Lecture 23:

Advanced Rasterization

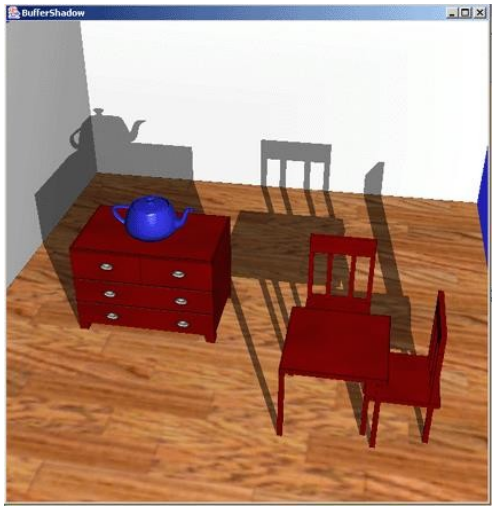
Contents

Occlusion queries: simplified Ray-Tracing operations

- Normal ray-scene intersection:
  - find **nearest** intersection with scene
- Occlusion-query:
  - find **any** intersection with scene (slightly faster)
- Rasterization context:
  - ray-scene intersection operation is not available

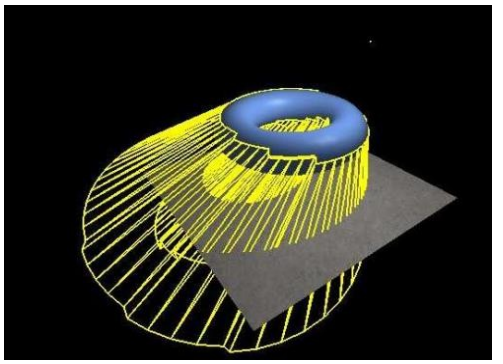
Projective Shadows (on plane)

- Project all vertices onto (offset) receiver plane
- Draw black triangles with (*e.g.* 50%) transparency
- Must avoid multiple overdraw (“double blending”)



Shadow Volume

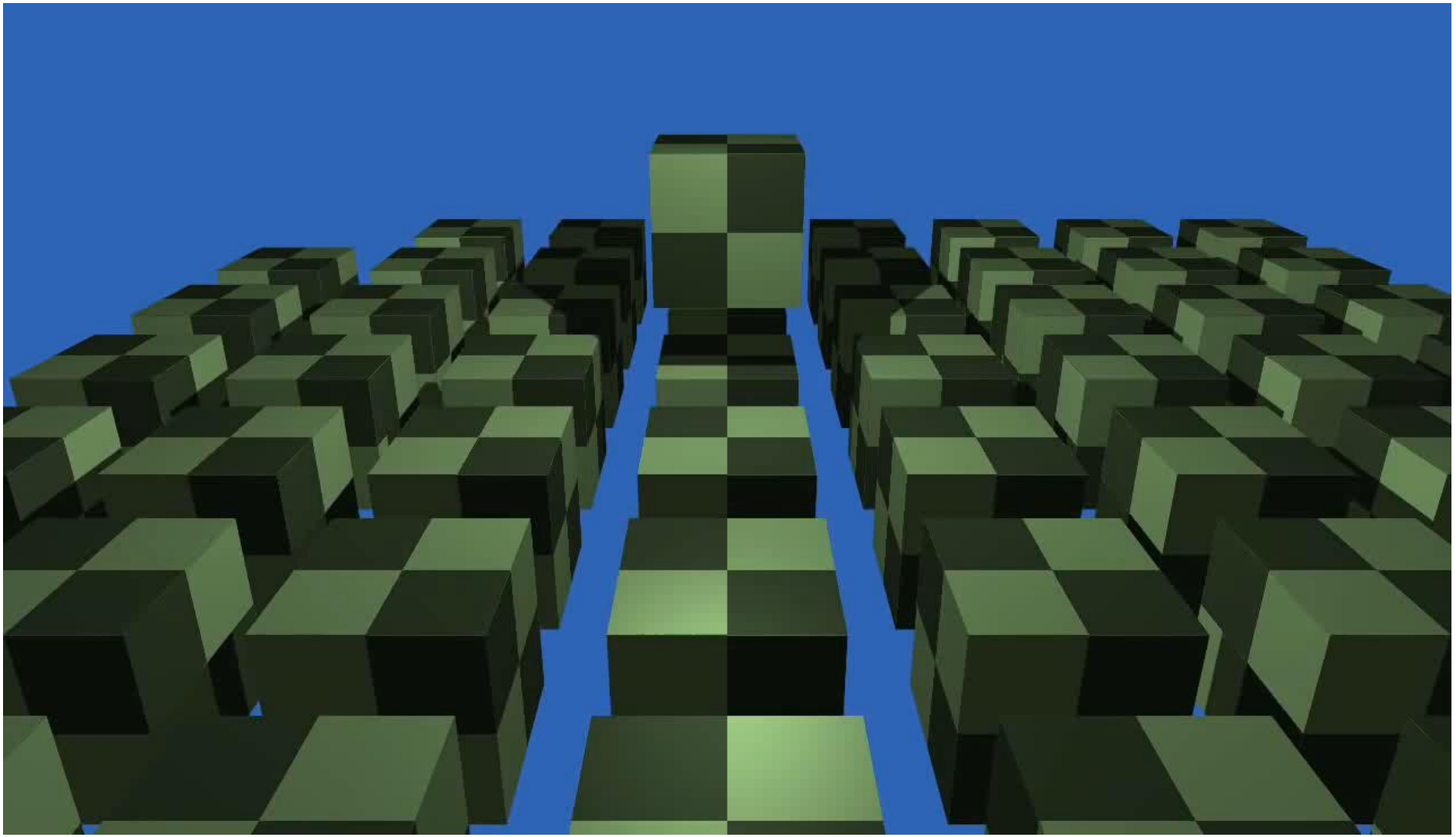
- Is a **mesh**, constructed by extruding volume of an object away from the light source
- Draw scene into the stencil buffer:
  1. Set stencil to 0 (1 if camera is inside volume)
  2. Draw volume, culling back faces, incrementing stencil buffer
  3. Draw volume, culling front faces, decrementing stencil buffer
  4. Draw scene with direct lighting, but only where stencil == 0
  5. Repeat for every light source



Shadow Maps

- Is a **depth texture** of the scene rendered from the light source in a certain direction

Shadow Volumes

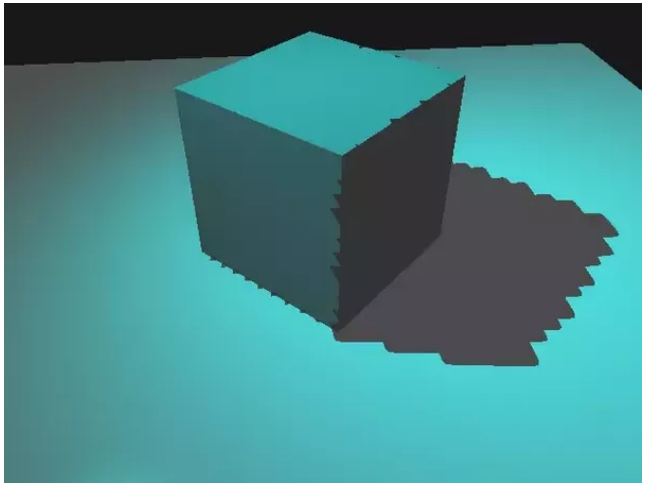


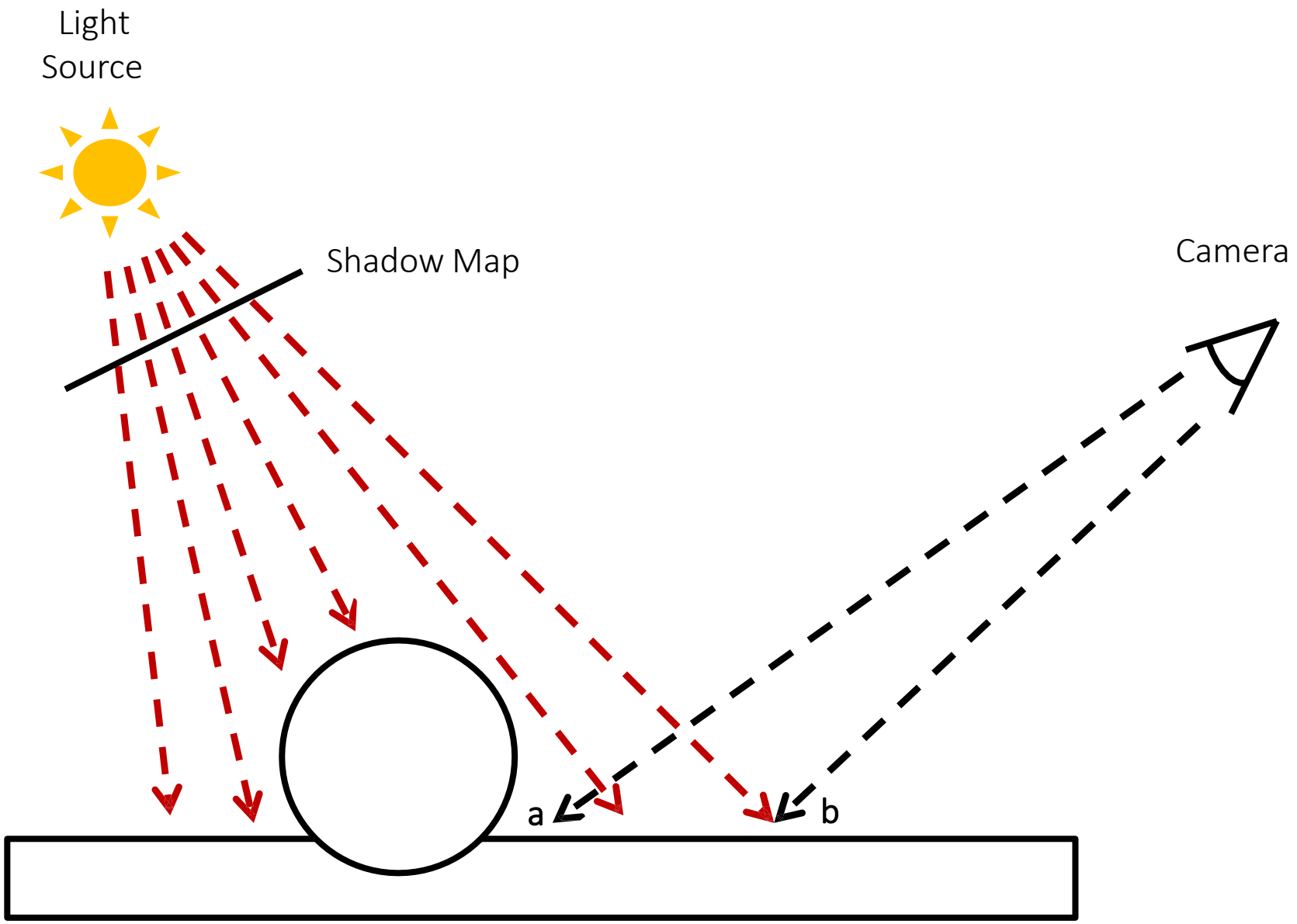
Problem of Shadow Volumes

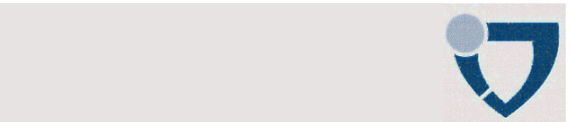
- Can have huge overdraw for complex objects – expensive (especially when polygons span the view frustum)

Idea:

- Render the scene from the viewpoint of the light, storing depth
- At each pixel, transform the visible point into view from the light
  - Computing pixel and depth in that view (simple matrix transform)
  - Compare depth to the depth value, stored in the light map
  - If map depth is smaller, than the point is in shadow – skip
    - Otherwise do normal shading and add color to frame buffer
- Repeat for every light source

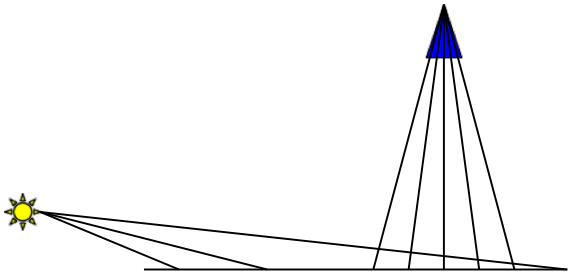



Light
Source
Shadow Map
Camera
a
b



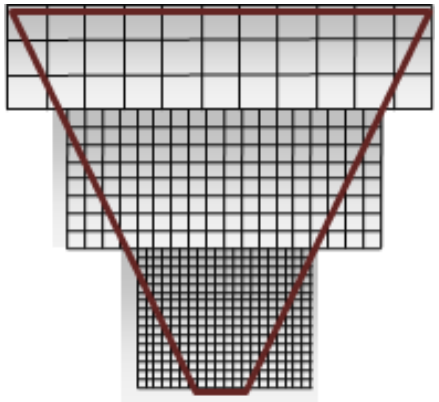
Sampling

- Shadow maps are discretely and regularly sampled (*e.g.* grid)
- Surfaces can have arbitrary orientation with respect to light
  - Can result in very bad sampling of a surface
- Essentially impossible to solve
  - Would need adaptive sampling
  - But the shadow map has to be generated in advance, no feedback
  - Solved in ray tracing, as we generate the sample adaptively

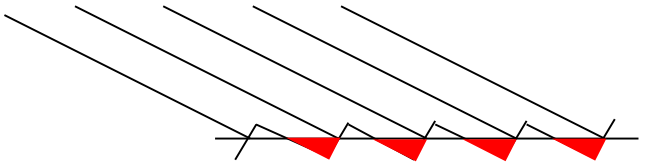


Resolution

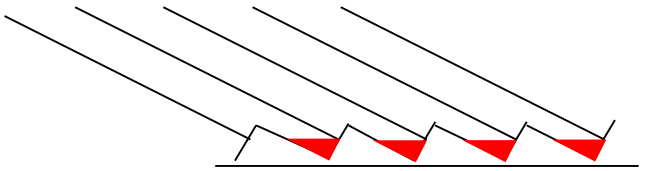
- Objects far from the camera should not be sampled finely
  - But shadow maps use a fixed grid
- Must adapt to preferred resolution
  - Use several resolutions
    - *E.g.* Split or Cascaded Shadow Maps
  - Transform geometry appropriately
    - *E.g.* Perspective or Trapezoid Shadow Maps



Interpolation / Filtering



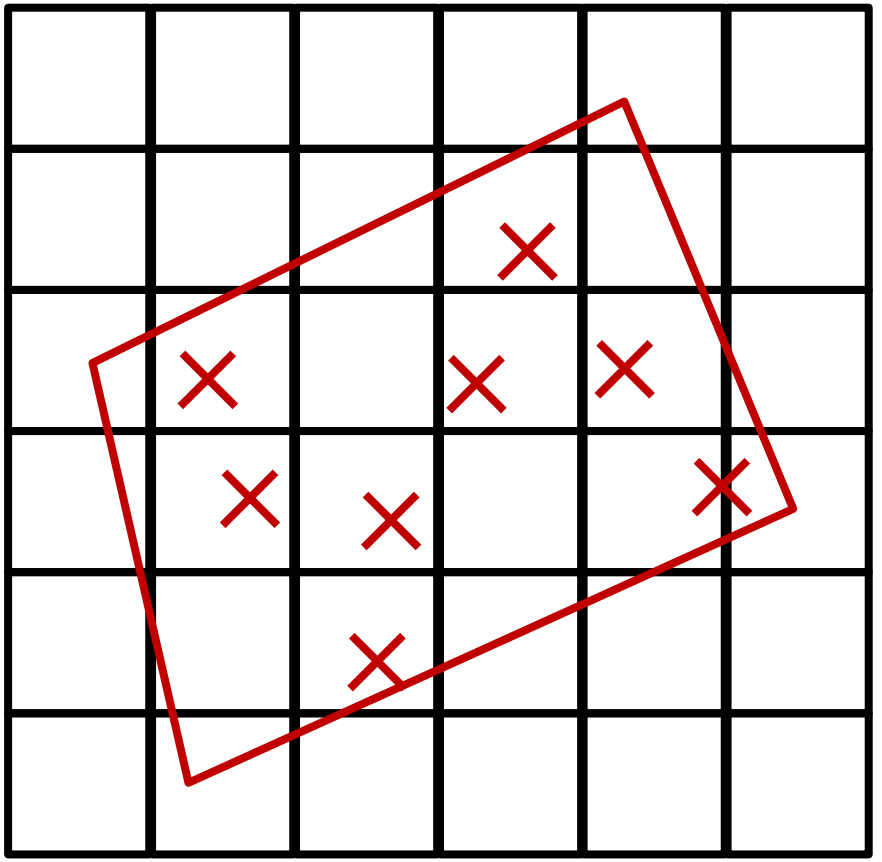
- Shadow maps contain point samples
  - We know nothing about what happens in between
  - Regular leads to self-occlusion (in red)
- Essentially impossible to solve without area information
  - *E.g.* min / max on depth
- Approaches (selected)
  - Polygon offset
    - Simply shift the depth values by some value
    - Do so proportional to cos of angle



  - Percentage Closer Filtering:
    - In SW: Randomly sample pixel footprint and compute ratio
    - In HW: bi-linearly interpolate depth difference from neighboring pixels
  - Variance Shadow Maps:
    - Store higher order information for better interpolation

Percentage-Closer Filtering

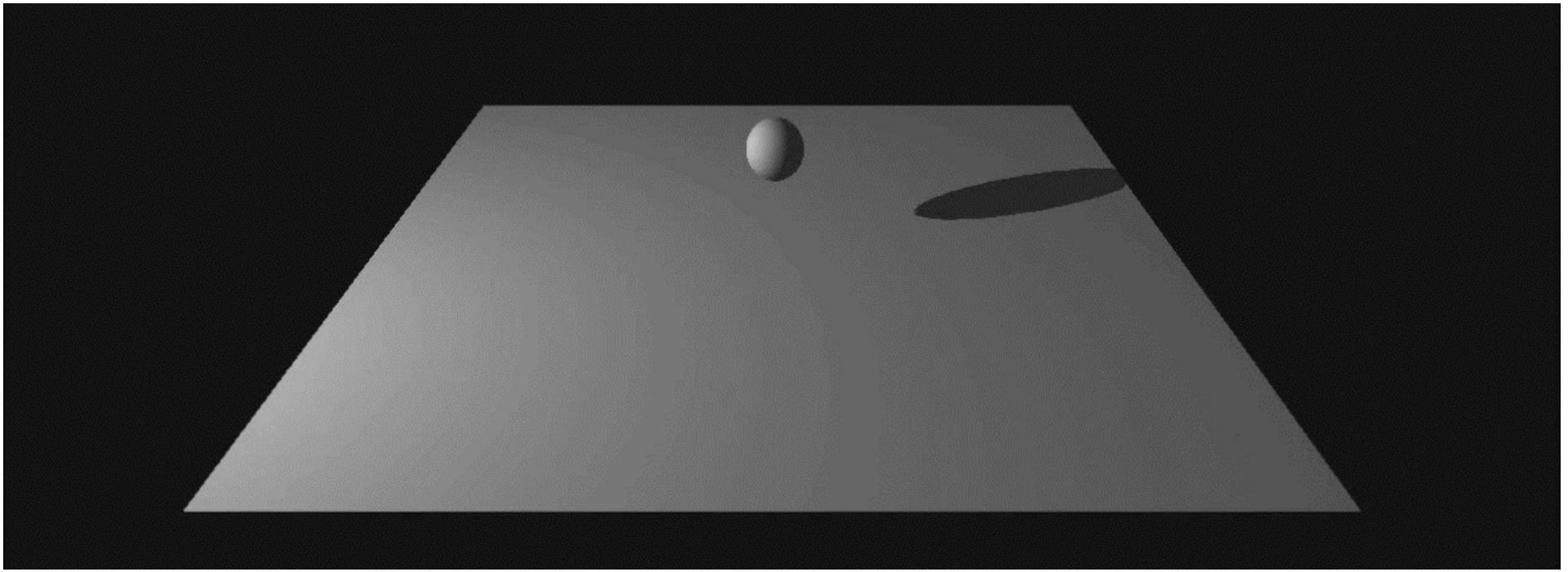
- Map area representing pixel to texture space
- Stochastically sample pixel to find percentage of surface in light

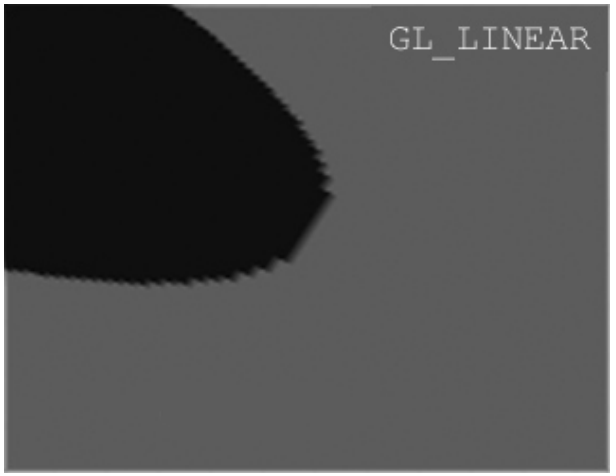


Shadow Map

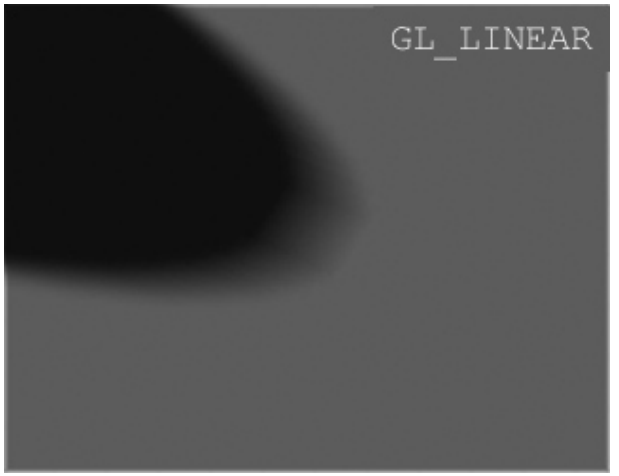
Pixel
(in texture space)

Percentage-Closer Filtering





Regular shadowmap



Percentage Closer Filtering 64 pixels kernel

Some Shadow Map Algorithms

Simple

- SSM "Simple"

Splitting

- PSSM "Parallel Split" http://http.developer.nvidia.com/GPUGems3/gpugems3_ch10.html
- CSM "Cascaded" http://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf

Warping

- LiSPSM "Light Space Perspective" http://www.cg.tuwien.ac.at/~scherzer/files/papers/LispSM_survey.pdf
- TSM "Trapezoid" http://www.comp.nus.edu.sg/~tants/tsm.html
- PSM "Perspective" http://www-sop.inria.fr/reves/Marc.Stamminger/psm/

Smoothing

- PCF "Percentage Closer Filtering" http://http.developer.nvidia.com/GPUGems/gpugems_ch11.html

Filtering

- ESM "Exponential" http://www.thomasannen.com/pub/gi2008esm.pdf
- CSM "Convolution" http://research.edm.uhasselt.be/~tmertens/slides/csm.ppt
- VSM "Variance" http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.2569&rep=rep1&type=pdf
- SAVSM "Summed Area Variance" http://http.developer.nvidia.com/GPUGems3/gpugems3_ch08.html

Soft Shadows

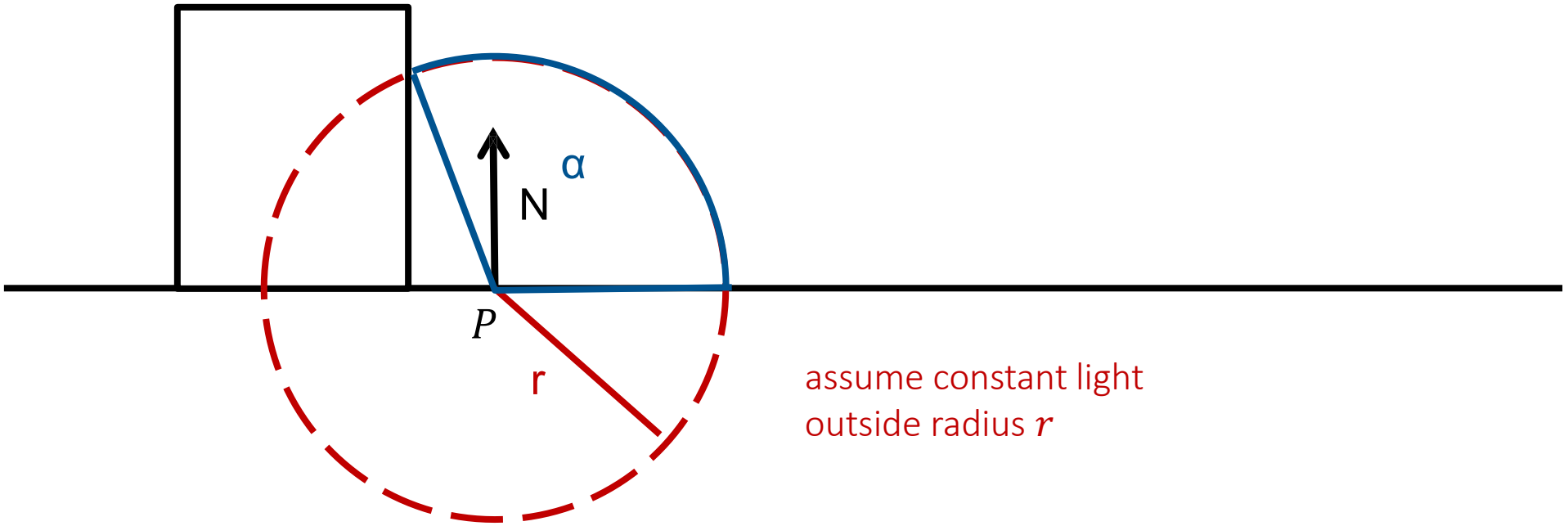
- PCSS "Percentage Closer" http://developer.download.nvidia.com/shaderlibrary/docs/shadow_PCSS.pdf

Assorted

- ASM "Adaptive" http://www.cs.cornell.edu/~kb/publications/ASM.pdf
- AVSM "Adaptive Volumetric" http://visual-computing.intel-research.net/art/publications/avsm/
- CSSM "Camera Space" http://free-zg.t-com.hr/cssm/
- DASM "Deep Adaptive"
- DPSM "Dual Paraboloid" http://sites.google.com/site/osmanbrian2/dpsm.pdf
- DSM "Deep" http://graphics.pixar.com/library/DeepShadows/paper.pdf
- FSM "Forward" http://www.cs.unc.edu/~zhangh/technotes/shadow/shadow.ps
- LPSM "Logarithmic" http://gamma.cs.unc.edu/LOGSM/
- MDSM "Multiple Depth" http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.3376&rep=rep1&type=pdf
- RMSM "Resolution Matched" http://www.idav.ucdavis.edu/func/return_pdf?pub_id=919
- SDSM "Sample Distribution" http://visual-computing.intel-research.net/art/publications/sdsm/
- SPPSM "Separating Plane Perspective" http://jgt.akpeters.com/papers/Mikkelsen07/sep_math.pdf
- SSSM "Shadow Silhouette" http://graphics.stanford.edu/papers/silmap/silmap.pdf

Calculates shadows against assumed constant ambient illumination

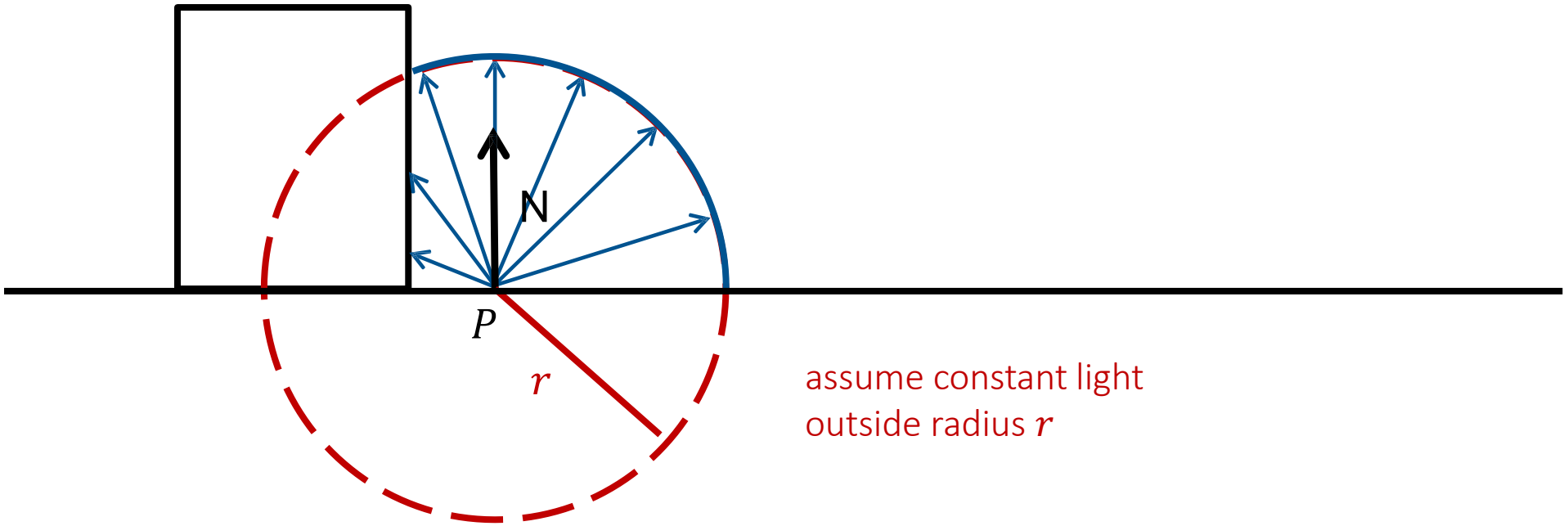
- Idea: in most environments, multiple light bounces lead to a very smooth component in the overall illumination
- For this component, incident light on a point is proportional to the part of the environment (opening angle) visible from the point
- Describes well contact shadows, dark corners



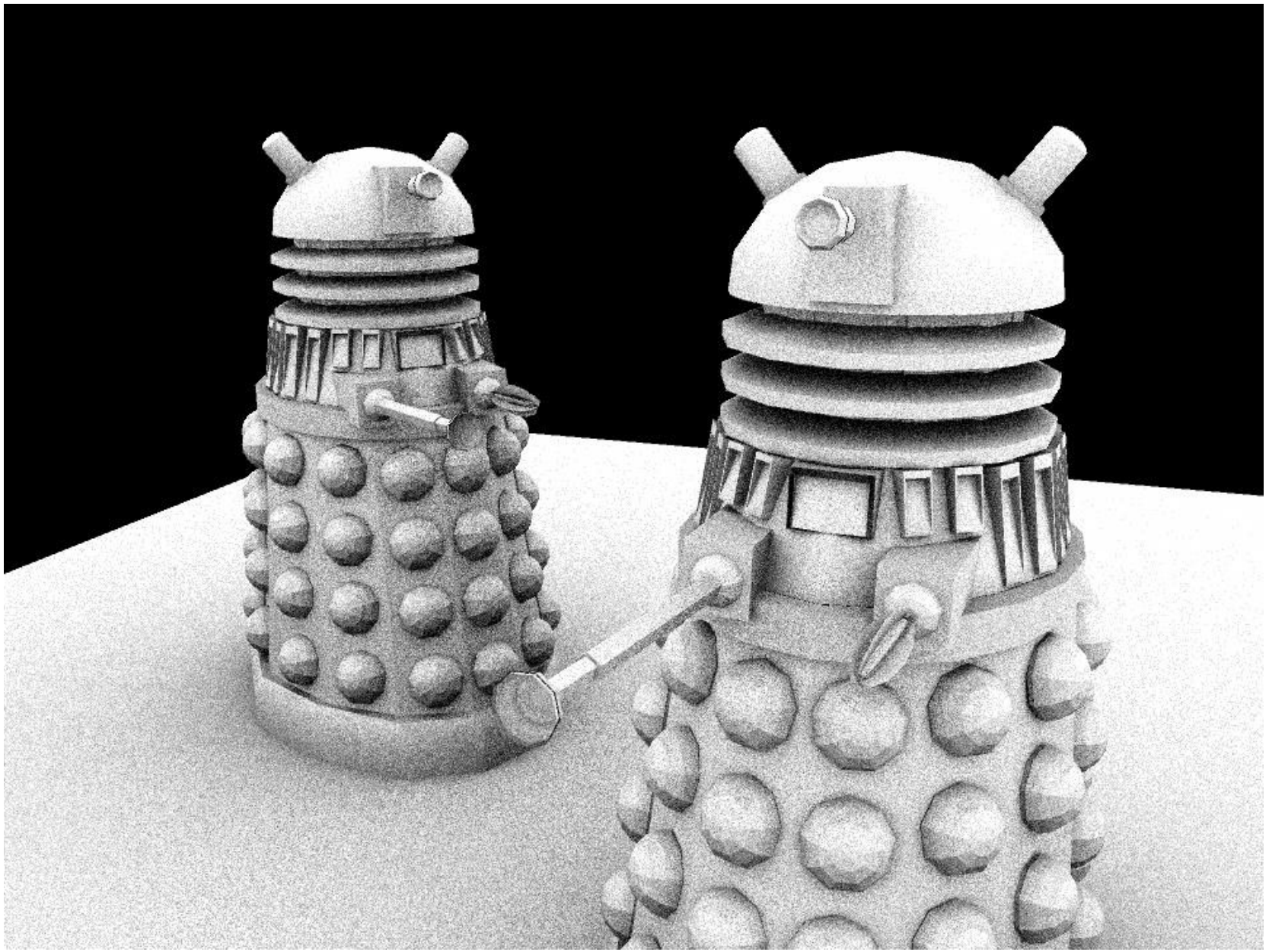
Ambient Occlusion (Visibility Map)

Computation using Ray-Tracing is straightforward

- Start at point P
- Sample n directions (D_1- D_n) from upper hemisphere
- Shot shadow rays from P to D_i with maximum length r
- Count how many rays reach the environment
- Gives correct result in the limit, but requires many rays to avoid noise (*i.e.* very slow)



Ambient Occlusion Using Ray-Tracing

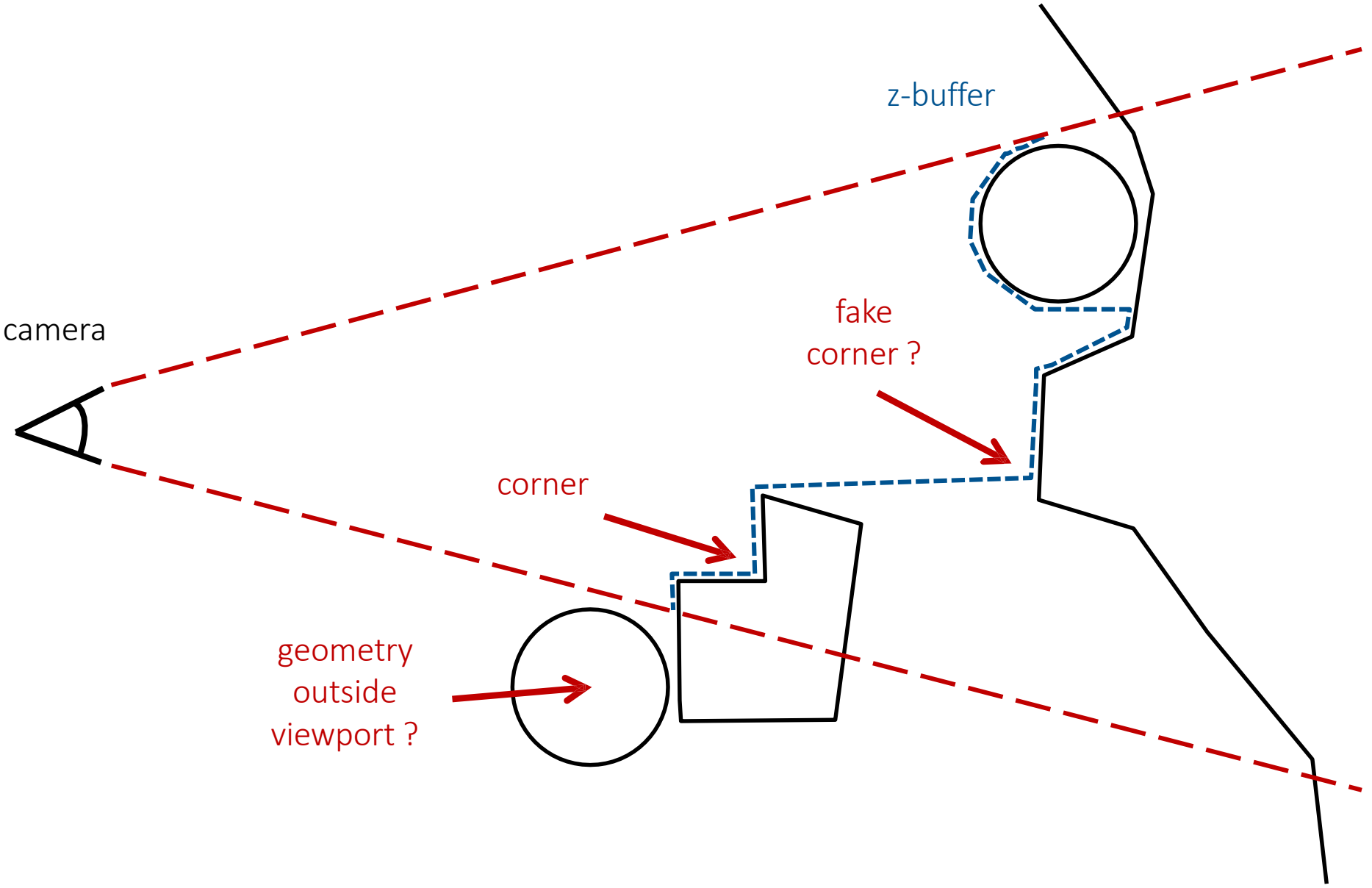


Can we approximate ambient occlusion in real-time?

Ray-scene intersection too slow

Idea: use z-buffer as scene approximation

- Horizontal and vertical position give position of point in x,y-direction (camera space)
- Z-buffer content gives position of point in z-direction (camera space)
- Contains discrete representation of all **visible** geometry
- Use ray-tracing against this simplified scene


z-buffer
camera
fake
corner ?
corner
geometry
outside
viewport ?

Tracing many rays is still expensive

- Often 200 and more samples are needed for good results

Approach

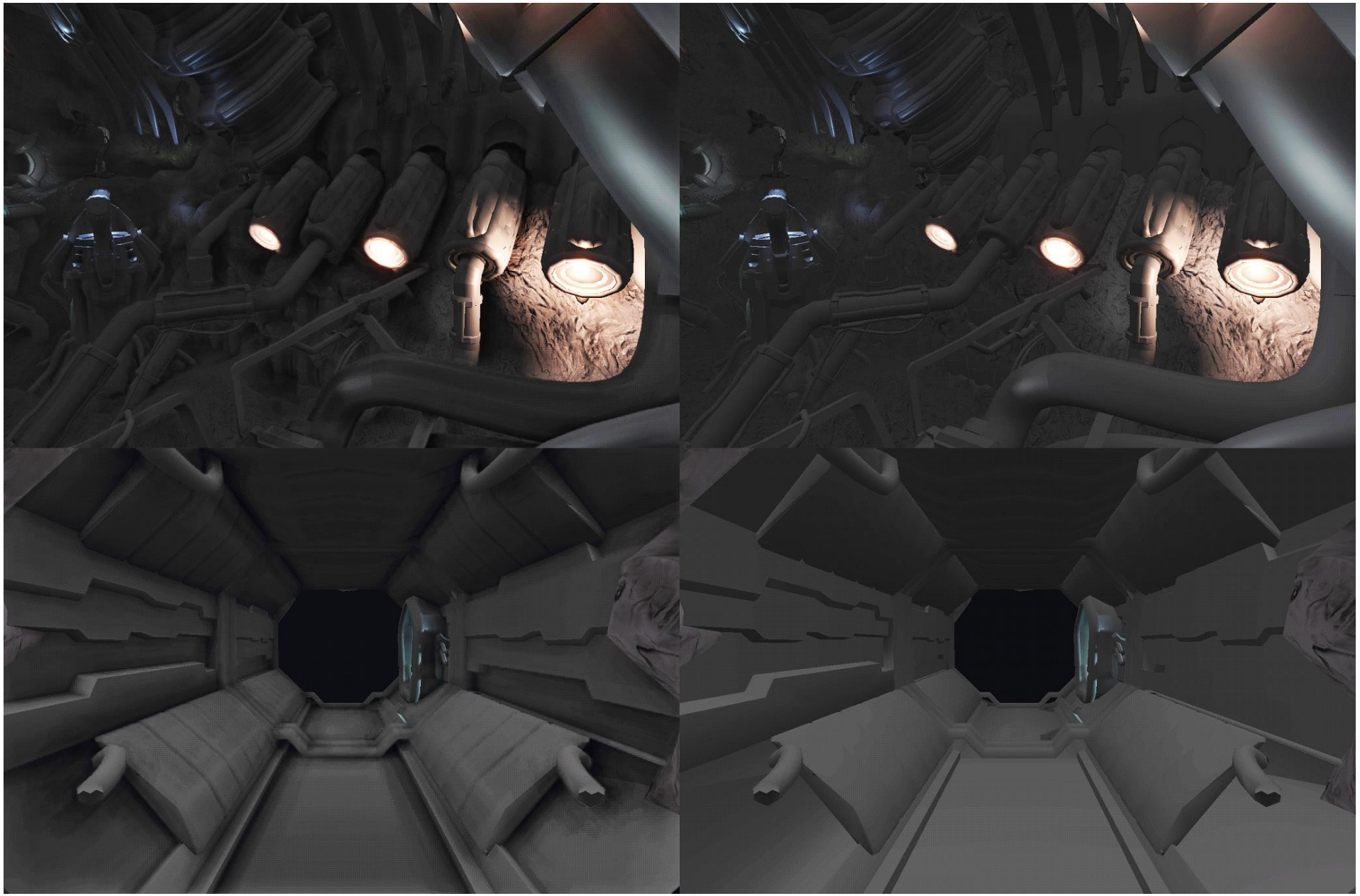
- For each pixel (Crytek approach, many others available)
  - Test a number of random points in sphere visible 3D point
    - Do not know surface orientation, so must test in all directions
  - If more than 50% pass we have full visibility
    - Otherwise scale AO with number of samples
  - Can still be quite costly
- Acceleration
  - Use different pseudo-random pattern for each pixel in NxN block
    - Gives slightly different values for each pixel
  - Filter over a NxN neighborhood
    - Uses all samples: *E.g.* 4x4 block with 16 samples each: 256 samples total
  - Make sure not to filter over wrong pixels (background)
    - Take distance, normal, *etc*. into account (→ bilateral filter)

Screen Space Ambient Occlusion

Without ambient occlusion

With ambient occlusion

Screen Space Ambient Occlusion



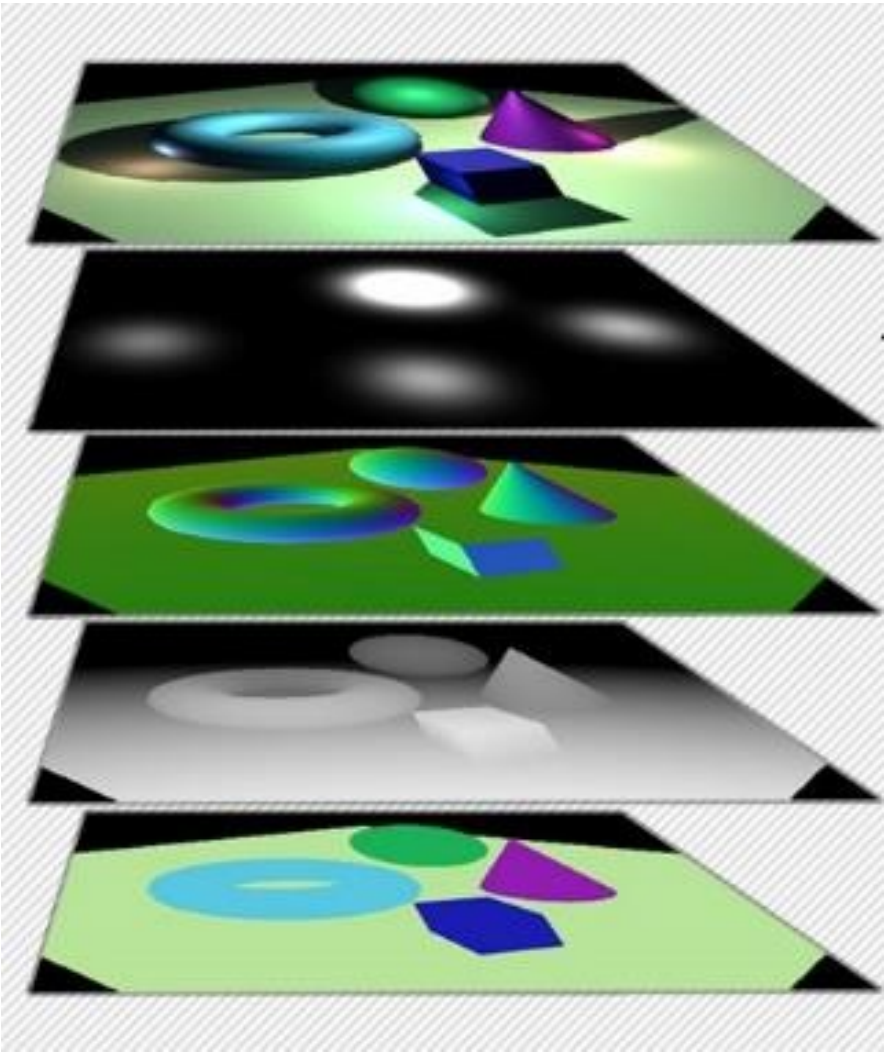
Screen-space shading technique

Avoid over-shading of fragments due to later occlusion

First pass gathers data relevant to shading into G-Buffer

- Color (albedo)
- Normal
- Depth

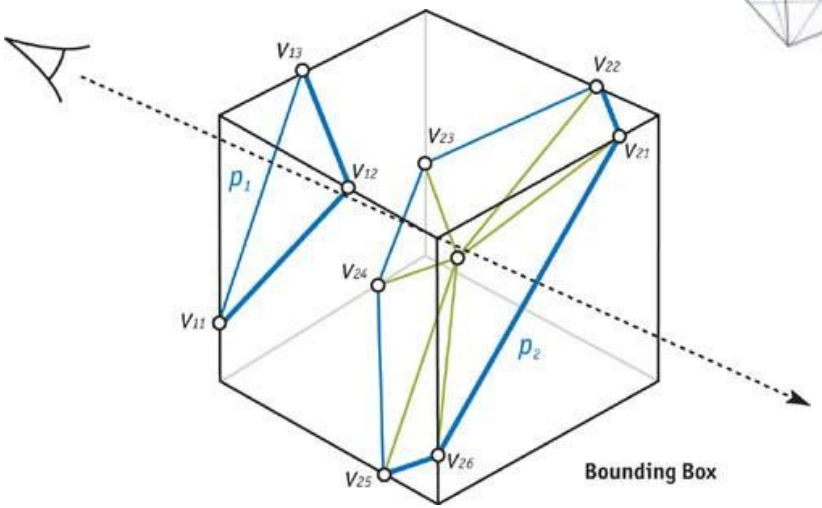
Second pass performs actual shading per pixel (*i.e.* only for visible fragments)

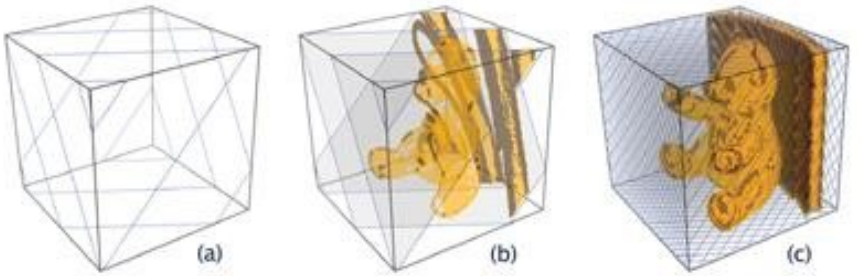


Texture-based volume rendering using view-aligned slicing of volume data

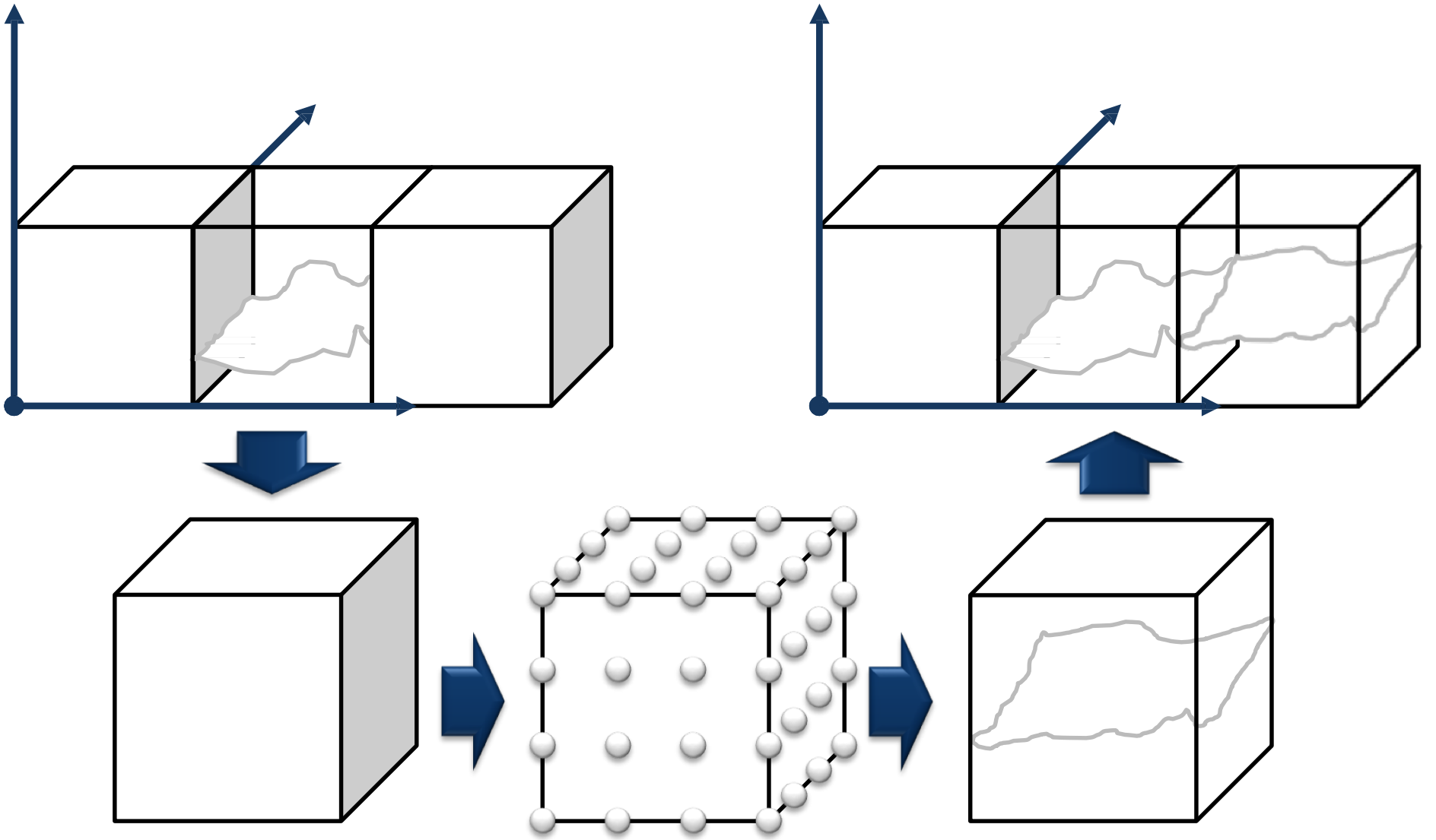
Proxy-Geometry for rasterization

Draw in back-to-front sorted order with alpha blending enabled



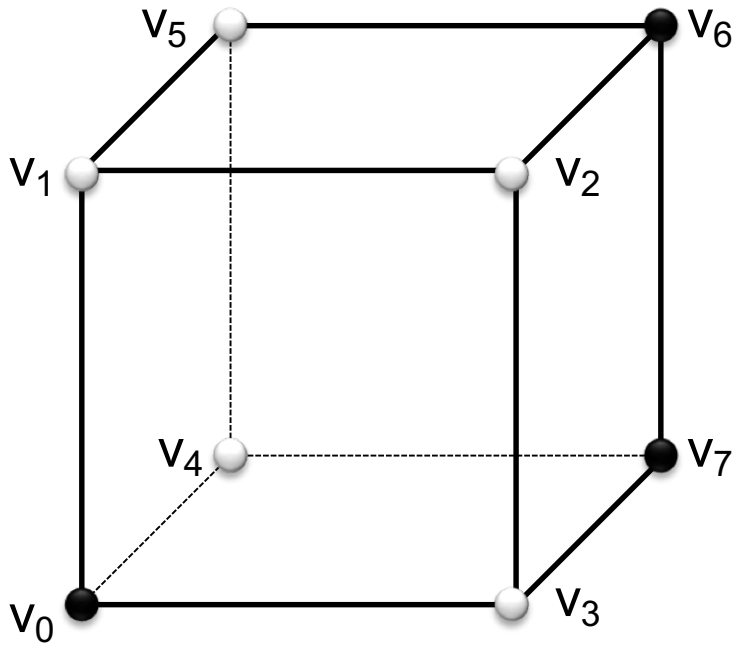


Isosurfaces from Volume Data



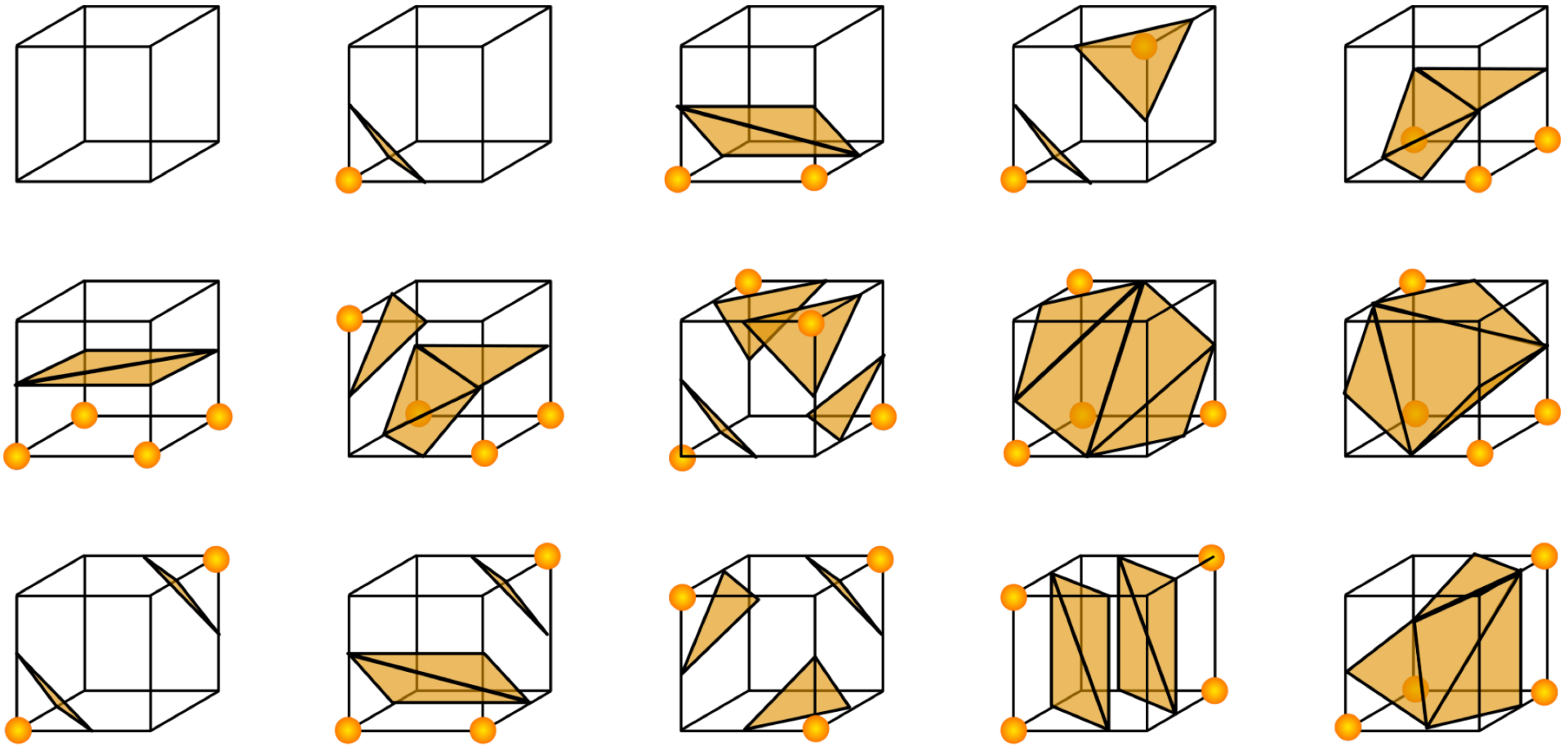
Isosurfaces

- Originated by *William E. Lorensen* and *Harvey E. Cline* in 1987
- caseBit[i] = density(v_i) > 0

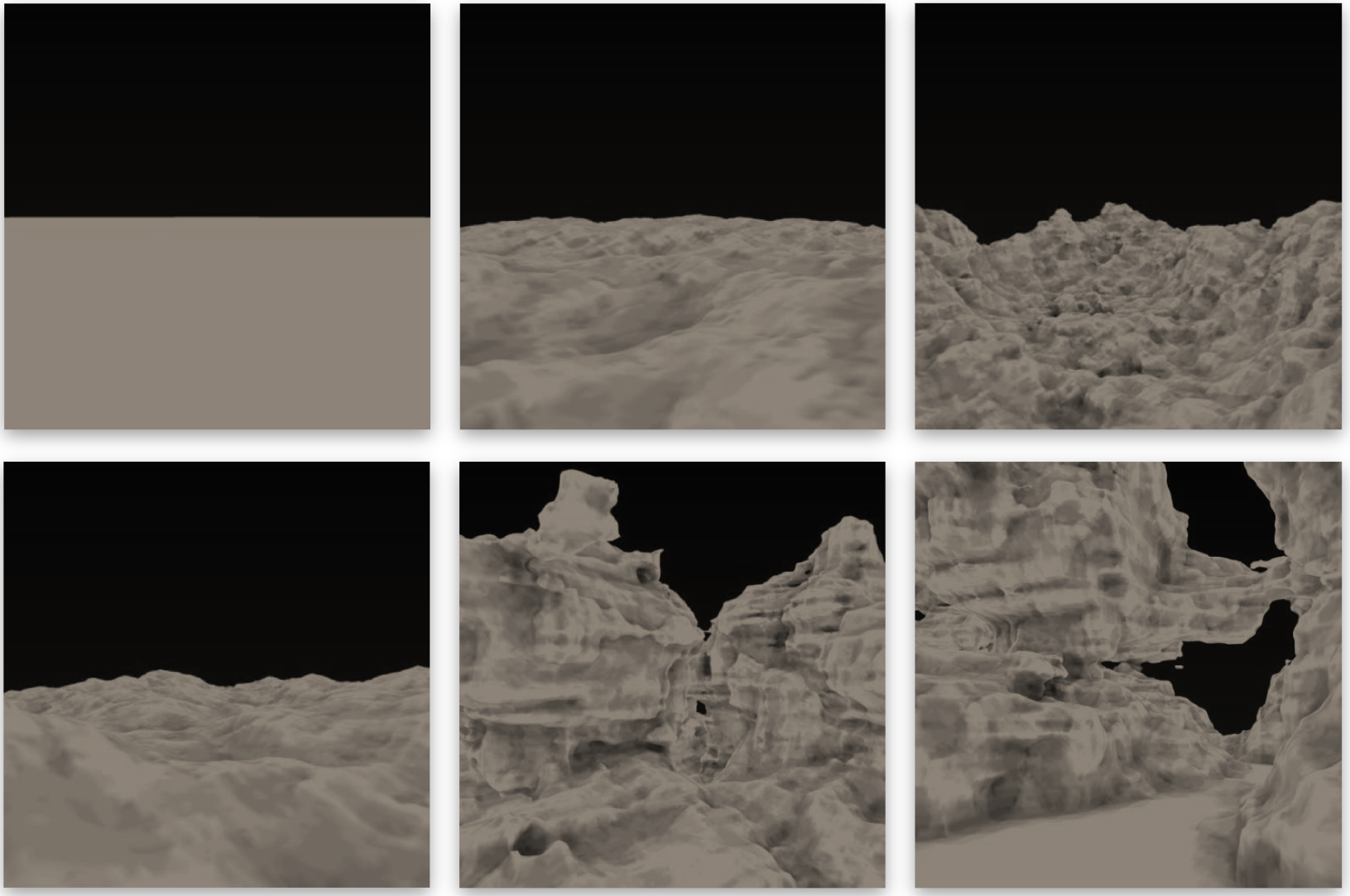


case = $v_7|v_6|v_5|v_4|v_3|v_2|v_1|v_0$
= 11000001
= 0xC1 = 193

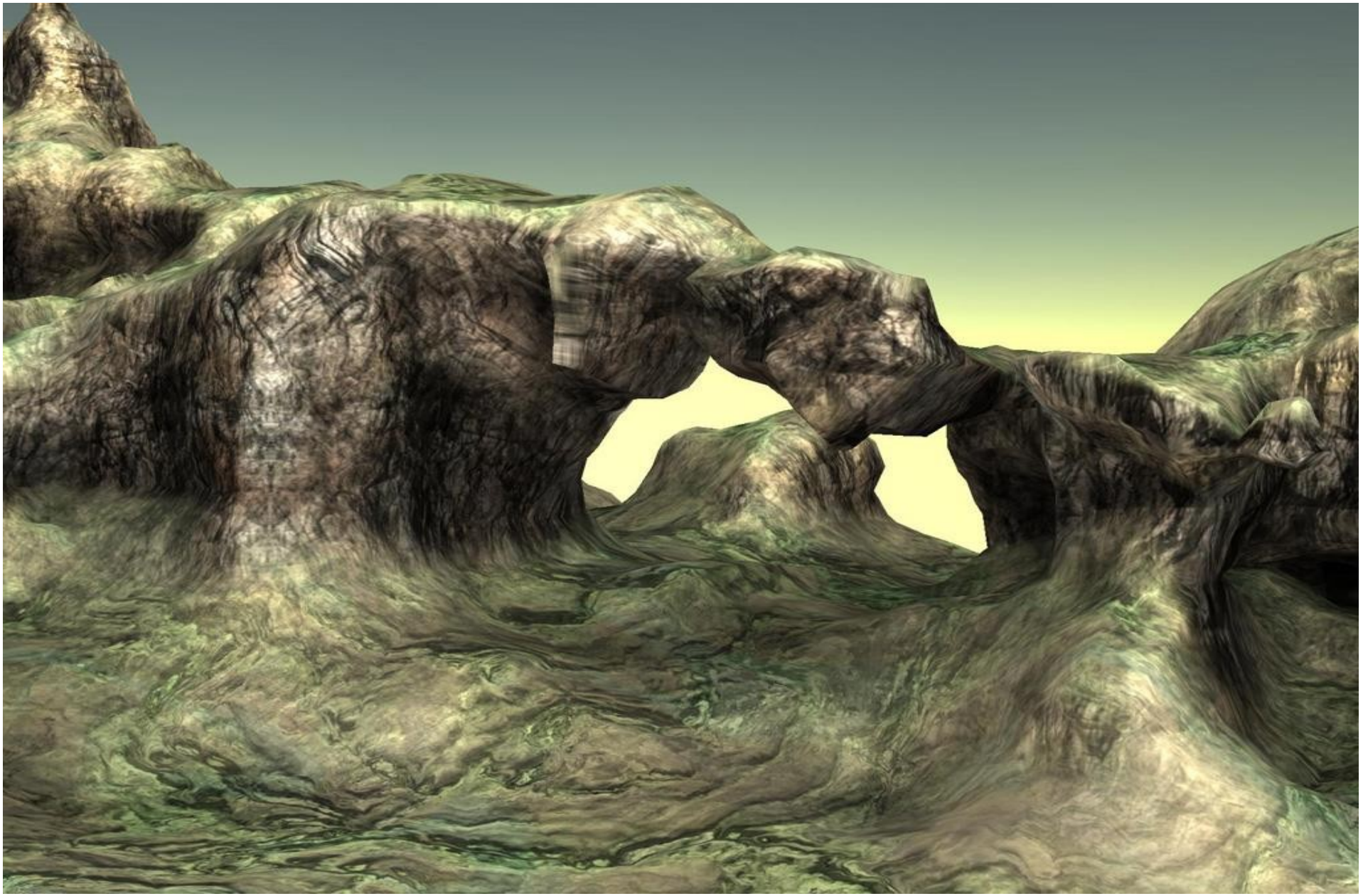
15 fundamental cases for Marching Cubes



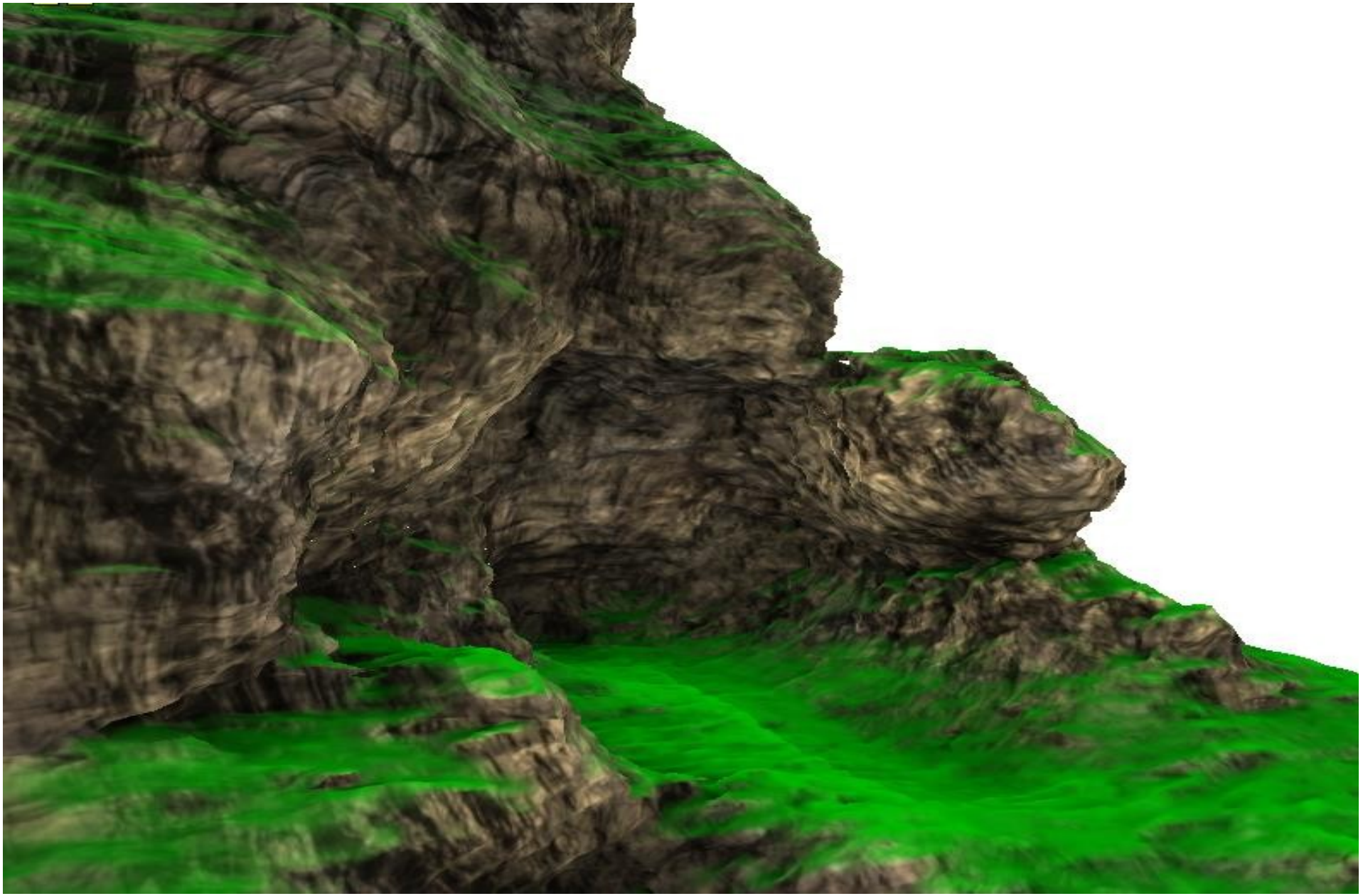
Isosurfaces from Noise



Procedural Terrain Generation



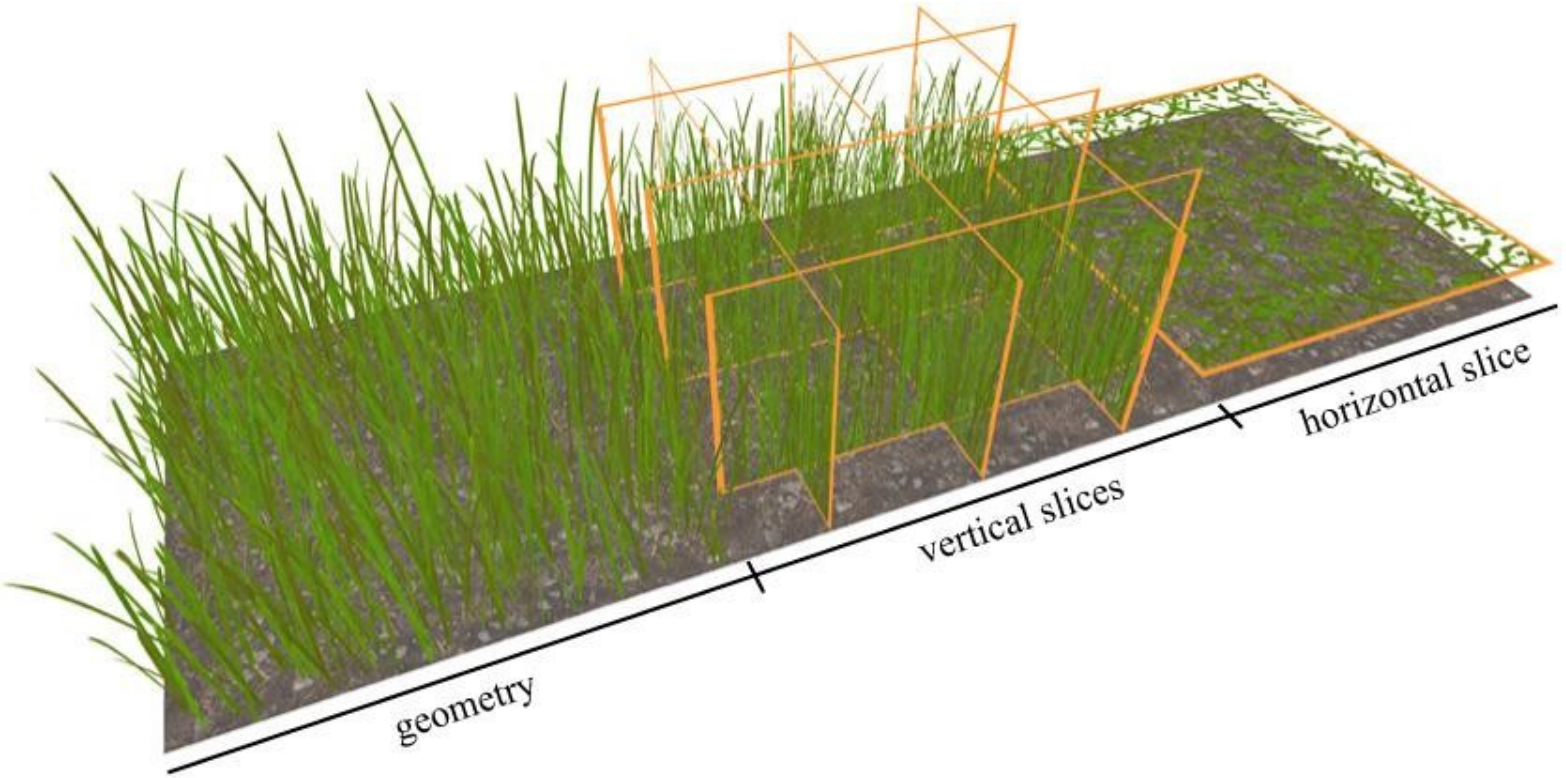
Procedural Terrain Generation



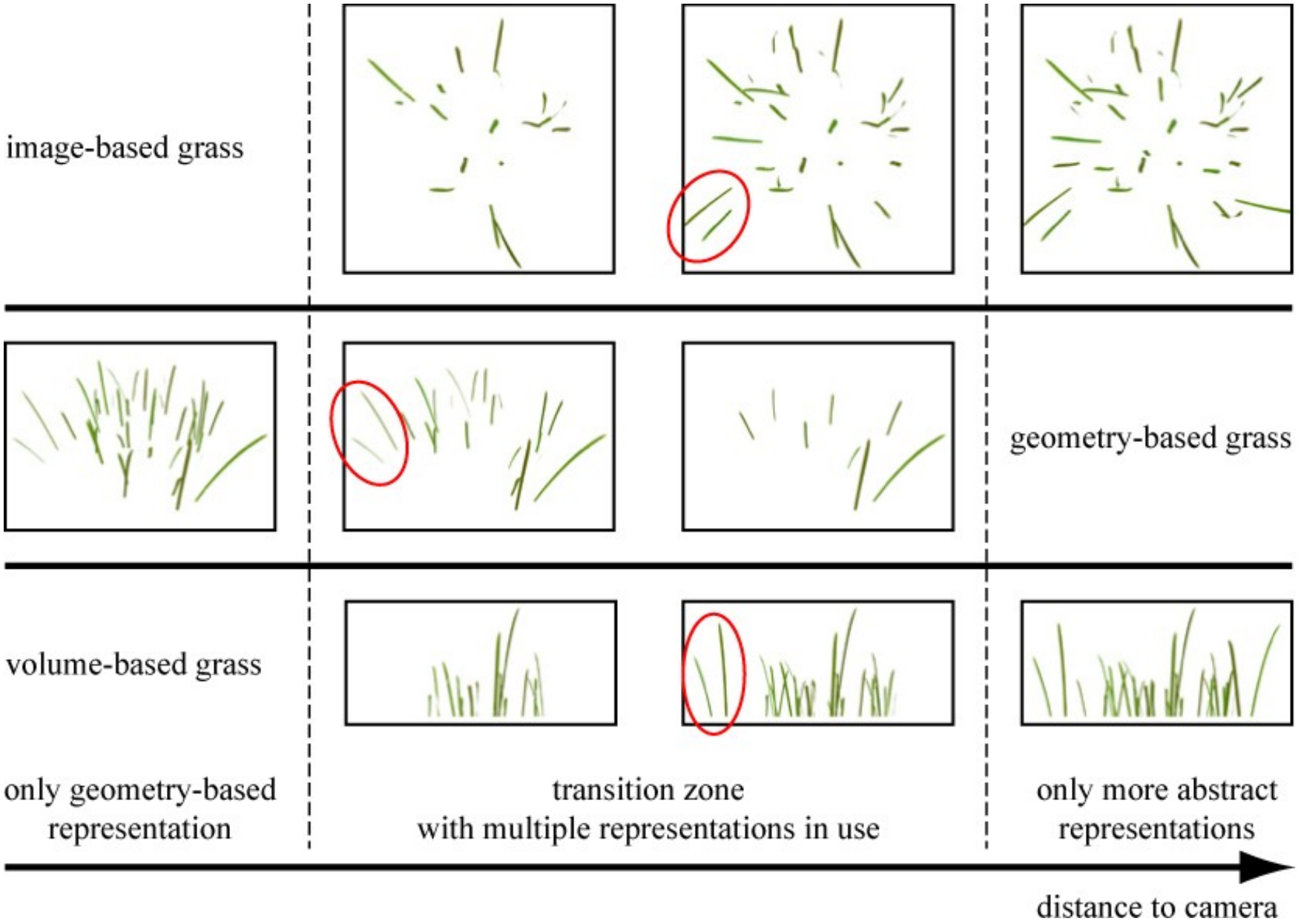
Decorating large-scale Terrain

Decorating large-scale Terrain

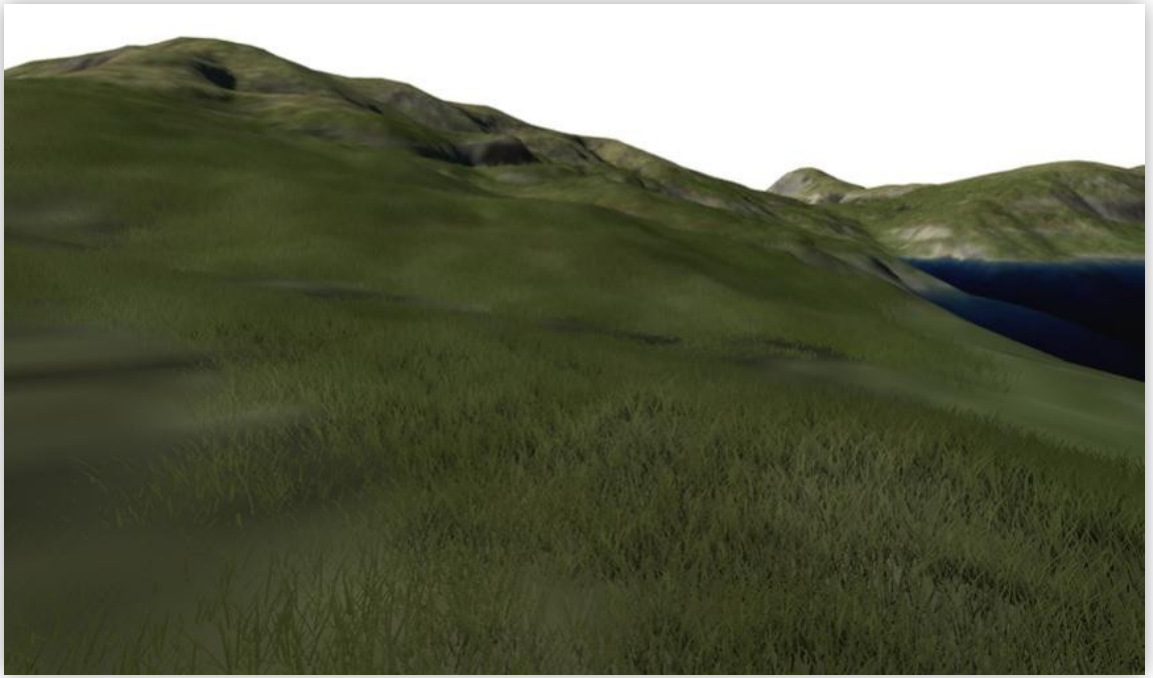
- Goal: cover large terrain surfaces with grass in real-time
- Thousands of millions of grass blades
- Multiple instances of a single grass patch – three different representations
- Arranged into the cells of a uniform grid

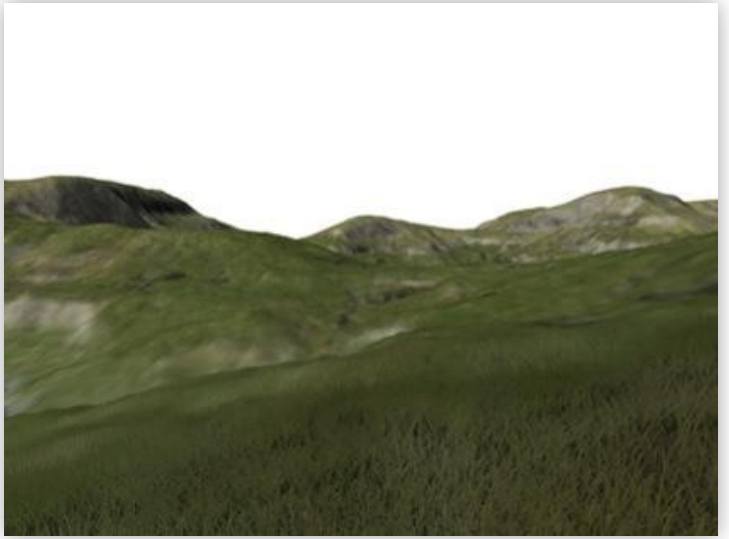


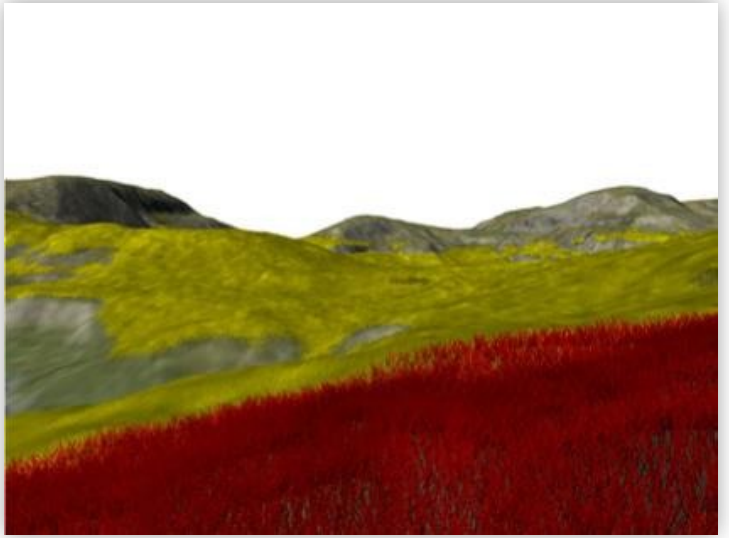
Level of Detail



Decorating large-scale Terrain



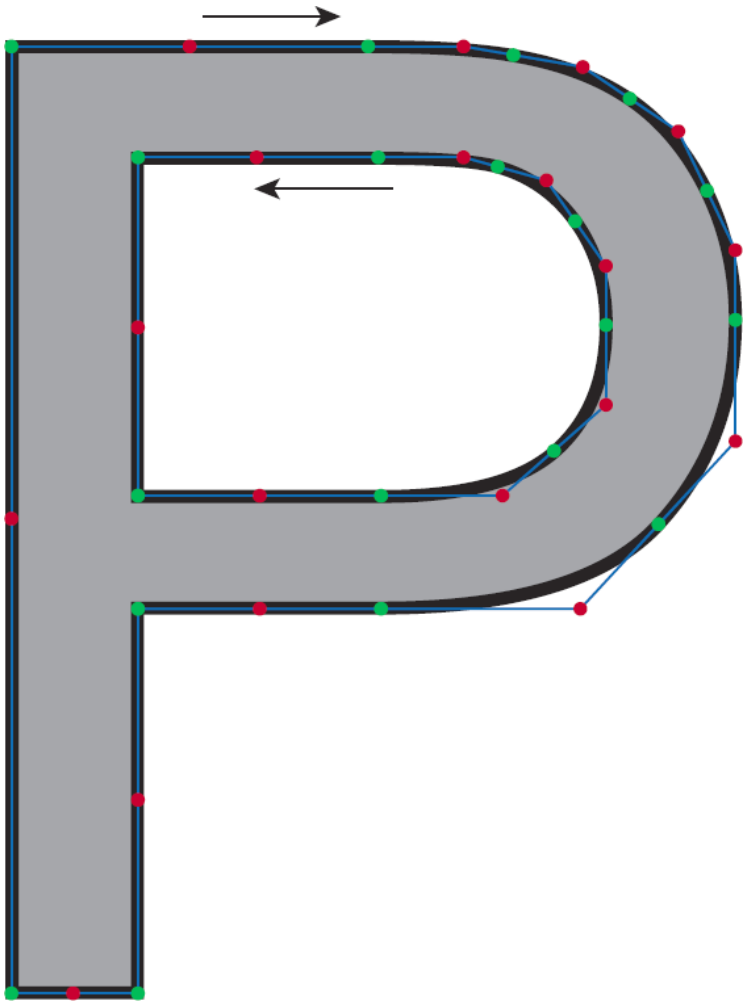




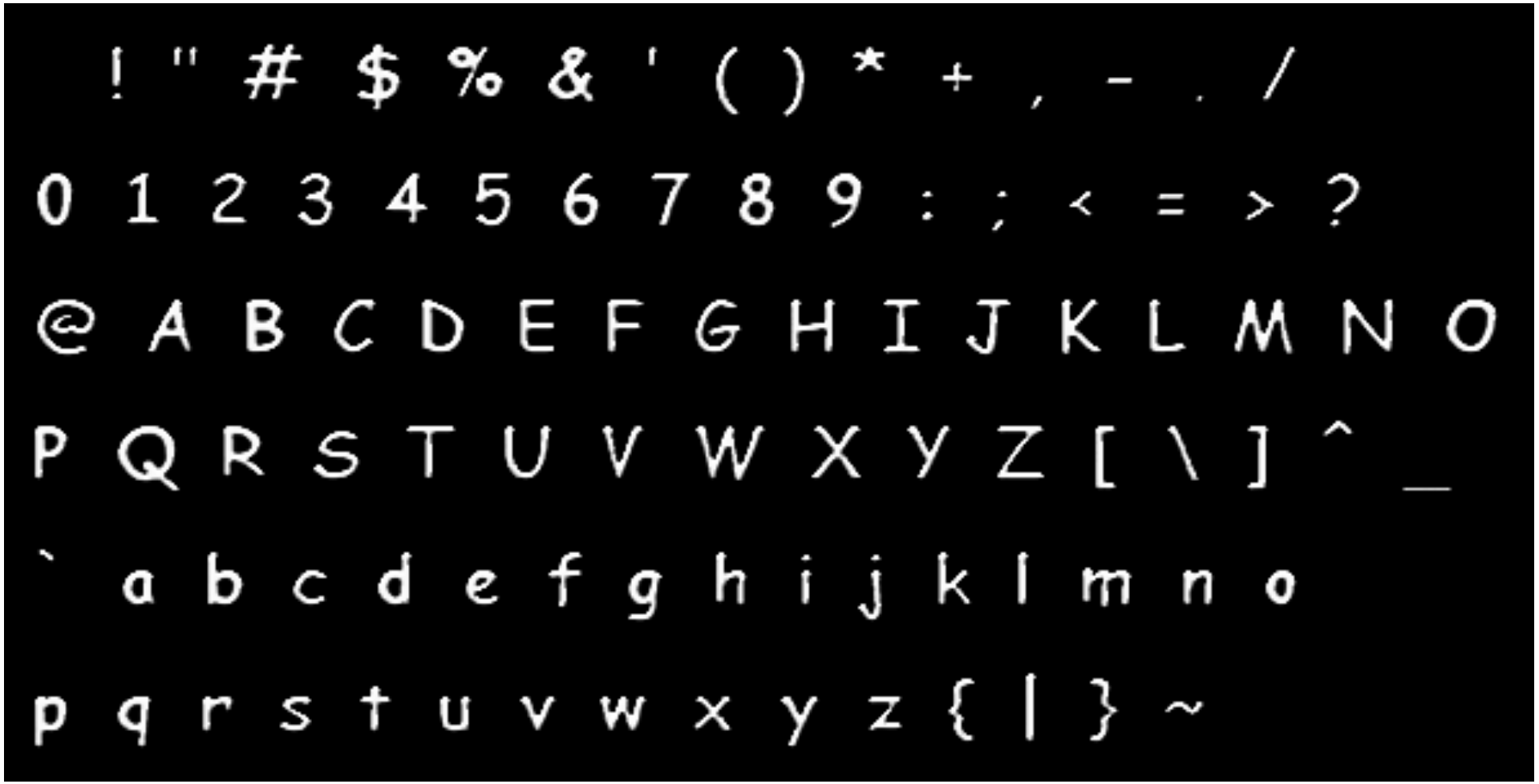
Rendering Text and Decals

Font Rendering

Glyph consists of splines as outline



Bitmap Fonts



Magnification using semi-transparent textures

64x64 texture,
alpha-blended

64x64 texture,
alpha tested

Magnification using distance fields

- Instead of storing the fonts as a bitmap, we store the distance from each texel to the edge of the character. We then use a shader to compute sharp edges

High resolution input

64x64 Distance field

Magnification using distance fields

64x64 texture,
alpha-blended

64x64 texture,
alpha tested

64x64 texture,
distance field