ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Code Example

October 8th, 2015

Rafael Radkowski

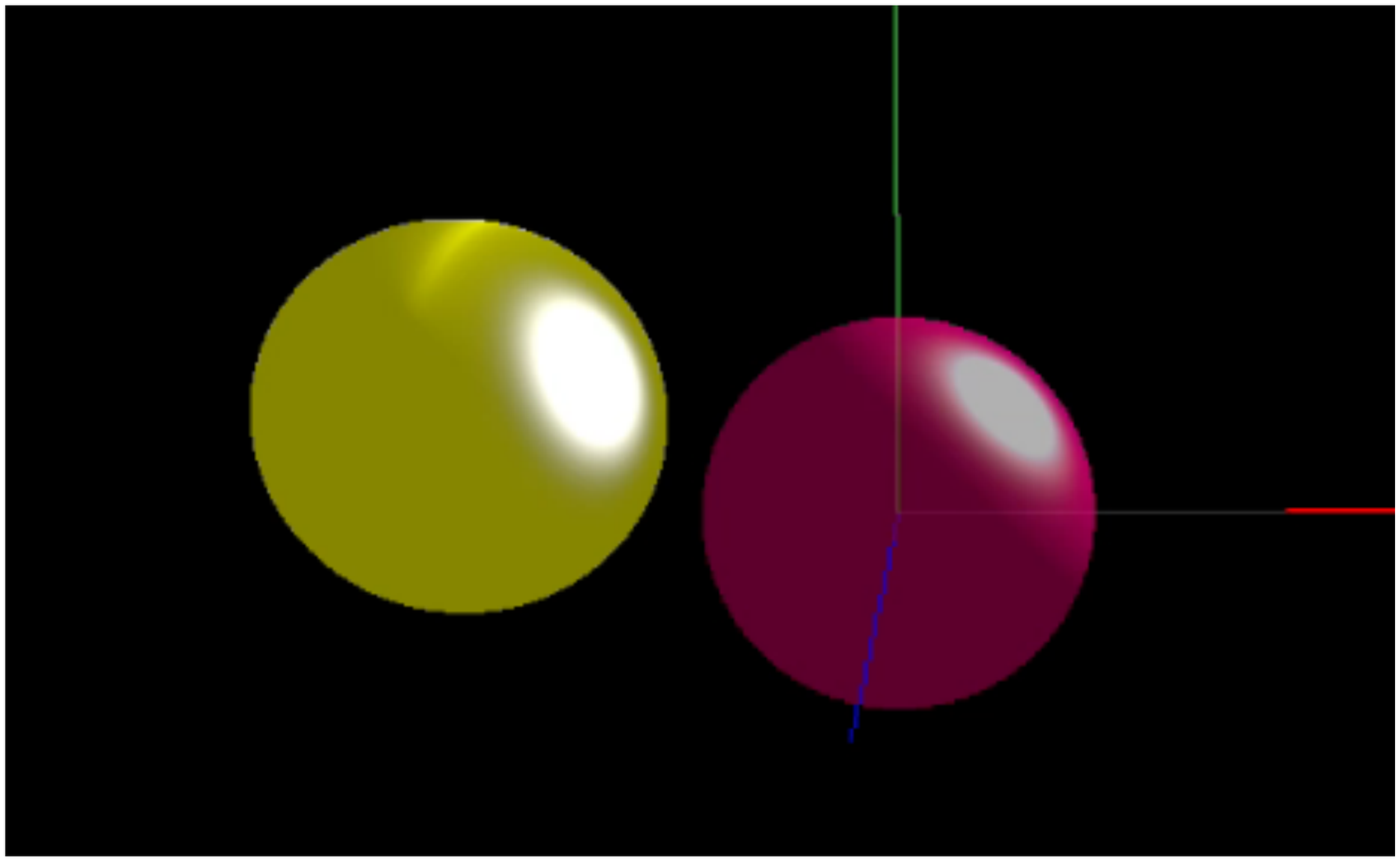
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Content

- New objects
- Blending example
- Texture example

ARLAB

main_transparency.cpp



Code Example

ARLAB

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//// Create some models

// coordinate system
CoordSystem* cs = new CoordSystem(40.0);

// create an apperance object.
GLAppearance* apperance_0 = new GLAppearance("../../data/shaders/multi_vertex_lights.vs", "../../data/shaders/multi_vertex_li
GLDirectLightSource  light_source;
light_source._lightPos = glm::vec4(20.0,20.0,0.0, 0.0);
light_source._ambient_intensity = 0.2;
light_source._specular_intensity = 5.5;
light_source._diffuse_intensity = 2.0;
light_source._attenuation_coeff = 0.0;

// add the light to this apperance object
apperance_0->addLightSource(light_source);

// Create a material object
GLMaterial material_0;
material_0._diffuse_material = glm::vec3(1.0, 0.0, 0.2);
material_0._ambient_material = glm::vec3(1.0, 0.0, 0.2);
material_0._specular_material = glm::vec3(1.0, 1.0, 1.0);
material_0._shininess = 12.0;
material_0._transparency = 0.4;

// Add the material to the apperance object
apperance_0->setMaterial(material_0);
apperance_0->finalize();

// create the sphere geometry
GLSphere3D* sphere_0 = new GLSphere3D(0.0, 0.0, 0.0, 10.0, 90, 50);
sphere_0->setApperance(*apperance_0);
sphere_0->init();

// If you want to change appearance parameters after you init the object, call the update function
apperance_0->updateLightSources();
```

GLAppearance: defines the appearance of an object

GLDirectLightSource: defines a direct light source

GLMaterial: defines the material.

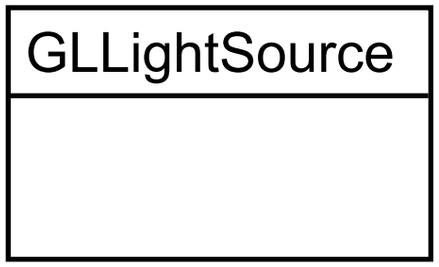
Code Example

ARLAB

```
  // create a second apperance object.
  GLAppearance* apperance_1 = new GLAppearance("../../data/shaders/multi_vertex_lights.vs", "../../data/shaders/
multi_vertex_lights.fs");

  // add the light to this apperance object
  apperance_1->addLightSource(light_source);

  // Create a material object
  GLMaterial material_1;
  material_1._diffuse_material = glm::vec3(1.0, 1.0, 0.0);
  material_1._ambient_material = glm::vec3(1.0, 1.0, 0.0);
  material_1._specular_material = glm::vec3(1.0, 1.0, 1.0);
  material_1._shininess = 8.0;
  material_1._transparency = 1.0;

// Add the material to the apperance object
apperance_1->setMaterial(material_1);
apperance_1->finalize();

GLSphere3D* sphere_1 = new GLSphere3D(0.0, 0.0, 0.0, 10.0, 90, 50);
sphere_1->setApperance(*apperance_1);
sphere_1->init();

glm::mat4 tranform = glm::translate(glm::mat4(1.0f), glm::vec3(-22.0f, 5.0f, 0.0f));
sphere_1->setMatrix(tranform);
```

GLLightSource

ARLAB



The class GLLightSource is the base class which keeps all the parameters for a light source along with the names and indices of the shader code variables:

- diffuse, specular, and ambient intensity
- light position / light direction variable
- attenuation coefficient

Constructor:

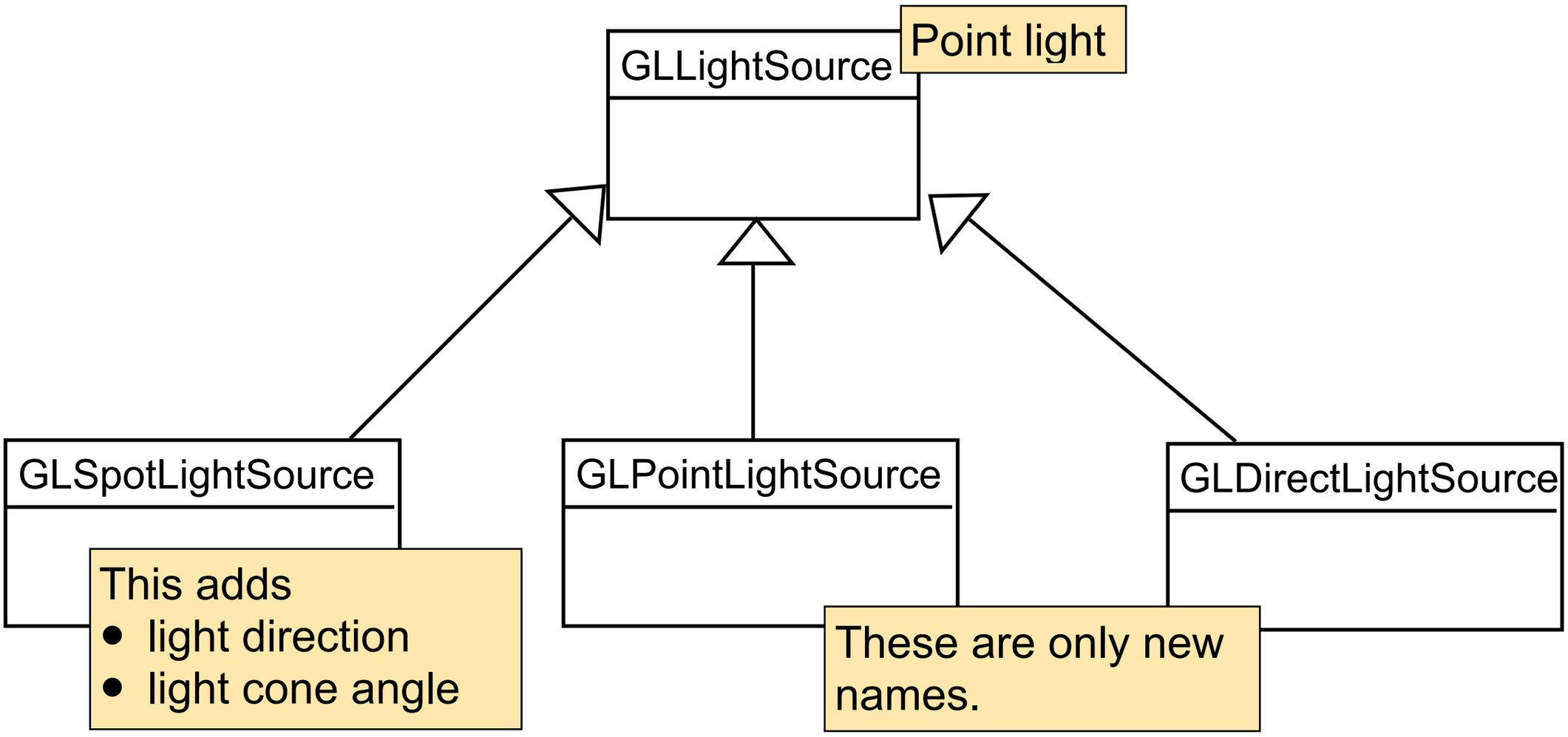
`GLLightSource()`

- no parameters

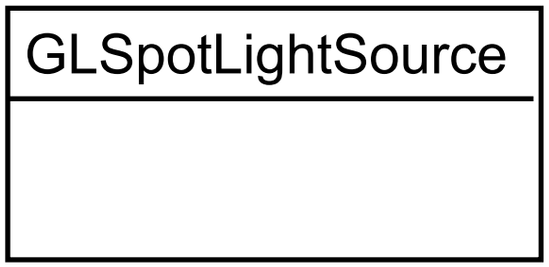
Example:

```
GLDirectLightSource  light_source;

light_source._lightPos = glm::vec4(20.0,20.0,0.0, 0.0);
light_source._ambient_intensity = 0.2;
light_source._specular_intensity = 5.5;
light_source._diffuse_intensity = 2.0;
light_source._attenuation_coeff = 0.0;
```

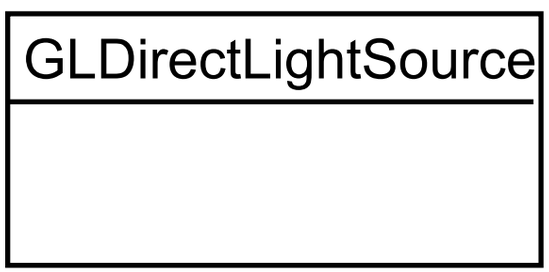
Light Source Hierarchy



Examples




```
// The spotlight object
GLSpotLightSource  light_source1;
light_source1._lightPos = glm::vec4(20.0,20.0,0.0, 1.0);
light_source1._ambient_intensity = 0.3;
light_source1._specular_intensity = 8.0;
light_source1._diffuse_intensity = 3.0;
light_source1._attenuation_coeff = 0.02;
light_source1._cone_angle = 12.0; // in degree
light_source1._cone_direction = glm::vec3(-1.0, -1.0, 0.0);
```




```
GLDirectLightSource  light_source2;
light_source2._lightPos = glm::vec4(20.0,0.0,0.0, 0.0);
light_source2._ambient_intensity = 0.1;
light_source2._specular_intensity = 5.5;
light_source2._diffuse_intensity = 1.0;
light_source2._attenuation_coeff = 0.02;
```

GLMaterial

ARLAB

GLMaterial

The class maintains the parameters for the diffuse, specular, and ambient reflection equations.

Constructor:

`GLMaterial()`

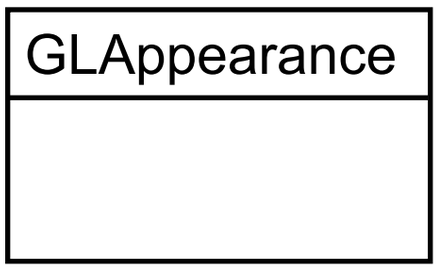
- no parameters

```
// Create a material object
GLMaterial material_1;
material_1._diffuse_material = glm::vec3(1.0, 1.0, 0.0);
material_1._ambient_material = glm::vec3(1.0, 1.0, 0.0);
material_1._specular_material = glm::vec3(1.0, 1.0, 1.0);
material_1._shininess = 8.0;

material_1._transparency = 1.0;
```

The parameter "transparency" is our alpha value; we use only one to simply the programs.

GLAppearance

ARLAB



The class GLAppearance allows us to define the appearance on an object. An instance of this class combines:

- the shader program with
- the material definition and
- light sources (can be more than one).

Constructor:

`GLAppearance(string vertex_shader_file, string fragment_shader_file);`

- vertex_shader_file - the path and file for the vertex shader code
- fragment_shader_file - the path and file for the fragment shader code

Example:

```
GLAppearance* apperance_1 =
     new GLAppearance( "../../data/shaders/multi_vertex_lights.vs",
                       "../../data/shaders/multi_vertex_lights.fs");
```

GLAppearance - Functions to create an apperance

ARLAB

`void addLightSource(GLLightSource& light_source);`

Add a light source that should illuminate this object. You can add multiple light sources

- light_source - a light source reference

```
// add the light to this apperance object
apperance_1->addLightSource(light_source);
```

`void setMaterial(GLMaterial& material);`

Set the material for this object. You can only set ONE material.

- material - a reference to a material object

```
// Add the material to the apperance object
apperance_1->setMaterial(material_1);
```

`void finalize(void);`

Finalize the program and all variables. CALL THIS, WHEN YOU ARE READY!

```
apperance_1->finalize();
```

GLAppearance - Functions to update an appearance

ARLAB

Call this functions whenever you change parameter values.

`void updateMaterial(void);`

Update all the materials if values changes

```
apperance_1->updateMaterial();
```

`void updateLightSources(void);`

Updates all the light sources

```
apperance_1->updateLightSources();
```

Keep in mind

To create an object, use the **pointer** operator along with the **new** operator:

```
Object * variable_name = new Object( ..... )
```

Pointer operator

new operator

```
GLAppearance* apperance_1 =
      new GLAppearance( "../../data/shaders/multi_vertex_lights.vs",
                        "../../data/shaders/multi_vertex_lights.fs");
```

To invoke a function, call the object and refer to the function using the **arrow** operator

```
variable_name->function( .... )
```

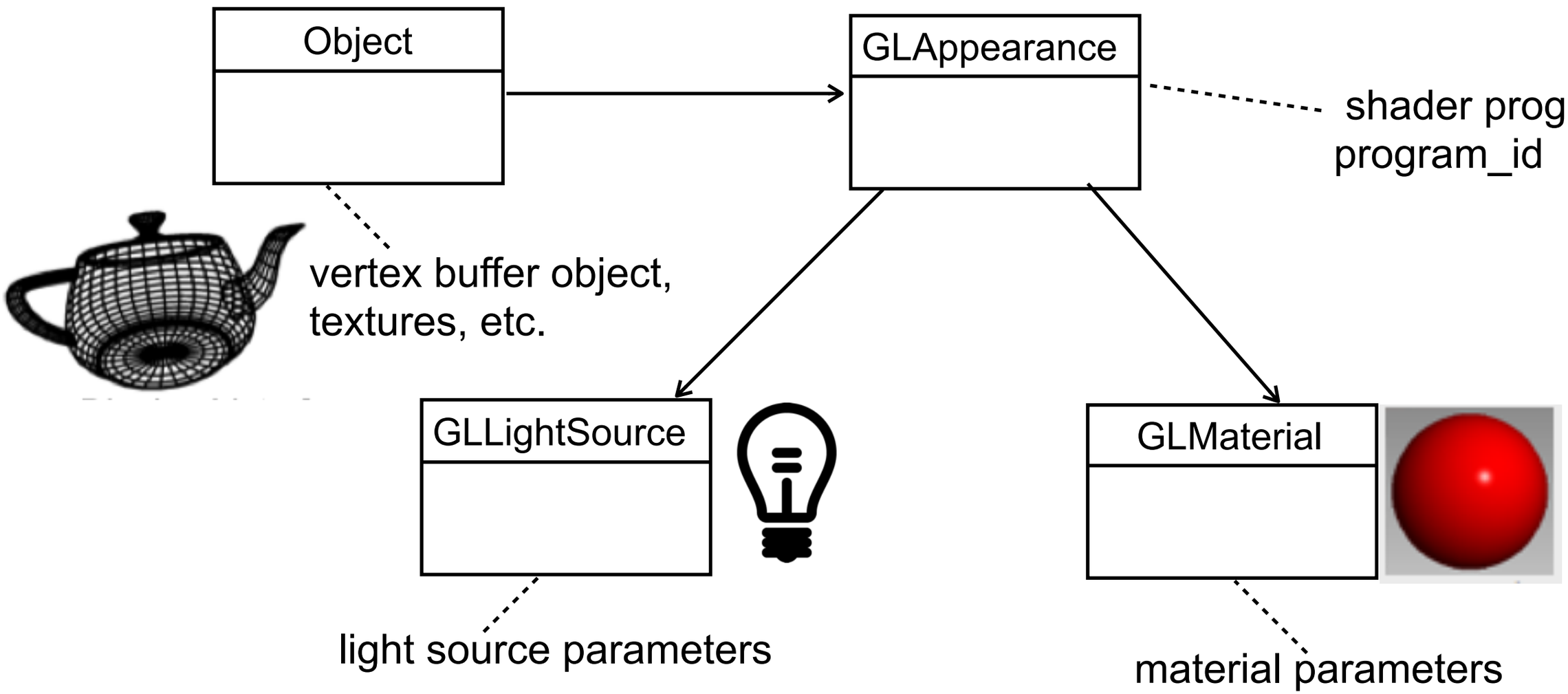
Arrow operator

```
apperance_1->updateLightSources();
```

Object Relations

ARLAB

The objects such as the GLSphere3D and others only maintain the geometry.



Advantage

- Reuse light sources
- Reuse material
- Simplifies the object handling (reusing code in general without the need for copy & paste)

Advantage

ARLAB

```
/*!
 A Material class, which allows us to define a material
 */
class GLMaterial : public GLVariable
{
protected:

    // These are the variable names which are used in our glsl shader programs.
    // Make sure that you use the correct names in your programs.
    string      _glsl_names[6] = { "ambient", "diffuse",  "specular",  "shininess", "emissive", "transparency"};
    string      _glsl_struct = "allMaterials";

public:
```

```
/*
 A light source class, which allows us to represent a light source.
 */
class GLLightSource  : public GLVariable
{
protected:

    // These are the variable names which are used in our glsl shader programs.
    // Make sure that you use the correct names in your programs.
    string      _glsl_names[5] = { "specular_intensity", "diffuse_intensity",  "ambient_intensity",
"attenuationCoefficient", "light_position"};
    string      _glsl_object = "allLights";


public:
```

The classes maintain a list of the related shader variables.

GLAppearance - Link the shader variables

ARLAB

```
virtual bool addVariablesToProgram(GLuint program, int variable_index);
```

Add all the variables of this material object to the shader program "program".

It expects that the program already exits and that the names in _glsl_names are used

- program - the glsl shader program integer id
- variable_index - the index number of the specific variable.

You do not need to call this function on your own!!

Host program

```
 string       _glsl_names[6] =
{ "ambient", "diffuse",
"specular",  "shininess",
"emissive", "transparency"};
```

GLSL Shader program

```
// The material parameters
uniform struct Material {
    vec3 diffuse;
    vec3 ambient;
    vec3 specular;
    vec3 emissive;
    float shininess;
    float transparency;
} allMaterials[1];
```

Struct

A struct - a data structure is a group of data elements grouped together under one name. These data elements, known as members, can have different types and different lengths.

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

```
// The material parameters
uniform struct Material {
    vec3 diffuse;
    vec3 ambient;
    vec3 specular;
    vec3 emissive;
    float shininess;
    float transparency;
} allMaterials[1];
```

We only add uniform when the data comes from the host program.

Access to structs:

Assume we have the struct allMaterials, which is an array with length 1.

```
vec3 diffuse = allMaterials[0].diffuse;
```

```
vec3 diffuse = name . variable_name;
```

Dot operator

Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 5001, USA

+1 515.294.5580
+1 515.294.5530(fax)

rafael@iastate.edu

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY