Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP

Gustaf Waldemarson & Michael Doggett

Arm Sweden AB, Lund University, Faculty of Engineering (LTH)

Introduction

Fast BVH construction is necessary to ensure fast ray-tracing, so a parallel build approach is needed. Thus, in this paper we propose a method of using OpenMP tasking to parallelize the splitting algorithm and thus improve build performance, improving the construction time by between 3 and 5 times on an 8-core machine with a minimal amount of work and negligible quality reduction of the final BVH.

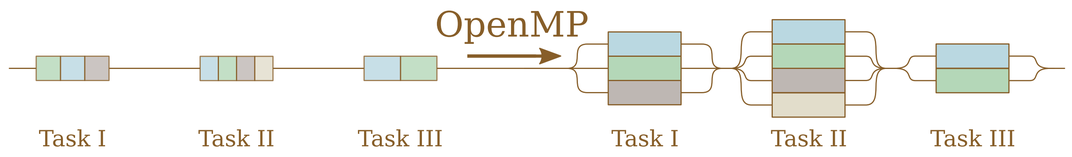


Figure 1: Graphical visualization of the typical OpenMP usage for splitting up tasks into parallel regions.

Tasking Variants

NoOpenMP / NoTasking

```
void sbvh_build(const vector<Reference> &refs)
{
    if (create_leaf_p(refs)) return leaf_node();
    ObjectSplit object = object_cost(refs);
    SpatialSplit spatial = spatial_cost(refs);
    NodeSplit split;
    if (spatial.cost < object.cost)
        split = spatial_split();
    else
        split = object_split();
    sbvh_build(split.left_references);
    sbvh_build(split.right_references);
}
```

Tasking / Default

```
#pragma omp task
    sbvh_build(split.left_references);
#pragma omp task
    sbvh_build(split.right_references);
#pragma omp taskwait
```

Figure 2: High level overview of the SBVH construction algorithm by Stich et al., 2009, and how OpenMP tasking is applied to it.

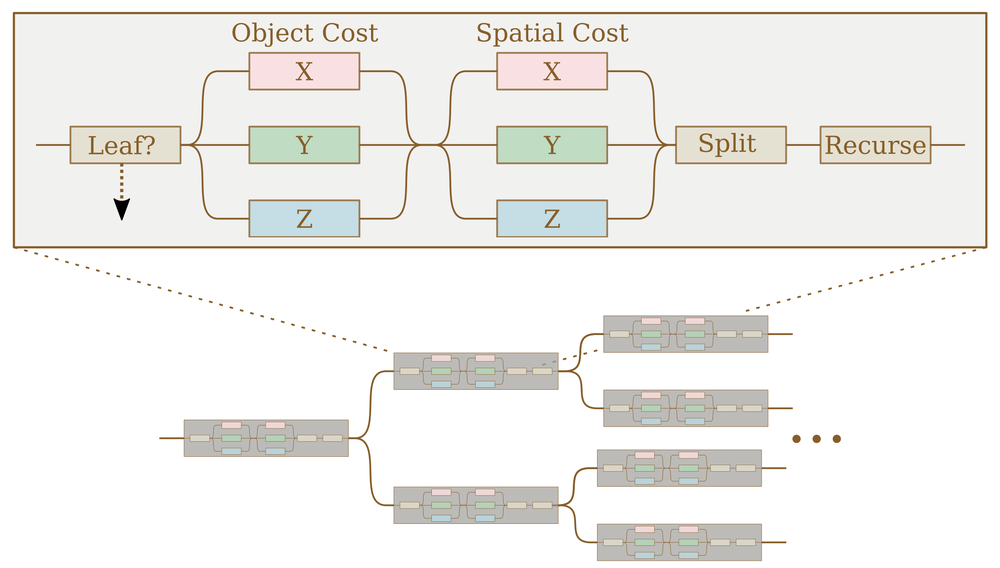


Figure 3: Visualization of how the parallel tasks are structured for our SBVH algorithm.

Axis Search Variants

NoOpenMP

```
SpatialSplit split_spatial()
{
for (size_t ax = 0; ax < 3; ++ax)
{
}
```

ParallelFor

```
#pragma omp parallel for
for (size_t ax = 0; ax < 3; ++ax)
{
}
```

Tasks

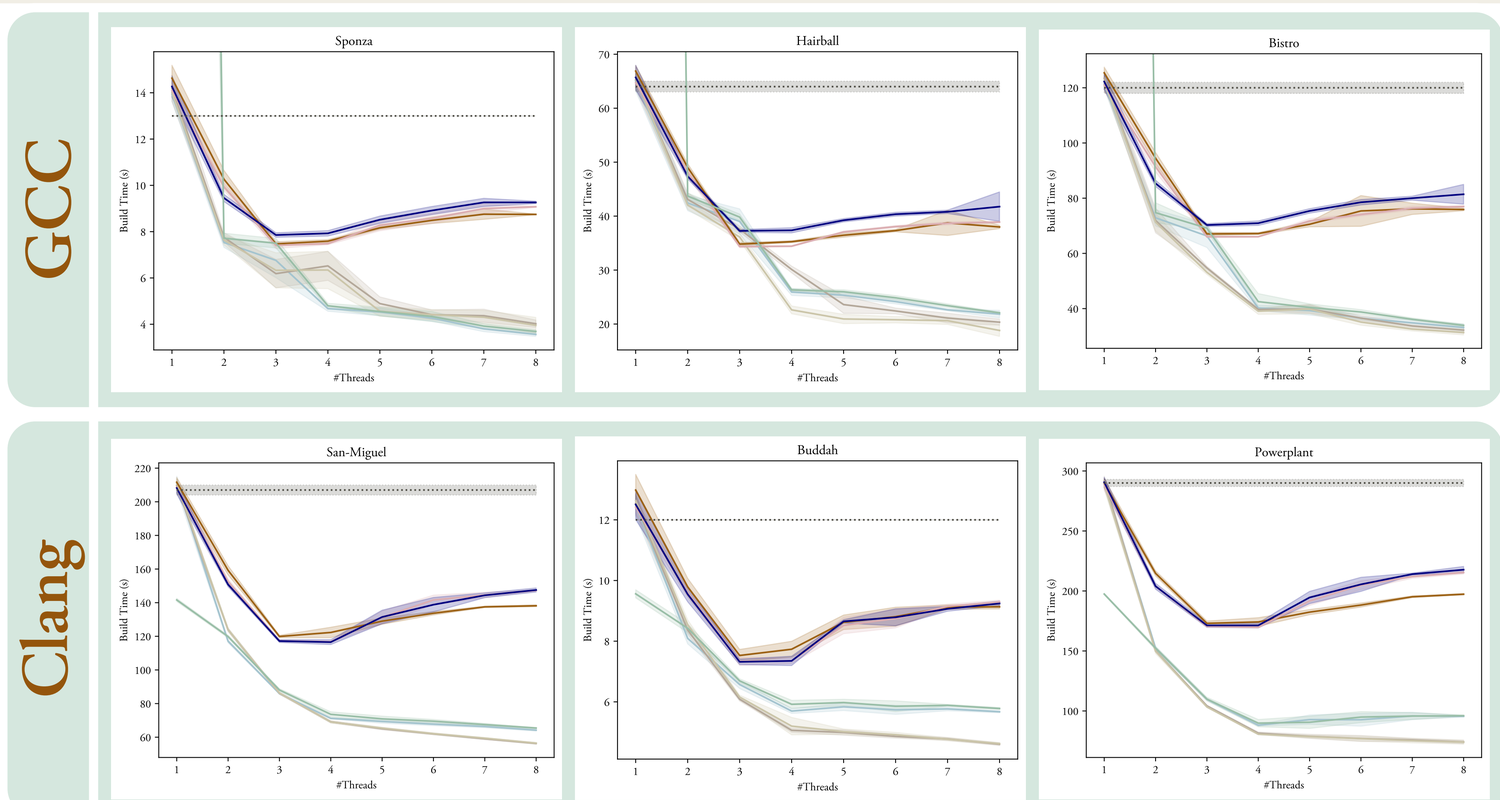
```
for (size_t ax = 0; ax < 3; ++ax)
#pragma omp task
{
}
#pragma omp taskwait
```

Taskloop

```
#pragma omp taskloop
for (size_t ax = 0; ax < 3; ++ax)
{
}
```

Figure 4: Beyond OpenMP tasking from figure 2, the object and spatial axis searches may also be accelerated, as each axis is independent. However, there are numerous ways to use OpenMP when tasking is already being applied elsewhere. Thus, the following OpenMP `#pragma` variants were applied to these to loop to find the best way. Thus, in total we have 8 different variations: With or without tasking, and 4 different ways to accelerate the inner axis `for`-loops.

Scaling Results



Rendering Times

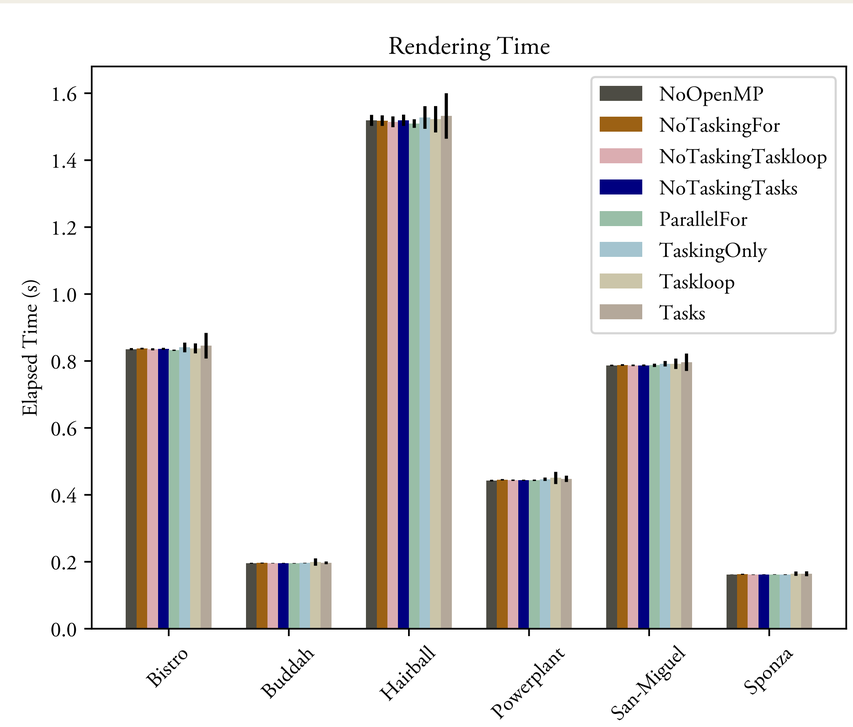


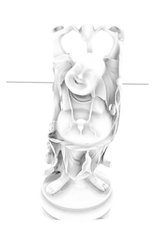
Figure 5: Estimated rendering times for each scene and variant.

Scenes

Sponza: 393 meshes, 262 267 triangles.

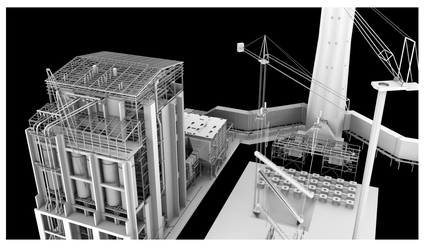
Bistro: 1 mesh, 3 847 246 triangles.

Hairball: 2 meshes, 2 880 002 triangles.



Buddah: 1 mesh, 1 087 720 triangles.

San-Miguel: 287 meshes, 9 980 699 triangles.



Powerplant: 21 meshes, 12 759 246 triangles.

References

Dagum, L., & Menon, R. (1998). Openmp: An industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, *5*(1), 46–55.

Stich, M., Friedrich, H., & Dietrich, A. (2009). Spatial splits in bounding volume hierarchies. *Proceedings of the Conference on High Performance Graphics 2009*, 7–13. https://doi.org/10.1145/1572769.1572771