Towards Interactive Rendering for Lighting Design

Mariusz Pasinski and Michał Chwesiuk[*]
Supervised by: Radoslaw Mantiuk[†]

Institute of Computer Science
West Pomeranian University of Technology
Szczecin / Poland

Abstract

We introduce an interactive method for computing illuminance for lighting design purposes, focusing on photometric quantities. The main goal of lighting design rendering is to achieve accurate photometric quantities on the desired calculation planes. The vast majority of lighting design applications are based on unbiased global illumination rendering techniques, which are executed entirely on CPU. The performance of those techniques is limited and makes it difficult to work interactively with the lighting project. We evaluate whether a renderer based on instant radiosity implemented in OpenGL guarantees the accuracy of lighting calculations comparable to the lighting design industry standards. Our renderer exploits photometric light sources defined by the IES standard and physically based materials to ensure the correct calculation of the illuminance. The results are compared with the Radiance renderer, which is considered to be a standard for the lighting design calculations. As it is work in progress, the results are tested using a simple scene, an isotropic light source and perfectly diffuse materials.

Keywords: rendering for lighting design, interactive rendering, GPU rendering, global illumination, lighting design, photometric rendering.

1 Introduction

Lighting design is concerned with the design of lighting systems that provide comfort to people [4]. A good lighting system provides interiors or exteriors where people can see clearly, easily and without discomfort. Such designs have rigid illumination constraints, some of which might be enforced by law. This raised a need for specialized light visualization applications that are powered by a *photometric renderer*, in which physically based and unbiased rendering [10] delivers data on the physical illuminance (expressed in *lux*) in certain locations in virtual scenes.

A recognized standard in the lighting design application is the *Radiance* renderer [11]. However, Radiance is performing all the computations exclusively on the CPU. Thus, despite the high accuracy of calculations, the rendering performance is limited and Radiance is hard to use in interactive systems.

In this paper we propose an algorithm tailored for lighting design applications, based on *instant radiosity* [8], which can exploit heterogeneous platforms and massive parallel architectures such as modern GPUs. Instant radiosity is a two phase rendering method. The first preprocessing phase creates *virtual point lights* (VPLs) by tracing photons in random directions starting at the position of each primary light source. The fact that each traced light path is independent makes this phase fully parallelizable and suitable for execution on GPUs. In addition, this phase can also be performed offline reducing the render times even further. The second phase is used to light the scene using previously created VPLs.

We compare the accuracy of our renderer with the ground-truth data obtained from Radiance. Three basic scenes sharing a single IES photometric light source are used in the tests. All materials are modeled using a Lambertian material (Lambertian BRDF).

In Section 2, we briefly introduce the basic concepts and theory related to photometric rendering. In Section 3, we describe our implementation of the instant radiosity algorithm. Section 4 presents the testing methodology used to measure the quality of this implementation in terms of accuracy and performance. Finally, in Section 5, we conclude the paper and point out some of the possible improvements that could be applied to our algorithm.

2 Background

Radiometry is a branch of physics, which concerns with measurements of the electromagnetic radiation [2]. Since visible light is a specific subset of the electromagnetic spectrum, with wavelengths ranging from 380*nm* (ultraviolet radiation) up to 780*nm* (infrared radiation), it can be described using basic radiometric quantities.

However, as shown in Figure 1, the sensitivity of the *human visual system* is not uniform across the whole electromagnetic spectrum. This observation led to the introduction of *CIE luminous efficiency functions* linking radiometry and *photometry*, which is used to study the perception of radiation by the human eye.

[*]mchwesiuk@wi.zut.edu.pl
[†]rmantiuk@wi.zut.edu.pl

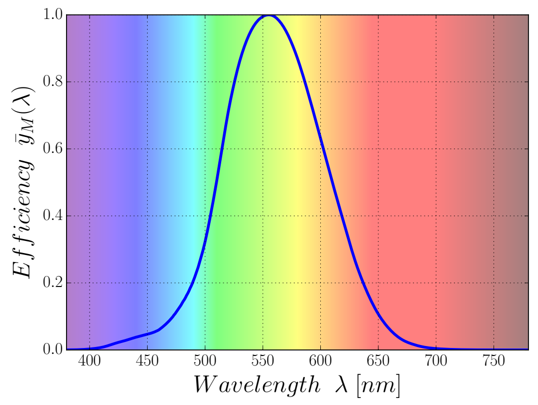


Figure 1: 1988 CIE Photopic Luminous Efficiency Function $\bar{y}_M(\lambda)$ (with Judd and Vos modifications). This plot was drawn based on data delivered by CIE.

Each radiometric quantity, like radiant intensity or irradiance, can be converted to its corresponding photometric counterpart by integrating against the CIE luminous efficiency function $\bar{y}(\lambda)$. The integration is necessary because the measured light can be a combination of wavelengths in a *spectral power distribution*.

While luminance is the most useful quantity in many areas in computer graphics, in lighting design communities the most important quantity is illuminance E. It is a measure of luminous flux Φ $[lm]$ incident on a surface area *A [m^2]*. The SI unit of illuminance is *lux [lx]*.

Photometric rendering besides focusing on physically based simulation of light, is heavily relying on data measured from real light sources. One of the most popular file formats describing *photometric light sources* are IES files (Illumination Engineering Society file format [4]). Such files are published online by most luminaire manufacturers. They provide a luminous intensity distribution measured in an *Ulbricht Integrating Sphere* or using a *Goniophotometer*.

2.1 Photometric rendering

Photometric rendering is a term used to describe unbiased rendering algorithms, which focus on computing the illuminance of the physical light delivered to certain regions in the scene. Since the *rendering equation* [6] has no analytic solution, in order to solve it, a *Monte Carlo integration* must be used. The difference between the expected value of used Monte Carlo estimator and the actual value of the integral is called *bias*. If this bias is equal to 0, then such an estimator is called *unbiased*. In this case the only error is due to the estimator variance which can be reduced by simply increasing the number of samples taken.

Biased renderers have many interesting properties, like faster convergence speed, but they usually introduce systematic errors. However, all unbiased renderers that attempt to solve the rendering equation might be still considered as biased to some degree since this equation is an approximation that ignores, for example wave phenomena like diffraction, polarization, or interference.

2.2 Rendering frameworks

There is a number of available renderers suitable for lightning design computations. This subsection briefly evaluates those rendering frameworks.

Radiance[1] is a collection of small, open-source, and cross-platform applications that follow the Unix philosophy. Such approach makes the whole rendering package unique and very versatile. It uses Monte Carlo techniques to estimate the lighting distribution. Authors claim, that the accuracy of their renderer has been scientifically validated. Although Radiance lacks GPU acceleration. *Mitsuba*[2] is an open-source, research-oriented, physically based modular rendering framework in the style of *Physically Based Rendering book*[3]. It supports both biased and unbiased algorithms, which are implemented in portable C++. One of the supported integration algorithms is Instant Radiosity. *LuxRender*[4] is an open-source physically based renderer derived from *PBRTv2*[5]. The GPU acceleration is currently experimental but gives very promising results. LuxRender has unique Light Groups feature that allows users to edit light properties without re-rendering the scene. *OctaneRender*[6] is an unbiased physically correct renderer. It has been implemented to exploit parallel compute capabilities of modern CUDA-powered GPUs. It supports custom shaders written in Open Shader Language and uses OpenVDB for particle simulation. *Arion*[7] is a stand-alone unbiased physically-based renderer with hybrid CPU and CUDA-GPU acceleration. It comes with a pre-made BSDF material library. *Indigo Renderer*[8] is an unbiased photorealistic renderer. It supports CUDA and OpenCL-enabled devices for rendering acceleration.

3 Instant Radiosity Renderer

3.1 Renderer architecture

Figure 2 shows the high-level diagram of our renderer architecture. At the beginning, the XML scene description and all referenced IES and 3D mesh files are being imported. The XML document contains camera properties, placement and orientation of IES light sources and description of triangulated mesh instances. This description also contains the material parameters needed for the BRDF used by each mesh.

[1]https://www.radiance-online.org/
[2]http://www.mitsuba-renderer.org/
[3]http://www.pbrt.org/
[4]http://www.luxrender.net/
[5]https://github.com/mmp/pbrt-v2
[6]https://home.otoy.com/render/octane-render/
[7]http://www.randomcontrol.com/arion
[8]http://www.indigorenderer.com

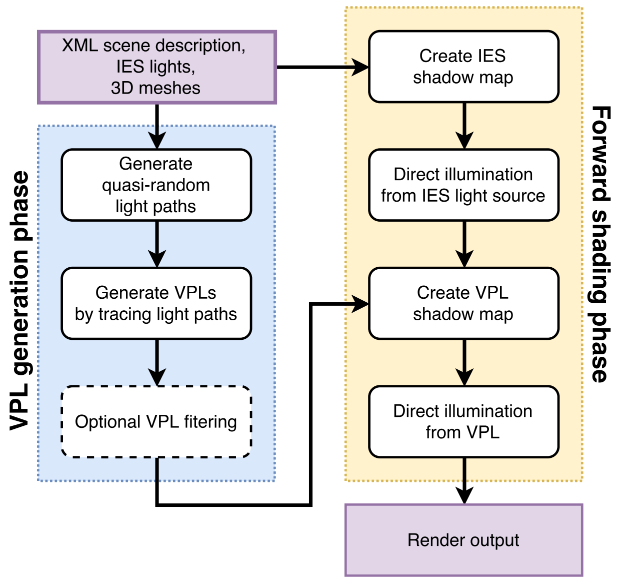

XML scene description, IES lights, 3D meshes
VPL generation phase
Generate quasi-random light paths
Generate VPLs by tracing light paths
Optional VPL fitering
Forward shading phase
Create IES shadow map
Direct illumination from IES light source
Create VPL shadow map
Direct illumination from VPL
Render output

Figure 2: Schematic diagram of our instant radiosity renderer.

Then, quasi-random light paths are being generated for each primary light source (see Sections 3.3 and 3.5). Once the light paths are generated, the energy of each light source is distributed across the scene. At each light path vertex, a virtual point light is created with the intensity proportional to the energy carried by the corresponding ray.

Once all the light paths are exhausted, the optional VPL filtering step takes place. This phase was introduced to remove all VPLs with zero intensity from further processing. Our benchmarks showed that this step greatly reduces the rendering times without introducing any bias.

At this moment, it is possible to start the rendering process. Forward shading is used to gather the contribution of IES and VPL lights and calculate the illuminance based on the used physical material model. In our case this is a perfectly diffuse model using Lambertian BRDF (see Section 3.4). At the end the image is added to the accumulation buffer.

We utilize the RadeonRays library (previously known as AMD FireRays) for ray casting on GPU compute units. It is heavily used during the VPL generation phase to calculate the ray intersection points, but also in our reference ray tracing render back-end for determining visibility using occlusion queries.

3.2 Rendering equations

Our algorithm calculates the direct and indirect illumination terms separately exploiting the linearity of light. This allows to directly accumulate the contribution of photometric light sources without approximating them with VPLs. This approach greatly increases the accuracy of direct illumination. We use the following equation to compute the illuminance originated from photometric light source:

$$E = \frac{I \cos\theta}{r^2}\ [lx], \tag{1}$$

where I is the luminous intensity of the light source, r depicts distance from the light source to surface, and θ is the angle between the light ray and surface normal (see Figure 3).

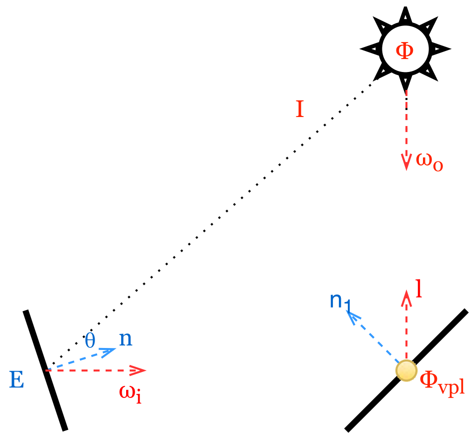


Figure 3: Simplified notation for rendering equation from Section 3.2.

The indirect term is approximated using VPLs. The Equation 2 is used to calculate the luminous power Φ_{vpl} of each VPL created at vertices of each traced light path:

$$\Phi_{vpl} = \frac{\rho\ \Phi\ |\hat{l} \cdot \hat{n}|}{p(\hat{\omega}_o)\ N}\ [lm], \tag{2}$$

where

- $\rho \in [0;1]$ is the reflectivity of the surface,
- $\hat{l}$ is a unit vector towards the light source,
- $\hat{n}$ is the surface normal vector,
- $p(\hat{\omega}_o)$ is the probability of emitting in direction $\hat{\omega}_o$,
- N is the total number of light samples taken.

The probability $p(\hat{\omega}_o)$ is received from the light's luminous intensity distribution after normalizing it by the total power of the light source Φ. Such power Φ is computed by integrating the luminous intensities I over all possible directions, as shown in the following equation:

$$\Phi = \int_\Omega I(\hat{\omega}_o)\ d\hat{\omega}_o\ [lm]. \tag{3}$$

Illumination of the surface originated from VPLs is calculated using the following equation:

$$E = \sum_{i=0}^{N} L_i(x, \hat{\omega}_i) \frac{|\hat{\omega}_i \cdot \hat{n}|\ |-\hat{\omega}_i \cdot \hat{n}_i|}{||x-y||^2} V(x,y)\ [lx], \tag{4}$$

where

- E is the illumination at shaded point x,
- N is the number of VPLs,
- $\hat{\omega}_i$ represents the direction towards the i^{th} VPL,
- $L_i(x, \hat{\omega}_i)$ is the luminance from i^{th} VPL,
- $\hat{n}$ is the normal at the shading point x,
- $\hat{n}_i$ is the normal at the surface on which the i^{th} VPL lies on,
- y is the position of i^{th} VPL.

The binary mutual visibility function $V(x,y)$ has been implemented using shadow mapping techniques.

3.3 IES-based light sources

We use the IES standard to describe the physical properties of light sources. The first version of this standard (defined by the LM-63 file format specification) was proposed in 1986 by the IESNA[9] and quickly became the industry standard for luminaire photometry. Later, in 1991 a new version replaced label lines with predefined set of keywords. The standard was revised again in 1995 to introduce new keywords, add support for near-field photometric data and deprecate the "ballast lamp factor" [1].

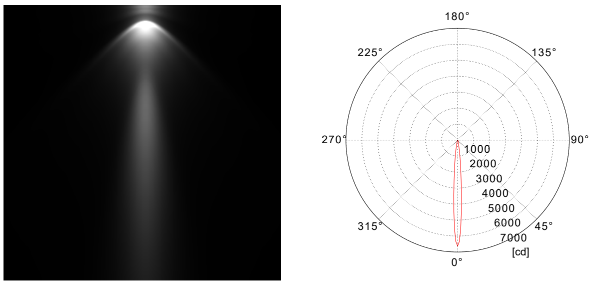


Figure 4: Example rendering of *Philips MASTERline ES 20W 8D* IES light source and the corresponding polar plot.

Such files are simple ASCII text files with `ies` extension. They contain a short description of the luminaire, a set of angles which describe the photometric web used for measurements (see example in Figure 4), and the values of luminous intensities in those directions. However, this format treats all light fixtures as point light sources, which might introduce errors. In order to minimize this error down to 1% it is highly recommended to use Lambert's *five times rule*, which says, that the distance to the light source should be greater than five times the largest dimension of the source [9]. This rule also helps to avoid the singularity in Equation 4, preventing the denominator approaching zero.

Our implementation parses such IES files and builds a floating-point lookup texture, which will be used in fragment shaders to perform Hermite interpolation using GPU texture samplers. This approach enables us to express the luminous intensity distribution function $I(\hat{\omega}_o)$ by mapping the *spherical coordinates* of outgoing direction $\hat{\omega}_o(\theta_o, \phi_o)$ as two dimensional texture coordinate $(\vec{u}, \vec{v})$.

[9]Illumination Engineering Society of North America

3.4 Physical materials

Since our algorithm supports only fully opaque materials, we can use *Bidirectional Reflectance Distribution Function* [12]. From energy conservation, it follows that such functions must have values only in the $[0,1]$ interval. In addition, such functions must satisfy the Stokes-Helmholtz reciprocity principle [5].

Since all surfaces in our scenes exhibit diffuse reflections only, we use a Lambertian BRDF to model the material response:

$$f_r(\hat{\omega}_i, \hat{\omega}_o) = f_r(\hat{\omega}_o, \hat{\omega}_i) = \frac{\rho}{\pi}, \tag{5}$$

where $\hat{\omega}_i$ is the direction of the incoming light, $\hat{\omega}_o$ is the direction of the light reflected from the surface and ρ depicts the surface reflectance coefficient.

3.5 Implementation

As shown in Figure 5, our implementation is split into smaller command-line utilities following Radiance's architecture. This approach enabled us to validate each functionality in isolation and reduce code duplication. Furthermore, it gives more advanced users the scripting abilities and allows them to insert or replace any application in the rendering pipeline.

Nearly all utilities are implemented using C++, following the *Orthodox C++* guidelines [7]. However, we also maintain Python scripts for less compute intensive tasks, for example, for comparing rendered images and generating plots.

The VPL generation phase described in Subsection 3.1 is implemented as `vplgen` program. This program is heavily relying on RadeonRays for accelerating the ray tracing of light paths. All light paths are scheduled for execution by RadeonRays' kernels at the same time. In order to hide the latency, we also implemented a batching mechanism using *OpenMP*, that takes advantage of all available CPU cores. All generated VPLs are then written to *standard output*, which can be redirected to file or to another utility, like `vplvis` shown in Figure 6. If the VPL output is redirected to a file, we call this file "VPL cache".

To generate random light paths, we randomize a point on an unit disk using a *Low-discrepancy Hammersley sequence* and map those points to an unit hemisphere. Resulting sample distribution has been shown in Figure 7.

It is possible, that VPLs with zero intensity will be created. This is the case when a given light path does not hit any surface or the energy of such light path becomes zero. Despite the fact, this side effect enables additional features

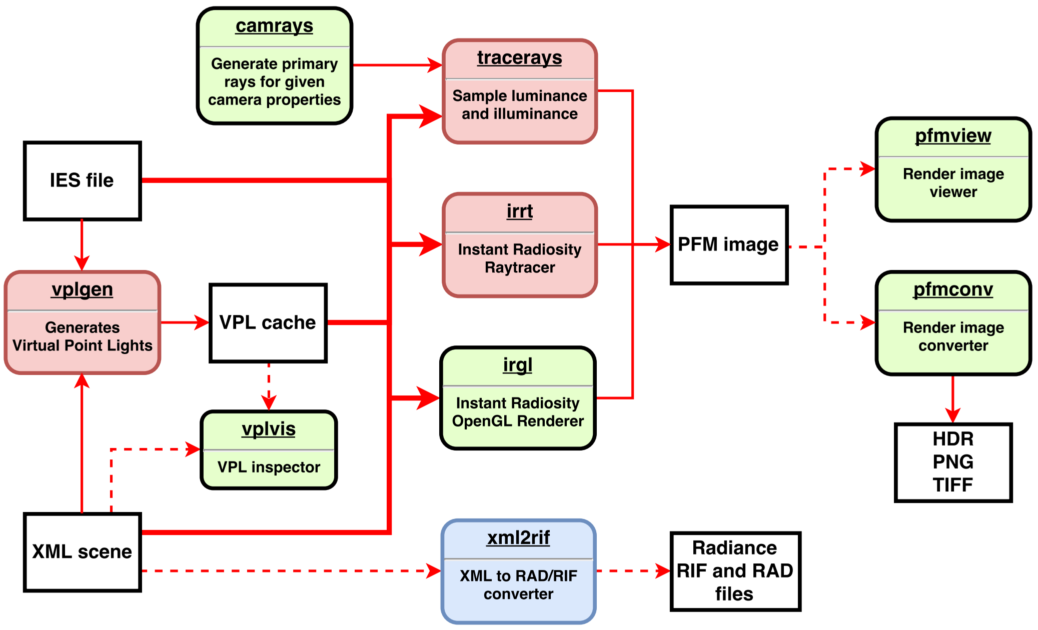

camrays
Generate primary rays for given camera properties
tracerays
Sample luminance and illuminance
IES file
irrt
Instant Radiosity Raytracer
PFM image
pfmview
Render image viewer
pfmconv
Render image converter
vplgen
Generates Virtual Point Lights
VPL cache
irgl
Instant Radiosity OpenGL Renderer
vplvis
VPL inspector
HDR
PNG
TIFF
XML scene
xml2rif
XML to RAD/RIF converter
Radiance RIF and RAD files

Figure 5: Data flow in our implementation. Green boxes are implemented in C++, blue boxes are Python scripts and red boxes are C++ applications using RadeonRays. Dashed lines represent optional data flow.

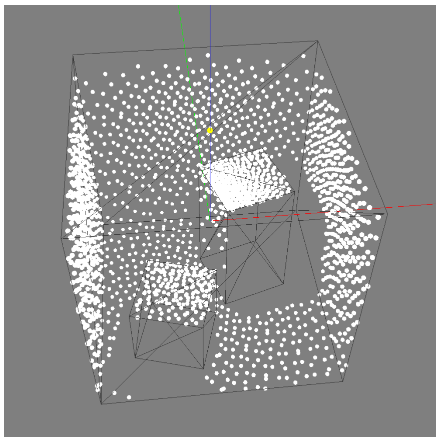


Figure 6: Screenshot from `vplvis` utility showing the distribution of 6803 generated VPLs.

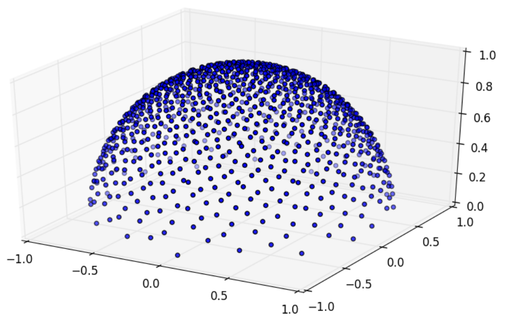


Figure 7: Example distribution of 1000 cosine weighted samples used for sampling IES light sources. The density of samples depends on the number of traced light paths.

- such as editing light intensities without the need for updating the light paths. In order to avoid wasting CPU and GPU cycles on processing such VPLs, an optional thresholding mechanism was introduced. If used carefully, it can filter out those VPLs without introducing any bias.

The OpenGL renderer `irgl` is running in a forward-compatible OpenGL 4 Core Profile. In order to fulfill the interactivity requirement, the illuminance is integrated across multiple frames. This iterative approach allows for faster inspection without waiting for the rendering process to complete. Shadow mapping has been utilized for computing the occlusion, where for each VPL we write the linear depth into a cubemap render target. This technique is commonly used in real-time game engines for casting shadows from point light sources [3].

4 Accuracy and Performance Tests

Our main goal is to provide an reasonable alternative algorithm for lighting design applications. It should be as accurate as possible, while still maintaining interactive frame rates. In order to test the quality of our renderings, we will compare both the accuracy, and the performance of our renderer with Radiance.

4.1 Testbed and stimuli

Both accuracy and performance tests were executed on a personal computer running Arch Linux distribution. This unit was equipped with a quad-core, Intel Core i7-2600K

CPU with 32 GB of system memory and an AMD Radeon HD 6950 GPU.

We prepared an automated testing framework in the form of a simple collection of Bash and Python scripts. This framework takes care of measuring the average execution time with the help of the `time` command. All tested applications were measured 10 times, while deleting all temporary and generated files between runs to make sure that each measurement is taken in the same initial conditions. We stored copies of the rendered images, in order to compare the rendering quality in different configurations. They will be compared using the *mean squared error* (MSE) metric.

We used the `rad` utility from Radiance to acquire the all reference images. It was invoked with a *rif* file, translated with `xml2rif` ahead of time from our XML scene file, with quality set to *high*. The resulting image was generated as a HDR file which contains irradiance values. Since our renderer uses only photometric units, we preprocessed the HDR files with a Python script that loaded the output image, converted irradiance to illuminance and saved the converted results to a `mat` file.

The accuracy tests were performed on two different scenes, that focus on different aspects of rendering. All of them share a single photometric IES light source and a pin-hole camera with 45° horizontal field of view. The first scene (see Figure 8, left) was used to test the accuracy of direct illumination without the surface inter-reflections. The camera has been placed in the shadow volume to make sure that the light source is not visible. The second scene (see Figure 8, right) was used to test the accuracy of indirect illumination calculated on the *P2* plane located in the shadow. This plane was illuminated exclusively by the light reflected from the plane *P1*.

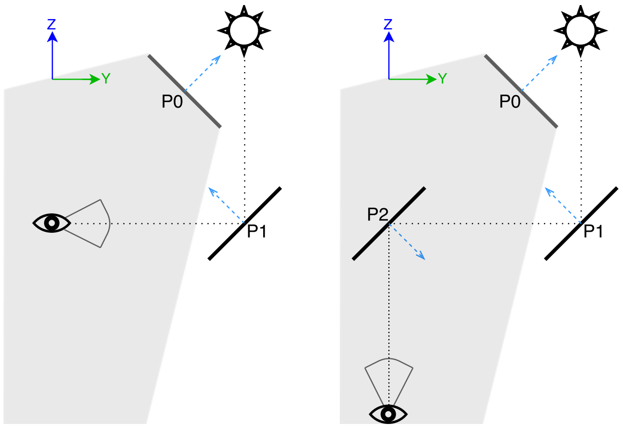


Figure 8: The layouts of the test scenes. Black lines represent $2m^2$ planes. Dashed blue arrows depicts directions of the normal vectors.

4.2 Accuracy tests

We noticed that the accuracy of the illuminance calculation is much worse at the edges of the polygons. In lighting design applications, the light intensity is not computed for the whole scene, but on specified calculation planes consisting of a set of points. Generally, these points are not located at the edges of the objects (e.g. interior of the desk surface is more important than its edges). Therefore, we crop the resulting renderings to focus only on the region of interest and do not exaggerate the mean error.

The results presented in Figure 9 show that our renderer computes illumination values for the direct illumination with accuracy very close to Radiance. The average mean-squared-error (MSE) for the selected region (see Figure 9, bottom) is less than 0.004 *lx*.

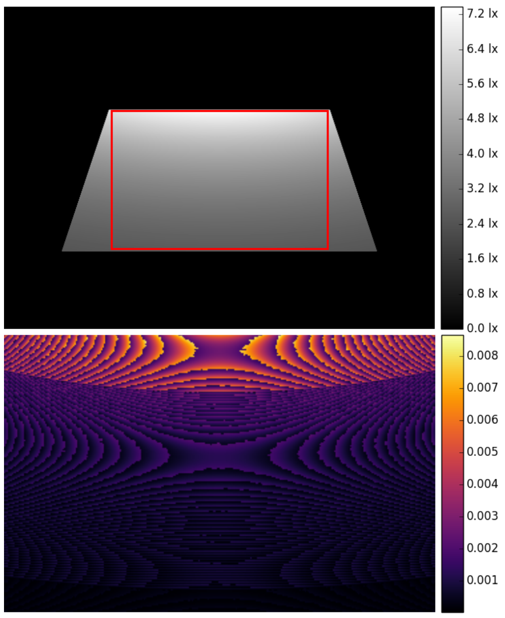


Figure 9: Top: rendering of the scene with the direct illumination generated by `irgl` using shadow map of a 128x128 pixel resolution (without VPLs). The red rectangle depicts the cropped region for which the MSE was calculated. Bottom: MSE difference between results obtained in `rad` and `irgl`.

In Figure 10 the results for indirect illumination are presented. The maximum MSE difference is below 6 *lx* and average error equal to 0.55 *lx*. The difference map in Figure 11 reveals the local difference between the instant radiosity and rendering technique used in Radiance.

Using Monte Carlo estimation, the MSE difference should decrease with increasing number of VPLs. To test this issue we analyzed the `irgl` results generated for various numbers of VPLs. The results presented in Figure 12 show that for more than 1000 VPLs there are no significant increase of the accuracy. The plot in Figure 12 also reveals that shadow maps of higher resolution than 128x128 pixels do not improve the results.

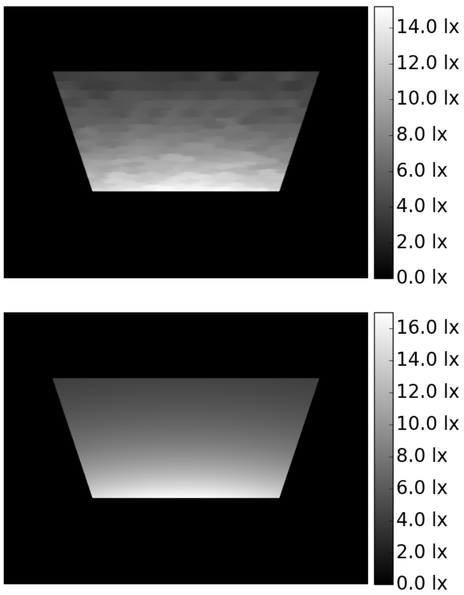


Figure 10: Results for indirect illumination computed in Radiance (top) and `irgl` (bottom) (for 1000 VPLs and a 128x128 pixel shadow map).

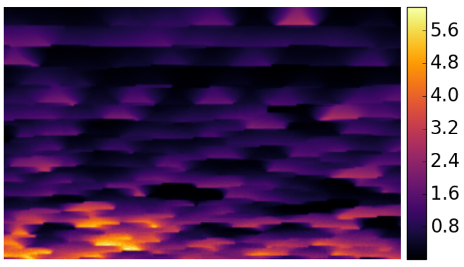


Figure 11: The MSE difference between Radiance and `irgl` for indirect illumination (see renderings in Figure 10).

In the `irgl` renderer we do not create VPLs for the second and further light bounces from the surfaces. This issue seems to cause the main inaccuracies in the surface illumination computations. The *P2* plane from Fig. 8 should be additionally illuminated by the light reflected from the bottom of the *P0* plane, which is not the case in the current version of our renderer. We plan to add this feature in the future work.

4.3 Performance tests

The performance was tested based on another scene - Cornell Box with two cubes (see Figure 13). We could not test the accuracy based on this more complex scene, because the current version of the instant radiosity renderer generates VPLs only after first reflection. As a result the Cornell Box image is much darker than results generated in Radiance (compare images in Figure 13). On the other

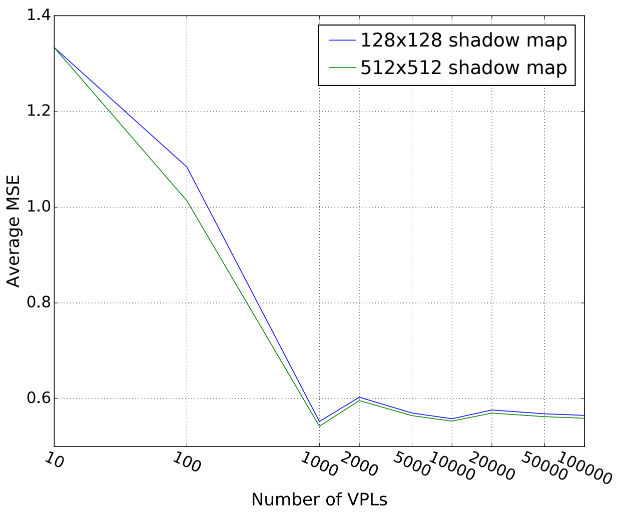


Figure 12: MSE difference for increasing numbers of VLPs and different size of the shadow map.

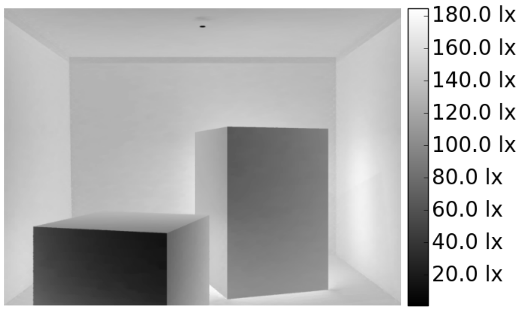
hand, the performance measurement for Cornell Box are more representative than for simple scenes presented in Figure 8. Cornell Box scene enforces more complex ray paths comparable to that used in the lighting design applications. We are aware that the presented results have only illustrative meaning but we include them because they allow to estimate a possible performance boost.

As we develop the renderer for the lighting design applications, the main goal is to calculate illuminance on selected surfaces. The renderer should allow to recalculate the results after light source modification (changing intensity and/or location) as fast as possible. In Table 1 we summarize the performance of the renderer modules. The total rendering time below 20 ms is significant, however, RadeonRays needs additional 193 ms to read the scene and IES light source, and 462 ms for initialization. `irgl` also must be initialize by reading the scene, which takes 187 ms.

Rendering phase	Time
`vplgen`	
light rays tracing and VPL generation	17.00 ms
`irgl`	
shadow maping	1.39 ms
shading	1.07 ms
accumulation and blending	0.19 ms
total	19.65 ms

Table 1: The instant radiosity renderer performance measured for different phases of rendering using shadow maps of 128x128 pixel resolution and 1000 VPLs.

Radiance requires 1625.68 ms to initialize and render the Cornell Box scene. Our renderer even including initializations is twice faster than Radiance. We plan to speed-up the setup phases of our renderer and advantage of repeating calculations without costly preprocessing. Inter-



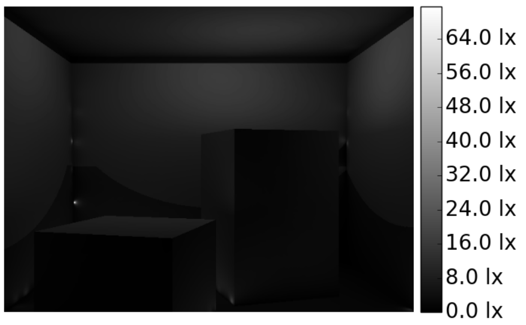


Figure 13: The rendering of Cornell Box generated by Radiance (top) and `irgl` (bottom).

estingly, shadow mapping for a large number of VLPs can be more costly than the ray tracing technique. We plan to investigate this approach in the future work.

5 Conclusions and Future Work

In this work we present implementation of a renderer based on the instant radiosity technique. The goal of this renderer is to calculate the physically correct illumination levels on the selected calculation planes rather than render images covering the whole light field visible from the camera. We compare the illuminance levels calculated by our renderer with Radiance, which is the industry standard for the lighting design applications. The results for a simple scene reveal the MSE difference of about 40%, which indicate the need for further improvements. The results of the performance tests have rather informative meaning but show that using the instant radiosity technique a significant speed-up of the illuminance calculation can be achieved.

In future work we plan to implement the full version of the instant radiosity algorithm with the VPL generation after second and further bounces. We will test various, not only regular, distribution of light intensity from the light sources using the IES format. These modification should improve the accuracy of the illuminance calculation. Then, we plan to speed-up the rendering replacing the shadow map generation with the ray tracing technique.

References

[1] Ian Ashdown. Parsing the iesna lm-63 photometric data file. `http://lumen.iee.put.poznan.pl/kw/iesna.txt`, March 1998.

[2] Ian Ashdown. Photometry and radiometry: A tour guide for computer graphics enthusiasts. `http://www.helios32.com/Measuring%20Light.pdf`, October 2002.

[3] Philipp S. Gerasimov. Chapter 12. omnidirectional shadow mapping. `http://http.developer.nvidia.com/GPUGems/gpugems_ch12.html`, 2004.

[4] Zumtobel Lighting GmbH. *The Lighting Handbook*. Zumtobel Lighting GmbH, 2013.

[5] Naty Hoffman. Background: Physics and math of shading. `http://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_notes.pdf`, 2013.

[6] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[7] Branimir Karadzic. Orthodox c++. `https://gist.github.com/bkaradzic/2e39896bc7d8c34e042b`, January 2017.

[8] Alexander Keller. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[9] J.H. Lambert and E. Anding. *Ostwalds Klassiker der exakten Wissenschaften:*. Lamberts Photometrie: (Photometria, sive De mensura et gradibus luminis, colorum et umbrae) (1760). W. Engelmann, 1892.

[10] Matt Pharr and Greg Humphreys. *Physically Based Rendering: from Theory to Implementation*. Morgan Kaufmann, first edition, 2004.

[11] Gregory J Ward. The radiance lighting simulation and rendering system. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 459–472. ACM, 1994.

[12] Chris Wynn. An introduction to brdf-based lighting. `http://www.cs.princeton.edu/courses/archive/fall06/cos526/tmp/wynn.pdf`, 2006.