

```
ics
& (depth < MAXDEPTH)
{
    inside / inside;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
}
{
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, classically
    if;
    radiance = SampleLight( &rand, I, &t, &light;
    e.x + radiance.y + radiance.z) > 0) && (rand <
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following
    survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```

/INFOMOV/

Optimization & Vectorization

J. Bikker - Sep-Nov 2015 - Lecture 4: "Low Level (2)"

Welcome!



Today's Agenda:

- Optimizing GlassBall

```
...ics
& (depth < MAXDEPTH)
{
    ...
    t = inside / (1 + refl);
    nt = nt / nc; nct = nct / nc;
    cos2t = 1.0f - nnt * nct;
    ...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * cosTheta);
    Tr) R = (D * nnt - N * (1 - D));
    ...
    E * diffuse;
    ...
    refl + refr)) && (depth < MAXDEPTH)
    ...
    D, N );
    refl * E * diffuse;
    ...
    MAXDEPTH)
    ...
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &t, &align, &pdf );
    e.x + radiance.y + radiance.z) > 0) && (cosTheta > 0)
    ...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;
    ...
    random walk - done properly, closely following
    survive)
    ...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    ...
}
```



Rules of Engagement

Common Opportunities in Low-level Optimization

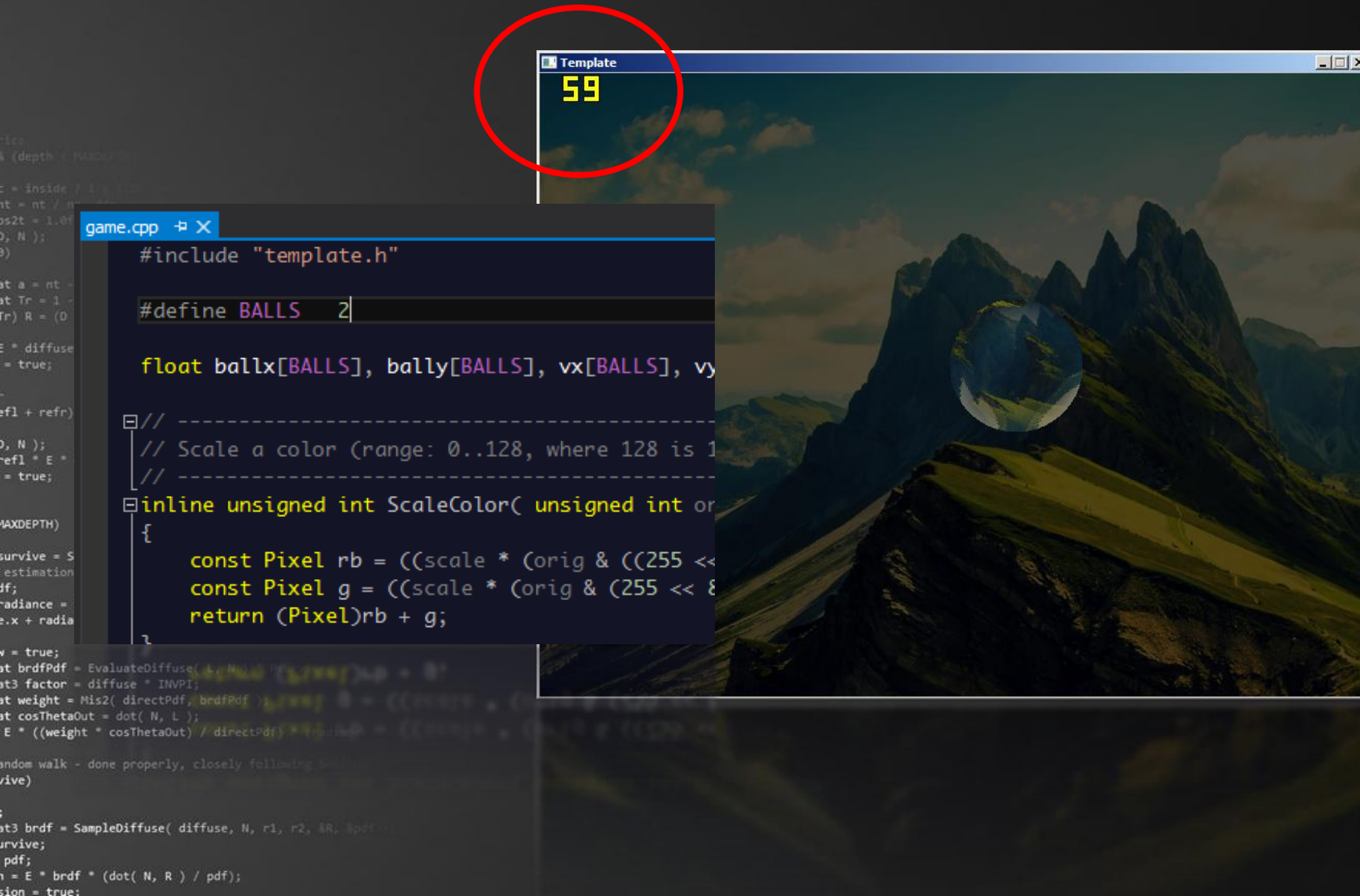
1. Avoid Costly Operations
2. Precalculate
3. Pick the Right Data Type
4. Avoid Conditional Branches
5. Early Out
6. Use the Power of Two
7. Do Things Simultaneously

Consistent Approach

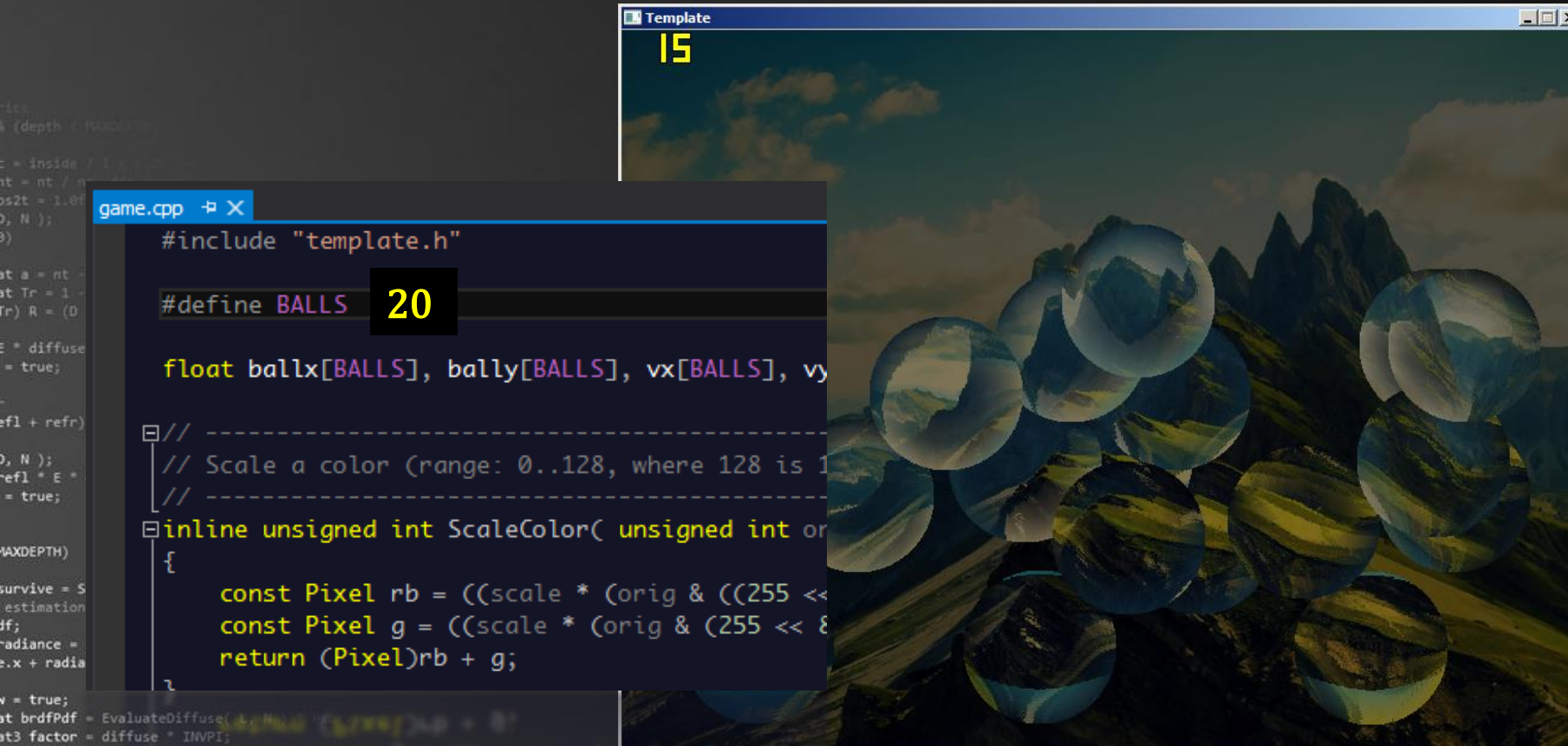
- (0.) Determine optimization requirements
1. **Profile: determine hotspots**
2. Analyze hotspots: determine scalability
3. Apply high level optimizations to hotspots
4. Profile again.
5. Parallelize
6. Use GPGPU
7. Profile again.
8. Apply low level optimizations to hotspots
9. Repeat steps 7 and 8 until time runs out
10. Report.



Gathering Intel



Gathering Intel



The image shows a code editor window titled "game.cpp" with a dark theme. A yellow highlight is on the line `#define BALLS 20`. To the right, a window titled "Template" shows a 3D scene with a blue sky and a mountain range. In the foreground, there is a cluster of reflective spheres that mirror the sky and mountains. The code in the editor includes a header file, a macro for the number of spheres, and a function to scale a color.

```

#include "template.h"

#define BALLS 20

float ballx[BALLS], bally[BALLS], vx[BALLS], vy[BALLS];

// -----
// Scale a color (range: 0..128, where 128 is 1.0)
// -----
inline unsigned int ScaleColor( unsigned int orig, float scale )
{
    const Pixel rb = ((scale * (orig & ((255 << 8) & 0xFF)) + 0.5f) >> 8);
    const Pixel g = ((scale * (orig & (255 << 8) & 0xFF)) + 0.5f) >> 8;
    return (Pixel)rb + g;
}

```



Gathering Intel

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (1.0f - n * nc);
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, clean
if;
radiance = SampleLight( @rand, I, @t, @light
e.x + radiance.y + radiance.z) > 0) && (rand
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (n
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, @R, @pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Tool 1:

VerySleepy™

(will tell us relative cost of parts of the code)

Tool 2:

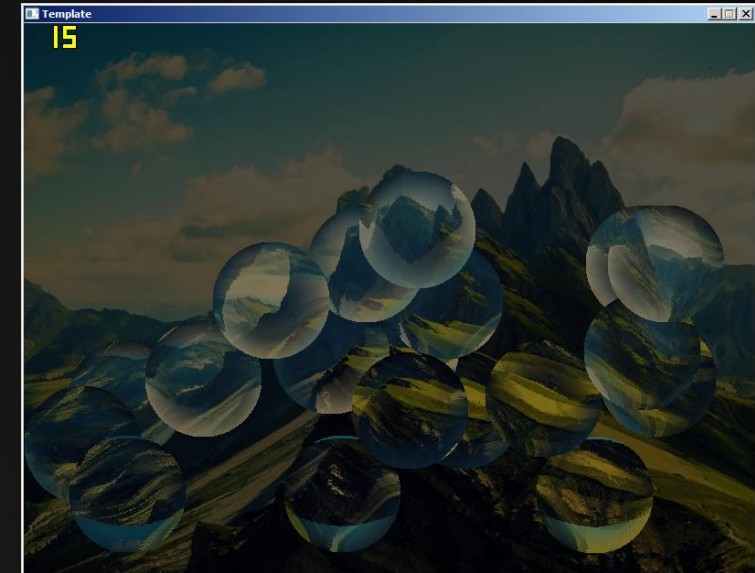
Custom timer

(will tell us absolute cost of large tasks)

Tool 3:

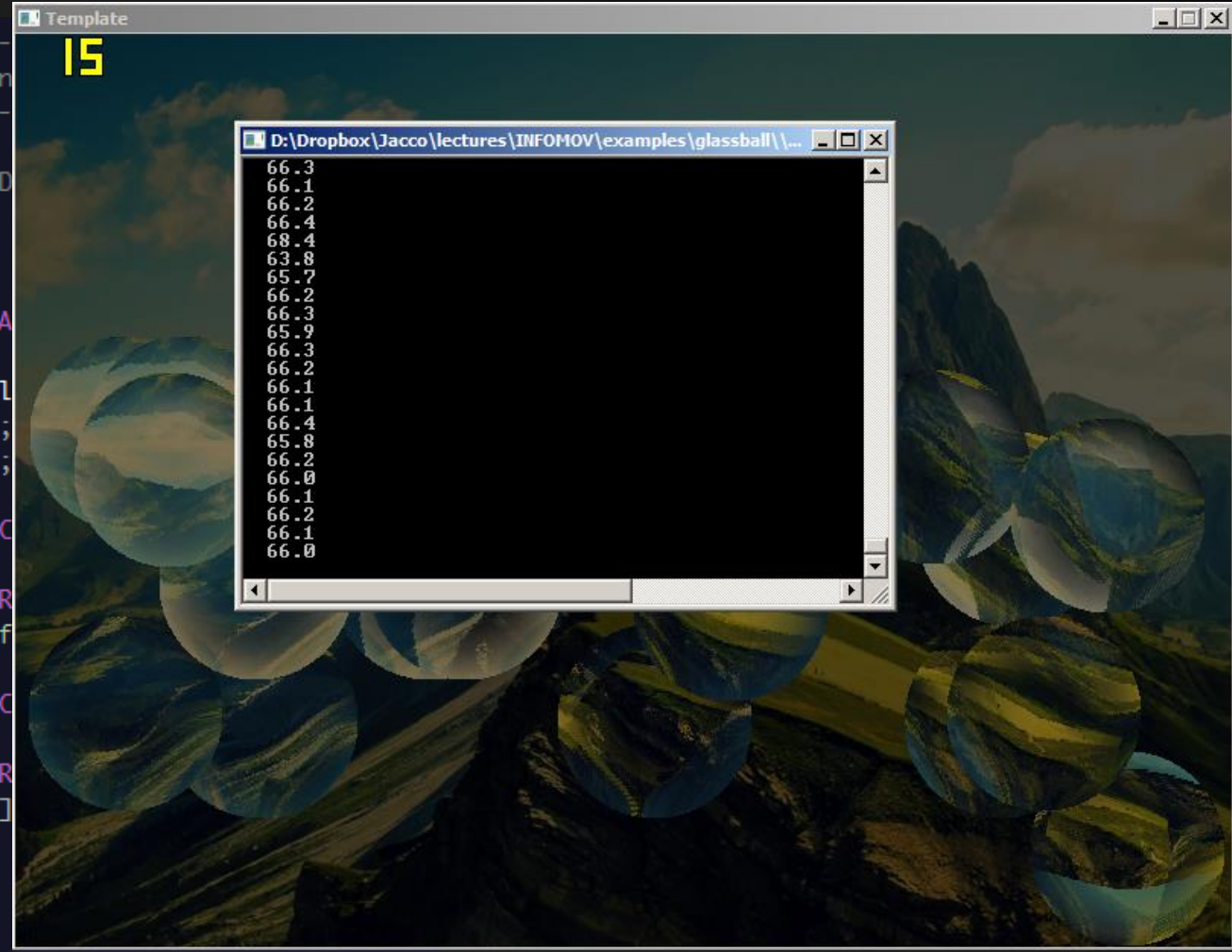
Running the app with various workloads

(to measure scalability)



Gathering Intel

```
// -----
// Main game tick function
// -----
timer t;
void Game::Tick( float a_D
{
    screen->Clear( 0 );
    DrawBackdrop();
    for( int i = 0; i < BA
    {
        DrawBall( (int)bal
        bally[i] += vy[i];
        ballx[i] += vx[i];
        vy[i] += 0.2f;
        if (bally[i] > (SC
        {
            bally[i] = SCR
            vy[i] = -0.96f
        }
        if (ballx[i] > (SC
        {
            ballx[i] = SCR
            vx[i] = -vx[i]
        }
        if (ballx[i] < 0)
        {
            ballx[i] = 0;
            vx[i] = -vx[i];
        }
    }
    printf( "%6.1f\n", t.elapsed() );
    t.reset();
}
```



Gathering Intel

```
// -----  
// Main game tick function  
// -----  
timer t;  
float smoothed = 60;  
bool firstFrame = true;  
void Game::Tick( float d_DT )  
{  
    screen->Clear( 0 );  
    DrawBackdrop();  
    for( int i = 0; i < BALLS; i++ )  
    {  
        DrawBall( (int)ballx[i], (int)bally[i] );  
        bally[i] += vy[i];  
        ballx[i] += vx[i];  
        vy[i] += 0.2f;  
        if (bally[i] > (SCRHEIGHT - 128))  
        {  
            bally[i] = SCRHEIGHT - 128;  
            vy[i] = -0.96f * vy[i];  
        }  
        if (ballx[i] > (SCRWIDTH - 128))  
        {  
            ballx[i] = SCRWIDTH - 128;  
            vx[i] = -vx[i];  
        }  
        if (ballx[i] < 0)  
        {  
            ballx[i] = 0;  
            vx[i] = -vx[i];  
        }  
    }  
    if (!firstFrame) smoothed = 0.95f * smoothed + 0.05f * t.elapsed();  
    firstFrame = false;  
    printf( "%0.1f\n", smoothed );  
    t.reset();  
}
```



Gathering Intel

Very Sleepy CS - C:\Users\Jacco\AppData\Local\Temp\BF5B.tmp

File View Help

Functions

Name	Exclu...	Inclusive	% Exclu...	% Inclusive	Module	Source File
Tmpl8::Game::DrawBall	18.08s	19.62s	81.00%	87.93%	Template	d:\dropbox\jacco\lectures\infomov\examples\glassball\game.cpp
Tmpl8::Game::DrawBackdrop	0.73s	0.81s	3.27%	3.62%	Template	d:\dropbox\jacco\lectures\infomov\examples\glassball\game.cpp
Tmpl8::Game::Tick	0.00s	21.43s	0.00%	96.05%	Template	d:\dropbox\jacco\lectures\infomov\examples\glassball\game.cpp
Tmpl8::Surface::Clear	0.41s	0.41s	1.83%	1.83%	Template	d:\dropbox\jacco\lectures\infomov\examples\glassball\game.cpp
SDL_main	0.00s	21.45s	0.00%	96.14%	Template	d:\dropbox\jacco\lectures\infomov\examples\glassball\game.cpp
_ftol2_sse	0.24s	0.24s	1.07%	1.07%	Template	f:\dd\vctools\crt\fpw32\tran\386
_ftol2_pentium4	1.55s	1.55s	6.93%	6.93%	Template	f:\dd\vctools\crt\fpw32\tran\386
__tmainCRTStartup	0.00s	21.56s	0.00%	96.63%	Template	f:\dd\vctools\crt\crtw32\dllstuff\c
WinMain	0.00s	21.56s	0.00%	96.63%	Template	d:\active\sd2-2.0.3\src\main\win
main	0.00s	21.56s	0.00%	96.63%	Template	d:\active\sd2-2.0.3\src\main\win
_imp__QueryPerformanceCounter	0.00s	0.75s	0.00%	3.35%	Template	[unknown]

Averages Call Stacks **Filters**

Main

Function name	Module	Source File
	Template	

Source Log

```

// -----
// Draw a glass ball using fake reflection & refraction
// -----
void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = image->GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        float dy = (float)(y - 64);
        for ( int x = 0; x < 128; x++ )
        {
            float dx = (float)(x - 64);
            float dist = sqrtf( dx * dx + dy * dy );
            if ( dist < 64 )

```

Source file: d:\dropbox\jacco\lectures\infomov\examples\glassball\game.cpp Line 1



Assess

```

...
    & (depth < MAXDEPTH)
...
    c = inside / (1 + cos2t);
    nt = nt / nc;
    cos2t = 1.0f - nnt * nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * c);
    Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( @rand, I, @t, @light;
e.x + radiance.y + radiance.z) > 0) && (rand(0,1) < survive)
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, @R, @pdf;
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```

Optimization targets:

1. DrawBall
2. DrawBackdrop
3. Surface::Clear

1. Avoid Costly Operations
2. Precalculate
3. Pick the Right Data Type
4. Avoid Conditional Branches
5. Early Out
6. Use the Power of Two
7. Do Things Simultaneously



Assess

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = image->GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        float dy = (float)(y - 64);
        for ( int x = 0; x < 128; x++ )
        {
            float dx = (float)(x - 64);
            float dist = sqrtf( dx * dx + dy * dy );
            if (dist < 64)
            {
                int xoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(x - 50) ));
                int yoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(y - 50) ));
                int u1 = ((bx + x) - 4 * xoffs) % SCRWIDTH;
                int v1 = ((by + y) - 4 * yoffs) % SCRHEIGHT;
                int u2 = ((bx + x) + 2 * xoffs) % SCRWIDTH;
                int v2 = ((by + y) + 2 * yoffs) % SCRHEIGHT;
                Pixel refl = src[max( 0, u1 ) + max( 0, v1 ) * image->GetWidth()];
                Pixel refr = src[max( 0, u2 ) + max( 0, v2 ) * image->GetWidth()];
                int reflscale = (int)(63.0f - 0.015f * (1 - dist) * (1 - dist));
                int refrscale = (int)(0.015f * (1 - dist) * (1 - dist));
                Pixel color1 = ScaleColor( refl, 63 - reflscale );
                Pixel color2 = ScaleColor( refr, 63 - refrscale );
                unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                dst[x + y * screen->GetPitch()] = red + grn + blu;
            }
        }
    }
}

```



Assess

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (1.0f - nct);
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, clean
if;
radiance = SampleLight( &rand, I, &t, &light
e.x + radiance.y + radiance.z) > 0) && (rand
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (rand
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Costly operations:

- **sinf, cosf**
- **sqrt**
- **%**

Precalculation opportunities:

- **xoffs, yoffs, dist**
- **reflscale, refrscale**

Data type issues:

- **float, int, unsigned int**

Conditional branches:

- **for loops, dist check, max**

Early out:

- **it's a circle (so: convex)**

Use the power of 2:

- **already happening**

Do things simultaneously:

- **?**

1. **Avoid Costly Operations**
2. **Precalculate**
3. **Pick the Right Data Type**
4. **Avoid Conditional Branches**
5. **Early Out**
6. **Use the Power of Two**
7. **Do Things Simultaneously**



Practical

```

...ics
& (depth < MAXDEPTH)
{
    // Inside / Outside
    int nt = nc; // inside
    int nc = nt; // outside
    double r = 1.0f - nnt * nnt;
    double D, N );
}

// Inside / Outside
int a = nt - nc; b = nt;
int Tr = 1 - (R0 + (1 - R0) * r);
int R = (D * nnt - N * (1 - nnt));

E * diffuse;
= true;

...efl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    -efl * E * diffuse;
    = true;

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &t, &light );
e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
{
    v = true;
    int brdfPdf = EvaluateDiffuse( L, N );
    int3 factor = diffuse * INVPI;
    int weight = Mis2( directPdf, brdfPdf );
    int cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance);

random walk - done properly, closely following
ive)

;
int3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

PART 1

- Basics -

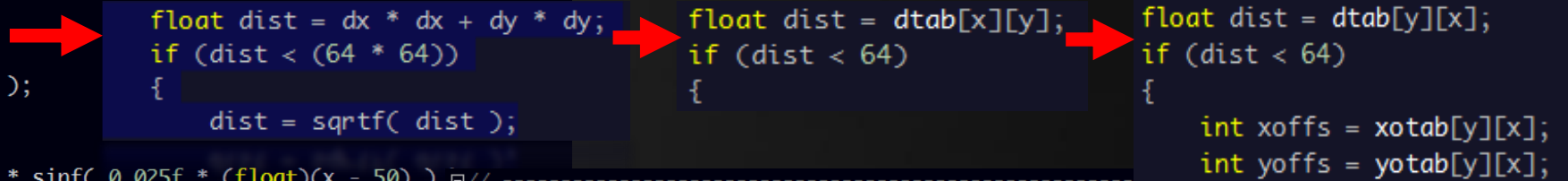


Practical



24.5
24.3
24.5
24.3
24.5
24.3
24.5
24.3
24.5
24.3
24.5
24.3

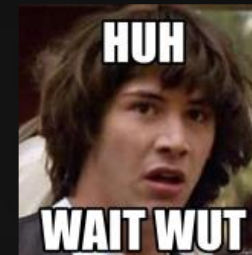
```
void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = image->GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        float dy = (float)(y - 64);
        for ( int x = 0; x < 128; x++ )
        {
            float dx = (float)(x - 64);
            float dist = sqrtf( dx * dx + dy * dy );
            if (dist < 64)
            {
                int xoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(x - 50) ));
                int yoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(y - 50) ));
                int u1 = ((bx + x) - 4 * xoffs) % SCRWIDTH;
                int v1 = ((by + y) - 4 * yoffs) % SCRHEIGHT;
                int u2 = ((bx + x) + 2 * xoffs) % SCRWIDTH;
                int v2 = ((by + y) + 2 * yoffs) % SCRHEIGHT;
                Pixel refl = src[max( 0, u1 ) + max( 0, v1 ) * image->GetWidth()];
                Pixel refr = src[max( 0, u2 ) + max( 0, v2 ) * image->GetWidth()];
                int reflscale = (int)(63.0f - 0.015f * (1 - dist) * (1 - dist));
                int refrscale = (int)(0.015f * (1 - dist) * (1 - dist));
                Pixel color1 = ScaleColor( refl, 63 - reflscale );
                Pixel color2 = ScaleColor( refr, 63 - refrscale );
                unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xf
                unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x0
                unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0
                dst[x + y * screen->GetPitch()] = red + grn + blu;
            }
        }
    }
}
```



```
void Game::Init()
{
    image = new Surface( "testdata/mountains.png" );
    for( int i = 0; i < BALLS; i++ )
    {
        ballx[i] = 100 + Rand( 500 );
        bally[i] = 200 - Rand( 150 );
        vx[i] = (Rand( 1.0f ) > 0.5f) ? 1.6f : -1.6f;
        vy[i] = 0;
    }
    for( int x = 0; x < 128; x++ ) for( int y = 0; y < 128; y++ )
    {
        float dy = (float)(y - 64);
        float dx = (float)(x - 64);
        float dist = sqrtf( dx * dx + dy * dy );
        dtab[y][x] = dist;
        xotab[y][x] = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(x - 50) ));
        yotab[y][x] = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(y - 50) ));
    }
}
```



Practical



24.3
24.5
24.3
24.5
24.3
24.5
24.3
24.5
24.3
24.5
24.3
24.6

```
void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = image->GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        for ( int x = 0; x < 128; x++ )
        {
            float dist = dtab[y][x];
            if (dist < 64)
            {
                int xoffs = xotab[y][x];
                int yoffs = yotab[y][x];
                int u1 = ((bx + x) - 4 * xoffs) % SCRWIDTH;
                int v1 = ((by + y) - 4 * yoffs) % SCRHEIGHT;
                int u2 = ((bx + x) + 2 * xoffs) % SCRWIDTH;
                int v2 = ((by + y) + 2 * yoffs) % SCRHEIGHT;
                Pixel refl = src[max( 0, u1 ) + max( 0, v1 ) * image->GetPitch()];
                Pixel refr = src[max( 0, u2 ) + max( 0, v2 ) * image->GetPitch()];
                int reflscale = (int)(63.0f - 0.015f * (1 - dist) * (1 - dist));
                int refrscale = (int)(0.015f * (1 - dist) * (1 - dist));
                Pixel color1 = ScaleColor( refl, 63 - reflscale );
                Pixel color2 = ScaleColor( refr, 63 - refrscale );
                unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                dst[x + y * screen->GetPitch()] = red + grn + blu;
            }
        }
    }
}
```

```
struct lookupData
{
    float dist;
    int xoffs;
    int yoffs;
    int reflscale;
    int refrscale;
};
```

$$5 * 4 * 128 * 128 = 320Kb$$

```
for( int x = 0; x < 128; x++ ) for( int y = 0; y < 128; y++ )
{
    float dy = (float)(y - 64);
    float dx = (float)(x - 64);
    float dist = sqrtf( dx * dx + dy * dy );
    lookup[y][x].dist = dist;
    lookup[y][x].xoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(x - 50) ));
    lookup[y][x].yoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(y - 50) ));
    lookup[y][x].reflscale = 63 - (int)(63.0f - 0.015f * (1 - dist) * (1 - dist));
    lookup[y][x].refrscale = 63 - (int)(0.015f * (1 - dist) * (1 - dist));
}
```

```
dtab[y][x] = dist;
xotab[y][x] = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(x - 50) ));
yotab[y][x] = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(y - 50) ));
refltab[y][x] = (int)(63.0f - 0.015f * (1 - dist) * (1 - dist));
refrtab[y][x] = (int)(0.015f * (1 - dist) * (1 - dist));
```



Practical

Very Sleepy CS - C:\Users\Jacco\AppData\Local\Temp\F732.tmp

File View Help

Functions

Name	Exclu...	Inclusive	% Exclusive	% Inclusive	Module	Source File
Tmpl8::Game::DrawBall	14.81s	14.81s	67.98%	67.98%	Template	d:\dropbox\jacco\lectures\infomov\examples\...
Tmpl8::Game::DrawBackdrop	1.55s	2.27s	7.12%	10.43%	Template	d:\dropbox\jacco\lectures\infomov\examples\...
Tmpl8::Surface::Clear	0.31s	0.31s	1.44%	1.44%	Template	d:\dropbox\jacco\lectures\infomov\examples\...
[72F3F8DD]	0.28s	0.28s	1.30%	1.30%	Template	
[72F3F8B6]	0.24s	0.24s	1.10%	1.10%	Template	
[72F3F8E7]	0.05s	0.05s	0.24%	0.24%	Template	
[72F3F8EC]	0.05s	0.05s	0.23%	0.23%	Template	

Averages Call Stacks Filters

Main

Function Name	Module	Source File
	Template	

Source Log

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = image->GetBuffer();
    0.01s   for ( int y = 0; y < 128; y++ )
    {
        0.29s   for ( int x = 0; x < 128; x++ )
        {
            0.25s   float dist = lookup[y][x].dist;
                    if (dist < 64)
                    {
                        0.01s   int xoffs = lookup[y][x].xoffs;
                        0.01s   int yoffs = lookup[y][x].yoffs;
                        0.28s   int u1 = ((bx + x) - 4 * xoffs) % SCRWIDTH;
                        2.87s   int v1 = ((by + y) - 4 * yoffs) % SCRHEIGHT;
                        int u2 = ((bx + x) + 2 * xoffs) % SCRWIDTH;
                        int v2 = ((by + y) + 2 * yoffs) % SCRHEIGHT;
                        Pixel refl = src[max( 0, u1 ) + max( 0, v1 ) * image->GetWidth()];
                        Pixel refr = src[max( 0, u2 ) + max( 0, v2 ) * image->GetWidth()];
                        Pixel color1 = ScaleColor( refl, lookup[y][x].reflscale );
                        Pixel color2 = ScaleColor( refr, lookup[y][x].refrscale );
                        0.66s   unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                        0.53s   unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                        0.18s   unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                        1.20s   dst[x + y * screen->GetPitch()] = red + grn + blu;
                    }
        }
    }
}

```

Source file: d:\dropbox\jacco\lectures\infomov\examples\glassball - basics\game.cpp Line 1



Practical

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = image->GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        for ( int x = 0; x < 128; x++ )
        {
            float dist = lookup[y][x].dist;
            if (dist < 64)
            {
                int xoffs = lookup[y][x].xoffs;
                int yoffs = lookup[y][x].yoffs;
                int u1 = ((bx + x) - 4 * xoffs) % SCRWIDTH;
                int v1 = ((by + y) - 4 * yoffs) % SCRHEIGHT;
                int u2 = ((bx + x) + 2 * xoffs) % SCRWIDTH;
                int v2 = ((by + y) + 2 * yoffs) % SCRHEIGHT;
                Pixel refl = src[max( 0, u1 ) + max( 0, v1 ) * image->GetWidth()];
                Pixel refr = src[max( 0, u2 ) + max( 0, v2 ) * image->GetWidth()];
                Pixel color1 = ScaleColor( refl, lookup[y][x].reflscale );
                Pixel color2 = ScaleColor( refr, lookup[y][x].refrscale );
                unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                dst[x + y * screen->GetPitch()] = red + grn + blu;
            }
        }
    }
}

```

```
Surface large( SCRWIDTH * 3, SCRHEIGHT * 3 );
```

```

void Game::Init()
{
    image = new Surface( "testdata/mountains.png" );
    for( int x = 0; x < 3; x++ ) for( int y = 0; y < 3; y++ )
        image->CopyTo( &large, x * SCRWIDTH, y * SCRHEIGHT );
}

```

```

int u1 = ((bx + x) - 4 * xoffs) + SCRWIDTH;
int v1 = ((by + y) - 4 * yoffs) + SCRHEIGHT;
int u2 = ((bx + x) + 2 * xoffs) + SCRWIDTH;
int v2 = ((by + y) + 2 * yoffs) + SCRHEIGHT;
Pixel refl = src[u1 + v1 * large.GetWidth()];
Pixel refr = src[u2 + v2 * large.GetWidth()];

```

```

D:\Dropbox\Jacco\le
16.0
16.0
16.0
16.1
16.0
16.0
16.1
16.0
16.0
16.0
16.0
16.0
16.0
16.0
16.0
16.0
16.0
16.0
16.0
16.0

```



Practical

Very Sleepy CS - C:\Users\Jacco\AppData\Local\Temp\69A8.tmp

File View Help

Functions

Name	Exclu...	Inclusive	% Exclusive	% Inclusive	Module	Source File
Tmpl8::Game::DrawBall	14.60s	14.60s	75.46%	75.46%	Template	d:\dropbox\jacco
Tmpl8::Game::DrawBackdrop	1.37s	1.60s	7.08%	8.27%	Template	d:\dropbox\jacco
Tmpl8::Surface::Clear	0.69s	0.69s	3.54%	3.54%	Template	d:\dropbox\jacco
[101DD1A1]	0.02s	0.02s	0.09%	0.09%	Template	
[1003B078]	0.00s	0.00s	0.02%	0.02%	Template	
[1016ECD5]	0.00s	0.00s	0.02%	0.02%	Template	

Averages Call Stacks Filters

Main

Function Name

Module Template

Source File

Source Log

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = large.GetBuffer();
    0.01s   for ( int y = 0; y < 128; y++ )
    {
        0.48s   for ( int x = 0; x < 128; x++ )
        {
            0.46s   float dist = lookup[y][x].dist;
                    if (dist < 64)
                    {
                        0.06s   int xoffs = lookup[y][x].xoffs;
                                int yoffs = lookup[y][x].yoffs;
                                int u1 = ((bx + x) - 4 * xoffs) + SCRWIDTH;
                                int v1 = ((by + y) - 4 * yoffs) + SCRHEIGHT;
                                int u2 = ((bx + x) + 2 * xoffs) + SCRWIDTH;
                                int v2 = ((by + y) + 2 * yoffs) + SCRHEIGHT;
                                Pixel refl = src[u1 + v1 * large.GetWidth()];
                                Pixel refr = src[u2 + v2 * large.GetWidth()];
                                Pixel color1 = ScaleColor( refl, lookup[y][x].reflscale );
                                Pixel color2 = ScaleColor( refr, lookup[y][x].refrscale );
                                1.53s   unsigned int red = MIN( 0xffff0000, (color1 & 0xffff0000) + (color2 & 0xffff0000) );
                                1.32s   unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                                0.92s   unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                                2.69s   dst[x + y * screen->GetPitch()] = red + grn + blu;
                    }
        }
    }
}
    
```

Source file: d:\dropbox\jacco\lectures\infomov\examples\glassball - basics\game.cpp Line 1



Practical

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = large.GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        for ( int x = 0; x < 128; x++ )
        {
            float dist = lookup[y][x].dist;
            if (dist < 64)
            {
                int xoffs = lookup[y][x].xoffs;
                int yoffs = lookup[y][x].yoffs;
                int u1 = ((bx + x) - 4 * xoffs) + SCRWIDTH;
                int v1 = ((by + y) - 4 * yoffs) + SCRHEIGHT;
                int u2 = ((bx + x) + 2 * xoffs) + SCRWIDTH;
                int v2 = ((by + y) + 2 * yoffs) + SCRHEIGHT;
                Pixel refl = src[u1 + v1 * large.GetWidth()];
                Pixel refr = src[u2 + v2 * large.GetWidth()];
                Pixel color1 = ScaleColor( refl, lookup[y][x].reflscale );
                Pixel color2 = ScaleColor( refr, lookup[y][x].refrscale );
                unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                dst[x + y * screen->GetPitch()] = red + grn + blu;
            }
        }
    }
}

```

```

for ( int y = 0; y < 128; y++ )
{
    Pixel* dst2 = dst + y * screen->GetPitch();
    for ( int x = 0; x < 128; x++, dst2++ )

```

```
*dst2 = red + grn + blu;
```

D:\Dropbox\Jacco\lectures\INFOMOV

```

24.6
24.4
24.6
24.4
24.6
24.4
24.5
24.4
24.5
24.4
24.5
24.4
24.5
24.4
24.5
24.4
24.5
24.4
24.5
24.4
24.6
24.4
24.5
24.4

```



Practical

Very Sleepy CS - C:\Users\Jacco\AppData\Local\Temp\FA82.tmp

File View Help

Functions

Name	Exclu...	Inclusive	% Exclusive	% Inclusive	Module	Source File
Tpl8::Game::DrawBall		19.46s	72.95%	72.95%	Template	d:\dropbox\jacco\lect...
Tpl8::Game::DrawBackdrop		2.27s	8.52%	10.18%	Template	d:\dropbox\jacco\lect...
Tpl8::Surface::Clear		1.17s	4.39%	4.39%	Template	d:\dropbox\jacco\lect...
Tpl8::Surface::CopyTo		0.00s	0.02%	0.02%	Template	d:\dropbox\jacco\lect...
Tpl8::Game::Tick		0.00s	0.01%	89.61%	Template	d:\dropbox\jacco\lect...
startTime		0.00s	0.00%	0.03%	Template	[unknown]
imp_QueryPerformanceCounter		0.00s	0.00%	8.83%	Template	[unknown]

Source Log

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    Pixel* src = large.GetBuffer();
    0.04s   for ( int y = 0; y < 128; y++ )
    {
        0.04s   Pixel* dst2 = dst + y * screen->GetPitch();
        0.81s   for ( int x = 0; x < 128; x++, dst2++ )
        {
            0.59s   float dist = lookup[y][x].dist;
                if (dist < 64)
                {
                    0.08s   int xoffs = lookup[y][x].xoffs;
                        int yoffs = lookup[y][x].yoffs;
                        int u1 = ((bx + x) - 4 * xoffs) + SCRWIDTH;
                        int v1 = ((by + y) - 4 * yoffs) + SCRHEIGHT;
                        int u2 = ((bx + x) + 2 * xoffs) + SCRWIDTH;
                        int v2 = ((by + y) + 2 * yoffs) + SCRHEIGHT;
                        Pixel refl = src[u1 + v1 * large.GetWidth()];
                        Pixel refr = src[u2 + v2 * large.GetWidth()];
                        Pixel color1 = ScaleColor( refl, lookup[y][x].reflscale );
                        Pixel color2 = ScaleColor( refr, lookup[y][x].refrscale );
                        2.38s   unsigned int red = MIN( 0xffff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                        2.10s   unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                        1.66s   unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                        1.27s   *dst2 = red + grn + blu;
                }
            }
        }
    }
}
    
```

Source file: d:\dropbox\jacco\lectures\infomov\examples\glassball - basics\game.cpp Line 1

Averages Call Stacks Filters

Main

Function Name	Module	Source File
	Template	



Practical

```

    }
    & (depth < MAXDEPTH)
    {
        // Inside / Outside
        int nt = nt / nc, ndd = ndd / nc;
        double cos2t = 1.0f - nnt * nnt;
        Vec D, N );
    }

    Vec a = nt - nc, b = nt + nc;
    double Tr = 1 - (R0 + (1 - R0) * cos2t);
    Vec R = (D * nnt - N * (1 - nnt));

    E * diffuse;
    bool = true;

    // Refractive index
    double refl + refr)) && (depth < MAXDEPTH)
    {
        Vec D, N );
        refl * E * diffuse;
        bool = true;
    }

    MAXDEPTH)

    survive = SurvivalProbability( diffuse );
    // Estimation - doing it properly, classically
    if (
        radiance = SampleLight( &rand, I, &t, &light );
        Vec e(x + radiance.y + radiance.z) > 0) && (depth <
    {
        bool = true;
        Vec brdfPdf = EvaluateDiffuse( L, N ) * survive;
        Vec3 factor = diffuse * INVPI;
        Vec3 weight = Mis2( directPdf, brdfPdf );
        Vec3 cosThetaOut = dot( N, L );
        Vec3 E = ((weight * cosThetaOut) / directPdf) * radiance;
    }

    // Random walk - done properly, closely following Monte Carlo
    survive)

    Vec3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    Vec3 pdf;
    Vec3 r1 = E * brdf * (dot( N, R ) / pdf);
    bool = true;

```

PART 2

- Advanced -



Practical

```

...
    & (depth < MAXDEPTH)
...
    inside / 1.0f;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, close
if;
radiance = SampleLight( @rand, I, R, Align
e.x + radiance.y + radiance.z) > 0) && (refr
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Re
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Observations:

- We are suffering cache misses – a lot.
- Scaling colors is expensive.

Solution:

Reduce data size.

Lookup data:

- Reflscale, refrscale are in the range 0..63 (so: 6 bits each);
- Xoffs, yoffs are in the range ~ -80..+80.

So, each should fit in a byte. ☺



Practical

```

struct lookupData
{
    float dist;
    unsigned int data;
};

```

```

void Game::Init()
{
    image = new Surface( "testdata/mountains.png" );
    for( int x = 0; x < 3; x++ ) for( int y = 0; y < 3; y++ )
        image->CopyTo( &large, x * SCRWIDTH, y * SCRHEIGHT );
    for( int i = 0; i < BALLS; i++ )
    {
        ballx[i] = 100 + Rand( 500 );
        bally[i] = 200 - Rand( 150 );
        vx[i] = (Rand( 1.0f ) > 0.5f) ? 1.6f : -1.6f;
        vy[i] = 0;
    }
    for( int x = 0; x < 128; x++ ) for( int y = 0; y < 128; y++ )
    {
        float dy = (float)(y - 64);
        float dx = (float)(x - 64);
        float dist = sqrtf( dx * dx + dy * dy );
        lookup[y][x].dist = dist;
        int xoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(x - 50) ));
        int yoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(y - 50) ));
        int reflscale = 63 - (int)(63.0f - 0.015f * (1 - dist) * (1 - dist));
        int refrscale = 63 - (int)(0.015f * (1 - dist) * (1 - dist));
        lookup[y][x].data = reflscale + (refrscale << 8) + ((xoffs + 64) << 16) + ((yoffs + 64) << 24);
    }
}

```

```

unsigned int data = lookup[y][x].data;
int xoffs = ((data >> 16) & 255) - 64;
int yoffs = ((data >> 24) & 255) - 64;
int u1 = ((bx + x) - 4 * xoffs) + SCRWIDTH;
int v1 = ((by + y) - 4 * yoffs) + SCRHEIGHT;
int u2 = ((bx + x) + 2 * xoffs) + SCRWIDTH;
int v2 = ((by + y) + 2 * yoffs) + SCRHEIGHT;
Pixel refl = src[u1 + v1 * large.GetWidth()];
Pixel refr = src[u2 + v2 * large.GetWidth()];
Pixel color1 = ScaleColor( refl, data & 255 );
Pixel color2 = ScaleColor( refr, (data >> 8) & 255 );

```



Practical

Observations:

- ~~▪ We are suffering cache misses a lot.~~
- Scaling colors is expensive.

Solution:

Precalculate scaling.

```

void Game::DrawBackdrop()
{
    Pixel* pal = image8->palette[20];
    for( int y = 0; y < SCRHEIGHT; y++ )
    {
        unsigned char* src = image8->GetBuffer() + y * 1024;
        Pixel* dst = screen->GetBuffer() + y * SCRWIDTH;
        for( int x = 0; x < SCRWIDTH; x++ ) *dst++ = pal[*src++];
    }
}
    
```

```

class Surface8
{
public:
    Surface8( char* a_File );
    Surface8( int a_Width, int a_Height );
    ~Surface8();
    unsigned char* GetBuffer() { return m_Buffer; }
    Pixel* GetPalette( int a_Idx ) { return palette[a_Idx]; }
    int GetWidth() { return m_Width; }
    int GetHeight() { return m_Height; }
    void CopyTo( Surface8* a_Dst, int a_X, int a_Y );
    void LoadImage( char* a_File );
    unsigned char* m_Buffer;
    Pixel* palette[PALETTE_LEVELS];
    int m_Width, m_Height, m_Pitch;
};
    
```

```

for( int i = 0; i < PALETTE_LEVELS; i++ )
{
    palette[i] = new Pixel[256];
    for( int j = 0; j < 256; j++ )
    {
        int r = min( 255, (pal[j].rgbRed * i) >> 6 );
        int g = min( 255, (pal[j].rgbGreen * i) >> 6 );
        int b = min( 255, (pal[j].rgbBlue * i) >> 6 );
        palette[i][j] = (r << 16) + (g << 8) + b;
    }
}
    
```

```

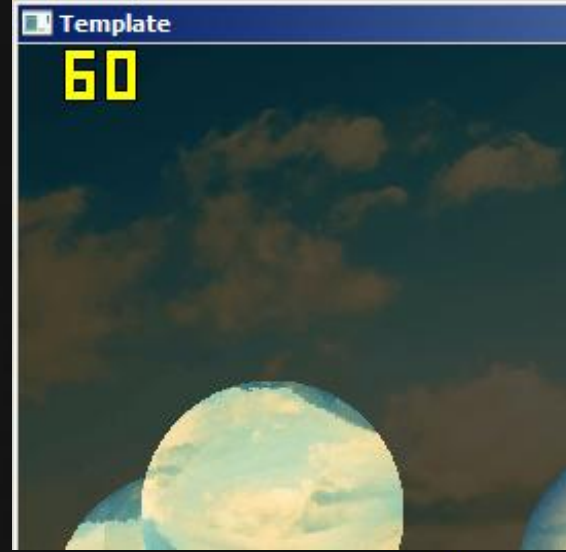
image8 = new Surface8( "testdata/mountains.png" );
for( int x = 0; x < 3; x++ ) for( int y = 0; y < 3; y++ )
    image8->CopyTo( &large8, x * SCRWIDTH, y * SCRHEIGHT );
for( int i = 0; i < PALETTE_LEVELS; i++ ) large8.palette[i] = image8->palette[i];
    
```

Practical

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    unsigned char* src = large8.GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        Pixel* dst2 = dst + y * screen->GetPitch();
        for ( int x = 0; x < 128; x++, dst2++ )
        {
            float dist = lookup[y][x].dist;
            if (dist < 64)
            {
                int xoffs = lookup[y][x].xoffs;
                int yoffs = lookup[y][x].yoffs;
                int u1 = ((bx + x) - 4 * xoffs) + SCRWIDTH;
                int v1 = ((by + y) - 4 * yoffs) + SCRHEIGHT;
                int u2 = ((bx + x) + 2 * xoffs) + SCRWIDTH;
                int v2 = ((by + y) + 2 * yoffs) + SCRHEIGHT;
                uint pixelIdx1 = src[u1 + v1 * large8.GetWidth()];
                uint pixelIdx2 = src[u2 + v2 * large8.GetWidth()];
                uint paletteIdx1 = lookup[y][x].reflscale;
                uint paletteIdx2 = lookup[y][x].refrscale;
                Pixel color1 = large8.palette[paletteIdx1][pixelIdx1];
                Pixel color2 = large8.palette[paletteIdx2][pixelIdx2];
                unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                *dst2 = red + grn + blu;
            }
        }
    }
}

```



Practical

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    unsigned char* src = large8.GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        Pixel* dst2 = dst + y * screen->GetPitch();
        for ( int x = 0; x < 128; x++, dst2++ )
        {
            float dist = lookup[y][x].dist;
            if (dist < 64)
            {
                int xoffs = lookup[y][x].xoffs;
                int yoffs = lookup[y][x].yoffs;
                int u1 = ((bx + x) - 4 * xoffs) + SCRWIDTH;
                int v1 = ((by + y) - 4 * yoffs) + SCRHEIGHT;
                int u2 = ((bx + x) + 2 * xoffs) + SCRWIDTH;
                int v2 = ((by + y) + 2 * yoffs) + SCRHEIGHT;
                uint pixelIdx1 = src[u1 + v1 * large8.GetWidth()];
                uint pixelIdx2 = src[u2 + v2 * large8.GetWidth()];
                uint paletteIdx1 = lookup[y][x].reflscale;
                uint paletteIdx2 = lookup[y][x].refrscale;
                Pixel color1 = large8.palette[paletteIdx1][pixelIdx1];
                Pixel color2 = large8.palette[paletteIdx2][pixelIdx2];
                unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                *dst2 = red + grn + blu;
            }
        }
    }
}

```

What if...

- We precalculate which pixels are inside the ball?

We could store, per pixel:

- ~~x, y~~
- an offset from the top-left corner
- u1, v1, u2, v2 (just add bx, by)

We could drop:

- one loop
- xoffs, yoffs, dist
- The radius test

Let's do this. 😊



Practical

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    unsigned char* src = large8.GetBuffer();
    for ( int y = 0; y < 128; y++ )
    {
        Pixel* dst2 = dst + y * screen->GetPitch();
        for ( int x = 0; x < 128; x++, dst2++ )
        {
            float dist = lookup[y][x].dist;
            if (dist < 64)
            {
                int xoffs = lookup[y][x].xoffs;
                int yoffs = lookup[y][x].yoffs;
                int u1 = ((bx + x) - 4 * xoffs) + SCRWIDTH;
                int v1 = ((by + y) - 4 * yoffs) + SCRHEIGHT;
                int u2 = ((bx + x) + 2 * xoffs) + SCRWIDTH;
                int v2 = ((by + y) + 2 * yoffs) + SCRHEIGHT;
                uint pixelIdx1 = src[u1 + v1 * large8.GetWidth()];
                uint pixelIdx2 = src[u2 + v2 * large8.GetWidth()];
                uint paletteIdx1 = lookup[y][x].reflscale;
                uint paletteIdx2 = lookup[y][x].refrscale;
                Pixel color1 = large8.palette[paletteIdx1][pixelIdx1];
                Pixel color2 = large8.palette[paletteIdx2][pixelIdx2];
                unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
                unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
                unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
                *dst2 = red + grn + blu;
            }
        }
    }
}

```

```

struct lookupData
{
    int reflscale;
    int refrscale;
    int offset;
    int u1, v1, u2, v2;
};

```

```

for( int x = 0; x < 128; x++ ) for( int y = 0; y < 128; y++ )
{
    float dy = (float)(y - 64);
    float dx = (float)(x - 64);
    float dist = sqrtf( dx * dx + dy * dy );
    if (dist < 64)
    {
        lookup[pixels].offset = x + y * SCRWIDTH;
        int xoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(x - 50) ));
        int yoffs = (int)((dist / 2 + 10) * sinf( 0.025f * (float)(y - 50) ));
        lookup[pixels].u1 = (x - 4 * xoffs) + SCRWIDTH;
        lookup[pixels].v1 = (y - 4 * yoffs) + SCRHEIGHT;
        lookup[pixels].u2 = (x + 2 * xoffs) + SCRWIDTH;
        lookup[pixels].v2 = (y + 2 * yoffs) + SCRHEIGHT;
        lookup[pixels].reflscale = 63 - (int)(63.0f - 0.015f * (1 - dist) * (1 - dist));
        lookup[pixels].refrscale = 63 - (int)(0.015f * (1 - dist) * (1 - dist));
        pixels++;
    }
}

```



Practical

```

void Game::DrawBall( int bx, int by )
{
    Pixel* dst = screen->GetBuffer() + bx + by * screen->GetPitch();
    unsigned char* src = large8.GetBuffer();
    for( int i = 0; i < pixels; i++ )
    {
        int u1 = bx + lookup[i].u1;
        int v1 = by + lookup[i].v1;
        int u2 = bx + lookup[i].u2;
        int v2 = by + lookup[i].v2;
        uint pixelIdx1 = src[u1 + v1 * large8.GetWidth()];
        uint pixelIdx2 = src[u2 + v2 * large8.GetWidth()];
        uint paletteIdx1 = lookup[i].reflscale;
        uint paletteIdx2 = lookup[i].refrscale;
        Pixel color1 = large8.palette[paletteIdx1][pixelIdx1];
        Pixel color2 = large8.palette[paletteIdx2][pixelIdx2];
        unsigned int red = MIN( 0xff0000, (color1 & 0xff0000) + (color2 & 0xff0000) );
        unsigned int grn = MIN( 0x00ff00, (color1 & 0x00ff00) + (color2 & 0x00ff00) );
        unsigned int blu = MIN( 0x0000ff, (color1 & 0x0000ff) + (color2 & 0x0000ff) );
        dst[lookup[i].offset] = red + grn + blu;
    }
}

```

```

void Game::DrawBall( int bx, int by )
{
    Pixel*dst=screen->GetBuffer()+bx+by*800;
    unsigned char*src=large8.GetBuffer();
    for(int i=0;i<pixels;i++)
    {
        int u1=bx+lookup[i].u1,v1=by+lookup[i].v1;
        int u2=bx+lookup[i].u2,v2=by+lookup[i].v2;
        uint a=src[u1+v1*large8.GetWidth()];
        uint b=src[u2+v2*large8.GetWidth()];
        uint c=lookup[i].reflscale,d=lookup[i].refrscale;
        Pixel e=large8.palette[c][a],f=large8.palette[d][b];
        uint r=MIN(0xff0000,(e&0xff0000)+(f&0xff0000));
        uint g=MIN(0x00ff00,(e&0x00ff00)+(f&0x00ff00));
        uint x=MIN(255,(e&255)+(f&255));
        dst[lookup[i].offset]=r+g+x;
    }
}

```

Final steps:

- Remove Clear(0).
- Precalculate scaled backdrop.



Practical

```

    }
    & (depth < MAXDEPTH)
    {
        double r1 = inside / (1.0 + abs(inside));
        double nt = nt / nc, ndd = ndd / nc;
        double cos2t = 1.0f - nt * nt;
        double u, v;
        Vec D, N );
    }

    Vec a = nt - nc, b = nt + nc;
    double Tr = 1 - (R0 + (1 - R0) * r1);
    double R = (D * nnt - N * (ndd * r1));

    E * diffuse;
    = true;

    }

    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }

    MAXDEPTH)

    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, clearly
    if;
    radiance = SampleLight( &rand, I, &t, &light );
    e.x + radiance.y + radiance.z) > 0) && (survive)
    {
        v = true;
        double brdfPdf = EvaluateDiffuse( L, N ) * survive;
        double t3 factor = diffuse * INVPI;
        double weight = Mis2( directPdf, brdfPdf );
        double cosThetaOut = dot( N, L );
        double E * ((weight * cosThetaOut) / directPdf) * (radiance);
    }

    random walk - done properly, closely following Monte Carlo
    (survive)

    };
    double t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    r1 = E * brdf * (dot( N, R ) / pdf);
    = true;
}

```

PART 2

- Insane -



Practical

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (inside + outside);
    nt = nt / nc; ndd = ndd / n;
    cos2t = 1.0f - nt * nt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * t);
    Tr) R = (D * nnt - N * (ndd *
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse, //
estimation - doing it properly, clearly
if;
radiance = SampleLight( &rand, I, &t, &light, //
e.x + radiance.y + radiance.z) > 0) && (depth <
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Observations:

- many pixels are overdrawn...
- there is a big penalty for writing the final pixel.
- everything is running on a single core.
- we are not using the GPU.



/INFOMOV/

END of “Low Level (2)”

next lecture: “caching (2)”

```
ics
& (depth < MAXDEPTH)
{
    inside / inside
    nt = nt / nc;
    cos2t = 1.0f - nnt * nnt;
    D, N );
}

at a = nt - nc, b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * r);
Tr) R = (D * nnt - N * (a * r + b * (1 - r)));

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)
D, N );
-efl * E * diffuse;
= true;

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &t, &align, &pdf );
e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * radiance;

random walk - done properly, closely following
ive)

;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

