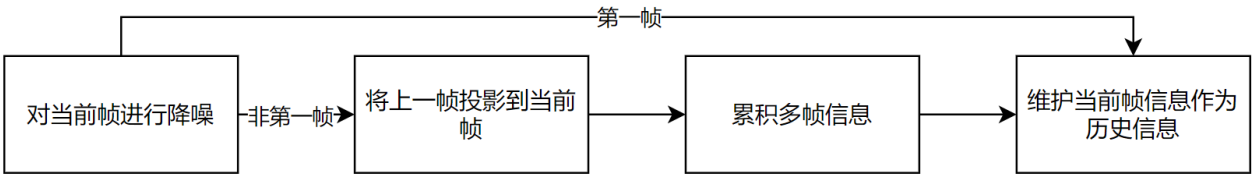
1 总览

我们需要实现一个**简单的**实时光线追踪的降噪方法。光线追踪的渲染结果，G-Buffer 及其他相关信息会以文件的形式提供给大家。本次作业的重点在于对渲染结果的降噪，而不是如何进行渲染。为了在作业框架中实现上述目标，我们将作业分为三个部分：单帧图像的降噪，计算 motion-vector，累积帧间信息。

2 作业流程

对于本次作业，我们的整体流程如下图所示：



你需要在 src/denoiser.cpp 中实现缺失的内容。在 src/main.cpp 中，我们会将当前帧的信息通过函数 **Denoiser::ProcessFrame(FrameInfo)** 传入，并得到降噪后的图像，在 **Denoiser** 类中我们会维护住历史帧的信息。

结构体 **FrameInfo** 中保存着一帧的信息，

- **m_beauty** 代表渲染结果。
- **m_depth** 代表每个像素的深度值。
- **m_normal** 代表每个像素的法线向量 (模长为 1)。
- **m_position** 代表每个像素在世界坐标系中的位置。
- **m_id** 代表每个像素对应的物体标号，对于没有物体的部分 (背景) 其标号为-1。
- **m_matrix** 中保存了多个矩阵。**m_matrix[i]** 表示标号为 i 的物体从物体坐标系到世界坐标系的矩阵。此外，**m_matrix** 中的倒数第 2 个和倒数第 1 个分别为，世界坐标系到摄像机坐标系和世界坐标系到屏幕坐标系 ($[0, W)\times[0, H)$) 的矩阵。

2.1 单帧降噪

在这个部分，你需要对有噪声的输入图像 $\widetilde{C}$，使用**联合双边滤波**核 J 进行降噪，最终得到降噪后的图像 $\overline{C}^{(J)}$，我们的联合双边滤波核定义如下：

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

对于 $D_{\text{normal}}(\mathbf{i},\mathbf{j})$ 项，我们可以考虑以下的情况：对于一个立方体，任意相邻的两个面我们都不希望它们互相有贡献，因此我们定义 D_{normal} 项为两个法线间的夹角 (弧度)，即：

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

对于 $D_{\text{plane}}(\mathbf{i},\mathbf{j})$ 项，我们可以考虑以下的情况：有一本书放在一张桌子上，桌子和书的平面完全平行。我们一定不希望书和桌子上的像素会互相贡献。因此，我们定义这项为点 $\mathbf{i}$ 到 $\mathbf{j}$ 的单位向量与 $\mathbf{i}$ 点法线的点积，即：

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

此外，$D_{\text{plane}}(\mathbf{i},\mathbf{j})$ 项提供了一种比只是简单计算两个深度的差值更好的指标。考虑 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。

这里你需要完成函数 **Denoiser::Filter**，它的输入参数为当前帧的信息，返回降噪后的图像。

2.2 投影上一帧结果

在这个部分，你需要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。

我们利用已知的几何信息来找到对应的上一帧像素，公式如下：

$$\text{Screen}_{i-1} = P_{i-1}V_{i-1}M_{i-1}M^{-1}\text{World}_i$$

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，我们需要检查是否合法，这里我们使用两个简单的指标来检查：

- 上一帧是否在屏幕内。
- 上一帧和当前帧的物体的标号。

这里你需要完成函数 **Denoiser::Reprojection**，它的输入参数为当前帧的信息。该函数会将上一帧的结果 (保存在 **m_accColor**) 投影到当前帧 (保存在 **m_accColor**)。并将投影是否合法保存在 **m_valid** 以供我们在累积多帧信息时使用。

2.3 累积多帧信息

在这个部分，你需要将已经降噪的当前帧图像 $\overline{C}_i$，与已经降噪的上一帧图像 $\overline{C}_{i-1}$ 进行结合，公式如下：

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

其中对于 α 的选择，当我们在上一帧没有找到合法的对应点时，将 α 设为 1。

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

这里你需要完成函数 **Denoiser::TemporalAccumulation**，它的输入参数为我们降噪过的当前帧图像。该函数会将最终结果保存在 **m_accColor**，这也就是我们最终的降噪结果。

2.4 提示

- 在 src/util/mathutil.h 中我们提供了一些简单的数学函数。
- 一些参数 ($\alpha, \sigma_p, \sigma_c, \sigma_n, \sigma_d$，k) 定义在类 **Denoiser** 中。
- 由于本次作业的降噪算法比较简单，它在某些场景下的降噪的效果并不会非常好，你可以通过调节参数来取舍一些效果。

- 我们推荐你首先实现单帧的降噪，然后将上一帧的结果投影到当前帧，最后实现对多帧信息的累积。
- 我们的输入和输出为 OpenEXR 格式，你可以使用tev来查看以及保存图像为其他格式。

2.5 编译

根据你的系统，你可以选择运行 build.bat 或 build.sh 来构建和编译作业框架。

3 数据与参考结果

我们本次提供两个场景：box 和 pink room。由于本次数据较大，需要单独下载，具体下载连接请查看论坛公告。下载后你会得到一个 example 文件夹，推荐放在 Denoise 文件夹内。你需要在 src/main.cpp 中的 **main()** 函数中将文件路径设置正确。

参考结果放在各自场景文件下的 reference 文件夹内，包括 box/pinkroom-filter.mp4，box/pinkroom-project.mp4 和 box/pinkroom-result.mp4，分别代表只对单帧图像降噪，不对单帧图像降噪只做多帧投影和累积，以及完整降噪的结果。以上结果仅供参考，最终评分时效果合理即可。

4 评分与提交

提交时需要删除/build, /examples 文件夹，并建立 results 文件夹，将你输出的图像序列转为视频保存在 results 文件夹中。同时在 README.md 中简要描述本轮工作，尤其是完成了作业的哪些部分。如果完成的任务中包含了提高部分请注明改动了哪些代码、文件以方便助教同学批改。

你可以使用 ffmpeg 来将图像序列转换为视频。