Eurographics Italian Chapter Conference (2008)
V. Scarano, R. De Chiara, and U. Erra (Editors)

Indirect illuminance computation on GPU for lighting CAD using CIE Basic Method

S. Mazzocchi[1] F. Musante[2] M. Rossi[2] D. Selmo[1]

[1] D.I.Co. - Università degli Studi di Milano
[2] Dip. In.D.A.Co - Politecnico di Milano

Abstract

In this work, we focus our attention to near real time illuminance for Lighting CAD. The implemented model is the Basic Method CIE which is a simplified method used in interior lighting design; a pre-computational phase is required, for form factor evaluation, and it is done by the CPU while GPU is used for the direct component evaluation to render the pseudo color map of the computed result. Our approach offers a further improvement in comparison with traditional lighting CAD, since it allows lighting designer to obtain an interactive solutions and the interactivity is reached using graphic hardware, in fact the pre-computation process is independent from the position of luminaries into the scene. The presented method is applied on a simple geometry, a parallepided room, where the surface is assumed to show a Lambertian's behaviour, but the model can be improved to remove these limitation.

Categories and Subject Descriptors (according to ACM CCS):
I.3.7 [Computer Graphics]: Colour, shading, shadowing, and texture
J.6 [Computer Applications]: Computer-aided design (CAD)

1. Introduction

Predicting the radiative flux transfer from sources to receiving surfaces is fundamental to all lighting CAD software; lighting calculation is done with mathematical model of complex physical phenomenon which occurs within a lighted space and since these model are never accurate in every detail, all the computation can be considered an approximation of real situation. The accuracy of calculated values is defined as the degree which exist between predicated values and reality. The goal of lighting CAD is to make available a tool to choose between design alternatives or to refine a particular design and these calculations are performed during the design stage to achieve important information about the lighting system performance. For lighting CAD computation in interiors, we are not interested in producing fine rendering but in providing the lighting designer with a tool for positioning and orienting luminaries through a technical real time representation of illuminance of surfaces. The main purpose of our method is to light a real scene by the use of a real light source described by an intensity table which is derived from a IESNA file formats [LM-02], [LM-05] . In the past we proposed a method for real time representation of direct illuminance level on the surface of a parallelepiped environment through the use of GPU calculation capabilities [SMR06]. Now, the proposed algorithm takes in account the direct contribution of the luminaries placed in the scene and the indirect contribution of the room surface which is considered as a Lambert's perfect diffuser.

2. Previous works

The first attempt to use the hardware acceleration to compute global illumination is dates long time before programmable graphic hardware should be wide available on the market. Keller [Kel97] uses OpenGL to accelerate the radiosity computation, he creates a quasi Monte-Carlo particle simulation for light transport on CPU and he uses the OpenGL point light sources, placed at particles position, setting the power of them in accordance with the particle's power along the random walk path. The contribution of the particles are integrated by using an accumulation buffer. Some of the previous work use a precomputed form factor to compute and display radiosity solution in real time. Coombe et al. [CHL03]

use progressive refinement radiosity for computing a global illumination solution of diffuse environment; this technique don't use explicit storage of the radiance's matrix and the model can be displayed as the solution progress. The radiosity and residual energy are stored as a texture connected to geometry model of the scene; to mesh the scene, instead of subdividing the geometry as in classical radiosity approach the authors use a texel, the scene is rendered from the shooter's point of view and the result is stored in a item buffer, for each receiving polygons and for each receiving textel, if the textel is visible in the item buffer, the form factor is computed and the energy and residual are computed and stored as a texture.

Carr at al. [CHH03] use a floating-point texture format to hold the radiosity matrix and given a precomputed form factor the algorithm is able to compute and display the radiosity solution entirely on the GPU. While the geometry model of the scene is fixed, the emittance of surface's model is not, and the GPU algorithm is able to support modification either of patch reflectance and dynamic relighting. The algorithm makes use of Jacobi iteration formula which can take better advantage from the GPU architecture, and it creates a texture atlas to store indices from form factor array and it uses these to create the surface texture which contains the calculated illumination values.
Another type of algorithm makes use of cube maps to capture the light emitted by the light source or bounced by other surfaces of the synthetic scene. Cube maps are traditionally used for hardware simulation of reflection of environment on shiny surface and the novel idea is to use them for global illumination. Nijasure et al. [NPG05] create an algorithms which simulates the transport of light in a digital environment by following the light emitted from the light sources through multiple bounces on the scene's surfaces. They divide the volume of the 3D scene into a uniform volumetric grid and then the algorithm creates 3D maps for each subdivision which values represent the incoming light field at the grid point. A cube maps (cube maps are the projections of the environment on the six faces of cube with camera position placed at the centre of each faces) is rendered at each grid point of the volumetric grid and in this way it is possible to capture the incoming radiance field due to the light reflected off by the surface's scene.
Other approach to achieve interactive global illumination is to change the rendering equation in accordance with the GPU capabilities; anyway a precomputation stage on CPU is required. Sunshine et al. [SF06] propose a precomputation-based approach for real time rendering of scene which includes complex lighting phenomena such as radiosity, surface scattering and allows interactive modification of the camera and lighting parameters. The main idea of the method is a new parameterization of the rendering equation based on a set of offset transfer maps. An extension of Instant Radiosity is proposed by Segovia et al. [SI06]: the suggested method is based on building several estimators and efficiently combine them to find a set of virtual point light sources which is relevant for the scene seen by the camera. In this way the presented algorithm is much faster than classical solution to global illumination rendering. The aim of the algorithm is to find a virtual set of point light source to describe the radiant field and use these source to render the areas seen by the camera position.

2.1. L.I.D. representation and interpolation

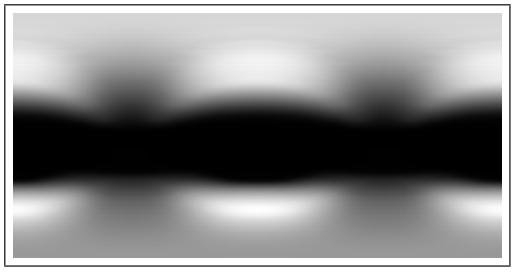

Figure 1: *LID texture encoding example.*

In order to make available intensity values, encoded in Light Intensities Distributions (L.I.D.), during pixel shader execution, we need to store this information in a data texture file. In illuminating engineering, L.I.D is normally measured and represented in a parametric form (the candlepower distribution $I(C,\gamma)$ of a lamp or luminarie is the variation of luminous intensity around the C plane and around the vertical angle γ) which can be used to build the texture file. So we can easily create a map M from intensity values $I(C,\gamma)$ to texels $T(s,t)$ of a gray-level texture as shown in figure 1.

$$M : I(C,\gamma) \mapsto T(s,t) \qquad (1)$$
$$C \in [0,2\pi] \quad \gamma \in [0,\pi] \quad (s,t) \in [0,1]$$

During shader execution, we need to get an intensity value, from the L.I.D., in order to compute illuminance for each point of the scene geometry. Given a point P, we can calculate the relative normalized direction D (from P to L.I.D. center) with the respect of the L.I.D. reference frame. Texture lookup can be easily implemented using the inverse mapping M^{-1} applied to angles derived from direction D. In particular u,v coordinates for texture lookup can be calculated as:

$$u = \begin{cases} \arctan(D_z/D_x) & \text{if } \arctan(D_z/D_x) > 0 \\ \arctan(D_z/D_x) + 2\pi & \text{if } \arctan(D_z/D_x) < 0 \end{cases} \qquad (2)$$

$$v = \arccos(clamp(D_y,-1,1))/\pi \qquad (3)$$

3. Direct illuminance calculation

If the source is punctual (the light is originating from a point but not with an isotropic angular distribution), the direct illu-

minance E_v on the surface can be calculated as a function of light intensity I_v, falling on surface with an angle θ to surface normal and with a distance r from the light source.

$$E_v = \frac{I_{v1000lm}(C,\gamma)\Phi}{r^2}\cos(\theta) \tag{4}$$

where $I_{v1000lm}$ is the luminous intensity in cd/1000lm.

4. Global Illumination calculation

Since we are not primarily interested in obtaining aestically pleasing images,but in computing a technical representation for lighting CAD in interior, our algorithm is derived from the CIE *Basic Method* [40-78] . The method is called *Basic* because it is a basis for a lot of applied method [52-82]. Our implementation can be applied if the following hypotheses are satisfied:

- The interior is a rectangular parallelepiped
- The room's surface reflects the light uniformly in accordance to the Lambert's law; the reflection factor considered during the calculation's stage are:
  - r1: reflectance of ceiling
  - r2: reflectance of frieze (this is the part of vertical wall lies between the ceiling and the plane of luminaries)
  - r3: reflectance of the wall (defined as the vertical surfaces between the working plain and the plane of the luminaries)
  - r4: reflectance of working plane
- The direct flux on working plane doesn't vary significantly if the luminaries are rotated along the their vertical axis; this is the case of most fluorescent luminaries used in interior lighting

For interflection calculation we consider only four surfaces and in details these surfaces are:

- the ceiling
- the frieze
- the walls
- the working plane

The total flux received by a surface j is equal to the sum of the F_j direct radiate to the surface and the flux IF_j derived from interreflections:

$$\Phi_j = F_j + IF_j \qquad (j = 1,2,3,4) \tag{5}$$

where Φ_j is the direct flux received from the j surface of the room and IF_j is the interflection component on j surface of the room.

The flux due to interflections is given by the expression :

$$IF_j = \sum_{i=1}^{4} \frac{g_{i,j}R_i}{A_i}\Phi_i \qquad (j = 1,2,3,4) \tag{6}$$

where $g_{i,j}$ is the exchange coefficient between the surface i and j, R_i the reflectance of surface and A_i the area of surface i

After substitution of equation (6) in (5), the resulting expression, can be written as:

$$\Phi = F + gRA^{-1}\Phi \tag{7}$$

where Φ is the column matrix with coefficients F_j $(j = 1,2,3,4)$, F is the column matrix with coefficients F_j $(j = 1,2,3,4)$, G is the matrix $(4,4)$ with coefficients $g_{i,j}$, R is the 4-th order diagonal matrix with r_i $(i = 1,2,3,4)$ and A is the 4-th order diagonal matrix storing surfaces area $(i = 1,2,3,4)$.

Equation (7) can be rewritten as:

$$F = [I - gRA^{-1}]\Phi \tag{8}$$

After same equation manipulation, the resulting expression can be written as:

$$[I - gRA^{-1}]^{-1}F = \Phi \tag{9}$$

Now, we can solve the last equation and replace the value of F_j in the equation (5) and solve respect the IF_j term to obtain the indirect illuminance mean value on surface j:

$$IF_j = \Phi_j - F_j \tag{10}$$

The illuminance value of indirect component can be computed as the ratio between the luminous flux and the surface which is receiving the flux:

$$IE_j = IF_j/A_j \tag{11}$$

where IE_j is an offset value which to be added to the direct component at each pixel of the room surface j.

5. Implementation details

The CIE basic method has been implemented into commercial lighting software as simplified method for illuminance calculation. Our approach offers a further improuvements since it allows lighting designer to obtain an *interactive* solutions. Interactivity has been reached using graphics hardware capabilities. Indeed our implementation is strongly based on GPU computation. In particular, the application divides the illuminance calculation in three steps.

5.1. Pre-computation phase

In this stage the exchange coefficient are calculated between the different surface's room as we have described in the previous paragraph *Indirect contribution calculation*. All the computation in this step is done in CPU.

5.2. Shader Activation and execution

The direct lighting is calculated in GPU using shaders written in the OpengGL Shading Language [KBR04]. The scene is rendered using texture-enconded photometric webs evaluating special purpose vertex and pixel shaders. The first shaders used are *direct.vert* and *direct.frag* to perform the calculation of the direct illumination's component of the scene; in detail the vertex program task is to manage the surface geometry and to calculate the normal of the scene object (this information is used to perform the direct illuminance calculation, as described in [SMR06]). Direct illuminance calculation pseudocode is presented below, each term of equation (4) is calculated and the result is displayed by a luminance gradient color map [RA05]. Since the LID has been coded into a texture, intensity values are retrievied using texture lookup function. Texture coordinates for lookup are evaluated converting directional information into angular information.

```
// cos(q) calculation
NdotL = max(dot( normalVector,
        normalize(toLightDirection)), 0.0 )

// texture lookup coordinates based on
// light angle u = sphericalTheta(lightVector);
v = sphericalPhi(lightVector);

// read light intensity from texture and
// takes only the first component
vec4 = texture2D(LIDtexture ,vec2(u,v));
float intesity = textcolor.x;

// Illuminance Calculation
float illuminance = NdotL*flux*intensity /
(distance* distance);

// normalize the illuminance value in [0,1]
float normIlluminance =
      (clamp( illuminance/maxIlluminance,
       0.0, 1.0));

// calculate the color for the normalized
// illuminance value
vec4 mappedColor = texture2D(colorMapTexture,
 vec2(normIlluminance, 0.5));

// write final color to fragment
gl_FragColor = mappedColor;
```

The algorithm creates six textures, one for each surface, where it stores the illuminance distribution created by the direct illumination component. Illuminance values are stored using floating point textures in order to maintain numerical accuracy. In detail, for each luminarie in the scene, the mean illuminance value on the room's surface is calculated by the use of orthographic projection of room's surfaces, in such a way there isn't a perspective distortion of the image due to different distance of observed object from the camera position. The image is stored in a texture (render-to-texture process) where only red channel is used; the buffer is read and all data are used to calculate the mean illuminance value. The described process is repeated until all the luminaries in the scene have been considered by the calculation process. On the basis of direct illuminance values calculated for each surfaces, indirect component is found as described in the previous section; in this way an offset value, one for each surfaces, which represents the mean indirect contribution of room's surface to illuminance, is added to each pixel of the corresponding surface.

ceiling	80%
frieze	55 %
walls	55 %
working plane	40 %

Table 1: *Reflection factors (in percentage) for each surface in the example room.*

5.3. Final Texture creation

For each surface of the environment, a texture is created by the use of calculated values at previous stage and then the texture is applied to each surface. The final pseudo color representation is obtained by the use of the 1D color map texture which is linear interpolated in accordance with the calculated illuminance value.

6. Results

Our method has been implemented on a Windows workstation using a single core Intel Centrino 1.7 Ghz processor, 1 Gb memory and ATI 9600 graphic card. For testing purposes, a simple rectangular room has been used and luminaries has been placed in different positions in order to obtain heterogeneous illuminance levels on room surfaces. The installation height is the same for all considered luminaries and the reflection factor for each surface is presented in the table 1. The considered luminarie shows a double distribution, a part of luminous flux is direct toward floor and a part is addressed toward the ceil of the room; in this way we can test a real common situation in interior lighting design. The figures from 2 to 4 represent the illuminance value, which has been calculated by the algorithm, for two different configuration of the luminarie; position and orientation of each luminarie can be interactively modified by the user in accordance with the wished illuminance value on each surface.

N. Lum	Acq	Sum	Rendering	Calc	Total
1	0.165	0.451	0.012	0.338	0.966
2	0.33	0.464	0.015	0.378	1.187
3	0.508	0.463	0.015	0.37	1.356
4	0.644	0.498	0.012	0.391	1.545
5	0.847	0.492	0.015	0.404	1.758
6	1.028	0.488	0.015	0.394	1.925
7	1.137	0.557	0.017	0.436	2.147
8	1.311	0.561	0.014	0.454	2.34

Table 2: *Execution times (all times in seconds). Columns show: number of luminaires (N. Lum), textures acquisition time (Acq), textures accumulation time (Sum), rendering time, CPU calculations time (Calc) and total time.*

The table 2 shows execution times for the described examples and it presents the relationship between the execution times and the number of luminaries placed in the scene.

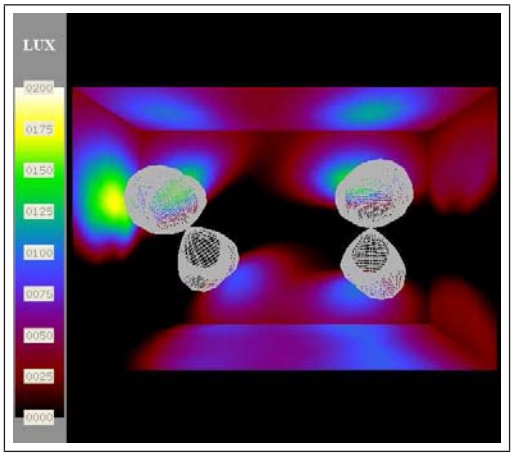

Figure 2: *Example 1: Direct illuminance calculation. Room measures: 6 x 2.5 meters, height 4 meters.)*

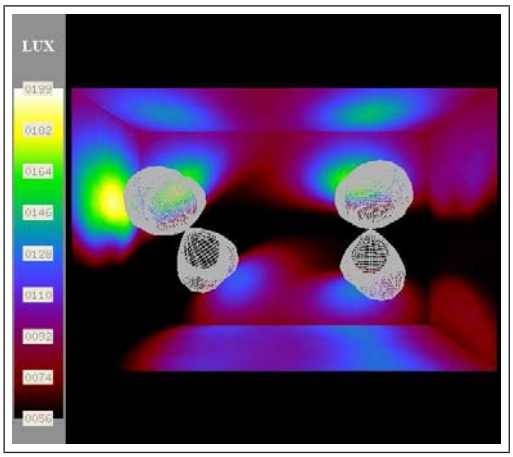

Figure 3: *Example 1: Illuminance values representation with indirect illuminance contribution. Room measures: 6 x 2.5 meters, height 4 meters.*

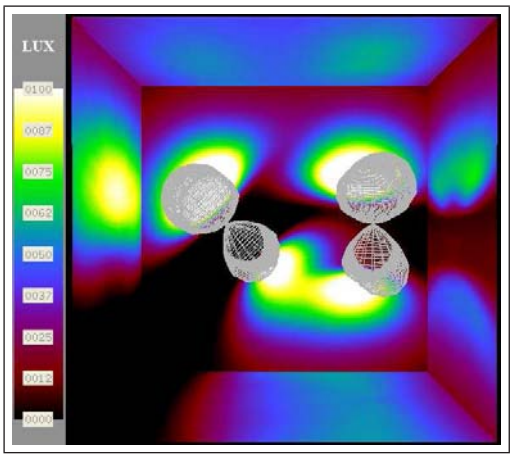

Figure 4: *Example 2: Direct illuminance calculation. Room measures: 5.8 x 2.5 meters, height 5.8 meters.*

7. Conclusion and future works

The main aim of this research is not focused on quality rendering but in real-time interactive tools providing the designer with function for the positioning and orientation of luminaries. Technical results of lighting computation are displayed on the fly as color map representing the illuminance values on surfaces, and these results strictly conform to lighting engineering standards. The future algorithm's improvement will concern two different aspects. The first one is connected with the calculation of indirect component distribution and not only the mean value as in current implementation which is treated as an offset value to add to direct component distribution in each pixel. The second one is the environment's shape which could be different from a rectangular room and the geometry should be defined by user; to reach this objective the Combe [CHL03] approach seems to be a feasible solution, but it must be extended for considering real luminaries with LID. The proposed method could also be easily applied to exterior lighting calculation, for example in road or tunnel lighting: in these particular applications even the illumination calculation over the interest's surface can also consider the luminance evaluation problem through the material response that should be introduced in the model calculation.

References

[40-78] 40-1978 C.: Calculations for interior lighting: Basic method. *CIE Pubblications* (1978).

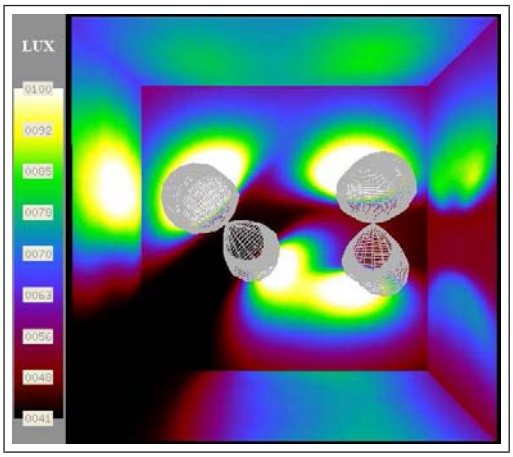

Figure 5: *Example 2: Illuminance values representation with indirect illuminance contribution. Room measures: 5.8 x 2.5 meters, height 5.8 meters.*

[52-82] 52-1982 C.: Calculations for interior lighting: Applied method. *CIE Pubblications* (1982).

[CHH03] CARR N., HALL J., HART J.: Gpu algorithms for radiosity and subsurface scattering, 2003.

[CHL03] COOMBE G., HARRIS M., LASTRA A.: Radiosity on graphic hardware. *Graphics Hardware* (2003).

[KBR04] KESSENICH J., BALDWIN D., R.ROST: The opengl shading language. *Language Version 1.10* (April 2004).

[Kel97] KELLER: An instant radiosity. *In Proc. SIGGRAPH 97* (1997), 49–56.

[LM-02] LM-63-02 A.: Ansi approved standard file format for electronic transfer of photometric data and related informationn. *IESNA Pubblications* (2002).

[LM-05] LM-74-05 I.: Iesna standard file format for the electronic transfer of luminarie component data. *IESNA Pubblications* (2005).

[NPG05] NIJASURE M., PATTANAIK S. N., GOEL V.: Real-time global illumination on gpu. *Journal of Graphics Tools 10(2)* (2005), 55–71.

[RA05] ROSSI M., ALGERI T.: Real-time 3D color scale representation of computed illuminance levels. *SIOF-Color Conference* (October 2005), 1–8.

[SF06] SUNSHINE B., FACULTOS P.: Photorealistic lighting with offset radiance transfer mapping. *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006), 15–21.

[SI06] SEGOVIA, IEHL J.: Bidirectional instant radiosity. *Eurographics Symposium on Rendering* (2006).

[SMR06] SELMO D., MUSANTE F., ROSSI M.: Gpu based direct illuminance values computation for interactive lighting cad. In *Eurographics Italian Chapter Conference* (2006), pp. 219–224.