

# Processamento Digital de Imagens

Prof. Bogdan Tomoyuki Nassu



# Hoje

- Implementaremos uma versão simples de *bloom lighting* artificial.
- *Bloom*: é um artefato visual produzido por câmeras reais!
  - Artefato visual: anomalia / erro.
    - No caso, vemos algo que não existe!
    - Efeito: o brilho de regiões claras da imagem “vaza”.
- O efeito *bloom* se tornou **extremamente** comum em jogos após o seu uso em Ico (2001).
  - Este efeito é criticado às vezes, porque:
    - É um artefato!
    - Não funciona como a visão humana.
    - O uso às vezes é considerado exagerado.









# *Bloom*



# *Bloom*



# *Bloom*



If you're a Zelda series fan, you may know what this image is.

# *Bloom*



# *Bloom*



# Aviso

- Antes de tudo: faremos um efeito 100% pós-processamento.
  - → diferenças para o efeito normalmente usado na prática.

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

# Implementação: ideia geral

1. Criamos uma máscara contendo as fontes de luz.
  - Esta máscara é uma imagem!
2. Borramos a máscara, fazendo a luz “vazar” para os arredores.
3. “Colamos” a máscara sobre a imagem original.

# 1. Isolando as fontes de luz

- No primeiro passo, criamos uma versão da imagem onde aparecem somente as fontes de luz.
  - Céu, janelas, lâmpadas, fogo, etc.
- Em implementações mais sofisticadas, as fontes são determinadas em tempo de renderização.
  - Não temos acesso a este tipo de dado, então podemos usar um simples teste “*bright-pass*”.
  - = Uma versão da imagem contendo apenas os pixels com intensidade ou luminância mais alta.
- Nota: em implementações reais, a máscara costuma ter resolução menor do que a imagem final.
  - Isso é feito para acelerar o processamento.
  - Não usaremos este “truque” nos nossos exemplos.







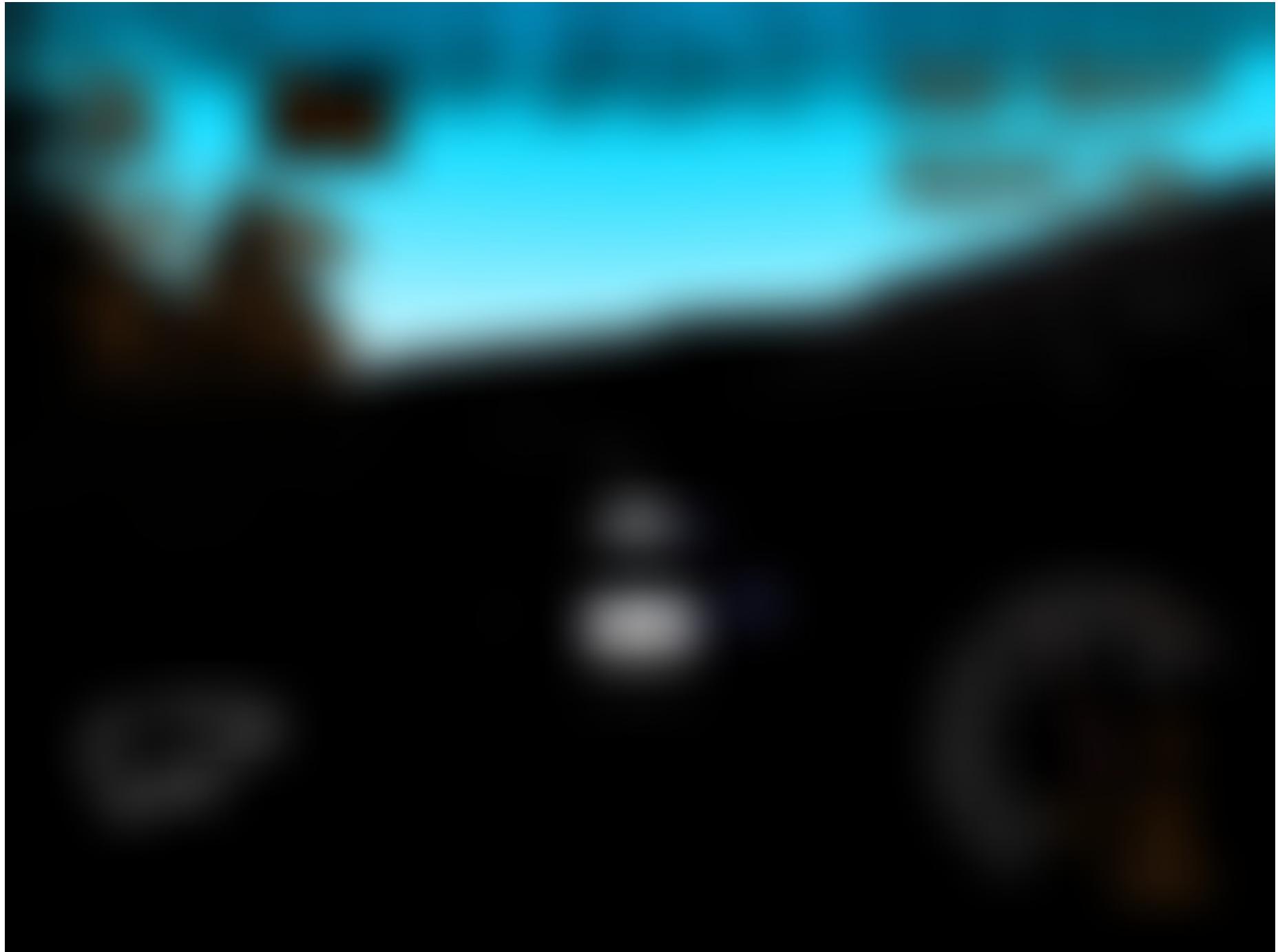


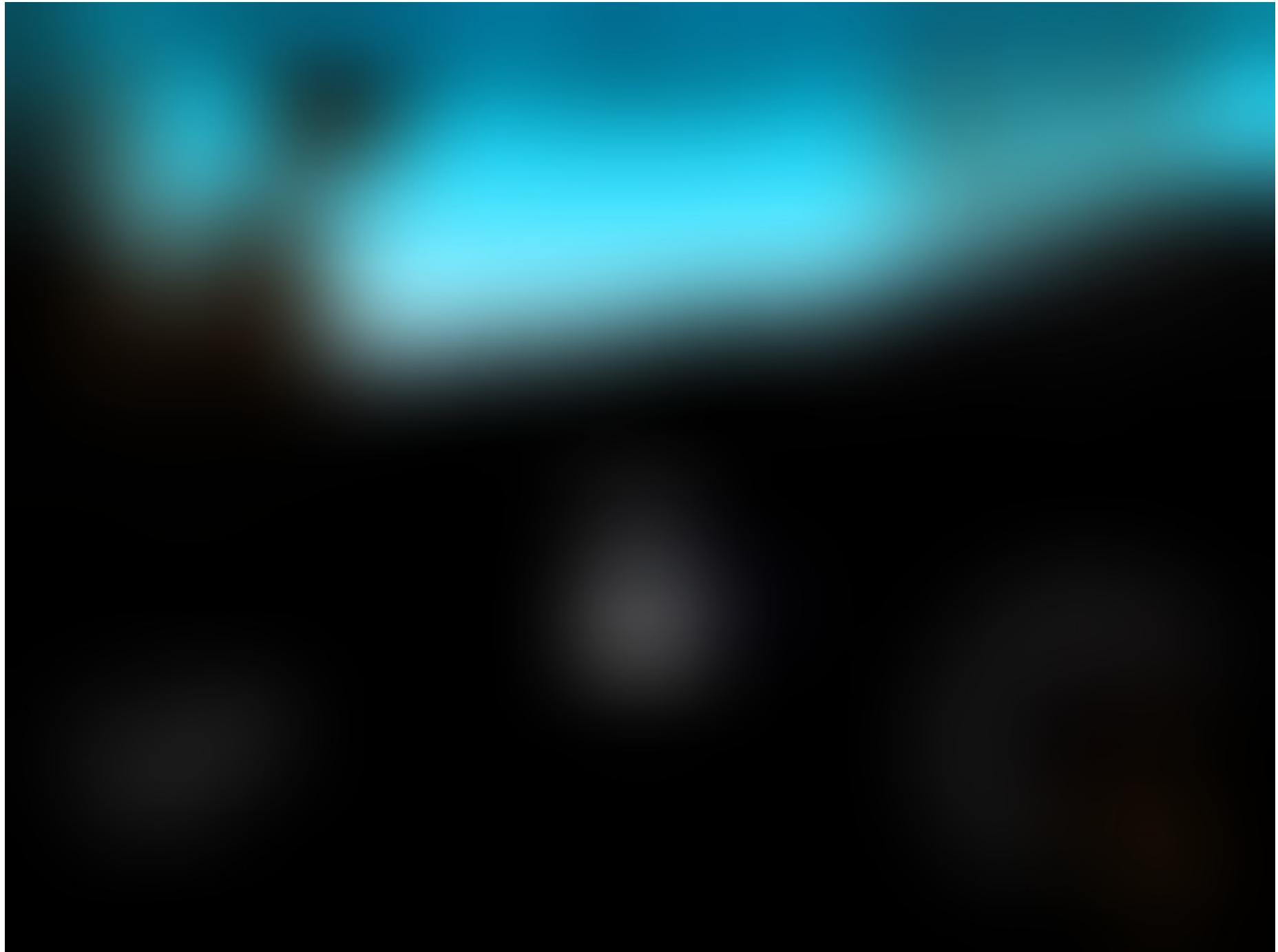
## 2. Borrando a máscara

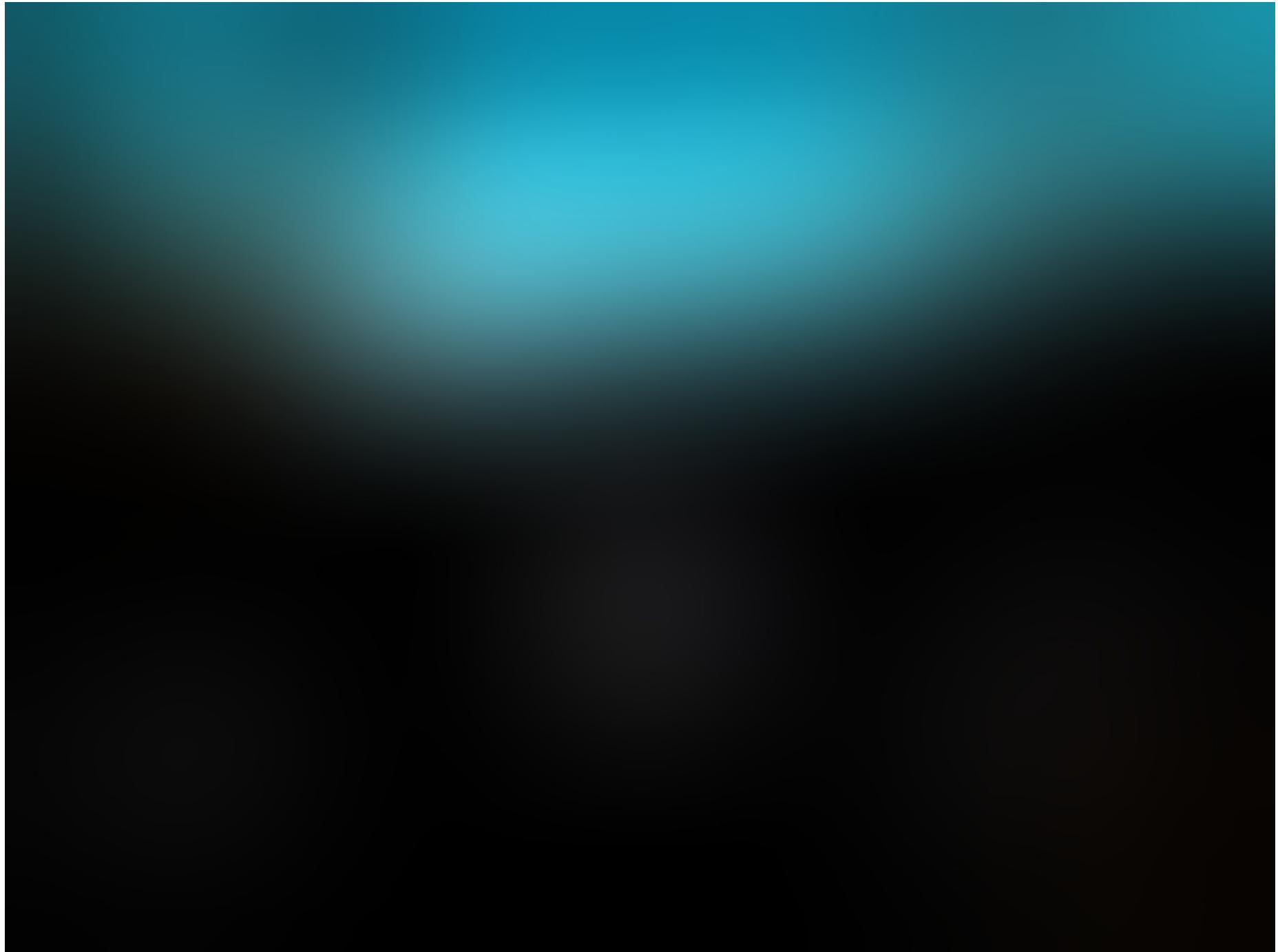
- No segundo passo, a máscara é borrada.
  - O objetivo é obter uma imagem na qual a luz “vaza”.
- A imagem “ideal” normalmente é obtida borrando a imagem original várias vezes, e **somando** as imagens borradadas.
  - Ex: suavização Gaussiana, dobrando o valor de  $\sigma$  a cada imagem.
- Como a soma pode ter um valor grande, você precisa escolher:
  - Truncar o valor em 1?
  - Truncar o valor em algum limite maior que 1?
  - Manter o valor como ficar?
- Nota: normalmente, as imagens são renderizadas com alcance dinâmico maior do que o usado para a apresentação final.
  - *High Dynamic Range – HDR*.
  - O termo é usado em outras circunstâncias, como veremos no futuro.

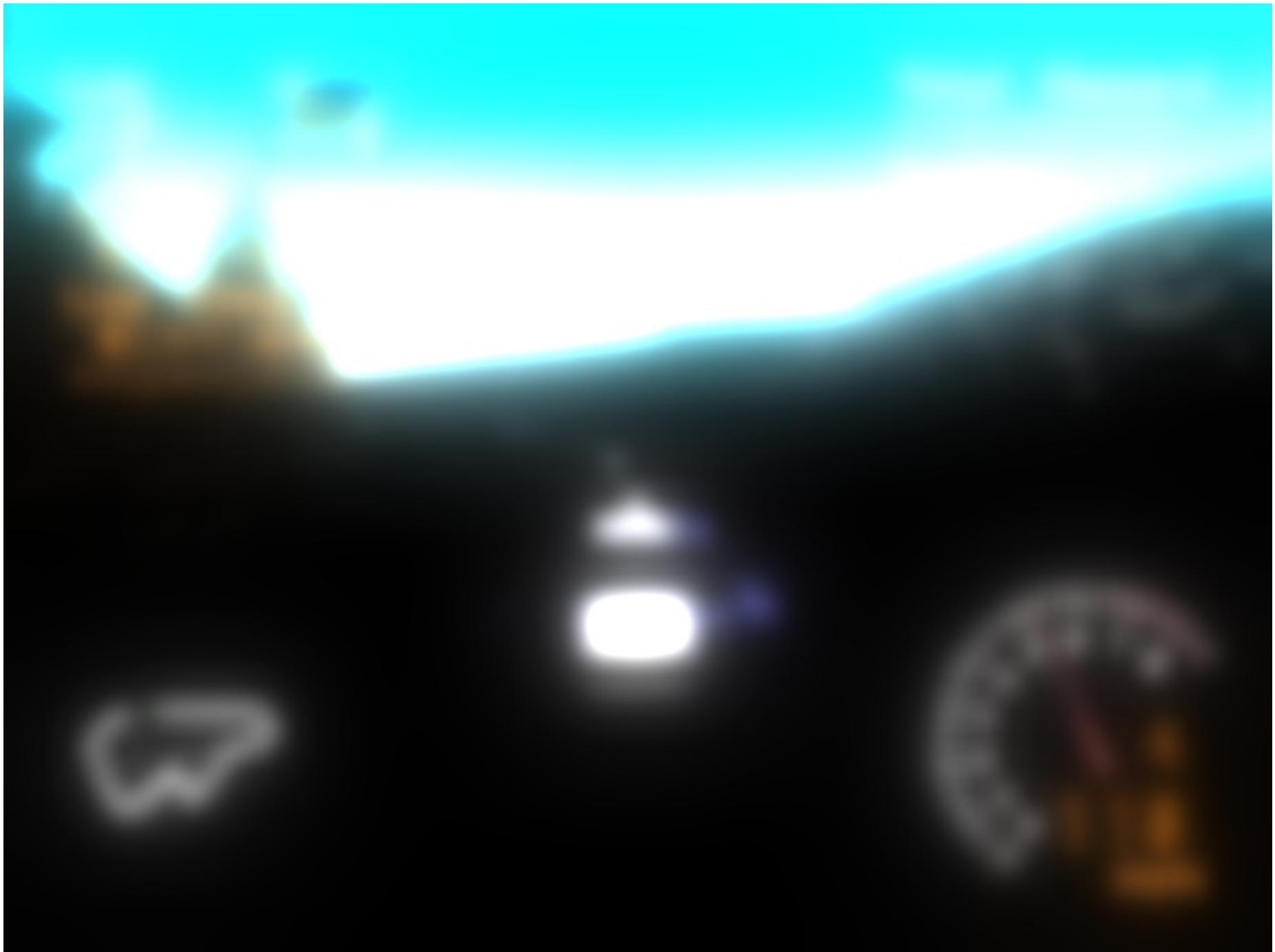






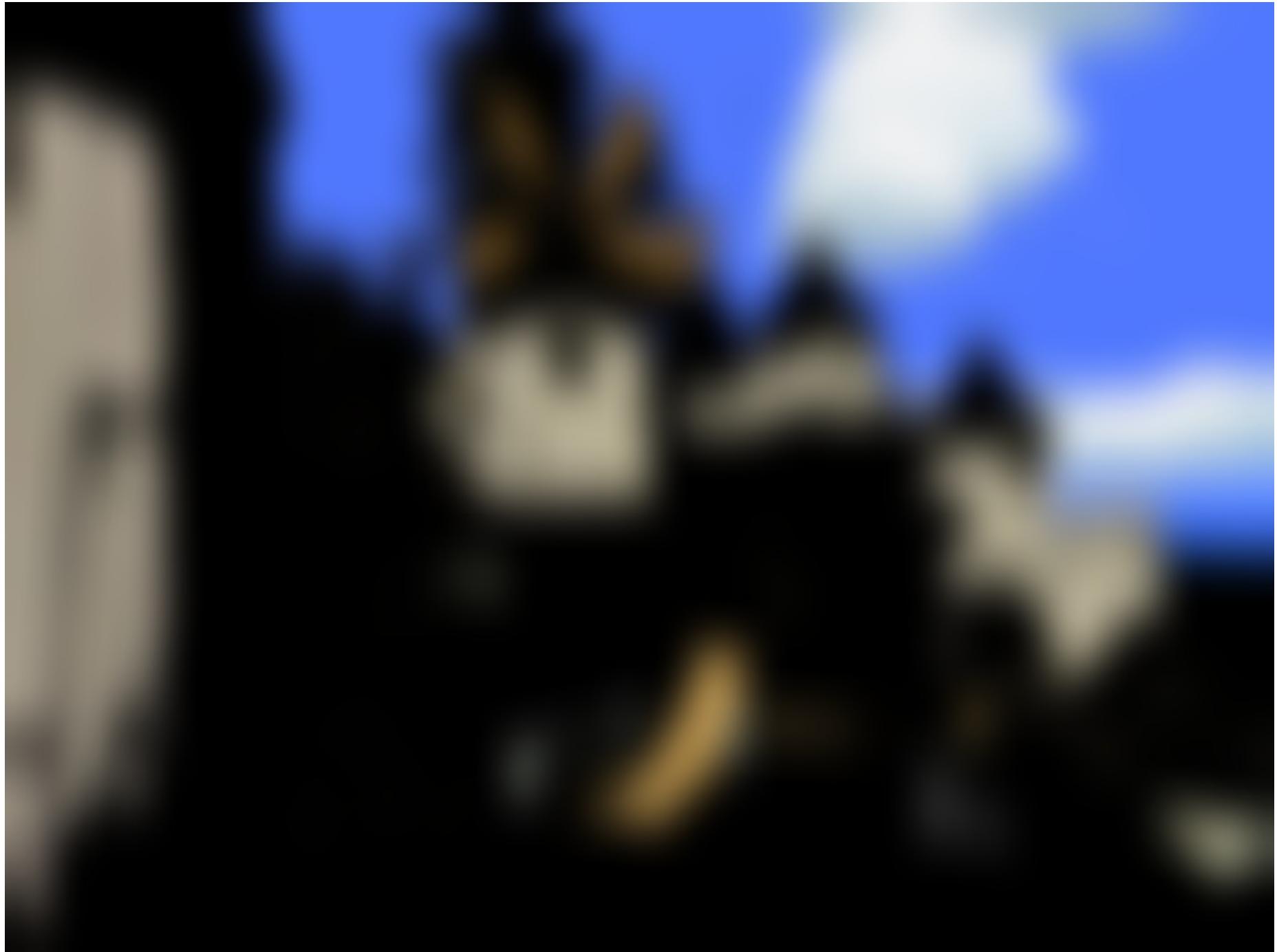


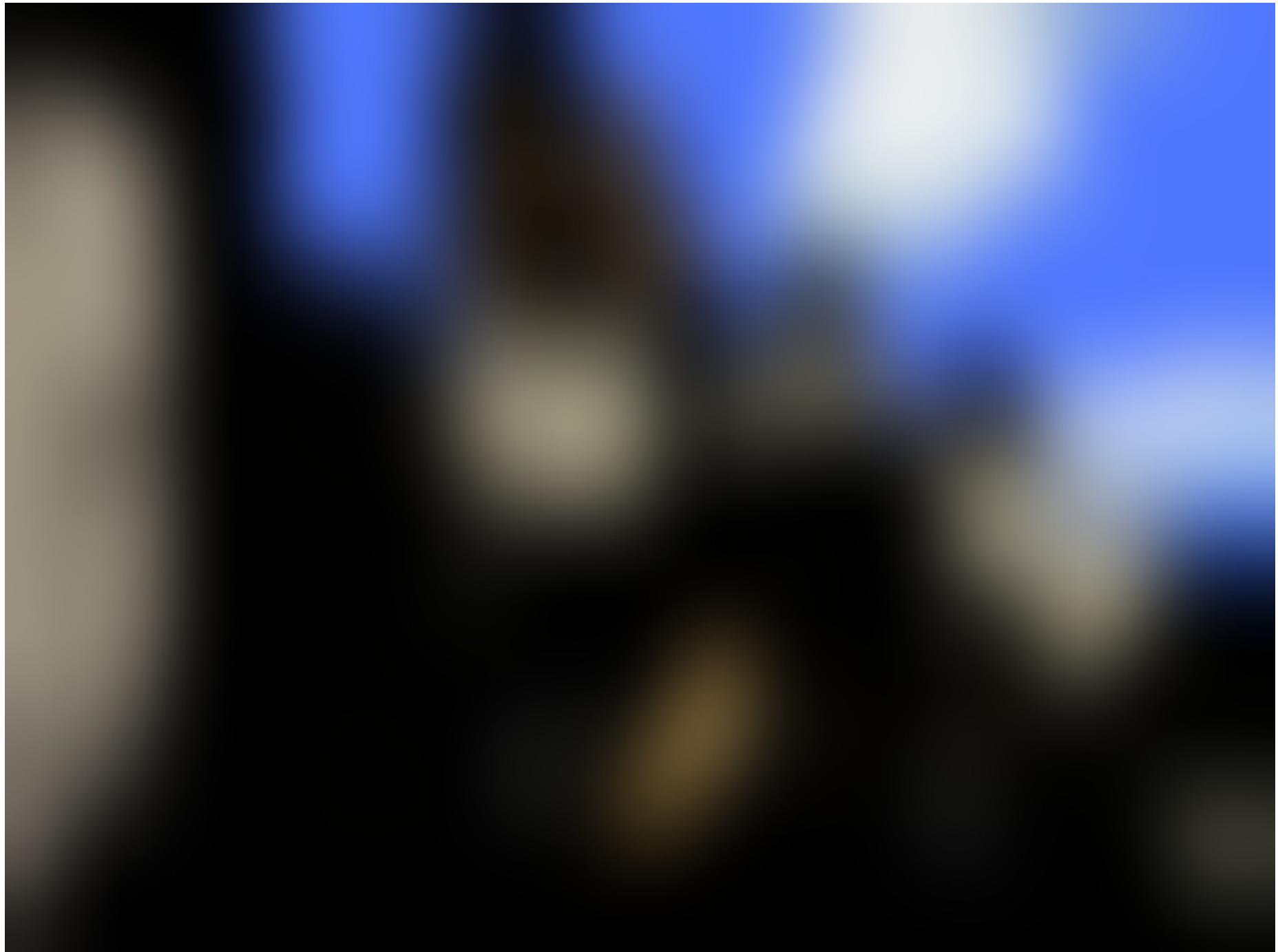


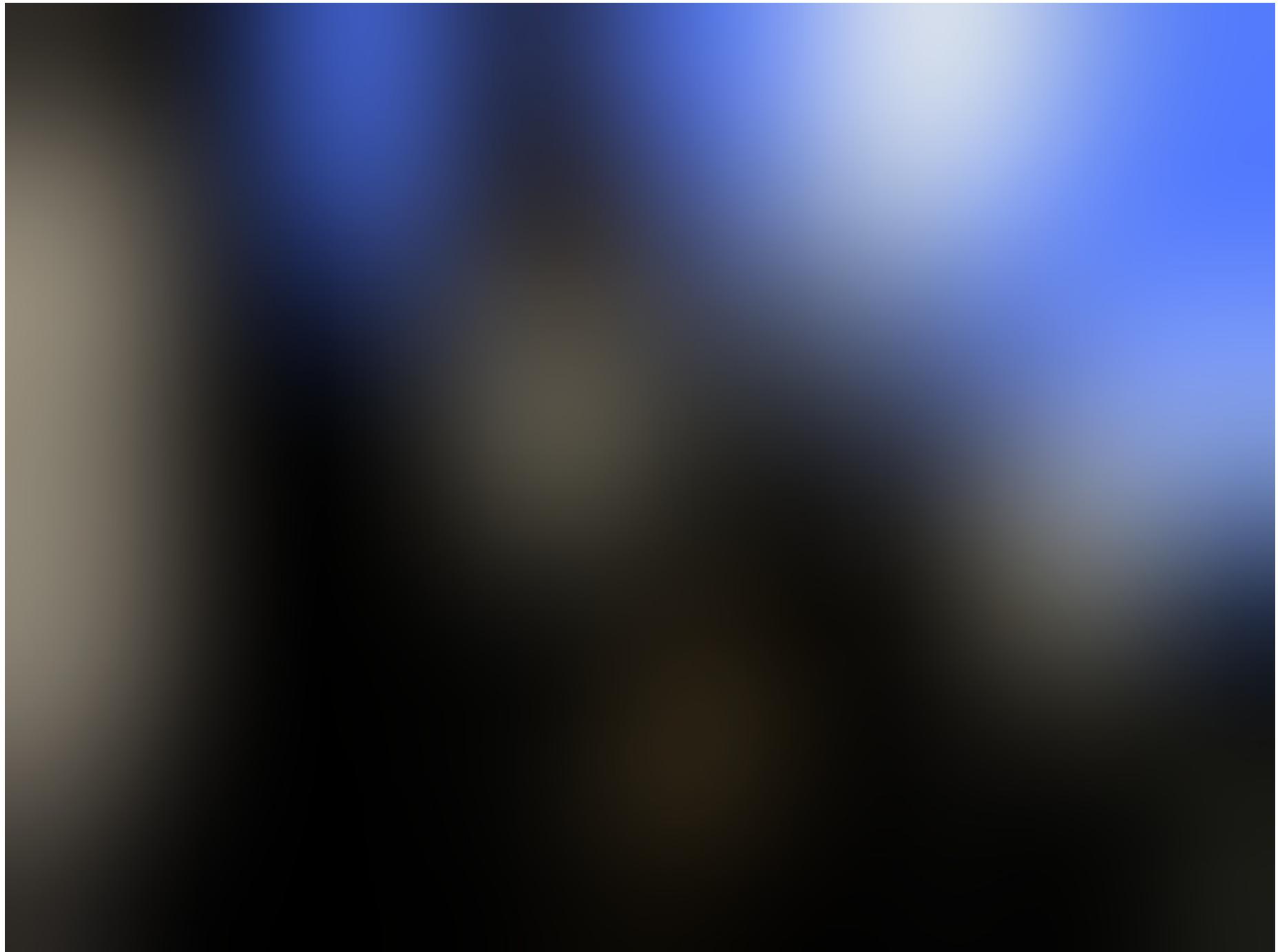


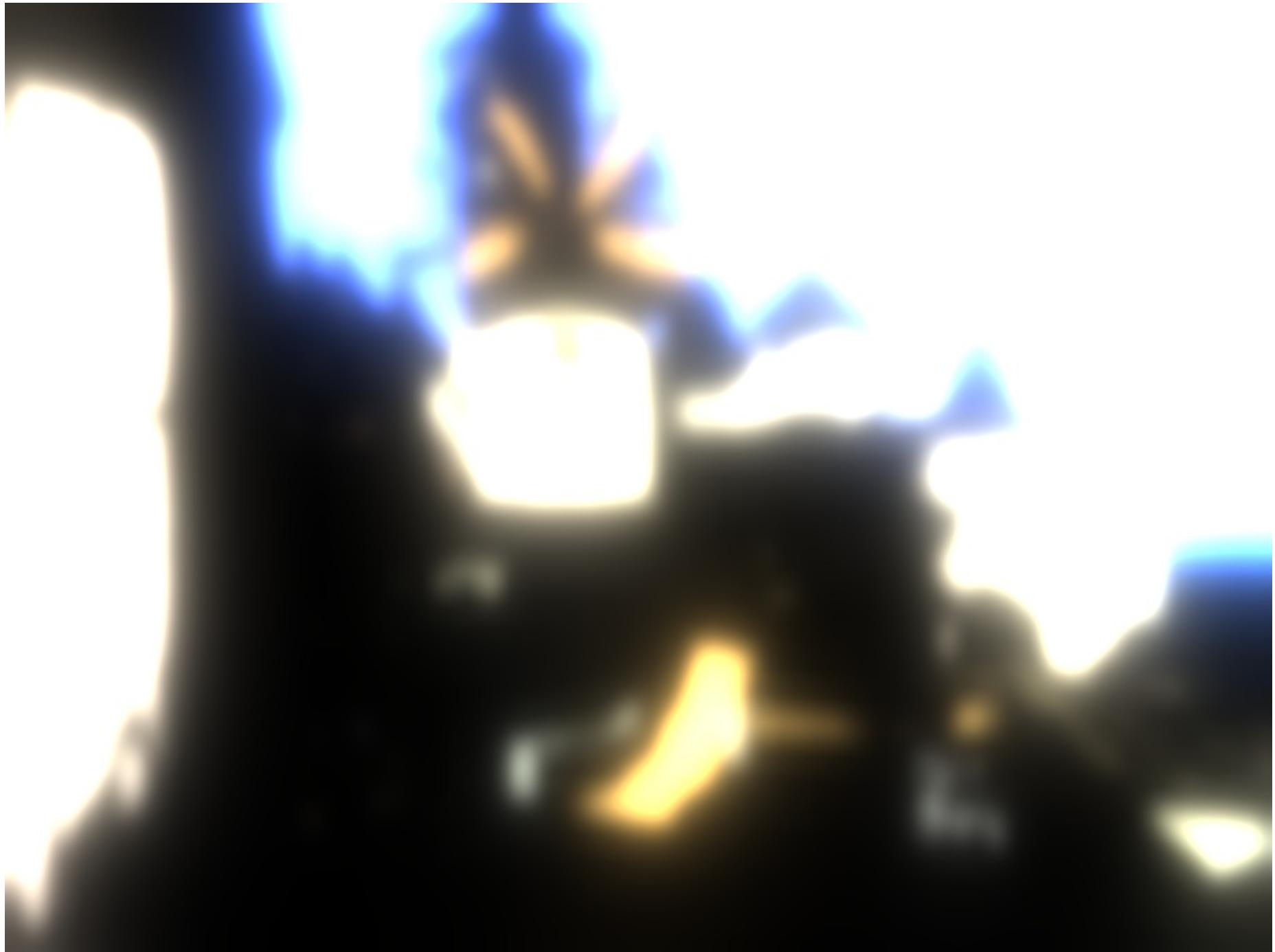












# Otimizando...

- A filtragem Gaussiana pode ser um pouco cara, especialmente para valores grandes de  $\sigma$ .
- Propriedade matemática importante:
  - Seja uma imagem  $I$  com  $N \times M$  pixels, borrada usando suavização Gaussiana, com  $\sigma = x$ .
  - Seja uma imagem  $J$  com  $N/2 \times M/2$  pixels, borrada usando suavização Gaussiana, com  $\sigma = x/2$ .
  - $I$  e  $J$  têm exatamente os mesmos dados
    - Matematicamente é assim, na prática existem arredondamentos, mas o resultado é “próximo o suficiente”.
- Como usar isso para otimizar o procedimento?

# Otimizando...

- A filtragem Gaussiana pode ser um pouco cara, especialmente para valores grandes de  $\sigma$ .
- Propriedade matemática importante:
  - Seja uma imagem  $I$  com  $N \times M$  pixels, borrada usando suavização Gaussiana, com  $\sigma = x$ .
  - Seja uma imagem  $J$  com  $N/2 \times M/2$  pixels, borrada usando suavização Gaussiana, com  $\sigma = x/2$ .
  - $I$  e  $J$  têm exatamente os mesmos dados
    - Matematicamente é assim, na prática existem arredondamentos, mas o resultado é “próximo o suficiente”.
- Como usar isso para otimizar o procedimento?
  - R: em vez de dobrar o valor de  $\sigma$  a cada imagem, usamos o mesmo valor de  $\sigma$ , mas para uma imagem com metade da altura e largura!
  - A imagem depois é reescalada para o tamanho original.
    - Não vamos usar esta otimização hoje...

# Otimizando (mais)

- Em alguns casos, mesmo usar um  $\sigma$  menor não é o suficiente.
- Nestes casos, usamos uma aproximação...
- É mais fácil de entender com um exemplo.

# Suavização Gaussiana ( $\sigma=10$ )



# Filtro da média (5 vezes, 15x15)



# Filtro da média (3 vezes, 19x19)



# Otimizando (mais)

- Moral da história: você pode aproximar uma suavização Gaussiana aplicando o filtro da média sucessivamente!
  - A ideia é “borrar a imagem borrada” várias vezes para simular uma aplicação do filtro Gaussiano.
- Existe um artigo que mostra o cálculo exato para fazer esta aproximação, mas para nós aqui, o importante é o conceito!

# 3. Juntando tudo

- Podemos agora “colar” a máscara e a imagem.
  - De forma direta, basta somar as duas imagens, mas você precisará encontrar formas de evitar que o resultado fique muito “estourado”.
  - A soma pode ser ponderada por fatores de correção.
    - $g(x,y) = \alpha \cdot f(x,y) + \beta \cdot \text{máscara}(x,y)$
  - A imagem resultante pode também sofrer ajustes de brilho, saturação ou contraste.





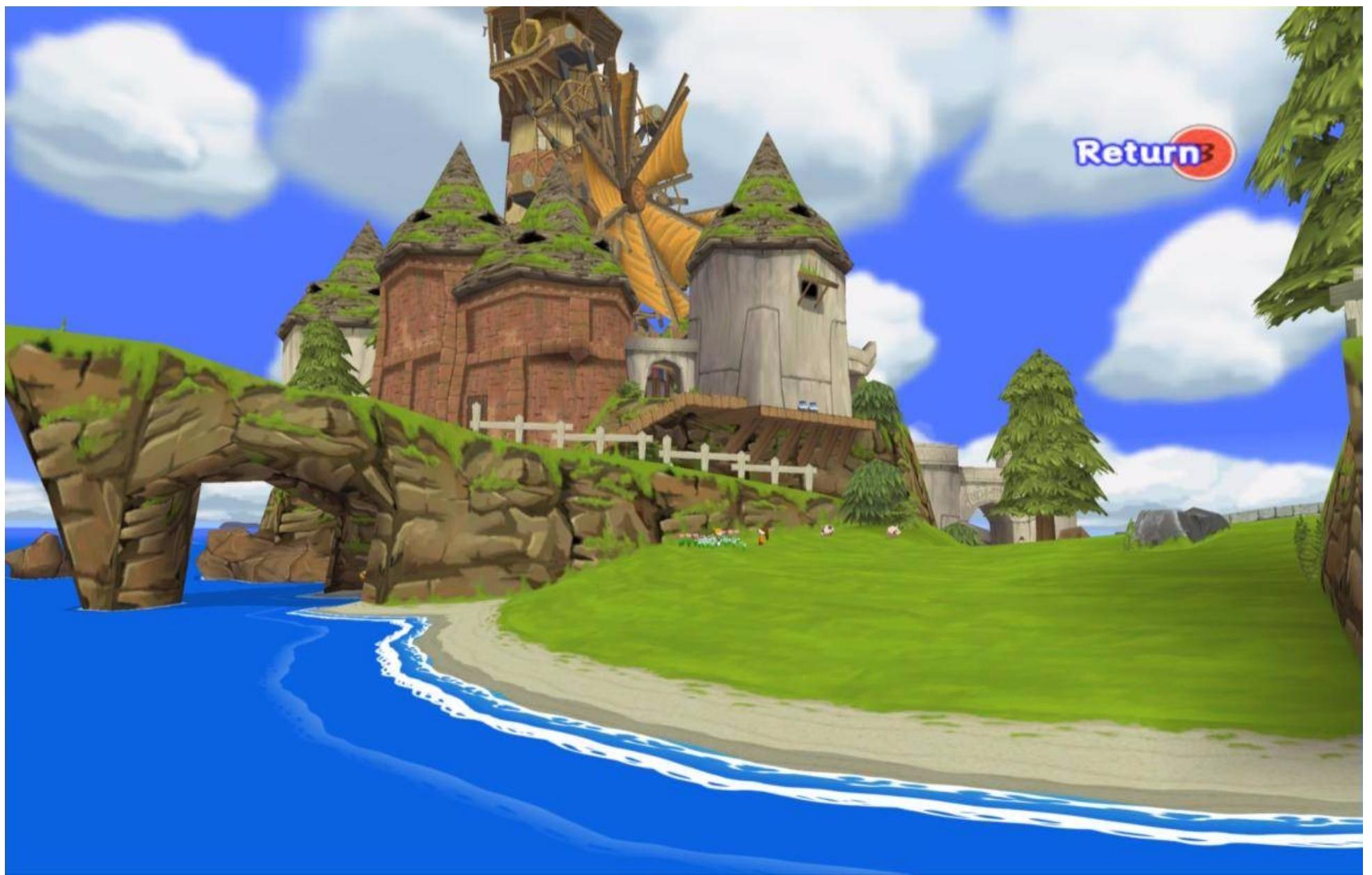














# Trabalho 3

- Implemente o efeito *bloom* em 2 versões, com filtragem Gaussiana e box blur.
  - Para o *bright-pass*, não faça binarização independente de 3 canais!
  - Observe que os valores de  $\sigma$  aumentam bastante entre os filtros.
  - Lembre-se que a substituição não é de uma aplicação do filtro Gaussiano por uma do filtro da média; cada aplicação do filtro Gaussiano é aproximada com várias aplicações sucessivas do filtro da média!

# Algumas funções úteis

- Normalização:

`normaliza`

`normalizaSemExtremos8bpp`

`normLocalSimples`

- Conversão de cores:

`RGBParaCinza`

`cinzaParaRGB`

`RGBParaHSL`

`HSLParaRGB`

- Ajuste de cores:

`inverte`

`ajustaBrilhoEContraste`

`ajustaGama`

`ajustaHSL`

- Filtros:

`blur`

`filtroGaussiano`

`unsharpMasking`

`filtroMediana8bpp`

`maxLocal`

`minLocal`

- Manipulação básica:

`soma`

`criaImagem`

`destroiImagem`

`abreImagem`

`salvaImagem`

`clonaImagem`

`copiaConteudo`

Nota: o pacote tem outras funções além destas.