Ray Tracing from the Ground Up

Notes on Texturing

1. Texture mapping and transformed objects

Although texture maps can only be applied to generic objects, for example unit spheres at the origin, we still have some flexibility in the way we can write the build functions. Listing 1 shows the part of the build function for Figure 29.2 that constructs the Earth sphere. Here, the Earth image texture is applied to a generic sphere, which is then transformed.

```
Sphere* sphere_ptr = new Sphere;
sphere_ptr->set_material(sv_matte_ptr3);

Instance* earth_ptr = new Instance(sphere_ptr);
earth_ptr->rotate_y(75);
earth_ptr->scale(3.0);
earth_ptr->translate(-11, 8, 12);
add_object(earth_ptr);
```

Listing 1

A shorter way of writing this is in Listing 2. In this version, the material is applied to the instance object. This also works for the following reason. Provided the instance is constructed with a *generic* object, the rays will ultimately be intersected with this generic object, and the texture color will be correctly rendered.

```
Instance* earth_ptr = new Instance(new Sphere);
earth_ptr->set_material(sv_matte_ptr3);
earth_ptr->rotate_y(75);
earth_ptr->scale(3.0);
earth_ptr->translate(-11, 8, 12);
add_object(earth_ptr);
```

Listing 2

I've used both techniques in the book.

2. Skydomes

A number of Figures in Chapter 11 use a skydome image with clouds. These include the page one image and Figure 11.7. In Chapter 11 this was just a textured background – not an environment light. These images were rendered using a large sphere, a spherical map, and a textured material. See Listing 3, which is an extract from the Figure 11.7 build function.

```
// image:

Image* image_ptr = new Image;
image_ptr->read_ppm_file("CloudsLargeWithBlack.ppm");

// spherical map:

SphericalMap* spherical_map_ptr = new SphericalMap;

// image textute:

ImageTexture* image_texture_ptr = new ImageTexture(image_ptr);
image_texture_ptr->set_mapping(spherical_map_ptr);

// spatially varying material:

SV_Matte* sv_matte_ptr2 = new SV_Matte;
sv_matte_ptr2->set_ka(1.0);
sv_matte_ptr2->set_kd(0.85);
sv_matte_ptr2->set_cd(image_texture_ptr);

// large sphere:

Instance* sphere_ptr1 = new Instance(new Sphere);
sphere_ptr1->scale(1000000.0);
sphere_ptr1->set_material(sv_matte_ptr2);
add_object(sphere_ptr1);
```

Listing 3

With a spherical map, the skydome image must have an aspect ratio of 2:1 for mapping onto the sphere, where the bottom half of the image is black. See Figure 1.

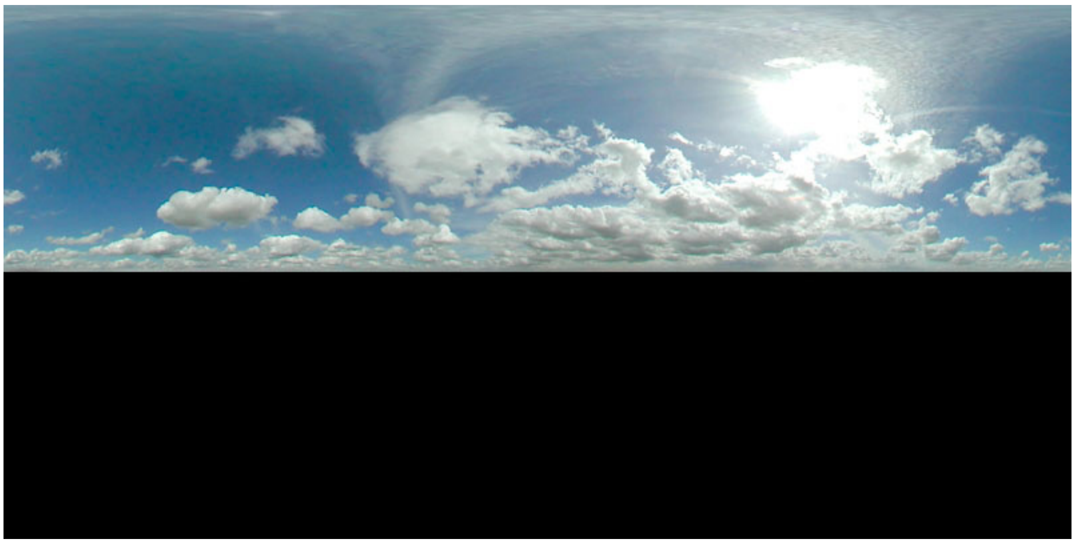

Figure 1 The skydome image used in Figure 11.7.

Because all the figures in Chapter 11 that use the skydome have a ground plane, there's another way of doing this. We can use a hemispherical map and a texture image that only has the clouds. Listing 2 has the relevant code.

```
Image* image_ptr = new Image;
image_ptr->read_ppm_file("CloudsLarge.ppm");

HemisphericalMap* hemispherical_ptr = new HemisphericalMap;

ImageTexture* image_texture_ptr = new ImageTexture(image_ptr);
image_texture_ptr->set_mapping(hemispherical_ptr);

SV_Matte* sv_matte_ptr2 = new SV_Matte;
sv_matte_ptr2->set_ka(1);
sv_matte_ptr2->set_kd(0.85);
sv_matte_ptr2->set_cd(image_texture_ptr);

Instance* sphere_ptr = new Instance(new Sphere);
sphere_ptr->scale(1000000.0);
sphere_ptr->set_material(sv_matte_ptr2);
add_object(sphere_ptr);
```

Listing 4

The texture image, which now has an aspect ratio of 4:1, appears in Figure 2.

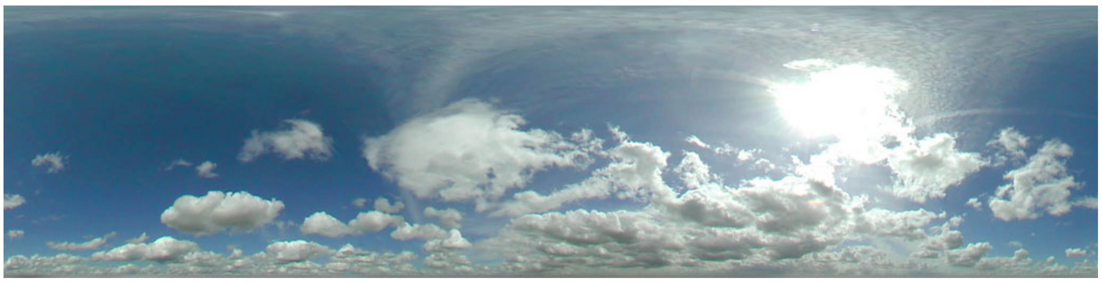

Figure 2

Note that in Listing 4 I've still used the sphere. We can use a sphere, a convex hemisphere, or a convex part sphere set up as a hemisphere. We can't use a concave object, because the texture is only shaded with ambient illumination, and the scene contains a point light with no distance attenuation. The normal therefore needs to point outwards so that there is no diffuse illumination on the inside surface of the sphere. As this is contrived, the most natural way to render these images is with a spatially varying emissive material, and a concave object. This is illustrated in Listing 5.

```
HemisphericalMap* hemispherical_ptr = new HemisphericalMap;

ImageTexture* image_texture_ptr = new ImageTexture(image_ptr);
image_texture_ptr->set_mapping(hemispherical_ptr);

SV_Emissive* sv_emissive_ptr = new SV_Emissive;
sv_emissive_ptr->set_ce(image_texture_ptr);
sv_emissive_ptr->scale_radiance(1.0);

Instance* hemisphere_ptr = new Instance(new ConcaveHemisphere);
hemisphere_ptr->scale(1000000.0);
hemisphere_ptr->set_material(sv_emissive_ptr);
add_object(hemisphere_ptr);
```

Listing 5

The figures in Chapter 11 and in the ray traced images download were rendered with a low resolution 800 x 200 pixel cloud image. This resulted in obvious pixelation. (I forgot that I had a high resolution image.) These images are also CMYK, which affects the colors of the boxes. I've therefore re-rendered Figure 11.7 with a 2024 x 512 pixel cloud image, and included RGB versions in the Chapter 29 download. The low resolution cloud image is now 1024 x 256 pixels. I haven't had time to re-render any of the other figures that use the clouds. The texture files download has the high and low resolution files with and without the black bottom halves.

3. Textured environment lights

I originally had a section in Chapter 29 on textured environment lights, where I used Figure 29.27 as an example. Just before the book went to press I discovered that there was no mechanism in the shading architecture for shading with textured environment lights, and so I set this as an exercise.

For ambient and diffuse shading with an environment light, we distribute shadow rays in a cosine distribution about the normal at the hit point. To compute the texel coordinates for each ray, we need to compute where the ray hits the unit hemisphere that has the skydome image applied. A shadow ray will only hit the hemisphere if it doesn't hit any other objects in the scene. There are two problems here. The first is that the `shadow_hit` functions don't compute the local hit point coordinates; the second is that the large skydome hemisphere (or sphere) must have shadows set to false for ambient occlusion to work. Figure 3 shows the function call sequence for shading a matte material with an environment light. For this to work, the `ShadeRec` object passed into the `ImageTexture::get_color` function in Listing 29.7 must have the local hit point coordinates stored in it.

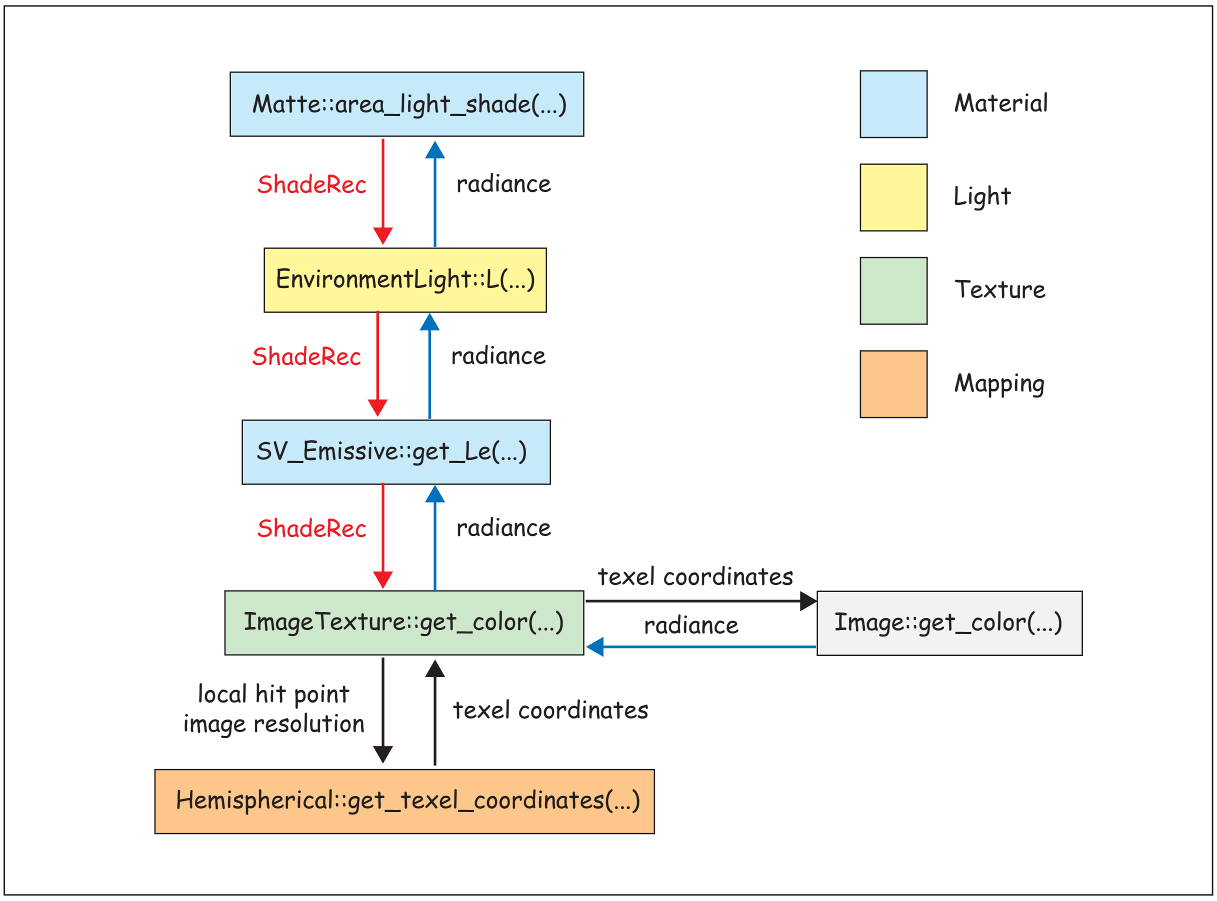

Figure 3

Fortunately, this is simple to accomplish. The only change we have to make to the shading code is to add the single statement `sr.local_hit_point = wi` to the `EnvironmentLight::get_direction` function in Listing 18.5. The revised function is in Listing 6.

```
Vector3D
EnvironmentLight::get_direction(ShadeRec& sr) {
    w = sr.normal;
    v = Vector3D(0.0034, 1, 0.0071) ^ w;
    v.normalize();
    u = v ^ w;
    Point3D sp = sampler_ptr->sample_hemisphere();
    wi = sp.x * u + sp.y * v + sp.z * w;
    sr.local_hit_point = wi;

    return (wi);
}
```

Listing 6

Because the new statement assigns a vector to a point, we also need to add the member function in Listing 7 to the `Point3D` class.

```
Point3D&
Point3D::operator= (const Vector3D& rhs) {
    x = rhs.x; y = rhs.y; z = rhs.z;
    return (*this);
}
```

Listing 7

To test that this works, the scene discussed below consists of a sphere and plane with the same gray matte material, shaded with ambient occlusion and an environment light with the texture image shown in Figure 4.

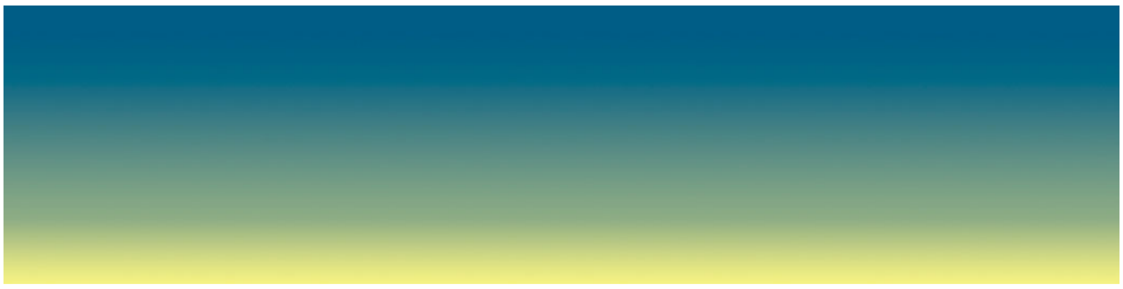

Figure 4 The texture file CyanToYellow.

Figure 5(a) shows a horizontal view of the scene, where the sphere has a slight yellow tint. This is caused by the shadow rays hitting the skydome near the horizon. A test for correctness is that the top of the sphere should be the same color as the plane, for points that are far enough away from the sphere as not to be affected by the ambient occlusion. Figure 5(b) is a top-down view where the top of the sphere appears to be lighter than the plane, but it's not. This is a contrast illusion.

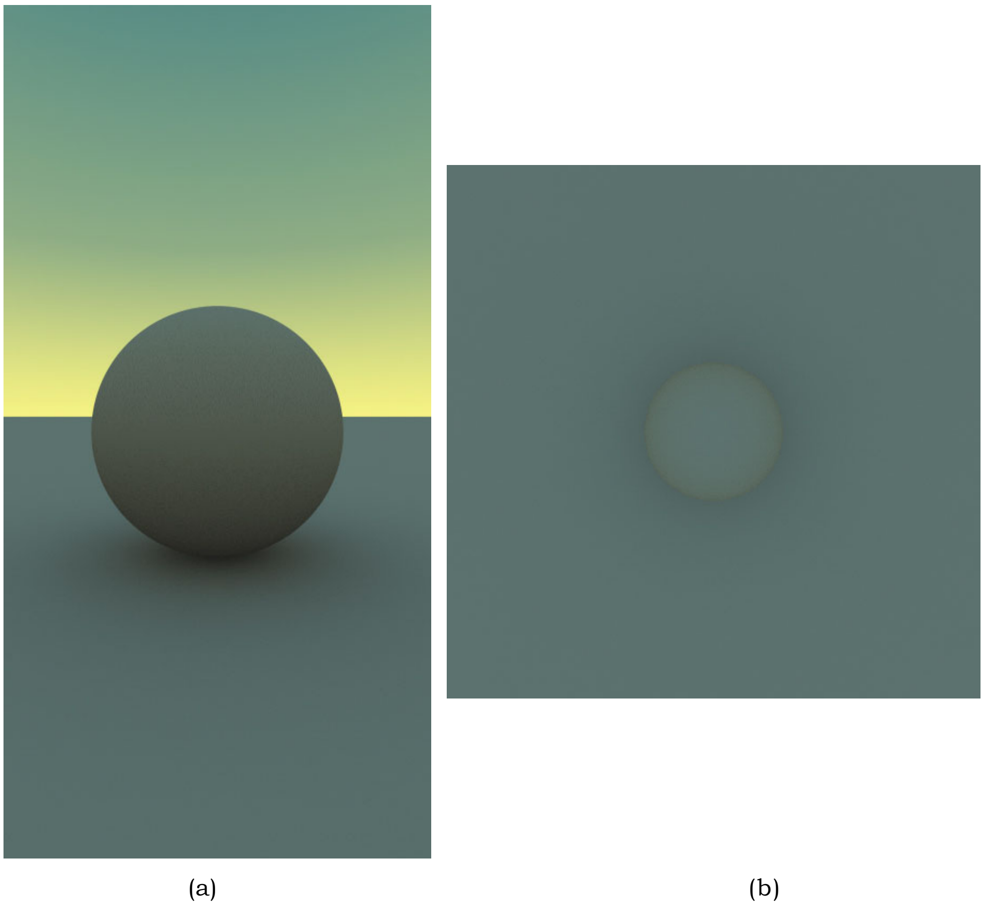

Figure 5

To reinforce this, Figure 6(a) shows a top-down view of a solid cylinder and a plane. Here, the top of the cylinder is a constant color identical to the plane, but again, it looks lighter. Figures 6(b)&(c) show the process of cutting and pasting part of the plane onto the top of cylinder, where it disappears.

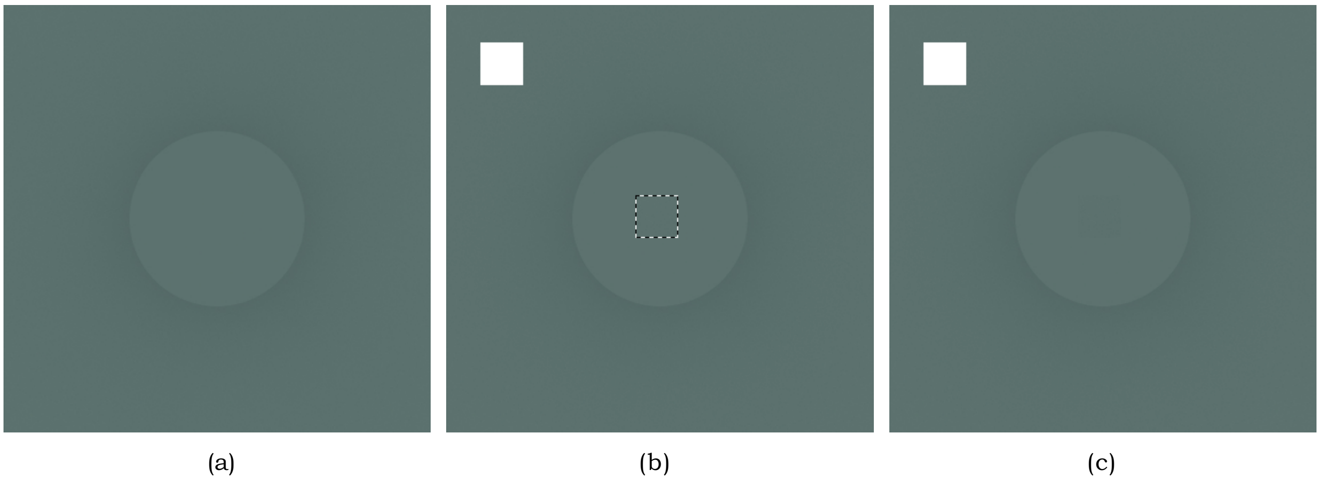

Figure 6

The Chapter 29 download contains Figures 5 and 6(a), and their build functions.

In view of the above discussion, the images in Figure 29.27 are incorrectly shaded. The correct images appear in Figure 7. Here, the shading from the environment light is not as strong as in the original images.

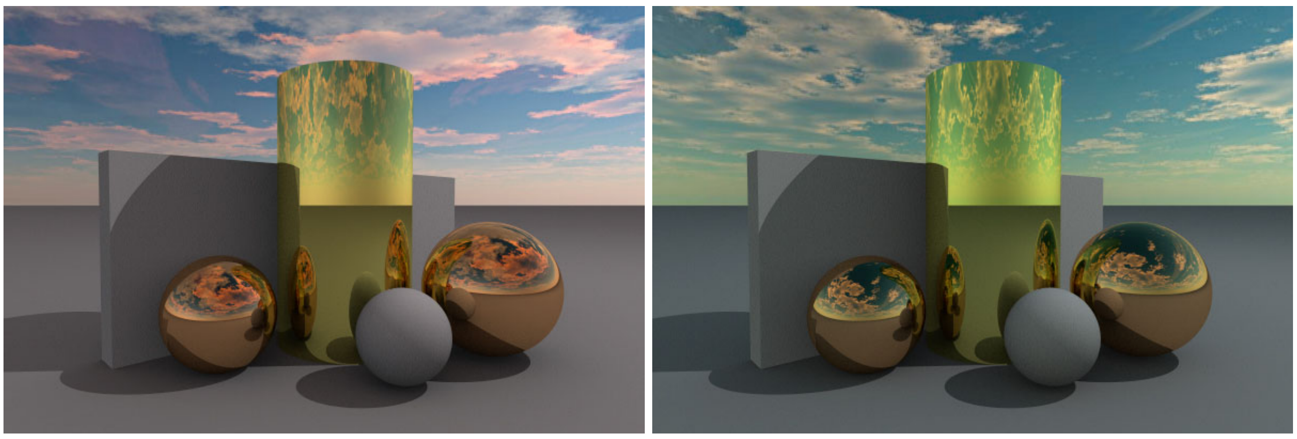

Figure 7 Corrected Figure 29.27.

The direct rendering of the skydome images with primary and reflected rays was always correct because the `hit` functions always return the local hit point coordinates.