Fast Filtering and Tone Mapping using Importance sampling

Balázs Tóth and László Szirmay-Kalos
tbalazs@sch.bme.hu - szirmay@iit.bme.hu
Dept. of Control Engineering and Information Technology
Budapest University of Technology and Economics
Magyar Tudósok krt. 2., H-1117, Hungary

ABSTRACT

This paper examines the process of tone mapping and blooming, and then discusses its real-time implementation on current graphics hardware (GPU). The main contribution of this paper is a fast Gaussian filtering algorithm that significantly reduces the number of texture fetches and thus runs at interactive frame rates on the GPU. The reduction of the number of texture fetches is made possible by the rewriting of the filtering integral according to the concept of importance sampling. The proposed method can be used not only in tone mapping but also in other screen space camera effects as well, like bloom, glow or depth of field.

Keywords: Filtering, GPU programming, Importance sampling.

1 INTRODUCTION

Full screen framebuffer effects are integral parts of the rendering process if we want to create high quality images. It is important to make these effects as fast as possible, because they are applied to the whole framebuffer. These effects are computed by rendering a full screen quadrilateral to get the pixel shader to process every pixel.

Full screen frame buffer effects include many important particular methods, such as tone mapping, glow generation, or depth of field. These methods require weighted averages of pixel values at different neighborhoods, which can be obtained by convolving the image with the particular filter kernel. Thus the efficient implementation of filtering is a key to the fast realization of such effects. In this paper we consider the efficient realization of the 2D Gaussian filter. However, we should emphasize that the basic ideas can be used for other filters as well. Having proposed a fast filtering scheme, we present different applications, including bloom and temporal tone mapping.

2 FILTERING METHOD

Gaussian filtering is the most time critical part of the GPU implementation of many screen space methods. If we do it naively, the fragment shader needs to access the texture memory many times to fetch values in the neighborhood.

The general form of the Gaussian filter is

$$L'(X,Y) = \int_{y=-\infty}^{\infty}\int_{x=-\infty}^{\infty} L(X-x,Y-y)\frac{1}{2\pi\sigma^2}e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)}dxdy.$$

Taking advantage of the fact that the exponential function diminishes quickly, the infinite integrals can be approximated by finite integrals:

$$L'(X,Y) \approx \int_{y=-S}^{S}\int_{x=-S}^{S} L(X-x,Y-y)\frac{1}{2\pi\sigma^2}e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)}dxdy.$$

This double integral is replaced by double summations for discrete images. If the domain of $[-S,S]\times[-S,S]$ contains $N\times N$ discrete samples, then the discrete integral approximation requires the evaluation of N^2 kernel values, multiplications, and additions, which is rather costly when repeated for every pixel of the screen.

2.1 Separation of dimensions

One common way of reducing the computation burden of 2D filtering is to exploit the separability of the filter kernel. It is based on the recognition that the two dimensional convolution can be replaced by a vertical and a horizontal one-dimensional convolutions. The double integral is computed in two passes. The first pass results in the following 2D function:

$$F(X,Y) = \int_{-S}^{S} L(X-x,Y)\frac{1}{\sqrt{2\pi}\sigma}e^{(-\frac{x^2}{2\sigma^2})}dx.$$

Then the final result can be obtained by filtering F again with a similar one dimensional filter:

$$L'(X,Y) \approx \int_{-S}^{S} F(X,Y-y)\frac{1}{\sqrt{2\pi}\sigma}e^{(-\frac{y^2}{2\sigma^2})}dy.$$

In this way the computation complexity can be reduced from N^2 evaluations to $2N$ evaluations.

2.2 Importance sampling

In this paper, we propose another trick to further reduce the required computations. This approach is based on the concept of *importance sampling*. Let us consider the $\int_T Y(x-t) \cdot g(t)\, dt$ convolution where Y is the image value (e.g. luminance) and g is the filter kernel, i.e. the Gaussian in our case, and find a function $\tau(t)$ together with its inverse $t(\tau)$ so that the following conditions hold

$$\frac{d\tau}{dt} = g(t) \;\rightarrow\; \tau(t) = \int^{t} g(t')dt'.$$

If g is known, then τ can be computed and inverted off line. Substituting the precomputed $t(\tau)$ function into the integral we obtain

$$\int_T Y(x-t) \cdot g(t)\, dt = \int_{\tau(T)} Y(x - t(\tau))\, d\tau$$

Approximating the transformed integral taking uniformly distributed samples corresponds to a quadrature of the original integral taking N' non-uniform samples:

$$\int_{\tau(T)} Y(x-t(\tau))\, d\tau \approx \frac{|\tau(T)|}{N'} \cdot \sum_{i=1}^{N'} Y(x - t(\tau_i))$$

where $|\tau(T)|$ is the size of the integration domain.

This way we take samples densely where the filter kernel is large and fetch samples less often farther away, but do not apply weighting. Note that this allows us to use a smaller number of samples ($N' < N$) and not to access every pixel in the neighborhood since far from the center of the filter kernel, the weighting would eliminate the contribution anyway, so taking dense samples far from the center would be a waste of time.

The implementation of this approach is quite straightforward. The required $t(\tau)$ function is computed by integrating the standard Gaussian function and inverting the integral. The samples of the resulting $t(\tau)$ are hardwired into the shader $(-3.8697, -1.7229, 0, 1.7229, 3.8697)$. These constants determine where the texture should be fetched.

The convolution is executed separately for the two directions (this is possible because of the separability of the Gaussian filter). The horizontal and the vertical passes are implemented by the following fragment shaders:

```
texture LumTex;
half4 DownScaleH(float2 tex0:TEXCOORD0):COLOR
{
   float2 du1 = float2(1.7229/Width, 0));
   float2 du2 = float2(3.8697/Width, 0));
   half4 texLookUp;
   texLookUp = tex2D(LumTex, tex0 - du2).r +
               tex2D(LumTex, tex0 - du1).r +
               tex2D(LumTex, tex0).r +
               tex2D(LumTex, tex0 + du1).r +
               tex2D(LumTex, tex0 + du2).r;
   return half4(texLookUp / 5, 0, 0, 1);
}

half4 DownScaleV(float2 tex0:TEXCOORD0):COLOR
{
   float2 dv1 = float2(0, 1.7229/Height));
   float2 dv2 = float2(0, 3.8697/Height));
   half4 texLookUp;
   texLookUp = tex2D(LumTex, tex0 - dv2).r +
               tex2D(LumTex, tex0 - dv1).r +
               tex2D(LumTex, tex0).r +
               tex2D(LumTex, tex0 + dv1).r +
               tex2D(LumTex, tex0 + dv2).r;
   return half4(texLookUp / 5, 0, 0, 1);
}
```

In the following sections we apply this filtering scheme for bloom and tone mapping effects.

3 BLOOM EFFECT

Due to the scattering of light in the optical system of the eye, sources of relatively strong light cause the decrease of contrast in their vicinity. This phenomenon is called the *bloom*. Such an effect cannot be naturally evoked while perceiving an image on a display due to different viewing conditions and limited maximum luminance of such devices. Thus we should account for it during rendering.

The attenuation due to blooming at frequency ρ of the visible spectrum under a given pupil aperture d is modeled by an *Ocular Transfer Function* (*OTF*) [1]:

$$OTF(\rho, d) = e^{-\frac{\rho}{20.9-2.1\cdot d}1.3-0.07\cdot d}$$

where

$$d(\tilde{Y}) = 4.9 - 3\tanh(0.4\log_{10}\tilde{Y} + 1),$$

and $\tilde{Y}$ is the logarithmic average of the luminance in the scene. If we want to simulate this effect in a physically correct way, then we have implement the computation of the *OTF*. However in most of the computer graphics applications, like computer games, the spectacular but fast result is more important than the physical correctness. Thus we should rather use fast approximations.

An approximative approach differentiates strongly and weakly radiating parts of the image. Then the strongly radiating parts are blurred and are added to the darker regions.

The implementation of this method requires blurring, which is a Gaussian filtering computed by the method

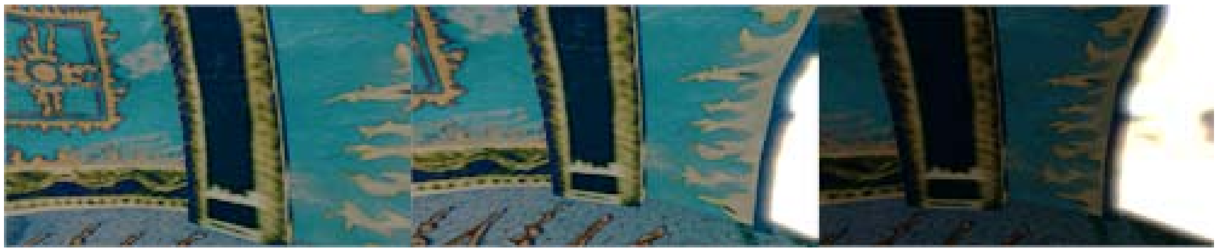

Figure 1: Bloom effect

of the previous section. First the scene is rendered into a floating point buffer. In this buffer the pixels with high intensities represents the glowing parts of the scene. We use a low pass filter to separate these parts to another image buffer. The second step is to blur this image using convolution with Gaussian kernel. After the blurring pass the original image is combined with the generated glow image pixel by pixel. To get good results, we need to use high dynamic range source image, otherwise we may catch too much or too less "glowing emitter" part of the image.

We may also add an interesting motion blur like trailing effect to the glow easily. If we store the blurred glow image we can modulate the next frame's glow image with it. The length of the trail can be controlled during the composition with a dimming parameter.

4 TONE MAPPING

Off the shelf monitors can control the intensity just in a limited, *low dynamic range* (*LDR*). Therefore the values written into the frame buffer are unsigned bytes in the range of [0, 255], representing values in [0,1], where 1 corresponds to the maximum intensity of the monitor. However, global illumination computations result in *high dynamic range* (*HDR*) luminance values that are not restricted to the range of the monitors. The conversion of HDR image values to displayable LDR values is called *tone mapping* [5]. The conversion is based on the luminance the human eye is adapted to. Assuming that our view spans over the image, the adaptation luminance will be the average luminance of the whole image.

Having the adaptation luminance, source luminance values Y are first mapped to relative luminance Y_r:

$$Y_r = \frac{\alpha \cdot Y}{\tilde{Y}},$$

where α is a constant of the mapping, which is called the *key value*.

The relative luminance values are then mapped to the displayable [0,1] pixel intensities L using the following function:

$$L = \frac{Y_r}{1+Y_r}. \tag{1}$$

This formula maps all luminance values to the $[0,1]$ range in such way that relative luminance $Y_r = 1$ is mapped to pixel intensity $L = 0.5$. This property is used to map a desired luminance level of the scene to the middle intensity on the display. Mapping a higher luminance level to middle gray results in a subjectively dark image whereas mapping lower luminance to middle gray will give a bright result. Images which we perceive at low light condition are relatively dark compared to what we see during a day. We can simulate this impression by modulating the key value with respect to the adapting luminance in the screen.

Key value α controls whether the tone mapped image appears relatively bright or relatively dark. Its exact value can be left as user choice, or it can be estimated automatically based on the relations between minimum, maximum, and average luminance in the scene [4]. Unfortunately, the critical changes in the absolute luminance values may not always affect the relation between these three values. For example, this may lead to dark night scenes appearing as too bright.

Krawczyk [3] proposed an empirical method to calculate the key value. His method is based on the absolute luminance. Since the key value was introduced in photography, there is no scientifically based experimental data which would provide an appropriate relation between the key value and the luminance. The low key is 0.05, the typical choice for moderate illumination is 0.18, and 0.8 is the high key. Krawczyk empirically specified key values for several illumination conditions and interpolated the rest using the following formula:

$$\alpha(\tilde{Y}) = 1.03 - \frac{2}{2+\log_{10}(\tilde{Y}+1)}.$$

This basic tone mapping process can be extended in several ways. In the following subsections, we discuss a local version, the incorporation of the bloom effect into the tone mapping process, the extension to temporally varying image sequence, and to cope with scotopic vision.

4.1 Local tone mapping

The tone mapping function of equation 1 may lead to the loss of details in the scene due to extensive contrast compression. Reinhard et al. [4] proposed a solution to preserve local details by employing a spatially variant local adaptation value V in equation 1:

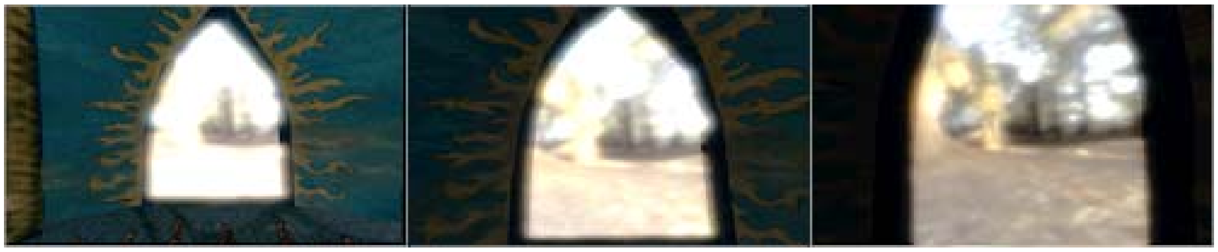
Figure 2: The adaptation process

$$L(x,y) = \frac{Y_r(x,y)}{1+V(x,y)},$$

where x, y are the pixel coordinates.

The local adaptation V equals to the average luminance in a neighborhood of the pixel. The problem lies however in the estimation of how large the surround of the pixel should be. The goal is to have as wide surround as possible, however too large area may lead to well known inverse gradient artifacts called *halos*. The solution is to successively increase the size of a surround on each scale of the pyramid, checking each time if no artifacts are introduced.

For this purpose a Gaussian pyramid is constructed with successively increasing kernel size. The Gaussian for the first scale is one pixel wide, setting kernel size to $s = (2\sqrt{2})^{-1}$, on each subsequent scale s is 1.6 times larger.

Having the current and the previous scales, we update the perceptual data on a per pixel basis in a separate rendering pass. The local adaptation is computed using the measure of the difference between the previous and the current scale as described in [4].

4.2 Bloom integration into tone mapping

To take account the additional light scattering during the tone mapping process, we have to create a bloom map based on the absolute luminance of the picture.

For the bloom map, we first estimate the proper scale for the luminance of the current pixel. It depends on the adapting luminance and it is uniform for the whole frame so we supply it as a parameter to the fragment program. Before descending to the next scale of the Gaussian pyramid, the texture containing the current scale becomes the previous scale, and the texture with the current set of the perceptual data becomes the previous set.

After descending to the lowest scale of the Gaussian pyramid, the perceptual data texture is complete. In the final rendering step, we tone map the HDR frame and apply the perceptual effects with the equation

$$L(x,y) = \frac{Y_r + Y_{bloom}}{1+V(x,y)},$$

where L is the final pixel intensity value, Y_r the relative luminance, Y_{bloom} is the amount of additional light scattering in the eye, and V is the local adaptation map.

4.3 Temporal Luminance adaptation

While tone mapping the sequence of HDR frames, it is important to note that the luminance conditions can change drastically from frame to frame. The human vision reacts to such changes through the temporal adaptation processes. The time course of adaptation differs depending on whether we adapt to light or to darkness, and whether we perceive mainly using rods (during night) or cones (during a day).

To take into account the adaptation process, a filtered $\tilde{Y}_a$ value can be used instead of the actual adapting luminance $\tilde{Y}$. The filtered value changes according to the adaptation processes in human vision, eventually reaching the actual value if the adapting luminance is stable for some time. The process of adaptation can be modeled using an exponential decay function:

$$\tilde{Y}_a^{new} = \tilde{Y}_a + (\tilde{Y} - \tilde{Y}_a) \cdot (1 - e^{-\frac{T}{\tau}})$$

where T is the discrete time step between the display of two frames, and τ is the time constant describing the speed of the adaptation process. These time constants are different for rods and cones:

$$\tau_{rods} \approx 0.4s, \quad \tau_{cones} \approx 0.1s$$

Therefore, the speed of the adaptation depends on the level of the illumination in the scene. The time required to reach the fully adapted state depends also on whether the observer is adapting to light or dark conditions. The above numbers describe the adaptation to light. The full adaptation to dark takes up to tens of minutes, so it's not simulated.

4.4 Scotopic vision

On low light conditions only the rods are active, so color discrimination is not possible. The image becomes less colorful. The cones start to loose sensitivity at $3.4\frac{cd}{m^2}$ and become completely insensitive at $0.03\frac{cd}{m^2}$ where the rods are dominant. We can model the sensitivity of rods with the following equation:

$$\sigma(Y) = \frac{0.04}{0.04+Y}$$

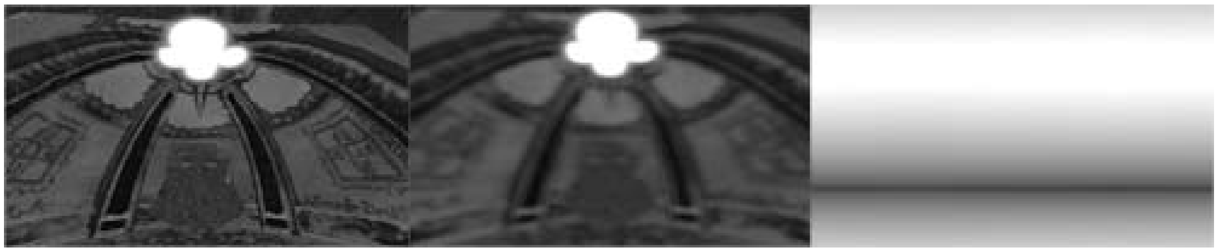
Figure 3: The illumination levels during the filtering: original, local, global

where Y denotes the luminance. The value $\sigma = 1$ describes the monochromatic vision and $\sigma = 0$ the full color discrimination.

4.5 Implementation

The global tone mapping operators require the computation of the global average of the luminance. In the first step we calculate the luminance value of every pixel using the standard CIE XYZ transform (D65 white point):

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7132 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The Y component of the XYZ vector is the luminance, so we can compute it with the following equation:

$$Y = 0.2126 \cdot R + 0.7132 \cdot G + 0.0722 \cdot B.$$

When we have the luminance image we can calculate the global average with Gaussian filtering [2]. This can be a multi step process, as we scale down the luminance image to one pixel in several passes. We can reduce the size of the luminance image in every step to reduce the computation.

We account for the last perceptual effect, the scotopic vision, while applying the final pixel intensity value to the RGB channels in the original HDR frame. Using the following formula, we calculate the tone mapped RGB values as a combination of the color information and the monochromatic intensity proportionally to the scotopic sensitivity:

$$\begin{bmatrix} R_L \\ G_L \\ B_L \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \frac{L \cdot (1 - \sigma(Y))}{Y} + \begin{bmatrix} 1.05 \\ 0.97 \\ 1.27 \end{bmatrix} L\sigma(Y),$$

where $[R_L, G_L, B_L]$ denotes the tone mapped intensities, $[R, G, B]$ are the original HDR values, Y is the luminance, L is the tone mapped luminance, and σ is the scotopic sensitivity. The constant coefficients in the monochromatic part account for the blue shift of the subjective hue of colors for the night scenes.

During the tone mapping process, in every pass we render a full screen quadrilateral and let the fragment shader visit each texel. The pixel shader computing the luminance for each pixel is:

```
texture SourceTex; // source HDR image
float Luminance(in float2 Tex : TEXCOORD0)
: COLOR
{
   float3 col = tex2D(SourceTex, Tex).rgb;
   return dot(col, float3(0.21, 0.71, 0.08));
}
```

In order to downscale the luminance image, Gaussian filter is used, which is implemented according to the proposed importance sampling method. The final pass takes the average luminance value of the neighborhood and scales the color accordingly:

```
texture AvgLumTex;

float4 FinalPS(float2 Tex : TEXCOORD0):COLOR
{
   float key = 1.03 - 2/
       (2+log10(tex2D(AvgLumTex,Tex).r+1));

   float relLum=key*tex2D(LumTex, Tex).r /
                 tex2D(AvgLumTex, Tex).r;
   float Lum = relLum / (1+relLum);
   float4 col = tex2D(SourceTex, Tex) * Lum;
   float gamma = float3(1.05,0.97,1.27)
   return pow(col * gamma, 1/2.2);
}
```

5 CONCLUSION

The proposed filtering method can be effectively implemented on current GPUs. With this variation of the Gaussian filter we can achieve interactive framerates during the tone mapping process. The sample shaders has been implemented in HLSL and integrated into a game engine. Our test application runs on NV7800 GPU. Without tone mapping the average frame rate is around 380 fps, with the suggested filtering method we added postprocessing effects to the application. With tone mapping the average frame rate is around 300 fps.

The reviewed tone mapping algorithm eliminates the typical problems of global methods and preserves the details of the images. The local adaptation part is modified to take account the large variation of the luminance values of the pictures. This removes the disturbing halo artifacts at the luminance jumps. With the simulation of the temporal adaptation of human eyes, it can be used to tone map image streams.

6 ACKNOWLEDGEMENT

This work has been supported by OTKA (T042735), GameTools FP6 (IST-2-004363) project, and by the National Office for Research and Technology (Hungary)

REFERENCES

[1] R.J. Deelay, N. Drasdo, and W. N. Charman. A simple parametric model of the human ocular modulation transfer function. In *Ophthalmology and Physiological Optics*, pages 91–93, 1991.

[2] Nolan Goodnight, Rui Wang, and Greg Humphreys. Interactive time-dependent tone mapping using programmable graphics hardware. In *Proceedings of Eurographics Symposium on Rendering 2003*. ACM SIGGRAPH, June 2003.

[3] Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Perceptual effects in real-time tone mapping. In Bert Jüttler, editor, *Spring Conference on Computer Graphics 2005*, pages 195–202, Budmerice, Slovakia, 2005. ACM.

[4] Erik Reinhard. Parameter estimation for photographic tone reproduction. In *Journal of Graphics Tools*, 2002.

[5] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging*. Morgan Kaufman, 2005.

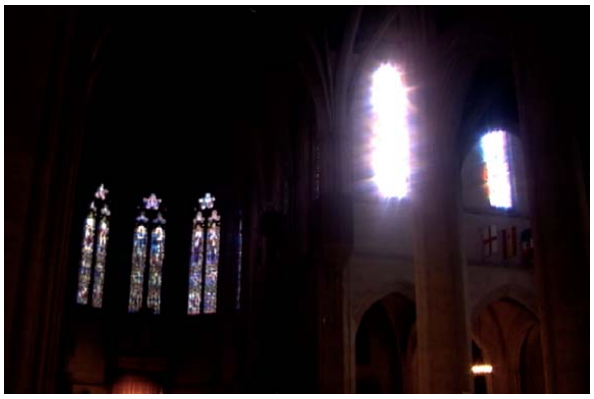
Figure 4: Original image without tone mapping

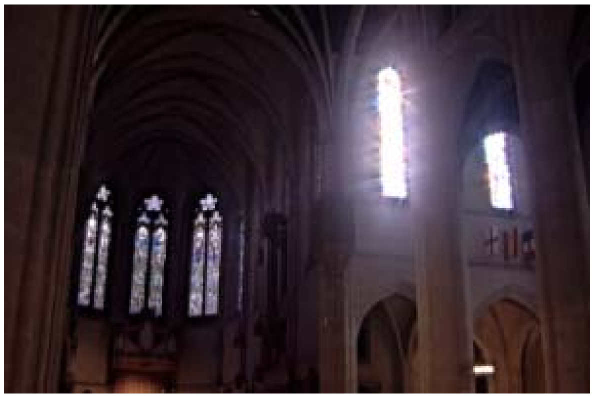
Figure 5: Tone mapping with normal light conditions

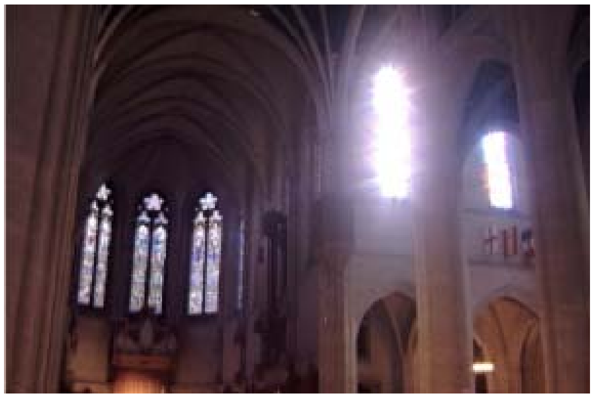
Figure 6: Tone mapping with bright light conditions

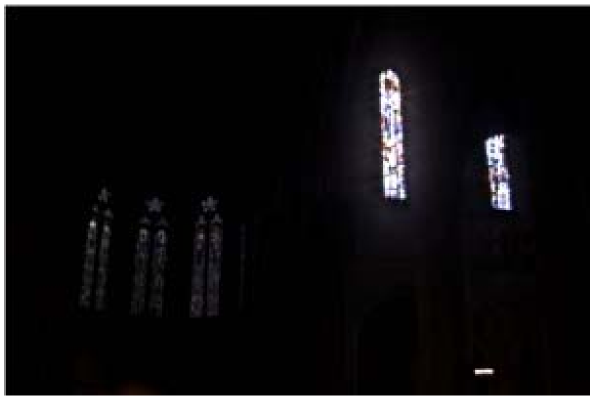
Figure 7: Tone mapping with low light conditions