Revised Lighting Classes

Objectives

- Rework the lighting experiments into a coherent object model, incorporating:
    - LightSetting
    - AmbientLight
    - SpotLight
- Encapsulate these into a LightingModel class

Current Lighting Approach

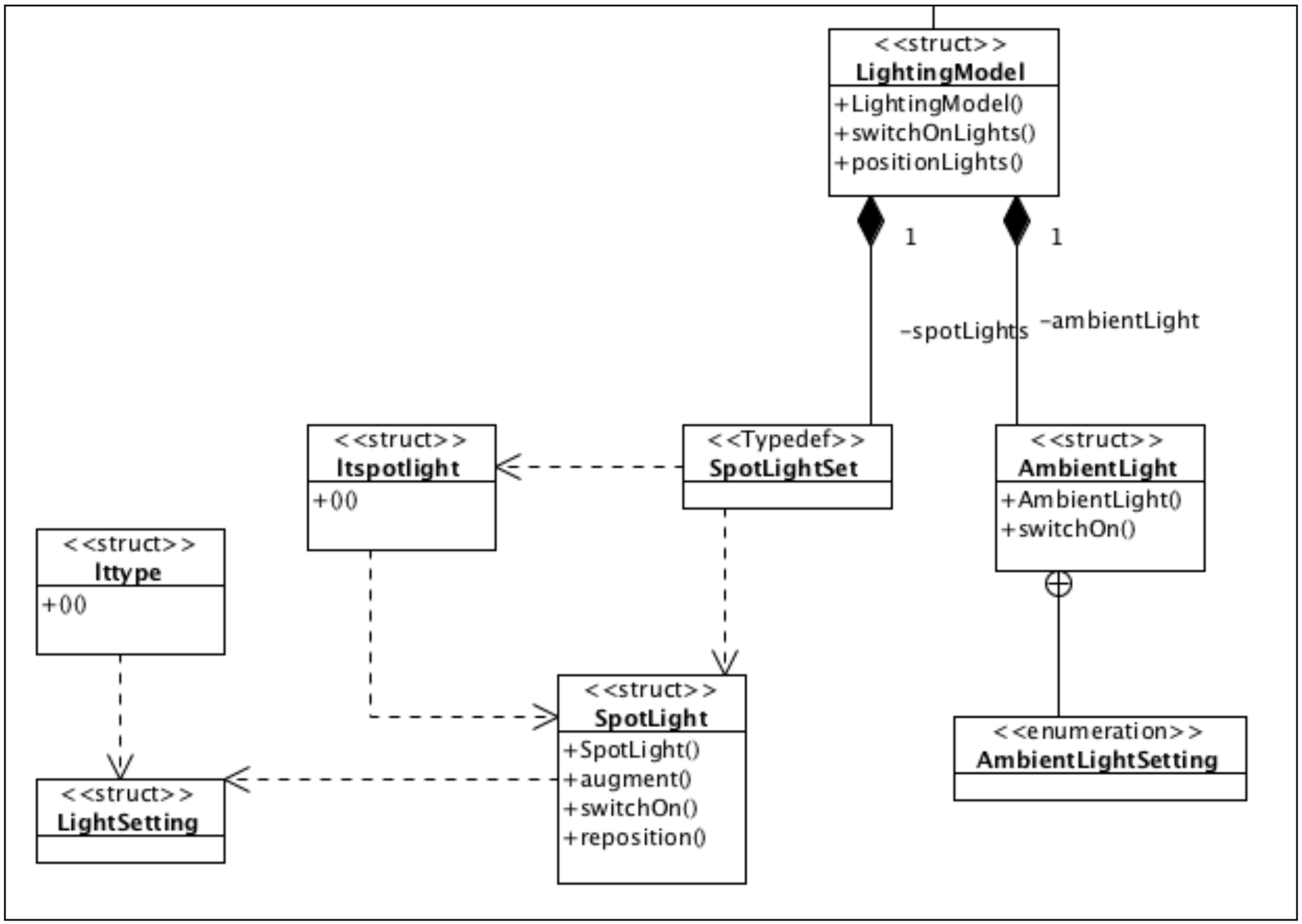
```
enum MaterialTypes {flatRed, flatYellow, flatBlue,
flatGreen, flatGray, plasticRed, shinyWhite, brass, bronze,
  chrome, copper, gold, pewter, silver, polishSilver,
plasticBlack} ;

void applyMaterial(MaterialTypes material);

void basicAmbient();
void grayMaterial();
void colourTracking();
void positionLight();
void lightSource();
```

- Unencapsulated ‘experiments’

```
struct Material
{
  GLfloat ambient[4];
  GLfloat diffuse[4];
  GLfloat specular[4];
  GLfloat shiny;
};

float ambientLightFull[] = { 1.0f, 1.0f, 1.0f, 1.0f };
float ambientLightHalf[] = { 0.5f, 0.5f, 0.5f, 1.0f };
float ambientLightDefault[] = { 0.2f, 0.2f, 0.2f, 1.0f };
```

```
void applyMaterial(MaterialTypes material)
{
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,
materials[material].ambient);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,
materials[material].diffuse);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,
materials[material].specular);
  glMaterialf (GL_FRONT_AND_BACK, GL_SHININESS, materials[material].shiny);
}

void basicAmbient()
{
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLightFull);
}

void grayMaterial()
{
  float gray[] =  { 0.75f, 0.75f, 0.75f, 1.0f };
  glMaterialfv(GL_FRONT,  GL_AMBIENT_AND_DIFFUSE, gray);
}

void colourTracking()
{
  glEnable(GL_COLOR_MATERIAL);
  glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
}

void lightSource()
{
  float  ambientLightModerate[] = { 0.3f, 0.3f, 0.3f, 1.0f };
  float  diffuseLightModerate[] = { 0.7f, 0.7f, 0.7f, 1.0f };

  glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLightModerate);
  glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLightModerate);

  glEnable(GL_LIGHT0);
  glEnable(GL_NORMALIZE);
}

void positionLight()
{
  float lightPos[] = { -50.f, 50.0f, 0.0f, 1.0f };
  glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
}
```

Proposed Model



LightingModel

- LightingModel - holds references to:
  - AmbientLight
  - SpotLightSet
- The lights can be turned on, and the spot lights ‘positioned’ within a scene

```
struct LightingModel
{
  LightingModel();
  void switchOnLights();
  void positionLights();

  SpotLightSet spotLights;
  AmbientLight ambientLight;
};
```

AmbientLight

```
struct AmbientLight
{
  enum AmbientLightSetting {fullLight, halfLight, defaultLight};

  AmbientLight(AmbientLightSetting setting);
  void switchOn();

  AmbientLightSetting setting;
};
```

- Preprogrammed ‘settings’ of full, half, default

AmbientLight Implementation

```
float settingTable[][4] =
{
  { 1.0f, 1.0f, 1.0f, 1.0f }, // full
  { 0.5f, 0.5f, 0.5f, 1.0f }, // half
  { 0.2f, 0.2f, 0.2f, 1.0f }  // default
};

AmbientLight::AmbientLight(AmbientLightSetting setting)
: setting(setting)
{
}

void AmbientLight::switchOn()
{
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, settingTable[setting]);
}
```

SpotLight

- LightSetting - the components + light type (Ambient, Diffuse, Specular, Shiny)

- Spotlight contains light id, position and a set of light settings

```
struct LightSetting
{
  float*  components;
  int     type;
};

struct lttype
{
  bool operator() (const LightSetting& s1, const LightSetting& s2) const
  {
    return (s1.type < s2.type);
  }
};
```

```
struct SpotLight
{
  SpotLight (GLenum id, Vector3 p);
  void augment(LightSetting &setting);

  void switchOn();
  void reposition();

  GLenum  id;
  Vector3 position;
  std::set<LightSetting, lttype> lightSettings;
};

struct ltspotlight
{
  bool operator() (const SpotLight& s1, const SpotLight& s2) const
  {
    return (s1.id < s2.id);
  }
};

typedef std::set<SpotLight, ltspotlight> SpotLightSet;
```

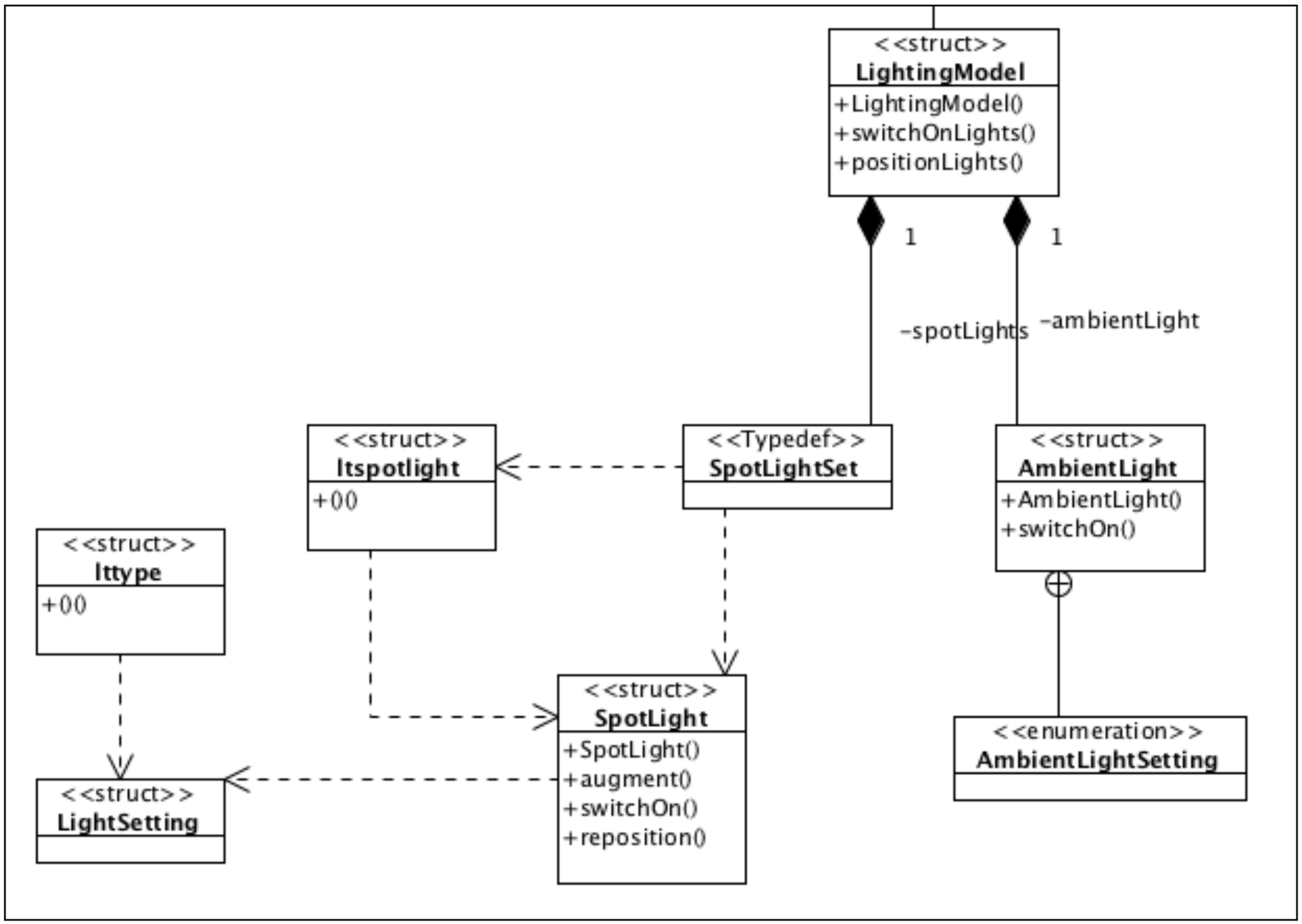
SpotLight Implementation

```
SpotLight::SpotLight (GLenum id, Vector3 p)
: id(id), position(p)
{
}

void SpotLight::augment(LightSetting &setting)
{
  lightSettings.insert(setting);
}

void SpotLight::switchOn()
{
  foreach (LightSetting setting, lightSettings)
  {
    glLightfv(id, setting.type, setting.components);
  }
  glEnable(id);
}

void SpotLight::reposition()
{
  float pos[4];
  pos[0] = position.X;
  pos[1] = position.Y;
  pos[2] = position.Z;
  pos[3] = 1;

  glLightfv(id, GL_POSITION, pos);
}
```

LightingModel Implementation

```
float  ambientLightModerate[] = { 0.3f, 0.3f, 0.3f, 1.0f };
float  diffuseLightModerate[] = { 0.7f, 0.7f, 0.7f, 1.0f };

LightingModel::LightingModel()
: ambientLight(AmbientLight::fullLight)
{
  SpotLight spotLight0 ((GLenum)GL_LIGHT0, Vector3(-50, 50, 0));

  LightSetting setting1 = {ambientLightModerate, GL_AMBIENT};
  LightSetting setting2 = {diffuseLightModerate, GL_DIFFUSE};

  spotLight0.augment(setting1);
  spotLight0.augment(setting2);

  spotLights.insert(spotLight0);
}


void LightingModel::switchOnLights()
{
  ambientLight.switchOn();
  foreach (SpotLight spotLight, spotLights)
  {
    spotLight.switchOn();
  }
}

void LightingModel::positionLights()
{
  foreach (SpotLight spotLight, spotLights)
  {
    spotLight.reposition();
  }
}
```

LightingModel



Exercises

- The lighting settings should ideally be extenalised to a resource file.
- This could contain ambient light settings, and then specifications for each spot light.
- LightModel would then load this and render it into the scene.

- For this type of information, the Yaml file format is useful format:
  - http://en.wikipedia.org/wiki/YAML
- There are many parsers available. This one looks reasonably active, and includes a cross-platform version:
  - http://code.google.com/p/yaml-cpp/

YAML

- YAML: YAML Ain't Markup Language
- What It Is: YAML is a human friendly data serialization standard for all programming languages.
- Goals:
    - YAML is easily readable by humans.
    - YAML data is portable between programming languages.
    - YAML matches the native data structures of agile languages.
    - YAML has a consistent model to support generic tools.
    - YAML supports one-pass processing.
    - YAML is expressive and extensible.
    - YAML is easy to implement and use.

Example

- Includes support for specifying
  - lists,
  - associative arrays,
  - key/value pairs

```
# sequencer protocols for Laser eye surgery
---
- step:  &id001                  # defines anchor label &id001
    instrument:      Lasik 2000
    pulseEnergy:     5.4
    pulseDuration:   12
    repetition:      1000
    spotSize:        1mm

- step: &id002
    instrument:      Lasik 2000
    pulseEnergy:     5.0
    pulseDuration:   10
    repetition:      500
    spotSize:        2mm

- step: *id001                  # refers to the first step (with anchor &id001)
- step: *id002                  # refers to the second step
- step: *id001
- step: *id002
```