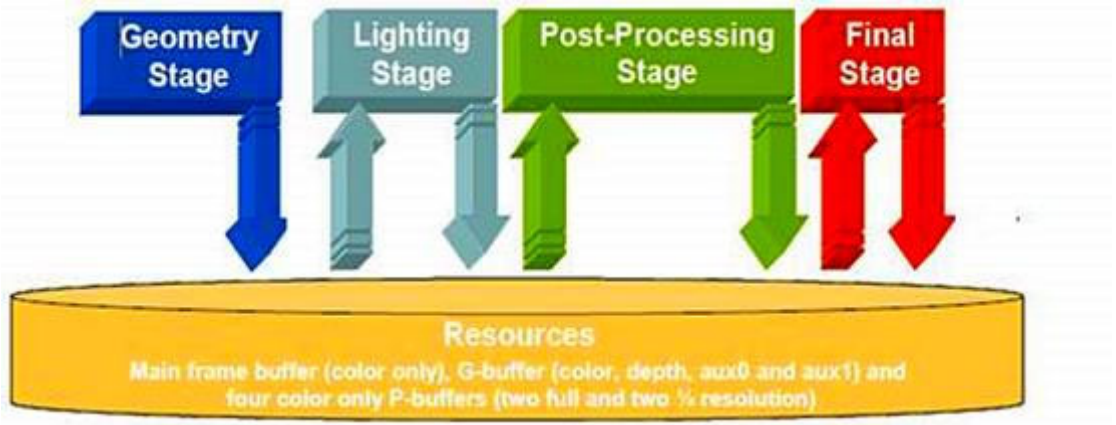
Dieter Schmalstieg

Image-Space Special Effects

Based on material by Daniel Wagner, Michael Kenzel, Thomas Neff

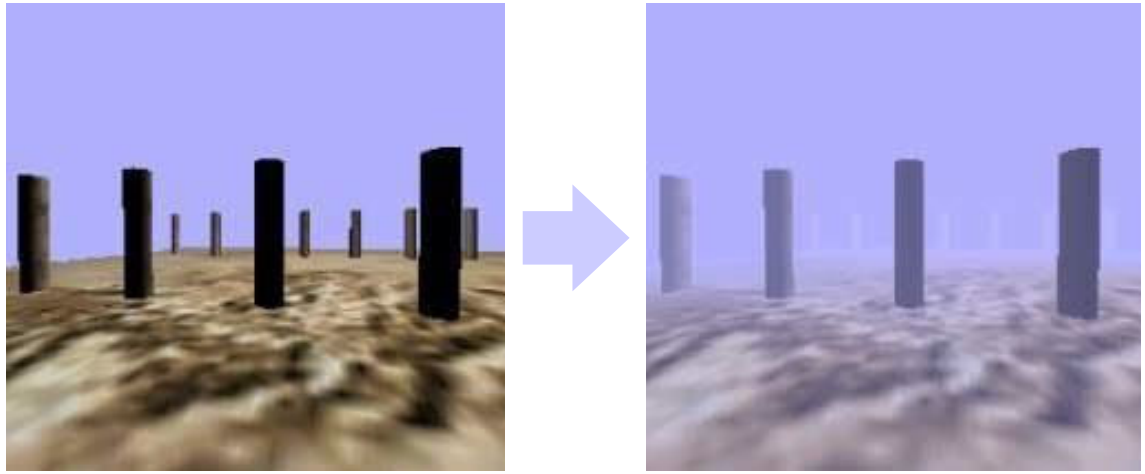
Motivation



- Utilize deferred rendering framework to add special effects *in image space* after rendering
- Independent of geometric scene complexity
- Often uses image processing techniques
- Can operate on G-buffer (not just color buffer)
- Can composit *many* G-buffers (e.g., per object)

Postprocessing

- Postprocessing Idea
  - Run a fragment shader for every pixel
  - Apply some modification to pixel
- Example: fog
  - Saturation decreases with distance to camera

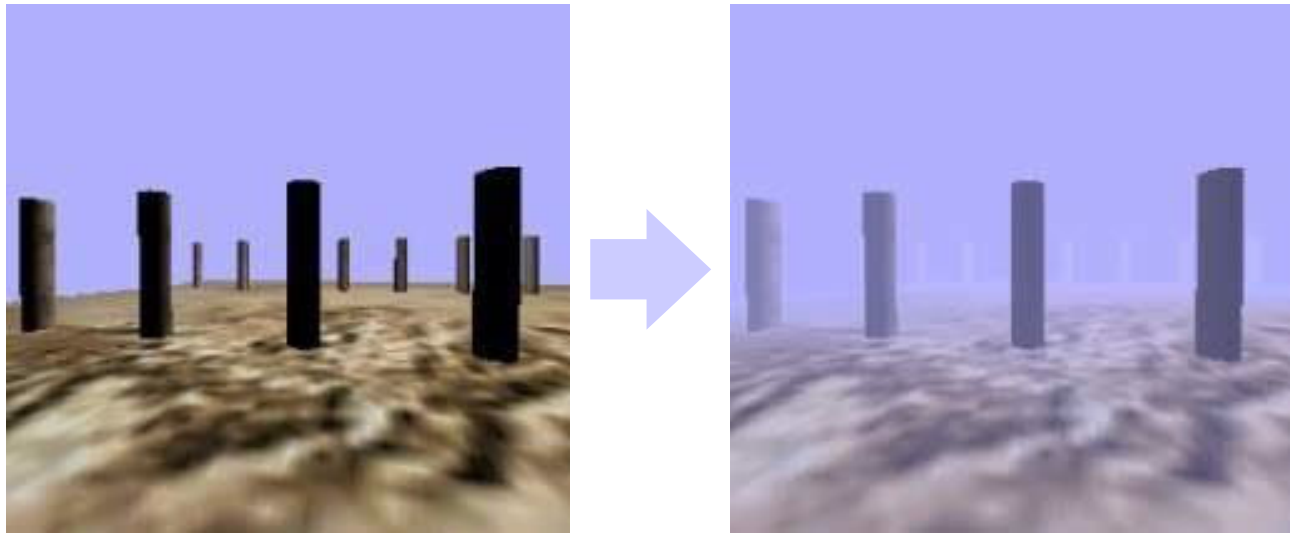


Distance Fog

- Blend surface color with fog color

$$\mathbf{c} = f\,\mathbf{c}_s + (1 - f)\mathbf{c}_f$$

$\mathbf{c}_s$	surface color
$\mathbf{c}_f$	fog color
f	fog factor



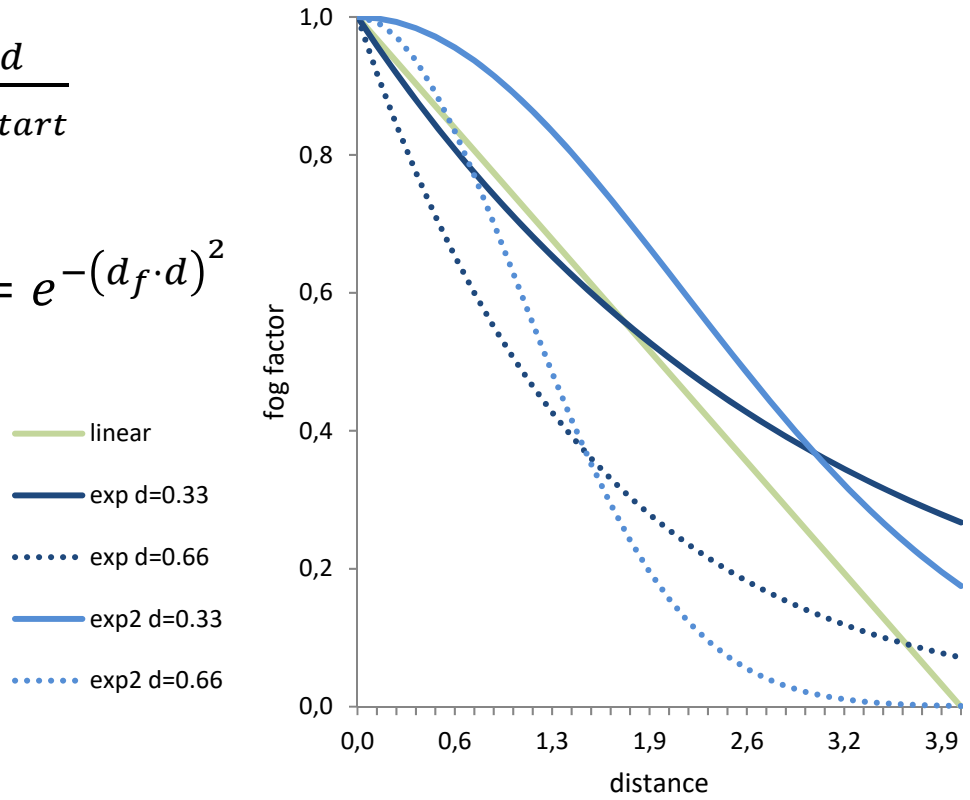
Fog Function

- Blend surface color with fog color: $\mathbf{c} = f\mathbf{c}_s + (1-f)\mathbf{c}_f$

  $\mathbf{c}_s$ surface color, $\mathbf{c}_f$ fog color, f fog factor

- Linear fog: $f = \frac{d_{end}-d}{d_{end}-d_{start}}$
- Exponential fog: $f = e^{-d_f \cdot d}$
- Squared exponential fog: $f = e^{-(d_f \cdot d)^2}$

d	fragment distance
d_{start}	fog start
d_{end}	fog end
d_f	fog density



Distance Fog – Implementation

```
1  #version 330
2  #include <framework/utils/GLSL/camera>
3
4  uniform vec3 c_d;
5  uniform vec3 c_f;
   uniform float d_f;
6
7  in vec3 p; // surface position
8  in vec3 c_s; // surface
9  layout(location = 0) out vec4 color;
10
11 void main()
12 {
13     vec3 v = camera.position - p;
14     float d = length(v);
15
       float f = exp(-d_f * d);
16
       color = vec4(f * c_s + (1.0f - f) * c_f, 1.0f);
17 }
18
```

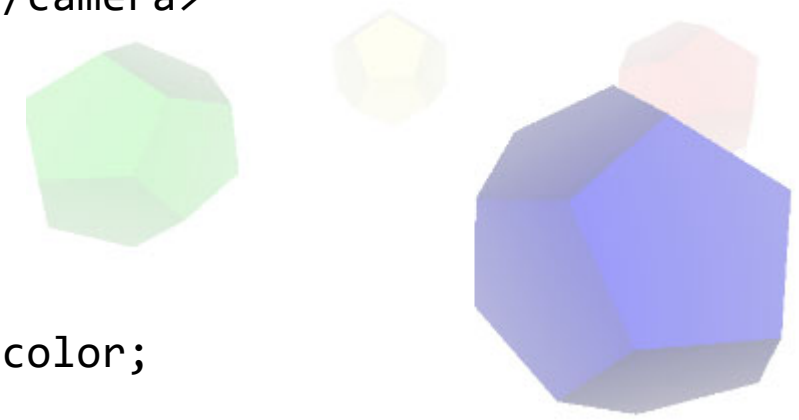
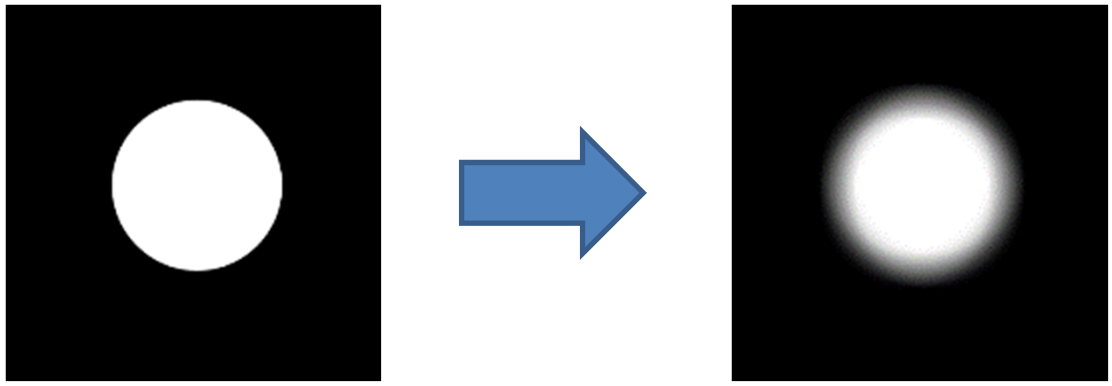


Image Processing

- Image Processing Idea
  - Compute some derivative function per pixel
- Example: Gaussian blur



Gaussian Filter

- Many effects based on Gaussian filter
- 5x5 Gaussian filter requires 25 texture lookups
  - Too slow and too expensive

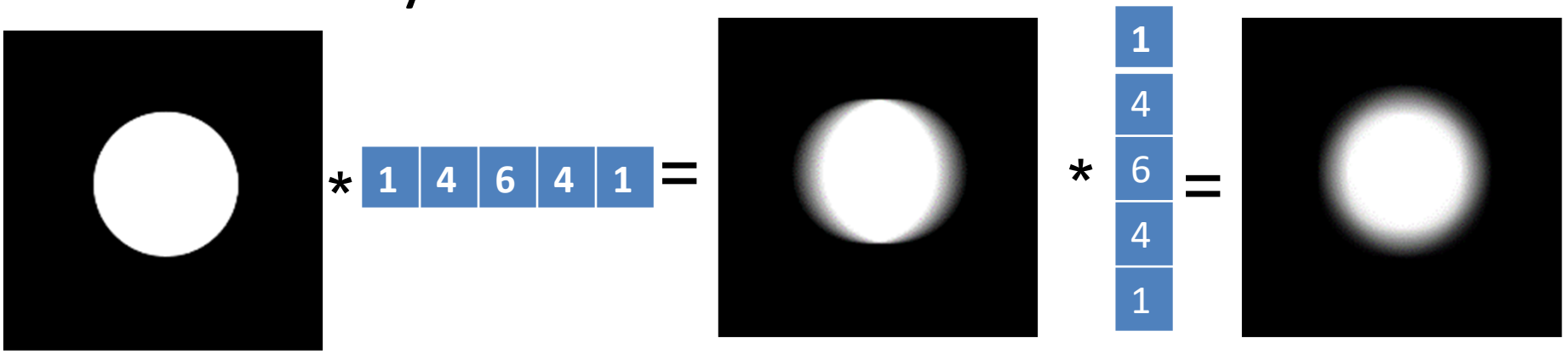
1	4	6	4	1
4	16	26	16	4
6	26	41	26	6
4	16	26	16	4
1	4	6	4	1

* 1/25

- But: Gaussian filter is separable!

Separable Gaussian Filter

- Separate 5x5 filter into 2 passes
- Perform 5x1 filter in *u*
- Followed by 1x5 filter in *v*



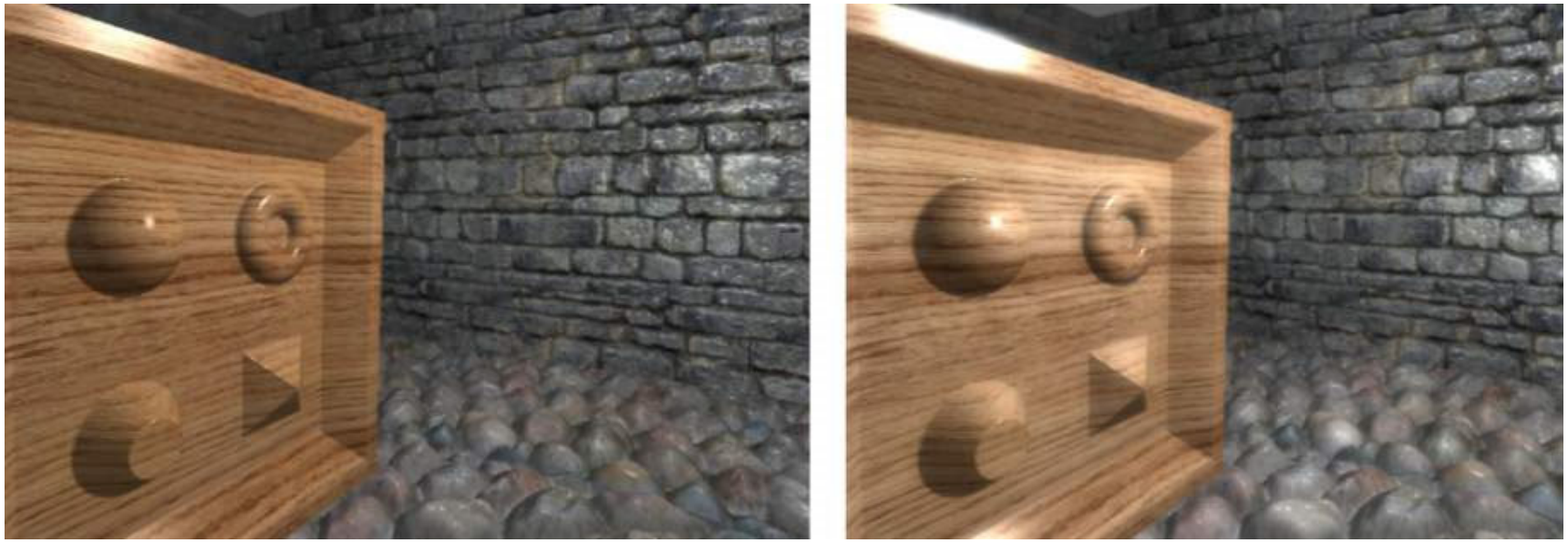
- Lookups can be performed via linear filtering
  - 5x1 filter with 3 lookups

Separable Box Blur Shader

```
#version 330
uniform sampler2D color_buffer;
uniform float texel_size;
const int num_samples = 32;
#ifdef BLUR_Y
const vec2 dir = vec2(0.0f, 1.0f);
#else
const vec2 dir = vec2(1.0f, 0.0f);
#endif
in vec2 t; // incoming location
layout(location = 0) out vec4 color;
void main()
{
    vec4 c = vec4(0.0f, 0.0f, 0.0f, 0.0f);
    for (int i = 0; i < num_samples; ++i)
    {
        vec2 d = dir * texel_size * ((i - num_samples / 2) + 0.5f);
        c += texture(color_buffer, t + d) * (1.0f / num_samples);
    }
    color = c;
}
```

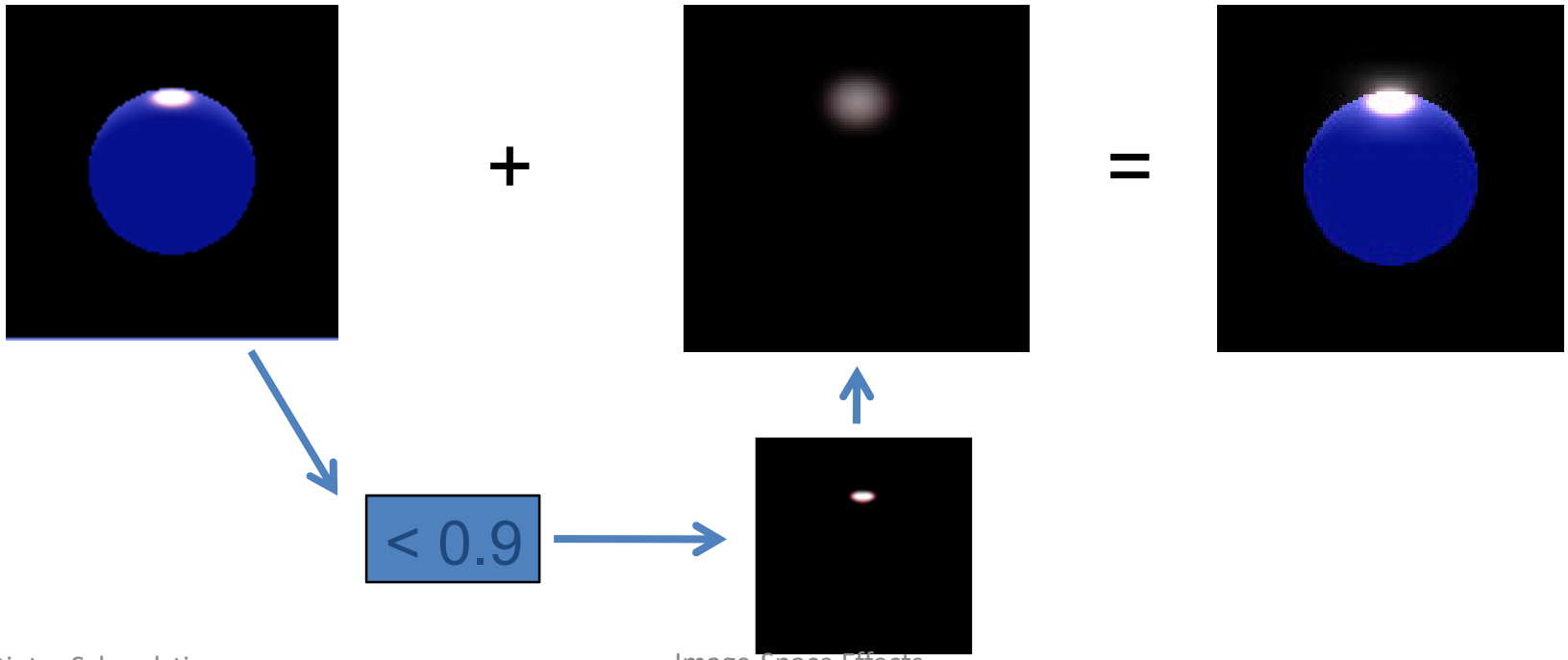
Bloom

- Bright areas seem to “spill” light to vicinity
- Apply blur to simulate this effect



Bloom

- Before Gaussian filtering
  - Modify rendered texture intensities
  - Clamp or glowing-object-only pass
  - Exponential weight
- Add filtered image to original image



Bloom Discussion

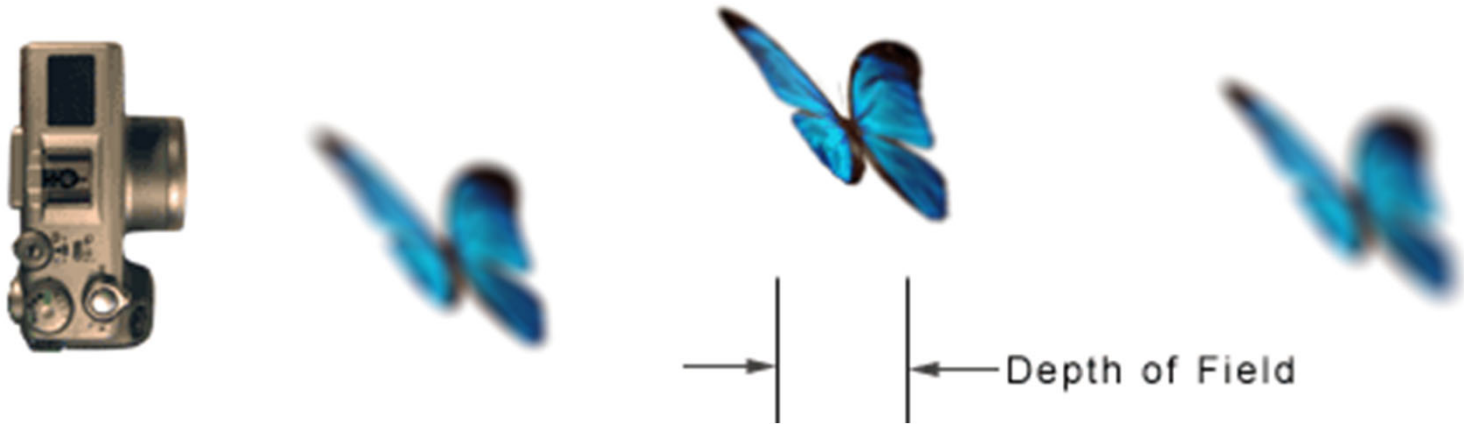
- Usually applied to downsampled render textures
  - 2x or 4x downsampled
  - Effectively increases kernel size
  - But: sharp highlights are lost
  - Combination of differently downsampled and filtered render textures possible
  - Allows good control of bloom effect
- Star effect 
  - Filter in *u* and *v* separately
  - Add *u* and *v* texture separately

Bloom Remarks

- Disguises aliasing artifacts
- Works best for shiny materials and sun/sky
  - Only render sun and sky to blur pass
  - Only render specular term to blur pass
- A little bit overused these days
  - Use sparsely for most effect
- Can smudge out a scene too much
  - Contrast and sharp features are lost (fairytale look)

Depth of Field

- DoF simulates camera property
  - Lens can only focus on one depth level
- Objects around that depth level appear sharp
- Rest is blurred, depending on distance to focal plane



Depth of Field Example

Depth of Field Application

- Guide the user‘s attention towards something

Postprocessing Depth of Field

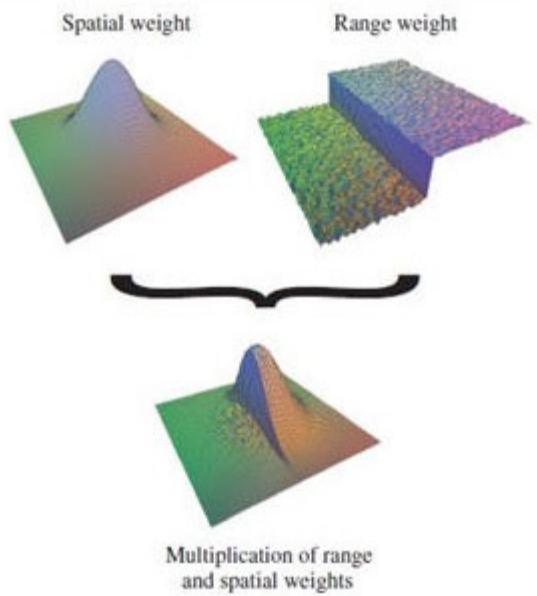
- For each pixel in fragment shader
  - Compute the circle of confusion (CoC) based on depth buffer
  - Blur image using convolution
  - Window size depends on the CoC

- Problem
  - Sharp foreground objects leak on blurry background
- Solution
  - Compare depth values, discard closer pixels
  - We need *depth-aware filtering*

Bilateral Filter

- Gaussian filter (or bilinear filter)
  - Only spatial weight
  - Always blurry

- Bilateral filter
  - Consider also colors (range weight)
  - Similar colors weighted stronger
  - Preserves color discontinuities
  - Color edges not blurred

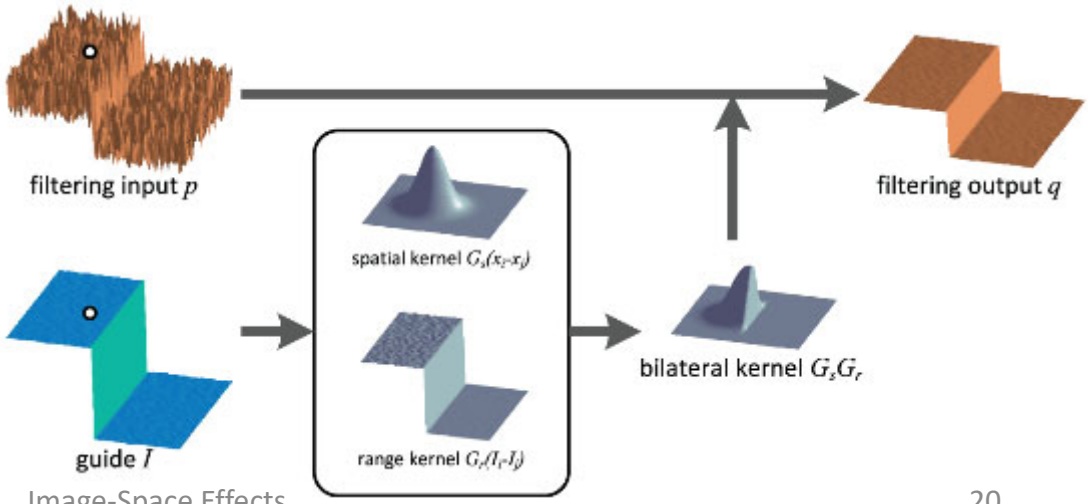


Joint Bilateral Filter

- Bilateral filter with range weights from secondary source channel
- For instance, depth buffer
- Joint bilateral filter avoids bleeding in depth of field

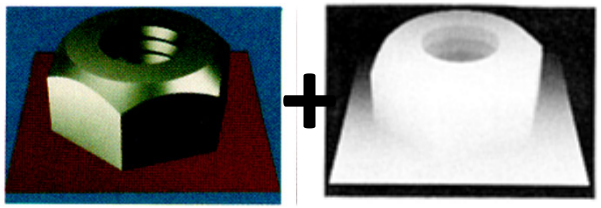
$$q_i = \sum_{j \in N(i)} W_{ij}(I) p_j$$

bilateral filter: $I=p$

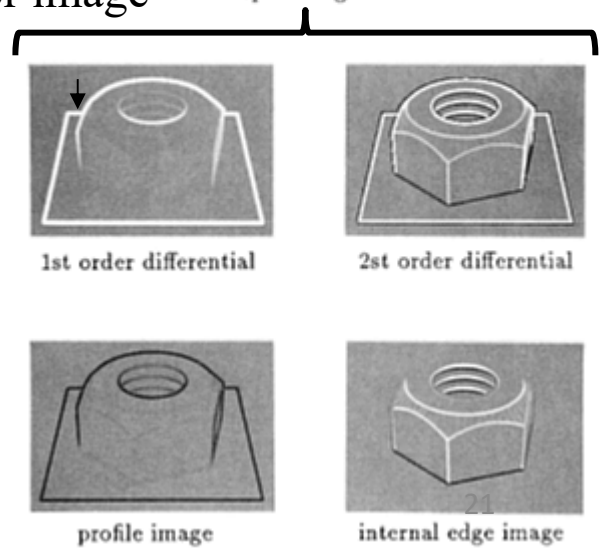


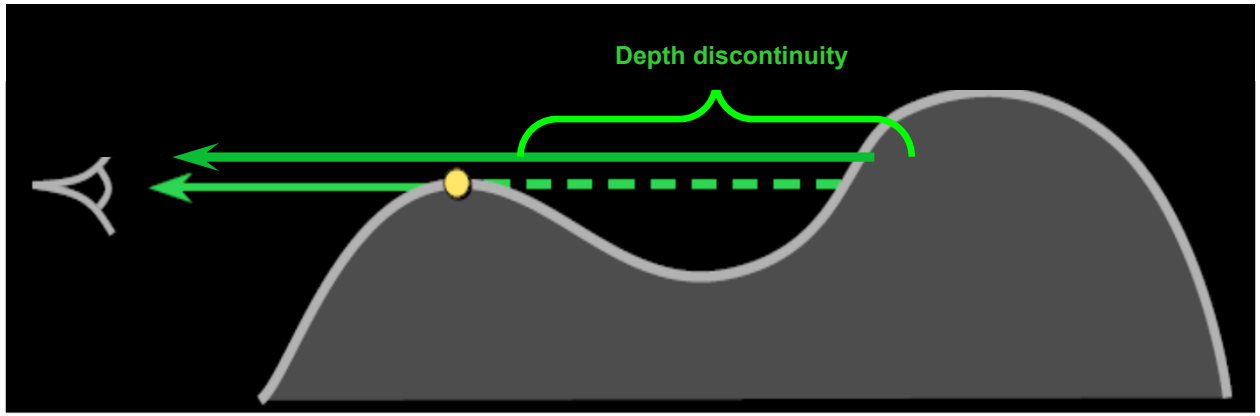
Edge Detection in Image Space

- Run filter on depth buffer
- Profile: 1st order differential
  - E.g., Sobel operator
- Internal: 2nd order differential
  - E.g., Laplace operator



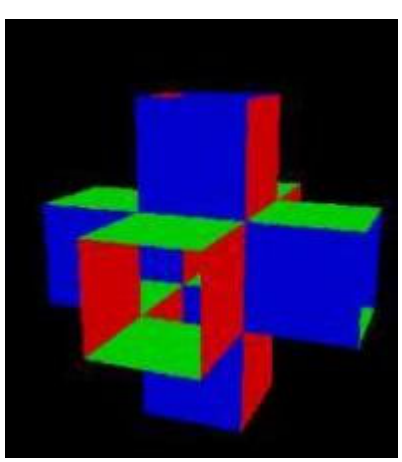
color image depth image

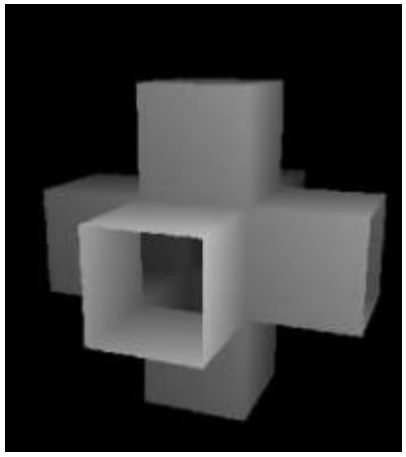


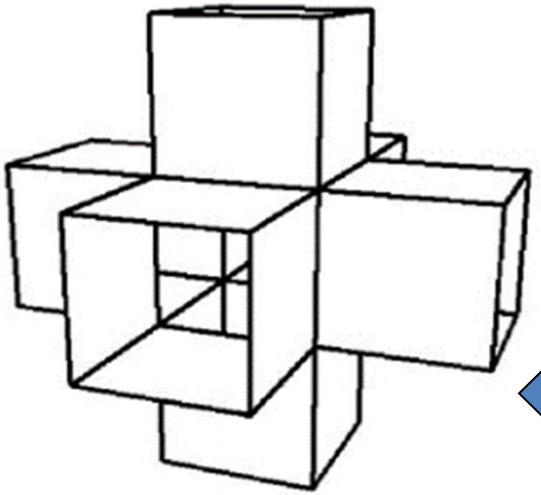


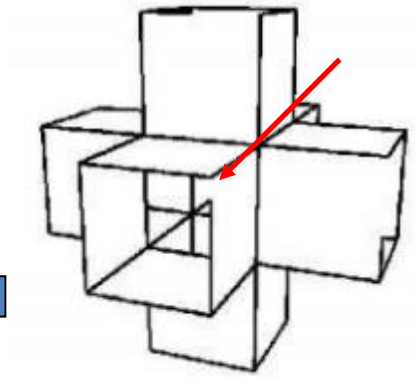
Edge Detection on G-Buffer

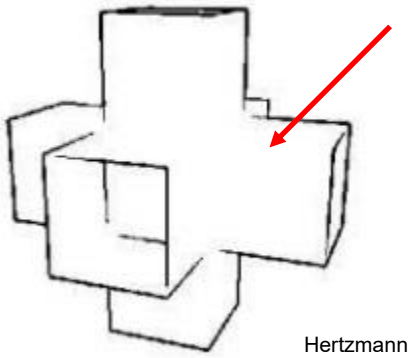
- Make use of all available data
- Detect edges on depth buffer AND normal buffer







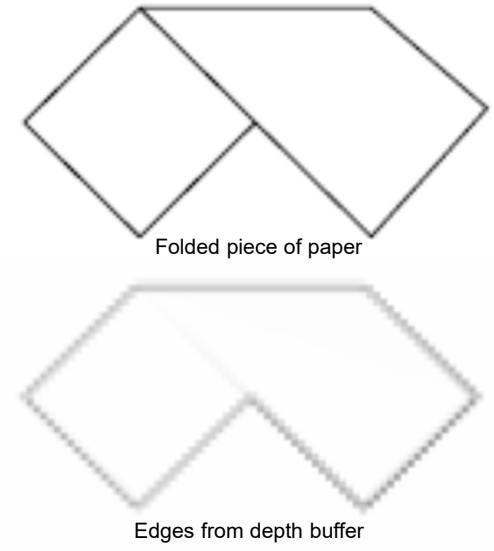


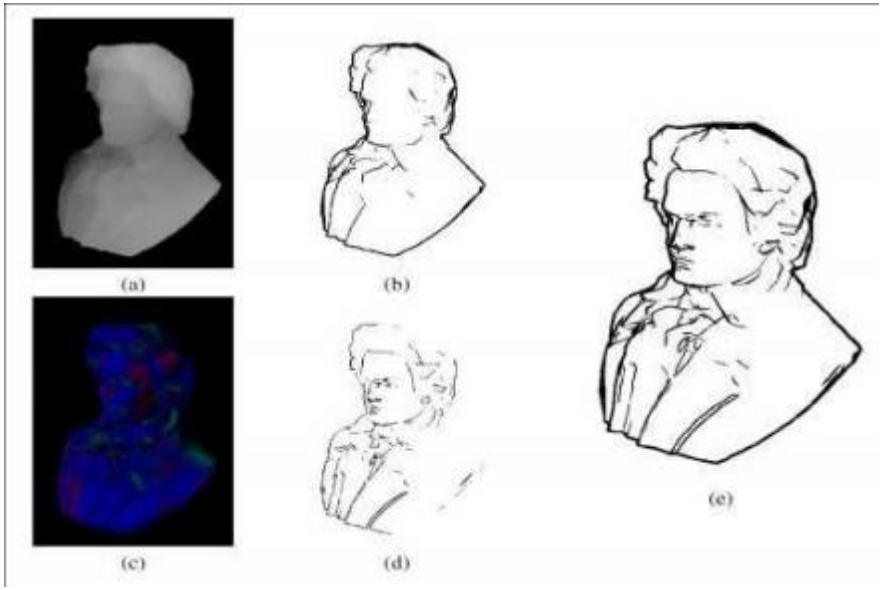


Hertzmann 99

Analysis

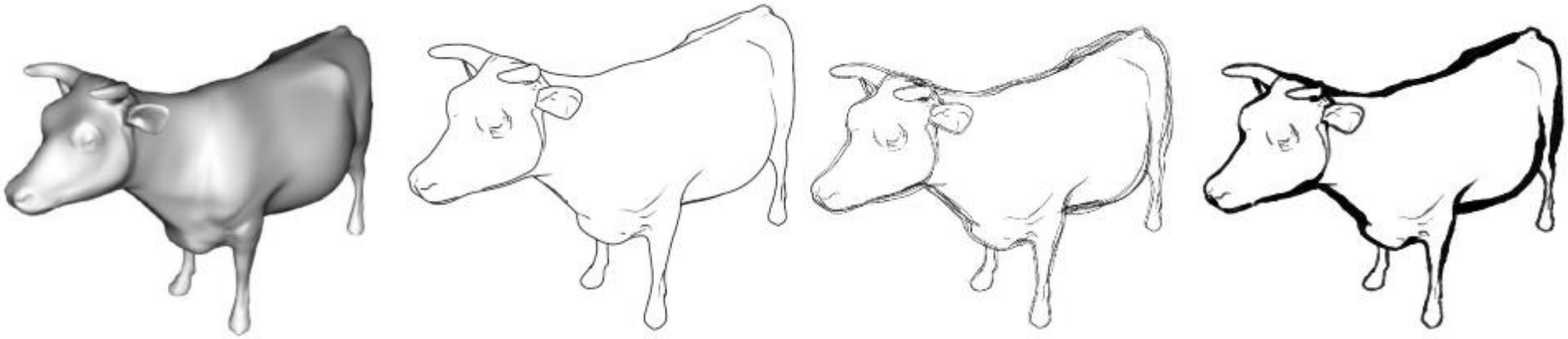
- Pro: Independent of polygon count
- Con: Dependent on image space resolution
  - Different results if zoomed in/out
- Con: Depth fighting can lead to noisy results





Non-Photorealistic Rendering

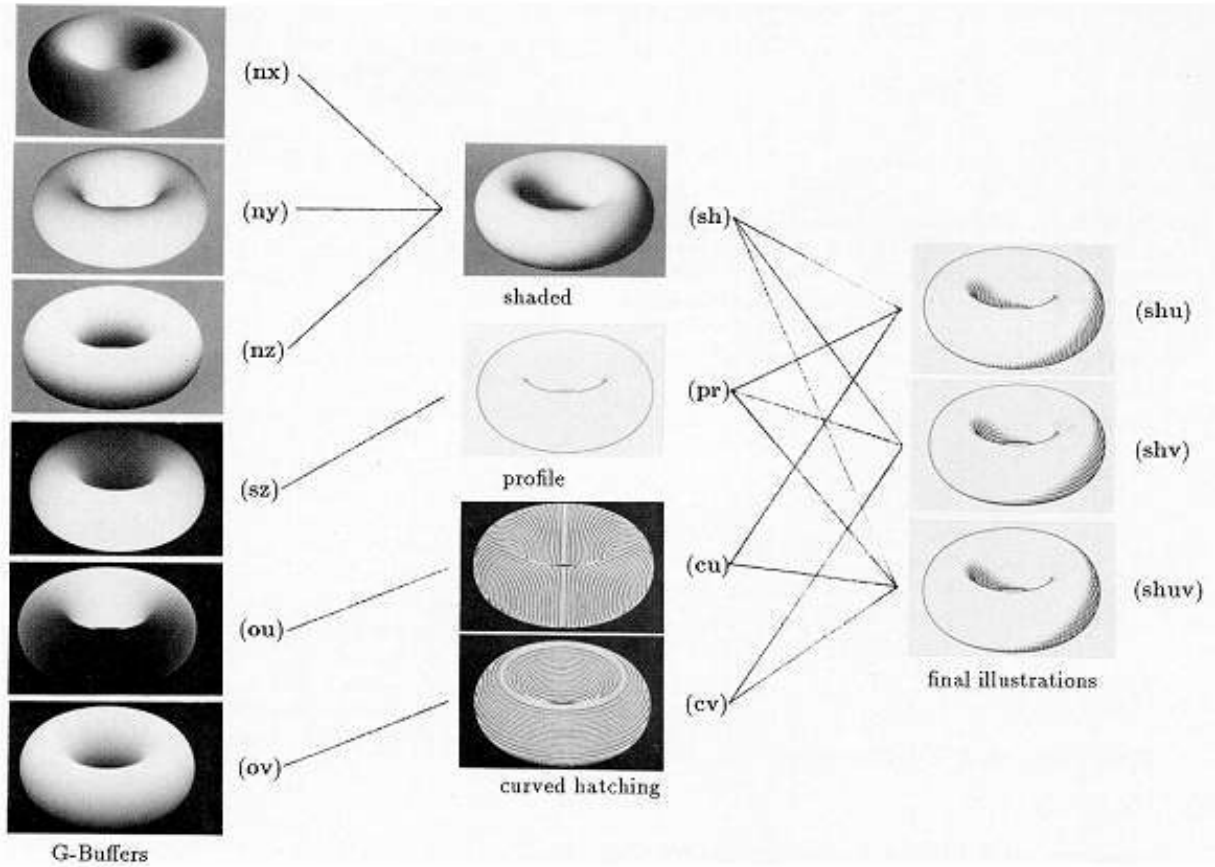
NPR often emphasizes object edges, silhouettes



G-Buffer for Illustrations

- Hatching: create texture from brush strokes which follow the surface coodinates (*u*,*v*) of the object

- *nx*: normal vector z
- *ny*: normal vector y
- *nz*: normal vector z
- *sz*: screen coordinate
- *ou*: coordinate *u* (on curved surface)
- *ov*: coordinate v (on curved surface)



Upsampling

- Naïve (bilinear) upsampling suppresses high frequencies
- Joint bilateral upsampling can help here as well

Original

Nearest neighbor

Bilinear filtering

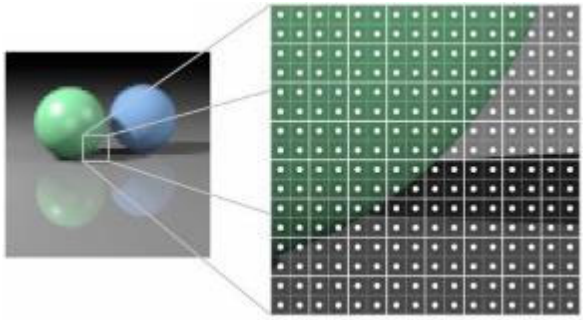
Joint Bilteral Upsampling on PS4

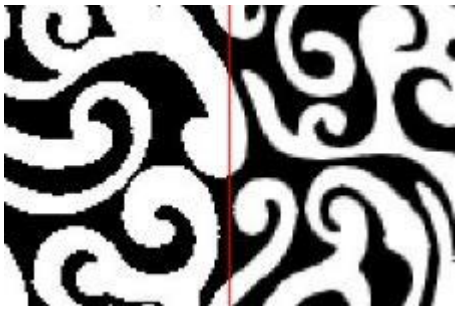
- PS4: 1080p resolution, PS4 Pro: 4K resolution
  - PS4 Pro generates 4x more pixels with only 2x more GPU cores
- Upsampling must work only in driver
  - Cannot assume access to game engine code
- Render color buffer in lower resolution (1/4)
- Render edge buffer in full resolution
  - Depth discontinuities between pixels
  - ID-buffer discontinuities between pixels
- Joint bilateral upsampling of color
  - Color buffer as primary channel
  - Edge buffer as secondary channel

Anti-aliasing in Real-Time Graphics

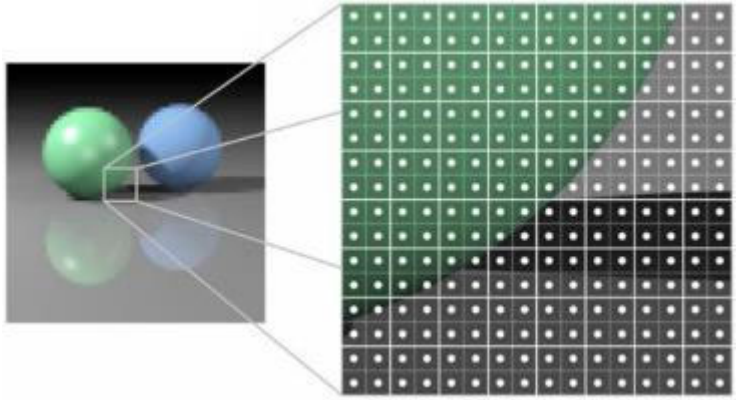
Anti-aliasing can mainly be categorized in

- Better sampling (SSAA, MSAA, TAA)
  - Take more samples during rasterization
- Better filtering (MLAA, TAA)
  - After rasterization as a post process
  - Often incorporates temporal and/or spatial information for filtering





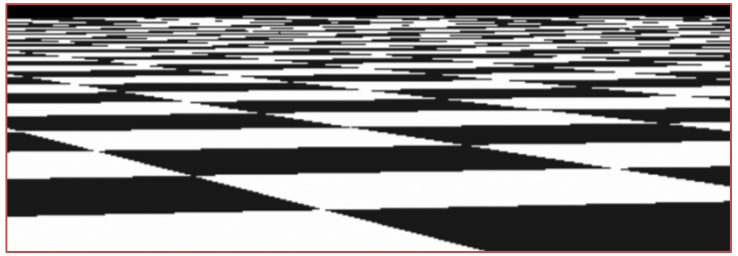
Supersampled Anti-Aliasing (SSAA)



- Multiple samples per pixel
- Accumulated via reconstruction filter
- Equivalent to rendering in higher resolution, then downsampling
- Most accurate antialiasing method
- Helps with all forms of spatial aliasing
- But very expensive, since load increases across whole pipeline

SSAA Algorithm

- Allocate all screen-sized textures and buffers with *N* x resolution
- Render as usual (at higher resolution)
- For display, downsample final color texture to target resolution using fullscreen shader
- Box filter, Lanczos filter, Gaussian…
- Display final image in target resolution

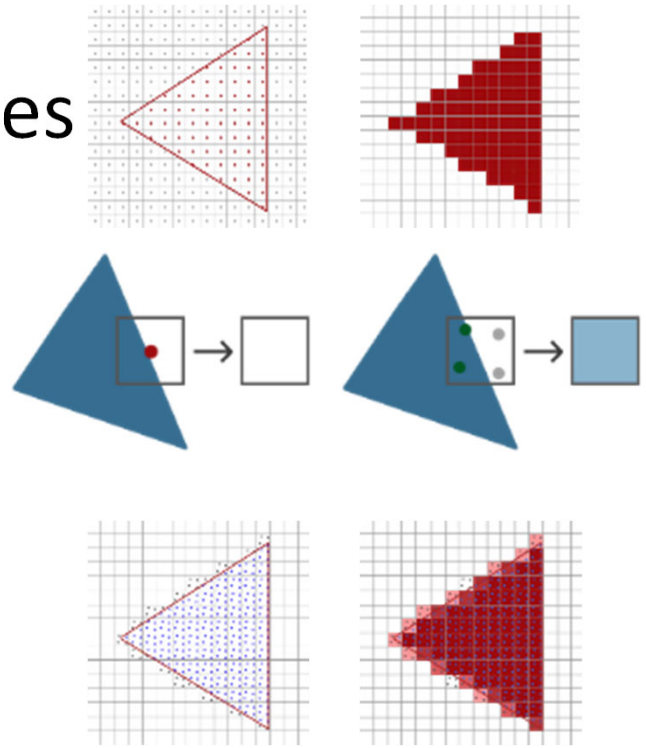


1 sample per pixel

256 samples per pixel

Multisampling (MSAA)

- Most aliasing effects related to “jaggies”
  - Depth and coverage aliasing
- MSAA places additional subsamples
  - Fragment shader is only invoked once per pixel
  - Output color is blended based on number of covered samples
- Hardware support in graphics API

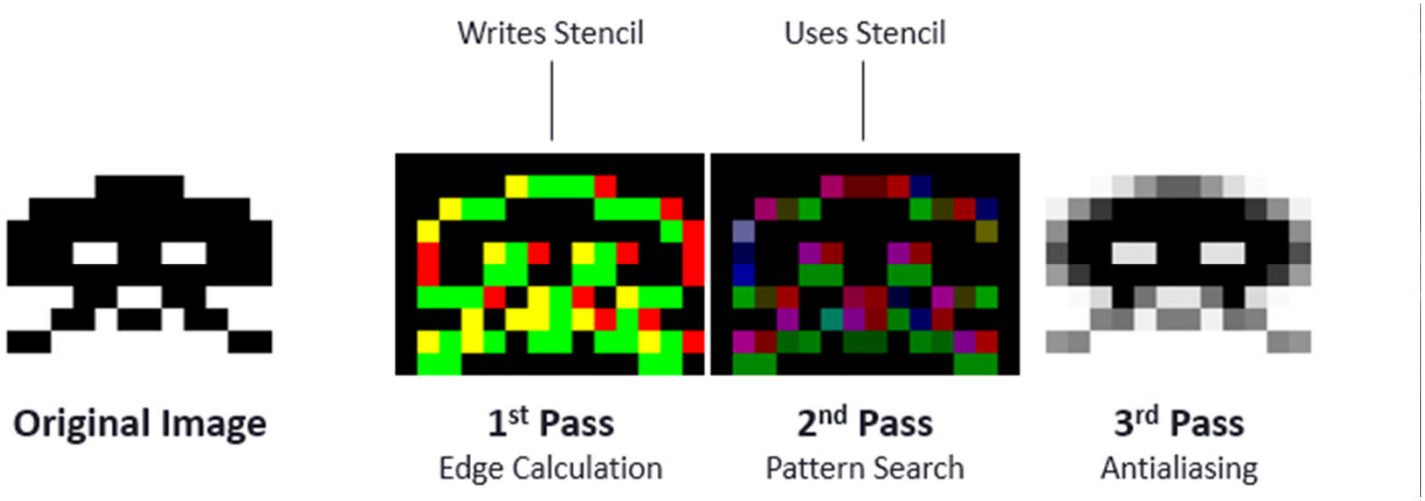


MSAA in OpenGL

```
// Create multisample texture (used as attachment)
#define NUM_MSAA_SAMPLES 4
glBindTexture(GL_TEXTURE_2D_MULTISAMPLE, tex);
glTexImage2DMultisample(GL_TEXTURE_2D_MULTISAMPLE, NUM_MSAA_SAMPLES, GL_RGB, width, height, GL_TRUE);
glBindTexture(GL_TEXTURE_2D_MULTISAMPLE, 0);
// Attach the multisample texture to our framebuffer as color attachment
glBindFramebuffer(GL_FRAMEBUFFER, multisampledFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D_MULTISAMPLE, tex, 0);
// Render
render();
// Resolve the multisample FBO by blitting into the default (display) framebuffer
glBindFramebuffer(GL_READ_FRAMEBUFFER, multisampledFBO);
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
glBlitFramebuffer(0, 0, width, height, 0, 0, width, height, GL_COLOR_BUFFER_BIT, GL_NEAREST);
```

Morphological Anti-Aliasing (MLAA)

1. Edge detection in input image
2. Compute blending texture based on distance to line end/crossing and line shape
3. Anti-Aliasing using blending map from step 2



MLAA Example

SSAA vs. MSAA vs. MLAA

- SSAA
  - Spatial anti-aliasing for everything (textures, edges, shading)
  - Very expensive, therefore hardly used for real-time rendering
- MSAA
  - Resolves aliasing for edges only (not for textures and shading)
  - Too expensive with deferred rendering, alpha aliasing causes issues
  - Current gold standard for VR
  - Not applicable to deferred rendering engines
- MLAA
  - Blurs mostly along edges, resolving edge-aliasing only
  - Very cheap post-process even for weaker systems
  - Can look overly blurred
  - Can cause issues with fine details such as text, requires fine-tuning
  - Not temporally stable → flickering
  - Preferred solution for deferred rendering engines

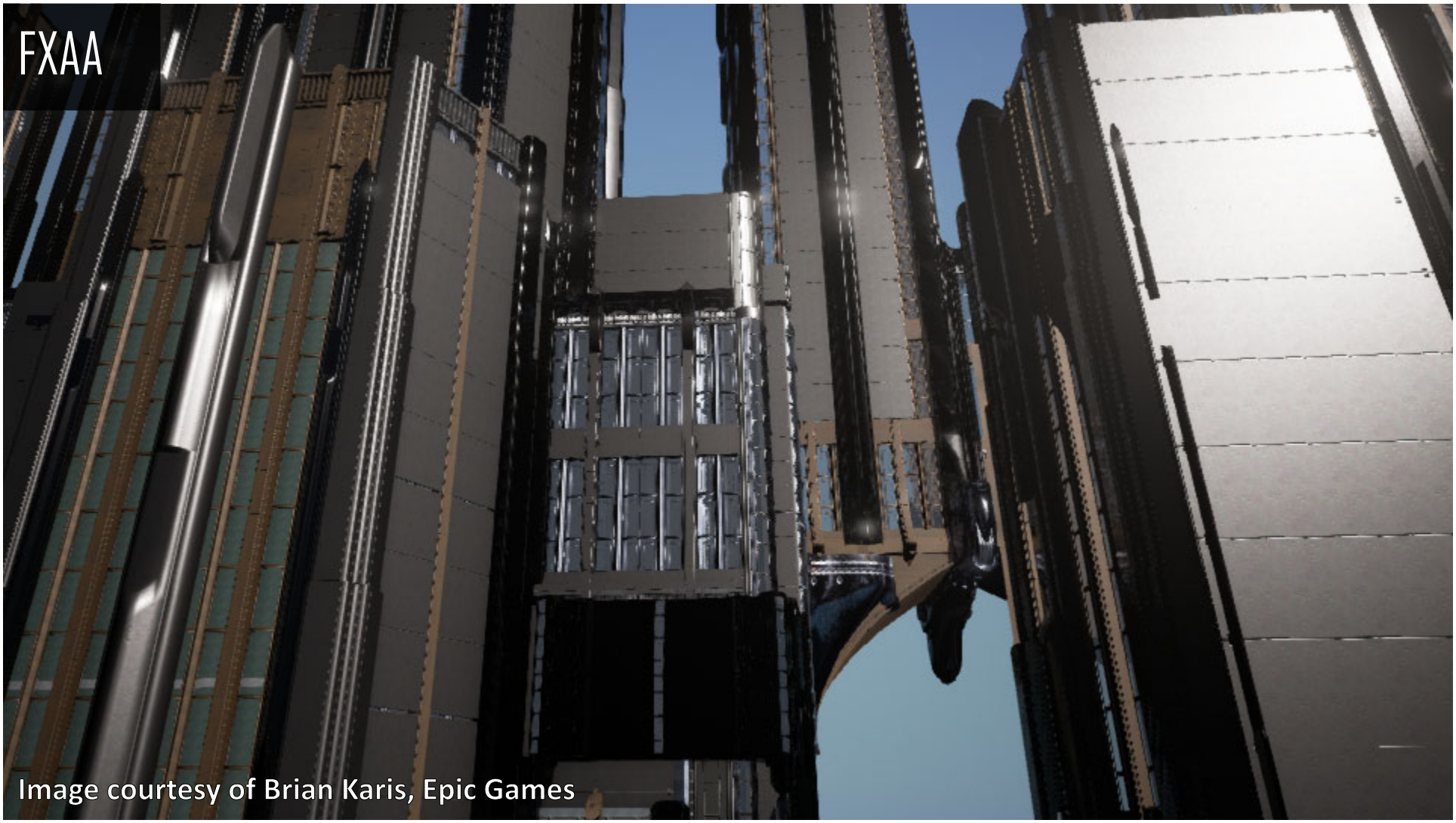
Temporal Accumulation

- Idea: combine multiple samples taken at different times
- Past samples are stored in a buffer
- Examples
  - Motion blur
  - Temporal anti-aliasing

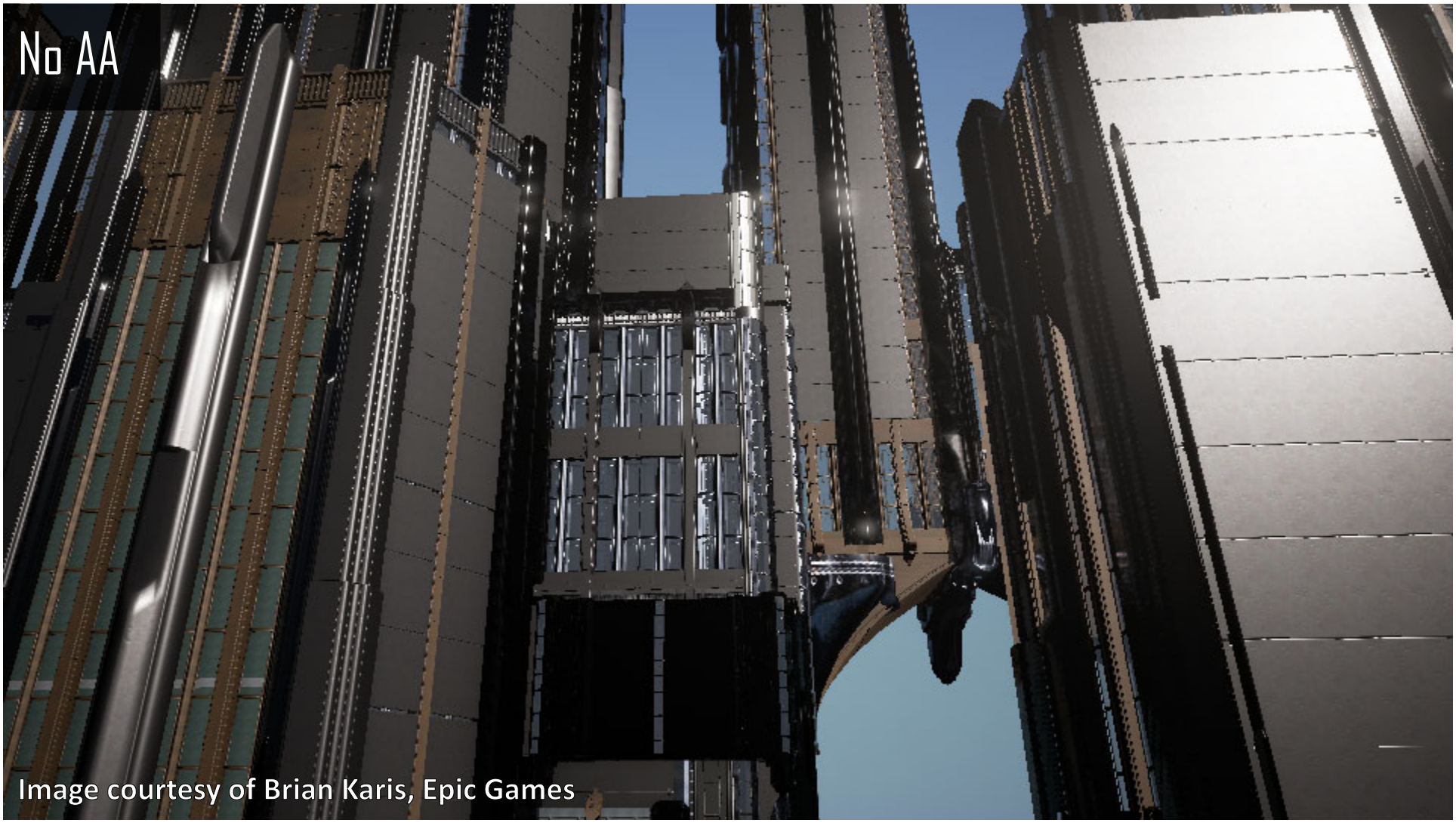
Temporal Anti-Aliasing (TAA)

- Idea: Blend previous frames' **jittered shading result** with successive frames
  - Stochastic/subpixel supersampling
  - Works for **geometric and shading aliasing**
- Current gold standard for game engines
  - Works with any rendering architecture
- Requires more involved modifications to the renderer compared to other AA solutions

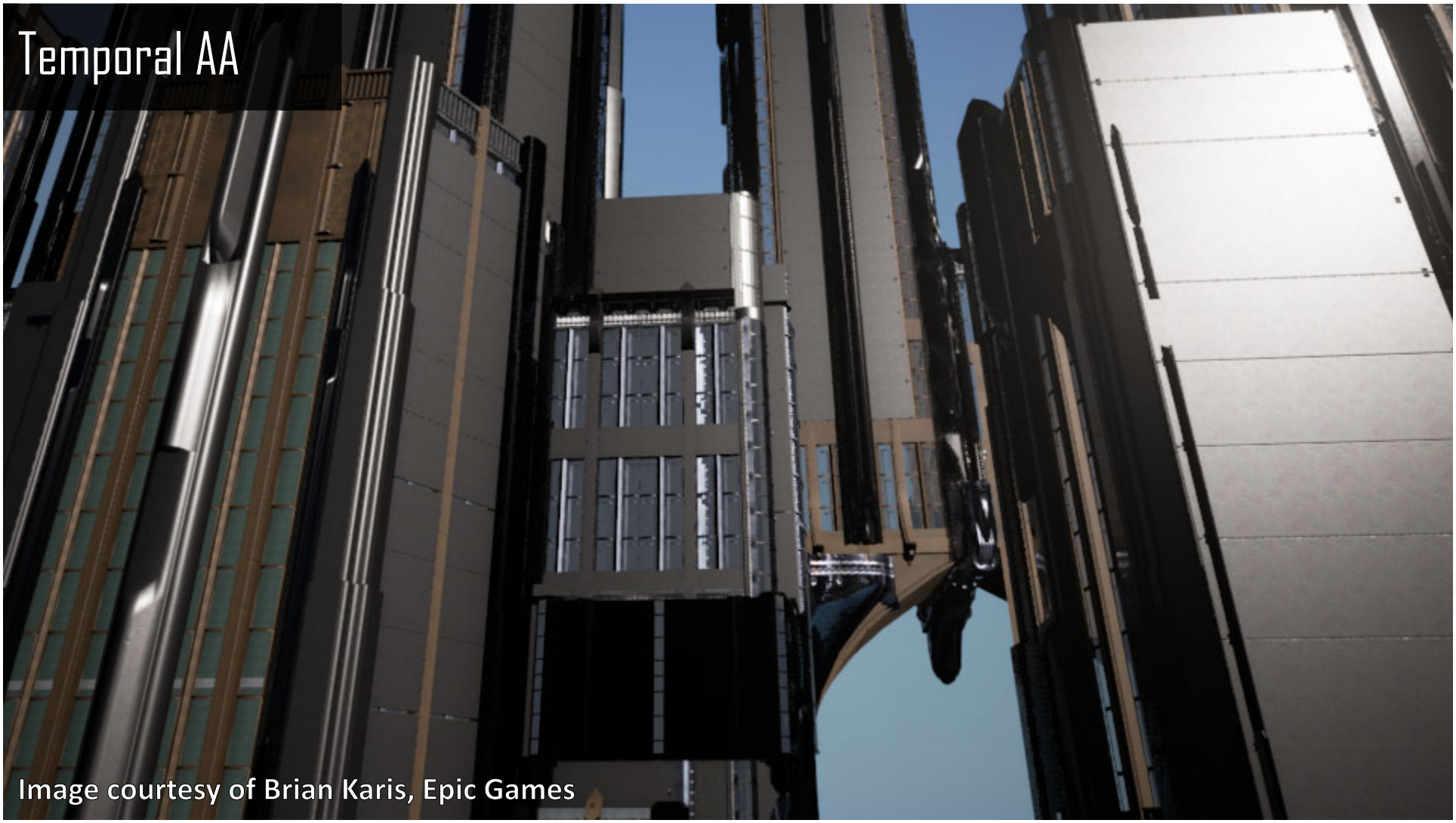
Temporal Anti-Aliasing (TAA)



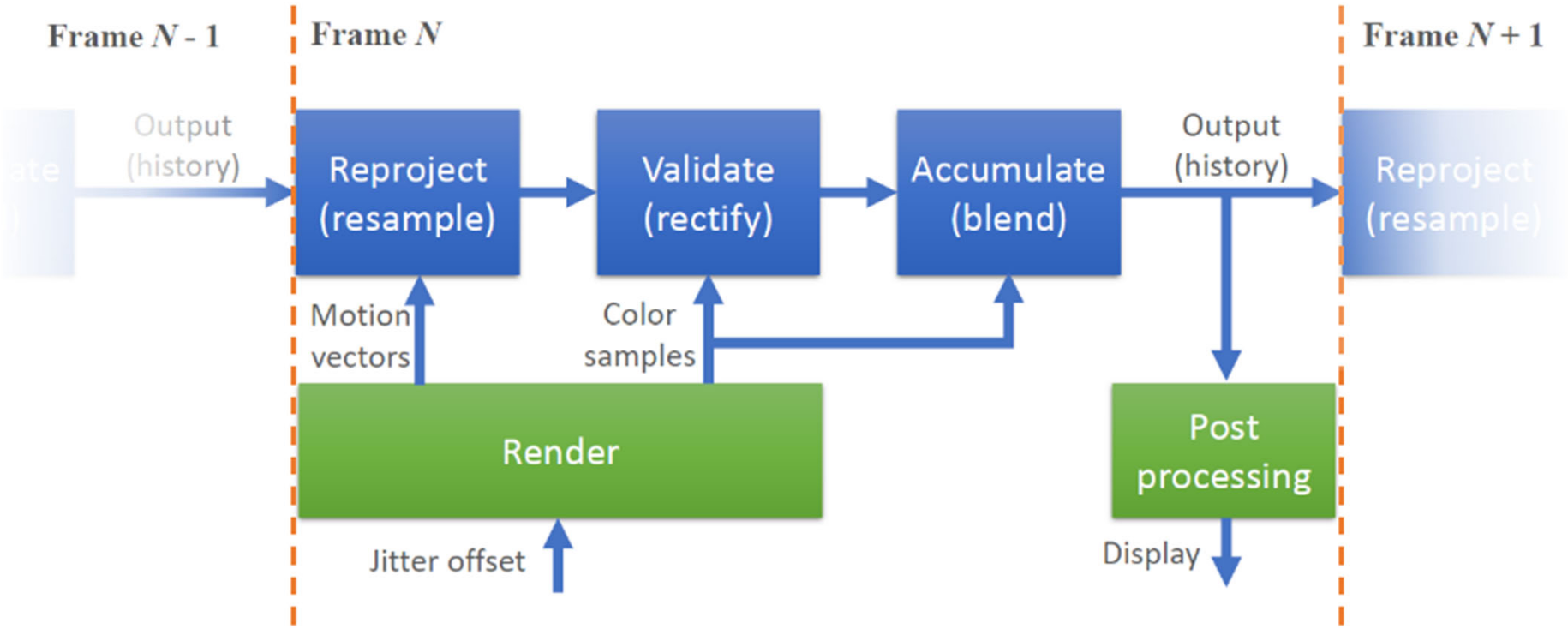
Temporal Anti-Aliasing (TAA)



Temporal Anti-Aliasing (TAA)



Temporal Anti-Aliasing (TAA)



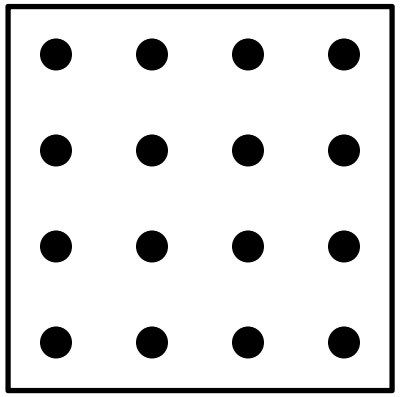
Yang, L., Liu, S. and Salvi, M. (2020), A Survey of Temporal Antialiasing Techniques. Computer Graphics Forum, 39: 607-621. https://doi.org/10.1111/cgf.14018

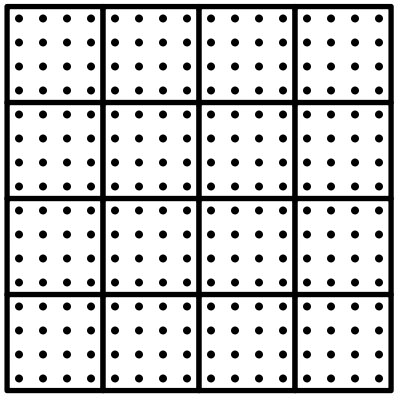
TAA: Jittering

- To generate sampling locations for temporal supersampling, we **jitter the projection matrix**

```
ProjMatrix[2][0] += ( SampleX * 2.0f – 1.0f ) / ViewRect.Width();
ProjMatrix[2][1] += ( SampleY * 2.0f – 1.0f ) / ViewRect.Height();
```

Regular grid

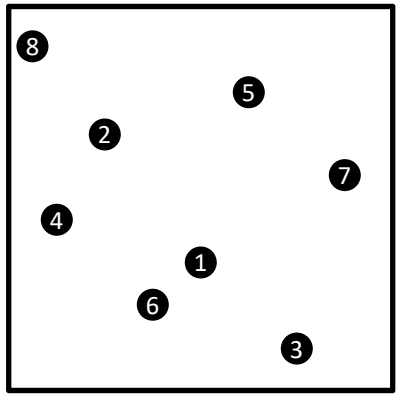


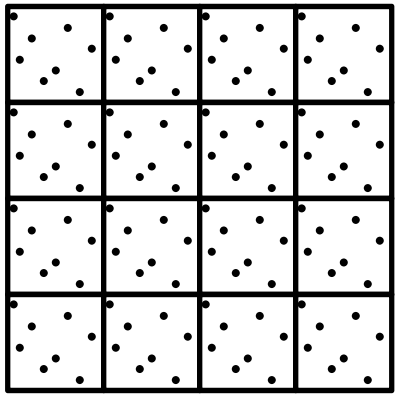


TAA: Jittering Sequence

- For fast convergence, each subsequence should be evenly distributed in the pixel domain
- **Halton(2, 3)** is a good candidate (used in UE4)

Halton(2,3)



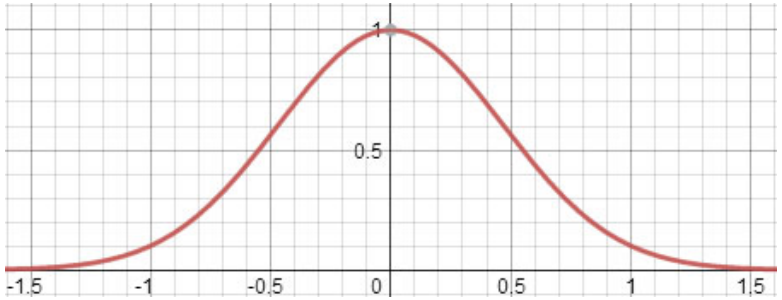


TAA: History Buffering

- Simple moving average is not feasible
  - Would require N times storage for N samples
- Exponential moving average
  - Nearly “infinite” number of samples with fixed storage
  - When α is small, approaches a simple moving average
  - α = 0.1 is common
- This can simply be stored in a screen-sized buffer!

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1}$$

TAA: Reconstruction



$$W(x) = e^{-2.29x^2}$$

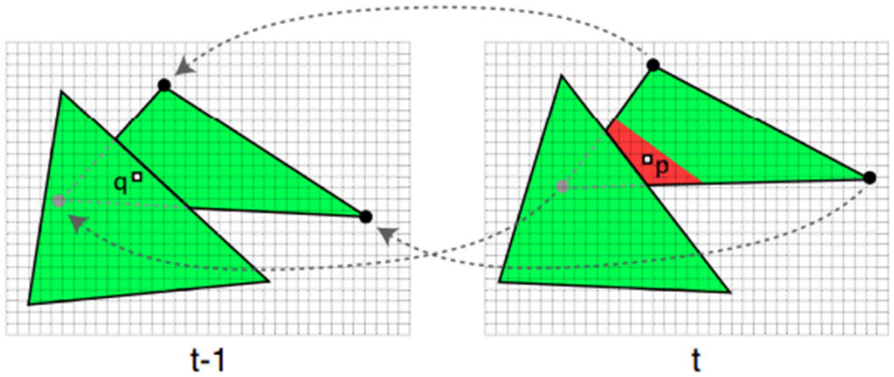
- Simple approach
  - Average all subpixels inside pixel rectangle (Box filter)
  - However, box filter not stable under motion
- Alternative approach: Gaussian
  - UE4 uses a Gaussian fit to Blackman-Harris 3.3
  - Since every pixel shares the same jitter, filter weights are precomputed and passed as shader uniforms

TAA: Dynamic Scenes

- So far, we discussed jittering, averaging and reconstruction
- What happens if the sample location changes between frames?
  - Animated objects, moving camera…
- For a given pixel, **we need to locate its previous location in the history buffer**
  - **Reprojection**

TAA: Reprojection

- The history for the current pixel may be at a different location, or might not exist at all
- The geometry is transformed twice, using
  - 1.) the current frame's transformation matrices
  - 2.) the previous frame's transformation matrices
- The resulting offsets are stored into a **motion vector texture**
- Using this texture, we can look up the previous location in the history buffer

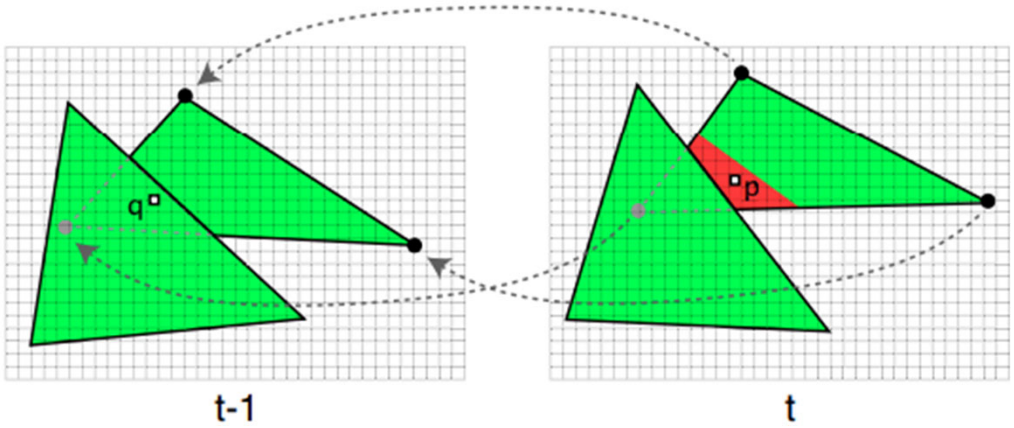


Diego Nehab, Pedro V. Sander, Jason Lawrence, Natalya Tatarchuk, and John R. Isidoro. 2007. Accelerating real-time shading with reverse reprojection caching. https://dl.acm.org/doi/10.5555/1280094.1280098

TAA: Reprojection Ghosting

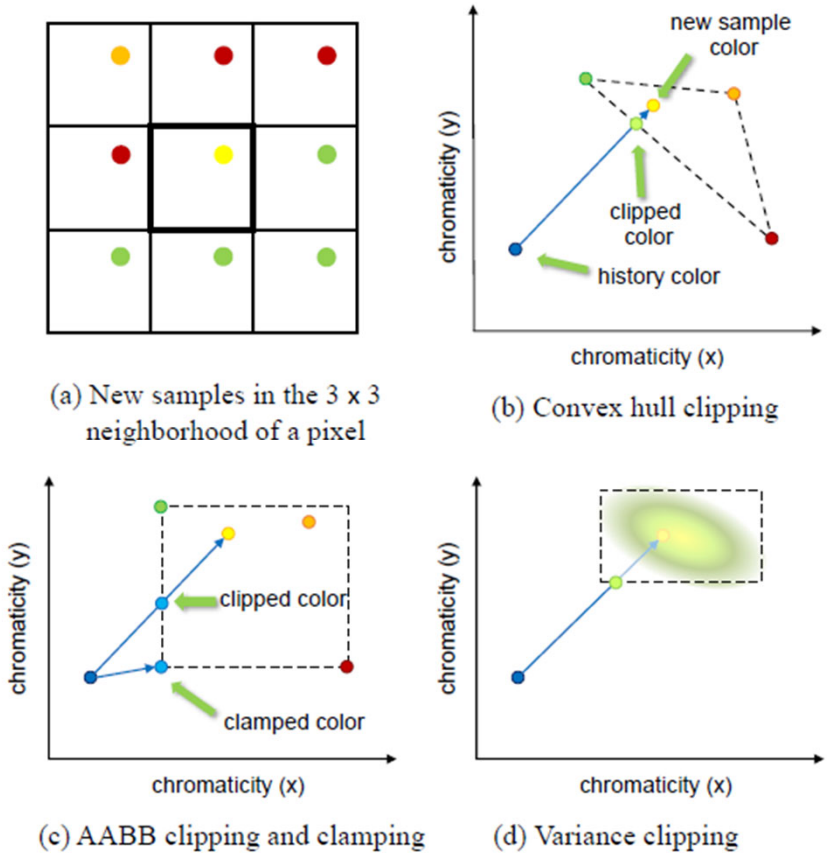
TAA: History Rejection

- When reprojecting, the history might be invalid
  - Occlusion/Disocclusion events
  - Shading changes (lights turning on/off…)
- Use **geometry data** (depth, normal, object ID, motion vector) or **color variance** to detect history confidence
- When history is invalid, simply clear the history buffer (set α to 1)



TAA: History Rectification

- Instead of rejecting history, we can make data more consistent with the new samples
  - Clip existing history to neighborhood of new sample
  - Further reduces temporal artifacts



(a) New samples in the 3 x 3 neighborhood of a pixel

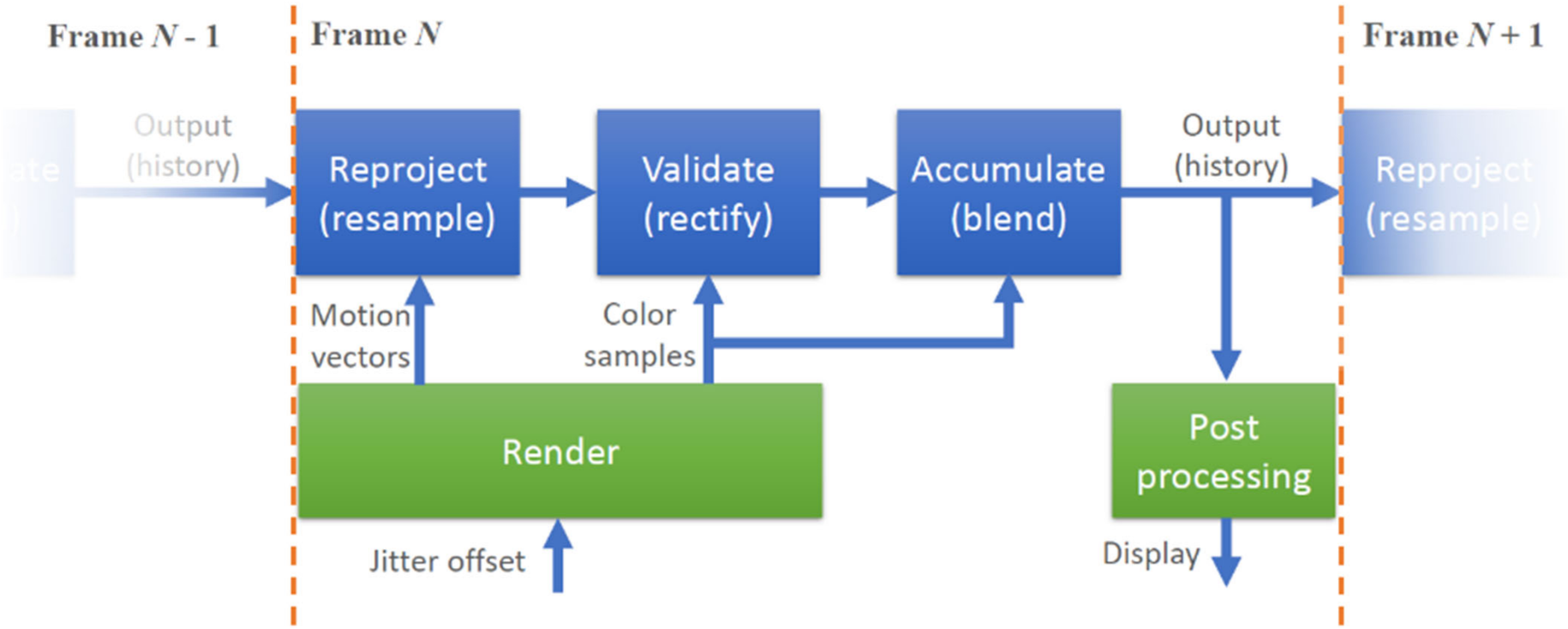
(b) Convex hull clipping

(c) AABB clipping and clamping

(d) Variance clipping

Yang, L., Liu, S. and Salvi, M. (2020), A Survey of Temporal Antialiasing Techniques. Computer Graphics Forum, 39: 607-621. https://doi.org/10.1111/cgf.14018

TAA: Review



Yang, L., Liu, S. and Salvi, M. (2020), A Survey of Temporal Antialiasing Techniques. Computer Graphics Forum, 39: 607-621. https://doi.org/10.1111/cgf.14018

TAA: Steps for Assignment 3

1. Implement a history buffer
2. Jitter the camera projection and accumulate/reconstruct samples
   - Test with a static camera at this point
   - This should already anti-alias your static image nicely
3. Generate motion vector texture
   - Since you need to extend shaders to have the previous and the current transform, it's ok to just do it for the camera for the exercise
4. Implement history rejection based on depth
   - At this point, you should be able to move the camera around with a lot fewer artifacts
5. Optional: implement further improvements

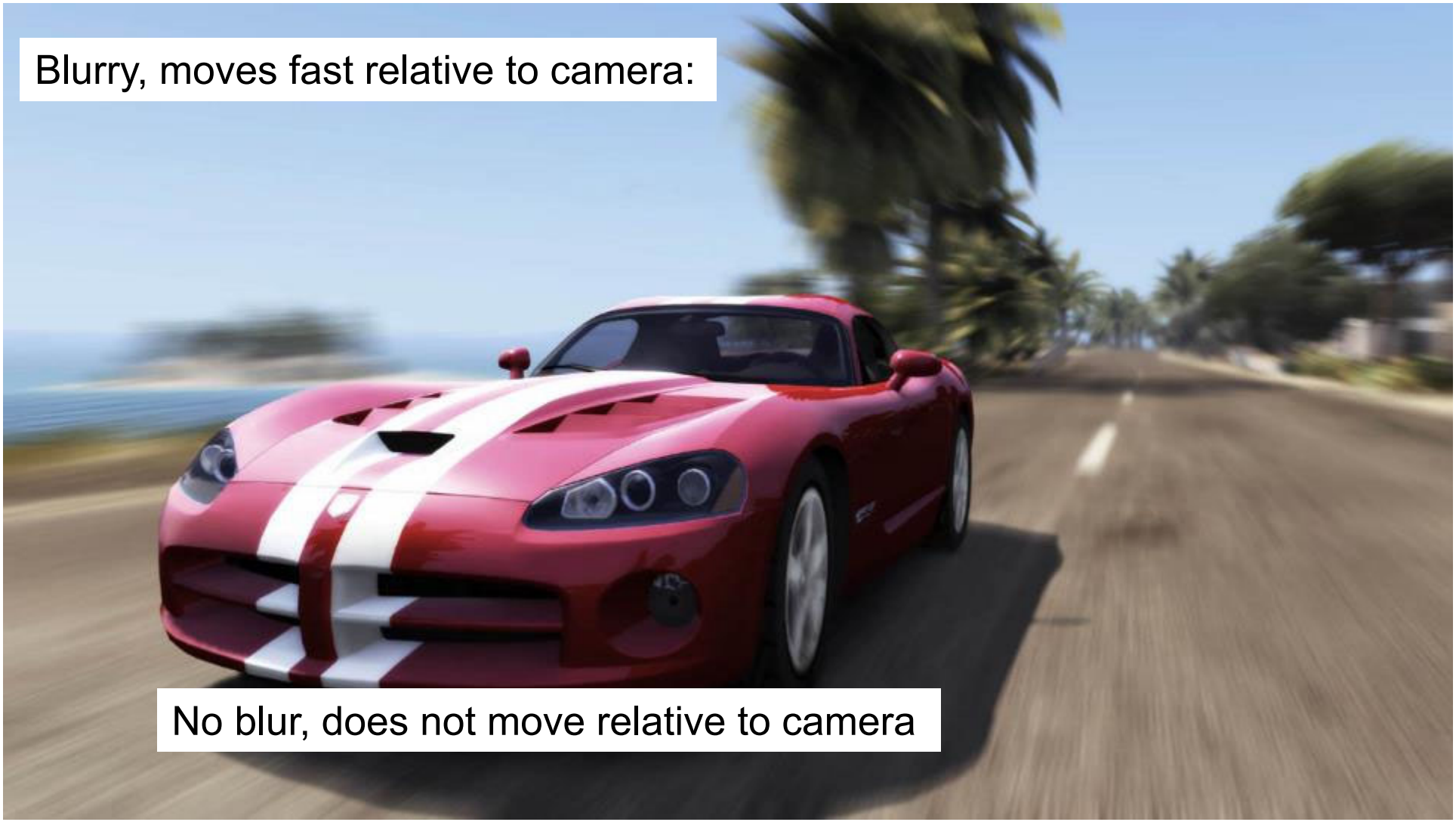
Motion Blur

- Fast moving objects appear blurry
- Property of the human eye and cameras
- Cameras: too long exposure
- Humans: moving the eye causes blur
- Advantages:
  - Looks good/realistic
  - Can cover performance problems

Motion Blur Example

Picture: Crysis (Object Based Motion Blur)

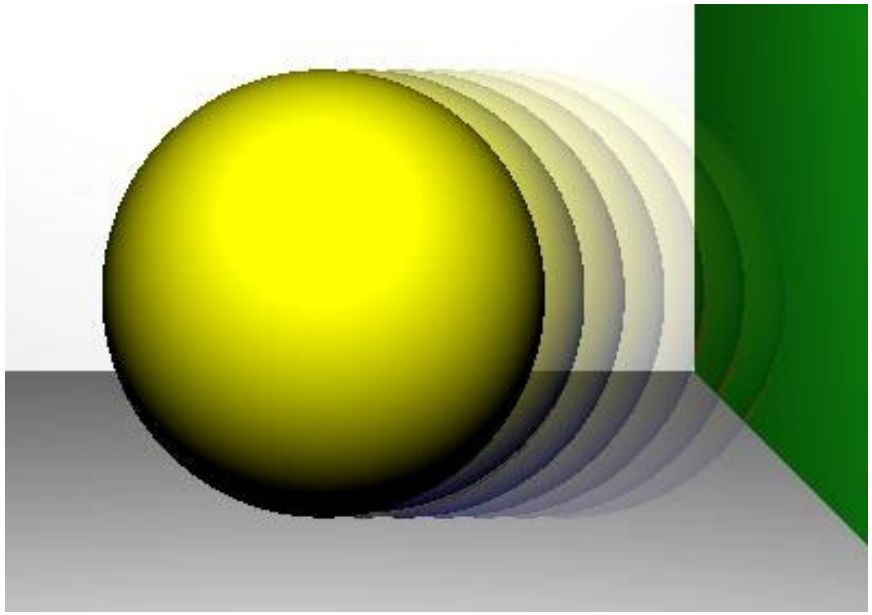
Motion Blur with Moving Camera



Continuous vs Discrete

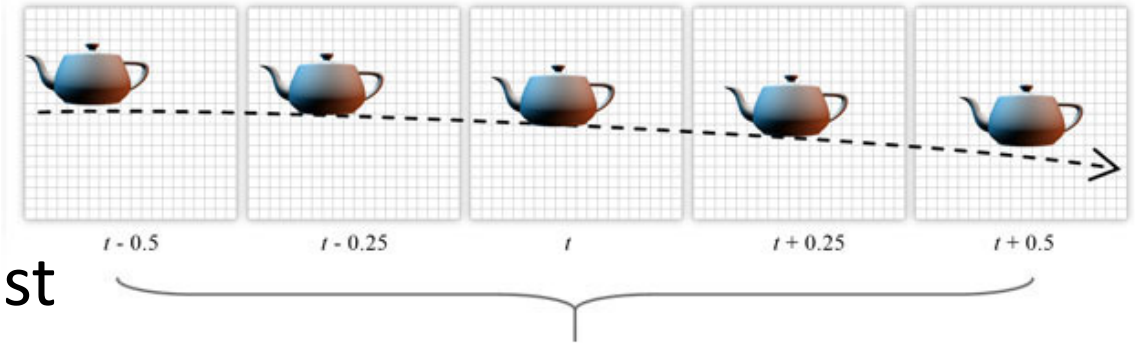
Is a continuous effect

Correct, continuous MB

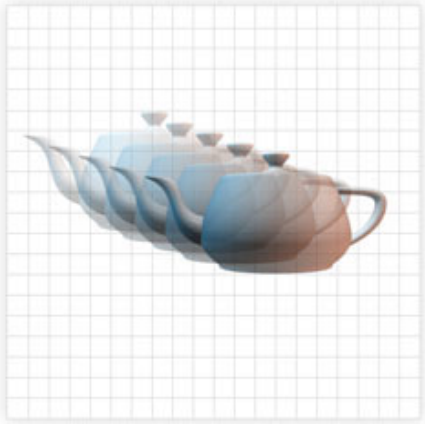


Approximated, discrete MB

Discrete Methods



- Simplest method
  - Render object at past positions with varying transparency
  - Object needs to be rendered multiple times

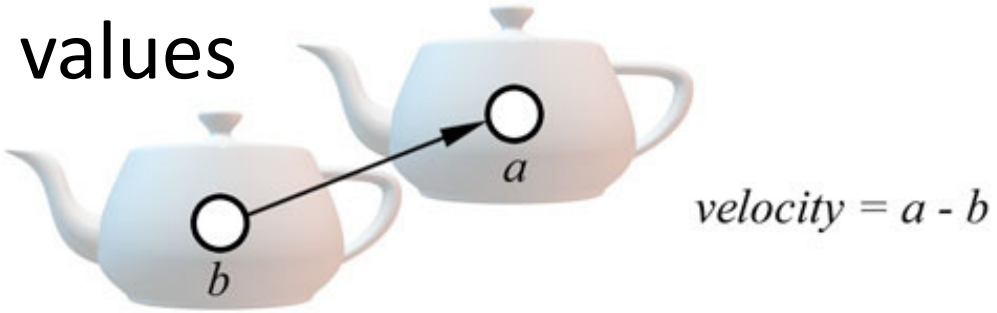


- Image Space Motion Blur
  - Render object to buffer
  - Copy buffer with varying transparency
  - More efficient

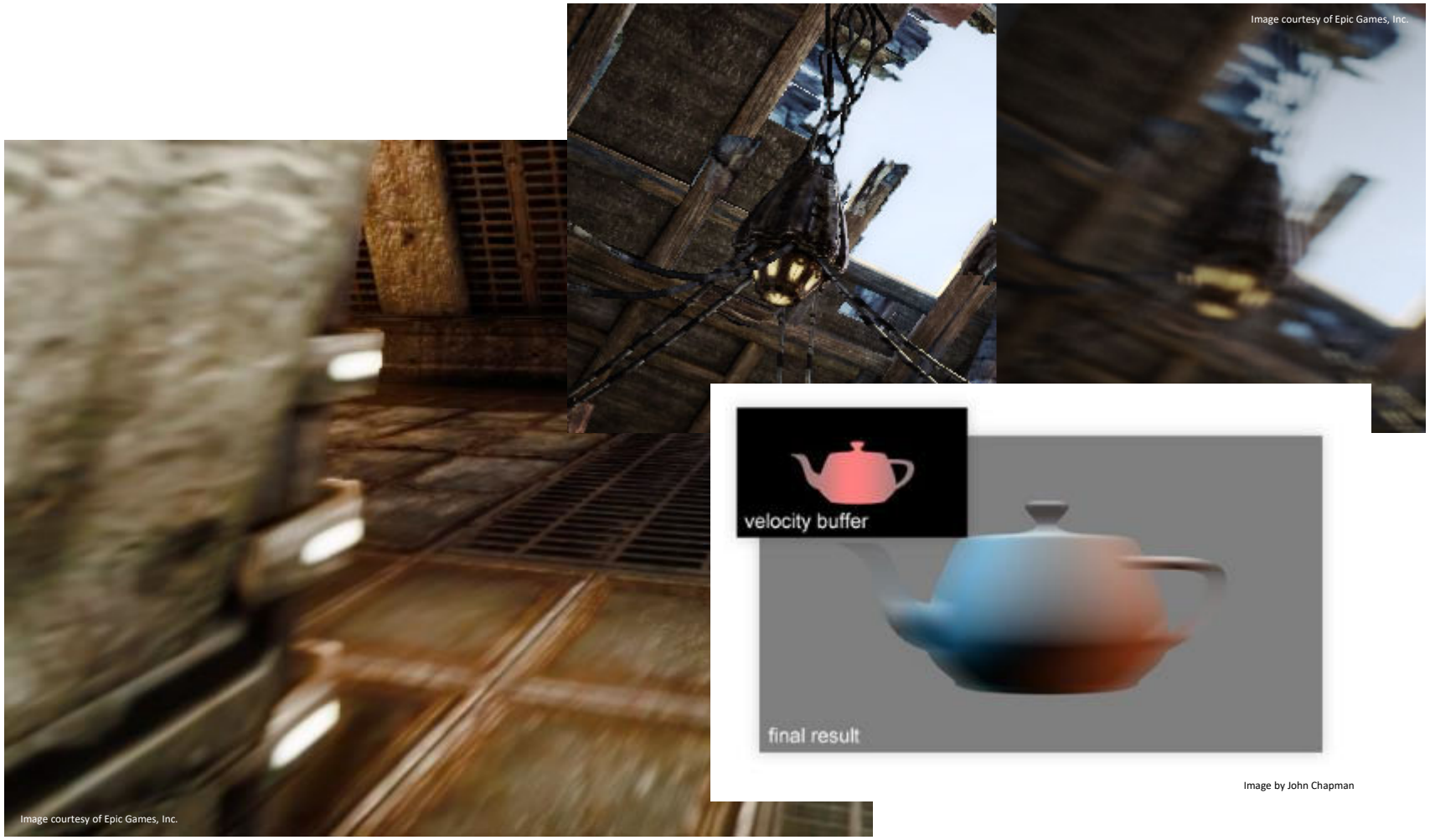
Continuous Motion Blur

For each pixel:

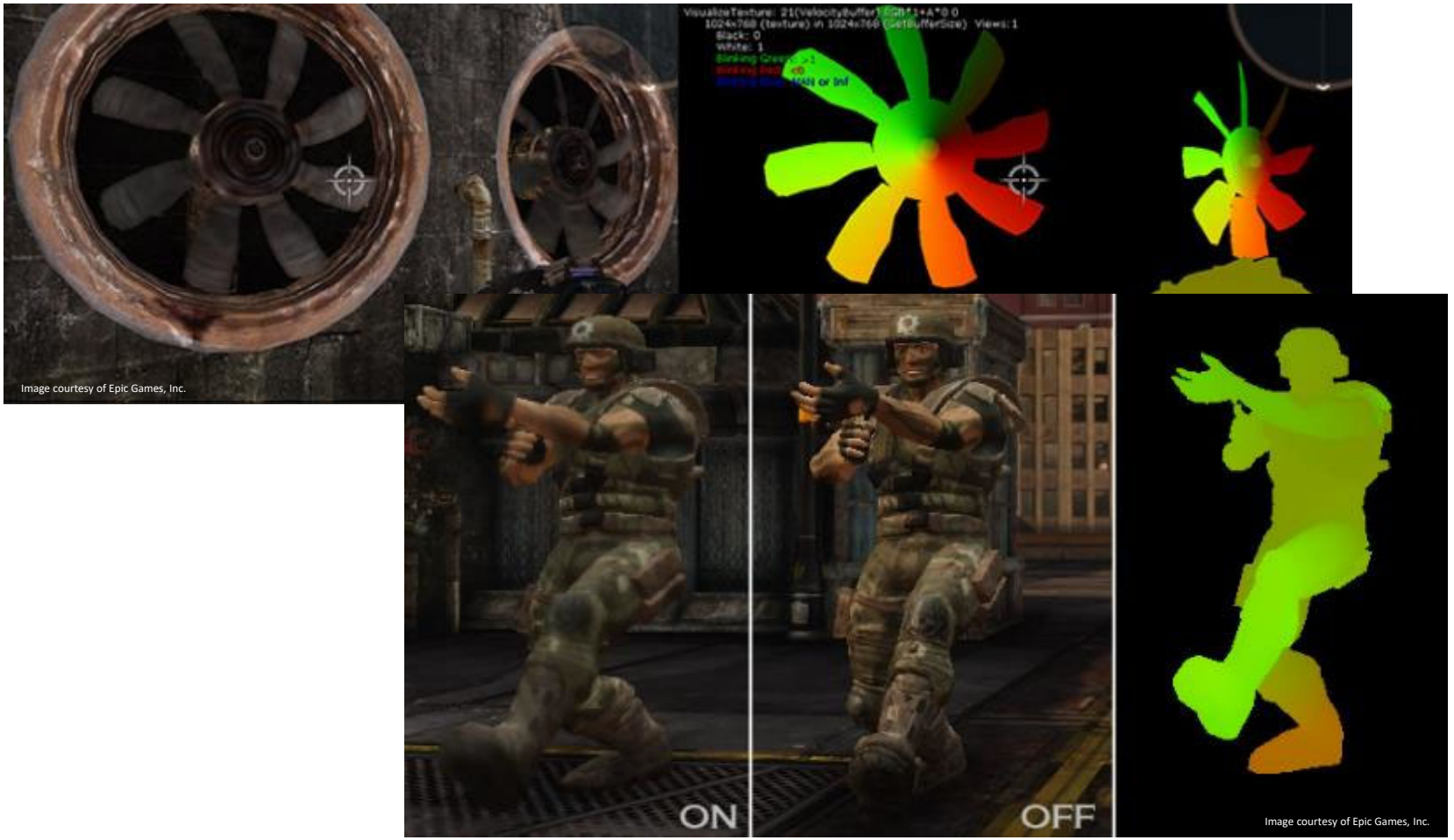
- Compute how pixel moves over time
- Current and previous model-view projection matrix form *velocity buffer*
- Sample line along that direction
- Accumulate color values



Continuous Motion Blur – Examples 1



Continuous Motion Blur – Examples 2

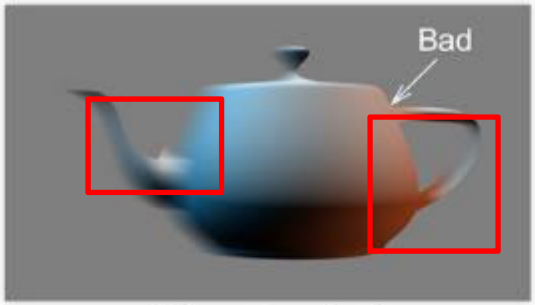


Continuous Motion Blur – Artifacts

- Color bleeding
  - Slow foreground objects bleed into fast background objects

Images by John Chapman

- Discontinuities at silhouettes



Blur not centred

Blur centred

Image Compositing

- Image compositing idea
  - Use alpha channel to combine multiple buffers
  - Buffers contain partial rendering results
- Examples
  - Lens flare
  - Billboards
  - Particle systems

Lens Flare

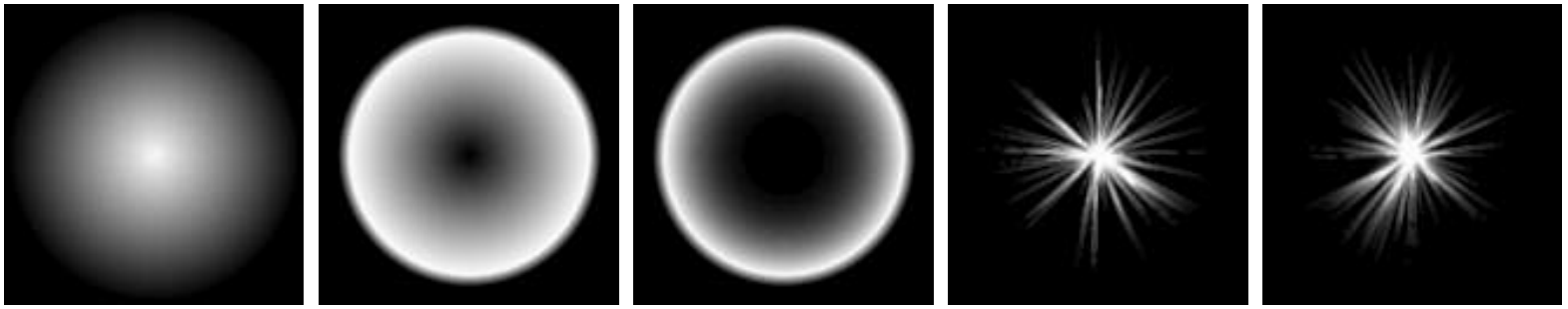
- A shortcoming of cameras that photographers try to avoid
- However: looks realistic and fancy
- Effect occurs inside lens system
  - Always on top
- Happens when light source inside image
- Star, ring or hexagonal shapes

Lens Flare Example 1

Lens Flare Example 2

Lens Flare Rendering

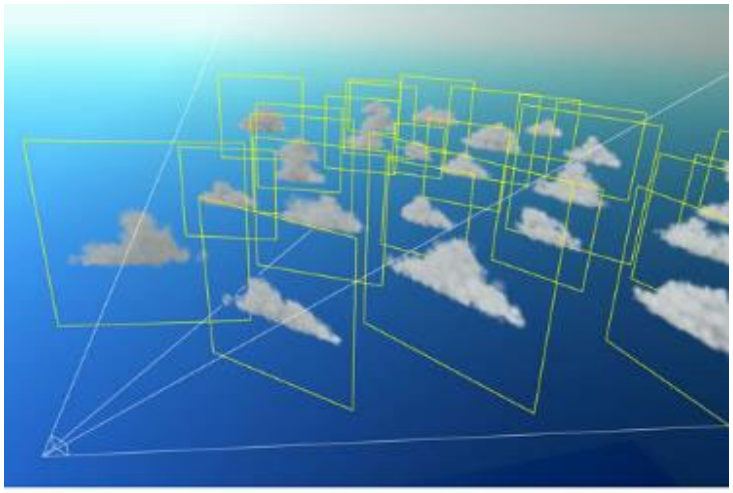
- Choose a lens flare texture
- All lens flares lie on the line between light source and image center
- Rendered with differently sized transparent billboards (alpha blending)



Don‘t overdo it!

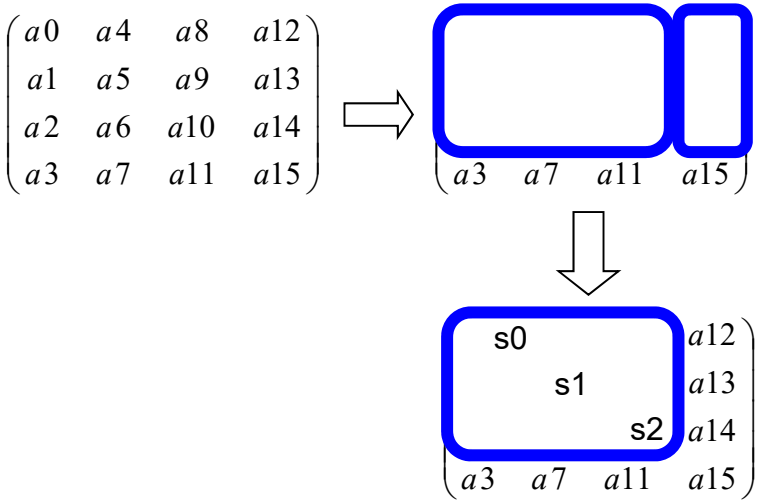
Billboards

- Synonyms: impostors, sprites
- Textured rectangles which
  - Face the viewer, or
  - Are aligned with some axis
- Can be used in large quantities
  - Simple, only 2 textured triangles
- Low memory footprint
- Only look good at a distance or when small
- Example: clouds in a game



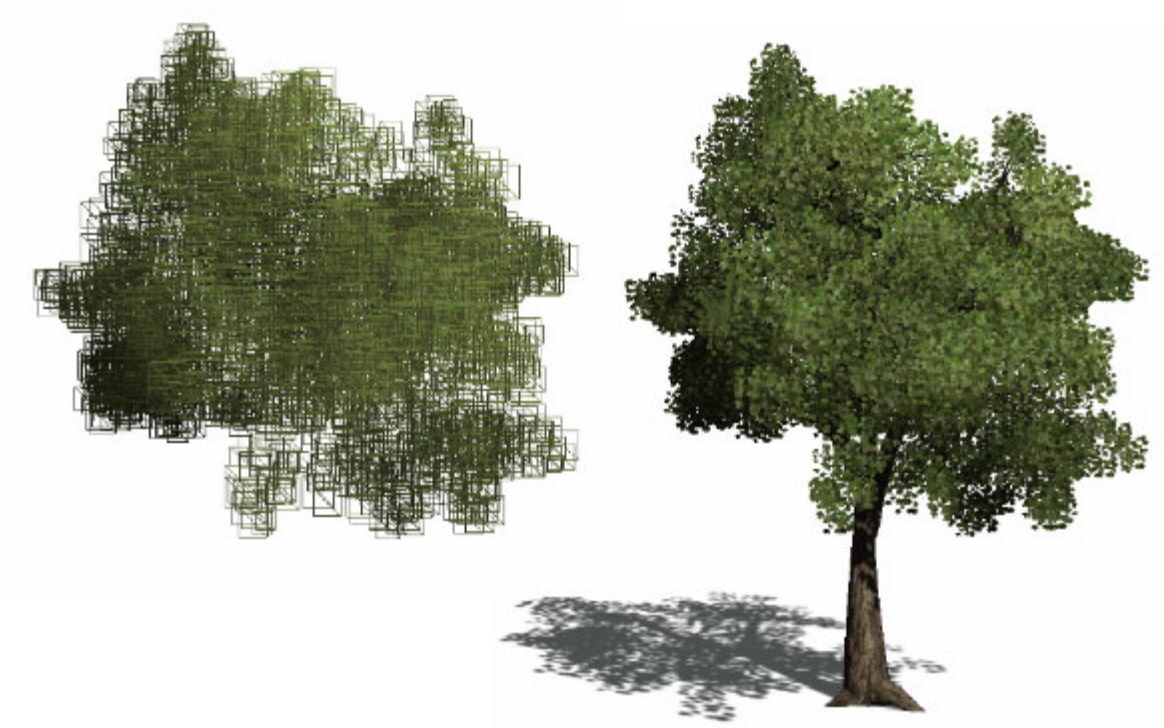
Billboard needs to face camera

- How: modify the ModelView matrix (remove rotation)



- Maintain scale!
- Result: BB will appear at the right position and distance, but will face camera

Billboard Clouds



- A set of billboards with different size/orientation
- Created procedurally (from 3D model or rule set)
- Can be animated by physical simulation

Particle Systems: Introduction

- Modeling of objects changing over time
  - Flowing
  - Billowing
  - Spattering
  - Expanding

- Typically many small objects
  - Rendering with billboards
- State-less vs. state-full particles

Applications

- Modeling of natural phenomena:
  - Rain, snow, clouds
  - Explosions, fireworks, smoke, fire
  - Sprays, waterfalls

https://youtu.be/H_lco7TSUIY

https://youtu.be/UnZMp17lqy4

https://youtu.be/RpiDGqjhOX8

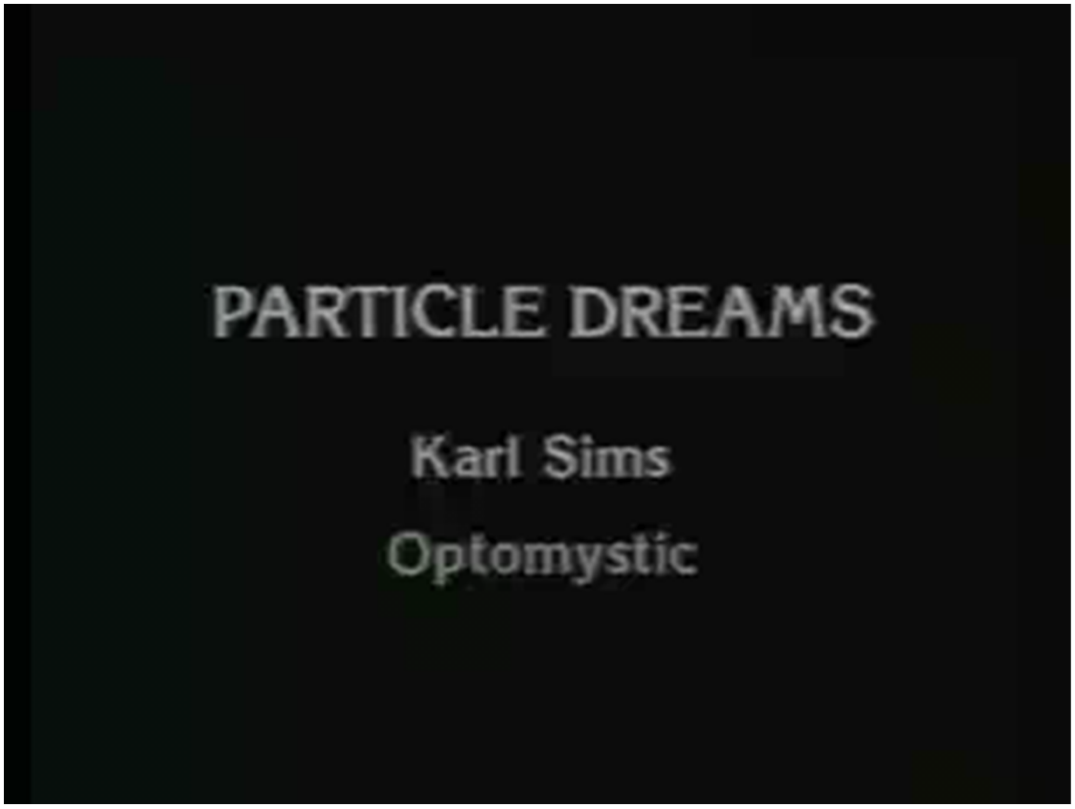
History of Particle Systems 1

- 1982 Star Trek II: “Genesis Sequence”

https://youtu.be/fwTeO40hm6g

History of Particle Systems 2

- 1988: Carl Sims – Particle Dreams



https://youtu.be/bfSGqLpuzNM

Procedure

- All particles of a system use the same update method (share the same properties)
- The particle system handles
  - Initializing
  - Updating
  - Randomness
  - Rendering

Particle System Parameters

- Particle parameters change over time:
  - Location, Speed, lifetime
- Particles “die” after some time
- Particle shapes
  - points, spheres, boxes, arbitrary models
  - Size and shape may vary over time

```
struct particle
{
  float t;           // life time
  float v;           // speed
  float x, y, z;     // coordinates
  float xd, yd, zd; // direction
  float alpha;       // fade alpha
};
```

Particle Physics

- Motion may be controlled by external forces
  - E.g., gravity, collision
- Particles can interfere with other particles
- Causes a more entropic movement, e.g., sprays of liquids

Things to Consider

- Requires fast physics and collision detection
- Correct modeling not important
- Memory consumption important
- Rendering speed important

Questions?