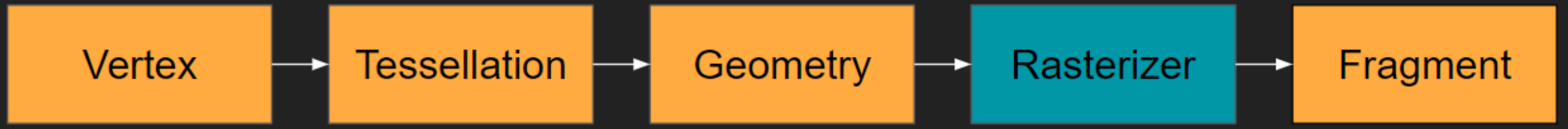
FSHADE

WHY FUNCTION COMPOSITION MATTERS

Georg Haaser

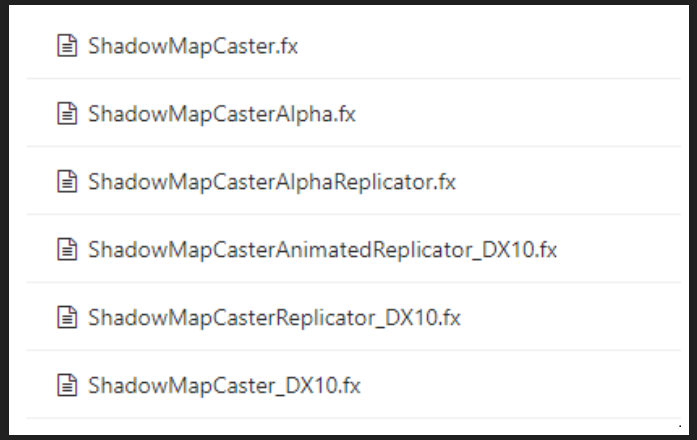
WHAT ARE SHADERS?



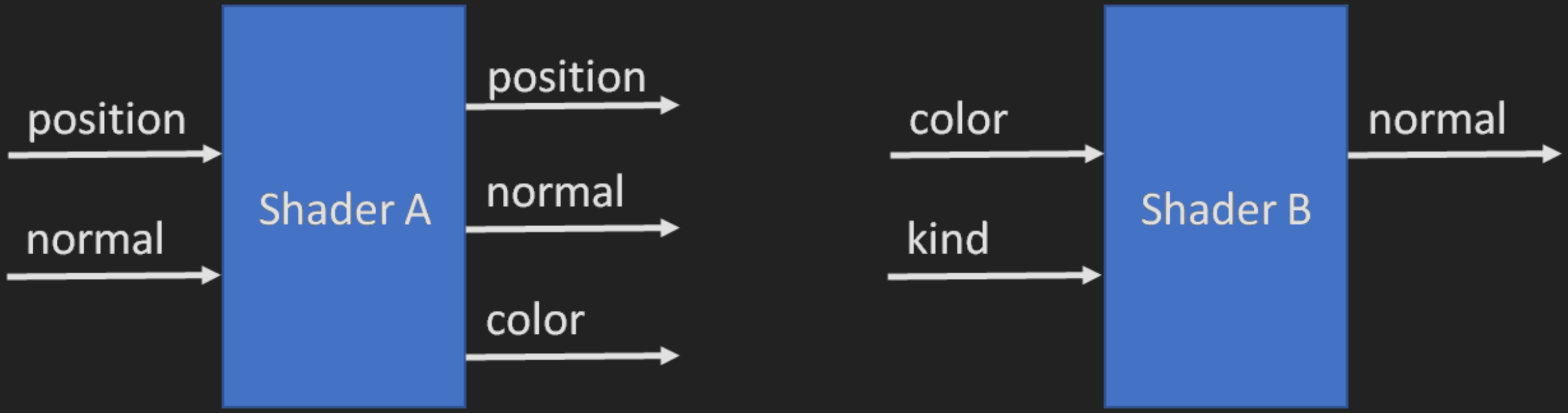
- pure multi-valued-functions
- associated to a hardware stage
- inputs
  - frequency: attributes/uniforms
  - kind:
    vertex/primitive/fragment

WHY FSHADE?

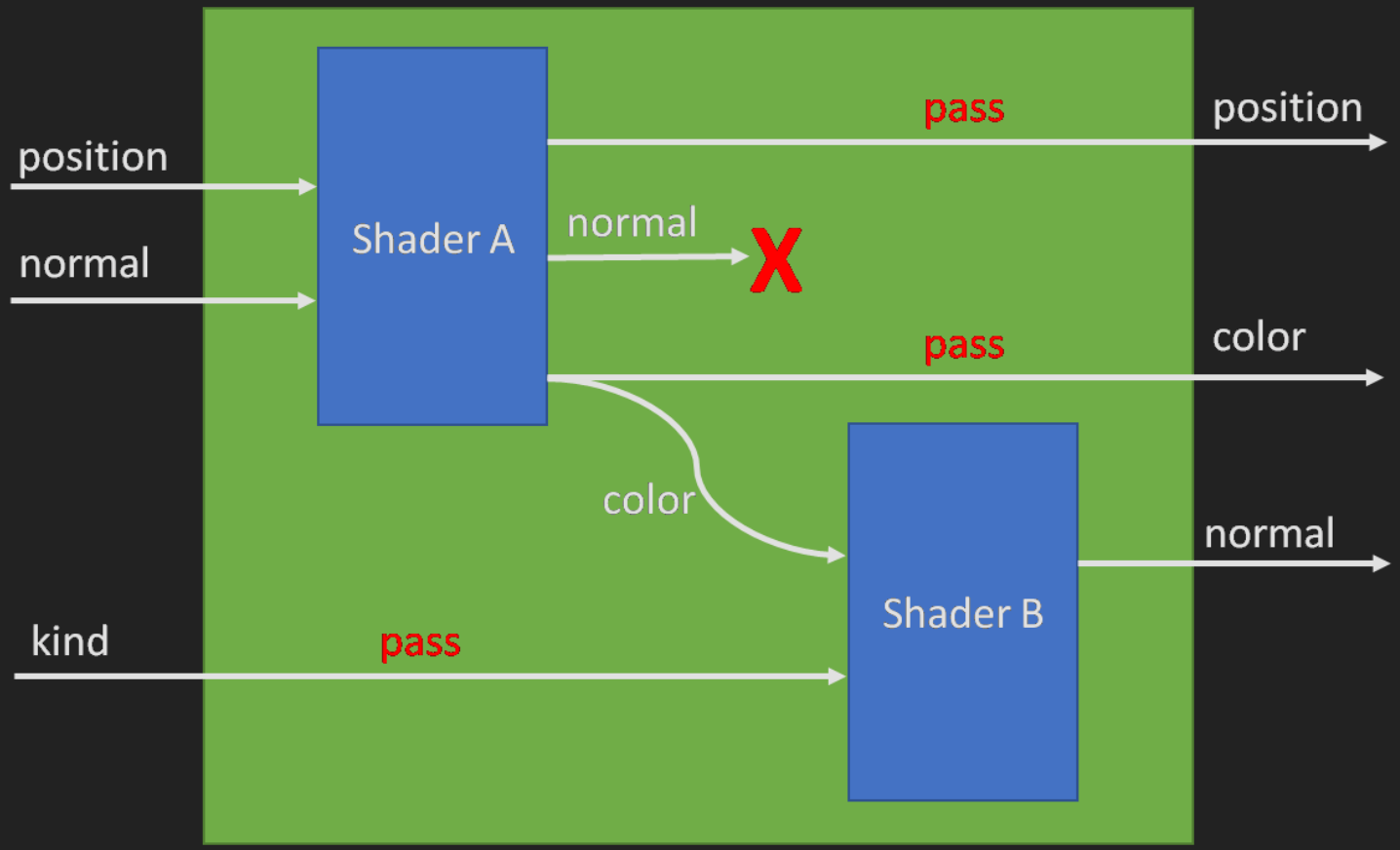
- combinatorial explosion
- programmatic manipulation
- multiple target languages



SHADER MODULES



COMPOSITION



Shader Composition: Example

Transform

```
vec4 pp = MVPMatrix * InPosition;

OutPosition = pp;
OutNormal = NMatrix * InNormal;
OutTexCoord = InTexCoord;
```

Offset

```
vec4 p = InPosition;
vec4 n = InNormal;

OutPosition = p + 0.5 * n;
OutNormal = InNormal;
```

Shader Composition: Example

Compose [Transform; Offset]

```
vec4 pp = MVPMatrix * InPosition;

OutPosition = pp;
OutNormal = NMatrix * InNormal;
OutTexCoord = InTexCoord;

vec4 p = InPosition;
vec4 n = InNormal;

OutPosition = p + 0.5 * n;
OutNormal = InNormal;
```

Shader Composition: Example

Compose [Transform; Offset]

```
vec4 pp = MVPMatrix * InPosition;

vec4 PC = pp;
vec3 NC = NMatrix * InNormal
vec2 TCC = InTexCoord

vec4 p = PC;
vec3 n = NC;

OutPosition = p + 0.5 * n;
OutNormal = NC;
OutTexCoord = TCC;
```

```
vec4 pp= MVPMatrix * InPosition;

OutPosition = pp;
OutNormal = NMatrix * InNormal;
OutTexCoord = InTexCoord;

vec4 p = InPosition;
vec4 n = InNormal;

OutPosition = p + 0.5 * n;
OutNormal = InNormal;
```

IMPLEMENTATION

- F# quotations provide typed syntax tree (TAST)
- embedded in Aardvark
- Extends to other languages
  - Template Haskell
  - Clojure
  - C++ via templates (if you're brave enough)
- standalone parser

LIGHTING

```
type Vertex =
    {
        [<Position>]      pos : V4d
        [<Normal>]        n : V3d
        [<LightDir>]      l : V3d
        [<CamDir>]        c : V3d
        [<Color>]         color : V4d
        [<SpecularColor>] spec : V4d
    }

let lighting (v : Vertex) =
    fragment {
        let n = Vec.normalize v.n
        let l = Vec.normalize v.l
        let c = Vec.normalize v.c
        let diffuse = Vec.dot n l |> clamp 0.0 1.0
        let spec = Vec.dot (Vec.reflect l n) (-c) |> clamp 0.0 1.0
        let specc = v.spec.XYZ
        return v.color.XYZ * diffuse + specc * pow spec uniform.Shininess
    }
```

VERTEX TRANSFORMATION

```
let transform (v : Vertex) =
    vertex {
        let light = uniform.LightLocation
        let wp = uniform.ModelMatrix * v.pos.XYZ

        return {
            pos = uniform.ModelViewProjMatrix * v.pos
            n = uniform.ModelViewMatrixInv * v.n
            b = uniform.ModelViewMatrix * v.b
            t = uniform.ModelViewMatrix * v.t
            tc = v.tc
            l = uniform.ViewMatrix * (light - wp)
            c = -uniform.ViewMatrix * wp
            color = uniform.DiffuseColor
            spec = uniform.SpecularColor
        }

    }
```

transform

skinning

texture

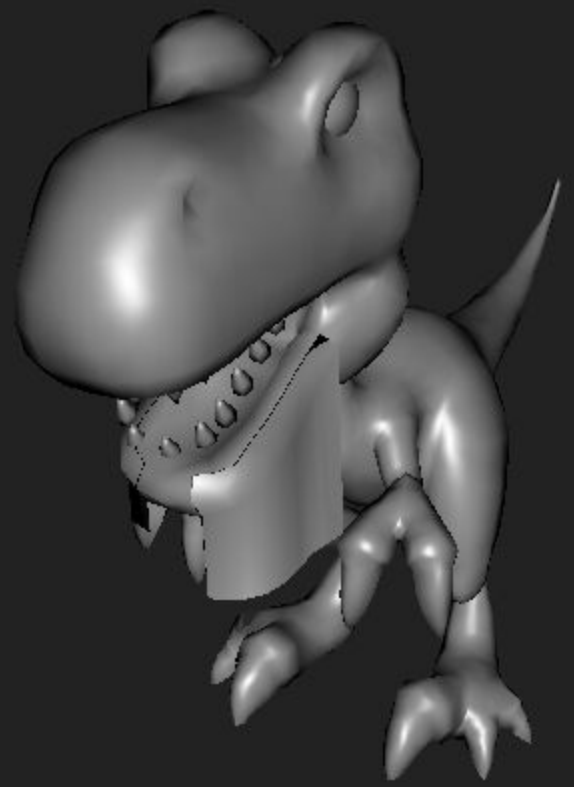
alpha-test

specular

normal

lighting

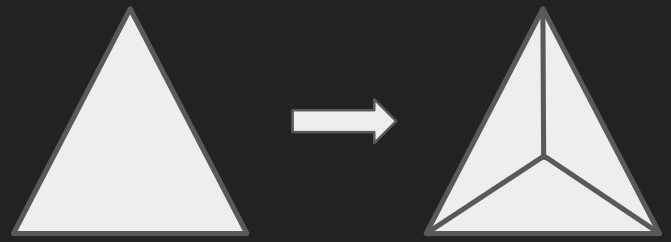
animation none



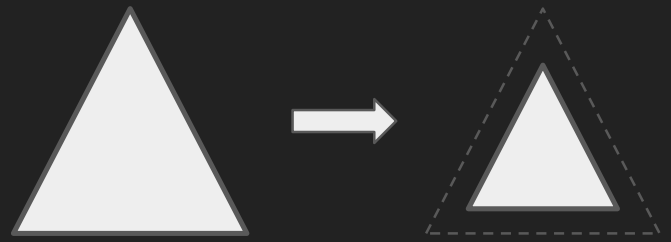
```
};
layout(std140, binding = 2)
uniform PerModel
{
    mat4x4 ModelTrafo;
    mat4x4 ModelViewProjTrafo;
    mat4x4 ModelViewTrafoInv;
};
layout(std140, binding = 3)
uniform PerView
{
    mat4x4 ViewTrafo;
};
#ifdef Vertex
layout(location = 0) in vec3 Normals;
layout(location = 1) in vec4 Positions;
layout(location = 0) out vec3 fs_CameraDirection;
layout(location = 1) out vec4 fs_Colors;
layout(location = 2) out vec3 fs_LightDirection;
layout(location = 3) out vec3 fs_Normals;
layout(location = 4) out vec4 fs_SpecularColor;
void main()
{
    vec3 wp = (vec4(Positions.xyz, 1.0) * ModelTrafo).xyz;
    fs_CameraDirection = (-(vec4(wp, 1.0) * ViewTrafo).xyz);
    fs_Colors = DiffuseColor;
    fs_LightDirection = (vec4((LightLocation - wp), 0.0) * V
    fs_Normals = (ModelViewTrafoInv * vec4(Normals, 0.0)).xy
    gl_Position = (Positions * ModelViewProjTrafo);
    fs_SpecularColor = SpecularColor;
}
#endif
#ifdef Fragment
layout(location = 0) in vec3 fs_CameraDirection;
layout(location = 1) in vec4 fs_Colors;
layout(location = 2) in vec3 fs_LightDirection;
layout(location = 3) in vec3 fs_Normals;
layout(location = 4) in vec4 fs_SpecularColor;
layout(location = 0) out vec4 ColorsOut;
void main()
{
    vec3 n = normalize(fs_Normals);
    vec3 l = normalize(fs_LightDirection);
    vec3 c = normalize(fs_CameraDirection);
    float diffuse = clamp(dot(n, l), 0.0, 1.0);
    float spec = clamp(dot(reflect(l, n), (-c)), 0.0, 1.0);
    vec3 specc = fs_SpecularColor.xyz;
    vec3 color = ((fs_Colors.xyz * diffuse) + (specc * pow(s
    ColorsOut = vec4(color, fs_Colors.w);
}
```

Geometry Composition

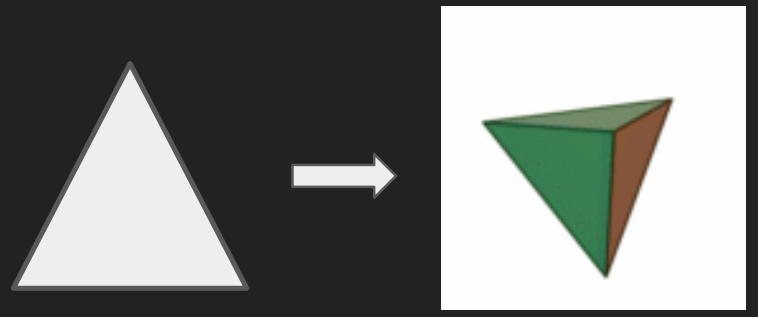
Divide



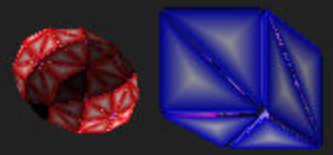
Shrink



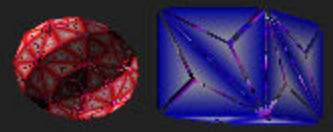
Extrude



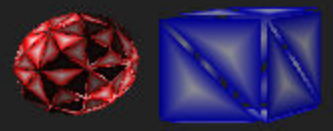
[ext, shr, div]



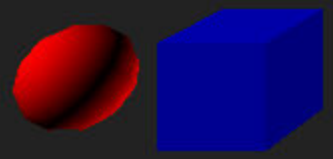
[div, shr, ext]



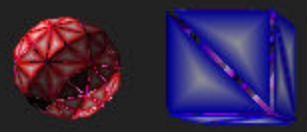
[shr, div]



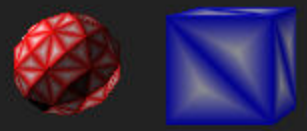
empty



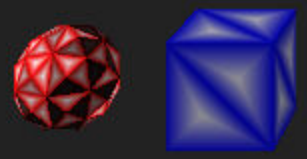
[shr, div, ext]



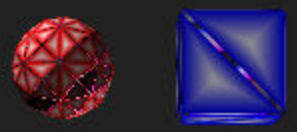
[div, ext]



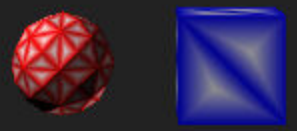
[div]



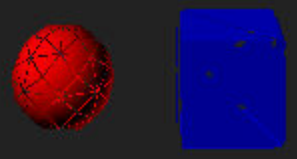
[shr, ext, div]



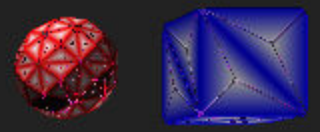
[ext, div]



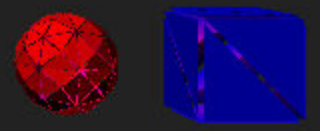
[shr]



[div, ext, shr]

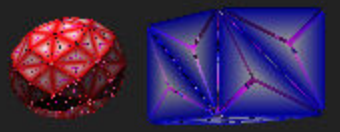


[shr, ext]

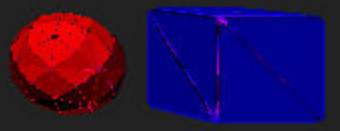


[ext]

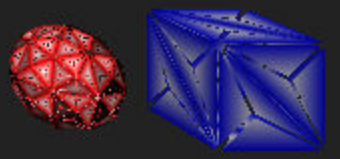
[ext, div, shr]



[ext, shr]



[div, shr]



ADVANCED TECHNIQUES

- instancing
- single pass stereo
- platform adjustments (depth range, etc.)
- specialization
- unification
- many more...

LIMITATIONS

- lambda functions
- recursive functions and types
- dynamic allocation
- OOP constructs (references, subtyping, etc.)

QUESTIONS?

github.com/krauthaufen/FShade
fshade.org
Thanks to Manuel Wieser for the Eigi Model,www.manuelwieser.com