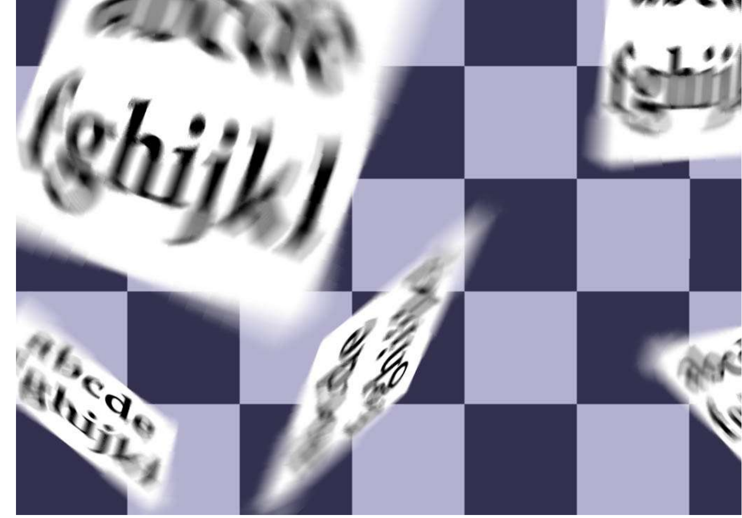
Motion Blur for Textures by Means of Anisotropic Filtering

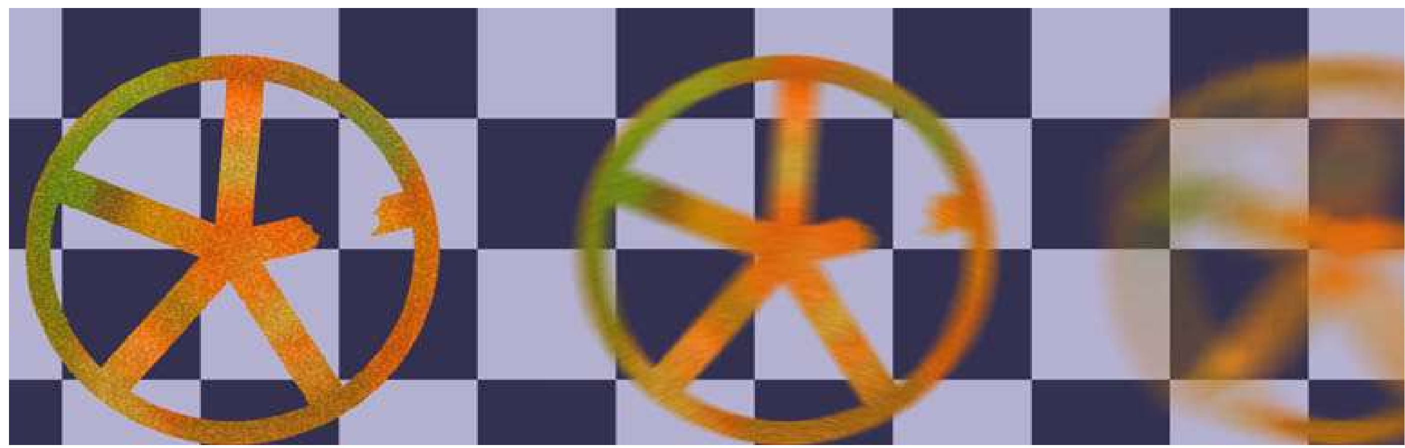
Jörn Loviscach
Hochschule Bremen

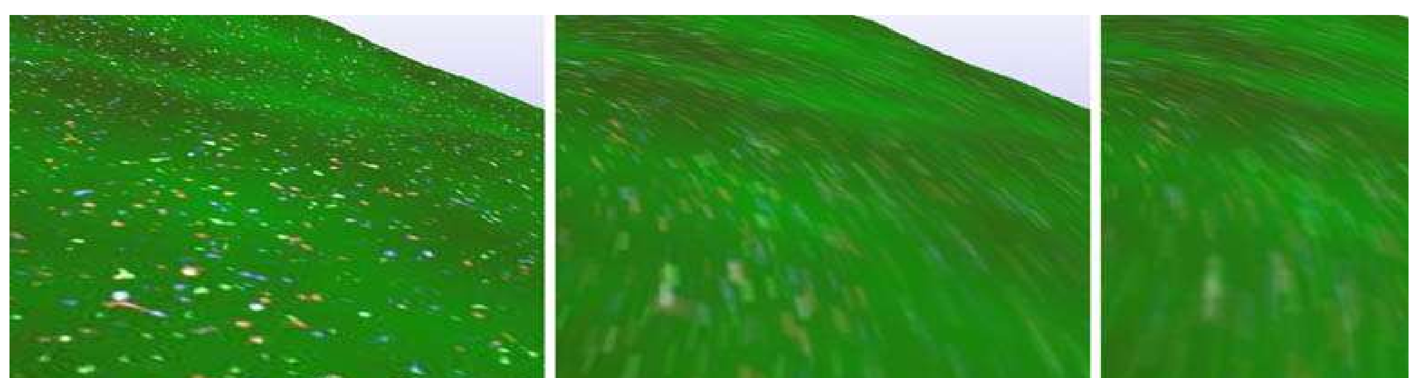
Introduction: Motion Blur



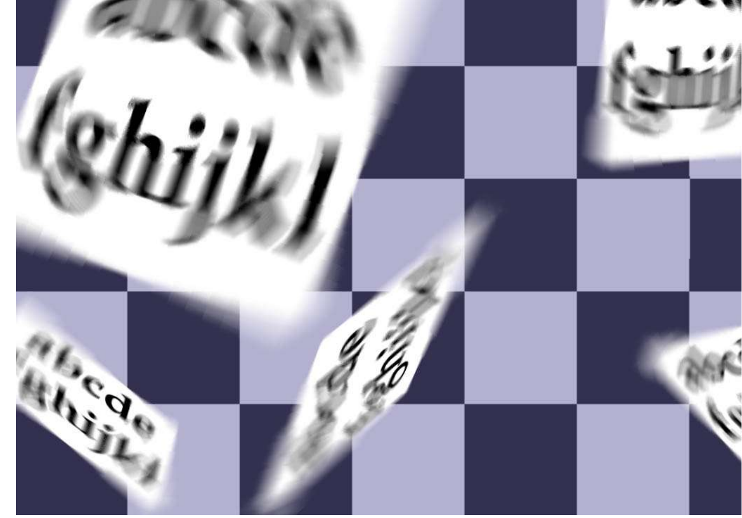
Motion blur is often needed
for (nearly) flat objects
with or without cookie-cutting:

- terrains
- billboards
- spoke wheels
- sword blades
- air-screws
- ...



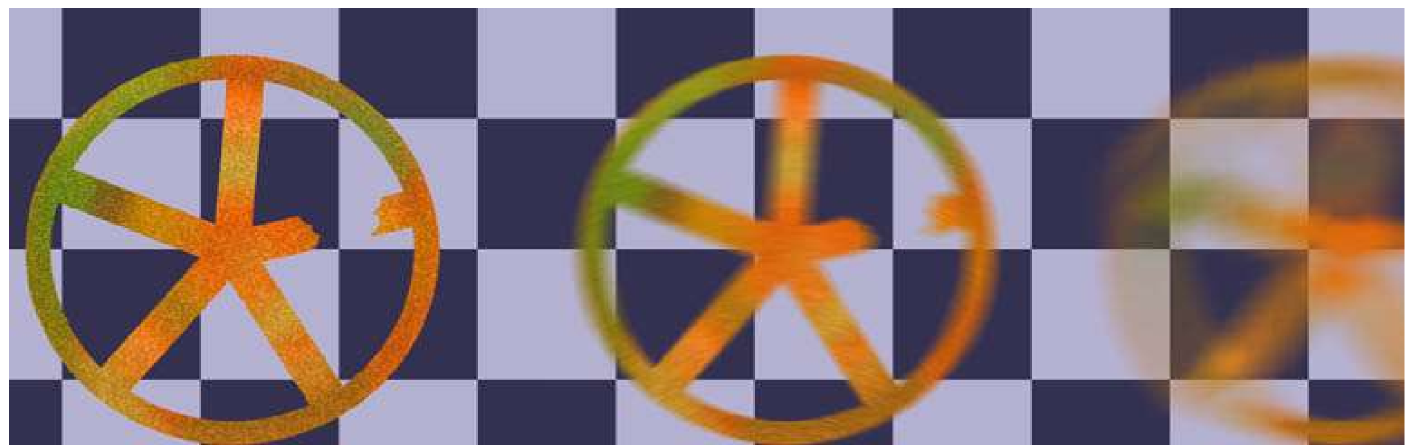


Introduction: Motion Blur

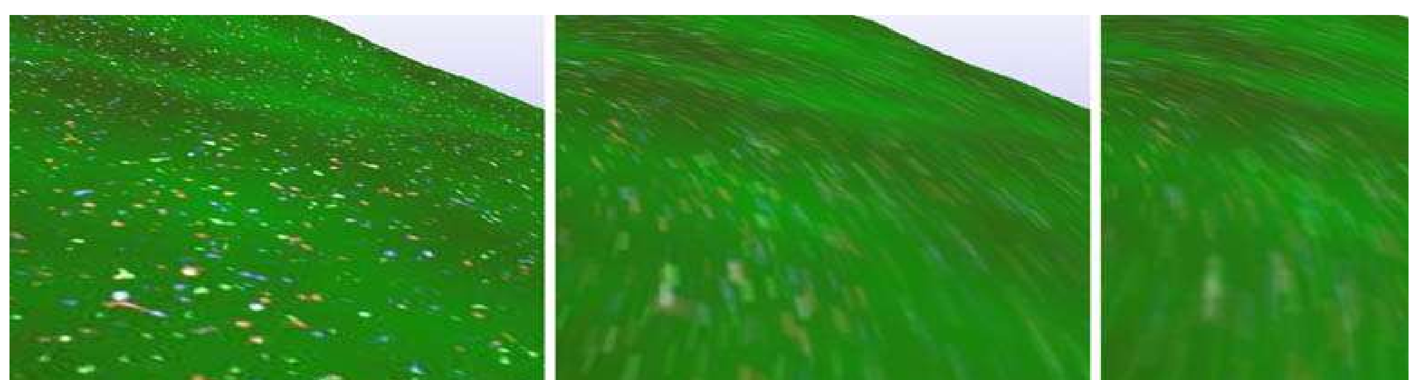


Motion blur is often needed
for (nearly) flat objects
with or without cookie-cutting:

- terrains
- billboards
- spoke wheels
- sword blades
- air-screws
- ...

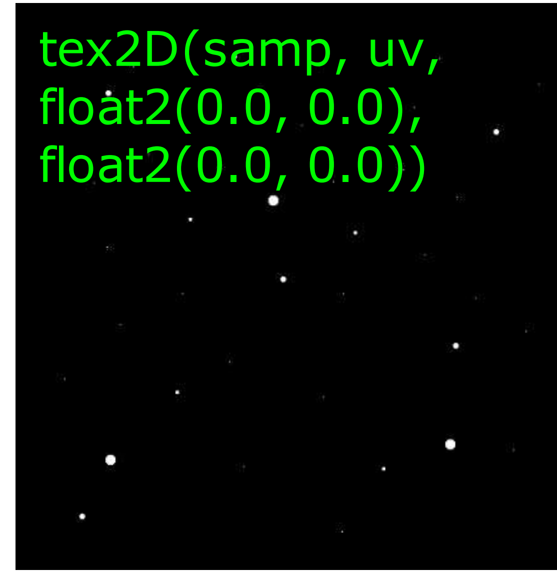


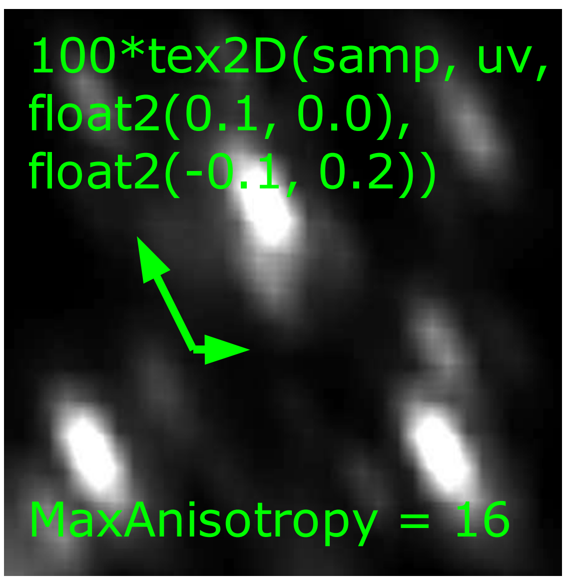
Apply motion blur only to texture!

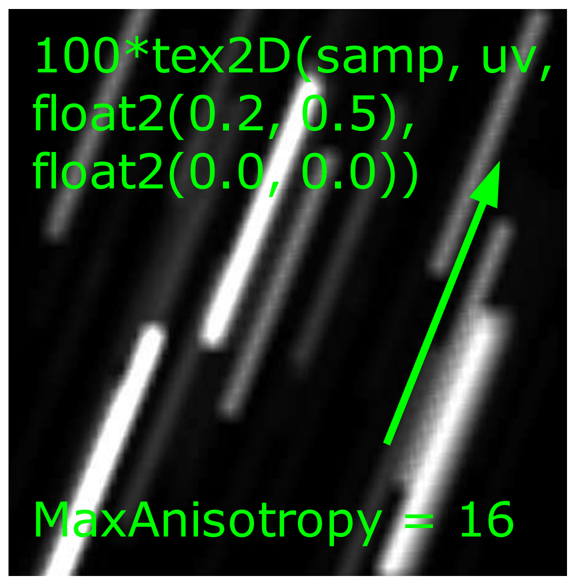


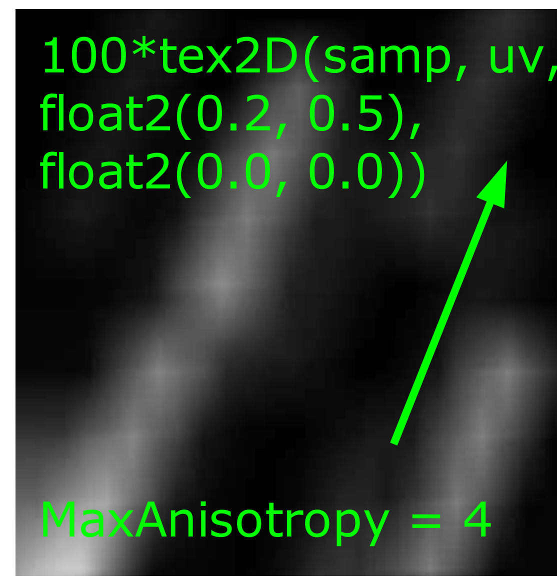
Introduction: tex2D

tex2D instruction of HLSL:
footprint for anisotropic filtering
may be passed as parameter



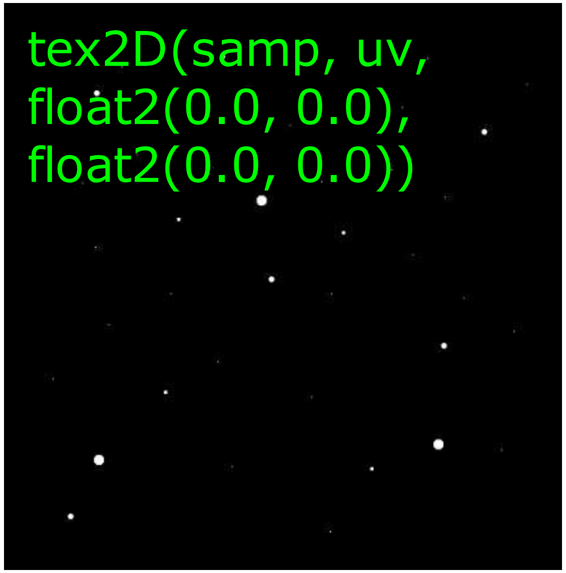


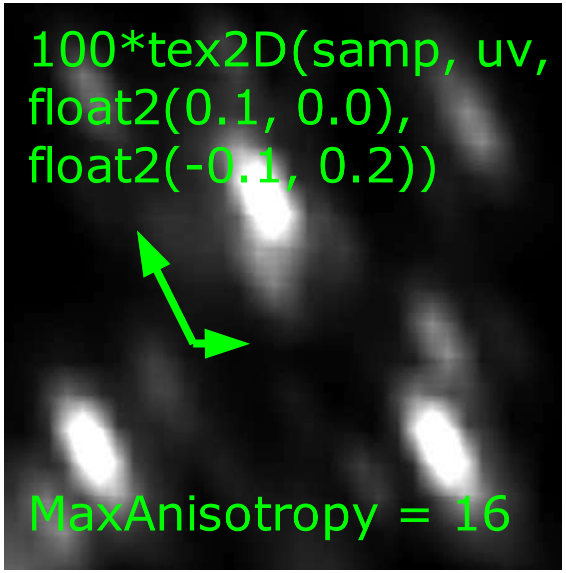


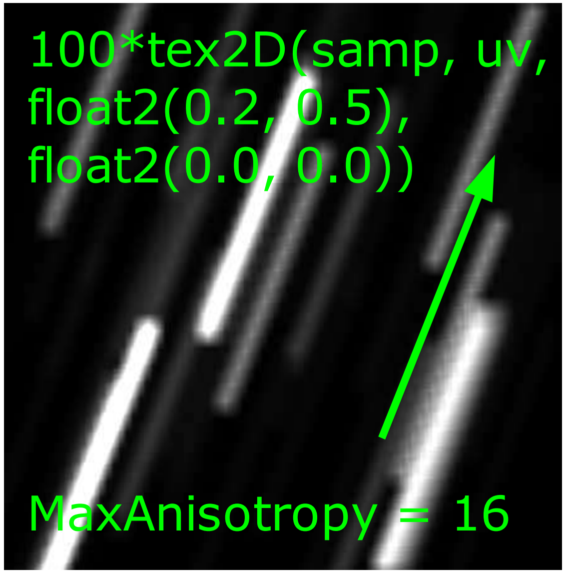


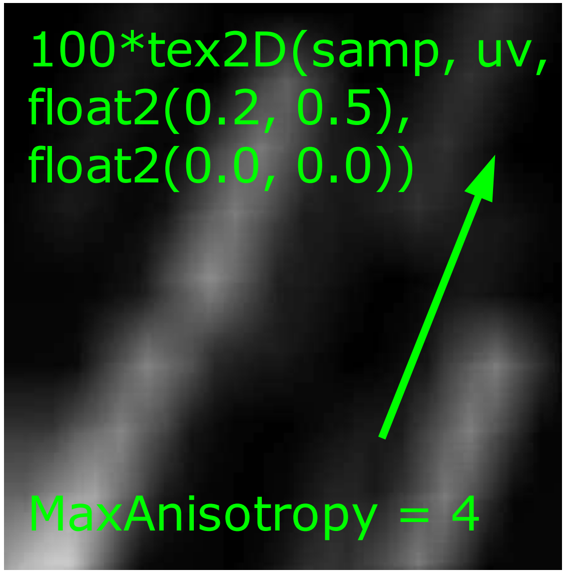
Introduction: tex2D

tex2D instruction of HLSL:
footprint for anisotropic filtering
may be passed as parameter









Motion blur with tex2D: Examples

Outline

- Introduction
- Related work
- Motion blur in texture coordinates
- Combining spatial and temporal aniosotropy
- Shader-based implementation
- Results
- Conclusion and Outlook

Related Work

Standard technique for real-time motion blur of 3D objects, including non-flat ones:

Extrude geometry along direction of motion; optionally apply motion blur to texture by temporal supersampling. [e.g., Green 2003]

Related Work

Standard technique for real-time motion blur of 3D objects, including non-flat ones:

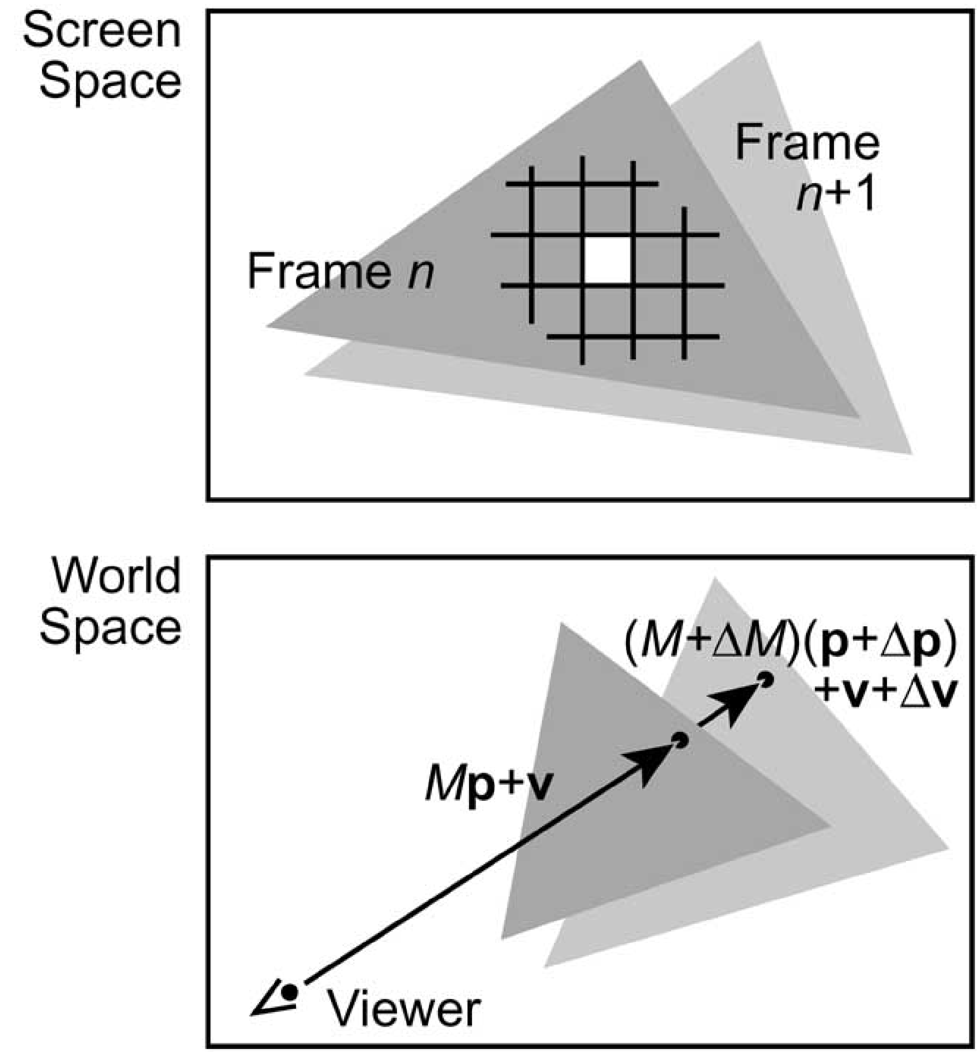
Extrude geometry along direction of motion; optionally apply motion blur to texture by temporal supersampling.
[e.g., Green 2003]

Texture-based, but limited to a small set of precomputed directions and speeds:

Pre-blurred textures for terrains [Hargreaves 2004]

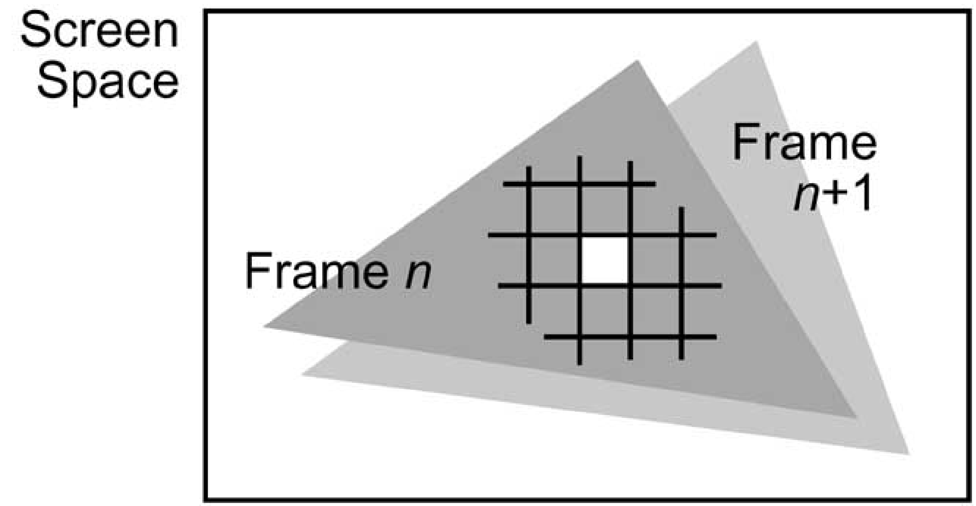
Motion Blur in Texture Coordinates

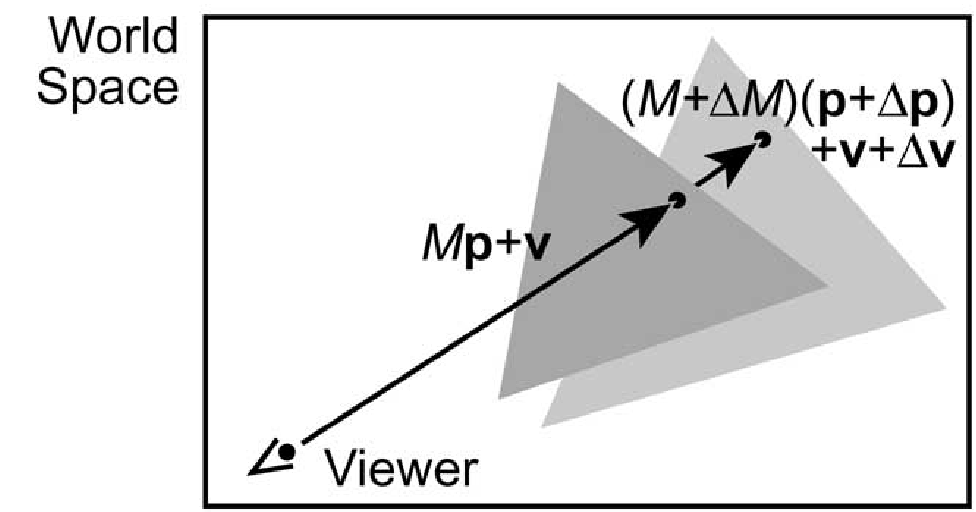
Determine temporal change of uv coordinates for a fixed screen pixel.



Motion Blur in Texture Coordinates

Determine temporal change of uv coordinates for a fixed screen pixel.





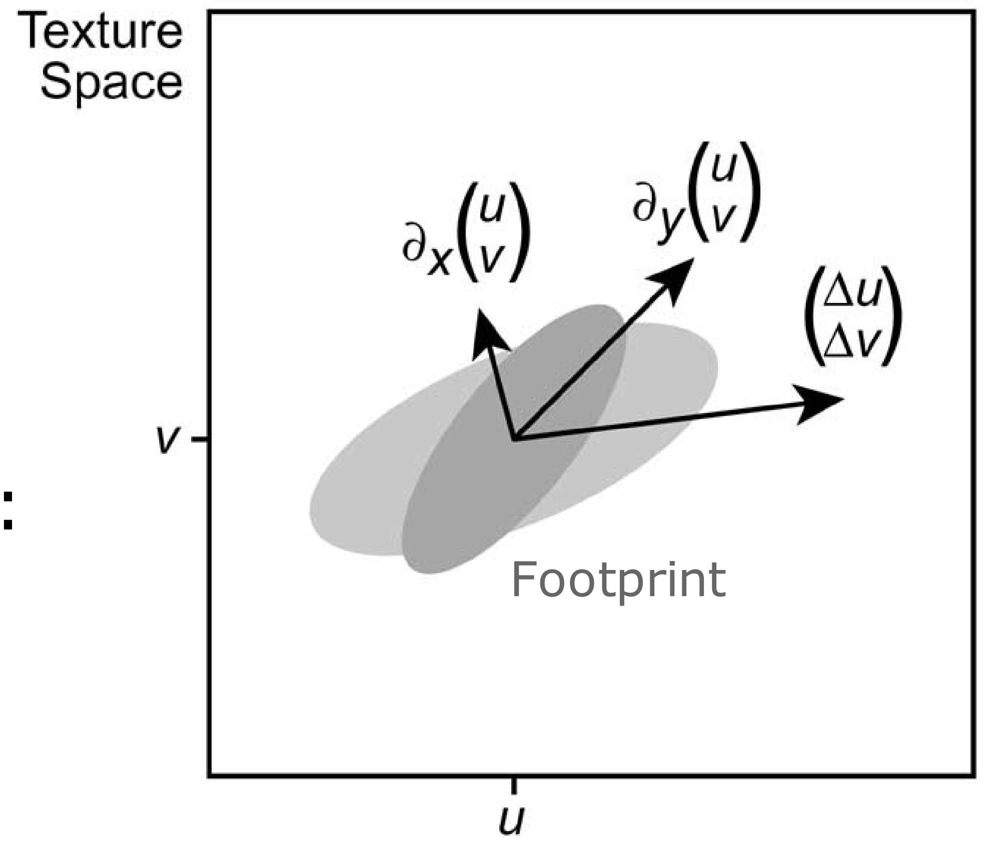
$$\mathbf{r} := M\mathbf{p} + \mathbf{v}$$

$$\mathbf{s} := \Delta M\mathbf{p} + \Delta\mathbf{v}$$

$$\begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} = -UM^{-1}\mathbf{s} + \frac{(M^{-1\mathrm{T}}\mathbf{n}) \cdot \mathbf{s}}{(M^{-1\mathrm{T}}\mathbf{n}) \cdot \mathbf{r}} UM^{-1}\mathbf{r}$$

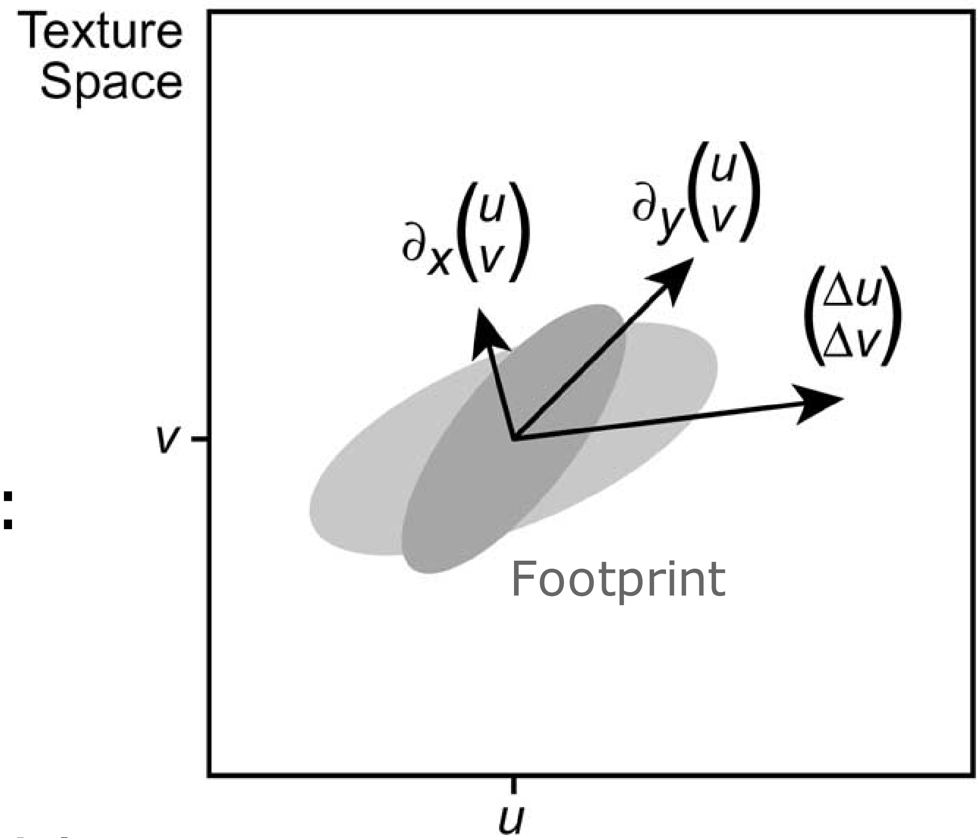
Spatial and Temporal Aniosotropy

- Incorporate spatial anisotropic filtering, too
- ddx, ddy instructions: pre-image of screen pixel
- Combine with motion blur: unified footprint to average over



Spatial and Temporal Aniosotropy

- Incorporate spatial anisotropic filtering, too
- ddx, ddy instructions: pre-image of screen pixel
- Combine with motion blur: unified footprint to average over



Model: α, β, γ random variables

$$\begin{pmatrix}u\\v\end{pmatrix}+\alpha\partial_x\begin{pmatrix}u\\v\end{pmatrix}+\beta\partial_y\begin{pmatrix}u\\v\end{pmatrix}+\gamma\begin{pmatrix}\Delta u\\\Delta v\end{pmatrix}$$

For tex2D convert into *e*, *f*, *g*, *h*:

$$\begin{pmatrix}u\\v\end{pmatrix}+\alpha\begin{pmatrix}e\\f\end{pmatrix}+\beta\begin{pmatrix}g\\h\end{pmatrix}$$

Shader-Based Implementation

Variant 1: heavily pixel-based

- 👍 works with large polygons
- 👎 pixel shader of 30 instructions

$$\begin{pmatrix}\Delta u\\ \Delta v\end{pmatrix} = -UM^{-1}\mathbf{s} + \frac{(M^{-1\mathrm{T}}\mathbf{n})\cdot\mathbf{s}}{(M^{-1\mathrm{T}}\mathbf{n})\cdot\mathbf{r}}UM^{-1}\mathbf{r}$$

Vertex shader
Pixel shader

Shader-Based Implementation

Variant 1: heavily pixel-based

- 👍 works with large polygons
- 👎 pixel shader of 30 instructions

$$\begin{pmatrix}\Delta u\\ \Delta v\end{pmatrix} = -UM^{-1}\mathbf{s} + \frac{(M^{-1\mathrm{T}}\mathbf{n})\cdot\mathbf{s}}{(M^{-1\mathrm{T}}\mathbf{n})\cdot\mathbf{r}}UM^{-1}\mathbf{r}$$

Vertex shader
Pixel shader

Variant 2: heavily vertex-based

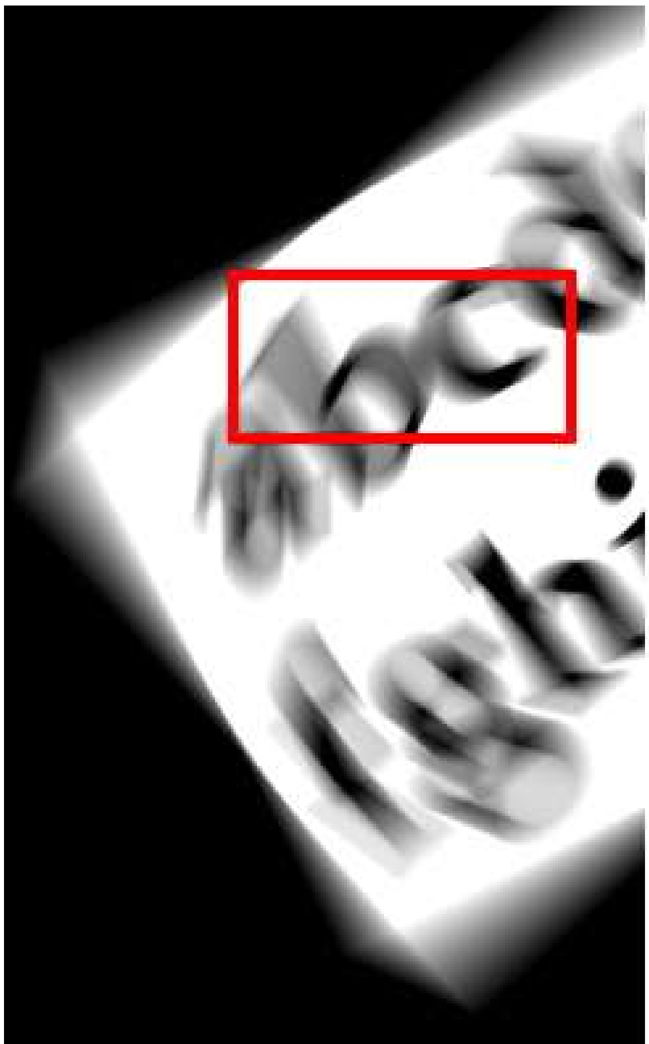
- 👍 short pixel shader
- 👎 requires small polygons

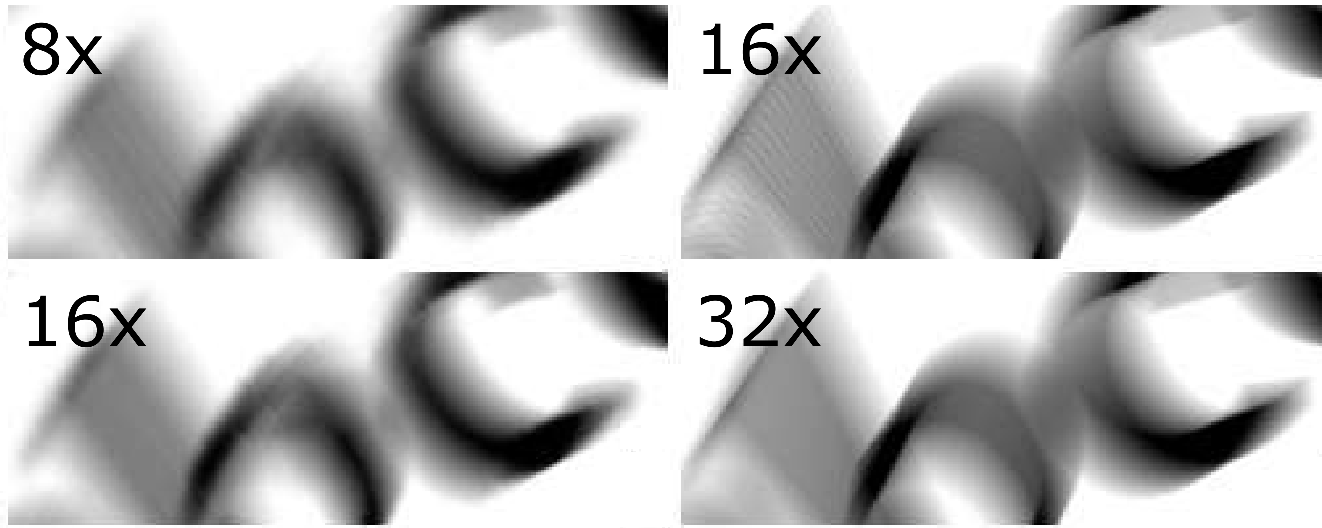
Basic idea: Compute footprint in vertex shader
Problems that had to be solved:

- How to interpolate the footprint?
- No ddx/ddy instructions in vertex shader

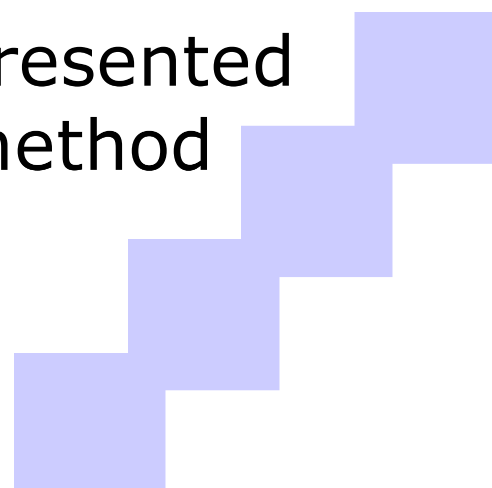
Results

Artifacts look different from standard multiple rendering:

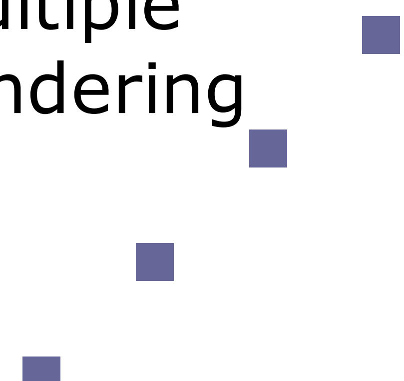




Presented method

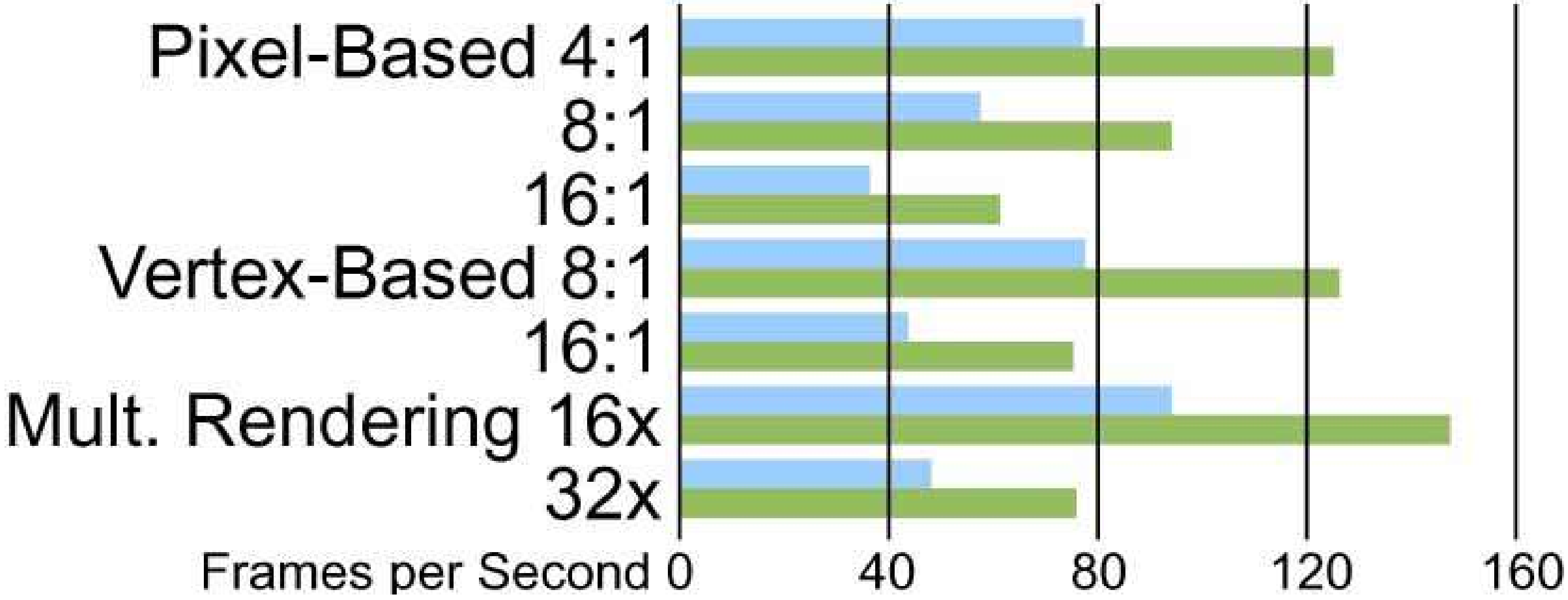


Multiple rendering



Results

Speed comparable to that
of a very fast and non-general
implementation of multiple rendering:



Flight over terrain
20,000 triangles

Spinning sphere
528 triangles

Conclusion and Outlook

Features of the proposed method:

- Natural unification of spatial and temporal filtering
- One pass; no deep color buffer needed
- Efficiency still close to multiple rendering.
  Future optimizations in anisotropic filtering?
- Covers some types of 3D objects that are vital for real-time applications


THURBO
SBB CFF FFS


THURBO

Conclusion and Outlook

Features of the proposed method:

- Natural unification of spatial and temporal filtering
- One pass; no deep color buffer needed
- Efficiency still close to multiple rendering. Future optimizations in anisotropic filtering?
- Covers some types of 3D objects that are vital for real-time applications

Possible future work:

- Include lighting
- Handle time-varying deformation
- Combine with methods based on extrusion