graphics hardware 06

Realistic Soft Shadows by Penumbra-Wedges Blending

Vincent Forest, Loïc Barthe,
Mathias Paulin

IRIT-UPS-CNRS University of Toulouse, France

www.irit.fr/~Vincent.Forest/softshadows.php

Outline

- Introduction
- Previous works
- The Penumbra-wedges approach
- Realistic soft shadows by penumbra-wedges blending
- Hardware implementation & results
- Conclusion & perspectives

Hard VS soft shadows

Hard shadows:

- Is the light source visible?
  - Easy and fast to compute
  - Based on the assumption that lights are points

Soft shadows:

- What area of the light source is visible?
  - Shadows for extended light source
  - Require more efforts/horsepower than hard shadows

Previous works [1/2]

Two approaches for hard shadows generation

- Image based (shadow map [Williams 78])
  - Use one or many shadow maps to define if a fragment is visible from the light
- Geometry based (shadow volume [Crow 77])
  - Build a shadow volume by object space silhouette extraction that contains shadowed fragments

Soft shadows algorithms are derived from these two approaches

Previous works [2/2]

Image based soft shadows algorithms

- Visually plausible soft shadows
  - PCF, Smoothie [Chan *et al.* 03], Penumbra-map [Wyman *et al.* 03], ...
- Physically plausible soft shadows
  - Soft shadows by back projection [Guennebaud *et al.* 06], Sampling of light source visibility [Atty *et al.* 05]

Geometry based soft shadows algorithm

- A physically plausible algorithm
  - Penumbra-Wedges [Assarsson *et al.* 03]

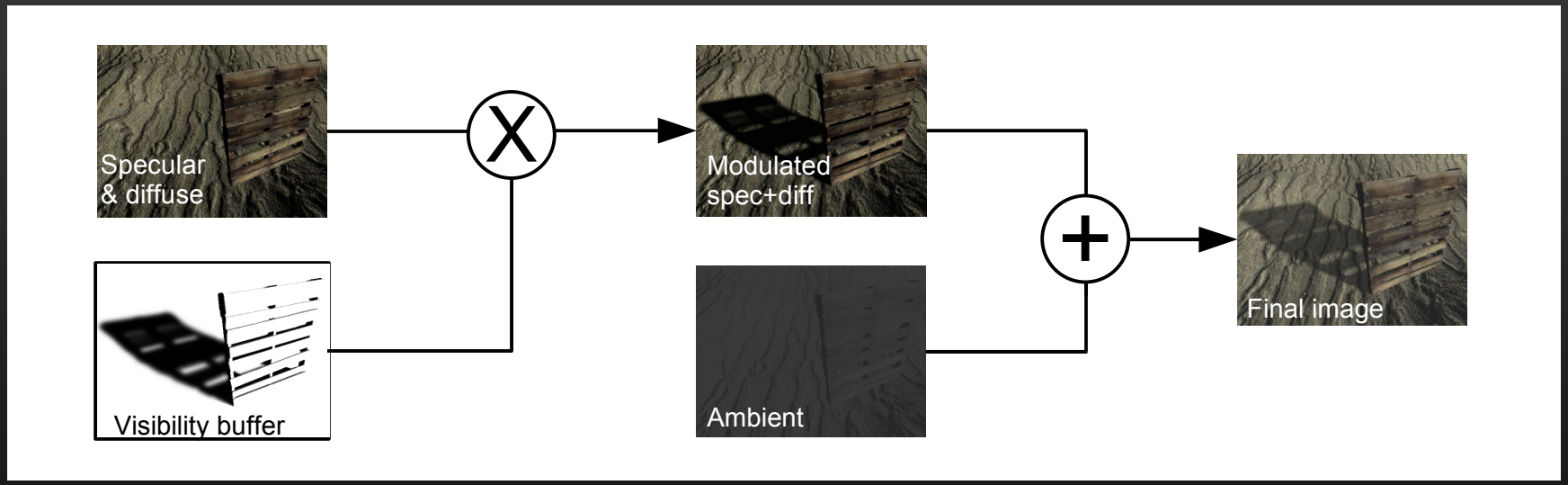
Penumbra-wedges [1/4]

Penumbra-wedges algorithm

- Takes advantage of the shadow-volume evolutions for efficient/robust real-time implementation
  - Z-fail algorithm [Carmack 00]
  - Split-plane shadow volume [Laine 05]
  - ...
- Generates non-aliased and « physically plausible » soft shadows

Penumbra-wedges [2/4]

Overview

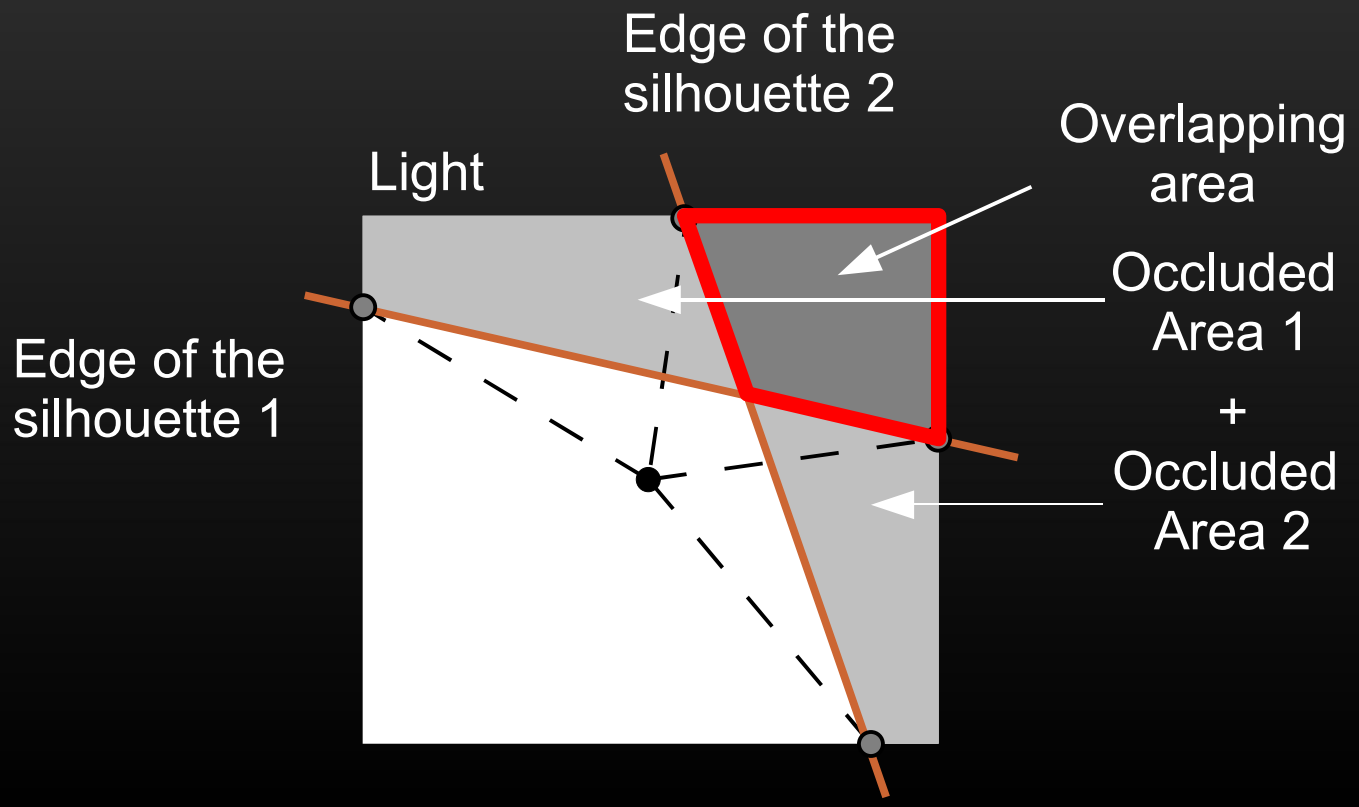


Visibility buffer computation

- First pass: draw hard shadow → Shadow-volumes
- Second pass: modulate the visibility buffer for fragments in penumbra → Penumbra-wedges

Penumbra-wedges [3/4]

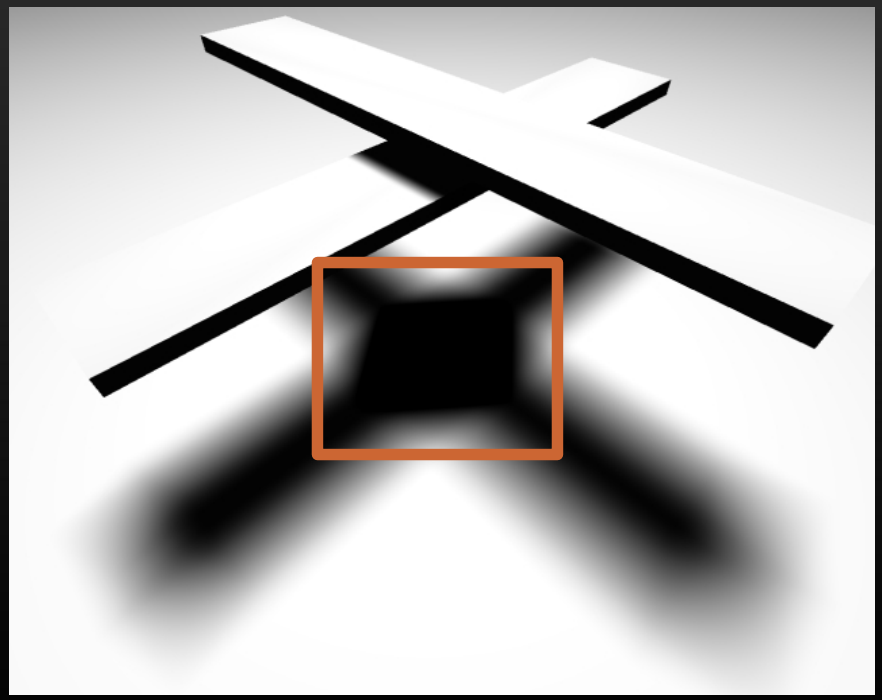
Penumbra-wedges algorithm is based on the assumption that silhouettes are non-overlapping



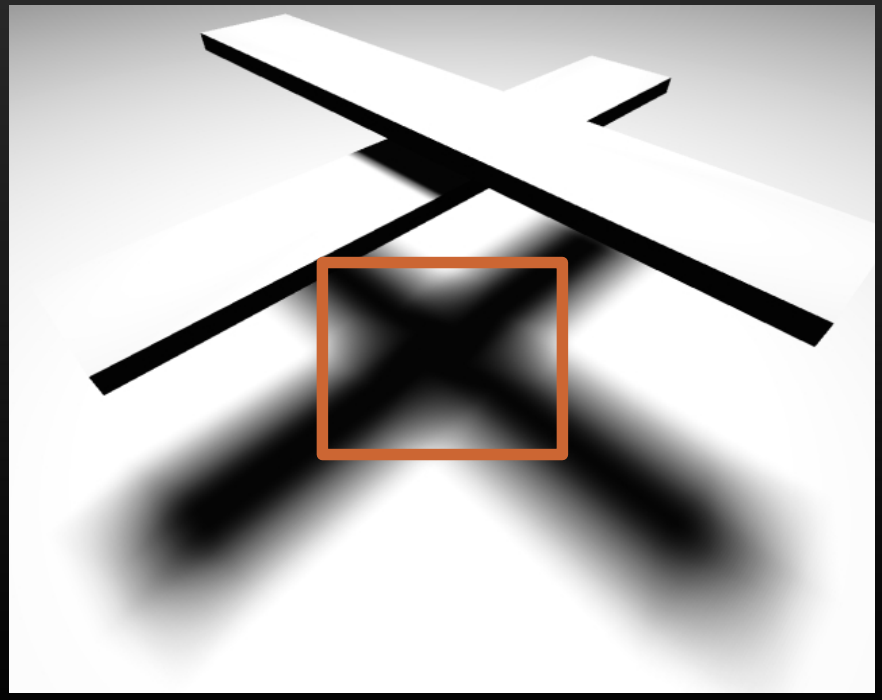
Penumbra-wedges [4/4]

Penumbra-wedges algorithm

- Generates non-aliased and « physically plausible » soft shadows except when silhouettes are overlapping



Penumbra-wedges



Reference

Penumbra blending [1/6]

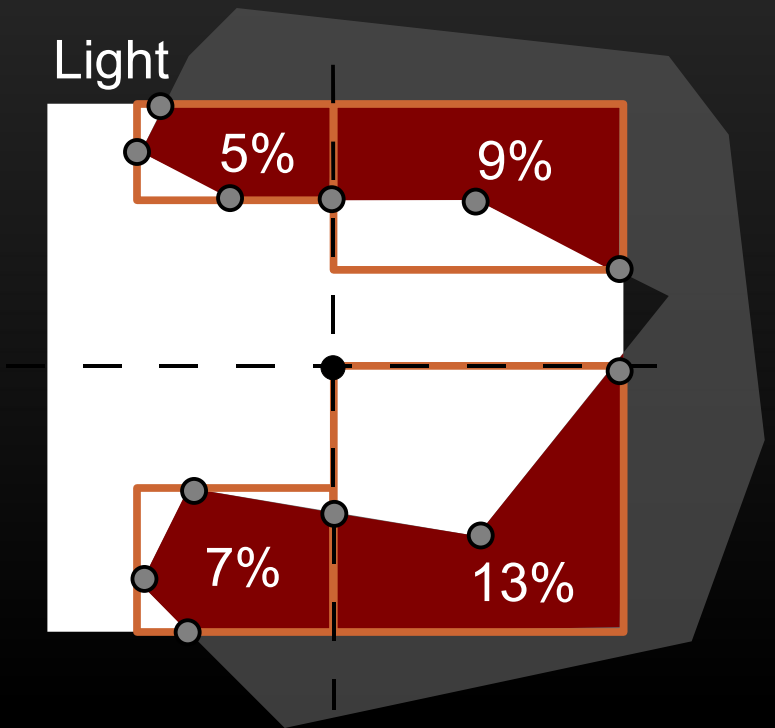
The accurate blending of penumbrae requires the knowledge of the geometry of the occluded light region for each fragment

- Compute a vBuffer per silhouette
- Blend the per silhouette vBuffer within a final vBuffer

Penumbra blending [2/6]

Step 1: Per silhouette vBuffer computations

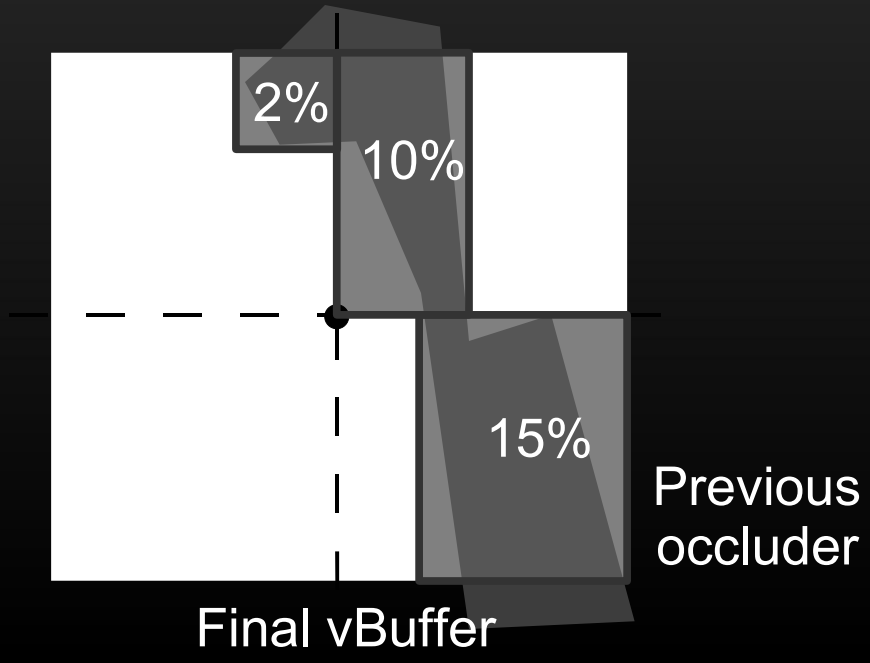
- The occluded area is bounded by a rectangle
- For a better approximation of the occluder geometry the light is subdivided in 4 parts
- The vCoef is then computed independently per radial part

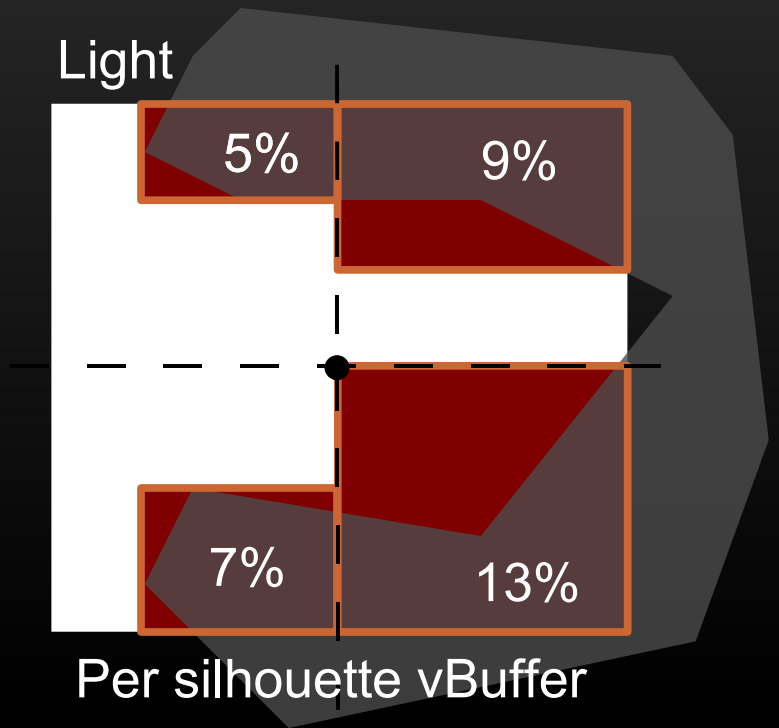


Penumbra blending [3/6]

The per silhouette vBuffer stores an approximation of the geometry of the occluder and its coverage percentage

- The next step updates the final vBuffer

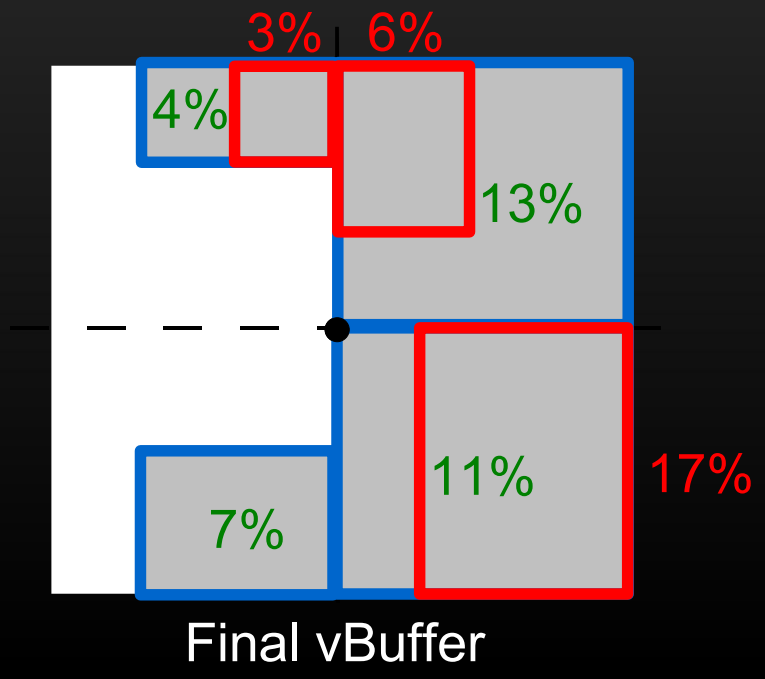




Penumbra blending [4/6]

Step 2: Per silhouette vBuffer blending

- Accumulate the light occlusion percentage
- Identify the possible overlaps in occluded regions
- Compute the occluded light percentage of the possible overlapping regions
- Subtract it from the percentage computed before
- Update bounding rectangle



Penumbra blending [5/6]

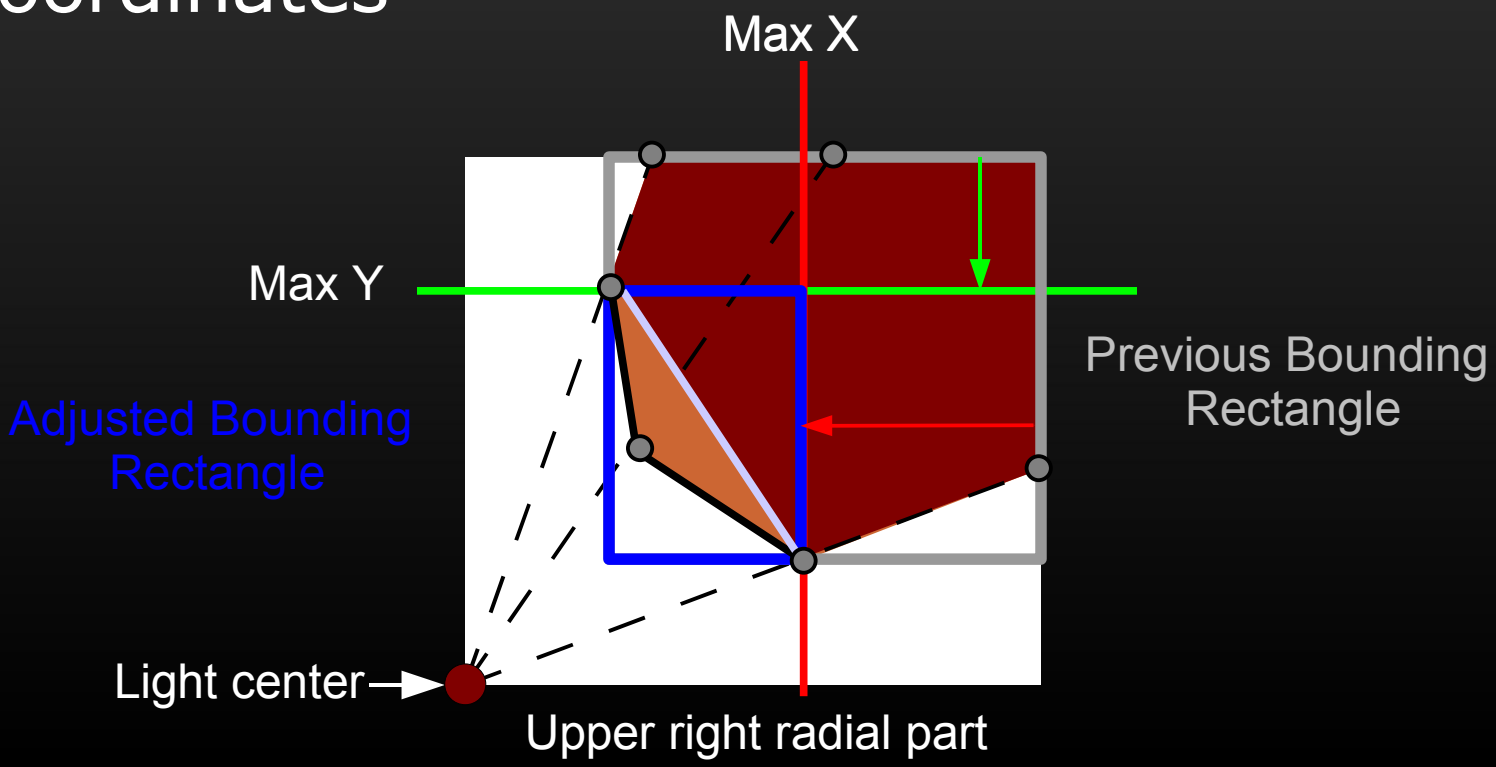
Since bounding rectangles region can be partially occluded, the overlapped area is approximated as follow:

$$E = \frac{A_f}{B_f} \cdot \frac{A_s}{B_s} \cdot b$$

- E: effective overlapped area
- b: bounding rectangle intersection area
- $_{s,\,f}$: indices indentifying respectively the silhouette and the final vBuffer
- A_x: light occlusion area
- B_x: bounding rectangle area

Penumbra blending [6/6]

The computation of the silhouette bounding rectangle requires the knowledge of the maximum in X and Y of the occluding edges coordinates



GPU implementation [1/4]

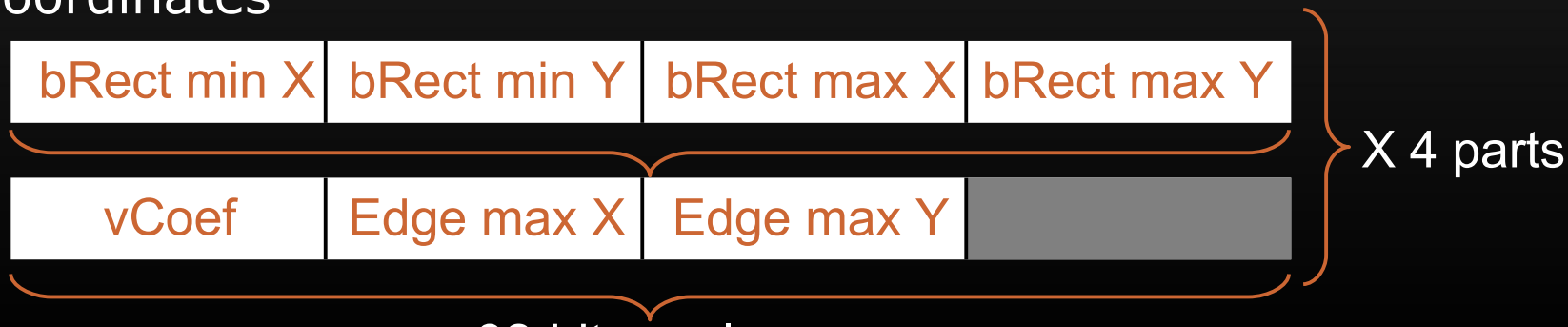
3 passes algorithm

- Init the silhouette vBuffer by a shadow-volume pass
- Compute the vCoef of the fragments in the penumbra of the silhouette
  - Subdivide light in 4 radial parts, and update vCoef independently for each part
  - In each radial part, approximate the occluder geometry by a bounding rectangle
- Blend the silhouette vBuffer with the final vBuffer
  - Use the bounding rectangles to determine the overlapping area and to correct the penumbrae blending

GPU implementation [2/4]

Parameters are stored with a precision of 8-bits and packed in 32-bits scalars

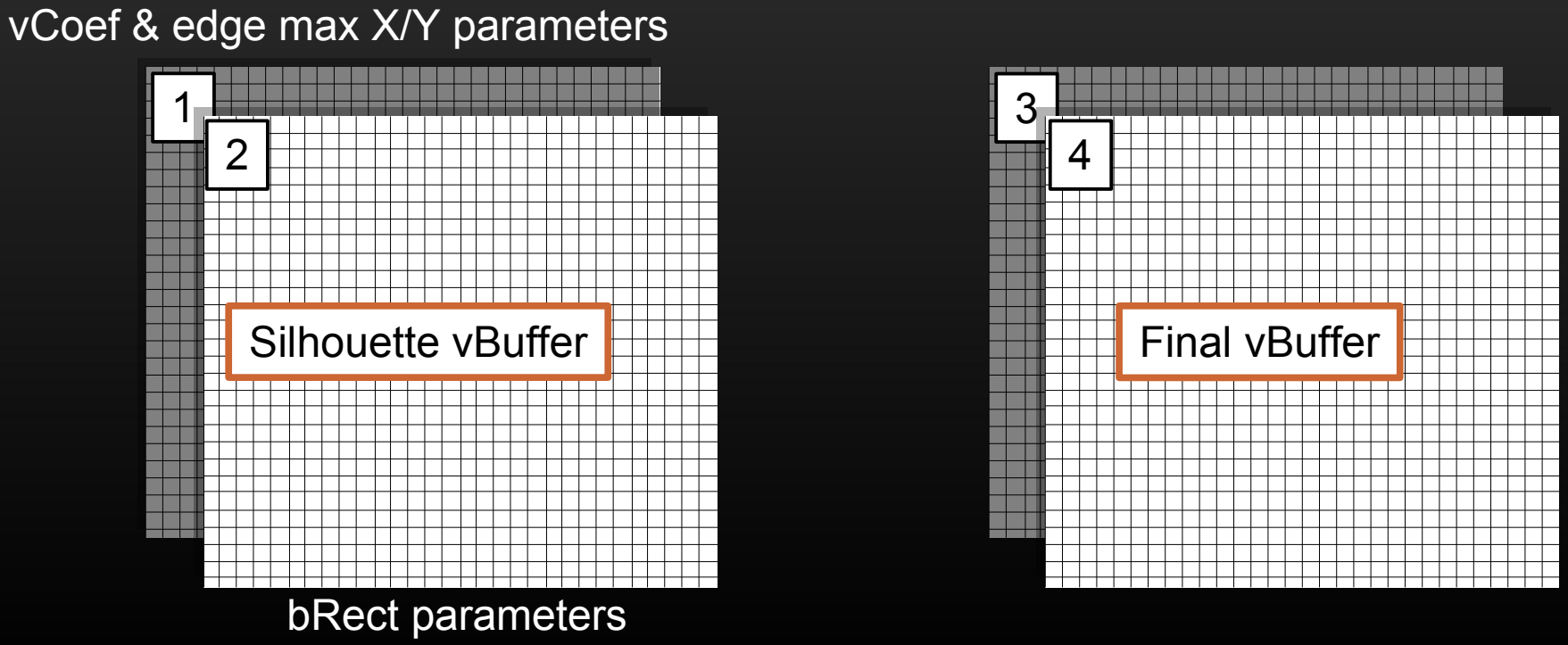
- Per radial part parameters:
  - Top, bottom, right and left coordinates of the bounding rectangle
  - Light occlusion percentage
  - The maximum in X and Y of the occluding edges coordinates



GPU implementation [3/4]

4 RGBA simple precision float buffers

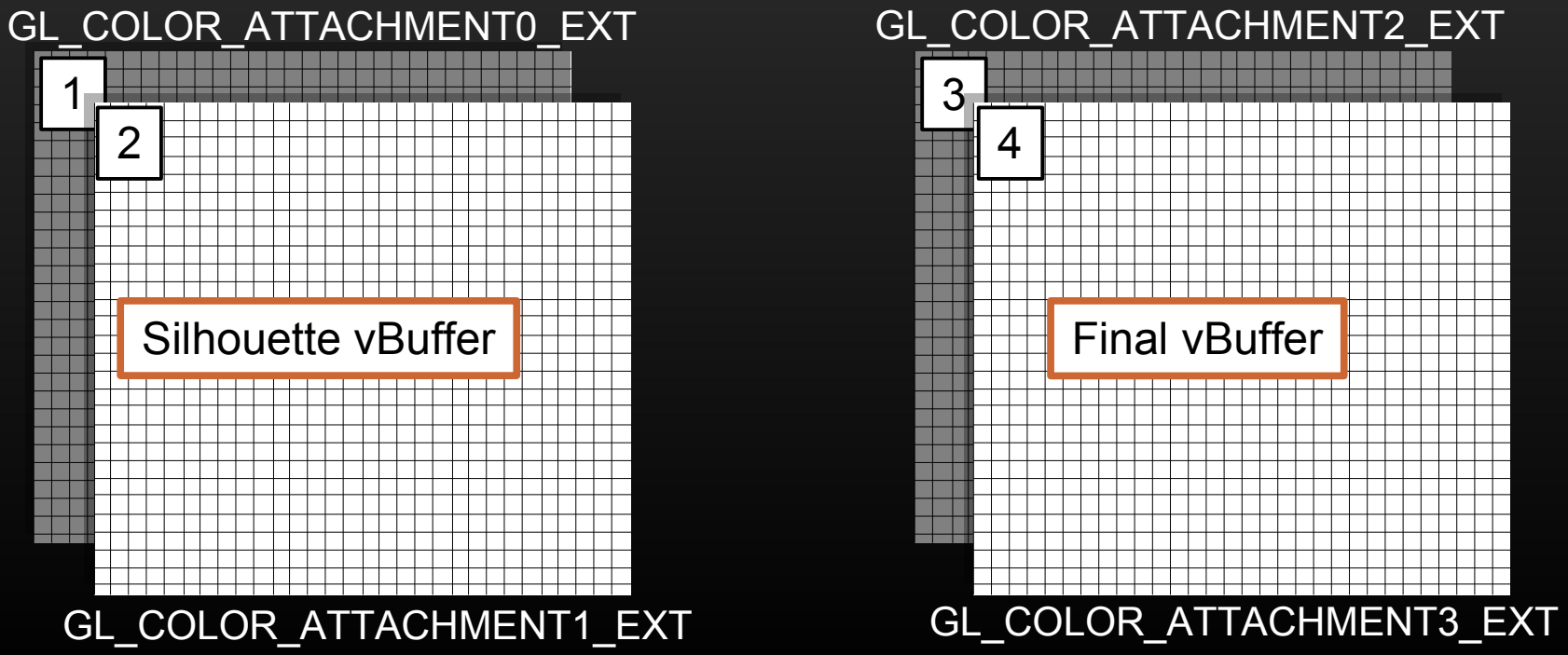
- Needs MRT for parameters update
- Uses FBO for render to texture



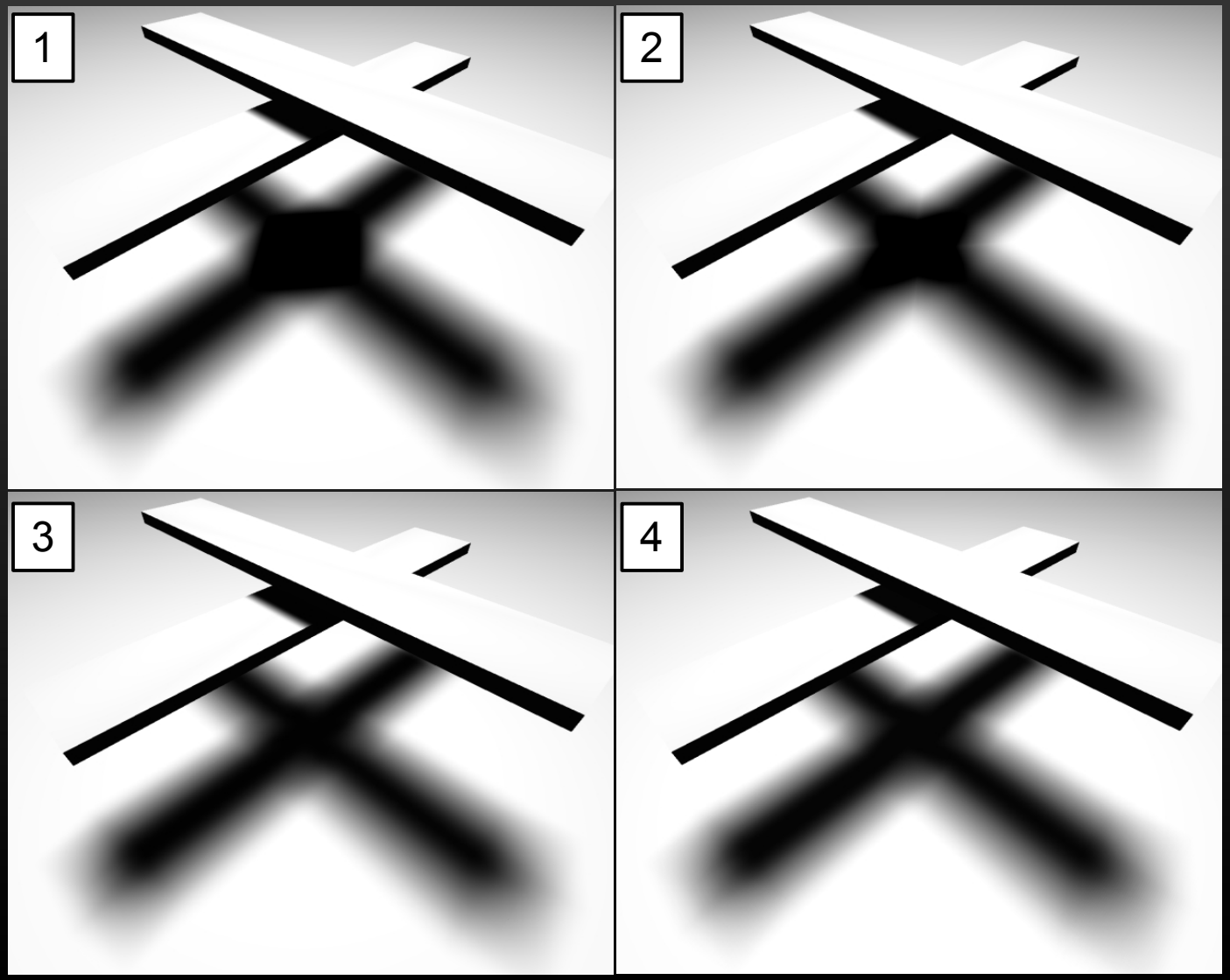
GPU implementation [4/4]

4 RGBA simple precision float buffers

- GL_MAX_COLOR_ATTACHMENTS_EXT = 4
- Needs only one FBO



Results [1/2]



1 Penumbra-Wedges

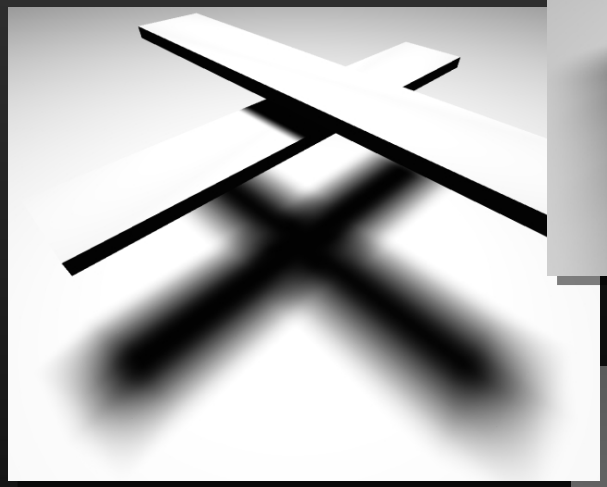
2 Probabilistic approach
[Assarsson *et al.* 03]

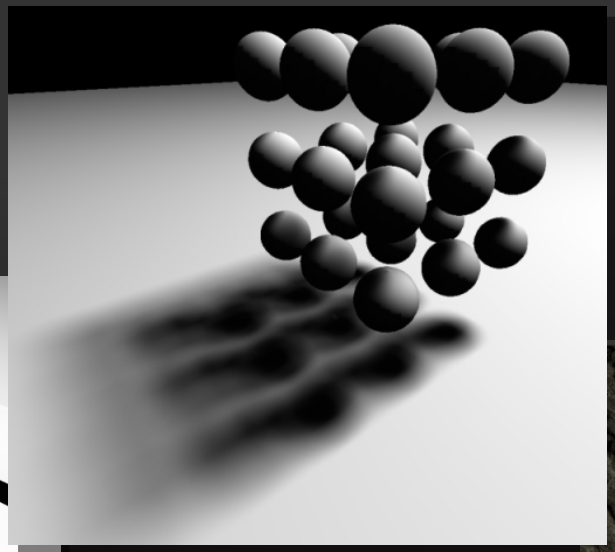
3 Our algorithm

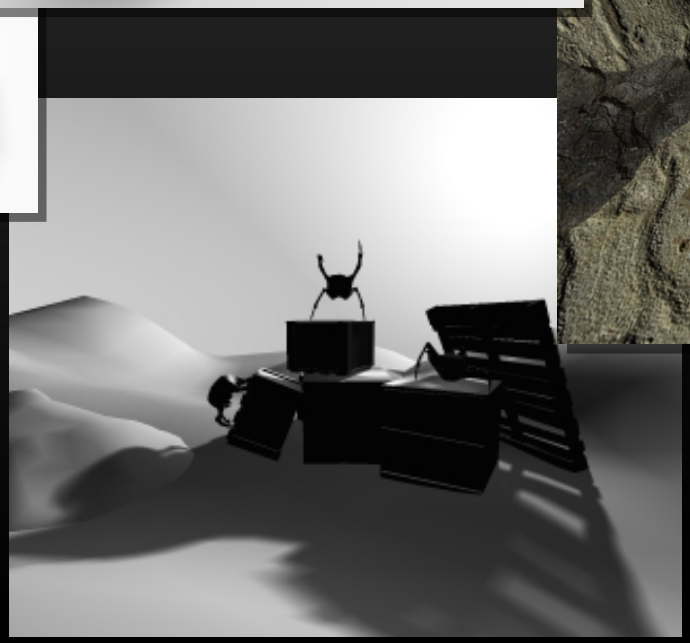
4 Reference

Results [2/2]

Videos

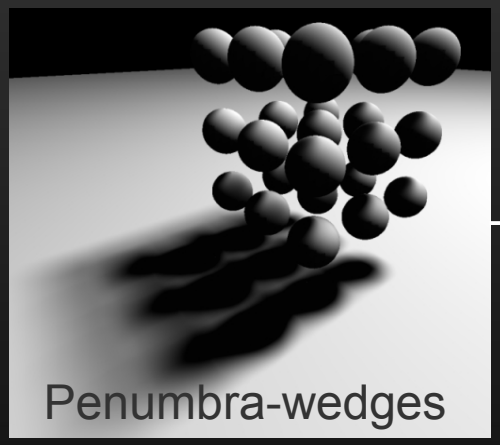


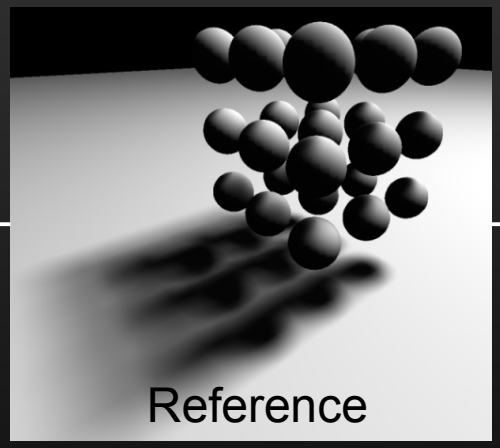


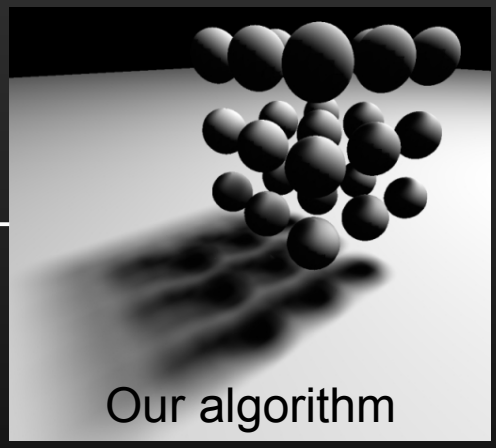


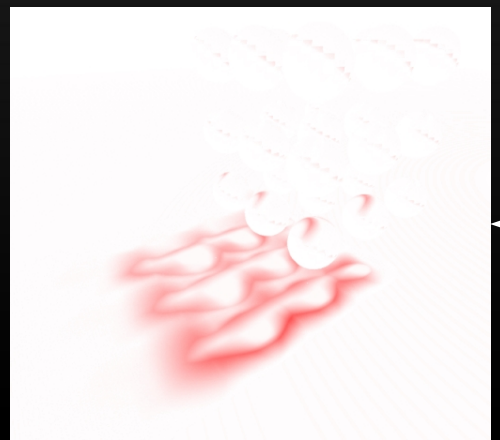
Conclusion

- We have proposed a physically plausible soft shadows algorithm based on the penumbra-wedges
- Our algorithm significantly reduces the overlapping artefacts





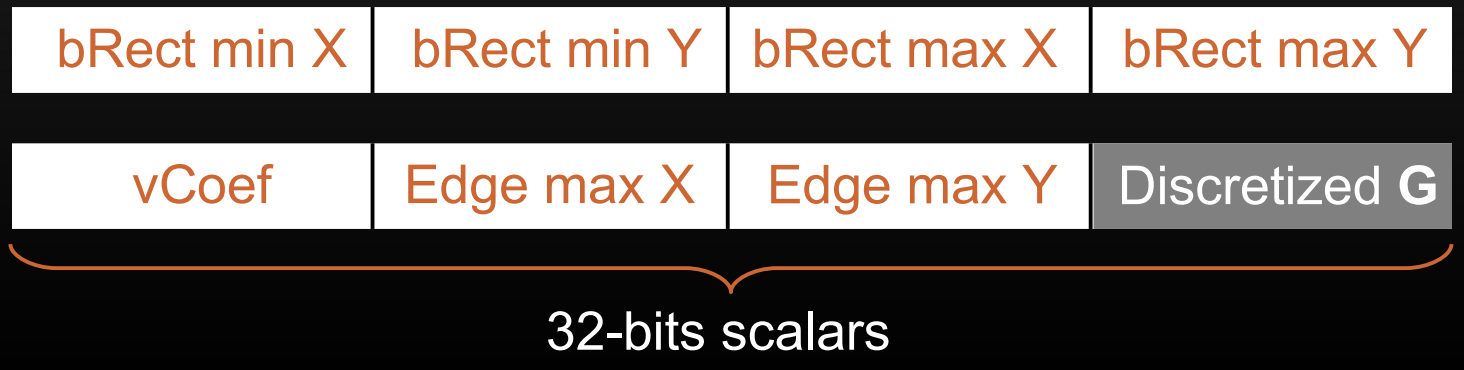


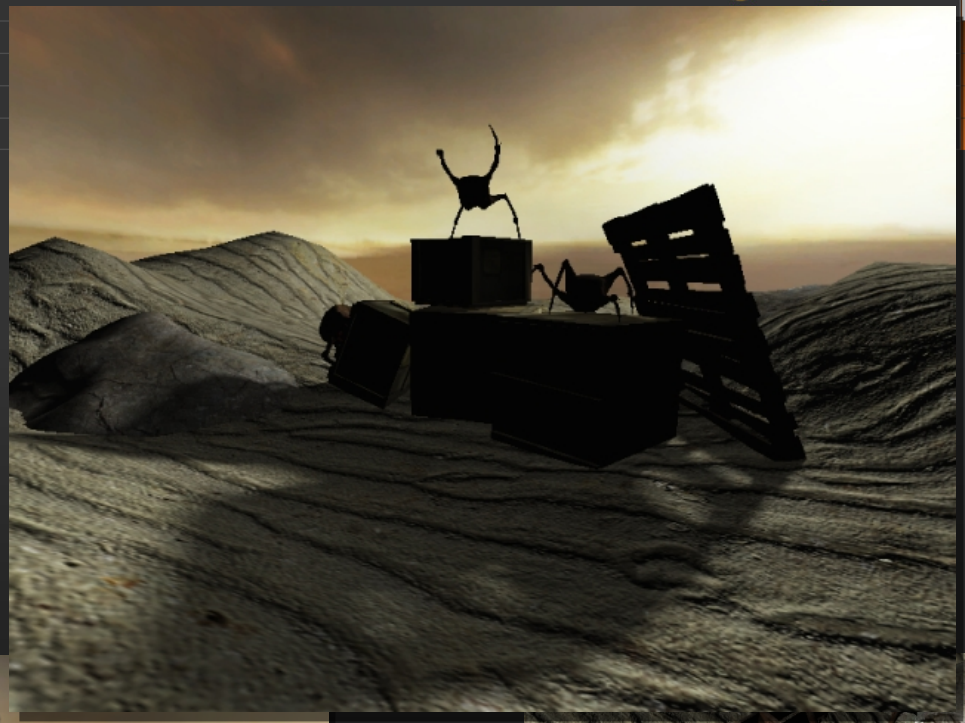


Perspectives

The approximation of the occluder geometry may be insufficient when the occlusion area is concentrated in a part of the bounding rectangle

- The center of gravity of the light occlusion area could be used to weight the bounding rectangles intersection area





Thanks to Valve software who authorized us to use the models and the materials of the Half-life² video game