Inspecting Block Closures

To Generate Shaders for GPU Execution

Ronie Salgado
ISCLab, DCC, University of Chile

Talk Outline

- Motivation.
- GPU Hardware and Programming Model.
- Smalltalk -> Shader: Code Generation Pipeline.
- Case Studies:
  - Procedural Texture Generation.
  - Particle Simulation and Rendering.
- Future Work.

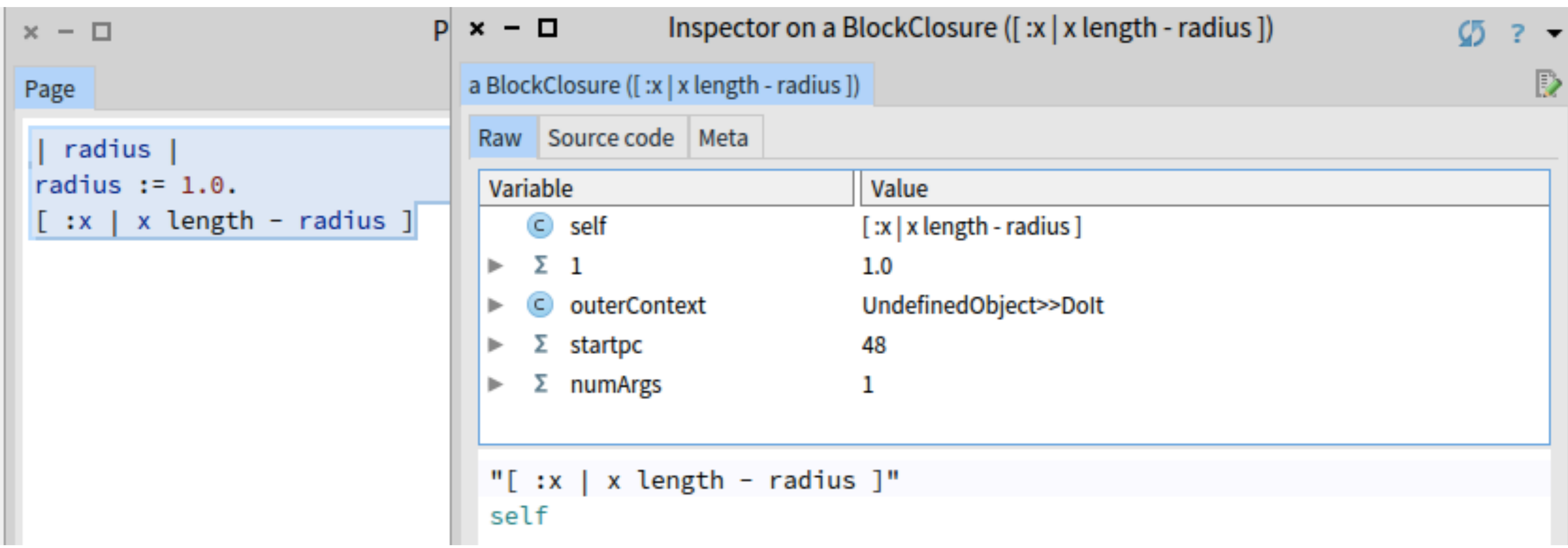
Motivation

- High performance for parallel task.
- Facilitate shader debugging.
- Smalltalk on the GPU.
- Reduce impedance mismatch between CPU <-> GPU.

Why BlockClosure?

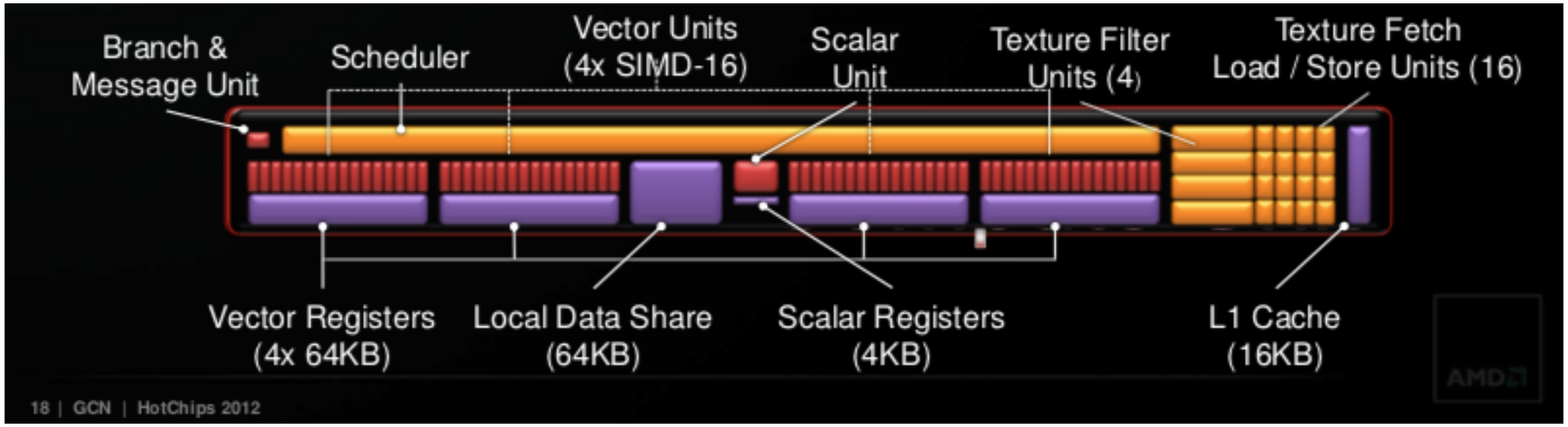
- Closures look like functions: f := [:x | …]
- They encapsulate **Code** and **Data**.
- Easy to use for **scripting** in a **Playground**.
- **Map**-Reduce style computations.

Why BlockClosure?



GPU Hardware

- In the case of the AMD GCN architecture.
- Multiple independent Compute Unit.
    - Scalar ALU (Control Flow)
    - Vectorial ALU (SIMD, Data Processing)



Mike Mantor. 2012. AMD Radeon™ HD 7970 with graphics core next (GCN) architecture. In 2012 IEEE Hot Chips 24 Symposium (HCS). IEEE

Programming Environment Constraints

- No Dynamic Lookup.
- No Arithmetic Traps (e.g SmallInteger -> LargeInteger).
- No Dynamic Memory Allocation inside the GPU (No objects or GC).

GPU Programming Model

- Graphics pipeline (OpenGL, Metal, Vulkan, D3D):
  - Vertex Shader.
  - … Rasterization …
  - Fragment Shader.
- Compute pipeline (Graphics APIs, CUDA, OpenCL):
  - Compute Shader.

Smalltalk -> Shader Pipeline "Parsing"

- Take a block closure.

```
heightFunction := [ :u :v | |x y d|
    x := u *2.0 - 1.0.
    y := v *2.0 - 1.0.
    d := ((x*x) + (y*y)) sqrt.
    (d * 10.0) sin * 0.5 + 0.5
].
```

- Obtain the AST node from the closure.
- ```
  closureNode := closure sourceNode.
  ```
- Obtain the captured variable values

```
copiedVariables := closureNode scope inComingCopiedVars asArray.
```
```

“Semantic Analysis”

- Local type inference
  - Literals.
  - Captured variables.
  - Special objects (e.g: Color Ramp).
- Type information is used for mapping messages to functions.
- Some messages are always mapped to the same function name (e.g. abs, cos, sin).
- AST is visited, and type information is propagated.

“Code Generation”

- Generate the AST of another shader language.
- Woden Engine has the custom shader language Dastrel.
- Dastrel has C++ 11 style type inference (**auto** keyword).
- The full Dastrel compiler is written in Pharo.
- Dastrel output is the Slovim (Smalltalk Low-Level Virtual Machine) SSA IR. Heavily based on LLVM.
- Slovim has a Spir-V backend for (SSA IR for Vulkan).

Translation difficulties

- Dastrel syntax is statement based like C.
- Type inference ambiguities.

```
|a b|
someCondition ifTrue: [
   a := 1.0.
   b := 2.
] ifFalse: [
   a := 1.
   b := 2.0.
].
```

“Runtime System”

```
import fragment.stage;
import fragment.screenQuad;
import procedural.noise;

code_block(fragment) main
{
    let uv = FragmentInput.texcoord;
    let color = colorFunction(uv.x, uv.y);
    FragmentStage.colorOutput0 = color;
}
```

```
colorShaderForFunction: aColorFunction
    codeConverter := DASLPharoCodeConverter new.
    codeConverter convertFunction: aColorFunction name: #colorFunction argumentTypes: #(float float)
returnType: #float4.

    ^ self compileShader: 'procedural/coloredTextureInterface.dastrel' injectingNodes: codeConverter
generatedNodes
```

Final shader compilation

```
compileShader: shaderFileName injectingNodes: nodesToInject
   | compiler spirv |
   compiler := DASLCompiler new.
   spirv := compiler
      target: #'spir-v';
      withDebugInformation;
      optimizationLevel: 2;
      addIncludeDirectory: self shadersDirectory;
      sourceFromFileNamed: (self shadersDirectory resolve: shaderFileName asFileReference)
injectingNodes: nodesToInject;
      compile;
      generatedCode.

   compiler ssaModule globalNamed: #main.
   spirv saveTo: 'test.spv'.
   "self halt."

   ^ spirv
```

Shader compilation result

```
              %void = OpTypeVoid
                 %3 = OpTypeFunction %void
             %float = OpTypeFloat 32
           %v2float = OpTypeVector %float 2
%_ptr_Input_v2float = OpTypePointer Input %v2float
%FragmentInput_sve_texcoord = OpVariable %_ptr_Input_v2float Input
           %v4float = OpTypeVector %float 4
                %16 = OpTypeFunction %v4float %float %float
           %float_2 = OpConstant %float 2
           %float_1 = OpConstant %float 1
          %float_10 = OpConstant %float 10
         %float_0_5 = OpConstant %float 0.5
                %38 = OpTypeFunction %v4float %float
              %bool = OpTypeBool
           %float_0 = OpConstant %float 0
                %48 = OpConstantComposite %v4float %float_1 %float_0 %float_0 %float_1
                %54 = OpConstantComposite %v4float %float_0 %float_0 %float_1 %float_1
                %64 = OpConstantComposite %v4float %float_1 %float_0 %float_0 %float_1
                %65 = OpConstantComposite %v4float %float_0 %float_1 %float_0 %float_1
                %71 = OpConstantComposite %v4float %float_0 %float_1 %float_0 %float_1
                %72 = OpConstantComposite %v4float %float_0 %float_0 %float_1 %float_1
%_ptr_Output_v4float = OpTypePointer Output %v4float
%FragmentStage_sve_colorOutput0 = OpVariable %_ptr_Output_v4float Output
           %_anonF0 = OpFunction %v4float None %38
                %39 = OpFunctionParameter %float
                %40 = OpLabel
                %41 = OpFOrdLessThan %bool %39 %float_0
                      OpSelectionMerge %46 None
                      OpBranchConditional %41 %45 %46
                %45 = OpLabel
                      OpReturnValue %48
                %46 = OpLabel
                %49 = OpFOrdGreaterThan %bool %39 %float_1
                      OpSelectionMerge %52 None
                      OpBranchConditional %49 %51 %52
                %51 = OpLabel
                      OpReturnValue %54
```

Fragment of a Spir-V shader disassembly.
This is an encoding for a control flow graph.

Code Generation for Other Backends

- The Khronos Group maintains spirv-cross
- Decompile Spir-V to other shader languages.
- Integrated in the graphics API abstraction layer.

Spir-V to Metal

```
MacBook-Air-de-Ronie:woden-esug-2019-demo ronie$ spirv-cross --msl test.spv
#pragma clang diagnostic ignored "-Wmissing-prototypes"
#include <metal_stdlib>
#include <simd/simd.h>
using namespace metal;
struct main0_out
{
    float4 FragmentStage_sve_colorOutput0 [[color(0)]];
};
struct main0_in
{
    float2 FragmentInput_sve_texcoord [[user(locn0)]];
};
float4 _anonF0(float _39)
{
    if (_39 < 0.0)
    {
        return float4(1.0, 0.0, 0.0, 1.0);
```

Graphics API Shading Languages

- Vulkan: Spir-V SSA IR (GLSL compiler available)
- D3D12: HLSL, DirectX Bytecode, DirectX IR (LLVM Bitcode)
- OpenGL: GLSL
- Metal: Metal Shading Language (Modified C++ 11, compiled to undocumented and packed LLVM Bitcode)

Case Study: Procedural Texture Generation

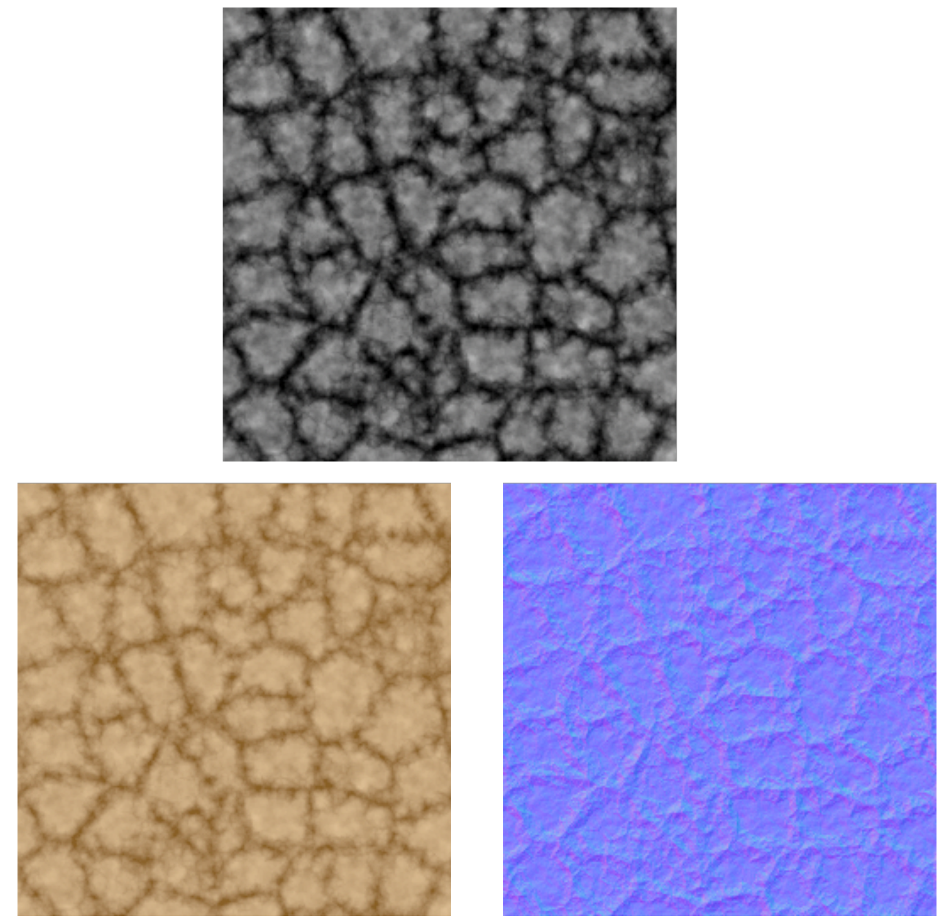
- A texture is a function F that assign a Color (R, G, B, A) to a point (u,v) in a 2D surface:

- $F(u, v) \ with \ u, v \in [0,1]$

- Generating a texture consists on evaluating F in all point in the texture grid.

Procedural Texture Generation Sample

```
| textureSize colorRamp heightFunction |
textureSize := 7.0@7.0.
colorRamp := WDCLinearRamp with: {
   0.0 -> '8a6025' asColor.
   1.0 -> 'f7d8ac' asColor.
}.
heightFunction := [ :s :t |
  | cracks st bumps height |
  st := s@t.
  cracks := (st*textureSize fbmWorleyNoiseOctaves: 4
    lacunarity: 3.0 tiledWith: textureSize)*3.0 min:
    1.0.
  bumps := st*textureSize*4.0
    fbmSignedGradientNoiseOctaves: 4 lacunarity: 2.0
    tiledWith: textureSize*4.0.
  height := (cracks*0.5) + (bumps*0.5).
].
WDCPharoProceduralGPUScriptEvaluator forInspector
   textureExtent: 512@512;
   heightFunction: heightFunction;
   colorMapFunction: colorRamp;
evaluate
```



Speedup factor: 262.45

Case Study: Particle Simulation

- A particle p has a state $Q_{p,t}$ in a given instant of time t.
- Particle simulation consists on computing $Q_{p,t+\Delta t} = S(Q_{p,t}, p, \Delta t)$ for each particle p in a particle system.
- S is a function that simulates a single particle

Particle State Q

- Position.
- Velocity.
- Remaining time of life.
- Size.
- Random number generation seed.

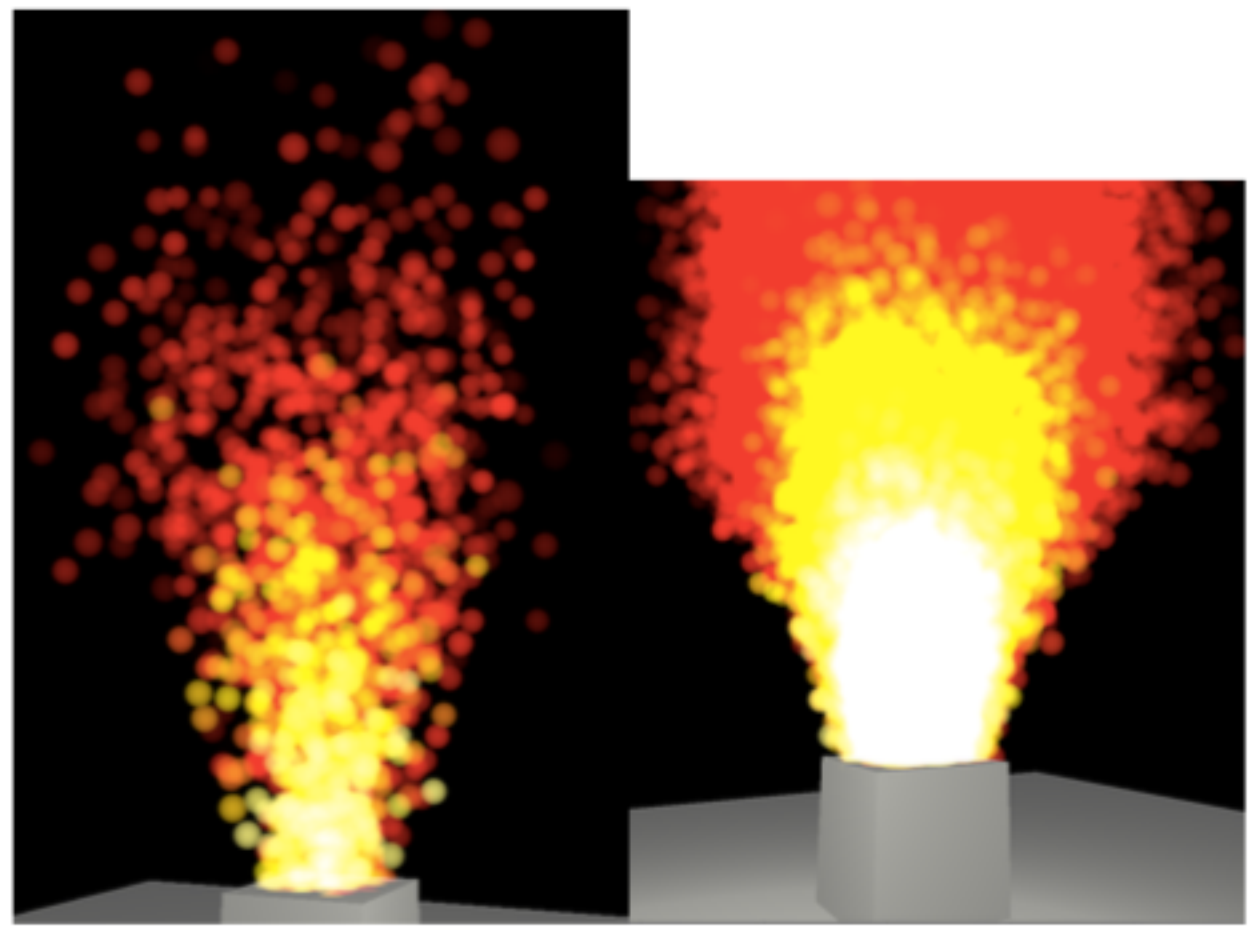
Particle Simulation

```
    0.0 -> '000000' asColor asWMVector3F asWMVector4F.
    0.6 -> 'ff0000' asColor asWMVector3F asWMVector4F.
    0.90 -> 'ffff00' asColor asWMVector3F asWMVector4F.
    1.0 -> 'ffff80' asColor asWMVector3F asWMVector4F.
}.

particleSystemRenderable simulationBlock: [ :particleState :index :delta |
    | lifeTime color flickering |
    lifeTime := particleState lifeTime  - delta.
    lifeTime <= 0.0 ifTrue:[
        lifeTime := 1.7 + particleState nextRandom*1.5.
        particleState
            startingUp: false;
            position: particleState nextRandomVector3F * 0.25;
            velocity: (WMVector3F
                x: particleState nextRandom*0.5
                y: 2.0 + (particleState nextRandom *0.5)
                z: particleState nextRandom*0.5).
    ].

    color := colorRamp value: lifeTime / 3.0.
    flickering := (lifeTime*25.0) signedGradientNoise  *0.4 + 0.6.

    particleState
        size: (WMVector3F x: 0.2 y: 0.2);
        velocity: (particleState velocity + (WMVector3F y: -9.8 * delta*0.04));
        position: (particleState position + (particleState velocity *delta));
        color: color * flickering;
        lifeTime: lifeTime.
].
```



(a) $N = 2 * 10^3$ (b) $N = 10^5$

Results

- Significant speedup factor by using BlockClosures translated to shaders. (Between 14 and 262 times)
- Feasible to translate restricted subset of Smalltalk to shaders.

Limitations

- Benchmarking biasing against CPU performance.
- Manual message mapping between Pharo methods and target language is required.
- Type inference failures.

Limitation: message mapping

```
noise
    <messageMaps>
    self
        mapMessage: #randomNoise toFunction: #randomNoise returnType: #float;
        mapMessage: #signedRandomNoise toFunction: #signedRandomNoise returnType: #float;

        mapMessage: #valueNoise toFunction: #valueNoise returnType: #float;
        mapMessage: #signedValueNoise toFunction: #signedValueNoise returnType: #float;

        mapMessage: #gradientNoise toFunction: #gradientNoise returnType: #float;
        mapMessage: #signedGradientNoise toFunction: #signedGradientNoise returnType: #float;

        mapMessage: #voronoiNoise toFunction: #voronoiNoise returnType: #float;
        mapMessage: #signedVoronoiNoise toFunction: #signedVoronoiNoise returnType: #float;

        mapMessage: #worleyNoise toFunction: #worleyNoise returnType: #float;
        mapMessage: #signedWorleyNoise toFunction: #signedWorleyNoise returnType: #float;

        mapMessage: #fbmValueNoiseOctaves:lacunarity: toFunction: #fbmValueNoiseOctaves returnType:
#float;
```

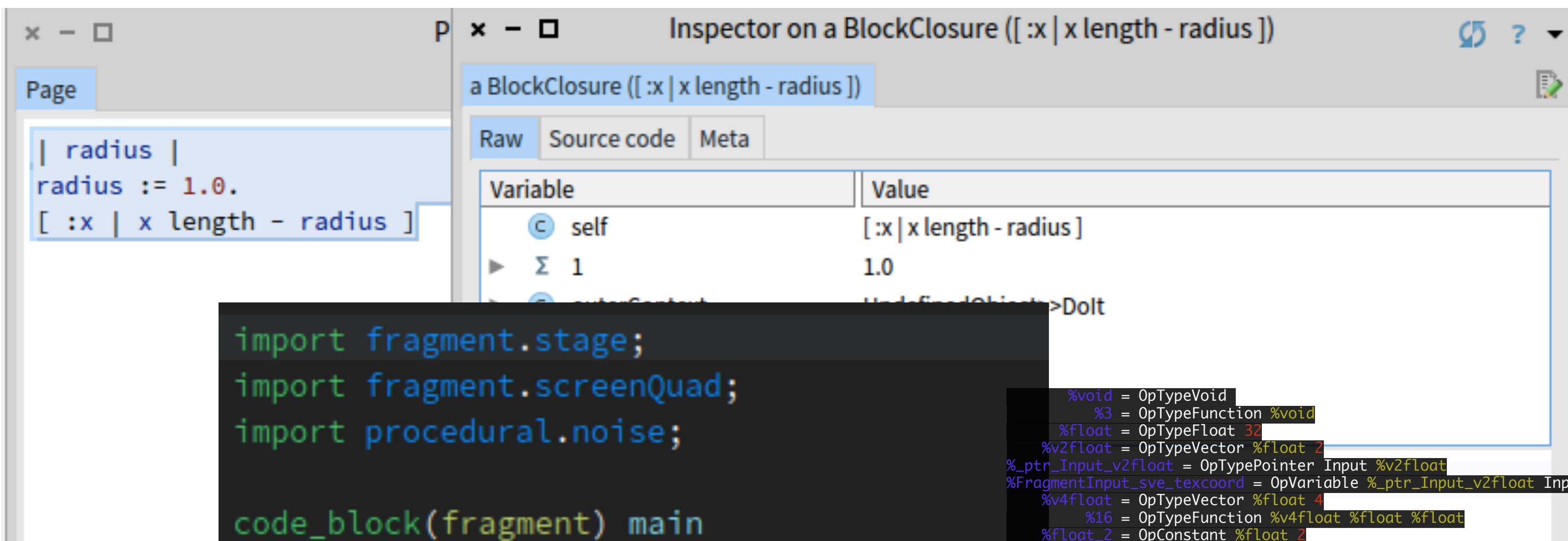
Related Work

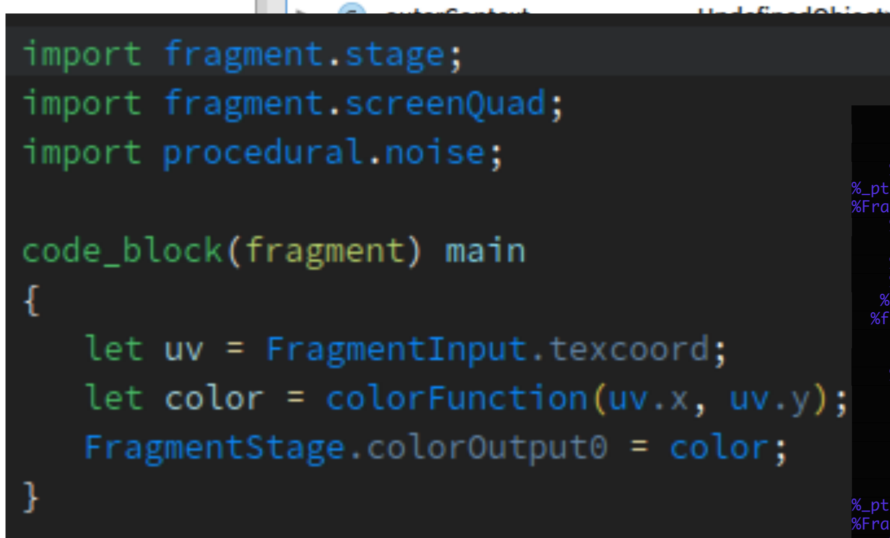
- Slang.
- ShaderToy.
- Others DSLs targeting the GPU.

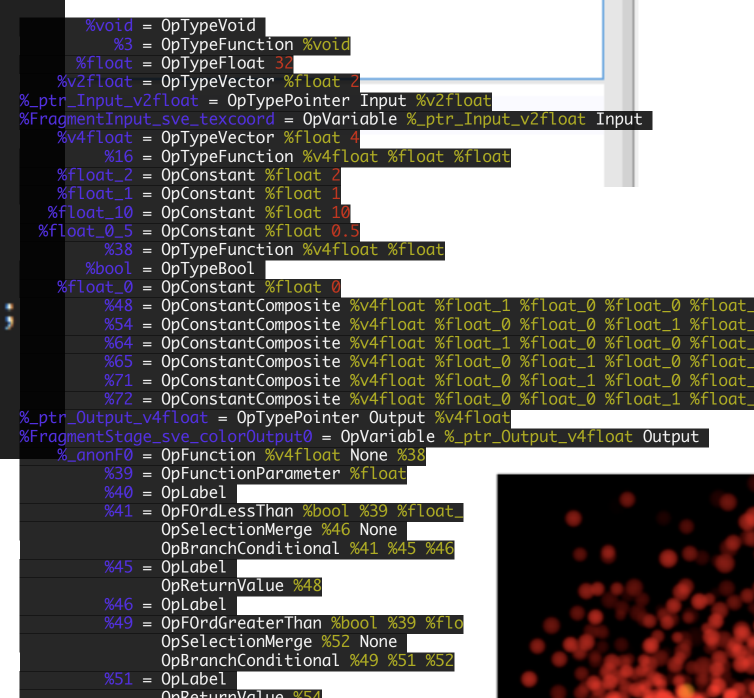
Conclusions and Future Work

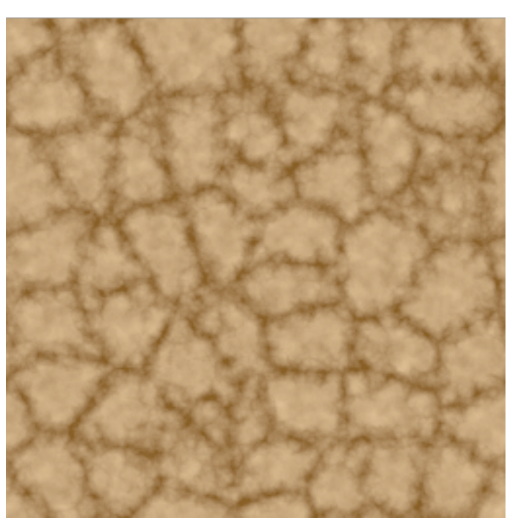
- Feasibility of translating Pharo BlockClosures to shaders.
- Speedup factor between 14 and 262 for procedural texture generation.
- Use a target AST with more direct mapping from Smalltalk.

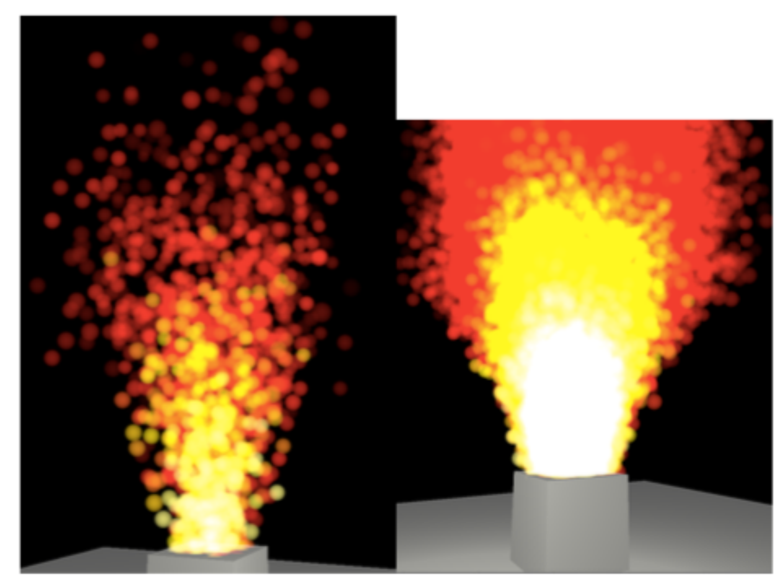
Questions?











(a) N = $2 * 10^3$ (b) N = 10^5