

Mestrado em
Engenharia Informática

VI-RT
Ambient Light
Ambient Shader
Standard Renderer

Visualização e Iluminação

Ambient Light (light.hpp)

```
class Light {                // light.hpp
public:
    bool isAmbient;
    Light () {}
    ~Light () {}

    // return the Light RGB radiance for a given point : p
    virtual RGB L (Point p) {return RGB();}

    // return a point p and RGB radiance for a given probability
    prob[2]
    virtual RGB Sample_L (float *prob, Point *p) {return RGB();}

    // return the probability of p
    virtual float pdf(Point p) {return 1.;}
};
```

Ambient Light (light.hpp)

```
class AmbientLight: public Light {
public:
    RGB color;
    AmbientLight (RGB _color): color(_color) { isAmbient = true; }
    ~AmbientLight () {}

    // return the Light RGB radiance for a given point : p
    RGB L (Point p) {return color;}

    // return a point p and RGB radiance for a given probability
    prob[2]
    RGB Sample_L (float *prob, Point *p) {
        p = NULL;
        return color;
    }
};
```

Shaders (shader.hpp)

```
class Shader {  
protected:  
    Scene *scene;  
public:  
    Shader (Scene *_scene): scene(_scene) {}  
    ~Shader () {}  
    virtual RGB shade (bool intersected,  
                        Intersection isect) {  
        return RGB();  
    }  
};
```

Ambient Shader (AmbientShader.hpp)

```
class AmbientShader: public Shader {  
    RGB background;  
  
public:  
    AmbientShader (Scene *scene, RGB bg):  
background(bg), Shader(scene) {}  
  
    RGB shade (bool intersected, Intersection isect);  
};
```

Ambient Shader (AmbientShader.cpp)

```
RGB AmbientShader::shade(bool intersected, Intersection  
isect) {  
    RGB color(0.,0.,0.);  
  
    // if no intersection, return background  
    if (!intersected) {  
        return (background);  
    }  
  
    // ...
```

Ambient Shader (AmbientShader.cpp)

```
RGB AmbientShader::shade(bool intersected, Intersection  
isect) {  
    // ...  
  
    // verify whether the intersected object has an ambient  
    component  
    Phong *f = (Phong *)isect.f;  
    if (f->Ka.isZero()) return color;  
    RGB Ka = f->Ka;  
  
    // ...
```

Ambient Shader (AmbientShader.cpp)

```
RGB AmbientShader::shade(bool intersected, Intersection  
isect) {  
    // ...  
    // Loop over scene's light sources: process Ambient  
    for (auto l = scene->lights.begin() ;  
         l != scene->lights.end() ; l++) {  
        Point p_dummy;  
  
        if (!light_itr->isAmbient) continue;  
        color += Ka * light_itr->L(p_dummy);  
    }  
    return color;  
};
```


Renderer (renderer.hpp)

```
class Renderer {  
protected:  
    Camera *cam;  
    Scene *scene;  
    Image *img;  
    Shader *shd;  
public:  
    Renderer (Camera *c, Scene * sc, Image * i, Shader *shd):  
    cam(c), scene(sc), img(i), shd(shd) {}  
    virtual void Render () {}  
};
```

Standard Renderer (StandardRenderer.cpp)

```
void StandardRenderer::Render () {  
    int W=0,H=0;  // resolution  
  
    // get resolution from the camera  
    // ...  
  
    // main rendering loop: get primary rays from the camera  
    for (y=0 ; y< H ; y++) {  // loop over rows  
        for (x=0 ; x< W ; x++) {  // loop over columns  
  
            // for each pixel x,y  
  
        } // loop over columns  
    }   // loop over rows  
}
```

Standard Renderer (StandardRenderer.cpp)

```
// for each pixel x,y

Ray primary;
Intersection isect;
bool intersected;
RGB color;

// Generate Ray (camera)

// trace ray (scene)

// shade this intersection (shader)

// write the result into the image frame buffer (image)
img->set(x,y,color);
```

main.cpp

```
int main(int argc, const char * argv[]) {  
    Scene scene;  
    Perspective *cam; // Camera  
    ImagePPM *img;    // Image  
    Shader *shd;  
  
    // Load scene ...  
  
    // add an ambient light to the scene  
    AmbientLight ambient(0.9,0.9,0.9);  
    scene.lights.push_back(ambient);  
    scene.numLights++;  
  
    // ...  
}
```

main.cpp

```
int main(int argc, const char * argv[]) {  
    // ...  
  
    // Create Image ...  
    // Create Camera ...  
    // Create Shader ...  
    // declare the renderer  
    StandardRenderer myRender (cam, &scene, img, shd);  
    // render  
    myRender.Render();  
  
    // save the image  
    img->Save("MyImage.ppm");  
  
}
```

main.cpp

```
int main(int argc, const char * argv[]) {
    Scene scene;
    Perspective *cam; // Camera
    ImagePPM *img;    // Image
    Shader *shd;

    // Load scene

    // add an ambient light to the scene
    AmbientLight ambient(0.9,0.9,0.9);
    scene.lights.push_back(ambient);
    scene.numLights++;

    // Image resolution
    const int W= 640;
    const int H= 480;
```

main.cpp

```
int main(int argc, const char * argv[]) {
    Scene scene;
    Perspective *cam; // Camera
    ImagePPM *img;    // Image
    Shader *shd;

    // Load scene

    // add an ambient light to the scene
    AmbientLight ambient(0.9,0.9,0.9);
    scene.lights.push_back(ambient);
    scene.numLights++;

    // Image resolution
    const int W= 640;
    const int H= 480;
```