I developed a forward and deferred shader for the Blinn-Phong lighting model, a heuristic for deciding which of the two to use based only on framerate information, and a benchmarking system for profiling the performance of both shaders and the heuristic. Surprisingly, there were many cases where the forward renderer outperformed the deferred renderer; however the heuristic was able to consistently pick the faster one. Additionally, for a given static scene and lighting setup, one of the two shaders would consistently outperform the other one.

I implemented the renderer and shaders in OpenGL. My data structures/abstractions consisted of the following:

- Model: a .obj file consisting of vertices, normals, and texture coordinates
- Light: contains a position, color, and power
- Scene: a model and set of lights
- Benchmark: a scene and a path (list of positions and orientations for the camera)

For both shaders, the final output color of a fragment, defined by the RGB vector \( c \), is defined by the following constants and equations:

- \( A \), the ambient light, set to (0.1,0.1,0.1)
- \( E \), the vector from the camera to the fragment
- \( N \), the surface normal at the fragment
- \( T \), the albedo at the fragment
- \( C_i \), the color of light \( i \)
The forward shader consists of a single geometry and fragment shader. Light positions, powers, and colors are passed as uniform arrays. The fragment shader loops over all lights, performing lighting calculations provided that $P_i > 0.01$, where $P_i$ is the power of the $i$th light and $d_i$ is the distance from the $i$th light to the current fragment being shaded.

The deferred shader consists of 3 passes. In the first pass, the actual scene geometry is rendered, but with no lighting; the albedo, position, and normals of the fragments are outputted into separate textures. Additionally, the stencil buffer is filled in wherever scene geometry was rendered.

The second pass of deferred shading creates spherical meshes for each light, with the mesh’s center being the light’s position and the radius of the mesh being $\sqrt{\frac{P_i}{0.01}}$. Note this radius corresponds exactly to the distance cull in the forward shader. Additionally, the geometry shader gets an additional property for each vertex, an index indicating which light the vertex/mesh is associated with. Then, depth testing is set to greater than instead of less than, writing to the depth buffer is disabled, front-face culling is enabled, color blending is enabled (which just adds colors together with no regard for alpha values), and stenciling is set up such that fragments are only generated where geometry was rendered during the last pass. Then the sphere meshes are rendered; the result of all this setup is that a fragment is generated for every sphere which overlaps a geometry fragment from the first pass. Furthermore, since the front halves of the spheres were removed due to front-face culling, this means that any fragment generated from this pass corresponds to exactly the lighting contribution from a single light. In the fragment shader, lighting contribution is performed identically to that of forward shading, but the magnitude of the diffuse lighting and the specular color are outputted to separate textures. Note that instead of outputting the diffuse color, I output its magnitude (color sans the albedo term, which is a constant for the diffuse color and independent of any lighting calculations); the albedo is blended in the third pass with the diffuse lighting magnitude.

In the final pass, the albedo is blended with the diffuse lighting magnitude, specular color, and ambient lighting term; this is the final output of the deferred shader.

The shading heuristic will alternate between shaders under the following conditions:

\begin{align*}
  c &= A + D + S \\
  A &= (0.1, 0.1, 0.1) \\
  D &= \sum_i TC_i(N \cdot L_i) \frac{P_i}{d_i^2} \\
  S &= \sum_i 0.3C_i(E \cdot R_i)^5 \frac{P_i}{d_i^2}
\end{align*}
• If the heuristic made a switch in the previous frame, which resulted in at least a 33% lower framerate than the second to last frame, switch again.

• Every 10 frames, if the prior frame’s framerate was 10% worse than the average framerate from the duration of the render, switch.

The first condition is a catch for erroneous switches, and the second will switch if there appears to be a better policy (i.e. the current shader is slightly worse than the average). It only switches every 10 frames to avoid unnecessary jittering between the two shaders.

3: Results

I ran 7 benchmarks on 4 different test scenes. Some of these benchmarks were random movements throughout a scene with random lighting setup, and some I designed to test different performance/features of the shaders. In the graphs, the x-axis corresponds to time, and the y-axis corresponds to framerate. These graphs record only the rendering time per frame, which is everything that conceivably goes into drawing a frame (e.g. clearing color and depth buffers, sending buffers to the GPU, actually rendering the scene, copying and reading textures in deferred shading, etc.). However, this rendering time does not include the initial setup time, such as loading models into vertex buffers. Note that I measured time intervals with millisecond precision. Therefore there are large jumps in framerate for higher framerate values. For instance, the max framerate is 1000 FPS (1 millisecond delay in rendering), and the next highest value that is recorded is 500 FPS (2 millisecond delay).
Charger Sun Model vertices, triangles: 542000, 2540000.
Light vertices, triangles: 233000, 410000 triangles.
This is one of the few tests where the actual model complexity outweighs the complexity of the light meshes. There are 910 lights in the scene, 10 of which have sufficient power to cover the entire model, and the other 900 cover nothing.
Sponza Minitorch  Model vertices, triangles: 47811, 62890
This scene used 800 lights in a single area, which covered a portion of the room.

Sponza Sprites  This scene used 4 groups of 150 lights each, each covering a small area, spread out across the scene.
**Sponza Torch** Identical to Minitorch, but the lights had larger power.

![Sponza Torch Graph](image1)

**Suzanne Offscreen** Model vertices, triangles: 590, 968  
Light vertices, triangles: 256000, 450000  
This test is a specifically designed test to see how much a performance difference the light-culling aspect of deferred shading made. It consists of 1000 lights which do not contribute any light to the model and are not in the view frustum for the entire benchmark. As this test shows, it creates a massive perf difference, even moreso than not shading occluded samples.

![Suzanne Offscreen Graph](image2)
Suzanne 1000 Suns Identical to Suzanne Offscreen, but now all 1000 lights have sufficient power to each light the entire scene. This demonstrates that with enough lights and a simple enough model, forward shading surpasses deferred shading by a significant margin.

![Suzanne 1000 Suns Diagram](image)

Tower Highlights Model vertices, triangles: 10638, 4794. Random test scene; 100 suns which cover the entire scene. Notable that although the scene is relatively simple, and that there are often at least 3 occluded fragments per pixel in the scene, deferred shading performs much worse than forward shading.

![Tower Highlights Diagram](image)
4: Conclusion

Deferred shading was not as effective as I hoped; of the 7 test scenes, deferred rendering only performed better in 2 of them. I suspect this is because for all scenes, I grouped large numbers of lights together. However, deferred shading does have a significant advantage in its ability to cull lights. Interestingly, for all my scenes, there was a shader in the benchmark which was consistently better than the other one. I suspect this is due to them being static scenes (lights retain their initial properties of position and power, and models are static). However, even in my 3 benchmarks of the Sponza scene in which I tried various lighting conditions, forward rendering was consistently the best choice. So it may very well be possible to compute what the best shader is for a scene, and then always use it. My deferred shading heuristic performed well, in the sense it was able to pick the best shader consistently, and only dipped into the suboptimal shader rarely.

5: References

https://github.com/syoyo/tinyobjloader - Used to load .obj files in my renderer.
http://www.opengl-tutorial.org/ - Provided the basis for my renderer.