Implementation of viewing frustum culling in rasterization pipeline

Victor Hansjons Vegeborn
victorhv@kth.se

May 15, 2017

1 Background

In real-time graphics, the rasterization process is essentially transforming and projecting vertices representing a tree-dimensional world onto a two-dimensional plane i.e a camera lens from which we may view this world. The rasterization pipeline consists of different sections or algorithms which play specific roles in turning the tree-dimensional world into something viewable, and some to ease the computational demand on the system. One such computational-easing technique is clipping [1, 3].

When computing the tree-dimensional world, many of these vertices will often be offset outside of the viewable space. The need to compute these vertecies and the lines they make up will only be costly and generate strange rendering problems. By performing clipping they are no longer computed by the rasterizer and better performance is achived. However, when a polygon triangle is only partially outside the camera lens the clipping becomes more complex.

2 Problem

If the viewing volume (in the shape of a pyramid with its peak at the cameras position) is infinitely large and the rasterizer still computes vertices outside this volume, the resulting rendered frame may look incorrect depending on the camera direction. The world may be rendered up-side down or even look unrecognizable. By limiting the viewing volume (by truncating the pyramid i.e the viewing frustum) and implementing a clipping algorithm, can these rendering problem be eliminated?

3 Implementation

The Sutherland-Hodman algorithm, is a polygon-clipping algorithm that utilizes the intersections between polygon-lines and clipping-lines [4, 2]. The clipping-line is extended to infinity in both directions and divides a two-dimensional plane into a including and excluding area. Depending on whether there is a intersection, a new vertex is inserted and the part of the polygon in the excluding area is discarded. By extending the algorithm to handle clipping-planes instead of clipping-lines, the algorithm can be utilized as a viewing frustum culling algorithm [4].

4 Step-by-step plan

1. For each frame, calculate the frustum.
2. Find pseudo code for the Sutherland-Hodman algorithm [4] and extend it to handle planes instead of lines.
3. Extend lab 3 and implement the extended sutherland-hodman algorithm as a viewing frustrum culling algorithm.

5 References

References

[1] U. Assarsson, *View Frustum Culling and Animated Ray Tracing: Improvements and Methodological Considerations*, Chalmers, 2001, *URL:http://www.cse.chalmers.se/ũffe/lic.pdf*

[2] C. Ericson, *Real-Time Collision Detection*, [chapter 8.3.4], 2005, ISBN: 1-55860-732-3.

[3] Wikipedia, *Clipping (compute graphics)*, URL:https://en.wikipedia.org/wiki/Clipping_(computer_graphics).

[4] Wikipedia, *Sutherland–Hodgman algorithm*, *URL:https://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman*$_a$*lgorithm*.