Project 1: Deferred Shading

Mike Riches

CS 562, Summer 2015

May 31st, 2015

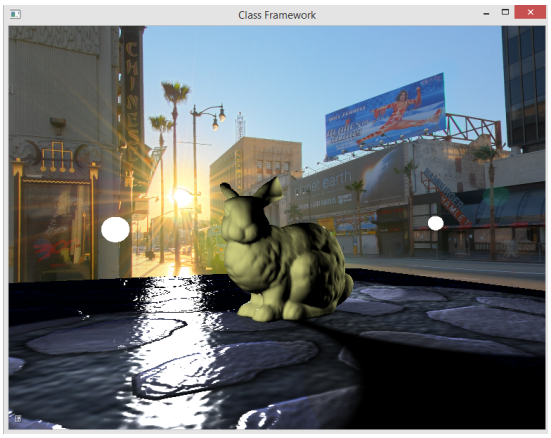


Table of Contents

Report

Project description

I started my project by continuing where I left off with Project 5 in last semester's CS 541 class. I expanded the framework to a multi-pass rendering framework by adding the capability to have more than one global light, adding an array of moving local lights, and writing the necessary g-buffer and light buffer code to enable multiple passes over the same scene. Along the way, I found some interesting ways of rendering local lights, and I figured out how to combine it with HDR and IBL.

I kept the functionality for image-based lighting by using a preprocessor definition to toggle between HDR and BRDF at compile-time. The sky box is also still present in the project, and I moved it to a better place in the pipeline to reduce overdraw – it's now drawn after all the other geometry is drawn, filling in the empty portions of the screen. This involved adding the Z-value trick to place the sky box behind everything, all the time. My project can also draw the light locations, which are represented as flat-lit globes; since the sky box and light globes each use flat lighting, I built a "flatLighting" flag into the multi-pass rendering framework, allowing drawing passes to skip those pixels when efficient. The reflection code has been removed for the time being.

Project info:

- Supports up to 32 local lights. (This is an arbitrary choice, it could probably do 64 or more.)
- Local lights have range-based brightness.
- Local lights don't contribute specular values by default, but this can be changed – see appendix.
- 3 global light positions are preprogrammed, with the capability of adding more later.
- Includes teapot, sphere, Buddha, bunny, dragon, and horse.
- Loads the same normal maps, etc. that I used last semester.

I added new debug options:

- Toggle light globes
- Toggle global lighting pass
- Toggle local lighting pass (many moving lights)
- New debug rendering modes for the G-buffer

What I learned

Here’s a list of the things I learned by working on this project:

- How to create a multi-pass rendering framework
- The differences between a single-pass, and a multi-pass rendering framework
- OpenGL specifics:
    - How to create a multi-buffer FBO, and render to it
    - How to create VBOs
    - How to use blending in OpenGL
- Graphics techniques:
    - How to create a light buffer with contributions from multiple lights, including lights that cast shadows
    - How to render ranged lights using a full-screen quad
    - How to limit light range using geometry, instead of a full-screen quad (i.e. get screen-space coordinates for looking up G-buffer data)
    - How to combine multiple BRDF sources with an ambient value to create the final light value at a pixel
    - Applying “HDR” techniques to limit the range of values of the summed light sources to [0, 1]
- Debugging techniques:
    - Displaying each of the values in the G-buffer
    - Adding toggles for each lighting pass
    - Adding multiple lights
    - Compiling a separate version that uses IBL instead of BRDF

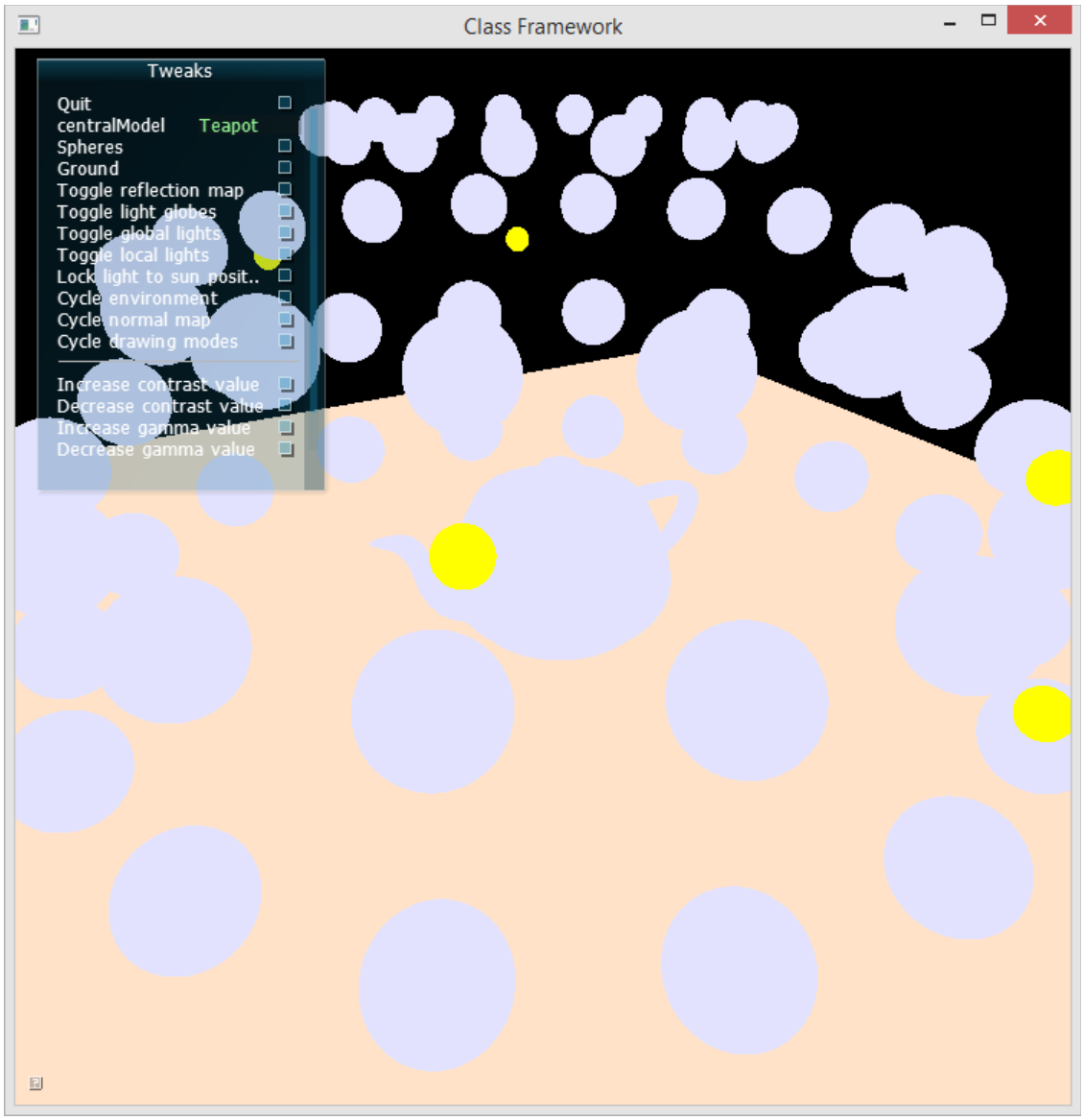
Required images

This section contains the images of my project’s output that are required for grading.

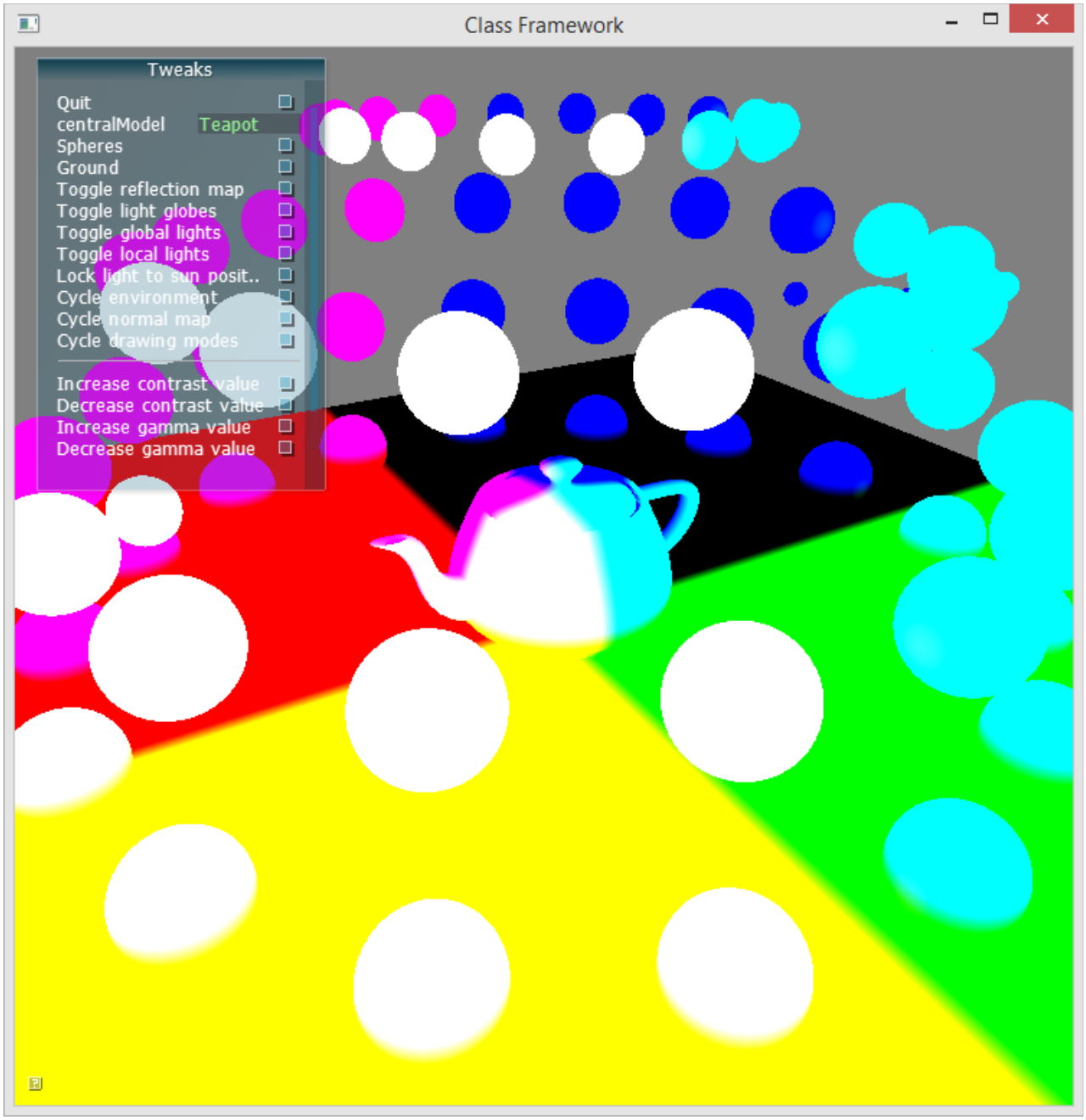
Images of each of the quantities computed and stored into the G-buffer

Here are pictures of the g-buffer values. Note that some values are mapped to a visible color range.

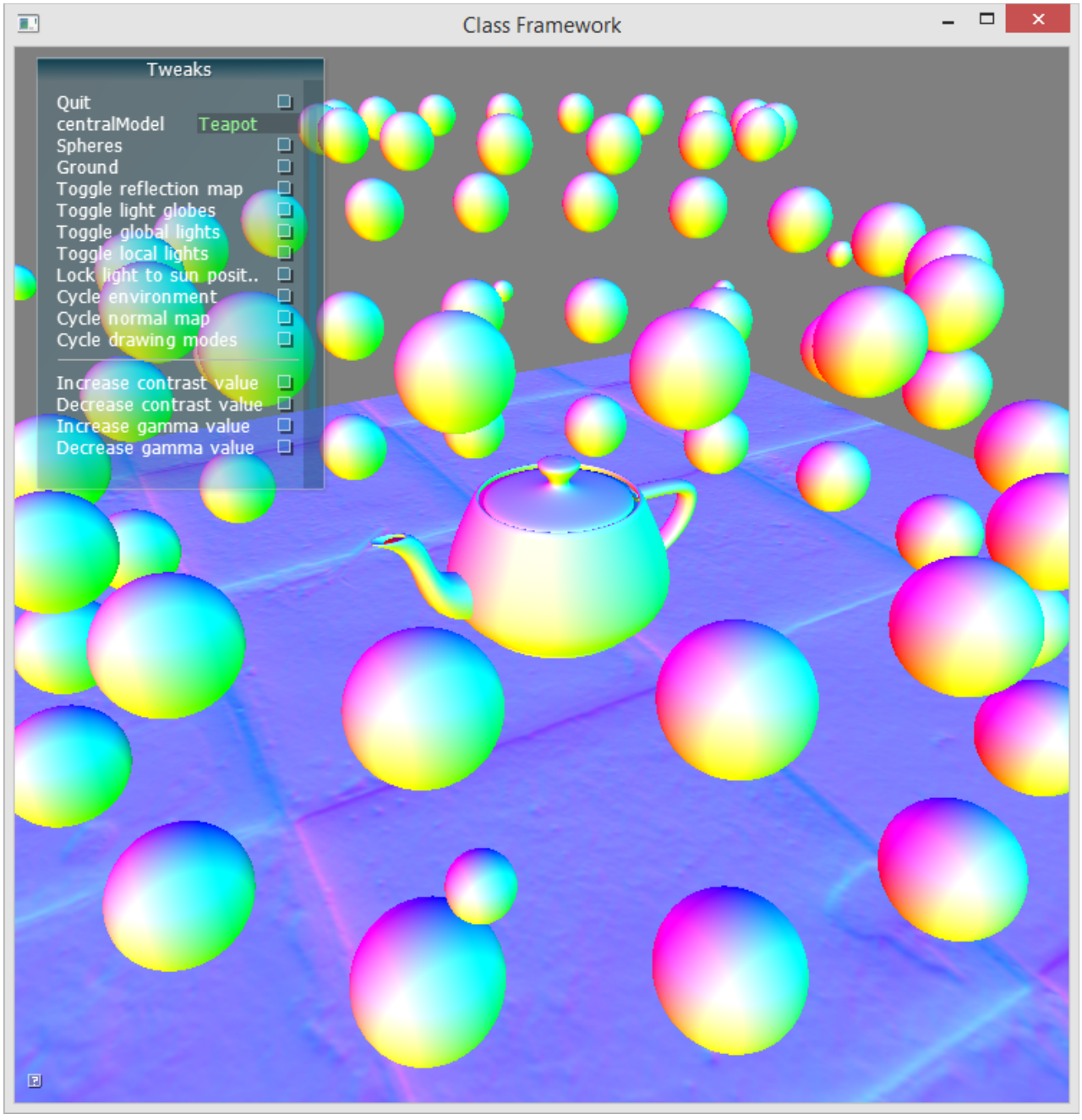
Pow(Specular color, 12):


Class Framework
Tweaks
Quit
centralModel
Teapot
Spheres
Ground
Toggle reflection map
Toggle light globes
Toggle global lights
Toggle local lights
Lock light to sun posit..
Cycle environment
Cycle normal map
Cycle drawing modes
Increase contrast value
Decrease contrast value
Increase gamma value
Decrease gamma value

(World vertex + 1) / 2:



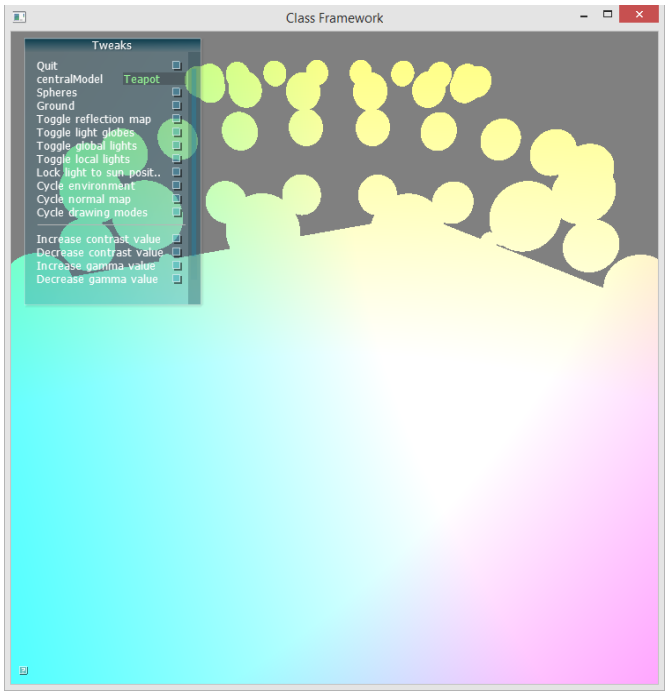
(Normal vector + 1) / 2:

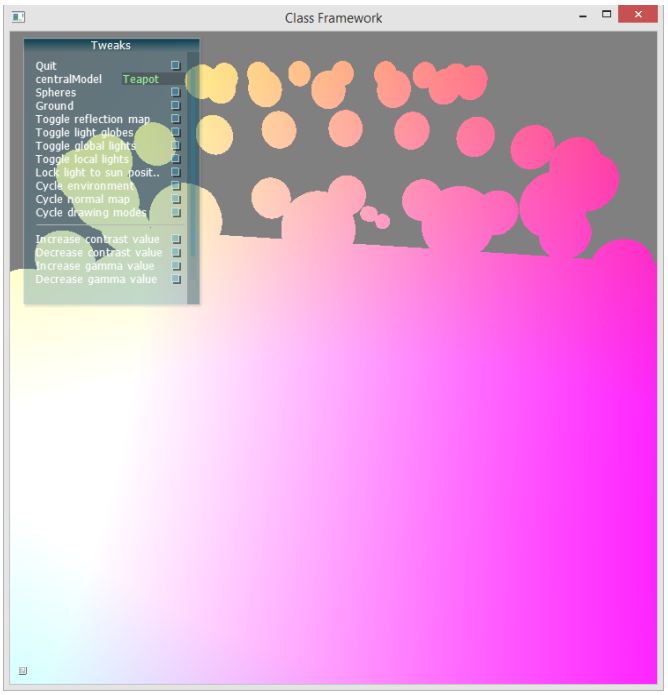


Diffuse color:

(Eye vector + 1) / 2

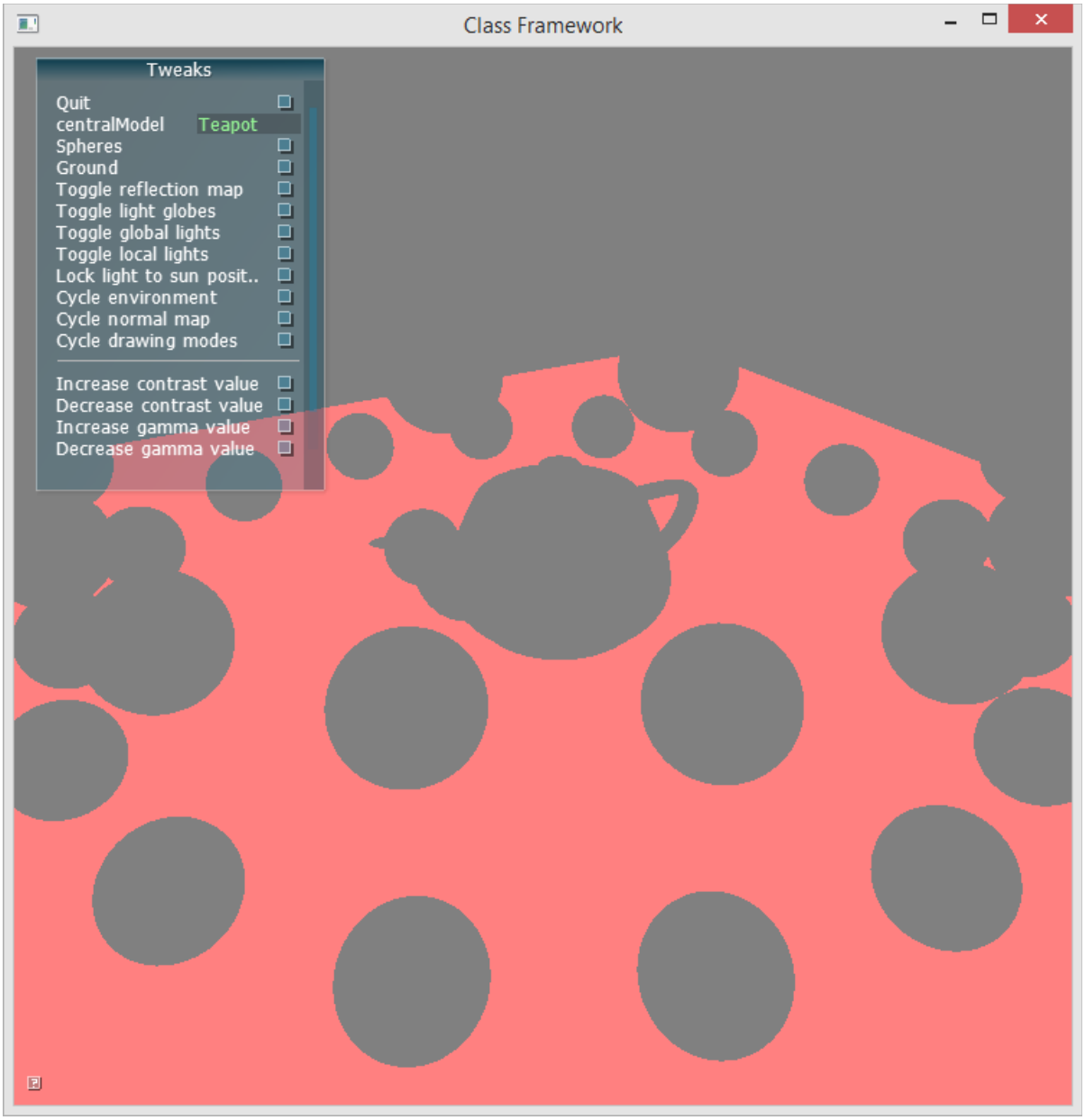
Since this value changes as you rotate the eye around, here are two separate images from different viewpoints:





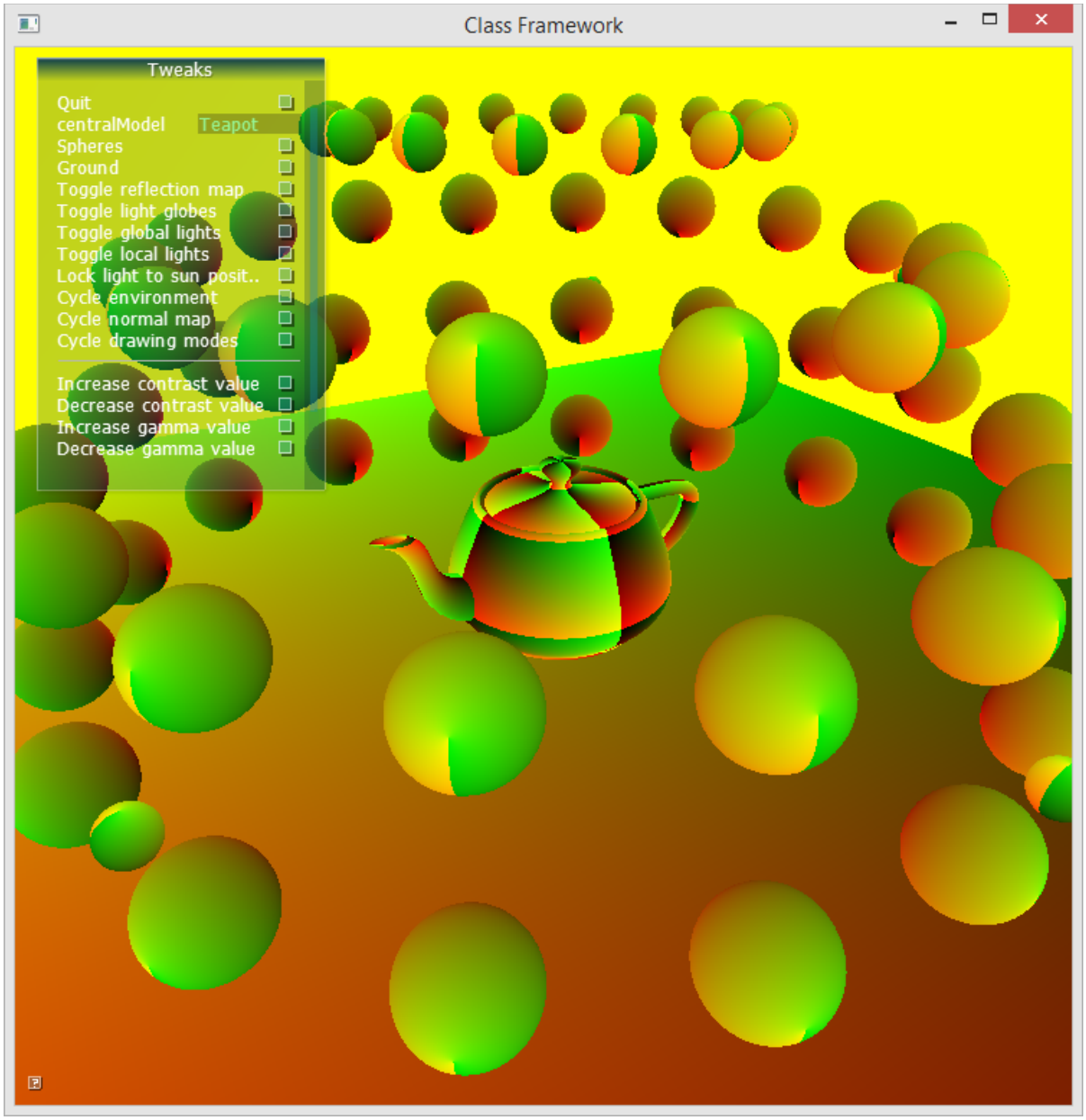
(Tangent + 1) / 2:

Only the ground has a tangent – nothing else uses that.

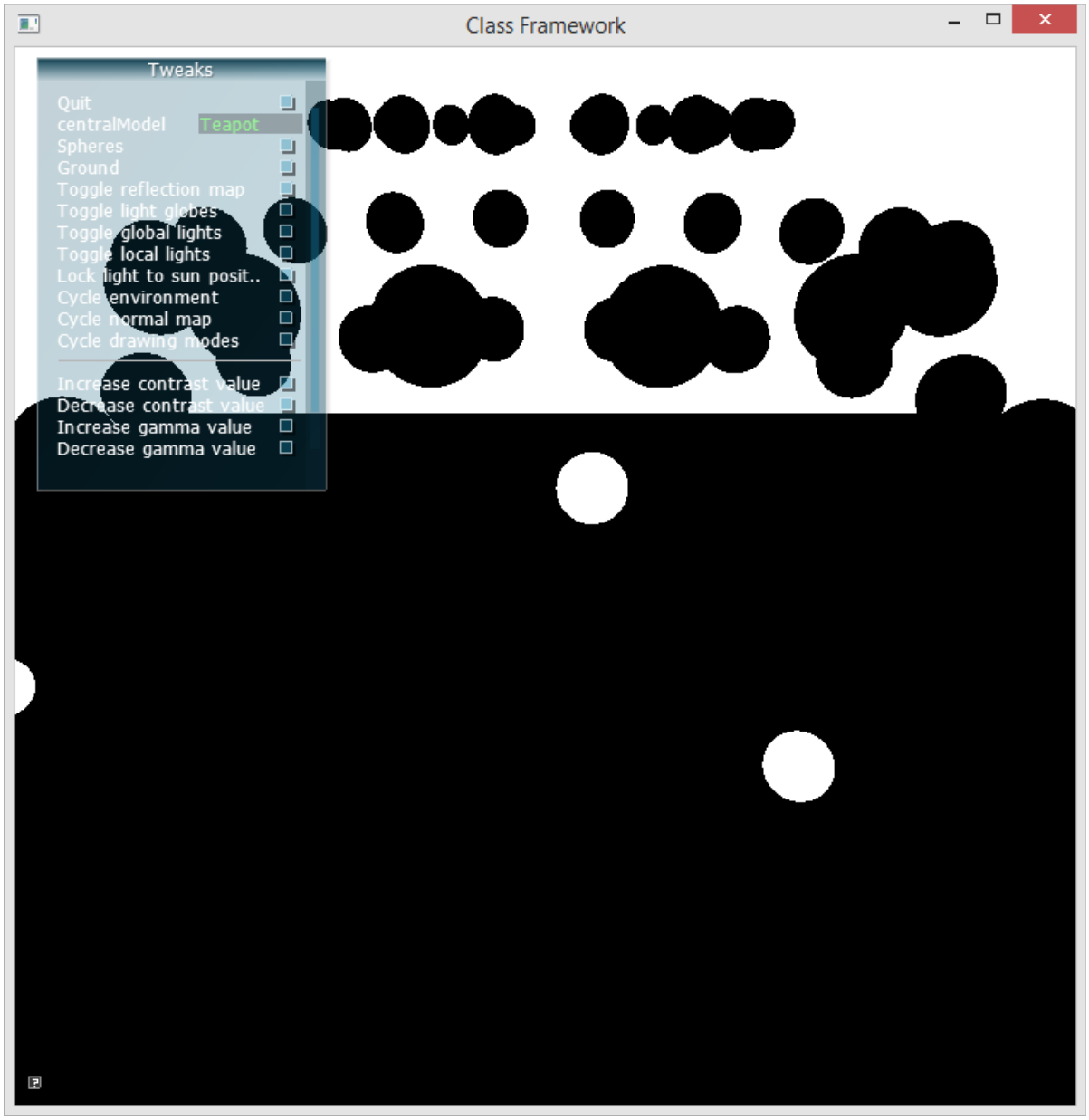


Phong shininess/ 1024:

Vec3(Texture coordinates, 0.f):

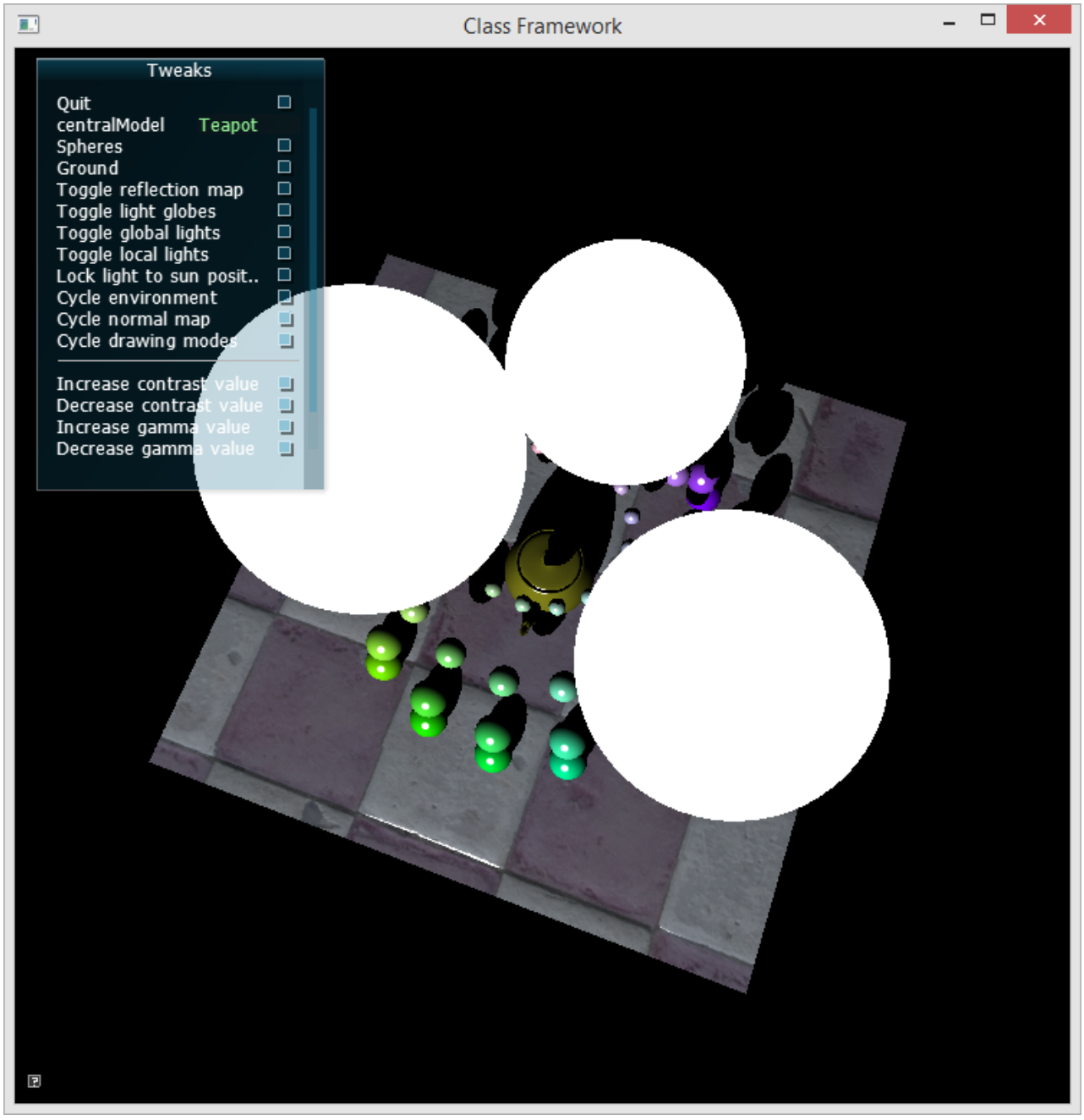


“flatLighting” toggle:



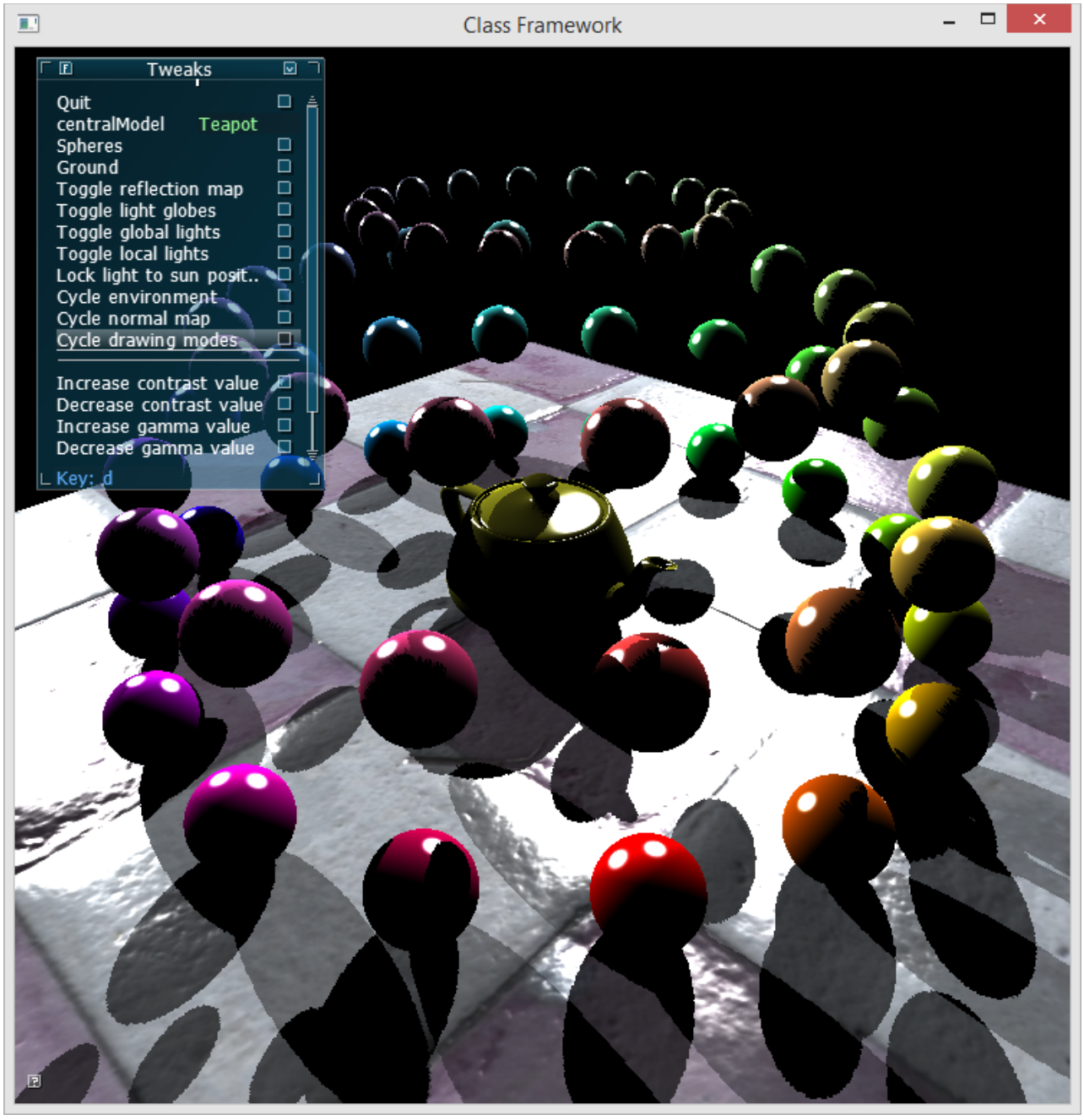
Some indication of how I determined the displayed buffers are correct

To determine that things are correct, I started by looking at debug output of each of the render targets in the G-buffer FBO, rendered as color data (as above). This let me see that the world position is correct, specular and diffuse color values are correct, etc. While building the local lighting pass, I also added a debug view to show me the local light geometry – that is, the range of each local light. This assured me that only the range across each light was being drawn on. This is best observed with just a few local lights:

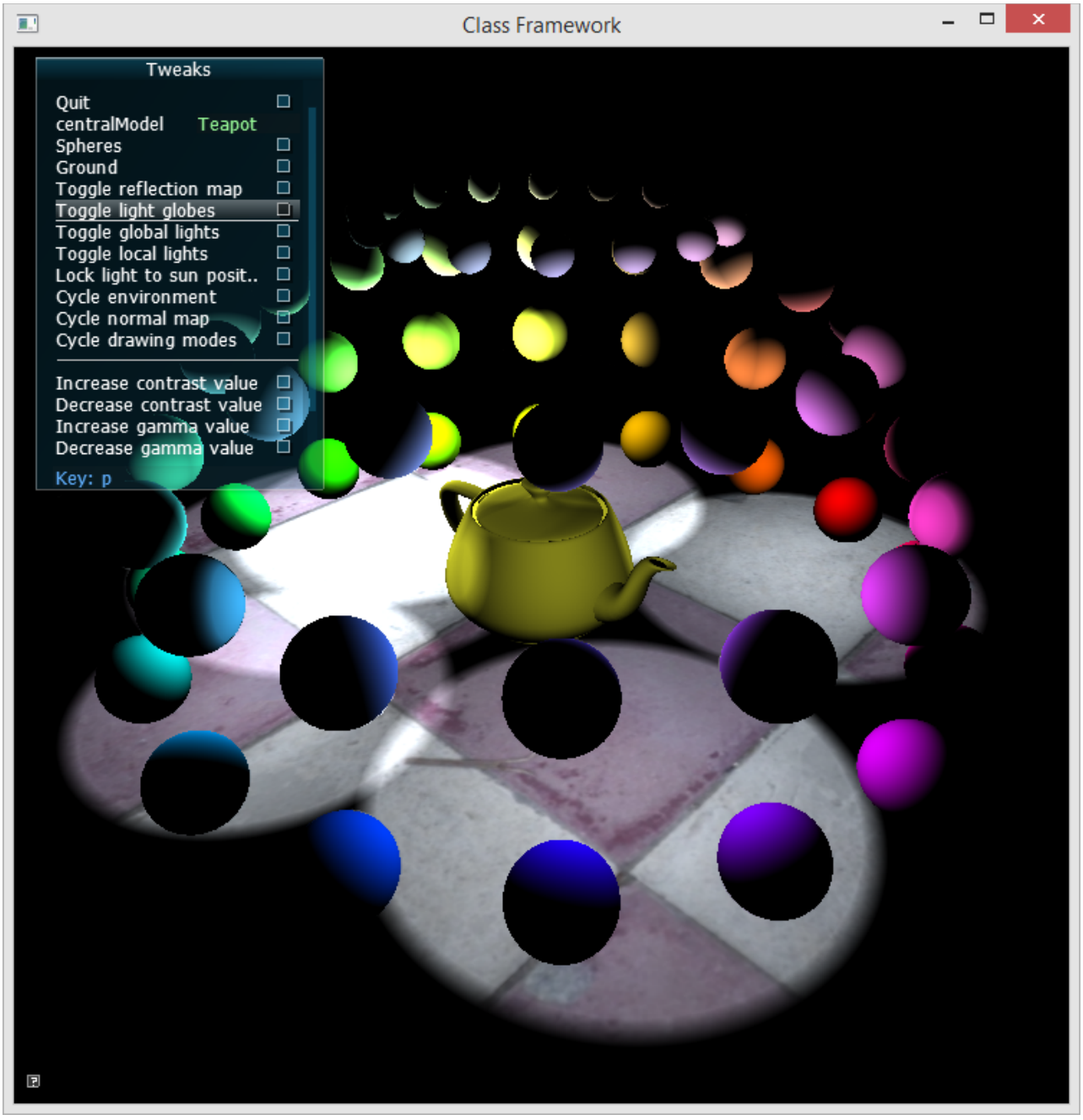


To put everything together, I created a debug mode to inspect the light buffer. My light buffer stores the result of the diffuse+ambient terms from each light before being sent to the final shader. Since my project includes the ability to toggle the global and local lighting passes, looking at the light buffer by itself allows me to inspect the contribution of each of those passes. I went through several iterations where I did not get correct output, proving the usefulness of this debug view. Following are images of correct output.

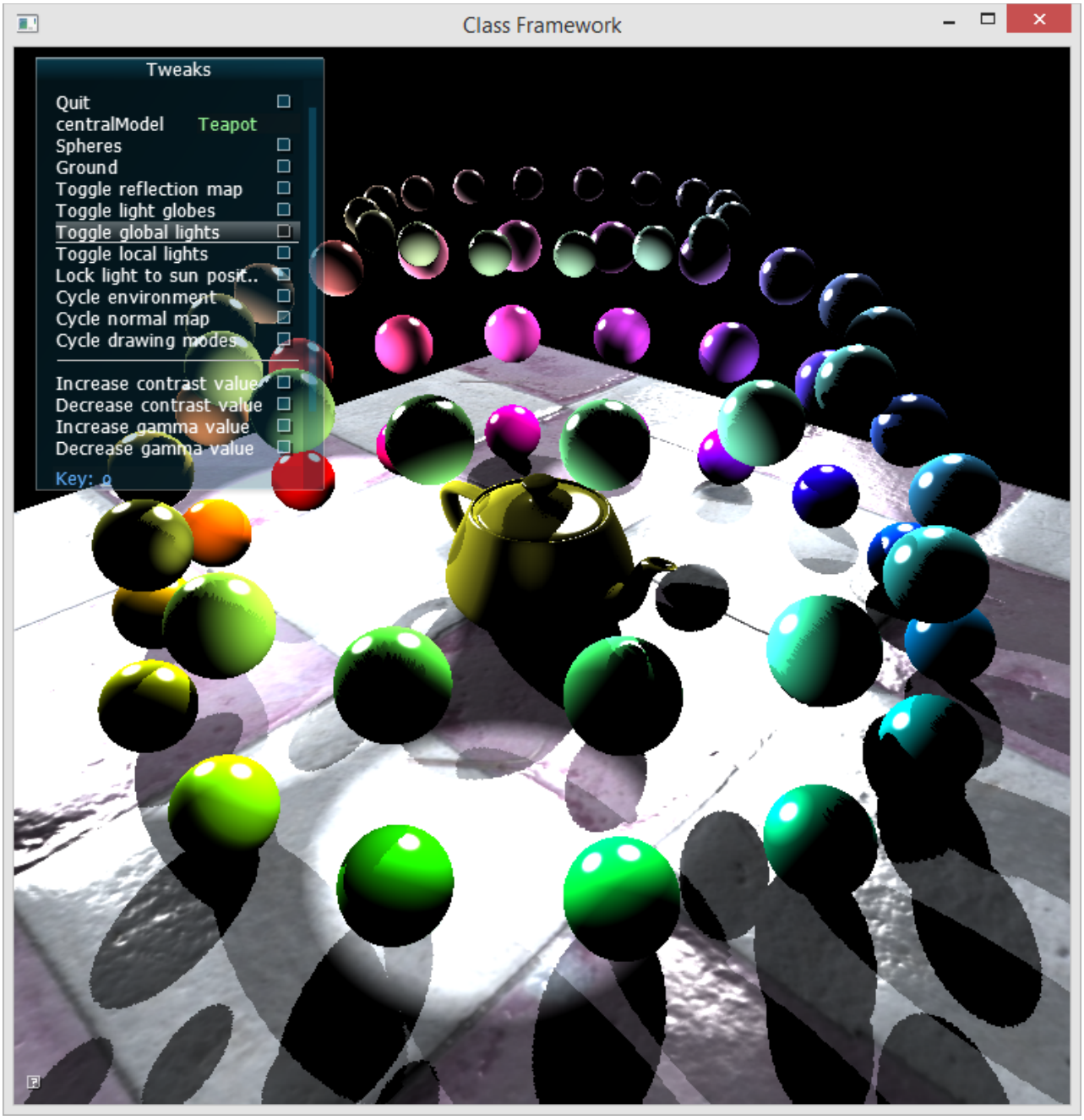
This is the output of the global lighting pass, showing two lights, each casting shadows and providing separate specular and diffuse contributions:



This is the same viewpoint, this time showing the contribution from the local lighting pass:



Here is the combined contribution of both lighting passes in the debug view:

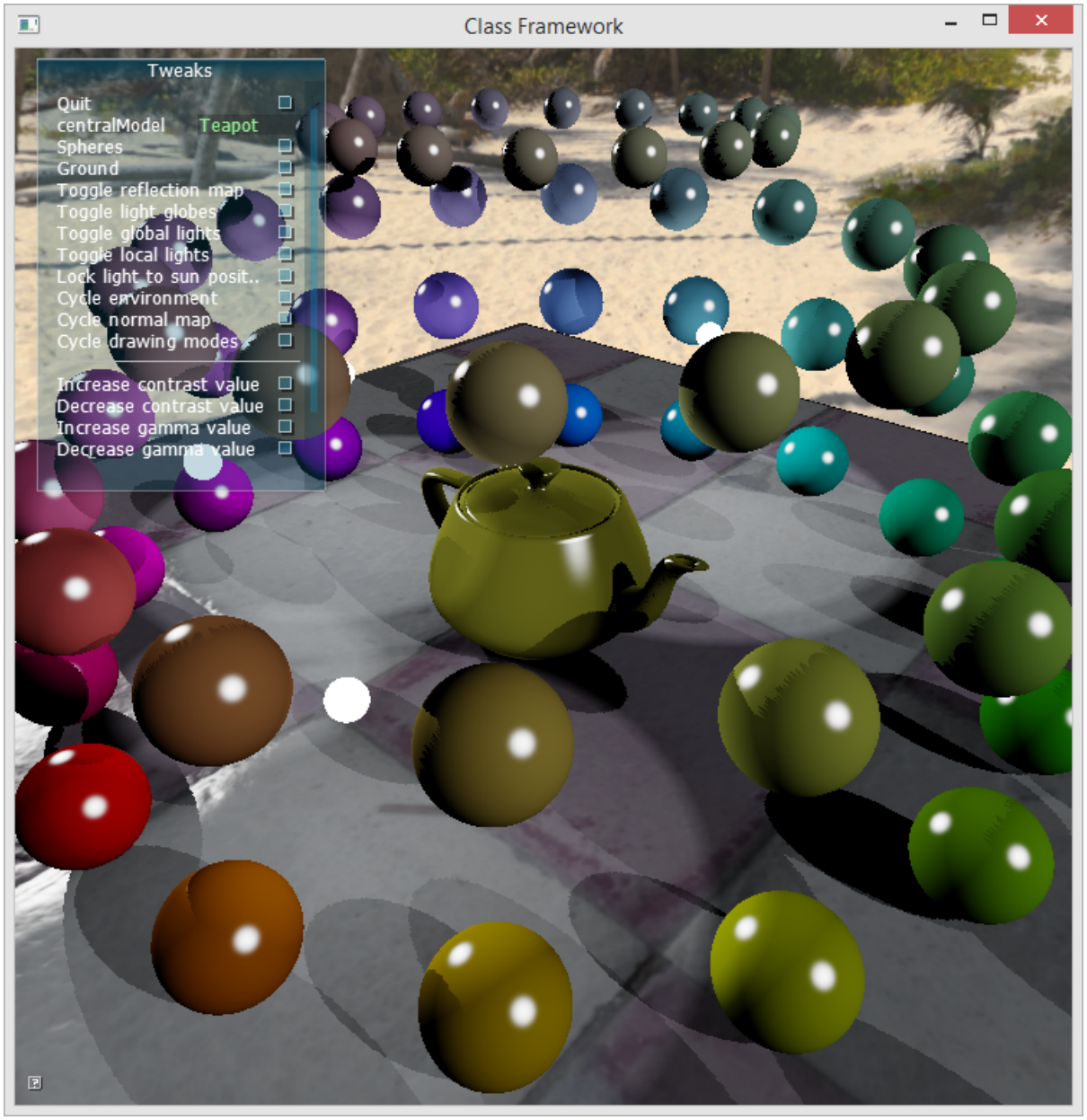


Final images showing the scene correctly lit with many finite range lights

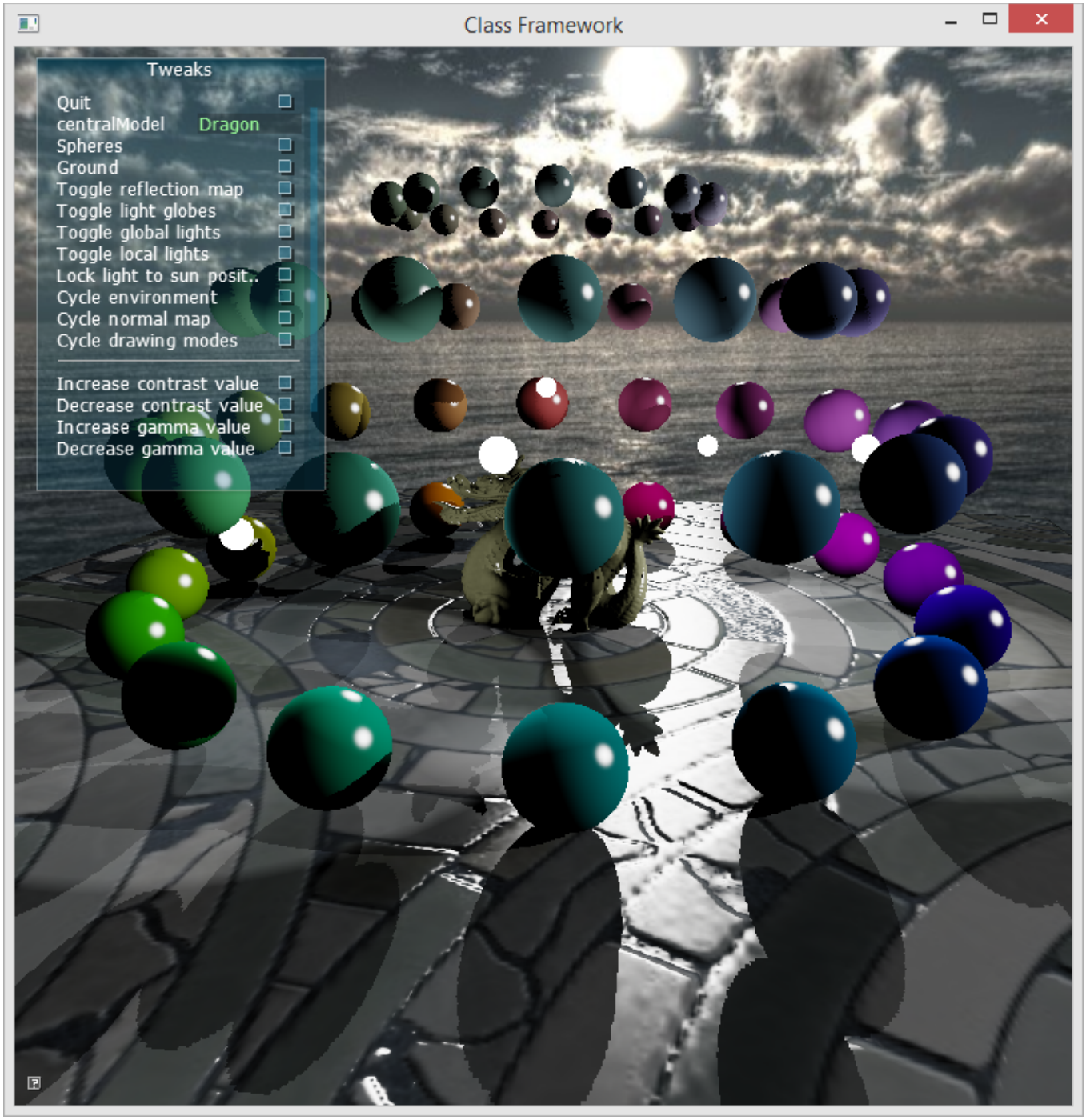
Here are several final images of several different versions of the scene. Note that the lights are more muted than the images above because the light range is being mapped from [0, infinity) to [0, 1] using the following equation.

```
vec3 eC             = gl_FragColor.xyz;
vec3 infinityFactor =          eC /
                    (eC + vec3(1.0,1.0,1.0));
gl_FragColor.xyz    = pow(infinityFactor, vec3(1.2, 1.2, 1.2));
```

Beach with teapot, two global lights, and six local lights:



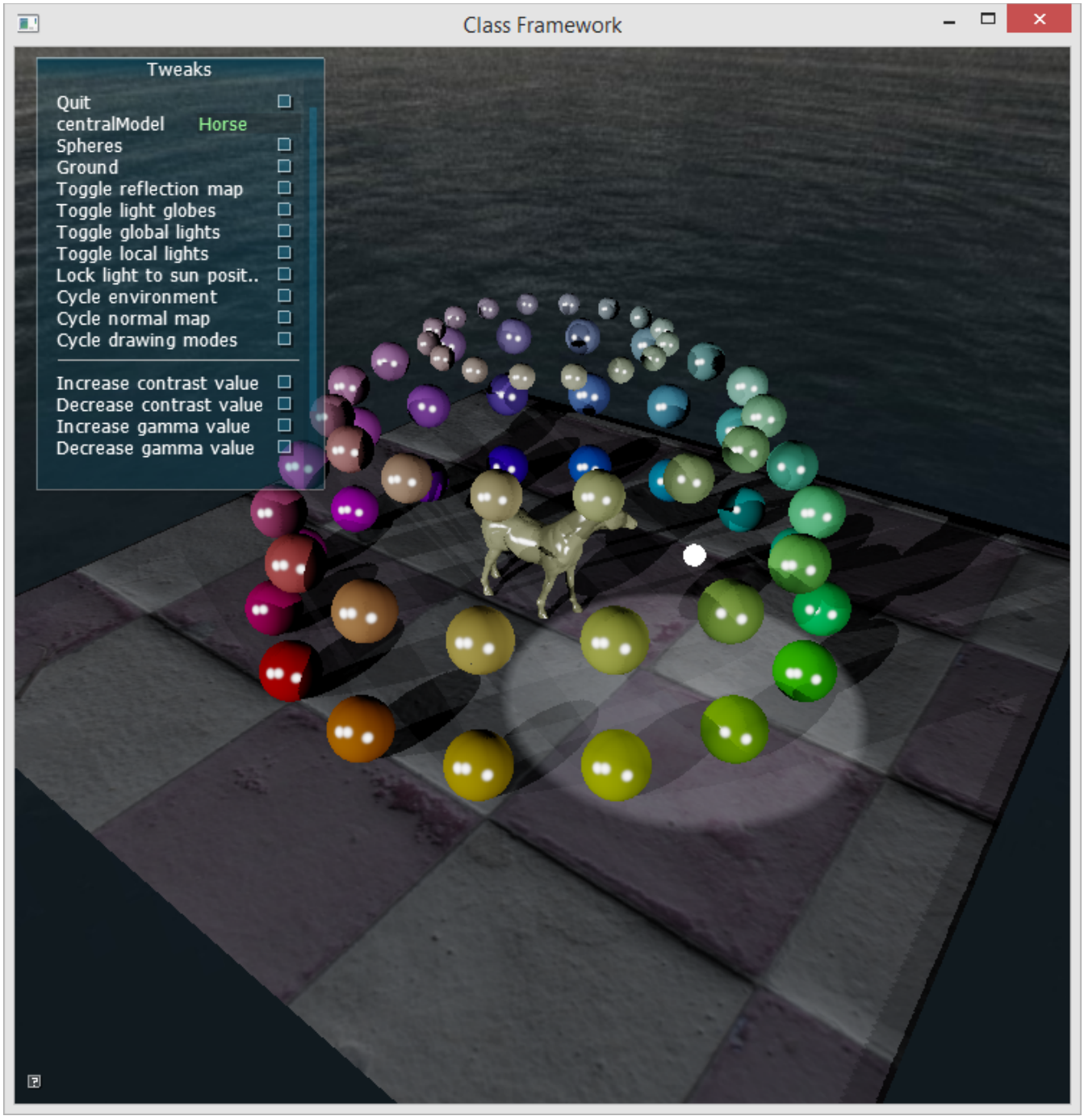
Ocean scene with dragon, two global lights, and six local lights:



Beach with Buddha, one global light, and 32 local lights:

Again, with different local light positions (since they are moving):

Ocean scene, horse with three global lights and one local light:



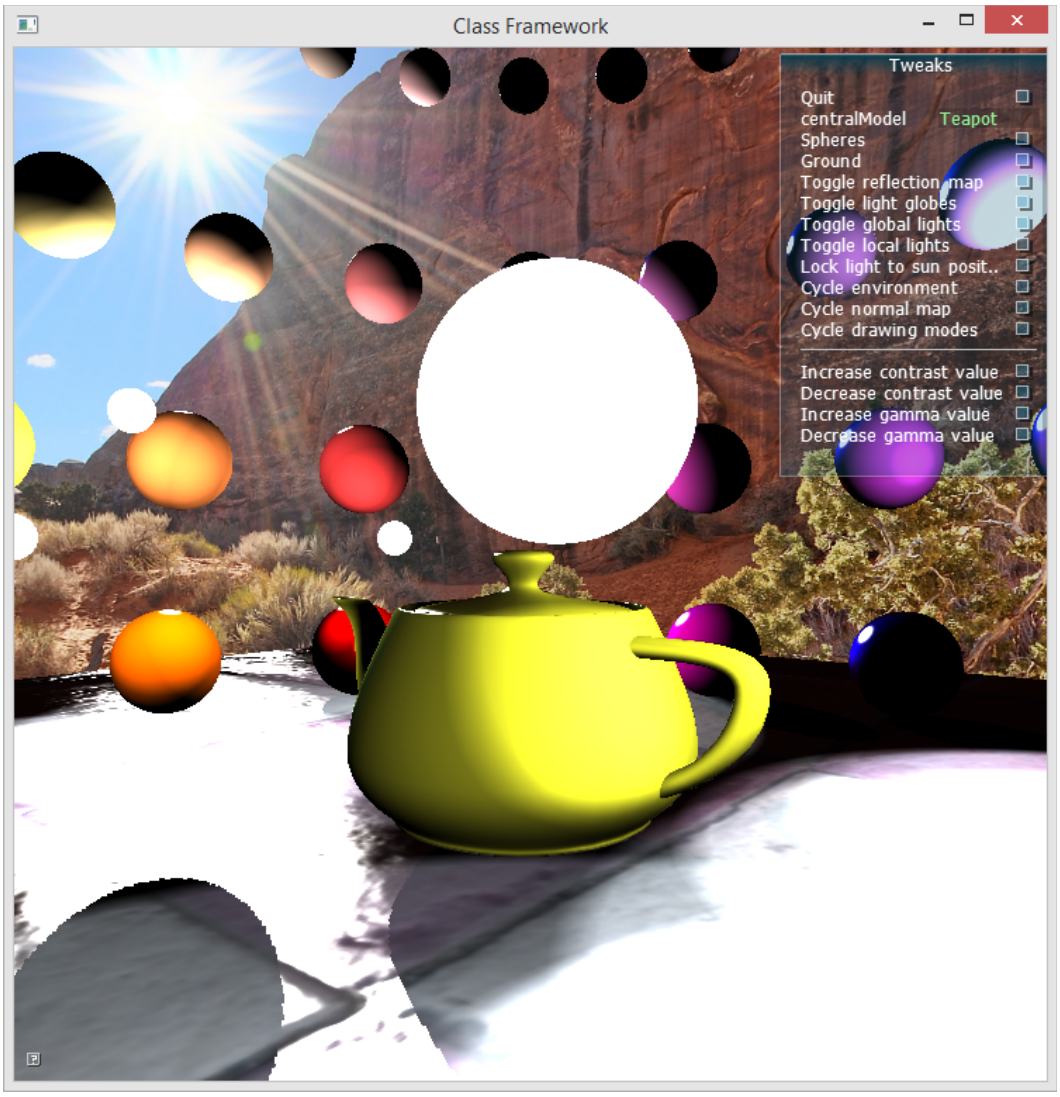
Appendix sections

This section includes items that are not required to be in the report, but which may be of interest.

Appendix A: Image-based lighting mode with High Dynamic Range

Including support for image-based lighting allowed me to see the contribution of each light, relative to brighter or darker light sources. Here are two examples.

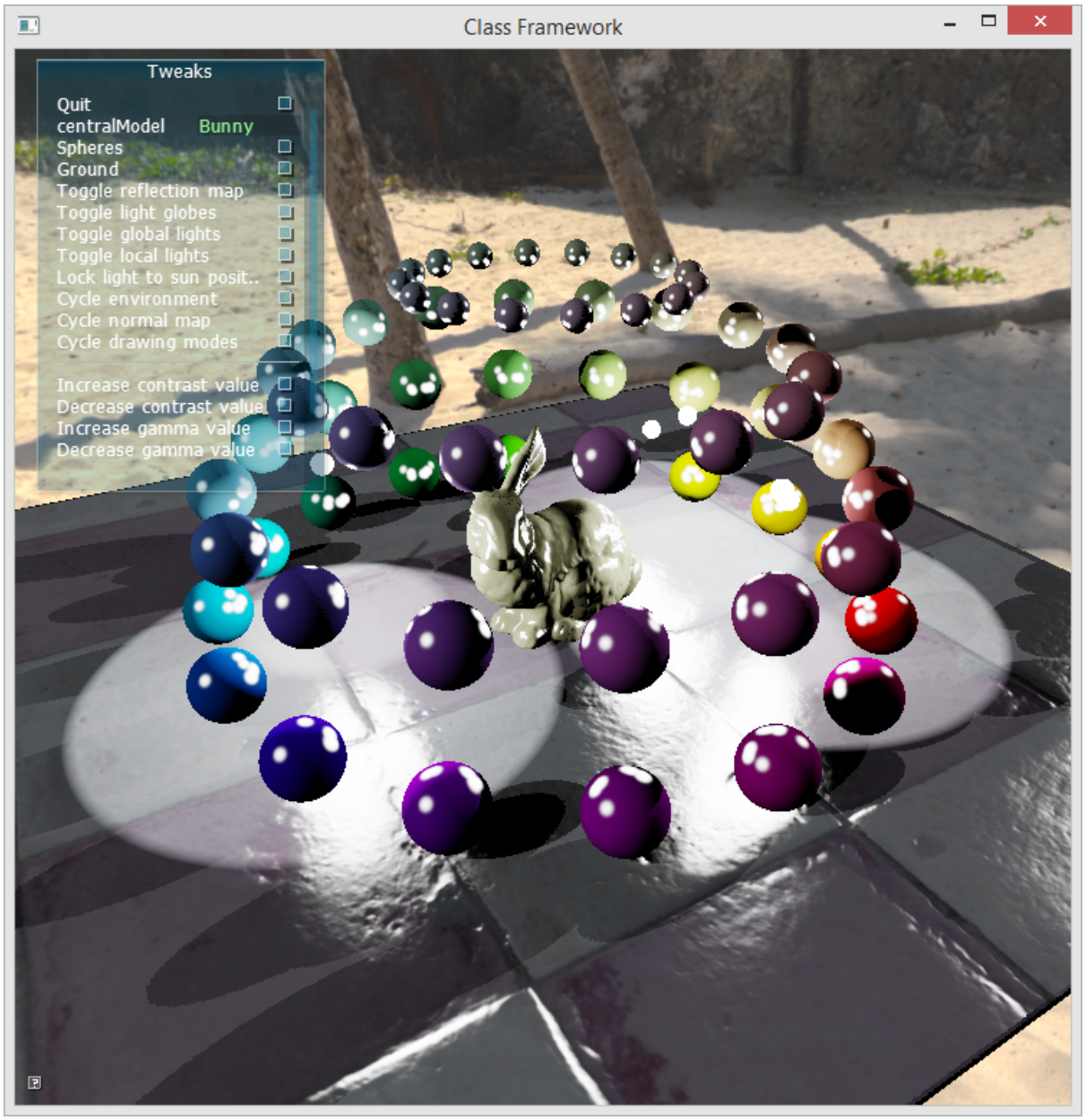
The Utah teapot, at the Arches national park in Utah. Normally in this scene, IBL makes the ground dark because the arch right above it is dark. But the local lights compensate for that, now that they are part of the framework. (The big white globe in front of the camera is a moving light.)


Class Framework
Tweaks
Quit
centralModel
Teapot
Spheres
Ground
Toggle reflection map
Toggle light globes
Toggle global lights
Toggle local lights
Lock light to sun posit..
Cycle environment
Cycle normal map
Cycle drawing modes
Increase contrast value
Decrease contrast value
Increase gamma value
Decrease gamma value

Buddha, looking good with IBL and also local lights (spheres are turned off):

Appendix B: Extra lighting mode for local lights

Along the way of finishing the project, I discovered that if you let the smaller lights contribute a specular value it can be really cool looking. It makes the ground material look like it is wet. To accomplish this, it's best to allow the specular contributions to render outside the range of the light, otherwise it gets cut off and looks incorrect; so I programmed a version of the local lighting shader that limits only the diffuse contribution by range, and always allows the specular contribution of each local light. It provides a unique look that might be useful in some scenarios, for example you could use this in a scene with streetlights on a rainy night. To see this mode in the project, compile it after commenting out "#define USE_LOCAL_LIGHTS_FOR_GEOMETRY" in OpenGLHelpers.h:



Appendix C: Build and test environment

Build information:

- Using the provided project framework with the code from CS 541.
- Using VS 2013 for development, but with VC 2012 installed for compatibility with the existing build target and GL libs.
- The GL libs are looked for in the same place the project template looks for them.
- My test machines use a version of the GL libs that I compiled myself (rather than the provided ones) to add support for release builds.
- Since code is only required for completeness, some Windows-specific constructs were used to save time.

Test configurations:

- Tested on a Dell Studio 17 laptop (ATI Radeon) and a Microsoft Surface Pro 3 (Intel HD graphics).
- Tested on a custom-built high-end computer with dual Radeon 6670 GPUs.
- Tested on an HP z420 workstation with a Quadro 600.
- Tested on Windows version 8.1 build 9600, and Windows 10 preview build.

Appendix D: References

sIBL archive at HDR Labs, http://hdrlabs.com/sibl/archive.html. Images used for testing:

- Arches PineTree
- Footprint Court
- Winter Forest

Images and assets provided in CS 541 are also used by the framework. These are not included in the.ZIP file to save space.