2020_07_19__采样_图像重构.md

目录

图像重构

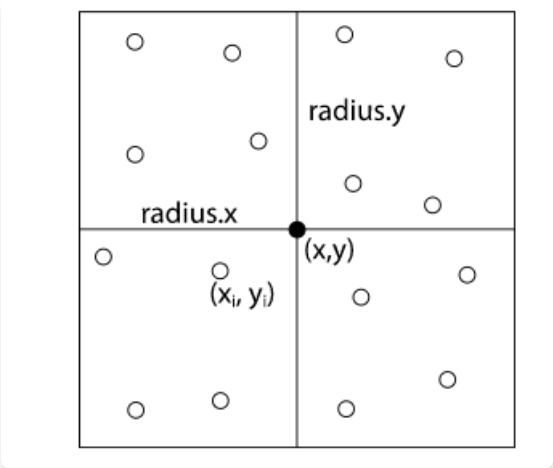
当我们选择了恰当的采样器后, 我们需要把样本, 和他们计算出来的辐射度, 转换成用于显示的像素值, 或者存储起来。

根据信号处理的理论, 我们需要做这三件事情, 来计算最终的图像上的每一个像素的值

- 从图像样本中, 重构一个连续的图形函数 L
- 对L 进行预处理, 将频率过高的部分, 也就是因为像素间距而不能表现的细节, 做一次过滤
- 在像素的具体位置, 对 L 进行采样, 并计算最终的像素值

因为我们知道, 在最后, 对 L 进行采样的是, 只需要在像素的位置即可。所以我们不需要重构一个连续的函数表达。对于第一步和第二步, 可以整合为一个 滤波器即可。

注意到, 如果我们使用, 比 **Nyquist频率** 更高的频率去采样, 并且使用 sinc 去做重构。那么能得到完美的原始图像。(这是因为, 我们纯粹的使用了点采样)。但往往我们达不到这么高的采样率, 所以使用非均匀采样, 在 aliasing 和 噪声 之间做权衡。



1

为了计算实心处的 (x, y) 的像素值, 在 radius.x 和 radius.y 之内的值都要被考虑。我们对其做加权平均

$$I(x,y) = \frac{\sum_i f(x-x_i, y-y_i)\, w(x_i,y_i)\, L(x_i,y_i)}{\sum_i f(x-x_i, y-y_i)},$$

2

- $L(x_i, y_i)$ 表示第 i 个样本, 在 (x_i, y_i) 处的辐射度
- $w(x_i, y_i)$ 表示的是权重
- f 表示的是滤波函数

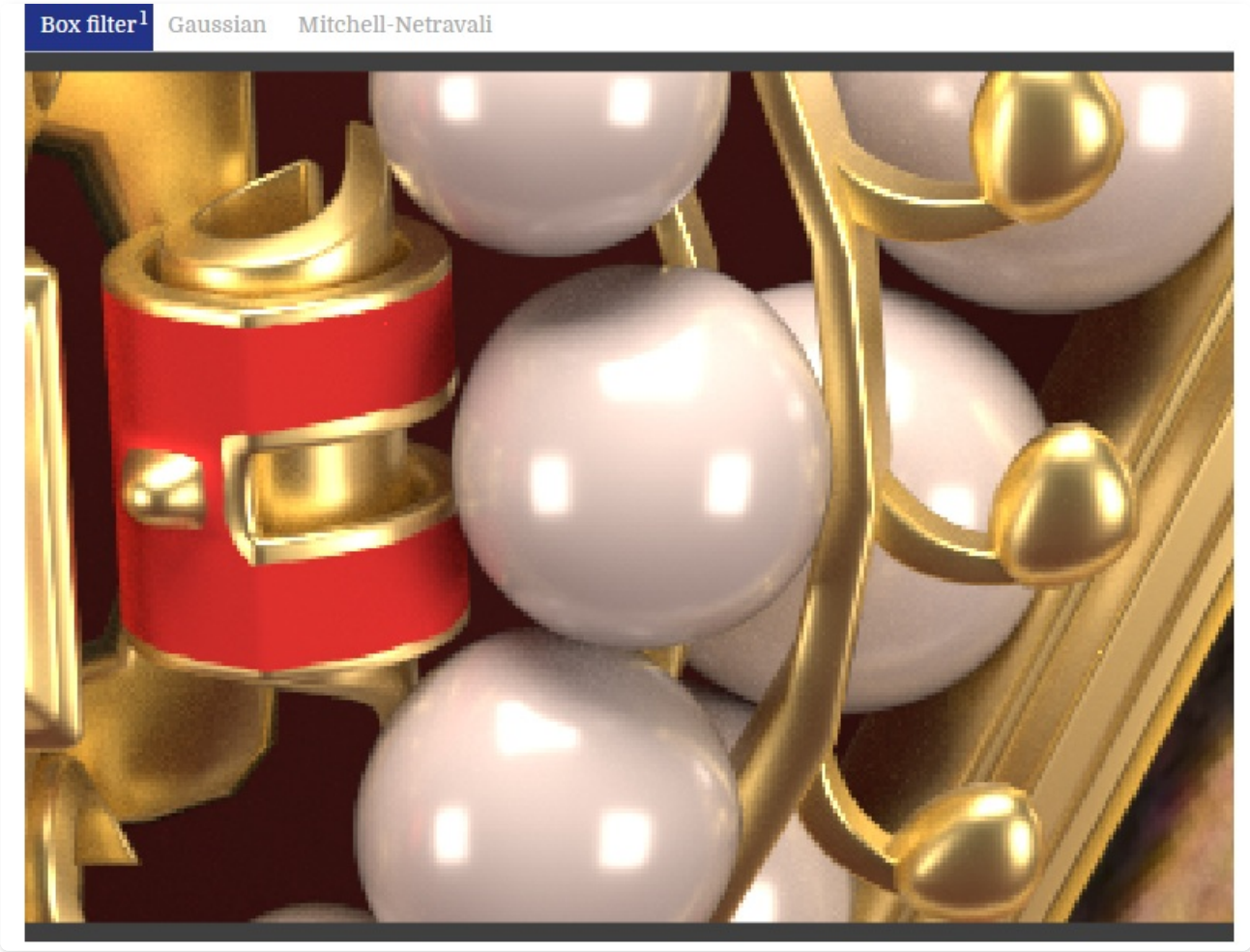
在这里, sinc滤波并不是最佳的选择, 因为对于超出频率极限的滤波, 容易产生振铃。而且 sinc 的波形是无限长

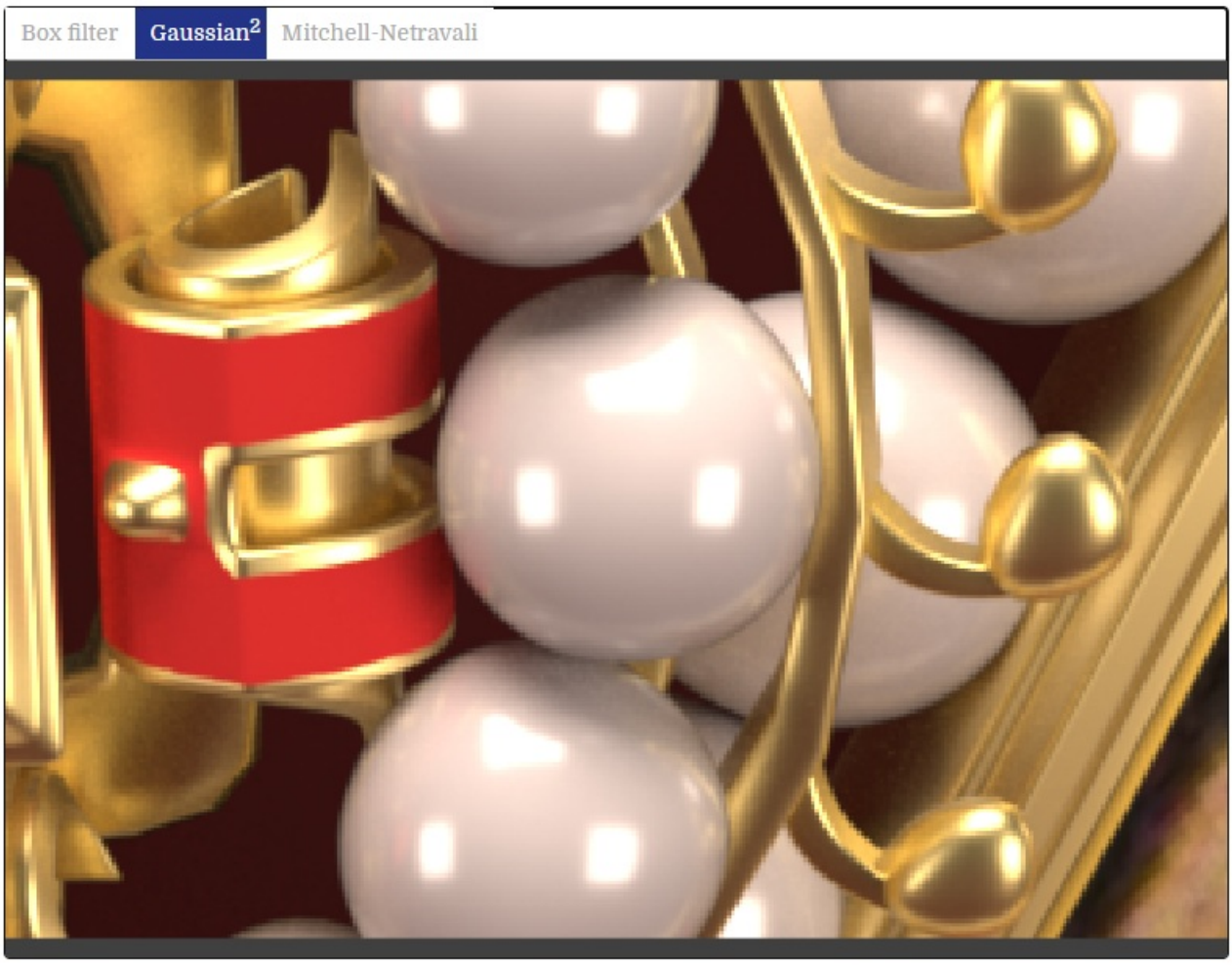
的, 所以离中心

滤波函数

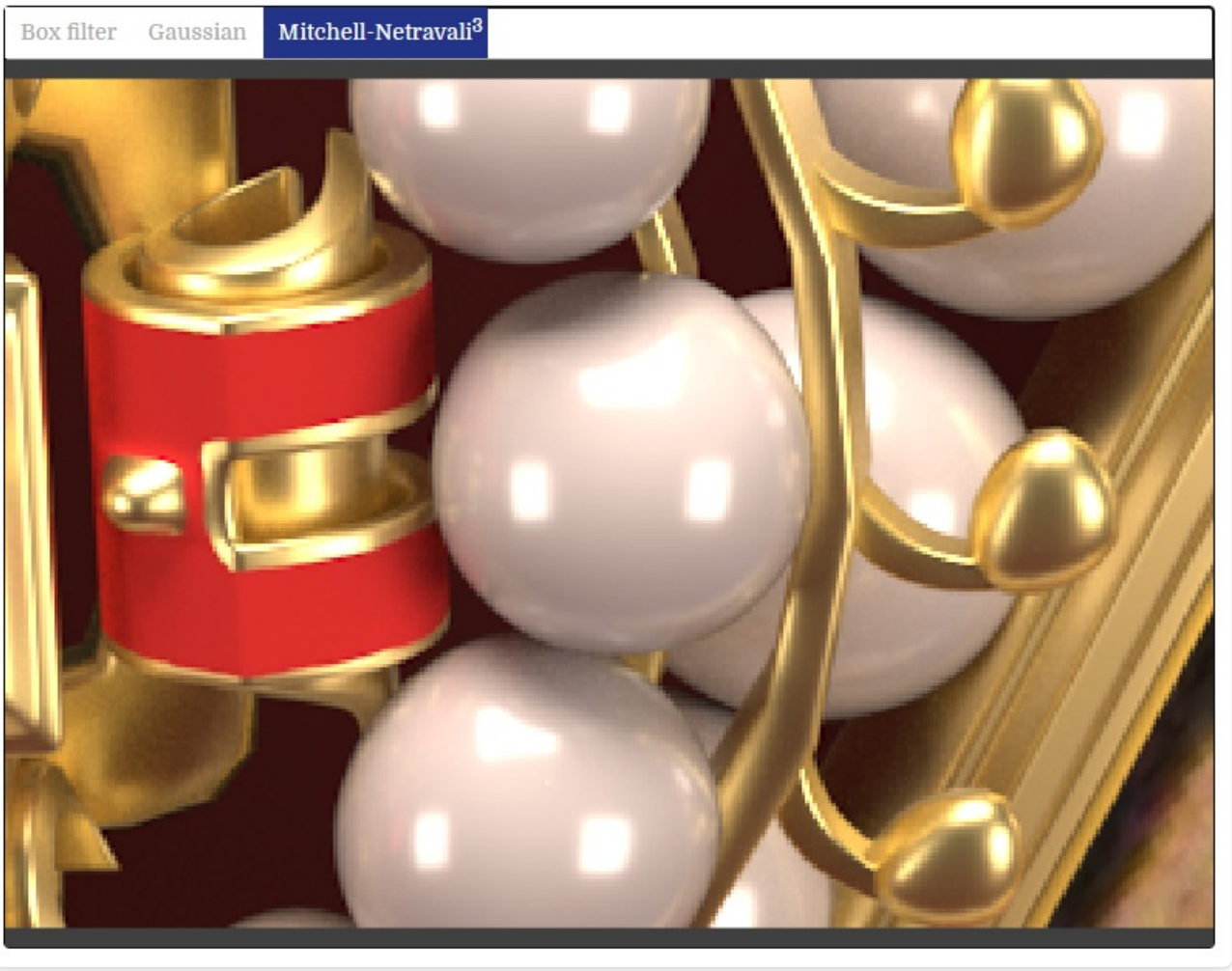
所有的滤波函数都继承 `Filter` 类, 提供对应的滤波函数。`Film` 类会存储一个指向 `Filter` 的指针, `Film` 类在下一节中介绍。

下面介绍了几种滤波器, 使用不同的滤波器, 会对最终成型的图像有不同的影响:





4



5

- 高斯滤波：会模糊边缘
- Mitchell滤波：最清晰
- Box滤波：是最差的滤波器，因为sinc混了高频的信息进去

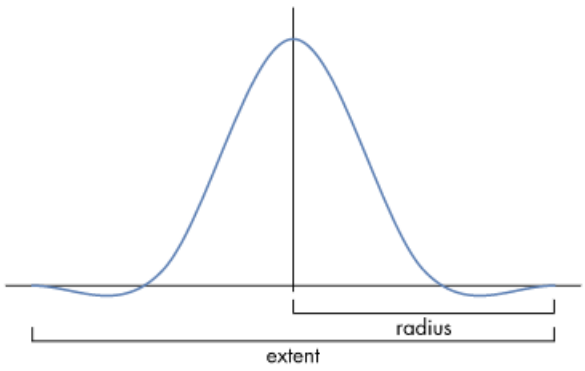


Figure 7.40: The extent of filters in `pbrt` is specified in terms of their radius from the origin to its cutoff point. The support of a filter is its total non-zero extent, here equal to twice its radius.

6

pbrt的滤波器, 有一个 extent 的定义, 其范围是根据从原点到截止点的半径来确定的。extent 是其半径的两倍。

```
45  namespace pbrt {
46
47  // Filter Declarations
48  class Filter {
49    public:
50      // Filter Interface
51      virtual ~Filter();
52      Filter(const Vector2f &radius)
53          : radius(radius), invRadius(Vector2f(1 / radius.x, 1 / radius.y)) {}
54      // 滤波器需要实现的唯一接口
55      // 传入一个点，这是采样点相对于滤波器中心的位置
56      // 返回滤波的值
57      // 传入的值，永远保证在滤波的半径范围内，所以无需检查
58      virtual Float Evaluate(const Point2f &p) const = 0;
59
60      // Filter Public Data
61      // 滤波的半径，在xy两个方向上有范围，超过了就是 0，预先存储倒数做计算上的优化
62      const Vector2f radius, invRadius;
63  };
64
65  } // namespace pbrt
66
67  #endif // PBRT_CORE_FILTER_H
```

7

Box 滤波器

Box 的滤波器是最常见的滤波器。盒型滤波就是对图像正方形区域内的所有样本做平均加权。虽然效率是最高的, 但是最差的滤波器。

下图是盒型滤波器和三角形滤波器的示意图:

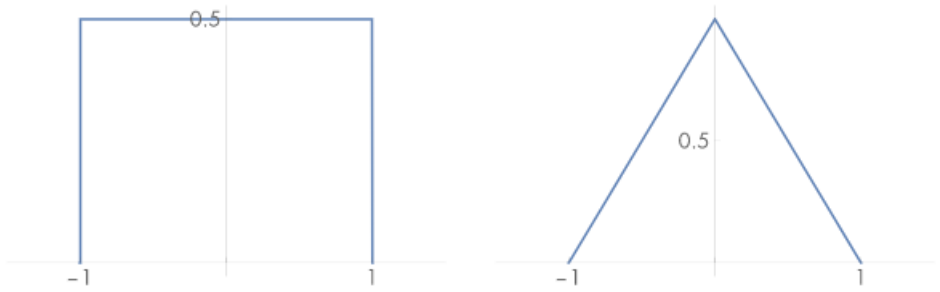


Figure 7.41: Graphs of the (a) box filter and (b) triangle filter. Although neither of these is a particularly good filter, they are both computationally efficient, easy to implement, and good baselines for evaluating other filters.

8

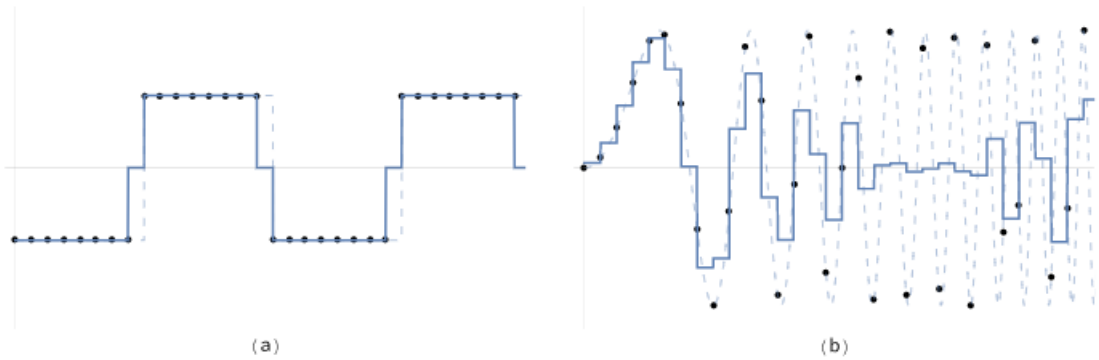


Figure 7.42: The box filter reconstructing (a) a step function and (b) a sinusoidal function with increasing frequency as x increases. This filter does well with the step function, as expected, but does an extremely poor job with the sinusoidal function.

9

上图是, 盒型滤波器, 重构 a 阶跃函数, 和 b 正弦曲线的重构。可以看到, 对 sin 函数的重构表现很差。

```
40  // Box Filter Method Definitions
41  // Box 的频率就是 1，他就是取平均数
42  Float BoxFilter::Evaluate(const Point2f &p) const { return 1.; }
43
```

10

三角 滤波器

三角滤波器的效果比盒型滤波器好, 他是在滤波器的平方范围内, 权重从中心开始最高, 到边缘下降

```
40  // Triangle Filter Method Definitions
41  // 三角滤波，中心的权重是最高的，越往旁边，权重越低
42  Float TriangleFilter::Evaluate(const Point2f &p) const {
43      return std::max((Float)0, radius.x - std::abs(p.x)) *
44             std::max((Float)0, radius.y - std::abs(p.y));
45  }
```

11

这就是在中心, 权重会很高, 越在旁边, 权重就越低

高斯 滤波器

高斯滤波器, 使用的是高斯函数, 但是, 在大于 Radius 的范围外, 我们将其置为 0。

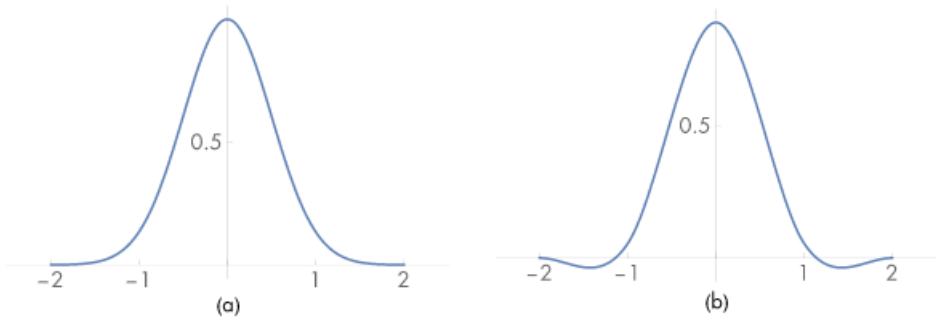


Figure 7.43: Graphs of (a) the Gaussian filter and (b) the Mitchell filter with $B = \frac{1}{3}$ and $C = \frac{1}{3}$, each with a width of 2. The Gaussian gives images that tend to be a bit blurry, while the negative lobes of the Mitchell filter help to accentuate and sharpen edges in final images.

12

这是高斯滤波器, 和米切尔滤波器的对比。相比来说, 高斯滤波器会带来模糊, 而米切尔滤波器, 会锐化图像的边缘。

1D 的高斯函数是:

$$f(x) = \mathrm{e}^{-\alpha x^2} - \mathrm{e}^{-\alpha r^2},$$

13

- α 控制滤波器衰减的速率。这个值越小, 图像成像越模糊。
- 第二个因子, 保证了 x 在到达 r 时, 图像平缓下降到 0

为了效率运算, 第一项可以直接计算出结果。

```cpp
// Gaussian Filter Declarations
class GaussianFilter : public Filter {
  public:
    // GaussianFilter Public Methods
    GaussianFilter(const Vector2f &radius, Float alpha)
        : Filter(radius),
          alpha(alpha),
          expX(std::exp(-alpha * radius.x * radius.x)),
          expY(std::exp(-alpha * radius.y * radius.y)) {}
    Float Evaluate(const Point2f &p) const;

  private:
    // GaussianFilter Private Data
    const Float alpha;
    // 提前计算第一项，提高效率
    const Float expX, expY;

    // GaussianFilter Utility Functions
    // 计算高斯函数
    Float Gaussian(Float d, Float expv) const {
        return std::max((Float)0, Float(std::exp(-alpha * d * d) - expv));
    }
};
```

14

```cpp
// Gaussian Filter Method Definitions
// 2 阶的高斯函数 等于 1阶的高斯函数的乘积
Float GaussianFilter::Evaluate(const Point2f &p) const {
    return Gaussian(p.x, expX) * Gaussian(p.y, expY);
}
```

15