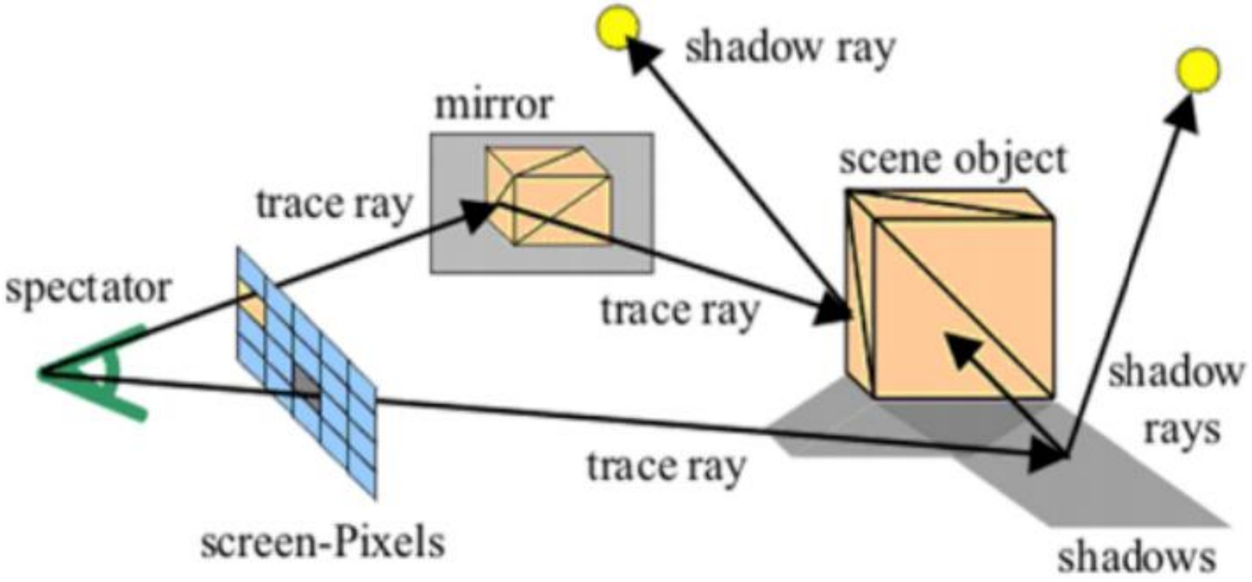
Tutorial 7

TA: Mengyun Liu, Hongtu Xu

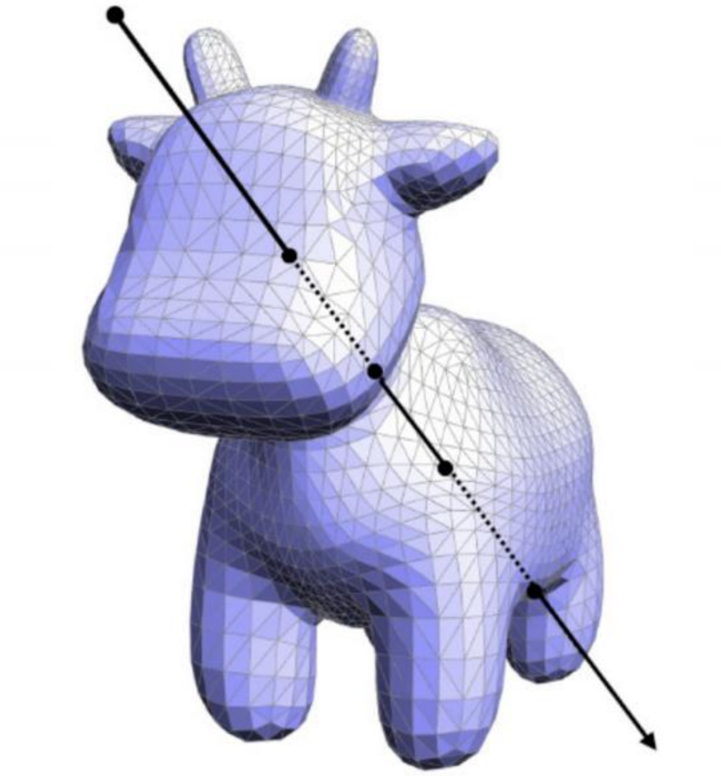
Recall the routine of ray-tracing

- Generate rays from the camera
- **Detect the interaction between rays and objects**
- Tracing the radiance back to the camera according to the rendering equation.



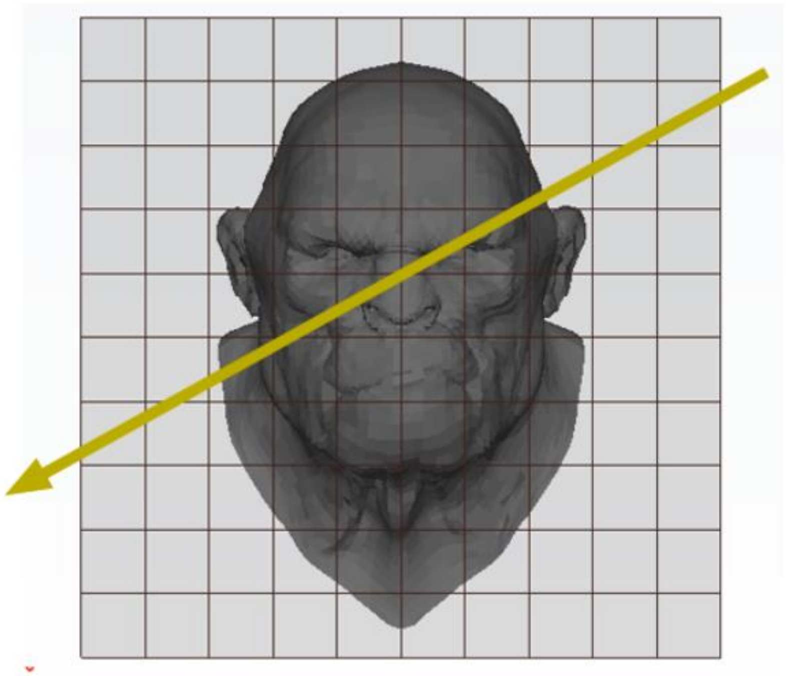
Ray-mesh intersection

- In general, meshes are composed of triangles.
- Basically, detect the interaction between each triangle and the ray.



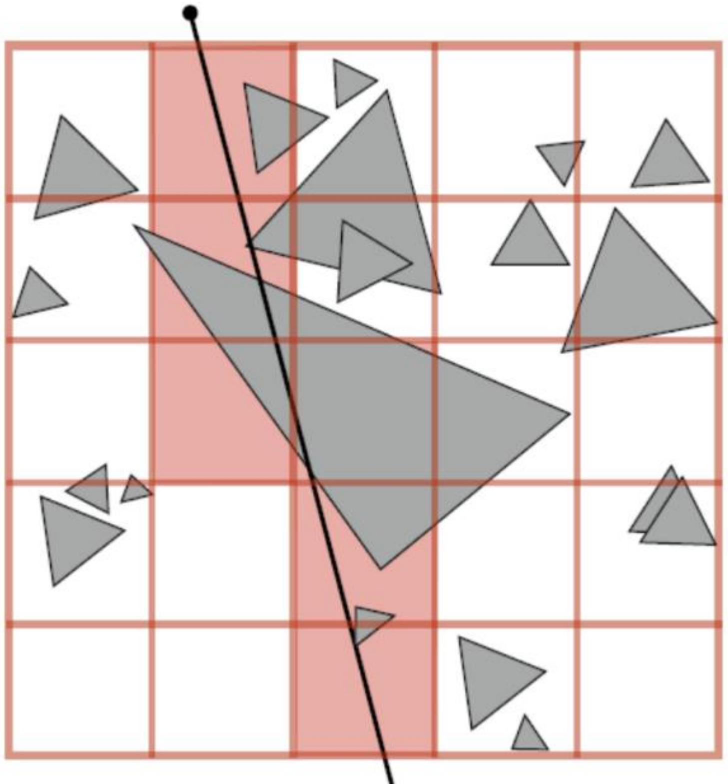
Why acceleration structures

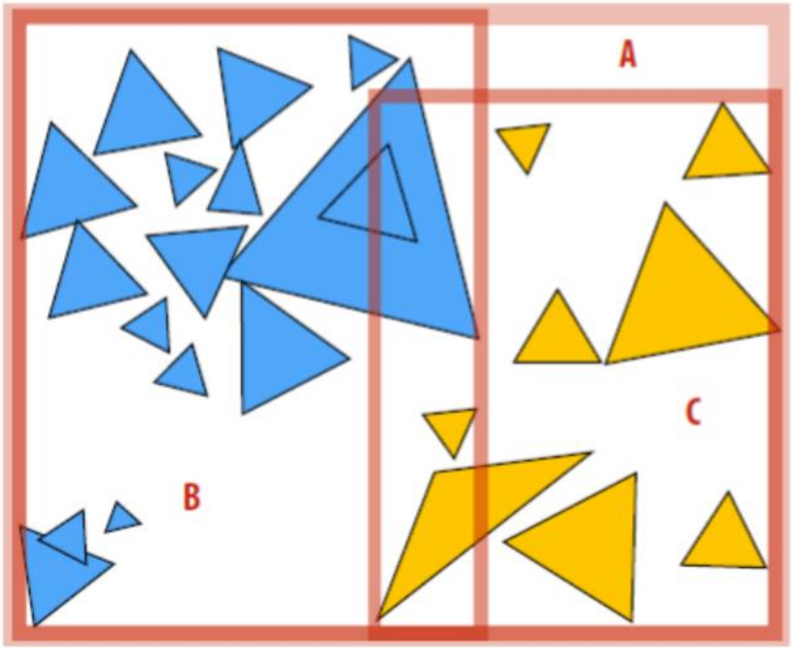
- A mesh may have too many triangles.
- However, the ray may just go through few triangles.
- Acceleration structure is used to left out some triangles.

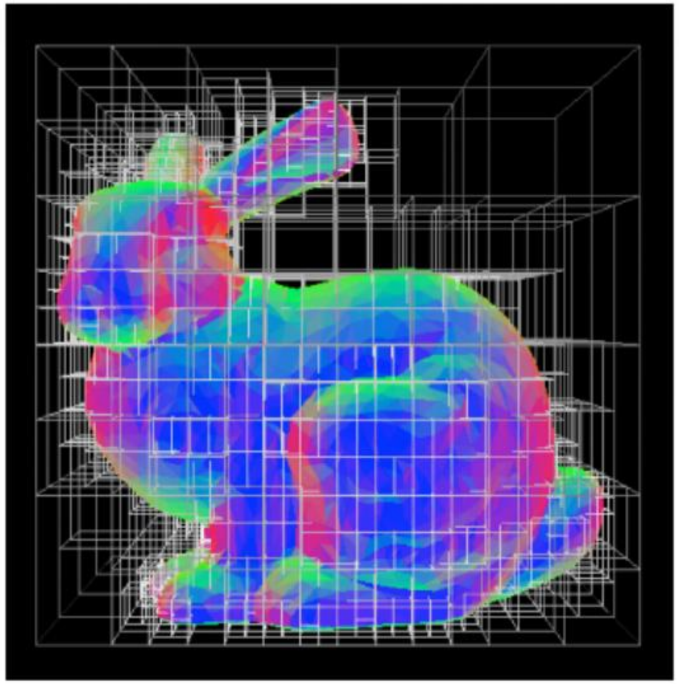


Acceleration structures

- Uniform Grids
- Bounding Volume Hierarchies
- K-D Tree







Uniform Grids

- A very basic structure.
- Divide bounding box to several grids (or cells)
- Each grid has a container of triangles.
- Construction:
  - If a triangle is intersected with a grid, store the triangle in the grid.
- Once the ray goes through a grid, just check the interaction of triangles inside the grid

Uniform Grids - Construction

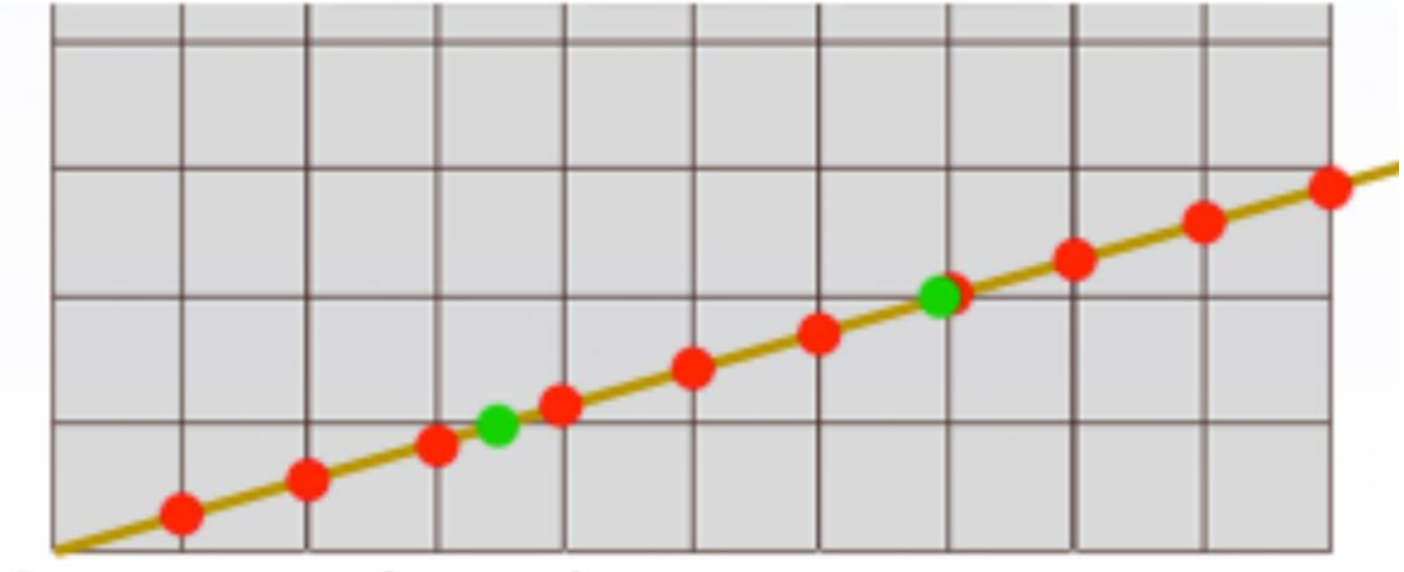
- Construction:
  - If a triangle is intersected with a grid, store the triangle in the grid.

```
for (uint32_t z = zmin; z <= zmax; ++z) {
  for (uint32_t y = ymin; y <= ymax; ++y) {
    for (uint32_t x = xmin; x <= xmax; ++x) {
      uint32_t o = z * res[0] * res[1] + y * res[0] + x;
      cells[o]->insert(triangle);
    }
  }
}
```

More Details: [Introduction to Acceleration Structures (Grid) (scratchapixel.com)]()

Uniform Grids - Traversal

- Intersection between ray and grids
- Consider a 2D case:
  - Similar to the DDA algorithm
  - Find each pixel of the line



More Details: [Introduction to Acceleration Structures (Grid) (scratchapixel.com)]

Uniform Grids - Traversal

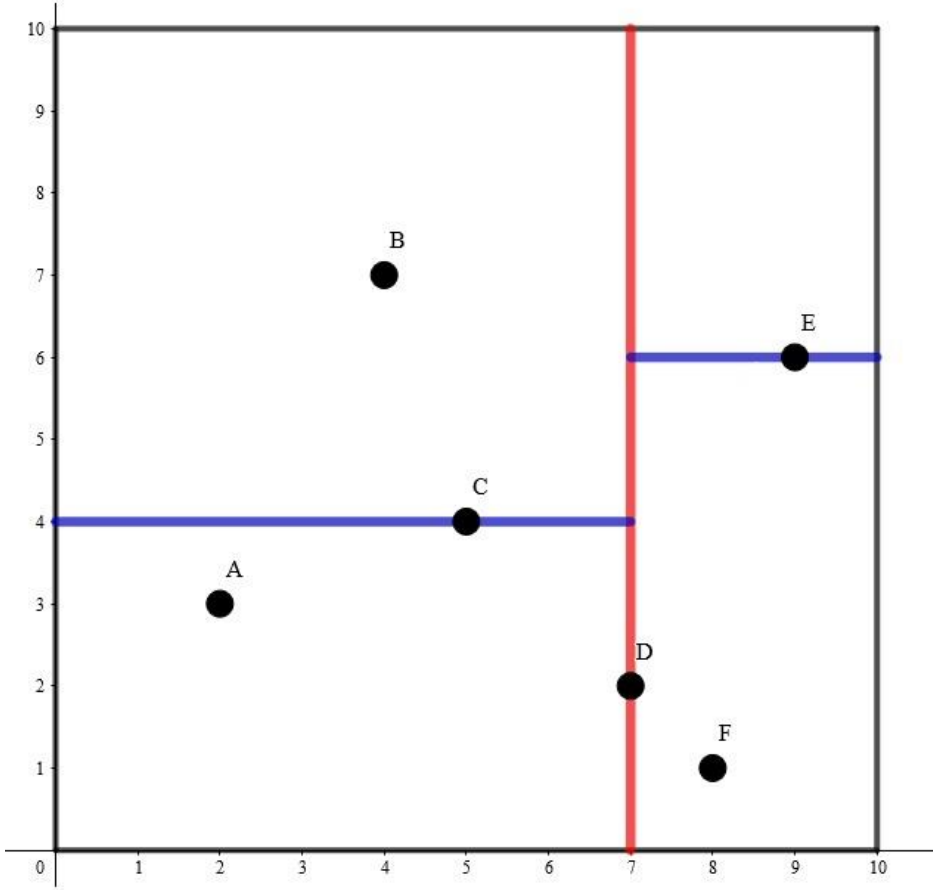
- tMaxX: init with the value of t at which the ray crosses the first vertical voxel boundary
- tDeltaX: how far along the ray we must move (in units of t) for the horizontal component of such a movement to equal the width of a voxel
- stepX: either 1 or -1, the moving direction along the ray
- Similar for tMaxY, tDeltaY and stepY

```
loop{
    if(tMaxX < tMaxY) {
        tMaxX= tMaxX + tDeltaX;
        X= X + stepX;
    } else  {
        tMaxY= tMaxY + tDeltaY;
        Y= Y + stepY;
    }
    NextVoxel(X,Y);
}
```

More Details: [A Fast Voxel Traversal Algorithm - J Amanatides, A Woo - Eurographics, 1987]

K-D Tree (General)

- General K-D Tree, not for ray tracing
- A fast data structure for processing high-dimension information



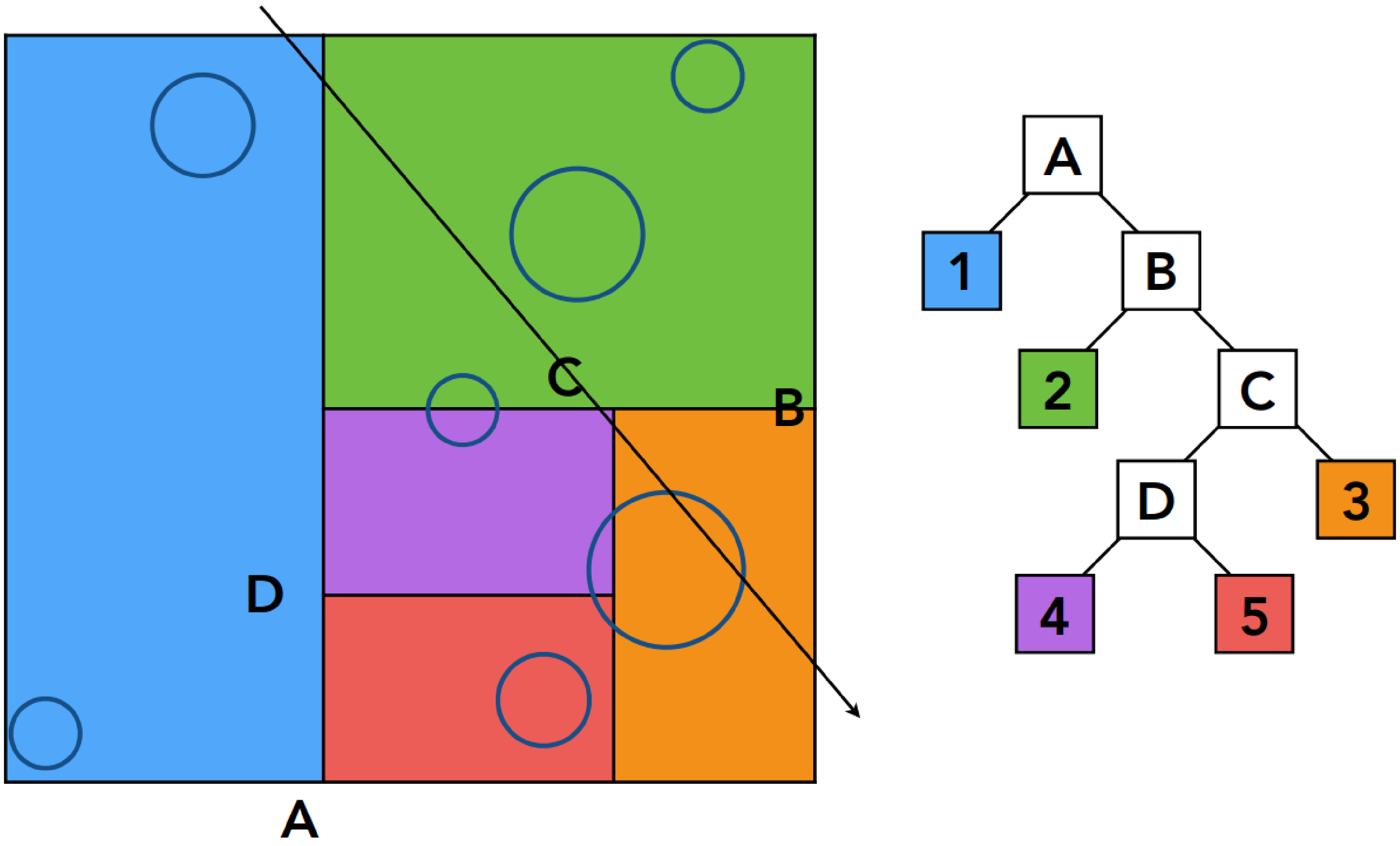
K-D Tree construction

- Use the median to divide
- For ray tracing: may need SAH
- Improving construction:
  - Bottleneck: find the median
  - Quick Selection
  - Median of medians

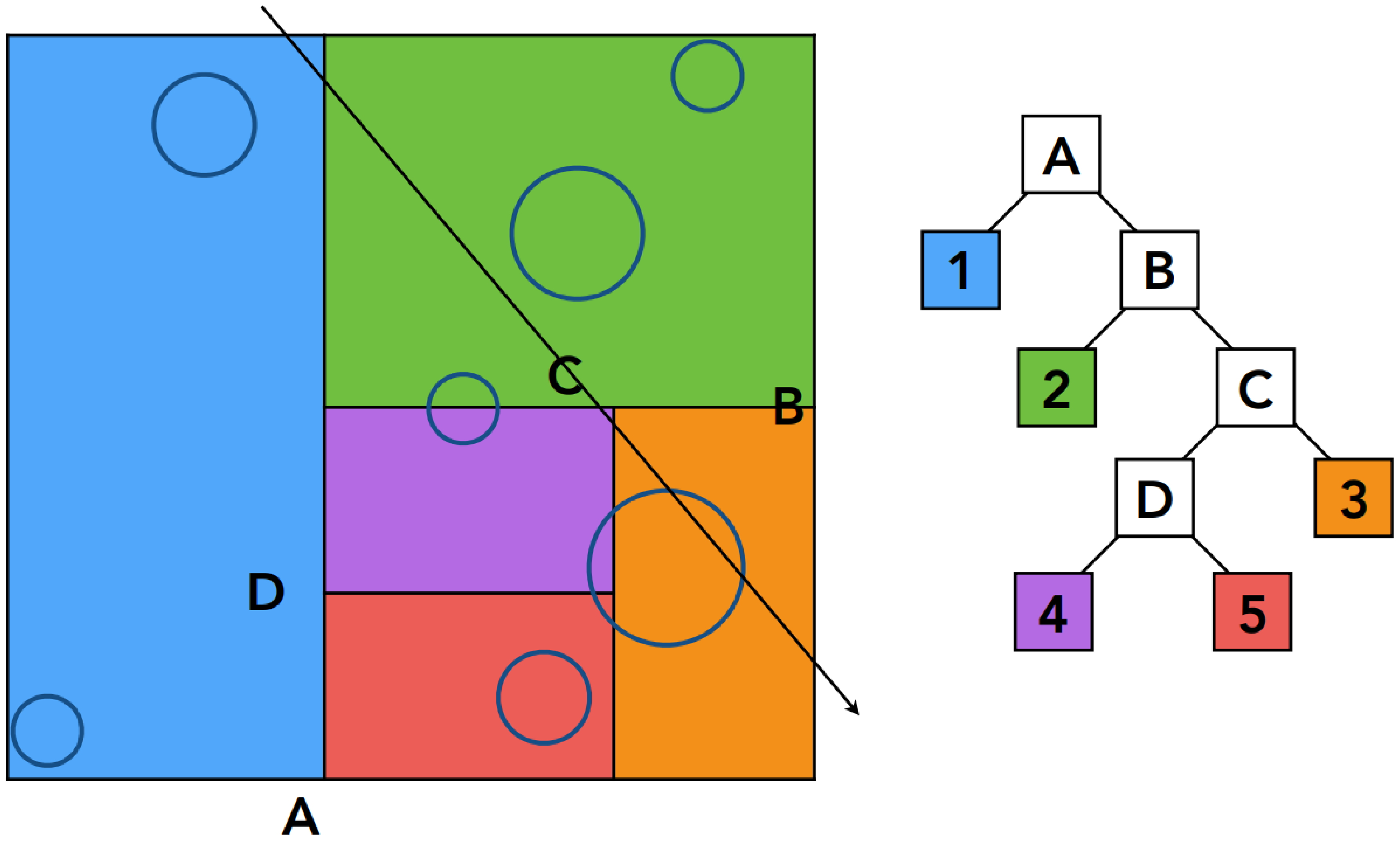
Algorithm $\textsc{BuildKdTree}(P, \mathit{depth})$

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** depth is even
4. **then** Split P with a vertical line ℓ through the median x-coordinate into P_1 (left of or on ℓ) and P_2 (right of ℓ)
5. **else** Split P with a horizontal line ℓ through the median y-coordinate into P_1 (below or on ℓ) and P_2 (above ℓ)
6. $\nu_{\text{left}} \leftarrow \textsc{BuildKdTree}(P_1, \mathit{depth}+1)$
7. $\nu_{\text{right}} \leftarrow \textsc{BuildKdTree}(P_2, \mathit{depth}+1)$
8. Create a node ν storing ℓ, make ν_{left} the left child of ν, and make ν_{right} the right child of ν.
9. **return** ν

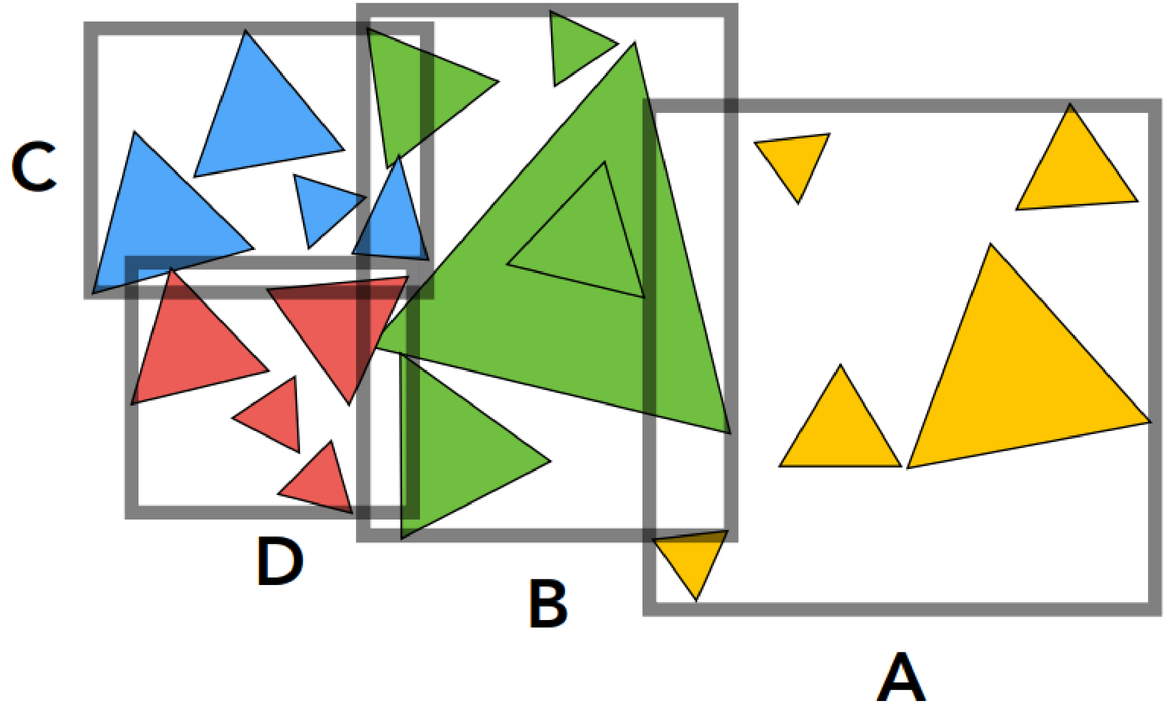
K-D Tree for Ray Tracing

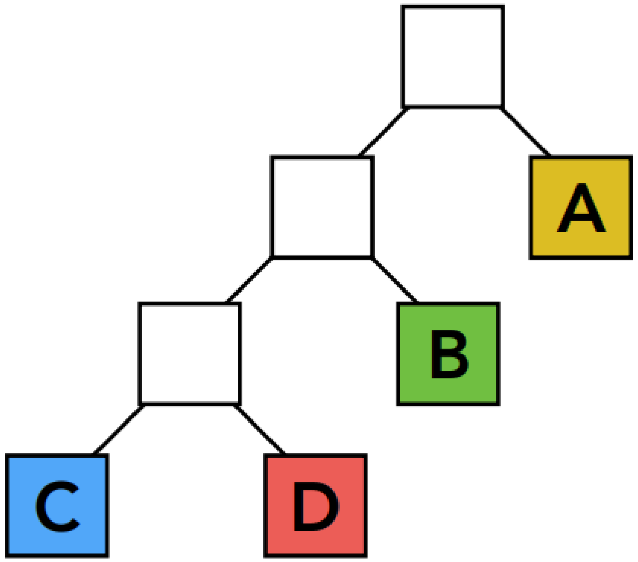


K-D Tree for Ray Tracing



Bounding Volume Hierarchy (BVH)





BVH Node

- Two child nodes
- An AABB bounding box
- Triangles (only store in leaf node)

```
struct Node {
  Node *c[2];
  AABB box;
  std::vector<int> triangles;
};
```

BVH Construction

```cpp
void build(Node *&p, std::vector<int> &triangles, const std::vector<int> &verticesIdx,
           const std::vector<Eigen::Vector3f> &vertices, int l, int r, int depth) {

  p = newNode(triangles, verticesIdx, vertices, l, r);
  if (r - l <= THRESHOLD) {
    p->triangles = std::vector<int>(triangles.begin() + l, triangles.begin() + r + 1);
    return;
  }
  int axis = 0; // choose an axis to divide
  int mid = (l + r) / 2;
  std::nth_element(triangles.begin() + l, triangles.begin() + mid, triangles.begin() + r + 1,
  [&](int a, int b) {
    return aMid < bMid; // compare by middle point or SAH
  }); // divide into two parts
  build(p->c[0], triangles, verticesIdx, vertices, l, mid, depth + 1);
  build(p->c[1], triangles, verticesIdx, vertices, mid + 1, r, depth + 1);
}
```

BVH Traversal

```
void traverse(Node *p, Interaction &interaction, const Ray &ray) {
  if (!p) return;
  Float tIn, tOut;
  if (!p->box.rayIntersection(ray, tIn, tOut)) return; // not hit the bounding box
  if (p->isLeaf()) { // traverse the triangles stored in the leaf node
    for (int idx : p->triangles) {
      Interaction it;
      if (intersect(it, ray, idx) && (interaction.t == -1 || it.t < interaction.t)) {
        interaction = it;
      }
    }
    return;
  }
  traverse(p->c[0], interaction, ray);
  traverse(p->c[1], interaction, ray);
}
```

References

- CS171 lecture 10 - efficient ray-geometry
- Introduction to Acceleration Structures (Grid) (scratchapixel.com)
- A Fast Voxel Traversal Algorithm - J Amanatides, A Woo - Eurographics, 1987
- GAMES101_Lecture_14 (ucsb.edu)

Thanks