# 02562 Rendering - Introduction

## Spatial Data Structures for Ray Tracing

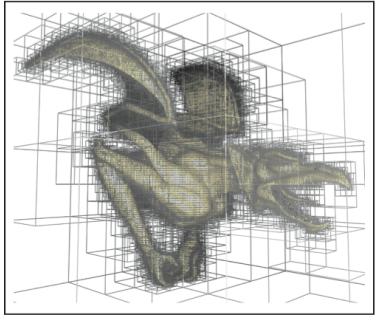
Jeppe Revall Frisvad

October 2023

# Spatial subdivision

- To model arbitrary geometry with triangles, we need many triangles.
- A million triangles and a million pixels are common numbers.
- Testing all triangles for all pixels requires $10^{12}$ ray-triangle intersection tests.
- If we do a million tests per millisecond, it will still take more than 15 minutes.
- This is prohibitive. We need to find the relevant triangles.

- Spatial data structures offer logarithmic complexity instead of linear.
- A million tests become twenty operations $\left(\log_2 10^6 \approx 20\right)$.
- 15 minutes become 20 milliseconds.



Gargoyle embedded in oct tree [Hughes et al. 2014].

# Spatial data structures

- Grid.
  - Mapping from world space position to grid cell.
  - Key challenge: ray traversal of the grid.
- Oct tree (octree).
  - Using subdivision centers that subdivide space into eight octants.
  - Key challenge: book keeping with 8 child pointers per node.
- Bounding volume hierarchy (BVH)
  - Most often using axis-aligned bounding boxes (AABB).
  - Key challenge: traversal cost optimization.
- Binary space partitioning tree (BSP tree).
  - Most often restricted to axis-aligned planes ($k$d tree).
  - Key challenge: splitting plane selection.

# Grid

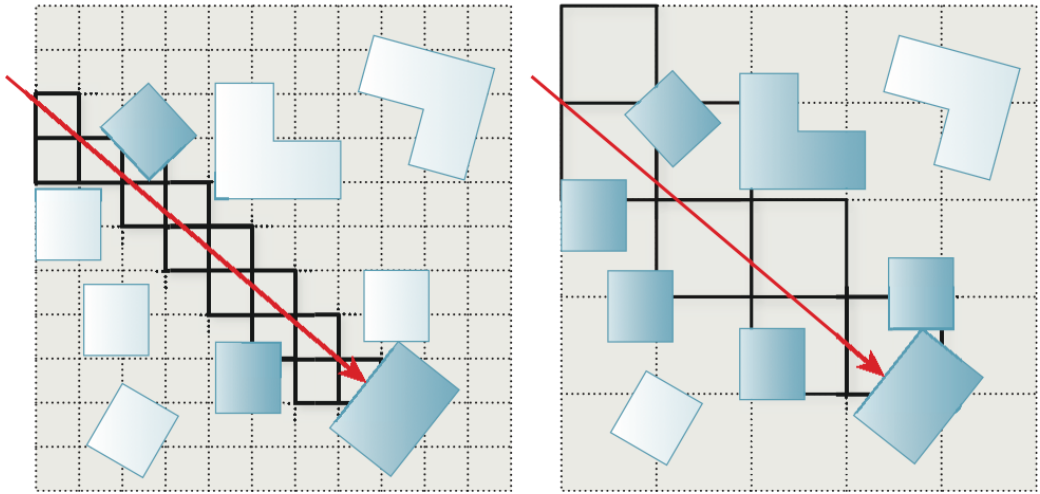


Figure 37.16 of the textbook [Hughes et al. 2014].

# Quadtree (2d version of the octree)

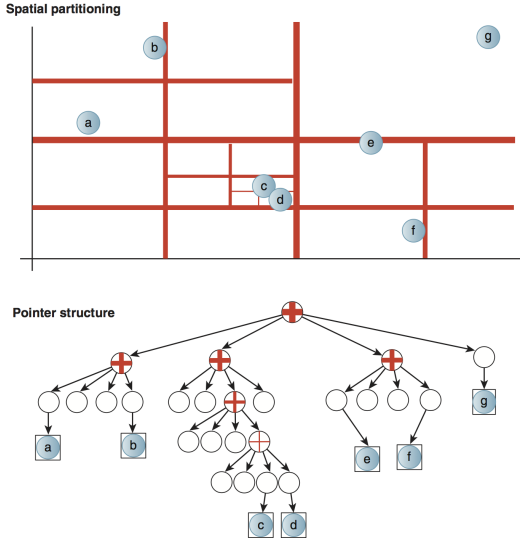



Figure 37.11 of the textbook [Hughes et al. 2014].

# Bounding volume hierarchy (BVH)

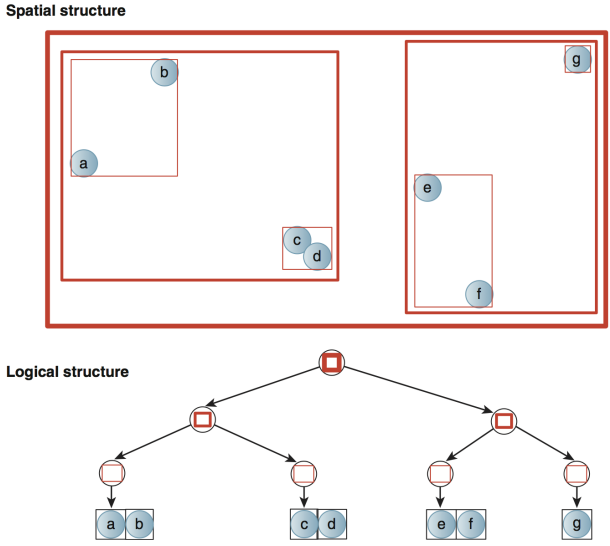
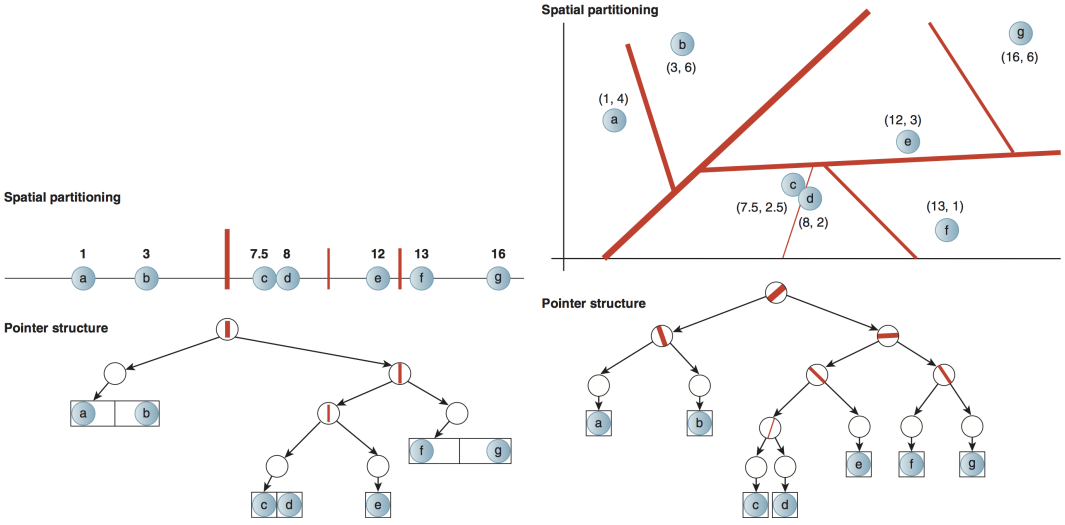


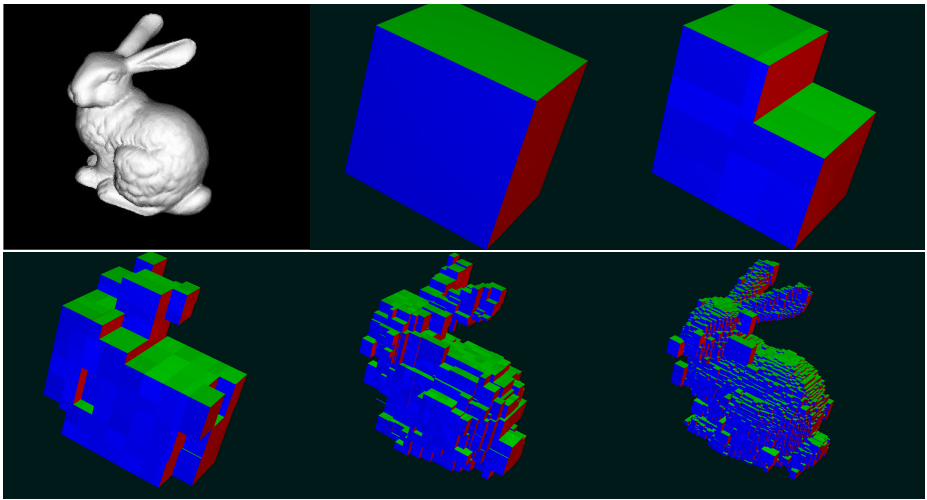

Figure 37.12 of the textbook [Hughes et al. 2014].

# Binary space partitioning tree (BSP tree)




Figures 37.8 and 37.9 of the textbook [Hughes et al. 2014].

# Axis-aligned BSP tree (*k*d tree)

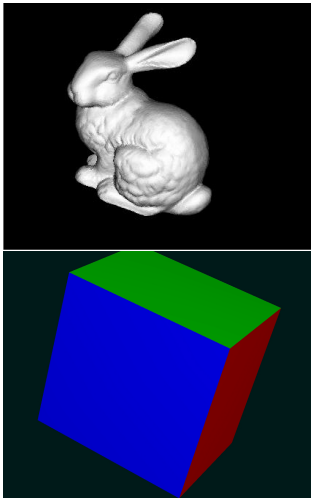


The surface area heuristic (SAH) for selecting splitting planes:

$\text{cost} = \text{SA}_{\text{left}}\Delta_{\text{left}} + \text{SA}_{\text{right}}\Delta_{\text{right}}$ , where $\Delta$ is triangle count.

# Avoid tracing rays that do not intersect the scene bounding box

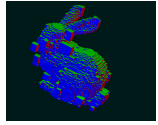
```
struct Aabb {
  min: vec4f,
  max: vec4f,
};
@group(0) @binding(i) var<uniform> aabb: Aabb;


fn intersect_min_max(r: ptr<function, Ray>) -> bool
{
  var tmin = 1.0e32f;
  var tmax = -1.0e32f;
  for(var i = 0u; i < 3u; i++) {
    if(abs((*r).direction[i]) > 1.0e-8f) {
      let p1 = (aabb.min[i] - (*r).origin[i])/(*r).direction[i];
      let p2 = (aabb.max[i] - (*r).origin[i])/(*r).direction[i];
      let pmin = min(p1, p2);
      let pmax = max(p1, p2);
      tmin = min(tmin, pmin);
      tmax = max(tmax, pmax);
    }
  }
  if(tmin > tmax || tmin > (*r).tmax || tmax < (*r).tmin) {
    return false;
  }
  (*r).tmin = max(tmin - 1.0e-4f, (*r).tmin);
  (*r).tmax = min(tmax + 1.0e-4f, (*r).tmax);
  return true;
}
```



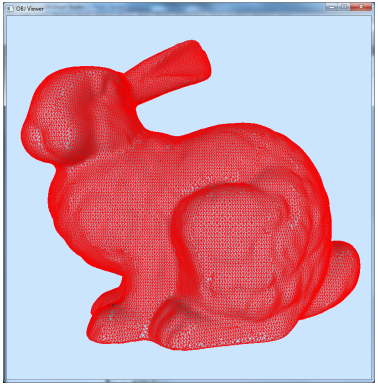
# Traversing the BSP tree

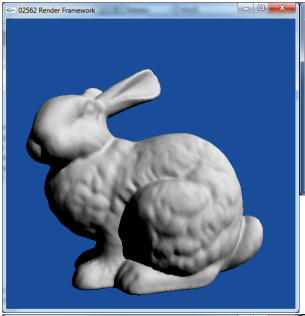
```
fn intersect_trimesh(r: ptr<function, Ray>, hit: ptr<function, HitInfo>) -> bool
{
  var branch_lvl = 0u;
  var near_node = 0u;
  var far_node = 0u;
  var t = 0.0f;
  var node = 0u;
  for(var i = 0u; i <= MAX_LEVEL; i++) {
    let tree_node = bspTree[node];
    let node_axis_leaf = tree_node.x&3u;
    if(node_axis_leaf == BSP_LEAF) {
      // A leaf was found
      ⋮
    }
    let axis_direction = (*r).direction[node_axis_leaf];
    let axis_origin = (*r).origin[node_axis_leaf];
    if(axis_direction >= 0.0f) {
      near_node = tree_node.z; // left
      far_node = tree_node.w;  // right
    }
    else {
      near_node = tree_node.w; // right
      far_node = tree_node.z;  // left
    }
    let node_plane = bspPlanes[node];
    let denom = select(axis_direction, 1.0e-8f, abs(axis_direction) < 1.0e-8f);
    t = (node_plane - axis_origin)/denom;
    if(t > (*r).tmax) { node = near_node; }
    else if(t < (*r).tmin) { node = far_node; }
    else {
      branch_node[branch_lvl].x = i;
      branch_node[branch_lvl].y = far_node;
      branch_ray[branch_lvl].x = t;
      branch_ray[branch_lvl].y = (*r).tmax;
      branch_lvl++;
      (*r).tmax = t;
      node = near_node;
    }
  }
  return false;
}
```

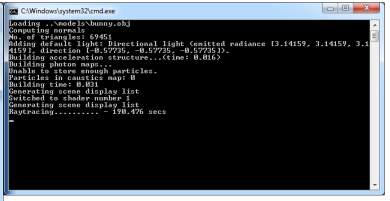
```
@group(0) @binding(i1) var<storage> treeIds: array<u32>;
@group(0) @binding(i2) var<storage> bspTree: array<vec4u>;
@group(0) @binding(i3) var<storage> bspPlanes: array<f32>;

const MAX_LEVEL = 20u;
const BSP_LEAF = 3u;
var<private> branch_node: array<vec2u, MAX_LEVEL>;
var<private> branch_ray: array<vec2f, MAX_LEVEL>;
```

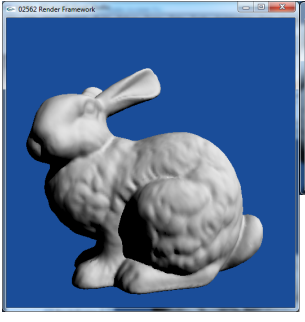


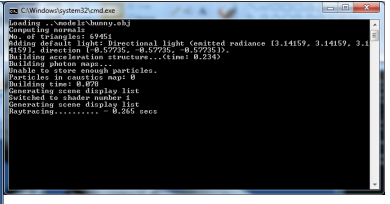
```
// A leaf was found
let node_count = tree_node.x>>2u;
let node_id = tree_node.y;
var found = false;
for(var j = 0u; j < node_count; j++) {
  let obj_idx = treeIds[node_id + j];
  if(intersect_triangle(*r, hit, obj_idx)) {
    (*r).tmax = (*hit).dist;
    found = true;
  }
}
if(found) { return true; }
else if(branch_lvl == 0u) { return false; }
else {
  branch_lvl--;
  i = branch_node[branch_lvl].x;
  node = branch_node[branch_lvl].y;
  (*r).tmin = branch_ray[branch_lvl].x;
  (*r).tmax = branch_ray[branch_lvl].y;
  continue;
}
```


OBJ Viewer

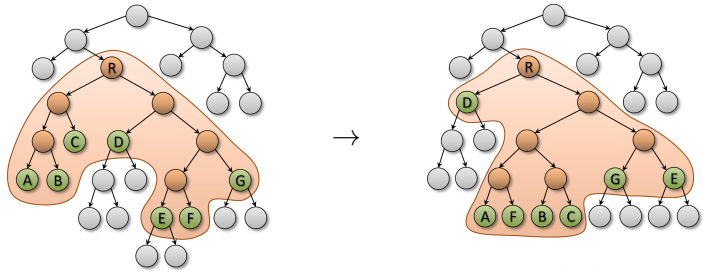

02562 Render Framework


C:\Windows\system32\cmd.exe
Loading ..\models\bunny.obj
Computing normals
No. of triangles: 69451
Adding default light: Directional light (emitted radiance (3.14159, 3.14159, 3.14159), direction (-0.57735, -0.57735, -0.57735)).
Building acceleration structure...(time: 0.016)
Building photon maps...
Unable to store enough particles.
Particles in caustics map: 0
Building time: 0.031
Generating scene display list
Switched to shader number 1
Generating scene display list
Raytracing..... - 190.476 secs


02562 Render Framework


C:\Windows\system32\cmd.exe
Loading ..\models\bunny.obj
Computing normals
No. of triangles: 69451
Adding default light: Directional light (emitted radiance (3.14159, 3.14159, 3.14159), direction (-0.57735, -0.57735, -0.57735)).
Building acceleration structure...(time: 0.234)
Building photon maps...
Unable to store enough particles.
Particles in caustics map: 0
Building time: 0.078
Generating scene display list
Switched to shader number 1
Generating scene display list
Raytracing..... - 0.265 secs

# Treelet restructuring bounding volume hierarchy for spatial subdivision




treelet → reorganized treelet

- Practical GPU-based bounding volume hierarchy (BVH) builder.
  1. Build a low-quality BVH (parallel linear BVH).
  2. Optimize node topology by parallel treelet restructuring (keeping leaves and their subtrees intact).
  3. Post-process for fast traversal.

References

- Karras, T., and Aila, T. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of HPG 2013*, pp. 89–99. ACM, July 2013.

# Exercises

- Use an axis-aligned BSP tree to accelerate ray tracing of triangle meshes and explain how it works.
- Do interleaving of buffers to stay within the current WebGPU limit of 8 storage buffers per shader stage.
- Combine triangle mesh rendering with rendering of mathematically defined spheres (ray-sphere intersection).