

Splat/Mesh Blending, Perspective Rasterization and Transparency for Point-Based Rendering

Gaël Guennebaud

Loïc Barthe, Mathias Paulin

IRIT – UPS – CNRS

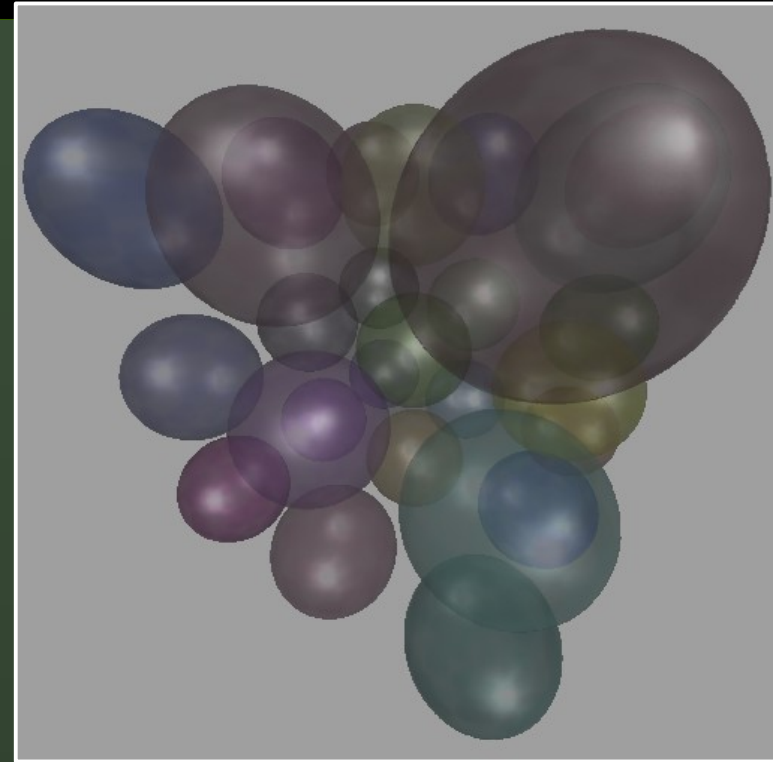
TOULOUSE – FRANCE

<http://www.irit.fr/~Gael.Guennebaud/>

Approximate depth-peeling for Transparency

Transparency via depth-peeling

- Standard depth-peeling
 - advantages:
 - no pre-process
 - no sort
 - suitable for per-pixel lighting
 - drawback:
 - may requires several rendering passes
- => “approximate depth-peeling” ?

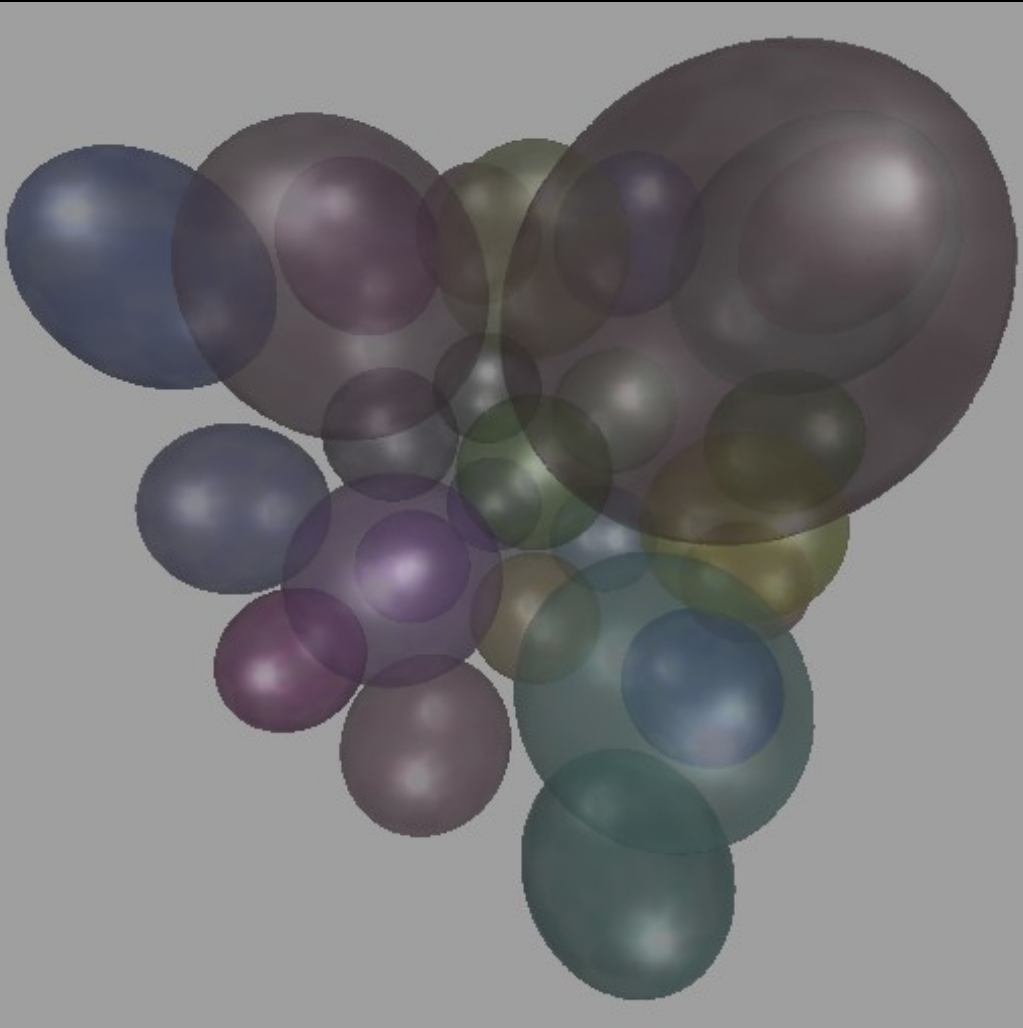


Approximate depth-peeling

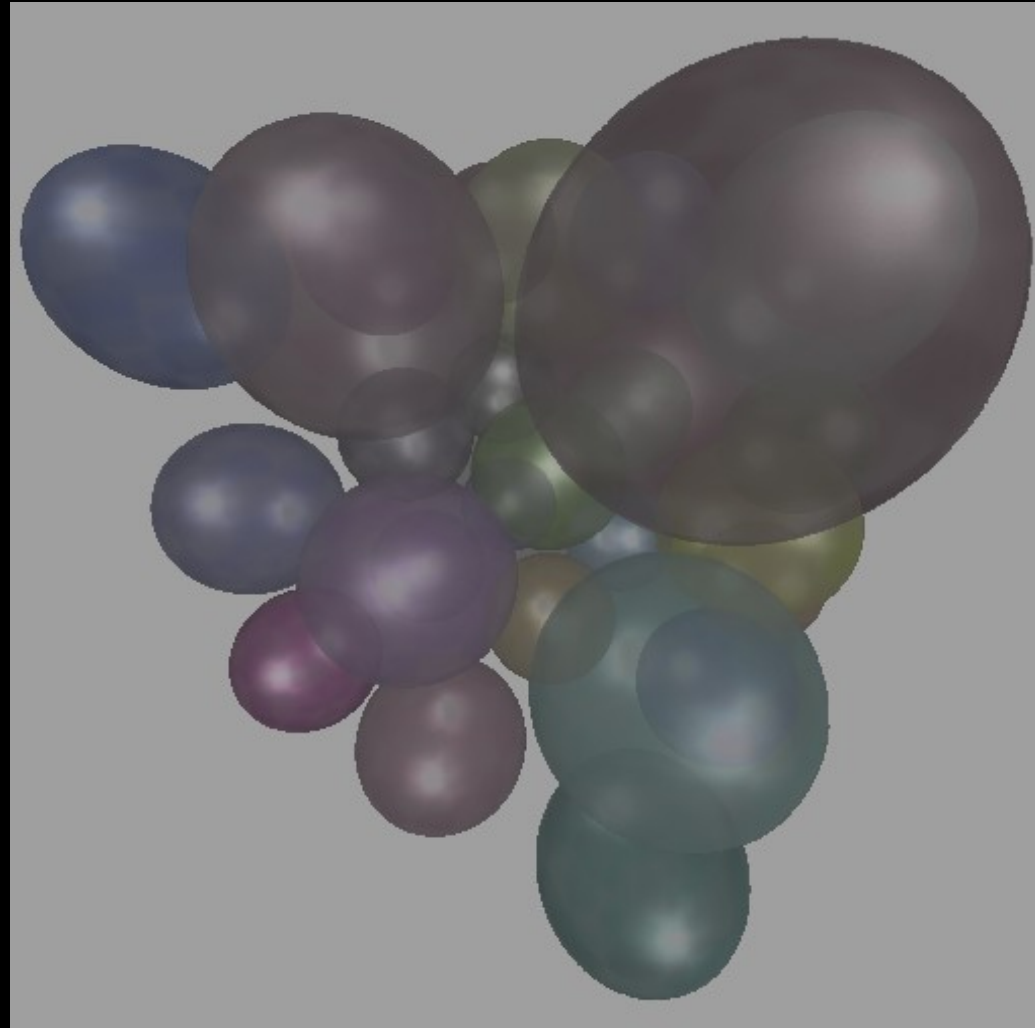
- Idea:
 - bound the number of rendering passes
 - + approximate blending for the last layer

- blending heuristic:
$$C(\mathbf{x}) = \sum_i \phi'_i(\mathbf{x}) \alpha_i C_i$$
- no deferred shading

Approximate depth-peeling (results with 2 layers)

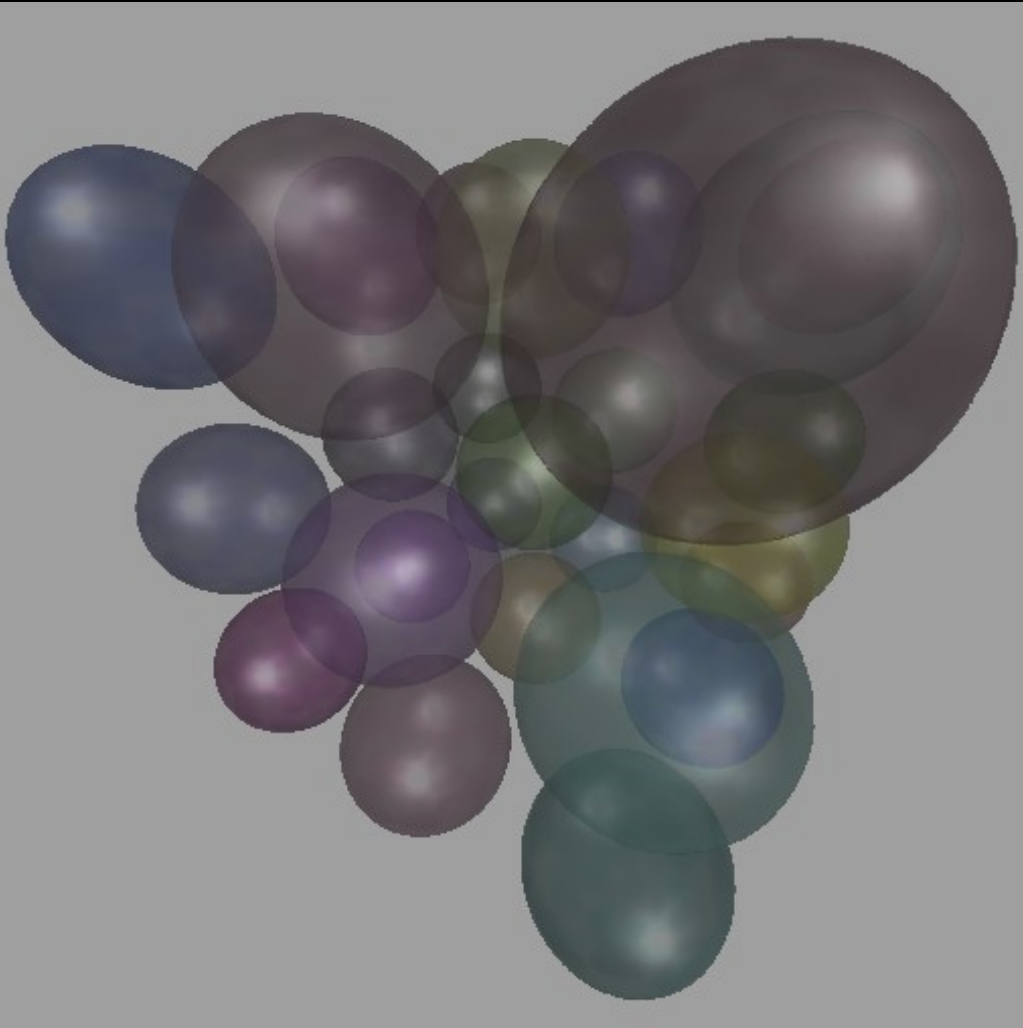


complete
depth-peeling

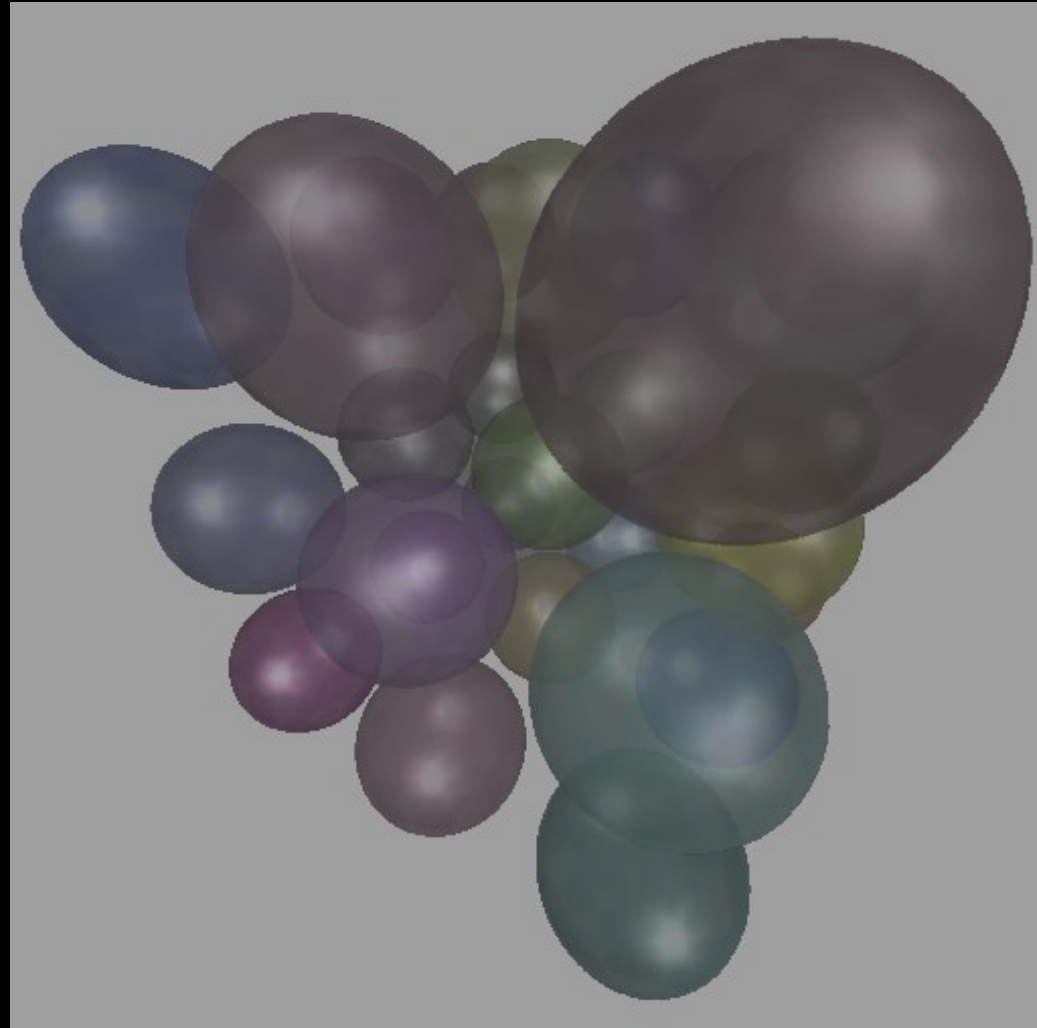


only the first layer
+ a 2nd layer with approx blending

Approximate depth-peeling (results with 3 layers)



complete
depth-peeling



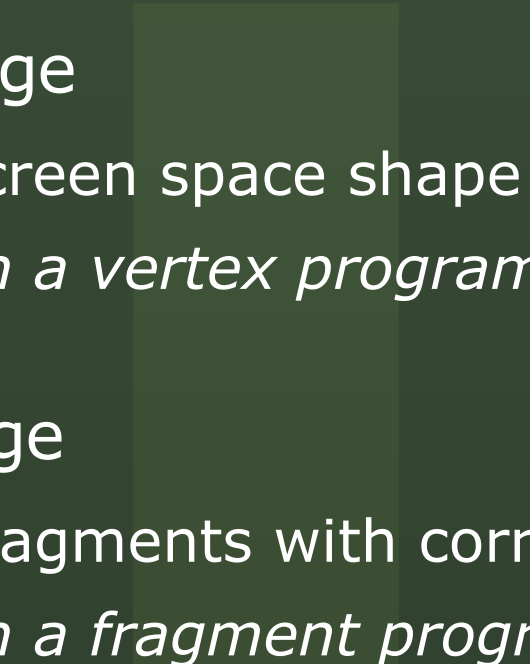
only the first 2 layers
+ a 3th layer with approx blending

Splat rasterization

Point Cloud Rendering

- Ray-cast a reconstructed surface (MLS)
 - Best quality but slow, requires pre-process...
- Rasterization (splatting)
 - best quality criteria:
 - perspective correct splat rasterization
 - per-pixel shading (\Rightarrow deferred shading)
 - high frequency filtering (aliasing)
 - performance criteria:
 - use the best of current GPU
 - incremental calculations for the rasterization

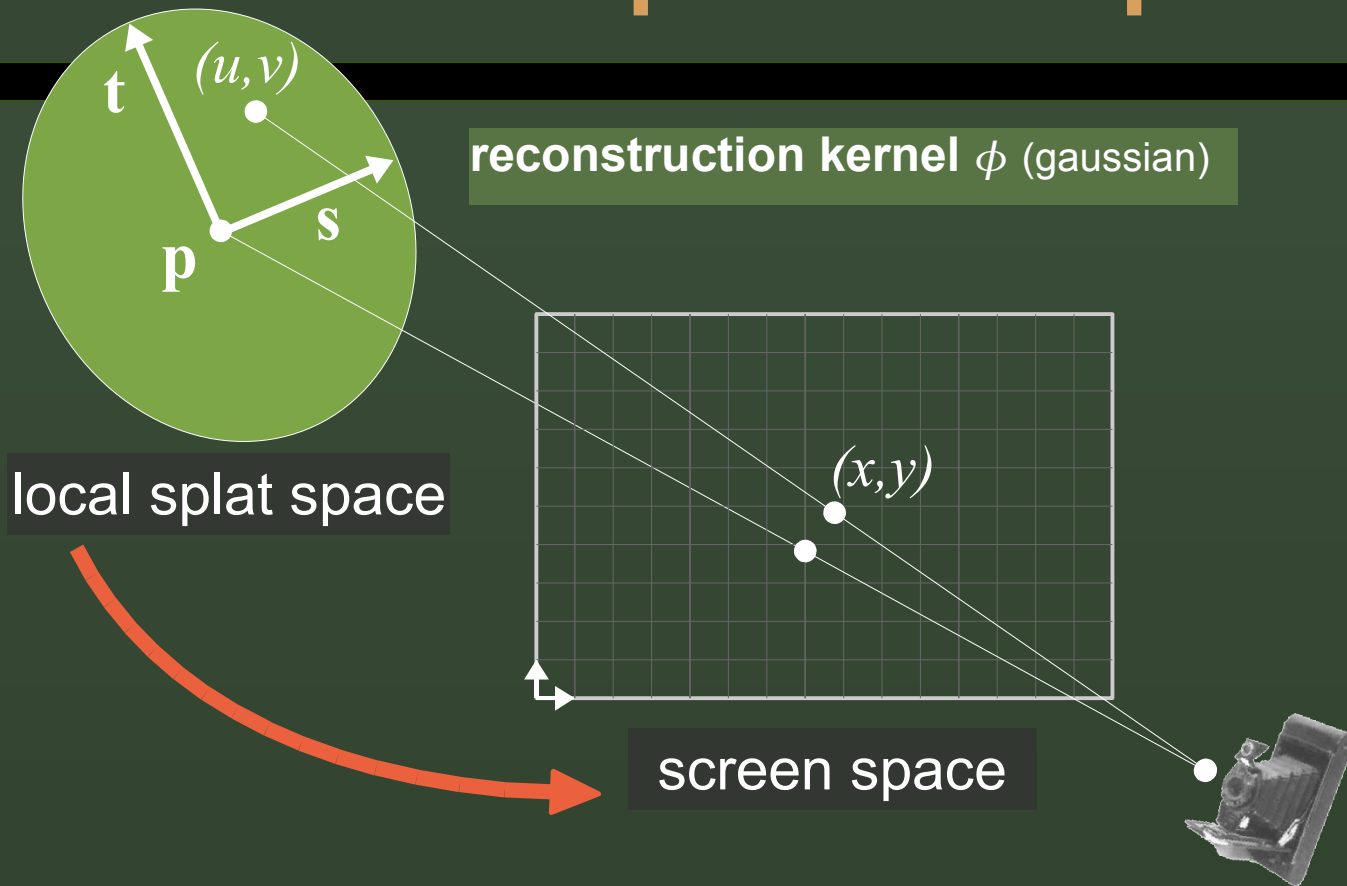
Splat rasterization

- Decomposed as two stages:
 - “splat setup” stage
 - compute the screen space shape of the splat
 - *implemented in a vertex program*
 - rasterization stage
 - generate the fragments with correct depth and weight
 - *implemented in a fragment program (+ point sprite)*
- 

splat rasterization implementations

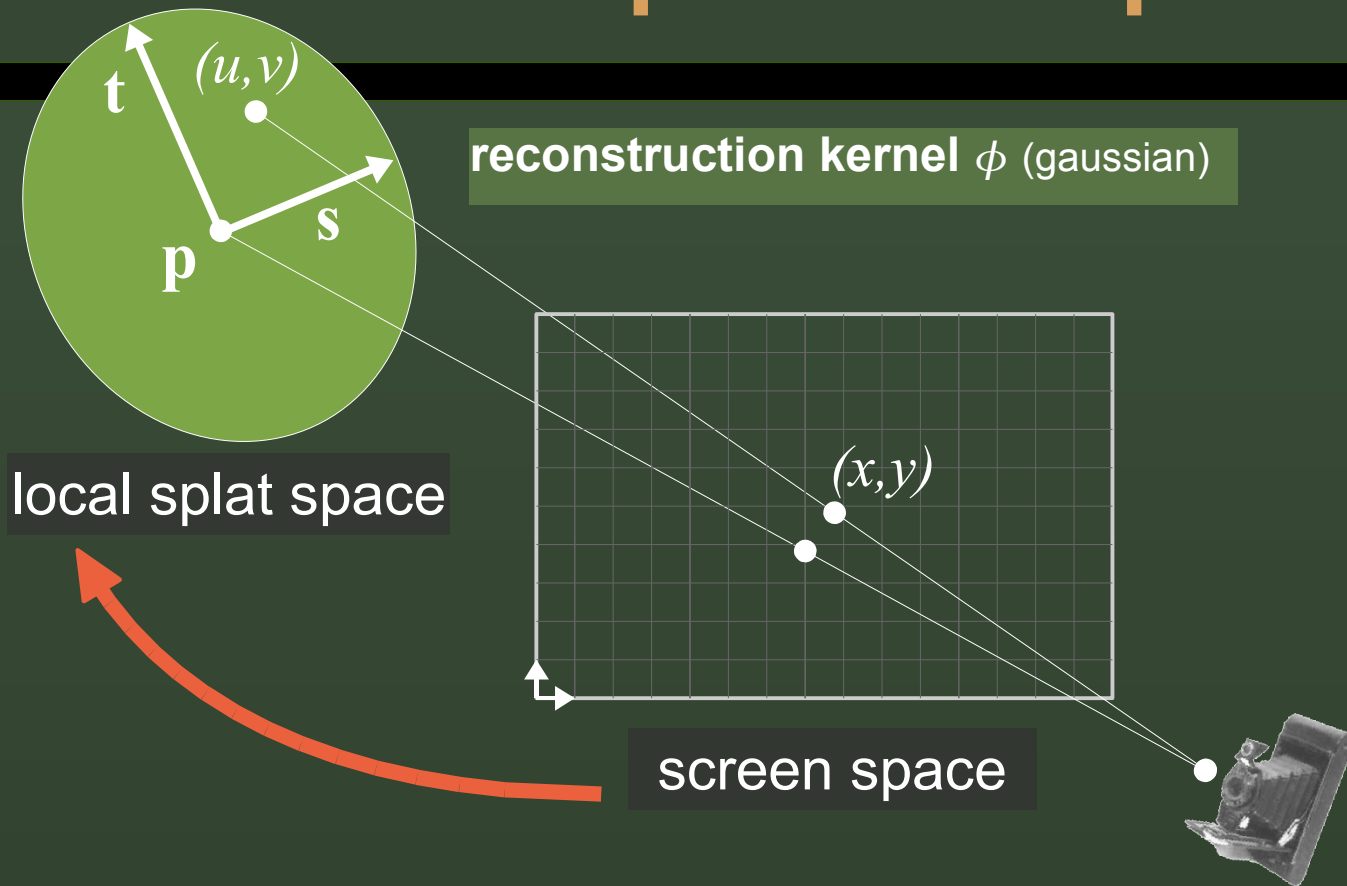
	perspective OK	EWA filtering	suitable for incremental computation	# instr. setup	# instr. raster
<i>EWA Splatting</i> [Zwicker01]	✗	✓	✓	software	
[Guennebaud03]	✗	✓	✓	51	6
Perspec. Accu. [Zwicker04]	(✓)	✓	✓	93	8
[Botsch05] (ray casting)	✓	✓	✗	35	13
[PBG06]	✓	(✓)	✓	58	3

Perspective splatting



$$\begin{bmatrix} xz \\ yz \\ z \end{bmatrix} = M \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{s} & \mathbf{t} & \mathbf{p} \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Perspective splatting



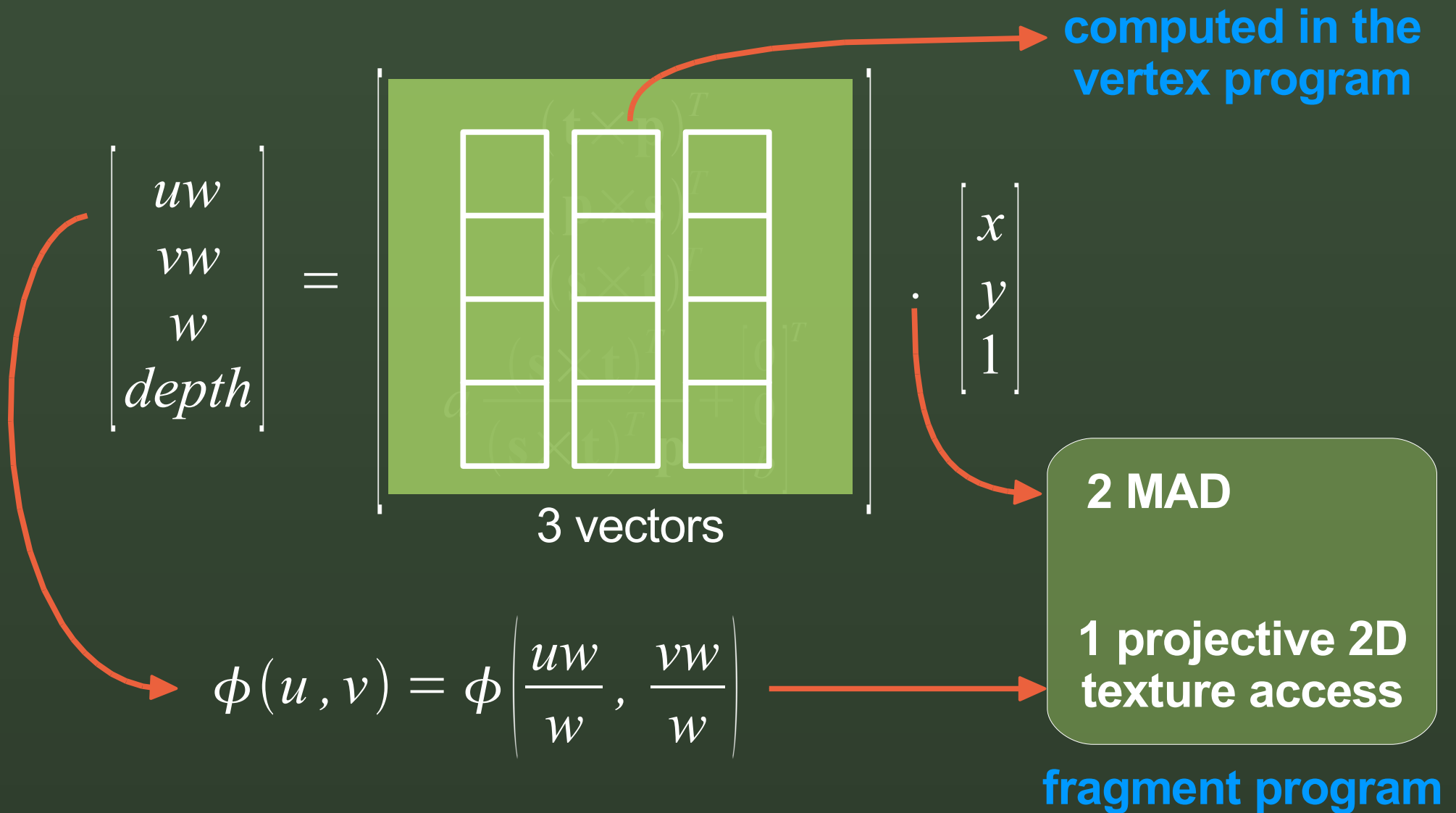
$$\begin{bmatrix} uw \\ vw \\ w \end{bmatrix} = \begin{bmatrix} (\mathbf{t} \times \mathbf{p})^T \\ (\mathbf{p} \times \mathbf{s})^T \\ (\mathbf{s} \times \mathbf{t})^T \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Depth value

- We have $w = \alpha \frac{1}{z}$ and $depth = a \frac{1}{z} + b$
- Hence:

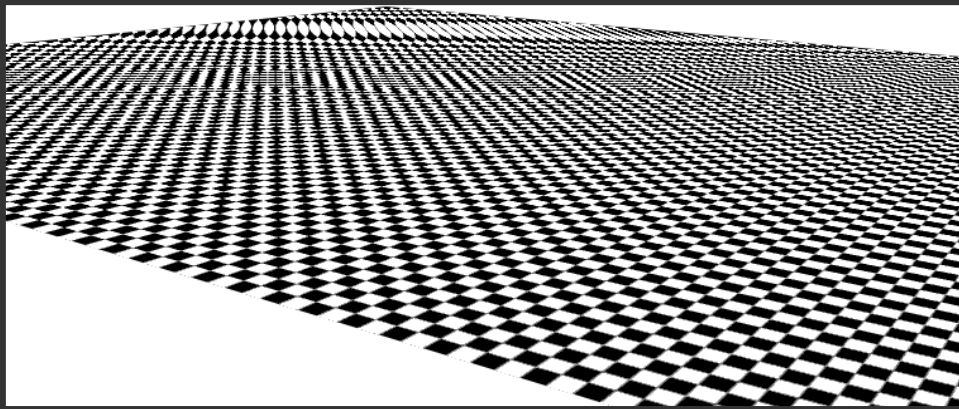
$$\begin{bmatrix} uw \\ vw \\ w \\ depth \end{bmatrix} = \begin{bmatrix} (\mathbf{t} \times \mathbf{p})^T \\ (\mathbf{p} \times \mathbf{s})^T \\ (\mathbf{s} \times \mathbf{t})^T \\ a \frac{(\mathbf{s} \times \mathbf{t})^T}{(\mathbf{s} \times \mathbf{t})^T \mathbf{p}} + \begin{bmatrix} 0 \\ 0 \\ b \end{bmatrix}^T \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

GPU implementation



EWA filtering

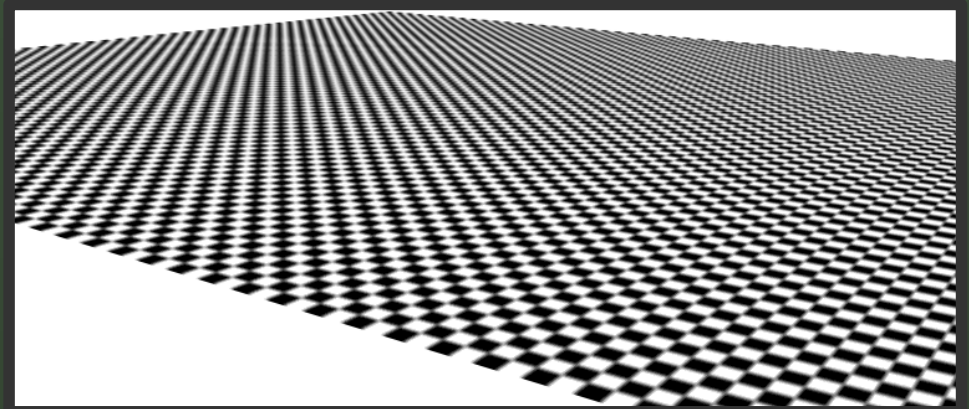
- Object-space filter only



$$\phi'(\mathbf{x})$$

↑
warped
reconstruction
kernel

- EWA filtering



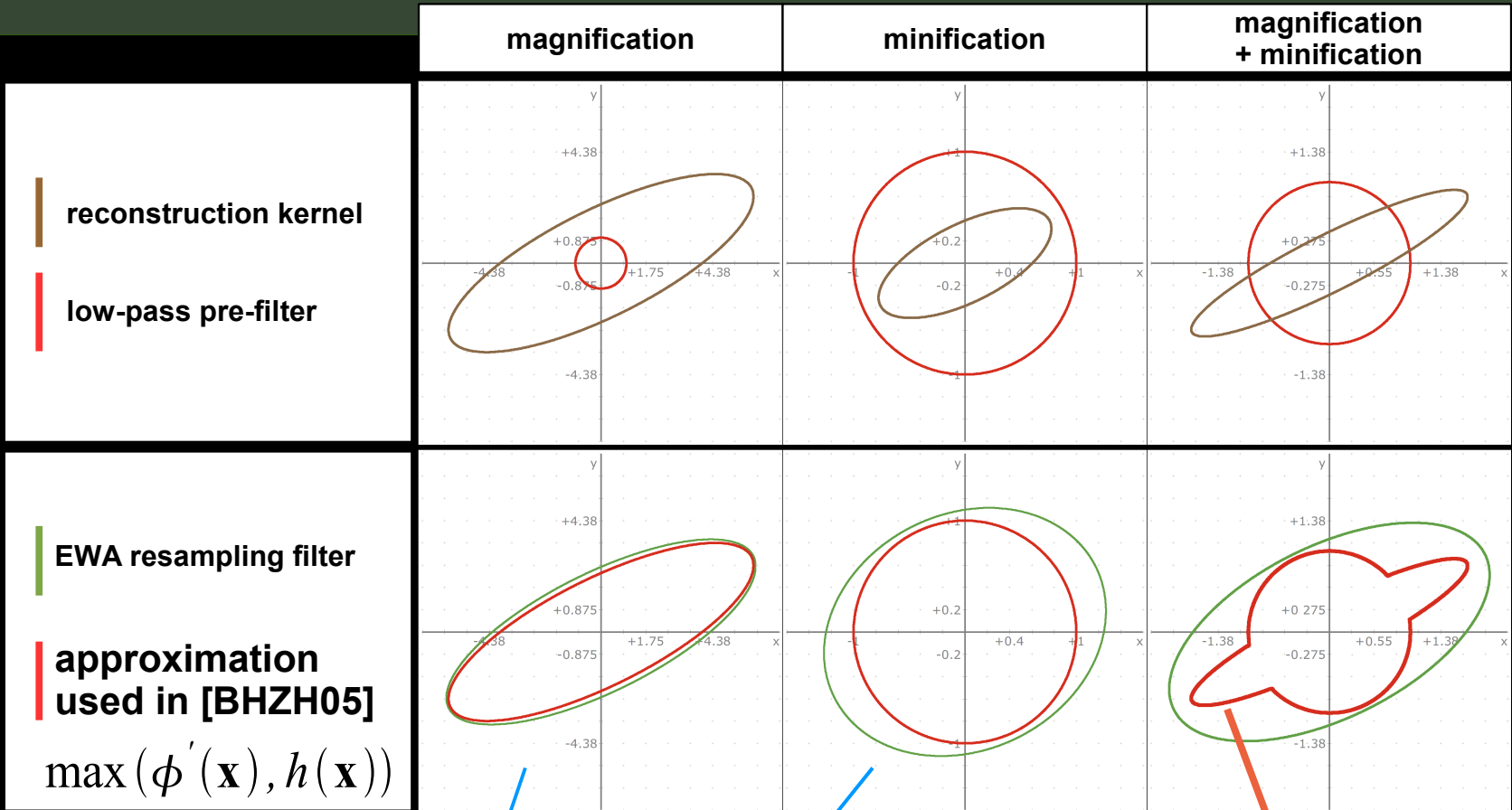
$$(\phi' \otimes h)(\mathbf{x})$$

↑
reconstruction
kernel

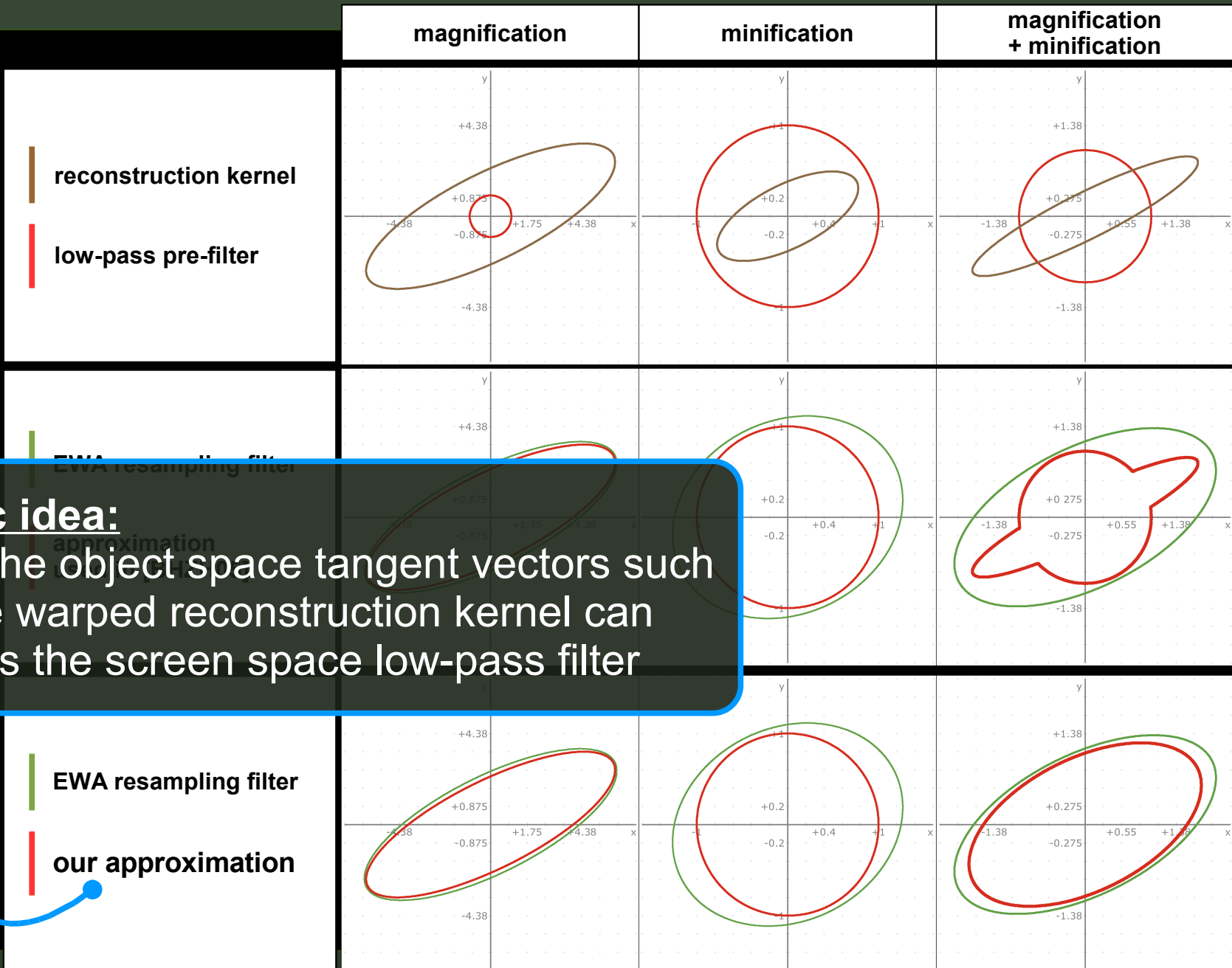
↑
low-pass filter

- OK for affine mapping only

EWA filtering approximations



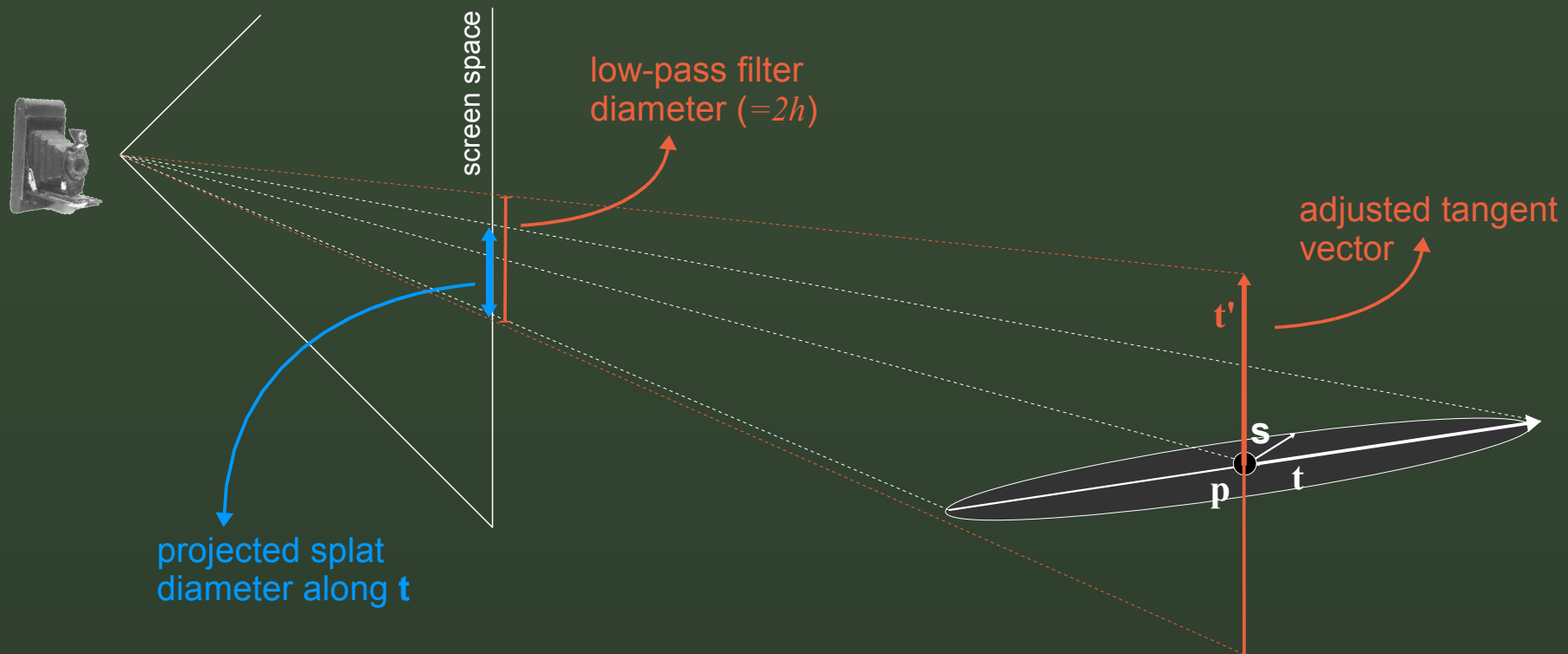
EWA filtering approximations



• **Basic idea:**
 adjust the object space tangent vectors such that the warped reconstruction kernel can contain the screen space low-pass filter

EWA filtering approximation

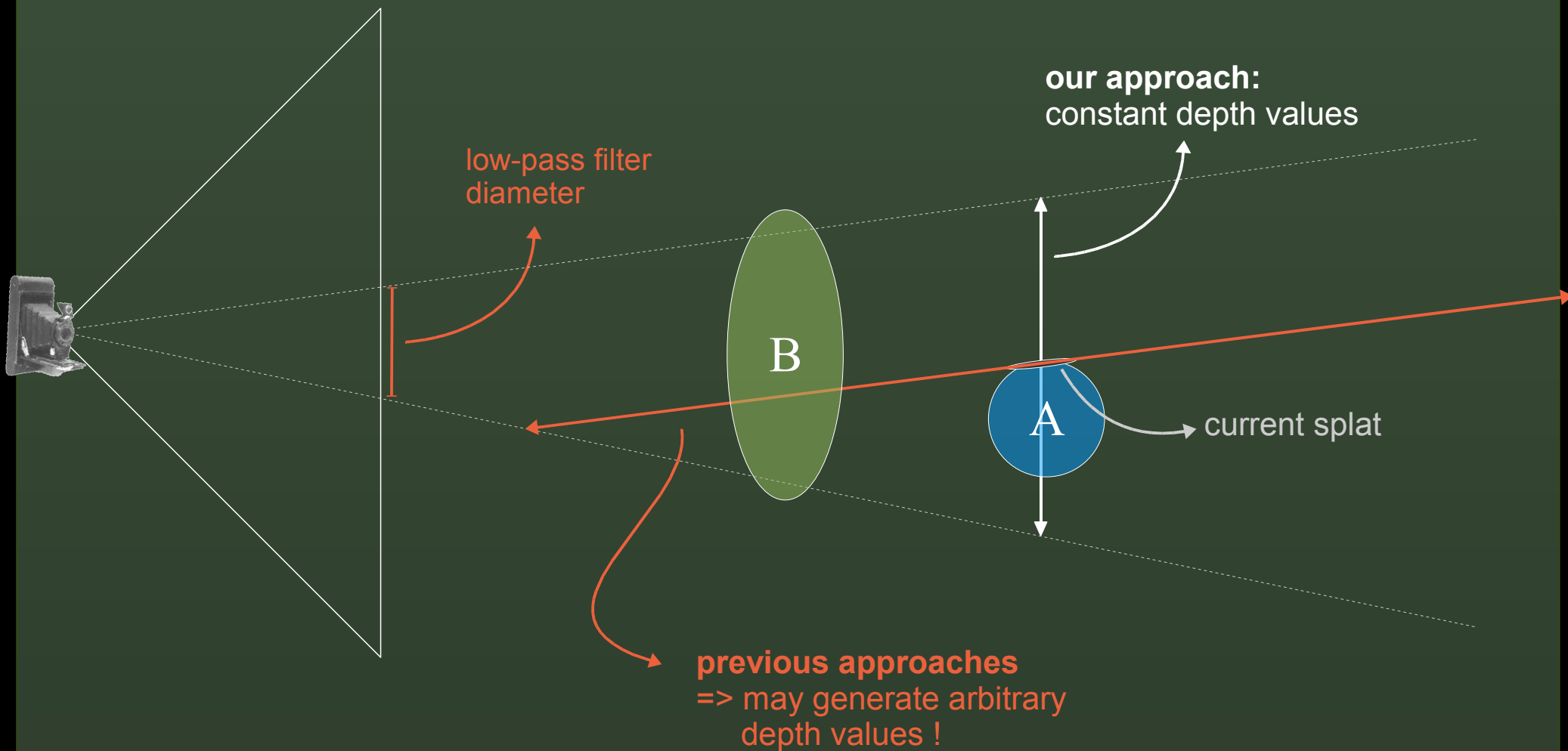
- Tangent vectors adjustment
 - only check along the tangent vector directions



EWA filtering approximation

- Provides the expected result if and only if the projected tangent vectors are orthogonal
 - => on the fly re-parametrization ?
 - too much expensive
 - => efficient heuristic for isotropic splats (disks):
 - $\mathbf{s} = \mathbf{p} \times \mathbf{n}$
 - $\mathbf{t} = \mathbf{n} \times \mathbf{s}$
 - exact at the screen center
 - exact for splats parallel to the screen plane
 - “good” worst case

About depth values and EWA filtering



EWA filtering approximation (results)



[Botsch et al. 05]



our new approximation

Performances



screen resolution: 1024x1024

	Perspective splatting	raycasting [BHZK05]
#instr.	46/3	34/9
visibility pass		
attribute pass	58/3	35/13
154k	50 (7.7)	33 (5)
460k	40 (18.4)	26 (12)
1.4M	22 (31)	13 (18.2)
2.5M	15 (37.5)	9.2 (23)
5M	6.5 (32.5)	5 (25)
	<i>fps (M splats/s)</i>	<i>fps (M splats/s)</i>

Hybrid rendering

Hybrid rendering (motivations)

- Flat surface or large zoom
 - points are inefficient (both in speed and quality)
 - ⇒ **hybrid rendering**
points and polygons are complementary

use triangles when points become less efficient



- What about the transitions ?

Hybrid rendering (transition smoothing)



- straightforward !
- no additional rendering cost !

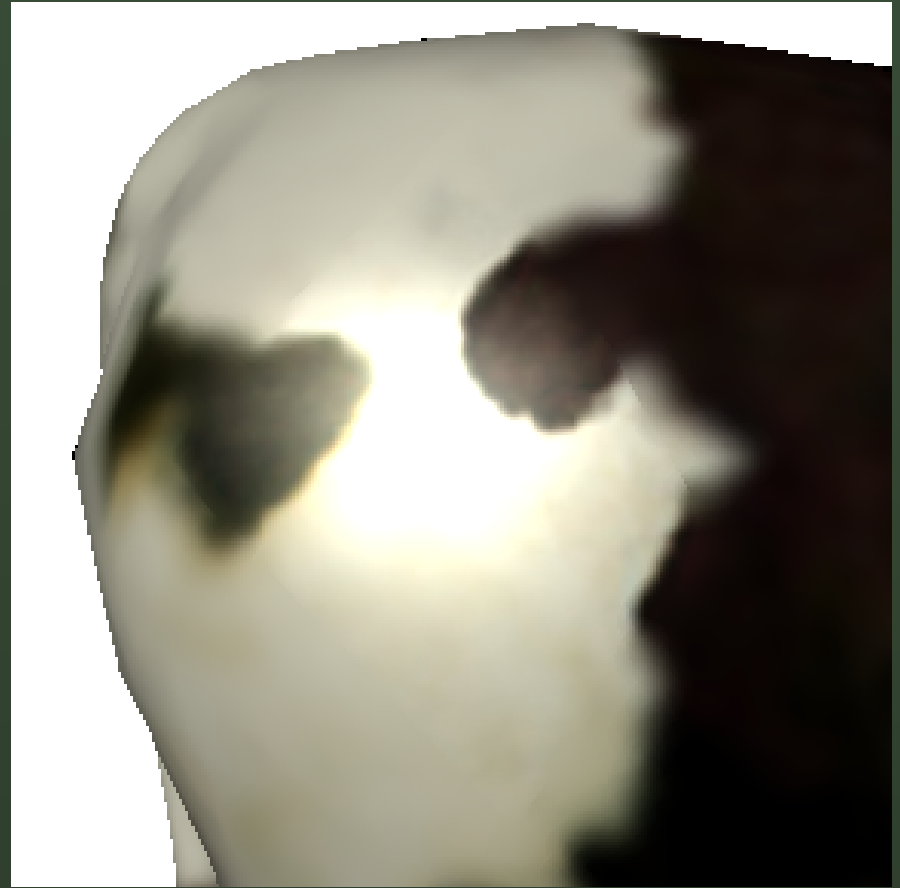
standard splats &
polygons rendering

splatting + \sum weights

hybrid rendering
with alpha-blending

Hybrid rendering (transition smoothing)

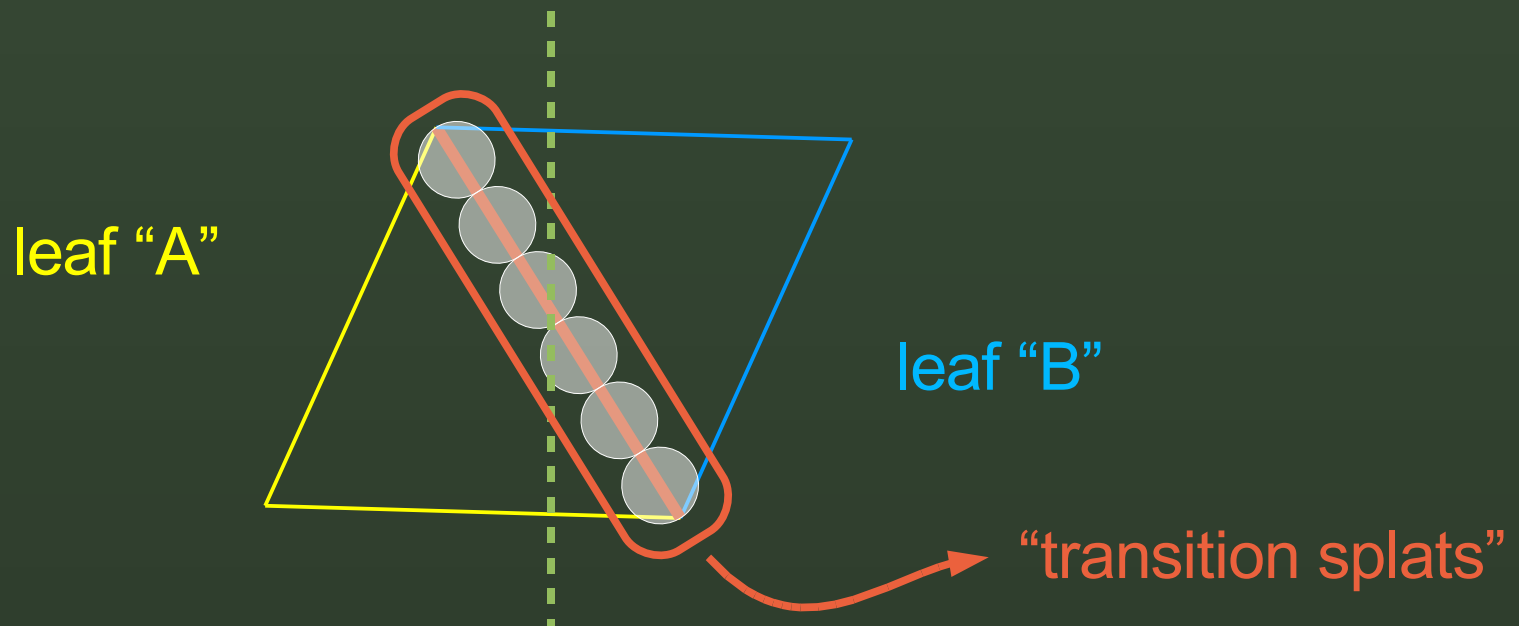
- Too much straightforward ?



- Best quality => uniform sampling of the "transition edges"

Hybrid rendering (implementation example)

- Multi-resolution hierarchy of points
 - Leaves store both points and polygons
- At the sampling time:
 - explicitly sample the edges shared by two faces stored in two different leaves



Hybrid rendering

(implementation example)

- Hybrid rendering rules:
 - render the polygons (instead of the splats) of all visible and not dense enough leaf node.
 - render the transition splats shared by at least one leaf rendered as a set of splats

Conclusion

- Summary:
 - Approximate depth-peeling for efficient transparency
 - Perspectively correct splat rasterization
 - efficient on current GPU
 - allows efficient dedicated implementation (incremental computation)
 - EWA filtering approximation
 - same quality as full EWA filtering
 - **only for isotropic splats**
 - Splat/polygon transitions smoothing

- Ray-casting -> splatting -> EWA splatting
- splat rasterization, 2 class of approaches:
 - perspective approx
 - match the center or the contour (better)
 - allow EWA filtering (by an analytic convolution)
 - expensive splat setup
 - suitable for incr. rasterization
 - ray casting
 - simple to implement
 - perspective correct
 - simple splat setup (all the computation are performed at the fragment level)
 - expensive rasterization shader

EWA filtering approximation

- Basic idea:
 - adjust the tangent vectors s and t such that the warped reconstruction kernel contains the screen space low-pass filter
 - \sim adjust the tangent vectors s and t such that their screen space length are greater than the radius of the screen space low-pass filter
 - OK if and only if the tangent vectors are still orthogonal in the screen space and the low pass filter is radially symmetric