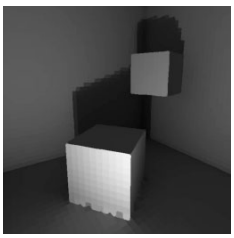
Global illumination



Description	Global Illumination implementation
Period	Spring 2010
Languages & Libs	C++, SDL
Tags	Rendering

I implemented a simple global illumination algorithm, which constructs and solves the sparse matrix which describes how much tiles reflect light to each other. It supported only gray-scale rendering and wouldn't scale up to bigger scenes, but it was nevertheless an interesting project and I learnt a lot about 3D geometry, linear algebra and computer graphics. Graphics was drawn by a software renderer which uses only standard SDL primitive draw calls.

When the program is started, scene data is read from a XML file and object surfaces are generated. Their spatial relationships are calculated, and the $n \times n$ symmetric matrix $\mathbf{A}$ is generated, which encodes the surface-to-surface reflection magnitudes. The global illumination equation simply becomes $\mathbf{A}\bar{x} = \bar{b}$, where $\bar{b}$ is mostly filled with zeros, indicating that the number of inbound photos matches the number of absorbed and reflected photons. Light sources function the same way as normal surfaces, with the only difference that they only "reflect" photos, without receiving any. Light sources cause the only non-zero values in $\bar{b}$.

The equation $\mathbf{A}\bar{x} = \bar{b}$ is solved by having an initial estimate of uniform luminosity, and then iteratively refining it using *conjugate gradient method*. It is a very easy method to implement and suits well symmetric sparse matrices, but it may have slow convergence if the $\mathbf{A}$ is somewhat ill-conditioned. At some complex geometries it wasn't able to converge with a few iterations, but instead the luminosity seemed to fluctuate from region to region. These pre-calculated values are then used when the scene is rendered to the screen. Example results can been in figures 1 and 2.

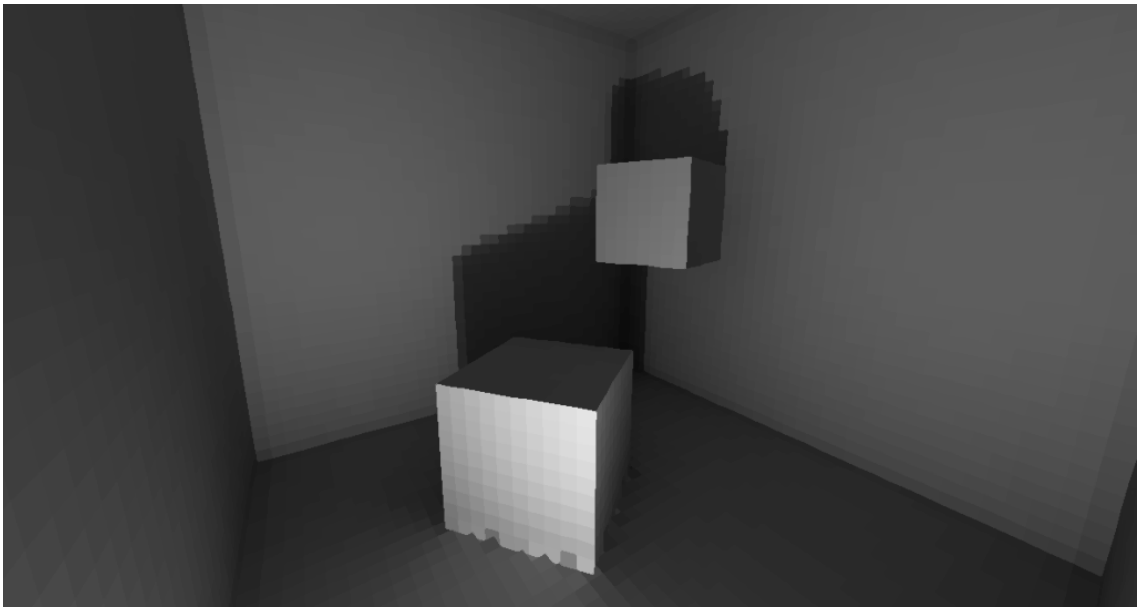


Figure 1: Shadows are a natural outcome from global illumination.

In Figure 1 objects can be seen "casting" a shadow on the wall behind them, and darkened corners, proving that the geometry was interpreted correctly. The blockiness of shadows is a typical problem in simple tile-based global illumination renderings, because the luminosity is forced to be uniform across each tile, and thus gradients cannot be represented. One solution would be to use more tiles, but then calculations would be a lot more time consuming, and it wouldn't solve the underlying issue. In future I might have a second look at this problem, and use OpenGL for graphics with bi-linear interpolation. It would also do the rendering in color, and maybe even support caustics and user-controlled light sources. Some of the calculations could be done in CUDA, and the rest would be executed in multiple threads where possible.

An other shadow phenomenon can be seen in Figure 2, in which the shadows on the ceiling and floor are very different. This occurs because the light source is low on the floor, in which case the wall's corner causes the shadow in the ceiling have a strong edge. In contrast to this, the shadow on the floor is very soft. This occurs because for photons to hit the floor, they have to bounce from the ceiling and walls, making it possible to reach each floor tile from multiple paths, and the luminosity decreases gradually.

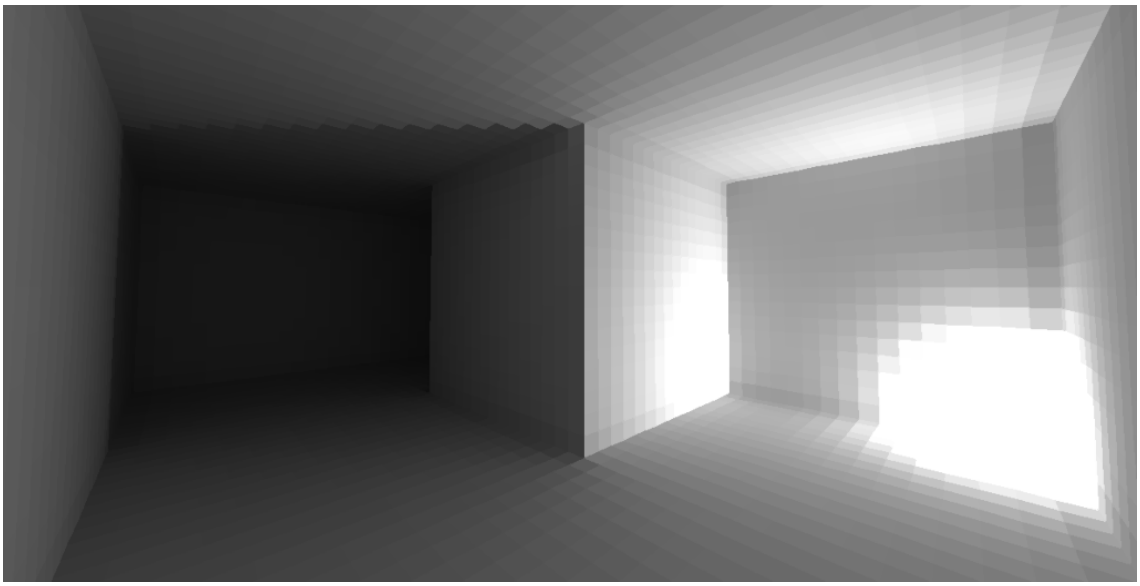


Figure 2: Since the light source is low on the floor, it is worth noting that the shadow on the ceiling is hard, where as the shadow on the floor is very soft.