Using Enlighten with Unity

Enlighten by Geomerics is the lighting system used in Unity for computing global illumination - see the Geomerics Enlighten homepage for more information.

Benefits of Enlighten

The benefits of using Enlighten in Unity include:

- **Solutions for everyone**: A choice between real-time lighting or “baked” (precomputed, static) lighting in real time lets you pick the lighting solution that best meets your needs.
- **Flexible, reactive lighting**: Real-time indirect lighting makes it possible to move any Light in the Scene and have complex lighting effects update as it happens.
- **Smooth workflow**: Iteration time is very fast.
- **Efficient area lights**: Emissive surfaces have very low CPU overhead, and act like Area Lights with accurate visibility.

Workflow

The intended workflow when using Enlighten is:

1. Set up the Scene
2. Precompute it
3. Light it

See the Unity Learn Introduction to Precomputed Realtime GI for a more detailed overview.

The global illumination calculated by Enlighten updates in real time if you change Light or Material properties. Moving a lightmap static GameObject while lighting the Scene invalidates the precomputed data and triggers a new precompute, so doing so is not recommended. Dynamic objects should be lit using Light Probes.

After the precompute, indirect light in lightmaps and Light Probes can be updated at runtime. As this needs to be updated in real time, the resolution should be kept reasonably low. However, indirect or bounced light has a much lower frequency than direct light, meaning it can be captured in a lower resolution Texture without artifacts.

Under the hood

Global Illumination is described by a complex equation called **the rendering equation**:

$$L_{\mathrm{o}}(\mathbf{x},\, \omega_{\mathrm{o}},\, \lambda,\, t) \;=\; L_{e}(\mathbf{x},\, \omega_{\mathrm{o}},\, \lambda,\, t) \;+\; \int_{\Omega} f_{r}(\mathbf{x},\, \omega_{\mathrm{i}},\, \omega_{\mathrm{o}},\, \lambda,\, t)\, L_{\mathrm{i}}(\mathbf{x},\, \omega_{\mathrm{i}},\, \lambda,\, t)\, (\omega_{\mathrm{i}} \cdot \mathbf{n})\, \mathrm{d}\,\omega_{\mathrm{i}}$$

The rendering equation calculates how to define the light leaving a point on a surface. Although it is the basis of the majority of realistic rendering techniques in the last 30 years, it is way too complex to solve quickly. However, it can be approximated - see the Wikipedia page on Rendering equations for more information.

Enlighten narrows down the problem by assuming there is a finite set of static elements and only diffuse light transport. This method is also known as radiosity.

The material dependency can be moved out of the integral. This makes the integral a sum of the radiosities of other elements, multiplied by the form factor between two elements. The form factor is the fraction of light leaving element **j** arriving on element *i*. Multiply this by the material properties and then add whatever light it is generating (or emitting).

We can solve discrete clusters iteratively over multiple frames. The solver might need to run a few times for the solution to converge, but solving this is entirely independent of the frame rate, and the human eye is not sensitive to slow and subtle changes in indirect illumination.

The following is the rendering equation simplified:

$$B_i = L_e + \rho_i \sum_{j=1}^{n} F_{ij} L_j$$

- **Bi** is light from point **i**
- **Le** is light emitted directly by cluster **i**
- **ρi** is the material properties
- **n** is the budget
- **Fij** are the form factors (how much **i** can see **j**)
- **Lj** is the light from cluster **j** (last frame)

Example

This simplified equation can be explained visually as seen below. First, the scene geometry is grouped into systems that Enlighten can work on in parallel. Next, the geometry of each system is cut into discrete clusters. Think of this as a voxelization of the geometry - see documentation on GI visualization in the Scene view for more information.

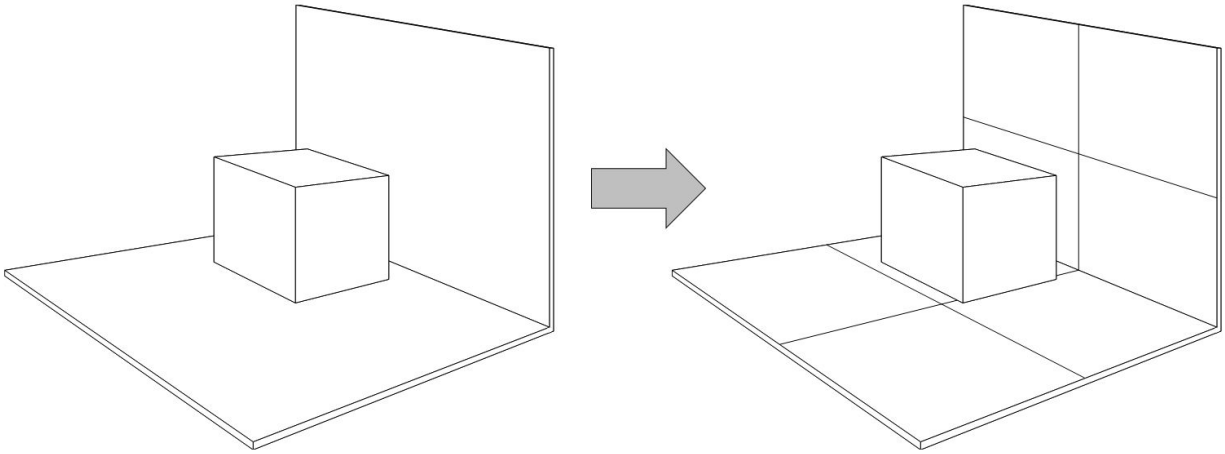

Image source: Geomerics

When calculating the indirect lighting for each pixel, the clusters the pixel can see are precomputed, and at runtime all the cluster colors are added up.

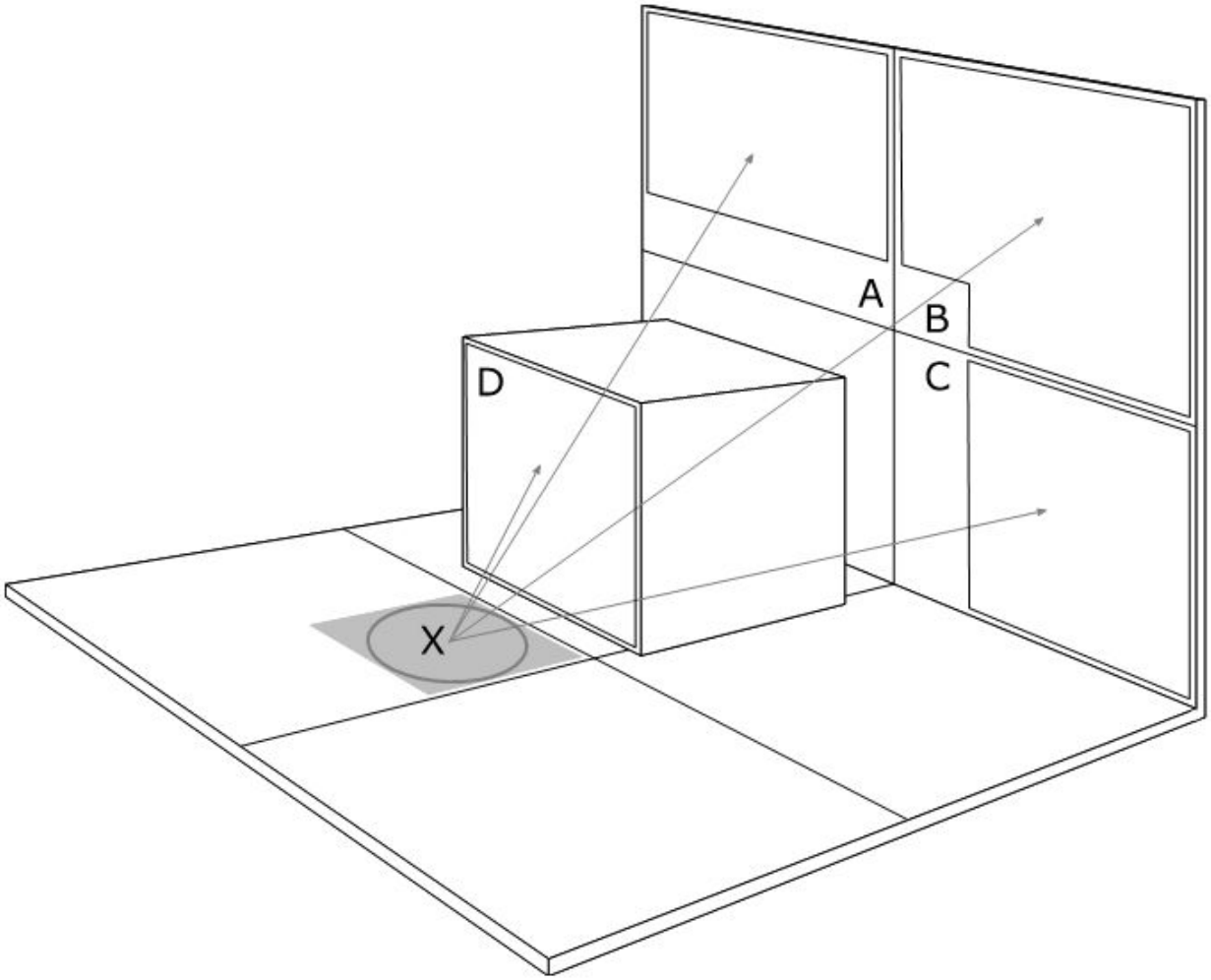

Image source: Geomerics

In this example, the radiosity form factors Ω for A, B, C and D are found in the precompute step.

At runtime, Enlighten will compute the weighted sum:

$$X = \Omega*A + \Omega*B + \Omega*C + \Omega*D$$

So a red cluster and three black ones will result in a slightly red result.

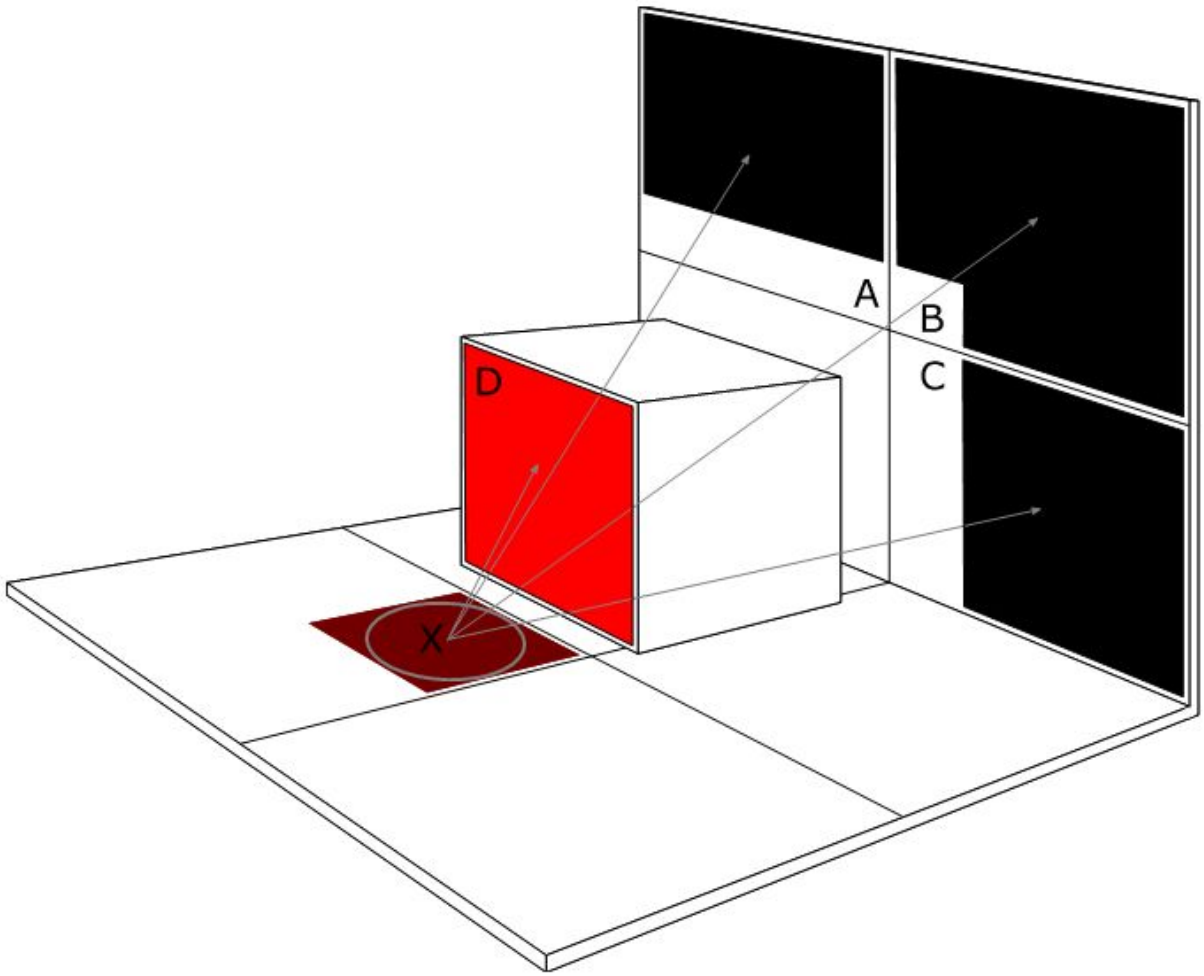

Image source: Geomerics

Four slightly red input clusters will result in the same output.

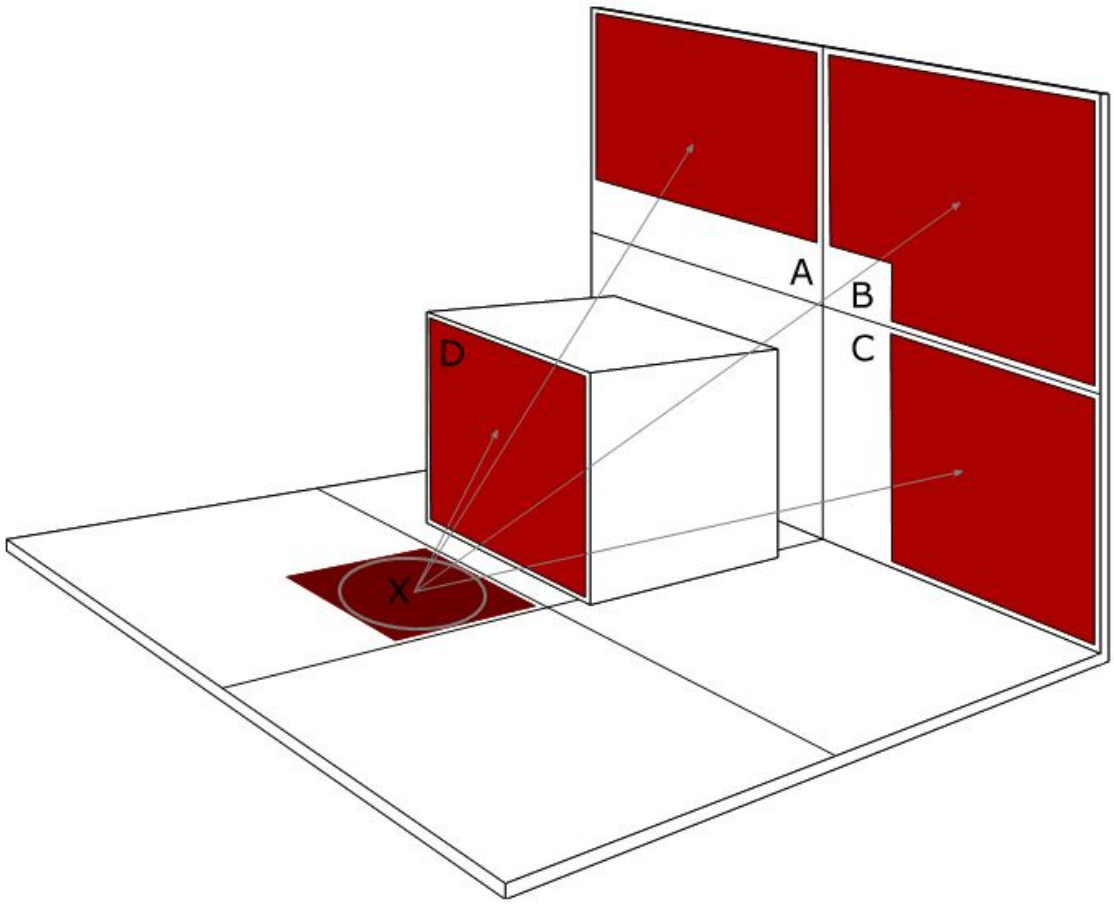

Image source: Geomerics

Here is another example that uses black, red, blue and green input clusters, resulting in a dark green output.

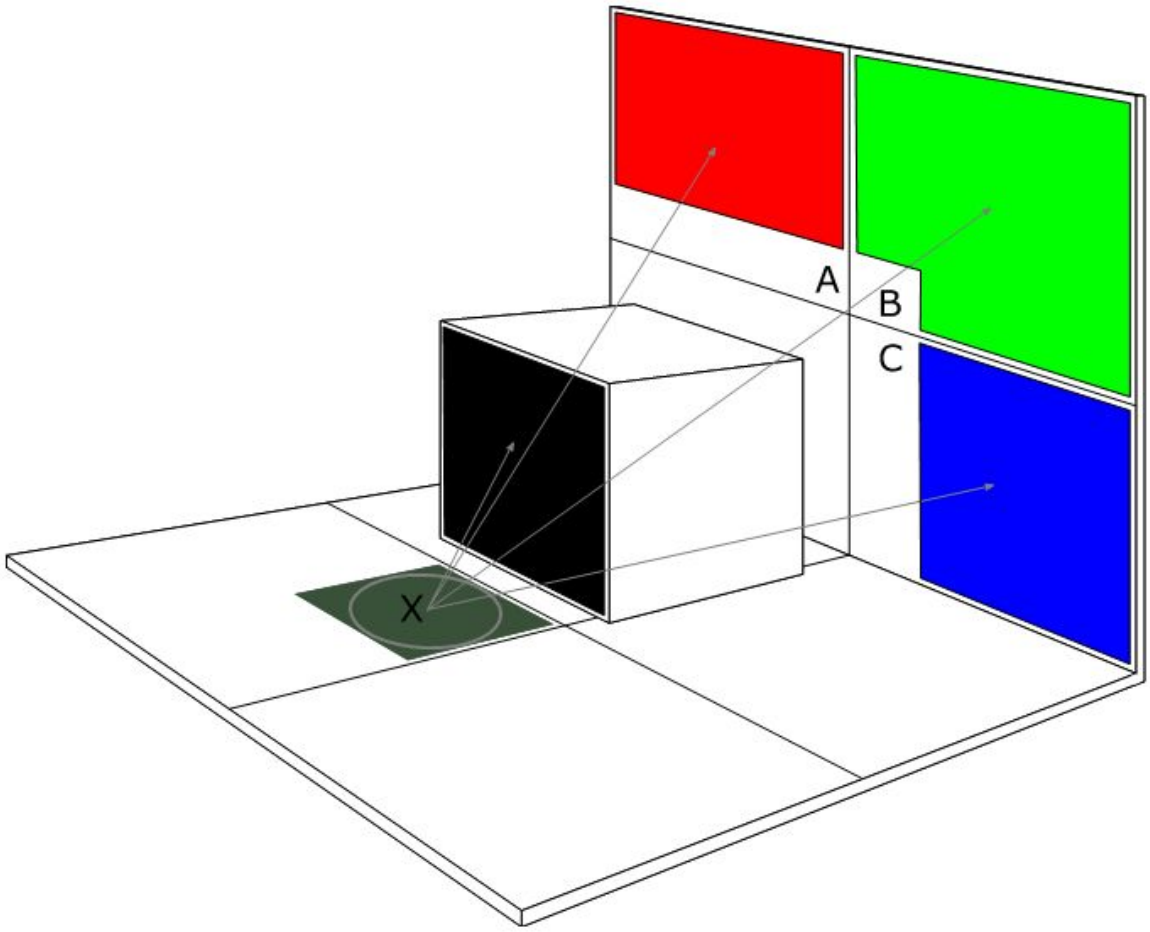

Image source: Geomerics

Clusters and systems

When calculating lighting, Enlighten views the world as a collection of surface patches (clusters). These are converted into a tree of clusters that have a similar view of the world. All lighting is applied to the clusters, both from discrete light sources and emissive surfaces.

Clustering is not just used by a system, but for a system and the systems that are dependent on it. Clusters can span across systems, and can also see (and so have form factors for) clusters in other systems. Use the **Systems** mode in Scene view to get an overview (see documentation on GI visualization in the Scene view for more information).

The light transport stage calculates visibility between clusters. Accuracy is determined by the **Irradiance Quality** setting, which dictates the number of rays cast, in the Lightmap Parameters menu. Given a quality setting, a number of form factors are selected in a way where visibility error is minimized.

Irradiance Budget defines the number of form factors to keep. Form factors will include leaf nodes in the tree of clusters and nodes higher up in the tree structure (non-leaf nodes). Note that these nodes may include clusters with different enough lighting to cause a leak. The usual fix for this is to increase budget (at a higher runtime cost).

When using precomputed realtime GI, a number of background CPU worker threads do the lighting calculations asynchronously. The number of threads are defined by the **CPU Usage** setting in the Lighting window.

Light probes are updated in real time as well and work the same way as lightmap texels. Form factors per probe are stored to calculate the lighting from clusters.

For more information, see documentation for Unity developers produced by ARM, the company behind Geomerics.