Occlusion culling?
A technique that bypasses hidden objects in rendering and thereby accelerates performance.
- Previously visible objects are chosen as occluders
- Bounding boxes of the occludee are occlusion-tested

Occlusion culling requires rendering and thereby a rasterization overhead for the occlusion test.

Previous Approaches
CHC++ (Wettach 2008)
- Hierarchical GPU occlusion query
Raster occlusion culling [Wettach 2005, Wettach 2016]
- Direct writing to a GPU buffer with early Z
- Indirect multidraw to hide read-back latency

Our Approach
We use a geometry-level BVH to batch-test the visibilities of the occludees with iterative visibility tests through top-down traversal. This allows us to avoid brute-force tests for the individual occludees, achieving real-time performance even for large-scale scenes (more than dozens of thousands objects).

Batch raster culling with fewer drawcalls:
- Top-down traversal within a pair of top and bottom depths in BVH
- Significant reduction in the number of effective occlusion culling

Results
- Evaluated techniques: no culling (NOCULL), ideal culling (IF), view frustum culling (VFC), raster occlusion culling (ROC), iterative occlusion culling (IOC)
- Better performance in large-scale scenes with the same culling rate of ROC

Motivation
Raster occlusion culling (ROC) requires to render the bounding boxes of all objects not seen in the previous frame to test the visibilities of the occludees. ROC works well up to medium-size scenes, but does not scale well with massive scenes due to the excessive rasterization overhead for the occlusion test.

Challenges
Scalable raster occlusion culling with light-weight test
- Fewer draw calls to determine the visibility of many objects
- Reduction of redundant rasterization for the occlusion culling