Geometry Types for Graphics Programming: Supplementary Material

Draft

ANONYMOUS AUTHOR(S)

1 GLSL PHONG SOURCE CODE

This section lists the full code for the Phong lighting model in plain GLSL [1].

```
precision mediump float;

// External Function Declarations
uniform mat4 uModel;
uniform mat4 uView;
varying vec3 vNormal;
uniform vec3 uLight;
varying vec3 vPosition;

void main() {
  vec3 ambient = vec3(.1, 0., 0.);
  vec3 lightColor = vec3(0.4, 0.3, 0.9);
  vec3 specColor = vec3(1., 1., 1.);

  vec4 homWorldPos = uModel*vec4(vPosition, 1.0);
  vec3 camPos = normalize(vec3(uView*homWorldPos));
  vec3 worldNorm =
  normalize(vec3(uModel*vec4(vNormal, 0.0)));

  vec3 lightDir =
    normalize(uLight - vec3(homWorldPos));
  vec3 reflectDir = reflect(lightDir, worldNorm);

  vec3 diffuse =
    max(lightWorldDot, 0.0) * lightColor;

  float spec = pow(max(-dot(
    camPos, reflectDir), 0.), 32.);
  vec3 specular = spec * specColor;

  gl_FragColor = vec4(ambient+diffuse+specular, 1.0);
}
```

2 GATOR PHONG SOURCE CODE

This section lists equivalent code in Gator.

2020. 2475-1421/2020/1-ART1 $15.00
https://doi.org/

```
using "../glsl_defs.lgl";

type color is vec3;
type alphaColor is vec4;

// Reference Frame Declarations

frame model has dimension 3;
frame world has dimension 3;
frame camera has dimension 3;
frame light has dimension 3;

// Global Variables

varying cart3<model>.point vPosition;
uniform hom<model>.transformation<world> uModel;
uniform hom<world>.transformation<camera> uView;
varying cart3<model>.vector vNormal;
uniform cart3<light>.point uLight;
uniform hom<light>.transformation<world> uLightTrans;

// Boilerplate application definitions to help in expressions
canon hom<world>.point app_uModel(hom<model>.point v) {
  return uModel * v;
}
canon hom<world>.vector app_uModel(hom<model>.vector v) {
  return uModel * v;
}
canon hom<camera>.point app_uView(hom<world>.point v) {
  return uView * v;
}
canon hom<camera>.vector app_uView(hom<world>.vector v) {
  return uView * v;
}
canon hom<world>.point app_uLightTrans(hom<light>.point v) {
  return uLightTrans * v;
}
canon hom<world>.vector app_uLightTrans(hom<light>.vector v) {
  return uLightTrans * v;
}

// Shader Code

void main() {
  color ambient = [.1, 0., 0.];
  color diffColor = [0.4, 0.3, 0.9];
  color specColor = [1.0, 1.0, 1.0];


```

```
  auto worldPos = vPosition in world;
  auto camPos = worldPos in camera;
  auto worldNorm = normalize(vNormal in world);

  auto lightDir = normalize((uLight in world) - worldPos);
  auto lightWorldDot = dot(lightDir, worldNorm);
  scalar diffuse = max(lightWorldDot, 0.0);

  auto reflectDir = normalize(reflect(-lightDir, worldNorm) in camera);

  scalar specular = pow(max(dot(normalize(-camPos), reflectDir), 0.), 32.);

 vec4 gl_FragColor = vec4(ambient + diffuse * diffColor + specular * specColor, 1.0);
}
```

3 FULL FORMAL SEMANTICS

[Pretty much everything after this line requires some semantic updates] xxx

3.1 Gator's Syntax

3.1.1 *Type System.*

$$c \in \text{constants}$$
$$x \in \text{variables}$$
$$p \in \text{primitives}$$
$$t \in \text{types}$$
$$\tau ::= \text{unit} \mid \top_p \mid \bot_p \mid t$$
$$f \in \text{function names}$$
$$e ::= v \mid c \mid f(e_1, e_2)$$
$$c ::= \tau\ x = e \mid x = e \mid e$$
$$P = c; P \mid \epsilon$$

3.2 Gator's static rules

$$\frac{\tau_1 \leq \tau_2 \qquad \Gamma \vdash e : \tau_1, \Gamma}{\Gamma \vdash e : \tau_2, \Gamma} \qquad \frac{\Gamma \vdash X(c) = p}{\Gamma \vdash c : \bot_p, \Gamma} \qquad \frac{\Gamma \vdash \Gamma(v) = \tau}{\Gamma \vdash v : \tau, \Gamma} \qquad \frac{\Gamma \vdash e : \tau, \Gamma'}{\Gamma \vdash \tau\ x := e : \text{unit}, \Gamma', x \mapsto \tau}$$

$$\frac{\Gamma \vdash \Gamma(x) = \tau \qquad \Gamma \vdash e : \tau, \Gamma'}{\Gamma \vdash x := e : \text{unit}, \Gamma'} \qquad \frac{\Gamma \vdash e_1 : \tau_1, \Gamma' \qquad \Gamma', \vdash e_2 : \tau_2, \Gamma'' \qquad \Gamma'' \vdash \Phi(f, \tau_1, \tau_2) = \tau_3, \Gamma''}{\Gamma \vdash f(e_1, e_2) : \tau_3, \Gamma''}$$

3.3 Ordering rules

$$\frac{}{\tau \leq \tau} \qquad \frac{\tau_1 \leq \tau_2 \qquad \tau_2 \leq \tau_3}{\tau_1 \leq \tau_3} \qquad \frac{}{\tau \leq \text{unit}}$$

$$\frac{}{\bot_p \leq \top_p} \qquad \frac{\tau \leq \top_p}{\bot_p \leq \tau}$$

3.4 Target language grammar

Hatchling is an abstraction over sound imperative languages.

It has an *operation context* Ξ that maps operator names to their their implementation. Ξ : (operator names, expressions, expressions, σ) $\rightarrow$ expressions. Further, Hatchling has an *operator typing context* ξ : operator names $\rightarrow$ types that maps the operator to its input types and the output type. ξ is sound in that should it allow an operation to type check to some type τ then Ξ would return an expression of type τ. For simplicity, and unlike Gator, Hatchling does not allow function overloading.

We use the symbol $||\vdash$ for judgment in the target language

3.4.1 *Hatchling's syntax.*

$$c \in \text{constants}$$
$$x \in \text{variables}$$
$$p \in \text{primitives}$$
$$t \in \text{types}$$
$$\tau ::= \text{unit} \mid \top_p \mid \bot_p$$
$$o \in \text{operation names}$$
$$e ::= v \mid c \mid o(e_1, e_2)$$
$$c ::= \tau\, x = e \mid x = e \mid e$$
$$P = c; P \mid \epsilon$$

3.4.2 *Hatchling's static semantics.* Subsumption

$$\frac{\Gamma \mid\mid\vdash e : \tau, \Gamma}{\Gamma \mid\mid\vdash e : \text{unit}, \Gamma}\ \text{SUBSUMPTION}$$

Constants and variable declarations

$$\frac{}{\Gamma \mid\mid\vdash () : \text{unit}, \Gamma}\ UNIT$$
$$\frac{\Gamma \vdash \Gamma(v) = \tau}{\Gamma \vdash v : \tau, \Gamma}\ VAR$$
$$\frac{\Gamma \vdash X(c) = p}{\Gamma \vdash c : \bot_p, \Gamma}\ PRIMITIVE$$
$$\frac{\Gamma \mid\mid\vdash e : \tau, \Gamma'}{\Gamma \mid\mid\vdash \tau x := e : \text{unit}, \Gamma', x \mapsto \tau}\ DECL$$
$$\frac{\Gamma \vdash e : \tau, \Gamma' \quad \Gamma(x) = \tau}{\Gamma \mid\mid\vdash x := e : \text{unit}, \Gamma', x \mapsto \tau}\ ASSIGN$$

Operations

$$\frac{\Gamma \vdash e_1 : \tau_1, \Gamma' \qquad \Gamma', \vdash e_2 : \tau_2, \Gamma'' \qquad \Gamma'' \vdash \Xi(o) = \tau_3, \Gamma''}{\Gamma \vdash o(e_1, e_2) : \tau_3, \Gamma''}\ OP$$

3.5 Translation

We translate Gator to hatchling. Every translation is under a given Δ and so we treat $[\![]\!]$ as meaning $[\![]\!]_\Delta$ unless otherwise specified.

3.5.1 Literals.

$$[\![c]\!] \triangleq c$$

3.5.2 Types.

$$[\![x]\!] \triangleq x$$

$$[\![\tau x := e]\!] \triangleq [\![\tau]\!][\![x]\!] := [\![e]\!]$$

$$[\![x := e]\!] \triangleq [\![x]\!] := [\![e]\!]$$

$$[\![f(e_1 : \tau_1, e_2 : \tau_2)]\!] \triangleq [\![\Psi(f, e_1, e_2, \tau_1, \tau_2)]\!]$$

3.5.3 Expressions.

$$[\![T]\!] \triangleq \top_p$$

Where we know $\top_n$ is always defined because of the structure imposed on a well formed Δ.

$$[\![\top_p]\!] \triangleq \top_p$$

$$[\![\bot_p]\!] \triangleq \top_p$$

$$[\![\text{unit}]\!] \triangleq \text{unit}$$

3.5.4 . Additionally, we define the translation of Γ and Φ; in the target language, we replace every τ in the range of Γ and Φ with its translation to get $[\![\Gamma]\!]$ and $[\![\Phi]\!]$. This is a different translation function than the one previously discussed, but for convenience we will use the same notation $[\![\Gamma]\!]$ and $[\![\Phi]\!]$.

Lemma 1.

$$\frac{[\![\Gamma \vdash \Gamma(x) : \tau]\!]}{[\![\Gamma]\!] \mid\!\vdash [\![\Gamma]\!]([\![x]\!]) : [\![\tau]\!]}$$

$$\frac{[\![\Gamma \vdash \Phi(f, \tau_1, \tau_2) : \tau_3]\!]}{[\![\Gamma]\!] \mid\!\vdash [\![\Gamma]\!]([\![\Phi(f, \tau_1, \tau_2)]\!]) : [\![\tau]\!]}$$

Proof. Follows from the definitions of $[\![\Gamma]\!]$ and $[\![\Phi]\!]$. □

3.6 Proof that if source type checks then translation type checks

We use structural induction over Gator commands to show that any translation under a valid Δ type checks. In the process of the induction we use the theorem that if a Gator expression type checks then its translation does too. This is also shown using structural induction over all Gator expressions.

Theorem 1. *Given any Gator expression e that type checks under some Γ, Δ to produce τ, Γ', we prove that its translation $[\![e]\!]$ also type checks under $[\![\Gamma]\!]$ to produce $[\![\tau]\!], [\![\Gamma']\!]$.*

Theorem 2. *Given any Gator expression e that type checks under some Γ, Δ to produce τ, Γ', we prove that its translation $[\![e]\!]$ also type checks under $[\![\Gamma]\!]$ to produce $[\![\tau]\!], [\![\Gamma']\!]$.*

Below, I.H. = Inductive Hypothesis. Sometimes the inductive hypothesis is used over a general τ. This implicitly excludes the special case of the polymorphic type.
We use the inversion of the type checking rules in Gator; if a statement in Gator type checks then we know that the premise of the corresponding type checking rule is true. Using this rule will be

marked as T.C.

We split the proof into cases, one for each of the typing judgement rules in Gator.

3.6.1 Expressions Type Check. We present the most interesting case, for primitives. The other cases follow vry similarly.

Primitive rule.

$$\frac{\dfrac{}{[\![\Gamma]\!] \mid\!\vdash c : \top_p, [\![\Gamma]\!]}\ \text{PRIM} \qquad [\![\top_p]\!] \triangleq \top_p \qquad [\![c]\!] \triangleq c}{[\![\Gamma]\!] \mid\!\vdash [\![c]\!] : [\![\top_p]\!], [\![\Gamma]\!]}\ \text{SUBST}$$

Functions type check because of the constraints on a well formed Ψ.

$$\frac{\dfrac{\Gamma \vdash f(e_1 : \tau_1, e_2 : \tau_2) : \tau_3, \Gamma''}{\Phi(f, \tau_1, \tau_2) = \tau_3} \qquad [\![\Gamma]\!] \mid\!\vdash \Psi(f, e_1, e_2, \tau_1, \tau_2) = [\![\Phi(f, \tau_1, \tau_2)]\!], [\![\Gamma'']\!]}{[\![\Gamma]\!] \mid\!\vdash [\![f(e_1 : \tau_1, e_2 : \tau_2)]\!] : [\![\tau_3]\!], [\![\Gamma']\!]}\ \text{SUBST}$$

3.6.2 Commands Type Check.

3.6.3 Declaration. One case is presented here, and others follow suit along the same lines.

Declaration.

$$\frac{\dfrac{\dfrac{[\![\Gamma \vdash \tau x := e : \text{unit}, \Gamma', x \mapsto \tau]\!]}{[\![\Gamma \vdash e : \tau, \Gamma']\!]}\ \text{T.C.}}{[\![\Gamma]\!] \mid\!\vdash [\![e]\!] : [\![\tau]\!], [\![\Gamma']\!]}\ \text{I.H.}}{[\![\Gamma]\!] \mid\!\vdash [\![\tau]\!][\![x]\!] := [\![e]\!] : \text{unit}, [\![\Gamma']\!], [\![x]\!] \mapsto [\![\tau]\!]}\ \text{DECL} \qquad [\![\tau x := e]\!] \triangleq [\![\tau]\!][\![x]\!] = [\![e]\!]$$

$$\frac{[\![\Gamma]\!] \mid\!\vdash [\![\tau x := e]\!] : \text{unit}, [\![\Gamma']\!], [\![x]\!] \mapsto [\![\tau]\!] \qquad [\![\Gamma', x \mapsto \tau]\!] \triangleq \quad [\![\Gamma']\!], [\![x]\!] \mapsto [\![\tau]\!] \quad [\![\text{un}}{[\![\Gamma]\!] \mid\!\vdash [\![\tau x := e]\!] : [\![\text{unit}]\!], [\![\Gamma', x \mapsto \tau]\!]}\ \text{SUBST}$$

REFERENCES

[1] The Khronos Group Inc. [n. d.]. *The OpenGL ES Shading Language* (1.0 ed.). The Khronos Group Inc.