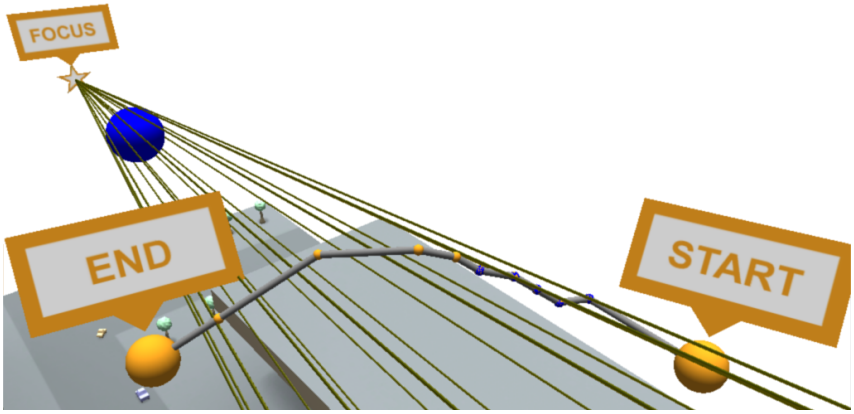
Dynamic Objects in Global Camera Control



Serkan Bozyigit

Semester Thesis
January 2009

Supervisors:
Thomas Oskam
Prof. Dr. Markus Gross

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
Computer Graphics Laboratory ETH Zurich

Abstract

This semester thesis presents approaches to extend Oskam's global camera control in [Osk08] by dynamic objects. By inspecting the nodes of a precomputed roadmap datastructure and its connections for intersections with bounding spheres, we are able to make the global camera control algorithm become aware of moving obstacles in a 3D environment. Furthermore, to guarantee the continuity of visibility-ensuring properties of the preliminary work, we introduce a visibility cone structure to satisfy these requirements. Finally, adapting the iterative path post-processing algorithm is necessary to make the extension fully comply with the user's visual needs.

Contents

List of Figures

1

Introduction

Automatic camera control is an interesting subject for several applications as computer games, interactive applications and other virtual environments. The approches to this problem may fairly differ from one another. They reach from no automatic control, where the user controls the camera himself, over local control, which is based on algorithms that solely consider obstacles in the immediate vincinity of the camera, to global control, where a path finding algorithm focuses on large-scale camera transition within the virtual environment. This thesis specifically studies the work of Oskam in [Osk08] who uses an elegant trade-off between global pre-computation and local adjustment during runtime. For this thesis, data structures and algorithms of Oskam's work are adapted and extended in order to meet this thesis' dynamic requirements.

If we consider the camera control in the given work as it is, it is not able to handle dynamic objects correctly, i.e. recognize and build a path around objects that move during runtime, like driving cars, opening or closing doors, or jumping balls. Its path finding algorithm works on a roadmap, a pre-computed data structure capturing the static objects in a scene, that is inherently not able to detect dynamic objects. Hence, it would not make sense to consider the objects' positions for any kind of calculations as these are most probably going to change at runtime anyway.

In the following chapters, different approaches are presented to extend the camera control framework to be able to consider dynamic objects during runtime. To achieve this, we have to add or modify three parts of the given framework:

- **Circle-Circle-Connection (CCC):** Describes the area between two portal circles in 3D which ought not to be entered by any dynamic object.
- **Visibility Cone:** A cone that is basically the "visibility shadow" of a dynamic object cast by a given focus point.

- **Path Smoothing:** This item already exists in the given framework, but has to be extended to work correctly with the above enhancements.

2

Related Work

This semester thesis is based on Thomas Oskam's work in [Osk08] and extends it in such a way that visibility transition planning for cameras is not only feasible for static, but also for dynamic environments.

2.1. Basis: Visiblity Transition Planning For Real-Time Camera Control

Oskam's framework uses an algorithm that is able to calculate large, occlusion-free visibility transitions for cameras in real time in complex static environments. The algorithm is based on a pre-computed global roadmap data structure that captures empty space of the environment by sampling spheres in order to build a coarse map of all possible collision-free paths. The speciality lies in the incorporation of a visibility measure between all given spheres which is used for a freely choosable focus point. During runtime, the algorithm seeks to satisfy the visibility condition implied by the focus point's position, and chooses an optimal solution in terms of visibility on the coarse roadmap by employing an A-star path finding algorithm.
In an additional step, an iterative smoothing algorithm is applied on the coarse path to tune it on a finer scale. This step is based on so called occlusion maps computed for sampling spheres with partial visiblity values. Such values imply that the according sphere is only partially visible from the focus point. The same path finding algorithm as above is used on these maps to smooth path segments according to the prevailing environmental geometry. Finally, a dynamic controller uses this path to guide the camera in a complex 3D environment.

3

Circle to Circle Connections

Given the preliminary work, we need to analyse the way the path finding algorithm works, and adapt it in an appropriate manner to make it work for a dynamic environment. Dynamics is seen here in the sense of movable obstacles. Obviously all static objects - such as houses, trees or bridges - are not part of any solution path nor are they colliding with any. But how is that actually achieved?
To guarantee a smooth and collision-free path, Oskam introduces a global roadmap data structure. This roadmap grasps empty space, the negative of the (static) geometry of the virtual 3D environment so to speak. Therefore, the path search itself only operates on a reduced environment and inherently avoids any environmental structures. The roadmap is constructed by sampling many overlapping spheres distributed in the empty space of differing sizes in order to capture possible finer structures in the geometry. All spheres avoid to intersect with any environmental geometry, but they may intersect with neighboring spheres. In fact, these resulting intersection circles are of major interest, because their centers make up the connection points of a solution path. Figure 3.1 shows such a 3D environment, the empty space approximating spheres, their intersection circles and their nodes.

Since the virtual environment, on which we are operating, can be quite large and therefore result in expensive computations, the roadmap has to be pre-computed on a given (static) geometry. The insertion of any dynamic component that would change its position during runtime would not make any sense at this point. An instance of a dynamic obstacle can be a driving car, a bouncing ball, or an opening door. These objects are actually using the same space as the player or respectively the camera, which is the empty space covered by the roadmap. This leads us to our approach.

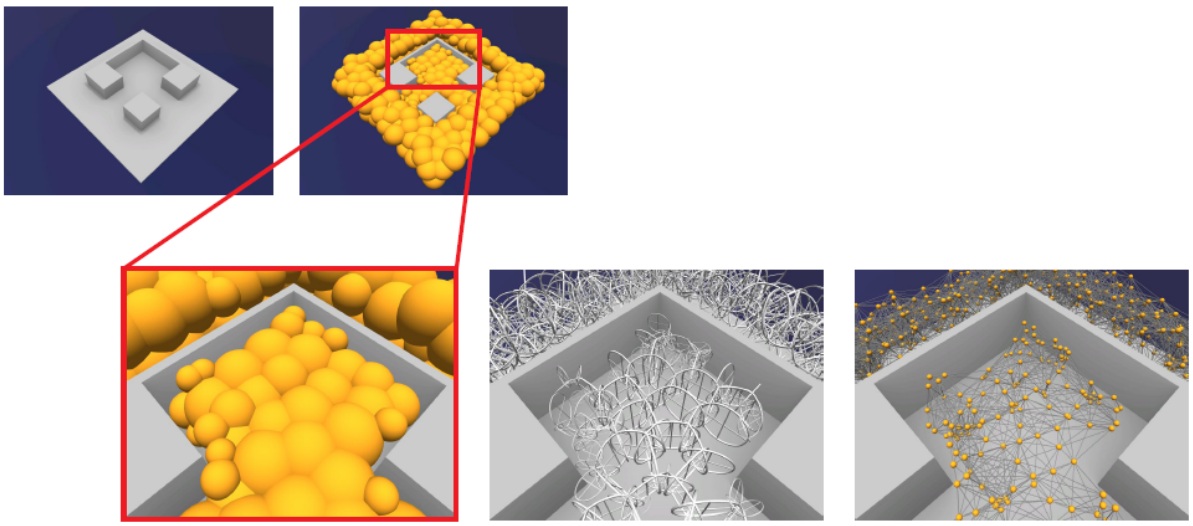


Figure 3.1.: *This image shows the 3D environment. In the first picture of the second row, the empty space gets sampled with diffently sized spheres, whereas the following one shows the spheres' intersection circles and finally the nodes and connections to their neighbors which represent the roadmap*[1].

3.1. Making the Algorithm Become Aware of Dynamic Obstacles

The path finding algorithm is based on the A-Star Best-First search algorithm presented by Dechter in [DP85]. Given a focus, a start, and an end point, the algorithm returns the optimal path between these points. Optimal in the sense of visibility towards the focus point. This visibility refers to the mutual visibility probability between spheres.
The visibility of a certain point in space can only be evaluated from another point in space. Either they see each other (visibility is symmetrical) or they do not. But our data structure is made out of spheres and we could only make a guess - at pre-computation time - where the points may lie within the spheres. Hence, we adapt the visibility value from being a boolean to being a number between 0 and 1 (borders included) that determines the probability of how well a sphere sees another sphere.
Each of these spheres have a list of visibility probabilities for every other sphere in space, which are also taken care of in the pre-computation step. However, for this first part of the thesis we do not need to worry about the visiblity, because above all, we first need the algorithm to recognize dynamic obstacles and need it to avoid those. Visibility issues will be tackled in chapter 4.

A path p can be described as a sequence of segments

$$p = s_0, s_1, \ldots, s_{n-1}, s_n$$

and starts with the start node s_0 and ends with the end node s_n which can be anywhere in the empty space. Both are obviously not part of the roadmap because they are newly defined points, but the following respectively preceeding node in the path is part of the roadmap and it will be

[1] Picture taken from Oskam's work.

the node with the smallest euclidian distance to the start respectively end node. Such a regular node is the center of an intersection circle of two *parent* spheres. A node can only connect to nodes that reside in either one of the parenting spheres. The solution path, therefore, constitutes of line segments which always have only *one* parent sphere, except the start and end segment. Both, the start and the end segment may be inside a sphere or they may even reach a space which is not enclosed by a sphere, because not the entire space is necessarily covered by spheres.

In order to control the path's choice of path segments, we have to restrict it to those which are not intersecting with any dynamic obstacle. Generally, a dynamic obstacle can be approximated with one or more bounding spheres, depending on the shape and size of the geometry that shall be represented.
An intersection can be detected by checking if the node-to-node connection intersects a bounding sphere is *not* sufficient since the smoothing algorithm of a later step will be allowed to move a node within its circle. We therefore need to check if the volume between two circles and the circles themselves intersect a bounding sphere. Depending on the mutual positions and orientations of the portal-circles, the shape of the volume between two circles can take versatile forms and make this task nontrivial. Some valid and invalid examples are shown in Figure 3.2.

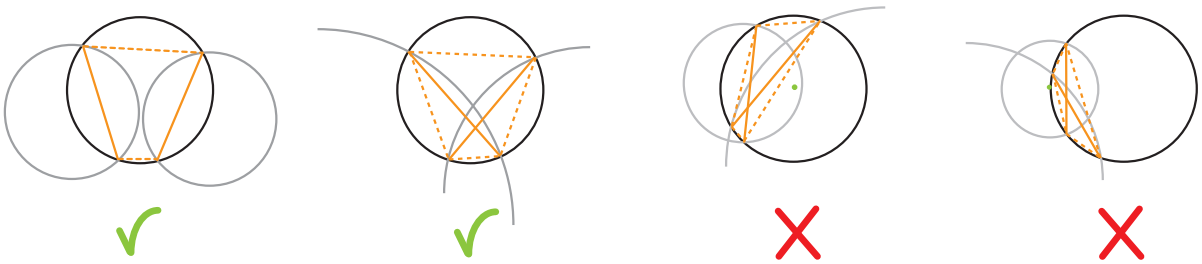


Figure 3.2.: *This visualisation shows 2D examples of three intersecting spheres with their inscribed valid and invalid intersection circles. During the creation of the roadmap a rule is applied stating that no sphere is allowed to contain another sphere's center which makes the two last examples invalid. The area between the dashed lines defines the CCC volume that is discussed in section 3.4*

If we look at the different kinds of constellations of two circles within a sphere we notice that some restrictions apply, and certain constellations are not possible. Taking advantage of this insight we can rule out some exotic cases. Nonetheless, there are numerous cases to go through. To make the nomenclature a bit easier and shorter we call these shapes in all their varieties *circle to circle connections* (CCC).

To keep the intersection detection algorithm general and to be able to use it on all parts of the path, i.e. also on the starting and ending segment, three cases have to be distinguished where two of them are reduced types of CCCs:

- point to point
- point to circle
- circle to circle

All types of intersection tests precede a bounding sphere test in order to rule out cases where the moving object is so far away from the *connection* that they cannot possibly intersect.

3.2. Point to Point Connection

Point to point connections may appear when we have a very short euclidian distance between start and end node. In most of the cases this computation will result in a straight line. Nevertheless, we need to verify if this line segment could pierce a dynamic obstacle and in that case build a path around it.

The intersection test between a sphere and a line segment is rather straight forward. A line segment consists of two distinct three dimensional points in space and the route between those points. Figure 3.3 shows all intersection cases between a sphere and a line segment. The trivial case is not shown.

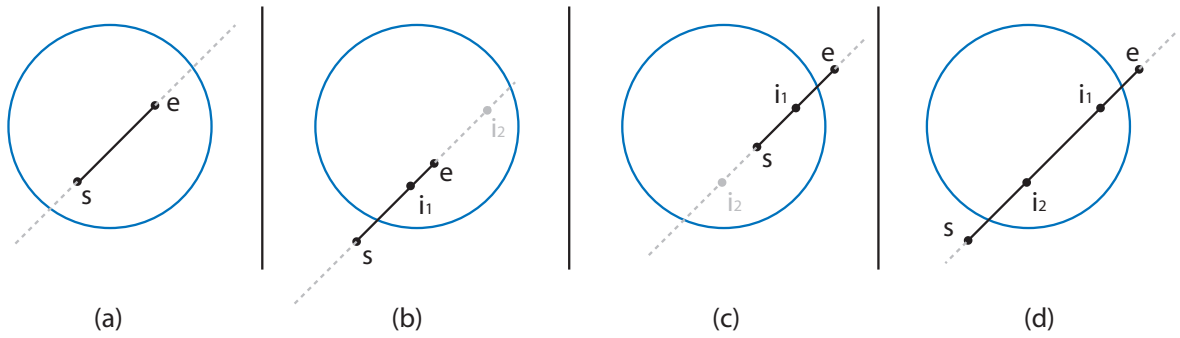


Figure 3.3.: *These images show four different cases of intersections between a line segment and a sphere. In (a) the segment is fully contained, in (b) the end point is contained, in (c) the start point is contained and in (d) a subpart of the line segment is intersected.* i_1 *and* i_2 *mark the intersection points with the sphere.*

3.2.1. Intersection Test

1. Check whether the sphere contains the line segment's start or end point.
2. If the points are indeed two distinct points, check whether the line intersects the sphere
3. Check whether the intersection points are between the start and end point. Note that both points have to be either between the start and end point or outside. Otherwise we would have detected the intersection in the first test.

3.3. Point to Circle Connection

Point to circle connections appear in almost every path and are situated at the start and end node. The path's start and destination point do not have an according circle around them like any ordinary nodes of the road map. But nodes just after or before the start node, respectively the end node, have a circle. Therefore the connection results in a shape is a more general cone since its mathematical axis (circle center to vertex) does not necessarily need to be perpendicular

to the circle's plane. Figure 3.4 shows a draft of such a cone.
The intersection test between a general cone and a sphere is more demanding than the one from the previous section.

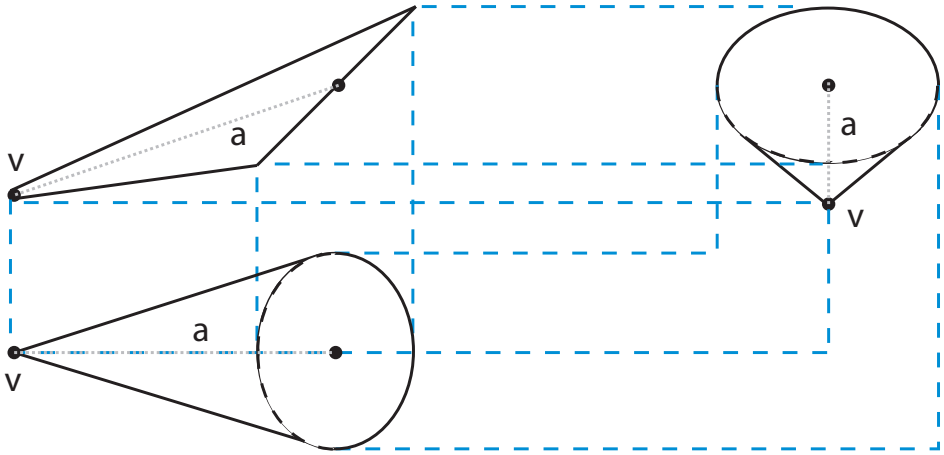


Figure 3.4.: *This figure shows a draft of the general cone. Note that looking in the direction of the axis a (circle center - vertex v) shows that the circle appears to be an ellipse which also affects the form of the cone's outer coat.*

3.3.1. Intersection Test

1. Build a bounding sphere around the whole structure and check whether the sphere contains it.
2. Check if the cone's vertex v is inside the sphere.
3. Calculate the closest point to the cone's circle and clamp it to the rim of the circle. Check whether the line segment, made out of the apex and the new point, intersects the sphere.
    - If the line intersects the sphere but is outside of the segment, then check if it intersects the circle plane inside the circle.
4. Check whether the sphere intersects the circle plane and is inside the radius' value.
5. Calculate two vectors which are used to determine whether the point is in the correct cone. Some calculations with the cone may actually be ambiguous because the mathematical definition of a cone with rays may become a definition with (infinite) lines at some point due to squaring of the equation.
6. Use the vectors to see if the sphere's midpont is inside the cone.
7. Check if the point is on the right side of the circle, i.e. not outside of the cut cone.

3.4. Circle to Circle Connection

This *connection* type is the most common one and has to be tested when two ordinary nodes are connected. The circles may basically have any size and orientation as long as they fit in their parent sphere. However, there is a rule that is applied while creating the spheres stating that no new sphere is allowed to contain another already existing sphere's center. Practically, this gives the advantage that we can make assuptions about the positions of the spheres relatively to each other which gives us indirectly insight into the circle's positions and orientations.

Figure 3.2 shows the different kinds of positions two circle can or cannot have within a sphere. Note that we can always rotate a CCC around its axis in order to see it as they are shown in the sketches. These sketches are not complete in numbers, the circle sizes are assumed to be similar, but of course they may vary significantly which does not influence the way of testing the intersection. As shown in Figure 3.2 we can divide the cases of mutual positioning roughly into two groups, the ones where the circles overlap and the ones where they are disjunct.

In order to perform an intersection test we need to define what belongs into the volume of a shape and what does not. In the case of a sphere we simply take the mathematical definition of its volume. The definition of the volume between two non-overlapping circles in space is nontrivial but also rather straight forward compared to the volume of two overlapping circles. The central axis of such a constellation is always well-defined because of the sphere-center-containment-rule mentioned before. For example a constellation as in the third picture of Figure 3.2 cannot occur since the sphere that is creating the circle by overlapping the *main* sphere would contain its center. There are basically three possible ways of defining such a volume:

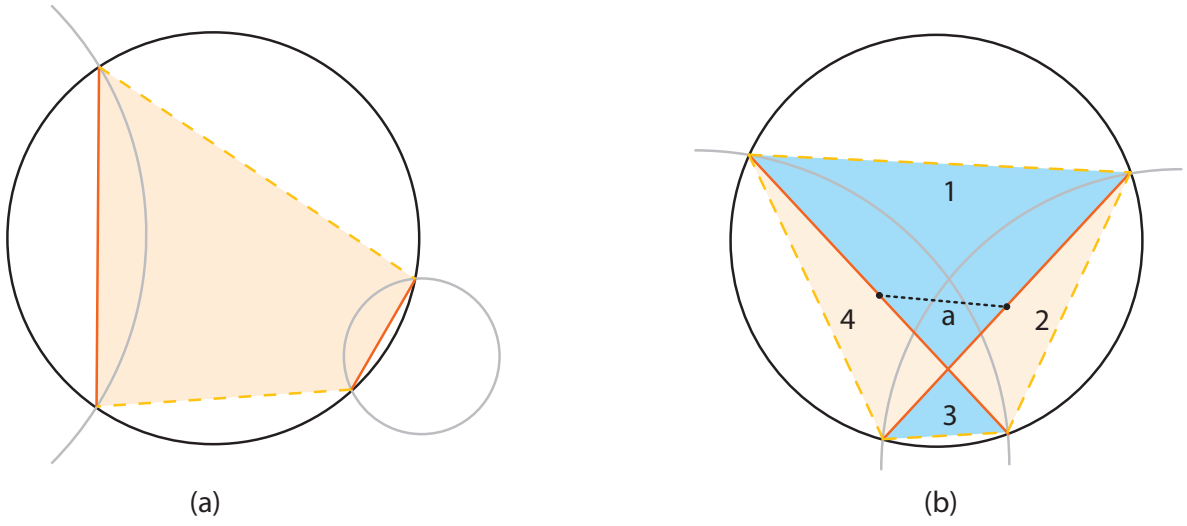


Figure 3.5.: *The left drawing shows three spheres with non-overlapping intersection circles. The right drawing has the same setup but with overlapping circles instead. The central axis between the centers of both circles is denoted with a. Four regions (1-4) are defined due to the crossing of the circles, with 1 being the region where a lies in.*

1. The volume that is spanned if we connected every point on one circle with every point on

the other circle, and vice versa.

2. The above described volume can be splitted into four regions and resembles the shape of an **X**, as seen in Figure 3.5. One of those regions contains the CCC's central axis which we call region 1. Note that the central axis can come close to **X**'s crossing point but never goes through it. Adjacent to region 1 we have region 3 and accordingly the remaining regions are labeled 2 and 4.
   If we look at the path as a whole it is surrounded by a felxible pipe with a pointy start and end. Intuitively it would make the most sense to use the volume of region 1 and 3 for a definition since they sort of describe the volume between two overlapping, and on the other side, non-overlapping pipe segments. Figure 3.5 shows an example with the labeled regions.

3. Just for the sake of completeness we give the last possible way of defining such a volume although it does not make much sense for our purpose. Instead of using the region 1 and 3 we might use 2 and 4 which could be useful in an other setup. But here it would give us the volume inside the pipe segments that precede or are to follow, which would not be of practical use for us.

Option one of the above enumeration would be the safest bet to take, but having a look at path instances as for example in Figure 3.6 we can see that often certain spaces are covered by two or more CCCs. As this algorithm is supposed to run in real-time we try to minimize the overall number of calculations for intersection tests. Hence a trade-off is made between correctness and responsiveness.
Following this thought we consider to drop the calculations for region 2 and 4 and show that option two is practically sufficient to see whether any movable obstacle interstects with the "flexible pipe" and the path inside it. In doing so we are only omitting small fringe regions.
Note that this part is not about the intersection itself but the coverage of the volume two circles might span.

- In the case of a path without overlapping circles all segments are cylinder-like or apex-less cone shapes as shown in picture (a) of Figure 3.5, and do not have any special overlapping region. Without having overlapping regions, the question whether to check a certain special part of the segment does not even pop up - the whole segment always has to be checked.

- Consider a cylindrical segment s_{i-1} and a segment with overlapping circles s_i. This setup involves in total three circles and the middle one is shared by both segments s_{i-1} and s_i. Because one of them is shared we can be sure that the halfspace of the segment s_{i-1} has already been checked. In fact, in this case region 3 is checked twice. Obviously, region 2 lies in segment s_{i-1}'s space as well and does not need to be tested. Region 1 has to be tested because the connection between the circle centers lies within it. The remaining region will not be checked in segment s_i's turn, because segment s_{i+1} will cover it, if necessary. Segment s_{i+1}'s central connection either points to the region 1 or region 4. If it is pointing towards region 1, this case was already covered, if it points towards region 4, it will be covered by segment s_{i+1} processing because it is the region 1 of that segment. The pictures (a) and (b) in Figure 3.6 show the two cases where segment s_{i+1}'s central connection either points to s_i's region 1 or to region 4. There is a chance that the final

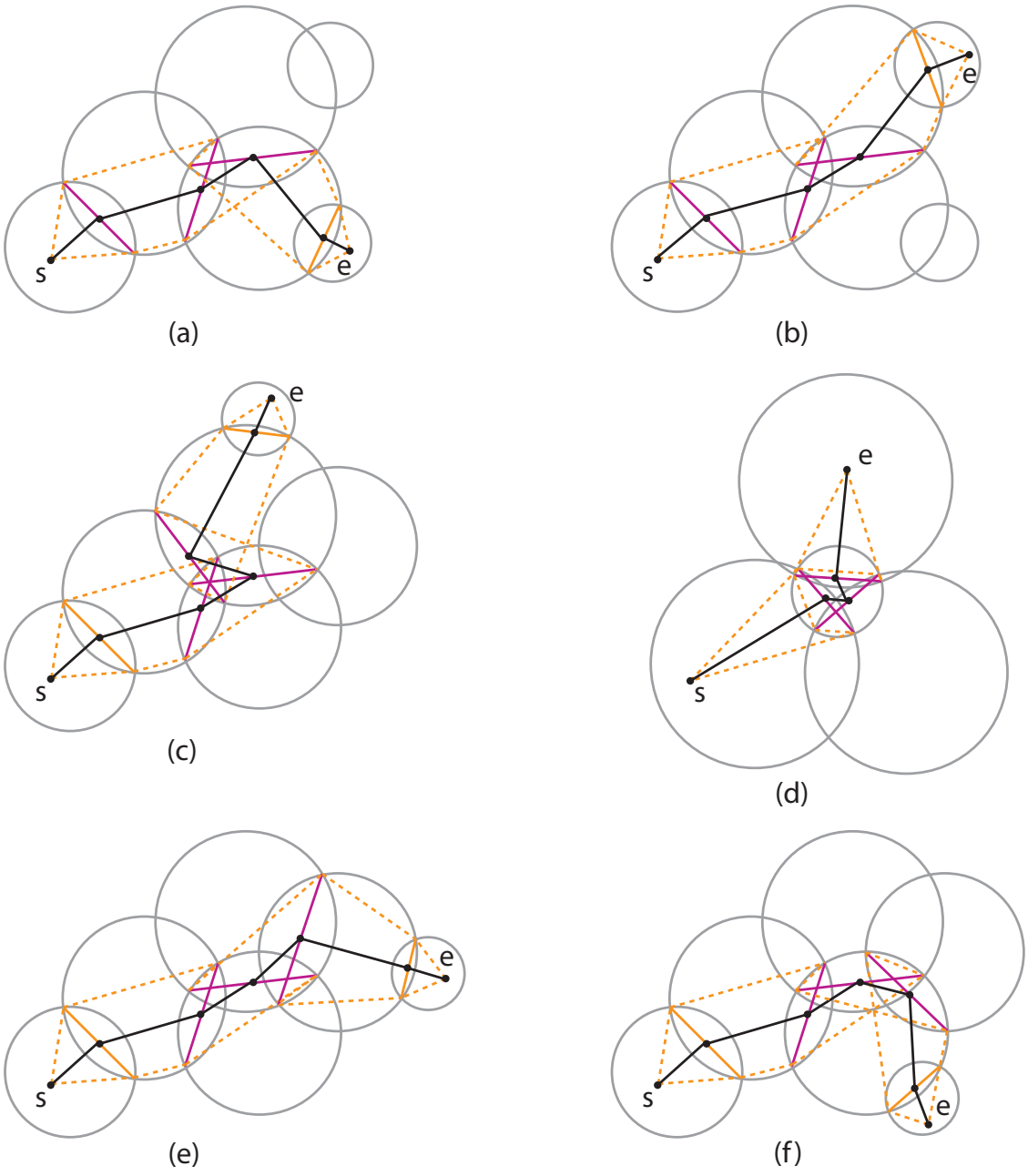


Figure 3.6.: *These pictures show the different constellations of circles of two consecutive segments s_{i-1} and s_i. Path pipes are dashed, the three circles of the two consecutive segments are in rasberry-red and the initial path connecting the circle centers in black.*

smoothed path, as opposed to the black initial path in (a), moves into region 4 and would therefore be unscreened. However, through smoothing the final path's total length tends to become *smaller* than the initial path's length and therefore leaves region 4 untouched.

- There is also the above item's opposite case, where segment s_{i-1} is the one with the overlapping circles and segment s_i is the cylinder-like volume. This case is the mirrored version of the above reason and can be seen in the same pictures (a) and (b) in Figure 3.6 by exchanging the start and end nodes' labels.
- The last case consists of two segments with overlapping circles, meaning we still have three circles involved. Here we have four possible constellations of spheres and their circles. The pictures (c) through (f) in Figure 3.6 show the four variants. All constellations in the figure are arranged in the simplest possible way in order to keep the picture clear. Since we are dealing with three circles in 3D it is rarely possible to rotate the whole structure and then to project it orthogonally in such a way, that all circles represent a line instead of an ellipse. Actually most of the variants are quite challenging for the imagination and it would be even more challenging to bring random constellations reasonably on paper as a sketch.

    1. Figure 3.6's picture (c) shows the three circles from above forming a star shape. They are created by three spheres which are intersecting each other mutually. We only look at segment s_{i-1} and s_i, the one before (s_{i-2}) and after (s_{i+1}) follows the same reasoning as in the second item above and do not need to be examined again. Regions 1 and 3 are covered as usual, but we can also see from the picture that the region 2 and 4 are either covered by s_{i-2} and s_{i+1} or they lie within the region 1 of the neighboring circle.
    2. Picture (d)'s triangle constellation might be rather rare. It can only appear if a sphere is surrounded by three or more much bigger spheres that happen to intersect each other too. Besides these conditions the actual path length is increasing by adding little indirections, which most often makes the path finding algorithm favor longer line segments. However, in this case both segments cover biggest parts of each other's spaces. There is one noticable region in this picture and depending on s_{i+1}'s central axis direction this region will either be covered by s_{i+1} region 1 or it will look as we have it in the picture. But due to the path's form it is unlikely that the path gets extended to this region during smoothing.
    3. In the "Z" constellation in picture (e) the regions 2 and 4 around the middle circle are largely mutually covered by their according 1st regions. Although there might be little volumes that are not covered by the other CCCs region 1, they do not really carry weight because on one hand they have a very small volume as already mentioned and on the other hand they are on the outermost part of the whole CCC which gives them a low probability that the path will end up in those volumes. Once again the regions 2 and 4 which are shared with the next, respectively previous, segments are taken care of by those.
    4. The region 1 of both CCCs in the "U" constellation in picture (f) are a bit unfortunate in the sense that they do not cover mutually the other regions, as it was in the previous cases. The only remaining area is the one behind the middle circle, but due

to the CCC's shape it is, again, very improbable that the path would move outwards to the unchecked area during smoothing.

3.4.1. Intersection Test

The previous section has shown that the CCC can have quite different shapes depending on the size, mutual position and mutual orientation of both circles. Defining its volume is a demanding task, but we do not need it to calculate the capacity, but to actually know what is inside or outside. Now that we have a volume definiton we can use it to test it for intersections, in our case with a sphere. The sphere is the easiest geometric form to test against intersection. Once a point is closer to the sphere's center than the radius' value it will be deemed inside it. The following nested list shows the steps the intersection algorithm is going through from top to bottom to determine whether the sphere intersects the CCC's volume. Some of the tests cannot be answered in the "first attempt", they have to be examined further.
The CCC consists of two circles C_1 and C_2, the moveable obstacle's sphere is called S. For further geometric details and explanations refer to the appendix.

1. Create a bounding sphere around the CCC and check it for intersection with the movable obstacle's sphere S.
2. Calculate the closest point to the CCC's *first* circle C_1 and clamp it to the rim of the circle. Check if the sphere S contains this point.
3. Calculate the closest point to the CCC's *second* circle C_2 and clamp it to the rim of the circle. Check if the sphere S contains this point.
4. Connect the previous two points C_1 and C_2 and check whether this line segment intersects the sphere S.
    - If the line intersects the sphere S but is outside of the segment, then check if it intersects the circle (C_1 or C_2) plane which is closest to the sphere S.
5. Check if the sphere S intersects the CCC's *first* circle C_1's plane and if the center of this new circle C_n is closer to C_1's center than the radius' value.
6. Check if the sphere S intersects the CCC's *second* cricle C_2's plane and if the center of this new circle C_m is closer to C_2's center than the radius' value.
7. At this point, it means, that the sphere S does not interfere with any "boundary entity" of the CCC. Therefore, there is no clue yet whether it is inside or outside the CCC. Find out whether the sphere's center is inside the volume using the midpoint test.
8. In case the upper test was positive we have to build in a condition for overlapping circles which determines in which region the sphere is situated.

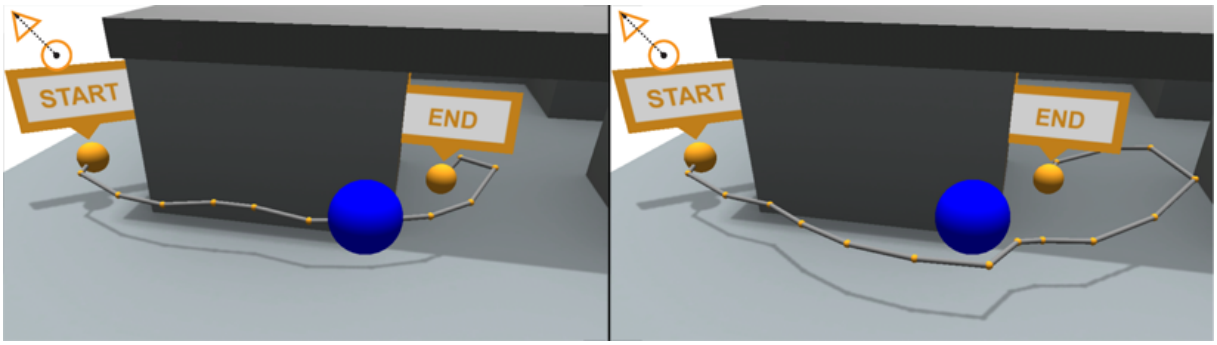


Figure 3.7.: *The left screenshot shows how the regular path finding algorithm ignores the moving obstacle depicted as blue sphere. In the right picture the path built on the basis of the adapted algorithm avoids to run through the object.*

3.5. Alternative Implementations

In the current implementation region 2 and 4, as depicted in Figure 3.5, are omitted with the reason that they are often covered by preceeding or following path segment tests. Doing so is supposed to speed up the process of detecting and preventing possible intrusions but also brings imprecisions with itself. Despite the sparseness of these flaws due to geometric and probabilistic reasons, we might want to improve this rate.
Adding region 2 and 4 into the test cases would cover the missing parts, but would also increase the number of calculations to be done. As shown in the section before, the CCC intersection test is quite elaborate and might be taking too long to answer within a beneficial time in a complex setup with multiple dynamic objects.
In order to save time during this phase of the algorithm we could simplify the checks for object intrusion. Currently, by looking at CCCs we are examining parts of a sphere that contains the particular CCC. Instead of concentrating on the odd shapes of the CCC, an intrusion test for the whole sphere would suffice. Checking whether two spheres are intersecting, namely the moving obstacle and the sphere that is sampling the empty space of the environment, is mathematically very easy and ends up being a pure distance check of two points, their centers. The downside of this idea would be that the sampling spheres may be relatively big, which in turn would disqualify a vast area of space once an object dynamically enters this region. Shortest paths might no longer be as short as they used to be. To counteract against these large redirections one could simply lower the size limit of the sampling spheres which would increase the number of total sampling spheres. As opposed to the current implementation, this solution would disqualify space that lies outside the possible moving area of path segments.

Yet another possible approach could be to simply test the node-to-node connections for intersection with movable objects instead of the whole CCC. Again, line-sphere intersections are mathematically rather simple and would save us computation time. However, the disadvantage of this approach is that the computational load would just be transfered to a later stage, i.e. the path post-processing, which will be discussed in section 5.2.

4

Visibility Cone

Until now, the visibility-considering property of the framework has not yet been examined. Leaving the framework as it is, with all the changes made to this end (i.e. the detection of dynamic obstacles using CCCs), the path would avoid to go through the obstacles but would not care about enhanced visibility towards a focus point. A path passing by the rear side of a dynamic obstacle relatively to the focus point would, in fact, have avoided the obstacle, but the camera on the path would have a better visibility if it did not pass through the object's "shadow" when looking from the focus point.

As described in the previous section, each sphere approximating the empty space has a list of visibility probabilities. These probabilities, however, only consider the static environment. Nevertheless, they will be of use later on. It would come in handy if we could simply take the already existing visibility probability, since these probabilities are still true for the static obstacles, and combine them to obtain a temporary visibility value for movable objects. Due to the dynamic obstacle's movable nature we only use this value for the current calculation. Note that the temporary value can only be equal to or worse than the pre-computed value, since adding an object into the scene cannot improve the visibility.

A focus point within the scene can be compared to a point light. Everywhere it casts a shadow the visibility drops to zero. Movable obstacles are approximated by bounding spheres, therefore their shadow spaces form a cone with a truncated apex and at the cut-off having the moveable obstacle.
For each segment of the path that is being considered by the path finding algorithm the visibility probability has to be calculated. This value is directly taken from the parent sphere it resides in and adapted. To adapt this probability for dynamic circumstances we use the following formula:

$$p_{temp} = p_{static} * penalty.$$

Where p_{static} denotes the visibility probability of this path segment and $penalty$ the penalty we

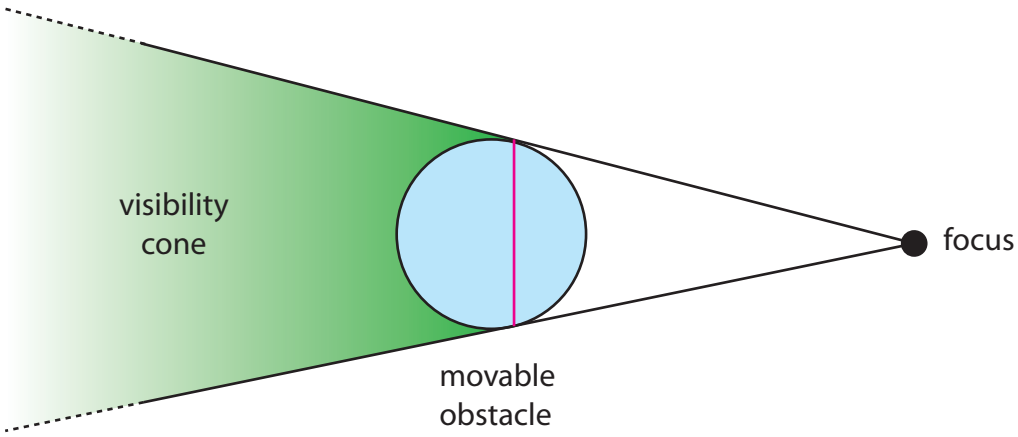


Figure 4.1.: *This drawing illustrates a visibility cone created by a focus point and a moving obstacle. The purple line indicates the touching circle, where the visibility shadow starts from and expands against the focus point.*

still need to define. There are several ways to define the penalty:

- The penalty could be calculated by taking the intersection volume between the cone and the sphere divided by the sphere volume. This would tell us how much of the sphere is visible from the focus point only considering the dynamic obstacles. To exactly determine this volume we would need to perform a rather costly calculation of a triple-integral, which is probably preceded by a long mathematical derivation.
- The first item's costly calculation is a thorn in our flesh. Substituting the costly part by an approximation could work, but since the cone frustum is not a regular cone it would imply, once again, many cases and differentiations to work through. Despite the number of different intersection cases, it was not the main reason to abandon this approach. Besides simple cases, we had with at least one case a very hard time to come up with a decent approximation. For the sake of comprehention, performance and simplicity we decided to move on to another solution.
- Calculating the intersection volume brings many issues about itself as shown before. In the following section a different approach is presented where nothing is calculated involving the sphere volume but the percentage of a path segment that is inside the cone frustum. In doing so we circumvent calculating approximations and are in most cases closer to the real visibility probability than using the approximations.

4.1. Cone Frustum against Line Segments

To be able to intersect a cone frustum with a line segment the cone frustum has to be defined first. The cone is created by the focus point and the bounding sphere around a movable obstacle, as shown in Figure 4.1. The connection from the focus point to the bounding sphere's center defines the cone's direction and axis. A path that is passing between the focus point and the bounding sphere is able to perfectly see the focus point, therefore the part from the apex to the

bounding sphere shall not be a valid area for the intersection calculation. As shown in the figure, the cone frustum starts from the sphere's "touching circle" downwards. The touching circle is naturally defined by the rays that are emitted by the focus point and lie tangential on the sphere. Due to the previous section's constraints the path will not go through the moving obstacle, which helps to rule out some cases where a line could start in the apex area and pass through the object and end in the valid frustum. Figure 4.2 visualizes the different kinds of valid and

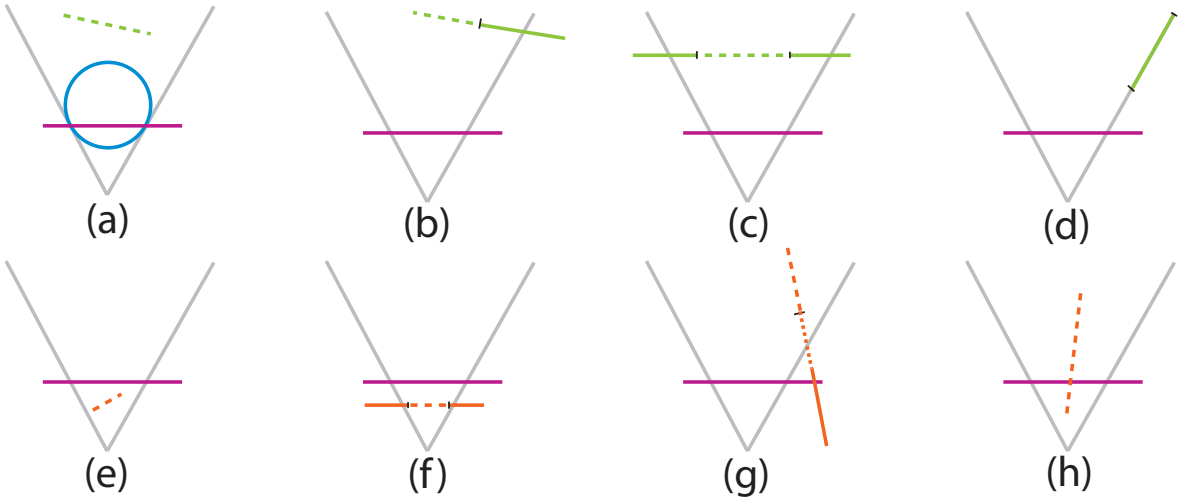


Figure 4.2.: *This visualization shows in the first row valid intersections between a line segment and a cone frustum. Picture (a) shows the sphere that is inscribed into the cone, the purple line denotes the touching circle's plane. The second row show some of the invalid cases, where as (e), (f) and (g) mean "does not count as intersection" and (h) is not possible. Note that picture (h) would pierce the movable object's bounding sphere.*

invalid intersections. The following pseudo code implements the test:

```
if (line segment's start and end point are in cone)
{
  if (on the valid side of the touching circle plane)
    return 100% length;
}
else if (start point inside && end point not inside the cone)
{
  if (on the valid side of the touching circle plane)
  {
    determine the line's piercing point and calculate
    the percentage of the line that is inside the cone;
  }
}
else if (start point not inside && end point inside the cone)
{
  if (on the valid side of the touching circle plane)
```

```
  {
    determine the line's piercing point and calculate
    the percentage of the line that is inside the cone;
  }
}
else // start and end point are not within the cone
{
  calculate piercing points if any;

  if (line intersects the cone between start and end point)
  {
    if (on the valid side of the touching circle plane)
    {
      determine the line's percentage that is covered
      by the cone using the before calculated points;
    }
  }
}
```

Mathematically, a cone can be described[1] by a vertex $\mathbf{V}$, an axis direction vector $\mathbf{A}$ and an opening angle θ between axis and outer edge. The cone is assumed to be acute, which means $\theta \in (0, \frac{\pi}{2})$. A point $\mathbf{X}$ is said to be inside or on the cone if the angle between $(\mathbf{X} - \mathbf{V})$ and $\mathbf{A}$ is smaller or equal to θ. To reformulate this condition the scalar product $a \cdot b = |a||b| \cos \alpha$ can be used:

$$\mathbf{A} \cdot \left(\frac{\mathbf{X} - \mathbf{V}}{|\mathbf{X} - \mathbf{V}|} \right) \geq \cos \theta,$$

where $\mathbf{A}$ is of unit length. The inequality makes the equation include the cone's interior.

A line's mathematical definition can be written as $\mathbf{X}(t) = \mathbf{P} + t\mathbf{D}$ where $\mathbf{P}$ is a point on the line, $\mathbf{D}$ the line's direction vector and $t \in \mathbb{R}$. In order to obtain the piercing points we have to insert $\mathbf{X}(t)$ into the cone equation and solve for t. This procedure might give us for t:

- no solution, which means that the line does not intersect the cone
- one solution, meaning the line touches the cone
- two solutions, which means that the line pierces the cone twice, resulting in two solution points.

For mathematical and implementation details see Appendix A.4.

By being able to detect the path intrusion into the movable obstacle's shadow cone we penalize those paths that pass through the shadow and make them less interesting for the path finding algorithm. This affects the algorithms choice whereon it also avoids the shadow as much as possible. For the $penalty$ term we can forumlate the following equation:

$$penalty = 1 - \frac{\textit{length of line segment inside the cone}}{\textit{total line segment length}}$$

[1] Based on documents from http://www.geometrictools.com

Obviously, if the start or end point lies within the shadow, there is not much the algorithm can do about it but to find the optimal way out of the visibility cone.

Figure 4.3 compares two screenshots where the first one shows the path that uses the regular path finding algorithm. It obviously does not notice that a blue object is obstructing its way and that it passes through the visibility cone that is visualized by rays emitted by the focus. The second picture succeeds in doing so.

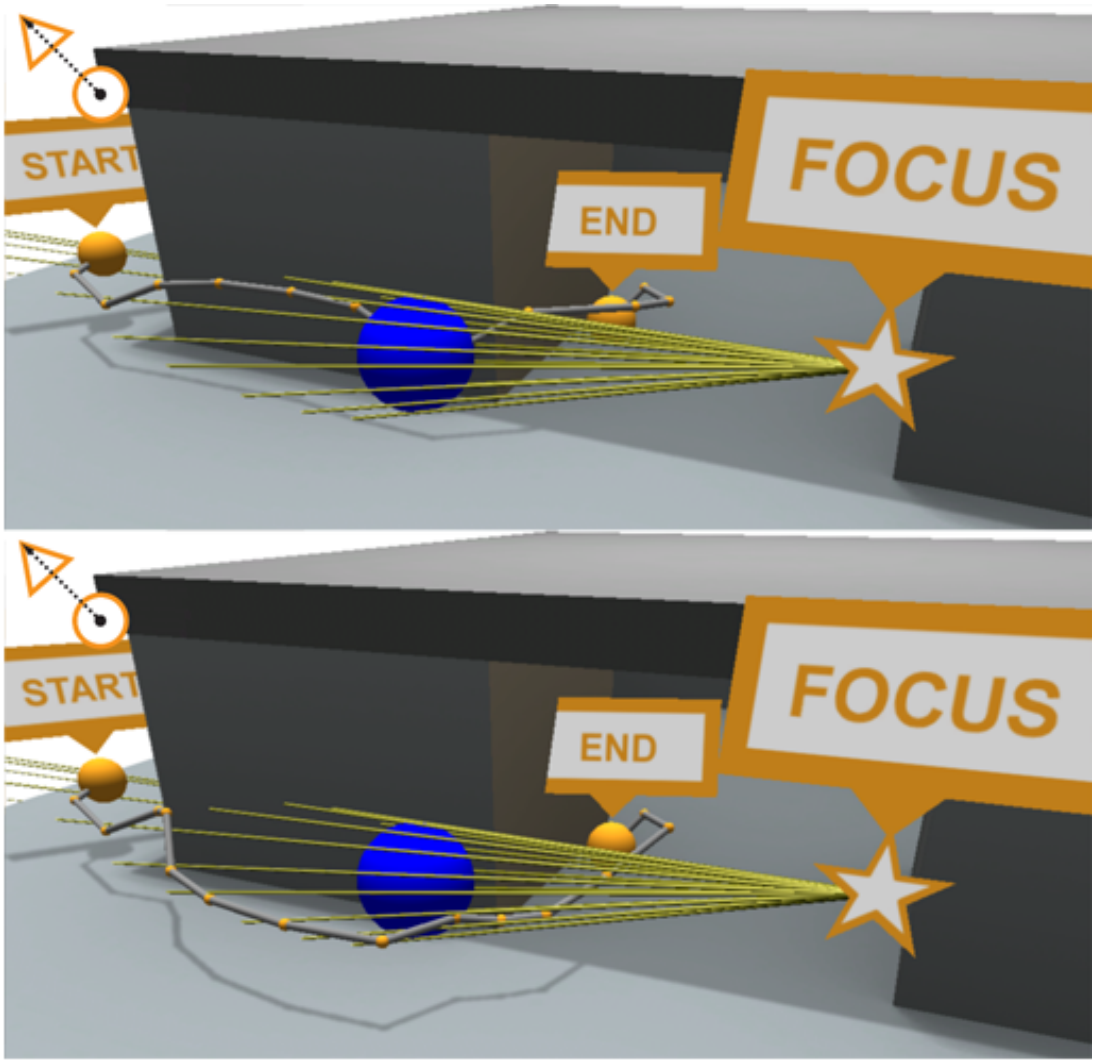


Figure 4.3.: *Screenshot of before adapting and after adapting the algorithm. Note that in the second picture the path not only avoids the blue obstacle, but also dodges the visibility cone* ***behind*** *the obstacle, whereas the area between the focus and the obstacle is free to be entered.*

5

Enhanced Smoothing

The sections before have described how the path finding algorithm can be modified in such a way that it becomes aware of dynamic obstacles in an environment and circumvent their visibility cones. But the returned result should rather be seen as an initial guess than the perfect solution. This is due to two reasons. On one hand, connecting the circle centers may introduce small indirections, because a node may be in the vincinity of a point in space but not necessarily covered. On the other hand, spheres in vast empty space can be relatively big and can leave a lot of space for path arrangement. In the given framework there is a post-processing step that takes care of these circumstances by iteratively smoothing the initial path. The changes made to this end did not address any of the post-processing algorithms. In order to still obtain a correct smoothing without any dynamic object or cone shadow intrusions the post-processing part has to be adapted as well.

The iterative smoothing algorithm optimizes the path according to two criteria. First, paths within spheres with full or no visibility are optimized to the shortest path, because the visibility will not change when the path is reshaped. Second, paths passing through spheres with partial visibility are redirected with the help of an occlusion map that is projected from the focus point. These occlusion maps contain border points which enter or exit the visibility region and shed light on in which environment the sphere is situated and accordingly, the path will be adapted.
Although we used CCCs as described in section 3.4 to prevent that due to smoothing the path runs through a moving obstacle, we cannot anticipate which spheres may be superfluous due to visibility reasons and therefore be skipped. Once a sphere is skipped, the path is not guaranteed to go through the "safe" CCC anymore and may indeed pierce a moving object.
Additionally to the piericing issue, there are visibility cones handicaping the path's passage. Using CCCs only helps to prevent direct dynamic object piercings, but has nothing to do with the visibility cones. When we adapted the algorithm for visibility cones we made it check for line intersections. A smoothing step, therefore, could simply intrude the cone area by moving

the node in its process, since it does not have a penalty mechanism.

5.1. Adapted Path Post-Processing

The smoothing algorithm works iteratively as mentioned before. It begins with the second node and works itself through to the second last node. The smoothing does not apply on the start and end node because they cannot be relocated. This procedure can be repeated until a desired threshold is reached for the difference between the current and the previous path.
In every step, a node n_i and its preceding n_{i-1} and subsequend node n_{i+1} are considered, an adequate smoothing is applied solely depending on their positions and their parent sphere's properties. The most radiacal smoothing we can apply to the node n_i is to skip it, meaning that nodes n_{i-1} and n_{i+1} will directly be connected. In doing so, this path segment would be a connection between two spheres, whereas, until now, the segments belonged clearly to only one sphere. Furthermore, the line segment could now even pierce a movable object or go through a cone area.

Based on this observation we can make a distinction of two cases that can occur: If n_{i-1} and n_{i+1}'s direct connections intersects either a dynamic obstacle or its cone it shall be processed with an *adapted* smoothing, otherwise the *regular* smoothing algorithm can be applied.
Each node in a sphere with partial visibility has an according occlusion map. An occlusion map is a projection of the environment on a circle plane from the focus point and allows us to see which part of the sphere is occluded, and which is free to be used. Besides the fact that the current node's predecessor and successor cannot be inside a dynamic object, there can basically occur four possible cases for their placement, but three of them are not relevant.
Those three cases consist of (1) both n_{i-1} and n_{i+1} being inside the cone, (2) either n_{i-1} inside the cone and n_{i+1} not or (3) vice versa.
These cases may occur when the path's starting or end node lie within the cone. If they appear, the part of the path that is inside the cone can be optimized by using the regular smoothing, with the slight modification checking for intersections with moving objects in case a node is being skipped.
The fourth case, where both points are outside the cone, should be the standard case because the path-finding algorithm tries to avoid intersections with the cone. By using the occlusion map and projecting the nodes on it, their positions can be determined and the right procedure can be used. An example for a possible situation is shown in Figure 5.1.

A recursive method is used to find a path minimally intersecting with the cone. First, by bisecting the hypothetical line segments (n_{i-1}, n_{i+1}) which returns a middle point m_1. This middle point m_1 gets projected onto the occulsion map. Then we use this 2D point $m_{1,2D}$ to find the closest point to $m_{1,2D}$ and reconstruct it in 3D and call it cp_1. The point will be node n_i's new position. Now the new n_i is closer to the cone and has shortened the path. This displacement, though, does not guarantee that the new segments will be intersection free. We need to test if they are intersection free, if not, the new segments can be tested subsequently, i.e. predecessor - new-current and new-current - successor, by calling the recursive method. This can be continued until a satisfying precision of the circumvention is reached. Although some smaller segment parts will still be intersecting the cone, a major penetration is prevented. This procedure might

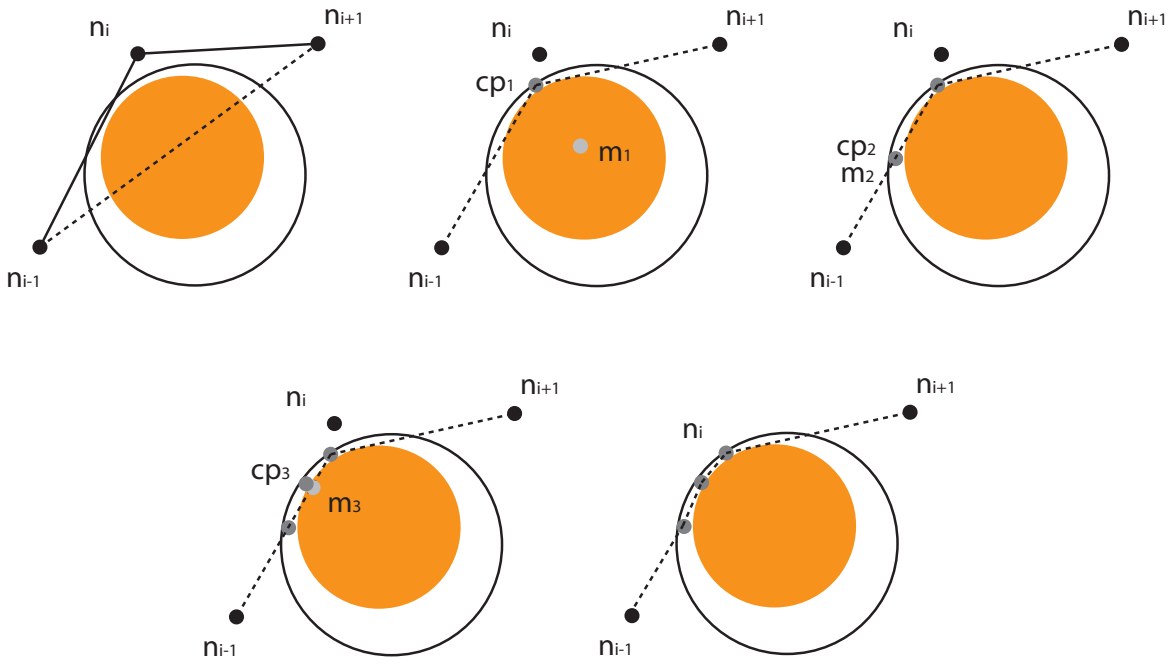


Figure 5.1.: *These series of images depict how the recursive method subDivide works. The orange circle within the white circle represents the cut of the occlusion map plane with the cone. This representation is not entirely correct in the sense that the projection step from 3D to 2D and vice versa has been skipped.*

create a number of new points which are all part of the path. As they are not regular nodes these points are defined to "belong" to node n_i and marked to be left in this state in a possible later iteration because they cannot be smoothed any further. The following pseudo-code implements the recursive method:

```
subDivide(prev, next, ref ListOfPoints, int iterations)
{
  if (iterations <= 0)
  {
    return;
  }
  else
  {
    mid = midpont between next & prev;
    new_p = closest point to mid that is not in the occluded area;

    if (left subdivision intersects cone && !threshold reached)
    {
      subDivide(prev, new_p, ref ListOfPoints, iterations - 1);
    }

    if (!threshold reached) LisfOfPoints.add(new_p);
```

```
    if (right subdivision intersects cone && !threshold reached)
    {
      subDivide(new_p, right, ref ListOfPoints, iterations - 1);
    }
  }
}
```

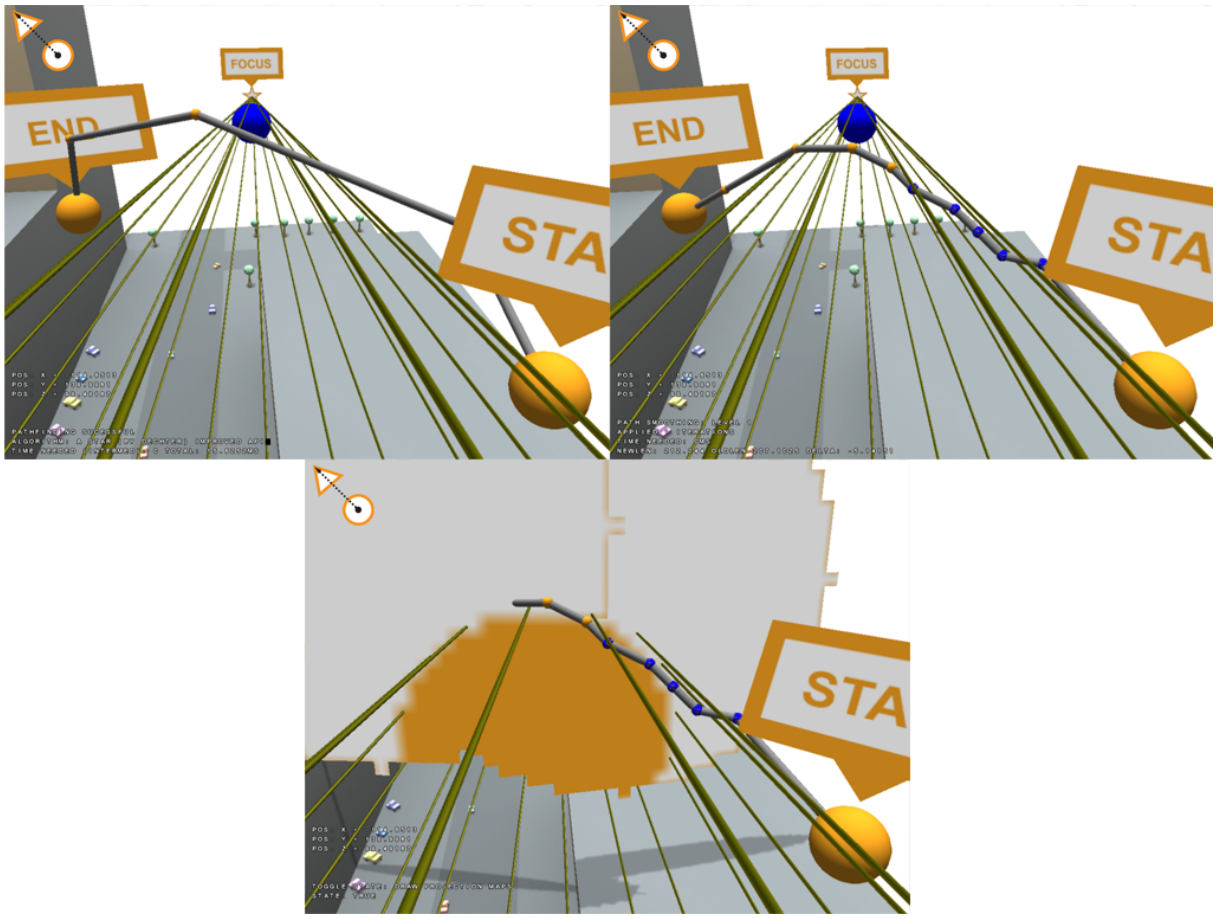


Figure 5.2.: *These pictures show an instance of an adapted smoothing. One can clearly see that the path nestles up against the cone. The third picture also includes an occlusion map with an orange colored circle approximation which is used to calculate the new points.*

Figure 5.2 shows nicely how the path goes around the cone keeping the visiual contact to the focus point. By using the smoothing operation described above, the second picture shows final version of such a smoothed path on which the recursive method was applied.

5.2. Thoughts on the Alternative Implementation

The alternative implementation as described in section 3.5 would have the advantage that the computation and testing of the CCCs would drop out in favour of computationally less heavy

line-sphere intersections tests. However, the load that has been taken from the path-finding-algorithm comes to lay on the post-processing part as soon as path smoothing is desired. Although the testing that would have to be done after a smoothing step would only involve intersection tests in the form of line-cone and line-sphere. A possible disadvantage of this alternative might be that due to the missing CCCs there might not be enough space to move the nodes during the smoothing step such that the resulting path would end up a lot more "edged" and less straight than the implemented solution.

6

Conclusion and Outlook

The goal of this thesis was to find a way to make the global camera control framework of [Osk08] notice moving objects in a 3D environment and include these objects into the path finding calculations.
In section 3.1 we have introduced a circle-to-circle-connection (CCC) detection structure and incorporated it into the Best-First search algorithm for it to be able to algorithmically "see" these new obstacles. In doing so, we enabled the algorithm to avoid these obstacles and return a valid path.
As the camera control framework concentrates also on the enhanced visibility towards a focus point, we needed to further adapt the apparatus in order to comply with these requirements. Any newly added object in the 3D environment has the effect, that a low- or non-visibility area towards a focus is cast behind that object. Analyzing and considering this shadow cone we extended the Best-First search algorithm once more. After these changes the algorithm was able to circumvent areas with lower visibility due to the shadow casting.
Letting the whole framework run as described now, it would still have given us visually incorrect solutions in some cases. This is due to the fact that the solution path we get returned from the adapted algorithm is not the final path. There is a smoothing step following that is also adapted. The smoothing is used to straighten the path and iron out offshoots as good as it is possible. We introduced a recursive subdividing algortihm that uses the already existing occlusion maps to smooth paths without allowing them to pass through forbidden areas.

We have intensively been studying the correctness of this implementation for one or more obstacles in the environment and did not focus on speed. Nonetheless, the implementation is usable in real-time applications. However, future work could focus on the several other possiblities to make the algorithm become aware of dynamically moving environments as pointed out in the alternative implementation section 3.5. To see whether the actual implemented CCC testing or the described alternative implementations would be more efficient and as pleasing as the current

one, would need further investigations, and also other implementations. Unfortunately due to time restrictions we were not able to undertake such a comparison, which gives material for future work.

Concluding, we can say that requirements of this thesis are fulfilled by the presented approaches. They are incorporated and working well in Oskam's currently continuing project.

A

Geometric Constructions and Implementation Details

This thesis presents an approach to make the given camera transition planning algorithm become aware of dynamic obstacles. For doing so, several geometrical and algebraic constructions are required and used in the preceding chapters. They are addressed in more detail here.

Generally, a circle C in 3D is defined by its center position m_c, its radius r_c and its normal vector $C = (m_c, r_c, \mathbf{n_c})$. In a similar way a sphere S is given by its center position and its radius $S = (m_s, r_s)$. A cone K is given by its vertex, its axis direction and the apex angle $K = (v_k, \mathbf{a_k}, \alpha_k)$.

A.1. Closest Point on a Circle

In section 3.3 and 3.4 we need to calculate the closest point on the rim of a circle to a sphere. We can use $\mathbf{n_c}$ to form a line l with the sphere center c_s

$$l = c_s + t\mathbf{n_c},$$

where $t \in \mathbb{R}$ and intersect l with the circle plane defined by $p_c = (m_c, \mathbf{n_c})$. By setting the line l and the plane p_c equal we can solve the vector equation. The intersection point i is used to create a vector v from the circle center m_c to i. Then v is normalized. This normalized vector is scaled by the circle radius r_c resulting in v_c. Adding the scaled vector to the circle center we get the closest point p_c we were looking for:

$$p_c = m_c + v_c$$

A.2. Circle - Sphere Intersection

To find out whether a circle in 3D is intersecting a sphere we can simply intersect the circle's plane with the sphere and compare whether the new circle's center and the original circle's center are farther away from each other than the sum of both radii.

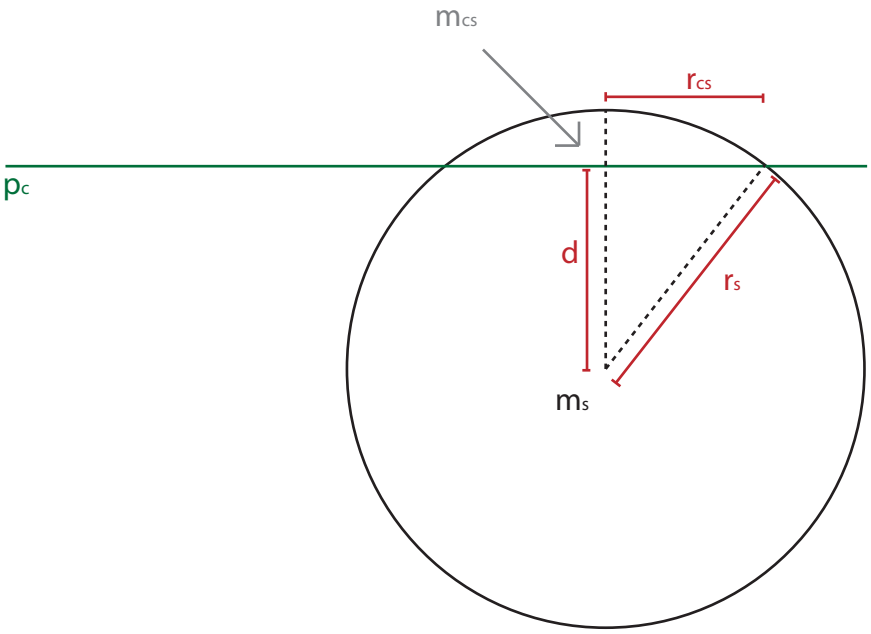


Figure A.1.: *This is a visualization of the intersection of a circle plane and sphere. The resulting circle's radius may be calculated using the Pythagorean theorem.*

Details:

- Express the circle's plane p_c with $(m_c, \mathbf{n_c})$.
- By intersecting p_c with S we get the circle C_s with center m_{cs}.
- To calculate the radius r_{cs} of C_s the geometric property shown in Figure A.1 can be used which results in $r_{cs} = \sqrt{r_s^2 - d^2}$, where d is the distance between the sphere center m_s and m_{cs}.
- If there is an intersection circle, compare distance $|m_{cs}m_c|$ with sum of radii $r_cs + r_c$

A.3. Midpoint Test for CCCs

This test is used once we have not found any intersection with the 'easier' intersection test. If the algorithm arrives at this test, the sphere S did not have any intersections with both circles C_1 and C_2 of a CCC. Neither did it intersect the closest line l_1 that constitutes of the above mentioned closest points to both circles.

By using the scalar resolute $\vec{a} \cdot \vec{b} = |a||b| \cos \alpha$ the line segment between the first circle's center and the sphere center $\overline{m_{c1}m_s}$ can be projected on the line segment between the two circle centers $\overline{m_{c1}m_{c2}}$ and result in a point P as shown in Figure A.2. $\overline{m_{c1}m_{c2}}$ has to be normalized:

$$\overline{m_{c1}m_{c2}} \cdot \overline{m_{c1}m_s} = \underbrace{|m_{c1}m_{c2}|}_{=1} |m_{c1}m_s| \cos \alpha = |m_{c1}P|$$

Adding $\overline{m_{c1}P}$ to center m_{c1} we get P. Then, we create a line l_2 between the point P and sphere center m_s. l_2 happens to be in the same plane as $l1$ and therefore we can intersect them and get a point Q. Due to numerical errors it might be that these two lines are skew. Hence, we need a robust method to get Q. The center of the shortest distance between two lines can be used here. Now considering that the algorithm went through other tests amonst others the closest line test, the following condition is sufficient to detect whether S is situated inside a non-overlapping CCC:

$$|Pm_s| < |PQ| \tag{A.1}$$

For overlapping CCCs another test has to be attached as described in 3.4.1.

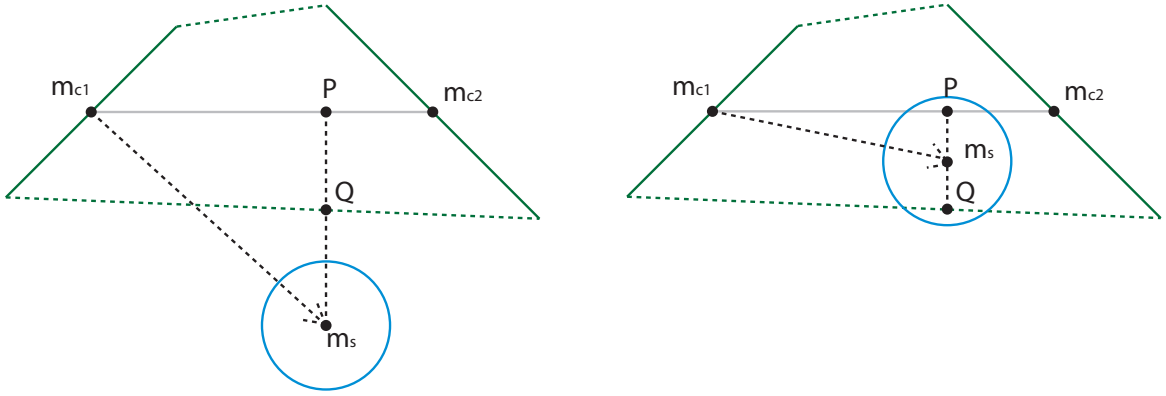


Figure A.2.: *The left image shows the case when condition* (A.1) *is false and the right one when it is true.*

A.4. Line - Cone Intersection

This section shows a mathematical derivation of the line-cone intersection solution described in section 4.1. For this section we rewrite the definition of a line with point P on the line and its direction D:

$$X(t) = P + tD, \text{ where } t \in \mathbb{R}$$

And here as a reminder to the reader the definition of a cone, with cone axis A, vertex V and apex angle α:

$$A^T(X - V) \geq |X - V| \cos \alpha$$

First we need to reforumlate the cone inequation. To get rid of the absolute value we need to square and substitute $Q = X - V$:

$$(A^T(X - V))^2 \geq |X - V|^2 \cos^2 \alpha$$

$$(A^TQ)^2 = (A^TQ)^T(A^TQ) = Q^TAA^TQ \geq Q^2\cos^2\alpha = Q^TQ\cos^2\alpha$$
$$Q^TAA^TQ - Q^TQ\cos^2\alpha \geq 0$$

Now we plug in $X(t)$ into the inequality:

$$(P + tD - V)^T(AA^T - I_3\cos^2\alpha)(P + tD - V) \geq 0$$

Now we expand the brackets and isolate the terms with t^2 and t:

$$t^2((D^TAA^TD) - (D^TD\cos^2\alpha))+$$
$$t(D^TAA^TQ - D^TQ\cos^2\alpha + Q^TAA^TD - Q^TD\cos^2\alpha)+$$
$$(Q^TAA^TQ - Q^TQ\cos^2\alpha) \geq 0$$

The coefficient can be declared as:

$$a = D^TAA^TD - D^TD\cos^2\alpha$$
$$b = 2 \cdot (D^TAA^TQ - D^TQ\cos^2\alpha)$$
$$c = Q^TAA^TQ - Q^TQ\cos^2\alpha$$

which can then be used in the well known formula for quadratic equations:

$$t_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Bibliography

[DP85] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *J. ACM*, 32(3):505–536, July 1985.

[Osk08] Thomas Oskam. Visibility transition planning for real-time camera control. *Master Thesis, ETH Zurich*, 2008.