
}
```

Optimização 3: Normalize()
(Inline)

```
extern void Normalize (Vector *v)
{
  double n;
/*    n = Norma (v);   */
  n = sqrt(v->X*v->X + v->Y*v->Y + v->Z*v->Z);
  if (fabs(n) > FZERO) {
      v->X /= n;
      v->Y /= n;
      v->Z /= n;
  }
}
```


Optimização 4: Norma()
(Inline)

```
Searching for 'Norma'...
C:\pirt\AuxGeom.cpp:      cosseno /= Norma(V1);
C:\pirt\AuxGeom.cpp:      cosseno /= Norma(V2);
C:\pirt\AuxVect.cpp:      n = Norma (v);
C:\pirt\GenRay.cpp:       t = tan (Angle) / Norma (&down);
C:\pirt\GenRay.cpp:       t = Norma(&down)/Norma (&right);
C:\pirt\Intersec.cpp:             *t *= Norma (&(r->Dir));  /* t must be
   the real distance */
C:\pirt\Intersec.cpp:     distaux = (*t)/Norma (&(r->Dir));
C:\pirt\Intersec.cpp:             *tn = tnear * Norma(&(r->Dir));
C:\pirt\Intersec.cpp:             *tn = tfar * Norma(&(r->Dir));
C:\pirt\Intersec.cpp:                     norma = Norma (&(r->Dir));
C:\pirt\Intersec.cpp:                     norma = Norma (&(L->Dir));
C:\pirt\Render.cpp:       normaV = Norma (&(V->Dir));
C:\pirt\Render.cpp:       normaN = Norma(&N);
C:\pirt\Render.cpp:               distlight = Norma(&(ToLight.Dir));
C:\pirt\Render.cpp:               normaL = Norma(&(ToLight.Dir));
C:\pirt\Render.cpp:                       RVcosseno /= Norma (&R);
C:\pirt\Render.cpp:                       RVcosseno /= Norma (&(V->Dir));
17 occurrence(s) have been found.
```


Optimização 5: PutPixel(), GetPixel()
(Reduzir repetição de expressões de cálculo)

```
extern int PutPixel (Pixel *p, int hndl, int x, int y)
{
  int auxy, auxx;
  auxx = x-img[hndl].x0;
  auxy = y-img[hndl].y0;
  if ((x>=img[hndl].width) || (y>=img[hndl].height))
      return (-1);

  img[hndl].data[auxy*img[hndl].width+auxx].R = p->R;
  img[hndl].data[auxy*img[hndl].width+auxx].G = p->G;
  img[hndl].data[auxy*img[hndl].width+auxx].B = p->B;
  return (1);
}
```


Optimização 5: PutPixel(), GetPixel()
(Reduzir repetição de expressões de cálculo)

```
extern int PutPixel (Pixel *p, int hndl, int x, int y)
{
  int auxy, auxx, pos;
  auxx = x-img[hndl].x0;
  auxy = y-img[hndl].y0;
  if ((x>=img[hndl].width) || (y>=img[hndl].height))
      return (-1);

  pos = auxy*img[hndl].width+auxx;
  img[hndl].data[pos].R = p->R;
  img[hndl].data[pos].G = p->G;
  img[hndl].data[pos].B = p->B;
  return (1);
}
```

+ 1 variável
+ 1 atribuição
- 2 expressões matemáticas

Optimização ASM: fabs() e sqrt()
(Inclusão de assembly)

```
#include "stdafx.h"
#define myfabs(x)  fabs(x)
#define mysqrt(x)  sqrt(x)

double myfabs(double x)
{
 asm
 {
  FLD [x];        /* Carrega x para ST0*/
  FABS;           /* Calcula o valor abs de ST0 e armazena resul em ST0*/
  FSTP [x];       /* Armazena ST0 em x e faz pop da stack*/
 }
 return x;
}
double mysqrt(double x)
{
_asm
 {
  FLD [x];        /* Carrega x para ST0*/
  FSQRT;          /* Calcula o sqrt de ST0 e armazena resul em ST0*/
  FSTP [x];       /* Armazena ST0 em x e faz pop da stack*/
 }
 return x;
}
```


Resultados – com Código C

Tempo sem optimizações: **31004** milissegundos

	TEMPOS:			
AlignedBoxInt() - Optim_4	31245	-241	**-0,78%**	Optim_4 **PIOROU**
PartIsItVisible() - Optim_5	30644	360	**1,16%**	Optim_5 MELHOROU
Normalize() - Optim_1	30854	150	**0,48%**	Optim_1 MELHOROU
PartIsItVisible() Normalize() - Optim_1, Optim_5	30594	410	**1,32%**	Optim_1, Optim_5 MELHOROU
Norma() sqrt() -Optim_2	30964	40	**0,13%**	Optim_2 MELHOROU
PutPixel() GetPixel() - Optim_3	30865	139	**0,45%**	Optim_3 MELHOROU
Optim_1, 2, 3, 4 and 5	30435	569	**1,84%**	Todas sem ASM MELHOROU
without Optim_4	29443	1561	**5,03%**	Optim_1, 2, 3, 5 MELHOROU

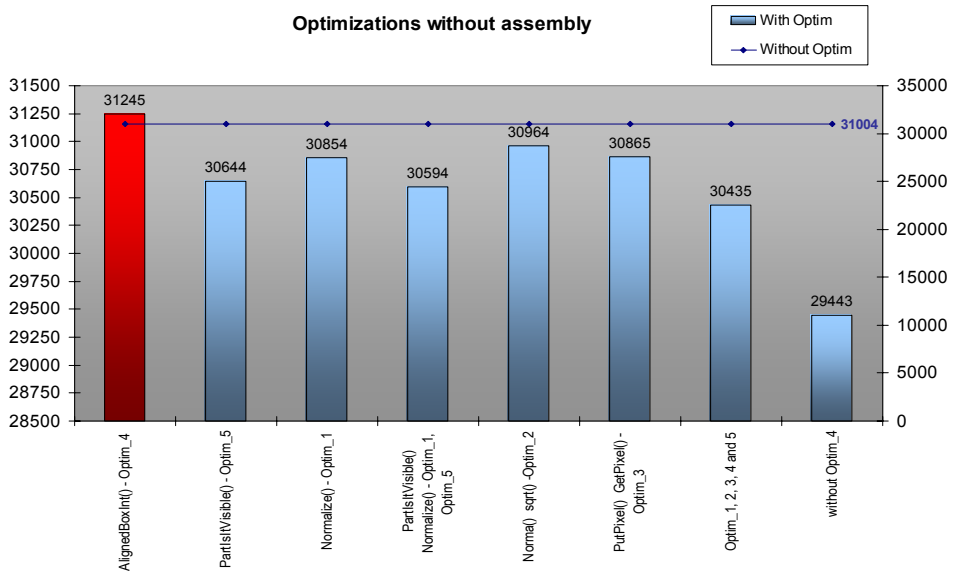
Resultados – com Assembly

fabs() sqrt() - Optim_ASM	30594	410	**1,32%**	Optim_ASM MELHOROU

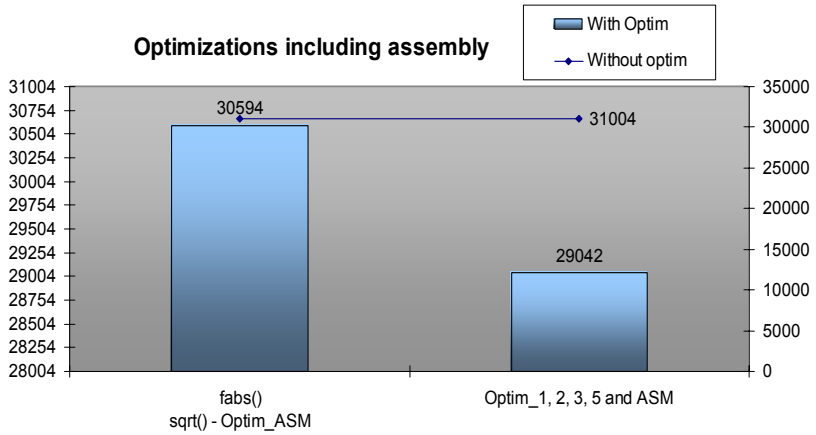
Resultados – com C + Assembly

Optim_1, 2, 3, 5 and ASM	29042	1962	**6,33%**	Optim_1, 2, 3, 5 e ASM MELHOROU

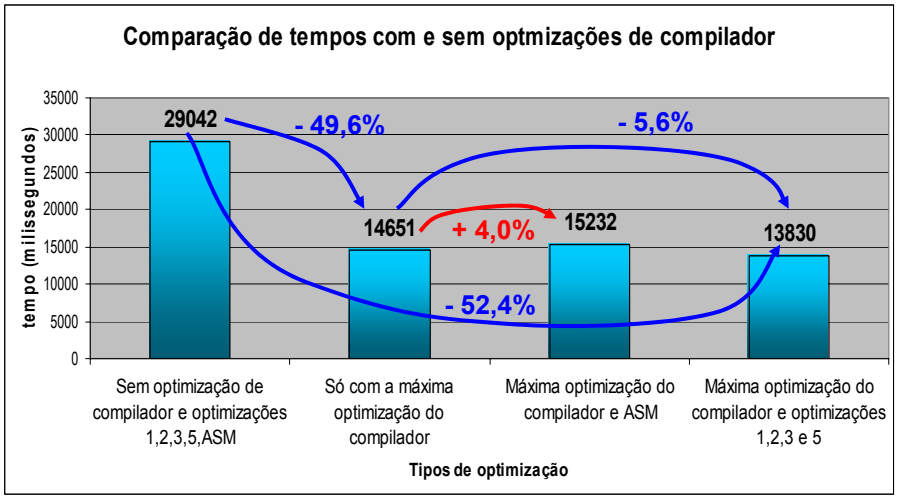
Resultados - comparação



Resultados - comparação



Com optimizações do compilador



Tempo final inferior a 50% tempo inicial !!!

Conclusões

- Gasta-se muito tempo a fazer optimizações
- As optimizações valem a pena em situações críticas
- A forma como se escreve código influencia o desempenho
- Os compiladores fazem muitas optimizações
- No caso estudado, as optimizações do compilador são melhores sem a inclusão de assembly no código

Errata do artigo

- Pág 115 (nos endereços de e-mail)
  ???@*mestrado.*???
- Pág 116
  Onde se lê
  "Avoiding calculations in loops"
  Devia ler-se
  "Reducing calculations"
- Pág 117 (no gráfico)
  Na legenda da 5ª barra riscar: *"and Optim_3"*