Real-Time Ray Tracing Denoising

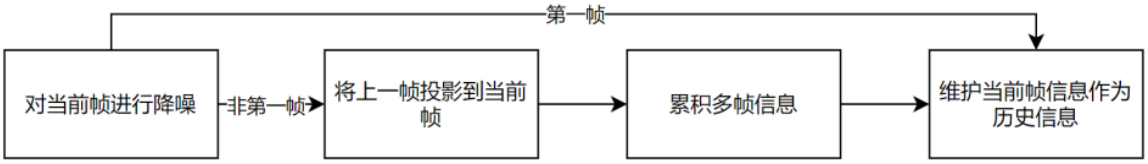
Mengzhu Wang

2024 年 5 月 19 日

1 Overview

实时光线追踪降噪方法中常用的一种是基于 temporal 的降噪，主要思想是假设前一帧被降噪且能复用，通过 motion vector 找到当前帧对应到上一帧所在位置，并进行混合。可以分为三个部分：

- 单帧图像的降噪
- 计算 motion vector 投影上一帧结果
- 累积帧间信息



2 Spatial Filtering

联合双边滤波采用 G-buffer 引导滤波。因为在渲染中可以通过光栅化免费获取很多 G-buffer 的信息，如深度法线、颜色等。光栅化过程与多次 bounce 无关，因此 G-buffer 也是没有噪声的。

对有噪声的输入图像 $\tilde{C}$ 使用联合双边滤波核 J 进行降噪，最终得到降噪后的图像 $\bar{C}^(J)$，其中联合双边滤波核定义如下：

$$J(i,j) = \exp(-\frac{\|i-j\|^2}{2\sigma_p^2} - \frac{\|\tilde{C}[i]-\tilde{C}[j]\|^2}{2\sigma_c^2} - \frac{D_{normal}(i,j)^2}{2\sigma_n^2} - \frac{D_{plane}(i,j)^2}{2\sigma_d^2})$$

确定滤波核的大小后，在该范围内遍历每个像素，计算滤波结果作为权重，累加图像加权的结果和权重和，结束循环后用图像加权和除以权重和作为结果。对应的伪代码如下：

```
For each pixel i
    sum_of_weights = sum_of_weighted_values = 0.0
```

```
    For each pixel j around i
        Calculate the weight w_ij = G(|i - j|, sigma)
        sum_of_weighted_values += w_ij * C^{input}[j]
        sum_of_weights += w_ij
    C^{output}[I] = sum_of_weighted_values / sum_of_weights
```

对于 $D_{normal}(i,j)$ 项，考虑一个立方体任意相邻的两个面不希望互相有贡献，因此定义为两个法线间的夹角，即：

$$D_{normal}(i,j) = arccos(Normal[i] \cdot Normal[j])$$

对于 $D_{plane}(i,j)$ 项，考虑一本书放在一张桌子上，桌子和书的平面完全平行。不希望书和桌子上的像素会互相贡献。因此定义为点 i 到点 j 的单位向量与 i 点法线的点积，即

$$D_{plane}(i,j) = Normal[i] \cdot \frac{Position[j] - Position[i]}{\|Position[j] - Position[i]\|}$$

$D_{plane}(i,j)$ 是一种比只简单计算两个深度的差值更好的指标。在 Cornell box 场景中，左右两侧的墙几乎平行于视线方向，即深度变化较快。在这种情况下，简单的深度差值会使得同一面墙上的很多像素点无法贡献到这个墙本身。

```
Buffer2D<Float3> Denoiser::Filter(const FrameInfo &frameInfo) {
    int height = frameInfo.m_beauty.m_height;
    int width = frameInfo.m_beauty.m_width;
    Buffer2D<Float3> filteredImage = CreateBuffer2D<Float3>(width, height);
    int kernelRadius = 16;
#pragma omp parallel for
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            // TODO: Joint bilateral filter
            int x_start = std::max(0, x - kernelRadius);
            int x_end = std::min(width - 1, x + kernelRadius);
            int y_start = std::max(0, y - kernelRadius);
            int y_end = std::min(height - 1, y + kernelRadius);
            Float3 normal_center = frameInfo.m_normal(x, y);
            Float3 pos_center = frameInfo.m_position(x, y);
            Float3 color_center = frameInfo.m_beauty(x, y);
            float w_sum = 0.f;
            for (int i = x_start; i <= x_end; i++)
                for (int j = y_start; j <= y_end; j++) {
                    Float3 pos_cur = frameInfo.m_position(i, j);
                    float w_dis = SqrDistance(pos_center, pos_cur) /
                        (2.0f * m_sigmaCoord * m_sigmaCoord);

                    Float3 color_cur = frameInfo.m_beauty(i, j);
```

```
                    float w_color = SqrDistance(color_center, color_cur) /
                        (2.0f * m_sigmaColor * m_sigmaColor);

                    Float3 normal_cur = frameInfo.m_normal(i, j);
                    float D_normal = SafeAcos(Dot(normal_center, normal_cur));
                    float w_normal = (D_normal * D_normal) /
                        (2.0f * m_sigmaNormal * m_sigmaNormal);

                    float D_plane = 0.f;
                    if (w_dis > 0.f)
                        D_plane = Dot(normal_center, Normalize(pos_cur - pos_center));
                    float w_plane = (D_plane * D_plane) /
                        (2.0f * m_sigmaPlane * m_sigmaPlane);

                    float w = std::exp(-w_dis - w_color - w_normal - w_plane);

                    w_sum += w;
                    filteredImage(x, y) += frameInfo.m_beauty(i, j) * w;
                }
            }
            if (w_sum == 0.f) filteredImage(x, y) = frameInfo.m_beauty(x, y);
            else filteredImage(x, y) /= w_sum;
        }
    }
    return filteredImage;
}
```

需要注意的是，代码中计算 D_{plane} 部分 `Normalize(pos_cur - pos_center)` 由于遍历时必然出现 cur 是 center 的情况，即方向向量模为 0，此时无法做除法，因此需要排除这种情况，否则会出现 `Runtime Error`。

3 Back Projection

计算当前帧每个像素在上一帧的对应点，并将上一帧的结果投影到当前帧。利用已知的几何信息来找到对应的上一帧像素，公式如下：

$$Screen_{i-1} = P_{i-1}V_{i-1}M_{i-1}M_i^{-1}World_i$$

其中下角标的 i 代表第 i 帧，M 表示物体坐标系到世界坐标系的矩阵，V 表示世界坐标系到摄像机坐标系的矩阵，P 表示摄像机坐标系到屏幕坐标系的矩阵。

在找到对应像素后，需要检查是否合法。这里检查两个指标：一是上一帧是否在屏幕内，二是上一帧和当前帧的物体的标号。然后将保存在 `m_accColor` 的上一帧的结果投影到当前帧，同样保

存在 m_accColor，并将投影是否合法保存在 m_valid 以供在累积多帧信息时使用。

```
void Denoiser::Reprojection(const FrameInfo &frameInfo) {
    int height = m_accColor.m_height;
    int width = m_accColor.m_width;
    Matrix4x4 preWorldToScreen =
        m_preFrameInfo.m_matrix[m_preFrameInfo.m_matrix.size() - 1];
    Matrix4x4 preWorldToCamera =
        m_preFrameInfo.m_matrix[m_preFrameInfo.m_matrix.size() - 2];
#pragma omp parallel for
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            // TODO: Reproject
            m_valid(x, y) = false;
            m_misc(x, y) = Float3(0.f);

            float cur_id = frameInfo.m_id(x, y);
            if (cur_id < 0.f) continue;
            Float3 curWorldCoord = frameInfo.m_position(x, y);
            Matrix4x4 curWorldToObj = Inverse(frameInfo.m_matrix[cur_id]);
            Matrix4x4 preObjToWorld = m_preFrameInfo.m_matrix[cur_id];
            Float3 preScreenCoord = preWorldToScreen(preObjToWorld(curWorldToObj(
                curWorldCoord, Float3::Point), Float3::Point), Float3::Point);

            if (preScreenCoord.x < 0 || preScreenCoord.x >= width ||
                preScreenCoord.y < 0 || preScreenCoord.y >= height) {
                continue;
            }
            float pre_id = m_preFrameInfo.m_id(preScreenCoord.x, preScreenCoord.y);
            if (pre_id != cur_id) {
                continue;
            }
            m_valid(x, y) = true;
            m_misc(x, y) = m_accColor(preScreenCoord.x, preScreenCoord.y);
        }
    }
    std::swap(m_misc, m_accColor);
}
```

4 Temporal Accumulation

将已经降噪的当前帧图像 $\bar{C}_i$，与已经降噪的上一帧图像 $\bar{C}_{i-1}$ 进行结合，公式如下：

$$\bar{C}_i \leftarrow \alpha \bar{C}_i + (1-\alpha) Clamp(\bar{C}_{i-1})$$

若上一帧没有找到合法的对应点时，将 α 设为 1，否则设置为 0.2。

对于 Clamp 部分，首先需要计算 $\bar{C}_i$ 在 7×7 的邻域内的均值 μ 和方差 σ，然后将上一帧的颜色 $\bar{C}_{i-1}$Clamp 在 $(\mu - k\sigma, \mu + k\sigma)$ 范围内。

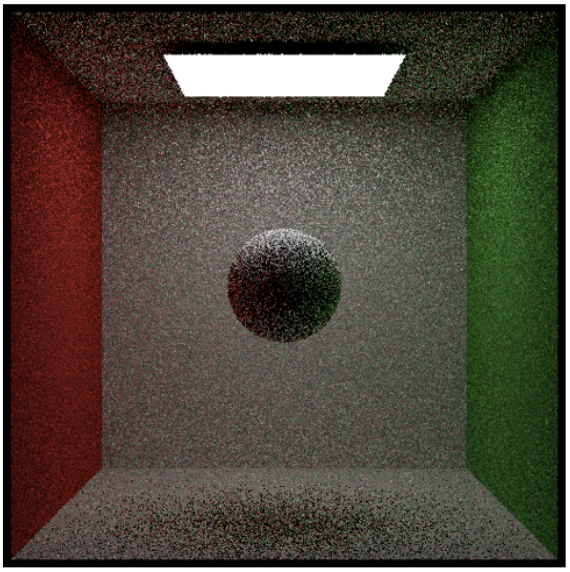
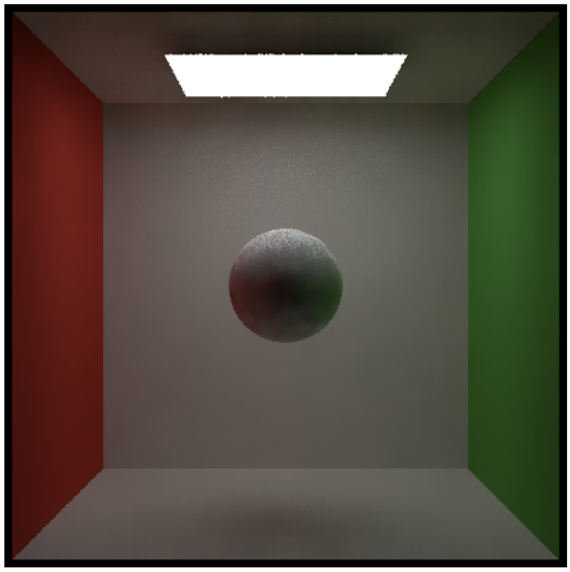
```
void Denoiser::TemporalAccumulation(const Buffer2D<Float3> &curFilteredColor) {
    int height = m_accColor.m_height;
    int width = m_accColor.m_width;
    int kernelRadius = 3;
#pragma omp parallel for
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            // TODO: Temporal clamp
            Float3 color = m_accColor(x, y);
            // TODO: Exponential moving average
            float alpha = 1.0f;
            if (m_valid(x, y)) {
                alpha = m_alpha;

                int x_start = std::max(0, x - kernelRadius);
                int x_end = std::min(width - 1, x + kernelRadius);
                int y_start = std::max(0, y - kernelRadius);
                int y_end = std::min(height - 1, y + kernelRadius);
                Float3 mu, sigma;
                for (int i = x_start; i <= x_end; i++)
                    for (int j = y_start; j <= y_end; j++) {
                        mu += curFilteredColor(i, j);
                        sigma += Sqr(curFilteredColor(x, y) - curFilteredColor(i, j));
                    }
                float cnt = (2 * kernelRadius + 1) * (2 * kernelRadius + 1);
                mu /= float(cnt);
                sigma = SafeSqrt(sigma / float(cnt));
                color = Clamp(color, mu - sigma * m_colorBoxK, mu + sigma * m_colorBoxK);
            }
            m_misc(x, y) = Lerp(color, curFilteredColor(x, y), alpha);
        }
    }
```

```
    std::swap(m_misc, m_accColor);
}
```

5 Run

```
mkdir build
cd build
cmake ..
make -j8
./Denoise
```

运行结果保存在 `example/box(pink-room)/output` 中,为 OpenEXR 格式。接下来运行 `image2video.sh` 借助 ffmpeg 来将图像序列转换为视频。

pink room 降噪时需要调整 `m_sigmaColor` 为 10 左右。