The Technical University of Denmark

02562 Rendering - Introduction

Assignment 3

Written by

Anders Bo Sørensen s125010

Supervisor

Jeppe Revall Frisvad

Hand-in date

October 4, 2017

1 Results

Q: Explain how the framework finds out what image file to load for a given object. Use the plane in the default scene as an example in your explanation. The default scene is loaded in the function load_files of the file RenderEngine.cpp.
A: If no input arguments are given to *load_files* the plane will be defined by the following call:

```
scene.add_plane(make_float3(0.0f, 0.0f, 0.0f),
    make_float3(0.0f, 1.0f, 0.0f),
    "../models/default_scene.mtl", 1, 0.2f);
```

The function *add_plane* takes the arguments:

```
add_plane(const float3& position, const float3& normal,
    const string& mtl_file, unsigned int idx, float tex_scale)
```

Thus, the file name is hardcoded to $../models/default_scene.mtl$ and then passed to *add_plane*. Inside *add_plane* the following happens:

```
  if(!mtl_file.empty())
    mtl_load(mtl_file, m);
  if(m.size() == 0)
          m.push_back(ObjMaterial());
  Plane* plane = new Plane(position, normal, idx < m.size() ? m[idx] : m.back(), tex_scale);
  planes.push_back(plane);
```

Here we can see if the file is not empty it is loaded as a new object material.

Q: Explain how texture colour is used in a rendering. Look at the functions in the file Textured.cpp.
A: A hit is passed to *get_emission*. If the hit does not have a texture *get_emission* in *Emission.cpp* is called. If it does have a texture a material is taken from the hit. If the material is not there $0.2f$ is returned. Otherwise the material's ambient values are put into a $float3$ named *emission*. If the material does not have a texture *emission* is returned, otherwise x, y and z of *emission* are each divided by that material's diffusion value and have their minimum value clamped to 0, as no material returns negative light. This emission vector is multiplied by what is supposed to be bilinear sampling of a texture lookup, however in the given state, it merely returns $0.4f$.
A hit can also be passed to the $get\_diffuse$ function. This function is a lot like *get_emission*, however after extracting the material, if there is no texture it merely returns the diffuse values of the material and if there is a texture it does bilinear sampling of the texture.

Q: Compute texture coordinates for planes. Do this in the function get_uv of the file Plane.cpp. Compute the texture coordinates by finding the vector from the plane origin (position) to the intersection point and projecting it onto the tangent and binormal of the plane, respectively (this is an inverse mapping). Use the texture scaling factor (tex_scale) to scale the texture coordinates. In the function intersect, use material.has_texture to find out whether a texture was loaded for a plane. If yes, call the function get_uv to get the texture coordinates.
A: *get_uv* in Plane.cpp

```
        u = hit_pos.x*tex_scale + 0.5f;
        v = -hit_pos.z*tex_scale + 0.5f;
```

The addition to *intersect* in Plane.cpp

```
        float u, v;
        if (&material.has_texture)
        {
                get_uv(hit.position, u, v);
                hit.texcoord = make_float3(u, v, 0.0f);
        }
```

Q: Press 'x' on the keyboard while the render program is running to switch on textures. If the function get_uv was implemented, a texture should appear on the plane in the preview when 'x' is pressed. Take a screenshot of the preview
A:

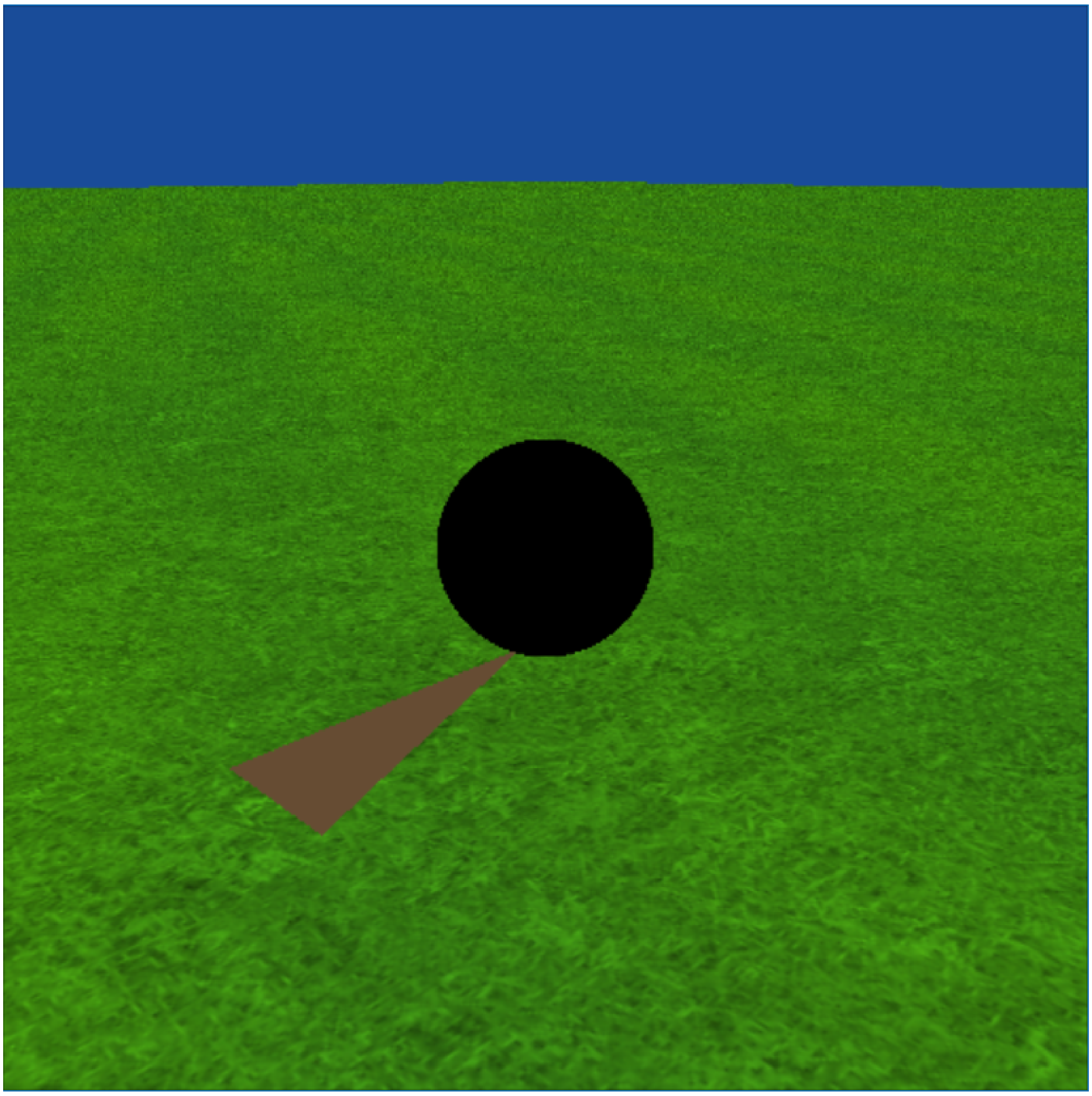

Figure 1: Texture preview.

Q: Implement the texture look-up function sample$_n$earest in the file Texture.cpp. After being loaded, the texture is available in a flat array of 4-vectors (float4) called fdata. To make a look-up into the texture, we need to map a given set of texture coordinates to an index into the loaded texture array. We want the texture to repeat itself in each unit square in texture space. Use this information to find (u, v)-coordinates in $[0, 1]^2$ *that point out the position in texture space where we would like to look-up the texture colour. The texture resolution (width, height) specifies how many texels (texture elements) each unit square in texture space is divided into. Map the (u, v)-coordinates to the 2D index (U, V) of the nearest texel. Map (U, V) to an index into the texture array, but please take into account that the texels in the v-direction are reversed in the texture array. Return the 4-vector in the texture array at the computed index position*
A:

The following code was written *sample_nearest* in *Glossy.cpp*

```
int u = floor((texcoord.x - floor(texcoord.x))*(width - 1) + 0.5f);
int v = floor((1 - texcoord.y - floor(texcoord.y))*(height - 1) + 0.5f);

return fdata[u + v*width];
```

Q: Render the default scene using base colours (press '0' on the keyboard to choose this default shader)

and using the Lambertian shader (press '1'). Use gamma correction with the Lambertian shader (press '' after rendering) to ensure that the images are not too dark. Compare these results to the preview.*
A:

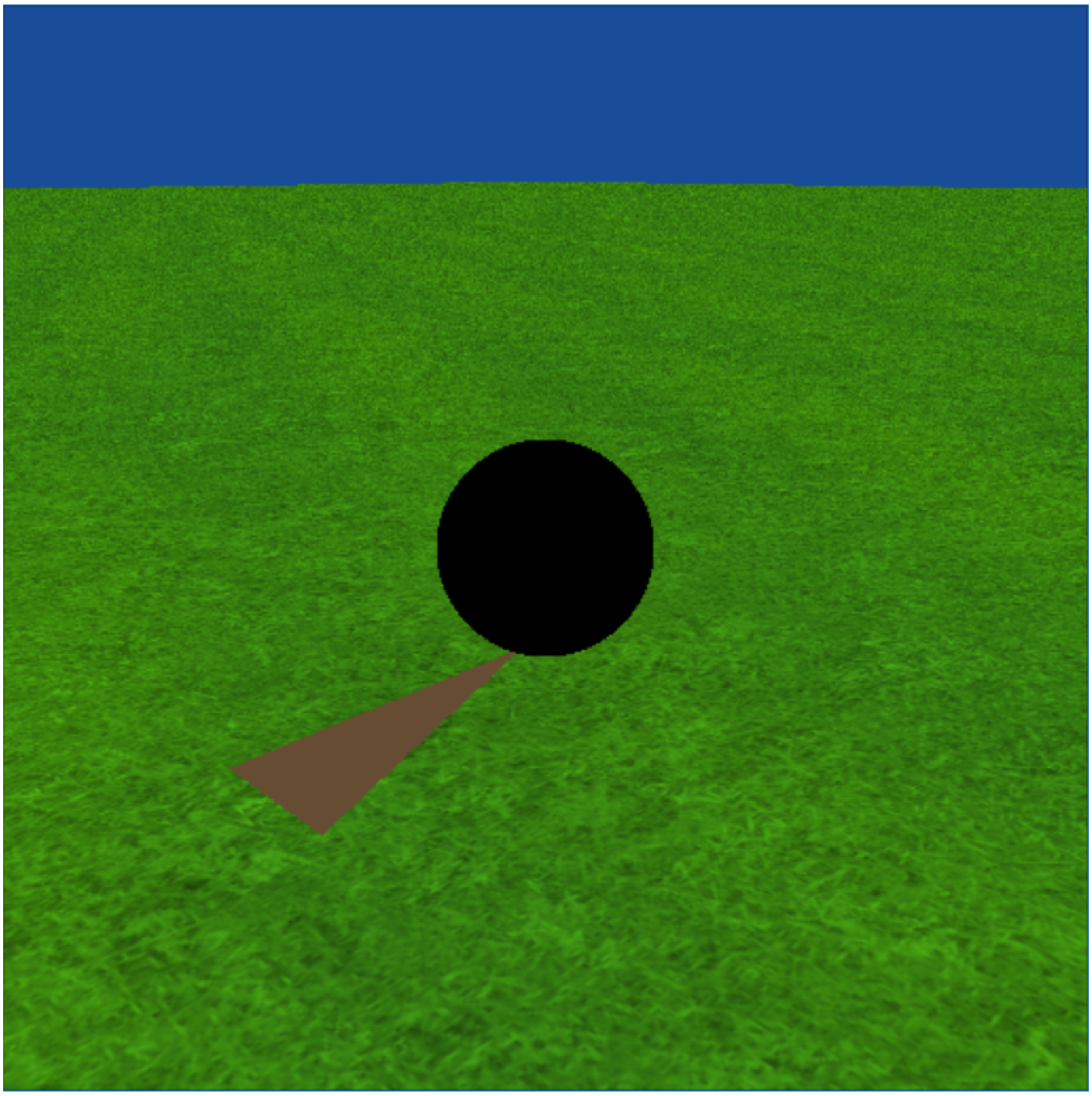
Figure 2: Default shader preview

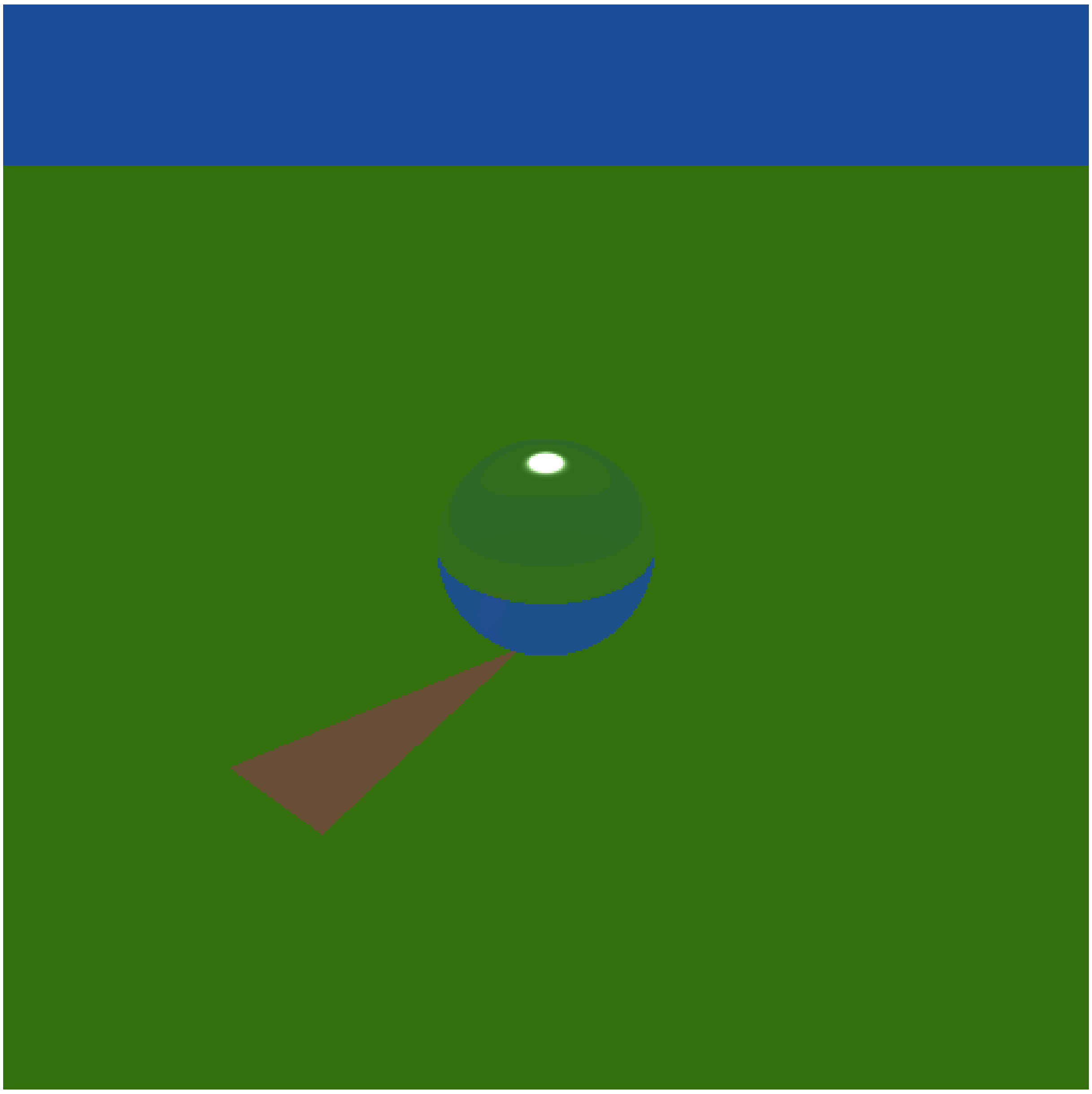

Figure 3: Default shader render

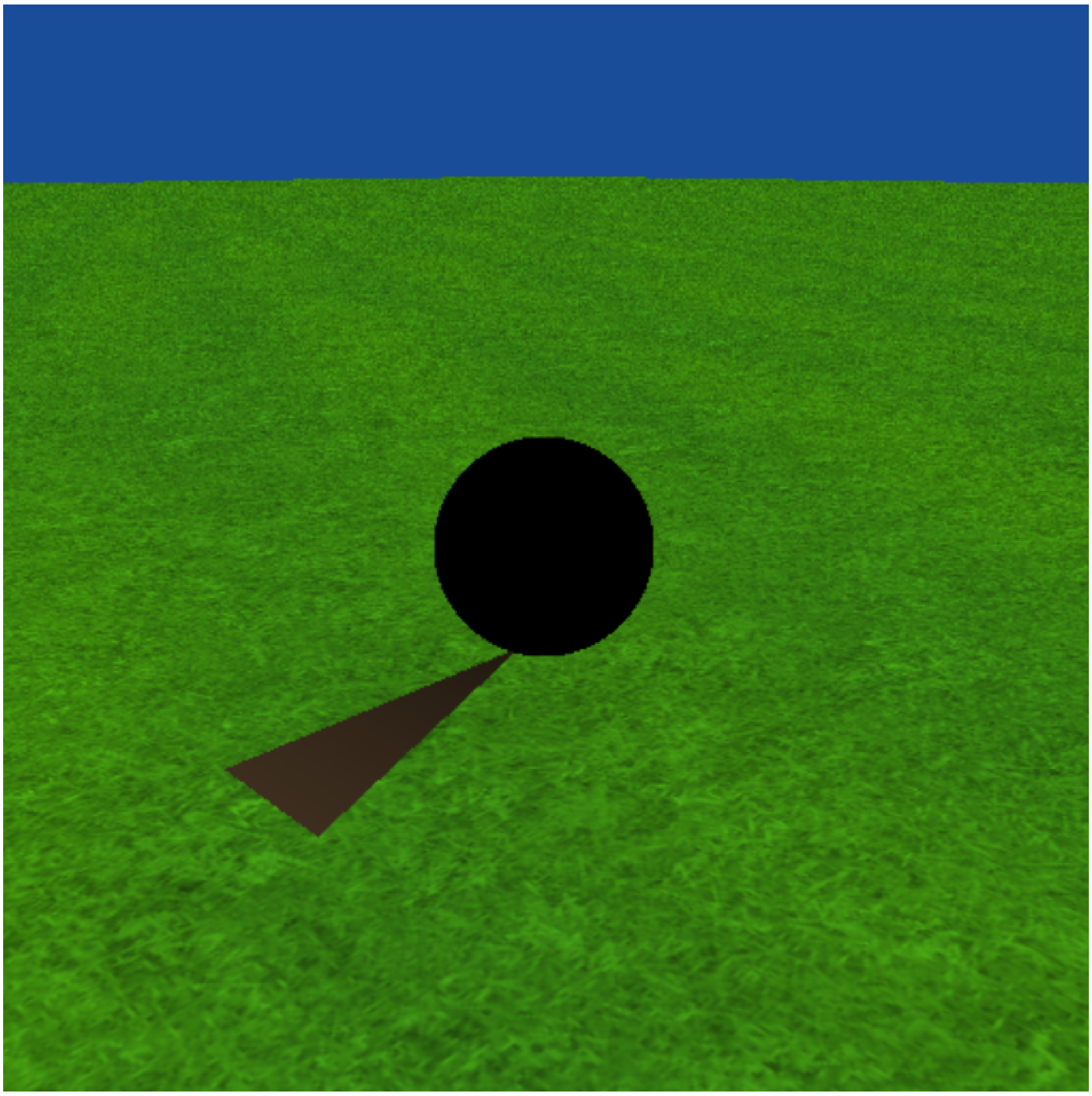
Figure 4: Lambertian shader preview

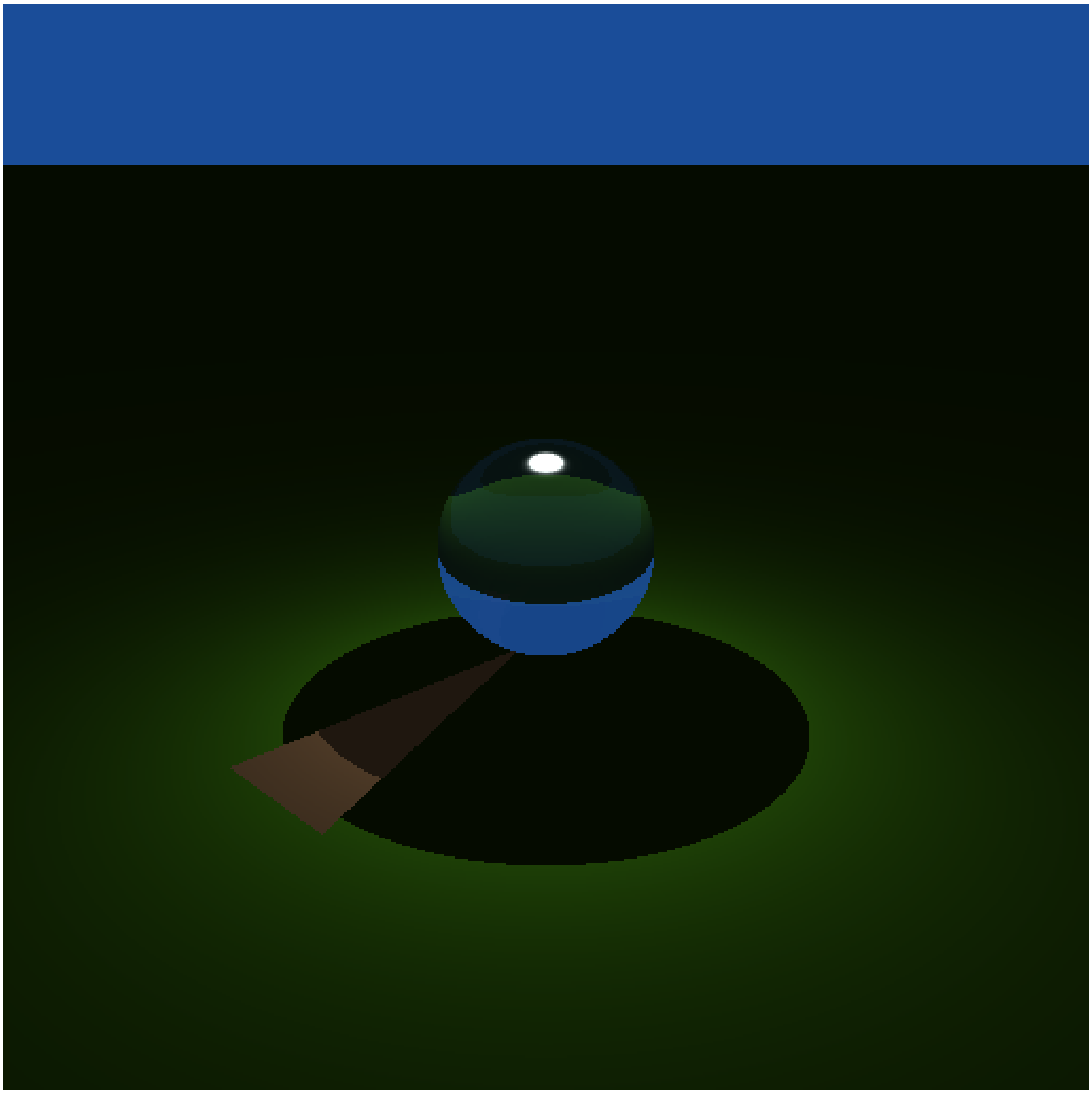

Figure 5: Lambertian shader render

Q: Instead of looking up the nearest texel, look up the four nearest texels and use bilinear interpolation to find the texture colour that best represents a given set of texture coordinates. Do this by implementing the function sample_linear in the file Texture.cpp.
A: The following code was written *shade* in *Glossy.cpp*

```
float a, b;
float2 zerozero, onezero, zeroone, oneone;
float4 result1, result2;
a = (texcoord.x - floor(texcoord.x)) * width;
b = (texcoord.y - floor(texcoord.y)) * height;

zerozero = make_float2(floor(a), floor(b));
onezero = make_float2(fmodf(ceil(a), width), floor(b));
zeroone = make_float2(floor(a), fmodf(ceil(b), height));
oneone = make_float2(fmodf(ceil(a), width), fmodf(ceil(b), height));

result1 = lerp(fdata[(int)zerozero.x + (int)zerozero.y * width], fdata[(int)onezero.x + (int
result2 = lerp(fdata[(int)zeroone.x + (int)zeroone.y * width], fdata[(int)oneone.x + (int)one

return lerp(result1, result2, b - floor(b));
```

Q: Explain how scaling the texture coordinates affects the rendered texture. Use tex_scale (which is set in the function load_files of the file RenderEngine.cpp) to magnify the texture by a factor 10. Use renderings of the magnified texture to verify and compare your implementations of sample_nearest and sample_linear.
A: If scaling the texture coordinates upwards the texture, which might previously have had a one to one relation to a point in a texture and a hit point, might begin relegating several hit points to a single point in the texture.