ANIMATING A CAMERA FOR VIEWING A PLANAR POLYGON

DANIEL BRUNSTEIN
ANIMATING A CAMERA FOR VIEWING A PLANAR POLYGON

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
IN COMPUTER SCIENCE

DANIEL BRUNSTEIN

SUBMITTED TO THE SENATE OF THE TECHNION — ISRAEL INSTITUTE OF TECHNOLOGY
SIVAN, 5763 HAIFA JUNE, 2003
ACKNOWLEDGMENT

I wish to express my gratitude to my supervisors Dr. Gill Barequet and Assoc. Prof. Craig Gotsman for their guidance, support and help.

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GRATEFULLY ACKNOWLEDGED
## Contents

1 Introduction ......................................................... 1
   1.1 Related Work .................................................. 2

2 Camera Zones ....................................................... 4
   2.1 View Angle Criterion ......................................... 4
      2.1.1 Vertex-Vertex Visibility Segments ...................... 8
      2.1.2 Computing AVR ........................................... 11
      2.1.3 Creating the Visibility Zone ............................ 15
   2.2 Distance Criterion ............................................ 15
   2.3 Camera Zones .................................................. 16
   2.4 Common Zones ................................................ 17

3 Path Generation ................................................... 18
   3.1 Natural Cubic Spline Implementation ....................... 19
   3.2 Camera and Target Parameterization ........................ 20
   3.3 Path Cost Function ........................................... 21
      3.3.1 Keeping the Curve Within the Zone .................... 22
      3.3.2 Curvature Control ....................................... 23
3.3.3 Curve Length ........................................ 24
3.4 Minimizing the Cost Function .......................... 25
3.5 Scaling ..................................................... 26
3.6 Edge Affects Curve Length ............................. 27

4 Experimental Results ........................................ 29
  4.1 Implementation Details ................................ 32
  4.2 The Video .............................................. 33
  4.3 Applications ........................................... 33
    4.3.1 Overall inspection ................................. 33
    4.3.2 Guaranteed resolution ............................. 34

5 Summary and Future Work .................................. 35

A Computing Nonuniform Hermite Polynomials .......... 37
B Tangents of Nonuniform Natural Cubic Splines ........ 39
C Tangents in a Periodic Dimension ...................... 42
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Visibility Zone for non-occluded edge $e$.</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Occlusion-free, top, bottom, and mutually occluded regions.</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Circular arc $a_1$ in the bottom-occluded region.</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td>External occlusions, creating five areas.</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>Visibility contours of edge $e$ with occluder $o$.</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>A general ‘LR’ VVV.</td>
<td>9</td>
</tr>
<tr>
<td>2.7</td>
<td>All possible type combinations for a VVV.</td>
<td>10</td>
</tr>
<tr>
<td>2.8</td>
<td>Algorithm for computing VVV$s$.</td>
<td>10</td>
</tr>
<tr>
<td>2.9</td>
<td>AVR data structure of edge $e$.</td>
<td>12</td>
</tr>
<tr>
<td>2.10</td>
<td>All possible cases for updating the AVR.</td>
<td>12</td>
</tr>
<tr>
<td>2.11</td>
<td>Four Different AVRs.</td>
<td>14</td>
</tr>
<tr>
<td>2.12</td>
<td>Different camera zones.</td>
<td>16</td>
</tr>
<tr>
<td>2.13</td>
<td>Common zones.</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>General Hermite curve.</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Curve exceeding camera zone.</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>Control point perturbation order.</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>Edges affects curve example.</td>
<td>28</td>
</tr>
</tbody>
</table>
4.1 Path results I. ......................................................... 30
4.2 Path results II. ....................................................... 31
Abstract

Many applications, ranging from visualization applications (such as architectural walkthroughs) to robotic applications (such as surveillance), could benefit from an automatic camera trajectory planner. This thesis deals with that problem. We have automated the process of inspecting the outside of a simple two-dimensional polygon, given a few user parameters. Our algorithm preprocesses the polygon using visibility-graph-like concepts, and creates a visibility data structure for each polygon edge. From these structures, “good” camera zones are computed. Natural cubic splines are then used to create a closed camera trajectory that passes solely inside the zones. An iterative process refines the trajectory by minimizing a cost function until it converges to a good result.
Chapter 1

Introduction

Consider a planar shape that we want to visualize, or whose outside we want to inspect. In real life, this could be the floor plan of a 3D architectural model, an art gallery, a prison’s exterior, or even a toy within a complex scene of a motion picture film. In all these cases the shape may be represented as a simple polygon in the plane. Our aim is to find a closed path for the camera’s position and direction, so that the shape is filmed in a visually-pleasing way. Creating such a trajectory manually is a time-consuming task for an unexperienced animator, architect or 3D graphics artist. Many keyframes for the camera location and its target must be placed and iteratively revised. Hence a utility which automatically creates such a trajectory will release the animator from a lot of grunt work, allowing him to concentrate on the more important cinematographic aspects.

A trivial solution to the problem, namely placing control points next to edges or along the angular bisectors of vertices and connecting them to form a path, is far from adequate. Various problems such as path parameterization, camera look-at directions, occlusions and quality of the result immediately arise.

We would like to construct a camera trajectory that results in a visually pleasing animation of the polygon’s exterior. By “visually pleasing” we mean that the edges are sufficiently visible throughout the animation, the camera is not too close to the polygon, and the trajectory is sufficiently smooth and not unnecessarily long. In Chapter 2 we elaborate on the first two criteria, which define regions where certain discrete viewpoints of the camera should be placed. The visi-
1.1. RELATED WORK

bility criterion is the more difficult one, and most of our efforts focus on this criterion. In Chapter 3 we elaborate on the last two requirements which relate to the nature of the entire curve. Our results are presented in Chapter 4, and a summary and future work is discussed in Chapter 5. An appendix is included, in which the mathematical derivations are described.

In a nutshell, our algorithm proceeds as follows: In the first stage we preprocess the given polygon and create a data structure that contains visibility information related to the polygon. In the second stage we define regions for each of the polygon edges, based on the data structures. The regions, called zones, comply with certain user-defined requirements (constraints). A camera trajectory through these zones is created in the last stage by minimizing a user-defined cost of the curve.

1.1 Related Work

The focus of this work is cinematographic, since we are trying to find a trajectory for a camera which produces a given desired effect. This type of inverse problem has attracted the attention of computer graphics practitioners since the seminal paper “The virtual cinematographer” by He et al. [5]. The focus of their work is primarily on capturing the interaction of virtual actors. Our intention is not to follow a specific animated character, but to act as an observer. In another context, for a moving camera filming a two-dimensional landscape where historical data is visualized, a different approach was taken by Stoev et al. [11]. They position a virtual camera such that the projected area and the depth of the scene are maximized.

Cameras play an important role in robotic applications, and visual servoing (also known as image-based control) is widely used to control robots [6]. The paper most related to ours in the robotic arena [8] treats the control of a camera in a virtual environment, with the addition of constraints in order to react automatically to changes in the environment. A similar work by Halper et al. [4] arises from the field of computer games, where an automatic camera tracks the movement of a player. Halper et al. emphasize the trade-off between constraint satisfaction (guiding the camera controls) and frame coherence (smooth transitions with appropriate cuts). In both these works, which operate in near real-time scenarios, there is little emphasis on the
1.1. RELATED WORK

perfection of the visual quality and the global continuity of the camera path.

Another robotic approach to a similar problem is discussed in [13]. Here, near-optimal viewing positions for a robot-mounted camera are found heuristically. A path between a given sequence of singular visual tasks is generated. However, there is little emphasis on the viewing in between the visual tasks. In next-best-view algorithms [2], the 3D environment is unknown in advance, and the robot must sample the environment, and guess where the next best sampling point should be. Museum tour-guide algorithms [1] operate in real time, where the robot has to design a path and avoid colliding with other spectators.

Another aspect of our work is related to visibility computation. The visibility of a polygon’s edge, possibly occluded by other edges, is analogous to the case in which the edge acts as a uniform light emitter. This linear light source may cast shadows creating different regions (such as antiumbra and antipenumbra\(^1\)), as discussed in [12]. Stark et al. [10] deals with the illumination of a radiant polygon (in 3D space) obscured by a polygonal occluder, and the resulting shadow cast on a plane.

\(^1\)These terms correspond to different regions which are illuminated or partially illuminated by some linear light source.
Chapter 2

Camera Zones

The first stage of our algorithm is based on determining optimal camera location and view direction per polygon edge. These locations are such that the visible part of the edge extends a view angle of at least $\theta$ from the camera, and the camera is no closer than distance $d$ from the edge. In this section we show how to compute regions of the plane per edge which are the locus of the viewpoints satisfying these two requirement. These regions will be called camera zones.

2.1 View Angle Criterion

The view angle criterion, which is the foremost one, follows from the constraint that the view angle extended by the visible part of the edge be at least $\theta$. For a single edge $e$, the set of points satisfying this is a bounded region, which we call a visibility zone. The zone is created by preprocessing the polygon and creating the Angular Visibility Regions (AVR) data structure, which characterizes the visibility of each polygon edge in the plane.\(^1\) In the trivial case where there are no occluders in front of $e$, the visibility zone is the interior of a circular arc whose chord is $e$ and arc length (in radians) is $2\pi - \theta$ (see Figure 2.1).

\(^1\)The AVR is a multi-purpose data structure that can be used in various applications, such as optimal static camera placement.
2.1. VIEW ANGLE CRITERION

by \( v_1 \) and \( v_2 \) (see Figure 2.2). A convex vertex\(^2\) \( v_t \), reaching over (spiraling) from the top side of the edge, intrudes on the space in front of \( e \) and creates a region that is occluded from \( v_2 \) (we call this a “top-occluded” region). Another convex vertex \( v_b \) creates the symmetric region in which \( v_1 \) is occluded (the so-called “bottom-occluded” region). These occlusions may coexist and result in a region that is mutually occluded, namely neither \( v_1 \) nor \( v_2 \) are visible (the so-called “mutually-occluded” region).

In these four different regions (occlusion-free, top-occluded, bottom-occluded, and mutually-occluded) the visibility zone is computed differently. However, it is still bounded by circular arcs. The occlusion-free region is identical to the trivial case discussed above. In the bottom-occluded region, excluding the mutually-occluded region, the set of points seeing \( e \) with view angle \( \theta \), is a circular arc \( a_1 \) whose chord is \( v_2v_b \). This is because all points of that circular locus view \( v_2v_b \), hence \( v_2v_1' \) (where \( v_1' \) is the extension of \( v_b \) on \( e \) from the viewpoint), at the same angle \( \theta \) (Figure 2.3). Similarly, the top-occluded region creates a circular arc \( a_2 \) whose chord is \( v_1v_t \). In the mutually-occluding region, this locus is a circular arc \( a_3 \) whose chord is \( v_bv_t \), since all the locus points can see the edge \( e \) at the same angle \( \theta \) allowed by the opening of \( v_b \) and \( v_t \).

The union of these four regions yields the desired visibility zone for this case (shown in green in Figure 2.2).

\(^2\)A convex vertex is a polygon vertex whose internal angle is smaller than \( \theta \).
2.1. VIEW ANGLE CRITERION

Figure 2.2: Occlusion-free, top, bottom, and mutually occluded regions. The resulting visibility zone is overlaid in green.

Figure 2.3: Circular arc $a_1$ in the bottom-occluded region.
2.1. VIEW ANGLE CRITERION

A third, more complex case is when we want to support multiple polygons. Though not implemented, we describe the complexity of this scenario, in which several nontrivial occlusion situations are created. To illustrate the complexity of this case we consider a simple layout consisting of an edge \( e \), whose area in the front space is occluded by a single occluding line segment \( o \) (Figure 2.4).

This scenario introduces two new area types. The old types, “free,” “bottom,” and “top,” have the same properties as in the self-occlusion case. Circular arcs can be constructed, corresponding to different chords of circular arcs all having the same angular visibility \( \theta \). The area where \( e \) is completely occluded by \( o \) is called the umbra \([12]\). This area can be easily excluded from the zone. The direct implication of this is that our data structure has to support zones with holes. The problematic area is the “contained” area where the occluder \( o \) interferes with the visibility of the edge \( e \) by obscuring its middle part. From any point in the contained area, both \( v_1 \) and \( v_2 \) are visible, while the middle section of \( e \) is not. Points close to the umbra see the edge \( e \) partially from above and partially from below, but the sum of angles may be less than \( \theta \). Points further away might see the obstacle as very small, and the accumulated visibility of the edge \( e \) may be greater than \( \theta \). We are looking for the locus of all points which view the edge \( e \) at the same angle \( \theta \). For the time being, we do not know the explicit function that creates such a locus of points. In Figure 2.5 we present a numerical solution, showing equiangular contours. It can be clearly seen that these lines are far more complex than simple circular arcs. A slight change
2.1. VIEW ANGLE CRITERION

Figure 2.5: Numerical calculation of equiangular visibility contours of an edge $e$ with occluder $o$. Circular arcs are created in the occlusion-free, top and bottom-occluded areas, while the umbra is empty. In the contained area, the equiangular contours show nontrivial loci. Angles increase from blue to red.

in the orientation of the occluding edge $o$ causes a significant change in the contour shapes. In extreme cases where $o$ is large, or when its orientation towards $e$ is acute, no contained area is created at all. Handling this case highly increases our zone complexity, since holes within the zones must be allowed, and noncircular arcs must be treated.

2.1.1 Vertex-Vertex Visibility Segments

To construct the AVR data structure for an edge $e$, we need to detect the combinatorial changes in the visibility of $e$ that occur due to occluders. We define a VVV (Vertex-Vertex Visibility) to be a (possibly unbounded) directed line segment, passing through vertices $v_i$ and $v_j$, if they are mutually visible (i.e., the line segment between them lies completely outside the polygon), and at least one of them is convex. We associate up to four different edges with each VVV (see below). Crossing a VVV may change the visibility of some of these edges. We have extended the classical Visibility Graph [3] by adding this information to each VVV (Figure 2.6).

A VVV where the two vertices are nonconvex is not interesting since crossing it does not affect the visibility of any of the four adjacent edges. The VVV direction is arbitrary, but once fixed, the vertices and edges associated with the VVV are classified according to that direction.
2.1. VIEW ANGLE CRITERION

Each of the two vertices is classified as one of the types ‘R’, ‘L’, and ‘B’, for a total of six combinations (see Figure 2.7). The type specifies whether the designated vertex touches the VVV from its right side, from its left side, or from both sides (in case the vertex is nonconvex). Traversing the VVV along its defined direction, the source edge (if it exists) intersects it immediately before \( v_i \), and the target edge (if it exists) intersects it immediately after \( v_j \). The internal source edge is the edge incident to \( v_i \), that lies in the direction of the VVV. Symmetrically, the internal target edge is the edge incident to \( v_j \), that lies in the opposite direction of the VVV. By definition, there may be at most one source edge and/or one target edge per VVV. In the case where the vertex \( v_i \) is of type ‘B’, the VVV has two internal source edges (and naturally no source edge at all). The same holds for VVVs with vertex \( v_j \) of type ‘B’, which have two internal target edges and no target edges.

The algorithm to compute these VVVs is given in Figure 2.8. The running-time complexity of the procedure \( \text{CALCVVVS}() \) is clearly \( \Theta(n^3) \) in the worst case since there are \( \Theta(n^2) \) pairwise-visible vertices, and for each such pair seek an intersection with each of the \( n \) edges. By using ray-shooting techniques we can theoretically achieve a better running-time complexity.
Procedure \textit{CALC\textit{VVVs}}(\textit{P})

\textit{Input}: A simple polygon \textit{P} of \textit{n} vertices.
\textit{Output}: An array of valid VVV's.
\textbf{for each} \textit{vvv} of all possible vertex pairs of polygon \textit{P}
\hspace{1em} Set \textit{vvv} type (depicted in Figure 2.7);
\hspace{1em} \textbf{if} \textit{vvv} penetrates \textit{P} (by one of its two vertices) \textbf{or}
\hspace{1em} \hspace{1em} \textit{vvv} is of type ‘BB’
\hspace{1em} \hspace{1em} \textbf{then} Skip this \textit{vvv};
\hspace{1em} \hspace{1em} \textbf{endif}
\textbf{end for}
\textit{VVV}'s internal edges;
\textbf{for each} edge \textit{e} of the polygon \textit{P}
\hspace{1em} Find the intersection between \textit{e} and the \textit{vvv};
\hspace{1em} \textbf{if} no intersection
\hspace{1em} \hspace{1em} \textbf{then} skip the edge \textit{e};
\hspace{1em} \hspace{1em} \textbf{endif}
\hspace{1em} \textbf{if} the intersection lies between the \textit{vvv} vertices
\hspace{1em} \hspace{1em} \textbf{then} Exit edge loop and
\hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} Skip this \textit{vvv};
\hspace{1em} \hspace{1em} \textbf{endif}
\hspace{1em} \textbf{Link} edge \textit{e} to the \textit{vvv};
\textbf{end for}
\textbf{end for}

Figure 2.8: Algorithm for computing VVV's.
2.1.2 Computing AVRs

The AVR is a data structure that maintains, per edge, the different visibility regions created in the self-occluding case (as in Figure 2.2). As such, it is a planar map. The AVRs are constructed based on the VVVs. All the four different regions in the self-occluding case are simple polygons whose visibility relationships with the edge are different. Their visibilities are different since each region is created due to a different occluding vertex, and sees a different endpoint. This means storing, for each region, the occluding vertex and the relevant visible vertex. These two vertices are actually the endpoints of the chord of the circular arc for that region (see Figure 2.3). The mutual occlusion zone can be derived from the top and bottom occlusion regions; thus it is not explicitly present in the AVR.

We now describe the structure of a single AVR of an edge (depicted in Figure 2.9). The AVR consists of three fields: (a) A single polygon defining the “occlusion free” region which is linked to the two edge vertices \( v_i \) and \( v_{i+1} \); (b) A list of “top occluded” regions, which is a simple linked list of regions that are occluded from the edge by a top convex vertex. Each of these regions is stored as a polygon and linked to the appropriate chord, e.g., \( v_1 \) and \( v_t \) (for the single top-occluded area in Figure 2.2); and (c) A list of “bottom occluded” regions that are occluded from the bottom, with their corresponding chords, e.g., \( v_2 \) and \( v_b \).

The three AVR fields are initialized to the region in front of the edge exterior to the polygon, clipped to a large enough bounding-box. This is done by activating a clipping method on the edge \( e \), which finds the intersection between the halfplane defined by \( e \) and the polygon’s exterior. In order to deal with finite regions, we bound the polygon’s exterior by a bounding-box. Next, having already computed the VVVs, we collect for each edge all its VVVs and process them to create the AVR for that edge. Since each VVV causes a change in the region in front of the edge, we process each VVV separately and update the AVR. These changes can affect the full, top, or bottom occluding region list, by either removing or splitting part of the region (see the different cases in Figure 2.10).

Since the VVV is directed, we consider only the source and internal source edges. Thus the visibility effect occurs at the second VVV vertex (the second vertex along it). The target and
2.1. VIEW ANGLE CRITERION

Figure 2.9: AVR data structure of edge e.

Figure 2.10: All possible cases for updating the AVR. The edge in question is highlighted.
internal target edges are handled similarly. The two main actions are ‘Remove’ and ‘Split.’ Their input is either ‘left’ or ‘right,’ indicating the direction in which the action should be performed. ‘Remove left’ is activated when the region to the left of the VVV, after the second vertex, can be removed since it is completely obscured from the edge (see Figure 2.10(a), top case). The removal of the left region takes place in the three AVR data fields. In contrast, ‘Split left’ is activated when the region to the left of the VVV, after the second vertex, is about to become a “top occluded” region, since the second VVV vertex occludes the top vertex of the edge (see Figure 2.10(a), bottom case). In a ‘Split’ case we modify only the top or bottom occluded regions, depending on the specific situation, by splitting an existing region and creating two new subregions. In the ‘left’ case, the new left subregion will have the second VVV vertex as its occluder, while the right subregion will maintain its old occluder. Figure 2.11 shows some examples of AVRs.

Since there are \( O(n^2) \) VVVs, and each is connected to a constant number of edges, there are \( O(n^2) \) cases to treat. The Split() and Remove() methods are linear operations on the AVR polygonal structure, which requires \( O(n) \) time per edge. This is because we need to (a) Find the offending vertex (which is part of one of the top or bottom occluded AVR regions), (b) Calculate the intersection between the extension of the VVV and an edge of that region, and finally, (c) Split or remove the new region. The total complexity of the algorithm is therefore \( O(n^3) \).

The AVR induces an arrangement of line segments (possibly rays), defining a set of cells in a planar map. The visibility of the edge is the same for all points in a given cell, and different from cell to cell. With each cell we associate the two relevant vertices that are the endpoints of the chord of the circle whose visibility angle of the edge is \( \theta \). Computing the cells explicitly is done using polygon boolean operations on the arrangements. Apart from the cell that contains the edge \( e \), all the other cells are convex, since they originate from halfplane intersections and are bounded by externally convex portions of the polygon boundary.
Figure 2.11: Four different AVRs, two per polygon. The polygon is in blue, and the relevant edge is highlighted. The occlusion-free region is colored in transparent blue. The top-occluding region is colored in transparent purple, while the bottom-occluded region is colored in transparent green. Note the color mixing in the mutually occluded region.
2.1.3 Creating the Visibility Zone

The last step in the creation of the visibility zone is using the cells and the given view angle $\theta$ to find the zone per cell. In each cell, the angle $\theta$ defines a circle whose chord is delimited by the relevant vertices (shared with the cell). The zone per cell is the intersection of the cell and its circle (this intersection can be empty as well). Moreover, each point within the intersection region complies with the desired angle requirement.

By uniting all the zones obtained from the cells into one large region, we create the visibility zone for the given edge. All the interior points of the zone have a view angle to the edge larger than $\theta$, and all the boundary points “see” the edge at an angle of exactly $\theta$. The zone is a simple shape consisting of straight edges and circular arcs. In our implementation we approximated the arcs by polylines to facilitate boolean operations on the zones.

For the entire polygon, there are a total of $O(n^3)$ such induced arrangement cells. This is because there are exactly $n$ AVRIs, and for each AVR there is a total of $O(n)$ top and bottom occluding regions (which induce a total of $O(n^2)$ cells per AVR). The intersection between each cell and its corresponding circle can be computed in $O(\log n)$ time (due to convexity). Approximating the arcs by polylines varies only the complexity constant. Unitng these cell zones is again an operation whose running time is linear in the number of cells. This amounts to a complexity of $O(n^3 \log n)$. Theoretically, a better time complexity of $O(n^3)$ can be achieved by eliminating the process of explicitly computing the cells, and directly finding the boundary of the visibility zone. Although there can be $O(n^2)$ cells in the arrangement per AVR, the arc-bounded zone can pass at most $O(n)$ such cells. Thus we can track the visibility zone boundary along circular arcs, each time searching for the closest intersection with an AVR top or bottom region, and advancing only where needed.

2.2 Distance Criterion

Another important criterion for edge viewing is based on distance. We would like to prevent the camera from approaching an edge closer than a minimum given offset distance $d$. This value
2.3 Camera Zones

The intersection of the visibility zone and the offset zone is the camera zone. Using these criteria with varying parameters $\theta$ and $d$ allows us to control the quality and functionality of the camera trajectory.

A variety of different camera zones result from different combinations of the parameters (see Figure 2.12). For example, by using a small $\theta$ and a large $d$, we produce camera zones that are far away from the edges, which are suitable for scenic footage. A large $\theta$ and very small $d$ would yield simpler camera zones that are extremely close to the edges.
2.4. Common Zones

Consider the situation in which the camera zones of two consecutive edges do not intersect. This may easily happen at convex vertices if $\theta$ and $d$ are large enough. More precisely, if $\theta > 2\alpha$, where $\alpha$ is the angle at the vertex (e.g., consider $v_{i+2}$ in Figure 2.13 and increase the offset and angle). If the camera zones are disjoint, any path will pass through some points outside the zones. Since our requirement is that all points satisfy the criteria, we cannot allow this. Thus a region, called the **common zone**, that is the intersection of two consecutive camera zones, is defined in front of each common vertex. In the next section we use these regions to place the initial camera trajectory control points.

If the given parameters create an empty intersection between the camera zones of two consecutive edges of the polygon, our algorithm solves this locally. When we detect this situation, the criteria for the two corresponding edges are relaxed until the intersection of the two zones becomes nonempty. Namely, $\theta$ and $d$ are reduced, thereby increasing the areas of the two zones. The user is informed of this deviation from his specifications.
Chapter 3

Path Generation

In this section we show how to generate, given a polygon, an optimal camera trajectory. Having already processed the polygon, we have the camera zones—the locus of points from which the edge is seen at the desired qualitative level—for each edge.

We now define the path generation problem. Given an ordered list of simple polygons (the camera zones), each two consecutive polygons with a nonempty intersection, find a path lying entirely in the union of these polygons, passing through each in turn, and minimizing a given cost function.

We choose to use an interpolation scheme, since it allows simple yet powerful control of the complete path. By restricting a set of control points to certain areas, we are able to control the generation of the path. The smoothness of the path is achieved by imposing $C^2$ continuity. This is sufficient to remove any degree of freedom: each set of control points defines a unique path.

Thus, we restrict ourselves to paths that are cubic spline curves, interpolating one control point per common zone. In our setting a curve is a cubic curve connecting two control points, and a path is the closed $C^2$ curve connecting the point sequence.

A trivial uniform parameterization $t \in [0, 1]$ of each curve is inadequate in our application. We are interested in a smooth transition from one curve to another, and a uniform parameterization would cause the transition from a very long curve to a very short curve to be abrupt. The “velocity” in one curve should be similar to that in the neighboring curve. Hence we used an
approximation of an arclength parameterization, namely \( t \in [0, L_i] \), where \( L_i = |P_{i+1} - P_i| \) and the \( P_i \)'s are the control points.

The most important requirement from the path—that it reside completely inside the camera zones—is achieved by ensuring that every curve passes within its two corresponding common zones, hence we place the control points in these zones (see Figure 2.13).

## 3.1 Natural Cubic Spline Implementation

The natural cubic spline was implemented by using nonuniform Hermite curves for each pair of consecutive control points. A nonuniform Hermite curve is a cubic parametric curve, defined in the interval \( t \in [0, L] \). It connects between a pair of points \( P_0 \) and \( P_1 \), having tangents \( T_0 \) and \( T_1 \), respectively (Figure 3.1). The explicit formula is

\[
\gamma(t) = P_0 H_{00}^L + P_1 H_{10}^L + T_0 H_{01}^L + T_1 H_{11}^L, \tag{3.1}
\]

using the Hermite polynomials (see Appendix A for their derivation).

In the above formula

\[
H_{00}^L = \frac{2t^3}{L^3} - \frac{3t^2}{L^2} + 1, \quad H_{01}^L = \frac{-2t^3}{L^3} + \frac{3t^2}{L^2}, \quad H_{10}^L = \frac{t^3}{L^2} - \frac{2t^2}{L} + t, \quad \text{and} \quad H_{11}^L = \frac{-t^3}{L^2} - \frac{t^2}{L}.
\]

We note that the \( C^2 \) continuity requirement, and the piecewise concatenation of many such curves into one closed path, result in a natural cubic spline. Our only degree of freedom is the
movements of the points, so in order to find the explicit path, given a specific set of $n$ control points, we seek the appropriate tangent.

We now analyze the degrees of freedom we have in finding these tangents. We need a set of $n$ control points for a closed piecewise polynomial path of $n$ curves. In the equations $n$ Hermite curves there are a total of $4n$ unknowns. The $n$ given control points induce $2n$ equations (two equations per curve, one for each endpoint). Equating the tangent vectors (a tangent at the end of a curve should coincide with the tangent at the beginning of the next curve) generates $n$ more equations. By $C^2$ continuity at the meeting points of the curves, we obtain the remaining $n$ equations needed to determine the tangents.

Let $P$ be a column vector of the $n$ control points, and $T$ be a column vector of the $n$ corresponding tangents. We now use the given vector of points $P$ and solve the following system of equations in order to find $T$ (see Appendices B and C for the detailed calculation of the $n \times n$ matrix $M$, both in the planar $x$ and $y$ dimensions, and in the periodic angular $\alpha$ dimension):

$$T = M \cdot P.$$  \hspace{1cm} (3.2)

Once we have the specific control points and have computed the corresponding tangents, the unique natural cubic spline path is well defined.

### 3.2 Camera and Target Parameterization

A very important aspect, which has not yet been addressed, concerns the camera target. At which point in the polygon is the camera looking?

A conventional approach considers the target as a two-dimensional point in the plane, and creates a path connecting such points. The control points of this path may be the polygon vertices, and the path could be either an interpolated or an extrapolated one. The only constraint would be that the control points of the camera match the control points of the target path. This means that each camera path’s control point (that lies in the common zone) would look at the target specified by the vertex shared by the two edges. This approach was tested and rejected since the target path
can move significantly away from the polygon, enabling the camera path to pass in-between it and the polygon, thereby making the camera film away from the polygon.

The second approach considers the target not as an independent point in the plane, but rather as a vector pointing from the control point to the respective vertex of the polygon. We maintain an array containing the radial angles by which the camera “looks” from the control points at the corresponding vertices of the polygon. To create a smooth camera path, we interpolate the angles and rotate the camera accordingly. This introduces a parameterization problem. When we parameterize the angle uniformly, the curve stepping becomes discontinuous, and the film footage is bumpy. However, when we parameterize the curve position uniformly, the angle rotation stepping becomes discontinuous, and the film footage is bumpy again. We seeks for a way to parameterize these two dimensions together.

We found that a good way to treat the problem is to elevate it to one higher dimension. We view the path as a curve in the three-dimensional space of $x$, $y$, and $\alpha$. Each control point consists of its coordinates and the angle at which it “looks” at the corresponding vertex of the polygon. The parameterization in this space will make the transition from one control point to another smooth both in the spatial and the angular domains. Moreover, using this approach allows us to incorporate in the cost function factors that are solely dependent on the viewing direction.

### 3.3 Path Cost Function

The key to a successful path is a well-defined cost function, one that incorporates the features that we want, with the appropriate weights indicating the importance of each feature.

Our cost function incorporates both local and global features. We want the weights to be unitless, so features such as curvature and length are normalized by dividing the respective terms by the corresponding terms of a “normal” path. The normal path that we use is the unconstrained path whose control points are the polygon’s vertices. This is a path that, albeit partially penetrating into the polygon, has curvature and length proportional to the features of the polygon. Thus, it is a good entity to normalize with, and makes the minimum invariant under similarity...
3.3. PATH COST FUNCTION

The cost function that we use is:

\[ C = W_{\text{OutOfZone}} \cdot C_{\text{OutOfZone}} + W_{\text{Curvature}} \cdot \frac{C_{\text{Curvature}}}{C_{\text{DefaultCurvature}}} + W_{\text{Length}} \cdot \frac{C_{\text{Length}}}{C_{\text{DefaultLength}}} . \]

We now describe the different terms in this function.

### 3.3.1 Keeping the Curve Within the Zone

The most important property of the path we want to enforce is that each curve reside entirely within its corresponding camera zone. The two control points lie inside the zone, but nothing guarantees that the entire curve does so as well. We charge a penalty for each time the curve leaves the camera zone, by summing up the lengths of the zone boundaries that correspond to the portions of the curve where it passes outside the polygon (\( b_1 + b_2 \) in Figure 3.2). We call this zone boundary the *external boundary*.

By minimizing the external boundary, we “encourage” the iterative process to move the curve control points to a configuration in which the curve lies completely inside the zone.

In order to find the external boundary we process the edges of the zone in order, and for each edge compute its possible intersections with the curve. This amounts to solving a cubic equation. Note that the crossing points of the curve in and out of the zone partitions its boundary into two complementary portions. We choose the shorter one to be the external boundary. Though this is not theoretically guaranteed, we found experimentally that this choice was always correct. However, we could easily perform one ray-shooting query to determine the correct external boundary.

![Figure 3.2: The external boundary of the camera zone (in bold red), consisting of two zone boundaries, \( b_1 \) and \( b_2 \), that correspond to external portions of the curve.](image)
The cost is the sum of the lengths of these segments.

Although the most appropriate entity to minimize is the area contained between the zone and the exceeding curve, our method is much simpler and seems to have the same effect in practice. Initially we considered the minimization of the length of the curve outside of the zone. However, this was not always sufficient. Refer to the external boundary on the left, \( b_1 \), in Figure 3.2. If the zone edges were parallel in that region, the curve outside of the zone would be of approximately the same length all along, and the process would not “encourage” the control point to move leftwards.

### 3.3.2 Curvature Control

Another important constraint we impose on the path of the camera is that it have small curvature everywhere, as much as possible.

The cubic splines we use, \( \gamma(t) \), have a non-arclength parameterization (this is true for most polynomial curves). Its curvature \( \kappa \) is given by

\[
\kappa(t) = \frac{\| \dot{\gamma}(t) \times \ddot{\gamma}(t) \|}{\| \dot{\gamma}(t) \|^3}.
\]

The maximum of this function cannot be expressed analytically, so we compute it numerically by sampling the curve. The higher the curvature is (for example, at the vicinity of a small peak), the denser the sampling should be to capture it. Conversely, a straight line (having \( \kappa = 0 \)) naturally needs only two sampling points. We compute the maximum curvature of each curve and the cost is the sum of these maxima for all the curves of the path.

The cubic polynomial curve can be written as \( \gamma(t) = At^3 + Bt^2 + Ct + D \). Its derivatives in three dimensions are:

\[
\gamma(t) = \begin{bmatrix}
A_x t^3 + B_x t^2 + C_x t + D_x \\
A_y t^3 + B_y t^2 + C_y t + D_y \\
A_\alpha t^3 + B_\alpha t^2 + C_\alpha t + D_\alpha
\end{bmatrix},
\]
3.3. PATH COST FUNCTION

\[ \dot{\gamma}(t) = \begin{bmatrix} 3A_x t^2 + 2B_x t + C_x \\ 3A_y t^2 + 2B_y t + C_y \\ 3A_\alpha t^2 + 2B_\alpha t + C_\alpha \end{bmatrix}, \]

\[ \ddot{\gamma}(t) = \begin{bmatrix} 6A_x t + 2B_x \\ 6A_y t + 2B_y \\ 6A_\alpha t + 2B_\alpha \end{bmatrix}. \]

The numerator of \( \kappa(t) \) is:

\[ \dot{\gamma}(t) \times \ddot{\gamma}(t) = \begin{bmatrix} 6(A_\alpha B_y - A_y B_\alpha) t^2 + 6(A_\alpha C_y - A_y C_\alpha) t + 2(B_\alpha C_y - B_y C_\alpha) \\ 6(A_x B_\alpha - A_\alpha B_x) t^2 + 6(A_x C_\alpha - A_\alpha C_x) t + 2(B_x C_\alpha - B_\alpha C_x) \\ 6(A_y B_x - A_x B_y) t^2 + 6(A_y C_x - A_x C_y) t + 2(B_y C_x - B_x C_y) \end{bmatrix}. \]

We also have

\[ \|\dot{\gamma}(t)\| = \sqrt{(3A_x t^2 + 2B_x t + C_x)^2 + (3A_y t^2 + 2B_y t + C_y)^2 + (3A_\alpha t^2 + 2B_\alpha t + C_\alpha)^2}, \]

so the denominator of \( \kappa(t) \) is

\[ \|\dot{\gamma}(t)\|^3 = [9(A_x^2 + A_y^2 + A_\alpha^2)t^4 + 12(A_x B_x + A_y B_y + A_\alpha B_\alpha)t^3 \\
+ (6A_x C_x + 6A_y C_y + 6A_\alpha C_\alpha + 4B_x^2 + 4B_y^2 + 4B_\alpha^2)t^2 \\
+ 4(B_x C_x + B_y C_y + B_\alpha C_\alpha)t + (C_x^2 + C_y^2 + C_\alpha^2)]^{\frac{3}{2}}. \]

These equations provide us with a discrete representation of the curvature of a curve. We track the maximum curvature of each curve which we use in two ways: (a) Keep the global maximum curvature of the entire path; and (b) Sum the maxima for all the curves (since (a) does not take into account the curvature distribution apart from the maximum). In this way we incorporate both a local and a global curvature minimization scheme.

### 3.3.3 Curve Length

Another important feature which we wish to consider is the total length of the path. We want the path to be as short as possible, so that the camera will neither need to travel far, nor rotate much. In order to do that, we sum up the lengths of all the curves along the path.
In general, the length of any parametric curve \( \gamma(t) \) in the range \( t \in [t_0, t_1] \) is calculated by integrating over the norm of the first derivative:

\[
S = \int_{t_0}^{t_1} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2 + \dot{\alpha}(t)^2} \, dt.
\]

In our case the length is

\[
S = \int_{t_0}^{t_1} \sqrt{9(a_x^2 + a_y^2 + a_\alpha^2)t^4 + 12(a_x b_x + a_y b_y + a_\alpha b_\alpha)t^3}
+ 12(a_x d_x + a_y d_y + a_\alpha d_\alpha)t^2
+ 4(b_x d_x + b_y d_y + b_\alpha d_\alpha)t + (c_x^2 + c_y^2 + c_\alpha^2) \, dt.
\]

The integral representing the curve length is approximated numerically using a recursive-adaptive Simpson quadrature. The total sum over all curves of the path is then accumulated.

### 3.4 Minimizing the Cost Function

We use a steepest-descent procedure to find a path that minimizes the cost function. In each iteration we locally perturb each control point, one at a time. The order of perturbing the points is random. A set of candidate control points is checked in a circular neighborhood of the current point \( p_i \), excluding those lying outside the common zones. The path and cost function is evaluated for all candidates. If the cost of the best path is better (lower) than that of the original path, then the respective point replaces the point \( p_i \).

Updating the control points in a sequential order may cause the procedure to get stuck in a local minimum (see Figure 3.3). This is evident when it generates an asymmetric solution for a symmetric input polygon. A random permutation of point updates produces a better result, converging to a symmetrical path.
3.5 Scaling

We encountered the problem of considering simultaneously the $x$, $y$, and $\alpha$ domains, arising from mixing two incompatible types of dimensions. The axes $x$ and $y$ represent coordinates in the spatial domain of the polygon, whereas $\alpha$ represents angles. We need to calibrate the effects of the different dimensions in order to control them. For example, we may amplify the effect of the angular dimension with respect to that of the planar dimensions by applying a larger scaling factor on the $\alpha$ axis.

If we scale one dimension by a scalar factor $s$, then the tangents are scaled by the same factor. This is implied directly by Equation 3.2. Multiplying the point coordinate vector by the scalar, $s$, will result in scaling the tangent vector, becoming $sT$.

The first and second derivatives of a curve whose control points have been scaled by $s$ are also linearly scaled, since the curve is a linear combination of points and tangents (Equation 3.1), becoming $s\dot{\gamma}(t)$ and $s\ddot{\gamma}(t)$.

The curvature is now

$$\kappa(t) = \frac{\| (s\dot{\gamma}(t)) \times (s\ddot{\gamma}(t)) \|}{\| s\dot{\gamma}(t) \|^3}.$$
3.6. EDGE AFFECTS CURVE LENGTH

where

\[(s\dot{\gamma}(t)) \times (s\ddot{\gamma}(t)) = \begin{bmatrix} sy s_\alpha \\ sx s_\alpha \\ sx sy \end{bmatrix} (\dot{\gamma}(t) \times \ddot{\gamma}(t))\]

and

\[\|s\dot{\gamma}(t)\|^3 = [s_x^2 \dot{\gamma}_x^2(t) + s_y^2 \dot{\gamma}_y^2(t) + s_\alpha^2 \dot{\gamma}_\alpha^2(t)]^{\frac{3}{2}}.\]

Similarly, the length is now

\[S = \int_0^L \sqrt{s_x^2 \dot{\gamma}_x^2(t) + s_y^2 \dot{\gamma}_y^2(t) + s_\alpha^2 \dot{\gamma}_\alpha^2(t)} \, dt.\]

3.6 Edge Affects Curve Length

A so-far unattended issue is the problem that the polygon geometry does not affect directly the generated path. The curve between two consecutive control points in the \(xyo\) space is parameterized regardless of the size (length), distance from the control points, and orientation of the edge that lies in-between the two corresponding vertices. Consider Figure 3.4 and the two curves \(\gamma_{i,i+1}(t)\) and \(\gamma_{j,j+1}(t)\) defined by pairs of control points. Since the control points of each pair are equally far apart (both in the spatial and angular space), the parameterization is identical. This is incorrect. We would like to have a different parameterization implied by the respective polygon edges.

We solve this by assuming the length of the curve to be a combination of the Euclidean distance between the control points, and the length of the respective edge. This solves the problem since, although the control points are not moved, their tangents are affected by the artificial length, which also affects the parameterization of the path.
Figure 3.4: Two identical curves, corresponding to two different edges. The parameterizations are affected by the edge lengths.
Chapter 4

Experimental Results

The three-dimensional path that we obtain is smooth and the visualization of the polygon when traveling the path is visually pleasing. Within its confines, the user has several degrees of freedom: The first user choice is the view angle $\theta$ and offset $d$. Adjusting these parameters influences the path directly, as these define the regions through which the path must pass. The second degree of freedom is the cost function with its different weights, and the relative $xy\alpha$ axis scaling factors.

Some static path results are depicted in Figure 4.1 and Figure 4.2, including the ones featured in the video accompanying this thesis.

A full set of results is shown in Table 4.1. In our implementation, we set $W_{OutOfZone} = 10^6$ (six orders of magnitude larger than the other weights), to ensure that the path does not leave the zones. The other three weights were given varying values, and modifying them affected the path accordingly. The following weights were tested and produce the best results: $W_{GlobalCurvature} = 0.8$, $W_{LocalCurvature} = 0.4$, and $W_{Length} = 0.4$. The axis scaling factors used in almost all the polygons were $(C_x = 1, C_y = 1, C_\alpha = 1)$. In the Abraham model the factors used were $(C_x = 1, C_y = 1, C_\alpha = 1.5)$, giving a slightly higher importance to the angular axis. This gave us the desired effect of scanning the interior of the model slowly. All control points were initially placed at the common zone’s bounding box center (if the center was inside the common zone), or otherwise on the closest common zone point. The control points were updated in random order in each iteration. The “Edge Affects Curve Length” add-on was activated only in the Abraham
Figure 4.1: The TShape, CmpPoly, Snail, BigPoly and Wierd polygons camera paths (in blue) generated by our algorithm. The red line segments are the camera view directions.
Figure 4.2: The Comb, Ear and Abraham polygons camera paths generated by our algorithm.
Table 4.1: Some statistics of our experiments. The polygons complexity (Comp.) is the number of vertices of the polygon. The angle $\theta$ is specified in degrees, and $d$ is the offset distance multiplied by the edge length. All time measurements are in seconds. The average (Avg.) AVR is the average number of regions per AVR in the polygon. The maximum (Max) AVR is the maximum number of regions found in an AVR. The respective maximum cells (Max Cells) counts the number of cells in that particular AVR.

Polygon, since this parameterization problem was the most influential in this particular example.

Since the algorithm is nondeterministic, several runs of the algorithm were performed with different initial conditions to obtain different paths with different costs (corresponding to different local minima of the cost function), and then the best result are kept.

### 4.1 Implementation Details

We implemented the procedures described here in the Matlab [7] environment. This environment has a large variety of built-in tools, such as polygon boolean operations, numerical solvers, and 2D and 3D graphics capabilities, which permitted relatively easy implementation.

Our implementation has not yet been optimized for speed. Matlab is considerably slower than C++ due to its interpretative nature. There are numerous steps in which the performance of our algorithm could be boosted, such as using a faster interpolating scheme (rather than natural cubic splines), faster MinMax solvers (rather than naïve steepest descent), and more efficient line-sweep algorithms to manipulate the planar maps.
Table 4.1 summarizes some experimental results obtained by running our algorithm on a 2.4Ghz Pentium 4 with 512MB of RDRAM. It seems that the run time of the AVR computation is linear in polygon size, while the run time of the path optimization is a different order of magnitude. On average, the AVR size is small, however the maximal AVR size is linked to the geometrical complexity of the polygon (the more cavities and convex vertices the polygon has, the higher the size of AVRs and their complexity).

4.2 The Video

A five-minute video accompanies this thesis. It may be downloaded (in a variety of formats) from ftp://ftp.cs.technion.ac.il/pub/misc/cgge/public/Dani/VMV2003/ . The video demonstrates the various stages of our algorithm and gives two examples: the exteriors of a simple polygon and of a furnished house. These animations demonstrate the good quality of the camera paths, which were generated automatically.

The two-dimensional video clips were generated using Matlab. The results (camera trajectory and view direction) were outputed in VRML format, and imported into the scenes using 3D Studio Max, which was used also to render the video clips. The final clip was edited in Adobe Premiere, where the vocal sound tracks were added. The final high quality video was generated with the DivX codec,\(^1\) which encorporates MPEG4 compression, available from www.divx.com . The video resolution is \(524 \times 412\) at a frame rate of 30 frames per second. Other lower resolution clips are also available, using the well known Intel Indeo 5.1 codec.

4.3 Applications

4.3.1 Overall inspection

The basic application of our technique is for inspecting a two-dimensional model to appreciate its overall appearance. This includes virtual situations, such as the examination of a futuristic building (exterior and interior), as well as real-life situations such as navigating a complex maze.

\(^1\)Codec: COding and DECoding blackbox plugins.
In such scenarios we are not interested in the exact details of the model’s surface, but rather in the general idea of its geometry.

Since our goal is to step back from the model as much as possible in order to get an overview of it, we specify a small view angle parameter (allowing for distant shots) and a large offset parameter. This results in big far-away zones, that allow distant picture shots, and the paths thus generated result in the desired viewing clips.

### 4.3.2 Guaranteed resolution

Surveillance or close visual inspection of a wall is another natural application of our algorithm. Consider a prison guard who needs to check the integrity of the prison’s perimeter fence; a border patrol checking the border dirt path for an intruder’s footprints; or an automatic robot probe scanning the integrity of a nuclear reactor wall. All these problems are basically the same: inspecting or scanning a two-dimensional object, where it is most important to maintain a minimal degree of visual resolution. Our algorithm provides a lower bound on the resolution of the scanned edge.
Chapter 5

Summary and Future Work

We have automated the task of filming the exterior of an arbitrarily planar shape. We show how to create a closed camera trajectory for filming the outside of a two-dimensional polygon to a pleasing level. The user need only modify the two viewing parameters, the cost function weights, and/or the scaling factors. It is important to note that since our path is constrained by the number of control points, consists of cubic curves, and is $C^2$ continuous, there are scenarios for which no solution exists. For example there is no path that can enter a long, narrow, alley.

The theoretical overall running-time complexity of the algorithm that is described in Chapter 2 is $O(n^3)$. (The time complexity of our actual implementation is a little higher.) The complexity of the iterative process that minimizes a heuristic cost function (described in Chapter 3) is difficult to calculate. In fact, optimizing a “blackbox” unknown function is intractable [9]. Our experiments show that this stage is by far the most time-consuming part of the overall algorithm. Note, however, that this can be improved significantly by applying many local improvements.

We see several different avenues for future work. For architecture-based applications and animations, more focus should be placed on visualizing the complete structure (less than on each edge). A proper walking or flying sensation should also be created, for example, by looking in the direction of the camera movement, thus always moving forward rather than sideways panning. Walking up stairs, or tilting the view direction in order to stare at the building’s height can also be done. For robotic inspection, we should possibly focus on guaranteeing details and resolution on the scanned surface, maintaining a short and collision-free path.
This work addresses the problem of a single input polygon. In real-world applications, multiple polygons are present in the scene. This creates several nontrivial occlusion situations, and, among other complications, requires our data structure to support polygons with holes.

Finally, another natural extension is the generalization of our technique to three dimensions, namely, for planning a camera path to inspecting the boundary of a spatial polyhedron.
Appendix A

Computing Nonuniform Hermite Polynomials

The following notations are used throughout this thesis:

The distance between the points \( P_i \) and \( P_{i+1} \) is denoted as \( L_i \). This implies that each curve parameterization is \( t \in [0, L_i] \). The second derivative of the Hermite function \( H_{00} \) in the interval with distance \( L_i \) is denoted as \( \ddot{H}_{00}^{L_i} \). The second derivative of the curve \( \gamma \) between the points \( P_i \) and \( P_{i+1} \) with the parameterization of \( L_i \) at point \( t \) is denoted as \( \dot{\gamma}_{L_i}^{i}(t) \), for \( 1 \leq i \leq n \) (where \( P_{n+1} \equiv P_1 \)).

Following is the mathematical derivation of the nonuniform Hermite polynomials, \( H_{00}^L, H_{01}^L, H_{10}^L \) and \( H_{11}^L \), mentioned in section 3.1. We have a cubic curve in the form of \( \gamma(t) = at^3 + bt^2 + ct + d \) in the range of \( t \in [0, L] \), which we want to interpolate between the points \( P_i \) and \( P_{i+1} \), having tangents \( T_i \) and \( T_{i+1} \) respectively.

The curve can be also written as

\[
\gamma(t) = \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, \quad (A.1)
\]

and its derivative as

\[
\dot{\gamma}(t) = \begin{pmatrix} 3t^2 & 2t & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}.
\]
Our boundary coditions are

\[ \gamma(0) = P_0 \]
\[ \gamma(L) = P_1 \]
\[ \dot{\gamma}(0) = T_0 \]
\[ \dot{\gamma}(L) = T_1 \]

Substituting 0 and \( L \) in \( \gamma(t) \), we can write these conditions in matrix form:

\[
\begin{pmatrix}
P_0 \\
P_1 \\
T_0 \\
T_1
\end{pmatrix}
= 
\begin{pmatrix}
0 & 0 & 0 & 1 \\
0 & L^3 & L^2 & L & 1 \\
0 & 0 & 1 & 0 \\
3L^2 & 2L & 1 & 0
\end{pmatrix}
\cdot
\begin{pmatrix}
a \\
b \\
c \\
d
\end{pmatrix}.
\]

Solving for \((a, b, c, d)^T\) we get

\[
\begin{pmatrix}
a \\
b \\
c \\
d
\end{pmatrix}
= 
\begin{pmatrix}
0 & 0 & 0 & 1 \\
0 & L^3 & L^2 & L & 1 \\
0 & 0 & 1 & 0 \\
3L^2 & 2L & 1 & 0
\end{pmatrix}^{-1}
\cdot
\begin{pmatrix}
P_0 \\
P_1 \\
T_0 \\
T_1
\end{pmatrix}
= 
\begin{pmatrix}
\frac{2}{L^3} & \frac{-2}{L^2} & \frac{1}{L} & \frac{1}{L} \\
\frac{2}{L^3} & \frac{-2}{L^2} & \frac{1}{L} & \frac{1}{L} \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{pmatrix}
\cdot
\begin{pmatrix}
P_0 \\
P_1 \\
T_0 \\
T_1
\end{pmatrix}.
\]

When we substitute this back in Equation A.1, we get the Hermite polynomials

\[
H_{00}^L = \frac{2t^3}{L^3} - \frac{3t^2}{L^2} + 1, \quad H_{01}^L = \frac{-2t^3}{L^3} + \frac{3t^2}{L^2},
\]
\[
H_{10}^L = \frac{t^3}{L^2} - \frac{2t^2}{L} + t, \quad \text{and} \quad H_{11}^L = \frac{t^3}{L^2} - \frac{t^2}{L}.
\]
Appendix B

Tangents of Nonuniform Natural Cubic Splines

We now solve the problem presented in section 3.1 of finding the tangents for a set of control points of a natural cubic spline. We are given a sequence of \( n \) points, \( P_1, P_2, P_3, \ldots, P_n \), with the convention that \( P_{n+1} \equiv P_1 \). We need to find the corresponding tangent vectors \( T_1, T_2, T_3, \ldots, T_n \) at the above points, such that we have \( C^2 \) continuity.

We require that the second derivative at the end of the segment \([i, i+1]\) be equal to the second derivative at the start of the subsequent segment \([i+1, i+2]\):

\[
\dddot{\gamma}_i(L_i) = \dddot{\gamma}_{i+1}(0) \quad \text{for} \quad 1 \leq i \leq n.
\]

The expansion of the curves becomes

\[
P_i \dddot{H}_{00}^{L_i}(L_i) + P_{i+1} \dddot{H}_{01}^{L_i}(L_i) + T_i \dddot{H}_{10}^{L_i}(L_i) + T_{i+1} \dddot{H}_{11}^{L_i}(L_i)
= P_{i+1} \dddot{H}_{00}^{L_{i+1}}(0) + P_{i+2} \dddot{H}_{01}^{L_{i+1}}(0) + T_{i+1} \dddot{H}_{10}^{L_{i+1}}(0) + T_{i+2} \dddot{H}_{11}^{L_{i+1}}(0),
\]

for \( 1 \leq i \leq n \).
By substituting the Hermite polynomials at \( t = L_i \) and \( t = 0 \), we get:

\[
\frac{6}{L_i^2} P_i - \frac{6}{L_i^2} P_{i+1} + \frac{2}{L_i} T_i + \frac{4}{L_i} T_{i+1} = -\frac{6}{L_{i+1}^2} P_{i+1} + \frac{6}{L_{i+1}^2} P_{i+2} - \frac{4}{L_{i+1}} T_{i+1} - \frac{2}{L_{i+1}} T_{i+2},
\]

for \( 1 \leq i \leq n \).

By reordering the terms we obtain

\[
\frac{2}{L_i} T_i + 4 \left( \frac{1}{L_i} + \frac{1}{L_{i+1}} \right) T_{i+1} + \frac{2}{L_{i+1}} T_{i+2} = -\frac{6}{L_i^2} P_i + 6 \left( \frac{1}{L_i^2} - \frac{1}{L_{i+1}^2} \right) P_{i+1} + \frac{6}{L_{i+1}^2} P_{i+2},
\]

for \( 1 \leq i \leq n \).

In a matrix form this becomes

\[
\begin{pmatrix}
\frac{2}{L_1} & \frac{4}{L_1} & \frac{2}{L_2} & 0 & \ldots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \frac{2}{L_i} & \frac{4}{L_i} & \frac{2}{L_{i+1}} & \ddots & \ddots \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots \\
\frac{2}{L_n} & 0 & 0 & \ldots & \frac{2}{L_{n-1}} & \frac{4}{L_n} + \frac{4}{L_n} \\
\frac{4}{L_n} + \frac{4}{L_1} & \frac{2}{L_1} & 0 & 0 & \ldots & \frac{4}{L_{n-1}} + \frac{4}{L_n}
\end{pmatrix}
\begin{pmatrix}
T_1 \\
T_2 \\
\vdots \\
T_i \\
T_{i+1} \\
T_{i+2} \\
\vdots \\
T_{n-1} \\
T_n
\end{pmatrix}
= \begin{pmatrix}
-\frac{6}{L_1^2} & \frac{6}{L_1^2} & -\frac{6}{L_2^2} & \frac{6}{L_2^2} & 0 & \ldots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & -\frac{6}{L_i^2} & \frac{6}{L_i^2} & -\frac{6}{L_{i+1}^2} & \frac{6}{L_{i+1}^2} & \ddots & \ddots \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\frac{6}{L_n^2} & 0 & 0 & \ldots & -\frac{6}{L_{n-1}^2} & \frac{6}{L_{n-1}^2} & -\frac{6}{L_n^2} \\
\frac{6}{L_n^2} - \frac{6}{L_1^2} & \frac{6}{L_1^2} & 0 & 0 & \ldots & -\frac{6}{L_{n-1}^2} & -\frac{6}{L_n^2}
\end{pmatrix}
\begin{pmatrix}
P_1 \\
P_2 \\
\vdots \\
P_i \\
P_{i+1} \\
P_{i+2} \\
\vdots \\
P_{n-1} \\
P_n
\end{pmatrix}
\]
Note that $L_n$ denotes the distance between $P_n$ and $P_{n+1}$, which currently is identical to $P_1$.

Let $P$ be the vector of points, and $T$ be the vector of tangents. The matrix form equations which can be symbolically written as $M \cdot T = C \cdot P$ where $M$ and $C$ are the corresponding matrices in the above equation. This set of linear equations is easily solved by inverting $M$:

$$T = M^{-1} \cdot C \cdot P.$$
Appendix C

Tangents in a Periodic Dimension

In this section we compute the same tangents as in the previous section, with the slight difference that the problem domain is periodic. Consider the dimension of the camera’s angle, whose cycle is $2\pi$ radians. This periodic domain needs to be considered differently. For example, consider a control point with the value $(2\pi - 0.1)$ rad, and its consecutive control point with the value $0.1$ rad. Solving this in a regular nonperiodic dimension would yield a curve running all the way down from $(2\pi - 0.1)$ to $0.1$ rad. The correct solution in the periodic dimension would be to go from $(2\pi - 0.1)$ rad into $2\pi$, and continue from $0$ to $0.1$ rad.

We solve this by unrolling and spreading the points in the angular periodic dimension to a regular planar dimension by adding and subtracting multiples of $2\pi$ to each point, as needed. For each pair of consecutive points, we choose their smaller periodic distance and update the second point accordingly. In the prior example we would recognize that $(2\pi + 0.1)$ is closer than $0.1$ to $(2\pi - 0.1)$ rad, thus we update the second point to be such, and proceed to the following point.

The problem in this process is that the starting point $P_0$ can now be different from $P_{n+1}$, and still satisfy $P_1 = P_{n+1}$ (mod $2\pi$). We solve this by keeping track of $n + 1$ points, including the new point $P_{n+1}$ which initially identifies with $P_1$. We follow the same solution as before, where instead of cycling the last curve to the beginning, we set the last curve apart. The last two second-derivative equalities now become:

$$\tilde{\gamma}_{n-1}^{L_{n-1}}(L_{n-1}) = \tilde{\gamma}_{n}^{L_{n}}(0) ,$$
$$\tilde{\gamma}_{n}^{L_{n}}(L_{n}) = \tilde{\gamma}_{1}^{L_{1}}(0) .$$
Note that in this section, $L_n$ denotes the distance between $P_n$ and $P_{n+1}$, which is NOT identical to $P_1$, so we must keep track of the $(n + 1)^{st}$ point.

Expanding these last two equalities we get

\[
\frac{6}{L_{n-1}} P_{n-1} - \frac{6}{L_{n-1}} P_n + \frac{2}{L_{n-1}} T_{n-1} + \frac{4}{L_{n-1}} T_n = -\frac{6}{L_n} P_n + \frac{6}{L_n} P_{n+1} - \frac{4}{L_n} T_n - \frac{2}{L_n} T_{n+1}
\]

and

\[
\frac{6}{L_n} P_n - \frac{6}{L_n} P_{n+1} + \frac{2}{L_n} T_n + \frac{4}{L_n} T_{n+1} = -\frac{6}{L_1} P_1 + \frac{6}{L_1} P_2 - \frac{4}{L_1} T_1 - \frac{2}{L_1} T_2 .
\]

Despite keeping track of $P_{n+1}$, there is no need to keep track of $T_{n+1}$, since we actually want $T_{n+1}$ and $T_1$ to be equal. By collecting terms we obtain

\[
\frac{2}{L_{n-1}} T_{n-1} + \left( \frac{4}{L_{n-1}} + \frac{4}{L_n} \right) T_n + \frac{2}{L_n} T_1 = -\frac{6}{L_{n-1}} P_{n-1} + \left( \frac{6}{L_{n-1}^2} - \frac{6}{L_n^2} \right) P_n + \frac{6}{L_n^2} P_{n+1} ,
\]

and

\[
\frac{2}{L_n} T_n + \left( \frac{4}{L_n} + \frac{4}{L_1} \right) T_1 + \frac{2}{L_1} T_2 = -\frac{6}{L_n} P_n + \frac{6}{L_n} P_{n+1} - \frac{6}{L_1} P_1 + \frac{6}{L_1^2} P_2 .
\]

The new term we need to incorporate into our calculation is $P_{n+1}$ which is double underlined. We therefore modify the matrix $C$ in Appendix B to be compatible with this different scheme, by including the $(n + 1)^{st}$ point into it.
Bibliography


The minimization of the energy function leads to the Steepest Descent method for finding the minimum of a function. The method involves calculating the gradient of the function at each point and moving in the direction of the steepest descent. The steps are repeated until convergence is achieved.

The process of minimizing the energy function is crucial for finding the minimum of a function. The method is widely used in various applications, such as computer graphics, image processing, and robotics.

The energy function is a mathematical expression that represents the relationship between the input variables and the desired output. The goal is to find the minimum value of the energy function, which corresponds to the optimal solution.

In computer graphics, the energy function is used to optimize the appearance of objects in a scene. The energy function is defined as a combination of terms, each representing a specific aspect of the object's appearance. The terms are weighted based on the importance of each aspect.

In robotics, the energy function is used to optimize the movement of a robot. The energy function is defined as a combination of terms, each representing a specific aspect of the robot's movement. The terms are weighted based on the importance of each aspect.

The energy function is a powerful tool for solving optimization problems. It is widely used in various fields, such as computer graphics, image processing, and robotics. The method of minimizing the energy function is an essential technique for finding the optimal solution.
ה娭єת חכש על האזורים עם מקדם מסלול אופטימלי לאורך. הביצוע הגיאומטרי
הנה: בדנתך רшение סדרה של פוליגונים פוטרכיים, כו שיתוף עם פוליגונים עקבים ורג
ריך, מזג מסלול תבנית באוזן כל הפוליגונים. עובד בכל אתר בחירה, ומענה פונקציית התיח
כלשה. אם שיתוף המשים-ב
אПетרליצד בוער נקודת הבקרה, שמעון בחר האורו המשמשים. שימש בקופרון ו
מבית ריצוף העון מתאם תחת לקופרונות פעקוף (עקור המגזרת בי כל זה ו
ניקוד בתכונת פוטרכיות), אם שיתוף המשים פוטרטיצי ה الكبرى (פרפריציית אריות-שקת, מש
א tempered, L_i = |P_{i+1} - P_i| t ∈ [0, L_i] \text{ אנא אושרו תופר חולם לא ר מיקום המצלמה, אם של }
ה解放ת ההגמה הל. עד לאושרו תופר חולם לא ר מיקום המצלמה, אם של
שלה, אם שיתוף המשים על הביצוע תחת-았다 במדרג במקל יש
קארנואיאונטו במדרג ביקס, y \text{ קוכס החסכמה של כל הקופרונות המ싼 בר}
שתונים של הפוליגון (ראה איור 2.13). דרישות התווכת והפרפוריציון הקופר בוער לתשל תחת-
וה,UGE נ 정도 מביר היי חולם ובר מיקום ובר בטספ המצלמה. מוכן שופרדו ואלו אנה
נואים, אם שיתוף המשים פוטרכיות למדרג ה UGE שאכנ על כל אדה-
לדגומה, גולדת משלקל של המידות והוותי, \alpha, \text{ הענין השבש רוח ירח לנהיגה והתיחות של }
ה解放ת המצלמה.

המשחק המסולם טוב את האזורית פונקציה מחיי שובה, ושאר המסולם חנות גלואלאית
מקימיות ב-
תיקון. מקוון ישישני גורמי בדלי ייחודי Şוונית, אנו מתחלמי גורמי אלי על יד
הளוקה המסולם המ HomeComponent ייווח ואור הפוליגון. פונקציה המחור הב השיתוף הניה:

\[ C = W_{\text{OutOfZone}} \cdot C_{\text{OutOfZone}} + W_{\text{Curvature}} \cdot C_{\text{Curvature}} + W_{\text{Length}} \cdot C_{\text{Length}} \]

הנגורים הראונים Dü או שולקופרונות יישאר בתר אזור המצלמה המיחס לשלאחרון ממנ
(ראה איור 2.1). מקוון שלקופרונות הנקודות בוער שולקופרונות וועדו בヶ tts בƒנעם, אם
מדחא א Catalonia אנקודות שבחותה עקורות ואמדך, ומענה פוטרכיות על קטע הפוליגון
אחת את מגרידות. הנגורים הם שונים לכל שולקופרונות התווכת בטולע פוטרכיות נמוכה. אха
עקורותמעט או מקורים על יד דוגר של עקורות. אם דרגים שחקני עקורותית נמוכה
(כל עקור) הנסלולהה (מסולם השלם), הנסלולה שלישית היא לכל שארך עקורות היד.
במרחב שונים ושלצללים. זה כמו בעבודה מתוארת הנראות גרף של הרחבת (visibility graph) של ללא מפתת \( d \) ברוחב \( \theta \) של ללא מפתת, ומרחוקים מלבני לפי התוכן ( AVR – Angular Visibility Regions) של ראשון ואופטימונים של הצילום הזוויתי, (אנו נקראו) על במבוסס האלגוריתם של אברהם נרצות והורייתם. שעשו ניסיון של צלעות בנוהל של החלב (ראה איור 2.2). לцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות ל апрור נרצות והורייתם (_visibility graph) המראזוויתיות אזוריות. זה גלי해야 של צלעות עם קשתות של שילוב. שכל בנייו של ח-ב (angular zone) AVR בעל ראות שווה, גלי הלבביתו שני קודקודים ש"דרכים" ואני זריכים לרחוב את החלב \( \theta \) (ראוי איור 2.3). (שים קודקודיםếu צלילים עדיף קודקודים הצילום של איור 11 (ראה איור 2.11). ההעתקים עם הפוליגון החשיבותיים את הצילום \( d \) ברוחב \( \theta \) של ללא מפתת ובית מספרים של החלב (ראה איור 2.1). לцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות ל апрור נרצות והורייתם. שעשו ניסיון של צלעות בנוהל של החלב (ראה איור 2.2). לцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות ל апрור נרצות והורייתם. שעשו ניסיון של צלעות בנוהל של החלב (ראה איור 2.2). לцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות ל апрור נרצות והורייתם. שעשו ניסיון של צלעות בנוהל של החלב (ראה איור 2.2). לцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות ל апрור נרצות והורייתם. שעשו ניסיון של צלעות בנוהל של החלב (ראה איור 2.2). Lцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות L hẹב AVR - Angular Visibility Regions) של ראשון ואופטימונים של הצילום הזוויתי, (אנו נקראו) על בمبוסס האלגוריתם של אברהם נרצות והורייתם. שעשו ניסיון של צלעות בנוהל של החלב (ראה איור 2.2). Lцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות L hẹב AVR - Angular Visibility Regions) של ראשון ואופטימונים של הצילום הזוויתי, (אנו נקראו) על בمبוסס האלגוריתם של אברהם נרצות והורייתם. שעשו ניסיון של צלעות בנוהל של החלב (ראה איור 2.2). Lцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות L/ext \( d \) ברוחב \( \theta \) של ללא מפתת ובית מספרים של החלב (ראה איור 2.2). Lцовרי בטין-ה החלב, זוהי AVR המרחבניותائم של גרי היראות L/ext \( d \) ברוחב \( \theta \) של ללא מפתת ובית מספרים של החלב (ראה איור 2.2). Lcoded עבור \( \theta \) (ראה איור 2.12). Now מגדירים את מבנה המaptic (ראה איור 2.13) בהתחלה \( \theta \) של צלעות שצולמו בצלעם. מכיוון שה מרכזי \( \theta \) של צלעות שצולמו בצלעם. מכיוון שה מרכזי \( \theta \) של צלעות שצולמו בצלעם. מכיוון שה מרכזי \( \theta \) של צלעות שצולמו בצלעם. מכיוון שה מרכזי \( \theta \) של צלעות שצולמו בצלעם. מכיוון שה מרכזי \( \theta \) של צלעות שצולמו בצלעם. מכיוון שה.GetFiles בצינון של המיפוי.
תקציר

שלביםreatment, סטטוס וה HLS אופטימיזציה, עיון לשניים מתקני מלאכות והמקנים מלאכות אחרים. פ컷 ב giữa וצריכים

הנהלה בכמה נושאים שונים, ולא כל פנינים מפרים (ללא התחזיות עצמית ווללה מטריה) במישור. מנטרות ונצואל הספרים שונים מהפקודות והdonnees בתוכי, שלכל לילהאנון שבעזרת היא חשוף ו尕נה את כל האפשרות לנבור בספקת מחקר

הנהלה כולם מהלך התנועה והצפתחות לא מתקרבים מטייל בין. פסבדה חלקי מ.checkSelfPermission ה-

לבולום, ויוצרーム המбереж והיותו של תמציתים את האדונים שואגו עם מחזור הכוכבים של התצלום. הפרדה מהawl מקודר

וידר סיומם שהבולום של תמציתים את האדונים שואגו עם מחזור הכוכבים של התצלום. הפרדה מהawl מקודר

ופנקציית מולד שמלצול.

שלת האינטגרציה של תצוגת הצבעים על קולות מבית. להזחב שדרוג מקודר, הפנקציית מולד שמלצול.


אובייקט אחד או יותר. הקוור אנר במקורות גורמה והתנהגות ה-

עדות בצלום נ슘 התמקדות ה-

ב연구 בתצוגת המברשת. התכלדה של

מנפץ הכל בכמויות קולוניות כלשהן, קוי שברטוגון שואם התצוגה עדות

היא מחמקת באטרקציה של דמיון וירטואליות. מסגרת ביווט, גני

דגון בצלום נ伊拉 התמקדות ה-

אובייקט אחד או יותר. הקוור אנר במקורות גורמה התנהגות ה-
רשימת אורות

<table>
<thead>
<tr>
<th>מספר</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
</tr>
<tr>
<td>2.2</td>
</tr>
<tr>
<td>2.3</td>
</tr>
<tr>
<td>2.4</td>
</tr>
<tr>
<td>2.5</td>
</tr>
<tr>
<td>2.6</td>
</tr>
<tr>
<td>2.7</td>
</tr>
<tr>
<td>2.8</td>
</tr>
<tr>
<td>2.9</td>
</tr>
<tr>
<td>3.1</td>
</tr>
<tr>
<td>3.2</td>
</tr>
<tr>
<td>3.3</td>
</tr>
<tr>
<td>3.4</td>
</tr>
<tr>
<td>4.1</td>
</tr>
<tr>
<td>4.1</td>
</tr>
</tbody>
</table>

- 2.1 נוראות לצולע לא נסות
  - 2.2 אורות למות, הזורע עד, הזיתים
  - 2.3 קשת מעגולה בઈים והופשת התוף
  - 2.4 המיתר והנPickerController, עירו המ(edit) אורות
  - 2.5 קוה-בכה של נראות צולע י购置
  - 2.6 'LR' לכלולים
  - 2.7 כל המגדירים יشؤون
  - 2.8 ו-VVV
  - 2.9 מובטח תונות
  - 2.10 כל המגדירים
  - 2.11 עורות-
  - 2.12 מילאים
  - 2.13 מילאים

- 3.1 עקומת הרומט
  - 3.2 עקומת יונית
  - 3.3 סדר הפרבוון
  - 3.4 דרגת צולע פיטר

- 4.1 תוצאת מתלול
  - 4.1 תוצאת מתלול
3.5 סילום.................................................................
3.6 השפעת זרлиц על אורך העקומה

4.1 מימוש פרטי........................................................................
4.2 תוריון..............................................................................
4.3 ישומם............................................................................
4.3.1 טעות הכללת...............................................................4.3.2 רולוטיה מובשת

5. סיכום וобщеודה ת爿יתית............................................................

A הוראות פוליםרגס ואחרים לא אסידים

B מכמי Natural Cubic Splines לא אסידים

C מｷים מכמד מתחרים
חוכמ ענינים

טכניון בחיפה

מבוא

1.1 קשורות

אזור מסלולות

2.1 קוטפים וויט ראות

2.1.1 VVV

2.1.2 AVR

2.1.3 קריטוריון מרחק

2.2 קריטוריון מרכז

2.3 אור מסלולות

2.4 אור משוטות

3.1 מימוש Natural Cubic Splines

3.2 פירוטי גזרת מסלולות ופרטיה

3.3 פונקציה יחידה מזרי המסלול

3.3.1 שיפורה העקומת בנקודה ראשית

3.3.2 שילוב עקומתיות

3.3.3 אור עקום

3.4 מוזיאון פונקציה יחידה המזרי
حيحר על מחקר עשו הדרכט
פרופ"ח' חיים גוטסמן וד"ר גל ברקט
בפקולטה למה ש🐺しましょう

הכרת תודה

ברצוני להודות ל"ד"ר גל ברקט ופרופ"ח' חיים גוטסמן על
הباحثת העצולה, התמיכה, ו beberapa שלבי המחקר.

אני מודה לסקנינים על התמיכה וה 파일ה 논ויבה בטבתהתיי.
הנפקת הצולמהلسירקקת מצולעת מישורי

דניאל ברונשטיין

과학 ל셈ת העירונין — מכון טכנולוגי לישראל
יוני 2003
הנפקת מחבלית לסריקת מצולעים

דרינלא ברחובשטין