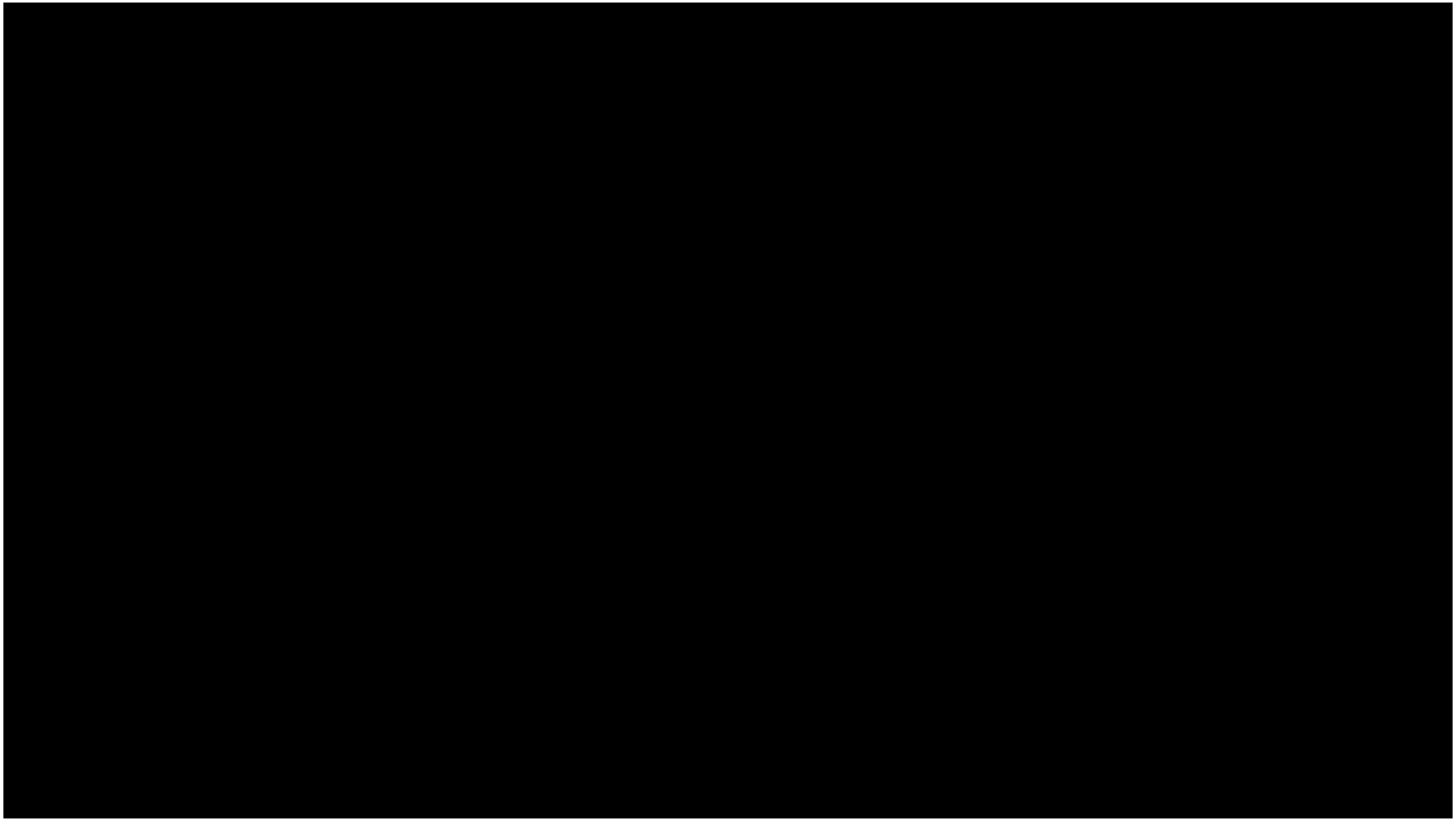
Interactive Computer Graphics: Lecture 16

Special effects

Some slides adopted from
Daniel Wagner, Michael Kenzel, TU-Graz

Motivation



Motivation

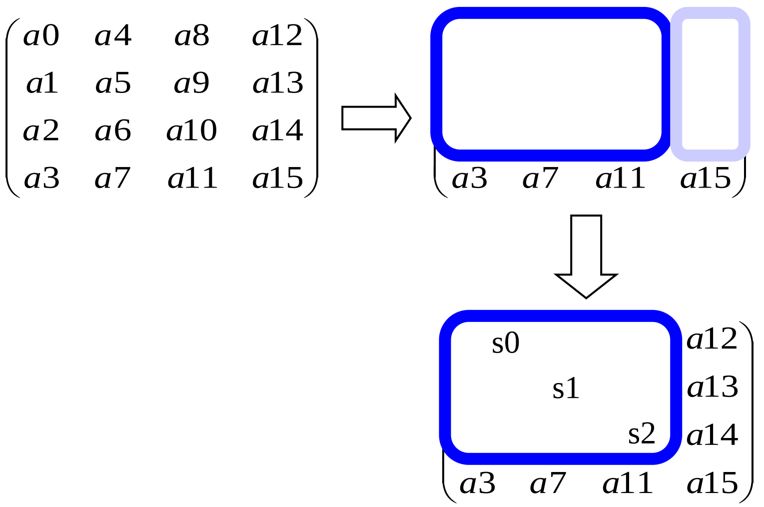
- Add special effects in image space after rendering
  - Independent of geometric scene complexity
  - Often uses image processing techniques
- Irregular objects: billboards, particle systems
- Distance to camera: fog, depth of field
- Camera exposure: motion blur
- Camera aperture: bokeh, lens flare
- Semi-global illumination: reflection, transparency, ambient occlusion
- Non-photorealistic rendering

Billboards

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

Billboards

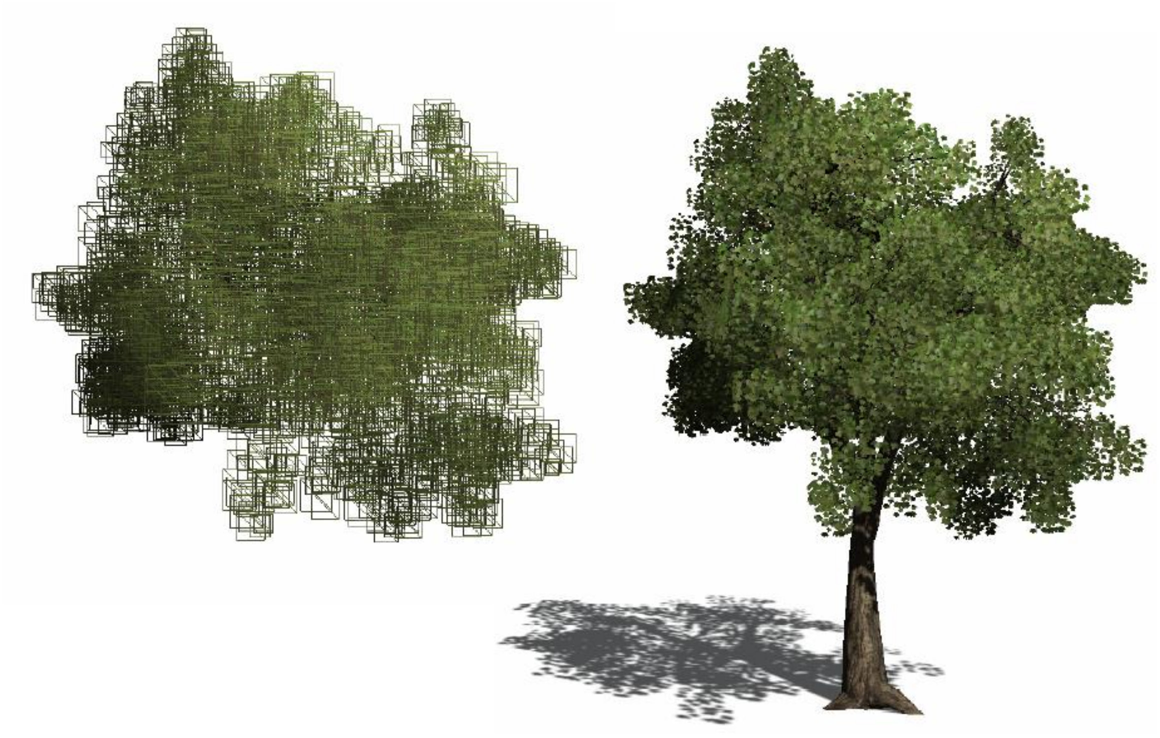
- How: modify the ModelView matrix (remove rotation)



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

Billboards

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



Particle systems

- a system to control collection of a number of individual elements over time (points, line, triangle or Sprit texture), which act independently but share some common attributes:
  - position (3D)
  - velocity (vector: speed and direction)
  - color +(transparency)
  - lifetime
  - size, shape

Particle systems

- The first CG paper about particle systems by William T. Reeves: Particle Systems A Technique for Modeling a Class of Fuzzy Objects. Computer Graphics, vol. 17-3, July 1983
- in “Star Trek II: The Wrath of Kahn” 1983

Particle systems

- “Star Trek II: The Wrath of Kahn” 1983

Particle systems

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

Particle systems

- All particles of a system use the same update method (share the same properties)
- The particle system handles
  - Initializing
  - Updating
  - Randomness
  - Rendering
- Particle parameters change
  - Location, Speed, lifetime
- Particles are emitted somewhere and “die” after some time

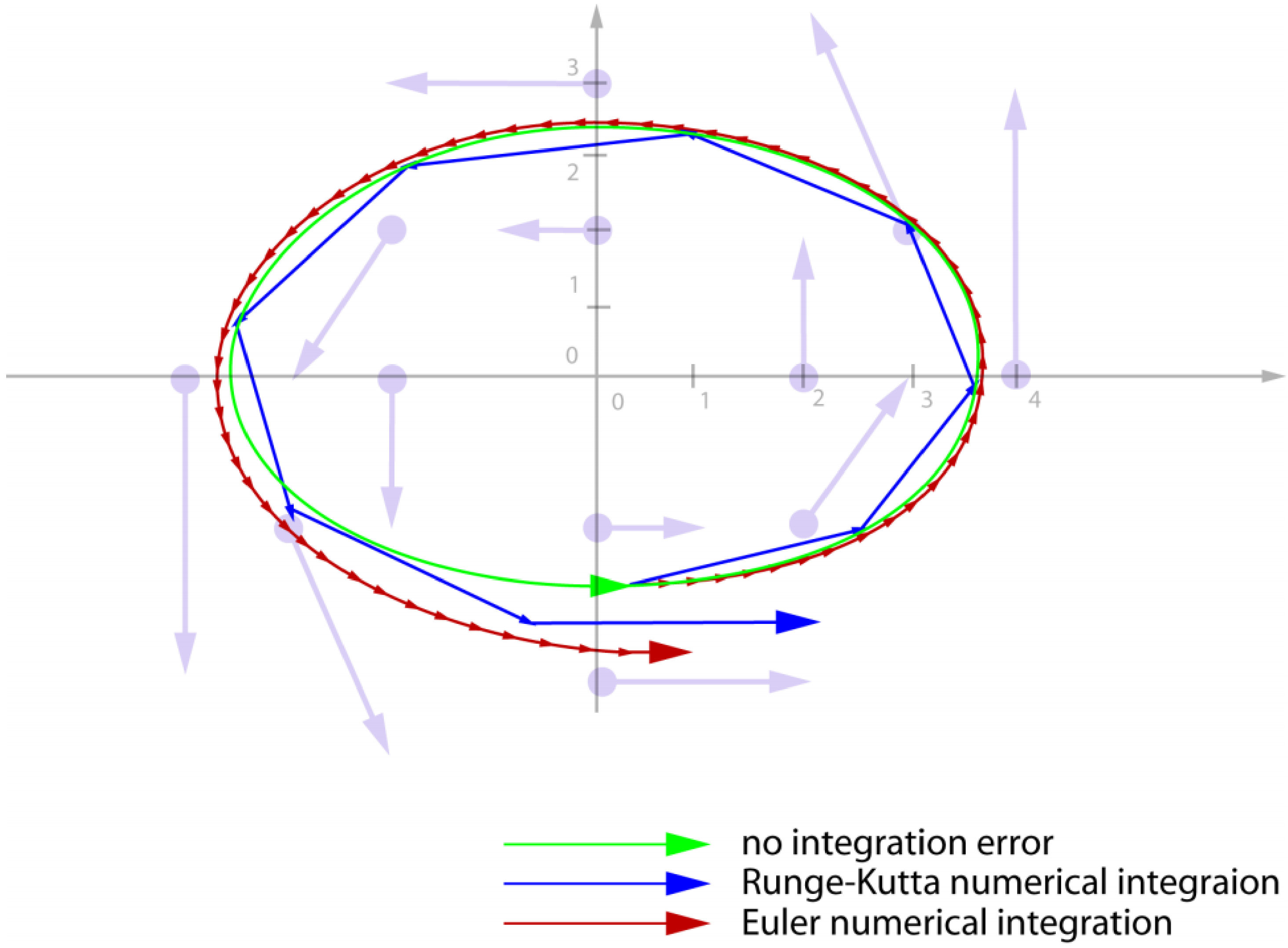
```
struct particle
{
  float t;          // life time
  float v;          // speed
  float x, y, z;    // coordinates
  float xd, yd, zd; // direction
  float alpha;      // fade alpha
};
```

Particle systems / Physics

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

MatthiasM Videos

Particle systems / Integration



Particle systems / Integration / Euler

- the continuous movement of a massless particle under the influence of an evenly varying vector field

$$\frac{\partial x}{\partial t} = v(x(t), \tau), \quad x(t_0) = x_0, \quad x : \mathbb{R} \rightarrow \mathbb{R}^n$$

- v is a sampled vector field whose sampled values depend on the current position of an particle x(t)
- The simplest form to solve the initial value problem is the standard explicit Euler-approach
- Step size Δt = h > 0

$$t_{k+1} = t_k + h,$$

$$x_{k+1} = x_k + hv(x_k, t_k, \tau).$$

- accuracy depends on the selected step size Δt

Particle systems / Integration / Runge-Kutta

- Reduce integration error or computational effort with intermediate steps:

$$x_{k+1} = x_k + h \sum_{j=1}^{n} b_j c_j,$$

- With coefficients bj and intermediate steps cj. Each cj is a basic Euler integration step. E.g., n = 4 (Runge-Kutta fourth order, RK4):

$$x_{k+1} = x_k + \frac{h}{6}(c_1 + 2c_2 + 2c_3 + c_4),\ where$$

$$c_1 = v(x_k, t_k, \tau),$$

$$c_2 = v(x_k + \frac{h}{2}c_1, t_k + \frac{h}{2}, \tau),$$

$$c_3 = v(x_k + \frac{h}{2}c_2, t_k + \frac{h}{2}, \tau)\ and$$

$$c_4 = v(x_k + hc_3, t_k + h, \tau).$$

Particle systems

- Interactive animation: http://demonstrations.wolfram.com/UnderstandingRungeKutta/

Fog

- Atmospheric effect (scattering of light)
  - Stylistic element
  - Depth cue
  - Hide artifacts
    - Limited viewing range/clipping at far plane
    - Billboard updates
    - …
- Fog intensity scales with distance to camera

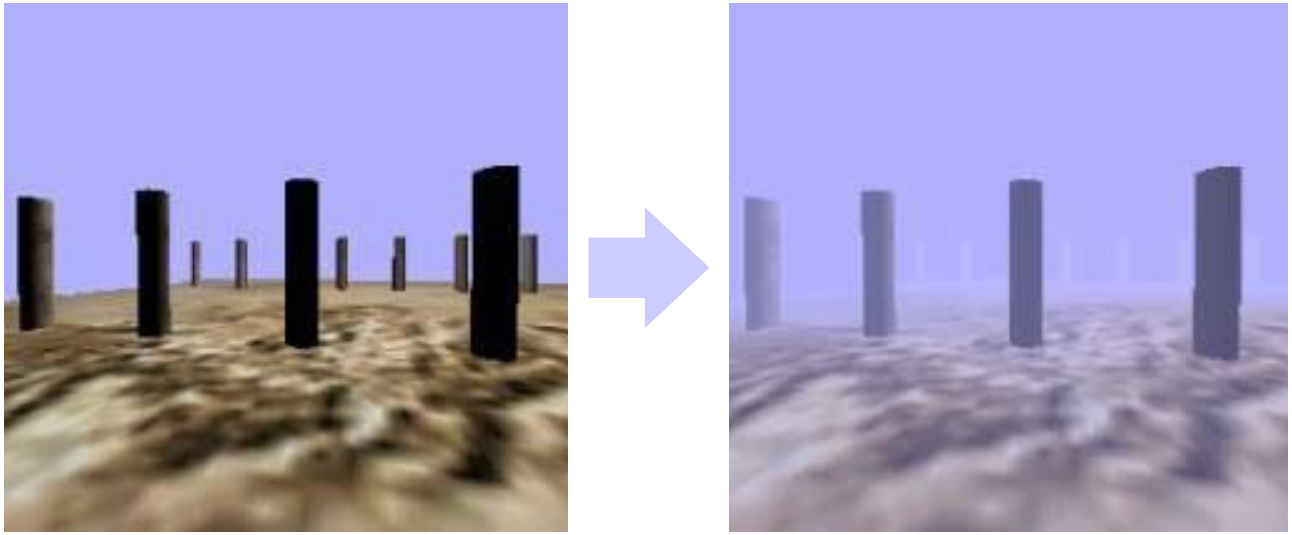
  → Distance Fog

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



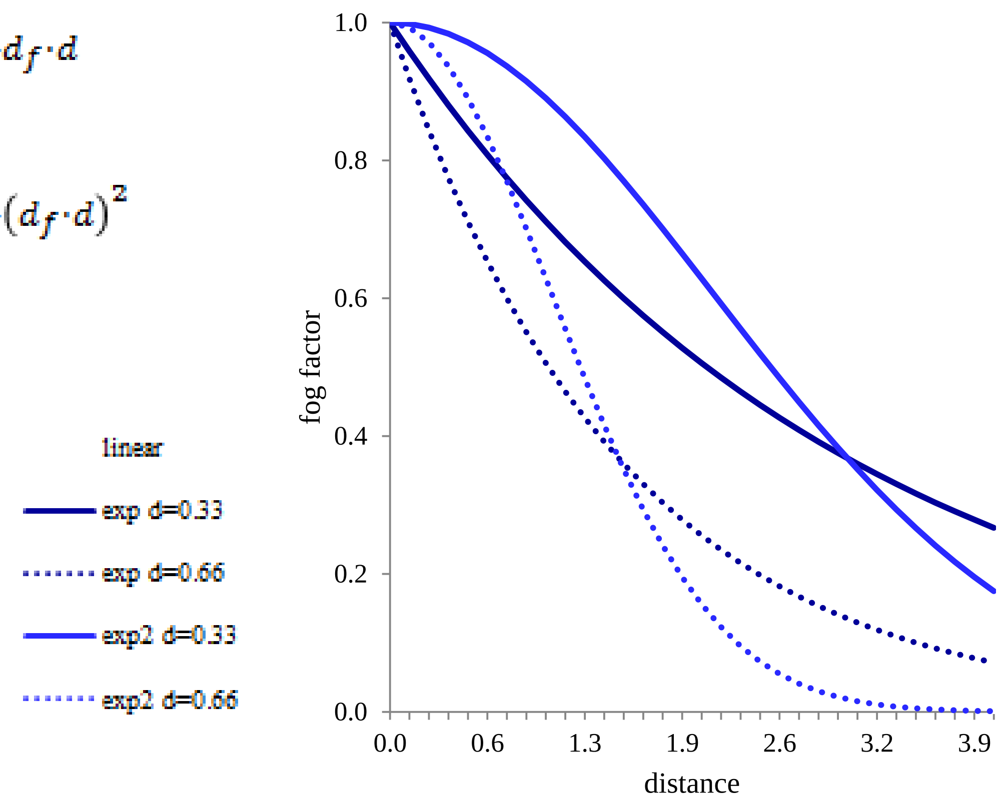
Fog

- Linear fog: $f = \frac{d_{end} - d}{d_{end} - d_{start}}$

- Exponential fog: $f = e^{-d_f \cdot d}$

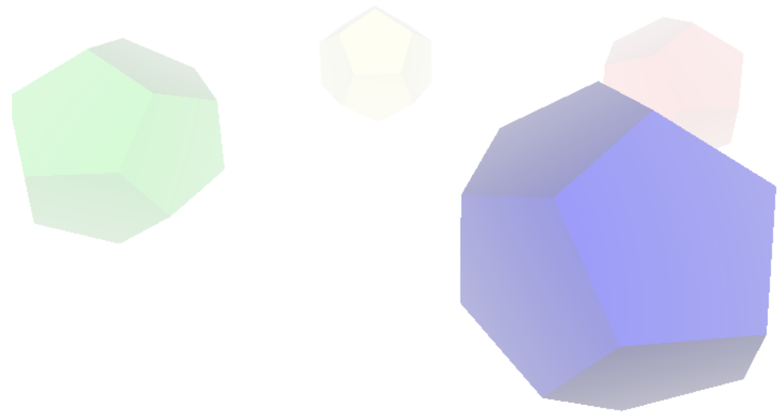
- Squared exponential fog: $f = e^{-(d_f \cdot d)^2}$

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



Fog

```
#version 330

#include <framework/utils/GLSL/camera>

uniform vec3 c_d;
uniform vec3 c_f;
uniform float d_f;

in vec3 p;
in vec3 normal;

layout(location = 0) out vec4 color;

void main()
{
    vec3 v = camera.position - p;
    float d = length(v);
    vec3 c_s = ...;

    float f = exp(-d_f * d);

    color = vec4(f * c_s + (1.0f - f) * c_f, 1.0f);
}
```



Post Processing Effects

1. Render scene into textures
   – Color
   – Depth
   – …
2. Render screen-filling primitives
   – Fragment shader samples rendered textures
   – Can implement
      • Image filters
      • Color transformations
      • …

Depth of Field

- Simulate 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

Depth of Field

- Guide the user‘s attention towards something

Depth of Field

- Effect does not occur with small apertures
- CG mostly uses pinhole cameras
  - Infinitely small aperture
- Simulating depth of field (DoF):
  - Adapt camera model
    - Not possible using standard OpenGL pipeline
  - Approximate DoF by blurring image based on depth buffer values

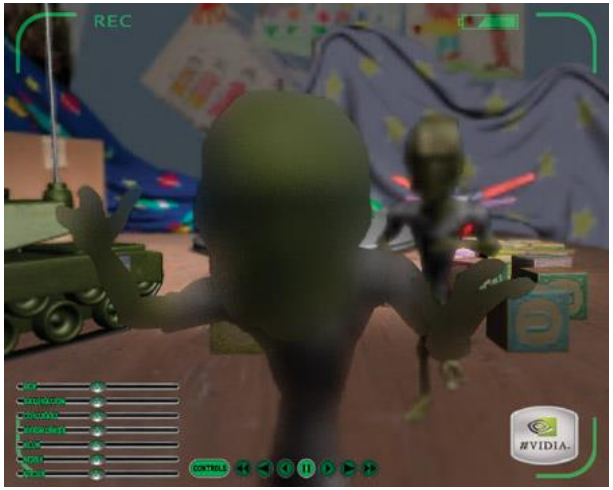
Depth of Field

1. Render scene to texture
2. Draw fullscreen quad
   - Compute the circle of confusion (CoC)
     - Based on the scene depth buffer
   - Blur the image using convolution or random sampling
     - Window size depends on the CoC

Depth of Field -- Artifacts

- Color bleeding

- Discontinuities at silhouettes

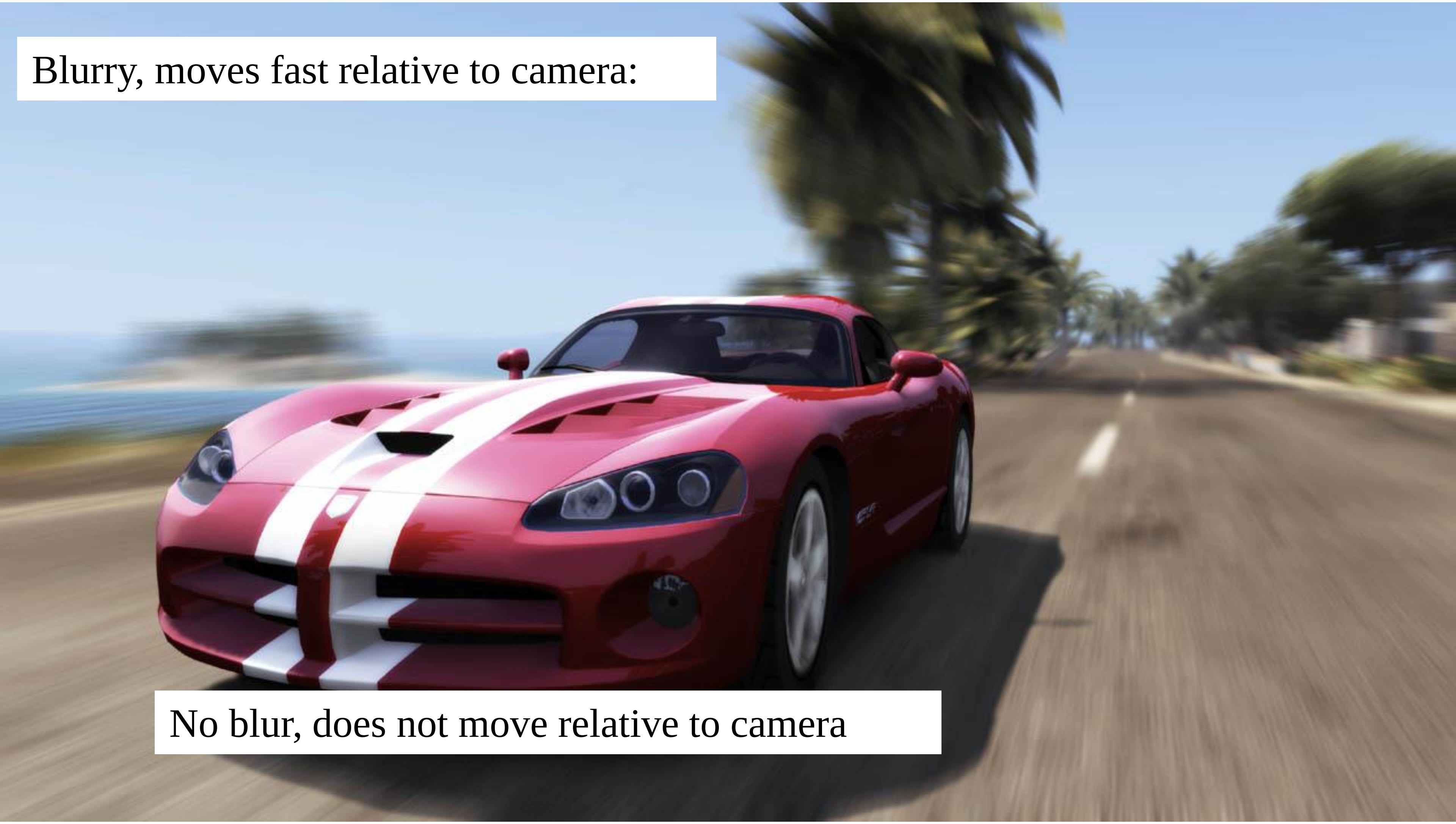


- Solutions:
  - Use bilateral filter
  - Advanced techniques
    - Diffusion based methods
    - ...

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

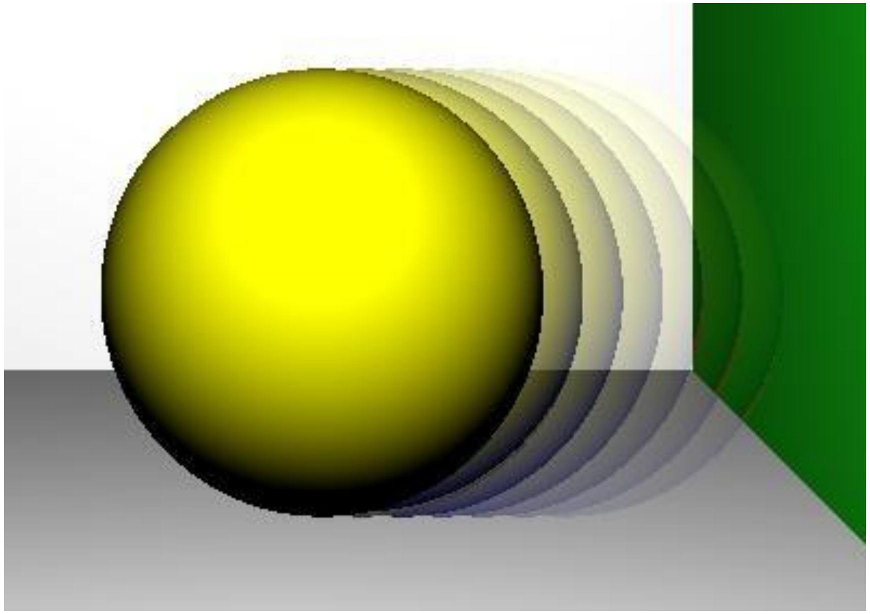

Blurry, moves fast relative to camera:
No blur, does not move relative to camera

Motion Blur

Motion Blur

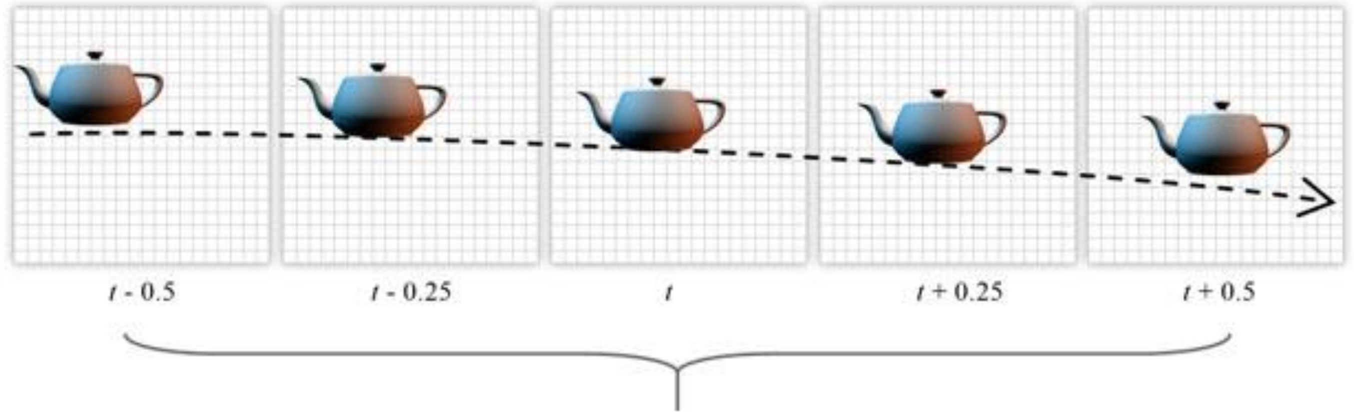
- Continuous vs Discrete

Correct, continuous MB

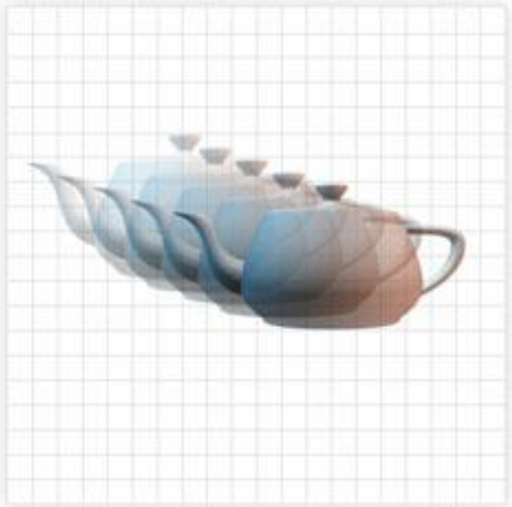


Approximated, discrete MB

Motion Blur – Discrete Methods



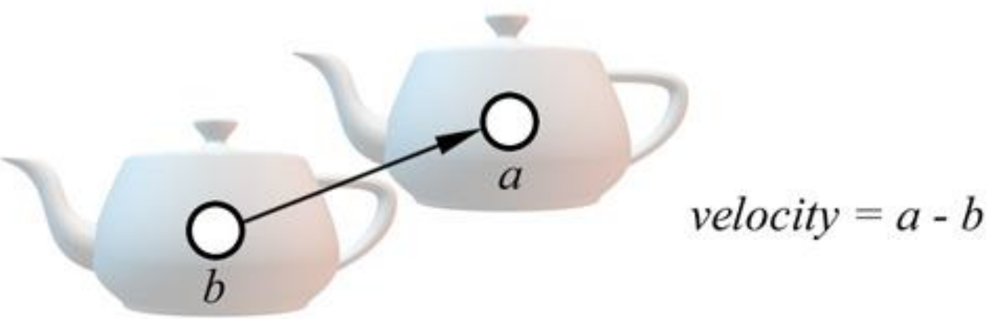
- Simplest method
  - Render object at past positions with varying transparency
  - Object needs to be rendered multiple times
- Image Space Motion Blur
  - Render object to buffer
  - Copy buffer with varying transparency
  - More efficient



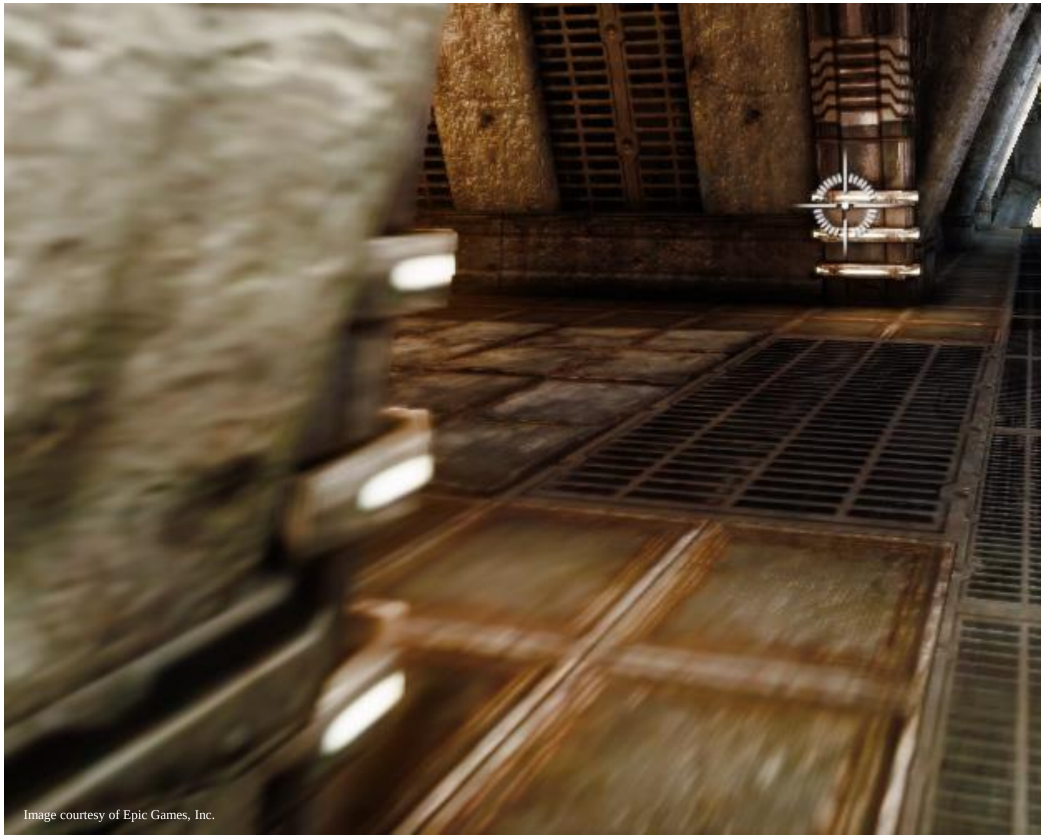
Continuous Motion Blur

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



Continuous Motion Blur – Examples

Continuous Motion Blur – Examples

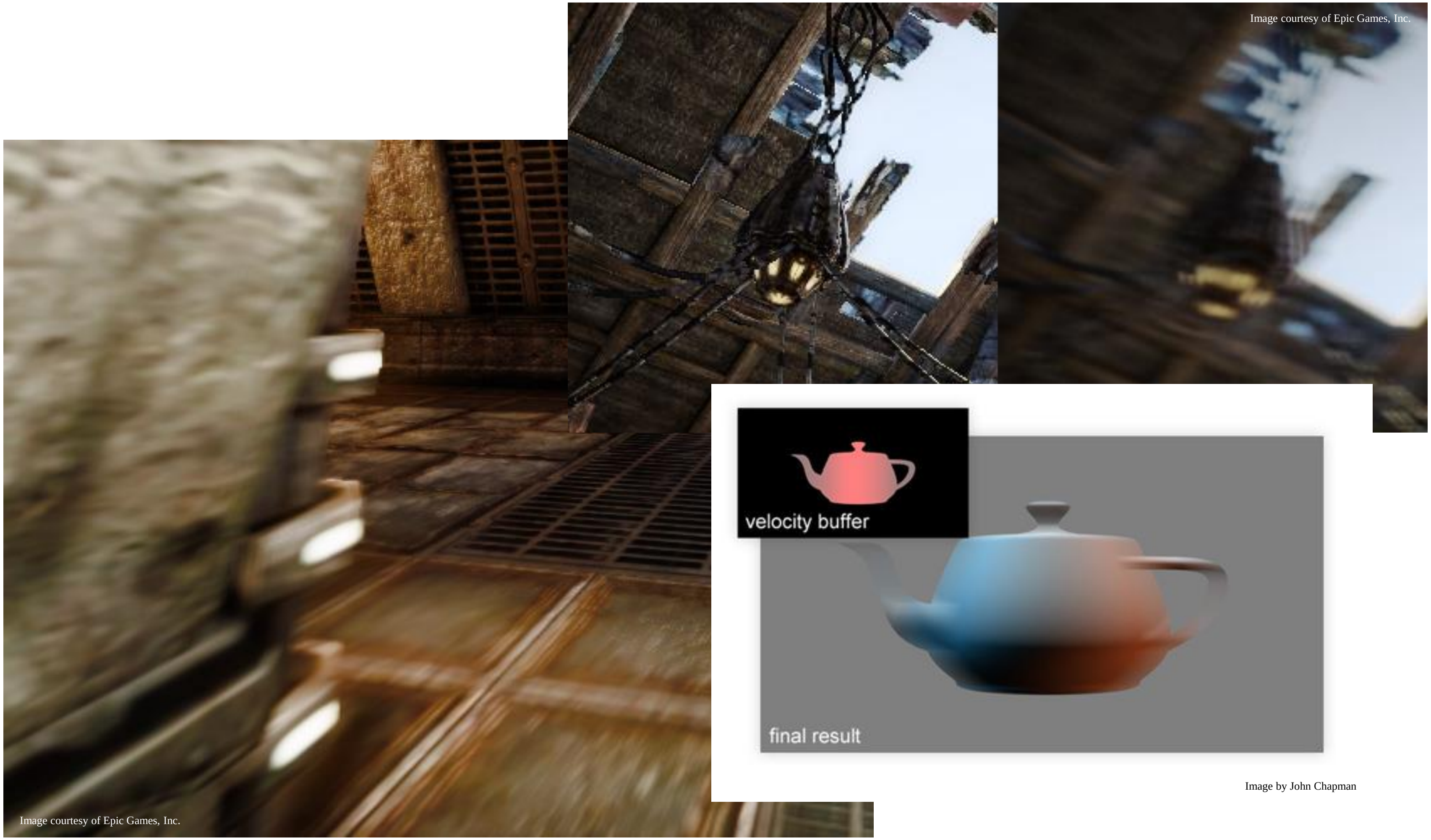
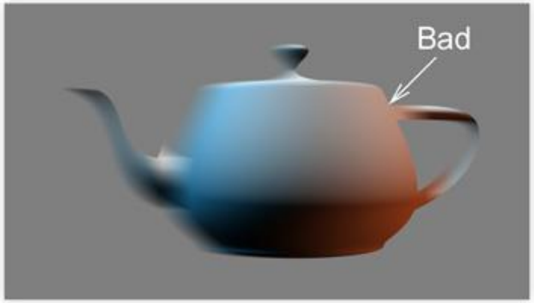


Image courtesy of Epic Games, Inc.
Image by John Chapman
Image courtesy of Epic Games, Inc.

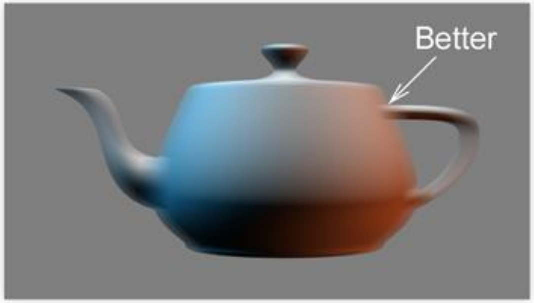
Continuous Motion Blur – Artifacts

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

- Discontinuities at silhouettes

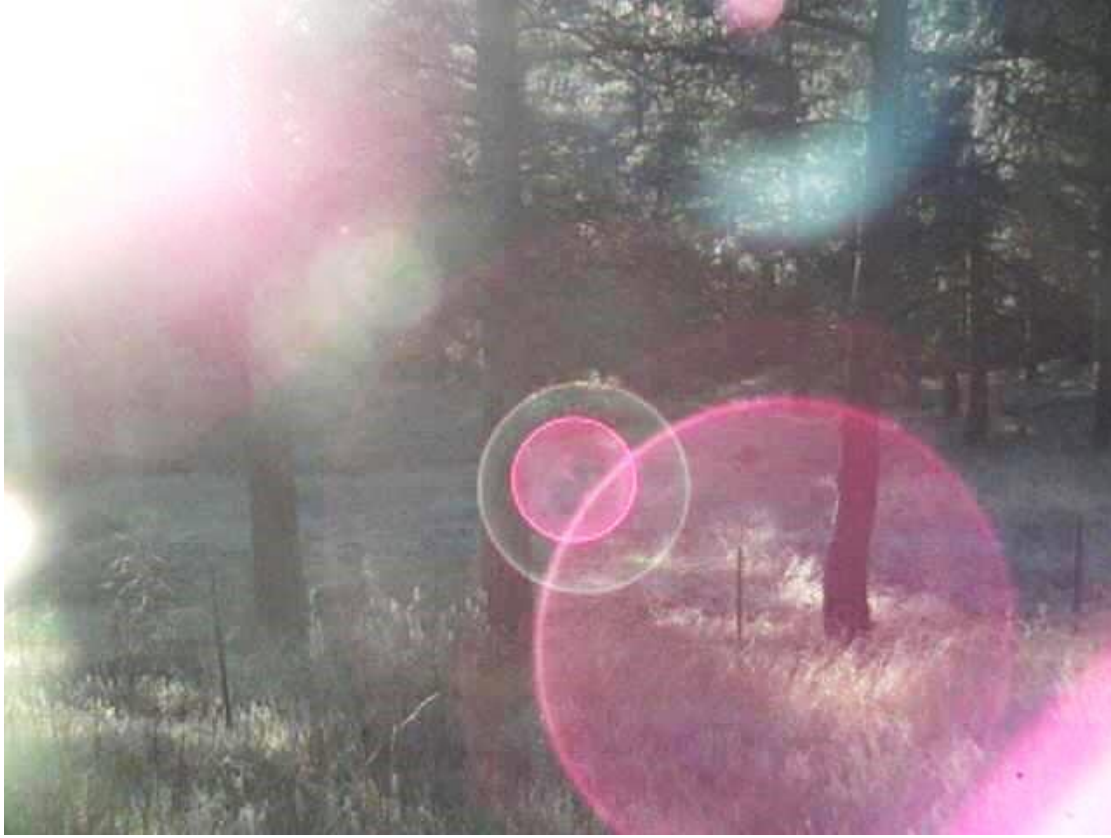


Blur not centred



Blur centred

Lens Flare

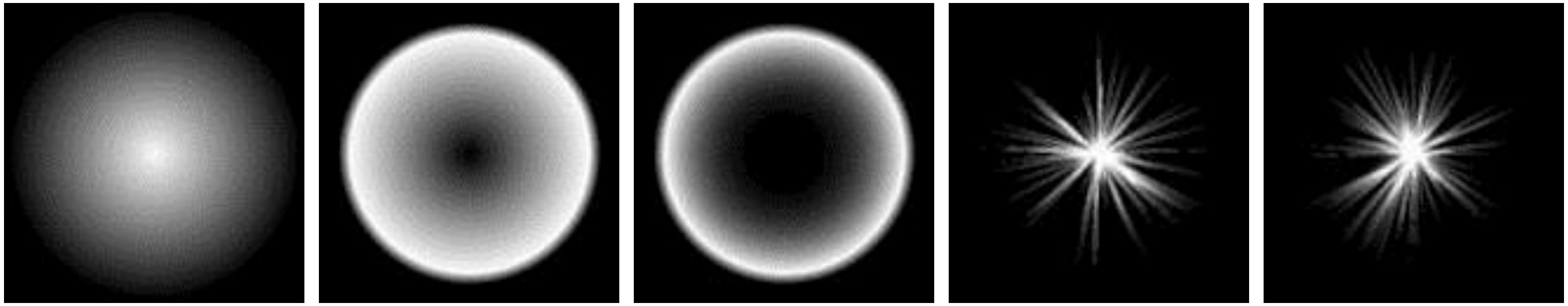


- A shortcoming of cameras that photographers try to avoid
- However: looks realistic and fancy
- Effect occurs inside lens system
  - Always on top
- Happens when light source inside image
- Star, ring or hexagonal shapes

Lens Flare

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 textured quads and alpha blending



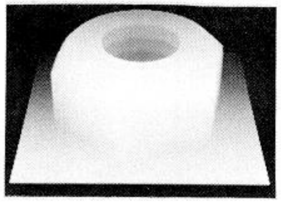
Lens Flare Rendering

- Don‘t overdo it!

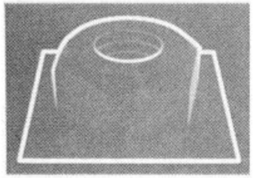
Non-Photorealistic Rendering

- Emphasizes object edges and silhouettes

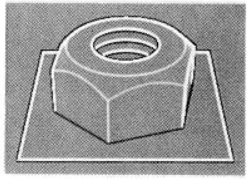
- Either from z-buffer or in object space
- Profile: 1^{st} order differential operator (e.g., Sobel)
- Internal: 2^{nd} order differential operator (e.g., Laplace)
- …



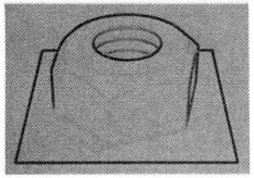
depth image



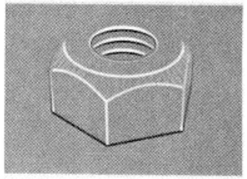
1st order differential



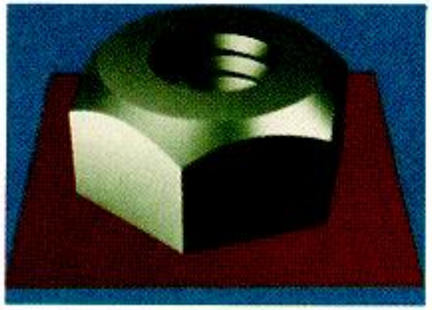
2st order differential

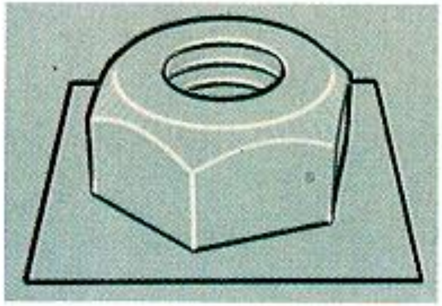


profile image



internal edge image

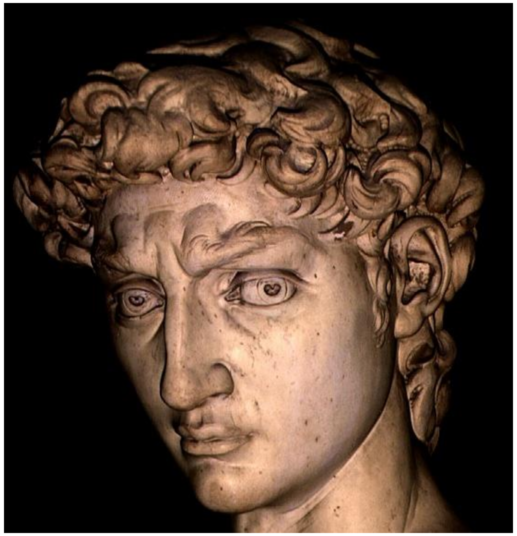




Saito 90

Line Classification

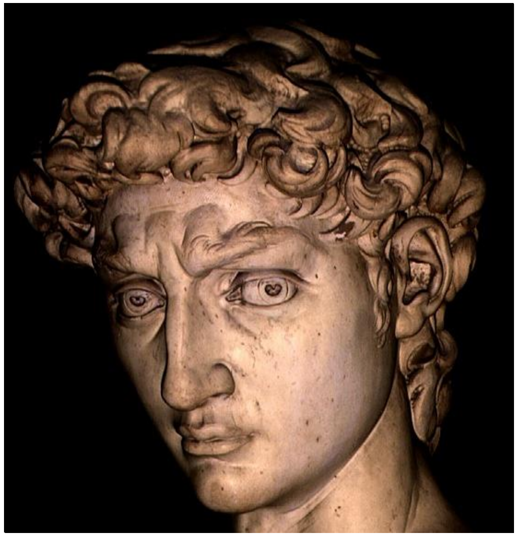
- Silhouette
  - Contour (Outer Silhouette)



Rusinkiewicz 05

Line Classification

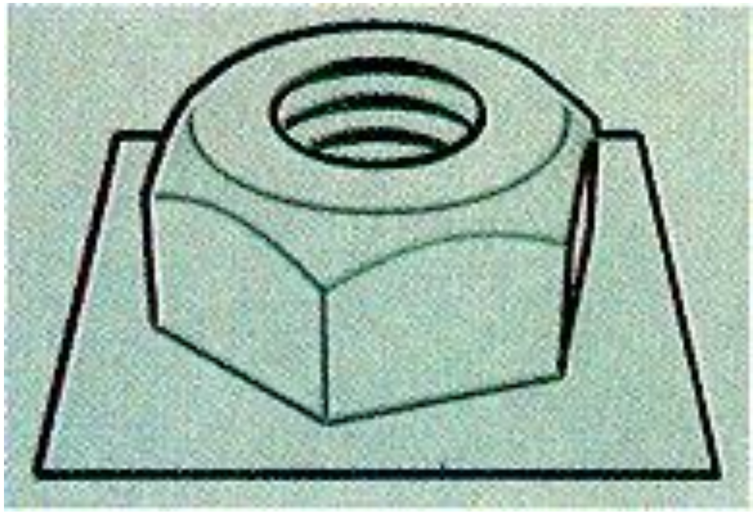
- Silhouette
  - Contour (Outer Silhouette)
  - **Occluding** contour (Inner Silhouette)

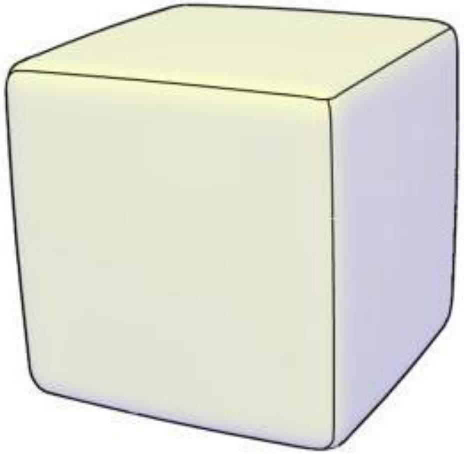


Rusinkiewicz 05

Line Classification

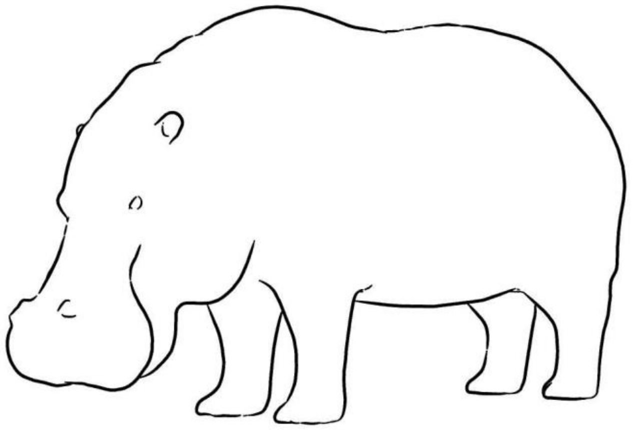
- Creases
  - Local maxima and minima of curvature
  - Ridges / Valleys

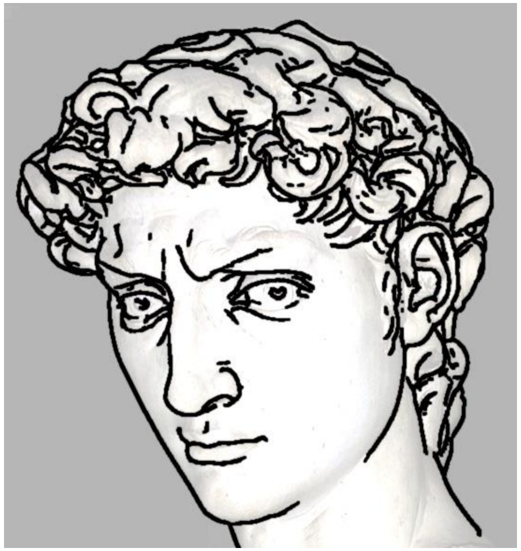


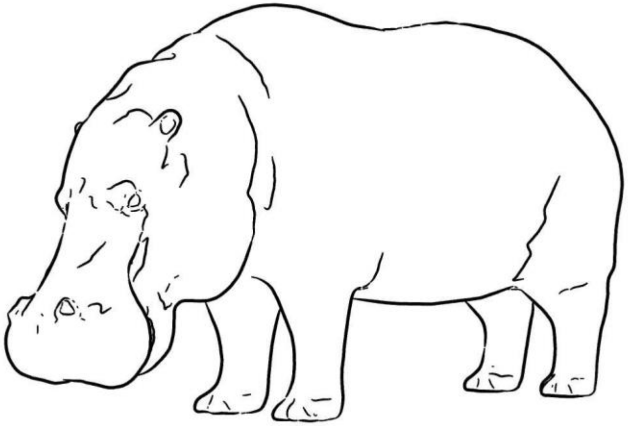


Line Classification

- Suggestive Contours
  - "Almost contours"
  - Points that become contours in nearby views

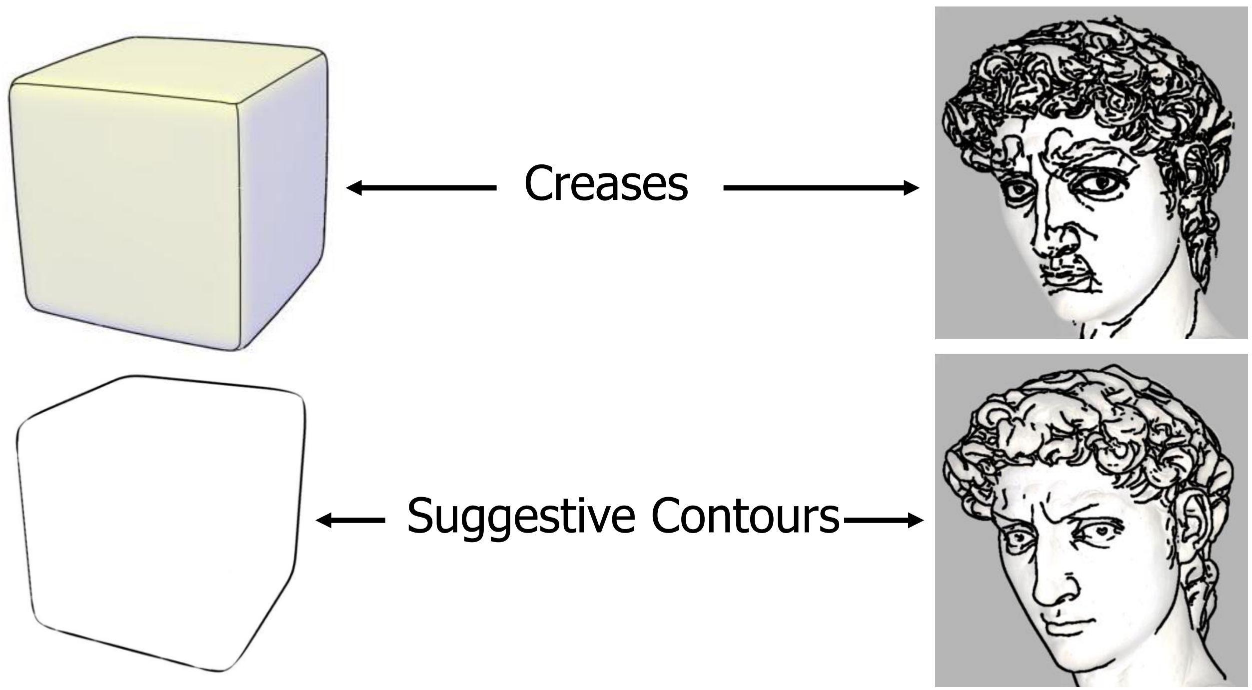






Line Classification

- Which Lines to Draw?
- Some objects do not have suggestive contours

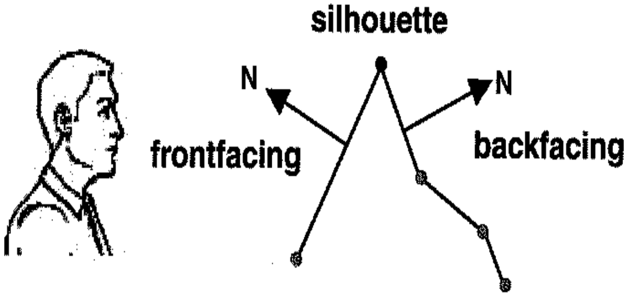


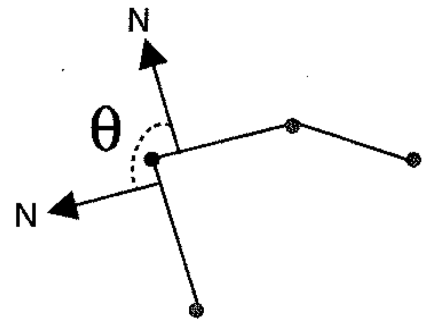
Rusinkiewicz 05

=> No universal rule which lines to draw <=

Line Detection in Object Space

- Silhouette
  - Points at which n_dot_v = 0
- Creases
  - Points at which angle > threshold

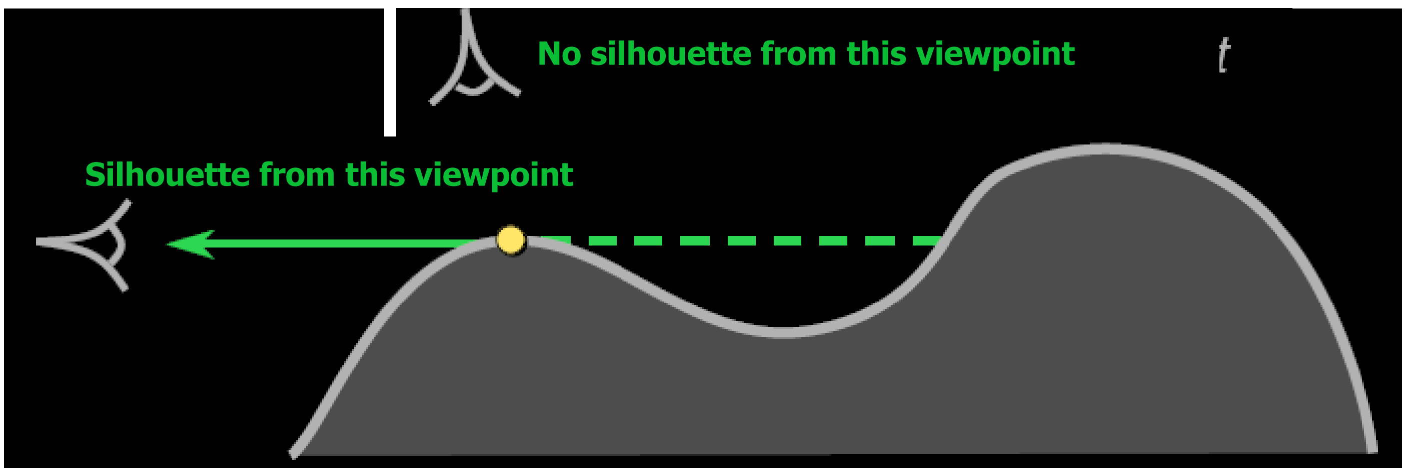




Gooche 01

Line Detection in Object Space

- Silhouette
  - View dependent
  - Online computation
- Creases
  - View independent
  - Pre-processing



Rusinkiewicz 05

Qestions?