Worksheet 3

If we compare rendered images to photographs, flatly coloured surfaces with no visual detail is one of the first giveaways that an image is a rendering. To obtain realism in rendered images, it is important to add visual detail. One way to add visual detail is to use texturing. This set of exercises is about how to do texturing in ray tracing.

Learning Objectives

- Use texture mapping (mapping an image to a surface) to heighten the level of visual detail.
- Compute texture coordinates for a plane.
- Investigate aliasing problems when texturing.
- Apply bilinear interpolation for texture magnification filtering.

Texture Mapping

Texture mapping is in four steps: (1) loading the texture image, (2) computing texture coordinates for the object to which the texture should be mapped, (3) looking up a colour in the texture image for a given set of texture coordinates, and (4) using the texture colour in a shader. The following exercises use the ray tracing render framework. Your job is to explain steps 1 and 4 and to implement steps 2 and 3.

- 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`.
- Explain how texture colour is used in a rendering. Look at the functions in the file `Textured.cpp`.
- 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.
- 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.
- Implement the texture look-up function `sample_nearest` 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. Somewhere in this uv to index transformation, 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.
- 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.

- 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`.
- 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`.

Worksheet 3 Deliverables

Renderings of the default scene with a textured plane (e.g. using look-up of nearest texel, bilinear magnification filtering, and using nearest texel or bilinear filtering with a 10-fold magnified texture). Also provide the explanations and comparisons mentioned above. Include relevant code snippets. Please insert all this into your lab journal.

Reading Material

The curriculum for Worksheet 3 is (25 pages)

B Sections 11-11.3. *Texture Mapping*.