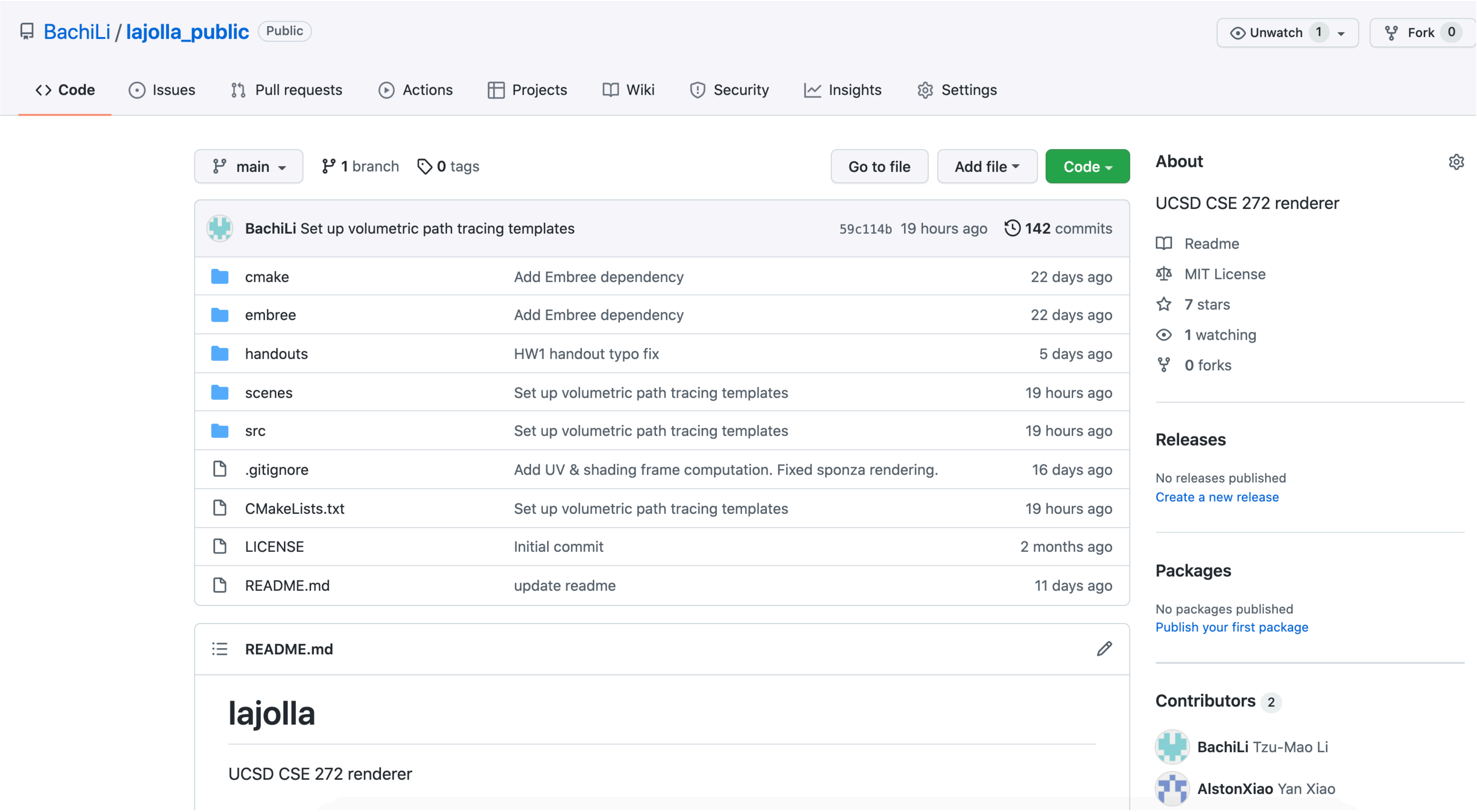
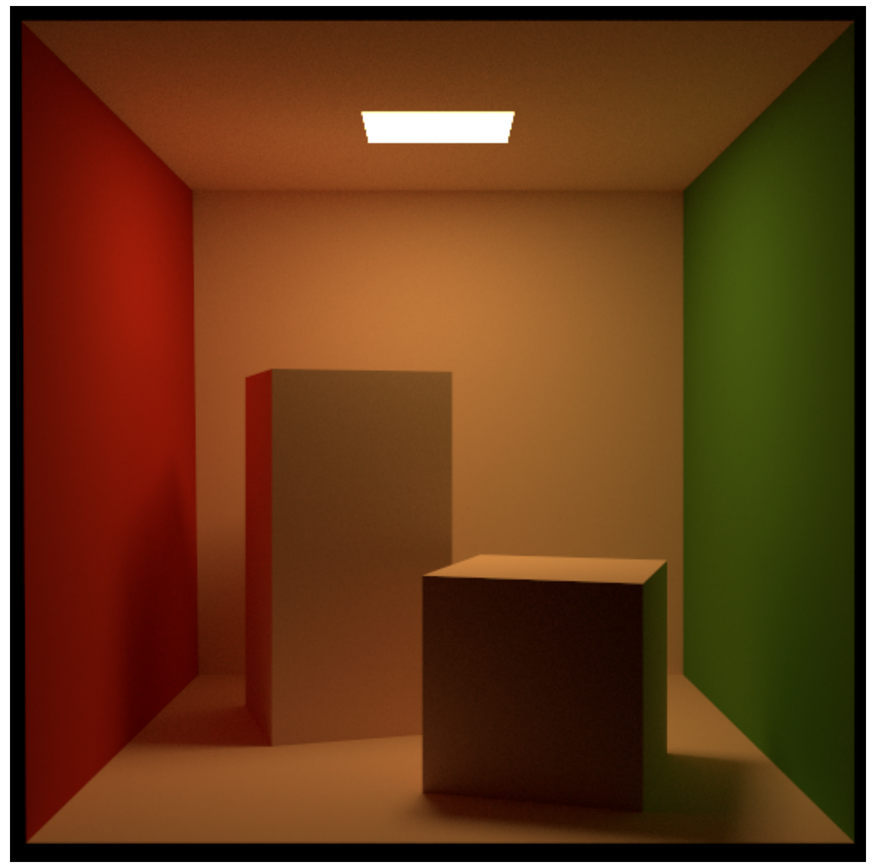
The lajolla renderer

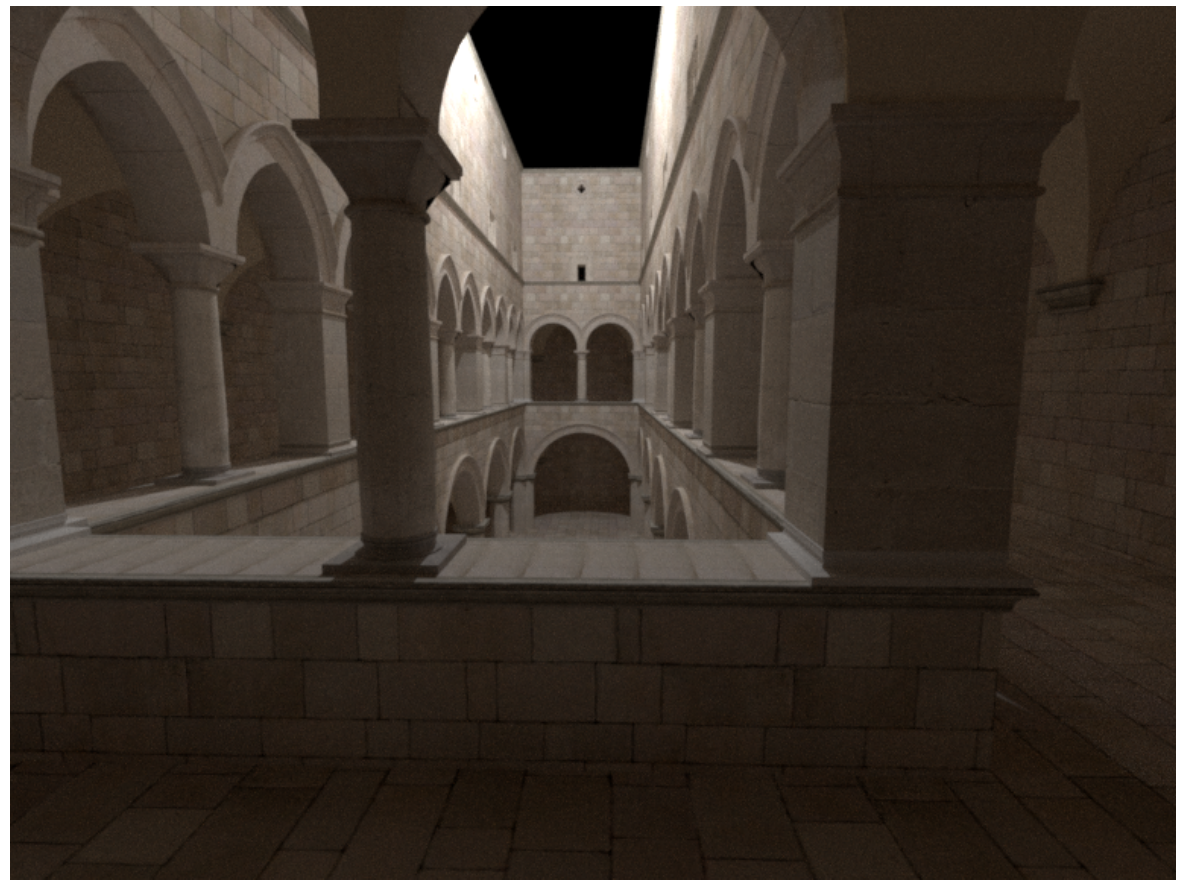
UCSD CSE 272
Advanced Image Synthesis

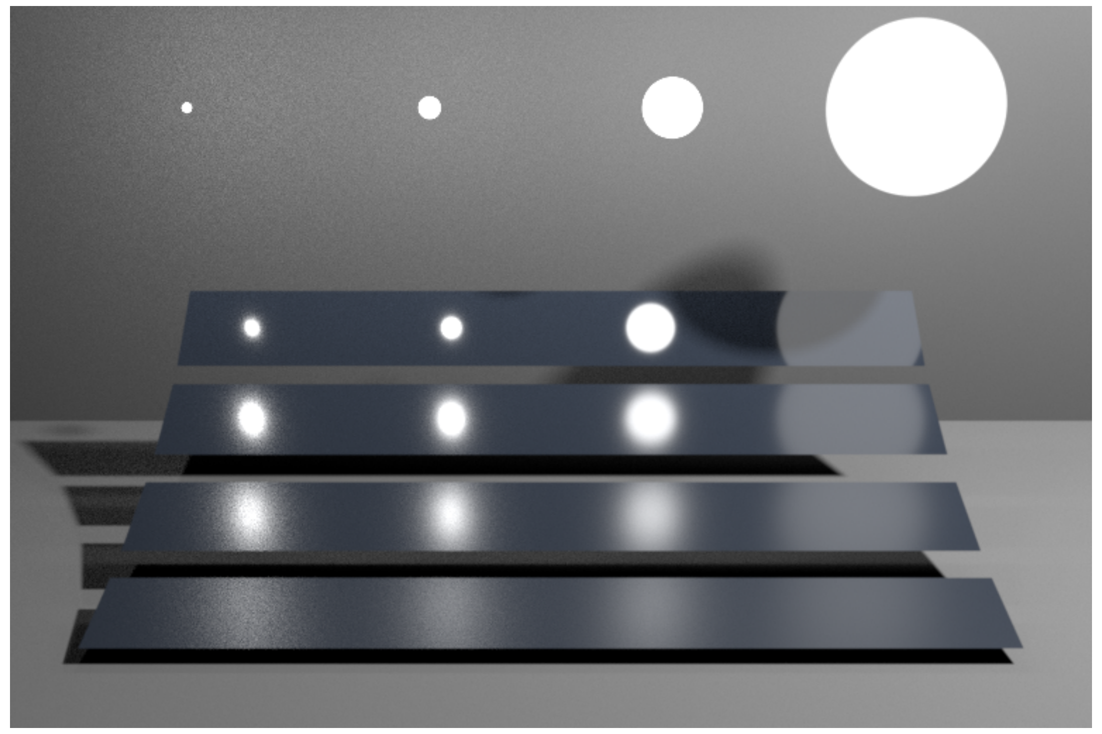
Tzu-Mao Li

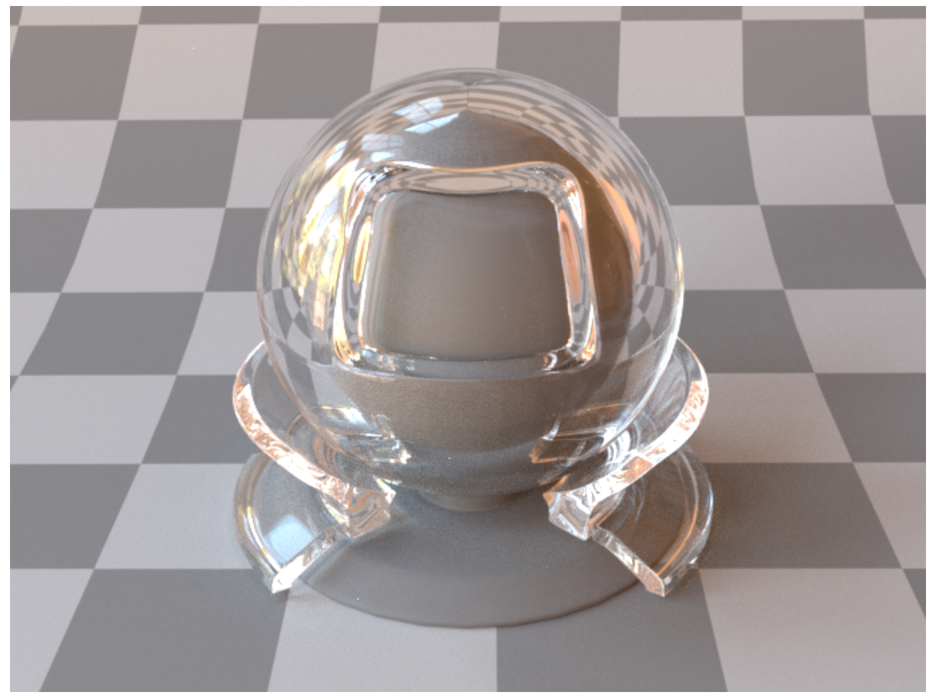
From smallpt to lajolla



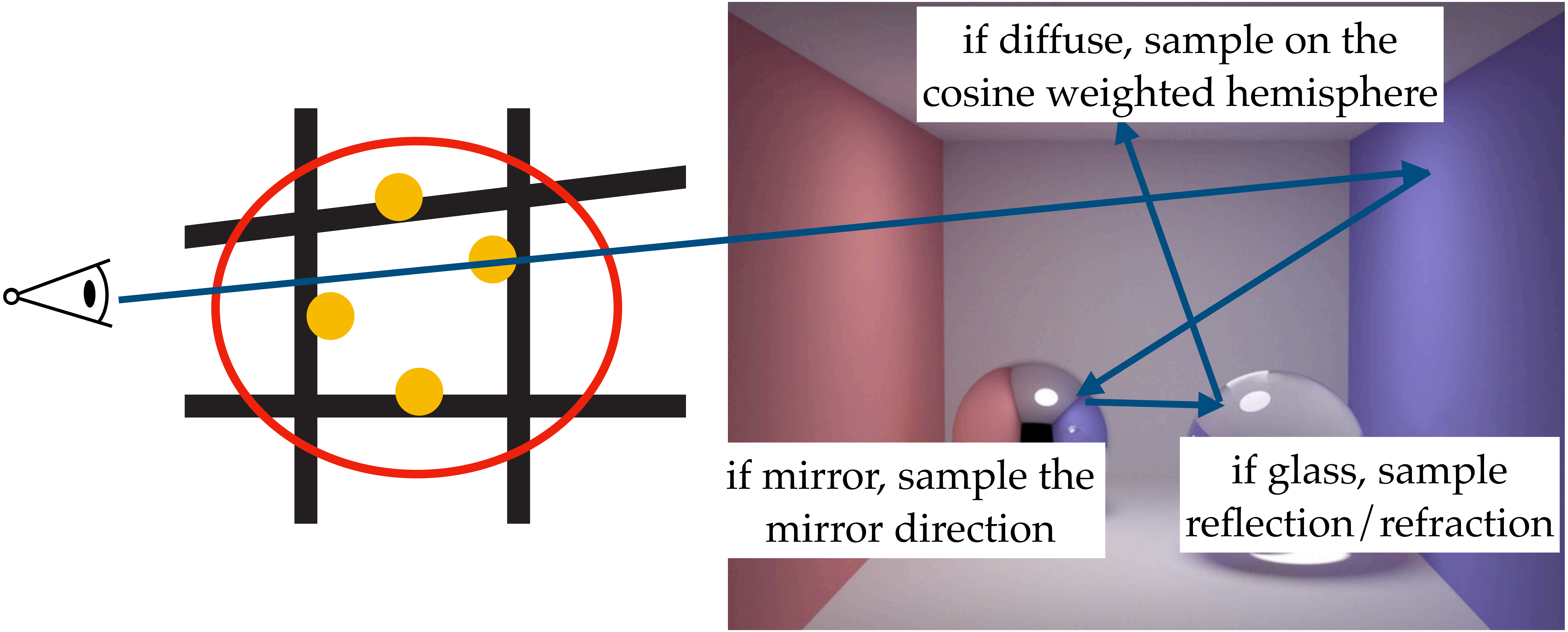






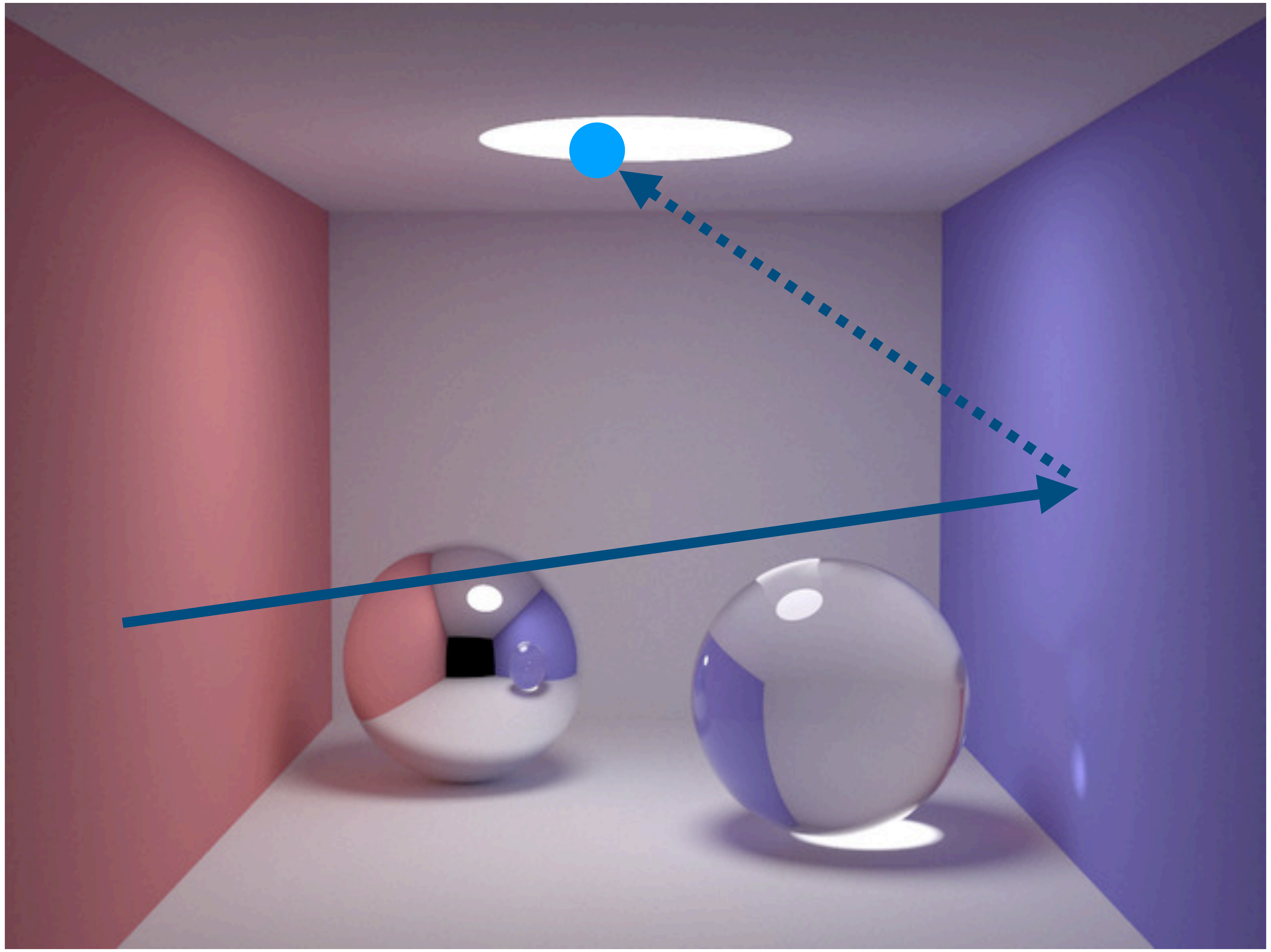


Smallpt: hope to hit light with directional sampling



Next event estimation

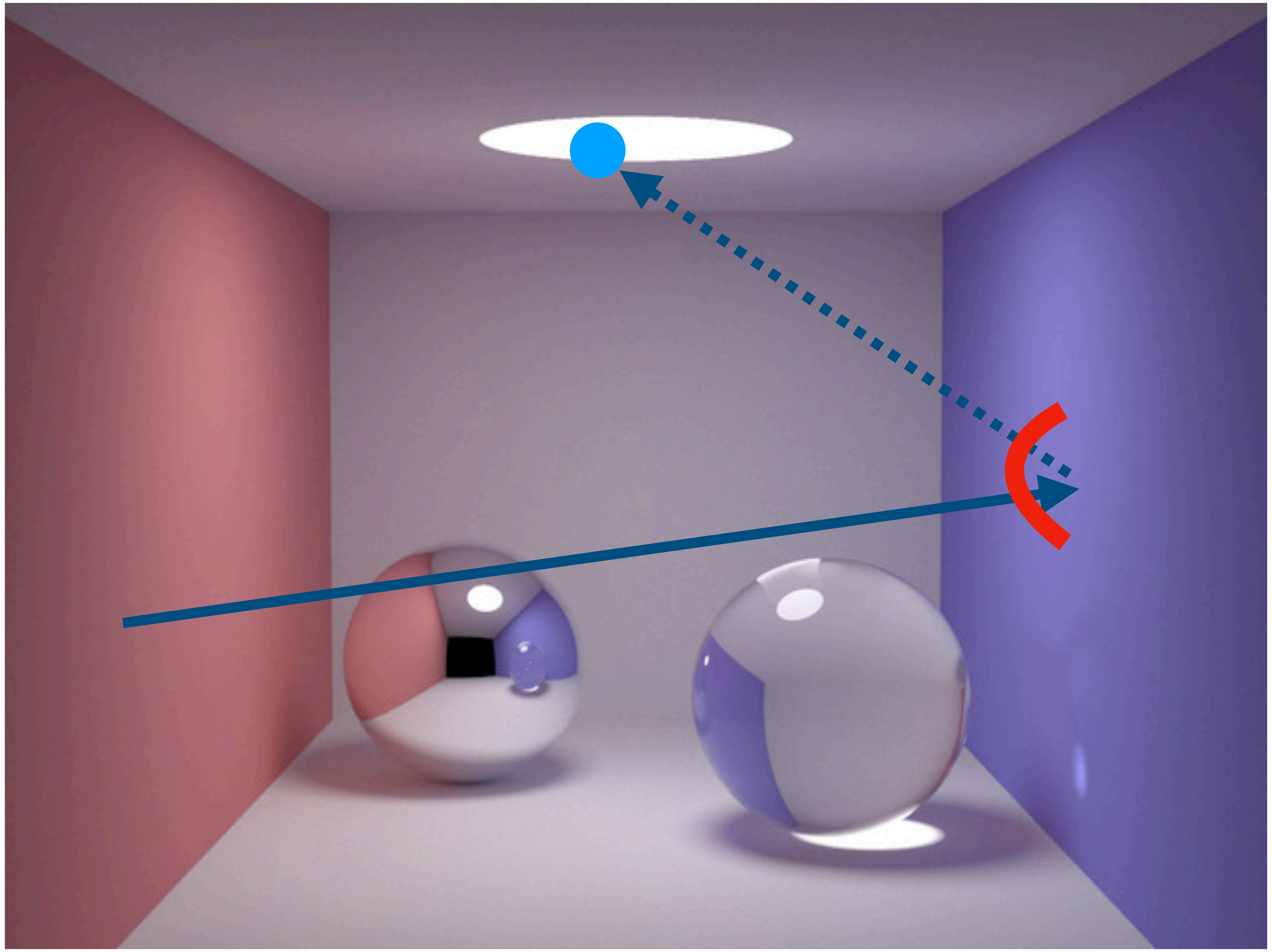
- in addition to cosine-weighted hemisphere sampling, also sample a point on light



Next event estimation is also a change of variable

focus on the rays that hit the light source

$$\iint L_e'(\omega')\,|\,\omega'\cdot n\,|\,\mathrm{d}\omega'$$



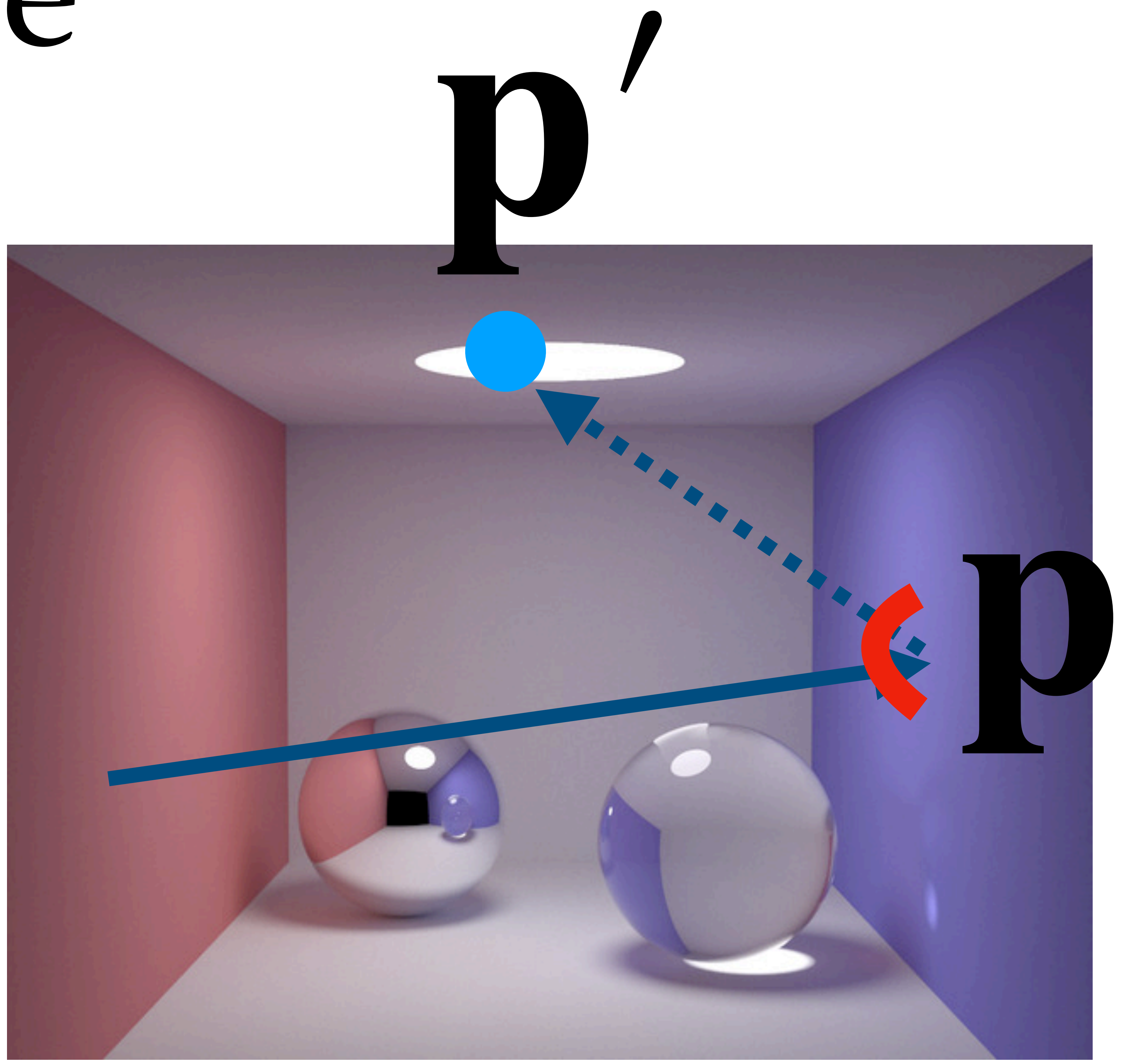
Next event estimation is also a change of variable

focus on the rays that hit the light source

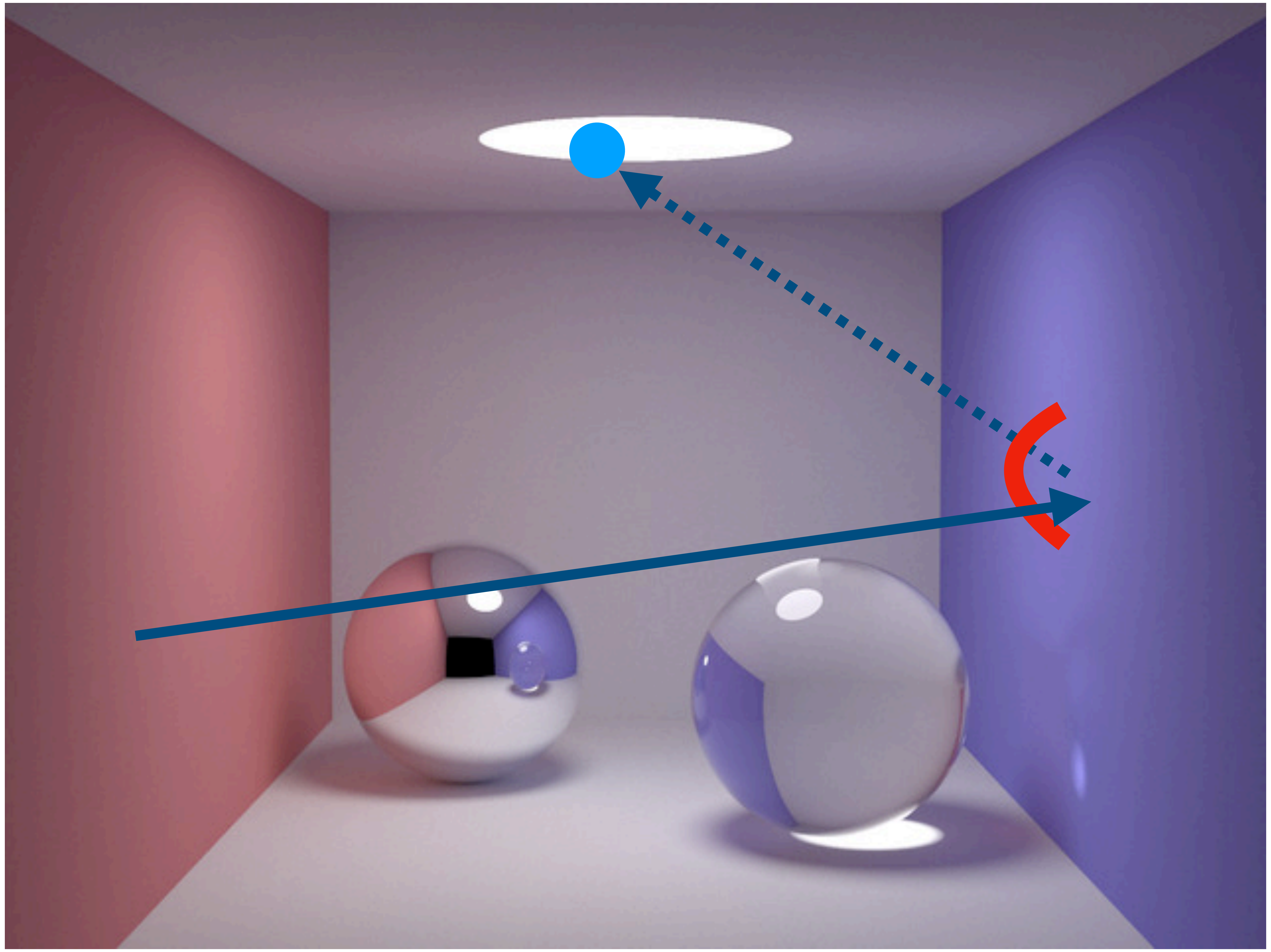
$$\iint L_e'(\omega')\,|\,\omega' \cdot n\,|\,\mathrm{d}\omega'$$

$$\iint L_e'(\omega'(\mathbf{p}'))\,|\,\omega'(\mathbf{p}') \cdot n_{\mathbf{p}}\,|\,\frac{|\,\omega'(\mathbf{p}') \cdot n_{\mathbf{p}'}\,|}{\|\mathbf{p} - \mathbf{p}'\|^2}\,\mathrm{visible}(\mathbf{p}, \mathbf{p}')\,\mathrm{d}\mathbf{p}'$$

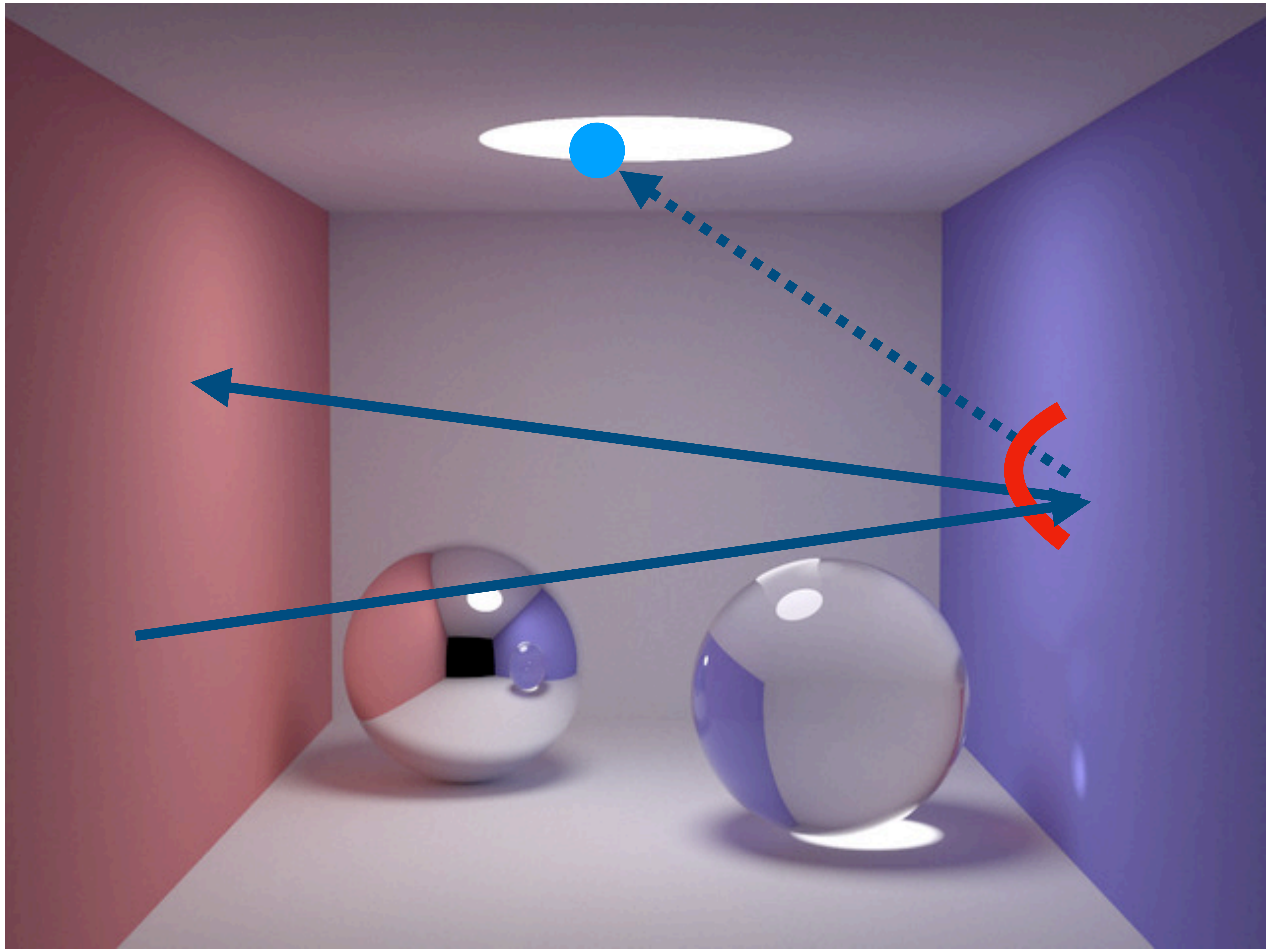
the Jacobian (often called "geometry term")



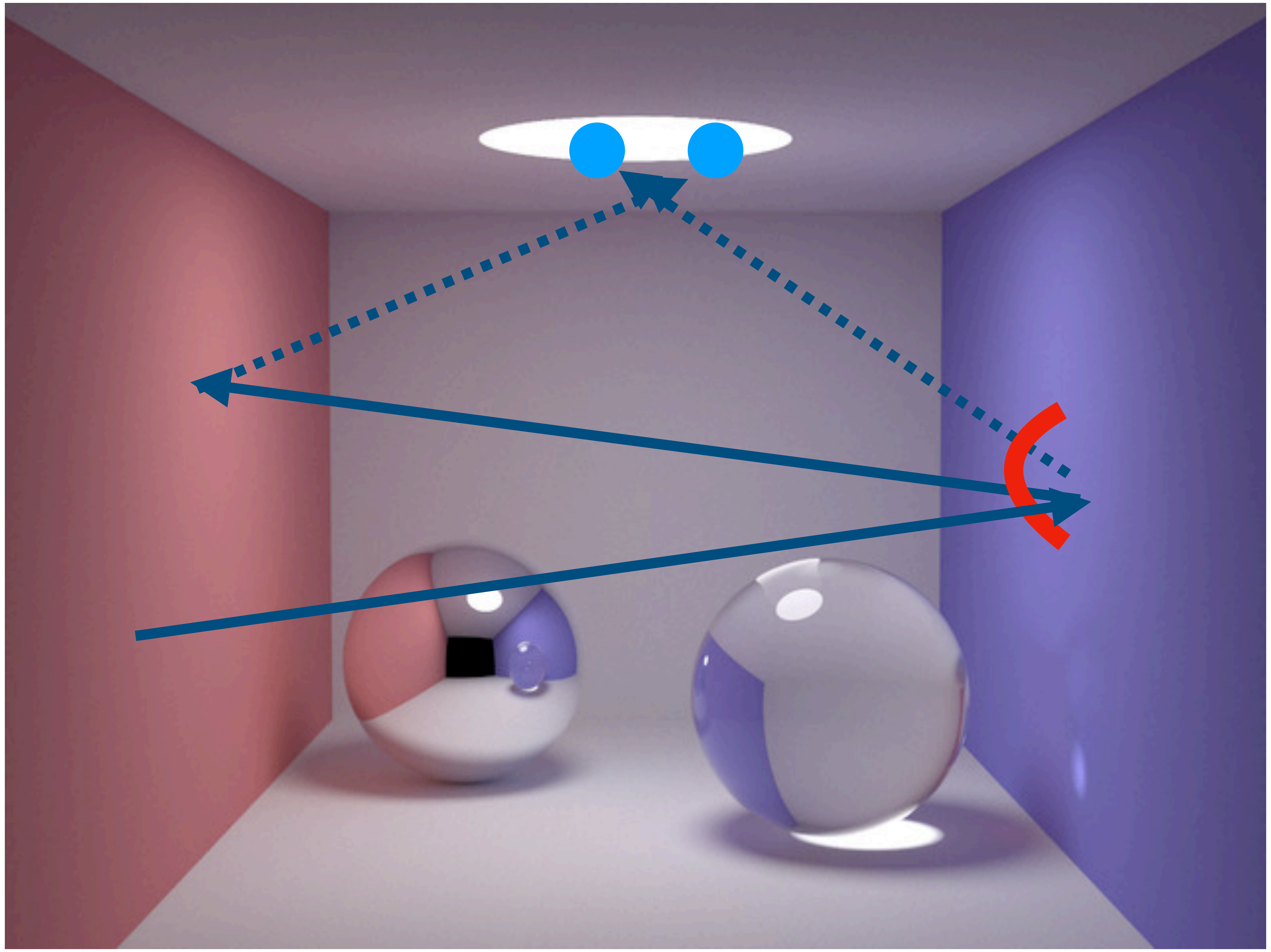
Handling multiple bounces



Handling multiple bounces

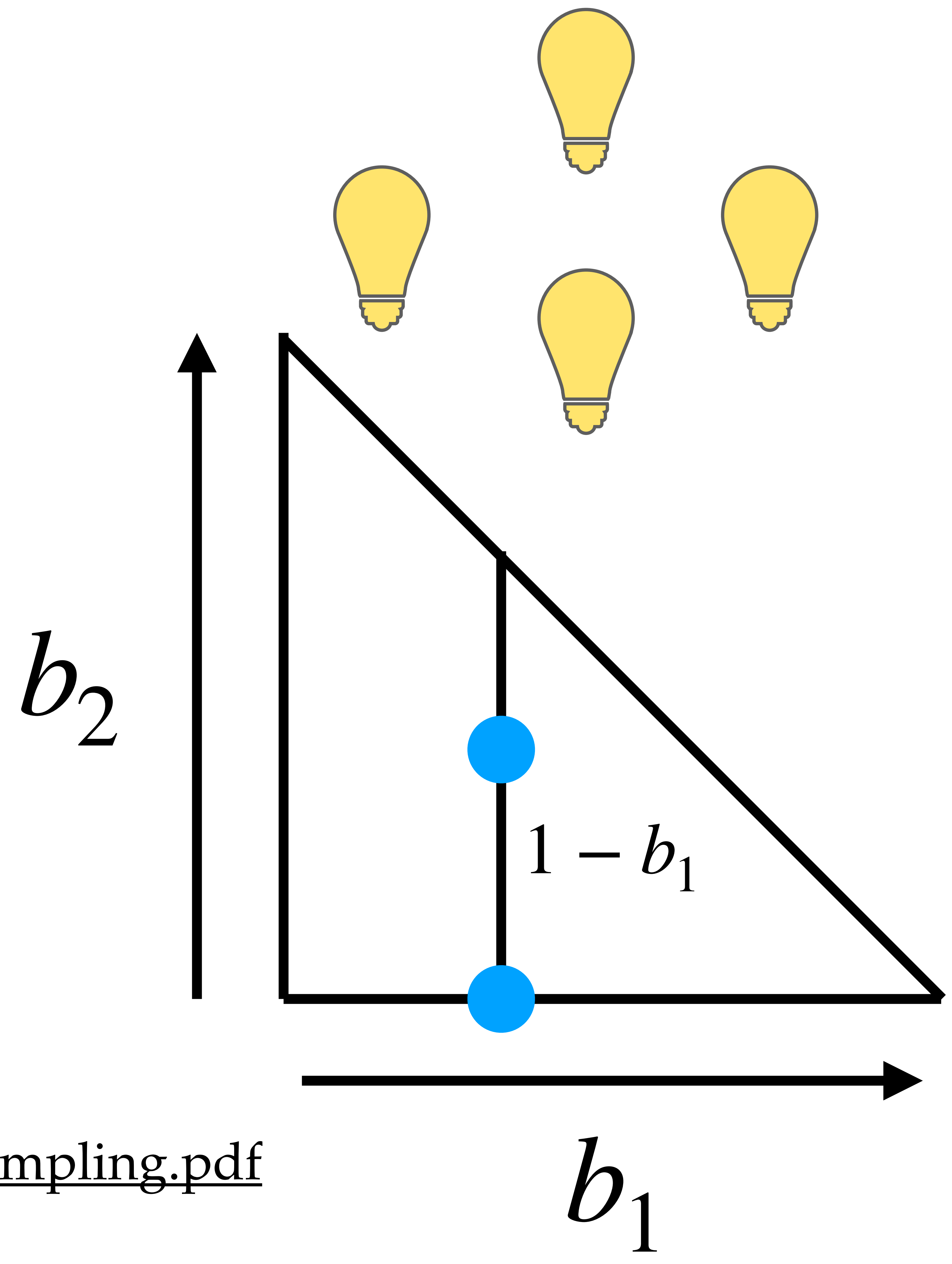


Handling multiple bounces



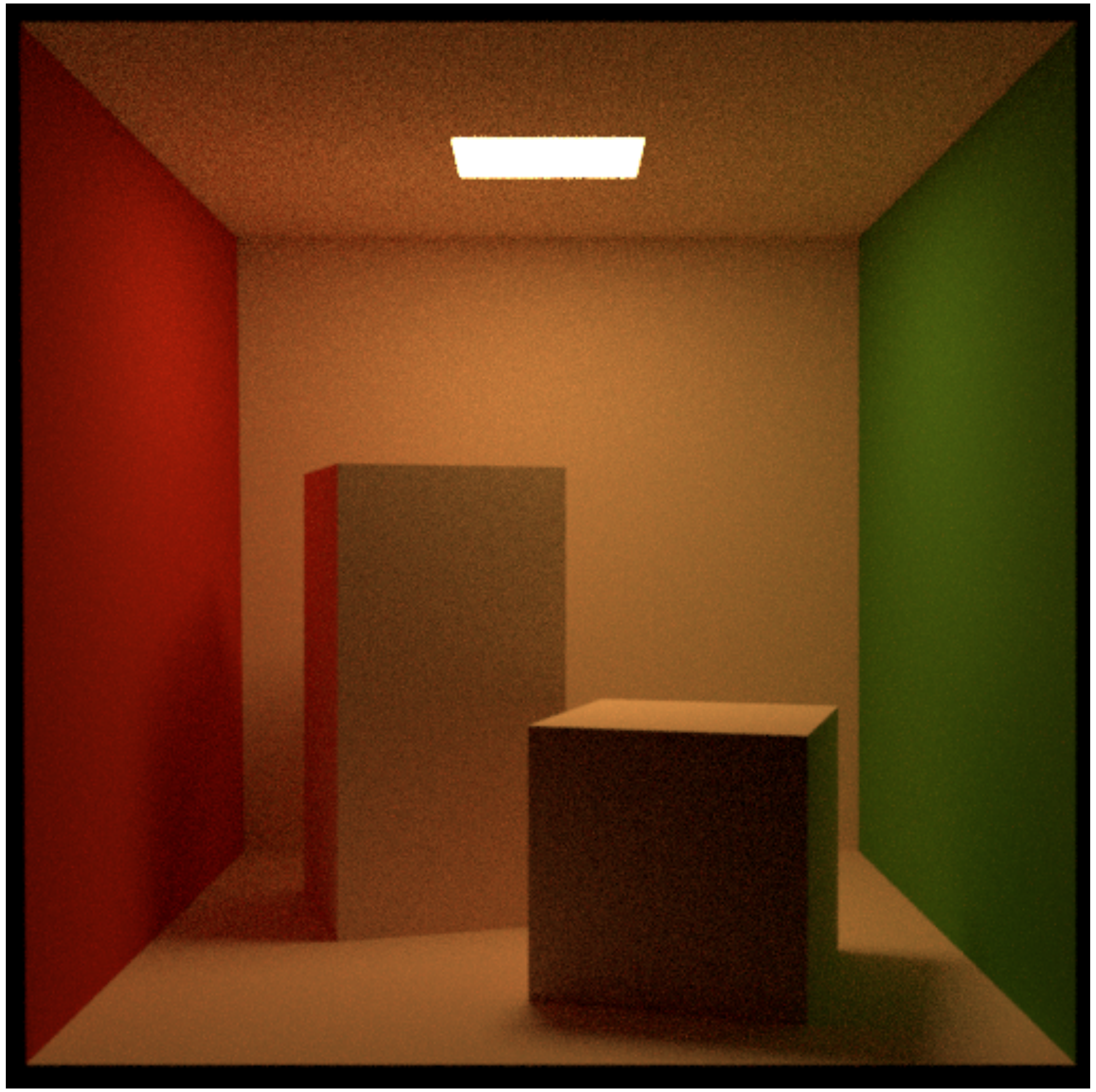
Triangle-mesh light sampling:

1. pick a light based on their intensities
2. pick a triangle based on its area
3. pick a point on the triangle

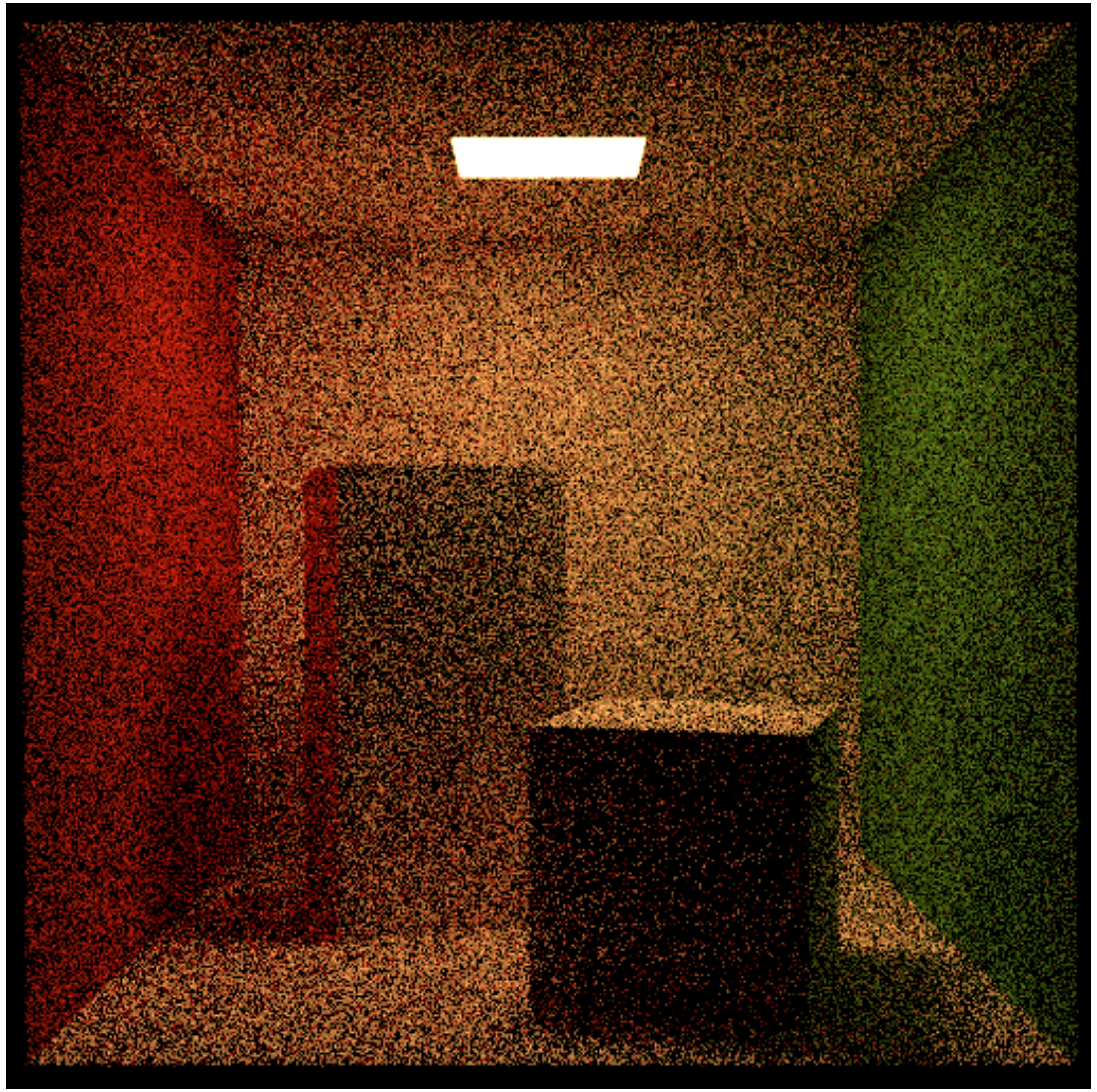


see https://cseweb.ucsd.edu/~tzli/cse272/wi2023/lectures/triangle_sampling.pdf
for notes on triangle sampling

Next event estimation is good at small lights

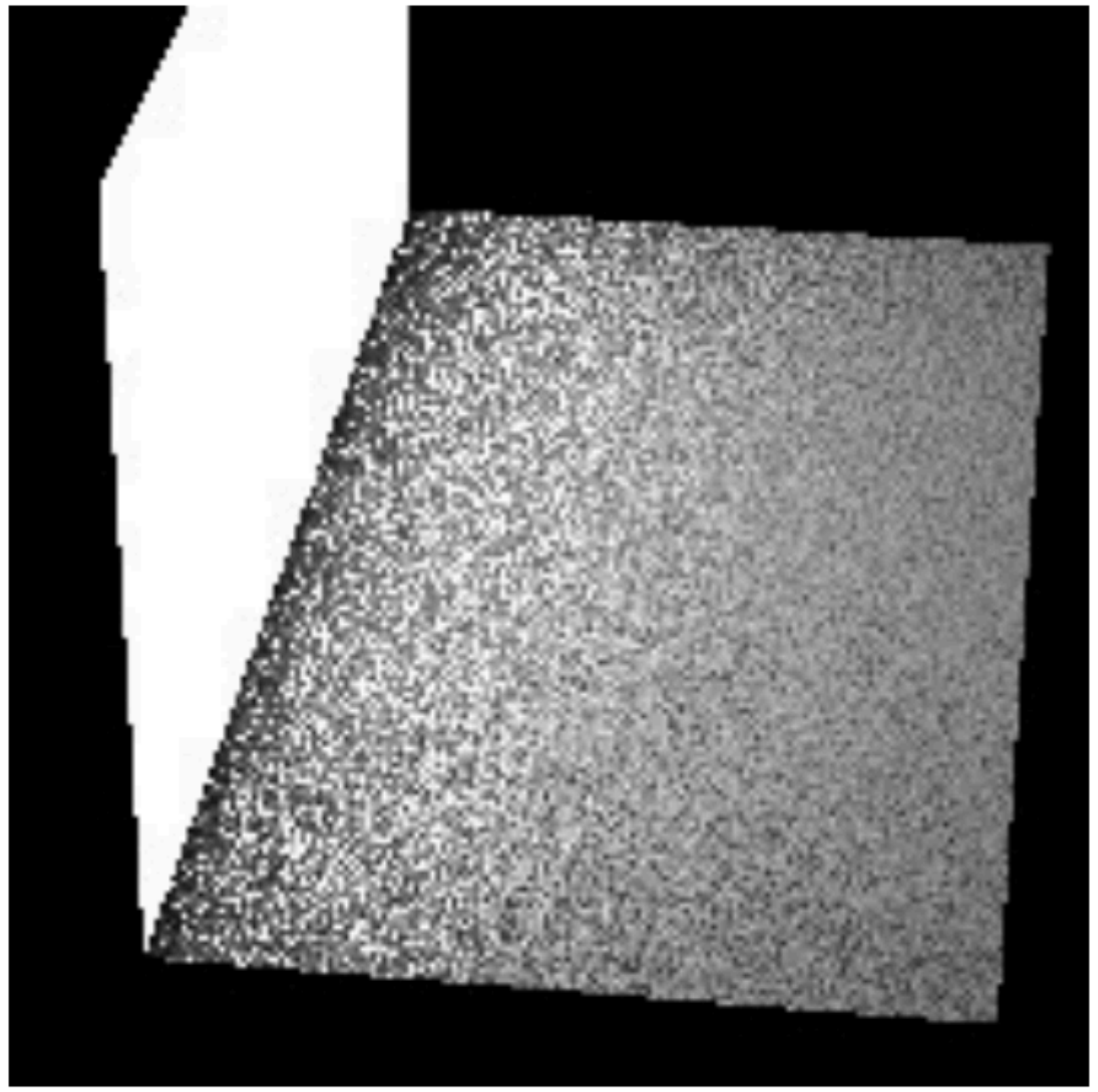


next event estimation
(64 samples per pixel)

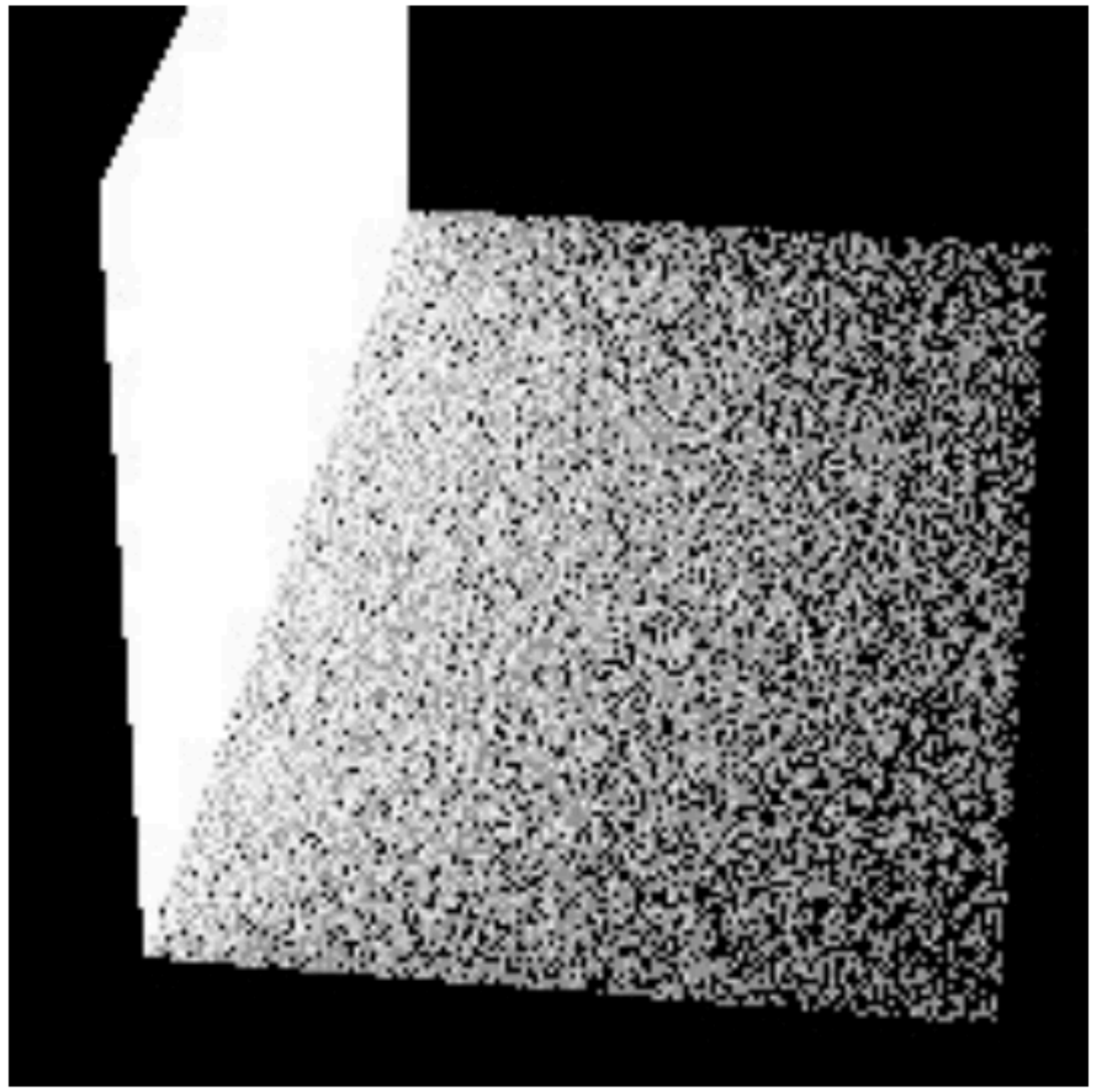


cosine weighted hemisphere
(64 samples per pixel)

When the point is close to the light, cosine-weighted hemisphere sampling is better



next event estimation



cosine-weighted hemisphere sampling

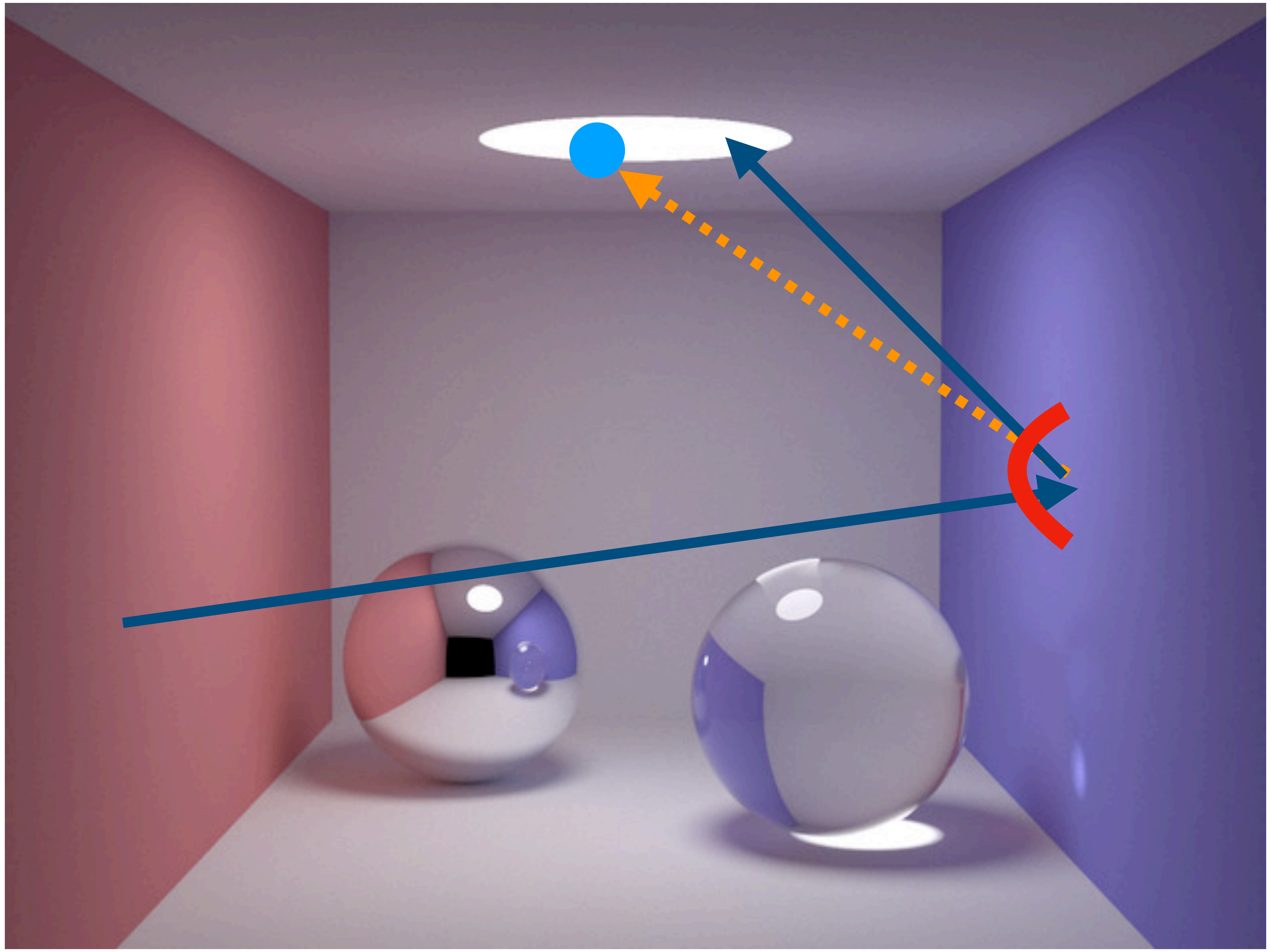
"Robust Monte Carlo Methods for Light Transport Simulation", Eric Veach

Multiple importance sampling: combining next event estimation and hemisphere sampling

$$\iint L'(\omega')\,|\,\omega' \cdot n\,|\,\mathrm{d}\omega'$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\frac{L'(\omega_i')\,|\,\omega_i' \cdot n\,|}{p_{\text{nee}}(\omega_i')} \qquad ?$$

$$\approx \frac{1}{N}\sum_{j=1}^{N}\frac{L'(\omega_j')\,|\,\omega_j' \cdot n\,|}{p_{\text{hemi}}(\omega_j')} \qquad ?$$



Multiple importance sampling: combining next event estimation and hemisphere sampling

- idea: assign higher weight when p is high

$$\frac{1}{N}\left(\sum_{i=1}^{N} w_i^{\mathrm{n}} \frac{L'(\omega_i')\,|\,\omega_i' \cdot n\,|}{p_{\mathrm{nee}}(\omega_i')} + \sum_{j=1}^{N} w_j^{\mathrm{h}} \frac{L'(\omega_j')\,|\,\omega_j' \cdot n\,|}{p_{\mathrm{hemi}}(\omega_j')}\right)$$

Multiple importance sampling: combining next event estimation and hemisphere sampling

- idea: assign higher weight when p is high

$$\frac{1}{N}\left(\sum_{i=1}^{N} w_i^{\text{n}} \frac{L'(\omega_i')\,|\,\omega_i' \cdot n\,|}{p_{\text{nee}}(\omega_i')} + \sum_{j=1}^{N} w_j^{\text{h}} \frac{L'(\omega_j')\,|\,\omega_j' \cdot n\,|}{p_{\text{hemi}}(\omega_j')}\right)$$

$$w_i^n = \frac{p_{\text{nee}}(\omega_i)}{p_{\text{nee}}(\omega_i) + p_{\text{hemi}}(\omega_i)}$$

Multiple importance sampling: combining next event estimation and hemisphere sampling

- idea: assign higher weight when p is high

$$\frac{1}{N}\left(\sum_{i=1}^{N} w_i^{\mathrm{n}} \frac{L'(\omega_i')\,|\,\omega_i' \cdot n\,|}{p_{\mathrm{nee}}(\omega_i')} + \sum_{j=1}^{N} w_j^{\mathrm{h}} \frac{L'(\omega_j')\,|\,\omega_j' \cdot n\,|}{p_{\mathrm{hemi}}(\omega_j')}\right)$$

$$w_i^n = \frac{p_{\mathrm{nee}}(\omega_i)}{p_{\mathrm{nee}}(\omega_i) + p_{\mathrm{hemi}}(\omega_i)} \qquad w_j^h = \frac{p_{\mathrm{hemi}}(\omega_j)}{p_{\mathrm{nee}}(\omega_j) + p_{\mathrm{hemi}}(\omega_j)}$$

Multiple importance sampling: combining next event estimation and hemisphere sampling

- idea: assign higher weight when p is high

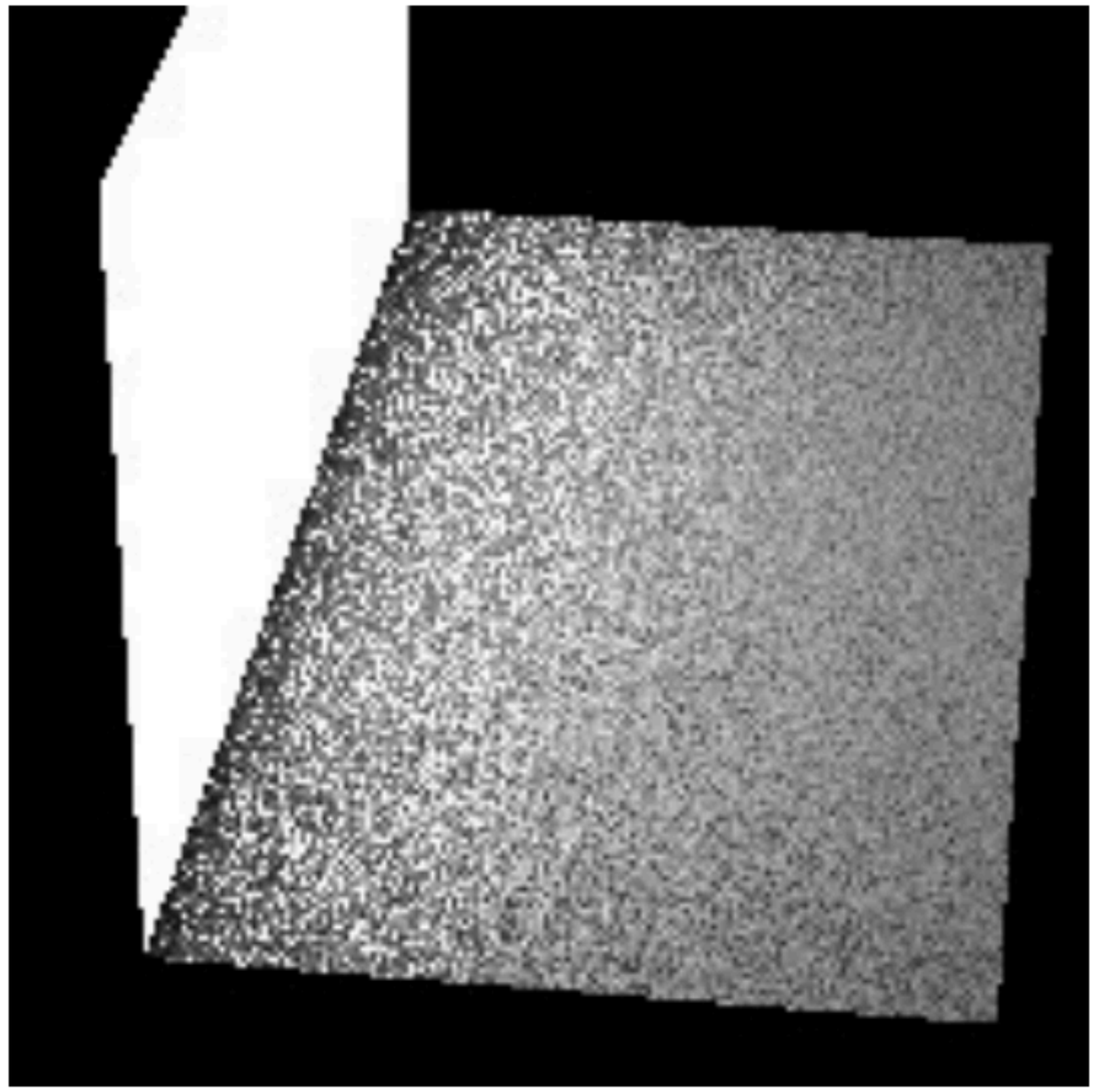
$$\frac{1}{N}\left(\sum_{i=1}^{N} w_i^{\text{n}} \frac{L'(\omega_i') \left|\omega_i' \cdot n\right|}{p_{\text{nee}}(\omega_i')} + \sum_{j=1}^{N} w_j^{\text{h}} \frac{L'(\omega_j') \left|\omega_j' \cdot n\right|}{p_{\text{hemi}}(\omega_j')}\right)$$

$$w_i^n = \frac{p_{\text{nee}}(\omega_i)}{p_{\text{nee}}(\omega_i) + p_{\text{hemi}}(\omega_i)} \qquad w_j^h = \frac{p_{\text{hemi}}(\omega_j)}{p_{\text{nee}}(\omega_j) + p_{\text{hemi}}(\omega_j)}$$

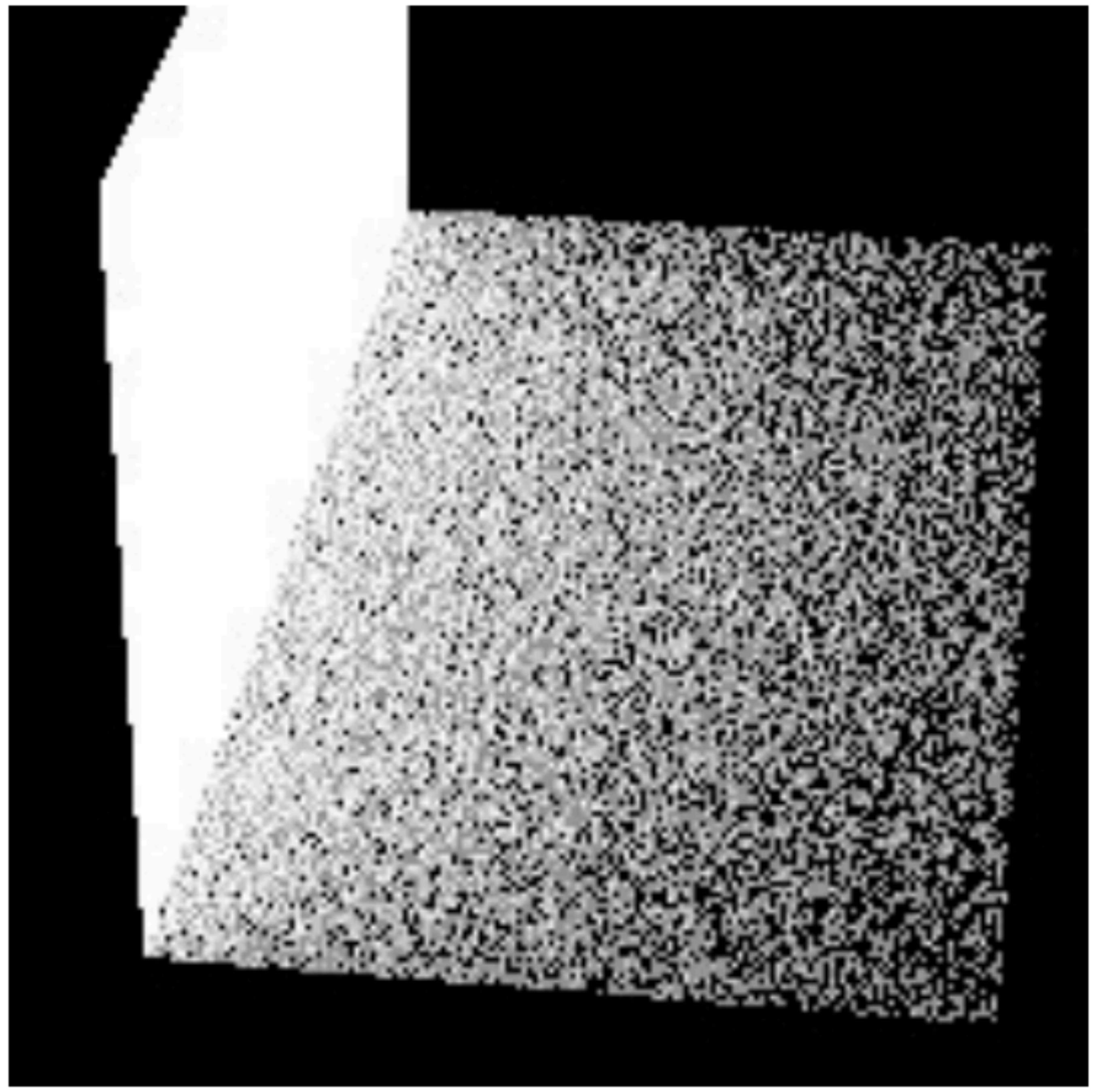
see https://github.com/BachiLi/lajolla_public/blob/main/src/path_tracing.h for the implementation

https://cseweb.ucsd.edu/~tzli/cse168/lectures/12_multiple_importance_sampling.pdf for the math

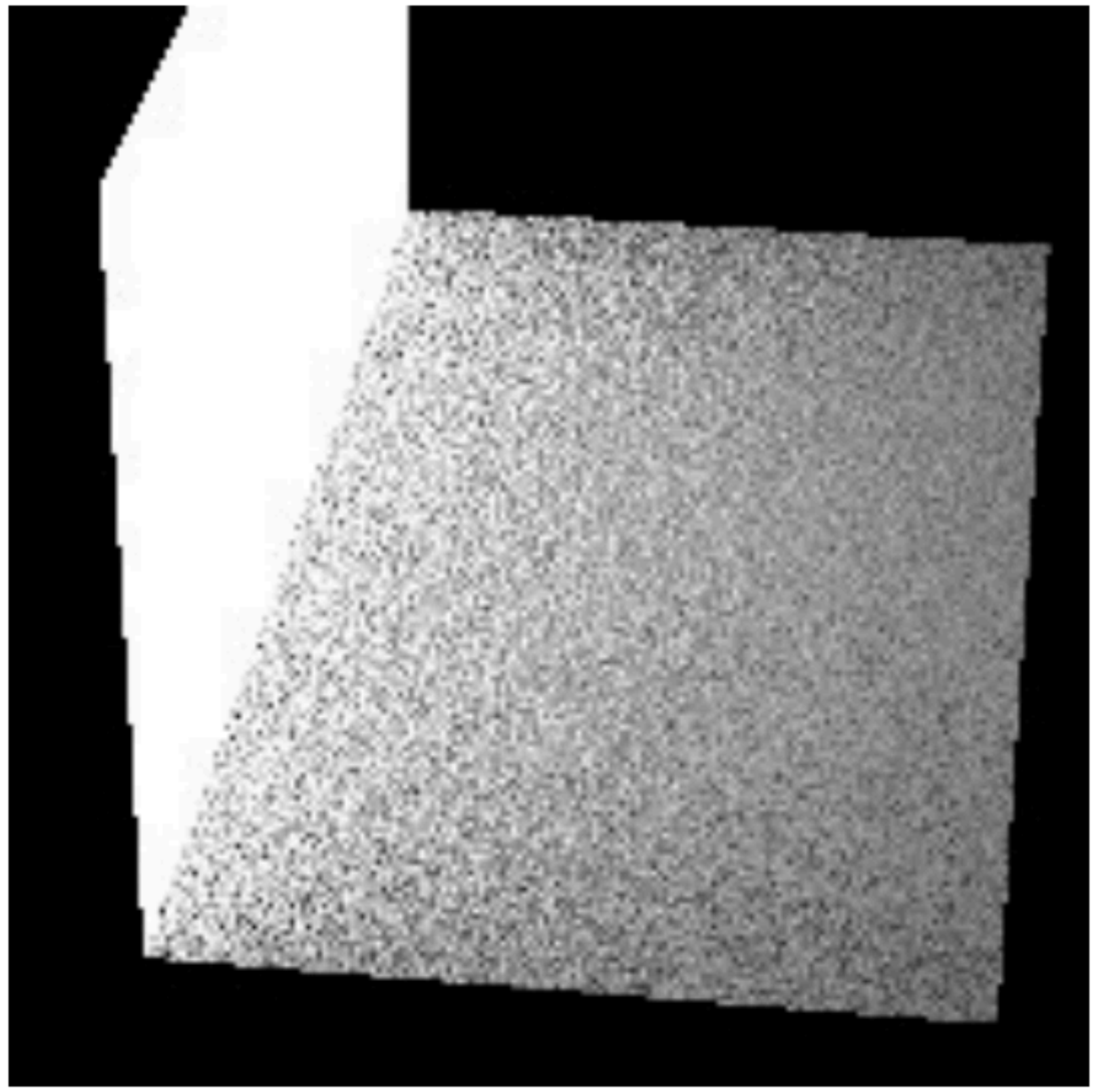
MIS combines the best of both worlds



next event estimation



cosine-weighted
hemisphere sampling

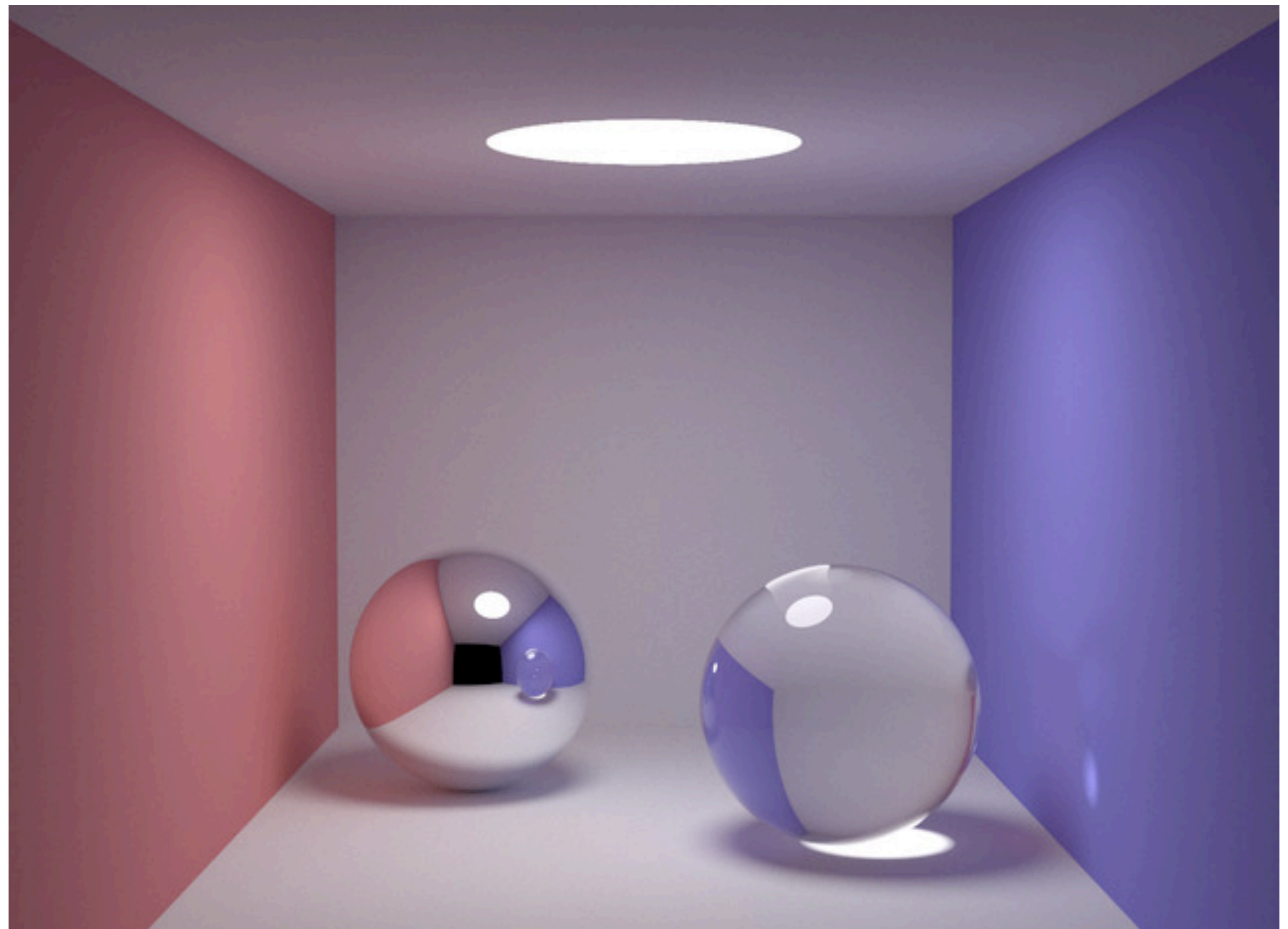


multiple importance sampling

“Robust Monte Carlo Methods for Light Transport Simulation”, Eric Veach

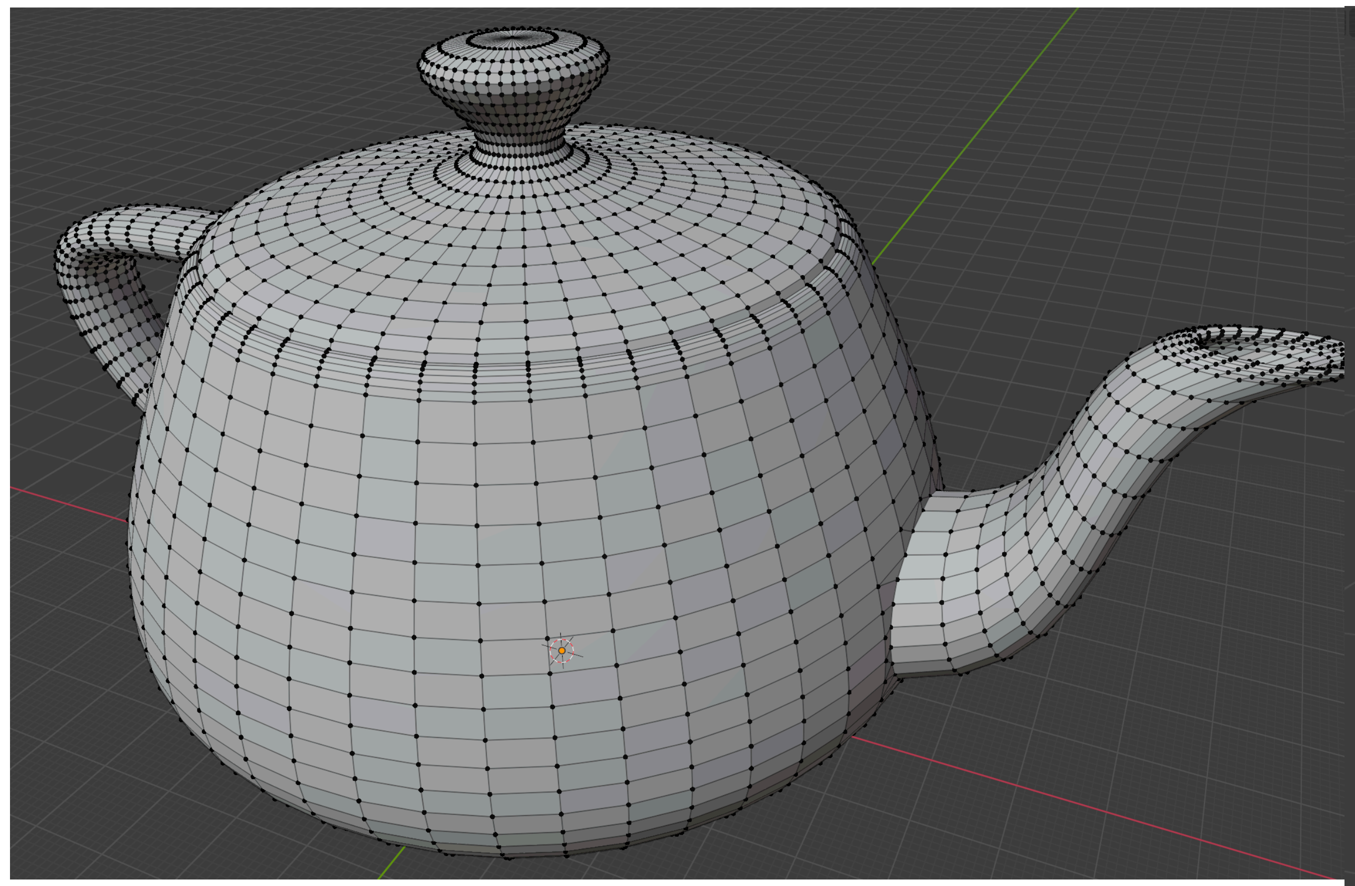
Smallpt: sphere geometry only

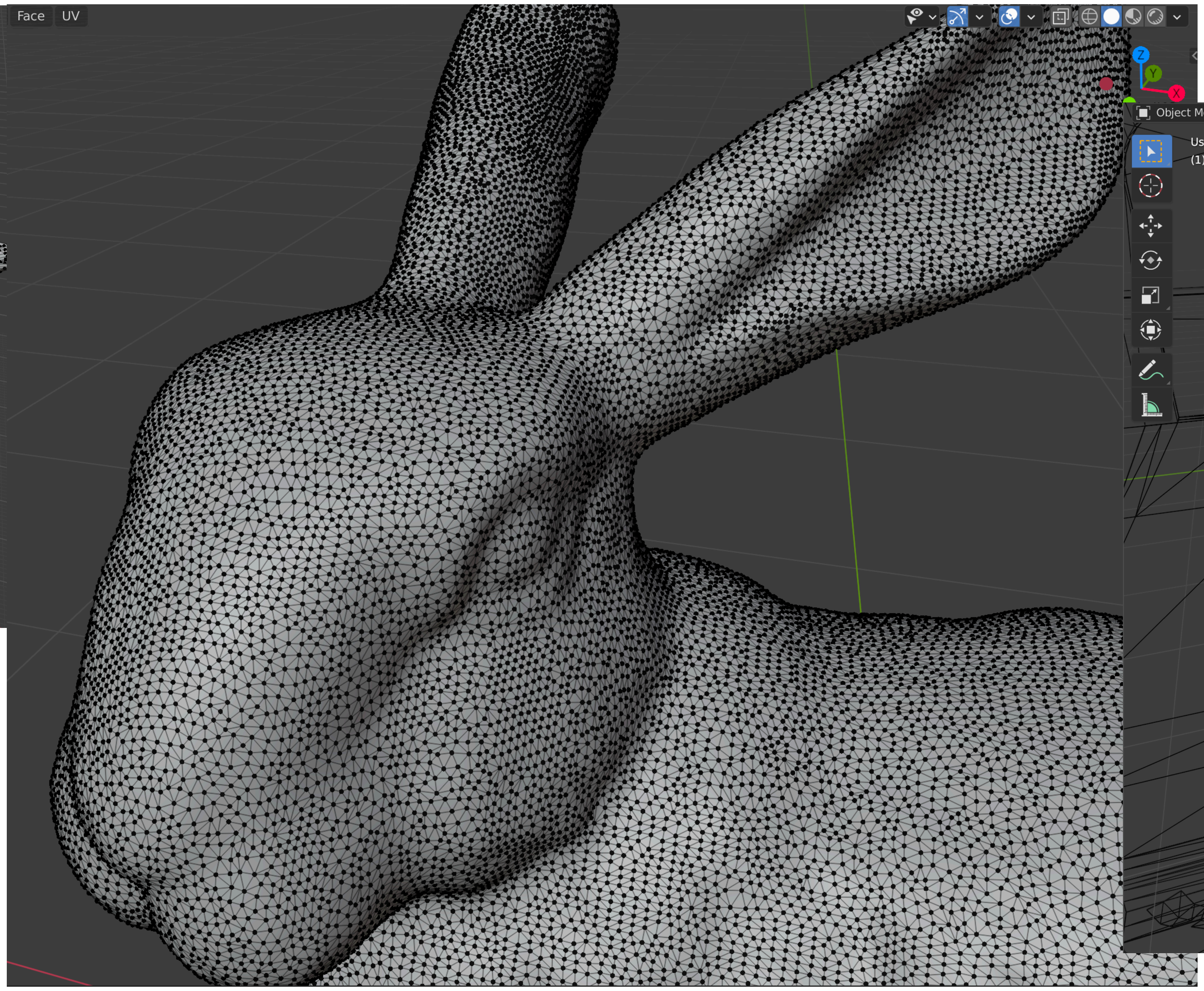
```
Sphere spheres[] = {//Scene: radius, position, emission, color, material
  Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
  Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
  Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
  Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
  Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
  Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
  Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
  Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
  Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF) //Lite
};
```

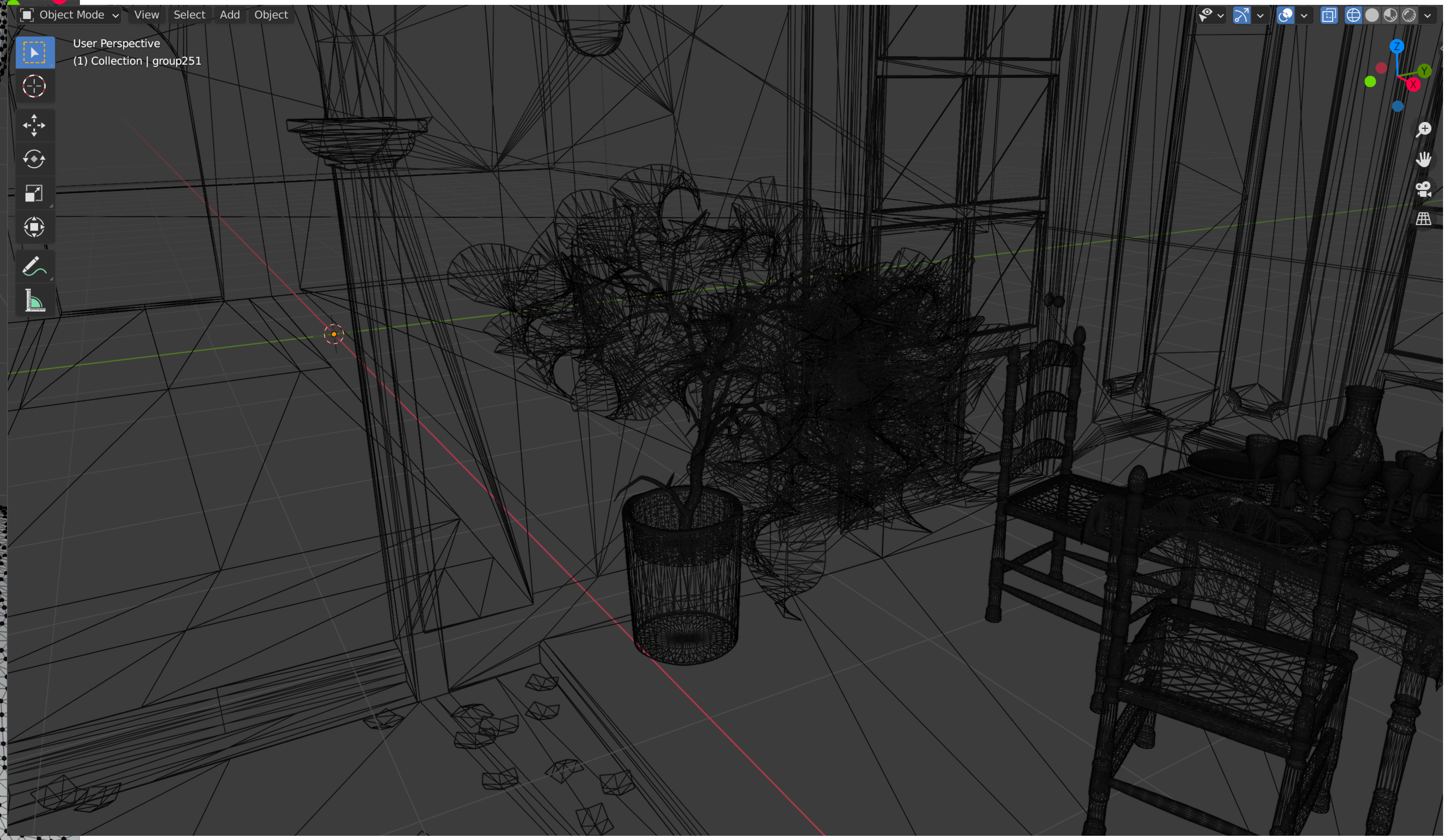


More commonly used geometry primitive: triangle mesh

quiz: why?

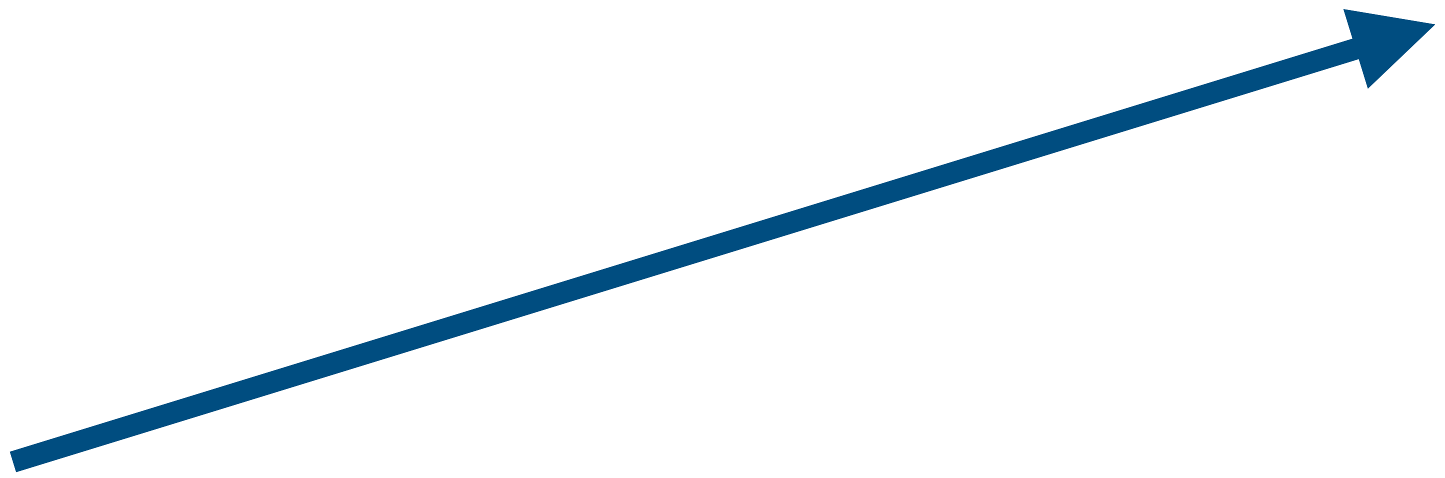




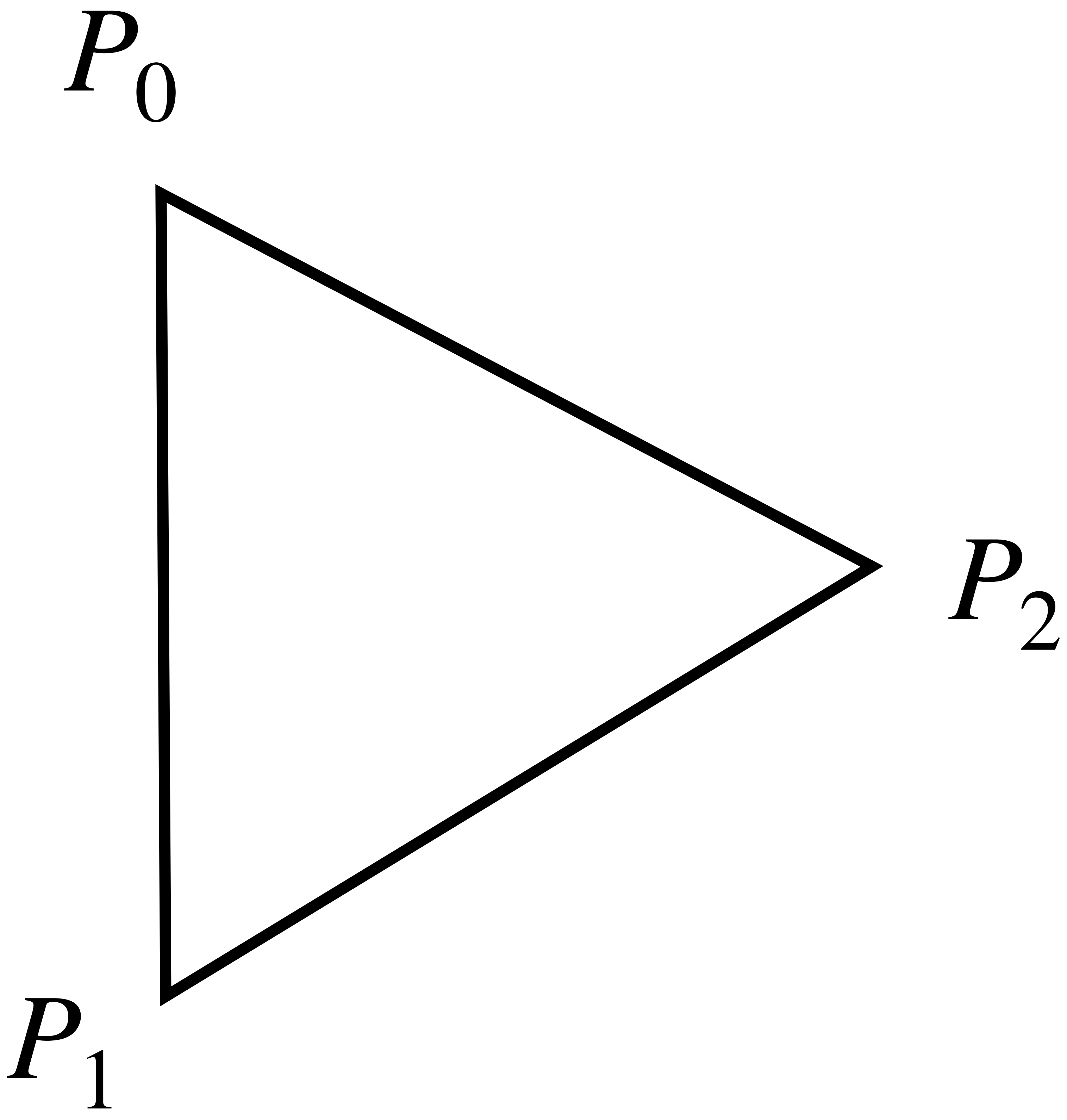


Ray-triangle intersection

quiz: how do we solve for intersection?



$$\mathbf{x} = \mathbf{o} + t \cdot \mathbf{d}$$



$$\mathbf{x} = (1 - b_1 - b_2)\mathbf{P}_0 + b_1\mathbf{P}_1 + b_2\mathbf{P}_2$$

https://en.wikipedia.org/wiki/Barycentric_coordinate_system

Ray-triangle intersection

$$\mathbf{o} + t \cdot \mathbf{d} = (1 - b_1 - b_2)\mathbf{P}_0 + b_1\mathbf{P}_1 + b_2\mathbf{P}_2$$

$$o_x + t \cdot d_x = (1 - b_1 - b_2)P_{0_x} + b_1 P_{1_x} + b_2 P_{2_x}$$
$$o_y + t \cdot d_y = (1 - b_1 - b_2)P_{0_y} + b_1 P_{1_y} + b_2 P_{2_y}$$
$$o_z + t \cdot d_z = (1 - b_1 - b_2)P_{0_z} + b_1 P_{1_z} + b_2 P_{2_z}$$

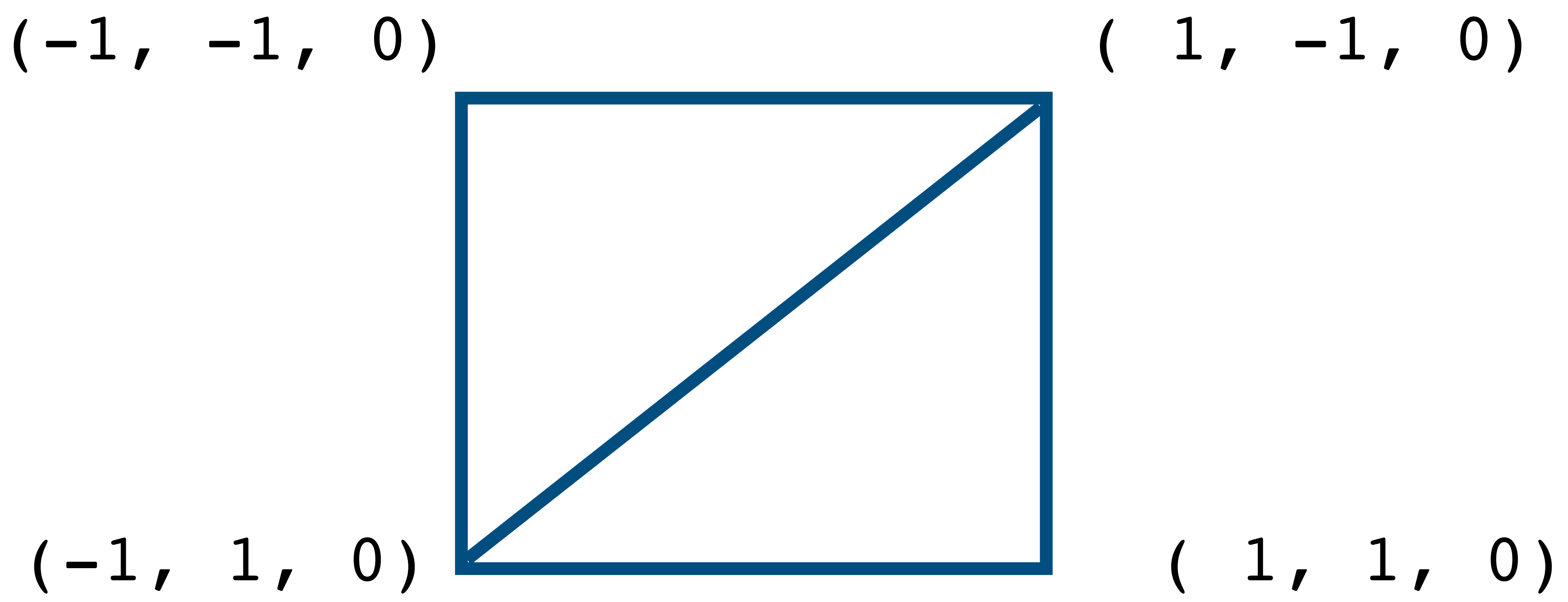
3 unknowns (t, b1, b2), 3 linear equations

"Moller-Trumbore algorithm"

Lajolla supports triangle meshes and spheres

A triangle mesh representing a quad:

```
positions = {{-1, -1, 0},
             { 1, -1, 0},
             {-1,  1, 0},
             { 1,  1, 0}};
indices = {{0, 1, 2},
           {2, 1, 3}}
```



quiz: why don't we just represent everything as individual triangles?

https://github.com/BachiLi/lajolla_public/blob/main/src/shape.h

Lajolla supports triangle meshes and spheres

```cpp
struct ShapeBase {
    int material_id = -1;
    // …
};

struct Sphere : public ShapeBase {
    Vector3 position;
    Real radius;
};

struct TriangleMesh : public ShapeBase {
    std::vector<Vector3> positions;
    std::vector<Vector3i> indices;
    std::vector<Vector3> normals;
    std::vector<Vector2> uvs;
    // …
};

using Shape = std::variant<Sphere, TriangleMesh>;
```

see HW0 for the reasoning of using std::variant

Mitsuba scene format

```xml
<shape type="sphere">
  <point name="center" x="0" y="0" z="0"/>
  <float name="radius" value="1.0"/>
  <!-- ... -->
</shape>

<shape type="obj">
  <string name="filename" value="meshes/cbox_floor.obj"/>
  <!-- ... -->
</shape>
```

Wavefront obj file (ASCII)

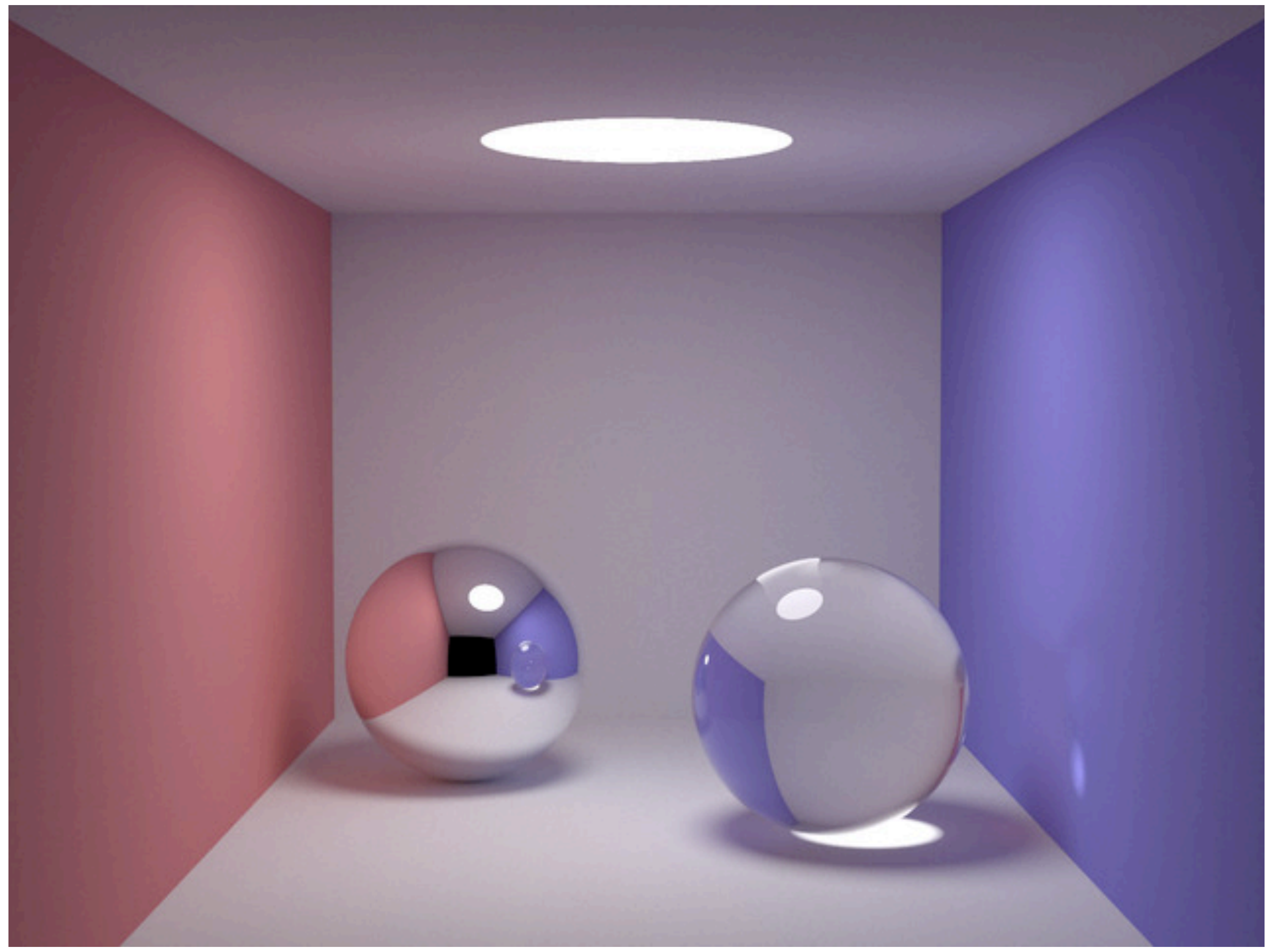
https://en.wikipedia.org/wiki/Wavefront_.obj_file

Mitsuba serialized mesh (binary)

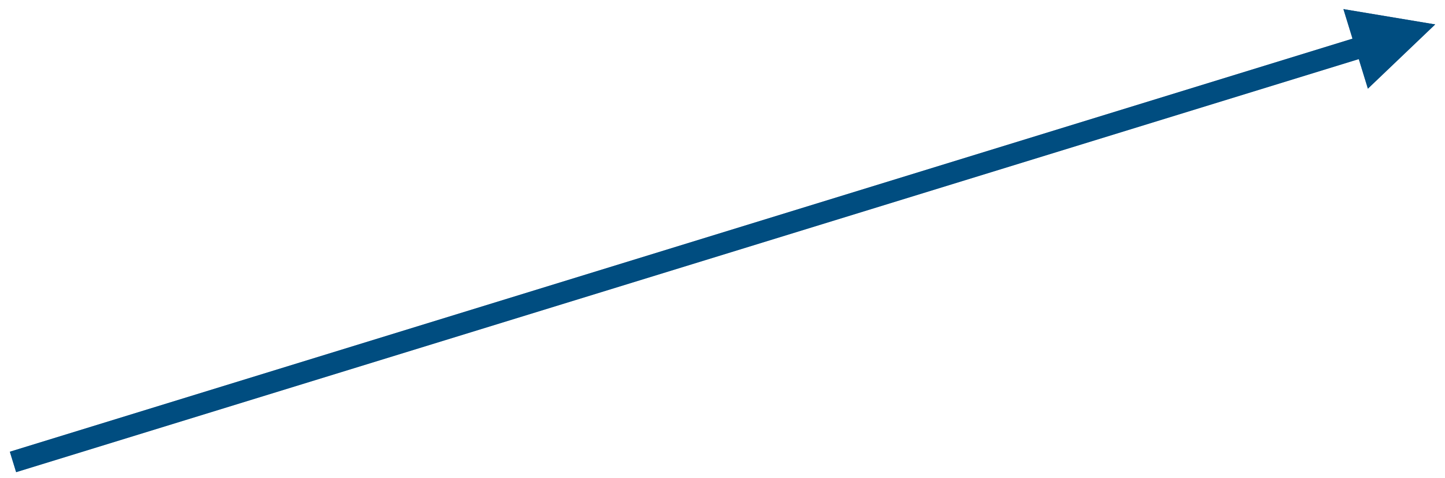
```xml
<shape type="serialized">
  <string name="filename" value="matpreview.serialized"/>
  <integer name="shapeIndex" value="2"/>
  <transform name="toWorld">
    <matrix value="0.614046 0.614047 0 -1.78814e-07 -0.614047
    <translate z="0.01"/>
  </transform>
  <!-- ... -->
</shape>
```

Smallpt: loop over all spheres to test intersections

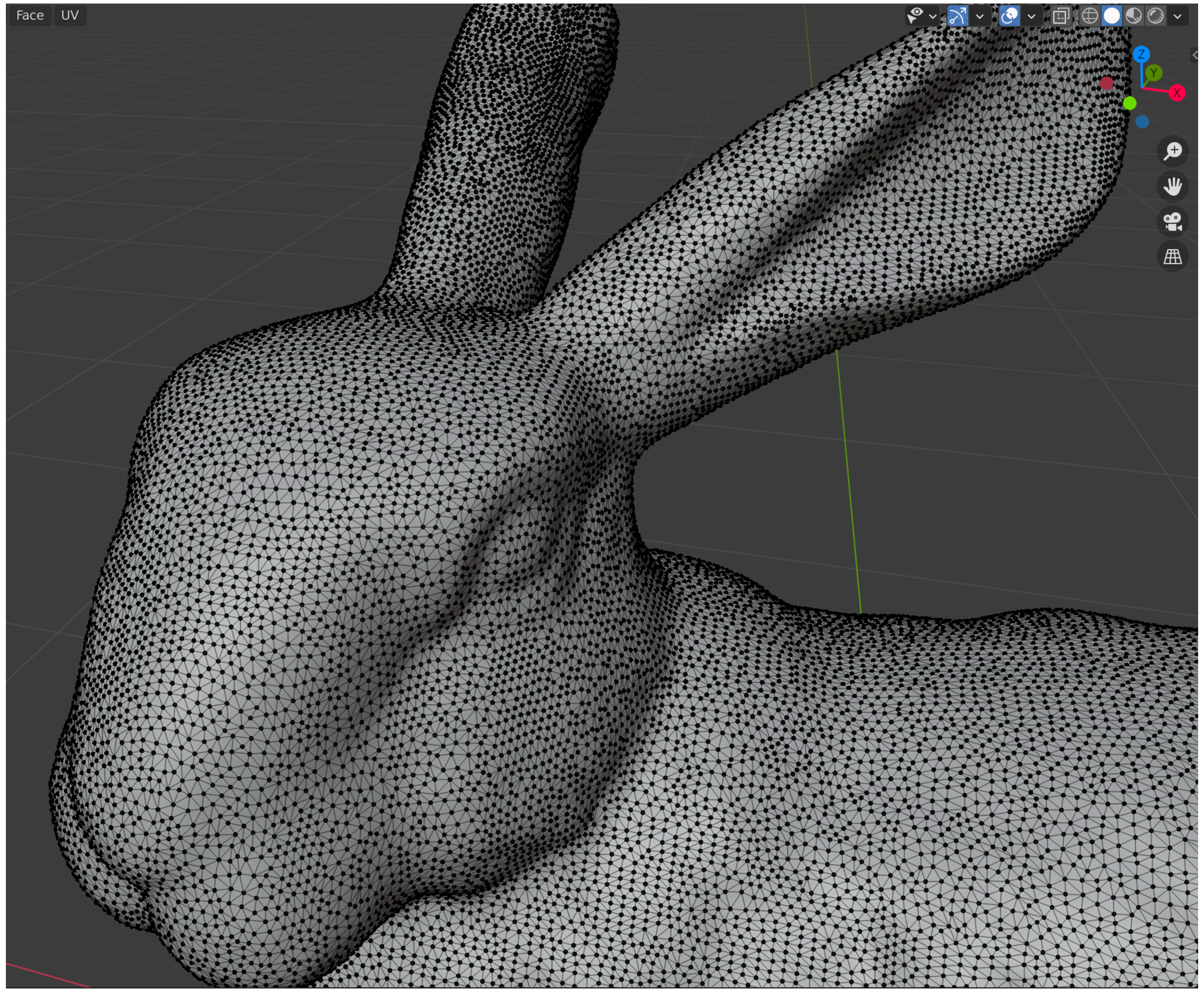
```
inline bool intersect(const Ray &r, double &t, int &id){
  double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
  for(int i=int(n);i--;) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
  return t<inf;
}
```



Lots of triangles:
how to go through them efficiently?

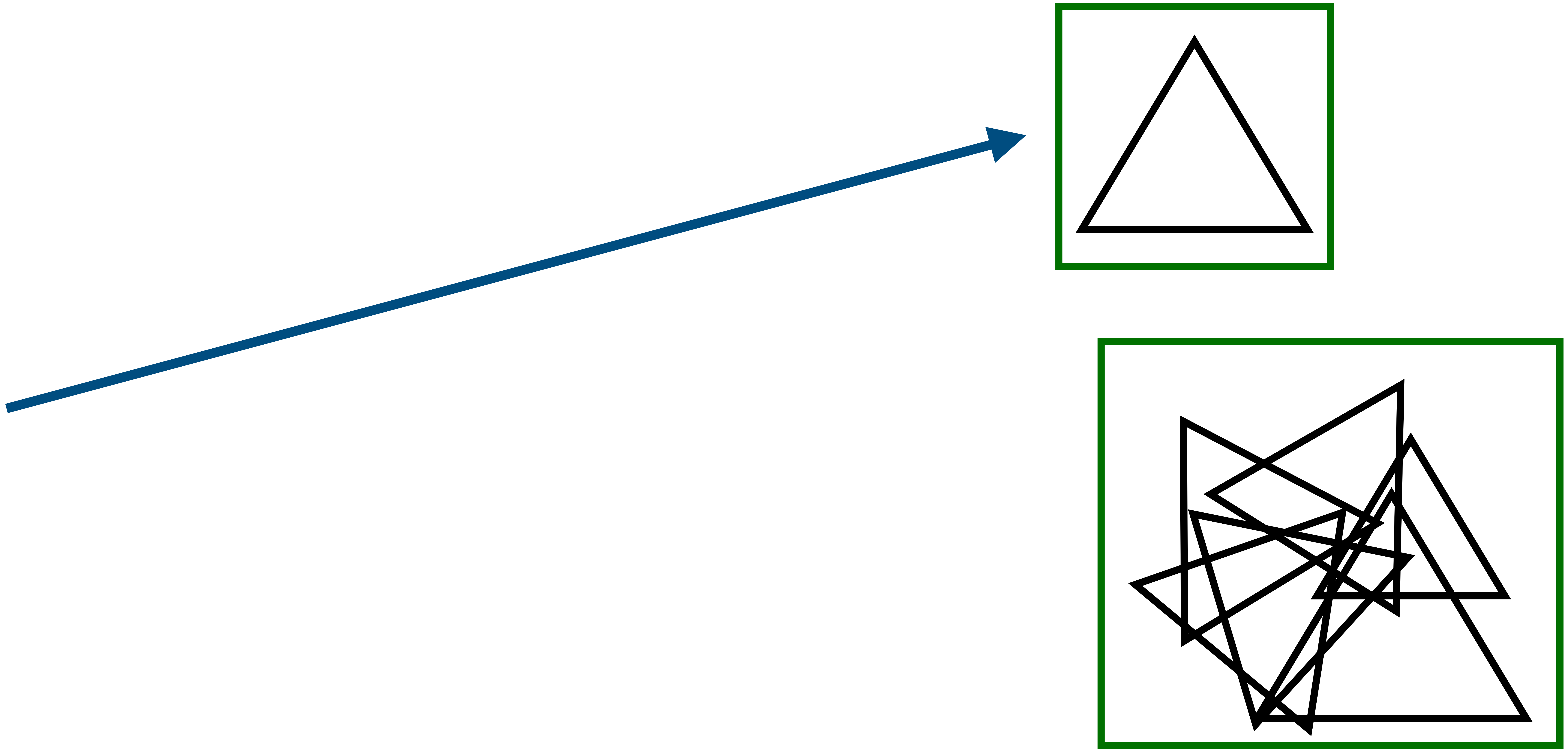


$$\mathbf{x} = \mathbf{o} + t \cdot \mathbf{d}$$



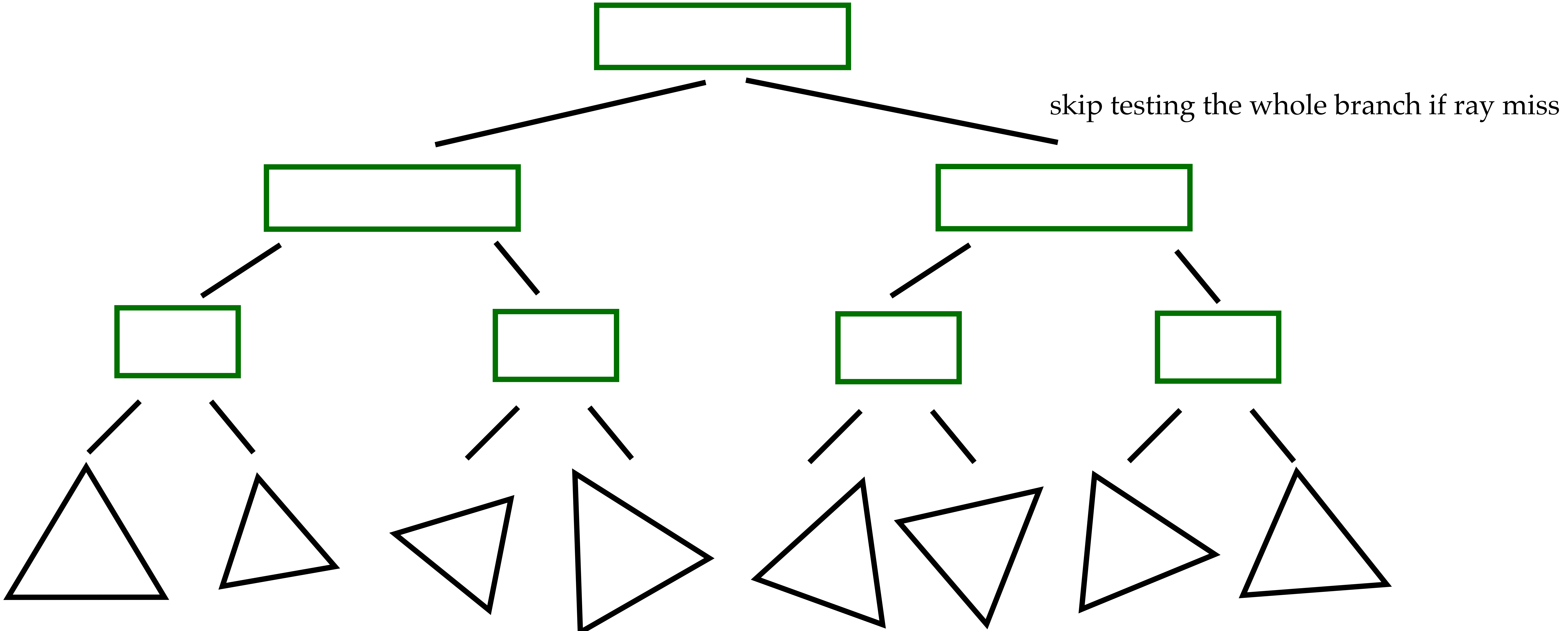
Bounding Volumes Hierarchy

- idea: test a group of triangles at a time by looking at their bounding volumes



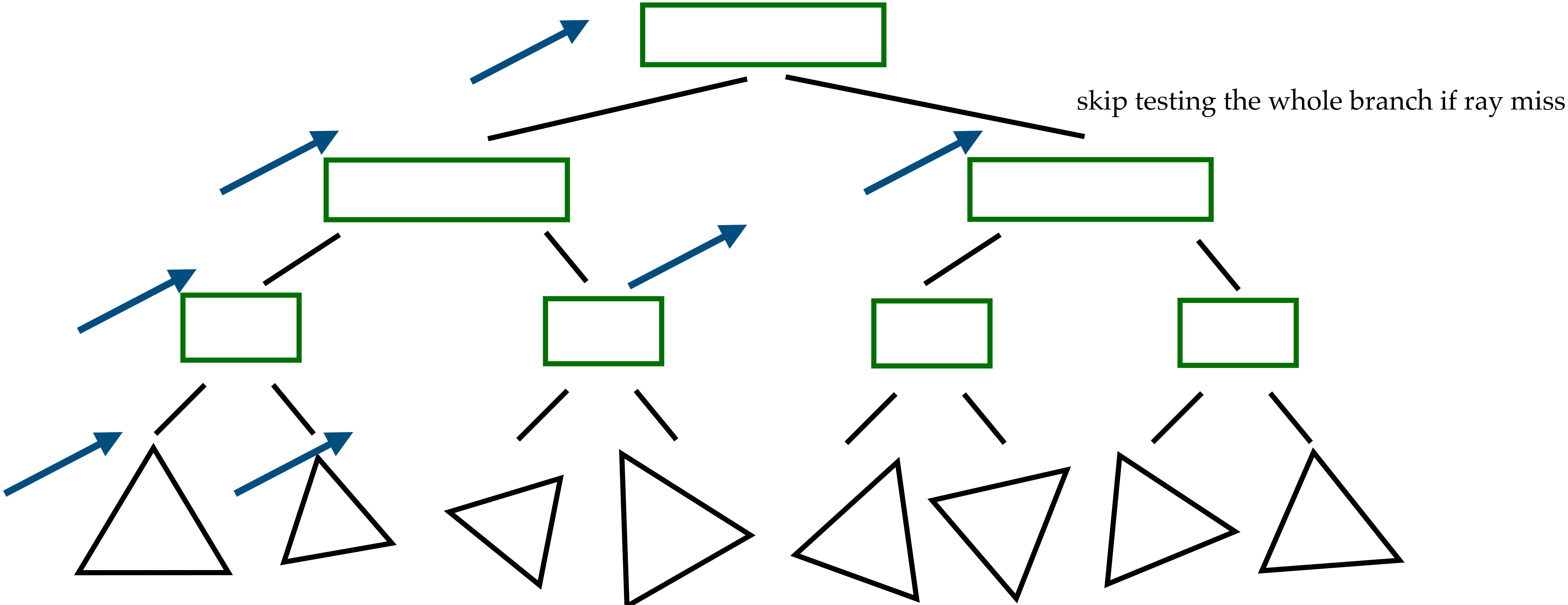
Bounding Volumes Hierarchy

- idea: test a group of triangles at a time by looking at their bounding volumes



Bounding Volumes Hierarchy

- idea: test a group of triangles at a time by looking at their bounding volumes



Decades of research on acceleration structures

- optimized to death, though still an active research area

Ray Tracing Complex Scenes

Timothy L. Kay
James T. Kajiya
California Institute of Technology
Pasadena, CA 91125

Space Subdivision for Fast Ray Tracing

Andrew S. Glassner
University of North Carolina at Chapel Hill

Fast Ray Tracing by Ray Classification

James Arvo
David Kirk
Apollo Computer, Inc.
330 Billerica Road
Chelmsford, MA 01824

Efficiency Issues for Ray Tracing

Brian Smits*
University of Utah
February 19, 1999

B-KD Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes

Sven Woop†, Gerd Marmitt‡, and Philipp Slusallek§
Computer Graphics Lab, Saarland University, Germany

Improved Two-Level BVHs using Partial Re-Braiding

Carsten Benthin, Intel Corporation — Sven Woop, Intel Corporation — Ingo Wald, Intel Corporation — Attila T. Áfra, Intel Corporation

Filtering, Clustering and Hierarchy Construction: a New Solution for Ray-Tracing Complex Scenes

Frédéric Cazals,[1] George Drettakis,[1,2] Claude Puech [1]

Dual-Split Trees

Daqi Lin, University of Utah — Konstantin Shkurko, University of Utah — Ian Mallett, Univers — Cem Yuksel

Compressed-Leaf Bounding Volume Hierarchies (originally submitted, un-shortened version)

Carsten Benthin — Ingo Wald — Sven Woop — Attila T. Áfra
Intel Corporation

Wide BVH Traversal with a Short Stack

K. Vaidyanathan — S. Woop — C. Benthin
Intel Corporation

Two-Level Grids for Ray Tracing on GPUs

Javor Kalojanov[1] and Markus Billeter[2] and Philipp Slusallek[1,3]

HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry

J. Pantaleoni[1] and D. Luebke[1]
[1]NVIDIA Research

An N-ary BVH Child Node Sorting Technique for Occlusion Tests

Shinji Ogaki, OLM Digital, Inc. — Alexandre Derouet-Jourdan, OLM Digital, Inc. / JST CREST

Bonsai: Rapid Bounding Volume Hierarchy eneration using Mini Trees

[2], R. Barringer[2], M. Doggett[2], and T. Akenine-Möller[1,2]
[1]Intel Corporation [2]Lund University

Interactive Rendering with Coherent Ray Tracing

Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner
Computer Graphics Group, Saarland University

Understanding the Efficiency of Ray Traversal on GPUs

Timo Aila*, NVIDIA Research — Samuli Laine*, NVIDIA Research

On Quality Metrics of Bounding Volume Hierarchies

Timo Aila — Tero Karras — Samuli Laine
NVIDIA

Fast Parallel Construction of High-Quality Bounding Volume Hierarchies

Tero Karras — Timo Aila
NVIDIA

Ray Tracing Lossy Compressed Grid Primitives

Carsten Benthin — Karthik Vaidyanathan — Sven Woop
Intel Corporation

Efficient Incoherent Ray Traversal on GPUs Through Compressed Wide BVHs

Henri Ylitie, NVIDIA — Tero Karras, NVIDIA — Samuli Laine, NVIDIA

MergeTree: A Fast Hardware HLBVH Constructor for Animated Ray Tracing

TIMO VIITANEN, MATIAS KOSKELA, PEKKA JÄÄSKELÄINEN, HEIKKI KULTALA, and JARMO TAKALA,
Tampere University of Technology, Finland

Using Hardware Ray Transfor Ray/Primitive Intersections fo

I. WALD[(1)] N. MORRICAL[(1,3)] S. ZEL HUANG[(2)] V. PASCUCCI[(3)], [(1)]NVIDIA [(2

Lajolla uses the Embree library

- highly-optimized ray intersection routines
    - used by almost all production ray tracers

```
/* Intersects a single ray with the scene. */
void rtcIntersect1(RTCScene scene, RTCIntersectContext* context, RTCRayHit* rayhit);
```

Embree: A Kernel Framework for Efficient CPU Ray Tracing

Ingo Wald [†] Sven Woop [†] Carsten Benthin [†] Gregory S. Johnson [†] Manfred Ernst [‡]
Intel Corporation

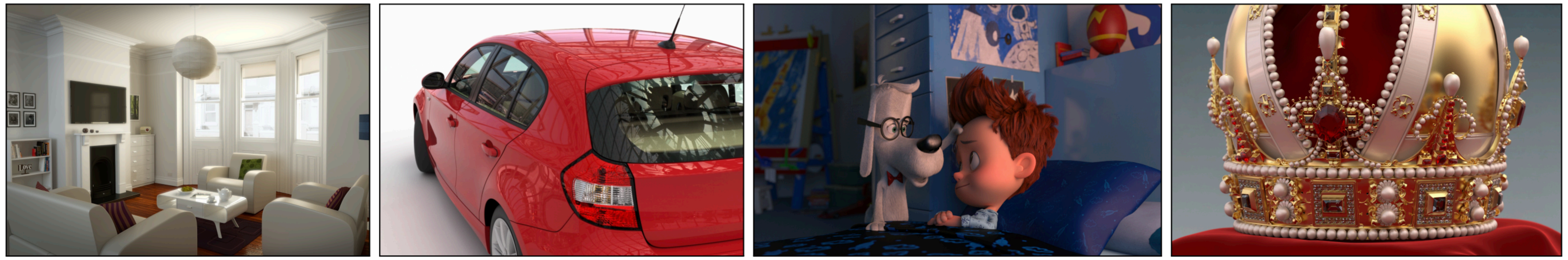


Figure 1: *Images produced by renderers which use the open source Embree ray tracing kernels. These scenes are computationally challenging due to complex geometry and spatially incoherent secondary rays. From left to right: The White Room model by Jay Hardy rendered in Autodesk RapidRT, a car model rendered in the Embree path tracer, a scene from the DreamWorks Animation movie "Peabody & Sherman" rendered with a prototype path tracer, and the Imperial Crown of Austria model by Martin Lubich rendered in the Embree path tracer.*

https://github.com/BachiLi/lajolla_public/blob/main/src/intersection.h

Intersection function in lajolla

```
/// Intersect a ray with a scene. If the ray doesn't hit anything,
/// returns an invalid optional output.
std::optional<PathVertex> intersect(const Scene &scene,
                                    const Ray &ray,
                                    const RayDifferential &ray_diff = RayDifferential{});
```

Intersection function in lajolla

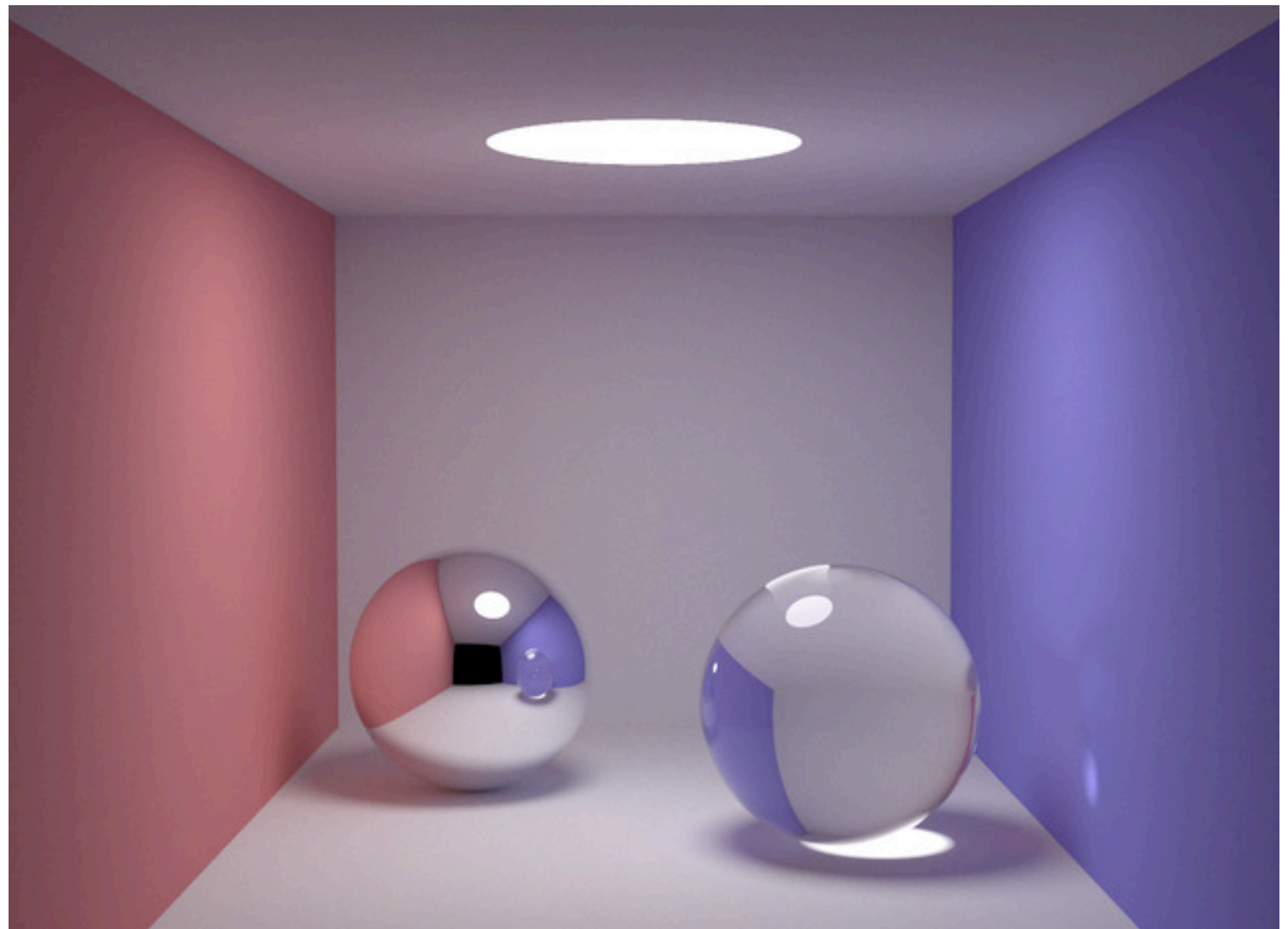
```
/// Intersect a ray with a scene. If the ray doesn't hit anything,
/// returns an invalid optional output.
std::optional<PathVertex> intersect(const Scene &scene,
                                    const Ray &ray,
                                    const RayDifferential &ray_diff = RayDifferential{});


struct PathVertex {
    Vector3 position;
    Vector3 geometry_normal; // always face at the same direction at shading_frame.n
    Frame shading_frame;
    Vector2 st; // A 2D parametrization of the surface. Irrelavant to UV mapping.
                // for triangle this is the barycentric coordinates, which we use
                // for interpolating the uv map.
    Vector2 uv; // The actual UV we use for texture fetching.
    // For texture filtering, stores approximatedly min(abs(du/dx), abs(dv/dx), abs(du/dy), abs(dv/dy))
    Real uv_screen_size;
    Real mean_curvature; // For ray differential propagation.
    Real ray_radius; // For ray differential propagation.
    int shape_id = -1;
    int primitive_id = -1; // For triangle meshes. This indicates which triangle it hits.
    int material_id = -1;

    // …
};
```

Smallpt: constant color across the surface

```
Sphere spheres[] = {//Scene: radius, position, emission, color, material
   Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
   Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
   Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
   Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
   Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
   Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
   Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
   Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
   Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF) //Lite
};
```

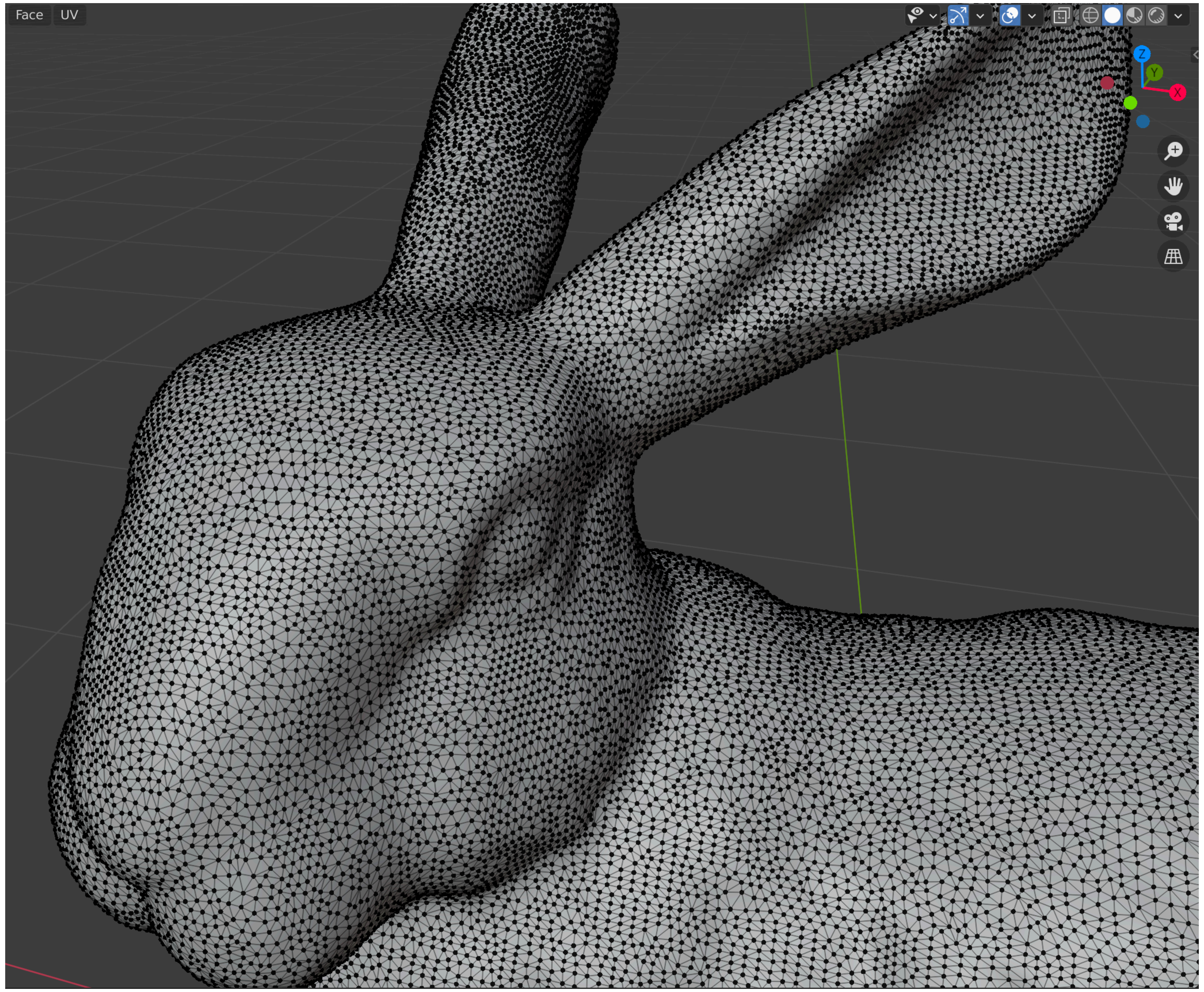


Real-world surfaces are colorful!

photo from K.C. Alfred
https://www.sandiegouniontribune.com/sdut-snake-path-2012jan20-htmlstory.html

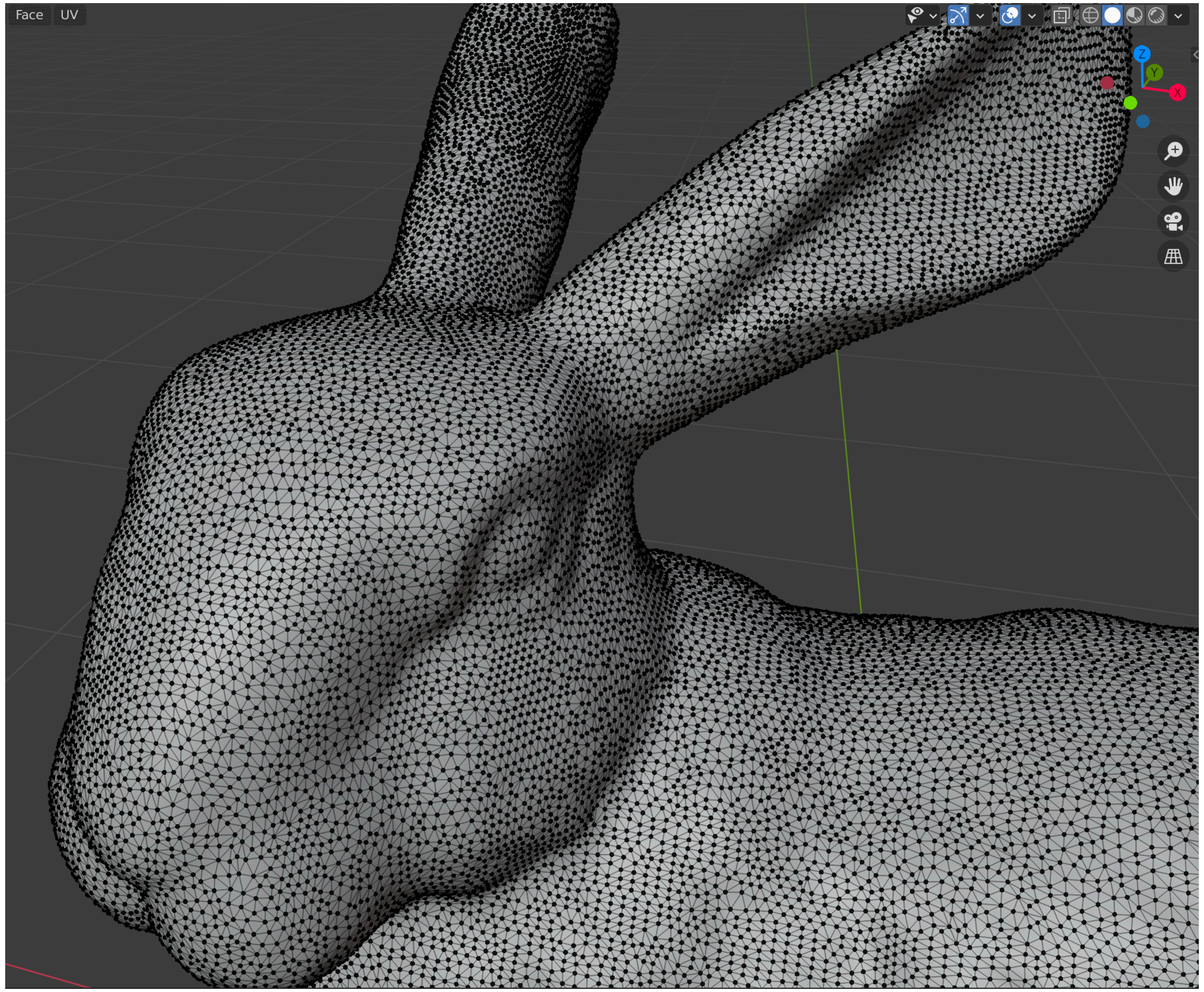
An option: assign a color to each triangle

quiz: what are the pros and cons?



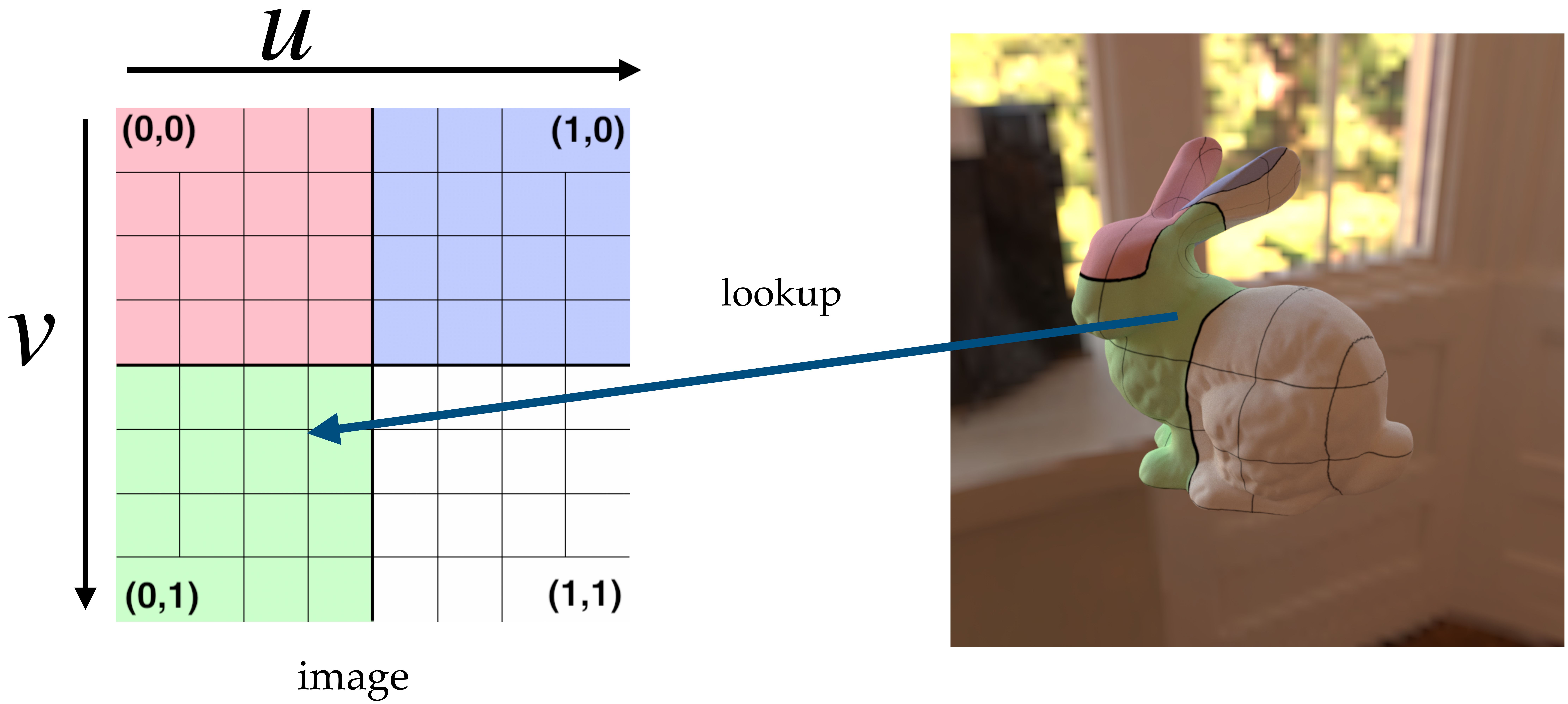
An option: assign a color to each triangle

- pros
  - simple
  - easy to edit (just paint on triangles)
- cons
  - couples geometric complexity with color complexity
  - hard to **filter**
    - more on this later



In practice: texture mapping

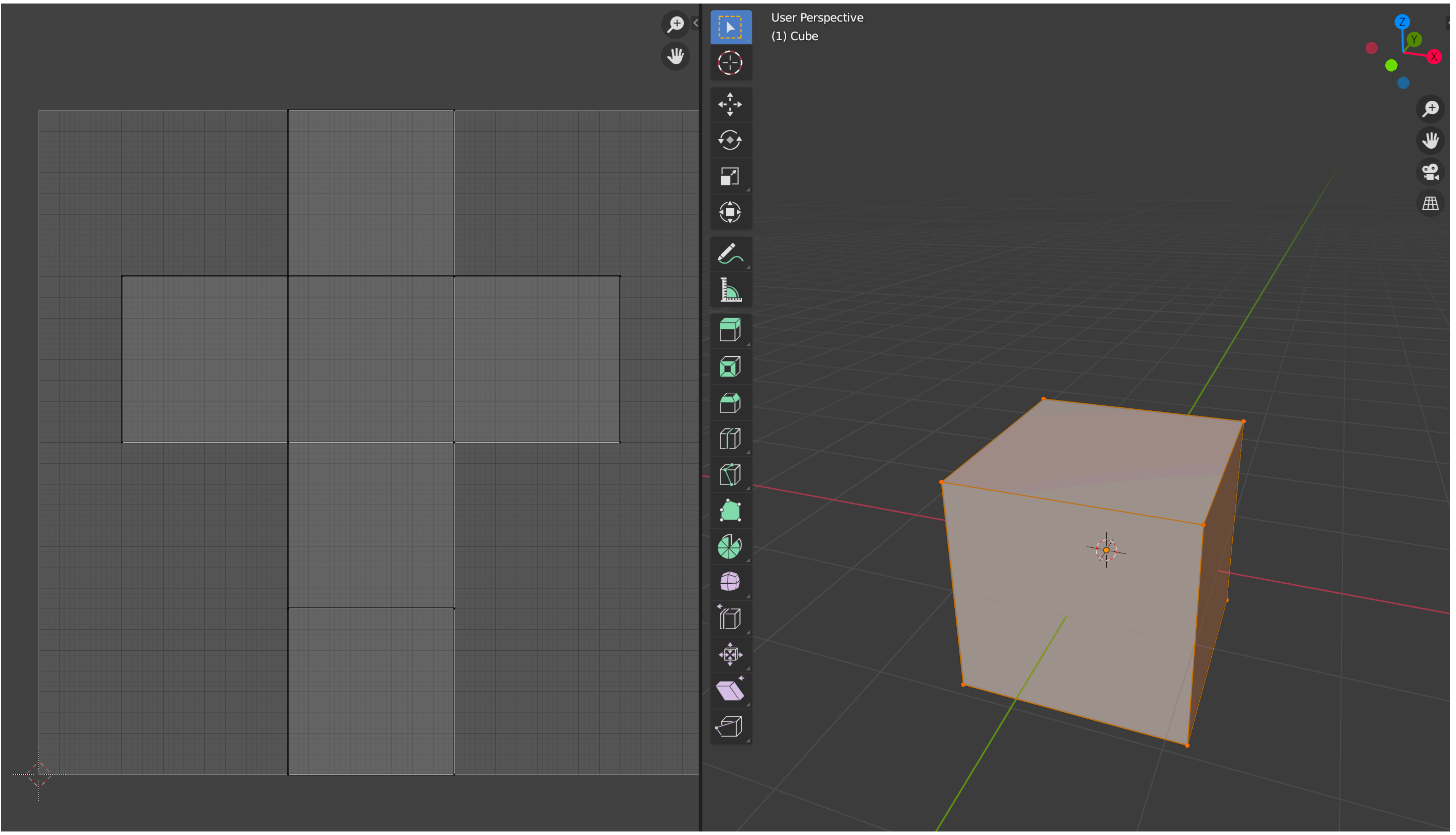
- assign a "UV" 2D vector to each point on the surface

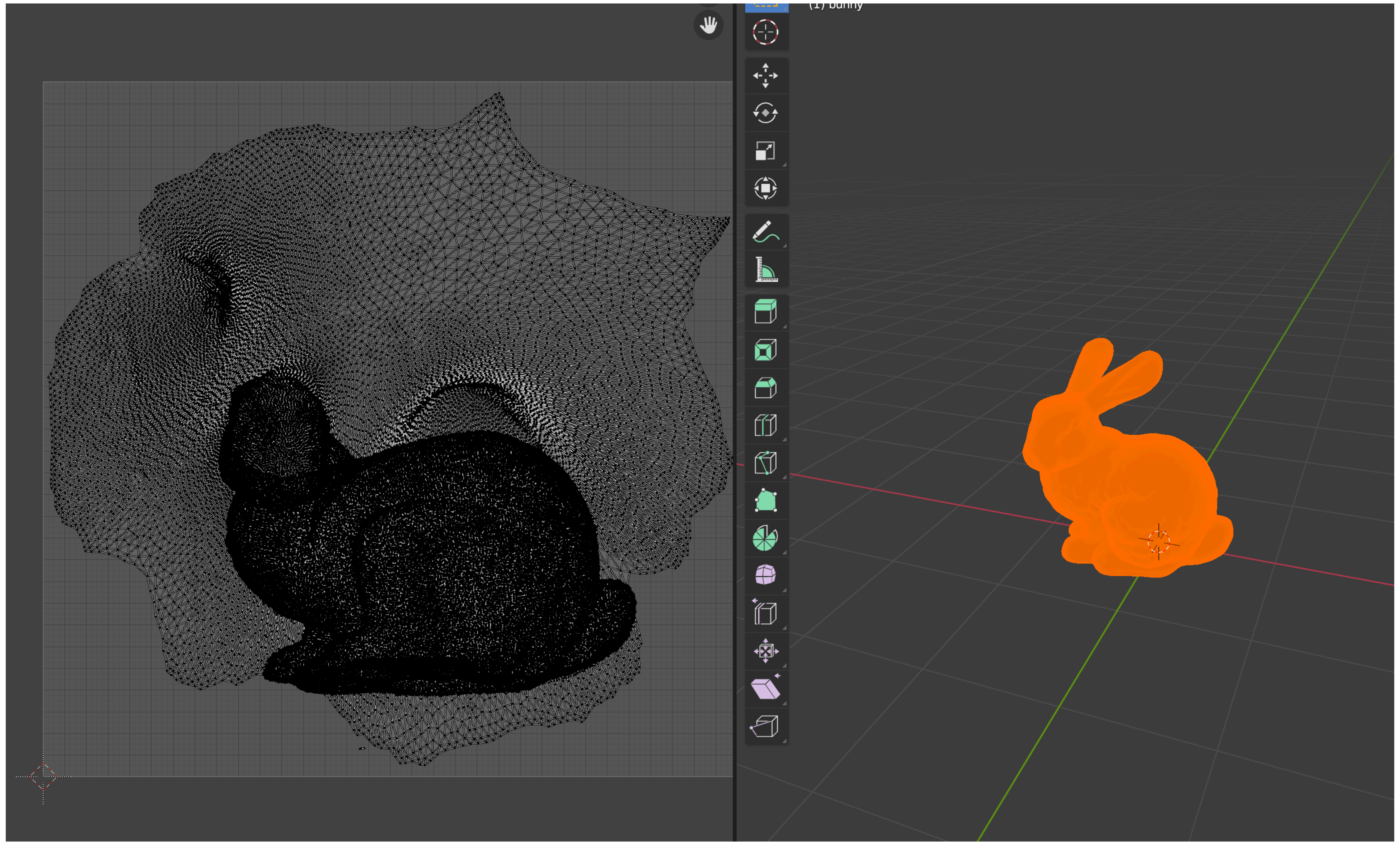


(texture from pbrt-v2 pbrt.org/scenes-v2) http://staff.ustc.edu.cn/~fuxm/projects/Peeling/index.html

UV mapping

- “unwrap” a surface and map it to a 2D square
  - automatic UV mapping is an active research area





In lajolla

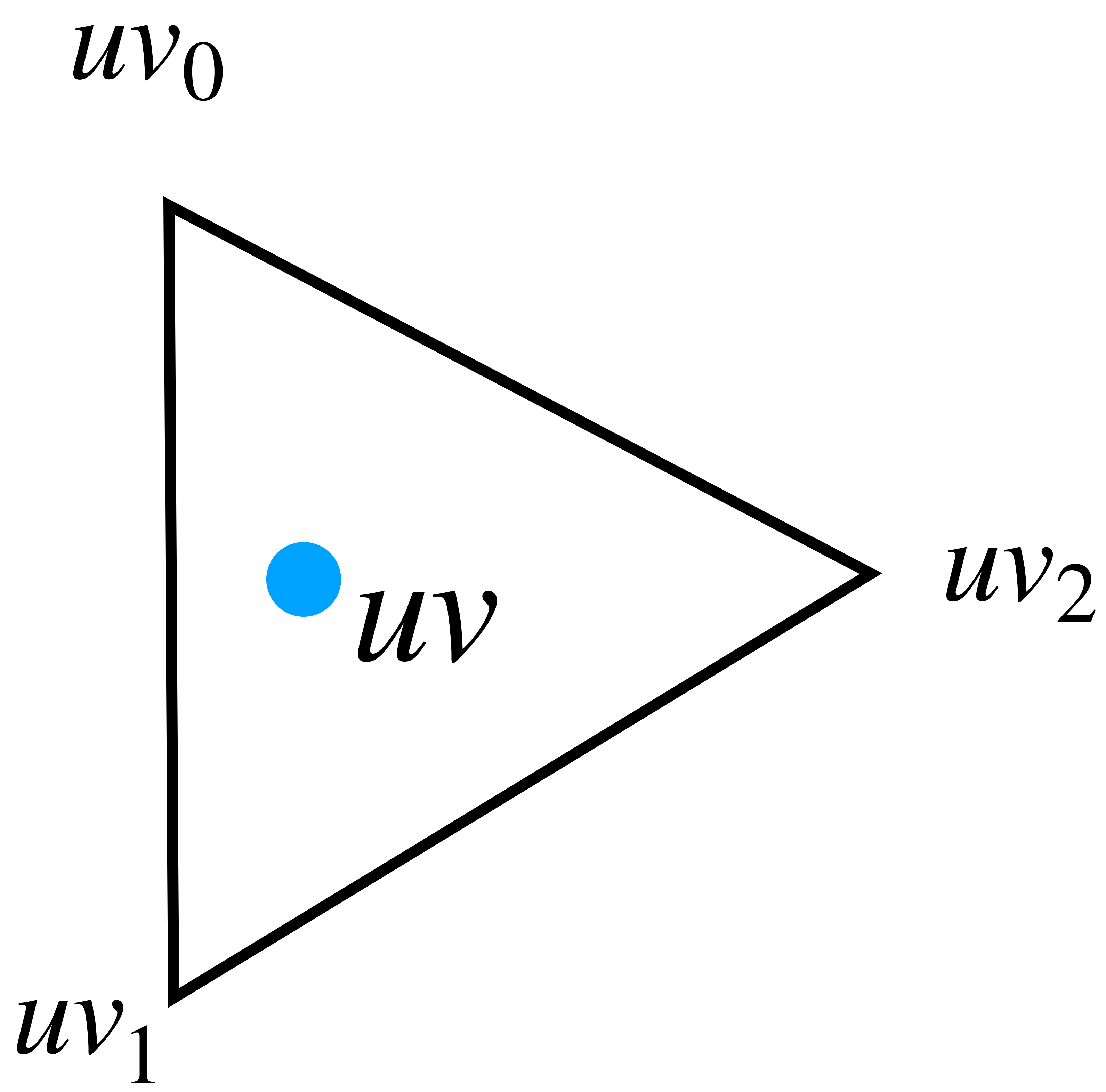
```
struct TriangleMesh : public ShapeBase {
    std::vector<Vector3> positions;
    std::vector<Vector3i> indices;
    std::vector<Vector3> normals;
    std::vector<Vector2> uvs;
    // …
};
```

uv per-vertex

```
// Barycentric coordinates are stored in vertex.st
Vector2 uv = (1 - vertex.st[0] - vertex.st[1]) * uvs[0] +
             vertex.st[0] * uvs[1] +
             vertex.st[1] * uvs[2];
```

```
struct PathVertex {
    // …
    Vector2 st; // A 2D parametrization of the surface. Irrelavant to UV mapping.
                // for triangle this is the barycentric coordinates, which we use
                // for interpolating the uv map.
    Vector2 uv; // The actual UV we use for texture fetching.
    // …
};
```

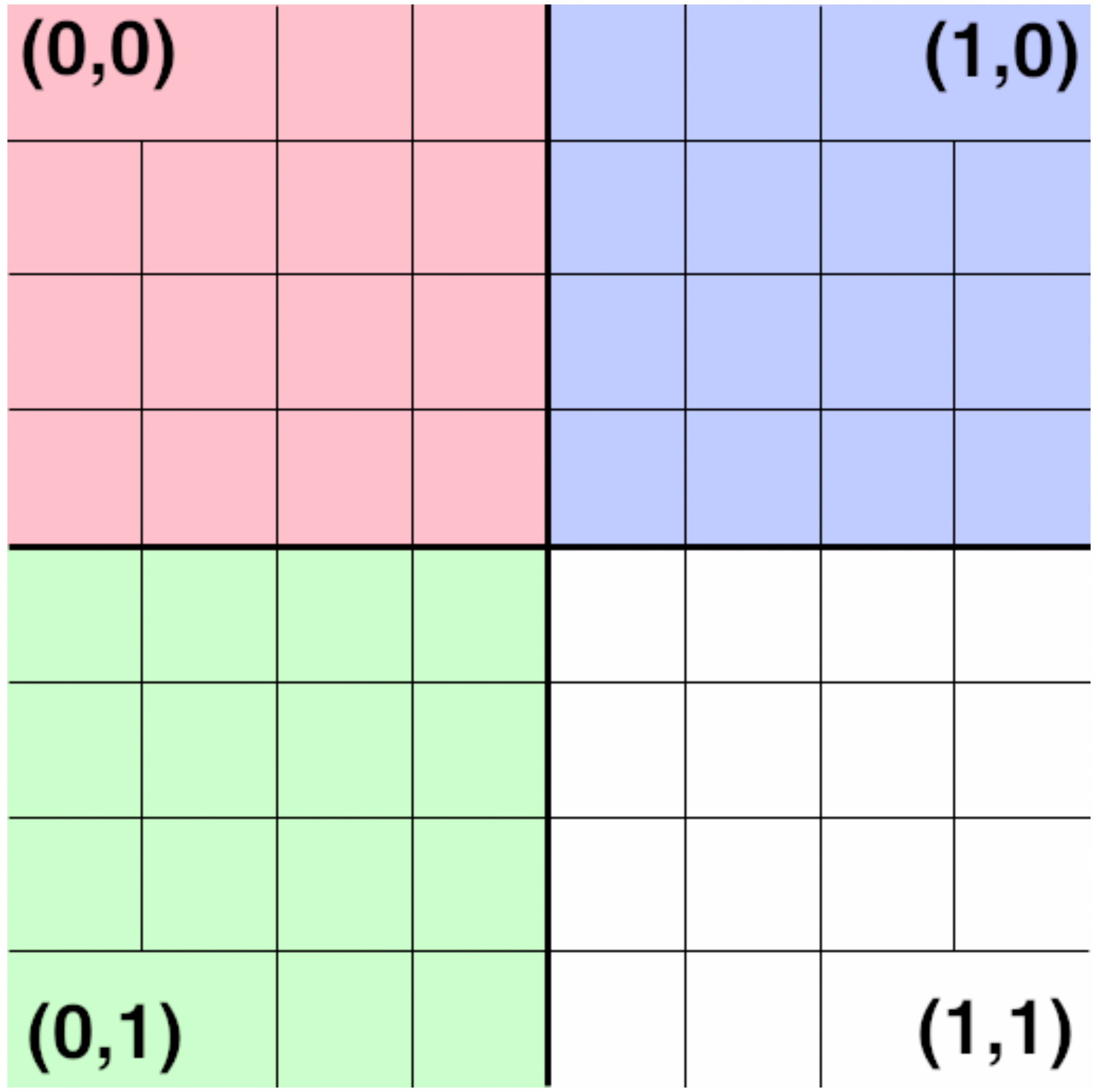
Obtain UV by interpolating values from vertices



$$uv = (1 - b_1 - b_2)uv_0 + b_1 uv_1 + b_2 uv_2$$

Texture mapping

quiz: what are the pros and cons?



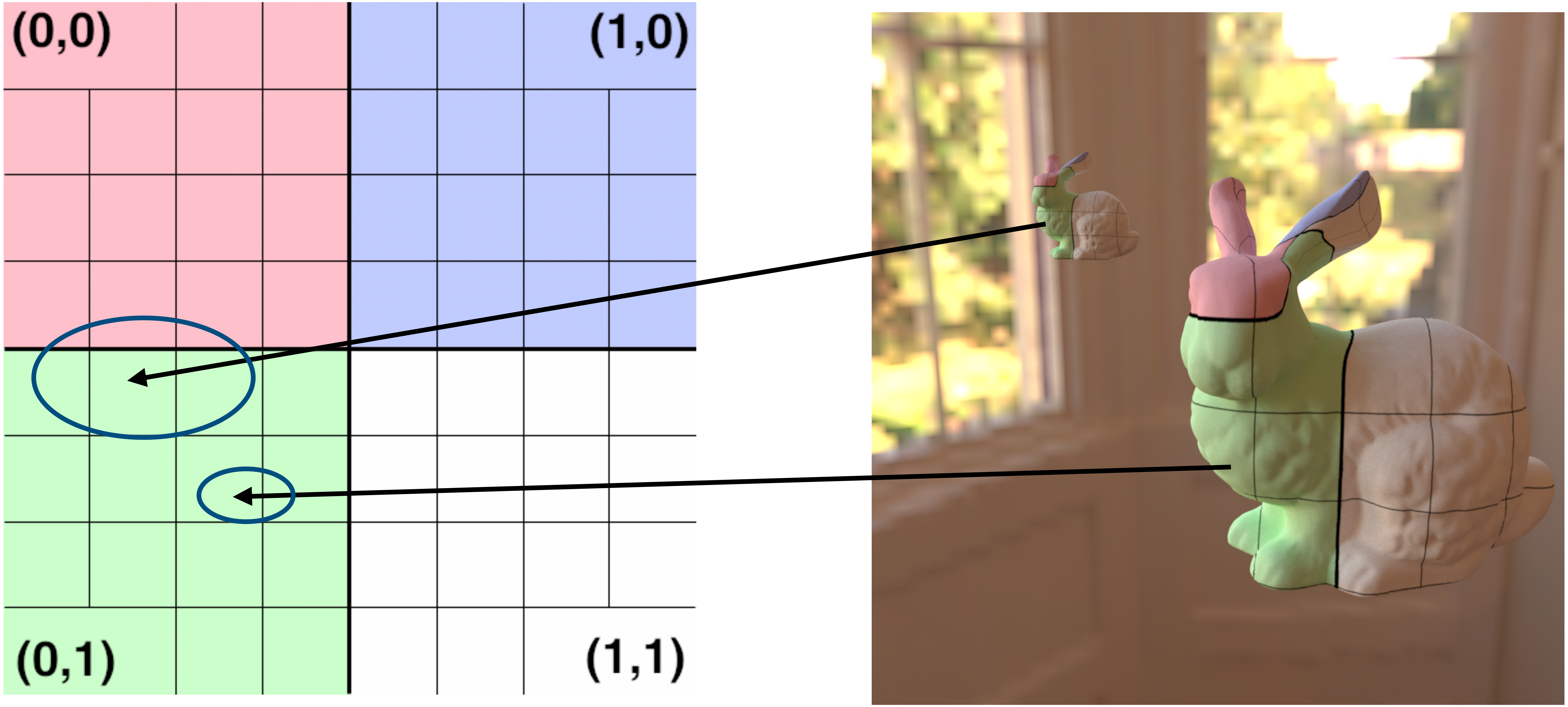
Texture mapping: pros and cons

- pros
  - different sampling rates for geometry and color
  - much easier to filter
- cons
  - uv mapping is hard

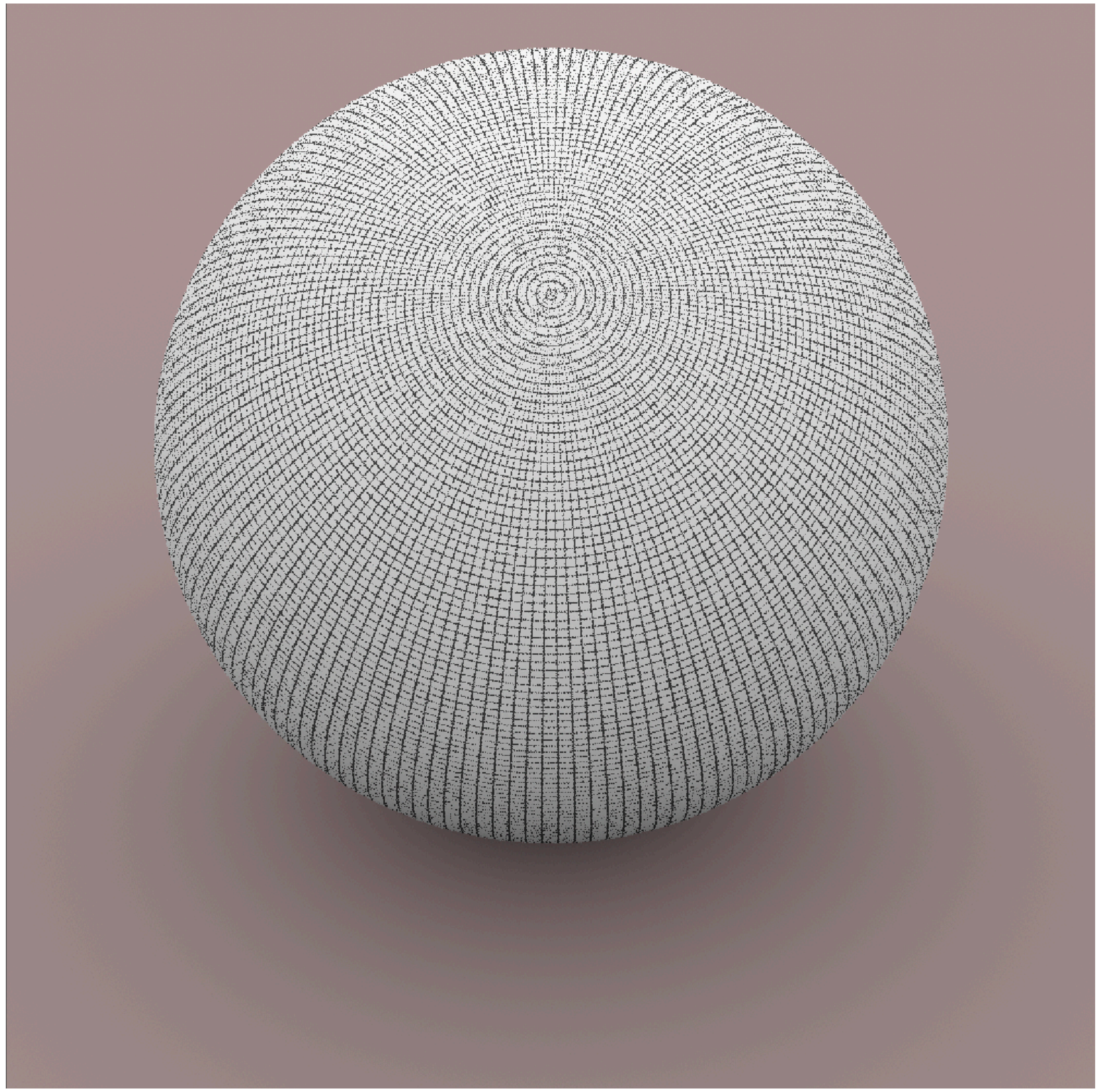
Texture mapping: pros and cons

- pros
  - different sampling rates for geometry and color
  - much easier to filter
- cons
  - uv mapping is hard

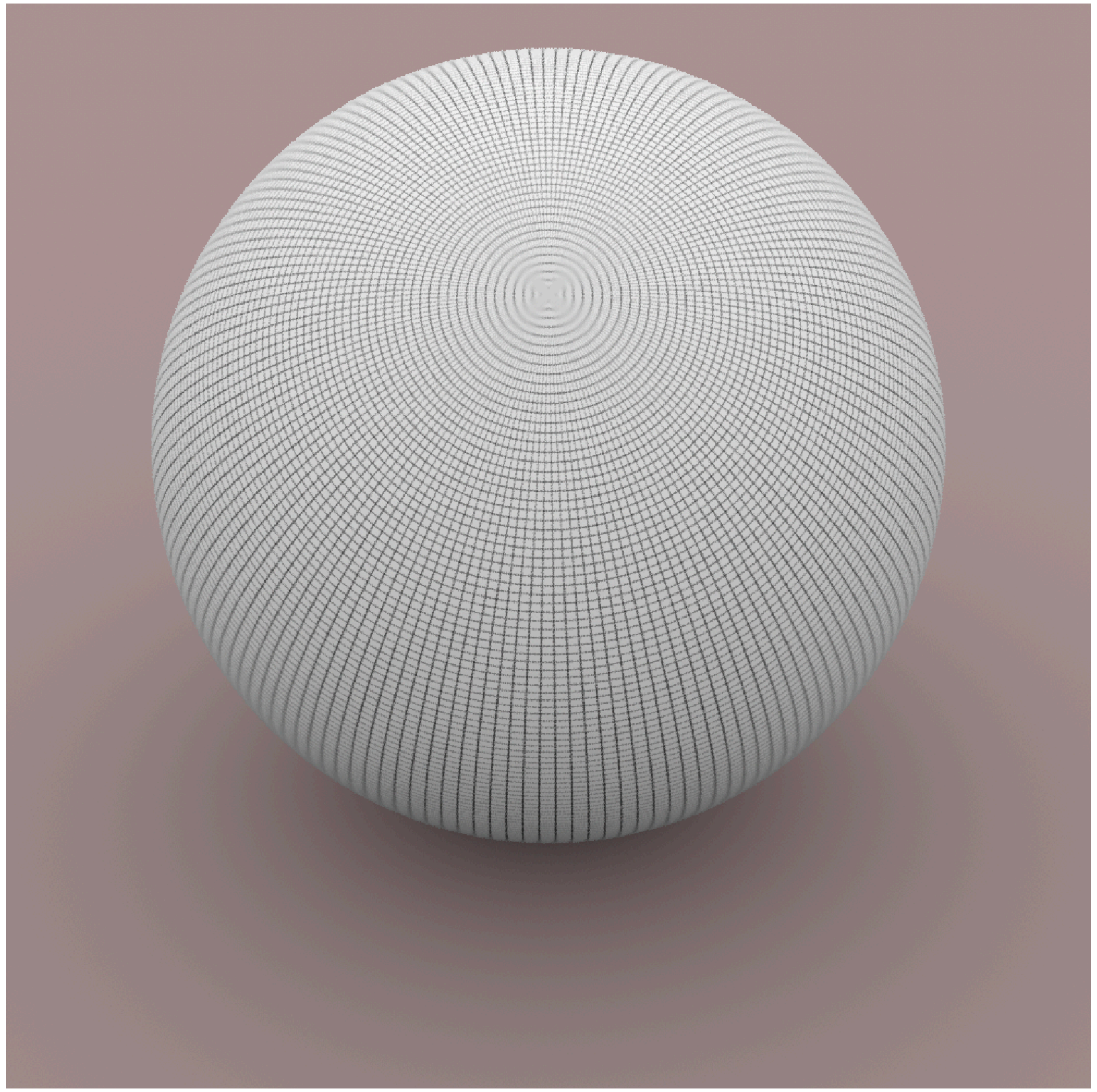
A pixel can cover a large region in the texture



Fail to account for all texels in the region can lead to noise/aliasing



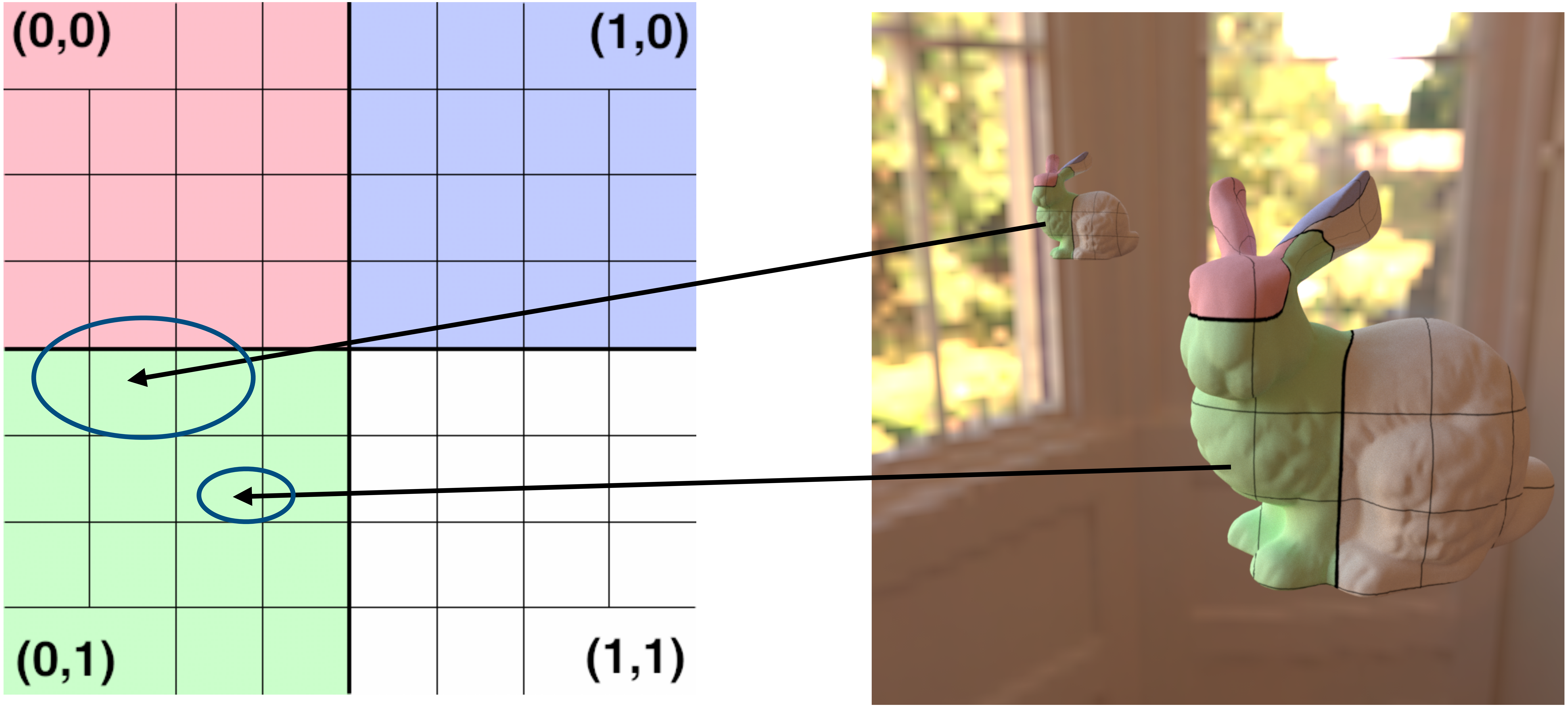
no filtering



with filtering

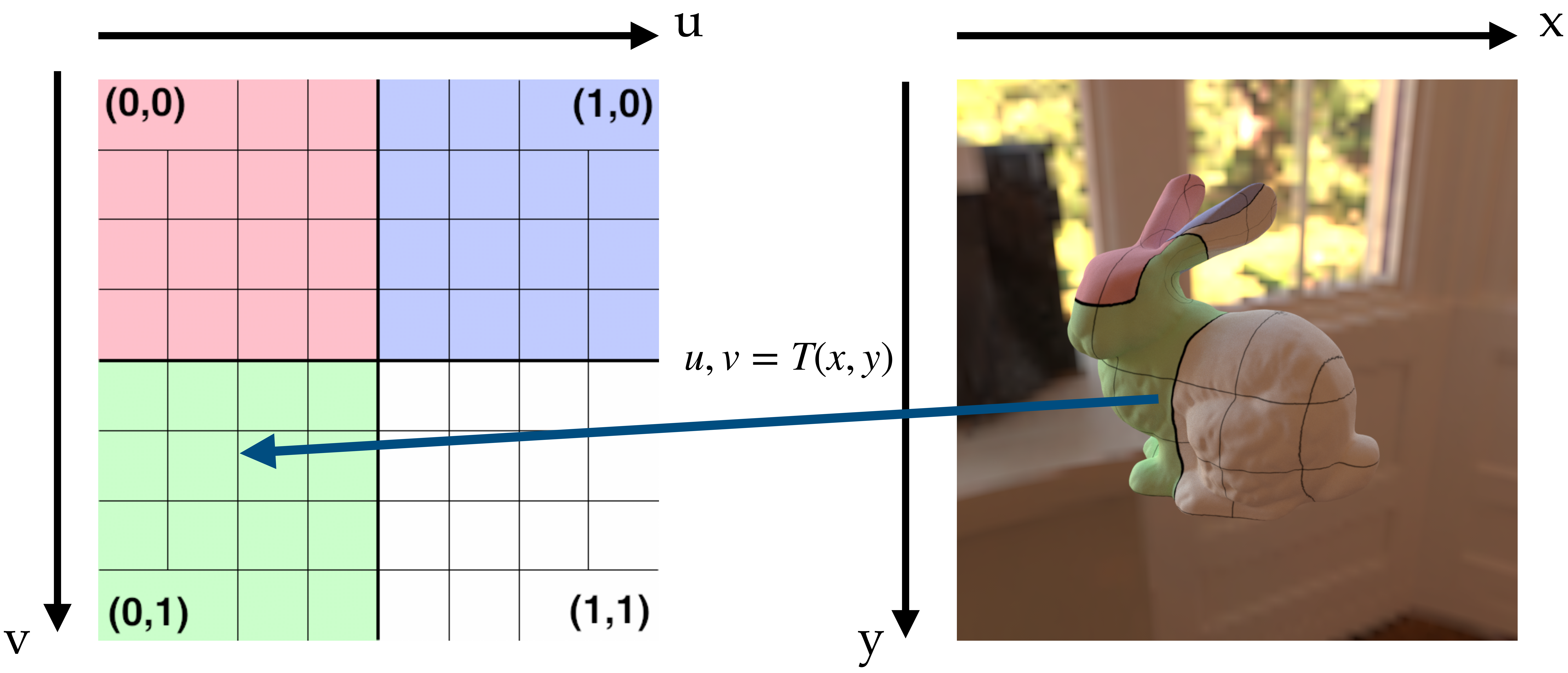
example from pbrt-v3 https://www.pbr-book.org/3ed-2018/Texture

Goal: average all texture values inside the region



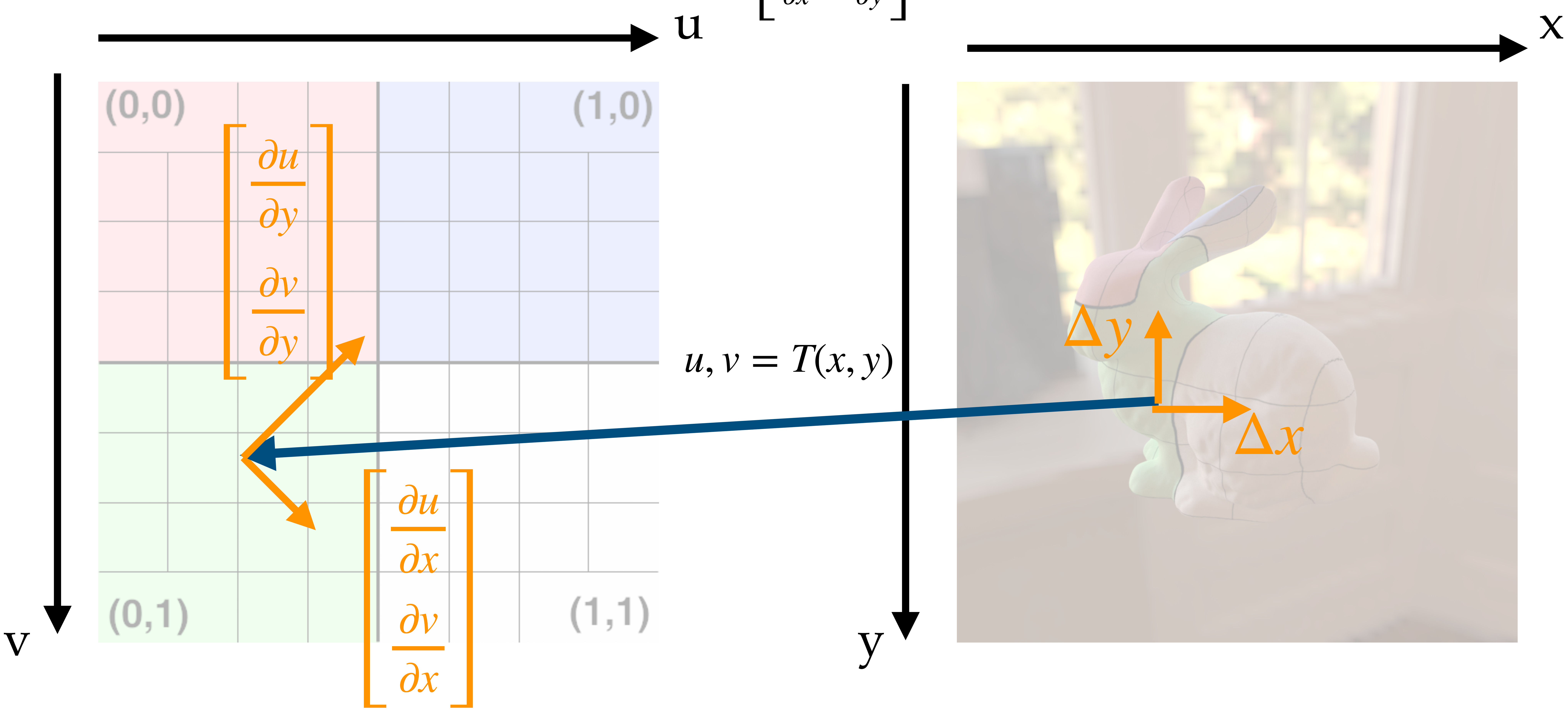
The filtering region is determined by the mapping between image space and texture space

problem: T can be complicated



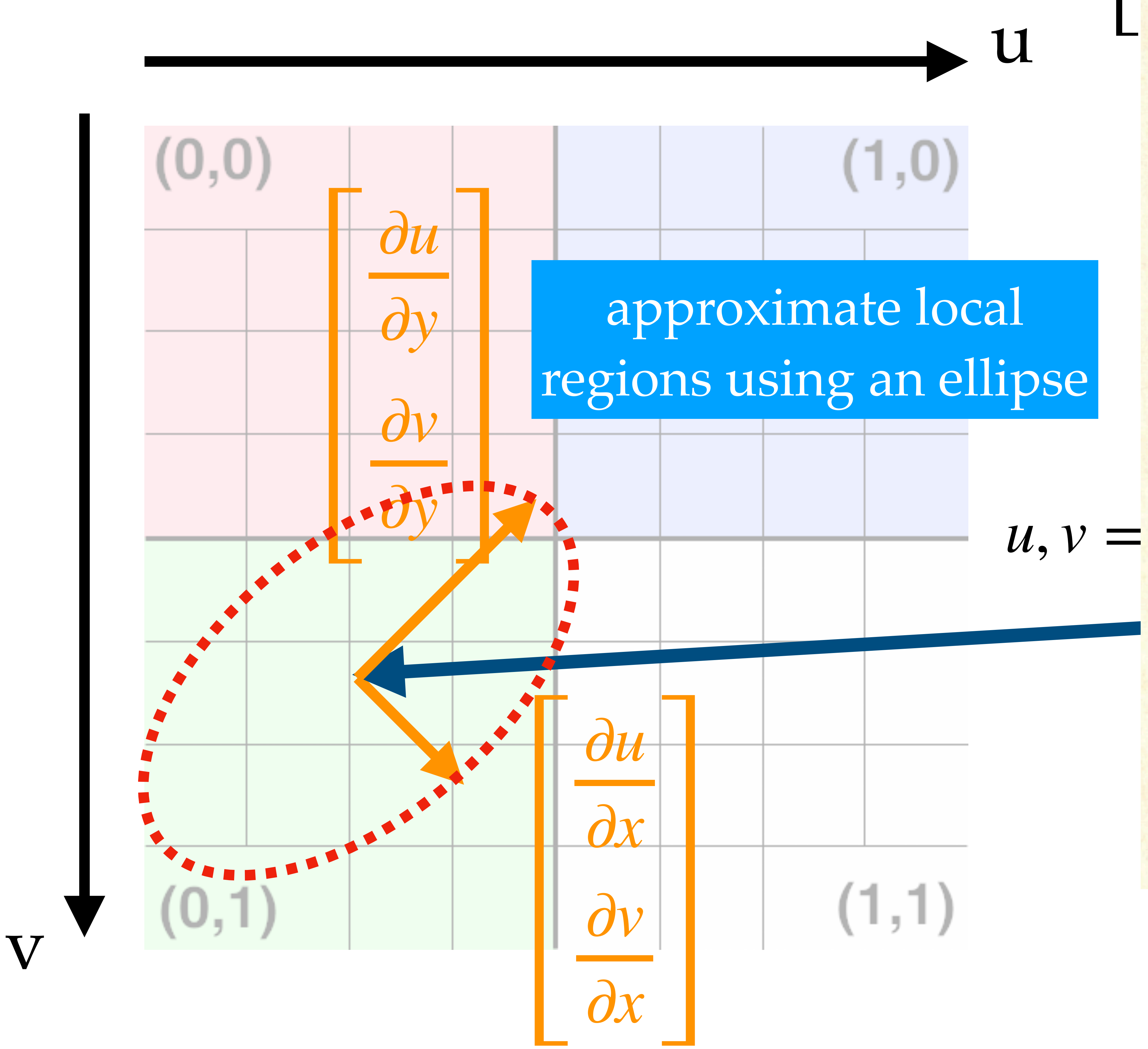
We can approximate the local mapping using first-order Taylor expansion

$$u, v \approx T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$



We can approximate the local mapping using first-order Taylor expansion

$$u, v \approx T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$



$u, v =$

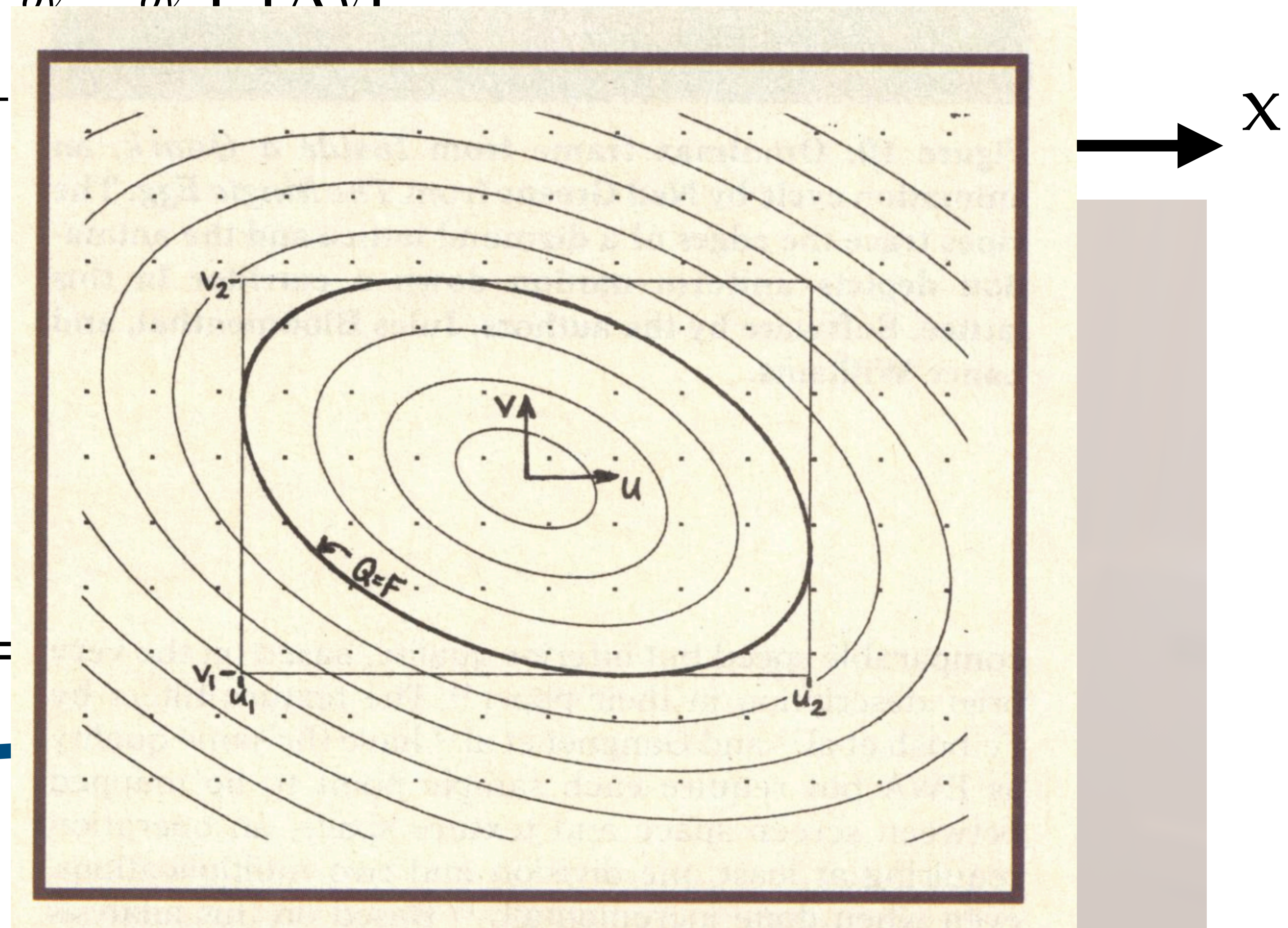
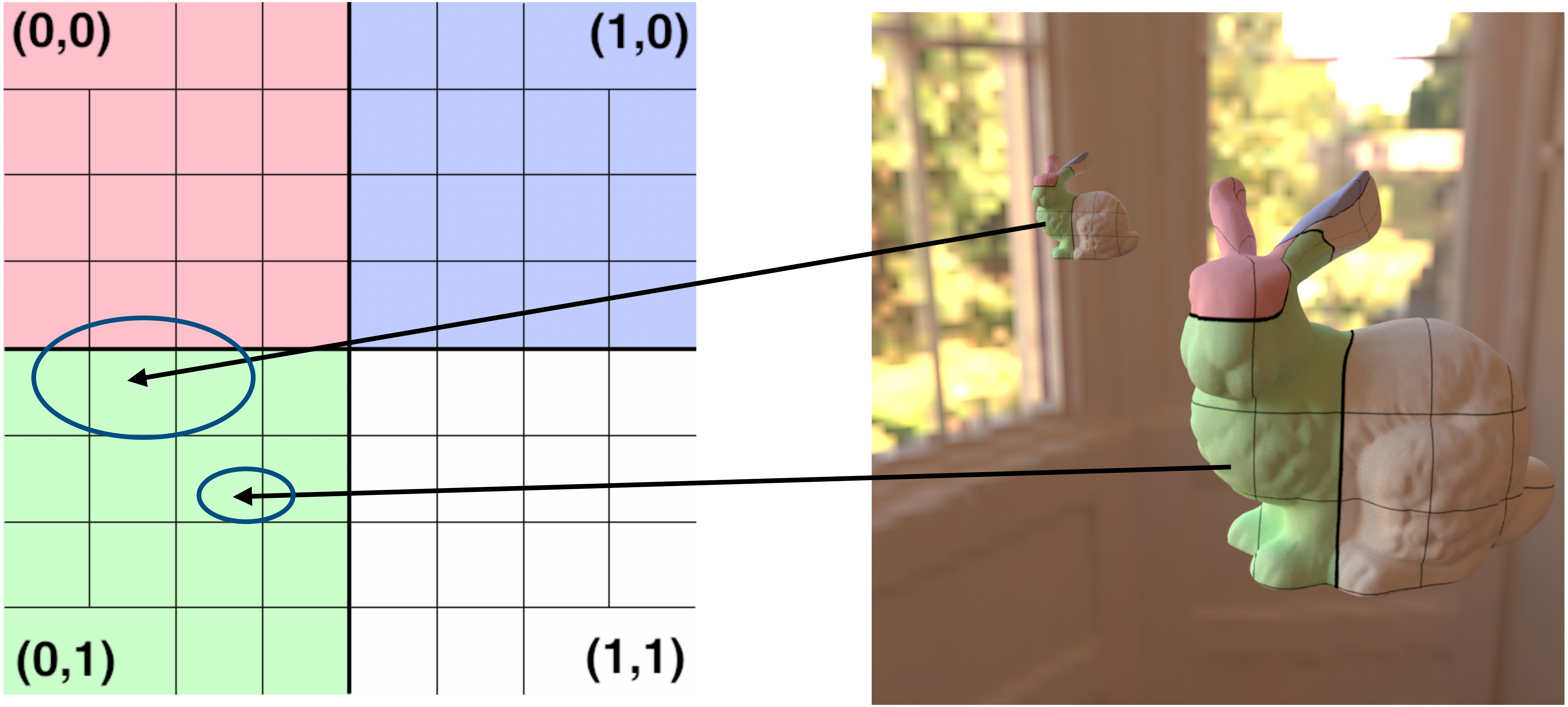


Figure 8. Contours of elliptical paraboloid Q and box around $Q = F$. Dots are centers of texture space pixels.

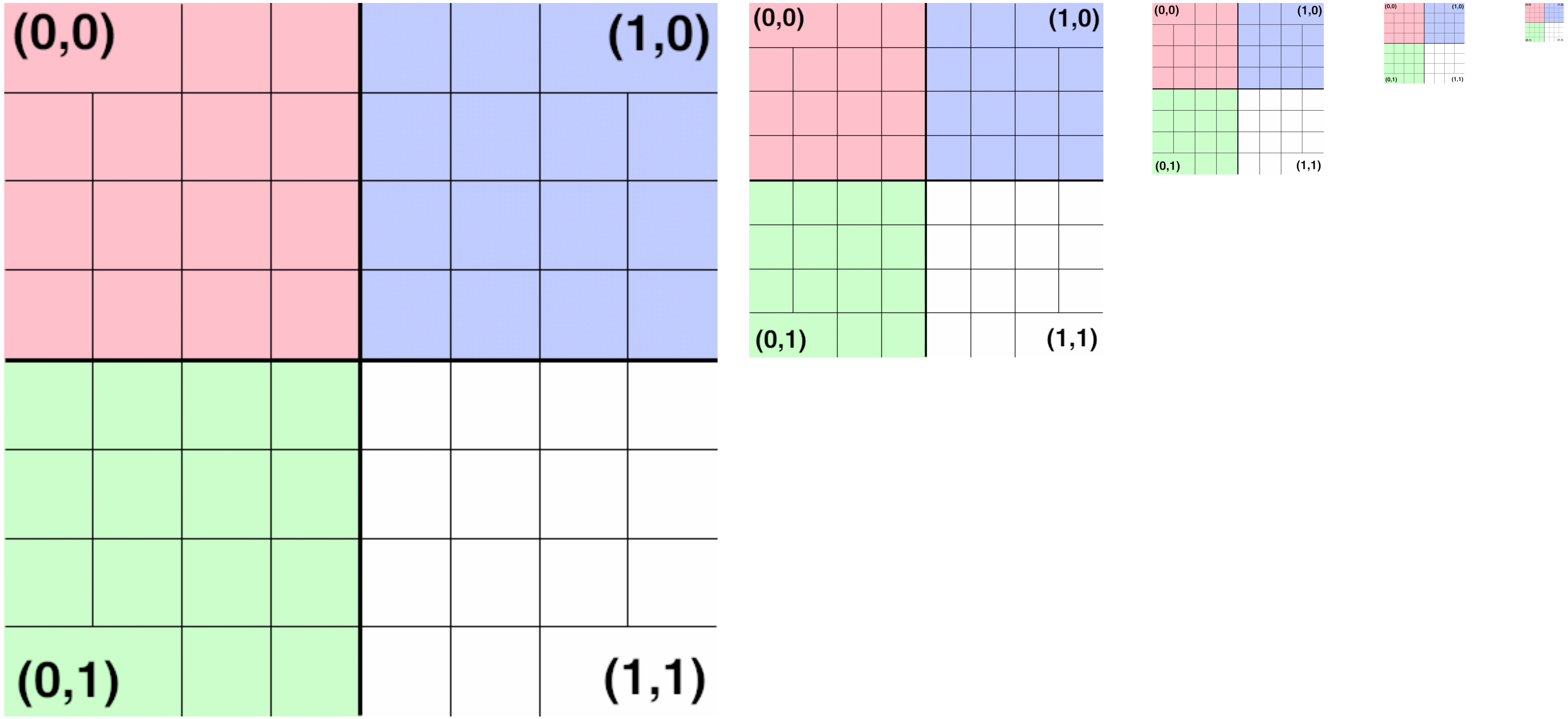
"Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter", Greene and Heckbert 1986

Goal: average all texture values inside the region

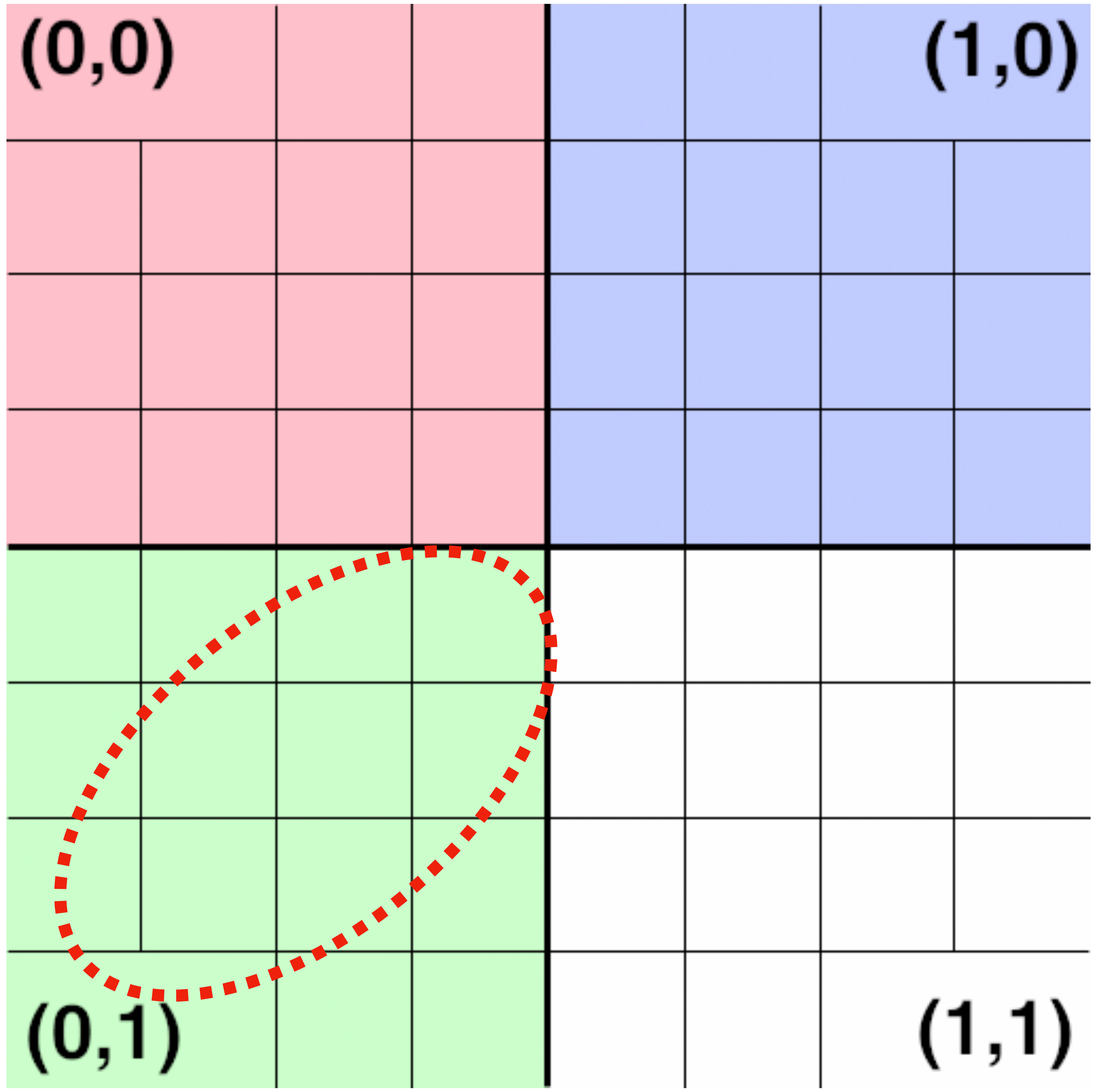


Downsample the texture for fast average

- usually called "mipmapping"

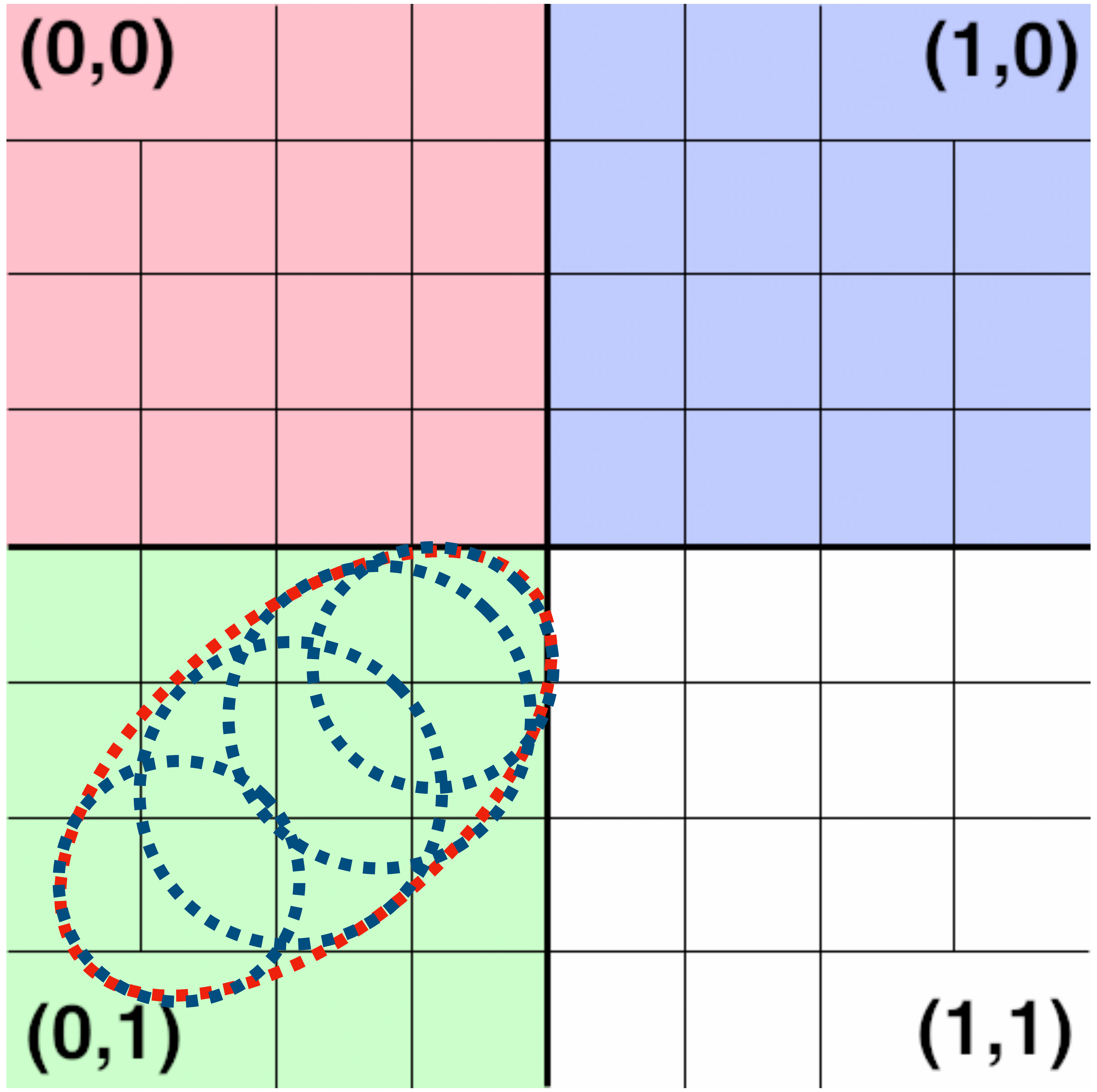


Elliptical weighted averaging algorithm



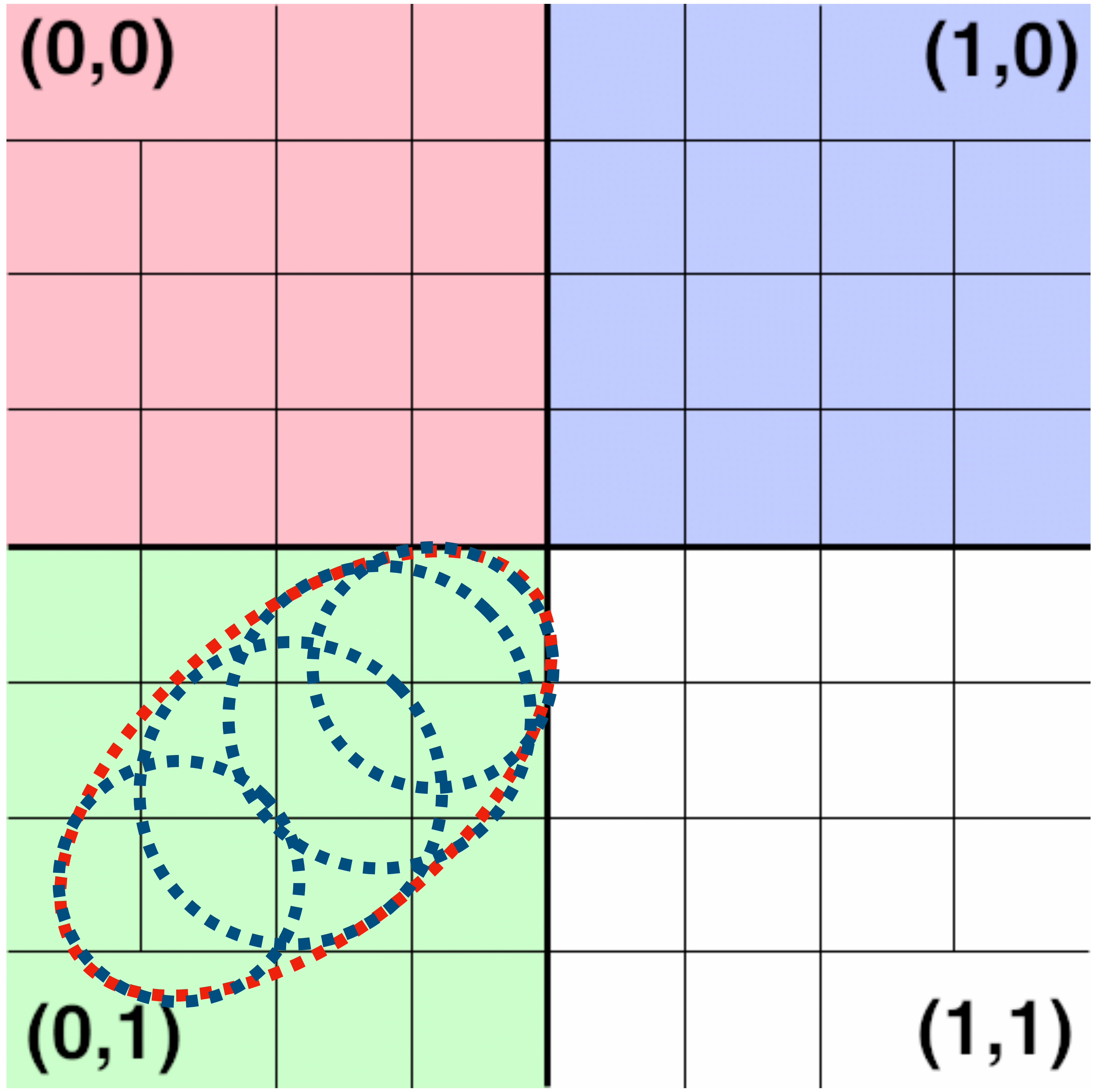
Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

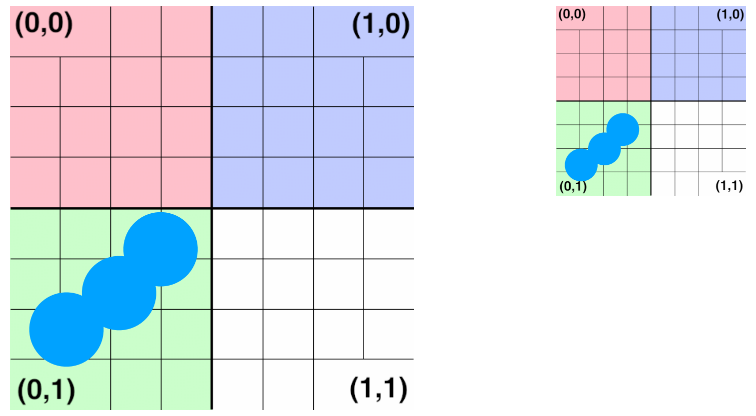


Elliptical weighted averaging algorithm

1. approximate the ellipse with circles

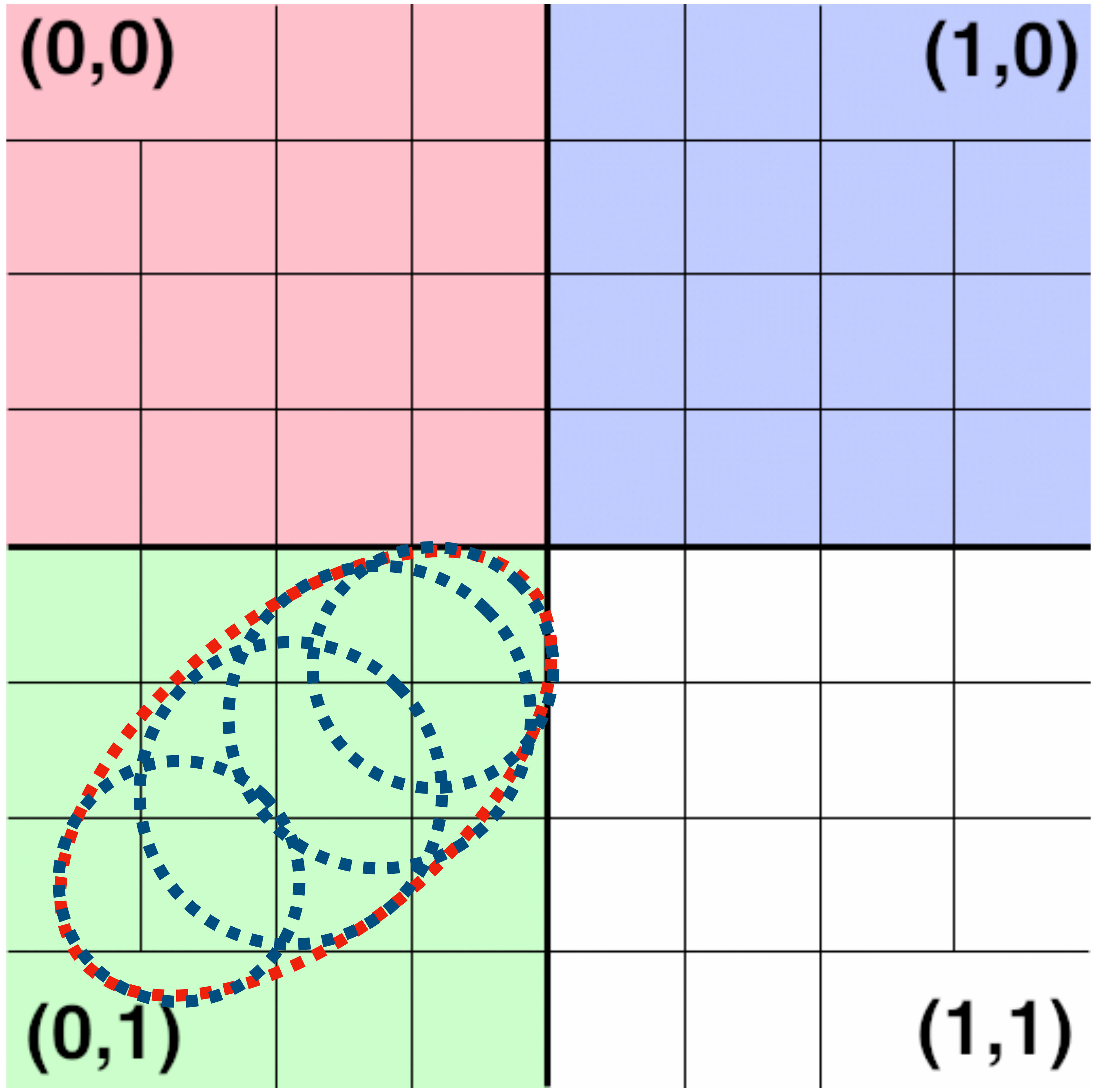


2. find two appropriate mipmap levels s.t. each circle maps to ~1 texel

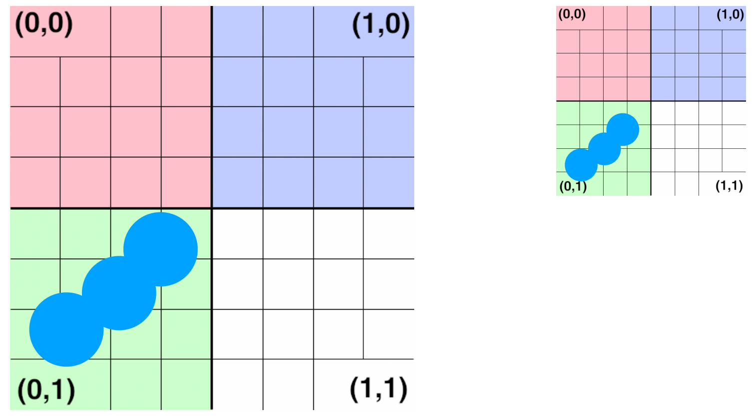


Elliptical weighted averaging algorithm

1. approximate the ellipse with circles



2. find two appropriate mipmap levels s.t. each circle maps to ~1 texel



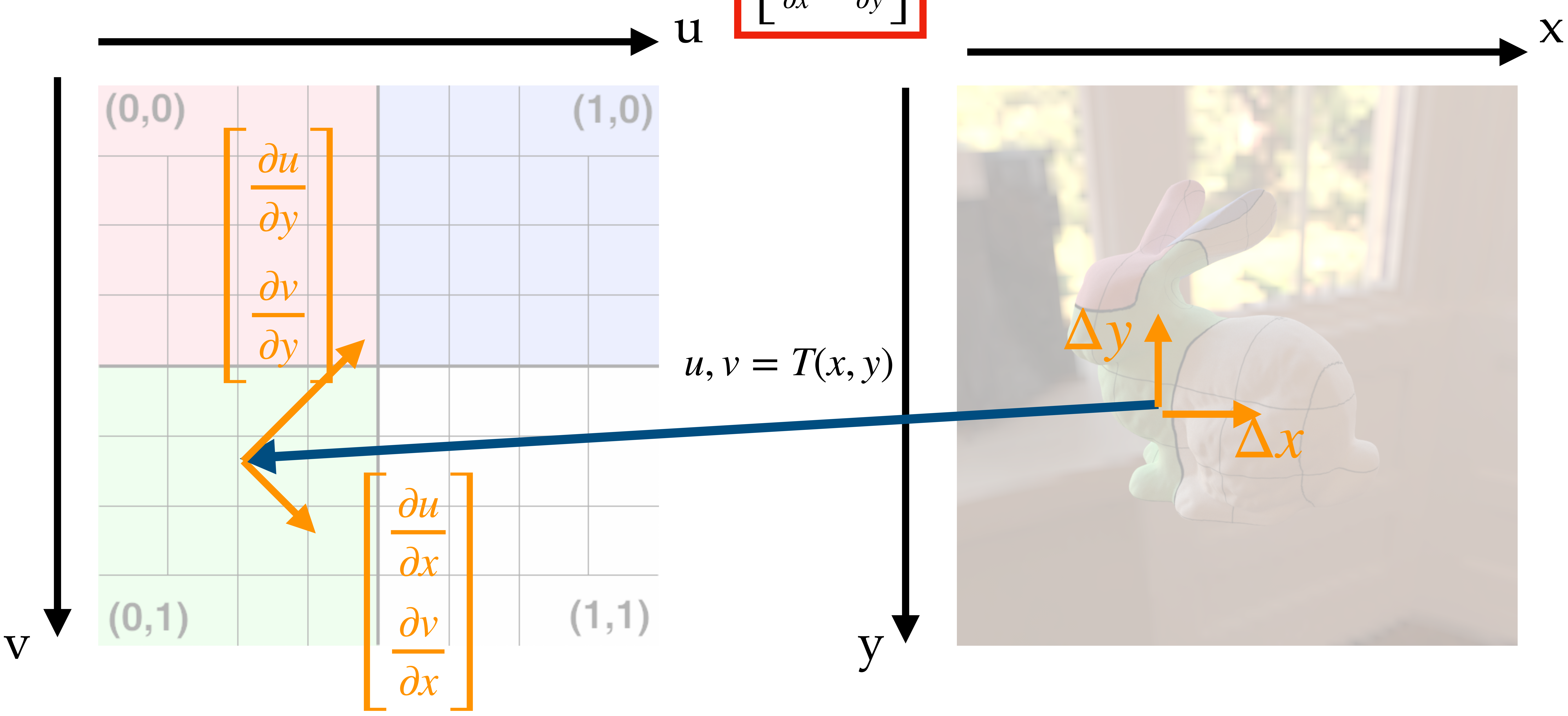
3. linearly interpolate between pixels and mipmap levels

Modern video games/renderers use EWA filtering!

https://www.techspot.com/review/336-cod-black-ops-performance/page3.html

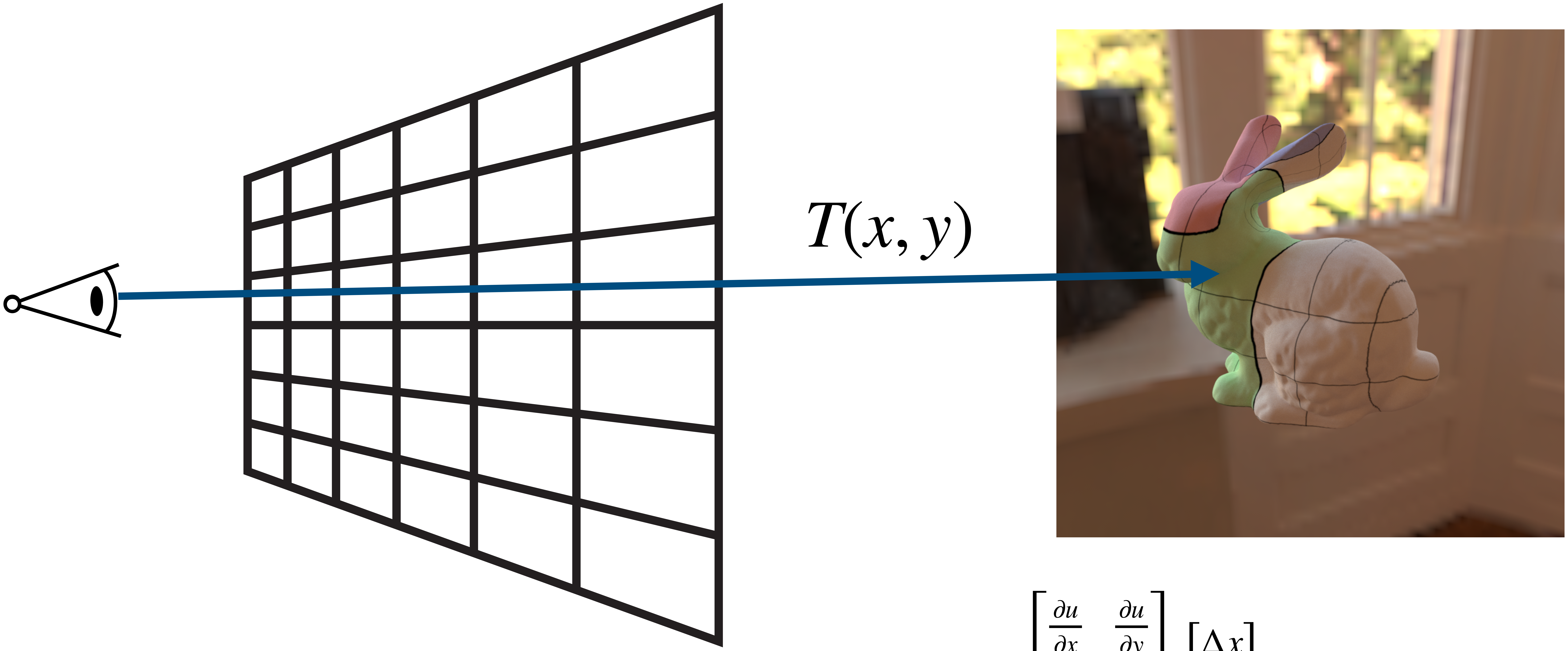
How do we obtain the derivatives?

$$u, v = T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$



The mapping T is the ray casting function

- so we can simply apply chain rule and differentiate ray casting



$$u, v = T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

The mapping T is the ray casting function

- so we can simply apply chain rule and differentiate ray casting

Tracing Ray Differentials

Homan Igehy

Computer Science Department

Stanford University

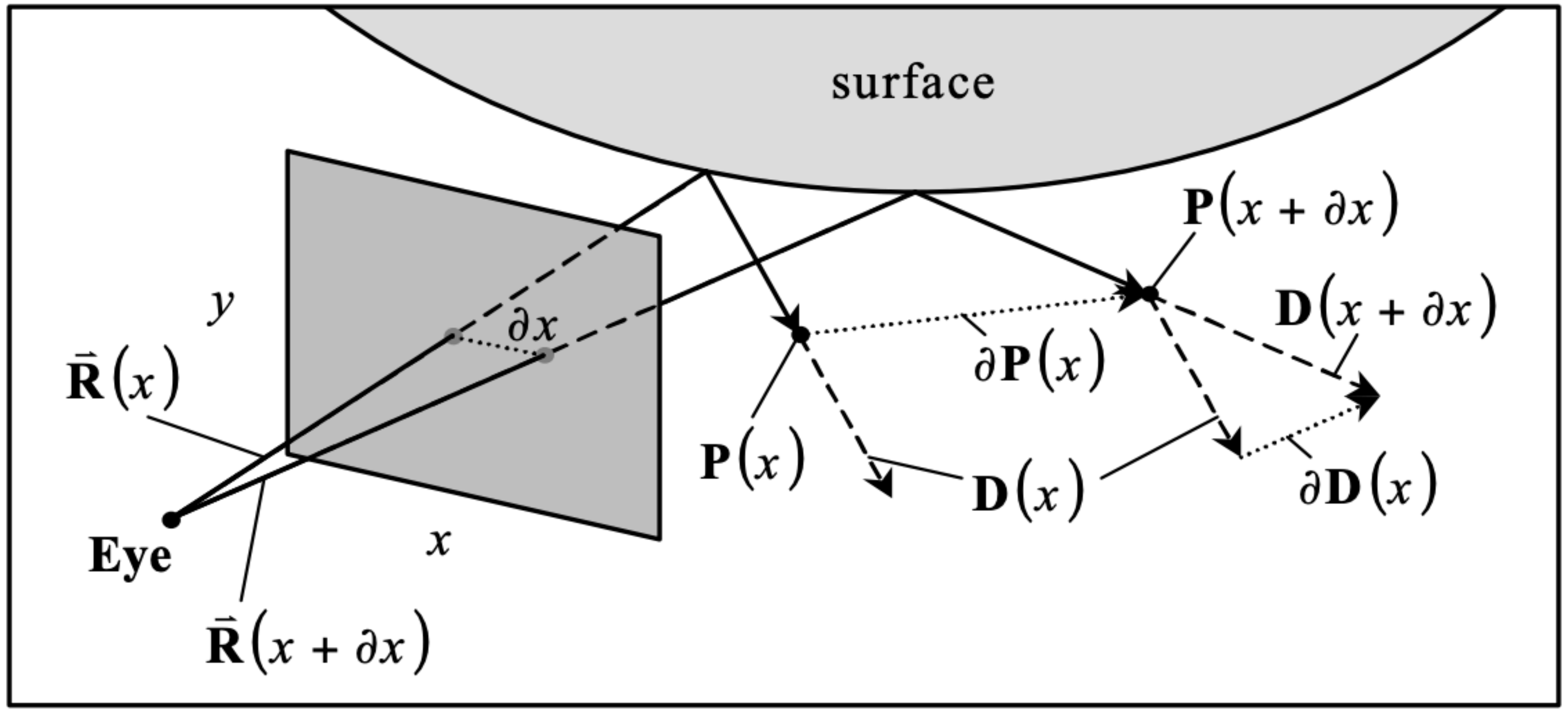


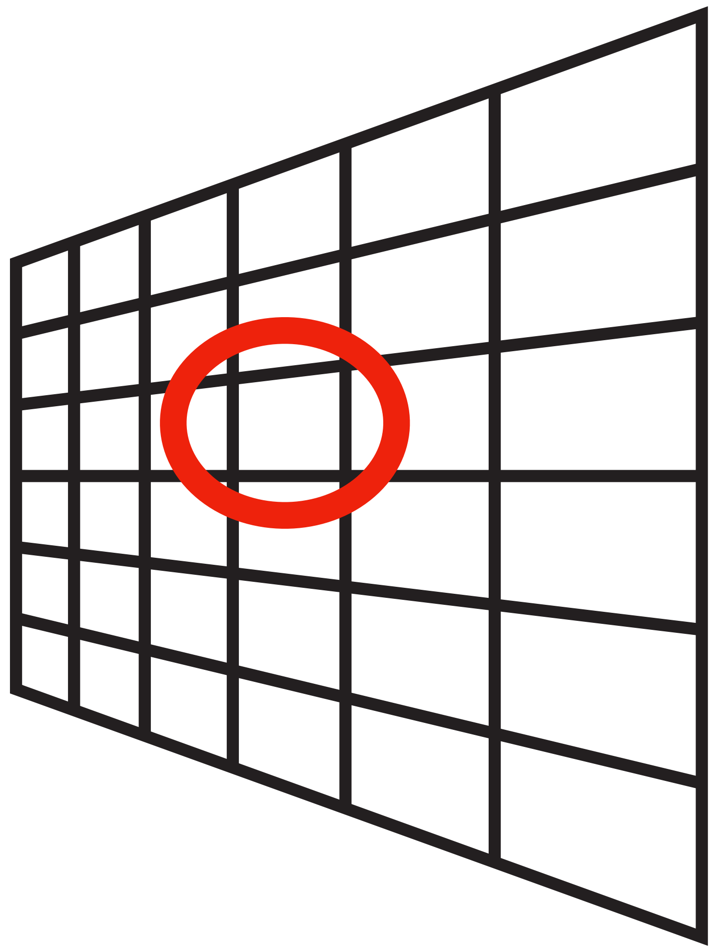
Figure 1: A Ray Differential. The diagram above illustrates the positions and directions of a ray and a differentially offset ray after a reflection. The difference between these positions and directions represents a ray differential.

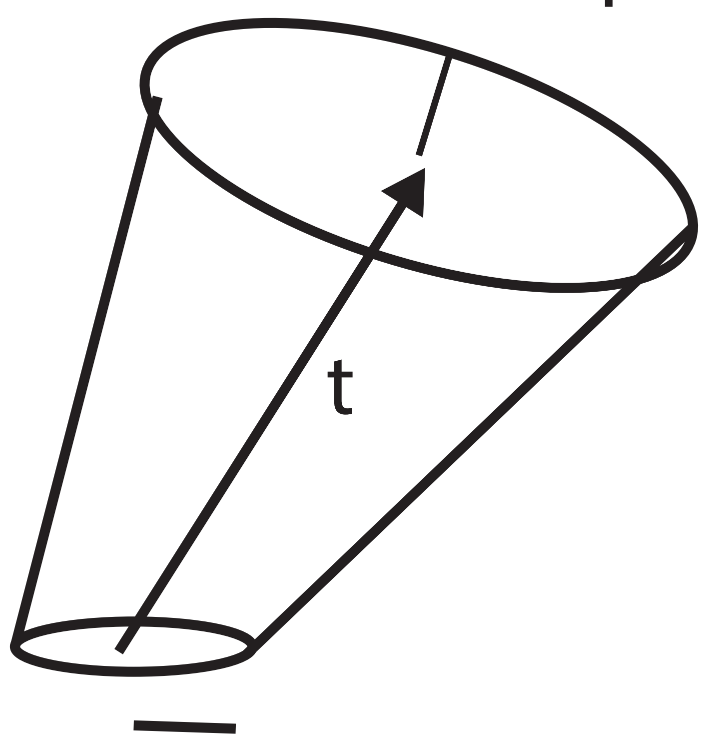
$$u, v = T(x_0, y_0) + \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

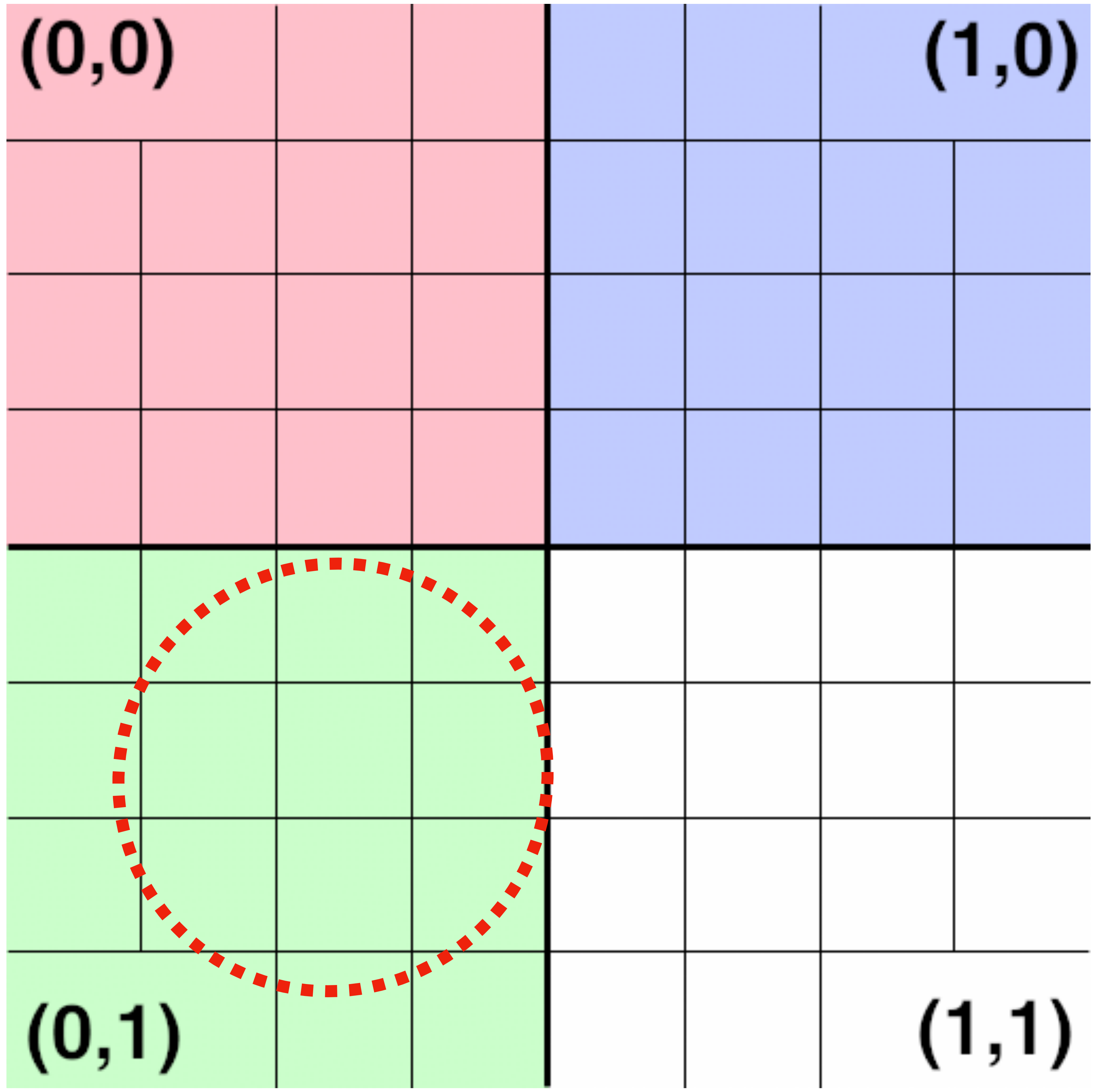
Lajolla uses a heuristic method to approximate the ellipse with a circle

- see homework 0 for more details and the reasoning

initial spread = (pixel size) / 4







more about texture filtering next Monday!

https://github.com/BachiLi/lajolla_public/blob/main/src/ray.h
https://github.com/BachiLi/lajolla_public/blob/main/src/texture.h

Texture mapping: pros and cons

- pros
  - different sampling rates for geometry and color
  - much easier to filter
- cons
  - uv mapping is hard

In movie production: use UV generated by mesh subdivision

- will talk more about this in the later lectures

Ptex: Per-Face Texture Mapping for Production Rendering

Brent Burley[†1] and Dylan Lacewell[‡1,2]

[1]Walt Disney Animation Studios
[2]University of Utah

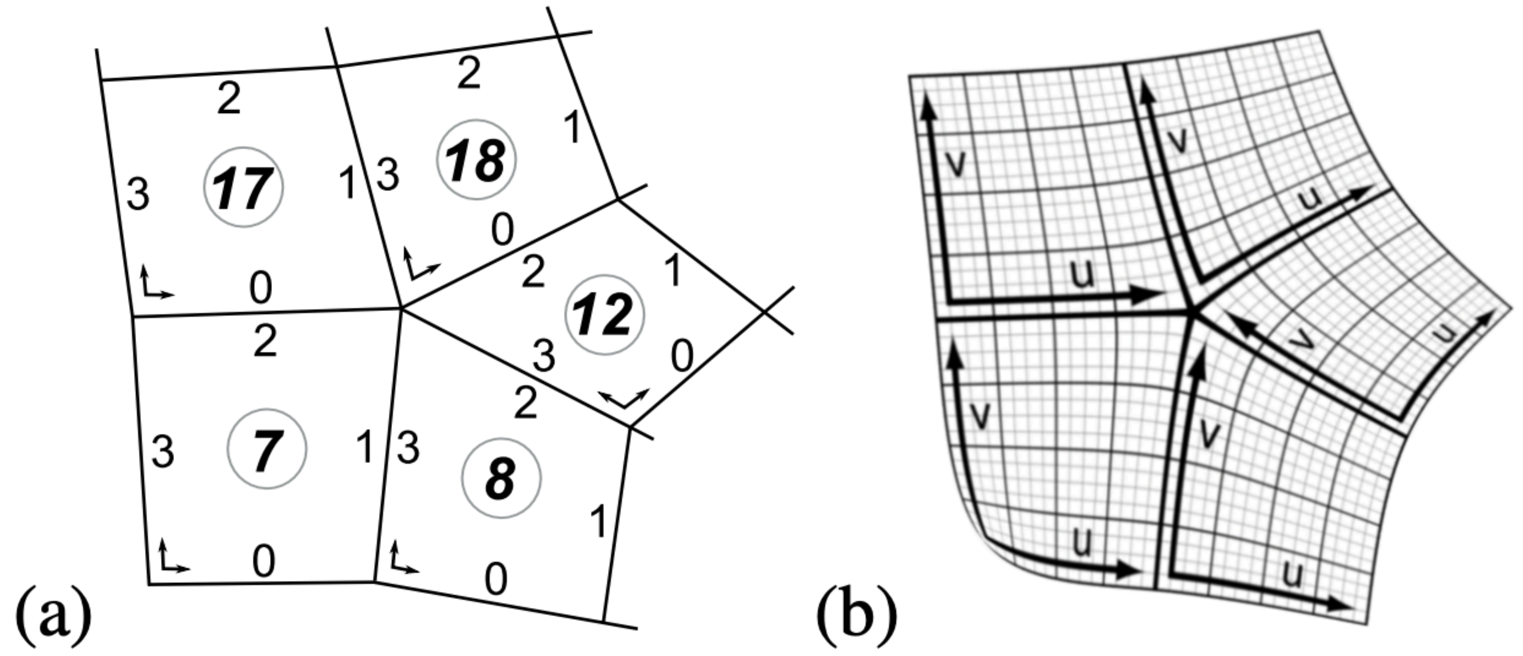


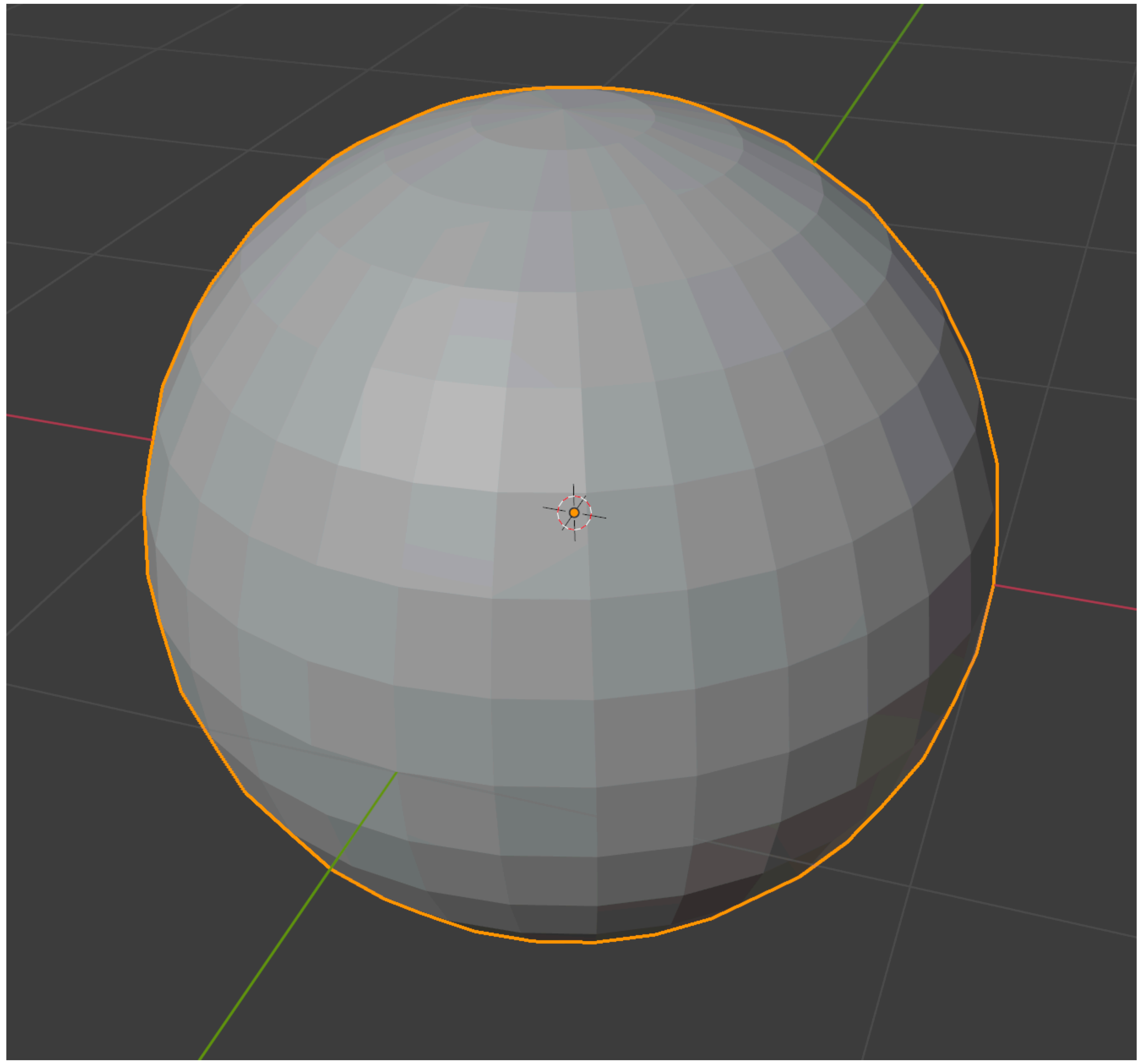
Figure 4: ***a)** Portion of a control mesh showing intrinsic faceids and edgeids. **b)** Corresponding limit surface showing continuous isolines across faces.*

https://www.youtube.com/watch?v=GxNlAlOuQQQ

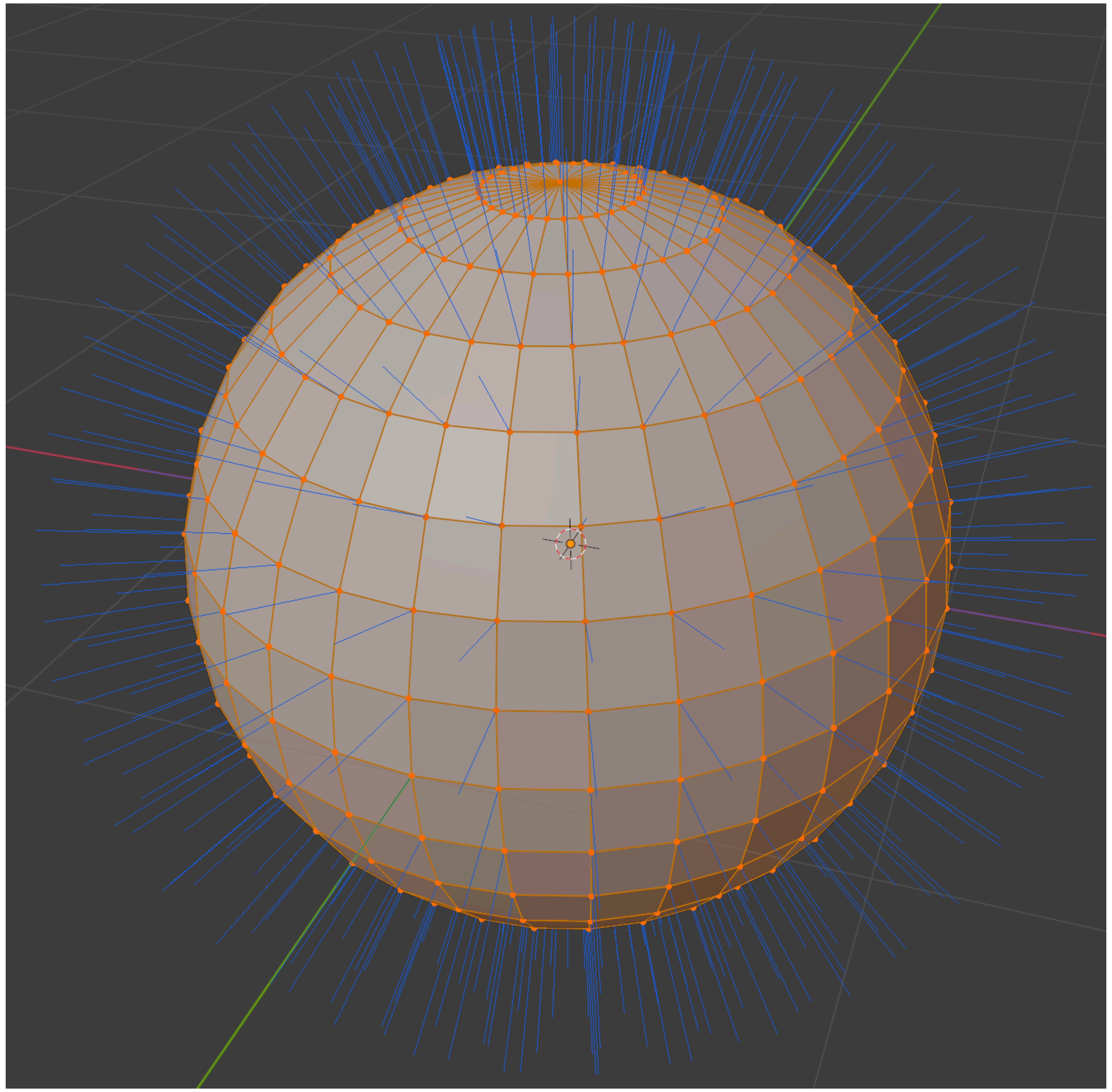
Shading normals

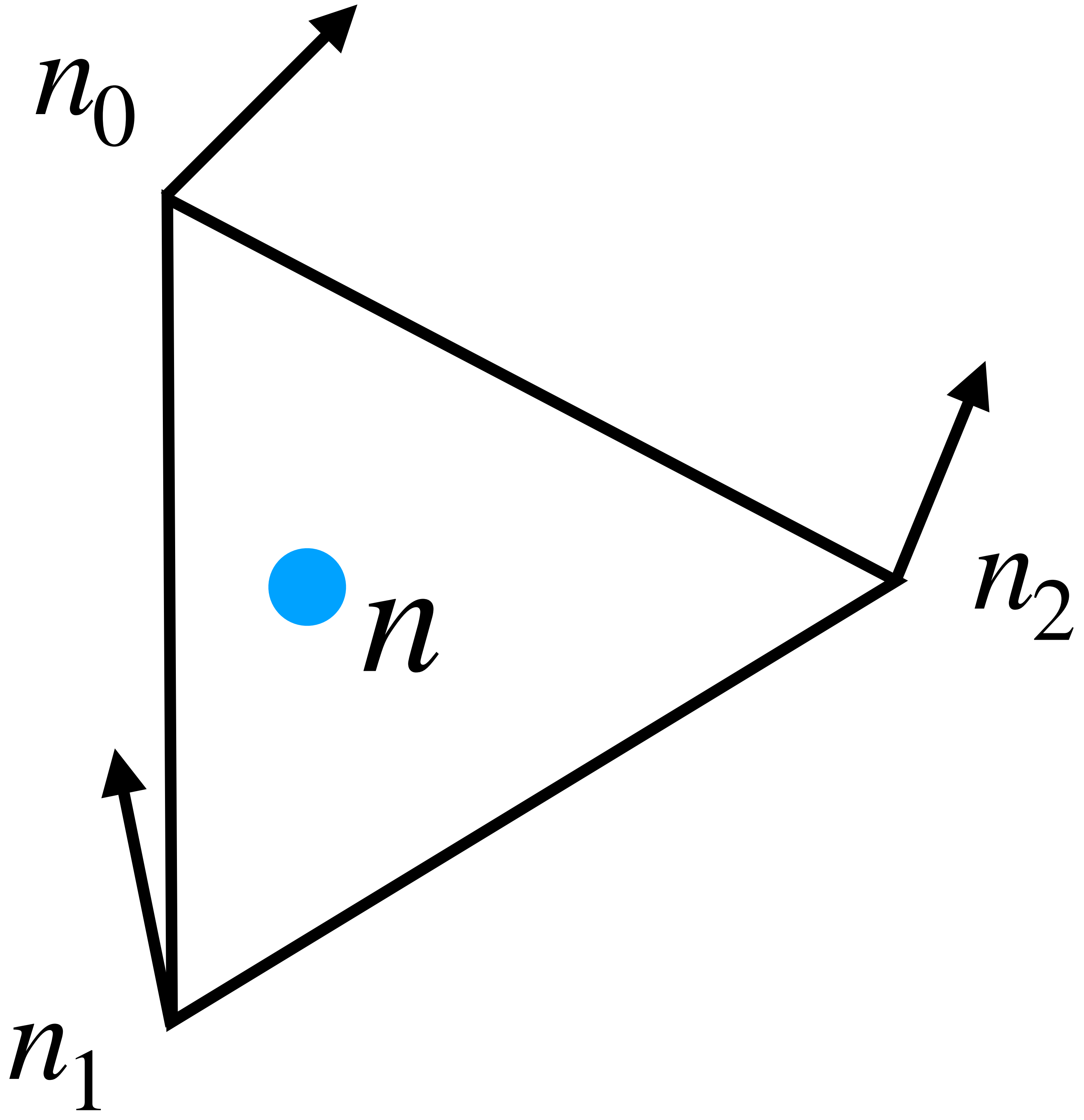
- triangle meshes are planar approximations of a smooth surface: can look faceted.

quiz: how would you solve this?

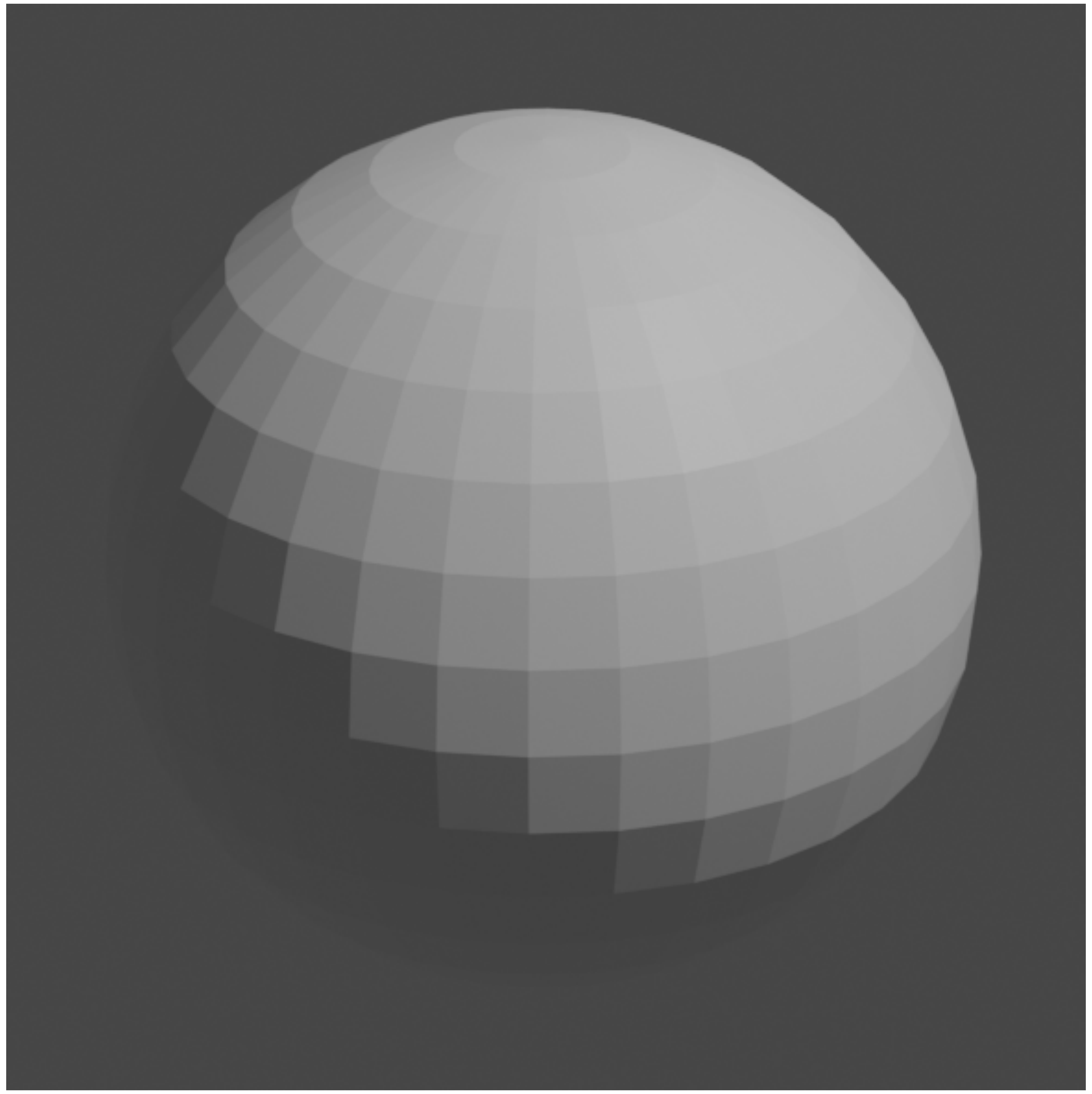


Trick: assign a normal per vertex, then interpolate

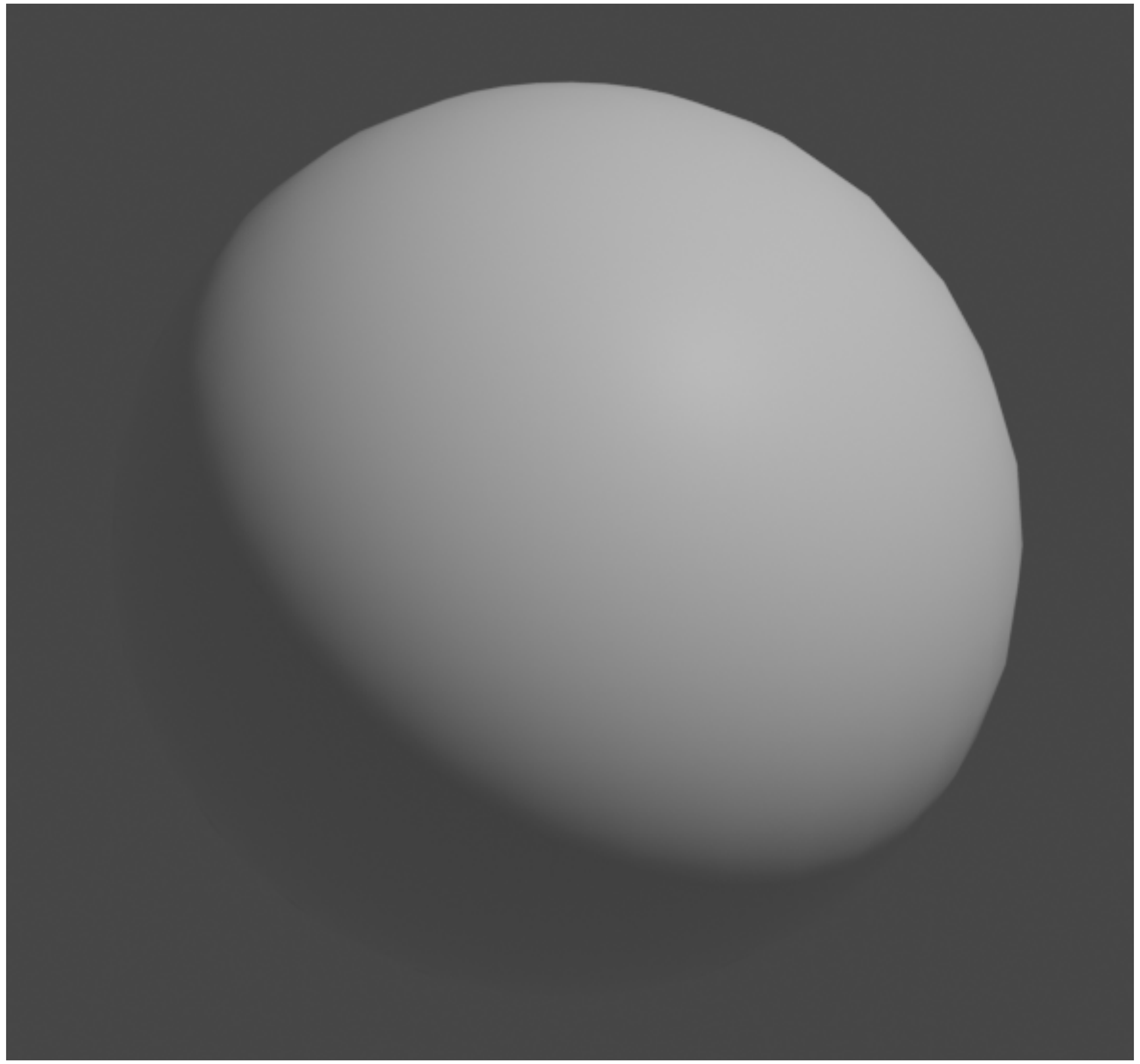




$$n = \text{normalize}\left((1 - b_1 - b_2)n_0 + b_1 n_1 + b_2 n_2\right)$$



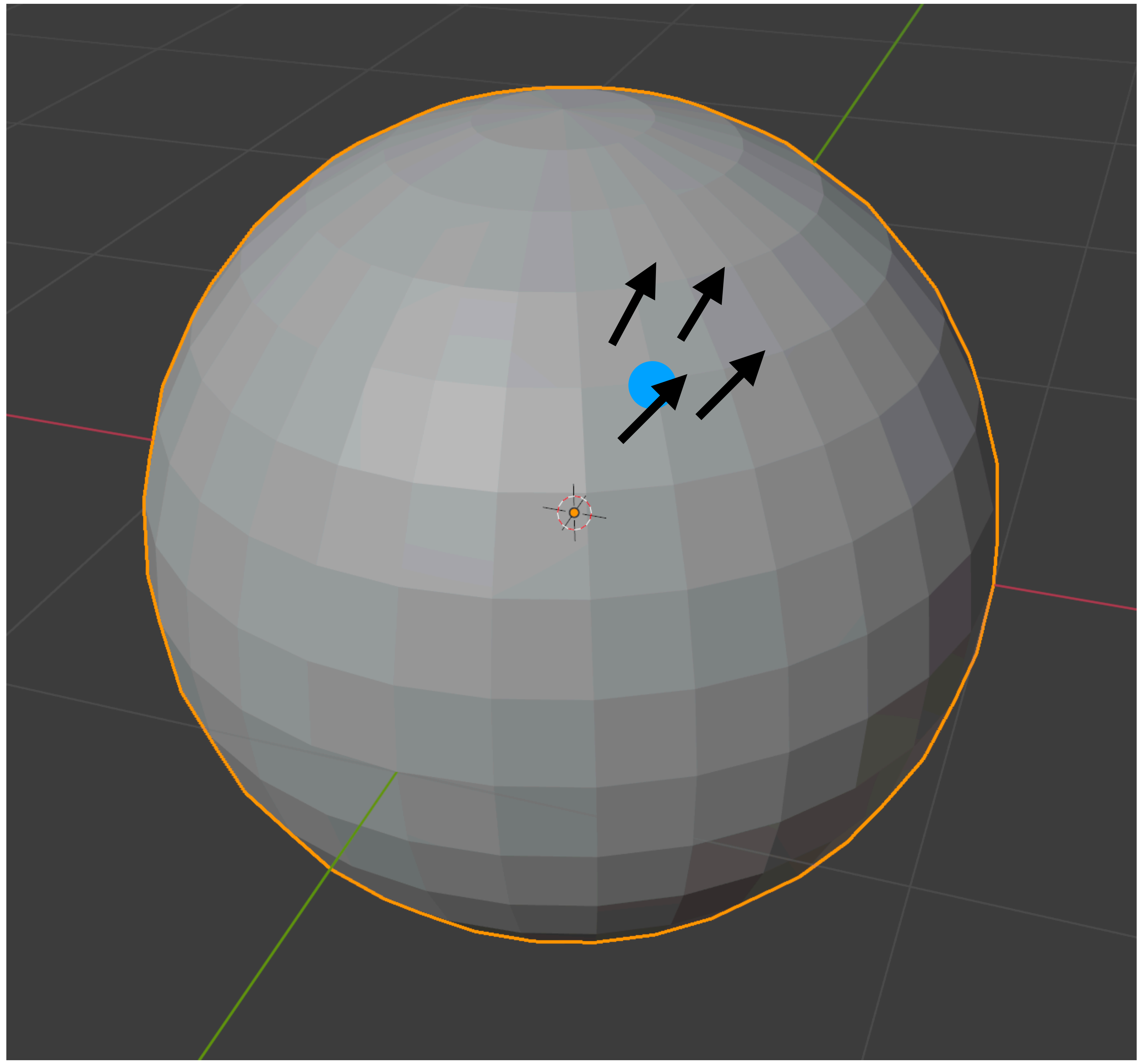
w/o per-vertex normal



w/ per-vertex normal

How to get vertex normal?

- weighted average of normals of nearby faces

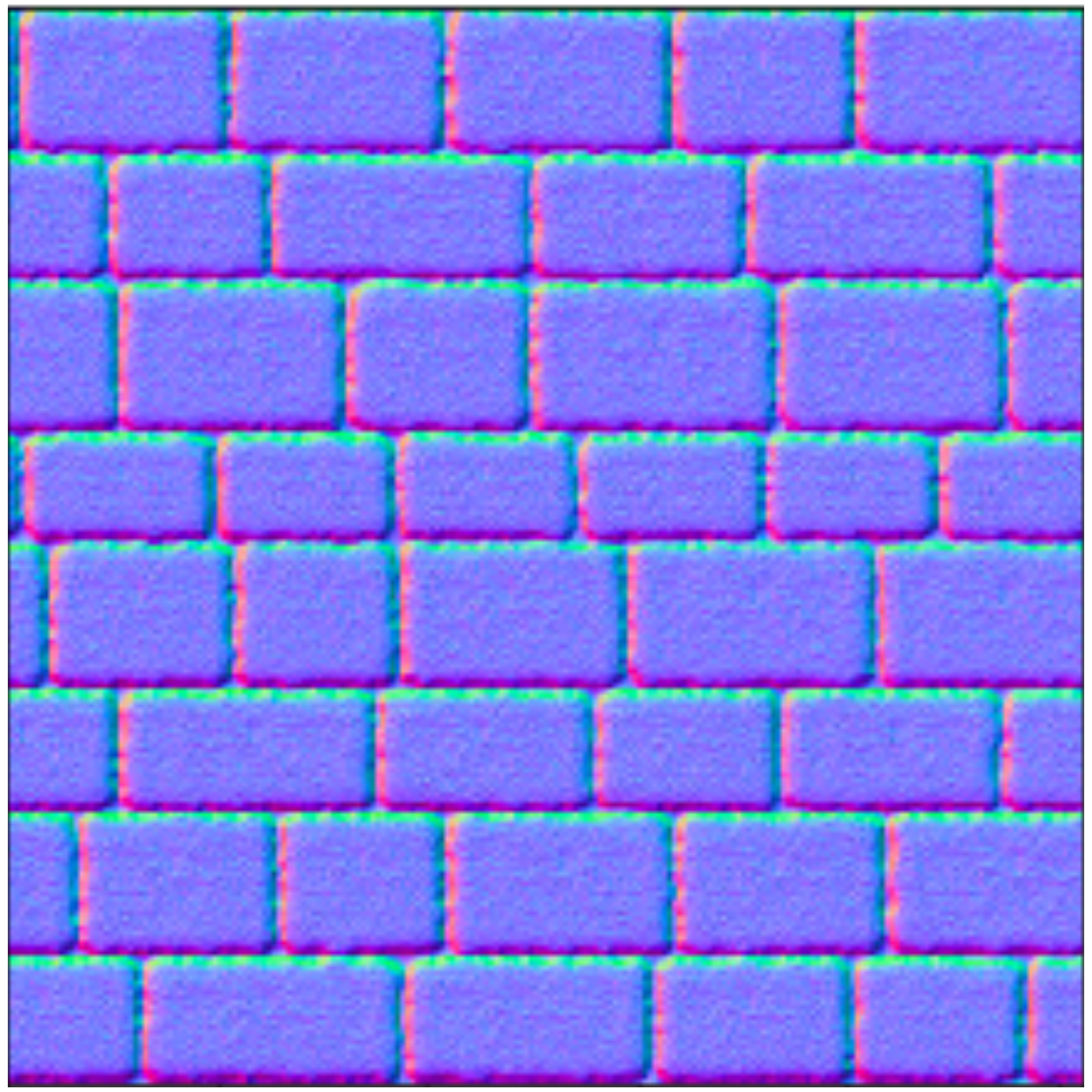


Weights for Computing Vertex Normals from Facet Normals

Nelson Max

Lawrence Livermore National Laboratory

Alternative: normal mapping



Discrepancies between shading normal and real geometry can lead to artifacts

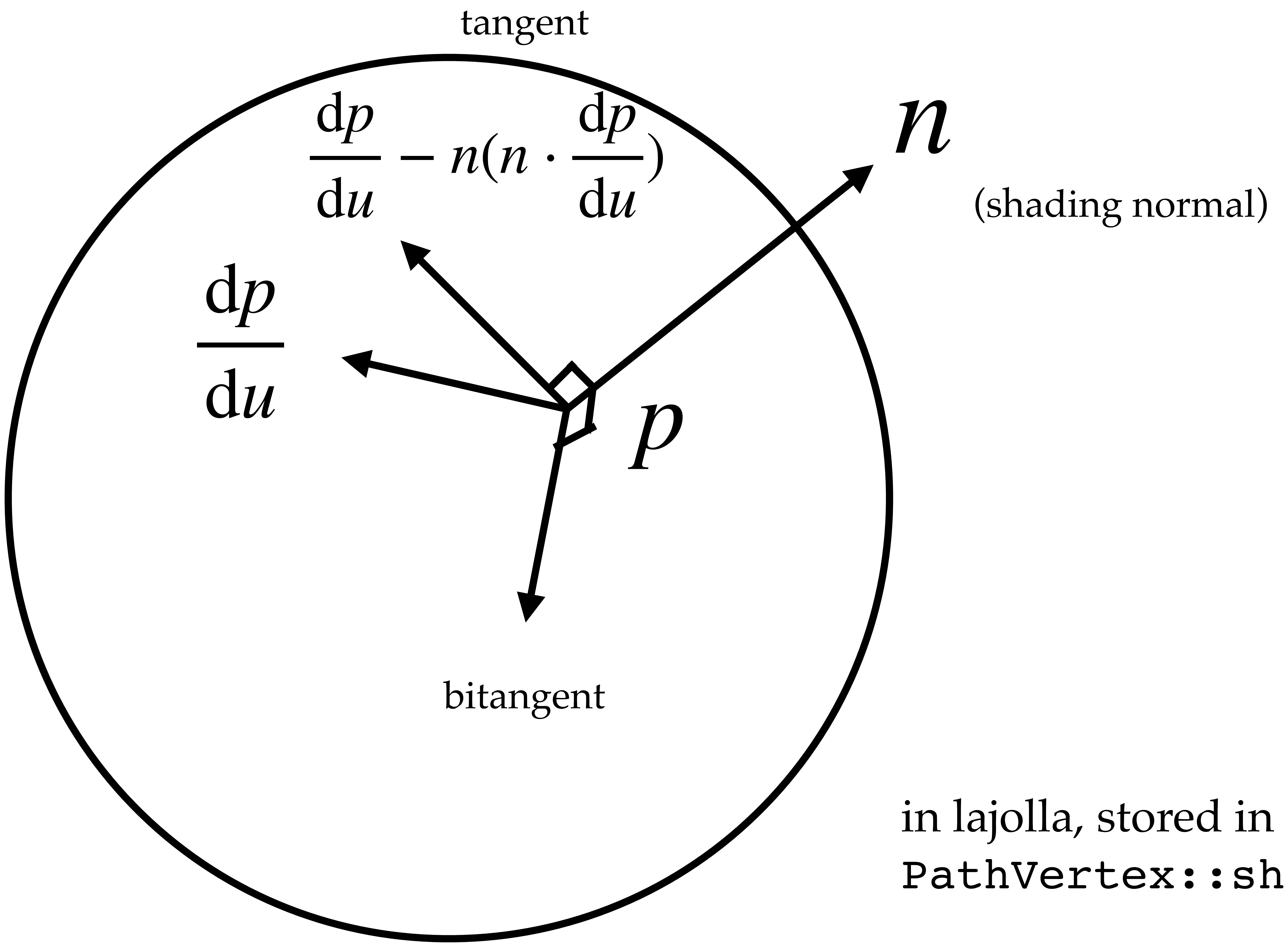
From "Hacking the Shadow Terminator", Johannes Hanika
https://jo.dreggn.org/home/2021_terminator.pdf

Discrepancies between shading normal and real geometry can lead to artifacts

Lajolla doesn’t implement this!

From “Hacking the Shadow Terminator”, Johannes Hanika
https://jo.dreggn.org/home/2021_terminator.pdf

For shading, we need a local coordinate basis



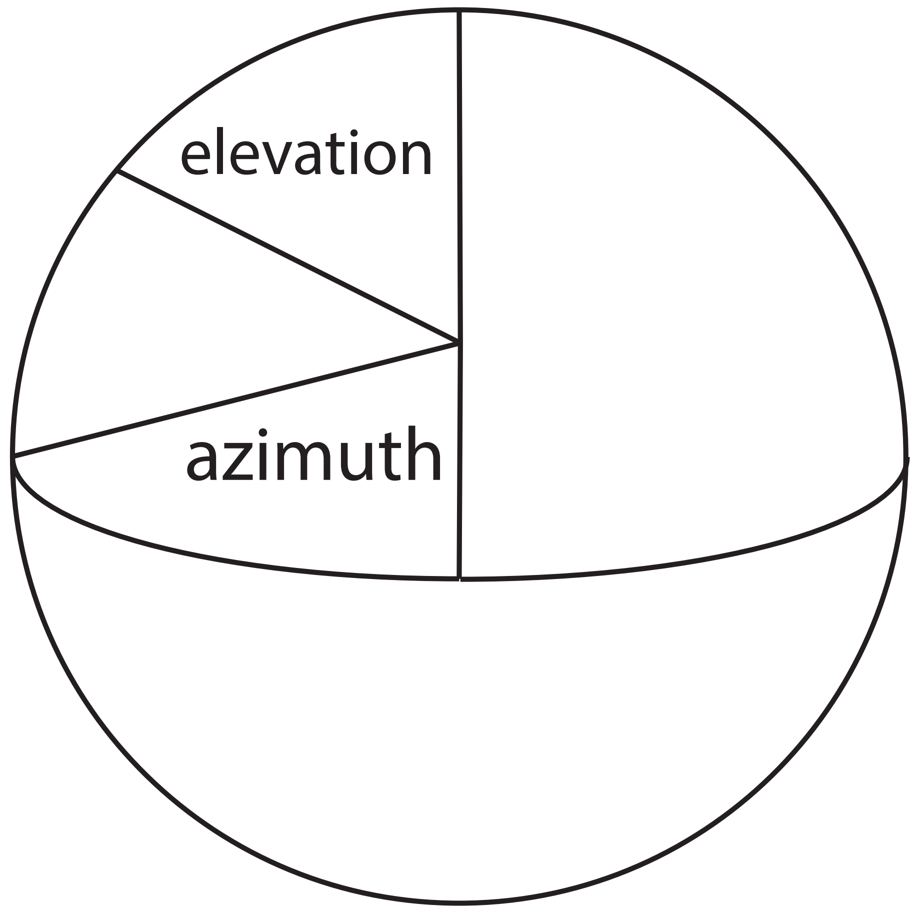
in lajolla, stored in `PathVertex::shading_frame`

In lajolla

```
struct PathVertex {
    // …
    Vector3 geometry_normal; // always face at the same direction at shading_frame.n
    Frame shading_frame;
    // …
};
```

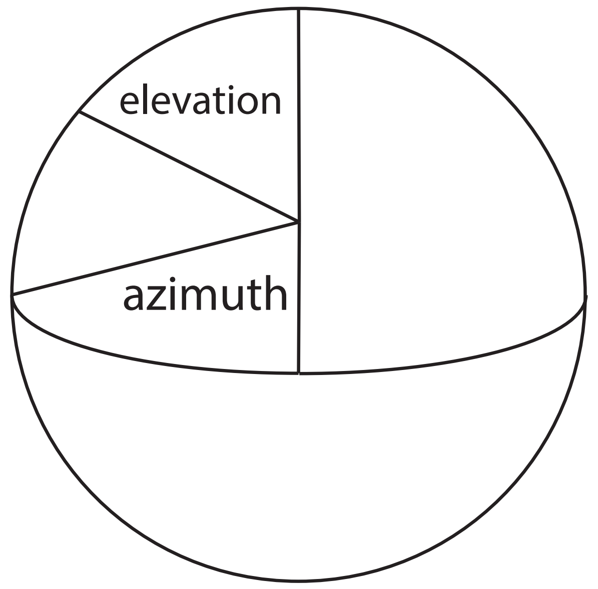
Environment maps

- an infinitely far area light source represented as an image



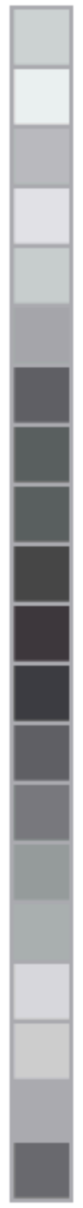
Environment maps

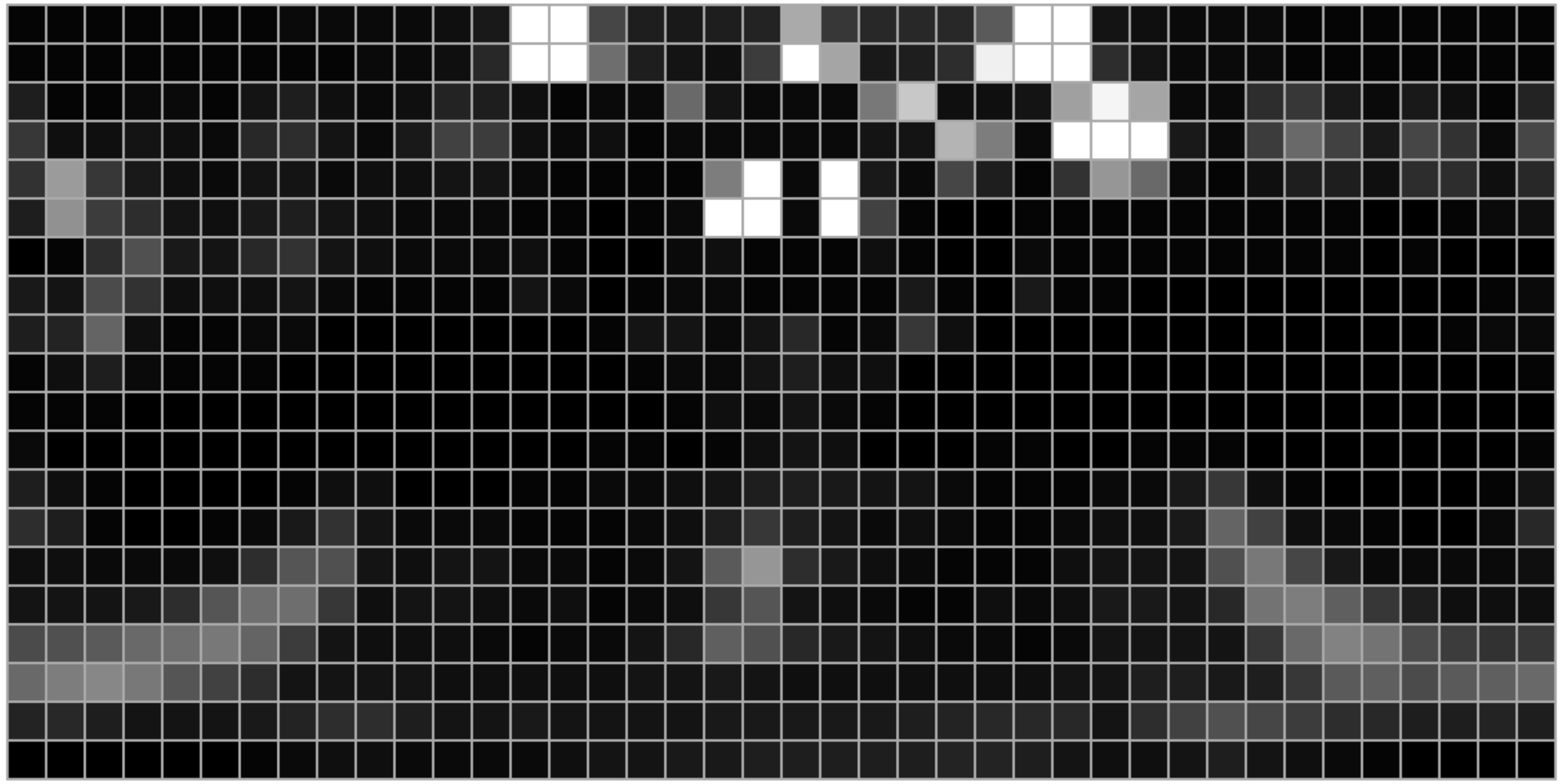
- an infinitely far area light source represented as an image



Environment maps sampling: treat it as a big discrete 2D table

- first sample a row, then sample a column

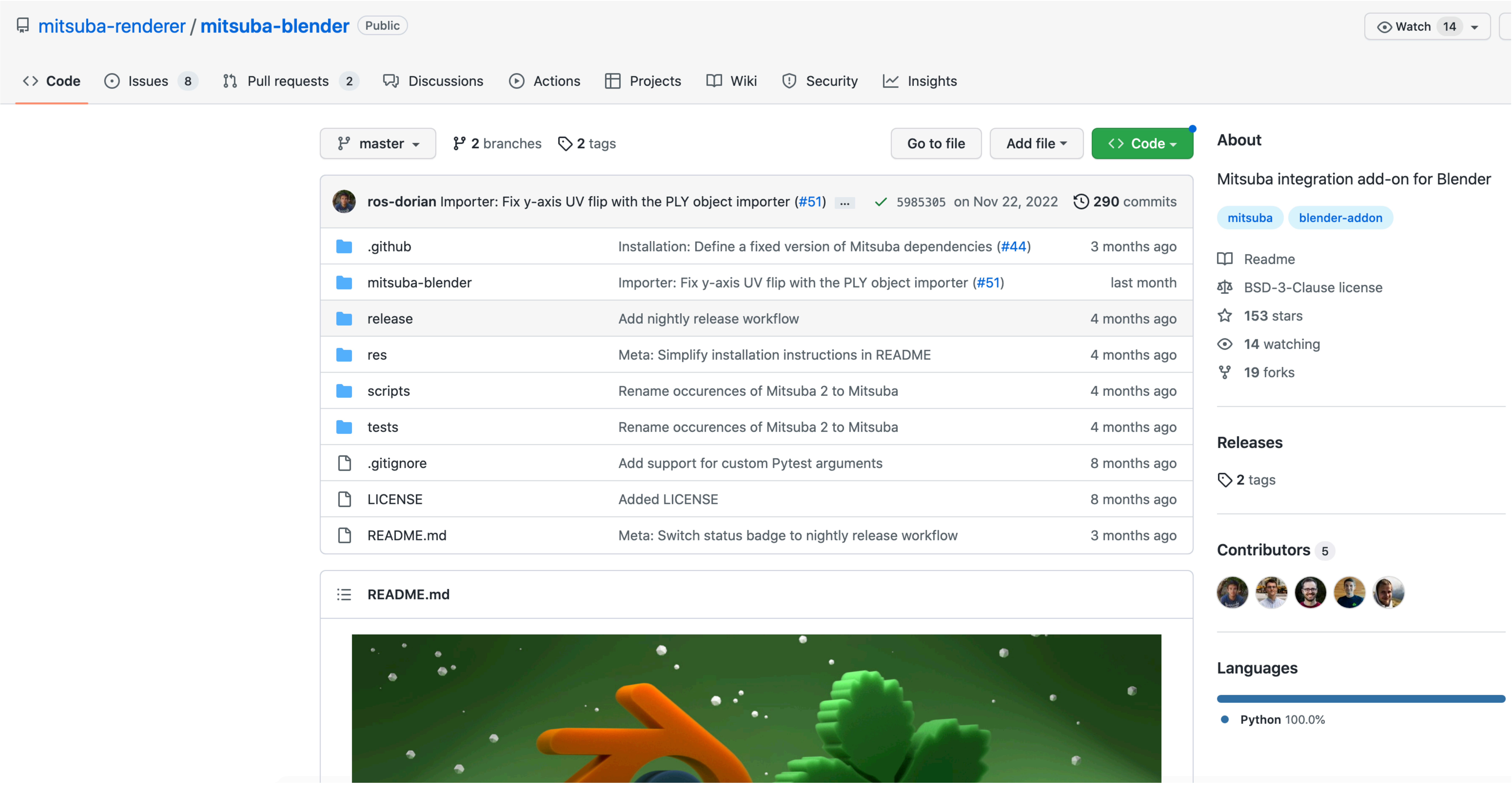




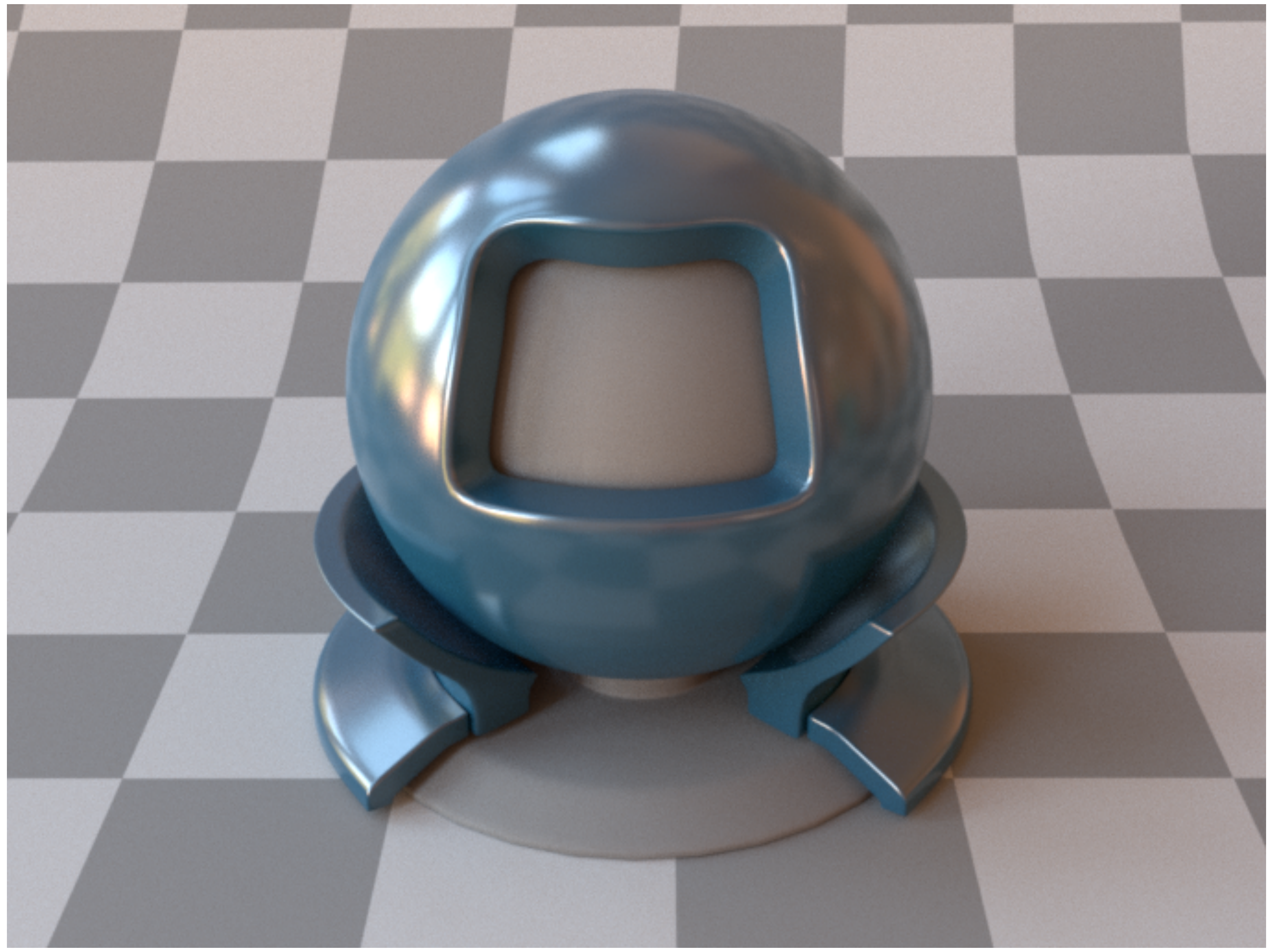
images from https://www.pbr-book.org/3ed-2018/Monte_Carlo_Integration/2D_Sampling_with_Multidimensional_Transformations

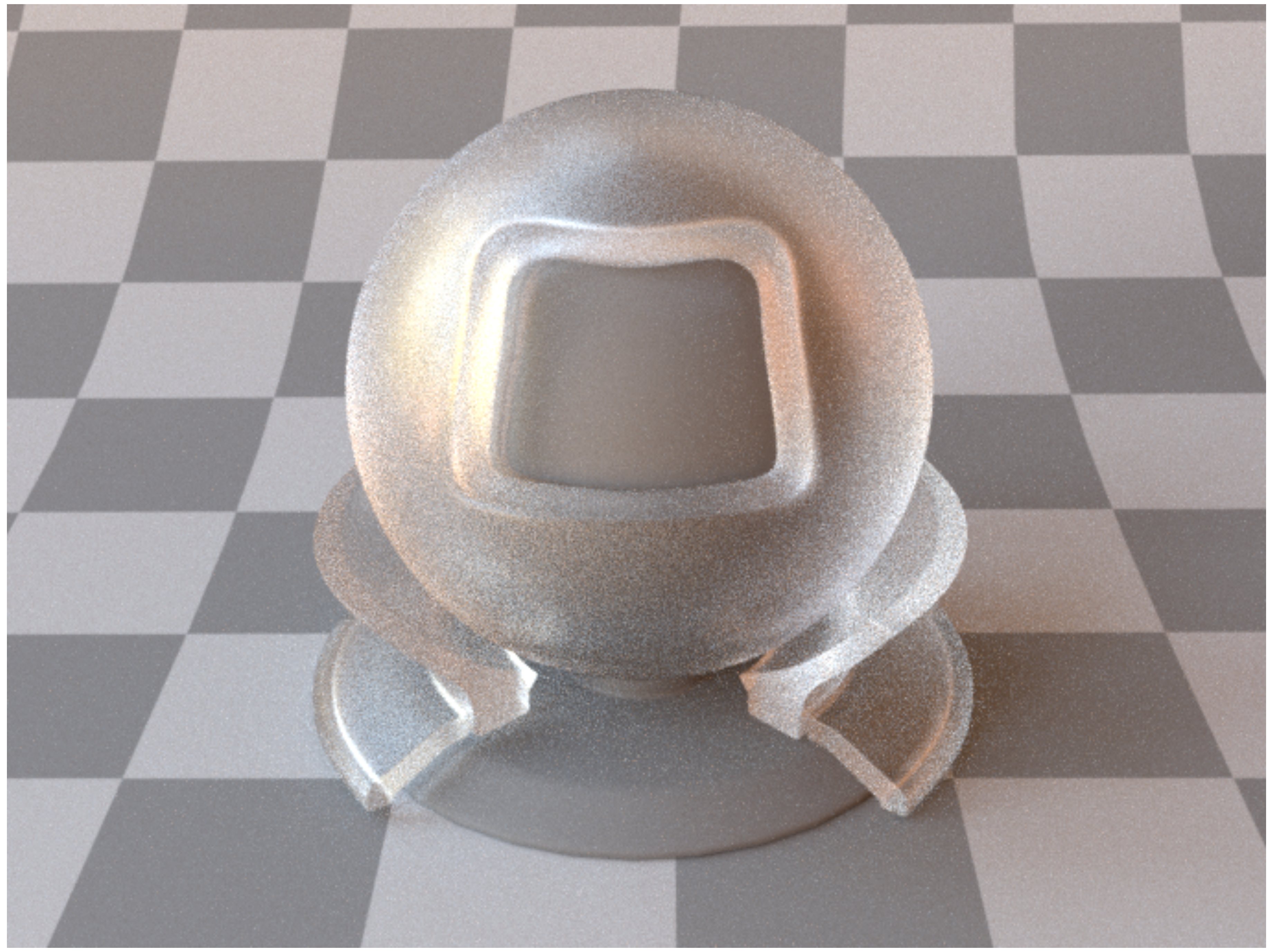
see https://www.pbr-book.org/3ed-2018/Light_Transport_I_Surface_Reflection/Sampling_Light_Sources#InfiniteAreaLights for more details

Blender-Mitsuba converter



Next time:
bidirectional scattering distribution function





$$L(\mathbf{p}, \omega) = L_e(\mathbf{p}, \omega) + \int f_{\mathbf{p}}(\omega, \omega') L(\mathbf{p}', -\omega') \, | n_{\mathbf{p}} \cdot \omega' | \, \mathrm{d}\omega'$$