Implementation of viewing frustum culling in a rasterization pipeline

Victor Hansjons Vegeborn
victorhv@kth.se

DH2323 Computer Graphics and Interaction
Project Report
KTH Royal Institute of Technology

1 Introduction

A rasterization pipeline is arguably useless if the point of view can not be changed without rendering issues. This is the case with pipelines that does not contain some viewing frustum culling technique. If all of the vertices that make up the interactive world were to be transformed onto the view plane, even when they are out of view, the rasterization pipeline will produce unwanted frames. By disregarding these vertices, a coherent world can be rendered when the camera is moved by interaction.

In this project, this problem is solved for the rasterization lab of the course by calculating a frustum, relative to pinhole camera, for every frame. Moreover, the culling is performed by an extended Sutherland-Hodgman algorithm for usage within a three dimensional space.

The result of the project is presented with images taken while running the project executable. A video recorded of the same session, can be watched on Youtube at `https://youtu.be/WbogZZgy6co` and better captures the viewing frustum culling in action.

2 Technical background

2.a Pinhole camera configuration

The pinhole camera has a position $\mathbf{c}$ and world-relative directions; forward $\mathbf{f}$, right $\mathbf{r}$ and down $\mathbf{d}$. In addition, the view plane is computed from the screen width W, screen height H as well as the focal length F, which are given in pixels. As the frustum is relative to the camera configuration and its momentary

directions, the width of the frustum near plane, which is needed to compute the frustum, was pre-computed as:

$$W_{width} = \frac{W}{F} d_{near} \quad (1)$$

where d_{near} is the predefined distance from the camera to the near plane. This follows from the simplifications of the geometrical relationships where $vFOV$ is the vertical field of view:

$$vFOV = 2 \arctan\left(\frac{W}{2F}\right) \quad (2)$$

$$W_{width} = 2 \tan\left(\frac{vFOV}{2}\right) d_{near} \quad (3)$$

2.b Frustum

The viewing frustum is made up of six planes P_i with their normals $\mathbf{n_i}$ pointing inwards. The normal direction is important, as it is used when detecting whether a vertex is within the volume. As the planes span infinitely in their local axis, the planes can be represented with three values, its normal, a point $\mathbf{p0_i} \in P_i$ and the offset from the origin $d_i = -\mathbf{n_i} \cdot \mathbf{p0_i}$.

Arbitrary planes are given by the equation $ax+by+cz+d = 0$ where $\mathbf{n} = (a, b, c)$ and $\mathbf{p0} = (x, y, z)$. The simplified equation for d_i holds for all planes.

2.c Signed distance

The signed distance is used to determine the orientation of a vertex relative to a plane. let $D(\mathbf{v}, P)$ be the signed distance, where $\mathbf{v}$ is some arbitrary vertex, and P some arbitrary plane as defined in section 2.b. Calculating the signed distance is done by:

$$S(\mathbf{v}, P) = \mathbf{n} \cdot \mathbf{v} + d \quad (4)$$

where $\mathbf{n}, d \in P$. If the planes normal $\mathbf{n}$ has a orientation towards the inside of the frustum, then $0 \leq D(\mathbf{v}, P)$ is equivalent to $\mathbf{v}$ is on the inside of the frustum.

2.d Intersection

If two points $\mathbf{s}, \mathbf{e}$ are on opposite sides of a plane P and form a line that intersects the plane at a point $\mathbf{p}$, the computation of such a point can be done algebraically. The equations for a line and a plane can be expressed as:

$$(\mathbf{p} - \mathbf{p0}) \cdot \mathbf{n} = 0, \ \mathbf{p0}, \mathbf{n} \in P \quad (5)$$

$$\mathbf{p} = x\hat{\mathbf{l}} + \mathbf{s}, \ \hat{\mathbf{l}} = \frac{\mathbf{e}-\mathbf{s}}{|\mathbf{e}-\mathbf{s}|}, \text{ and } x \in \mathbb{R} \quad (6)$$

By substituting $\mathbf{p}$ in the equations, we obtain the scalar x, and the intersecting point $\mathbf{p}$ is given by:

$$\mathbf{p} = \mathbf{I}(\mathbf{s}, \mathbf{e}, P) = \frac{(\mathbf{p0}-\mathbf{s})\cdot\mathbf{n}}{\hat{\mathbf{l}}\cdot\mathbf{n}}\hat{\mathbf{l}} + \mathbf{s} \quad (7)$$

Note that the denominator of $\mathbf{I}(\mathbf{s}, \mathbf{e}, P)$ can be equal to zero. If this is the case, the line parallel to the plane. How this is handled is later explained in the implementation section.

2.e The extended Sutherland-Hodgeman algorithm

The sutherland-hodgeman algorithm is used for clipping two dimensional polygons. In this project, the algorithm was extended to tree dimensions in order to work with planes instead of lines. There is one drawback of the algorithm, it does not work on polygons that are concave. As the polygons in this project are rendered per triangle, this limitation does not imply any issues. Below the algorithm is described in pseudo code and is written with the definitions of planes and functions to be coherent with the above sections.

```
input  : A set of vertices V = {v1, v2, v3} that together forms a
         triangle and the six planes P of the frustum
output: A set of vertices V' that forms the clipped polygon
1  V' ← V foreach Plane P_i ∈ P do
2  |  V'' ← V'
3  |  V' ← ∅
4  |  s ← v3 ∈ V''
5  |  foreach vertex e ∈ V'' do
6  |  |  if 0 ≤ S(e, P_i) then
7  |  |  |  if 0 ≤ S(s, P_i) then
8  |  |  |  |  V' ← I(e, s, P_i)
9  |  |  |  end
10 |  |  |  V' ← e
11 |  |  else if 0 ≤ S(s, P_i) then
12 |  |  |  V' ← I(s, e, P_i)
13 |  |  end
14 |  |  s ← e
15 |  end
16 |  if V' = ∅ then
17 |  |  break
18 |  end
19 end
```

Algorithm 1: pseudo code for the extended sutherland-hodgeman algorithm.

3 Implementation

In this section the implemented class Clipper is presented. All of the class functions are defined and further explained with the intent to clarify how the technical background is utilized within this specific project, with an exception of the signed distance function which was implementet as stated in 2.c.

3.a Clipping constructor

The class must be initialized with two values, the screen width and the focal length of the pinhole camera, in order to calculate the width of the near plane W_{width} as it should be relative to the camera. This is calculated with equation 1. Moreover, two global variables are set upon object creation - the distances to the near and far planes of the frustum i.e d_{near} and d_{far} respectivly. These values are used in the the function that generates the frustum planes.

3.b Generate frustum

The frustum, consisting of six planes with their normals pointing inwards, is calculated every frame. This is done in the generateFrustum-function that takes the normalized momentary camera orientation vectors, the camera position $\mathbf{c}$ and the forward $\mathbf{f}$, up $\mathbf{u}$ and right $\mathbf{r}$ directions as arguments. Moreover, the center points of the near plane $\mathbf{c_N}$ and the far plane $\mathbf{c_F}$ is needed to compute the frustum, and is given by:

$$\mathbf{c_N} = \mathbf{c} + \mathbf{f} \cdot d_{near} \quad (8)$$

$$\mathbf{c_F} = \mathbf{c} + \mathbf{f} \cdot d_{far} \quad (9)$$

In section 2.b Frustum, the definition of planes makes the calculations of the frustrum planes simple. Let the frustum $P = \{P_{near}, P_{far}, P_{right}, P_{left}, P_{top}, P_{bottom}\}$ where the subscript of each $P_i = \{\mathbf{n_i}, \mathbf{p0_i}, d_i\}$ corresponds to one of the six planes that make up the frustum (according to the definition in section 2.b) and the width of the near plane is W_{width}, then each planes normal and point is calculated by:

$$P_{near} = \{\mathbf{f},\ \mathbf{c_N},\ d_{near}\} \quad (10)$$

$$P_{far} = \{-\mathbf{f},\ \mathbf{c_F},\ d_{far}\} \quad (11)$$

$$P_{right} = \{\mathbf{u} \times \hat{\mathbf{a}}_{\mathbf{right}},\ \mathbf{c}, d_{right}\},\ \hat{\mathbf{a}}_{\mathbf{right}} = \frac{\mathbf{c_N} + \mathbf{r} \cdot 0{,}5 \cdot W_{width} - \mathbf{c}}{|\mathbf{c_N} + \mathbf{r} \cdot 0{,}5 \cdot W_{width} - \mathbf{c}|} \quad (12)$$

$$P_{left} = \{-\mathbf{u} \times \hat{\mathbf{a}}_{\mathbf{left}},\ \mathbf{c}, d_{left}\},\ \hat{\mathbf{a}}_{\mathbf{left}} = \frac{\mathbf{c_N} - \mathbf{r} \cdot 0{,}5 \cdot W_{width} - \mathbf{c}}{|\mathbf{c_N} - \mathbf{r} \cdot 0{,}5 \cdot W_{width} - \mathbf{c}|} \quad (13)$$

$$P_{top} = \{\hat{\mathbf{a}}_{\mathbf{top}} \times \mathbf{r},\ \mathbf{c}, d_{top}\},\ \hat{\mathbf{a}}_{\mathbf{top}} = \frac{\mathbf{c_N} + \mathbf{u} \cdot 0{,}5 \cdot W_{width} - \mathbf{c}}{|\mathbf{c_N} + \mathbf{u} \cdot 0{,}5 \cdot W_{width} - \mathbf{c}|} \quad (14)$$

$$P_{bottom} = \{\hat{\mathbf{a}}_{\mathbf{bottom}} \times -\mathbf{r},\ \mathbf{c}, d_{top}\},\ \hat{\mathbf{a}}_{\mathbf{bottom}} = \frac{\mathbf{c_N} - \mathbf{u} \cdot 0{,}5 \cdot W_{width} - \mathbf{c}}{|\mathbf{c_N} - \mathbf{u} \cdot 0{,}5 \cdot W_{width} - \mathbf{c}|} \quad (15)$$

Equation 10 to 15 may seem a bit confusing, but is the result of the geometrical relationship between the frustum and camera orientation. The near plane normal is always equivalent to the normalized forward vector, as is the far planes normal but in the reverse direction.

The computation of the other normals are a bit more involved. $\hat{\mathbf{a}}_\mathbf{i}$ are computed vectors, parallel to each corrisponding plane. The normal is then computed by taking the cross product of these plane-parallel vectors with one of the direction vectors that result in a normal pointing inwards into the frustum.

For each plane, a point **p0** is needed to fully compute the offset constant d. Geometrically, these points are already available as can be seen in the equations 10-15. For the near plane, the point $\mathbf{c_N} \in P_{near}$ and the point $\mathbf{c_F} \in P_{far}$. As the planes span infinitely in its local axis, the camera position is always in the planes parallel to the $\hat{\mathbf{a}}_\mathbf{i}$ vectors. The computation of the constant d_i is done as stated in section 2.b, after the normal and point has been computed for each plane.

3.c Compute intersection

The function that computes the intersection does so in two parts. This is to handle the special case described in the section 2.d. First, the denominator of equation 6 is computed. If the denominator is equal to zero, the line is parallel to the plane and the end-point is returned. This also ensures that the function never divides with zero. Otherwise, the computation of the intersection continues and the rest of equation 6 is computed.

3.d Clip to frustum

The implementation of the Sutherland-Hodgeman algorithm was done by following the pseudo code in section 2.e. There is little difference from the pseudo code in the actual source code, which can be seen when comparing them.

4 Result

In figure 1 and 2 are two different still image sequences taken during the execution of the resulting program. The viewing frustum culling adds little overhead to the execution time. As less pixels has to be interpolated in the rasterizer pipeline, the overhead added from the implemented Clipping class is proportional to the amount of pixels inside the frustum.

Rotating the camera around the y-axis and "walking" inside the room was impossible before implementing the viewing frustum culling class. Vertices outside the view-able volume was still being computed and froze the running program. The culling class has now solved both problems.

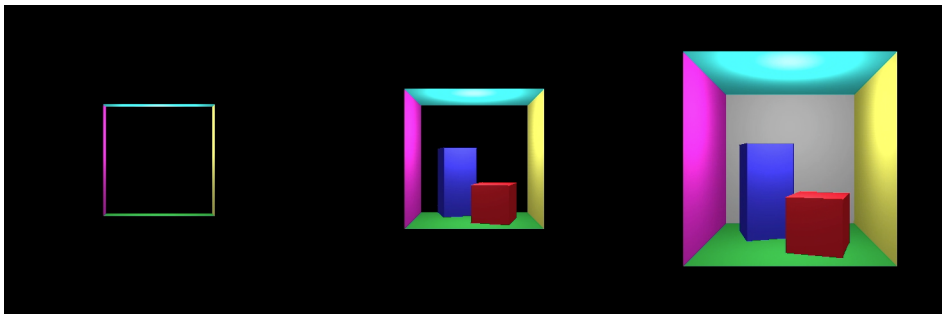

Fig. 1: The culling when approaching from the front of the room.

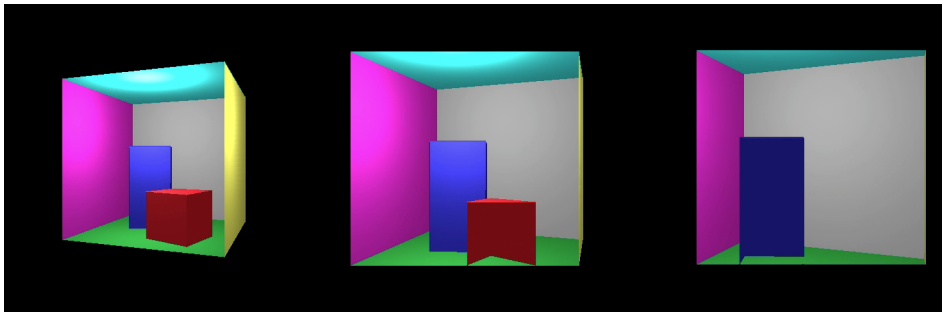

Fig. 2: The culling when passing through the room from the side.