Optimizing *The Sacrifice*

Nico Zevallos

***Abstract*—This work aims to finally fix a glaring issue in Andrei Tarkovsky's final opus *Offret* (The Sacrifice) [1] namely, the inefficiencies of its final six minute shot.**

I. Introduction

Fig. 1. A frame from the final shot of *Offret*

THERE are perhaps a handful of directors whose oeuvres are as universally venerated as that of Andrei Tarkovsky. Who among us could find fault in the work of a man Ingmar Bergman called "The greatest, the one who invented a new language, true to the nature of film, as it captures life as a reflection, life as a dream [2]." And yet as one watches *Offret*, one error looms. This 1986 masterpiece of Swedish cinema is marred by its famous final sequence. In it, the paterfamilias, Alexander, burns the family's home to the ground in an attempt to honor an agreement with God to undo a nuclear apocalypse. In a single, six minute long take, the family chases him as their house burns to the ground, splashing through and falling into the mud, occasionally collapsing in despair, and finally pushing him onto an ambulance that rolls away into the swamp. The family then watches as their house collapses onto itself in a glorious blaze, all the grandiose posturing of an aging academic conflagrated by a single act of faith.

As one watches this moving scene, one cannot help but think: Couldn't the family have captured and committed its crazed patriarch much more efficiently? The shot itself had to be re-done when a camera jammed during the first attempt, causing the crew to rebuild a perfect replica of the house from scratch to set fire to it a second time. Although nothing can be done at this point about the inefficiencies of the film's production, this paper seeks to optimize the paths the family takes as they rush to confront Alexander such that they may watch their home burn without the additional discomfort of getting their feet wet, as well as complete their task much faster.

II. Recreating the scene

The method for determining optimal, dry paths was as follows. The first step was to track the movement of the camera in the scene. This was done in the 3D modelling software Blender by creating rough models of the static objects in the scene, estimating camera parameters by eye, and manually moving the camera through the scene so that the static objects match up. Based on footage from a documentary about the film [3], the author constrained the movement of the camera to a single degree of freedom of movement along the axis of the dolly that the camera was on as well as two rotational degrees of freedom to account for horizontal and vertical pans. The camera focal lengths tested were 50mm, 70mm, 75mm, 85mm, and 100mm. Focal lengths below 50mm and above 100mm forced the author to make objects such as the house and the cars unreasonably small or large in order to line them up with the footage. After a rough alignment at each of these focal lengths, 75mm was chosen as the ideal focal length because it produced the fewest artifacts and distortions. Following this decision, the footage was meticulously tracked by hand, adding about 1 key frame for every 5 frames of footage and adjusting both the position of the camera and objects in the scene until the entire six minute shot was aligned. Once the shot was aligned, non-static objects representing the characters could also be key-framed to match their position from the camera's point of view, thereby finding their trajectory in the recreated 3D space. The recreated scene, including renderings of the house and cars present in the film as well as Alexander's trajectory through the scene can be seen in Fig. 2.

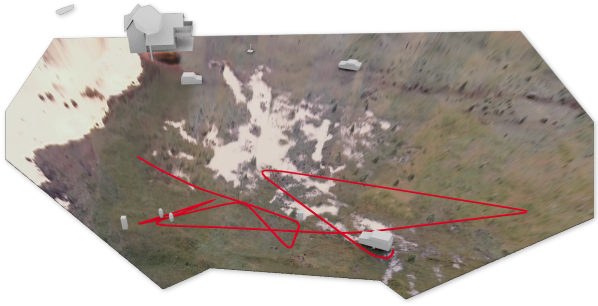

Fig. 2. Recreated scene with Alexander's path overlayed

The second step was to create a cost map for the path planner. The aligned footage was projected onto a plane to recreate the ground plane. This was done by repeatedly finding a frame in which a portion of the ground was visible, and then painting that portion of the ground onto the recreated ground plane similarly to the 'screen space brush' method

described by Hanrahan and Haeberli[4]. This reconstructed ground plane had a threshold applied to create a cost map, and unreachable areas such as those inside the house or under cars were manually marked as infinite cost. The resulting cost map can be seen in Fig. 3.

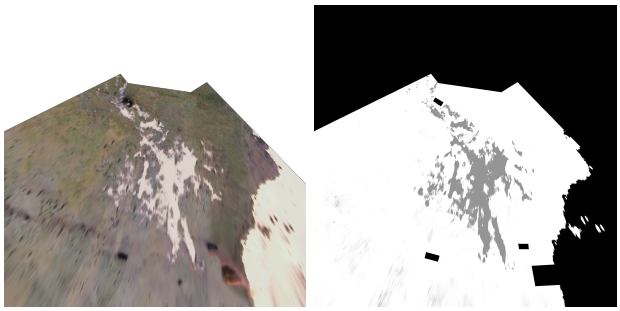

Fig. 3. Texture-painted ground plane on left and generated cost map on right

III. Calculating Optimal Paths

Once this cost map was created, the optimization could begin. This cost map was used to calculate how much of the path passed through 'water' pixels. This was done using Bresenham's line algorithm [5] to find the pixels covered by the path and finding the ratio of grey (in the water) to white (on dry land) pixels. This cost map was also used in conjunction with the A* planning algorithm [6] as an 8-connected grid in order to find paths that avoided wet areas. A* was chosen as it has the pleasant properties of being extremely fast, complete, and guaranteed to produce optimal paths.

One downside to the use of a grid representation, however, was that the paths were optimal only on the discrete space, where at each pixel the family member only has eight directions they could move. In reality the family could move in an infinite number of directions and weren't restricted to moving from the center of one pixel to another. To mitigate this effect, euclidean distance was used as the heuristic for A* rather than the more traditional diagonal distance used in most 8-connected grid representations. This was chosen because although diagonal distance is admissible in the 8-connected case, it can overestimate distance in the continuous case. In practice what this heuristic means is that paths tend to be closer to what they would be in a continuous space, and less likely to zig-zag and take advantage of cheap diagonal movement. An existing implementation found here was used with some modifications: changing the heuristic to Euclidean distance and changing the cost of diagonal movement to $\sqrt{2}$. Finally, an additional relaxation step was applied in order to produce smoother, shorter paths as proposed by Thorpe and Matthies[7]. An overview of the path-finding is provided in Algorithm 1.

Algorithm 1: Calculating optimal paths

input : pursuer location α_T,
n 2D target locations $\alpha = [\alpha_i, i \in \mathbb{N} : i < n]$
target location α_T
$m \times n$ grid of costs G

output: relaxed A* path p

Function `Relax(`p, G`)`**:**
  $\nabla G \leftarrow \nabla$ `GaussianBlur(`G`)`
  $gradient \leftarrow [0, 0, ..., 0]$
  for $i \leftarrow 1$ **to** $maxIterations$ **do**
    for $i \leftarrow 0$ **to** `Length(`p`)` **do**
      $x, y \leftarrow p_i$
      $gradient_i \leftarrow \nabla G_{xy}$
    $optimizeStep \leftarrow gradient + \Delta p$
    $optimizeStep_{start}, optimizeStep_{end} \leftarrow 0$
    if $||optimizeStep|| < 0.1$ **then**
      break
    $p \leftarrow p - optimizeStep$
  return $path$

Function `Main(`α, α_T, G`)`**:**
  $p \leftarrow$ `AstarSearch(`α_i, α_T, G`)`
  if `Length(`$path_{new}$`)` > 0 **then**
    $p \leftarrow$ `Relax (`$path_{new}$, G`)`
  else
    continue
  return p

IV. Optimal Path Results

As can clearly be seen in Table I, A* produces perfectly dry paths. Even after relaxation, our method produces paths which are 99% dry. The paths that are taken in the film range from 8-20% wet, leading two characters to fall over face first into the mud. In addition, Table I shows that the paths taken in the film are only marginally shorter than those found by A* and extremely close to those found using Relaxed A*. All of these paths are sub-optimal in terms of distance, which is clearly shown in the comparison to running in a straight line. It is almost as if the characters are going out of their way to get wet. Worse still, neither Marta nor Julia ever reach Alexander, choosing instead to fall to their knees a few meters short and are generally unhelpful for the rest of the sequence. For ease of comparison, this excess distance was added onto their 'Path in Film' lengths. As an additional visualization of the irrational path choice of the characters, Algorithm 1 was repeated for each frame and overlaid over the actual video. The video can be found here.

A portion of the sequence was chosen in which the entire family as well as Alexander were all clearly visible, so that we could have an apples to apples comparison of path quality without having to guess the location of off-screen characters. The starting point for each of the characters was set to their location at the first frame of the sequence. Because Alexander stays in approximately the same place for the entirety of this segment, the goal for each path was set to Alexander's median location.

TABLE I
PERCENTAGE OF PATH SPENT IN WATER

Character	A*	Relaxed A*	Path in film	Straight line to goal
Marta	0%	1.2%	12.8%	8.2%
Victor	0%	1.3%	7.7%	21.8%
Julia	0%	1.1%	16.1%	9.7%
Adelaide	0%	1.0%	19.5%	10.0%

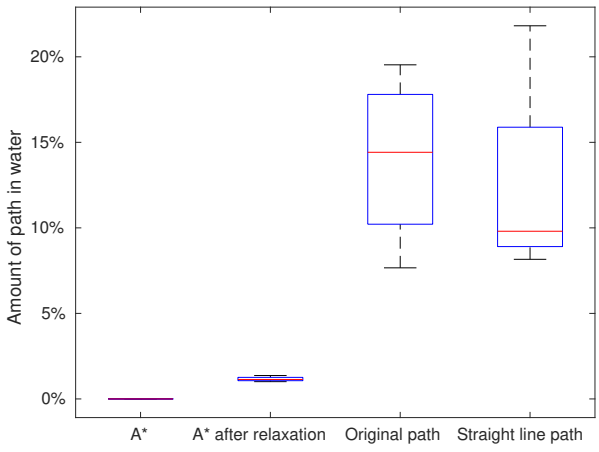

Fig. 4. Box plot showing results from Table I

TABLE II
LENGTHS OF PATHS

Character	A*	Relaxed A*	Path in film	Straight line to goal
Marta	76.7 m	72.8 m	72.0 m	70.7 m
Victor	63.3 m	60.2 m	61.8 m	58.0 m
Julia	84.3 m	80.5 m	79.6 m	77.6 m
Adelaide	82.7 m	78.6 m	79.3 m	76.1 m

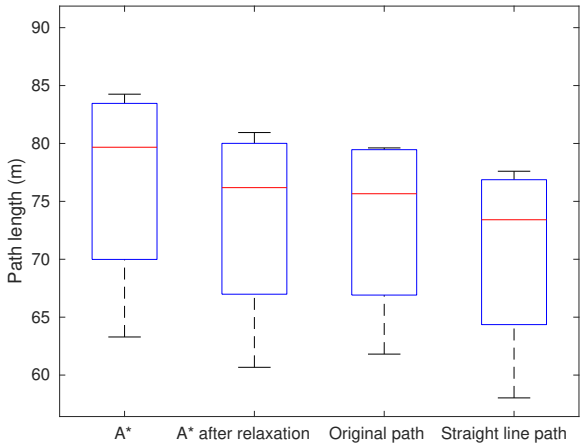

Fig. 5. Box plot showing results from Table II

V. PURSUIT AND EVASION

Although these comparisons are elucidating with regards to pursuer path quality, the author was additionally curious about what would happen if Alexander were actively trying to avoid capture, rather than wandering about in despair. In order to test this behaviour, we formulate the scene as an evasion-pursuit problem. First, a boundary is created which is a convex hull roughly approximating the borders of the cost map, ignoring obstacles.

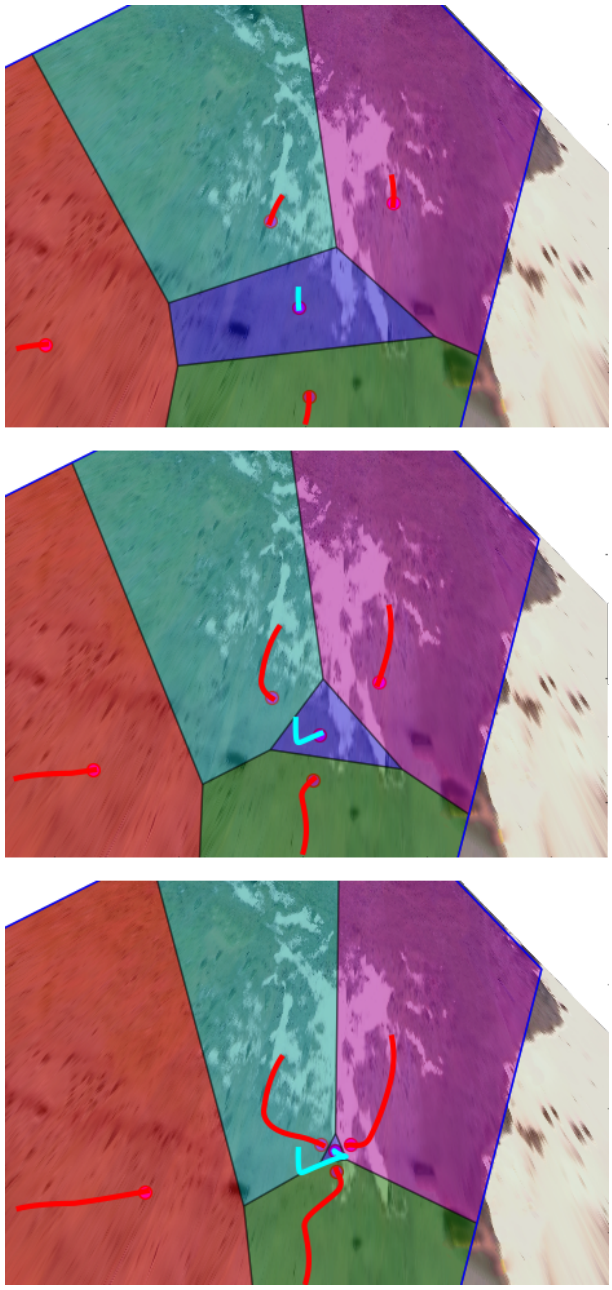

Fig. 6. Result of the pursuit algorithm, with pursuer paths marked in red and evader path marked in cyan.

Alexander employs the fleeing behaviour found in Algorithm 2. This is a very basic fleeing behaviour that avoids pursuers and coastal edges weighted by their distance from Alexander. The pursuers attempt to catch Alexander using Algorithm 3 which was developed by Huang et al.[8]. This algorithm was chosen as it guarantees capture in finite time and can be calculated in real-time. In this algorithm, rather than pursuing the target directly, a Voronoi diagram is constructed. Each pursuer checks if their region of the Voronoi diagram

borders the target's region. If it does, they move toward the center of the line defining the border between the two regions as fast as they can. If the pursuers region does not share any border edges with the target region, it simply moves toward the target as fast as it can. This results in the pursuers surrounding the target rather than bunching up during pursuit.

Previous work on this topic only performs simulation experiments on regions with square boundaries[8][9], which makes the calculation of distance to edge as well as the calculation of the bounded Voronoi diagram much simpler. In our case, the calculation was performed by first reflecting all points about every edge in the boundary polygon. The Voronoi diagram was then calculated using the union of the original points and their reflections. These reflections could additionally be used to calculate the distance from any point to all of the boundary edges by taking the differences between a point and each of its reflections and dividing by two.

For the purposes of the experiments, all actors were assumed to run at a human's optimal hunting speed of 3.3 m/s[10].

Algorithm 2: Evasion algorithm

in : n pursuer locations $\alpha = [\alpha_i, \forall i \in \mathbb{N} : i < n]$
target location α_T
convex hull B with k edges $[B_i, \forall i \in \mathbb{N} : i < k]$

out: Direction of evasion

Function `CalcPursuerDir(`α_T, α`)`**:**
  $\delta \leftarrow [0, 0]$
  for $i \leftarrow 0$ **to** `Length(`α`)` **do**
    $\epsilon \leftarrow \alpha_i - \alpha_T$
    $\delta_i \leftarrow \epsilon / \|\epsilon\|^2$
  return δ

Function `CalcEdgeDir(`α_T, B`)`**:**
  $\delta \leftarrow [0, 0]$
  for $i \leftarrow 0$ **to** `Length(`B`)` **do**
    $\epsilon \leftarrow$ `CalcEdgeDist(`B_i, α_T`)`
    $\delta_i \leftarrow \epsilon / \|\epsilon\|^2$
  return δ

Function `Main(`α_T, *pursuers*, B`)`**:**
  $edgeDir \leftarrow$ `CalcEdgeDir(`α_T, B`)`
  $pursuerDir \leftarrow$ `CalcPursuerDir(`α_T, α`)`
  $\delta \leftarrow \sum edgeDir + \sum pursuerDir$
  return $\delta / \|\delta\| * evaderSpeed$

VI. Pursuit and Evasion Results

To evaluate the effectiveness of our pursuit strategy, 100 trials were run with random starting positions for all of the actors. For each of these trials, the pursuers were given a maximum of one minute to capture Alexander. Based on observing the original film footage, it was determined that

Algorithm 3: Pursuit algorithm

in : n pursuer locations $\alpha = [\alpha_i, \forall i \in \mathbb{N} : i < n]$
target location α_T
convex hull B with k edges $[B_i, \forall i \in \mathbb{N} : i < k]$

out: Direction of pursuit for each pursuer

Function `Main(`α_T, α, B`)`**:**
  $\alpha_{reflected} \leftarrow$ `Reflect(`$[\alpha_T, \alpha]$, B`)`
  $regions \leftarrow$ `Voronoi(`$\alpha_{reflected}$`)`
  for $i \leftarrow 0$ **to** `Length(`α`)` **do**
    $edge \leftarrow$ `SharedEdge(`$regions_i$, $regions_T$`)`
    if $edge \neq \emptyset$ **then**
      $target \leftarrow (edge_a + edge_b)/2$
      $\delta \leftarrow \alpha_i - target$
      $\delta \leftarrow \delta / \|\delta\|$
    else
      $\delta \leftarrow \alpha_i - \alpha_T$
      $\delta \leftarrow \delta / \|\delta\|$
  return $\delta * pursuerSpeed$

it would take three pursuers to restrain Alexander and thus successfully 'capture' him. Once a pursuer reached Alexander, they would be removed from the Voronoi diagram and travel with him until three pursuers had reached him, at which point the trial was ended and reported successful. If three pursuers had not reached Alexander by the end of the one-minute mark, the trial was reported as a failure.

Alexander was caught in 100% of trials in an average of 16.3 seconds with a standard deviation of 4.5 seconds. The maximum time to capture was 28.0 seconds. In the film, Alexander runs free for a grand total of 5 minutes and 37 seconds before he is forced into the ambulance.

VII. Conclusion

The results of this paper confirm our suspicion that the final sequence of *Offret* contains a grave oversight when it comes to common-sense path planning. Clearly, Tarkovsky and his cast did not do even a cursory literary review. Otherwise, they would certainly have come across A* ([6] was published more than a decade prior to the release of the movie) and path relaxation ([7] was published more than a year before principal shooting began). As our pursuit and evasion results have shown, had the family employed even a simple surrounding technique, they would have caught Alexander an order of magnitude faster. Had the actors and director had the forsight to move in a more coordinated fashion, Tarkovsky may not have had to shoot this sequence twice, and even if he had, he would certainly have saved some money on film. These unfortunate oversights show yet again that the value of meticulous academic research trumps the brash, emotional decisions of even the greatest artists. As roboticists, scientists, and people of Reason, we continue to wonder: "Will they ever learn?"

REFERENCES

[1] Andrei Tarkovsky. *Offret*. Svenska Filminstitutet (SFI), 1986

[2] Paul Coates. *Film at the Intersection of High and Mass Culture*. Cambridge University Press. pp. 157-158, 1994.

[3] Michal Leszczylowski *Regi Andrej Tarkovskij*. Svenska Filminstitutet (SFI), 1988

[4] Pat Hanrahan, Paul Haeberli. Direct WYSIWYG Painting and Texturing on 3D Shapes. *Computer Graphics*, vol. 24, no. 4, pp. 215-223 August 1990.

[5] Jack Bresenham. Incremental Line Compaction. *The Computer Journal*, vol. 25, no. 1, pp.116-120, February 1982

[6] Peter E. Hart, Nils J. Nilsson, Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Intelligence/sigart Bulletin - SIGART* vol. 37. pp. 28-29, 1972.

[7] Charles E. Thorpe, L.H. Matthies. Path relaxation: Path planning for a mobile robot. *Proc. AAAI*, pp. 318-321, 1984.

[8] Haomiao Huang, Wei Zhang, Jerry Ding, Duan M. Stipanovi, Claire J. Tomlin. Guaranteed Decentralized Pursuit-Evasion in the Plane with Multiple Pursuers. *Proceedings of the IEEE Conference on Decision and Control*, 2011.

[9] Zhengyuan Zhou, Wei Zhang, Jerry Ding, Haomiao Huang, Duan M. Stipanovi, Claire J. Tomlin. Cooperative pursuit with Voronoi partitions. *Automatica* vol. 72, pp. 64-72, 2016.

[10] Karen L.Steudel-Numbers, Cara M.Wall-Scheffler. Optimal running speed and the evolution of hominin hunting strategies. *Journal of Human Evolution*. vol. 56, no. 4, pp. 355-360, April 2009.