UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

Computer Graphics

Class 31. Rendering. State of the art and/or some current applications in industry of the topics covered by this part of the course.

Professor: Eric Biagioli

TRANSFORMATEC

Today

- Implicit surfaces (continuation). Another way to sub-divide: interval arithmetic.
- Basic ideas on rendering
  - Painter's Algorithm
  - Ray tracing
  - Radiosity
- State of the art / applications of meshes, rendering, etc.:
  - Gaussian Splatting.
  - PBRT, Blender, Unreal, Unity.
  - Random Parametric Filtering (2012).
  - Shadow harmonization

References for the class of today: (next slide)

Today

References for the class of today: (part 1)

- L. H. de Figueiredo and J. Stolfi (1996), *Adaptive enumeration of implicit surfaces with affine arithmetic*. Computer Graphics Forum, 15 5, 287–296.
- L. H. de Figueiredo and J. Stolfi (2004) *Affine arithmetic: concepts and applications*. Numerical Algorithms 37 (1–4), 147–158.
- Hughes, J. F., van Dam, A., McGuire, M., Sklar, D. F., Foley, J. D., Feiner, S., and Akeley, K. *Computer Graphics: Principles and Practice, 3 ed*. Addison-Wesley, Upper Saddle River, NJ, 2013. → **Sections 36.4.1 (The Painter's Algorithm), 15.1, 15.2 and 15.4 (Ray casting), 31.10 (Radiosity)**
- https://en.wikipedia.org/wiki/Painter%27s_algorithm#:~:text=The%20name%20%22painter's%20algorithm%22%20refers,some%20areas%20of%20distant%20parts. **(Painter's Algorithm)**
- https://www.geeksforgeeks.org/painters-algorithm-in-computer-graphics/ **(Painter's Algorithm)**
- https://en.wikipedia.org/wiki/Ray_tracing_(graphics) (**Ray tracing)**
- James Buck. *The Ray Tracer Challenge. A Test-Driven Guide to your first 3D Renderer*. The Pragmatic Programmers, 2019. **(Ray tracing.** Not used today, and not expected for the exam, but interesting bibliography that explains in a completely hands-on approach**)**

Today

References for the class of today: (part 2)

- https://www.geeksforgeeks.org/radiosity-rendering-in-computer-graphics/ **(Radiosity)**
- https://en.wikipedia.org/wiki/Radiosity_(computer_graphics) **(Radiosity)**
- https://medium.com/@coderfromnineteen/computer-graphics-theory-radiosity-global-illumination-8c473607e6cd **(Radiosity)**

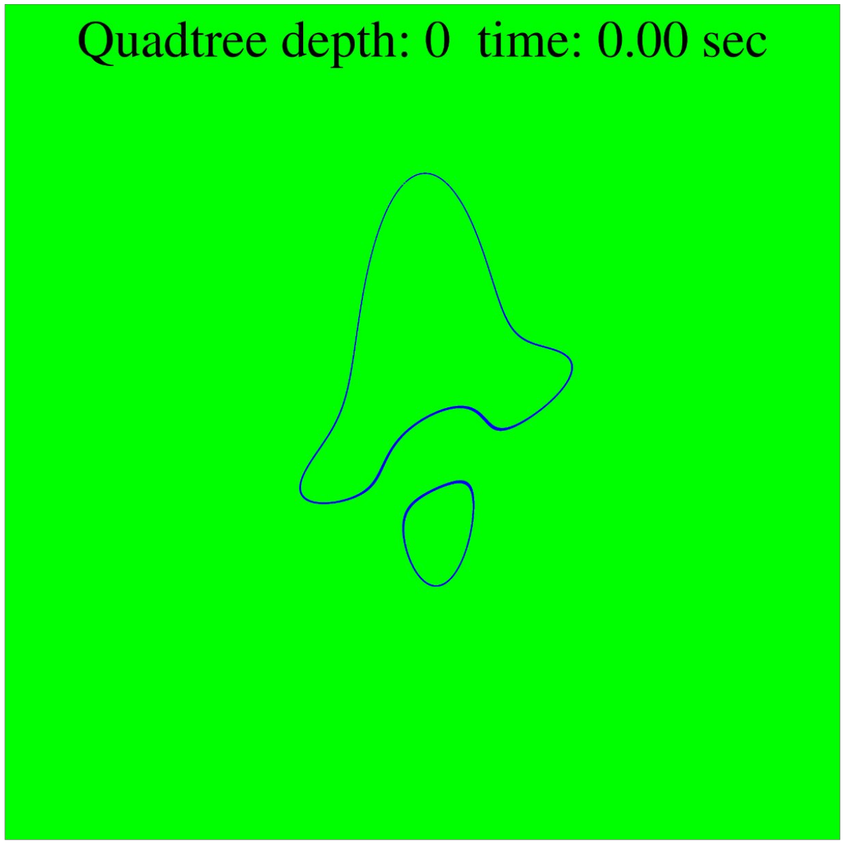
Intervals arithmetic

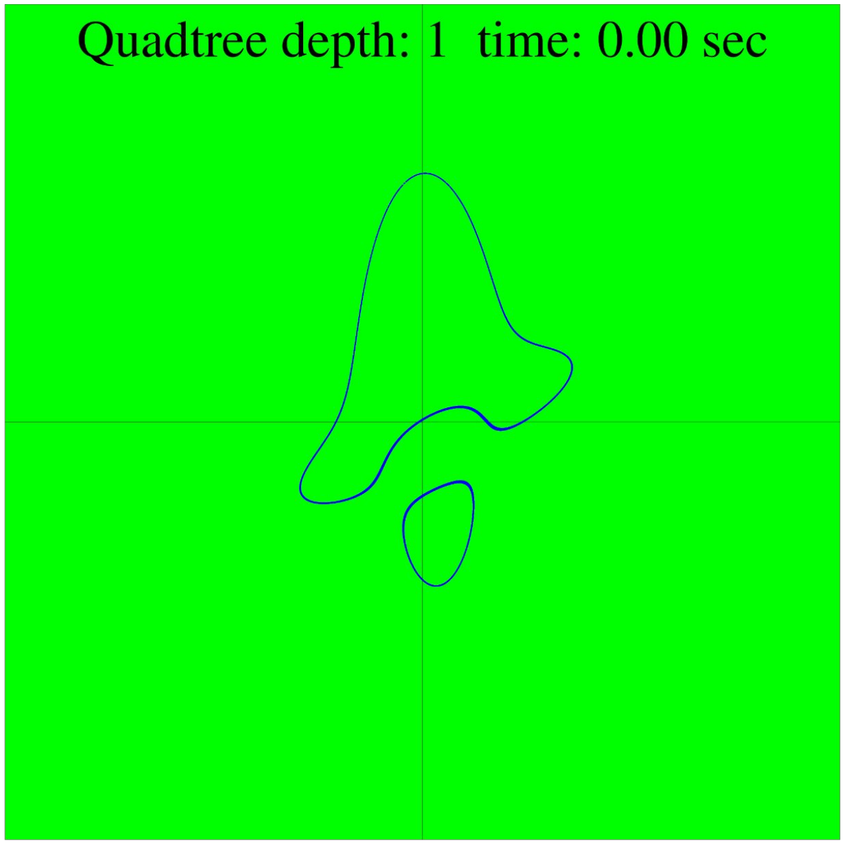
Intervals arithmetic

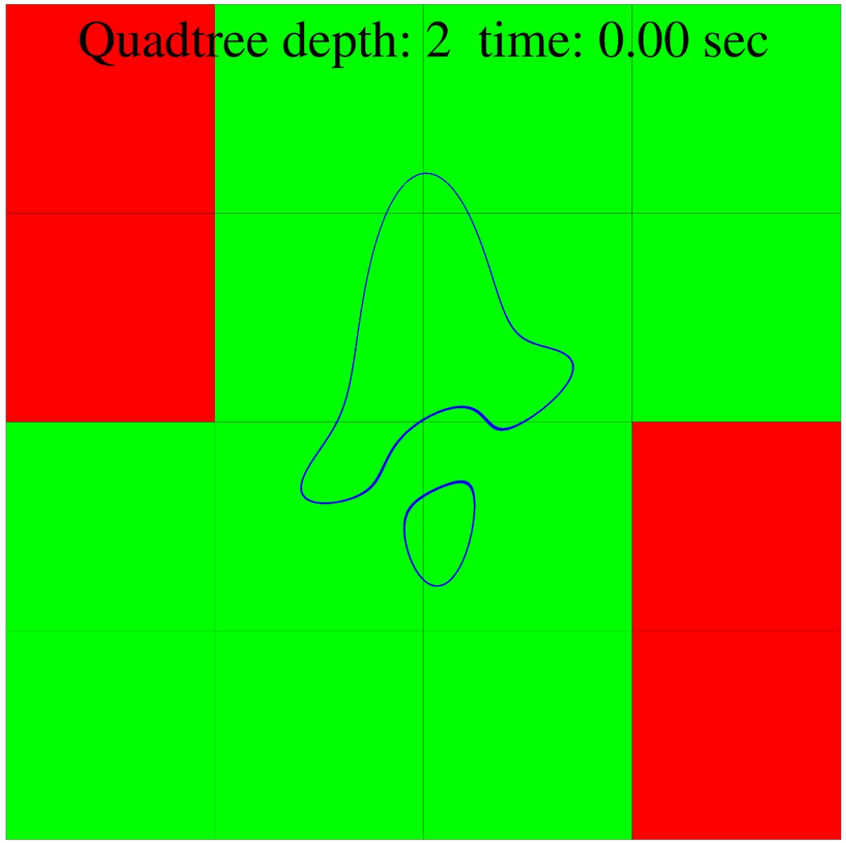
- `f(x, y, z) = x^2 + y^2 + z^2 - R^2`
- `f(x, y) = 0.004 + 0.110 * x - 0.177 * y - 0.174 * x^2 + 0.224 * x * y - 0.303 * y^2 - 0.168 * x^3 + 0.327 * x^2 * y - 0.087 * x * y^2 - 0.013 * y^3 + 0.235 * x^4 - 0.667 * x^3 * y + 0.745 * x^2 * y^2 - 0.029 * x * y^3 + 0.072 * y^4`
- How did we sub-divide? → We sampled many time the function and counted how many times the function was positive, negative and zero
- Some other way in which we can do this?

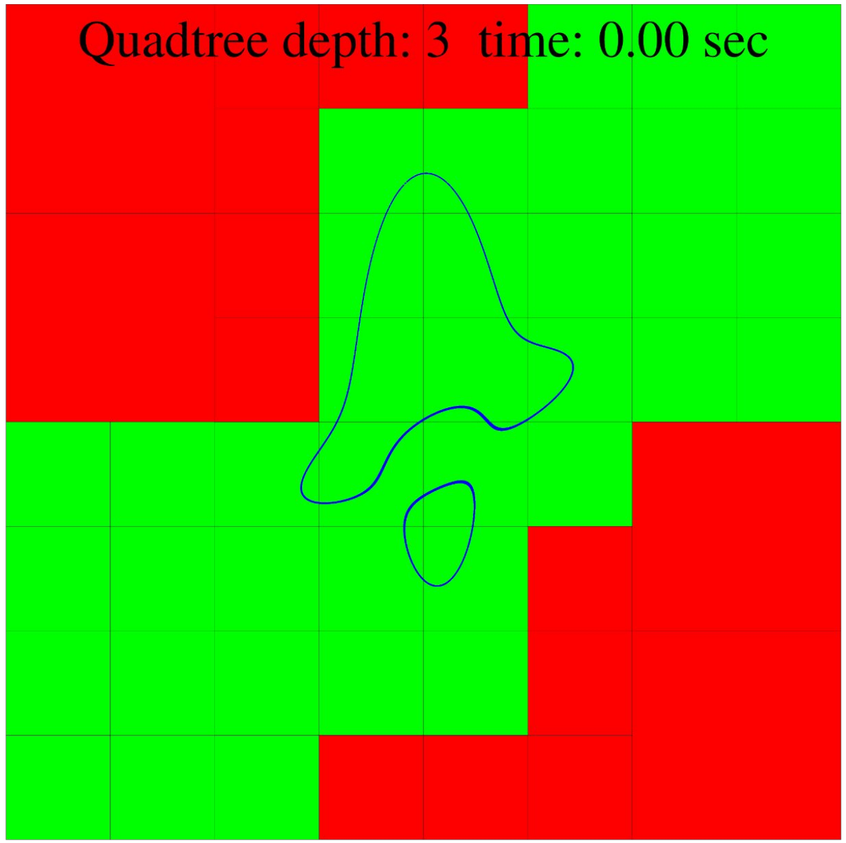
Intervals arithmetic

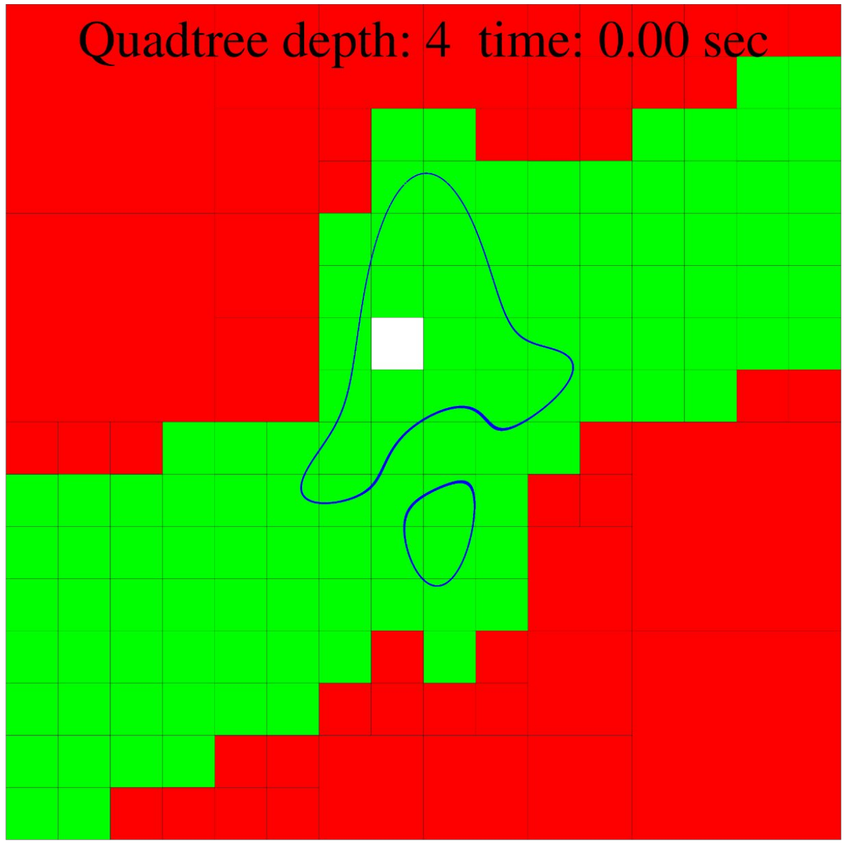
```
const Interval operator*(const Interval& i, const double d)
{
    return Interval(i.a * d, i.b * d);
}

const Interval operator*(const double d, const Interval& i)
{
    return i * d;
}

const Interval operator*(const Interval& i1, const Interval& i2)
{
    double aa = i1.a * i2.a;
    double ab = i1.a * i2.b;
    double ba = i1.b * i2.a;
    double bb = i1.b * i2.b;

    double m = min(min(aa, ab), min(ba, bb));
    double M = max(max(aa, ab), max(ba, bb));

    return Interval(m, M);
}

const Interval operator+(const Interval& i1, const Interval& i2)
{
    return Interval(i1.a + i2.a, i1.b + i2.b);
}

const Interval operator-(const Interval& i1, const Interval& i2)
{
    return Interval(i1.a - i2.b, i1.b - i2.a);
}
```

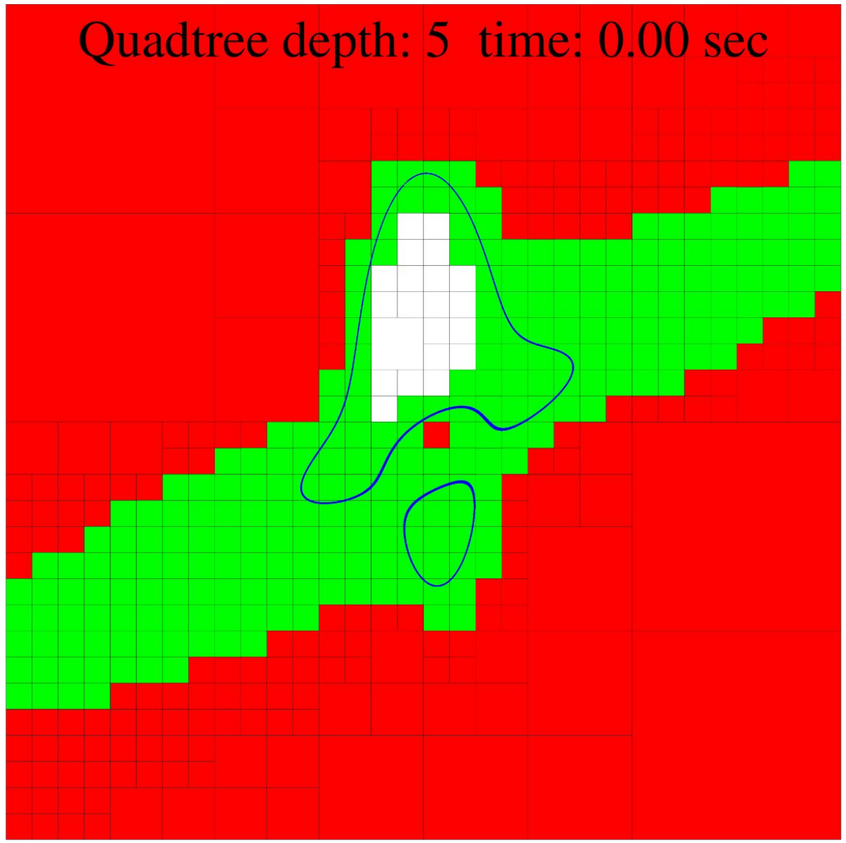

Quadtree depth: 0 time: 0.00 sec

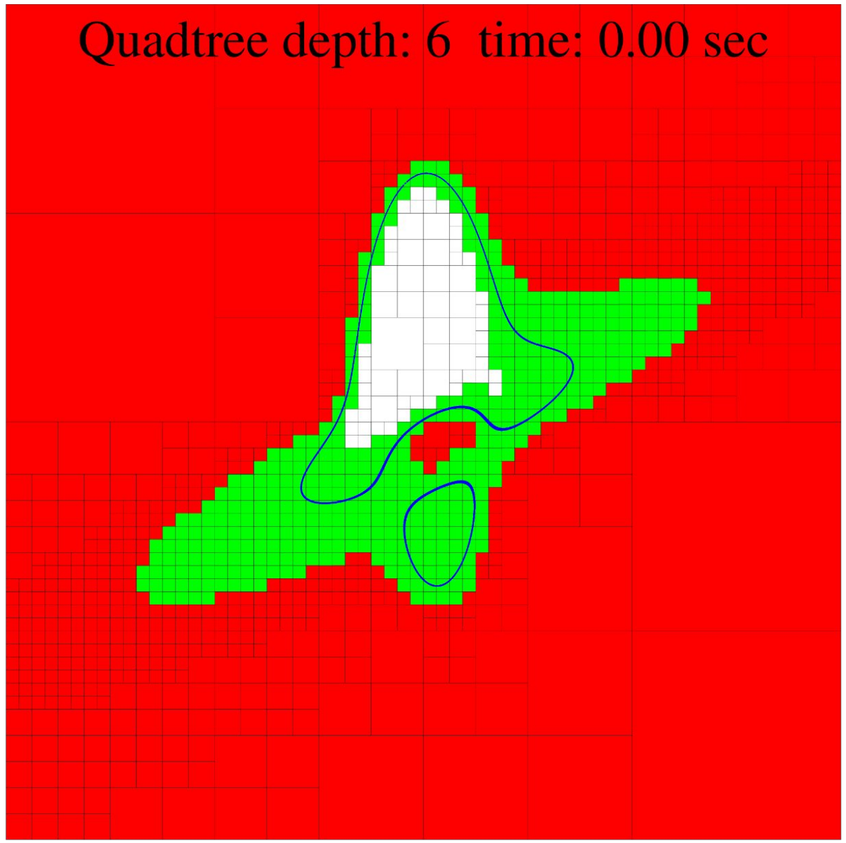

Quadtree depth: 1 time: 0.00 sec

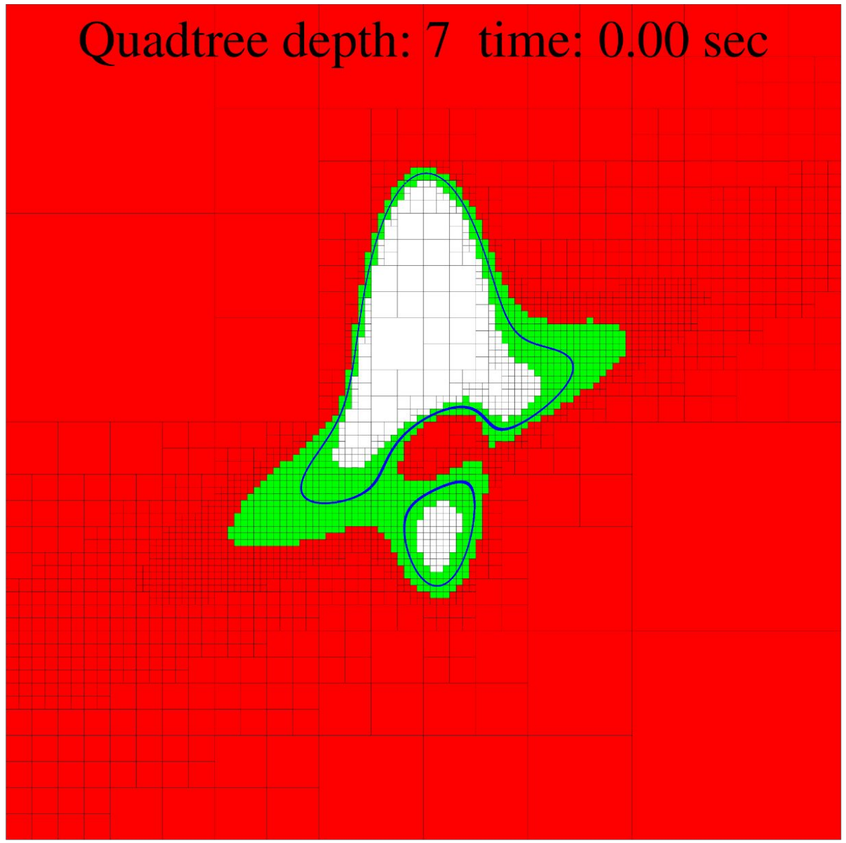

Quadtree depth: 2 time: 0.00 sec

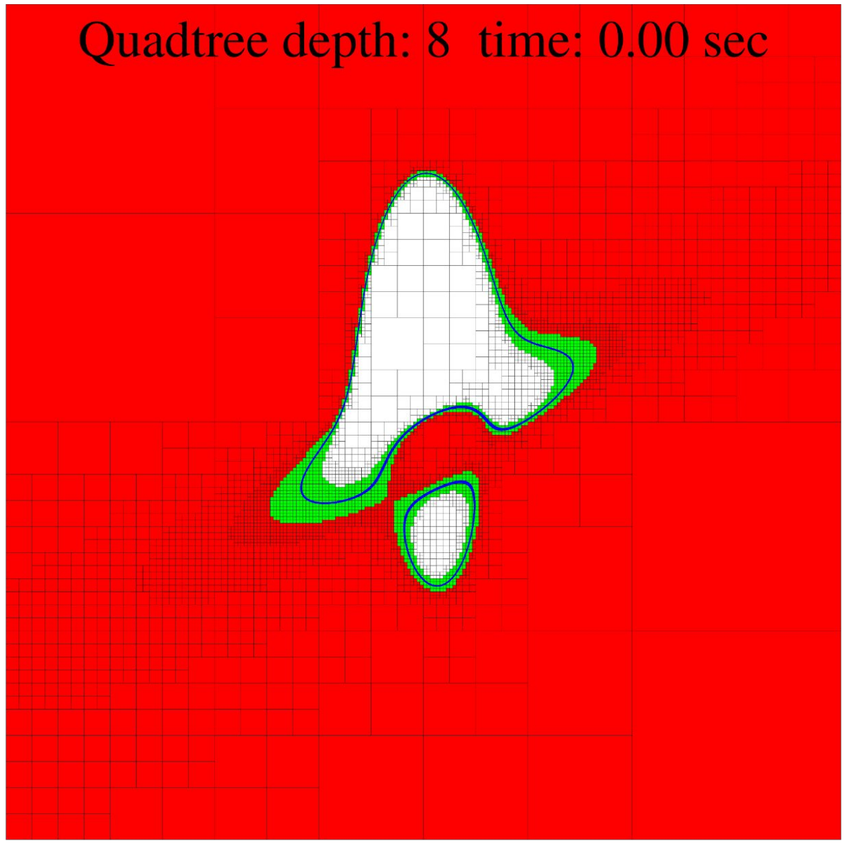

Quadtree depth: 3 time: 0.00 sec

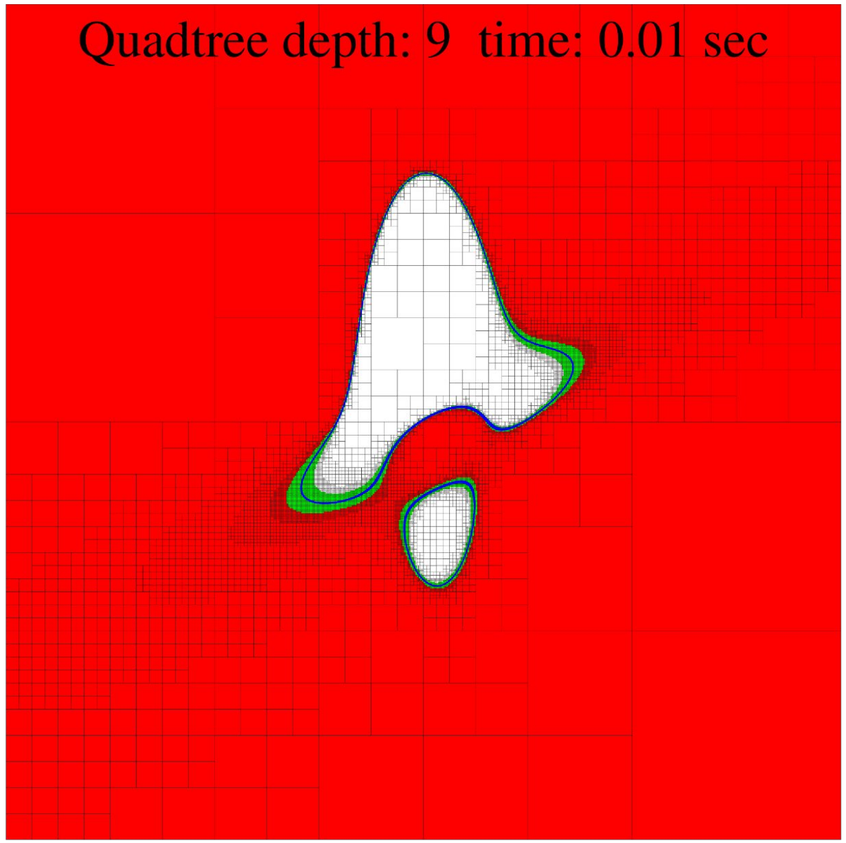

Quadtree depth: 4 time: 0.00 sec

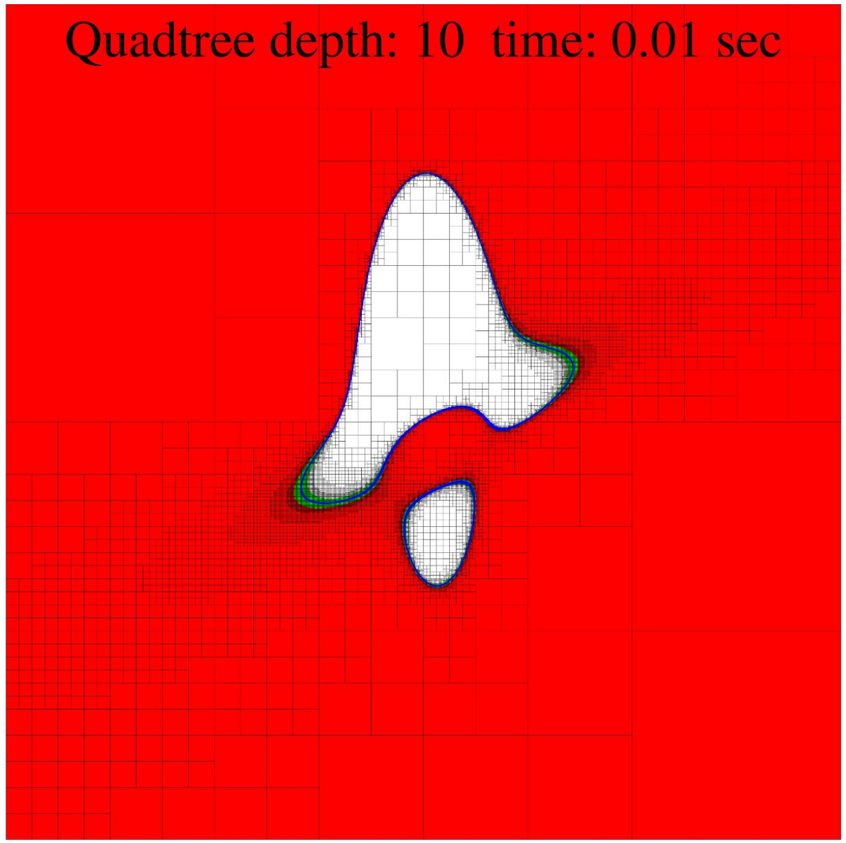

Quadtree depth: 5 time: 0.00 sec

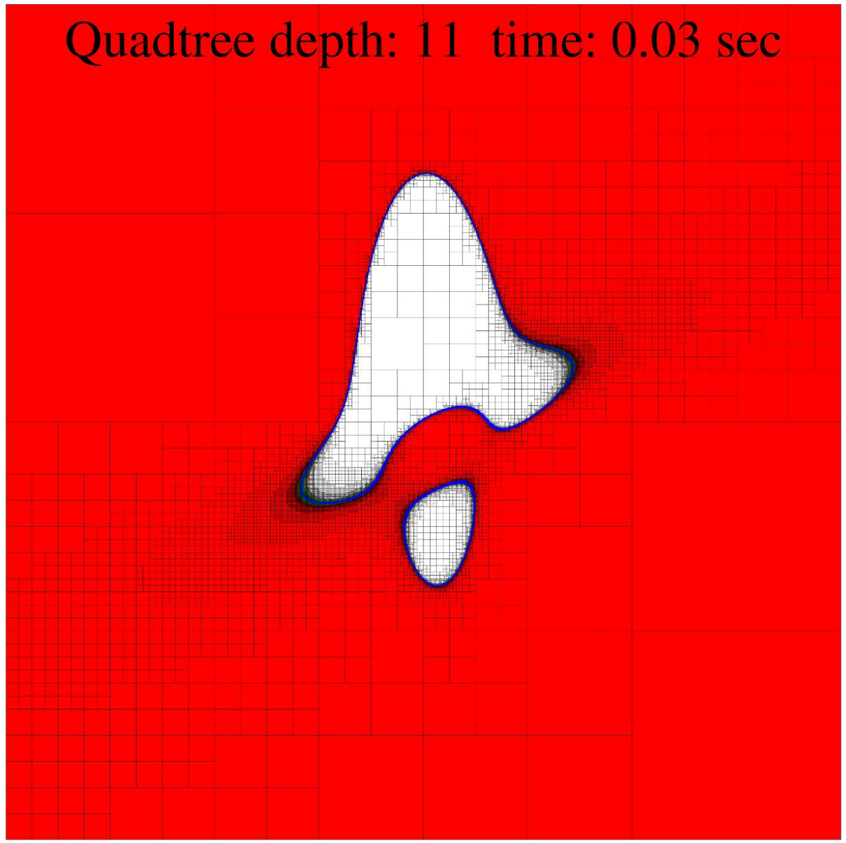

Quadtree depth: 6 time: 0.00 sec

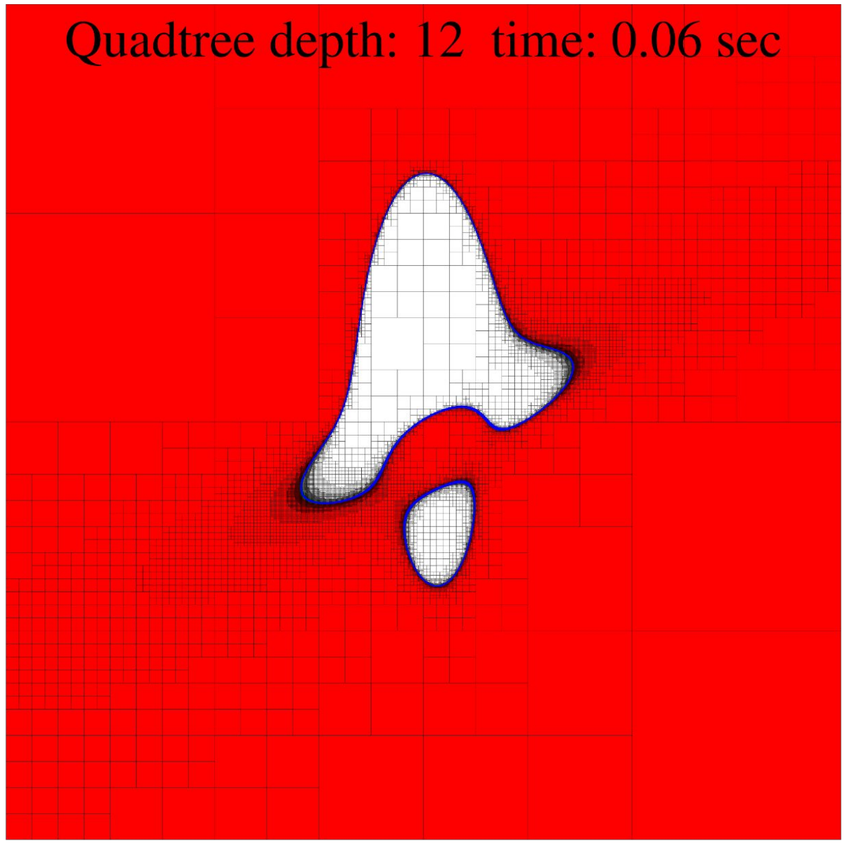

Quadtree depth: 7 time: 0.00 sec

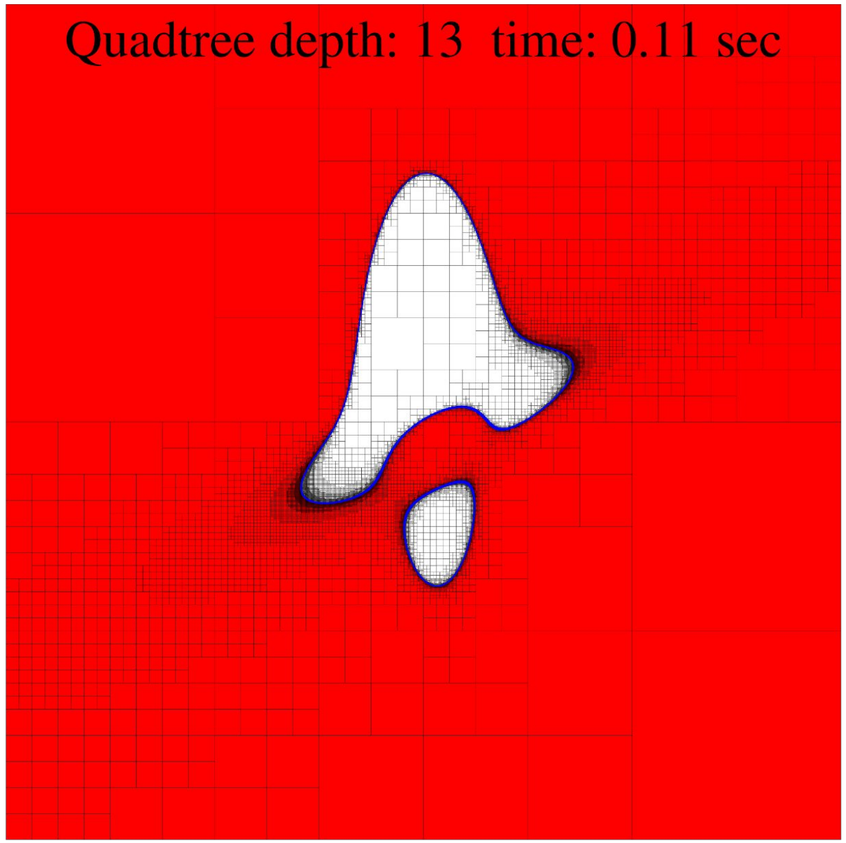

Quadtree depth: 8 time: 0.00 sec

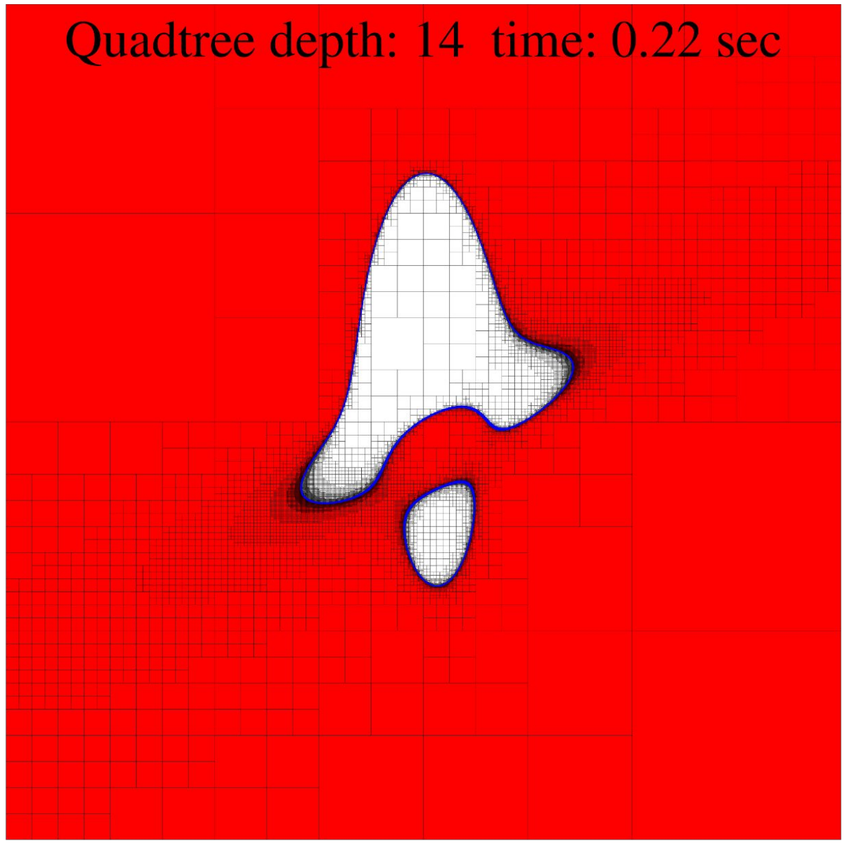

Quadtree depth: 9 time: 0.01 sec


Quadtree depth: 10 time: 0.01 sec


Quadtree depth: 11 time: 0.03 sec


Quadtree depth: 12 time: 0.06 sec


Quadtree depth: 13 time: 0.11 sec


Quadtree depth: 14 time: 0.22 sec

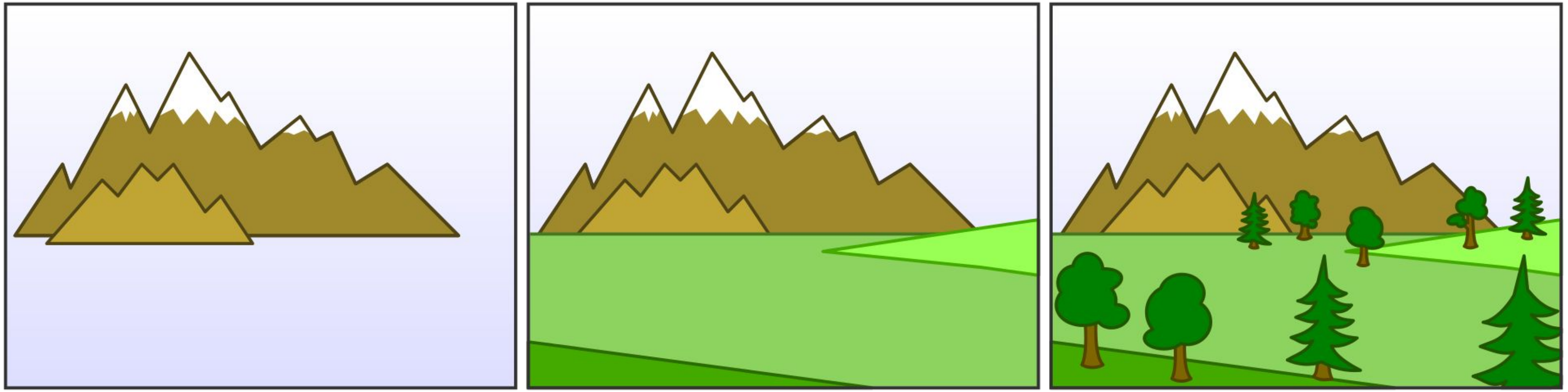
Basic ideas on rendering

Rendering

- What is "to render"? → to generate an image (or animation) from a model.
- What is the importance of the problem of rendering in Computer Graphics?

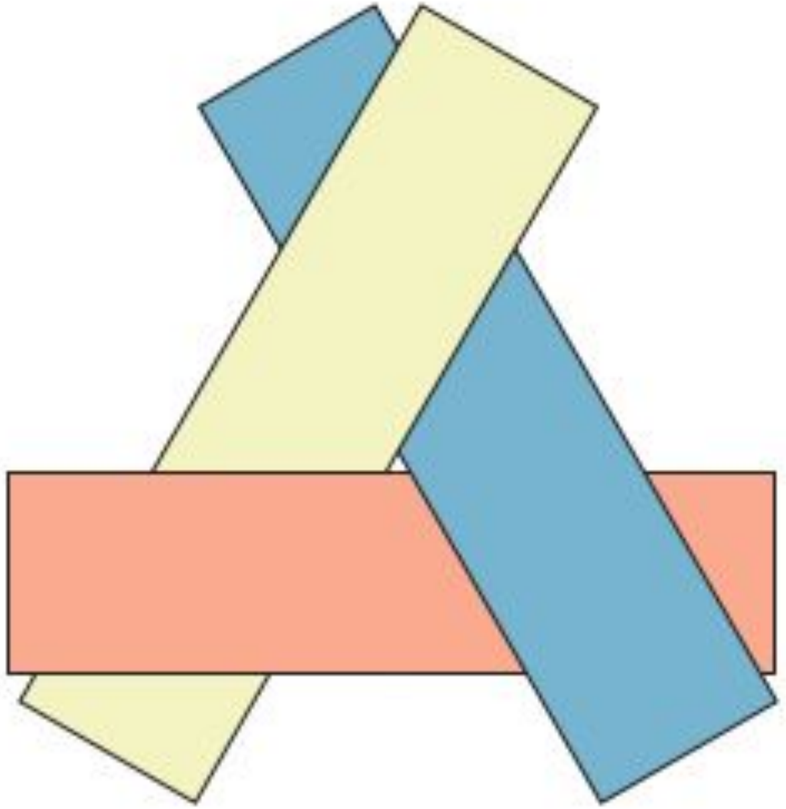
Rendering: The Painter's algorithm

- In principle, simple! Draw the polygons that are further first…
- How we do that?
  - We sort the polygons from "further" to "closer" and we proceed painting them in that order
- Any problem with this approach?



Rendering: The Painter's algorithm

- How do we paint this?



Rendering: The Painter's algorithm

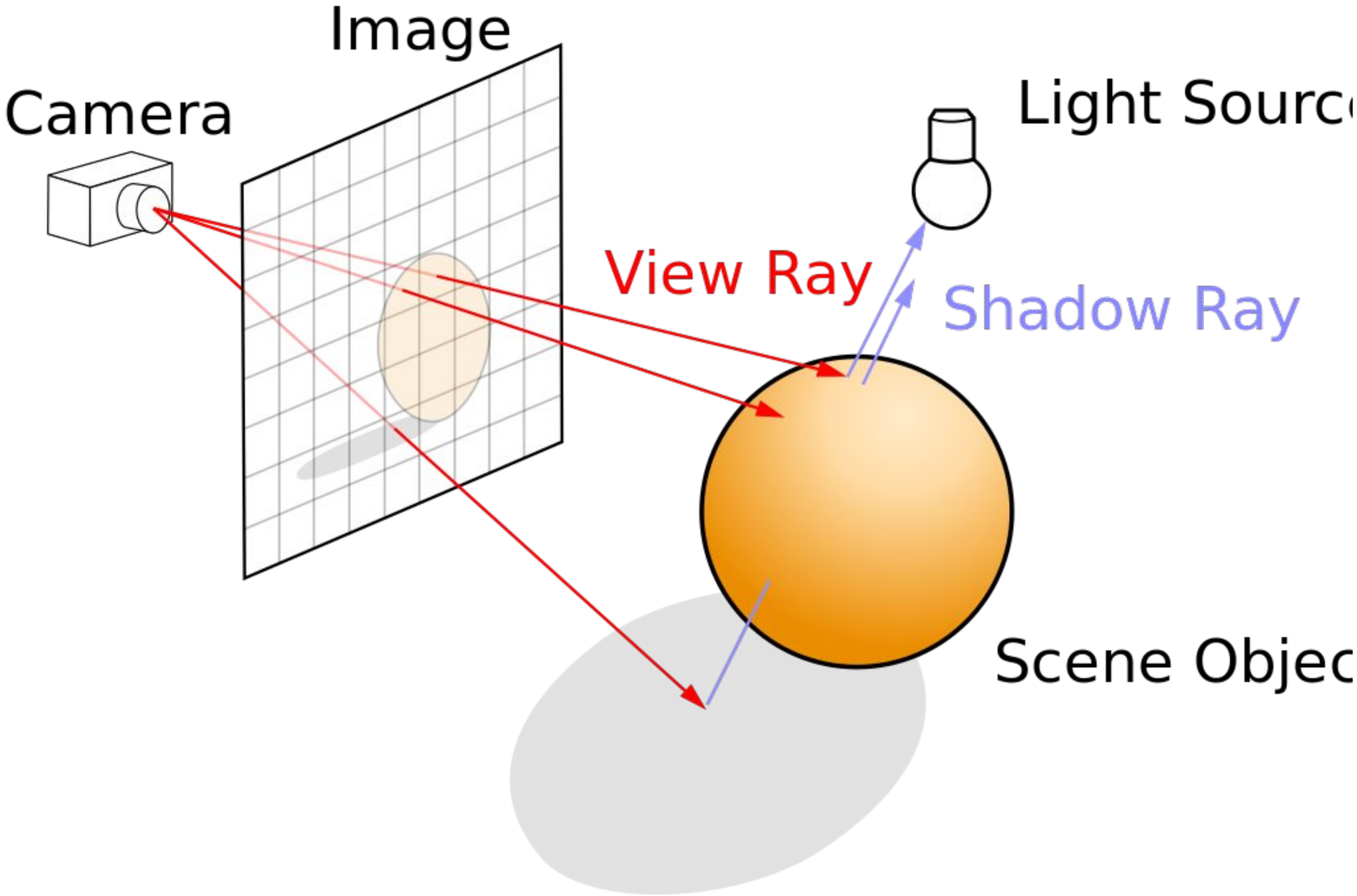
- The Painter's algorithm is the algorithm used in almost all user interfaces, where the mentioned problem does not exist.
- There are multiple ways to solve the problem. The most widely used is: when there is ambiguity, divide the underlying polygons in a way that removes the ambiguity. (A.K.A. "Depth-sort algorithm")

Rendering: Ray tracing

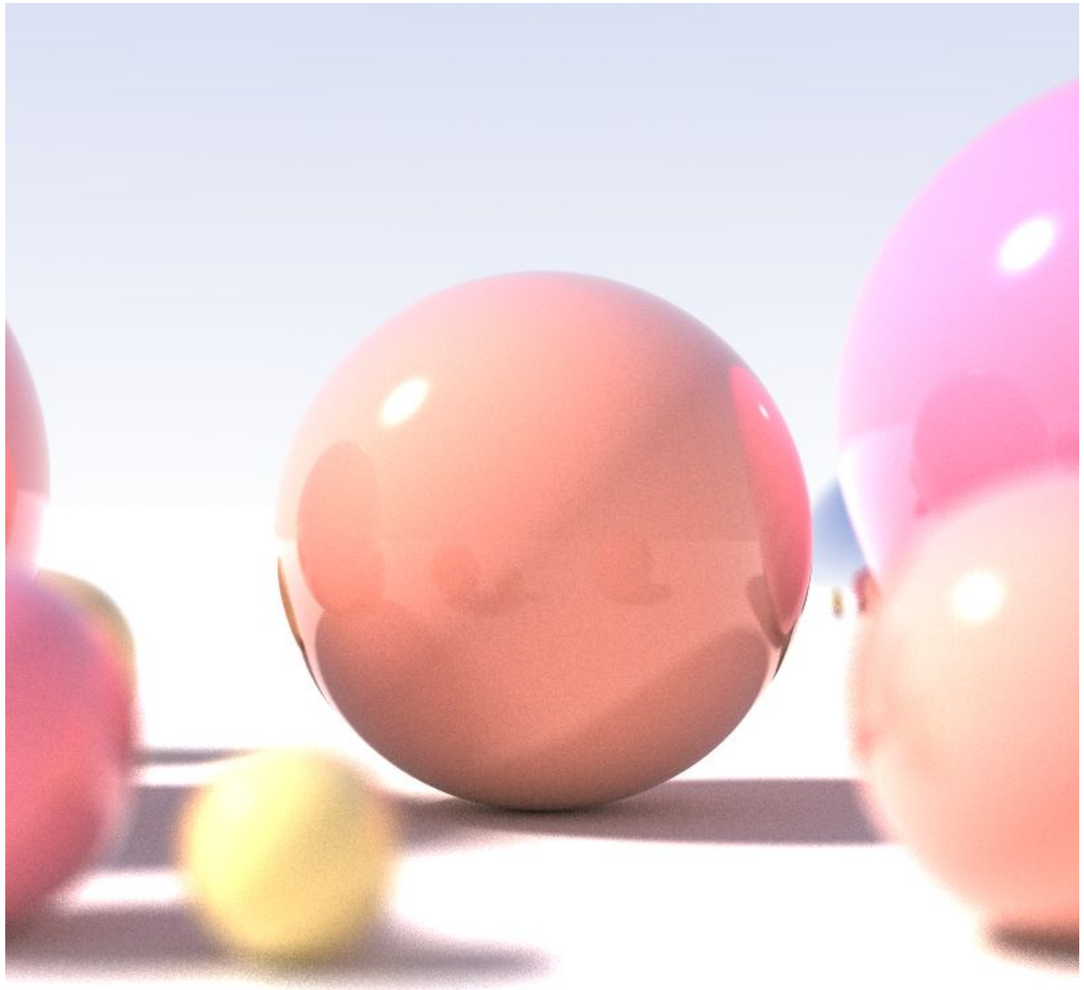
Listing 15.11: Detailed pseudocode for a ray-casting renderer.

```
1  for each pixel row y:
2      for each pixel column x:
3          let R = ray through screen space position (x + 0.5, y + 0.5)
4          closest = ∞
5          for each triangle T:
6              d = intersect(T, R)
7              if (d < closest)
8                  closest = d
9                  sum = 0
10                 let P be the intersection point
11                 for each direction ω_i:
12                     sum += light scattered at P from ω_i to ω_o
13
14                 image[x, y] = sum
```

Rendering: Ray tracing

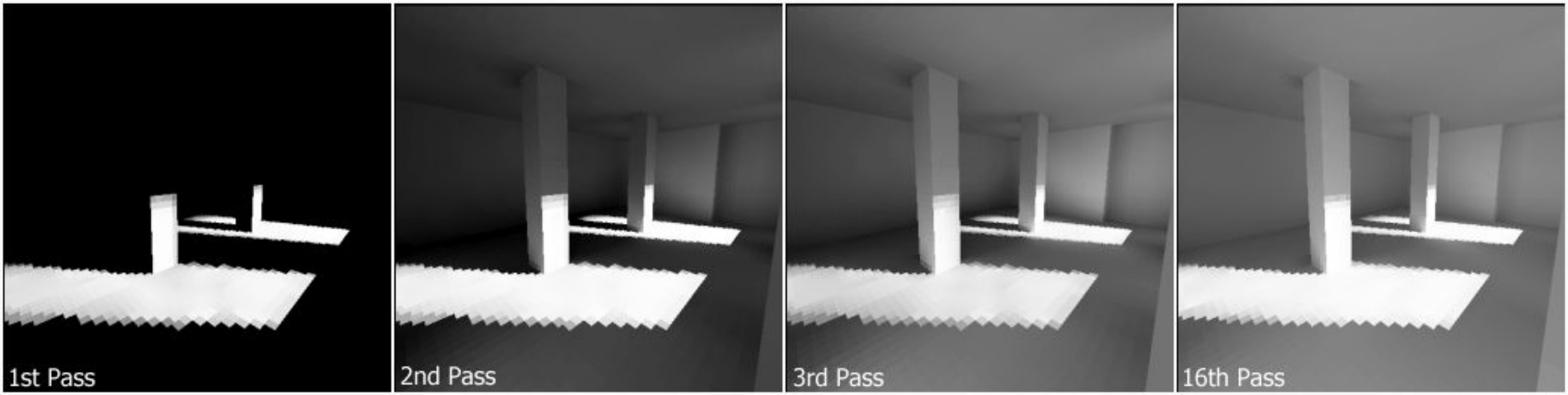


Rendering: Ray tracing

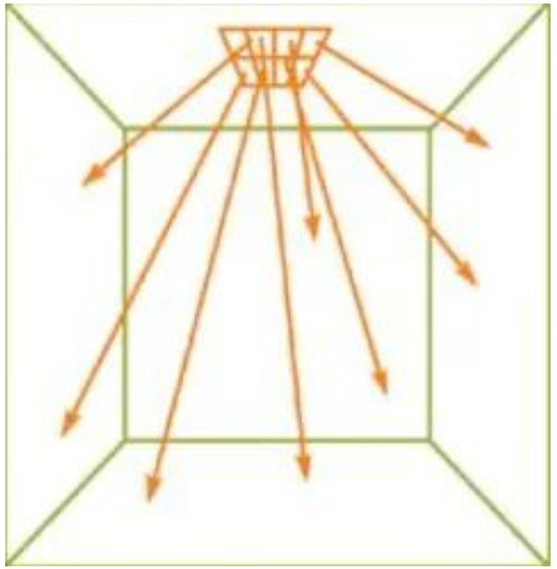


Rendering: Radiosity

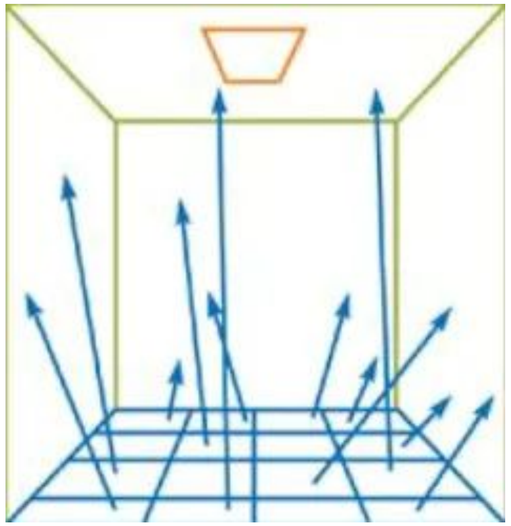
- global illumination for diffuse surfaces
- view-independent
- soft shadows, color bleeding
- energy transport between surface patches (using area light sources)
- Energy transport means how a energy held by light sources is transported to another object.



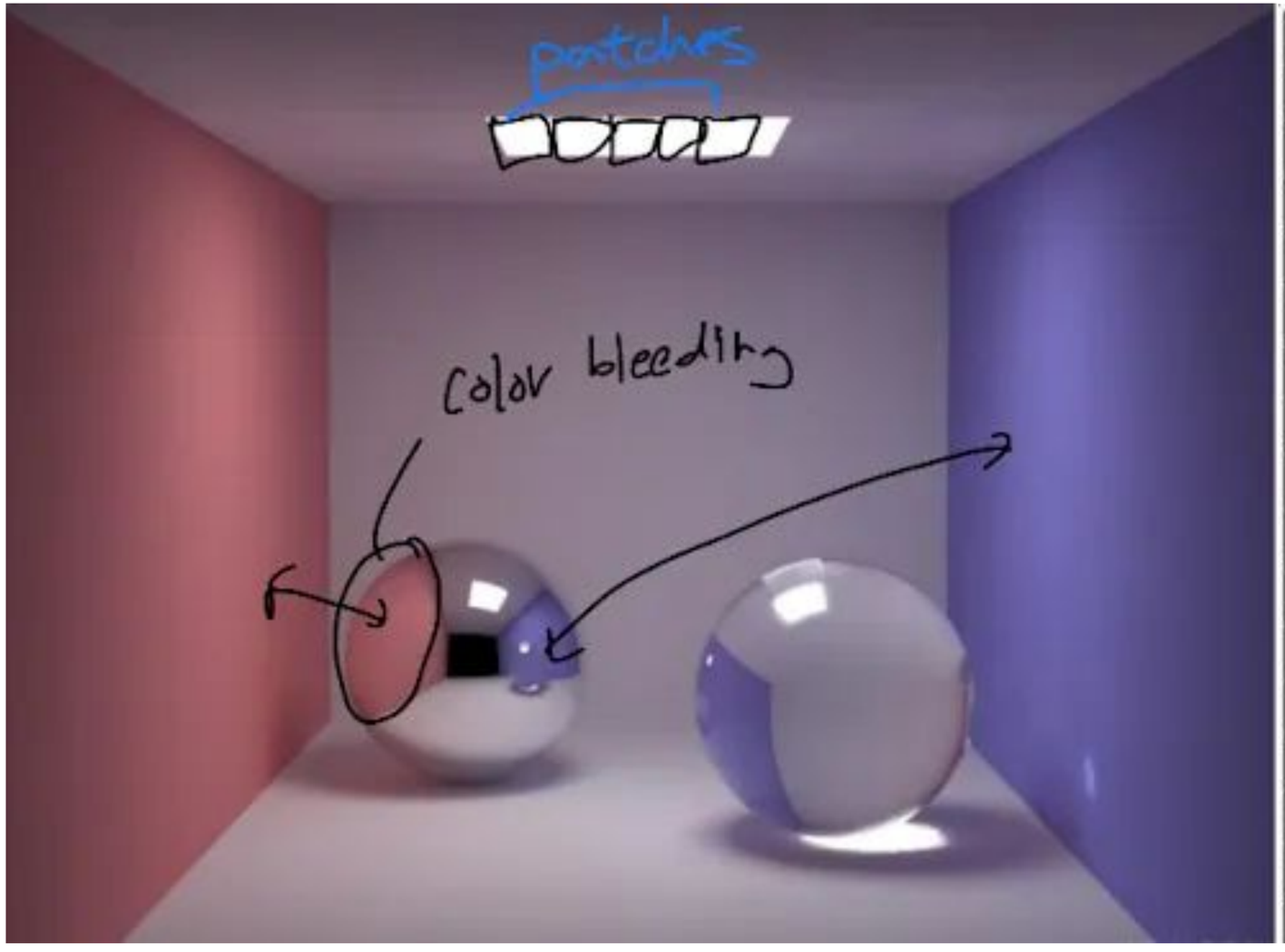
1. Dive the surface into patches
2. For every pair of patches (A, B), compute a *form factor* describing how much energy flows from A to B
3. Calculate the lighting of all directly-lit patches

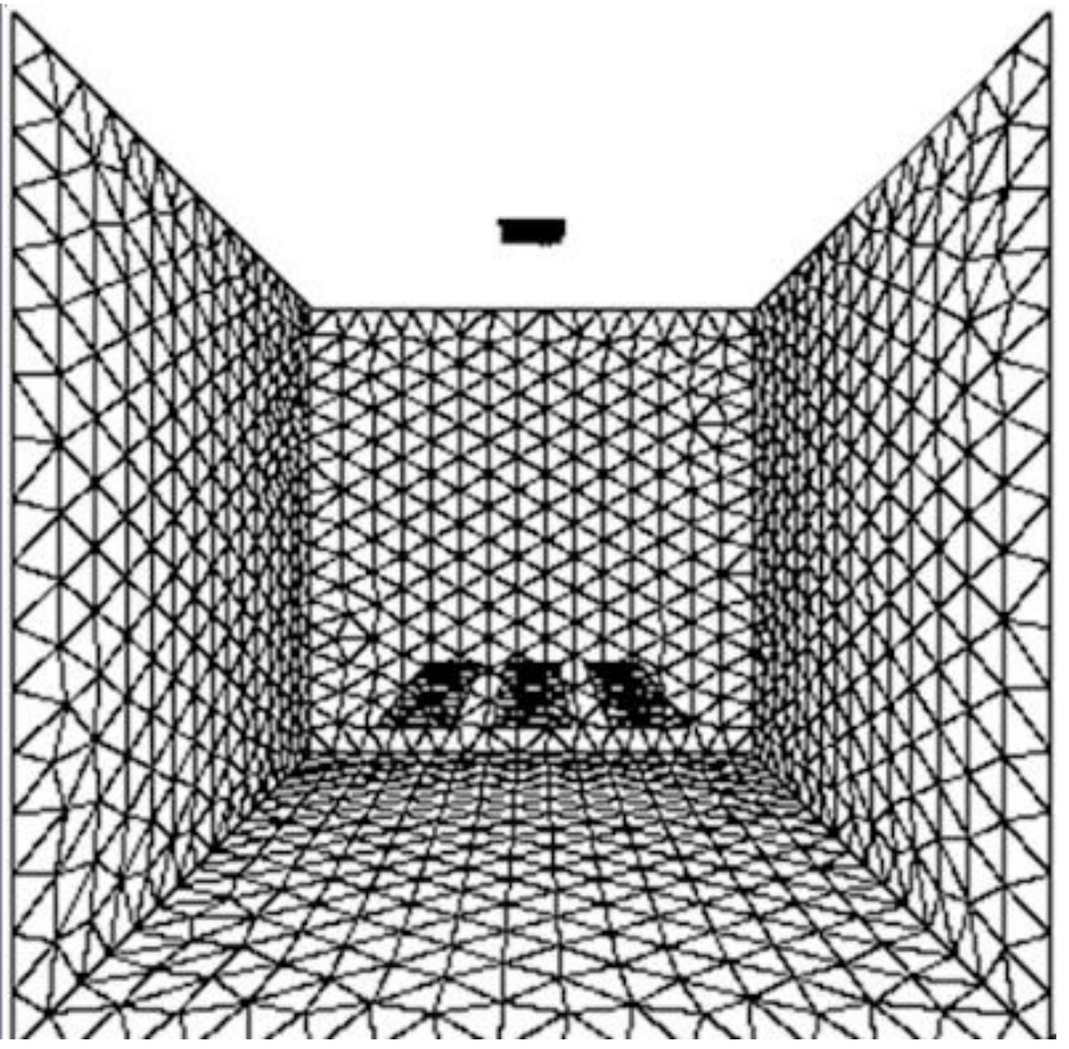


4. Bounce the light from all lit patches to all those they light, Carrying more light to patches with higher form factors



5. Repeat

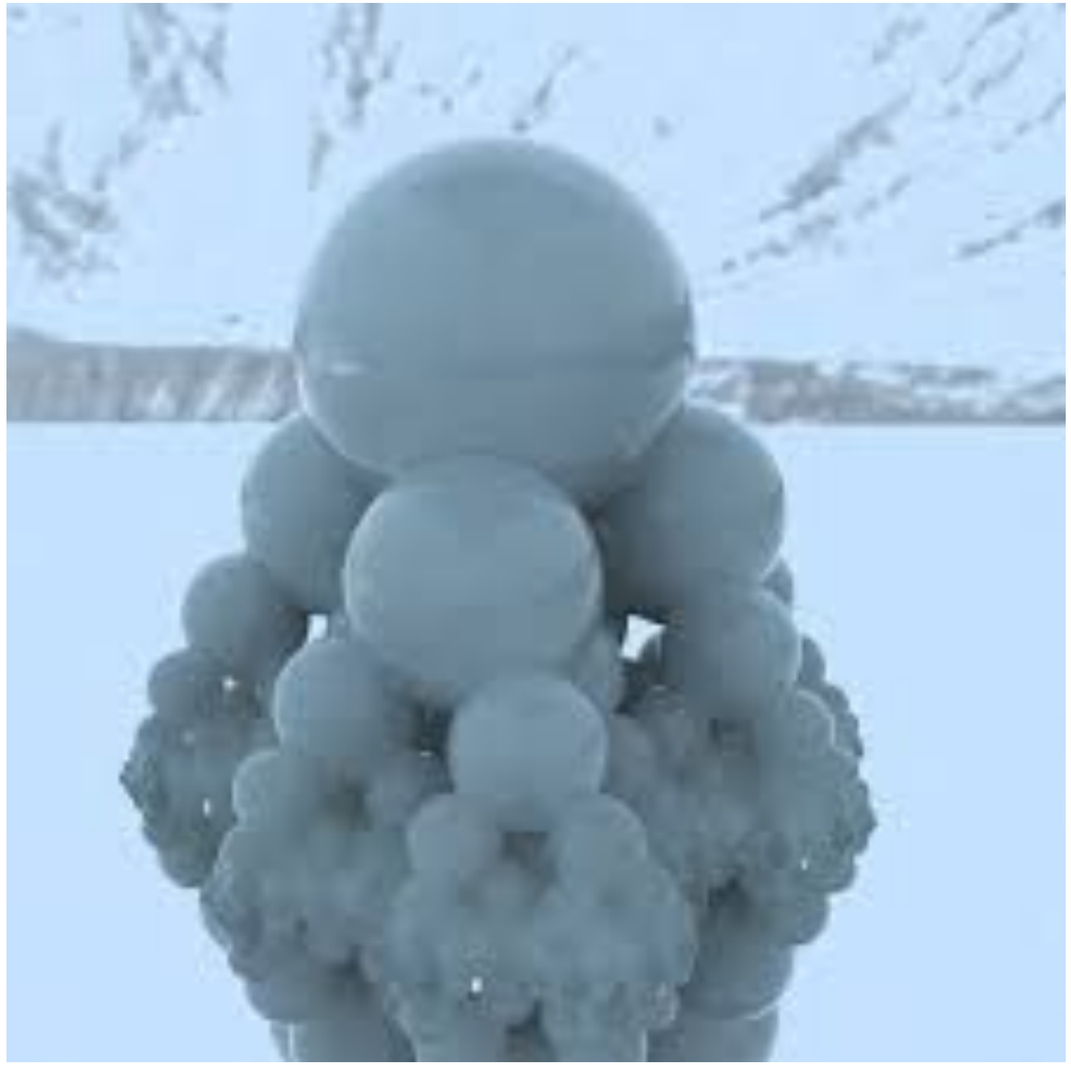

patches
color bleeding



Rendering: Radiosity

Applications, state of the art

PBRT, Blender



PBRT: specialized for research and high-precision rendering. Ideal for academics and professionals who need physically accurate simulations.

Blender: general-purpose 3D suite suitable for a wide range of creative and technical applications. Was

Gaussian Splatting

Random Parametric Filtering (2012)

ithm takes as input a small set of MC samples, which are fast to compute but very noisy. We then estimate the func

Shadow harmonization

(a) input

(b) traditional IBL

(c) ours

This week…

- PC3 will be made available on Friday. **Remark: this time the grading will be fully automatic. The implementations that don't follow the very strict definitions that will be specified in the statements for input, parameters and output, will not be considered correct, independently of its contents.**

- The laboratory exercises will cover
  - Extend the marching cubes and marching squares for boolean operations of implicit surfaces → Let's draw more complex models.
  - Interval arithmetic
  - Rendering: Painters's algorithm, raytracing, radiance
  - Rendering animations

Summary of today

- Interval arithmetic
- Basic ideas on rendering
  - Painter's Algorithm
  - Ray-casting
  - Radiosity
- State of the art: some current applications and some current ideas in recent publications.
  - Gaussian Splatting.
  - PBRT, Blender, Unreal, Unity.
  - Random Parametric Filtering (2012).
  - Shadow harmonization

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

Thank you

TRANSFORMATEC