Real-Time High Quality Rendering

GAMES202, Lingqi Yan, UC Santa Barbara

Lecture 13:
Real-Time Ray Tracing 2

Announcements

- GAMES101 resubmission will start soon
- GAMES202 homework 3 has been released
  - Due Jun 12
- GAMES202 homework 4 has almost finished
  - Will be about implementing Kulla-Conty
- Next Saturday, **last lecture** for GAMES202!

Last Lecture

- Real-Time Ray Tracing
  - Basic idea
  - Motion vector
  - Temporal accumulation / filtering
  - Failure cases

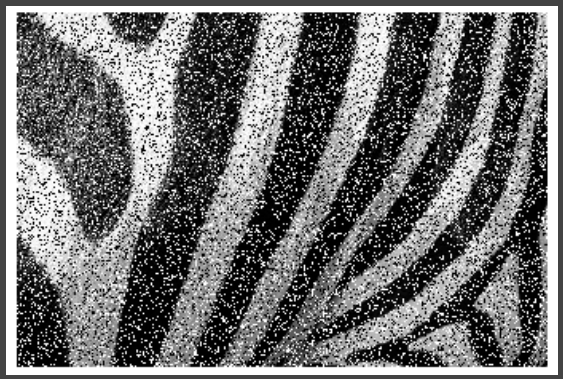
Today

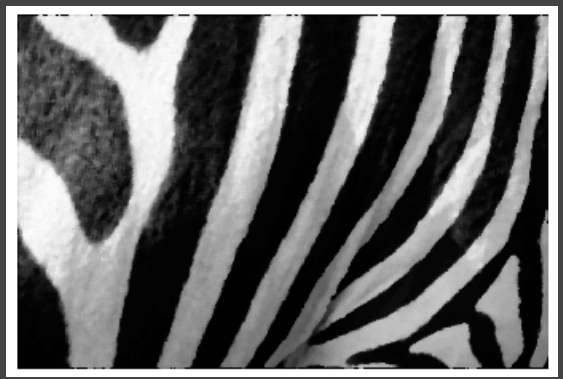
- Implementing a spatial filter
  - Cross / joint bilateral filtering
  - Implementing large filters
  - Outlier removal
- Specific filtering approaches for RTRT
  - Spatiotemporal Variance-Guided Filtering (SVGF)
  - Recurrent AutoEncoder (RAE)

Implementation of Filtering

Implementation of filtering

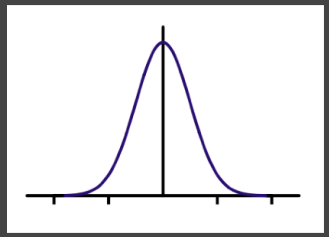
- Suppose we want to (low-pass) filter an image
  - To remove (usually high-frequency) noise
  - Now only focus on the spatial domain
- Inputs
  - A noisy image $\tilde{C}$
  - A filter kernel K, could vary per pixel
- Output — a filtered image $\bar{C}$





Implementation of filtering

- Let's assume a Gaussian filter centered at pixel i (2D)
  - Any pixel j in the neighborhood of i would contribute
  - Based on the distance between i and j



```
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
```

Implementation of filtering

- Some Notes
  - Keep track of sum_of_weights for “normalization”
  - Test whether sum_of_weights is zero (for other kernels)
  - Color can be multi-channel

```
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
```

Bilateral Filtering

Bilateral filtering

双边滤波

- Problem of Gaussian filtering
  - Also blurs the boundary 边界模糊
  - But the boundary is the high frequency that we want to keep

边界是我们希望保留的高频

https://www.mathworks.com/help/images/ref/imgaussfilt.html

Bilateral filtering

- Observation
  - The boundary <-> drastically changing colors
- Idea
  - How to keep the boundary?
  - Let pixel j contribute less if its color is too different to i
  - Simply add more control to the kernel

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_r^2}\right)$$

https://www.mathworks.com/help/images/ref/imgaussfilt.html

Bilateral filtering

- Pretty good results

https://en.wikipedia.org/wiki/Bilateral_filter

Joint Bilateral Filtering

Joint Bilateral filtering

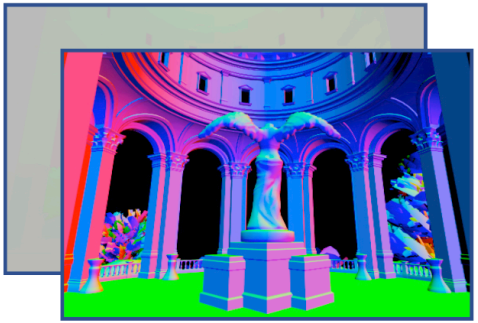
- Observation
  - Gaussian filtering: 1 metric (distance)
  - Bilateral filtering: 2 metrics (position dist. & color dist.)
  - Can we use more “features” to better guide filtering?
- Yes! This is Cross / Joint Bilateral Filtering
- Especially good at denoising path traced rendering results!

Joint Bilateral filtering

- Unique advantages in rendering
  - A lot of **free** “features” known as G-buffers
  - Normal, depth, position, object ID, etc., mostly geometric
- Even better
  - G-buffers are **noise-free** as they are not related to multi-bounces

G-buffers 无噪声. 因为与多次反弹无关.

- You will be implementing this in homework 5

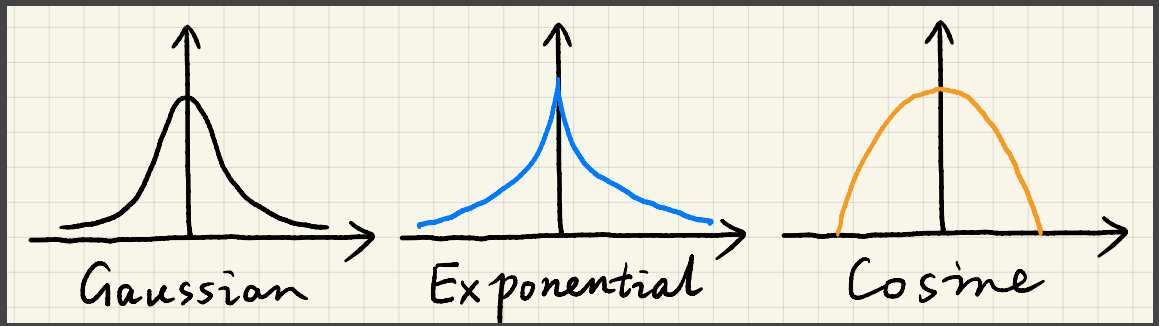


Pos./Normal

Albedo

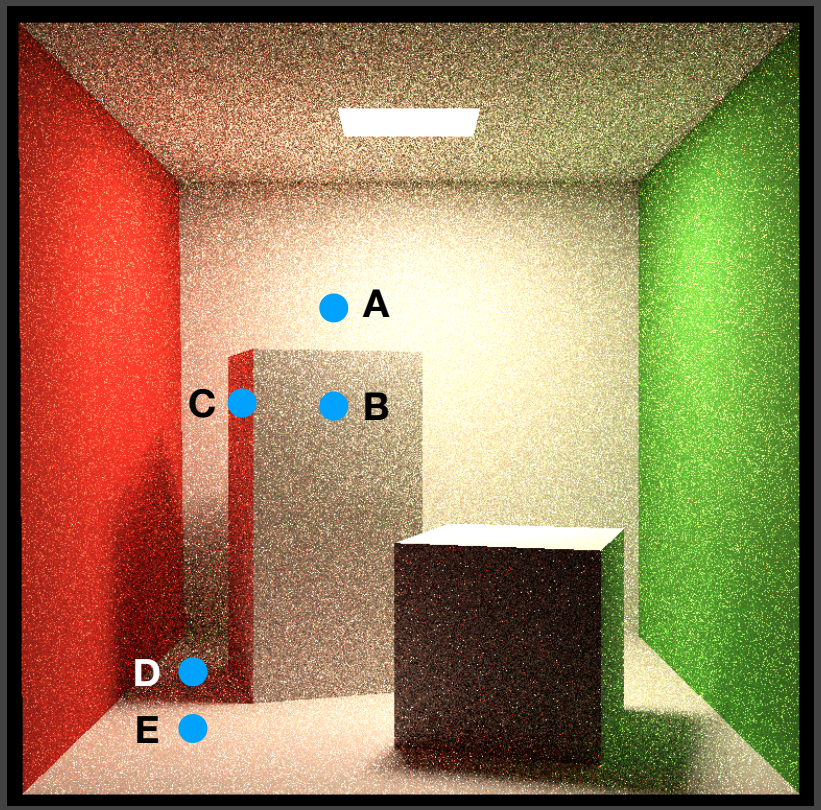
Joint Bilateral filtering — Notes

- The metric itself does not have to be normalized
  - The filtering process does the normalization
- Gaussian is not the only choice
  - Any function that decreases with "distance" would work
  - Exponential (absolute), cosine (clamped), etc.



Joint Bilateral Filtering — Example

- Suppose we consider
  - Depth
  - Normal
  - Color
- Why we do not blur the boundary between
  - A and B: depth
  - B and C: normal
  - D and E: color



Questions?

Implementing Large Filters

Implementing Large Filters

实现大的滤波核

- Recall: for each pixel, we need to loop over all its NxN neighborhood

遍历所有NxN邻域

- Observation
  - For small filters, this is fine (e.g. 7x7)
  - For large filters, this can be prohibitively heavy (e.g. 64x64)
- Two different solutions to large filters

Solution 1: Separate Passes

- Consider a 2D Gaussian filter
  - Separate it into a horizontal pass (1xN) and a vertical pass (Nx1)
  - #queries: $N^2 \rightarrow N + N$

[DOTA 2]

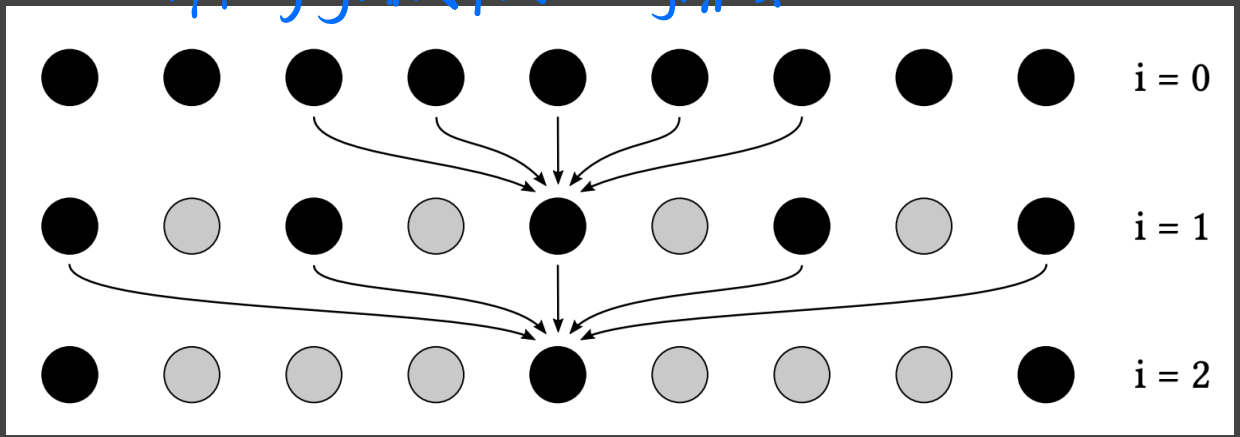
Original | After horizontal filtering | After horizontal + vertical filtering

Solution 1: Separate Passes

- A deeper understanding
  - Why can we separate a 2D Gaussian filter into two 1D Gaussian filters?
- A 2D Gaussian filter kernel is separable
  - $G_{2D}(x, y) = G_{1D}(x) \cdot G_{1D}(y)$
- Recall: filtering == convolution
  - $$\iint F(x_0, y_0) G_{2D}(x_0 - x, y_0 - y)\, \mathrm{d}x\mathrm{d}y = \int \left(\int F(x_0, y_0) G_{1D}(x_0 - x)\, \mathrm{d}x \right) G_{1D}(y_0 - y)\, \mathrm{d}y$$
- So, separate passes require separable filter kernels
  (i.e. **in theory**, bilateral filters cannot be separately implemented)

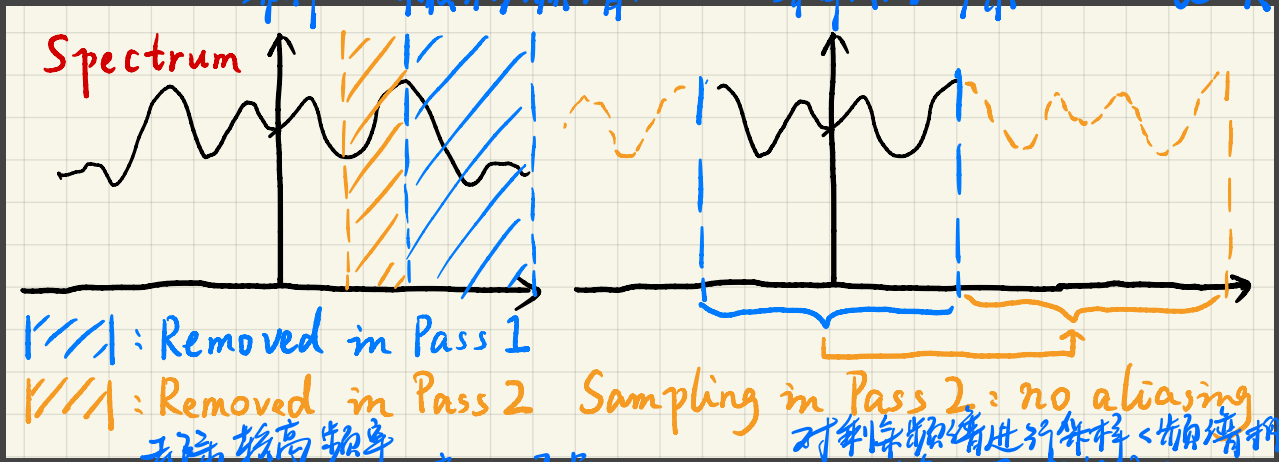
Sol. 2: Progressively Growing Sizes

- Idea: filter multiple times with growing sizes
- Specifically, a-trous wavelet
  - Multiple passes, each is a 5x5 filter
  - The interval between samples is growing (2^i) (save e.g. $64^2 \Rightarrow 5^2 \times 5$)



Sol. 2: Progressively Growing Sizes

- A deeper understanding
  - Why growing sizes?
    - Applying larger filter == removing lower frequencies
  - Why is it safe to skip samples?
    - Sampling == repeating the spectrum



Questions?

(Note: the abovementioned filtering approaches can be applied to denoising PCSS, SSR, etc. in your homework!)

Outlier Removal
(and temporal clamping)

Outlier Removal

- Filtering is not almighty
  - Sometimes the filtered results are still noisy, even **blocky**
  - Mostly due to extremely bright pixels (outliers)
- Idea
  - Can we remove those outliers **BEFORE** filtering?
  - How do we define outliers?

https://clarissewiki.com/4.0/fireflies-filtering.html

Outlier Detection and Clamping

Outlier 检测和剔除.

<去掉outlier之前世界>

- Outlier detection
  - For each pixel, take a look at its e.g. 7x7 neighborhood
  - Compute mean and variance
  - Value outside $[\mu - k\sigma, \mu + k\sigma]$ -> outlier!

eg. k 在 (1,3)

- Outlier removal
  - Clamp any value outside $[\mu - k\sigma, \mu + k\sigma]$ to this range

clamp 到有效范围中.

  - Note: this is NOT throwing away (zeroing out) the outlier

Temporal Clamping

- Recall: directly using the temporal color may result in ghosting
  - This is because $C^{(i-1)}$ can be very different to $\bar{C}^{(i)}$
  - In temporal reuse, we can clamp $C^{(i-1)}$ towards $\bar{C}^{(i)}$ so they'll be close

$$C^{(i)} = \alpha \bar{C}^{(i)} + (1-\alpha) C^{(i-1)}$$

spatial filtering

<7×7> 得 μ, σ

<是噪声和残影的折衷>

$$\Rightarrow \textbf{clamp}(C^{(i-1)}, \mu - k\sigma, \mu + k\sigma)$$

- Notes
  - Temporal clamping is a tradeoff between noise and lagging
  - Clamping $C^{(i-1)}$ towards $\bar{C}^{(i)}$, not the inverse

too different 不再相信.

Questions?

Next Lecture

- Practical Industrial Solutions in RTR

[Unreal Engine 5]

Thank you!