INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2015-04-29

EXAMENSARBETE Real-time screen space reflections and refractions using sparse voxel octrees
STUDENT Filip Nilsson
HANDLEDARE Michael Doggett (LTH)
EXAMINATOR Tomas Akenine-Möller (LTH)

Reflections and refractions using a real-time ray tracer

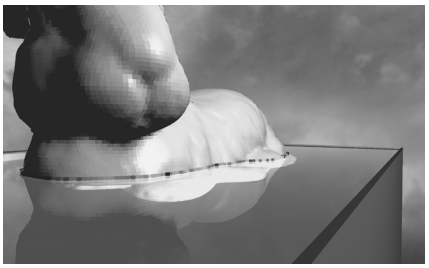
POPULÄRVETENSKAPLIG SAMMANFATTNING **Filip Nilsson**

To reflect and refract objects in a correct way ray tracing has to be used. This is very computationally demanding in real-time applications. However, by combining three different methods the performance can vastly improve.

What we see as reflections and refractions in games and other real-time applications is in most cases just an image being sampled to create an illusion of the effect. When rendering a frame on the screen only objects in front of the camera is computed, so it's very hard to reflect the rest of the scene. To do this we have to use ray tracing instead. This technique works by tracing rays through the camera into different parts of the scene. These rays can be compared to photon rays, but instead of coming from the scene we trace them backwards from the camera. We can then emulate real-world effects realistically by for example letting them bounce to get reflections. Ray tracing is very computationally demanding, as a single movie frame can take days to render. In real-time applications we normally want at least 24 frames per second to get smooth motions, so a lot of optimizations have to be made.

The octree structure

To see what a ray hits we perform intersection tests with objects in the scene. In the most basic form of ray tracing we test a ray against every triangle in the scene and see which one we hit first. This is unfeasible as a scene normally have thousands to millions triangles. We instead use an octree, which is a tree structure consisting of nodes containing voxels. A voxel is a cube that takes up space in the scene. The first node in the tree is the root node, which contains the whole scene. This node has 8 children, each taking up 1/8 of its space. The children in turn have their own 8 children, and so it continues until the tree has reached a certain depth. The tree is also sparse, meaning that if a voxel doesn't contain any triangles it will not be created, saving space.



The water both reflects and refracts the scene.

The voxelization process

To find out which triangles belong to which voxels three tests are performed. They vary in performance and accuracy, where they can be faster but might include triangles that are not in the voxel, or more expensive and accurate. The cheap tests are useful for ruling out the vast majority of triangles, leaving a smaller amount for the expensive ones.

On the lowest depth the triangles in a voxel are condensed into a polyhedron based on the average shapes, colours and other attributes of the triangles. This is all done once during the preparation part of the program.

The traversal algorithm

Once we start rendering frames we use an algorithm made for this specific type of octree to quickly find what voxel will be hit. This is done by first testing which of the 8 children of the root node that are intersected by the ray. In order of intersection their children are then tested in a similar way until we reach a node on the max depth of the tree that is intersected, and we render the polyhedron of that voxel.

This reduces the amount of intersection tests to a fraction of the total number of triangles, increasing ray tracing performance enough for real-time. Using reflections and refractions decrease the performance, but are visually correct. This work shows the use of real-time ray tracing to create more realistic reflections and refractions than typical rasterization algorithms can achieve.