Techniques for Simulating Shadows in Real-Time Graphics

Drew Gottlieb
08 May 2018

Goal

Global Illumination techniques can produce photorealistic results, but are time consuming and computationally intensive. Real-time graphics permit interactivity, but must largely rely on direct illumination methods that are hardly photorealistic. In particular, lack of shadows and subtle edge shading of ambient illumination eliminate significant perception of depth.

My project comprised of two mini projects to address each of these, learning and implementing techniques popularly used in modern real-time graphics programming.

Shadow mapping is a technique for computing what surfaces are directly visible to light sources and modulating diffuse and specular shading accordingly.

Screen-space Ambient Occlusion (**SSAO**) is a technique for cheapy approximating subtle shading in concave surface edges.

Algorithms

With standard Phong shading, materials have three additive lighting components: ambient, diffuse, and specular. Ambient is a "constant" lighting value added to all materials within range of a light source. It is used to make up for lack of indirect illumination, since surfaces that don't directly see a light source would still receive ambient lighting. Diffuse illumination is a lambertian shading taking into account how much a surface normal points towards its light source. Specular illumination is a simulated reflection of the light source, taking into account surface normal as well as viewer and light source positions.

Shadow Mapping

Since diffuse and specular illumination are intended to model surface illumination caused directly by the light, it makes no sense for surface so receive such illumination when there is another object between the surface and the light source. The algorithm for shadow mapping determines if a surface has line-of-sight, and if not, skips diffuse and specular illumination, only having ambient illumination.

To determine whether an object's surface can see the light source, a texture called a *shadow map* is produced at the start of each frame's render. To produce a shadow map, a virtual camera is effectively constructed that represents what the light sees. In the case of a spot light, the light's camera would have a view matrix matching the light's transform, and a frustum projection matrix matching the spot light's field of view.

The scene is then rendered into a framebuffer, only storing the depths. This framebuffer serves as the shadow map, and is then fed into each shader when doing the final scene render, along with the view and projection matrix of the light.

When rendering the final material, the world-space position of the fragment being rendered is multiplied by the light's view and projection matrix to get the x and y position within the shadow map that corresponds to the ray from the light that would intersect the fragment. Then the z position of the fragment in the shadow map is compared to the value sampled from that texel of the shadow map. If the z is a greater value than the sample, the position is not the closest object to the light along that ray, and is therefore in a shadow.

Screen-Space Ambient Occlusion

Ambient illumination–which intends to make up for lack of global illumination–also needs to take nearby geometry into account. Various techniques exist to make convex geometry receive last light, all called ambient occlusion. A particular version which is well suited to real-time graphics is screen-space ambient occlusion, meaning that it's computed once per screen-space pixel as a single post-processing step.

The idea depends on having a deferred rendering pipeline. With deferred rendering, we render all geometry to multiple textures, not as final shaded pixels, but as data encoded as pixels: position, normal, diffuse color (a.k.a. albedo), specular intensity, etc. Then, a single screen-space quad is rendered using each of these values to shade the final screen pixel.

SSAO introduces an intermediate pass between generating the initial textures (know as g-buffers), and the final screen texture (known as the lighting pass). The SSAO pass uses the screen-space pixel's position and normal as well as its neighbors to determine an occlusion factor.

The SSAO sampling process works as follows:

1. Generate a bunch of sample positions in a half-sphere hemisphere oriented around the pixel's surface normal, centered at its surface position.
2. Convert those sample positions to screen space, and look up their depths from the camera in the depth buffer.
3. If the sample position is behind the depth buffer value for the corresponding screen coordinate, there is geometry there, so consider that single sample to be occluded.
4. Average all sample occlusion values to get the overall occlusion factor.

Then feed the SSAO occlusion factor texture into the final lighting pass shader and multiply against it when calculating ambient illumination.

Implementation

The basis of my implementations is based on tutorials from learnopengl.com[1,2], integrated into my own engine architecture.

I made a FrameBuffer class that maintains a depth texture and one or more color buffer textures onto which OpenGL can render.

The base class for all scenes, Scene.cpp, implements the pipeline for rendering the depth map for the active light in the scene.

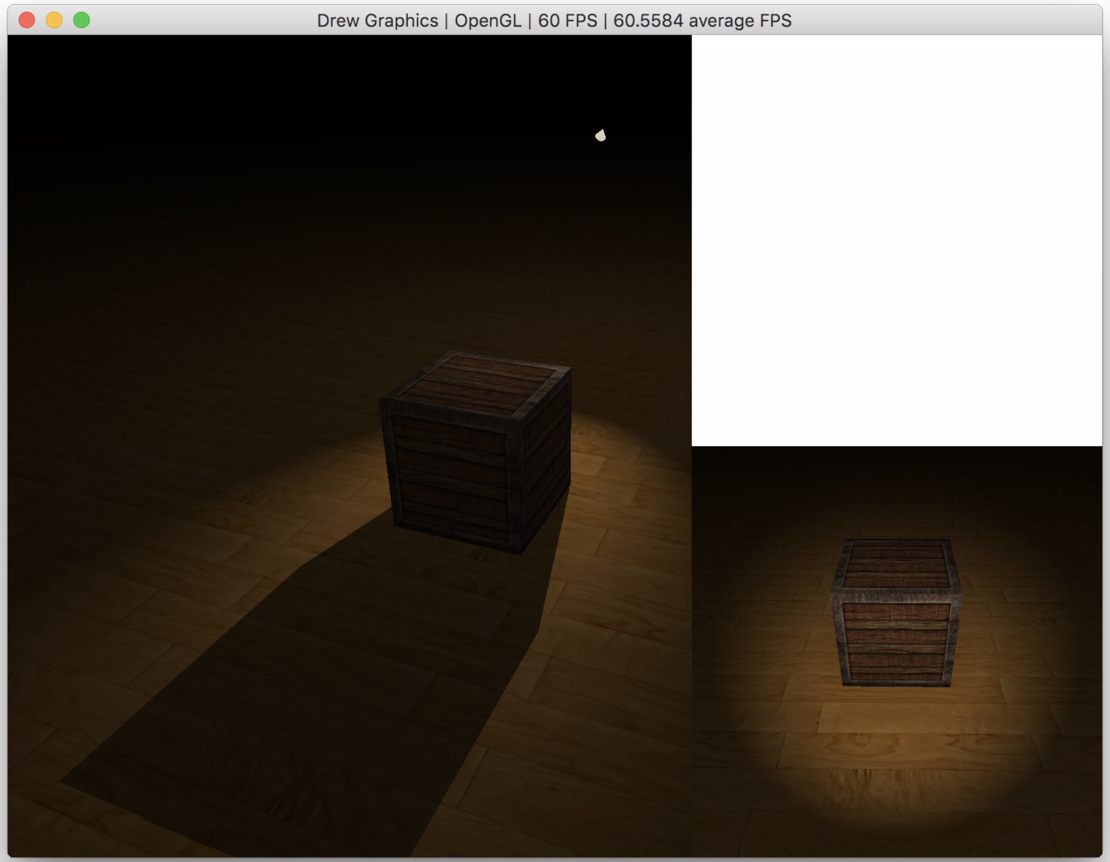
By default, my engine is a forward renderer. A deferred rendering pipeline and SSAO pass is implemented in AOScene.cpp.

The latest code with build instructions is available on GitHub:
https://github.com/dag10/DrewGraphics

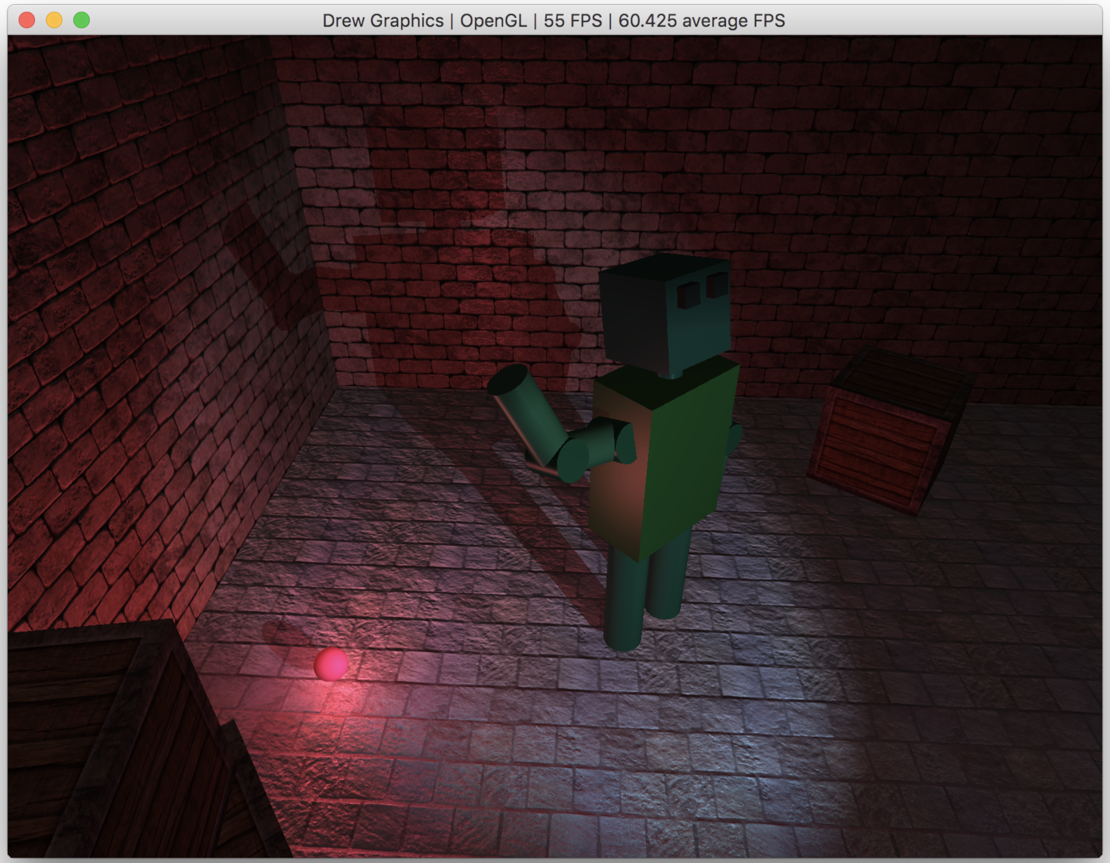
For posterity, here's a link to the commit from this report's submission date:
https://github.com/dag10/DrewGraphics/tree/e774b29a99a4b49b42df17a79ce924b9fb71ff02

[1] https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping
[2] https://learnopengl.com/Advanced-Lighting/SSAO

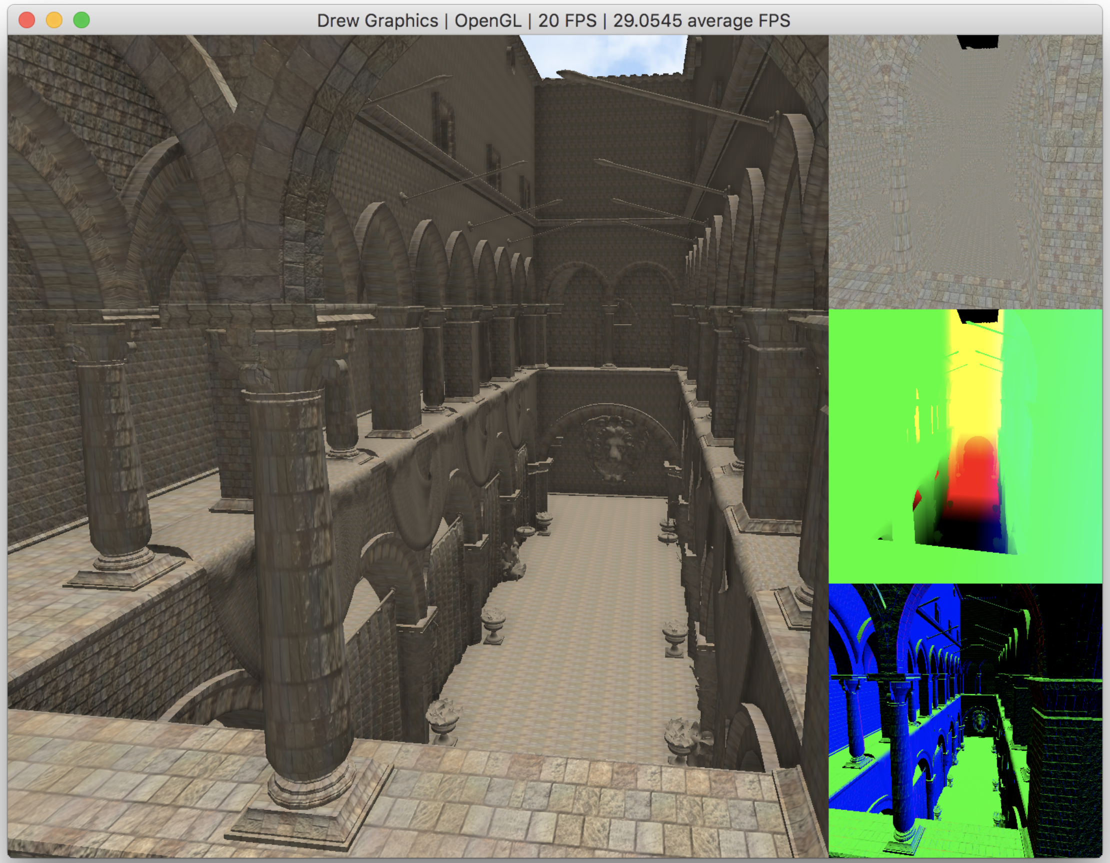
Results



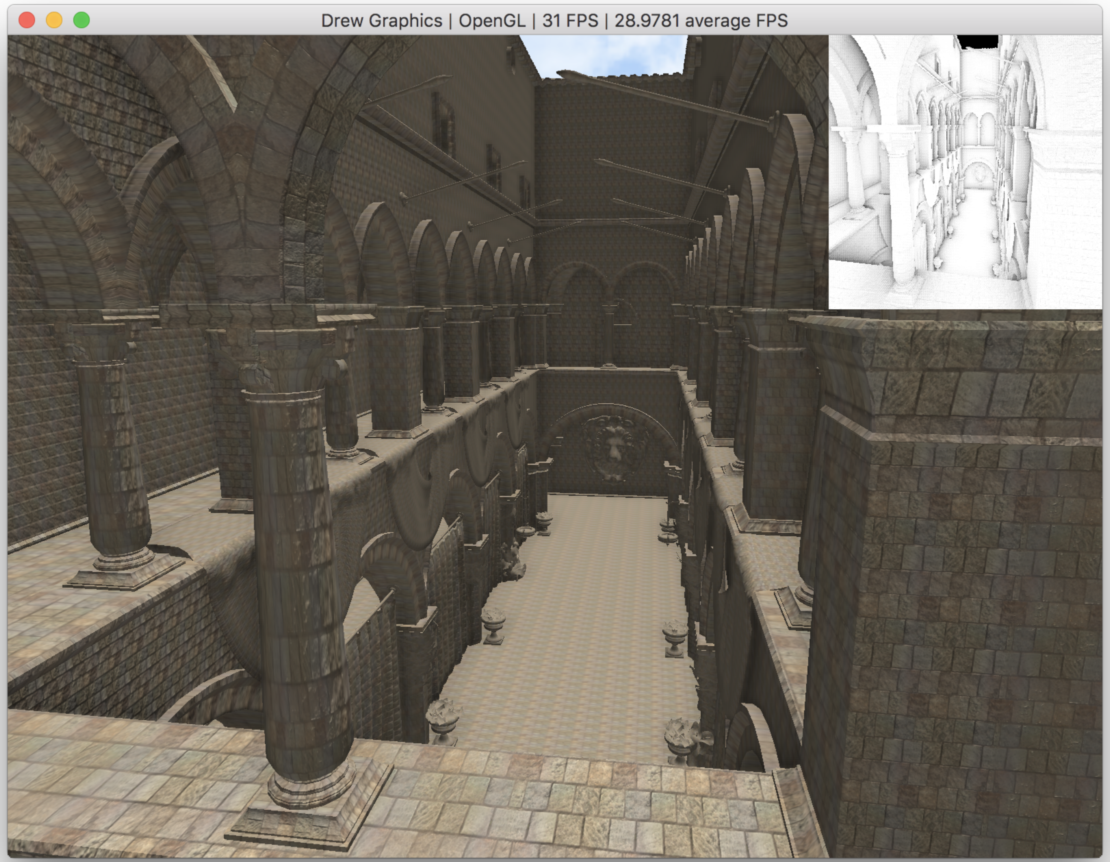
Scene showing a real-time shadow on a box. Overlaid is a depth map (low-contrast, top-right quad) and a view of what the light "sees" (bottom-right quad).



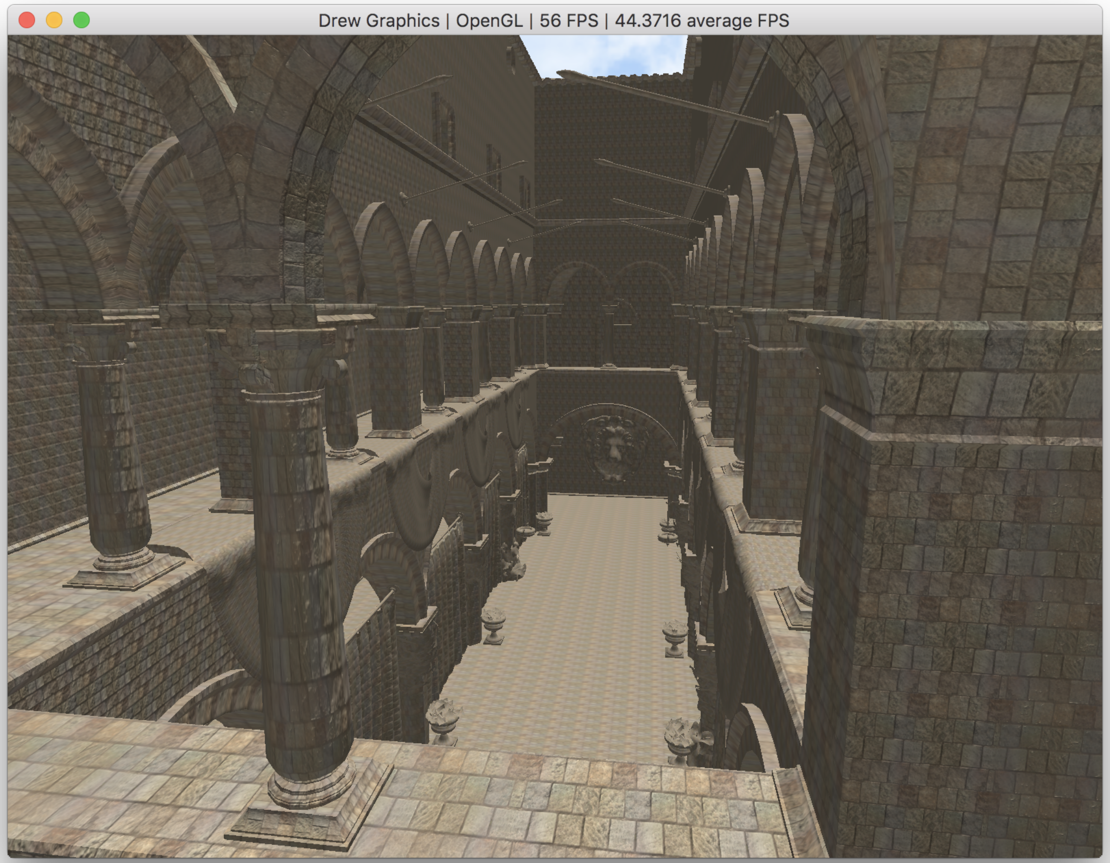
Scene showing real-time shadow of a more complex scene.



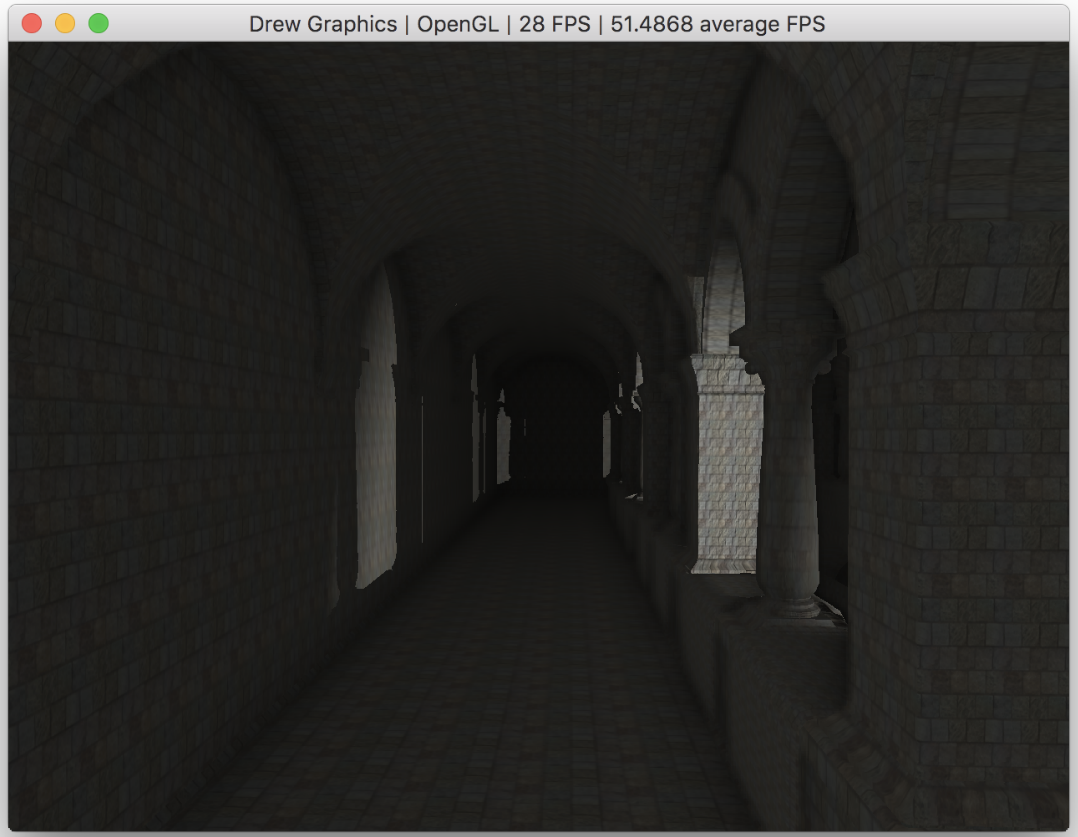
Scene showing a deferred rendering pipeline. Overlaid quads are, from top-to-bottom: albedo, world-space position, world-space normal.



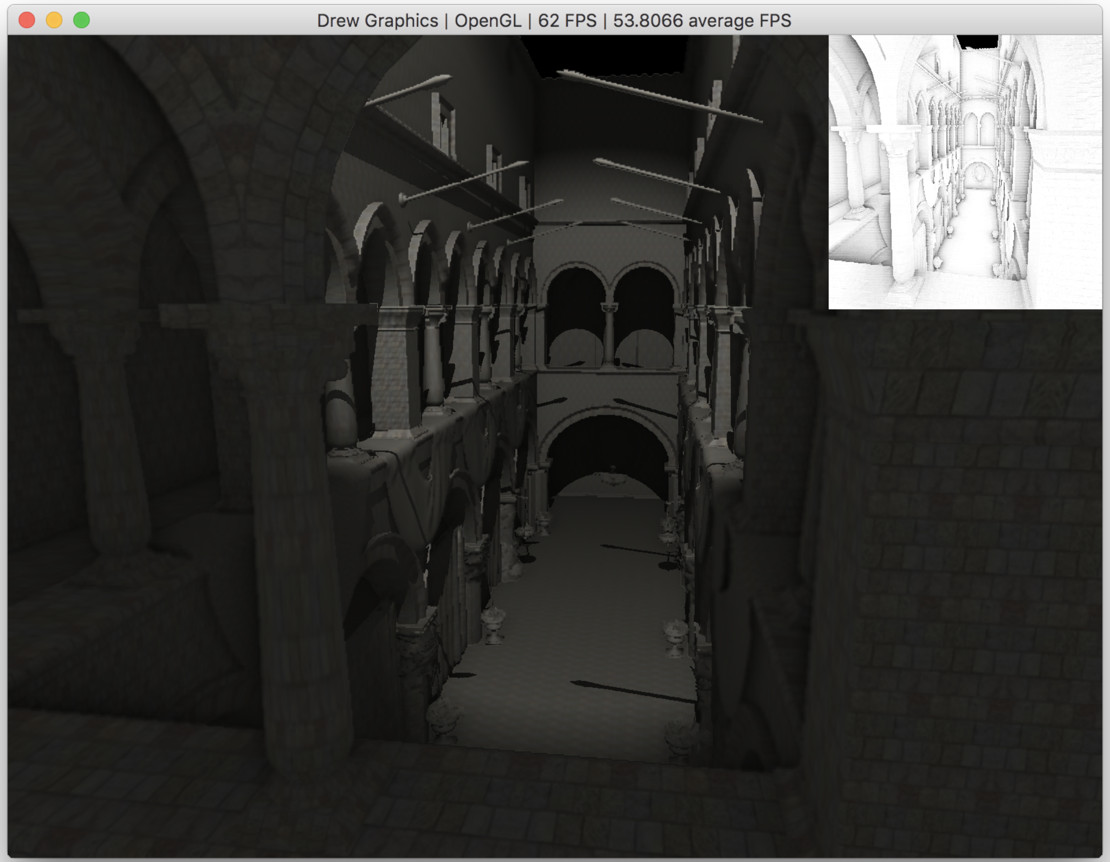
Same scene showing the ambient occlusion factor texture overlaid in top-right quad.



Same scene without any ambient occlusion, just flat ambient along with regular diffuse and specular illumination.



Scene with both real-time shadows and ambient occlusion. Spotlight is off camera to the right.



Scene with both real-time shadows and ambient occlusion.