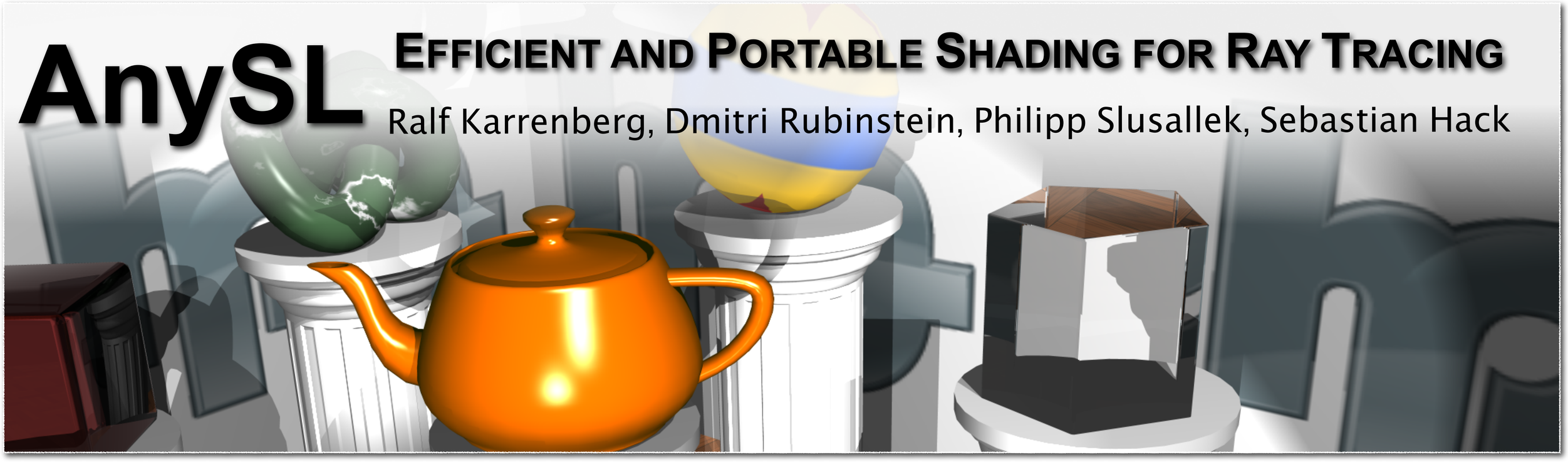
AnySL

Efficient and Portable Shading for Ray Tracing

Ralf Karrenberg, Dmitri Rubinstein, Philipp Slusallek, Sebastian Hack



Setting

There are three options how to implement shading in a ray tracer:

1) Native shading - internal, renderer language, fast, not portable, not flexible

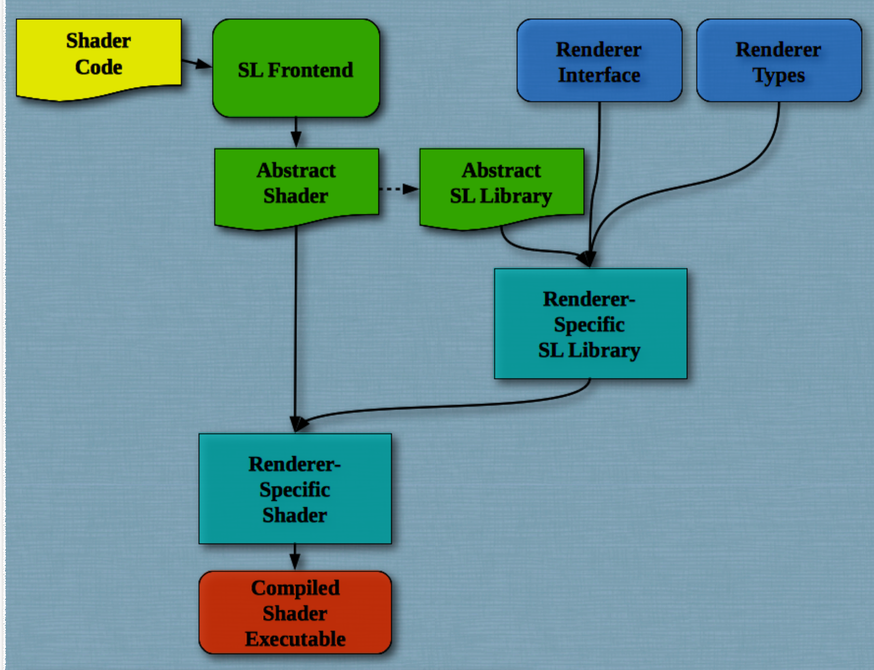
2) Shading languages - external, domain-specific, widely adopted, full compiler toolchain required

3) Shading languages via AnySL - “simplicity of interpreter, performance of native shading”

AnySL: A Library for High-Performance Shading

AnySL combines an embedded just-in-time compiler (LLVM), a customized linker that includes special code transformation passes, and a generic shading library for common operations. Together with the employment of a portable representation for shaders, these components enable a renderer-writer to integrate existing shading languages efficiently, flexible, and with minimized effort.

Example: RenderMan Lambertian Shader




```
surface lambertian(color Cc(1, 1, 1)) {
    color D = color(0, 0, 0);
    illuminance(P, N, PI/2) {
        D += Cl * L.N;
    }
    Ci = Cc * D;
}
```

```
void shade(void* context, void* params) {
    void* Cc = anysl_rsl_getParam(“Cc”);
    void* N = anysl_rsl_getPredef(“N”);
    void* P = anysl_rsl_getPredef(“P”);
    void* Ci = anysl_rsl_getPredef(“Ci”);
    void* D = anysl_rsl_alloca(“color”);
    void* PI_2 = anysl_rsl_alloca(“float”);
    anysl_rsl_illuminance(P, N, PI_2, body, body_args);
    anysl_rsl_mulvv(Ci, Cc, D);
}
void body(void* P, void* L, void* N, void* Cl, void* D) {
    void* tmp = anysl_rsl_alloca(“float”);
    anysl_rsl_dot(tmp, L, N);
    anysl_rsl_mulvf(D, Cl, tmp);
}

void anysl_rsl_reflect(void* R, void* I, void* N) {
    void* tmpv = anysl_rsl_alloca(“vector”);
    void* tmpf = anysl_rsl_alloca(“float”);
    void* c = anysl_rsl_float(2.f);

    anysl_rsl_dot(tmpf, i, n);
    anysl_rsl_mulff(tmpf, tmpf, c);
    anysl_rsl_mulvf(tmpv, n, tmpf);
    anysl_rsl_subvv(res, i, tmpv);
}
```

```
#include “Renderer/Types.h”
#include “Renderer/Interface.h”

typedef float Renderer::Float;
typedef struct { float x; float y; float z; }
        Renderer::Vector;

void shade(void* context, void* params) {
    ... // parameters & predefined values omitted
    Renderer::Color* cc = (Renderer::Color*)Cc;
    Renderer::Normal n = *(Renderer::Normal*)N;
    Renderer::Point p = *(Renderer::Point*)P;
    Renderer::Color* ci = (Renderer::Color*)Ci;
    Renderer::Vector d;
    Renderer::Float pi_2;

    // inlined illuminance()
    int lights = Renderer::getNumLights(context);
    Renderer::Light L;
    for (int i=0; i < lights; ++i) {
        L = Renderer::sampleLight(context, ...);
        if (!Renderer::lightContributes(context, L))
            continue;
        Renderer::Vector l = L.dir;
        Renderer::Vector cl = L.color;
        // inlined body
        float dot = l.x * n.x + l.y * n.y + l.z * n.z;
        d.x += cl.x * dot;
        d.y += cl.y * dot;
        d.z += cl.z * dot;
    }

    ci.x = cc.x * d.x;
    ci.y = cc.y * d.y;
    ci.z = cc.z * d.z;
}
```

```
#include “Renderer/Types.h”

void anysl_rsl_mulff(void* R, void* A, void* B) {
    Renderer::Float* res = (Renderer::Float*)R;
    Renderer::Float a = *(Renderer::Float*)A;
    Renderer::Float b = *(Renderer::Float*)B;

    Renderer::mulff(res, a, b);
}
```

Standard Arithmetic

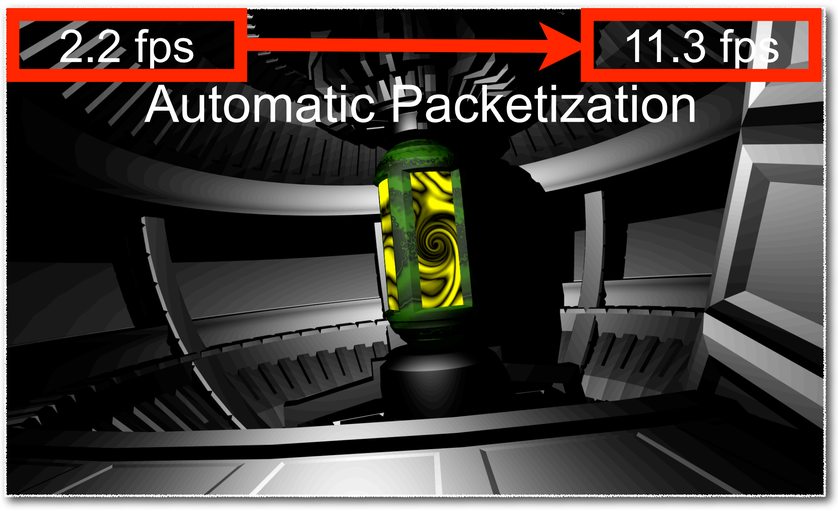
```
typedef float
        Renderer::Float;

void anysl_rsl_mulff(void* R, void* A, void* B) {
    Renderer::Float* res = (Renderer::Float*)R;
    Renderer::Float a = *(Renderer::Float*)A;
    Renderer::Float b = *(Renderer::Float*)B;

    *res = a * b;
}
```

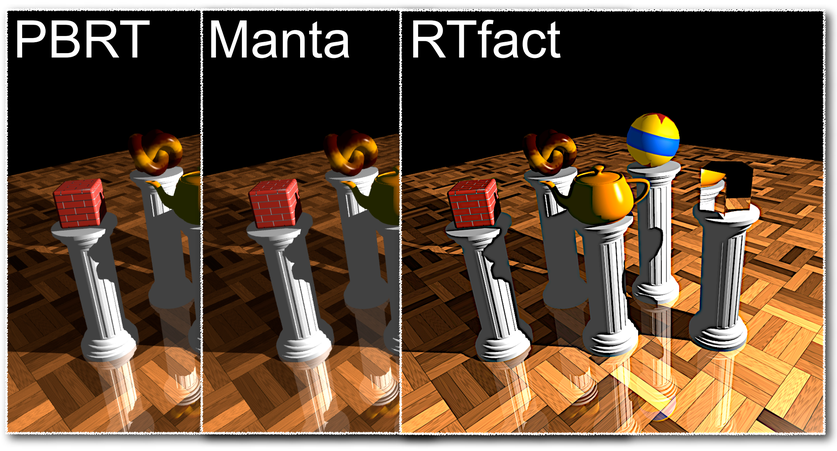
Automatic Differentiation

```
typedef struct { float f; float u; float v; }
        Renderer::Float;

void anysl_rsl_mulff(void* R, void* A, void* B) {
    Renderer::Float* res = (Renderer::Float*)R;
    Renderer::Float a = *(Renderer::Float*)A;
    Renderer::Float b = *(Renderer::Float*)B;

    res->f = a.f * b.f
    res->u = a.u * b.f + a.f * b.u;
    res->v = a.v * b.f + a.f * b.v;
}
```

Rasterization & GPUs

Supported backends include PTX, GLSL/HLSL, OpenCL. We obtained first results with AnySL integrated into a deferred-shading backend of the open-source rasterizer OGRE. RenderMan shaders are compiled to PTX from our portable threaded code and executed in the shading stage of OGRE.





Benefits

Efficiency

Use just-in-time compilation to “glue” shaders to the renderer and optimize wrapper-code away. On SIMD architectures, *Automatic Packetization* of scalar shaders speeds up rendering by a factor of 3.9 on average.

Portability

Shaders are compiled to *Subroutine-Threaded Code*: all operators are translated to function calls with abstract types, yielding renderer- and platform-independence. We demonstrate equal visual appearance across three different ray tracers: RTfact, Manta, and PBRT.

Simplicity

Implement a frontend, renderer-dependent language features, and an interpreter-like interface for your renderer instead of a full-fledged compiler toolchain.

Flexibility

Shaders can be recompiled at runtime, enabling features like editing of code and parameter specialization without sacrificing performance. Advanced techniques such as automatic differentiation can be implemented by resolving threaded code with different types and operations.

http://www.prog.uni-saarland.de/projects/anysl
{ karrenberg, rubinstein, slusallek, hack} @ cs.uni-saarland.de