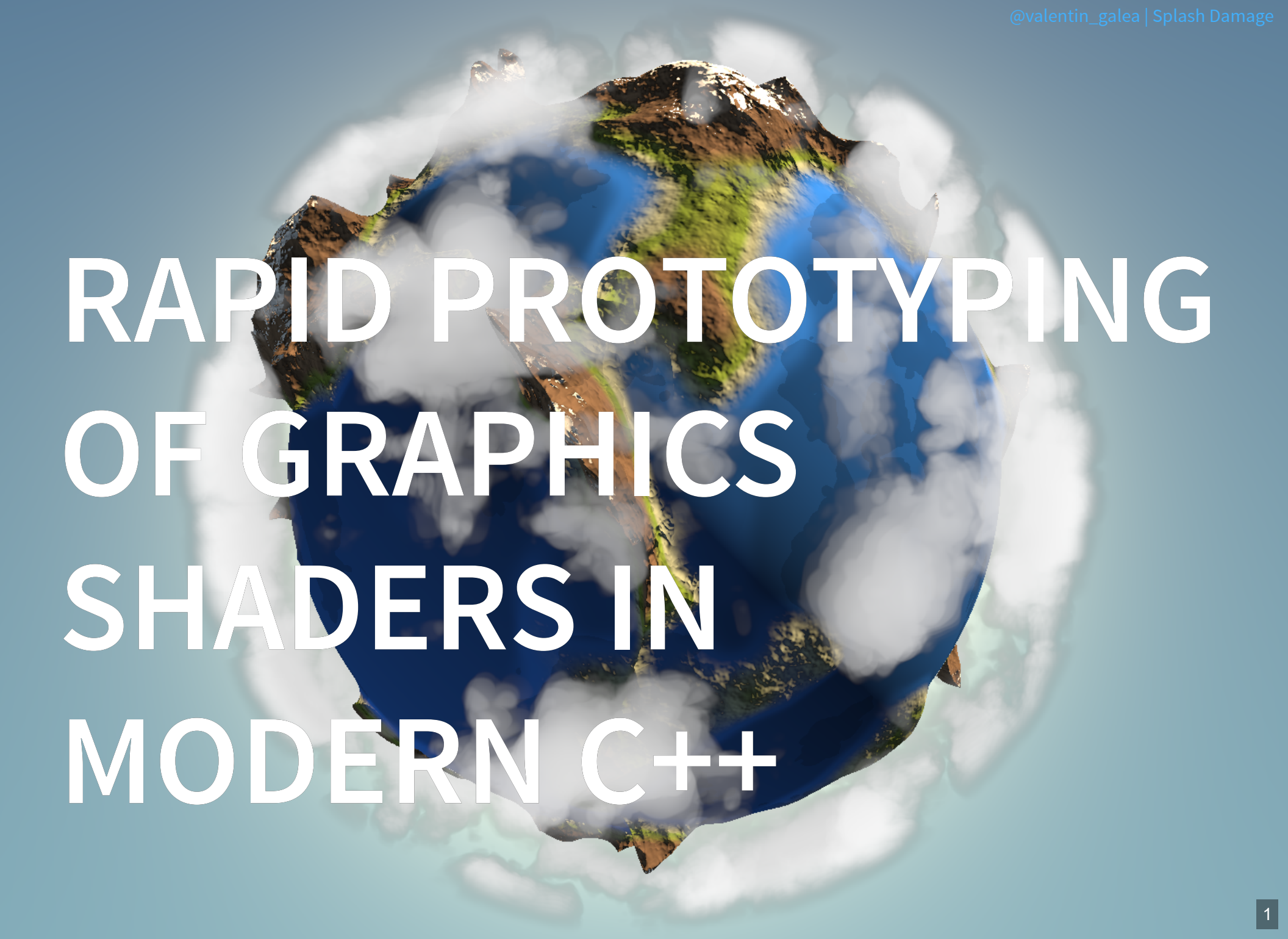


RAPID PROTOTYPING OF GRAPHICS SHADERS IN MODERN C++

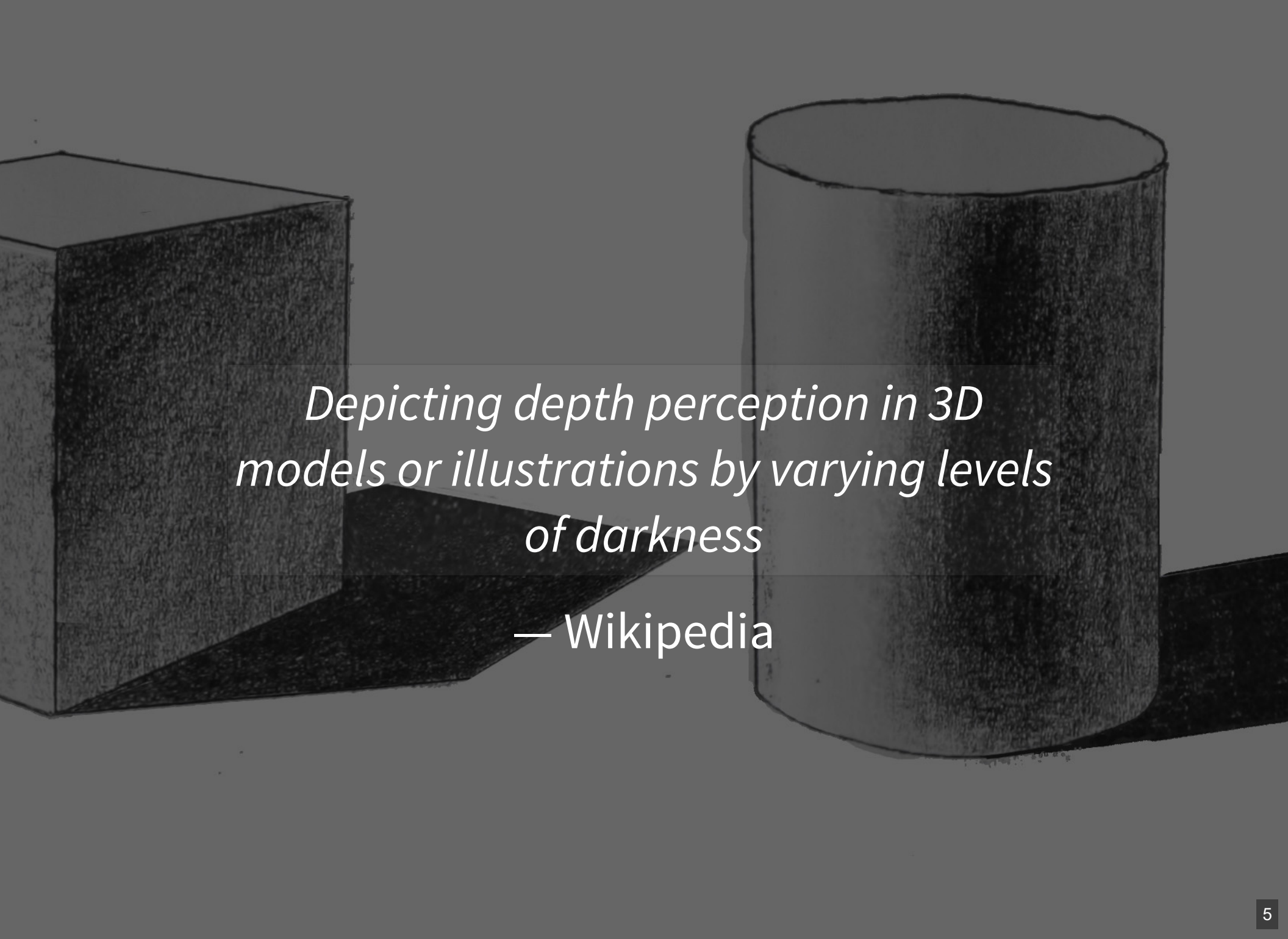

VALENTIN GALEA
@valentin_galea
More than 10 years in *mobile*, *indie* and *AAA* games
splash damage

AGENDA

- Intro and Motivation
- Shading Languages
- C++
- Showcase

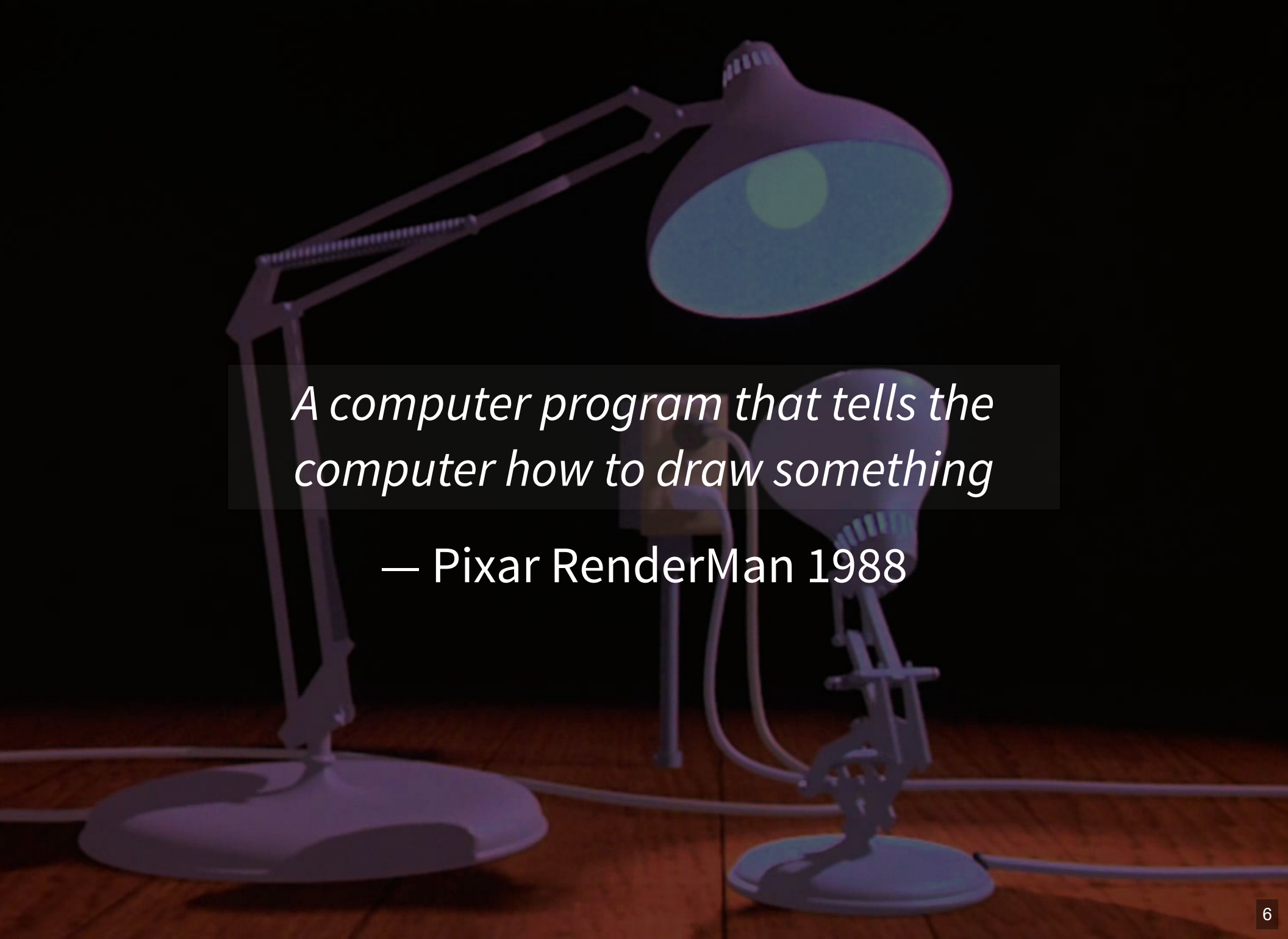
SHADERS

*A computer program that is used to do **shading***



Depicting depth perception in 3D models or illustrations by varying levels of darkness

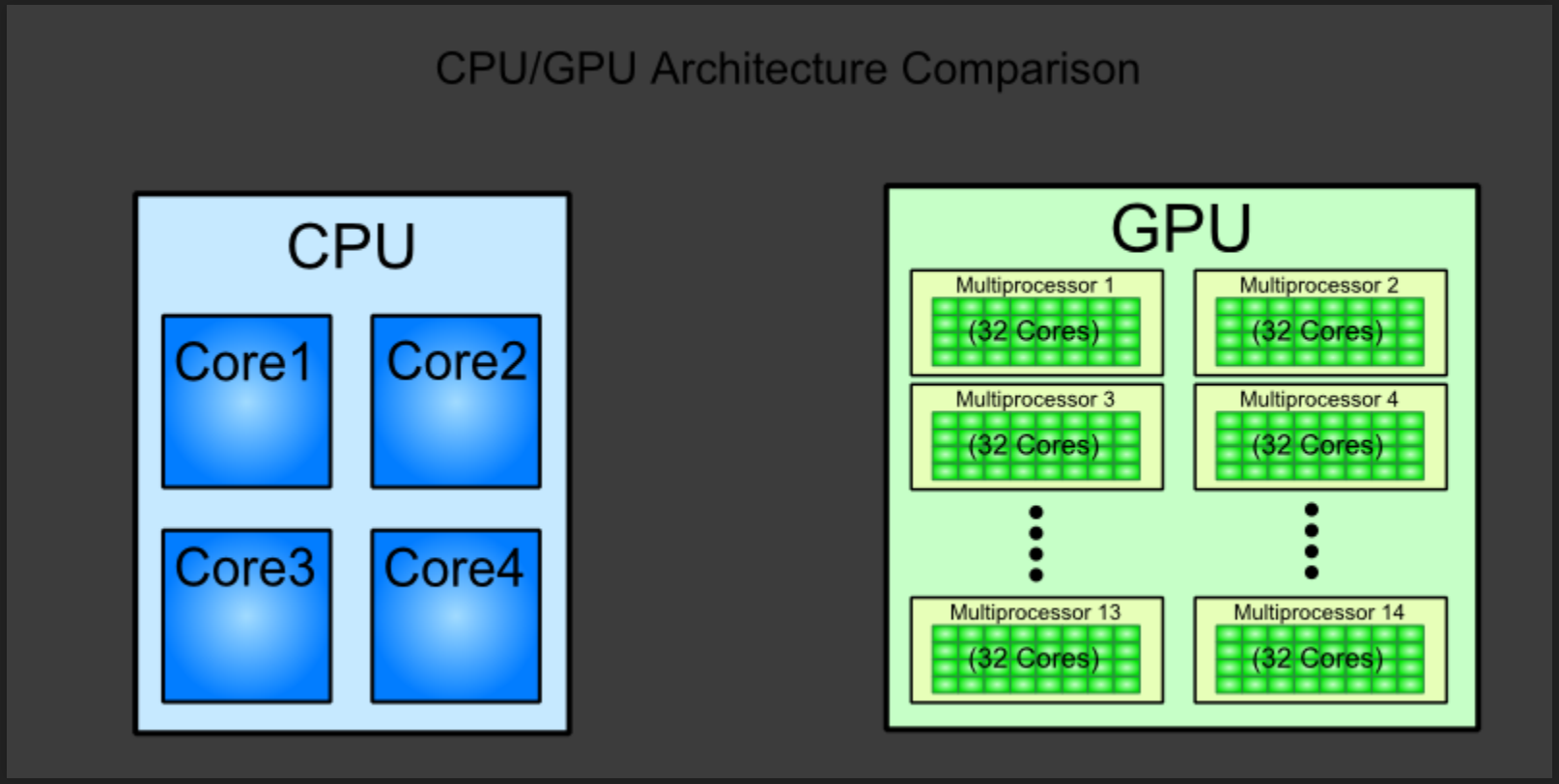
— Wikipedia


A computer program that tells the computer how to draw something
— Pixar RenderMan 1988

SHADERS ON MODERN GPU'S

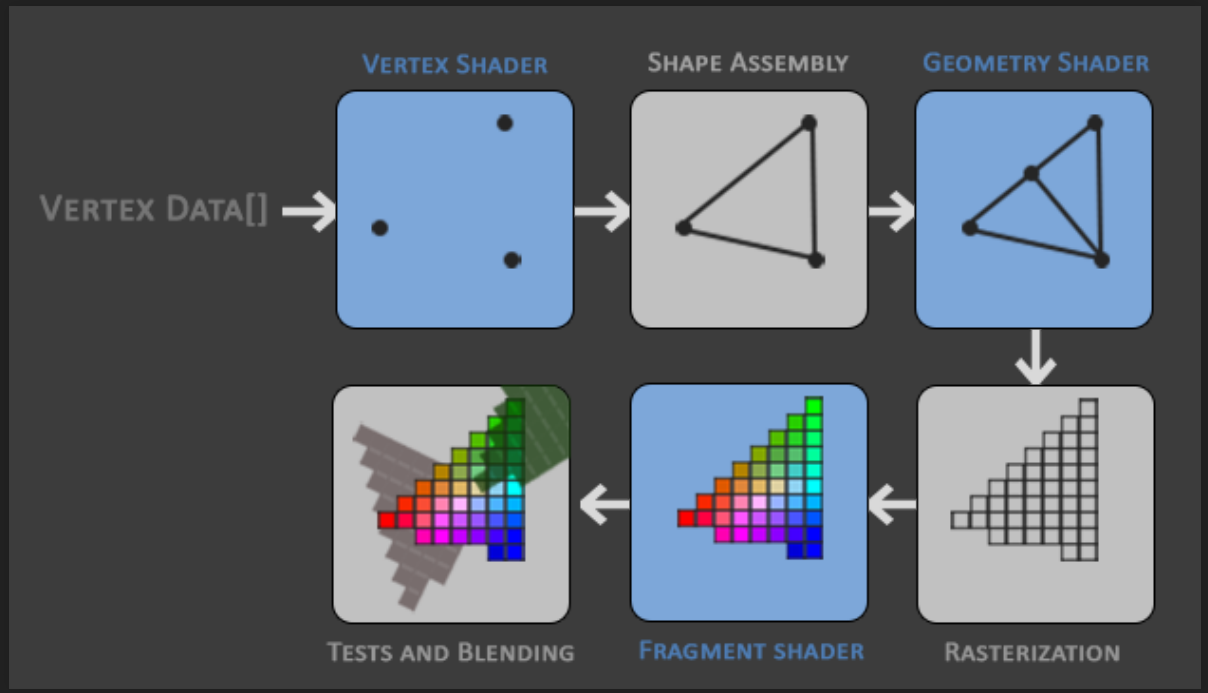
- Computer graphics / Images
- Highly parallel computing
- Mining for cryptocurrency :)

WHY ON GPU


CPU/GPU Architecture Comparison
CPU
Core1
Core2
Core3
Core4
GPU
Multiprocessor 1
(32 Cores)
Multiprocessor 2
(32 Cores)
Multiprocessor 3
(32 Cores)
Multiprocessor 4
(32 Cores)
Multiprocessor 13
(32 Cores)
Multiprocessor 14
(32 Cores)

TYPES OF SHADERS

Vertex
Geometry
Tessellation
Fragment(Pixel)
- - - - -
Compute



PIXEL/FRAGMENT SHADER

We will concentrate on (procedurally generated) image-only shaders

MOTIVATION

I wanted to create real-time effects like…

on more limited devices…

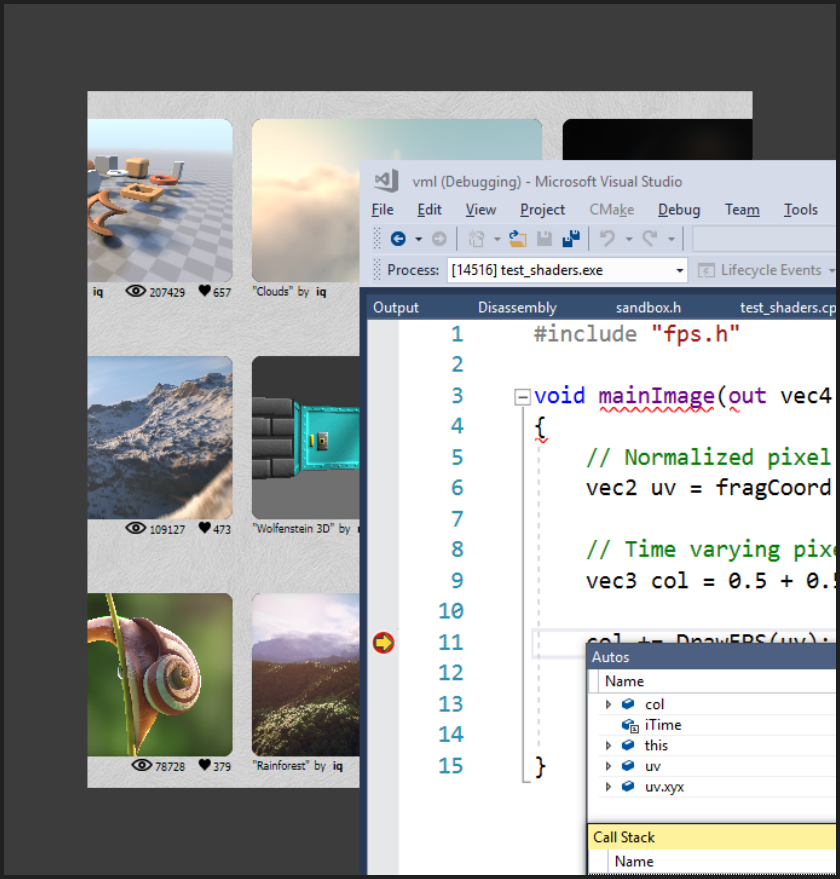

Victoria Station
1 h 3 min
every 20 min
Southfields Station
1 h 7 min
every 20 min
…because of long commutes!
London
Watford
Borehamwood
Enfield
Edgware
Harrow
Wembley
Uxbridge
Colne Valley Regional Park
High Wycombe
Beaconsfield
Marlow
Maidenhead
Slough
Windsor
Hayes
Egham
Richmond
Chigwell
Brentwood
Warley
Romford
Hornchurch
Ilford
Barking
Dagenham
River Thames
Grays
Bexleyheath
Dartford
Sidcup
Croydon
Orpington
Weybridge
Camberley
Frimley
Woking
Epsom
Leatherhead
Farnborough
Aldershot
Guildford
East Horsley
Surrey Hills Area of Outstanding Natural Beauty
Dorking
Redhill
Caterham
Oxted
Sevenoaks
West Kingsdown
Farnham
Godalming
Horley
Lingfield
Edenbridge
Hildenborough
Tonbridge
Cranleigh
Copthorne
Felbridge
East Grinstead
Royal Tunbridge
Google
Map data ©2018 Google
United Kingdom
Terms


also because GPU driver render bugs
taken on desktop PC / Nvidia GTX 1060

GIVEAWAY: VML

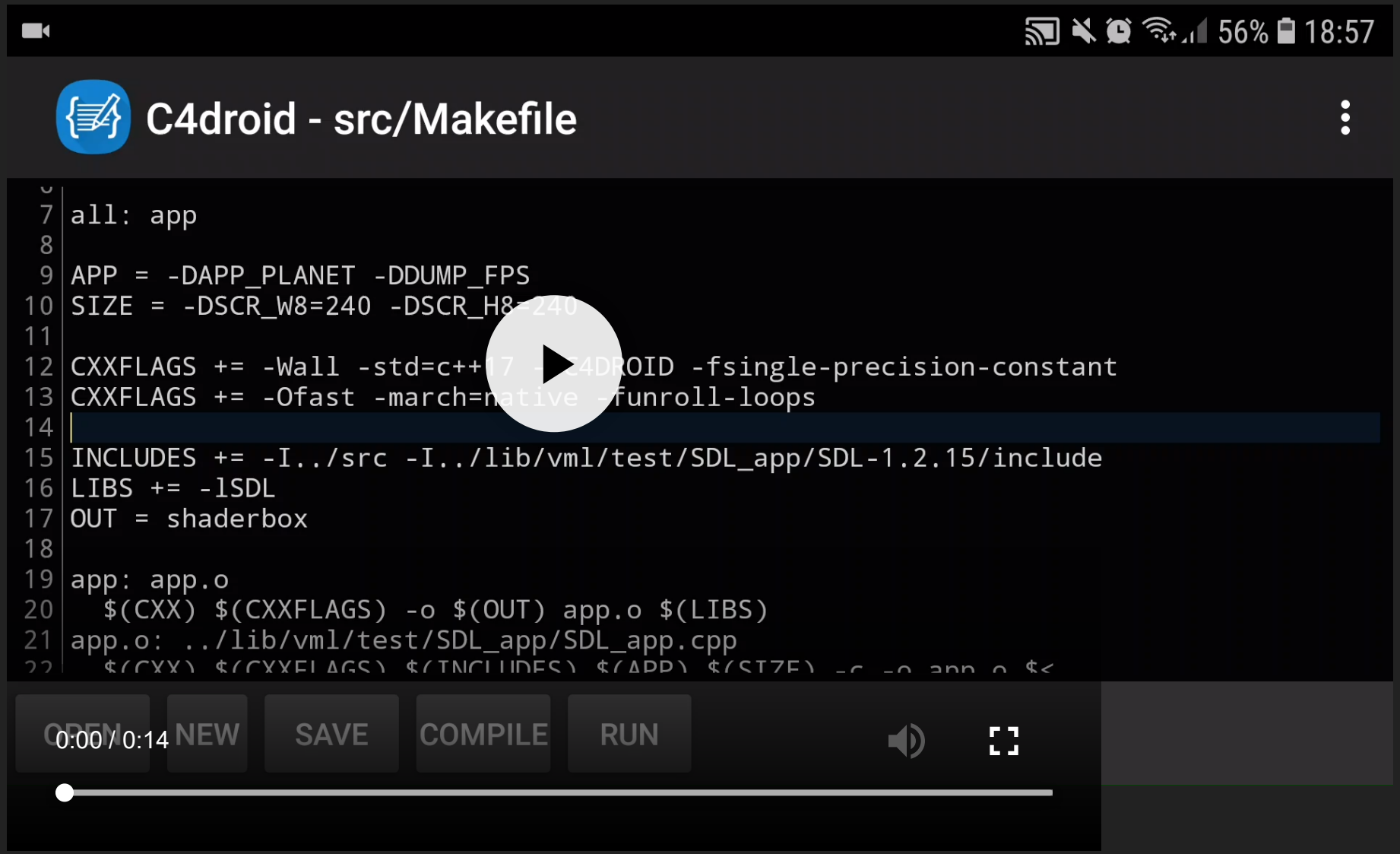
https://github.com/valentingalea/vml

- Debug / Decompile algorithms (from Shadertoy, etc)
- Texture generator (noise patterns, etc)
- Unit Test shaders!
- Quick prototype



DEMO: GPU VIA SHADERTOY.COM

DEMO: CPU VIA GCC 8.0 (MOBILE)



SHADING LANGUAGES

PIXAR RENDERMAN LANGUAGE

```
/*
 * red mesh
 */
surface basic() {
    Ci = (1.0, 0.0, 0.0);
    Oi = 1;
}
```

red shaded mesh

```
surface simple(color myOpacity = 1) {
    color myColor = (1.0, 0.0, 0.0);
    normal Nn = normalize(N);
    Ci = myColor * myOpacity * diff;
    Oi = myOpacity;
}
```

SHADING LANGUAGES HISTORY

For real-time rendering:

- Early: ARB assembly, Cg
- OpenGL shading language (**GLSL**)
- DirectX High-Level Shader Language (**HLSL**)
- PlayStation Shader Language (similar to HLSL)

GLSL VS HLSL

```
varying vec3 N;
varying vec3 v;
void main(void)
{
   vec3 L = normalize(gl_LightSource[0].position.xyz - v);
   vec4 Idiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0);
   Idiff = clamp(Idiff, 0.0, 1.0);
   gl_FragColor = Idiff;
}
```

```
float4 main(
    float3 Light: TEXCOORD0,
    float3 Norm : TEXCOORD1) : COLOR
{
    float4 diffuse = { 1.0, 0.0, 0.0, 1.0 };
    float4 ambient = { 0.1, 0.0, 0.0, 1.0 };
    return ambient + diffuse * saturate(dot(Light, Norm));
}
```

LANGUAGES: TYPES

	GLSL	HLSL
Scalar	`bool`, `int`, `uint`, `float`, `double`	
Vector	`vec2`, `vec3`, …	`float2`, `float3`, …
Matrix	`mat2`, `mat3`, …	`float2x2`, `float3x3`, …

…textures, samplers, precision modifiers etc

LANGUAGES: DECLARATIONS

GLSL	HLSL	C++
C-style types and arrays		
C-style `struct`		
`T name = T ( … )`	`T name = { … }`	both

LANGUAGES: FUNC ARGS

GLSL and HLSL	C++
`in T`	`T`
`out T`	`T &`
`inout T`	`T &`
`const […] T`	

VECTORS AND MATRICES

```
// vectors are generic
vec2 texcoord1, texcoord2;
vec3 position;
vec4 myRGBA;
ivec2 textureLookup;
bvec3 less;
```

```
// matrices are floating point only
mat2 mat2D;
mat3 optMatrix;
mat4 view, projection;
```

VECTOR SWIZZLE

Syntactic sugar for easy referring to components (or combination of)

{ x, y, z, w }	to represent points or normals
{ r, g, b, a }	to refer to colors (a is alpha/translucency)
{ s, t, p, q }	texture coordinates

VECTOR SWIZZLE - EXAMPLES

subcomponents mix & match

```
vec4 v4;
v4.rgba;  // is a vec4 and the same as just using v4,
v4.rgb;   // is a vec3,
v4.b;     // is a float,
v4.xy;    // is a vec2,
```

```
vec4 pos = vec4(1.0, 2.0, 3.0, 4.0);
vec4 swiz= pos.wzyx; // swiz = (4.0, 3.0, 2.0, 1.0)
vec4 dup = vec4(pos.xx, pos.yy);
```

l-value assignment

```
pos.xw = vec2(5.0, 6.0); // pos = (5.0, 2.0, 3.0, 6.0)
pos.xx = vec2(3.0, 4.0); // illegal - 'x' used twice
```

VECTOR SWIZZLE - MOTIVATION

```
vec3 calcNormal( in vec3 pos )
{
    vec2 e = vec2(1.0, -1.0) * 0.0005;

    return normalize(
        e.xyy * map( pos + e.xyy ).x +
        e.yyx * map( pos + e.yyx ).x +
        e.yxy * map( pos + e.yxy ).x +
        e.xxx * map( pos + e.xxx ).x );
}
```

OPERATORS

syntax	equivalent

`w = v + u;`	`w.x = v.x + u.x;`
	`w.y = v.y + u.y;`
	`w.z = v.z + u.z;`

`u = v * m;`	`u.x = dot(v, m[0]);`
	`u.y = dot(v, m[1]);`
	`u.z = dot(v, m[2]);`
	`/* dot(a,b) is the inner product of a and b */`

"STANDARD LIBRARY"

Math	`sin`, `cos`, `radians`, `pow`, `exp`, etc
Common	`abs`, `sign`, `floor`, `mod`, `min`, etc
Utility	`mix`, `step`, `smoothstep`, etc
Geometry	`length`, `dot`, `cross`, `distance`, etc

Specific texture and image sampling …

SHADING LANGUAGES FUTURE

Basically C++ (usually via LLVM)

SHADING LANGUAGES FUTURE

Basically C++ (usually via LLVM)

- Metal Shading Language (C++14, Apple)
  - only on iOS devices

SHADING LANGUAGES FUTURE

Basically C++ (usually via LLVM)

- Metal Shading Language (C++14, Apple)
  - only on iOS devices
- CUDA Heterogeneous Computing (C++11, NVidia)
  - only for computing, not graphics

SHADING LANGUAGES FUTURE

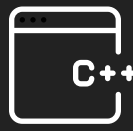
Basically C++ (usually via LLVM)

- Metal Shading Language (C++14, Apple)
  - only on iOS devices
- CUDA Heterogeneous Computing (C++11, NVidia)
  - only for computing, not graphics
- HLSL 6.x (C++98’ish, Microsoft)
  - not released yet

Let’s see how C++ can help out, NOW!

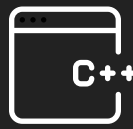
THE PLAN

THE PLAN



Pick a shading language and twist C++ to accept it as source code!

THE PLAN



Pick a shading language and twist C++ to accept it as source code!

BONUS: use the preprocessor for transcription back to the original language(s)!

THE PLAN (CONT.)

THE PLAN (CONT.)

- obligatory preprocessor layer

THE PLAN (CONT.)

- obligatory preprocessor layer
- vector (linear algebra) types
  - swizzle support

THE PLAN (CONT.)

- obligatory preprocessor layer
- vector (linear algebra) types
  - swizzle support
- matrix types

THE PLAN (CONT.)

- obligatory preprocessor layer
- vector (linear algebra) types
  - swizzle support
- matrix types
- operators

THE PLAN (CONT.)

- obligatory preprocessor layer
- vector (linear algebra) types
  - swizzle support
- matrix types
- operators
- "standard library" utility/math functions

PLACE YOUR BETS

We will chose **GLSL** as it's used on *desktop*, *web* and *mobile*

only a subset of it - concentrate on procedural graphics thus minimize/eliminate inputs (textures, vertex data, etc)

DESIGN OF `vector<>`

DESIGN OF vector<>

```
template<typename T, size_t N>
struct vector :
    public vector_base<T, N>
{
```

DESIGN OF vector<>

```
template<typename T, size_t N>
struct vector :
    public vector_base<T, N>
{
```

```
    vector();
    explicit vector(scalar_type s);
    template<typename... Args> explicit vector(Args... args);
```

DESIGN OF vector<>

```
template<typename T, size_t N>
struct vector :
    public vector_base<T, N>
{
```

```
    vector();
    explicit vector(scalar_type s);
    template<typename... Args> explicit vector(Args... args);
```

```
    scalar_type& operator[](size_t i);
    scalar_type& operator[](size_t i);

    vector_type& operator +=(scalar_type s);
    vector_type& operator +=(const vector_type& v);
(etc)
```

vector<> CTOR - BASIC

```
vector()
{
    static_for<0, N>()([this](size_t i) {
        data[i] = 0;
    });
}
```

```
explicit vector(scalar_type s)
{
    static_for<0, N>()([s, this](size_t i) {
        data[i] = s;
    });
}
```

`static_for` UTILITY

static_for UTILITY

```
template<size_t Begin, size_t End>
struct static_for
{
    template<class Func>
    constexpr void operator ()(Func&& f)
    {
        f(Begin);
        static_for<Begin + 1, End>()(std::forward<Func>(f));
    }
};
```

static_for UTILITY

```
template<size_t Begin, size_t End>
struct static_for
{
    template<class Func>
    constexpr void operator ()(Func&& f)
    {
        f(Begin);
        static_for<Begin + 1, End>()(std::forward<Func>(f));
    }
};
```

```
template<size_t N>
struct static_for<N, N>
{
    template<class Func>
    constexpr void operator ()(Func&&) { /* terminate */ }
};
```

`vector<>` CTOR - ADVANCED

vector<> CTOR - ADVANCED

```
template<typename A0, typename... Args,
```

vector<> CTOR - ADVANCED

```
template<typename A0, typename... Args,
```

```
    class = typename std::enable_if<
        ((sizeof... (Args) >= 1) ||
        ((sizeof... (Args) == 0) && !std::is_scalar_v<A0>))
    >::type>
```

vector<> CTOR - ADVANCED

```
template<typename A0, typename... Args,
```

```
    class = typename std::enable_if<
        ((sizeof... (Args) >= 1) ||
        ((sizeof... (Args) == 0) && !std::is_scalar_v<A0>))
    >::type>
```

```
explicit vector(A0&& a0, Args&&... args)
{
    static_recurse<0>(
        std::forward<A0>(a0),
        std::forward<Args>(args)...
    );
}
```

`vector<>` CTOR - ADVANCED (2)

vector<> CTOR - ADVANCED (2)

```
template<size_t I, typename Arg0, typename... Args>
void static_recurse(Arg0&& a0, Args&&... args)
{
    construct_at_index<I>(std::forward<Arg0>(a0));

    static_recurse<I + get_size<Arg0>()>(
        std::forward<Args>(args)...
    );
}
```

vector<> CTOR - ADVANCED (2)

```
template<size_t I, typename Arg0, typename... Args>
void static_recurse(Arg0&& a0, Args&&... args)
{
    construct_at_index<I>(std::forward<Arg0>(a0));

    static_recurse<I + get_size<Arg0>()>(
        std::forward<Args>(args)...
    );
}
```

```
template<size_t I>
void static_recurse()
{ /* terminate */ }
```

`vector<>` CTOR - ADVANCED (3)

vector<> CTOR - ADVANCED (3)

```
template<size_t i>
void construct_at_index(scalar_type arg)
{
    data[i] = arg;
}
```

vector<> CTOR - ADVANCED (3)

```
template<size_t i>
void construct_at_index(scalar_type arg)
{
    data[i] = arg;
}
```

```
template<size_t i, typename Other, size_t Other_N>
void construct_at_index(vector<Other, Other_N>&& arg)
{
    constexpr auto count = std::min(i + Other_N, num_components);

    static_for<i, count>([&](size_t j) {
        data[j] = arg.data[j - i];
    });
}
```

vector<> CTOR IN ACTION

```
using vec2 = vector<int, 2>;
using vec3 = vector<int, 3>;

vec3 v = vec3(98, vec2(99, 100));
//             ^    ^
//             |    |
//             `-- scalar construct gets called
//                  |
//                  `---- sub-vector construct gets called
//                        and then recursively again
```

GODBOLT

```
int main()
{
    float a, b;
    scanf("%f %f", &a, &b);

    auto v = vec3(1.f, vec2(a, b));

    printf("%f %f", v.x, v.y);
}
```

`-std=c++17 -Wall -O2` (source)

GODBOLT (CONT.)

5.x/6.x)

```
af
0, dword ptr [rsp + 4]
    xmm1, xmm0
0, dword ptr [rsp]
    xmm2, xmm0
0, qword ptr [rip + .LCPI0_0]
 offset .L.str.1
 3
ntf
```

gcc (7.x/8.x)

```
call    scanf
pxor    xmm2, xmm2
pxor    xmm1, xmm1
movsd   xmm0, QWORD PTR .LC1[rip]
mov     edi, OFFSET FLAT:.LC2
mov     eax, 3
cvtss2sd        xmm2, DWORD PTR [rsp+12]
cvtss2sd        xmm1, DWORD PTR [rsp+8]
call    printf
```

msvc (2017)

```
call    scanf
movss   xmm1, DWORD PTR b$
lea     rcx, OFFSET FLAT:`
movss   xmm0, DWORD PTR a$
movss   DWORD PTR $T1[rsp+
movsd   xmm1, QWORD PTR __
movss   DWORD PTR $T1[rsp]
movq    rdx, xmm1
mov     rax, QWORD PTR $T1
mov     QWORD PTR <args_1>
movss   xmm3, DWORD PTR <a
movss   xmm2, DWORD PTR <a
cvtps2pd xmm3, xmm3
cvtps2pd xmm2, xmm2
movq    r9, xmm3
movq    r8, xmm2
call    printf
```

PROBLEM: DEBUG

```
void static_for<0ul, 3ul>::operator()<vector<float, 3ul>...
    push    rbp
    mov     rbp, rsp
    sub     rsp, 32
    mov     QWORD PTR [rbp-24], rdi
    mov     QWORD PTR [rbp-32], rsi
    mov     rax, QWORD PTR [rbp-32]
    mov     esi, 0
    mov     rdi, rax
    call    vector<float, 3ul>::vector(float)::{lambda(unsigned...
    mov     rax, QWORD PTR [rbp-32]
    mov     rdi, rax
    call    vector<float, 3ul>::vector(float)::{lambda(unsigned...
    mov     rdx, rax
    lea     rax, [rbp-1]
    mov     rsi, rdx
    mov     rdi, rax
    call    void static_for<1ul, 3ul>::operator()<vector...
    nop
    leave
    ret
```

SOLUTION: C++17 FOLD EXPRESSIONS

Unary right fold

$(E\ \text{op}\ \ldots) \rightarrow (E_1\ \text{op}\ (\ldots\ \text{op}\ (E_{N-1}\ \text{op}\ E_N)))$

---

clang 3.6+ gcc 6+ MSVC 2017 15.5+

IMPROVED DESIGN OF `vector<>`

IMPROVED DESIGN OF vector<>

```
template<typename T, size_t... Ns>
struct vector :
    public vector_base<T, Ns...>
{
```

IMPROVED DESIGN OF vector<>

```
template<typename T, size_t... Ns>
struct vector :
    public vector_base<T, Ns...>
{
```

```
    vector()
    {
        ((data[Ns] = 0), ...);
    }
```

IMPROVED DESIGN OF vector<>

```
template<typename T, size_t... Ns>
struct vector :
    public vector_base<T, Ns...>
{
```

```
    vector()
    {
        ((data[Ns] = 0), ...);
    }
```

```
    explicit vector(scalar_type s)
    {
        ((data[Ns] = s), ...);
    }

    template<typename A0, typename... Args> explicit vector
(etc)
```

FOLDING EXPRESSIONS IN ACTION

declaration

```
explicit vector(scalar_type s)
{
    ((data[Ns] = s), ...);
}
```

instantiation

```
vector<float, 0, 1, 2>

explicit vector(float s)
{
    data[0] = s,
    data[1] = s,
    data[2] = s;
}
```

IMPROVED vector<> CTOR

```
template<typename A0, typename... Args>
explicit vector(A0&& a0, Args&&... args)
{
    size_t i = 0; // advances as we consume args

    // consume the first one
    construct_at_index(i, std::forward<A0>(a0));

    // consume the rest, if any
    (construct_at_index(i, std::forward<Args>(args)), ...);
}
```

SWIZZLING

vector_base NAIVE IMPL

```
template<typename T>
struct vector_base<T, 2>
{
    union
    {
        T data[2];
        struct { T x, y; };
        struct { T s, t; };
        struct { T u, v; };
    }
};
```

vector_base NAIVE IMPL (2)

```
template<typename T>
struct vector_base<T, 3>
{
    union
    {
        T data[3];
        struct { T x, y, z; };
        struct { T r, g, b; };
        struct { T s, t, p; };
    }
};
```

vector_base NAIVE IMPL (3)

```
template<typename T>
struct vector_base<T, 4>
{
    union
    {
        T data[4];
        struct { T x, y, z, w; };
        struct { T r, g, b, a; };
        struct { T s, t, p, q; };
    }
};
```

`vector_base` NOTES

- both anonymous `struct` and `union` are permitted, only MSVC complains with warning
- `union` active member switching can be tricky `[10.5]` but we'll use only trivial types with trivial assignment

SWIZZLE

We introduce an additional proxy class that allows custom access to the indices and we create all possible permutations (per GLSL/HLSL standard)

```
template<class vector_type, class T, size_t N, size_t... indices>
struct swizzler
{
    T data[N];

(etc)
```

SWIZZLE FOR vector<T, 3>

```
union
{
    T data[3];

    struct {
        swizzler<0>::type x;
        swizzler<1>::type y;
        swizzler<2>::type z;
    };
    struct {
        swizzler<0>::type r;
        swizzler<1>::type g;
        swizzler<2>::type b;
    };
    struct {
        swizzler<0>::type s;
        swizzler<1>::type t;
        swizzler<2>::type p;
    };
    ...
```

SWIZZLE (CONT.)

```
...
swizzler<0, 0>::type xx, rr, ss;
swizzler<0, 1>::type xy, rg, st;
swizzler<0, 2>::type xz, rb, sp;
swizzler<1, 0>::type yx, gr, ts;
swizzler<1, 1>::type yy, gg, tt;
swizzler<1, 2>::type yz, gb, tp;
swizzler<2, 0>::type zx, br, ps;
swizzler<2, 1>::type zy, bg, pt;
swizzler<2, 2>::type zz, bb, pp;
...
```

...MORE SWIZZLE

```
...
swizzler<0, 0, 0>::type xxx, rrr, sss;
swizzler<0, 0, 1>::type xxy, rrg, sst;
swizzler<0, 0, 2>::type xxz, rrb, ssp;
swizzler<0, 1, 0>::type xyx, rgr, sts;
swizzler<0, 1, 1>::type xyy, rgg, stt;
swizzler<0, 1, 2>::type xyz, rgb, stp;
swizzler<0, 2, 0>::type xzx, rbr, sps;
swizzler<0, 2, 1>::type xzy, rbg, spt;
swizzler<0, 2, 2>::type xzz, rbb, spp;
swizzler<1, 0, 0>::type yxx, grr, tss;
swizzler<1, 0, 1>::type yxy, grg, tst;
swizzler<1, 0, 2>::type yxz, grb, tsp;
...
```

...EVEN MORE SWIZZLE!

```
    ...
    swizzler<2, 1, 2, 0>::type zyzx, bgbr, ptps;
    swizzler<2, 1, 2, 1>::type zyzy, bgbg, ptpt;
    swizzler<2, 1, 2, 2>::type zyzz, bgbb, ptpp;
    swizzler<2, 2, 0, 0>::type zzxx, bbrr, ppss;
    swizzler<2, 2, 0, 1>::type zzxy, bbrg, ppst;
    swizzler<2, 2, 0, 2>::type zzxz, bbrb, ppsp;
    swizzler<2, 2, 1, 0>::type zzyx, bbgr, ppts;
    swizzler<2, 2, 1, 1>::type zzyy, bbgg, pptt;
    swizzler<2, 2, 1, 2>::type zzyz, bbgb, pptp;
    swizzler<2, 2, 2, 0>::type zzzx, bbbr, ppps;
    swizzler<2, 2, 2, 1>::type zzzy, bbbg, pppt;
    swizzler<2, 2, 2, 2>::type zzzz, bbbb, pppp;
};
```

swizzler<> DESIGN

```
template<
    typename vector_type,
    typename scalar_type,
    size_t N,
    size_t... indices>
struct swizzler
{
    T data[N];
    // N might differ from vector_type::num_components
    // ex: .xxxx from vec2

(etc)
```

swizzler<> CONVERSIONS

`swizzler<>` CONVERSIONS

Needs to implicitly convert/assign to its `vector<>` equivalent

`swizzler<>` CONVERSIONS

Needs to implicitly convert/assign to its `vector<>` equivalent

```
operator vector_type()
{
    vector_type vec;
    assign_across(vec, 0, indices...);
    return vec;
}
```

swizzler<> CONVERSIONS

Needs to implicitly convert/assign to its vector<> equivalent

```
operator vector_type()
{
    vector_type vec;
    assign_across(vec, 0, indices...);
    return vec;
}
```

```
swizzler& operator=(const vector_type& vec)
{
    assign_across(vec, 0, indices...);
    return *this;
}
```

`swizzler<>` DESIGN (CONT.)

We use same fold expression trick

`swizzler<>` DESIGN (CONT.)

We use same fold expression trick

```
template<typename... Indices>
void assign_across(vector_type& vec, size_t i, Indices ...j) const
{
    ((vec[i++] = data[j]), ...);
}
```

`swizzler<>` DESIGN (CONT.)

We use same fold expression trick

```
template<typename... Indices>
void assign_across(vector_type& vec, size_t i, Indices ...j) const
{
    ((vec[i++] = data[j]), ...);
}
```

```
template<typename... Indices>
void assign_across(const vector_type& vec, size_t i, Indices ...j)
{
    ((data[j] = vec[i++]), ...);
}
```

swizzler<> PROBLEM

```
vec3 v = vec4(other.xy, other.zw);
```

```
> error: no matching function for call to [...]
> template argument deduction/substitution failed: [...]
```

swizzler<> PROBLEM

```
vec3 v = vec4(other.xy, other.zw);
```

```
> error: no matching function for call to [...]
> template argument deduction/substitution failed: [...]
```

- Solution? Introduce another abstraction layer!

swizzler<> PROBLEM

```
vec3 v = vec4(other.xy, other.zw);
```

```
> error: no matching function for call to [...]
> template argument deduction/substitution failed: [...]
```

- Solution? Introduce another abstraction layer!
- `(construct_at_index(i, decay(std::forward<Args>(args))), ...);`

swizzler<> PROBLEM

```
vec3 v = vec4(other.xy, other.zw);
```

```
> error: no matching function for call to [...]
> template argument deduction/substitution failed: [...]
```

- Solution? Introduce another abstraction layer!
- `(construct_at_index(i, decay(std::forward<Args>(args))), ...);`
- decay calls equivalent member function (or does nothing for scalar)

`swizzler<>` PROBLEM

```
vec3 v = vec4(other.xy, other.zw);
```

```
> error: no matching function for call to [...]
> template argument deduction/substitution failed: [...]
```

- Solution? Introduce another abstraction layer!
- `(construct_at_index(i, decay(std::forward<Args>(args))), ...);`
- `decay` calls equivalent member function (or does nothing for scalar)
- both `vector` and `swizzler` have one so they can interchange easily

OPERATORS AND FUNCTIONS

We will need to re-create a lot of generic utility functions

We will need to re-create a lot of generic utility functions

Example: the dot (inner) product of two vectors

We will need to re-create a lot of generic utility functions

Example: the dot (inner) product of two vectors

```
template<typename T, size_t... Ns>
T dot(const vector<T, Ns...> &, const vector<T, Ns...> &);
```

```
float n = dot(vec3(1, 0, 0), vec3(0, 0, 1));
```

We immediately hit a big problem!

```
vec3 v = vec3(1, 0, 0);
float n = dot(v.xzx, v.zyx);
```

```
> 'dot': no matching overloaded function found
> could not deduce template argument
```

Type deduction does not consider implicit conversions!

Type deduction does not consider implicit conversions!

- Possible fixes:

Type deduction does not consider implicit conversions!

- Possible fixes:
- ```
  float n = dot<float, 0, 1, 2>(v.xzx, v.zyx);
  ```
  ```

Type deduction does not consider implicit conversions!

- Possible fixes:
- ```
  float n = dot<float, 0, 1, 2>(v.xzx, v.zyx);
  ```
- create by hand all scalar/size combinations :(
  ```

Type deduction does not consider implicit conversions!

- Possible fixes:
- ```
  float n = dot<float, 0, 1, 2>(v.xzx, v.zyx);
  ```
- create by hand all scalar/size combinations :(
- SFINAE tricks
  ```

A BETTER FIX

we place the functions in a non-deduced context: inside `vector<>` itself!

A BETTER FIX

- we place the functions in a non-deduced context: inside `vector<>` itself!

```
template<typename T, size_t... Ns>
struct vector
{

    friend T dot(const vector& a, const vector& b)
    {
        /* inline friend found via ADL */
    }

(etc)
```

ARE WE DONE?

 No!

ARE WE DONE?

 No!

Lots of shader code uses scalar types only:

```
float opS(float d1, float d2)
{
    return max(-d2, d1);
}
```

ARE WE DONE?

 No!

Lots of shader code uses scalar types only:

```
float opS(float d1, float d2)
{
    return max(-d2, d1);
}
```

We only provide the vector variant

```
friend vector max(const vector& a, const vector& b)
{
    return vector((a.data[Ns] < b.data[Ns] ? a.data[Ns] : b.data[Ns])
}
```

MORE TROUBLE

MORE TROUBLE

Ambiguity with literals:

```
smoothstep(0, 1, v.xyz);
//           ^
//           could be `int`, `float` or `double`
```

MORE TROUBLE

Ambiguity with literals:

```
smoothstep(0, 1, v.xyz);
//           ^
//           could be `int`, `float` or `double`
```

```
friend vector smoothstep(scalar_type, scalar_type, const vector&);
//       ^
//       only templated on vector
```

SOLUTION

Inspect the list of function args and deduct a vector type using `std::common_type` techniques

SOLUTION

Inspect the list of function args and deduct a vector type using `std::common_type` techniques

First: need to make `vec1` convert to/from scalars

```
std::is_convertible<vec1, float>::value == true
```

SOLUTION

Inspect the list of function args and deduct a vector type using `std::common_type` techniques

First: need to make `vec1` convert to/from scalars

```
std::is_convertible<vec1, float>::value == true
```

Then: provide custom type traits

```
promote_to_vec< float                    >::type == vec1
```

SOLUTION

Inspect the list of function args and deduct a vector type using `std::common_type` techniques

First: need to make `vec1` convert to/from scalars

```
std::is_convertible<vec1, float>::value == true
```

Then: provide custom type traits

```
promote_to_vec< float                    >::type == vec1
```

```
promote_to_vec< vec3                     >::type == vec3
promote_to_vec< decltype(vec3().xyz) >::type == vec3
```

SOLUTION

Inspect the list of function args and deduct a vector type using `std::common_type` techniques

First: need to make `vec1` convert to/from scalars

```
std::is_convertible<vec1, float>::value == true
```

Then: provide custom type traits

```
promote_to_vec< float                    >::type == vec1
```

```
promote_to_vec< vec3                     >::type == vec3
promote_to_vec< decltype(vec3().xyz) >::type == vec3
```

```
promote_to_vec< vec3, float          >::type == vec3
promote_to_vec< vec3, float, double  >::type == vec3
```

SOLUTION (CONT.)

SOLUTION (CONT.)

Insted of `friend` functions are `static`

```
template<template<class, size_t...> class vector, class T, size_t...
struct builtin_func_lib
{
    static vector max(const vector& a, const vector& b)
(etc)
```

SOLUTION (CONT.)

Insted of `friend` functions are `static`

```
template<template<class, size_t...> class vector, class T, size_t...
struct builtin_func_lib
{
    static vector max(const vector& a, const vector& b)
(etc)
```

Create an all-forwarding monster function

```
template<class... Args>
inline auto func(Args&&... args) ->
    decltype(decay(
        promote_to_vec<Args...>::type::
        func(std::forward<Args>(args)...)))
{
    return
        promote_to_vec<Args...>::type::
        func(std::forward<Args>(args)...);
}
```

SOLUTION - COMPROMISE

```
...
MAKE_LIB_FUNC(abs)
MAKE_LIB_FUNC(sign)
MAKE_LIB_FUNC(floor)
MAKE_LIB_FUNC(trunc)
MAKE_LIB_FUNC(ceil)
MAKE_LIB_FUNC(fract)
MAKE_LIB_FUNC(mod)
MAKE_LIB_FUNC(min)
MAKE_LIB_FUNC(max)
MAKE_LIB_FUNC(clamp)
MAKE_LIB_FUNC(mix)
MAKE_LIB_FUNC(step)
MAKE_LIB_FUNC(smoothstep)
...
```

SOLUTION - DEMO

```
return max(-d2, d1);
//             ||
//             ||
//             \/
    promote_to_vec<float, float>
//             ||
//             ||
//             \/
return vector<float, 0>::max(-d2, d1);
```

THE `matrix<>` DATATYPE

Now that we have `vector<>` a matrix is more straightforward

```
template<
    typename,
    template<typename, size_t...> class vector_type,
    typename...
>
struct matrix;
```

FOLDING HELPER

two dimensional → introduce helper type for the indices

```
template<size_t...>
struct indices_pack;
```

THE `matrix<>` DATATYPE (CONT.)

THE `matrix<>` DATATYPE (CONT.)

```
template<
    typename scalar_type,
    template<typename, size_t...> class vector_type,
    size_t... Columns,
    size_t... Rows
>
struct matrix<scalar_type, vector_type,
    indices_pack<Columns...>, indices_pack<Rows...>>
```

THE matrix<> DATATYPE (CONT.)

```
template<
    typename scalar_type,
    template<typename, size_t...> class vector_type,
    size_t... Columns,
    size_t... Rows
>
struct matrix<scalar_type, vector_type,
    indices_pack<Columns...>, indices_pack<Rows...>>
```

```
{
    static constexpr auto N = sizeof...(Columns);
    static constexpr auto M = sizeof...(Rows);

    using column_type = vector_type<scalar_type, Columns...>;
    using row_type = vector_type<scalar_type, Rows...>;

    column_type data[M];
(etc)
```

THE `matrix<>` DECLARATION

```
using vec2 = vector<float, 0, 1>;
using vec3 = vector<float, 0, 1, 2>;

using mat2 = matrix<float, vector,
    indices_pack<0, 1>, indices_pack<0, 1>>;

using mat3 = matrix<float, vector,
    indices_pack<0, 1, 2>, indices_pack<0, 1, 2>>;

using mat2x3 = matrix<float, vector,
    indices_pack<0, 1>, indices_pack<0, 1, 2>>;
```

THE `matrix<>` CONSTRUCTORS

THE `matrix<>` CONSTRUCTORS

```
matrix() = default; // zeroes all data
```

THE `matrix<>` CONSTRUCTORS

```
matrix() = default; // zeroes all data
```

```
explicit matrix(scalar_type s) // fill in diagonally
{
    ((data[Rows][Rows] = s), ...);
}
```

THE matrix<> CONSTRUCTORS

```
matrix() = default; // zeroes all data
```

```
explicit matrix(scalar_type s) // fill in diagonally
{
    ((data[Rows][Rows] = s), ...);
}
```

```
template<typename... Args>
explicit matrix(Args&&... args)
{
    size_t i = 0;
    (construct_at_index(i,
        decay(std::forward<Args>(args))), ...);
}
```

THE `matrix<>` OPS AND FUNCS

- can recycle the same binary operators as vector if written generic
- except *multiplication*
  - which needs to be handled differently
  - for all variations of `matrix`, `row_type`, `column_type`

PRIOR ART

 not invented here :)

- clang vector extensions
  - `typedef float vec3 __attribute__((ext_vector_type(3)));`
  - PRO: supports full swizzling
  - CON: very limited in initializations

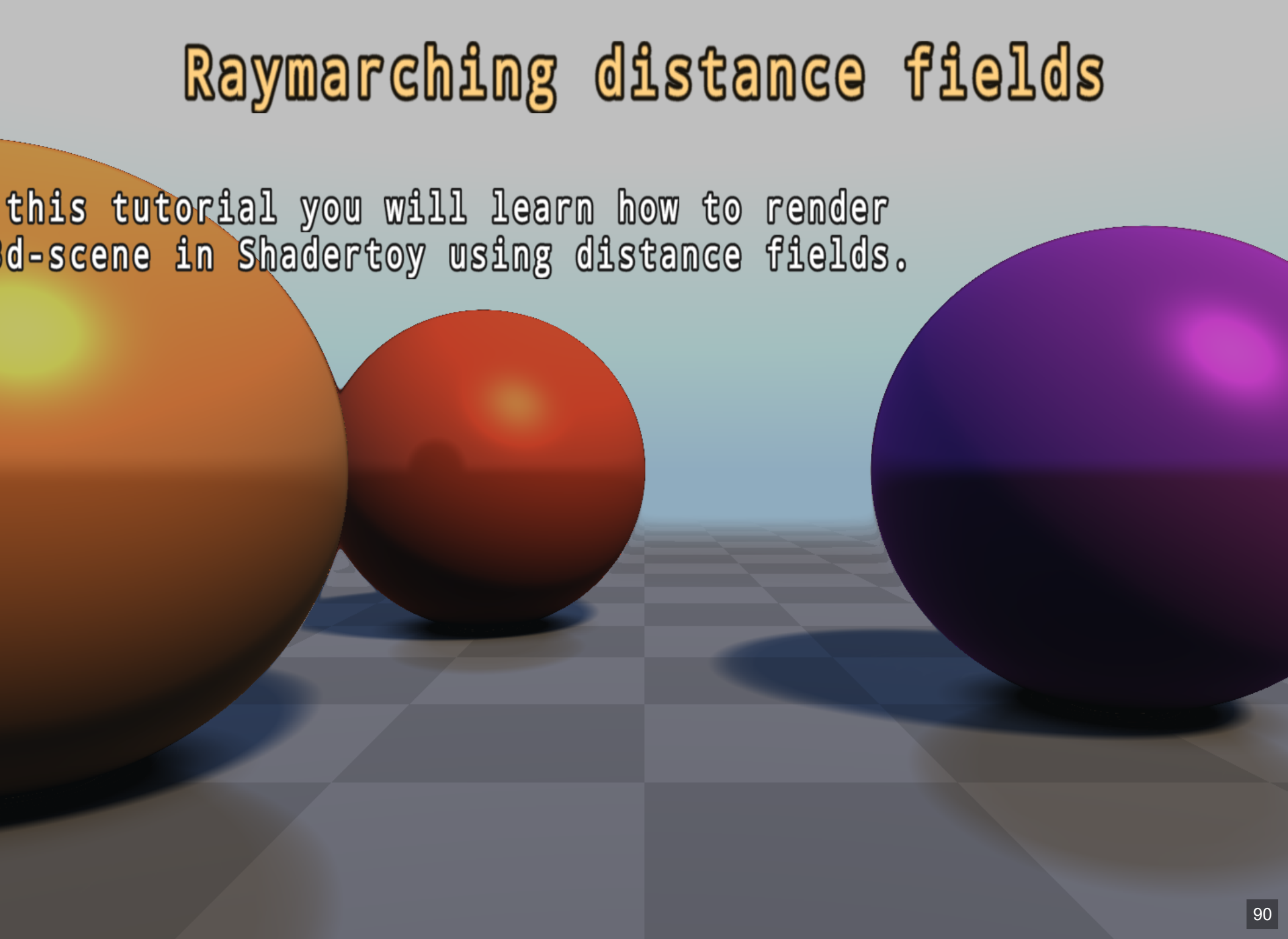
- 3rd party libraries
  - GLM
    - `.xyz()` style only
    - horrible preprocessor heavy implementation
  - CXXSwizzle
    - full spec
    - slow debug

RESULTS

…but first: Crash Course into Procedural Graphics!

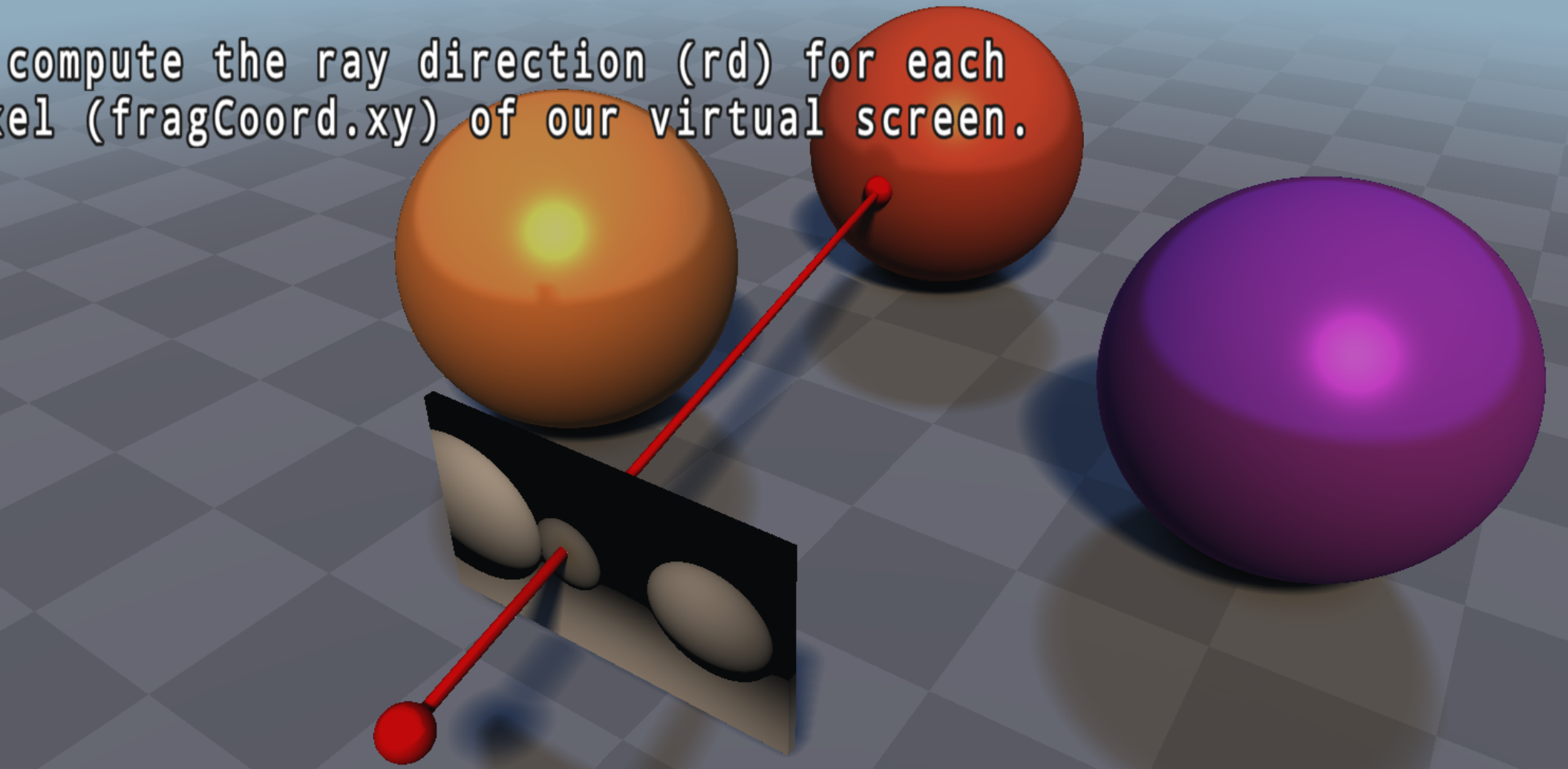
- Courtesy of @ReinderNijhoff
  https://www.shadertoy.com/view/4dSfRc

Raymarching distance fields



Create a ray

compute the ray direction (rd) for each
xel (fragCoord.xy) of our virtual screen.



Distance fields

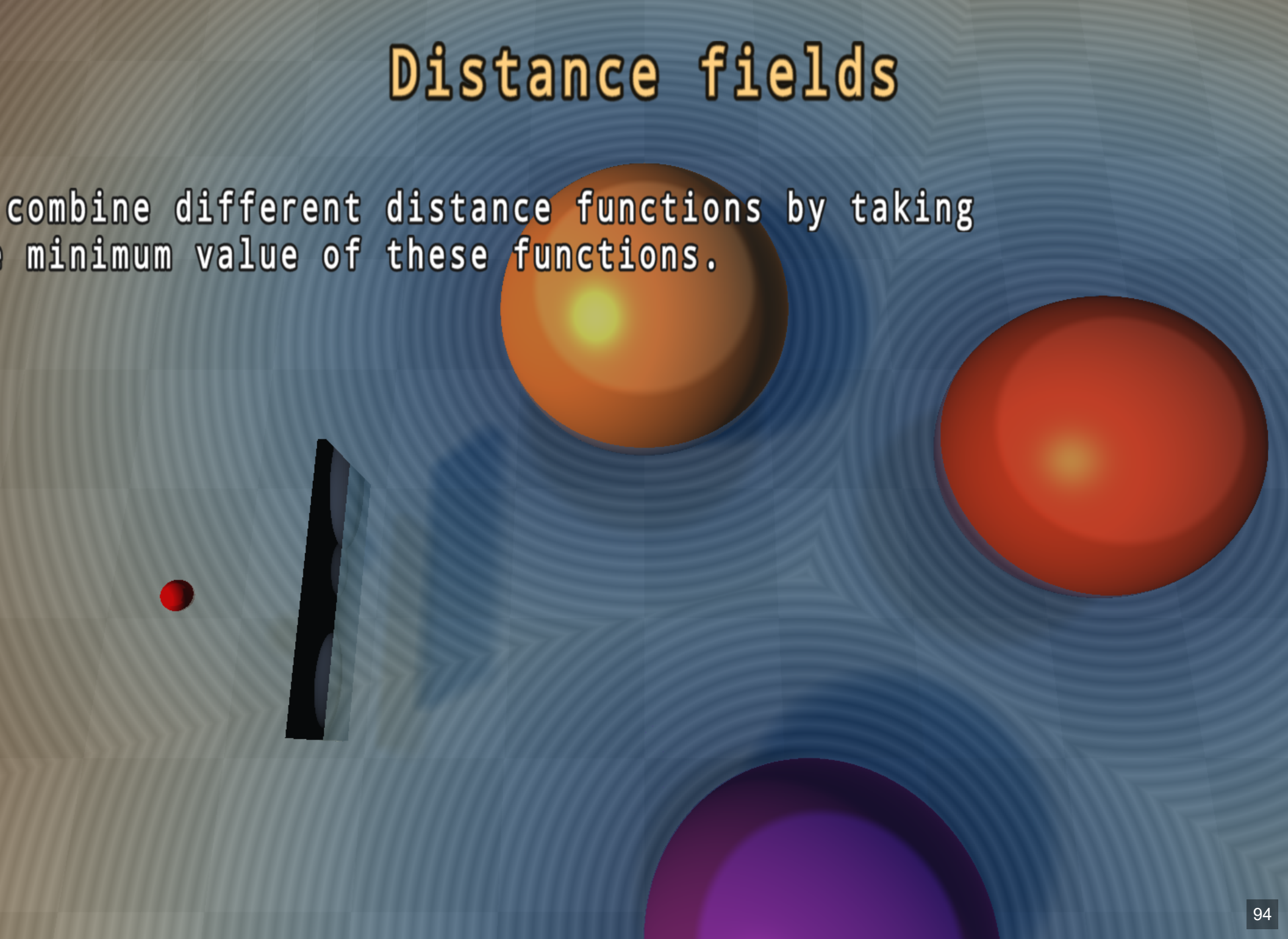
istance field is used to find the
ersection of our ray (ro, rd) and the spheres
plane of the scene.

Distance fields

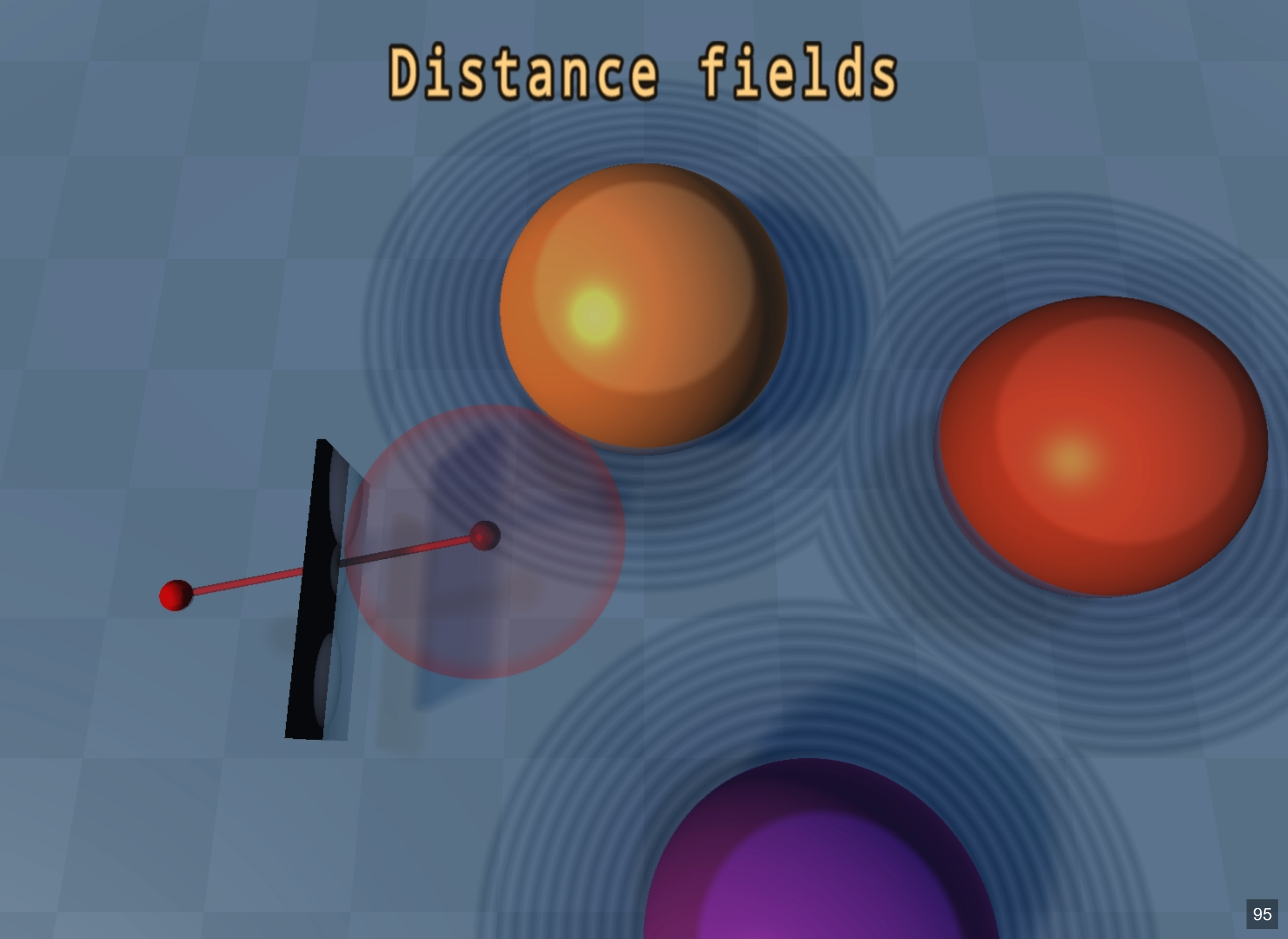
e distance function for a sphere is the distance to
e center of the sphere minus the radius of the sphere.

Distance fields

combine different distance functions by taking
minimum value of these functions.



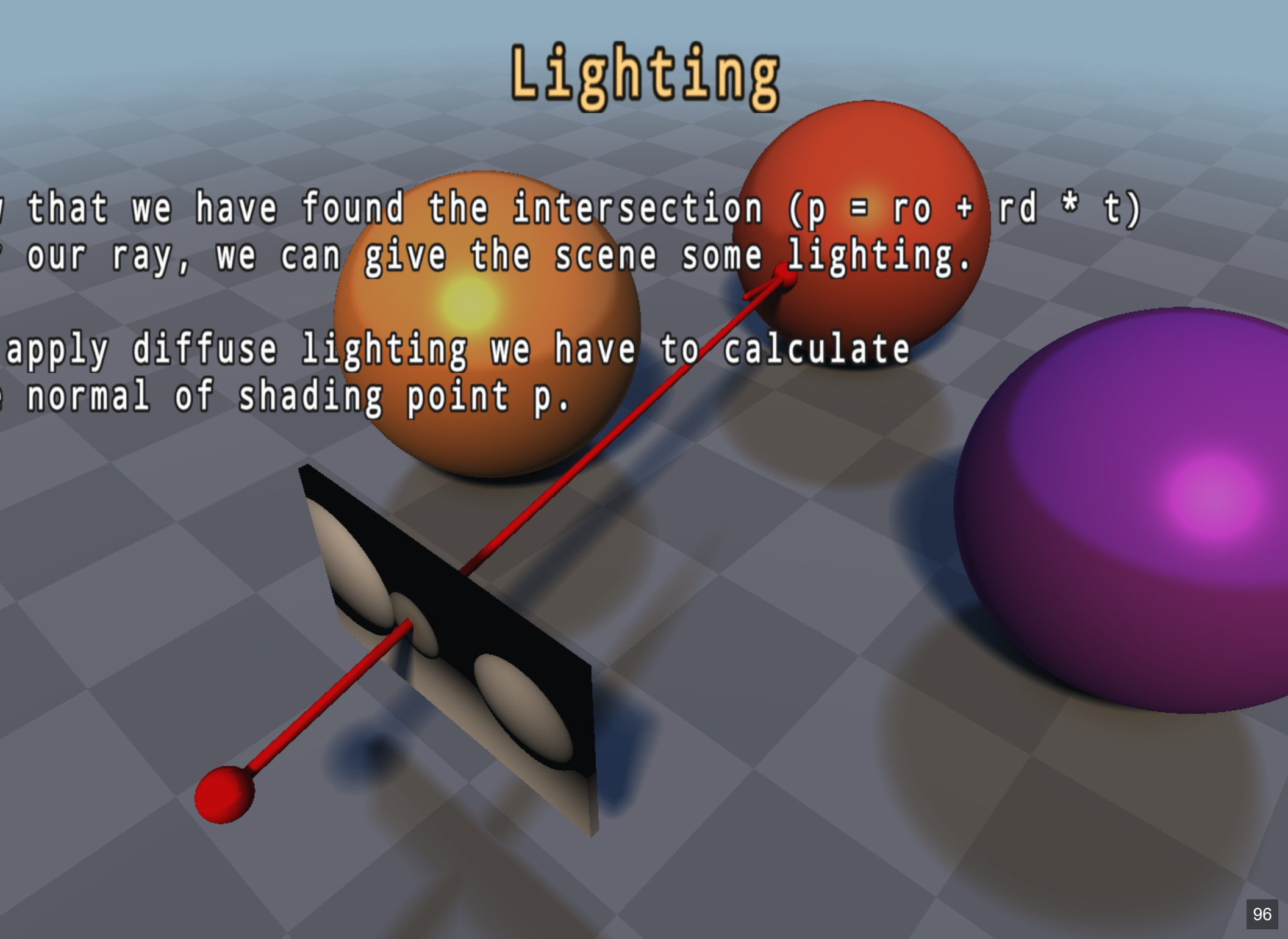
Distance fields



Lighting

that we have found the intersection (p = ro + rd * t)
our ray, we can give the scene some lighting.

apply diffuse lighting we have to calculate
normal of shading point p.



Lighting

e normal can be calculated by taking the central
fferences on the distance field:

```
3 calcNormal(in vec3 p) {
  vec2 e = vec2(1.0, -1.0) * 0.0005;
  return normalize(
        e.xyy * map(p + e.xyy) +
        e.yyx * map(p + e.yyx) +
        e.yxy * map(p + e.yxy) +
        e.xxx * map(p + e.xxx));
```

SHOWCASE

GPU / DESKTOP PC

- https://www.shadertoy.com/user/valentingalea
- Nvidia GeForce 1060
- 1080p

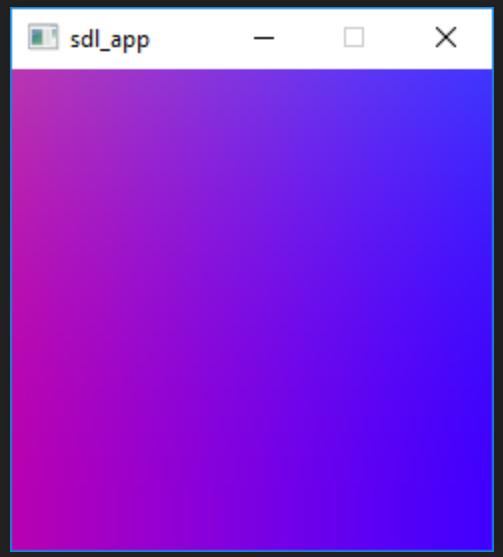
CPU / DESKTOP PC

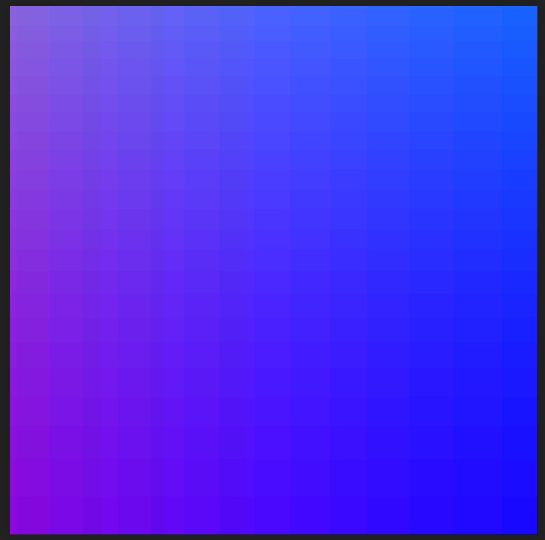
- minimal draw app with https://www.libsdl.org/
- AMD FX 8350 8-core 4.00 GHz
- Microsoft Visual C++ 2017
  - `/O2 /Ob2 /fp:fast /fp:except-`

CPU / MOBILE PHONE

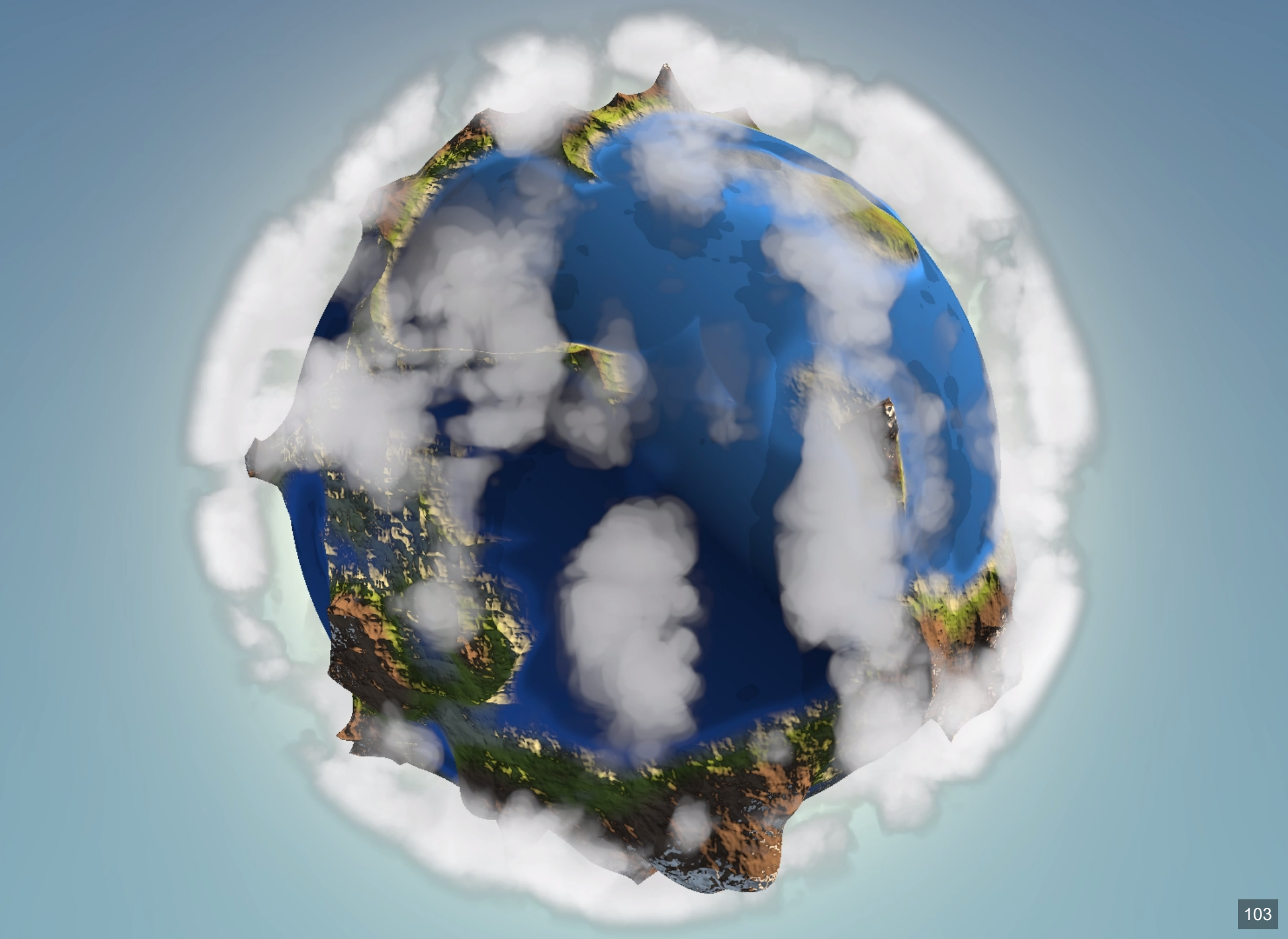
- C4Droid app (https://play.google.com/store/apps/details?id=com.n0n3m4.droidc)
- Samsung Galaxy S7
- GCC 8.0
  - `-Ofast -march=native -funroll-loops`

HELLO WORLD (CPU)

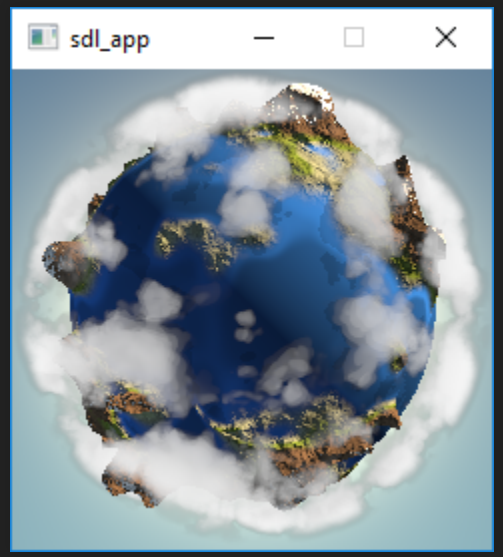




240x240 px	85.62 FPS	166.77 FPS
120x120 px	100.27 FPS	468.49 FPS

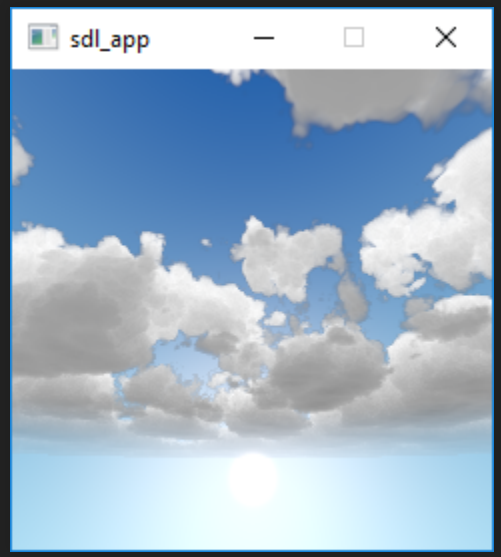


PLANET (CPU)



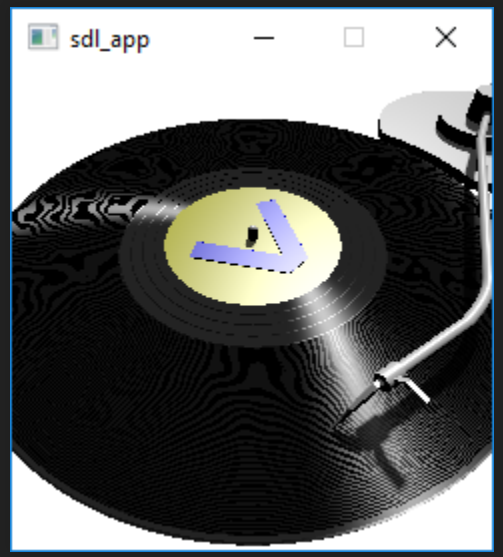
240x240 px	1.92 FPS	0.83 FPS
120x120 px	7.30 FPS	3.34 FPS

CLOUDS (CPU)

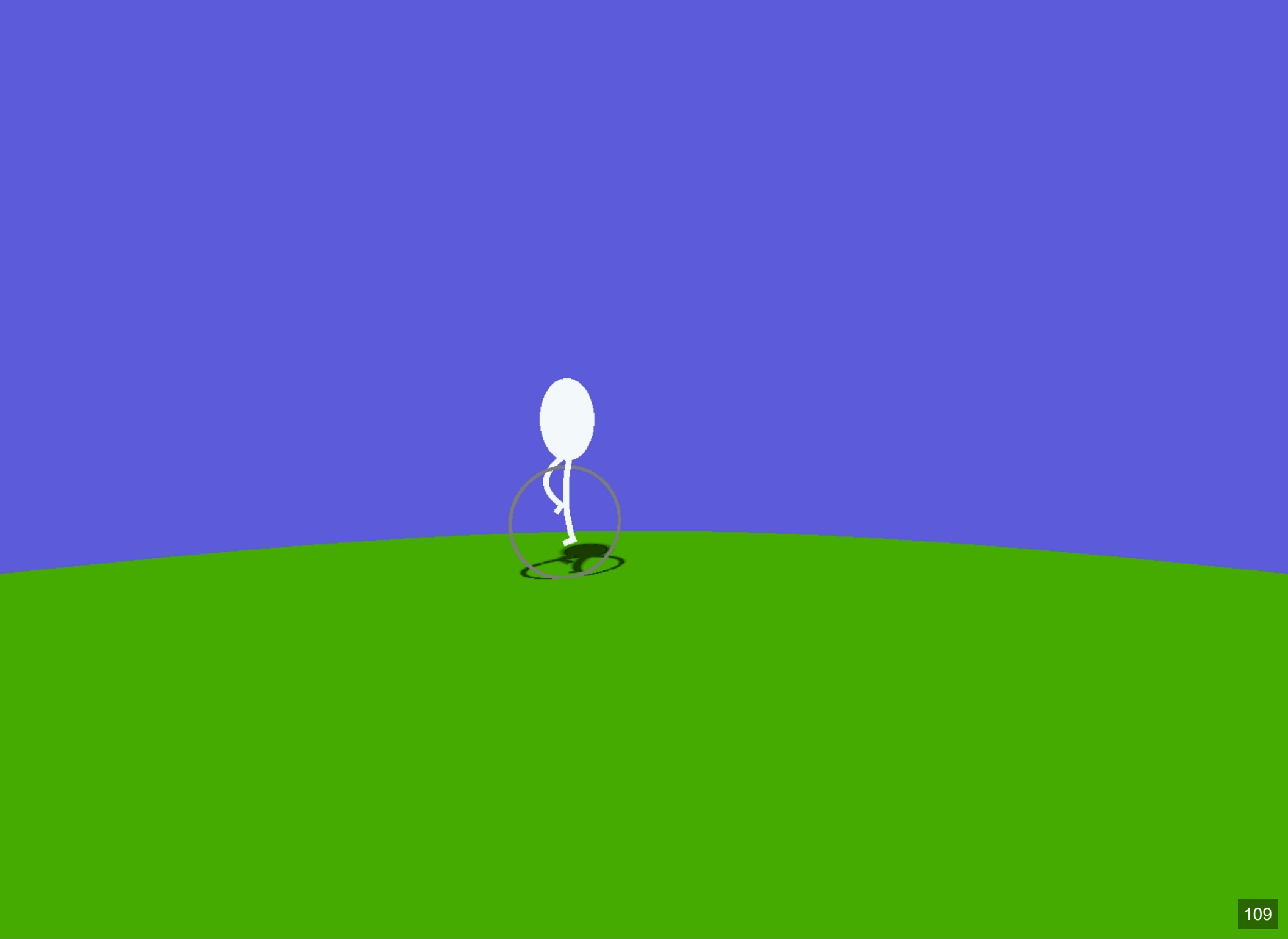


240x240 px	2.54 FPS	2.44 FPS
120x120 px	9.63 FPS	9.64 FPS

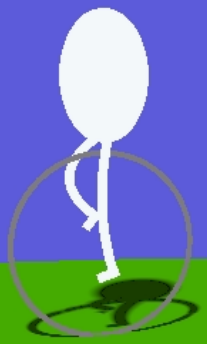
VINYL TURNTABLE (CPU)



240x240 px	8.44 FPS	2.94 FPS
120x120 px	28.11 FPS	12.82 FPS

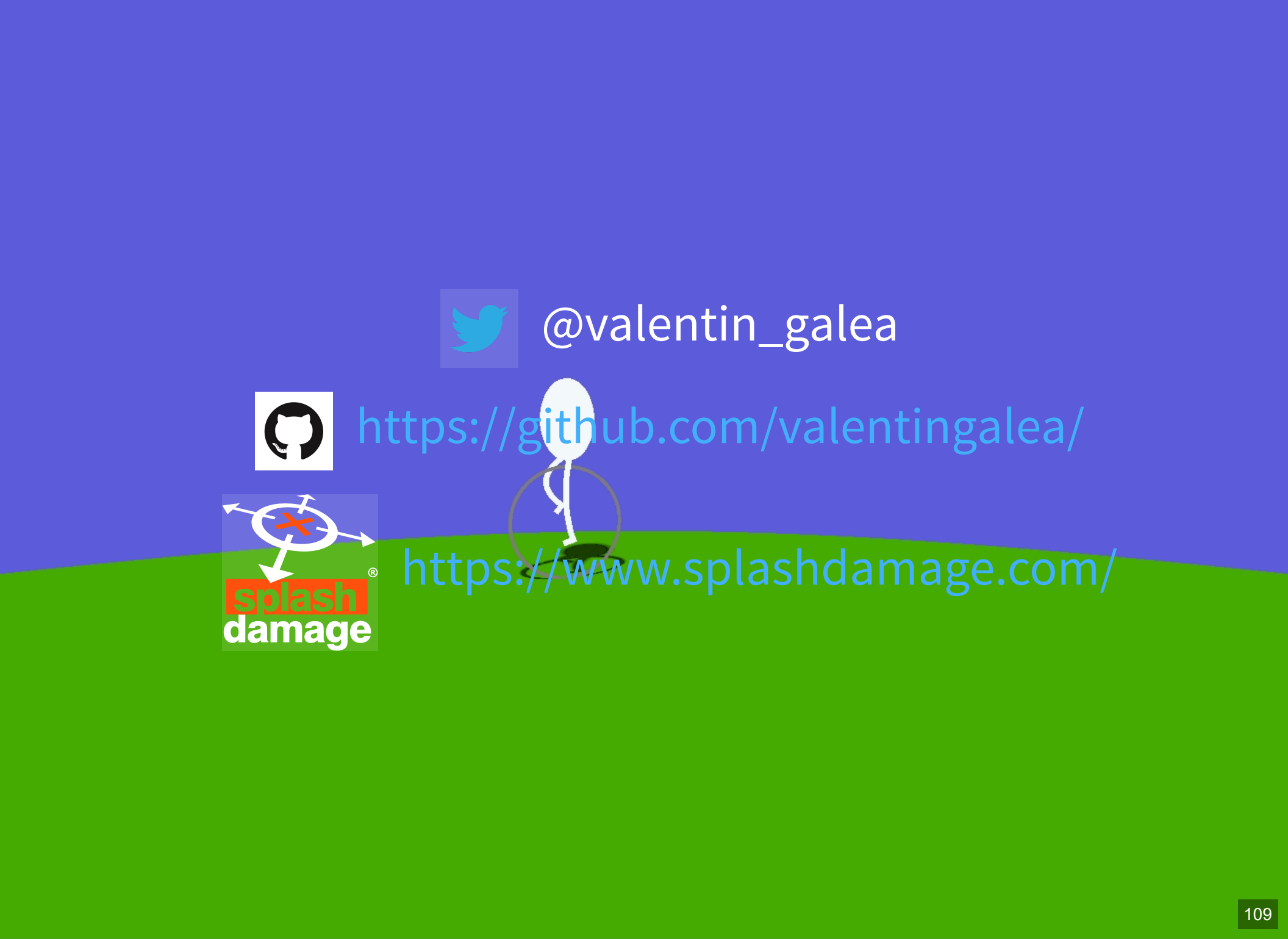


@valentin_galea



@valentin_galea

https://github.com/valentingalea/


@valentin_galea
https://github.com/valentingalea/
splash
damage
https://www.splashdamage.com/

ATTRIBUTION

Piotr Gwiazdowski @gwiazdorrr for original inspiration and help
Shading and Renderman: Jaume Sanchez | @thespite
Motivation Shaders: Inigo Quilez
https://www.shadertoy.com/view/ld3Gz2
https://www.shadertoy.com/view/ldScDh
https://www.shadertoy.com/view/4ttSWf
GPU pipeline: https://open.gl/ (CC BY-SA 4.0)
All other images under "Fair Use"/"Fair Dealing"