CGI2012 manuscript No.
(will be inserted by the editor)

Interactive rendering of urban models with global illumination

Oscar Argudo · Carlos Andujar · Gustavo Patow

Abstract In this paper we propose a photon mapping-based technique for the efficient rendering of urban landscapes. Unlike traditional photon mapping approaches, we accumulate the photon energy into a collection of persistent 2D photon buffers encoding the incoming radiance for a superset of the surfaces contributing to the current image. We define an implicit parameterization to map surface points onto photon buffer locations. This is achieved through a cylindrical projection for the building blocks plus an orthogonal projection for the terrain. An adaptive scheme is used to adapt the resolution of the photon buffers to the viewing conditions. Our customized photon mapping algorithm combines multiple acceleration strategies to provide efficient rendering during walkthroughs and flythroughs with minimal temporal artifacts. To the best of our knowledge, the algorithm we present in this paper is the first one to address the problem of interactive global illumination for large urban landscapes.

1 Introduction

In the last decades we have witnessed enormous improvements both in urban modeling, mainly through procedural techniques, and in the interactive rendering of large scenes. However, the target of accurate rendering of large urban scenes has been barely touched until now. Applications like GoogleMaps and Nokia Maps are good examples of this, where only direct illumination is used for all kinds of visualizations. A more general solution, including inter-reflections and other indirect illumination problems should be considered in order to achieve an accurate visualization of urban environments under changing lighting conditions.

The algorithm we propose is specifically designed to take advantage of the intrinsic characteristics of urban landscapes to achieve interactive rendering and navigation capabilities, through a specifically tailored photon map algorithm. This tailoring includes a novel parameterization for the city blocks, a persistent hierarchical data structure for storing the photons, and an adaptive photon splatting strategy.

Basic ingredients of our approach include:

- An implicit parameterization of the block facades allowing the real-time construction of persistent photon buffers providing constant-time gathering.
- A splatting strategy providing anti-aliased photon maps. Unlike competing approaches using screen-space photon maps, we splat photons onto all surfaces contributing to the final image, including occluded surfaces and out-of-frustum surfaces that can be seen through mirror surfaces.
- The use of per-block visibility flags to optimize space allocation and write operations on photon maps while preserving the consistency of photon mapping.
- An adaptive photon mapping strategy combined with a form of algorithmic level of detail.

Our prototype implementation over the NVidia's OptiX ray tracing engine [17] allows real-time progressive rendering of urban landscapes containing thousands of buildings in commodity hardware. Full source code of our implementation is provided[1].

O. Argudo and C. Andujar
MOVING Group, ViRVIG, Universitat Politècnica de Catalunya, Barcelona, Spain

G. Patow
GGG, ViRVIG, Universitat de Girona, Spain

[1] `http://www.lsi.upc.edu/~virtual/urban/`

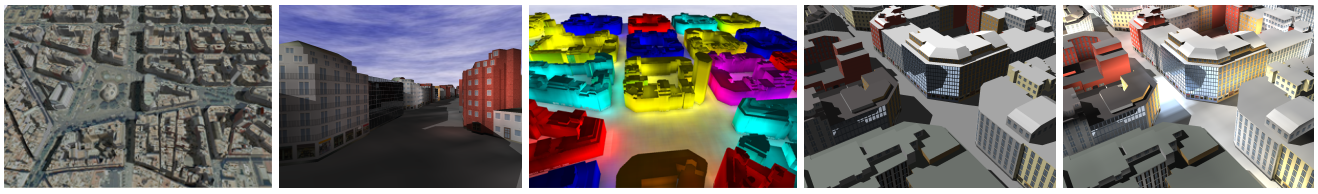

Fig. 1 Urban scene rendered in real-time with our approach. The fourth image (with no indirect diffuse light inter-reflections) is provided for comparison.

2 Previous work

2.1 Real-time city rendering

In the last years a number of algorithms for real-time rendering of urban environments have been proposed [5, 7, 2]. These approaches achieve interactive frame rates by adapting traditional acceleration strategies (level-of-detail, image-based rendering and visibility culling) to the particular properties of city models: 2.5D overall shape, plane-dominant geometry, regular structure, dense occlusion, and large texture datasets.

The poor performance of traditional mesh simplification algorithms on urban models has motivated the development of algorithms specifically designed for buildings [10, 8] and groups of buildings [4, 28], most of them generating discrete LoD representations.

Several image-based representations have been proposed to accelerate the rendering of distant buildings, including impostors [15], textured depth meshes [21], and point-based impostors [27]. Some recent approaches represent the geometry of 2.5D cities as hierarchies of displacement maps which are rendered using relief mapping techniques [5, 7, 2].

Despite all the above techniques, current approaches for large-scale urban rendering only support direct illumination plus shadow mapping. Indirect illumination effects in urban rendering are limited to environments maps and precomputed ambient occlusion [7], thus limiting photorealistic appearance under changing lighting conditions [25], see Figure 1.

2.2 Interactive global illumination

Global illumination has been one of the most active areas in Computer Graphics, and recently the interest for real-time techniques has grown rapidly. Here we are going to mention only those works more closely related to our approach, but the interested reader may refer to the book by Szirmay-Kalos et al. [24] or the survey by Ritschel et al. [19] for more on the subject.

Up to now, there has been considerable interest in developing interactive rendering solutions, like [26], or more recently [1]. Currently, there exist a few real-time ray-tracing solutions based on current GPU processing capabilities, like OptiX [17]. Our system is implemented on top of the latter one.

Screen-space techniques are specially appealing for GPU acceleration. Lavignotte and Paulin [14] proposed a two-pass screen splatting solution which considerably increased scalability with respect to previous approaches. Later, Ritschel et al. [20] estimated indirect illumination in image space, which is a scene-independent approximation. Dachsbacher and Stamminger [6] computed the secondary lights contribution by splatting photons in a deferred shading process, decoupling of rendering from scene complexity. Herzog et al. [12] splatted photons directly to eye samples, improving image quality with the same number of photons. Later, McGuire and Luebke [16] presented a hardware-accelerated method for global illumination by image-space photon mapping. Our technique does not work in screen-space, but uses a urban-specific, block-oriented parameterization which improves quality and provides persistence at the expense of extra memory storage. We also provide a progressive solution that improves image stability between frames and accelerates computations.

3 Urban Photon Mapping

3.1 Overview

The input of our urban rendering algorithm is a textured polygonal model consisting of building geometry, facade textures (real photos or procedural images) and orthophoto textures, which are aerial photographs representing nearly-horizontal surfaces (with normals close enough to the vertical direction) such as the ground, small sloped surfaces, ceilings and terraces. We assume that the buildings have been grouped into connected components that we call *blocks*, which will be our working unit. From now on through this paper, we will use the block level as our working urban unit.

For illumination computations, our objective is to find the final radiance L that an observer sees from every surface in the city. L is computed as $L = L_d + L_i$,

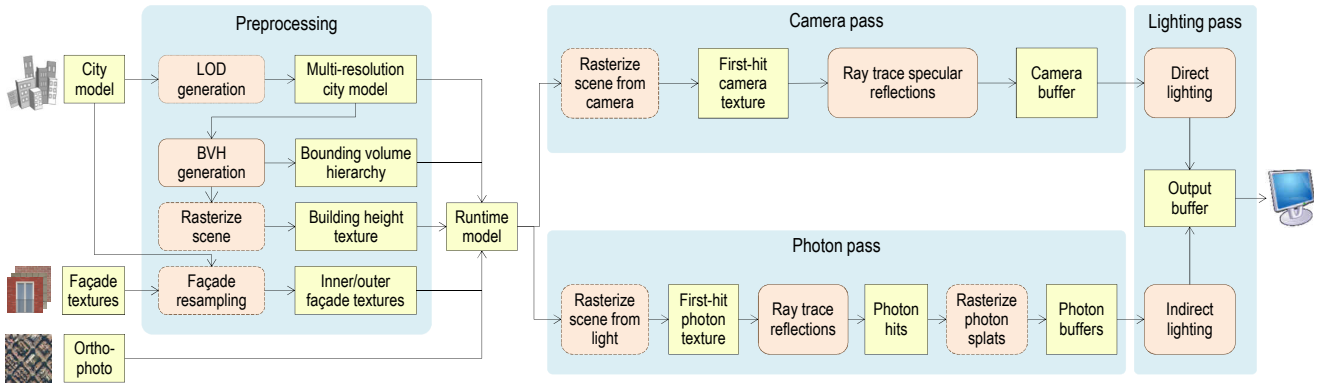

Fig. 2 Overview of our urban rendering pipeline. Our algorithm is structured into four main stages: Preprocessing (Section 3.2), Camera pass (ray tracing pass, Section 3.3), Photon pass (Section 3.4), and Lighting pass(Section 3.5), each one with its own sub-stages. All runtime steps are executed in the GPU, using OpenGL (dashed contours) and OptiX/CUDA (solid contours).

where L_d is the direct component and L_i the indirect one. The direct light contribution simply is $L_d = I_e \cdot K_d \cdot (\mathbf{N} \cdot \mathbf{L}) \cdot V(P, \mathbf{L}) + I_e \cdot K_s \cdot (\mathbf{R} \cdot \mathbf{V})^\alpha$, where I_e is the light intensity at the city level, $\mathbf{N}$ is the unit normal, $\mathbf{L}$ is the unit light vector, $\mathbf{R}$ is the reflected vector, $\mathbf{V}$ is the viewer direction vector, α is the roughness coefficient, and $V(P, \mathbf{L})$ is the visibility function.

On the other hand, the indirect light is computed as $L_i = (\Phi_c \cdot K_d)/(\Delta A \cdot M)$, where Φ_c is the accumulated flux. ΔA is the area in world space of the photon buffer texel, and M is the number of emitted photons.

So, one important aspect is that the flux Φ_c is computed by splatting photons and accumulating their energy onto a collection of 2D photon buffers. Figure 2 shows the main steps of our algorithm, grouped into the three main passes of progressive photon mapping [9]: the *camera pass* (or ray tracing pass), where rays are traced from the camera into the scene, bouncing at reflective surfaces and storing the hits against diffuse surfaces (Section 3.3); the *photon pass*, which shoots rays (photons) from the light sources towards the scene, blending the photon colors with the information in the color texture as the photons traverse the scene, and storing the terminal hits in a photon map (Section 3.4), and the *lighting pass* which renders the final image combining direct and indirect lighting (Section 3.5). Our method follows all $LS^*D^*S^*E$ light paths [11], which cover most of the light effects in an urban environment.

A key ingredient of our approach is the photon map representation, which has important consequences for the photon and lighting passes. Unlike traditional photon mapping algorithms, which usually store photon bounces in a kd-tree [13], our photon map is represented with a collection of 2D buffers using a block-wise continuous parameterization. Using textures in object space was first proposed by Arvo [3], but here we use a specific city-oriented parameterization, which allows to greatly improve speed and quality. For nearly-horizontal surfaces we use a (nearly-isometric) orthogonal projection onto a ground plane; for the building facades within a block, we use a double cylindrical projection (one for the outer facades and one for the inner facades, if any). This parameterization is used for resampling texture facades into a low-resolution version (to be used for indirect illumination) and, most importantly, for storing and fetching the photons. This representation provides a number of advantages. First, we do not require to build or update the kd-tree every frame (which is often done in the CPU, with the consequent GPU-CPU-GPU data transfers). Second, since our parameterization is continuous over a building block, we can efficiently splat the photons onto the 2D buffer during the photon pass. This results in an anti-aliased photon map which can be sampled in constant time, rather than the $O(\log n)$ kd-tree search of traditional approaches (complexity is in terms of photons).

3.2 Pre-processing

The pre-process stage consists of the following tasks, see the corresponding entries in Figure 2: simplification of individual blocks and groups of blocks, generation of the data structures for accelerating ray-scene intersections, resampling of the facade textures, and generation of the block height texture. The first two steps are quite standard in urban rendering and ray tracing engines and thus will be described succinctly; the last two are specific to our method and are described in detail below.

Our rendering approach works with a LoD representation of the city. For this purpose, any of the simplification techniques in Section 2 can be used. We adopted

Fig. 3 Cylinders used for projecting block facades. The radii shown are somewhat arbitrary.

a simple technique where each block is represented with two levels: the original one, and a coarser one produced by vertically extruding a simplified version of the building using the convex hull of its footprint.

Concerning acceleration data structures, the regular structure of urban models suggests using a uniform grid-like data structure [23]. However, since our current system is implemented over OptiX, which does not support grids, we use instead bounding volume hierarchies (BVH) both for the polygons within a block, and for the blocks within the city. BVH construction and traversal is managed by OptiX and described elsewhere [17].

As mentioned, facades within a block are parameterized using a double cylindrical projection: exterior facades facing the street are parameterized onto a bounding cylinder, while the inner ones (i.e. the ones that face the interior courtyard, if any) are parameterized onto another one. The rationale of this projection is that, in modern urban areas, blocks follow a cylindrical distribution around a (possibly empty) courtyard (see Figure 3). The center of projection is chosen to be the centroid of the block bounding box. In more complex cases involving blocks with multiple courtyards, it is advisable to use as many interior parametric cylinders as there are courtyards. However, in our examples we have seen that a single map for a single courtyard is a reasonable supposition as courtyards tend to be large central spaces. For more involved cases, an automatic method for detecting the block structure would be needed [18].

The parameterization above maps block facades onto rectangular areas, which are accomodated within a large texture atlas. Since all blocks have the same size in texture-space, a block ID suffices for computing the chart's origin within the atlas. In addition to the traditional texture coordinates for the block vertices, we also use the cylindrical parameterization for representing the photon map and for resampling facade colors (used for indirect illumination only, as direct lighting is computed using the original unlighted facade textures). This resampling is accomplished by rasterizing each block facade into its own chart, using the cylindrical projection (see Figure 4). Front faces are projected onto the *outer facade texture* and back faces (if any) are projected onto the *inner facade texture*. Since urban models typically lack complex BRDF's, each texel stores just the K_d coefficients (in RGB channels), and a reflectivity flag in the alpha channel to handle light reflection at perfect specular surfaces such as windows. In the case of photon buffers, the allocation of block charts within the atlas is dynamic, to provide different chart resolutions for different blocks to generate adaptive photon maps (discussed in Section 3.6).

Fig. 4 Chart example from a resampled facade texture.

During the photon splatting step (Section 3.4) we need to know the height of each texel in the orthophoto to prevent splats hitting the floor from affecting texels on the ceiling and vice-versa. For this purpose we also render the scene using an orthogonal camera from a zenithal direction, adding the resulting depth as an extra channel to the original orthophoto texture.

3.3 Camera Pass

Generally speaking, this stage shoots rays from the camera to the scene, bouncing at reflective surfaces and storing the hits with diffuse surfaces onto a *camera buffer*. Our camera pass differs from traditional implementations in two aspects. First, we rasterize the scene (with depth test enabled) into a temporary texture (*first-hit camera texture*) to find the visible surfaces from the current camera; this step is performed with OpenGL adopting traditional acceleration techniques, including LoD rendering. This way we save all intersection tests with the primary camera rays, and we only need to ray trace (with OptiX) the reflections at specular surfaces (with a Fresnel's factor), stopping at the first diffuse surface encountered. Observe that we still use the same photon buffers and resampled facade textures, so consistency is kept independently of the LoD used. Second, we also store the block ID corresponding to all visible fragments. This piece of information is key for visibility optimizations during the photon pass (discussed below). The result of this stage is a *camera buffer* (as large as the output window) containing, for each pixel, the position, normal, (possibly attenuated) K_d coefficient and block ID of the first diffuse surface encountered.

3.4 Photon Pass

The photon pass deals with the propagation of discrete photons from the light sources towards the scene. Whenever a photon hits a surface, the intersection point and the incoming direction are recorded (except for the first bounce, which corresponds to direct illumination) and scattering is computed. Since we adopt a progressive rendering approach [9], each frame a fixed number of photons are emitted whose power is combined with that of previously propagated photons.

Our photon pass differs from traditional implementations in several aspects. We accelerate the computation of first bounces by rasterizing the scene with OpenGL from the sun position into a temporary screen-space texture (*first-hit photon texture*) which records, for each emitted photon, its first-bounce position, normal vector, diffuse and specular colors K_d and K_s, and a specularity flag. This step is not required to be executed every frame, as multiple photons can be reflected from each first-hit record during successive frames. Nevertheless, to avoid re-using exactly the same first-hit record every frame, we render to an oversized photon texture. If we intend to shoot $m \times m$ photons every frame, we render to a $4m \times 4m$ first-hit photon texture once, and let successive frames to re-use the same texture. Since the texture provides 4×4 more first-hit records than required at each frame, the actual first-hit record is reused only once every 16 frames.

Reflected photons are traced recursively as follows. When a photon hits a diffuse surface, we first check whether the surface is nearly horizontal (terrain, ceiling, terraces) or not (facades). In the case of facades, we use the block visibility flag (set during the camera pass) to check whether the block is visible from the camera, either directly or through reflective mirrors. If the hit surface is visible, we compute the cylindrical projection of the hit position and splat the photon onto the corresponding outer/inner facade photon buffer. The world-space splat radius r_w should be proportional to the traveled distance of the photon, but in practice the distance from the source (the sun) to the city is so large that the traveled distance can be considered constant. Therefore, its projection in texture-space r_t is assumed to be constant within a block (for a given frame), but changes from block to block as their projections change, which greatly simplifies kernel computation. The splat kernel is represented with a quad, and each of the n texels within the kernel with size r_t receives an energy flux proportional to r_t^{-2}, which is accumulated as Φ_c. In our implementation, we linearly adjust the trade-off between convergence rate and blurriness of indirect lighting, given by the number, size r_t and energy of the splatted photons, to obtain quicker but blurrier images during the first frames of progressive rendering, and nicer (and more accurate) renderings in the subsequent frames. Also, if the light position does not change and the user movement does not require a LoD change, we further re-use the photon buffers.

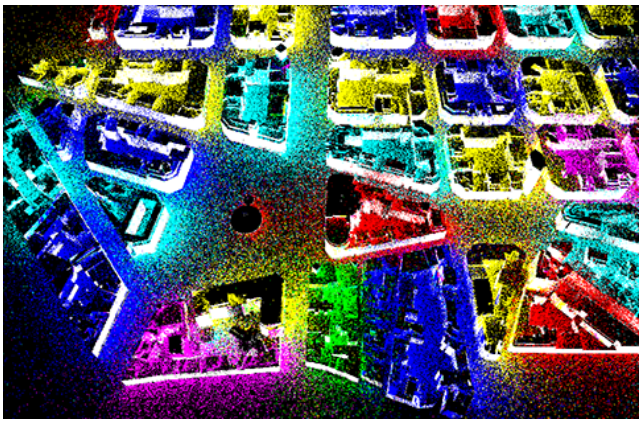

Fig. 5 Visualization of photon hits over the scene for a Sun light located to the right at an elevation of 45 degrees.

Photon bounces on horizontal surfaces are processed in a similar way but splatting occurs onto the zenithal photon buffers instead of on the outer/inner facade buffers. The hits can be visualized as simple dots, as shown in Figure 5, where each block has been painted with false color to illustrate the color bleeding effect.

To improve performance, we first store in a linear buffer all hits (in OptiX) and then we use traditional hardware rasterization to generate the splats and project them onto the photon buffers. All the information needed for later stages is additively blended at this point. This has proven to be significantly faster than software CUDA rasterization.

3.5 Lighting Pass

This stage produces the output image by adding the contribution of direct and indirect lighting, as explained in Section 3.1. Note that traditional photon mapping estimates the outgoing radiance at each visible point from nearby samples stored in a kd-tree, which is $O(\log n)$, while our implementation simply requires a lookup into the corresponding photon buffer, being $O(1)$.

3.6 Adaptive photon mapping

In order to improve accuracy for the parts closer to the viewer, we use adaptive-resolution photon buffers, giving these blocks closer the highest resolution for the

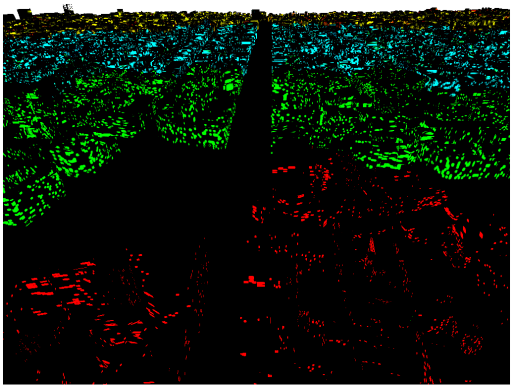

Fig. 6 Photon buffers for closer regions use higher resolutions. The image shows photon impacts colored according to the region (red = Level 0, green = Level 1, light blue = Level 2, yellow = Level 3. Level 4 buildings are too far away to be visible in this image)

photon buffers and the farthest ones coarser ones. Figure 6 depicts this idea. Also, we have observed that the impact of indirect illumination in very distant blocks can be neglected, and that a direct illumination approach suffices for blocks with a small screen footprint. We thus push adaptivity further by excluding from global illumination distant blocks, thus providing a form of algorithmic level-of-detail, which in general provides good results. In the camera pass we rasterize *the whole scene* (using geometric LoD) storing in the camera buffer an extra flag indicating whether the fragment comes from a distant city block. Reflected camera rays and photon rays are computed without considering distant blocks. The lighting pass computes the indirect lighting term as usual except for those pixels marked with the distant flag, where the indirect lighting term is approximated with a constant ambient value. This simplification greatly reduces computation times. Since distant blocks of buldings have a significant impact on both memory footprint and ray tracing cost, this approach improves performance and scalability without introducing perceptible artifacts in the final image.

For the sake of simplicity, our current prototype tries to maximize the occupancy of the different photon buffers, by dynamically classifying each city block into one of following categories, depending on their distance d to the observer and the average block diameter b:

Level 0 Close-up blocks in this level are assigned the highest resolution charts, and rendered at full geometric resolution. A block is here if it is completely inside the viewing frustum and has $d < 3b$.

Level 1 Nearby blocks here are assigned one half or one quarter the highest resolution, but still rendered at full geometric resolution. Here are the blocks that are inside the frustum and satisfy $d < 6b$.

Level 2 Same as the previous one, but for blocks within the frustum and that satisfy $d < 12b$, or the ones with $d < 3b$ but not completely inside the frustum. Note that nearby out-of-frustum blocks are likely to appear in mirror reflections and thus cannot be completely discarded.

Level 3 These middle-range blocks (within the frustum and with $d < 24b$) are assigned one eighth of the resolution, and rendered using a simplified mesh textured with the resampled facade texture.

Level 4 : Far-away blocks are rasterized in the camera pass using a simplified LoD (with the resampled texture), but excluded from the rest of the global illumination pipeline. All blocks not selected in the previous levels are here.

In our prototype implementation we use 512x512 facade photon buffers for each resolution. To simplify the mapping of chart coordinates to global photon buffer coordinates, level 0 accommodates 1 column of 10 blocks, and levels 1-3 accommodate 2 columns of 20 blocks, 4 columns of 40 blocks and 8 columns of 80 blocks, respectively. This accounts for a total of 850 building blocks for which global illumination is computed accurately. In a second step we assign blocks to levels according to the budget above (i.e. a maximum of 10, 40, 160 and 640 blocks at each level). This is done by traversing the priority queues according to distance, and inserting the block in the proper level. When the corresponding level is full, the next non-empty level is assigned (level 4 has no limit). Even for large cities with thousands of blocks, this LoD selection strategy has negligible overhead while maximizing the use of available resources.

4 Results

4.1 Implementation details

Platform: We have implemented a prototype version of the system over NVidia OptiX 2.1.1 [17] (source code is available as additional material). Running times were measured on an Intel Core i7 960 with 12 GB of RAM, equipped with a NVIDIA GeForce 570 with 1280 MB. For progressive rendering, the photon pass was configured to shoot 64 K photons every frame, which proved to be a good tradeoff between quality and speed.

Test dataset: We tested our approach with the 3D model of Barcelona from *Tele Atlas*. The city model consisted in 2,182 buildings and about 600K polygons. Facade textures were represented as a collection of 5,289 RGB textures occupying about 230 MB. We manually added a reflectivity channel to the textures containing windows. The resulting RGBA textures were grouped into 9

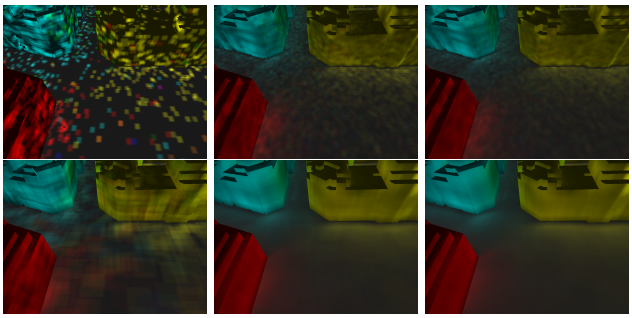

Fig. 7 Screen shots showing progressive rendering with small (top) and large (bottom) values for the blurriness of indirect lighting.

1024×1024 texture atlases to minimize texture switching during the scene rasterization steps.

Preprocessing: The preprocessing took about 30 min on the test hardware, including the generation of a simplified LoD version for each building block, the generation of a 1024×1024 height texture, and the resampling of the outer/inner facade textures onto 1024×1024 texture atlases. The whole city model plus its LoD levels needed less than 1GB of memory.

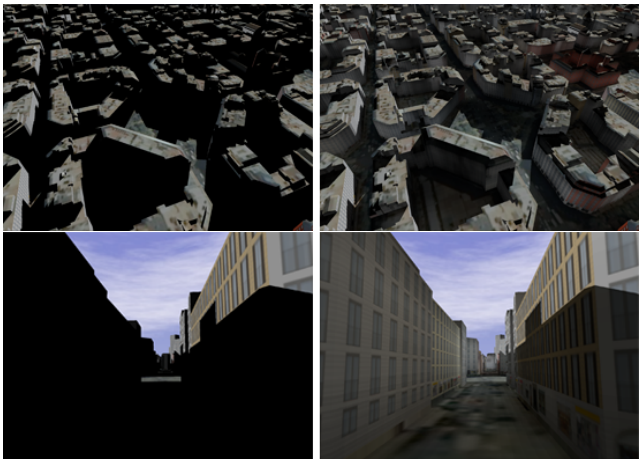

Fig. 8 Aerial and ground-level views comparing direct-only lighting and our global illumination approach.

4.2 Illustrative results

Figure 7 shows the progressive refinement of the photon buffer when the camera is still. Figure 8 shows several images rendered in real-time with our approach for both aerial and street-level views, showing mostly diffuse inter-reflections. Figure 9 shows reflection at mirror-reflective and mirror-refractive surfaces. Note that accurate and efficient simulation of perfect reflections is important in urban environments because substantial parts of block facades are often occupied by windows. See Figure 11.

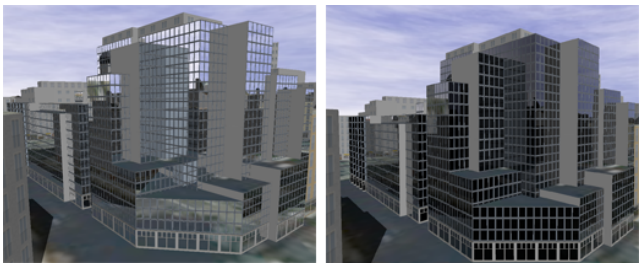

Fig. 9 Reflection at specular surfaces: as completely reflectant surface (left) and using the Schlick approximation of the Fresnel factor (right).

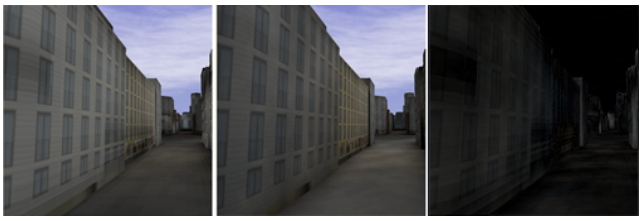

Fig. 10 Scene rendered with our method (left), with a kd-tree (middle), and the 2x scaled difference. Images computed with the same number of photons.

Figure 10 compares our results with that of a traditional photon mapping implementation at the same precision. For this purpose we used the NVidia's progressive photon mapping implementation accompanying OptiX SDK. Note that the image differences are hardly noticeable.

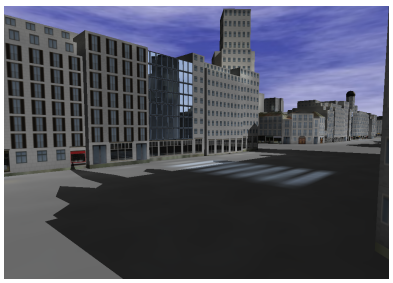

Fig. 11 In an urban environment light is often reflected at windows producing a noticeable lighting effect.

4.3 Performance

We have benchmarked our approach against the NVidia's progressive photon mapping implementation, which uses

		Camera pass			Photon pass				Lighting pass			Totals		
View	Algorithm	Render	Tracing	Total	Render	Tracing	Splatting	Total	Kdtree	Gathering	Total	Complete	Progressive	Slow-motion
Aerial	kd-tree	-	12.1	12.1	-	8.1	-	8.1	16.8	12.6	29.4	49.6	37.5	41.5
Aerial	ours	3.1	4.6	7.7	2.8	6.0	1.0	9.8	-	7.4	7.4	24.9	14.4	15.1
Street	kd-tree	-	17.0	17.0	-	8.1	-	8.1	16.8	7.6	24.4	49.5	32.5	41.4
Street	ours	2.7	4.2	7.0	2.7	5.7	0.6	9.0	-	3.6	3.6	19.6	9.9	10.6

Table 1 Performance comparison between our approach and a traditional photon-mapping implementation using a kd-tree. Running times in ms.

a kd-tree for density estimation. For the sake of fairness, running times were measured by rendering all geometric models at their original resolution, and global illumination was computed for all blocks, including distant ones, as using simplified versions would clearly favor our approach. The viewport size was 1024×768, and 64 K photons were shot every frame.

Table 1 shows the results of our benchmark for the test scene. Concerning the camera pass, our approach is 1.5-2.4× faster than the traditional implementation with our settings; and rasterizing the whole scene proved to be faster than tracing primary rays for finding the first hit records; also, our approach only raytraces camera rays reflected at specular surfaces. Regarding the photon pass, our approach is slightly (20%) slower. Since only 64K photons are shot every frame (as opposed to the 768K camera rays in the camera pass), the benefits of the first-hit speed-up are less noticeable. Note though that the photon splatting step has a negligible overhead. The highest speed-ups are achieved in the lighting pass: our approach is about 4-6× faster, mainly because of the $O(1)$ irradiance retrieval cost. Also, we avoid constructing the kd-tree for dynamic scenes (for static ones the kd-tree can be reused), which is also important.

Concerning total rendering times, we distinguish thee types of frames, depending on which passes are executed: (a) *complete frames*, which require all three passes; these frames are rendered during fast camera motion or when lighting conditions change, (b) *progressive frames*, which avoid unnecessary re-computations when the camera stops[9], and (c) *slow-motion frames*, where photon buffers from previous frames are re-used. Note that all types of frames require the lighting pass, which is indeed our most optimized stage. When rendering complete frames, our approach achieves a 2-2.5× speed-up, providing about 40-50 fps. These speed-ups are even higher (2.5-3.2×) when rendering progressive frames. Furthermore, the omni-directional nature of our photon buffers allows for a limited re-usability during slow camera motion. In this case, the speed-ups are even more significant, up to 4×.

4.4 Comparison to other photon mapping approaches

Screen-space techniques: Thanks to our parameterization, our approach provides multiple advantages over screen-space methods. First, our photon buffers enable us to store photons at occluded or out-of-frustum surfaces that contribute to the final image through reflective mirrors (thanks to our per-block visibility flag). In general, screen-space techniques will usually be missing this computation, or would require special treatments [16] that would severely impinge on performance. This reuse makes one think of comparisons between our technique and PRT [22], but this is a technique for low-frequency static environments, while ours can be used in a larger range of frequencies and dynamic scenarios. Also, it is important to mention that our technique considerably reduces flickering as the photon buffer reuse adds coherence from one frame to the following one, without producing noticeable artifacts. Second, competing screen-space methods need to evaluate a scatter filter at every visible pixel affected by the splat, typically involving a fragment shader for computing the kernel. Since our splats always touch texels corresponding to world-space neighboring points, we can compute a single kernel per splat instead of per fragment. Third, screen-space photon maps need to be regenerated every frame during camera motion, as they encode incident radiance only for visible pixels. Our photon buffers encode irradiance in a much more view-independent manner, allowing us to re-use the photon buffers during multiple frames (thus saving the photon pass), provided that the camera does not deviate drastically from that used in the last update. This makes our approach much more efficient for walk-through and fly-through applications, as well as for head-tracking and 3D stereo viewing. On the downside, our photon buffers are less memory-efficient, although thanks to our adaptive resolution scheme they still fall in the same order of magnitude than screen-space maps.

Traditional photon mapping: As shown in the previous section, our approach is faster than kd-tree implementations, as we save the kd-tree construction step and we can benefit from hardware rasterization to splat pho-

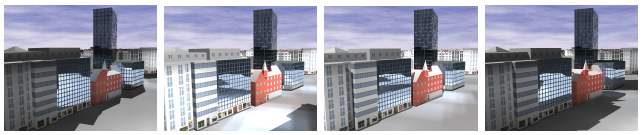

Fig. 12 Our approach allows for real-time editing of lighting conditions.

tons onto 2D photon buffers, resulting in a $O(1)$ density estimation instead of the $O(\log n)$ kd-tree gathering. Unlike our approach, traditional progressive photon mapping offers little options for re-using the photon maps under slow camera motion, as each gathering pass is used to accumulate photon power only at the hit points found in the camera pass [9]. On the downside, kd-tree implementations provide support for high-frequency indirect lighting effects such as caustics.

5 Conclusions & Future Work

We have introduced a new algorithm for global illumination in urban landscapes. The usage of specially tailored photon maps allows a considerable improvement over traditional *kd-tree*-based photon mapping, resulting in up to 4× speed-ups and rendering times above 40-50 fps in large urban landscapes. As its main advantages, we can mention its simplicity and ease of implementation (our experimental implementation consists of a few hundred lines of C++/GLSL/OptiX code), and that the results are almost identical to photon mapping. It is not completely identical to progressive photon mapping, mainly because the grid based approach here is always biased because of the finite grid resolution and the approximations introduced so far. Other improvements we have used, like the first hit OpenGL acceleration, the usage of levels of detail and the adaptive buffers are valuable in any photon mapping context, but have proven specially useful in the one we are dealing here: urban landscapes.

Our technique is not free from limitations though. An immediate limitation of the approach is that it may lead to artifacts in situations where the geometry cannot reasonably be parameterized onto cylinders (e.g. when there are building sides that have a radial direction with respect to the center of projection). In this context, it would be interesting to explore alternatives, such as using the parameterization of the geometry, which is used for conventional texturing, as the parameterization for photon splatting. This would be more accurate, but probably would incur a significant performance penalty, or lead to other problems, for example related to seams and different sampling densities in the photon maps. Of course it would also be possible to define other parameterizations that lead to less distortion than the cylindrical one, avoiding some of the problems already mentioned. This is left as an area for further research. Also, it is important to mention that all blocks that share the same adaptive-resolution photon buffer still have the same texture resolution, which might cause non-uniform texel sizes if the blocks vary too much in size. However, in all our experiments we have not seen any noticeable artifact related to this.

Another limitation is that it does not support area light sources, although it would be easy to accommodate in a photon mapping setting. In our case, this includes skylight, which is essential in a cloudy situation. Also, photons can only be deposited on surfaces, and hence we do not support participative media. Another issue we observed is that the orthophotos we were provided already had shadows from the buildings on them, which resulted sometimes in weird "double shadows" effects (Figure 13). Clearly, a method for automatic shadow detection and removal is needed.

Finally, in this paper we have focused on single-light, daylight illumination, but it is straightforward to extend it to multiple light sources like for night lighting by simple redesigning the shooting strategy. However, this would probably require many more photons with smaller maps, thus considerably reducing performance. Probably, a specifically tailored algorithm for night-lighting should be researched.

Acknowledgements

We thank the anonymous reviewers for their constructive comments. This work was partially funded with grants TIN2010-20590-C02-01 and TIN2010-20590-C02-02 from the *Ministerio de Ciencia e Innovación*.

References

1. Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on GPUs. In *Proc. of the ACM Conference on High Performance Graphics 2009*, HPG'09, pages 145–149, 2009.
2. C. Andujar, P. Brunet, A. Chica, and I. Navazo. Visualization of large-scale urban models through multi-level relief impostors. *Computer Graphics Forum*, 29(8):2456–2468, 2010.
3. James Arvo. Backward ray tracing. In *In ACM SIGGRAPH'86 Course Notes - Developments in Ray Tracing*, pages 259–263, 1986.
4. R. Chang, T. Butkiewicz, C. Ziemkiewicz, Z. Wartell, W. Ribarsky, and N. Pollard. Legible simplification of textured urban models. *IEEE Computer Graphics and Applications*, 28(3):27–36, 2008.
5. P. Cignoni, M. Di Benedetto, F. Ganovelli, E. Gobbetti, F. Marton, and R. Scopigno. Ray-casted blockmaps for

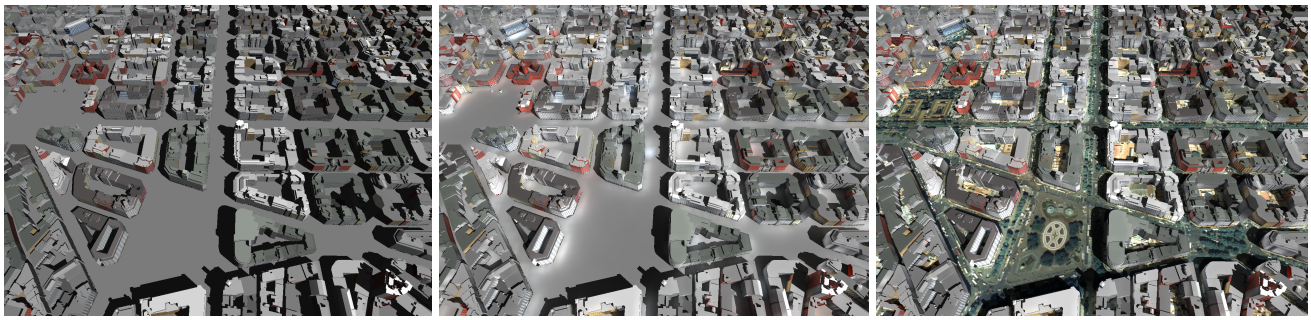

Fig. 13 Urban scene rendered with no indirect lighting (left) and with our approach (middle and right).

large urban models visualization. *Computer Graphics Forum*, 26(3):405–413, 2007.
6. Carsten Dachsbacher and Marc Stamminger. Splatting indirect illumination. In *Proc. of the ACM symposium on Interactive 3D graphics and games*, I3D'06, pages 93–100, 2006.
7. Marco Di Benedetto, Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, and Roberto Scopigno. Interactive remote exploration of massive cityscapes. In *Proc. of the International Symposium on Virtual Reality, Archaeology and Cultural Heritage*, 2009.
8. Jürgen Döllner and Henrik Buchholz. Continuous level-of-detail modeling of buildings in 3d city models. In *Proc. of the ACM international workshop on geographic information systems*, GIS'05, pages 173–181, 2005.
9. Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Trans. Graph.*, 27:130:1–130:8, December 2008.
10. Jan-Henrik Haunert and Alexander Wolff. Optimal and topologically safe simplification of building footprints. In *Proc. of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS'10, pages 192–201, 2010.
11. Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *SIGGRAPH Comput. Graph.*, 24(4):145–154, September 1990.
12. Robert Herzog, Vlastimil Havran, Shinichi Kinuwaki, Karol Myszkowski, and Hans-Peter Seidel. Global illumination using photon ray splatting. *Computer Graphics Forum*, 26(3):503–513, 2007.
13. Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.
14. Fabien Lavignotte and Mathias Paulin. Scalable photon splatting for global illumination. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE'03, pages 203–ff, New York, NY, USA, 2003. ACM.
15. Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Proc. of the 1995 symposium on Interactive 3D graphics*, I3D'95, pages 95–ff., 1995.
16. Morgan McGuire and David Luebke. Hardware-accelerated global illumination by image space photon mapping. In *Proc. of the Conference on High Performance Graphics 2009*, HPG'09, pages 77–89, New York, NY, USA, 2009. ACM.
17. S.G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, et al. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics*, 29(4):1–13, 2010.
18. Oriol Pueyo and Gustavo Patow. Structuring urban data. Technical Report IMA12-01-RR, Departament IMA, Universitat de Girona, 2012.
19. Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. *Comput. Graph. Forum*, 31(1):160–188, 2012.
20. Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *Proc. of the 2009 symposium on Interactive 3D graphics and games*, I3D'09, pages 75–82, 2009.
21. F. X. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum*, 16(3):207–218, 1997.
22. Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph.*, 21:527–536, July 2002.
23. John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tessellations. *SIGGRAPH Comput. Graph.*, 21:119–128, August 1987.
24. Laszlo Szirmay-Kalos, Laszlo Szécsi, and Mateu Sbert. *GPU-based Techniques for Global Illumination Effects (Synthesis Lectures on Computer Graphics)*. Morgan and Claypool Publishers, 2008.
25. C. A. Vanegas, D. G. Aliaga, P. Wonka, P. MÃijller, P. Waddell, and B. Watson. Modelling the appearance and behaviour of urban spaces. *Computer Graphics Forum*, 29(1):25–42, 2010.
26. Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, pages 153–164, 2001.
27. M. Wimmer, P. Wonka, and F. Sillion. Point-based impostors for real-time visualization. In *Proc. of the Eurographics Workshop on Rendering 2001: London, United Kingdom, June 25-27, 2001*, pages 163–176. Springer Verlag Wien, 2001.
28. Ling Yang, Liqiang Zhang, Jingtao Ma, Jinghan Xie, and Liu Liu. Interactive visualization of multi-resolution urban building models considering spatial cognition. *International Journal of Geographical Information Science*, 25:5–24, February 2011.