Shadow Silhouette Maps

Eric Chan

Massachusetts Institute of Technology

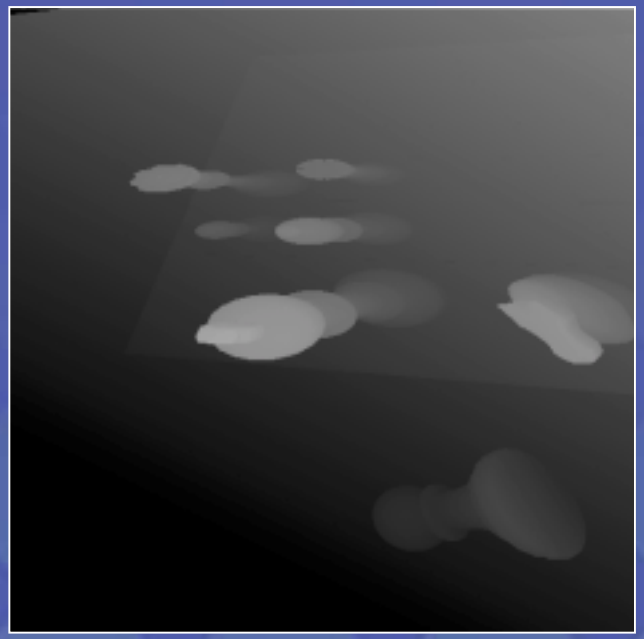
Game Plan

Motivation
Algorithm
Implementation
Examples
Comparison

Motivation

Why another shadow algorithm?

Why not use perspective shadow maps?



Stamminger and Drettakis, SIGGRAPH 2002

Perspective Shadow Maps

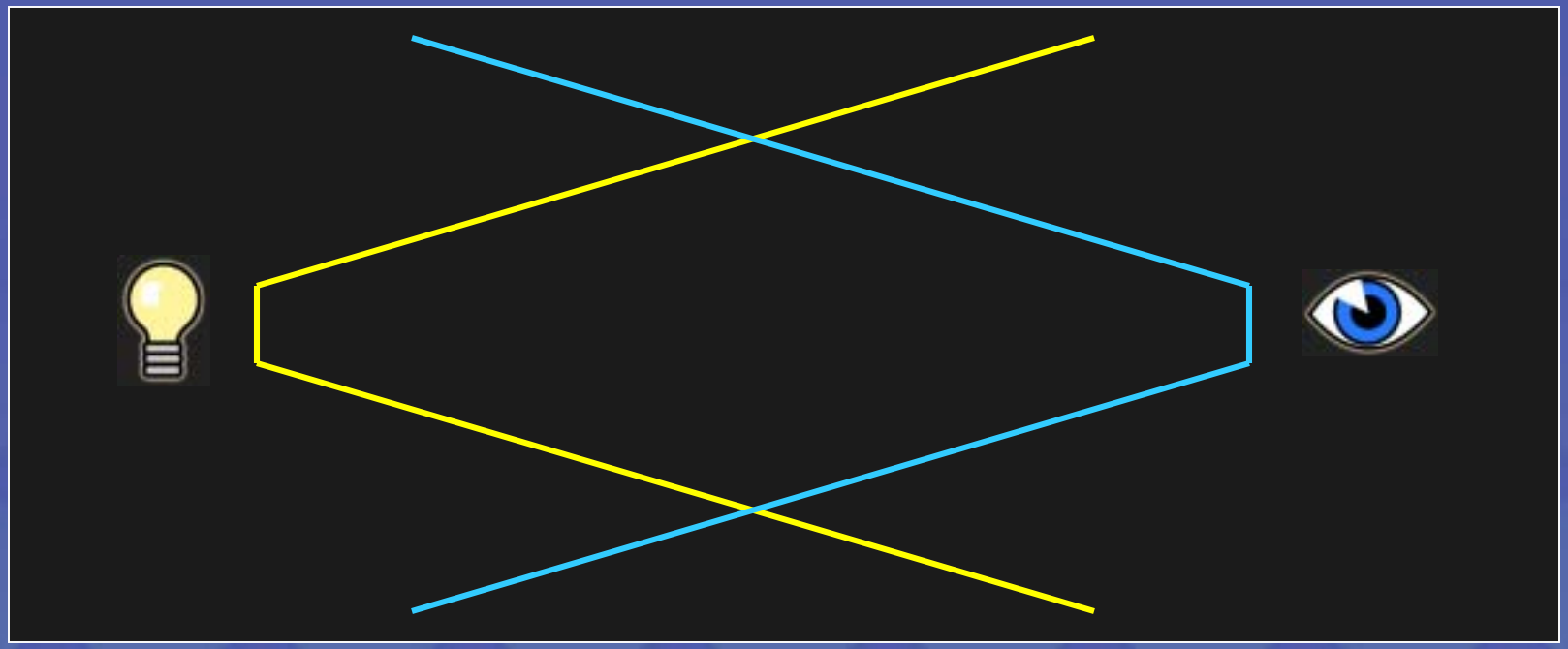
Addresses perspective aliasing

Optimizes distribution of depth samples

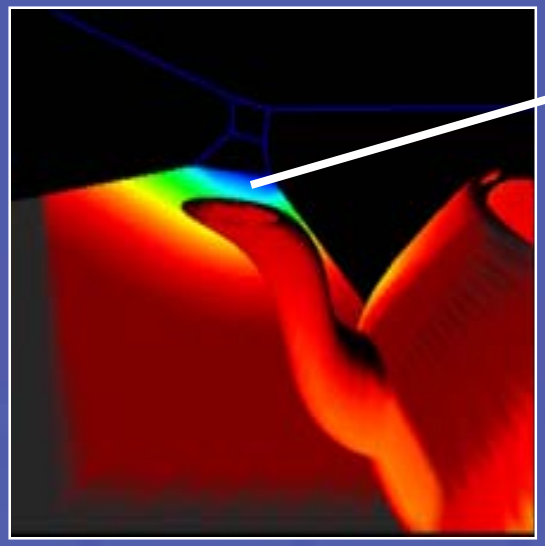
Difficulties:

- Does not handle projection aliasing
- Dueling frusta problem

Dueling Frusta Problem

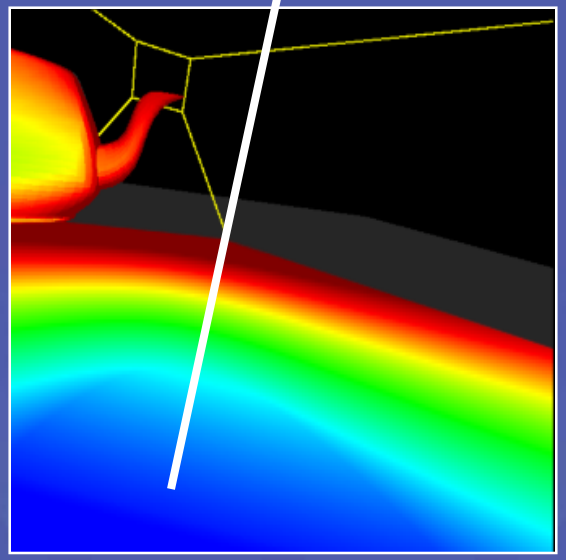


Dueling Frusta Problem

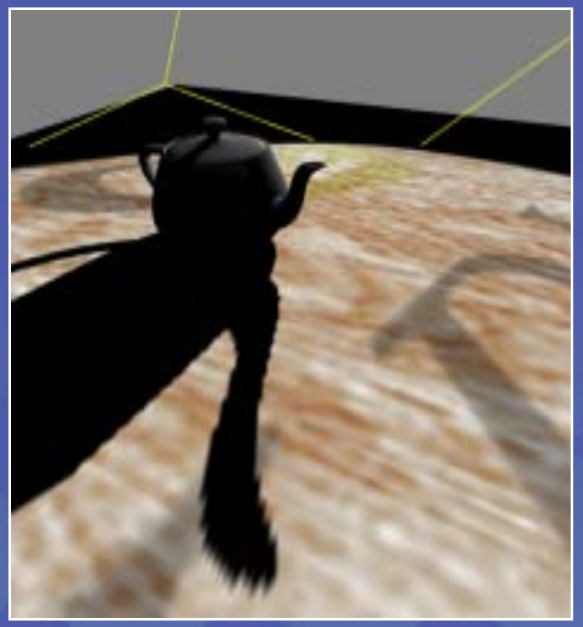


light's view

high magnification



eye's view



aliasing artifacts

Mark Kilgard, NVIDIA

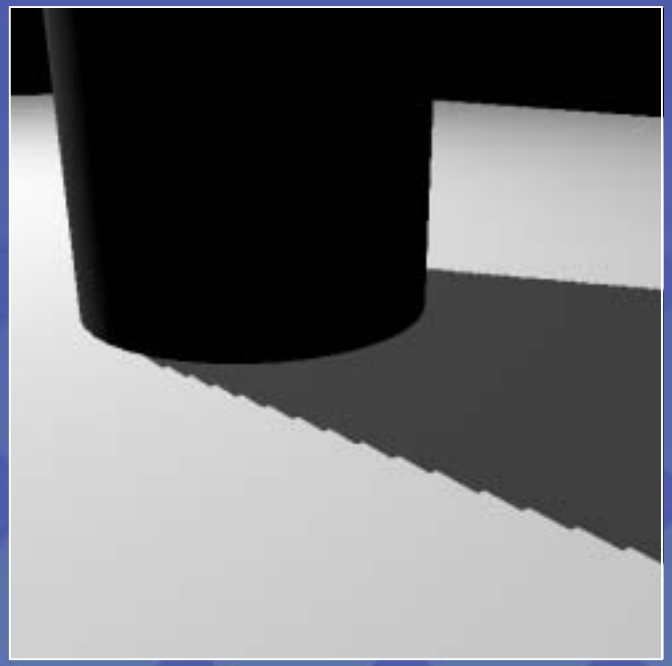
Shadow Silhouette Maps

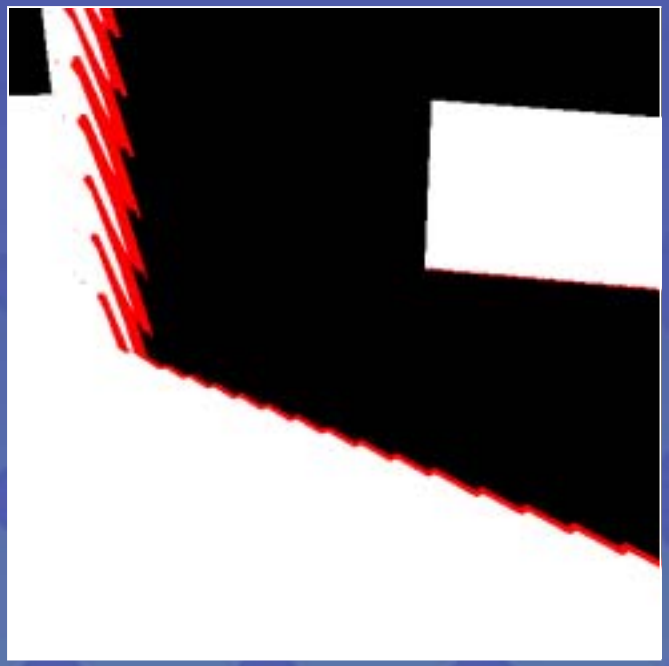
- Research at Stanford University
  - P. Sen, M. Cammarano, and P. Hanrahan
  - Proceedings of SIGGRAPH 2003
- See course notes
- Also available online

Observation

Shadow maps

- undersampling can occur anywhere
- artifacts visible only at shadow edges





Observation

Shadow volumes

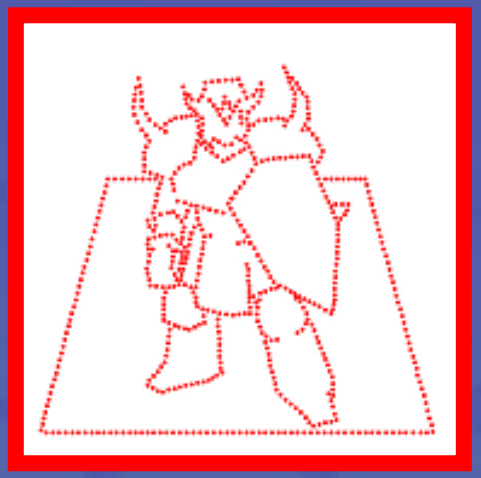
- accurate everywhere, but high fillrate
- accuracy only needed at silhouettes

Algorithm Goals

- Accuracy of shadow volumes
- Efficiency of shadow maps
- Treats perspective and projection aliasing
- Supports dynamic scenes
- Maps to graphics hardware

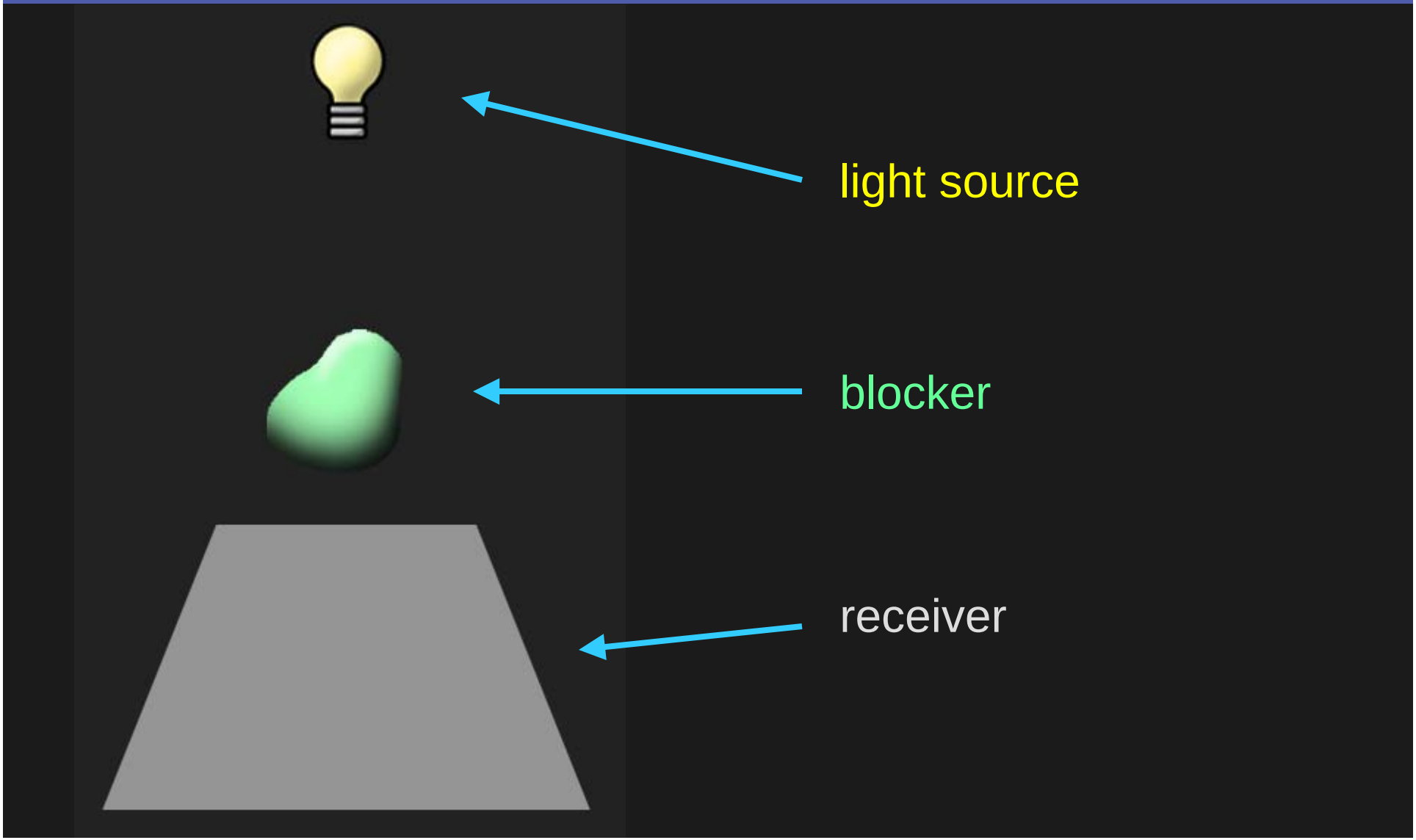
Overview

depth map

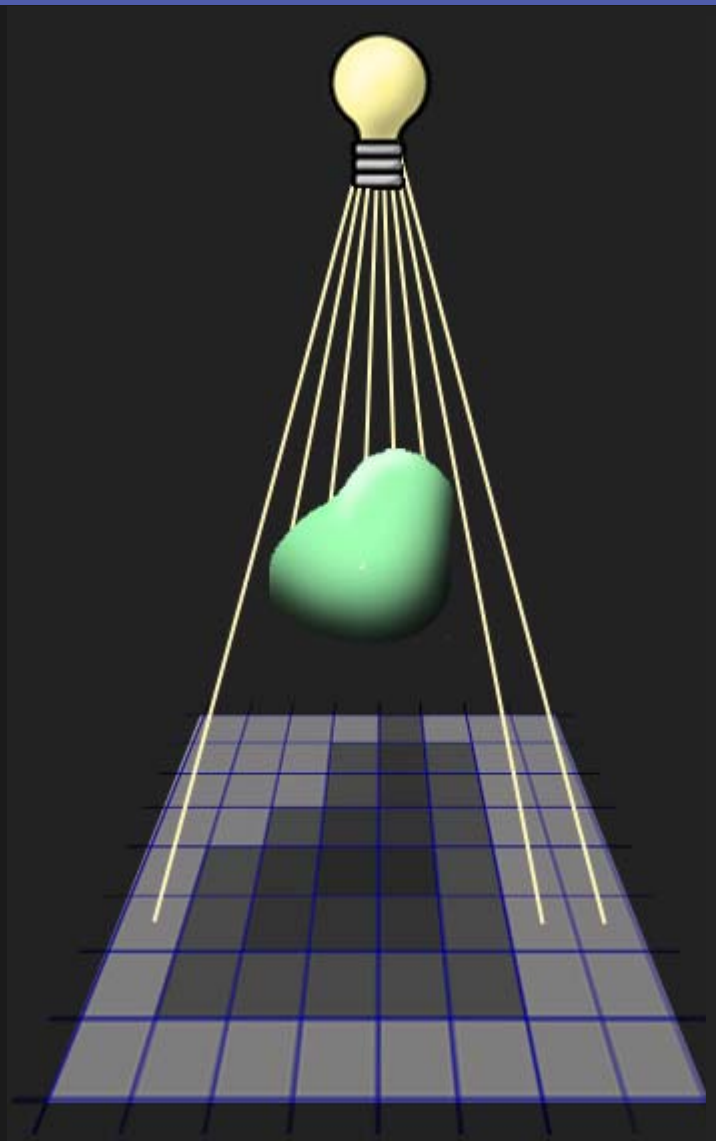


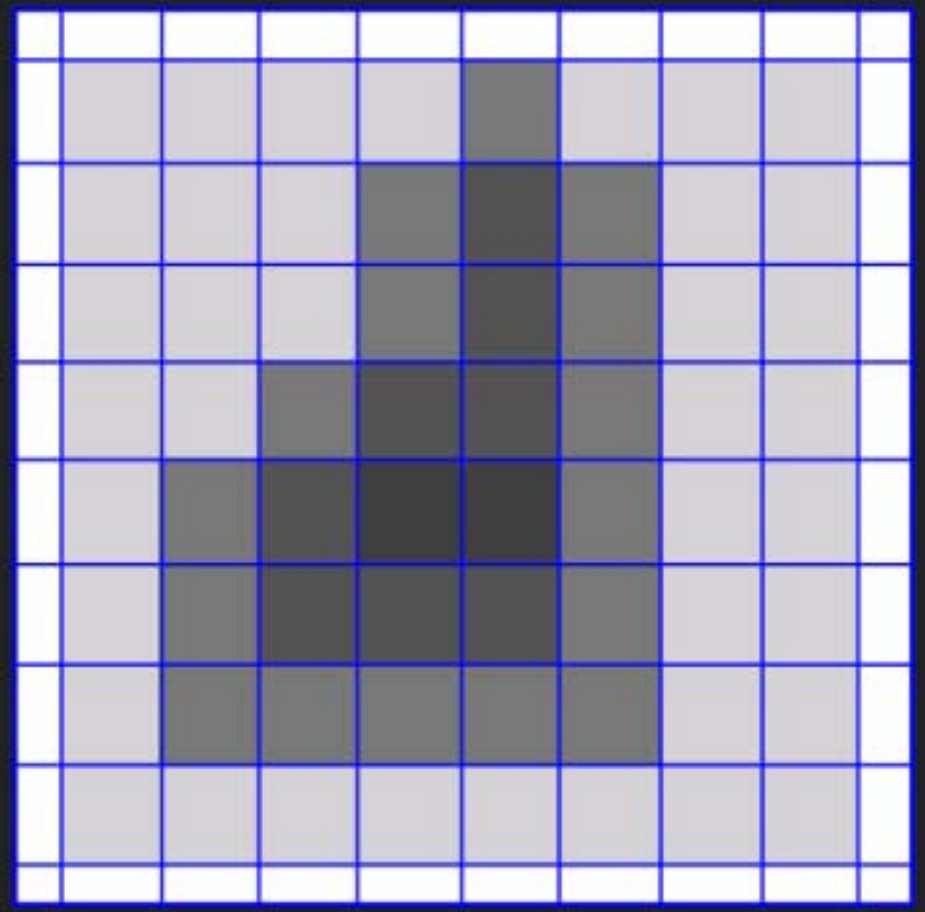
silhouette map

Shadow Map (Review)



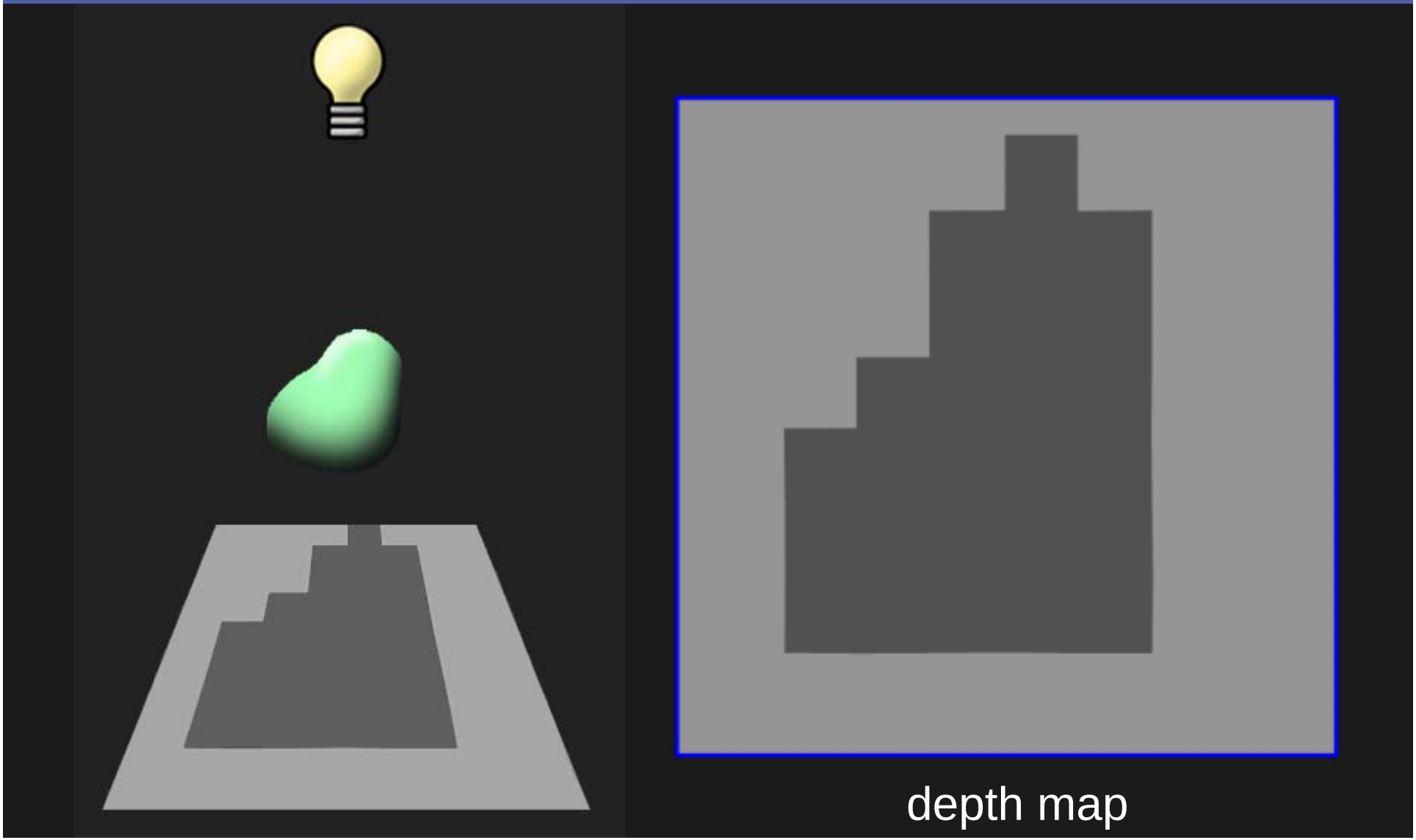
Shadow Map (Review)



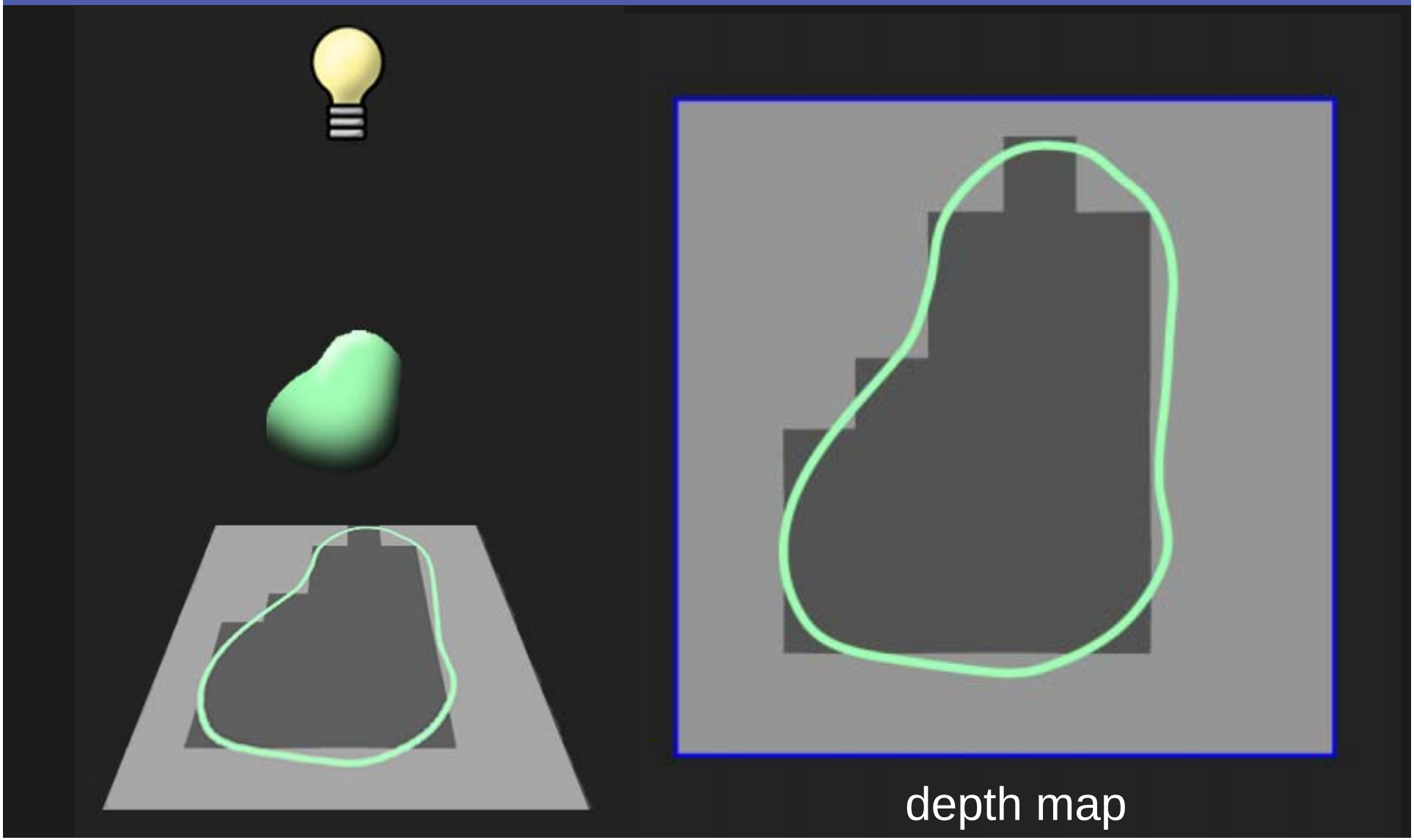


depth map

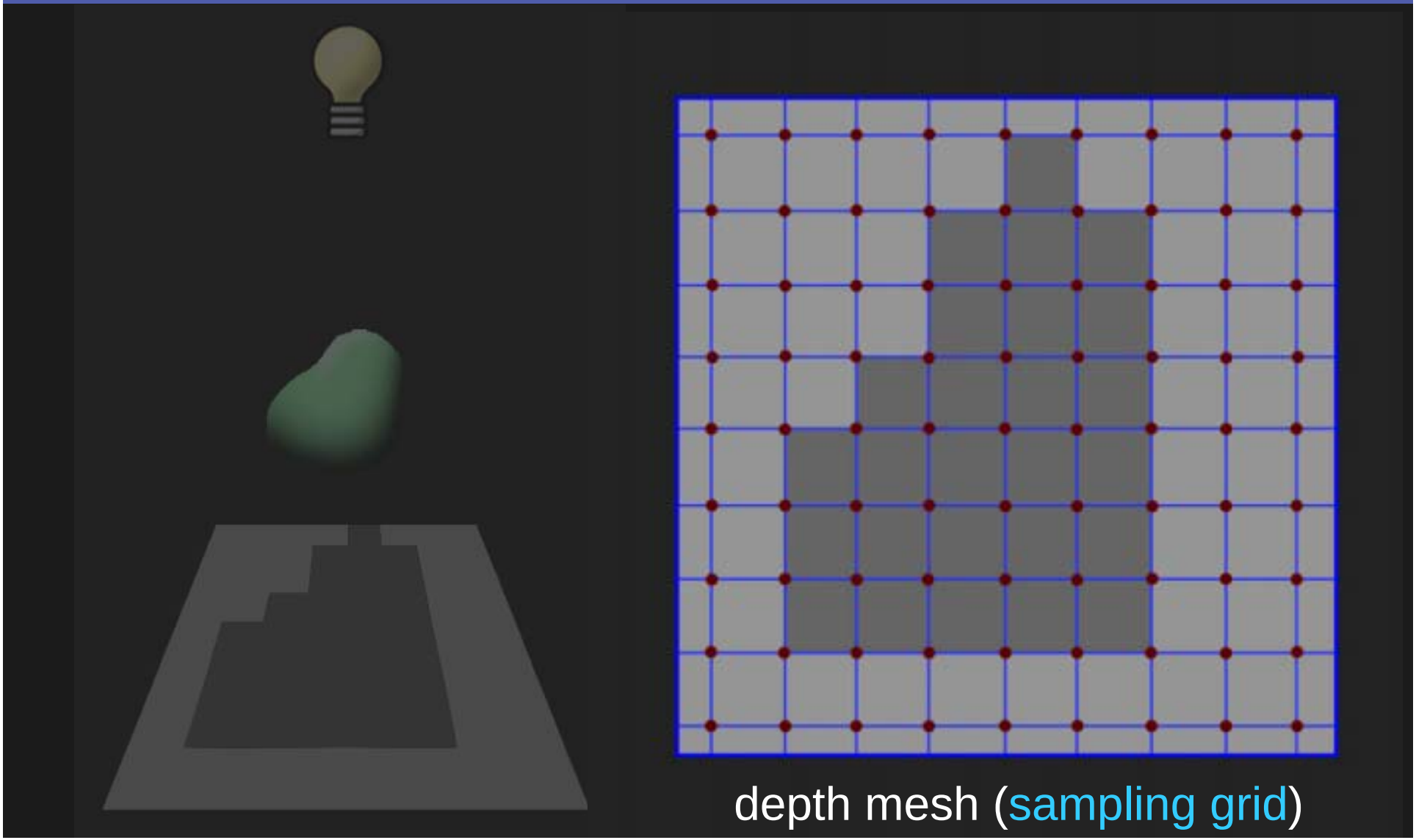
Shadow Map (Review)



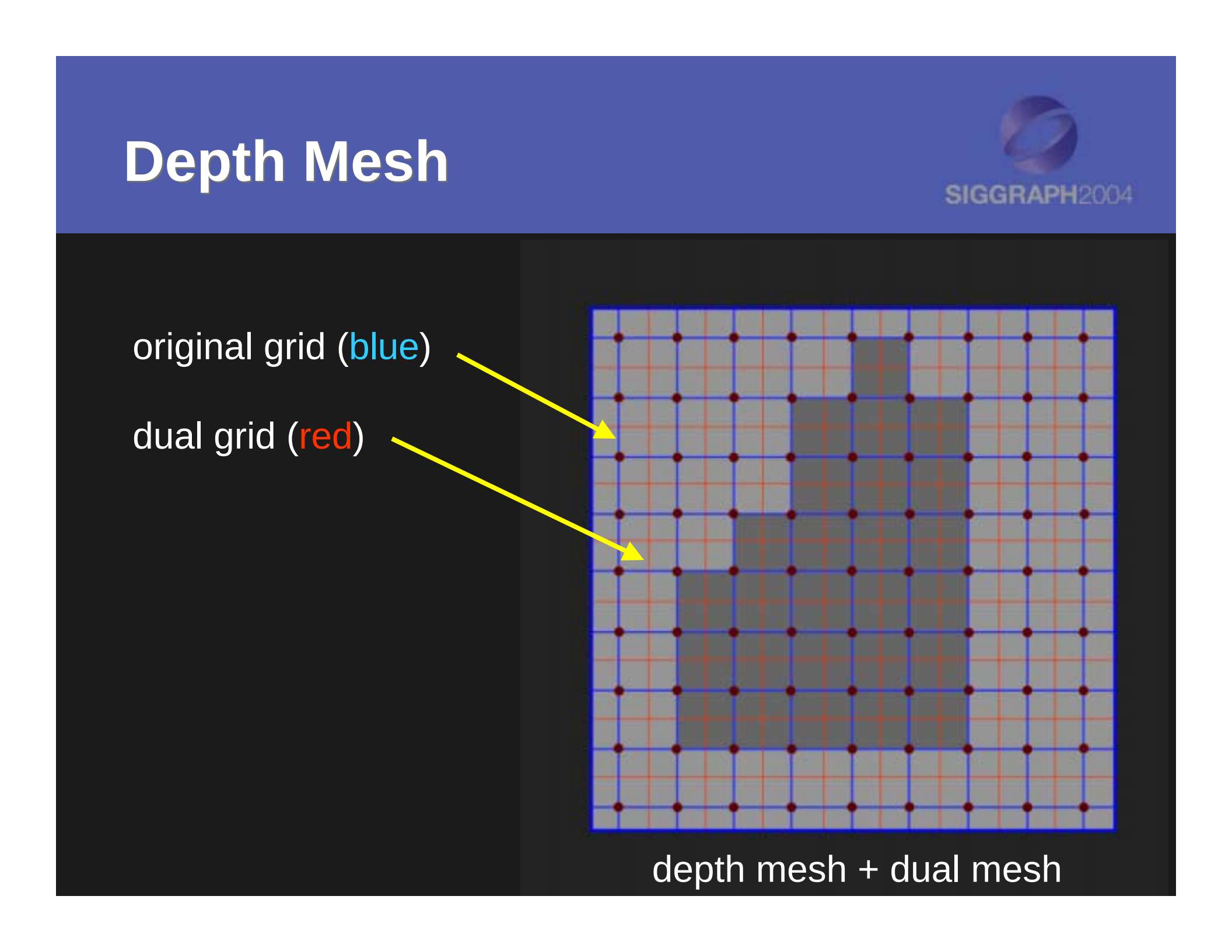
Shadow Map (Review)



Depth Mesh



Depth Mesh



original grid (blue)

dual grid (red)

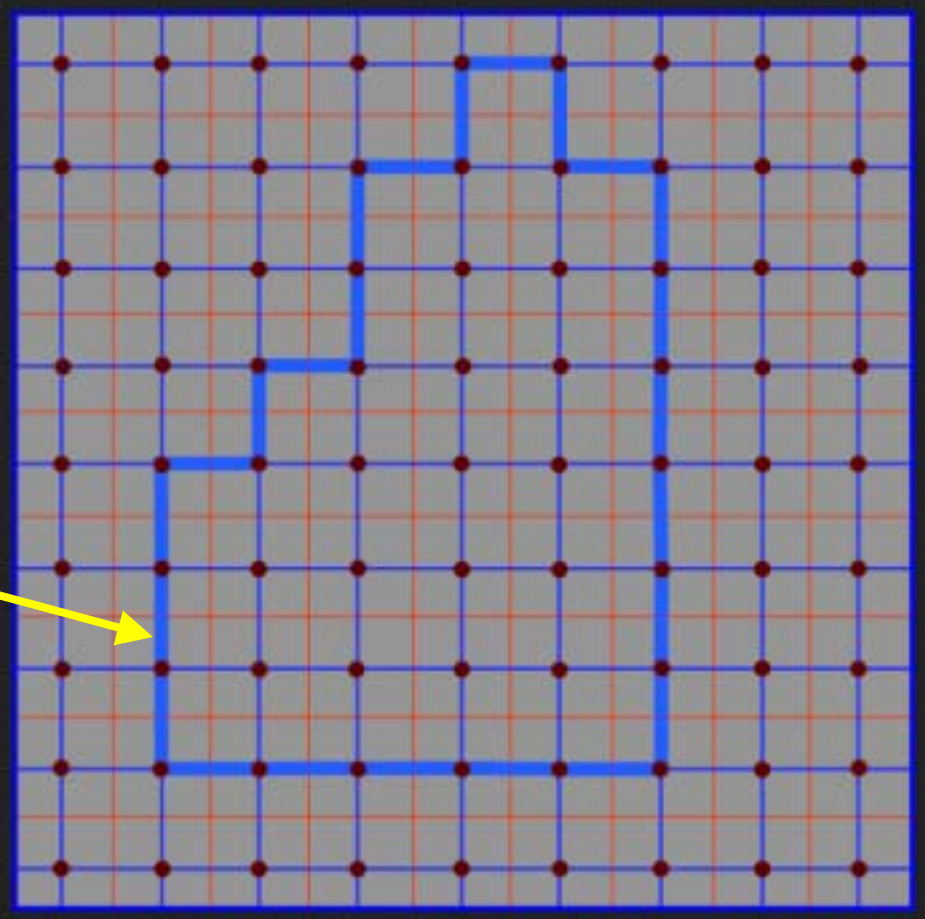
depth mesh + dual mesh

Depth Mesh

original grid (blue)

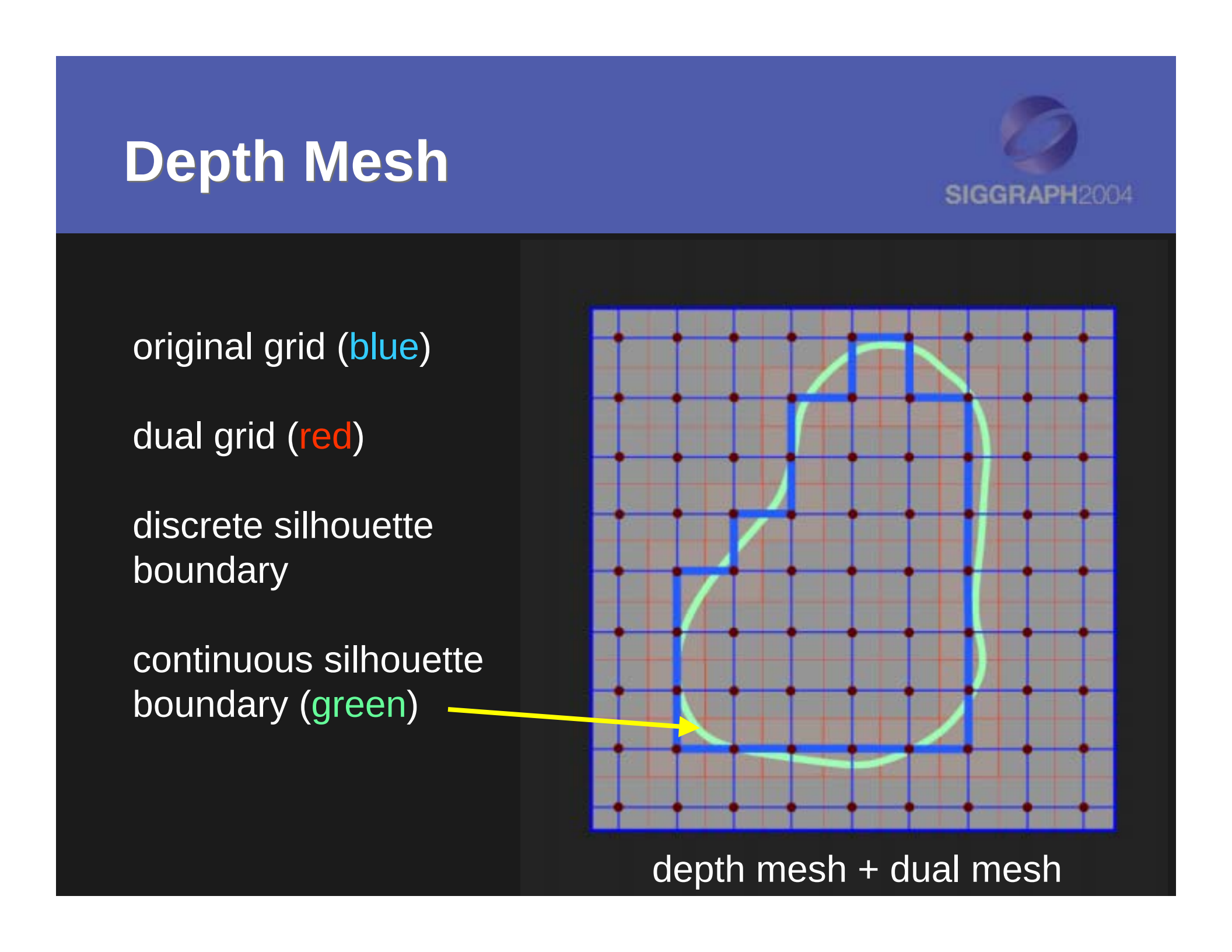
dual grid (red)

discrete silhouette boundary



depth mesh + dual mesh

Depth Mesh



original grid (blue)

dual grid (red)

discrete silhouette boundary

continuous silhouette boundary (green)

depth mesh + dual mesh

Depth Mesh

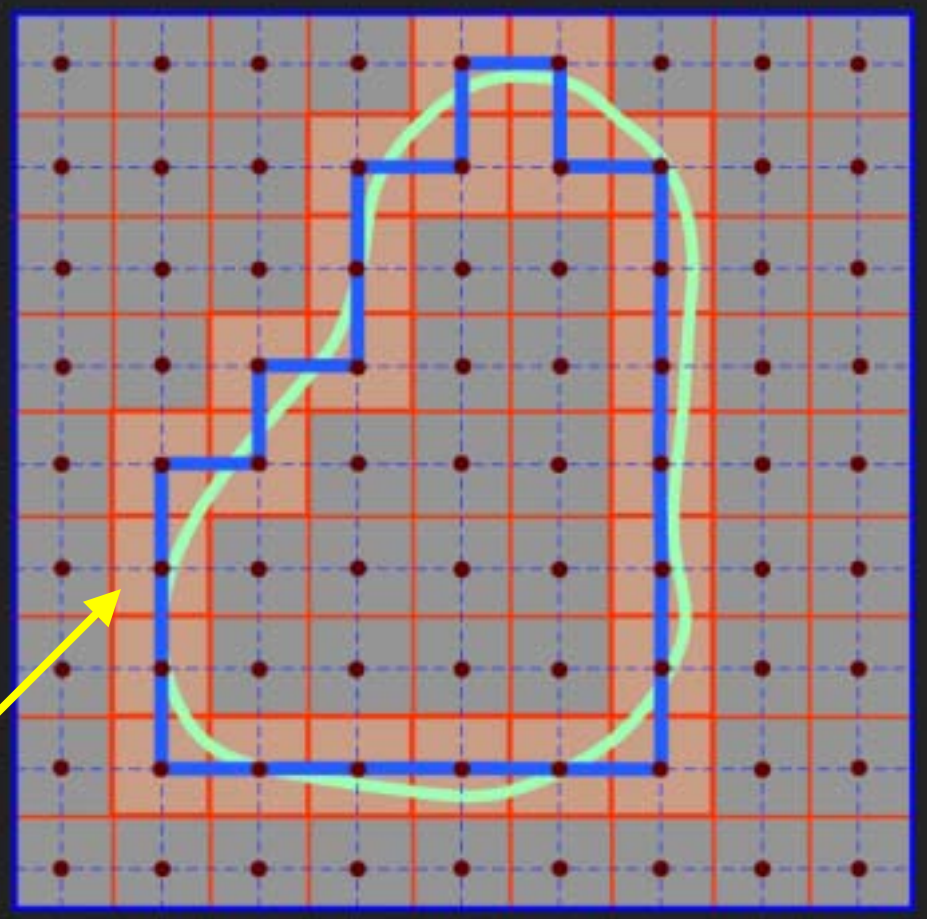
original grid (blue)

dual grid (red)

discrete silhouette boundary

continuous silhouette boundary (green)

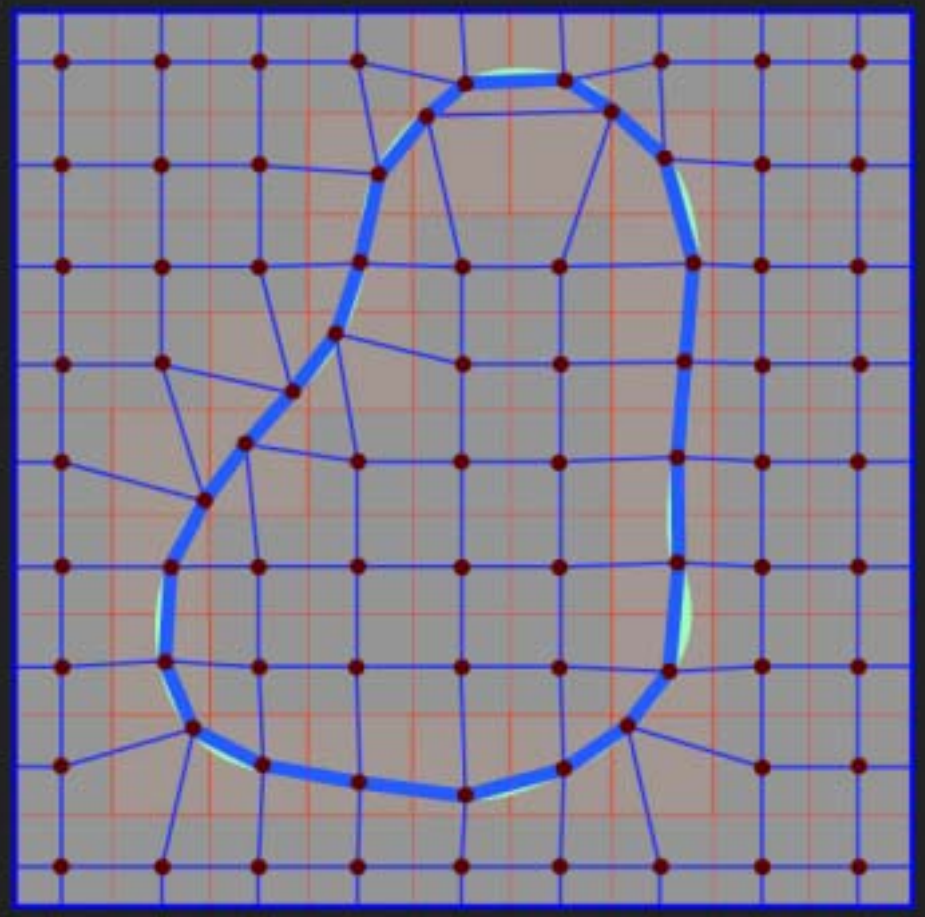
silhouette map pixels



depth mesh + dual mesh

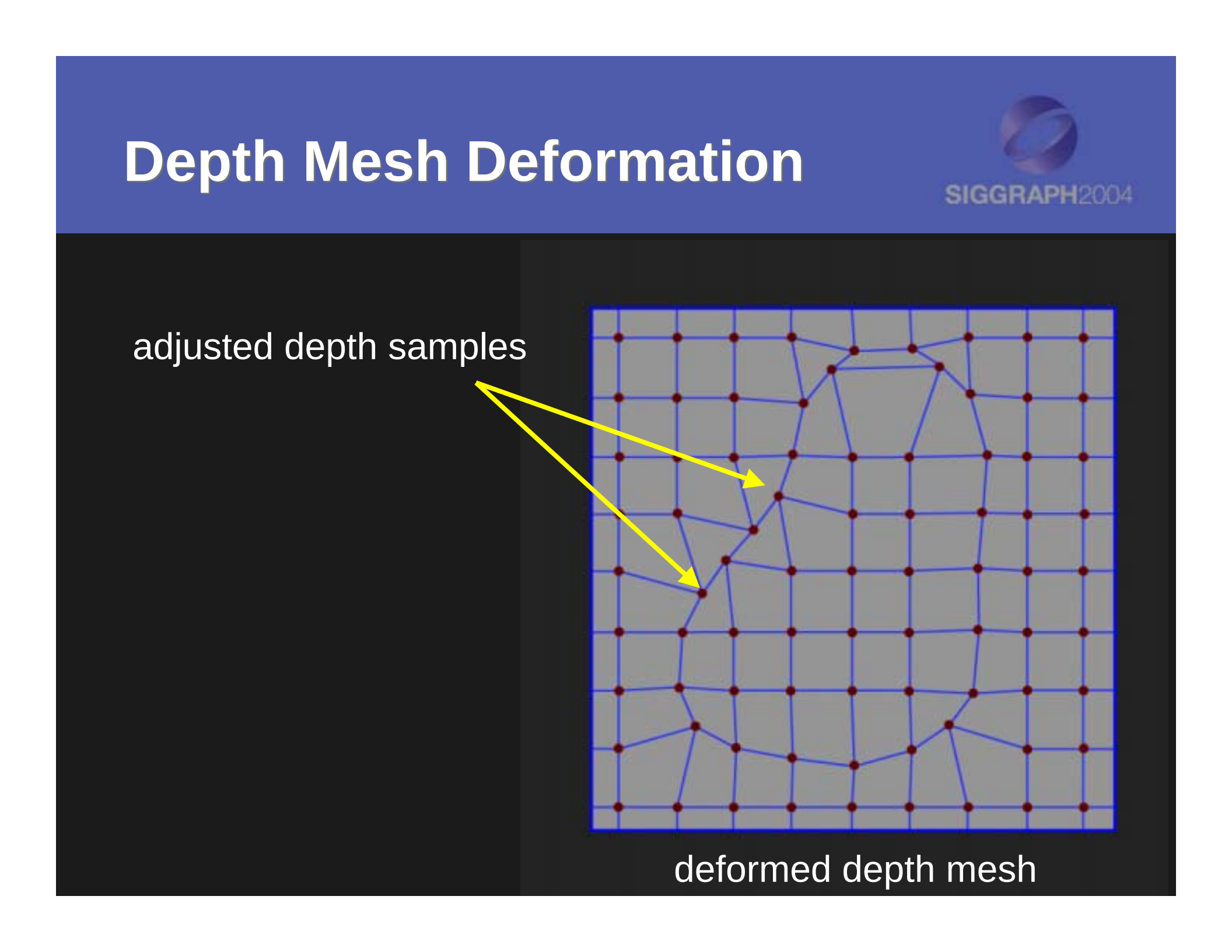
Depth Mesh Deformation

Move depth samples
to lie on silhouette curve

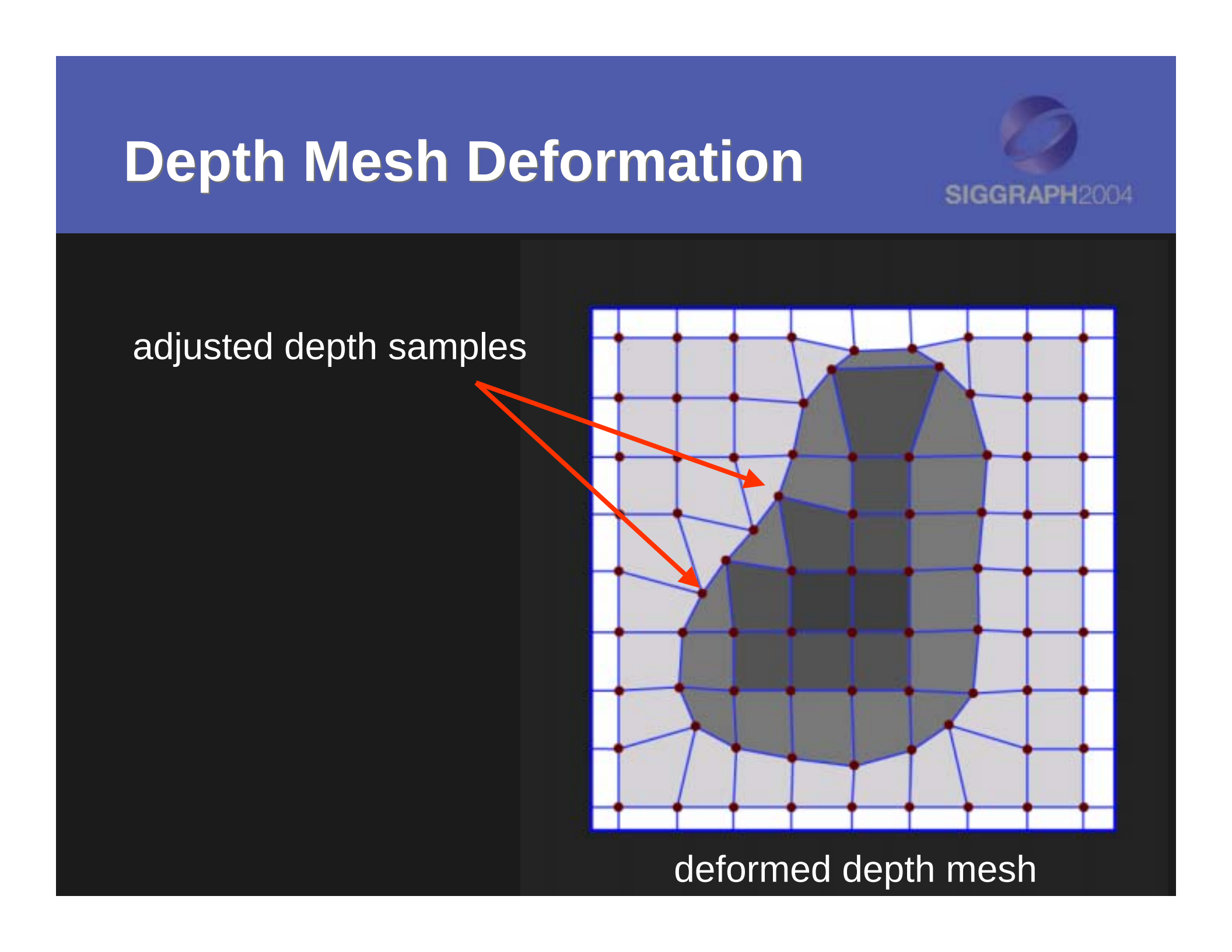


deformed depth mesh

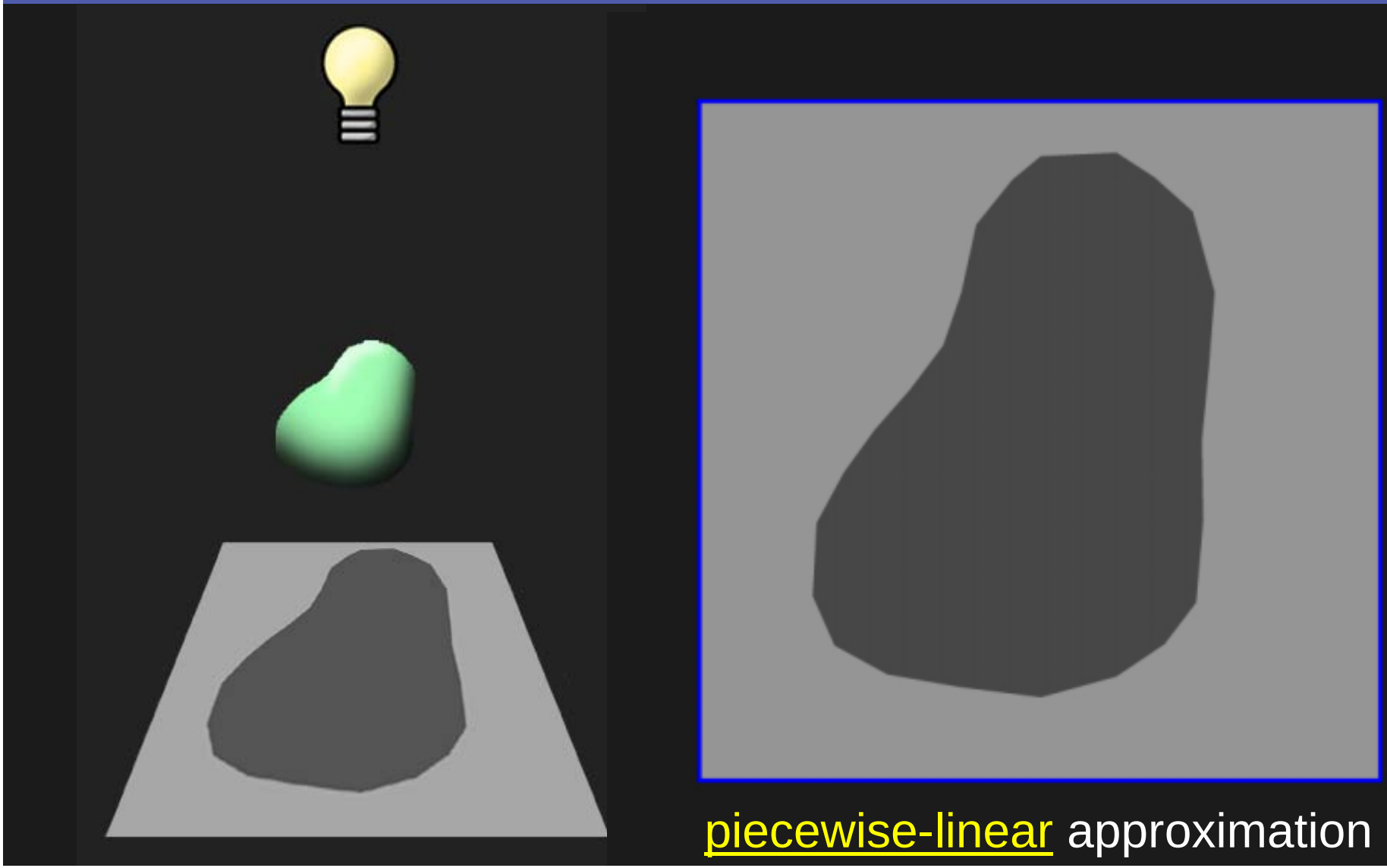
Depth Mesh Deformation



Depth Mesh Deformation



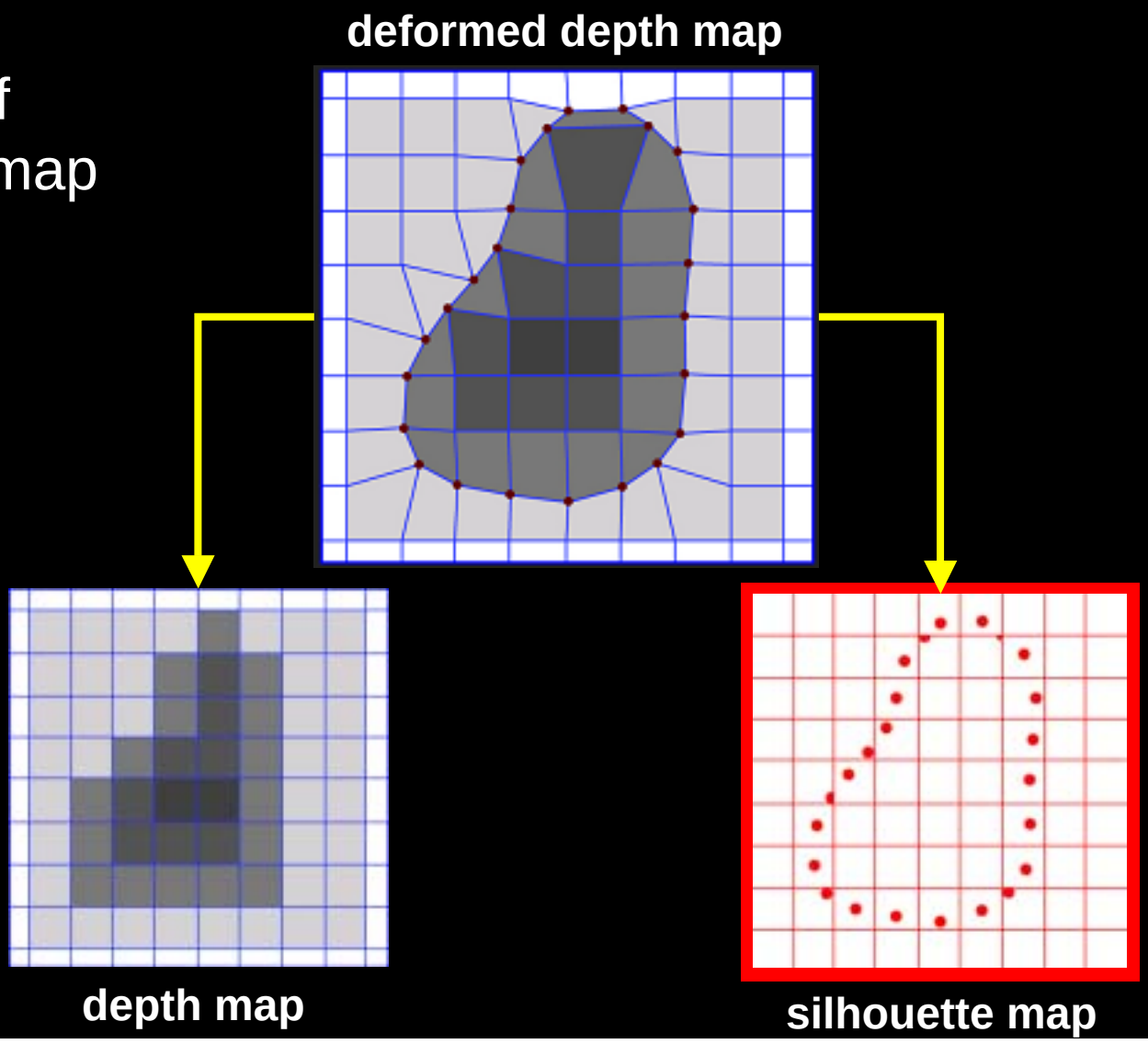
Better Approximation



piecewise-linear approximation

Silhouette Map

Decomposition of deformed depth map



What is a Silhouette Map?

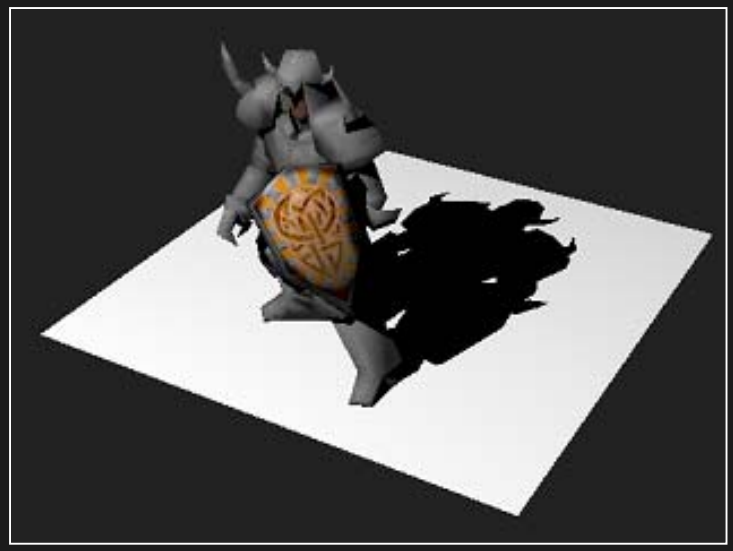
Many ways to think about it:

- Edge representation
- 2D image, same resolution as depth map
- Offset from depth map by ½ pixel in x, y
- Stores xy-coordinates of silhouette points
- Stores only one silhouette point per texel
- Piecewise-linear approximation

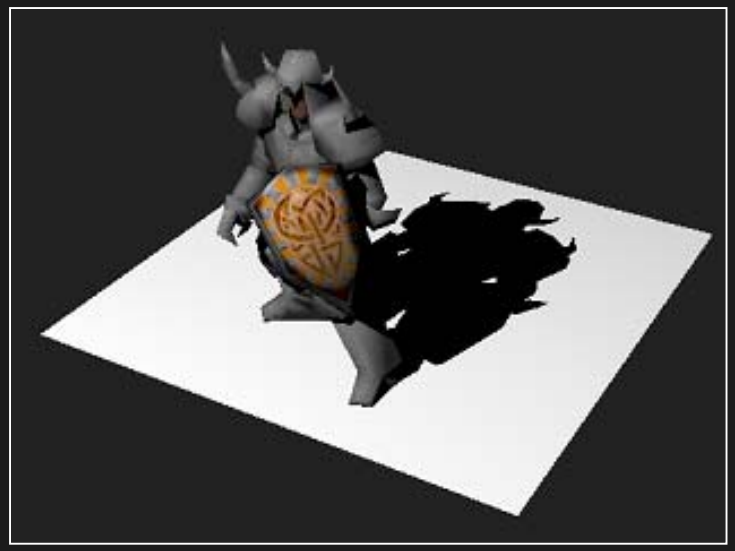
Algorithm

Algorithm Overview

Image-space algorithm



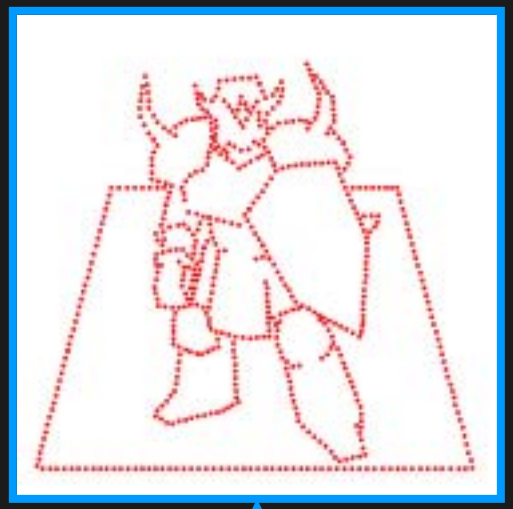
Algorithm Overview

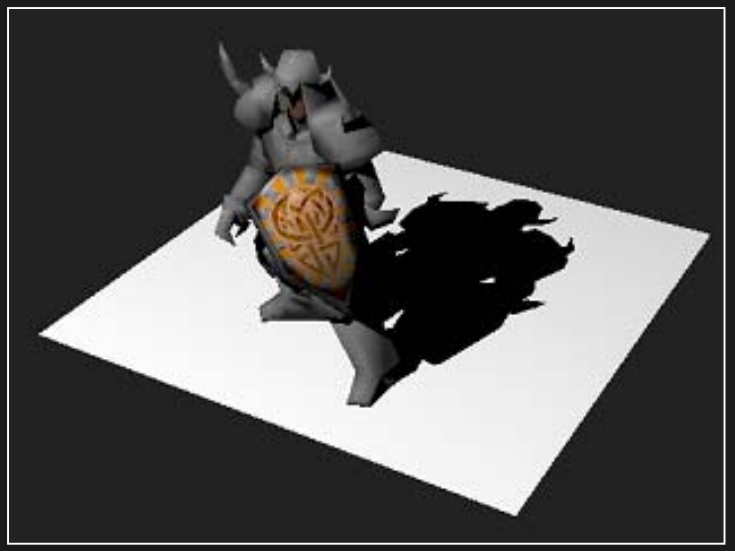


Step 1

Create depth map

Algorithm Overview

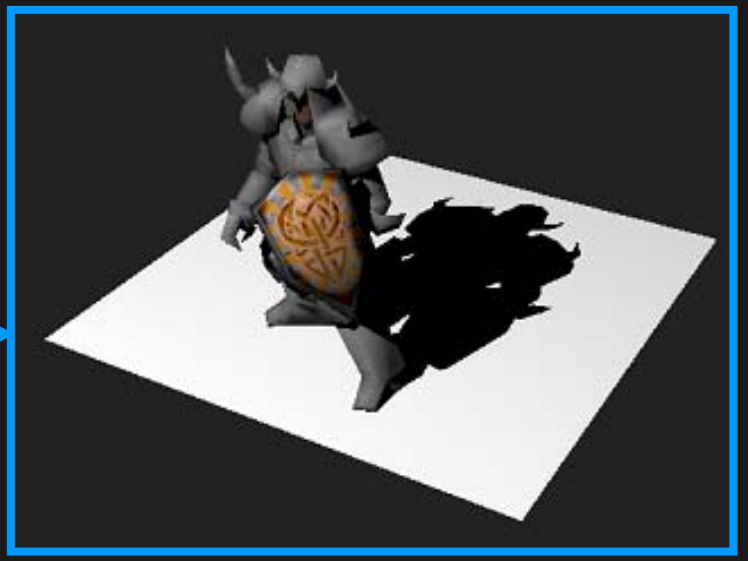




Step 2

Create silhouette map

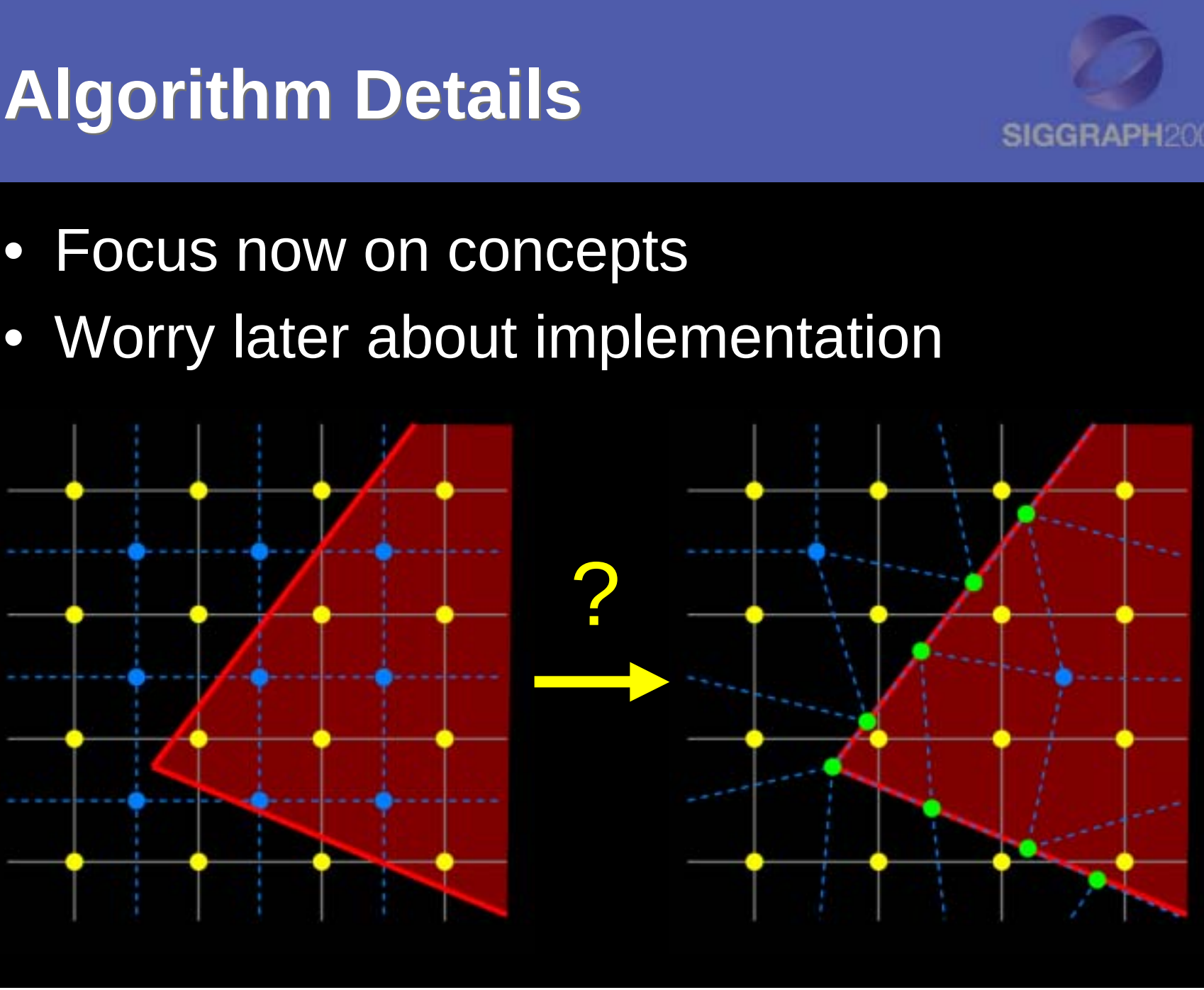
Algorithm Overview



Step 3

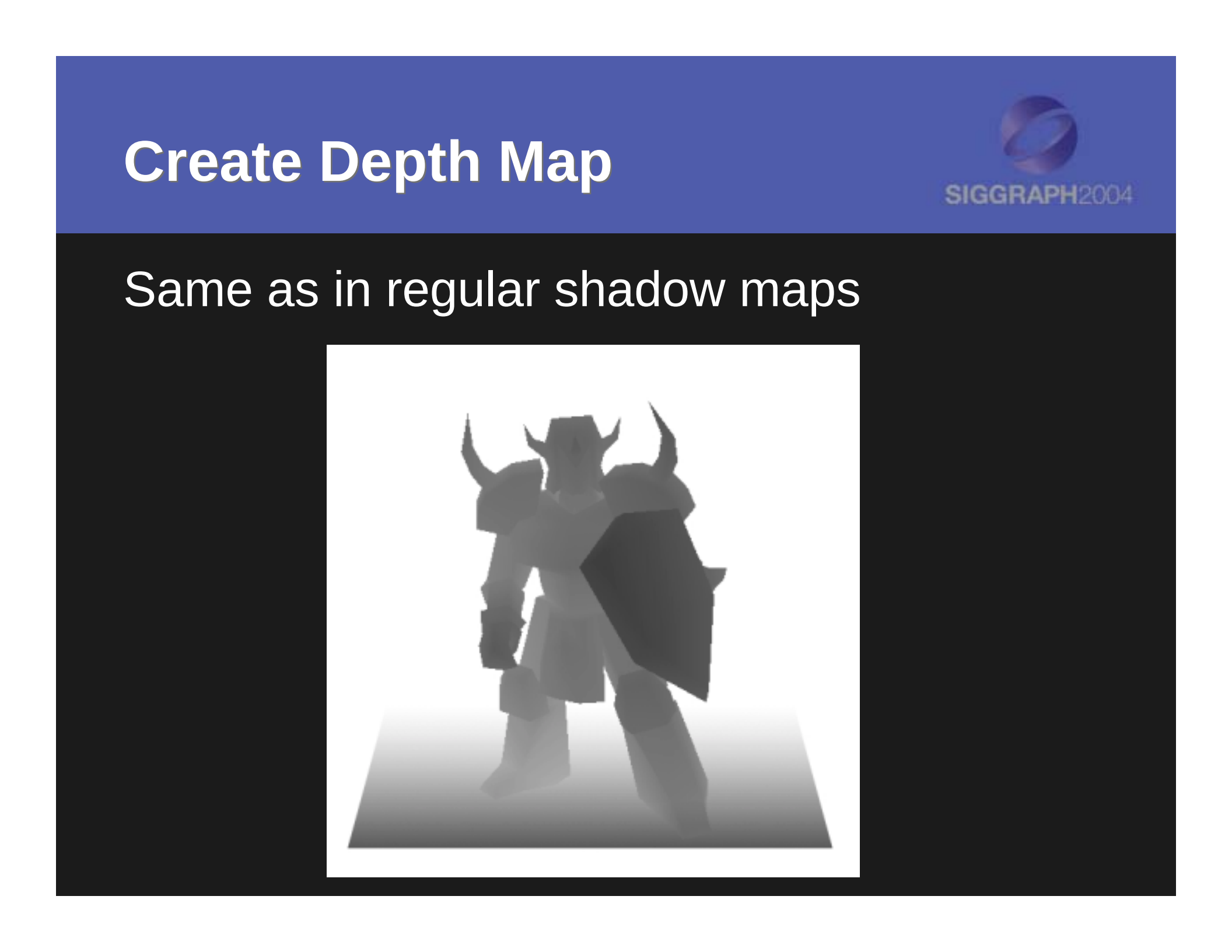
Render scene and shadows

Algorithm Details



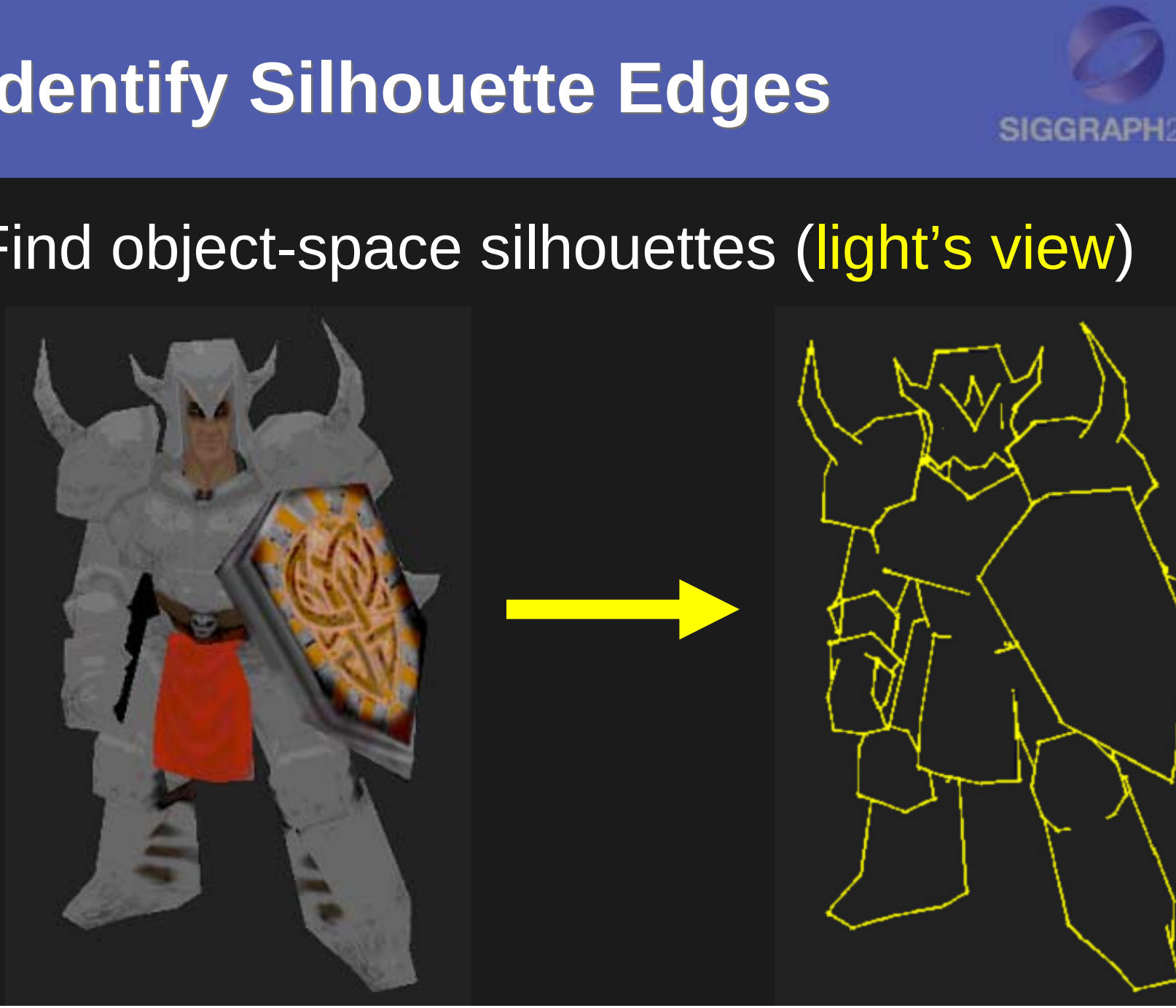
- Focus now on concepts
- Worry later about implementation

Create Depth Map



Same as in regular shadow maps

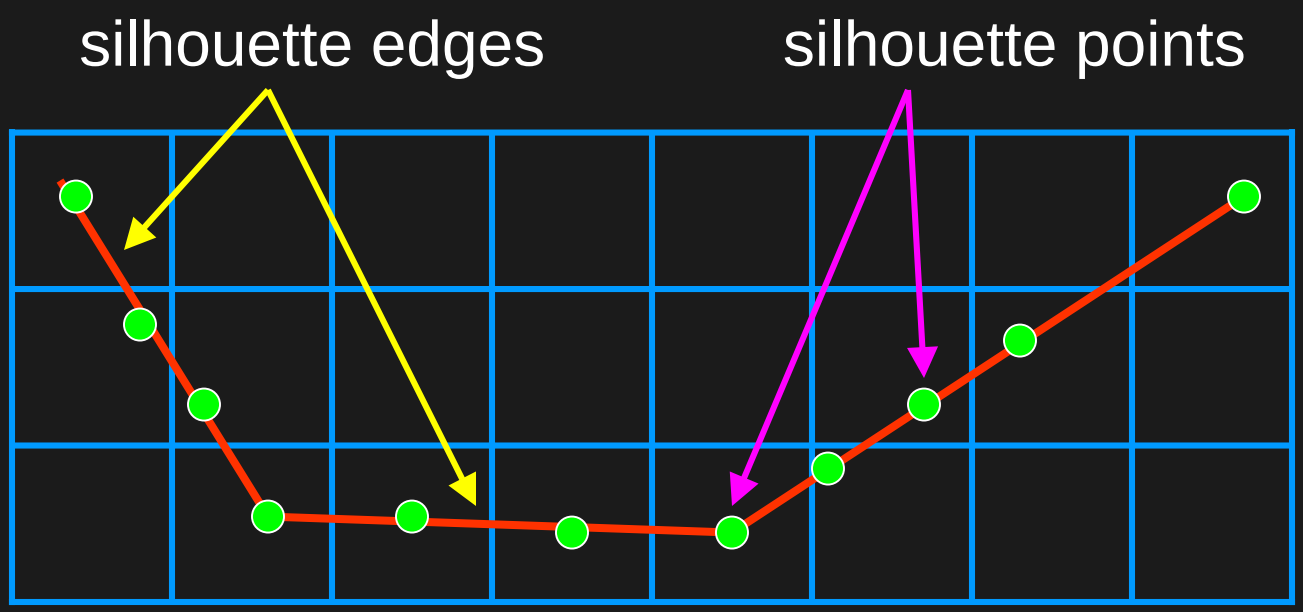
Identify Silhouette Edges



Find object-space silhouettes (light's view)

Create Silhouette Map

- Rasterize silhouette edges (light's view)
- Find points that lie on silhouette edges
- Store one such point per texel

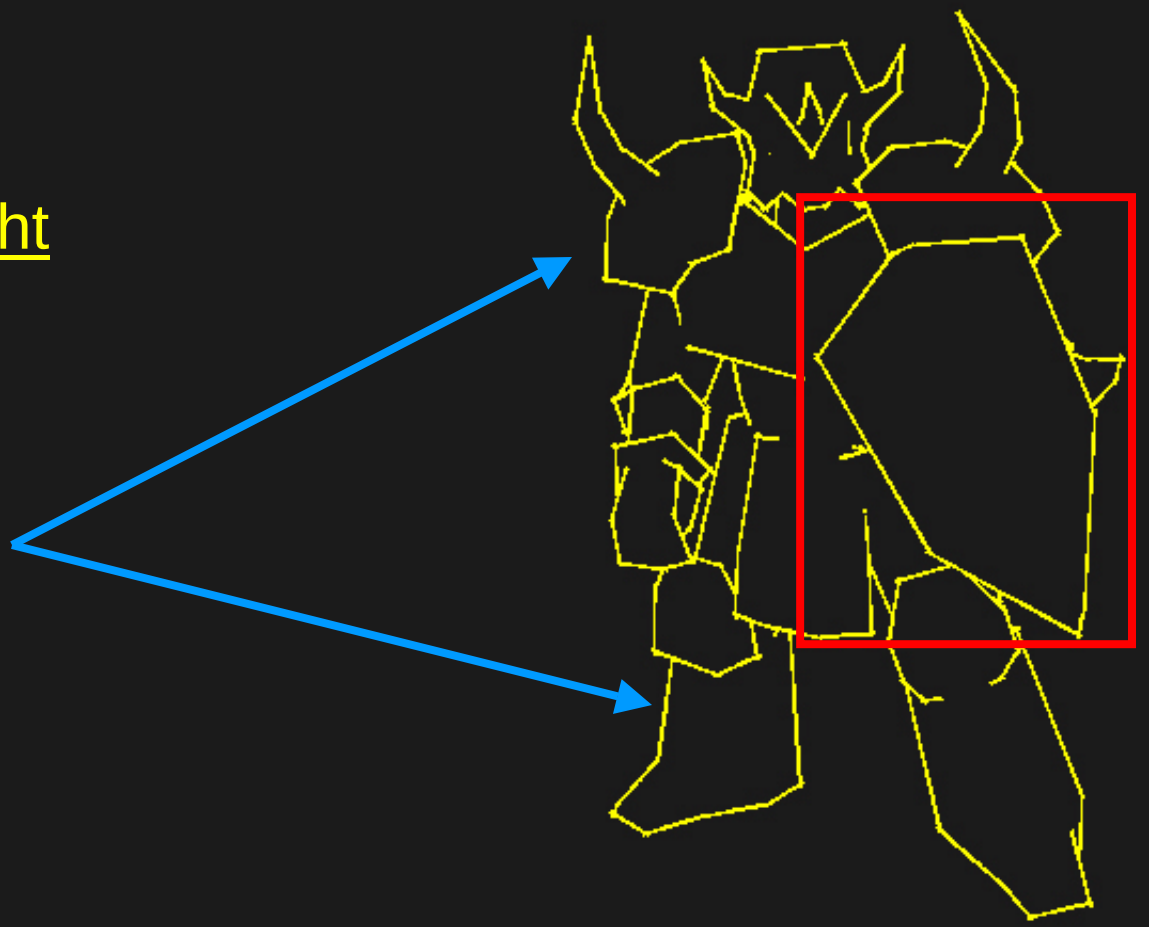


Compute Silhouette Points

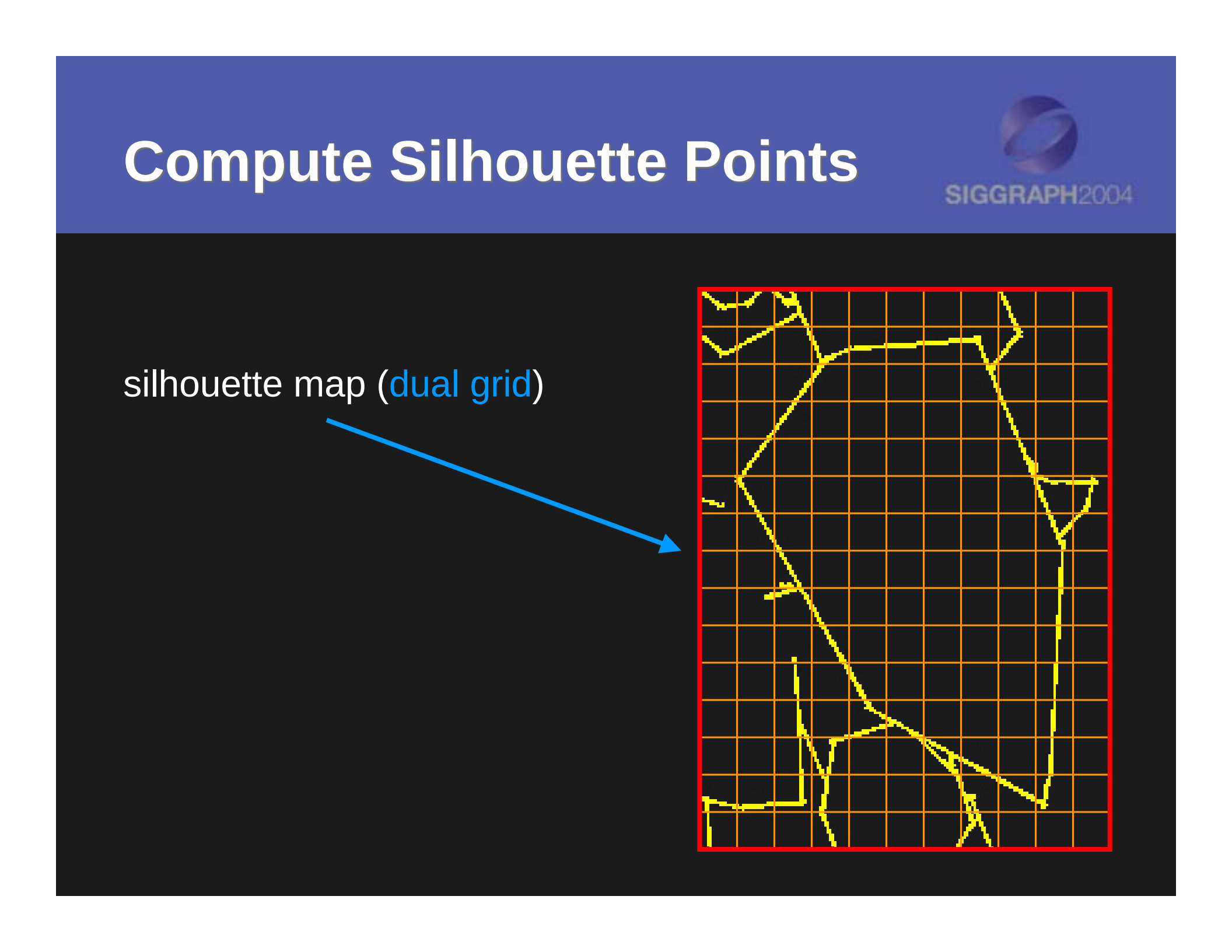
Example:

point of view of light

silhouette edges

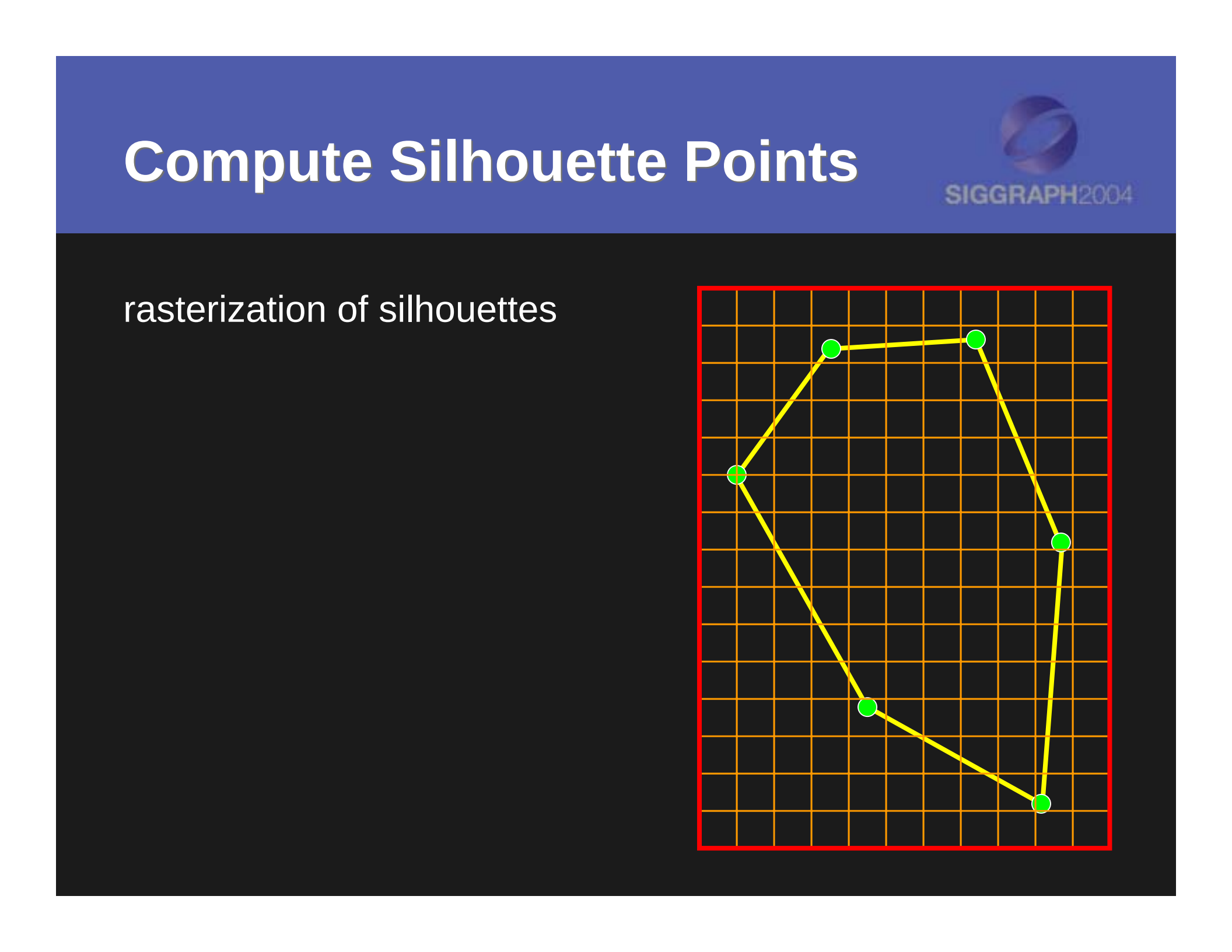


Compute Silhouette Points



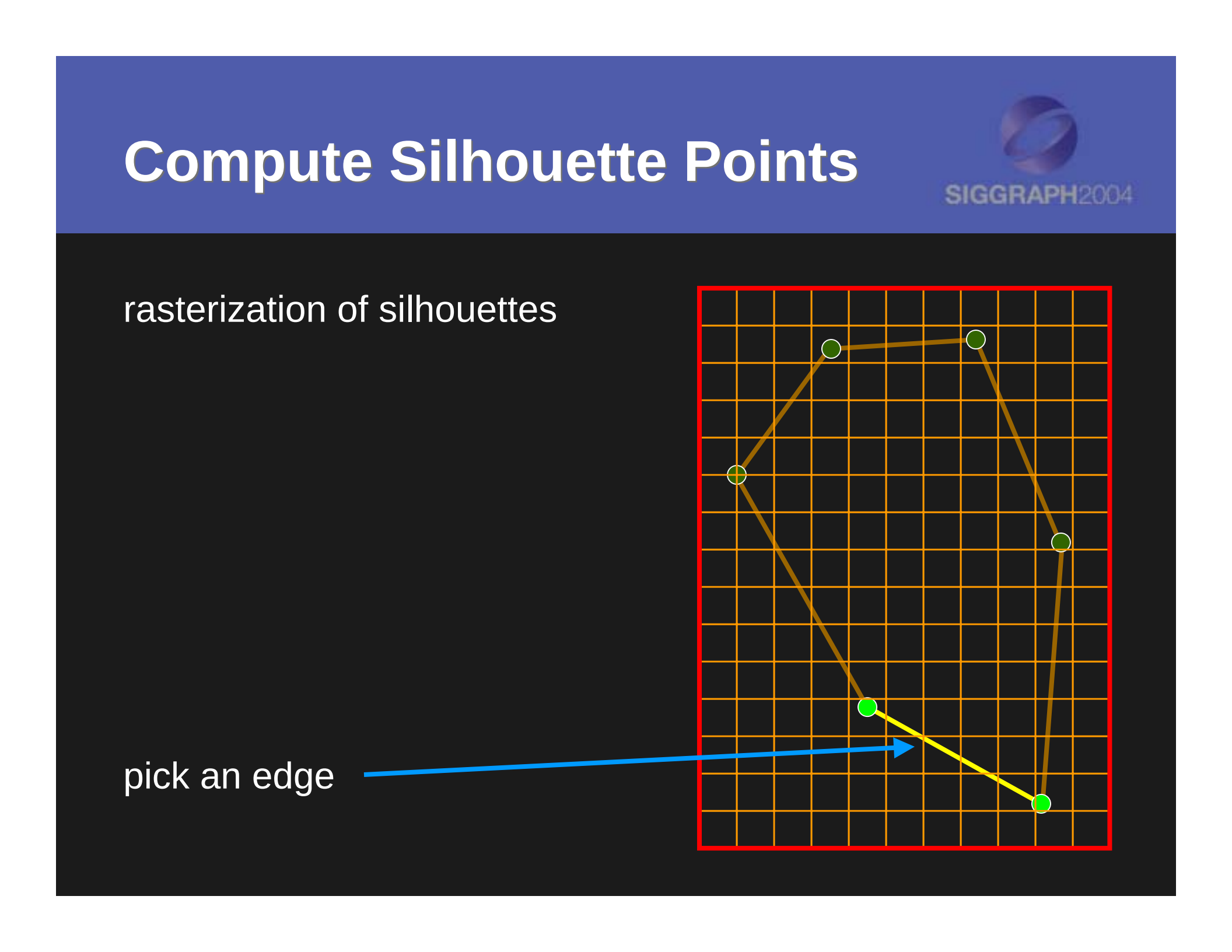
silhouette map (dual grid)

Compute Silhouette Points



rasterization of silhouettes

Compute Silhouette Points



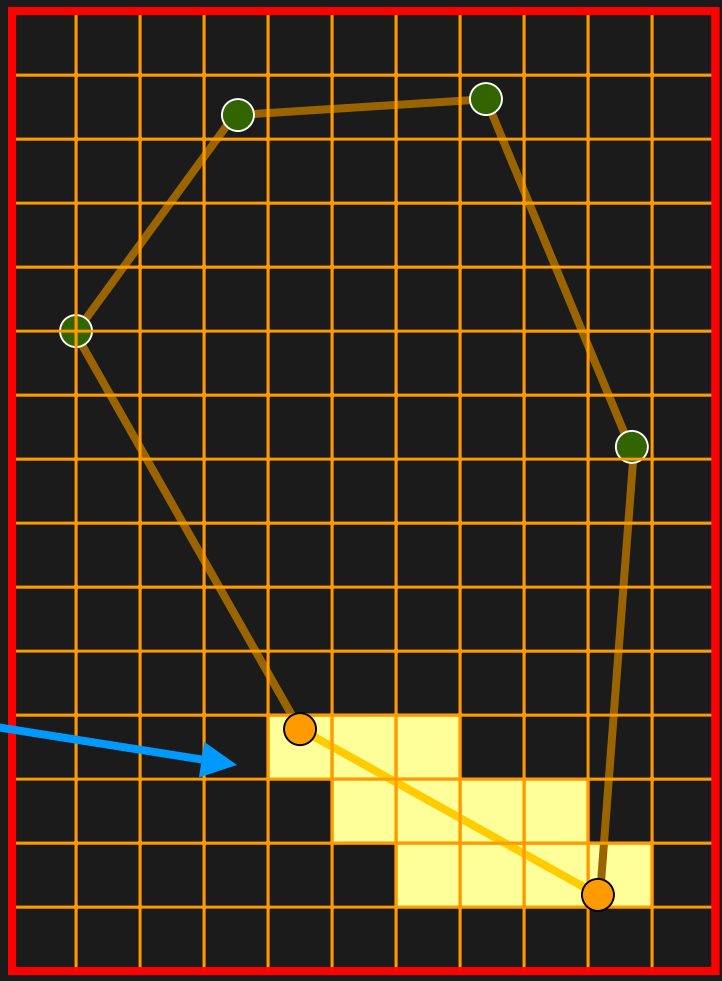
rasterization of silhouettes

pick an edge

Compute Silhouette Points

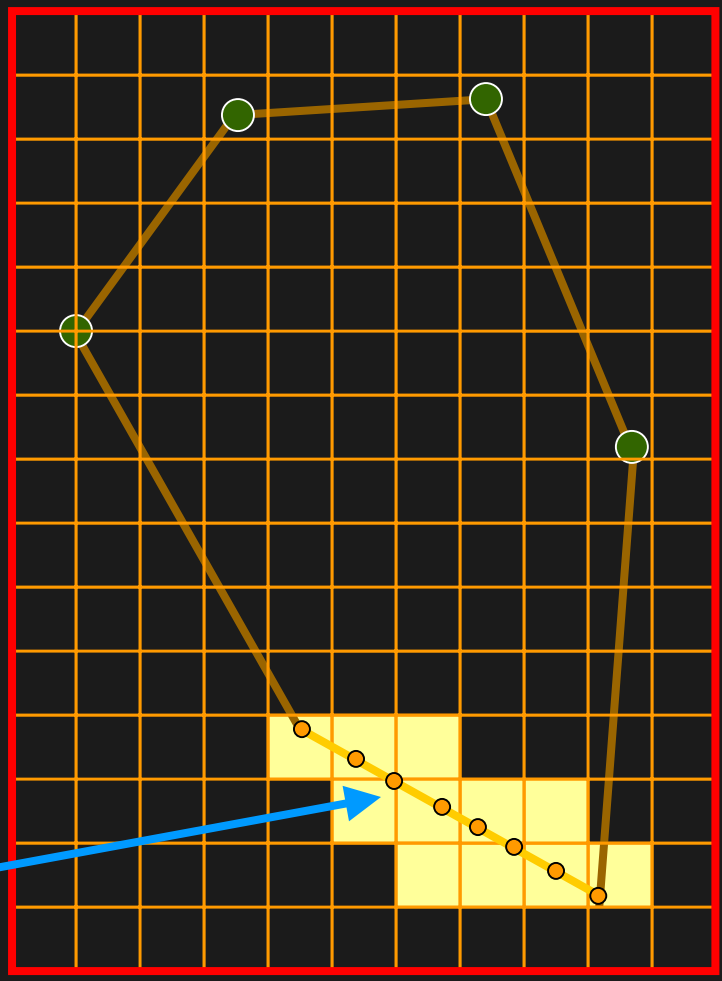
rasterization of silhouettes

rasterize edge conservatively:
be sure to generate fragments
for silhouette pixels



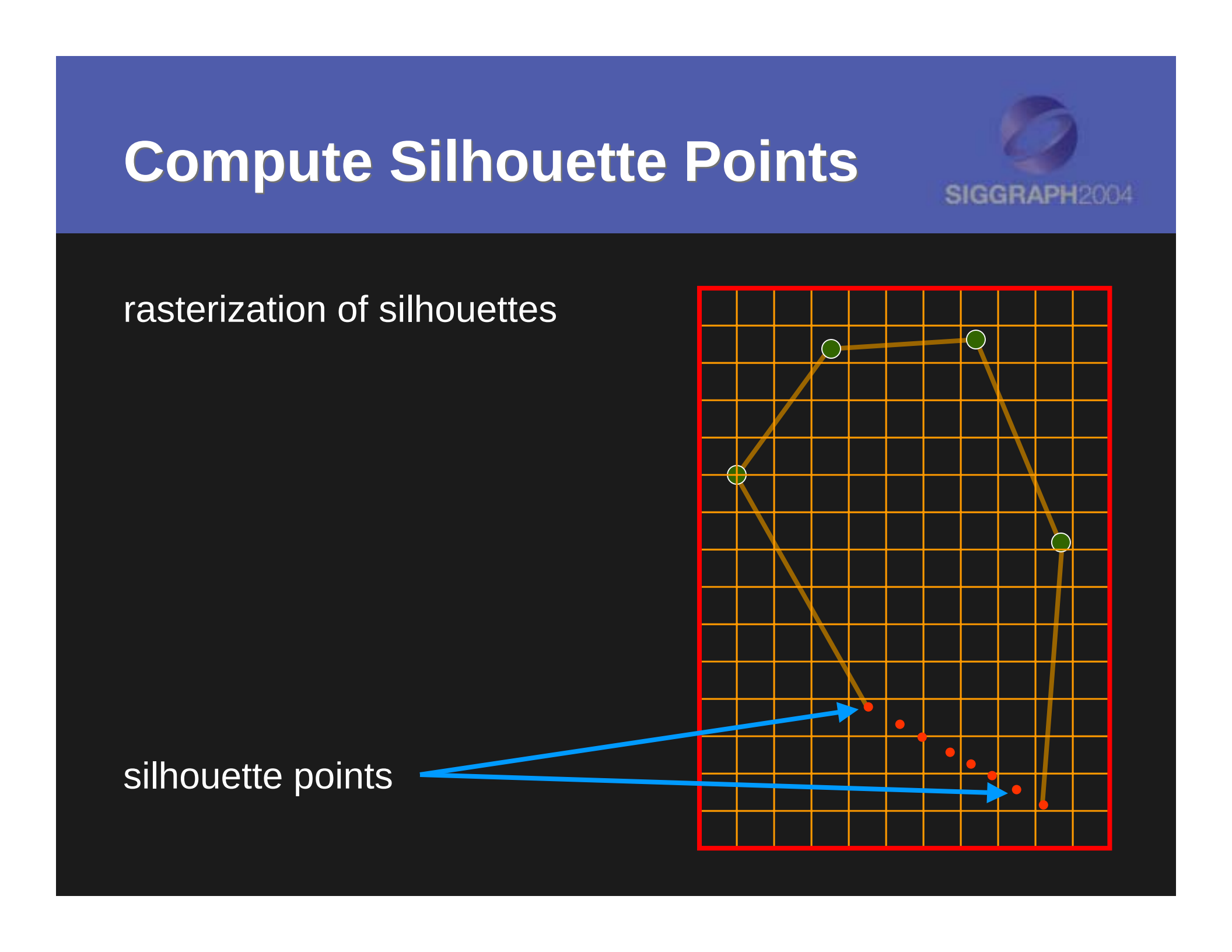
Compute Silhouette Points

rasterization of silhouettes



for each fragment:
pick a point on the edge

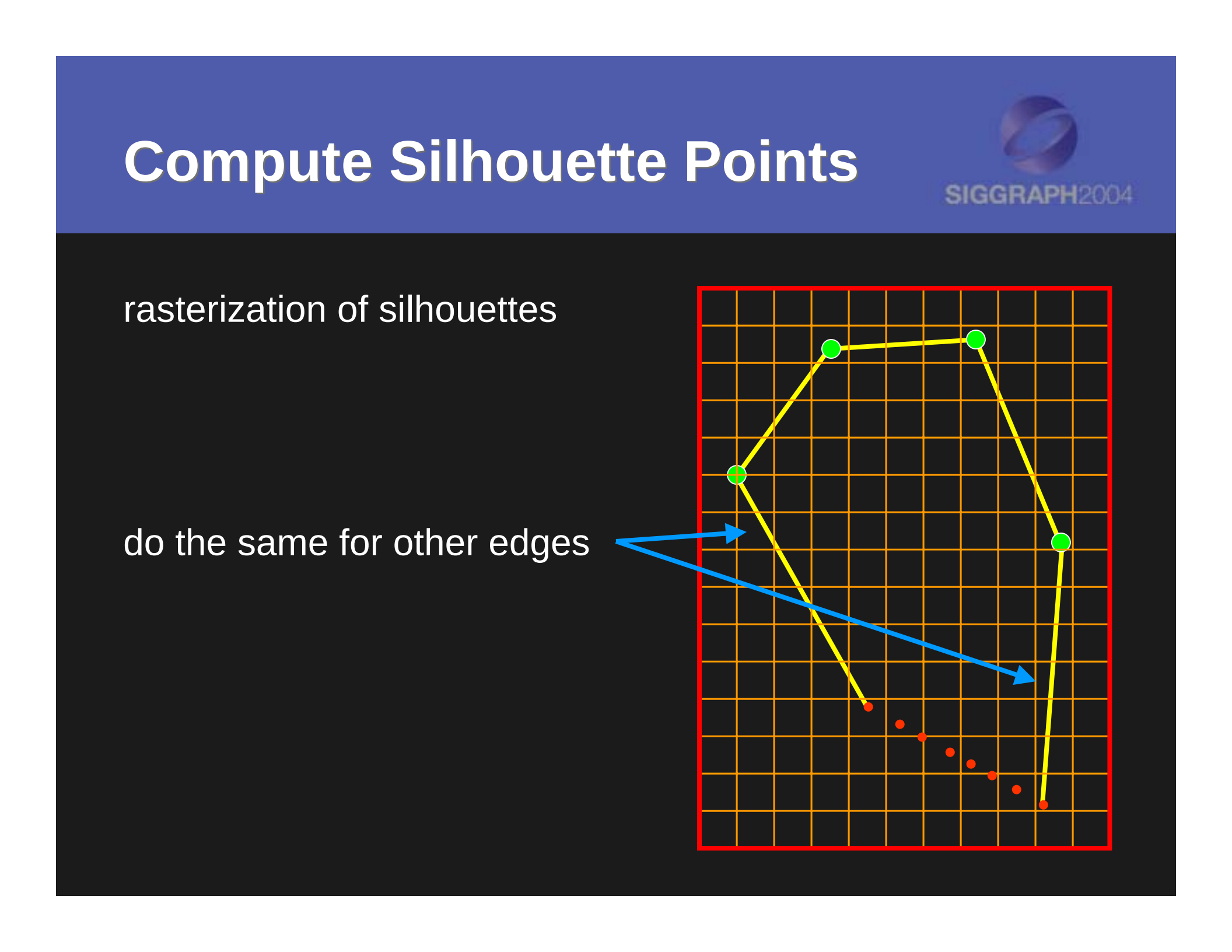
Compute Silhouette Points



rasterization of silhouettes

silhouette points

Compute Silhouette Points



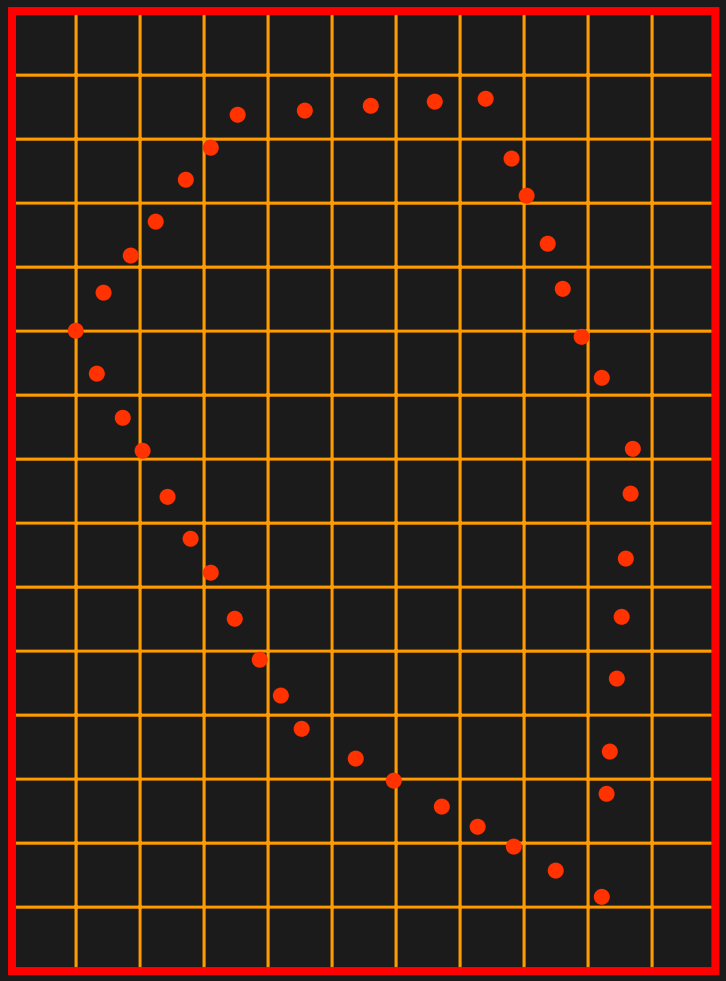
rasterization of silhouettes

do the same for other edges

Compute Silhouette Points

rasterization of silhouettes

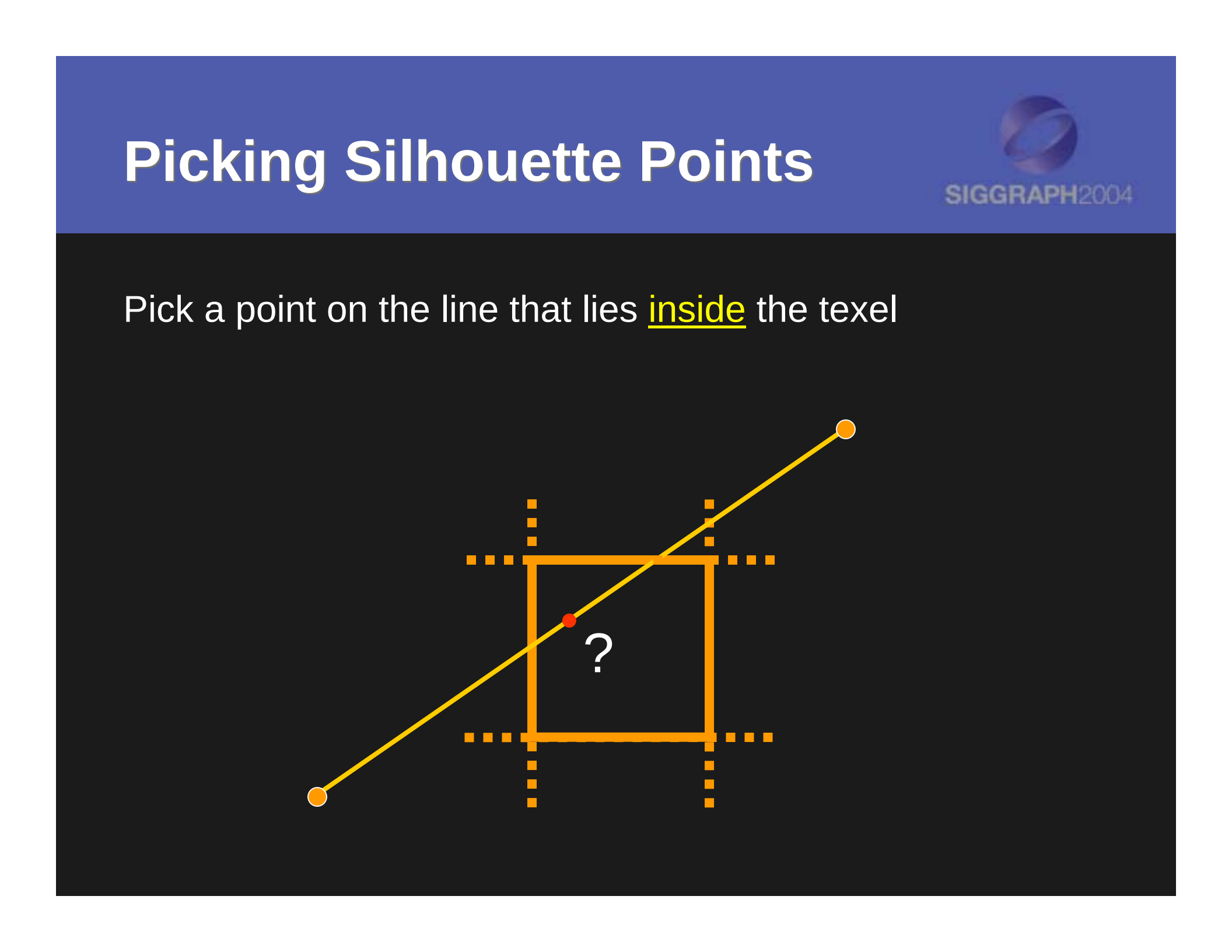
completed silhouette map



subtle issues:
- only one point per texel
- new values overwrite old ones

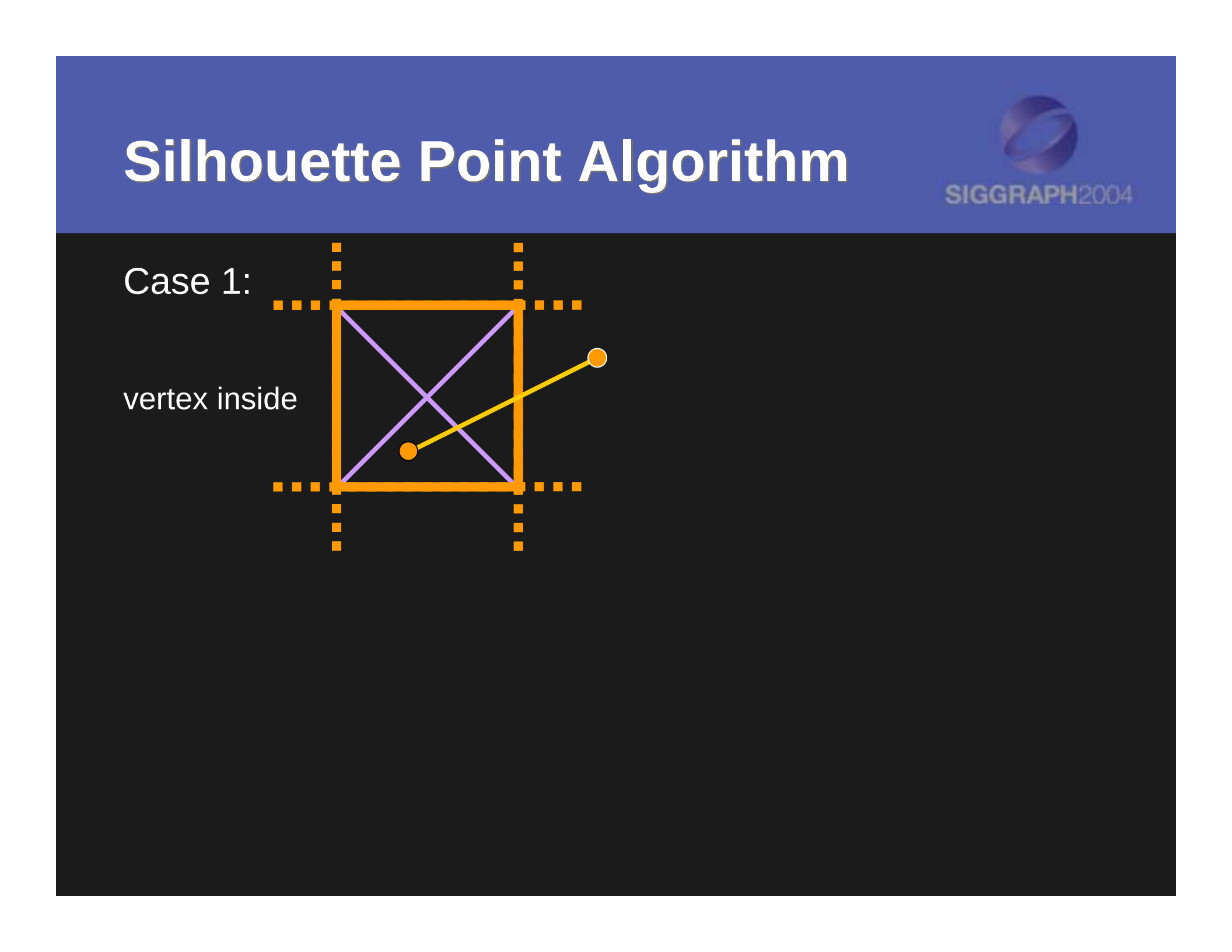
how to pick silhouette points?

Picking Silhouette Points



Pick a point on the line that lies inside the texel

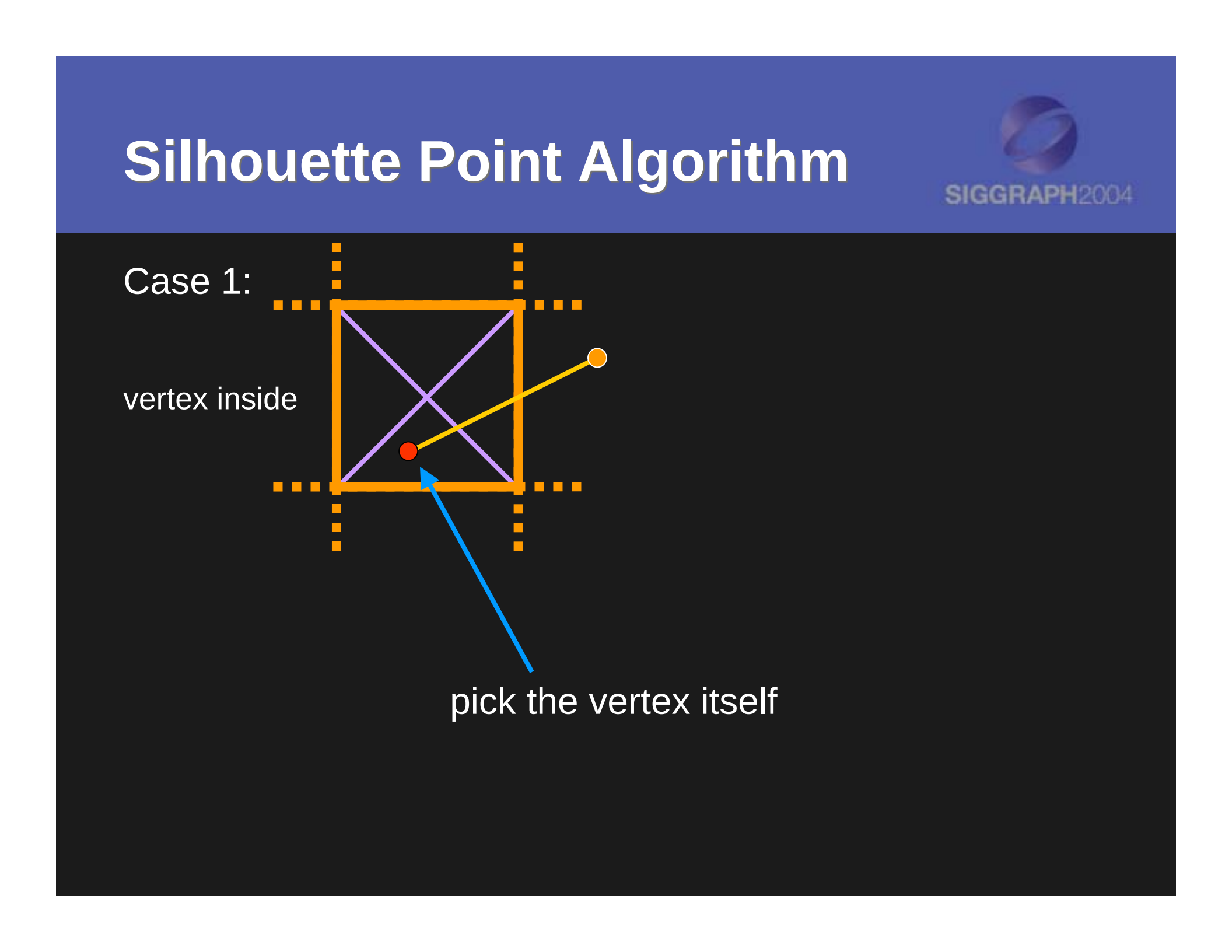
Silhouette Point Algorithm



Case 1:

vertex inside

Silhouette Point Algorithm

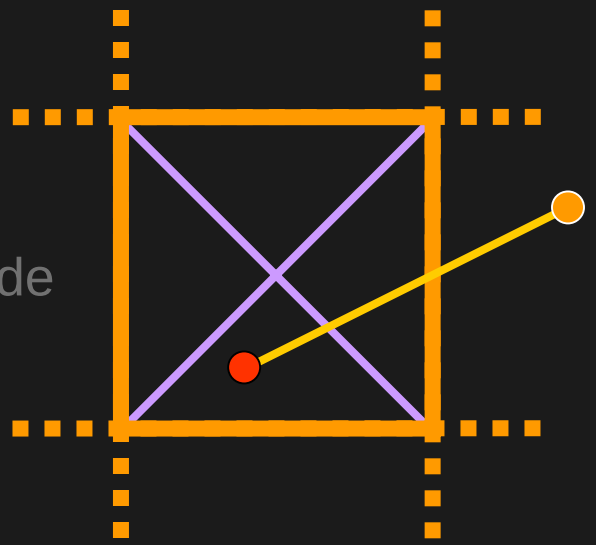


Case 1:

vertex inside

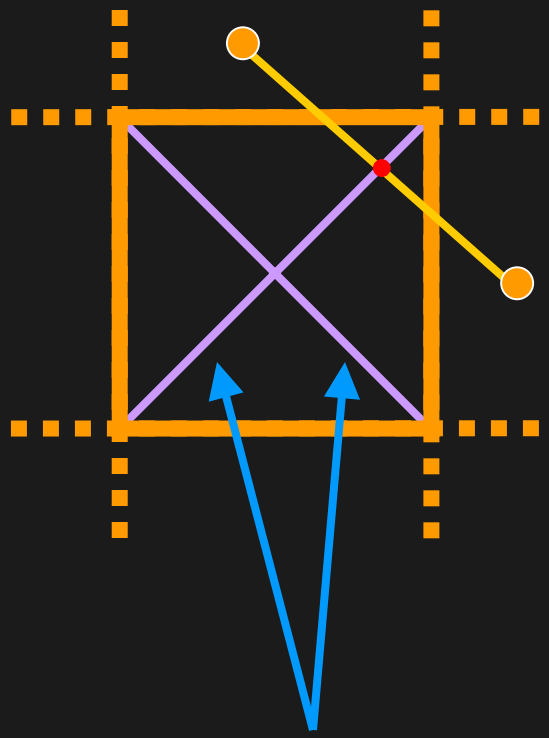
pick the vertex itself

Silhouette Point Algorithm



Case 1:

vertex inside

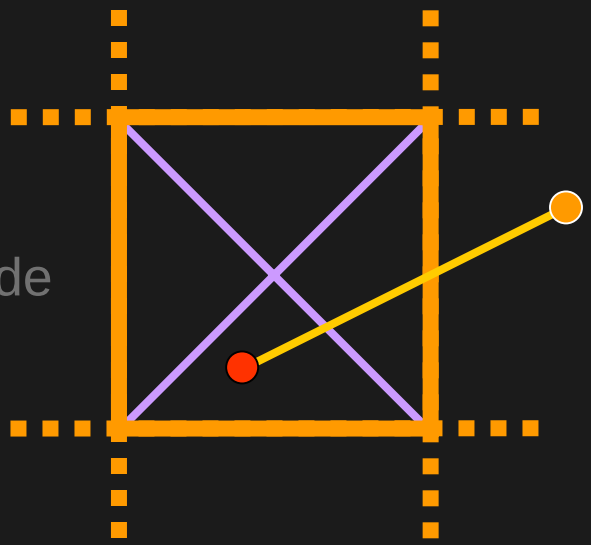


Case 2:

one intersection

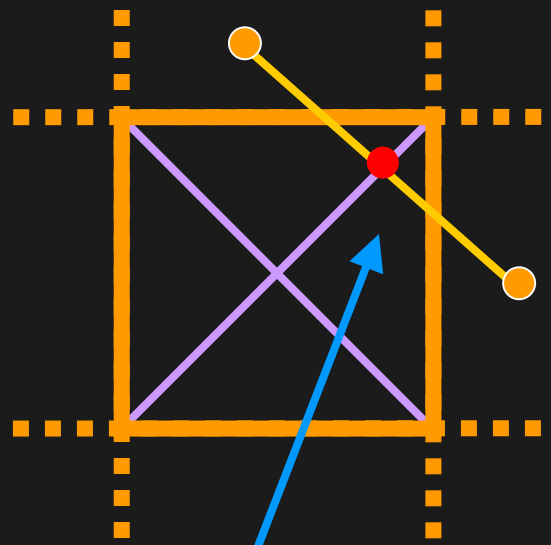
test for intersection against two diagonals

Silhouette Point Algorithm



Case 1:

vertex inside

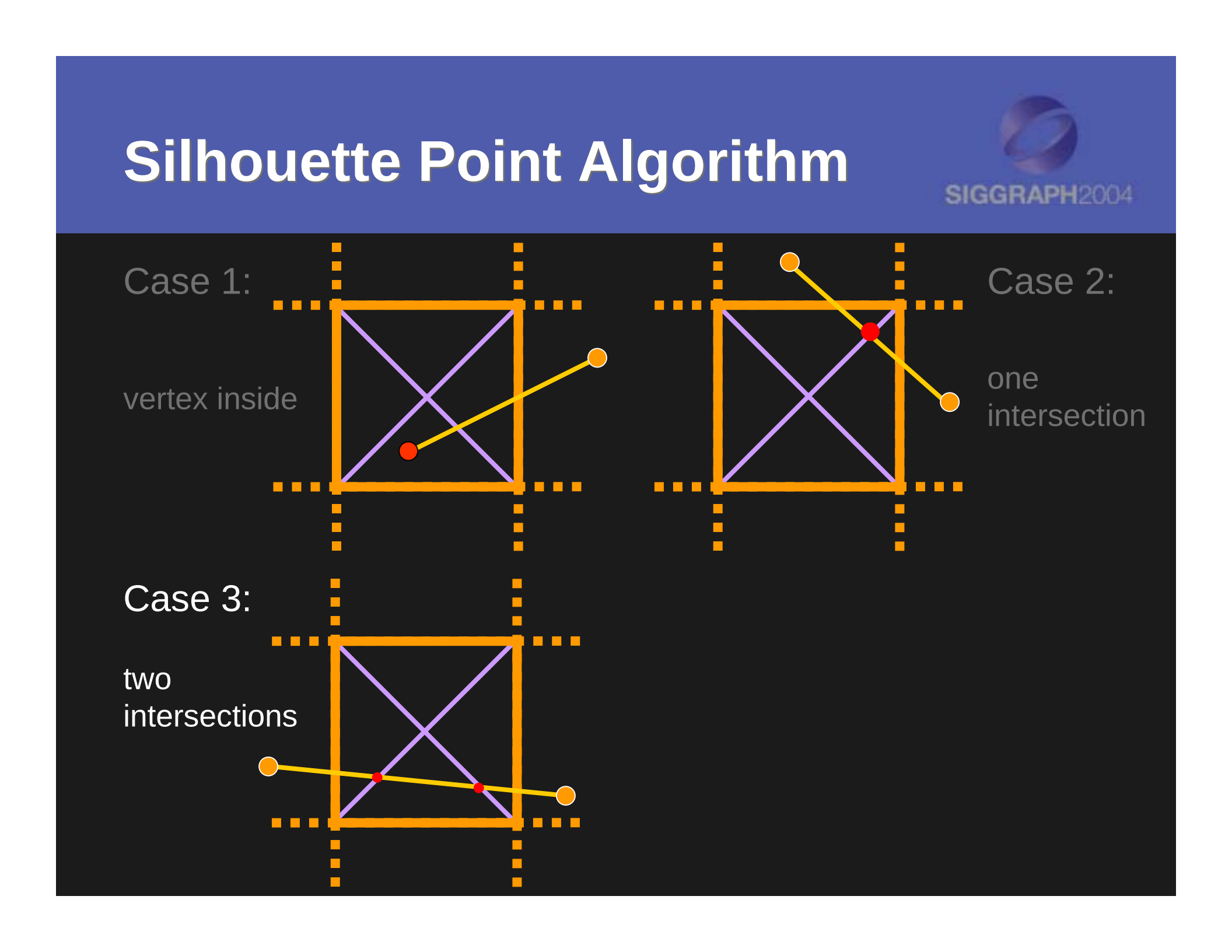


Case 2:

one intersection

pick the intersection point itself

Silhouette Point Algorithm



Case 1:

vertex inside

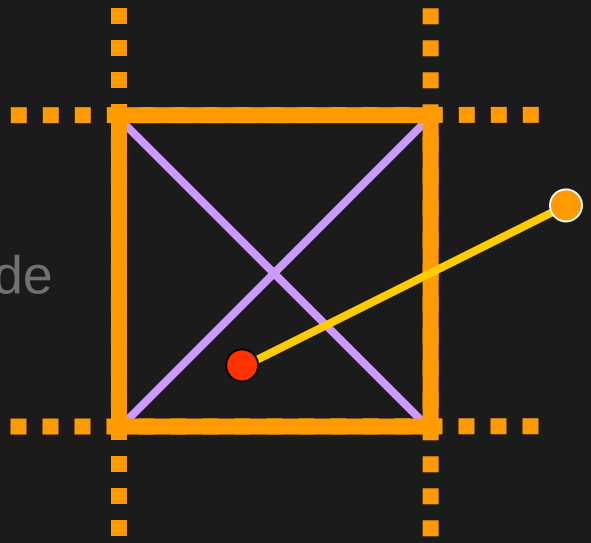
Case 2:

one intersection

Case 3:

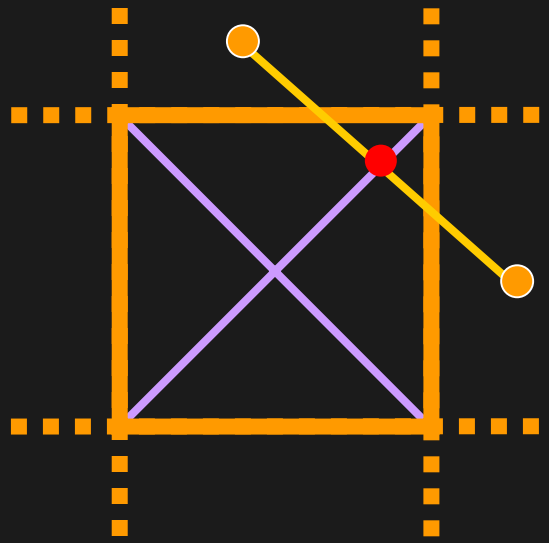
two intersections

Silhouette Point Algorithm



Case 1:

vertex inside

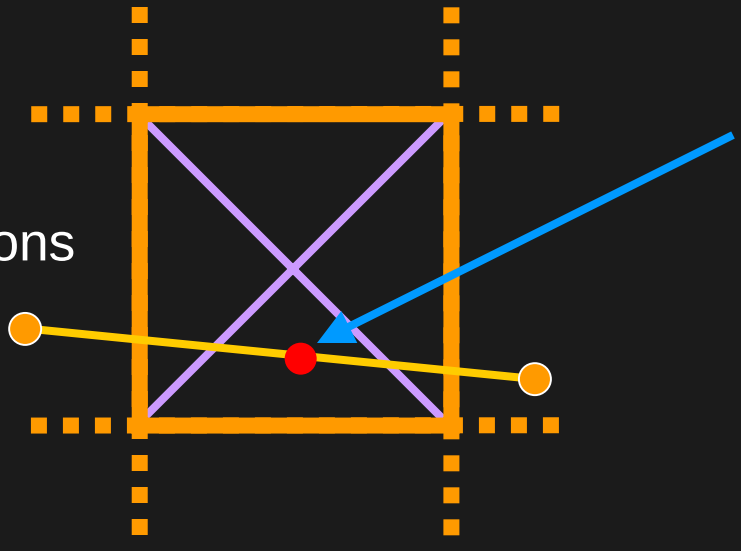


Case 2:

one intersection

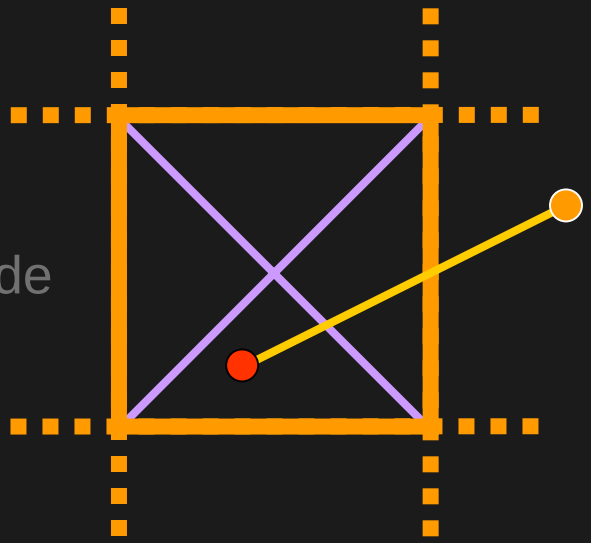
Case 3:

two intersections



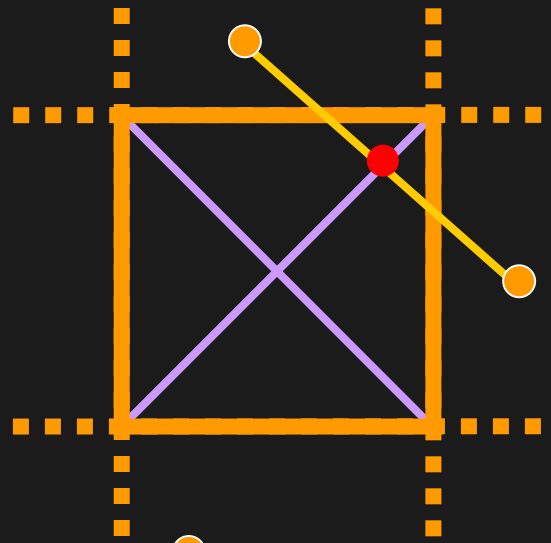
use midpoint

Silhouette Point Algorithm



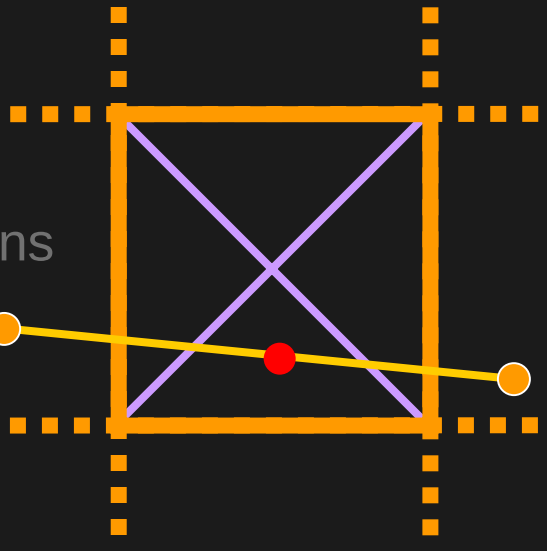
Case 1:

vertex inside



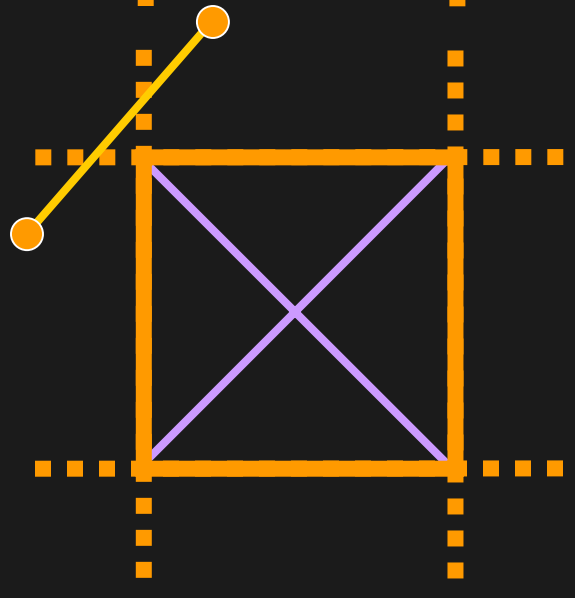
Case 2:

one intersection



Case 3:

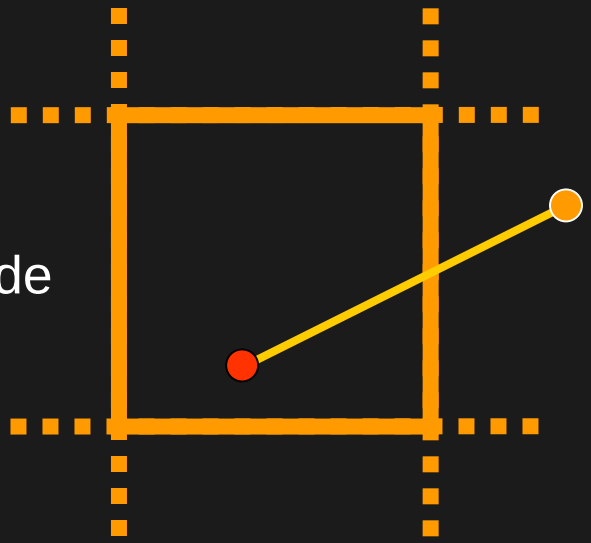
two intersections



Case 4:

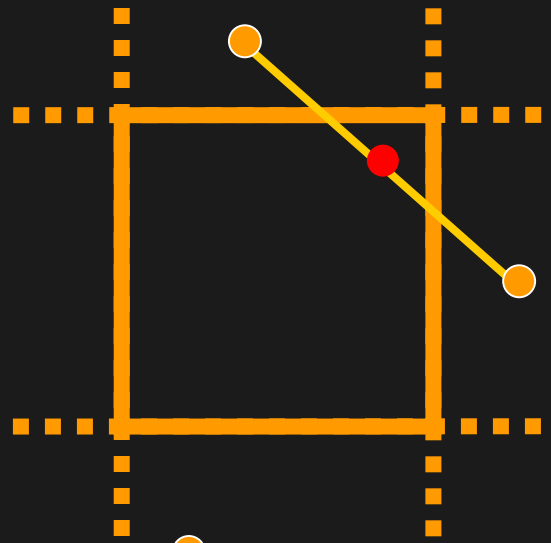
no intersections

Silhouette Point Algorithm



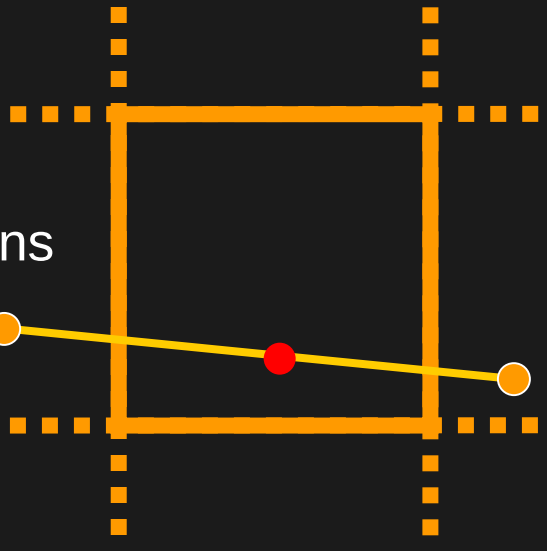
Case 1:

vertex inside



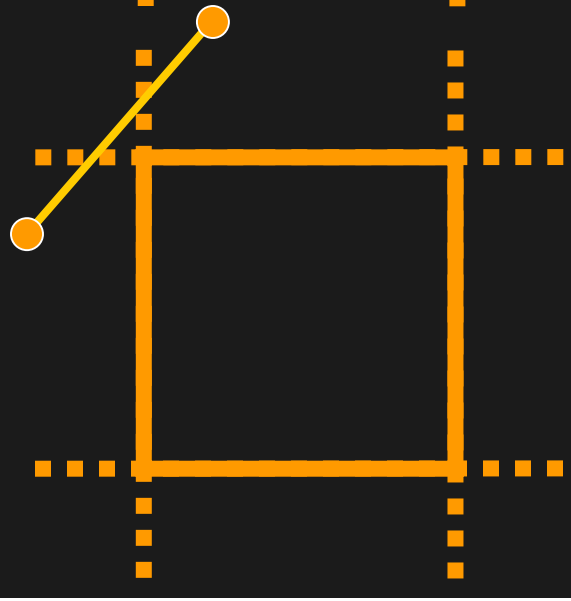
Case 2:

one intersection



Case 3:

two intersections



Case 4:

no intersections

Render scene

How to compute shadows?

Split problem into two parts:

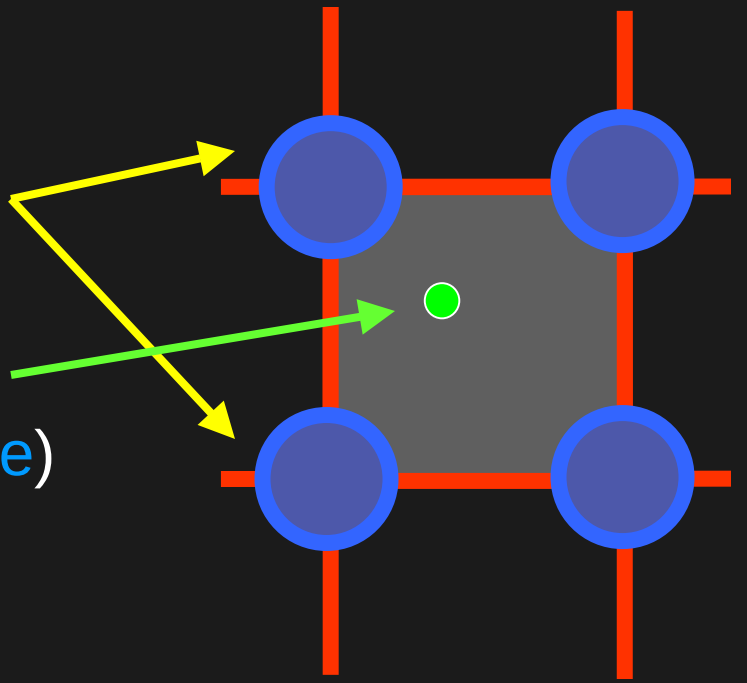
- non-silhouette pixels: use shadow map
- silhouette pixels: use silhouette map

Find Silhouette Pixels

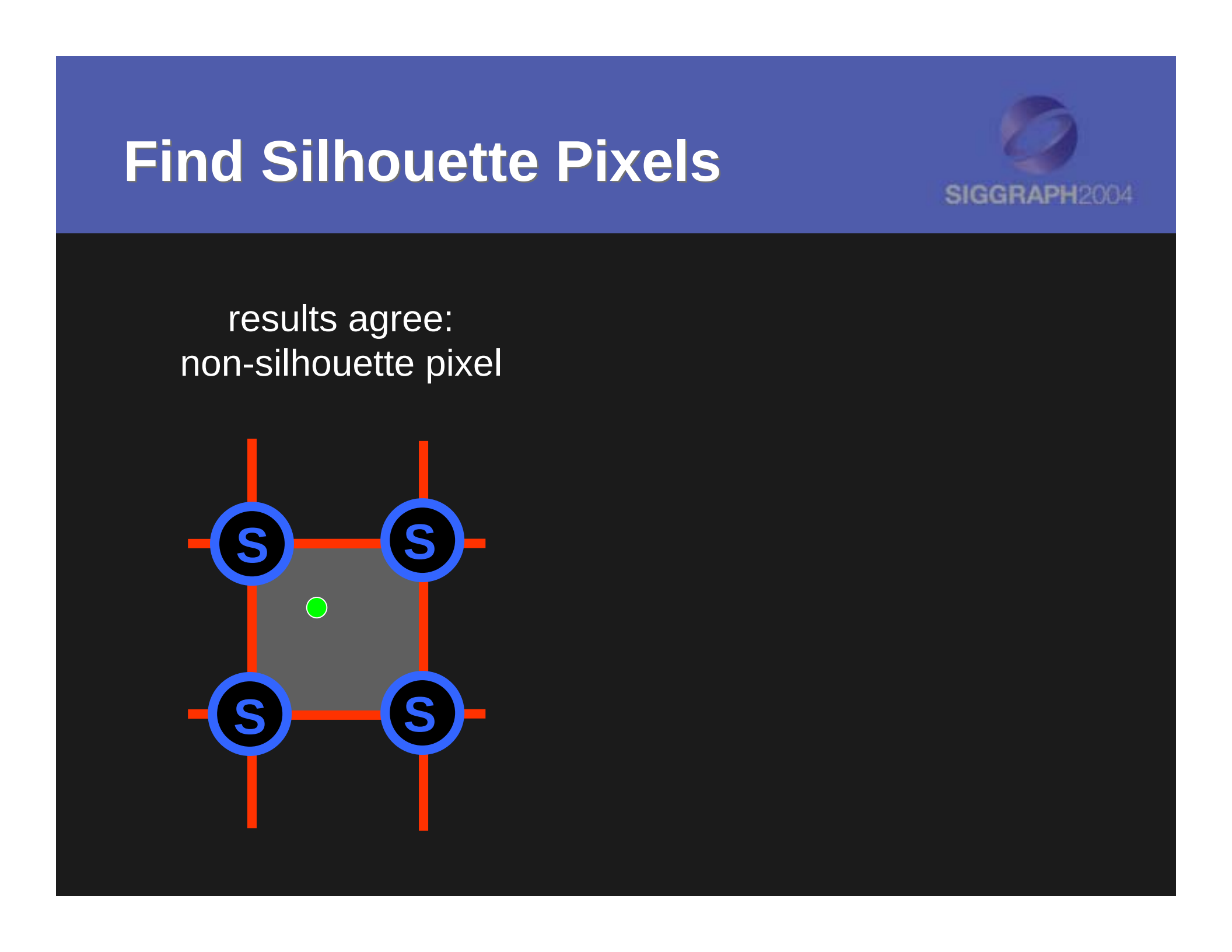
- Project sample into light space
- Compare depth against 4 nearest samples in shadow map

shadow map samples

sample to be shaded
(projected to light space)



Find Silhouette Pixels

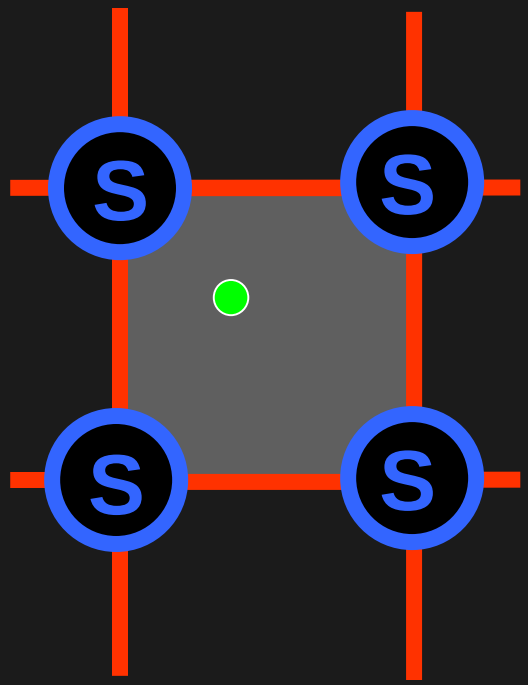


results agree:
non-silhouette pixel

Find Silhouette Pixels

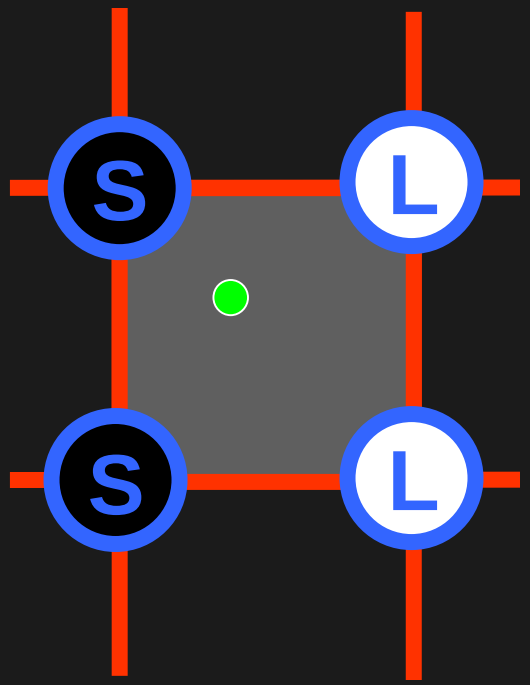
Case #1

results agree:
non-silhouette pixel



Case #2

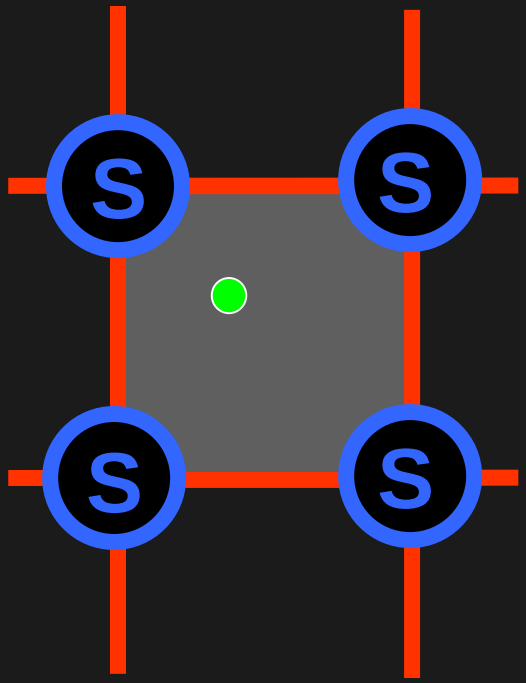
results disagree:
silhouette pixel



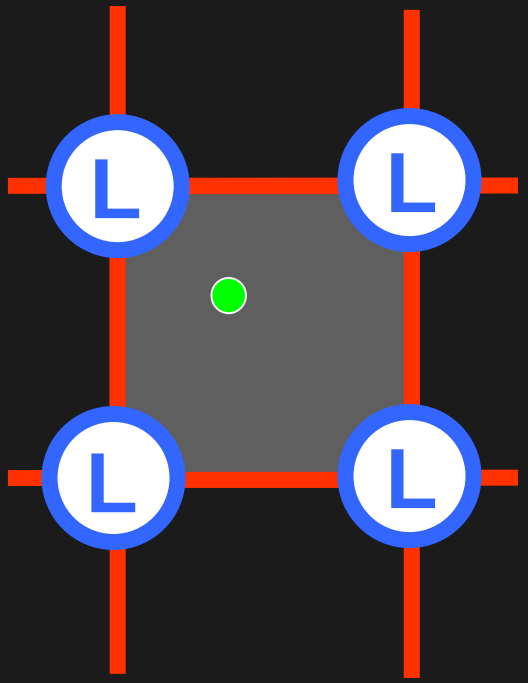
Treat Non-Silhouette Pixels

Easy: use depth comparison result

in shadow



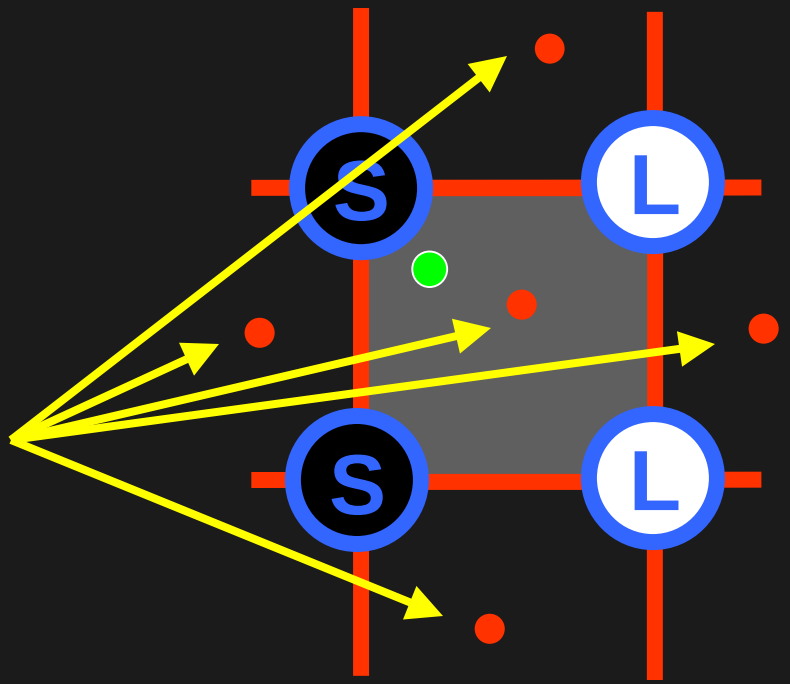
illuminated



Treat Silhouette Pixels

Reconstruct edge using silhouette map

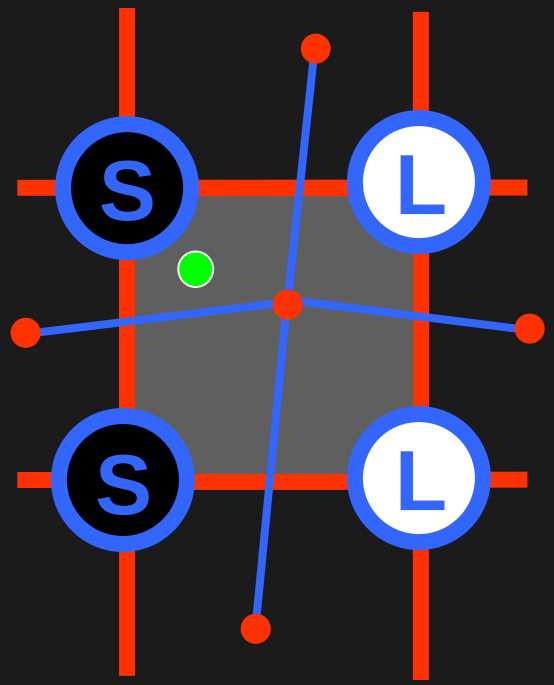
fetch five silhouette points



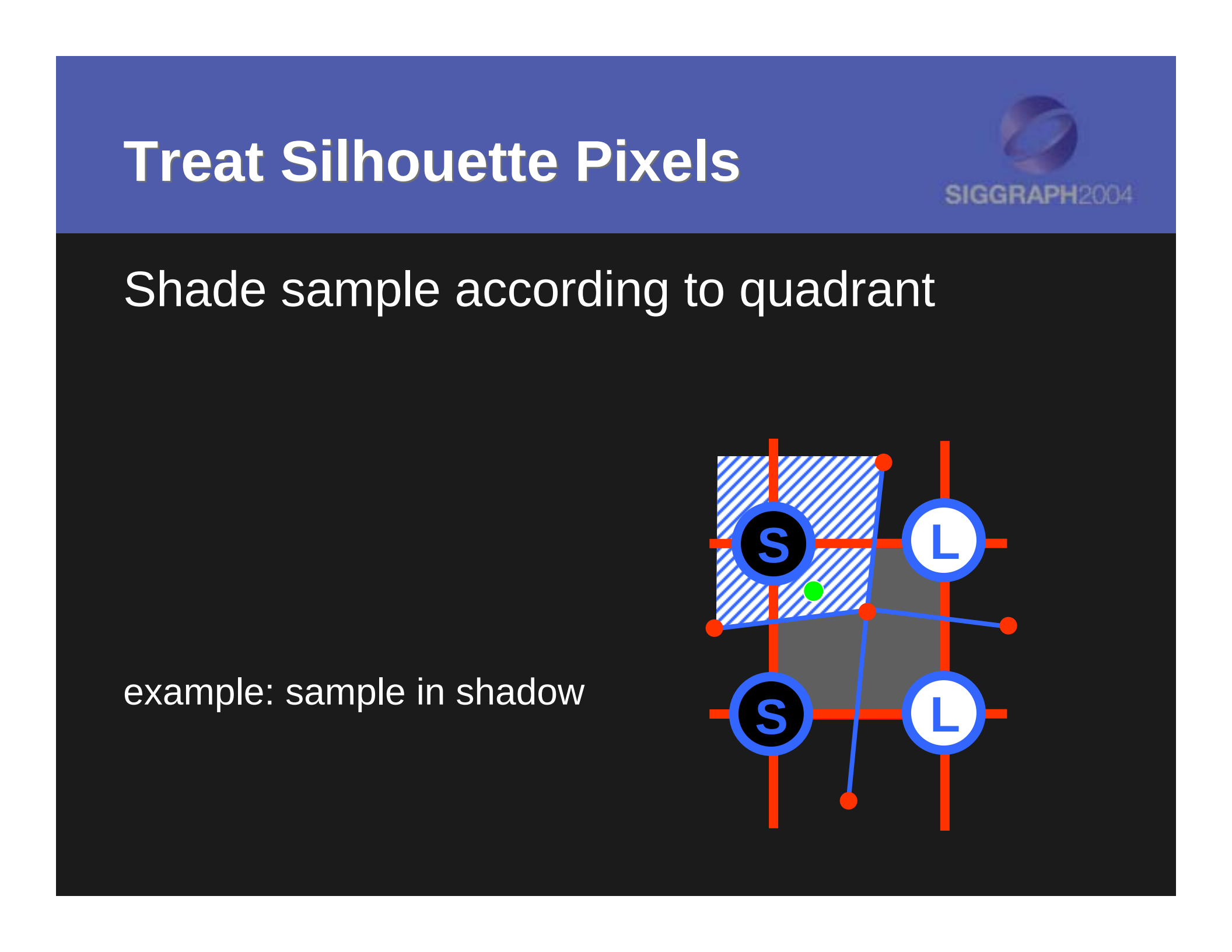
Treat Silhouette Pixels

Reconstruct edge using silhouette map

splits cell into four quadrants



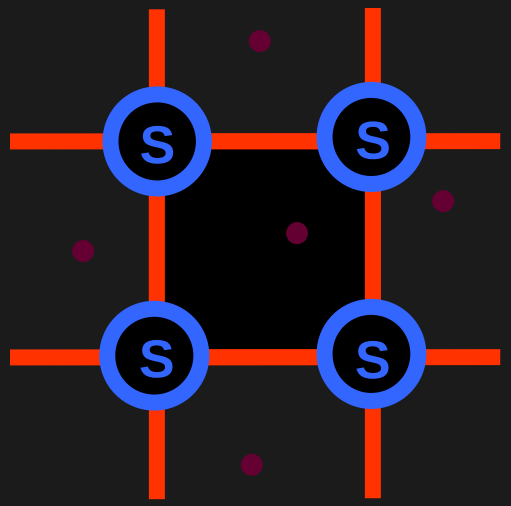
Treat Silhouette Pixels



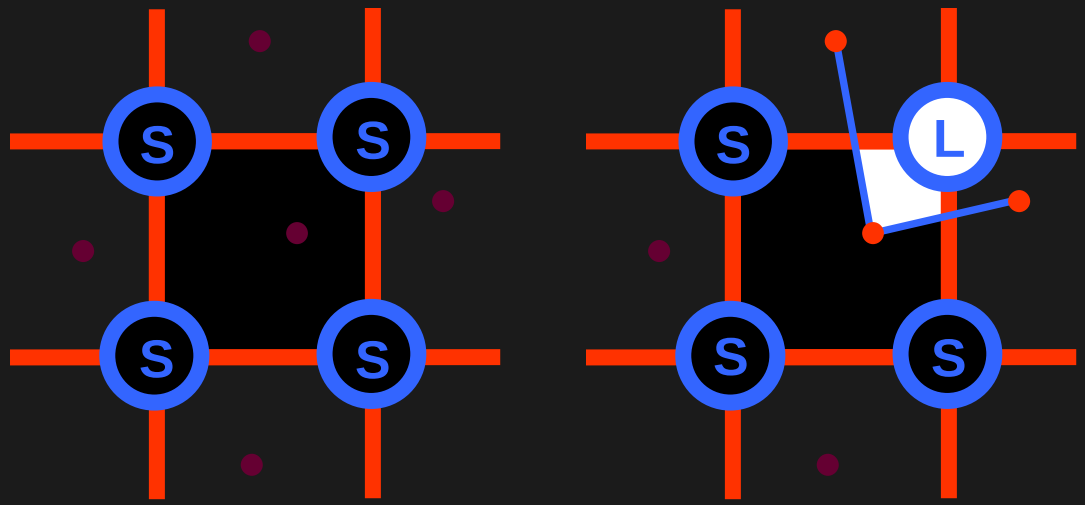
Shade sample according to quadrant

example: sample in shadow

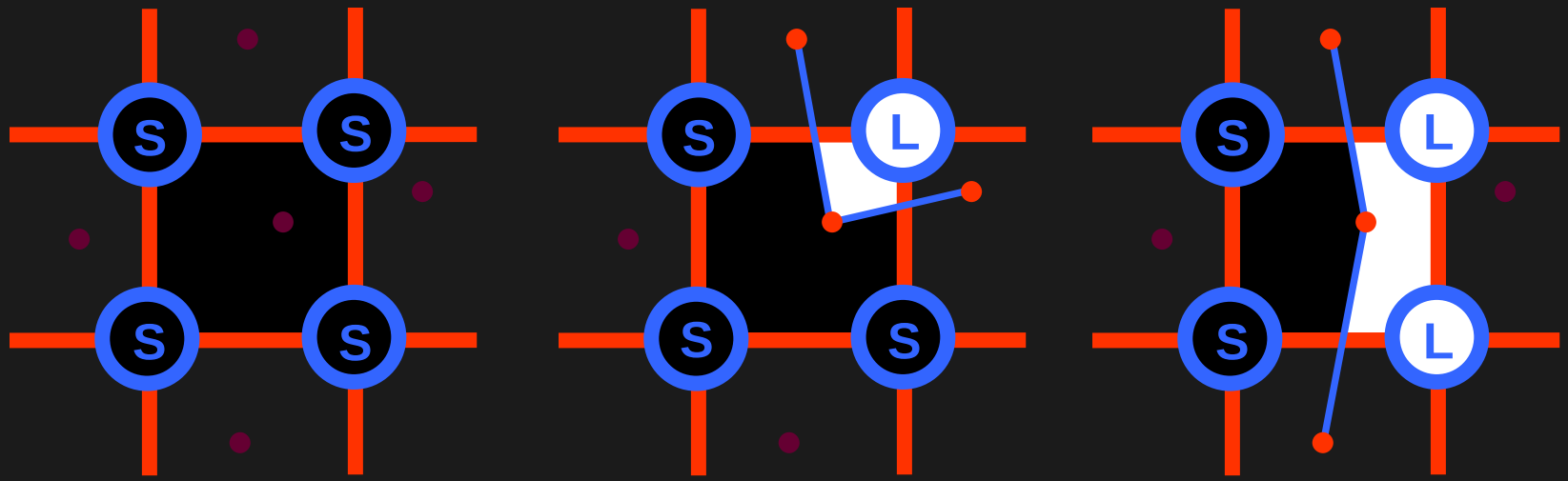
Six Combinations (1 of 6)



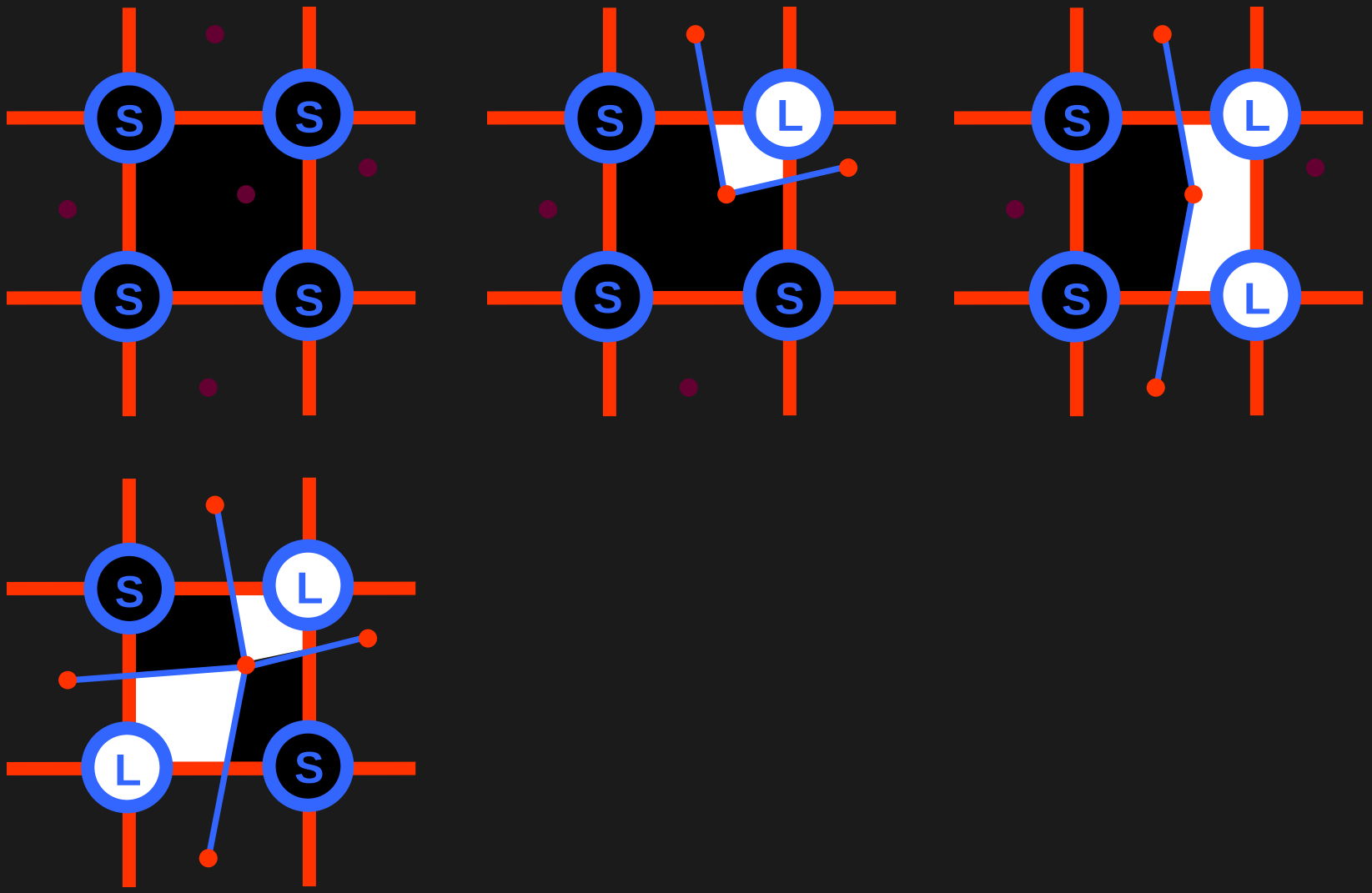
Six Combinations (2 of 6)



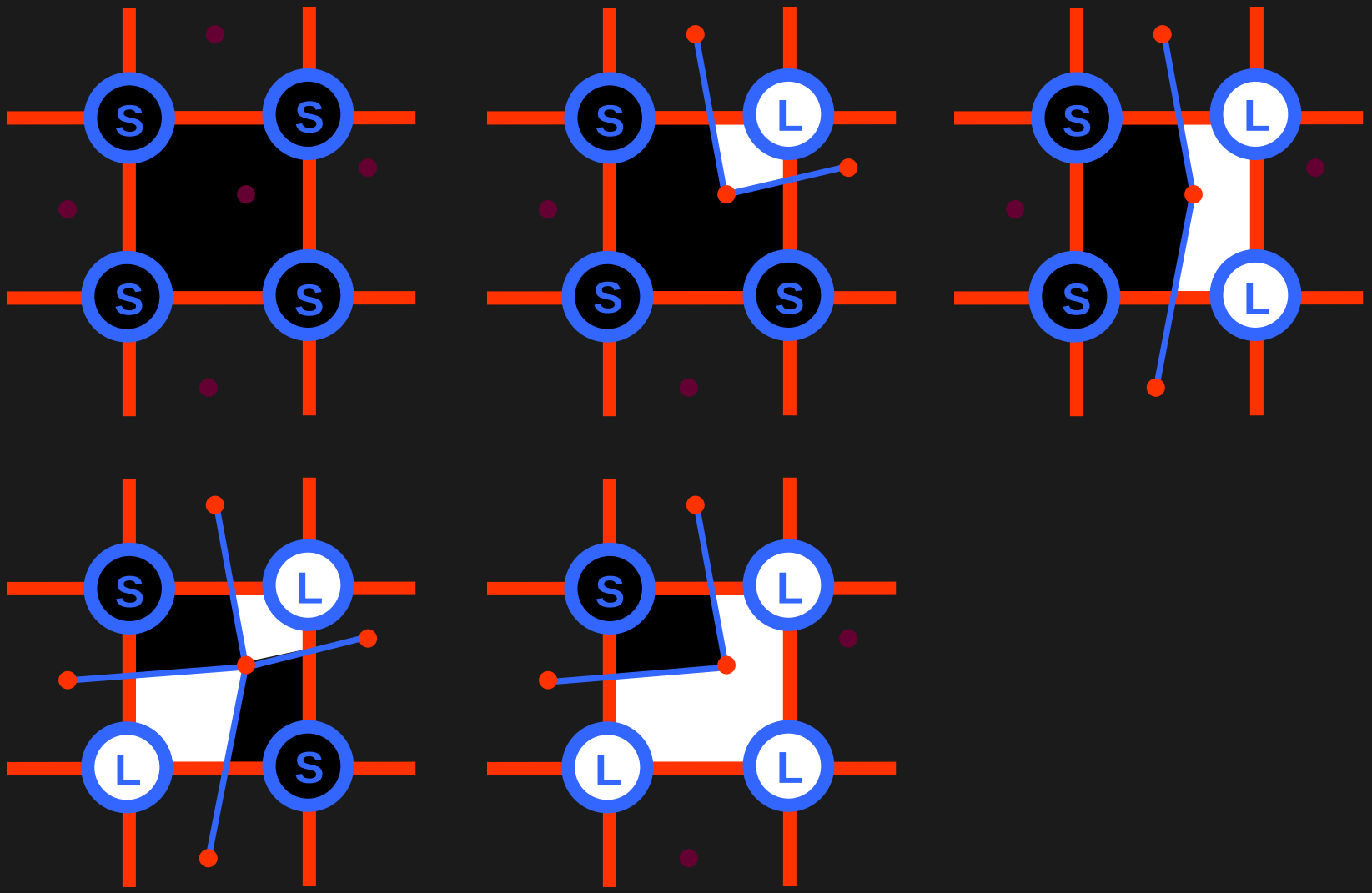
Six Combinations (3 of 6)



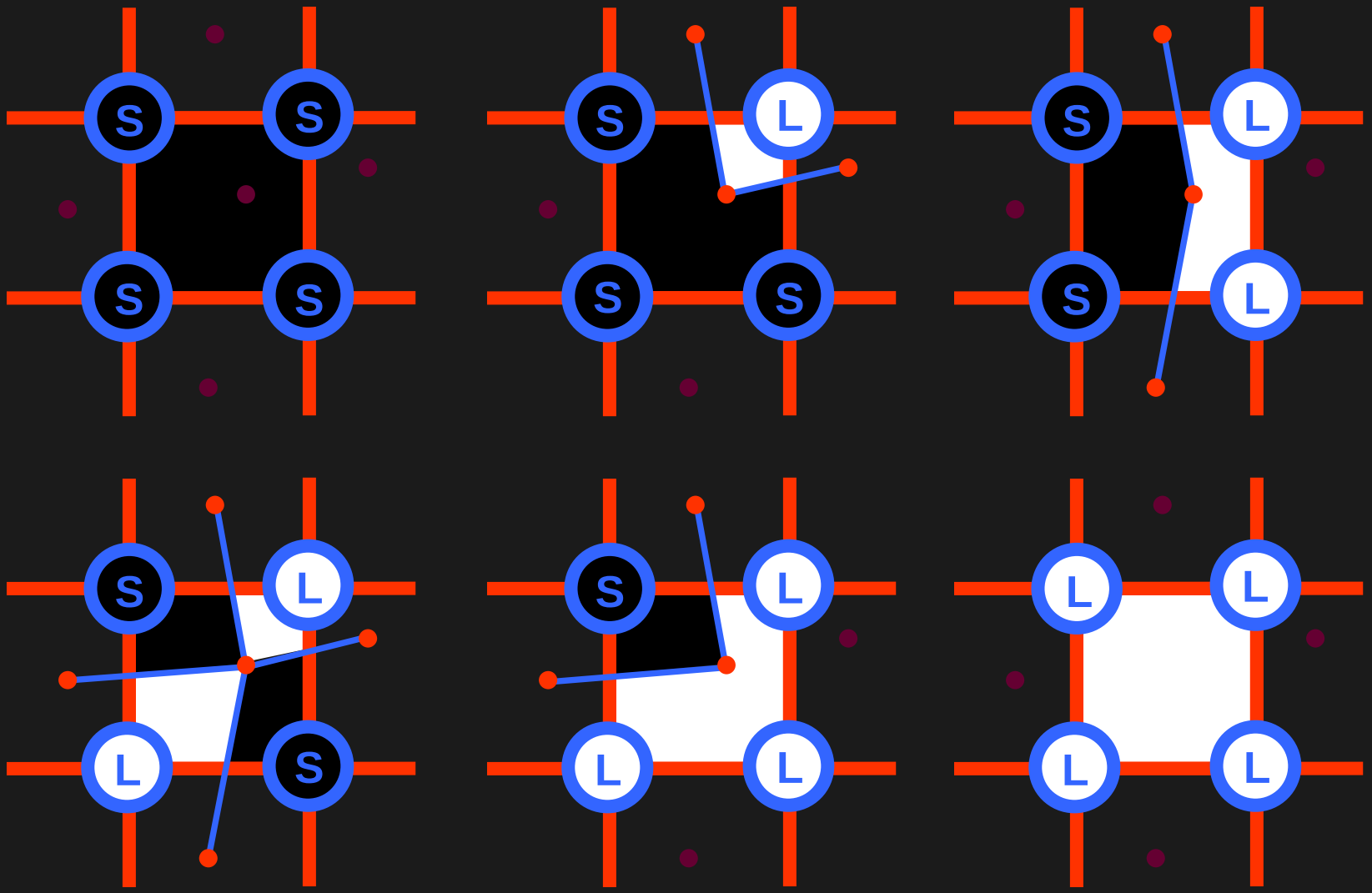
Six Combinations (4 of 6)



Six Combinations (5 of 6)

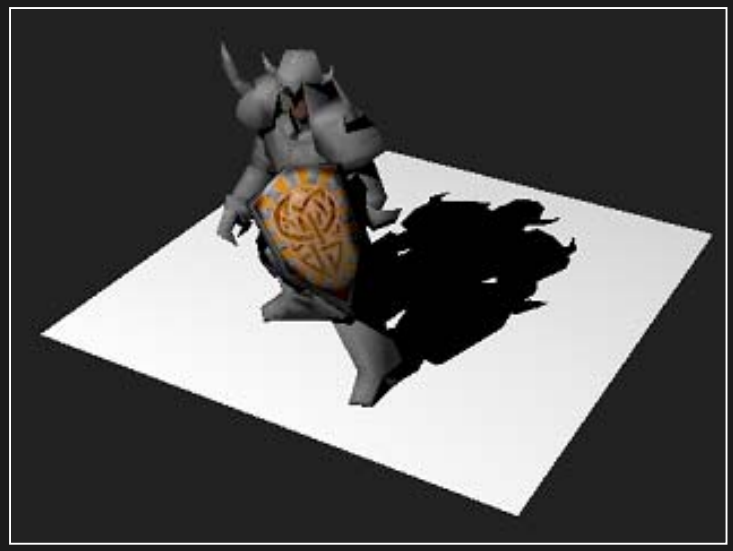


Six Combinations (6 of 6)



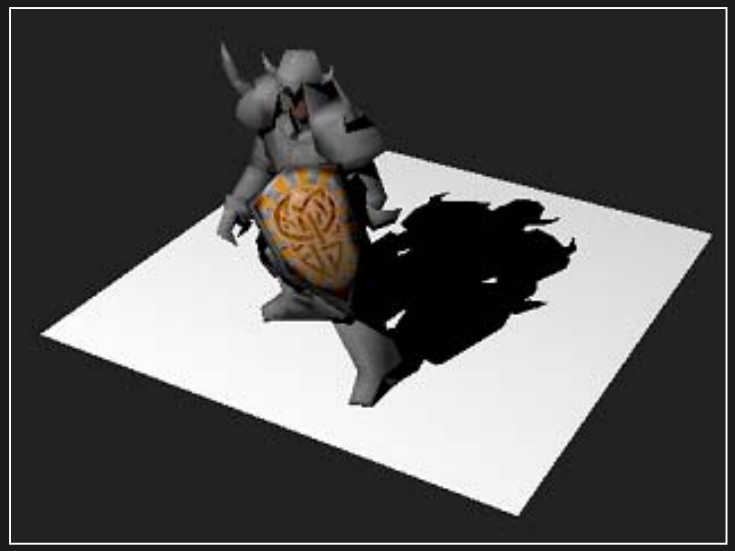
Algorithm Recap

Image-space algorithm



Algorithm Recap (1 of 3)

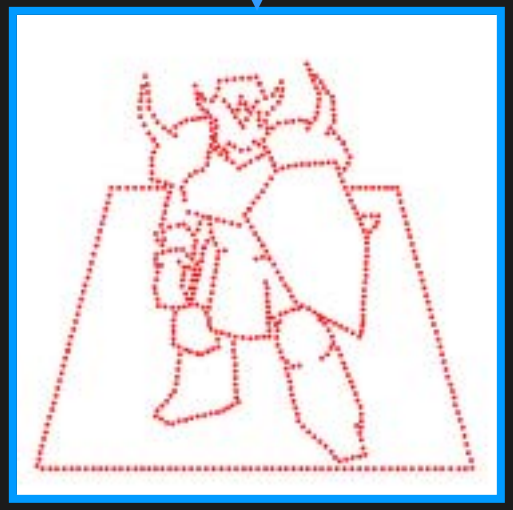
Create depth map

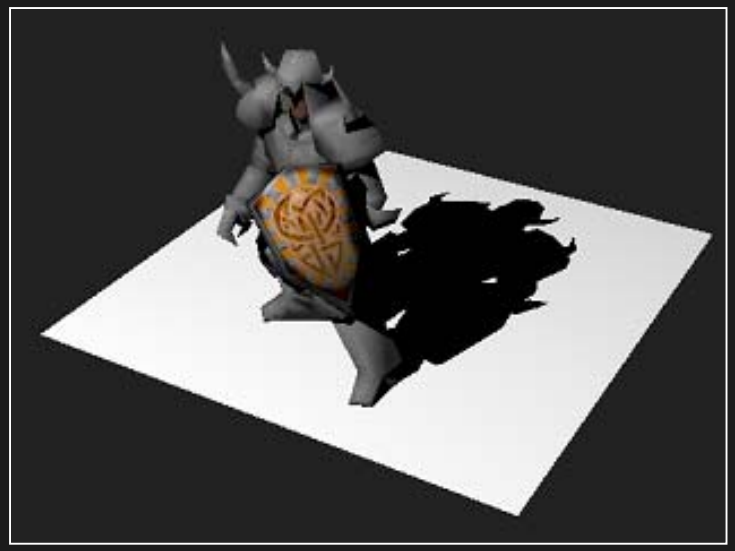


Easy: just like regular shadow map

Algorithm Recap (2 of 3)

Create silhouette map



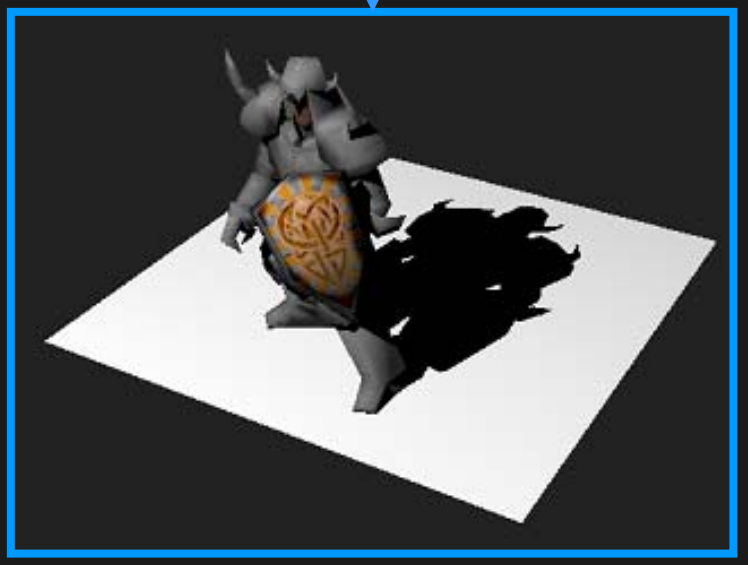


Rasterize silhouette edges

Pick silhouette points, 1 per texel

Algorithm Recap (3 of 3)

Render scene and shadows



Fetch local silhouette points

Reconstruct shadow edge

Implementation

Implementation

- Details (OpenGL)
- Hardware acceleration
- Optimizations

Create Shadow Map

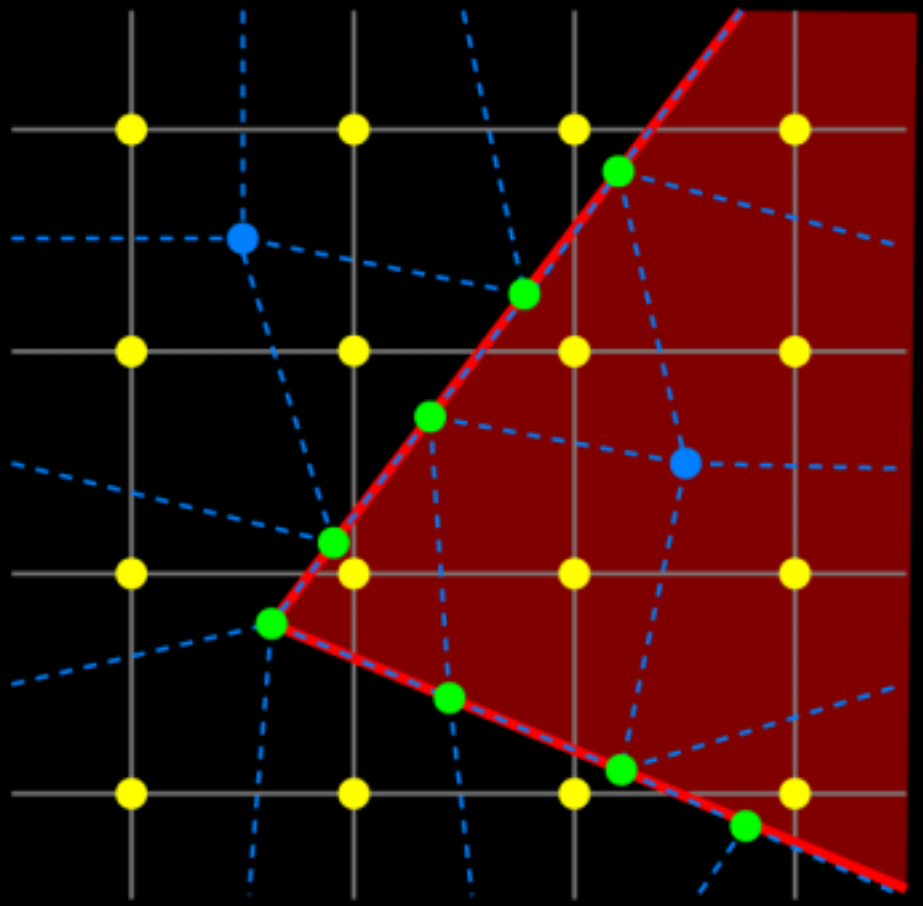
Render to standard OpenGL depth buffer

Optimizations

- for closed models, cull back faces
- turn off shading, color writes
- only send vertex positions
- draw roughly front-to-back

Create Silhouette Map

Goal: store points that lie on silhouette



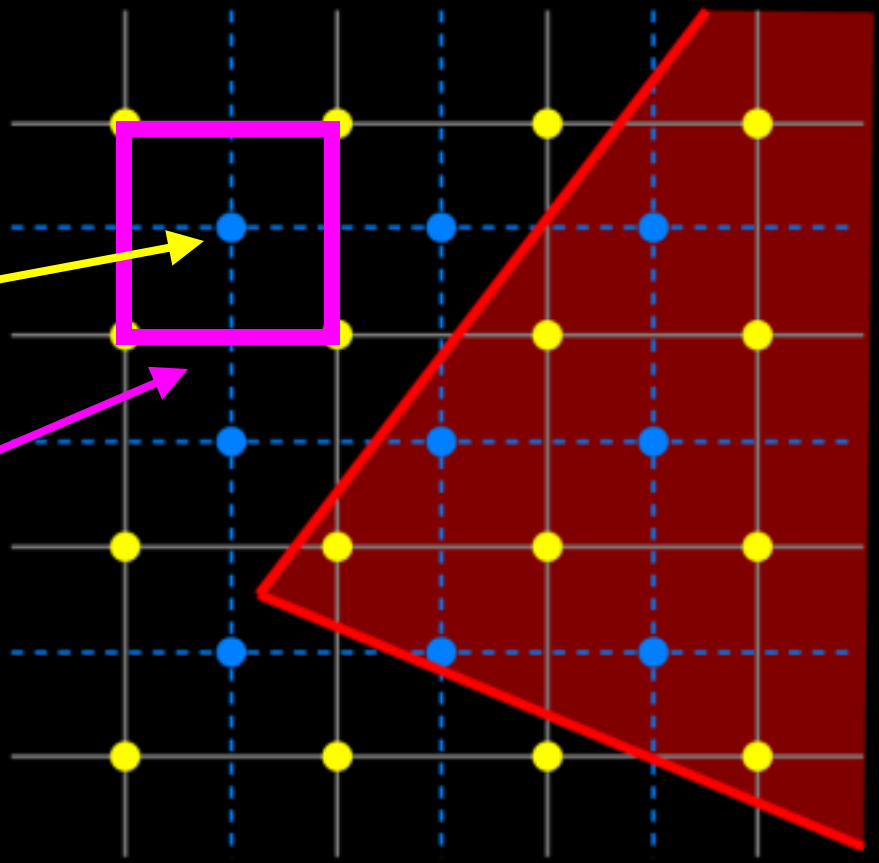
Create Silhouette Map

Place default point at texel center

default silhouette point

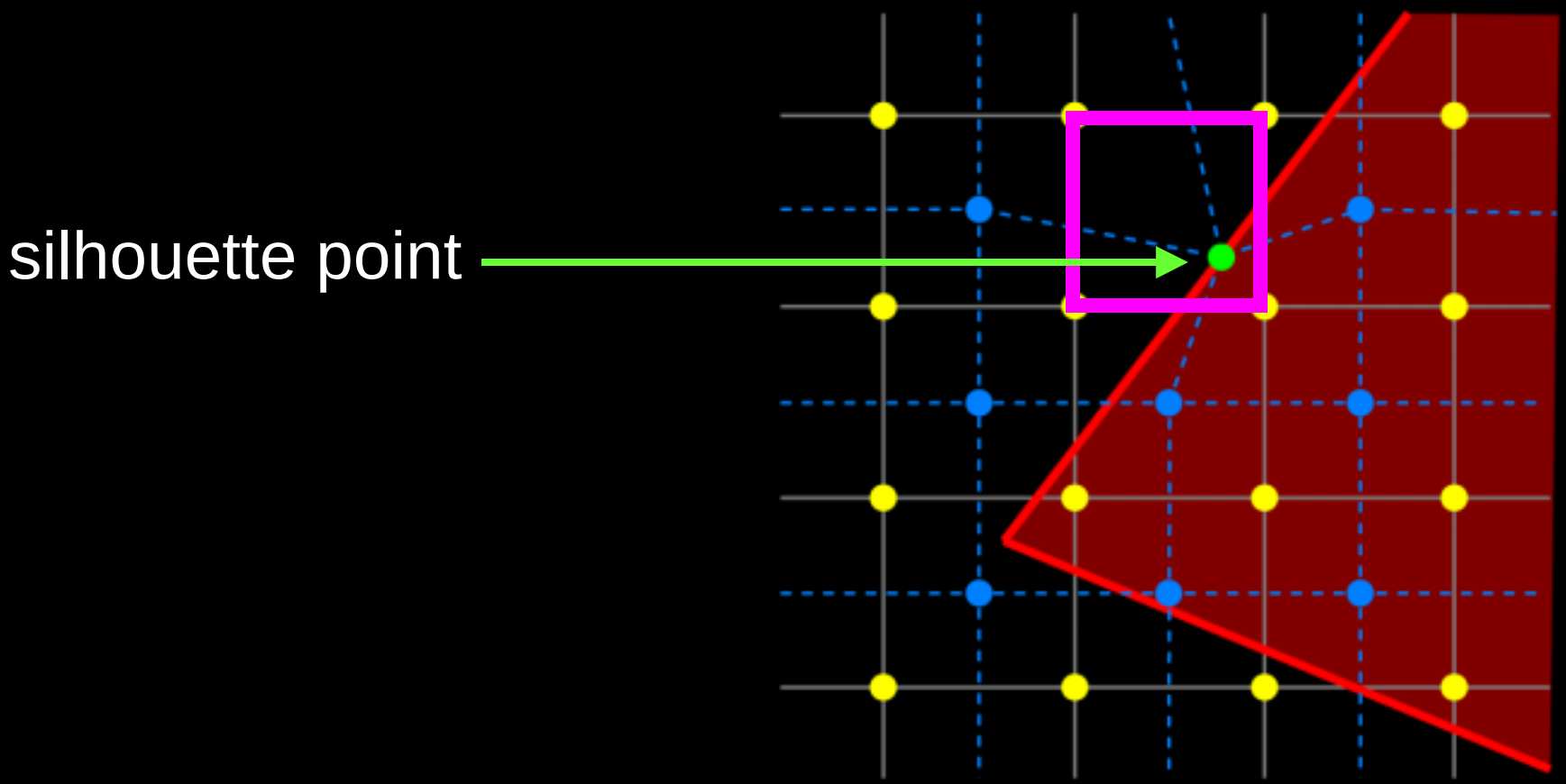
silhouette map texel

use glClear(…)



Create Silhouette Map

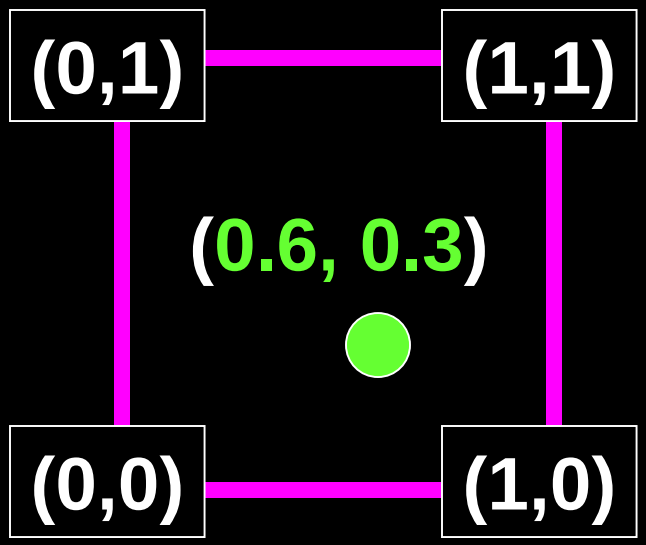
Fragment program finds silhouette points

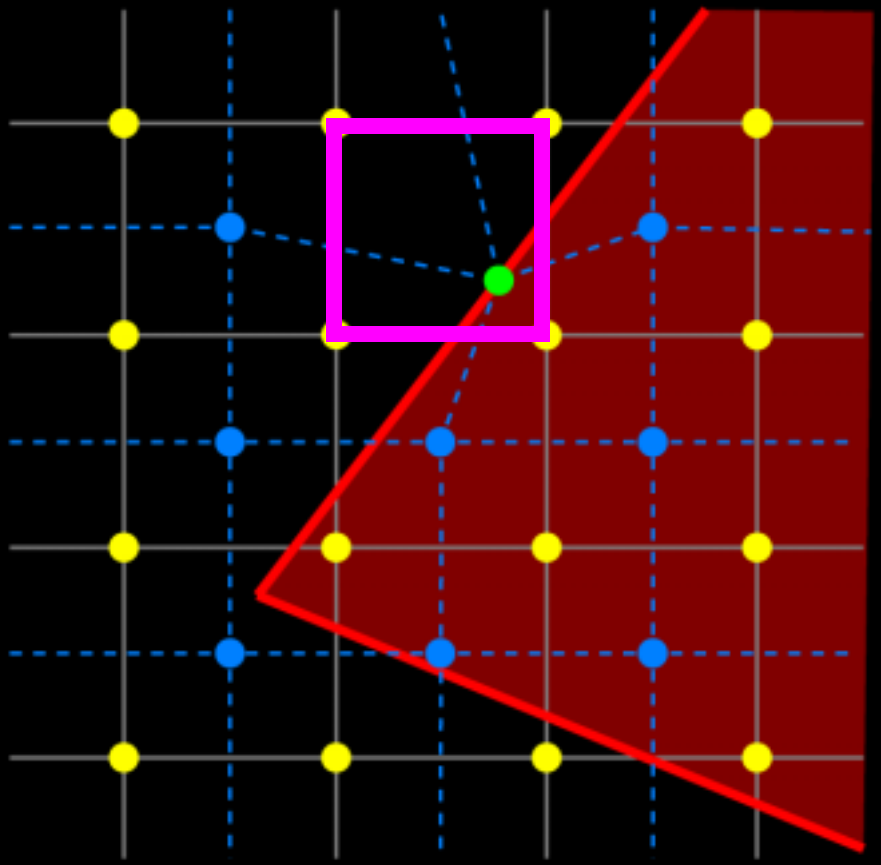


Create Silhouette Map

Fragment program finds silhouette points

- use local coordinates
- store only xy offsets

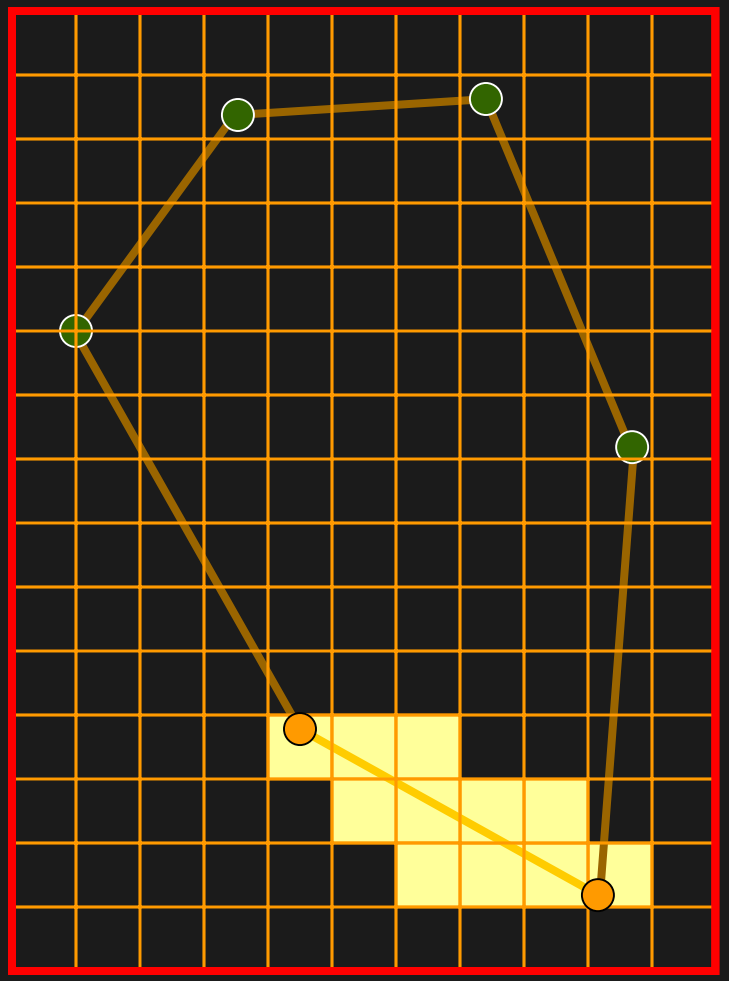




Rasterizing Silhouettes

Two issues:

- must guarantee generation of silhouette pixels
- discard occluded silhouettes



Rasterizing Silhouettes

Rasterize conservatively

- Be careful using OpenGL wide lines
- Use width of at least 3

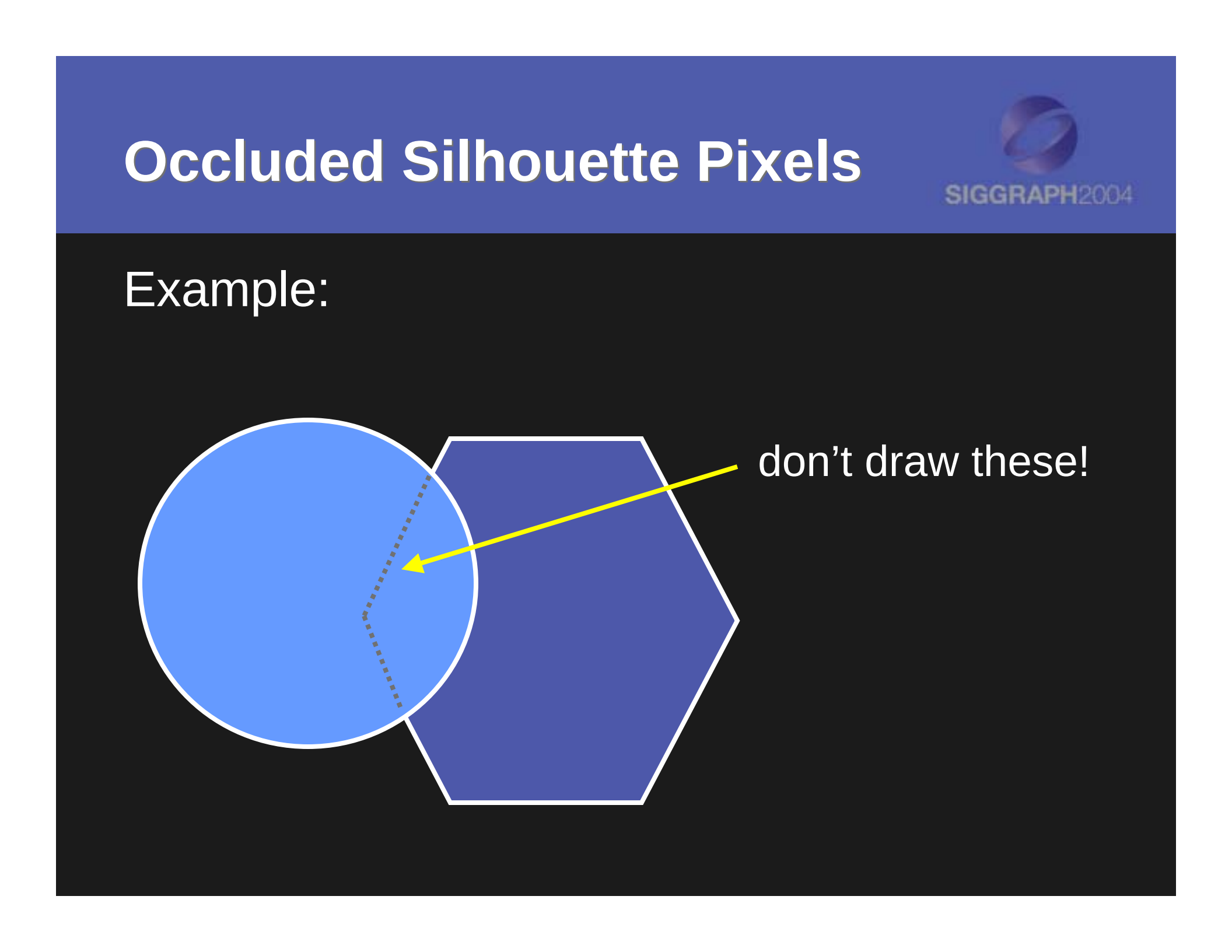
glLineWidth(3);

- Make lines slightly longer to cover endpoints

Another solution: use thin quads, not lines

- See Sen et al. [SIG2003] paper

Occluded Silhouette Pixels

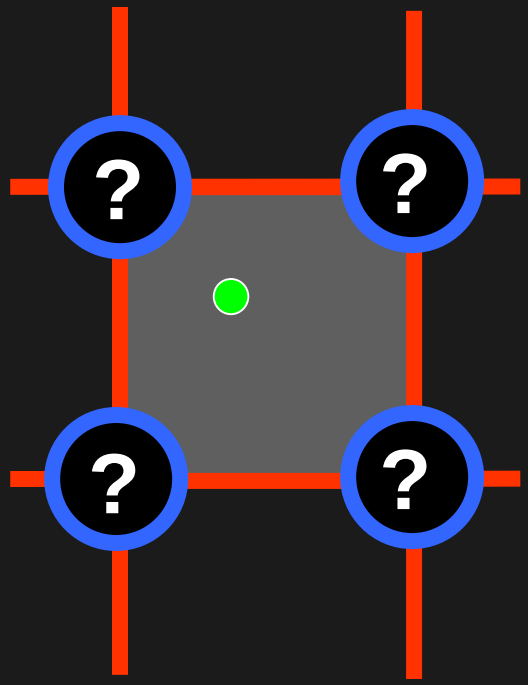


Example:

Occluded Silhouette Pixels

Implementing occlusion:

- Use depth map from first pass
- Recall silhouette map offset by ½ pixel
- Use fragment kill if depth is greater than 4 nearest samples in depth map



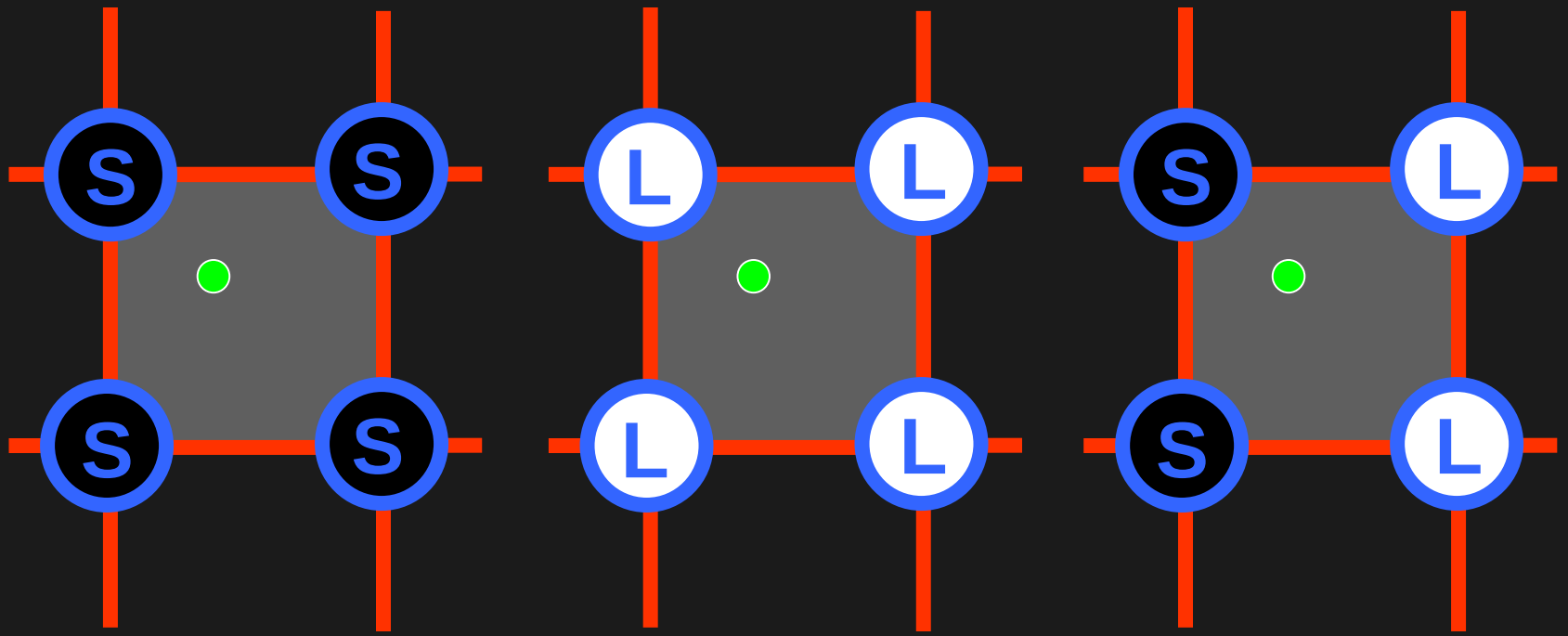
Rendering Final Image

Recall

- Draw from observer's view
- Identify silhouette vs. non-silhouette pixels
- Use shadow map for non-silhouette pixels
- Use silhouette map for silhouette pixels

Identify Silhouette Pixels

- Take advantage of hardware shadow mapping
- Use percentage closer filtering

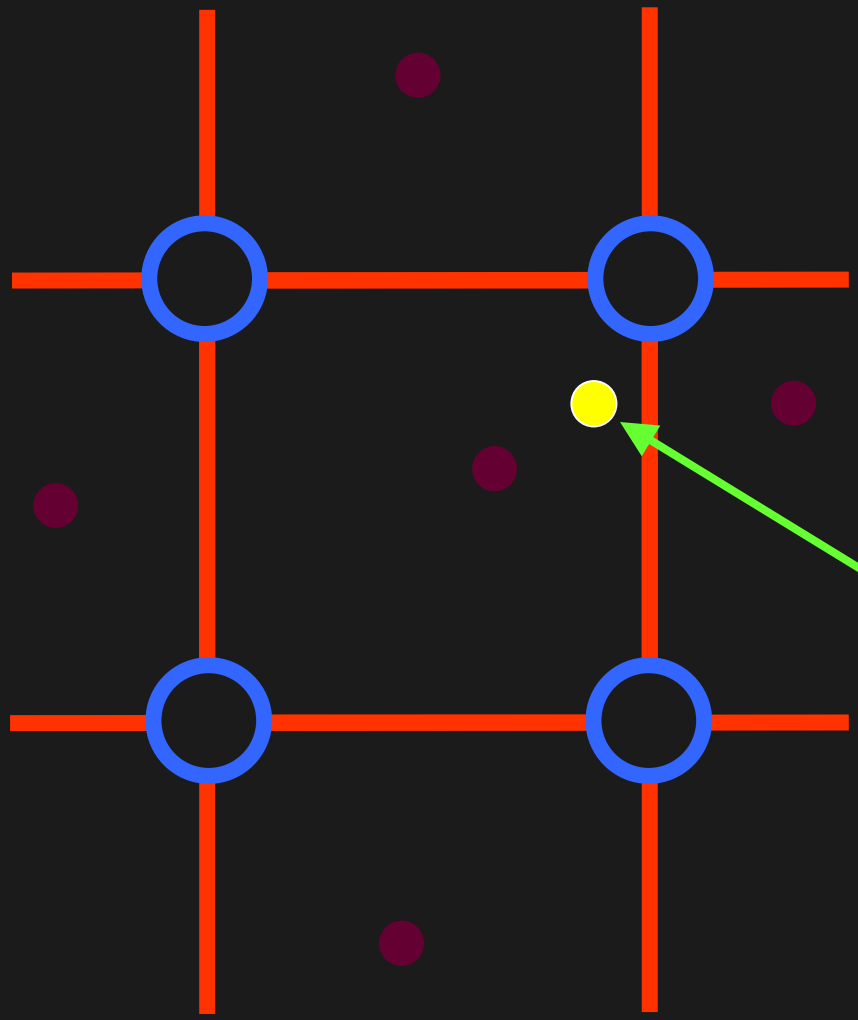


non-silhouette pixel
value is 0

non-silhouette pixel
value is 1

silhouette pixel
0 < value < 1

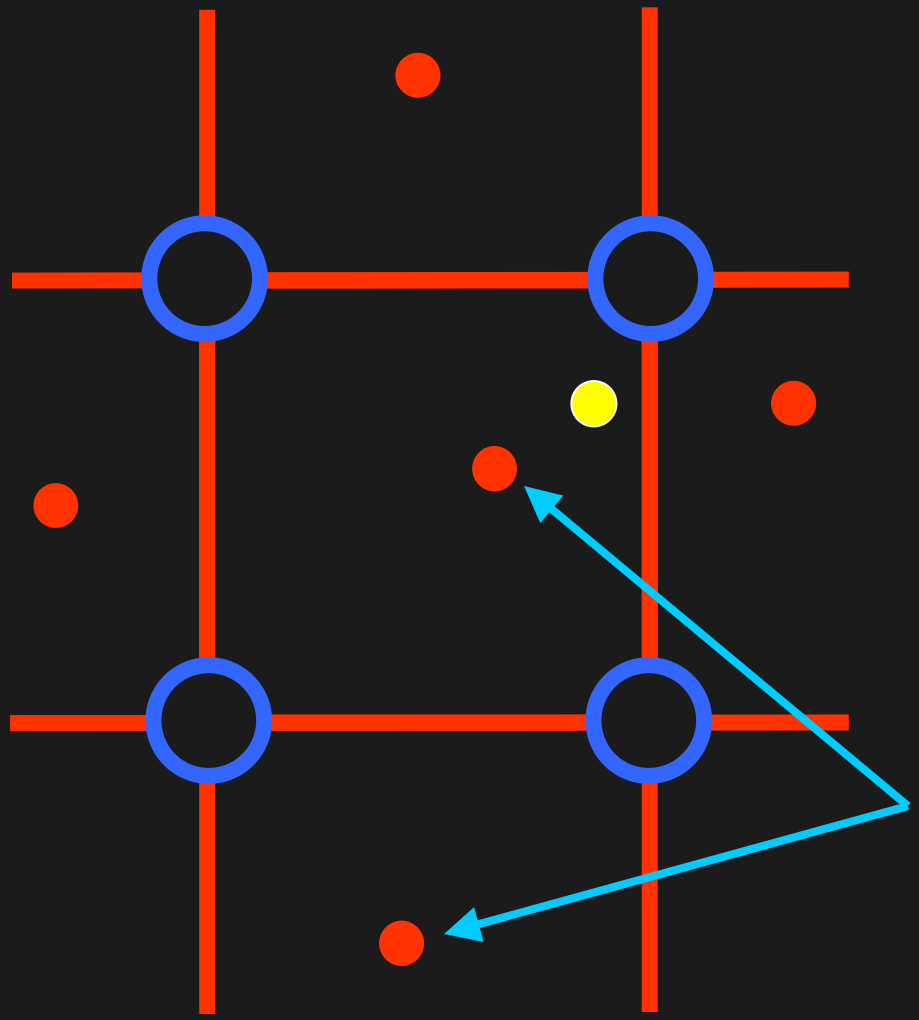
Silhouette Reconstruction



Use a fragment program to compute the shadows

sample point

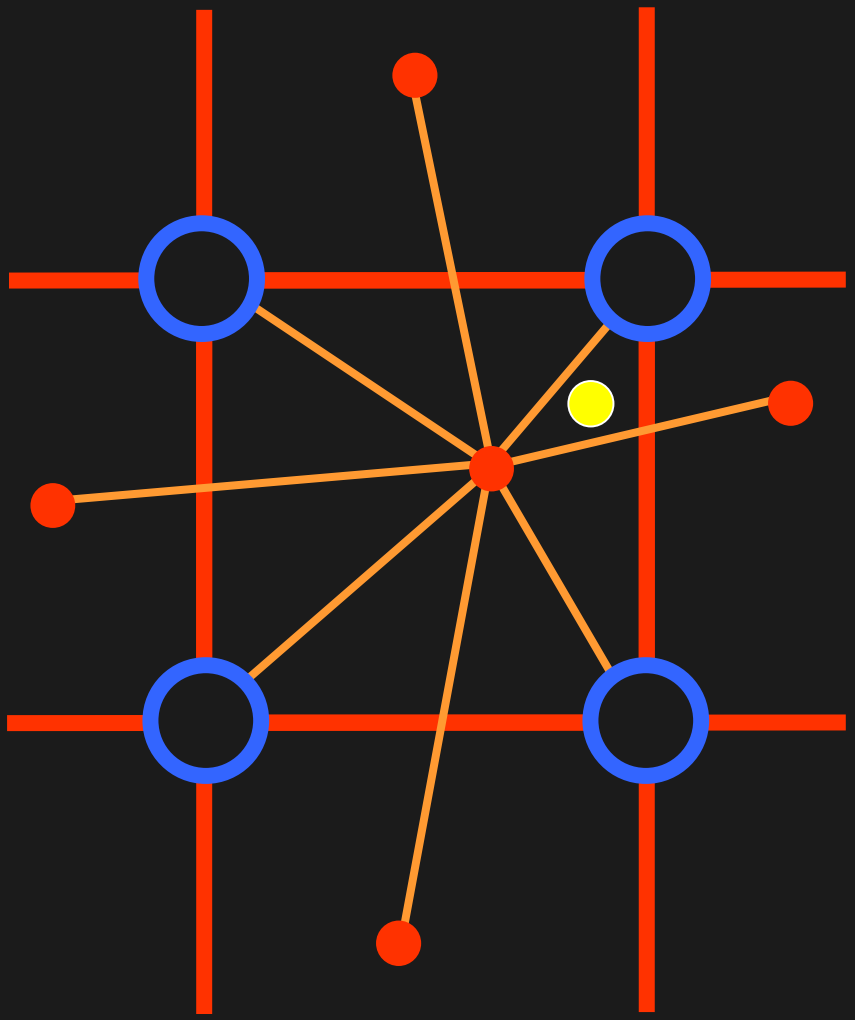
Silhouette Reconstruction



Fetch silhouette points
- 1 interior point
- 4 neighbors

silhouette points

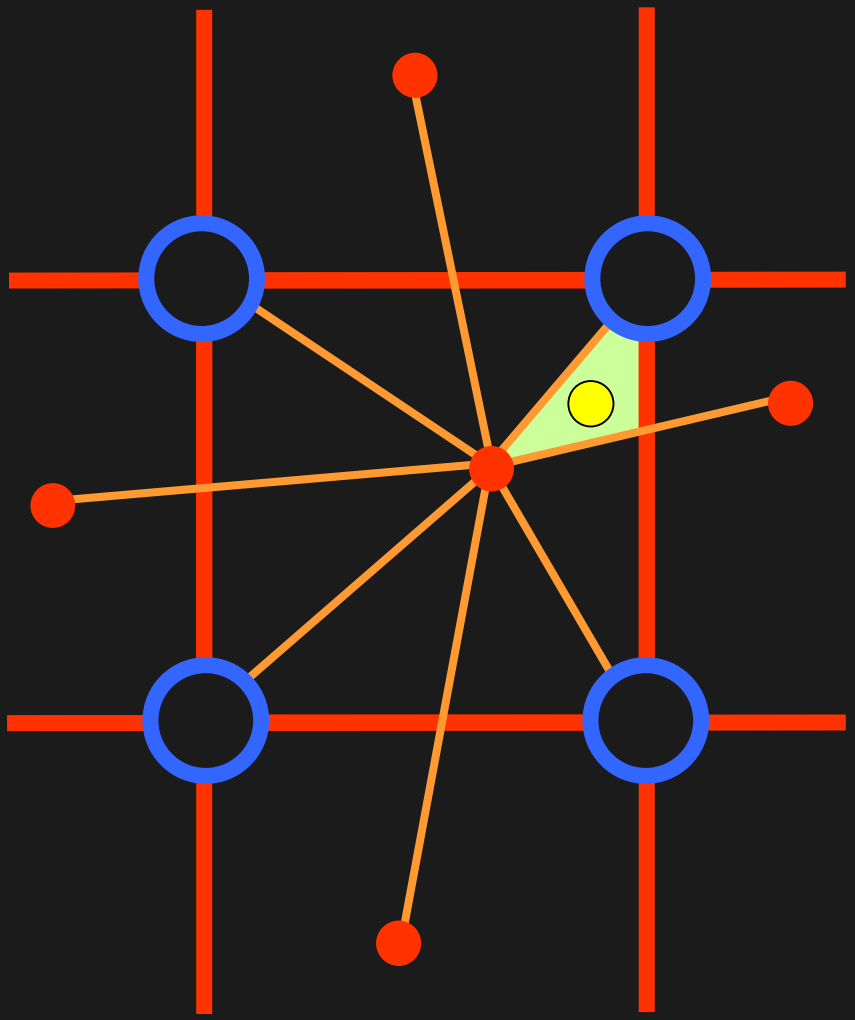
Silhouette Reconstruction



Fetch silhouette points
- 1 interior point
- 4 neighbors

Create eight wedges

Silhouette Reconstruction

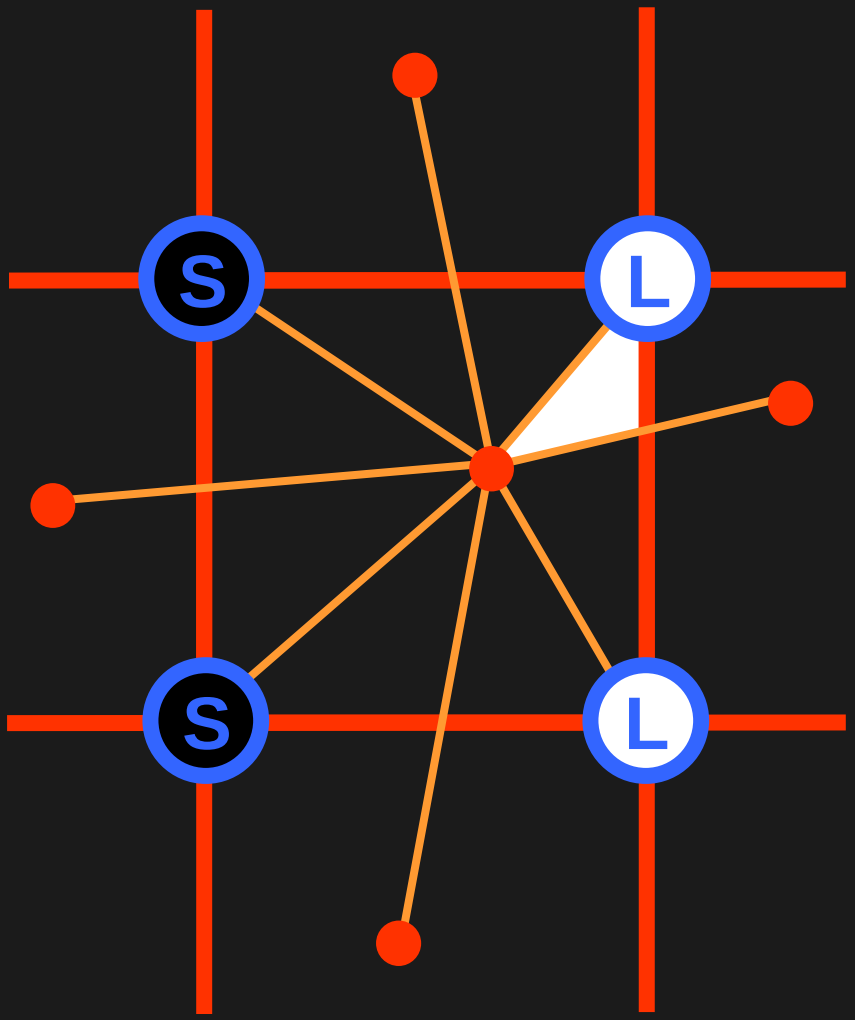


Fetch silhouette points
- 1 interior point
- 4 neighbors

Create eight wedges

Find enclosing wedge
- point-in-triangle tests

Silhouette Reconstruction



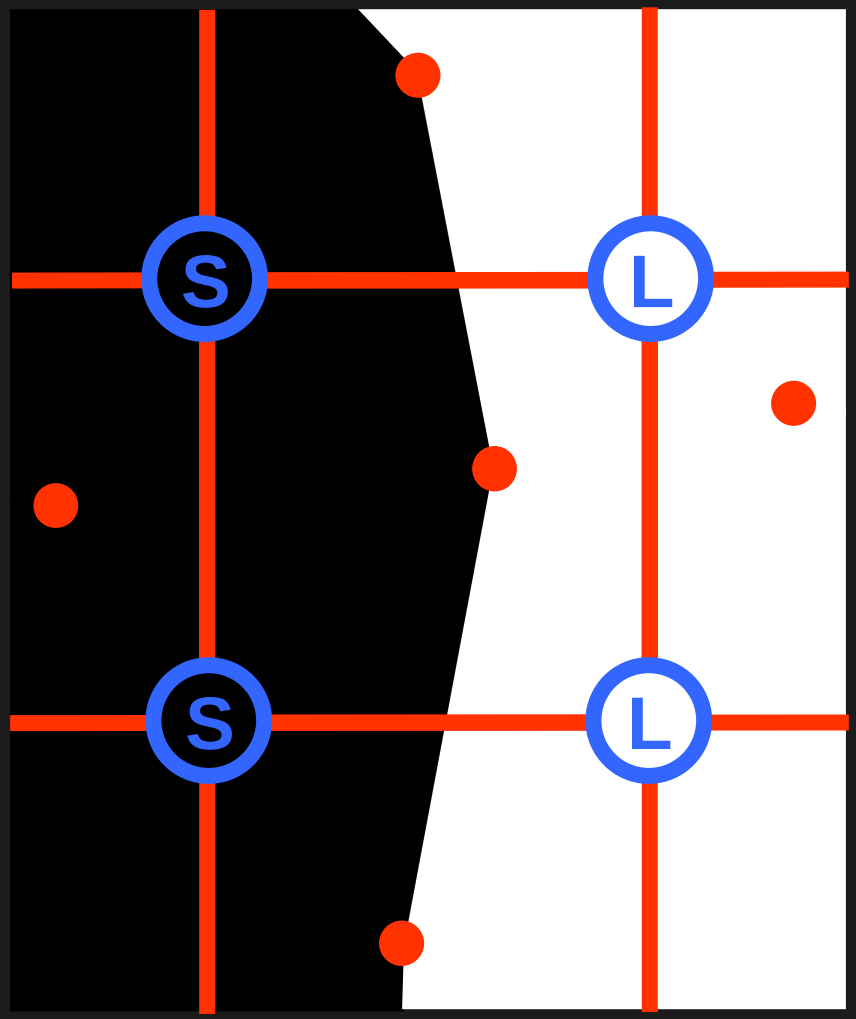
Fetch silhouette points
- 1 interior point
- 4 neighbors

Create eight wedges

Find enclosing wedge
- point-in-triangle tests

Shade the sample using wedge's depth test result

Silhouette Reconstruction



Fetch silhouette points
- 1 interior point
- 4 neighbors

Create eight wedges

Find enclosing wedge
- point-in-triangle tests

Shade the sample using wedge's depth test result

Repeat for all samples

Optimizations

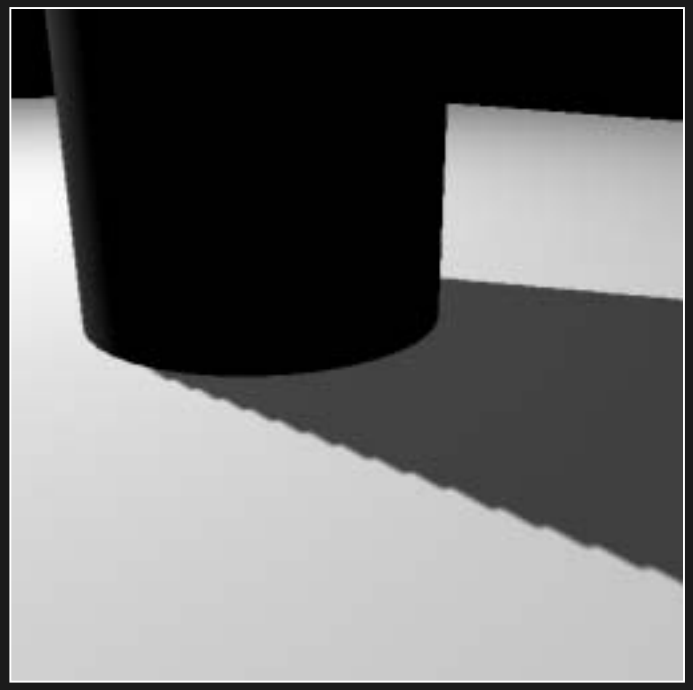
Fragment program is expensive

- lots of arithmetic
- lots of texture reads (5 silhouette points)

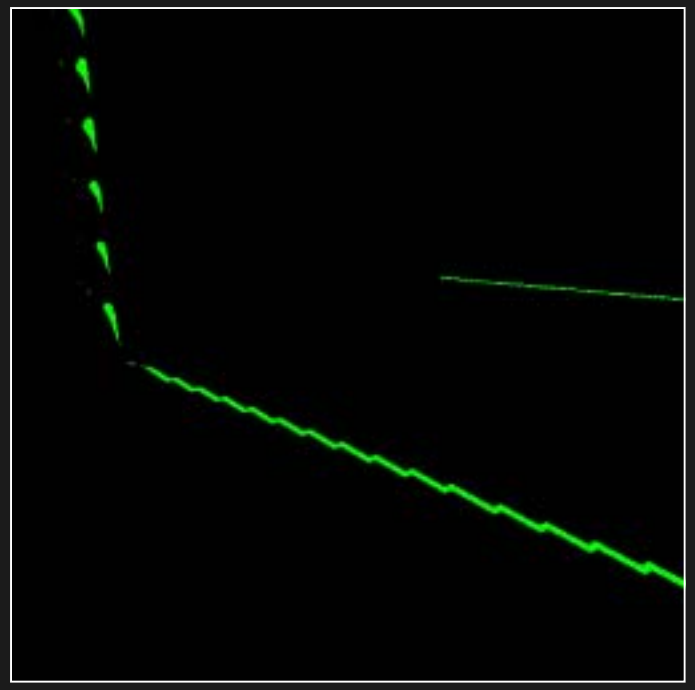
However, only required for silhouette pixels!

Optimizations

Very few silhouette pixels in practice



original scene



silhouette pixels
(1% total image)

Optimizations

Use fragment program branching

- Potentially huge performance wins
- Only available in latest hardware

Examples and Analysis

Example 1

shadow maps

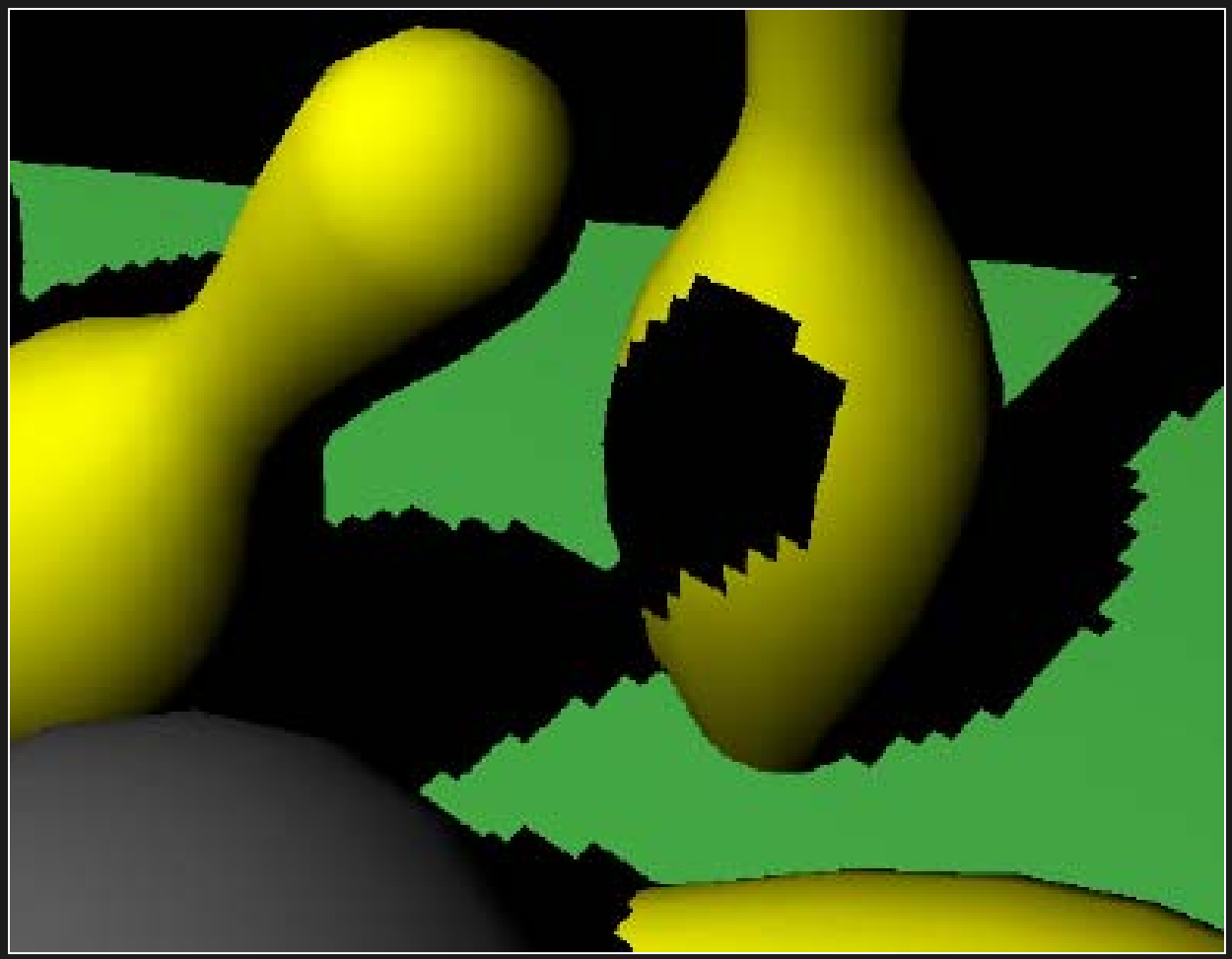
shadow volumes

silhouette maps

Example 1 (closeup)

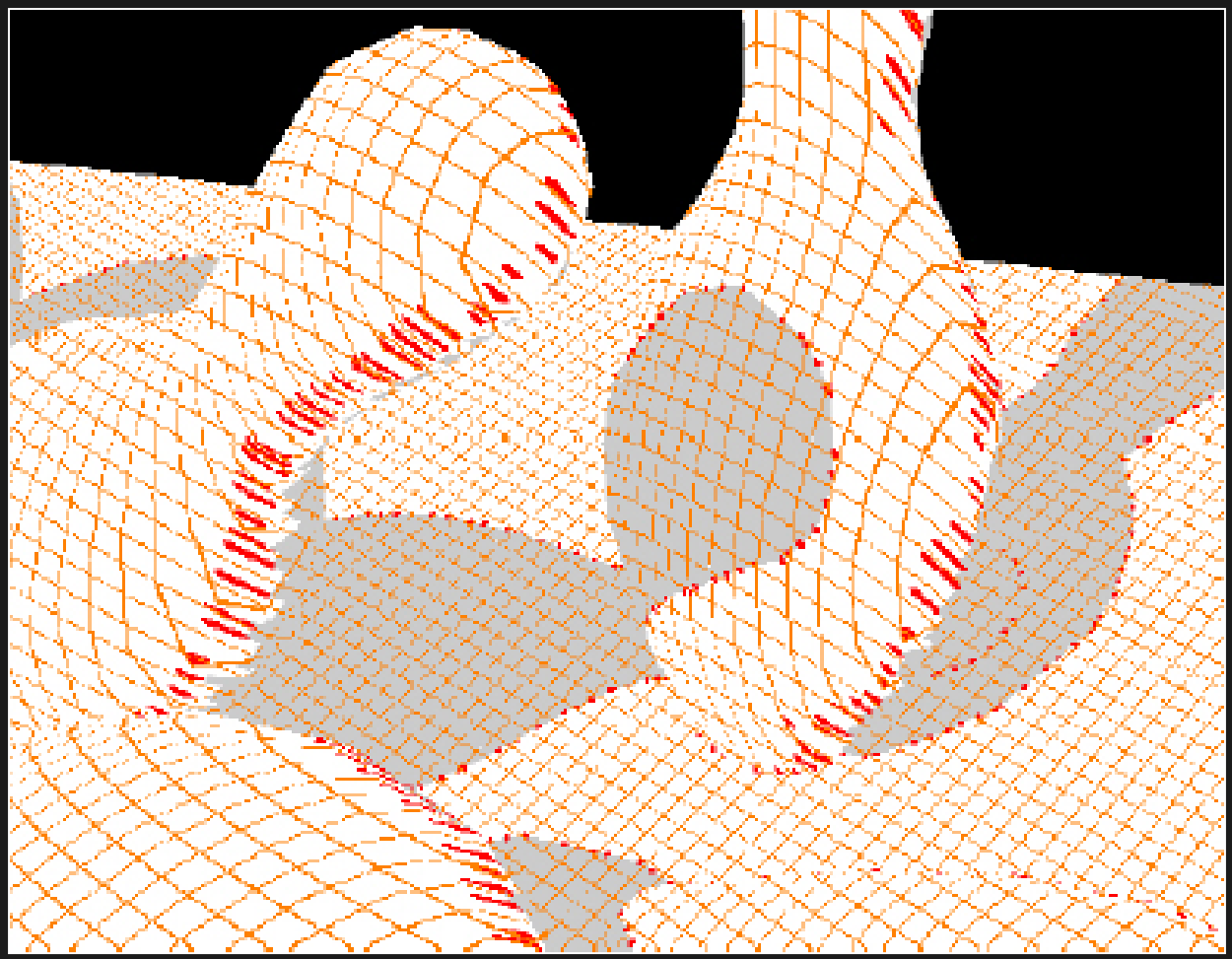
shadow maps | shadow volumes | silhouette maps

Example 2



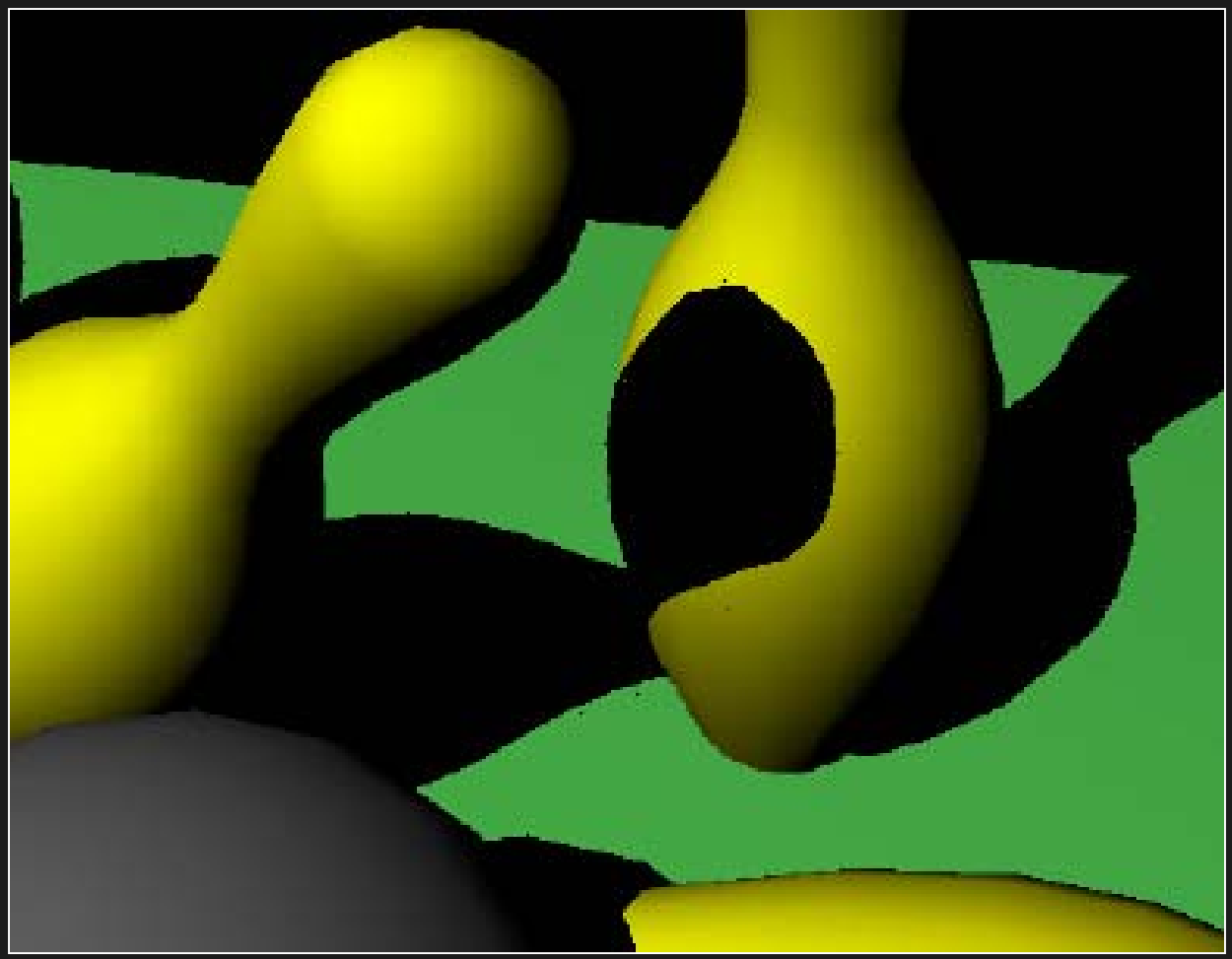
shadow maps

Example 2



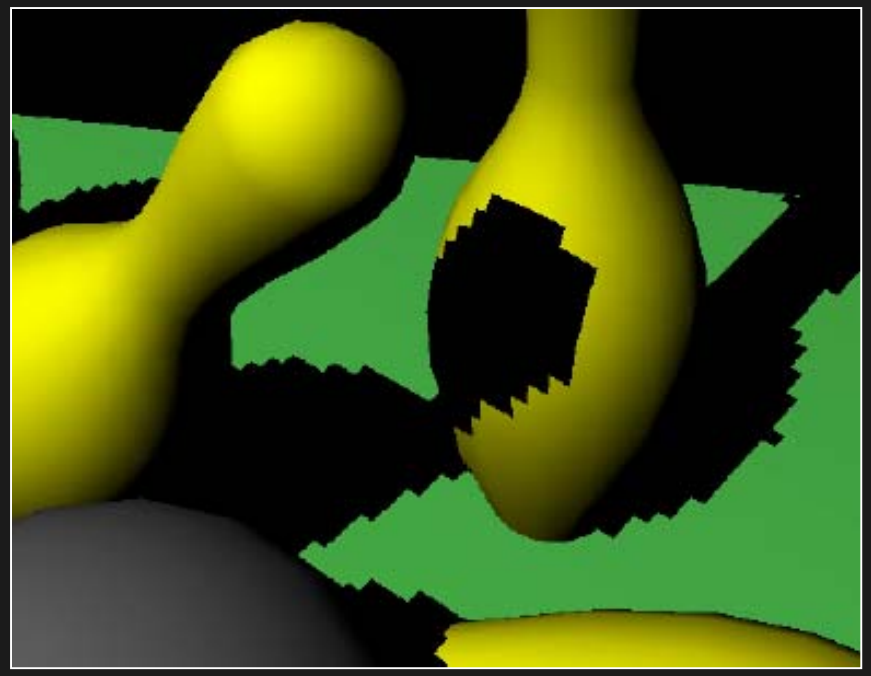
projected silhouette map

Example 2

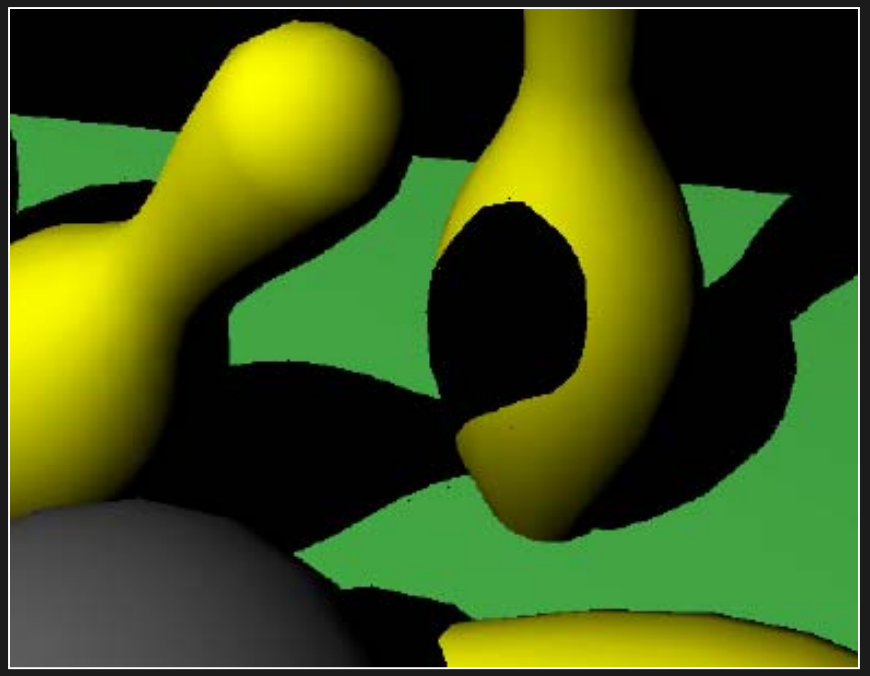


shadows using silhouette map

Quality Comparison



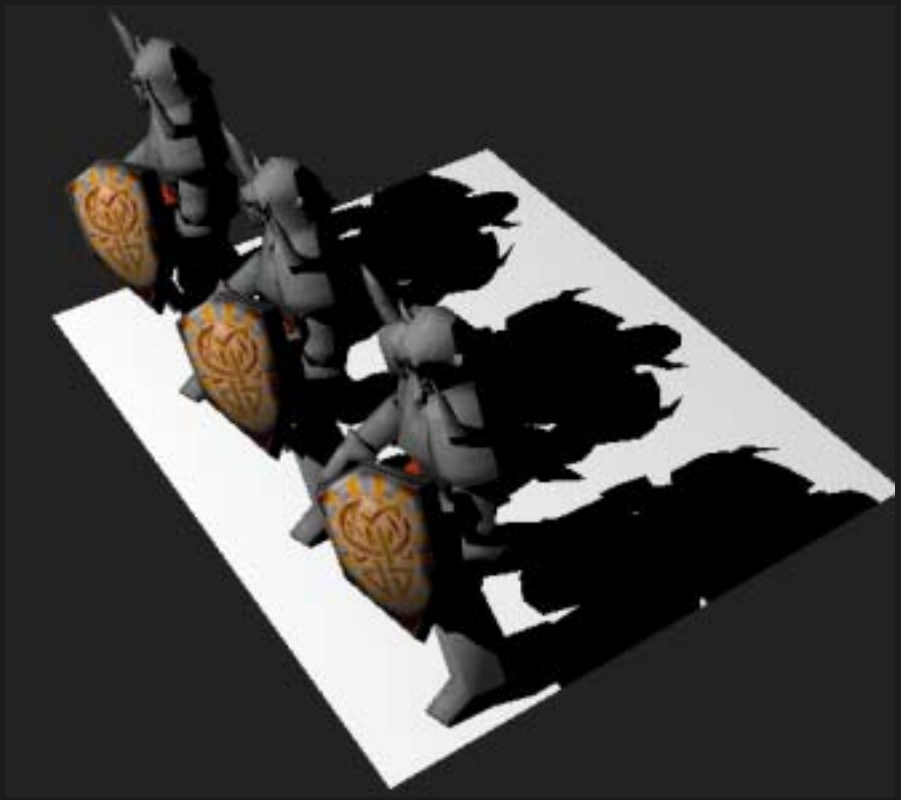
shadow map



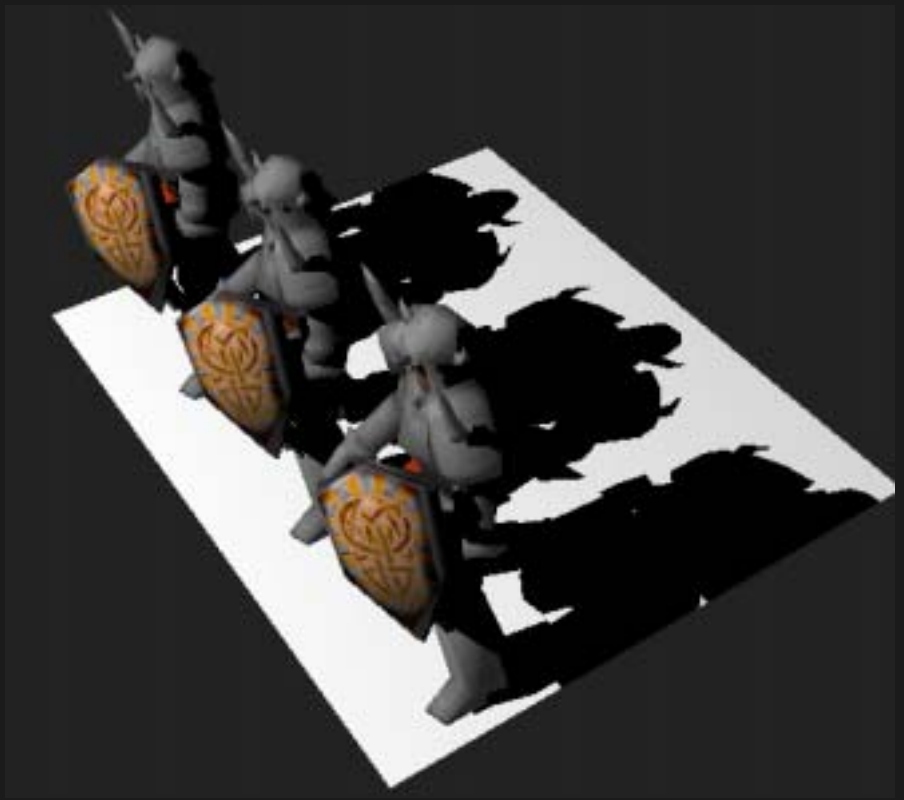
silhouette map

Bandwidth Comparison

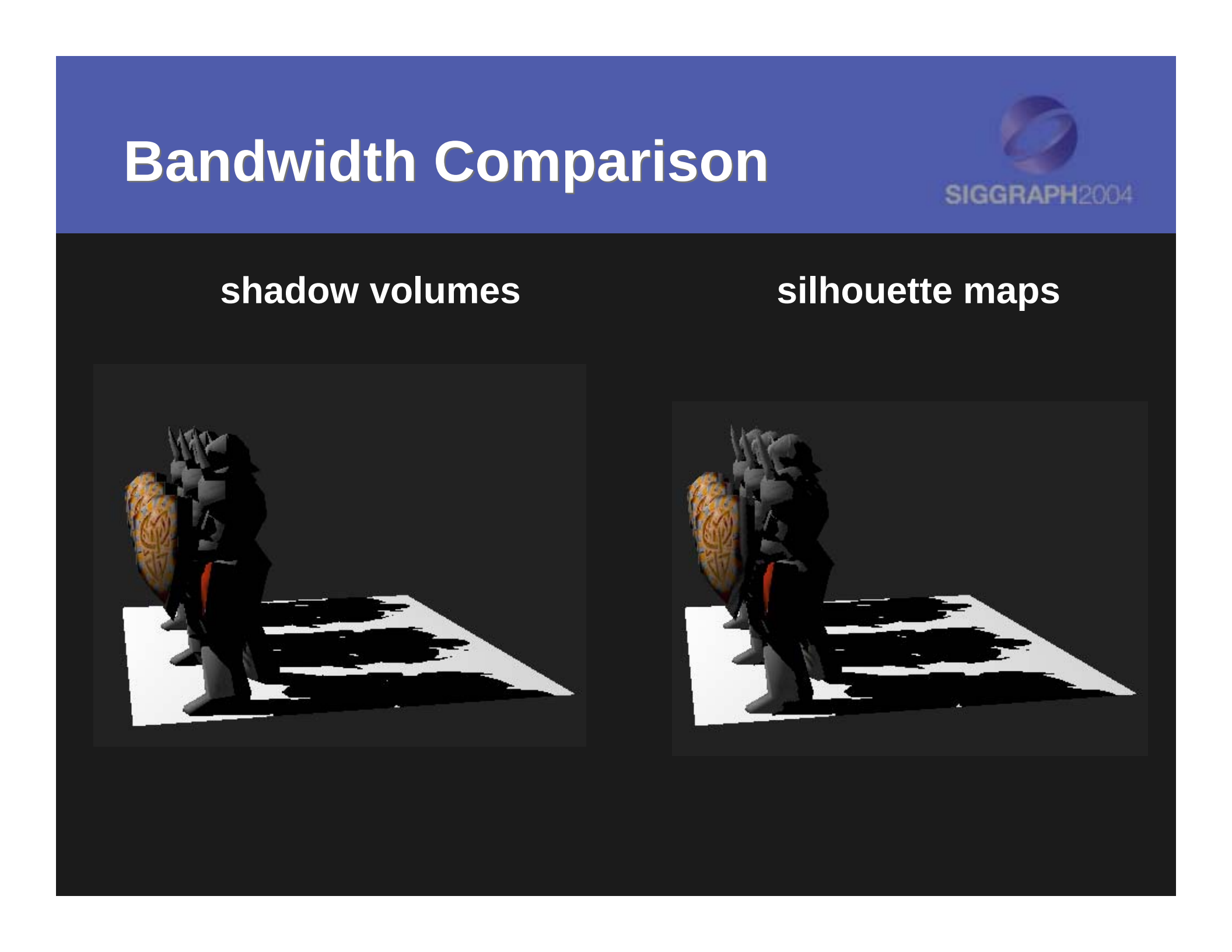
shadow volumes



silhouette maps



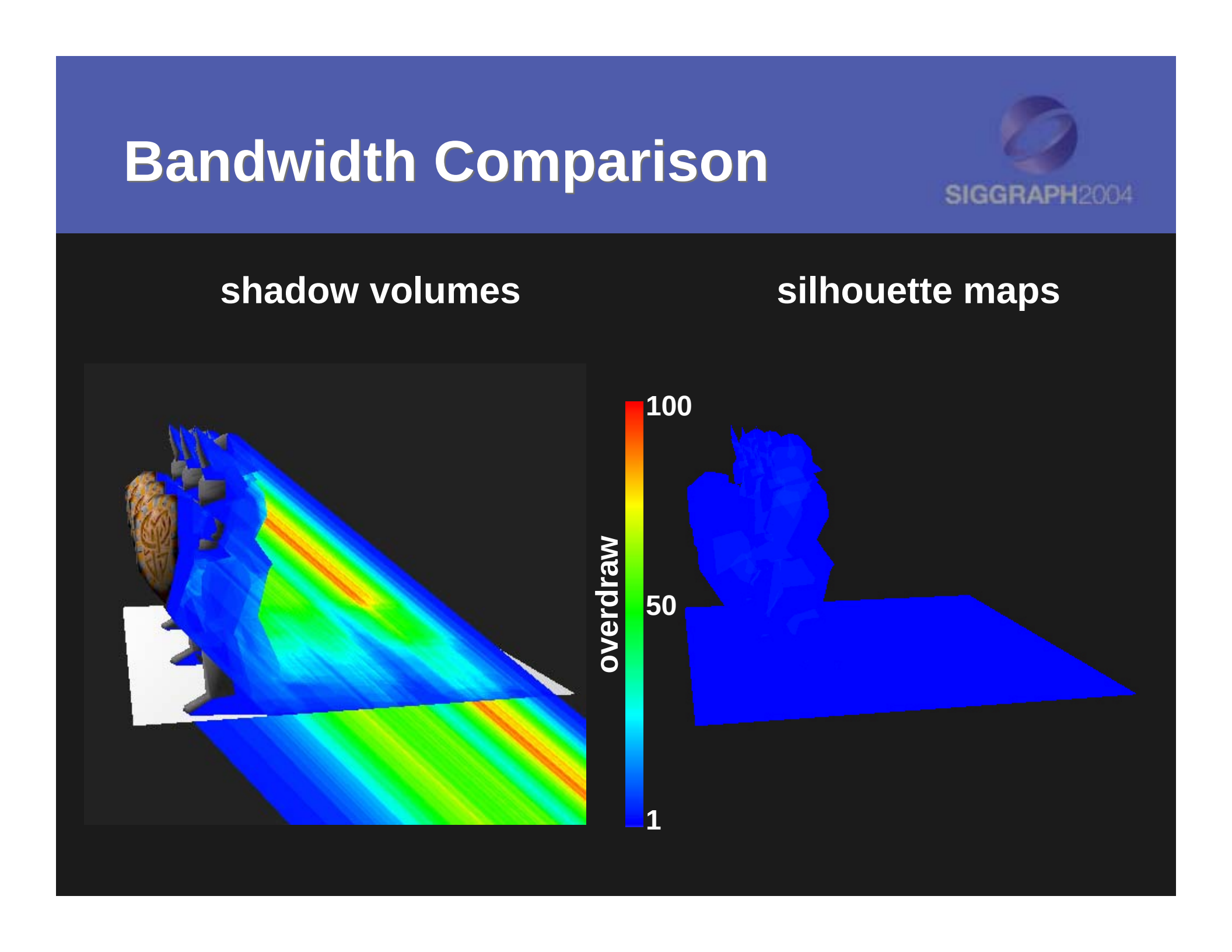
Bandwidth Comparison



shadow volumes

silhouette maps

Bandwidth Comparison

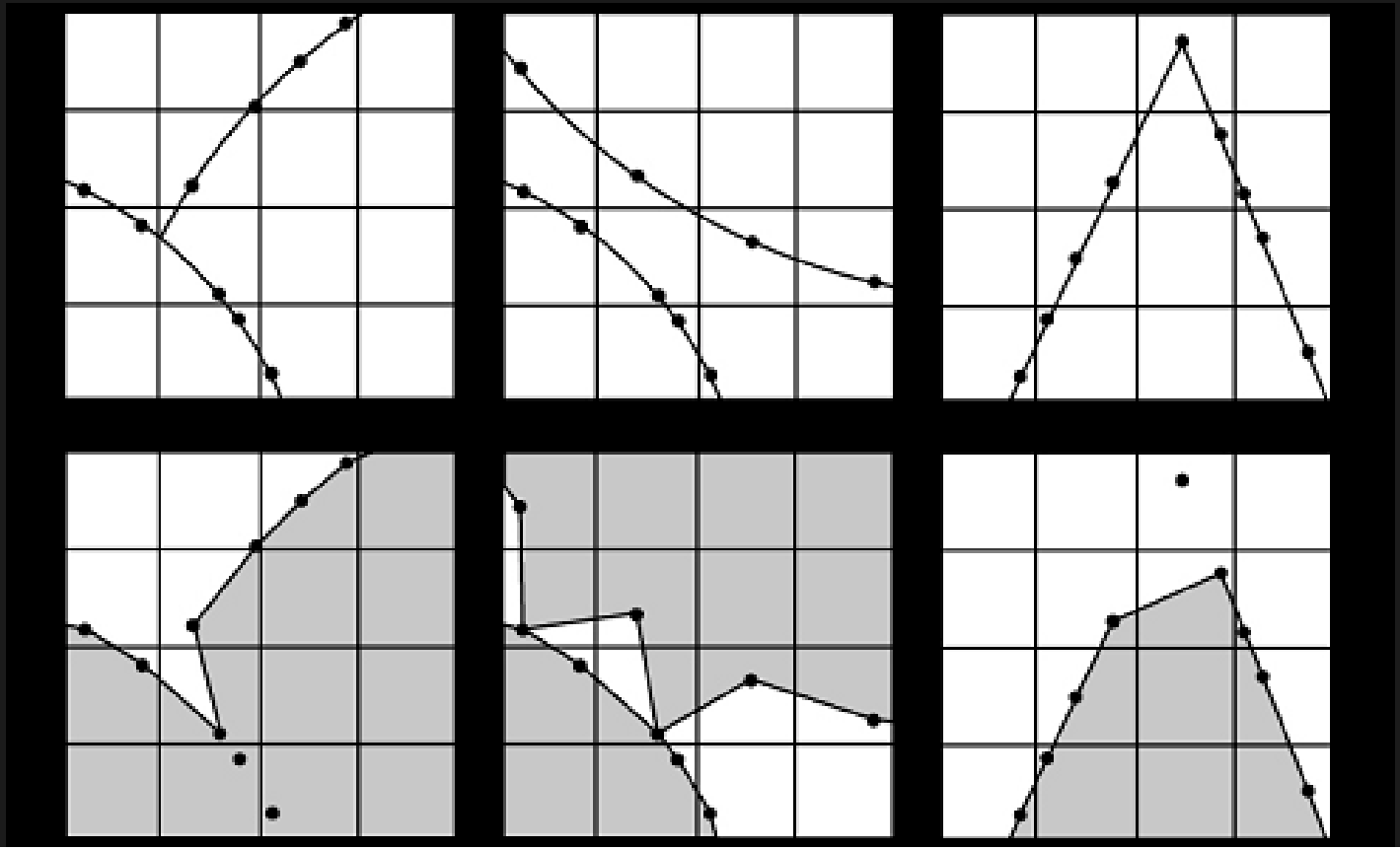


Bandwidth Comparison

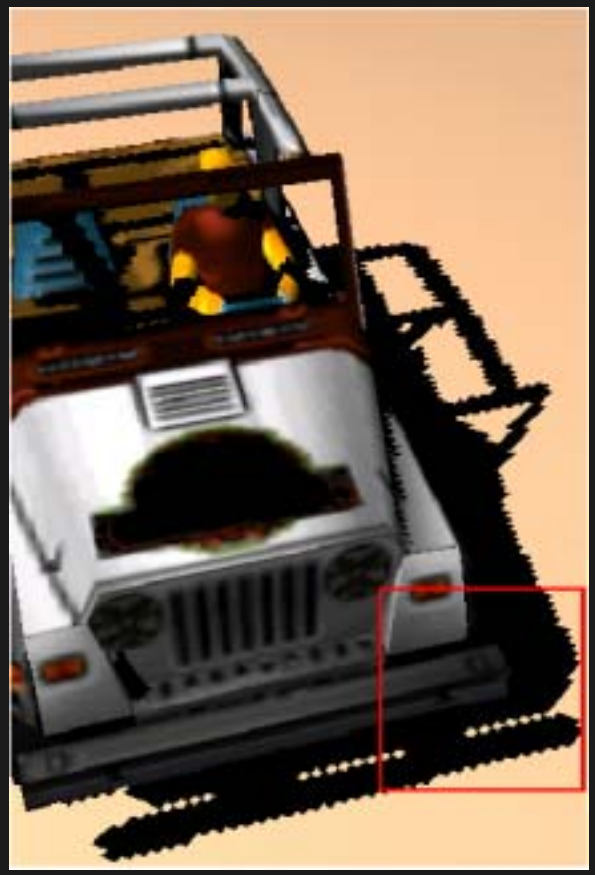
	1200 triangles	14,800 triangles
Shadow volumes	**5.94 MB**	**126.3 MB**
Silhouette maps	**1.53 MB**	**1.07 MB**
Bandwidth ratio	**3.9 : 1**	**118:1**

Artifacts

- Silhouette map: one point per texel
- Multiple edges inside a texel

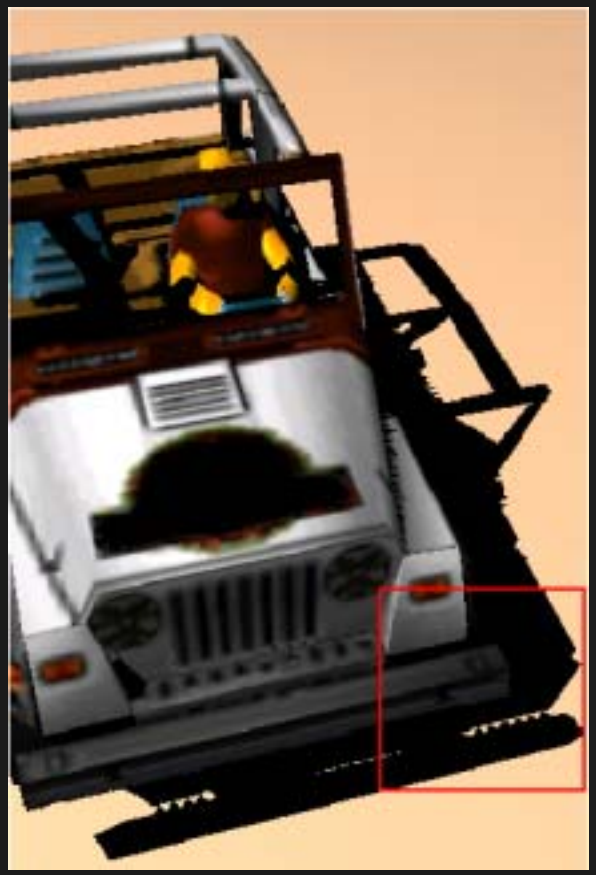


Artifacts



shadow maps

shadow volumes

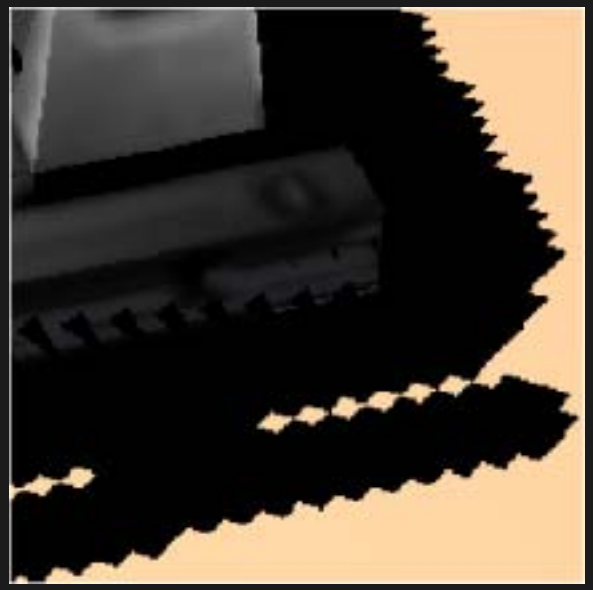


silhouette maps

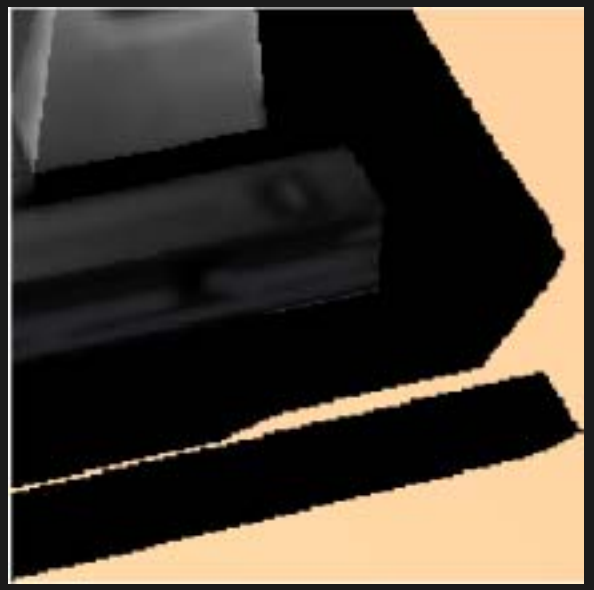
Artifacts (closeup)

Artifacts due to multiple edges

More noticeable when animated



shadow maps



shadow volumes

silhouette maps

Algorithm Comparison

Perspective Shadow Maps:

- same generality as shadow maps
- minimal overhead (2 passes)
- doesn’t address aliasing in all cases

Shadow Silhouette Maps:

- addresses aliasing more generally
- more overhead (3 passes + big shaders)
- less general than shadow maps

Combination of Algorithms

Why not combine techniques?

Perspective shadow map:

- Optimizes depth sample distribution
- More samples closer to viewer

Shadow silhouette map:

- Optimizes depth sample information
- Exact silhouette edge locations

Summary

- Image-space algorithm
- Silhouette map: deformed depth map
- Piecewise-linear approximation
- Scalable (compared to shadow volumes)

Compared to (perspective) shadow maps:

- Removes aliasing in more cases
- Additional overhead and requirements

SIGGRAPH2004